Kapitola 7

Nejkratší cesty mezi všemi páry uzlů

Pro nalezení nejkratších cest mezi všemi dvojicemi uzlů jistě stačí opakovaně použít algoritmus hledání cest z jediného výchozího uzlu do všech ostatních. V této kapitole budeme kvůli stručnosti označovat počet uzlů $|U|$ grafu symbolem n. V případě grafu s nezáporným ohodnocením hran dostaneme n-násobným použitím Dijkstrova algoritmu (za předpokladu realizace prioritní fronty pomocí Fibonacciho haldy) postup s asymptotickou časovou složitostí $O(n^2 \lg n + n \cdot |H|)$. Použití binární haldy vede na složitost $O(|H| \cdot n \lg n)$ a při sekvenčním uložením fronty v poli získáme algoritmus složitosti $O(n^3)$.

Pokud graf obsahuje i záporně ohodnocené hrany, nelze Dijkstrova algoritmu použít, takže bychom museli opakovat hledání pomocí Bellmanova-Fordova algoritmu. V tomto případě bude časová složitost nalezení nejkratších cest mezi všemi dvojicemi uzlů rovna $O(n^2|H|)$, což pro husté grafy dává hodnotu $O(n^4)$. Naším cílem v této kapitole je ukázat, že lze postupovat efektivněji.

7.1 Nejkratší cesty a násobení matic

Výsledkem algoritmu hledání nejkratších cest mezi všemi dvojicemi uzlů je soubor $n \times n$ hodnot (w-vzdáleností popř. předchůdců na nejkratších cestách), pro které je přirozenou formou uložení čtvercová matice. Budeme tedy předpokládat, že i výchozí graf G je reprezentován maticí. Protože kromě struktury grafu musíme vyjádřit i ohodnocení jednotlivých hran, vytvoříme jedinou matici, která kombinuje matici sousednosti grafu s funkcí ohodnocení w. Z důvodů, které jsme vysvětlili již v předchozím odstavci, budeme předpokládat, že graf neobsahuje cykly záporné w-délky.

Definice 7.1: Nechť je dán jednoduchý orientovaný graf $G = \langle H, U \rangle$ s reálným ohodnocením hran $w : H \mapsto \mathbf{R}$. **Maticí w-délek** grafu G nazýváme čtvercovou matici $\mathbf{W} = [w_{ij}]$ řádu n, jejíž prvky jsou definovány vztahem

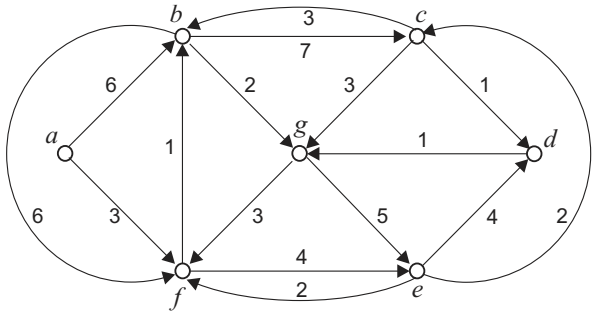
$$w_{ij} \;=\; \begin{cases} 0 & \text{pokud } i = j \\ w(u_i, u_j) & \text{pokud } i \neq j, (u_i, u_j) \in H \\ \infty & \text{pokud } i \neq j, (u_i, u_j) \notin H \end{cases} \tag{7.1}$$

Maticí w-vzdáleností grafu G nazýváme čtvercovou matici $\mathbf{D} = [d_{ij}]$ řádu n, jejíž prvky jsou definovány vztahem

$$d_{ij} \;=\; d_w(u_i, u_j). \tag{7.2}$$

Maticí předchůdců grafu G (zkráceně též **p-maticí**) nazýváme čtvercovou matici $\mathbf{P} = [p_{ij}]$ řádu n, jejíž prvky jsou definovány vztahem

$$p_{ij} \;=\; \begin{cases} 0 & \text{pokud } i = j \text{ nebo neexistuje cesta z } u_i \text{ do } u_j \\ u_k & \text{kde } u_k \text{ je předchůdce uzlu } u_j \text{ na nějaké minimální cestě z } u_i \text{ do } u_j \end{cases} \tag{7.3}$$



$W =$

	a	b	c	d	e	f	g
a	0	6	∞	∞	∞	3	∞
b	∞	0	7	∞	∞	6	2
c	∞	3	0	1	∞	∞	3
d	∞	∞	∞	0	∞	∞	1
e	∞	∞	2	4	0	2	∞
f	∞	1	∞	∞	4	0	∞
g	∞	∞	∞	∞	5	3	0

$D =$

	a	b	c	d	e	f	g
a	0	4	9	10	7	3	6
b	∞	0	7	8	7	5	2
c	∞	3	0	1	7	5	2
d	∞	5	8	0	6	4	1
e	∞	3	2	3	0	2	4
f	∞	1	6	7	4	0	3
g	∞	4	7	8	5	3	0

$P =$

	a	b	c	d	e	f	g
a	0	f	e	c	f	a	b
b	0	0	b	c	g	g	b
c	0	c	0	c	g	g	d
d	0	f	e	0	g	g	d
e	0	f	e	c	0	b	d
f	0	f	e	c	f	0	b
g	0	f	e	c	g	g	0

Obrázek 7.1: Matice w-délek, w-vzdáleností a předchůdců

Na obr. 7.1 ukazujeme příklad grafu a jemu odpovídajících matic w-délek, w-vzdáleností a předchůdců. Graf má v tomto případě vesměs kladná ohodnocení hran, která představují čísla uvedená u hran v obrázku grafu. Postupům, pomocí nichž se určí matice w-vzdáleností, se budeme věnovat později v tomto a v následujícím odstavci. Jejich rozšíření vedoucí k získání matice předchůdců ponecháme jako cvičení, ukážeme zde pouze význam a využití této matice.

Hodnoty $p_{ij}, j = 1, 2, \ldots, n$ ležící v i-tém řádku matice předchůdců $\mathbf{P}$ obsahují stejnou informaci jako (jednorozměrné) pole hodnot $p[u]$ po skončení algoritmu hledání cest z uzlu $s = u_i$ do všech uzlů grafu. Jim odpovídající podgraf je stromem nejkratších cest z uzlu u_i, takže výčet uzlů tvořících nejkratší cestu z u_i do u_j dostaneme následující úpravou algoritmu CESTA uvedeného v závěru odstavce 4.2.

Algoritmus 7.2 Určení cesty pomocí p-matice

CESTA-1(P,i, j**)**

```
1  if i = j                                          Cesta končí ...
2     then write(i)                                  výstupem počátečního uzlu.
3     else if p_ij = 0                               Uzel u_j nemá předchůdce?
4             then write('cesta z ',u_i,' do ',u_j, 'neexistuje')
5             else CESTA-1(P, i, p_ij)               Napřed cesta k předchůdci
6                  write(j)                          a pak koncový uzel u_j.
```

Cesty s omezeným počtem hran

Nejkratší cesty v grafu o n uzlech (a bez záporných cyklů) neobsahují více než $(n-1)$ hran. Zkusíme tedy postupně určovat w-délky nejkratších cest obsahujících zvyšující se maximální počet hran. Nechť pro dva různé uzly $u_i, u_j \in U$ je $P : u_i \longrightarrow u_j$ nějaká nejkratší cesta obsahující nejvýše m hran. Je-li uzel u_k předchůdcem uzlu u_j na této cestě, pak je také (viz lemma 6.2) dílčí cesta $P' : u_i \longrightarrow u_k$ nejkratší cestou z u_i do u_k obsahující nejvýše $(m-1)$ hran. Kromě toho platí $w(P) = w(P') + w_{kj}$.

Označme tedy jako $d_{ij}^{(m)}$ minimální w-délku cesty z u_i do u_j tvořené nejvýše m hranami. Pro $m = 0$ se jedná o cestu neobsahující žádnou hranu, která může spojovat jen každý uzel sám se sebou. Dostáváme tedy

$$d_{ij}^{(0)} = \begin{cases} 0 & \text{pokud } i = j \\ \infty & \text{pokud } i \neq j. \end{cases} \tag{7.4}$$

Pro $m \geq 1$ určíme $d_{ij}^{(m)}$ jako minimum z hodnoty $d_{ij}^{(m-1)}$ (t.j. w-délky nejkratší cesty z u_i do u_j tvořené nejvýše $m-1$ hranami) a w-délky nekratší cesty z u_i do u_j tvořené nejvýše m hranami. Takovou cestu určíme probírkou cest o nejvýše $m-1$ hranách z u_i ke všem předchůdcům u_k uzlu u_j a jejich prodloužením o hranu (u_k, u_j), přičemž se jejich w-délka zvětší o w_{kj}. Z této úvahy vyplývá rekurzivní pravidlo

$$d_{ij}^{(m)} = \min(d_{ij}^{(m-1)}, \min_{1 \leq k \leq n} (d_{ik}^{(m-1)} + w_{kj})) = \min_{1 \leq k \leq n} (d_{ik}^{(m-1)} + w_{kj}). \tag{7.5}$$

Výsledná úprava v předchozím vztahu plyne z toho, že $w_{jj} = 0$ pro všechna j.

Určíme-li nyní pomocí tohoto pravidla w-délky nejkratších cest tvořených nejvýše $n-1$ hranami, dostaneme skutečné w-vzdálenosti mezi jednotlivými uzly. Další zvyšování m již nemůže hodnoty $d_{ij}^{(m)}$ změnit, takže lze psát

$$d_{ij} = d_w(u_i, u_j) = d_{ij}^{(n-1)} = d_{ij}^{(n)} = d_{ij}^{(n+1)} = \cdots. \tag{7.6}$$

Pomocí zadané matice w-délek můžeme tedy postupně určit posloupnost matic $\mathbf{D}^{(1)}$, $\mathbf{D}^{(2)}$, …, $\mathbf{D}^{(n-1)}$, kde $\mathbf{D}^{(m)} = [d_{ij}^{(m)}]$ pro $m = 1, 2, \ldots, n-1$. Přitom platí $\mathbf{D}^{(1)} = \mathbf{W}$ a poslední matice $\mathbf{D}^{(n-1)} = \mathbf{D}$ je hledanou maticí w-vzdáleností. Přechod od předchozí matice $\mathbf{D}'$ k následující matici $\mathbf{D}''$ v této posloupnosti provedeme pomocí následující procedury:

Algoritmus 7.3 Prodloužení nejkratších cest o jednu hranu
PRODLUŽ-CESTY(D', W, D")

```
1   for i := 1 to n do                          Pro každý počáteční uzel u_i
3       for j := 1 to n                         a každý koncový uzel u_j
4           do x := ∞                           nastav odhad minima
5               for k := 1 to n                 a zkus nalézt nejkratší cestu
6                   do x := min(x, d'_ik + w_kj)  prodloužením cesty odpovídající
7               d''_ij := x                     výchozí matici D'.
```

Složitost této procedury je díky třem vnořeným cyklům a konstantní složitosti vnitřní operace určení minima rovna $\Theta(n^3)$. Její struktura je shodná jako u procedury násobení matic – stačí vzít počáteční hodnotu 0 (namísto ∞ na řádce 4), operaci sčítání (namísto minima) a násobení (namísto součtu) na řádce 6. Připomeňme, že právě pomocí násobení matic (viz věta 4.6) je možné určit počet různých spojení určité délky mezi všemi dvojicemi uzlů.

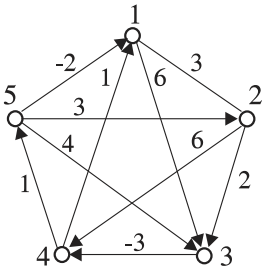
S ohledem na uvedenou podobnost označíme výsledek prodloužení cest s použitím multiplikativní notace jako $\mathbf{D}'' = \mathbf{D}' \bullet \mathbf{W}$, a můžeme pak pro posloupnost matic $\mathbf{D}^{(1)}$, $\mathbf{D}^{(2)}$, …, $\mathbf{D}^{(n-1)}$ psát

$$\mathbf{D}^{(1)} = \mathbf{D}^{(0)} \bullet \mathbf{W} = \mathbf{W}, \quad \mathbf{D}^{(2)} = \mathbf{D}^{(1)} \bullet \mathbf{W} = \mathbf{W}^2, \quad \cdots \quad \mathbf{D}^{(n-1)} = \mathbf{D}^{(n-2)} \bullet \mathbf{W} = \mathbf{W}^{n-1}$$

Jelikož jsme již dříve ukázali, že matice $\mathbf{D}^{(n-1)}$ je hledanou maticí nejkratších cest, můžeme její výpočet provést pomocí následující procedury s časovou složitostí $O(n^4)$:

Algoritmus 7.4 Určení w-délek všech nejkratších cest
NEJKRATŠÍ-CESTY(W)

```
1   D^(1) := W
2       for m := 2 to n − 1
3           do PRODLUŽ-CESTY(D^(m−1), W, D^(m))
4   return D^(n−1)
```



$W = D^{(1)}$

	1	2	3	4	5
1	0	3	6	∞	∞
2	∞	0	2	6	∞
3	∞	∞	0	-3	∞
4	1	∞	∞	0	1
5	-2	3	4	∞	0

$D^{(2)}$

0	3	5	3	∞
7	0	2	-1	7
-2	∞	0	-3	-2
-1	4	5	0	1
-2	1	4	1	0

$D^{(3)}$

0	3	5	2	4
0	0	2	-1	0
-4	1	0	-3	-2
-1	2	5	0	1
-2	1	3	1	0

$D^{(4)}$

0	3	5	2	3
-2	0	2	-1	0
-4	-1	0	-3	-2
-1	2	4	0	1
-2	1	3	0	0

Obrázek 7.2: Nejkratší cesty

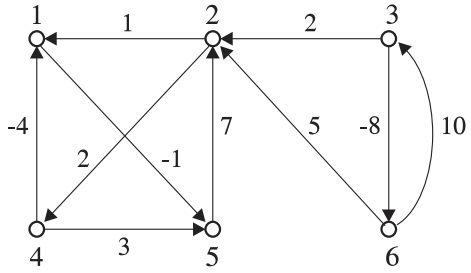
Použití uvedeného algoritmu ukazuje graf a posloupnost matic na obr. 7.2 dostaneme následující posloupnost matic:

Popsaný algoritmus je ovšem možné zjednodušit, neboť požadovaným výsledkem nejsou jednotlivé matice $\mathbf{D}^{(m)}$, ale až poslední matice $\mathbf{D}^{(n-1)}$. K jejímu výpočtu totiž stačí provést $\lceil \lg(n-1) \rceil$ součinů matic. Vezmeme-li v úvahu, že v grafech bez cyklů záporné w-délky platí $\mathbf{D}^{(m)} = \mathbf{D}^{(n-1)}$ pro všechna $m \geq n-1$, pak můžeme počítat jen následující matice

$$\begin{aligned}
\mathbf{D}^{(1)} &= \mathbf{W} \\
\mathbf{D}^{(2)} &= \mathbf{D}^{(1)} \bullet \mathbf{D}^{(1)} = \mathbf{W}^2 \\
\mathbf{D}^{(4)} &= \mathbf{D}^{(2)} \bullet \mathbf{D}^{(2)} = \mathbf{W}^4 \\
&\vdots \quad\quad \vdots \\
\mathbf{D}^{(2^k)} &= \mathbf{D}^{(2^{k-1})} \bullet \mathbf{D}^{(2^{k-1})} = \mathbf{W}^{2^k}
\end{aligned}$$

Poslední matice má pro $k = \lceil \lg(n-1) \rceil$ exponent $2^k \geq n-1$, takže je rovna hledané matici w-vzdáleností $\mathbf{D}$.

Provedení odpovídající úpravy procedury NEJKRATŠÍ-CESTY vedoucí na algoritmus s časovou složitostí $\Theta(n^3 \lg n)$ ponecháváme jako cvičení. Předpokládaný algoritmus je pochopitelně možné implementovat tak, že jeho paměťová složitost bude nižší než $O(n^2 \lg n)$, jak by vycházelo při uložení všech počítaných matic. Při výpočtu se vždy potřebuje pouze předchozí a následující matice v uvedené posloupnosti, takže vystačíme jen se dvěma maticemi řádu $n \times n$ (případně se třemi, chceme-li uchovat i matici $\mathbf{W}$). Paměťová složitost bude tedy jen $O(n^2)$.



Obrázek 7.3: Určení nejkratších cest

Cvičení

7.1-1. Simulujte provedení původního a zrychleného algoritmu NEJKRATŠÍ-CESTY na grafu z obr. 7.3.

7.1-2. Pokud je kromě w-délek nejkratších cest zapotřebí zjistit, kterými uzly tyto cesty procházejí, je možné zahrnout do uvedených algoritmů i výpočet matice předchůdců. Proveďte příslušnou úpravu.

7.1-3. Vedle možnosti počítat matici předchůdců průběžně existuje i možnost spočítat ji až po získání výsledné matice $\mathbf{D}$. Navrhněte algoritmus složitosti $O(n^3)$, který určí matici předchůdců $\mathbf{P}$ z matice w-vzdáleností $\mathbf{D}$.

7.1-4. Doplňte algoritmus NEJKRATŠÍ-CESTY tak, aby se při jeho použití detekovala existence cyklů záporné w-délky.

7.1-5. Navrhněte efektivní algoritmus, který určí cyklus se zápornou w-délkou tvořený nejmenším počtem hran.

7.2 Floydův-Warshallův algoritmus

Při výpočtu posloupnosti matic $\mathbf{D}^{(m)}$ použité k získání matice w-vzdáleností v předchozím odstavci jsme postupně zpřesňovali odhad w-délky nejkratších cest uvažováním stále delších (co do počtu hran) cest. Floydův-Warshallův algoritmus, jemuž se věnujeme v tomto odstavci, má rovněž charakter postupného zpřesňování, ale tentokrát budeme rozšiřovat množinu přípustných vnitřních uzlů nejkratších cest. **Vnitřním uzlem** cesty $P = \langle u_1, u_2, \ldots, u_k \rangle$ je každý její uzel s výjimkou krajních uzlů u_1 a u_k, tedy libovolný z uzlů $\{u_2, u_3, \ldots, u_{k-1}\}$. Připomeňme, že stále uvažujeme pouze grafy bez cyklů záporné w-délky.

Nechť $\mathbf{D}^{(k)} = [d_{ij}^{(k)}]$ je matice, jejíž každý prvek $d_{ij}^{(k)}$ vyjadřuje w-délku nejkratší cesty z uzlu u_i do uzlu u_j mající vnitřní uzly pouze z množiny $\{u_1, u_2, \ldots, u_k\}$. Potom bude zřejmě $d_{ij}^{(0)} = w_{ij}$ a $d_{ij}^{(n)} = d_{ij}$. Od libovolné matice $\mathbf{D}^{(k-1)}$ se dostaneme k matici $\mathbf{D}^{(k)}$ prostřednictvím této úvahy:

- Jestliže nejkratší cesta z u_i do u_j mající vnitřní uzly z množiny $\{u_1, u_2, \ldots, u_k\}$ neprochází uzlem u_k, potom se hodnota prvku $d_{ij}^{(k)}$ rovná hodnotě $d_{ij}^{(k-1)}$.
- Jestliže nejkratší cesta $P : u_i \longrightarrow u_j$ mající vnitřní uzly z množiny $\{u_1, u_2, \ldots, u_k\}$ prochází uzlem u_k, potom si ji můžeme představit rozdělenou do dílčích cest $P_1 : u_i \longrightarrow u_k$ a $P_2 : u_k \longrightarrow u_j$. Cesty P_1 a P_2 jsou pak nutně nejkratšími cestami mezi svými krajními uzly a jejich vnitřní uzly jsou vybrány pouze z množiny $\{u_1, u_2, \ldots, u_{k-1}\}$.

Je tedy možné formulovat následující rekurentní definici hodnot prvků matice $\mathbf{D}^{(k)}$:

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{pokud } k = 0 \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{pokud } k \geq 1. \end{cases} \tag{7.7}$$

Z této rekurentní definice již bezprostředně plyne formulace následujícího algoritmu:

Algoritmus 7.5 Floydův-Warshallův algoritmus určení nejkratších cest

```
FLOYD-WARSHALL(W)
1  D^(0) := W                                  Počáteční nastavení hodnotami w_ij
2  for k := 1 to n do                          Pro narůstající množinu
3      for i := 1 to n do                      vnitřních uzlů nejkratších cest
4          for j := 1 to n do                  spočti další matici D^(k).
5              d_ij^(k) := min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
6  return D^(n)
```

Určení asymptotické časové složitosti Floydova-Warshallova algoritmu je snadné: provedení minimalizace uvnitř vnořeného cyklu trvá konstantní čas a opakuje se n^3-krát. Algoritmus má tedy časovou složitost $O(n^3)$, jeho řídicí struktura je velmi jednoduchá a používají se v něm jen maticově strukturovaná data. Pokud se týče jeho paměťové složitosti, stejnou úvahou jako u algoritmu uvedeného v předchozím odstavci lze ukázat, že vystačí s používáním dvou čtvercových matic řádu n. Floydův-Warshallův algoritmus je však ještě méně náročný – lze provést s použitím pouze jediné matice (viz cvič. 7.2-2). Asymptotická paměťová složitost je ovšem stále $O(n^2)$.

Floydův-Warshallův algoritmus můžeme opět doplnit výpočtem (se složitostí $O(n^3)$) matice předchůdců $\mathbf{P}$ poté, co se dokončil výpočet matice w-vzdáleností. Je také možné počítat spolu

$\mathbf{D}^{(0)}$

0	3	6	∞	∞
∞	0	2	6	∞
∞	∞	0	-3	∞
1	∞	∞	0	1
-2	3	4	∞	0

$\mathbf{D}^{(1)}$

0	3	6	∞	∞
∞	0	2	6	∞
∞	∞	0	-3	∞
1	4	7	0	1
-2	1	4	∞	0

$\mathbf{D}^{(2)}$

0	3	5	9	∞
∞	0	2	6	∞
∞	∞	0	-3	∞
1	4	6	0	1
-2	1	3	7	0

$\mathbf{D}^{(3)}$

0	3	5	2	∞
∞	0	2	-1	∞
∞	∞	0	-3	∞
1	4	6	0	1
-2	1	3	0	0

$\mathbf{D}^{(4)}$

0	3	5	2	3
0	0	2	-1	0
-2	1	0	-3	-2
1	4	6	0	1
-2	1	3	0	0

$\mathbf{P}^{(0)}$

0	1	1	0	0
0	0	2	2	0
0	0	0	3	0
4	0	0	0	4
5	5	5	0	0

$\mathbf{P}^{(1)}$

0	1	1	0	0
0	0	2	2	0
0	0	0	3	0
4	1	1	0	4
5	1	5	0	0

$\mathbf{P}^{(2)}$

0	1	2	2	0
0	0	2	2	0
0	0	0	3	0
4	1	2	0	4
5	1	2	2	0

$\mathbf{P}^{(3)}$

0	1	2	3	0
0	0	2	3	0
0	0	0	3	0
4	1	2	0	4
5	1	2	3	0

$\mathbf{P}^{(4)}$

0	1	2	3	4
4	0	2	3	4
4	1	0	3	4
4	1	2	0	4
5	1	2	3	0

Obrázek 7.4: Matice vzdáleností a přechůdců na nejkratších cestách

s posloupností matic $\mathbf{D}^{(k)}$ průběžně i posloupnost matic $\mathbf{P}^{(k)}$, neboť pro ni platí následující rekurentní vztahy:

$$p_{ij}^{(0)} = \begin{cases} 0 & \text{pokud } i = j \text{ nebo } w_{ij} = \infty, \\ i & \text{v ostatních případech,} \end{cases} \tag{7.8}$$

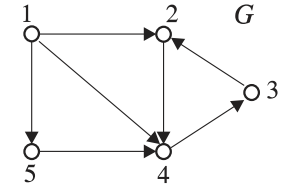
$$p_{ij}^{(k)} = \begin{cases} p_{ij}^{(k-1)} & \text{pokud } \ d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, \\ p_{kj}^{(k-1)} & \text{pokud } \ d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}. \end{cases} \tag{7.9}$$

Použitím Floydova-Warshallova algoritmu (doplněného o výpočet matice předchůdců) na graf z obr. 7.2 dostaneme posloupnost matic uvedenou na obr. 7.4.

Představme si nyní, že použijeme Floydova-Warshallova algoritmu pro jednoduchý orientovaný graf $G = \langle H, U \rangle$ bez ohodnocení hran. Jako výchozí matici místo $\mathbf{W}$ použijeme matici sousednosti $\mathbf{V}$ doplněnou jedničkami na hlavní diagonále. Na řádce 5 algoritmu provedeme místo minima operaci Booleovského součtu a místo číselného součtu operaci Booleovského součinu:

$$d_{ij}^{(k)} := d_{ij}^{(k-1)} \vee (d_{ik}^{(k-1)} \& d_{kj}^{(k-1)})$$

Takto získaná hodnota prvku $d_{ij}^{(k)}$ zjevně určuje, zda v grafu G existuje orientovaná cesta z uzlu u_i do uzlu u_j mající vnitřní uzly z množiny $\{u_1, u_2, \ldots, u_k\}$. Touto modifikací Floydova-Warshallova algoritmu je tedy možné určit matici sousednosti reflexivně-tranzitivního uzávěru grafu G.



$\mathbf{D}^{(0)} =$

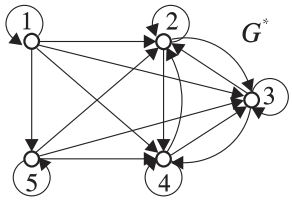
1	1	0	1	1
0	1	0	1	0
0	1	1	0	0
0	0	1	1	0
0	0	0	1	1

$\mathbf{D}^{(1)} =$

1	1	0	1	1
0	1	0	1	0
0	1	1	0	0
0	0	1	1	0
0	0	0	1	1

$\mathbf{D}^{(2)} =$

1	1	0	1	1
0	1	0	1	0
0	1	1	1	0
0	0	1	1	0
0	0	0	1	1



$\mathbf{D}^{(3)} =$

1	1	0	1	1
0	1	0	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	1	1

$\mathbf{D}^{(4)} =$

1	1	1	1	1
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	1

$\mathbf{D}^{(5)} =$

1	1	1	1	1
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	1

Obrázek 7.5: Výpočet reflexivně-tranzitivního uzávěru

Výpočet matice sousednosti reflexivně-tranzitivního uzávěru grafu G na obr. 7.5 uvedeným postupem vede na posloupnost matic, která je spolu s výsledným uzávěrem G^* znázorněna rovněž na obr. 7.5.

V odstavci 7.4 ukážeme, že existuje ještě řada dalších modifikací tohoto algoritmu lišících se pouze výběrem dvojoperace prováděné při výpočtu hodnoty $d_{ij}^{(k)}$, jež poskytují řešení jiným příbuzným typům úloh na grafech.

Cvičení

7.2-1. Doplňte Floydův-Warshallův algoritmus o výpočet matice předchůdců a simulujte jeho provedení pro graf z obr. 7.3. Porovnejte získané matice $\mathbf{D}^{(k)}$ a $\mathbf{P}^{(k)}$ s maticemi získanými řešením cvičení 7.1-1.

7.2-2. Ukažte, že na výsledku Floydova-Warshallova algoritmu se nic nezmění, pokud se vypustí horní indexy u všech použitých prvků d_{ij}, t.j. pokud se algoritmus upraví do následující podoby:

```
FLOYD-WARSHALL-1(W)
1  D := W                                  Počáteční nastavení hodnotami w_ij.
2  for k := 1 to n do                      Pro narůstající množinu
3      for i := 1 to n do                  vnitřních uzlů u_k nejkratších cest
4          for j := 1 to n do              spočti nové prvky matice D.
5              d_ij := min(d_ij, d_ik + d_kj)
6  return D
```

(*Návod:* Zvažte, zda se při k-tém průchodu vnějším cyklem mohou změnit hodnoty prvků d_{ik} a d_{kj}, na kterých závisí výpočet ostatních prvků matice v tomto průchodu.)

7.2-3. Předpokládejme, že jsme vztah (7.9) pro výpočet matice předchůdců změnili na následující:

$$p_{ij}^{(k)} = \begin{cases} p_{ij}^{(k-1)} & \text{pokud } d_{ij}^{(k-1)} < d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, \\ p_{kj}^{(k-1)} & \text{pokud } d_{ij}^{(k-1)} \geq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}. \end{cases} \tag{7.10}$$

Bude takto vypočtená matice předchůdců správná?

7.2-4. Doplňte Floydův-Warshallův algoritmus o zjištění výskytu cyklů záporné w-délky.

7.2-5. Navrhněte algoritmus výpočtu reflexivně-tranzitivního uzávěru orientovaného grafu $G = \langle H, U \rangle$ s asymptotickou časovou složitostí $O(|H| \cdot |U|)$.

7.3 Johnsonův algoritmus

Přes nespornou jednoduchost a snadnou implementovatelnost Floydova-Warshallova algoritmu je možné výpočet nejkratších cest realizovat ještě úsporněji, pokud budeme uvažovat **řídké grafy**. V řídkých grafech neroste počet hran se čtvercem počtu uzlů, takže asymptotická složitost $O(|H| \cdot n)$ je pak výrazně lepší než $O(n^3)$. V tomto odstavci ukážeme algoritmus navržený Johnsonem (viz [14]), jehož asymptotická složitost pro řídké grafy je $O(n^2 \lg n + n|H|)$. Výsledkem tohoto algoritmu je matice w-vzdáleností nebo signalizace existence cyklů záporné w-délky.

Základní myšlenkou Johnsonova algoritmu je n-násobné použití Dijkstrova algoritmu. O této možnosti jsme se zmínili již na začátku kapitoly s tím, že je omezena jen na grafy s nezáporným ohodnocením hran. Aby se tedy mohl Dijkstrův algoritmus v obecném případě použít, je třeba nejprve upravit původní ohodnocení w hran na nové ohodnocení $\widehat{w}$ tak, aby splňovalo následující podmínky:

- Pro každou dvojici uzlů $u, v \in U$ je nejkratší cesta pro ohodnocení w shodná s nejkratší cestou pro ohodnocení $\widehat{w}$.
- Pro každou hranu $(u, v) \in H$ je ohodnocení $\widehat{w}(u, v)$ nezáporné.

Proces získání nového ohodnocení $\widehat{w}$ budeme nazývat **přehodnocením hran** grafu G a nejprve ukážeme, že lze provést v čase $O(n|H|)$. V následujícím lemmatu zavedeme obecný způsob přehodnocení hran, který zaručuje splnění první z výše uvedených podmínek. Pro zjednodušení zápisu vyjadřujeme symbolem $d(u, v)$ vzdálenost uzlů u a v platnou při ohodnocení hran w a symbolem $\widehat{d}(u, v)$ vzdálenost při ohodnocení $\widehat{w}$.

Lemma 7.6: Nechť je dán jednoduchý orientovaný graf $G = \langle H, U\rangle$ s ohodnocením hran $w : H \mapsto \mathbf{R}$ a mějme libovolnou funkci $h : U \mapsto \mathbf{R}$, která přiřazuje každému uzlu grafu G nějaké reálné číslo. Pro každou hranu $(u, v) \in H$ nyní definujme

$$\widehat{w}(u, v) = w(u, v) + h(u) - h(v) \tag{7.11}$$

Potom pro libovolnou orientovanou cestu $L = \langle u_0, u_1, \ldots, u_k\rangle$ z uzlu u_0 do uzlu u_k bude platit $w(L) = d(u_0, u_k)$ právě tehdy, když platí $\widehat{w}(L) = \widehat{d}(u_0, u_k)$. Dále platí, že graf G obsahuje cyklus záporné w-délky, právě když obsahuje cyklus se zápornou $\widehat{w}$-délkou.

Důkaz: Dokážeme nejprve, že pro cestu L platí

$$\widehat{w}(L) = w(L) + h(u_0) - h(u_k). \tag{7.12}$$

Podle definice w-délky cesty a zavedení ohodnocení $\widehat{w}$ dostáváme

$$\begin{aligned}\widehat{w}(L) &= \sum_{i=1}^{k} \widehat{w}(u_{i-1}, u_i) = \sum_{i=1}^{k} (w(u_{i-1}, u_i) + h(u_{i-1}) - h(u_i)) = \\ &= \sum_{i=1}^{k} w(u_{i-1}, u_i) + h(u_0) - h(u_k) = w(L) + h(u_0) - h(u_k).\end{aligned}$$

Předpokládejme nyní, že L je nejkratší cesta z u_0 do u_k při ohodnocení hran w, tedy $w(L) = d(u_0, u_k)$. Dokážeme sporem, že pak platí i $\widehat{w}(L) = \widehat{d}(u_0, u_k)$, neboli L je nejkratší i při ohodnocení $\widehat{w}$. Nechť tedy pro ohodnocení $\widehat{w}$ existuje kratší cesta L' z uzlu u_0 do u_k, t.j. $\widehat{w}(L') < \widehat{w}(L)$. S použitím vztahu (7.12) pak dostáváme

$$w(L') + h(u_0) - h(u_k) = \widehat{w}(L') < \widehat{w}(L) = w(L) + h(u_0) - h(u_k).$$

To ale znamená, že cesta L' má menší délku než cesta L i při ohodnocení w, což je spor s volbou cesty L. Důkaz zpětné implikace se provede obdobně.

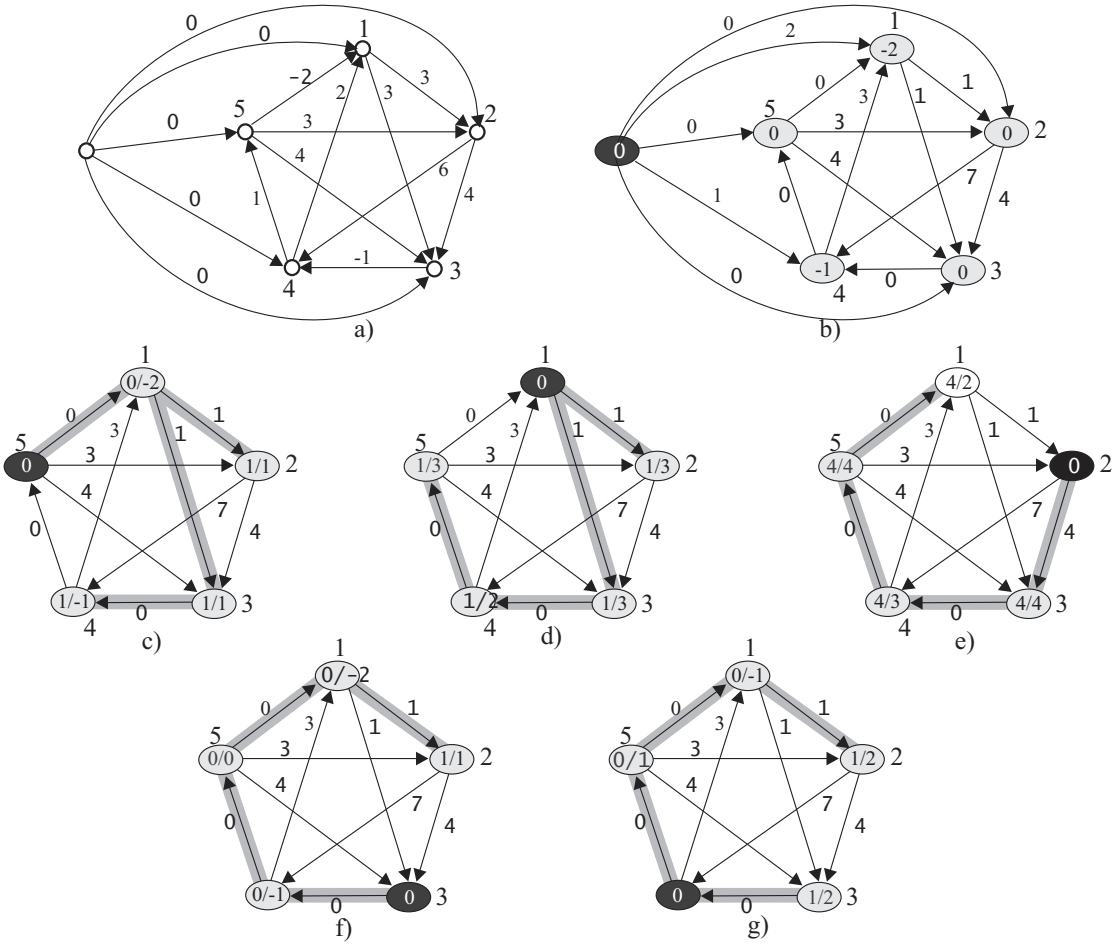
Zbývá dokázat tvrzení týkající se existence cyklů záporné délky. Mějme tedy nějaký cyklus $C = \langle u_0, u_1, \ldots, u_k\rangle$, $u_0 = u_k$. Podle vztahu (7.12) pak dostaneme

$$\widehat{w}(C) = w(C) + h(u_0) - h(u_k) = w(C),$$

takže C může mít zápornou délku při ohodnocení w právě tehdy, má-li zápornou délku při ohodnocení $\widehat{w}$. △

Nyní se zaměříme na to, jak zvolit funkci h použitou při přehodnocení hran způsobem zavedeným v předchozím lemmatu, abychom dosáhli splnění druhé z výše uvedených podmínek – nezápornosti ohodnocení hran $\widehat{w}$.

Zadaný graf $G = \langle H, U\rangle$ s ohodnocením hran $w : H \mapsto \mathbf{R}$ rozšíříme na graf $G' = \langle H', U'\rangle$ takto: přidáme nový uzel s, z něhož vedeme orientované hrany do všech uzlů grafu G. Platí tedy $U' = U \cup \{s\}, H' = H \cup \{(s, u) : u \in U\}$. Všem nově přidaným hranám určíme ohodnocení w rovné nule.



Obrázek 7.6: Provádění Johnsonova algoritmu

V takto vytvořeném grafu nemůže uzel s ležet na žádné orientované cestě spojující dvojici uzlů původního grafu G. Uzel s bude tedy pouze počátečním uzlem nejkratších cest vycházejících z něj do všech uzlů původního grafu. Graf G' obsahuje navíc cykly se zápornou w-délkou, právě když takové cykly obsahoval původní graf G. Na obr. 7.6a) ukazujeme graf G' odpovídající grafu G na obr. 7.2.

Předpokládejme nyní, že graf G (a tedy ani G') nemá cykly se zápornou w-délkou. Protože jsou z uzlu s dostupné všechny ostatní uzly, jsou v grafu G' korektně definovány vzdálenosti $d(s,u)$ a jsou konečné. Můžeme tedy položit $h(u) = d(s,u)$ pro všechna $u \in U'$. Podle lemmatu 6.3 platí $h(v) \leq h(u) + w(u,v)$ pro libovolnou hranu $(u,v) \in H'$, takže definujeme-li ohodnocení $\widehat{w}$ podle vztahu (7.11), bude platit $\widehat{w}(u,v) = w(u,v) + h(u) - h(v) \geq 0$.

Touto volbou funkce h je tedy zaručeno splnění druhé z podmínek uvedených pro přehodnocení hran. Na obr. 7.6b) ukazujeme graf G' z obr. 7.6a) po provedení výpočtu hodnot funkce $h(u)$ pro jednotlivé uzly a po využití této funkce při přehodnocení hran na nezáporné hodnoty $\widehat{w}(u,v)$.

Nyní je již možné vyjádřit strukturu Johnsonova algoritmu. Jako podprogramy se v něm používají Bellmanův-Fordův algoritmus (viz odst. 6.3) a Dijkstrův algoritmus (viz odst. 6.2). S ohledem na tyto algoritmy předpokládáme, že výchozí graf G je zadán spojovou reprezentací pomocí seznamů následníků. Výsledkem algoritmu jsou w-vzdálenosti uložené do matice $\mathbf{D}$ nebo signalizace výskytu cyklů se zápornou w-délkou. Pro zjednodušení zápisu algoritmu předpokládáme, že každý uzel je identifikován přirozeným číslem z intervalu $\langle 1, n \rangle$.

Algoritmus 7.7 Nejkratší cesty v řídkém grafu

JOHNSON(G)
1 Rozšíření grafu G na graf G' — Přidání uzlu s a hran (s, u).
2 **if** not BELLMAN-FORD(G', w, s) — Našly se v G' cykly
3 **then** write „graf obsahuje cyklus záporné délky“ — záporné w-délky?
4 **else** **for** každý uzel $u \in U'$
5 **do** $h(u) := d(s, u)$ — S výsledky B-F algoritmu
6 **for** každou hranu $(u, v) \in H'$ — proveď přehodnocení hran.
7 **do** $\widehat{w}(u, v) := w(u, v) + h(u) - h(v)$
8 **for** každý uzel $u \in U$ — Pro každý uzel původního grafu
9 **do** DIJKSTRA($G, \widehat{w}, u$) — spočti $\widehat{w}$-vzdálenosti $\widehat{d}(u, v)$
10 **for** každý uzel $v \in U$ — a uprav je na w-vzdálenosti.
11 **do** $d_{u,v} := \widehat{d}(u, v) + h(v) - h(u)$
12 **return D**

Grafy na obr. 7.6(c) až (g) ukazují výsledky opakovaného provádění Dijkstrova algoritmu na řádcích 8 až 11. Výchozí uzel u je vyznačen tmavě, hrany tvořící strom minimálních cest jsou zakresleny tučně. Čísla uvedená vedle uzlů jsou hodnoty funkce h, kromě toho je v každém uzlu v uvedena dvojice hodnot x/y, kde $x = \widehat{d}(u, v), y = d(u, v) = \widehat{d}(u, v) + h(v) - h(u)$.

Pro určení asymptotické časové složitosti Johnsonova algoritmu je rozhodující n-krát opakované provedení Dijkstrova algoritmu, neboť to spotřebuje čas $O(n^2 \lg n + n \cdot |H|)$, kdežto jedno provedení Bellmanova-Fordova algoritmu potřebuje jen $O(n \cdot |H|)$. Tento odhad ovšem předpokládá, že pro implementaci prioritní fronty v Dijkstrově algoritmu se použilo Fibonacciho haldy. Pokud použijeme jen obyčejnou binární haldu, dostáváme výslednou složitost Johnsonova algoritmu $O(n \cdot |H| \lg n)$, což je stále pro řídké grafy asymptoticky rychlejší než Floydův-Warshallův algoritmus se složitostí $O(n^3)$. Pro paměťovou složitost jsou rozhodující nároky na uložení matice vzdáleností $\mathbf{D}$, které činí $O(n^2)$.

Cvičení

7.3-1. Pomocí Johnsonova algoritmu určete nejkratší cesty mezi všemi dvojicemi uzlů v grafu na obr. 7.3. Hodnoty h a $\widehat{w}$ znázorněte podobným způsobem, jaký je použitý na obr. 7.6

7.3-2. Co vyjadřuje hodnota $h(u)$ spočítaná pro každý uzel grafu G' během přípravy přehodnocení hran?
(*Návod:* Uvažujte nejmenší w-délky cest grafu G končících v uzlu u.)

7.3-3. Nechť je ohodnocení všech hran grafu G nezáporné. Jaký bude potom vztah mezi ohodnocením w a ohodnocením $\widehat{w}$ získaným po přehodnocení podle Johnsonova algoritmu?

7.4 Algebraické souvislosti úloh o spojeních

Už v odstavci 7.2 jsme upozornili na to, že kromě výpočtu vzdáleností mezi všemi uzly je možné použít Floydova-Warshallova algoritmu také k určení reflexivně-tranzitivního uzávěru grafu. Po vhodné úpravě centrální dvojoperace může tento algoritmus poskytnout řešení i dalších grafových úloh, v nichž je třeba určit nějaké hodnoty závislé na orientovaných spojeních v grafu, přičemž hodnota příslušná každému spojení se odvozuje od ohodnocení jeho hran.

Připomeňme nejprve, jaký je význam jednotlivých operací ve zmiňované dvojoperaci uvedené v algoritmu ve tvaru

$$d_{ij}^{(k)} := \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$$

Součet $d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$ se týká případu, kdy spojení z u_i do u_j vznikne složením dvou spojení $u_i \longrightarrow u_k$ a $u_k \longrightarrow u_j$. Znamená to tedy, že ohodnocení odpovídající složení dvou orientovaných spojení je dáno součtem dílčích ohodnocení. Dovedeno na úroveň jednotlivých hran toto

pravidlo stanoví, že ohodnocení každého spojení se získá jako součet ohodnocení jej tvořících hran. Druhá operace, kterou je zde je výběr minima, určuje, jakým způsobem kombinujeme alternativní spojení mezi stejnými uzly: vybíráme „kratší“ z nich.

Abychom mohli obecně formulovat potřebné vlastnosti obou složek centrální dvojoperace, zavedeme pro ně zvláštní symboly:

$\odot$ vyjadřuje efekt skládání dvou spojení,
$\oplus$ vyjadřuje způsob kombinování alternativ.

Příslušné vlastnosti těchto operací jsou zahrnuty v definici algebraických struktur, které se nazývají polookruh, případně uzavřený polookruh.

Definice 7.8: Nechť je dána neprázdná množina P, na níž jsou definovány dvě binární operace $\oplus$ (sčítání) $\odot$ (násobení) a dvě nulární operace $\mathbf{0}$ (nulový prvek) a $\mathbf{1}$ (jednotkový prvek). Nechť tyto operace splňují pro všechna $a, b, c \in P$ následující podmínky:

1. $a \oplus (b \oplus c) = (a \oplus b) \oplus c$ (asociativnost sčítání)
2. $a \oplus \mathbf{0} = \mathbf{0} \oplus a = a$ (jednotka pro sčítání)
3. $a \odot (b \odot c) = (a \odot b) \odot c$ (asociativnost násobení)
4. $a \odot \mathbf{1} = \mathbf{1} \odot a = a$ (jednotka pro násobení)
5. $a \odot \mathbf{0} = \mathbf{0} \odot a = \mathbf{0}$ (nula pro násobení)
6. $a \oplus b = b \oplus a$ (komutativnost sčítání)
7. $a \oplus a = a$ (idempotence sčítání)
8. $a \odot (b \oplus c) = (a \odot b) \oplus (a \odot c)$ (distributivnost násobení zleva vůči sčítání)
9. $(a \oplus b) \odot c = (a \odot c) \oplus (b \odot c)$ (distributivnost násobení zprava vůči sčítání)

Systém $\mathcal{P} = \langle P, \oplus, \odot, \mathbf{0}, \mathbf{1} \rangle$ pak nazýváme **polookruhem**. Splňují-li operace polookruhu navíc ještě následující podmínky:

10. pro libovolnou posloupnost prvků $a_1, a_2, a_3, \ldots$ prvků z P je definován součet $a_1 \oplus a_2 \oplus a_3 \oplus \cdots$
11. asociativnost, komutativnost a idempotence sčítání platí i pro nekonečnou posloupnost sčítanců
12. distributivnost násobení platí i pro nekonečnou posloupnost sčítanců, t.j.
$a \odot (b_1 \oplus b_2 \oplus b_3 \oplus \cdots) = (a \odot b_1) \oplus (a \odot b_2) \oplus (a \odot b_3) \oplus \cdots$
$(a_1 \oplus a_2 \oplus a_3 \oplus \cdots) \odot b = (a_1 \odot b) \oplus (a_2 \odot b) \oplus (a_3 \odot b) \oplus \cdots$

potom nazýváme polookruh $\mathcal{P}$ **uzavřeným polookruhem**.

Podmínky 1, 2 a 6 v definici znamenají, že struktura $\langle P, \oplus, \mathbf{0} \rangle$ je komutativní monoid, podmínky 3 a 4 charakterizují jako monoid také strukturu $\langle P, \odot, \mathbf{1} \rangle$. Množina P nemusí být nutně číselná, takže ani operace sčítání a násobení polookruhu pak nemají číselnou povahu. Polookruhem je např. i Booleovská struktura $\mathcal{B} = \langle \{\mathbf{0}, \mathbf{1}\}, \vee, \wedge, \mathbf{0}, \mathbf{1} \rangle$, kterou jsme použili pro výpočet reflexivně-tranzitivního uzávěru grafu.

Operace součinu $\odot$ polookruhu odpovídá skládání orientovaných spojení nebo jednotlivých hran do posloupností tvořících spojení – prvek $\mathbf{1}$ vyjadřuje triviální spojení (tvořené jediným uzlem) a $\mathbf{0}$ vyjadřuje neexistenci spojení. Z tohoto pohledu se jeví přirozené požadovat splnění podmínek 3 (asociativnost skládání), 4 (neutrálnost triviálního spojení) i 5 (složení zahrnující neexistující spojení dává opět neexistující spojení).

Operace součtu $\oplus$ polookruhu odpovídá „sumarizaci“ účinku alternativních orientovaných spojení. Zde je přirozené požadovat podmínkou 2 neutralitu prvku $\mathbf{0}$ (neexistuje spojení), podmínka 1 stanoví asociativnost a podmínka 6 nezávislost na pořadí u alternativních spojení. Požadavek idempotence (podmínka 7) je pojistkou pro to, aby případné vícenásobné použití téhož spojení v alternativě neznamenalo žádný rozdíl oproti jedinému použití.

Velmi důležitý je požadavek distributivnosti – stanoví vzájemný vztah mezi skládáním spojení a sumarizací alternativ. Distributivnost tedy umožňuje vyjádřit, jaký efekt má prodloužení dvou alternativ o stejný přidaný úsek.

Uzavřenost polookruhu zajišťuje korektnost operace sumarizace i pro nekonečné (spočetné) množiny spojení, díky komutativitě, asociativitě a idempotenci pak připouští přerovnat a zjednodušit libovolný nekonečný součet $(a_1 \oplus a_2 \oplus a_3 \oplus \cdots)$ tak, aby se v něm každý člen vyskytoval pouze jednou. Víme, že nekonečný počet (různých) spojení v orientovaném grafu může způsobit pouze existence cyklů. Označíme-li hodnotu odpovídající nějakému cyklu jako c, potom lze v libovolném uzlu tohoto cyklu vytvořit nekonečnou posloupnost spojení vzniklých žádným ($\mathbf{1}$), jedním (c), dvěma ($c \odot c$), atd. průchody daným cyklem. Sumarizací všech těchto možností dostaneme tzv. **uzávěr** definovaný vztahem

$$c^* = \mathbf{1} \oplus c \oplus (c \odot c) \oplus (c \odot c \odot c) \oplus (c \odot c \odot c \odot c) \oplus \cdots \tag{7.13}$$

Z vlastností uzavřeného polookruhu bezprostředně plynou pro operaci uzávěru následující jednoduché vztahy:

$$\mathbf{0}^* = \mathbf{1}, \quad c \odot c^* = c^* \odot c, \quad c^* = \mathbf{1} \oplus (c \odot c^*).$$

Obecná formulace úlohy o spojeních a její řešení

Předpokládejme, že je dán jednoduchý orientovaný graf $G = \langle H, U \rangle$ a nějaké zobrazení $\omega : H \mapsto P$ přiřazující hranám grafu G nenulové prvky polookruhu $\mathcal{P} = \langle P, \oplus, \odot, \mathbf{0}, \mathbf{1} \rangle$. Aby bylo zobrazení ω definováno pro všechny dvojice uzlů, doplníme je hodnotami $\omega(u, v) = \mathbf{0}$ pro takové uspořádané dvojice (u, v), které nepředstavují hranu grafu G.

Zobrazení ω rozšíříme nyní na spojení grafu G tak, že pro každé spojení $L = \langle u_1, u_2, \ldots, u_k \rangle$ definujeme

$$\omega(L) = \omega(u_1, u_2) \odot \omega(u_2, u_3) \odot \cdots \odot \omega(u_{k-1}, u_k).$$

Prázdnému spojení je tímto vztahem v souladu s běžnou zvyklostí přiřazen jednotkový prvek polookruhu $\mathcal{P}$ vůči operaci násobení – tedy $\mathbf{1}$.

Pro složení $L_1 \circ L_2$ dvou spojení $L_1 = \langle u_1, u_2, \ldots, u_k \rangle, L_2 = \langle u_k, u_{k+1}, \ldots, u_l \rangle$ dostáváme díky asociativitě operace součinu polookruhu

$$\begin{aligned} \omega(L_1 \circ L_2) &= \omega(\langle u_1, u_2, \ldots, u_k, \ldots, u_{l-1}, u_l \rangle) \\ &= \bigodot_{i=1}^{l-1} \omega(u_i, u_{i+1}) = (\bigodot_{i=1}^{k-1} \omega(u_i, u_{i+1})) \odot (\bigodot_{i=k}^{l-1} \omega(u_i, u_{i+1})) \\ &= \omega(L_1) \odot \omega(L_2). \end{aligned}$$

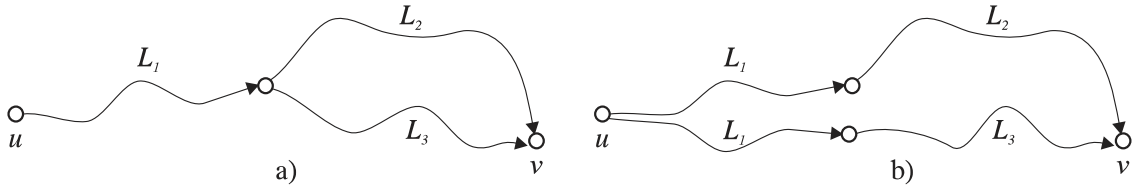
Vzhledem k tomu, že složením libovolné (konečné) posloupnosti na sebe navazujících spojení $L_1, L_2, \ldots, L_r$ vznikne opět spojení, bude platit obecně

$$\omega(L_1 \circ L_2 \circ \cdots \circ L_r) = \omega(L_1) \odot \omega(L_2) \odot \cdots \odot \omega(L_r).$$

Zobrazení ω rozšíříme rovněž na libovolné nejvýše spočetné množiny $\{L_i\}_{i \in I}, I \subseteq \mathbf{N}$ spojení grafu G, která všechna vycházejí ze stejného uzlu a stejným uzlem také všechna končí. Pro takové množiny definujeme

$$\omega(\{L_i\}_{i \in I}) = \bigoplus_{i \in I} \omega(L_i).$$

Operace součtu polookruhu se tedy rezervuje pro vyjádření hodnoty zobrazení ω libovolné množiny alternativních („paralelních“) spojení grafu propojujících stejnou dvojici uzlů. Distributivita násobení vůči sčítání přitom zaručuje konzistentní chování sumarizace alternativ při skládání navazujících spojení. Na obr. 7.7 ukazujeme dva grafy, které jsou rovnocenné z hlediska



Obrázek 7.7:

ohodnocení spojení z uzlu u do uzlu v – části a) odpovídá ohodnocení $\omega(L_1) \odot (\omega(L_2) \oplus \omega(L_3))$, části b) pak díky distributivnímu zákonu stejná hodnota $(\omega(L_1) \odot \omega(L_2)) \oplus (\omega(L_1) \odot \omega(L_3))$.

Zobecněním Floydova-Warshallova algoritmu získáme postup, pomocí něhož lze pro všechny dvojice uzlů $u, v \in U$ určit hodnotu

$$s_{uv} = \bigoplus_{L:u \longrightarrow v} \omega(L),$$

kde v součtu na pravé straně se uvažují všechna spojení grafu G vedoucí z uzlu u do uzlu v. Protože každé spojení má konečný počet hran, je množina všech těchto spojení nejvýše spočetná, a výsledek součtu je tedy (v uzavřeném polookruhu) vždy definován.

Předpokládejme opět pro jednoduchost, že uzly grafu G jsou označeny přímo čísly $1, 2, \ldots, |U|$ a podobně jako v odvození původního Floydova-Warshallova algoritmu uvažujme množinu všech spojení z uzlu i do uzlu j, která mají vnitřní uzly pouze z množiny $\{1, 2, \ldots, k\}$ – označme tuto množinu spojení symbolem $L(i, j, k)$.

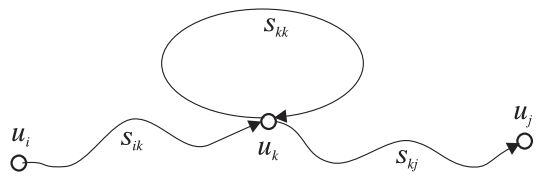
Označme nyní výsledek sumarizace ohodnocení ω pro spojení obsažená v této množině:

$$s_{ij}^{(k)} = \bigoplus_{L \in L(i,j,k)} \omega(L).$$

Pro hodnoty $s_{ij}^{(k)}$ bude platit rekurentní vztah

$$s_{ij}^{(k)} = s_{ij}^{(k-1)} \oplus (s_{ik}^{(k-1)} \odot (s_{kk}^{(k-1)})^* \odot s_{kj}^{(k-1)}). \tag{7.14}$$

Tento rekurentní vztah se velmi podobá rekurenci (7.7) s tím, že navíc obsahuje středový faktor $(s_{kk}^{(k-1)})^*$. Zahrnutí tohoto faktoru však vyjadřuje skutečnost, že při propojení dvou spojení v uzlu k je třeba uvažovat uzávěr všech uzavřených spojení procházejících uzlem k a majících vnitřní uzly jen z množiny $\{1, 2, \ldots, k-1\}$ – viz obr. 7.8.



Obrázek 7.8: Skládání spojení

Zbývá tedy definovat počáteční hodnoty, od nichž odstartuje rekurence podle vztahu (7.14), k čemuž použijeme hodnot

$$s_{ij}^{(0)} = \begin{cases} \omega(i,j) & \text{pro } i \neq j, \\ \mathbf{1} \oplus \omega(i,j) & \text{pro } i = j. \end{cases} \tag{7.15}$$

Zdůvodnění diagonálních hodnot vyplývá ze skutečnosti, že pro spojení z každého uzlu do sebe sama musíme uvažovat i triviální alternativu – prázdné spojení.

Nyní již můžeme formulovat zobecněný Floydův-Warshallův algoritmus pro výpočet sumarizujících ohodnocení pro všechna alternativní spojení mezi všemi dvojicemi uzlů.

Algoritmus 7.9 Zobecněný Floydův-Warshallův algoritmus

FLOYD-WARSHALL-G(G, ω)

```
1   for i := 1 to n do                          Nastavení
2       for j := 1 to n                         počátečních hodnot
```

```
3          do s_ij^(0) := ω(i, j)                    se zvláštní péčí
4       s_ii^(0) := 1 ⊕ ω(i, i)                      diagonále.
5    for k := 1 to n do                              Pro narůstající množinu
6       for i := 1 to n do                           vnitřních uzlů
7          for j := 1 to n do                        se sumarizují alternativy.
8             s_ij^(k) := s_ij^(k−1) ⊕ (s_ik^(k−1) ⊙ (s_kk^(k−1))* ⊙ s_kj^(k−1))
9    return S^(n)
```

Časová složitost tohoto algoritmu závisí na tom, jak dlouho trvá výpočet součtu, součinu a uzávěru v polookruhu. Pokud tyto časy označíme po řadě jako $T_\oplus, T_\odot$ a T_*, můžeme asymptotickou časovou složitost algoritmu určit jako $O(n^3(T_\oplus + T_\odot + T_*))$, což je např. v případě konstantní složitosti operací v polookruhu rovno $O(n^3)$.

Příklady polookruhů a odpovídajících úloh

$\mathcal{P}_1 \;=\; \langle \mathbf{R}^+ \cup \{\infty\}, \min, +, \infty, 0\rangle$
Nosičem tohoto polookruhu je množina nezáporných reálných čísel doplněná o nevlastní maximální prvek ∞. Struktury $\langle \mathbf{R}^+ \cup \{\infty\}, \min, \infty\rangle$ a $\langle \mathbf{R}^+ \cup \{\infty\}, +, 0\rangle$ jsou zjevně monoidy, operace minimalizace je navíc komutativní a idempotentní. Nevlastní prvek ∞ je nulovým prvkem vůči druhé operaci, neboť při počítání s ním pokládáme $a + \infty = \infty + a \;= \infty$ pro libovolné $a \in \mathbf{R}^+ \cup \{\infty\}$.

Pro nekonečnou množinu operandů provádí operace minimalizace výběr infima (největší dolní meze) této množiny, jehož existence je v $\mathbf{R}^+$ zaručena. Infimum také splňuje podmínku 11 o asociativnosti, komutativnosti a idempotenci, stejně jako distributivní zákon vůči součtu (podmínka 12).

Operace uzávěru je podle volby základních operací tohoto polookruhu rovna

$$a^* \;=\; \min(0, a, a+a, a+a+a, \ldots) \;=\; \inf_{k=0}^{\infty}\{ka\} \;=\; 0$$

pro libovolné $a \in \mathbf{R}^+ \cup \{\infty\}$, takže nemusíme středový faktor ve vztahu (7.14) uvažovat, a dostáváme tak známou rekurentní formuli pro hledání nejkratších cest

$$s_{ij}^{(k)} = \min(s_{ij}^{(k-1)}, s_{ik}^{(k-1)} + s_{kj}^{(k-1)})$$

My ovšem víme, že v případě nezáporného ohodnocení hran grafu G je s ohledem na výpočetní složitost vhodné namísto Floydova-Warshallova algoritmu použít n-násobného opakování Dijkstrova algoritmu.

$\mathcal{P}_2 \;=\; \langle \mathbf{R} \cup \{-\infty, \infty\}, \min, +, \infty, 0\rangle$
Tato struktura je rovněž uzavřeným polookruhem, což lze dokázat podobně jako v případě polookruhu $\mathcal{P}_1$ s tím, že uvažujeme vlastnosti druhého přidaného nevlastního prvku $-\infty$ vůči polookruhovým operacím.

Zavedení tohoto prvku je nutné pro zajištění existence infima pro zdola neomezené posloupnosti reálných čísel. Aditivní vlastnosti tohoto nevlastního prvku jsou podobné jako pro prvek ∞, ale nemohou být zcela stejné, neboť bak by existovaly dva různé nulové prvky pro druhou operaci v polookruhu, což není možné (viz cvič. 7.4-1).

Pro operaci uzávěru tentokrát dostáváme

$$a^* \;=\; \begin{cases} 0 & \text{je-li } a \;\geq 0 \\ -\infty & \text{je-li } a < 0. \end{cases}$$

I v případě polookruhu $\mathcal{P}_2$ se jedná o úlohu hledání nejkratších cest, tentokrát však v grafu s libovolným (reálným) ohodnocením hran. Druhý případ ve výsledku operace uzávěru při

promítnutí do rekurentního vztahu (7.14) vyjadřuje důsledek existence cyklu se záporným ohodnocením, neboť tím se degeneruje na $-\infty$ vzdálenost libovolné dvojice uzlů, na jejichž spojení se takový cyklus nachází.

$\mathcal{P}_3 = \langle \mathbf{R} \cup \{-\infty, \infty\}, \max, +, -\infty, 0 \rangle$
Tento polookruh získáme z předchozího přechodem k opačnému uspořádání, takže je k němu v duálním vztahu. Protože se alternativy kombinují výběrem maxim, jedná se při jeho použití o hledání nejdelších cest mezi uzly, a nebezpečí tak představují cykly s kladnou délkou. Pro operaci uzávěru v tomto polookruhu platí

$$a^* = \begin{cases} 0 & \text{je-li } a \leq 0 \\ \infty & \text{je-li } a > 0, \end{cases}$$

což odpovídá vytváření neomezeně dlouhých spojení opakovaným připojováním kladných cyklů.

$\mathcal{P}_4 = \langle \langle 0, 1 \rangle, \max, \cdot, 0, 1 \rangle$
Snadno se přesvědčíme, že reálná čísla z intervalu $\langle 0, 1 \rangle$ splňují pro operace maxima a součinu všechny podmínky uzavřeného polookruhu. Při aplikaci Floydova-Warshallova algoritmu pro tento polookruh je možné využít toho, že uzávěr libovolného prvku je roven 1, takže v rekurentním vztahu (7.14) odpadne středový činitel. To jinými slovy znamená, že od žádných cyklů grafu nehrozí nebezpečí neomezeného protahování spojení. [1].

Význam využití polookruhu $\mathcal{P}_\triangle$ snadno pochopíme na příkladu, kdy graf je modelem nějakého komunikačního systému. Každé hraně můžeme přiřadit spolehlivost odpovídající linky, tedy pravděpodobnost bezchybného přenosu např. během jedné minuty provozu. Celková spolehlivost nějaké komunikační cesty je pak dána součinem spolehlivostí jednotlivých hran a zajímá nás přirozeně zjištění těch nejspolehlivějších cest.

$\mathcal{P}_5 = \langle \langle 0, 1 \rangle \cup \{\lambda\}, \min, \cdot, \lambda, 1 \rangle$
Představme si, že bychom namísto nejspolehlivější cesty potřebovali naopak znát cestu nejméně spolehlivou. Zdá se nejjednodušší použít polookruhu $\mathcal{P}_4$ a zaměnit v něm operaci maxima operací minima. Pro tuto operaci bychom ale jako jednotkový prvek museli vzít horní mez intervalu, tedy číslo 1. Tím by ale měly obě operace polookruhu stejný jednotkový prvek, a ten by už nemohl současně hrát roli nulového prvku pro operaci násobení.

Z této situace se dostaneme prostě tak, že k číselnému intervalu $\langle 0, 1 \rangle$ přidáme nevlastní prvek λ, pro který dodefinujeme operace minima a součinu tak, aby byl λ jednotkovým prvkem pro první operaci a nulovým prvkem pro druhou. Položíme tedy

$$\begin{aligned} \min(a, \lambda) &= \min(\lambda, a) = a \\ a \cdot \lambda &= \lambda \cdot a = \lambda \end{aligned}$$

Po tomto rozšíření nosiče struktury a doplnění pravidel pro obě polookruhové operace jsme již docílili splnění všech potřebných podmínek a výsledná struktura je tedy polookruhem.

$\mathcal{P}_6 = \langle R_k, \max, \min, r_{\min}, r_{\max} \rangle$
Nechť R_k představuje libovolnou konečnou množinu reálných čísel (popř. konečnou podmnožinu libovolné úplně uspořádané množiny) a $r_{\min}$, resp. $r_{\max}$ její minimální, resp. maximální prvek. Operace maxima i minima – jak víme z předchozích příkladů polookruhů – splňují požadované podmínky, dokonce jsou obě idempotentní a jednotkový prvek vůči jedné z nich je nulovým prvkem vůči druhé. Rovněž distributivní zákon platí jak pro kombinaci $\min(a, \max(b, c))$, tak i pro kombinaci $\max(a, \min(b, c))$. Z uvedeného vyplývá, že polookruhem je i duální struktura $\mathcal{P}_{6'} = \langle R_k, \min, \max, r_{\max}, r_{\min} \rangle$.

Vzhledem ke konečnosti množiny R_k obsahuje libovolná nekonečná posloupnost prvků z R_k jen konečně mnoho různých hodnot, a tak je zaručeno rovněž splnění požadavků pro uzavřené

[1] Uvedené vlastnosti plynou bezprostředně z toho, že polookruh $\mathcal{P}_\triangle$ je izomorfním obrazem polookruhu $\mathcal{P}_\infty$ při zobrazení $h : \mathbf{R}^+ \cup \{\infty\} \leftrightarrow \langle 0, 1 \rangle,\ h(x) = \exp(-x)$

polookruhy. Operace uzávěru pak dává pro polookruh $\mathcal{P}_6$ hodnotu $a^* = r_{\max}$ a pro $\mathcal{P}_{6'}$ duální hodnotu $a^* = r_{\min}$.

I polookruh $\mathcal{P}_6$ má uplatnění při aplikaci grafového modelu komunikační sítě. Tentokrát bude každá linka (hrana) ohodnocena svou propustností (kapacitou, šířkou), propustnost spojení je pak dána jeho nejužším místem, tedy minimem propustností jeho hran. Při výběru z alternativních spojení volíme ten s větší propustností. Použitím Floydova-Warshallova algoritmu tedy řešíme úlohu určení maximální propustnosti spojení mezi všemi dvojicemi uzlů. Díky konstantní hodnotě uzávěru nemá případná existence cyklů na řešení této úlohy žádný vliv a středový činitel v rekurentním vztahu (7.14) můžeme vynechat. Podobným způsobem je možné charakterizovat i použití duálního polookruhu $\mathcal{P}_{6'}$.

Cvičení

7.4-1. Aby $\mathcal{P}_2$ byl skutečně polookruhem, je třeba vhodně stanovit výsledek operací součtu a součinu v případě, že jedním operandem je ∞ a druhým $-\infty$. Stanovte tento výsledek s ohledem na grafovou interpretaci součtu (alternativní spojení) a součinu (skládání navazujících spojení).

7.4-2. Struktura $\mathcal{P}_7 = \langle \mathbf{N} \cup \{\infty\}, +, \cdot, 0, 1\rangle$ není uzavřeným polookruhem (proč?), přesto dává použití zobecněného Floydova-Warshallova algoritmu s takto definovanými operacemi snadno interpretovatelný výsledek. Předpokládejte, že použitá výchozí matice je rovna matici sousednosti grafu, a určete význam prvků $s_{ij}^{(n)}$ ve výsledné matici. Určete výsledek operace uzávěru v této struktuře.

7.4-3. Určete, zda $\mathcal{P}_8 = \langle \mathbf{R}^+ \cup \{\infty\}, +, \cdot, 0, 1\rangle$ představuje uzavřený polookruh. Určete rovněž výsledek operace uzávěru.

7.4-4. Je možné použít Dijkstrova algoritmu pro orientovaný graf s ohodnocením hran z libovolného uzavřeného polookruhu? Jak je to v případě Bellmanova-Fordova nebo zrychlené verze algoritmu NEJKRATŠÍ-CESTY ze závěru odst. 7.1?

7.4-5. Je možné pro zobecněný Floydův-Warshallův algoritmus počítat všechny iterace hodnot $s_{ij}^{(k)}$ v jediné matici analogicky jako se to provádí v algoritmu FLOYD-WARSHALL-1 ze cvičení 7.2-2?

7.4-6. Nechť $A = \{a_1, a_2, \ldots, a_k\}$ je nějaká abeceda, A^* je množina všech slov nad touto abecedou. Jazykem nad abecedou A nazýváme libovolnou podmnožinu $L \subseteq A^*$. Pro dva jazyky $L_1, L_2 \subseteq A^*$ definujeme jejich zřetězení $L_1 \cdot L_2$ vztahem

$$L_1 \cdot L_2 = \{w_1 w_2 : w_1 \in L_1 \ \&\ w_2 \in L_2\},$$

kde $w_1 w_2$ znamená zřetězení slov w_1 a w_2. Určete, zda množina všech jazyků nad abecedou A s operacemi sjednocení a zřetězení představuje polookruh. Které jazyky hrají roli jednotkových prvků pro uvedené operace? Jaký význam má v tomto případě operace uzávěru?

7.5 Dynamické programování

Hledání nejkratších cest mezi všemi dvojicemi uzlů je typickým příkladem **optimalizačního problému**, pro jehož řešení je vhodné použít metody **dynamického programování**. Základní myšlenka postupu je podobná jako u algoritmů typu „rozděl a panuj“ – tedy rozložit problém na podproblémy, ty rekurzivně vyřešit, a pak kombinovat dílčí řešení do celkového výsledku.

Tento postup dává dobré výsledky, pokud jsou vznikající podproblémy navzájem nezávislé, vede však k extrémně neefektivním algoritmům, jestliže se stejné podúlohy opakují vícekrát. V algoritmech založených na principu dynamického programování se každý podproblém vyřeší pouze jednou, použitelné dílčí výsledky nižších úrovní rozkladu problému se uchovávají, a tak mohou být později opakovaně použity pro určení řešení a vyšší úrovni rozkladu.

Jako jednoduchý příklad aplikace této myšlenky může posloužit návrh algoritmu pro výpočet n-tého prvku Fibonacciho posloupnosti definované známým vztahem $F_n = F_{n-1} + F_{n-2}$ pro $n > 1$ s počátečními hodnotami $F_0 = 0$ a $F_1 = 1$. Pokud tuto úlohu řešíme rekurzivním algoritmem kopírujícím tvar definiční rekurentní formule, dostaneme řešení s exponenciální časovou složitostí. Naproti tomu jednoduchý iterační algoritmus, který ve dvou pomocných proměnných uchovává poslední a předposlední spočtený prvek posloupnosti a na jejich základě určí prvek následující, má pouze lineární časovou složitost. Algoritmy dynamického programování mají většinou lineární či kvadratickou paměťovou složitost, neboť dílčí výsledky uchovávají formou jedno- či dvourozměrných tabulek.

Návrh algoritmů založených na dynamickém programování je obecně možné rozdělit do následujících kroků:

1. určení struktury optimálního řešení
2. vyjádření hodnoty optimálního řešení rekurzivní (rekurentní) formou
3. určení jednotlivých kroků a jejich pořadí při výpočtu tabelovaných hodnot postupem zdola nahoru
4. sestrojení optimálního řešení z vypočtených hodnot

Při hledání nejkratších cest Floydovým-Warshallovým algoritmem odpovídá kroku 1 rozhodnutí určovat postupně nejkratší cesty s vnitřními uzly z množin $\{u_1, u_2, \ldots, u_k\}$ pro $k = 0, 1, \ldots, n$. Krok 2 končí získáním rekurentního vztahu (7.7), pomocí kterého v rámci kroku 3 navrhneme algoritmus 7.5 FLOYD-WARSHALL. Hodnoty uložené ve výsledné matici vzdáleností jsou ovšem jen w-délky nejkratších cest, můžeme z nich však (pokud je to požadováno) určit i cesty samotné – to je smysl kroku 4. Někdy je žádoucí rozšířit výpočet hodnot navrhovaný v kroku 3 o získání pomocných údajů, které sestrojení optimálního řešení nějak usnadní – takové údaje reprezentovala v naší úloze matice předchůdců.

Při použití postupů dynamického programování se nelze opírat o nějaký společný teoretický základ, který bychom aplikovali na jakýkoliv vhodně formulovaný problém. Společné rysy algoritmů dynamického programování je možné poznat pouze srovnáním několika konkrétních příkladů.

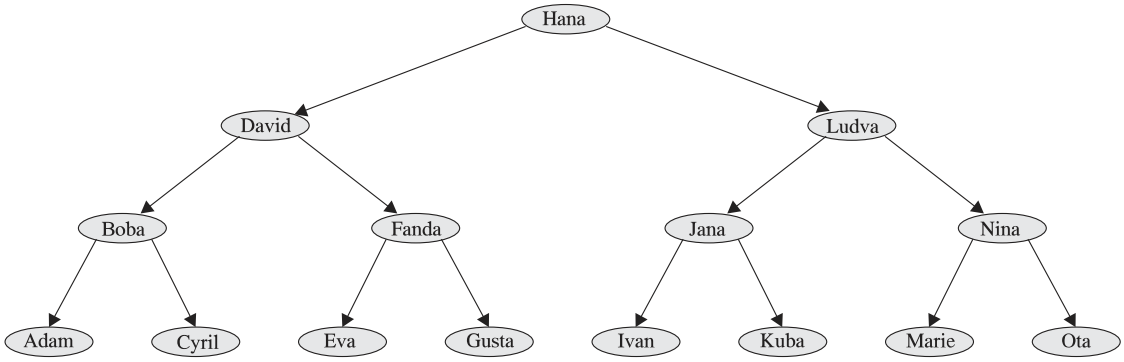
Konstrukce optimálních binárních vyhledávacích stromů

Binární vyhledávací stromy představují jednu z často používaných datových struktur, neboť za předpokladu vhodného vyvážení dovolují vyhledat zadanou hodnotu (nebo zjistit její neexistenci) jednoduchým algoritmem s logaritmickou časovou složitostí (měřeno počtem provedených srovnání klíčů uložených v uzlech stromu). Je ovšem zřejmé, že pro vyhledání klíče ležícího blíže ke kořeni je zapotřebí provést méně srovnání, než pro klíče ležící ve větší hloubce. Není-li pravděpodobnost hledání všech klíčů stejná, nabízí se otázka, jak sestrojit binární vyhledávací strom, který bude s ohledem na průměrnou složitost nalezení existujícího klíče optimální. Pokusme se nejprve tento problém přesně formulovat, abychom jej mohli vyřešit algoritmem dynamického programování.

Nechť je zadána množina klíčů $\{K_1, K_2, \ldots, K_n\}$ a odpovídající pravděpodobnosti jejich hledání $\{p_1, p_2, \ldots, p_n\}$. Předpokládejme, že jsme klíče uložili do uzlů binárního vyhledávacího stromu T, přičemž počet srovnání potřebných pro nalezení klíče K_i je c_i (je to tedy hloubka příslušného uzlu zvětšená o 1). Průměrný počet srovnání při hledání klíče ve stromu T se potom rovná

$$A(T) = \sum_{i=1}^{n} p_i c_i.$$

Chceme určit strukturu stromu T, pro který bude tato hodnota minimální. Na obr. 7.9 ukazujeme příklad binárního vyhledávacího stromu pro množinu klíčů a pravděpodobností uvedenou v následující tabulce. Tato tabulka obsahuje i odpovídající hodnoty koeficientů c_i a součiny



Obrázek 7.9: Binární vyhledávací strom

$p_i c_i$ potřebné pro určení průměrného počtu srovnání. Pro strom na našem obrázku dostáváme hodnotu $A(T) = 3.44$.

Předpokládejme vzestupné uspořádání klíčů v pořadí $K_1 < K_2 < \cdots < K_n$. Vybereme-li pro kořen stromu T klíč K_k, bude levý podstrom obsahovat klíče $K_1,\ldots,K_{k-1}$ a pravý podstrom klíče $K_{k+1},\ldots,K_n$. Jak levý, tak i pravý podstrom musí být opět sestrojeny optimálně. Nevíme ovšem předem, jaká je optimální volba k, takže je musíme nalézt probírkou všech možností.

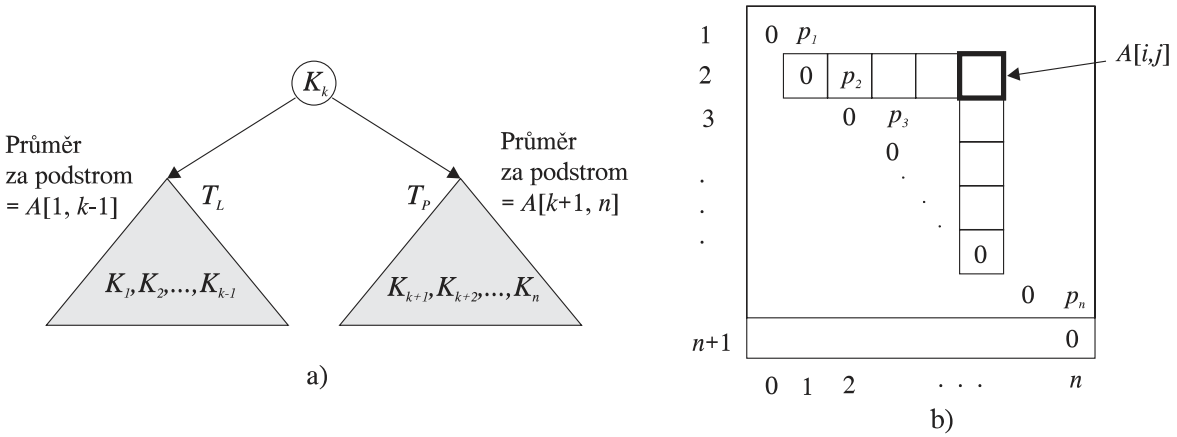
Klíč	p_i	c_i	$p_i c_i$
Adam	0.120	4	0.480
Boba	0.060	3	0.180
Cyril	0.010	4	0.040
David	0.030	2	0.060
Eva	0.150	4	0.600
Fanda	0.020	3	0.060
Gusta	0.050	4	0.200
Hana	0.040	1	0.040
Ivan	0.110	4	0.440
Jana	0.200	3	0.600
Kuba	0.085	4	0.340
Ludva	0.025	2	0.050
Marie	0.030	4	0.120
Nina	0.050	3	0.150
Ota	0.020	4	0.080
celkem:	1.000		3.440

Označme jako $A(i,j)$ minimální hodnotu průměrného počtu srovnání, kterého lze dosáhnout vytvořením binárního vyhledávacího stromu pro podmnožinu klíčů $K_i < \cdots < K_j$, $1 \leq i \leq j \leq n$. Pro hodnoty $A(i,j)$ platí okrajová podmínka

$$A(i,i) = p_i \cdot 1 = p_i,$$

neboť existuje pouze jeden binární strom s jedním uzlem, v němž bude uložen klíč K_i. Vytváříme-li naproti tomu binární strom s levým podstromem T_L, pravým podstromem T_P a kořenem s klíčem K_k, pak se jeho průměrný počet srovnání určí součtem následujících položek:

- průměr za podstrom T_L zvětšený o $1 \cdot p_s$ za každý klíč $K_s \in T_L$
- průměr za podstrom T_P zvětšený o $1 \cdot p_s$ za každý klíč $K_s \in T_P$
- příspěvek za klíč kořene o velikosti $1 \cdot p_k$



Obrázek 7.10: Konstrukce optimálního vyhledávacího stromu

Pro hodnoty argumentů $i < j$ lze tedy odvodit rekurentní vztah určující hodnoty $A(i,j)$ takto:

$$\begin{aligned} A(i,j) &= \min_{i \le k \le j} \left(A(i,k-1) + \sum_{s=i}^{k-1} p_s + A(k+1,j) + \sum_{s=k+1}^{j} p_s + p_k \right) = \\ &= \min_{i \le k \le j} \left(A(i,k-1) + A(k+1,j) + \sum_{s=i}^{j} p_s \right) = \\ &= \min_{i \le k \le j} \left(A(i,k-1) + A(k+1,j) \right) + \sum_{s=i}^{j} p_s \end{aligned}$$

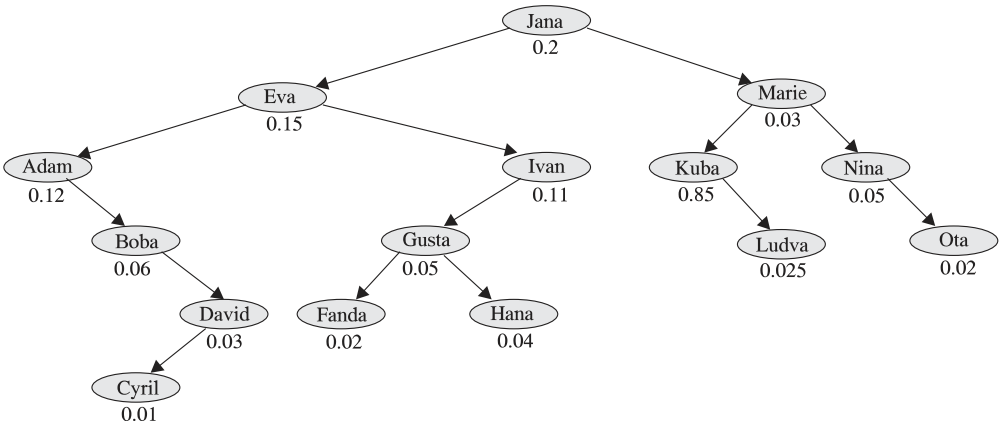
Kdybychom podle tohoto rekurentního vztahu navrhli rekurzivní algoritmus výpočtu hodnot $A(i,j)$, měl by zaručeně exponenciální složitost vinou opakovaného volání této funkce pro sobě se přibližující hodnoty argumentů. Namísto toho budeme hodnoty $A(i,j)$ reprezentovat horní trojúhelníkovou polovinou pole (viz obr. 7.10b), jehož prvky budeme počítat v přesně určeném pořadí na základě výše odvozeného rekurentního vztahu.

V poli A nejprve inicializujeme prvky $A[i,i]$ na hlavní diagonále hodnotami p_i, a pak postupně počítáme prvky v rovnoběžných diagonálách nad hlavní diagonálou, až dosáhneme pravého horního prvku $A[1,n]$. Uvedené pořadí plyne z toho, že pro výpočet $A[i,j]$ musíme znát hodnoty v i-tém řádku nalevo od prvku $A[i,j]$ a v j-tém sloupci pod prvkem $A[i,j]$. Nulové hodnoty pod diagonálou jsou zapotřebí proto, že rekurentní vztah pro $A[i,j]$ se odvolává i na prvky $A[i,i-1]$ a $A[j+1,j]$. Pro konstrukci optimálního stromu je důležité zaznamenat, pro kterou hodnotu k bylo dosaženo minima ve výrazu pro $A[i,j]$ – tuto hodnotu proto ukládáme do dalšího pole $r[i,j]$.

Algoritmus 7.10 Konstrukce optimálního vyhledávacího stromu

OPTIM-STROM(p)

```
1  for i := 1 to n do                          Inicializace prvků
2      A[i,i] := p[i]                          na hlavní diagonále,
3      A[i,i-1] := 0                           těsně pod ní
4      r[i,i] := i                             a také kořenů.
5  A[n+1,n] := 0                               Poslední pod diagonálou.
6  for diag := 1 to n-1 do                     Pro každou další diagonálu
7      for i := 1 to n-diag do                 procházej a počítej
8          j := i+diag                         hodnotu prvku A[i,j]
9          A[i,j] := min(A[i,k-1] + A[k+1,j]) pro i ≤ k ≤ j
```



Obrázek 7.11: Optimální vyhledávací strom

10	$r[i,j]$:='k minimalizující $A[i,j]$'	a kořene $r[i,j]$.
11	$A[i,j] := A[i,j] + \sum p_s$	Přidají se $p_i, \ldots, p_j$
12	return **A, r**	

Jádro algoritmu představuje dvojitý cyklus na řádcích 6 až 11, jehož tělo má časovou složitost $\Theta(n)$ pouze vinou určování minima – výpočet součtů $\sum p_s$ na řádce 11 lze organizovat tak, že postačí čas $O(n^2)$ na všechny (viz cvič. 7.5-2). Celková asymptotická složitost je tedy $\Theta(n^3)$.

Použitím algoritmu OPTIM-STROM na množinu klíčů s pravděpodobnostmi uvedenými v předchozím příkladu dostaneme optimální vyhledávací strom znázorněný na obr. 7.11 s průměrným počtem srovnání 2,785, což je o 0,655 méně než měl strom na obr. 7.9, takže jsme dosáhli zlepšení o téměř 20%.

Nejdelší společná podposloupnost

Další úlohou, na které ukážeme možnosti dynamického programování, je hledání nejdelší společné podposloupnosti dvou konečných posloupností. Posloupnost $A = \langle a_1, a_2, \ldots, a_m \rangle$ nazýváme **podposloupností** posloupnosti $B = \langle b_1, b_2, \ldots, b_n \rangle$, pokud je $m \leq n$ a existuje taková rostoucí posloupnost indexů $\langle i_1, i_2, \ldots, i_m \rangle$ v B, že je $a_j = b_{i_j}$ pro $j = 1, 2, \ldots, m$. Neformálně řečeno: podposloupnost vznikne vypuštěním libovolného počtu prvků dané posloupnosti.

Pro dané dvě posloupnosti A a B říkáme, že posloupnost C je **společnou podposloupností** A a B, pokud je současně podposloupností A i B. Je-li např. $A = \langle a, b, a, c, a, d, b \rangle$ a $B = \langle b, d, a, c, d, a, b \rangle$, potom $\langle b, a, c \rangle$ je společnou podposloupností A a B, není ovšem nejdelší možná. Naším úkolem je navrhnout algoritmus, který pro zadanou dvojici posloupností $A = \langle a_1, a_2, \ldots, a_m \rangle$ a $B = \langle b_1, b_2, \ldots, b_n \rangle$ nalezne jejich **nejdelší společnou podposloupnost** (NSP).

Při postupu hrubou silou bychom mohli postupně probírat každou z celkem 2^m podmnožin indexů posloupnosti A a testovat, zda příslušná podposloupnost je současně podposloupností B. Takový postup s exponenciální složitostí je zřejmě nevyhovující. Pro aplikaci metody dynamického programování se musíme pokusit odvodit pro NSP nějaké vhodné rekurentní vztahy. Ukazuje se, že je vhodné uvažovat vlastnosti začátečních částí posloupností, tzv. prefixů. Pro libovolné přirozené $k \leq m$ nazýváme **prefixem délky** k posloupnosti $A = \langle a_1, a_2, \ldots, a_m \rangle$ její podposloupnost $A_k = \langle a_1, a_2, \ldots, a_k \rangle$. Následující tvrzení odhaluje důležitý vztah mezi NSP posloupností a jejich prefixů.

Věta 7.11: Nechť je posloupnost $C = \langle c_1, c_2, \ldots, c_k \rangle$ nejdelší společnou podposloupností dvojice posloupností $A = \langle a_1, a_2, \ldots, a_m \rangle$ a $B = \langle b_1, b_2, \ldots, b_n \rangle$. Potom platí

(1) je-li $a_m = b_n$, potom je $c_k = a_m = b_n$ a C_{k-1} je NSP pro dvojici A_{m-1} a B_{n-1},

(2) je-li $a_m \neq b_n$ a $c_k \neq a_m$, potom je C NSP pro dvojici A_{m-1} a B,

(3) je-li $a_m \neq b_n$ a $c_k \neq b_n$, potom je C NSP pro dvojici A a B_{n-1}.

Důkaz: (1) Kdyby bylo $c_k \neq a_m$, potom bychom mohli k C připojit poslední prvek $a_m = b_n$ a dostat tak delší společnou podposloupnost, což je spor s předpokladem o C. Musí tedy platit $c_k = a_m = b_n$ a C_{k-1} je jistě společnou podposloupností pro dvojici A_{m-1} a B_{n-1}. Důkaz, že je současně i nejdelší, lze opět snadno provést sporem.

(2) Není-li poslední prvek z A obsažen v C, pak je C společnou podposloupností pro dvojici A_{m-1} a B. Kdyby ovšem existovala nějaká NSP pro dvojici A_{m-1} a B o délce větší než k, pak by byla společnou podposloupností i pro dvojici A a B, což je spor s volbou C.

(3) Není-li poslední prvek z B obsažen v C, pak je C společnou podposloupností pro dvojici A a B_{n-1}. Kdyby ovšem existovala nějaká NSP pro dvojici A a B_{n-1} o délce větší než k, pak by byla společnou podposloupností i pro dvojici A a B, což je spor s volbou C. △

Předchozí tvrzení lze shrnout tak, že NSP dvou posloupností v sobě obsahuje NSP prefixů těchto posloupností. To je velmi důležitá vlastnost **optimality podstruktur**, které využijeme pro formulaci rekurentních vztahů pro funkci Φ dvou argumentů – posloupností, která určí jejich nejdelší společnou podposloupnost. Symbolem $\cdot$ zde budeme vyjadřovat operaci připojení posledního prvku k posloupnosti, tedy

$$\langle a_1, a_2, \ldots, a_m \rangle \cdot b \;=\; \langle a_1, a_2, \ldots, a_m, b \rangle.$$

Symbolem $d(A)$ vyjadřujeme délku posloupnosti A (tj. počet jejích prvků) a ε značí prázdnou posloupnost.

Okrajové podmínky rekurence pro funkci Φ představuje prázdná posloupnost v alespoň jednom z argumentů – výsledkem je pak rovněž prázdná posloupnost. Pro neprázdné posloupnosti rozhoduje o výsledku shoda jejich posledních prvků a NSP jejich prefixů.

$$\Phi(A,B) = \begin{cases} \varepsilon & \text{pro } d(A)=0 \text{ nebo } d(B)=0 \\ \Phi(A_{m-1}, B_{n-1}) \cdot a_m & \text{pro } m=d(A)>0, n=d(B)>0, a_m = b_n \\ \max(\Phi(A_{m-1}, B), \Phi(A, B_{n-1})) & \text{pro } m=d(A)>0, n=d(B)>0, a_m \neq b_n \end{cases}$$

Přímá rekurzivní implementace funkce Φ by opět vedla na algoritmus exponenciální časové složitosti. Přitom existuje jen mn různých kombinací (neprázdných) prefixů v argumentu funkce Φ, takže její hodnoty bychom mohli snadno ukládat do obdélníkového pole. Těmito hodnotami jsou však posloupnosti, a tak namísto nich budeme počítat **délky** $s[i,j]$ nejdelších společných

j	0	1	2	3	4	5	6
i	y_j	B	D	C	A	B	A
0 x_i	0	0	0	0	0	0	0
1 A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2 B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3 C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4 B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5 D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6 A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7 B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Obrázek 7.12: Nejdelší podposloupnost

podposloupností v závislosti na délkách i, j uvažovaných prefixů posloupností A, B. Následující rekurentní vztah pro délky jsme dostali snadným přepisem vztahu pro funkci Φ.

$$s[i,j] = \begin{cases} 0 & \text{pro } i = 0 \text{ nebo } j = 0 \\ s[i-1,j-1]+1 & \text{pro } i > 0, j > 0, a_i = b_j \\ \max(s[i-1,j], s[i,j-1]) & \text{pro } i > 0, j > 0, a_i \neq b_j \end{cases}$$

V následujícím algoritmu počítáme kromě $s[i,j]$ také hodnoty $t[i,j]$, které nám dovolí zkonstruovat nalezenou nejdelší podposloupnost, resp. všechny takové podposloupnosti.

Algoritmus 7.12 Nejdelší společná podposloupnost

NSP(A, B)

```
1   m := d(A); n := d(B)                          Určení délek posloupností
2   for i := 1 to m do s[i,0] := 0                a inicializace
3   for j := 1 to n do s[0,j] := 0                okrajů pole s.
4   for i := 1 to m do                            Hlavní cyklus
5       for j := 1 to n do                        běží po řádcích:
6           if  a_i = b_j                         je-li společný konec,
7               then s[i,j] := s[i-1,j-1] + 1     použije se.
8                    t[i,j] := " ↖ "
9               else if s[i-1,j] ≥ s[i,j-1]       Jinak se bere
10                   then s[i,j] := s[i-1,j]       delší ze dvou
11                        t[i,j] := " ↑ "          uvažovaných možností.
12                   else  s[i,j] := s[i,j-1]
13                        t[i,j] := " ← "
14  return s, t
```

Časová složitost tohoto algoritmu je $O(mn)$, neboť všechny operace těla vnořeného cyklu na řádcích 6 až 13 lze provést v konstantním čase. Na obr. 7.12 ukazujeme v jediné kombinované tabulce výsledné obsazení polí s i t po provedení algoritmu NSP. Z této tabulky také vyplývá, jak interpretovat hodnoty v poli t: diagonální šipka znamená shodu prvků (a tedy jeho zařazení do NSP), svislá, resp. vodorovná šipka znamená ignorovat prvek v první, resp. druhé posloupnosti. Tmavými políčky je vyznačena serie délek začínající v pravém dolním rohu, která určuje jednu nejdelší společnou podposloupnost. Prvky této podposloupnosti jsou vyznačeny obdobně na okraji tabulky. Formulaci algoritmu pro konstrukci nejdelší podposloupnosti z hodnot s a t ponecháváme jako cvičení (viz cvič. 7.5-4).

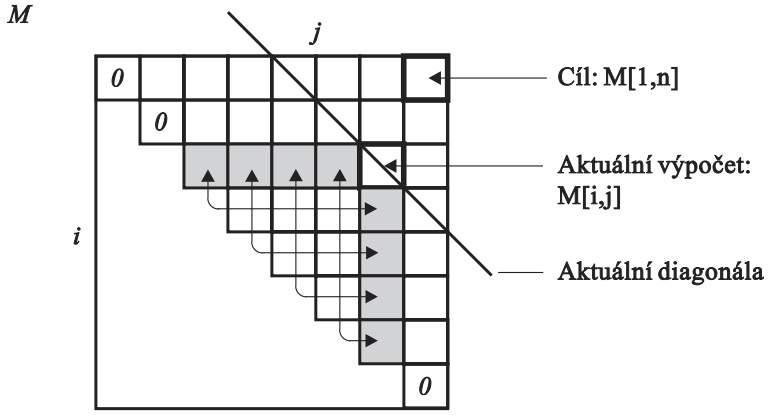
Násobení posloupnosti matic

Z lineární algebry známe definici operace násobení matic, ze které dokážeme snadno určit, že pro vynásobení matice A typu (m,n) maticí B typu (n,p) je třeba provést $m \times n \times p$ operací násobení jejich prvků. Je také známo, že násobení matic je asociativní, tedy pro matice A, B, C, které jsou po řadě typů $(m,n), (n,p), (p,r)$, platí

$$A \times (B \times C) \;=\; (A \times B) \times C.$$

Přestože je tedy výsledek nějaké posloupnosti operací násobení matic stejný, ať jej počítáme asociováním zprava, zleva, nebo jakkoliv zvoleným způsobem sdružování dvojic matic, počet operací násobení prvků se může v jednotlivých postupech dost podstatně lišit. Tak např. pro trojici matic A, B, C typů po řadě $(30,2), (2,40), (40,5)$ dostáváme pro dva možné způsoby výpočtu následující počty potřebných operací násobení prvků:

$$\begin{array}{lll} A \times (B \times C) & 30 \times 2 \times 5 + 2 \times 40 \times 5 = 300 + 400 & = \quad 700 \\ (A \times B) \times C & 30 \times 2 \times 40 + 30 \times 40 \times 5 = 2400 + 6000 & = \quad 8400 \end{array}$$



Obrázek 7.13: Násobení posloupnosti matic – postup výpočtu hodnot $M[i,j]$

Je vidět, že pokud se v případě výpočtu dvojice operací násobení matic mohou potřebné počty součinů prvků lišit více než desetinásobně, bude u delší posloupnosti operací násobení rozsáhlých matic jistě žádoucí provést výpočet optimálním způsobem.

Nechť jsou dány matice $A_1, A_2, \ldots, A_n$ a nechť typ každé matice A_i je (d_{i-1}, d_i) pro $i = 1, 2, \ldots, n$. Máme určit minimální počet násobení prvků potřebný k výpočtu součinu

$$A_1 \times A_2 \times \ldots \times A_n$$

a samozřejmě také odpovídající pořadí, v němž se dané matice mají násobit. S použitím metody dynamického programování vyřešíme nejdříve první z uvedených problémů a pak algoritmus doplníme tak, aby poskytoval i odpověď na druhou část zadání.

Pro dvojici i, j $(1 \leq i \leq j \leq n)$ označíme jako $M(i,j)$ minimální počet součinů prvků potřebný pro výpočet $A_i \times A_{i+1} \times \ldots \times A_j$. Při výpočtu součinu této posloupnosti matic potřebujeme rozhodnout, jak ji rozdělit optimálním způsobem na dvě části

$$(A_i \times \ldots \times A_k) \times (A_{k+1} \times \ldots \times A_j).$$

Součin v každé z těchto částí předpokládejme počítat optimálním způsobem, a na závěr tak budeme násobit dvě matice typu (d_{i-1}, d_k) a (d_k, d_j). Pro hodnoty $M(i,j)$ pak dostáváme následující rekurenci:

$$\begin{aligned} M(i,i) &= 0 \\ M(i,j) &= \min_{i \leq k \leq j-1} (M(i,k) + M(k+1,j) + d_{i-1} d_k d_j) \quad \text{pro } 1 \leq i \leq j \leq n \end{aligned}$$

Podobně jako při určování nejdelší společné podposloupnosti, také nyní budeme hodnoty $M(i,j)$ ukládat do čtvercové matice, v níž vyplníme na začátku hlavní diagonálu nulovými hodnotami (viz obr. 7.13). Jako další pak počítáme prvky vpravo od hlavní diagonály matice, a postupujeme pak k dalším řadám v diagonálním směru. Jelikož naše úloha vyžaduje vedle optimálního počtu součinů prvků zjistit také, v jakém pořadí se mají matice násobit, budeme si v samostatné matici KK pamatovat, jakým způsobem se posloupnost matic v součinu optimálně člení na dvě části. Popsaný postup lze vyjádřit následujícím algoritmem.

Algoritmus 7.13 Násobení posloupnosti matic

NasobeniMatic$(d, n, PocNas, KK)$

```
1   for i := 1 to n do M[i,i] := 0;          inicializuje se lokální matice M
2   for diag := 1 to n − 1 do                zajistí se počítání diagonál,
3       for i := 1 to n − diag do            pak se prochází podle diagonály
4           j := i + diag;
```

```
5        M[i,j] := min_{i≤k ≤j−1}(M[i,k] + M[k+1,j] + d[i−1]·d[k]·d[j]);
6        KK[i,j] :=„k použité pro získání M[i,j]“
7   PocNas := M[1,n];
8   return PocNas, KK
```

Výsledkem právě uvedeného algoritmu je minimální potřebný počet ($PocNas$) operací násobení dvojice čísel pro získání požadovaného součinu posloupnosti matic a informace o tom, v jakém pořadí je třeba zadané matice násobit. Druhý z výsledků je ovšem ukrytý v poli KK, takže ještě ukážeme, jak je možné informaci v poli KK využít. Následující procedura zobrazí posloupnost násobených matic $A_i \times \ldots \times A_j$ s vloženými závorkami, které explicitně ukazují pořadí výpočtu jednotlivých maticových součinů.

```
ZobrazPoradi(i, j, KK)
1   if i = j then write('A',i)                    žádný součin, jen jedna matice A_i
2   else  k := KK[i,j]; write('(');               výstup levé závorky
3         ZobrazPoradi(i,k); write('*');          zobrazí součin A_i × ... × A_k
4         ZobrazPoradi(k+1,j); write(')');        zobrazí součin A_{k+1} × ... × A_j
```

Na uvedených třech typech úloh řešených metodou dynamického programování jsme ukázali konkrétní uplatnění obecně formulovaných zásad. Je vidět, že pro úspěch použití metody dynamického programování je vedle správné formulace rekurentních vztahů potřebné především vhodně zvolit použité datové struktury a způsob práce s nimi.

Cvičení

7.5-1. Navrhněte algoritmus časové složitosti $O(n)$, který na základě hodnot $r[i,j]$ spočítaných algoritmem OPTIM-STROM a rostoucí posloupnosti klíčů $\{K_i\}_1^n$ vytvoří strukturu odpovídajícího vyhledávacího stromu.

7.5-2. Navrhněte takový způsob výpočtu součtů $\sum_{s=i}^{j} p_s$ a odpovídající úpravu algoritmu OPTIM-STROM, které zajistí získání všech potřebných hodnot v čase $O(n^2)$.
(*Návod:* Využijte části pod „nulovou“ diagonálou pole A.)

7.5-3. V čem se liší určení optimálního vyhledávacího stromu od úlohy řešené algoritmem 5.18 HUFFMAN v odstavci 5.5? Navrhněte jednoduchý hladový algoritmus pro nalezení optimálního vyhledávacího stromu a rozhodněte, zda jeho výsledkem je vždy opravdu optimální binární vyhledávací strom.

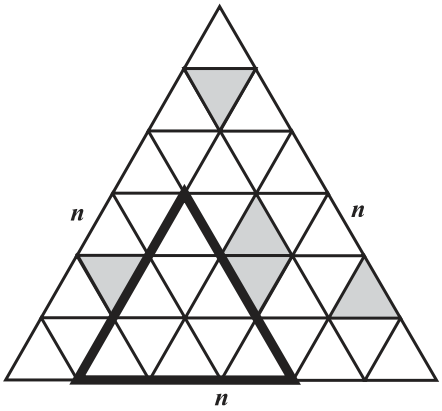
7.5-4. Navrhněte algoritmus, který na základě první posloupnosti A a výsledné matice t algoritmu 7.12 NSP zobrazí (nebo vytvoří) poslední algoritmem nalezenou nejdelší společnou podposloupnost. Časová složitost tohoto algoritmu by měla být $O(\max(m,n))$.

7.5-5. Určete NSP posloupností $\langle a,b,b,a,b,a,b,a\rangle$ a $\langle b,a,b,a,a,b,a,a,b\rangle$.

7.5-6. Vzhledem k tomu, že hodnota $c[i,j]$ závisí pouze na prvcích $c[i-1,j-1], c[i-1,j]$, $c[i,j-1]$ a na vztahu a_i a b_j, je možné zpětně určit, ze které z těchto hodnot se $c[i,j]$ v algoritmu NSP určilo. Hodnoty $t[i,j]$ není tedy nezbytně nutné ukládat. Navrhněte odpovídající modifikaci algoritmu pro zobrazení nejdelší společné podposloupnosti vycházející pouze z matice c a určete její časovou složitost.

7.5-7. Navrhněte modifikaci algoritmu NSP, která určí délku nejdelší společné podposloupnosti s použitím pouze dvou řádkových vektorů délky $\min(m,n)$ namísto celé matice c. Je možné paměťové nároky ještě dále snížit na $\min(m,n)+O(1)$? Jak by se v tomto případě rekonstruovala výsledná podposloupnost a s jakou časovou složitostí?

7.5-8. Navrhněte algoritmus časové složitosti $O(n^2)$, který nalezne nejdelší rostoucí podposloupnost n-prvkové posloupnosti čísel.



Obrázek 7.14: Hledání největšího bílého trojúhelníka

7.5-9. Navrhněte algoritmus, který určí s použitím hodnot uložených v poli KK algoritmem 7.13 pro optimalizaci násobení posloupnosti matic, jakým způsobem ozávorkovat posloupnost matic v součinu, aby se dosáhlo optimálního počtu operací při výpočtu jejich součinu.

7.5-10. Je zadána rovinná oblast ve tvaru rovnostranného trojúhelníka o straně dlouhé n, která je rozdělena do n^2 rovnostranných trojúhelníků s jednotkovou délkou (viz obr. 7.14). Některé z malých trojůhelníků jsou černé, ostatní bílé. S použitím metody dynamického programování navrhněte algoritmus, který určí délku strany rovnostranného trojúhelníka tvořeného největším počtem bílých trojúhelníků.
(*Návod:* Vytvořte vhodnou maticovou reprezentaci zadané oblasti, z ní ve vhodném pořadí určete prvky matice, které budou určovat velikost maximálního bílého trojúhelníka majícího horní/dolní vrchol v odpovídající pozici. Nezapomeňte, že výsledný trojúhelník může být „postavený na špičku“.)