Matematika III – 12. přednáška
Minimální kostra, toky v sítích, párování

Michal Bulant

Masarykova univerzita
Fakulta informatiky

5. 12. 2012

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.
  H. S. Wilf, **Generatingfunctionology**, Academic Press, druhé vydání, 1994 , (rovněž `http://www.math.upenn.edu/~wilf/DownldGF.html`)

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.
  H. S. Wilf, **Generatingfunctionology**, Academic Press, druhé vydání, 1994 , (rovněž `http://www.math.upenn.edu/~wilf/DownldGF.html`)
- R. Graham, D. Knuth, O. Patashnik, **Concrete Mathematics**, druhé vydání, Addison-Wesley, 1994.

Plán přednášky

Algoritmus pro nalezení kostry 1

Definice

Libovolný strom $T = (V, E')$ v grafu $G = (V, E)$, $E' \subseteq E$ se nazývá **kostra** (*spanning tree*) grafu G (tj. faktor grafu, který neobsahuje kružnice).

Algoritmus pro nalezení kostry 1

Definice

Libovolný strom $T = (V, E')$ v grafu $G = (V, E)$, $E' \subseteq E$ se nazývá **kostra** (*spanning tree*) grafu G (tj. faktor grafu, který neobsahuje kružnice).

Seřadíme zcela libovolně všechny hrany $e_1, \ldots, e_m$ v E do pořadí a postupně budujeme množiny hran E_i $(i = 0, \ldots, m)$ tak, že $E_0 = \emptyset$ a v t-tém kroku přidáme hranu e_i k E_{i-1} (tj. $E_i = E_{i-1} \cup \{e_i\}$), jestliže tím nevznikne v grafu $G_i = (V, E_i)$ kružnice, a ponecháme $E_i = E_{i-1}$ beze změny v případě opačném. Algoritmus skončí pokud buď má již graf G_i pro nějaké i právě $n - 1$ hran nebo je již $i = m$. Pokud zastavujeme z druhého důvodu, byl původní graf nesouvislý a kostra neexistuje.

Věta

Výsledkem předchozího algoritmu je vždy les T. Jestliže algoritmus skončí s $k \leq n-1$ hranami, má původní graf $n-k$ komponent. Zejména je tedy T kostrou, právě když algoritmus skončí pro dosažení $n-1$ hran.

Věta

Výsledkem předchozího algoritmu je vždy les T. Jestliže algoritmus skončí s $k \leq n-1$ hranami, má původní graf $n-k$ komponent. Zejména je tedy T kostrou, právě když algoritmus skončí pro dosažení $n-1$ hran.

Důkaz.

Tvrzení, že výsledný graf T je lesem, je zřejmé z postupu konstrukce. Je-li $k = n-1$, je navíc T strom podle charakterizační věty o stromech. Je-li $k < n-1$, je T lesem, s $n-k$ stromovými komponentami, neboť každá další komponenta přispívá jedničkou k hodnotě $(n-1)-k$ (rozdíl počtu hran ve stromu na n vrcholech a počtu hran v grafu T). □

Poznámka (složitost algoritmu)

Kružnice přidáním nové hrany vznikne tehdy a jen tehdy, jestli její koncové vrcholy leží ve stejné souvislé komponentě budovaného lesu T. Stačí nám proto průběžně udržovat znalost souvislých komponent.

Poznámka (složitost algoritmu)

Kružnice přidáním nové hrany vznikne tehdy a jen tehdy, jestli její koncové vrcholy leží ve stejné souvislé komponentě budovaného lesu T. Stačí nám proto průběžně udržovat znalost souvislých komponent.

V abstraktní podobě nám stačí umět pro již zadané třídy ekvivalence na dané množině (v našem případě jsou to vrcholy) slučovat dvě třídy ekvivalence do jedné a nalézat pro daný prvek, do které třídy patří. Pro sjednocení jistě potřebujeme $O(k)$ času, kde k je počet prvků slučovaných tříd a jistě můžeme použít ohraničení počtu k celkovým počtem vrcholů n. Se třídami si můžeme pamatovat i počty jejich prvků a průběžně pro každý vrchol uchovávat informaci do které třídy patří. Sjednocení dvou tříd tedy představuje přeznačení jména u všech prvků jedné z nich. Máme tedy $n - 1$ operací sjednocení a m operací testování ekvivalence vrcholů, proto lze složitost ohraničit $O(n^2 + m)$.

Poznámka (složitost algoritmu)

Kružnice přidáním nové hrany vznikne tehdy a jen tehdy, jestli její koncové vrcholy leží ve stejné souvislé komponentě budovaného lesu T. Stačí nám proto průběžně udržovat znalost souvislých komponent.

V abstraktní podobě nám stačí umět pro již zadané třídy ekvivalence na dané množině (v našem případě jsou to vrcholy) slučovat dvě třídy ekvivalence do jedné a nalézat pro daný prvek, do které třídy patří. Pro sjednocení jistě potřebujeme $O(k)$ času, kde k je počet prvků slučovaných tříd a jistě můžeme použít ohraničení počtu k celkovým počtem vrcholů n. Se třídami si můžeme pamatovat i počty jejich prvků a průběžně pro každý vrchol uchovávat informaci do které třídy patří. Sjednocení dvou tříd tedy představuje přeznačení jména u všech prvků jedné z nich. Máme tedy $n-1$ operací sjednocení a m operací testování ekvivalence vrcholů, proto lze složitost ohraničit $O(n^2+m)$. Budeme-li vždy přeznačovat menší ze slučovaných tříd, pak celkový počet operací v našem algoritmu lze ohraničit $O(n \log n + m)$.

Algoritmus pro nalezení kostry 2

Jiný postup: Budeme v grafu $G = (V, E)$ s n vrcholy a m hranami postupně budovat strom T. Začneme v libovolně zvoleném vrcholu v a prázdnou množinou hran, tj. $T_0 = (\{v\}, \emptyset)$. V i–tém kroku hledáme mezi hranami, které dosud nejsou v T_{i-1}, mají v T_{i-1} jeden koncový vrchol, ale druhý koncový vrchol do T_{i-1} nepatří. Některou takovou hranu přidáme i s druhým koncovým vrcholem a získáme tak T_i. Algoritmus skončí, až taková hrana neexistuje.

Algoritmus pro nalezení kostry 2

Jiný postup: Budeme v grafu $G = (V, E)$ s n vrcholy a m hranami postupně budovat strom T. Začneme v libovolně zvoleném vrcholu v a prázdnou množinou hran, tj. $T_0 = (\{v\}, \emptyset)$. V i–tém kroku hledáme mezi hranami, které dosud nejsou v T_{i-1}, mají v T_{i-1} jeden koncový vrchol, ale druhý koncový vrchol do T_{i-1} nepatří. Některou takovou hranu přidáme i s druhým koncovým vrcholem a získáme tak T_i. Algoritmus skončí, až taková hrana neexistuje.

Evidentně je výsledný graf T (v případě, že má n vrcholů) souvislý a podle počtu vrcholů a hran je to strom. Vrcholy T splývají s vrcholy souvislé komponenty G.

Algoritmus pro nalezení kostry 2

Jiný postup: Budeme v grafu $G = (V, E)$ s n vrcholy a m hranami postupně budovat strom T. Začneme v libovolně zvoleném vrcholu v a prázdnou množinou hran, tj. $T_0 = (\{v\}, \emptyset)$. V i–tém kroku hledáme mezi hranami, které dosud nejsou v T_{i-1}, mají v T_{i-1} jeden koncový vrchol, ale druhý koncový vrchol do T_{i-1} nepatří. Některou takovou hranu přidáme i s druhým koncovým vrcholem a získáme tak T_i. Algoritmus skončí, až taková hrana neexistuje.

Evidentně je výsledný graf T (v případě, že má n vrcholů) souvislý a podle počtu vrcholů a hran je to strom. Vrcholy T splývají s vrcholy souvislé komponenty G.

Algoritmus v čase $O(n + m)$ nalezne kostru souvislé komponenty zvoleného počátečního vrcholu v.

Plán přednášky

Kromě nalezení kostry je často účelné znát nejlepší možnou kostru vzhledem k nějakému ohodnocení hran. Protože je to obecnou vlastností stromů, každá kostra grafu G má stejný počet hran. V grafech s ohodnocenými hranami, budeme hledat kostry s minimálním součtem ohodnocení použitých hran.

Definice

Nechť $G = (V, E, w)$ je souvislý graf s ohodnocenými hranami s nezápornými vahami $w(e)$ pro všechny hrany. Jeho **minimální kostra** (*minimum spanning tree*) T je taková kostra grafu G, která má mezi všemi jeho kostrami minimální součet ohodnocení všech hran.

Kromě nalezení kostry je často účelné znát nejlepší možnou kostru vzhledem k nějakému ohodnocení hran. Protože je to obecnou vlastností stromů, každá kostra grafu G má stejný počet hran. V grafech s ohodnocenými hranami, budeme hledat kostry s minimálním součtem ohodnocení použitých hran.

Definice

Nechť $G = (V, E, w)$ je souvislý graf s ohodnocenými hranami s nezápornými vahami $w(e)$ pro všechny hrany. Jeho **minimální kostra** (*minimum spanning tree*) T je taková kostra grafu G, která má mezi všemi jeho kostrami minimální součet ohodnocení všech hran.

O praktičnosti takové úlohy můžete přemýšlet třeba v souvislosti s rozvodnými sítěmi elektřiny, plynu, vody apod (viz např. problém elektrifikace části jižní Moravy, který vyřešil Otakar Borůvka v roce 1926 pomocí algoritmu – v dnešní terminologii – minimální kostry, přestože obor algoritmické teorie grafů čekal ještě cca 10 let na svůj oficiální vznik).

Kruskalův algoritmus

Předpokládejme, že jsou všechna ohodnocení $w(e)$ hran v grafu G nezáporná. Následujícímu postupu se říká **Kruskalův algoritmus**:

- Setřídíme všech m hran v E tak, aby $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_m)$.
- v tomto pořadí aplikujeme na hrany postup z Algoritmu 1 pro kostru.

Kruskalův algoritmus

Předpokládejme, že jsou všechna ohodnocení $w(e)$ hran v grafu G nezáporná. Následujícímu postupu se říká **Kruskalův algoritmus**:

- Setřídíme všech m hran v E tak, aby $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_m)$.
- v tomto pořadí aplikujeme na hrany postup z Algoritmu 1 pro kostru.

Jde o typický příklad takzvaného „hladového (greedy) přístupu“, kdy se k maximalizaci zisku (nebo minimalizaci nákladů) snažíme dostat výběrem momentálně nejvýhodnějšího kroku. Často tento přístup zklame, protože nízké náklady na začátku procesu mohou zavinit vysoké na jeho konci.
V tomto případě (naštěstí) hladový přístup funguje, nemusíme tedy prohledávat a porovnávat až n^{n-2} koster na daném grafu.

Věta

Kruskalův algoritmus správně řeší problém minimální kostry pro každý souvislý graf G s nezáporným ohodnocením hran. Algoritmus pracuje v čase $O(m \log m)$, kde m je počet hran v G.

Věta

Kruskalův algoritmus správně řeší problém minimální kostry pro každý souvislý graf G s nezáporným ohodnocením hran. Algoritmus pracuje v čase $O(m \log m)$, kde m je počet hran v G.

Důkaz.

Buď T výsledná kostra z algoritmu, T_0 taková minimální kostra na G, která se s T shoduje na co nejvíce (seřazených) hranách. Sporem dokážeme, že $T_0 = T$.

Věta

Kruskalův algoritmus správně řeší problém minimální kostry pro každý souvislý graf G s nezáporným ohodnocením hran. Algoritmus pracuje v čase $O(m \log m)$, kde m je počet hran v G.

Důkaz.

Buď T výsledná kostra z algoritmu, T_0 taková minimální kostra na G, která se s T shoduje na co nejvíce (seřazených) hranách.
Sporem dokážeme, že $T_0 = T$.
Předpokládejme $T_0 \neq T$ a buď j nejmenší index, takový, že se T a T_0 liší v hraně e_j (zřejmě $e_j \in T \setminus T_0$).

Věta

Kruskalův algoritmus správně řeší problém minimální kostry pro každý souvislý graf G s nezáporným ohodnocením hran. Algoritmus pracuje v čase $O(m \log m)$, kde m je počet hran v G.

Důkaz.

Buď T výsledná kostra z algoritmu, T_0 taková minimální kostra na G, která se s T shoduje na co nejvíce (seřazených) hranách.
Sporem dokážeme, že $T_0 = T$.
Předpokládejme $T_0 \neq T$ a buď j nejmenší index, takový, že se T a T_0 liší v hraně e_j (zřejmě $e_j \in T \setminus T_0$). Pak $T_0 \cup \{e_j\}$ obsahuje právě jednu kružnici C.

Věta

Kruskalův algoritmus správně řeší problém minimální kostry pro každý souvislý graf G s nezáporným ohodnocením hran. Algoritmus pracuje v čase $O(m \log m)$, kde m je počet hran v G.

Důkaz.

Buď T výsledná kostra z algoritmu, T_0 taková minimální kostra na G, která se s T shoduje na co nejvíce (seřazených) hranách.
Sporem dokážeme, že $T_0 = T$.
Předpokládejme $T_0 \neq T$ a buď j nejmenší index, takový, že se T a T_0 liší v hraně e_j (zřejmě $e_j \in T \setminus T_0$). Pak $T_0 \cup \{e_j\}$ obsahuje právě jednu kružnici C. Na této kružnici tedy existuje hrana $e_k (k > j)$, která není v T. Pak ale $w(e_k) \geq w(e_j)$ a kostra s hranami $T_0 \setminus \{e_k\} \cup \{e_j\}$ není horší než T_0 a protože se od T liší „později“, měli jsme ji na začátku vybrat místo T_0. Spor. □

I druhý z našich algoritmů pro kostru grafu v předchozím odstavci vede na minimální kostru, když v každém okamžiku volíme ze všech možných hran $e_i = \{v_i, v_{i+1}\}$, $v_i \in V_i$, $v_{i+1} \in V \setminus \{v_i\}$ tu, která má minimální ohodnocení.

I druhý z našich algoritmů pro kostru grafu v předchozím odstavci vede na minimální kostru, když v každém okamžiku volíme ze všech možných hran $e_i = \{v_i, v_{i+1}\}$, $v_i \in V_i$, $v_{i+1} \in V \setminus \{v_i\}$ tu, která má minimální ohodnocení.
Výsledný postup je **Primův algoritmus** z roku 1957. Byl ale popsán Jarníkem již v roce 1930. Říkáme mu tedy **Jarníkův algoritmus**. Jarník vycházel z algoritmu O. Borůvky z r. 1926.

Věta

Jarníkův algoritmus najde minimální kostru pro každý souvislý graf s libovolným ohodnocením hran.

I druhý z našich algoritmů pro kostru grafu v předchozím odstavci vede na minimální kostru, když v každém okamžiku volíme ze všech možných hran $e_i = \{v_i, v_{i+1}\}$, $v_i \in V_i$, $v_{i+1} \in V \setminus \{v_i\}$ tu, která má minimální ohodnocení.
Výsledný postup je **Primův algoritmus** z roku 1957. Byl ale popsán Jarníkem již v roce 1930. Říkáme mu tedy **Jarníkův algoritmus**. Jarník vycházel z algoritmu O. Borůvky z r. 1926.

Věta

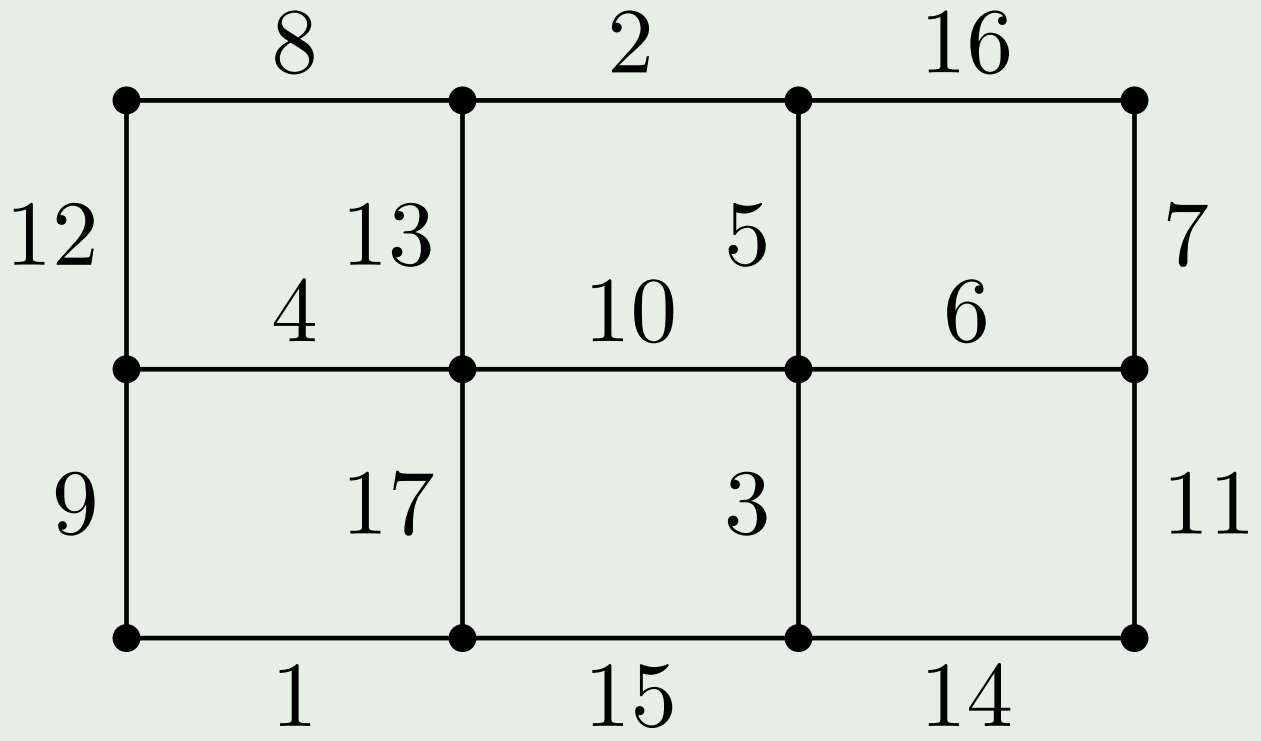
Jarníkův algoritmus najde minimální kostru pro každý souvislý graf s libovolným ohodnocením hran.

Poznámka

Borůvkův algoritmus tvoří stále co nejvíce souvislých komponent zaráz. Začneme s jednoprvkovými komponentami v grafu $T_0 = (V, \emptyset)$ a pak postupně každou komponentu propojíme nejkratší možnou hranou s komponentou jinou. Opět takto obdržíme minimální kostru. Tento algoritmu je základem nejrychlejšího známého algoritmu, běžícího v čase $O(n + m)$.

Příklad

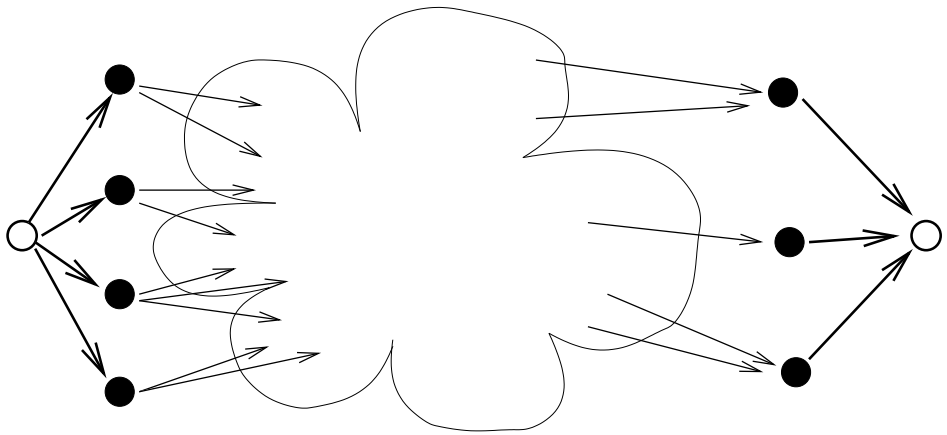
Určete pomocí uvedených algoritmů minimální kostru grafu



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 se 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 (*flow network*)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 (*flow network*)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 (někdy nazýváno *Kirchhoffův zákon*), 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 (*flow network*)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 (někdy nazýváno *Kirchhoffův zákon*), 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

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

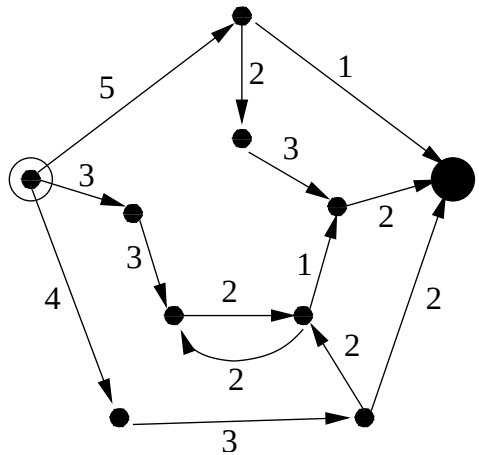
Z definice je zřejmé, že velikost toku můžeme stejně dobře vypočíst jako hodnotu

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

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.

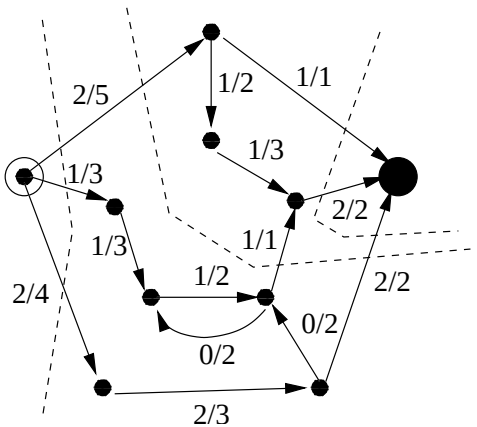
Definice

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 **kapacita (velikost) řezu** C.

Evidentně platí, že nikdy nemůžeme najít větší tok, než je kapacita 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 kapacitě 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 kapacitě 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 kapacitě 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.

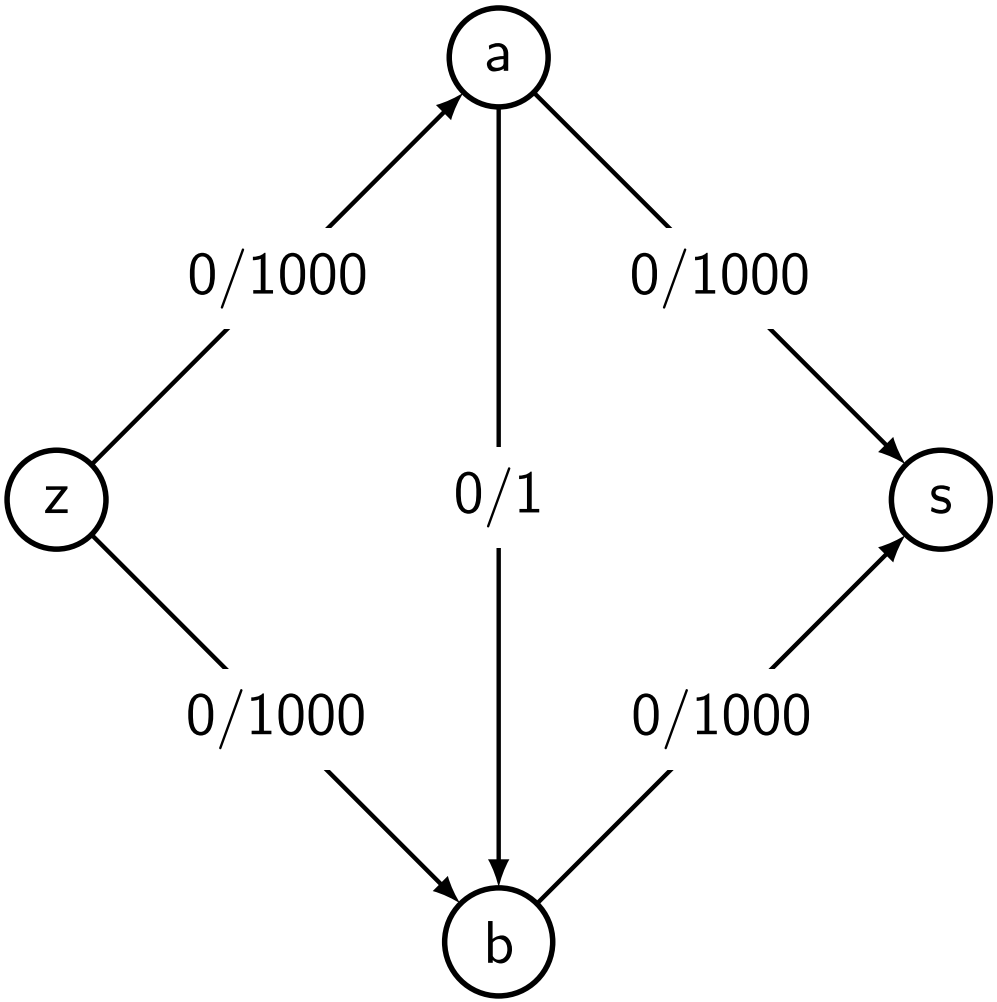
Důkaz.

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

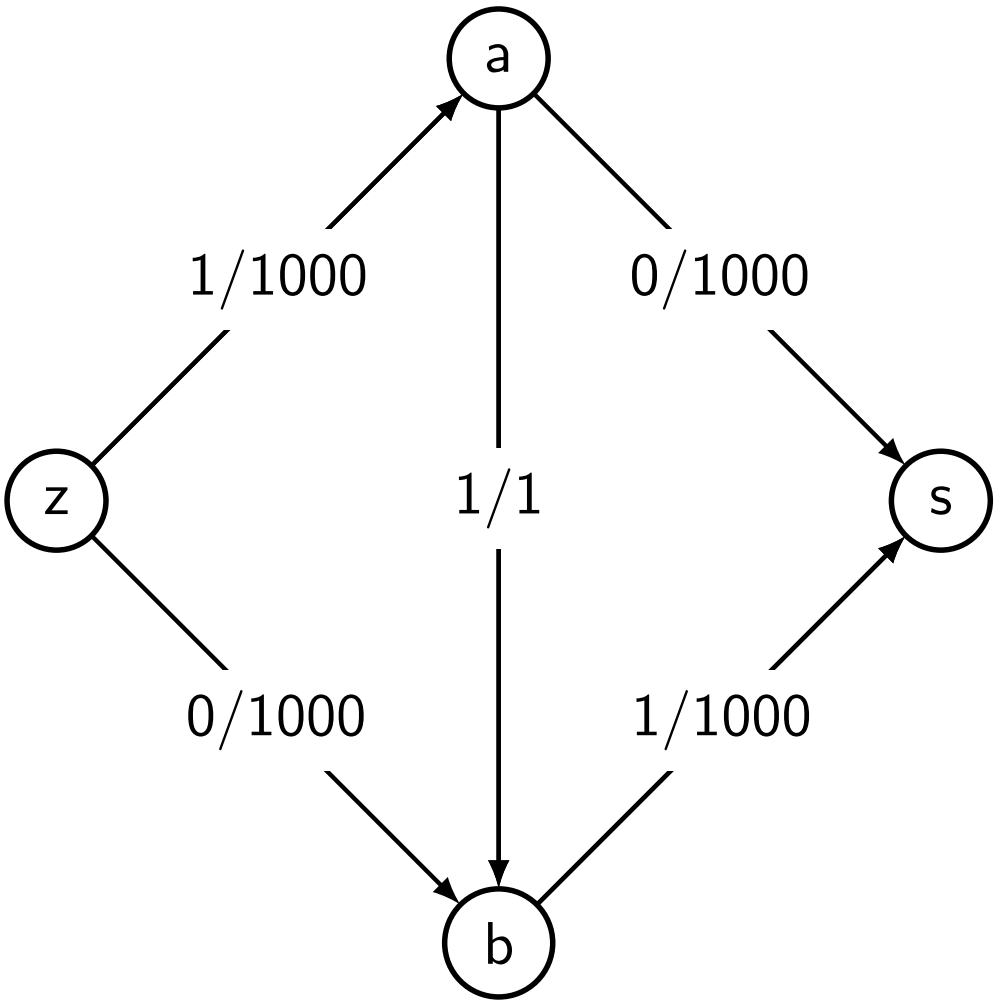
Poznámka

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

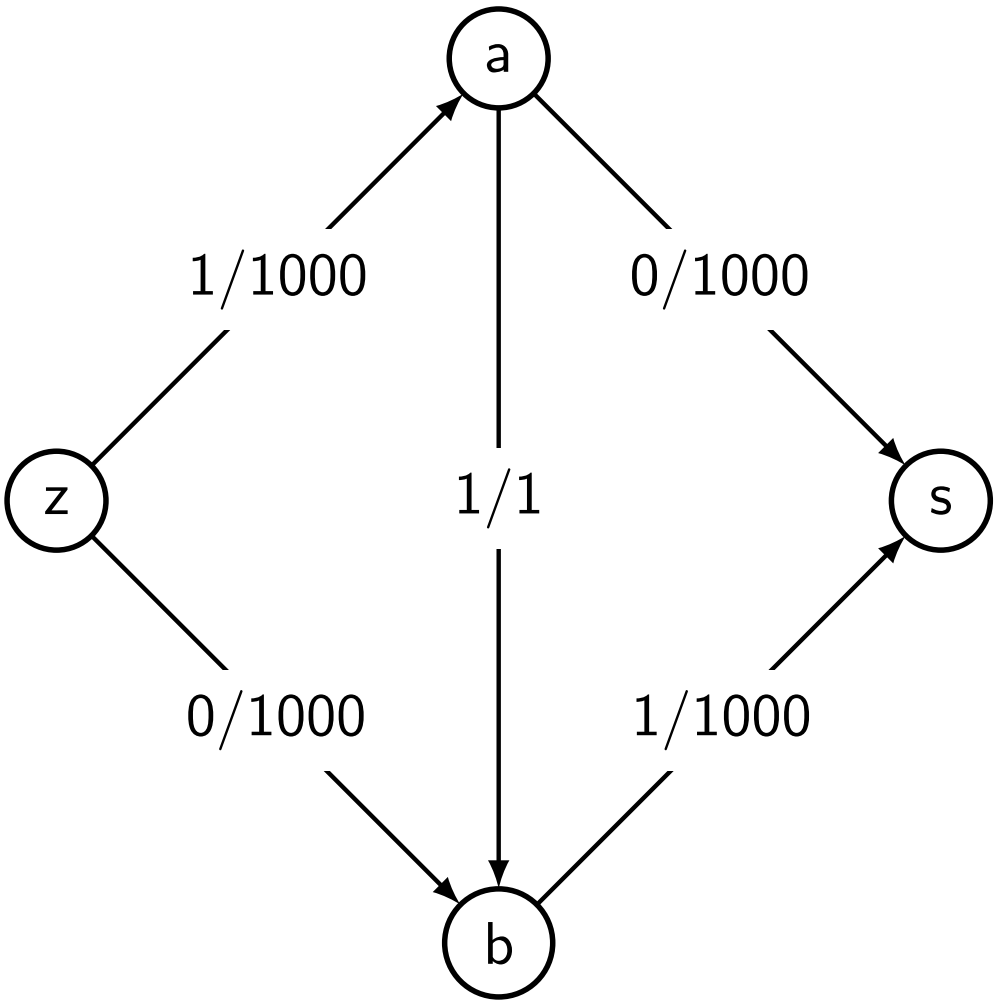
Příklad špatného chování F-F algoritmu



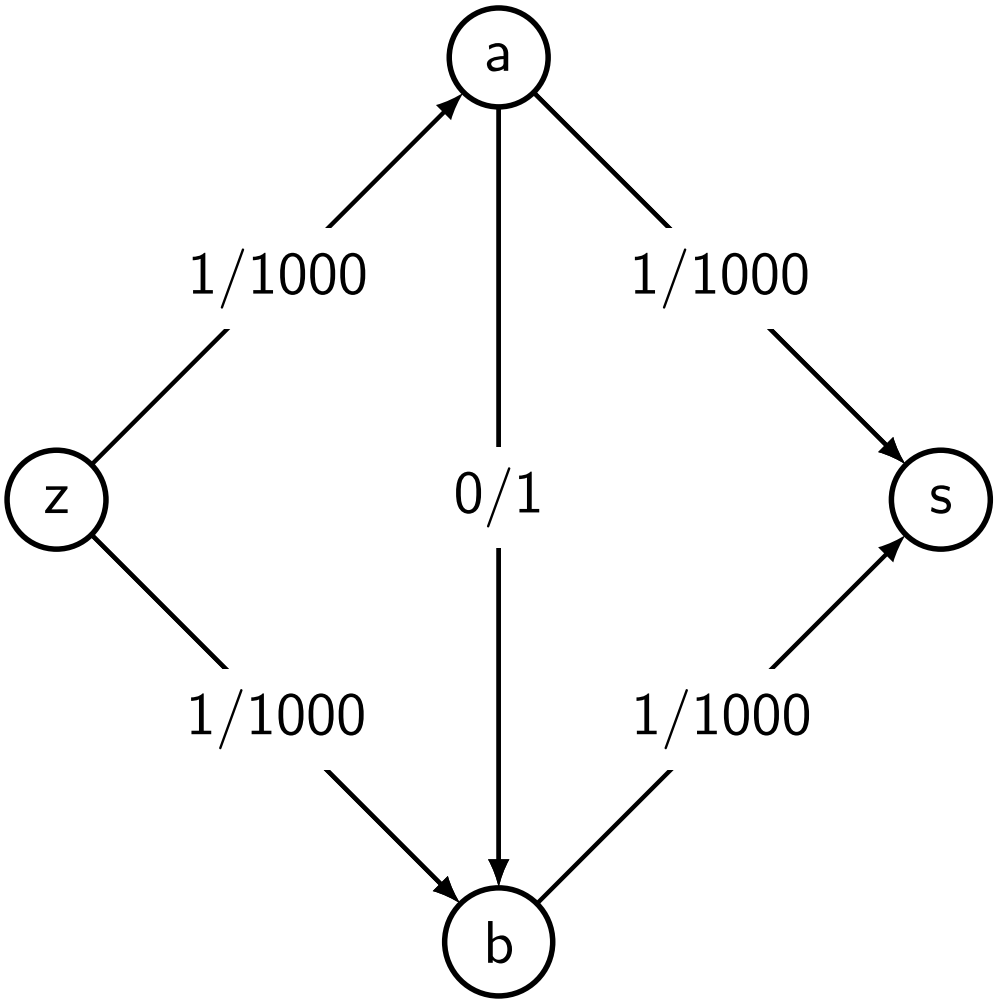
Příklad špatného chování F-F algoritmu



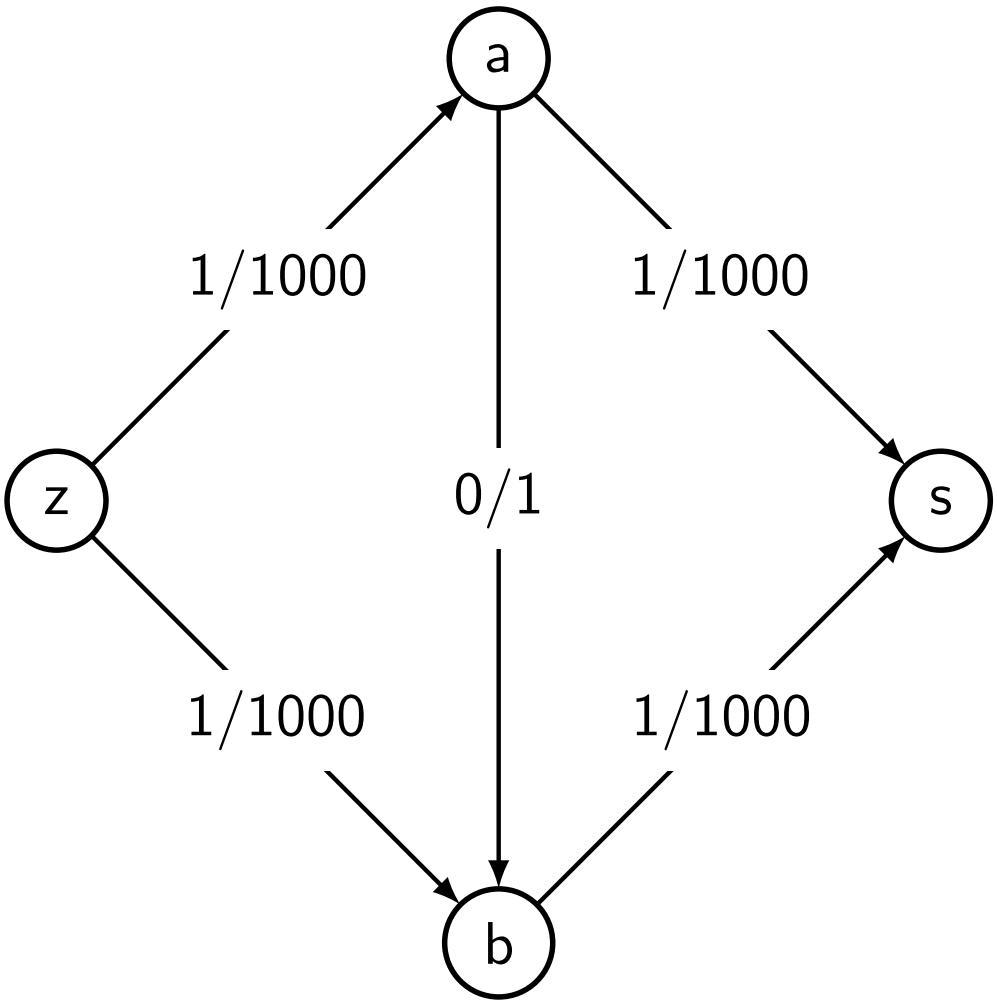
Příklad špatného chování F-F algoritmu



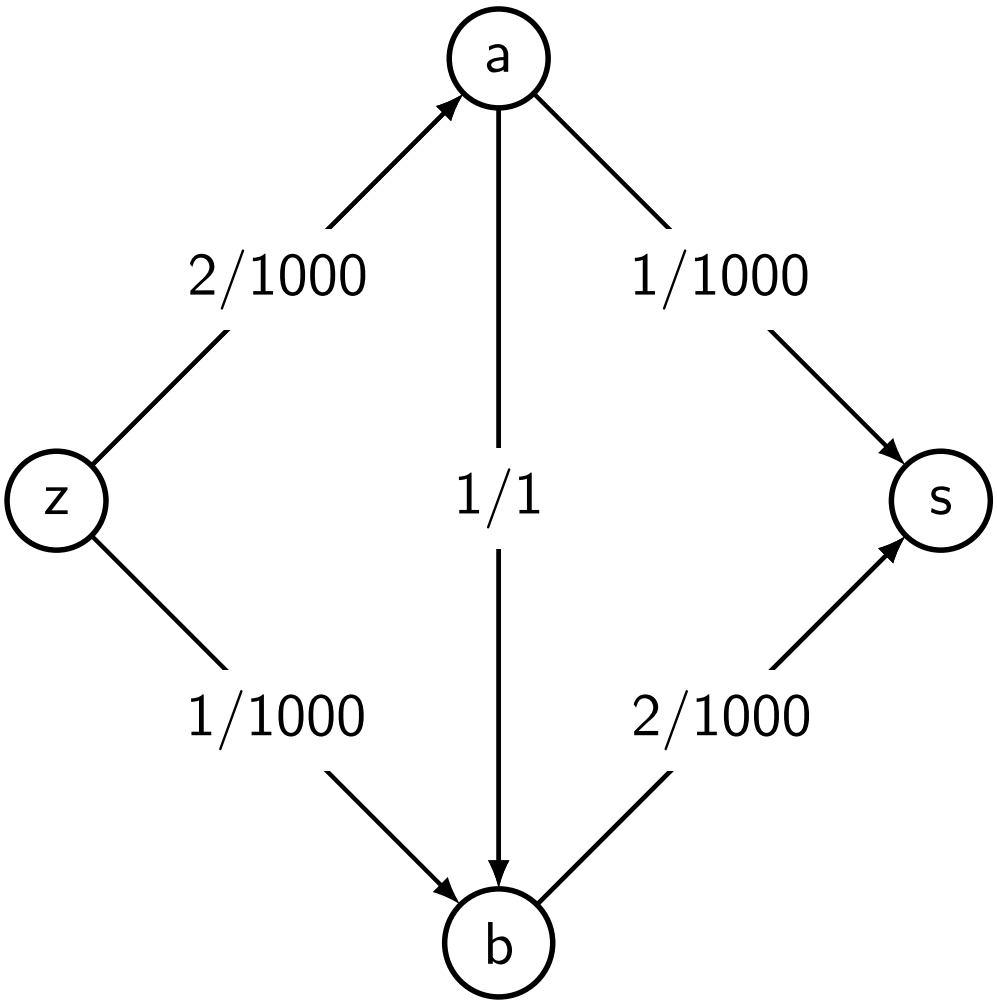
Příklad špatného chování F-F algoritmu



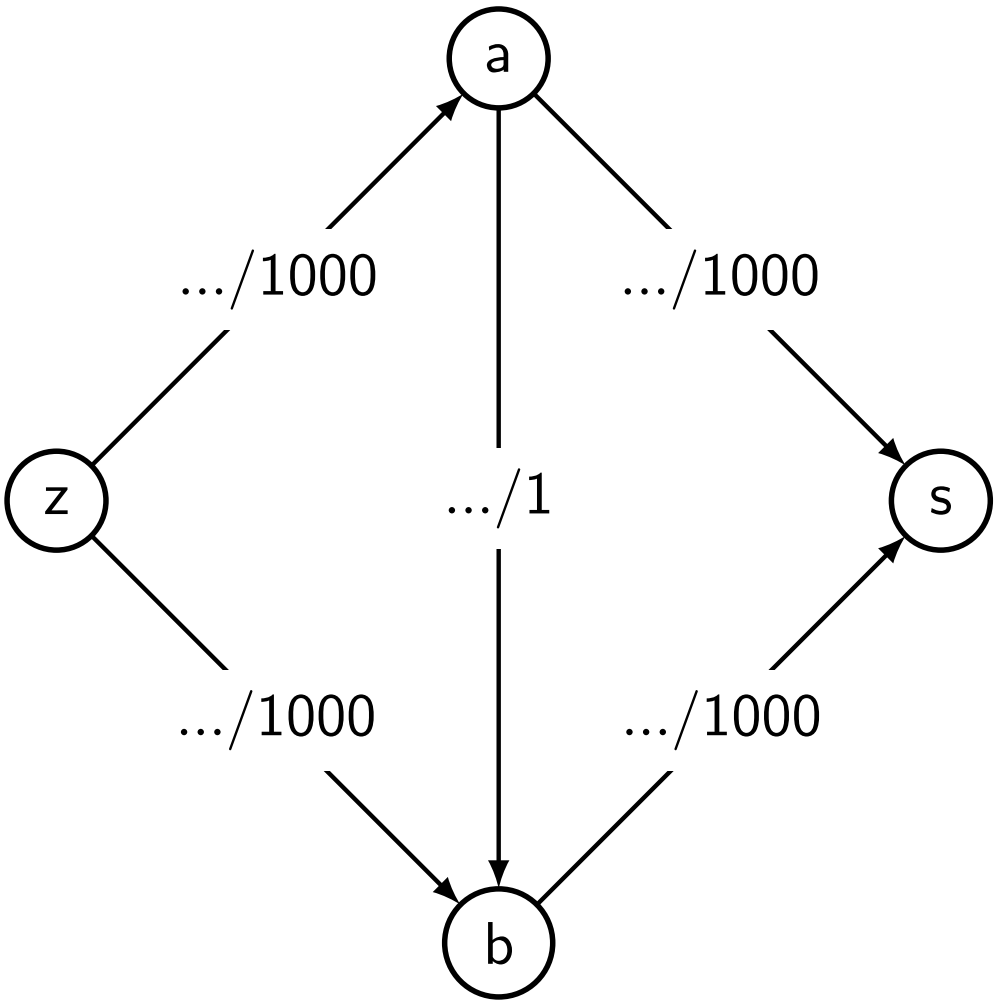
Příklad špatného chování F-F algoritmu



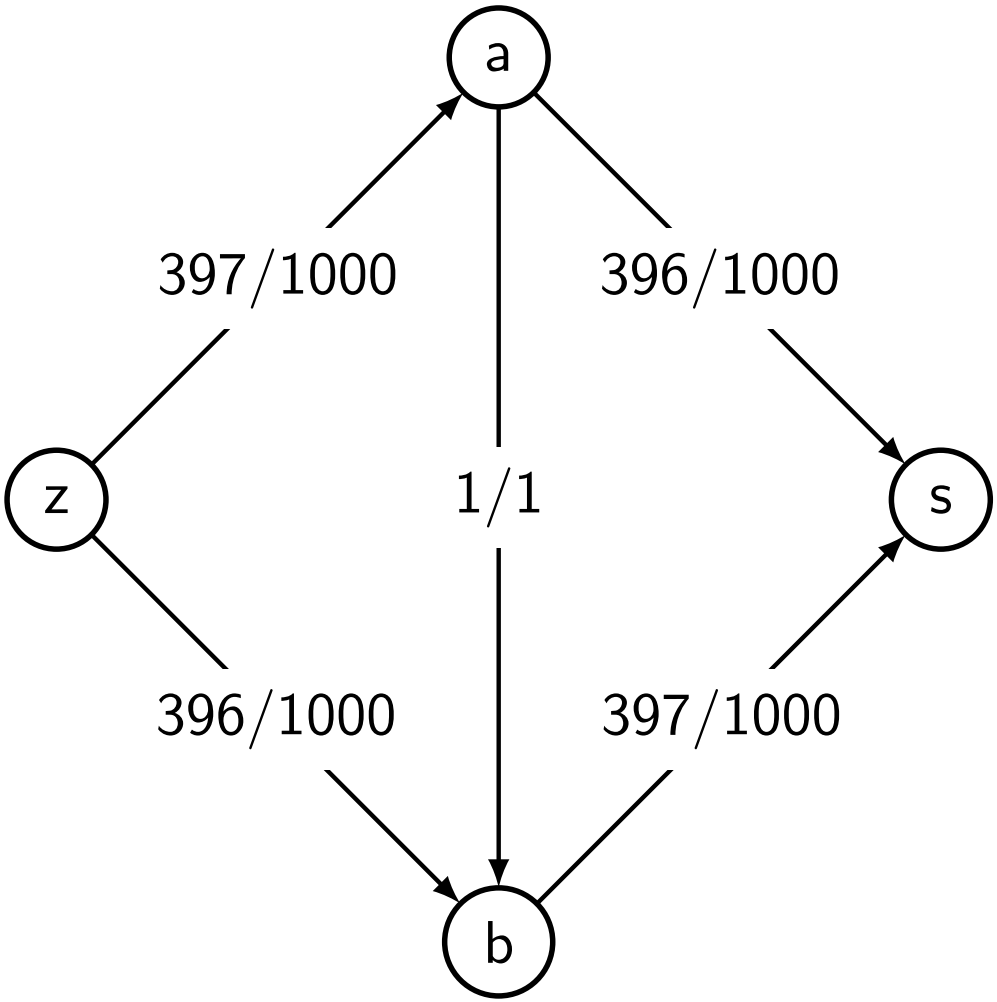
Příklad špatného chování F-F algoritmu



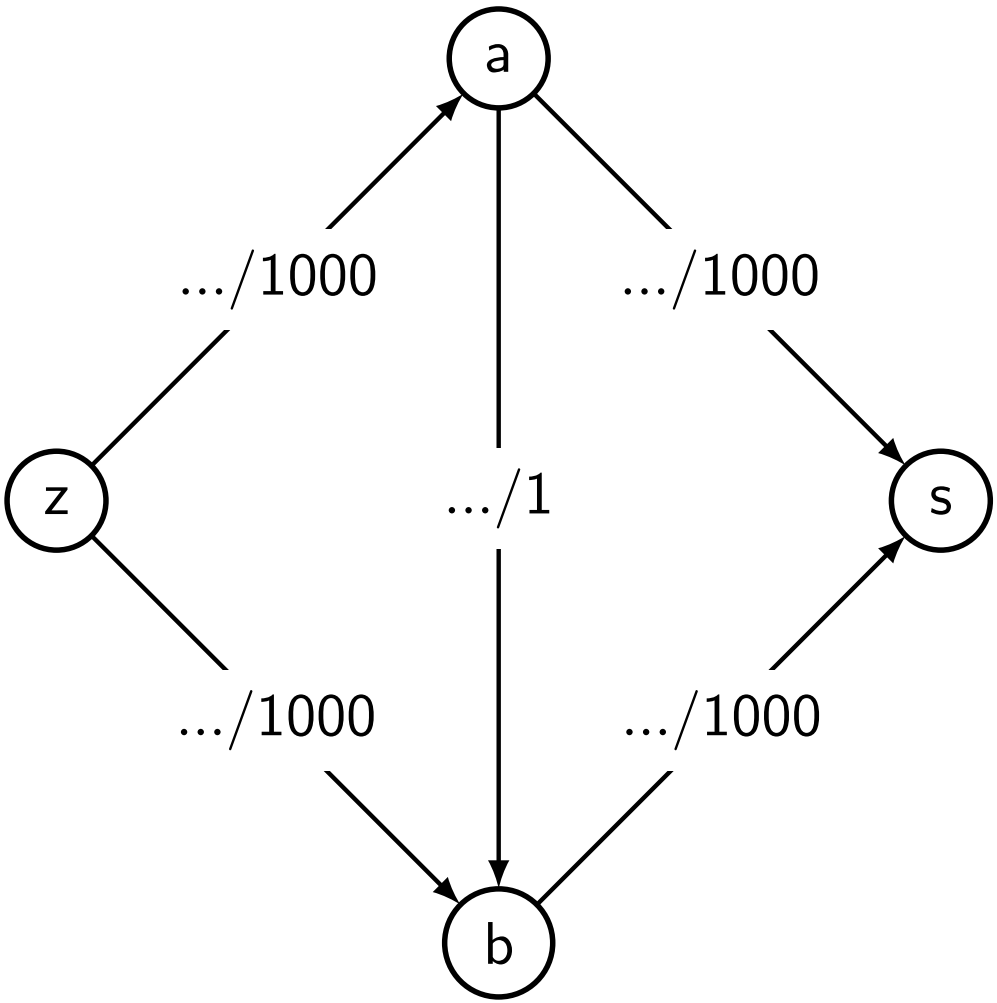
Příklad špatného chování F-F algoritmu



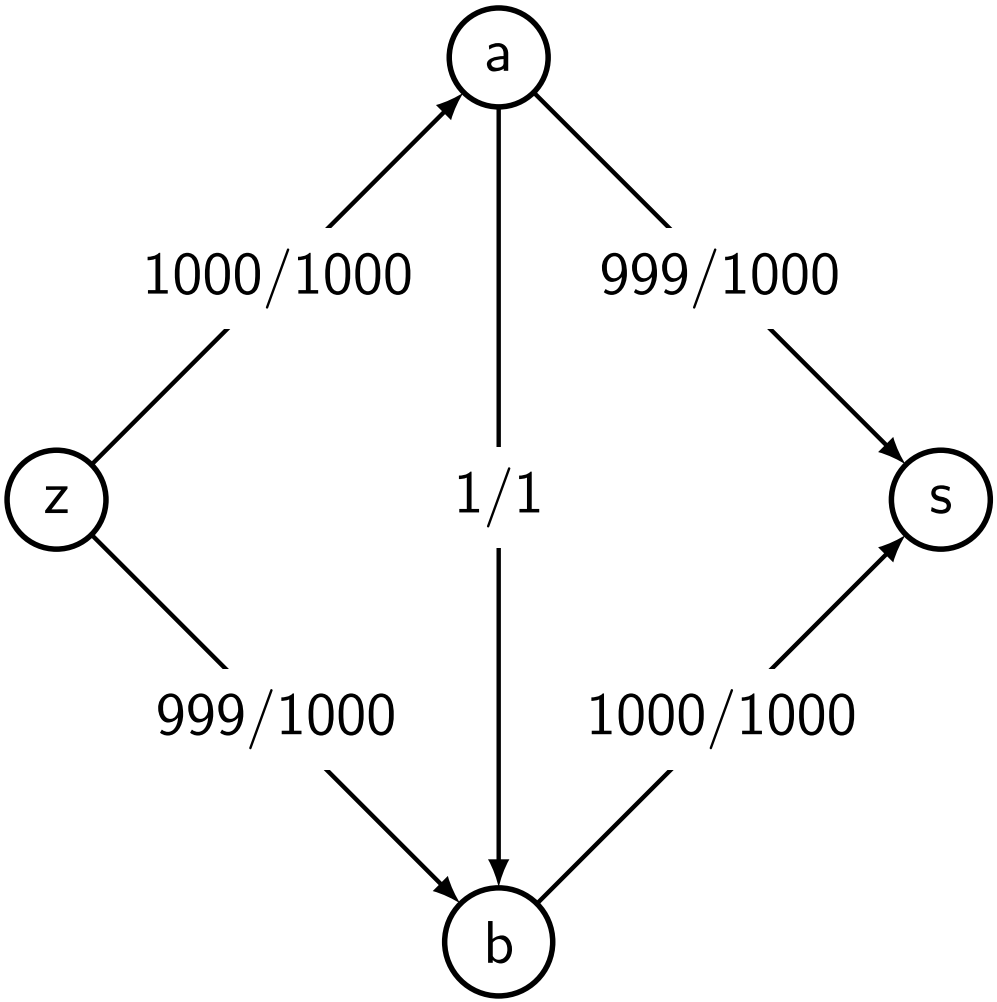
Příklad špatného chování F-F algoritmu



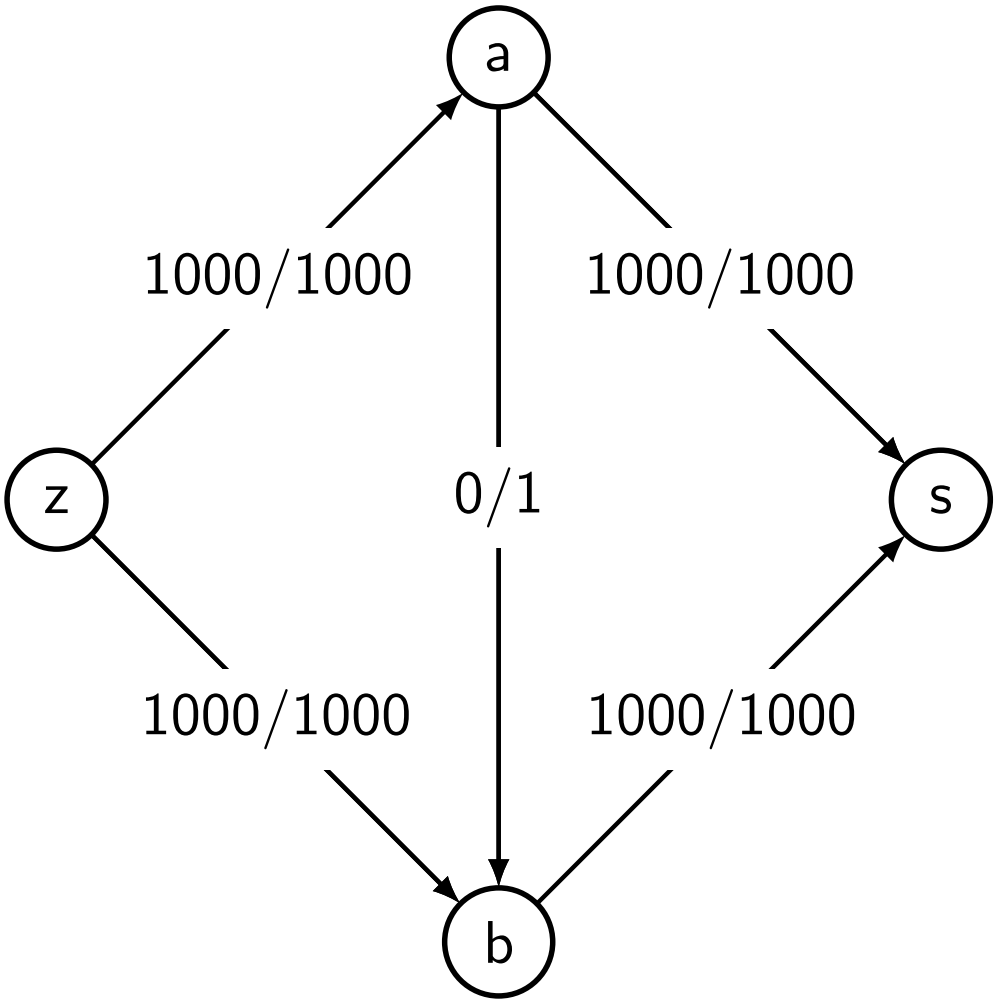
Příklad špatného chování F-F algoritmu



Příklad špatného chování F-F algoritmu

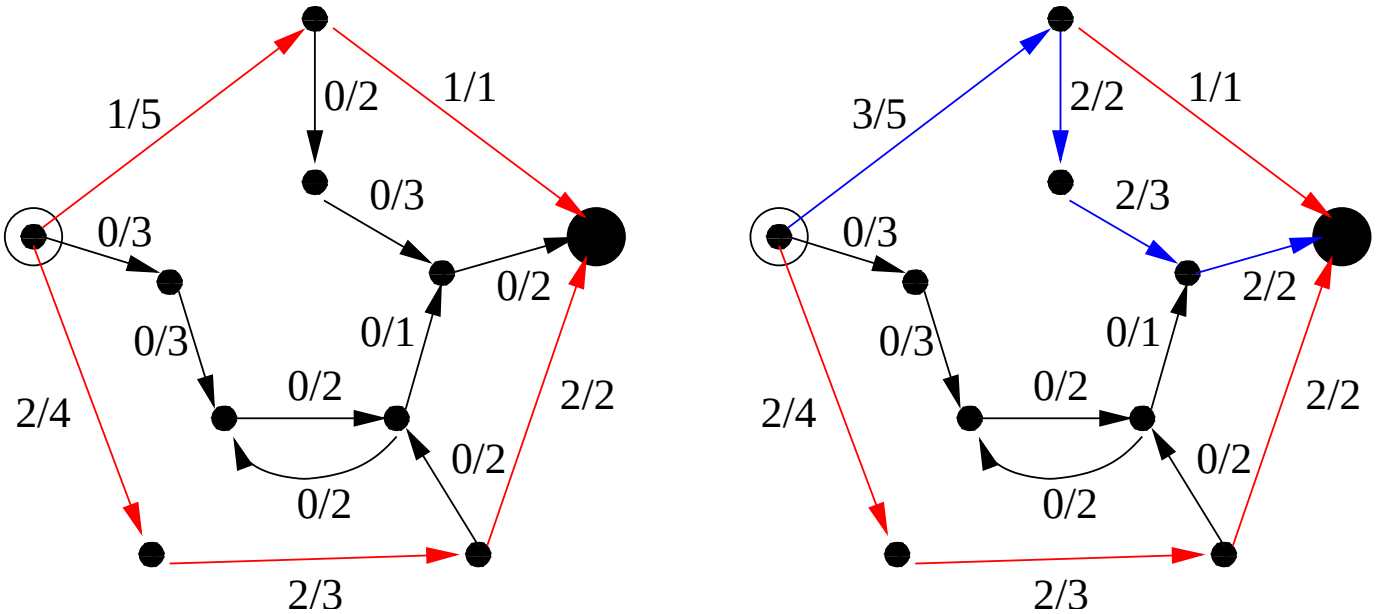


Příklad špatného chování F-F algoritmu



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.

Moderní algoritmy pro maximální tok

Česky viz např. Petr Parubják, XXVI. ASR '2001 Seminar, Instruments and Control, Ostrava, April 26 - 27, 2001 (`http://www.fs.vsb.cz/akce/2001/asr2001/Proceedings/papers/55.pdf`)

Diničův algoritmus

Zjednodušuje hledání nenasycené cesty konstrukcí tzv. *úrovňového grafu*, kdy *zlepšující hrany* uvažujeme pouze tehdy, pokud vedou mezi vrcholy různých vzdáleností od zdroje.
Složitost je $O(V^2E)$, což je u hustých grafů významné vylepšení oproti složitosti $O(VE^2)$ algoritmu Edmondse-Karpa.

Moderní algoritmy pro maximální tok

Česky viz např. Petr Parubják, XXVI. ASR '2001 Seminar, Instruments and Control, Ostrava, April 26 - 27, 2001 (`http://www.fs.vsb.cz/akce/2001/asr2001/Proceedings/papers/55.pdf`)

Diničův algoritmus

Zjednodušuje hledání nenasycené cesty konstrukcí tzv. *úrovňového grafu*, kdy *zlepšující hrany* uvažujeme pouze tehdy, pokud vedou mezi vrcholy různých vzdáleností od zdroje.
Složitost je $O(V^2E)$, což je u hustých grafů významné vylepšení oproti složitosti $O(VE^2)$ algoritmu Edmondse-Karpa.

MPM algoritmus

Malhotra, Pramodh Kumar a Maheshwari přišli s algoritmem složitosti $O(V^3)$. Konstruují stejným způsobem úrovňový graf, ale vylepšují hledání maximální toku v tomto úrovňovém grafu.

Viz `http://www.cosc.canterbury.ac.nz/tad.takaoka/alg/`

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.