

Matici délky A:

$$a(i,j) = \begin{cases} 0 & \text{pro } i=j \\ a(e) & \text{pro } e = (i,j) \in E \\ \infty & \text{pro } (i,j) \notin E \end{cases}$$

Přednáška

12.3

délka z vrcholu do stejného vrcholu je 0

délka po hraně je cena hrany

pokud vrcholy nejsou spojeny hranou, cena je ∞

Matici vzdáleností U:

$$u(i,j) = \begin{cases} 0 & \text{pro } i=j \\ \text{cena cesty z } i \text{ do } j, \text{ pokud taková cesta existuje} \\ \infty & \text{pokud cesta z } i \text{ do } j \text{ neexistuje} \end{cases}$$

Trojúhelníková nerovnost v grafu: Ještěž v grafu G neexistuje cyklus sítomé délky, pak pro každé tři vrcholy x, y, z platí:

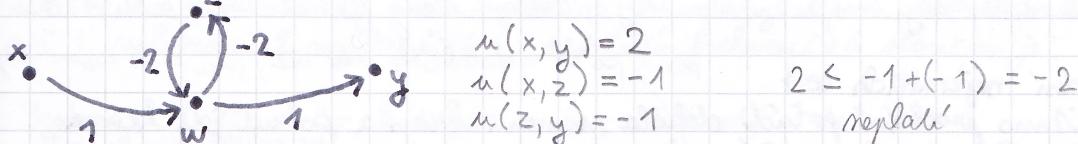
$$u(x,y) \leq u(x,z) + u(z,y)$$

Důkaz: Pokud cesta z x do z měla se z do y neexistuje, bude prava strana nekonečno a tvrzení platí. Cesta z x do y pak totiž bude existovat a tedy bude menší než nekonečno, měla existovat nebude a pak $\infty \leq \infty$ lze říct platí.

Pokud cesty existují, označme C_1 nejkratší cestu z x do z , a C_2 nejkratší cestu z z do y . Upojím C_1 a C_2 vnitře sled $C = C_1 \cup C_2$ s délkou rovnou součtu cest C_1 a C_2 . Pokudž nene, že v grafu nejsou žádny sítomé délky, tak sled určitě obahý cestu, která je kratší nebo stejně dlouhá jako C . Našli jsme tedy cestu, která nesmí splňovat, proto to určitě bude opakovat i nejkratší cestu.

Příklad grafu s cyklem sítomé délky

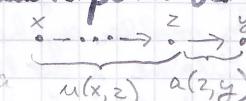
[Sled: posloupnost vrcholů a hran
cesta: sled, kde se neopakují vrcholy]



Bellmannův princip optimality: Ještěž v grafu G neexistuje cyklus sítomé délky, pak pro každé tři vrcholy x, y, z platí:

$$u(x,y) = \min_{z \neq y} (u(x,z) + a(z,y))$$

\approx nejkratší cesta se rozdělí na
a následně se nejlepší z



$z \neq y$, protože
(to nazýváme maximem)

Důkaz: Pokud cesta z x do y neexistuje, princip sítomé délky ($\infty = \infty$)

Předpokládejme, že cesta existuje, tedy $u(x,y) < \infty$. Protože pro každé dva vrcholy z, y platí $u(z,y) \leq a(z,y)$, vyvážíme možuhelníkovou nerovnost: $u(x,y) \leq u(x,z) + a(z,y)$.

$$\text{Proto: } u(x,y) \leq \min_{z \neq y} (u(x,z) + a(z,y))$$

Rovnost nastane pro nějaké z , které je předposlední na cestě z x do y .

Obecné schéma pro hledání nejkratší cesty z vrcholu r ($u(r,v) \Rightarrow U(v)$)

1) (inicializace)

$$U(r) := 0, U(v) := \infty \text{ pro } v \neq r$$

= máli jsme lepsi cestu do w ,
nová cesta povede přes v

2) (upracování hran)

existuje-li hranu $e = (v,w)$ taková, že $U(w) > U(v) + a(e)$
potom $U(w) := U(v) + a(e)$

3) (ukončení)

ještěž $U(w) \leq U(v) + a(e)$ pro každou hranu $e = (v,w)$

stop

jinak pokračuj směrem 2

✓ délka všech cest je nejméně $|V|-1$

- složitost takového algoritmu je $O(|V|-1) \cdot |E|$

✓ budeme procházet všechny hrany

Tvrzení: Ještě když G neobsahuje cyklus sáproné délky, a $U(v) \neq \infty$, pak $U(v)$ je délka nějaké cesty z r do v.

Důkaz: Označme $U_k(y)$ hodnotu $U(y)$ v čase k. Předpokládejme, že $U(v) < \infty$, v nějakém kroku jsme museli hodnotu $U(v)$ změnit z ∞ na něco menšího. Označme tento krok S_k , a hodnotu po směně $U_{k+1}(v)$. Máme tedy sled:

$$v_1, e_1, v_2, e_2, \dots, v_{k-1}, e_{k-1}, v_k = v$$

kde cena $U_{S_k}(v) = U_{S_{k-1}}(v_{k-1}) + a(e_{k-1})$, obecněji $U_{S_i}(v_i) = \sum_{j=1}^{i-1} a(e_j)$, kde $v_i = r$

Nyní je potřeba ukázat, že se nejdřív o sled, ale o cestu. Udeláme to sporem. Kdyby to nelyšla cesta, tak main někde ve sledu $v_i = v_j$, $i < j$, a protože algoritmus poskytuje hodnoty U snížuje, tak $U_j(v_j) < U_i(v_j)$. To ale znamená, že v j leží na cyklu sáproné délky, což je opor.

Věta: Ještě když graf G neobsahuje cyklus sáproné délky a hodnoty $U(v)$ byly někdy podle schématu z předchozí okamž, pak $U(v) = u(r, v)$.

Důkaz (sporem): Kdyby věta neplatila, tak by po ukončení práce algoritmu existoval vrchol v takový, že $U(v) > u(r, v)$. To lze znamenat, že $u(r, v) < \infty$. Označme nejkratší cestu C z vrcholu r do vrcholu v. Na této cestě je první vrchol východ' a pro něj platí $U(r) = u(r, r) = 0$, a poslední vrchol je v, pro který $U(v) > u(r, v)$. Označme na cestě první hranu $e = (x, y)$ takovou, že $U(x) = u(r, x)$, ale $U(y) > u(r, y)$.

$$U(y) > u(r, y) = u(r, x) + a(x, y) = u(r, x) + a(e)$$

Následující hrazení e, která by mohla snížit $U(y)$, schéma tedy nemůže skončit.

- Algoritmu na hledání nejkratších cest počít po n vrcholech, alg. stále běží, jsou tam sáproné hrany
- 1) jednoduchý algoritmus prochází pořád okolo sítí a pokud jede hranu, kterou už neslouží, udělá ho. Návštěvost je $O(m \cdot n)$, $m = |E|$, $n = |V|$
 - 2) sofistikovanější algoritmus si udržuje minciem M, ve kterém jsou vrcholy, se kterými by mohla ozechánout slegoucí hranu. Na začátku $M = \{r\}$, a když slegoume $U(x)$, přidáme x do M. Vrcholy odebíráme z M a posléze procházejeme jejich hranami.
 - jak správně upřímnit vrcholy z M ke spracování?
 - Dijkstra: bere nejlevnější vrchol (ale předpokládá nezáporné hranu)
 - Fito fronta - funguje i v obecných případech, vrcholy můžou být v M víckrát, ale užíváme čas na hledání minima

Hledání nejkratších cest mezi všemi vrcholy nazíváme - Floydův algoritmus. Označme $V = \{1, 2, 3, \dots, n\}$. Floydův alg. konstruuje matice $U_K = (u_k(i, j))$ řádu n, $k = 0, 1, \dots, n$, kde $u_k(i, j)$ je délka nejkratší cesty z i do j, při tom nejkratší cesta prochází vrcholy s čísly $0, 1, \dots, k$.

Tvrzení: U_0 je matice délek A - cesty mezi jinými vrcholy, jediné píspustné cesty jsou hranou samotné.

U_n je matice vrstevnosti - to, co chceme zjistit.

Matice U_{K+1} vznikne z matice U_K takto:

$$u_{K+1}(i, j) = \min \{u_k(i, j), u_k(i, k+1) + u_k(k+1, j)\}$$

Důkaz: U_0 a U_n plynou z definice. Dleží pro $k+1$ platit, že pro nejkratší cestu z i do j, se vrcholy $k+1$ nejdříve, pak má cenu $u_k(i, j)$, nebo když k němu vrcholem procházet a pak má cenu $u_k(i, k+1) + u_k(k+1, j)$.

Efektivní kódování: Je dán text nad abecedou C a pro každé $c \in C$ je dána frekvence výskytu c v textu: $c.\text{freq}$. Též je nějak binární kód textu C . Jde možnosti:

- kód stejné délky: kódová slova mají délku k , $|C| \leq 2^k$ počet různých znaků v C
- bezprefiksový kód: slova mají různou délku, ale žádoucí není prefisem jiného.

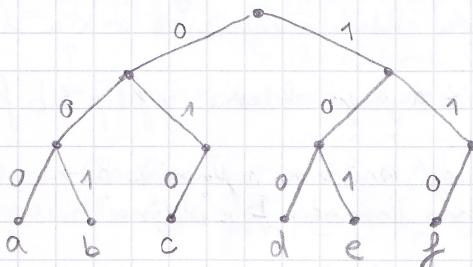
Je dán text nad abecedou C s frekvencemi $c.\text{freq}$ pro $c \in C$. Pak délka kódu u je

$$B(w) = \sum_{c \in C} c.\text{freq} \cdot |w(c)|, \text{ kde } w(c) \text{ je kódové slovo odpovídající } c.$$

Kádý kód si můžeme představit jako binární strom T , kde knotty jsou 0 nebo 1 a listy jsou $c \in C$. Délka dat po komprezji je

$$B(T) = \sum_{c \in C} c.\text{freq} \cdot d_T(c), \text{ kde } d_T(c) \text{ je hloubka } c \text{ ve stromě.}$$

Pr: $C = \{a, b, c, d, e, f\}$, $w(a) = 000$, $w(b) = 001$, $w(c) = 010$, $w(d) = 100$, $w(e) = 101$, $w(f) = 110$.

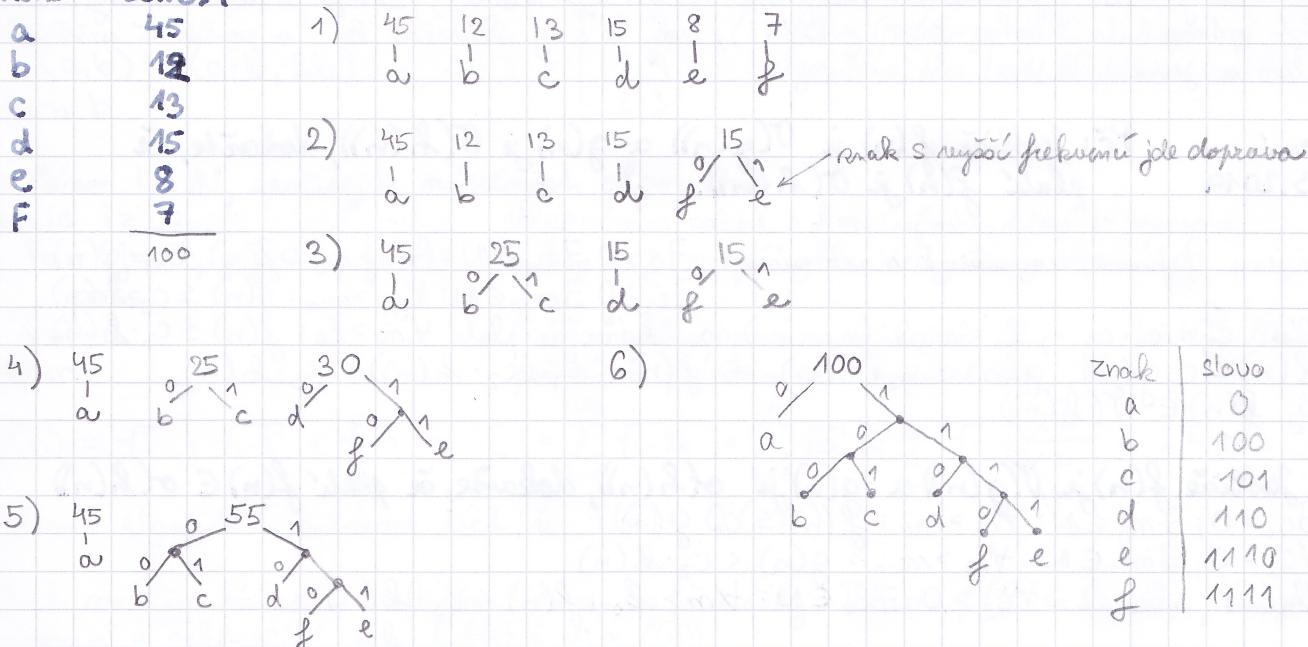


Je to kód stejné délky, ale užití nebude optimální. Mohli bychom ho skrátit pomocí $w(c) = 01$, $w(f) = 11$.

Huffmanův kód: přiřadí znakům kódy tak, aby byl vnitřní kód optimální

-konstrukce: pro každý znak vytvořím samostatný strom. Dleto mám méně abecedy než 1, vybíudem 2 nejménější (z nejménší frekvencí) a sloučím je.

Pr: Vystavte Huffmanův kód pro znaky s četností ve tabulce
znak četnost



Tvrzení 2: Nechť C je abeceda s $c.\text{freq}$ a $x, y \in C$ jsou symboly s nejmenším $c.\text{freq}$.
Ovnačme $C' = (C \setminus \{x, y\}) \cup \{z\}$, $z.\text{freq} = x.\text{freq} + y.\text{freq}$. Předpokládejme, že T' je optimální strom abecedy C' . Pak T' vložené nahrazením x a y za x a y je optimální strom C .

Tvrzení 1: Nechť C je abeceda s $c.\text{freq}$, $c \in C$. Předpokládejme, že $x, y \in C$ jsou dva symboly s nejmenším $c.\text{freq}$. Pak existuje optimální strom T , že x a y mají kódová slova stejné délky a liší se pouze v posledním bitu.

Důkaz tvrzení 2: Námeře, že optimální kód v abecedě C má strukturu délky B. Z ní je délka B příslušná jazyku C a strukturou Tabulkou následovně.

$$B = B' - z \cdot \text{freq} \cdot d_T(z) + (x \cdot \text{freq} + y \cdot \text{freq}) \cdot (d_T(z) + 1) =$$

\hookrightarrow píjdeme x a y o nízově míří
 \hookrightarrow oddeleme z

$\downarrow z=x+y$

$$= B' - z \cdot d + (x+y)(d+1) = B' - (x+y) \cdot d + (x+y) \cdot d + (x+y) \cdot 1 = B' + x+y$$

$$B(T) = B(T') + x \cdot \text{freq} + y \cdot \text{freq}$$

?

přeznačíme	$x \cdot \text{freq} \rightarrow x$
$y \cdot \text{freq} \rightarrow y$	$z \cdot \text{freq} \rightarrow z$
$d_T(z) \rightarrow d$	

Příklad: Máme matice $A, B \in \mathbb{R}^{n \times n}$. Jak můžeme je množením ležet výsledkem? ($A \cdot B$)

Vynásobení řádku A se sloupcem z B mává $\Theta(n^2)$. Této opakuje pro každý řádek, tedy n -krát, proto celé to má množnost $\Theta(n^3)$.

- Budeme množit Gausseuv algoritmus, pro matice $n = 2^k$. Gausse u rozdělí matice na 4 menší čtvrtce a spočítáme množením v nich:

$$A \cdot B = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} E & F \\ G & H \end{pmatrix}$$

$$S_1 = (B - D)(G + H)$$

$$S_2 = (A + D)(E + H)$$

$$S_3 = (A - C)(E + F)$$

$$S_4 = (A + B) \cdot H$$

$$S_5 = A \cdot (F - H)$$

$$S_6 = D \cdot (G - E)$$

$$S_7 = (C + D) \cdot E$$

$$A \cdot B = \begin{pmatrix} S_1 + S_2 - S_4 + S_6 & S_4 + S_5 \\ S_6 + S_7 & S_2 - S_3 + S_5 - S_7 \end{pmatrix}$$

Spočítejte časovou množnost Gausseova algoritmu.

$$T(n) = 7 \cdot T\left(\frac{n}{2}\right) + 18 \cdot \left(\frac{n}{2}\right)^2 = 7T(n) + \frac{9}{2}n^2$$

\uparrow 18x sčítání (10x n S_i a 8x při skládání matice)

\downarrow 7x množení matice polovičního řádu (pro $S_1 \dots S_7$)

Použijeme Master Theorem: $a=7, b=2, f(n) = \frac{9}{2}n^2, \log_b a = \log_2 7 \approx 3$ skoro tři podle pravidla 1: $f(n) \in \Theta(n^{\log_2 7 - \epsilon})$, kde platí pro $\epsilon < \log_2 7 - \log_2 4, \epsilon > 0$

Proto můžeme říct, že $T(n) \in \Theta(n^{\log_2 7}) \approx \Theta(n^{2.8\dots})$, tedy je to lepší než $\Theta(n^3)$

Příklad: spočítejte množnost algoritmu: $a, b > 0$

while $a > 0$:

 if $a < b$:

$$(a, b) := (2a, b-a)$$

 else:

$$(a, b) := (a-b, 2b)$$

return b

Ukusíme pro par vstupy:

$$\begin{array}{rcl} 1 & 1 & \rightarrow 2 & 3 \\ 0 & 2 & \rightarrow 2 & \\ & & \left[\begin{array}{c} 4 \\ 1 \\ 3 \\ 2 \\ 1 \\ 4 \\ 2 \\ 3 \end{array} \right] & \end{array}$$

Algoritmus musí skončit pro všechny vstupy, když není správný algoritmus, složitost hledat nebudeme.

Příklad: Je dáno $i \in \mathbb{N}$, spočítejte množnost algoritmu:

while $i > 0$:

 for ($j := 0, j \cdot j < i, j++$):

 pass

$i := i - 2$

return

Algoritmus začne s koncovým i , které se posypne smíruje o 2, nější cyklus tedy vždy skončí.

Triviální cyklus nemá na výsledek vliv, v každém kroku počítá na předešlou cosi o množnosti $T(i)$

$$g(n) = \sqrt{i} + \sqrt{i-2} + \sqrt{i-4} + \dots = \sum_{k=0}^{i/2} \sqrt{i-2k} = \sum_{k=1}^{i/2} \sqrt{2k} \leq \frac{i}{2} \cdot \sqrt{i} = \frac{i^{3/2}}{2} \approx i^{3/2}$$

Tento algoritmu můžeme říct, že $T(n) \in \Theta(i^{3/2})$

Odhad můžeme rískat i pomocím voreni a přednášky: Pohud $f\left(\frac{n}{2}\right) \in \Theta(f(n))$, $f(n)$ je nezáporná a rostoucí, pak $\sum_{i=1}^n f(i) \in \Theta(n \cdot f(n))$.

- nezáporná a neklesající ✓

- $f(n/2) \in \Theta(f(n))$: $\sqrt{n/2} = \sqrt{n/2} \cdot \sqrt{n}$, tedy taky platímo ✓

$$\text{Proto } \sum_{k=1}^{i/2} \sqrt{2k} \in \Theta\left(\frac{i}{2} \cdot \sqrt{i/2}\right) = \Theta\left(\frac{i \cdot \sqrt{i}}{2 \cdot \sqrt{2}}\right) = \Theta(i^{3/2})$$

Pr: Ypocítaje složitost algoritmu, $N \in \mathbb{N}$, $N = 3^k$ pro $k \in \mathbb{N}$. (k nenechte)

$i := N (=n)$

Algoritmus skončí, protože i se pořád směruje.

while $i > 0$:

$j := 1$

$i := i/3$

while $j < i$:

$j++$

return

Vnější cyklus proběhne pro $i = n, n/3, n/9, \dots, n/3^k$, kde $k = \log_3 n$, tedy celkem $(k+1)$ -krát.

Vnitřní cyklus taky proběhne $(k+1)$ -krát, a pokudé bude trvat krabí a krabí dobu: $n, n/3, n/9, \dots, n/3^k = 1$

$$g(n) = \sum_{i=1}^{\log_3 N} \frac{n}{3^k} = n \cdot \sum_{i=1}^{\log_3 N} \frac{1}{3^k} \leq n \cdot \sum_{i=1}^{\infty} \frac{1}{3^k} = n \cdot \left[\frac{\frac{1}{3}}{1 - \frac{1}{3}} \right] = n \cdot \frac{1}{2} = \frac{n}{2}, g(n) \in \Theta(n)$$

Pr: Ypocítaje složitost algoritmu, $N \in \mathbb{N}$

while $N > 1$:

Klikneme pro $n = 4$: 4, 5, 3, 2, 3

if (N je liché):

For($i=0, i < N, i++$): Algoritmu neskončí, nemá smysl hledat složitost.

print('*')

$N := (N+1)/2$

else:

$N++$

return

Sčítání proběhne v konstantním čase, stejně jako našebeni mocninou dvojkdy

Pr: Máme binární čísla a, b a chceme sčítat jejich součin. Ypocítaje složitosti:

1) použijeme naivní algoritmus: $a \cdot b = \sum_{i=1}^{\infty} b$

$m := 0$

For($i=0, i < \omega, i++$):

$m += b$

return m

$\Theta(a) = \Theta(2^n)$ délka a binárně

2) použijeme rekurenci, libovolné $2N$ čísla napsáme jako $2^N A + B$, kde A, B jsou N -cifrové. Našebek dvou řad pak je:

$$(A \cdot N) = (2^N A + B) \cdot (2^N C + D) = 2^{2N} A \cdot C + 2^N (AD + BC) + BD$$

$$T(n) = 4 \cdot T\left(\frac{n}{2}\right) + 2, a = 4, b = 2, \log_b a = 2$$

Použijeme MT 1, protože $2 \in \Theta(n^{2-\varepsilon})$, $\varepsilon = 1$, proto $T(n) \in \Theta(n^2)$