Kapitola 5

Minimální kostry a stromy

V algoritmech prohledávání grafu je výsledné pořadí objevení uzlů do značné míry náhodné – záleží pouze na pořadí probírání uzlů a na pořadí jejich uvedení v seznamech sousedů. Úlohy, kterými se budeme zabývat v této kapitole, vyžadují pro své řešení algoritmy, které systematicky projdou třeba jen určitou část grafu, ale v každém kroku volí mezi několika možnostmi výběru uzlu nebo hrany tu nejlepší. Potřeba takového výběru naznačuje, že časová složitost těchto algoritmů bude obecně nelineární, tedy vyšší než $O(|H| + |U|)$.

5.1 Generování všech koster grafu

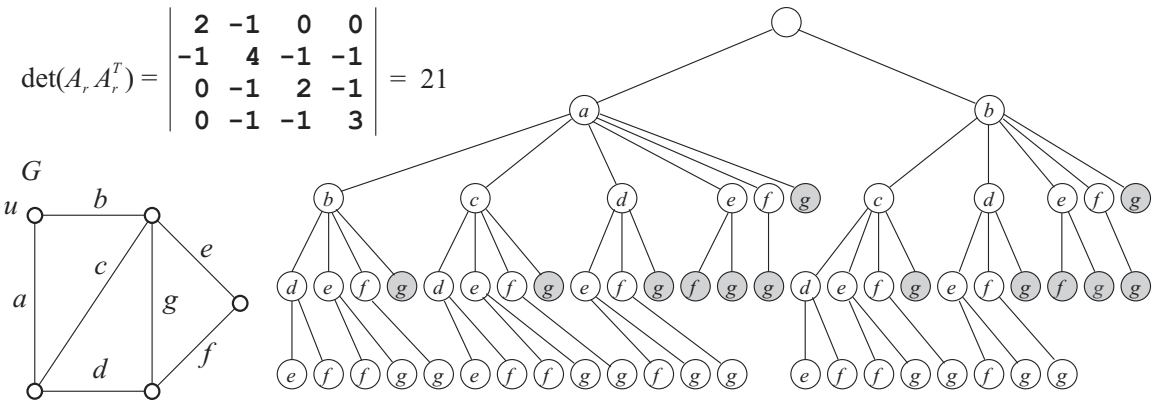
Kromě velmi jednoduchých grafů bývá počet koster grafu velmi vysoký. Tento počet je sice možné určit poměrně jednoduchým postupem vycházejícím z jeho matice incidence nebo matice sousednosti (viz výraz (4.16)), určení struktury všech koster je však výpočetně mnohem náročnější. Při generování všech koster grafu $G = \langle H, U \rangle$ bychom mohli vyjít ze všech podgrafů obsahujících právě $|U| - 1$ hran: pokud podgraf neobsahuje kružnici, tvoří podle části (e) věty 3.26 kostru grafu G. Takový postup je ovšem extrémně neefektivní. Pro $|H| = m, |U| = n$ máme $\binom{m}{n} = O(m^n)$ takových podmnožin hran, na hledání kružnice bychom v každé z nich potřebovali dobu $O(n)$, takže celková složitost činí $O(n.m^n)$. To je jistě příliš i ve srovnání s počtem koster úplného grafu o n uzlech, kterých je „pouze“ n^{n-2}.

Výhodnější jistě bude testovat vznik kružnic již v průběhu systematického generování podgrafů o $n-1$ hranách postupným přidáváním a ubíráním hran. Před každým přidáním hrany se testuje, zda by vznikla kružnice, a v takovém případě se hrana nepřidá a zkouší se hrana další. Dosáhne-li počet hran $n - 1$, získali jsme kostru a další postup je stejný jako u neúspěšného přidání. Tento způsob generování všech koster vyjádříme nyní ve formě pseudokódu takto:

Algoritmus 5.1 Generování všech koster grafu

```
GSP(G)
1   INIT_STACK(T)                          Nastav prázdný zásobník
2   k := |U| − 1                           a urči počet hran kostry.
3   GenTree(H, T, k)                       Vlastní generování koster

GenTree(H, T, k)
1   if |T| = k then DISPLAY_STACK(T)       Zobraz kostru v zásobníku.
2   else if |H| + |T| ≥ k                  Jinak je-li šance na kostru,
3       then h := libovolná hrana z H      vezmi nějakou hranu,
4            if h nevytvoří v T kružnici   a je-li vhodná,
5              then PUSH(T, h)             přidej ji do zásobníku,
6                   GenTree(H − {h}, T, k) pokračuj v generování
7                   POP(T)                 a pak ji opět vyjmi.
8            GenTree(H − {h}, T, k)        Zkus to i bez hrany h.
```



Obrázek 5.1: Generování koster grafu

Algoritmus používá pro ukládání vybraných hran zásobník T. Probírá hrany v libovolném pořadí, vhodnou hranu uloží do zásobníku a zkouší pokračovat v generování všech možných zbytků kostry s použitím zbývajících hran. Bez ohledu na vhodnost hrany pak ještě generuje všechny kostry neobsahující aktuální hranu. Není obtížné dokázat, že uvedený algoritmus opravdu generuje všechny kostry grafu G. Provedeme to prostřednictvím následujícího tvrzení.

Lemma 5.2: Nechť H a T jsou disjunktní podmnožiny množiny hran nějakého grafu, T neobsahuje žádné kružnice a $k \geq |T|$ je přirozené číslo. Potom se provedením procedury GenTree(H, T, k) zobrazí všechny podmnožiny hran z množiny $H \cup T$, které mají mohutnost k, neobsahují kružnice a obsahují všechny hrany z množiny T.

Důkaz: (indukcí podle mohutnosti $|H|$) Můžeme předpokládat, že platí $|H| + |T| \geq k$, jinak je tvrzení triviálně splněno – procedura GenTree nezobrazí žádnou podmnožinu hran.

1. Pro prázdnou množinu H musí být vzhledem k předpokladu lemmatu $k \geq |T|$, současně má být $|H| + |T| = |T| \geq k$, takže platí $k = |T|$. Test na řádce 1 je tedy splněn a procedura zobrazí jedinou podmnožinu hran splňující požadavky lemmatu – tedy samotné T.

2. Nechť tvrzení platí pro množiny s $n(\geq 0)$ hranami a mějme množinu H obsahující $n + 1$ hran. Pokud je $|T| = k$, pak požadavky lemmatu splňuje pouze samotná množina T, a ta bude jako jediná zobrazena díky splněné podmínce na řádce 1. Pokud je $|T| < k$, provede se větev procedury začínající na řádce 3. V ní se do T přidá hrana h pouze tehdy, jestliže nevytváří (v T) žádnou kružnici, načež se rekurzivním voláním procedury na řádce 6 zobrazí podle indukčního předpokladu všechny požadované podmnožiny obsahující hranu h. Dalším voláním na řádce 8 se podle indukčního předpokladu zobrazí všechny požadované podmnožiny neobsahující hranu h. Tím se ale vyčerpávají všechny možnosti pro podmnožiny vyhovující požadavkům dokazovaného tvrzení. △

Věta 5.3: Provedením algoritmu GSP(G) se zobrazí všechny kostry grafu G.

Důkaz: V algoritmu GSP se vyvolá procedura GenTree po předchozím vyprázdnění zásobníku T a nastavení hodnoty k na počet hran kostry grafu G. Podle lemmatu 5.2 se tedy voláním GenTree zobrazí všechny podmnožiny o k hranách neobsahující kružnice – to jsou ale všechny kostry grafu G. △

Proces vytváření podmnožin hran všech koster můžeme vyjádřit stromem, jehož kořen odpovídá prázdné podmnožině hran a každá úroveň následníků vyjadřuje všechna možná přidání jedné hrany k podmnožině reprezentované otcovským uzlem. Ukážeme to pro graf G na obr. 5.1, jehož počet koster je určen podle věty 4.12 pomocí determinantu $\det(\mathbf{A}_r \mathbf{A}_r^T)$. Postupné vytváření koster vyjadřuje strom, v němž jsme pro jednoduchost nezachytili některé neperspektivní podmnožiny hran – tak je např. zbytečné provádět od kořene větvení odpovídající

podmnožinám hran $\{c,\ldots\},\{d,\ldots\}$, atd., neboť ty ponechávají uzel u izolovaný, a nemohou tedy tvořit kostru. Podobně nejsou uvedeny ani hrany, jejichž přidáním vznikne kružnice.

Je zřejmé, že výpočetní složitost algoritmu GSP je vinou exponenciálního počtu koster natolik vysoká, že ji nemůže příliš zhoršit ani případné neefektivní testování vzniku kružnic přidáním hrany. Při řešení jednodušších úloh, např. pokud hledáme pouze jedinou kostru s nějakými specifickými vlastnostmi, hraje složitost testování vzniku kružnic klíčovou roli. Této otázce se proto věnujeme podrobněji v následujícím odstavci.

5.2 Reprezentace množinových rozkladů

Přidávání hran do zásobníku T lze také chápat jako spojování disjunktních podstromů v daném grafu – na počátku (při prázdném zásobníku) je každý podstrom tvořen izolovaným uzlem. Přidáním vhodné hrany se dva podstromy spojí do jediného. Vhodná hrana je tedy taková, která má své krajní uzly v různých podstromech. Pokud má hrana oba krajní uzly v témže podstromu, vznikla by jejím přidáním kružnice, a hranu tedy musíme odmítnout. Je vidět, že se jedná o vytváření rozkladu množiny uzlů grafu do disjunktních podmnožin, přičemž potřebné operace jsou sjednocení a test, zda dva prvky patří do téže podmnožiny.

Strukturu **množinového rozkladu** charakterizuje to, že se jedná o implementaci souboru dynamicky se měnících podmnožin, z nichž každá je reprezentována nějakým svým prvkem – **reprezentantem** dané podmnožiny. Přípustné operace pro tuto datovou strukturu jsou následující:

- ◇ **MAKE-SET**(x) vytvoří novou podmnožinu, jejímž jediným prvkem (a tedy i reprezentantem) je prvek x. Předpokládá se přitom, že x není prvkem žádné jiné podmnožiny.
- ◇ **UNION**(x,y) provede sjednocení podmnožin obsahujících prvky x a y. Tyto podmnožiny jsou před provedením sjednocení disjunktní, důsledkem operace je současně i to, že původní podmnožiny ze zobrazovaného rozkladu zmizí a naopak do něj přibude jejich sjednocení. Reprezentantem sjednocení může být libovolný prvek, typicky to ovšem bývá jeden ze dvou reprezentantů výchozích podmnožin.
- ◇ **FIND-SET**(x) vrací jako svou hodnotu reprezentanta podmnožiny obsahující prvek x.

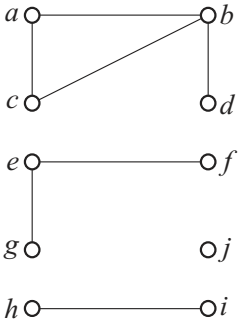
Jako jednoduchý příklad použití datové struktury množinového rozkladu uvedeme algoritmus určení komponent neorientovaného grafu $G=\langle H,U\rangle$. V tomto algoritmu se v libovolném pořadí procházejí hrany grafu, přičemž se sjednocují (dosud) oddělené komponenty odpovídající krajním uzlům jednotlivých hran.

Algoritmus 5.4 Určení komponent neorientovaného grafu

KOMP(G)
1 **for** každý uzel $u\in U$ **do** MAKE-SET(u)
2 **for** každou hranu $[u,v]\in H$ **do**
3 **if** FIND-SET(u) $\neq$ FIND-SET(v) **then** UNION(u,v)

Po provedení procedury KOMP(G) odpovídá výsledný rozklad množiny uzlů komponentám grafu, takže jeho prvky jsou (disjunkní) podmnožiny uzlů tvořící vždy jednu komponentu. Prostým testem FIND-SET(u) = FIND-SET(v) můžeme pak zjišťovat, zda uzly u a v patří do stejné komponenty. Asymptotická časová složitost tohoto algoritmu nemůže být nižší než u algoritmu zjišťujícího komponenty procházením grafu do hloubky, neboť ten pracuje v lineárním čase ve vztahu ke spojové reprezentaci grafu. Cyklus na řádku 1 trvá alespoň $\Theta(|U|)$ a druhý cyklus potřebuje nejméně $\Theta(|H|)$, neboť operace určení reprezentanta a sjednocení podmnožin se mohou v nejlepším případě provést v konstantním čase.

Přesto má tato varianta určování komponent své opodstatnění – je totiž výhodná pro grafy, jejichž množina hran se dynamicky doplňuje. V takovém případě je jistě snazší provést pro



Zpracované hrany	Disjunktní podmnožiny uzlů									
	{a}	{b}	{c}	{d}	{e}	{f}	{g}	{h}	{i}	{j}
(b,d)	{a}	{b,d}	{c}		{e}	{f}	{g}	{h}	{i}	{j}
(e,g)	{a}	{b,d}	{c}		{e,g}	{f}		{h}	{i}	{j}
(a,c)	{a,c}	{b,d}			{e,g}	{f}		{h}	{i}	{j}
(h,i)	{a,c}	{b,d}			{e,g}	{f}		{h,i}		{j}
(a,b)	{a,b,c,d}				{e,g}	{f}		{h,i}		{j}
(e,f)	{a,b,c,d}				{e,f,g}			{h,i}		{j}
(b,c)	{a,b,c,d}				{e,f,g}			{h,i}		{j}

Obrázek 5.2: Rozklad grafu na komponenty

každou novou hranu pouze dvě operace FIND-SET a (možná) jednu operaci UNION, nežli spustit znovu celé prohledání do hloubky. Průběh algoritmu KOMP(G) se zachycením obsahu jednotlivých podmnožin uzlů v každém kroku cyklu na řádcích 2 a 3 ukazujeme na obr. 5.2.

Vraťme se nyní k původní otázce efektivní implementace množinového rozkladu a operací s ním. Je jistě možné si představit reprezentaci každé v něm obsažené podmnožiny jako spojového seznamu, v němž každý záznam obsahuje kromě identifikace prvku podmnožiny a odkazu na další záznam také odkaz na reprezentanta, kterým je první prvek v každém seznamu. Operace FIND-SET pak má konstantní časovou složitost a operaci UNION(u, v) provedeme napojením seznamu prvku u na konec seznamu prvku v. To ale znamená provést úpravu odkazů na reprezentanta v celém přemisťovaném seznamu, takže časová složitost operace UNION je lineární v závislosti na počtu prvků tohoto seznamu.

Ke snížení složitosti vede jednoduchá heuristika, označovaná jako **vyvažované sjednocování** (angl. **weighted-union**), při které v operaci UNION připojujeme vždy kratší seznam za delší. To ovšem vyžaduje, aby součástí záznamu představujícího reprezentanta každé podmnožiny byla i délka seznamu (tedy mohutnost podmnožiny). Tento údaj lze snadno udržovat v aktuálním stavu, takže může časové složitosti jen pomoci. Přesné zhodnocení této varianty podává následující tvzení.

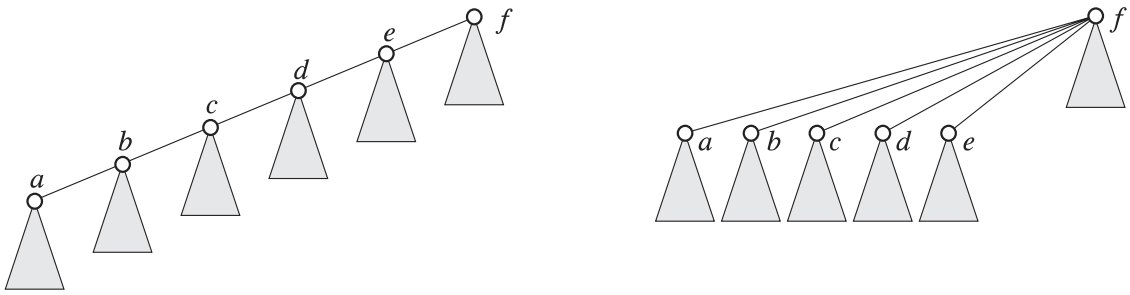
Věta 5.5: Při použití spojové reprezentace množinového rozkladu a heuristiky vyvažovaného sjednocování je úhrná časová složitost posloupnosti m operací MAKE-SET, UNION a FIND-SET, z nichž je právě n operací MAKE-SET, rovna $O(m + n \lg n)$.

Důkaz: Určíme nejprve horní mez počtu úprav odkazu na reprezentanta pro každý prvek n-prvkové množiny. Uvažujme tedy libovolný prvek x, při každé úpravě odkazu na reprezentanta musel tento prvek ležet v menší ze dvou sjednocovaných podmnožin. Prvním sjednocením zahrnujícím prvek x musela tedy vzniknout podmnožina s nejméně dvěma prvky, při druhém sjednocení pak nejméně se čtyřmi prvky, při třetím nejméně s osmi, atd. Po r-té úpravě odkazu vznikla tedy podmnožina alespoň se 2^r prvky, přitom ovšem musí být $2^r \leq n$. U každého z n prvků během provádění všech operací UNION tak mohlo dojít nejvýše ke $\lceil \lg n \rceil$ úpravám odkazů, celkově tedy tyto úpravy trvaly $O(n \lg n)$.

Vedle toho se provádělo také nejvýše m operací MAKE-SET a FIND-SET, které mají konstantní časovou složitost, takže trvaly $O(m)$. Pro celou posloupnost operací tedy dostáváme složitost $O(m + n \lg n)$. △

Při použití datové struktury množinového rozkladu na určení jediné kostry grafu $G = \langle H, U \rangle$ se opearce MAKE-SET provede $|U|$-krát, pak následuje nejvýše $2|H|$ operací FIND-SET a právě $|U| - 1$ operací UNION. Je tedy $n = |U|$ a $m \leq 2(|H| + |U|)$, takže výsledná složitost podle předchozí věty činí

$$O(2|H| + 2|U| + |U| \lg |U|) \;=\; O(|H| + |U| \lg |U|).$$



Obrázek 5.3: Heuristika zkracování cesty

Pro implementaci množinového rozkladu se však nabízí výhodnější reprezentace, která umožňuje docílit ještě lepší časové složitosti. Každou podmnožinu vyjádříme jako kořenový strom, v němž každý prvek obsahuje pouze odkaz na svého otce. Množinový rozklad tak tvoří les kořenových stromů, v nichž je každý kořen reprezentantem celého stromu (a svým vlastním otcem).

Operace MAKE-SET vytvoří kořenový strom s jediným uzlem, operace FIND-SET spočívá v opakování přechodu na otce uzlu, až se dostaneme ke kořenu. Operaci UNION provedeme prostou úpravou odkazu na otce v kořenu jednoho ze stromů tak, aby odkazoval na kořen druhého stromu. V této podobě se ovšem zdá, že co jsme proti předchozí spojové reprezentaci získali zjednodušením operace UNION, ztrácíme složitějším určováním reprezentantů. Této nevýhody se ale zbavíme použitím dvou významných heuristik: **sjednocováním podle hodnosti** v operaci UNION a **zkracováním cesty** v operaci FIND-SET. S jejich pomocí se získá asymptoticky nejrychlejší známá metoda implementace množinového rozkladu.

Heuristika sjednocování podle hodnosti se podobá heuristice používané u spojové reprezentace v tom, že ke kořenu stromu s větším počtem uzlů připojujeme jako následníka kořen stromu s menším počtem uzlů. Namísto počtu uzlů budeme pracovat s tzv. **hodností** stromu, která je aproximací logaritmu počtu uzlů a horní mezí hloubky stromu. V operaci UNION se tedy kořenu s menší hodností upraví odkaz na otce tak, aby směřoval ke kořenu s větší hodností.

Zkracování cesty je rovněž velmi jednoduchá, ale velmi účinná heuristika. Po postupu vzhůru od uzlu ke kořeni stromu v operaci FIND-SET se uzlům na této cestě opraví odkazy na otce tak, aby všechny směřovaly ke kořenu celého stromu. Hodnosti uzlů se však přitom nemění. Na obr. 5.3 ukazujeme schematicky situaci před provedením operace FIND-SET(a) ve stromu s kořenem f a po provedení této operace. Vidíme, že všechny dílčí podstromy se díky této heuristice přiblížily ke kořenu.

V následujícím zápisu algoritmů operací s množinovým rozkladem, které vycházejí ze stromové reprezentace a uvedených dvou heuristik, je součástí záznamu o každém uzlu u odkaz na jeho otce ve stromu $p[u]$ a dále hodnost $hod[u]$, která je aproximací výšky uzlu u ve stromu (t.j. vzdálenosti k nejvzdálenějšímu listu stromu). Hodnost počátečních jednouzlových stromů se pokládá rovna nule, operací FIND-SET se hodnost žádného uzlu nemění. V operaci UNION se strom s nižší hodností stává podstromem kořene s vyšší hodností, kterou opět není třeba měnit. Pouze v případě rovnosti hodností kořenů sjednocovaných stromů se zvýší hodnost libovolného ze dvou kořenů vybraného jako výsledný kořen o 1.

Algoritmus 5.6 Operace s množinovým rozkladem

MAKE-SET(x)
1 $p[x] := x$
2 $hod[x] := 0$

UNION(x, y)
1 LINK(FIND-SET(x), FIND-SET(y))

```
LINK(x, y)
1   if hod[x] > hod[y]  then p[y] := x                    Kořenem se stává x,
2      else p[x] := y                                     nebo se kořenem stává y.
3           if hod[x] = hod[y] then hod[y] := hod[y] + 1  Při shodě hodností se přidá.

FIND-SET(x)
1   if x ≠ p[x] then p[x] :=FIND-SET(p[x])
2   return p[x]
```

Algoritmus operace FIND-SET má v sobě uschovány dva průchody uzly na cestě od x ke kořenu. Pokud uzel x není kořenem, rekurzivní volání postupuje k jeho otci tak dlouho, až se dosáhne kořene. Pro kořen se již rekurze ukončí a jako výsledná se vrátí hodnota odkaz na jeho otce, kterým je samotný kořen. Stejná hodnota se potom po návratu z rekurzivních volání dosazuje do odkazů na otce všech uzlů na cestě. Výhodné vlastnosti této varianty implementace množinového rozkladu shrnuje následující tvrzení.

Věta 5.7: Při použití reprezentace množinového rozkladu pomocí kořenových stromů a heuristik sjednocování podle hodnosti a zkracování cesty je úhrná časová složitost posloupnosti m operací MAKE-SET, UNION a FIND-SET, z nichž je právě n operací MAKE-SET, rovna $O(m \lg^* n)$.

Důkaz: (viz např. [8])

Cvičení

5.2-1. Simulujte provedení algoritmu KOMP(G) pro graf $G = \langle H, U\rangle$, kde $U = \{a, b, c, d, e, f, g, h, i, j, k\}$ a hrany grafu se procházejí v pořadí $(d, i), (f, k), (g, i), (b, g), (a, h), (i, j), (d, k), (b, j), (d, f), (g, j), (a, e), (i, d)$. Průběh algoritmu vyjádřete podobným způsobem, jaký je použitý na obr. 5.2.

5.2-2. Mějme zadaný neorientovaný graf $G = \langle H, U\rangle$ obsahující právě k komponent. Určete pomocí $|H|, |U|$ a k, kolikrát se při provedení algoritmu KOMP s grafem G vyvolají operace FIND-SET a UNION.

5.2-3. Navrhněte procedury realizující operace MAKE-SET, FIND-SET a UNION při spojové reprezentaci množinového rozkladu s použitím heuristiky vyvažovaného sjednocení. Do reprezentace každého prvku x podmnožiny zahrňte složku $rep[x]$ obsahující odkaz na reprezentanta, $last[x]$ odkazující na poslední prvek podmnožiny a $size[x]$ určující mohutnost podmnožiny. Složky $last[x]$ a $size[x]$ je možné udržovat aktualizované pouze u reprezentantů.

5.2-4. Simulujte výpočet následujícího úseku programu při spojové reprezentaci množinového rozkladu s použitím heuristiky vyvažovaného sjednocení. Znázorněte stav seznamů a hodnoty výsledků po provedení závěrečných operací FIND-SET.

```
1   for i := 1 to 16 do MAKE-SET(x_i)
2   for i := 1 to 15 step 2 do UNION(x_i, x_{i+1})
3   for i := 1 to 13 step 4 do UNION(x_i, x_{i+2})
4   UNION(x_1, x_5);  UNION(x_11, x_13);  UNION(x_1, x_10)
5   FIND-SET(x_2);  FIND-SET(x_9)
```

5.2-5. Proveďte předchozí cvičení s tím, že podmnožiny v rozkladu reprezentujete kořenovými stromy a operace předpokládáte implementované s použitím heuristik sjednocování podle hodnosti a zkracování cesty.

5.2-6. Určování hloubky uzlu ve stromu
Je třeba udržovat strukturu lesa $\mathcal{L} = \{T_1, T_2, \ldots, T_k\}$ tvořeného kořenovými stromy, v němž se

používají tyto tři operace:

- ◇ **MAKE-TREE**(v) vytvoří kořenový strom s jediným uzlem v.
- ◇ **FIND-DEPTH**(v) určí hloubku uzlu v (tzn. vzdálenost uzlu v od kořene stromu).
- ◇ **GRAFT**(r, v) připojí uzel r, který je kořenem stromu, jako následníka uzlu v, který je obsažen v jiném stromu (uzel v může a nemusí být kořenem).

(a) Předpokládejte, že reprezentace kořenových stromů je podobná jako při implementaci množinového rozkladu: $p[x]$ odkazuje na předchůdce uzlu x, pro kořen je $p[x] = x$. Operaci GRAFT(r, v) můžeme vyjádřit pomocí $p[r] := v$, operaci FIND-DEPTH(v) provedeme průchodem cesty od v ke kořeni s počítáním její délky. Ukažte, že v nejhorším případě je časová složitost posloupnosti m operací MAKE-TREE, FIND-DEPTH a GRAFT rovna $\Theta(m^2)$.

Použitím heuristik sjednocení podle hodnosti a zkracování cesty je možné časovou složitost zlepšit. K reprezentaci lesa $\mathcal{L}$ použijeme soustavu $\mathcal{S} = \{S_i\}$ tvořící množinový rozklad implementovaný opět jako kořenové stromy, přičemž každému stromu $T_i \in T$ odpovídá právě jedna množina $S_i \in S$. Struktury T_i a S_i však nejsou nutně jako grafy isomorfní – tedy S_i nevyjadřuje stejnou relaci předcházení jako T_i, ale přesto dovolí určit hloubku každého uzlu ve stromu T_i.

Základní myšlenka spočívá v tom, že každý uzel v stromu S_i nese hodnotu pseudovzdálenosti $d[v]$, která je definována tak, že součet pseudovzdáleností podél cesty z v do kořene ve stromu S_i je roven vzdálenosti uzlu v od kořene ve stromu T_i. To znamená, že je-li $\langle v, v_1, v_2, \ldots, v_k\rangle$ cesta z uzlu v ke kořeni v_k, pak je hloubka uzlu v ve stromu T_i rovna $\sum_{j=0}^{k} d[v_j]$.

(b) Navrhněte algoritmus operace MAKE-TREE.

(c) Ukažte, jak získat algoritmus operace FIND-DEPTH prostřednictvím vhodné úpravy algoritmu pro FIND-SET. Využívá se zkracování cest a časová složitost algoritmu by měla být lineární vzhledem k délce cesty ke kořeni. Zaměřte se na správné určování pseudovzdálenosti.

(d) Ukažte, jak se upraví algoritmy pro LINK a UNION, abychom získali GRAFT(r, v), pomocí kterého se spojují stromy obsahující r a v. Zajistěte správnou úpravu pseudovzdáleností při této operaci. Berte v úvahu, že kořen stromu vyjadřujícího množinu S_i není nutně kořenem odpovídajícího stromu T_i.

5.2-7. Při systematickém generování všech koster grafu se hrany do vytvářené kostry nejen přidávají, ale také se odebírají, aby se mohla ověřit další kombinace hran. Navrhněte efektivní implementaci operace SPLIT-SET(u, v), jejímž úkolem je rozdělit množinu obsahující uzly u a v do dvou disjunktních podmnožin, které budou obsahovat každá pouze jeden z těchto uzlů. Odvoďte časovou složitost operace SPLIT-SET. Důležitým požadavkem při rozdělení je to, aby vzniklé podmnožiny opět odpovídaly stavu faktorové množiny před provedením jejich někdejšího sjednocení.

5.3 Minimální kostry

Při generování všech koster grafu jsou různé kostry daného grafu v zásadě rovnocenné – všechny obsahují stejný počet hran $h(G)$. Jiná situace ovšem nastane, pokud každé hraně přiřadíme nějakou nezápornou reálnou délku. V takovém případě má smysl zabývat se hledáním takové kostry, která má nejmenší součet délek svých hran. Tuto úlohu lze obecně formulovat následujícím způsobem.

Definice 5.8: Nechť je dán souvislý neorientovaný graf $G = \langle H, U\rangle$ s nezáporným ohodnocením hran $w : H \mapsto \mathbf{R}^+$. **Problém minimální kostry** [1] grafu G spočívá v nalezení takové jeho kostry $T = \langle H_v, U\rangle$, která splňuje následující podmínku:

$$\sum_{h \in H_v} w(h) \quad \text{je minimální.} \tag{5.1}$$

[1] Stručné pojmenování minimální kostra se již vžilo, jedná se ovšem o kostru s minimální váhou – bez ohodnocení hran ztrácí pojem minimální kostra smysl.

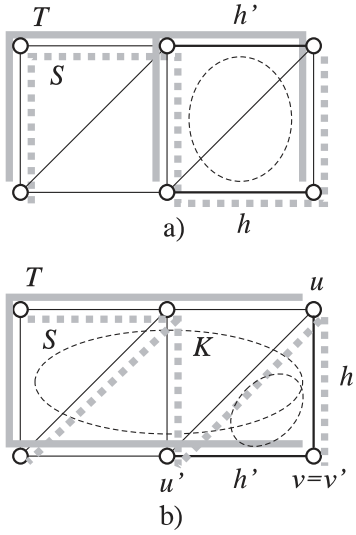
Z formulace problému minimální kostry lze snadno poznat, že v grafu může existovat několik koster splňujících podmínku minimálnosti (5.1), řešení problému není tedy obecně jednoznačné. Při hledání nějaké minimální kostry je účelné mít k dispozici test, pomocí něhož můžeme ověřit, zda nějaká kostra je minimální, aniž bychom museli její ohodnocení srovnávat se všemi možnými kostrami grafu. Pro získání takového testu dokážeme nejprve jedno pomocné tvrzení.

Lemma 5.9: Nechť S, T jsou dvě různé kostry neorientovaného grafu $G = \langle H, U\rangle$. Potom mezi množinami hran grafů $S - T$ a $T - S$ existuje bijekce ω splňující následující podmínku: pro každou hranu $h \in (S-T)$ obsahuje jediná kružnice grafu $T \cup \{h\}$ i hranu $\omega(h)$ a jediná kružnice grafu $S \cup \{\omega(h)\}$ obsahuje i hranu h.

Důkaz: S a T budeme chápat přímo jako množiny hran příslušných koster. Protože se jedná o kostry téhož grafu, mají stejný počet hran $|U - 1|$, takže platí i $|S - T| = |S| - |S \cap T| = |T| - |T \cap S| = |T - S|$. Postupujeme tedy indukcí podle mohutnosti rozdílu $S - T$.

1. Pro $|S - T| = 1$ se kostry liší pouze v jediné hraně, nechť $S - T = \{h\}$, $T - S = \{h'\}$ (viz obr. 5.4a). Graf $T \cup \{h\}$ obsahuje jedinou kružnici, která nemůže celá ležet v S, takže v ní existuje hrana z množiny $T - S$. Tou hranou je ale nutně h', a tak volbou $\omega(h) = h'$ dostaneme přiřazení s požadovanou vlastností.

2. Nechť tvrzení platí pro kostry lišící se v n a méně hranách a mějme kostry S a T takové, že $|S - T| = n + 1$. Zvolme libovolně $h = [u, v] \in (S - T)$ a označme jako K jedinou kružnici grafu $T \cup \{h\}$. Na kružnici K musí existovat alespoň jedna hrana, která není obsažena v S (viz obr. 5.4b). Alespoň jedna z takových hran $h' = [u', v'] \in (K - S)$ má navíc tu vlastnost, že ve stromu S jednoznačně určené cesty u — u' a v — v' jsou disjunktní (libovolná z nich může být přitom prázdná). Spojením cest a hran v pořadí $[u, v]$, v — v', $[v', u']$, u' — u dostaneme kružnici, která odpovídá ve stromu S tětivě h'.



Obrázek 5.4: K důkazu lemmatu

Dvojice hran h a h' splňuje tedy vlastnosti požadované v tvrzení dokazovaného lemmatu, takže můžeme položit $\omega(h) = h'$. Položme nyní $S_1 = S - \{h\} \cup \{h'\}$, takže $|S_1 - T| = n$. Podle indukčního předpokladu existuje požadovaná bijekce ω_1 mezi hranami $S_1 - T$ a $T - S_1$, která se doplněním o dvojici $h \mapsto h'$ stane hledanou bijekcí ω mezi $S - T$ a $T - S$. △

Tvrzení předchozího lemmatu lze chápat také tak, že pro dvojici koster S, T téhož grafu lišící se v n hranách (tzn. $|S - T| = n$) je možné pomocí n záměn hran $h \in (S - T)$ hranami $\omega(h) \in (T - S)$ přejít od kostry S ke kostře T. Nyní už budeme formulovat avizovaný test minimálnosti kostry.

Věta 5.10: Kostra $T = \langle H_v, U\rangle$ grafu $G = \langle H, U\rangle$ s nezáporným ohodnocením hran w je minimální kostrou právě tehdy, když splňuje podmínku

$$w(t) \geq \max_{h \in K} w(h) \tag{5.2}$$

pro libovolnou tětivu $t \in (H - H_v)$, kde K je jediná kružnice faktoru $T \cup \{t\}$.

Důkaz: $\Rightarrow$ Minimální kostra T jistě splňuje podmínku (5.2) – kdyby totiž existovala tětiva t taková, že $w(t) < w(h_1)$ pro nějakou hranu $h_1 \in K$, pak by $(T \cup \{t\}) - \{h_1\}$ byla kostra grafu G s menším ohodnocením než T, což je spor.

$\Leftarrow$ Nechť T splňuje podmínku (5.2) a S je nějaká minimální kostra grafu G. Označme jako k

počet hran kostry S, které nejsou obsaženy v kostře T, takže jsou jejími tětivami.

- Je-li $k = 0$, je T shodná s S, takže je minimální kostrou.
- Je-li $k \geq 1$, vybereme hranu $t \in (S - T)$ a jí odpovídající hranu $t' = \omega(t) \in (T - S)$, kde ω je přiřazení uvedené v lemmatu 5.9. Z minimálnosti kostry S plyne, že $w(t) \leq w(t')$, z podmínky (5.2) pro kostru T a jedinou kružnici grafu $T \cup \{t\}$ odpovídající tětivě t dostáváme $w(t) \geq w(t')$. Je tedy $w(t) = w(t')$ a kostra $S - \{t\} \cup \{t'\}$ má stejné ohodnocení jako S, přitom se liší od T už jen v $(k-1)$ hranách. Opakováním tohoto kroku přejdeme ke kostře T se zachováním ohodnocení původní kostry S, takže i T je minimální. △

Hledání minimální kostry lze koncipovat několika způsoby. Jejich společným rysem je to, že pracují iteračně: postupně konstruují takové podgrafy daného grafu, které se přibližují hledané minimální kostře. Přibližování se může dosahovat buď vypouštěním hran výchozího grafu, nebo přidáváním hran k podgrafu, který je na počátku prázdný, anebo konečně výměnou hran v nějaké výchozí kostře. Jedná se samozřejmě o to, aby tyto iterace byly výpočetně co nejjednodušší a postupovaly co nejrychleji ke hledané kostře.

Kritérium efektivnosti splňují nejlépe algoritmy používající postupné přidávání hran. Minimální kostra se začíná vytvářet počínaje prázdnou množinou hran T (podobně jako při systematickém generování všech koster grafu), ke které se přidávají vhodné hrany tak dlouho, až se získá úplná minimální kostra. Základní invariantou tohoto postupu je podmínka, že množina hran T je podmnožinou hran nějaké minimální kostry – tato podmínka se nesmí přidáním nové hrany porušit. Uvedený postup lze popsat následujícím generickým algoritmem.

Algoritmus 5.11 Hledání minimální kostry grafu

GENERIC-MST(G, w)

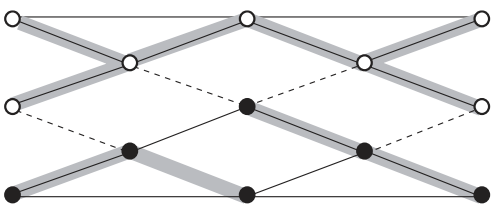
```
1  T := ∅
2  while T netvoří kostru
3      do najdi vhodnou hranu [u, v] pro T
4         T := T ∪ {[u, v]}
5  return T
```

Zbývá nalézt efektivní způsob, jak poznat v daném okamžiku nějakou vhodnou hranu pro přidání do dílčí minimální kostry. Následující tvrzení poskytuje obecné pravidlo, k němuž později doplníme i náležitý způsob jeho využití.

Věta 5.12: Nechť P je podstrom dosud vytvořené části minimální kostry grafu $G = \langle H, U \rangle$. Nalezneme jiný podstrom Q a hranu $[p, q] \in H, p \in P, q \in Q$, pro kterou platí

$$w(p, q) = \min w(u, v), \tag{5.3}$$

kde minimum se bere přes všechny možné podstromy $Q \neq P$ a hrany $[u, v] : u \in P, v \in Q$. Potom lze hranu $[p, q]$ přidat k vytvářené minimální kostře.



Obrázek 5.5: Vhodné hrany pro přidání

Důkaz: Nechť T je minimální kostra obsahující dosud vytvořené části P, Q. Předpokládejme, že hrana $[p, q]$ v kostře T neleží – jinak bychom byli hotovi. Označme U_P množinu uzlů podstromu P a $U_Q = U - U_P$ množinu zbývajících uzlů grafu G. Předpokládejme, že v kostře T leží hrana $h \neq [p, q]$ s jedním krajním uzlem v U_P a druhým v U_Q taková, že $w(h) > w(p, q)$. Potom má ale kostra $(T - \{h\}) \cup \{[p, q]\}$ menší ohodnocení než T, a to je spor s minimálností T.

V kostře T tedy leží hrana $h = [u, v] \neq [p, q], u \in P, v \in Q$ taková, že $w(h) = w(p, q)$. Pak ale můžeme sestrojit minimální kostru $T - \{h\} \cup \{[p, q]\}$, což znamená, že hrana $[p, q]$ je vhodná pro přidání do vytvářené minimální kostry. △

Na obr. 5.5 ukazujeme, které hrany se berou v úvahu v určitém okamžiku provádění algoritmu GENERIC-MST. Hrany tvořící součást dosud vytvořené minimální kostry jsou znázorněny tučně, bíle vyplněné uzly vyznačují podstrom P, uzly s tmavou výplní jsou z množiny U_Q. Hrany propojující P a U_Q jsou znázorněny čárkovaně, a právě z nich se vybere hrana s minimální délkou $w(h)$, která bude vhodná pro přidání do kostry.

Cvičení

5.3-1. Nechť h je hrana s minimálním ohodnocením $w(h)$ v neorientovaném grafu G. Dokažte, že existuje minimální kostra grafu G obsahující hranu h.

5.3-2. Nechť h je hrana s maximálním ohodnocením $w(h)$ v nějaké kružnici neorientovaného grafu $G = \langle H, U\rangle$. Dokažte, že existuje minimální kostra T grafu $G' = \langle H - \{h\}, U\rangle$, která je současně minimální kostrou grafu G.

5.3-3. Dokažte, že pro kladné ohodnocení hran w neorientovaného grafu G tvoří každý souvislý faktor grafu G s minimálním součtem ohodnocení všech svých hran kostru grafu G. Ukažte současně na vhodném příkladu, že v případě existence hran s nulovým nebo záporným ohodnocením tomu tak být nemusí.

5.3-4. Nechť T je minimální kostra grafu G s ohodnocením w a označme jako $\langle w_1, w_2, \ldots, w_{n-1}\rangle$ vzestupně uspořádanou posloupnost ohodnocení jejích hran. Dokažte, že i každé jiné kostře T' grafu G bude odpovídat stejná uspořádaná posloupnost vah jejích hran.

5.3-5. Nechť T je minimální kostra neorientovaného grafu $G = \langle H, U\rangle$ s ohodnocením w. Označme jako $G' = \langle H', U'\rangle$ podgraf indukovaný podmnožinou uzlů $U' \subseteq U$. Dokažte, že je-li podgraf $T' = G' \cap T$ souvislý, potom je minimální kostrou grafu G'.

5.3-6. Zjistěte, zda následující tvrzení jsou pravdivá či nikoliv. Pravdivá dokažte, nepravdivá vyvraťte protipříkladem.

(a) Jsou-li váhy všech hran grafu navzájem různé, pak je jeho minimální kostra určena jednoznačně.

(b) Jsou-li váhy všech hran grafu navzájem různé, pak mají jeho různé kostry rozdílné celkové ohodnocení.

(c) Nechť h je hrana, jejíž váha je menší než váha jakékoliv jiné hrany. Potom je h obsažena v každé minimální kostře grafu.

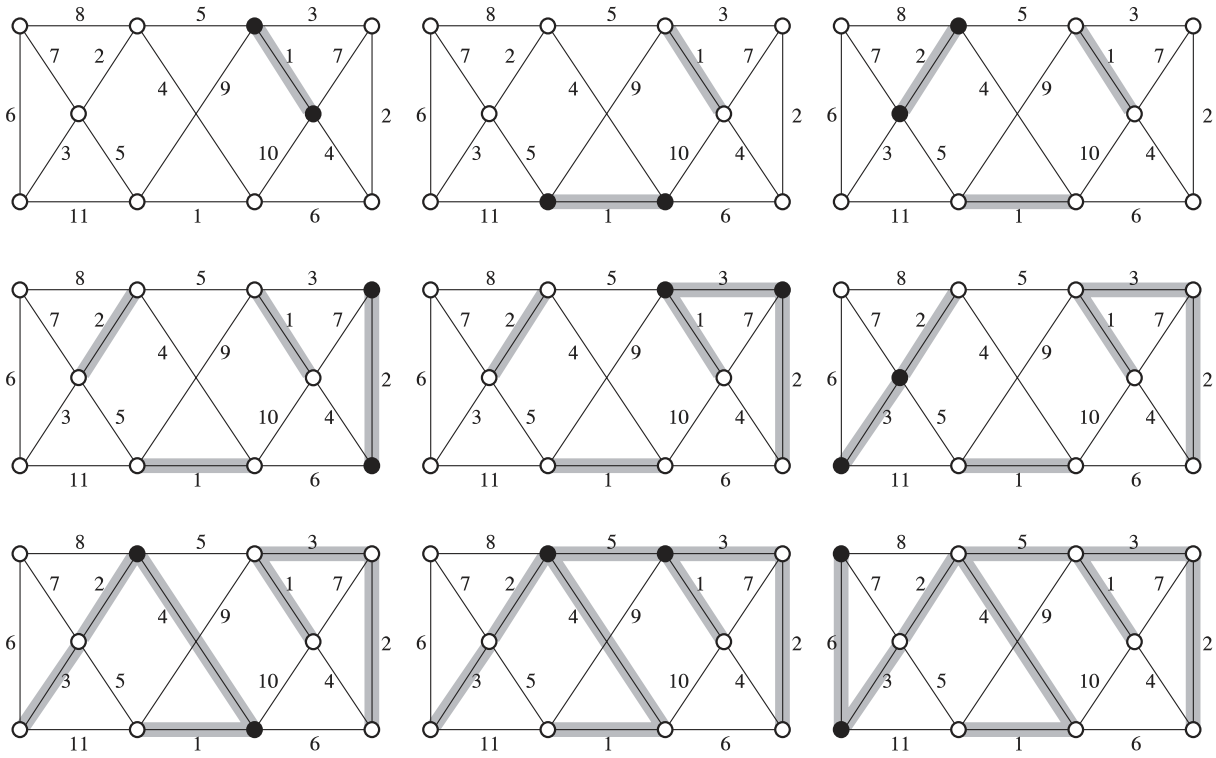
5.4 Algoritmy Borůvky a Jarníka

Problém určení minimální kostry je motivován mnoha praktickými aplikacemi a má také svoji historii, do které se úspěšně zapsali i dva čeští matematici. Nejprve jej formuloval a úspěšně vyřešil v polovině 20. let Otakar Borůvka v souvislosti s návrhem sítě pro rozvod elektřiny na Moravě. Efektivnější metodu řešení navrhl r. 1930 Vojtěch Jarník. Používání prvních počítačů po r. 1950 přineslo oživení zájmu o algoritmické řešení problému minimální kostry, a tak byly algoritmy Otakara Borůvky i Vojtěcha Jarníka znovu nezávisle objeveny a publikovány v USA, takže se teprve potom staly všeobecně známými (přehled lze nalézt např. v [9]).

Oba zmiňované algoritmy jsou variantami generického algoritmu uvedeného v předchozím odstavci, liší se pouze ve způsobu výběru další hrany pro přidání do dílčí vytvořené kostry na řádce 3.

Borůvkův-Kruskalův algoritmus

Základní myšlenkou Borůvkova-Kruskalova algoritmu je, že pro přidání vybírá takovou hranu, která má ze všech možných hran spojujících dva různé podstromy ve vytvářené kostře tu



Obrázek 5.6: Provádění Borůvkova-Kruskalova algoritmu

nejmenší váhu. Podle věty (5.2) je takový postup zcela korektní. Implementace Borůvkova-Kruskalova algoritmu používá pro reprezentaci podstromů vytvářené kostry strukturu množinového rozkladu a až na doplnění týkající se výběru hran se shoduje s algoritmem 5.4 pro určení komponent neorientovaného grafu.

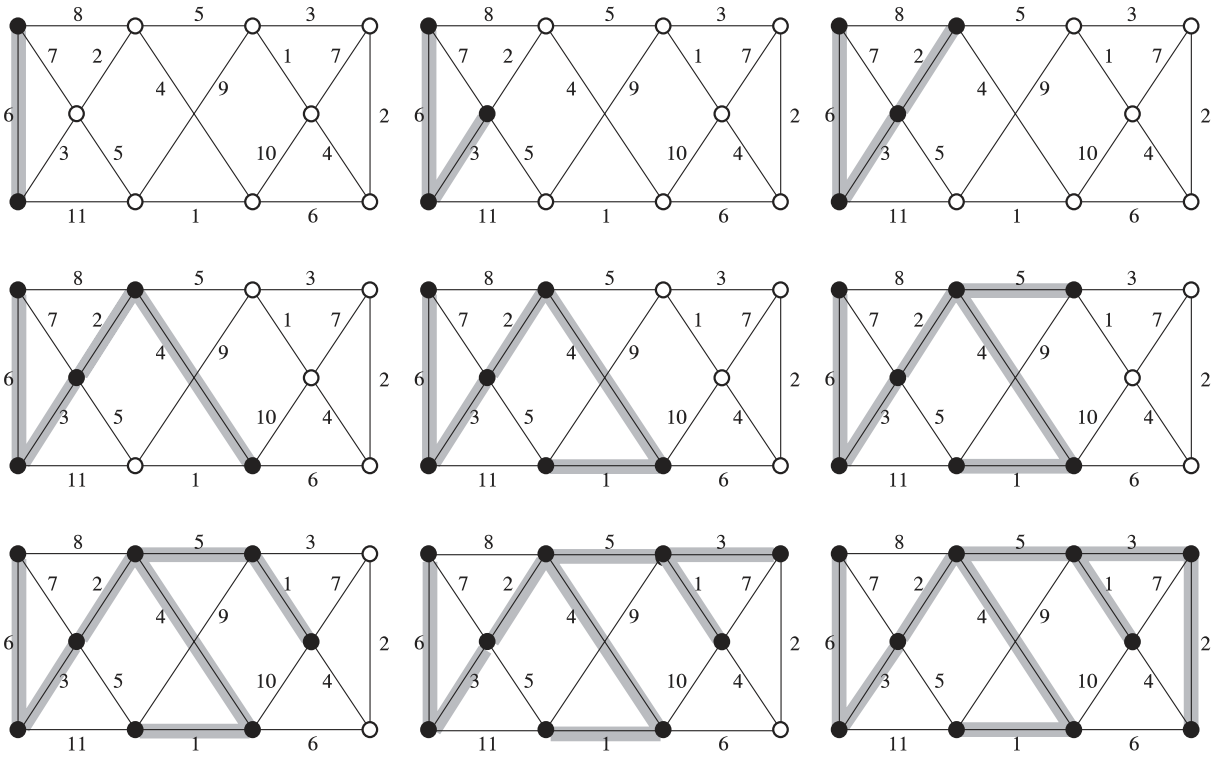
Každý dílčí podstrom vyjadřuje podmnožina jeho uzlů, operace FIND-SET(u) vrací reprezentanta podstromu, do něhož patří uzel u. Srovnáním hodnot FIND-SET(u) a FIND-SET(v) tedy snadno zjistíme, zda se uzly u a v nalézají ve stejném podstromu. Spojení dvou podstromů do jednoho se provádí pomocí operace UNION.

Algoritmus 5.13 Určení minimální kostry grafu (Borůvka – Kruskal)

KB-MST(G, w)

```
1  A := ∅                                          Vybraná kostra zatím prázdná,
2  for každý uzel u ∈ U                            pro každý uzel se vytvoří
3      do MAKE-SET(u)                              samostatný podstrom.
4  uspořádej H do neklesající posloupnosti podle váhy w
5  for každou hranu [u, v] ∈ H v pořadí neklesajících vah
6      do if FIND-SET(u) ≠ FIND-SET(v)             Hrana [u, v] je vhodná,
7            then A := A ∪ {[u, v]}                přidej ji do kostry
8                 UNION(u, v)                      a spoj odpovídající podstromy.
9  return A
```

V uvedené podobě má algoritmus poněkud obecnější funkci – pro nesouvislý graf totiž vytvoří jeho minimální les. Pro souvislý graf je možné ukončit cyklus na řádce 5, jakmile má vytvářená kostra potřebný počet (t.j. $|U| - 1$) hran. Činnost algoritmu ilustrujeme na obr. 5.6 – tučně vytažené hrany vyznačují vytvářenou minimální kostru, dvojice tmavě zobrazených uzlů určuje hranu $[u, v]$ přidávanou v daném kroku do minimální kostry (čísla u hran vyjadřují jejich ohodnocení w). Časová složitost Borůvkova-Kruskalova algoritmu závisí na



Obrázek 5.7: Provádění Jarníkova-Primova algoritmu

způsobu implementace řazení a operací s množinovým rozkladem. Pro graf $G = \langle H, U \rangle$ dosáhneme seřazení jeho hran v čase $O(|H| \lg |H|)$, použitím heuristik sjednocování podle hodnosti a zkracování cesty dosáhneme pro $O(|H|)$ operací s množinovým rozkladem celkového času $O(|H| \lg^* |U|)$ (viz věta 5.7). Počáteční nastavení trvá $O(|U|)$, takže celkově lze asymptotickou složitost Borůvkova-Kruskalova algoritmu stanovit rovnou $O(|H| \lg |H|)$.

Jarníkův-Primův algoritmus

Jarníkova-Primova varianta algoritmu určení minimální kostry vychází z toho, že hrany vybrané do množiny A tvoří stále jediný podstrom, k němuž se připojí přidanou hranou vždy pouze jeden nový uzel. Protože uzel u přidávané hrany $[u, v]$ leží vždy ve stejném dílčím podstromu, je podle věty 5.12 nutné vybírat tuto hranu tak, aby její ohodnocení $w(u, v)$ bylo minimální ze všech takových hran. Na obr. 5.7 ukazujeme průběh Jarníkova-Primova algoritmu. Hrany vytvářené kostry jsou vytaženy tučně, tmavě znázorněné uzly ukazují dosud propojené uzly v dílčím podstromu. První uzel tohoto podstromu je možné zvolit libovolně.

Pro efektivní implementaci Jarníkova-Primova algoritmu je podstatné rychlé provádění výběru hrany pro přidání do množiny A. Je zajištěno tak, že uzly dosud nepatřící do dílčího podstromu jsou uloženy v **prioritní frontě** Q uspořádané podle hodnoty $d[v]$ přiřazené každému takovému uzlu. Tato hodnota určuje minimální vzdálenost uzlu v od dílčího podstromu, tedy nejmenší ohodnocení hrany $[u, v]$, která má druhý krajní uzel v dílčím podstromu. Příslušný uzel u budeme současně označovat jako **předchůdce uzlu** v a identifikujeme jej pomocí složky $p[v]$ každého uzlu.

V průběhu algoritmu se tedy struktura vytvářené kostry vyjadřuje implicitně pomocí odkazů na předchůdce podobně jako u algoritmů prohledávání grafu do šířky a do hloubky. Je-li za kořen dílčího podstromu zvolen uzel r, pak budou množinu hran A tvořit hrany

$$A \;=\; \{[p[v], v] : v \;\in U - \{r\} - Q\}.$$

Výsledná kostra grafu G se získá po vyprázdnění fronty Q, takže její množina hran bude rovna

$$A \;=\; \{[p[v], v] : v \;\in U - \{r\}\}.$$

Algoritmus 5.14 Určení minimální kostry grafu (Jarník – Prim)

JP-MST(G, w, r)

```
1   Q := U                                  Do fronty ulož všechny uzly
2   for každý uzel u ∈ Q                    a urči jim nekonečnou
3       do d[u] := ∞                        vzdálenost od podstromu.
4   d[r] := 0; p[r] := NIL                  Pouze kořen nemá předchůdce.
5   while Q ≠ ∅ do                          Dokud není fronta prázdná,
6       u:=EXTRACT-MIN(Q)                   vyber nejbližší uzel a sousedům z Q
7           for každý uzel v ∈ Adj[u] do    zkus, zda mají blíže k u.
8               if (v ∈ Q) and (w(u,v) < d[v])
9                   then p[v] := u          Pokud ano, zadej jim předchůdce u
10                       d[v] := w(u,v)     a menší vzdálenost.
```

Časová složitost tohoto algoritmu závisí na tom, jakým způsobem implementujeme prioritní frontu Q. Pokud ji reprezentujeme jako **binární haldu**, pak bude inicializace na řádcích 1 – 3 trvat $O(|U|)$. Hlavní cyklus se provádí $|U|$-krát a každé provedení EXTRACT-MIN zabere $O(\lg |U|)$, takže na vybírání z fronty bude zapotřebí $O(|U| \lg |U|)$. Vnitřní cyklus na řádcích 7 – 10 se provádí celkem $O(|H|)$-krát, protože celková délka seznamů sousedů je $2|H|$. Abychom zrychlili test náležení do fronty Q, můžeme ji doplnit bitovou mapou uzlů, čímž dosáhneme konstantní složitosti testu, zda $v \in Q$. Přiřazení nové hodnoty $d[v]$ znamená upravit i pořadí daného uzlu v prioritní frontě, což lze provést v čase $O(\lg |U|)$. Celková složitost Jarníkova-Primova algoritmu je tedy

$$O(|U|) + O(|U| \lg |U|) + O(|H| \lg |U|) \;=\; O(|H| \lg |U|)$$

Asymptotickou časovou složitost Jarníkova-Primova algoritmu je možné dále zlepšit použitím Fibonacciho haldy, kdy se operace úpravy vzdálenosti uzlu ve frontě provádí (v průměru) v konstantním čase. Složitost určení minimální kostry se pak zlepší na $O(|H| + |U| \lg |U|)$.

Problém určení minimální kostry může být ovšem ještě dále zobecněn. Velmi zajímavá (ale současně mnohem obtížněji řešitelná) je tzv. **Steinerova úloha**, kdy se pro daný graf $G = \langle H, U \rangle$ s nezáporným ohodnocením hran má nalézt minimální strom obsahující jistou podmnožinu uzlů $V \subseteq U$. Ostatní uzly z množiny $U - V$ v tomto stromu nemusí být zahrnuty. Steinerova úloha tedy vlastně představuje hledání minimální kostry všech podgrafů grafu G indukovaných podmnožinami uzlů $V' : V \subseteq V' \subseteq U$.

Původní zadání Steinerovy úlohy bylo geometrické: V rovině je zadána množina bodů V, které se mají propojit čarami tak, aby celková délka čar byla minimální. Pokud se čáry nemohou spojovat jinde, než v zadaných bodech, jedná se o hledání minimální kostry úplného grafu o $|V|$ uzlech, v němž jsou délky hran rovny euklidovské vzdálenosti odpovídajících bodů. Pokud je ale přípustné spojování i mimo vyznačené body, lze dosáhnout dalšího zkrácení celkové délky, ale úloha se značně komplikuje.

Pro speciální případ, kdy zadané body tvoří vrcholy pravidelného n-úhelníka, vyřešili tento problém čeští matematici Jarník a Kössler [13]. V obecném případě však dosud uspokojivé řešení neexistuje, i když je známa celá řada vlastností, které musí výsledné spojnice mít (podrobněji viz [32].

Praktickou variantu této úlohy představuje propojení zadané množiny pájecích bodů při návrhu plošných desek tištěných spojů. Zde se ovšem neuvažuje euklidovská, ale tzv. **Manhattanská metrika**, která bodům $\langle x_1, y_1 \rangle$, $\langle x_2, y_2 \rangle$ určuje vzdálenost $d = |x_1 - x_2| + |y_1 - y_2|$. V tomto případě se sice Steinerova úloha redukuje na klasické hledání minimální kostry, ale

problematika návrhu tištěných spojů zahrnuje další doplňující požadavky, takže ji nelze chápat jen jako Steinerovu úlohu.

Problém určení minimální kostry lze přenést i na orientované grafy a doplnit jej např. požadavkem, že se musí jednat o minimální kořenovou kostru (kořen může nebo nemusí být zadán). Řešení této úlohy již nelze získat pouhým použitím orientovaného znění uvedených algoritmů – o tom se může čtenář snadno přesvědčit. Základní algoritmy hledání minimální kořenové kostry orientovaného grafu jsou popsány např. v [32].

Cvičení

5.4-1. Nechť jsou v grafu $G = \langle H, U \rangle$ zadány dvě podmnožiny hran $A, B \subseteq H$, $A \cap B = \emptyset$, pro které je třeba nalézt faktor $T = \langle H', U \rangle$ grafu G s minimální sumou ohodnocení hran takový, že platí

$$H' \subseteq (H - A) \quad \& \quad (B \subseteq H').$$

(a) Navrhněte vhodnou úpravu algoritmů určení minimální kostry, pomocí které se získá algoritmus řešení této rozšířené úlohy a určete jeho asymptotickou časovou složitost.
(b) Určete, za jakých podmínek bude nalezený faktor T kostrou grafu G.

5.4-2. Borůvkův-Kruskalův algoritmus může vytvořit různé minimální kostry téhož grafu G v závislosti na tom, v jakém pořadí jsou v první části algoritmu seřazeny hrany se stejným ohodnocením. Ukažte, že pro libovolnou minimální kostru T existuje takové uspořádání hran grafu G podle ohodnocení w, pro které Borůvkův-Kruskalův algoritmus vytvoří kostru T.

5.4-3. Pro graf $G = \langle H, U \rangle$ reprezentovaný maticí sousednosti navrhněte takovou realizaci Jarníkova-Primova algoritmu, která bude mít asymptotickou časovou složitost $O(|U|^2)$.

5.4-4. Má implementace Jarníkova-Primova algoritmu pomocí Fibonacciho haldy pro řídký graf $G = \langle H, U \rangle$, $|H| = \Theta(|U|)$ lepší asymptotickou časovou složitost nežli implementace používající binární haldu? Jak je to v případě hustého grafu, kde $|H| = \Theta(|U|^2)$? Jaký musí být vztah mezi mohutnostmi $|U|$ a $|H|$, aby byla implementace pomocí Fibonacciho haldy asymptoticky rychlejší než implementace pomocí binární haldy?

5.4-5. Předpokládejte, že ohodnocení hran grafu $G = \langle H, U \rangle$ jsou celá čísla z intervalu $\langle 1, |U| \rangle$. Jak je možné zrychlit v tomto případě Borůvkův-Kruskalův algoritmus? Je možné nějak využít informace, že ohodnocení hran jsou celá čísla z intervalu $\langle 1, W \rangle$ pro nějakou konstantu W?

5.4-6. Navrhněte efektivní algoritmus, který pro zadaný neorientovaný graf G určí takovou jeho kostru, jejíž maximální ohodnocení hrany je nejmenší ze všech koster grafu G.

5.4-7. Ukažte na příkladu, že algoritmy pro hledání minimální kostry nelze v této podobě použít pro hledání minimální kořenové kostry.

5.4-8. Předpokládejme, že ohodnocení hran grafu jsou rovnoměrně rozdělena v polouzavřeném intervalu $\langle 0, 1)$. Pro který ze dvou uvedených algoritmů určení minimální kostry lze tuto informaci využít k jeho zrychlení?

5.4-9. Předpokládejme, že pro daný neorientovaný graf G již byla určena minimální kostra. Jak složité je tuto kostru upravit, pokud ke grafu G přidáme nový uzel a s ním incidující hrany?

5.4-10. Druhá nejlepší minimální kostra grafu
Nechť $G = \langle H, U \rangle$ je souvislý neorientovaný graf s ohodnocením hran $w : H \mapsto \mathbf{R}^+$, pro který je $|H| \geq |U|$.

(a) Nechť T je minimální kostra grafu G. Dokažte, že existuje dvojice hran $h \in T, h' \notin T$ taková, že $T - \{h\} \cup \{h'\}$ je druhá nejlepší minimální kostra grafu G.
(b) Nechť T je kostra grafu G a pro libovolnou dvojici uzlů $u, v \in U$ označme jako $\max(u, v)$ hranu s největším ohodnocením na jediné cestě mezi uzly u a v na kostře T. Navrhněte algoritmus asymptotické časové složitosti $O(|U|^2)$, který pro zadanou kostru T určí $\max(u, v)$ pro všechny dvojice uzlů $u, v \in U$.
(c) Navrhněte efektivní algoritmus pro určení druhé nejlepší minimální kostry neorientovaného grafu G.

5.5 Huffmanovo kódování

Při určení minimální kostry neorientovaného grafu pomocí Jarníkova-Primova algoritmu je postup tvořen posloupností rozhodnutí o tom, kterou hranu přidat do vytvářené dílčí kostry. Každé z takových rozhodnutí vybírá jednu z aktuálně existujících variant podle určitého lokálního optimalizačního kritéria, nakonec se však dospěje k celkovému řešení, které splňuje globální podmínku optimality. Obdobným způsobem, charakterizovaným obecně jako **hladový algoritmus** (angl. **greedy algorithm**), je možné řešit i některé další optimalizační úlohy, má však současně svá důležitá omezení, s nimiž je třeba se obeznámit.

Pro rozhodnutí o tom, zda k řešení určité specifické úlohy bude možné použít hladového algoritmu, je důležité zjistit, zda tato úloha má dvě základní charakteristiky: *vlastnost hladového výběru* a *optimální podstruktury*.

Vlastnost hladového výběru

Pro aplikovatelnost metody hladového algoritmu je prvním požadavkem **vlastnost hladového výběru**: ke globálně optimálnímu řešení lze dospět prostřednictvím lokálně optimálních (hladových) výběrů. Podmínka, podle níž se lokální výběr provádí, by neměla záviset na řešení podproblémů, ale pouze na okamžitém stavu řešené úlohy. Princip hladového výběru spočívá v tom, že provedeme výběr a **potom** řešíme vzniklý podproblém.

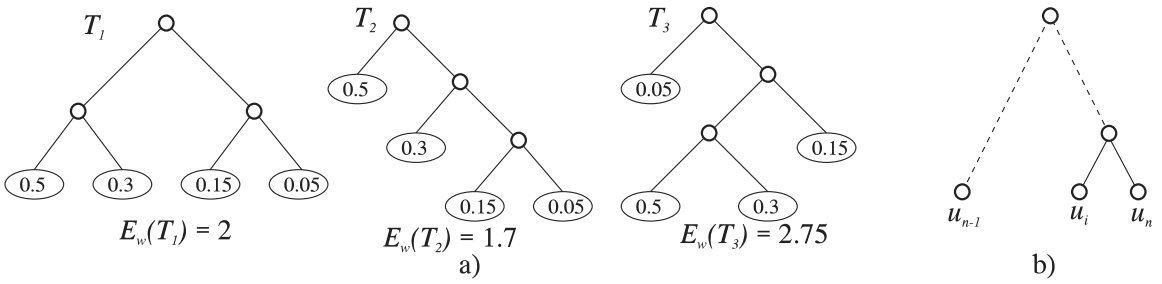
Zatímco druhá metodika řešení optimalizačních problémů – **dynamické programování** – postupuje při řešení **zdola nahoru** (tedy od podproblémů k nadřazeným problémům), hladové algoritmy postupují **shora dolů**. Postupným prováděním lokálně optimálních výběrů se každá instance problému redukuje na instanci menší.

Abychom dokázali, že určitý problém má vlastnost hladového výběru, postupujeme zpravidla tak, že uvažujeme nějaké globálně optimální řešení problému. Pro toto řešení je pak třeba dokázat, že lze změnit tak, aby jeho první krok odpovídal hladovému výběru. Tím se problém redukuje na svou jednodušší instanci a vlastnost hladového výběru se dokáže matematickou indukcí.

Optimální podstruktura

Aby měl přechod k redukované instanci problému řešeného hladovým algoritmem nějaký smysl, musí problém splňovat požadavek **optimální podstruktury**, tzn. optimální řešení problému v sobě musí obsahovat optimální řešení podproblémů. Tato vlastnost je společnou podmínkou aplikace jak hladových algoritmů tak i dynamického programování.

Hledáme-li např. nejkratší cestu spojující dva uzly v neorientovaném grafu, pak libovolná část této cesty bude současně nejkratší cestou mezi příslušnou dvojicí krajních uzlů této částečné cesty. Podobně libovolný podstrom T' minimální kostry T grafu G představuje minimální kostru podgrafu indukovaného v grafu G podmnožinou uzlů podstromu T'. Podrobný výklad teoretických základů hladových algoritmů je možné nalézt např. v [8].



Obrázek 5.8: Vnější w-délka pravidelného stromu

Stromy s minimální w-délkou

Ukážeme nyní další možnou aplikaci hladového algoritmu při hledání stromů s optimální strukturou. V odst. 3.2 jsme uvedli, jak vypadají pravidelné stromy s minimální vnější délkou. Analogicky jako v případě problému minimální kostry grafu můžeme zadání grafu obohatit o ohodnocení (tentokrát však uzlů), a dostaneme tak novou úlohu se zajímavými aplikacemi.

Definice 5.15: Nechť je dán pravidelný strom T_u stupně r s n listy, jehož každý list u_i má přiřazeno reálné číslo $w_i = w(u_i)$. **Vnější w-délkou** $E_w(T_u)$ stromu T_u nazýváme číslo definované vztahem

$$E_w(T_u) = \sum_{i=1}^{n} w_i \cdot hl(u_i),$$

kde $hl(u_i)$ je hloubka uzlu u_i, tj. jeho vzdálenost od kořene u.

Každou vzdálenost ke koncovému uzlu u_i započítáváme tedy s váhou w_i přiřazenou tomuto uzlu. Naším úkolem je pro zadané reálné hodnoty $w_1, w_2, \ldots, w_n$ určit pravidelný strom s minimální w-délkou $E_w(T_u)$ a samozřejmě též tuto minimální délku. V následujícím výkladu se omezíme pouze na případ $r = 2$, tedy na pravidelné stromy stupně 2, a namísto dlouhého termínu **pravidelný strom stupně 2 s minimální vnější w-délkou** zde budeme používat stručné označení **minimální strom**.

Praktický význam této úlohy je zřejmý: pravidelný strom stupně 2 může představovat např. větvící část nějakého programu, při níž se má rozlišit n možných případů, přitom je předem známo, že pravděpodobnost výskytu jednotlivých případů není stejná. Snažíme se tedy větvící část programu navrhnout tak, aby pravděpodobnější případy zjišťovala rychleji – to je smysl kritéria minimálnosti vnější w-délky $E_w(T_u)$. Další praktickou aplikaci související s kompresí dat uvedeme později.

Položíme-li např. $w_1 = 0,5, w_2 = 0,3, w_3 = 0,15, w_4 = 0,05$, potom pro stromy na obr. 5.8a dostaneme následující hodnoty:

$$\begin{aligned} E_w(T_1) &= 2 \cdot (0,5 + 0,3 + 0,15 + 0,05) = 2 \ , \\ E_w(T_2) &= 1 \cdot 0,5 + 2 \cdot 0,3 + 3 \cdot (0,15 + 0,05) = 0,5 + 0,6 + 0,6 = 1,7 \ , \\ E_w(T_3) &= 1 \cdot 0,05 + 2 \cdot 0,15 + 3 \cdot (0,5 + 0,3) = 0,05 + 0,3 + 2,4 = 2,75 \ . \end{aligned}$$

Ukazuje se tedy, že v tomto případě úplnost pravidelného stromu nezaručuje dosažení minimální hodnoty vnější w-délky, jako tomu bylo u vnější délky nerozlišující váhu jednotlivých listů. Velmi jednoduchý postup pro řešení této úlohy vyplyne z následujících tvrzení.

Lemma 5.16: Pro posloupnost vah $w_1 \geq w_2 \geq \ldots \geq w_n$ existuje minimální strom, v němž mají listy u_{n-1} a u_n téhož předchůdce.

Důkaz: Existuje konečně mnoho různých pravidelných stromů stupně 2 s n listy, takže pro zadanou posloupnost existuje nějaký minimální strom – označme jej T'. Předpokládejme, že

v T' nemá uzel u_n za sourozence u_{n-1}, ale uzel u_i odpovídající hodnotě $w_i > w_{n-1}$ – sourozencem musí totiž být list, neboť žádný uzel s větší vahou nemůže ležet hlouběji než u_n. Sporem dokážeme, že uzly u_i a u_{n-1} pak leží ve stejné hloubce (viz obr. 5.8b). Jejich váhy můžeme tedy zaměnit beze změny celkové w-délky stromu, a tak dostaneme strom požadovaných vlastností. △

Lemma 5.17: Strom T je minimálním stromem pro posloupnost $w_1 \geq w_2 \geq \ldots \geq w_n$ právě tehdy, je-li jeho podstrom $T' = T - \{u_{n-1}, u_n\}$ minimálním stromem pro posloupnost $w' = \{w_1, w_2, \ldots, w_{n-2}, (w_{n-1} + w_n)\}$.

Důkaz: Můžeme předpokládat, že podle předchozího lemmatu mají listy u_{n-1} a u_n v minimálním stromě T stejného předchůdce, který se po jejich vypuštění stane listem stromu T'. Pak ale platí $E_w(T) = E_w(T') + (w_{n-1} + w_n)$, takže z minimálnosti $E_w(T)$ plyne minimálnost $E_w(T')$ a naopak. △

Předchozí lemma ukazuje, že hledání minimálního stromu je úloha s vlastností hladového výběru i optimalitou podstruktur, takže můžeme použít hladového algoritmu. Pro zadanou posloupnost vah $w_1 \geq w_2 \geq \ldots \geq w_n$ se minimální strom vytváří zdola počínaje od n listů. V každém z následujících $n-1$ kroků se napojí dva listy s nejnižšími vahami na společného předchůdce, kterému se přiřadí váha rovná součtu vah těchto uzlů.

Na rozdíl od kořenových stromů, které jsme až dosud v různých algoritmech vytvářeli, náš postup vede tentokrát zdola nahoru. Pro reprezentaci stromu použijeme strukturu binárního stromu, záznam o každém uzlu bude tedy u obsahovat odkazy $left[u]$ na levého a $right[u]$ na pravého následníka, nikoliv odkaz na předchůdce. Kromě toho bude složka $w[u]$ určovat váhu uzlu u. Množina uzlů pro výběr minimálních vah se uchovává v prioritní frontě Q.

Algoritmus 5.18 Určení pravidelného stromu stupně 2 s minimální w-délkou

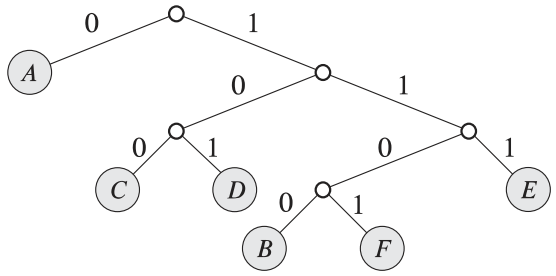
```
HUFFMAN(w, n)
1   for i := 1 to n                    Pro zadanou posloupnost vah
2       do u :=MAKE-NODE(w_i)          se vytvoří list pro každou váhu w_i
3          INSERT(u, Q)                a uloží se do prioritní fronty Q.
4   for i := 1 to n − 1                (n − 1)-krát se opakuje sloučení uzlů:
5       do x :=EXTRACT-MIN(Q)          vyber uzel s nejmenší vahou,
6          y :=EXTRACT-MIN(Q)          vyber uzel s druhou nejmenší vahou,
7          z :=MAKE-NODE(w[x] + w[y])  vytvoř jejich předchůdce,
8          left[z] := x; right[z] := y nastav jeho následníky
9          INSERT(z, Q)                a ulož jej do fronty Q.
10  return EXTRACT-MIN(Q)
```

Při určení časové složitosti algoritmu HUFFMAN předpokládáme, že prioritní frontu implementujeme jako binární haldu. Operaci MAKE-NODE lze provést v konstantním čase, takže počáteční cyklus na řádcích 1 – 3 potřebuje $O(n)$. Cyklus na řádcích 4 – 9 proběhne právě $(n-1)$-krát a na každou z operací EXTRACT-MIN a INSERT se spotřebuje $O(\lg n)$. Celková časová složitost algoritmu tedy je $O(n \lg n)$.

Prefixové kódy

Právě popsaný algoritmus je zobecněnou verzí algoritmu, který r. 1952 navrhl Huffman pro optimální kódování (viz [17]). Při běžném kódování dat se používají většinou kódy s pevnou délkou kódového slova (např. 8-bitový ASCII kód pro znaky). V případě potřeby co největší komprese dat je však účelné zvolit pro jednotlivé znaky kódová slova proměnné délky v závislosti na četnosti jejich výskytu. Huffmanovo kódování se týká použití tzv. **prefixových kódů**, u nichž žádné kódové slovo není prefixem (t.j. netvoří počátek) jiného kódového slova.

A	*B*	*C*	*D*	*E*	*F*
0	1100	100	101	111	1101



Obrázek 5.9: Binární strom prefixového kódu

Při kódování řetězu znaků se spojí odpovídající kódová slova za sebe. Pokud by tedy např. kódy znaků A až F byly zadány tabulkou z obr. 5.9a, pak bychom řetěz znaků 'ABADACAAEF' kódovali jako

A	B	A	D	A	C	A	A	E	F		
0	1100	0	101	0	100	0	0	111	1101	=	01100010101000011111101

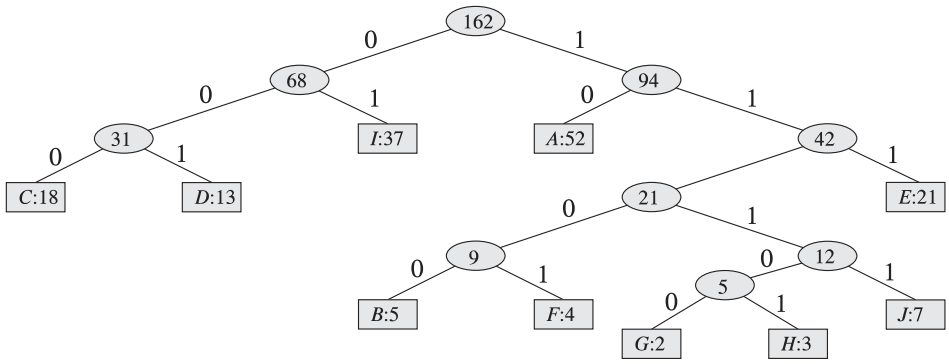
Při dekódování prefixového kódu je vhodné představit si vyjádření kódu ve formě binárního stromu, jehož hrany vedoucí k levým následníkům jsou ohodnoceny symbolem 0 a hrany k pravým následníkům symbolem 1. Každému listu odpovídá jeden kódovaný znak, přičemž odpovídající kódové slovo se získá jako posloupnost symbolů tvořících ohodnocení cesty z kořene do daného listu (viz obr. 5.9b).

Optimální prefixový kód se získá na základě znalosti četnosti výskytu jednotlivých znaků v kódovaném textu. Hodnoty četností tvoří pak posloupnost vah listů, pro kterou vytvoříme odpovídající minimální strom pomocí Huffmanova algoritmu.

Příklad 5.19: Máme nalézt optimální kódování znaků A až J prefixovým kódem pro text, v němž je četnost výskytu jednotlivých znaků dána následující tabulkou:

znak	A	B	C	D	E	F	G	H	I	J
četnost	52	5	18	13	21	4	2	3	35	7

Postupem podle Huffmanova algoritmu získáme minimální binární strom na obr. 5.10. V uzlech stromu uvádíme pro přehlednost hodnoty používaných vah, v listech stromu jsou kromě toho uvedeny i odpovídající znaky. Vlastní zakódování znaků provedeme nyní tak, že každé hraně směřující ve výsledném stromu vlevo, resp. vpravo přiřadíme číslici 0, resp. 1, a pro každý list vyjádříme k němu vedoucí cestu pomocí odpovídající posloupnosti číslic 0 a 1. Tímto způsobem dostaneme kódy uvedené v následující tabulce:



Obrázek 5.10: Binární strom optimálního prefixového kódu

znak	A	B	C	D	E	F	G	H	I	J
kód	10	11000	000	001	111	11001	110100	110101	01	11011

Výsledný binární strom má vnější w-délku dánu výrazem

$$E_w \;=\; 2 \cdot (37 + 52) + 3 \cdot (18 + 13 + 21) + 5 \cdot (5 + 4 + 7) + 6 \cdot (2 + 3) \;=\; 444,$$

takže pro zakódování daného textu bude zapotřebí 444 bitů. Při kódování s pevnou délkou musíme použít nejméně 4 bity na znak, což dává celkem 648 bitů na celý text. To znamená, že jsme text komprimovali na 68,5% minimálního objemu pro pevnou délku kódu.

Popsaný algoritmus má ještě další výhody – pokud při jeho provádění zařazujeme uzel odpovídající součtu $(w_n + w_{n+1})$ do zbytku posloupnosti vah co nejvíce vlevo (tzn. $w_i > (w_n + w_{n+1}) \geq w_{i+1}$), získáme současně binární strom s minimální hloubkou a minimální (neváženou) vnější délkou mezi všemi stromy, které minimalizují E_w. Snadná je rovněž úprava algoritmu pro obecný případ pravidelných stromů stupně $r > 2$.

Cvičení

5.5-1. Vytvořte optimální Huffmanův strom pro posloupnost vah tvořenou prvními osmi prvky Fibonacciho posloupnosti, t.j. 1,1,2,3,5,8,13,21. Je možné výsledek zobecnit na strom odpovídající posloupnosti prvních n prvků Fibonacciho posloupnosti?

5.5-2. Dokažte, že hodnotu E_w vnější w-délky binárního stromu získaného Huffmanovým algoritmem je také možné spočítat jako součet hodnot $w[u]$ přiřazených v průběhu algoritmu vnitřním uzlům stromu.

5.5-3. Dokažte, že pokud seřadíme znaky kódované pomocí Huffmanova algoritmu v nerostoucím pořadí jejich četností výskytu, potom odpovídající posloupnost délek kódů těchto znaků bude neklesající.

5.5-4. Nechť je dána množina znaků $C = \{c_1, c_2, \ldots, c_n\}$. Ukažte, že libovolný optimální způsob prefixového kódování těchto znaků lze popsat posloupností $2n - 1 + n\lceil \lg n \rceil$ bitů.
(*Návod:* Uvažujte vyjádření struktury odpovídajícího kódovacího stromu pomocí $2n - 1$ bitů tak, že vytvoříte posloupnost nul a jedniček odpovídajících ohodnocení hran v pořadí průchodu stromem do hloubky.)

5.5-5. Navrhněte zobecnění Huffmanova algoritmu pro určení pravidelného stromu stupně $r > 2$. Použijte je pak pro návrh optimálního ternárního kódování (kódová slova obsahují symboly 0, 1 a 2).

5.5-6. Předpokládejme, že textový soubor obsahuje všechny znaky osmibitového kódu ASCII bez větších rozdílů četností: maximální četnost není ani dvojnásobkem četnosti minimální. Ukažte, že v tomto případě nepřinese Huffmanovo kódování žádnou výhodu oproti kódu pevné délky.

5.5-7. Nechť je dán neorientovaný graf $G = \langle H, U \rangle$ s nezáporným ohodnocením hran $w : H \mapsto \mathbf{R}^+$. Navrhněte efektivní algoritmus určení podmnožiny hran H' grafu G, která neobsahuje kružnici a má maximální součet ohodnocení vah svých hran.