Jiří Demel

Operační výzkum

(pro předměty 128opv, 128opv1 a 128opv2)
7. prosince 2021

Obsah

Kapitola 1

Úvod do operačního výzkumu

1.1 Proč tak divný název

Za 2. světové války běžel ve Velké Británii výzkumný projekt nazvaný "Research in Military Operations". Jeho cílem bylo zvýšit efektivitu vojenských operací, snížit ztráty v námořní dopravě nebo při bombardování nepřátelského území. Při řešení byly používány převážně matematické metody. Řešené problémy byly převáděny na matematické úlohy, které daný problém modelovaly. Na základě výsledků řešení těchto matematických úloh pak byly formulovány závěry pro původní problém.

Po válce se zjistilo, že podobné metody lze použít i v civilní oblasti. Vzniklý obor se začal vyučovat na univerzitách pod názvem "Operations Research" popř. "Operational Analysis". Odtud české "operační výzkum" nebo "operační analýza".

1.2 Modely

1.2.1 K čemu jsou modely. Pro operační výzkum je charakteristické používání tzv. *modelů*. Pomocí modelu zobrazujeme podstatné vlastnosti modelovaného systému. Modelovat lze část reálného světa, ale i nějaký jiný model.

Model, jako lidský výtvor, nikdy nemůže zobrazovat celý svět (neboť je sám jeho součástí) a nemůže zobrazovat všechny jeho vlastnosti. Z (možná nekonečně mnoha) vlastností reality si tedy při tvorbě modelu musíme vybrat ty, které pokládáme pro řešení daného problému za důležité a se kterými dovedeme pracovat.

Neexistuje univerzální návod pro tvorbu dobrého modelu.

Model, který zachycuje příliš mnoho vlastností, je těžké sestavit a je zpravidla těžké s ním pracovat. Model, který zachycuje příliš málo vlastností je zpravidla méně přesný, ale může být snáze zvládnutelný.

Model lze rozličnými metodami zkoumat, s modelem pak lze například experimentovat, lze na něm vyhodnocovat různé veličiny, lze optimalizovat jeho parametry. Podstatné je, že zkoumání modelu lze dělat mimo modelovaný kus světa, neboť práce s modelem je zpravidla levnější a rychlejší a lze dělat i takové experimenty, které by v reálném světě mohly mít katastrofické následky.

Důležitou stránkou modelování je přenést výsledky práce s modelem zpět do reálného světa. Přitom jde jednak o překlad závěrů z řeči modelu do řeči původní problémové oblasti, jednak o prosazení a uskutečnění těchto závěrů.

Existuje mnoho druhů modelů. Liší se podle toho, které vlastnosti modelované části světa zobrazují a podle toho, jak jsou realizovány.

1.2.2 Fyzikální modely jsou zpravidla zmenšené kopie modelovaného objektu. Příkladem jsou papírové modely budov, které používají architekti nebo model (části) letadla ofukovaný v aerodynamickém tunelu.

1.2.3 Matematické modely zobrazují modelovanou část světa pomocí matematických vztahů. Úlohy o modelovaném objektu se tak převádějí na matematické úlohy, které pak lze řešit matematickými metodami.

Matematické modely jsou obecně nesmírně rozmanité. Význačnými typy matematických modelů jsou modely vyhodnocovací a modely optimalizační.

1.2.4 Vyhodnocovací modely slouží k výpočtu neznámých veličin z veličin známých. Tyto modely často mají podobu vzorců nebo rovnic. Často jde o soustavy diferenciálních rovnic, jejich řešení pak jsou funkce, které lze chápat jako předpověď budoucího chování modelovaného systému.

K řešení některých vyhodnocovacích úloh lze použít simulaci (viz 1.2.6).

1.2.5 Optimalizační modely slouží k hledání nejlepšího (tzv. optimálního) řešení. Modelovaný systém je zobrazen pomocí matematické *optimalizační úlohy* (viz. 1.4).

Poznamenejme, že k hledání co nejlepšího řešení lze využít i vyhodnocovací model, a to tak, že experimentujeme se změnami parametrů a pamatujeme si nejlepší dosud nalezené řešení. Jistotu, že nalezené řešení je skutečným optimem takto získáte jen naprosto výjimečně.

1.2.6 Simulační modely jsou zvláštním druhem vyhodnocovacích modelů, a to takové, kde *čas* modelovaného systému je zobrazen jako čas v modelu. Tedy časové uspořádání událostí (co předchází a co následuje) je v modelu stejné jako ve skutečnosti. V simulačním modelu tedy mohou probíhat děje, které jsou obdobou dějů skutečných. Simulovat znamená napodobovat.

Se simulačními modely lze experimentovat, přičemž cílem experimentů bývá nejen hledání nejvhodnějších parametrů modelu (optimalizace), ale např. výcvik obsluhujícího personálu. Např. letecké nebo automobilové trenažery jsou vlastně simulačními modely.

Podle realizace se simulační modely dělí na fyzikální a počítačové.

Simulační modely se dále dělí na deterministické (nepřipouštějící náhodu) a stochastické (připouštějící náhodu a s náhodou počítající).

K simulaci náhodných dějů se používá tzv. metoda Monte-Carlo, tj. simulační experiment se buď mnohokrát náhodně opakuje, nebo se nechá (náhodně) probíhat dostatečně dlouho, a výsledky se vyhodnotí statistickými metodami.

Pozor, nezaměňujte pojmy "simulace" a "metoda Monte-Carlo". Metoda Monte-Carlo jako taková spočívá v provádění náhodných experimentů (které vůbec nemusí být simulační) a v jejich statistickém vyhodnocení. Zdaleka ne každá simulace se dělá metodou Monte-Carlo (např. simulace deterministických dějů) a naopak metodu Monte-Carlo lze použít i pro jiné účely než pro simulaci.

1.3 Klasické části operačního výzkumu

Některé části operačního výzkumu jsou charakterizovány použitými matematickými metodami, jiné části nesou svůj název podle aplikační oblasti.

Lineární programování je předmětem kapitoly 2, str. 9. Jde o velmi široce použitelnou matemaatickou optimalizační metodu.

Teorie front se zabývá nesouladem mezi časově nesourodými požadavky na poskytnutí nějaké služby a omezenými možnostmi tu službu poskytovat, viz kapitola 12, str. 145.

Teorie zásob řeší problém kdy a jak doplňovat zásoby tak, aby náklady s tím spojené byly minimální, viz kapitola 6, str. 81.

Teorie rozvrhování se zabývá přidělováním práce tak, aby všechny práce byly (pokud možno) dokončeny včas nebo aby se minimalizovalo zpoždění (penále).

Atd...

Některé pojmy a triky se uplatňují ve více aplikačních oblastech. Platí to zejména o lineárním programování (kapitola 2, str. 9) a teorii grafů (kapitola 8, str. 89).

1.4 Optimalizační úlohy

Úkolem je najít nejlepší přípustné řešení. Je třeba vysvětlit, co to je *řešení*, které řešení je *přípustné* a co je míněno slovem *nejlepší*.

1.4.1 Množina přípustných řešení. V optimalizačních úlohách hledáme rešení, které je nejlepší mezi všemi tzv. přípustnými řešeními. Množina všech přípustných řešení však obvykle není dána seznamem svých prvků. Obvykle máme dánu množinu, které říkáme *prostor řešení*, a seznam podmínek, kterým říkáme *omezující podmínky*. Množina přípustných řešení pak je tvořena všemi řešeními (tj. všemi prvky prostoru řešení), která splňují omezující podmínky.

Je-li omezujících podmínek několik, musí je přípustné řešení splňovat všechny najednou.

Tutéž množinu přípustných řešení lze často definovat několika způsoby.

Hledáme-li například vlakové spojení v jízdním řádu, můžeme za prostor řešení pokládat všechna spojení z výchozí do cílové stanice. Omezující podmínky pak mohou určovat, jak velká má být časová rezerva při přestupování. Můžeme však také za prostor řešení prohlásit všechna vlaková spojení odkudkoli kamkoli a formou omezujících podmínek požadovat, aby jízda začínala a končila ve správné stanici.

Způsob, jakým je definována množina přípustných řešení, má velice podstatný vliv na způsob řešení úlohy, protože při řešení, chtěj nechtěj, musíme vycházet z tvaru a způsobu, jak je úloha zadána.

1.4.2 Účelová funkce. Které řešení je lepší a které horší určuje v optimalizačních úlohách tzv. *účelová funkce*, což je zobrazení, které každému přípustnému řešení přiřazuje číslo, kterému říkáme *hodnota účelové funkce*.

Optimální řešení je pak takové přípustné řešení, které má mezi všemi přípustnými řešeními nejmenší nebo naopak největší hodnotu účelové funkce. Zde záleží na charakteru úlohy – např. náklady obvykle minimalizujeme, zatímco zisk zpravidla chceme co nejvetší, ale nemusí to tak být vždy.

Může se stát, že úloha má několik optimálních řešení. Všechna optimální řešení téže úlohy ovšem musí mít stejnou hodnotu účelové funkce.

1.4.3 Typ úlohy, instance úlohy a zadání úlohy. Slovem "úloha" bývají často označovány dvě dosti odlišné věci:

Typ úlohy určuje způsob, jak je zadána účelová funkce, zda jde o minimalizaci nebo o maximalizaci, a způsob, jak je zadána množina přípustných rešení. V tomto smyslu tedy mluvíme např. o úloze lineárního programování, čímž máme na mysli obecný typ (druh) úlohy.

Úlohy určitého typu mohou mít spousty různých instancí (konkrétních případů), které se liší např. v konkrétních hodnotách číselných parametrů.

Instance úlohy je konkrétní případ úlohy, který je zadán tak konkrétně, že má smysl ji řešit (počítat) nebo se o to alespoň pokoušet. Zadání úlohy jsou vlastně data (mnohdy číselná), kterými se odlišují jednotlivé instance úlohy.

Například pro úlohu (tj. typ úlohy) najít v grafu nejkratší cestu mezi dvěma vrcholy musí zadání úlohy obsahovat popis konkrétního grafu, ohodnocení hran jejich délkami a počáteční a koncový

vrchol hledané cesty. Teprve máme-li k danému typu úlohy tyto konkrétní údaje, má smysl začít počítat.

Naopak obecný algoritmus pro řešení úloh daného typu (tj. pro řešení všech instancí) lze vymýšlet na základě znalosti typu úlohy, tj. i bez konkrétního zadání (bez dat).

1.4.4 Vícekriteriální optimalizace se zabývá úlohami, v nichž máme několik účelových funkcí. Například hledáme cestu a chceme, aby byla co nejlevnější a současně co nejrychlejší. Najít řešení, které by optimalizovalo všechny účelové funkce současně, obvykle není možné. Ostatně právě proto těch účelových funkcí je několik a ne jedna univerzální.

Metodami vícekriteriální optimalizace se (v tomto skriptu) nebudeme zabývat, to by bylo na samostatnou přednášku, ale nastíníme alespoň některé základní možnosti:

- Nahradit všechny účelové funkce jejich váženým průměrem. Tím se z několika účelových funkcí stane funkce jediná. Samozřejmě je problém korektně stanovit váhy jednotlivých účelových funkcí. (Pozor, účelovou manipulací s vahami lze někdy dělat divy a korupce pak kvete.)
- Hledat přípustné řešení, které je nejblíže nějakému ideálnímu (ale nepřípustnému) řešení. Zde je problém jednak v definici ideálního řešení, jednak ve způsobu měření vzdálenosti od tohoto řešení.
- Hledat takové přípustné řešení, které při srovnání s kterýmkoli jiným přípustným řešením je v alespoň jedné účelové funkci lepší nebo alespoň stejně dobré. To ovšem zdaleka nemusí být jednoznačné.

Kapitola 2

Lineární programování

Lineární programování (dále budeme často užívat zkratku LP) je význačnou a klasickou součástí operačního výzkumu. Z matematického hlediska jde o typ optimalizační úlohy, o metody jejího řešení a o teorii, na které jsou tyto metody založeny.

Slovo „programování“ v názvy je z dnešního pohledu poněkud matoucí, protože dnes se programují zejména počítače, ale v době vzniku operačního výzkumu slovo „programming“ vyjadřovalo stanovení pořadí, času a způsobu plánovaných událostí, obvykle s cílem udělat to „co nejlépe“. V kontextu operačního výzkumu slovo „programování“ chápejte jako „optimalizaci“.

2.1 Formulace úlohy a aplikace.

2.1.1 Obecný tvar úlohy LP. Prostorem řešení úlohy lineárního programování (LP) je n-rozměrný vektorový prostor $\mathbb{R}^n$. Řešeními úlohy tedy jsou n-rozměrné vektory, tj. uspořádané n-tice reálných čísel $(x_1, \ldots, x_n)$.

Účelová funkce úlohy LP je lineární funkcí n proměnných, má tedy tvar

$$L(x) = c_1x_1 + c_2x_2 + \ldots + c_nx_n \ ,$$

kde $c_1, \ldots, c_n$ jsou konstanty, kterým říkáme *cenové koeficienty* nebo také *koeficienty účelové funkce*. Účelová funkce se buď maximalizuje nebo minimalizuje, což zapisujeme jako

$$\begin{aligned} L(x) \to \max \quad &\text{nebo} \quad \max L(x) \ , \\ L(x) \to \min \quad &\text{nebo} \quad \min L(x) \ . \end{aligned}$$

Omezující podmínky jsou v úloze lineárního programování dvou typů:

Podmínky pro jednotlivé proměnné omezují pro každou jednotlivou proměnnou x_j množinu hodnot, kterých tato proměnná smí nabývat. V praxi nejběžnější jsou podmínky nezápornosti, tj. podmínky tvaru $x_j \geqq 0$. Dalšími možnými případy jsou podmínka nekladnosti, tedy $x_j \leqq 0$ a také “vůbec žádná podmínka”, tj. “neomezeno”.

Strukturní podmínky mají tvar lineárních nerovností nebo rovnic, mají tedy tvar

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + \ldots a_{1,n}x_n \quad &\lesseqgtr \quad b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \ldots a_{2,n}x_n \quad &\lesseqgtr \quad b_2 \\ \ldots \quad &\quad \ldots \\ a_{m,1}x_1 + a_{m,2}x_2 + \ldots a_{m,n}x_n \quad &\lesseqgtr \quad b_m \end{aligned}$$

kde na místě označeném $\lesseqgtr$ se může vyskytnout symbol $\leqq$, $\geqq$ nebo $=$.

2.1.2 Příklad – výroba při omezených zdrojích. Jde o nejznámější aplikaci úlohy LP. Úkolem je maximalizovat zisk z výroby n možných výrobků, přičemž výrobní zdroje (materiály, energie, práce) jsou omezeny.

Výrobky, jejichž výrobu uvažujeme, označme (pevně) přirozenými čísly $1, \ldots, n$. Předpokládáme, že jsou známy ceny $c_1, \ldots, c_n$, za které lze tyto výrobky prodat na trhu a že tyto ceny nezávisí na množství prodaných výrobků.

Při výrobě spotřebováváme m zdrojů. Typickým zdrojem je materiál, ale může to být i energie, lidská práce (měřená ve člověkohodinách) apod. Předpokládáme, že spotřeba zdrojů je přímo úměrná velikosti výroby, přičemž koeficienty této úměrnosti jsou známy. Označme a_{ij} spotřebu i-tého zdroje na výrobu jednotkového množství j-tého výrobku. Dále předpokládáme, že jsou známa disponibilní množství b_i jednotlivých zdrojů, která lze pro výrobu použít.

Tuto úlohu lze matematicky modelovat úlohou LP. Označme

x_j	kolik se má vyrobit j-tého výrobku,
c_j	cena, za kterou lze prodat jednotkové množství j-tého výrobku,
b_i	disponibilní množství i-tého výrobního zdroje (kolik ho máme k dispozici),
a_{ij}	spotřeba i-tého zdroje na výrobu jednotkového množství j-tého výrobku.

Úloha LP pak má tvar

$$\begin{aligned} c_1x_1 + c_2x_2 + \ldots + c_nx_n &\to \max \\ a_{1,1}x_1 + a_{1,2}x_2 + \ldots a_{1,n}x_n &\leqq b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \ldots a_{2,n}x_n &\leqq b_2 \\ \ldots &\quad \ldots \\ a_{m,1}x_1 + a_{m,2}x_2 + \ldots a_{m,n}x_n &\leqq b_m \\ x_1, \ldots, x_n &\geqq 0 \end{aligned}$$

Uvědomte si, že jde o zjednodušený model reálné situace. Formulujte, v čem tato zjednodušení spočívají.

2.1.3 Maticový zápis úlohy LP. Označíme-li $c = (c_1, \ldots, c_n)^{\mathrm{T}}$, lze účelovou funkci vyjádřit jako skalární součin $L(x) = c^{\mathrm{T}}x$.

Koeficienty strukturních podmínek lze pokládat za prvky matice

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \ldots & a_{1,n} \\ a_{2,1} & a_{2,2} & \ldots & a_{2,n} \\ \vdots & & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \ldots & a_{m,n} \end{pmatrix}$$

a koeficienty pravých stran strukturních podmínek můžeme pokládat za sloupcový vektor $b = (b_1, \ldots, b_m)^{\mathrm{T}}$. Jsou-li všechny strukturní omezující podmínky stejného typu, tj. všechny typu "$\leqq$", všechny typu "$\geqq$" nebo všechny typu "=", lze strukturní podmínky elegantně vyjádřit maticovým zápisem

$$Ax \leqq b \qquad \text{nebo} \qquad Ax \geqq b \qquad \text{nebo} \qquad Ax = b\,.$$

Podobně, jsou-li podmínky pro jednotlivé proměnné všechny stejného typu, tj. všechny typu $x_j \geqq 0$ nebo všechny typu $x_j \leqq 0$, lze je vyjádřit vektorově jako

$$x \geqq 0 \qquad \text{nebo} \qquad x \leqq 0\,.$$

Jako příklad maticového zápisu úlohy uvedeme zápis úlohy plánování výroby při omezených zdrojích 2.1.2.

$$\begin{aligned} c^{\mathrm{T}}x &\to \max \\ Ax &\leqq b \\ x &\geqq 0\,. \end{aligned}$$

2.1.4 Doporučený postup tvorby modelu LP.

1. Rozhodněte, co budou vyjadřovat proměnné úlohy LP a v jakých to bude jednotkách.
2. Formulujte slovy, čeho chcete optimalizací dosáhnout a vyjádřete to pomocí účelové funkce.
3. Formulujte slovně všechny omezující podmínky a přeložte toto slovní vyjádření do řeči nerovností a rovnic. Určete jednotky, v nichž jsou vyjádřeny.
4. Specifikujte významy konstant obsažených v zadání úlohy LP (koeficienty účelové funkce, koeficienty strukturních podmínek, koeficienty pravých stran). Určete jednotky, v nichž jsou vyjádřeny.
5. Zkontrolujte, zda obě strany každé strukturní omezující podmínky jsou vyjádřeny ve stejných jednotkách. Samo o sobě to správnost modelu nezaručuje, ale často tak lze odhalit některé chyby.
6. Nad výsledkem se zamyslete, zda opravdu modeluje to, co modelovat má.

2.1.5 Úloha o směsi je další klasickou aplikací lineárního programování. Úkolem je z daných surovin připravit co nejlevnější směs s předepsaným složením, přičemž každá surovina je sama směsí. Předpokládáme, že o každé surovině známe její cenu i její přesné složení.

Jako proměnné v modelu LP zvolíme množství jednotlivých surovin použitých ve výsledné směsi. Označme

x_j	množství j-té suroviny vložené do směsi,
c_j	cena za jednotkové množství j-té suroviny,
b_i	množství látky i ve výsledné směsi,
a_{ij}	množství látky i v jednotkovém množství j-té suroviny.

Úloha LP pak má tvar

$$\begin{aligned} c^{\mathrm{T}}x &\to \min \\ Ax &= b \\ x &\geqq 0\,. \end{aligned}$$

2.1.6 Problém výživy je de facto speciálním případem úlohy o směsi. Hledáme, kolik máme sníst kterého jídla, abychom snědli předepsaná množství jednotlivých živin a přitom abychom se stravovali co nejlevněji.

Viz též žertovná aplikace v 2.10.3, str. 42

2.1.7 Dopravní úloha. Máme m dodavatelů, kteří dodávají stejný druh zboží a n spotřebitelů, kteří toto zboží spotřebovávají. Známe ceny za dopravu mezi všemi dodavateli a všemo spotřebiteli. Úkolem je zorganizovat dopravu co nejlevněji. Označme

a_i = kapacita i-tého dodavatele (kolik zboží je schopen dodat),
b_j = požadavek j-tého spotřebitele (kolik zboží chce spotřebovat),
c_{ij} = cena za dopravu jednotkového množství zboží od i-tého dodavatele k j-tému spotřebiteli,
x_{ij} = množství zboží dopravované od i-tého dodavatele k j-tému spotřebiteli.

Pak má úloha LP tvar

$$\begin{aligned}
\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} &\to \min \\
\sum_{j=1}^{n} x_{ij} &= a_i \\
\sum_{i=1}^{m} x_{ij} &= b_j \\
x_{ij} &\geqq 0 \qquad \text{pro všechna } i,j
\end{aligned}$$

Všimněte si, že zde máme mn proměnných, které jsme pro pohodlí indexovali dvěma indexy, lze se tedy na ně dívat jako na matici. Přesto lze proměnné seřadit do posloupnosti (např. po řádcích) a celou úlohu přepsat ve tvaru úlohy lineárního programování.

Dopravní úlohou se budeme zabývat v kapitole 3, str. 47.

2.2 Řešení dvourozměrných úloh LP

Úlohy, které mají jen dvě proměnné, lze snadno řešit graficky, tedy nakreslením obrázku. Na dvourozměrných příkladech ukážeme, co se může stát.

2.2.1 Grafické řešeni úloh LP.

- Nakreslete množinu přípustných řešení (jde o průnik přímek a polorovin).
- Je-li průnik prázdný, stop, úloha nemá přípustné řešení.
- Nakreslete směr optimalizace.
- Zvolte zkusmo nějakou hodnotu účelové funkce H a nakreslete množinu bodů, které mají hodnotu účelové funkce rovnou H. Je to přímka kolmá na směr optimalizace a je určena rovnicí $L(x) = H$.
- Najděte rovnoběžku s touto přímkou a to takovou, která má neprázdný průnik s množinou přípustných řešení a je nejzazší ve směru optimalizace. Tento průnik je množinou optimálních řešení.

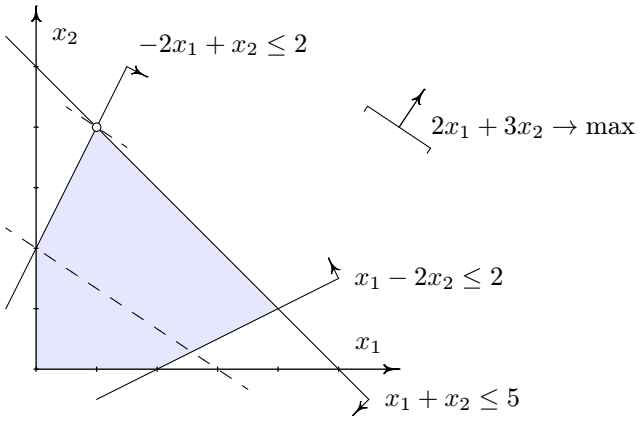
2.2.2 Příklad – jedno optimální řešení.

Řešme úlohu

$$\begin{aligned}
2x_1 + 3x_2 &\to \max \\
x_1 - 2x_2 &\leqq 2 \\
-2x_1 + x_2 &\leqq 2 \\
x_1 + x_2 &\leqq 5 \\
x_1 &\geqq 0 \\
x_2 &\geqq 0
\end{aligned}$$

Viz obrázek 2.1. Pro začátek jsme zvolili hodnotu účelové funkce $H = 6$, příslušná přímka je nakreslena čárkovaně. Posunujeme ji rovnoběžně ve směru optimalizace (tj. zvětšujeme H), dokud tato přímka má neprázdný průnik s množinou přípustných řešení.

Bod $(1, 4)$, který je optimálním řešením, leží v průsečíku přímek určených prvou a třetí omezující podmínkou.



Obrázek 2.1: Jedno optimální řešení.

Omezujícím podmínkám, které takto určují optimální řešení, říkáme *aktivní omezující podmínky*. Jejich "aktivita" spočívá v tom, že jakákoli drobná změna aktivních podmínek má za následek změnu optimálního řešení. Všimněte si, že u ostatních podmínek by jejich drobná změna neměla vliv na souřadnice optimálního řešení. Dokonce i odstraněním neaktivní podmínky by se optimální řešení nezměnilo.

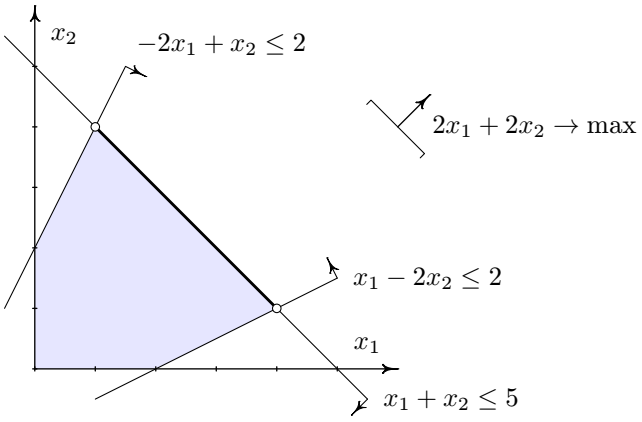
Která omezující podmínka je aktivní, to samozřejmě záleží na směru optimalizace, tedy na koeficientech účelové funkce. Změnou těchto koeficientů lze směr optimalizace libovolně měnit (otáčet). Každý směr může být směrem optimalizace.

Všimněte si, že každý krajní bod množiny přípustných řešení může být optimálním řešením při vhodně zvolené účelové funkci.

2.2.3 Příklad – více optimálních řešení. Řešme úlohu z příkladu 2.2.2, ale s pozměněnou účelovou funkcí (a tedy s jiným směrem optimalizace):

$$2x_1 + 2x_2 \to \max\,,$$

Na obrázku 2.9 je množina optimálních řešení nakreslena tlustě.

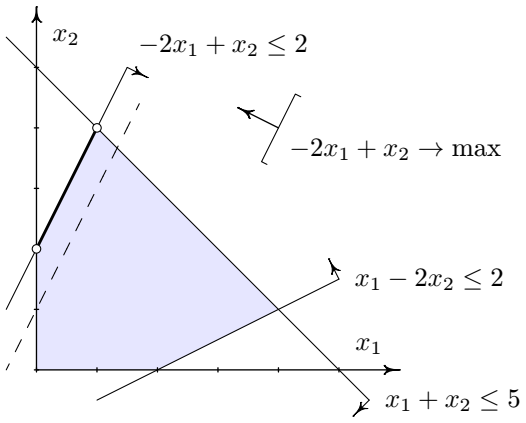


Obrázek 2.2: Více optimálních řešení (příklad 2.2.3).

2.2.4 Příklad – více optimálních řešení II. Podobný příklad, ale s účelovou funkcí

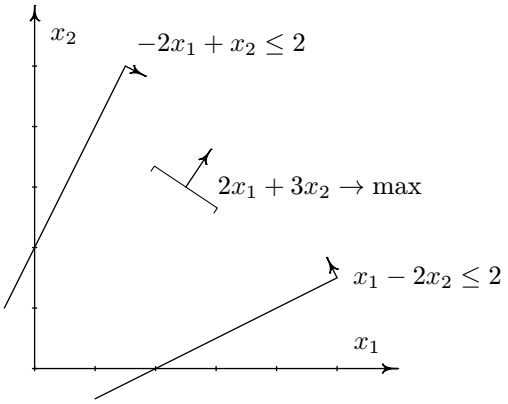
$$-2x_1 + x_2 \to \max \ .$$

je na obrázku 2.3. Množinu optimálních řešení tvoří jiná úsečka.



Obrázek 2.3: Více optimálních řešení (příklad 2.2.4).

2.2.5 Příklad – neomezená účelová funkce. V úloze 2.2.2 vynechejme třetí omezující podmínku. Z obrázku 2.4 je zřejmé, že úloha sice má mnoho přípustných řešení, ale žádné z nich není optimální. Ke každému přípustnému řešení totiž existuje přípustné řešení s lepší hodnotou účelová funkce.



Obrázek 2.4: Neomezená účelová funkce, úloha nemá optimální řešení.

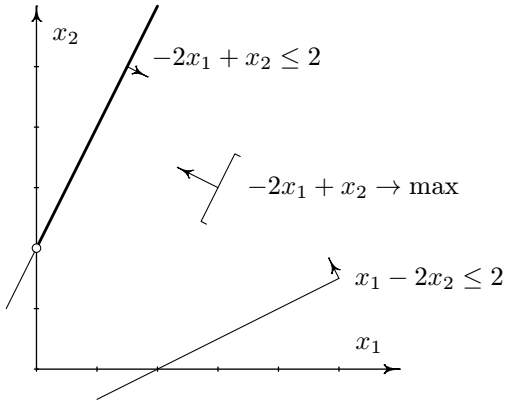
Ve dvourozměrném prostoru to snadno vidíme z obrázku. Ve vícerozměrných prostorech takový případ může nastat také, ale k jeho rozpoznání budeme potřebovat výpočetní aparát.

Všimněte si, že nutným předpokladem, aby účelová funkce mohla být neomezená, je to, že i množina přípustných řešení je neomezená, tj. nevejde se do jakkoli veliké (ale konečné) krychle. Ovšem pozor — naopak to neplatí — i když je množina přípustných řešení neomezená, úloha přesto může mít optimální řešení. Například v naší úloze zaměňte maximalizaci za minimalizaci. Optimálním řešením pak bude bod $(0, 0)$ navzdory tomu, že množina přípustných řešení je neomezená.

2.2.6 Příklad – neomezená množina optimálních řešení. V předchozím příkladě změníme účelovou funkci na

$$-2x_1 + x_2 \to \max \ .$$

Množina přípustných řešení je stále neomezená, ale polopřímka, která je na obrázku 2.10 nakreslena tlustě, je tvořena samými optimálními řešeními.



Obrázek 2.5: Neomezená množina optimálních řešení (příklad 2.2.6).

Poznámka: Kdybychom při stejné množině přípustných řešení měli účelovou funkci $x_1 + x_2 \to \min$, úloha by měla jediné optimální řešení a to v počátku souřadnic.

2.2.7 Příklad – žádné přípustné řešení. Množina přípustných řešení může být prázdná. Jinými slovy, může se stát, že žádný bod prostoru řešení nesplňuje všechny omezující podmínky, nebo, jinak řečeno, každý bod prostoru porušuje alespoň jednu omezující podmínku.

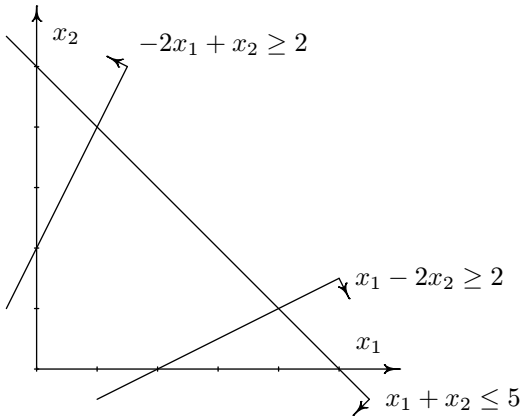
Příklad této situace získáme, když v úloze 2.2.2 obrátíme nerovnosti ve strukturních podmínkách. Viz obrázek 2.6.

$$\begin{aligned} 2x_1 + 3x_2 &\to \max \\ -2x_1 + x2 &\geqq 2 \\ x_1 - 2x_2 &\geqq 2 \\ x_1 + x_2 &\leqq 5 \\ x_1, x_2 &\geqq 0 \end{aligned}$$

Opět platí, že ve dvourozměrném prostoru to snadno vidíme z obrázku. Ve vícerozměrných prostorech to již zřejmé není a opět: k rozpoznání tohoto případu budeme potřebovat výpočetní aparát a hlubší porozumění.

2.3 Geometrický pohled na lineární programování

2.3.1 Bod. Uspořádanou n-tici $(x_1, \ldots, x_n)$ reálných čísel můžeme pokládat za bod n-rozměrného lineárního postoru $\mathbb{R}^n$. Zároveň můžeme tyto n-tice pokládat za n-členné vektory, tj. n-tice lze (po složkách) sčítat a lze je násobit číslem, přičemž výsledkem je zase nějaká n-tice čísel, tedy vektor (a bod).



Obrázek 2.6: Žádné přípustné řešení.

2.3.2 Lineární kombinace. Mějme k bodů $a_1, \ldots, a_k$ v n-rozměrném prostoru a k reálných čísel $t_1, \ldots, t_k$. Pak bod

$$t_1 a_1 + t_2 a_2 + \ldots + t_k a_k$$

nazýváme *lineární kombinací* bodů $a_1, \ldots, a_k$. Čísla $t_1, \ldots, t_k$ nazýváme *koeficienty lineární kombinace.*

2.3.3 Afinní a konvexní kombinace jsou speciání případy lineárních kombinací. Lineární kombinaci $t_1 a_1 + t_2 a_2 + \ldots + t_k a_k$ bodů $a_1, \ldots, a_k$ nazýváme:

afinní kombinací, jestliže $\sum_{i=1}^{k} t_i = 1$,

konvexní kombinací, jestliže $\sum_{i=1}^{k} t_i = 1$ a zároveň všechna t_i jsou nezáporná.

Máme-li dva různé body a, b, pak množina všech jejich konvexních kombinací je úsečka s krajními body a, b a jejich afinní kombinací je přímka těmito body procházející.

Máme-li tři různé body $a, b, c \in \mathbb{R}^3$, které neleží na přímce, pak množina jejich konvexních kombinací je trojúhelník s vrcholy a, b, c a jejich afinní kombinace je rovina, která těmito body prochází.

Na konvexní kombinaci se můžeme také dívat jako na souřadnice těžiště soustavy k hmotných bodů $a_1, \ldots, a_k$ o hmotnostech $t_1, \ldots, t_k$.

2.3.4 Konvexní množina je taková množina bodů, která s každými dvěma svými body obsahuje i celou úsečku, která tyto dva body spojuje.

Příklady konvexních množin jsou úsečka, trojúhelník (včetně svého vnitřku) nebo krychle (opět včetně vnitřku). Také jednoprvková množina tvořená jediným bodem je konvexní množinou.

Příkladem množiny, která není konvexní, je množina bodů, které tvoří hranici čtverce. Jiným příkladem je množina tvořená úsečkou a bodem, který na té úsečce neleží.

2.3.5 Nadrovina, poloprostor. Množina bodů $x = (x_1, \ldots, x_n)$, které vyhovují rovnici

$$a_1 x_1, \ldots, a_n x_n = b$$

(kde a_i a b jsou dané konstanty), se nazývá *nadrovina.*

Množina bodů $x = (x_1, \ldots, x_n)$, které vyhovují nerovnici

$$a_1 x_1, \ldots, a_n x_n \leqq b \qquad \text{popř.} \qquad a_1 x_1, \ldots, a_n x_n \geqq b$$

(kde a_i a b jsou dané konstanty), se nazývá *poloprostor*.

Každý poloprostor i nadrovina jsou konvexními množinami. Samozřejmě i celý prostor $\mathbb{R}^n$ je konvexní množinou.

2.3.6 Průnik konvexních množin je konvexní množina.

Důkaz: Soustavu konvexních množin, z nich děláme průnik, označme S. Vezměme libovolné dva body z průniku. Úsečka mezi těmito body leží celá v každé množině z S (neboť každá tato množina je konvexní) a tudíž ta úsečka leží i v průniku množin z S.

Důsledek: Množina všech přípustných řešení úlohy LP je konvexní množinou (neboť je průnikem poloprostorů a nadrovin, což jsou konvexní množiny). Podobně i množina všech optimálních řešení úlohy LP je konvexní množinou (neboť je průnikem konvexní množiny přípustných řešení a nadroviny, která je určena rovnicí $L(x) = \text{konstanta}$).

2.3.7 Uzavřená množina. Množina bodů M se nazývá *uzavřená*, když pro každou konvergentní posloupnost bodů z M limita této posloupnosti je také bodem množiny M.

Prakticky to znamená, že množina M obsahuje svou hranici. Příkladem (v jednorozměrném prostoru) je uzavřený interval reálnách čísel $\langle a, b\rangle$.

Množina přípustných řešení úlohy LP je vždy uzavřená. Všechny omezující podmínky v LP jsou zásadně neostré nerovnosti (tedy nerovnosti typu „nebo rovno“), nebo jsou to rovnice (a tedy vlastně dvě opačné neostré nerovnosti) právě proto, aby množina přípustných řešení byla uzavřená. Je to důležitá podmínka pro existenci optimálního řešení.

2.3.8 Omezená množina. Množina bodů se nazývá *omezenou*, je-li podmnožinou nějaké (konečně velké) krychle.

Ekvivalentně, množina M je omezená, existuje-li konstanta $K > 0$ taková, že všechny souřadnice všech bodů množiny M jsou menší než K.

2.3.9 O exitenci optima. Z matematické analýzy je známo, že každá spojitá funkce na omezené a uzavřené množině má minimum a maximum. Tedy, je-li množina přípustných řešení úlohy LP omezená, pak je zaručeno, že tato úloha má optimální řešení.

2.3.10 Konvexní obal množiny bodů M je množina všech bodů, které lze získat jako konvexní kombinace bodů množiny M.

Tedy např. konvexním obalem tří různých bodů a, b, c, které neleží na přímce, je trojúhelník s vrcholy a, b, c. Pokud tyto tři body leží na přímce, je jejich konvexním obalem úsečka.

2.3.11 Krajní bod konvexní množiny M je takový bod $x \in M$, který není vnitřním bodem žádné úsečky, která leží celá v M.

Krajními body čtverce (chápaného včetně jeho obvodu) jsou jeho vrcholy, ale jiné body na jeho obvodu krajními body nejsou. Krajními body kruhu je však celý jeho obvod.

2.3.12 Konvexní obal množiny krajních bodů. Každá omezená a uzavřená konvexní množina je konvexním obalem svých krajních bodů.

Je-li tedy množina přípustných řešení úlohy LP omezená, lze ji elegantně (a velmi srozumitelně) popsat seznamem všech jejích krajních bodů. To je z praktického hlediska daleko užitečnější než popis pomocí rovnic a nerovností.

2.3.13 Tvrzení. Má-li úloha LP optimální řešení, pak má (také) optimální řešení v některém krajním bodě množiny P všech přípustných řešení. Jinak řečeno, úloha lineárního programování může mít i jiná optimální řešení, ale některé z optimálních řešení je vždy v krajním bodě.

Platí dokonce více: Je-li množina přípustných řešení úlohy LP omezená a je-li S množina všech krajních bodů, které jsou optimálními řešeními této úlohy, pak množina všech optimálních řešení této úlohy je konvexním obalem množiny S.

To má velmi praktický důsledek: Při hledání optimálních řešení LP se stačí zajímat jen o krajní body množiny přípustných řešení a těch, jak uvidíme dále, je konečně mnoho.

2.4 Kanonický tvar úlohy LP

2.4.1 Kanonický tvar úlohy LP má význam pro výpočet. Běžná metoda výpočtu, tzv. simplexová metoda, totiž funguje jen pro úlohy v kanonickém tvaru. Obecná úloha se řeší tak, že se nejprve převede na (v jistém smyslu ekvivalentní) úlohu v kanonickém tvaru a z jejího řešení se pak odvodí řešení původní obecné úlohy. Podrobnosti vysvětlíme dále v 2.4.7.

Úloha lineárního programování je *v kanonickém tvaru*, jestliže

- účelová funkce se maximalizuje,
- na všechny proměnné se klade podmínka nezápornosti a
- všechny strukturní podmínky jsou typu rovnice, tj. „=“ a
- včechny koeficienty b_i pravých stran jsou nezáporné ($b_i \geqq 0$).

Tytéž podmínky zapsány maticově jsou:

$$\begin{aligned} c^{\mathrm{T}} x &\to \max \\ Ax &= b \\ x &\geqq 0 \end{aligned}$$

Všimněte si, že podmínka $b_i \geqq 0$ není součástí zadání úlohy lineárního programování. Je to podmínka, které musí vyhovovat zadání (koeficienty pravých stran), aby se instance úlohy mohla nazývat úlohou v kanonickém tvaru.

2.4.2 K čemu je kanonický tvar. Převod na kanonický tvar má dvojí smysl:

- Se soustavami rovnic se dobře počítá, zejména lze provádět ekvivalentní řádkové úpravy, aniž by to mělo vliv na množinu přípustných řešení. S nerovnostmi se to tak jednoduše dělat nedá.
- Úlohu v kanonickém tvaru lze mnohem jednodušeji sdělit. K zadání úlohy v kanonickém tvaru stačí zadat cenový vektor c, matici A a vektor pravých stran b a nic víc. Vše ostatní plyne z obecných vlastností úloh v kanonickém tvaru.

Jakoukoli obecnou úlohu lineárního programování lze převést na úlohu v kanonickém tvaru.

2.4.3 Převod minimalizace na maximalizaci je snadný: stačí obrátit znaménka u všech koeficientů účelové funkce.

2.4.4 Převod podmínek pro jednotlivé proměnné se dělá pomocí substituce.

Jestliže v původní úloze je podmínka $x_j \leqq 0$, pak provedeme substituci $x_j = -x'_j$. Podmínka $x_j \leqq 0$ tím přejde na podmínku nezápornosti $x'_j \geqq 0$. Po provedení výpočtu ovšem je nutno výslednou hodnotu původní proměnné x_j získat z hodnoty x'_j zpětnou substitucí.

Jestliže v původní úloze není hodnota proměnné x_j vůbec nijak omezena, je převod paradoxně trochu složitější. V tomto případě použijeme substituci $x_j = x'_j - x''_j$, přičemž u nových proměnných x'_j a x''_j budeme požadovat jejich nezápornost. Jinými slovy, každý výskyt proměnné x_j nahradíme rozdílem dvou nezáporných proměnných. Po provedení výpočtu pak hodnotu původní proměnné x_j získáme zpětnou substitucí, tj. jako rozdíl oněch dvou proměnných.

2.4.5 Převod nerovností $\leqq$ na rovnice se dělá přidáním nových, tzv. *doplňkových proměnných*. Nerovnost typu $\leqq$ převedeme na rovnici tak, že (nezápornou) hodnotu, o kterou je levá strana menší, než pravá, k levé straně přidáme jako hodnotu doplňkové proměnné, přičemž na tuto novou proměnnou klademe požadavek nezápornosti. Doplňková proměnná má hodnotu rozdílu mezi původní levou a pravou stranou, je tedy rovna rezervě, s jakou je původní nerovnost splněna.

Za každou nerovnost, kterou takto převádíme na rovnici, přidáváme samostatnou doplňkovou proměnnou, čímž zvětšujeme dimenzi prostoru řešení.

Celkový počet nerovností v úloze se zavedením doplňkových proměnných nezmění. Podstatné ovšem je, že nerovnosti ve strukturních omezujících podmínkách, se kterými se špatně pracuje, jsou nahrazeny omezeními nezápornosti kladenými na doplňkové proměnné.

2.4.6 Převod nerovností $\geqq$ na rovnice se dělá podobně, tedy také zavedením doplňkových proměnných, ale v tomto případě doplňkové proměnné od levých stran nerovnosti odečítáme. Také na tyto doplňkové proměnné klademe požadavek nezápornosti.

2.4.7 Vztah úlohy v obecném a kanonickém tvaru. Má-li jedna z úloh přípustné řešení, má je i druhá a lze ho i snadno získat.

Pokud původní úloha měla přípustné řešení $x_1, \ldots, x_n$, pak úloha v kanonickém tvaru, která je výsledkem převodu, má rovněž přípustné řešení a to takové, že původní proměnné své hodnoty zachovají a doplňkové proměnné jsou rovny rozdílům mezi levou a pravou stranou, tedy vlastně velikosti rezervy ve splnění nerovnosti.

Také naopak, pokud úloha v kanonickém tvaru má přípustné řešení, pak vynecháním doplňkových proměnných dostaneme přípustné řešení původní úlohy.

2.4.8 Příklad. Převedeme na kanonický tvar úlohu

$$\begin{aligned} 2x_1 + 3x_2 &\to \min \\ x_1 - 2x_2 &\leqq 2 \\ -2x_1 + x_2 &\leqq 2 \\ x_1 + x_2 &\geqq 5 \\ x_1, x_2 &\geqq 0 \end{aligned}$$

Kvůli třem strukturním podmínkám musíme zavést tři doplňkové proměnné x_3, x_4, x_5.

$$\begin{array}{rrrrrcl} -2x_1 & -3x_2 & & & & \to & \max \\ x_1 & -2x_2 & +x_3 & & & = & 2 \\ -2x_1 & +x_2 & & +x_4 & & = & 2 \\ x_1 & +x & & & -x_5 & = & 5 \\ & & & x_1, \ldots, x_n & & \geqq & 0 \end{array}$$

Vezměme libovolné přípustné řešení původní úlohy, např. řešení $(3, 4)$. Toto řešení splňuje strukturní podmínky s rezervami po řadě 7, 4 a 2. Bod $(3, 4, 7, 4, 2)$ je řešením odpovídající úlohy v kanonickém tvaru, přičemž rezervy 7, 4, 2 jsou hodnotami doplňkových proměnných $x_3, \ldots, x_5$.

Naopak, vezmeme-li libovolné řešení kanonického tvaru úlohy, např. $(4, 1, 0, 9, 0)$, pak prvé dvě souřadnice $(4, 1)$ jsou přípustným řešením původní úlohy a hodnoty doplňkových proměnných sdělují, s jakými rezervami jsou splněny jednotlivé strukturní omezující podmínky. Nulová rezerva znamená, že omezující podmínka je splněna „na doraz“ a příslušný bod leží na přímce (obecně nadrovině), určené tou omezující podmínkou.

2.5 Bázická přípustná řešení

V tomto oddíle se budeme zabývat přípustnými řešeními úlohy v kanonickém tvaru, tj. nezápornými řešeními soustavy lineárních rovnic $Ax = b$ a ukážeme, jak to souvisí s množinou krajních bodů množiny přípustných řešení.

Budeme předpokládat, že řádky matice A jsou lineárně nezávislé, tj. že hodnost matice A je rovna počtu řádků m. Později uvidíme, že tento předpoklad bývá v úlohách LP triviálně splněn.

2.5.1 Sloupcový prostor je vektorový prostor generovaný všemi sloupci matice A.

Řešíme-li soustavu rovnic $Ax = b$, pak vlastně hledáme lineární kombinaci sloupců matice A, jejímž výsledkem je vektor pravých stran. Koeficienty této lineární kombinace (pokud existuje) jsou rovny souřadnicím řešení $x_1, \ldots, x_n$ soustavy rovnic.

Úloha $Ax = b$ má přípustné řešení právě tehdy, když vektor b je prvkem sloupcového prostoru, tj. když jej lze vygenerovat ze sloupců matice A jako lineární kombinaci.

2.5.2 Bázické řešení soustavy rovnic $Ax = b$ je jakékoli řešení x, které dostaneme takto:

- zvolíme množinu sloupců matice A, označme je a_{t_i} pro $i = 1, \ldots, m$, a to tak, aby tyto sloupce tvořily bázi sloupcového prostoru generovaného všemi sloupci matice A. Těmto sloupcům budeme říkat *bázické sloupce*.
- Ze sloupců a_{t_i} sestavíme čtvercovou matici A' a řešením soustavy rovnic $A'x = b$ dostaneme hodnoty proměnných x_{t_i} pro $i = 1, \ldots, m$. Tyto proměnné nazýváme *bázickými proměnnými*.
- Ostatní proměnné x_j, kde $j \notin \{t_1, \ldots, t_m\}$, položíme rovny nule. Tyto proměnné nazýváme *nebázickými proměnnými*.

Jsou-li všechny souřadnice bázického řešení nezáporné, nazýváme toto řešení *bázickým přípustným řešením*. V dalším textu budeme občas používat zkratku BPŘ.

Dodejme, že matice A' je regulární. Proto je bázické řešení jednoznačně určeno výběrem bázických sloupců. To mimochodem znamená, že bázckých řešení je vždy konečně mnoho, nemůže jich být více než $\binom{n}{m}$.

2.5.3 Jednotkový vektor je vektor, jehož všechny souřadnice jsou nulové, pouze jedna souřadnice má hodnotu 1. Jednotkový vektor, který má jedničku na i-tém místě, označíme e_i.

2.5.4 Kanonická báze je báze sloupcového prostoru, která je tvořena jednotkovými vektory, které jsou sloupci matice A.

Ze sloupců, které tvoří kanonickou bázi, tedy lze složit jednotkovou matici řádu m. Dodejme, že sloupce, které tvoří kanonickou bázi, se nemusí vyskytovat v matici A ve stejném pořadí, ani tsně za sebou jako v jednotkové matici. Bázické sloupce mohou být v matici A na přeskáčku rozptýleny. Pro snazší orientaci zavedeme toto značení:

2.5.5 Indexy bázických souřadnic. Máme-li v matici A kanonickou bázi, pak pro každý řádek i označme t_i číslo bázickho sloupce matice A, který je jednotkovým vektorem s jedničkou na i-té pozici. Jinými slovy, $t_i = j$ pro takové j, že sloupec a_j je v bázi a je jednotkovým vektorem e_j.

Kanonická báze je tedy tvořena množinou sloupců $\{a_{t_i} \mid i = 1, \ldots, m\}$.

2.5.6 Řešení reprezentované tabulkou. Soustavu rovnic $Ax = b$ lze bez ztráty informace reprezentovat rozšířenou maticí soustavy. Pokud matice soustavy A obsahuje kanonickou bázi tvořenou sloupci a_{t_i} pro $i = 1, \ldots, m$, pak bázické řešení získané z této báze nazýváme *řešením reprezentovaným tabulkou* (přičemž tabulkou se zde míní rozšířená matice soustavy).

Díky speciálnímu tvaru kanonické báze platí

$$\begin{array}{lcll} x_{t_i} & = & b_i & \text{pro bázické proměnné } t_i \ , \\ x_j & = & 0 & \text{pro ostatní (tj. nebázické) proměnné.} \end{array}$$

Při výpočtu LP budeme pracovat pouze s řešeními, která budou reprezentována tabulkami, přičemž další tabulky budeme získávat ekvivalentními úpravami rozšířené matice soustavy.

2.5.7 Krajní body a bázická přípustná řešení. Lze dokázat, že bod je bázickým přípustným řešením právě tehdy, když je krajním bodem množiny přípustných řešení.

Důsledek: soustředíme se na bázická přípustná řešení.

2.5.8 Jordanova eliminace. Popíšeme metodu, jak z jedné tabulky, která obsahuje kanonickou bázi, získat další tabulku, která je s ní ekvivalentní a rovněž obsahuje kanonickou bázi.

- Zvolte nenulový klíčový prvek $a_{ij} \neq 0$. Jak jej volit uvedeme dále.
- Klíčový řádek A_i vydělte klíčovým prvkem a_{ij}. Tím na místě klíčového prvku dostanete jedničku.
- Ke každému neklíčovému řádku přičtěte takový násobek klíčového řádku, aby se hodnota v klíčovém sloupci vynulovala.

Toto budeme nazývat *jedním krokem Jordanovy eliminace.*

Provedením Jordanovy eliminace se změní kanonická báze. Z kanonické báze bude vyloučen sloupec s indexem t_j (tento sloupec se eliminací poškodí) a bude nahrazen jednotkovým vektorem, který vznikne v klíčovém sloupci j.

Pozor – má-li výsledná tabulka obsahovat kanonickou bázi, je třeba Jordanovu eliminaci dělat přesně výše uvedeným způsobem. Pouze klíčový řádek smí být násoben (číslem různým od jedničky). K ostatním řádkům smíme pouze přičítat násobky řádku klíčového. Ověřte na příkladě, že jakýkoli jiný postup kanonickou bázi poškodí.

2.5.9 Jak volit klíčový prvek v daném sloupci j. Pokud řešení reprezentované tabulkou bylo přípustné (tj. všechny pravé strany byly nezáporné) a volíme-li klíčový prvek tak, že

1. klíčový prvek je kladný, tj. $a_{ij} > 0$ a
2. podíl b_i/a_{ij} je minimální (nejbližší k nule),

pak po provedení Jordanovy eliminace bude řešení reprezentované tabulkou opět přípustné (tj. všechny pravé strany budou opět nezáporné).

Pokud klíčový prvek ve sloupci zvolíte jinak (tj. porušíte-li některé z výše uvedených pravidel), dostanete v příští tabulce zápornou hodnotu na pravé straně a řešení reprezentované tabulkou tedy bude nepřípustné. Vyzkoušejte na příkladě.

2.5.10 Sousední řešení. Dvě řešení, která jsou reprezentována tabulkami, které lze získat jednu z druhé provedením jednoho kroku Jordanovy eliminace, nazýváme *sousedními* řešeními.

Pokud množinu přípustných řešení pokládáme za mnohostěn v n-rozměrném prostoru, pak sousední řešení jsou spojena hranou mnohostěnu.

Až na degenerované případy má každé bázické přípustné řešení nejvýše $n-m$ sousedů. Sousedů je vždy konečně mnoho.

2.5.11 Jak získat všechna bázická přípustná řešení. Předpokládejme, že máme tabulku, která obsahuje kanonickou bázi a tedy reprezentuje nějaké bázické přípustné řešení. Tuto tabulku spolu s řešením, které reprezentuje, a s množinou indexů bázických sloupců, zařaďme jako první položku seznamu.

Dále pro každou tabulku v seznamu proveďme kontrolu, zda všechny sousední tabulky jsou také obsaženy v seznamu. Pokud ne, pak tuto tabulku vypočteme (Jordanovou eliminací) a zařadíme do seznamu. Takto postupujte, dokud ke každé tabulce nemáme v seznamu všechny sousedy.

Vzhledem k větě 2.3.13, str. 17 by bylo možné tímto způsobem najít optimální řešení, neboť všech bázických přípustných řešení je konečně mnoho (určitě ne více než $\binom{n}{m}$). Pro velká n a m by však tento postup byl velmi pracný, až nerealizovatelný. Naštěstí existuje způsob, jak předem, ještě před provedením kroku Jordanovy eliminace, poznat, zda se tímto krokem účelová funkce zlepší nebo zhorší.

2.5.12 Pohyb k sousednímu řešení po hraně. Mějme rozšířenou matici soustavy s úplnou kanonickou bází. V řešení reprezentovaném tabulkou chtějme změnit (tj. zvětšit) nebázickou souřadnici x_l o hodnotu Δ.

Při změně je třeba zachovat platnost všech rovnic. Toho lze nejjednodušeji dosáhnout změnami bázických proměnných, protože každá z nich má vliv pouze na jednu rovnici.

Výsledkem změny je řešení, které leží na polopřímce, jejímž krajním bodem je BPŘ reprezentované tabulkou a směrový vektor bude odvozen z l-tého sloupce matice A takto:

- x_l se změní z nuly na Δ,
- každá bázická souřadnice x_{t_i} se změní na hodnotu $x_{t_i} - a_{il} \cdot \Delta$,
- ostatní souřadnice zůstanou nulové.

Kvůli přípustnosti souřadnice x_l musí být $\Delta \geqq 0$. Proto mluvíme o polopřímce.

Při rostoucím Δ mohou bázické souřadnice x_{t_i} růst i klesat, záleží to na hodnotách prvků a_{il} ve sloupci l.

Pokud při rostoucím Δ žádná bázická souřadnice neklesá (tj. pokud ve sloupci a_l není žádný kladný prvek), pak pro všechna $\Delta \geqq 0$ dostáváme přípustné řešení. Celá polopřímka je tedy tvořena přípustnými řešeními. Tímto poněkud zvláštním případem, zejména otázkou, co se na té polopřímce děje s účelovou funkcí, se budeme zabývat dále v 2.6.10, str. 26 a v příkladech 2.6.11 a 2.6.14.

Pokud při rostoucím Δ některá bázická souřadnice klesá, pak nejmenší nezáporná hodnota Δ, při které se některá bázická souřadnice x_{t_k} vynuluje, je rovna b_k/a_{kl} pro některé k. Při této hodnotě Δ máme všechny souřadnice nezáporné, a počet nenulových proměnných opět nepřesahuje m.

Výsledné souřadnice jsou rovny bázickému přípustnému řešení, které bychom alternativně mohli dostat Jordanovou eliminací, kdybychom jako klíčový prvek zvolili a_{kl}. (Opět, ověřte na příkladě.) Všmněte si, že hodnota $\Delta = b_k/a_{kl}$ je rovna minimálnímu podílu b_i/a_{ij} z 2.5.9.

K sousednímu bázickému přiustnému řešení (tj. k sousednímu vrcholu mnohostěnu) se tedy můžeme dostat dvěma způsoby: pohybem po hraně, nebo Jordanovou eliminací.

2.5.13 Změna účelové funkce při pohybu k sousednímu řešení po hraně. Sledujeme-li změnu účelové funkce při pohybu po hraně k sousednímu řešení, pak změna účelové fce je přmo úměrná velikosti parametru Δ a závisí na cenových koeficientech c_l a $c_{t_1}, \ldots, c_{t_m}$ a to tak, že přírůstek účelové funkce je roven

$$-\left(\left(\sum_{i=1}^{m} c_{t_i} \cdot a_{il}\right) - c_l\right) \cdot \Delta \,.$$

Můžete si to ověřit tím, že sečtete změny jednotlivých souřadnic vynásobené jejich cenovými koeficienty. Zkuste si to alespoň na příkladě.

Výraz ve vnější závorce je velmi důležitý a vžilo se pro něj označení $(z_j - c_j)$. Budeme jej nazývat *relativní cenou*. Přírůstek účelové funkce tedy je roven

$$-(z_j - c_j) \cdot \Delta \ .$$

Poněvadž víme, při jakém Δ se dostaneme do sousedního bázického přípustného řešení, můžeme tak snadno předpovědět přírůstek účelové funkce i pro Jordanovu eliminaci a to ještě před tím, než ji provedeme.

Zejména je důležité, že podle relativní ceny lze velmi snadno poznat, zda hrana (úsečka nebo polopřímka) vede směrem k lepším (tj. větším) hodnotám účelové funkce. Toto zjištění má dalekosáhlé důsledky, kerými se budeme zabývat dále. Nejdůležitější z nich je tzv. sipmplexová metoda.

2.6 Základní simplexová metoda

V tomto oddíle vyložíme pouze základní simplexovou metodu. Omezíme se na případ, kdy rozšířená matice soustavy lineárních rovnic $Ax = b$ obsahuje úplnou kanonickou bázi. Případem, kdy tento předpoklad není splněn, se budeme zabývat v dalším oddíle 2.7

Simplexová metoda, velmi zhruba řečeno, spočívá v opakovaném provádění Jordanovy eliminace, geometricky jde o pohyb po hranách mnohostěnu přípustných řešení. Přitom hranu, po které půjdeme dál tedy klíčový prvek pro Jordanovu eliminaci, vybíráme podle relativních cen tak, aby se účelová finkce zvětšovala.

2.6.1 Předověď změny účelové funkce před Jordanovou eliminací. Provedením Jordanovy eliminace získáme tabulku, která reprezentuje sousední bázické přípustné řešení (sousední bod mnohostěnu) a do téhož bodu se dostaneme při pohybu po hraně k sousednímu řešení, jak je popsáno v 2.5.12. Navíc, podle 2.5.13 již víme, že přírůstek účelové funkce je roven

$$-(z_j - c_j) \cdot \frac{b_i}{a_{ij}} \ ,$$

kde $(z_j - c_j)$ je tzv. relativní cena.

2.6.2 Relativní cena pro sloupec j je definována vzorcem

$$(z_j - c_j) = \left(\sum_{i=1}^{m} c_{t_i} \cdot a_{ij} \right) - c_j \ .$$

Je-li sloupec j v bázi, je vždy $(z_j - c_j) = 0$.

2.6.3 Přírůstek účelové funkce v příštím kroku Jordanovy eliminace při volbě klíčového prvku a_{ij} bude roven

$$-(z_j - c_j) \cdot \frac{b_i}{a_{ij}}$$

Důkaz:

Poznámka: Z tohoto tvrzení lze velmi jednoduše odvodit řadu důležitých tvrzení. V tomto smyslu jde o jakési "zlaté pravidlo lineárního programování".

2.6.4 Simplexová metoda. Pro výchozí tabulku vypočtěte relativní ceny. Dokud existuje záporná relativní cena opakujte následující:

- Vyber klíčový sloupec l, aby $z_l - c_l < 0$.
- Vyber klíčový řádek k podle pravidla 2.5.9 pro výběr klíčového prvku ve sloupci.
- Jordanovou eliminací spočti novou tabulku.
- Vypočti nové relativní ceny.

2.6.5 Simplexová tabulka. Ruční výpočet se dělá v tzv. *simplexové tabulce* tohoto tvaru:

		c^{T}	max
c_B	t_i	A	b
		$(z-c)^{\mathrm{T}}$	L

Základ simplexové tabulky tvoří rozšířená matice soustavy rovnic $(\,A\,|b)$. Nad maticí A je řádek cenových koeficientů c^{T}. Tento řádek se (jako jediný) během výpočtu nemění, proto jej do dalších tabulek zpravidla neopisujeme.

Vlevo od matice A jsou dva sloupce. Blíže k matici A je sloupec indexů t_i, které ukazují na sloupce tvořící kanonickou bázi. Vlevo od něj je vektor bázických cenových koeficientů c_B. Jeho i-tá složka je cenový koeficient c_{t_i}.

Pod maticí A je řádek relativních cen $(z-c)$. Vpravo od něj, pod sloupcem pravých stran, je hodnota účelové funkce řešení, které je reprezentováno touto tabulkou. Je to skalární součin $c_B^{\mathrm{T}} b$.

2.6.6 Příklad výpočtu simplexovou metodou. Řešme příklad 2.2.2, str. 12. Převod na kanonický tvar viz 2.4.8

		2	3	0	0	0	max
0	3.	1	−2	1	0	0	2
0	4.	−2	1	0	1	0	2
0	5.	1	1	0	0	1	5
		−2	−3	0	0	0	0
0	3.	−3	0	1	2	0	6
3	2.	−2	1	0	1	0	2
0	5.	3	0	0	−1	1	3
		−8	0	0	3	0	6
0	3.	0	0	1	1	1	9
3	2.	0	1	0	$\frac{1}{3}$	$\frac{2}{3}$	4
2	1.	1	0	0	$-\frac{1}{3}$	$\frac{1}{3}$	1
		0	0	0	$\frac{1}{3}$	$2\frac{2}{3}$	14

Prvý řádek, který obsahuje cenový vektor, se během výpočtu nemění, proto se v tabulce neopisuje.

Poslední (třetí) simplexová tabulka reprezentuje řešení $(1,4,9,0,0)^{\mathrm{T}}$. Díky nezáporným relativním cenám v této tabulce je toto řešení optimální

2.6.7 Jak volit klíčový sloupec. Především je třeba zdůraznit, že jako klíčový sloupec můžete volit libovolný nebázický sloupec. Jen musíte být připraveni na to, že nevhodnou volbou můžete účelovou funkci zhoršit. Obvykle však chceme účelovou funkci zlepšovat, k tomu je nutné volit klíčový sloupec se zápornou relativní cenou. Často však máme několik možností.

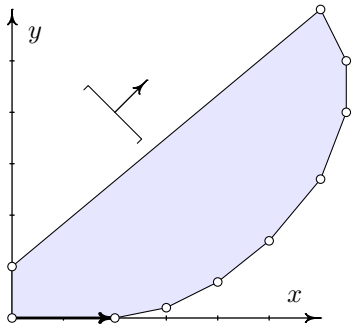
Základní pravidla pro volbu klíčového sloupce jsou:

1. volit sloupec s nejvíce zápornou relativní cenou,
2. volit sloupec tak, aby byl přírůstek účelové funkce největší,
3. volit zleva prvý sloupec se zápornou relativní cenou.

Výběr podle pravidla 3 je nejméně pracný, ale přírůstky účelové funkce jsou v průměru horší než u ostatních pravidel. Toto pravidlo má význam zejména jako ochrana před zacyklením výpočtu v případě tzv. degenerace, viz dále v 2.6.8.

Pravidlo 2 naopak poskytuje v průměru nejlepší (tj. největší) přírůstky účelové funkce a tedy výpočet s nejmenším průměrným počtem iterací. Bohužel námaha (objem výpočtů) nutná pro výběr klíčového prvku není zanedbatelná a zpravidla převáží. Navíc, toto pravidlo nic nezaručuje, úspora v počtu iterací je nevýrazná a existují úlohy, kdy chamtivá snaha o co největší okamžitý zisk vede k situaci (tabulce), ze které cesta k optimu vede přes velký počet simplexových tabulek, viz příklad na obrázku 2.7

Jako dobrý kompromis se tedy jeví pravidlo 1, tedy výběr sloupce s nejzápornější relativní cenou. Výběr minima z relativních cen je poměrně snadný a přírůstky účelové funkce jsou v průměru lepší než u pravidla 3.



Obrázek 2.7: Příklad, že chamtivá snaha o co největší okamžitý přírůstek účelové funkce může vést k velmi dlouhému výpočtu (viz 2.6.7).

2.6.8 Degenerované řešení je takové bázické řešení, v němž je některá bázická souřadnice rovna nule. V simplexové tabulce je některá pravá strana nulová.

Proč to vadí: I když zvolíme klíčový sloupec se zápornou relativní cenou, pokud vyjde klíčový prvek do řádku i s nulovou pravou stranou b_i, bude přírůstek účelové funkce nulový. Pravděpodobnost, že se to stane, je poměrně vysoká, neboť stačí, aby bylo $a_{ij} > 0$ a pravidlo 2.5.9 nemilosrdně vybere v j-tém řádku jako klíčový prvek zrovna a_{ij}.

Možnost nulového přírůstku účelové funkce vzbuzuje obavu, zda výpočet simplexovou metodou vůbec někdy skončí. Co když se nulový přírůstek bude opakovat v každém kroku simplexové metody?

Ukazuje se, že tato možnost nekonečného výpočtu je reálná. Existuje (uměle vymyšlený) příklad úlohy, kde simplexová metoda s volbou klíčového sloupce podle pravidla 2.6.7(1) opravdu nikdy neskončí, neustále se opakuje táž posloupnost několika simplexových tabulek, výpočet se zacyklí.

V literatuře se uvádí, že v praktických úlohách zacyklení nebylo zaznamenáno, údajně „díky“ zaokrouhlovacím chybám, které výpočet z cyklu vyvedou. Je ovšem lepší mít jistotu. Naštěstí existuje velmi jednoduchá pomoc. Je dokázáno, že při volbě klíčového sloupce podle pravidla 2.6.7(3) se výpočet nemůže zacyklit. Praktická rada tedy je tato:

> Za normálních okolností volte klíčový sloupec podle pravidla 1 (tedy nejzápornější relativní cenu), ale pokud se vyskytne nulový přírůstek účelové funkce, pak, dokud je řešení degenerované, používejte pravidlo 3 (prvý sloupec se zápornou relativní cenou).

Další komplikace s degenerovaným řešením je ta, že ne vždy hned poznáme optimální řešení. Degenerované řešení totiž je bázickým řešením pro několik různých bází. Může se tedy stát, že řešení reprezentované tabulkou již je de facto optimální, ale ne všechny relativní ceny jsou nezáporné. Ve výpočtu tedy je nutno pokračovat, přičemž následující simplexové tabulky reprezentují číselně totéž řešení, ale vyjádřené v jiné bázi. Viz příklad 2.6.9.

Poznámka: Je-li volba klíčového prvku ve sloupci nejednoznačná, tj. pokud se minimální podíl b_i/a_{ij} nabývá pro dva nebo více řádků i, není to nic závadného, klidně zvolte kterýkoli z nich.

V následující simplexové tabulce pak vyjde na pravé straně aspoň jedna nula a to v řádku, který není klíčový, ale týkal se jej onen minimální podíl. Řešení reprezentované touto následující tabulkou bude tedy degenerované.

2.6.9 Příklad — degenerace.

$$\begin{aligned} 4x_1 + x_2 + 4x_3 + 3x_4 &\to \max \\ x_1 + x_2 + 2x_4 &\leqq 2 \\ 2x_1 + x_2 + 4x_3 &\leqq 5 \\ x_1 + 4x_2 + 2x_3 + 4x_4 &\leqq 2 \\ x_1, \ldots, x_4 &\geqq 0 \end{aligned}$$

		4	1	4	3	0	0	0	max
0	5.	1	1	0	2	1	0	0	2
0	6.	2	1	4	0	0	1	0	5
0	7.	1	4	2	4	0	0	1	2
		−4	−1	−4	−3	0	0	0	0
4	1.	1	1	0	2	1	0	0	2
0	6.	0	−1	4	−4	−2	1	0	1
0	7.	0	3	2	2	−1	0	1	0
		0	3	−4	5	4	0	0	8
4	1.	1	1	0	2	1	0	0	2
0	6.	0	−7	0	−8	0	1	−2	1
4	3.	0	1½	1	1	−½	0	½	0
		0	9	0	9	2	0	2	8

V první tabulce volíme prvý sloupec jako klíčový, neboť je zde nejzápornější relativní cena. Volba klíčového prvku v tomto sloupci je nejednoznačná. Tedy ještě před provedením Jordanovy eliminace víme, že příští tabulka bude obsahovat degenerované řešení. Řešení reprezentované touto (první) tabulkou ovšem degenerované není.

Řešení reprezentované druhou tabulkou je degenerované (jak jsme již dříve předpověděli). Později se ukáže, že toto řešení již de facto je optimální, ale podle relativních cen to tak nevypadá. Chceme-li optimální řešení, musíme pokračovat volbou klíčového prvku ve třetím sloupci. Všimněte si, že díky nule na pravé straně volíme klíčový prvek a_{33}. Důsledkem je, že účelová funkce nevzroste.

V další (třetí) tabulce již víme, že máme optimální řešení. Je to ovšem stejné řešení jako v předchozí tabulce, pouze je vyjádřeno v jiné bázi.

2.6.10 Shora neomezená účelová funkce. Jde o případ znázorněný na obrázku 2.4, str. 14. Úloha má přípustné řešení, dokonce jich má mnoho, ale žádné z nich není optimální, neboť ke každému z nich existuje přípustné řešení s lepší účelovou funkcí.

Tuto situaci lze při výpočtu simplexovou metodou poznat poměrně snadno:

- nebázický sloupec a_j má všechny prvky nekladné, tj. $a_{ij} \leqq 0$,
- příslušná relativní cena je záporná, tj. $(z_j - c_j) < 0$.

V takovém případě, vezmeme-li polopřímku zmíněnou v 2.5.12, str. 22, pak všechny body této polopřímky jsou přípustnými řešeními, neboť pro rostoucí Δ všechny rovnice zůstávají zachovány a žádná souřadnice neklesá. Hodnota účelové funkce je pro každý bod této polopřímky rovna $L(x) - (z_j - c_j) \cdot \Delta$. To znamená, že zvolíme-li dostatečně velké Δ, můžeme na této polopřímce dostat přípustné řešení s libovolně vysokou hodnotou účelové funkce.

V takovém případě tedy optimum neexistuje. Je-li cílem úlohy nalezení optima, nemá smysl počítat dál, a to ani kdyby to šlo, tj. i kdyby bylo možno zvolit klíčový prvek v nějakém dalším

sloupci se zápornou relativní cenou. Sice bychom našli bázické přípustné řešení s lepší účelovou funkcí, na faktu neomezenosti to ovšem nic změnit nemůže, pouze bychom se zbytečně namáhali.

Poznámka: Pokud při řešení praktické úlohy zjistíte, že účelová funkce je shora neomezená, zpravidla to znamená, že v zadání úlohy chybí nějaká docela podstatná omezující podmínka.

2.6.11 Příklad — neomezená účelová funkce. Řešme úlohu 2.2.5, str. 14. Pro pohodlí čtenáře zde zopakujeme její zadání i obrázek 2.8:

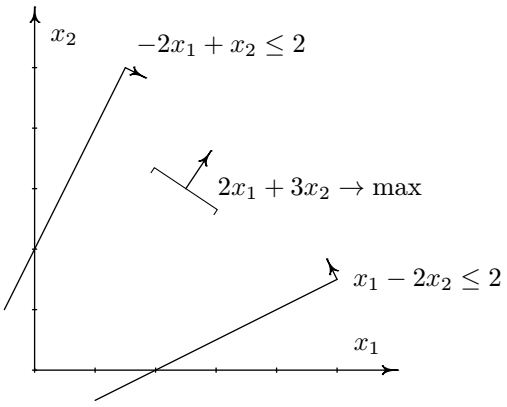
$$\begin{aligned} 2x_1 + 3x_2 &\to \max \\ x_1 - 2x_2 &\leqq 2 \\ -2x_1 + x_2 &\leqq 2 \\ x_1 &\geqq 0 \\ x_2 &\geqq 0 \end{aligned}$$

		2	3	0	0	max
0	3.	1	−2	1	0	2
0	4.	−2	1	0	1	2
		−2	−3	0	0	0
0	3.	−3	0	1	2	6
3	2.	−2	1	0	1	2
		−8	0	0	3	6

Druhá tabulka reprezentuje řešení $(0, 2, 6, 0)^{\mathrm{T}}$. Díky relativní ceně $(z_1 - c_1) = -8$ toto řešení nelze pokládat za optimální, ale ve výpočtu simplexovou metodou pokračovat nelze, protože v prvém sloupci není prvek, který by mohl být použit jako klíčový.

Relativní cena $(z_1 - c_1) = -8$ naznačuje, že by bylo vhodné zvyšovat prvou souřadnici. Vyjděme tedy z bodu $(0, 2, 6, 0)^{\mathrm{T}}$ a pohybujme se po hraně podle 2.5.12. Všechny body polopřímky $(0, 2, 6, 0)^{\mathrm{T}} + \Delta(1, 2, 3, 0)^{\mathrm{T}}$ jsou přípustnými řešeními. Pro bod polopřímky daný parametrem Δ je podle 2.6.1 hodnota účelové funkce rovna $-(z_1 - c_1) \cdot \Delta = -(-8)\Delta = 8\Delta$. Účelová funkce může nabývat libovolně vysokých hodnot, stačí zvolit vhodné $\Delta > 0$.

Žádné řešení tedy není optimální, ke každému přípustnému řešení existuje (na naší polopřímce) bod s lepší hodnotou účelové funkce.



Obrázek 2.8: Neomezená účelová funkce, úloha nemá optimální řešení.

Jako cvičení ověřte, že i kdybychom v prvé simplexové tabulce zvolili klíčový prvek v prvním sloupci, dostali bychom sice jiný bod a jinou polopřímku, ale závěr, že účelová funkce je shora neomezená, by byl stejný. Najděte obě polopřímky na obrázku 2.8.

2.6.12 (Ne)jednoznačnost optimálního řešení. Jsou-li relativní ceny všech nebázických sloupců kladné, pak řešení reprezentované touto simplexovou tabulkou je jediným optimálním řešením. Je totiž nejen optimální, ale navíc všechny směry k sousedním bázickým přípustným řešením vedou ve směru k ostře horším řešením.

V ostatních případech optimálního řešení, tj. když

- řešení reprezentované tabulkou je optimální,
- existuje nebázický sloupec s nulovou relativní cenou,

pak (až na výjimku degenerovaného případu) existuje další optimální řešení. Toto řešení zpravidla dostaneme jako sousední bázické přípustné řešení, které má stejnou hodnotu účelové funkce a je tedy také optimální.

Může se také stát, že v patřičném směru sousední řešení neexistuje, z bodu reprezentovaného tabulkou vychází polopřímka zmíněná v 2.5.12 a poněvadž $(z_j - c_j) = 0$, mají všechny body polopřímky stejnou účelovou funkci. A poněvadž počáteční bod polopřímky je optimálním řešením, jsou optimálními řešeními všechny body polopřímky.

Připomeňme, že množina optimálních řešení je vždy konvexní. Máme-li tedy dvě různá optimální řešení, pak všechny body úsečky, která tato dvě řešení spojuje, jsou optimálními řešeními.

Chceme-li získat všechna optimální řešení, je třeba nejprve najít všechny krajní body, které jsou optimálními řešeními (dělá se to podobně jako hledání všech BPŘ). Dále je třeba najít všechny případné krajní směry, které z těchto bodů vycházejí. Množina všech optimálních řešení pak je konvexním obalem všech těchto bodů.

2.6.13 Příklad — více optimálních řešení. Řešme úlohu z příkladu 2.2.3, str. 13. Tato úloha se od příkladu 2.6.6 liší pouze v účelové funkci. Pro pohodlí čtenáře zde zopakujeme celé zadání i obrázek 2.9:

$$\begin{aligned} 2x_1 + 2x_2 &\to \max \\ x_1 - 2x_2 &\leqq 2 \\ -2x_1 + x_2 &\leqq 2 \\ x_1 + x_2 &\leqq 5 \\ x_1, x_2 &\geqq 0 \end{aligned}$$

Po převodu na kanonický tvar dostaneme tyto simplexvé tabulky:

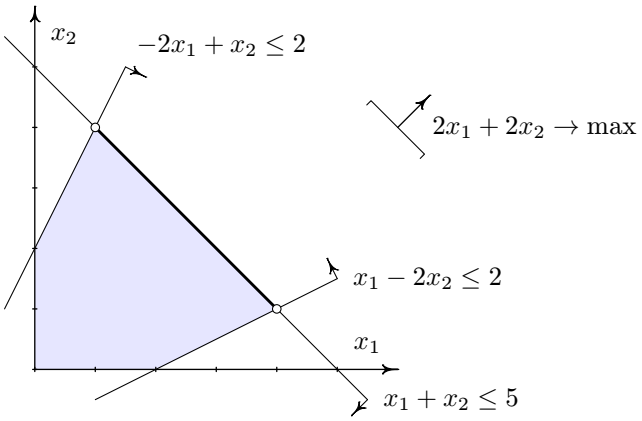
		2	2	0	0	0	max
0	3.	1	−2	1	0	0	2
0	4.	−2	1	0	1	0	2
0	5.	1	1	0	0	1	5
		−2	−2	0	0	0	0
0	3.	−3	0	1	2	0	6
2	2.	−2	1	0	1	0	2
0	5.	3	0	0	−1	1	3
		−6	0	0	2	0	4
0	3.	0	0	1	1	1	9
2	2.	0	1	0	$\frac{1}{3}$	$\frac{2}{3}$	4
2	1.	1	0	0	$-\frac{1}{3}$	$\frac{1}{3}$	1
		0	0	0	0	2	10

Všimněte si, že tyto simplexové tabulky obsahují stejné rozšířené matice soustavy rovnic, jako v příkladu 2.6.6, str. 24. Liší se pouze cenový vektor, vektory bázických cenových koeficientů a řádky relativních cen.

Třetí simplexová tabulka reprezentuje optimální řešení $(1, 4, 9, 0, 0)^{\mathrm{T}}$. Nulová relativní cena $(z_4 - c_4)$ v nebázickém sloupci ovšem signalizuje, že sousední BPŘ bude mít stejnou hodnotu účelové funkce a bude tedy také optimální. Vskutku, zvolíme-li klíčový prvek ve 4. sloupci, dostaneme další simplexovou tabulku

		2	2	0	0	0	max
0	4.	0	0	1	1	1	9
2	2.	0	1	$-\frac{1}{3}$	0	$\frac{1}{3}$	1
2	1.	1	0	$\frac{1}{3}$	0	$\frac{2}{3}$	4
		0	0	0	0	2	10

která reprezentuje další optimální řešení $(4, 1, 0, 9, 0)^{\mathrm{T}}$. Na obrázku 2.9 je množina optimálních řešení nakreslena tlustě.



Obrázek 2.9: Více optimálních řešení (příklad 2.2.3).

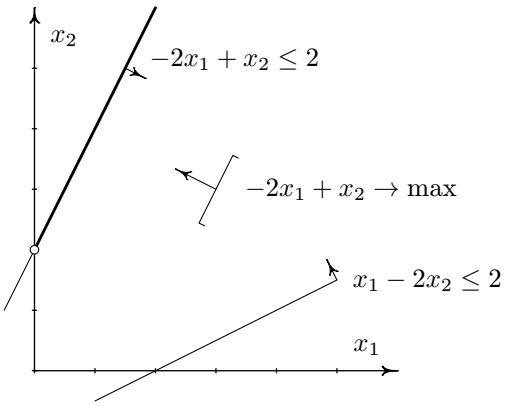
2.6.14 Příklad — neomezená množina optimálních řešení. Řešme úlohu 2.2.6, str. 15. Pro pohodlí čtenáře opět zopakujeme její zadání i obrázek 2.10:

$$\begin{aligned} -2x_1 + x_2 &\to \max \\ x_1 - 2x_2 &\leqq 2 \\ -2x_1 + x_2 &\leqq 2 \\ x_1 &\geqq 0 \\ x_2 &\geqq 0 \end{aligned}$$

		-2	1	0	0	max
0	3.	1	-2	1	0	2
0	4.	-2	1	0	1	2
		2	-1	0	0	0
0	3.	-3	0	1	2	6
1	2.	-2	1	0	1	2
		0	0	0	1	2

Druhá tabulka reprezentuje řešení $(0, 2, 6, 0)^{\mathrm{T}}$. Všechny relativní ceny jsou nezáporné, toto řešení je tedy optimální. Relativní cena $(z_1 - c_1) = 0$ signalizuje, že by úloha mohla mít další optimální řešení. Tentokrát však v prvém sloupci nelze volit klíčový prvek. Co teď?

Nulová relativní cena naznačuje, že při pohybu po hraně podle 2.5.12 z bodu $(0,2,6,0)^{\mathrm{T}}$ se hodnota účelové funkce nebude měnit. Vskutku, vezměme polopřímku tvořenou body $(0,2,6,0)+$ $+\Delta(1,2,3,0)$ pro $\Delta \geqq 0$. Všechny body této polopřímky jsou přípustnými řešeními (podobně jako v příkladu 2.6.11), ale navíc zde všechny body polopřímky mají stejnou hodnotu účelové funkce jako její počátek, všechny body polopřímky jsou tedy také optimálními řešeními.



Obrázek 2.10: Neomezená množina optimálních řešení (příklad **??**).

2.7 Umělé proměnné

2.7.1 Existence přípustného řešení. Máme-li úlohu LP v kanonickém tvaru a její simplexová tabulka obsahuje úplnou kanonickou bázi, pak tato úloha má přípustné řešení. Konkrétně totiž např. řešení reprezentované touto simplexovou tabulkou je bázickým přípustným řešením.

Pokud úplnou kanonickou bázi v simplexové tabulce nemáme, přípustné řešení dané úlohy může a nemusí existovat. A právě o tom pojednává tato sekce.

2.7.2 Umělé proměnné a M-úloha. Jestliže výchozí simplexová tabulka neobsahuje úplnou kanonickou bázi, úvahy uvedené v předchozím oddíle 2.6 neplatí, neboť tyto úvahy závisely na předpokladu, že úplnou kanonickou bázi v tabulce máme. Lze si však snadno pomoci:

Chybějící bázické sloupce zcela formálně do simplexové tabulky přidáme. Proměnné odpovídající těmto sloupcům nazýváme *umělými proměnnými*. Aby výsledná úloha byla v kanonickém tvaru, požadujeme i pro umělé proměnné jejich nezápornost.

Trik je založen na této úvaze: Pokud se při výpočtu podaří všechny umělé proměnné vynulovat, bude je možno ze simplexové tabulky odstranit, a to, co zbude, bude přípustným řešením původní úlohy. K vynulování umělých proměnných použijeme již známou simplexovou metodu a vhodnou „cenovou politiku“.

Cenové koeficienty umělých proměnných proto volíme rovny $-M$, kde $M \gg 0$. Záporné ceny volíme proto, aby maximalizace účelové funkce měla za následek zmenšení hodnot umělých proměnných, hodnoty umělých proměnných jsou zápornými cenami „tlačeny k nule“. Zároveň tyto ceny potřebujeme „velmi záporné“ (a tedy hodnotu $M \gg 0$), aby vliv těchto cenových koeficientů $-M$ zaručeně převážil nad cenami původních, tj. ne-umělých proměnných.

Takto vzniklou rozšířenou úlohu LP nazýváme M-úlohou.

Umělé proměnné byly do M-úlohy přidány ze zcela formálních důvodů, reálný význam od nich nebyl požadován (jsou umělé), lze je však chápat i tak, že vyjadřují míru nesplnění omezujících podmínek. Pokud umělé proměnné vynulujete, máte přípustné řešení.

2.7.3 Konstanta M. Konstantu M je třeba volit tak, aby její vliv v cenových koeficientech převážil nad vlivem jiných cenových koeficientů. Hodnota M nemusí být nekonečná (s tou by se špatně počítalo). Stačí, aby $M > m \cdot (\max |c_j|) \cdot (\max |x_j|)/(\min |x_j|)$, kde $\max |x_j|$ a $(\min |x_j|)$ jsou maximální a minimální absolutní hodnota nenulové souřadnice některého BPŘ a $(\max |c_j|)$ je maximum absolutních hodnot všech cenových koeficientů původní úlohy.

Samozřejmě lze zvolit i konstantu větší, předem se však obtížně odhaduje jak velkou. Navíc výpočet s tak velkými čísly by byl zatížen značnými zaokrouhlovacími chybami.

Proto se konstanta M používá jen v teoretických úvahách. Při praktickém výpočtu konstantu M používáme jen formálně: Všechny relativní ceny a všechny vybrané cenové koeficienty c_B chápeme jako výrazy tvaru $p + qM$, kde p, q jsou reálná čísla. Výraz $p + qM$ lze zaznamenat jako uspořádanou dvojici čísel (p, q). Tyto dvojice lze po složkách sečítat, lze je po složkách násobit číslem. Potřebujeme-li tyto výrazy porovnávat, řídíme se nejprve podle koeficientů u M a teprve když ty nerozhodnou, porovnáme absolutní členy výrazů. Tedy

$$p + qM < p' + q'M \qquad \Longleftrightarrow \qquad (q < q') \quad \text{nebo} \quad [(q = q') \ a \ (p < p')]$$

Pokud chcete mít v simplexové tabulce jen obyčejná čísla a ne výrazy typu $p+qM$, lze to vyřešit tak, že řádek relativních cen nahradíte dvěma řádky, z nichž jeden bude obsahovat koeficienty u M a druhý bude obsahovat absolutní členy. Dobrá zpráva je, že Jordanovu eliminaci lze dělat s oběma těmito řádkami samostatně.

2.7.4 M-úloha má vždy přípustné řešení, neboť je to úloha v kanonickém tvaru a má úplnou kanonickou bázi. Tím nic neříkáme o existenci přípustného řešení původní úlohy.

2.7.5 Neexistence přípustného řešení. Pokud optimální řešení M-úlohy má alespoň jednu umělou proměnnou nenulovou (tedy kladnou), tj. pokud účelová funkce tohoto řešení je $p - qM$ pro nějaké nenulové $q > 0$, pak původní úloha nemá žádné přípustné řešení.

Důkaz: Kdyby totiž nějaké přípustné řešení měla, pak by toto řešení, doplněné o nulové umělé proměnné, bylo lepším řešením M-úlohy, než nalezené optimální řešení, neboť hodnota účelové funkce by měla nulovou složku q. To je ovšem spor, protože žádné řešení nemůže být lepší než optimální. Původní úloha tedy nemá žádné přípustné řešení.

2.7.6 Existence přípustného řešení. Když nějaké přípustné řešení M-úlohy má všechny umělé proměnné nulové, pak vynecháním umělých proměnných dostaneme přípustné řešení původní úlohy.

2.7.7 Optimální řešení M-úlohy. Je-li řešení M-úlohy optimální a má všechny umělé proměnné rovny nule, pak vynecháním umělých proměnných dostaneme řešení, které je nejen přípustným, ale dokonce optimálním řešením původní úlohy.

2.7.8 Příklad – neexistence přípustného řešení. Řešme tuto úlohu (viz obr. 2.11):

$$\begin{aligned} 2x_1 + 3x_2 &\to \max \\ 2x_1 - x_2 &\geqq 2 \\ -x_1 + 2x_2 &\geqq 2 \\ x_1 + x_2 &\leqq 3 \\ x_1, x_2 &\geqq 0 \end{aligned}$$

Po převodu na kanonický tvar dostaneme úlohu s touto simplexovou tabulkou:

		2	3	0	0	0	max
	.	2	−1	−1	0	0	2
	.	−1	2	0	−1	0	2
0	5.	1	1	0	0	1	3

Tato tabulka neobsahuje kanonickou bázi, proto ani nemá smysl počítat relativní ceny $z_j - c_j$. Po doplnění dvou umělých proměnných dostaneme simplexovou tabulku:

		2	3	0	0	0	$-M$	$-M$	max
$-M$	6.	2	−1	−1	0	0	1	0	2
$-M$	7.	−1	2	0	−1	0	0	1	2
0	5.	1	1	0	0	1	0	0	3
		−2	−3	0	0	0	0	0	0
		$-M$	$-M$	M	M	0	0	0	$-4M$

Řádek relativních cen $z_j - c_j$ je zde rozdělen na dvě části. V horní je složka „bez M“ a v dolní polovině řádky je složka „s násobky M“. Tedy např. pro prvý sloupec je $z_1 - c_1 = -2 - M$.

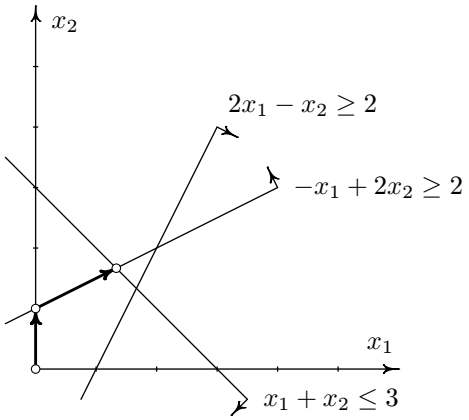
		2	3	0	0	0	$-M$	$-M$	max
$-M$	6.	2	−1	−1	0	0	1	0	2
$-M$	7.	−1	2	0	−1	0	0	1	2
0	5.	1	1	0	0	1	0	0	3
		−2	−3	0	0	0	0	0	0
		$-M$	$-M$	M	M	0	0	0	$-4M$
$-M$	6.	$1+1/2$	0	−1	$-\frac{1}{2}$	0	1	$\frac{1}{2}$	3
3	2.	$-1/2$	1	0	$-\frac{1}{2}$	0	0	$\frac{1}{2}$	1
0	5.	$1+1/2$	0	0	$\frac{1}{2}$	1	0	$-\frac{1}{2}$	2
		$-3-1/2$	0	0	$-1\frac{1}{2}$	0	0	$1\frac{1}{2}$	3
		$-M-1/2M$	0	M	$\frac{1}{2}M$	0	0	$\frac{1}{2}M$	$-3M$
$-M$	6.	0	0	−1	−1	−1	1	1	1
3	2.	0	1	0	$-\frac{1}{3}$	$\frac{1}{3}$	0	$\frac{1}{3}$	$1\frac{2}{3}$
2	1.	1	0	0	$\frac{1}{3}$	$\frac{2}{3}$	0	$-\frac{1}{3}$	$1\frac{1}{3}$
		0	0	0	$-\frac{1}{3}$	$2\frac{1}{3}$	0	$\frac{1}{3}$	$7\frac{2}{3}$
		0	0	M	M	M	0	0	$-M$

M-úloha má optimální řešení $(1\frac{1}{3}, 1\frac{2}{3}, 0, 0, 0, 1, 0)^{\mathrm{T}}$. V tomto řešení je nenulová umělá proměnná $x_6 = 1$. Že je některá umělá proměnná nenulová, je na první pohled vidět z hodnoty účelové funkce $L = -M + 7\frac{2}{3}$, konkrétně z nenulové složky $-M$. Původní úloha tedy nemá žádné přípustné řešení.

2.7.9 Příklad – optimální řešení. Řešme úlohu (viz obr. 2.12), která se od úlohy 2.7.8 jen nepatrně liší na pravé straně třetí strukturní podmínky:

$$\begin{aligned} 2x_1 + 3x_2 &\to \max \\ 2x_1 - x_2 &\geqq 2 \\ -x_1 + 2x_2 &\geqq 2 \\ x_1 + x_2 &\leqq 5 \\ x_1, x_2 &\geqq 0 \end{aligned}$$

Podobně jako v úloze 2.7.8 je třeba přidat dvě umělé proměnné.



Obrázek 2.11: Příklad k M-úloze — nemá přípustné řešení (viz 2.7.8).

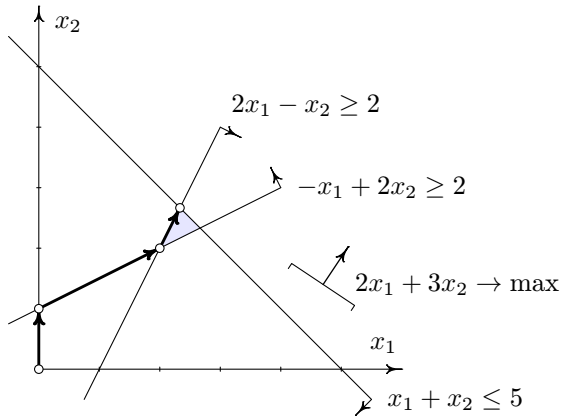
		2	3	0	0	0	$-M$	$-M$	max
$-M$	6.	2	-1	-1	0	0	1	0	2
$-M$	7.	-1	2	0	-1	0	0	1	2
0	5.	1	1	0	0	1	0	0	5
		-2	-3	0	0	0	0	0	0
		$-M$	$-M$	M	M	0	0	0	$-4M$
$-M$	6.	$1 + 1/2$	0	-1	$-\frac{1}{2}$	0	1	$\frac{1}{2}$	3
3	2.	$-1/2$	1	0	$-\frac{1}{2}$	0	0	$\frac{1}{2}$	1
0	5.	$1 + 1/2$	0	0	$\frac{1}{2}$	1	0	$-\frac{1}{2}$	4
		$-3 - 1/2$	0	0	$-1\frac{1}{2}$	0	0	$1\frac{1}{2}$	3
		$-M - 1/2M$	0	M	$\frac{1}{2}M$	0	0	$\frac{1}{2}M$	$-3M$
2	1.	1	0	$-\frac{2}{3}$	$-\frac{1}{3}$	0	$\frac{2}{3}$	$\frac{1}{3}$	2
3	2.	0	1	$-\frac{1}{3}$	$-\frac{2}{3}$	0	$\frac{1}{3}$	$\frac{2}{3}$	2
0	5.	0	0	1	1	1	-1	-1	1
		0	0	$-2\frac{1}{3}$	$-2\frac{2}{3}$	0	$2\frac{1}{3}$	$2\frac{2}{3}$	10
		0	0	0	0	0	M	M	0
2	1.	1	0	$-\frac{1}{3}$	0	$\frac{1}{3}$	$\frac{1}{3}$	0	$2\frac{1}{3}$
3	2.	0	1	$\frac{1}{3}$	0	$\frac{2}{3}$	$-\frac{1}{3}$	0	$2\frac{2}{3}$
0	4.	0	0	1	1	1	-1	-1	1
		0	0	$\frac{1}{3}$	0	$2\frac{2}{3}$	$-\frac{1}{3}$	0	$12\frac{2}{3}$
		0	0	0	0	0	M	M	0

Zde optimální řešení M-úlohy je $(2\frac{1}{3}, 2\frac{2}{3}, 0, 1, 0, 0, 0)^{\mathrm{T}}$. Obě umělé proměnné jsou zde nulové, tedy vektor $(2\frac{1}{3}, 2\frac{2}{3}, 0, 1, 0)^{\mathrm{T}}$ je optimálním (a přípustným) řešením úlohy v kanonickém tvaru a vektor $(2\frac{1}{3}, 2\frac{2}{3})^{\mathrm{T}}$ je optimálním řešením původní úlohy.

Poznámka: Všimněte si, že již ve 3. simplexové tabulce bylo jasné, že původní úloha má přípustné řešení, neboť všechny (obě) umělé proměnné byly nulové. Všimněte si také, že když se umělé proměnné dostaly ven z báze, jejich kladné a velmi vysoké relativní ceny brání tomu, aby se do báze znovu dostaly.

2.7.10 Vylučovat umělé proměnné? Pokud je umělá proměnná vyloučena z báze, pak se díky relativním cenám nemůže znovu do báze vrátit. Je tedy možné celý sloupec této proměnné odstranit ze simplexové tabulky a dále počítat bez něj. Jde-li nám pouze o optimální řešení a nechceme dělat další výpočty, můžeme si vynecháváním umělých proměnných trochu usnadnit výpočet.

Pro řešení některých složitějších otázek však potřebujeme ve výsledné simplexové tabulce znát,



Obrázek 2.12: Příklad k M-úloze — má optimální řešení (viz 2.7.9).

jak vypadá sloupec, který byl členem původní báze. Pak samozřejmě musíme sloupce všech umělých proměnných v simplexové tabulce ponechat a provádět výpočet i s nimi.

2.7.11 Dvoufázová simplexová metoda.

1. fáze: Cílem prvé fáze je transformovat původní úlohu tak, aby simplexová tabulka obsahovala úplnou kanonickou bázi.

Do úlohy zavedeme umělé proměnné a jejich ceny zvolíme -1, ceny ostatních proměnných volíme v 1. fázi nulové. Tuto pomocnou úlohu řešíme běžnou simplexovou metodou a najdeme její optimální řešení (zaručeně existuje).

Pokud optimální řešení prvé fáze má nenulovou umělou proměnnou (a tedy i nenulovou hodnotu účelové funkce), pak původní úloha nemá žádné přípustné řešení (z podobných důvodů jako v 2.7.5). Výpočet v tomto případě končí.

2. fáze: Pokud výpočet neskončil v 1. fázi, odstraníme všechny umělé proměnné (jsou totiž nulové), vrátíme se k cenovým koeficientům z původní úlohy a úlohu vyřešíme běžnou simplexovou metodou.

Dvoufázová simplexová metoda má proti M-úloze jednu výhodu a dvě nevýhody.

Výhodou dvoufázové metody je, že se v ní používá zcela standardní simplexová metoda, kde všechny cenové koeficienty i relativní ceny jsou obyčejná reálná čísla a ne výrazy typu $p + qM$.

Dvoufázová metoda je však jednodušší pouze zdánlivě, je totiž komplikována následující možností: výsledkem prvé fáze může být degenerované řešení, ve kterém umělá proměnná sice má nulovou hodnotu, ale je součástí kanonické báze. Pokud k tomu dojde, nelze všechny umělé proměnné jednoduše odstranit, protože bychom tak přišli o kanonickou bázi. Způsob, jak postupovat v takové situaci sice existuje, ale není zcela triviální (zvolíme záporný klíčový prvek podle 2.9.7, bod 2.).

Hlavní nevýhodou dvoufázové metody je nižší efektivnost. V prvé fázi se totiž hledá jakékoli přípustné řešení bez ohledu na účelovou funkci původní úlohy. Přitom pomocná úloha řešená v prvé fázi typicky má víceznačné optimum. Je tedy zcela náhodné, které řešení se stane východiskem pro druhou fázi výpočtu. Kdybychom v prvé fázi v případě nejednoznačnosti mohli vzít v úvahu i cenové koeficienty původní účelové funkce, mohla by druhá fáze výpočtu začínat z lepšího výchozího řešení a mohla by tedy být (v průměru) rychlejší. Právě na této myšlence je založena M-úloha a proto se v ní používají tak divné cenové koeficienty.

2.7.12 Dvě fáze řešení M-úlohy. Při řešení M-úlohy lze také rozlišovat dvě fáze řešení. Prvá fáze trvá, dokud má alespoň jedna umělá proměnná nenulovou hodnotu. Pak následuje druhá fáze. Ve druhé fázi již hodnoty umělých proměnných zůstávají nulové. (Tím nic neříkáme o tom, zda umělé proměnné jsou nebo nejsou v bázi. V případě degenerace v bázi zůstat mohou.)

2.8 Součinový tvar matice soustavy

2.8.1 Řádkové úpravy a násobení matic. Mějme libovolnou matici, označme ji třeba A. Řádkovými úpravami matice A zde rozumíme

- násobení řádku matice A číslem,
- přičtení násobku řádku matice A k jinému řádku téže matice,
- záměnu dvou řádků matice A.

Stejného efektu jako těmito řádkovými úpravami lze dosáhnout také tak, že matici A vynásobíme zleva vhodnou čtvercovou maticí B.

Kde matici B vezmeme? Vezměte jednotkovou matici E řádu m a proveďte s ní zamýšlenou řádkovou úpravu. Výsledná čtvercová matice (označme ji B) má tu vlastnost, že když touto maticí B vynásobíme zleva jakoukoli matici A o m řádcích, výsledek bude stejný jako kdybychom tu řádkovou úpravu provedli přímo s maticí A. Vyzkoušejte si to na několika příkladech, opravdu to funguje.

Pozor, je třeba násobit maticí B zleva. Násobení zprava by dělalo sloupcové úpravy a ty zde teď nepotřebujeme.

Provádíme-li sérii několika řádkových úprav, můžeme se na to dívat tak, že jsme matici postupně vynásobili zleva několika čtvercovými maticemi. A poněvadž násobení matic je asociativní (tj. nezáleží na uzávorkování), i série mnoha řádkových úprav je totéž jako vynásobení vhodnou čtvercovou maticí zleva.

2.8.2 Součinový tvar matice soustavy. Uvažujme dvě simplexové tabulky: první (výchozí), ze které byl zahájen výpočet a poslední, která je výsledkem dosavadního (možná i neúplného) výpočtu. Označme B čtvercovou matici tvořenou těmi sloupci výchozí tabulky, které jsou v poslední tabulce členy kanonické báze. Pořadí sloupců v matici B je důležité a je odvozeno od pořadí sloupců v poslední kanonické bázi.

Rozšířenou matici soustavy rovnic v poslední tabulce jsme dostali posloupností řádkových úprav. Téhož výsledku jsme mohli dosáhnout také tím, že bychom výchozí matici soustavy vynásobili zleva maticí B^{-1}, tj. inverzní maticí k matici B.

2.8.3 Příklad. V úloze 2.7.9 je výchozí kanonická báze tvořena sloupci 6, 7 a 5 v tomto pořadí. Pozor, pořadí je důležité.

Vezměme poslední simplexovou tabulku a z ní tytéž sloupce 6, 7 a 5 ve stejném pořadí, v jakém byly ve výchozí tabulce. Dostaneme matici

$$B^{-1} = \begin{pmatrix} 1/3 & 0 & 1/3 \\ -1/3 & 0 & 2/3 \\ -1 & -1 & 1 \end{pmatrix}$$

Kdybychom tuto matici znali předem, mohli bychom poslední simplexovou tabulku (přesněji, její podstatnou část, totiž rozšířenou matici soustavy rovnic) získat tak, že bychom matici soustavy z výchozí tabulky vynásobili zleva touto maticí B^{-1}. Ověřte.

Mimochodem, toto platí nejen pro poslední tabulku, ale pro každou. Každá má ovšem „svou“ matici.

2.8.4 Revidovaná simplexová metoda se od běžné simplexové metody liší tím, že výchozí rozšířenou matici soustavy ponechává v původním stavu. Namísto, aby se Jordanovou eliminací upravovala celá matice soustavy, upravuje se pouze menší matice B^{-1}. Relativní ceny $z_j - c_j$ se počítají podle vzorce $z_j - c_j = B^{-1}a_j - c_j$ kde a_j je j-tý sloupec původní matice soustavy.

Proč tak složitě? Výhoda se projeví u úloh, jejichž matice soustavy je hodně velká a řídká (má hodně malé procento nenulových prvků). Řídké matice lze v paměti počítače ukládat úsporněji a lze s nimi efektivně pracovat, ale Jordanovou eliminací by řídkost matice velmi rychle vzala za své.

Počítačové programy pro řešení rozsáhlých úloh LP jsou zpravidla založeny na revidované simplexové metodě (ale využívají ještě další triky).

2.9 Změny ve vypočtené úloze

Může se stát, že máte vypočteno optimální řešení a dodatečně dojde ke změně zadání. Ukážeme, že není třeba opakovat celý výpočet. Předpokladem je, že ke stávajícímu optimálnímu řešení je k dispozici i odpovídající simplexová tabulka.

2.9.1 Rozbor stability řešení se zabývá otázkou, jak moc se může změnit zadání úlohy, aby to nemělo vliv na optimální řešení.

Měnit se mohou cenové koeficieny (což je v praxi velmi časté), pravé strany (např. disponibilní množství materiálu), nebo strukturní koeficienty (např. vlivem změny technologie výroby).

2.9.2 Jednorázová změna cen. Je třeba si uvědomit, že množina přípustných řešení se nezměnila. V simplexové tabulce se tedy změní pouze horní řádek c^{T} a tato změna má vliv na sloupec bázických cenových koeficientů c_B a na řádek relativních cen $z_j - c_j$. Zbytek tabulky, tj. zejména rozšířená matice soustavy rovnic, zůstává beze změny.

Stačí tedy spočítat nové relativní ceny. Vyjdou-li všechny nezáporné, pak řešení reprezentované tabulkou je optimálním řešením i po změně cen. Vyjde-li některá relativní cena záporná, lze jednoduše pokračovat ve výpočtu simplexovou metodou a najít nové optimální řešení nové úlohy.

2.9.3 Příklad. V příkladě 2.6.6 změníme původní cenový vektor $(2, 3)$ na $(3, 4)$. Připomeňme, že simplexová tabulka s optimálním řešením byla

		2	3	0	0	0	max
0	3.	0	0	1	1	1	9
3	2.	0	1	0	$\frac{1}{3}$	$\frac{2}{3}$	4
2	1.	1	0	0	$-\frac{1}{3}$	$\frac{1}{3}$	1
		0	0	0	$\frac{1}{3}$	$2\frac{2}{3}$	14

Změní-li se cenový vektor např. na $(3, 4)$, změní se tabulka takto:

		3	4	0	0	0	max
0	3.	0	0	1	1	1	9
4	2.	0	1	0	$\frac{1}{3}$	$\frac{2}{3}$	4
3	1.	1	0	0	$-\frac{1}{3}$	$\frac{1}{3}$	1
		0	0	0	$\frac{1}{3}$	$3\frac{2}{3}$	19

Relativní ceny se sice změnily, ale zůstaly nezáporné. Řešení $(1, 4, 9, 0, 0)$ tedy zůstalo optimální.

Uvažujme nyní jinou změnu cen, řekněme na $(4, 3)$. V tomto případě se simplexová tabulka změní na

		4	3	0	0	0	max
0	3.	0	0	1	1	1	9
3	2.	0	1	0	$\frac{1}{3}$	$\frac{2}{3}$	4
4	1.	1	0	0	$-\frac{1}{3}$	$\frac{1}{3}$	1
		0	0	0	$-\frac{1}{3}$	$3\frac{1}{3}$	16

Tentokrát relativní ceny nepotvrzují, že řešení $(1, 4, 9, 0, 0)$ je optimální. Chceme-li optimální řešení změněné úlohy, je třeba pokračovat ve výpočtu. Podstatné ovšem je, že není třeba opakovat celý předcházející výpočet od začátku, lze si ušetřit spoustu práce tím, že budeme pokračovat od poslední simplexové tabulky, samozřejmě po provedení výše popsané změny.

2.9.4 Změna cen lineárně závislá na parametru. Ceny se v praxi neustále mění. Uvažujme změnu cen lineárně závislou na čase t a to takovou, že původní ceny (pro které máme vypočteno optimální řešení) odpovídají času $t = 0$. Cenový vektor $c + td$ tedy je lineární funkcí času.

Otázka je, v jakém rozsahu parametru t zůstane stávající řešení optimální. Výsledkem bude interval hodnot parametru t.

Řešení je opět snadné, stačí v simplexové tabulce počítat s cenami závislými na parametru t. Z předchozího víme, že změna se týká pouze bázických cenových koeficientů a (hlavně) relativních cen. Relativní ceny ovšem vyjdou závislé na parametru t.

Aby řešení reprezentované tabulkou bylo možno pokládat za optimální, musí být všechny relativní ceny nezáporné. Pro každou nebázickou relativní cenu tedy máme nerovnost (která říká, že tato relativní cena je nezáporná). Řešením takto vzniklé soustavy lineárních nerovností je interval a nebo prázdná množina.

2.9.5 Příklad lineární změny cen. V příkladě 2.6.6 uvažujme změnu původního cenového vektoru $(2, 3)$ na vektor $(2 + 2t, 3 + t)$, kde t je parametr. Simplexová tabulka se (podobně jako v 2.9.3) změní na

		$2+2t$	$3+t$	0	0	0	max
0	3.	0	0	1	1	1	9
$3+t$	2.	0	1	0	$\frac{1}{3}$	$\frac{2}{3}$	4
$2+2t$	1.	1	0	0	$-\frac{1}{3}$	$\frac{1}{3}$	1
		0	0	0	$\frac{1}{3} - \frac{1}{3}t$	$2\frac{2}{3} + 1\frac{1}{3}t$	$14 + 6t$

Řešení $(1, 4, 9, 0, 0)$ reprezentované tabulkou bude optimálním řešením, budou-li relativní ceny nezáporné, tedy bude-li platit

$$\begin{aligned} \tfrac{1}{3} - \tfrac{1}{3}t &\geqq 0 \\ 2\tfrac{2}{3} + 1\tfrac{1}{3}t &\geqq 0 \end{aligned}$$

tedy bude-li $-2 \leqq t \leqq 1$.

Pro $t > 1$ již stávající řešení není optimální, neboť relativní cena $z_4 - c_4$ je záporná. Chceme-li znát optimální řešení pro $t > 1$, zvolíme klíčový prvek ve 4. sloupci a provedeme další krok simplexové metody. Dostaneme tabulku

		$2+2t$	$3+t$	0	0	0	max
0	4.	0	0	1	1	1	9
$3+t$	2.	0	1	$-\frac{1}{3}$	0	$\frac{1}{3}$	1
$2+2t$	1.	1	0	$\frac{1}{3}$	0	$\frac{2}{3}$	4
		0	0	$-\frac{1}{3} + \frac{1}{3}t$	0	$2\frac{1}{3} + 1\frac{2}{3}t$	$11 + 9t$

Tato tabulka reprezentuje řešení $(4,1,0,9,0)$ a toto řešení je optimálním řešením pro všechna $t \geqq 1$.

Všimněte si, že pro $t = 1$ zde máme dvě simplexové tabulky, které reprezentují dvě různá optimální řešení $(1,4,9,0,0)$ a $(4,1,0,9,0)$. Množina optimálních řešení je v tomto případě úsečka mezi těmito dvěma body.

Podobně, chceme-li znát optimum pro $t < -2$, zvolíme klíčový prvek v 5. sloupci a po provedení jednoho kroku simplexové metody dostaneme tabulku

		$2+2t$	$3+t$	0	0	0	max
0	3.	-3	0	1	2	0	6
$3+t$	2.	-2	1	0	1	0	2
0	5.	3	0	0	-1	1	3
		$-8-4t$	0	0	$3+t$	0	$6+2t$

která reprezentuje řešení $(0,2,6,0,3)$. Toto řešení je optimálním řešením pro hodnoty parametru $-3 \leqq t \leqq -2$.

Pro hodnoty parametru $t < -3$ zvolíme klíčový prvek ve 4. sloupci a po provedení eliminace dostaneme simplexovou tabulku

		$2+2t$	$3+t$	0	0	0	max
0	3.	1	-2	1	0	0	2
0	4.	-2	1	0	1	0	2
0	5.	1	1	0	0	1	5
		$-2-2t$	$-3-t$	0	0	0	0

Řešení $(0,0,2,2,5)$ reprezentované touto tabulkou je optimálním řešením pro všechna $t < -3$.

Závislost optimálního řešení na hodnotě parametru t lze shrnout takto:

parametr	optimální řešení
$t < -3$	$(0,0,2,2,5)$
$t = -3$	úsečka $(0,0,2,2,5)(0,2,6,0,3)$
$-3 < t < -2$	$(0,2,6,0,3)$
$t = -2$	úsečka $(0,2,6,0,3)(1,4,9,0,0)$
$-2 < t < 1$	$(1,4,9,0,0)$
$t = 1$	úsečka $(1,4,9,0,0)(4,1,0,9,0)$
$t > 1$	$(4,1,0,9,0)$

2.9.6 Změna pravých stran může v praxi znamenat např. změnu disponibilního množství nějakého zdroje. Materiál mohla vzít velká voda, dodavatel mohl vypovědět smlouvu nebo naopak uvažujeme o koupi dalšího materiálu. Ve všech těchto případech je třeba zjistit vliv změny omezujících podmínek na optimální řešení a na jeho účelovou funkci.

Na rozdíl od měnících se cen, které mají za následek skokové změny optimálního řešení, měnící se pravé strany mají za následek změny spojité. Rovněž zde není třeba opakovat celý výpočet. Je-li k dispozici simplexová tabulka s optimálním řešením, a víme-li jak na to, lze účinek změny pravých stran poměrně snadno vyhodnotit.

Je-li v úloze původní vektor pravých stran b nahrazen vektorem b', pak v simplexové tabulce, která reprezentuje optimální řešení, se změní pouze vektor pravých stran. Vyplývá to z poznatků 2.8.2 o součinovém tvaru matice soustavy rovnic. Zatímco v původní úloze to bylo $B^{-1}b$, ve změněné úloze to bude $B^{-1}b'$.

Máme-li tedy simplexovou tabulku, která reprezentuje optimální řešení (tj. relativní ceny jsou nezáporné), a známe-li polohu původní kanonické báze ve výchozí simplexové tabulce, snadno ze simplexové tabulky optimálního řešení zjistíme matici B^{-1} a vypočteme změněnou pravou stranu $B^{-1}b'$.

Pokud ve změněné tabulce vyjdou všechny pravé strany nezáporné, pak řešení reprezentované touto tabukou je optimálním řešením změněné úlohy.

Vyjde-li některá pravá strana záporná, pak řešení reprezentované tabulkou není přípustné. V takovém případě si lze pomoci takzvanou duálně simplexovou metodou.

2.9.7 Duálně simplexová metoda se od normální simplexové metody liší výchozími předpoklady a jinou volbou klíčovho prvku. Předpokládáme, že relativní ceny jsou nezáporné.

1. Zvolíme klíčový řádek i, ve kterém $b_i < 0$.
2. V i-tém řádku zvolíme klíčový prvek $a_{ij} < 0$ takový, že podíl $(z_j - c_j)/a_{ij}$ je největší (ovšem záporný, tedy nejblíže nule).
3. Pak provedeme Jordanovu eliminaci jako v běžné simplexové metodě.

Tyto kroky opakujeme, dokud nejsou všechny pravé strany nezáporné.

Duálně simplexová metoda zachovává nezápornost relativních cen a snaží se dosáhnout nezápornosti pravých stran.

2.9.8 Příklad změny pravých stran. V příkladě 2.6.6, str. 24 uvažujme změnu původního vektoru pravých stran $(2, 2, 5)$ na nový vektor $(2, 3, 4)$.

Připomeňme, že ve výchozí simplexové tabulce tvořily kanonickou bázi sloupce 3., 4. a 5. (v tomto pořadí) a simplexová tabulka s optimálním řešením byla

		2	3	0	0	0	max
0	3.	0	0	1	1	1	9
3	2.	0	1	0	$\frac{1}{3}$	$\frac{2}{3}$	4
2	1.	1	0	0	$-\frac{1}{3}$	$\frac{1}{3}$	1
		0	0	0	$\frac{1}{3}$	$2\frac{2}{3}$	14

Nová pravá strana v tabulce s optimálním řešením je

$$B^{-1} \cdot \begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 0 & \frac{1}{3} & \frac{2}{3} \\ 0 & -\frac{1}{3} & \frac{1}{3} \end{pmatrix} \cdot \begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix} = \begin{pmatrix} 9 \\ 3\frac{2}{3} \\ \frac{1}{3} \end{pmatrix}$$

a nová simplexová tabulka vypadá takto:

		2	3	0	0	0	max
0	3.	0	0	1	1	1	9
3	2.	0	1	0	$\frac{1}{3}$	$\frac{2}{3}$	$3\frac{2}{3}$
2	1.	1	0	0	$-\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$
		0	0	0	$\frac{1}{3}$	$2\frac{2}{3}$	$11\frac{2}{3}$

Optimální řešení se tedy změní na $(\frac{1}{3}, 3\frac{2}{3}, 9, 0, 0)$.

2.9.9 Změna pravých stran závislá na parametru. Jsou-li pravé strany strukturních podmínek lineárně závislé na parametry t, můžeme pomocí násobení maticí B^{-1} zleva změnu promítnout do simplexové tabulky, která representuje optimální řešení. Výsledná pravá strana, tj. vlastně souřadnice optimálního řešení budou samozřejmě také lineárně záviset na parametru t.

Výsledné optimální řešení ovšem musí být přípustné, tj. všechny hodnoty pravých stran musí být nezáporné. To vede na soustavu lineárních nerovností, jejímž řešením bude interval, ve kterém je dané bázické řešení přípustné a je tedy optimálním řešením úlohy s parametrem.

Poznamenejme, že závislost optimálního řešení na parametru zde není skoková, ale spojitá.

2.9.10 Příklad lineární změny pravých stran. V příkladě 2.6.6, str. 24 uvažujme změnu původního vektoru pravých stran $(2, 2, 5)$ na nový vektor $(2, 2+t, 5)$.

Připomeňme, že ve výchozí simplexové tabulce tvořily kanonickou bázi sloupce 3., 4. a 5. (v tomto pořadí) a simplexová tabulka s optimálním řešením byla

		2	3	0	0	0	*max*
0	3.	0	0	1	1	1	9
3	2.	0	1	0	$\frac{1}{3}$	$\frac{2}{3}$	4
2	1.	1	0	0	$-\frac{1}{3}$	$\frac{1}{3}$	1
		0	0	0	$\frac{1}{3}$	$2\frac{2}{3}$	14

Nová pravá strana v tabulce s optimálním řešením je

$$B^{-1} \cdot \begin{pmatrix} 2 \\ 2+t \\ 5 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 0 & \frac{1}{3} & \frac{2}{3} \\ 0 & -\frac{1}{3} & \frac{1}{3} \end{pmatrix} \cdot \begin{pmatrix} 2 \\ 2+t \\ 5 \end{pmatrix} = \begin{pmatrix} 9+t \\ 4+\frac{1}{3}t \\ 1-\frac{1}{3}t \end{pmatrix}$$

Optimální řešení závislé na parametru t tedy bude $(1-\frac{1}{3}t, 4+\frac{1}{3}t, 9+t, 0, 0)$, toto však platí pouze pro takové hodnoty parametru, při kterých jsou splněny nerovnosti

$$\begin{aligned} 9+t &\geqq 0 \\ 4+\tfrac{1}{3}t &\geqq 0 \\ 1-\tfrac{1}{3}t &\geqq 0 \end{aligned}$$

tedy v rozsahu $-9 \leqq t \leqq 3$. Účelová funkce také závisí na parametru t a to tak, že je rovna $2 \cdot (1-\frac{1}{3}t) + 3 \cdot (4+\frac{1}{3}t) = 14+\frac{1}{3}t$.

Pokud parametr t bude mimo výše uvedený interval $\langle -9, 3\rangle$, bude mít optimální řešení jinou bázi (pokud bude existovat).

2.9.11 Přidání nové omezující podmínky je klíčovou součástí některých výpočetních metod, např. metody větví a mezí 4.3, metody sečných nadrovin nebo metody generování omezujících podmínek 2.9.13. Samozřejmě lze zde popsaný postup použít i v situaci, kdy jsme na nějakou podstatnou omezující podmínku zapomněli.

Máme-li vypočteno optimální řešení a přidáme-li k úloze omezující podmínku, jsou dvě možnosti. Buď stávající optimální řešení novou podmínku splňuje nebo nikoli. Pokud splňuje, je vlastně přidání takové podmínky zbytečné: úloha má stejné optimální řešení, ať už podmínku přidáme nebo nikoli.

Dále se tedy zabývejme případem, kdy optimální řešení novou omezující podmínku nesplňuje a budeme předpokládat, že známe nejen optimální řešení, ale i příslušnou simplexovou tabulku, která toto řešení reprezentuje.

Výpočet při přidání nové omezující podmínky spočívá ve třech krocích:

- přidání nového řádku a sloupce do simplexové tabulky,
- úprava tabulky s cílem získat opět kanonickou bázi a
- nalezení přípustného řešení.

Nyní tyto tři kroky popíšeme podrobněji:

1. Do simplexové tabulky přidáme řádek a sloupec. Provedení se liší podle typu přidávané podmínky:

   Podmínku typu $\leqq$ převedeme na rovnici zavedením nové doplňkové proměnné. V simplexové tabulce jednoduše přidáme řádek a sloupec. Sloupec nové doplňkové proměnné bude členem báze.

Podmínku typu $=$ přidáme jako nový řádek a pro tento řádek přidáme sloupec nové umělé proměnné (s cenovým koeficientem $-M$). Sloupec umělé proměnné bude členem báze.

Podmínku typu $\geqq$ vynásobíme koeficientem -1, čímž ji převedeme na podmínku typu $\leqq$, tedy na případ popsaný výše. (Záporná pravá strana nám nevadí.)

2. Obnovíme kanonickou bázi. Dosavadní bázické sloupce mohou v přidané řádce obsahovat (a typicky obsahují) nenulové koeficienty. Proto k nové řádce přičteme vhodné násobky ostatních řádků tak, aby byla kanonická báze obnovena. Poznamenejme, že dosavadní bázické sloupce v bázi zůstávají (proto je nutné jejich tvar obnovit) a k nim přibývá nově přidaný sloupec (se kterým není nutno nic dělat).

3. Vypočteme relativní ceny.

4. Je-li ve výsledné tabulce záporná některá pravá strana (typicky je), použijeme duálně simplexovou metodu 2.9.7, str. 39.

2.9.12 Příklad — přidání omezující podmínky. K úloze 2.2.2, str. 12 vyřešené v 2.6.6, str. 24 přidáme novou omezující podmínku $x_1 + 2x_2 \leqq 7$.

Simplexová tabulka, reprezentující optimální řešení je

		2	3	0	0	0	max
0	3.	0	0	1	1	1	9
3	2.	0	1	0	$\frac{1}{3}$	$\frac{2}{3}$	4
2	1.	1	0	0	$-\frac{1}{3}$	$\frac{1}{3}$	1
		0	0	0	$\frac{1}{3}$	$2\frac{2}{3}$	14

Ve shodě s 2.9.11 přidáme řádek nové podmínky a jednu doplňkovou proměnnou. Dostaneme tabulku

		2	3	0	0	0	0	max
0	3.	0	0	1	1	1	0	9
3	2.	0	1	0	$\frac{1}{3}$	$\frac{2}{3}$	0	4
2	1.	1	0	0	$-\frac{1}{3}$	$\frac{1}{3}$	0	1
0	6.	1	2	0	0	0	1	7

Tato tabulka neobsahuje kanonickou bázi – její prvé dva sloupce jsou poškozeny novým řádkem. Abychom obnovili kanonickou bázi, od nového řádku odečteme třetí řádek a dvojnásobek druhého řádku. Tím dostaneme tabulku, která již obsahuje kanonickou bázi:

		2	3	0	0	0	0	max
0	3.	0	0	1	1	1	0	9
3	2.	0	1	0	$\frac{1}{3}$	$\frac{2}{3}$	0	4
2	1.	1	0	0	$-\frac{1}{3}$	$\frac{1}{3}$	0	1
0	6.	0	0	0	$-\frac{1}{3}$	$-1\frac{2}{3}$	1	-2
		0	0	0	$\frac{1}{3}$	$2\frac{2}{3}$	0	14

Řešení reprezentované touto tabulkou není přípustné. Pokračujeme tedy duálně simplexovou metodou 2.9.7, str. 39.

		2	3	0	0	0	0	max
0	3.	0	0	1	0	-4	3	3
3	2.	0	1	0	0	-1	1	2
2	1.	1	0	0	0	2	-1	3
0	4.	0	0	0	1	5	-3	6
		0	0	0	0	1	1	12

a výsledná tabulka již obsahuje optimální řešení změněné úlohy $(3, 2, 3, 6, 0, 0)$.

2.9.13 Generování omezujících podmínek. Je-li všech omezujících podmínek příliš mnoho, můžeme postupovat takto:

Předpokládáme, že mezi omezujícími podmínkami úlohy je množina podmínek (označme ji Q) taková, že podmínek v množině Q je mnoho, ale pro jakékoli konkrétní řešení je relativně snadné zkontrolovat platnost všech omezujících podmínek z množiny Q.

1. Úlohu řešíme bez omezujících podmínek z množiny Q.
2. Zkontrolujeme, zda nalezené optimální řešení x splňuje všechny podmínky z množiny Q. Pokud splňuje, výpočet končí, řešení x je optimálním řešením i úlohy se všemi podmínkami z množiny Q.
3. Z množiny Q vybereme některou omezující podmínku, která není stávajícím optimálním řešením splněna, a tuto podmínku k úloze přidáme.
4. Vypočteme optimální řešení rozšířené úlohy a pokračujeme krokem 2.

Často stačí přidat jen nepatrný zlomek z celkového počtu všech podmínek z množiny Q. To může představovat velikou úsporu námahy při výpočtu. Samozřejmě při tom velmi záleží na způsobu výběru nesplněné omezující podmínky v kroku 3.

2.10 Dualita

2.10.1 Standardní tvar úlohy LP. Úloha lineárního programování je ve standardním tvaru, jestliže

- účelová funkce se maximalizuje,
- na všechny proměnné se klade podmínka nezápornosti a
- všechny strukturní podmínky jsou typu "$\leqq$".

Jinými slovy, úloha má tvar

$$\begin{aligned} & c^T x \to \max \\ & Ax \leqq b \\ & x \geqq 0 \end{aligned} \quad .$$

Každou úlohu LP lze převést na standardní tvar.

2.10.2 Symetrické duální úlohy je dvojice úloh LP tohoto tvaru:

$$\begin{array}{lll} P: & \begin{aligned} & c^T x \to \max \\ & Ax \leqq b \\ & x \geqq 0 \end{aligned} & \qquad D: \qquad \begin{aligned} & b^T y \to \min \\ & A^{\mathrm{T}} y \geqq c \\ & y \geqq 0 \end{aligned} \end{array}$$

Tedy s nadsázkou „obrať co můžeš". Ale pozor, vektory x a y typicky mají různé počty souřadnic. Složky vektoru y odpovídají (počtem i významem) strukturním podmínkám úlohy P a naopak složky vektoru x odpovídají strukturním podmínkám úlohy D.

Symetrii je třeba chápat takto: K úloze A, která je ve standardním tvaru, sestrojíme duální úlohu B. Úlohu B převedeme na standardní tvar, dostaneme úlohu C. K úloze C sestrojíme duální úlohu D. Když pak úlohu D převedeme na standardní tvar, dostaneme původní úlohu A.

2.10.3 Příklad — problém výživy a výrobce pilulek. Problém výživy spočívá v určení, kolik kterého jídla máme sníst, abychom se stravovali co nejlevněji. Již bylo zmíněno v 2.1.6, str. 11, že jde o speciální případ úlohy o směsi. Předpokládejme následující označení:

x_j	kolik jídla j máme sníst za rok,
c_j	cena jednotkového množství jídla j,
b_i	roční potřeba živiny i,
a_{ij}	množství živiny i v jednotkovém množství jídla j.

Problém výživy pak má tvar

$$\begin{aligned} c^{\mathrm{T}}x &\to \min \\ Ax &\geqq b \\ x &\geqq 0\,, \end{aligned}$$

jde tedy o úlohu, která svou strukturou je shodná s úlohou D z 2.10.2, jen vektor proměnných y je zde označen x.

Představme si nyní poněkud sci-fi situaci, kdy existuje výrobce čistých živin v podobě pilulek a tento výrobce chce pilulky prodávat a cenově konkurovat přirozené stravě. Předpokládejme dále, že spotřebitelé jsou již tak degenerovaní, že se řídí pouze cenou, nikoli tím, co jim chutná. Přirozenou snahou výrobce pilulek je maximalizace tržeb za pilulky, ale nesmí to s cenami pilulek přehnat. Aby pilulky byly konkurenceschopné, je nutné, aby žádné jídlo nebylo levnější, než jeho „pilulkový ekvivalent“. Výrobce také nechce žádné pilulky dotovat, tj. platit lidem za to, že je jí. Špatně by se mu to kontrolovalo.

Označíme-li y_i prodejní cenu jednotkového množství pilulek obsahujících živinu i, lze problém výrobce pilulek formulovat jako

$$\begin{aligned} b^{\mathrm{T}}y &\to \max \\ A^{\mathrm{T}}y &\leqq c \\ y &\geqq 0\,, \end{aligned}$$

což je tvar úlohy P z 2.10.2, kde opět máme y namísto x.

Cíle strávníků a výrobce pilulek jsou protichůdné. Strávníci se snaží utratit co nejméně, zatímco výrobce pilulek se snaží, aby mu strávníci zaplatili co nejvíc. Matematický vztah duality mezi oběma úlohami je matematickým obrazem tohoto antagonismu. Podle věty o dualitě 2.10.9 Má-li jedna z úloh optimální řešení, má je i druhá a obě přitom dosáhnou stejné hodnoty účelové funkce.

2.10.4 Duální úloha k obecné úloze LP. Vztah duality lze rozšířit na obecné úlohy LP. Počet proměnných v jedné úloze je roven počtu strukturních podmínek ve druhé úloze. Matice strukturních podmínek se transponuje, cenové koeficienty a koeficienty pravých stran si navzájem vymění roli. Zbývá určit orientaci nerovností u strukturních podmínek a omezující podmínky pro jednotlivé proměnné. Ty jsou vzájemně provázány podle této tabulky

P	D
$c^{\mathrm{T}}x \to \max$	$b^{\mathrm{T}}y \to \min$
$A_i x \leqq b_i$	$y_i \geqq 0$
$A_i x \geqq b_i$	$y_i \leqq 0$
$A_i x = b_i$	y_i neomezeno
$x_j \geqq 0$	$a_j^{\mathrm{T}} y \geqq c_j$
$x_j \leqq 0$	$a_j^{\mathrm{T}} y \leqq c_j$
x_j neomezeno	$a_j^{\mathrm{T}} y = c_j$

kde značení A, c, b, x se vztahuje k úloze P a y je vektor proměnných úlohy D. Symbolem A_i značíme i-tý řádek a symbolem a_j značíme j-tý sloupec matice A.

Pozor, všimněte si, že ne všechny nerovnosti se obrací.

Tabulku si není třeba pamatovat. Která nerovnost se obrací lze snadno odvodit z lehce zapamatovatelného vztahu mezi symetrickými duálními úlohami. Protějškem strukturní podmínky typu rovnice je neomezená proměnná v protější úloze.

2.10.5 Duální úloha k duální úloze je ekvivalentní s původní úlohou. Ověřte to na příkladě.

2.10.6 Věta. Je-li x přípustné v úloze P a y přípustné v úloze D, pak platí $c^{\mathrm{T}}x \leqq b^{\mathrm{T}}y$.

Důkaz: Platí

$$c^{\mathrm{T}}x \leqq (A^{\mathrm{T}}y)^{\mathrm{T}}x = y^{\mathrm{T}}Ax \leqq y^{\mathrm{T}}b = b^{\mathrm{T}}y \quad .$$

Zde prvá nerovnost vyplývá z podmínky $A^{\mathrm{T}}y \geqq c$, druhá z podmínky $Ax \leqq b$. □

2.10.7 Důsledky.

1. Má-li úloha P shora neomezenou účelovou funkci, pak D nemá přípustné řešení.
2. Má-li úloha D zdola neomezenou účelovou funkci, pak P nemá přípustné řešení.

2.10.8 Věta. Jsou-li x a y přípustná řešení (každé ve své úloze) a platí-li $c^Tx = b^Ty$, pak x a y jsou optimálními řešeními (každé ve své úloze).

Tato věta je triviálním, ale pozoruhodným, důsledkem tvrzení 2.10.6, neboť platí nezávisle na způsobu, jak jsme získali řešení x a y, která splňují předpoklady. Mohli jsme je i uhodnout.

2.10.9 Věta o dualitě. Má-li úloha P optimální řešení, označme je x, pak i k ní duální úloha D má optimální řešení (označme je y) a obě optimální řešení mají stejnou hodnotu účelové funkce, tedy platí $c^Tx = b^Ty$.

Důkaz: vyplyne z následujícího:

2.10.10 Jak najít duální řešení v simplexové tabulce primární úlohy. Vyjděme z úlohy P ve standardním tvaru. Výchozí simplexová tabulka má tvar:

		c^T	0	max
0		A	E	b
		$-c^T$	0	0

Výsledná simplexová tabulka s optimálním řešením pak má tvar:

		c^T	0	max
c_B		$B^{-1}A$	B^{-1}	$B^{-1}b$
		$c_B^TB^{-1}A - c^T$	$c_B^TB^{-1}$	$c_B^TB^{-1}b$

Tvrzení: vektor $y = (c_B^TB^{-1})^T$ je optimálním řešením duální úlohy.

A vskutku. Řešení y je přípustným řešením, neboť ve výsledné tabulce jsou všechny relativní ceny nezáporné. A účelová funkce duálního řešení $L(y) = y^{\mathrm{T}}b = c_B^TB^{-1}b$ je rovna účelové funkci $L(x)$. Tedy podle 2.10.8 je vektor y optimálním řešením duální úlohy.

2.10.11 Věta o rovnováze. Nechť x a y jsou přípustná řešení (každé ve své úloze). Pak x a y jsou optimální řešení (každé ve své úloze) právě tehdy když

$$\left[\begin{array}{lll} \forall i & A_i x < b_i & \Rightarrow \quad y_i = 0 \\ \forall j & a_j^T y > c_j & \Rightarrow \quad x_j = 0 \end{array}\right]$$

Poznámka: Na základě věty o rovnováze bylo vymyšleno několik efektivních algoritmů pro řešení úloh založených na lineárním programování. Tyto algoritmy hledají současně řešení obou úloh, primární i duální, a to tak, že se různými triky snaží splnit podmínky věty o rovnováze. Jakmile toho dosáhnou, podle věty o rovnováze máme optimální řešení obou úloh. Příkladem takového algoritmu je metoda MODI pro řešení dopravní úlohy, viz 3.3, str. 51.

Důkaz: Platí

$$\left[\begin{array}{lll} \forall i & A_i x < b_i & \Rightarrow \quad y_i = 0 \\ \forall j & a_j^T y > c_j & \Rightarrow \quad x_j = 0 \end{array}\right] \Longleftrightarrow \left[\begin{array}{ll} \forall i & y_i(A_i x - b_i) = 0 \\ \forall j & x_j(a_j^T y - c_j) = 0 \end{array}\right]$$

□

2.10.12 Výpočty s pomocí duality. Některé úlohy (kde by bylo mnoho umělých proměnných) může být výhodnější řešit oklikou přes duální úlohu takto: Danou úlohu označme A. Zformulujeme k ní duální úlohu B, vyřešíme ji (najdeme optimum) a v poslední simplexové tabulce najedeme optimální řešení nejen úlohy B, ale i optimální řešení úlohy C, která je duální k B a tudíž je ekvivalentní k úloze A.

2.10.13 Duální ceny. Složky y_i optimálního řešení duální úlohy jsou často nazývány *duálními cenami*. Název je inspirován faktem, že jde vlastně o ocenění pravých stran strukturních omezujících podmínek v následujícím smyslu: když (v primární úloze) zvýšíme pravou stranu b_i o malou hodnotu Δ, zvýší se účelová funkce o hodnotu $\Delta \cdot y_i$.

Například v klasické aplikaci LP na plánování výroby při omezených zdrojích duální cena y_i vyjadřuje, jak dovedeme omezující i-tý zdroj přeměnit na zisk. To je velmi důležitý údaj. Chceme-li zvýšit zisk, vidíme hned, za jaké ceny má nebo nemá smysl omezující zdroje nakupovat.

Podrobnější výklad týkající se změn pravých stran, např. jak velké smí být to Δ, aby duální cena y_i měla popsaný význam, viz 2.9.6.

Poznamenejme, že omezující zdroj, který nejsme schopni plně využít (jeho doplňková proměnná je kladná), má duální cenu nulovou. Ostatně to tvrdí i věta o rovnováze.

Všimněte si, že i v příkladě 2.10.3 ceny pilulek odpovídají tomu jak se mění hodnota účelové funkce problému výživy. Kdyby potřeba živiny i klesla o Δ, hodnota účelové funkce by klesla o $y_i\Delta$.

A také naopak, ceny jídel jsou duálními cenami pro problém výrobce pilulek. Kdyby se snížila cena jídla j o hodnotu Δ, výrobce pilulek by reagoval změnou cen pilulek a jeho maximální možná tržba by se snížila o $x_j\Delta$.

Kapitola 3

Dopravní úloha

Dopravní úlohu jsme definovali již v 2.1.7, str. 11 a ukázali jsme, že jde o speciální případ úlohy lineárního programování. V této kapitole (kromě aplikací a několika základních fakt) uvedeme podstatně výhodnější algoritmy.

3.1 Základní fakta a aplikace

Pro pohodlí čtenáře zopakujme definici dopravní úlohy.

3.1.1 Dopravní úloha. Máme m dodavatelů, kteří dodávají stejný druh zboží a n spotřebitelů, kteří je spotřebovávají. Známe ceny za dopravu mezi všemi dodavateli a všemi spotřebiteli. Úkolem je zorganizovat dopravu co nejlevněji. Označme

a_i	kapacita i-tého dodavatele (kolik je schopen dodat),
b_j	požadavek j-tého spotřebitele (kolik chce spotřebovat),
c_{ij}	cena za dopravu jednotkového množství od i-tého dodavatele k j-tému spotřebiteli,
x_{ij}	množství dopravované od i-tého dodavatele k j-tému spotřebiteli.

Pak úloha LP pak má tvar

$$\begin{aligned} \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} &\to \min \\ \sum_{j=1}^{n} x_{ij} &= a_i \\ \sum_{i=1}^{m} x_{ij} &= b_j \\ x_{ij} &\geqq 0 \qquad \text{pro všechna } i,j \end{aligned}$$

Všimněte si, že zde máme mn proměnných, které jsme pro pohodlí indexovali dvěma indexy, lze se tedy na ně dívat jako na matici. Přesto lze proměnné seřadit do jedné posloupnosti (např. po řádcích) a celou úlohu přepsat do obvyklého tvaru úlohy lineárního programování.

Fakt, že dopravní úloha je speciálním případem úlohy lineárního programování nepoužijeme přímo k výpočtu. Pomocí dulaity odvodíme výhodnější algoritmus.

Zadání dopravní úlohy i její řešení budeme zapisovat do tabulky typu „dodavatelé × spotřebitelé“. Pro stručnost budeme tuto tabulku nazývat *dopravní tabulkou*.

	10	20	20	10
20	6 **10**	3	7	8 **10**
20	2	7	1 **20**	8
20	3	1 **20**	4	2

Řádky odpovídají dodavatelům, sloupce spotřebitelům. Vlevo před každým řádkem je uvedena kapacita dodavatele, nad každým sloupcem je požadavek spotřebitele. Ceny za přepravu jsou uvedeny v každém políčku tabulky drobným písmem v pravém horním rohu políčka. Vlevo dole jsou normálním písmem uvedeny nenulové velikosti přepravy x_{ij}. Není-li x_{ij} uvedeno, znamená to, že $x_{ij} = 0$.

Poznamenejme, že v dopravní tabulce máme dvě matice typu $m \times n$. Matice cenových koeficientů c je součástí zadání a během výpočtu se nemění. Druhou maticí je matice x, která je výsledkem výpočtu a během výpočtu se mění. Pro ruční výpočet je výhodné mít obě matice pohromadě v téže tabulce. Je třeba upozornit, že někteří autoři, snad kvůli typografickým obtížím, zapisují obě matice zvlášť.

Dále poznamenejme, že dopravní tabulka má podstatně menší rozměry (a menší paměťové nároky v počítači) než simplexová tabulka pro tutéž dopravní úlohu.

3.1.2 Vyvážená dopravní úloha je taková, kde nabídka dodavatelů je v součtu rovna poptávce spotřebitelů, tj. kde $\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j$. Poznamenejme, že tato rovnost není součástí omezujících podmínek úlohy, to je vlastnost instance dopravní úlohy.

Dopravní úloha, tak jak byla definována výše, má přípustné řešení pouze je-li vyvážená.

3.1.3 Nevyvážená dopravní úloha. V praxi se často setkáváme s úlohami, které vyvážené nejsou. V takovém případě je nutno slevit z omezujících podmínek, nahradit rovnice nerovnostmi a smířit se s tím, že nějaké zboží nebude dodáno nebo některá poptávka nebude uspokojena. Nevyvážená úloha se řeší pomocí převodu na úlohu vyváženou a to tak, že k úloze přidáme fiktivního dodavatele nebo fiktivního spotřebitele.

Je-li nabídka dodavatelů menší než poptávka, tj. $\sum_{i=1}^{m} a_i < \sum_{j=1}^{n} b_j$, přidáme formálně jednoho tzv. *fiktivního dodavatele*, který chybějící zboží (fiktivně) dodá. Kapacita fiktivního dodavatele se stanovuje tak, aby se úloha stala vyváženou, tj. jako $\sum_{j=1}^{n} b_j - \sum_{i=1}^{m} a_i$. Proto vždy stačí přidávat jen jednoho fiktivního dodavatele. V dopravní tabulce tedy přibude jeden řádek. Je samozřejmé, že spotřebitel, který dostane fiktivní zboží, má smůlu, neboť de facto nedostane nic nebo jen část toho, co chtěl.

Ceny za dopravu fiktivního zboží od fiktivního dodavatele ke všem spotřebitelům se obvykle volí nulové. Pokud tyto ceny budou třeba i nenulové, ale pro všechny spotřebitele stejné, pak těmito cenami žádný spotřebitel není zvýhodněn ani znevýhodněn. Kteří spotřebitelé dostanou fiktivní zboží, záleží na výsledku celkové optimalizace, ale zjednodušeně lze říci, že vzdálení spotřebitelé jsou v nevýhodě. Pokud to vadí, je třeba použít priority, viz 3.1.5

Podobně se řeší případ, kdy nabídka převyšuje poptávku, tj. $\sum_{i=1}^{m} a_i > \sum_{j=1}^{n} b_j$. V tomto případě přidáváme fiktivního spotřebitele, který fiktivně odebere zboží, které na straně dodavatelů přebývá. V dopravní tabulce přibude jeden sloupec. Dodavatel, který pak má dodávat fiktivní zboží fiktivnímu spotřebiteli, má smůlu, toto zboží mu de facto zůstane.

3.1.4 Prohibitivní ceny. Někdy je třeba zajistit, aby některý dodavatel i nemohl dodávat některému spotřebiteli j. Může to být např. proto, že zboží od dodavatele i nesplňuje nároky na kvalitu ze strany spotřebitele j.

Tento požadavek lze v dopravní úloze snadno zajistit tím, že cenu za dopravu c_{ij} stanovíme velmi vysokou. Bude-li cena dostatečně vysoká, pak, pokud to je vůbec možné, bude doprava realizována jinudy. Vysoká cena tedy má zakazující (prohibitivní) účinek.

Podobný trik (použití velmi nevýhodné ceny) jsme použili v tzv. M-úloze lineárního programování, viz 2.7.2. Onu velmi vysokou prohibitivní cenu budeme v dalším textu značit M.

3.1.5 Priority v nevyvážené úloze. Někdy se v nevyvážené úloze chce, aby požadavek nějakých privilegovaných spotřebitelů byl uspokojen přednostně, tj. aby tito spotřebitelé zaručeně dostali skutečné zboží (pokud to je možné) a aby fiktivní zboží dostal někdo jiný, neprivilegovaný. Takový požadavek lze v dopravní úloze snadno zařídit pomocí prohibitivních cen za dopravu fiktivního zboží od fiktivního dodavatele k privilegovaným spotřebitelům.

Pokud by součet kapacit dodavatelů nestačil pro všechny privilegované spotřebitele, a chceme-li i v takové situaci mít kontrolu nad tím, kdo dostane zboží skutečné a kdo fiktivní, lze to řešit zavedením dvou nebo i více stupňů priorit. Prohibitivní ceny by pak byly M, $2M$, $3M$, atd.

3.1.6 Přiřazovací úloha je definována takto: Máme n pracovníků a n pracovních úkolů. Pro každou dvojici pracovník–úkol známe náklady c_{ij} na provedení j-tého úkolu i-tým pracovníkem. Máme přiřadit každému pracovníkovi přesně jeden úkol a to tak, aby součet nákladů byl nejnižší.

Přiřazovací úlohu lze snadno převést na úlohu dopravní. Pracovníky budeme pokládat za dodavatele, pracovní úkoly za spotřebitele. Kapacity všech dodavatelů i spotřebitelů budou jednotkové. Ceny za dopravu budou rovny nákladům c_{ij}.

Výsledek dopravní úlohy budeme interpretovat tak, že když vyjde $x_{ij} = 1$, přiřadíme i-tému pracovníkovi j-tý pracovní úkol.

3.2 Heuristiky pro dopravní úlohu

Cílem heuristických algoritmů je najít přípustné řešení, pokud možno co nejlepší, ale aby to nedalo příliš mnoho práce.

Výsledek heuristických algoritmů má dvojí použití: Pokud nebylo požadováno nalezení přesného optima, a je-li hodnota účelové funkce přijatelná, lze výsledek heuristiky přímo prakticky aplikovat. Druhé použití spočívá v tom, že výsledek heuristiky může sloužit jako základ pro další zlepšování.

Připomeňme, že se zabýváme pouze vyváženými úlohami.

3.2.1 Obecné schéma heuristik pro dopravní úlohu je toto

1. Nějakým způsobem zvolíme dvojici i, j, kde kapacita dodavatele i není zcela vyčerpána a zároveň požadavek spotřebitele j není zcela uspokojen.
2. Hodnotu x_{ij} zvolíme největší přípustnou vzhledem ke dřívějším volbám.
3. Kroky 1 a 2 opakujeme, dokud zbývá nějaká kapacita dodavatelů.

Jednotlivé heuristické algoritmy se liší pouze v metodě výběru dvojice i, j v kroku 1. Krok 2 je všem metodám společný.

Každým provedením kroku 2 buď zcela vyčerpáme zbývající kapacitu dodavatele i nebo zcela nasytíme spotřebitele j, případně obojí. Takto vyčerpaný dodavatel nebo spotřebitel pak již nemůže být vybrán v kroku 1. Možnosti volby v kroku 1 se tedy stále zmenšují. Podobně se zmenšují kapacity dodavatelů a požadavky spotřebitelů, kteří dosud nebyli vyloučeni.

Je zřejmé, že po nejvýše $m+n-1$ opakováních výpočet skončí. Při posledním provedení kroku 1 totiž zbývá jen jeden dodavatel a jen jeden spotřebitel a díky vyváženosti úlohy budou v kroku 2 oba vyčerpáni současně. Dalším důsledkem je, že výsledné řešení bude mít nejvýše $m+n-1$ nenulových složek. To se nám bude hodit při následné optimalizaci.

3.2.2 Metoda severozápadního rohu vybírá v kroku 1 vždy dvojici s nejmenším i a s nejmenším j, tedy prvek, který je v tabulce v levém horním rohu, tedy tam, kde je na mapě severozápad. Odtud velmi populární název metody.

Tuto metodu zde uvádíme spíše z terminologické povinnosti, neboť je uváděna ve všech učebnicích. Z hlediska kvality výsledku **je to ta nejhloupější metoda**, kterou si lze představit. Nebere totiž vůbec v úvahu cenové koeficienty, proto výsledkem může být zrovna tak řešení nejlepší jako nejhorší.

3.2.3 Indexová metoda je asi nejjednodušší prakticky použitelnou heuristikou pro dopravní úlohu. Ze všech dosud nevyřazených dodavatelů a spotřebitelů v indexové metodě vždy vybíráme takovou dvojici, která má nejnižší cenu c_{ij}.

Příklad:

	10	20	20	10
20	6 **10**	3	7	8 **10**
20	2	7	1 **20**	8
20	3	1 **20**	4	2

Výpočet indexovou metodou zde probíhal takto:

- Vybráno $c_{32} = 1$, tedy $x_{32} = 20$, vyčerpán 3. dodavatel i 2. spotřebitel.
- Vybráno $c_{23} = 1$, tedy $x_{23} = 20$, vyčerpán 2. dodavatel i 3. spotřebitel.
- Vybráno $c_{11} = 6$, tedy $x_{11} = 10$, vyčerpán 1. spotřebitel.
- Vybráno $c_{14} = 8$, tedy $x_{14} = 10$, vyčerpán 1. dodavatel a 4. spotřebitel.

Kvalita řešení získaného indexovou metodou zpravidla není špatná, může se však poměrně snadno stát že výsledek není nejlepší, může být dokonce i nejhorší možný. Zde je příklad:

	10	10
10	4 **10**	5
10	5	1000 **10**

Zde jsme nejprve chamtivě vybrali $c_{11} = 4$ a pak nezbyla jiná možnost než využít velmi drahou trasu $(2, 2)$ s cenou $c_{22} = 1000$. Snadno ověříte, že horší řešení neexistuje.

3.2.4 Vogelova metoda se snaží překonat výše zmíněný nedostek indexové metody.

Pro každý řádek vypočteme rozdíl mezi nejmenší a druhou nejmenší cenou v rámci řádky. Vybereme řádek s největším rozdílem a v tomto řádku vybereme prvek s nejmenší cenou.

Použijeme-li Vogelovu metodu na příklad uvedený ve 3.2.3, dostaneme pro jednotlivé řádky tyto rozdíly mezi nejmenší a druhou nejmenší cenou: 3, 1, 1. Největší rozdíl je v prvém řádku, nejlevnější cena v prvém řádku je v prvku $(1, 2)$, vybereme tedy tento prvek a stanovíme $x_{12} = 20$. Tím je vyčerpán první dodavatel a současně i druhý spotřebitel.

Prvý řádek a druhý sloupec vypadly ze hry, což má vliv na rozdíly mezi nejnižší a druhou nejnižší cenou ve zbývajících řádcích. V našem případě ovšem oba rozdíly náhodou zůstaly rovny jedné. Vybereme si tedy celkově nižší cenu $c_{23} = 1$ a stanovíme $x_{23} = 20$.

Tím ze hry vypadl druhý řádek a třetí sloupec. Ve hře tedy zůstávají už jen prvky $(3, 1)$ a $(3, 4)$, kde už není z čeho vybírat a nezbývá než zvolit $x_{31} = 10$ a $x_{34} = 10$. Jak uvidíme dále v 3.3.16, dostali jsme jako výsledek optimální řešení, ale pozor, to jsme jen měli štěstí, byla to náhoda.

Je třeba zdůraznit, že **Vogelova metoda je pouze heuristikou**, je běžné, že řešení které touto metodou získáme, není optimální. Ve srovnání s indexovou metodou je Vogelova metoda pracnější, ale dává zpravidla lepší výsledky.

Dále poznamenejme, že alternativně lze Vogelovu metodu dělat se sloupci namísto s řádkami, popřípadě s řádkami i se sloupci najednou. Existuje také mnoho variant, jak si počínat v nerozhodných případech.

3.2.5 Transformace cen. Když v nějakém řádku i dopravní tabulky změníme (např. snížíme) všechny ceny c_{ij} o stejnou hodnotu Δ, dostaneme novou dopravní úlohu, která bude mít stejné optimální řešení (se stejnými x_{ij}), jako původní úloha.

Proč? Obě úlohy mají stejnou množinu přípustných řešení (protože omezující podmínky zůstaly nezměněny) a pro všechna přípustná řešení platí, že hodnota účelové funkce se změní (např. zmenší) o stejnou hodnotu $a_i \cdot \Delta$. Tedy řešení, které bylo optimální před změnou cen, bude optimální i po změně (a naopak), jen bude mít jinou hodnoru účelové funkce.

Totéž platí i o změně cen v libovolném sloupci a tyto transformace cen v řádcích i ve sloupcích lze libovolně kombinovat. Pokud transformací dostaneme některé ceny záporné, vůbec to nevadí, výsledná úloha totiž také má stejné optimální řešení (tj. stejná přepravovaná množství z_{ij}) jako úloha původní.

Některé heuristiky využívají transformaci cen jako svůj první krok. Příkladem je tzv. *frekvenční metoda*.

3.2.6 Frekvenční metoda nejprve provede transformaci cen a to tak, že od každého cenového koeficientu odečte průměr cen v řádku a průměr cen ve sloupci. Na výslednou transformovanou úlohu se pak použije indexová metoda.

3.3 Metoda MODI

Trochu divný název metody má připomínat, že jde vlastně o modifikovanou simplexovou metodu.

3.3.1 Primární a duální úloha. Připomeňme tvar primární úlohy:

$$\begin{aligned}
\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} &\to \min \\
\sum_{j=1}^{n} x_{ij} &= a_i \\
\sum_{i=1}^{m} x_{ij} &= b_j \\
x_{ij} &\geqq 0 \qquad \text{pro všechna } i,j
\end{aligned}$$

Například pro 3 dodavatele a 4 spotřebitele vypadá výchozí simplexová tabulka takto:

1	1	1	1									a_1
				1	1	1	1					a_2
								1	1	1	1	a_3
1				1				1				b_1
	1				1				1			b_2
		1				1				1		b_3
			1				1				1	b_4

Soustava rovnic je závislá, jednu z rovnic bychom tedy mohli vynechat. Z důvodů úhlednější teorie ovšem nebudeme nic vynechávat.

V duální úloze rozlišíme proměnné u_i, které se týkají dodavatelů a proměnné v_j, které se týkají spotřebitelů. Duální úloha pak má tvar

$$\begin{aligned} \sum_{i=1}^{m} a_i u_i + \sum_{j=1}^{n} b_j v_j \quad &\to \quad \max \\ u_i + v_j \quad &\leqq \quad c_{ij} \quad \text{pro všechna } i,j \\ u_i, v_j \quad &\quad \text{neomezeny} \end{aligned}$$

1	1	c_{11}
1	1	c_{12}
1	1	c_{13}
1	1	c_{14}
1	1	c_{21}
1	1	c_{22}
1	1	c_{23}
1	1	c_{24}
1	1	c_{31}
1	1	c_{32}
1	1	c_{33}
1	1	c_{34}

Duální úloha může mít tuto interpretaci: Přepravce nabízí, že zboží za úplatu dopraví a chce maximalizovat své tržby z toho plynoucí. Cenu za přepravu jednotkového množství zboží chce stanovit jako součet ceny u_i, kterou bude chtít od dodavatele a ceny v_j, kterou bude chtít od spotřebitele. Tyto ceny mohou být i záporné (tj. přepravce je ochoten i připlácet), ale musí dodržet podmínku $u_i + v_j \leqq c_{ij}$, jinak by si to dodavatelé a spotřebitelé dopravili sami levněji, totiž za cenu c_{ij}.

Přeformulováním věty o rovnováze 2.10.11, str. 45 podle právě zavedeného značení dostaneme toto tvrzení:

3.3.2 Kritérium optimality. Řešení x a u, v jsou optimálními řešeními (každé ve své úloze) právě tehdy, když splňují tyto podmínky:

1. x je přípustným řešením primární úlohy,
2. u, v je přípustným řešením duální úlohy, tj. $u_i + v_j \leqq c_{ij}$ pro všechna i, j,
3. pro všechna i, j platí $\quad x_{ij} > 0 \;\Rightarrow\; u_i + v_j = c_{ij}$.

3.3.3 Důkaz kritéria optimality bez použití duality. Předpokládejme, že x a u, v splňují kritérium optimality 3.3.2. Hodnoty u, v použijme k transformaci cen podle 3.2.5, tj. každý cenový koeficient nahraďme cenou $c'_{ij} := c_{ij} - u_i - v_j$. Výsledná úloha má podle 3.2.5 stejné optimální řešení jako úloha původní. Dále, poněvadž u, v je přípustným řešením duální úlohy, platí $c'_{ij} := c_{ij} - u_i - v_j \geqq 0$. Navíc platí, že řešení x je nejen přípustné (podmínka 1), ale v transformované úloze má nulovou hodnotu účelové funkce (podmínka 3). Řešení x je tedy přípustné a zadarmo, přičemž žádné přípustné řešení nemůže mít účelovou funkci zápornou, neboť transformované ceny jsou všechny nezáporné. Řešení x je tedy optimálním řešením transformované úlohy, a tedy podle 3.2.5 je také optimálním řešením původní úlohy.

3.3.4 Upravené kritérium optimality. Výpočet optimálního řešení bude založen na kritériu optimality 3.3.2, str. 52. Abychom nemuseli dělat výjimky kvůli degenerovaným řešením, která mají menší počet nenulových hodnot x_{ij}, budeme podmínku 3 požadovat v poněkud silnější formě: Budeme požadovat, aby řešení x bylo bázické a platnost rovnic $u_i + v_j = c_{ij}$ budeme požadovat pro všechny dvojice (i, j), které jsou v bázi.

Jestliže

1. x je bázickým přípustným řešením (primární) dopravní úlohy,
2. u, v je přípustným řešením duální úlohy, tj. $u_i + v_j \leqq c_{ij}$ pro všechna i, j,
3. pro všechny dvojice i, j v bázi platí $u_i + v_j = c_{ij}$,

pak řešení x a u, v jsou optimálními řešeními (každé ve své úloze).

Abychom mohli toto kritérium použít pro výpočet, musíme vědět něco o tom, jak vypadají báze v dopravní tabulce.

3.3.5 Cesty a kružnice v dopravní tabulce. Posloupnost prvků matice typu $m \times n$ nazveme *cestou*, jestliže prvky, které jsou v této posloupnosti sousední, leží ve společném sloupci nebo ve společném řádku a jestliže navíc žádné tři po sobě jdoucí prvky posloupnosti neleží ve společném řádku ani sloupci.

Prakticky to znamená, že pokud prvý a druhý prvek posloupnosti leží ve společném řádku, pak druhý a třetí prvek leží ve společném sloupci, třetí a čtvrtý prvek zase ve společném řádku, atd.

Uzavřenou cestu, tj. cestu, která začíná a končí ve stejném prvku, budeme nazývat *kružnice*.

Pokud víte, jak se v šachu tahá figurkami, tak výše definovaný pojem cesty odpovídá tahu věží.

3.3.6 Závislost a nezávislost. Sloupce simplexové tabulky odpovídají prvkům dopravní tabulky. Díky velmi speciálnímu tvaru matice soustavy v simplexové tabulce (viz 3.3.1) platí, že množina prvků (i, j) dopravní tabulky je závislá právě tehdy, když obsahuje kružnici, tj. uzavřenou cestu.

3.3.7 Báze. Množina B prvků dopravní tabulky tvoří bázi, jsou-li splněny následující podmínky:

1. Souvislost: Pro každý řádek i a každý sloupec j tabulky existuje cesta tvořená (některými) prvky množiny B, která začíná v řádku i a končí ve soupci j.

2. Absence kružnic: Neexistuje kružnice (tj. uzavřená cesta) tvořená prvky množiny B. Jinak řečeno, množina B je nezávislá.

3. Počet prvků množiny B je roven $m + n - 1$.

Pomocí teorie grafů (viz 8.5.2) je dokázáno, že jsou-li splněny kterékoli dvě z těchto podmínek, je splněna i třetí. Při praktickém výpočtu se nejsnáze ověřuje souvislost a počet prvků.

Počet prvků $m + n - 1$ je nejmenší, aby množina B mohla být souvislá a zároveň je největší, aby množina B ještě nemusela obsahovat kružnici.

3.3.8 Bázické přípustné řešení. Přípustné řešení, které získáme některým heuristickým postupem popsaným v 3.2, je vždy takové, že množina prvků (i, j) s nenulovým $x_{ij} > 0$ zaručeně neobsahuje kružnici a počet těchto prvků zaručeně není větší než $m + n - 1$.

Je-li počet nenulových prvků přesně roven $m + n - 1$, pak tyto nenulové prvky tvoří bázi.

Je-li počet nenulových prvků menší než $m + n - 1$, pak tyto prvky bázi netvoří (je jich málo). Proto formálně do báze doplníme některé nulové prvky. Musíme je ovšem doplnit tak, aby skutečně vznikla báze, tedy souvislá množina bez kružnic a o správném počtu prvků.

Zde je příklad správně doplněné báze:

	10	20	20	10
20	6 **10**	3	7 **0**	8 **10**
20	2	7	1 **20**	8
20	3 **0**	1 **20**	4	2

A zde je odstrašující příklad, jak báze vypadat nemá:

	10	20	20	10
20	6 **10**	3	7	8 **10**
20	2	7	1 **20**	8
20	3 **0**	1 **20**	4	2 **0**

Počet prvků je sice správný, ale je zde kružnice tvořená rohovými prvky a navíc druhý řádek a třetí sloupec není propojen se zbytkem tabulky.

3.3.9 Metoda MODI.

1. Najdeme bázické přípustné řešení x.
2. Vypočteme duální proměnné u, v tak, aby pro všechny bázické prvky (i, j) platilo $u_i + v_j = c_{ij}$. Postup výpočtu viz 3.3.10.
3. Zkontrolujeme, zda pro všechny dvojice (i, j) platí $u_i + v_j \leqq c_{ij}$. Pokud ano, výpočet končí, x je podle 3.3.4 optimálním řešením dopravní úlohy a u, v je optimálním řešením duální úlohy. Pokud ne, pokračujeme ve výpočtu.
4. Vybereme některý prvek (i, j), pro který neplatí podmínka $u_i + v_j \leqq c_{ij}$, tj. platí opak: $u_i + v_j > c_{ij}$. Tento prvek zavedeme do báze, tj. provedeme změnu řešení x podle 3.3.12 a pokračujeme krokem 2.

3.3.10 Výpočet duálních proměnných u_i a v_j spočívá v řešení soustavy rovnic. Pro každý prvek báze (i, j) máme rovnici $u_i + v_j = c_{ij}$, kde c_{ij} je konstanta. Máme $m+n-1$ rovnic pro $m+n$ neznámých, tedy zdánlivě o jednu rovnici méně, než je třeba. Je zřejmé, že soustava má nekonečně mnoho řešení.

Naštěstí mají rovnice velmi speciální tvar, vždy je to "některé u" plus "některé v". Když o nějakou hodnotu zvětšíme všechna u_i a o stejnou hodnotu zmenšíme všechna v_j, rovnice zůstanou zachovány. Navíc platí, že v metodě MODI potřebujeme proměnné u_i a v_j vždy jen v součtu $u_i + v_j$. Proto nám nejednoznačnost řešení nevadí. Klidně tedy můžeme zcela libovolně zvolit jednu (kteroukoli) z duálních proměnných a ostatní proměnné dopočítat.

Soustavu rovnic $u_i + v_j = c_{ij}$ pro bázické dvojice (i, j) není třeba explicite zapisovat. Výpočet lze provést přímo v dopravní tabulce. Stačí, když si uvědomíme, že každý bázický prvek představuje rovnici. Hodnoty proměnných u_i a v_j zapisujeme na pravém a dolním okraji tabulky.

Například v tabulce uvedené v 3.3.7 mohou duální proměnné vyjít takto:

	10	20	20	10	
20	6 10	3	7 0	8 10	6
20	2	7	1 20	8	0
20	3 0	1 20	4	2	3
	0	-2	1	2	

Postup výpočtu byl tento:

- Zvolili jsme $v_1 = 0$.
- $(1,1)$ je v bázi, z rovnice $u_1 + v_1 = 6$ tedy dopočteme $u_1 = 6$.
- $(1,3)$ je v bázi, z rovnice $u_1 + v_3 = 7$ tedy dopočteme $v_3 = 1$.
- $(1,4)$ je v bázi, z rovnice $u_1 + v_4 = 8$ tedy dopočteme $v_4 = 2$.
- $(2,3)$ je v bázi, z rovnice $u_2 + v_3 = 1$ tedy dopočteme $u_2 = 0$.
- $(3,1)$ je v bázi, z rovnice $u_3 + v_1 = 3$ tedy dopočteme $u_3 = 3$.
- $(3,2)$ je v bázi, z rovnice $u_3 + v_2 = 1$ tedy dopočteme $v_2 = -2$.

Poněvadž báze je souvislá a neobsahuje kružnici, lze každou duální proměnnou dopočítat přesně jedním způsobem. Připomeňme, že záporné hodnoty duálních proměnných nevadí a obecně se jim nelze vyhnout.

3.3.11 Kontrola přípustnosti duálního řešení je snadná. Stačí vzít všechny nebázické prvky dopravní tabulky a zkontrolovat, zda platí $u_i + v_j \leqq c_{ij}$. Pro bázické prvky díky předchozímu výpočtu musí platit rovnost (ovšem není na škodu to zkontrolovat).

Mnemotechnická pomůcka: je-li $c_{ij} < u_i + v_j$, je přeprava po trase $i \to j$ za cenu c_{ij} laciná vzhledem k ostatním cenám (jejichž vliv se zde uplatňuje přes u_i a v_j) a tedy se vyplatí na této trase zvýšit přepravu, tedy zvětšit x_{ij} na nenulovou hodnotu.

Všimněte si, že je opravdu lhostejné, jak byla zvolena výchozí hodnota při výpočtu duálního řešení. Na součty $u_i + v_j$ to nemá vliv.

Upozornění: počet prvků, které porušují přípustnost duálního řešení nemá žádný vztah k délce výpočtu, který nás ještě čeká.

3.3.12 Změna řešení. Pokud k bázi v dopravní tabulce přidáme jakýkoli další prvek, zvětšená množina prvků vždy obsahuje kružnici, která prochází přidaným prvkem. Tento fakt využijeme při výpočtu.

K bázi přidáme prvek (i,j) dopravní tabulky takový, že $c_{ij} < u_i + v_j$. (Doporučuje se vybírat prvek s největším rozdílem.) Tento prvek označme znaménkem $+$ na znamení toho, že zde chceme přepravu zvyšovat. Ostatní prvky kružnice označme střídavě znaménky $-$ a $+$ tak, aby prvky, které jsou v kružnici sousední, měly opačná znaménka. Kružnice má vždy sudý počet prvků. V prvcích označených $+$ budeme přepravu x_{ij} o nějakou (všude stejnou) hodnotu Δ zvyšovat a v prvcích označených $-$ ji budeme o stejnou hodnotu snižovat. Velikost změny Δ určíme jako minimum z hodnot x_{ij} označených $-$.

Řádkové a sloupcové součty hodnot x_{ij} se výše popsanou operací nezmění. Navíc všechna x_{ij} zůstanou nezáporná. Provedením změny tedy dostaneme opět přípustné řešení.

Nejméně jedna hodnota x_{ij} přitom klesne na nulu. Příslušný prvek tedy vyřadíme z báze. Pokud by kleslo na nulu několik hodnot x_{ij}, vyřadíme z báze jen jednu z nich, ostatní v bázi ponecháme (jde o degenerované řešení).

3.3.13 Pokračování příkladu. Pokračujme v našem příkladě a vyberme k zavedení do báze prvek $(1,2)$.

	10	20	20	10	
20	− 6 10	+ 3	7 0	8 10	6
20	2	7	1 20	8	0
20	+ 3 0	− 1 20	4	2	3
	0	-2	1	2	

Velikost změny zde vychází $\Delta = 10$. Po provedení změny dostaneme následující tabulku. Všimněte si, že zde máme jinou bázi, tedy jinou soustavu rovnic pro duální proměnné, tedy i hodnoty duálních proměnných vyšly jiné.

	10	20	20	10	
20	6	+ 3 10	7 0	− 8 10	3
20	2	7	1 20	8	-3
20	3 10	− 1 10	4	+ 2	1
	2	0	4	5	

Kritérium optimality stále není splněno a to v prvcích $(3,3)$ a $(3,4)$. Vybereme si prvek $(3,4)$, neboť má větší rozdíl $u_i + v_j - c_{ij}$. Opět vychází $\Delta = 10$ a po provedení změny dostaneme tuto tabulku:

	10	20	20	10	
20	6	+ 3 20	− 7 0	8	5
20	2	7	1 20	8	-1
20	3 10	− 1 0	+ 4	2 10	3
	0	-2	2	-1	

Ani zde ještě není kritérium optimality splněno a to kvůli prvku $(3,3)$. Poněvadž zde vychází $\Delta = 0$, odložíme tento zapeklitý případ na 3.3.16.

3.3.14 O kolik se zlepší účelová funkce. Zavedením prvku (i,j) do báze se účelová funkce zlepší (zmenší) o hodnotu $(u_i + v_j - c_{ij})\Delta$.

Toto tvrzení lze odvodit z platnosti soustavy rovnic pro duální proměnné a z toho, že přepravu měníme v souladu s 3.3.12 pouze v prvcích báze a v prvku, který do báze zavádíme. Nejste-li tak hloubaví, ověřte alespoň, že to platí na příkladě změn uvedených v 3.3.13.

Toto tvrzení je přímou analogií zlatého pravidla simplexové metody 2.6.3. Výrazy $(u_i + v_j - c_{ij})$ jsou analogií relativních cen.

Je-li několik prvků, které se nabízejí k zavedení do báze (tj. platí pro ně $u_i + v_j > c_{ij}$), nabízejí se tyto možnosti:

1. Vybírat prvek, který dá největší zlepšení účelové funkce.
2. Vybírat prvek s největším rozdílem $(u_i + v_j - c_{ij})$.
3. Vybírat prvek s nejnižším číslem řádku a v rámci takto vybraného řádku vybrat sloupec s nejnižším číslem sloupce.

Podobně jako u lineárního programování (viz 2.6.7), i zde má každé pravidlo své klady a zápory. První pravidlo dá víc práce při výpočtu hodnot Δ, ale rychleji zlepšuje účelovou funkci. Třetí pravidlo poskytuje jistotu, že se výpočet nezacyklí kvůli degenerovaným řešením, ale zlepšuje účelovou funkci nejpomaleji. Druhé pravidlo představuje rozumný kompromis.

3.3.15 Cesty bývají klikaté. Je třeba upozornit, že kružnice (tedy uzavřená cesta v tabulce) nemusí vždy vypadat jako úhledný obdélníček. Může být delší, klikatější a může vypadat docela propleteně. Nenechte se tím odradit.

Pokud se klikaté cesty zaleknete a rozhodnete se měnit řešení jinak než podle 3.3.12, nic vám v tom nezabrání, ale už nebudete mít záruku, že provedením změny řešení nezhoršíte. Předpověď změny účelové funkce popsaná v 3.3.14 platí pouze za předpokladu, že změna řešení x je prováděna v souladu s 3.3.12.

3.3.16 Degenerované řešení. Řešení dopravní úlohy nazýváme degenerovaným, má-li méně než $m+n-1$ nenulových složek (srovnejte s 2.6.8, str. 25). Degenerovaných řešení se týkají čtyři druhy problémů.

První problém se týká situace, kdy při provedení změny podle 3.3.12 klesne na nulu více než jedna hodnota x_{ij}. V takovém případě vyřaďte z báze jednu z vynulovaných hodnot a ostatní v bázi ponechejte. Tím zaručeně zachováte korektní bázi. Na otázku, kterou z nově vynulovaných hodnot vyřadit a kterou v bázi nechat, není jednoduchá odpověď, ale může to být kterákoli.

Druhý problém se týká situace, kdy vyjde velikost změny $\Delta = 0$. I v takovém případě změnu formálně proveďte. Hodnoty x se tím sice nezmění, ale změní se báze. Prvek, který měl být do báze zaveden skutečně do báze zaveďte a z báze vyřaďte některý z prvků, díky kterým vyšlo $\Delta = 0$, tedy prvek, kde $x_{ij} = 0$ a který je označen znaménkem $-$ (na znamení, že zde chceme hodnotu x_{ij} snižovat.

Vyzbrojeni těmito radami můžeme pokračovat v našem příkladě. Pro pohodlí čtenáře zopakujme poslední tabulku:

	10	20	20	10	
20	6	+ 3 20	− 7 0	8	5
20	2	7	1 20	8	-1
20	3 10	− 1 0	+ 4	2 10	3
	0	-2	2	-1	

Zde vychází $\Delta = 0$ dokonce díky dvěma nulovým prvkům báze. Ponecháme-li v bázi prvek $(1,3)$, dostaneme tuto tabulku:

	10	20	20	10	
20	6	3 20	7 0	8	6
20	2	7	1 20	8	0
20	3 10	1	4 0	2 10	3
	0	-3	1	-1	

Konečně je jsme dosáhli stavu, kdy je splněno kritérium optimality 3.3.4, máme tedy optimální řešení. Všimněte si, že stejné řešení x jsme měli již v předchozí tabulce, ale poněvadž jsme měli jinou bázi, nevěděli jsme, že je optimální.

Třetí problém s degenerovaným řešením se týká právě popsané situace, kdy řešení x již bylo de facto optimální, ale hodnoty duálního řešení to ještě nepotvrzovaly. Teprve po změně báze se

ukázalo, že řešení již bylo optimální. Někdy může být zapotřebí i několika změn báze, než se ukáže, že již chvíli máme optimální řešení.

Konečně čtvrtý problém s degenerovaným řešením je ten, že může vzniknout falešný dojem, že úloha má více optimálních řešení.

3.3.17 Vícenásobné optimum. Pokud v dopravní tabulce, která obsahuje optimální řešení x, pro některý nebázický prvek (i, j) platí $c_{ij} = u_i + v_j$, pak zavedením tohoto prvku do báze dostaneme řešení, které má stejnou hodnotu účelové funkce a je tedy také optimální.

Je zde malá záludnost — řešení, které takto dostaneme, sice je určitě také optimální, ale nemusí být různé od původního optimálního řešení, může se lišit pouze jinou bází. Všimněte si, že tento případ nastal v poslední tabulce našeho příkladu v 3.3.16. Zavedeme-li do báze prvek $(1, 1)$, bude $\Delta = 0$ a proto provedením změny dostaneme stejné řešení. O vícenásobné optimum se tedy nejedná.

3.3.18 Stabilita řešení. Rozbor stability optimálního řešení vzhledem ke změnám jednotlivých cen je poměrně snadný.

Jde-li o změnu ceny c_{ij} nebázického prvku (i, j), pak, dokud cena c_{ij} neklesne pod součet $u_i + v_j$, zůstává stávající řešení optimální. V našem příkladě např. snadno vidíme, že pokud cena c_{14} neklesne pod hodnotu $5 = 6 - 1$, stávající řešení zůstane optimálním. Již méně snadno, totiž změnou báze, lze zjistit, že totéž řešení zůstane optimálním dokonce i když c_{14} neklesne pod hodnotu 4.

Mění-li se cena bázického prvku, je situace složitější, je třeba nově spočítat duální řešení u, v a zkontrolovat jeho přípustnost.

3.3.19 Poznámka o celočíselnosti. Jsou-li kapacity dodavatelů a požadavky spotřebitelů celočíselné, pak dopravní úloha má celočíselné optimální řešení. Je to proto, že v celém výše popsaném postupu výpočtu se vyskytují jen operace sečítání a odčítání. Všimněte si, že celočíselnost nebo neceločíselnost cenových koeficientů nemá na celočíselnost řešení x vliv.

Kapitola 4

Celočíselné lineární programování

Celočíselné lineární programování se od obecného lineárního programování liší „pouze“ tím, že (některé) proměnné smějí nabývat pouze celočíselných hodnot. Tato zdánlivá maličkost dokáže úlohu pořádně zkomplikovat. Úlohy celočíselného LP jsou obecně velmi obtížné.

Celočíselné lineární programování bývá označováno zkratkou ILP (Integer Linear Programming).

4.1 Typy úloh a aplikace

4.1.1 Typy proměnných v celočíselných úlohách. Obvykle se rozlišují tři základní typy proměnných:

Spojité, které mohou nabývat všech reálných hodnot z nějakého intervalu. Množina hodnot proměnné je omezena pomocí běžné podmínky pro jednotlivou proměnnou, zpravidla jde o podmínku nezápornosti.

Celočíselné, které mohou nabývat pouze celočíselných hodnot.

Binární (též nula-jedničkové), které mohou nabývat pouze hodnot 0 nebo 1. Striktně vzato, jde o kombinaci podmínky celočíselnosti a podmínky omezení na interval $\langle 0, 1\rangle$, tedy o zvláštní případ výše popsaného, ale jde o případ velmi důležitý jak z hlediska praxe, tak i z hlediska metod výpočtu.

V kapitole 2 jsme se zabývali pouze případem, kdy všechny proměnné byly spojitého typu.

4.1.2 Příklady aplikací jsou časté a nesmírně rozmanité.

Nejjednodušší příklady se týkají výroby kusového zboží při omezených zdrojích. Samozřejmě záleží na charakteru praktické úlohy. Např. velkopekárna, která denně chrlí statisíce rohlíků, může celočíselnou povahu úlohy s klidným svědomím zanedbat, protože zaokrouhlením výroby na celá čísla vznikne nepatrná chyba, srovnatelná s jinými chybami, které model oproti realitě má. Pokud počty vyráběných výrobků budou malé a jejich cena veliká, pak již je třeba celočíselnost vzít v úvahu.

Další příklady se týkají rozhodování o investicích. Zde obvykle pracujeme s binárními proměnnými a omezující podmínky vyjadřují jednak nároky na zdroje, jednak případné vztahy logické podmíněnosti. Například k továrně na výrobu hliníku je třeba postavit elektrárnu. Nebo k jaderné elektrárně je vhodné mít elektrárnu přečerpávací.

Zejména je však třeba zmínit kombinatorické úlohy.

4.1.3 Problém batohu. Lupič našel v trezoru n předmětů, které váží $w_1, \ldots, w_n$ a mají ceny $c_1, \ldots, c_n$. Lup chce odnést v batohu, který má nosnost K. Lupič ví, že nesmí batoh přetížit, jinak mu praskne a bude chycen. Jeho snahou je odnést lup s co největší hodnotou.

4.1.4 Optimální nezávislá množina. Lupič našel v trezoru n předmětů o cenách $c_1, \ldots, c_n$. Na rozdíl od problému batohu však tentokrát není omezen váhou předmětů, ale tím, které dvojice předmětů lze nést spolu ve stejném batohu.

Úlohu lze modelovat jako úlohu ILP. Pro každou dvojici vylučujících se předmětů i, j zavedeme strukturní podmínku $x_i + x_j \leqq 1$.

Poznamenejme, že totéž lze modelovat pomocí teorie grafů. Předměty jsou vrcholy grafu, hrany grafu jsou dvojice vylučujících se předmětů. Hledáme tzv. nezávislou množinu vrcholů – její vrcholy nesmí být spojeny hranou – a to takovou, aby součet cen byl maximální.

4.1.5 Úloha o řezných plánech je také známa jako úloha o dělení tyčovitého materiálu a v angličtině pod názvem *bin packing*.

Ze zásoby tyčí o délce D máme za úkol nařezat p_i kusů o délce d_i pro $i = 1, \ldots, P$ a to tak, abychom výchozích tyčí o délce D spotřebovali co nejméně.

Existuje jen konečně mnoho způsobů, tzv. řezných plánů, jak z tyče o délce D nařezat několik kusů o délkách z množiny $\{d_1, \ldots, d_P\}$. Všechny tyto způsoby očíslujme čísly $j = 1, \ldots, n$ a zavedeme n celočíselných proměnných x_j, které vyjadřují, kolikrát má být j-tý řezný plán použit. Účelovou funkcí je počet spotřebovaných tyčí, tj. prostý součet všech proměnných $\sum_{i=1}^{n} x_j$, a tento počet minimalizujeme. Lineární omezující podmínky pak vyjadřují, že použitím řezných plánů dostaneme požadované počty výsledných tyčí. Tedy koeficient a_{ij} je roven počtu tyčí o délce d_i v řezném plánu j. Všechny strukturní omezující podmínky jsou typu üvvětší nebo rovno a pravé strany jsou d_i.

Pozor! Výše popsané řešení problému dělení tyčovitého materiálu převodem na celočíselné lineární programování je velmi neefektivní, ale zaručeně poskytuje přesné optimum. Existují jednoduché (až triviální) heuristické postupy, které poskytují skoro vždy optimální řešení a vždy řešení, které se od optimálního liší jen málo, viz 4.5.6, str. 73.

4.1.6 Diskrétní proměnné Úlohu, v níž proměnná x nabývá diskrétních hodnot z konečné množiny $\{k_1, k_2, \ldots, k_r\}$ o r prvcích, lze převést na úlohu s r binárními proměnnými $x_1, \ldots, x_r$.

$$\begin{aligned} -x + k_1x_1 + k_2x_2 + \ldots + k_rx_r &= 0 \\ x_1 + x_2 + \ldots + x_r &= 1\,. \end{aligned}$$

Tyto dvě rovnice spolu s podmínkami, že proměnné $x_1, \ldots, x_r$ jsou binární, zaručují, že proměnná x může nabýt kterékoli hodnoty z množiny $\{k_1, k_2, \ldots, k_r\}$ a žádné jiné hodnoty nabýt nemůže.

4.1.7 Modelování logických podmínek. Binární proměnné obvykle modelují rozhodnutí typu ano–ne, popř. pravda–nepravda. Zpravidla se dodržuje konvence, že hodnota 1 znamená "ano", "pravda" a hodnota 0 znamená "ne", "nepravda".

Binární proměnné jsou často svázány podmínkami logického charakteru.

Vezměme např. situaci, kdy rozhodujeme o několika investicích a z povahy věci vyplývá omezení, že investice 1 má smysl pouze pokud je současně provedena i investice 2. Mezi omezujícími podmínkami úlohy tedy je podmínka, kterou lze vyjádřit logickou výrokovou formulí $x_1 \Rightarrow x_2$, my však potřebujeme tuto podmínku vyjádřit ve tvaru lineární nerovnosti nebo rovnice. V našem případě lze použít nerovnost $x_1 \leqq x_2$. Vskutku, jestliže $x_1 = 0$, pak proměnná x_2 může být jakákoli (tedy 0 nebo 1), ale pokud $x_1 = 1$, pak nerovnost $x_1 \leqq x_2$ způsobí, že také $x_2 = 1$. A to je přesně to, co vyjadřovala logická podmínka $x_1 \Rightarrow x_2$.

Zde je několik dalších příkladů jednoduchých formulí:

$$\begin{array}{ll} x_2 \Leftrightarrow \neg x_1 & x_1 + x_2 = 1 \\ x_1 \Rightarrow x_2 & x_1 \leqq x_2 \\ x_1 \Rightarrow (x_2 \vee x_3) & x_1 \leqq x_2 + x_3 \\ (x_2 \vee x_3) \Rightarrow x_1 & x_2 + x_3 \leqq 2x_1 \\ x_1 \Leftrightarrow (x_2 \vee x_3) & 2x_1 \geqq x_2 + x_3 \geqq x_1 \end{array}$$

Pro jednoduché formule se obvykle podaří jejich vyjádření uhodnout a ověřit, že to funguje. Ověření spočívá v tom, že vezmeme všechny kombinace logických hodnot všech proměnných, které se ve formuli vyskytují, a zkontrolujeme, zda ve všech případech formule i soustava lineárních podmínek vyjadřují totéž. Pokud kontrola vyjde, máme vyhráno. Pokud nevyjde, můžeme zkusit lineární podmínky nějak "vyspravit" a samozřejmě znovu zkontrolovat.

Pokud vám metoda "uhodni a ověř" nevyhovuje, existuje metoda silnějšího kalibru, ovšem vyžaduje trochu znalosti z matematické logiky. Každou výrokovou formuli (jakkoli složitou) totiž lze přepsat do tzv. *konjunktivní normální formy*, tj. do tvaru konjunkce několika takzvaných klauzulí, přičemž každá klauzule je disjunkcí takzvaných literálů a literál je buď logická proměnná nebo negace proměnné. Příkladem klauzule je třeba $(x_3 \vee \neg x_4 \vee x_6)$, kde x_3, $\neg x_4$ a x_6 jsou literály. Vtip je v tom, že každou klauzuli lze snadno vyjádřit jako lineární nerovnost, kde každou negaci $\neg x_i$ nahradíme výrazem $(1 - x_i)$. Klauzuli z našeho příkladu lze tedy vyjádřit jako nerovnost $x_3 + (1 - x_4) + x_6 \geqq 1$, tedy po zjednodušení $x_3 - x_4 + x_6 \geqq 0$.

Například výrokovou formuli

$$x_1 \Rightarrow (x_2 \vee (x_3 \wedge x_4))$$

lze ekvivalentně vyjádřit v konjunktivní normální formě

$$(\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$$

a tu již lze snadno převést na soustavu nerovností

$$\begin{array}{rcl} (1 - x_1) + x_2 + x_3 & \geqq & 1 \\ (1 - x_1) + x_2 + x_4 & \geqq & 1 \end{array}$$

4.1.8 Počet nenulových proměnných. Mějme úlohu, kde proměnné $x_1, \ldots, x_s$ jsou spojité a omezené na interval $\langle 0, K)$, ale z těchto s proměnných jich nesmí být kladných více než $r < s$. Například obalovna může kvůli technologickým omezením z celkového sortimentu s druhů směsi vyrábět v jednom dni pouze r z nich. Kolik budeme vyrábět jednotlivých druhů vyjádříme hodnotami (spojitých) proměnných $x_1, \ldots, x_s$, ale pouze r z těchto proměnných smí nabývat kladné hodnoty.

Zavedeme s pomocných binárních proměnných $x'_1, \ldots, x'_s$. Proměnná x'_i bude vyjadřovat, že odpovídající proměnná x_i smí nabývat kladné hodnoty. Zařídíme to pomocí podmínek ve tvaru $x_i \leqq K x'_i$. Platí tedy $x_i > 0 \Rightarrow x'_i = 1$. A nyní již stačí přidat podmínku $x'_1 + x'_2 + \ldots + x'_s \leqq r$.

Poznamenejme, že s konstantou K v praktických úlohách nebývá problém. V našem případě lze jako K použít například denní kapacitu obalovny.

4.1.9 Metody řešení celočíselných úloh. První nápad obvykle je tento: Zanedbáme (tzv. *relaxujeme*) podmínky celočíselnosti, úlohu vyřešíme bez nich a pak výsledek zaokrouhlíme na celá čísla. Je třeba upozornit, že zaokrouhlením často dostaneme nepřípustné řešení, ostatně ani není jasné, zda se má zaokrouhlovat nahoru nebo dolů. Další potíž je v tom, že řešení relaxované úlohy může být i velmi vzdáleno od celočíselného optima. A konečně, vůbec není jisté, že celočíselná úloha má vůbec nějaké celočíselné řešení.

Dodejme ovšem, že v praxi se zaokrouhlení používá a to v situaci, kdy víme, že chyba, které se tím dopustíme, je přijatelná.

Metoda hrubé síly (též enumerativní metoda) spočívá ve vyzkoušení všech možností. Lze ji použít, pokud množina přípustných řešení relaxované úlohy je omezená, tj. je celá obsažena v nějakém konečném (mnohorozměrném) kvádru. K tomu stačí pro každou proměnnou x_i najít interval takový, že pro všechna přípustná řešení proměnná x_i určitě leží v tomto intervalu. Nyní postupně vyzkoušíme všechny body kvádru s celočíselnými souřadnicemi, pro každý z nich ověříme platnost omezujících podmínek a pokud tento bod je přípustným řešením, vypočteme hodnotu účelové funkce a porovnáme ji s nejlepším dosud nalezeným řešením.

Závažnou nevýhodou této metody je (zpravidla) obrovská výpočetní náročnost. Úlohu s deseti binárními proměnnými takto řešit lze, počet možností je "jen" 1024. Ovšem úlohu se stovkou binárních proměnných by všechny počítače světa řešily tisíce let.

Metoda sečných nadrovin je popsána v sekci 4.2

Metoda větví a mezí je popsána dále v sekci 64.

4.2 Metoda sečných nadrovin

4.2.1 Metoda sečných nadrovin vychází z optimálního řešení relaxované úlohy. Je-li optimální řešení relaxované úlohy celočíselné, pak jsme hotovi, protože toto řešení je zároveň optimálním řešením původní celočíselné úlohy.

Je-li řešení relaxované úlohy neceločíselné, přidává se k úloze další omezující podmínka a to taková, která

- nevyloučí žádné celočíselné přípustné řešení, ale
- vyloučí optimum relaxované úlohy (učiní ho nepřípustným).

Geometricky vzato, přidaná omezující podmínka určuje poloprostor ohraničený nadrovinou a tato nadrovina „usekne" z množiny přípustných řešení relaxované úlohy nějaká neceločíselná řešení. Proto mluvíme o metodě sečných nadrovin (anglicky cutting plane method).

Úlohu s přidanou omezující podmínkou řešíme zcela stejnou metodou jako úlohu původní: relaxujeme podmínky celočíselnosti, najdeme optimum relaxované úlohy a opět testujeme, zda jsme dostali celočíselné řešení. To se opakuje, dokud nejsou všechny souřadnice řešení celočíselné.

Sečné nadroviny lze generovat mnoha způsoby. Nejznámější je tzv. Gomoryho metoda. K jejímu výkladu budeme potřebovat následující pojem:

4.2.2 Dolní celá část reálného čísla x je největší celé číslo, které není větší než x. Dolní celou část čísla x značíme $\lfloor x \rfloor$. Tedy například

$$\lfloor 2.5 \rfloor = 2 \qquad\qquad \lfloor -2.5 \rfloor = -3 \,.$$

Každé reálné číslo x lze zapsat ve tvaru $x = \lfloor x \rfloor + r$, kde neceločíselná část r splňuje $0 \leqq r < 1$.

4.2.3 Gomoryho metoda je použitelná na tzv. celočíselně celočíselné úlohy, tj. na úlohy, kde všechny proměnné jsou celočíselné a všechny konstanty v omezujících podmínkách (tj. všechny a_{ij} a b_i) jsou celá čísla. Z toho mj. plyne, že celočíselné řešení bude mít i všechny doplňkové proměnné celá čísla a tedy i po převodu úlohy na kanonický tvar dostaneme celočíselně celočíselnou úlohu.

Gomoryho algoritmus odvozuje sečnou nadrovinu ze simlpexové tabulky, která representuje optimální řešení relaxované úlohy, a to z takového jejího i-tého řádku, který má na pravé straně neceločíselnou hodnotu b_i. Tento i-tý řádek reprezentuje rovnici

$$a_{i,1}x_1 + a_{i,2}x_2 + \ldots a_{i,n}x_n = b_i \,. \tag{4.1}$$

Konstanty $a_{i,j}$ a b_i, které se vyskytují v této rovnici rozložíme na dolní celou část a zbytek

$$a_{i,j} = \lfloor a_{i,j} \rfloor + r_j \qquad a \qquad b_i = \lfloor b_i \rfloor + r \,, \tag{4.2}$$

k úloze přidáme omezující podmínku

$$r_1 x_1 + r_2 x_2 + \ldots + r_n x_n \geqq r \,. \tag{4.3}$$

Jak uvidíme dále v 4.2.4, přidaná podmínka má dvě důležité vlastnosti:

1. přidáním této podmínky nepřijdeme o žádné přípustné celočíselné řešení (tj. všechna přípustná celočíselná řešení splňují i tuto nově přidnou podmínku),

2. dosavadní relaxované optimum, které má neceločíselnou souřadnci $x_{t_i} = b_i$, přidanou podmínku nesplňuje.

Přidaná podmínka tedy z množiny (spojitých) přípustných řešení „odsekne“ dosavadní relaxované optimum, ale odsekne ho „šetrným způsobem“, totiž tak, že neodsekne žádný přípustný celočíselný bod. Toto „odsekávání“ (odtud název metody) se opakuje, až nakonec je odsekáváním „obnažen“ celočíselný bod, který je optimálním celočíselným řešením původní úlohy.

Připomeňme, že přidání omezující podmínky k simplexové tabulce je popsáno v 2.9.11, str. 40.

4.2.4 Zdůvodnění Gomoryho metody. V rovnici (4.1), která platí proto, že nám vyšla v simplexové tabulce, rozepíšeme všechny konstanty $a_{i,j}$ a b_i podle (4.2) a rovnici upravíme tak, že na levou stranu převedeme všechny členy s neceločíselnými koeficienty a na pravou stranu všechny členy s koeficienty celočíselnými. Dostaneme tak rovnici:

$$r_1 x_1 + r_2 x_2 + \ldots + r_n x_n - r = \lfloor b_i \rfloor - \lfloor a_{i,1} \rfloor x_1 - \lfloor a_{i,2} \rfloor x_2 - \ldots \lfloor a_{i,n} \rfloor x_n \tag{4.4}$$

Poněvadž jsou všechna x_j nezáporná a poněvadž $r < 1$, o levé straně rovnice 4.4 víme, že

$$r_1 x_1 + r_2 x_2 + \ldots + r_n x_n - r > -1 \,, \tag{4.5}$$

Zároveň ovšem víme, že pravá strana rovnice 4.4 je celočíselná, neboť všechna x_j mají být celá čísla. Proto musí i levá strana být celé číslo a to celé číslo ostře větší než -1. Je-li tedy x přípustné a celočíselné, musí platit

$$r_1 x_1 + r_2 x_2 + \ldots + r_n x_n - r \geqq 0 \,. \tag{4.6}$$

Odtud již snadno plyne podmínka (4.3)

4.2.5 Příklad. Řešme celočíselně celočíselnou úlohu (porovnejte s 2.2.2, str. 12):

$$\begin{aligned} 2x_1 + 3x_2 &\rightarrow \max \\ x_1 - 2x_2 &\leqq 2 \\ -2x_1 + x_2 &\leqq 2 \\ x_1 + x_2 &\leqq 4 \\ x_1 &\geqq 0 \\ x_2 &\geqq 0 \end{aligned}$$

Nejprve najdeme optimální řešení relaxované úlohy

		2	3	0	0	0	max
0	3.	1	−2	1	0	0	2
0	4.	−2	1	0	1	0	2
0	5.	1	1	0	0	1	4
		−2	−3	0	0	0	0
0	3.	−3	0	1	2	0	6
3	2.	−2	1	0	1	0	2
0	5.	3	0	0	−1	1	2
		−8	0	0	3	0	6
0	3.	0	0	1	1	1	8
3	2.	0	1	0	$\frac{1}{3}$	$\frac{2}{3}$	$3\frac{1}{3}$
2	1.	1	0	0	$-\frac{1}{3}$	$\frac{1}{3}$	$\frac{2}{3}$
		0	0	0	$\frac{1}{3}$	$2\frac{2}{3}$	$11\frac{1}{3}$

Optimální řešení relaxované úlohy $(\frac{2}{3}, 3\frac{1}{3}, 8, 0, 0)$ není celočíselné. Gomoryho sečnou nadrovinu vygenerujeme podle první neceločíselné souřadnice $x_1 = \frac{2}{3}$. Přidáváme omezující podmínku

$$\tfrac{2}{3}x_4 + \tfrac{1}{3}x_5 \geqq 1/3 \,,$$

což se v simplexové tabulce projeví jako čtvrtý řádek a šestý sloupec.

		2	3	0	0	0	0	max
0	3.	0	0	1	1	1	0	8
3	2.	0	1	0	$\frac{1}{3}$	$\frac{2}{3}$	0	$3\frac{1}{3}$
2	1.	1	0	0	$-\frac{1}{3}$	$\frac{1}{3}$	0	$\frac{2}{3}$
0	6.	0	0	0	$-\frac{2}{3}$	$-\frac{1}{3}$	1	$-\frac{2}{3}$
		0	0	0	$\frac{1}{3}$	$2\frac{2}{3}$	0	$11\frac{1}{3}$

Poněvadž máme záporné číslo na pravé straně, pokračujeme duálně simplexovou metodou. Klíčový prvek bude na místě (4,4). Po provedení Jordanovy eliminace dostaneme:

		2	3	0	0	0	0	max
0	3.	0	0	1	0	$\frac{1}{2}$	−1	7
3	2.	0	1	0	0	$\frac{2}{3}$	$0\frac{1}{2}$	3
2	1.	1	0	0	0	$\frac{1}{2}$	$-\frac{1}{2}$	1
0	6.	0	0	0	1	$\frac{1}{2}$	$-1\frac{1}{2}$	1
		0	0	0	0	$2\frac{1}{2}$	$\frac{1}{2}$	11

Optimální celočíselné řešení je $(1, 3, 7, 1, 0, 0)$.

4.2.6 Poznámky Gomoryho metoda sečných nadrovin zpravidla nepracuje moc efektivně, obvykle potřebuje značný počet iterací.

Pro speciální úlohy (např pro problém obchodního cestujícího) jsou známy dokonalejší metody konstrukce sečných nadrovin, které „odsekávají“ nepotřebná řešení „po větších kusech“ a tedy vedou k řešení rychleji.

4.3 Metoda větví a mezí

Metoda větví a mezí je značně univerzální metodou k řešení optimalizačních úloh. Lze ji použít k řešení úloh celočíselného lineárního programování, ale i pro úlohy, které s lineárním programováním nemají skoro nic společného.

V této sekci podáme obecný výklad metody větví a mezí a uvedeme dva příklady použití.

4.3.1 Větvení množiny přípustných řešení. Množinu přípustných řešení úlohy U označme symbolem $\text{Př}(U)$.

Mějme optimalizační úlohu U s účelovou funkcí f. Úlohu U můžeme řešit (mimo jiné) tak, že vytvoříme úlohy $U_1, \ldots, U_n$ se stejnou účelovou funkcí (tzv. *podúlohy* úlohy U) takové, že

$$\text{Př}(U) = \text{Př}(U_1) \cup \ldots \cup \text{Př}(U_n) \,.$$

Optimální řešení úlohy U pak lze získat tak, že vybereme nejlepší ze všech optimálních řešení podúloh $U_1, \ldots, U_n$. Samozřejmě se při tom snažíme (v tom je smysl větvení), aby podúlohy $U_1, \ldots, U_n$ byly v nějakém smyslu snazší než původní úloha U, tj. aby např. byly menší. Dodejme, že při tom je výhodné, když množiny $\text{Př}(U_1), \ldots, \text{Př}(U_n)$ jsou navzájem disjunktní, ale není to nutné.

Nalezení optimálního řešení úlohy U je snadné, pokud pro každou podúlohu U_i:

1. buď najdeme optimální řešení úlohy U_i,
2. nebo prokážeme, že úloha U_i nemá žádné přípustné řešení,
3. nebo prokážeme, že žádné přípustné řešení úlohy U_i není lepší než nějaké v té době již známé přípustné řešení úlohy U.

Často se ovšem stává, že pro některou podúlohu (a často ne jen jednu) nejsme schopni přímo zjistit ani jednu z výše uvedených tří možností. V tom případě pro každou takovou podúlohu, označme ji U_i, použijeme stejný postup jako pro úlohu U, tj. rozvětvíme ji na podúlohy $U_{i_1}, \ldots, U_{i_m}$ a pro takto získané podúlohy se opět snažíme zjistit některou z výše uvedených možností. Některé z takto získaných podúloh může být nutno dále větvit.

4.3.2 Strom řešení. Jednotlivé podúlohy můžeme pokládat za vrcholy kořenového stromu, který nazýváme stromem řešení. Kořenem je výchozí úloha U a z každé úlohy, která byla větvena na podúlohy, vedou hrany ke všem jejím podúlohám. Během výpočtu se strom řešení mění (zvětšuje).

Pro výpočet jsou zajímavé pouze listy stromu řešení. Listy dělíme na tzv. *živé* a *mrtvé*. Za mrtvé pokládáme ty listy (podúlohy), u nichž se již podařilo prokázat některou ze tří možností uvedených v 4.3.1. Těmito podúlohami se již není třeba dále zabývat. Ostatní listy (podúlohy) jsou živé.

Živou podúlohu je třeba buď umrtvit (zjištěním některé ze tří možností z 4.3.1), anebo rozvětvit. Rozvětvením podúloha přestává být listem, ve stromě řešení se však místo ní objeví několik nových listů. Výpočet končí v okamžiku, kdy všechny listy stromu řešení jsou mrtvé.

4.3.3 Odhady a meze. Pokud pro některou podúlohu U_i zjistíme, že platí možnost 3, tj. pokud prokážeme, že žádné přípustné řešení úlohy U_i není lepší než nějaké v té době již známé řešení úlohy U, je to při řešení dobrá zpráva, protože tuto podúlohu není třeba větvit. Nejlepší dosud známé řešení úlohy U se však během výpočtu obvykle mění (zlepšuje), a tak se snadno stane, že možnost 3 se o nějaké podúloze podaří prokázat, teprve až najdeme dostatečně dobré řešení celé úlohy U.

Proto bývá výpočet uspořádán tak, že jednak evidujeme nejlepší dosud známé řešení a hodnotu jeho účelové funkce, jednak pro každou podúlohu U_i vypočteme číslo $\text{Odh}(U_i)$ takové, že žádné přípustné řešení úlohy U_i není lepší než $\text{Odh}(U_i)$.

Číslo $\text{Odh}(U_i)$ bývá v literatuře nazýváno u minimalizačních úloh *dolním odhadem* a u maximalizačních úloh *horním odhadem*. Čím je odhad blíže skutečné hodnotě optimálního řešení podúlohy, tím je tento odhad užitečnější, neboť tím snáze jeho pomocí prokážeme, že pro podúlohu platí možnost 3. Odhad je tedy tím lepší, čím je méně optimistický. Někdy lze volit mezi několika metodami výpočtu odhadu, které dávají různě kvalitní výsledky.

Způsob výpočtu odhadu samozřejmě velice závisí na řešené úloze a na způsobu vytváření podúloh. Jako odhad pro podúlohu U_i se často používá hodnota optimálního řešení tzv. relaxované podúlohy:

4.3.4 Relaxace nepříjemných omezujících podmínek. Poněvadž množina přípustných řešení úlohy je vždy popsána (zadána) pomocí omezujících podmínek, provádí se i větvení úlohy na její podúlohy pomocí dodatečných omezujících podmínek, které se k větvené úloze přidávají.

Obvyklá situace je tato: Omezující podmínky výchozí úlohy U lze rozložit na dvě skupiny, nazvěme je *příjemné* a *nepříjemné*. Vynecháme-li (takzvaně *relaxujeme*) nepříjemné omezující podmínky, dostaneme tzv. relaxovanou úlohu, která má již jenom příjemné omezující podmínky a kterou dovedeme rychle řešit. Jinak řečeno, úlohu s příjemnými podmínkami umíme rychle řešit, nepříjemné podmínky úlohu komplikují a činí ji těžkou.

Najdeme-li optimální řešení r úlohy, která vznikla relaxací z U, jsou dvě možnosti: buď řešení r splňuje, nebo nesplňuje nepříjemné omezující podmínky. Pokud splňuje, pak máme štěstí, protože řešení r je zároveň optimálním řešením i samotné úlohy U, a úlohu U tedy není třeba větvit. Obvykle však štěstí nemáme a řešení r nepříjemné omezující podmínky nesplňuje. V takové situaci uděláme dvě věci: za prvé, hodnotu optimálního řešení relaxované úlohy můžeme použít jako odhad, a za druhé, platnost oněch nepříjemných podmínek si snažíme vynutit přidáváním podmínek příjemných, a to i za cenu toho, že se nám úloha rozpadne (rozvětví) na několik podúloh.

Podstatné je, že při větvení úlohy přidáváme vždy jen příjemné omezující podmínky. Tím je totiž zaručeno, že takto vzniklé podúlohy jsou stejného typu jako větvená úloha, a můžeme tedy opět použít onen trik s relaxací a řešením relaxované podúlohy.

Které omezující podmínky můžeme pokládat za příjemné, je dáno naší schopností rozumně rychle řešit příslušné relaxované podúlohy. Zpravidla pro danou úlohu existuje několik možností, jak zvolit prostor řešení a jak přeformulovat omezující podmínky, přitom obojí má značný vliv na pracnost řešení metodou větví a mezí.

4.3.5 Volba podúlohy k větvení by mohla být založena na prohledávání stromu řešení např. do šířky nebo do hloubky. Častěji se však pro každou podúlohu vypočte hodnota, nazvěme ji prioritou, a mezi živými podúlohami vybíráme k větvení tu, která má prioritu nejlepší. Běžně se v roli priority používá (dolní nebo horní) odhad. Vybíráme pak podúlohu, která má odhad nejlepší (nejnadějnější), neboť u této podúlohy lze očekávat, že zlepšíme nejlepší dosud nalezené řešení a pomocí odhadů pak umrtvíme některé dosud živé podúlohy.

4.3.6 Shrnutí metody větví a mezí. Při výpočtu udržujeme nejlepší dosud nalezené přípustné řešení $\check{r}$ dané úlohy U a jeho hodnotu účelové funkce L. Na začátku $\check{r}$ není definováno a jako hodnotu L použijeme nejhorší možnou hodnotu ($+\infty$ nebo $-\infty$).

Doporučuje se však ještě před zahájením výpočtu metodou větví a mezí získat nějakým heuristickým postupem (viz 4.5) co nejlepší přípustné řešení $\check{r}$ a jeho hodnotu L, neboť to může následující výpočet urychlit.

Dále při výpočtu udržujeme seznam živých podúloh.

Obvykle ještě před zařazením do seznamu pro každou podúlohu vypočteme odhad, nejčastěji řešením relaxované verze příslušné podúlohy. Pokud přitom dostaneme přípustné řešení, porovnáme je s řešením $\check{r}$ a dále uchováváme lepší z nich. Pokud bylo řešení $\check{r}$ nahrazeno lepším, tj. pokud se zlepšila hodnota L, projdeme seznam živých podúloh a vyřadíme ty, které jsou díky nové hodnotě L umrtveny.

Pokud podúloha nemá žádné přípustné řešení nebo pokud pro ni dostaneme odhad, který je horší než L, pak tuto podúlohu okamžitě umrtvíme a do seznamu živých podúloh ji vůbec nezapisujeme.

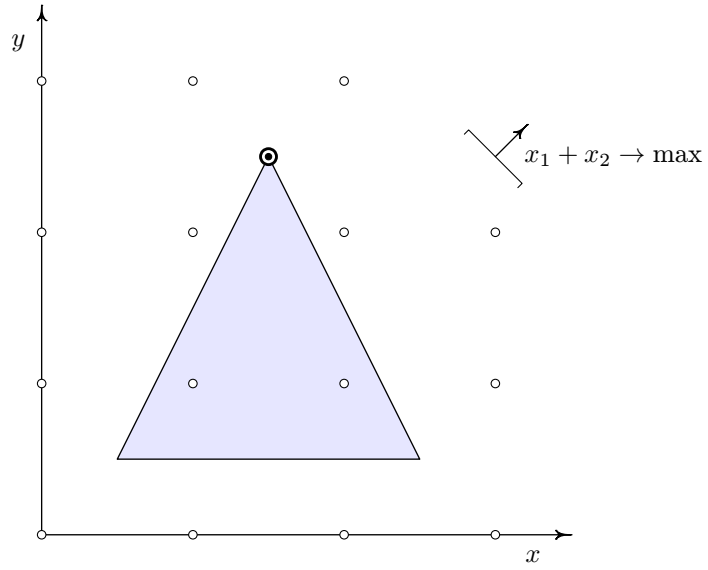
Ze seznamu živých podúloh vybíráme zpravidla tu, která má nejlepší odhad. Tuto podúlohu ze seznamu odstraníme, rozvětvíme ji na podúlohy, pro každou z nich ihned vypočteme odhad a případně ji zařadíme do seznamu živých podúloh, jak bylo popsáno výše.

Tento postup opakujeme, dokud je v seznamu nějaká živá podúloha. Výsledné řešení $\check{r}$ pak je optimálním řešením úlohy.

4.3.7 Příklad řešení úlohy ILP metodou větví a mezí

Řešme úlohu[1]

$$\begin{aligned} x_1 + x_2 &\to \max \\ 2x_1 + x_2 &\leqq 5\,\tfrac{1}{2} \\ 2x_1 - x_2 &\geqq \tfrac{1}{2} \\ x_2 &\geqq \tfrac{1}{2} \\ x_1, x_2 &\geqq 0 \\ x_1, x_2 &\text{ celočíselné} \end{aligned}$$



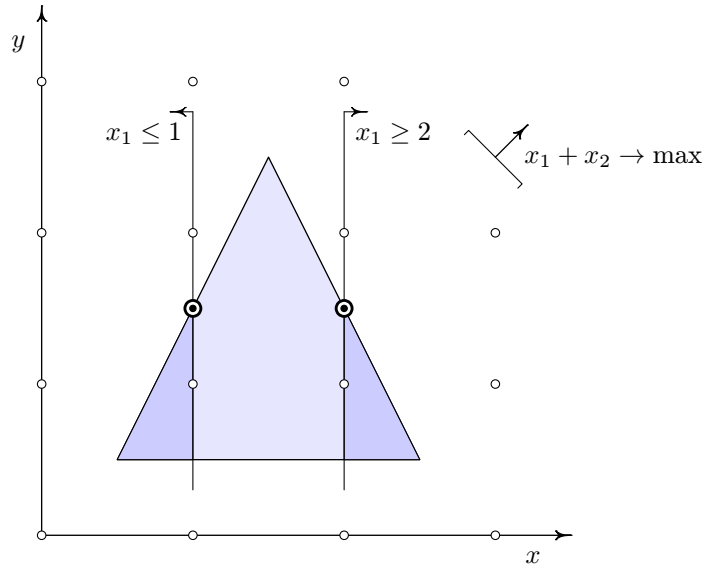
Obrázek 4.1: Řešení relaxované úlohy 4.3.7

Množina přípustných řešení relaxované úlohy je na obrázku 4.1. Optimální řešení $(1\,\tfrac{1}{2}, 2\,\tfrac{1}{2})$ relaxované úlohy má neceločíselné obě souřadnice. K větvení na podúlohy si vybereme souřadnici x_1, přidáme tedy omezující podmínky $x_1 \leqq 1$ a $x_1 \geqq 2$. Vzniklé podúlohy a optimální řešení jejich relaxovaných verzí jsou na obrázku 4.2.

číslo podúlohy	přidaná omezení	odhad	komentář
1	—	4	větveno na podúlohy 2 a 3
2	$x_1 \leqq 1$	$2\,\tfrac{1}{2}$	živá
3	$x_1 \geqq 2$	$3\,\tfrac{1}{2}$	živá

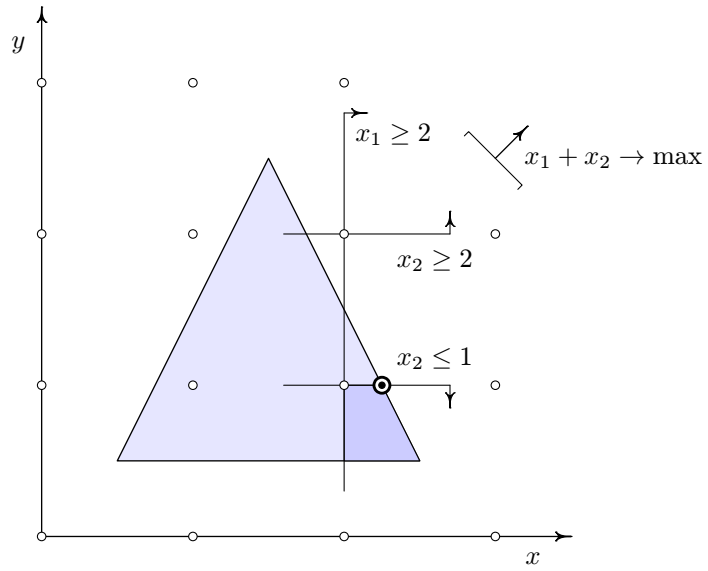
Z obou živých podúloh vezmeme k dalšímu větvení podúlohu 3, neboť má lepší (nadějnější) odhad. Větvíme podle proměnné x_2. Po rozvětvení vznikne situace v následující tabulce a znázorněná obrázkem 4.3.

[1] Příklad převzat z knihy Papadimitriou, Steiglitz: Cmbinatorial Optimization



Obrázek 4.2: Podúlohy úlohy 4.3.7 po prvém větvení.

číslo podúlohy	přidaná omezení	řešení	odhad	komentář
1	—	$(1\,1/2, 2\,1/2)$	4	větveno na podúlohy 2 a 3
2	$x_1 \leqq 1$	$(1, 1\,1/2)$	$2\,1/2$	živá
3	$x_1 \geqq 2$	$(2, 1\,1/2)$	$3\,1/2$	větveno na podúlohy 4 a 5
4	$x_1 \geqq 2$, $x_2 \leqq 1$	$(2\,1/4, 1)$	$3\,1/4$	živá
5	$x_1 \geqq 2$, $x_2 \geqq 2$	—	—	nemá přípustné řešení



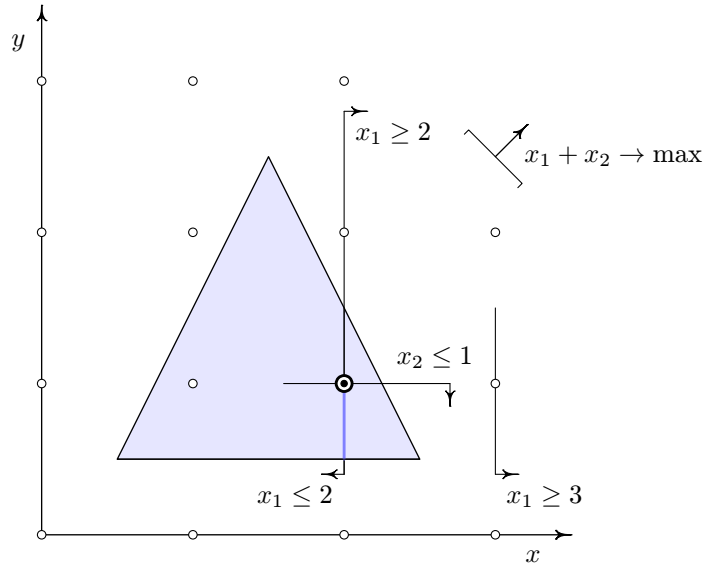
Obrázek 4.3: Podúlohy úlohy 4.3.7 po druhém větvení.

Opět máme dvě živé podúlohy, k dalšímu větvení opět vybereme nadějnější z nich, tedy podúlohu 4 a větvíme podle proměnné x_1.

Podúloha 6 vznikne přidáním podmínky $x_1 \leqq 2$. Množina přípustných řešení relaxované podúlohy se tím de facto redukuje na úsečku. Optimální řešení této relaxované podúlohy je $(2, 1)$, tedy celočíselné řešení. Hodnotu 3 účelové funkce tohoto řešení nyní můžeme použít k umrtvení živých podúloh, které mají horší hodnotu odhadu. V našem případě takto umrtvíme podúlohu 2.

Podúloha 7 vznikne přidáním podmínky $x_1 \geqq 3$ a tato podúloha nemá žádné přípustné řešení.

číslo podúlohy	přidaná omezení	řešení	odhad	komentář
1	—	$(1\,\frac{1}{2}, 2\,\frac{1}{2})$	4	větveno na podúlohy 2 a 3
2	$x_1 \leqq 1$	$(1, 1\,\frac{1}{2})$	$2\,\frac{1}{2}$	umrtveno díky odhadu
3	$x_1 \geqq 2$	$(2, 1\,\frac{1}{2})$	$3\,\frac{1}{2}$	větveno na podúlohy 4 a 5
4	$x_1 \geqq 2$, $x_2 \leqq 1$	$(2\,\frac{1}{4}, 1)$	$3\,\frac{1}{4}$	větveno na podúlohy 6 a 7
5	$x_1 \geqq 2$, $x_2 \geqq 2$	—	—	nemá přípustné řešení
6	$x_1 \geqq 2$, $x_2 \leqq 1$, $x_1 \leqq 2$	$(2, 1)$	3	celočíselné optimum
7	$x_1 \geqq 2$, $x_2 \leqq 1$, $x_1 \geqq 3$	—	—	nemá přípustné řešení



Obrázek 4.4: Podúlohy úlohy 4.3.7 po třetím větvení.

4.3.8 Problém batohu. Lupič má batoh, do kterého může dát nejvýše K kilogramů kořisti a ani o gram více. Jeho lup se skládá z n předmětů o hmotnosti $w_1, w_2, \ldots, w_n$ kilogramů a cenách $c_1, c_2, \ldots, c_n$ Kč. Snahou lupiče je odnést v batohu co nejdražší kořist, ale nesmí batoh přetížit.

Tato úloha má mnoho poctivějších podob, zejména při sestavování rozvrhů a při plánování paralelně probíhajících procesů, které spotřebovávají nějaký materiál, energii nebo práci. Vždy se snažíme co nejužitečněji vytížit daný materiálový, energetický nebo lidský zdroj.

Mějme batoh o kapacitě 40 a pět předmětů s těmito vahami a cenami:

předmět i:	A	B	C	D	E
cena c_i:	50	107	31	31	69
váha v_i:	14	30	9	10	27

Jako prostor řešení vezmeme $\mathbb{R}^5$, jeho prvky jsou uspořádané pětice čísel $x_1, \ldots, x_5$, která určují, kolikrát zabalíme do batohu jednotlivé předměty. Omezující podmínky určují, že $0 \leqq x_i \leqq 1$

a $\sum_{i=1}^{5} v_i x_i \leqq 40$. Nepříjemnou omezující podmínkou zde je, že všechna x_i musí být navíc celočíselná.

Pro účely snadného řešení relaxované úlohy si předměty seřadíme sestupně podle jejich „měrné ceny“, tj. podle podílu cena/váha. Ve výše uvedeném zadání je toto seřazení již provedeno.

Optimální řešení relaxované úlohy pak získáme velmi jednoduše tak, že do batohu budeme ukládat celé předměty v pořadí klesající měrné ceny, přičemž poslední z takto zařazených předmětů možná nebude zařazen celý. Pro naši úlohu tak dostaneme řešení o hodnotě 142,73 tvořené celým předmětem A a částí 26/30 předmětu B. Jako horní odhad použijeme celou část této hodnoty, tj. zde 142, neboť všechna přípustná řešení původní (nerelaxované) úlohy U mají hodnotu účelové funkce celočíselnou, a ta tedy nemůže přesáhnout 142.

Větvení budeme provádět vždy na dvě podúlohy, a to tak, že si vybereme některý předmět, o němž v dané podúloze není ještě rozhodnuto, a v jedné z podúloh jej pevně zařadíme, zatímco ve druhé jej zakážeme. V obou podúlohách se tedy rozhoduje o menším počtu předmětů. Přidané omezující podmínky, tj. příkazy a zákazy, budeme zkráceně označovat symboly + a - tak, že např. `-+-..` bude znamenat, že 1. a 3. předmět jsou zakázány, 2. předmět je přikázán a o ostatních předmětech není v podúloze rozhodnuto.

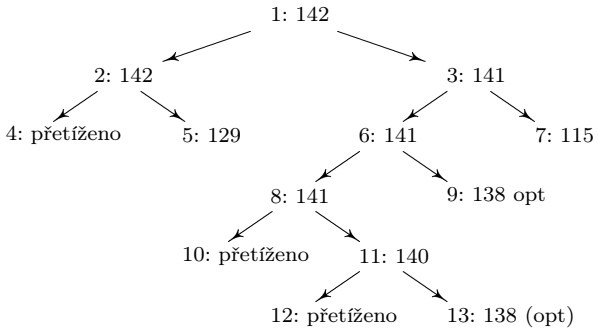
Průběh řešení je zaznamenán v následující tabulce. Podúlohy jsou očíslovány pořadovými čísly v pořadí, jak byly vytvořeny větvením. K větvení byla vybírána vždy podúloha s nejlepším odhadem, a to v pořadí 1, 2, 3, 6, 8, 11.

číslo podúlohy	přidaná omezení	horní odhad	komentář
1	`.....`	142	větveno podle A na podúlohy 2 a 3
2	`+....`	142	větveno podle B na podúlohy 4 a 5
3	`-....`	141	větveno podle B na podúlohy 6 a 7
4	`++...`	–	přetíženo
5	`+-...`	129	umrtveno po zpracování podúlohy 9
6	`-+...`	141	větveno podle C na podúlohy 8 a 9
7	`--...`	115	umrtveno po zpracování podúlohy 9
8	`-++..`	141	větveno podle D na podúlohy 10 a 11
9	`-+-..`	138	(optimum) viz komentář v textu
10	`-+++.`	–	přetíženo
11	`-++-.`	140	větveno podle E na podúlohy 12 a 13
12	`-++-+`	–	přetíženo
13	`-++--`	138	další optimum

Strom řešení je na obrázku 4.5.

Podúloha 9 vyžaduje podrobnější komentář: Řešením relaxované verze podúlohy 9 jsme zde (náhodou) dostali celočíselné řešení. Získali jsme tedy přípustné řešení celé úlohy, aniž by bylo nutno podúlohu větvit. V té době to bylo první přípustné řešení, které jsme nalezli. Použili jsme je ihned k umrtvení podúloh 5 a 7, které v té době byly ještě živé, ale měly ve srovnání s podúlohou 9 horší odhad.

Stojí za zmínku, že v této chvíli jsme již měli řešení, o kterém se později ukázalo, že je optimální, ale v té době jsme to ještě nevěděli, protože stále ještě zbývala živá podúloha 8 s odhadem, který dával naději na zlepšení. Bylo tedy nutno ve výpočtu pokračovat a teprve po rozvětvení podúloh 8 a 11 se ukázalo, že lepší řešení neexistuje.



Obrázek 4.5: Strom řešení k příkladu 4.3.8. U vrcholů jsou napsána čísla podúloh a hodnoty horních odhadů.

4.4 Backtracking

4.4.1 Posloupnosti a strom řešení. Backtracking lze použít v těch situacích, kdy každé řešení úlohy (tj. každý prvek prostoru řešení) můžeme považovat za konečnou posloupnost, přičemž každý prvek posloupnosti musí být vybrán z nějaké předem dané konečné množiny. Při troše fantazie se na většinu kombinatorických úloh můžeme dívat tímto způsobem.

Kromě posloupností, které představují (úplná) řešení úlohy budeme uvažovat také všechny jejich počáteční úseky. Tyto počáteční úseky odpovídají částečným (neúplně určeným) řešením. Všechny takovéto posloupnosti můžeme pokládat za vrcholy orientovaného grafu, v němž každá částečná posloupnost má jako své následníky všechna svá prodloužení o jeden prvek. Tento graf je kořenovým stromem (kořenem je prázdná posloupnost) a nazýváme jej *stromem řešení*.

4.4.2 Backtracking. Nejjednodušší verze backtrackingu spočívá v prohledávání stromu řešení do hloubky. Vždy, když se při prohledávání dostaneme k posloupnosti, která odpovídá úplnému řešení, provedeme test, zda toto řešení je přípustné. U optimalizační úlohy navíc evidujeme nejlepší dosud nalezené přípustné řešení.

Takto jednoduchý backtracking bez použití dalších triků ovšem není v podstatě ničím jiným než systematicky provedenou metodou hrubé síly. Triky, kterými lze výpočet mnohdy i značně urychlit, spočívají v tom, že vynecháme prohledávání některých podstromů.

Má smysl prodlužovat jen ty posloupnosti, u nichž je naděje, že mohou být prodlouženy na přípustné řešení. Pokud jsme tedy schopni s jistotou poznat, že žádné prodloužení už nemůže být přípustné, můžeme ušetřit čas tím, že prohledávání příslušného podstromu přeskočíme (provedením návratu).

Řešíme-li optimalizační úlohu, můžeme dosáhnout další úspory času, budeme-li schopni pro každou posloupnost $x_1, \ldots, x_i$ určit číslo D, které nazveme dolním, popř. horním *odhadem* (pro minimalizační, popř. maximalizační úlohu) a které má tu vlastnost, že prodlužováním posloupnosti $x_1, \ldots, x_i$ nelze dosáhnout přípustného řešení lepšího než D. Bude-li tedy pro posloupnost $x_1, \ldots, x_i$ tento odhad horší než nějaké přípustné řešení, které už v té chvíli známe, nemá další prodlužování posloupnosti $x_1, \ldots, x_i$ smysl a prohledávání příslušného podstromu můžeme opět přeskočit.

I přes uvedené možnosti úspor bývá backtracking časově hodně náročný.

4.4.3 Příklad – problém batohu. Předměty libovolně, ale pevně seřadíme do posloupnosti. Může to být pořadí podle měrné ceny jako v 4.3.8, ale není to nutné. Do batohu budeme ve zvoleném pořadí přidávat předměty, dokud se vejdou. Když se předmět, který je na řadě nevejde, vezmeme další. Když už není co přidávat, vyndáme naposledy přidaný předmět a zkusíme přidávat předměty, které následují za ním.

Takto postupně vyzkoušíme všechna přípustná řešení, tj. všechny množiny předmětů, které

batoh nepřetíží. Pokud přitom budeme zaznamenávat největší dosaženou cenu předmětů v batohu, najdeme tímto způsobem optimální řešení.

Vyzkoušejte to na příkladě 4.3.8 jako cvičení.

4.4.4 Backtracking versus větve a meze. Obě metody mají některé rysy společné a u některých algoritmů je těžké říci, o kterou z obou metod jde.

Backtracking použitý k řešení optimalizační úlohy lze pokládat za speciální případ metody větví a mezí. Na posloupnost, která v backtrackingu tvoří částečné řešení, se totiž můžeme dívat jako na podúlohu, v níž několik prvých členů posloupnosti je pomocí přidaných omezujících podmínek pevně určeno. Prodloužení posloupnosti pak znamená přidání nové omezující podmínky.

Při backtrackingu však prohledáváme strom řešení vždy do hloubky (jinak by to nebyl backtracking), zatímco v metodě větví a mezí lze další postup řešení volit svobodněji, a tedy zpravidla výhodněji (např. pomocí priorit, viz 4.3.5). V backtrackingu lze k zamezení zbytečného větvení použít i jiné triky než dolní, popř. horní odhady. A konečně, backtracking lze použít také k řešení úloh, které nemají optimalizační charakter.

4.5 Heuristické algoritmy

V praxi mnohdy vystačíme s řešením, které není nutně optimální, ale je je „dostatečně dobré“, které však získáme rychle. Algoritmy, které poskytují takováto řešení, nazýváme *heuristické*. Fungují bez záruky, nebo (méně často) jen s omezenou zárukou. Tyto algoritmy bývají založeny na nejrůznějších principech. Zmíníme se stručně o nejdůležitějších z nich.

4.5.1 Heuristiky hladového typu. Mnohé heuristické algoritmy pracují tzv. *hladovým* způsobem: řešení vytváří (konstruují) postupně, přičemž v každém kroku se (chamtivě) snaží o co největší zlepšení účelové funkce, popř. o co největší úsporu.

Například při řešení problému batohu 4.3.8 budeme dávat do batohu vždy předmět, který má nejlepší měrnou cenu a do batohu se celý vejde. Samozřejmě se může stát, že se „chamtivost nevyplatí“ a výsledkem je řešení, které není optimální. Příkladem je právě instance úlohy batohu řešená v 4.3.8 (vyzkoušejte).

Dalším příkladem heuristiky hladového typu ne indexová metoda řešení dopravní úlohy 3.2.3, str. 50.

Do třetice je třeba zmínit tzv. hladový algoritmus pro hledání minimální kostry grafu 8.5.9, str. 100, který také funguje hladovým způsobem, ale na rozdíl od předchozích dvou příkladů poskytuje zaručeně (dokazatelně) optimální řešení. Úlohy, v nichž hladový postup zaručeně vede k optimálnímu řešení, jsou ovšem velmi výjimečné.

4.5.2 Heuristiky využívající náhodu. Některé heuristické algoritmy řešení mnohokrát opakují s využitím náhody a z takto získaných řešení vybírají nejlepší.

4.5.3 Lokální průzkum. Další heuristické algoritmy využívají tzv. *lokální průzkum*: vezmou nějaké stávající řešení (zpravidla získané nějakou jinou heuristikou) a systematicky se je pokouší toto řešeení pozměnit, tj. v jistém smyslu prozkoumávají okolí stávajícího řešení. Pokud některé z pozměněných řešení je lepší než řešení stávající, prohlásí toto lepší řešení za stávající a pokračují průzkumem okolí tohoto nového řešení. Toto se opakuje, dokud se daří řešení zlepšovat. Samozřejmě můžeme výpočet ukončit i dříve, pokud najdeme řešení, jehož kvalita nám postačuje nebo pokud už nemáme čas na další zlepšování.

Metodu lokálního průzkumu lze dále zdokonalit s využitím náhody. Jednak můžeme s nějakou pravděpodobností akceptovat i zhoršení účelové funkce v naději, že takto později objevíme celkově lepší řešení, jednak můžeme celý postup mnohokrát opakovat z jiného, náhodně zvoleného výchozího řešení.

4.5.4 Genetické algoritmy jsou inspirovány přirozeným výběrem, který funguje v přírodě.

Na rozdíl od lokálního průzkumu genetické algoritmy sledují nikoli jedno řešení, ale nějakou jejich množinu, tzv. populaci. Z jedinců (tj. řešení) obsažených v populaci se opakovaně vytvářejí noví jedinci jednak křížením (tj. zkombinováním několika jedinců), jednak mutacemi (tj. pozměněním jednoho řešení) a jedinci, kteří se jeví jako neperspektivní se z populace vyřazují. Po dostatečném opakování tohoto postupu se nakonec z populace vybere jedinec s nejlepší účelovou funkcí.

4.5.5 Heuristiky se šijí na míru typu úlohy a často i typickým instancím, které mají být heuristikou řešeny. Jejich úspěšnost závisí na tom, jak se podaří využít specifických rysů daného typu úlohy i specifických rysů instancí, které chceme řešit. Při jejich návrhu hodně záleží i na zkušenosti a citu pro řešený problém.

4.5.6 Příklad – dělení tyčového materiálu. Pro úlohu dělení tyčového materiálu, kterou jsme zmínili v 4.1.5, str. 60, je známa řada dobře fungujících heuristik.

Pro připomenutí: Z tyčí o délce D máme za úkol nařezat p_i kusů o délce d_i pro $i = 1, \ldots, k$ a to tak, abychom použili co nejmenší počet celých tyčí.

First Fit: Vezmeme požadovanou délku d_i a uřízneme ji z první tyče, ze které tuto délku lze uříznout. Pořadí uřezávaných délek d_i není určeno.

First Fit Decreasing: Jako First Fit, ale požadované délky d_i řežeme v pořadí od nejdelší po nejkratší.

Best Fit Vezmeme požadovanou délku d_i a uřízneme ji z tyče, ze které po uřezání zbude nejmenší zbytek. Pořadí uřezávaných délek d_i není určeno.

Best Fit Decreasing: Jako Best Fit, ale požadované délky d_i řežeme v pořadí od nejdelší po nejkratší.

Je dokázáno, že označíme-li M optimální hodnotu účelové funkce, tedy počet rozřezaných tyčí, pak výsledky heuristiky First Fit nejsou horší než $1.7M$ a výsledky heuristiky First Fit Decreasing nejsou horší než $11/9M + 6/9$, tedy ne více než o 22 % horší než optimum. To je docela optimistické sdělení: ani v nejhorším myslitelném případě nebudou výsledky těchto heuristik nijak špatné. V typickém případě tyto heuristiky fungují velmi dobře, průměrná odchylka od optima bývá mnohem menší.

Kapitola 5

Dynamické programování

Dynamické programování je poměrně obecným a účinným nástrojem k řešení *některých* optimalizačních úloh.

5.1 Bellmanův princip optimality

Dynamické programování je založeno na myšlence, že část optimálního řešení by měla být také optimálním řešením (menší úlohy). Například část nejkratší cesty by měla být také nejkratší cestou.

Vycházíme přitom z předpokladu, že řešení optimalizační úlohy lze rozložit na části, které lze pokládat za řešení optimalizačních úloh stejného typu, ale menšího rozsahu. Pokud lze řešení dílčích úloh spolu libovolně kombinovat, pak celkové řešení nemůže být optimální, budeme-li schopni některou jeho část „lokálně“ vylepšit.

Dynamické programování se nejčastěji aplikuje na úlohy, jejichž řešení si lze představit jako posloupnost rozhodnutí o dalších a dalších částech řešení. Rozhodujeme se dynamicky vždy podle situace, do které jsme se dostali na základě předchozích rozhodnutí. Odtud ostatně pochází název metody.

5.1.1 Bellmanův princip optimality.

> Optimální strategie má tuto vlastnost: ať je výchozí stav a výchozí rozhodnutí jakékoli, posloupnost následujících rozhodnutí musí tvořit optimální strategii vzhledem ke stavu, který vyplývá z výchozího rozhodnutí.

5.1.2 Podmínky platnosti Bellmanova principu optimality. Pokud se při rozhodování musíme řídit nejen současným stavem, ale i způsobem, jak jsme se do tohoto stavu dostali, pak Bellmanův princip optimality ve výše zmíněné jednoduché podobě nelze použít. Aby jej použít šlo, je nutno zahrnout do pojmu „stav“ vše, co ovliňuje možnosti dalšího pokračování.

5.2 Příklad: alokace investic

5.2.1 Úloha o alokaci investic. Máme Q jednotek nějakého ekonomického zdroje (třeba peněz) a naším úkolem je rozdělit je mezi n investičních programů. Pro každý investiční program i a pro každou celočíselnou velikost investice $0 \leqq x \leqq Q$ známe velikost výnosu $g_i(x)$. Celkový výnos je součtem výnosů ze všech n programů. Úkolem je najít velikosti investic $x_1, \ldots, x_n$ do jednotlivých programů tak, aby celkový výnos byl maximální možný.

Předpokládáme, že investice do jednotlivých programů jsou vzájemně nezávislé a také, že výnosy z jednotlivých programů nezávisí na investicích do ostatních programů.

5.2.2 Obecné řešení úlohy o alokaci investic. Označme

$F_j(q)$ = optimální (tj. nejvyšší možný) výnos z programů $1, \ldots, j$ při celkové velikosti investic q.

$d_j(q)$ = velikost x_j investice do j-tého programu, při níž celkový výnos z programů $1, \ldots, j$ nabývá maximální hodnoty $F_j(q)$.

Platí

$$F_1(q) = g_1(q)\,, \tag{5.1}$$

$$F_j(q) = \max_{0 \leqq x \leqq q} \{g_j(x) + F_{j-1}(q-x)\} \tag{5.2}$$

Podle vzorců (5.1) a (5.2) lze počítat postupně F_1, F_2, F_3, …, F_n. Je vhodné současně zaznamenávat hodnoty d_j, protože s jejich pomocí lze pak vypočítat hodnoty $x_1, \ldots, x_n$.

5.2.3 Proč tak složitě? Nešlo by jednoduše vyzkoušet všechny možnosti? Je jich přece konečně mnoho.

Samozřejmě by to šlo. Bylo by to dokonce velmi jednoduché, ale pro úlohy většího rozměru také nesmírně pracné. Počet všech možností, kterými lze investovat q peněz do n programů, je totiž roven[1]

$$\binom{n+Q-1}{Q} = \frac{(n+Q-1)!}{Q!(n-1)!}$$

a pro každou z nich je třeba provést $(n-1)$ sečítání.

Např. pro $Q = 50$ a $n = 25$ je počet všech možností přibližně $1.75 \cdot 10^{19}$ a celý výpočet by vyžadoval zhruba $8.6 \cdot 10^{20}$ sečítání. Při rychlosti miliarda operací za vteřinu by výpočet trval přes 7 let.

Výpočet pomocí dynamického programování je sice komplikovanější, ale mnohem méně pracný. Počítáme $Q(n-1)$ hodnot $F_j(q)$, výpočet každé z nich vyžaduje v průměru $Q/2$ sečítání a porovnání dvou čísel. Celkem tedy nejvýše $Q^2(n-1)/2$ sečítání a porovnání.

Pro $Q = 50$ a $n = 25$ potřebujeme pouze asi 30 tisíc operací sečítání a porovnání, což srovnatelně rychlý počítač zvládne během zlomku vteřiny.

5.2.4 Příklad. Řešme úlohu o alokaci investic, kde $Q = 6$ a výnosy jsou dány touto tabulkou:

x	g_1	g_2	g_3	g_4
0	0	0	0	0
1	4	15	5	10
2	6	16	10	14
3	12	17	15	18
4	24	30	20	22
5	30	35	25	30
6	35	40	30	40

Podle vzorce (5.1) získáme nejprve $F_1 = g_1$ a pak podle vzorce (5.2) vypočteme nejprve F_2, pak z něj F_3 a nakonec hodnotu $F_4(6)$.

[1] Tento vzorec lze odvodit např. touto úvahou: Řešení si představíme jako proces. Začneme u prvního programu a provedeme celkem $(n+Q-1)$ dílčích kroků, kdy buď zvětšíme o jednotku investici do stávajícího programu a nebo přejdeme na další program. Zvětšení o jednotku provedeme celkem Q-krát, přechod k dalšímu programu celkem $(n-1)$-krát. Množinu Q operací typu zvětšení investice o jednotku lze v celé posloupnosti všech $(n+Q-1)$ operací rozmístit celkem $\binom{n+Q-1}{Q}$ způsoby.

q	g_1	g_2	g_3	g_4	F_1	F_2	F_3	F_4
0	0	0	0	0	0_0	0_0	0_0	
1	4	15	5	10	4_1	15_1	15_0	
2	6	16	10	14	6_2	19_1	20_1	
3	12	17	15	18	12_3	21_1	25_2	
4	24	30	20	22	24_4	30_4	$30_{0,5}$	
5	30	35	25	30	30_5	39_1	39_0	
6	35	40	30	40	35_6	45_1	45_0	49_1

Jako indexy jsou zde uváděny hodnoty $d_j(q)$, které použijeme pro zpětný výpočet hodnot x_i:

Poněvadž $d_4(6) = 1$, bude v optimálním řešení $x_4 = 1$. Pro investice do programů $1, \ldots, 3$ tedy zbude 5 jednotek. Z prvých tří programů tedy dostaneme výnos $F_3(5) = 39$, přičemž třetí program dostane nulovou investici, neboť $d_3(5) = 0$. Odtud $x_3 = 0$. Pro investice do prvých dvou programů tedy zbude 5 jednotek. Z prvých dvou programů tedy dostaneme výnos $F_2(5) = 39$, přičemž druhý program dostane investici ve výši $d_2(5) = 1$, tedy $x_2 = 1$. Pro investici do prvého programu tedy zbudou 4 jednotky, tedy $x_1 = 4$.

Vskutku, $g_1(4) + g_2(1) + g_3(0) + g_4(1) = 49$.

5.3 Optimální doplňování zásob

5.3.1 Optimální doplňování zásob. Předpokládejme, že poptávka je předem přesně známa a to tak, že v n okamžicích $t_1, \ldots, t_n$ bude mít poptávka velikost $q_1, \ldots, q_n$. Náklady na skladování jsou C_s za jednotku času a množství, náklady spojené s pořízením zásoby jsou C_p. Úkolem je naplánovat optimální doplňování zásob tak, aby celkové náklady spojené s doplňováním a udržováním zásoby byly minimální.

5.3.2 Řešení úlohy o optimálním doplňování zásob. Ještě než začneme počítat, provedeme několik přípravných úvah, kterými zredukujeme počet možností, kterými má smysl se zabývat.

Především omezíme okamžiky, kdy má smysl pořizovat zásoby. Zásobu nemá smysl doplňovat jindy než v okamžicích z množiny $T = \{t_1, \ldots, t_n\}$. Kdybychom totiž zásobu doplnili v okamžiku t, který není v této množině, mohli bychom celé řešení zlevnit tím, že bychom doplnění zásoby odložili na nejbližší příští okamžik z množiny T.

Dále omezíme různost množství, které budeme skladovat. Velikost zásoby by v každém okamžiku měla být buď nulová a nebo rovna součtu jedné nebo několika následujících poptávek. Kdyby totiž velikost zásoby v nějakém okamžiku byla jiná, bylo by možno zmenšit velikost předchozí dodávky do skladu a tím zmenšit náklady na skladování.

Je tedy zřejmé, že v optimálním řešení musí být první dodávka do skladu v čase t_1 a další dodávky musí být v některých (možná žádných, možná všech) okamžicích $t_2, \ldots, t_n$. Výběrem této množiny okamžiků je pak již celé řešení jednoznačně určeno. Velikost každé dodávky totiž musí být právě taková, aby stačila na krytí poptávky do nejbližší další dodávky popř. až do okamžiku poslední poptávky t_n.

Všimněte si, že jsme velmi podstatně zredukovali počet řešení, mezi kterými hledáme optimum. Použili jsme k tomu myšlenky dynamického programování, aniž jsme prozatím cokoli počítali.

Počet řešení, ze kterých vybíráme optimum, je však i po této redukci roven počtu podmnožin $(n-1)$-prvkové množiny, tedy $2^{(n-1)}$, což je pořád příliš mnoho. Dynamické programování nám opět pomůže, ale teď již budeme muset trochu počítat.

Označme F_j náklady na krytí poptávky v období $t_1, \ldots, t_j$ včetně. Zřejmě $F_1 = C_p$. A aby nám další úvahy pěkně vycházely, budeme brát $F(0) = 0$.

Platí

$$F_j = \min_{i \leqq j} \{F_{i-1} + N(i,j)\} \tag{5.3}$$

kde $N(i,j)$ jsou náklady na jedno doplnění zásob v čase t_i a udržování zásoby do okamžiku t_j včetně. Platí

$$N(i,j) = C_p + C_s \sum_{k=i}^{j} q_k(t_k - t_i) \tag{5.4}$$

Hodnoty $N(i,j)$ pro $i \leqq j$ je vhodné vypočítat předem. Přitom je vhodné postupovat po řádcích, neboť každou následující hodnotu $N(i,j+1)$ dostaneme snadno tak, že k předchozí hodnotě $N(i,j)$ přičteme přírůstek $C_s q_{j+1}(t_{j+1} - t_i)$.

Obvykle není třeba počítat všechny hodnoty $N(i,j)$ pro $i \leqq j$. Je-li totiž přírůstek větší než náklady C_p na pořízení nové zásoby, je zřejmé, že výsledná hodnota $N(i,j+1)$ nemůže být součástí optimálního řešení, neboť nové pořízení zásoby by bylo levnější než onen přírůstek. V takovém případě pak nemá smysl počítat ani následující hodnoty až do konce řádky matice N.

V další fázi výpočtu pak postupně podle vzorce (5.3) počítáme hodnoty F_j pro $j = 1, \ldots, n$.

Cílem výpočtu ovšem není jen hodnota účelové funkce F_n, ale zejména časy a velikosti jednotlivých objednávek. Pro tento účel je třeba při výpočtu hodnot F_j podle vzorce (5.3) zaznamenávat, při kterém i bylo dosaženo minima.

Ve třetí fázi výpočtu tedy zjišťujeme časy jednotlivých objednávek. Postupujeme přitom pozpátku, od poslední objednávky k první. Z časů objednávek pak již snadno zjistíme jejich velikosti.

5.3.3 Příklad. Řešme úlohu o optimálním doplňování zásob s těmito daty: Náklady na pořízení zásoby jsou $C_p = 70$, náklady na skladování jsou $C_s = 1$. Poptávka je dána touto tabulkou:

i	t_i	q_i
1	0	40
2	1	50
3	2	30
4	3	20
5	4	20
6	6	15
7	8	10

Nejprve vypočteme hodnoty $N(i,j)$.

	1	2	3	4	5	6	7
1	70	120	180	240	–	–	–
2		70	100	140	200	–	–
3			70	90	130	190	250
4				70	90	135	185
5					70	100	140
6						70	90
7							70

Ve druhé fázi vypočteme hodnoty F_j:

$$\begin{aligned}
F(0) &= 0 \\
F(1) &= 70 \\
F(2) &= \min \begin{cases} F_1 + N(2,2) = 70 + 70 = 140 \\ F_0 + N(1,2) = 0 + 120 = \underline{120} \end{cases} \\
F(3) &= \min \begin{cases} F_2 + N(3,3) = 120 + 70 = 190 \\ F_1 + N(2,3) = 70 + 100 = \underline{170} \\ F_0 + N(1,3) = 0 + 180 = 180 \end{cases}
\end{aligned}$$

$$
\begin{aligned}
F(4) &= \min \begin{cases} F_3 + N(4,4) = 170 + 70 = 240 \\ F_2 + N(3,4) = 120 + 90 = \underline{210} \\ F_1 + N(2,4) = 70 + 140 = \underline{210} \\ F_0 + N(1,4) = 0 + 240 = 240 \end{cases} \\
F(5) &= \min \begin{cases} F_4 + N(5,5) = 210 + 70 = 280 \\ F_3 + N(4,5) = 170 + 90 = 260 \\ F_2 + N(3,5) = 100 + 130 = \underline{250} \\ F_1 + N(2,5) = 70 + 200 = 270 \end{cases} \\
F(6) &= \min \begin{cases} F_5 + N(6,6) = 250 + 70 = 320 \\ F_4 + N(5,6) = 210 + 100 = 310 \\ F_3 + N(4,6) = 170 + 135 = \underline{305} \\ F_2 + N(3,6) = 120 + 190 = 310 \end{cases} \\
F(7) &= \min \begin{cases} F_6 + N(7,7) = 305 + 70 = 375 \\ F_5 + N(6,7) = 250 + 90 = \underline{340} \\ F_4 + N(5,7) = 210 + 140 = 350 \\ F_3 + N(4,7) = 170 + 185 = 355 \\ F_2 + N(3,7) = 120 + 250 = 370 \end{cases}
\end{aligned}
$$

Abychom dosáhli optimálních celkových nákladů $F_7 = 340$, musí poslední dodávka do skladu být v čase $t_6 = 6$. Předcházející období t_1 až t_5 musí být pokryto optimálním způsobem, tedy s náklady $F(5) = 250$ a s poslední dodávkou do skladu v čase $t_3 = 2$. Období t_1 až t_2 pak bude optimálně pokryto dodávkou v čase $t_1 = 0$.

Z časů dodávek pak odvodíme jejich velikosti:

čas	velikost objednávky	na období
$t_1 = 0$	90	t_1 až t_2
$t_3 = 2$	70	t_3 až t_5
$t_6 = 6$	25	t_6 až t_7

5.3.4 Poznámka o časovém horizontu. Velikost počáteční optimální objednávky v čase t_1 závisí na celé následující poptávce, tj. na poptávce ve všech následujících okamžicích. Změna i časově velmi vzdálené poptávky může ovlivnit počáteční rozhodnutí.

To znamená, že kdybychom omezili časový horizont a zanedbali (tj. ve výpočtu neuvažovali) časově vzdálenou poptávku, mohli bychom dostat neoptimální, tedy dražší řešení. Rozdíl v účelové funkci pak můžeme pokládat za cenu informace o budoucí poptávce.

Například kdyby se v předchozím příkladě zmenšila velikost poslední poptávky na $q_7 = 4$, změnil by se sice v matici N jen poslední sloupec

	1	2	3	4	5	6	7
1	70	120	180	240	–	–	–
2		70	100	140	200	–	–
3			70	90	130	190	214
4				70	90	135	155
5					70	100	116
6						70	78
7							70

Ve druhé fázi výpočtu se změní pouze hodnota F_7:

$$F(7) \quad = \quad \min \begin{cases} F_6 + N(7,7) = 305 + 70 = 375 \\ F_5 + N(6,7) = 250 + 78 = 328 \\ F_4 + N(5,7) = 210 + 116 = 326 \\ F_3 + N(4,7) = 170 + 155 = \underline{325} \\ F_2 + N(3,7) = 120 + 214 = 334 \end{cases}$$

Časy a velikosti objednávek se však změní velmi podstatně:

čas	velikost objednávky	na období
$t_1 = 0$	40	t_1
$t_2 = 1$	80	t_2 až t_3
$t_5 = 4$	65	t_4 až t_7

Kapitola 6

Teorie zásob

6.1 Základní pojmy

6.1.1 Zásoba je libovolný pohotový ekonomický zdroj, který je v nějakém čase k disposici a jehož spotřeba může být odložena do budoucnosti.

Teorie zásob se zabývá otázkou, jak stanovit a řídit velikost zásoby, aby byly minimalizovány náklady, které jsou důsledkem zvolené strategie řízení. O jednotlivých typech nákladů pojednáme dále v 6.1.4.

6.1.2 Poptávka se v teorii zásob chápe jako proces zmenšování zásoby. Příklady: poptávka na trhu, spotřeba materiálu pro výrobu, spotřeba náhradních dílů.

Poptávku zpravidla nemůžeme ovlivnit, často je náhodná a předem neznáma.

6.1.3 Doplňování zásob je proces, který vede ke zvyšování zásoby. Zásoba se doplňuje zpravidla skokově. Zpravidla rozlišujeme vystavení objednávky a vlastní dodávku, protože tyto dva úkony zpravidla nenastávají najednou.

Proces doplňování zásoby je to, čím můžeme ovlivňovat velikost zásoby a náklady se zásobami spojené.

Zjednodušeně lze říci, že základní otázky teorie zásob jsou

- kdy objednávat,
- kolik objednávat.

6.1.4 Typy nákladů. Teorie zásob se nezabývá tím, kde a za kolik zásobu nakoupíme, ale pouze tím, kdy a jak velkou zásobu si opatříme. Přitom rozlišujeme tyto typy nákladů:

Náklady na udržování zásoby C_s — jsou úměrné skladovanému množství a době skladování. Největší část těchto nákladů tvoří náklady z vázanosti peněz v zásobách. Tyto náklady závisí na možnostech alternativního zhodnocení peněz, které jsou jinak vázány v zásobách. Dále sem patří nákladu na samotné skladování, např. pronájem skladových prostor, ošetřování zásob, znehodnocení zásob, manipulace se zásobami, pojištění.

Náklady na doplnění zásoby C_d — zpravidla konstantní náklady spojené s jedním doplněním zásoby. Jde o náklady na přípravu a vystavení objednávky, na dopravu (je-li cena za dopravu nezávislá na velikosti objednávky), ale třeba i náklady na seřízení strojů na jinou výrobní šarži materiálu.

Náklady z nedostatku C_n — zboží, které nenáme pohotově na skaldě již odběratel nebude chtít, jde o ztráta příležitosti, např. prodeje. Tyto náklady jsou úměrné chybějícímu množství.

Náklady ze zpoždění C_z — zboží, které nemáme pohotově na skladě, dodáme později a utrpíme náklady úměrné chybějícímu množství a délce zpoždění. Příklady: penále, odložení výroby.

6.2 Náhodné dynamické modely

U náhodných modelů se poptávka pokládá za náhodný proces. Proces doplňování je zpravidla deterministický (zboží je dodáno vždy v objednaném množství a čase), ale může být také náhodný jak v čase tak i v množství.

U těchto modelů nejde o to stanovit čas a velikost jedné objednávky. Nelze ani předem naplánovat sérii objednávek. Cílem tedy je najít algoritmus (postup, strategii), jak pomocí objednávek řídit velikost zásoby.

V literatuře jsou popsány dva význačné způsoby řízení zásob:

6.2.1 P-systém řízení zásob spočívá v periodickém objednávání. Objednává se množství, které v okamžiku objednávky ve skladu chybí do jeho předepsané kapacity. Objednané množství zpravidla není dodáno okamžitě, takže v okamžiku dodávky velikost zásoby stoupne na hodnotu, která je obvykle nižší, než kapacita skladu.

Parametry, které mohou být předmětem optimalizace, jsou délka periody a kapacita skladu.

6.2.2 Q-systém řízení zásob odvozuje čas objednání od velikosti zásoby. Objednává se v okamžiku, kdy zásoba poklesne na tzv. *signální úroveň* (též *velikost pojistné zásoby*). Objednává se množství, které v okamžiku objednávky ve skladu chybí do jeho předepsané kapacity.

Parametry, které mohou být předmětem optimalizace, jsou velikost signální úrovně a kapacita skladu.

6.2.3 Srovnání P-systému a Q-systému řízení zásob. Oba systémy se do jisté míry adaptují na změny náhodného procesu poptávky, každý ovšem jinak.

Optimalizace parametrů P-systémů a Q-systémů je závislá na znalosti náhodného procesu poptávky. Často se používá simulace.

6.3 Jednorázová náhodná poptávka

6.3.1 Základní model. Předpokládáme, že velikost budoucí poptávky je náhodná veličina, označme ji X, s rozložením, které je dáno pravděpodobnostmi $p_k = P(X = k)$. Odtud lze odvodit ditribuční funkci $F(k) = \sum_{i=0}^{k} p_i$.

Velikost předem pořízené zásoby označme q.

Náklady spojené s přebytkem zásob označme C_p na jednotku přebytku. Podobně náklady spojené s nedostatkem zásob označme C_n na jednotku množství, které se nedostává. V obou těchto případech náklady uvažujeme vzhledem k optimální variantě, tj. o kolik více nás bude stát každý kus zásoby, kterou pořídíme navíc nebo která bude chybět.

Cílem je najít optimální velikost předem pořízené zásoby q tak, aby střední hodnota nákladů byla nejmenší.

6.3.2 Řešení. Celkové náklady jsou náhodnou veličinou, její střední hodnota $N_c(q)$ závisí na q a skládá se ze dvou složek, ze středních nákladů z přebytku a středních nákladů z nedostatku: Střední náklady z přebytku jsou

$$N_p(q) = C_p \sum_{x=0}^{q} (q - x) p_x$$

a střední náklady z nedostatku jsou

$$N_n(q) = C_n \sum_{x=q+1}^{\infty} (x-q)p_x$$

Hledáme nezáporné celé q, pro které funkce $N_c(q) = N_p(q) + N_n(q)$ nabývá minima.

Při změně velikosti zásoby z $q = i$ na hodnotu $q = i+1$ se $N_p(q)$ zvýší o $C_p \sum_{x=0}^{i} = C_p F(i)$ a $N_n(q)$ se sníží o $C_n \sum_{x=q+1}^{\infty} = C_n(1 - F(i))$, celkové střední náklady $N_c(q)$ tedy stoupnou o $(C_p - C_n)F(i) - C_n$.

Odtud plyne, že pro optimální velikost zásoby q musí platit

$$F(q-1) \leqq \frac{C_n}{C_p + C_n} \leqq F(q)$$

6.3.3 Jednorázový prodej. Zboží nakoupíme za cenu C_N, budeme je prodávat za cenu C_P a přebytek zásob prodáme ve výprodeji za cenu C_V.

Ve výše popsaném modelu jsou jednotkové náklady z přebytku $C_p = C_N - C_V$ (ztráta, se kterou prodáváme ve výprodeji) a náklady z nedostatku jsou $C_n = C_P - C_N$ (ušlý zisk).

Dodejme, že výprodejní cena C_V může být i záporná, pokud za likvidaci přebývajícího zboží musíme platit.

6.3.4 Zásoba náhradních dílů. Při pořizování speciálního stroje máme možnost si současně objednat zásobu náhradních dílů za cenu C_H za kus. Dodatečná výroba téhož jednotlivého náhradního dílu by stála C_1. Typicky $C_1 > C_H$.

Ve výše popsaném modelu jsou jednotkové náklady z přebytku $C_p = C_H$ (zbytečně vyrobený náhradní díl) a náklady z nedostatku jsou $C_n = C_1 - C_H$ (kolik jednotlivě vyrobený díl stojí navíc).

6.4 Konstantní poptávka

6.4.1 Základní model. Předpokládáme, že poptávka je v čase spojitá a má konstantní intenzitu r. Doplnění zásob je deterministické. Zásobu doplňujeme vždy v okamžiku, kdy její velikost klesne na nulu. Nedostatek zásob nepřipouštíme. Jednotkové náklady na udržování zásoby jsou C_s, náklady na pořízení jsou C_p na jednu objednávku. Úkolem je najít optimální časový interval t mezi objednávkami a optimální velikost dodávky q.

6.4.2 Řešení – Wilsonův vzorec. Uvažujme model ve velmi dlouhém časovém intervalu T. Celkové náklady závisí na intervalu t mezi objednávkami a jsou vyjádřeny vzorcem

$$N(t) = C_s T r t/2 + C_p T/t \ ,$$

kde první sčítanec vyjadřuje náklady na udržování zásoby (úměrné průměrné velikosti zásob $rt/2$ krát čas T) a druhý sčítanec náklady na objednávání (úměrné počtu objednávek T/t za dobu T).

Z rozboru průběhu této funkce plyne, že v intervalu $(0, infty)$ má všude derivaci a má zde jediné minimum, které lze zjistit pomocí nulové derivace. Derivace je $N'(t) = C_s T r/2 - C_p T/t^2$ a položíme-li ji rovnu nule, po jednoduchém výpočtu dostaneme, že optimální čas mezi objednávkami je

$$t = \sqrt{\frac{2C_p}{rC_s}} \ .$$

Optimální velikost dodávky pak je $q = tr$.

6.4.3 Několik současně objednávaných materiálů. Od základního modelu 6.4.1 se tento model liší pouze tím, že skladujeme n materiálů $i = 1, \dots, n$, poptávka po i-tém materiálu je konstantní s intenzitou r_i a jednotkové náklady spojené s udržováním zásoby i-tého materiálu jsou C_{si}.

Celkové náklady jsou

$$N(t) = T\frac{\sum_{i=1}^{n} C_{si} r_i t}{2} + C_p T/t \,,$$

a po velmi podobném výpočtu vyjde optimální interval mezi objednávkami

$$t = \sqrt{\frac{2C_p}{\sum_{i=1}^{n} r_i C_{si}}} \,.$$

6.4.4 Neceločíselná doba mezi objednávkami je zprvidla nepřijatelná. Z průběhu funkce $N(t)$ plyne, že optimálním řešením je nejbližší vyšší nebo nejbližší nižší celé číslo k optimální hodnotě t. Které z nich to bude, je třeba zjistit dosazením do funkce $N(t)$.

6.5 Deterministická diskrétní poptávka

Předpokládáme, že poptávka se bude realizovat v diskrétních časových okamžicích (tj. nikoli spojitě) a že časy a velikost poptávky přesně známe. Pak lze předem vypočítat jak optimálně doplňovat zásobu. Řešením se budeme zabývat v kapitole 5

Kapitola 7

Strukturní model

Strukturn model (někdy nazývaný po svém tvůrci Leontěvův strukturní model) je ve své základní podobě jedním z nejjednodušších modelů operačního výzkumu.

7.0.1 Předpoklady, z nichž vychází Leontěvův strukturní model, jsou poměrně silné.

Hospodářství se skládá z n odvětví. Každý produkt, kterým se model zabývá, je produkem přesně jednoho z těchto n odvětví.

Každé odvětví produkuje jediný produkt. Jinými slovy, mezi různými věcmi, které produkuje i-té odvětví, pro účely strukturního modelu nerozlišujeme a pokládáme je dohromady za jediný produkt i-tého odvětví. Velikost výroby i-tého odvětví budeme značit x_i. Velikosti výroby v jednotlivých odvětvích $x_1, \ldots, x_n$ dohromady pokládáme za (sloupcový) vektor, který označíme X a nazeme jej *vektorem výroby*.

Spotřeba i-tého produktu pro výrobu produktu j-tého je závislá pouze na velikosti výroby j--tého produktu a je přímo úměrná velikosti této výroby. Tedy čím více j-té odvětví vyrábí, tím více spotřebovává všeho, co spotřebovává, a tato závislost je lineární.

Tento nevinně vypadající předpoklad je ve skutečnosti velmi omezující a poněkud nerealistický. Nepřipouští totiž v žádném odvětví možnost nahradit některý spotřebovávaný produkt jiným produktem. Z tohoto předpokladu např. vyplývá, že snížíme-li výrobu elektřiny o 10%, pak spotřeba všeho, z čeho elektřinu vyrábíme, pokesne rovněž o 10%. Ve skutečnosti by se ovšem pokles spotřeby týkal zejména surovin, které jsou drahé, špatně dostupné nebo neekologické.

Konečně budeme předpokládat, že všechno, co se v modelovaném hospodářství vyrobí, bude spotřebováno a to buď pro výrobu v některém odvětví, nebo pro nějaký jiný účel, který nazveme *konečnou spotřebou*. Velikosti konečné spotřeby jednotlivých produktů budeme značit $y_1, \ldots, y_n$ a dohromady je budeme pokládat za (sloupcový) *vektor konečné spotřeby* Y.

7.0.2 Jednotky. Velikosti výroby a spotřeby jednotlivých odvětví můžeme vyjadřovat v jakýchkoli jednotkách, klidně pro každé odvětví jiných. Může jít o jednotky fyzické (tuny, litry, megawathodiny, atd.) nebo peněžní (vzhledem k cenám jednotlivých produktů). Jedinou nezbytnou podmínkou je, aby pro každé odvětví (a jeho produkt) byla v celém modelu používána vždy táž jednotka.

7.0.3 Matice technických koeficientů. Symbolem a_{ij} značíme spotřebu i-tého produktu na výrobu jednotkového množství produktu j-tého. Jde vlastně o koeficienty přímé úměrnosti mezi výrobou a výrobní spotřebou. Tyto koeficienty nazýváme *technickými koeficienty* nebo též *normami přímé spotřeby*.

Dohromnady tyto koeficienty tvoří matici, kterou značíme A a nazýváme *maticí technických koeficientů* nebo také *maticí norem přímé spotřeby*.

Koeficienty, které se týkají spotřeby v j-tém odvětví, najdete v j-tém sloupci.

Technické koeficienty lze určit ze statistiky, kolik které odvětví vyrobilo a kolik přitom spotřebovalo jednotlivých produktů.

Technické koeficienty jsou v podstatě určeny technologiemi, které se používají v jednotlivých odvětvích a proto se v čase příliš rychle nemění. Proto lze matici technických koeficientů, která odpovídá současnému stavu hospodářství, použít i k výpočtům, které se týkají nepříliš vzdálené budoucnosti.

7.0.4 Výrobní spotřeba. Celková spotřeba i-tého produktu pro výrobu ve všech odvětvích je rovna $\sum_{j=1}^{n} a_{ij}x_j$. V maticovém vyjádření je tedy vektor výrobní spotřeby roven součinu AX.

7.0.5 Uzavřený strukturní model je charakterizován předpokladem, že všechny vyrobené produkty budou beze zbytku spotřebovány pro výrobu. V uzavřeném modelu platí $X = AX$, tedy

$$(E - A)X = 0 \, .$$

Vektor výroby je řešením této soustavy lineárních rovnic. Tato soustava je homogenní (nulový vektor na pravé straně), proto její řešení není jednoznačné. Každý (nezáporný) násobek přípustného vektoru výroby je tedy opět přípustným vektorem výroby.

7.0.6 Otevřený strukturní model připouští i nenulovou konečnou spotřebu Y a je charakterisován rovnicí

$$X = AX + Y$$

nebo ekvivalentně

$$(E - A)X = Y \, . \tag{7.1}$$

Je-li matice $(E - A)$ regulární a existuje-li tedy inverzní matice $(E - A)^{-1}$, platí také

$$X = (E - A)^{-1}Y \, . \tag{7.2}$$

7.0.7 Jednoduché úlohy. Výše uvedené rovnice umožňují řešit jednoduché úlohy.

Nejčastěji počítáme nový vektor výroby, známe-li matici technických koeficientů a je-li dán nový (změněný) vektor konečné spotřeby. Jde o prosté řešení soustavy lineárních rovnic (7.1).

Ještě jednodušší je výpočet vektoru konečné spotřeby (prosté dosazení do soustavy (7.1)).

Jiná úloha může spočívat ve výpočtu vektoru výroby při stejné konečné spotřebě, pokud víme, že se technické koeficienty změní nám známým způsobem.

Poněkud složitější je kombinovaná úloha, kdy při známé matici technickýcjh koeficientů máme vypočítat vektor výroby a vektor konečné spotřeby, je-li pro některá odvětví dána velikost výroby a pro ostatní odvětví je předepsána velikost konečné spotřeby. I tato úloha však nakonec vede na řešení soustavy lineárních rovnic.

7.0.8 Matice norem plné spotřeby je matice $H = (E - A)^{-1}$ (viz rovnice (7.2)).

Prvek h_{ij} vyjadřuje velikost výroby i-tého produktu, pokud v konečné spotřebě má být spotřebována (pouze) jedna jednotka j-tého produktu (a nulová množství ostatních produktů). Toto lze snadno odvodit z rovnice (7.2), když jako Y vezmeme vektor tvořený samými nulami a jedinou jedničkou na j-té pozici.

Prvky h_{ij} nazýváme *normami plné spotřeby*. Slovo „plné“ je třeba chápat jako protiklad ke spotřebě přímé, která je vyjádřena maticí technických koeficientů. Normy plné spotřeby h_{ij} zahrnují kromě spotřeby přímé i spotřebu nepřímou, která je zprostředkována pomocí přímo spotřebovávaných produktů.

Příklad: K výrobě podkovy (přímo) potřebujeme železo a koks pro kovářskou výheň. Technické koeficienty vyjadřují, kolik železa a kolik koksu spotřebujeme přímo při výrobě jedné podkovy. K výrobě železa (přímo) potřebujeme železnou rudu a koks a i zde technické koeficienty vyjadřují, kolik rudy a koksu spotřebujeme přímo při výrobě jednotkového množství železa. Koks se tedy

„dostává do podkovy“ dvěma cestami: jednak přímo, jednak prostřednictvím železa. Kromě toho z koeficientu plné spotřeby bude také vidět, kolik na jednu podkovu potřebujeme železné rudy, třebaže ji při výrobě podkovy vůbec nespotřebováváme přímo, ale jen zprostředkovaně v podobě železa.

7.0.9 Příklad. Máme tři výrobní odvětví. Velikost výroby, výrobní spotřeby a konečné spotřeby je dána tabulkou

Odvětví	Výroba	spotřeba v odv.			konečná
		1.	2.	3.	spotřeba
1.	150	15	20	20	95
2.	200	30	0	60	110
3.	100	60	20	10	10

Matice norem plné spotřeby je

$$A = \begin{pmatrix} 0.1 & 0.1 & 0.2 \\ 0.2 & 0.0 & 0.6 \\ 0.4 & 0.1 & 0.1 \end{pmatrix}$$

odtud

$$E - A = \begin{pmatrix} 0.9 & -0.1 & -0.2 \\ -0.2 & 1.0 & -0.6 \\ -0.4 & -0.1 & 0.9 \end{pmatrix}$$

a dále

$$H = (E - A)^{-1} = \begin{pmatrix} 1.3333 & 0.1746 & 0.4127 \\ 0.6667 & 1.1587 & 0.9206 \\ 0.6667 & 0.2063 & 1.3968 \end{pmatrix} .$$

Pokud se vektor konené spotřeby změní z dosavadního $Y = (95, 110, 10)$ na $Y = (95, 100, 10)$, pak nový vektor výroby dostaneme jako $X = HY = (E - A)^{-1} \cdot Y = (148.27, 188.40, 97.92)^T$.

Dodejme, že pro jednorázový výpočet vektoru výroby není nutné počítat inverzní matici $H = (E - A)^{-1}$. Vektor výroby lze získat řešením soustavy lineárních rovnic $(E - A)X = Y$, tedy v našem příkladě řešením soustavy rovnic s rozšířenou maticí

$$\left(\begin{array}{ccc|c} 0.9 & -0.1 & -0.2 & 95 \\ -0.2 & 1.0 & -0.6 & 100 \\ -0.4 & -0.1 & 0.9 & 10 \end{array}\right) .$$

7.0.10 Energetická náročnost je celkové množství energie přímo či nepřímo spotřebované na výrobu jednotkového množství produktu.

Vzhledem k tomu, že výroba energie zpravidla je zahrnuta ve strukturním modelu jako odvětví, lze energetickou náročnost určit přímo z matice norem plné spotřeby H.

7.0.11 Celková pracnost produktu je celkové množství lidské práce, která byla vynaložena na výrobu jednotkového množství produktu, ať již byla vložena přímo v odvětví, které tento produkt vyrábí, nebo nepřímo, prostřednictvím produktů, které byly k výrobě použity (spotřebovány).

Poněvadž lidskou práci obvykle nepokládáme za produkt zvláštního odvětví strukturního modelu, musíme si pomoci jinak:

Předpokládejme, že známe množství práce vynaložené přímo na jednotku výroby v jednotlivých odvětvích. Lze to zjistit např. z počtu zaměstnanců v odvětví a velikosti výroby. Označme vektor těchto hodnot W, přičemž tentokrát půjde o vektor řádkový.

Dále označme T řádkový vektor, jehož složky t_j vyjadřují celkové množství práce obsažené v produktu j.

Celkové množství práce t_j je rovno součtu přímé práce w_j a práce, která je obsažena v produktech, které v j-tém odvětví spotřebováváme. Tedy celková pracnost jednotkového množství j-tého produktu je rovna

$$t_j = w_j + \sum_{i=1}^{n} t_i a_{ij} \ .$$

Vyjádřeno maticově

$$T = W + TA \ ,$$

tedy

$$T(E - A) = W$$

a pokud matice (E-A) je regulární, pak

$$T = W(E - A)^{-1} \ .$$

Tedy i zde využijeme matici norem plné spotřeby H.

Kapitola 8

Teorie grafů

Pojem grafu je přirozeným prostředkem k formálnímu vyjádření párových vztahů, tj. vztahů mezi dvojicemi nějakých objektů. Přes zdánlivou jednoduchost mají grafy mnoho velmi různorodých aplikací.

8.1 Definice grafu

8.1.1 Grafy orientované a neorientované. Graf se skládá z tzv. *vrcholů* a tzv. *hran*. Množinu vrcholů značíme obvykle V, množinu hran E (od anglického edge).

Hrana vždy spojuje dva vrcholy (ne nutně různé) a je buď orientovaná, nebo neorientovaná.

U hran orientovaných rozlišujeme počáteční a koncový vrchol a říkáme, že hrana vede z počátečního do koncového vrcholu. Počáteční vrchol hrany e značíme $\textsc{Pv}(e)$, koncový vrchol $\textsc{Kv}(e)$.

Neorientované hrany chápeme jako symetrické spojení dvou vrcholů a nerozlišujeme, který z obou vrcholů byl uveden dříve.

V obou případech říkáme, že hrana je *incidentní* (popř. inciduje) s vrcholy, které spojuje a také každý s těchto vrcholů je *incidentní* s touto hranou. Vztah incidence lze formalizovat jako zobrazení ε, které každé hraně přiřazuje dvojici vrcholů (pro orientované grafy přiřazuje dvojici uspořádanou, pro neorientované grafy dvojici neuspořádanou.

Pokud hrana spojuje nějaký vrchol se sebou samým, nazýváme ji *smyčkou*. Smyčka může být orientovaná nebo neorientovaná.

Orientovaný graf má všechny hrany orientované, neorientovaný graf má všechny hrany neorientované. Existují i smíšené grafy, které mají oba druhy hran, ale těmi se nebudeme zabývat.

8.1.2 Kreslení grafů. Grafy (orientované i neorientované) lze s výhodou znázorňovat kreslením. Ostatně odtud dostaly jméno.

Vrcholy obvykle kreslíme jako body (kroužky), hrany jako čáry (křivky, úsečky, oblouky), které spojují příslušné dvojice vrcholů. Je-li hrana orientovaná, značíme orientaci šipkou od počátečního ke koncovému vrcholu. Čára, která je obrazem hrany, nesmí mít jako vnitřní bod obraz žádného vrcholu. Čáry se mohou křížit, samo křížení se nepovažuje za vrchol.

Je třeba mít na paměti, že graf lze většinou nakreslit mnoha k nepoznání různými způsoby.

Rovinné nakreslení je takové nakreslení grafu, že libovolné dvě křivky přiřazené různým hranám grafu mají společné nejvýše své krajní body, tj. body přiřazené vrcholům grafu. *Rovinný graf* (též *planární*) je takový graf, ke kterému existuje rovinné nakreslení. Ne každý graf je rovinný a i rovinný graf lze nakreslit nerovinným způsobem.

Přesto, že se grafy často kreslí, je chybou chápat vrcholy a hrany jen jako body a křivky v prostoru. I nehmotná myšlenka může být vrcholem grafu.

8.1.3 Cvičení. V následujících příkladech grafových popisů reálných situací rozhodněte, který druh grafu (orientovaný, neorientovaný) lépe vystihuje skutečnost:

Situace: Silniční síť
Vrcholy: Křižovatky
Hrany: Křižovatky i, j jsou spojeny přímou silnicí

Situace: Silniční síť
Vrcholy: Silnice
Hrany: Silnice i, j mají společnou křižovatku

Situace: Elektrické zařízení
Vrcholy: Vodiče (uzly)
Hrany: Mezi vodiči i, j je zapojena součástka

Situace: Šachy
Vrcholy: Postavení figurek na šachovnici spolu s informací, kdo je na tahu
Hrany: Z postavení i lze jedním tahem dostat postavení j

Situace: Úřad
Vrcholy: Úředníci
Hrany: Úředník i je nadřízeným úředníka j

Situace: Národní hospodářství
Vrcholy: Produkty a služby
Hrany: Produkt nebo služba i je zapotřebí k výrobě nebo vykonání j

Situace: Přidělování úkolů
Vrcholy: Pracovníci a úkoly
Hrany: Pracovník i je schopen vykonat úkol j

Situace: Složitá činnost, proces
Vrcholy: Jednoduché činnosti, úkony
Hrany: Činnost i musí být dokončena před započetím činnosti j

Situace: Rodokmen
Vrcholy: Lidé
Hrany: Osoba i je otcem osoby j

Situace: Rodinné vztahy
Vrcholy: Lidé
Hrany: Osoba i je rodičem osoby j

Situace: Mnohostěn
Vrcholy: Vrcholy mnohostěnu
Hrany: Vrcholy i, j leží na stejné hraně mnohostěnu

Situace: Úloha lineárního programování
Vrcholy: Bázická přípustná řešení, tj. krajní body množiny přípustných řešení
Hrany: Jejich simplexové tabulky lze získat jednu z druhé Jordanovou eliminací

8.1.4 Značení množin vrcholů a hran.

$V_G^+(x)$ $= \{z \in V \mid (x,z) \in \varepsilon(E)\}$ = množina *následníků vrcholu* x,
tj. množina vrcholů, do nichž vede hrana z x.

$V_G^-(x)$ $= \{z \in V \mid (z,x) \in \varepsilon(E)\}$ = množina *předchůdců vrcholu* x,
tj. množina vrcholů, z nichž vede hrana do x.

$V_G(x)$ $= V_G^+(x) \cup V_G^-(x)$ = množina *sousedů vrcholu* x,
tj. množina vrcholů spojených hranou s vrcholem x.

$V_G(A)$ $= \bigcup_{x \in A} V_G(x)$,
tj. množina vrcholů spojených hranou s některým vrcholem z A.

$E_G^+(x)$ $= \{e \in E \mid \text{Pv}(e) = x\} =$ *výstupní okolí vrcholu* x,
tj. množina hran s počátečním vrcholem x.

$E_G^-(x)$ $= \{e \in E \mid \text{Kv}(e) = x\} =$ *vstupní okolí vrcholu* x,
tj. množina hran s koncovým vrcholem x.

$E_G(x)$ $= E_G^+(x) \cup E_G^-(x) =$ *okolí vrcholu* x,
tj. množina hran incidentních s vrcholem x.

$d_G^+(x)$ $= |E_G^+(x)| =$ *výstupní stupeň vrcholu* x,
tj. počet hran vycházejících z vrcholu x.

$d_G^-(x)$ $= |E_G^-(x)| =$ *vstupní stupeň vrcholu* x,
tj. počet hran vcházejících do vrcholu x.

$d_G(x)$ $= d_G^+(x) + d_G^-(x) =$ *stupeň vrcholu* x,
tj. počet hran incidentních s vrcholem x, přičemž smyčky počítáme dvakrát.

8.1.5 Ohodnocený graf je graf, jehož vrcholy a (nebo) hrany jsou opatřeny nějakou dodatečnou informací. Zpravidla jde o číselné hodnoty, které vyjadřují např. doby trvání nebo náklady činností, propustnosti potrubí, pravděpodobnosti událostí apod. V mnoha aplikacích se bez ohodnocení neobejdeme. Ohodnocený graf bývá též nazýván *sítí*.

U jednotlivých příkladů z 8.1.3 posuďte možnosti a vhodnost rozličných ohodnocení hran či vrcholů.

8.1.6 Násobnost hran, prostý graf a multigraf. Hrany, které v orientovaném grafu mají stejný počáteční i stejný koncový vrchol, nazýváme *rovnoběžné*. V neorientovaném grafu nazýváme rovnoběžnými hrany, které spojují stejné dva vrcholy. V obou případech počet navzájem rovnoběžných hran nazýváme *násobnost hrany*.

Prostý graf je graf, v němž násobnost každé hrany je nejvýše rovna jedné, tedy graf, kde žádné dvě různé hrany nejsou rovnoběžné.

Multigraf je graf, v němž násobnosti hran mohou být i větší než jedna.

Pozor, v literatuře je často slovem graf označován pouze prostý graf a slovo multigraf se pak používá k pojmenování obecnějšího případu, kdy násobnosti hran mohou být i větší než jedna.

V této knize je tedy multigraf totéž, co (obecný) graf. Slovo graf zde používáme obecněji, než je obvyklé, jednak s ohledem na aplikace, jednak proto, že řada tvrzení i algoritmů funguje bez úprav i pro multigrafy, tj. pro grafy, které nemusí být prosté. Slovo multigraf budeme používat pro zdůraznění, že graf nemusí být prostý.

8.1.7 Speciální grafy *Diskrétní graf* je graf, který nemá žádnou hranu. Podle potřeby jej můžeme považovat za orientovaný či neorientovaný.

Úplný (neorientovaný) graf je prostý neorientovaný graf bez smyček, jehož každé dva různé vrcholy jsou spojeny hranou. Má-li n vrcholů, značíme jej K_n (obr. 8.1).

Bipartitní graf je takový graf G, jehož množina vrcholů $V(G)$ je disjunktním sjednocením dvou neprázdných množin S, T, přičemž platí, že každá hrana má jeden krajní vrchol v množině S a druhý v množině T. Množiny S, T nazýváme *stranami* bipartitního grafu. U orientovaného bipartitního grafu zpravidla požadujeme, aby všechny hrany byly orientovány souhlasně, tj. z S do T.

Úplný bipartitní graf je takový bipartitní graf, kde každá dvojice vrcholů $s \in S$, $t \in T$ je spojena přesně jednou hranou. Úplný bipartitní neorientovaný graf, jehož strany S, T mají $m = |S|$ a $n = |T|$ prvků, značíme $K_{m,n}$ (viz obrázek 8.1).

8.1.8 Rovnost grafů. Řekneme, že dva grafy $G = (V, E, \varepsilon)$ a $G' = (V', E', \varepsilon')$ jsou si rovny, jestliže $V = V'$, $E = E'$ a $\varepsilon = \varepsilon'$.

Obrázek 8.1: Úplný graf K_5 a úplný bipartitní graf $K_{3,4}$.

8.1.9 Izomorfismus. Grafy G, G' se nazývají vzájemně *izomorfní*, když existují dvě vzájemně jednoznačná zobrazení $f : V \to V'$ a $g : E \to E'$ taková, že zachovávají vztahy incidence ε a ε'.

Vztah izomorfismu značíme $G \cong G'$.

Mnoho vlastností grafů se přenáší izomorfismem. Přesněji, mnoho vlastností $\mathcal{V}$ je takových, že má-li graf G_1 vlastnost $\mathcal{V}$ a platí-li $G_1 \cong G_2$, pak i graf G_2 má vlastnost $\mathcal{V}$. Teorie grafů se téměř výlučně zabývá právě takovýmito vlastnostmi.

Příkladem vlastnosti, která se zachovává izomorfismem, je třeba „počet vrcholů stupně 2, které mají alespoň jednoho souseda stupně 3“.

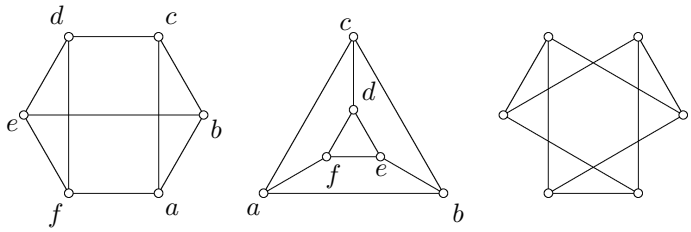
Chceme-li dokázat, že dva grafy jsou izomorfní, stačí najít (stačí uhodnout) izomorfismus (tj. zmíněná dvě zobrazení vrcholů a hran) a ověřit, že to jsou ta správná zobrazení, tj. že splňují výše uvedené podmínky. Jsou-li oba grafy prosté, stačí dokonce najít jen přiřazení vrcholů, přiřazení hran pak totiž je jednoznačně určeno. Na obrázku 8.2 jsou tři navzájem izomorfní grafy. Pro prvé dva je izomorfismus naznačen shodným pojmenováním vrcholů.

Zjednodušeně lze říci, že dva grafy jsou izomorfní, když se liší pouze nakreslením a označením (pojmenováním) vrcholů a hran.

Chceme-li naopak dokázat, že dva grafy izomorfní nejsou, je třeba buď vyzkoušet všechna možná zobrazení (což obvykle je nesmírně pracné), anebo najít vlastnost, která se přenáší izomorfismem a kterou má jen jeden z obou grafů.

Rozhodnutí, zda dva dané grafy jsou izomorfní, je v řadě případů obtížným úkolem. Při zjišťování izomorfismu lze často využít tato jednoduchá pozorování:

1. Izomorfní grafy musí mít stejné počty vrcholů i hran.
2. Vrcholu stupně k lze izomorfismem přiřadit pouze vrchol stejného stupně k.
3. Dvojici sousedních vrcholů může být izomorfismem přiřazena opět jen dvojice sousedních vrcholů.



Obrázek 8.2: Vzájemně izomorfní grafy.

8.1.10 Cvičení. O grafu na obr. 8.2 vpravo dokažte, že je izomorfní s předchozími dvěma. Návod: doplňte pojmenování vrcholů a hran, čímž vlastně získáte zobrazení f a g, a ověřte, že zachovávají vztahy incidence. Pokud nezachovávají, zkuste jiná zobrazení.

8.1.11 Podgrafy. Graf G' je *podgrafem* grafu G, vznikne-li z grafu G vynecháním nějakých (nebo žádných) vrcholů a hran. Podstatné je, že podgraf musí být také grafem: spolu s každou hranou, která je v podgrafu, tam musí být i oba její krajní vrcholy. Poznamenejme, že každý graf pokládáme za podgraf sebe sama.

Tzv. *faktor* je podgraf, který vznikne vynecháním některých hran a ponecháním všech vrcholů.

Podgraf indukovaný množinou vrcholů $M \subseteq V$ naopak vznikne tak, že z grafu odstraníme všechny vrcholy, které neleží v M (a s nimi samozřejmě i všechny hrany, které jsou incidentní s odstraněnými vrcholy.

Na obecný podgraf se tedy můžeme dívat jako na faktor indukovaného podgrafu, nebo na indukovaný podgraf faktoru.

8.2 Způsoby zadávání grafů

Pokud není graf příliš velký (zejména nemá-li mnoho hran), je zřejmě pro každého nejsrozumitelnější obrázek. Jeho snad jedinou nevýhodou je fakt, že často svádí k jednostrannému pohledu, viz např. obrázek 8.2.

Situace se ale podstatně změní, jestliže chceme pracovat s rozsáhlými grafy, které již neobsáhneme pohledem, a zejména, je-li třeba zadat graf do počítače.

8.2.1 Seznamy vrcholů a hran. Množina vrcholů je popsána prostým výčtem (seznamem) prvků, množina hran je popsána seznamem trojic tvořených jménem hrany a jejím počátečním a koncovým vrcholem. V podstatě se jedná o úplný popis grafu podle definice. Neorientované grafy popisujeme formálně stejným způsobem jako orientované, ale pořadí vrcholů jednotlivých hran volíme libovolně.

Je-li graf multigrafem, budou v seznamu hran vzájemně rovnoběžné hrany zaznamenány stejně. Proto mluvíme o seznamu, a ne o množině. V množině by opakování nebylo možné.

8.2.2 Seznam vrcholů a seznamy okolí vrcholů. Tento způsob je vlastně úspornější variantou předchozího způsobu. Množina vrcholů je opět popsána seznamem prvků, ale hrany jsou popisovány po skupinách: pro každý vrchol x je vždy uveden seznam množiny hran, které v tomto vrcholu začínají. Každá hrana pak je popsána pouze svým jménem a koncovým vrcholem, neboť počáteční vrchol x je pro celou skupinu hran společný.

Je samozřejmé, že bychom místo množin $E^+(x)$ mohli uvádět též množiny $E^-(x)$. K popisu neorientovaného grafu můžeme použít popisu některé jeho orientace anebo můžeme uvádět seznamy množin $E(x)$, což je ovšem méně úsporné, neboť každá hrana (kromě smyček) je uvedena ve dvou seznamech.

8.2.3 Matice sousednosti. Nechť G je orientovaný graf. Zvolíme-li (libovolně, ale pevně) pořadí jeho vrcholů $v_1, \ldots, v_n$, můžeme grafu G přiřadit *matici sousednosti* M_G^+ řádu n předpisem

$$m_{ij}^+ = \text{počet hran vedoucích z vrcholu } i \text{ do vrcholu } j\ .$$

Pro neorientované grafy definujeme matici sousednosti M_G předpisem

$$m_{ij} = \text{počet hran spojujících vrcholy } i \text{ a } j\ .$$

To znamená, že matice sousednosti neorientovaného grafu je vždy symetrická: $m_{ij} = m_{ji}$ pro všechna i, j. Grafy G a H na obr. 8.3 mají tyto matice sousednosti:

$$M_G^+ = \begin{pmatrix} 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \qquad M_H = \begin{pmatrix} 0 & 2 & 1 & 0 \\ 2 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}.$$

Poznamenejme, že mají-li dva grafy, oba orientované, nebo oba neorientované, stejnou matici sousednosti, pak tyto grafy jsou navzájem izomorfní. Naopak to však neplatí: změna pořadí vrcholů má za následek stejné změny v pořadí sloupců a řádků matice sousednosti. Vzájemně izomorfní grafy tedy mohou mít různé matice sousednosti.

8.2.4 Matice incidence. Nechť G je orientovaný graf bez smyček. Zvolíme-li (libovolně, ale pevně) nejen pořadí vrcholů $v_1, \ldots, v_n$, ale i pořadí hran $e_1, \ldots, e_m$, můžeme grafu G přiřadit *matici incidence* (též *incidenční matici*) B_G typu (n, m) předpisem

$$b_{ij} = \begin{cases} 1, & \text{jestliže } v_i \text{ je počátečním vrcholem hrany } e_j, \\ -1, & \text{jestliže } v_i \text{ je koncovým vrcholem hrany } e_j, \\ 0 & \text{v ostatních případech.} \end{cases}$$

Incidenční matice grafu G z obr. 8.3 je

$$B_G = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & -1 \\ -1 & -1 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & -1 & 0 & -1 & 0 \end{pmatrix}.$$

Poznamenejme, že každý sloupec obsahuje přesně jednu 1 a přesně jednu -1 (proč?) a dále, že součet hodnot v i-tém řádku je roven $d^+(v_i) - d^-(v_i)$.

Pro neorientované grafy bez smyček se incidenční matice definuje předpisem

$$b_{ij} = \begin{cases} 1, & \text{jestliže } v_i \text{ je incidentní s hranou } e_j, \\ 0 & \text{v ostatních případech.} \end{cases}$$

Incidenční matice grafu H z obr. 8.3 je

$$B_H = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}.$$

Poznamenejme, že součet hodnot v i-tém řádku je roven $d(v_i)$ a že každý sloupec obsahuje přesně dvě jedničky.

8.2.5 Matice sousednosti bipartitního grafu. Tento způsob je použitelný pouze pro bipartitní grafy a je založen na faktu, že vrcholy bipartitního grafu lze uspořádat tak, aby matice sousednosti měla tvar

$$M_G^+ = \begin{pmatrix} 0 & A \\ 0 & 0 \end{pmatrix} \qquad \text{nebo} \qquad M_G = \begin{pmatrix} 0 & A \\ A^{\mathrm{T}} & 0 \end{pmatrix}$$

v závislosti na tom, zda se jedná o bipartitní graf orientovaný, nebo neorientovaný. Podmatici A pak nazýváme maticí sousednosti bipartitního grafu. Viz též obr. 8.4.

8.2.6 Nepřímý popis grafu. V některých případech lze graf zadávat nepřímo pomocí algoritmů. Mnohdy totiž nepotřebujeme znát např. seznam hran jako celek, stačí, když pro kterýkoli vrchol v umíme nějakým algoritmem zjistit postupně všechny hrany, které z vrcholu v vycházejí. Často i množinu vrcholů nemusíme znát explicite předem: stačí, když se o existenci vrcholu dovíme díky objevení (zpracování) hrany, která v něm končí nebo začíná. Tento přístup se používá při zpracování rozsáhlých grafů, které jsou definovány nepřímo prostřednictvím popisu nějaké situace nebo soustavy.

Obrázek 8.3: Grafy, jejichž matice sousednosti jsou M_G^+ a M_H.

8.3 Sledy a odvozené pojmy

Pojem sledu odpovídá představě procházky grafem podél jeho hran.

8.3.1 Sled. Posloupnost vrcholů a hran $v_0, e_1, v_1, e_2, v_2, \ldots, e_k, v_k$ nazýváme *orientovaným sledem*, jestliže pro každou hranu e_i z této posloupnosti platí $\mathrm{Pv}(e_i) = v_{i-1}$ a $\mathrm{Kv}(e_i) = v_i$.

Posloupnost vrcholů a hran $v_0, e_1, v_1, e_2, v_2, \ldots, e_k, v_k$ nazýváme *neorientovaným sledem*, jestliže každá hrana e_i z této posloupnosti spojuje vrcholy v_{i-1}, v_i.

Vrchol v_0 v obou případech nazýváme *počátečním* a vrchol v_k *koncovým vrcholem sledu*. O sledu říkáme, že *vede z vrcholu* v_0 *do vrcholu* v_k, nebo také, že *spojuje vrcholy* v_0, v_k.

V orientovaném i neorientovaném sledu na sebe vrcholy i hrany „navazují“. U orientovaného sledu však navíc požadujeme, aby všechny hrany byly orientovány „vpřed ve směru sledu“. U neorientovaného sledu na orientaci nezáleží. Proto má pojem neorientovaného sledu smysl pro orientované i neorientované grafy. Navíc, každý orientovaný sled je také sledem neorientovaným (neboť splňuje podmínky, aby se takto mohl nazývat). Konečně poznamenejme, že v obecných sledech se mohou vrcholy i hrany opakovat.

V grafu G na obr. 8.3, str. 94, je posloupnost $v_3, e_6, v_1, e_1, v_2, e_3, v_4$ orientovaným (a samozřejmě zároveň i neorientovaným) sledem, zatímco posloupnost $v_3, e_4, v_2, e_1, v_1, e_1, v_2$ je pouze neorientovaným sledem a posloupnost v_3, e_2, v_4, e_4, v_1 vůbec není sledem.

V grafu silniční sítě s jednosměrnými silnicemi smí auta jezdit jen podle orientovaných sledů, zatímco chodci smí chodit i podle sledů neorientovaných.

Triviální sled je sled, který obsahuje jediný vrchol a žádnou hranu. Triviální sled lze pokládat za orientovaný i neorientovaný.

Každý sled (kromě triviálního) je jednoznačně určen posloupností svých hran. V prostém grafu je sled určen i posloupností vrcholů.

8.3.2 Tah, cesta. Orientovaný (neorientovaný) sled, v němž se žádná hrana neopakuje, nazýváme *orientovaným (neorientovaným) tahem*. Orientovaný (neorientovaný) sled, v němž se neopakuje žádný vrchol, nazýváme *orientovanou (neorientovanou) cestou*.

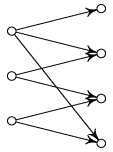
Název tah je pravděpodobně odvozen od známé úlohy nakreslit „domeček“ nebo jiný obrázek (graf) jedním tahem.

8.3.3 Uzavřené sledy. Sled (orientovaný nebo neorientovaný), který má alespoň jednu hranu a jehož počáteční a koncový vrchol splývají, nazýváme *uzavřeným sledem*. Podobně mluvíme o *uzavřeném tahu*.

Uzavřená cesta je uzavřený sled, v němž se neopakují vrcholy (kromě toho, že $v_0 = v_k$) a navíc se neopakují ani hrany. Pro uzavřené cesty se používají speciální názvy: *kružnice* je neorientovaná uzavřená cesta a *cyklus* je orientovaná uzavřená cesta. Opět platí, že cyklus je zároveň i kružnicí, ale naopak to neplatí.

Kružnice, která má tři hrany, se nazývá *trojúhelník*.

Poznamenejme, že triviální sled nepokládáme za sled uzavřený. Pro definici uzavřených cest jsme museli zakázat kromě opakování vrcholů také opakování hran proto, aby se posloupnost v_0, e, v_1, e, v_0 nemohla nazývat uzavřenou cestou.



$$A = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

Obrázek 8.4: Orientovaný bipartitní graf a jeho matice sousednosti.

8.3.4 Sledy v ohodnocených grafech. V ohodnocených grafech má často velmi dobrý smysl hovořit o součtu ohodnocení hran v nějakém sledu, cestě, cyklu apod. Jestliže např. ohodnocení hrany vyjadřuje vzdálenost, množství práce, času nebo peněz potřebných k průchodu hranou, pak součet ohodnocení hran má jistě dobrý praktický smysl. Přitom ohodnocení každé hrany počítáme vždy tolikrát, kolikrát se hrana ve sledu vyskytuje. Pro tento součet ohodnocení hran se vžil termín *délka sledu (cesty, tahu, kružnice, cyklu)* a v tomto smyslu se hovoří o *nejkratším* nebo *nejdelším sledu, cestě* atd. Hledání nejkratších nebo nejdelších cest patří k nejčastěji aplikovaným úlohám teorie grafů.

Pozor, termín délka sledu (cesty, atd.) je často užíván také pro označení *počtu hran* ve sledu, cestě apod. Tento rozpor ve vžité terminologii lze formálně řešit úmluvou, že počet hran ve sledu budeme pokládat za součet ohodnocení, je-li každá hrana ohodnocena jedničkou.

Dodejme, že hledání cest s nejmenším počtem hran je podstatně jednodušší (viz 8.6.4) než hledání cest s nejmenším součtem ohodnocení (viz 8.9).

My budeme používat termíny délka sledu a nejkratší nebo nejdelší sled, cesta atd. téměř vždy ve smyslu „součet ohodnocení“ a jen výjimečně ve smyslu „počet hran“, přičemž na tyto výjimky vždy zvlášť upozorníme.

8.3.5 Dostupnost. Řekneme, že vrchol y je *orientovaně (neorientovaně) dostupný z vrcholu* x, jestliže existuje orientovaný (neorientovaný) sled vedoucí z vrcholu x do vrcholu y. Označme

$D^+_G(x)$ = množina všech vrcholů orientovaně dostupných v grafu G z vrcholu x,

$D^-_G(x)$ = množina všech vrcholů, z nichž je v grafu G orientovaně dostupný vrchol x.

Nebude-li hrozit nedorozumění, budeme místo $D^+_G(x)$ a $D^-_G(x)$ psát stručněji $D^+(x)$ a $D^-(x)$.

V definici dostupnosti jsme samozřejmě mohli se stejným výsledkem místo sledu použít cestu. (Proč?)

Množiny $D^+(x)$ lze snadno nalézt pomocí algoritmů pro prohledávání grafu uvedených v oddíle 8.6, str. 101. Množiny $D^-(x)$ lze nalézt použitím týchž algoritmů na obrácený graf (graf, který získáme obrácením orientace všech hran).

8.4 Souvislost a silná souvislost

8.4.1 Souvislost. Graf nazýváme *souvislým*, jestliže každé jeho dva vrcholy jsou spojeny neorientovanou cestou.

Pro odlišení od tzv. silné souvislosti (viz 8.4.6) někdy mluvíme o *slabé* nebo také *obyčejné souvislosti*.

Všimněte si, že jsme souvislost definovali zároveň pro grafy orientované i neorientované. Je-li graf orientovaný, na orientaci hran nebereme ohled a zajímáme se pouze o neorientované cesty.

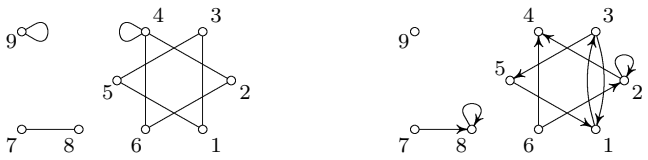
8.4.2 Komponenta souvislosti grafu G (též *souvislá komponenta* nebo i jen *komponenta*) je každý podgraf H grafu G, který je souvislý a který je maximální s touto vlastností, tj. není částí většího souvislého podgrafu.

Dva vrcholy x a y leží ve stejné komponentě souvislosti grafu G právě tehdy, když v grafu G existuje neorientovaná cesta z vrcholu x do vrcholu y.

Každý vrchol grafu leží v přesně jedné komponentě souvislosti. Mezi různými komponentami souvislosti nikdy nevede hrana.

8.4.3 Příklad. Oba grafy na obr. 8.5 mají každý čtyři komponenty souvislosti s množinami vrcholů $\{1,3,5\}$, $\{2,4,6\}$, $\{7,8\}$, $\{9\}$.

8.4.4 Hledání komponent souvislosti. Zvolíme libovolně vrchol r a pomocí kteréhokoli algoritmu pro hledání neorientovaných cest (viz kap. 8.6, str. 101) najdeme množinu vrcholů



Obrázek 8.5: Grafy k příkladu 8.4.3.

neorientovaně dostupných z vrcholu r. Podgraf indukovaný touto množinou je komponentou souvislosti.

Pokud zbývá nějaký vrchol mimo dosud nalezené komponenty, zvolíme jako r některý z nich a celý postup opakujeme, dokud není každý vrchol zařazen do některé komponenty.

8.4.5 Cvičení. V následujících neformálně popsaných grafech objasněte význam pojmu komponenty souvislosti:

1. Vrcholy jsou atomy, hrany jsou chemické vazby mezi nimi.
2. Vrcholy jsou lidé a hrana z vrcholu x do vrcholu y vede právě tehdy, když x je otcem nebo matkou osoby y. (Všimněte si, že tento graf je orientovaný, ale my se ptáme na komponenty souvislosti, u nichž na orientaci hran nezáleží.)

8.4.6 Silná souvislost. Orientovaný graf G nazýváme *silně souvislým*, když pro každou dvojici jeho vrcholů x, y existuje v G orientovaná cesta z x do y a také zpět z y do x.

8.4.7 Silně souvislá komponenta grafu G (též *silná komponenta* nebo *komponenta silné souvislosti*) je podgraf H grafu G, který je silně souvislý a který je maximální s touto vlastností, tj. není částí většího silně souvislého podgrafu.

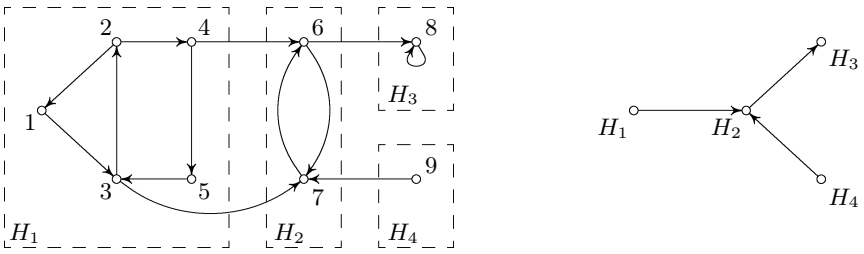
Dva vrcholy x a y leží ve stejné silně souvislé komponentě grafu G právě tehdy, když v grafu G existuje orientovaná cesta z x do y a také z y do x.

Každý vrchol leží v přesně jedné silně souvislé komponentě. Mezi různými silně souvislými komponentami mohou vést hrany, ale vždy jen v jednom směru.

každý cyklus je vždy celý obsažen v jedné silné komponentě.

Hrana je obsažena v nějakém cyklu právě tehdy, když oba její vrcholy (počáteční i koncový) leží v téže silně souvislé komponentě.

Silně souvislá komponenta, která obsahuje alespoň dva různé vrcholy, nutně obsahuje aspoň jeden cyklus.



Obrázek 8.6: Orientovaný graf, jeho silné komponenty a kondenzace.

8.4.8 Příklad. Graf G na obr. 8.6 má čtyři silně souvislé komponenty H_1, H_2, H_3, H_4. Podgraf indukovaný množinou $\{1, 2, 3\}$ je silně souvislý, ale není silně souvislou komponentou, neboť je částí většího silně souvislého podgrafu H_1.

Všimněte si, že graf G je souvislý, ale nikoli silně souvislý. Dále si všimněte, že v komponentě H_1 sice každá hrana leží na nějakém cyklu, ale neexistuje cyklus, který by procházel přes všechny vrcholy.

8.4.9 Kondenzace. *Kondenzace* grafu G je prostý orientovaný graf G' bez smyček takový, že:

1. Vrcholy grafu G' jsou všechny silně souvislé komponenty grafu G.
2. Vrcholy H_1, H_2 (tj. silně souvislé komponenty grafu G) jsou v grafu G' spojeny hranou z H_1 do H_2 právě tehdy, když $H_1 \neq H_2$ a v grafu G existuje hrana z některého vrcholu komponenty H_1 do některého vrcholu komponenty H_2.

Kondenzace orientovaného grafu neobsahuje žádný cyklus.

8.4.10 Věta. Silně souvislá komponenta obsahující vrchol x má množinu vrcholů rovnu $D^+(x) \cap$ $\cap\, D^-(x)$. (Množiny $D^+(x)$ a $D^-(x)$ jsme definovali v 8.3.5.)

8.4.11 Hledání silně souvislých komponent. Předchozí věta dává jednoduchý návod k sestrojení silně souvislé komponenty, která obsahuje daný vrchol r.

Všechny silně souvislé komponenty pak lze sestrojit opakováním tohoto postupu: zvolíme vrchol, který neleží v žádné dosud nalezené komponentě, a najdeme silnou komponentu, která tento vrchol obsahuje. Výpočet končí, jsou-li všechny vrcholy zařazeny do silných komponent.

V 8.6.10, str. 105 uvedeme podstatně rychlejší algoritmus založený na prohledávání grafu do hloubky.

8.5 Stromy a kostry

8.5.1 Les a strom. *Les* je graf, který neobsahuje kružnici. *Strom* je graf, který neobsahuje kružnici a je navíc souvislý.

Komponentami souvislosti lesa jsou tedy stromy.

8.5.2 Vlastnosti stromů. Nechť G je graf, kter má $n \geqq 1$ vrcholů. Potom následující podmínky jsou ekvivalentní:

1. G je strom.
2. G neobsahuje kružnici a má přesně $n-1$ hran.
3. G neobsahuje kružnici a má alespoň $n-1$ hran.
4. G je souvislý a má přesně $n-1$ hran.
5. G je souvislý a má nejvýše $n-1$ hran.
6. G je souvislý a odebráním kterékoli hrany přestane být souvislý.
7. G neobsahuje kružnici a po přidání libovolné hrany bude obsahovat přesně jednu kružnici.
8. Každá dvojice vrcholů je spojena přesně jednou neorientovanou cestou.

Lze tedy konstatovat, že strom má nejmenší možný počet hran, aby mohl být souvislý, ale zároveň má i největší možný počet hran, aby v něm ještě nemusela být kružnice.

8.5.3 Kořenový strom je orientovaný graf, v němž existuje význačný vrchol r, tzv. *kořen*, takový, že

- všechny vrcholy grafu jsou z kořene orientovaně dostupné,
- do kořene nevede žádná hrana a
- do každého jiného vrcholu vede přesně jedna hrana.

Kořenový strom tedy je stromem (tj. je souvislý a má počet hran o 1 menší než počet vrcholů).

Kořenové stromy patří k nejužitečnějším a nejčastěji používaným grafům. Obrázky kořenových stromů si pro znázornění hierarchické struktury kreslí i lidé, kteří o teorii grafů nemají ani tušení.

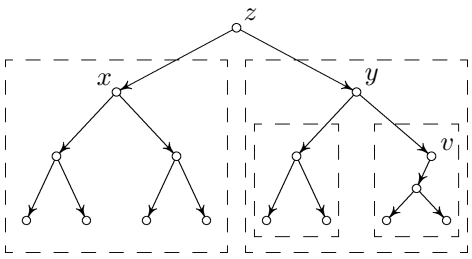
V kořenovém stromě do každého vrcholu vede z kořene přesně jedna orientovaná cesta. Na obr. 8.7 jsou dva příklady kořenových stromů. (Najděte kořeny.)

Obrázek 8.7: Kořenové stromy.

Pro kořenové stromy se často používá zvláštní „přírodopisně-rodinná“ terminologie. Vede-li v kořenovém stromě hrana z vrcholu x do vrcholu y, pak vrchol x nazýváme *otcem vrcholu* y a vrchol y nazýváme *synem vrcholu* x. O vrcholech, které mají společného otce, mluvíme jako o bratrech. Vrchol, který nemá žádného syna, nazýváme *listem*. Poznamenejme, že každý vrchol kořenového stromu má přesně jednoho otce s výjimkou kořene, který otce nemá.

Kořenový strom se obvykle kreslí tak, že kořen je nahoře a všechny hrany vedou shora dolů tak, aby se nekřížily. Jiný oblíbený způsob je kreslit kořen vlevo a hrany zleva doprava. Strom na obrázku 8.7 vpravo je tedy nakreslen poněkud netradičním způsobem.

Uspořádaný kořenový strom je kořenový strom, v němž je pro každý vrchol určeno uspořádání jeho synů. Uspořádaný kořenový strom kreslíme zpravidla tak, aby synové byli uspořádáni zleva doprava.



Obrázek 8.8: Binární kořenový strom a podstromy vrcholů z a y.

Binární kořenový strom je kořenový strom, jehož každý vrchol má nejvýše dva syny. Obvykle přitom rozlišujeme pořadí synů a ve shodě s obvyklým způsobem kreslení pak mluvíme o levém a pravém synovi.

Podstromem kořenového stromu se zpravidla nenazývá jakýkoli podgraf, který je kořenovým stromem. Je-li x vrcholem kořenového stromu G, pak *podstrom určený vrcholem* x nebo *podstrom s kořenem* x je podgraf, který je indukován množinou všech vrcholů orientovaně dostupných z vrcholu x.

8.5.4 Věta. Každý souvislý graf má faktor (tj. podgraf vzniklý vynecháním hran), který je stromem.

Důkaz: Obsahuje-li graf kružnici, odstraníme libovolnou hranu této kružnice. Souvislost grafu tím zůstane zachována. Takto opakovaným odstraňováním hran získáme podgraf, který obsahuje všechny vrcholy původního grafu, je souvislý a neobsahuje již žádnou kružnici. □

8.5.5 Kostra grafu. Faktor grafu G, který je stromem, nazýváme *kostrou grafu* G (též *napnutým stromem* grafu G).

Předchozí věta 8.5.4 říká, že každý souvislý graf má kostru.

8.5.6 Minimální kostra Je dán souvislý graf G, jehož hrany jsou ohodnoceny reálnými čísly, která budeme nazývat *cenami*. Kostru grafu G nazveme *minimální kostrou*, jestliže má nejmenší cenu, tj. nejmenší součet ohodnocení hran (mezi všemi kostrami grafu G).

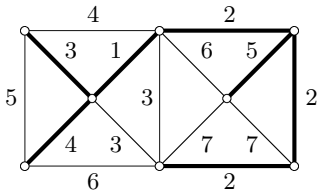
8.5.7 Aplikace úlohy o minimální kostře. Předpokládejme, že máme za úkol propojit n míst $1, 2, \ldots, n$ vedením vysokého napětí. Pro každou dvojici míst i, j máme k dispozici odhad nákladů na postavení přímé linky mezi místy i, j. Naší snahou je propojit všech n míst co nejlevněji, přitom však nesmíme vedení větvit mimo místa, která propojujeme.

Je zřejmé, že vybudované linky nesmějí tvořit kružnici, jinak by bylo možno vynecháním některé linky snížit celkové náklady. Vybudované linky tedy musí tvořit minimální kostru. Všimněte si, že toto nejlevnější řešení není ideální z hlediska spolehlivosti: porucha kterékoli linky způsobí rozpad sítě na dvě komponenty souvislosti.

Jiným příkladem může být silniční síť, z níž chceme v zimě udržovat sjízdnou pouze co nejkratší část, která vzájemně propojí všechny obce v dané oblasti.

Kromě těchto přímých aplikací se s minimální kostrou setkáváme jako s dílčím problémem při řešení složitějších kombinatorických úloh.

8.5.8 Příklad. V grafu na obr. 8.9 je minimální kostra nakreslena silně.



Obrázek 8.9: Minimální kostra.

8.5.9 Hladový algoritmus (též Kruskalův) pro minimální kostru. Vezmeme všechny vrcholy daného grafu G a vytvoříme z nich diskrétní graf a označíme jej L. Hrany grafu G uspořádáme podle jejich cen do neklesajícího pořadí, tj. od nejlevnější po nejdražší. Pak hrany v tomto pořadí postupně bereme a přidáváme je do grafu L. Přitom přidáváme jen ty hrany, které v grafu L nezpůsobí vznik kružnice. Přidáním hrany se tedy vždy spojí dvě komponenty lesa L. Výpočet končí, když se les L stane souvislým (a tedy stromem).

Název „hladový algoritmus“ vystihuje fakt, že k lesu L přidáváme vždy tu nejlevnější hranu, která v dané situaci připadá v úvahu.

8.5.10 Věta. Pro každý souvislý graf G a pro každé ohodnocení jeho hran reálnými čísly hladový algoritmus 8.5.9 nalezne minimální kostru.

Poznámka: Úloha o minimální kostře patří k vzácnému typu optimalizačních úloh, pro které algoritmus hladového typu poskytuje skutečné optimum. Pro většinu úloh algoritmy hladového typu fungují pouze jako heuristiky (viz 4.5.1, str. 72), které dávají pouze přibližné řešení různé kvality.

8.5.11 Věta (charakterizace minimální kostry). Kostra K grafu G je minimální kostrou právě tehdy, když

(∗∗) pro každou hranu $e \notin E(K)$ a pro každou hranu h na (jediné) cestě, která v kostře K spojuje krajní vrcholy hrany e, platí $c(e) \geqq c(h)$.

8.6 Prohledávání grafů

Účelem prohledávání grafů je zjišťování dostupnosti, tj. zjišťování, do kterých vrcholů grafu vedou cesty z daného výchozího vrcholu, případně též nalezení těchto cest, Dalším cílem prohledávání často je vykonání nějaké akce v každém dostupném vrcholu, popř. v každé dostupné hraně.

Prohledávat „ručně“ graf, který je dán přehledným obrázkem, je hračka. Je-li však graf veliký a nepřehledný, potřebujeme systematický postup, jinak snadno uděláme chybu. Má-li graf prohledávat počítač, je systematický postup nevyhnutelný. A u grafů, které jsou zadány nepřímo pomocí procedur pro generování hran a vrcholů (viz 8.2.6), je systematické prohledávání základním způsobem, jak se o grafu vůbec něco dovědět.

Uvedeme tři způsoby prohledávání. Všechny tyto postupy prohledávání popisujeme pouze ve verzi pro orientované cesty. Modifikace pro hledání neorientovaných cest jsou snadné a jsou přenechány čtenáři jako cvičení.

8.6.1 Značkování vrcholů je vlastně jen „zformulovaná trivialita“. Vrcholům grafu přiřazujeme značky. Má-li vrchol značku, znamená to, že do něj vede (nějaká) cesta z výchozího vrcholu r. Vlastní algoritmus je velmi prostý:

1. [*Inicializace.*] Označkujeme výchozí vrchol r, ostatní vrcholy jsou beze značek.
2. [*Výběr hrany.*] Vybereme libovolnou hranu e, jejíž počáteční vrchol má značku a koncový vrchol nikoli; pokračujeme podle kroku 3. Jestliže taková hrana neexistuje, výpočet končí.
3. [*Značkování.*] Označkujeme vrchol $\textsc{Kv}(e)$ a pokračujeme ve výpočtu podle kroku 2.

8.6.2 Praktické provedení značkovací procedury. Asi nejjednodušší je opakovaně (cyklicky) procházet seznamem všech hran, pro každou hranu přitom testovat, zda ji lze použít k označkování dalšího vrcholu (krok 2 algoritmu 8.6.1) a případně přidělit značku. Jestliže při celém jednom průchodu seznamem hran nebyl označkován žádný další vrchol, nemá smysl pokračovat, výpočet končí. V opačném případě pokračujeme tím, že znovu projdeme seznam hran.

Poněkud důmyslnější postupy jsou založeny na udržování seznamu vrcholů, kterým byla přidělena značka a které v tomto seznamu čekají na zpracování. Toto zpracování spočívá v postupném prozkoumání všech hran, které z vrcholu vycházejí, a v případném označkování dalších vrcholů. Nově označkované vrcholy přidáváme do seznamu, vrcholy, které jsou zcela zpracovány (nebo s jejichž zpracováním právě začínáme) ze seznamu odstraňujeme. Výpočet končí, není-li co zpracovávat, tj. je-li seznam prázdný. Důležité varianty tohoto postupu uvedeme v 8.6.4 a 8.6.7.

8.6.3 Zapamatování cest, strom nalezených cest. Sdělení, že do vrcholu v vede blíže neurčená cesta z výchozího vrcholu r, často nestačí. Často chceme vědět kudy cesta vede. Pomoc je překvapivě snadná.

Stačí si ke každému označkovanému vrcholu zapamatovat hranu e, která byla k označkování použita. Tedy ihned po označkování vrcholu $\textsc{Kv}(e)$ přiřadíme tomuto vrcholu hodnotu $\textsc{Odkud}(\textsc{Kv}(e)) := e$, kde e je hrana vybraná v kroku 2.

Takto doplněný algoritmus 8.6.1 samozřejmě dělá všechno co algoritmus nerozšířený, tj. označkuje právě všechny dostupné vrcholy, ale navíc platí:

> je-li označkován vrchol $v \neq r$, pak hodnota $\textsc{Odkud}(v)$ je jménem poslední hrany v (nějaké) cestě z vrcholu r do vrcholu v.

Cestu z počátečního vrcholu r do označkovaného vrcholu $v \neq r$ lze po skončení značkovacího algoritmu snadno nalézt zpětným postupem pomocí hodnot $\textsc{Odkud}$: Hodnota $\textsc{Odkud}(v)$ je poslední hranou v hledané cestě. Označme w počáteční vrchol hrany $\textsc{Odkud}(v)$. Je-li $w \neq r$, pak hodnota $\textsc{Odkud}(w)$ je předposlední hranou v cestě z vrcholu r do vrcholu w atd.

Označkované vrcholy a hrany obsažené v hodnotách $\textsc{Odkud}$ navíc tvoří kořenový strom s kořenem r (viz 8.5.3, str. 98). Budeme jej nazývat *strom nalezených cest.*

8.6.4 Prohledávání grafu do šířky lze stručně charakterizovat takto: nejprve označkujeme vrchol r, pak všechny jeho následníky, pak všechny dosud neoznačkované následníky těchto následníků atd. Lze si to představovat i tak, že graf současně prozkoumává velký počet průzkumníků, kteří „zaplavují“ postupně „po hladinách“ dostupnou část grafu.

Na prohledávání do šířky je pozoruhodné, že vede k nalezení nejkratších cest, tj. cest s nejmenším počtem hran.

VSTUP: Orientovaný graf G a jeho vrchol r.
ÚKOL: Najít všechny vrcholy v, do nichž vede orientovaná cesta z vrcholu r, do každého z nich najít cestu o nejmenším počtu hran a zjistit délku VZD(v) této nejkratší cesty, tj. vzdálenost vrcholu v od vrcholu r.
POMOCNÉ PROMĚNNÉ: Všechny vrcholy, do nichž bude nalezena cesta, budou označkovány a budou jim přiřazeny hodnoty ODKUD a VZD. Dále použijeme seznam FRONTA, který bude obsahovat ty vrcholy, které již byly označkovány, ale z nichž jsme ještě nezkoumali možnosti dalšího postupu.

ALGORITMUS:

1. [*Inicializace.*] Označkujeme vrchol r, ostatní vrcholy jsou beze značek. Položme VZD(r) := 0, seznam FRONTA nechť obsahuje jen vrchol r.

2. [*Test ukončení.*] Je-li seznam FRONTA prázdný, výpočet končí.

3. [*Volba vrcholu.*] Ze začátku seznamu FRONTA odebereme vrchol a označíme jej v.

4. [*Postup do šířky z vrcholu v.*] Pro každou hranu $e \in E^+(v)$ označíme w := KV(e) a jestliže w dosud nemá značku, provedeme tyto operace: Označkujeme vrchol w; ODKUD(w) := e; VZD(w) := VZD(v) + 1; vrchol w připíšeme na konec seznamu FRONTA. Po zpracování všech hran z množiny $E^+(v)$ pokračujeme podle kroku 2.

ČASOVÉ NÁROKY: Je-li graf zadán ve tvaru seznamu vrcholů a seznamů výstupních okolí všech vrcholů (viz 8.2.2), proběhne hledání v čase $O(m+n)$.

POZNÁMKA: Pro hledání do šířky je podstatné, že ze seznamu FRONTA odebíráme vždy ten prvek (zde vrchol), který je v něm nejdéle. Taková datová struktura bývá označována termínem *fronta*.

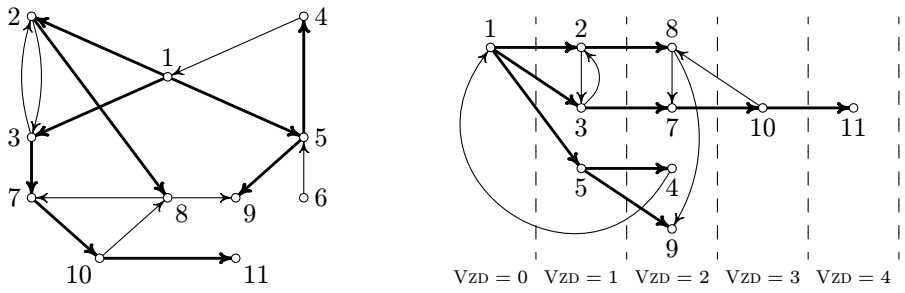
8.6.5 Příklad. Použijeme algoritmus hledání do šířky na graf G (viz obr. 8.10) s množinou vrcholů $V(G) = \{1, 2, \ldots, 11\}$ a s množinou hran $E(G)$ danou seznamem $(1,2)$, $(1,3)$, $(1,5)$, $(2,3)$, $(2,8)$, $(3,2)$, $(3,7)$, $(4,1)$, $(5,4)$, $(5,9)$, $(6,5)$, $(7,10)$, $(8,7)$, $(8,9)$, $(10,8)$, $(10,11)$. Jako výchozí vrchol vezmeme vrchol 1. Na obr. 8.10 jsou silně nakresleny hrany, které jsou po skončení algoritmu obsaženy v hodnotách ODKUD a které vytvářejí systém nejkratších cest do všech dostupných vrcholů. Vrcholy byly značkovány v pořadí 1; 2, 3, 5; 8, 7, 4, 9; 10; 11. Středníkem jsou zde odděleny množiny vrcholů, kterým byla vypočtena stejná vzdálenost od vrcholu 1. Vrchol 6 nebyl označkován, neboť není orientovaně dostupný z vrcholu 1.

Všimněte si, že obrázek k výpočtu nepotřebujeme. Napoak, obrázek je spíše na škodu. Při ručním výpočtu ovšem můžete obrázek kreslit. Viz

8.6.6 Cvičení. V grafu z příkladu 8.6.5 proveďte hledání do šířky z výchozího vrcholu 6. Řešte bez použití obrázku!

8.6.7 Prohledávání grafu do hloubky Hledání do hloubky si lze představit jako průzkum grafu cestovatelem, který cestuje po hranách grafu a vrací se důsledně cestou, po které přišel. Je to tedy dobrý návod, jak se chovat v bludišti.

Navíc je hledání do hloubky základem řady dalších, mnohdy velmi výkonných algoritmů, např. hledání silných komponent 8.6.10, str. 105, nebo nalezení topologického uspořádání 8.7.8, str. 107.



Obrázek 8.10: K příkladu 8.6.5 hledání do šířky.

Vstup: Orientovaný graf G a jeho vrchol r.
Úkol: Najít všechny vrcholy, do nichž vede orientovaná cesta z vrcholu r.
Pomocné proměnné: Proměnná v bude obsahovat jméno aktuálního vrcholu, tj. vrcholu „kde právě teď jsme“. Seznam Trasa bude obsahovat vždy posloupnost hran tvořících tzv. aktuální cestu, tj. cestu z výchozího vrcholu r do aktuálního vrcholu v. Kromě toho budeme vrcholy, do nichž byla nalezena cesta, značkovat podobně jako v algoritmu 8.6.1.

Algoritmus:

1. [*Inicializace.*] $v := r$; Trasa $:= \emptyset$; označkujeme vrchol r, ostatní vrcholy jsou beze značek.

2. [*Volba hrany e.*] Vezmeme některou dosud nepoužitou hranu e začínající ve vrcholu v a pokračujeme podle kroku 3. Pokud taková hrana neexistuje, pokračujeme podle kroku 5.

3. [*Test vhodnosti hrany e.*] Označme w koncový vrchol hrany e. Je-li vrchol w označkován, pokračujeme podle kroku 2, v opačném případě podle kroku 4.

4. [*Postup do hloubky.*] Hranu e připíšeme na konec seznamu Trasa, položíme $v := w$ a označkujeme vrchol v. Pokračujeme krokem 2.

5. [*Návrat z vrcholu v.*] Je-li seznam Trasa neprázdný, odebereme z jeho konce hranu e, její počáteční vrchol označíme v a pokračujeme podle kroku 2. Je-li seznam Trasa prázdný, výpočet končí.

Časové nároky: Je-li graf zadán ve tvaru seznamu vrcholů a seznamů výstupních okolí všech vrcholů (viz 8.2.2), proběhne hledání v čase $O(m + n)$.

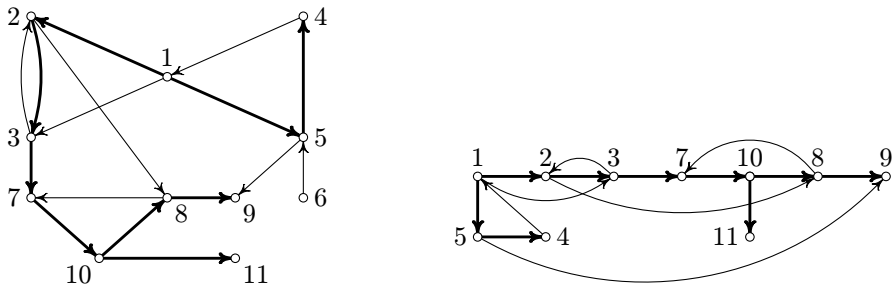
Poznámka: Pro algoritmus hledání do hloubky je podstatné, že ze seznamu Trasa odebíráme vždy hranu, která je v seznamu nejkratší dobu. Tato datová struktura bývá označována názvem *zásobník* (anglicky stack) a je v jistém smyslu opakem fronty, která je naopak podstatná při prohledávání do šířky.

Poznámka: Představa cestovatele, který chodí orientovaným grafem a vrací se, kudy přišel, má drobnou vadu na kráse: při postupu vpřed smíme jít pouze ve směru hrany, ale při návratu jdeme klidně v protisměru (ovšem při hledání do hloubky to jinak nejde).

8.6.8 Příklad. Použijeme hledání do hloubky na graf z příkladu 8.6.5. Jako výchozí vrchol vezmeme opět vrchol 1.

Tento graf je znovu nakreslen na obr. 8.11, hrany, které byly použity k postupu do hloubky, jsou zde nakresleny silně. V kroku 2 jsme přitom volili hrany vycházející z vrcholu v vždy v pořadí, ve kterém jsou uvedeny v seznamu hran $E(G)$.

Průběh výpočtu je tento: postup vpřed do vrcholů 1, 2, 3, 7, 10, 8, 9, návrat do vrcholu 8, test hrany $(8, 7)$, návrat do vrcholu 10, postup vpřed do vrcholu 11, návrat do vrcholů 10, 7 a 3, test

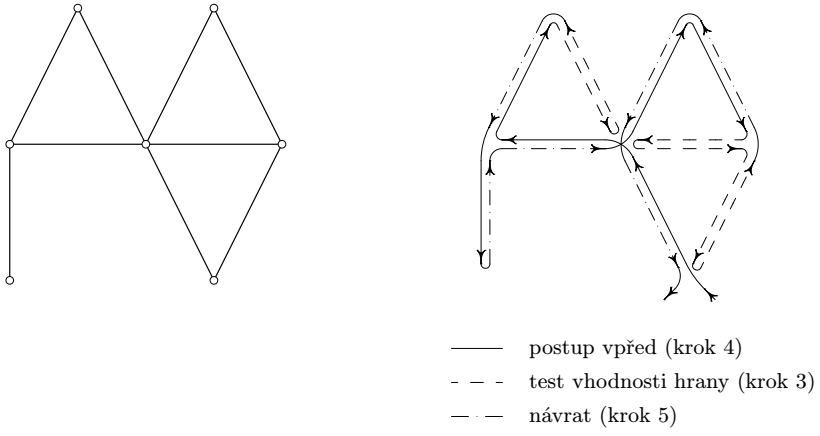


Obrázek 8.11: K příkladu hledání do hloubky.

hrany $(3, 2)$, návrat do vrcholu 2, test hrany $(2, 8)$, návrat do vrcholu 1, test hrany $(1, 3)$, postup vpřed do vrcholů 5 a 4, test hrany $(4, 1)$, návrat do vrcholu 5, test hrany $(5, 9)$, návrat do vrcholu 1.

Obrázek 8.11 vpravo byl nakreslen na základě průběhu hledání do hloubky. Všimněte si, že strom nalezených cest je zcela jiný než při hledání do šířky.

8.6.9 Návod k prohledávání bludiště. Představte si, že jste v bludišti, které se skládá z neznámého počtu místností a z neznámého počtu chodeb, každá chodba spojuje vždy dvě místnosti. Vaším úkolem je prozkoumat systematicky všechny chodby a místnosti a najít východ z bludiště nebo spolehlivě zjistit, že východ neexistuje. Předpokládejme, že máte s sebou křídu, kterou si můžete dělat na stěny nebo podlahy chodeb a místností značky.



Obrázek 8.12: Příklad bludiště a jeho prohledání metodou do hloubky.

Tuto situaci lze modelovat neorientovaným grafem, jehož vrcholy odpovídají místnostem a hrany chodbám. Vy ovšem tento graf předem neznáte. Jste-li v některé místnosti (vrcholu), vidíte pouze ty chodby (hrany), které vycházejí z této místnosti.

K prohledávání bludiště lze použít hledání do hloubky. Při hledání je nutné označovat použité chodby a odlišit je od chodeb, po nichž se budeme teprve vracet. Jednou z možností je kreslit při prvém průchodu chodbou jednu čáru a při návratu druhou. Z místnosti pak odcházíme pokud možno neoznačenou chodbou (krok 2, postup do hloubky). Vede-li tato chodba do dříve navštívené místnosti, nevstupujeme do ní, a okamžitě se vrátíme touž chodbou zpět (krok 3, test vhodnosti hrany). Pokud jsme přišli do dosud nenavštívené místnosti, postupujeme z ní dále (krok 4). Jsou-li již všechny chodby vedoucí z místnosti použity (alespoň jedenkrát), vrátíme se chodbou, ve které je jen jedna čára (krok 5, návrat). Konec hledání poznáme podle toho, že jsme v místnosti, kde všechny chodby jsou označeny dvěma čarami. V takovém případě stojíme v místě, kde jsme

začínali. Pokud východ z bludiště existuje a je dosažitelný, museli jsme jít kolem něj.

Příklad bludiště a možný způsob jeho prohledání je na obr. 8.12. Poznamenejme, že seznam TRASA je zde realizován jaksi rozptýleně v podobě všech chodeb s jednou čárou.

8.6.10 Algoritmus pro hledání silně souvislých komponent uvedeme jako ukázku algoritmu, který je založen na prohledávní do hloubky, ale jeho cílem není zjištění dostupnosti.

Během prohledávání budeme „sbírat informace“ o příslušnosti vrcholů k silným komponentám. Přesněji, budeme udržovat rozklad množiny vrcholů $V(G)$ na silné komponenty dosud prohledaného podgrafu, tj. grafu, který je tvořen všemi vrcholy grafu G a všemi dosud prohledanými (použitými) hranami. Nakonec, až bude prohledán celý graf, budeme mít rozklad na silné komponenty celého původního grafu.

Připomeňme (viz 8.6.7), že při hledání do hloubky seznam TRASA obsahuje hrany tvořící aktuální cestu, tj. cestu z výchozího vrcholu r do aktuálního vrcholu v. Silné komponenty, které obsahují některý vrchol z aktuální cesty, budou pro nás důležité, nazvěme je otevřenými komponentami. Tyto komponenty se ještě mohou během dalšího výpočtu změnit.

Silnou komponentu, ze které již byl proveden návrat, tj. jejíž žádný vrchol již neleží na aktuální cestě, nazvěme uzavřenou komponentou, neboť taková komponenta se již nemůže změnit, vše, co k ní patří, již je prohledáno.

Na začátku výpočtu bude každý vrchol sám ve své vlastní komponentě. Prohledávání do hloubky zahájíme v libovolném vrcholu.

Vždy, když při prohledávání do hloubky (v kroku 2 algoritmu 8.6.7) najdeme hranu, která z aktuálního vrcholu v vede do nějaké otevřené komponenty (tj. ne do nového vrcholu, nebo do uzavřené komponenty), našli jsme cyklus, který prochází přes jednu nebo několik otevřených komponent. Všechny tyto komponenty, přes které cyklus prochází, nahradíme jejich sjednocením a tato sjednocená komponenta je samozřejmě otevřená, neboť obsahuje aktuální vrchol.

Pokud prohledávání do hloubky skončí a není prohledán celý graf, vezmeme kterýkoli vrchol, který není v uzavřené komponentě a zahájíme nové hledání do hloubky.

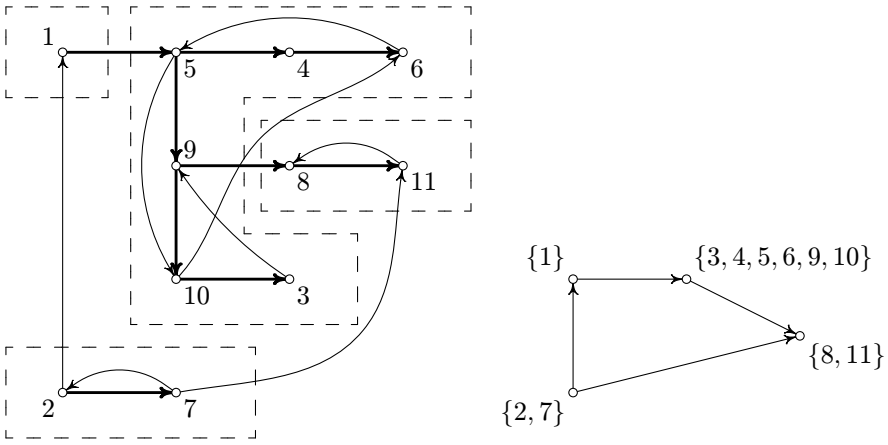
8.6.11 Hledání silných komponent. Použijeme algoritmus 8.6.10 k nalezení silných komponent v grafu s množinou vrcholů $\{1,\dots,11\}$ a množinou hran

$$\begin{array}{llllll} (1,5) & (3,9) & (5,9) & (7,2) & (9,8) & (10,6) \\ (2,1) & (4,6) & (5,10) & (7,11) & (9,10) & (11,8) \\ (2,7) & (5,4) & (6,5) & (8,11) & (10,3) & \end{array}$$

- postup do hloubky do vrcholů 1, 5, 4, 6,
- po zpracování hrany $(6,5)$ nahradíme komponenty $\{5\}$, $\{4\}$, $\{6\}$ jejich sjednocením $\{5,4,6\}$,
- návrat do vrcholu 5 a postup vpřed do vrcholů 9, 8, 11,
- po zpracování hrany $(11,8)$ nahradíme komponenty $\{8\}$ a $\{11\}$ jejich sjednocením $\{8,11\}$,
- návrat do vrcholu 9, komponenta $(11,8)$ se stala uzavřenou, postup vpřed do vrcholů 10, 3,
- po zpracování hrany $(3,9)$ nahradíme komponenty $\{9\}$, $\{10\}$, $\{3\}$ jejich sjednocením $\{9,10,3\}$,
- návrat do vrcholu 10,
- po zpracování hrany $(10,6)$ nahradíme komponenty $\{5,4,6\}$ a $\{9,10,3\}$ jejich sjednocením $\{5,4,6,9,10,3\}$,
- návrat do vrcholu 5,
- zpracováním hrany $(5,10)$ se nic nezmění,
- návrat do vrcholu 1, komponenta $\{5,4,6,9,10,3\}$ se stala uzavřenou,
- návrat z vrcholu 1, tj. konec hledání, které ve vrcholu 1 začalo, komponenta $\{1\}$ se stala uzavřenou,
- nové hledání z vrcholu 2,
- postup do hloubky do vrcholů 2, 7,
- po zpracování hrany $(7,2)$ nahradíme komponenty $\{2\}$ a $\{7\}$ jejich sjednocením $\{2,7\}$,
- návrat do vrcholu 2,

- zpracováním hrany $(2,1)$ se komponenty nezmění,
- návrat z vrcholu 2, komponenta $\{2,7\}$ se stala uzavřenou,
- všechny vrcholy jsou v uzavřených komponentách, výpočet končí.

Výsledkem jsou silně souvislé komponenty indukované množinami $\{1\}$, $\{2,7\}$, $\{3,4,5,6,9,10\}$ a $\{8,11\}$. Na obrázku 8.13 je zvýrazněn strom nalezených cest.



Obrázek 8.13: K příkladu 8.6.11 hledání silně souvislých komponent. Vpravo je nakreslena kondenzace.

8.7 Acyklické grafy

8.7.1 Acyklický graf je orientovaný graf, který neobsahuje žádný cyklus (tedy ani smyčku).

Každý acyklický graf obsahuje alespoň jeden vrchol x, do kterého nevede žádná hrana (tj. $d^-(x) = 0$), jinak bychom opakovaným postupem proti hranám našli cyklus. Podobně v acyklckém grafu vždy existuje aspoň jeden vrchol, ze kterého nevychází žádná hrana (tj. $d^+(x) = 0$).

Kondenzace libovolného orientovaného grafu (viz 8.4.9, str. 98) je vždy acyklickým grafem. Také naopak, každá silně souvislá komponenta acyklického grafu má podle 8.4.7 pouze jeden vrchol, acyklický graf je tedy sám svou kondenzací.

8.7.2 Topologické uspořádání vrcholů grafu G je taková posloupnost $v_1, v_2, \ldots, v_n$ všech vrcholů grafu, že kdykoli vede orientovaná hrana z v_i do v_j, platí $i < j$. Jinými slovy: počáteční vrchol každé hrany je v topologickém uspořádání uveden vždy dříve než koncový vrchol této hrany.

8.7.3 Topologické uspořádání hran grafu G je taková posloupnost $e_1, e_2, \ldots e_m$ všech hran grafu, že kdykoli $\textsc{Kv}(e_i) = \textsc{Pv}(e_j)$, platí $i < j$. Jinými slovy: pro každý vrchol v platí, že všechny hrany, které ve vrcholu v končí, v topologickém uspořádání předcházejí všem hranám z $E^+(v)$.

8.7.4 Aplikace topologických uspořádání. Představte si, že máte vykonat nějakou množinu činností. Činnosti musíte vykonat jednu po druhé, tj. nemůžete nebo nesmíte dělat více činností najednou a vykonávání žádné činnosti nesmí být přerušeno jinou činností. Vykonání některých činností je navíc podmíněno dokončením některých jiných činností. Například ponožku na levou nohu je třeba obout před obutím levé boty a základy domu se dělají dříve než zdi. Hledáte přípustné pořadí, ve kterém lze vykonat všechny činnosti.

Činnosti můžeme pokládat za vrcholy grafu a podmínky za jeho hrany: je-li činnost j podmíněna dokončením činnosti i, bude v grafu hrana z vrcholu i do vrcholu j. Všechna přípustná pořadí činností pak vzájemně jednoznačně odpovídají všem topologickým uspořádáním vrcholů.

Srovnejte tuto aplikaci se síťovými grafy a najděte podstatné odlišnosti obou úloh.

Významné použití topologických uspořádání je v algoritmech, které pracují s acyklickými grafy. Mnohé algoritmy se výrazně zrychlí nebo zjednoduší, pokud nějaký dílčí výpočet provádíme pro všechny vrcholy nebo hrany v pořadí podle topologického uspořádání. Příkladem je výpočet síťového grafu.

8.7.5 Věta (vlastnosti acyklických grafů). Nechť G je orientovaný graf. Pak následující podmínky jsou ekvivalentní:

1. G je acyklický.
2. G nemá smyčky a každá jeho silná komponenta má jen jeden vrchol.
3. Existuje topologické uspořádání vrcholů grafu G.
4. Existuje topologické uspořádání hran grafu G.

8.7.6 Algoritmus pro topologické uspořádání vrcholů. Posloupnost vrcholů vytvářejme postupně tak, že k ní na její konec budeme přidávat vrcholy. Pro zařazení do posloupnosti vybereme vždy takový vrchol, jehož všichni předchůdci jsou již do posloupnosti zařazeni.

Takový vrchol (jehož všichni předchůdci jsou již zařazeni) vždy existuje, neboť podgraf indukovaný nezařazenými vrcholy je acyklický a proto obsahuje vrchol, do kterého (v podgrafu) nevede žádná hrana. Na začátku, kdy je posloupnost prázná, bude vybrán vrchol, který nemá žádného předchůdce, tj. všichni jeho (neexistující) předchůdci jsou v posloupnosti zařazeni. , tj. má vstupní stupeň nula.

Pokud by nešlo zařadit všechny vrcholy grafu (tj. pokud by každý vrchol měl nezařazeného předchůdce), znamenalo by to, že v grafu je cyklus, tedy že graf není acyklický.

Pro usnadnění výpočtu je vhodné pro každý vrchol v udržovat číslo $P(v)$, které vyjadřuje počet hran, které do tohoto vrcholu vedou z dosud nezařazených předchůdců. Na začátku výpočtu budou tato čísla rovna vstupním stupňům vrcholů. Kdykoli v průběhu výpočtu nějaký vrchol v zařadíme do posloupnosti, musíme u všech jeho následníků hodnotu $P(v)$ snížit. Přesněji, abychom se správně vypořádali s násobnými hranami, pro všechny hrany z $E^+(v)$ snížíme hodnotu $P(\mathrm{K}v(e))$ o jedničku. Když se přitom pro některý vrchol v hodnota $P(v)$ vynuluje, zařadíme vrchol v do seznamu kandidátů na zařazení do posloupnosti.

Je docela běžné, že pro stejný graf existuje několik různých topologických uspořádání vrcholů. Totéž mimochodem platí i o topologických uspořádáních hran.

8.7.7 Algoritmus pro topologické uspořádání hran. Vezmeme všechny vrcholy grafu v pořadí podle topologického uspořádání vrcholů a pro každý vrchol x do posloupnosti hran zařadíme (v libovolném pořadí) všechny hrany, které z vrcholu x vycházejí.

Takto získané uspořádání hran bude jistě topologické, neboť před vrcholem x byli zpracováni všichni jeho předchůdci a tedy při zařazování hran z množiny $E^+(x)$ už jsou zařazeny všechny hrany, které z těchto předchůdců vedou do vrcholu x.

Topologické uspořádání hran lze samozřejmě vytvářet již během vytváření posloupnosti vrcholů podle 8.7.6, tj. obě topologická uspořádání (vrcholů i hran) mohou vznikat současně.

8.7.8 Topologické uspořádání hledáním do hloubky. Při prohledávání grafu do hloubky jsou návraty z vrcholů (tj. krok 5 algoritmu 8.6.7) prováděny v pořadí, které je obrácením topologického uspořádání vrcholů.

8.8 Jádro grafu a hry

8.8.1 Jádro grafu je množina vrcholů K taková, že platí $V^+(K) \cap K = \emptyset$ a také $K \cup V^-(K) = V(G)$.

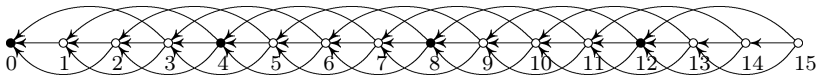
Jinými slovy, z jádra vedou hrany pouze mimo jádro a každý vrchol grafu buď sám leží v jádru, nebo z něj vede alespoň jedna hrana do jádra.

8.8.2 Hra dvou hráčů, která má konečný počet stavů a v níž se hráči střídají v tazích, může být modelována grafem. Vrcholy odpovídají stavům hry, hrany představují přípustné tahy, tj. změny stavů. Průběh hry odpovídá sledu, ve kterém jeden hráč volil vždy liché a druhý hráč sudé hrany. Jestliže hráč nemůže táhnout, tj. jestliže z příslušného vrcholu nevychází žádná hrana, pak tento hráč prohrál. Obsahuje-li graf cyklus, hra nemusí nikdy skončit.

Každého hráče jistě napadla otázka, jak hrát, aby zaručeně neprohrál. Pokud graf hry má jádro, lze neprohrávající strategii vyjádřit větou: „táhni do jádra“.

Pokud se vám podaří svým tahem dostat do některého vrcholu jádra, pak sice nelze zaručit, že vyhrajete, ale lze zaručit, že neprohrajete. Pokud totiž soupeř vůbec může táhnout, jeho tah povede ven z jádra. Vy pak můžete táhnout, a to dokonce zpět do jádra. Jinak řečeno, ať soupeř udělá cokoli, vy budete moci ve hře pokračovat, a to způsobem, který vám zachová „záruku dalšího tahu“.

8.8.3 Příklad. Na hromádce je 15 zápalek. Dva hráči se střídají v tazích, tah spočívá v odebrání 1–3 zápalek. Kdo nemůže táhnout, tj. kdo už nemá co odebrat, prohrál. Graf této hry je na obr. 8.14, kde jádro grafu je vyznačeno plnými tečkami.



Obrázek 8.14: Graf hry z příkladu 8.8.3.

8.8.4 Hledání jádra. Ne každý orientovaný graf má jádro. Příkladem je graf o třech vrcholech a třech hranách, které tvoří cyklus. Naopak některé grafy mají jader několik. Příkladem je graf o čtyřech vrcholech a čtyřech hranách, které tvoří cyklus. V plné obecnosti je úloha najít jádro dokonce NP-těžká. V řadě speciálních případů však je existence jádra zaručena a mnohdy lze jádro i snadno sestrojit.

Každý acyklický graf má přesně jedno jádro. Lze je sestrojit tak, že probíráme vrcholy v obráceném topologickém pořadí: vrchol x zařadíme do jádra právě tehdy, když žádný jeho následník do jádra zařazen nebyl.

Také každý orientovaný graf, který neobsahuje cyklus liché délky, má jádro.

8.8.5 Cvičení. Ve hře z příkladu 8.8.3 změňme pravidla: prohrává hráč, který odebere poslední zápalku. Graf této úlohy je jistě acyklický, má tedy jádro. Nakreslete tento graf a jádro najděte.

8.9 Nejkratší cesty

Úlohy o nejkratších cestách patří k nejčastěji aplikovaným úlohám teorie grafů.

8.9.1 Úlohy o nejkratších cestách. Mějme orientovaný graf G, jehož každá hrana $e \in E(G)$ je ohodnocena reálným číslem $a(e)$, které nazýváme *délkou hrany*. *Délka cesty* je součet délek jednotlivých hran tvořících cestu (viz 8.3.4, str. 96). Jsou-li x, y dva vrcholy grafu, pak *vzdálenost* $u(x, y)$ z vrcholu x do vrcholu y definujeme jako délku nejkratší cesty z x do y, pokud vůbec nějaká cesta z x do y existuje. Jestliže neexistuje, definujeme vzdálenost $u(x, y) = \infty$.

Úkolem je najít nejkratší orientovanou cestu (popř. vzdálenost):

a) z daného výchozího vrcholu do daného cílového vrcholu,
b) z daného výchozího vrcholu do každého vrcholu grafu,

c) z každého vrcholu do daného cílového vrcholu,
d) mezi všemi uspořádanými dvojicemi vrcholů.

Dále se budeme zabývat pouze řešením úloh b) a d). Úlohu c) lze snadno převést na úlohu b) obrácením orientace hran (nebo jednoduchou úpravou algoritmu). Úloha a) bývá řešena pomocí algoritmů určených pro úlohy b), c) nebo d). Postup, který by byl obecně výhodnější, není znám.

8.9.2 Matice délek hran a matice vzdáleností. Hledáme-li nejkratší cesty v multigrafech, můžeme předem z každé množiny rovnoběžných hran vynechat všechny kromě nejkratší z nich, aniž by to mělo vliv na délku nejkratší cesty. Také smyčky můžeme vyloučit, neboť nemohou být obsaženy v žádné cestě.

Pro $i \neq j$ označme $a(i,j)$ délku nejkratší hrany vedoucí z vrcholu i do vrcholu j. Pokud žádná hrana z i do j nevede, položme $a(i,j) = \infty$. Dále položme $a(i,i) = 0$. Hodnoty $a(i,j)$ můžeme uspořádat do tvaru matice, budeme ji značit A a nazývat *maticí délek hran*. Tato matice obsahuje všechny informace, které jsou třeba k výpočtu vzdáleností.

Podobně můžeme výsledné vzdálenosti $u(i,j)$ uspořádat do tvaru matice, kterou značíme U a nazýváme *maticí vzdáleností*.

8.9.3 Záporné délky hran? Ve všech zmíněných úlohách připouštíme i záporné délky hran. Není to takový nesmysl, jak to na prvý pohled vypadá. V roli délek hran totiž mohou vystupovat například náklady na průchod hranou. Záporná délka pak odpovídá situaci, kdy za průchod hranou dostaneme naopak zaplaceno. Je ovšem pravdou, že úlohy, kde délky hran jsou nezáporné, jsou v praxi suverénně nejčastější a jejich řešení je o něco snazší.

Algoritmy, které dále uvedeme se zápornými délkami hran dovedou pracovat za předpokladu, že v grafu není cyklus se zpornou délkou.

8.9.4 Nejkratší cesta versus nejkratší sled. Jak víme z 8.3.2, str. 95, v cestě se nesmí opakovat vrcholy. To znamená že v konečném grafu s konečně mnoha vrcholy existuje jen konečně mnoho cest. Proto můžeme s jistotou tvrdit, že pokud mezi nějakými dvěma vrcholy x a y, existuje aspoň jedna cesta, existuje i nejkratší z nich.

O sledech to obecně neplatí: Pokud v grafu existuje cyklus se zápornou délkou, pak nejkratší sled z x do y nemusí existovat, i když cesta z x do y vede. Stačí, aby aspoň jeden sled z x do y procházel přes vrchol záporného cyklu. Pak žádný sled z x do y není nejkratší, protože ke každému takovému sledu existuje sled ještě kratší – stačí dostatečně dlouho obíha

8.9.5 Věta. Jestliže v grafu existuje nějaká cesta z vrcholu a do vrcholu b, pak v něm existuje i nejkratší cesta z a do b.

Důkaz: Počet hran v cestě je omezen počtem vrcholů grafu. Všech cest z a do b je tedy konečně mnoho, proto některá z nich je nejkratší. □

Poznámka: Pro sledy podobná věta neplatí: Obsahuje-li graf cyklus se zápornou délkou, pak pro některé vrcholy a, b sice existuje sled z a do b, ale neexistuje nejkratší z nich. Ke každému sledu lze totiž vytvořit sled o něco kratší, a to tím, že budeme dostatečně dlouho procházet onen záporný cyklus.

8.9.6 Věta. Jestliže v grafu není cyklus se zápornou nebo nulovou délkou, pak každý nejkratší sled z vrcholu a do vrcholu b je i nejkratší cestou z a do b.

Jestliže v grafu není cyklus se zápornou délkou, pak každý nejkratší sled z a do b obsahuje nejkratší cestu z a do b a délka této cesty je stejná.

8.9.7 Věta (trojúhelníková nerovnost). Jestliže graf neobsahuje cyklus se zápornou délkou, pak pro všechny trojice vrcholů i, j, k vzdálenost u splňuje nerovnost

$$u(i,j) \leqq u(i,k) + u(k,j)\ . \tag{8.2}$$

POZNÁMKA: Název je odvozen od analogického vztahu mezi délkami stran trojúhelníka.

DŮSLEDEK: Jestliže graf neobsahuje cyklus se zápornou délkou, pro každou trojici vrcholů i, j, k platí:

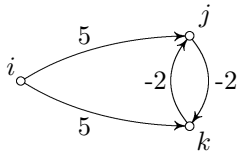
$$\begin{aligned} u(i,j) &\leqq a(i,j)\ , \\ u(i,j) &\leqq u(i,k) + a(k,j)\ , \\ u(i,j) &\leqq a(i,k) + u(k,j)\ . \end{aligned} \tag{8.3}$$

8.9.8 Věta. Předpokládejme, že graf neobsahuje cyklus se zápornou délkou. Jestliže nejkratší cesta z vrcholu i do vrcholu j prochází přes vrchol k, pak

- počáteční úsek cesty mezi i a k je nejkratší cestou z i do k,
- koncový úsek cesty mezi k a j je nejkratší cestou z k do j,
- platí $u(i,j) = u(i,k) + u(k,j)$.

DŮSLEDEK: Nechť graf neobsahuje cyklus se zápornou délkou a nechť i, j jsou dva různé vrcholy. Jestliže k je počátečním vrcholem poslední hrany v nejkratší cestě z i do j, pak platí $u(i,j) = u(i,k) + a(k,j)$.
POZNÁMKA: Věta 8.9.8 je jednou z forem tzv. *Bellmanova principu optimality*, který je základem poměrně obecné optimalizační metody, tzv. *dynamického programování*.

8.9.9 Cykly se zápornou délkou. V grafech, které obsahují cyklus se zápornou délkou, výše uvedené věty 8.9.6 až 8.9.8 ani jejich důsledky neplatí. Jednoduchý příklad je na obr. 8.15.



Obrázek 8.15: Příklad grafu, v němž neplatí trojúhelníková nerovnost.

Algoritmy pro nejkratší cesty, které dále uvádíme, však na platnost těchto vět spoléhají. Rychlost těchto algoritmů je totiž založena na tom, že se nestarají o opakování vrcholů v cestě, takže vlastně nehledají nejkratší cesty, ale nejkratší sledy. V grafech bez záporných cyklů to ovšem vyjde nastejno (viz věta 8.9.6).

8.9.10 Základní schéma výpočtu vzdáleností z výchozího vrcholu r je velmi jednoduché. Pro každý vrchol x si budeme pamatovat hodnotu $U(x)$, která bude vždy rovna délce nejkratší dosud nalezené cesty z daného výchozího vrcholu r do vrcholu x. Jestliže jsme žádnou cestu z r do x dosud nenašli, bude $U(x) = \infty$.

Kdyby hodnoty U byly skutečnými vzdálenostmi vrcholů grafu od vrcholu r, musela by pro každou hranu grafu (x, y) platit trojúhelníková nerovnost

$$U(y) \leqq U(x) + a(x,y)\ . \tag{8.4}$$

Jestliže neplatí, znamená to, že $U(y)$ není délkou nejkratší cesty z r do y, protože přes vrchol x vede do vrcholu y cesta kratší. Proto v takovém případě hodnotu $U(y)$ upravíme (snížíme) provedením příkazu

$$U(y) := U(x) + a(x,y)\ .$$

Snadno lze nahlédnout, že pak $U(y)$ bude opět délkou některé (a to kratší) cesty z vrcholu r do vrcholu y. Navíc tím nerovnost 8.4 pro hranu (x, y) začne platit, je ovšem možné, že se tím pokazila tato nerovnost pro jinou hranu.

Testy trojúhelníkové nerovnosti (8.4) a z nich případně vyplývající úpravy hodnot U budeme provádět tak dlouho, dokud nedosáhneme platnosti trojúhelníkové nerovnosti (8.4) pro všechny hrany grafu.

Na začátku výpočtu volíme $U(x) := \infty$ pro $x \neq r$ a $U(r) := 0$, neboť zpočátku neznáme žádnou cestu z vrcholu r kromě cesty triviální, která má nulovou délku. Základní schéma výpočtu vzdáleností lze shrnout takto:

1. Polož $U(r) := 0$ a pro všechny vrcholy $j \neq r$ polož $U(j) := \infty$.

2. Vyber libovolnou hranu grafu e a zkontroluj, zda pro ni platí trojúhelníková nerovnost (8.4), tj. zda $U(\text{Kv}(e)) \leqq U(\text{Pv}(e)) + a(e)$. Jestliže neplatí (tj. platí-li opak), proveď $U(\text{Kv}(e)) := U(\text{Pv}(e)) + a(e)$.

3. Jestliže nerovnost (8.4) platí pro všechny hrany grafu, výpočet končí.

Poznamenejme, že táž hrana může být použita i několikrát ke snížení hodnoty U ve svém koncovém vrcholu. Pořadí výběru hran k testu nerovnosti (8.4) i způsob zjišťování, zda tato nerovnost již platí pro všechny hrany, má, jak uvidíme dále, velký vliv na rychlost výpočtu.

8.9.11 Věta. Jestliže graf neobsahuje cyklus záporné délky, pak výpočet podle 8.9.10 skončí po konečně mnoha změnách hodnoty U a po ukončení bude pro všechny vrcholy x platit $U(x) = = u(r, x)$, tj. vzdálenosti U budou vypočteny správně.

Poznámka: Tato věta nic neříká o počtu testů trojúhelníkové nerovnosti (8.4). Při nešikovné volbě hran pro testování by výpočet nemusel vůbec skončit (kdybychom např. testovali stále tutéž hranu).

8.9.12 Cvičení. Ověřte si na nějakém příkladě (třeba na grafu z obrázku 8.15, str. 110), že pokud graf obsahuje cyklus se zápornou délkou a vrcholy tohoto cyklu jsou dostupné z výchozího vrcholu r, pak výpočet podle schématu 8.9.10 nikdy neskončí.

8.9.13 Konstrukce nejkratších cest. Vždy, když dojde ke snížení hodnoty $U(v)$ pro některý vrchol v, zaznamenáme si pro tento vrchol hranu, která snížení způsobila. K zapamatování použijeme hodnotu Odkud(v). Na začátku výpočtu nechť není hodnota Odkud definována pro žádný vrchol.

Je-li hodnota Odkud(v) definována, pak je jménem poslední hrany v nějaké cestě o délce $U(v)$ z vrcholu r do vrcholu v. Hodnoty Odkud lze po ukončení výpočtu využít ke zpětné rekonstrukci nejkratších cest (podobně jako v 8.6.3, str. 101). Poznamenejme, že během výpočtu se hodnota Odkud(v) může několikrát změnit.

Dále poznamenejme, že evidování hodnot Odkud nijak neovlivňuje postup výpočtu podle schématu 8.9.10 nebo podle kteréhokoli jeho zlepšení.

8.9.14 Věta (kořenový strom nejkratších cest). Předpokládejme, že hodnoty Odkud(v) byly získány podle 8.9.13 a že graf neobsahuje cyklus o záporné délce. Potom po ukončení výpočtu platí:

1. Množina vrcholů $D = \{v \mid \text{Odkud}(v) \text{ je definováno}\}$ je tvořena všemi vrcholy orientovaně dostupnými z r a různými od r.

2. Je-li Odkud(v) definováno, pak Odkud(v) je poslední hranou v (některé) nejkratší cestě z r do v.

3. Pro každou hranu Odkud(y) platí $U(x) + a(x, y) = U(y)$, kde x je počáteční vrchol hrany Odkud(y).

4. Množina hran $T = \{\textsc{Odkud}(v) \mid v \in D\}$ tvoří kořenový strom s kořenem r na množině vrcholů $D \cup \{r\}$.

5. Každá cesta z kořene r do libovolného vrcholu $x \in D$ v kořenovém stromě T je zároveň nejkratší cestou z r do x v původním grafu.

8.9.15 K čemu je kořenový strom nejkratších cest. Kořenový strom nejkratších cest poskytuje velmi dobrý přehled o struktuře nejkratších cest z daného výchozího vrcholu r. Výborně se hodí k prezentaci výsledků výpočtu, totiž ke sdělení, kudy nejkratší cesty vedou. Například po výpočtu nejkratších cest v pražské silniční síti stačí do plánu Prahy zakreslit hrany, které náleží ke stromu, a výsledné nejkratší cesty jsou všechny patrné na prvý pohled.

Nejkratších cest z vrcholu r do vrcholu v může být několik, v kořenovém stromě se samozřejmě objeví jen jedna z nich.

Kořenový strom nejkratších cest a větu 8.9.14 lze také využít ke kontrole správnosti vypočtených vzdáleností: stačí pro všechny hrany grafu zkontrolovat trojúhelníkovou nerovnost (8.4), přičemž pro stromové hrany v ní musí platit rovnost.

8.9.16 Poznámka k počítání s nekonečnem. Algoritmy v tomto oddíle pracují často s nekonečnem, neboť to zjednodušuje výklad. Běžné počítače a programovací jazyky ovšem s nekonečnem pracovat nedovedou. Jsou dva způsoby, jak si pomoci:

Nejjednodušší je místo symbolu ∞ použít nějakou velkou konstantu, která bude *spolehlivě* větší než dvojnásobek délek všech cest v grafu. Při použití tohoto triku musíme ovšem dávat pozor, abychom ani v mezivýsledcích nepřesáhli maximální číslo zobrazitelné v počítači. Došlo by totiž k tzv. přetečení, což vede ke zcela chybným výsledkům.

Jinou možností je využít hodnotu Odkud: Je-li $\textsc{Odkud}(x)$ jménem nějaké hrany, pak $U(x)$ je délkou nějaké cesty, a tedy $U(x) < \infty$. Jestliže naopak hodnota $\textsc{Odkud}(x)$ není ještě jménem žádné hrany (není dosud definována), pak $U(x) = \infty$.

8.9.17 Algoritmus s množinou podezřelých vrcholů. Základní schéma výpočtu vzdáleností 8.9.10 lze dále zlepšit tím, že zabráníme zbytečnému testování hran, o nichž předem víme, že trojúhelníkovou nerovnost (8.4) již splňují.

Jestliže nějaká hrana e trojúhelníkovou nerovnost (8.4) již splňovala, může k porušení této nerovnosti dojít pouze snížením hodnoty U v jejím počátečním vrcholu. Naopak, došlo-li ke snížení hodnoty $U(x)$, je nutno otestovat nerovnost (8.4) pro všechny hrany vycházející z vrcholu x, neboť pro tyto hrany by mohla být platnost nerovnosti (8.4) porušena. Budeme tedy během výpočtu udržovat množinu M „podezřelých“ vrcholů, v nichž došlo ke snížení hodnoty U a které v množině M čekají na testování hran, které z nich vycházejí.

Vstup: Graf G, ohodnocení hran $a : E(G) \to \mathbb{R}$ a výchozí vrchol r.
Výstup: Pro každý vrchol v hodnoty $U(v)$, $\textsc{Odkud}(v)$.
Pomocné proměnné: Jako pomocnou proměnnou použijeme množinu M, která bude mít tuto vlastnost:

(8.5) Jestliže $\textsc{Pv}(e) \notin M$, pak hrana e splňuje nerovnost (8.4).

Algoritmus:

1. [*Inicializace.*] $U(r) := 0$, pro $v \neq r$ položíme $U(v) := \infty$ a dále $M := \{r\}$.

2. [*Výběr vrcholu.*] Je-li $M = \emptyset$, výpočet končí. Jestliže $M \neq \emptyset$, odebereme z množiny M některý vrchol a označíme jej x.

3. [*Zpracování hran vycházejících z vrcholu x.*] Pro každou hranu $e \in E^+(x)$ provedeme krok 3a a po zpracování všech hran pokračujeme podle kroku 2.

3a. Položíme $y := \text{Kv}(e)$. Jestliže platí $U(x) + a(e) < U(y)$, pak provedeme $U(y) := U(x) + a(e)$, $\text{Odkud}(y) := e$ a navíc, pokud $y \notin M$, vložíme vrchol y do množiny M.

Poznámka: Volbu vrcholu z množiny podezřelých M lze v kroku 2 dělat v podstatě libovolným způsobem. Nemá to vliv na správnost algoritmu (viz věta **??**), ovlivňuje to však jeho rychlost. Uvedeme dva důležité případy:

8.9.18 Algoritmus s frontou podezřelých vrcholů volí v kroku 2 vrchol, který je v množině M nejdéle. S množinou M tedy pracujeme jako s frontou (viz 8.6.4), tj. jako se seznamem, z něhož odebíráme na opačném konci než přidáváme.

Výpočetní experimenty ukazují, že přes svou jednoduchost je tento algoritmus vhodný k hledání nejkratších cest ve dvou situacích: jednak v řídkých grafech, tj. v grafech, kde počet hran m nepřesahuje nějaký nevelký násobek počtu vrcholů n, jednak v grafech, kde je velké procento hran se zápornou délkou. Za zvláštní zmínku stojí skutečnost, že dokonce i na grafech s nezápornými délkami hran, pokud je graf řídký, je tento algoritmus v průměru rychlejší i než často doporučovaný tzv. Dijkstrův algoritmus.

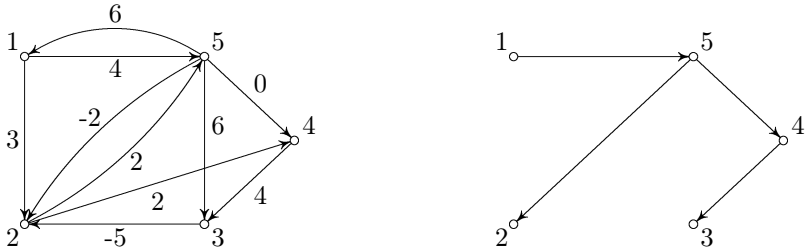
8.9.19 Modifikovaný Dijkstrův algoritmus volí v kroku 2 vrchol x s nejnižší hodnotou $U(x)$. Tato volba se odůvodňuje nadějí, že takto zvolená hodnota $U(x)$ je tak nízká, že se již v dalším výpočtu nebude dále snižovat a že už tedy tento vrchol nebude znovu zařazen do množiny M. Spolehnout se na to však lze jen v grafech, kde všechny hrany mají nezáporné délky. V obecných grafech je tento algoritmus také poměrně dobře použitelný, ale v ojedinělých případech se může stát, že týž vrchol bude do množiny M zařazen dokonce mnohokrát (až 2^{n-1}-krát).

8.9.20 Originální Dijkstrův algoritmus na grafech s nezápornými délkami hran funguje stejně jako výše uvedený 8.9.19. Mají-li některé hrany zápornou délku, Dijkstrův algoritmus může dát chybné výsledky.

Rozdíl je v tom, že originální Dijkstrův algoritmus dělí množinu vrcholů na dvě množiny: otevřené a uzavřené. Na počátku jsou všechny vrcholy otevřené. Dijkstrův algoritmus vybírá z otevřených vrcholů ten s nejnižší hodnotou U, zpracuje hrany, které z tohoto vrcholu vycházejí, a prohlásí vrchol za uzavřený a již nikdy se jím nezabývá.

Náš algoritmus 8.9.19 dovede vrchol, ve kterém klesla hodnota U(y znovu otevřít (totiž zařadit jej do množiny podezřelých). Proto funguje správně i se zápornými hranami.

8.9.21 Příklad. Použijeme modifikovaný Dijkstrův algoritmus 8.9.19 k vyhledání nejkratších cest z vrcholu 1 v grafu na obr. 8.16. Z množiny M tedy budeme vybírat vždy vrchol s nejnižší hodnotou U.



Obrázek 8.16: Graf k příkladu 8.9.21 a výsledný strom nejkratších cest.

Průběh výpočtu bude tento:

	$M = \{1\}$
$x = 1$:	$M = \emptyset$
$U(2) = 3$,	$M = \{2\}$

	$U(5)=4$,	$M=\{2,5\}$
$x=2:$		$M=\{5\}$
	$U(4)=5$,	$M=\{4,5\}$
	$U(5)$ beze změny	$M=\{4,5\}$
$x=5:$		$M=\{4\}$
	$U(2)=2$,	$M=\{2,4\}$
	$U(3)=10$,	$M=\{2,3,4\}$
	$U(4)=4$,	$M=\{2,3,4\}$
$x=2:$		$M=\{3,4\}$
	$U(4)$ beze změny	$M=\{3,4\}$
	$U(5)$ beze změny	$M=\{3,4\}$
$x=4:$		$M=\{3\}$
	$U(3)=8$,	$M=\{3\}$
$x=3:$		$M=\emptyset$
	$U(2)$ beze změny	$M=\emptyset$
		$M=\emptyset$

Všimněte si, že vrchol 2 byl do množiny M zařazen dvakrát. Sledujte také, jak se v průběhu výpočtu mění kořenový strom nejkratších cest.

Výsledek výpočtu lze shrnout do této tabulky:

vrchol x	1	2	3	4	5
vzdálenost $U(x)$	0	2	8	4	4
Odkud(x)	–	(5,2)	(4,3)	(5,4)	(1,5)

Počítáme-li ručně (tj. ne na počítači), je vhodné do takové tabulky zapisovat mezivýsledky a změněné hodnoty škrtat.

8.10 Časové plánování, metoda CPM

Při řízení projektů se k časovému plánování používají grafové modely – tzv. síťové grafy. Projekt zde chápeme jako množinu činností, které mají být všechny vykonány, přičemž se připouští paralelní vykonávání.

8.10.1 Síťový graf. je grafový model projektu, který modeluje doby trvání jednotlivých činností a jejich vzájemnou podmíněnost. Předpokládá se, že o každé činnosti známe dobu jejího trvání a dále, že pro každou činnost známe množinu činností, které musí být dokončeny před tím, než tato činnost může začít.

Cílem je vypočítat, jak dlouho bude trvat celý projekt, a naplánovat, kdy mají začínat a končit jednotlivé činnosi. Dalším cílem je zjistit, které činnosti jsou takzvaně kritické v tom smyslu, že jejich prodloužení způsobí prodloužení celého projektu.

Používají se dva druhy síťových grafů:

8.10.2 Síťový graf s ohodnocenými vrcholy má činnosti, ze kterých se projekt skládá, reprezentovány pomocí vrcholů. Každý vrchol je ohodnocen dobou trvání činnosti.

Hrany grafu vyjadřují návaznosti činností: jestliže ukončení činnosti i je podmínkou pro zahájení činnosti j, modelujeme tuto podmíněnost orientovanou hranou z vrcholu i do vrcholu j. Viz příklad na obr. 8.17 uprostřed.

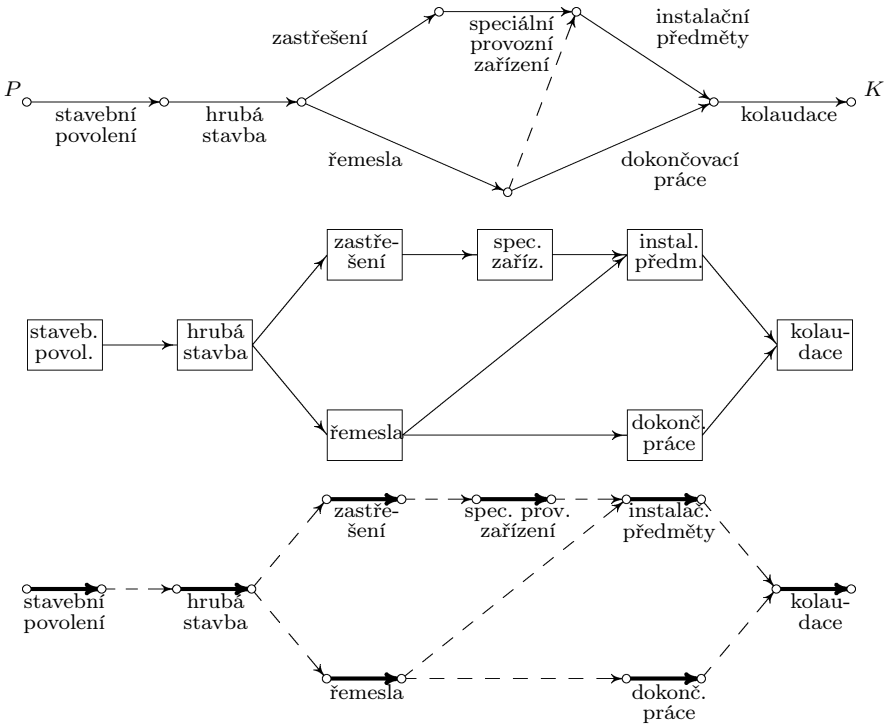
8.10.3 Síťový graf s ohodnocenými hranami má činnosti reprezentovány pomocí hran. Každá hrana je ohodnocena dobou trvání příslušné činnosti.

Návaznosti činností jsou vyjádřeny nepřímo přes návaznosti hran ve vrcholech: jestliže dvě hrany e_1, e_2 na sebe navazují, tj. platí-li Kv(e_1) = Pv(e_2), pak činnost odpovídající hraně e_1 musí

být ukončena před zahájením činnosti odpovídající hraně e_2. Jinak řečeno, činnost smí začít teprve když všechny bezprostředně předcházející činnosti jsou dokončeny.

Často se stává, že k zajištění všech požadovaných návazností je nutno použít tzv. *fiktivní činnosti* s nulovou dobou trvání. Jak název napovídá, nejde o reálnou činnost, která stojí peníze, jde jen o vyjádření povinnosti počkat.

Např. čárkovaná hrana na obrázku 8.17 nahoře vyjadřuje, že pro zahájení činnosti „instalační předměty“ je nutné počkat nejen na dokončení činností „spec. provozní zařízení“ (což je vyjádřeno návazností ve vrcholu), ale je nutno počkat také na dokončení činnosti „řemesla“, což je v modelu zajištěno pomocí fiktivní činnosti.



Obrázek 8.17: Příklady síťových grafů popisující týž proces stavby domu. Čárkované hrany představují fiktivní činnosti.

8.10.4 Co mají společného a čím se liší. V obou případech musí být síťový graf acyklický (tj. nesmí obsahovat cykly), jinak by činnosti tvořící cyklus nebylo možno vůbec zahájit.

Dále se požaduje, aby v síťovém grafu existoval *počáteční vrchol* P, ze kterého jsou orientovaně dostupné všechny vrcholu grafu, a podobně se požaduje, aby existoval *koncový vrchol* K, který je orientovaně dostupný ze všech vrcholů grafu. Každý vrchol síťového grafu tedy leží na nějaké orientované cestě z P do K. V případě grafu s ohodnocenými vrcholy to znamená, že má existovat počáteční a koncová činnost – mohou to být i činnposti fiktivní.

Síťový graf s ohodnocenými hranami je z matematického hlediska elegantnější a v literatuře suverénně nejčastější, ale jeho sestavení, zejména správné modelování všech časových návazností, je trochu náročnější a vyžaduje trochu cviku. Síťový graf s ohodnocenými vrcholy je naproti tomu snáze pochopitelný a dokáže jej sestavit i člověk, který to nikdy předtím nedělal.

Oba druhy grafů však umožňují vyjádřit zhruba totéž a lze je snadno a zcela formálním způsobem vzájemně převádět jeden na druhý. Metody těchto převodů jsou patrné z obr. 8.17, na kterém jsou nakresleny tři různé síťové grafy popisující týž proces stavby domu[1].

[1] Za příklad síťového grafu z obr. 8.17 děkuji Ludmile Hačkajlové.

8.10.5 Metoda kritické cesty (též známá pod zkratkou CPM z anglického Critical Path Method) je metoda výpočtu síťového grafu s ohodnocenými hranami. Označme

$a(e)$ = doba trvání činnosti e,

$T_1(v)$ = nejdřívější čas, kdy mohou být dokončeny všechny činnosti, které ve vrcholu v končí, a kdy tedy mohou být zahájeny činnosti, které ve vrcholu v začínají,

$T_2(v)$ = nejpozdější čas, kdy je třeba zahájit činnosti, které ve vrcholu v začínají, aby bylo možno dodržet termín $T_1(K)$ dokončení celého projektu.

Časy T_1 lze počítat podle vzorce

$$T_1(v) = \max\{T_1(\text{PV}(e)) + a(e) \mid e \in E^-(v)\} \tag{8.6}$$

přičemž časy T_1 jednotlivých vrcholů je třeba počítat v topologickém pořadí (viz 8.7.3, str. 106), neboť pro výpočet času T_1 vrcholu v potřebujeme znát časy T_1 všech jeho předchůdců. Počáteční vrchol nemá předchůdce, proto výpočet začíná konstatováním, že $T_1(P) := 0$.

Jednoduchu úvahou lze zjistit, že hodnota $T_1(v)$ je rovna délce nejdelší cesty z počátečního vrcholu P do vrcholu v.

Podobně časy T_2 lze počítat podle vzorce

$$T_2(v) = \min\{T_2(\text{KV}(e)) - a(e) \mid e \in E^+(v)\} \tag{8.7}$$

přičemž časy T_2 jednotlivých vrcholů je nutno počítat v obráceném topologickém pořadí, neboť pro výpočet času T_2 vrcholu v potřebujeme znát časy T_2 všech jeho následníků. Koncový vrchol nemá žádného následníka, proto výpočet časů T_2 začíná konstatováním, že $T_2(K) := T_1(K)$.

Jednoduchou úvahou lze zjistit, že doba $T_2(K) - T_2(v)$ je délkou nejdelší cesty z vrcholu v do koncového vrcholu K.

Po výpočtu nejdřívějších časů T_1 a nejpozdějších časů T_2 následuje v metodě CPM výpočet tzv. rezerv a nalezení kritické cesty, popř. kritických cest.

8.10.6 Rezerva činnosti e je nejvyšší přípustné zdržení (tj. prodloužení doby trvání) činnosti e, při kterém ještě lze dodržet termín $T_1(K)$ dokončení celého projektu, za předpokladu, že ostatní činnosti budou bez zdržení Rezervu $R(e)$ činnosti e lze vypočítat podle vzorce

$$R(e) = T_2(\text{KV}(e)) - T_1(\text{PV}(e)) - a(e) \tag{8.8}$$

8.10.7 Kritické činnosti a kritické cesty. Činnosti s nulovou rezervou se nazývají *kritické*. Lze ukázat, že kritické činnosti se v síťovém grafu nevyskytují osamoceně. Každá kritická činnost leží na tzv. *kritické cestě*, což je orientovaná cesta z P do K, jejíž všechny hrany (tj. činnosti) jsou kritické. Kritických cest může být v grafu několik, každá z nich je zároveň nejdelší cestou z P do K.

Kritickým činnostem se věnuje zvýšená pozornost, neboť jakékoli jejich prodloužení se projeví prodloužením doby trvání celého projektu. Při řízení složitých projektů se výpočet síťového grafu (tj. výpočty termínů T_1, T_2 a rezerv) pravidelně opakuje s údaji, které jsou aktualizovány podle skutečného průběhu činností.

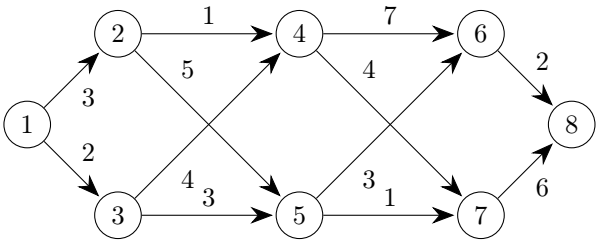
Dodejme, že specialisté na síťové grafy rozeznávají i další druhy rezerv a že pojem kritické cesty a rezervy lze zavést i pro síťové grafy s ohodnocenými vrcholy.

8.10.8 Příklad.

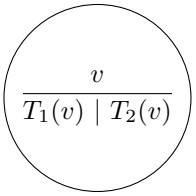
PV	KV	a
1	2	3
1	3	2
2	4	1
2	5	5
3	4	4
3	5	3

PV	KV	a
4	6	7
4	7	4
5	6	3
5	7	1
6	8	2
7	8	6

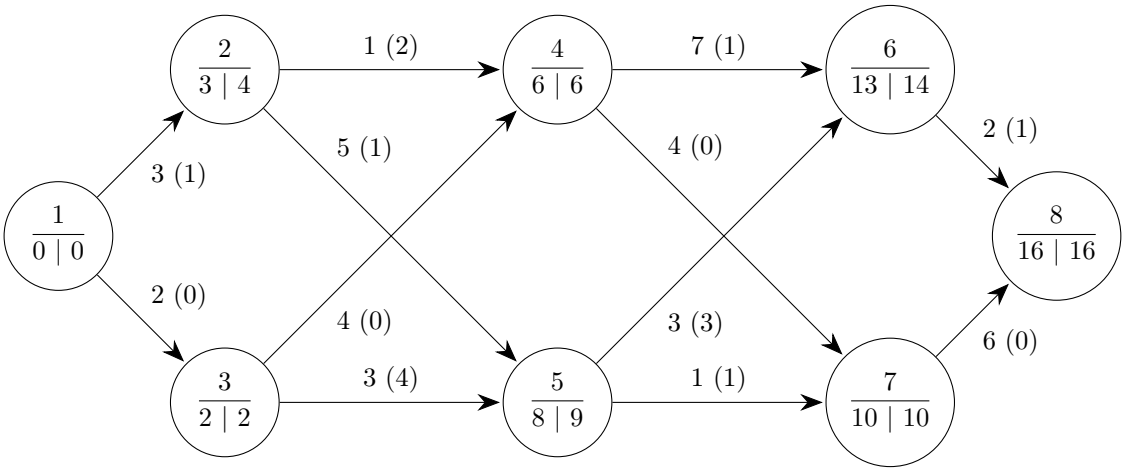
Vrcholy grafu jsou již očíslovány podle topologického uspořádání.



Časy T_1, T_2 bývá zvykem v obrázku zaznamenávat přímo u vrcholu podle tohoto schématu:



Výsledek výpočtu je zde:



Rezerva činnosti $R(5,6) = T_2(6) - T_1(5) - a(5,6) = 14 - 8 - 3 = 3$. Všimněte si, že vyčerpáním t0to rezervy, tedy prodloužením trvání činnosti na 6 časových jednotek, ovlivní rezervu následující činnosti (6,8).

Kapitola 9

Náhodná čísla a metoda Monte Carlo

9.1 Náhodné veličiny

Tato sekce nenahrazuje přednášku o teorii pravděpodobnosti, pouze shrnuje některá základní fakta, která budeme dále potřebovat.

9.1.1 Náhodný jev se v teorii pravděpodobnosti chápe jako možnost, která může nebo nemusí nastat. Je to předpověď výsledku *náhodného pokusu*,[1] která se po provedení pokusu potvrdí nebo nepotvrdí.

Házíme-li kostkou ze hry Člověče nezlob se, příklady jevů jsou: „padne lichý počet bodů“, „padne šestka“, „padne počet bodů menší než 4“. Jevem by také mohlo být „kostku už nenajdeme“, ale tuto možnost nebudeme dále uvažovat.

9.1.2 Elementární náhodný jev se v teorii pravděpodobnosti chápe jako jev, který nelze (nebo nechceme, nepotřebujeme) rozložit na dílčí „elementárnější“ jevy. Elementární jevy se navzájem vylučují a jeden z nich určitě nastane. Obecný náhodný jev lze chápat jako množinu elementárních jevů.

Házíme-li kostkou ze hry člověče nezlob se, je elementárním jevem „padne šestka“ nebo „padne dvojka“, ale nikoli jev „padne lichý počet bodů“, neboť ten je množinou elementárních jevů $\{1, 3, 5\}$.

Elementárních jevů může být konečně mnoho (jako v případě kostky), ale i nekonečně mnoho (např. teplota vyjádřená reálným číslem ve °C).

9.1.3 Jevové pole je množina jevů, tedy podmnožin množiny elementárních jevů S. Jevové pole nemusí obsahovat všechny podmnožiny množiny S, ale musí obsahovat

- *nemožný jev*, tj. prázdnou množinu $\emptyset$,
- *jistý jev*, tj. celou množinu S,
- s každým jevem A také jeho doplněk $S \setminus A$,
- s každou konečnou nebo spočetnou množinou jevů také jejich sjednocení a průnik.

Smysl jevového pole je ten, že jevům z jevového pole budeme umět přiřadit pravděpodobnosti. Některým „ošklivým“ množinám totiž pravděpodobnost rozumně přiřadit nejde.

[1] Náhodný pokus zde chápeme obecněji, než v běžném životě. Je to jakýkoli děj, proces nebo čin, jehož výsledek předem neznáme. Např. tvrzení, že se Země do 50 let srazí s asteroidem, je náhodným jevem a proces, kterého se to týká, pokládáme z hlediska teorie pravděpodobnosti za náhodný pokus, ačkoli z hlediska obecné češtiny bychom takové experimentování netolerovali.

9.1.4 Pravděpodobnost je funkce, která jevům (z nějakého jevového pole) přiřazuje reálná čísla z intervalu $\langle 0, 1\rangle$ a to tak, že

- pravděpodobnost jistého jevu S je $P(S) = 1$,
- pravděpodobnost nemožného jevu $\emptyset$ je $P(\emptyset) = 0$,
- pro každou konečnou nebo spočetnou množinu jevů $A_1, A_2, \ldots$, které jsou po dvou disjunktní, platí
$$P(\bigcup_i A_i) = \sum_i P(A_i)\ .$$

9.1.5 Jak se pravděpodobnost zjišťuje. Existují dvě základní možnosti: spekulativně (kombinatorickým výpočtem) a ze statistiky obdobných náhodných pokusů.

Spekulativní zjištění pravděpodobnosti bývá založeno na kombinatorickém výpočtu. Lze je použít v situaci, kdy elementárních jevů je konečně mnoho (řekněme n) a kdy nevidíme důvod, proč by některý elementární jev měl být pravděpodobnější než jiný. Pak všechny elementární jevy mají stejnou pravděpodonost $1/n$. Pravděpodobnosti obecných (neelementárních) jevů se pak počítají kombinatorickými metodami jako podíl počtu „příznivých“ možností a všech možností.

Tedy např. pokud nemáme podezření, že kostka je falešná, předpokládáme, že všech šest elementárních jevů má stejnou pravděpodobnst $1/6$. Pravděpodobnost jevu „padne lichý počet bodů“ pak je $3/6 = 1/2$. Pokud ovšem máme podezření, že kostka je falešná, spekulativní metodu nelze použít.

Statistické zjišťování pravděpodobnosti bývá založeno na tom, že relativní četnosti nějakého jevu se při mnoha opakováních pokusu jen málo liší od nějakého čísla a čím více pokusů provedeme, tím více se k tomuto číslu blíží. Tedy zjištěnou relativní četnost jevu A vezmeme jako pravděpodobnost $P(A)$. Statistickou metodu lze použít pouze tehdy, lze-li shromáždit data o dostatečně velkém počtu pokusů a je-li přitom oprávněný předpoklad, že podmínky těchto pokusů jsou dostatečně podobné případu, který nás zajímá.

Jsou ovšem situace, kdy nelze použít ani spekulaci ani statistiku. Někdy lze pravděpodobnost odvodit jinými úvahami, někdy ovšem nezbývá než pravděpodobnost odhadnout.

9.1.6 Nezávislost jevů. Jevy A a B nazýváme navzájem *stochasticky nezávislými*, jestliže $P(A \cap B) = P(A) \cdot P(B)$.

9.1.7 Náhodná veličina je (zhruba řečeno) veličina, která náhodně nabude nějaké číselné hodnoty.[2] Elementárními jevy jsou reálná čísla.

Informaci o tom, jak pravděpodobné jsou různé množiny hodnot (tedy jevy), nazýváme *rozdělením pravděpodobnosti*. Matematicky se rozdělění pravděpodobnosti popisuje pomocí distribuční funkce.

9.1.8 Distribuční funkce náhodné veličiny X je funkce F definovaná předpisem $F(x) = = P(X \leqq x)$. Distribuční funkce argumentu x přiřazuje pravděpodobnost, že veličina X nabude hodnoty menší nebo rovné x. **Každá náhodná veličina má distribuční funkci.**

Každá distribuční funkce má tyto vlastnosti:

- F je neklesající zprava spojitá funkce
- $lim_{x\to-\infty}F(x) = 0$,
- $lim_{x\to\infty}F(x) = 1$.

Platí to i naopak: jakákoli reálná funkce jedné proměnné, která má tyto vlastnosti, je distribuční funkcí nějaké náhodné veličiny.

[2] Toto není formální definice.

Pravděpodobnost, že veličina nabude hodnoty z polootevřeného intervalu $(a, b\rangle$ lze z distribuční funkce zjistit podle vzorce

$$P(x \in (a, b\rangle) = F(b) - F(a) \ .$$

Pozor, pro uzavřený interval $\langle a, b\rangle$ nebo pro otevřený interval (a, b) to nemusí být tak jednoduché.

9.1.9 Hustota náhodné veličiny. Má-li náhodná veličina X distribuční funkci F a existuje-li nezáporná reálná funkce f taková, že

$$F(x) = \int_{-\infty}^{x} f(t)dt \ , \tag{9.1}$$

pak funkci f nazýváme *hustotou* náhodné veličiny X.

Pozor, mnohé náhodné veličiny hustotu nemají.

Má-li náhodná veličina hustotu f, pak pravděpodobnost, že tato veličina nabude hodnoty z intervalu $\langle a, b\rangle$ (nebo (a, b)), je rovna

$$P(x \in \langle a, b\rangle) = \int_{a}^{b} f(t)dt \ .$$

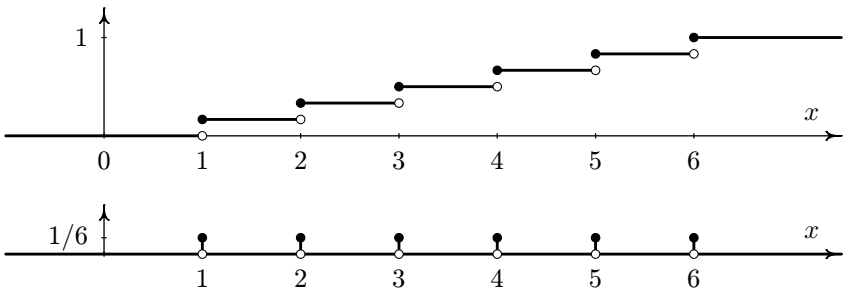
Graf hustoty (pokud hustota existuje) bývá mnohem názornější než graf distribuční funkce.

9.1.10 Spojitá náhodná veličina (též náhodná veličina spojitého typu) je taková, která má hustotu. Distribuční funkce spojité náhodné veličiny je spojitá. Střední hodnota je $E(X) = \int_{-\infty}^{\infty} xf(x)dx$

9.1.11 Diskrétní náhodná veličina je taková veličina, která nabývá pouze hodnot z nějaké konečné nebo spočetné množiny[3] $\{x_1, x_2, \ldots\}$. Distribuční funkce diskrétní náhodné veličiny je

$$F(x) = \sum_{x_i \leqq x} x_i \ .$$

Tato funkce je po částech konstantní, v každé hodnotě x_i je skok o $P(x_i)$ nahoru.



Obrázek 9.1: Distribuční a pravděpodobnostní funkce náhodné veličiny „počet bodů na kostce“.

Diskrétní náhodná veličina nemá hustotu (ve smyslu 9.1.9). Místo ní se používá tzv. *pravděpodobnostní funkce* $P(x) = P(X = x)$. Tato funkce je definována pro všechna reálná čísla a pro čísla mimo množinu $\{x_1, x_2, \ldots\}$ dává nulu.

[3] Množina je spočetná, lze-li její prvky seřadit do posloupnosti. Základním příkladem spočetné množiny je mnoožina všech přirozených čísel. Pozor, zdaleka ne každá nekonečná množina je spočetná. Slavnými příklady nespočetných množin jsou množina všech reálných čísel a množina všech podmnožin množiny přirozených čísel.

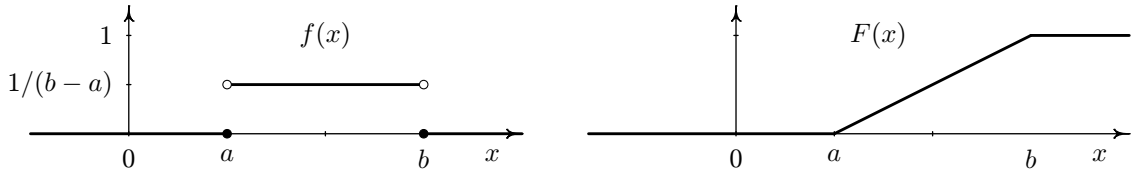
Typickými příklady diskrétních náhodných veličin jsou veličiny, které nabývají celočíselných hodnot, jako např. počet poruch nějakého zařízení nebo počet bodů na kostce od Člověče nezlob se (viz obr. 9.1).

Hodnoty diskrétní náhodné veličiny však nemusí být jen celočíselné. Příkladem je třeba teplota zaokrouhlená na desetinu stupně. nebo náhodná veličina, která nabývá pouze tří hodnot 3.14, 7.224 a 19.2.

9.1.12 Rozdělení smíšeného typu. Příklad: doba čekání ve frontě. Zpravidla je nenulová pravděpodobnost, že nebudeme čekat, tj. že budeme čekat nulovou dobu. V ostatních případech čekat budeme a doba čekání pak má zpravidla rozdělení spojitého typu.

9.1.13 Rovnoměrné spojité rozložení. Náhodná veličina X má rovnoměrné spojité rozložení v intervalu (a, b), má-li hustotu

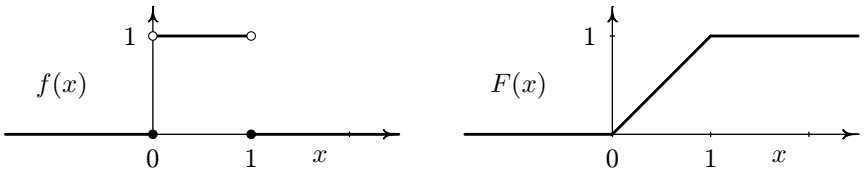
$$f(x) = \begin{cases} 1/(b-a) & \text{pro } x \in (a,b), \\ 0 & \text{jinak.} \end{cases}$$



Obrázek 9.2: Hustota a distribuční funkce rovnoměrného spojitého rozložení.

9.1.14 Standardní rovnoměrné rozložení je rovnoměrné spojité rozložení v intervalu $(0, 1)$.

Má-li veličina X standardní rovnoměrné rozložení, pak veličina $(1 - X)$ má totéž standardní rovnoměrné rozložení.



Obrázek 9.3: Hustota a distribuční funkce standardního rovnoměrného rozložení.

Většina programovacích jazyků má ve standardní knihovně podprogram pro generování pseudonáhodných čísel se standardním rovnoměrným rozložením.

9.1.15 Diskrétní rovnoměrné rozložení Náhodná veličina má diskrétní rovnoměrné rozložení v intervalu $\langle a, b\rangle$ s krokem k, nabývá-li $(n+1)$ různých hodnot z množiny $\{a+ki \mid i = 0, \dots n\}$, kde $n = (b-a)/d$, přičemž všech těchto hodnot nabývá se stejnou pravděpodobností $1/(n+1)$.

Příkladem diskrétního rovnoměrného rozložení v intervalu $\langle 1, 6\rangle$ s krokem 1 je počet bodů na kostce ze hry Člověče nezlob se. Distribuční funkce a pravděpodobnostní funkce jsou na obr. 9.1.

9.1.16 Normální rozdělení (též Gaussovo) označované symbolem $N(\mu, \sigma^2)$, kde μ je střední hodnota a σ^2 je rozptyl, má hustotu

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Název „normální“ napovídá, že mnoho náhodných veličin „ze života“ má přibližně toto rozložení. Následující větu lze chápat jako matematický pokus o vysvětlení, proč tomu tak je.

9.1.17 Centrální limitní věta. Máme-li n navzájem nezávislých náhodných veličin X_i se stejným rozložením se střední hodnotou μ a s konečným rozptylem σ^2, pak součet těchto veličin, tj. veličina $\sum_{i=1}^{n} X_i$, má pro velká n rozložení, které se blíží normálnímu rozložení $N(n\mu, n\sigma^2)$ a aritmetický průměr těchto veličin, tj. veličina $\sum_{i=1}^{n} X_i/n$, má pro velká n rozložení, které se blíží normálnímu rozložení $N(\mu, \sigma^2/n)$.

Pozoruhodné je, že předpokladům centrální limitní věty vyhovuje obrovské množství „skoro jakýchkoli“ rozdělení veličin X_i. (samozřejmě pro všechny veličiny X_i stejné). Tedy např. i aritmetický průměr veličin s rovnoměrným rozložením konverguje k rozložení normálnímu.

Další pozoruhodností je, rychlost konvergence, tj. že počet veličin n nemusí být nijak obrovský. Některé prameny uvádějí, že často stačí n kolem 30.

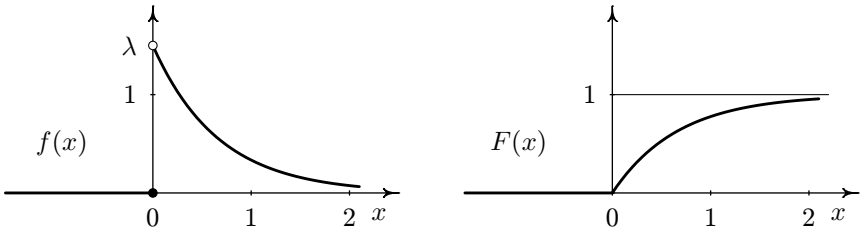
9.1.18 Exponenciální rozložení s parametrem λ má hustotu

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & \text{pro } x \geqq 0 \\ 0 & \text{pro } x < 0 \end{cases}$$

a distribuční funkci

$$F(x) = \begin{cases} 1 - e^{-\lambda x} & \text{pro } x \geqq 0 \\ 0 & \text{pro } x < 0 \end{cases} .$$

Střední hodnota je $E = 1/\lambda$ a rozptyl je $1/\lambda^2$.



Obrázek 9.4: Hustota a distribuční funkce exponenciálního rozložení.

9.1.19 Erlangovo rozložení. Náhodná veličina má Erlangovo rozložení s parametry k a λ, je-li součtem k navzájem nezávislých náhodných veličin se stejným exponenciálním rozložením s parametrem λ.

9.1.20 Poissonovo rozložení s parametrem λ je diskrétní rozdložení, které nabývá nezáporných celočíselných hodnot a má pravděpodobnostní funkci

$$p_k = P(X = k) = \frac{e^{-\lambda}\lambda^k}{k!} .$$

Střední hodnota je rovna λ.

9.2 Generování náhodných čísel

9.2.1 K čemu jsou náhodná čísla.

- Metoda Monte Carlo — náhodný experiment a jeho vyhodnocení,
- Simulace náhodných procesů — důležitý případ metody Monte Carlo,

- Kryptografie — generování šifrovacích klíčů,
- Počítačové hry — např. činnost virtuálního protihráče,
- Testování softwaru — náhodné vstupy, náhodné akce uživatele,
- Optimalizace — heuristické optimalizační algoritmy,
- Statistika — volba vzorků pro statistické průzkumy,
- eLearning — zkoušení studentů, generování úloh.

V malém měřítku a mimo počítač lze skutečně náhodná čísla získat např. házením mincí nebo kostkou ze hry "člověče nezlob se". Na počítači se to musí dělat nějak jinak.

9.2.2 Počítač versus náhoda. Od počítače (který zde chápeme dohromady se softwarem) zpravidla požadujeme striktně deterministické chování. Je-li počítač ve stejném stavu (např. po restartu) a dostane-li stejné vstupy (např. stejné stisknuté klávesy, idetické pohyby myší), požadujeme od počítače stejnou reakci. Nesplnění tohoto požadavku bývá dobrým důvodem k reklamaci.

Když od deterministického zařízení (počítače) chceme, aby se najednou a to jen ve velmi vymezené oblasti chovalo náhodně, je to těžko splnitelný požadavek. Beze zbytku mu lze vyhovět pouze tak, že k jinak deterministickému počítači připojíme zařízení (kus hardwaru), které bude fungovat jako „zdroj náhody". Hardwarové generátory ovšem nejsou běžnou výbavou počítače.

Na běžném počítači se proto ustupuje od požadavku opravdové náhodnosti a používají se generátory tzv. pseudonáhodných čísel.

9.2.3 Softwarové generátory pseudonáhodných čísel jsou zpravidla naprogramovány jako tzv. funkce, které

- jsou volány z aplikačního programu,
- vrací aplikačnímu programu hodnotu — další pseudonáhodné číslo,
- pamatují si svůj stav do příštího volání (ve vymezeném úseku paměti).

Činnost generátoru spočívá typicky v těchto krocích:

- vezme dosavadní stav,
- provedením nějakého výpočtu jej transformuje na nový stav,
- nový stav uloží pro použití při příštím vyvolání a
- ze stavu odvodí číslo, které vrátí volajícímu programu

Opakovaným voláním generátoru se jeho stav mění, takže jako výsledek dostáváme posloupnost čísel, která **vypadá jako náhodná** (i když ve skutečnosti vůbec náhodná není).

Běžné generátory produkují rovnoměrné rozložení a to buď standardní spojité v intervalu $\langle 0, 1\rangle$ nebo diskrétní. Pseudonáhodná čísla s jiným rozložením se odvozují z rozložení rovnoměrného pomocí triků, které vyložíme v 9.4.

9.2.4 Obecné vlastnosti softwarových generátorů. Především, generovaná posloupnost vůbec není náhodná, naopak, je zcela deterministická. Stejný generátor (program) spuštěný ze stejného výchozího stavu produkuje vždy stejnou posloupnost čísel.

Dále, počet stavů generátoru je konečný, neboť pro uchování stavu se používá omezený úsek paměti počítače. Z toho plyne, že generujeme-li hodně dlouhou posloupnost, stavy generátoru se dříve nebo později opakují a proto se také **periodicky opakují generovaná čísla**. Samozřejmý požadavek je, aby perioda byla co nejdelší, ale není to jediný požadavek.

9.2.5 Požadavky na posloupnosti pseudonáhodných čísel se liší podle účelu, ke kterému mají být čísla použita.

Pro simulaci náhodných procesů metodou Monte Carlo se požaduje, aby generátor co nejlépe vyhověl řadě statistických testů. Výsledky testů aplikované na skutečně vygenerovanou posloupnst by se měly co nejvíce shodovat s teoretickým chováním skutečně náhodné posloupnosti.

Pro kryptografii se navíc požaduje, aby bez znalosti algoritmu a vnitřního stavu generátoru bylo obtížné z části pseudonáhodné posloupnosti předpovědět příští pseudonáhodné číslo nebo naopak odvodit pseudonáhodná čísla, která pozorovaným číslům předcházela.

9.2.6 Statistické testy pseudonáhodných generátorů (neúplný výčet):

- střední hodnota, směrodatná odchylka, ...
- správné rozložení – test dobré shody,
- správné rozložení k-tic v k-rozměrném prostoru,
- rozložení mezer mezi sousedními výskyty čísla z daného intervalu,
- délky vzestupných (sestupných) posloupností,
- četnosti permutací k-tic
- rozložení maxima (minima) z k-tic,
- poker test,
- coupon collection test[4] — zjišťuje se, kolik je třeba v průměru vygenerovat čísel, aby se vygenerovala alespoň jedna hodnota v každém z předem daných intervalů.
- narozeninový test[5] — zjišťuje se, kolik je třeba v průměru vygenerovat čísel, aby se vygenerovaly dvě hodnoty v některém z předem daných intervalů.
- atd....

Týmž testům by měly vyhovět i podposloupnosti, které získáme výběrem z hlavní posloupnosti (třeba tak, že vezmeme každé třetí vygenerované číslo). V praxi se totiž jeden generátor používá pro několik účelů, takže pro jeden izolovaný účel se používá podposloupnost.

Příklad selhání kdysi velmi populárního generátoru RANDU lze najít ve Wikipedii na http://en.wikipedia.org/wiki/Image:Randu.png. Trojice po sobě vygenerovaných čísel zde byly použity jako souřadnice bodů v trojrozměrné krychli a tyto body jsou zobrazeny. Ideálně by vykreslené body měly být v krychli "rovnoměrně" rozptýleny, ale zde jsou soustředěny do blízkosti patnácti rovin.

Navrhnout dobrý generátor je překvapivě těžké. Sestrojit generátor, který by vyhověl všem myslitelným testům, je asi nemožné.

9.2.7 Inicializace generátoru

je nastavení generátoru do výchozího stavu. Je-li stejný generátor stejně inicializován, poskytuje tutéž posloupnost.

Možnost inicializace je klíčová pro ladění a testování počítačových programů, které pseudonáhodná čísla používají. Je totiž žádoucí, aby situace, při níž program selhal (havaroval nebo dal chybný výsledek), byla opakovatelná. Jinak by bylo velmi obtížné chybu diagnostikovat a po jejím opravení pak ověřit, zda byla chyba skutečně odstraněna. Opakovatelnosti se dosahuje pomocí stejné inicializace generátoru.

Kde vzít inicializaci? Obecně lze inicializační údaje

- „jen tak vymyslet“,
- odvodit ze systémového času počítače,
- odvodit z obtížně opakovatelných vstupů od uživatele (pohyby myší, mačkání kláves).
- odvodit z hardwarového generátoru.

Opakovatelnosti lze dosáhnout tak, že se jakkoli získaný inicializační údaj prostě zapamatuje pro příští použití.

V kryptografických aplikacích je opakovatelnost nežádoucí.

[4] Název odvozen od představy sběratele, který se snaží výběrem z náhodné posloupnost získat úplnou sadu lístků.

[5] Název je odvozen od tzv. narozeninového paradoxu — kolik se musí sejít lidí, aby pravděpodobnost, že alespoň dva z nich mají narozeniny ve stejný den, byla alespoň 0.5? Odpověď je 23, což je překvapivě málo.

9.3 Generátory rovnoměrného rozložení

Generátory rovnoměrného rozložení jsou základem generátorů jiných (obecných) rozložení. Uvedeme jen několik jednoduchých příkladů. Dodejme, že vnitřně většina generátorů pracuje s celými čísly a výsledné číslo mezi 0 a 1 se získá tak, že vygenerované celé číslo dělíme číselným rozsahem.

9.3.1 Von-Neumanův generátor byl pravděpodobně prvním softwarovým generátorem pseudonáhodných čísel.

Stavem generátoru je desetimístné celé číslo v desítkové soustavě. Transformace stavu spočívá v tom, že toto číslo umocníme na druhou (čímž dostaneme číslo zhruba dvacetimístné) a z výsledku vybereme prostředních deset číslic.

Byla to ve své době revoluční myšlenka. Bohužel, testováním se ukázalo, že generovaná posloupnost je poměrně nekvalitní.

9.3.2 Lineární kongruenční generátor. Máme celočíselné konstanty (parametry) a, c, m. Stavem generátoru je nezáporné celé číslo $X < m$. Transformace stavu generátoru se provádí podle vzorce

$$X_{n+1} := (aX_n + c) \bmod m \ ,$$

kde a, c a m jsou celočíselné parametry (konstanty) a operace mod je definována předpisem

$$a \bmod b \quad = \quad \text{zbytek při dělení } a/b.$$

Kvalita těchto generátorů je velmi závislá na volbě parametrů. Doporučuje se například, aby c bylo nesoudělné s m.

Často se používá $m = 2^w$, kde w je počet bitů počítačového slova, např. 32 nebo 64. Důvodem této volby není kvalita generátoru, ale to, že pak není třeba počítat operaci modulo, počítá se totiž „sama“ prostě tím, že bity, které se při násobení nevejdou do počítačového slova, se na většině počítačů “zapomenou” (dojde k tzv. přetečení) a to, co po přetečení „zbude“, je požadovaný zbytek při dělení.

9.3.3 Aditivní generátor. Stavem generátoru je předchozích 55 vygenerovaných celých čísel. Transformace se provádí podle vzorce

$$X_n := (X_{n-24} + X_{n-55}) \bmod m$$

9.4 Generování obecných rozložení

V této sekci budeme předpokládat, že je k disposici generátor standardního spojitého rovnoměrného rozložení. Volání (použití) tohoto generátoru budeme dále značit `random()`. Někdy je třeba i několik volání na jedno výsledné číslo. Pozor: `2*random()` není totéž co `random()+random()`, neboť

- `2*random()` generuje rovnoměrné rozložení v intervalu $(0, 2)$,
- `random()+random()` volá generátor dvakrát, výsledkem je součet dvou (obvykle) různých náhodných čísel a výsledná náhodná veličina má trojúhelníkové rozložení v intervalu $(0, 2)$.

9.4.1 Rovnoměrné spojité rozložení v intervalu (a, b) získáme výrazem `random()*(b-a)+a`. Nejprve náhodné číslo `random()` vynásobíme délkou $(b-a)$ požadovaného intervalu, čímž dostaneme náhodná čísla v intervalu 0 až $(b - a)$. Výsledek pak na reálné ose posuneme přičtením konstanty a na požadovaný interval (a, b).

9.4.2 Rovnoměrné diskrétní rozložení získáme ze spojitého rozložení zaokrouhlením.

Náhodná celá čísla v rozsahu a až b včetně získáme výrazem `int(random()*(b-a+1))+a`, kde funkce `int()` provádí zaokrouhlení dolů na nejbližší celé číslo.

Tedy např. celá čísla 0, ..., 10 získáme výrazem `int(random()*11)`. (Zdůvodněte, proč 11, když maximum má být 10?)

Podobně čísla do Sportky (tj. celá čísla 1 až 49) lze generovat výrazem `int(random()*49)+1`. (Zdůvodněte, proč násobíme 49 a ne 48.)

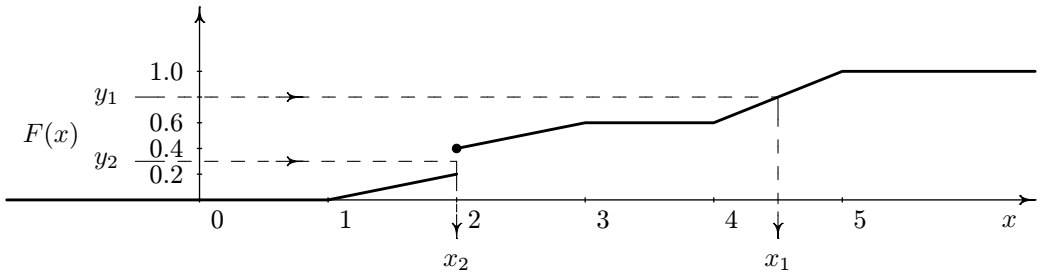
9.4.3 Metoda inverzní transformace je univerzálně použitelná pro generování libovolné náhodné veličiny s distribuční funkcí F za předpokladu, že umíme počítat inverzní funkci F^{-1}. Pro tyto účely definici inverzní funkce F^{-1} oproti běžnému chápání poněkud rozšíříme:

$$F^{-1}(y) = \min\{x \mid F(x) \geqq y\} \ .$$

Takto je inverzní funkce F^{-1} dobře definovaná i pro nespojitou distribuční funkci se skoky a konstantními částmi.

Metoda inverzní transformace je překvapivě jednoduchá: výraz F^{-1}(`random()`) má rozložení s distribuční funkcí F.

Vskutku, vezmeme-li libovolný neprázdný interval $(a, b\rangle$, pak pravděpodobnost, že náhodná veličina s distribuční funkcí F nabude hodnoty z tohoto intervalu, je rovna $P(a < X \leqq b) = F(b) - F(a)$ a to je přesně pravděpodobnost, že vygenerované náhodné číslo se standardním rovnoměrným rozložením nabude hodnoty z intervalu $(F(a), F(b)\rangle$.



Obrázek 9.5: Metoda inverzní transformace.

9.4.4 Empirické diskrétní rozložení. Hodnoty $h_1, \ldots, h_k$, která se mají vyskytovat s pravděpodobnostmi $p_1, \ldots, p_k$, lze generovat takto:

Interval $(0, 1)$ rozdělíme na k menších disjunktních intervalů o délkách $p_1, \ldots, p_k$ a každému z těchto intervalů přiřadíme výslednou hodnotu $h_1, \ldots, h_k$. Když pak vygenerujeme náhodné číslo `random` v intervalu $(0, 1)$, zjistíme do kterého z k dílčích intrervalů toto číslo padlo a vydáme příslušnou hodnotu:

```
x := random();
i := 0;
repeat
  i := i + 1;
  x := x - p[i];
until x <= 0.0;
return h[i];
```

Jinou možností je použít metodu inverzní transformace. Distribuční funkce je po částech konstantní se skokem velikosti p_i pro každou hodnotu h_i.

9.4.5 Exponenciální rozložení lze generovat metodou inverzní transformace, konkrétně výrazem `-ln(random()) / lambda`, kde `ln()` značí přirozený logaritmus (tj. logaritmus o základu $e \doteq 2.71828$).

Připomeňme, že distribuční funkce exponenciálního rozložení je pro $x \geqq 0$ dána vzorcem: $F(x) = 1 - e^{-\lambda x}$. Inverzní funkce je $F^{-1}(r) = \ln(1-r)/-\lambda$.

Kontrolní otázka: Není zde chyba? Neměl by se pro generování raději používat výraz `-ln(1.0-random()) / lambda`?

9.4.6 Vylučovací metoda je použitelná pro generování veličin s rozložením, které je dáno hustotou f, přičemž

- umíme počítat hodnotu hustoty $f(x)$,
- hustota je nulová mimo omezený interval $\langle a, b\rangle$, tedy $f(x) = 0$ pro všechna $x \notin \langle a, b\rangle$,
- hustota je shora omezena konstantou M, tedy $f(x) < M$ pro všechna x.

```
repeat
  x := random() * (b-a) + a;
  y := random() * M;
until y < f(x);
return x;
```

Vygenerujeme čísla $x \in (a, b)$ a $y \in \langle 0, M)$, obě s rovnoměrným rozložením. Jestliže $y < f(x)$, pak číslo x vydáme jako výsledek (tedy pseudonáhodné číslo s požadovaným rozložením). Jestliže naopak $y \geqq f(x)$, pak celý postup opakujeme, tj. znovu vygenerujeme x a y.

Zdůvodnění, že výsledná hodnota má požadované rozložení, se opírá o fakt, že hodnota x je potvrzena (a vygenerována jako výsledek) s pravděpodobností, která je úměrná hodnotě hustoty $f(x)$.

Je zřejmé, že než dostaneme nějaký výsledek, může se vygenerovat někdy i hodně velký počet dvojic x a y.

9.4.7 Normální rozložení lze generovat snadno zapamatovatelnou metodou. V teorii pravděpodobnosti tzv. centrální limitní věta 9.1.17 říká, že aritmetický průměr n navzájem nezávislých náhodných veličin se stejným rozložením, které splňuje velmi obecné předpoklady, např. má konečný rozptyl, konverguje pro $n \to \infty$ k normálnímu rozložení.

Náhodná čísla s normovaným normálním rozložením $\mathrm{N}(0, 1)$ (tj. se střední hodnotou 0 a s rozptylem 1) lze přibližně generovat jako součet dvanácti náhodných čísel s rovnoměrným rozložením v intervalu $(-0.5, 0.5)$. (Dvanácti proto, že rovnoměrné rozložení na intervalu délky 1 má rozptyl $1/12$ a rozptyly se sečítají.) Příslušný úsek programu je

```
S := -6.0;
for i:=1 to 12 do S := S + random();
return S;
```

Požadujeme-li větší přesnost, můžeme počítat průměr z většího počtu náhodných čísel s rovnoměrným rozložením a výsledek upravit, aby měl požadovaný rozptyl, např. takto:

```
S := -24.0;
for i:=1 to 48 do S := S + random();
return S / 2.0;
```

Dokonalejší a rychlejší generátor normálního rozložení využívá trik podobný vylučovací metodě a výsledek transformuje. Matematické zdůvodnění přesahuje rámec tohoto textu.

```
repeat
  v1 := 2*random() - 1.0;
  v2 := 2*random() - 1.0;
```

```
  S := v1*v1 + v2*v2;
 until S < 1.0; {bod (v1,v2) leží v jednotkovém kruhu}
 return v1 * sqrt(-2.0 * ln(s) / s);
```

9.4.8 Poissonovo rozložení. Poissonovo rozložení lze generovat podle poznámky uvedené v **??**.

Náhodná veličina s Poissonovým rozložením je tedy rovna počtu, kolik je třeba sečíst nezávislých náhodných veličin s exponenciálním rozložením s parametrem λ, aby součet přesáhl jedničku. Náhodné číslo s exponenciálním rozložením se podle 9.4.5 získává logaritmováním čísla s rovnoměrným rozložením (viz 9.4.5). Postup lze zjednodušit tím, že budeme náhodná čísla z intervalu $(0, 1)$ násobit namísto sečítání jejich logaritmů. Celý algoritmus vypadá takto:

```
x := -1;
a := exp(-lambda);
S := 1.0;
 repeat
  S := S * random();
  x := x + 1;
 until S < a;
 return x;
```

9.4.9 Cvičení. Napište úsek programu, který generuje náhodná čísla s diskrétním rovnoměrným rozložením na množině $\{7, 9, 11, 13\}$.

9.4.10 Cvičení. Napište úsek programu, který generuje náhodná čísla s diskrétním rovnoměrným rozložením na množině $\{7, 9, 10, 11, 15\}$.

9.4.11 Cvičení. Napište úsek programu, který generuje náhodná čísla s normálním rozložením se střední hodnotou 7.5 a rozptylem 3.

9.4.12 Cvičení. V některých společenských hrách se ke generování náhodných čísel používají speciální kostky — desetistěnná, dvanáctistěnná a dvacetistěnná. Navrhněte co nejjednodušší způsob, jak použití těchto kostek nahradit opakovaným házením běžnou (šestistěnnou) kostkou ze hry „Člověče nezlob se“. Pozor: je třeba zajistit, aby výsledná náhodná veličina měla rovnoměrné rozložení.

9.4.13 Cvičení. Navrhněte způsob, jak běžnou kostku z „Člověče, nezlob se“ nahradit házením mincí.

9.4.14 Cvičení. Napište funkci, která generuje náhodná čísla s diskrétním rozložením, které nabývá hodnot $h_1, \ldots, h_n$ s pravděpodobnostmi $p_1, \ldots, p_n$. Předpokládejte, že hodnoty h_i a pravděpodobnosti p_i jsou předávány jako parametry typu pole.

9.5 Metoda Monte Carlo

Metoda Monte Carlo spočívá v provádění náhodných experimentů a jejich statistickém vyhodnocení. Náhodný experiment může ale nemusí být simulační.

9.5.1 Určení čísla π házením jehly je lehce kuriózní, zcela nepraktickou, ale poučnou ukázkou metody Monte Carlo.

Náhodný experiment spočívá v házení jehly na linkovaný papír. Lze matematicky dokázat, že pokud se délka jehly rovná vzdálenosti mezi linkami, je pravděpodobnost, že jehla protne některou linku, rovna $2/\pi$.

Provedeme-li tedy n experimentů (tj. n-krát hodíme jehlu), přičemž k z těchto pokusů dopadne tak, že jehla protne linku, pak k/n bude odhad pravděpodobnosti, že jehla protne linku. Tedy $k/n \approx 2/\pi$, tedy číslo $2n/k$ je statistickým odhadem čísla π. Přesnost závisí na počtu pokusů a nutně bude vztažena k nějaké hladině pravděpodobnosti.

Je třeba zdůraznit, že existují mnohem přesnější a efektivnější metody, jak počítat číslo π s libovolnou přesností a bez statistických chyb.

9.5.2 Přibližný výpočet integrálu. Numerická integrace funguje dobře v prostorech nízké dimenze (nejlépe v prostoru dimenze 1). V mnohorozměrných prostorech se stává výpočetně nesmírně náročnou. Metodou Monte Carlo lze integrál *přibližně* počítat jako aritmetický průměr funkčních hodnot v náhodně vygenerovaných bodech.

9.5.3 Podpora rozhodování. Hodnocení alternativ, mezi kterými se rozhodujeme, je často závislé na mnoha náhodných veličinách, často komplikovaně definovaných. Metoda Monte Carlo je poměrně jednoduchým a univerzálním nástrojem pro vyhodnocení.

9.5.4 Simulace náhodných procesů je nejčastějším použitím metody Monte Carlo. Viz následující kapitola.

Kapitola 10

Náhodné procesy

10.1 Náhodné procesy obecně

10.1.1 Náhodný proces je zobrazení $P : T \to S$, které hodnotě $t \in T$, kterou je nejčastěji čas, přiřazuje náhodný jev, nejčastěji náhodnou veličinu (jednorozměrnou nebo vícerozměrnou).

Pro minulé časové okamžiky zpravidla víme, v jakém stavu proces byl, jaká byla historie procesu. Pro budoucí okamžiky stav procesu neznáme a pokládáme jej za náhodný jev.

Většinu dějů „ze života“ lze modelovat jako náhodné procesy.

U náhodného procesu nás může zajímat a více či méně úspěšně lze zjišťovat

- předpověď budoucího chování procesu na základě znalosti jeho typu, parametrů a případně minulého průběhu, Příkladem může být předpověď počasí, předpověď vývoje kurzu akcií, předpověď poptávky po nějakém typu zboží.

- dlouhodobé charakteristiky průběhu procesu na základě znalosti typu a parametrů procesu. Příkladem může být průměrná délka fronty, rozložení doby čekání na úřadě apod.

Řešení těchto otázek lze získat buď analyticky, tj. výpočtem za vydatné pomoci teorie náhodných procesů a nebo simulací (napodobením) procesu metodou Monte Carlo (viz kapitola 13, str. 151).

10.1.2 Příklad – hazardní hra. Představte si jednoduchou hazardní hru dvou hráčů. Oba hráči mají dohromady K korun. Jedno kolo hry spočívá v tom, že oba hráči hodí kostkou. Komu padne na kostce méně bodů než protihráči, ten zaplatí druhému korunu. Kdo nemá na zaplacení, ten celkově prohrál a hra tím končí.

Za stav procesu zde můžeme pokládat počet korun, které má jeden z hráčů. Stavový prostor tedy bude množina $\{0, 1, \ldots, K\}$. V každém kole hry se stav procesu zvětší o 1, zmenší o 1, nebo zůstane stejný.

10.1.3 Stavový prostor náhodného procesu je jevové pole, tj. množina stavů, kterých může proces nabýt.

Stav procesu může být vyjádřen jednorozměrnou náhodnou veličinou, může to být vícerozměrná náhodná veličina nebo to dokonce může být obecná množina.

Stav procesu nazýváme diskrétním, když není limitou posloupnosti jiných stavů. Typickými případy diskrétních stavů jsou stavy vyjádřené celočíselnými náhodnými veličinami nebo veličinami zaokrouhlenými na nějaý počet desetinných míst. Také jsou-li stavy procesu obecnými prvky nějaké obecné množiny, která není nijak strukturována, i zde jde o diskrétní stavový prostor.

10.1.4 Procesy se spojitým časem jsou takové, kde stavy procesu uvažujeme ve všech okamžicích vyjádřených reálným číslem. Mezi každými dvěma okamžiky je nekonečně mnoho různých okamžiků. Změna stavu procesu je myslitelná kdykoli, tj. V jakémkoli okamžiku.

I když je čas spojitý, změny stavu spojité být mohou a nemusí. Příkladem je proces přicházení zákazníků do obchodu. Zákazník může přijít kdykoli, ale počet zákazníků je vždy celé číslo.

10.1.5 Procesy s diskrétním časem jsou takové, kde ke změnám stavu dochází pouze v posloupnosti diskrétních okamžiků $t_0, t_1, t_2, \ldots$ Mezi těmito okamžiky se stav procesu nemění (nic se neděje). U procesu s diskrétním časem má smysl mluvit o bezprostředně následujícím časovém okamžiku.

Diskrétní čas můžeme dostat dvěma základními způsoby. Často úmyslně omezíme svůj zájem pouze na diskrétní okamžiky (např. budeme měřit teplotu nikoli kontinuálně, ale jen v každou celou hodinu, nebo HDP zjišťujeme vždy za rok).

Druhá možnost, jak můžeme dostat diskrétní čas, je odvodit časové okamžiky od nějakých událostí, např. od změn stavu procesu. Příkladem je hazardní hra z příkladu 10.1.2, kde za příští okamžik můžeme vzít dobu, kdy oba hráči hodí kostkou, nebo dobu, kdy dojde ke změně stavu (což není totéž).

10.1.6 Příklady.

Spojitý čas, spojitý stavový prostor – průběh venkovní teploty, Brownův pohyb.

Spojitý čas, diskrétní stavový prostor – počet zákazníků v obchodě.

Diskrétní čas, spojitý stavový prostor – teplota každý den v 7 hodin ráno.

Diskrétní čas, diskrétní stavový prostor – teplota každý den měřená s přesností na desetinu stupně, ale také hazardní hra z příkladu 10.1.2.

10.1.7 Homogenní náhodný proces je takový, jehož pravděpodobnostní charakteristiky se v čase nemění. Tedy například pravděpodobnost, že dojde k události během minutového intervalu, nezávisí na tom, kde je tento interval umístěn na časové ose (tj. např. na tom, kolik je zrovna hodin nebo které je roční období).

Je-li náhodný proces homogenní, velice to zjednodušuje jeho analýzu. V praxi je jen málo procesů naprosto homogenních. Ve vhodně zvoleném časovém intervalu však předpoklad homogenity může být přijatelný.

10.1.8 Čítací proces je náhodný proces, který je tvořen událostmi stejného typu. Poněvadž všechny události jsou stejné, je na čítacím procesu zajímavé pouze to, jak jsou události rozloženy v čase, tedy např. kolik událostí nastalo v závislosti na čase.

Čítací procesy mají spojitý čas a diskrétní stavový prostor.

Příklady čítacích procesů: Příchody zákazníků do obchodu, otřesy zemské kůry, poruchy stroje.

10.1.9 Náhodný proces bez paměti je takový náhodný proces, kde budoucí výskyt události je stochasticky nezávislý na předchozí historii procesu. To mimo jiné znamená, že znalost historie procesu je pro předpovídání budoucího průběhu zcela bezcenná.

Příklad: opakovaně házíme kostkou, stavem procesu je počet bodů v naposledy provedeném hodu. Tento proces je homogenní a bez paměti. To, že vám už stokrát nepadla šestka nijak nezvyšuje ani nesnižuje pravděpodobnost, že v příštím hodu šestka padne.

10.2 Poissonův proces

Poissonův proces je matematickým vyjádřením představy „čistě náhodného výskytu událostí“.

10.2.1 Poissonův proces je náhodný proces, ktetrý je čítací, homogenní a bez paměti.

Tyto tři vlastnosti definují Poissonův proces jednoznačně až na jediný parametr, totiž jeho intenzitu (průměrný počet událostí za jednotku času). Jinými slovy, dva Poissonovy procesy jsou stejné právě tehdy, když mají stejnou intenzitu.

Praktičtější definice jsou uvedeny v 10.2.7.

10.2.2 Intenzita Poissonova procesu je průměrný počet událostí za jednotku času.

Pozor, nezapomeňte na slovo *průměrný*. Skutečný počet událostí za jednotku času je náhodná veličina s poissonovým rozložením, viz 9.1.20.

10.2.3 Doba čekání na nejbližší událost v Poissonově procesu má exponenciální rozložení.

Důkaz: Označme $G(x)$ pravděpodobnost, že během časového intervalu o délce x nedojde v Poissonově procesu k žádné události. Toto označení má dobrý smysl: je-li proces Poissonův, pak zmíněná pravděpodobnost opravdu nezáleží na ničem jiném než na délce intervalu. Poněvadž je proces bez paměti, pravděpodobnost nezávisí na předchozí historii procesu a z homogennosti plyne, že pravděpodobnost nezávisí ani na umístění intervalu na časové ose.

Nyní vezměme dva libovolné na sebe navazující časové intervaly o délkách s a t. Pravděpodobnosti $G(s)$ a $G(t)$, že během nich nedojde k žádné události, jsou stochasticky nezávislé (poněvadž proces je bez paměti). Proto pro pravděpodobnost $G(s+t)$, že nedojde k žádné události během intervalu o délce $s+t$ platí

$$G(s+t) = G(s) \cdot G(t)$$

V matematické analýze je dokázáno, že této rovnici vyhovuje jediný typ funkce, totiž funkce exponenciální. Funkce $G(x)$ tedy má tvar $G(x) = e^{-\lambda x}$, kde λ je nějaká kladná konstanta.

Zkusme nyní čekat na nejbližší další událost. Čekání zahájíme v čase $t = 0$, náhodnou veličinu "doba čekání" označme X. O distribuční funkci F této veličiny pro $x > 0$ platí $F(x) = P(X \leqq x) = 1 - G(x)$, neboť jev „událost nastane nejpozději v čase x" je doplňkem jevu „událost nenastane v intervalu $(0, x\rangle$", tj. jevu „událost nastane později než v čase x". Platí tedy

$$F(x) = P(X \leqq x) = \begin{cases} 1 - G(x) = 1 - e^{-\lambda x} & \text{pro } x > 0\,, \\ 0 & \text{pro } x \leqq 0\,, \end{cases}$$

což je vzorec distribuční funkce exponenciálního rozložení (viz 9.1.18). □

10.2.4 Intervaly mezi událostmi Poissonova procesu jsou navzájem stochasticky nezávislé a mají všechny stejné exponenciální rozložení. Parametrem tohoto rozložení je intenzita procesu λ.

Že jde o exponenciální rozložení, triviálně vyplývá z 10.2.3, význam parametru λ pak vyplývá ze vzorce pro střední hodnotu exponenciálního rozložení.

10.2.5 Počet událostí Poissonova procesu za jednotku času má Poissonovo rozložení s parametrem λ rovným intenzitě procesu. Pravděpodobnost p_k, že během intervalu jednotkové délky dojde k přesně k událostem, je

$$p_k = \frac{e^{-\lambda}\lambda^k}{k!}\,.$$

Střední počet událostí za jednotku času je samozřejmě rovem λ.

10.2.6 Počet událostí Poissonova procesu v obecném časovém intervalu délky t má rovněž Poissonovo rozložení, ovšem s parametrem λt. Tedy pravděpodobnost, že dojde k přesně k událostem během intervalu délky t, je

$$p_k = \frac{e^{-\lambda t}(\lambda t)^k}{k!}\,.$$

10.2.7 Alternativní definice Poissonova procesu.

- Čítací proces je Poissonův právě tehdy, když je homogenní a intervaly mezi událostmi jsou stochasticky nezávislé a mají všechny stejné exponenciální rozložení.
- Čítací proces je Poissonův právě tehdy, když je homogenní a počty událostí v časovém intervalu mají Poissonovo rozložení.
- Čítací proces je Poissonův právě tehdy, když pravděpodobnost p_Δ, že nastane událost v malém časovém intervalu délky Δ, dělená délkou tohoto intervalu Δ, konverguje pro $\Delta \to 0_+$ k intenzitě procesu λ. (Jinak řečeno, pro malé Δ platí $p_\Delta \approx \lambda\Delta$.)

Tyto alternativní definice (zejména první z nich) jsou užitečné při statistickém testování, zda nějaký proces, který pozorujeme, lze pokládat za proces Poissonův.

10.2.8 Praktické příklady Poissonova procesu. Největší praktický význam je dvojí

- Poissonovým procesem lze modelovat výskyty poruch u zařízení, u nichž opotřebení hraje zanedbatelnou roli a poruchy mají jiné příčiny. Typickým příkladem jsou elektronická zařízení s minimem pohyblivých součástí.

  V technických specifikacích různých výrobků bývá místo intenzity poruch uváděn údaj se zkratkou MTBF (Mean Time Between Failures), což je převrácená hodnota intenzity procesu.
- V teorii front se Poissonův proces často vyskytuje jako vstupní proces, tj. proces přicházení zákazníků.

Dalšími příklady jsou rozpady částic ve vzorku radioaktivního materiálu nebo průlety částic kosmického záření.

Obecně lze Poissonův proces očekávat tam, kde události stejného typu nastávají jednotlivě a přitom nezávisle na sobě navzájem i nezávisle na čase.

10.2.9 Příklad. Předpokládejme, že poruchy stroje tvoří Poissonův proces, střední doba mezi poruchami (MTBF) je 80 provozních hodin. Předpokládáme 8 hodin provozu denně. Máme tři náhradní díly, další dostaneme až za 5 dní. Jaká je pravděpodobnost, že zásoba náhradních dílů nebude stačit.

Za časovou jednotku zvolme 5 dní. Intenzita procesu poruch pak je 1/2. Pravděpodobnost, že náhradní díly nebudou stačit, je rovna pravděpodobnosti, že v Poissonově procesu dojde během časové jednotky (5 dní) ke čtyřem nebo více poruchám. Ze vzorce pro Poissonovo rozložení 9.1.20 dostaneme $p_0 = 0.60653, p_1 = 0.30327, p_2 = 0.07582, p_3 = 0.01264$. Pravděpodobnost, že zásoba nebude stačit, je rovna $1 - p_0 - p_1 - p_2 - p_3 = 1 - 0.60653 - 0.30327 - 0.07582 - 0.01264 = 1 - - 0.99825 = 0.00175$.

Všimněte si, že v této úloze jsme čas uvažovali jako dobu provozu. To mj. znamená, že možnost vzniku poruchy, když je stroj mimo provoz, tedy např. vlivem stáří, zde zanedbáváme.

10.2.10 Cvičení. Předpokládejme, že poruchy pneumatiky tvoří Poissonův proces vzhledem k ujeté vzdálenosti (tedy „čas“ Poissonova procesu zde měříme jako ujetou vzdálenost). Průměrně nastane jedna porucha pneumatiky po 50 000 km jízdy. V autě máme jedno rezervní kolo. Jaká je pravděpodobnost, že kvůli poruchám pneumatik nedojedeme do cíle vzdáleného 5 000 km?

10.2.11 Generování Poissonova procesu. Časové intervaly mezi událostmi Poissonova procesu mají (viz 10.2.4) exponenciální rozložení s parametrem λ, kde λ je intenzita procesu.

Stačí tedy generovat náhodná čísla s exponenciálním rozložením (podle 9.4.5). Čas, kdy nastane další událost dostaneme tak, že k času, kdy nastala událost předchozí, přičteme vždy další z vygenerovaných čísel s exponenciálním rozlořením.

Kapitola 11

Markovské řetězce

11.1 Základní pojmy

11.1.1 Markovský řetězec je náhodný proces s diskrétní množinou stavů, diskrétním časem a takový, že pravděpodobnost $p_i^{(t)}$, že v čase t bude proces ve stavu i, je stochasticky závislá pouze na stavu v předchozím okamžiku, tj. na stavu v čase $t-1$.

Budeme se zabývat pouze tzv. *homogenními* markovskými řetězci, tj. takovými, kde výše zmíněné pravděpodobnosti nezávisí na čase t.

A dále, budeme se zabývat pouze konečnými markovskými řetězci, tj. takovými, jejichž množina stavů je konečná. Často přitom budeme předpokládat, že stavy máme očíslovány přirozenými čísly $1, 2, 3, \ldots$

11.1.2 Diskrétní čas. Předpoklad, že v markovském řetězci je čas diskrétní, znamená, že nás stavy procesu zajímají pouze v okamžicích, které tvoří (potenciálně nekonečnou) rostoucí posloupnost. Mezi těmito okamžiky se v markovském řetězci nic neděje, čas mezi těmito okamžiky v modelu neexistuje.

Zejména to znamená, že pro okamžik t má smysl hovořit o *následujícím* časovém okamžiku – budeme jej značit jako okamžik $t+1$. Pro spojitý čas toto neplatí: ve spojitém čase mezi každými dvěma různými okamžiky t_1, t_2 leží nekonečně mnoho okamžiků t takových, že $t_1 < t < t_2$.

Při praktickém modelování nějakého děje pomocí markovského řetězce jsou okamžiky t_i zpravidla odvozeny od výskytu nějaké události. Často touto událostí bývá změna stavu řetězce, to odpovídá situaci, kdy nás zajímají jen změny stavů a nikoli doby, za jak dlouho ke změně stavu dochází. Okamžiky t_i však mohou být odvozeny i od jiných událostí, než pouze od změn stavu.

Jiný, rovněž častý, způsob modelování spočívá v tom, že stavy modelovaného děje sledujeme se zcela pravidelným časovým krokem, např. každých 5 milisekund nebo každého prvního v měsíci. To pak znamená, že ve skutečném ději (v ději, který modelujeme) může během časového kroku dojít i k několika změnám stavu, ale model (markovský řetězec) tyto změny nedokáže zachytit. Tento způsob modelování však na rozdíl od předchozího umožňuje modelovat dobu, která uplyne mezi změnami stavů nebo než je dosaženo nějakého cílového stavu.

11.1.3 Pravděpodobnostní vektor v čase t. Svou znalost (zpravidla neúplnou) o tom, ve kterém stavu se nacházel, nachází nebo bude nacházet markovský řetězec v čase t, budeme vyjadřovat jako pravděpodobnostní vektor

$$p^{(t)} = (p_1^{(t)}, p_2^{(t)}, \ldots, p_n^{(t)}) \,,$$

kde n je počet stavů markovského řetězce.

Pokud s jistotou víme, že v čase t řetězec byl ve stavu i, pak jde o speciální případ, kdy $p_i^{(t)} = 1$ a ostatní pravděpodobnosti $p_j^{(t)}$ pro $j \neq 0$ jsou nulové.

Pravděpodobnostní vektor má vždy všechny složky nezáporné a jejich součet je roven jedné. Pravděpodobnostní vektor budeme pokládat za vektor řádkový (to kvůli pohodlnému násobení tzv. přechodovou maticí).

11.1.4 Pravděpodobnosti přechodu, přechodová matice. Označme $p_{i,j}$ podmíněnou pravděpodobnost, že markovský řetězec bude v čase t ve stavu j, za podmínky, že v předchozím okamžiku $t-1$ byl ve stavu i.

Máme-li stavy pevně očíslovány $1, 2, \ldots, n$, pak pravděpodobnosti $p_{i,j}$ můžeme uspořádat do čtvercové matice, kterou nazýváme *přechodovou maticí*.

Přechodová matice má všechny prvky nezáporné a součet prvků v libovolném řádku je roven jedné. Její řádky jsou tedy pravděpodobnostní vektory ve smyslu 11.1.3.

Naopak, každá čtvercová matice, jejíž všechny prvky jsou nezáporné a součty řádků jsou rovny jedné, je přechodovou maticí nějakého markovského řetězce. Taková matice se nazývá *stochastická*.

11.1.5 Příklad – hazardní hra. Dva hráči mají dohromady 3 Kč. Hra se skládá z posloupnosti partií, v každé partii se hraje o jednu korunu: ten, kdo partii prohrál, musí zaplatit vítězi 1 Kč. Když hráč prohrál partii a nemá na zaplacení, pak tento hráč prohrál celou hru a hra tím končí. Všechny partie se hrají stejným způsobem a spočívají v tom, že oba hráči hodí kostkou a komu na kostce padne méně bodů, ten prohrál a zaplatí 1 Kč. Pokud oběma hráčům padne stejný počet bodů, je výsledek partie nerozhodný a nikdo nic neplatí a stav hry se nemění.

Na této hře nás může zajímat například to, jak závisí pravděpodobnost celkové prohry na výchozím stavu, tj. na tom, s kolika penězi náš hráč začínal hrát. Dále by nás mohlo zajímat, jaký bude střední počet partií než hra skončí, jaká je pravděpodobnost, že po pěti partiích již bude hra skončena nebo jaká je pravděpodobnost, že po čtyřech partiích bude náš hráč mít přesně 2 Kč.

Tuto hru lze modelovat markovským řetězcem o šesti stavech. Čtyři stavy odpovídají stavům financí jednoho z hráčů (0, 1, 2, 3), další dva stavy odpovídají celkové prohře a celkové výhře. Čas je v tomto modelu určen událostí, totiž sehráním další partie. Přechodová matice má tvar

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 5/12 & 1/6 & 5/12 & 0 & 0 & 0 \\ 0 & 5/12 & 1/6 & 5/12 & 0 & 0 \\ 0 & 0 & 5/12 & 1/6 & 5/12 & 0 \\ 0 & 0 & 0 & 5/12 & 1/6 & 5/12 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

11.1.6 Příklad – táž hazardní hra trochu jinak. Jiný model téže hry (tj. jiný markovský řetězec) dostaneme, když nerozhodné partie budeme pokládat za neplatné (nepodařené) a nebudeme je počítat. Tento markovský řetězec bude mít přechodovou matici

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

V obou modelech lze rovnocenným způsobem spočítat pravděpodobnost celkové prohry. Průměrný počet partií než hra skončí bude ovšem v obou modelech různý.

11.1.7 Přechodový graf markovského řetězce je orientovaný graf, jehož vrcholy odpovídají stavům markovského řetězce a orientované hrany odpovídají možným přímým změnám stavů, tj. změnám, které mají nenulovou pravděpodobnost. Každá hrana z vrcholu (stavu) i do vrcholu (stavu) j je ohodnocena pravděpodobností přechodu $p_{i,j} > 0$. Počet hran je tedy roven počtu nenulových prvků přechodové matice. Z každého vrcholu vychází alespoň jedna hrana a součet ohodnocení všech hran, které z vrcholu vycházejí, je roven jedné.

Graf markovského řetězce je důležitý nejen pro svou názornost, ale zejména proto, že mnoho důležitých vlastností markovského řetězce lze snadno odvodit z vlastností jeho grafu, zejména z jeho rozkladu na silně souvislé komponenty.

11.2 Jednoduché výpočty

11.2.1 Výpočet pravděpodobnostních vektorů. Známe-li přechodovou matici P a pravděpodobnostní vektor pro nějaký časový okamžik t, lze pravděpodobnosti jednotlivých stavů pro následující časový okamžik $t+1$ počítat podle vzorce

$$p_j^{(t+1)} = \sum_{i=1}^{n} p_i^{(t)} \cdot p_{i,j} \ .$$

Celý pravděpodobnostní vektor $p^{(t+1)}$ tedy lze získat z pravděpodobnostního vektoru $p^{(t)}$ násobením přechodovou maticí zprava:

$$p^{(t+1)} = p^{(t)} \cdot P \ .$$

Stejným postupem lze postupně počítat další a další pravděpodobnostní vektory. Obecně platí

$$p^{(t)} = p^{(0)} \cdot P^t \ ,$$

Výraz P^t na pravé straně rovnice je t-tá mocnina přechodové matice, tj. součin t stejných přechodových matic P.

11.2.2 Pravděpodobnosti přechodu za více kroků. Označme $p_{i,j}^{(t)}$ pravděpodobnost, že řetězec přejde ze stavu i za *přesně* t časových kroků do stavu j. Pro pevný počet časových kroků t můžeme tyto pravděpodobnosti uspořádat do čtvercové matice, označme ji $P^{(t)}$.

Přechodová matice P je pak speciálním případem, kde je počet časových kroků $t = 1$.

Platí

$$P^{(t)} = P^t \ .$$

11.2.3 Příklad. V markovském řetězci z příkladu 11.1.5 předpokládejme, že náš hráč začíná ve stavu 3, tj. na začátku má 1 Kč. S jakou pravděpodobností bude mít po čtyřech partiích přesně 3 Kč? Kolika způsoby může tohoto stavu dosáhnout?

Výchozí pravděpodobnostní vektor je

$$p^{(0)} = (0, 0, 1, 0, 0, 0) \ .$$

Pravděpodobnostní vektor $p^{(4)}$ bychom mohli zjistit tak, že bychom vektor $p^{(0)}$ čtyřikrát vynásobili maticí P zprava a z výsledného vektoru bychom vzali pátou složku, tj. $p_5^{(4)}$. To je postup vhodný pro počítač (nebo pro kalkulačku, která umí násobit matice). Pro ruční výpočet je to poněkud pracné a nepohodlné.

Jiný způsob by mohl být tento: Napíšeme seznam všech možností, jak se náš řetězec může dostat ze stavu 3 do stavu 5. Každá tato možnost je vlastně posloupnost výsledků čtyř partií. Pro každou z těchto posloupností vypočteme pravděpodobnost, že průběh hry bude přesně takový (půjde o součin čtyř pravděpodobností pro čtyři partie). Pravděpodobnosti jednotlivých posloupností pak sečteme.

$$
\begin{array}{llll}
1 & \text{PVVV} & 5/12 \cdot 5/12 \cdot 5/12 \cdot 5/12 & = 0.0301408 \\
1 & \text{VPVV} & 5/12 \cdot 5/12 \cdot 5/12 \cdot 5/12 & = 0.0301408 \\
1 & \text{VVPV} & 5/12 \cdot 5/12 \cdot 5/12 \cdot 5/12 & = 0.0301408 \\
1 & \text{NNVV} & 2/12 \cdot 2/12 \cdot 5/12 \cdot 5/12 & = 0.0048225 \\
1 & \text{NVNV} & 2/12 \cdot 5/12 \cdot 2/12 \cdot 5/12 & = 0.0048225 \\
1 & \text{NVVN} & 2/12 \cdot 5/12 \cdot 5/12 \cdot 2/12 & = 0.0048225 \\
1 & \text{VNNV} & 5/12 \cdot 2/12 \cdot 2/12 \cdot 5/12 & = 0.0048225 \\
1 & \text{VNVN} & 5/12 \cdot 2/12 \cdot 5/12 \cdot 2/12 & = 0.0048225 \\
1 & \text{VVNN} & 5/12 \cdot 5/12 \cdot 2/12 \cdot 2/12 & = 0.0048225 \\
\hline
 & & & = 0.1193576
\end{array}
$$

Tato „metoda hrubé síly“ sice zaručeně vede ke správnému výsledku, ale je zřejmé, že je velice pracná, zejména pro větší hodnoty t je dokonce mnohem pracnější než výpočet mocniny matice P^t. Další nevýhodou této metody jsou problémy se zaokrouhlováním, neboť výsledek získáme jako součet velkého počtu velmi malých čísel.

Pro ruční výpočet je **v našem jednoduchém příkladě** vhodnější počítat pravděpodobnostní vektory postupně, tedy vlastně stejně jako při násobení přechodovou maticí, ale namísto nepohodlného násobení maticí budeme postupovat podle schématu

$$
\begin{array}{ccccc}
p_{i-1}^{(t)} & & p_i^{(t)} & & p_{i+1}^{(t)} \\
& \searrow & \downarrow & \swarrow & \\
& & p_i^{(t+1)} & &
\end{array}
$$

přičemž samozřejmě nesmíme počítat neexistující přechody ze stavů 1 a 6.

Podle stejného schématu lze snadno spočítat o počet možností, jen přitom používáme prostě sečítáme příslušné hodnoty z předchozího řádku. Poněvadž výpočet počtu možností je jednodušší, předvedeme jej jako první:

t	1	2	3	4	5	6
0	0	0	1	0	0	0
1	0	1	1	1	0	0
2	1	2	3	2	1	0
3	3	5	7	6	3	1
4	8	12	18	16	9	4

Všimněte si, že z poslední řádky stačilo spočítat jen jednu hodnotu, která nás zajímá (tj. 9 možností) a podobně jsme mohli vynechat i několik dalších hodnot.

Nyní již předvedeme výpočet pravděpodobnosti $p_5^{(4)}$:

t	1	2	3	4	5	6
0			1			
1		5/12	2/12	5/12		
2			54/144	20/144	25/144	
3				435/1728	150/1728	
4					2475/20736	

což dává výsledek shodný s předchozím.

11.3 Klasifikace stavů a typy řetězců

Většinu těchto pojmů definujeme pomocí vlastností přechodového grafu, neboť tyto vlastnosti lze v grafu poměrně snadno ověřit.

11.3.1 Stochasticky uzavřená množina stavů je taková množina, ze které nevychází žádná hrana ven. Jinak řečeno, pravděpodobnost, že markovský řetězec opustí stochasticky uzavřenou množinu, je nulová.

11.3.2 Ergodická množina stavů je stochasticky uzavřená množina, která neobsahuje menší stochasticky uzavřenou množinu.

Množina stavů je ergodická právě tehdy, když jí odpovídající podgraf přechodového grafu je silně souvislou komponentu, ze které nevychází ven žádná hrana.

Ergodický stav je stav, který je prvkem nějaké ergodické množiny. Každý markovský řetězec má alespoň jednu ergodickou množinu.

11.3.3 Transientní stav je stav, který není ergodický.

Transientní množina je množina všech transientních stavů. Transientní množina se tedy skládá z nula až několika silně souvislých komponent přechodového grafu.

11.3.4 Absorbující stav je stav, který sám tvoří jednoprvkovou ergodickou množinu.

Stav je absorbující právě tehdy, když v přechodovém grafu z tohoto stavu vychází jediná hrana a to smyčka (která má tudíž pravděpodobnost 1).

11.3.5 Absorbující řetězec je takový markovský řetězec, jehož všechny ergodické stavy jsou absorbující.

11.3.6 Ergodický markovský řetězec je takový, jehož přechodový graf je silně souvislý.

11.3.7 Stacionární markovský řetězec je takový, jehož přechodový graf je silně souvislý a největší společný dělitel délek všech cyklů je roven jedné (tj. délky všech cyklů jsou nesoudělné). Dodejme, že délku cyklu zde chápeme jako počet jeho hran.

Vysvětlení názvu stacionární je podáno v 11.6.

11.3.8 Periodický markovský řetězec je takový, jehož přechodový graf je silně souvislý a největší společný dělitel délek všech cyklů je větší než jedna (tj. délky všech cyklů jsou soudělné). Dodejme, že délku cyklu zde chápeme jako počet jeho hran.

Vysvětlení názvu periodický je podáno v 11.7.

11.4 Analýza obecného markovského řetězce

Prvým krokem v analýze obecného markovského řetězce je vždy nalezení silně souvislých komponent a jejich rozdělení na transientní a ergodické.

11.4.1 Analýza transientního chování. Má-li řetězec neprázdnou transientní část, pak základní otázky týkající se této části jsou:

- Je-li dán výchozí transientní stav a cílová ergodická množina, určit pravděpodobnost, že řetězec dosáhne této ergodické množiny
- Je-li dán výchozí transientní stav, určit střední počet kroků, než bude dosaženo některého ergodického stavu.
- Je-li dán výchozí transientní stav i a další transientní stav j, určit střední počet průchodů stavem j, než bude dosaženo některého ergodického stavu.

Pro účely řešení těchto otázek lze markovský řetězec zjednodušit tím, že každou ergodickou množinu stavů nahradíme jedním absorbujícím stavem. Pro řešení otázek 2 a 3 dokonce můžeme všechny ergodické množiny nahradit jedním společným absorbujícím stavem.

Řešením těchto otázek se budeme zabývat v 11.5.

Je-li transientní část řetězce prázdná, jsou všechny stavy řetězce ergodické a analýzu transientního chování přeskakujeme.

11.4.2 Analýza ergodického chování. Co se s řetězcem děje po dosažení ergodického stavu, lze zjišťovat pro každou ergodickou množinu zvlášť, neboť řetězec nemůže ergodickou množinu opustit.

Ukazuje se, podrobněji viz 11.6 a 11.7, že *dlouhodobé* chování ergodického řetězce je do značné míry nezávislé na výchozím stavu. Proto v analýze ergodického chování uvažujeme každou ergodickou množinu zvlášť jako samostatný ergodický markovský řetězec.

V ergodických řetězcích jsou základní otázky tyto:

- Pro každý stav i určit pravděpodobnost w_i, že v náhodně zvoleném okamžiku najdeme řetězec ve stavu i.
- Pro každý stav i určit střední počet kroků, po kterém se řetězec vrátí do stavu i.

Řešením těchto otázek se budeme zabývat v 11.6 a 11.7.

11.5 Absorbující řetězce

Uvedeme dva způsoby řešení otázek uvedených v 11.4.1. Jeden bude založen na úpravách grafu, druhý bude založen na maticích. Nejprve však důležité tvrzení.

11.5.1 Věta. Pro každý markovský řetězec a pro každý výchozí transientní stav i platí, že posloupnost pravděpodobností, že se řetězec v čase t nachází v transientním stavu, konverguje k nule pro $t \to \infty$.

Důkaz je třeba doplnit.

11.5.2 Kanonický tvar přechodové matice absorbujícího řetězce. Stavy absorbujícího řetězce lze přečíslovat tak, že nejnižšími čísly jsou očíslovány absorbující stavy a za nimi následují stavy transientní. Po takovém přečíslování má přechodová matice tvar

$$P = \left(\begin{array}{c|c} E & 0 \\ \hline R & Q \end{array}\right)$$

11.5.3 Věta. Mocniny matice Q konvergují k nulové matici, tj. $\lim_{t\to\infty} Q^t = 0$.

Existuje inverzní matice $(E - Q)^{-1}$

Součet maticové řady $0 + E + Q + Q^2 + Q^3 + \ldots$ existuje a je roven $(E - Q)^{-1}$.

11.5.4 Fundamentální matice absorbujícího řetězce je matice $H = (E - Q)^{-1}$.

11.5.5 Věta. Prvky fundamentální matice H mají tento význam: $h_{i,j}$ je střední počet, kolikrát řetězec projde stavem j, když začal ve stavu i.

Důsledek: Jestliže absorbující řetězec začal ve stavu i, pak střední počet časových kroků, než řetězec dosáhne absorbujícího stavu, je roven součtu hodnot v i-tém řádku fundamentální matice H.

11.5.6 Pravděpodobnost absorbce v daném stavu Je dán absorbující řetězec, výchozí stav i a absorbující stav j. Označme $b_{i,j}$ pravděpodobnost, že řetězec dosáhne stavu j, když (s pravděpodobností 1) začal ve stavu i.

Pravděpodobnosti $b_{i,j}$ lze uspořádat do matice B. Platí

$$B = R + QB\,.$$

Odtud plyne

$$(E - Q)B = R \tag{11.1}$$

a tedy

$$B = (E - Q)^{-1}R\,. \tag{11.2}$$

11.5.7 Výpočty pomocí elementárních úprav grafu. Pro řetězce s nevelkým počtem stavů a pro ruční výpočet je často vhodnější metoda založená na elementárních úpravách grafu:

Výpočet pravděpodobností absorbce pro výchozí stav i je snadný: Graf postupně redukujeme eliminováním smyček a vrcholů, až nakonec zbude pouze výchozí vrchol i a všechny absorbující vrcholy. Elementární úpravy zachovávají pravděpodobnost dosažení stavu, proto z výsledného grafu lze okamžitě zjistit hledané pravděpodobnosti absorbce v jednotlivých absorbujících stavech.

Výpočet středního počtu průchodů stavem j, když řetězec začal ve stavu i, lze také provést elementárními úpravami grafu. Nejprve všechny absorbující stavy nahradíme jediným absorbujícím stavem a, protože v této úloze se nezajímáme, kde k absorbci dojde, ale kdy k ní dojde. Dále pak graf zredukujeme elementárními úpravami tak, aby zbyly pouze tři vrcholy i, j, a. V tomto redukovaném grafu označme p pravděpodobnost smyčky ve vrcholu j a dále označme q pravděpodobnost přechodu ze stavu i do stavu j. Je dobré si uvědomit, že vzhledem k původnímu absorbujícímu řetězci je q pravděpodobnost, že ze stavu i bude dosaženo stavu j (tj. že nedojde k absorbci dříve než stavu j dosáhneme) a pravděpodobnost p je pravděpodobnost, že řetězec po průchodu stavem j do tohoto stavu ještě někdy vrátí (dříve než dojde k absorbci). Máme-li takto určeny pravděpodobnosti p a q, lze střední počet průchodů stavem j určit podle vzorce

$$h_{i,j} = \frac{q}{1-p}$$

11.6 Stacionární řetězce

Stacionární markovské řetězce by bylo možno pokládat za speciální případ řetězců periodických, kde perioda je rovna jedné. Jde však o případ z praktického hlediska velmi významný, proto mu věnujeme samostatnou sekci.

Lze dokázat, že ve stacionárním řetězci pro každý stav j existuje limita

$$\lim_{t\to\infty} p_{i,j}^{(t)} = w_j \tag{11.3}$$

a tato limita nezávisí na výchozím stavu i.

Dále, ve stacionárním řetězci mocniny přechodové matice P^t pro $t \to \infty$ konvergují a to k matici W, jejíž všechny řádky jsou rovny vektoru stacionárních pravděpodobností $w = (w_1, w_2, \ldots, w_n)$.

Z toho plyne, že matice W musí splňovat rovnici

$$w \cdot P = w\,. \tag{11.4}$$

11.6.1 Výpočet stacionárních pravděpodobností. Základem pro výpočet je soustava rovnic 11.4. Tuto soustavu lze získat také tak, že pro jednotlivé stavy uvažujeme všechny možné stavy předchozí: Pro každý stav j tak dostáváme rovnici

$$w_j = w_1 p_{1,j} + w_2 p_{2,j} + \ldots + w_n p_{n,j}$$

Soustava rovnic 11.4 je ovšem homogenní (po převedení všech proměnných na levou stranu je pravá strana nulová) a proto nemá jednoznačné řešení: libovolný násobek nějakého řešení je opět řešením této soustavy. Matice této soustavy $(P - E)$ je singulární, některý její řádek je lineární kombinací ostatních. Proto ze soustavy 11.4 můžeme jednu rovnici vynechat (lze ji odvodit z ostatních), a dále k soustavě přidáváme rovnici

$$\sum_{i=1}^{n} w_i = 1 \,,$$

Řešením takto vzniklé soustavy lineárních rovnic získáme vektor stacionárních pravděpodobností w.

11.6.2 Praktické využití stacionárních pravděpodobností. Při praktickém modelování je často každý stav i v modelovaném ději spojen s náklady c_i. Pro některé stavy to samozřejmě mohou být náklady nulové. Případné zisky můžeme pokládat za záporné náklady.

Ze znalosti stacionárních pravděpodobností w_i pak lze pro pro ustálené chování stacionárního markovského řetězce spočítat střední náklady na jednotku času (přesněji, na jeden časový krok) podle vzorce

$$C = \sum_{i=1}^{n} w_i c_i \,.$$

11.7 Periodické řetězce

Označme d největší společný dělitel délek všech cyklů. Toto číslo budeme nazývat periodou řetězce.

11.7.1 Periodické třídy. Množinu stavů periodického řetězce lze rozdělit do d disjunktních podmnožin $C_0, C_1, C_2, \ldots, C_{d-1}$ takových, že v přechodovém grafu vycházejí všechny hrany z množiny C_0 vedou pouze do množiny C_1, všechny hrany z množiny C_1 vedou pouze do množiny C_2, atd., až nakonec všechny hrany z množiny C_{d-1} vedou pouze do množiny C_0.

Připomeňme, že řetězec je periodický a tedy ergodický. Z libovolného stavu i je proto dosažitelný každý stav j, a to včetně stavu i. Existuje tedy cyklus, procházející stavem i. Z existence d periodických tříd však plyne, že délka každého cyklu je je násobkem periody d. Nemůže tedy např. být menší než d.

11.7.2 Periodické chování. Název periodického řetězce je odvozen z faktu, že byl-li řetězec v čase 0 ve třídě C_0, pak třída, ve které se bude nacházet v obecném čase t závisí zcela jednoznačně na zbytku při dělení času t periodou d.

Je-li např. $d = 2$, pak máme dvě periodické třídy. V jedné z nich se řetězec může nacházet pouze v lichých okamžicích, ve druhé pouze v sudých okamžicích.

Podobně je-li např. $d = 3$, máme tři periodické třídy. V jedné z nich se řetězec může nacházet pouze v okamžicích t dělitelných třemi, ve druhé v okamžicích, které při dělení třemi dávají zbytek 1 a konečně ve třetí třídě se může nacházet pouze v okamžicích, které při dělení třemi dávají zbytek 2.

11.7.3 Pravděpodobnosti výskytu stavů. V periodickém řetězci neexistují limity (11.3), a tedy nemá smysl mluvit o stacionárních pravděpodobnostech. Důvod neexistence limity je prostý: v posloupnosti jsou nenulové hodnoty pouze na každém d-tém místě a mezi nimi jsou nuly. Taková posloupnost nemůže konvergovat k ničemu nenulovému.

Pro praktické účely ovšem nepotřebujeme přesně tuto limitu. Potřebujeme vědět, jaké procento času stráví řetězec v jednotlivých stavech. Nebo jinak, jaká je pravděpodobnost w_i, že v náhodném (a dostatečně vzdáleném) časovém okamžiku t najdeme řetězec ve stavu i.

Předpokládejme, že budeme náhodně volit okamžik t tak, aby všechny možné zbytky při dělení času t periodou d měly stejnou pravděpodobnost $1/d$. Pak pravděpodobnosti w_i, že v náhodném okamžiku t najdeme periodický řetězec ve stavu i, můžeme počítat naprosto stejným postupem jako v 11.6.1.

Kapitola 12

Teorie front

Teorie front se zabývá situacemi, kdy požadavky na nějakou službu nejsou v souladu s možností tuto službu poskytnout. Subjekty, které službu požadují, musí na službu čekat. Alternativní názvy teorie front jsou **teorie hromadné obsluhy** nebo **teorie čekacích jevů**.

12.1 Základní pojmy teorie front

12.1.1 Příklad: holičství Do holičství přicházejí zákazníci. Zákazníky obsluhuje jeden holič. Přijde-li během obsluhy další zákazník, musí počkat. Nabízí se otázka, zda by se vyplatilo, aby zákazníky obsluhovalo více holičů nebo aby holič investoval do lepšího vybavení a zlepšil tak svůj výkon.

12.1.2 Zákazník reprezentuje požadavek na provedení služby. Slovo zákazník chápejte jako termín. Zákazníkem může být třeba letadlo čekající na volnou přistávací dráhu, telefonní hovor čekající na spojení, úloha v počítači čekající na volný procesor, porouchaný stroj čekající na opravu, datový paket čekající na odeslání v počítačové síti apod.

12.1.3 Fronta je v teorii front chápána jako spojení dvou věcí: množiny zákazníků, kteří čekají na (tutéž) obsluhu, a tzv. *režimu fronty*, který určuje, který zákazník bude obsluhován jako další.

Fronta může být hmatatelná a dobře viditelná, např. fronta u pokladny v supermarketu, může však být i fyzicky rozptýlená. Např. na úřadě, kde každý klient při příchodu dostane papírek s číslem a úředníci si zákazníky volají ke svým přepážkám podle čísel pomocí světelné tabule. Podobně to je třeba s čekateli na přidělení obecního bytu.

Ve složitějších systémech s několikastupňovou obsluhou může být front několik. V jedné frontě jsou vždy ti zákazníci, kteří čekají na tutéž obsluhu.

12.1.4 Režim fronty je pravidlo, které určuje, který zákazník bude obsluhován jako další. Základní režimy jsou tyto:

FIFO (z anglického „first in first out“) — vybírá se zákazník, který ve frontě čeká nejdéle. Je to klasická spravedlivá fronta bez předbíhání.

LIFO (z anglického „last in first out“) vybírá se zákazník, který ve frontě čeká nejkratší dobu. Příklad: zásoba materiálu, který čeká na zpracování a je ve skladu ukládán na sebe.

Prioritní režim — vybírá se zákazník, který má ze všech zákazníků ve frontě nejvyšší prioritu. Co je tou prioritou a jak jsou priority uspořádány, to je předmětem modelu, tj. je třeba to specifikovat.

Náhodný režim fronty — zákazník se vybírá náhodně, přičemž je třeba specifikovat, s jakou pravděpodobností bude vybrán ten který zákazník. Z hlediska modelu je nejjednodušší, když všichni zákazníci ve frontě mají stejnou pravděpodobnost, ale v některých situacích to může být přílišné zjednodušení. Příklad: káď s kapry při předvánočním prodeji.

12.1.5 Kanál obsluhy je to, co poskytuje službu. Kanál obsluhy je zpravidla schopen obsluhovat v každém okamžiku jen jednoho zákazníka, ale v systému může být několik kanálů. Kanál obsluhy je charakterizován dobou obsluhy, která je buď náhodná s nějakým rozložením, nebo deterministická.

Je-li kanálů obsluhy více, mohou fungovat paralelně a vybírat zákazníky z téže fronty. Příkladem je úřad, kde se zákazníkům přidělují při příchodu čísla.

Ve složitějších systémech může zákazník procházet postupně několika kanály obsluhy, v takovém případě bývá i více front.

12.1.6 Uzavřené systémy hromadné obsluhy jsou takové, kde žádný zákazník zvnějšku do systému nepřichází, žádný zákazník systém neopouští, všichni zákazníci jsou součástí systému. Počet zákazníků v systému je tedy konstantní.

12.1.7 Příklad. V dílně je 10 stejných a dost poruchových strojů. Porouchaný stroj není možné provozovat, takový stroj čeká na opravu. Množina čekajících strojů tvoří frontu. Porouchané stroje opravuje jeden opravář (kanál obsluhy). Je zřejmé, že počet poruch za jednotku času závisí na tom, kolik strojů již je poroucháno. V extrémním případě, když jsou všechny stroje porouchány nebo zrovna opravovány, k žádné další poruše nemůže dojít, dokud není nějaký stroj opraven a uveden do provozu.

12.1.8 Otevřené systémy hromadné obsluhy jsou takové, kde zákazníci po dokončení obsluhy odcházejí ze systému pryč a naopak z vnějšku systému přicházejí zákazníci noví. Počet zákazníků v systému tedy obvykle není konstantní, naopak, často bývá shora neomezený.

Pro zkoumání otevřených systémů hromadné obsluhy má zásadní význam tzv. **vstupní proces**.

12.1.9 Vstupní proces je proces přicházení zákazníků do systému hromadné obsluhy z vnějšku. Může být deterministický nebo náhodný.

Nejjednodušším příkladem deterministického vstupního procesu jsou zcela pravidelné příchody, třeba každých 7 vteřin jeden zákazník, ale deterministický vstupní proces může být i složitější, například vždy po 10 minutách přijdou tři zákazníci půl minuty po sobě.

Náhodné vstupní procesy se vyskytují v nepřeberném množství druhů. Některé z nich mohou být založeny na deterministickém procesu, který byl ovlivněn nějakým náhodným vlivem. Příkladem mohou být příhochody každých 7 vteřin jeden zákazník, který ovšem může být až 3 vteřiny náhodně opožděn. Nebo každých 10 minut autobus přiveze náhodný počet zákazníků. V obou těchto případech by pro účely modelu bylo nutno specifikovat rozložení příslušných náhodných veličin (velikost zpoždění popř. počet zákazníků v autobuse).

Nejčastějším typem náhodného vstupního procesu jsou tzv. čítací procesy, kde zákazníci přicházejí po jednom. V praxi je velmi častý tzv. Poissonův vsupní proces popsaný v 10.2.

Vstupní proces obvykle bývá nezávislý na počtu zákazníků v systému. Tento předpoklad znamená, že počet zákazníků v systému je shora neomezený a že vně systému vždy je potenciálně nekonečná „zásoba zákazníků“, kteří by do systému ještě mohli vstoupit.

To je samozřejmě zjednodušující předpoklad. V praxi je počet zákazníků vždy nějak shora omezen. Důležité je, jaký vliv má toto omezení na vstupní proces. Je-li tento vliv malý, lze jeho zanedbáním model vstupního procesu výrazně zjednodušit.

12.1.10 Intenzita vstupního procesu (též zjednodušeně *intenzita vstupu*) je průměrný počet zákazníků, kteří do systému vstoupí za jednotku času. Intenzitu vstupu dále značíme λ.

Obecně se intenzita vstupu může měnit v čase (např. sezonní vlivy nebo vliv denní doby), ale v teoretických modelech často předpokládáme, že vstupní proces je homogenní a tedy intenzita vstupu je konstantní.

12.1.11 Intenzita obsluhy se vztahuje vždy na jeden kanál obsluhy a je to průměrný počet zákazníků, které by kanál obsluhy obsloužil za jednotku času, pokud by tito zákazníci byli připraveni ve frontě. Intenzitu obsluhy dále značíme μ.

Pozor, nezaměňujte intenzitu obsluhy s počtem zákazníků, které systém obsluhy skutečně obslouží. Systém nemůže obsloužit zákazníky, kteří do systému nepřišli, proto kanály obsluhy mívají prostoje, kdy čekají na příchod zákazníka.

12.1.12 Cvičení. Promyslete způsob jak prakticky (pozorováním skutečného systému hromadné obsluhy) zjišťovat intenzitu obsluhy.

12.2 Typy systémů hromadné obsluhy

12.2.1 Jednoduché systémy hromadné obsluhy. Dále se budeme zabývat pouze jednoduchými otevřenými systémy hromadné obsluhy, které mají jen jednu frontu a jeden nebo více stejně výkonných kanálů obsluhy, které navíc splňují tyto předpoklady:

- Zákazník, který vstoupil do systému, musí projít obsluhou, tj. nemůže odejít neobsloužen.
- Započatou obsluhu nelze přerušit.
- Není přípustné, aby byl volný kanál obsluhy a přitom nějaký zákazník čekal ve frontě.

12.2.2 Kendallova klasifikace. Výše zmíněné jednoduché systémy lze klasifikovat podle tzv. Kendallovy klasifikace. Typ systému se zapisuje ve formě výrazu $X/Y/S$, kde X označuje typ vstupního procesu, Y určuje rozložení doby obsluhy a S je počet kanálů obsluhy.

Typ	Vstupní proces	rozložení doby obsluhy
M	Poissonův proces	exponenciální rozložení
E_k	Erlangův s parametrem k	Erlangovo s parametrem k
K_n	χ^2 s n stupni volnosti	χ^2 s n stupni volnosti
D	deterministický	konstantní
G	obecný	obecné

Například systém $M/M/1$ je jednokanálový systém, jehož vstupní proces je Poissonův a doba obsluhy je náhodná s exponenciálním rozložením. Systém $M/D/1$ se liší tím, že doba obsluhy všech zákazníků je konstantní.

12.2.3 Určující parametry jednoduchých otevřených systémů jsou

λ intenzita příchodů (viz 12.1.10)
μ intenzita obsluhy (viz 12.1.11)
S počet kanálů obsluhy

Tyto parametry obvykle o systému známe nebo je můžeme snadno změřit, můžeme je přímo ovlivnit a obvykle jsou předmětem optimalizace.

12.2.4 Provozní charakteristiky. Další hodnoty charakterizují typickou činnost systému. Většinu z nich nemůžeme ovlivnit přímo, ale jen skrze výše uvedené určující parametry. U jednodušších modelů pro tyto hodnoty existují vzorce, ve složitějších případech se tyto hodnoty zjišťují pomocí simulace metodou Monte Carlo, tedy náhodnými pokusy a jejich statstickým vyhodnocením.

Důležitými provozními charakteristikami jsou zejména

$\overline{n_s}$	průměrný počet zákazníků v systmu
$\overline{n_f}$	průměrný počet zákazníků ve frontě
$\overline{n_o}$	průměrný počet zákazníků v obsluze
$\overline{t_s}$	průměrný čas strávený zákazníkem v systmu
$\overline{t_f}$	průměrný čas strávený zákazníkem ve frontě
$\overline{t_o}$	průměrný čas strávený zákazníkem v obsluze
p_0	pravděpodobnost, že systém je prázdný

Poznamenejme, že průměrné počty (v systému, frontě a obsluze) se počítají jako vážený průměr, kde v roli váhy vystupuje doba, po kterou byl ten počet platný.

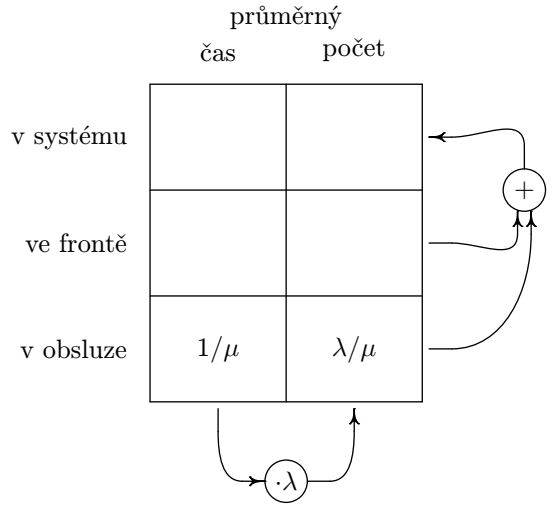
Tedy pokud například během 15-minutového intervalu byl systém (celkem) 2 minuty prázdný, po dobu (celkem) 5 minut byl v systému jeden zákazník a po dobu (celkem) 8 minut byli v systému 2 zákazníci, pak průměrný počet zákazníků byl $(2\cdot 0+5\cdot 1+8\cdot 2)/(2+5+8) = 21/15 = 1.4$. Přitom je lhostejné, zda přesně jeden zákazník byl v systému v jednom nepřerušeném pětiminutovém intervalu, nebo zda to bylo v několika dílčích intervalech o celkové délce 5 minut.

Průměrnou dobu obsluhy $\overline{t_o}$ lze triviálně odvodit z intenzity obsluhy: $\overline{t_o} = 1/\mu$.

Dále platí, že průměrný počet (v systému, ve frontě nebo v obsluze) je λ-krát odpovídající průměrný čas (v systému, ve frontě nebo v obsluze).

Jestliže každý zákazník strávil průměrně v systému čas $\overline{t_s}$ a takových zákazníků prošlo systémem v průměru λ za časovou jednotku, pak průměrný počet zákazníků v systému je $\overline{n_s} = \lambda\overline{t_s}$. Podobně pro počty a časy ve frontě a v obsluze.

Schematicky jsou tyto vztahy znázorněny na obrázku 12.1. Praktický dopad těcho úvah je ten, že stačí získat vzorec pro jedno (kterékoli) z prázdných polí a ostatní lze triviálně dopočítat.



Obrázek 12.1: Vztahy mezi průměrnými parametry systémů hromadné obsluhy.

12.2.5 Systém $M/M/1$. Vstupní proces je Poissonovský s intenzitou λ, doba obsluhy je náhodná s exponenciálním rozložením s parametrem μ, kanál obsluhy je jeden.

	Průměrný	
	čas	počet
v systému	$\frac{1}{\mu-\lambda}$	$\frac{\lambda}{\mu-\lambda}$
ve frontě	$\frac{\lambda}{\mu(\mu-\lambda)}$	$\frac{\lambda^2}{\mu(\mu-\lambda)}$
v obsluze	$\frac{1}{\mu}$	$\frac{\lambda}{\mu}$

Někdy se tyto vzorce zapisují alternativně pomocí veličiny $\eta = \lambda/\mu$. Pak vychází $n_s = \eta/(1-\eta)$ a $n_f = \eta^2/(1-\eta)$. Pro vyjádření průměrných časů t_s, t_f a t_o ovšem tak jako tak potřebujeme veličinu λ nebo μ.

Dále platí $p_0 = 1 - \lambda/\mu = 1 - \eta$.

12.2.6 Systém $M/M/S$. Vstupní proces je Poissonovský s intenzitou λ. Systém má S stejných kanálů obsluhy, každý z nich má intenzitu obsluhy μ. Doba obsluhy v každém kanále je náhodná se stejným exponenciálním rozložením s parametrem μ,

Označme $\eta = \lambda/\mu$. Platí

$$p_0 = \frac{1}{\frac{\eta^S}{S!(1-\eta/S)} + \sum_{k=0}^{S-1} \frac{\eta^k}{k!}}$$

$$\overline{n_f} = \frac{\eta^{S+1} \cdot p_0}{S \cdot S!(1-\eta/S)^2}$$

Ostatní hodnoty lze dopočítat podle obrázku 12.1.

Předpoklad, že všechny kanály mají stejnou intenzitu obsluhy, je z praktického hlediska nerealistický, ale pro výše uvedené vzorce je podstatný. Pokud by kanály nebyly stejné, tj. pokud by měly různé intenzity obsluhy, závisel by výsledek na způsobu přidělování práce jednotlivým kanálům. Tedy např. zda v situaci, kdy je více volných kanálů, přidělíme práci kanálu nejrychlejšímu, nejpomalejšímu, náhodně zvolenému, tomu, který dosud obsloužil nejméně zákazníků a nebo nějak jinak, fantazii se meze nekladou. Různé intenzity obsluhy v jednotlivých kanálech a způsob přidělování práce by bylo nutno ve vzorcích zohlednit a vzorce by tak byly podstatně složitější. Tyto případy je již lépe řešit simulací.

12.2.7 Cvičení. Porovnejte dvoukanálový systém $M/M/2$ s intenzitou obsluhy μ v každém ze dvou kanálů a jednokanálový systém $M/M/1$, jehož kanál obsluhy má dvojnásobnou intenzitu obsluhy 2μ.

Oba systémy jsou dlouhodobě schopny obsloužit stejný počet zákazníků. Přesto je ve fungování obou systémů podstatný rozdíl. Zjistěte, v čem rozdíl spočívá a vysvětlete jeho příčinu. Kterému z těchto systémů byste jako zákazníci dali přednost? Kde je kratší doba čekání ve frontě a kde je kratší celková doba pobytu v systému?

Čemu jako zákazníci dáme přednost, záleží na osobních preferencích. Chceme-li si obsluhu „užít“, asi dáme přednost pomalejší obsluze a kratšímu čekání. Chceme-li naopak v systému strávit co nejméně času, dáme přednost rychlejší obsluze a to i za cenu delšího čekání ve frontě.

12.2.8 Systém $M/D/1$. Vstupní proces je Poissonovský s intenzitou λ. Doba obsluhy je konstantní a rovná $1/\mu$, kde μ je intenzita obsluhy. Pak průměrná doba pobytu zákazníka v systému je

$$\overline{t_s} = \frac{2\mu - \lambda}{2\mu(\mu - \lambda)}$$

což je méně než u systému $M/M/1$. Průměrná doba čekání ve frontě vychází dokonce přesně poloviční ve srovnání se systémem $M/M/1$, totiž $\overline{t_f} = \lambda/2\mu(\mu - \lambda)$.

12.2.9 Systém $M/G/1$. Vstupní proces je Poissonovský s intenzitou λ. Doba obsluhy má obecné (blíže neurčené) rozložení. Za předpokladu, že doba obsluhy má konečný rozptyl D^2 a střední hodnota doby obsluhy je m (tedy intenzita obsluhy je $\mu = 1/m$), je průměrný počet zákazníků v systému dán tzv. Polaczekovou-Chinčinovou formulí

$$n_s = \eta + \frac{\lambda^2 D^2 + \eta^2}{2(1 - \eta)} \ ,$$

kde $\eta = \lambda/\mu$, tedy $\eta = \lambda m$, kde m je střední doba obsluhy.

12.2.10 Poučný příklad. Máme velkou dílnu s nepřetržitým provozem a s mnoha poruchovými stroji. Výskyty poruch tvoří Poissonovský proces s intenzitou 10 poruch za hodinu (samozřejmě za předpokladu, že počet porouchaných strojů je zanedbatelný vzhledem k „velkému“ počtu všech strojů). Stroje opravuje jeden opravář, který má stálou pohotovost, doba opravy má exponenciální rozložení, intenzita obsluhy je 15 oprav za hodinu. Ztráty z prostoje porouchaného stroje jsou 1000 Kč/hod., mzda opraváře včetně režie je 100 Kč/hod.

Průměrné náklady v Kč za hodinu jsou $100 + 1000 n_s = 2100$ Kč/hod.

Představme si, že se o práci opraváře uchází člověk, který dosahuje vyšší intenzity obsluhy 16 oprav za hodinu, ale má mnohem vyšší mzdové požadavky, jeho mzda by byla 200 Kč/hod. Vyplatí se dosavadního opraváře nahradit novým uchazečem?

Průměrné náklady v Kč za hodinu pro nového uchazeče vycházejí $200{+}1000 n_s = 1866.66$ Kč/hod., tedy nižší, než u původního opraváře.

Paradoxní (a poučné) je, že nový opravář se vyplatí navzdory tomu, že dostává vyšší mzdu a přitom větší procento pracovní doby „nic nedělá“, jen čeká, až se nějaký stroj porouchá. Podstatné ovšem je, že při tom „nic nedělání“ je neustále připraven okamžitě zahájit obsluhu (zde opravu porouchaného stroje).

12.2.11 Poučení z teorie front. Z předchozího příkladu si lze vzít obecně platné poučení: Jsou-li příchody zákazníků náhodné, je určitá rezerva ve výkonu kanálu obsluhy ekonomicky zdůvodnitelná. Snaha tuto rezervu eliminovat snížením intenzity obsluhy μ na hodnotu blížící se intenzitě vstupu λ má za následek velkou průměrnou délku fronty a velké ztráty z toho vyplývající.

Systém, jehož fungování ovlivňuje náhoda,
by se neměl provozovat bez rezervy ve výkonu.

Kapitola 13

Simulace

Simulační model je takový model, v němž je čas modelované soustavy modelován časem modelu. Dějům, které se odehrávají v modelované soustavě pak odpovídají děje v modelu. Simulovat lze procesy deterministické i náhodné. Zde, v tomto skriptu, se budeme zabývat výhradně simulací na číslicových počítačích. Pouze připomeňme, že se používají i simulační modely fyzikální, např. mechanické nebo elektrické.

Pokud modelovaný proces je náhodný, pak by i jeho simulační model měl vykazovat náhodné chování. Pokud s takovým simulačním modelem provádíme náhodné simulační experimenty, které statisticky vyhodnocujeme, mluvíme o simulaci metodou Monte Carlo.

Poznámka: Převážná část této kapitoly byla původně napsána pro jiné skriptum a pro studenty, u nichž se předpokládalo, že umí programovat. Proto jsou zde uváděny programy a jejich části, proto řada cvičení začíná slovy "napište program" nebo "upravte program".

Rozhodl jsem se nezatajovat, že počítačové simulace se programují a programy jsem v textu ponechal. Kdo jim porozumí, nechť z toho má užitek. I ten kdo programovat neumí, se z nich může trochu poučit.

Pokud se ve cvičeních požaduje vytvoření nebo úprava programu, chápejte to jako výzvu, abyste promysleli postup výpočtu a napsali instrukce pro někoho, kdo by ten výpočet měl dělat za vás. Instrukce by měly být tak jasné a jednoznačné, aby se ten, kdo by se jimi měl řídit, nemohl vymlouvat, že něco pochopil jinak.

13.1 Základní triky

13.1.1 Jak simulovat pomocí počítače. V podstatě je třeba pro časové okamžiky, které nás zajímají, vypočítat hodnoty stavových veličin. Pro simulaci je charakteristické, že tento výpočet probíhá "ve směru času", tj. výpočet budoucích stavů se počítá na základě stavů minulých.

Jednoduché simulační modely lze často vytvořit ad hoc bez složitých úvah. Komplikovanější modely vyžadují systematický přístup, který popisujeme v sekcích 13.2 a 13.3.

13.1.2 Statistické vyhodnocení simulace. Jsou v podstatě dvě možnosti. Jedna možnost je během simulace pouze zaznamenávat vše, co se v simulačním modelu stalo a teprve po skončení simulace pak tento záznam z různých hledisek analyzovat a statisticky vyhodnocovat. Nevýhodou tohoto postupu je velký objem zaznamenaných dat, výhodou je, že chceme-li dodatečně další satistické údaje lze je získat bez nutnosti opakovat samotnou simulaci.

Druhá možnost je během simulace nezaznamenávat jednotlivé události, ale pouze sbírat statistická data nutná pro odpověď na konkrétní předem známou otázku.

Pokud nás např. zajímá jen průměrná doba čekání zákazníka ve frontě, stačí, když během simulace budeme sečítat doby čekání a zjišťovat počet zákazníků, tedy během simulace udržovat

celkem dvě hodnoty. Pro výpočet průměru to stačí. Pokud však budeme později chtít nejen střední hodnotu, ale i rozptyl, medián nebo jinou statistiku, bude nutno simulační program upravit, aby zaznamenával a vyhodnocoval další i hodnoty a simulaci s rozšířeným programem zopakovat.

13.1.3 Počáteční a ustálené fungování. Je-li fungování simulovaného systému závislé na výchozích podmínkách, je třeba simulační experiment mnohokrát opakovat s týmiž výchozími podmínkami.

Příkad: Ve frontě je 100 zákazníků. Jaká je očekávaná doba, než se fronta vyprázdní? Výsledkem každého jednotlivého simulačního experimentu je jedna konkrétní doba, za kterou se fronta vyprázdnila. Expeimentů je třeba vykonat dostatečný počet a zjištěné doby statisticky zpracovat.

Pokud nás zajímá, jak simulovaný systém funguje "v ustáleném stavu", kdy je vliv počátečních podmínek zanedbatelný, je vhodné počáteční úsek simulace, který je počátečním stavem ovlivněn, nepočítat do výsledných statistik. Toto by se mělo udělat při každém simulačním experimentu.

Často je ovšem možné namísto mnoha simulačních experimentů provést jen jeden experiment dostatečně dlouhý. Tak dlouhý, aby vliv počátečního úseku na výsledné statistiky byl zanedbatelný. Příklad: zjištění průměrného počtu zákazníků ve frontě.

13.1.4 Jednoduchý příklad. Ve skladu máte 100 kusů zboží. Máte statisticky zjištěno, že velikosti poptávky v jednotlivých dnech jsou nezávislé náhodné veličiny se stejným rozložením daným touto tabulkou:

poptávka	pravděpodonost	poptávka	pravděpodonost
0	0.10	7	0.13
1	0.02	8	0.14
2	0.03	9	0.12
3	0.05	10	0.07
4	0.07	11	0.04
5	0.09	12	0.02
6	0.11	13	0.01

Dostali jste zprávu, že plánovaná dodávka zboží bude opožděná, zboží přijde až za 14 dní. Otázka zní, kdy začne zboží chybět a jaká je pravděpodobnost, že zásoba vystačí na celých 14 dní.

Tuto úlohu lze řešit i přesným výpočtem, k tomu bychom však potřebovali trochu teorie (tzv. Markovovy řetězce).

K sestavení simulačního modelu stačí trocha selského rozumu, počítač a schopnost napsat jednoduchý program. Výsledkem simulace metodou Monte Carlo ovšem nebude přesná hodnota.

Simulační experiment bude velmi jednoduchý: Vygenerujeme náhodnou poptávku pro jednolivé dny, tj. 14 náhodných čísel s rozložením daným tabulkou. Tato čísla budeme postupně odečítat od výchozího stavu zásob, dokud buď zásoba neklesne pod nulu nebo dokud nevyčerpáme všech 14 čísel. Výsledkem simulačního experimentu bude buď pořadové číslo dne, kdy poprve zásoba klesla pod nulu (tj. kdy začala zásoba chybět), a nebo konstatování, že zásoba vystačila na celých 14 dní.

Takových simulačních experimentů provedeme velký počet. Statistickým zpracováním výsledků snadno zjistíme střední počet dní, než zásoba začala chybět, popř. pravděpodobnost, že zásoba vystačí.

13.1.5 Simulace Poissonova procesu. Časové intervaly mezi událostmi Poissonova procesu mají exponenciální rozložení s parametrem λ, kde λ je intenzita procesu (viz 10.2.4.

Stačí tedy generovat náhodná čísla s exponenciálním rozložením (podle 9.4.5). Čas, kdy nastane další událost dostaneme tak, že k času, kdy nastala událost předchozí, přičteme vždy další z vygenerovaných čísel s exponenciálním rozložením.

13.1.6 Simulace jednokanálového systému hromadné obsluhy s režimem fronty FIFO je také velmi jednoduchá.

Pro každého zákazníka nás zajímají tři časové údaje

- čas příchodu do systému,
- čas začátku obsluhy a
- čas konce obsluhy.

Pokud je vstupní proces Poissonovský, pak časy příchodů nezávisí na tom, co se v systému děje a můžeme je vygenerovat podle 13.1.5. Jiný typ vstupního procesu by se samozřejmě musel simulovat jinak.

Doby obsluhy také můžeme generovat nezávisle na sobě.

Čas začátku obsluhy každého zákazníka určíme jako maximum z času příchodu tohoto zákazníka do systému a času konce obsluhy předchozího zákazníka. Výsledek simulace může vypadat např. takto:

č.	příchod	začátek obsluhy	konec obsluhy
1.	0	0	15
2.	6	15	19
3.	14	19	27
4.	23	27	30
5.	34	34	37
6.	36	37	42

Jako cvičení zjistěte, kolikrát byl systém prázdný a vypočtěte průměrný počet zákazníků v systému jednak za období 0–40, jednak za období 10–35.

Prakticky použitelná simulace by samozřejmě musela být mnohem delší.

13.2 Simulace s pevným časovým krokem

Podstatou simulace s pevným časovým krokem je periodické zjišťování stavu modelu v pravidelně rozložených okamžicích. Ve vhodných proměnných máme v počítači uloženy hodnoty, které popisují stav systému v okamžiku t_i. V nějaké proměnné (obvykle pojmenované `time`) máme zpravidla také uloženu hodnotu modelového času t_i. Srdcem simulačního výpočtu pak je úsek programu, který na základě předchozího stavu (v okamžiku t_i) vypočte stav systému v následujícím časovém okamžiku $t_{i+1} = t_i + \Delta$, kde Δ je (pevná) délka časového kroku.

Simulace s pevným časovým krokem se používá zejména k simulaci spojitých dynamických systémů, v nichž se stavové veličiny mění (převážně) spojitě. Jako příklady takových systémů lze uvést pohyb pístu ve spalovacím motoru nebo vývoj počasí v nějaké oblasti. Takové systémy bývají často popsány soustavou diferenciálních rovnic. Volba časového kroku při simulaci spojitých systémů není jednoduchá a má obvykle velký vliv na přesnost výpočtu.

Další oblastí, kde se používá simulace s pevným časovým krokem, jsou procesy, jejichž stavy se sice mění skokem, ale okamžiky, kdy dochází ke změnám stavů jsou pravidelné. Zde délka časového kroku přirozeným způsobem závisí na frekvenci změn stavu systému.

13.2.1 Příklad — P-systém řízení zásob. Předpokládejme, že máme statisticky zjištěno, že denní odběr materiálu ze skladu je náhodný s těmito pravděpodobnostmi:

odběr	pravděpodobnost	odběr	pravděpodobnost
0	0.10	7	0.13
1	0.02	8	0.14
2	0.03	9	0.12
3	0.05	10	0.07
4	0.07	11	0.04
5	0.09	12	0.02
6	0.11	13	0.01

Materiál do skladu je dodáván ve čtrnáctidenních intervalech vždy každý druhý pátek odpoledne po pracovní době. Velikost dodávky je třeba upřesnit vždy předchozí úterý odpoledne, rovněž po pracovní době. Objednává se vždy takové množství, které v okamžiku objednávky chybí do kapacity skladu. Kapacita skladu je 70 kusů.

Skladování jednoho kusu po jeden den stojí 30 Kč. Skladovací náklady se počítají pouze za pracovní dny a to podle velikosti zásoby na konci směny. Fixní náklady na provoz skladu (nezávislé na poptávce a dodávkách) jsou 300 Kč za každý pracovní den.

Požadavky na odběr, které není možno okamžitě uspokojit (neboť došlo k vyčerpání skladu), se odloží na dobu, kdy bude materiál dodán, zdržení však představuje ztrátu (penále) 500 Kč za každý kus a pracovní den.

Úkolem je zjistit průměrné náklady spojené s provozem skladu.

Řešení:

Tuto úlohu lze řešit simulací s pevným časovým krokem o délce 1 den. Vzhledem k tomu, že mimo pracovní dny se v systému nic neděje, můžeme čas měřit na pracovní dny. Dodávky tedy přicházejí každý desátý den a objednáváme, když číslo dne dává při dělení desíti zbytek 7.

```
const
  kapacita = 70;          { kapacita skladu }
  perioda  = 10;          { délka periody objednávámí }

var
  time       : integer;    { hodnota modelového času }
  sklad      : integer;    { množství na skladě }
  objednavka : integer;    { kolik je objednáno }
  naklady    : longint;    { suma nákladů od začátku simulace }

function  poptavka : integer;        { funkce pro generování }
    const                              { náhodné poptávky      }
      P : array[0..13] of real =
        (0.10, 0.02, 0.03, 0.05, 0.07, 0.09, 0.11,
         0.13, 0.14, 0.12, 0.07, 0.04, 0.02, 0.01);
    var  i:integer;  x:real;
  begin
    x := random;  i := -1;
    repeat
      i := i + 1;
      x := x - P[i];
    until  x <= 0;
    poptavka := i;
  end;

begin
  randseed:=9236789;   { inicializace generátoru náhodných čísel }
  time:=0;
  sklad:=kapacita;     { inicializace stavu systému }
  objednavka:=0;
```

```
  naklady:=0;

  repeat
    time := time + 1;

    sklad := sklad - poptavka;

    if  sklad > 0  then   naklady:=naklady+300+sklad*30
                   else   naklady:=naklady+300-sklad*500;

    if  (time mod 10)=7  then            { je-li doba pro objednání,    }
      objednavka := kapacita - sklad;    {   určíme velikost objednávky }

    if  (time mod 10)=0  then begin      { je-li doba příchodu dodávky, }
      sklad := sklad + objednavka;       {   započteme dodané množství  }
      objednavka := 0;                   {   čímž je objednávka splněna }
      end;
  until  time >= 5000;         { test ukončení simulace }

  writeln ('Naklady = ', naklady, '   čas = ', time,
           '   průměrné náklady = ', naklady/time:10:3);
end.
```

Výsledkem práce tohoto simulačního programu je řádka

```
Naklady = 10304440   čas = 5000   průměrné náklady =  2060.888
```

13.2.2 Cvičení. Experimentováním se simulačním programem z předchozího příkladu 13.2.1, str. 153 najděte optimální kapacitu skladu. Vyzkoušejte vliv inicializace generátoru a vliv délky simulace.

13.2.3 Cvičení. V příkladě 13.2.1, str. 153 vyčíslete úspory, které by plynuly ze zkrácení dodací lhůty o jeden den při pevné kapacitě skladu 70 kusů.

13.2.4 Cvičení. V příkladě 13.2.1, str. 153 vyčíslete úspory, které by plynuly ze zkrácení dodací lhůty o jeden den při optimální kapacitě skladu.

13.2.5 Cvičení. Simulační program z příkladu 13.2.1, str. 153 upravte tak, aby se provozní a skladovací náklady počítaly za všechny dny, tj. nejen pracovní. Penále počítejte opět jen za dny pracovní.

13.2.6 Cvičení. Napište program pro simulaci Q-systému řízení zásob: Objednávání se neprovádí pravidelně (jako v P-systému), ale tehdy, když velikost zásoby klesne pod předem stanovenou tzv. signální úroveň.

Návod: vyjděte z programu z příkladu 13.2.1, str. 153. Do tohoto programu doplňte konstanty `signalniUroven` a `zpozdeni`. Druhá z nich bude určovat kolik dnů po objednání přijde dodávka. Dále použijte proměnnou `kdyObjednano` a použijte ji spolu s proměnnou `zpozdeni` k rozpoznání okamžiku, kdy přišla dodávka.

13.3 Simulace s proměnným časovým krokem

Dochází-li ke změnám stavu simulované soustavy v diskrétních hodnotách času a nejsou-li okamžiky změn rovnoměrně rozloženy, používá se při simulaci proměnný časový krok. Nemá totiž smysl zabývat se stavem systému v okamžicích, kdy se v systému nic neděje. To znamená, že hodnoty času,

v nichž budeme systém sledovat, nelze stanovit předem, tyto hodnoty vyplynou ze simulovaného děje až v průběhu simulace.

Simulace s proměnným časovým krokem se často používá při řešení úloh z oblasti teorie front.

Při simulaci s proměnným časovým krokem obvykle využíváme tzv. *kalendář událostí*. Jsou v něm vždy zaznamenány všechny události, o nichž se v daném modelovém čase ví, kdy nastanou. O každé události je v kalendáři uveden (plánovaný) čas, kdy k události dojde, a dále informace, o kterou událost se jedná. Například při simulaci systému hromadné obsluhy budeme pracovat se dvěma typy událostí: ukončení obsluhy zákazníka a příchod nového zákazníka do systému.

Z kalendáře vždy vybíráme nejbližší událost, tj. událost s nejnižší hodnotou plánovaného času. Proměnnou, vyjadřující modelový čas, nastavíme na tuto hodnotu a provedeme změny stavových proměnných, které jsou bezprostředně způsobeny právě zpracovávanou událostí. Přitom může (ale nemusí) dojít k naplánování dalších událostí, které jsou důsledkem právě provedených změn stavu. Je-li vybraná událost takto zpracována, celý postup se opakuje: vybereme další událost, zpracujeme ji, vybereme další, atd.

Simulační programy lze samozřejmě psát v jakémkoli universálním programovacím jazyce, metoda s proměnným krokem však zpravidla vede k ne zcela jednoduchým a přehledným programům. Dodejme, že také existují jednak speciální jazyky pro psaní simulačních programů (SIMULA 67), jednak speciální programy, které pro určitou třídu modelů dovedou vygenerovat simulační program na základě poměrně jednoduchého popisu ve speciálním jazyce.

13.3.1 Příklad — vícekanálový systém hromadné obsluhy. Mějme vícekanálový systém hromadné obsluhy M/M/n, kde n je počet kanálů obsluhy. Intensita příchodů je 6 zákazníků za hodinu, průměrná doba obsluhy je 7 minut.

Úkolem je zjistit průměrný počet, kolikrát za hodinu bude systém prázdný.

Řešení: Kalendář událostí je v programu uložen v polích `Plan` a `Aktivni` a to tak, že pro `i>0` hodnota `Aktivni[i]` určuje, je-li `i`-tý kanál v činnosti a `Plan[i]` je čas plánovaného ukončení obsluhy. Hodnota `Plan[0]` je čas plánovaného příchodu dalšího zákazníka.

Poznamenejme, že to je velmi primitivní a ne moc efektivní způsob realizace kalendáře událostí. Navíc je tento způsob použitelný pouze v tomto speciálním případě.

```
const
  N             =  3;           { počet kanálů obsluhy        }
  prich  : real = 10;           { průměrná doba mezi příchody }
  obsl   : real =  7;           { průměrná doba obsluhy       }

var
  time    : real;                       { modelový čas }
  Plan    : array [0..N] of real;       { plánovaný čas v~kanále i    }
  Aktivni : array [1..N] of boolean;    { je-li kanál i aktivní       }
  vSystemu: integer;                    { počet zákazníků v~systému   }
  prazdny : integer;                    { kolikrát byl systém prázdný }
  i, kanal: integer;

function  negexp (prumer:real) : real;   { generátor náhodných čísel   }
  begin                                  { s~exponenciálním rozložením }
    negexp:=-ln(random)*prumer;
  end;

begin
  time:=0.0;                               { inicializace }
  vSystemu:=0;                             { na začátku je systém prázdný }
  for  i:=1 to N  do  Aktivni[i]:=false;
  Plan[0]:=0.0;                 { čas měříme od příchodu prvého zákazníka }
  prazdny:=0;
```

```
  repeat                          { hlavní cyklus }
    time:=Plan[0];                          { z˜kalendáře událostí       }
    kanal:=0;                               { vybereme nejbližší událost }
    for i:=1 to N do
      if  Aktivni[i] and (time > Plan[i])  then begin
        time:=Plan[i];
        Kanal:=i;
        end;

    if  kanal=0  then begin      { typ události = příchod zákazníka }

      Plan[0] := time + negexp(prich);    { plán příchodu zákazníka }

      if  vSystemu < N  then begin        { je-li volný kanál obsluhy, }
        for  i:=1 to N  do                {   najdeme,                 }
          if not Aktivni[i] then kanal:=i;{     který to je            }
        Plan[kanal]:=time+negexp(obsl);   { a naplánujeme jeho ukončení}
        end;
      vSystemu:=vSystemu+1;               { v˜systému je o˜zákazníka víc }
      end

    else begin  { kanal > 0 }  { typ události = konec obsluhy v˜kanále }

      Aktivni[kanal]:=false;             { kanál už není aktivní         }
      vSystemu:=vSystemu-1;              { v˜systému je o˜zákazníka méně }

      if  vSystemu >= N  then begin      { je-li někdo ve frontě,   }
        Aktivni[kanal]:=true;            {   zahájíme jeho obsluhu  }
        Plan[kanal]:=time+negexp(obsl);  {   a naplánujeme ukončení }
        end;

      if vSystemu=0 then prazdny:=prazdny+1;  { počítání prázdného systému }
      end;
  until  time >= 10000;            { test konce simulace }

  writeln ('Systém byl prázdný ', prazdny, '-krát');
  writeln ('tj. průměrně za hodinu ', prazdny*60.0/time:10:3, '-krát')
end.
```

Tento simulační program vytiskne jako výsledek

```
Systém byl prázdný 495-krát
tj. průměrně za hodinu      2.968-krát
```

13.3.2 Cvičení. Upravte simulační program z příkladu 13.3.1, str. 156 tak, aby vypočítal průměrný počet zákazníků v systému a průměrný počet aktivních kanálů obsluhy.

13.3.3 Cvičení. Představte si, že systém z příkladu 13.3.1, str. 156 obsahuje při svém spuštění 100 zákazníků ve frontě. Upravte simulační program tak, aby zjistil čas, kdy dojde k vyprázdnění systému.