



Algoritmizace

Dynamické programování

Jiří Vyskočil, Marko Genyg-Berezovskyj

2010

Rozděl a panuj (divide-and-conquer)

- **Rozděl (Divide):** Rozděl problém na několik podproblémů tak, aby tyto podproblémy odpovídaly původnímu problému, ale měly menší velikost.
- **Panuj (Conquer):** Rekurzivně vyřeš tyto podproblémy. Jestliže je podproblém dostatečně malý, vyřeš ho přímo (bez rekurze).
- **Kombinuj (Combine):** Zkombinuj řešení podproblémů do řešení výsledného původního problému.

Dynamické programování

- Řeší problémy podobně jako metoda rozděl a panuj (divide-and-conquer) kombinováním řešení podproblémů.
- Slovo programování zde neznamena psaní kódu, ale použití určité metody pro řešení (podobně jako lineární programování, atd).
- Metoda rozděl a panuj typicky dělí problém rekurzivně na **nezávislé** podproblémy.
- Naproti tomu dynamické programování se používá tam, kde podproblémy nejsou nezávislé (tj. **podproblémy sdílejí společné podpodproblémy**). Metoda rozděl a panuj by v takovém případě opakovaně řešila stejné společné podpodproblémy, čímž by vykonávala zbytečnou práci navíc.
- Dynamické programování řeší každý podpodproblém pouze jednou a jeho řešení si pamatuje v tabulce. Tím zamezí opakovanému řešení stejných podproblémů.

Dynamické programování

- Popis vývoje algoritmu dynamického programování:
 - 1) Charakterizace struktury optimálního řešení.
 - 2) Nalezení rekurzivní definice pro výpočet hodnoty optimálního řešení.
 - 3) Nalezení výpočtu hodnoty optimálního řešení zdola nahoru (od nejjednodušších podproblémů ke složitějším).
 - 4) Nalezení konstrukce optimálního řešení z vypočtených informací (tento krok může být vynechán pokud je hodnota optimálního řešení přímo vlastní optimální řešení).

Příklad: Nejdelší společná podposloupnost

■ Motivace

- V biologických aplikacích chceme často porovnávat řetězce DNA různých organismů a zjišťovat jejich podobnost.
- Řetězec DNA je posloupnost nukleotidů, kde nukleotid je buď (adenine A, guanine G, cytosine C, thymine T).
- Podobnost dvou řetězců DNA určuje délka jejich nejdelší společné podposloupnosti (dále už jen NSP).
- Podposloupnost je *společná* několika řetězcům, pokud je podposloupností každého řetězce jednotlivě.
- Mějme řetězec $X = \langle x_1, x_2, \dots, x_m \rangle$. Řetězec $Z = \langle z_1, z_2, \dots, z_k \rangle$ je *podposloupnost* X pokud existuje rostoucí posloupnost indexů $\langle i_1, i_2, \dots, i_k \rangle$ z X taková, že pro všechny $j = 1, 2, \dots, k$, platí $x_{i_j} = z_j$.
Příklad: $Z = \langle G, C, T, G \rangle$ je podposloupnost $X = \langle A, G, C, G, T, A, G \rangle$ s korespondující posloupností indexů $\langle 2, 3, 5, 7 \rangle$.

- Úloha: Najděte nejdelší společnou podposloupnost pro dva zadané řetězce DNA.

Příklad: Nejdelší společná podposloupnost

- 1) Charakterizace struktury optimálního řešení.
 - Pokud bychom chtěli najít podposloupnost “hrubou” silou.
 - To znamená otestovat všechny podmnožiny indexů $\{1, 2, \dots, m\}$ menšího ze vstupních řetězců DNA (m je jeho délka).
 - Takových podmnožin je ale 2^m . Složitost tohoto přístupu by tedy byla exponenciální.
 - Pro reálné DNA řetězce s alespoň tisíčovkami nukleotidů by tedy byla tato metoda zcela nepoužitelná.
 - Při detailnější analýze problému snadno zjistíme, že pro dva vstupní řetězce $X = \langle x_1, x_2, \dots, x_m \rangle$ a $Y = \langle y_1, y_2, \dots, y_n \rangle$ a hledanou výstupní podposloupnost $Z = \langle z_1, z_2, \dots, z_k \rangle$ platí:
 - Pokud $x_m = y_n$, potom $z_k = x_m = y_n$ a Z_{k-1} je NSP X_{m-1} a Y_{n-1} .
 - Pokud $x_m \neq y_n$, potom $z_k \neq x_m$ implikuje, že Z je NSP X_{m-1} a Y .
 - Pokud $x_m \neq y_n$, potom $z_k \neq y_n$ implikuje, že Z je NSP X a Y_{n-1} .

Příklad: Nejdelší společná podposloupnost

- 2) Nalezení rekurzivní definice pro výpočet hodnoty optimálního řešení.
- Z předchozí analýzy vyplývá, že musíme umět řešit podúlohu pro postupně se odzadu zkracující vstupní řetězce.
 - Nyní zavedeme pole $c[i,j]$ jako délku NSP pro prefixy vstupních řetězců X_i a Y_j .
 - Z předchozí analýzy si můžeme pole c definovat rekurzivně:

$$c[i, j] = \begin{cases} 0 & \text{pokud } i = 0 \text{ nebo } j = 0, \\ c[i - 1, j - 1] + 1 & \text{pokud } i, j > 0 \text{ a } x_i = y_j, \\ \max\{c[i, j - 1], c[i - 1, j]\} & \text{pokud } i, j > 0 \text{ a } x_i \neq y_j. \end{cases}$$

Příklad: Nejdelší společná podposloupnost

3) Nalezení výpočtu hodnoty optimálního řešení zdola nahoru.

Function DÉLKA-NSP(X, Y)

- 1) $m \leftarrow \text{length}[X];$
- 2) $n \leftarrow \text{length}[Y];$
- 3) **for** $i \leftarrow 1$ **to** m **do**
- 4) $c[i, 0] \leftarrow 0;$
- 5) **for** $j \leftarrow 0$ **to** n **do**
- 6) $c[0, j] \leftarrow 0;$
- 7) **for** $i \leftarrow 1$ **to** m **do**
- 8) **for** $j \leftarrow 1$ **to** n **do**
- 9) **if** $x_i = y_j$
- 10) **then** { $c[i, j] \leftarrow c[i-1, j-1] + 1;$
 $b[i, j] \leftarrow "\uparrow";$ }
- 11) **else if** $c[i-1, j] \geq c[i, j-1];$
- 12) **then** { $c[i, j] \leftarrow c[i-1, j];$
 $b[i, j] \leftarrow "\uparrow";$ }
- 13) **else** { $c[i, j] \leftarrow c[i, j-1];$
 $b[i, j] \leftarrow "\leftarrow";$ }
- 14) **else** { $c[i, j] \leftarrow c[i, j-1];$
 $b[i, j] \leftarrow "\leftarrow";$ }
- 15) **else** { $c[i, j] \leftarrow c[i, j-1];$
 $b[i, j] \leftarrow "\leftarrow";$ }
- 16) $b[i, j] \leftarrow "\leftarrow";$ }
- 17) **return** c and $b;$

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

Příklad: Nejdelší společná podposloupnost

- 4) Nalezení konstrukce optimálního řešení z vypočtených informací.

Procedure TISK-NSP(b, X, i, j)

- 1) **if** ($i = 0$) **or** ($j = 0$) **then exit**;
- 2) **if** $b[i, j] = "\nwarrow"$ **then** { TISK-NSP($b, X, i - 1, j - 1$);
- 3) **print** x_i ; }
- 4) **else if** $b[i, j] = "\uparrow"$ **then** TISK-NSP($b, X, i - 1, j$);
- 5) **else** TISK-NSP($b, X, i, j - 1$);

Příklad: Nejdelší společná podposloupnost

- Asymptotická složitost algoritmu:
 - Časová: $\Theta(mn) + O(m+n) = \Theta(mn)$
 - Paměťová: $\Theta(mn)$
- Možná vylepšení:
 - Pomocné pole b lze nahradit testem v poli c v konstantním čase.
 - Nepotřebujeme celé pole c , ale pouze stačí v paměti udržovat předposlední a poslední řádek a sloupec. (Pak ale nebudeme schopni v čase $O(m+n)$ rekonstruovat hledanou podposloupnost.)

Příklad: Optimální vyhledávací strom

■ Motivace:

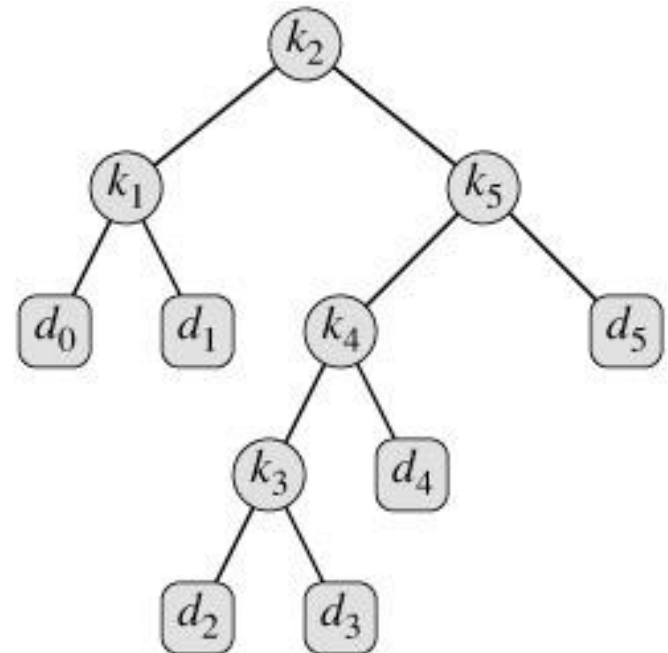
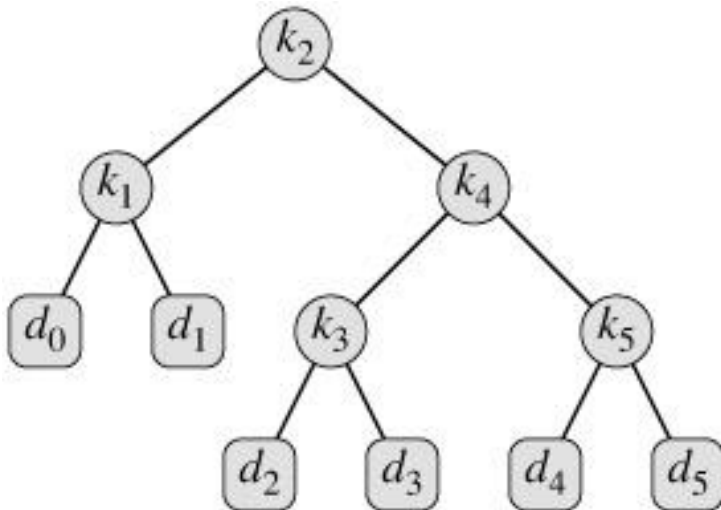
- Máme vytvořit překlad textu (posloupnosti slov) z angličtiny do češtiny.
- Přitom víme, že jak se jednotlivá anglická slova často v textu vyskytují.
- Kdybychom sestavili pouze obyčejný vyhledávací strom s hloubkou $O(\log n)$, mohlo by se stát, že slovo jako "mycophagist,, s velmi malou pravděpodobností výskytu by se dostalo do kořene takového stromu.
- Naopak slova s vysokou pravděpodobností výskytu jako „the“ by měli být blízko u kořene.

- Úloha: Sestavte optimální binární vyhledávací strom (BVS), když je dána množina klíčů $K = \langle k_1, k_2, \dots, k_n \rangle$ (tak, že $k_1 < k_2 < \dots < k_n$), ke každému klíči k_i známe pravděpodobnost výskytu p_i , navíc máme seznam prázdných klíčů $d_0, d_1, d_2, \dots, d_n$, které reprezentují intervaly mezi klíči, které nejsou ve slovníku K , ke každému prázdnému klíči d_j známe pravděpodobnost výskytu q_j .

Navíc platí
$$\sum_{i=1}^n p_i + \sum_{i=0}^n q_i = 1 .$$

Příklad: Optimální vyhledávací strom

- Příklad dvou vyhledávacích stromů s pěti klíči.



Příklad: Optimální vyhledávací strom

1) Charakterizace struktury optimálního řešení.

$$\begin{aligned} \text{celková cena vyhledávání v } T &= \sum_{i=1}^n (\text{hloubka}_T(k_i) + 1) \cdot p_i + \sum_{i=0}^n (\text{hloubka}_T(d_i) + 1) \cdot q_i \\ &= 1 + \sum_{i=1}^n \text{hloubka}_T(k_i) \cdot p_i + \sum_{i=0}^n \text{hloubka}_T(d_i) \cdot q_i \end{aligned}$$

- Hledáme binární vyhledávací strom jehož *celková cena vyhledávání* (tj. cena přes všechny dotazy na úspěšné i neúspěšné vyhledání slova) bude minimální.

Příklad: Optimální vyhledávací strom

- 2) Nalezení rekurzivní definice pro výpočet hodnoty optimálního řešení.
- Chceme umět řešit podúlohu pro klíče k_i, \dots, k_j , kde $i \geq 1, j \leq n$ a $j \geq i - 1$. (Pokud $j = i - 1$, potom použijeme prázdný klíč d_{i-1}).
 - Očekávanou cenu vyhledávání ve stromu e si definujeme rekurzivně

$$e[i, j] = \begin{cases} q_{i-1} & \text{pokud } j = i - 1, \\ \min_{i \leq r \leq j} \{e[i, r - 1] + e[r + 1, j] + w(i, j)\} & \text{pokud } i \leq j. \end{cases}$$

$$w(i, j) = \begin{cases} q_{i-1} & \text{pokud } j = i - 1, \\ w(i, j - 1) + p_j + q_j & \text{pokud } i \leq j. \end{cases}$$

- $w(i, j)$ je pravděpodobnost výskytu podstromu i, j .

tedy

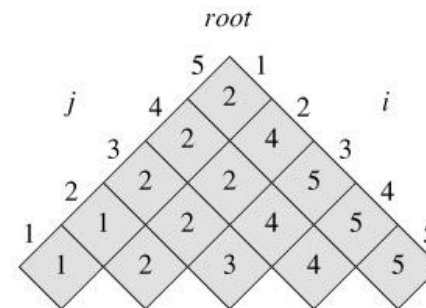
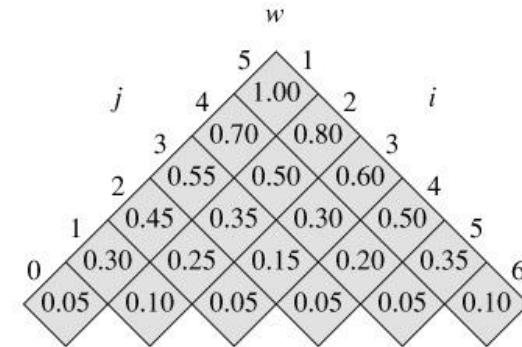
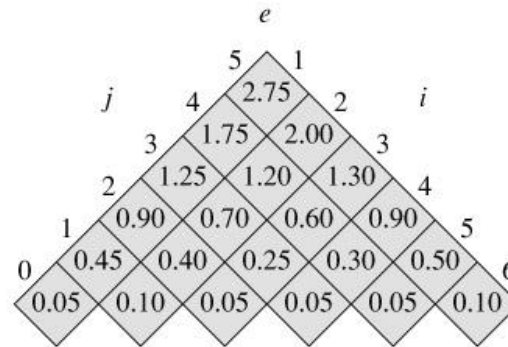
$$w(i, j) = \sum_{l=i}^j p_l + \sum_{l=i-1}^j q_l.$$

Příklad: Optimální vyhledávací strom

3) Nalezení výpočtu hodnoty optimálního řešení zdola nahoru.

Function OPTIMÁLNÍ-BVS(p, q, n)

- 1) **for** $i \leftarrow 1$ **to** $n + 1$ **do** {
- 2) $e[i, i - 1] \leftarrow q_{i-1}$;
- 3) $w[i, i - 1] \leftarrow q_{i-1}$; }
- 4) **for** $l \leftarrow 1$ **to** n **do**
- 5) **for** $i \leftarrow 1$ **to** $n - l + 1$ **do** {
- 6) $j \leftarrow i + l - 1$;
- 7) $e[i, j] \leftarrow \infty$;
- 8) $w[i, j] \leftarrow w[i, j - 1] + p_j + q_j$;
- 9) **for** $r \leftarrow i$ **to** j **do** {
- 10) $t \leftarrow e[i, r - 1] + e[r + 1, j] + w[i, j]$;
- 11) **if** $t < e[i, j]$
- 12) **then** { $e[i, j] \leftarrow t$;
- 13) $root[i, j] \leftarrow r$; } }
- 14) **return** e **and** $root$



Příklad: Optimální vyhledávací strom

- Asymptotická složitost algoritmu:
 - Časová: $O(n^3)$ a dokonce $\Omega(n^3) \Rightarrow \Theta(n^3)$
 - Paměťová: $\Theta(n^2)$