Matematika III – 11. přednáška
Toky v sítích

Michal Bulant

Masarykova univerzita
Fakulta informatiky

27. 11. 2007

Obsah přednášky

Doporučené zdroje

- Martin Panák, Jan Slovák, **Drsná matematika**, *e*-text.
- *Předmětové záložky v IS MU*

Doporučené zdroje

- Martin Panák, Jan Slovák, **Drsná matematika**, *e*-text.
- *Předmětové záložky v IS MU*
- Jiří Matoušek, Jaroslav Nešetřil, **Kapitoly z diskrétní matematiky**, Univerzita Karlova v Praze, Karolinum, Praha, 2000, 377 s.
- Petr Hliněný, **Teorie grafů**, studijní materiály, `http://www.fi.muni.cz/~{}hlineny/Vyuka/GT/`

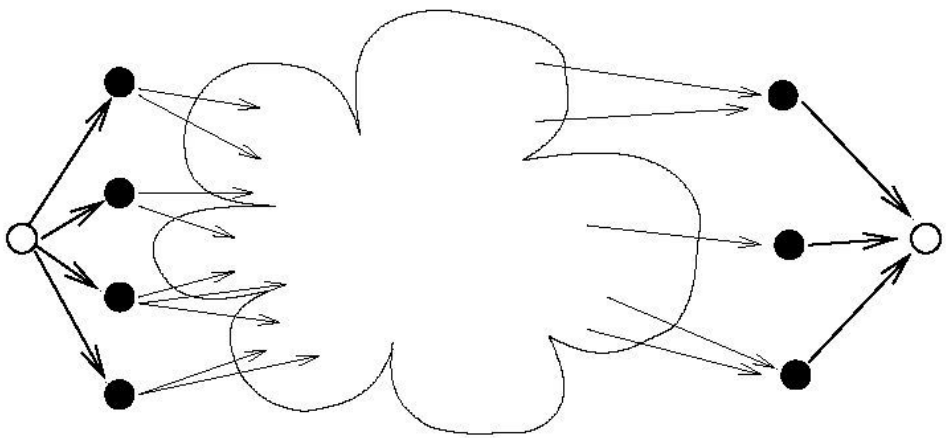
Doporučené zdroje

- Martin Panák, Jan Slovák, **Drsná matematika**, *e*-text.
- *Předmětové záložky v IS MU*
- Jiří Matoušek, Jaroslav Nešetřil, **Kapitoly z diskrétní matematiky**, Univerzita Karlova v Praze, Karolinum, Praha, 2000, 377 s.
- Petr Hliněný, **Teorie grafů**, studijní materiály, `http://www.fi.muni.cz/~{}hlineny/Vyuka/GT/`
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest a Clifford Stein , **Introduction to Algorithms**, MIT Press, 2001.

Plán přednášky

Další významná skupina aplikací jazyka teorie grafů se týká přesunu nějakého měřitelného materiálu v pevně zadané síti. Vrcholy v orientovaném grafu představují body, mezi kterými lze podél hran přenášet předem známá množství, která jsou zadána formou ohodnocení hran. Některé vybrané vrcholy představují **zdroj sítě**, jiné výstup ze sítě. Podle analogie potrubní sítě pro přenos kapaliny říkáme výstupním vrcholům **stok sítě**). Síť je tedy pro nás orientovaný graf s ohodnocenými hranami a vybranými vrcholy, kterým říkáme zdroje a stoky.

Je zřejmé, že se můžeme bez újmy na obecnosti omezit na orientované grafy s **jedním zdrojem** a **jedním stokem**.
V obecném případě totiž vždy můžeme přidat jeden stok a jeden zdroj navíc a spojit je vhodně orientovanými hranami s všemi zadanými zdroji a stoky tak, že ohodnocení přidaných hran bude zároveň zadávat maximální kapacity jednotlivých zdrojů a stoků.
Situace je naznačena na obrázku, kde černými vrcholy nalevo jsou zobrazeny všechny zadané zdroje, zatímco černé vrcholy napravo jsou všechny zadané stoky. Nalevo je jeden přidaný (virtuální) zdroj jako bílý vrchol a napravo jeden stok. Označení hran není v obrázku uvedeno.



Definice

Síť je orientovaný graf $G = (V, E)$ s vybraným jedním vrcholem z nazvaným **zdroj** a jiným vybraným vrcholem s nazvaným **stok**, spolu s nezáporným ohodnocením hran $w : E \to \mathbb{R}_0^+$, nazývaným **kapacita hran**.

Definice

Síť je orientovaný graf $G = (V, E)$ s vybraným jedním vrcholem z nazvaným **zdroj** a jiným vybraným vrcholem s nazvaným **stok**, spolu s nezáporným ohodnocením hran $w : E \to \mathbb{R}_0^+$, nazývaným **kapacita hran**. **Tokem** v síti $S = (V, E, z, s, w)$ rozumíme ohodnocení hran $f : E \to \mathbb{R}$ takové, že součet hodnot u vstupních hran u každého vrcholu v kromě zdroje a stoku je stejný jako součet u výstupních hran z téhož vrcholu, tj.

$$v \neq z, s \implies \sum_{e \in IN(v)} f(e) = \sum_{e \in OUT(v)} f(e)$$

a tok splňuje *kapacitní omezení* $f(e) \leq w(e)$.

Definice

Síť je orientovaný graf $G = (V, E)$ s vybraným jedním vrcholem z nazvaným **zdroj** a jiným vybraným vrcholem s nazvaným **stok**, spolu s nezáporným ohodnocením hran $w : E \to \mathbb{R}_0^+$, nazývaným **kapacita hran**. **Tokem** v síti $S = (V, E, z, s, w)$ rozumíme ohodnocení hran $f : E \to \mathbb{R}$ takové, že součet hodnot u vstupních hran u každého vrcholu v kromě zdroje a stoku je stejný jako součet u výstupních hran z téhož vrcholu, tj.

$$v \neq z, s \implies \sum_{e \in IN(v)} f(e) = \sum_{e \in OUT(v)} f(e)$$

a tok splňuje *kapacitní omezení* $f(e) \leq w(e)$. **Velikost toku** f je dána celkovou balancí hodnot u zdroje

$$|f| = \sum_{e \in OUT(z)} f(e) - \sum_{e \in IN(z)} f(e).$$

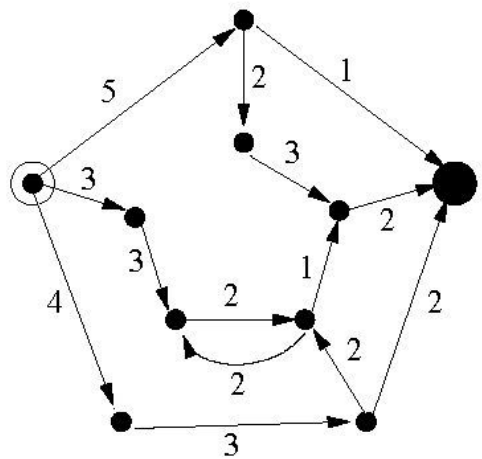
Z definice je zřejmé, že velikost toku můžeme stejně dobře vypočíst jako hodnotu

$$|f| = \sum_{e \in IN(s)} f(e) - \sum_{e \in OUT(s)} f(e).$$

Z definice je zřejmé, že velikost toku můžeme stejně dobře vypočíst jako hodnotu

$$|f| = \sum_{e \in IN(s)} f(e) - \sum_{e \in OUT(s)} f(e).$$

Na obrázku máme nakreslenu jednoduchou síť se zvýrazněným bílým zdrojem a černým stokem. Součtem maximálních kapacit hran vstupujících do stoku vidíme, že maximální možný tok v této síti je 5.



Plán přednášky

Naší úlohou bude pro zadanou síť na grafu G určit maximální možný tok. Jde vlastně o speciální případ úlohy lineárního (celočíselného) programování, kde neznámými jsou toky na hranách a omezení plynou z podmínek na tok. Ukáže se, že pro řešení této úlohy existují jednoduché a přitom rychlé algoritmy.

Naší úlohou bude pro zadanou síť na grafu G určit maximální možný tok. Jde vlastně o speciální případ úlohy lineárního (celočíselného) programování, kde neznámými jsou toky na hranách a omezení plynou z podmínek na tok. Ukáže se, že pro řešení této úlohy existují jednoduché a přitom rychlé algoritmy.

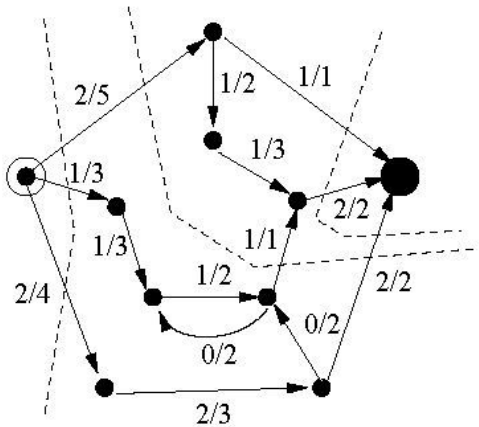
Definice

Řezem v síti $S = (V, E, z, s, w)$ rozumíme takovou množinu hran $C \subset E$, že po jejím odebrání nebude v grafu $G = (V, E \setminus C)$ žádná (orientovaná) cesta z z do s. Číslo

$$|C| = \sum_{e \in C} w(e)$$

nazýváme **velikost řezu** C.

Evidentně platí, že nikdy nemůžeme najít větší tok, než je hodnota kteréhokoliv z řezů. Na dalším obrázku máme zobrazen tok sítí s hodnotou 5 a čárkovanými lomenými čarami jsou naznačeny řezy o hodnotách 12, 8 a 5.



Poznámka

Tok a kapacitu hran v síti obvykle zapisujeme v obrázku ve tvaru f/c, kde f je hodnota toku na dané hraně a c její kapacita.

Sestavíme algoritmus, který pomocí postupných konstrukcí vhodných cest najde řez s minimální možnou hodnotou a zároveň najde tok, který tuto hodnotu realizuje. Tím dokážeme následující větu:

Věta

Maximální velikost toku v dané síti $S = (V, E, z, s, w)$ je rovna minimální velikosti řezu v této síti.

Myšlenka algoritmu – prohledáváme cesty mezi uzly grafu a snažíme se je „nasytit“ co největším tokem. Zavedeme si za tímto účelem terminologii.

Myšlenka algoritmu – prohledáváme cesty mezi uzly grafu a snažíme se je „nasytit“ co největším tokem. Zavedeme si za tímto účelem terminologii. O **neorientované** cestě v síti $S = (V, E, z, s, w)$ z vrcholu v do vrcholu w řekneme, že je **nenasycená**, jestliže pro všechny hrany této cesty orientované ve směru z v do w platí $f(e) < w(e)$ a $f(e) > 0$ pro hrany orientované opačně. Za **rezervu kapacity** hrany e pak označujeme číslo $w(e) - f(e)$ pro případ hrany orientované ve směru z v do w a číslo $f(e)$ při orientaci opačné. Pro zvolenou cestu bereme za rezervu kapacity minimální rezervu kapacity z jejích hran.

Ford-Fulkersonův algoritmus (1956)

Vstupem je síť $S = (V, E, z, s, w)$ a výstupem maximální možný tok $f : E \to \mathbb{R}$.

- *Inicializace:* zadáme $f(e) = 0$ pro všechny hrany $e \in E$ a najdeme množinu vrcholů $U \subset V$, do kterých existuje nenasycená cesta ze zdroje z.

Ford-Fulkersonův algoritmus (1956)

Vstupem je síť $S = (V, E, z, s, w)$ a výstupem maximální možný tok $f : E \to \mathbb{R}$.

- *Inicializace:* zadáme $f(e) = 0$ pro všechny hrany $e \in E$ a najdeme množinu vrcholů $U \subset V$, do kterých existuje nenasycená cesta ze zdroje z.
- *Hlavní cyklus:* Dokud $s \in U$ opakujeme

Ford-Fulkersonův algoritmus (1956)

Vstupem je síť $S = (V, E, z, s, w)$ a výstupem maximální možný tok $f : E \to \mathbb{R}$.

- *Inicializace:* zadáme $f(e) = 0$ pro všechny hrany $e \in E$ a najdeme množinu vrcholů $U \subset V$, do kterých existuje nenasycená cesta ze zdroje z.
- *Hlavní cyklus:* Dokud $s \in U$ opakujeme
  - zvolíme nenasycenou cestu P ze zdroje z do s a zvětšíme tok f u všech hran této cesty o její minimální rezervu

Ford-Fulkersonův algoritmus (1956)

Vstupem je síť $S = (V, E, z, s, w)$ a výstupem maximální možný tok $f : E \to \mathbb{R}$.

- *Inicializace:* zadáme $f(e) = 0$ pro všechny hrany $e \in E$ a najdeme množinu vrcholů $U \subset V$, do kterých existuje nenasycená cesta ze zdroje z.
- *Hlavní cyklus:* Dokud $s \in U$ opakujeme
  - zvolíme nenasycenou cestu P ze zdroje z do s a zvětšíme tok f u všech hran této cesty o její minimální rezervu
  - aktualizujeme U.

Ford-Fulkersonův algoritmus (1956)

Vstupem je síť $S = (V, E, z, s, w)$ a výstupem maximální možný tok $f : E \to \mathbb{R}$.

- *Inicializace:* zadáme $f(e) = 0$ pro všechny hrany $e \in E$ a najdeme množinu vrcholů $U \subset V$, do kterých existuje nenasycená cesta ze zdroje z.
- *Hlavní cyklus:* Dokud $s \in U$ opakujeme
  - zvolíme nenasycenou cestu P ze zdroje z do s a zvětšíme tok f u všech hran této cesty o její minimální rezervu
  - aktualizujeme U.
- na výstup dáme maximální tok f a minimální řez C tvořený všemi hranami vycházejícími z U a končícími v doplňku $V \setminus U$.

Důkaz správnosti algoritmu

Jak jsme viděli, velikost každého toku je nejvýše rovna hodnotě kteréhokoliv řezu. Stačí nám tedy ukázat, že v okamžiku zastavení algoritmu jsme vygenerovali řez i tok se stejnou hodnotou. Algoritmus se zastaví, jakmile neexistuje nenasycená cesta ze zdroje z do stoku s. To znamená, že U neobsahuje s a pro všechny hrany e z U do zbytku je $f(e) = w(e)$, jinak bychom museli koncový vrchol e přidat k U.

Důkaz správnosti algoritmu

Jak jsme viděli, velikost každého toku je nejvýše rovna hodnotě kteréhokoliv řezu. Stačí nám tedy ukázat, že v okamžiku zastavení algoritmu jsme vygenerovali řez i tok se stejnou hodnotou.
Algoritmus se zastaví, jakmile neexistuje nenasycená cesta ze zdroje z do stoku s. To znamená, že U neobsahuje s a pro všechny hrany e z U do zbytku je $f(e) = w(e)$, jinak bychom museli koncový vrchol e přidat k U.
Zároveň ze stejného důvodu všechny hrany e, které začínají v komplementu $V \setminus U$ a končí v U musí mít tok $f(e) = 0$.

Pro velikost toku celé sítě jistě platí

$$|f| = \sum_{\text{hrany z } U \text{ do } V \setminus U} f(e) - \sum_{\text{hrany z } V \setminus U \text{ do } U} f(e).$$

Tento výraz je ovšem v okamžiku zastavení roven

$$\sum_{\text{hrany z } U \text{ do } V \setminus U} f(e) = \sum_{\text{hrany z } U \text{ do } V \setminus U} w(e) = |C|,$$

což jsme chtěli dokázat.

Pro velikost toku celé sítě jistě platí

$$|f| = \sum_{\text{hrany z } U \text{ do } V \setminus U} f(e) - \sum_{\text{hrany z } V \setminus U \text{ do } U} f(e).$$

Tento výraz je ovšem v okamžiku zastavení roven

$$\sum_{\text{hrany z } U \text{ do } V \setminus U} f(e) = \sum_{\text{hrany z } U \text{ do } V \setminus U} w(e) = |C|,$$

což jsme chtěli dokázat.
Zbývá ovšem ukázat, že algoritmus skutečně zastaví.

Zastavení Ford-Fulkersonova algoritmu

Tvrzení

Pro celočíselné kapacity hran sítě uvedený algoritmus vždy skončí. V obecném případě nejen, že algoritmus skončit nemusí, příslušné toky dokonce ani nemusí k maximálnímu toku konvergovat.

Zastavení Ford-Fulkersonova algoritmu

Tvrzení

Pro celočíselné kapacity hran sítě uvedený algoritmus vždy skončí. V obecném případě nejen, že algoritmus skončit nemusí, příslušné toky dokonce ani nemusí k maximálnímu toku konvergovat.

Důkaz.

Důkaz ukončení v celočíselném případě vyplývá z toho, že vždy sytíme cestu o celočíselné hodnotě. □

Zastavení Ford-Fulkersonova algoritmu

Tvrzení

Pro celočíselné kapacity hran sítě uvedený algoritmus vždy skončí. V obecném případě nejen, že algoritmus skončit nemusí, příslušné toky dokonce ani nemusí k maximálnímu toku konvergovat.

Důkaz.

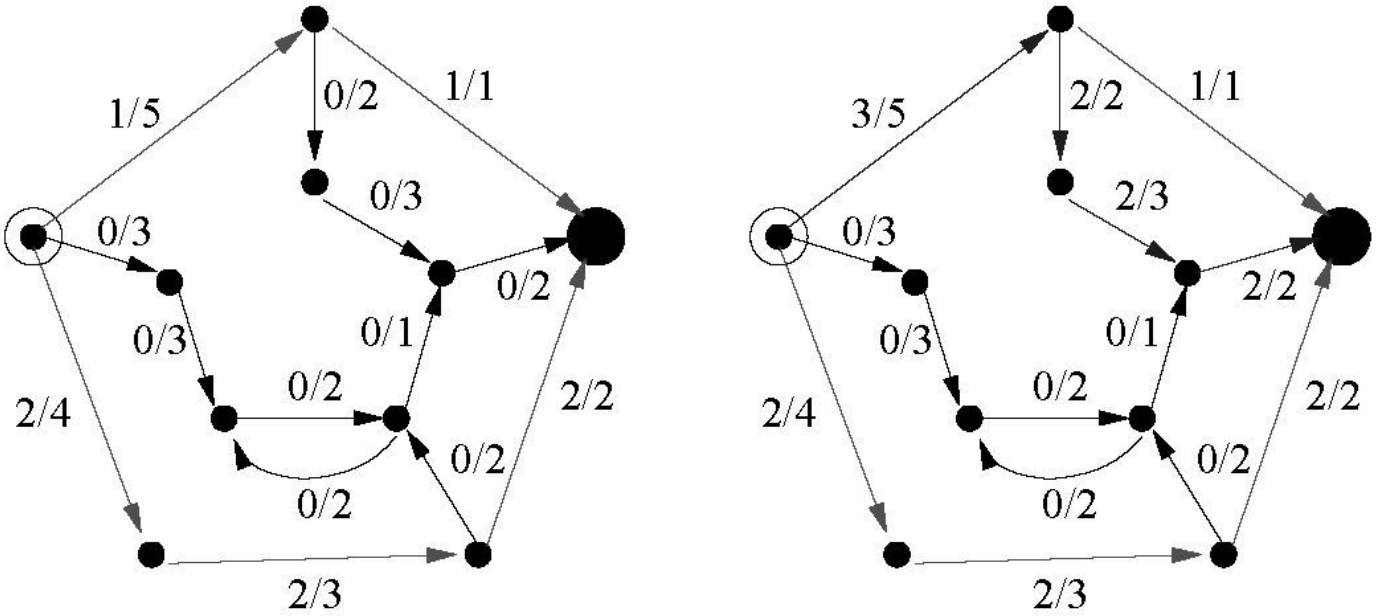
Důkaz ukončení v celočíselném případě vyplývá z toho, že vždy sytíme cestu o celočíselné hodnotě. □

Poznámka

Ford-Fulkersonův algoritmus má složitost v nejhorším případě $O(E \cdot |f|)$, kde $|f|$ je hodnota maximálního toku.

Edmonds-Karpův algoritmus

Vylepšením základního Ford-Fulkersonova algoritmu je *Edmonds-Karpův algoritmus*, ve kterém zvětšujeme tok podél *nejkratší* nenasycené cesty – tj. síť prohledáváme **do šířky**.
U tohoto algoritmu již je zaručeno zastavení, navíc je jeho časová složitost ohraničena $O(VE^2)$, nezávisle na hodnotě maximálního toku.



Chod algoritmu je ilustrován na obrázku. Vlevo jsou vybarveny dvě nejkratší nenasycené cesty ze zdroje do stoku (horní má dvě hrany, spodní tři). Jsou vyznačeny červeně. Napravo je pak nasycena další cesta v pořadí a je vyznačena modře. Je nyní zjevné, že nemůže existovat další nenasycená cesta ze zdroje do stoku. Proto algoritmus v tomto okamžiku skončí.

Zobecnění sítí

V praktických aplikacích mohou být na sítě kladeny další požadavky:

- maximální kapacita vrcholů – snadno se převede na základní případ *zdvojením* vrcholů, které spojíme hranou o dané kapacitě;
- minimální kapacita hran – např. aby nedocházelo k zanášení potrubí. V tomto případě lze modifikovat inicializaci, je ale třeba otestovat existenci přípustného toku (podobně jako v úloze lineárního progamování).

Plán přednášky

Věta (Mengerova)

Pro každé dva vrcholy v a w v grafu $G = (V, E)$ je počet hranově různých cest z v do w roven minimálnímu počtu hran, které je třeba odstranit, aby se v a w ocitly v různých komponentách vzniklého grafu.

Důkaz.

Plyne snadno z věty o maximálním toku a minimálním řezu v síti, kde jsou kapacity všech hran (v obou směrech) rovny 1. □

Bipartitní párování

Hezkým využitím toků v sítí je řešení úlohy bipartitního párování. Úkolem je najít v bipartitním grafu maximální podmnožinu hran takovou, aby žádné dvě hrany nesdílely vrchol.

Bipartitní párování

Hezkým využitím toků v sítí je řešení úlohy bipartitního párování. Úkolem je najít v bipartitním grafu maximální podmnožinu hran takovou, aby žádné dvě hrany nesdílely vrchol.
Jde o abstraktní variantu docela obvyklé úlohy – třeba spárování kluků a holek k tanci v tanečních, kdybychom měli předem známé možnosti, ze kterých vybíráme (např. aby dvojici netvořil pár příliš výškově nesourodý).

Bipartitní párování

Hezkým využitím toků v sítí je řešení úlohy bipartitního párování. Úkolem je najít v bipartitním grafu maximální podmnožinu hran takovou, aby žádné dvě hrany nesdílely vrchol.
Jde o abstraktní variantu docela obvyklé úlohy – třeba spárování kluků a holek k tanci v tanečních, kdybychom měli předem známé možnosti, ze kterých vybíráme (např. aby dvojici netvořil pár příliš výškově nesourodý).
Tento problém docela snadno převedeme na hledání maximálního toku. Přidáme si uměle navíc ke grafu zdroj, který propojíme hranami jdoucími do všech vrcholů v jedné skupině v bipartitním grafu, zatímco ze všech vrcholů ve druhé skupině vedeme hranu do přidaného stoku. Všechny hrany opatříme maximální kapacitou 1 a hledáme maximální tok. Za páry pak bereme hrany s nenulovým (tj. zřejmě jednotkovým) tokem.

Maďarský algoritmus (König, Egerváry, Kuhn)

Označujme kvůli jednoduchosti popisu vrcholy jedné skupiny v bipartitním grafu jako **červené**, vrcholy druhé skupiny jako **modré**. Budeme předpokládat, že vrcholy jsou obarveny tak, že počet červených vrcholů nepřevyšuje počet modrých.

Maďarský algoritmus (König, Egerváry, Kuhn)

Označujme kvůli jednoduchosti popisu vrcholy jedné skupiny v bipartitním grafu jako **červené**, vrcholy druhé skupiny jako **modré**. Budeme předpokládat, že vrcholy jsou obarveny tak, že počet červených vrcholů nepřevyšuje počet modrých.

Definice

Nechť je dán bipartitní graf $G = (V, E)$ a párování $M \subseteq E$. *M-alternující cestou v G* nazveme takovou cestu v G, jejíž hrany tvoří střídavě hrany z M a z $E \setminus M$.

Maďarský algoritmus (König, Egerváry, Kuhn)

Označujme kvůli jednoduchosti popisu vrcholy jedné skupiny v bipartitním grafu jako **červené**, vrcholy druhé skupiny jako **modré**. Budeme předpokládat, že vrcholy jsou obarveny tak, že počet červených vrcholů nepřevyšuje počet modrých.

Definice

Nechť je dán bipartitní graf $G = (V, E)$ a párování $M \subseteq E$. *M-alternující cestou v G* nazveme takovou cestu v G, jejíž hrany tvoří střídavě hrany z M a z $E \setminus M$. *M-rozšiřující cestou* v G nazveme M-alternující cestu, která spojuje dosud nepřiřazený červený vrchol u s dosud nepřiřazeným modrým vrcholem v.

Algoritmus pro nalezení maximálního párování

1. Nalezneme libovolné párování M. Označíme všechny červené vrcholy jako přípustné.
2. Vyberme některý dosud nepřiřazený přípustný červený vrchol v (pokud neexistuje, jsme hotovi) a nalezněme pro něj (např. prostřednictvím prohledávání do hloubky) M-alternující strom s kořenem ve v (pokud neexistuje, označme v jako nepřípustný a proces opakujme s jiným vrcholem). Pokud strom obsahuje nějakou M-rozšiřující cestu, odstraníme M-hrany v této cestě z M a ostatní hrany této cesty do M přidáme. Označmě všechny červené vrcholy jako přípustné a proces opakujme.

Datové struktury

Obvykle (Kořenové pěstěné) binární stromy nesoucí informaci v uzlech.

- AVL stromy (vyvážené), podobně *red-black* stromy

Datové struktury

Obvykle (Kořenové pěstěné) binární stromy nesoucí informaci v uzlech.

- AVL stromy (vyvážené), podobně *red-black* stromy
- B-stromy (2-3 stromy, B* stromy)

Datové struktury

Obvykle (Kořenové pěstěné) binární stromy nesoucí informaci v uzlech.

- AVL stromy (vyvážené), podobně *red-black* stromy
- B-stromy (2-3 stromy, B* stromy)
- binární halda, Fibonacciho halda

Datové struktury

Obvykle (Kořenové pěstěné) binární stromy nesoucí informaci v uzlech.

- AVL stromy (vyvážené), podobně *red-black* stromy
- B-stromy (2-3 stromy, B* stromy)
- binární halda, Fibonacciho halda
- a mnoho dalších

Datové struktury

Obvykle (Kořenové pěstěné) binární stromy nesoucí informaci v uzlech.

- AVL stromy (vyvážené), podobně *red-black* stromy
- B-stromy (2-3 stromy, B* stromy)
- binární halda, Fibonacciho halda
- a mnoho dalších

Datové struktury

Obvykle (Kořenové pěstěné) binární stromy nesoucí informaci v uzlech.

- AVL stromy (vyvážené), podobně *red-black* stromy
- B-stromy (2-3 stromy, B* stromy)
- binární halda, Fibonacciho halda
- a mnoho dalších

V této oblasti odkážeme na předmět *Návrh algoritmů* a další, my si pouze ukážeme využití binárních stromů v kódování (Huffmanův algoritmus).

Pracujeme s pěstěnými binárními stromy, kde máme navíc každou hranu *obarvenou* některým symbolem z dané výstupní abecedy A (často $A = \{0, 1\}$). Kódovými slovy C jsou slova nad abecedou A, na která převádíme symboly vstupní abecedy. Našim úkolem je reprezentovat daný text pomocí vhodných kódových slov nad výstupní abecedou.

Je snadno vidět, že je užitečné chtít, aby seznam kódových slov byl *bezprefixový* (v opačném případě může nastat problém s dekódováním).

Pracujeme s pěstěnými binárními stromy, kde máme navíc každou hranu *obarvenou* některým symbolem z dané výstupní abecedy A (často $A = \{0, 1\}$). Kódovými slovy C jsou slova nad abecedou A, na která převádíme symboly vstupní abecedy. Našim úkolem je reprezentovat daný text pomocí vhodných kódových slov nad výstupní abecedou.
Je snadno vidět, že je užitečné chtít, aby seznam kódových slov byl *bezprefixový* (v opačném případě může nastat problém s dekódováním).

Příklad

Text: `MADAM, I'M ADAM`
$C = \{0, 1, 00, 01, 10, 11, 000\}$ Pokud symboly přiřazujeme tak, jak přicházejí na řadu, dostaneme
$M = 0, A = 1, D = 00, , = 01, I = 10, ' = 11, . = 000$. Přitom ale např. není jasné dekódování řetězce 01001 – je to MADA, MI, nebo ,DA?

Prefixový kód C splňuje, že žádný prvek C není prefixem jiného slova z C.
Ke konstrukci binárních prefixových kódů (tj. nad abecedou $A = \{0, 1\}$) využijeme binárních stromů. Označíme-li hrany vycházející z každého uzlu 0, resp. 1, a označíme-li navíc listy stromu symboly vstupní abecedy, dostaneme prefixový kód nad A pro tyto symboly zřetězením označení hran na cestě z kořene do příslušného listu.
Takto vytvořený kód je zřejmě *prefixový*.

Prefixový kód C splňuje, že žádný prvek C není prefixem jiného slova z C.
Ke konstrukci binárních prefixových kódů (tj. nad abecedou $A = \{0, 1\}$) využijeme binárních stromů. Označíme-li hrany vycházející z každého uzlu 0, resp. 1, a označíme-li navíc listy stromu symboly vstupní abecedy, dostaneme prefixový kód nad A pro tyto symboly zřetězením označení hran na cestě z kořene do příslušného listu.
Takto vytvořený kód je zřejmě *prefixový*.
Uděláme-li tuto konstrukci navíc tak, abychom odrazili četnost symbolů vstupní abecedy v kódovaném textu, dosáhneme tak dokonce *bezztrátové komprese dat*.

Huffmanův algoritmus

Nechť M je seznam četností symbolů vstupní abecedy v textu. Algoritmus postupně zkonstruuje optimální binární strom (tzv. *minimum-weight binary tree*) a přiřazení symbolů listům.

- Vyber dvě nejmenší četnosti w_1, w_2 z M. Vyrob strom se dvěma listy označenými příslušnými symboly a kořenem označeným $w_1 + w_2$, odeber z M hodnoty w_1, w_2 a nahraď je hodnotou $w_1 + w_2$. Tento krok opakuj; pouze v případě, že vybraná hodnota z M je součtem, pak nevyráběj nový list, ale „připoj“ příslušný již existující podstrom.