1 O kombinatorické optimalizaci

Pro začátek si přiblížíme základní principy optimalizačních problémů a jejich řešení na několika poměrně jednoduchých úlohách vesměs spadajících do oblasti kombinatoriky (kombinatorické optimalizace).

Stručný přehled lekce

- Příklad nejjednodušší aplikace hladového postupu optimalizace.
- Řešení problému minimální kostry grafu – hladový a Jarníkův algoritmus.
- Matroidy – struktury, na nichž hladový algoritmus vždy funguje optimálně.
- Co je vlastně obecně "optimalizační úloha"?

1.1 Hladový algoritmus

Asi nejprimitivnějším možným přístupem při řešení optimalizačních úloh v kombinatorice je postup 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.

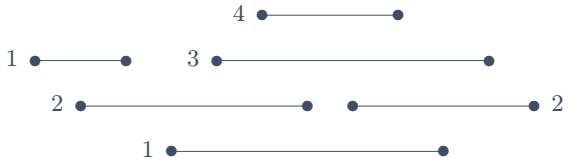
Jak asi každý ví, nenasystnost či hamounství nebývá v životě tím nejlepším postupem, ale kupodivu tento princip perfektně funguje v mnoha kombinatorických úlohách! Jedním známým příkladem je třeba *hledání minimální kostry* uvedené v příští lekci. Jiným příkladem je třeba jednoduchý problém přidělování (uniformních) pracovních úkolů, na němž si nejprve hladový algoritmus přiblížíme.

Úloha 1.1. 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.) Všichni pracovníci jsou si navzájem rovnocenní – uniformní, tj. každý zvládne všechno.

Vstup: Časové intervaly daných úkolů.
Výstup: Přidělení úkolů pracovníkům, aby bylo potřeba co nejméně pracovníků.

Pro příklad zadání takové úlohy 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: Uvedená úloha může být kombinatoricky popsána také jako problém optimálního obarvení daného intervalového grafu (vrcholy jsou intervaly úkolů a hrany znázorňují překrývání intervalů).

Algoritmus 1.2. Hladový algoritmus rozdělení pracovních úkolů.
Úloha 1.1 je vyřešena 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.

1.2 Problém minimální kostry

V dalším oddílu se podíváme na jeden problém z obasti grafů. Vzpomeňme si, že obecný graf bez kružnic je nazýván *les* a jeho souvislé komponenty jsou *stromy*.

Definice 1.3. Kostrou grafu G rozumíme podgraf v G,
který je lesem, obsahuje všechny vrcholy grafu G a na každé souvislé komponentě G indukuje strom.

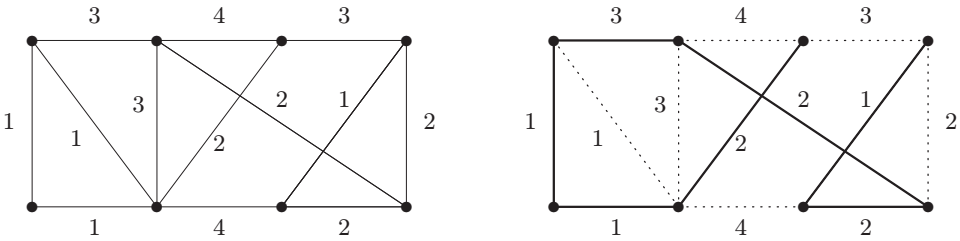
Kostra daného grafu je minimální podgraf, který zachovává souvislost každé komponenty původního grafu. Proto nám vlastně ukazuje "minimální propojení" daných vrcholů, ve kterém ještě existují cesty mezi všemi dvojicemi, které byly propojeny i původně.

Úloha 1.4. Problém minimální kostry (MST)
Otázkou je najít kostru T v nezáporně ohodnoceném souvislém grafu G, která má ze všech koster nejmenší celkové ohodnocení. Formálně:

Vstup: Souvislý graf G s nezáporným ohodnocením hran $w : E(G) \to \boldsymbol{R}^+$.
Výstup: Kostra v G s minimálním součtem hodnot hran

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

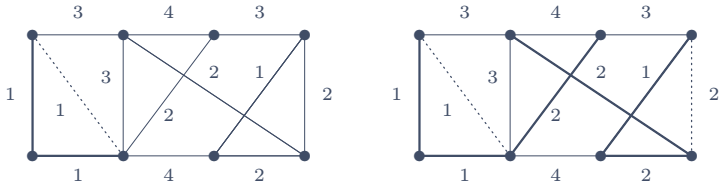
Praktickou formulací úlohy je třeba propojení domů elektrickým rozvodem, propojení škol internetem, atd. Zde nás ani tak nezajímají délky cest mezi propojenými body, ale hlavně celková délka či cena vedení/spojení, které musíme postavit. Vstupní graf nám přitom udává všechny možné realizovatelné propojky s jejich cenami. Příklad je uveden na následujícím obrázku i s vyznačenou minimální kostrou vpravo.



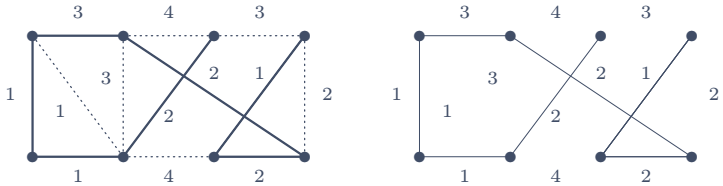
Algoritmus 1.5. Hladový algoritmus hledání minimální kostry v grafu.
Úloha 1.4 je vyřešena následující aplikací hladového postupu:

1. *Hrany grafu seřadíme podle jejich ohodnocení w od nejmenší.*
2. *Přidáváme do (budoucí) kostry postupně hrany, které nevytváří s dříve vybranými hranami kružnici. (Hrany uzavírající kružnice ignorujeme – "zahodíme".)*

Pro ilustraci si podrobně 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 postupně přidáváme do kostry... (tlusté čáry)



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.

Základní hladový algoritmus pro hledání minimální kostry byl popsán Kruskalem. Jiné (a mnohem starší) varianty výše popsaného algoritmu jsou následující:

- **Jarníkův algoritmus**
  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.

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

- **Borůvkův algoritmus**
  Toto je poněkud složitější algoritmus, chová se jako Jarníkův algoritmus spuštěný zároveň ze všech vrcholů grafu najednou. Jedná se o historicky vůbec první algoritmus pro minimální kostru z roku 1928, který se pak stal inspirací i pro Jarníkův algoritmus.

Důkaz správnosti hladového algoritmu pro hledání minimální kostry bude dále podán v obecnosti u pojmu matroidu v příští sekci.

1.3 Pojem matroidu

Definice 1.6. 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 $B \subseteq A \;\Rightarrow\; B \in \mathcal{N}$
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 1.7. *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 1.8. *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. □

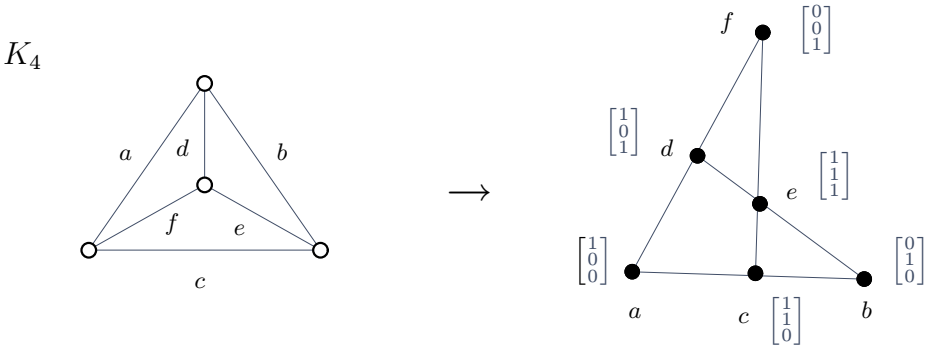
Definice: Řekneme, že podmnožina hran $F \subseteq E(G)$ je *acyklická*, pokud podgraf s vrcholy $V(G)$ a hranami z F nemá kružnici.

Lema 1.9. *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 1.8, 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 1.9 tvoří systém všech acyklických podmnožin hran v (libovolném) grafu G matroid. Tento matroid nazýváme *matroidem kružnic* grafu G.
V analogii s grafy používáme název *kružnice* pro minimální závislé množiny matroidu.

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



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 1.10. 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` $= \emptyset$;
`for (i=1; i<=n; i++)`
`if (nezavisla(B`$\cup\{$`x[i]`$\}$`))`
`B` $=$ `B`$\cup\{$`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 1.5 je přesně instancí Algoritmu 1.10.

Věta 1.11. *Algoritmus 1.10 (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 $w(x[1]) \leq \ldots \leq w(x[n])$. 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. □

Kdy hladový algoritmus nepracuje správně

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

Věta 1.12. *Nechť X je nosná množina se systémem "nezávislých" podmnožin $\mathcal{N}$ splňující podmínky (1,2) Definice 1.6. Pokud pro jakoukoliv váhovou funkci $w : X \to \boldsymbol{R}$ najde Algoritmus 1.10 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. □

Nakonec uvádíme několik příkladů, ve kterých hladový algoritmus výrazně selže:

Obarvení grafu. 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.

1.4 Co je optimalizační úloha

Definice 1.13. Optimalizační úloha
je výpočetní problém (zhruba) určený následujícími atributy zadání:

1. *Univerzem* $\mathcal{U}$ všech potenciálních řešení, jež je obvykle dáno jako vektorový prostor s jednotlivými proměnnými jako souřadnicemi.
2. *Omezujícími podmínkami*, které určují podmnožinu $\mathcal{P} \subseteq \mathcal{U}$ všech *přípustných řešení* úlohy.
3. *Účelovou funkcí* $\eta : \mathcal{U} \to \boldsymbol{R}$, která přiřazuje každému možnému řešení jeho hodnotu – cenu. Podle kontextu úlohy hodnotu účelové funkce buď *maximalizujeme* nebo *minimalizujeme*.

V Úloze 1.1 je univerzem prostor všech přiřazení čísel pracovníků jednotlivým úkolům (tj. celočíselný vektorový prostor $\boldsymbol{Z}^k$, kde k je počet úkolů na vstupu). Přípustnými řešeními jsou ta přiřazení, kdy čísla pracovníků jsou kladná celá a žádné překrývající se úkoly nemají stejného pracovníka. Účelovou funkcí pak je hodnota nejvyššího použitého čísla pracovníka, tuto funkci minimalizujeme. Formálně, jsou-li zadány intervaly $I_1, I_2, \ldots, I_k$ pracovních úkolů, pak $\mathcal{U} = \boldsymbol{Z}^k$ a složka z_i vektoru $\vec{z}$ určuje pracovníka pro úkol I_i, $\mathcal{P} = \{z \in \mathcal{U}, z > 0 : (i \neq j \wedge I_i \cap I_j \neq \emptyset) \Rightarrow z_i \neq z_j\}$ a $\eta(\vec{z}) = \max\{z_1, \ldots, z_k\}$. Promyslete si to sami!

V Úloze 1.4 je univerzem prostor m-složkových binárních vektorů, $\mathcal{U} = \boldsymbol{Z}_2^m$, kde m je počet hran daného grafu. Složka z_i vektoru říká, zda i-tou hranu grafu vybíráme do kostry. Přípustná řešení jsou tvořena acyklickými podmnožinami hran a účelová funkce je $\eta(\vec{z}) = \sum_{i=1}^{m} z_i \cdot w(e_i)$, tj. součet vah vybraných hran.