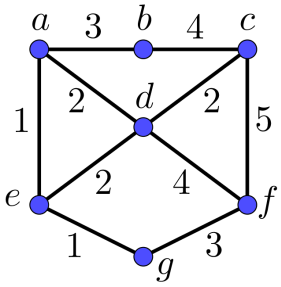
10. Minimálne kostry, Huffmanov kód

10.1 Ohodnotený graf

V praktických aplikáciach sa pomocou grafov reprezentujú rôzne druhy spojov, napríklad dopravná sieť, plošné spoje pre elektroniku, chemické väzby, náväznosti výrobných procesov atď. V týchto prípadoch obvykle nestačí mať len vrcholy a hrany grafu, ako sme mali doteraz, pretože nie je spoj ako spoj. Ak napríklad graf reprezentuje cestnú sieť, tak hrana spájajúca Bratislavu a Trnavu nemá rovnaké vlastnosti ako hrana spájajúca Bratislavu a Košice. Tieto trasy sú rôzne dlhé, majú rôzne náklady na výstavbu, rôzne prevýšenia atď. Preto potrebujeme zaviesť pojem *ohodnotený graf*.

Definícia 10.1.1 — Ohodnotený graf. Graf sa nazýva ohodnotený, ak každej jeho hrane h je priradené reálne číslo $w(h)$, ktoré nazývame *ohodnotenie hrany*, alebo *hodnota hrany* (w je skratka od „weight“=váha).

■ **Príklad 10.1**



Napríklad graf na obrázku vľavo je ohodnotený. Ak by napríklad vrcholy zodpovedali miestam na mape, hrany cestám medzi nimi a ohodnotenia hrán by boli dĺžky týchto ciest, tak by malo zmysel pýtať sa, ktorá z ciest z jedného mesta do iného mesta je najkratšia. Toto je často sa vyskytujúci praktický problém a nie vždy sa pritom musí jednať o miesta na mape a dĺžky ciest.

Napríklad na predošlom grafe má najkratšia cesta medzi vrcholmi a a f dĺžku 5 a je to cesta (a, e, g, f). ■

V príklade 10.1 sme použili pojem „najkratšia“ cesta, ktorý síce v danom kontexte bol použitý správne, ale v súvislosti s grafmi nie celkom korektne. Dĺžka cesty bola definovaná v definícii 3.3.3 (strana 55). V teórii grafov sa pod dĺžkou ťahu alebo cesty myslí vždy počet hrán daného ťahu alebo cesty. V ohodnotenom grafe, ako už bolo spomenuté, nemusia ohodnotenia hrán vždy

vyjadrovať dĺžku. V predošlom príklade by mohli vrcholy grafu predstavovať miesta na mape, hrany grafu cesty medzi nimi a ohodnotenia hrán by mohli predstavovať náklady na výstavbu týchto ciest. Otázka by potom mohla byť, ktoré z navrhnutých ciest sa majú postaviť tak, aby ich výstavba bola čo najlacnejšia. Vo všeobecnosti sa preto, v súvislosti s ohodnotenými grafmi, používa ekonomická terminológia a budeme hovoriť o hodnote sledu, ťahu alebo cesty a o najlacnejšej ceste.

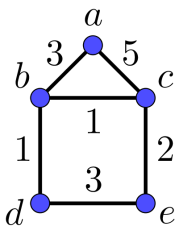
V nasledujúcej definícii si definujeme pojem *hodnota grafu*.

Definícia 10.1.2 — Hodnota grafu. Nech G je ohodnotený graf. Potom hodnota grafu, budeme ju označovať $w(G)$, je súčet ohodnotení všetkých hrán grafu G.

P S pojmom hodnota grafu sa budeme stretávať v súvislosti s hľadaním najlacnejších kostier daného grafu, čím sa budeme zaoberať v ďalšej podkapitole.

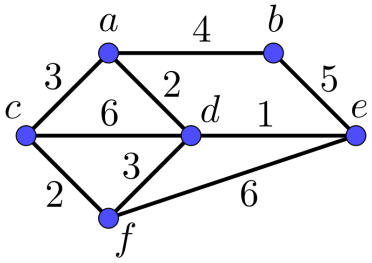
▪ Príklad 10.2

Hodnota grafu, ktorý vidíme na obrázku vpravo, je 15. Tento graf má 6 hrán, ktorých ohodnotenia usporiadané podľa veľkosti sú $(1,1,2,3,3,5)$.



▪

10.2 Minimálne kostry



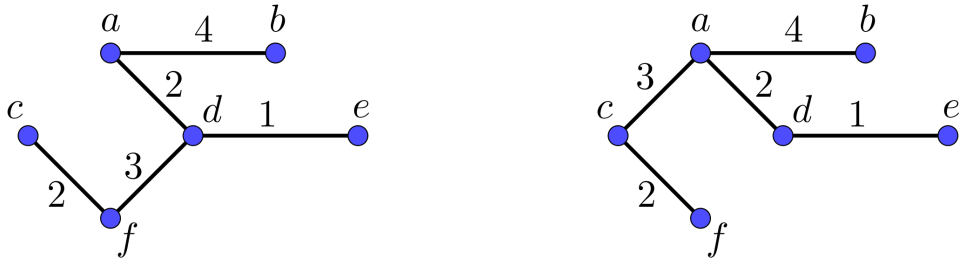
Predstavme si, že ohodnotený graf na obrázku vľavo predstavuje schématickú mapu šiestich miest. Vrcholy grafu sú mestá, hrany sú cesty medzi mestami a ohodnotenia hrán sú náklady potrebné na výstavbu príslušných ciest. Úlohou je vybudovať najlacnejší systém ciest medzi týmito mestami tak, aby medzi ľubovoľnými dvoma mestami existovala cesta. Je zrejmé, že riešením tejto úlohy bude kostra daného grafu. Výsledný graf totiž musí byť súvislý, musí obsahovať všetkých 6 vrcholov a dá sa ľahko ukázať, že ak by obsahoval cyklus, tak určite nebude najlacnejší. Budeme teda hľadať najlacnejšiu kostru grafu G. V nasledujúcej definícii si formálne definujeme pojem *minimálna kostra*, alebo *najlacnejšia kostra*.

V definícii 10.1.1 sme si definovali pojmy *ohodnotený graf* a *ohodnotenie hrany* a v definícii 10.1.2 sme definovali pojem *hodnota grafu*. Teraz si pomocou hodnoty grafu definujeme pojem *minimálna kostra*.

Definícia 10.2.1 — Minimálna, alebo najlacnejšia, kostra. Nech G je ohodnotený graf. Minimálna, alebo najlacnejšia, kostra grafu G je taká jeho kostra, ktorej hodnota je minimálna.

Podľa definície 10.2.1 môže mať graf aj viacero rôznych minimálnych kostier. Napríklad dve minimálne kostry horeuvedeného grafu, sú na obrázku 10.1. Obe tieto kostry majú hodnoty 12.

Existuje viacero algoritmov na hľadanie minimálnej kostry daného grafu. Medzi najznámejšie patria Borůvkov algoritmus, Jarníkov, v zahraničí nazývaný aj Primov, algoritmus a potom „pažravý“, Kruskalov algoritmus. Zhodou okolností je tu uvedené poradie algoritmov chronologické a zároveň aj výkonostné, pri usporiadaní od najvýkonejšieho po najmenej výkonný. V ďalšom texte však bude poradie uvedených algoritmov zmenené.



Obr. 10.1: Dve minimálne kostry grafu z úvodu tejto podkapitoly

10.2.1 Jarníkov (Primov) algoritmus

Jarníkov algoritmus na hľadanie minimálnej kostry grafu ako prvý opísal český matematik Vojtěch Jarník v roku 1930. Jeho algoritmus „znovuobjavil“ Robert Prim v roku 1957 a potom ho ešte raz „znovuobjavil“ holandský matematik Edsger Dijkstra v roku 1959. V zahraničí, najmä v anglosaskej literatúre, sa tento algoritmus väčšinou nazýva Primov. Časová zložitosť Jarníkovho algoritmu je $O\big(|E|\log|V|\big)$, avšak vhodnou implementáciou sa dá ešte urýchliť.

Princíp vytvárania kostry $S=(V',E')$ Jarníkovým algoritmom je nasledovný. Vyberieme si ľubovoľný vrchol v_1 grafu G a dáme ho do množiny V', čo je vrcholová množina S. Vrchol v_1 bude náš „štartovací“ vrchol. V každom kroku algoritmu prehľadáme všetky hrany grafu G, ktorých jeden koncový vrchol je vo V' a ich druhý koncový vrchol je vo $V\setminus V'$. Vyberieme najlacnejšiu z týchto hrán, túto hranu pridáme do E' a s ňou incidujúci vrchol z $V\setminus V'$ pridáme do V'. Algoritmus skončí keď $V'=V$.

Pri popise pseudokódu Jarníkovho algoritmu budeme používať funkciu $\texttt{minimum}(V',V\setminus V')$, ktorá nám vráti najlacnejšiu hranu (u,v), kde $u\in V'$ a $v\in V\setminus V'$ a zároveň túto hranu odstráni zo zoznamu hrán. Pseudokód Jarníkovho algoritmu je veľmi jednoduchý.

Hľadanie minimálnej kostry Jarníkovým (Primovým) algoritmom

VSTUP: Ohodnotený graf $G=(V,E)$, kde $\forall h\in E:\ w(h)\geq 0$.

VÝSTUP: Minimálna kostra $S=(V',E')$, grafu G.

INICIALIZÁCIA

$V':=\{v_1\};$ $\quad\big(v_1\in E$ je ľubovoľný vrchol grafu $G\big)$

$E':=\{\};$

VYTVÁRANIE KOSTRY S

while $\big\{V'\neq V\big)\big\}$ **do**

  $(u,v):=\texttt{minimum}(V',V\setminus V');$

  $E':=E'\cup(u,v);$

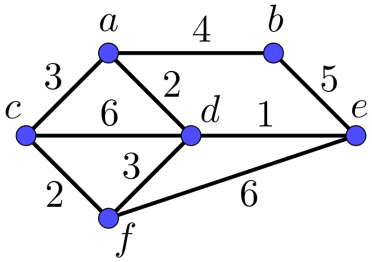
  $V':=V'\cup\{v\};$

end

Pri výbere najlacnejšej hrany medzi stromom S a vrcholom mimo stromu S v grafe G (funkcia $\texttt{minimum}(V',V\setminus V')\big)$ sa môže stať, že existuje viacero takých hrán, ktoré majú rovnakú hodnotu. V takom prípade môžeme aplikovať na výber hrany aj ďalšie kritériá, napríklad lexikografické

poradie vrcholov. Na hodnotu výslednej kostry to nemá vplyv. Avšak, v závislosti od toho aké kritériá pri výbere aplikujeme, môžeme dostať viacero rôznych minimálnych kostier. V príkladoch a v cvičeniach budeme zväčša vyberať hrany v lexikografickom usporiadaní vrcholov. Čiže ak budeme mať viaceré hrany $\{u_i, v_j\}$, ktorých hodnoty sú rovnaké, vyberieme hranu s najmenším indexom i a pokiaľ sú prvé indexy týchto hrán rovnaké, tak vyberieme hranu s najmenším indexom j. Fungovanie Jarníkovho algoritmu si ilustrujeme na príklade zo začiatku tejto časti.

■ **Príklad 10.3**

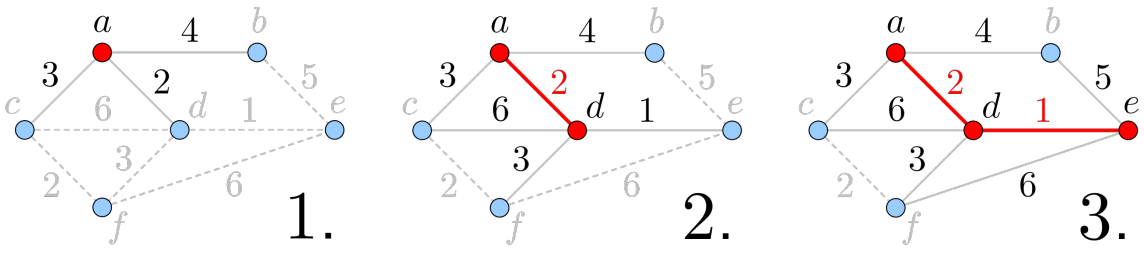


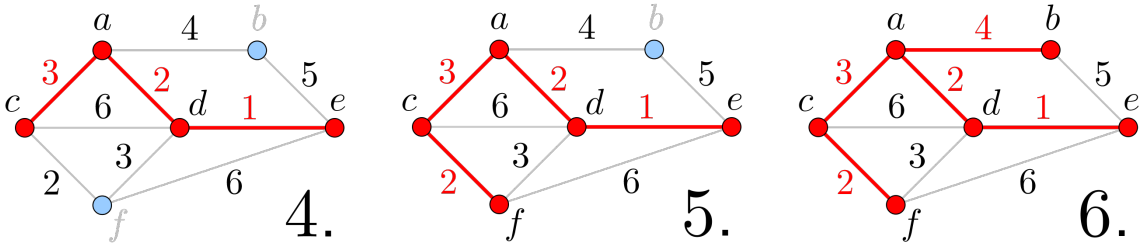
Daný je ohodnotený graf na obrázku vľavo. Pomocou Jarníkovho algoritmu nájdite jeho minimálnu kostru tak, že štartovací vrchol bude vrchol a a hrany vyberáme pri lexikografickom usporiadaní vrcholov.

Riešenie: beh algoritmu si najskôr zobrazíme v tabuľke a až potom ho znázorníme aj graficky. V uvedenej tabuľke bude každý riadok zodpovedať jednému kroku algoritmu, čiže pridaniu jednej hrany do stromu S. V prvom stĺpci tabuľky budú vrcholy, ktoré sme už pridali do konštruovaného stromu S. Algoritmus sa zastaví v momente, ked strom S bude obsahovať všetky vrcholy pôvodného grafu G. V druhom stĺpci tabuľky budú všetky hrany medzi množinami vrcholov V' a $V \setminus V'$ a hodnoty týchto hrán. Hrany budú usporiadané lexikograficky a v každom kroku vyberáme najlacnejšiu z týchto hrán. V poslednom stĺpci bude hrana, ktorú sme v príslušnom kroku vybrali.

Krok	Vrcholy V'	Hrany medzi V' a $V \setminus V'$	Vybraná hrana
1.	$\{a\}$	$(a,b):4,\ (a,c):3,\ (a,d):2$	$(a,d):2$
2.	$\{a,d\}$	$(a,b):4,\ (a,c):3,\ (d,c):6,\ (d,e):1,\ (d,f):3$	$(d,e):1$
3.	$\{a,d,e\}$	$(a,b):4,\ (a,c):3,\ (d,c):6$ $(d,f):3,\ (e,b):5,\ (e,f):6$	$(a,c):3$
4.	$\{a,c,d,e\}$	$(a,b):4,\ (c,f):2,\ (d,f):3,\ (e,b):5,\ (e,f):6$	$(c,f):2$
5.	$\{a,c,d,e,f\}$	$(a,b):4,\ (e,b):5$	$(a,b):4$
6.	$\{a,b,c,d,e,f\}$		

Graficky znázornený postup hľadania minimálnej kostry máme na nasledovnej sérii obrázkov. Vrcholy a hrany, ktoré sme už pridali do stromu S, sú zobrazené červenou farbou. Hrany, ktoré v aktuálnom kroku prehľadávame sú zobrazené súvislou čiarou. Všetky ostatné hrany sú zobrazené prerušovanou čiarou.





Vidíme, že v 3. kroku sme mali na výber hrany $\{a,c\}$ a $\{d,f\}$, obe s hodnotou 3. Keďže sme ale vyberali hrany pri lexikografickom usporiadaní vrcholov, museli sme vybrať hranu $\{a,c\}$. ∎

Dokážeme si teraz vetu, hovoriacu o koreknosti Jarníkovho algoritmu.

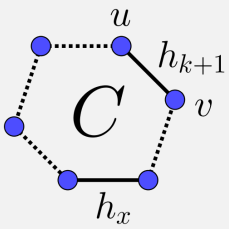
Veta 10.2.1 — O správnosti Jarníkovho algoritmu. Jarníkov algoritmus je správny, čiže po jeho skončení je S minimálna kostra daného grafu G.

Dôkaz: sa dá robiť viacerými spôsobmi, napríklad indukciou, alebo sporom. Základná myšlienka použitá v indukčnom kroku je však rovnaká ako základná myšlienka dôkazu sporom. Preto uvedieme dôkaz sporom, ktorý je o málo kratší.

Nech výsledná kostra $S = (V, E_S)$ má množinu vrcholov V, kde $|V| = n$ a množinu hrán E_S. Jarníkov algoritmus buduje kostru S ako postupnosť stromov $S_1, S_2, \ldots S_{n-1}$, pričom v každom kroku vyberie jednu hranu, ktorú potom, v nasledujúcom kroku, pridá do vznikajúcej kostry a $S_{n-1} = S$. Ďalej nech Jarníkov algoritmus vyberal hrany množiny E_S v poradí $(h_1, h_2, \ldots, h_{n-1})$. **Predpokladajme, že S nie je minimálna kostra grafu G.**

Vyberme $T = (V, E_T)$, minimálnu kostru grafu G, ktorá má maximálne také číslo k, že hrany $h_1, \ldots, h_k$ patria do kostier S aj T, avšak hrana $h_{k+1} \in E_S$ už nepatrí do kostry T.

Hrana h_{k+1} bola Jarníkovým algoritmom vybraná do kostry S v $(k+1)$. kroku, kedy už v strome S_k boli hrany $h_1, \ldots, h_k$. Nech koncové vrcholy hrany h_{k+1} sú u a v. Potom hrana $h_{k+1} = \{u, v\}$ má tú vlastnosť, že $u \in V_{S_k}$, $v \notin V_{S_k}$ a jej hodnota je najmenšia spomedzi všetkých takýchto hrán. Ak hranu h_{k+1} pridáme do kostry T, tak tam vznikne cyklus C, pretože T je kostra. V cykle C určite musia existovať hrany, rôzne od h_{k+1}, ktorých jeden vrchol patrí do V_{S_k} a druhý vrchol nepatrí do V_{S_k}. Platí totiž $u \in V_{S_k}$ a všetky hrany C, a teda aj ich vrcholy, nemôžu patriť do S_k, lebo by v S_{k+1} bol cyklus.



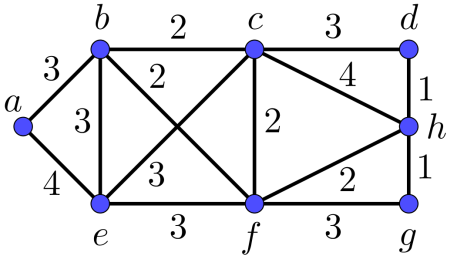
Nech h_x je takáto hrana. Potom, podľa Jarníkovho algoritmu, musí platiť $w(h_{k+1}) \leq w(h_x)$. Pozrime sa teraz na graf, ktorý z T dostaneme tak, že z neho vyhodíme hranu h_x a pridáme doň hranu h_{k+1}, čiže $T' = \big(T \setminus \{h_x\}\big) \cup \{h_{k+1}\}$. Je zrejmé, že T' bude tiež kostrou, pozri cvičenie 10.2 (str. 210). Platí preň

$$w(T') = w(T) - w(h_x) + w(h_{k+1}) \leq w(T) \;\Rightarrow\; w(T') \leq w(T)\,.$$

Posledná nerovnosť vedie k sporu. Pokiaľ je ostrá, tak je to spor s tým, že T bola minimálna kostra. A pokiaľ je uvedená nerovnosť neostrá, tak je to spor s tým, že k bolo maximálne také číslo, pre ktoré hrany $h_1, \ldots, h_k$ patria do kostier S aj nejakej minimálnej kostry grafu G, avšak hrana $h_{k+1} \in E_S$ už nepatrí do minimálnej kostry grafu G. Takže predpoklad bol nesprávny a S musí byť minimálna kostra grafu G. Q.E.D.

Uvedieme si teraz ešte jeden príklad hľadania minimálnej kostry pomocou Jarníkovho algoritmu.

■ **Príklad 10.4**

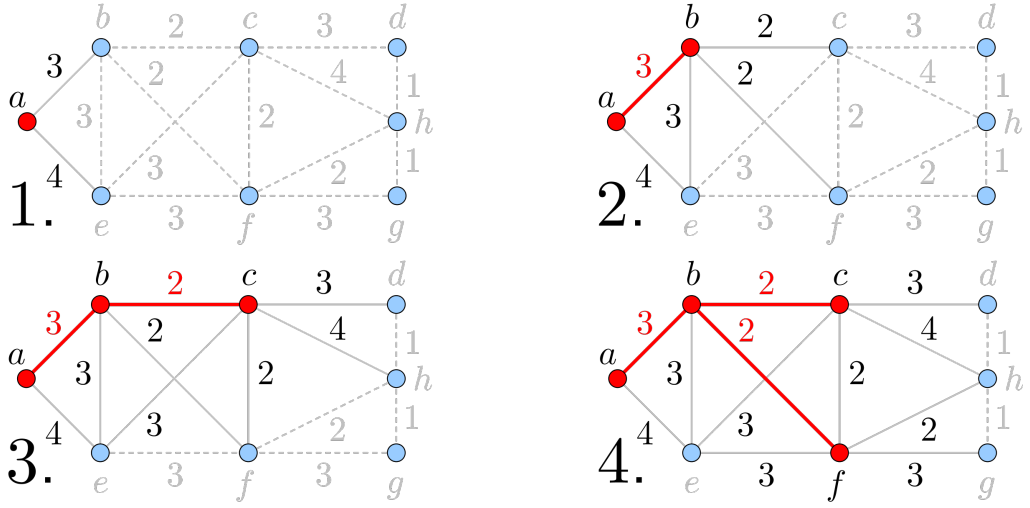


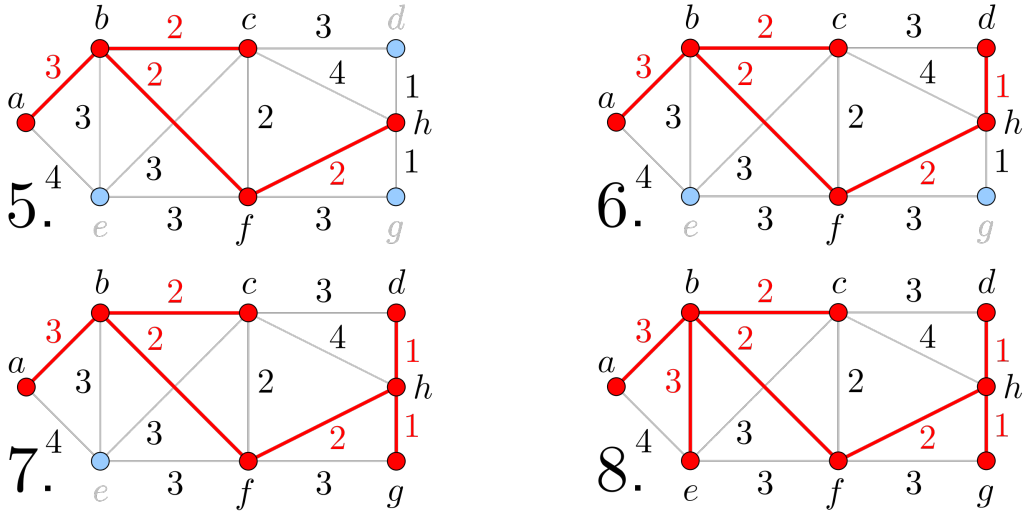
Daný je ohodnotený graf na obrázku vľavo. Pomocou Jarníkovho algoritmu nájdite jeho minimálnu kostru tak, že štartovací vrchol bude vrchol a a hrany vyberáme pri lexikografickom usporiadaní vrcholov.

Riešenie: beh algoritmu si, rovnako ako v predošlom príklade, najskôr zobrazíme v tabuľke a až potom ho znázorníme aj graficky. V uvedenej tabuľke bude každý riadok zodpovedať jednému kroku algoritmu, čiže pridaniu jednej hrany do stromu S. Na rozdiel od predošlého príkladu budeme do prvého stĺpca tabuľky písať vždy len aktuálne pridaný vrchol. Graf v tomto príklade má už viac vrcholov a bolo by nepraktické prepisovať všetky už pridané vrcholy v každom ďalšom kroku. Algoritmus sa zastaví v momente, ked prvý stĺpec tabuľky bude obsahovať všetky vrcholy pôvodného grafu G. V druhom stĺpci tabuľky budú všetky hrany medzi množinami vrcholov V' a $V \setminus V'$ a hodnoty týchto hrán. Hrany budú usporiadané lexikograficky a v každom kroku vyberáme najlacnejšiu z týchto hrán. V poslednom stĺpci bude hrana, ktorú sme v príslušnom kroku vybrali.

Krok	Pridaný vrchol	Hrany medzi V' a $V \setminus V'$	Vybraná hrana
1.	a	$(a,b):3,\ (a,e):4$	$(a,b):3$
2.	b	$(a,e):4,\ (b,c):2,\ (b,e):3,\ (b,f):2$	$(b,c):2$
3.	c	$(a,e):4,\ (b,e):3,\ (b,f):2,\ (c,d):3$ $(c,e):3,\ (c,f):2,\ (c,h):4$	$(b,f):2$
4.	f	$(a,e):4,\ (b,e):3,\ (c,d):3,\ (c,e):3$ $(c,h):4,\ (f,e):3,\ (f,g):3,\ (f,h):2$	$(f,h):2$
5.	h	$(a,e):4,\ (b,e):3,\ (c,d):3,\ (c,e):3$ $(f,e):3,\ (f,g):3,\ (h,d):1,\ (h,g):1$	$(h,d):1$
6.	d	$(a,e):4,\ (b,e):3,\ (c,e):3$ $(f,e):3,\ (f,g):3,\ (h,g):1$	$(h,g):1$
7.	g	$(a,e):4,\ (b,e):3,\ (c,e):3,\ (f,e):3$	$(b,e):3$
8.	e		

Graficky znázornený postup hľadania minimálnej kostry máme na nasledovnej sérii obrázkov. Vrcholy a hrany, ktoré sme už pridali do stromu S, sú zobrazené červenou farbou. Hrany, ktoré v aktuálnom kroku prehľadávame sú zobrazené súvislou čiarou.





■

10.2.2 Kruskalov algoritmus

Kruskalov algoritmus na hľadanie minimálnej kostry grafu je známy aj pod označením „pažravý algoritmus“. Prvý raz bol opísaný americkým matematikom Josephom Kruskalom v roku 1956. Tento algoritmus je veľmi jednoduchý na pochopenie aj realizáciu, ale jeho časová zložitosť je $O\big(|E|\log|E|\big)$, čiže je z tu spomínaných troch algoritmov najmenej efektívny.

Princíp konštrukcie minimálnej kostry pomocou Kruskalovho algoritmu je veľmi jednoduchý. Z grafu G postupne vyberáme do kostry hrany tak, že v každom kroku vyberieme hranu s najnižšou hodnotou, ktorej pridaním nevznikne cyklus. Akonáhle je počet vybraných hrán o 1 menší než počet vrcholov, môžeme skončiť, pretože už máme kostru.

Ak by sme na konštrukciu kostry mali nejaké špeciálne požiadavky, napríklad vyberanie hrán v nejakom vopred stanovenom poradí, tak to musíme zohľadniť pri pridávaní hrán do kostry. Samozrejme stále platí stratégia „pažravého“ algoritmu, čiže vždy vyberáme najlacnejšiu možnú hranu, ale ak máme na výber viacero najlacnejších hrán, môžeme aplikovať aj doplnkové kritéria, napríklad vyberanie hrán v lexikografickom usporiadaní.

Pri zápise Kruskalovho algoritmu budeme používať funkciu `usporiadaj_vzostupne`(E), ktorá nám množinu ohodnotených hrán usporiada vzostupne, ako usporiadanú k-ticu hrán. Okrem toho budeme používať aj funkciu `prvy`(P), ktorá nám z usporiadanej k-tice P vráti jej prvý prvok a zároveň tento prvok z P odstráni. Pseudokód Kruskalovho algoritmu je uvedený na strane 202 hore.

Dôkaz korektnosti Kruskalovho algoritmu, t. j. dôkaz toho, že výsledná kostra S skonštruovaná pomocou Kruskalovho algoritmu je minimálna, je jednoduchý. Uvádzať ho tu však nebudeme. Záujemci môžu tento dôkaz nájsť napríklad v knihe [15], v časti 4.4.3. Aj keď je tento algoritmus menej efektívny než Borůvkov, alebo Jarníkov algoritmus, pre mnohé aplikácie je postačujúci a jeho najväčšou výhodou je jeho jednoduchosť a ľahká pochopiteľnosť.

Najviac prekvapivým faktom pri Kruskalovom algoritme je to, že funguje! Stratégia „pažravosti“ je totiž známa aj z iných problémov diskrétnej matematiky, ale väčšinou nevedie k optimálnemu riešeniu. Dokonca pri viacerých problémoch vedie ku „katastrofickým“ výsledkom. Preto je veľmi prekvapivé, že tento „pažravý“ algoritmus dáva pri hľadaní minimálnej kostry grafu optimálne riešenie.

Teraz si uvedieme pseudokód Kruskalovho algoritmu a potom si jeho fungovanie ilustrujeme na ohodnotených grafoch z príkladov 10.3 a 10.4, čiže na rovnakých grafoch, na akých sme si predtým ilustrovali fungovanie Jarníkovho algoritmu.

Hľadanie minimálnej kostry Kruskalovým algoritmom

VSTUP: Ohodnotený graf $G = (V, E)$, kde $\forall h \in E : w(h) \geq 0$.

VÝSTUP: Minimálna kostra $S = (V, E')$, grafu G.

INICIALIZÁCIA

$P :=$ usporiadaj_vzostupne(E);

$E' := \{\}$;

VYTVÁRANIE KOSTRY S

while $\{\, |E'| < (|V| - 1) \,\}$ **do**

  $h :=$ `prvy`(P);

  if $\{$ *pridaním hrany* h *do* E', *nevznikne cyklus v* S $\}$ **then**

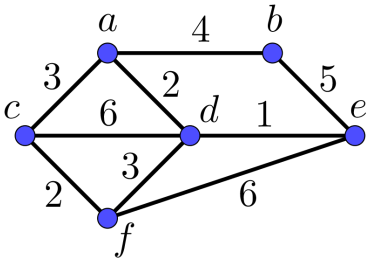
    $E' := E' \cup \{h\}$;

  end

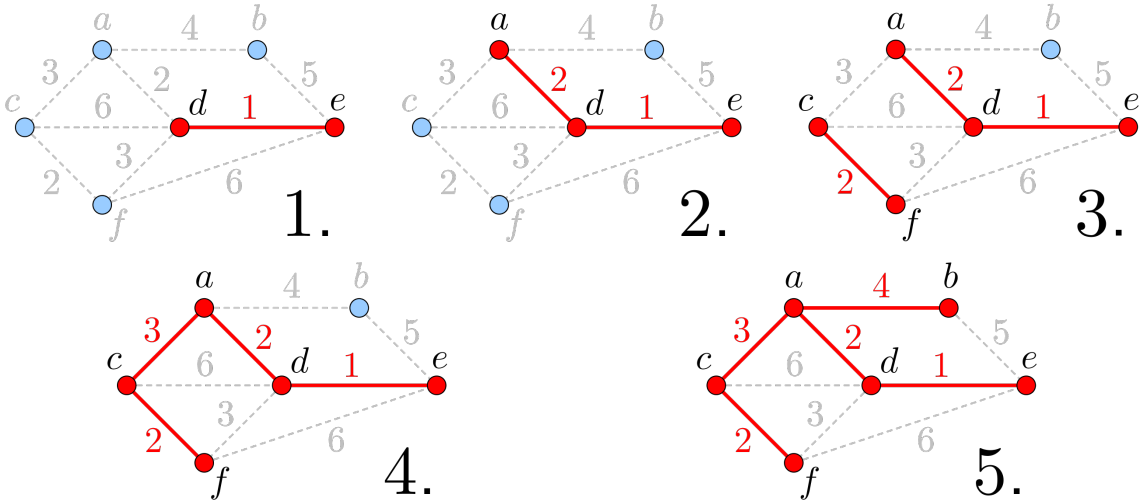
end

■ **Príklad 10.5**

Daný je ohodnotený graf na obrázku vpravo. Pomocou Kruskalovho algoritmu nájdite jeho minimálnu kostru tak, že štartovací vrchol bude vrchol a a hrany vyberáme pri lexikografickom usporiadaní vrcholov.

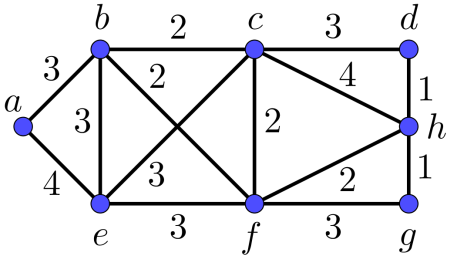


Riešenie: beh algoritmu si znázorníme len graficky. Hrany sa do kostry budú pridávať po jednej a v každom kroku dáme do konštruovanej kostry najlacnejšiu hranu, po ktorej pridaní nevznikne cyklus. Samozrejme spolu s hranou vždy pridávame aj s ňou incidujúce vrcholy. Vrcholy a hrany, ktoré sme už pridali do konštruovanej kostry, sú zobrazené červenou farbou.



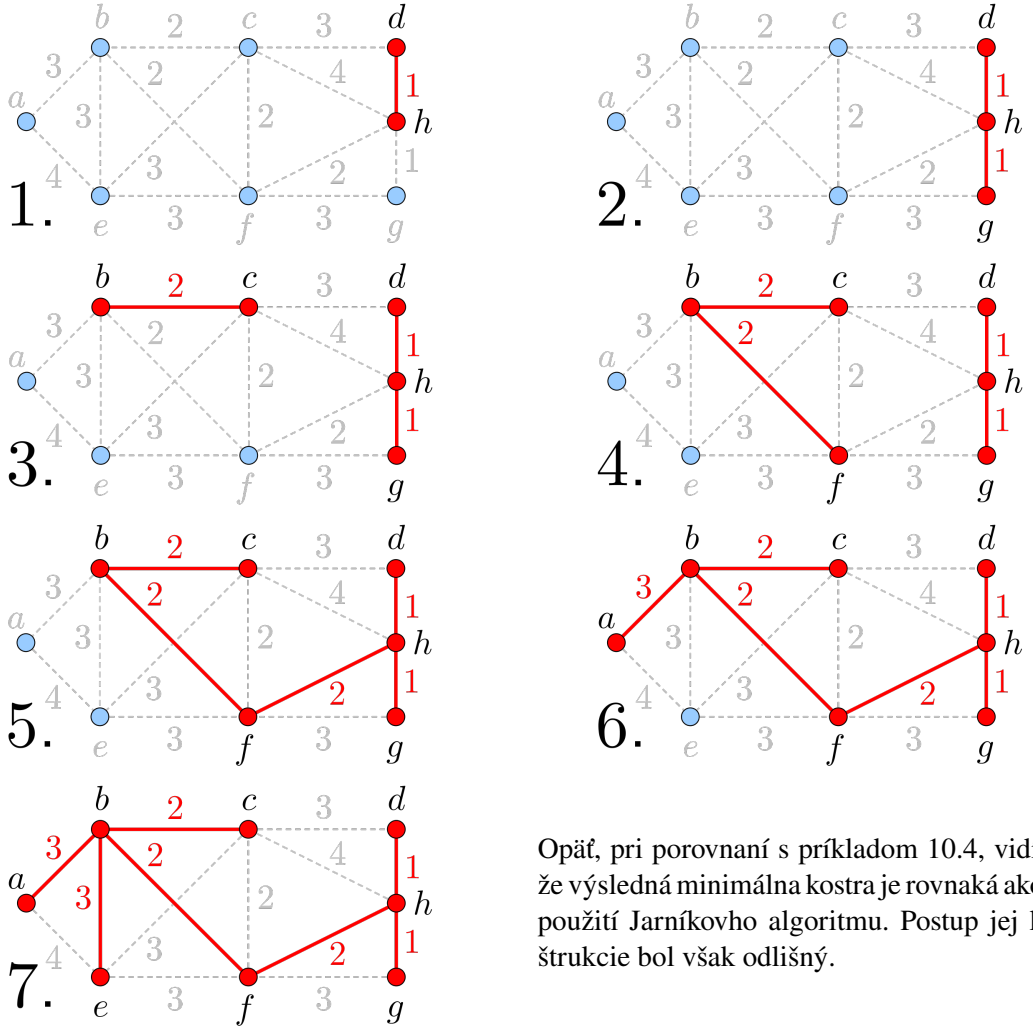
Ako môžeme pri porovnaní s príkladom 10.3 vidieť, dostali sme rovnakú najlacnejšiu kostru ako pri použití Jarníkovho algoritmu. Avšak postup konštrukcie tejto minimálnej kostry bol odlišný. Vo všeobecnosti nám Jarníkov a Kruskalov algoritmus nemusia dať rovnaké minimálne kostry, a to ani v prípade, že hrany vyberáme pri lexikografickom usporiadaní vrcholov. ■

■ **Príklad 10.6**



Daný je ohodnotený graf na obrázku vľavo. Pomocou Kruskalovho algoritmu nájdite jeho minimálnu kostru tak, že štartovací vrchol bude vrchol a a hrany vyberáme pri lexikografickom usporiadaní vrcholov.

Riešenie: beh algoritmu si znázorníme len graficky. Hrany sa do kostry budú pridávať po jednej a v každom kroku dáme do konštruovanej kostry najlacnejšiu hranu, po ktorej pridaní nevznikne cyklus. Samozrejme spolu s hranou vždy pridávame aj s ňou incidujúce vrcholy. Vrcholy a hrany, ktoré sme už pridali do konštruovanej kostry, sú zobrazené červenou farbou.



Opäť, pri porovnaní s príkladom 10.4, vidíme, že výsledná minimálna kostra je rovnaká ako pri použití Jarníkovho algoritmu. Postup jej konštrukcie bol však odlišný.

■

10.2.3 Borůvkov algoritmus

Borůvkov algoritmus na hľadanie minimálnej kostry grafu je najefektívnejší spomedzi troch tu spomínaných algoritmov. Tento algoritmus vymyslel v roku 1926 moravský matematik Otakar Borůvka pri riešeni návrhu optimálnej elektrorozvodnej siete na Morave. To znamená, že Borůvkov algoritmus vznikol pri riešení výsostne praktického problému v čase, keď teória grafov ako samostatná matematická disciplína ešte len vznikala. Ako to už ale býva, jeho algoritmus bol neskôr „znovuobjavený“ ešte aspoň piatimi ďalšími matematikmi. V západnej, predovšetkým anglosaskej,

literatúre sa Borůvkov algoritmus nazýva Sollinov algoritmus. Tento algoritmus sa používa hlavne pri paralelnom spracovaní dát. Jeho časová zložitosť je $O\big(|E|\log|V|\big)$, avšak pre niektoré kategórie grafov je ešte výrazne rýchlejší. Napríklad pre rovinné grafy je jeho časová zložitosť lineárna. Okrem toho z Borůvkovho algoritmu vychádza aj doteraz najrýchlejší algoritmus pre hľadanie minimálnej kostry, ktorého časová zložitosť je $O(|V|+|E|)$.

Borůvkov algoritmus v tomto texte nebudeme uvádzať, pretože je zložitejší než Kruskalov alebo Jarníkov algoritmus. V Borůvkovom algoritme sa napríklad vyžaduje, aby ohodnotenia ľubovoľných dvoch hrán daného grafu boli rôzne. Toto obmedzenie sa dá pomerne ľahko obísť, čiže Borůvkov algoritmus môžeme použiť aj na grafy, ktoré majú rovnako ohodnotené hrany, ale samozrejme je to potom zložitejšie než „pažravý“, alebo Jarníkov algoritmus. Záujemci môžu nájsť popis Borůvkovho algoritmu napríklad v knihe [15], v časti 4.5.3.

10.3 Huffmanov kód

Znaky v počítači sa obvykle reprezentujú binárnymi reťazcami pevnej dĺžky. Napríklad ASCII[1] kóduje každý znak 7-bitovým reťazcom. Čiastočný príklad ASCII kódu je zobrazený v tabuľke 10.1. Takýto spôsob kódovania znakov je neefektívny z hľadiska priemerného počtu bitov potreb-

Znak	ASCII kód	Znak	ASCII kód	Znak	ASCII kód	Znak	ASCII kód
*	0101010	0	0110000	A	1000001	a	1100001
+	0101011	1	0110001	B	1000010	b	1100010
–	0101101	2	0110010	C	1000011	c	1100011
/	0101111	3	0110011	D	1000100	d	1100100
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Tabuľka 10.1: Časť ASCII tabuľky

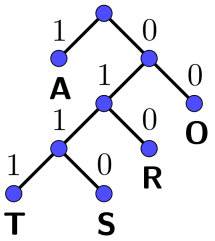
ných na zakódovanie daného textového reťazca, pretože má veľkú redundanciu. Huffmanov kód je výsledkom snahy optimalizovať tento priemerný počet bitov. V Huffmanovom kóde sa znaky kódujú binárnymi reťazcami premenlivej dĺžky. Často používané znaky sa kódujú kratšími a málo používané znaky dlhšími binárnymi reťazcami. Tým sa zredukuje priemerný počet bitov potrebných na zakódovanie daného textového reťazca. Huffmanov kód patrí medzi tzv. *entropické kódy*. Pri entropických kódoch sa frekventovanejšie znaky kódujú kratšími a menej frekventované znaky dlhšími kódovými slovami. V roku 1952 americký informatik David A. Huffman, spolu s Robertom Fanom v rámci seminárnej práce na MIT vyvinuli algoritmus na konštrukciu optimálneho P-kódu. Huffmanov kód má, pre daný súbor znakov s danými relatívnymi početnosťami (resp. len početnosťami) výskytu, minimálnu možnú redundanciu a používa sa napríklad pri bezstratovej kompresii. Ako už bolo spomenuté, je to prefixový kód[2] (P-kód) s kódovými slovami premenlivej dĺžky. Definuje sa pomocou koreňového stromu, pričom koncové vrcholy stromu reprezentujú kódované znaky a cesta od koreňa stromu po koncový vrchol predstavuje kódové slovo zodpovedajúce tomuto koncovému vrcholu. Pri dekódovaní binárneho reťazca postupujeme od koreňa stromu nadol, pričom vo vetveniach 1 znamená doľava a 0 znamená doprava[3]. Keď dôjdeme do koncového vrcholu stromu, zapíšeme si príslušný znak a pokračujeme opäť od koreňa smerom dole. Ukážeme si to na dvoch príkladoch.

[1] American Standard Code for Information Interchange

[2] Kód žiadneho znaku nie je prefixom (začiatkom) kódu iného znaku.

[3] Je to vecou dohody.

■ **Príklad 10.7**



V koreňovom strome, na obrázku vľavo, sú znaky A, O, R, S, T zakódované nasledovne

A	1
O	00
R	010
S	0110
T	0111

Pri kódovaní postupujeme od koreňa nadol, až po koncový vrchol predstavujúci kódovaný znak. Keďže ide o prefixový kód oddeľovacie znaky nepotrebujeme. Napríklad slová ROSA a RAST sa zakódujú ako

ROSA	0100001101
RAST	 010101100111

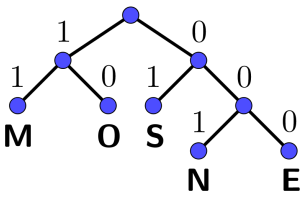
Pri dekódovaní postupujeme od koreňa po koncový vrchol, zapíšeme si príslušný znak a začneme opäť od koreňa. Napríklad

$$10001001111 \quad = \quad 1 \,|\, 00 \,|\, 010 \,|\, 0111 \,|\, 1 \quad = \quad \text{AORTA}$$
$$0111010101101 \quad = \quad 0111 \,|\, 010 \,|\, 1 \,|\, 0110 \,|\, 1 \quad = \quad \text{TRASA}$$

■

Uvedieme si ešte jeden príklad.

■ **Príklad 10.8**



Koreňovým stromom máme zakódované znaky M, O, S, N, E tak, ako to vidíme na obrázku vľavo.

M	11
O	10
S	01
N	001
E	000

Kódovanie

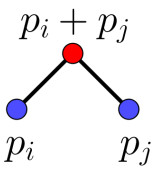
SEM	0100011
MENO	1100000110

Dekódovanie

$$001100111000 \quad = \quad 001 \,|\, 10 \,|\, 01 \,|\, 11 \,|\, 000 \quad = \quad \text{NOSME}$$

■

Ak jeden textový reťazec zakódujeme rôznymi Huffmanovými kódmi, tak môžeme dostať binárne reťazce rôznej dĺžky. Otázkou je, či je možné zostrojiť optimálny Huffmanov kód s minimálnou, ideálne nulovou, redundanciou (nadbytočnou informáciou). Odpoveď ná túto otázku je *áno*. Práve Huffmanov kód navrhnutý v roku 1952 má minimálnu možnú redundanciu.



Pri popise algoritmu na zostavovanie koreňového stromu pre optimálny Huffmanov kód, budeme používať operáciu *spájania stromov*. Pomocou tejto operácie budeme spájať dva existujúce koreňové stromy s ohodnotenými koreňmi do jedného nového koreňového stromu s ohodnoteným koreňom. Spájanie koreňových stromov bude prebiehať tak, že korene daných dvoch stromov spojíme hranami s novým vrcholom, ktorý bude koreňom novovzniknutého stromu. Okrem toho, ak pôvodné dva stromy mali korene s hodnotami p_i a p_j, tak koreň nového stromu bude mať hodnotu $p_i + p_j$. Na obrázku vyššie je uvedený príklad spojenia dvoch koreňových stromov. Modré vrcholy s hodnotami p_i a p_j sú korene pôvodných dvoch stromov a červený vrchol s hodnotou $p_i + p_j$ je koreň nového stromu vzniknutého ich spojením. Teraz si uvedieme algoritmus konštrukcie koreňového stromu pre optimálny Huffmanov kód.

Algoritmus konštrukcie koreňového stromu optimálneho Huffmanovho kódu

VSTUP: Zoznam n znakov s danými relatívnymi početnosťami (alebo len početnosťami).

VÝSTUP: Koreňový strom T, určujúci optimálny Huffmanov kód.

INICIALIZÁCIA

Každému znaku prideľ vrchol s hodnotou rovnajúcou sa jeho relatívnej početnosti;

KONŠTRUKCIA STROMU

while $\{$*výsledný graf nie je strom*$\}$ **do**

a) *Nájdi dva stromy s najlacnejšími koreňmi. V prípade viacerých stromov, ktorých korene majú rovnakú hodnotu, vyber dva s najmenšou výškou*;

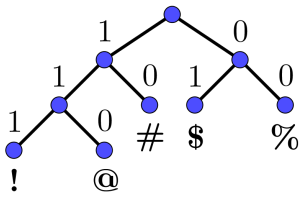
b) *Spoj vybrané dva stromy do jedného nového a jeho koreňu daj hodnotu, ktorá sa rovná súčtu hodnôt koreňov pôvodných dvoch stromov*;

end

Existuje viacero alternatívnych podôb zápisu uvedeného algoritmu. My si neskôr na príkladoch ukážeme ešte jeden ďalší spôsob konštrukcie koreňového stromu pre optimálny Huffmanov kód.

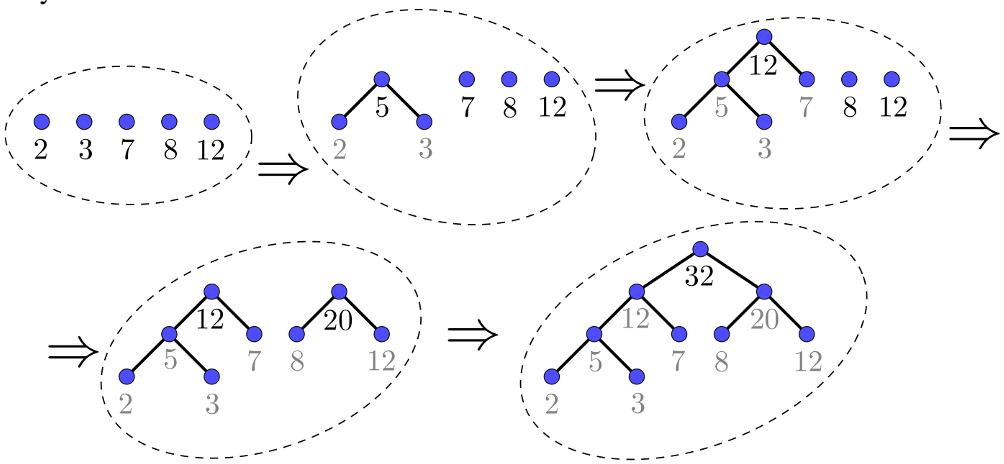
■ **Príklad 10.9** Zostrojte koreňový strom pre optimálny Huffmanov kód ak je daných 5 znakov, s početnosťami výskytu podľa nasledovnej tabuľky.

znak	početnosť
!	2
@	3
#	7
$	8
%	12



Obr. 10.2: Výsledný koreňový strom

Konštrukcia koreňového stromu bude, podľa uvedeného algoritmu, prebiehať v piatich krokoch zobrazených na nasledovnom obrázku.

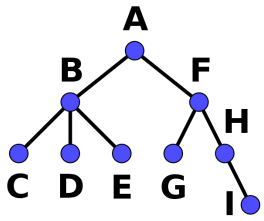


Výsledný koreňový strom pre optimálny Huffmanov kód je zobrazený na obrázku 10.2 vyššie. ■

Ako vidíme z príkladu 10.9, na začiatku máme les pozostávajúci z toľkých izolovaných vrcholov, koľko je kódovaných znakov. Tieto vrcholy postupne spájame, pričom v každom kroku

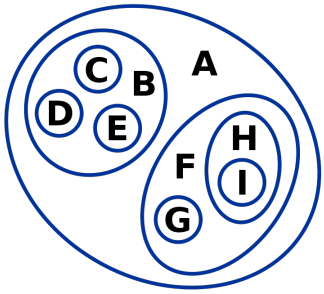
sa počet aktívnych vrcholov, čiže takých, ktoré ešte budeme spájať, zníži o 1. Skončíme vtedy, keď dostaneme strom, t. j. súvislý graf. Inak povedané, skončíme, keď nám zvýši už len 1 aktívny vrchol. Tento postup je veľmi názorný a aj popis algoritmu je jednoduchý. Pri realizácii na počítači však potrebujeme nejakú jednoduchšiu „neobrázkovú“ reprezentáciu konštruovaného stromu.

Pomocou koreňových stromov môžeme reprezentovať aj rôzne iné dátové štruktúry. Samozrejme platí to aj opačne, čiže pomocou iných, „neobrázkových“, dátových štruktúr vieme reprezentovať koreňové stromy.



Koreňový strom, ktorý je na obrázku vľavo, sa napríklad dá ekvivalentne reprezentovať uzátvorkovaným výrazom

$$\Big(A\big(B(C)(D)(E)\big)\big(F(G)(H(I))\big)\Big)$$



Alebo Vennovým diagramom „do seba zapadajúcich“ množín tak, ako to je zobrazené na obrázku vľavo.

Alebo pomocou odsadených riadkov, podobne ako sa to robí pri písaní písaní kapitol, podkapitol a odsekov v obsahu kníh. Ukážku vidíme vpravo.

A.....
B.....
C.....
D.....
E.....
F.....
G.....
H.....
I.....

Ako vidíme, vrcholy grafu môžeme nahradiť množinami a spájanie koreňových stromov nahradíme spájaním existujúcich množín do dvojprvkových množín. Pritom prvkom množiny môže byť aj iná množina. Predvedieme si to na rovnakej úlohe ako v príklade 10.9.

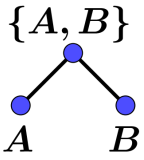
■ Príklad 10.10

znak	početnosť
!	2
@	3
#	7
$	8
%	12

Zostrojte koreňový strom pre optimálny Huffmanov kód, ak je daných 5 znakov s početnosťami výskytu podľa tabuľky, ktorú vidíme vľavo. Namiesto izolovaných vrcholov v prvom grafe z príkladu 10.9, začneme množinou obsahujúcou početnosti výskytu daných znakov.

- Na začiatku budeme mať množinu $\{2,3,7,8,12\}$.
- Pod vonkajšou množinou budeme rozumieť množinu ohraničenú krajnou ľavou a krajnou pravou zátvorkou.
- Pod hodnotou množiny budeme rozumieť súčet hodnôt všetkých jej *„prvkov“*. Čiže množina $\{2\}$ má hodnotu 2, množina $\{2,3\}$ má hodnotu 5 a množina $\{\{2,3\},7\}$ má hodnotu 12.
- Pokiaľ vonkajšia množina má viac než 2 prvky, tak budeme opakovať nasledovné kroky
  - Vo vonkajšej množine nájdeme 2 prvky s najmenšou hodnotou. Ak máme takých prvkov viac, tak vyberieme také dva, ktoré majú najmenšiu *„hĺbku“*, t. j. počet úrovní vnorených množín.
  - Vybrané dva prvky vonkajšej množiny spojíme do dvojprvkovej množiny.
- V zadanej úlohe bude postup nasledovný

$$\{2,3,7,8,12\} \to \{\{2,3\},7,8,12\} \to$$
$$\to \{8,12,\{\{2,3\},7\}\} \to \{\{\{2,3\},7\},\{8,12\}\}$$



Po poslednom kroku už má vonkajšia množina len 2 prvky a môžeme z nej zostrojiť koreňový strom pre optimálny Huffmanov kód. Každú dvojprvkovú množinu $\{A,B\}$, pričom A a B môžu byť tiež dvojprvkové množiny, zakreslíme tak, ako to vidíme na obrázku vľavo. Takto z množiny $\Big\{\{\{2,3\},7\},\{8,12\}\Big\}$ dostaneme rovnaký koreňový strom, ako v príklade 10.9 (obrázok 10.2). ■

V prípade množinového zápisu sme prvky vonkajšej množiny usporiadavali vzostupne podľa ich hodnôt a pri prvkoch s rovnakou hodnotou podľa ich *„hĺbky“*. Pri kreslení koreňového stromu, podľa uvedeného algoritmu, sme to nerobili. Už z toho vyplýva, že výsledný koreňový strom môže mať viacero podôb. Postup konštrukcie koreňového stromu pre optimálny Huffmanov kód si ukážeme ešte na jednom príklade.

■ **Príklad 10.11**

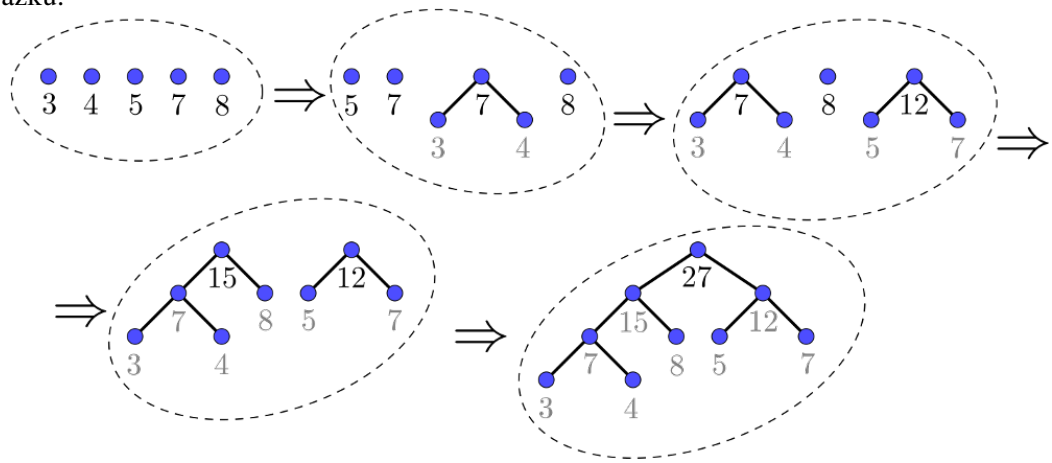
znak	početnosť
A	3
B	4
C	5
D	7
E	8

Zostrojte koreňový strom pre optimálny Huffmanov kód ak je daných 5 znakov, s početnosťami výskytu podľa tabuľky vľavo.
Úlohu budeme riešiť dvoma spôsobmi. Najskôr pomocou množinového zápisu a potom kreslením koreňového stromu podľa uvedeného algoritmu.

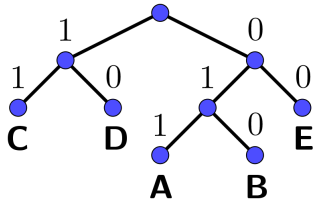
Postup pomocou množinového zápisu bude vyzerať nasledovne.

$$\Big\{3,4,5,7,8\Big\} \to \Big\{5,7,\{3,4\},8\Big\} \to \Big\{\{3,4\},8,\{5,7\}\Big\} \to \Big\{\{5,7\},\{\{3,4\},8\}\Big\}$$

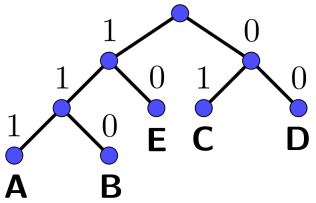
Postup konštrukcie koreňového stromu podľa uvedeného algoritmu je zobrazený na nasledujúcom obrázku.[4]



V poslednom koreňovom strome ešte treba nahradiť početnosti znakmi, ktorým zodpovedajú koncové vrcholy stromu. Výsledné koreňové stromy máme zobrazené na nasledujúcich obrázkoch.



Obr. 10.3: Strom získaný z množinového zápisu



Obr. 10.4: Strom získaný pomocou algoritmu

[4] Všimnite si, že vo všetkých krokoch, okrem posledného, sú stromy usporiadané vzostupne podľa ich hodnoty. V poslednom kroku sú usporiadané opačne, čoho dôsledkom je rôzna podoba výsledných koreňových stromov a kódu.

Ako vidíme, koreňový strom zrekonštruovaný z množinového zápisu je iný, než strom, ktorý sme dostali druhým spôsobom. Preto aj kódy niektorých znakov sa líšia, avšak oba koreňové stromy reprezentujú optimálny Huffmanov kód. ∎

Všimnime si, že Huffmanov kód tvorí úplný binárny strom. Ak sa pozrieme na algoritmus tvorby Huffmanovho kódu, tak vidíme, že podstromy vždy spájame tak, aby výsledný strom bol úplný. A ak si pozrieme algoritmus tvorby optimálneho Huffmanovho kódu, uvedený na strane 206, tak vidíme, že tento algoritmus vždy vytvorí nielen úplný binárny strom, ale aj binárny strom minimálnej možnej výšky, ktorá je pri daných vstupných údajoch a ich poradí možná. Pre tvorbu úplného binárneho stromu s minimalizovanou výškou je v uvedenom algoritme kľúčová podčiarknutá podmienka

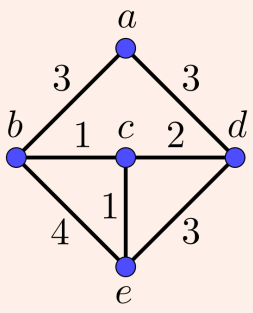
> *Nájdi dva stromy s najlacnejšími koreňmi. V prípade viacerých stromov, ktorých korene majú rovnakú hodnotu, vyber dva s najmenšou výškou.*

Aj bez podčiarknutej podmienky by uvedený algoritmus vytvoril optimálny Huffmanov kód, pokiaľ by sme ako jediné kritérium „optimálnosti“ brali redundanciu vytvoreného Huffmanovho kódu. Pri rýchlom prehľadávaní je však často dôležité, aby vytvorený kód mal nielen čo najnižšiu redundanciu, ale aby jeho kódové slová boli čo najkratšie. Bez podčiarknutej podmienky by sme mohli dostať kód, ktorý by mal niektoré kódové slová príliš dlhé a na druhej strane by tieto boli vykompenzované krátkymi kódovými slovami. Takže redundancia by bola minimálna, ale pre vyhľadávanie by takýto kód nebol ten najlepší možný. Preto pre praktické aplikácie potrebujeme mať úplný binárny strom s minimálnou možnou výškou.

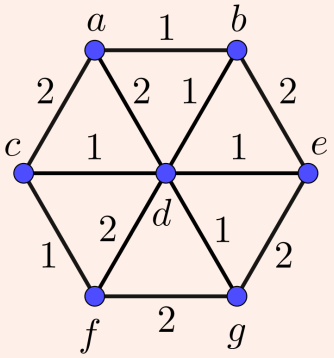
10.4 Cvičenia

Cvičenie 10.1 Pomocou Jarníkovho aj Kruskalovho algoritmu nájdite minimálne kostry daných grafov. Hrany vyberajte pri lexikografickom usporiadaní vrcholov.

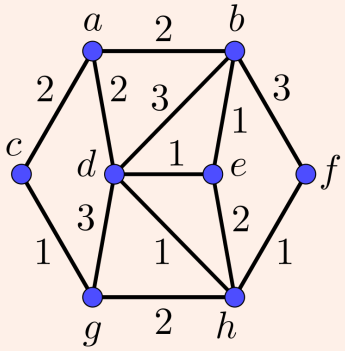
(a)



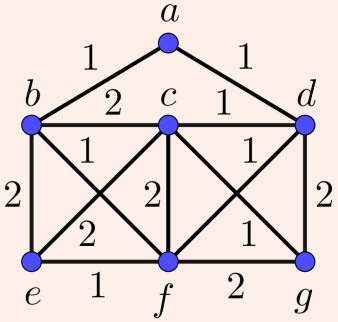
(b)



(c)



(d)



∎

Cvičenie 10.2 V dôkaze vety 10.2.1 (strana 199) konštruujeme z kostry T, graf T' nasledovným spôsobom

> *Graf T' dostaneme z T tak, že z neho vyhodíme hranu h_x a pridáme doň hranu h_{k+1}, čiže $T' = \big(T \setminus \{h_x\}\big) \cup \{h_{k+1}\}$.*

Dokážte, že graf T' je tiež kostra pôvodného grafu G. ■

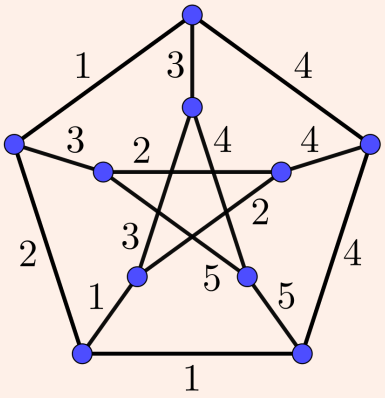
Cvičenie 10.3 V dôkaze vety 10.2.1 (strana 199) je uvedené tvrdenie

> *Ak hranu h_{k+1} pridáme do kostry T, tak tam vznikne cyklus C, pretože T je kostra.*

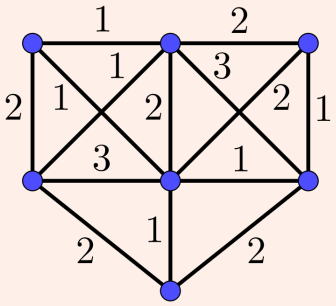
Dokážte toto tvrdenie. ■

Cvičenie 10.4 Pomocou Jarníkovho aj Kruskalovho algoritmu nájdite minimálne kostry daných grafov.

(a)



(b)



■

Cvičenie 10.5 Nájdite ohodnotenie hrán grafu K_6 číslami $\{1,2,\ldots,15\}$ (každé číslo sa použije práve raz) také, aby hodnota minimálnej kostry tohto grafu bola

(a) minimálna, (b) maximálna. ■

Cvičenie 10.6 Nájdite ohodnotenie hrán grafu $K_{3,4}$ číslami $\{1,2,\ldots,12\}$ (každé číslo sa použije práve raz) také, aby hodnota minimálnej kostry tohto grafu bola

(a) minimálna, (b) maximálna. ■

Cvičenie 10.7 Nájdite ohodnotenie hrán grafu Q_3 (pozri cvičenie 5.21, strana 118) číslami $\{1,2,\ldots,12\}$ (každé číslo sa použije práve raz) také, aby hodnota jeho minimálnej kostry bola

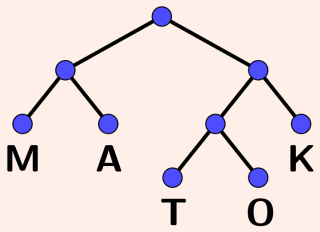
(a) minimálna, (b) maximálna. ■

Cvičenie 10.8 Nájdite ohodnotenie hrán grafu K_6 číslami $\{1,2,\ldots,15\}$ (každé číslo sa použije práve raz) také, aby hodnota minimálnej kostry tohto grafu bola maximálna a súčasne bola táto kostra cestou. ■

Cvičenie 10.9 V Pythone alebo v C naprogramujte vytvorenie minimálnej kostry daného grafu pomocou Kruskalovho algoritmu. ■

Cvičenie 10.10 V Pythone alebo v C naprogramujte vytvorenie minimálnej kostry daného grafu pomocou Jarníkovho algoritmu. ■

Cvičenie 10.11



Na obrázku vľavo je koreňový strom určujúci Huffmanov kód (ľavá hrana znamená 1 a pravá hrana znamená 0).

(a) Zakódujte slová

- MAT
- MOTAT
- MAMA
- KATKA

(b) Dekódujte slová

- 0111011
- 0111110
- 11100110010
- 0110010011

■

Cvičenie 10.12 Zostrojte optimálny Huffmanov kód pre dané znaky a ich početnosti.

(a)

znak	početnosť
A	5
B	6
C	6
D	11
E	20

(b)

znak	početnosť
A	2
B	3
C	5
D	8
E	13
F	21

■

Cvičenie 10.13 Koľko bitov potrebujeme na zakódovanie slova BRATISLAVA, ak použijeme optimálny Huffmanov kód pre toto slovo? ■

Cvičenie 10.14 Určte pravdivosť nasledujúcich výrokov.

(a) V Huffmanovom kóde žiadne kódové slovo nemôže byť prefixom iného kódového slova.

(b) Huffmanov kód je bezstratový spôsob kódovania (bez straty informácie).

(c) V niektorých prípadoch nemusí mať optimálny Huffmanov kód minimálnu redundanciu.

■

Cvičenie 10.15 Máme abecedu s piatimi znakmi, ktorých relatívne početnosti sú dané tabuľkou

znak	A	B	C	D	E
relatívna početnosť	0.34	0.06	0.13	0.07	0.4

Zostrojte optimálny Huffmanov kód pre danú abecedu a vypočítajte priemerný počet bitov potrebných na zakódovanie 100 znakovej správy. ■

Cvičenie 10.16 Počas 1. svetovej vojny používala nemecká armáda šifru ADFGVX. Jej názov je odvodený od šiestich znakov, pomocou ktorých sa šifrovali všetky správy.

znak	početnosť
A	23
D	20
F	7
G	13
V	21
X	16

Koľko bitov by sa ušetrilo, ak by sa 100 znaková zašifrovaná správa pred odoslaním zakódovala pomocou optimálneho Huffmanovho kódu? Početnosť znakov v zašifrovanej správe udáva tabuľka vľavo a jeden znak sa kóduje jedným bytom, t. j. 8 bitmi.

■

Použitá a odporúčaná literatúra

[1] Martin Aigner: **Combinatorial Theory**, SPRINGER VERLAG, HEIDELBERG 1979.

[2] Bohuslav Balcar, Petr Štěpánek: **Teorie množin**, ACADEMIA, PRAHA 1986.

[3] Berežný, Draženská, Kravecová: **Zbierka úloh z diskrétnej matematiky**, FEI TU, KOŠICE 2005. `http://web.tuke.sk/fei-km/sites/default/files/prilohy/10/Zbierka-DM.pdf`

[4] J. A. Bondy, U. S. Murty: **Graph Theory With Applications**, NORTH-HOLLAND 1976.

[5] Reinhard Diestel: **Graph Theory**, SPRINGER 2000.

[6] Harary, Frank: **Graph theory**, ADDISON-WESLEY 1969.

[7] Richard Johnsonbaugh: **Discrete Mathematics**, PEARSON 2017.

[8] Josef Kaucký: **Kombinatorické identity**, VEDA, VYDAVATEĽSTVO SAV, BRATISLAVA 1975.

[9] Martin Knor: **Úvod do matematickej logiky**, FIIT STU, BRATISLAVA 2016. `http://www.math.sk/jmkollar/literatura/Knor-Uvod_do_matematickej_logiky.pdf`

[10] Martin Knor: **Teória grafov**, SvF STU, BRATISLAVA 2008. `http://www.svf.stuba.sk/docs/dokumenty/skripta/teoria_grafov_martin_knor.pdf`

[11] Martin Knor, Jozef Kollár: **Matematika pre architektov**, STU BRATISLAVA, 2002.

[12] Donald E. Knuth: **The Art of Computer Programming, Volume 1**, ADDISON-WESLEY, 1997.

[13] Sergei K. Lando: **Lectures on Generating Functions**, AMERICAN MATHEMATICAL SOCIETY, STUDENT MATHEMATICAL LIBRARY 2003.

[14] Milan Mareš: **Příběhy matematiky**, PISTORIUS & OLŠANSKÁ, PŘÍBRAM 2011.

[15] J. Matoušek, J. Nešetřil: **Kapitoly z diskrétní matematiky**, KAROLINUM, PRAHA 2002.

[16] М. В. Меньшиков, А. М. Ревякин, А. Н. Копылова, Ю. Н. Макаров, Б. С. Стечкин: **Комбинаторный Анализ – Задачи и упражнения**, Издательство »НАУКА«, Москва 1982.

[17] В. А. Носов: **Комбинаторика и теория графов**, Москва 1999. `http://intsys.msu.ru/staff/vnosov/combgraph.htm` (PDF verzia)

[18] Stanislav Palúch: **Algoritmická teória grafov**, Žilinská univerzita, 2008. `http://frcatel.fri.uniza.sk/users/paluch/grafy.pdf`

[19] Ján Plesník: **Grafové algoritmy**, VEDA, Bratislava 1983.

[20] Franco P. Preparata, Raymond T. Yeh: **Úvod do téorie diskrétnych matematických štruktúr**, Alfa, Bratislava 1982.

[21] Edward M. Reingold, Jurg Nievergelt, Narsingh Deo: **Combinatorial Algorithms – Theory and Practice**, Prentice-Hall, New Jersey 1977.

[22] Beloslav Riečan a kol.: **Úlohy z matematiky pre 4. ročník gymnázia**, SPN, Bratislava 1976.

[23] Beloslav Riečan a kol.: **Matematika pre 4. ročník gymnázia**, SPN, Bratislava 1987.

[24] Beloslav Riečan a kol.: **Zbierka úloh z matematiky pre 4. ročník gymnázia**, SPN, Bratislava 1991.

[25] Karol Rovan a kol.: **Zbierka riešených úloh z algebry pre SVŠ a odborné školy**, SPN, Bratislava 1969.

[26] К. А. Рыбников: **Введение в Комбинаторный Анализ**, Издательство Московского Университета, Москва 1985.

[27] Raymond M. Smullyan: **A Beginner's Guide to Mathematical Logic**, Dover Publications, New York 2014.

[28] František Vejsada, František Talafous: **Zbierka úloh z matematiky pre SVŠ a gymnázia**, SPN, Bratislava 1973.

[29] Naum J. Vilenkin: **Rozhovory o množinách**, SPN, Bratislava 1972.

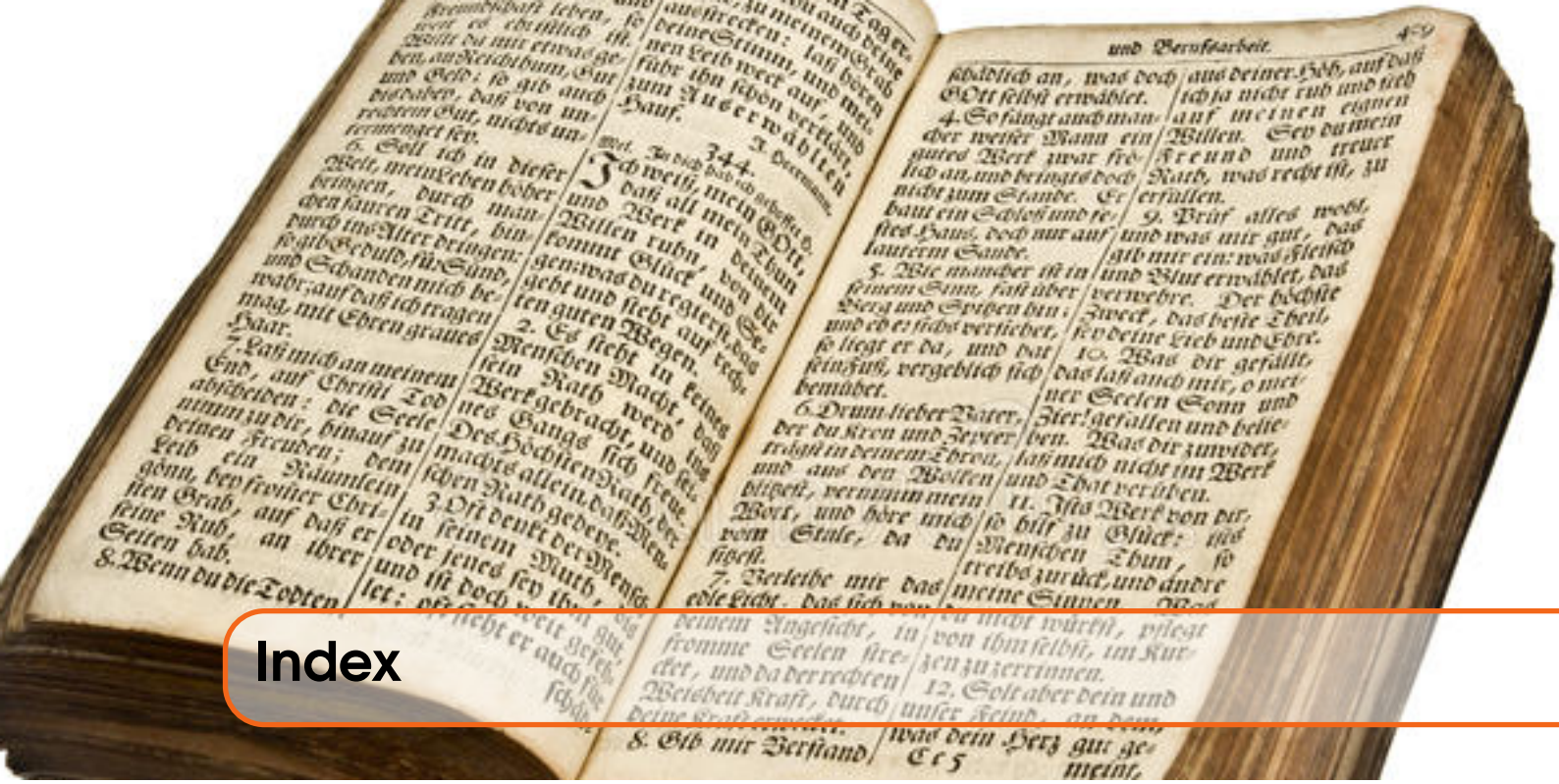
[30] Herbert S. Wilf: **generatingfunctionology**, Academic Press, Inc. 1994. `https://www.math.upenn.edu/~wilf/DownldGF.html`

[31] Niklaus Wirth: **Algoritmy a štruktúry údajov**, Alfa, Bratislava 1989.

[32] Štefan Znám: **Kombinatorika a teória grafov**, PF UK, Bratislava 1978.

IV

Index



Index

Symboly

A

B

C

Č

D

E

F

G

H

I

J

K

L

M

N

P

R

S

T

Ť

U

Ú

V

W

Z