6 Minimální kostry, Hladový algoritmus

Kromě teoretických "hrátek" mají kostry grafů (Oddíl 5.4) následující důležité praktické použití: Dříve jsme uvažovali spojení v grafech cestami jdoucími z jednoho místa do druhého, ale nyní se podíváme na jiný způsob "propojování" všech vrcholů grafu najednou.

Vezměme si třeba požadavky propojení domů elektrickým rozvodem, propojení škol internetem, atd. Zde nás ani tak nezajímají délky jednotlivých cest mezi propojenými body, ale hlavně celková délka či cena vedení/spojení, které musíme postavit. □

Našim cílem je tedy najít minimální souvislý podgraf daného grafu, tedy minimální způsob propojení (v daných podmínkách), ve kterém existují cesty mezi každou dvojicí vrcholů.

Stručný přehled lekce

- Řešení problému minimální kostry v grafu – hladový a Jarníkův algoritmus.
- Obecné pojetí hladového algoritmu.
- Matroidy – struktury, na nichž hladový algoritmus vždy funguje.

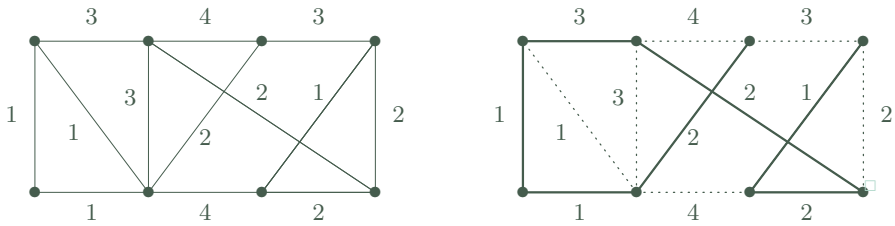
6.1 Hledání minimální kostry

Problém 6.1. Problém minimální kostry (MST)
Je dán (souvislý) vážený graf G, w s nezáporným ohodnocením hran w. Otázkou je najít kostru T v G, která má nejmenší možné celkové ohodnocení. Formálně

$$MST = \min_{\text{kostra } T \subset G} \left(\sum_{e \in E(T)} w(e) \right).$$

Kostra daného grafu je minimální podgraf, který zachovává souvislost každé komponenty původního grafu. Proto nám vlastně ukazuje "minimální propojení" daných vrcholů, ve kterém ještě existují cesty mezi všemi dvojicemi, které byly propojeny i původně.

Praktickou formulací problému je třeba propojení domů elektrickým rozvodem, škol internetem, atd. Jedná se tady o zadání, ve kterých nás zajímá především celková délka či cena propojení, které je třeba vytvořit. Příklad je uveden na následujícím obrázku i s vyznačenou minimální kostrou vpravo.

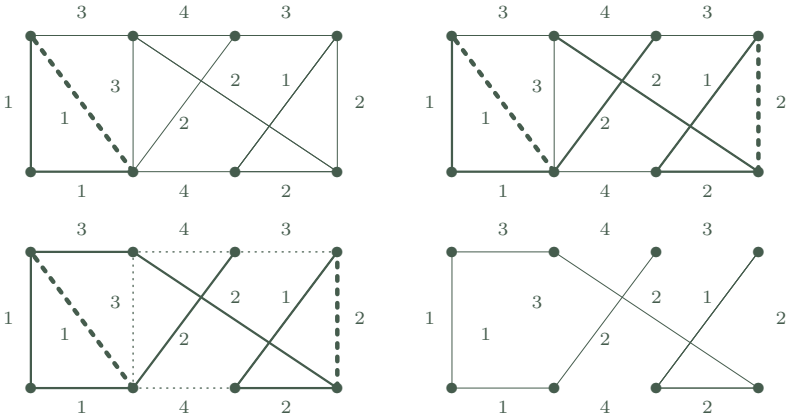


Algoritmus 6.2. „Hladový“ pro minimální kostru.
Je dán souvislý vážený graf G, w s nezáporným ohodnocením hran w.

- Seřadíme hrany grafu G vzestupně podle jejich ohodnocení, tj. $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_m)$. □
- Začneme s prázdnou množinou hran $T = \emptyset$ pro kostru.
- Pro $i = 1, 2, \ldots, m$ vezmeme hranu e_i a pokud $T \cup \{e_i\}$ nevytváří kružnici, přidáme e_i do T. Jinak e_i “zahodíme”. □
- Na konci množina T obsahuje hrany minimální kostry váženého grafu G, w.

Pro ilustraci si ukážeme postup hladového algoritmu pro vyhledání kostry výše zakresleného grafu. Hrany si nejprve seřadíme podle jejich vah $1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 4, 4$.

V obrázku průběhu algoritmu používáme tlusté čáry pro vybrané hrany kostry a tečkované čáry pro zahozené hrany. Hrany teď postupně přidáváme do kostry / zahazujeme...



Získáme tak minimální kostru velikosti $1+2+2+3+1+1+2 = 12$, která je v tomto případě (náhodou) cestou, na posledním obrázku vpravo.

Poznamenáváme, že při jiném seřazení hran stejné váhy by kostra mohla vyjít jinak, ale vždy bude mít stejnou velikost 12.

Důkaz správnosti Algoritmu 6.2:
Nechť T je množina hran získaná v Algoritmu 6.2 a nechť hrany jsou již seřazené $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_m)$. Vezměme množinu hran T_0 té minimální kostry (může jich být více se stejnou hodnotou), která se s T shoduje na co nejvíce prvních hranách. Pokud $T_0 = T$, algoritmus pracoval správně. □

Předpokládejme tedy, že $T_0 \neq T$, a ukážeme spor, tj. že toto nemůže ve skutečnosti nastat. Označme $j > 0$ takový index, že se množiny T_0 a T shodují na prvních $j-1$ hranách $e_1, \ldots, e_{j-1}$, ale neshodují se na e_j. To znamená, že $e_j \in T$, ale $e_j \notin T_0$. (Jistě nemůže nastat $e_j \notin T$, ale $e_j \in T_0$.) □

Podle Důsledku 5.5 obsahuje graf na hranách $T_0 \cup \{e_j\}$ právě jednu kružnici C. Kružnice C však nemůže být obsažena v nalezené kostře T, a proto existuje hrana e_k v C, která $e_k \notin T$ a zároveň $k > j$. Potom však je $w(e_k) \geq w(e_j)$ podle našeho seřazení hran, a tudíž kostra na hranách $\big(T_0 \setminus \{e_k\}\big) \cup \{e_j\}$ (vzniklá nahrazením hrany e_k hranou e_j) není horší než T_0 a měli jsme ji v naší úvaze zvolit místo T_0. To je hledaný spor. □ □

Správný pohled na předchozí důkaz by měl být následovný: Předpokládali jsme, že nalezená kostra T se s některou optimální kostrou shoduje aspoň na prvních $j-1$ hranách. Poté jsme ukázali, že některou další hranu e_k v (předpokládané) optimální kostře lze zaměnit hranou e_j, a tudíž dosáhnout shodu aspoň na prvních j hranách. Dalšími iteracemi záměn ukážeme úplnou shodu naší nalezené kostry T s některou optimální kostrou. V našem důkaze jsme se vlastně zaměřili na fakt, že ta poslední iterace záměn nemůže selhat. Nakreslete si tento důkaz obrázkem!

Základní hladový algoritmus pro hledání minimální kostry grafu byl poprvé explicitně popsán Kruskalem, ale už dříve byly objeveny jeho podobné varianty, které zde jen stručně zmíníme.

Algoritmus 6.3. Jarníkův pro minimální kostru.
Hrany na začátku neseřazujeme, ale začneme kostru vytvářet z jednoho vrcholu a v každém kroku přidáme nejmenší z hran, které vedou z již vytvořeného podstromu do zbytku grafu. □

Poznámka: Tento algoritmus je velmi vhodný pro praktické výpočty a je dodnes široce používaný. Málokdo ve světě však ví, že pochází od Vojtěcha Jarníka, známého českého matematika — ve světové literatuře se obvykle připisuje Američanu Primovi, který jej objevil až skoro 30 let po Jarníkovi.

Avšak historicky vůbec první algoritmus pro problém minimální kostry (z roku 1928) byl nalezen jiným českým (brněnským) matematikem: □

Algoritmus 6.4. Borůvkův pro minimální kostru (náznak).
Toto je poněkud složitější algoritmus, chová se jako Jarníkův algoritmus spuštěný zároveň ze všech vrcholů grafu najednou.

6.2 Hladový algoritmus v obecnosti

Asi nejprimitivnějším možným přístupem při řešení diskrétních optimalizačních úloh je postupovat stylem **beru vždy to nejlepší, co se zrovna nabízí**...

Tento postup obecně v češtině nazýváme *hladovým algoritmem*, i když lepší by bylo použít správnější překlad anglického "greedy", tedy nenasystný algoritmus. A ještě hezčí české spojení by bylo "algoritmus hamouna". Jednoduše bychom jej nastínili takto:

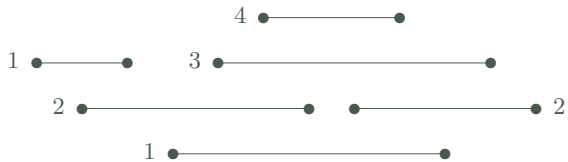
- Postupně v krocích vyber vždy to nejlepší, co se dá (nabízí).
- To vyžaduje zvolit *uspořádání na objektech*, ze kterých vybíráme.
- Průběh a úspěch algoritmu silně závisí na tomto zvoleném uspořádání.

Jak asi každý ví, nenasystnost či hamounství nebývá v životě tím nejlepším postupem, ale kupodivu tento princip perfektně funguje v mnoha kombinatorických úlohách! Jedním známým příkladem je výše uvedené hledání minimální kostry. Jiným příkladem je třeba jednoduchý problém přidělování (uniformních) pracovních úkolů, na němž si obecné schéma hladového algoritmu ilustrujeme.

Problém 6.5. Přidělení pracovních úkolů

Uvažujeme zadané pracovní úkoly, které mají přesně určený čas začátku i délku trvání. (Jednotlivé úkoly jsou tedy reprezentovány uzavřenými intervaly na časové ose.) Cílem je takové přidělení úkolů pracovníkům, aby jich celkově bylo potřeba co nejméně. Všichni pracovníci jsou si navzájem rovnocenní – uniformní, tj. každý zvládne všechno. □

Pro příklad zadání takové problému si vezměme následující intervaly úkolů:



Kolik je k jejich splnění potřeba nejméně pracovníků? □Asi sami snadno zjistíte, že 4 pracovníci stačí, viz zobrazené očíslování. Ale proč jich nemůže být méně?

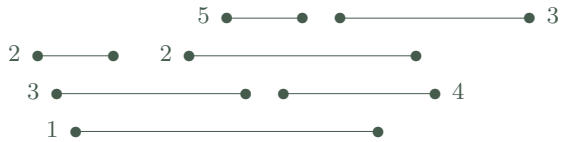
Poznámka: Uvedené zadání může být kombinatoricky popsáno také jako problém optimálního obarvení daného intervalového grafu (vrcholy jsou intervaly úkolů a hrany znázorňují překrývání intervalů).

Algoritmus 6.6. Hladový algoritmus rozdělení pracovních úkolů.
Problém 6.5 je vyřešen následující aplikací hladového postupu:

1. *Úkoly nejprve seřadíme podle časů začátků.*
2. *Každému úkolu v pořadí přidělíme volného pracovníka s nejnižším číslem.* □

Důkaz: Nechť náš algoritmus použije celkem k pracovníků. Dokážeme jednoduchou úvahou, že tento počet je optimální – nejlepší možný. V okamžiku, kdy začal pracovat pracovník číslo k, všichni $1, 2, \ldots, k-1$ také pracovali (jinak bychom vzali některého z nich). V tom okamžiku tedy máme k překrývajících se úkolů a každý z nich vyžaduje vlastního pracovníka. □ □

Příklad neoptimálního přidělení pracovních úkolů dostaneme například tak, že na začátku úkoly seřadíme podle jejich časové délky. (Tj. čím delší úkol, tím dříve mu hladově přiřadíme pracovníka.)



Jak vidíme na obrázku, nalezené řešení není optimální – vyžaduje 5 místo 4 pracovníků. Je tedy velmi důležité, podle jakého prinicipu seřadíme objekty (úkoly) na vstupu.

6.3 Pojem matroidu

Definice 6.7. Matroid na množině X, značený $M = (X, \mathcal{N})$,
je takový systém $\mathcal{N}$ podmnožin nosné množiny X, ve kterém platí následující:

1. $\emptyset \in \mathcal{N}$
2. $A \in \mathcal{N}$ a $B \subset A \;\Rightarrow\; B \in \mathcal{N}$
3. $A, B \in \mathcal{N}$ a $|A| < |B| \;\Rightarrow\; \exists y \in B \setminus A : A \cup \{y\} \in \mathcal{N}$

Množinám ze systému $\mathcal{N}$ říkáme *nezávislé množiny*. Těm ostatním pak říkáme *závislé*. Nezávislým množinám, do kterých již nelze přidat žádný prvek tak, že zůstanou nezávislé, říkáme *báze matroidu*. □

Nejdůležitější částí definice matroidu je zvýrazněný třetí bod. Přímo ukázkový příklad matroidu nám dává lineární algebra – všechny lineárně nezávislé podmnožiny vektorů tvoří matroid. Odtud také pocházejí pojmy nezávislosti a báze matroidu, které přímo odpovídají příslušným pojmům vektorového prostoru. □

Lema 6.8. *Všechny báze matroidu obsahují stejně mnoho prvků.*

Důkaz: Toto přímo vyplývá z třetí vlastnosti definice matroidu: Pokud nezávislá množina A má méně prvků než báze B, tak do A lze vždy přidat další prvek x tak, že zůstane $A \cup \{x\}$ nezávislá. □

Nyní uvedeme několik poznatků o stromech, které jsou relevantní pro zavedení "grafových" matroidů.

Lema 6.9. *Les na n vrcholech s c komponentami souvislosti má přesně $n - c$ hran.* □

Důkaz: Každý vrchol lesa L náleží právě jedné komponentě souvislosti z definice. Jak známo, každý strom, tj. komponenta lesa L, má o jednu hranu méně než vrcholů. Ve sjednocení c komponent tak bude právě o c méně hran než vrcholů. □

Definice: Řekneme, že podmnožina hran $F \subset E(G)$ je *acyklická*, pokud podgraf s vrcholy $V(G)$ a hranami z F nemá kružnici.

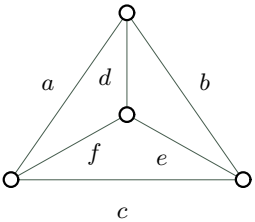
Lema 6.10. *Nechť F_1, F_2 jsou acyklické podmnožiny hran grafu G a $|F_1| < |F_2|$. Pak existuje hrana $f \in F_2 \setminus F_1$ taková, že $F_1 \cup \{f\}$ je také acyklická podmnožina.* □

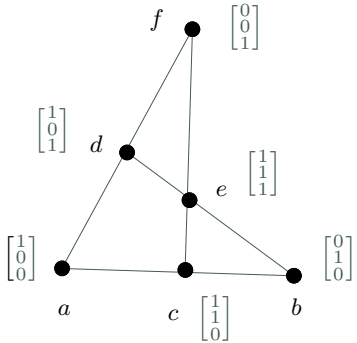
Důkaz: Jelikož $|F_1| < |F_2|$ a platí Lema 6.9, má podgraf G_1 tvořený hranami z F_1 více komponent než podgraf G_2 tvořený hranami z F_2. Potom však některá hrana $f \in F_2$ musí spojovat dvě různé komponenty podgrafu G_1, a tudíž přidáním f do F_1 nevznikne kružnice. □

Definice: Podle Lematu 6.10 tvoří systém všech acyklických podmnožin hran v (libovolném) grafu G matroid. Tento matroid nazýváme *matroidem kružnic* grafu G.
V analogií s grafy používáme název *kružnice* pro minimální závislé množiny matroidu.
□

Dva příklady matroidu jsou hezky ilustrovány v následujícím obrázku, který ukazuje, jak hrany grafu K_4 vlevo odpovídají vektorům v matroidu kružnic vpravo. Čáry (zvané "přímky") v pravém schématu vyznačují lineární závislosti mezi vektory; tj. nezávislé jsou ty trojice bodů, které neleží na žádné společné "přímce".

K_4





Abstraktní hladový algoritmus

V praxi se matroid obvykle nezadává výčtem všech nezávislých množin, protože těch je příliš mnoho (až 2^n pro n-prvkovou množinu X).
Místo toho bývá dána externí funkce pro testování nezávislosti dané podmnožiny. □

Algoritmus 6.11. Nalezení minim. báze matroidu – hladový algoritmus.
vstup: *množina X s váhovou funkcí $w : X \to \boldsymbol{R}$,*
matroid na X určený externí funkcí `nezavisla(Y);` □

`setřídit X=(x[1],x[2],...,x[n])` *tak,*
aby `w[x[1]]<=...<=w[x[n]];`
`B = ∅;`
`for (i=1; i<=n; i++)`
`  if (nezavisla(B∪{x[i]}))`
`    B = B∪{x[i]};`

výstup: *báze B daného matroidu s min. součtem ohodnocení vzhledem k w.* □

Poznámka: Pokud X v tomto algoritmu je množina hran grafu, w váhová funkce na hranách a nezávislost znamená acyklické podmnožiny hran (matroid kružnic grafu), pak Algoritmus 6.2 je přesně instancí Algoritmu 6.11.

Věta 6.12. *Algoritmus 6.11 (hladový algoritmus) pro danou nosnou množinu X s váhovou funkcí $w : X \to \boldsymbol{R}$ a pro daný matroid $\mathcal{N}$ na X správně najde bázi v $\mathcal{N}$ s nejmenším součtem vah.* □

Důkaz: Z definice matroidu je jasné, že k výsledné množině B již nelze přidat další prvek, aby zůstala nezávislá, proto je B báze. Seřaďme si prvky X podle vah jako v algoritmu $w(x[1]) \leq \cdots \leq w(x[n])$. Nechť indexy $i_1, i_2, \ldots, i_k$ určují vybranou k-prvkovou bázi B v algoritmu a nechť indexy $j_1, j_2, \ldots, j_k$ vyznačují (třeba jinou?) bázi $\{x[j_1], \ldots, x[j_k]\}$ s nejmenším možným součtem vah. □

Vezměme nejmenší $r \geq 1$ takové, že $w(x[i_r]) \neq w(x[j_r])$. Potom nutně $w(x[i_r]) < w(x[j_r])$, protože náš algoritmus je "hladový" a bral by menší $w(x[j_r])$ již dříve. Na druhou stranu, pokud by druhá báze $\{x[j_1], \ldots, x[j_k]\}$ dávala menší součet vah, muselo by existovat jiné $s \geq 1$ takové, že $w(x[i_s]) > w(x[j_s])$. Nyní vezměme nezávislé podmnožiny $A_1 = \{x[i_1], \ldots, x[i_{s-1}]\}$ a $A_2 = \{x[j_1], \ldots, x[j_s]\}$, kde A_2 má o jeden prvek více než A_1 a všechny prvky A_2 mají dle předpokladu menší váhu než $w(x[i_s])$. □

Podle definice matroidu existuje $y \in A_2 \setminus A_1$ takové, že $A_1 \cup \{y\}$ je nezávislá. Přitom samozřejmě $y = x[\ell]$ pro nějaké ℓ. Ale to není možné, protože, jak je výše napsáno, $w(y) < w(x[i_s])$, takže by náš hladový algoritmus musel $y = x[\ell]$, $\ell < i_s$ vzít dříve do vytvářené báze B než vzal $x[i_s]$. Proto jiná báze s menším součtem vah než nalezená B neexistuje. □

6.4 Kdy hladový algoritmus (ne)pracuje správně

Jak ukážeme, funkčnost hladového algoritmu je přímo svázána s matroidy.

Věta 6.13. *Nechť X je nosná množina se systémem "nezávislých" podmnožin $\mathcal{N}$ splňující podmínky (1,2) Definice 6.7. Pokud pro jakoukoliv váhovou funkci $w : X \to \boldsymbol{R}$ najde Algoritmus 6.11 optimální nezávislou množinu z $\mathcal{N}$, tak $\mathcal{N}$ splňuje také podmínku (3), a tudíž tvoří matroid na X.* □

Důkaz: Tvrzení dokazujeme sporem. Předpokládejme, že vlastnost (3) neplatí pro dvojici nezávislých množin A, B, tj. že $|A| < |B|$, ale pro žádný prvek $y \in B \setminus A$ není $A \cup \{y\}$ nezávislá. Nechť $|A| = a, |B| = b$, kde $2b > 2a + 1$. Zvolíme následující ohodnocení

- $w(x) = -2b$ pro $x \in A$,
- $w(x) = -2a - 1$ pro $x \in B \setminus A$,
- $w(x) = 0$ jinak.□

Hladový algoritmus přirozeně najde bázi B_1 obsahující A a disjunktní s $B \setminus A$ podle našeho předpokladu. Její ohodnocení je $w(B_1) = -2ab$. Avšak optimální bází je v tomto případě jiná B_2 obsahující celé B a mající ohodnocení nejvýše $w(B_2) \leq (-2a - 1)b = -2ab - b < w(B_1)$. To je ve sporu s dalším předpokladem, že i při námi zvoleném ohodnocení w nalezne hladový algoritmus optimální bázi. Proto je sporný náš předpoklad o množinách A, B a podmínka (3) je splněna. □

Příklad 6.14. *Při následujících dvou použitích paradigmatu hladového algoritmu dojde k situacím, kdy získaný výsledek může být zcela jiný než optimální odpověď.*

Obarvení grafu. Problém obarvení grafu žádá přiřazení co nejméně barev vrcholům tak, aby sousední dvojice dostaly různé barvy. Obecně hladově barvíme graf tak, že ve zvoleném pořadí vrcholů každému následujícímu přidělíme první volnou barvu.

Třeba v nakreslené cestě délky 3 můžeme barvit hladově v pořadí od vyznačených krajních vrcholů, a pak musíme použít 3 barvy místo optimálních dvou.

Vrcholové pokrytí. Problém vrcholového pokrytí se ptá na co nejmenší podmnožinu C vrcholů daného grafu takovou, že každá hrana má alespoň jeden konec v C. Přirozeným hladovým postupem by bylo vybírat od vrcholů nejvyšších stupňů ty, které sousedí s doposud nepokrytými hranami. Bohužel tento postup také obecně nefunguje.

□