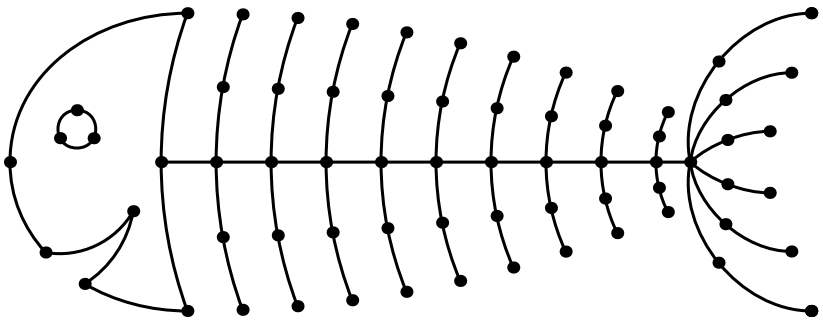
Kapitola 1

Minimální kostra



Motivace 1: (prohrnování silnic) V království je N měst a některá z nich jsou spojena přímou silnicí. Křižovatky jsou pouze ve městech. Mezi některými městy přímá silnice nevede, ale z každého města se dá po silnicích dostat do libovolného jiného. V noci se přihnala se pohádková vánice a zasněžila všechny silnice v celém království. Napadlo až metr sněhu. Odhrabovačů je málo a takovou sněhovou nadílku budou odklízet ještě hodně dlouho. Rozhodněte, které silnice se mají odhrabat jako první, aby mezi každýma dvěma městy vedla sjízdná silnice.

Motivace 2: (elektrifikace) Vysoko v tibetských horách se rozhodli začít využívat výhod moderního světa[1] a chtějí provést elektrifikaci všech N vesnic krásného údolí. Dráty se nesmí větvit jinde než ve vesnici. Znají vzdálenost mezi každýma dvěma vesnicemi, ale nevědí, jak natahat dráty elektrického vedení, aby jich spotřebovali co nejméně. Poradíte jim? Větší množství drátů než je nejmenší možné jim rada starších pro ochranu životního prostředí nepovolí.

Úkol: (grafový pohled) Je dán neorientovaný graf $G = (V, E)$ s ohodnocením hran $c : E \to \mathbb{R}$. Najděte kostru $T \subseteq G$ s nejmenším ohodnocením hran, tj. s nejmenším $\sum_{e \in T} c(e)$. Výstupem bude množina hran $M = E(T)$ tvořících kostru.

Místo ohodnocení hrany $c(e)$ často používáme pojem cena hrany. Nejdražší, respektive nejlevnější hrana je ta s největším, respektive nejmenším ohodnocením.[2]

Připomeňte si (viz kapitola **??** o grafech a stromech), že *strom* T je souvislý graf bez kružnic. Strom na všech vrcholech je také minimální souvislý podgraf (přidáním libovolné hrany k T vznikne kružnice). Mezi libovolnýma dvěma vrcholy stromu T

[1] Některé zdroje uvádějí, že k tomu byli donuceni Čínou.

[2] Často se také ohodnocení hrany nazývá váha hrany. Potom můžeme mluvit o nejtěžší, nejlehčí hraně.

vede jednoznačně určená cesta. Jednoznačnou cestu z vrcholu x do vrcholu y ve stromě T značíme xTy. *Kostra* grafu G je strom $T \subseteq G$ na všech n vrcholech. Také si připomeňte operace přidání a odebrání hrany z grafu G. Výsledky těchto operací značíme $G + e$, $G - e$.

Pro jednoduchost nebudeme v následujícím textu rozlišovat mezi množinou hran $M \subseteq E$ a podgrafem G obsahujícím právě hrany M. *Řez* určený množinou $A \subseteq V$ je množina hran $\delta(A) = \{uv \in E \,|\, u \in A \;\&\; v \notin A\}$. Každá cesta z $u \in A$ do $v \notin A$ prochází přes řez $\delta(A)$ (obsahuje hranu řezu). Platí, že graf je nesouvislý právě tehdy, když existuje řez $\delta(C) = \emptyset$ pro $\emptyset \neq C \subset V$.

Kostru v souvislém grafu najdeme jednoduše pomocí průchodu grafu do hloubky, protože DFS strom souvislého grafu je kostra. To dokazuje, že každý souvislý graf má alespoň jednu kostru. Taková kostra ovšem ještě nemusí být minimální, při hledání minimální kostry musíme postupovat malinko chytřeji. Pokud v zadání dostaneme nesouvislý graf, tak chceme najít řez $\delta(C) = \emptyset$ pro $C \neq \emptyset$, který je důkazem, že graf je nesouvislý a tedy že v grafu žádná kostra neexistuje. Stačí vzít řez určený jednou komponentou souvislosti.

Pozorování (o prohazování hran): Přidáním hrany e do kostry T vznikne právě jedna kružnice C. Vyhodíme-li z kružnice C libovolnou hranu f, dostaneme graf $T + e - f$, který je opět kostrou. Pokud $c(e) < c(f)$, tak bude mít nová kostra menší cenu.

Podobných pozorování využijeme v následujících algoritmech. Nejprve si ukážeme meta-algoritmus a z něj pak snadno odvodíme ostatní algoritmy pro hledání minimální kostry.

1.1 Základní meta-algoritmus

Množina hran $M \subseteq E$ je *rozšířitelná* do minimální kostry, pokud existuje minimální kostra T obsahující hrany M.

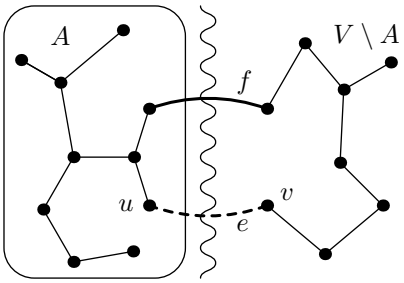
Meta-algoritmus: Postupně budeme konstruovat množinu M rozšířitelnou do minimální kostry. Začneme s prázdnou množinou M. Podle následujícího lemmatu 1 (o existenci řezu) najdeme řez neobsahující hranu M a podle lemmatu 2 (o nejlevnější hraně řezu) do kostry přidáme nejlevnější hranu tohoto řezu. Tento krok zopakujeme $(n-1)$-krát až skončíme s minimální kostrou.

Lemma 1 (o existenci řezu) *Nechť $M \subseteq E$ je množina rozšířitelná do minimální kostry, která ještě není minimální kostrou, pak existuje řez $\delta(A) \neq \emptyset$ takový, že $\delta(A) \cap M = \emptyset$.*

Důkaz: Zvolme si libovolný vrchol x a nechť A je množina vrcholů, do kterých z x vede cesta po hranách ležících v M. Jinými slovy A je komponenta souvislosti M obsahující x. Protože M není minimální kostrou, tak existuje vrchol $y \notin A$. Protože graf G je souvislý, tak existuje cesta xPy. Jelikož $x \in A$ a $y \notin A$, tak cesta xPy obsahuje hranu $e \in \delta(A)$. Ta dokazuje že $\delta(A) \neq \emptyset$. ∎

Následující lemma o nejlevnější hraně neříká nic složitějšího, než že nejlevnější hrana každého řezu patří do minimální kostry. Problémek ovšem nastane, pokud je těch nejlevnějších hran v řezu více. Potom do kostry můžeme dát kteroukoliv z nich, ale nejvýše jednu. Pojem rozšířitelnost do minimální kostry hlídá, jestli už jsme do M nepřidali jinou nejlevnější hranu téhož řezu (řez nesmí obsahovat žádnou hranu M).

Lemma 2 (o nejlevnější hraně řezu) *Nechť $M \subseteq E$ je množina rozšířitelná do minimální kostry a e je nejlevnější hrana řezu $\delta(A)$ splňujícího $\delta(A) \cap M = \emptyset$. Pak je i množina $M \cup \{e\}$ rozšířitelná do minimální kostry.*



Důkaz: M je rozšiřitelná do minimální kostry a proto existuje minimální kostra T obsahující hrany M. Pokud $e \in T$ tak jsme hotovi. Podívejme se na opačný případ $e \notin T$. Označme si konce hrany e pomocí u a v. Mezi u a v existuje v T jednoznačně určená cesta uTv. Jelikož $u \in A$ a $v \notin A$, tak na cestě uTv existuje hrana $f \in \delta(A)$. Cesta uTv spolu s e tvoří kružnici, proto i $T' = T - f + e$ tvoří kostru. Navíc $c(T') = c(T) - c(f) + c(e)$. Protože $c(e) \leq c(f)$, je i $c(T') \leq c(T)$ a T' je minimální kostra obsahující $M \cup \{e\}$. ∎

Uvedený meta-algoritmus je konečný, protože v každém kroku přidá do M jednu hranu a každá kostra má právě $n-1$ hran. Podle indukce a lemmatu 2 (o nejlevnější hraně řezu) bude množina M opravdu minimální kostrou.

Všechny používané algoritmy na nalezení minimální kostry se liší akorát tím, jakým způsobem procházejí vhodné řezy a také tím, jak v nich najdou nejlevnější hranu. Stačí uvažovat řezy určené jednou nebo více komponentami souvislosti M. Ostatní řezy obsahují hranu M a proto nevyhovují předpokladům lemmatu 2. Ukážeme si tři přístupy. Hladový (Kruskalův), Primův (Jarníkův) a Borůvkův algoritmus.

1.2 Kruskalův hladový algoritmus

Kruskalův algoritmus je nejjednoduší implementací meta-algoritmu. Zjenodušil si práci s výběrem nejlevnější hrany řezu tím, že prochází hrany v pořadí podle jejich velikosti. Každou procházenou hranu zkusí přidat do kostry. Pokud hrana $e = uv$ spojuje různé komponenty souvislosti množiny M, tak přidá hranu e do kostry. Aniž to algoritmus tuší, nalezl zároveň i vhodný řez pro meta-algoritmus. Je to řez určený komponentou souvislosti C_u nebo C_v, kde C_v značí komponentu souvislosti obsahující vrchol v. Hrana e je zaručeně nejlevnější hranou řezu $\delta(C_v)$, protože hrany procházíme podle velikosti a všechny menší hrany už leží uvnitř komponent souvislosti M. Správnost Kruskalova algoritmu plyne z předchozích lemmat meta-algoritmu.

Kruskalův hladový algoritmus:

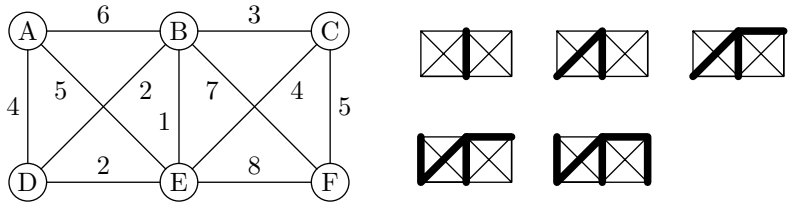
1. Seřaď hrany podle velikosti $e_1 \leq e_2 \leq \cdots \leq e_m$.
2. $M := \emptyset$. Procházej hrany v pořadí podle velikosti. Pokud hrana e_i spojuje různé komponenty souvislosti M, tak ji přidej do M a sjednoť příslušné komponenty souvislosti.
3. Na konci je M minimální kostra.

Algoritmus na začátku třídí m hran, což mu trvá $\mathcal{O}(m \log m)$. Zjišťování, jestli dva vrcholy patří do stejné komponenty souvislosti, případně sjednocení komponent, není nic jiného než Union-Find problém (viz kapitola **??**). Proto v bodě 2 provede algoritmus m-krát operaci `FIND` a $(n-1)$-krát operaci `UNION`. To mu zabere čas $\mathcal{O}((m+n)\log n)$. Operace `UNION` a `FIND` umíme realizovat i tak, aby druhá fáze trvala jen čas $\mathcal{O}((m+n)\alpha(n))$, kde $\alpha(n)$ je inverzní Ackermanova funkce. Ale díky časové složitosti první fáze to není potřeba, celkovou časovou složitost algoritmu $\mathcal{O}(m \log n)$ to nezmenší.[3]

[3]Můžeme si dovolit být líní a implementovat Union-Find pomocí pole s řetízky. Amortizovaná časová složitost UNION bude $\mathcal{O}(\log n)$.

Pokud dostaneme hrany už v setříděném pořadí, tak je časová složitost jenom $\mathcal{O}((m+n)\alpha(n))$. Tedy prakticky lineální ve velikosti grafu.

Příklad: Na následujícím obrázku vlevo je ohodnocený graf, ve kterém chceme najít minimální kostru.



Hrany si seřadíme podle velikosti a dostaneme pořadí BE, BD, DE, BC, AD, CE, AE, CF, AB, BF, EF.[4] V tomto pořadí hrany probíráme a zkoušíme je přidat do kostry. Pokud by přidáním hrany vznikla kružnice, tak ji nepřidáme. Průběh algoritmu je zachycen na obrázcích vpravo.

1.3 Jarníkův, Primův algoritmus

Jarníkův algoritmus si udržuje jen jednu komponetu souvislosti a tu postupně rozšiřuje. Začne s komponentou souvislosti, která obsahuje jen počáteční vrchol r. V každém kroku tuto komponentu[5] $T = (R, M)$ rozšíří o nejlevnější hranu řezu určeného touto komponentou.

Jarníkův/Primův algoritmus:

1. $T := (R, M)$, $R := \{r\}$, $M := \emptyset$.
2. V každém kroku přidáme do M nejlevnější hranu e řezu $\delta(R)$ a do R přidáme druhý konec hrany e neležící v R.
3. Po $n-1$ krocích se zastavíme, máme minimální kostru T.

Správnost algoritmu opět plyne z lemmat o meta-algoritmu a z faktu, že přidáváme nejlevnější hranu řezu určeného jedinou komponentou souvislosti M.

Implementace

Při realizaci algoritmu nemusíme probírat všechny hrany řezu. Pro každý vrchol, který ještě není v R, si budeme pamatovat hodnotu $\texttt{d}[v]$ = nejmenší cena hrany vedoucí z v do množiny R, a také $\texttt{odkud}[v]$ ta hrana vede. Na začátku bude hodnota všech $\texttt{d}[v] = \infty$, jen počáteční vrchol r bude mít $\texttt{d}[r] = 0$. Místo probírání všech hran řezu $\delta(R)$ stačí projít všechny hodnoty $\texttt{d}[v]$ a vybrat z nich tu nejmenší. Abychom v průběhu algoritmu udržovali obě pole aktuální, tak po každém přidání nového vrcholu do R zkontrolujeme, jestli z něj nevede levnější hrana do vrcholů $V \setminus R$ a případně aktualizujeme hodnoty $\texttt{d}[v]$ a $\texttt{odkud}[v]$.

```
Jarník, Prim:
    d[r] := 0 a ∀ v ∈ V \ {r} : d[v] := ∞
    ∀ v ∈ V : odkud[v] := nil
    vytvoř prioritní frontu H z vrcholů V s prioritami d[v]
    while fronta H neprázdná do
        v := DELETE_MIN(H)
        přidej hranu {v, odkud[v]} do kostry
```

[4] Hrany stejné velikosti jsme seřadili abecedně.

[5] Komponenta T je stromem. Značíme ji jako podgraf $T = (R, M)$ s vrcholy R a hranami M.

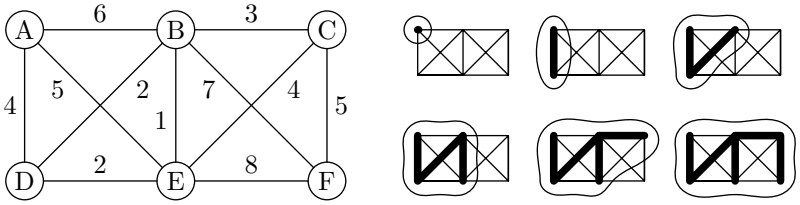
```
for each vw ∈ E do
   if d[w] > c(vw) then
      d[w] := c(vw)
      odkud[w] := v
      DECREASE_KEY(H, w)
```

Algoritmus je hodně podobný Dijkstrovu algoritmu pro hledání nejkratší cesty v grafu (viz. strana **??**). Diskuse o možných realizacích prioritní fronty a jejich vhodnosti pro konkrétní grafy je naprosto stejná jako u Dijkstrova algoritmu. Proto zde uvedeme jen výsledky.

Časová složitost Jarníkova algoritmu při realizaci prioritní fronty v poli je $\mathcal{O}(n^2+m)$. Při realizaci binární haldou je $\mathcal{O}((n+m)\log n)$. Při použití d-regulární haldy s volbou $d \approx m/n$ (průměrný stupeň grafu) dostaneme pro velmi řídké grafy (tj. s $m = \mathcal{O}(n)$) časovou složitost $\mathcal{O}(n \log n)$ (stejně jako u binární haldy) a pro ostatní grafy lineární časovou složitost.

Příklad: Zkusme najít minimální kostru grafu, který je na následujícím obrázku vlevo, tentokráte ale podle Jarníkova algoritmu. Vrcholy i hrany grafu budeme procházet v abecedním pořadí. Jednotlivé kroky algoritmu jsou na obrázcích vpravo. Vrcholy patřící do množiny R jsou na obrázcích zakroužkovány.

Jako počáteční vrchol si vybereme A. V prvním kroku mají všechny vrcholy až na A hodnotu $\mathtt{d}[v] = \infty$. Proto si jako vrchol s nejmenší hodnotou $\mathtt{d}[\,\cdot\,]$ vybereme vrchol A a přidáme ho do R. Při aktualizaci $\mathtt{d}[\,\cdot\,]$ snížíme $\mathtt{d}[B] = 6$, $\mathtt{d}[D] = 4$, $\mathtt{d}[E] = 5$ a spolu s tím upravíme $\mathtt{odkud}[\,\cdot\,]$. Ve druhém kroku si za vrchol s nejnižším $\mathtt{d}[\,\cdot\,]$ vybereme vrchol D. Přidáme ho do R a hranu $AD = \{D, \mathtt{odkud}[D]\}$ přidáme do kostry. Dále postupujeme podobně, až dokud nedostaneme minimální kostru.



1.4 Jednoznačnost minimální kostry

Některé algoritmy pro hledání minimální kostry fungují pouze na grafech, ve kterých je minimální kostra určena jednoznačně. Nemůžeme se tohoto předpokladu nějak zbavit, aby dané algoritmy fungovaly na všech grafech? Ano můžeme. Vysvětlíme si jak.

Který krok není jednoznačný?

Podívejme se na průběh meta-algoritmu. Do budované kostry postupně přidáváme nejlevnější hranu každého řezu. Bohužel se ale může stát, že je těch nejlevnějších hran více (například v grafu, kde mají všechny hrany stejné ohodnocení). Máme možnost volby. Rozhodnutí se pro jednu nejlevnější hranu může zamezit přidání ostatních nejlevnějších hran. Pro každou volbu můžeme dostat jinou minimální kostru.

Příklad: (minimální kostra v kružnici C_n s konstantním ohodnocením hran) Pro libovolnou hranu $e \in C_n$ existuje minimální kostra, která e obsahuje, ale i minimální kostra, která e neobsahuje. Zkuste hledat minimální kostru C_n meta-algoritmem. Prozkoumejte, kolik hran z nalezených řezů bude nakonec patřit do minimální kostry.

Lemma 3 (o nejlevnější hraně řezu) *Nechť G je souvislý graf a $\delta(A)$ libovolný řez. Pokud je nejlevnější hrana e řezu $\delta(A)$ určena jednoznačně, tak e patří do minimální kostry.*

Důkaz: Důkaz tohoto lemmatu je téměř shodný s důkazem lemma 2.

Graf G je souvislý, takže má minimální kostru T. Pokud $e \in T$, tak jsme hotovi. Podívejme se na opačný případ a předpokládejme pro spor, že $e \notin T$. Nechť $e = uv$. Přidáním e do T vznikne kružnice, která je tvořena hranou uv a cestou uTv. Vrchol $u \in A$ a $v \notin A$ a proto na cestě uTv existuje hrana $f \in \delta(A)$. Protože e je nejlevnější hrana řezu $\delta(A)$, tak $c(e) < c(f)$. Graf $T' := T - f + e$ je opět kostra. Navíc $c(T') = c(T) - c(f) + c(e) < c(T)$. To je spor s minimalitou kostry T. ∎

Výhoda lemmatu 3 je, že nepotřebuje pojem rozšířitelnosti do minimální kostry.

Pokud mají všechny hrany v grafu různá ohodnocení, tak v každém řezu existuje právě jedna nejlevnější hrana. Potom je postup meta-algoritmu určen jednoznačně.

Lemma 4 (jednoznačnost minimální kostry) *Pokud mají všechny hrany v grafu G různá ohodnocení, tak je minimální kostra určena jednoznačně.*

Zkuste si toto lemma o jednoznačnosti minimální kostry dokázat. S využitím lemmatu 3 (o nejlevnější hraně řezu) je to lehké cvičení.

Jak si zajistit jednoznačnost?

Podle lemmatu 4 víme, že když mají všechny hrany v grafu různá ohodnocení, tak už je minimální kostra určena jednoznačně. Proto si uspořádání na hranách podle ohodnocení rozšíříme do lineálního uspořádání[6] tím, že každé hraně přidáme jednoznačné ohodnocení $c_2(e)$. Hrany budou ohodnoceny dvojicí $(c(e), c_2(e))$. Hrany budeme porovnávat lexikograficky—nejprve podle původního ohodnocení a pokud se hodnoty obou hran shodují, tak podle druhého pomocného uspořádání.

Jaké lineární uspořádání na hranách můžeme zvolit jako pomocné? První možností je jednoznačné očíslování hran (například indexem pole, ve kterém jsou uloženy). Druhou možností je, že máme vrcholy očíslovány čísly 1 až n. Potom porovnáme hrany $e = xy$, $f = uv$ lexikograficky. Předpokládejme, že jsme si hrany označili tak, že $x < y$ a $u < v$. Platí $e <_{LEX} f$ právě tehdy když $x < u$ nebo když $x = u$ a $y < v$.

1.5 Borůvkův algoritmus

Borůvkův algoritmus funguje správně pouze na ohodnocených grafech, ve kterých je minimální kostra určena jednoznačně. (Jak si v každém grafu zajistit jednoznačnost minimální kostry je popsáno v předchozí sekci 1.4). Výhodou Borůvkova algoritmu je, že lze dobře paralelizovat.

Borůvkův algoritmus si v průběhu udržuje les T (množinu hran rozšířitelnou do minimální kostry). Na začátku je les tvořen izolovanými vrcholy (neobsahuje žádnou hranu). V každé iteraci vybereme pro každý strom T_i (komponentu souvislosti) lesa T nejlevnější hranu řezu $\delta(T_i)$ a na závěr iterace ji přidáme do T. Proč ji nepřidáme rovnou? Musíme si dát pozor, abychom nějakou hranu nepřidali dvakrát. Mohlo by se stát, že jedna hrana bude spojovat dvě komponenty souvislosti a pro obě komponenty bude vybrána jako nejlevnější.

Proč algoritmus vyžaduje jednoznačnost minimální kostry? Jinak by se mohlo stát, že pro komponetu T_u vybereme nejlevnější hranu řezu e_u a pro komponentu T_v jinou nejlevnější hranu e_v téhož řezu. Po přidání obou hran do kostry by vznikla kružnice.

[6] V lineárním uspořádání umíme pro každé dvě hrany určit, která z nich je menší.

Borůvka:
  $T := (V, \emptyset)$
  `while` T není souvislý `do`
    Pro každou komponentu souvislosti T_i grafu T
      vyber nejlevnější hranu e_i řezu $\delta(T_i)$.
    Všechny hrany e_i přidej do T.

Lemma 5 *V každé iteraci Borůvkova algoritmu klesne počet komponent souvislosti aspoň na polovinu.*

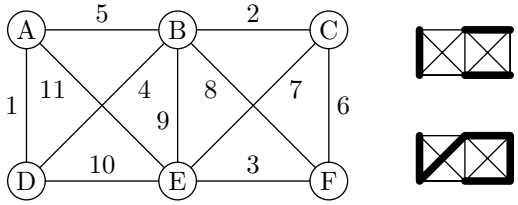
Lemma platí, protože se každá komponenta souvislosti T_i spojí s jinou komponentou souvislosti přes hranu e_i. Jako důsledek dostáváme, že Borůvkův algoritmus provede nejvýše $\log n$ iterací.

Lemma 6 *Borůvkův algoritmus najde minimální kostru.*

Důkaz: Borůvkův algoritmus opět funguje podle meta-algoritmu. Tentokrát ale s tou změnou, že používáme variantu lemmatu "o nejlevnější hraně řezu" pro grafy, jejich hrany mají vzájemně různé ohodnocení. Podle lemmatu 3 patří každá vybraná hrana e_i do jednoznačně určené minimální kostry. ∎

Zkuste si rozmyslet, proč během přidávání hran nevznikne kružnice.

Příklad: Najděte pomocí Borůvkova algoritmu minimální kostru grafu na následujícím obrázku. Jednotlivé iterace jsou na obrázcích vpravo. V první iteraci pro každý vrchol vybíráme nejlevnější hranu, která z něj vede. Pro vrchol A vybereme hranu AD, podobně pro vrchol D vybereme hranu AD. Dále pro vrcholy B a C vybereme hranu BC a pro vrcholy E a F hranu EF. V druhé iteraci podobně vybíráme nejlevnější hranu, která vede z každé „tlusté" komponenty souvislosti někam ven. Pro komponentu $\{A, D\}$ vybereme hranu DB, pro komponentu $\{B, C\}$ také hranu DB a pro komponentu $\{E, F\}$ hranu CF.



Implementace

Nechť T_v označuje komponentu souvislosti, ve které leží vrchol v. V každé iteraci chceme pro každou komponentu T_v nalézt nejlevnější hranu, která z ní vede ven. V každé komponentě souvislosti T_v zvolíme jednoho reprezentanta $\mathtt{r}[v] \in T_v$. Nejlevnější hranu vedoucí z komponenty T_v si zapamatujeme v položce $\mathtt{emin}[\,\cdot\,]$ u reprezentanta komponenty $\mathtt{r}[v]$. Iteraci provedeme následovně:

- Vytvoříme pomocný graf G_{pom} jehož vrcholy jsou reprezentanti komponent souvislosti. Postupně projdeme všechny hrany $e = xy \in E$ a zkusíme je přidat do pomocného grafu jako hrany $\mathtt{r}[x]\mathtt{r}[y]$. Při přidávání hran mohou vzniknout smyčky a násobné hrany. Smyčky rovnou vyhodíme a z násobných hran přidáme do pomocného grafu pouze tu nejlevnější.

  V Borůvkově algoritmu můžeme být velmi líní a nemusíme si z pomocného grafu skoro nic pamatovat. Zapamatujeme si pro každého reprezentanta komponenty pouze nejlevnější hranu $\mathtt{emin}[\mathtt{r}[v]]$, která z reprezentanta vede. Nejlevnější hrana vedoucí z reprezentanta $\mathtt{r}[v]$ odpovídá nejlevnější hraně řezu $\delta(C_v)$.

- Za každého reprezentanta $\mathtt{r}[v]$ přidáme do minimální kostry hranu $\mathtt{emin}[\mathtt{r}[v]]$. (Projdeme všechny vrcholy a najdeme reprezentanty.)
- Před další iterací musíme aktualizovat pole $\mathtt{r}[\,\cdot\,]$. Protože se nám mohlo sjednotit několik komponent souvislosti do jedné, je potřeba provést sjednocení komponent opatrněji než postupným sjednocováním dvojic komponent a přeznačováním $\mathtt{r}[u]$. Jinak by to mohlo trvat příliš dlouho. Pole $\mathtt{r}[\,\cdot\,]$ vyplníme úplně znova tak, že nalezneme komponenty souvislosti v grafu s hranami M (průchod do hloubky).[7]

Mohlo by vás napadnout, proč si neušetříme práci s aktualizací pole $\mathtt{r}[\,\cdot\,]$ a proč nepoužijeme řešení Union-Find problému pomocí přepojování stromečků s kompresí cestiček. Je pravda, že by se výrazně zjednodušila aktualizace pole $\mathtt{r}[\,\cdot\,]$, ale na druhou stranu by hledání nejlevnějších hran vycházejících z každé komponenty trvalo $\mathcal{O}(m\alpha(n))$. To je víc než $\mathcal{O}(m)$.

Výše popsaným postupem můžeme každou iteraci provést v čase $\mathcal{O}(m)$. Celkem bude Borůvkův algoritmus trvat čas $\mathcal{O}(m \log n)$. Výhodou Borůvkova algoritmu je, že v každé iteraci můžeme počítat nejlevnější hrany e_i paralelně.

1.6 Kontraktivní algoritmus

Kontraktivní algoritmus[8] vychází z Borůvkova algoritmu, malinko ho vylepšuje a díky tomu dosáhneme o něco lepší časové složitosti.

V každé iteraci Borůvkova algoritmu jsme konstruovali pomocný graf, ale u každého reprezentanta jsme si pamatovali jen nejlevnější hranu, která z něj vede. Tentokrát ten graf zkonstruujeme pořádně. Postupně projdeme všechny hrany $e = xy \in E$ a přidáme je do pomocného grafu jako hrany $\mathtt{r}[x]\mathtt{r}[y]$. Při přidávání hran mohou vzniknout smyčky a násobné hrany. Smyčky vyhodíme a z násobných hran si budeme pamatovat jen tu nejlevnější.[9] Tímto postupem dostaneme v k-té iteraci graf G_k. Jinými slovy, graf G_k vznikne z grafu G kontrakcí komponent souvislosti podgrafu $T \subseteq G$.

V k-té iteraci se pouze některé komponenty spojili do jedné. Proto nemusíme graf G_k konstruovat z G, ale můžeme ho konstruovat z menšího grafu G_{k-1}.

Nechť n_i označuje počet vrcholů grafu G_i a m_i jeho počet hran. Při postupném vytváření grafů $G_0, G_1, \ldots$ bude i-tá iterace Borůvkova algoritmu trvat čas $\mathcal{O}(m_i)$.

Lemma 7 *Na grafu s různým ohodnocením hran je časová složitost kontraktivního Borůvkova algoritmu* $\mathcal{O}(\min\{n^2, m \log n\})$.

Důkaz: Odhad $\mathcal{O}(m \log n)$ jsme si ukázali i pro nekontraktivní Borůvkův algoritmus. V kontraktivní variantě navíc ušetříme nějaký čas a tak odhad platí. Podívejme se na důkaz odhadu $\mathcal{O}(n^2)$. Podle lemmatu 5 klesne v každé iteraci počet komponent souvislosti aspoň na polovinu. Proto $n_i < n/2^i$. V každém grafu je $m_i < n_i^2$ a proto $m_i < n^2/4^i$. Každá iterace kontraktivního Borůvkova algoritmu trvá $\mathcal{O}(m_i)$ a tedy celkem algoritmus trvá čas $\mathcal{O}(\sum_i m_i) = \mathcal{O}(\sum_i n^2/4^i) = \mathcal{O}(n^2)$. ∎

Lemma 8 *Kontraktivní Borůvkův algoritmus má v rovinných grafech časovou složitost* $\mathcal{O}(n)$.

[7] Aktualizaci pole $\mathtt{r}[\,\cdot\,]$ můžeme udělat i rychleji. Nejprve nalezneme reprezentanty komponent souvislosti v pomocném grafu G_{pom}, který obsahuje pouze hrany přidávané do kostry. Pro každý vrchol $w \in V(G_{pom})$ pomocného grafu si spočítáme jeho reprezentanta do pole $\mathtt{rpom}[w]$. Průchodem do hloubky to zvládneme v lineárním čase v počtu reprezentantů. Samotná aktualizace proběhne tak, že pro všechny $v \in V$ provedeme $\mathtt{r}[v] := \mathtt{r}[\mathtt{rpom}[v]]$.

[8] Jestli je mi to dobře známo, tak pochází od Mareše [3].

[9] Rozmyslete si, jak to dělat, abychom to stihli v čase $\mathcal{O}(m)$.

Důkaz: Platí následující fakt. Kontrakcí a mazáním hran rovinného grafu vznikne rovinný graf. Víme, že $n_i \leq n/2^i$. V každém rovinném grafu $G_R = (V_R, E_R)$ platí $|E_R| \leq 3|V_R| - 6$. Proto $m_i \leq 3n_i \leq 3n/2^i$. Součtem přes všechny iterace dostaneme celkový čas algoritmu $\mathcal{O}(\sum_i m_i) = \mathcal{O}(3n \sum_i 1/2^i) = \mathcal{O}(n)$. ∎

1.7 Červenomodrý meta-algoritmus*

Základní meta-algoritmus můžeme ještě zobecnit o vymazávání hran, které do minimální kostry určitě nepatří. Pro jednoduchost předpokládejme, že všechny hrany mají různé ohodnocení a tím pádem že je minimální kostra určena jednoznačně. Vyhneme se tím pojmu "rozšířitelnost do minimální kostry" a výklad se zjednoduší.

Na začátku jsou všechny hrany grafu bezbarvé. Postupně je budeme barvit na červeno nebo na modro podle následujících lemmat. Barvíme tak dlouho, dokud není vše obarveno a dokud lze některé lemma aplikovat. Červené hrany $\check{C} \subseteq E$ jsou ty, které do kostry určitě nepatří. Klidně je místo barvení můžeme z grafu vymazávat. Minimální kostru budeme hledat v nečervených hranách. Modré hrany jsou ty, které do minimální kostry určitě patří.

Lemma 9 (červené lemma) *Je-li C kružnice v G, pak nejdražší hrana na kružnici e nepatří do minimální kostry. Hranu e obarvíme na červeno.*

Důkaz: Hrana e nepatří do žádné minimální kostry. Předpokládejme pro spor, že T je minimální kostra obsahující nejdražší hranu $e = xy$ kružnice C. Graf $T - e$ má dvě komponenty souvislosti. Protože x leží v jedné a y v druhé komponentě, tak na cestě $P = C \setminus e$ vedoucí z x do y existuje hrana f propojující obě komponenty. Proto je $T' = T + f - e$ kostra. Navíc $c(e) > c(f)$ a tedy $c(T') = c(T) + c(f) - c(e) < c(T)$. To je spor s minimalitou kostry T. ∎

Lemma 10 (modré lemma) *Nechť e je nejlevnější hrana řezu $\delta(A)$. Pak hrana e patří do minimální kostry. Hranu e obarvíme na modro.*

Důkaz: Důkaz tohoto lemmatu je shodný s důkazem lemma 3. (Předpokládejme pro spor, že existuje minimální kostra neobsahující e. Výměnou jedné hrany za e dostaneme levnější kostru a to je spor.) ∎

Lemma 11 (bezbarvé lemma) *Pokud by některá hrana zůstala neobarvená, tak lze ještě aplikovat červené nebo modré lemma.*

Důkaz: Označme neobarvenou hranu $e = xy$. Nechť W je množina vrcholů, do kterých se lze dostat z x po modrých hranách.

Buď $y \in W$, ale potom existuje cesta z modrých hran vedoucí z x do y a ta spolu s hranou $e = xy$ tvoří kružnici C. Každá kružnice obsahuje alespoň jednu červenou hranu, tu nejdražší. Všechny hrany kružnice C kromě e jsou modré a proto hrana e musí být nejdražší hranou kružnice C a můžeme ji podle červeného lemmatu obarvit na červeno.

Pokud $y \notin W$, tak je řez $\delta(W)$ určitě neprázdný (obsahuje hranu e) a neobsahuje žádnou modrou hranu. Proto na řez $\delta(W)$ můžeme aplikovat modré lemma. ∎

Červenomodrý algoritmus je konečný, protože v každém kroku obarví jednu hranu, hrany nepřebarvuje a všech hran je m. Podle lemmat najde minimální kostru, která bude tvořená modrými hranami.

1.8 Přehled algoritmů pro minimální kostru

Kruskalův hladový	$\mathcal{O}(m \log n)$
Kruskalův hladový (setříděné hrany)	$\mathcal{O}((n+m)\alpha(n))$
Jarníkův, Primův (pole)	$\mathcal{O}(n^2+m)$
Jarníkův, Primův (halda)	$\mathcal{O}((n+m)\log n)$
Borůvkův	$\mathcal{O}(m \log n)$

Asymptotické časové složitosti algoritmů pro nalezení minimální kostry se téměř neliší. Odhady můžeme zlepšit ve speciálních případech, kdy dostaneme dodatečná omezení na vstupní graf (více uvidíte v příkladech na konci kapitoly).

Nejrychlejší algoritmus na minimální kostru pochází od Chazelle [1], který s využitím SoftHeaps navrhnul algoritmus s časovou složitost $\mathcal{O}((n+m)\alpha(n))$. Pettie, Ramachandran [4] navrhli algoritmus, který už je optimální. Jen se dosud nepodařilo vyčíslit jeho časovou složitost. Je to někde mezi $\mathcal{O}(m)$ a $\mathcal{O}(m\alpha(n))$ (což už je prakticky jen multiplikativní konstanta).

Chong, Han, Lam [2] ukázali, že paralelní algoritmus s časovou složitostí $\mathcal{O}(\log n)$.

Hezky sepsaný přehled všech možných algoritmů týkajících se minimálních koster najdete v dizertační práci Martina Mareše [3].

1.9 Aplikace minimálních koster

Ukážeme si pár netriviálních aplikací, ve kterých se nám hodí to, že umíme najít minimální kostru grafu.

1.9.1 Steinerovy stromy

V motivačních problémech na začátku kapitoly o minimálních kostrách jsme si reálnou situaci poněkud zjednodušili. Při elektrifikaci Tibetu jsme zakázali, aby se dráty s elektrickým vedením větvily jinde než ve vesnicích. Pokud to povolíme, tak nám bude k propojení vesnic stačit mnohem méně drátu. Příklad 4 takových vesnic je na následujícím obrázku.

Jak určit vhodná místa pro větvení elektrického vedení? Zkuste se sami zamyslet nad tím, co taková místa musí splňovat. Kdy je takové místo optimální a kdy je naopak výhodnější ho posunout kousek vedle.

Předpokládejme, že už známe všechna možná místa větvení. V řadě úloh jsou tato místa známa. Například při prohrnování sněhu v reálné silniční síti chceme propojit všechna města. Kromě měst se silnice může větvit i na křižovatkách ležících mimo města. Graf silniční sítě má tedy dva druhy vrcholů, města která chceme vzájemně propojit a křižovatky, které odpovídají místům větvení. V optimálním řešení nám může po propojení všech měst zůstat spousta nedostupných křižovatek (cestu k nim není potřeba prohrnovat). To nás přivádí k následujícímu úkolu.

Úkol: (Steinerův strom) Je dán neorientovaný graf $G = (V, E)$ s nezáporným ohodnocením hran $c : E \to \mathbb{R}_+$. Vrcholy $V = R \cup S$ jsou dvou druhů, požadované R a Steinerovy S. Najděte strom $T \subseteq G$ s nejmenší cenou, který propojuje všechny požadované vrcholy. Cena stromu T se opět počítá jako $\sum_{e \in T} c(e)$.

Steinerův strom se od kostry liší tím, že nemusí použít všechny Steinerovy vrcholy. Nalezení optimálního Steinerova stromu patří mezi složité úlohy.[10] Není pro ně znám žádný polynomiální algoritmus.

Jediné, co umíme v polynomiálním čase spočítat, jsou přibližná řešení (viz. kapitola **??** o aproximačních algoritmech).

1.9.2 Aproximační algoritmus pro Steinerův strom

Následujcí aproximační algoritmus funguje pouze v grafech, jejichž ohodnocení hran splňuje *trojúhelníkovou nerovnost* (pro libovolné 3 hrany ab, bc, ac platí $c(ab) + c(bc) \geq c(ac)$).

Aproximační algoritmus:

1. Graf G obsahuje vrcholy $V = R \cup S$. Vytvoříme si pomocný úplný graf G', který obsahuje pouze požadované vrcholy R. Cena hrany $uv \in G'$ je cenou nejkratší cesty z u do v v grafu G. Grafu G' se říká *metrický uzávěr* grafu G. Graf G' už je úplným grafem (libovolná dvojice vrcholů určuje hranu). Cena Steinerova stromu v grafu G' je stejná nebo vyšší než cena Steinerova stromu v původním grafu G.

2. Najdeme minimální kostru T' v grafu G'. Ta je aproximací Steinerova stromu pro graf G'.

3. Některé hrany uv v kostře $T' \subseteq G'$ odpovídají v grafu G cestě spojující vrcholy u a v. Proto každou hranu $e \in T'$ nahradíme jí odpovídající cestou. Cesta má stejnou cenu, jako hrana, protože G' je metrickým uzávěrem G. Takto ale mohli vzniknout kružnice. Proto výsledný graf projdeme a přebytečné hrany odstraníme (opět nalezneme minimální kostru). Tím ještě snížíme cenu nalezeného řešení. Skončíme s kostrou $T \subseteq G$, která je aproximací Steinerova stromu v G.

Lemma 12 *Nechť OPT je cena optimálního Steinerova stromu T^* v grafu G. Cena stromu T nalezeného algoritmem je $OPT \leq c(T) \leq 2 \cdot OPT$.*

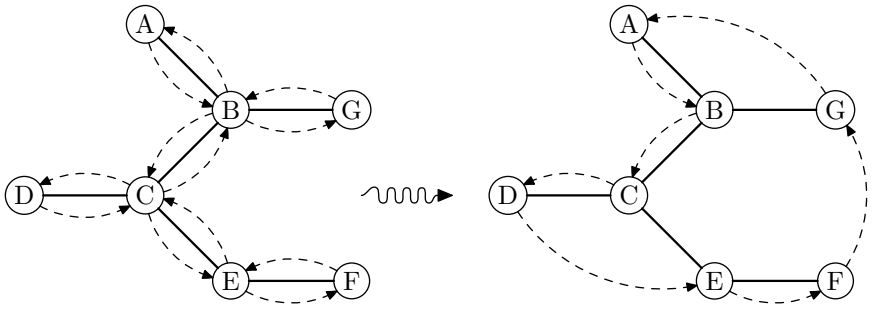
Důkaz: Algoritmem nalezený strom T je přípustným řešením úlohy. Proto $OPT \leq c(T)$.

Vezmeme optimální Steinerův strom T^* v grafu G. Zdvojíme hrany stromu T^* a nalezneme na nich uzavřený eulerovský tah W. Cena $c(W) = 2 \cdot OPT$.

Každý úsek tahu W, který prochází přes Steinerův vrchol u zkrátíme tak, že úsek $v \to u \to w$ procházející přes hrany vu, uw nahradíme zkratkou $v \to w$, která prochází pouze přes hranu vw. Protože hrany grafu splňují trojúhelníkovou nerovnost, dostaneme levnější tah. Zkracování tahu W budeme aplikovat tak dlouho, dokud nebudou všechny vrcholy tahu W patřit do požadovaných vrcholů R.

Potom tah projdeme ještě jednou a pomocí „zkratek“ vytvoříme Hamiltonovskou kružnici H. Uděláme to tak, že procházíme eulerovský tah. Pokud následující hrana tahu vede do vrcholu, který jsme už navštívili, tak půjdeme zkratkou do prvního z následujících vrcholů tahu, kde jsme ještě nebyli. Cena tahu mohla opět jen klesnout.

[10] Problém Steinerova stromu je NP-těžký a to i v případě, když se omezíme na grafy jejichž ceny hran splňují trojúhelníkovou nerovnost.

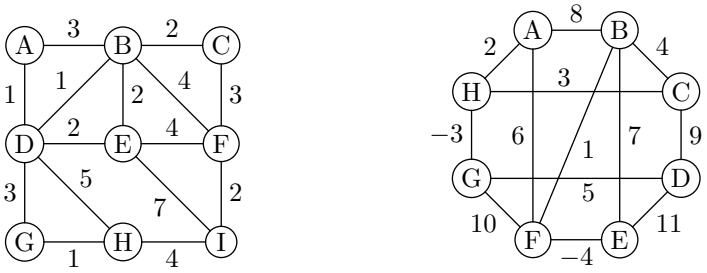


Vyhozením nejdražší hrany z kružnice H dostaneme strom T'^*. Cena stromu T'^* je $\leq 2 \cdot OPT$. Strom T'^* je kostrou v grafu G', ale nemusí být minimální kostrou. Cena minimální kostry je stejná a nebo menší. Protože T' je minimální kostra v G', dostáváme $c(T') \leq 2 \cdot OPT$. Ve 3. kroku algoritmu jsme z kostry $T' \subseteq G'$ vytvořili levnější kostru $T \subseteq G$. Proto je $c(T) \leq c(T') \leq 2 \cdot OPT$. ∎

1.10 Příklady

1.10.1 Přímé procvičení probraných algoritmů

1. V následujících grafech najděte minimální kostru:



   (a) Hladovým (Kruskalovým) algoritmem.

   (b) Jarníkovým (Primovým) algoritmem.

   (c) Borůvkovým algoritmem.

2. Najděte ohodnocený graf, který má jednoznačně určenou minimální kostru, ale jeho hrany nemají vzájemně různá ohodnocení.

3. Profesor Fishbone tvrdí, že pojem "rozšířitelnost do minimální kostry" není u základního meta-algoritmu vůbec potřeba a zbytečně meta-algoritmus komplikuje. Navrhuje meta-algoritmus, který postupně prochází libovolné řezy $\delta(W)$ a do minimální kostry přidá libovolnou nejlevnější hranu řezu $\delta(W)$. Meta-algoritmus skončí po přidání $n - 1$ hran, protože v ten moment bude mít minimální kostru.

   Rozhodněte, jestli má profesor Fishbone pravdu a své rozhodnutí dokažte.

4. (Test) Dostanete graf $G = (V, E)$ s cenami hran $c : E \to \mathbb{R}$. Následující tvrzení mohou, ale nemusí být pravdivé. Vždy buď dokažte, že je tvrzení pravdivé, nebo nalezněte protipříklad.

(a) Pokud má graf G více než $n-1$ hran, tak nejdražší hrana grafu určitě nepatří do minimální kostry.

(b) Jestli je e nejlevnější hranou v grafu G, tak určitě patří do minimální kostry.

(c) Pokud je nejlevnější hrana v grafu G určena jednoznačně (ostatní hrany jsou dražší), tak musí být obsažena v každé minimální kostře.

(d) Každá hrana, která je součástí minimální kostry, je nejlevnější hranou nějakého řezu.

(e) Pokud cyklus obsahuje pouze jednu nejlevnější hranu, tak tato hrana patří do minimální kostry.

(f) Nejkratší cesta mezi dvěma vrcholy určitě patří do minimální kostry.

(g) Strom nejkratší cesty obsahuje stejné hrany jako minimální kostra.

(h) Cesta P je r-levná, pokud je cena každé hrany na cestě P nejvýše r. Pokud graf G obsahuje nějakou r-levnou cestu mezi s a t, tak i každá minimální kostra T obsahuje r-levnou cestu sTt.

(i) Minimální kostra je souvislý podgraf takový, že součet cen jeho hran je nejmenší možný.

Správná odpověď je $114 = [011100010]_2$.

1.10.2 Na teorii

1. Je dáno n bodů v rovině. Definujme ohodnocení hran úplného grafu na těchto bodech tak, že ohodnocením hrany xy bude vzdálenost bodů x, y.

   (a) Ukažte, že maximální stupeň vrcholu v libovolné minimální kostře je nejvýše 6.

   (b) Ukažte, že existuje minimální kostra, která je rovinná (jejíž hrany se navzájem nekříží).

2. Dokažte, že neorientovaný graf G s n vrcholy a k komponentami souvislosti má alespoň $n-k$ hran.

3. Dostanete neorientovaný graf $G = (V, E)$, jehož hrany jsou ohodnoceny navzájem různými kladnými čísly. Dále dostanete vrchol $s \in V$. Je strom nejkratší cesty z s do všech ostatních vrcholů shodný s minimální kostrou? Pokud ano, tak uveďte důvod. Pokud ne, tak uveďte příklad grafu, kde se liší.

4. Nechť G je neorientovaný ohodnocený graf, $H \subseteq G$ jeho podgraf a T minimální kostra G. Ukažte, že $T \cap H$ je obsaženo v nějaké minimální kostře H.

1.10.3 Na algoritmy

1. (Maximální kostra) Najděte maximální kostru v grafu. Jaká je časová složitost vašeho algoritmu ve srovnání s algoritmy na nalezení minimální kostry?

2. (Varianty či nevarianty minimální kostry) Dostanete graf $G = (V, E)$ s ohodnocením hran $c : E \to \mathbb{R}$. Nalezněte kostru T s minimálním

   (a) $\max_{e \in T} c(e)$.

   (b) $\sum_{e \in T} c(e)$.

   (c) $\prod_{e \in T} c(e)$, kde $c(e) \geq 0$ pro všechny hrany e.

3. Dostanete graf, jehož hrany jsou ohodnoceny pouze čísly $1, 2, \ldots, k$. Jak rychle dovedete najít minimální kostru v tomto grafu?

   Dokážete to v čase $\mathcal{O}(kn + km)$? A v čase $\mathcal{O}(kn + m)$? No jestli dokážete i to, tak co v čase $\mathcal{O}((n + m)\alpha(n))$?

4. Dostanete souvislý neorientovaný graf G. Navrhněte algoritmus, který zjistí, jestli z G můžete smazat jednu hranu tak, aby graf zůstal souvislý. Nalezenou hranu vypište. Dovedete snížit časovou složitost vašeho algoritmu až na $\mathcal{O}(n)$?

5. Dostanete souvislý neorientovaný graf G s ohodnocením hran $c : E \to \mathbb{R}$. Navrhněte algoritmus, který najde nejlevnější množinu hran, které můžeme z grafu vymazat tak, aby graf zůstal souvislý, ale už neobsahoval žádnou kružnici.

6. Jedna velká železniční společnost propojuje n měst po celé Evropě. Některá města jsou propojeny přímým spojením. Fixní náklady na provoz přímého spojení mezi městy u a v jsou $c(uv)$ (bez ohledu na vytíženost spojení, jezdit se musí podle jízdního řádu). S několika přestupy existuje dopravní spojení mezi libovolnou dvojicí měst. Spočtěte, kolik nejvíc by železniční společnost mohla ušetřit, kdyby zrušina nadbytečná přímá spojení. (Stále musí být možné se dostat z libovolného města do libovolného jiného).

7. Firma Truhlík a syn má ve městě N budov a chce všechny svoje budovy propojit počítačovou sítí. Vedení firmy rozhodlo, ze pro K ($1 \leq K \leq N$) budov zakoupí vysokorychlostní připojení na Internet. Kromě toho mezi některými dvojicemi budov vybudují propojení optickým kabelem.

   Dvě budovy se nacházejí v téže komponentě sítě, pokud lze mezi nimi komunikovat pomocí optických kabelů (buď mají přímé spojení, nebo jsou spojeny nepřímo přes několik jiných budov). Aby bylo možné komunikovat mezi dvěma budovami ležícími v různých komponentách sítě, musí každá z těchto komponent obsahovat aspoň jeden počítač připojený na Internet.

   Soutěžní úloha: Na vstupu jsou dána čísla N a K a pro každou dvojici budov jedno kladné celé číslo $c(AB)$ = cena za propojení budov A a B optickým kabelem . Navrhněte efektivní algoritmus, jenž určí, kterých K budov se má připojit na Internet a které dvojice budov se mají propojit optickým kabelem tak, aby mezi každými dvěma budovami bylo možné komunikovat a přitom aby celková cena za položení optických kabelů byla co nejmenší.

8. Když chtěl Blátošlap konečně vyprat své ponožky tak, zjistil, že se mu na nich vyvinula celá kolonie neznámých živočichů. Protože to byl genetik, jal se hned tyto živočichy zkoumat a zjistil, ze i když se jedná o jediný druh, jeho zástupci jsou velmi rozmanití. DNA každého jedince má přesně 30 znaků, které ho charakterizují a které se mohou měnit pouze mutací. Dalším výzkumem zjistil, že v prádelníku má N různých poddruhů (poddruhy se navzájem liší alespoň v jednom znaku), i když jeden z nich převazuje.

   Ihned se dovtípil, že v jeho prádelníku došlo k evoluci. A protože ho zajímá její průběh, byli jste požádáni o vyřešení tohoto problému. Blátošlap Vám zadá N a dále DNA jednotlivých poddruhů. DNA každého druhu je zapsáno binárně jako číslo X, $0 \leq X \leq 2^{30} - 1$. i-tý bit tohoto čísla vyjadřuje, zda je i-tý z 30 znaků přítomen.

   Vaším úkolem je zjistit, jaký poddruh se vyvinul z jakého, a počet mutací, ke kterým muselo při tomto vývoji dojít. Předpokládejte, ze vývoj probíhal tak, že každý poddruh kromě toho, který Vám Blátošlap zadal jako první (to je ten nejpočetnější), se vyvinul právě z jednoho jiného poddruhu. Navíc

první zadaný poddruh je původním prapředkem všech poddruhů v prádelníku. Dále předpokládejte, že evoluce probíhala nejjednodušší možnou cestou, a tedy počet mutací v evoluci je nejmenší možný. Počet mutací je součet všech rozdílných znaků mezi každým poddruhem (kromě prvního zadaného) a jeho předkem. Navíc žádný poddruh v Blátošlapově prádelníku nevymřel, všechny evolucí vzniklé poddruhy přežily až do dnešní doby.

Příklad: Pro $N = 4$ a poddruhy 0, 3, 7, 12 je hledaná evoluce tato: poddruhy 3 a 12 se vyvinuly z poddruhu 0, poddruh 7 se vyvinul z poddruhu 3. Počet mutací, ke kterým muselo v evoluci dojít, je roven 5.

9. (Jednoznačnost minimální kostry) Dostanete souvislý ohodnocený graf G. Navrhněte co nejefektivnější algoritmus, který zjistí, jestli je minimální kostra grafu G určena jednoznačně.

10. (Počet koster) Jak je těžké spočítat, kolik má graf G koster? A jak je těžké spočítat, kolik má ohodnocený graf G minimálních koster? Navrhněte oba algoritmy a uveďte jejich časovou složitost.

11. (Bottleneck distance) Dostanete souvislý ohodnocený graf. *Úzké hrdlo* na cestě P je hrana $e \in P$, která má nejvyšší cenu. *Bottleneck distance* mezi vrcholy u a v je minimum z cen úzkého hrdla na všech cestách mezi u a v.

    (a) Navrhněte algoritmus, který pro každou dvojici vrcholů nalezne jejich bottleneck distance. Analyzujte časovou složitost vašeho řešení.

    (b) Co by se změnilo, kdyby úzkým hrdlem byla nejlevnější hrana na cestě? Navrhněte algoritmus, který pro každou dvojici vrcholů nalezne jejich bottleneck distance.

12. (Druhá nejmenší kostra). Když v ohodnoceném grafu G nalezneme minimální kostru T, tak z grafu vyhodíme všechny hrany kostry T a dostaneme graf G'. Minimální kostra v grafu G' je druhou nejmenší kostrou grafu G.

    (a) Navrhněte co nejrychlejší algoritmus, který najde druhou nejmenší kostru. Analyzujte jeho časovou složitost.

    (b) Co kdybychom chtěli nalézt k nejmenších koster? Jaká by byla časová složitost takového algoritmu?

13. (Hybridní algoritmus) Dostaneme graf G a chceme v něm nalézt minimální kostru. Podívejte se na algoritmus, který v první fázi provede k iterací Borůvkova algoritmu. Ve druhé fázi zkontrahuje nalezené komponenty do vrcholů (viz pomocný graf u Borůvkova algoritmu) a na tento graf pustí Jarníkův algoritmus (s Fibonacciho haldou).

    (a) Vyjádřete časovou složitost hybridního algoritmu pomocí n, m a k.

    (b) Pro jakou volbu parametru k bude časová složitost algoritmu nejmenší? Kolik bude časová složitost hybridního algoritmu pro vhodnou volbu k?

1.10.4 Aproximační algoritmy využívající minimální kostry

1. (Steinerovy stromy – dolní odhad pro aproximaci pomocí minimální kostry) Uvažujme pouze ohodnocené grafy takové, že ceny jejich hran splňují trojúhelníkovou nerovnost. Najděte graf $G = (V, E)$ a rozdělení jeho vrcholů V na požadované R a Steinerovy S tak, aby cena jeho minimální kostry byla $(2 - 1/n)$ krát větší než cena jeho optimálního Steinerova stromu ($n = |V|$, kde $V = R \cup S$).

2. (Problém obchodního cestujícího v grafech s trojúhelníkovou nerovností).[11] Obchodní cestující prodává svůj produkt v n městech. Chce si naplánovat takovou trasu, aby objel všechna města a najel co nejméně kilometrů.

   Dostaneme úplný graf, jehož hrany jsou ohodnoceny kladnými reálnými čísly. Chceme najít Hamiltonovskou kružnici[12] nejmenší ceny.

   Obecný problém obchodního cestujícího je NP-úplný, ale ve grafech s trojúhelníkovou nerovností lze aproximovat v polynomiálním čase s libovolnou pevnou přesností. V následujících příkladech budeme předpokládat, že ohodnocení hran grafu splňuje trojúhleníkovou nerovnost.

   (a) Navrhněte algoritmus, který nalezne řešení problému ochodního cestujícího, které je nevýše 2krát horší než optimální řešení.

   (b) Nalezněte příklad grafu na n vrcholech, ve kterém váš aproximační algoritmus nalezne 2krát horší řešení, než je to optimální.[13]

[11] Anglicky se označuje jako Traveling salesman problem (TSP).

[12] Hamiltonovská kružnice je kružnice, která obsahuje všechny vrcholy grafu.

[13] Tedy pokud jste postupovali podobně, jako algoritmus pro 2-aproximaci Steinerova stromu. Jinak ukažte horní i dolní odhad na velikost aproximačního faktoru vašeho algoritmu.

Literatura

[1] B. Chazelle. A minimum spanning tree algorithm with inverse-ackermann type complexity. *J. ACM*, 47:1028–1047, November 2000.

[2] K. W. Chong, Y. Han, and T. W. Lam. Concurrent threads and optimal parallel minimum spanning trees algorithm. *J. ACM*, 48:297–323, March 2001.

[3] M. Mareš. *Graph Algorithms (The Saga of Minimum Spanning Trees)*. PhD thesis, Charles University, Prague, Czech Republic, 2008. `http://mj.ucw.cz/papers/saga/`.

[4] S. Pettie and V. Ramachandran. An optimal minimum spanning tree algorithm. *J. ACM*, 49:16–34, January 2002.