Kapitola 8

Toky v sítích

Problematika toků v sítích tvoří jednu z teoreticky zajímavých a současně aplikačně velmi bohatých partií teorie grafů. Orientovaný graf je zde modelem sítě (např. dopravní, vodovodní, naftovodní, plynové, spojové apod.), po níž se přepravuje určité médium. Přepravní kapacita každé hrany je nějakým způsobem omezena, a tím jsou omezeny i celkové přepravní možnosti sítě, ať již mezi konkrétní dvojicí uzlů nebo mezi všemi uzly navzájem. Typické úlohy o sítích mají tedy opět optimalizační charakter, algoritmy jejich řešení se však přes určité společné rysy liší od těch, kterým jsme se věnovali v předchozích dvou kapitolách.

Významným impulsem pro studium toků bylo vydání monografie [11], od té doby bylo formulováno a řešeno mnoho variant úloh na sítích. V tomto textu se věnujeme především základní variantě – určení maximálního toku mezi dvěma zvolenými uzly sítě – a stručně naznačíme některé další jednoduché varianty.

8.1 Síť a úlohy na sítích

Definice 8.1: Sítí nazýváme čtveřici $S = \langle G, q, s, t\rangle$, kde $G = \langle H, U\rangle$ je obyčejný orientovaný graf, $s \in U$ je **zdroj sítě** S, $t \in U, t \neq s$ je **spotřebič sítě** S a $q : H \mapsto \mathbf{R}^+$ je nezáporné ohodnocení hran, nazývané **kapacita sítě** S. Pro každou hranu $h \in H$ nazýváme příslušnou hodnotu $q(h)$ **kapacitou hrany** h.

Jelikož kromě kapacity používáme ještě další ohodnocení hran sítě, značíme pro jednoduchost kapacitu hrany $h = (u, v)$ jako $q(u, v)$ nebo přímo q_{uv}. Obdobně postupujeme i u ostatních ohodnocení, z nichž první bude matematickým vyjádřením velikosti po hraně přepravované entity.

Definice 8.2: Tokem v síti $S = \langle G, q, s, t\rangle$ nazýváme takové hodnocení hran $f : H \mapsto \mathbf{R}^+$, které splňuje následující podmínky:

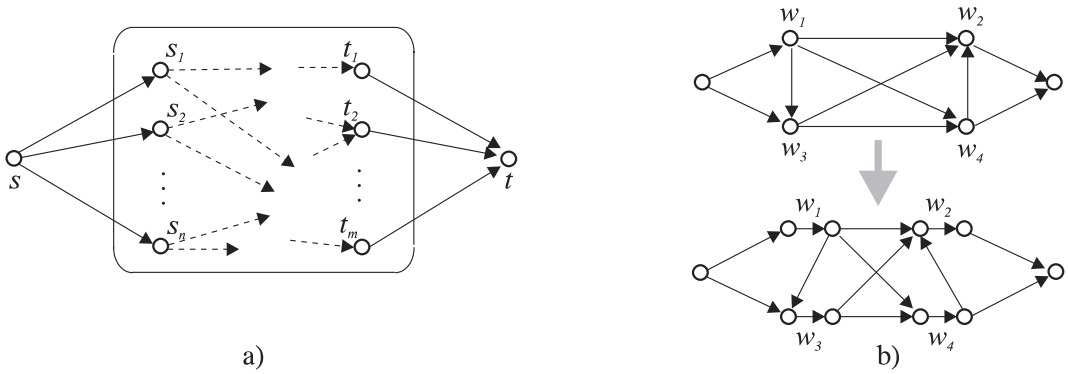
- pro každou hranu $(u, v) \in H$ platí $0 \leq f(u, v) \leq q(u, v)$
- pro každý uzel u sítě S různý od s i od t platí

$$\sum_{(u,v)\in H} f(u, v) - \sum_{(w,u)\in H} f(w, u) = 0 \tag{8.1}$$

Velikost $|f|$ **toku** f je dána rozdílem

$$|f| = \sum_{(s,v)\in H} f(s, v) - \sum_{(u,s)\in H} f(u, s) = \sum_{(u,t)\in H} f(u, t) - \sum_{(t,v)\in H} f(t, v). \tag{8.2}$$

První z podmínek stanoví, že tok hranou nesmí překročit její kapacitu. Druhá podmínka vyjadřuje skutečnost, že tok se v žádném vnitřním uzlu sítě nehromadí – kolik se jej přivede do uzlu vstupními hranami, tolik se jej také odvede hranami výstupními. Tuto podmínku nemusí



Obrázek 8.1: Síť a) s více zdroji a spotřebiči, b) s omezenou kapacitou uzlů

splňovat zdroj s ani spotřebič t – ve zdroji tok vzniká a ve spotřebiči pak stejné množství toku zaniká. Toto množství udává velikost toku.

Pokud se splnění podmínky 8.1 požaduje i pro zdroj a spotřebič, jedná se o tzv. **cirkulaci v síti**. Tok od zdroje ke spotřebiči převedeme na cirkulaci přidáním zpětné hrany (t, s) s neomezenou kapacitou.

Základní úlohou v síti je **nalezení maximálního toku**, tedy toku o maximální velikosti. Kromě této úlohy existuje řada variant, jejichž řešení lze získat buď převodem na základní úlohu nebo pomocí zvláštních postupů.

Sítě s více zdroji a spotřebiči

V síti existuje více zdrojů a/nebo spotřebičů a hledáme maximální celkový tok ze zdrojů do spotřebičů. Úlohu převedeme na základní variantu přidáním jednoho umělého zdroje spojeného se skutečnými zdroji hranami s nekonečnou kapacitou a jednoho umělého spotřebiče, do něhož vedou ze skutečných spotřebičů hrany s nekonečnou kapacitou (viz obr. 8.1a).

Sítě s omezenou kapacitou uzlů

Některé (nebo všechny) uzly sítě mohou mít stanovenou maximální hodnotu toku, který jsou schopny propustit. To znamená, že např. pro uzel v musí platit

$$\sum_u f(u, v) \leq w_v,$$

kde se sčítá přes všechny předchůdce $u \in \Gamma^{-1}(v)$ uzlu v. V tomto případě rozdělíme uzel v na dva uzly v^+ a v^- spojené hranou s kapacitou w_v. Do uzlu v^+ přivedeme všechny hrany, které dříve vstupovaly do v, a z uzlu v^- vyvedeme všechny hrany, které dříve vystupovaly z uzlu v. Na obr. 8.1b ukazujeme síť s omezenou kapacitou všech uzlů a ekvivalentní síť získanou popsaným způsobem. Kapacita původních hran sítě se nemění.

Sítě s omezeným minimálním tokem

Pro tento druh sítí má každá hrana vedle kapacity přiřazenu i minimální hodnotu toku $r(u_i, u_j)$, takže první podmínka v definici toku je nahrazena obecnější podmínkou

$$0 \leq r(u, v) \leq f(u, v) \leq q(u, v) \quad \text{pro všechny hrany } (u, v) \in H.$$

Takto formulovaná úloha vůbec nemusí mít řešení, neboť pro síť nemusí existovat přípustný tok dosahující stanovených minimálních hodnot. Postup řešení se tak rozpadá na dvě části – nejprve se získá počáteční přípustný tok (pokud nějaký existuje), a pak se od něj iteruje k maximálnímu toku. Podrobněji postup řešení popíšeme na konci dalšího odstavce.

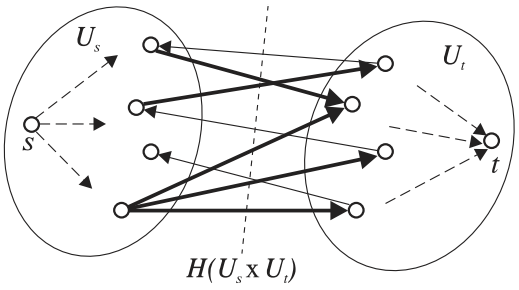
Sítě s oceněným tokem

Jedná se o další zobecnění předchozí varianty, kdy má každá hrana $h = (u, v)$ vedle omezení toku stanovenu i jednotkovou cenu $c(h)$ toku hranou h. Má-li tok hranou h hodnotu $f(h)$, pak je celková cena toku hranou h rovna $f(h) \cdot c(h)$. Úloha spočívá ve stanovení nejlevnější přípustné cirkulace, přičemž cena toku v síti je dána součtem cen v jednotlivých hranách:

$$c(f) = \sum_{(u,v)\in H} f(u,v) \cdot c(u,v).$$

Vhodným doplněním sítě nebo stanovením cen toku lze na tuto úlohu převést např. hledání maximálního toku (viz např. [10]).

Zaměříme se nyní na zkoumání těch vlastností sítě, které budou užitečné při určování maximálního toku. Pro síť $S = \langle G, q, s, t\rangle$ uvažujeme nejprve množiny hran grafu G, po jejichž odebrání se budou zdroj s a spotřebič t nacházet v různých komponentách. Takovou množinu hran nazýváme **řezem sítě** S a určujeme příslušným rozkladem množiny uzlů $\{U_s, U_t\}$, $s \in U_s, t \in U_t$, zapisujeme $H(U_s \times U_t)$. **Kapacitou řezu** sítě nazveme číslo



Obrázek 8.2: Kapacita hranového řezu

$$q(U_s \times U_t) = \sum_{(u,v)\in(U_s\times U_t)\cap H} q(u,v).$$

Při určení kapacity řezu sítě se tedy uvažují pouze hrany směřující z množiny U_s do množiny U_t (viz tučně vytažené hrany na obr. 8.2). Vztah mezi toky a řezy v síti charakterizuje následující tvrzení.

Věta 8.3: Nechť f je libovolný tok a $H(U_s \times U_t)$ libovolný řez sítě $S = \langle G, q, s, t\rangle$. Potom platí

$$|f| \le q(U_s \times U_t).$$

Důkaz: Při následujících úpravách pro přehlednost nevyjadřujeme, že se ve všech součtech sčítá pouze přes hrany sítě.

$$\begin{aligned}
q(U_s \times U_t) &= \sum_{(u,v)\in U_s\times U_t} q(u,v) = \sum_{(u,v)\in U_s\times U_t} f(u,v) - \sum_{(w,y)\in U_t\times U_s} f(w,u) = \\
&= \sum_{(u,s)} \left[\sum_{(u,v)\in H} f(u,v) - \sum_{v\in U_s} f(u,v) - \sum_{(u,v)\in H} f(w,u) + \sum_{w\in U_s} f(w,u) \right] = \\
&= \sum_{w\in U_s} \left[\sum_{(u,v)\in H} f(u,v) - \sum_{(w,u)\in H} f(w,u) \right] - \\
&- \sum_{(u,v)\in U_s\times U_s} f(u,v) + \sum_{(w,u)\in U_s\times U_s} f(w,u) = \\
&= \sum_{(s,v)} f(s,v) - \sum_{(w,s)} f(w,s) = |f|.
\end{aligned}$$

△

Věta 8.4: Tok f je maximálním tokem v síti $S = \langle G, q, s, t\rangle$, právě když neexistuje neorientovaná cesta $P = \langle u_0, u_1, \ldots, u_n\rangle, u_0 = s, u_n = t$, pro kterou platí

(a) $f(u_i, u_{i+1}) < q(u_i, u_{i+1})$, je-li $(u_i, u_{i+1}) \in P$

(b) $f(u_{i+1}, u_i) > 0$, je-li $(u_{i+1}, u_i) \in P$.

Důkaz: Nechť taková cesta existuje. Ukážeme, že velikost toku f je možné zvětšit právě podél cesty P – cestu těchto vlastností budeme nazývat **zlepšující cestou**. Označme

$$\begin{aligned} \delta_a &= \min(q(u_i, u_{i+1}) - f(u_i, u_{i+1})) \\ \delta_b &= \min f(u_{i+1}, u_i), \end{aligned}$$

kde v prvém případě uvažujeme minimum pro všechny hrany cesty P orientované ve směru $s \to t$, a vyhovující tedy podmínce (a), a ve druhém případě uvažujeme zbývající hrany cesty P orientované podle podmínky (b) proti směru $s \to t$. Položíme-li nyní

$$\delta = \min(\delta_a, \delta_b),$$

můžeme tok všech hran první skupiny zvětšit o δ a tok všech hran druhé skupiny zmenšit o δ (viz obr. 8.3). Tím se neporuší podmínky přípustnosti toku a získáme tok větší velikosti, než má f, takže f nemůže být maximální.

Nechť naopak žádná taková cesta P neexistuje. Potom označíme jako U_s množinu uzlů v sítě S, pro něž existuje cesta z s do v, jejíž hrany splňují podmínky (a), (b) uvedené ve větě. Zřejmě $t \notin U_s$ a množina hran $H(U_s \times U_t)$, kde $U_t = U - U_s$, tvoří řez sítě, pro jehož hrany platí:

$$f(x, y) = \begin{cases} q(x, y), & \text{je-li } (x, y) \in U_s \times U_t \\ 0, & \text{je-li } (x, y) \in U_t \times U_s. \end{cases}$$

Odtud dostaneme obdobným postupem jako v důkazu předchozí věty, že platí $q(U_s \times U_t) = |f|$, a podle věty 8.3 nemůže tedy existovat tok větší velikosti než má f. △

Věta 8.5: (Fordova-Fulkersonova) Velikost maximálního toku v síti je rovna kapacitě jejího minimálního řezu.

Důkaz: Podle věty 8.3 nemůže být velikost maximálního toku větší než kapacita řezu sítě. Podle druhé části důkazu věty 8.4 můžeme k maximálnímu toku sestrojit řez sítě s kapacitou rovnou velikosti tohoto toku – takový řez bude tedy mít minimální kapacitu. △

Přehled dalších variant úloh na síti spolu s odpovídajícími aplikacemi lze nalézt v [10].

8.2 Maximální tok v síti

Tvrzení věty 8.4 můžeme použít jako návod k vytvoření algoritmu pro určení maximálního toku. Vyjdeme z libovolného přípustného (např. nulového) toku a pomocí označování uzlů hledáme cestu, podél které bude možné tok zvětšit. Označování uzlů se provádí tak, aby se průběžně počítala hodnota δ, o kterou se tok zvětší. Po úpravě toku se označení uzlů zruší, a začíná se opět od začátku, což se opakuje tak dlouho, až se dosáhne maximálního toku.

Podstatnou část následujícího algoritmu představuje hledání zlepšující cesty v grafu – to zajistíme způsobem obdobným jako u prohledávání grafu. Každý uzel může být ve stavu FRESH (dosud nenalezen), OPEN (před expanzí) a CLOSED (po expanzi). Protože se hledá neorientovaná cesta, během expanze se procházejí nejen následníci, ale také předchůdci uzlu. Složka $p[u]$ každého uzlu určuje jednak předchozí uzel na zvyšující cestě, jednak orientaci příslušné

Obrázek 8.3: Zvýšení toku podél cesty

hrany ve směru nebo proti směru $s \to t$. Složka $d[u]$ vyjadřuje, o jakou hodnotu by bylo možné zvýšit tok od zdroje do uzlu u.

Po nalezení cesty se počínaje od spotřebiče upravují toky v hranách zvyšující cesty o hodnotu $\delta = d[t]$. Přitom se využívá informace, jak je příslušná hrana orientována vzhledem ke zvyšující cestě. Hranu, jejíž tok dosáhl hodnoty kapacity, nazýváme **nasycenou**.

Algoritmus 8.6 Fordův-Fulkersonův algoritmus výpočtu maximálního toku

Najdi-Cestu(S)

1	**for** každý uzel $u \in U$	Označení všech uzlů
2	**do** $stav[u]$:=FRESH	jako nenalezených.
3	$p[s] := +s; d[s] := \infty; stav[s]$:=OPEN	Začátek hledané cesty.
4	**repeat** u:= libovolný otevřený uzel	Výběr uzlu pro pokračování:
6	$stav[u]$:=CLOSED	nejprve se uzavře,
7	**for** každý uzel $v \in \Gamma(u)$ **do**	všichni noví následnící
8	**if** ($stav[v]$=FRESH) & ($f(u,v) < q(u,v)$) **then**	se otevřou,
9	$stav[v]$:=OPEN; $p[v] := +u$;	určí se kladný předchůdce
10	$d[v] := \min(d[u], q(u,v) - f(u,v))$	a možný přírůstek toku.
11	**for** každý uzel $v \in \Gamma^{-1}(u)$ **do**	Pro jeho předchůdce
12	**if** ($stav[v]$ =FRESH) & ($f(v,u) > 0$) **then**	se provede totéž,
13	$stav[v]$:=OPEN; $p[v] := -u$;	pouze záporný předchůdce
14	$d[v] := \min(d[u], f(v,u))$	a jiný možný přírůstek.
15	**until** (neexistuje otevřený uzel) $\vee$ ($u = t$)	
16	**return** $u = t$	TRUE při nalezení cesty

Zvyš-Tok(S)

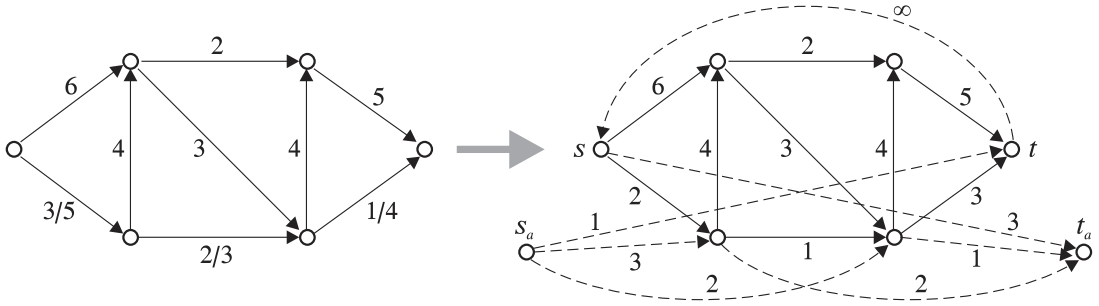
1	$x := t; \delta := d[t]$	Začíná se
2	**repeat** $v := x$	od spotřebiče t.
3	**if** $p[v] = +u$	Hrana je ve směru,
4	**then** $f(u,v) := f(u,v) + \delta$	zvyšuje se její tok,
5	**else** $f(v,u) := f(v,u) - \delta$	jinak se snižuje protitok.
6	$x := u$	Přechod na předchůdce,
7	**until** $v = s$	dokud nejsme u zdroje.

FORD-FULKERSON(S)

1	**for** každou hranu $(u,v) \in H$	Počáteční nastavení
2	**do** $f(u,v) := 0$	toku na nulu.
3	**while** Najdi-Cestu(S)	Dokud existuje cesta,
4	**do** Zvyš-Tok(S)	zvyšuj tok v síti.
5	**return** f	

Určení asymptotické časové složitosti tohoto algoritmu je problematické. Nalezení zvyšující cesty lze provést v čase $O(|U| + |H|)$, zvýšení toku dokonce v čase $O(|U|)$, ale nelze určit, kolikrát se tyto operace budou v hlavním cyklu opakovat. Ukončení výpočtu je totiž obecně zaručeno pouze pro racionální hodnoty kapacit hran, pro iracionální hodnoty nemusí cyklus nikdy skončit. I pro racionální kapacity však může počet průchodů cyklem záviset na jejich hodnotách, což je pro odhad časové složitosti velmi nežádoucí situace.

Této nepříjemnosti se lze vyhnout použitím Karpovy a Edmondsovy úpravy Fordova-Fulkersonova algoritmu: při hledání zvyšující cesty prohledáváme síť od zdroje do šířky, takže cesta bude (co do počtu hran) nejkratší. Uvažujeme-li celočíselné hodnoty kapacit (od racionálních bychom k nim přešli vynásobením nejmenším společným jmenovatelem), pak uvedená modifikace algoritmu nalezne maximální tok v čase $O(|U| \cdot |H|^2)$. Dalšími zlepšeními získal Dinic algoritmus složitosti $O(|U|^2 \cdot |H|)$ a Karzanov dokonce $O(|U|^3)$ (viz [23]).



Obrázek 8.4: Úprava sítě s omezeným minimálním tokem

Velmi rychlé určení maximálního toku dovolují planární sítě, ve kterých je možné propojit zdroj a spotřebič hranou bez porušení planarity. Takovou síť lze zakreslit do roviny tak, že zdroj je zcela vlevo, spotřebič zcela vpravo a všechny další uzly a hrany jsou uvnitř tohoto pásu. Jako zlepšující se vždy hledá nejhořejší cesta, která obsahuje jen hrany orientované ve směru $s \to t$. Tok hranou se pak pouze zvyšuje a v každém průchodu hlavním cyklem se alespoň jedna hrana stane nasycenou. Díky omezenosti počtu hran planárních grafů je časová složitost tohoto algoritmu pouze $O(|U|^2)$, po určitých úpravách (viz [23]) lze jeho čas snížit až na $O(|U| \lg |U|)$.

Věnujeme se ještě stručně jedné variantě úloh na síti, a sice jak určit maximální tok v síti s omezeným minimálním tokem. Nejprve je třeba zjistit, zda pro síť existuje přípustný tok. To provedeme určením maximálního toku v odvozené síti s nulovou dolní hranicí toku, kterou získáme takto:

- Síť doplníme umělým zdrojem s_a a umělým spotřebičem t_a a pro všechny hrany (u, v) mající $r_{uv} > 0$ zavedeme dvojici hran $(u, t_a), (s_a, v)$ s kapacitou r_{uv} (a dolní mezí 0).

- Snížíme kapacitu q_{uv} dotyčných hran na $q_{uv} - r_{uv}$ a dolní mez toku hranou (u, v) stanovíme rovnou nule.

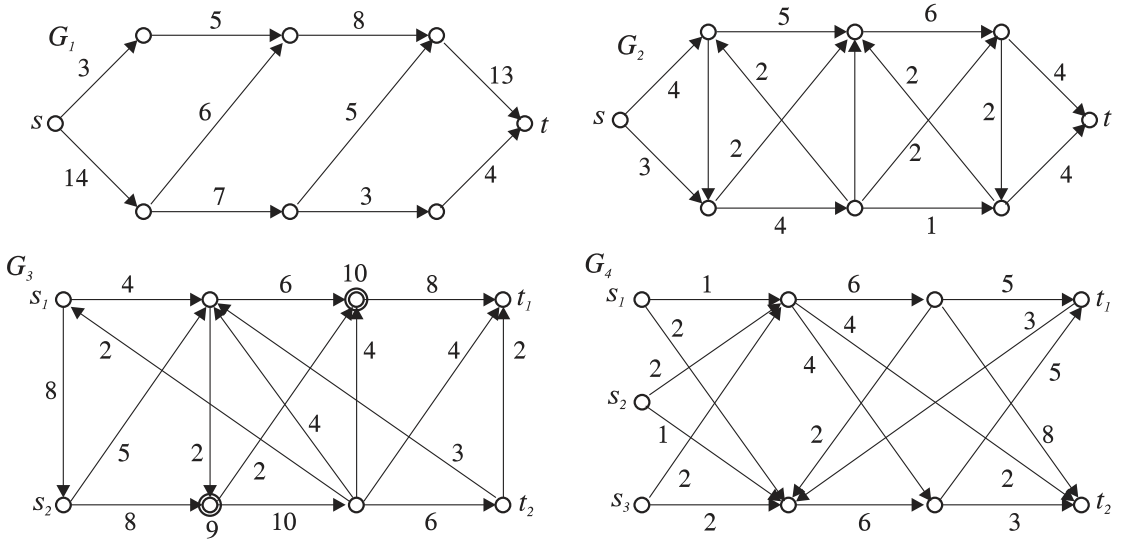
- Doplníme novou zpětnou hranu (t, s) s kapacitou $q_{ts} = \infty$ a dolní mezí 0 (viz příklad na obr. 8.4, hrany s omezeným minimálním tokem jsou označeny dvojicí $r_{u,v}/q_{u,v}$).

Má-li takto vzniklá síť maximální tok (z s_a do t_a) nasycující všechny přidané hrany, pak lze v původní síti nalézt přípustný tok velikosti f_{ts}, jinak přípustný tok neexistuje. Tento závěr vyplyne z následující úvahy.

Odebereme-li tok velikosti r_{uv} z hran (s_a, v) a (u, t_a) a přidáme-li jej hraně (u, v), pak se tok z s_a do t_a zmenší o r_{uv}, hodnota toku hranou (u, v) bude ležet v intervalu $\langle r_{uv}, q_{uv} \rangle$ a tok zpětnou hranou (t, s) zůstává nezměněn. Takto redukujeme tok z s_a do t_a na nulovou hodnotu a přidané umělé uzly se stávají zbytečnými. Vypustíme-li je a spolu s nimi i přidanou zpětnou hranu (t, s), dostaneme původní síť s tokem velikosti f_{ts} vyhovujícím podmínce na dolní mez toku hranami.

Určení počátečního přípustného toku s případnou signalizací jeho neexistence předřadíme hlavnímu cyklu Fordova-Fulkersonova algoritmu, v němž pak upravíme v proceduře NajdiCestu řádek 12 (požadujeme $f(v, u) > r(v, u)$) a řádek 14 (přiřazujeme $\min(\dot{[}u], f(v, u) - r(v, u))$). Hlavní cyklus pak zvyšuje tok až do nalezení maxima.

Aparát pro popis a řešení problémů toků v sítích má využití ve zdánlivě odlehlé „čistě grafové“ oblasti – při určování tzv. **maximálního párování** v grafu. Zavedeme tedy potřebné pojmy a uvedeme příslušný postup.



Obrázek 8.5: Párování v grafu

Definice 8.7: Nechť $G = \langle H, U, \rho \rangle$ je (orientovaný nebo neorientovaný) graf. **Párováním** v grafu G nazýváme takovou podmnožinu $P \subseteq H$ jeho hran, ve které žádné dvě hrany nemají společný uzel, tedy

$$\rho(h_1) \cap \rho(h_2) = \emptyset \quad \text{pro libovolné} \quad h_1, h_2 \in P, h_1 \neq h_2$$

Perfektní párování je takové, pro které je podgraf indukovaný jeho množinou hran P faktorem grafu G. **Maximální párování** v grafu G je takové, které obsahuje největší počet hran.

Jak je vidět, případná orientace hran nemá na definici ani vlastnosti párování žádný vliv. Budeme tedy předpokládat, že se párování týká neorientovaných grafů. Na obr. 8.5 ukazujeme příklady maximálních pokrytí úplných grafů K_4 a K_5, žádné pokrytí grafu K_5 zřejmě nemůže být perfektní. Vedle základní úlohy týkající se nalezení maximálního párování v zadaném grafu je možné pro grafy s ohodnocením hran $w : H \mapsto R^+$ formulovat ještě např. úlohu nalezení nejlevnějšího maximálního párování Q, resp. nejdražšího párování S, pro která platí

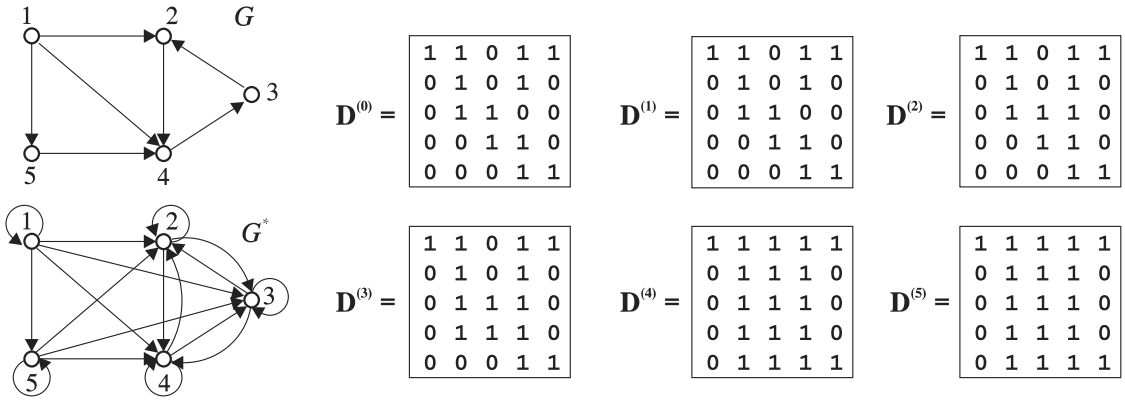
$$\begin{aligned} \Sigma_{h \in Q} w(h) &\leq \Sigma_{h \in P} w(h) \quad \text{pro libovolné maximální párování } P, \text{ resp.} \\ \Sigma_{h \in S} w(h) &\geq \Sigma_{h \in P} w(h) \quad \text{pro libovolné párování } P. \end{aligned}$$

Speciálním případem hledání nejlevnějšího maximálního párování je tzv. **přiřazovací úloha**, kdy se hledá nejlevnější perfektní párování v úplném bipartitním grafu $K_{n,n}$. Konkrétní variantou přiřazovací úlohy je např. nalezení nejlepšího výběru partnerských dvojic ve skupině tvořené stejným počtem mužů a žen.

Hledání maximálního párování v obecných grafech se provádí algoritmy, které mají podobný charakter jako Fordův-Fulkersonův algoritmus: začne se prázdným párováním a hledá se tzv. **střídavá cesta** v grafu, pomocí níž lze párování rozšířit. Časová složitost těchto algoritmů je $O(|U|^3)$ nebo i mírně nižší, jejich popis lze nalézt např. v [10]. V našem textu se problematice párování nebudeme podrobněji věnovat, takže pro ilustraci uvedeme jen algoritmus určení maximálního párování v bipartitním grafu.

Mějme dán bipartitní (neorientovaný) graf $G = \langle H, U, \rho \rangle$ a nechť $\{X, Y\}$ představuje rozklad množiny uzlů grafu G odpovídající bipartitnosti (tzn. každá hrana grafu G má jeden krajní uzel v X a druhý v Y). Graf G nyní doplníme následujícím způsobem na síť:

- přidáme nový uzel s jako zdroj sítě a propojíme jej orientovanou hranou (s, x) s každým uzlem $x \in X$



Obrázek 8.6: Sítě ke cvičení

- přidáme nový uzel t jako spotřebič sítě a propojíme jej orientovanou hranou (y, t) s každým uzlem $y \in Y$
- všechny původní hrany grafu G orientujeme ve směru od X do Y
- kapacitu všech hran takto vytvořené sítě položíme rovnu jedné

V takto vytvořené síti nalezneme maximální tok Fordovým-Fulkersonovým algoritmem. Vzhledem k popsanému způsobu vytvoření sítě a definici 8.7 obsahuje maximální párování grafu G právě ty jeho hrany, které mají v nalezeném maximálním toku přiřazen nenulový (tedy jednotkový) tok.

Cvičení

8.2-1. Určete kapacitu všech hranových řezů oddělujících uzly s a t v síti G_1 na obr. 8.6.

8.2-2. Určete maximální tok v síti G_1 z obr. 8.6. Je rozložení toku do jednotlivých hran určeno jednoznačně?

8.2-3. Určete maximální tok v sítích G_2, G_3, G_4 na obr. 8.6 (u sítě G_3 je omezená kapacita uzlů označených dvojitým kroužkem).

8.2-4. Uvažujme maximální párování v úplném grafu K_n.
(a) Určete počet hran v maximálním párování.
(b) Kolik různých maximálních párování graf K_n má?
(c) Jsou maximální párování v tomto grafu perfektní?

8.2-5. Uvažujme maximální párování v úplném bipartitním grafu $K_{m,n}, m \leq n$.
(a) Určete počet hran v maximálním párování.
(b) Kolik různých maximálních párování graf $K_{m,n}$ má?
(c) Jsou maximální párování v tomto grafu perfektní?

8.2-6. Určete, jakou největší a jakou nejmenší mohutnost může mít párování stromu s n uzly.

Kapitola 9

NP-úplné problémy

Řešení úloh, jimž jsme se až dosud podrobněji věnovali, byla reprezentována algoritmy s polynomiálně omezenou časovou složitostí. Existují ovšem také úlohy, které jsou algoritmicky neřešitelné (např. problém zastavení Turingova stroje), nebo úlohy se zaručeně vyšší než polynomiální složitostí (např. problém cykličnosti v atributových gramatikách). V prostoru mezi polynomiálně složitými úlohami na straně jedné a zaručeně superpolynomiálně složitými úlohami na straně druhé se nachází kompaktní skupina mnoha zajímavých a z praktického hlediska důležitých úloh, pro něž bylo zavedeno označení **NP-úplné problémy** a o nichž se dosud neví, kam vlastně patří.

Podnětem pro zavedení NP-úplných problémů bylo zjištění, že pro některé náročné úlohy je sice velmi obtížné nalézt jejich řešení, ale následné ověření správnosti tohoto řešení je již jednoduché. Tato třída úloh byla charakterizována jako **nedeterministicky řešitelné v polynomiálně omezeném čase** a dosud není známo, zda je to třída větší nebo shodná s třídou polynomiálně složitých úloh. Polynomiální složitost se chápe jako atribut prakticky zvládnutelných úloh, zatímco NP-úplné problémy se považují za první reprezentanty úloh nezvládnutelných. Existuje pro ně sice algoritmus řešení (typicky např. probíráním všech možností), ale ten dovoluje opravdu dosáhnout výsledku jen pro velmi omezený rozsah vstupních dat.

Studium algoritmických úloh z obecného hlediska vyžaduje zavedení formálního modelu počítače, pro který budou algoritmy řešení úloh určeny, spolu s určením dostupných operací a formy jejich vyjadřování v přepisu algoritmů. Je také třeba přesně vymezit, co se považuje za úlohu a v jakém tvaru ji předpokládáme formulovat. Přirozeným požadavkem přitom je, aby toto vymezení zahrnovalo prakticky zajímavé úlohy a současně nebylo nadbytečně široké.

Cílem této kapitoly je podat jen stručný nástin problematiky NP-úplnosti a přehled základních NP-úplných úloh, takže pomineme převážnou část jejích formálních základů. Podrobněji se NP-úplnosti věnuje např. kniha [23].

9.1 Třídy složitosti P a NP

U výpočetně zvládnutelných problémů, jako je např. hledání nejkratší cesty v grafu, jsme věnovali pozornost variantám algoritmů, které namísto asymptotické složitosti $O(|U|^2)$ dokáží pracovat v čase $O(|H| \lg |U|)$. Nejen tento, ale ani mnohem výraznější rozdíly ve složitosti polynomiálně omezených algoritmů nejsou z pohledu výpočetně nezvládnutelných úloh podstatné. Z důvodů, které vyplynou později, prezentujeme následující příklady obtížných úloh většinou ve dvou variantách – optimalizační a rozhodovací.

- **Barvení grafu**

  Již při zavedení chromatického čísla $\chi(G)$ grafu v odst. 3.1 jsme konstatovali, že jeho určení pro zadaný graf G je velmi obtížné. Výjimku tvoří pouze grafy, jejichž chromatické

číslo je rovno jedné nebo dvěma – pro ně lze tuto skutečnost snadno zjistit ($\chi(G) = 1$ mají pouze diskrétní grafy, návod na zjištění druhé možnosti poskytuje důkaz věty 3.19).

Optimalizační problém barevnosti grafu spočívá v určení chromatického čísla $\chi(G)$ zadaného grafu G, tzn. minimálního počtu barev a odpovídajícího obarvení uzlů.

Rozhodovací problém barevnosti grafu požaduje pro zadaný graf G a kladné celé číslo k určit, zda je graf k-chromatický, tzn. zda lze obarvit s použitím nejvýše k barev.

- **Nezávislé podmnožiny uzlů**

  Mezi další obtížně zjistitelné charakteristiky grafu patří jeho nezávislost $\alpha(G)$, tj. mohutnost maximální nezávislé podmnožiny uzlů grafu G. Podobně jako barevnost grafu souvisí i nezávislost s určitými problémy rozvrhování.

  Optimalizační problém nezávislých podmnožin uzlů grafu spočívá v určení maximální mohutnosti nezávislé podmnožiny uzlů.

  Rozhodovací problém nezávislých podmnožin uzlů grafu požaduje pro zadaný graf G a kladné celé číslo k určit, zda v grafu existuje nezávislá podmnožina obsahující k uzlů.

- **Hamiltonovské cesty a kružnice**

  Hamiltonovskou cestou, resp. kružnicí grafu G nazýváme takovou cestu, resp. kružnici, která obsahuje všechny uzly grafu G. Na rozdíl od Eulerovských tahů pokrývajících množinu hran grafu (viz Eulerovy grafy v odst. 3.1), se Hamiltonovské cesty a kružnice hledají velmi těžko.

  Problém obchodního cestujícího v optimalizační podobě je zobecněním hledání Hamiltonovské kružnice. Pro zadaný prostý neorientovaný graf $G = \langle H, U\rangle$ s ohodnocením hran $w : H \mapsto \mathbf{R}^+$ se má nalézt Hamiltonovská kružnice s minimální w-délkou. Název úlohy je odvozen od klasického transportního problému: je dán určitý počet míst, a je třeba navštívit okružní jízdou, která projede každým místem právě jednou a bude co nejkratší.

  Rozhodovací problém obchodního cestujícího se odvodí obvyklým způsobem: pro zadaný graf G s ohodnocením hran w a přirozené číslo k se má určit, zda v grafu existuje Hamiltonovská kružnice w-délky nejvýše rovné k.

- **Rozvrhování úkolů s penalizací**

  Nechť je dáno n úkolů $J_1, J_2, \ldots, J_n$, které je třeba v nějakém pořadí postupně provést. Pro každý úkol J_k je stanovena doba jeho trvání t_k, nejpozdější žádoucí doba dokončení d_k (měřeno od zahájení prvního úkolu) a penalizace p_k v případě překročení této doby (všechny hodnoty t_k, d_k a p_k jsou celá čísla). **Rozvrhem** těchto úkolů rozumíme permutaci $\pi = \langle i_1, i_2, \ldots, i_n\rangle$, která stanoví, že jako k-tý v pořadí se bude provádět úkol J_{i_k}. Celková penalizace pro rozvrh π je dána vztahem

$$p(\pi) = \sum_{k\in D} p_{i_k}, \quad \text{kde } D = \{j \in \langle 1, n\rangle : t_{i_1} + t_{i_2} + \cdots + t_{i_j} > d_{i_j}\},$$

  je to tedy součet penalizací za ty úkoly, které nebyly ukončeny včas.

  Optimalizační úloha rozvrhování spočívá v nalezení minimalní možné penalizace pro danou množinu úkolů (a v určení odpovídajícího rozvrhu úkolů).

  Rozhodovací úloha rozvrhování požaduje pro danou množinu úkolů a nezáporné celé k určit, zda existuje rozvrh úkolů π tak, že $p(\pi) \leq k$.

- **Plnění krabic**

  Předpokládejme, že máme k dispozici neomezené množství krabic s jednotkovým objemem a n objektů s objemem $s_1, s_2, \ldots, s_n$, kde $0 \leq s_i \leq 1$.

  Optimalizační úloha plnění spočívá v určení minimálního počtu krabic, do nichž lze poskládat uvedené objekty (a nalezení odpovídajícího rozdělení objektů do jednotlivých krabic).

  Rozhodovací úloha plnění požaduje pro zadané přirozené k určit, zda se objekty mohou naskládat do k krabic.

- **Problém batohu**

  Předpokládejme, že máme batoh s kapacitou K (kladné celé číslo) a n předmětů o velikostech $s_1, s_2, \ldots, s_n$ a cenách $c_1, c_2, \ldots, c_n$ (všechna s_i a c_i jsou kladná celá čísla).

  Optimalizační problém batohu spočívá v určení nejvyšší celkové ceny předmětů, které je možné naskládat do batohu (a nalezení příslušného výběru předmětů).

  Rozhodovací problém batohu znamená pro dané k určit, zda lze do batohu srovnat nějakou skupinu předmětů v celkové ceně alespoň k.

- **Součet podmnožiny**

  Optimalizační součtová úloha znamená pro daná kladná celá čísla $C, s_1, s_2, \ldots, s_n$ určit podmnožinu $\{s_{i_1}, s_{i_2}, \ldots, s_{i_k}\}$ s největším součtem, který není větší než C .

  Rozhodovací součtová úloha požaduje určit, zda existuje podmnožina $\{s_{i_1}, s_{i_2}, \ldots, s_{i_k}\}$ se součtem rovným C.

- **Splnitelnost logických formulí**

  Literálem nazýváme libovolnou logickou proměnnou nebo její negaci, **klausulí** pak logický součet libovolného počtu literálů. Logický výraz v **normální konjuktivní formě** má tvar logického součinu libovolného počtu klausulí.

  Rozhodovací problém splnitelnosti logických formulí spočívá v určení, zda existuje takové přiřazení pravdivostních hodnot *true, false* proměnným ve výrazu v normální konjunktivní formě, kterým získá celý výraz hodnotu *true*.

Třída složitosti P

Definice 9.1: Abstraktní úlohou $Q \subseteq I \times S$ budeme rozumět binární relaci mezi množinou **instancí úlohy** I a množinou **řešení úlohy** S.

Jako příklad uveďme **problém MIN-CESTA** – určení nejkratší cesty mezi dvěma uzly v prostém neorientovaném grafu. Instanci problému MIN-CESTA představuje trojice hodnot: prostý graf $G = \langle H, U \rangle$ a dva jeho uzly. Řešením problému MIN-CESTA je pak každá posloupnost uzlů vyjadřující minimální cestu grafu (prázdná posloupnost může vyjadřovat neexistenci cesty).

Teorie NP-úplnosti se pro zjednodušení omezuje pouze na tzv. **rozhodovací úlohy** – to jsou takové, kde výsledkem je jedna z hodnot ano/ne. Za abstraktní rozhodovací úlohu se tedy považuje relace z množiny instancí I do množiny řešení $\{0, 1\}$. Rozhodovací variantu problému MIN-CESTA lze formulovat např. takto: Určete, zda pro zadaný neorientovaný graf $G = \langle H, U \rangle$, dvojici uzlů $u, v \in U$ a nezáporné celé číslo k existuje v G cesta z u do v, jejíž délka je nejvýše rovna k. Nelze samozřejmě tvrdit, že tato varianta je ekvivalentní s původní úlohou. Pokud je ale k dispozici algoritmus jejího řešení složitosti $O(f(n))$ (předpokládáme $n = |U|$), můžeme v tomto případě navrhnout algoritmus složitosti $O(nf(n))$ pro řešení původní úlohy – stačí postupně uvažovat hodnoty $k = 0, 1, \ldots, n-1$.

Je možné namítnout, že teorie NP-úplnosti se vyhýbá velmi důležitým a častým **optimalizačním úlohám**, při nichž je třeba nalézt minimální nebo maximální hodnotu nějakého parametru řešení. Tato námitka je zčásti oprávněná, ovšem řadu optimalizačních úloh lze přeformulovat do rozhodovacích variant takovým způsobem, že od řešení této varianty lze dospět i k řešení původní optimalizační úlohy. To sice není možné ve všech případech, ale i pak má studium odvozené rozhodovací úlohy svůj význam: je-li tato úloha obtížná, je možno s jistotou považovat za nejméně stejně obtížnou i odpovídající úlohu optimalizační.

Vraťme se však ještě k definici abstraktní úlohy – je prozatím příliš obecná na to, abychom na ní mohli založit studium složitosti řešení algoritmických úloh. Abychom mohli nějakou úlohu řešit pomocí stanoveného postupu, musí mít její instance standardně definovanou formu, se kterou budou algoritmy schopné pracovat. Tento požadavek můžeme splnit předpokladem existence kódování c všech instancí úlohy prostřednictvím řetězů binárních číslic 0 a 1, tedy $c : I \mapsto \{0,1\}^*$.

Konkrétní úlohou budeme nyní nazývat úlohu, jejíž instanční množina je tvořena binárními řetězy. Algoritmus řeší konkrétní úlohu v čase $O(f(n))$, pokud pro každou instanci úlohy (tj. binární řetěz) i délky $n = |i|$ spočítá algoritmus její řešení v čase nejvýše $O(f(n))$. Konkrétní úlohu tedy považujeme za **řešitelnou v polynomiálně omezeném čase**, pokud existuje algoritmus, který ji řeší v čase $O(n^k)$ pro nějakou konstantu k.

Definice 9.2: Třídou složitosti P nazýváme množinu všech rozhodovacích úloh řešitelných v polynomiálně omezeném čase.

Je možné se ptát, proč se právě polynomiálně omezené úlohy zvolily jako formalizované synonymum zvládnutelných úloh, když např. úloha s polynomiální složitostí $O(n^{100})$ bude vyžadovat (pro malá n) mnohem více operací než úloha složitosti $O(2^n)$. Důvodů je několik – na prvním místě lze konstatovat, že známé polynomiálně složité úlohy jsou charakterizovány rozumně nízkým stupněm omezujícího polynomu. Naproti tomu úlohy nepatřící mezi polynomiálně omezené jsou vesměs výpočetně nezvládnutelné.

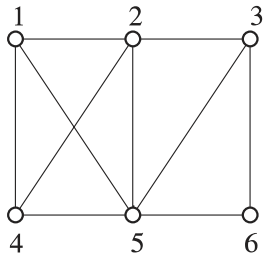
Druhým důvodem jsou uzávěrové vlastnosti polynomů – je to nejmenší třída funkcí uzavřená vůči sčítání, odčítání, násobení a skládání. To dovoluje vytvářet polynomiálně složité algoritmy pomocí kombinování dílčích polynomiálních algoritmů řešících podúlohy. Třetím důvodem je nezávislost třídy **P** na používaném formálním výpočetním modelu. Jednotlivé modely se mohou lišit přípustnými operacemi nebo paměťovými prostředky, takže asymptotické časové složitosti téže úlohy se pak mohou na různých modelech lišit – nebudou se však lišit v tom, zda patří nebo nikoliv do třídy **P** v obou modelech.

Velmi důležitým rysem při vymezení tříd složitosti je skutečnost, že doba výpočtu algoritmů se určuje v závislosti na délce (a nikoliv hodnotě!) vstupních dat. To znamená, že např. elementární algoritmus testování prvočíselnosti čísla n spočívající v nejvýše $(n-2)$-krát provedeném testu dělitelnosti n čísly $2, 3, \ldots, n-1$ nemá složitost lineární, ale exponenciální. Algoritmus sice proběhne v čase $O(n)$, ale počet k dvojkových číslic v zápisu hodnoty n je roven $\lfloor \lg n \rfloor + 1$, takže je $n = 2^k$.

Při řazení n-prvkové posloupnosti je délka vstupu nejméně n, takže je-li časová složitost algoritmu řazení dána polynomem v n, bude algoritmus polynomiálně omezený i vzhledem k přesné délce vstupu. Přesnější mírou délky vstupu je hodnota $n \lg Max$, kde Max je maximální absolutní hodnota prvku řazené posloupnosti. Každá z těchto měr se dá omezit polynomiální funkcí s argumentem rovným druhé míře, takže pro určení příslušnost této úlohy do třídy složitosti **P** nemusíme být zcela přesní ohledně délky vstupu. Pro algoritmy, jejichž výpočet závisí na hodnotě vstupních dat, si však tuto benevolenci nelze dovolit.

Třída složitosti NP

Kódování konkrétních úloh můžeme zobecnit z binárního na řetězy nad libovolnou konečnou abecedou – převod do konečné binární reprezentace je jen technickou záležitostí. Při vyjad-



Vstup:

4, 6, (1,2), (1,4), (1,5), (2,3), (2,4), (2,5), (3,5), (3,6), (4,5), (5,6)

počet uzlů
počet barev
výčet hran

Řetěz s	ABCDAB	ABCCDA	ABCD	BFLMPS
Výstup	ne	ano	ne (počet)	ne (syntaxe)

Obrázek 9.1: Nedeterministický algoritmus

řování číselných hodnot se pak předpokládá použití běžné polyadické notace, která pro zápis (přirozeného) čísla n potřebuje $O(\lg n)$ symbolů. Díky kódování se mohou všechny algoritmy řešení úloh chápat jako algoritmy rozhodující o příslušnosti zadaného vstupního řetězu k nějakému jazyku (přitom řetězy neodpovídající svým formátem smysluplnému zadání jsou rovněž zpracovány s negativním výsledkem).

U rozhodovacích úloh, které jsme uvedli dříve, se odpověď vždy opírá o existenci nějakého objektu splňujícího požadavky zadání – např. existenci jistého přiřazení hodnot logickým proměnným. Nazvěme tento objekt **předpokládaným řešením**, i tento objekt si snadno představíme standardním způsobem zakódovaný do tvaru řetězu symbolů.

Při definici třídy složitosti **NP** hraje základní roli pojem **nedeterministického algoritmu**. Nedeterminismus algoritmu dovoluje obejít obtížnou část problému – nalezení předpokládaného řešení – a deterministicky postupovat jen při jeho ověření. Nedeterministický algoritmus má totiž dvě fáze:

- **Nedeterministická fáze** – do paměti se při ní uloží nějaký řetěz znaků s, který můžeme považovat za (uhádnuté) předpokládané řešení. Obsah řetězu s se při různých výpočtech algoritmu může lišit.

- **Deterministická fáze** – znamená provedení běžného deterministického algoritmu, který využívá vedle vstupních dat úlohy rovněž řetězu s. Tuto fázi lze chápat jako ověření předpokládaného řešení s, které skončí rozhodnutím o výsledku (případně nekonečným cyklem).

Protože výsledek nedeterministického algoritmu může být pro stejná vstupní data x odlišný (závisí i na řetězu s), považujeme za kladný výsledek jeho aplikace na vstup x, pokud **existuje výpočet končící kladnou odpovědí** (jinými slovy existuje-li potřebné předpokládané řešení). Celkový počet kroků výpočtu nedeterministického algoritmu je roven součtu počtu kroků nedeterministické fáze (tj. délky řetězu s) a počtu kroků provedených při výpočtu detrministické fáze.

Uvažujme např. (rozhodovací) problém barevnosti, tj. určení, zda je graf $G = \langle H, U \rangle$ k-chromatický. V první fázi nedeterministického algoritmu se zaznamená nějaký řetěz $s = s_1 s_2 \dots s_n$, který interpretujeme jako zakódované předpokládané obarvení uzlů – každému uzlu u_i se jím přiděluje barva s_i. V determinstické fázi algoritmu se prohlídkou řetězu s ověří, že $n = |U|$ a že se opravdu použilo jen k různých barev, a probírkou všech hran grafu se zkontroluje, zda žádná nemá stejně obarvené oba své krajní uzly.

Nechť $k = 4$ a mějme graf G z obr. 9.1 jako vstup algoritmu. Potřebné čtyři barvy označíme pro jednoduchost velkými písmeny z počátku abecedy – tedy A,B,C a D. Na obrázku uvádíme rovněž několik různých tvarů řetězu s a jim odpovídající výsledky výpočtu deterministické fáze algoritmu. Protože existuje výpočet s výsledkem *ano*, je celkový výsledek algoritmu pro graf G *ano*.

Nedeterministický algoritmus nazveme **polynomiálně omezený**, pokud pro libovolná vstupní data délky n, pro která je výsledek *ano*, existuje výpočet algoritmu složitosti $O(n^k)$

pro nějakou konstantu k.

Definice 9.3: Třídou složitosti NP nazýváme množinu všech rozhodovacích úloh, pro které existuje polynomiálně omezený nedeterministický algoritmus jejich řešení.

Snadno nahlédneme, že všechny rozhodovací úlohy uvedené na začátku tohoto odstavce patří do třídy složitosti **NP**. Každý deterministický algoritmus lze považovat za speciální případ nedeterministického algoritmu s triviální první fází, takže platí

$$\mathbf{P} \subseteq \mathbf{NP}$$

Zdálo by se přirozené očekávat, že **P** je vlastní podmnožinou **NP**, dosud se to však nepodařilo dokázat. Vztah tříd **P** a **NP** bývá zařazován mezi nejdůležitější nevyřešené problémy moderní matematiky. Panuje sice téměř všeobecné přesvědčení, že **NP** je mnohem větší než **P**, ale pro žádnou úlohu z **NP** dosud nebylo dokázáno, že současně není v **P**. To platí i o všech dříve uvedených „obtížných" úlohách – zatím nebyla vyvrácena možnost existence polynomiálně složitého algoritmu pro řešení některé z nich. Jak vyplyne z vlastností NP-úplných úloh, nalezení takového algoritmu pro libovolnou z těchto úloh by znamenalo, že řešitelné v polynomiálně omezeném čase jsou i všechny zbývající, a tedy $\mathbf{P} = \mathbf{NP}$.

9.2 NP-úplné problémy

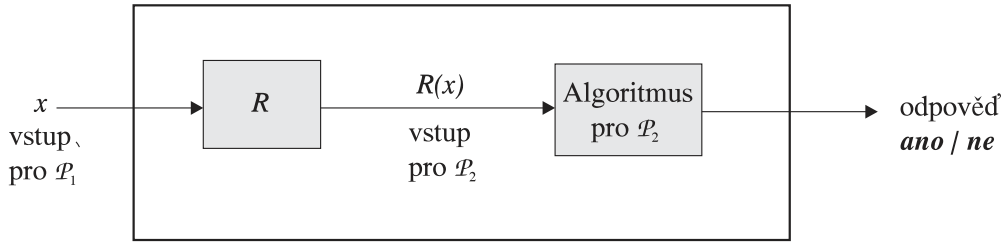
Ve třídě složitosti **NP** hrají klíčovou roli úlohy, které je možné charakterizovat jako v rámci této třídy nejobtížnější, nazývají se **NP-úplné problémy**. Jsou natolik obtížné, že každá jiná úloha třídy **NP** je na ně redukovatelná v polynomiálním čase. Jako první byla Cookem (viz [7]) dokázána NP-úplnost problému splnitelnosti logických formulí, později bylo přibylo několik stovek dalších NP-úplných problémů.

Základem NP-úplnosti je redukce, což je jen jiné využití principu matematické úspornosti: máme-li řešit nějakou úlohu, zkoušíme, zda není speciálním případem nějaké obecnější úlohy, jejíž algoritmus již známe. Předpokládejme tedy, že chceme řešit úlohu $\mathcal{P}_1$, přičemž už máme k dispozici algoritmus pro řešení úlohy $\mathcal{P}_2$. Nyní musíme navrhnout zobrazení R, které po zadání vstupu x pro úlohu $\mathcal{P}_1$ vytvoří vstup $R(x)$ pro úlohu $\mathcal{P}_2$ tak, že výsledek úlohy $\mathcal{P}_1$ pro vstup x je *ano*, právě když výsledek úlohy $\mathcal{P}_2$ pro vstup $R(x)$ je *ano*. Složením R a algoritmu pro úlohu $\mathcal{P}_2$ pak dostaneme algoritmus pro úlohu $\mathcal{P}_1$ (viz obr. 9.2).

Jako jednoduchou ilustraci redukce uvažujme úlohu $\mathcal{P}_1$: je dáno n logických proměnných, mají všechny hodnotu *true*? Nechť úloha $\mathcal{P}_2$ znamená: je dáno n celočíselných proměných, je jejich součet alespoň n? Transformační zobrazení R zvolíme takto:

$$R(x_1, x_2, \ldots, x_n) = y_1, y_2, \ldots, y_n, \quad \text{kde } y_i \;=\; \textbf{if } x_i \textbf{ then } 1 \textbf{ else } 0 \text{ pro } i = 1, 2, \ldots, n$$

Je vidět, že algoritmus řešící úlohu $\mathcal{P}_2$ pro vstup $y_1, y_2, \ldots, y_n$ poskytne výsledek úlohy $\mathcal{P}_1$ pro vstup $x_1, x_2, \ldots, x_n$.



Obrázek 9.2: Princip redukce úlohy

Definice 9.4: Nechť R je zobrazení množiny vstupů úlohy $\mathcal{P}_1$ do množiny vstupů úlohy $\mathcal{P}_2$. Zobrazení R nazýváme **polynomiální redukcí** (též **polynomiální transformací**) $\mathcal{P}_1$ na $\mathcal{P}_2$, pokud platí:

(1) R lze počítat v polynomiálně omezeném čase.

(2) Pro každý vstup x úlohy $\mathcal{P}_1$ je výsledek úlohy $\mathcal{P}_2$ na vstup $R(x)$ roven výsledku úlohy $\mathcal{P}_1$ na vstup x.

O úlohách $\mathcal{P}_1$ a $\mathcal{P}_2$, pro které existuje takové zobrazení R, říkáme, že $\mathcal{P}_1$ je **polynomiálně redukovatelná** na $\mathcal{P}_2$, zapisujeme $\mathcal{P}_1 \lhd \mathcal{P}_2$.

Věta 9.5: Polynomiální redukovatelnost je tranzitivní, neboli $\mathcal{P}_1 \lhd \mathcal{P}_2$ a $\mathcal{P}_2 \lhd \mathcal{P}_3$ implikuje $\mathcal{P}_1 \lhd \mathcal{P}_3$.

Důkaz: Označme R polynomiální redukci $\mathcal{P}_1$ na $\mathcal{P}_2$ a S polynomiální redukci $\mathcal{P}_2$ na $\mathcal{P}_3$. Pak je složené zobrazení $R \circ S$ polynomiální redukcí $\mathcal{P}_1$ na $\mathcal{P}_3$. △

Věta 9.6: Nechť je $\mathcal{P}_1 \lhd \mathcal{P}_2$. Potom platí:

- je-li $\mathcal{P}_2 \in \mathbf{P}$, pak je $\mathcal{P}_1 \in \mathbf{P}$
- je-li $\mathcal{P}_2 \in \mathbf{NP}$, pak je $\mathcal{P}_1 \in \mathbf{NP}$.

Důkaz: Provede se v obou případech s využitím faktu, že složením polynomiální redukce a polynomiálně omezeného (deterministického nebo nedeterministického) algoritmu dostaneme opět polynomiálně omezený algoritmus. △

Definice 9.7: Úlohu $\mathcal{P}$ patřící do třídy složitosti **NP** nazýváme **NP-úplnou**, pokud pro libovolnou úlohu $\mathcal{Q}$ ze třídy **NP** platí $\mathcal{Q} \lhd \mathcal{P}$.

Využitím první části tvrzení věty 9.6 dostáváme důležitý důsledek: pokud je jakýkoliv NP-úplný problém řešitelný v polynomiálně omezeném čase, pak platí $\mathbf{P} = \mathbf{NP}$. Následující základní tvrzení, jehož vyslovení na počátku 70. let otevřelo studium NP-úplných úloh, uvedeme bez důkazu.

Věta 9.8: (Cook) Problém splnitelnosti logických formulí je NP-úplný.

Věta 9.9: Nechť problém $\mathcal{P}$ patří do třídy složitosti **NP**. Potom je $\mathcal{P}$ NP-úplný právě tehdy, pokud existuje NP-úplný problém $\mathcal{Q}$ takový, že $\mathcal{Q} \lhd \mathcal{P}$.

Důkaz: 1. Je-li $\mathcal{P}$ NP-úplný, pak za $\mathcal{Q}$ lze vzít přímo $\mathcal{P}$.
2. Nechť platí druhá část tvrzení a mějme libovolnou úlohu $\mathcal{R}$ ze třídy **NP**. Potom je $\mathcal{R} \lhd \mathcal{Q}$ a podle věty 9.5 je také $\mathcal{R} \lhd \mathcal{P}$, takže $\mathcal{P}$ je NP-úplný. △

Díky předchozí větě lze důkaz NP-úplnosti nové úlohy provést formou nalezení známé NP-úplné úlohy, kterou lze na novou úlohu redukovat. Následující tvrzení opět uvedeme bez důkazu.

Věta 9.10: Problém barvení grafu, nezávislých podmnožin uzlů grafu, Hamiltonovských cest a kružnic, rozvrhování úkolů s penalizací, plnění krabic, problém batohu a součet podmnožiny jsou NP-úplné problémy.

Je zajímavé zjišťovat, do jaké míry je možné omezit vstupy výše uvedených problémů, aniž by se porušila jejich NP-úplnost. Znalost hranice, pod níž se daný problém dostává do třídy složitosti **P**, je velmi důležitá, neboť pak pro jeho řešení (na omezené množině vstupů) může existovat jednoduchý a velmi efektivní algoritmus.

Pro grafové úlohy je typickým omezením přechod k planárním grafům nebo omezení maximálního stupně uzlu. Problém barevnosti zůstane NP-úplný i tehdy, omezíme-li se na planární grafy a počet barev $k = 3$. NP-úplnost bude zachována dokonce i po dalším omezení na planární grafy se stupni uzlů nejvýše rovnými 4 (viz [23]). Podobně je možné omezit problém nezávislých podmnožin pouze na planární grafy a dokonce se stupni uzlů nejvýše rovnými 3

– i potom zůstává tato úloha NP-úplná. Problém určení existence Hamiltonovské cesty nebo kružnice zůstává NP-úplný i po omezení na planární kubické grafy (každý uzel má stupeň 3).

V případě splnitelnosti logických formulí je možné se omezit na výrazy, v nichž má každá klausule právě tři literály. Snížíme-li ovšem počet literálů na dva (v každé klausuli), jedná se o polynomiálně omezenou úlohu. Pro takový tvar výrazu je však NP-úplnou úlohou stanovit, zda existuje takové přiřazení hodnot proměnným, které zajistí hodnotu *true* u alespoň k klausulí.

Někdy způsobuje přechod od třídy **P** ke třídě **NP** jen velmi malá změna zadání úlohy. Tak např. hledání nejkratší cesty v grafu mezi dvěma uzly je snadné, takže i rozhodovací varianta (existuje cesta o délce nejvýše k) patří do **P**. Naproti tomu rozhodnout, zda mezi dvěma uzly existuje cesta o délce nejméně k, je NP-úplný problém. Je tedy velmi nesnadné obecně stanovit, v čem je podstata obtížnosti NP-úplných problémů. Vedle základní otázky, zda **P**=**NP**, existuje tedy v teorii NP-úplnosti ještě mnoho dalších zajímavých témat.

Cvičení

9.2-1. Dokažte, že neorientovaný bipartitní graf s lichým počtem uzlů neobsahuje Hamiltonovskou kružnici.

9.2-2. Navrhněte efektivní algoritmus pro úlohu nalezení orientované Hamiltonovské cesty v orientovaném acyklickém grafu, čímž se ověří, že tato úloha je polynomiálně omezená.

9.2-3. Ukažte, že problém určení splnitelnosti logických výrazů v disjunktivní normální formě je polynomiálně omezený.

9.2-4. Předpokládejme, že máme k dispozici algoritmus rozhodující splnitelnost logických formulí v konjunktivní normální formě. Popište, jak s jeho pomocí nalézt v polynomiálním čase konkrétní splňující přiřazení hodnot proměnným výrazu.

9.2-5. Dokažte, že problém splnitelnosti logických formulí, v nichž každá klausule obsahhuje přesně dva literály, je řešitelný v polynomiálně omezeném čase.
(*Návod:* Použijte toho, že $x \vee y$ je ekvivalentní s $\neg x \Rightarrow y$ a redukujte tento problém na grafovou úlohu, která polynomiálně omezená.)