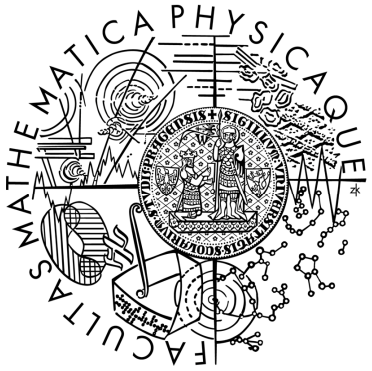
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Gabriela Znamenáčková

Hledání optimální cesty v grafech

Katedra pravděpodobnosti a matematické statistiky
Vedoucí bakalářské práce: doc. RNDr. Petr Lachout, CSc.

Studijní program: Matematika

Studijní obor: Finanční matematika

Praha 2011

Dovoluji si na tomto místě poděkovat doc. RNDr. Petru Lachoutovi, CSc., vedoucímu této práce, za podporu a cenné připomínky při vzniku této bakalářské práce.

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne..... Gabriela Znamenáčková

Název práce: Hledání optimální cesty v grafech

Autor: Gabriela Znamenáčková

Katedra / Ústav: Katedra pravděpodobnosti a matematické statistiky

Vedoucí bakalářské práce: doc. RNDr. Petr Lachout, CSc., Katedra pravděpodobnosti a matematické statistiky

Abstrakt: Mnoho rozhodovacích situací v praxi je možné modelovat pomocí ohodnoceného grafu. Podstatné je pak nalezení optimálního řešení dané situace na základě tohoto modelu. Předmětem této práce je především poskytnout přehled typických úloh kombinatorické optimalizace, které se zabývají hledáním optimální cesty v grafu vzhledem k daným kritériím, a algoritmů k nalezení jejich optimálního řešení. Jedná se především o úlohy nalezení nejkratší cesty v grafu, nalezení minimální kostry a minimálního Steinerova stromu, problém obchodního cestujícího a optimálního toku v síti. Činnost některých algoritmů je znázorněna na ilustrativních příkladech.

Klíčová slova: ohodnocený graf, kostra, Steinerův strom, hamiltonovská kružnice, tok v síti

Title: Searching for optimal paths in graphs

Author: Gabriela Znamenáčková

Department: Department of Probability and Mathematical Statistics

Supervisor: doc. RNDr. Petr Lachout, CSc., Department of Probability and Mathematical Statistics

Abstract: It's possible to simulate a lot of real decision-making situations by a weighted graph. Consequently it's important to find the optimal solution of a given situation based on this model. The subject of this Bachelor Thesis is to present the typical problems of combinatorial optimization, that deal with finding the optimal path in a graph considering the given conditions, and algorithms to find their optimal solution. It's focused on following problems: the shortest path problem, the minimum cost spanning-tree problem, the minimum cost Steiner tree problem, the travelling salesman problem and the optimal network flow. Working of some algorithms is shown on illustrative examples.

Keywords: weighted graph, spanning tree, Steiner tree, hamiltonian cycle, network flow

Obsah

Úvod

Ve své bakalářské práci blíže pojednávám o algoritmech pro hledání optimálních cest v grafech vzhledem k předem určeným podmínkám (např. projít všechny vrcholy za minimální cenu). V běžném životě se setkáváme se situacemi, kdy je potřeba takovou optimální cestu najít. Cílem práce je uvést základní algoritmy pro řešení problémů tohoto typu. V 1. kapitole uvádím některé základní pojmy a tvrzení o grafech, jejichž znalost je v následujících kapitolách pro pochopení dané problematiky nutná. Neuvádím je však zde s důkazem, stejně tak jako některá obecná úvodní tvrzení v dalších kapitolách, příslušné důkazy nejsou pro pochopení fungování algoritmu stěžejní a lze je najít v odkazované literatuře. Ve 3. kapitole formuluji obecně problém optimální cesty a ve 4. kapitole navazuji uvedením tří algoritmů pro hledání nejkratší cesty. V 5. kapitole pojednávám o problému hledání minimální kostry, kde ukazuji řešení pomocí tří algoritmů na konkrétním grafu. V 6. kapitole uvádím aproximační algoritmus pro minimální Steinerův strom. Dále se v 7. kapitole věnuji především problému obchodního cestujícího a 8.kapitola pak představuje náhled do problematiky toků v sítích. V závěrečné kapitole uvádím praktické příklady a nastiňuji metodu ACO.

Vzhledem k rozsahu práce není možné odvozovat u každého algoritmu jeho časovou složitost, proto časové složitosti uvádím bez odvození.

1.Úvod do teorie grafů

1.1.Základní pojmy a definice o grafech

Mnoho situací v reálném světě lze vystihnout pomocí schémat, která se skládají především z množiny bodů a spojnic mezi některými dvojicemi. Uvedeme-li nějaký příklad z praxe, pak body mohou reprezentovat účastníky turnaje a existence spojnic mezi body udává, kteří dva účastníci mezi sebou hráli. Dále lze grafy reprezentovat např. elektrické sítě nebo schéma cest v nějakém městě (křižovatky by zde byly reprezentovány body a ulice spojnicemi bodů). Body pak nazýváme *vrcholy* (nebo *uzly*) a spojnice *hrany*. Takovým schématům pak říkáme *grafy*. V následujícím textu čerpám především z knih [5] , [7], [8], [9], [10].

1.1.1 Definice. Graf G je uspořádaná dvojice (V, E), kde V je nějaká neprázdná množina a E je množina dvoubodových podmnožin množiny V. Prvky množiny V se jmenují vrcholy grafu G a prvky množiny E hrany grafu G.

Dále budeme předpokládat pouze grafy s konečnou množinou vrcholů. Množinu vrcholů grafu G budeme značit $V(G)$ a množinu hran grafu G budeme značit $E(G)$. Množinu všech dvoubodových podmnožin množiny V budeme značit $\binom{V}{2}$. Grafy znázorňujeme kreslením do roviny. Vrcholům grafu přiřazujeme body roviny (puntíky) a hrany se vyznačují jako oblouky spojující dva body v rovině. Uvedeme několik základních typů grafů.

Budeme předpokládat, že $V(G)$ je množina vrcholů. Potom K_V *je úplný graf* s množinou vrcholů $V(G)$ a jeho hrany tvoří každá dvojice různých vrcholů z $V(G)$. Můžeme-li množinu $V(G)$ rozdělit na dvě disjunktní podmnožiny V_1 a V_2 tak, že každá hrana $E(G)$ spojuje vrchol z V_1 s vrcholem V_2, pak se graf G nazývá *bipartitní*.

Uveďme nyní přehled základních druhů grafů:

1) *Úplný graf* K_n ($n \geq 1$), Obrázek 1.1.1.

Množina vrcholů $V = \{1, 2, ..., n\}$

Množina hran $E = \{\{v_i, v_j\}; i = 1, 2, ..., n, j = 1, 2, ..., m\}$

2) *Úplný bipartitní graf* $K_{n,m}$ $(n, m \geq 1)$: Obrázek 1.1.2.

$V = \{v_{11}, ..., v_{1n}\} \cap \{v_{21}, ..., v_{2m}\}$

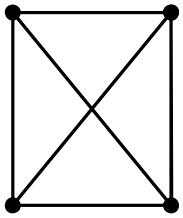
$E = \{\{v_{1i}, v_{2,j}\}; i=1, 2,..., n, j=1, 2, ..., m\}$

3) *Kružnice* C_n $(n \geq 3)$: Obrázek 1.1.3.

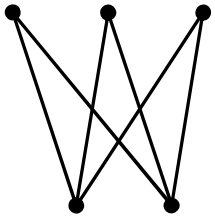
$V = \{1, 2, ..., n\}, E = \{\{i,i+1\}; i=1, ..., n-1\} \cap \{\{1, n\}\}$

4) *Cesta* P_n $(n \geq 0)$: Obrázek 1.1.4.

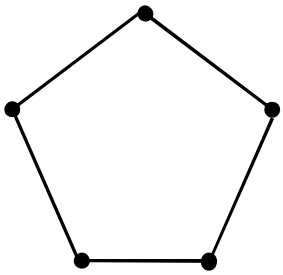
$V = \{0, 1,..., n\}, E = \{\{i-1,i\}; i=1,...,n\}$



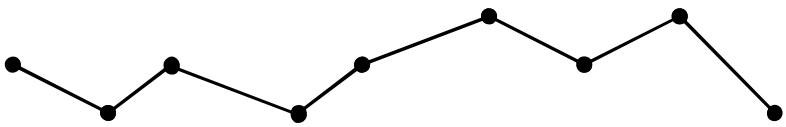
Obrázek 1.1.1.



Obrázek 1.1.2.



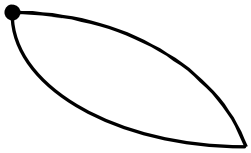
Obrázek 1.1.3.



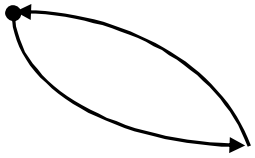
Obrázek 1.1.4.

V grafu se může vyskytovat mezi 2 vrcholy více hran, hrany mohou být orientované či spojovat vrchol sám se sebou (smyčky). Uveďme zde některé základní druhy hran a jejich znázornění

neorientovaná hrana 

neorientovaná smyčka 

orientovaná hrana (šipka) 

orientovaná smyčka 

Dva grafy jsou stejné, pokud jsou jejich množiny vrcholů a hran totožné. Liší-li se však dva grafy jen označením svých vrcholů a hran, pak hovoříme o isomorfismu.

1.1.2 Definice. Dva grafy $G = (V , E)$ a $G' = (V', E')$ se nazývají *isomorfní*, jestliže existuje vzájemně jednoznačné přiřazení $f : V \to V'$, (f je tedy zobrazení prosté a na) takové, že platí: $\{x, y\} \in E$, právě když $\{f(x), f(y)\} \in E'$. Zobrazení f se nazývá *isomorfismus* grafů G a G'.

1.2. Podgrafy, sledy, cesty, metrika grafu

1.2.1 Definice. Graf H je *podgrafem* grafu G, jestliže platí $V(H) \subseteq V(G)$ a $E(H) \subseteq E(G)$.

1.2.2 Definice. Nechť $G = (V, E)$ je graf. Posloupnost $v_o, e_1, v_1,...,e_n, v_n$ se nazývá *sled* v grafu G délky n z v_0 do v_n, jestliže platí $e_i = \{v_{i-1}, v_i\} \in E$ pro $i = 1, ..., n$. Platí-li $e_i = \{v_{i-1}, v_i\} \in E$ pro $i = 1, ..., n$ a $e_i \neq e_j$ pro $i \neq j$, pak se tato posloupnost nazývá *tah* v grafu G délky n z v_0 do v_n. Pokud platí $e_i = \{v_{i-1}, v_i\} \in E$ pro $i = 1, ..., n$ a $v_i \neq v_j$ pro $i \neq j$, pak se tato posloupnost nazývá cesta v grafu G délky n z v_0 do v_n.

Cestou v grafu G nazveme podgraf grafu G, který je isomorfní nějaké cestě P_t. Podgraf grafu G, který je isomorfní nějaké kružnici C_t ($t \geq 3$) se nazývá *kružnice* v grafu G.

1.2.3 Definice. Graf G je souvislý, jestliže pro každé jeho dva vrcholy x a y v něm existuje cesta z x do y.

1.2.4 Definice. Rozložíme-li množinu vrcholů $V = V_1 \cap V_2 \cap ... \cap V_k$ na třídy ekvivalence, potom V_i se nazývají *komponenty* grafu G. Graf G je sjednocením svých komponent a je souvislý, právě když $k = 1$.

1.2.5 Definice. Nechť $G = (V, E)$ je souvislý graf. Pro vrcholy v a v' definujme číslo $d_G(v, v')$ jako délku nejkratší cesty z v do v' v grafu G. Číslo $d_G(v, v')$ se nazývá vzdálenost vrcholů v a v' v grafu G. Funkci d_G: $V \times V \rightarrow \mathbf{R}$ nazýváme *metrika grafu* G a má tyto vlastnosti:

1. $d_G(v, v') \geq 0$ a $d_G(v, v') = 0$, právě když $v = v'$
2. $d_G(v, v') = d_G(v', v)$ pro každou dvojici vrcholů v a v'
3. $d_G(v, v'') \leq d_G(v, v') + d_G(v', v'')$ pro každou trojici vrcholů v, v', v'' z množiny V
4. $d_G(v, v')$ je nezáporné celé číslo pro každou dvojici v, v'

5. je-li $d_G(v, v'') > 1$, pak existuje vrchol v', $v \neq v'$, $v' \neq v''$ tak, že $d_G(v, v') + d_G(v', v'') = d_G(v, v'')$

1.3. Algebraická reprezentace grafu

Grafy je možné reprezentovat nejen geometricky, ale též pomocí algebraických prostředků. Algebraické reprezentace se využívá především při zadávání grafu počítači. Základními prostředky jsou matice sousednosti a matice incidence.

1.3.1 Definice. Nechť $G = (V, E)$ je graf s n vrcholy. Označíme jeho vrcholy $v_1,...,v_n$ v libovolném, ale pevném pořadí. Potom *matice sousednosti* grafu G je čtvercová matice $\boldsymbol{A_G} = (a_{ij})_{i,j=1}^{n}$ definovaná předpisem

$a_{ij} = 1$ *pro* $\{v_i, v_j\} \in E$

$a_{ij} = 0$ *pro* $\{v_i, v_j\} \notin E$.

Vidíme, že matice sousednosti je vždy symetrická čtvercová matice, která má prvky 0 a 1, na hlavní diagonále pouze 0. Je zřejmé, že matice sousednosti závisí na zvoleném očíslování vrcholů grafu.

1.3.2 Definice. Nechť $G = (V, E)$ je orientovaný graf bez smyček s n vrcholy. Označíme v libovolném, ale pevném pořadí jeho vrcholy $v_1,...,v_n$ a hrany $e_1, ..., e_n$. Grafu G pak přiřadíme matici incidence $\boldsymbol{B_G}$ typu m x n předpisem:

$b_{ij} = 1$	je-li vrchol v_i počátečním vrcholem hrany e_j
$b_{ij} = -1$	je-li vrchol v_i koncovým vrcholem hrany e_j
$b_{ij} = 0$	jinak

Řádky matice incidence reprezentují vrcholy grafu G, sloupce jednotlivé hrany.

1.4. Skóre grafu

Nechť v je vrchol v grafu G. Potom symbol $deg_G(v)$ označuje počet hran grafu G, které vrchol v obsahují. Číslo $deg_G(v)$ se nazývá stupeň vrcholu v v grafu G. Pokud označíme vrcholy $v_1, v_2, ..., v_n$ grafu G v libovolně zvoleném pořadí, potom posloupnost $(deg_G(v_1), deg_G(v_2), ..., deg_G(v_n))$ stupňů grafu G se nazývá *skóre* grafu G.

1.4.1 Tvrzení - Princip sudosti. *Pro každý graf $G = (V, E)$ platí*

$$\sum_{v \in V} deg_G(v) = 2\,|E|$$

Důkaz. Každá hrana obsahuje dva vrcholy a stupeň vrcholu udává počet hran grafu G, které obsahují vrchol v. Součet všech stupňů nám tedy musí dát dvojnásobek počtu hran.

Q.E.D.

1.4.2 Věta o skóre grafu. *Nechť $D = (d_1, d_2, ..., d_n)$ je posloupnost přirozených čísel. Budeme předpokládat, že $d_1 \leq d_2 \leq ... \leq d_n$,. Symbolem D' označíme posloupnost $(d_1', d_2', ..., d_n')$ takovou, že*

$d_i' = d_i$ *pro* $i < n - d_n$
$d_i' = d_i - 1$ *pro* $i \geq n - d_n$

Potom D je skóre grafu, právě když D´ je skóre grafu.

Důkaz. Lze najít v [7].

1.5. Eulerovské grafy

Budeme hledat uzavřený sled $(v_o, e_1, v_1, ..., e_{n-1}, v_{n-1}, e_n, v_o)$, ve kterém se každá hrana bude vyskytovat právě jednou a každý vrchol alespoň jednou. Takový sled je

pak *uzavřený eulerovský tah*. Říkáme, že graf G je eulerovský, právě když v něm lze najít alespoň jeden uzavřený eulerovský tah.

1.5.1 Věta o charakterizaci eulerovských grafů. *Graf G je eulerovský, právě když je souvislý a každý vrchol grafu G má sudý stupeň.*

Důkaz. Je uveden v [7].

Nyní zavedeme korektně pojem orientovaného grafu.

1.5.2 Definice. Orientovaný graf G je dvojice (V, E), kde E je podmnožina kartézského součinu $V \times V$. Prvky E se nazývají orientované hrany (šipky). Orientovanou hranu e zapisujeme ve tvaru $\{x, y\}$, což značí, že orientovaná hrana vychází z vrcholu x a končí ve vrcholu y.

1.5.3 Definice. Orientovaným tahem rozumíme posloupnost $(v_0, e_1, v_1, e_2, \ldots, e_n, v_n)$, pro kterou platí, že $e_i = (v_{i-1}, v_i) \in E$ pro každé $i = 1, 2, \ldots, n$ a navíc platí, že $e_i \neq e_j$ pro všechna $i \neq j$.

Pojmy orientovaný sled, orientovaná cesta a orientovaná kružnice (neboli cyklus) se definují podobně.

1.5.4 Definice. Orientovaný graf $G = (V, E)$ je eulerovský, existuje-li v něm uzavřený orientovaný tah, který obsahuje každou hranu právě jednou a každý vrchol alespoň jednou.

Číslo $deg_G^+(v)$ nazveme vstupním stupněm vrcholu, udává počet orientovaných hran, které končí ve vrcholu v. Naopak číslo $deg_G^-(v)$ označuje počet orientovaných hran, které z vrcholu v vystupují. Orientovaný graf je vyvážený, pokud pro každý vrchol $v \in V$ platí $deg_G^+(v) = deg_G^-(v)$. Každému orientovanému grafu $G = (V, E)$ můžeme přiřadit neorientovaný graf (tzv. *symetrizaci* grafu G) $sym(G) = (V, E')$, kde platí: $E' = \{\{x, y\}; \{x, y\} \in E$ nebo $\{y, x\} \in E\}$

1.5.5 Tvrzení. *Orientovaný graf je eulerovský, právě když je vyvážený a jeho symetrizace je souvislý graf.*

Důkaz. Je uveden v [7].

1.6 Stromy

1.6.1 Definice. Strom je souvislý graf neobsahující kružnici.

1.6.2 Lemma o koncovém vrcholu. *Pro každý strom s alespoň dvěma vrcholy existují alespoň dva vrcholy stupně 1. Vrchol stupně 1 se nazývá koncový vrchol nebo list.*

Důkaz. Nalezneme v [7].

1.6.3 Věta o charakterizaci stromů. *Pro graf G = (V, E) jsou následující podmínky ekvivalentní:*
1) G je strom.
2) Graf G je souvislý a vynecháme-li libovolnou hranu, pak vznikne graf nesouvislý.
3) Pro každé dva vrcholy x, y $\in$ *V existuje právě jedna cesta z x do y.*
4) Graf G neobsahuje kružnici a přidáme-li hranu, pak každý takto nově vzniklý graf již kružnici obsahuje.
5) Graf G je souvislý a platí $|V| = |E| + 1$

Důkaz. je uveden v [7].

Více se o problematice stromů (isomorfismu či jejich výstavbě) lze dočíst například v literatuře [7], [8] či [9] . Pro potřeby této práce je však tento drobný úvod postačující.

2. Základní třídy složitosti úloh

2.1 Složitost algoritmu

Algoritmy můžeme rozdělit do několika tříd složitosti podle doby prováděného algoritmu (časová složitost) a rozsahu použité operační paměti (prostorová složitost). Složitost algoritmu je závislá na velikosti vstupních dat, můžeme ji popsat jako funkci dat *T(n),* kde *n* označuje velikost vstupních dat. Zajímavý je především typ funkční závislosti, např. lineární, kvadratický atd. Za efektivní algoritmy se považují algoritmy, které mají polynomiální složitost (nejlépe s co nejnižším stupněm polynomu). Exponenciální algoritmy (např. 2^n) či algoritmy *T(n)* = n! mohou při malém navýšení vstupních dat trvat několik tisíců let!

Velice důležitý je tzv. horní odhad složitosti, protože nám dává složitost v nejhorším případě (tedy pro ta „nejnaschválnější“ data).

2.1.1 Definice. Složitost algoritmu *f(n)* je asymptoticky menší nebo rovna funkční závislosti udávající složitost *g(n),* značíme *f(n) = O(g(n)),* právě když:
$\exists\, c \in \mathbf{R}^+,\ c > 0,\ \exists\, n_0 > 0\ \forall\ n \in \mathbf{N},\ n \geq n_0 : 0 \leq f(n) \leq cg(n)$.

Dolní odhad složitosti nám naopak udává ideální složitost daného algoritmu (tedy nejrychlejší možné provedení), které je ale možné jen pro některé případy vstupních dat. Důkaz dolního odhadu složitosti úlohy bývá složitější, neboť je potřeba dokázat, že neexistuje algoritmus, který by daný problém dokázal vyřešit rychleji.

2.1.2 Definice. Složitost algoritmu *f(n)* je asymptoticky větší nebo rovna funkční závislosti udávající složitost *g(n),* značíme *f(n) = Ω(g(n))*, právě když:

$$\exists\, c \in \mathbf{R}^+,\ c > 0,\ \exists\, n_0 > 0\ \forall\ n \in \mathbf{N},\ n \geq n_0 : 0 \leq cg(n) \leq f(n)$$

Horního odhadu složitosti dosahuje algoritmus často jen ve velmi vzácných případech, o složitosti algoritmu pak lépe vypovídá průměrná složitost. Složitost v průměrném případě neboli očekávanou složitost vypočítáme jako střední hodnotu náhodné složitosti *T(n)* při nějakém rozložení vstupních dat.

2.1.3 Definice. Složitost algoritmu *f(n)* je asymptoticky stejná jako funkční závislosti udávající složitost g(n), značíme $f(n) = \theta(g(n))$, právě když:

$$\exists\, c_1, c_2 \in \mathbf{R}^+, c > 0, \exists\, n_0 > 0\ \forall\ n \in \mathbf{N}, n \geq n_0 : 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n).$$

2.2 Turingův stroj

Jedná se o teoretický model počítače, který se skládá z nekonečné oboustranné pásky rozdělené na políčka a pohyblivé hlavice. Hlavice může číst i zapisovat na nějaké políčko pásky jeden symbol z dané konečné abecedy (tj. libovolné konečné množiny, jejíž prvky nazveme symboly) $S_1, \ldots, S_m$. Abeceda musí obsahovat alespoň dva prvky. Řídící jednotka počítače se v každém časovém okamžiku $t = 1, 2, 3, \ldots$ nachází v jednom z konečných stavů $q_1, \ldots, q_n$. Tedy v nějakém časovém okamžiku t se pohyblivá hlavice nachází na některém políčku P pásky, kde přečte symbol S_i,, a řídicí jednotka je ve stavu q_j. Počítač pak může učinit jeden z následujících kroků:

1) řídicí hlavice vymaže symbol S_i z pole P a zapíše tam nějaký jiný symbol S_k
2) řídicí hlavice se posune o jedno políčko vpravo nebo vlevo
3) počítač se zastaví

Sestrojíme-li program pro Turingův počítač, pak se bude skládat z definování jedné ze tří výše uvedených možností a nového stavu řídicí jednotky pro každou dvojici (S_j, q_j). Úlohou pak bude zpracovat danou posloupnost symbolů na vstupu zapsanou na pásce.

2.3 NP – úplnost

Deterministickým algoritmem rozumíme takový algoritmus, který na stejné výchozí podmínky reaguje vždy stejně a v každém kroku máme definován krok následující. Opakem deterministického algoritmu je algoritmus nedeterministický. Budeme se zabývat podrobněji problémy třídy *NP* („nedeterministicky polynomiální´´). Jsou to takové množiny problémů, které můžeme řešit na nedeterministickém Turingově stroji (takový stroj, který na místě rozhodování uhodne správnou cestu výpočtu) v polynomiálně omezeném čase. Tedy *NP* je třída problémů, pro které existuje nedeterministický algoritmus, jehož rychlost je omezena shora nějakým polynomem.

Rozhodovacím problémem rozumíme úlohu, která má vydat úlohou specifikovaný výstup ve formě ANO / NE. Tedy v polynomiálně omezeném čase pro problém třídy *NP* umíme pro daný výsledek rozhodnout, zda je správný či nikoliv, ale nalézt řešení v polynomiálně omezeném čase neumíme.

2.3.1 Definice. Úloha *A* je *polynomiálně redukovatelná (P-redukovatelná)* na úlohu *B,* pokud existuje takový polynomiální algoritmus, který ze vstupních dat pro úlohu *A* vyrobí vstupní data pro úlohu *B*, přičemž odpovědi na obě úlohy budou stejné.

Pokud existuje *P-redukce* úlohy *A* na úlohu *B*, pak je úloha *A* lehčí nebo stejně obtížná jako úloha *B*. Je-li možná *P-redukce* v obou směrech, pak jsou obě úlohy stejně obtížné.

2.3.2 Definice. *NP-úplná* úloha je úloha patřící do *NP* a je ve třídě *NP* nejtěžší, jakákoli jiná *NP*-úloha je na ni *P-redukovatelná*.

2.3.3 Definice. *NP-těžká* úloha je taková úloha *B*, na kterou můžeme nějakou *NP*-úplnou úlohu *A P-redukovat*.

3. Základní formulace problému optimální cesty

3.1. Ohodnocený graf

Již jsme zavedli pojmy orientovaný sled, orientovaný tah, orientovaná cesta a cyklus. Každé hraně grafu můžeme přiřadit nějaké nezáporné číslo *w(e),* které se nazývá *váha*. Může představovat např. délku hrany či cenu za projití této hrany. Graf, ve kterém je každé hraně přiřazená nějaká váha, nazveme *ohodnocený graf.* V ohodnoceném grafu je délka sledu $(v_o, e_1,..., e_n, v_n)$ dána součtem ohodnocení hran, po kterých jsme „prošli". Je to tedy číslo $\sum_{i=1}^{n} w(e_i)$.

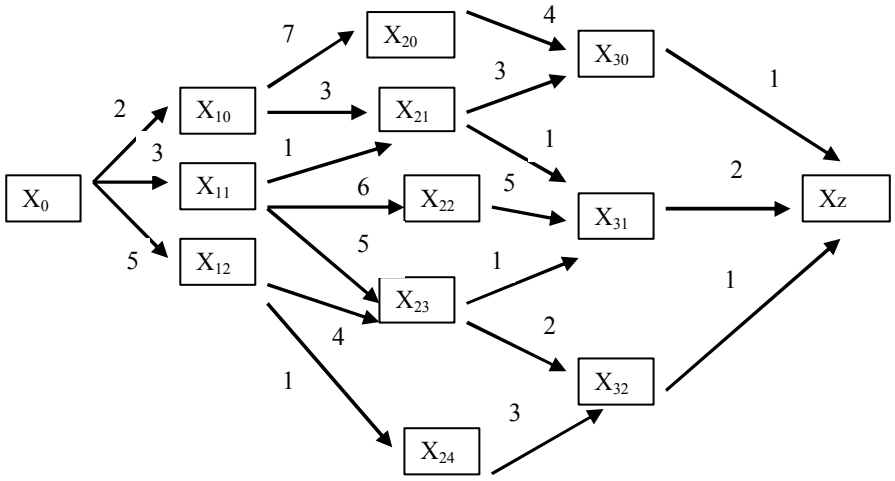
Uveďme jeden z nejčastěji řešených problému diskrétní optimalizace, a to problém nejkratší cesty.

3.2. Kritická cesta

Zabývejme se nejprve problémem nejkratší cesty a jeho formulací.

Úloha zní: Je dán orientovaný graf $G = (V, E)$ s nezáporným ohodnocením hran $w : E \to \mathbf{R}_0^+$. Pro danou dvojici vrcholů x a y nalezněme cestu z x do y s minimální možnou ohodnocenou délkou.Tato důležitá problematika se uplatňuje především v aplikacích grafů ve výpočetní technice, strojírenství a mnoha jiných oborech.

Budeme nyní uvažovat systém, který je v čase 0 ve stavu x_0 a v nějakém čase j může nabývat jednoho ze stavů $x_{j,1}, x_{j,2}, ..., x_{j,n(j)}$. Dále předpokládáme, že systém může ze stavu $x_{j,k}$ přejít pouze do některého ze stavů $x_{j+1,l}$ a navíc známe cenu tohoto přechodu $w(\{x_{j,k}, x_{j+1,l}\})$. Máme dále určen koncový stav x_z. Takový typ systému pak nazveme *mnohovrstvý rozhodovací systém* (obrázek 3.2.1.). Naším úkolem bude nyní najít takovou strategii, která nám zajistí přechod ze stavu x_0 do stavu x_z s co nejmenšími náklady (co nejlevnější). Zajímá nás tedy *minimální cesta.* V některých případech nás také zajímá, k jaké nejhorší variantě (v tomto případě nejdražší cestě) může dojít, tzn. budeme hledat *maximální cestu* ze stavu x_0 do stavu x_z. Minimální či maximální cesta se nazývá *kritická cesta.*



Obrázek 3.2.1. Mnohovrstvý rozhodovací systém**.**

Při hledání kritických cest je důležitý následující princip:

3.2.1 Věta – Princip optimálnosti. *Je-li* $x_0, x_{1,k(1)},\ldots, x_z$ *minimální cesta z* x_0 *do* x_z*, potom pro každé* $n < m$ *je* $x_{n,k(n)}$, $x_{n+1,k(j+1)}$,..., $x_{m,k(m)}$ *minimální cesta z* $x_{n,k(n)}$ *do* $x_{m,k(m)}$.

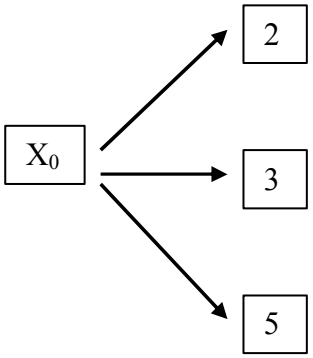
Důkaz. Je zřejmé, že kdyby existovala kratší cesta z $x_{n,k(n)}$ do $x_{m,k(m)}$, pak by přirozeně existovala kratší cesta z x_0 do x_z.

Q.E.D.

4. Hledání nejkratší a nejdelší cesty

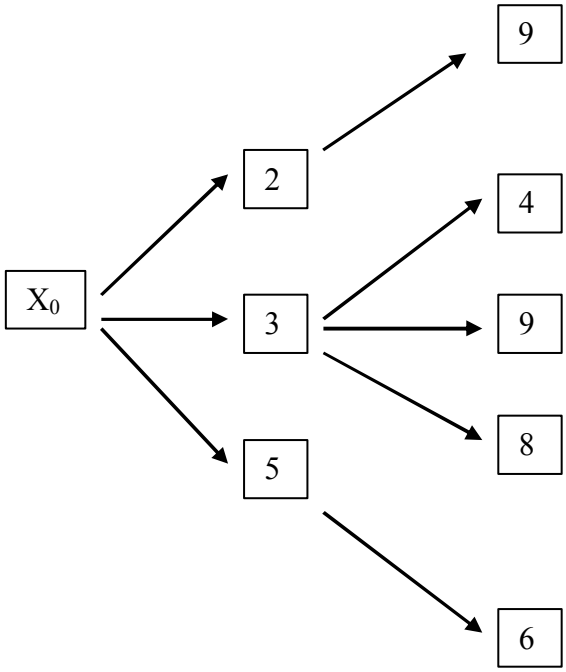
4.1 Princip optimálnosti v konkrétním grafu

Nyní demonstrujeme použití principu optimálnosti k nalezení cesty s minimální cenou z x_0 do x_z v obrázku 3.2.1. z předchozí kapitoly. Začneme ve startovním vrcholu x_0 a vezmeme v úvahu všechny hrany, které z x_0 vycházejí. Vrcholům, do kterých se z vrcholu x_0 po jednom kroku dostaneme, přiřadíme číslo, které odpovídá ceně tohoto přechodu. Tedy vrcholu x_{10} přiřadíme číslo 2 (tj. číslo $w(x_{10})$ přiřazené vrcholu x_{10} je rovno váze hrany x_0 a x_{10} neboli $w(\{x_0,x_{10}\})$), vrcholu x_{11} přiřadíme číslo 3 a vrcholu x_{12} přiřadíme číslo 5 (Obrázek 4.1.1.)



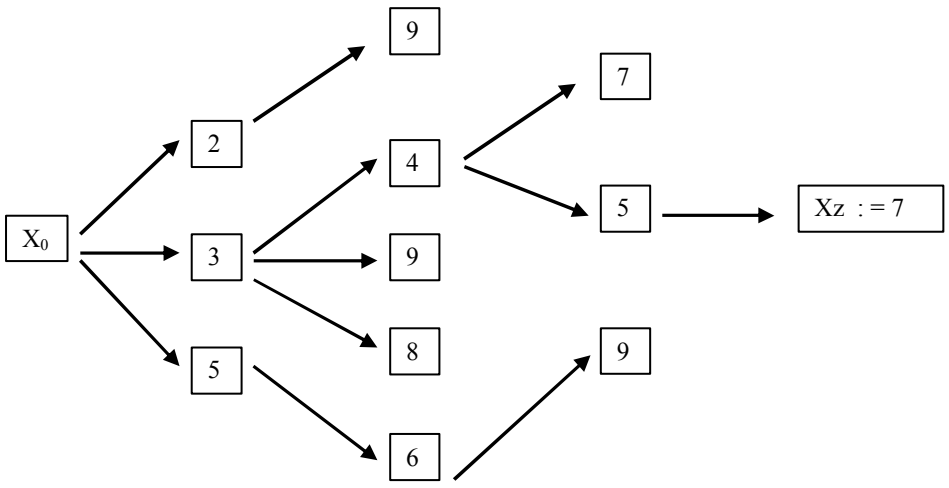
Obrázek 4.1.1. První fáze

Dále budeme chtít určit hrany mezi vrcholy $x_{1,k}$ a $x_{2,l,}$ které můžeme uvažovat pro minimální cestu. Nyní použijeme princip optimálnosti a to tak, že každému vrcholu $x_{2,l}$ přiřadíme číslo, které udává minimální hodnotu součtu vah $w(x_{1,k})$ + $w(\{x_{1,k}, x_{2,l}\})$, kde $w(x_{1,k})$ je číslo přiřazené vrcholu $x_{1,k}$ v předchozím kroku a $w(\{x_{1,k}, x_{2,l}\})$ je cena přechodu z vrcholu $x_{1,k}$ do vrcholu $x_{2,l}$. Určíme hrany, pro které platí $w(x_{2,l}) = w(x_{1,k}) + w(\{x_{1,k}, x_{2,l}\})$. Jen tyto hrany pak budeme nadále uvažovat pro hledání minimální cesty. Toto proběhne v druhé fázi (Obrázek 4.1.2.).



Obrázek 4.1.2. Druhá fáze.

Tento postup opakujeme, až dostaneme hodnotu $w(x_z)$ pro konečný vrchol x_z, do kterého jsme se chtěli dostat. Tato hodnota nám pak dává délku nejkratší cesty mezi vrcholy x_0 a x_z. Chceme-li najít konkrétní cestu, postupujeme od koncového vrcholu pozpátku. Hledáme-li maximální délku, postupujeme stejně, akorát v každém kroku přiřazujeme následujícímu vrcholu maximální hodnotu součtu $w(x_{j,k})$ + $w(\{x_{j,k}, x_{j+1,l}\})$. Závěrečná fáze je znázorněna na obrázku 4.1.3.



Obrázek 4.1.3. Třetí fáze.

4.2 Dijkstrův algoritmus

Tento algoritmus je efektivní pro hledání nejkratší cesty mezi dvěma vrcholy v ohodnoceném grafu. Hrany musí být ohodnoceny kladnými reálnými čísly. Důležité je, že graf nesmí obsahovat cyklus záporné délky (k čemuž by při záporném ohodnocení hran mohlo snadno dojít). Kdyby tomu tak bylo, mohli bychom zkonstruovat cestu s libovolně malou délkou. Zabývejme se nyní principem algoritmu.

Nechť máme dán graf $G = (V, E)$ s kladným ohodnocením hran. Dále budeme mít zadaný počáteční vrchol x. Pro každý vrchol v grafu G spočítáme vzdálenost z vrcholu x do vrcholu v, tj. $d_G(x, y)$. Postupujeme následujícím způsobem:

1. krok: Pro každý vrchol $v \in V$ zavedeme číselnou proměnnou $p(v)$, která je momentálním odhadem vzdálenosti $d_G(x, v)$. Vrcholu x přiřadíme trvalou hodnotu $p(x) := 0$ a všem ostatním vrcholům $v \in V$ přiřadíme dočasnou hodnotu $p(v) := \infty$

2.krok: Zavedeme množinu C jako množinu těch vrcholů $v \in V$, u kterých hodnota $p(v)$ nebyla prohlášena za definitivní. Na začátku procházení grafu volíme $C := V$.

3.krok: Testujeme: platí-li pro všechna $v \in C$: $p(v) := \infty$, algoritmus končí, jinak pokračujeme 4. krokem.

4.krok: Vytvoříme nyní množinu D těch vrcholů $v \in V$, pro které je $p(v)$ nejmenší mezi všemi vrcholy z C. Všechny vrcholy z množiny D odstraníme z množiny C a přepočítáme hodnoty $p(z)$ pro jejich sousední vrcholy.

5. krok: Pro všechny hrany $\{v, z\} \in E$ takové, že $v \in D$ a $z \in C$, porovnáme hodnoty $p(z)$ a $p(v) + w(\{v, z\})$. Pokud platí: $p(v) + w(\{v, z\}) < p(z)$, potom to znamená, že přes vrchol v vede do vrcholu z kratší cesta, než jsme prozatím znali, tudíž změníme hodnotu $p(z)$ na $p(v) + w(\{v, z\})$. Až vyčerpáme všechny takové hrany, vrátíme se na 3.krok.

Výše popsaný algoritmus skončí, pokud platí $p(v) = \infty$ pro všechna $v \in C$, tedy buď C je prázdná nebo jsou v ní jen vrcholy, které nejsou dosažitelné cestou z počátečního vrcholu x. Zde počítáme nejkratší cesty z vrcholu x do všech ostatních cestou dosažitelných vrcholů grafu a mezi nimi i cestu z vrcholu x do vrcholu y. Nemusíme však počítat minimální cesty do všech ostatních vrcholů, stačilo-by, kdyby se algoritmus zastavil v případě, kdy je našemu koncovému vrcholu y přiřazena trvalá hodnota. V praxi se místo počátečního přiřazení ∞ vrcholům používá nějaká velká číselná hodnota, která je zaručeně větší než délka nejdelší cesty, tedy např. pro graf s n hranami hodnota $\sum_{i=1}^{n} w(e_i) + 1$.

Časová složitost Dijkstrova algoritmu je $O(n^2+m)$.[1)], kde n značí počet vrcholů grafu a m počet hran.

Důkaz správnosti algoritmu. Nejdříve ukážeme indukcí, že pro každý vrchol $v \in V$ platí v každém kroku algoritmu nerovnost $p(v) \geq d_G(x, v)$. Na začátku platí $d_G(x, x) = 0$ a $d_G(x, v) \leq \infty$ pro všechny vrcholy $v \in V$. Ve chvíli, kdy se změní hodnota $p(z)$ pro nějaký vrchol $z \in V$, tak existuje nějaký vrchol $v \in D$ tak, že platí a $p(v) + w(\{v, z\}) < p(z)$. Podle našeho indukčního předpokladu pro vrchol v platí nerovnost $p(v) \geq d_G(x, v)$. K nejkratší cestě z vrcholu x do vrcholu v délky $d_G(x, v)$ jsme přidali hranu $\{v, z\}$, a tím jsme získali cestu z vrcholu x do vrcholu z, která má délku $d_G(x, v) + w(\{v, z\})$. Pro hodnotu p(z) po provedení změny dostáváme:

$$p(z) = p(v) + w(\{v, z\}) \geq d_G(x,v) + w(\{v, z\}) \geq d_G(x,z)$$

V další fázi budeme chtít dokázat, že po ukončení algoritmu bude platit $p(v) \leq d_G(x,v)$ pro všechny vrcholy $v \in V$.

Dokážeme nejdříve, že platí-li pro všechny různé konečné hodnoty vzdáleností vrcholů grafu od vrcholu x vztah $0 = d_1 < d_2 < \ldots < d_m < \infty$

---

1) Při použití tzv. binární haldy v implementaci lze dosáhnout časové složitosti $O((m+n) \log n)$.

a pokud označíme množiny

$$H_i = \{v \in V; d_G(x,v) = d_i\} \quad i = 1,2, ..., m$$

pak se algoritmus provede právě m-krát a pro každé i platí po i -tém vykonání 4. kroku $D = H_i$, $min \{p(v); v \in C\} = d_i$ a $V \setminus C = \bigcup_{j=1}^{i} H_j$.

Důkaz této části provedeme indukcí. Pro $i = 1$ je tvrzení zřejmé. Zvolíme nyní libovolné $i > 1$, libovolný vrchol $y \in H_i$ a budeme předpokládat, že tvrzení platí pro každé $j < i$. Je-li $d_G(x, y) = d_i < \infty$, potom existuje cesta $(x = v_0, e_1, v_1, ..., v_t, e_{t+1}, v_{t+1} = y)$ z vrcholu x do vrcholu y délky d_i. Potom platí $d_G(x, v_t) \leq d_i - d_G(x, v_t) < d_i$, tudíž vrchol $v_t \in H_j$ pro nějaký index $j < i$. Indukční předpoklad nám dává, že vrchol v_t je v j -tém opakování algoritmu prvkem množiny D a taktéž v j -tém opakování platí $p(v_t) = d_G(x, v_t)$. Dále v $(j+1)$ –ním opakování platí $p(y) = d_G(x, v_t) + d_G(x, v_t) = d_G(x, y) = d_i$, a tedy v i -tém opakování je $p(y) = d_i$. Vhledem k indukčnímu předpokladu je v i -tém opakování algoritmu $p(v) = d_G(x, v) < d_i$ pro každý vrchol $v \in V \setminus C^{(i-1)}$. Podle předchozího dostáváme, že v i -tém opakování algoritmu platí $p(v) \geq d_G(x, v) > d_i$ pro každý vrchol $v \in \bigcup_{i<j\leq k} H_j$. Proto platí, že $min \{p(v); v \in C\}$ je v i -tém opakování rovno d_i a množina D obsahuje v i -tém opakování stejné vrcholy jako množina H_i.

Q.E.D.

4.3 Využití heuristiky v Dijkstrově algoritmu

Dijkstrův algoritmus může zjevně prohledávat mnoho vrcholů zbytečně. Budeme-li např. chtít spočítat délku nejkratší cesty z Prahy do Ostravy, pustíme z Prahy „vlnu“, kde budeme zjišťovat i délku cesty z Prahy do Chebu, která je ale na první pohled zbytečná. Budeme tedy chtít hledat nekratší cestu správným směrem. Nechť je dán graf G s nezáporným ohodnocením hran w a cílový vrchol c. Heuristikou pak budeme rozumět takovou funkci h: $V(G) \rightarrow [0, \infty)$, která splňuje následující podmínky:

i) $h(c) = 0$

ii) pro každou hranu $e = \{u,v\} \in E$ platí $| h(u) - h(v) | \leq w(e)$

Pod hodnotou *h(v)* si můžeme představit nějaký dolní odhad vzdálenosti vrcholu *v* od vrcholu *c*. Zabýváme-li se pak v nějaké úloze dopravním spojením, pak *h(v)* může označovat např. vzdálenost mezi vrcholy *v* a *c* vzdušnou čarou. V Dijkstrově algoritmu bychom tedy mohli počítat s hodnotami vrcholů $p'(v) := p(v) + h(v)$.

4.4 Floyd-Warshallův algoritmus

Tento algoritmus je určen pro orientované grafy neobsahující záporné cykly (máme-li nezáporně ohodnocené hrany, pak je splnění této podmínky zaručeno). Nachází nejkratší orientované cesty pro každou dvojici vrcholů a ze všech cest, které mají stejnou délku, vybírá takovou, která má nejmenší počet hran.

Pro spočítání vzdálenosti mezi každou dvojicí vrcholů v grafu můžeme použít na každý vrchol zvlášť Dijkstrův algoritmus. Výhodou Floyd-Warshallova algoritmu vůči tomuto postupu je, že všechny vzdálenosti počítá přímo a je rychlejší. Další výhodou je relativně jednoduchá implementace.

Popis algoritmu:

Máme dán orientovaný graf s *n* vrcholy bez záporných cyklů.

1. Krok. Očíslujeme v libovolném pořadí vrcholy grafu čísly *1, 2, ..., n*. Vytvoříme matici *n x n*. Vzdálenost pak budeme počítat v *n* iteracích.

2. Krok. Na sestavíme matici D^0 vzdáleností, tj. prvek [*x, y*] matice D^0 udává, jaká je délka hrany z vrcholu *x* do vrcholu *y,* pokud existuje. Na diagonále jsou zřejmě samé nuly a mezi dvojicí vrcholů, kde neexistuje hrana, je přiřazeno nekonečno.

3. Krok. V *i* -té iteraci spočítáme matici D^i. Potom hodnota $D^i[x, y]$ udává délku nejkratší cesty z vrcholu *x* do vrcholu *y*, která však prochází pouze vrcholy *{1, 2, ..., i}*. (Tedy matice D^1 nám dá vzdálenost všech vrcholů s možností využití jednoho daného prostředníka, matice D^2 vzdálenost při možném použití dvou daných prostředníků atd.). Pro všechna $x = 1, ..., n$ a pro všechna $y = 1, ..., n$ provedeme následující:

$$D^i[x, y] := min \{D^{i-1}[x, y], D^{i-1}[u, i] + D^{i-1}[i, v]\}$$

4.Krok. V poslední n -té iteraci pak dostaneme matici D^n, o které víme, že bude obsahovat již všechny vzdálenosti, protože cesty mezi jednotlivými vrcholy smí procházet prostřednictvím všech vrcholů.

Výše popsaným postupem získáme délky nejkratších cest mezi jednotlivými vrcholy grafu, neříká nám však nic o tom, přes které vrcholy ty které cesty vedou. Chceme-li mít tuto dodatečnou informaci, lze to zařídit tak, že si v průběhu algoritmu budeme pamatovat pro každou dvojici vrcholů x a y nejvyšší číslo vrcholu, přes který nejkratší cesta vede (tzn. poslední volbu vrcholu, která vedla ke zkrácení cesty). Z toho pak lze nejkratší cestu snadno zkonstruovat rekurzí. Časová složitost Floyd-Warshallova algoritmu je $O(n^3)$.

4.5 Bellman – Fordův algoritmus

Bellman – Fordův algoritmus slouží k nalezení nejkratší cesty v orientovaném grafu s libovolným ohodnocením hran. Dokonce dokáže detekovat cyklus záporné délky. Nejprve však budeme předpokládat, že graf neobsahuje záporný cyklus. Budeme mít dán startovní vrchol s a budeme chtít najít nejkratší cestu z vrcholu s do všech ostatních vrcholů grafu. Pro všechny vrcholy v si budeme uchovávat hodnotu p(v), která vyjadřuje délku nějaké cesty z vrcholu s do vrcholu v (tedy se nemusí jednat o tu nejkratší). Tedy hodnota $p(v)$ je vždy rovna nebo větší délce nejkratší cesty z s do v.

V průběhu algoritmu se budeme snažit náš odhad $p(v)$ vylepšovat. Budeme používat tzv. *update hrany*. Předpokládejme, že se z vrcholu s dokážeme dostat do vrcholu x po cestě délky $p(x)$ a z vrcholu x vede do vrcholu y hrana $\{x, y\}$. Prodloužíme-li cestu z s do x o hranu $\{x, y\}$, pak určitě můžeme najít cestu z s do y délky $p(x) + w(\{x, y\})$. Další postup závisí na tom, zda je hrana $\{x, y\}$ *korektní* či nikoli.

4.5.1 Definice. Hrana $\{x, y\}$ je korektní, jestliže pro ni platí nerovnost $p(y) \leq p(x) + w(\{x, y\})$. Platí-li opak, pak je hrana $\{x, y\}$ nekorektní a odhad $p(y)$ můžeme vylepšit.

Vylepšení odhadu tedy znamená:

$p(y) := min \{p(y), p(x) + w(\{x, y\}) \}$

Update hrany vždy nastaví správnou hodnotu $p(y)$ za předpokladu, že x je předposlední vrchol na nejkratší cestě do y a hodnota $p(x)$ už byla správně nastavena. Také nemůže nikdy zmenšit hodnotu $p(v)$ pod délku nejkratší cesty do vrcholu v pro libovolný vrchol grafu.

Uvažujme nejkratší cestu $(s, e_1, v_1, e_2, v_2, ..., v_{k-1}, e_k, v_k)$ z vrcholu s do vrcholu v_k. Postupné updatování hran $e_1, ..., e_k$ zajišťuje, že $p(v_k)$ bude udávat délku nejkratší cesty z s do v_k Zabývejme se nyní pořadím updatování jednotlivých hran. Každá cesta mezi dvěma vrcholy má nejvýše $n-1$ hran. Z toho plyne, že postačí v $(n-1)$ iteracích updatovat všechny hrany grafu. Vytvoříme tedy posloupnost hran,

$$e_1,e_2,..., e_m, e_1, e_2, ..., e_m, e_1,... \quad ...e_m$$

která má celkem m x $(n-1)$ hran a posloupnost $n-1$ hran je v ní obsažena jako podposloupnost, tedy každou posloupnost $n-1$ hran updatujeme ve správném pořadí.

Popis algoritmu:

1.Krok. Startovnímu vrcholu s přiřadíme $p(s) := 0$, všem ostatním vrcholům v grafu přiřadíme $p(v) := \infty$.

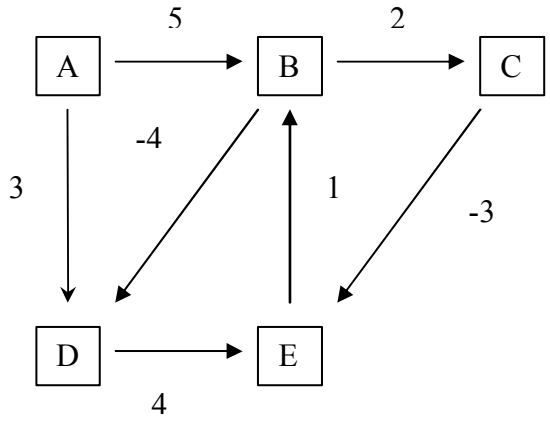
2.Krok. V nejvýše $(n-1)$ iteracích updatujeme všechny hrany grafu. Přitom v každé iteraci zjišťujeme, zda byla alespoň nějaká hrana nekorektní. Pokud tomu tak není, pak jsou všechny hrany grafu korektní, v další iteraci by se žádný update neprovedl, tudíž nemá smysl pokračovat dále a algoritmus končí.

Podle 2.kroku je zřejmé, že algoritmus může skončit po méně než $(n-1)$ iteracích. Je tomu tak, protože nejkratší cesta obsahuje ve většině případů mnohem méně než $(n-1)$ hran, vyplatí se tedy průběžná kontrola korektnosti všech hran.

Ukázali jsme činnost Bellman-Fordova algoritmu pro grafy neobsahující záporný cyklus. V takovém případě skončí algoritmus ve chvíli, kdy jsou

všechny hrany korektní. Pokud by však byl v grafu obsažen záporný cyklus, existovala by stále nekorektní hrana a mohli bychom odhady délek neustále „vylepšovat“ (resp. stále snižovat jejich hodnotu) pohybováním se po záporném cyklu. Stačí provést Bellman-Fordův algoritmus bez ohledu na korektnost hran, tj. provedeme přesně *(n-1)* iterací. Po skončení algoritmu otestujeme, jsou-li všechny hrany korektní. Pokud ano, nalezli jsme nejkratší cestu, pokud ne, graf obsahuje cyklus záporné délky a nejkratší řešení neexistuje. Časová složitost Bellman-Fordova algoritmu je *O(mn)*, kde *m* je počet hran grafu a *n* počet jeho vrcholů.

4.5.2 Příklad. Máme dán orientovaný ohodnocený graf (obrázek 4.5.1.). Budeme chtít najít nejkratší cestu z vrcholu *A* do všech ostatních vrcholů. Pomocí Bellman-Fordova algoritmu budeme daný problém řešit a sledovat po každé iteraci hodnoty *p(v)* v tabulce 4.5.2..



Obrázek 4.5.1. Vyšetřovaný graf

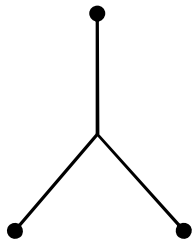
Vrchol	Iterace				
	0	**1**	**2**	**3**	**4**
A	0	0	0	0	0
B	∞	5	5	5	5
C	∞	7	7	7	7
D	∞	3	1	1	1
E	∞	7	4	4	4

Tabulka 4.5.2. Vzdálenosti ostatních vrcholů od vrcholu A po jednotlivých iteracích.

5. Problém minimální kostry

5.1 Definice pojmu kostra a formulace problému

Představme si množinu odběratelů vody (reprezentovanou v grafu množinou vrcholů). Je potřeba vymyslet takovou síť vodovodů, která bude zásobovat každého odběratele a zároveň bude nejlacinější. Dále budeme požadovat, aby se vodovody nevětvily mimo odběratele, tj. aby nenastala situace na obrázku 5.1.1.



Obrázek 5.1.1. Větvení mimo vrcholy

Řešení tohoto úkolu nás vede na řešení problému minimální kostry. Jako první se tímto problémem zabýval na začátku 20.století Otakar Borůvka při řešení nejkratšího spojení měst.

Budeme předpokládat graf s ohodnocením w, kde $w : E \rightarrow \mathbf{R}$. Takový graf nazýváme síť. Potom můžeme problém formulovat jako:
Pro souvislý graf G = (V, E) jehož hrany mají nezáporné ohodnocení nalezněme souvislý podgraf (V, E´) tak, že výraz

$$w(E') = \sum_{e \in E'} w(e)$$

je minimální.

Je zřejmé, že mezi řešeními této úlohy existuje vždy strom. Máme-li všechny hrany ohodnoceny kladně, pak je řešením jenom strom. V následujícím textu čerpám především ze zdroje [7].

5.1.1 Definice. Nechť $G = (V, E)$ je graf. Potom se libovolný strom ve tvaru (V, E'), kde $E' \subseteq E$, nazývá *kostrou* grafu G.

Pomocí předchozí definice můžeme náš problém formulovat jako nalezení kostry s nejmenší možnou hodnotou pro daný souvislý graf s nezáporným ohodnocením hran. Minimální kostru grafu budeme značit $T = (V, E')$ Podívejme se nyní na některé základní algoritmy pro řešení tohoto problému.

5.2 Kruskalův hladový algoritmus

Úlohu budeme řešit pro souvislý graf $G = (V, E)$, který má n vrcholů a m hran. Algoritmus vypadá následovně:

1.Krok. Seřadíme hrany $e_1, e_2, ..., e_m$ podle velikosti, tj. tak, aby platilo, že $w(e_1) \leq w(e_2) \leq ... \leq w(e_m)$.

2.Krok. Budeme konstruovat množiny $M_0, M_1, M_2, ... \subseteq E$, přičemž položíme $M_0 := \varnothing$

3.Krok. Budeme procházet hrany podle velikosti. Za předpokladu, že jsme již nalezli množinu M_{i-1}, spočítáme M_i takto:
$M_i := M_{i-1} \cup \{e_i\}$, pokud graf $(V, M_{i-1} \cup \{e_i\})$ neobsahuje kružnici, jinak $Mi := M_{i-1}$

4.Krok. Obsahuje-li M_i už všechny hrany nebo $i = m$, algoritmus končí. Dostáváme minimální kostru grafu $T = (V, M_k)$, kde M_k je množina, pro kterou se již algoritmus zastavil.

Důkaz správnosti algoritmu. Je zřejmé, že algoritmus v každém případě pro konečný graf skončí. V prvním kroku důkazu ukážeme, že T je kostrou grafu G. Zajisté graf T neobsahuje kružnici (plyne přímo ze způsobu vytváření jednotlivých množin M_i). Obsahuje-li graf T $n-1$ hran, pak nám věta 1.6.3 (5) dává, že T je strom, a tedy kostrou grafu G.
Dále budeme postupovat sporem. Předpokládáme, že T je nesouvislý a má méně než $n-1$ hran. Když je nesouvislý, má více než 1 komponentu. Zvolíme dva vrcholy c a d, které leží v různých komponentách grafu. Budeme uvažovat nějakou cestu z vrcholu

c do vrcholu d ($c=v_0, e_1, v_1, e_2, ..., v_k=d$) a označíme C tu komponentu T, která obsahuje vrchol c. Pro poslední index i takový, že v_i je obsaženo v komponentě C platí $i < k$ a v_{i+1} v komponentě C již neleží. Tedy hrana $\{v_i, v_{i+1}\}$ nepatří do T a musela být zamítnuta, protože tvořila s již vybranými hranami kružnici. Tedy kdybychom hranu $\{v_i, v_{i+1}\}$ přidali, graf $T+\{v_i, v_{i+1}\}$ by již obsahoval kružnici. Ale hrana $\{v_i, v_{i+1}\}$ spojuje různé komponenty, z čehož plyne, že $T+\{v_i, v_{i+1}\}$ kružnici obsahovat nemůže. Tím jsme ukázali, že T je kostra.

V druhé části důkazu budeme chtít ukázat, že T je minimální kostra. Vezmeme tedy jakoukoliv jinou kostru S a budeme chtít ukázat, že platí vztah $w(E(T)) \leq w(E(S))$, kde symbol $w(E(T))$ značí součet vah všech hran kostry T a analogicky symbol $w(E(S))$ součet vah všech hran kostry S. Uspořádáme hrany kostry T vzestupně podle vah a označíme $e_1', e_2', ..., e_{n-1}'$ nově setříděné hrany, pro které platí $w(e_1') \leq w(e_2') \leq ... \leq w(e_{n-1}')$. Stejně hrany kostry S také setřídíme podle vah a označíme $e_{s1}, e_{s2}, ..., e_{sn-1}$. Pokud ukážeme, že pro každý index $i = 1, ..., n-1$ platí $w(e_i') \leq w(e_{si})$, pak nám to dává, že T je minimální kostra.

Dokážeme sporem. Zvolíme nejmenší index i takový, že $w(e_i') > w(e_{si})$, *platí* $i > 1$. Budeme uvažovat množinu hran kostry T: $E' = \{e_1', ..., e_{i-1}'\}$ a množinu hran kostry S: $E_s = \{e_{s1}, ..., e_{si}\}$. Víme, že grafy (V, E') i (V, E_s) neobsahují kružnice (jsou to kostry). Pro spor budeme chtít ukázat, že můžeme najít hranu $e \in E_s$, pro kterou graf $(V, E' \cup \{e\})$ neobsahuje kružnici. Pak by totiž platilo $w(e) \leq w(e_{si}) \leq w(e_i')$ a bylo by vidět, že nebyl důvod hranu e během činnosti algoritmu zamítnout, tedy bychom hranu e mohli vybrat třeba místo hrany e_i'.

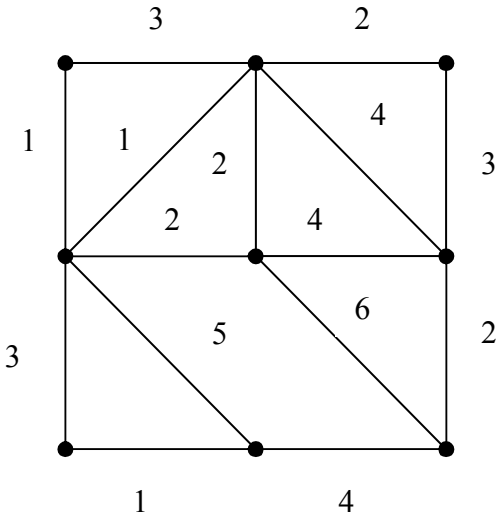
Budeme chtít ukázat, že jsou-li E' a E_s dvě množiny hran, kde (V, E_s) nemá kružnici, a platí-li $|E'| \leq |E_s|$, potom hrana $e \in E_s \setminus E'$ spojuje vrcholy dvou různých komponent grafu (V, E'). Provedeme rozklad množiny V na komponenty grafu (V, E'), tedy $V_1, V_2, ..., V_l$.

Je zřejmé, že platí $|E' \cap \binom{V_j}{2}| \geq |V_j| - 1$, a proto tedy $|E'| \geq n-l$. Protože E_s neobsahuje kružnice, dostáváme $|E_s \cap \binom{V_j}{2}| \leq |V_j| - 1$. Z toho plyne, že nějaká komponenta V_j obsahuje nejvýše $n-l$ hran z E_s. Dle předpokladu ale $|E'| \leq |E_s|$, z čehož plyne, že nějaká hrana $e \in E_s$ vede mezi dvěma komponentami.

Q.E.D.

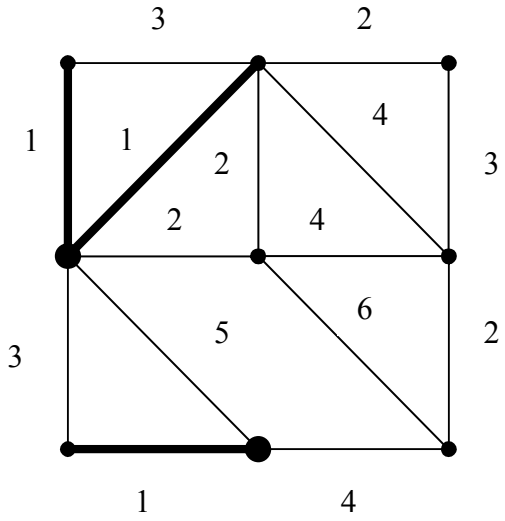
Časová složitost Kruskalova algoritmu provedeného v grafu s ještě nesetříděnými vahami je *O(m.log n),* kde *m* je počet hran a *n* je počet vrcholů grafu.

5.2.1. Příklad. Ukážeme činnost Kruskalova algoritmu.(Řešení příkladu zadaného v [13]). Na obrázku 5.2.1. máme výchozí graf, ve kterém budeme hledat minimální kostru.

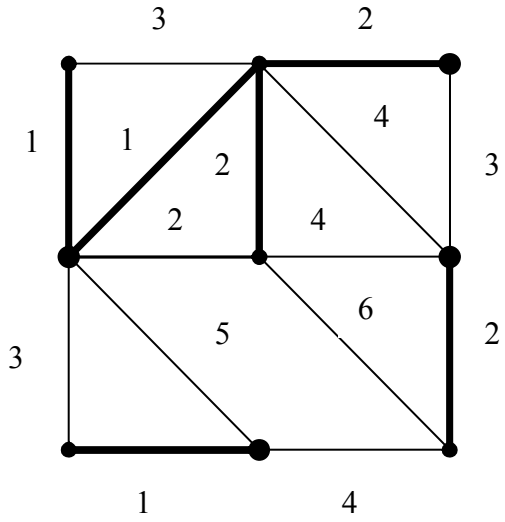


Obrázek 5.2.1. Graf pro hledání minimální kostry

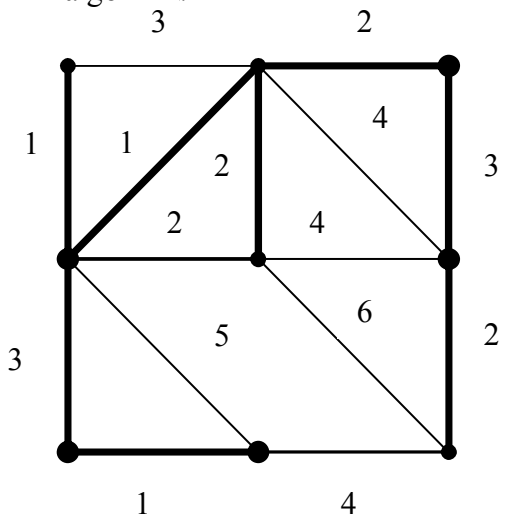
Nejdříve vybereme všechny hrany s váhou 1 tak, aby netvořily kružnici. Poté přidáme všechny takové hrany s vahou 2 tak, aby opět nevznikla kružnice. Takto budeme pokračovat až do chvíle, kdy projdeme všechny vrcholy. Další fáze algoritmu jsou demonstrovány na obrázcích 5.2.2., 5.2.3. a 5.2.4.



Obrázek 5.2.2. První fáze algoritmu



Obrázek 5.2.3. Druhá fáze algoritmu



Obrázek 5.2.4. Nalezená minimální kostra

5.3 Jarníkův – Primův algoritmus

Jedná se o další algoritmus na hledání minimální kostry v grafu. Tento algoritmus objevil poprvé v roce 1930 vynikající český matematik Vojtěch Jarník a v roce 1957 byl nezávisle objeven znovu americkým matematikem Robertem Primem.

Budeme uvažovat souvislý, nezáporně ohodnocený graf $G = (V, E)$ s n vrcholy a m hranami. Postupně vytvoříme množiny vrcholů V_o, V_1 ... $\subseteq$ V a množiny hran E_0, E_1,... $\subseteq$ E.

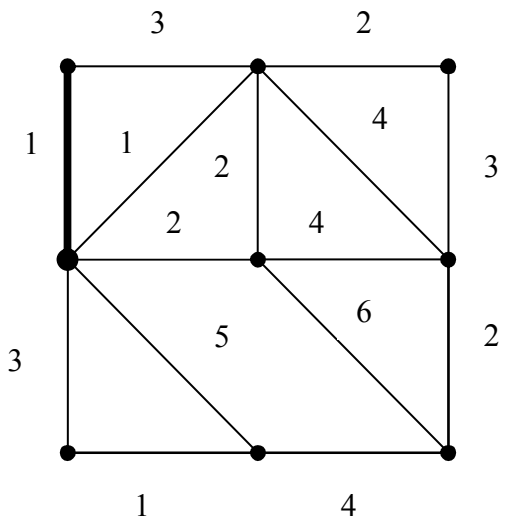
1.Krok. Položíme $E_0 := \emptyset$. Zvolíme libovolný vrchol $v \in V$ a přiřadíme $V_o := \{v\}$

2.Krok. Máme-li již vytvořené množiny V_{i-1} a E_{i-1}, pak se podíváme na množinu M_i všech hran $\{x_i, y_i\}$ takových, že vrchol $x_i \in V_{i-1}$ a $y_i \in V \setminus V_{i-1}$. Platí-li $M_i = \emptyset$, pak algoritmus končí (nelze již přidat žádnou hranu spojující různé komponenty). Jinak zvolíme takovou hranu $e_i \in M_i$, která má nejmenší váhu ze všech hran množiny M_i. Rozšíříme množinu vrcholů $V_i := V_{i-1} \cap \{y_i\}$ a množinu hran $E_i := E_{i-1} \cap \{e_i\}$. Tedy v každém kroku rozšíříme o nejlevnější hranu.

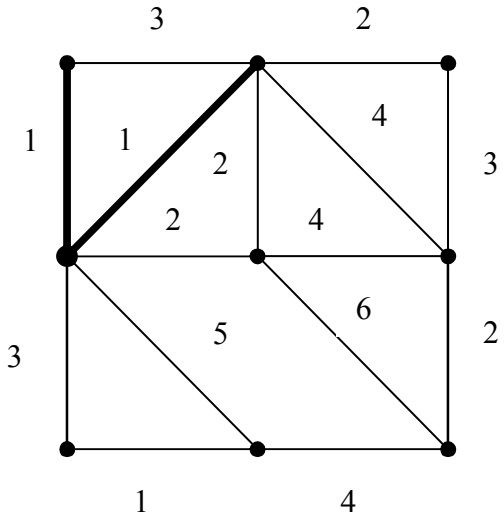
Důkaz správnosti algoritmu. Lze najít např. v knize [7].

Časová složitost Jarníkova – Primova algoritmu je $O(n+m)$, kde m je počet hran a n je počet vrcholů.

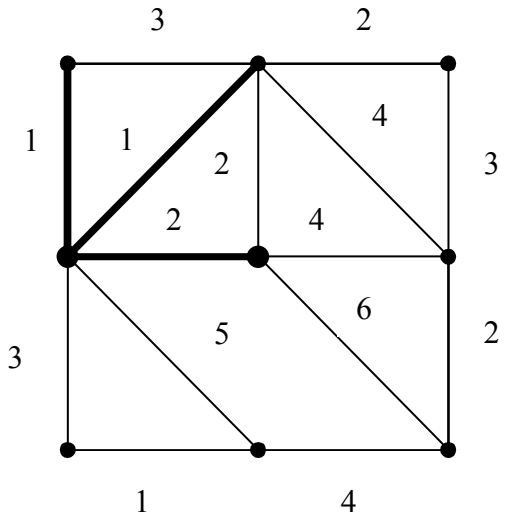
5.3.1 Příklad. Podíváme se nyní, jak aplikujeme Jarníkův-Primův algoritmus na následující graf na obrázku 5.3.1. Další přidávání hran podle pravidel algoritmu je znázorněno na obrázcích 5.3.2.-8.



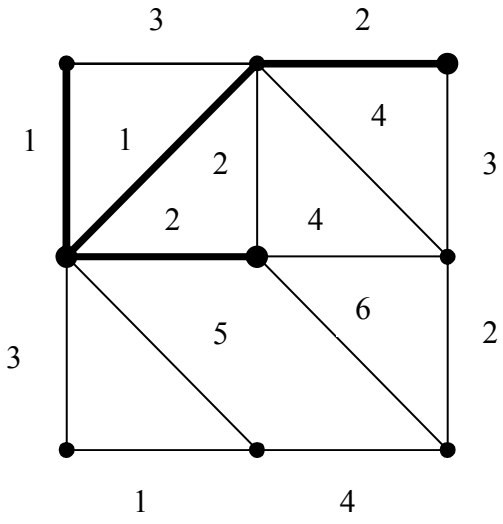
Obrázek 5.3.1. První fáze



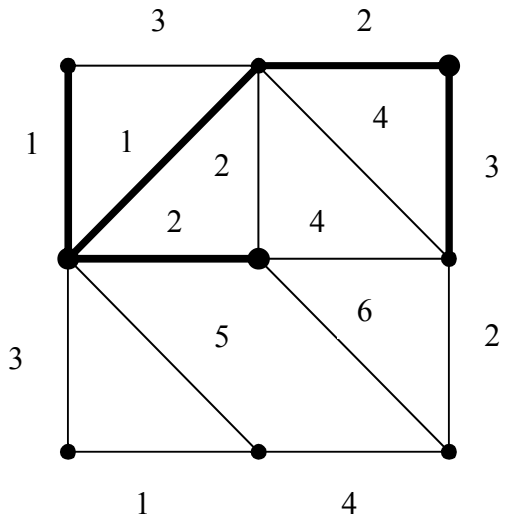
Obrázek 5.3.2. Druhá fáze



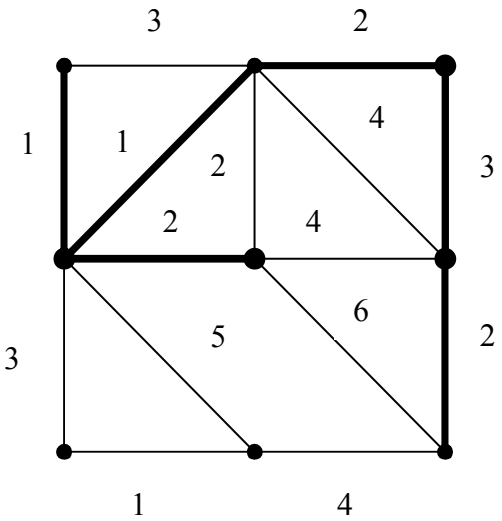
Obrázek 5.3.3. Třetí fáze



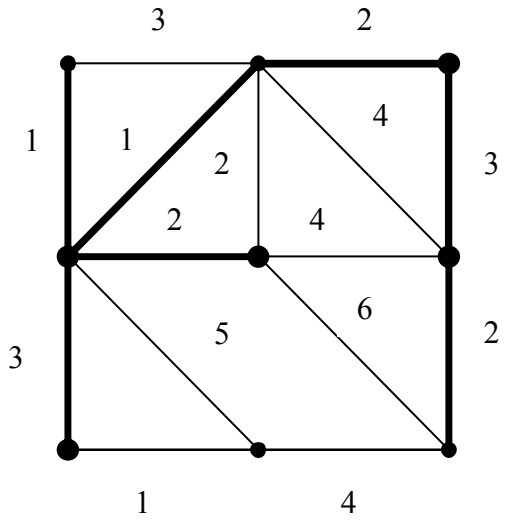
Obrázek 5.3.4. Čtvrtá fáze



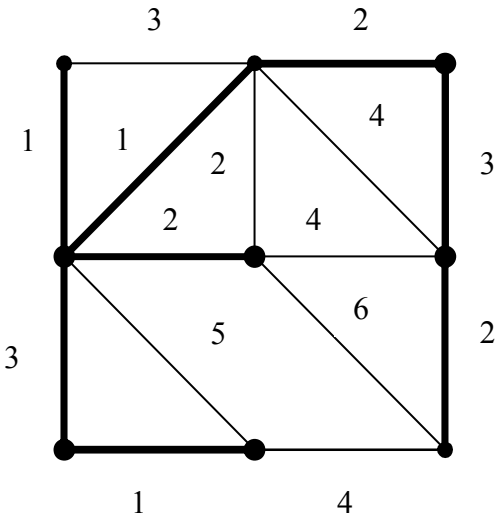
Obrázek 5.4.5. Pátá fáze



Obrázek 5.4.6. Šestá fáze



Obrázek 5.4.7. Sedmá fáze



Obrázek 5.4.8. Nalezená minimální kostra

5.4 Borůvkův algoritmus

Opět budeme hledat minimální kostru v ohodnoceném grafu $G = (V, E')$, tentokrát Borůvkovým algoritmem. Budeme navíc předpokládat, že jsou různé hrany ohodnoceny různými čísly. Postupně vytvoříme množiny hran $E_0, E_1, \ldots \subseteq E$.

1.Krok. Položíme $E_0 := \emptyset$.

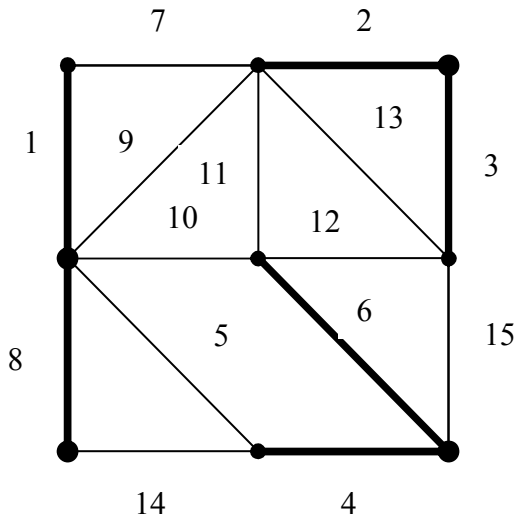
2.Krok. Máme-li již zkonstruovánu množinu E_{i-1} a rozklad množiny vrcholů V podle komponent souvislosti grafu $(V_1, V_2, \ldots, V_t)$, pak budeme pro každou třídu V_k našeho rozkladu hledat hranu $e_k = \{x_i, y_i\}$, pro kterou platí $x_i \in V_k$ a $y_i \notin V_k$ a jejíž váha má mezi hranami typu $\{x, y\}$, $x \in V_k$, $y \in V \setminus V_k$ nejmenší hodnotu. Pak $E_i := E_{i-1} \cup \{e_1, e_2, \ldots, e_k\}$.

3.Krok. V momentě, kdy graf (V, E_i) obsahuje jedinou komponentu, algoritmus končí.

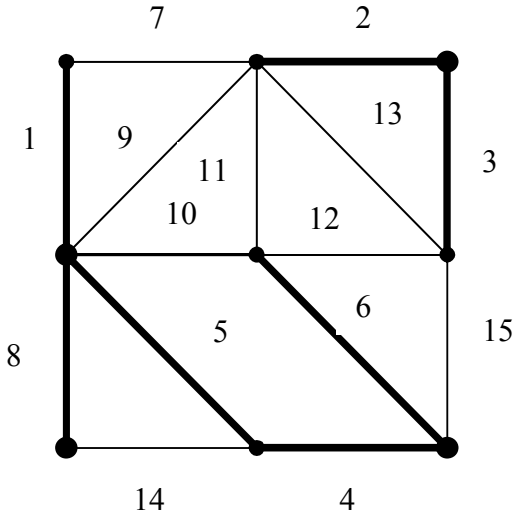
Důkaz. Je naznačen v [7].

Časová složitost Borůvkova algoritmu je $O(m.\log n)$, kde m je počet hran grafu a n počet jeho vrcholů.

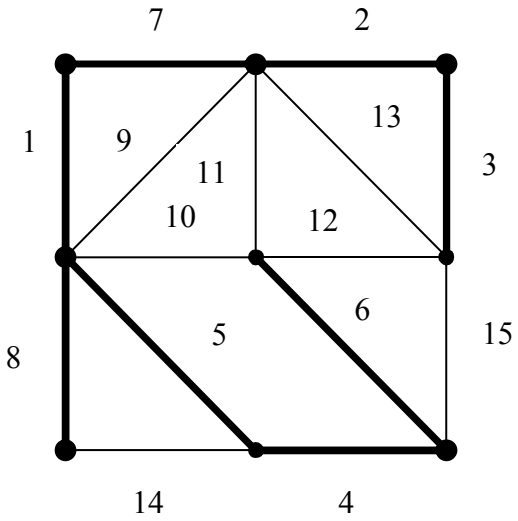
Příklad. Ukážeme použití Borůvkova algoritmu na obrázku 5.4.1,5.4.2 a 5.4.3. Máme dán graf s prostou funkcí w, tedy různým hranám máme přiřazeny různé váhy. Je vidět, že algoritmus končí již po 3 krocích, což je mnohem méně než u předchozích algoritmů.



Obrázek 5.4.1. První krok vytvoření komponent kostry



Obrázek 5.4.2. Postupné spojování komponent

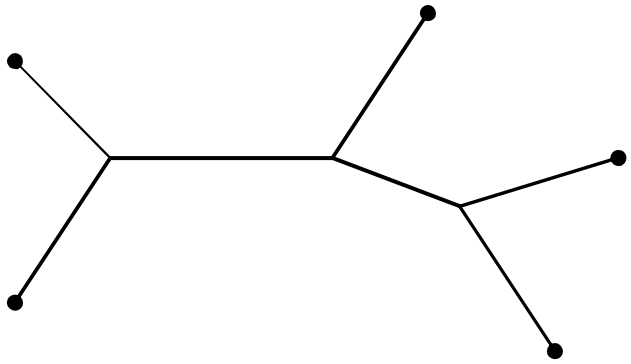


Obrázek 5.4.3. Nalezená minimální kostra

6. Steinerovy stromy

6.1 Motivační úloha

U motivační úlohy v kapitole o hledání minimální kostry jsme zakázali větvení vodovodu mimo spotřebitele. Kdybychom však tuto podmínku nepožadovali, spotřebovali bychom na propojení mnohem méně potrubí. Budeme vycházet z toho, že známe všechna možná místa větvení (viz obrázek 6.1.1.).



Obrázek 6.1.1. Steinerův strom

Jako příklad můžeme vzít situaci při odhrnování sněhu ze železniční sítě, kdy chceme propojit všechna města co nejlaciněji. Tedy kromě jednotlivých měst či vesnic se koleje mohou větvit na křižovatkách i mimo ně. V grafu železniční sítě pak můžeme najít dva druhy vrcholů – první typ představuje města a vesnice, které chceme navzájem propojit, druhým typem jsou pak křižovatky mimo města a vesnice (tzv. *Steinerovy vrcholy*). V optimálním případě můžeme dostat takové propojení měst a vesnic, ve kterém zůstane spousta nedostupných křižovatek. Tedy nemusíme použít všechny Steinerovy vrcholy, a to je základní odlišnost Steinerova stromu od kostry. Hledání optimálního Steinerova stromu patří mezi NP – těžké úlohy a není pro řešení tohoto problému zatím znám žádný polynomiální algoritmus. Umíme však v polynomiálním čase spočítat přibližné řešení pomocí aproximace.

Úloha zní: Máme daný neorientovaný graf $G = (V, E)$ s nezáporným ohodnocením hran w. V grafu G můžeme najít vrcholy dvou druhů, $V = P \cap S$, P jsou požadované vrcholy a S Steinerovy vrcholy. Nalezněme strom $T \subseteq G$ s minimální cenou, aby propojoval všechny požadované vrcholy P. Tedy minimalizujme výraz $\sum_{e \in T} w(e)$.

6.2 Řešení pomocí aproximačního algoritmu

Budeme se zabývat algoritmem, který platí pouze v grafech, kde váhy hran splňují trojúhelníkovou nerovnost (tedy pro trojici hran $\{x,y\}$, $\{y,z\}$, $\{x,z\}$ platí: $w(\{x,y\}) + w(\{y,z\}) \geq w(\{x,z\})$) . Algoritmus probíhá ve 3 krocích:

1. Krok. Vytvoříme pomocný graf G', který obsahuje jen požadované vrcholy P a je úplný. Váha hrany $\{x, y\} \in G'$ vyjadřuje délku nejkratší cesty z vrcholu x do vrcholu y v grafu G. V grafu G' určuje každá dvojic vrcholů hranu (z úplnosti), G' je metrickým uzávěrem grafu G. Zřejmě je cena Steinerova stromu v grafu G' stejná nebo vyšší než v grafu G.

2.Krok. Aproximací Steinerova stromu v grafu G' bude minimální kostra T' pro graf G'.

3.Krok. Některé hrany $\{x, y\}$ v kostře T' označují cestu mezi vrcholy x a y v grafu G. Každou hranu kostry T' nahradíme příslušnou cestou v grafu G, která má stejnou cenu jako nahrazovaná hrana (to plyne z toho, že graf G' je metrickým uzávěrem grafu G). V takto vzniklém grafu opět najdeme minimální kostru T (kvůli odstranění kružnic, které mohly vzniknout při nahrazování hran cestami). Minimální kostra T je pak aproximací Steinerova stromu v grafu G.

O přesnosti aproximace pojednává následující lemma.

6.2.1 Lemma. *Označíme-li OPT cenu optimálního Steinerova stromu T^* v grafu G, potom pro cenu stromu T nalezeného aproximačním algoritmem platí: $OPT \leq w(T) \leq 2.OPT$.*

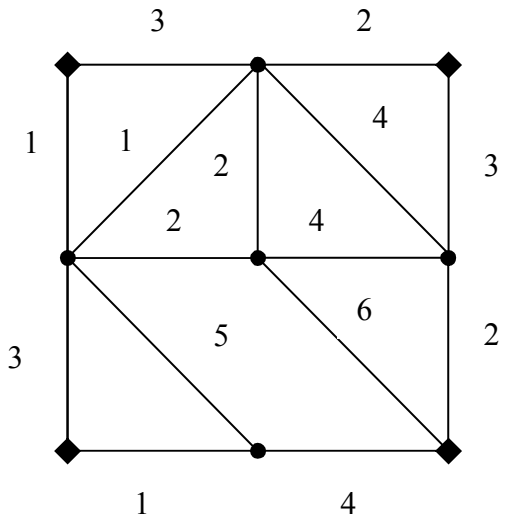
Důkaz. Víme, že nalezený Steinerův strom T je přípustným řešením, tedy platnost nerovnosti $OPT \leq w(T)$ je zřejmá.

Předpokládejme, že T^* je optimálním Steinerovým stromem v grafu G. Po zdvojení hran optimálního Steinerova stromu na nich nalezneme uzavřený eulerovský tah F. Pak pro cenu (délku) tahu F platí $w(F) = 2.OPT$. Budeme pak zkracovat úseky tahu F, které prochází přes Steinerovy vrcholy, a to tak, že každý úsek, který prochází přes hrany $\{x, y\}$ a $\{y, z\}$ se nahradí zkratkou procházející přes hranu $\{x, z\}$. Z předpokladů fungování algoritmu víme, že váhy hran splňují trojúhelníkovou nerovnost, tedy tímto zkrácením dostaneme levnější tah. Zkracování ukončíme ve chvíli, kdy všechny vrcholy tahu F budou patřit do množiny požadovaných vrcholů P.

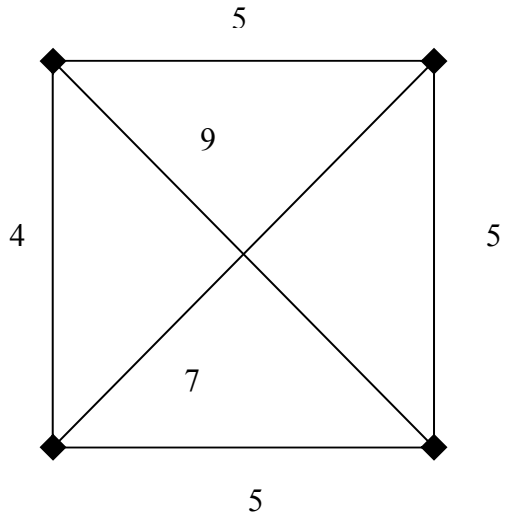
V další fázi budeme procházet eulerovský tah a pomocí zkratek vytvářet kružnici procházející všemi vrcholy (tzv. hamiltonovskou kružnici). Vždy budeme zkoumat, zda následující hrana tahu vede do již navštíveného vrcholu, a pokud tomu tak je, zkratkou se dostaneme do následujícího vrcholu tahu, který ještě navštíven nebyl. Tím cenu tahu jedině snížíme. Odstraníme-li nejdražší hranu takto vzniklé kružnice, dostaneme strom T'^*, pro jehož cenu platí $w(T'^*) \leq 2.\ OPT$. Tento strom tvoří nějakou kostru v grafu G', ale nemusí být nutně minimální kostrou, tedy cena minimální kostry T' v grafu G' je buď stejná nebo menší, z toho plyne, že $w(T') \leq 2.\ OPT$. Dle 3.kroku algoritmu z kostry $T' \subseteq G'$ vytvoříme levnější kostru T (podmnožina) G, tedy $w(T) \leq w(T') \leq 2.\ OPT$.

Q.E.D.

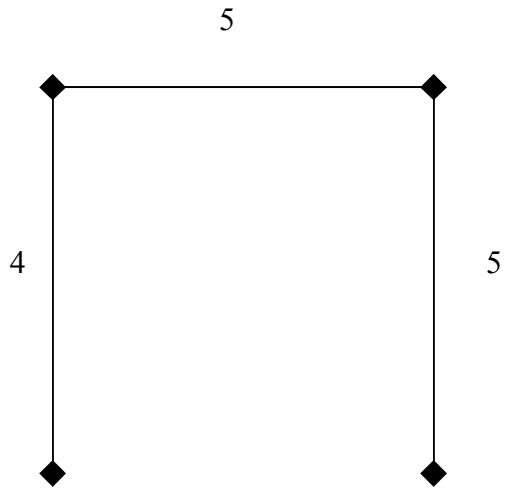
6.2.2. Příklad. Ukážeme činnost aproximačního algoritmu pro hledání minimálního Steinerova stromu. Vyjdeme z grafu na obrázku 5.2.1., kde Steinerovy vrcholy označíme jako tečku. Takový graf je pak na obrázku 6.2.1. Obrázek 6.2.2. představuje graf s požadovanými vrcholy a ohodnocení hran udává délku nejkratší cesty mezi jednotlivými vrcholy. Na obrázku 6.2.3. je ukázán druhý krok algoritmu, minimální kostra z grafu na obrázku 6.2.2. Minimální Steinerův strom nalezený aproximačním algoritmem je na obrázku 6.2.4.



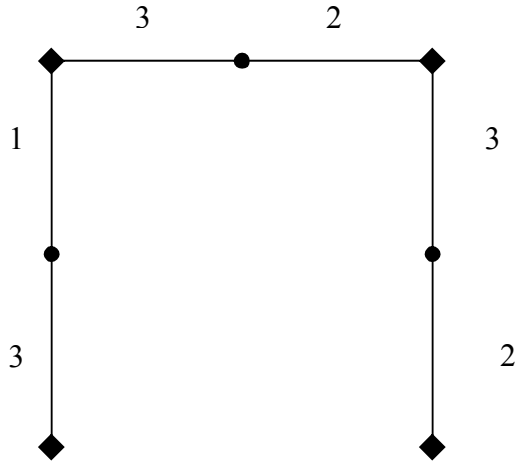
Obrázek 6.2.1.



Obrázek 6.2.2. První krok algoritmu, nejkratší cesty mezi požadovanými vrcholy



Obrázek 6.2.3. Minimální kostra



Obrázek 6.2.4. Minimální Steinerův strom nalezený aproximačním algoritmem

7. Optimalizační úlohy na kružnicích

V této části se budeme zabývat nalezením optimální (např. nejlevnější) kružnice za daných podmínek, např. navštívit každý vrchol či každou hranu alespoň jednou. Nejznámějšími úlohami jsou problém kropicího vozu či problém obchodního cestujícího.

7.1 Problém kropicího vozu neboli čínského pošťáka

Máme dán souvislý neorientovaný graf $G = (V, E)$ s nezáporným ohodnocením hran. Naším úkolem je nalézt takový uzavřený sled $(v_o, e_1, v_1, ..., e_n, v_0)$, který obsahuje každou hranu grafu G a pro který je součet vah hran, kterými jsme prošli, minimální, tj. minimalizujeme výraz $\sum_{i=1}^{n} w(e_i)$. Tento problém nazýváme problémem kropicího vozu či čínského pošťáka. Graf představuje ulice ve městě a jejich ohodnocení jejich délku či náklady na projití ulicí, naším úkolem je nalézt takovou jízdu (sled) kropicího vozu, aby provedl kropení všech ulic s nejmenšími možnými náklady. Analogicky u úlohy čínského pošťáka[1)] budeme požadovat projití všech ulic v uzavřeném sledu (vyjde z pošty a vrátí se zpět na poštu) tak, aby při roznášení pošty ušel co nejméně.

Máme dán souvislý neorientovaný graf $G = (V, E)$ s nezáporným ohodnocením hran. Budeme chtít nalézt graf $G' = (V', E')$, který splňuje následující podmínky:

1) $E \subseteq E'$
2) Graf (V', E') je eulerovský
3) Výraz $\sum_{e \in E'} w(e)$ je minimální mezi všemi grafy, které splňují předchozí dvě podmínky.

---

1) Tuto úlohu navrhl a řešil čínský matematik Kwan v roce 1962, proto úloha *čínského* pošťáka.

Je-li graf G eulerovský, pak existuje takový sled, že každou ulicí projdeme právě jednou a náklady jsou dány jako $\sum_{e \in E} w(e)$. Použijeme algoritmus pro nakreslení grafu jedním tahem:

1.Krok. Zvolíme libovolný vrchol $v_0 \in V$ a vytvoříme množinu $C_0 = v_0$.

2.Krok. Předpokládejme, že máme již zkonstruován tah $C_i = v_0, e_1, v_1, \ldots, e_i, v_i$. Další hranu $e_{i+1} \in E$ volíme následovně: $v_i \in e_{i+1}$, a je-li to možné, aby graf $(V, E \setminus \{e_1,\ldots,e_i\})$ a graf $(V, E \setminus \{e_1,\ldots,e_i, e_{i+1}\})$, vzniklý přidáním hrany e_{i+1}, měly stejný počet komponent.

3. Krok. Není-li možné 2.krok provést, potom jsme vyčerpali všechny hrany grafu a C_n je hledaný eulerovský tah.

V případě, že graf $G = (V, E)$ není eulerovský, pak obsahuje sudý počet vrcholů lichého stupně. Budeme nejprve předpokládat, že graf obsahuj právě dva takové vrcholy, označíme Z množinu, která oba tyto vrcholy obsahuje, tedy $Z = \{v, v'\}$. V grafu G najdeme některým ze známých algoritmů nejkratší cestu z vrcholu v do vrcholu v'. Sestrojíme nový graf G' tak, že zdvojíme všechny hrany vyskytující se na nejkratší cestě z vrcholu v do vrcholu v'. Graf G' pak bude splňovat námi požadované vlastnosti.

Má-li graf 4 vrcholy lichého stupně, řešíme problém následujícím způsobem: označíme Z množinu vrcholů lichého stupně, $Z = \{x, y, z, v\}$, Zjistíme vzdálenosti (tj. cenu nejkratší cesty vzhledem k ohodnocení grafu) mezi dvojicemi těchto vrcholů, označíme je pak $d_G(x, y)$, $d_G(x, z)$, $d_G(x, v)$, $d_G(y, z)$, $d_G(y, v)$, $d_G(z, v)$, použijeme k tomu některého z algoritmů popsaných v kapitole 3. Pak zkoumáme, který z následujících výrazů nabyde minimální hodnoty:

$$d_G(x, y) + d_G(z, v)$$
$$d_G(x, v) + d_G(y, z)$$
$$d_G(x, z) + d_G(y, v)$$

Bez újmy na obecnosti můžeme předpokládat, že minimální hodnoty nabývá výraz $d_G(x, y) + d_G(z, v)$. Najdeme nejkratší cestu vrcholu x do vrcholu y, označíme ji

P(x, y), a nejkratší cestu z vrcholu *z* do vrcholu *v*, tu označíme *P(z, v).* Délky těchto nejkratších cest potom odpovídají číslům $d_G(x, y)$ a $d_G(z, v)$. Jako v minulém případě provedeme zdvojení hran, kterými tyto cesty procházejí.

Zbývá ukázat, že takto vzniklý graf vyhovuje podmínkám 1), 2), 3), které jsme požadovali. Budeme uvažovat libovolný graf $G' = (V', E')$, který těmto podmínkám vyhovuje. Definujeme graf $G'' = (V, E \setminus E)$ vyhovující požadovaným podmínkám. Jediné vrcholy lichého stupně v grafu G'' jsou vrcholy *x, y, z, v*. Princip sudosti nám dává, že všechny vrcholy musí ležet v jedné komponentě grafu G'' nebo dva náleží do jedné komponenty grafu G'' a zbylé dva jiné komponentě grafu G''. Budeme bez újmy na obecnosti předpokládat, že těmito dvojicemi jsou dvojice vrcholů *x, v* a *y, z*. Tedy v grafu G'' existuje cesta z vrcholu *x* do *v* a cesta z vrcholu *y* do *z*. Dostáváme:

$$\sum_{\substack{e \in E' \\ e \notin E}} w(e) \geq \sum_{\substack{e \in E' \setminus E \\ e \in P(x,v)}} w(e) \geq d_G(x, y) + d_G(z, v)$$

Délka sledu v grafu G' je tedy stejná nebo větší než délka sledu vzniklého pomocí zdvojení hran obsažených v cestách $P(x, y)$ a $P(z, v)$.

7.2 Hamiltonovské kružnice

Hamiltonovská kružnice je taková kružnice v grafu, která prochází všemi vrcholy. Nalezení hamiltonovské kružnice patří mezi tzv. *NP*-úplné problémy. Pokud mají vrcholy grafu $G = (V, E)$ všechny velký stupeň, pak graf určitě obsahuje kružnici, která prochází všemi vrcholy. Např. je-li stupeň každého vrcholu $d_G(v) = |V|-1$, pak je graf *G* hamiltonovský kdykoliv obsahuje alespoň 3 vrcholy.

Uvažujme graf *G*, který má *n* vrcholů. Předpokládejme, že pro nějaké dva vrcholy *v* a v' grafu *G* platí:

1) $\{v, v'\} \notin E$

2) $d_G(v) + d_G(v') \geq n$.

Graf vzniklý přidáním hrany $\{v, v'\}$ ke grafu G budeme značit $G_{v, v'} = (V, E \cup \{v, v'\})$. Uvedeme dále několik užitečných lemmat a tvrzení, která nebudeme dokazovat. (Jejich důkazy lze najít v [8]).

7.2.1 Lemma. *Ke každému grafu G= (V, E) můžeme najít právě jeden takový graf H = (V, F), který vznikl z grafu G přidáváním hran splňujících podmínky 1) a 2) a ke kterému už žádnou další hranu splňující obě tyto podmínky přidat nelze. Graf H se nazývá (Chvátalovým) uzávěrem grafu G a značí se c(G).*

7.2.2 Tvrzení. *Graf G je hamiltonovský, právě když jeho uzávěr c(G) je hamiltonovský.*

7.2.3 Důsledek. *Graf G je hamiltonovský, jestliže jeho uzávěr c(G) je úplný graf.*

7.2.4 Důsledek. *Každý graf obsahující n vrcholů, který splňuje podmínku: ,,netvoří-li vrcholy v a v´ hranu, pak $d_G(v) + d_G(v') \geq n$", je hamiltonovský graf.*

7.2.5 Důsledek. *Každý graf G s n vrcholy, který pro každý vrchol v splňuje podmínku $d_G(v) \geq n/2$, je hamiltonovský.*

Je-li graf $G = (V, E)$ hamiltonovský, pak je to možné vyjádřit i následujícím způsobem: můžeme najít takové pořadí vrcholů $V = \{v_1, v_2, ..., v_n\}$, že platí $d_G(v_i, v_{i+1}) = 1$ pro $i = 1,..., n$, kde index $n+1 = 1$.

7.2.6 Definice. Graf G se nazývá k-hamiltonovský, je-li souvislý a existuje-li takové pořadí jeho vrcholů $V = \{v_1, v_2, ..., v_n\}$, které splňuje podmínku $d_G(v_i, v_{i+1}) \leq k$ pro $i = 1,\ldots, n$, index $n+1 = 1$.

7.2.7 Tvrzení. *Pro každý souvislý graf platí, že je 3-hamiltonovský.*

7.3 Problém obchodního cestujícího

S tímto problémem se setkáme v praxi, kdy má obchodní cestující navštívit každé z n měst právě jednou, vrátit se do výchozího města a k tomu procestovat co nejméně.

Dostáváme se tedy k řešení problému nalezení nejkratšího hamiltonovského cyklu v grafu. Úlohu pak můžeme formulovat následujícím způsobem. Máme dán graf $G = (V, E)$ s nezáporným ohodnocením. Nalezněme takové cyklické uspořádání vrcholů $v_1, v_2, \ldots, v_n$, že výraz $\sum_{i=1}^{n} w(v_i, v_{i+1})$ nabývá minimální hodnoty.

Vzhledem k tomu, že není známé efektivní řešení problému hamiltonovské kružnice v grafu, není ani známo řešení problému obchodního cestujícího. Je však známé přibližné řešení problému pro nejčastěji v praxi se vyskytující případ. Budeme předpokládat graf s nezáporným ohodnocením hran a n vrcholy, kde váhy jednotlivých hran splňují trojúhelníkovou nerovnost (neboli pro každou trojici hran $\{x, y\}$, $\{y, z\}$, $\{x, z\}$ platí: $w(\{x, y\}) + w(\{y, z\}) \geq w(\{x, z\})$). Budeme-li definovat cyklické uspořádání vrcholů množiny V, je to stejné, jako bychom definovali kružnici o délce n v úplném grafu K se stejnou množinou vrcholů. Ukážeme, jak takovou kružnici najít. Můžeme postupovat takto:

1. Krok. Pomocí některého z algoritmů uvedených ve 4. kapitole nalezneme minimální kostru grafu K. Budeme předpokládat, že $F = (V, E)$ je touto minimální kostrou.

2. Krok. Z grafu $F = (V, E)$ vyrobíme jeho symetrickou orientaci[1)], orientovaný graf $O = (V, E')$, kde množina hran $E' = \{ \{x, z\}; \{x, z\} \in E\}$.

3. Krok. Je zřejmé, že orientovaný graf je souvislý a vyvážený (tedy pro každý vrchol v platí $deg^+(v) = deg^-(v) = deg(v)$). Uvažujme uzavřený orientovaný tah $T_0 = e_{0,1}, e_{0,2}, \ldots, e_{0,2n-2}$, který obsahuje všechny orientované hrany grafu $O = (V, E')$. [2)]

---

1) Z neorientovaného grafu vyrobíme graf orientovaný tak, že každou neorientovanou hranu nahradíme dvěma navzájem opačně orientovanými hranami.

2) Lze nalézt postupem pro kreslení grafu jedním tahem, viz také [4], v tomto případě (symetrická orientace) je to jednoduché, stačí jít po obvodu.

4. Krok. Z uzavřeného orientovaného tahu T_0 vyrobíme další uzavřené orientované tahy $T_i = e_{i,1}, e_{i,2}, \ldots, e_{i,2n-2-i}$ pomocí indukce. Předpokládejme, že tah $T_i = e_{i,1}, e_{i,2}, \ldots, e_{i,2n-2-i}$, $i < n-2$, známe. Z nerovnosti $2n-2-i > n$ dostáváme, že v uzavřeném tahu T_i existuje takový vrchol $v \in V$, že $\deg_{Ti}(v) \geq 2$. Zvolíme hrany $e_{i,k}(v', v)$ a $e_{i,k+1}(v, v'')$, které vrchol v obsahují. Tah T_{i+1} pak získáme tak, že hrany $e_{i,k}(v', v)$ a $e_{i,k+1}(v, v'')$ vynecháme a nahradíme jedinou hranou $\{v', v''\}$, tedy tah $T_{i+1} = e_{i,1}\ldots, e_{i,k-1}, \{v', v''\}, e_{i,k+2}, \ldots, e_{i,2n-2-i}$ je uzavřený a graf (V, T_{i+1}) je souvislý.

5.Krok. Pro T_{n-2} je graf (V, T_{n-2}) orientovanou kružnicí délky n (vyčerpali jsme všechny hrany, v kružnici o n vrcholech jich je $n - 2$).

6. Krok. Utvoříme množinu hran $E'' = \{ \{x, z\}; \{x, z\} \in T_{n-2} \}$. Potom graf (V, E'') tvoří hamiltonovskou kružnici.

Následující tvrzení určuje meze pro výsledek výše uvedeného aproximačního algoritmu.

7.3.1 Tvrzení. *Nechť OPT značí skutečnou délku nejkratší hamiltonovské kružnice v grafu K a nechť (V, E'') je hamiltonovská kružnice nalezená aproximačním algoritmem. Potom platí*

$$OPT \leq \sum_{e \in E''} w(e) \leq 2.OPT.$$

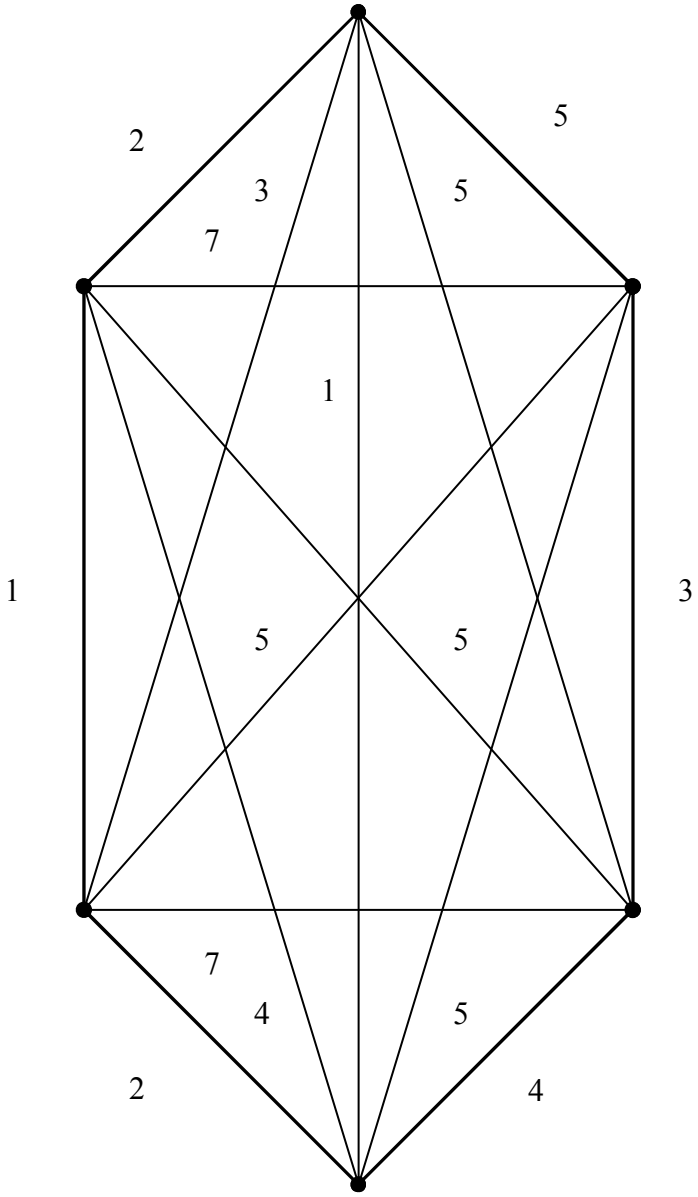
Důkaz. Nerovnost $OPT \leq \sum_{e \in E''} w(e)$ zřejmě platí, neboť algoritmus zcela jistě našel přípustné řešení. Zbývá jen dokázat nerovnost $\sum_{e \in E''} w(e) \leq 2.OPT$. Vynecháme-li z jakékoli hamiltonovské kružnice grafu K libovolnou hranu, dostaneme kostru. Minimální velikost kostry $F = (V, E)$ je pak menší než minimální délka hamiltonovské kružnice, tedy $\sum_{e \in E} w(e) \leq OPT$. Druhý krok algoritmu nám dává $\sum_{e \in T_0} w(e) = 2 \sum_{e \in E} w(e)$ (to plyne ze zdvojení hran při symetrické orientaci).

Vzhledem k tomu, že váhy hran splňují trojúhelníkovou nerovnost a po sobě následující tahy konstruujeme nahrazením dvou hran navazujících přes společný vrchol hranou kratší, dostáváme:

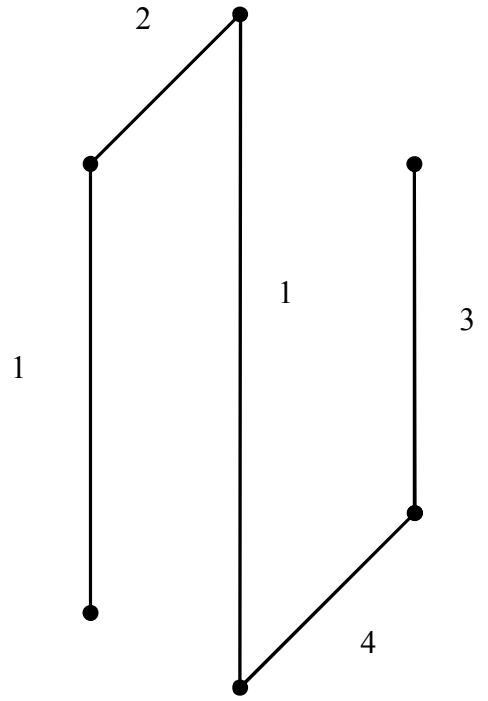
$$\sum_{e\in E''} w(e) \;=\; \sum_{e\in T_{n-2}} w(e) \;\le\; \sum_{e\in T_{n-3}} w(e) \le \ldots \le \sum_{e\in T_0} w(e) = 2\sum_{e\in E} w(e) \le 2.\mathrm{OPT}.$$

Q.E.D.

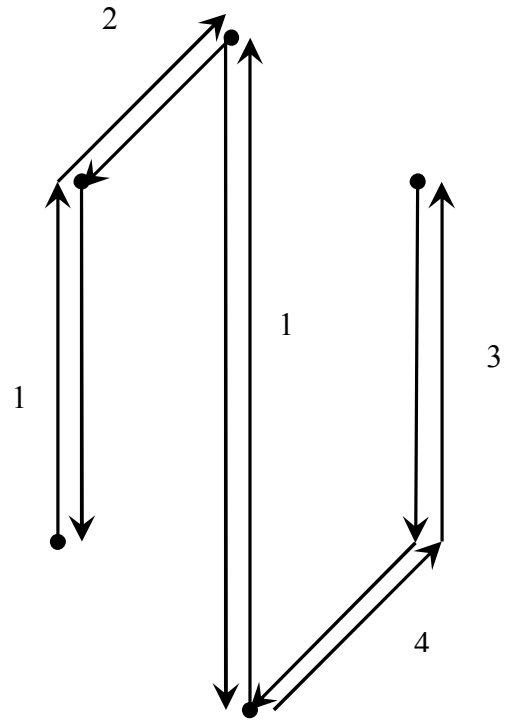
7.3.2. Příklad. Demonstrujeme činnost tohoto aproximačního algoritmu v následujícím grafu (obrázek 7.3.1.). Nejdeme nejlevnější kostru (obrázek 7.3.2.). Dále pokračujeme dle algoritmu (obrázky 7.3.3.,7.3.4., 7.3.5.,7.3.6.,7.3.7. a 7.3.8. – zmenšené oproti původnímu grafu na obrázku 7.3.1.). U uzavřených tahů T_1, T_2, T_3 a T_4 jsou na obrázcích hrany spojující vrcholy v' a v'' (po vynechání vrcholu v) vyznačeny čerchovaně.



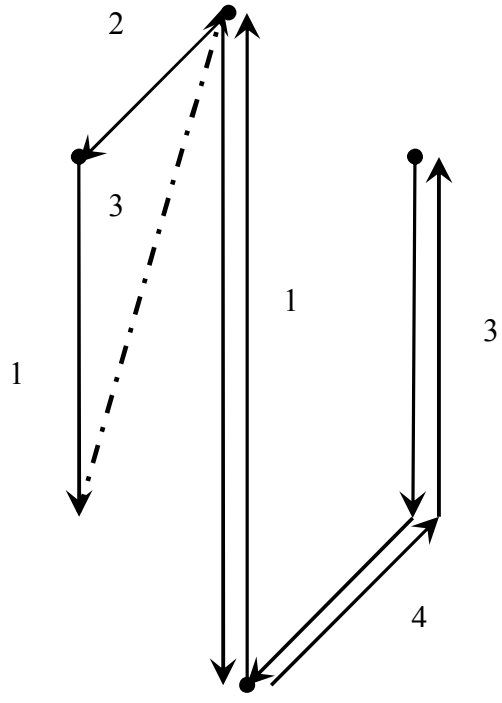
7.3.1. Obrázek. Výchozí graf pro hledání nejkratší hamiltonovské kružnice aproximačním algoritmem.



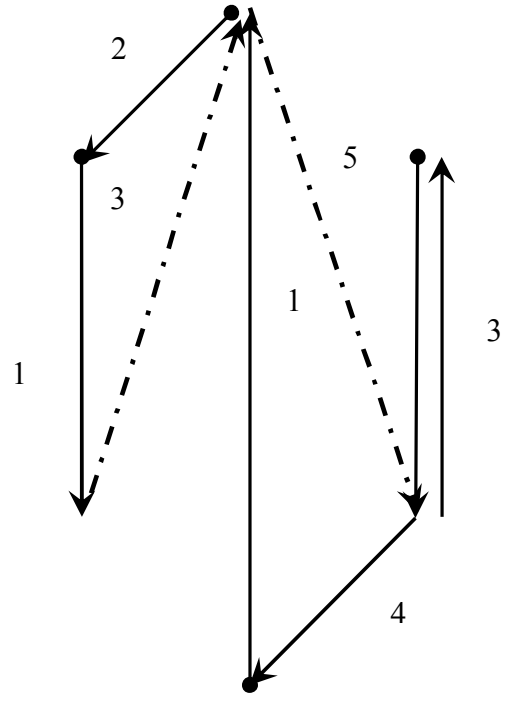
7.3.2. Obrázek. Nejlevnější kostra nalezená Kruskalovým algoritmem.



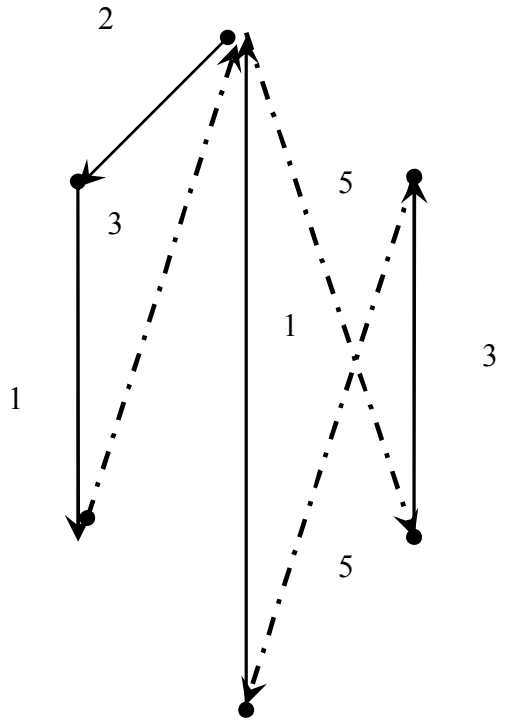
7.3.3. Obrázek. Symetrická orientace nejlevnější kostry.



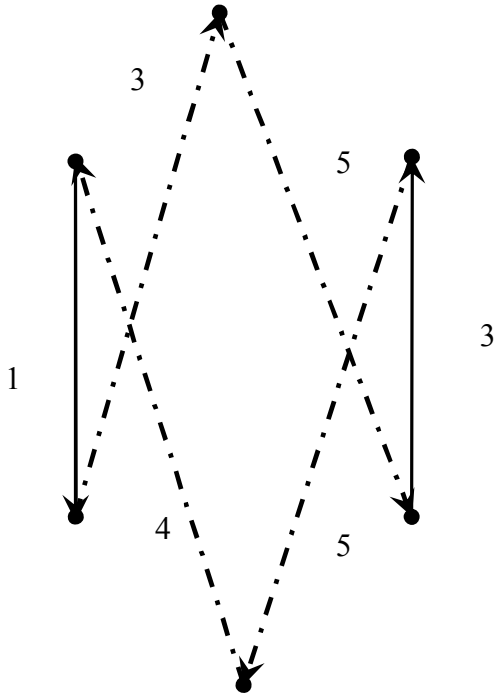
7.3.4. Obrázek. Uzavřený tah T_1.



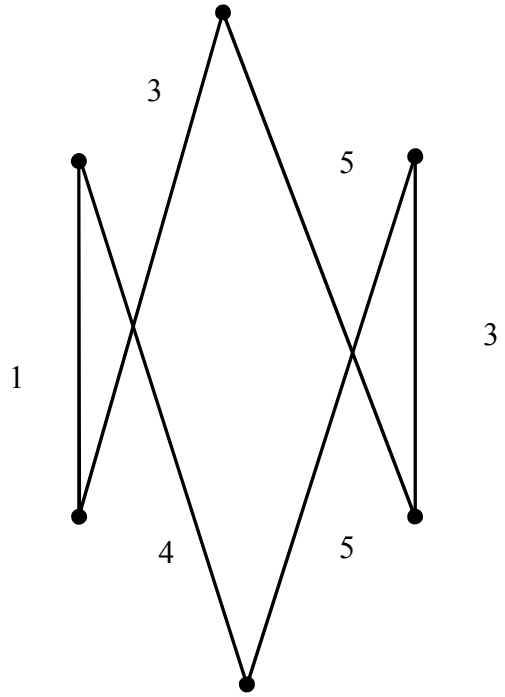
7.3.5.. Obrázek. Uzavřený tah T_2.



7.3.6. Obrázek. Uzavřený tah T_3.



7.3.7. Obrázek. Uzavřený tah T_4.



7.3.8. Obrázek. Výsledný nejkratší hamiltonovský cyklus nalazený aproximačním algoritmem.

8. Toky v sítích

Závěrečná kapitola se zabývá problematikou toků v sítích a základními algoritmy na její řešení. Motivací pro vznik této teorie byly dopravní problémy spojené s minimalizací přepravních nákladů a maximalizací využití přepravních kapacit. V praxi se jedná především o úlohy, kdy se snažíme optimalizovat nějaký tok (např. tok vody, elektřiny…) v odpovídajících sítích (vodovodní, elektrické…). Máme dánu síť potrubí, kde jednotlivé trubky mohou mít různé průměry a některé jsou jednosměrné. Kromě dvou míst, x a y, kudy může tekutina do sítě vtékat nebo odtékat, je síť uzavřená. Naším úkolem je určit, jaké maximální množství tekutiny může protékat sítí z x do y, neboli maximální tok. V následujících podkapitolách čerpám ze zdrojů [1], [2], [6], [9], [11] a [13].

8.1 Základní pojmy

8.1.1 Definice. *Sítí* nazveme uspořádanou pětici *(V, E, s, t, c),* kde *(V, E)* je orientovaný graf, $s \in V$ se nazývá zdroj (source), $t \in V$ se nazývá spotřebič neboli stok (target), $c : E \to \mathbf{R}^+$ je funkce, která udává kapacity jednotlivých hran.

8.1.2 Definice. Tok v síti je každá funkce $f : E \to \mathbf{R}^+$, která splňuje následující podmínky:

1) pro každou hranu $e \in E$ platí: $0 \leq f(e) \leq c(e)$

2) pro každý vrchol $v \in V \setminus \{s, t\}$ platí: $\sum_{\{x,v\} \in E} f(\{x,v\}) = \sum_{\{v,x\} \in E} f(\{v,x\})$

Budeme-li si síť představovat jako soustavu jednosměrných vodovodních trubek, kde u každé trubky známe její předepsanou maximální propustnost (kapacitu), udává tok, jak skutečně voda v trubkách proudí. Podle první podmínky z předchozí definice nemůže být tok záporný a trubkou nemůže téci více vody, než je její kapacita. Druhá podmínka říká, že co vteče do určitého vrcholu, to z něj opět vyteče ven. Je také známá jako zákon zachování toku, ve fyzice první Kirchhoffův zákon. Nahradíme-li v první podmínce předchozí definice v nerovnosti 0 nějakým číslem $l(e)$, nazveme tok, který splňuje $l(e) \leq f(e) \leq c(e)$ pro všechny hrany grafu, přípustným tokem.

Je-li dolní omezení toku u všech hran nulové, pak takovou síť nazveme transportní sítí.

8.1.3 Definice. Pro vrchol $v \in V$ označíme bilanci vrcholu $f(v)$. Je určena rozdílem mezi přítokem do vrcholu v a odtokem z vrcholu v neboli: $f(v) = \sum_{\{x,v\}\in E} f(\{x,v\}) - \sum_{\{v,x\}\in E} f(\{v,x\})$

8.1.4 Definice. Velikostí toku $|f|$ (lze značit i $w(f)$) rozumíme množství toku protékající ze zdroje do spotřebiče. Ze zákona zachování toku ho spočítáme jako bilanci vrcholu t nebo jako zápornou bilanci vrcholu s, neboli $|f| = f(t)$.

8.1.5 Definice. Řez mezi zdrojem s a stokem t v síti (V, E, s, t, c) je taková množina hran $R \subseteq E$, že v síti $(V, E \setminus R, s, t, c)$ neexistuje žádná orientovaná cesta ze zdroje s do stoku t. Kapacita řezu je $c(R) = \sum_{e\in R} c(e)$

Budeme se zabývat hledáním maximálního toku v daném grafu. Pro jeho hledání nahradíme každou neorientovanou hranu dvojicí orientovaných hran jdoucích proti sobě se stejnými kapacitami, jako byla kapacita původní hrany.

8.1.5 Definice. *Rezerva* hrany $e = \{v, w\}$ při toku f je definována jako $r(e) = (c(e) - f(e)) + f(e')$, kde $e' = \{w, v\}$.

V předchozí definici předpokládáme, že ke každé hraně existuje hrana opačná. Pokud tomu tak není, dodefinujeme ji a přiřadíme jí nulovou kapacitu. Hranu orientovanou směrem od zdroje ke spotřebiči nazýváme dopřednou, opačně orientovaná hrana se nazývá zpětná. *Rezerva po směru hrany e* je množství toku, které můžeme poslat po směru hrany, její velikost je $c(e) - f(e)$. *Rezerva proti směru hrany e* je množství toku, které můžeme poslat proti směru hrany, její velikost je $f(e)$. Cestou budeme myslet posloupnost $(v_0, e_1, v_1, e_2, \ldots, v_{m-1}, e_m, v_m)$, kde $v_1,\ldots,v_m$ jsou navzájem různé vrcholy uvažované sítě, a pro každé i je $e_i = (v_{i-1}, v_i)$ nebo $e_i = (v_i, v_{i-1})$.

8.1.6 Definice. Cesta se nazývá *nasycená,* pokud pro nějakou hranu $e_i = (v_{i-1}, v_i)$ orientovanou po směru je $f(e_i) = c(e_i)$ nebo pro nějakou hranu $e_i = (v_i, v_{i-1})$ orientovanou proti směru je $f(e_i) = 0$.

Cesta, která není nasycená, se nazývá nenasycená. Tok se nazývá nasycený, když je každá cesta ze zdroje do stoku nasycená. Jako (*s, t*)-cestu označíme cestu vedoucí ze zdroje *s* do spotřebiče *t*.

8.1.7. Definice. Nenasycená cesta ze zdroje do stoku se nazývá *vylepšující cesta (augmenting path).*

8.2 Maximální tok a minimální řez

8.2.1 Věta. *V každé síti existuje maximální tok a minimální řez.*

Důkaz. Uvedeme jen náznak důkazu. Existence minimálního řezu plyne z toho, že řezů je v každé síti konečně mnoho. K důkazu existence maximálního toku použijeme, že toky tvoří kompaktní množinu v prostoru všech ohodnocení hran a velikost toku je lineární funkce. Lineární funkce je spojitá, a proto nabývá na kompaktní množině svého maxima.

Q.E.D.

8.2.2 Věta. *Pro každý tok f a každý řez R je velikost toku* $w(f) \leq c(R)$*, neboli maximální tok je nejvýše roven minimálnímu řezu.*

Důkaz. Lze najít v [11].

V roce 1956 byla objevena Fordem a Fulkersonem následující hlavní věta o tocích:

8.2.3 Věta (o maximálním toku a minimálním řezu). *Tok f je maximální, právě když je nasycený. Ke každému maximálnímu toku f existuje řez R takový, že* $w(f) = c(R)$.

Důkaz. V první části dokážeme sporem, že maximální tok je nasycený. Nechť je tedy *f* maximální tok, ale nenasycený, tedy existuje vylepšující cesta *P*. Najdeme vylepšení toku ve směru ze zdroje do stoku $\lambda_1 = min\{c(e) - f(e);\ e \in$ P, orientovaná po směru$\}$ a vylepšení proti směru $\lambda_2 = min\{f(e);\ e \in P,$ orientovaná proti směru$\}$.

Položíme $\lambda_P = min\ \{ \lambda_1, \lambda_2\}$. Protože je cesta P nenasycená, platí $\lambda_P > 0$. Z toku f vyrobíme tok f' takto:

$f'(e) = f(e) + \lambda_P$ $\quad e \in P$, orientovaná po směru

$f'(e) = f(e) - \lambda_P$ $\quad e \in P$, orientovaná proti směru

$f'(e) = f(e)$ $\quad e \notin P$

Vzhledem k volbě λ_P tok f' nepřekročí kapacity hran a $f'(e)$ nikdy nenabyde záporné hodnoty. V jednotlivých vrcholech se λ_P přičte i odečte zároveň, zákon zachování toku tedy platí i nadále. Tok f' je korektně definovaný a navíc pro něj platí $w(f') = \sum f'(\{s,v\}) - \sum f'(\{v,s\}) = w(f) + \lambda_P$, tedy tok f' je lepší než tok f, tudíž tok f nebyl maximální, což je spor.

V druhé části ukážeme, že je-li tok f nasycený, pak je maximální. Množinu všech takových vrcholů v, pro které existuje nenasycená cesta ze zdroje do vrcholu v, označíme D. Protože je tok nasycený, máme $s \in D$, ale $t \notin D$. Označme $S(A, B)$ množinu hran, které vedou z části sítě A do části sítě B, neboli $S(A, B) = \{(x, z)$; $x \in A, y \in B, (x,y) \in E\}$. Pro každou hranu $e \in S(D, V \setminus D)$ platí $f(e) = c(e)$ a pro každou hranu $e \in S(V \setminus D, D)$ platí $f(e) = 0$. Velikost toku dostaneme následovně:

$w(f) = f(D, V \setminus D) - f(V \setminus D, D) = c(D, V \setminus D) - 0 = c(D, V \setminus D)$

Z věty 8.2.2 víme, že $w(f) \leq c(R)$ pro každý tok f i řez R. V předchozím kroku jsme dokonce našli takový řez R, že platí rovnost $w(f) = c(R)$, tedy tok f je největší možný. Touto konstrukcí jsme ale dostali i řez R, pro který platí rovnost $w(f) = c(R)$,

Q.E.D.

8.2.4 Důsledek. *Tok f je maximální, právě když neexistuje vylepšující cesta pro tok f.*

8.3 Ford-Fulkersonův algoritmus

Důkaz věty 8.2.3 dává návod, jak hledat maximální tok v síti. Na počátku zvolíme nějaký tok, který budeme v následujících krocích vylepšovat, a to tak, že budeme hledat nějakou nenasycenou cestu. Přitom budeme používat hrany, pro které platí $f(e) < c(e)$, a hrany, po kterých teče něco v protisměru. Po této nenasycené cestě pošleme to, co je nejvíce možné, přičemž tok v našem směru pak simulujeme odečtením od toku v protisměru. Popišme nyní Ford-Fulkersonův algoritmus v několika krocích:

1. Krok. Začneme nulovým tokem, tedy položíme $f(e) = 0$ pro všechny hrany $e \in E$

2. Krok. Existuje-li vylepšující cesta P, najdeme λ_P jako v důkazu věty 8.2.3 Vylepšíme cestu P. Dokud existuje nějaká nenasycená cesta, opakujeme tento krok.

3. Krok. Neexistuje-li již nenasycená cesta, našli jsme dle Důsledku 8.2.4 maximální tok f.

Podívejme se nyní na konečnost a jiné vlastnosti algoritmu. Jsou-li v síti kapacity hran racionální, pak je Ford-Fulkersonův algoritmus konečný, tedy existuje maximální tok. Tento maximální tok je také racionální. Jsou-li kapacity celočíselné, pak je maximální tok nalezený tímto algoritmem také celočíselný.

Vynásobíme-li všechny racionální kapacity jejich společným jmenovatelem, dostáváme celočíselné kapacity, s nimiž dále počítáme. V takovém případě je v algoritmu λ_P vždy alespoň 1, tedy v každém kroku se velikost toku zvýší alespoň o 1. Algoritmus najde nasycený tok, který je maximální, a určitě skončí. V průběhu činnosti algoritmu se pouze sčítá, tedy máme-li celočíselné kapacity, vznikne ve výsledku také celé číslo. V případě racionálních kapacit převedených na celočíselné vynásobením společným jmenovatelem se výsledek vydělí tímto společným jmenovatelem, čímž dostaneme opět číslo racionální. V síti s iracionálními kapacitami Ford-Fulkersonův algoritmus skončit nemusí!

Časová složitost Ford-Fulkersonova algoritmu může být pro některé sítě a špatné volby zlepšujících cest příliš velká. Edmonds a Karp časovou složitost tohoto algoritmus zlepšili tím, že za vylepšující cestu vzali vždy cestu s nejmenším počtem hran. Takto vylepšený algoritmus má časovou složitost $O(m^2n)$.

8.4 Dinicův algoritmus

Ve Ford-Fulkersonově algoritmu jsme přičítali zlepšující cesty, avšak tento postup lze zefektivnit, budeme-li přičítat rovnou zlepšující toky. Budeme hledat zlepšující toky takové, aby bylo snadné je najít a aby původní tok dostatečně obohatily.

8.4.1 Definice. Blokující tok je takový tok, kdy každá orientovaná *(s, t)*-cesta je nasycená, tedy obsahuje alespoň jednu nasycenou hranu.

Blokující tok je takový tok, který by našel Ford-Fulkersonův algoritmus, pokud bychom neuvažovali rezervy v protisměru.

8.4.2 Definice. Síť rezerv je taková síť, která vznikne z původní sítě tak, že ponecháme vrcholy a hrany a kapacity hran budou určeny rezervami v původní síti.

Popišme nyní činnost Dinicova algoritmu:

1. Krok. Začneme tokem f všude nulovým.

2. Krok. Vylepšíme tok f pomocí zlepšujících toků, iterativně (vnější cyklus):

2.1. Sestrojíme síť rezerv G^* a budeme s ní v dalších krocích pracovat.

2.2. V síti G^* najdeme nekratší *(s, t)* – cestu. Pokud žádná neexistuje, algoritmus končí.

2.3. Provedeme tzv. pročištění sítě, v síti G^* necháme jen vrcholy a hrany na nejkratších *(s, t)*-cestách.

2.4. V pročištěné síti najdeme blokující tok f_B, přičemž na začátku položíme $f_B = 0$, a budeme přidávat *(s, t)*-cesty (vnitřní cyklus):

2.4.1. Najdeme *(s, t)*-cestu a pošleme po ní co největší tok.

2.4.2. Smažeme nasycené hrany.

2.4.3. Dočistíme síť.

2.5. Vylepšíme tok f podle f_B.

Při provedení bodu 2.4.2. musíme dávat pozor, protože smazáním hran mohou vzniknout „slepé uličky“, což znečistí síť. Zde je způsob, jak provádět čištění a dočišťování sítě:

1) Vrcholy v síti rozvrstvíme podle vzdálenosti od zdroje s.

2) Cesty delší než do vrstvy spotřebiče t zařízneme.

3) Vytvoříme frontu vrcholů, pro které platí $deg^+ = 0$ nebo $deg^- = 0$. Z fronty budeme vrcholy vybírat a mazat včetně hran, které vedou do nich nebo z nich. Pokud při této operaci některým vrcholům klesl jeden ze stupňů na 0, přidáme je do fronty.

Algoritmus se může zastavit v bodě 2.2., neexistuje-li již žádná vylepšující cesta. Podle Ford-Fulkersonova algoritmu víme, že v této chvíli je nalezen maximální tok, tedy můžeme skončit.

Časová složitost Dinicova algoritmu je $O(n^2m)$.

8.5 Metoda tří Indů

Dinicův algoritmus lze zrychlit, použijeme-li efektivnější algoritmus pro hledání blokujícího toku ve vrstevnaté síti, který objevili Malhotra, Kumar a Maheshwari. Při použití tohoto efektivnějšího algoritmu na hledání blokujícího toku v Dinicově algoritmu dosáhneme časové složitosti $O(n^3)$.

8.5.1 Definice. Součet rezerv všech hran vstupujících do vrcholu v označíme $r^+(v)$, součet rezerv všech hran vystupujících z vrcholu v označíme $r^-(v)$ a přiřadíme $r(v) = min(r^+(v), r^-(v))$.

Předpokládejme, že máme dánu vrstevnatou síť. Pustíme nejdříve nulový tok a budeme se ho snažit vylepšit. Budeme si vždy udržovat rezervy hran $r(e)$ (stačí rezervy po směru hrany, hledáme blokující tok) a rezervy vrcholů $r^+(v)$, $r^-(v)$ a $r(v) = min(r^+(v), r^-(v))$.

V každé iteraci algoritmu provedeme následující kroky: nalezneme vrchol s nejnižší hodnotou *r(v)* a zvětšíme stávající tok tak, aby se tato rezerva vynulovala. K tomu potřebujeme nejdříve přepravit *r(v)* jednotek ze zdroje *s* do vrcholu *v*, přičemž si u každého vrcholu budeme pamatovat tzv. *plán p(w)*, což udává množství tekutiny, které budeme chtít přepravit ze zdroje *s* do vrcholu *w*. Na začátku položíme *p(w)* pro všechny vrcholy kromě *v*, kterému přiřadíme *p(v) = r(v)*. Postupujeme po vrstvách ve směru ke zdroji a budeme plnit plány vrcholů tak, že je převedeme na plány vrcholů v následující vrstvě. Takto pokračujeme až ke zdroji, jehož plán je evidentně splněn. Stejným způsobem pak pošleme množství *r(v)* z vrcholu v do spotřebiče. Při provádění jednotlivých operací stále přepočítáváme r^+, r^- a *r* podle změn rezerv jednotlivých hran a čistíme síť.

Zde je shrnuta Metoda tří Indů pro nalezení blokujícího toku:

1.Krok. Položíme $f_B = 0$.

2.Krok. Zjistíme rezervy všech hran a u každého vrcholu r^+, r^- a *r*. (Pozor, při každé změně toku je nutné znovu přepočítat!)

3. Krok. Mají-li všechny vrcholy nulovou rezervu, algoritmus končí. Existují-li v síti vrcholy s nenulovou rezervou, pak vezmeme vrchol *v* s nejmenší hodnotou *r(v)* a pokračujeme pro tento vrchol následujícími kroky:

> **3.1.** Položíme *p(v) = r(v)*, pro všechny ostatní vrcholy položíme *p(.) = 0*. (**.** značí všechny ostatní vrcholy kromě *v*).
>
> **3.2.** Budeme postupovat po jednotlivých vrcholech ve vrstvách sítě směrem od *v* k *s*, pro každý takový vrchol *w* provedeme následující operace, dokud bude platit *p(w) > 0* (while cyklus):
>
> > **3.2.1.** Zvolíme libovolnou hranu *{u, v}* a zvýšíme po ní tok o $\Delta = min\ (\ r(\{u, v\}), p(w))$. Plán *p(w)* tím snížíme o Δ a plán *p(u)* se naopak o Δ zvýší.
> >
> > **3.2.2.** Je-li nyní hrana *{u, v}* nasycená, odstraníme ji ze sítě a provedeme dočištění sítě.

4. Krok. Stejným způsobem jako v krocích 1 -3 převedeme *r(v)* jednotek z vrcholu *v* do *s*.

Ukažme, že algoritmus skončí a skutečně vydá blokující tok. Sečteme-li všechny *p(w)* přes každou vrstvu, je součet nejvýše roven *r(v),* a proto pro každé *w* platí $p(w) \leq r(v)$. Ve 3.kroku vybíráme vrchol *v* s nejmenší hodnotou *r(v)*, tedy platí nerovnosti $p(w) \leq r(v) \leq r(w) \leq r^{+}(w)$. Odtud plyne, že je možné splnit plán, požadovaný tok je kudy přivést.

8.6 Toky v sítích jako úloha lineárního programování

V této podkapitole se podíváme na toky v sítích z hlediska lineární algebry.

8.6.1 Definice. *Cirkulací* v orientovaném grafu $G = (V, E)$ nazveme libovolnou funkci $f : E \rightarrow \mathbf{R}$, která splňuje v každém vrcholu Kirchhoffův zákon, neboli $\sum_{\{x,v\}\in E} f(\{x,v\}) - \sum_{\{v,x\}\in E} f(\{v,x\}) = 0$.

Vyjádříme-li orientovaný graf *G* pomocí jemu příslušné matice incidence D_G a *f* jako sloupcový vektor indexovaný hranami, můžeme Kirchhoffův zákon zapsat jako $D_G f = 0$.

U toků v síti se Kirchhoffův zákon nepožaduje pro zdroj a spotřebič, ale upravíme-li graf, můžeme z každého toku udělat graf. V dané síti *(G, s, t, c)*, kde *G = (V, E)* budeme předpokládat, že hrana $\{t, s\} \notin E$. Vytvoříme nový orientovaný graf $G' = (V, E')$, kde $E' = E \cup \{t, s\}$. Kapacitu přidané hrany $c(\{t, s\})$ zvolíme „dost velkou", například jako součet kapacit všech ostatních hran, aby neomezovala tok zbytkem grafu.

Nechť je dán nějaký tok *f* v původní síti, na *G´* definujeme cirkulaci *f´* následujícím způsobem: $f'(e) = f(e)$ pro $e \in E$ a $f'(\{t, s\}) = w(f)$. Úloha hledání maximálního toku v síti je ekvivalentní hledání takové cirkulace *f´*v grafu *G´,* která maximalizuje $f'(\{t, s\})$ a pro každou jinou hranu leží její hodnota mezi 0 a *c(e).*

Takto formulovanou úlohu o nalezení maximálního toku lze zapsat jako úlohu lineárního programování. Budeme chtít nalézt takové řešení dané soustavy lineárních rovnic a nerovnic maximalizující určitou lineární funkci. Formální zápis úlohy:

$$max\{f'(\{t, s\}); f' \in \mathbf{R}^{|E'|}, D_G f' = 0, f' \leq c, f' \geq 0\}.$$

Zde představují f' a c jako $|E'|$ – složkové reálné vektory, rovnost či nerovnost vektorů v zápise symbolizuje rovnost nebo nerovnost po složkách.

8.7 Nejlevnější cirkulace

Již v minulé kapitole jsme v definici 8.6.1 zavedli pojem cirkulace. Nejdříve zjistíme, za jakých podmínek existuje v síti přípustná cirkulace.

8.7.1 Věta. *V síti s omezeními l a c existuje přípustná cirkulace, právě když pro každou množinu vrcholů A platí, že tok, který do ní musí vtéci kvůli dolnímu omezení l na hranách s koncovým vrcholem v množině A (přičemž jejich počáteční vrchol do množiny A nepatří), může z množiny A odtéci díky hornímu omezení c na hranách s počátečním vrcholem v A a koncovým mimo A.*

Důkaz. Označme $W^+(A)$ množinu hran s počátečním vrcholem v množině A a koncovým mimo A, $W^-(A)$ množinu hran s koncovým vrcholem v A a počátečním vrcholem mimo A. Existuje-li přípustná cirkulace, pak pro každý řez platí následující podmínka:

$$c(A) = \sum_{e \in W^+(A)} c(e) - \sum_{e \in W^-(A)} l(e) \geq \sum_{e \in W^+(A)} f(e) - \sum_{e \in W^-(A)} f(e) = 0.$$

Q.E.D.

Zaveďme označení $K(e)$ pro koncový vrchol hrany e, $P(e)$ pro počáteční vrchol hrany e a označíme $a(e)$ jednotkovou cenu toku v hraně e, přičemž součin $a(e)f(e)$ se nazývá cenou toku v hraně e. Předpokládejme, že máme dánu síť G s omezeními toku l, c, jednotkovými cenami toku a a libovolnou cirkulaci f.

8.7.2 Definice. *Cenou zlepšující kružnice* se rozumí součet jednotkových cen na hranách vpřed mínus součet jednotkových cen na hranách vzad.

8.7.3 Věta. *Přípustná cirkulace f je nejlevnější cirkulací, právě když vzhledem k cirkulaci f neexistuje žádná zlepšující kružnice se zápornou cenou.*

Důkaz. <= Za předpokladu existence zlepšující kružnice se zápornou cenou, bychom podle ní mohli cirkulaci změnit, čímž by se snížila celková cena, tedy *f* by už nebyla nejlevnější cirkulací.
=> Předpokládejme, že *f* je cirkulace, vzhledem k níž neexistuje zlepšující kružnice se zápornou cenou. Nejlevnější přípustnou cirkulaci označme f_1. Z cirkulace *f* dosáhneme cirkulace f_1 změnami podél zlepšujících kružnic vzhledem k cirkulaci *f* (přičemž žádná ze zlepšujících kružnic nemá zápornou cenu). Cirkulace *f* tedy už byla nejlevnější cirkulací a f_1 není levnější.

Q.E.D.

8.8 Algoritmus Out of Kilter

Jedná se o algoritmus pro hledání nejlevnější přípustné cirkulace. Pracuje s *potenciály*, což jsou hodnoty přiřazené vrcholům grafu. Právě rozdíly potenciálů na hranách jsou základem efektivního hledání zlepšujících kružnic se zápornou cenou či důkazu, že již taková kružnice neexistuje a máme tedy nejlevnější přípustnou cirkulaci.

8.8.1 Věta. *Nechť je dána síť G s omezeními toku l, c, jednotkovými cenami toku a* a *f je cirkulace. Existuje-li ohodnocení vrcholů* $u : V \rightarrow \boldsymbol{R}$ *takové, že pro každou hranu e jsou splněny následující tzv. kilter-podmínky:*

1) $l(e) \leq f(e) \leq c(e)$
2) platí-li $u(K(e)) - u(P(e)) > a(e)$, *pak* $f(e) = c(e)$
3) platí-li $u(K(e)) - u(P(e)) < a(e)$, *pak* $f(e) = l(e)$,

pak je cirkulace f nejlevnější přípustnou cirkulací.

Důkaz. Budeme dokazovat, že za splnění kilter podmínek žádná zlepšující kružnice vzhledem k cirkulaci f již nemá zápornou cenu, a tedy podle věty 8.7.3 je f nejlevnější cirkulace. Zvolíme libovolně nějakou zlepšující kružnici. Aplikujeme-li kilter podmínky, dostáváme pro hrany vpřed $u(K(e)) - u(P(e)) \leq a(e)$ a pro hrany vzad $u(K(e)) - u(P(e)) \leq -a(e)$. Sečteme-li zvlášť levé strany nerovností a pravé strany nerovností, dostáváme, že cena zlepšující kružnice je ≥ 0, tedy nemá zápornou cenu a cirkulace f je nejlevnější.

Q.E.D.

8.8.2 Definice. Defekt hrany e (značíme $D(e)$), která vede z vrcholu i do vrcholu j, definujeme takto:

$D(e) = |f(e) - l(e)|$, pokud $u(j) - u(i) < a(e)$

$D(e) = l(e) - f(e)$, pokud $u(j) - u(i) = a(e)$ a $f(e) < l(e)$

$D(e) = 0$, pokud $u(j) - u(i) = a(e)$ a $l(e) \leq f(e) \leq c(e)$

$D(e) = f(e) - c(e)$, pokud $u(j) - u(i) = a(e)$ a $c(e) < f(e)$

$D(e) = |f(e) - c(e)|$, pokud $u(j) - u(i) > a(e)$

Defekt je pro jakoukoli hranu nezáporný a je roven nule, pokud hrana splňuje kilter podmínky. Algoritmus Out of Kilter se snaží měnit tok a potenciály tak, aby defekt nikde nevzrostl a alespoň v jedné hraně klesl.

Pro hledání zlepšující kružnice budeme používat následující tzv. značkovací proceduru. Značkování začneme ve zdroji s a naším cílem bude označkovat spotřebič t. Pokud se nám to podaří, nalezli jsme cestu z s do t a po přidání hrany mezi s a t vytvoříme zlepšující kružnice. Procedura značkování probíhá následujícím způsobem:

1.Krok. Označkujeme zdroj s, přičemž ostatní vrcholy jsou bez značek.

2.Krok. Pokud existuje hrana e, pro kterou platí, že $P(e)$ je již označkovaný a $K(e)$ není označkovaný, a pro kterou platí následující podmínky:

1) $f(e) < l(e)$

2) $l(e) \leq f(e) < c(e)$ a $u(K(e)) - u(P(e)) \geq a(e)$

pak označkujeme vrchol $K(e)$

3.Krok. Pokud existuje hrana e, pro kterou platí, že $K(e)$ je již označkovaný a $P(e)$ není označkovaný, a pro kterou platí následující podmínky:

1)f(e) > c(e)

2) $l(e) < f(e) \leq c(e)$ a $u(K(e)) - u(P(e)) \leq a(e)$

pak označkujeme vrchol $P(e)$.

4.Krok. Máme-li již označkovaný spotřebič t, případně pokud nelze označkovat žádný jiný vrchol, značkovací procedura končí,

Pokud pomocí značkovací procedury nebude možné označkovat vrchol s, pak v řezu A určeném všemi dosud označkovanými vrcholy bude platit pro každou hranu jedna z následujících podmínek:

(11) $e \in W^+(A)$ a $f(e) \geq c(e)$

(22) $e \in W^+(A)$ a $l(e) \leq f(e) < c(e)$ a $u(K(e)) - u(P(e)) < a(e)$

(33) $e \in W^-(A)$ a $f(e) \leq l(e)$

(44) $e \in W^-(A)$ a $l(e) < f(e) \leq c(e)$ a $u(K(e)) - u(P(e)) > a(e)$

Předpokládáme, že máme dán orientovaný graf G, omezení toku l, c, jednotkové ceny toků a, výchozí cirkulaci f a výchozí potenciály vrcholů u (libovolná čísla). Budeme chtít změnit cirkulaci f a potenciály u takovým způsobem, aby všechny hrany splňovaly kilter-podmínky, nebo najít řez se zápornou kapacitou, což by bránilo existenci přípustné cirkulace. A takto funguje algoritmus Out of Kilter:

1.Krok. Nalezneme nějakou hranu h, která nesplňuje kilter-podmínky. Neexistuje-li taková hrana, algoritmus končí, dle věty 8.8.1 je cirkulace f nejlevnější přípustná.

2.Krok. Splňuje-li hrana h některou z podmínek 2.kroku značkovací procedury, položíme $s:= K(h)$ a $t:=P(h)$. Jinak položíme $s:=P(h)$ a $t:=K(h)$

3.Krok. Aplikujeme značkovací proceduru a označkujeme-li vrchol t, pokračujeme 4.krokem, jinak přejdeme na 7.krok.

4.Krok. Ke zlepšující cestě nalezené značkovací procedurou přidáme hranu h, tím vznikne zlepšující kružnice. Hrana h se bude považovat za hranu vpřed, platí-li pro ni alespoň jedna z podmínek 2.kroku značkovací procedury, a za hranu vzad, platí-li pro ni alespoň jedna z podmínek 3.kroku značkovací procedury.

5.Krok. Určíme kapacitu d zlepšující kružnice, a to jako minimum přes všechny hrany kružnice z následujících hodnot:

$D(e)$ pro hrany s $D(e) > 0$

$c(e) - f(e)$ pro hrany vpřed s $D(e) = 0$

$f(e) - l(e)$ pro hrany vzad s $D(e) = 0$.

6.Krok. Změníme cirkulaci na hranách zlepšující kružnice:

$f(e) := f(e) + d$ pro hrany vpřed

$f(e) := f(e) - d$ pro hrany vzad

Vrátíme se na 1.krok.

7.Krok. Nechť A je řez určený množinou všech již označkovaných vrcholů. Pokud v řezu pro všechny hrany z $W^+(A)$ platí podmínka (11) a zároveň pro všechny hrany z $W^-(A)$ platí podmínka (33), přejdeme na 9.krok, jinak budeme pokračovat 8.krokem.

8.Krok. Určíme velikost změny potenciálu σ následovně:

$\sigma = min\{ | u(K(e)) - u(P(e)) - a(e)|$, e jsou všechny hrany řezu určeného množinou A splňující podmínku (22) nebo (44) $\}$. Pokračujeme 10.krokem.

9.Krok. Platí-li $f(h) = l(h)$ nebo $f(h) = c(h)$, vypočteme velikost změny potenciálu σ následovně:

1) pokud $f(h) = l(h)$ pak $\sigma := u(s) - u(t) - a(h)$

2)pokud $f(h) = c(h)$ pak $\sigma := u(s) - u(t) + a(h)$

Pokračujeme 10.krokem. Pokud platí $l(h) \neq f(h) \neq c(h)$, algoritmus končí, protože neexistuje přípustná cirkulace a množina A určuje řez se zápornou kapacitou.

10.Krok. U všech doposud neoznačkovaných vrcholů zvýšíme jejich potenciál o σ, neboli $u(v) := u(v) + \sigma$ a vrátíme se na 1.krok.

8.9 Toky v sítích v praxi

V poslední podkapitole věnované tokům v sítích se podíváme na některé zajímavé praktické aplikace této teorie. Příklady čerpám ze zdrojů [1], [2], [14],

8.9.1 Příklad. Dopravní úloha. Představme si dopravní síť s několika dodavateli kukuřice a několika spotřebiteli. Síť reprezentujeme pomocí orientovaného grafu, přičemž některé vrcholy odpovídají dodavatelům a spotřebitelům, jiné vrcholy symbolizují křižovatky silnic, překladiště, přístavy atd., hrany grafů udávají úseky silnic, železnice, linky nákladní dopravy apod. Víme také, kolik kukuřice lze po jednotlivých hranách přepravit (kapacity hran), přičemž množství může být i neomezené. Zároveň jsou známy ceny za dopravu jednotkového množství kukuřice po jednotlivých hranách. Naším úkolem bude najít nejlevnější přípustný způsob dopravy kukuřice od dodavatelů ke spotřebitelům, neboli nelézt nejlevnější tok.

Přidáme do sítě umělý zdroj, který spojíme hranami se skutečnými zdroji (dodavateli), a umělý spotřebič, který spojíme hranami se skutečnými spotřebiči (spotřebiteli). V takto přidaných hranách omezíme tok zdola i shora, čímž vyjádříme požadavky spotřebitelů a kapacity dodavatelů. Pokud by kapacity dodavatelů nestačily k uspokojení poptávky a pokud by spotřebitelé měli různé priority, mohli bychom priority vyjádřit pomocí jednotkových cen toku v hranách od spotřebitelů k uměle přidanému spotřebiči. Tyto jednotkové ceny mohou být i záporné. Přidáme ještě klasicky tzv.návratovou hranu, která vede z umělého spotřebiče k umělému zdroji, a k řešení použijeme algoritmus pro hledání nejlevnější přípustné cirkulace (např. Algoritmus Out of Kilter).

Speciálním případem právě popsané úlohy je tzv. Hitchcockova dopravní úloha, kdy je síť reprezentována úplným bipartitním grafem. Hrany vedou od každého dodavatele ke všem spotřebitelům, přičemž jejich kapacity jsou neomezené. Často je ještě požadováno, aby byla dopravní úloha vyvážená, tedy aby

se součet kapacit dodavatelů rovnal součtu požadavku spotřebitelů. Velikost toku pro každého dodavatele i spotřebitele je v tomto případě přesně předepsána.

8.9.2 Příklad. Párování. Budeme řešit úlohu tzv. bipartitního párování, kdy bude naším úkolem v bipartitním grafu najít takovou podmnožinu hran, aby žádné dvě hrany nesdílely vrchol. Takovou úlohou je např. vytvoření tanečních párů v tanečních, kdy máme předem známe možnosti. Přidáme do grafu umělý zdroj, který spojíme hranami s jednou skupinou vrcholů (např. se všemi vrcholy, které reprezentují chlapce), a umělý spotřebič, který spojíme hranami s druhou skupinou vrcholů (např. dívkami). Všem hranám přiřadíme maximální kapacitu 1 a budeme hledat maximální tok. Pár pak budou tvořit vrcholy na hranách s nenulovým tokem.

8.9.3 Příklad. Nejlevnější zkracování činností. Síťový graf znázorňuje nějaký projekt, který se skládá z mnoha dílčích činností. Představme si, že máme síťový graf, kde pro každou činnost známe náklady navíc, které je potřeba vynaložit, budeme-li chtít trvání činnosti zkrátit o jednu časovou jednotku. Cílem je zkrátit trvání celého projektu znázorněného síťovým grafem co nejlevněji.

Budeme zkracovat jen činnosti (reprezentované hranami), které leží na kritické cestě (viz. kapitola 3.2). Graf však může obsahovat kritických cest více, my budeme chtít zkrátit takovou množinu kritických činností, aby byla na každé kritické cestě zkrácena nějaká činnost. Množina zkrácených činností tvoří v podgrafu určeném kritickými činnostmi řez oddělující počáteční a koncový vrchol síťového grafu. Bude nás zajímat množina s minimálním součtem nákladů, neboli řez s minimálním součtem ohodnocení. K nalezení takového řezu použijeme algoritmus pro hledání maximálního toku, který zároveň vydá i minimální řez. Kapacity hran představují náklady na jednotkové zkrácení činností.

Při zkrácení některých kritických činností se některé další činnosti mohou stát kritickými, musíme tedy algoritmus několikrát opakovat.

8.9.4 Příklad. Nejmenší počet lodí. Lodní společnost uzavřela smlouvu na přepravu potravin mezi různými destinacemi.Vzhledem k povaze zboží si jednotliví zákazníci určili, kdy přesně má být zboží dodáno. Lodní společnost chce určit nejmenší počet lodí, kterými je možné zboží dopravit podle požadavků zákazníků. Ilustrujeme to na příkladu se čtyřmi loděmi (viz. Tabulka 8.9.4.1-zdroj [1]).

Číslo lodi	Vyplouvá z přístavu	Má doplout do přístavu	Má zboží dodat za
1	A	C	3 dny
2	A	C	8 dní
3	B	D	3 dny
4	B	C	6 dní

8.9.4.1 Tabulka.

Máme k dispozici ještě časy, za které se lze dostat z přístavu do přístavu cestou tam (Tabulka 8.9.4.2) a cestou zpátky (Tabulka 8.9.4.3), obě tabulky zdroj [1]. V řádcích jsou přístavy, ze kterých loď vyplouvá, ve sloupcích pak přístavy, kam má doplout.

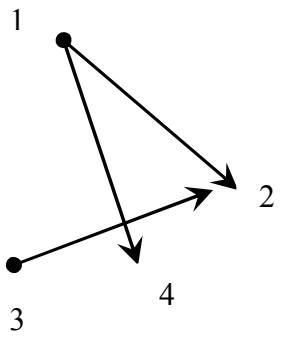
Přístav z a do	C	D
A	3	2
B	2	3

8.9.4.2 Tabulka.

Přístav z a do	A	B
C	2	1
D	1	2

8.9.4.3 Tabulka.

Sestrojíme orientovaný graf následujícím způsobem: vrcholy budou představovat lodě a hranu vedoucí z vrcholu *i* do vrcholu *j* sestrojíme v případě, že je možné doručit zboží z lodě *j* včas, a to za předpokladu, že loď *j* vypluje z přístavu do požadované destinace až po tom, co se loď *i*, která už odevzdala zboží v cílové destinaci, vrátí do výchozího přístavu. Pro náš konkrétní případ je taková síť na obrázku 8.9.4.4. (zdroj [1].)

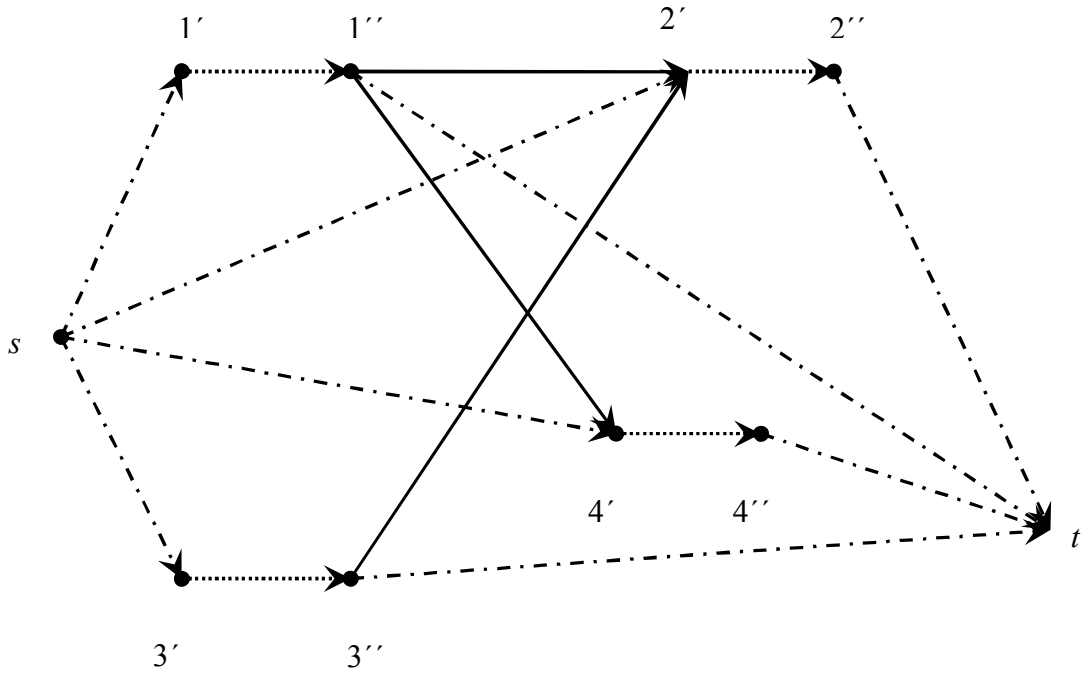


8.9.4.4 Obrázek

Orientovaná cesta v tomto případě odpovídá nějaké proveditelné sekvenci nakládání zboží na loď a doručování zboží v cílové destinaci.

Náš problém můžeme převést na problém hledání maximálního toku v síti. Z každého vrcholu vyrobíme vrcholy dva (*i´* a *i´´*) a spojíme je orientovanou hranou vedoucí z vrcholu *i´* do vrcholu *i´´,* přičemž nastavíme dolní omezení toku touto hranou na 1 (tedy *l(e)* =1). K takto upravené síti přidáme umělý zdroj *s*, který spojíme orientovanou hranou z *s* do každého z vrcholů *i´*, a umělý spotřebič *t,* který spojíme orientovanou hranou se všemi vrcholy *i´´.* Kapacitám těchto hran přiřadíme 1. Výsledná síť je pak na obrázku 8.9.4.5 (zdroj [1]).

Původní hrany z obrázku 8.9.4.4 jsou vyznačeny plnou čarou, tečkovaně jsou vyznačeny hrany mezi vrcholy *i´* a *i´´*, čerchovaně jsou vyznačeny hrany spojující zdroj a vrcholy *i´* a spotřebič s vrcholy *i´´* .



8.9.4.5 Obrázek. Výsledná síť pro hledání maximálního toku

Každá orientovaná cesta ze zdroje do spotřebiče na obrázku 8.9.4.5 představuje realizovatelný pohyb jednotlivých lodí. Přípustný tok o velikosti p se pak v této síti je určen „jízdním řádem“ p lodí. Nulový tok zde není přípustný, protože hranami mezi vrcholy i' a i'' musí protékat množství minimálně o velikosti 1. Použijeme algoritmus pro hledání maximálního toku.

8.9.5 Příklad. Evakuační plán. Při projektování budov musí architekti řešit, jak zajistit co nejlepší evakuaci budovy v případě zemětřesení, hlášené bomby aj. Musí stanovit také čas evakuace. Představíme si zde zjednodušený model. Sestavíme síť reprezentující budovu. Vrcholy budou označovat důležitá místa v budově – schodiště, zastávky výtahů, jednotlivá pracoviště a východy z budovy. Hrany pak budou reprezentovat možnosti, jak se lze přemístit mezi význačnými místy (např. chodby). Místa v budově s významnou koncentrací osob budeme považovat za zdroje, východy z budovy pak budou spotřebiči. Hodnota přiřazená vrcholu bude udávat počet osob zdržujících se na daném místě, které vrchol reprezentuje. Kapacity hran budou určeny počtem osob, který může daným místem, které hrana reprezentuje, projít ze nějakou pevně zvolenou jednotku času.

Použijeme obvyklý postup, kdy k síti připojíme umělý zdroj a umělý spotřebič, přičemž z umělého zdroje povede orientovaná hrana ke všem zdrojům a ze všech spotřebičů povede umělá hrana k umělému spotřebiči. Nyní už budeme předpokládat, že síť obsahuje pouze jeden zdroj a jeden spotřebič (ty uměle přidané). Pro zohlednění času sestrojíme nový graf tak, že sestrojíme p kopií každého vrcholu, kde p je nějaké dostatečně velké číslo, které zaručuje, že v čase p zvolených jednotek (stejných, jako byly zvoleny pro kapacity hran) bude určitě možné budovu evakuovat. Vrcholy spojíme hranami, pokud mezi stejnými vrcholy vedla hrana i v původním grafu. Nejkratší možný čas k evakuaci budovy je pak nejmenší index r takový, že maximální tok z vrcholů $s_1, \ldots, s_r$ do vrcholů $t_1, \ldots, t_r$ (kde $s_1, \ldots, s_r$ jsou repliky zdroje, $t_1, \ldots, t_r$ jsou repliky spotřebiče) je nejméně C osob, kde C je předem zadané číslo.

9. Zajímavé příklady na závěr

V poslední kapitole uvádím dva příklady vztahující se k problému nejkratší cesty a minimální (maximální) kostry, naznačuji krátce také možnost využití speciálního algoritmu ACO při řešení optimalizačních úloh. Čerpám ze zdrojů [1], [2], [15].

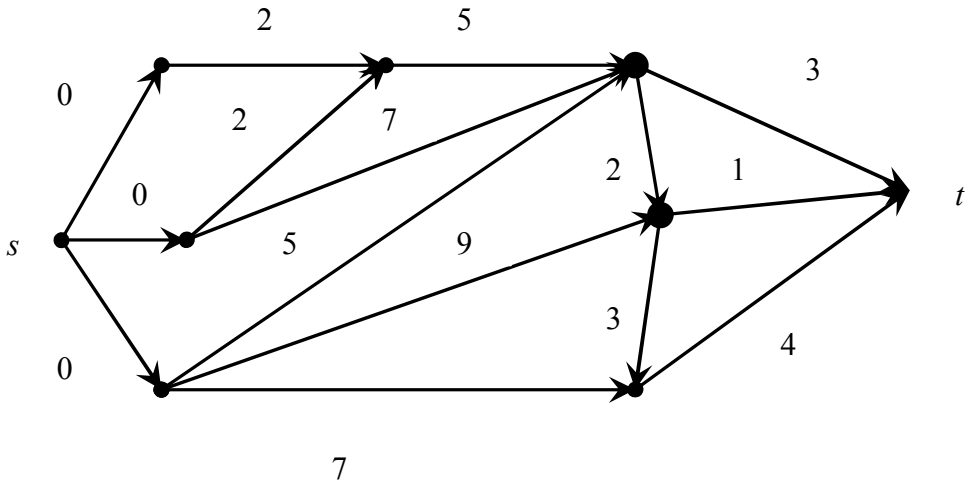
9.1 Příklad. Optimální předání důvěrné zprávy. Představme si, že ruská rozvědka má *n* tajných agentů v nepřátelské zemi. Každý agent se zná s některými z ostatních agentů a může si s agenty, které zná, sjednat schůzku. Předpokládejme, že známe pravděpodobnost p_{ij}, se kterou se jakákoliv zpráva předávaná mezi agenty *i* a *j* může dostat do rukou nepřítele, sejdou-li se tito dva agenti. Šéf agentů chce rozšířit důvěrnou zprávu mezi všechny agenty a minimalizovat při tom pravděpodobnost úniku této zprávy k nepříteli.

Sestrojíme graf, ve kterém budou vrcholy označovat jednotlivé agenty, hranu mezi vrcholy vedeme, mohou-li se tito dva agenti sejít (tj. znají-li se). V tomto grafu budeme chtít najít takovou kostru *T*, která bude minimalizovat výraz $\{1 - \prod_{\{i,j\}\in T}(1-p_{ij})\}$. Nebo můžeme najít kostru *K*, která bude maximalizovat výraz $\prod_{\{i,j\}\in T}(1-p_{ij})$. Přiřadíme-li hranám váhy tak, že hraně mezi dvěma vrcholy *i* a *j* přiřadíme odpovídající hodnotu $log(1 - p_{ij})$, stačí pak již v grafu s takovým ohodnocením najít maximální kostru (např. Kruskalovým algoritmem, kdy začneme s nejdražšími hranami a postupujeme až k těm nejlevnějším, neboli seřadíme na začátku váhy sestupně).

9.2. Příklad. Nejkratší doba potřebná k uskutečnění projektu. Představme si projekt, který se skládá z několika dílčích činností, přičemž k tomu, aby některé činnosti byly zahájeny, musí být ukončeny některé činnosti předtím. Chceme sestavit časový harmonogram tak, abychom dodrželi vztahy mezi jednotlivými činnostmi (co je potřeba mít hotové k tomu, aby něco jiného začalo) a zároveň dokončili projekt v nejkratším možném čase.

Situaci ilustrujeme orientovaným grafem, kde vrcholy budou představovat jednotlivé činnosti, hrana mezi vrcholy *i* a *j* pak bude znamenat, že činnost *i* je bezprostředním předchůdcem činnosti *j*. Hraně mezi vrcholy *i* a *j* přiřadíme váhu,

která odpovídá době trvání činnosti *i*. Ke grafu přidáme pro lepší orientaci tzv. počáteční vrchol (u toků v sítích jsme ho nazývali zdroj) *s*. který symbolizuje začátek procesu, a spojíme ho orientovanými hranami se všemi vrcholy grafu, které nemají přímého předchůdce, váhy těchto hran pak položíme rovny nule. Přidáme ještě koncový vrchol (u toků v sítích spotřebič) t, do něj povedou hrany ze všech vrcholů původního grafu, které nemají přímého následovníka, váha hrany *{i, t}* bude odpovídat době trvání činnosti *i*. Příklad takového grafu je na obrázku 9.2.1.



9.2.1 Obrázek. Orientovaný graf představující sled činností.

Označme *u(i)* nejmenší možný čas, kdy může činnost *i* začít s ohledem na vztah k předchozím činnostem. Doba trvání projektu je pak *u(t) – u(s)*. Cílem úlohy je tedy minimalizovat výraz *u(t) – u(s)* za podmínek*: u(j) – u(i)* ≥ *w({i, j})* (kde *w* je váha hrany) pro všechny vrcholy *i*, *j* mezi nimiž vede hrana, *u(j)* je bez omezení pro všechny vrcholy. Úlohu řešíme algoritmem pro nalezení nejkratší cesty.

9.3. Dodatek. Jiné zajímavé metody optimalizace. Kromě tradičních optimalizačních metod jako jsou lineární a nelineární programování, dynamické programování aj. lze použít i jiné metody optimalizace – genetické algoritmy, statistické teorie optimalizace či heuristiky. Výhodné je používání těchto algoritmů při řešení problémů, pro něž není známý efektivní algoritmus či které jsou příliš

rozsáhlé a klasický algoritmus by úlohu řešil „hodně dlouho“. Výsledkem je velmi dobré řešení, i když není optimální.

Jednou zajímavou neobvyklou metodou je heuristický optimalizační algoritmus, jehož základem je chování skutečných kolonií mravenců, kteří spolu komunikují na základě feromonových stop. Pro potřeby zkoumání algoritmů byly vyvinuty kolonie „umělých“ mravenců (kooperující stochastické konstruktivní procedury), které zanechávají feromonové stopy na grafu, jež zobrazuje danou situaci. V souvislosti s koloniemi mravenců (nebo včel, bakterií) se hovoří o rojové inteligenci, což je technika umělé inteligence, která je založena na studiu chování decentralizovaných samoorganizujících se systémů. Tyto systémy se skládají z populace agentů, kteří spolu komunikují přímo nebo nepřímo tím, že působí na lokální prostředí. Lokální interakce vede k tomu, že se jednoduché vzory chování agentů stávají chováním globálním.

Jedním ze známých algoritmů založených na rojové inteligenci, který byl úspěšně aplikován na řešení některých složitých optimalizačních úloh, jsou mravenčí kolonie (ACO – Ant Colony Optimization). ACO je novou metodou diskrétní optimalizace. Jednotliví mravenci se řídí jednoduchými pravidly a mravenčí kolonie jako celek svým chováním dokáží řešit náročné optimalizační úlohy.

Chování mravenců má za cíl zachování kolonie. Při získávání potravy zanechávají jednotliví mravenci po cestě od hnízda k potravě a zpět feromony, které jsou pak ostatními mravenci na cestě za potravou detekovány. Mravenci si při cestě za potravou vybírají cestu s nejvyšší koncentrací feromonů (a sami po cestě další feromony zanechávají). Při mnohonásobném opakování pak místo s nejvyšší koncentrací feromonů udává nejkratší cestu.

Pro modelování fungování optimalizačního algoritmu byl vytvořen umělý mravenec, jehož chování je podobné chování skutečných mravenců (kolonie, kde mravenci vzájemně kooperují, feromonové stopy, nepřímá komunikace feromony…), ale pro lepší výsledky algoritmů byly některé vlastnosti posíleny (detekované množství feromonu jako funkce kvality řešení, paměť zaznamenávající vykonané akce…). Mravenci vytvářejí celkové řešení interaktivním přidáváním komponent do částečného řešení. Další komponentu vybírají především na základě heuristické informace o řešeném problému a zkušeností, které získali všichni mravenci od začátku výpočtu. Zkušenosti si pak předávají feromonovými stopami.

Závěr

V práci byli uvedeny nejčastější problémy kombinatorické optimalizace a algoritmy k jejich řešení, které se zabývají nalezením optimální cesty v grafu vzhledem k daným omezením. Některé algoritmy byly znázorněny na příkladech pro lepší pochopení jejich činnosti. Pro řešení některých problémů bylo někdy uvedeno i několik algoritmů, lze je srovnat podle časové složitosti (bez odvození) či počtu kroků potřebných k nalezení řešení (viz hledání minimální kostry).

Seznam použité literatury

[1] Ahuja R.K.,Magnanti T.L., Orlin J.B.: Network Flows, Theory, Algorithms and Applications, Prentice Hall, New Jersey, 1993

[2] Demel J.: Grafy a jejich aplikace, Academia, Praha, 2002

[3] Hochbaum D.S.: Approximation Algorithms for NP-hard Problems, PWS Publishing Copany, Boston, 1997

[4] Kocay W., Kreher D.L.: Graphs, Algorithms, and Optimization, Chapman & Hall / CRC Press, USA, 2005

[5] Kučera L.: Kombinatorické algoritmy, SNTL -Nakladatelství technické litaratury, Praha, 1983.

[6] Mareš M.: Krajinou grafových algoritmů, průvodce pro středně pokročilé, Institut teoretické informatiky, Praha, 2007

[7] Matoušek J., Nešetřil J.: Kapitoly z diskrétní matematiky, Nakladatelství Karolinum, Praha, 2007.

[8] Nešetřil J.: Kombinatorika I., Grafy, Státní pedagogické nakladatelství Praha, Praha, 1983

[9] Nešetřil J.: Teorie grafů, SNTL-Nakladatelství technické literatury, Praha, 1979

[10] Plesník J.: Grafové algoritmy, VEDA, Vydavatel´stvo slovenskej Akademie vied, Bratislava, 1983.

[11] Valla T.,Matoušek J.: Kombinatorika a grafy I., Katedra aplikované matematiky MFF UK, Praha, 2005

[12] http://www.algoritmy.net/

[13] http://kam.mff.cuni.cz/~kuba/ka/

[14] http://is.muni.cz/el/1433/podzim2006/MB103/um/mIII-11.pdf?fakulta=1433;obdobi=3523;kod=MB103

[15] http://www.milosnemec.cz/download/dp.pdf