Dynamické programování

prof. Ing. Pavel Tvrdík CSc.

Katedra počítačových systémů FIT
České vysoké učení technické v Praze

DSA, ZS 2009/10, Předn. 12

http://service.felk.cvut.cz/courses/X36DSA/

Rozděl-a-panuj vs. dynamické programování

- Metody **rozděl-a-panuj** je vhodné pro řešení úloh, které lze rozdělit na **nezávislé podúlohy**:
  - Rozděl úlohu na nezávislé podúlohy.
  - Rekurzivně vyřeš podúlohy.
  - Zkombinuj řešení podúloh do celkového výsledku úlohy.
- **Dynamické programování** (DP) je vhodné pro řešení úloh, ve kterých se podúlohy **překrývají**, t.j., **řeší se stejné podpodúlohy**.
- Pro takové úlohy nejsou metody **rozděl-a-panuj** vhodné, protože sdílené podúlohy se **opakovaně** (**redundantně, zbytečně**) řeší znovu a znovu, což může vést na **řádově vyšší složitost**.
- Metody DP si hodnoty dříve vyřešených podúloh zapamatují a při jejich znovuobjevení je využijí.

Význam termínu DP

- Termín "programování" v "dynamickém programování" nemá standardní význam vytváření počítačového kódu.
- Je odvozen z termínu "matematické programování", což je synonymum pro optimalizaci.
- V této interpretaci, slovo "program" znamená optimální či přijatelný plán (rozpis, rozvrh) pro provedení nějaké množiny souvisejících úkonů.
- Programování v tomto smyslu tedy znamená metodu nalezení takového plánu či rozvrhu akcí.

Dynamické programování pro řešení optimalizačních úloh

Definice

Optimalizační úloha: *Z více možných řešení hledáme řešení s* **optimální cenou** *(maximální či minimální hodnotou, . . .).*

Vytvoření DP algoritmu pro řešení dané optimalizační úlohy se skládá ze 4 **kroků**:

1. Charakterizuj **strukturu** optimálního řešení.
2. Rekurzivně definuj **hodnotu** optimálního řešení.
3. Vypočítej efektivně **hodnotu** optimálního řešení:
   1. metodou shora dolů.
   2. metodou zdola nahoru.
4. (Zrekonstruuj **strukturu** optimálního řešení z vypočtených hodnot.)

Poslední bod odpadá, pokud stačí pouze cena (hodnota) optimálního řešení.

Řetězové násobení matic

Definice

Je dána posloupnost matic $A_1, A_2, \ldots, A_n$, matice A_i má rozměry $d_{i-1} \times d_i$. **Řetězové násobení matic** *(ŘNM) je úkol vypočítat* **součin** $A = A_1 A_2 \cdots A_n$ *s* **minimální aritmetickou složitostí** *(= cena řešení). Jinými slovy, úkolem je najít takové* **uzávorkování pořadí** *násobení matic, které vede na vynásobení s nejmenším počtem aritmetických operací. Předpokládáme klasické násobení matic "řádky krát sloupce". Složitost součinu $A_i A_{i+1}$ aproximujeme výrazem $d_{i-1} \cdot d_i \cdot d_{i+1}$ (zanedbáváme konstanty a nižší členy).*

Vlastnosti ŘNM

Poznámky:

- Násobení matic je **asociativní**, proto **jakékoli** uzávorkování dá správný výsledek A.
- Různé pořadí násobení má **dramatický** vliv na aritmetickou složitost výpočtu.

Příklad

Uvažujme $n = 3$ a $d_0 = 5$, $d_1 = 50$, $d_2 = 10$ a $d_3 = 30$. Pak

- uzávorkování $(A_1 A_2)A_3$ má aritmetickou složitost $5 \cdot 50 \cdot 10 + 5 \cdot 10 \cdot 30 = 2500 + 1500 = 4000$,
- kdežto uzávorkování $A_1(A_2 A_3)$ má složitost $50 \cdot 10 \cdot 30 + 5 \cdot 50 \cdot 30 = 15000 + 7500 = 22500$!!!

- Přitom kombinatorická složitost ŘNM je **exponenciální**, takže řešení hrubou silou, zkoušením všech možností, není myslitelná.

Počet různých uzávorkování

- Označme $Z(n)$ počet všech různých způsobů, jak uzávorkovat součin $A_1 A_2 \cdots A_n$.
- Posloupnost matic můžeme rozdělit na dvě části **hranicí** mezi A_k a A_{k+1} pro libovolné $k = 1, 2, \ldots, n-1$ a pak uzávorkovat **rekurzívně** a **nezávisle** na sobě obě vzniklé podposloupnosti $A_1, \ldots, A_k$ a $A_{k+1}, \ldots, A_n$.
- Proto platí rekurentní vztah

$$Z(n) = \begin{cases} 1 & \text{if } n = 1, \\ \sum_{k=1}^{n-1} Z(k) Z(n-k) & \text{if } n \geq 2. \end{cases}$$

- Řešením jsou tzv. **Catalanova** čísla: $Z(n) = C(n-1)$, viz `http://mathworld.wolfram.com/CatalanNumbers.html`, o kterých se ví, že

$$C(n) = \frac{1}{n+1} \binom{2n}{n} = \Omega\left(4^n / n^{\frac{3}{2}}\right).$$

První krok algoritmu DP: Charakterizování optimálního závorkování

- Označme $A_{i\ldots j}$ matici vzniklou vynásobením $A_i A_{i+1} \ldots A_j$, $i \leq j$.
- Pro každé optimální závorkování existuje **hranice** k, $1 \leq k < n$, taková, že rekurzivně se zkonstruuje závorkování pro výpočet $A_{1\ldots k}$, pak závorkování pro výpočet $A_{k+1\ldots n}$, a výslednou $A = A_{1\ldots n}$ získáme vynásobením $A_{1\ldots k} A_{k+1\ldots n}$.
- Aritmetická složitost ŘNM je tedy složitost výpočtu $A_{1\ldots k}$ + složitost výpočtu $A_{k+1\ldots n}$ + složitost násobení 2 matic $A_{1\ldots k} A_{k+1\ldots n}$.

Lemma

Nechť k je hranice optimálního závorkování (s minimální aritmetickou složitostí). Pak závorkování pro výpočet $A_{1\ldots k}$ i závorkování pro výpočet $A_{k+1\ldots n}$ musejí být optimální.

- **Důkaz.** Důkaz sporem. Pokud by např. první nebylo, existovalo by závorkování pro $A_{1\ldots k}$ s nižší cenou a tím by i výsledná cena byla nižší, což je spor, protože dané závorkování pro $A_{1\ldots n}$ je optimální. ■

Druhý krok algoritmu DP: Rekurzivní definice ceny optimálního závorkování

- Nechť $m[i,j]$ je minimální **cena závorkování** (=nejmenší aritmetická složitost násobení) $A_iA_{i+1}\ldots A_j$, $i \leq j$. Pak platí:

$$m[i,j] = \begin{cases} 0 & \text{if } i = j, \\ \min_{i \leq k < j}\{m[i,k] + m[k+1,j] + d_{i-1}d_kd_j\} & \text{if } i < j. \end{cases} \quad (1)$$

- Označme $h[i,j]$ právě takové k, které v (1) dává minimum, čili

$$m[i,j] = m[i,h[i,j]] + m[h[i,j]+1,j] + d_{i-1}d_{h[i,j]}d_j.$$

- Pak cena optimálního závorkování je

$$m[1,n] = m[1,h[1,n]] + m[h[1,n]+1,n] + d_0d_{h[1,n]}d_n.$$

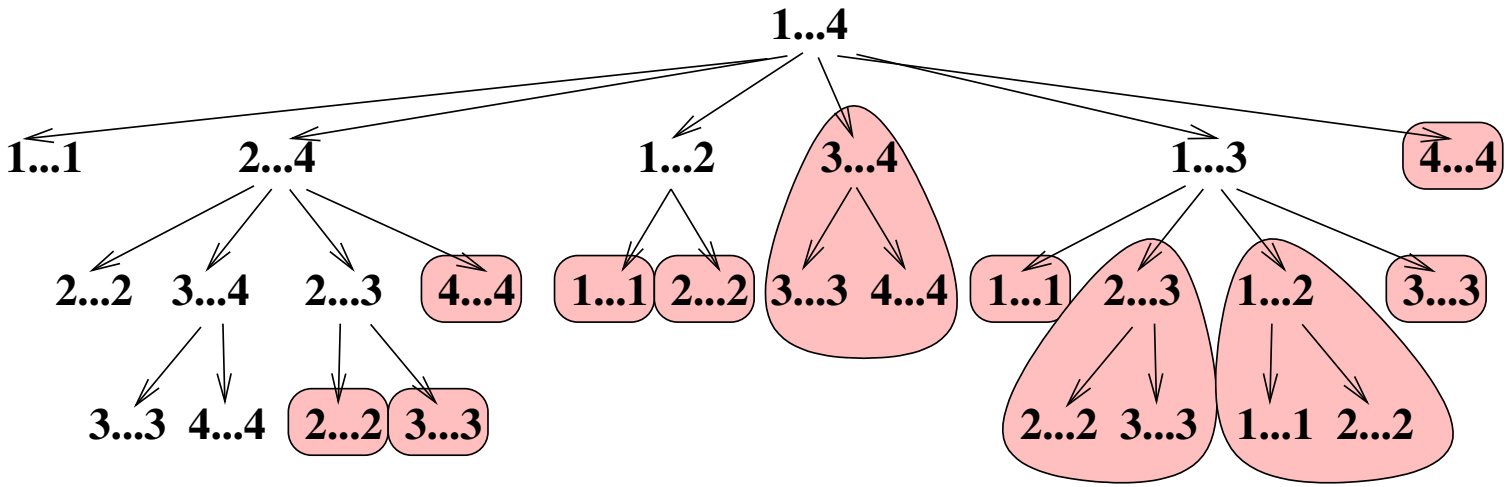
Třetí krok algoritmu DP: Algoritmus výpočtu ceny optimálního závorkování

Přímý rekurzivní algoritmus $\textsc{RecRNM}(d, 1, n)$ realizující výpočet $m[1, n]$ podle (1) nelze použít, protože má **exponenciální složitost**!!!

procedure $\textsc{RecRNM}(d, i, j)$
{
(1) **if** $(i = j)$ **then return**(0);
(2) $m[i, j] \leftarrow +\infty$;
(3) **for** $(k \leftarrow i$ **to** $j - 1)$
(4) **do** $\{q \leftarrow \textsc{RecRNM}(d, i, k) + \textsc{RecRNM}(d, k + 1, j)$
(5) $+d[i - 1] \cdot d[k] \cdot d[j]$;
(6) **if** $(q < m[i, j])$ **then** $\{ m[i, j] \leftarrow q;\ h[i, j] \leftarrow k\}\}$;
(7) **return**$(m[i, j], h[i, j])$;
}

Strom rekurzivních volání (SRV) RecRNM

- Důvod exponenciální složitosti je podobný jako u rekurzivního výpočtu Fibonacciho čísel (viz Slajdy 25 až 33 v Přednášce 8).
  - Celkový počet hodnot $m[*,*]$, které je třeba vypočítat pro zjištění hodnoty $m[1,n]$, je pouze počet všech dvojic i,j takových, že $1 \leq i \leq j \leq n$, což je $\binom{n}{2} + n = \binom{n+1}{2} \doteq \frac{n^2}{2}$.
  - Dochází však **masivně** k **opakovenému** výpočtu stejných hodnot, viz příklad stromu rekur. volání $\mathrm{RecRNM}(1,4)$.



Časová složitost $\textsc{RecRNM}(d, 1, n)$

Lemma

Časová složitost $t(n)$ *běhu* $\textsc{RecRNM}(d, 1, n)$ *je* $\Omega(2^{n-1})$.

Důkaz. (Substituční metodou.)

- Předpokládejme, že kód na řádku (1) a (6) trvá čas aspoň 1.
- Pak triviálně $t(1) \geq 1$.
- Dále pro $n > 1$ je $t(n) \geq 1 + \sum_{k=1}^{n-1}(t(k) + t(n-k) + 1)$.
- Díky symetrii sčítanců to je totéž jako $t(n) \geq n + 2\sum_{i=1}^{n-1} t(i)$.
- Ukažme indukcí, že $t(n) \geq 2^{n-1}$.
  - Indukční základ: $T(1) \geq 1 = 2^0$ triviálně z předpokladů.
  - Indukční krok pro $n \geq 2$:
    $t(n) \geq n + 2\sum_{i=1}^{n-1} 2^{i-1} = n + 2(2^{n-1} - 1) = n + 2^n - 2 \geq 2^{n-1}$. ■

Třetí krok DP algoritmu: Efektivní varianta 1. Nerekurzivní algoritmus výpočtu ceny optim. závorkování zdola nahoru

Předpoklady:

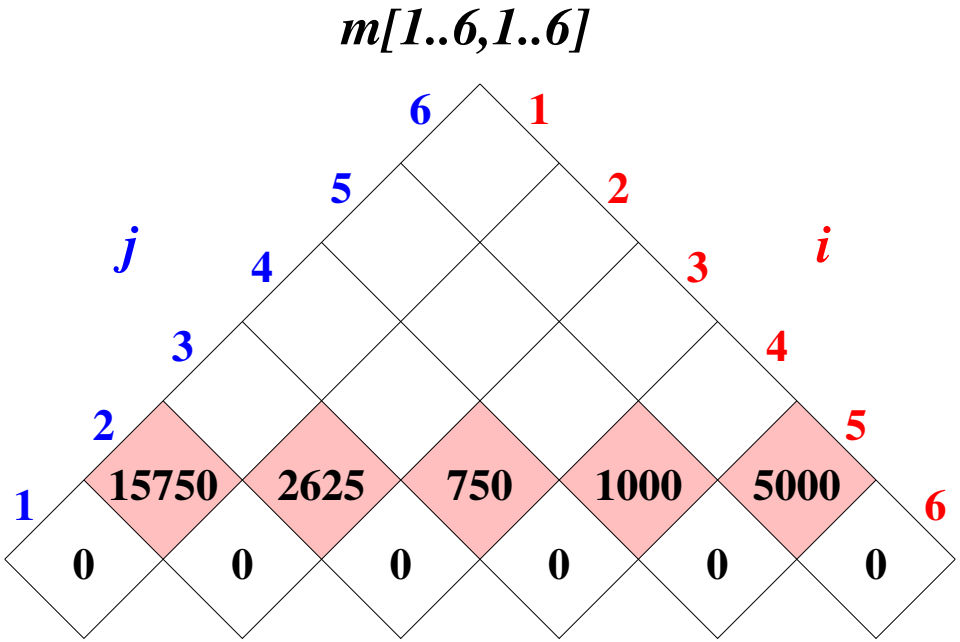
- Vstupem je pole $d[0, \ldots, n]$ rozměrů matic.
- Výstupem jsou 2 pole:
  - $m[1, \ldots, n, 1, \ldots, n]$, kam se ukládají ceny opt. závorkování pro výpočet $A_{i \ldots j}$,
  - $h[1 \ldots n-1, 2 \ldots n]$, kam se ukládají příslušné hodnoty hranic optimálního závorkování.

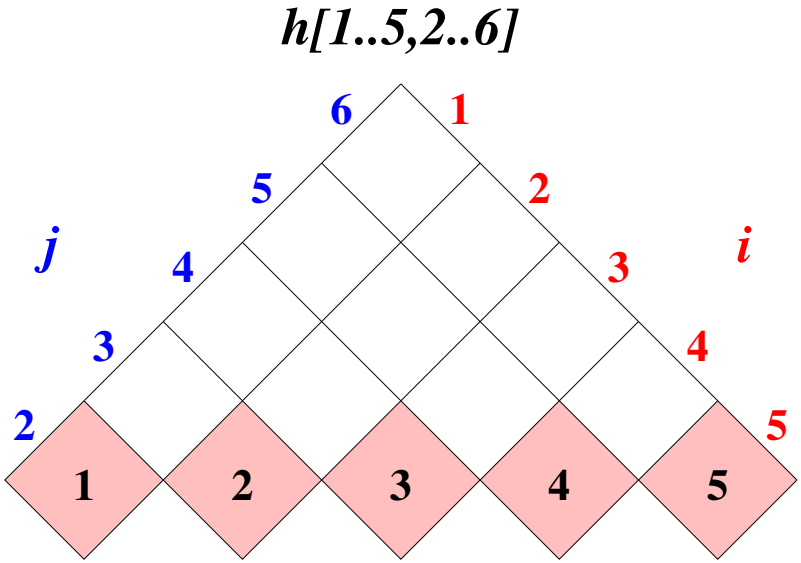
Princip:

Systematické zaplňování tabulky zdola nahoru pro všechny dvojice i, j, kde $1 \leq i \leq j \leq n$ ($\binom{n+1}{2}$ hodnot).

```
procedure BottomUpRNM(n, d, m, h)
{
(1)for (i ← 1 to n) do m[i, i] ← 0;
(2)for (l ← 2 to n)
(3)      for (i ← 1 to n − l + 1)
(4)           do { j ← i + l − 1;
(5)                m[i, j] ← +∞;
(6)                for (k ← i to j − 1)
(7)                     do { q ← m[i, k] + m[k + 1, j] + d[i − 1] · d[k] · d[j];
(8)                          if (q < m[i, j])
(9)                          then { m[i, j] ← q;
(10)                                h[i, j] ← k};
(11)               }       };
(12)return(m[1 . . . n, 1 . . . n], h[1 . . . n − 1, 2 . . . n]);
}
```

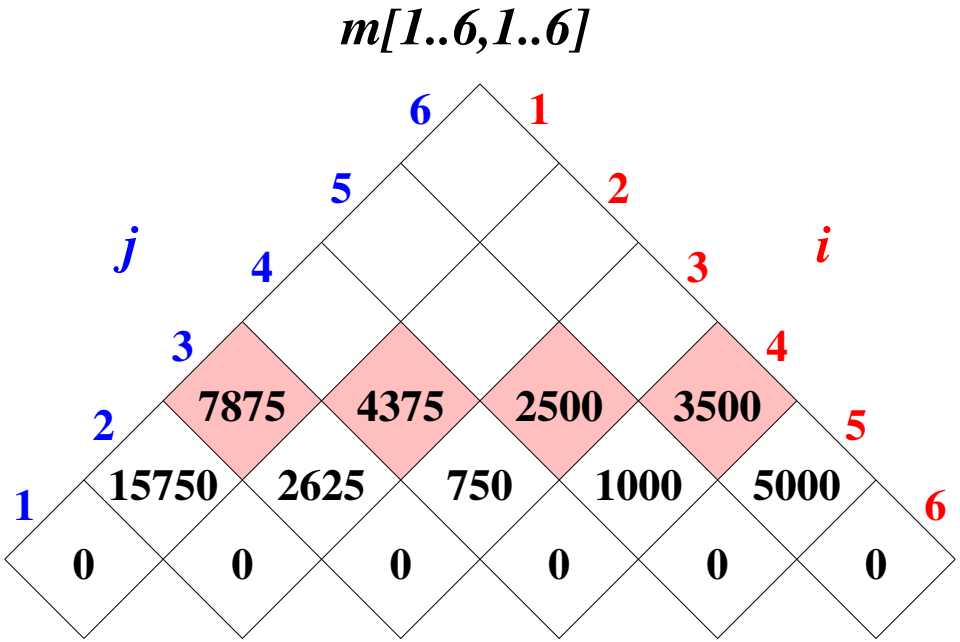
Příklad běhu BottomUpRNM$(6, d, m, h)$

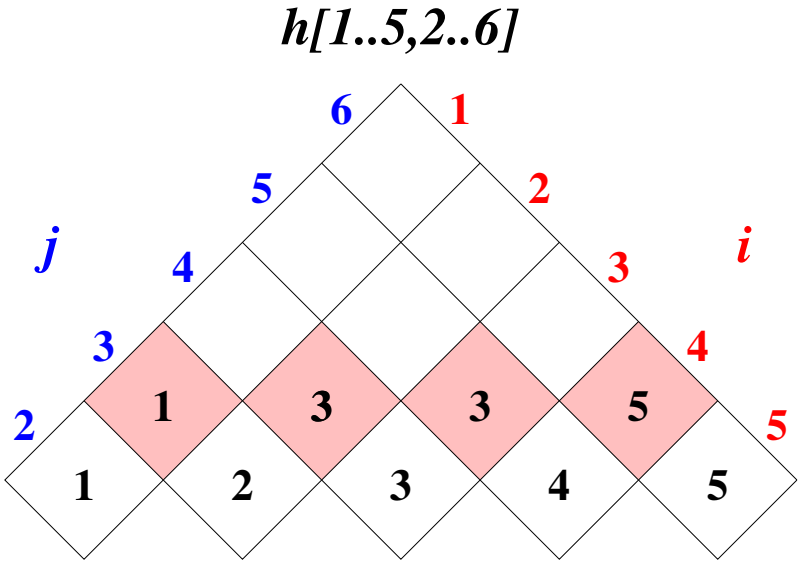




A_1	A_2	A_3	A_4	A_5	A_6
30×35	35×15	15×5	5×10	10×20	20×25

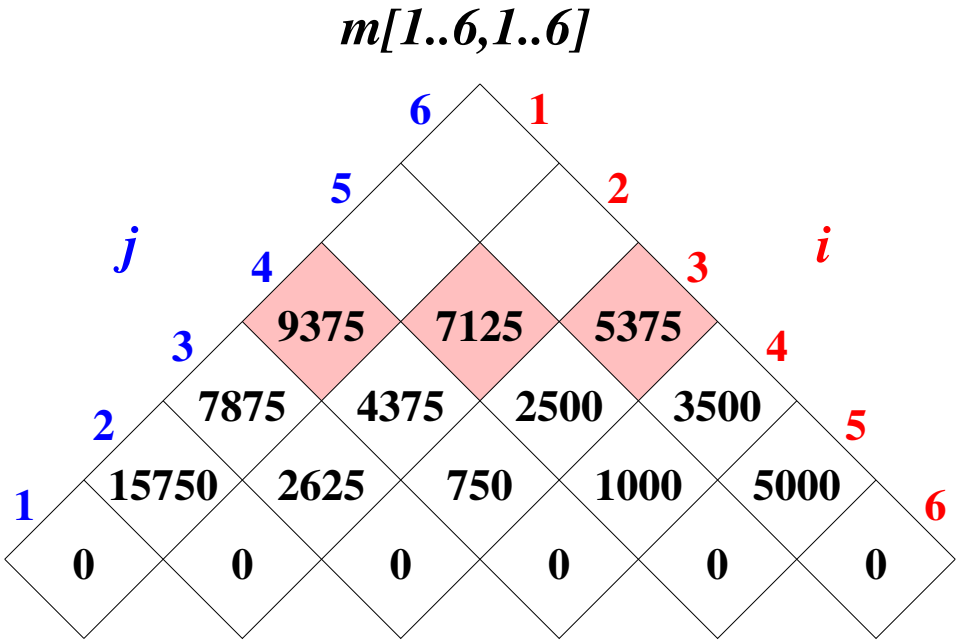
Příklad běhu BottomUpRNM$(6, d, m, h)$

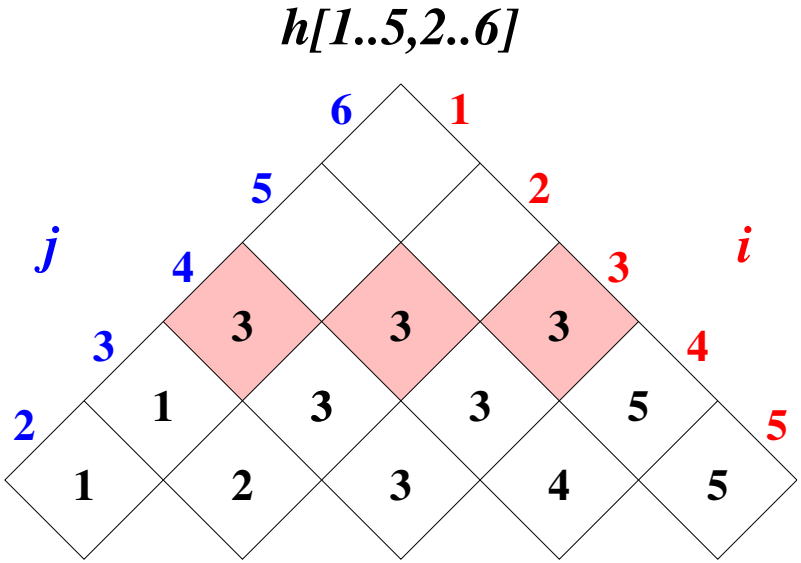




A_1	A_2	A_3	A_4	A_5	A_6
30×35	35×15	15×5	5×10	10×20	20×25

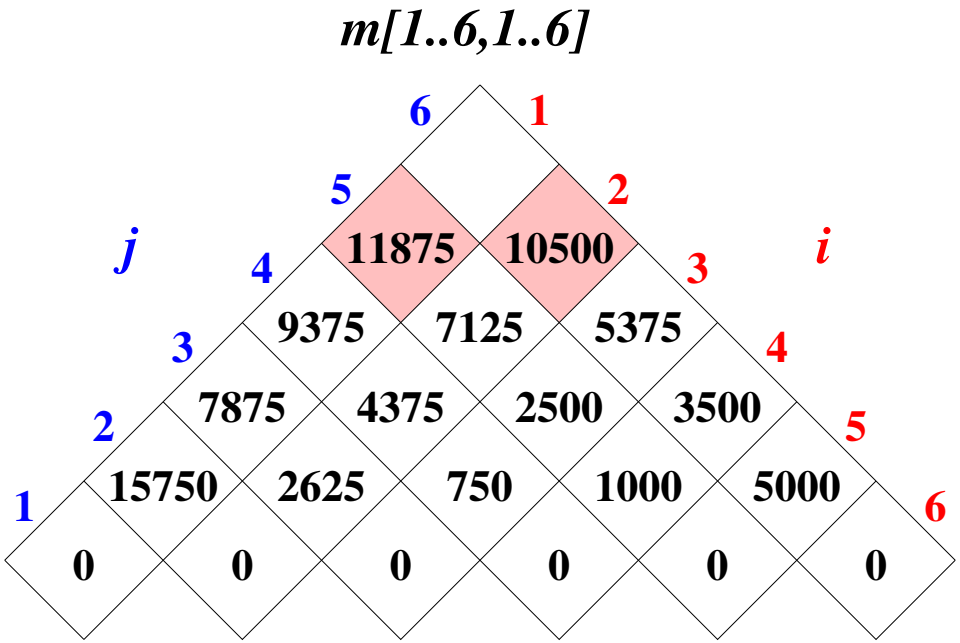
Příklad běhu BottomUpRNM$(6, d, m, h)$

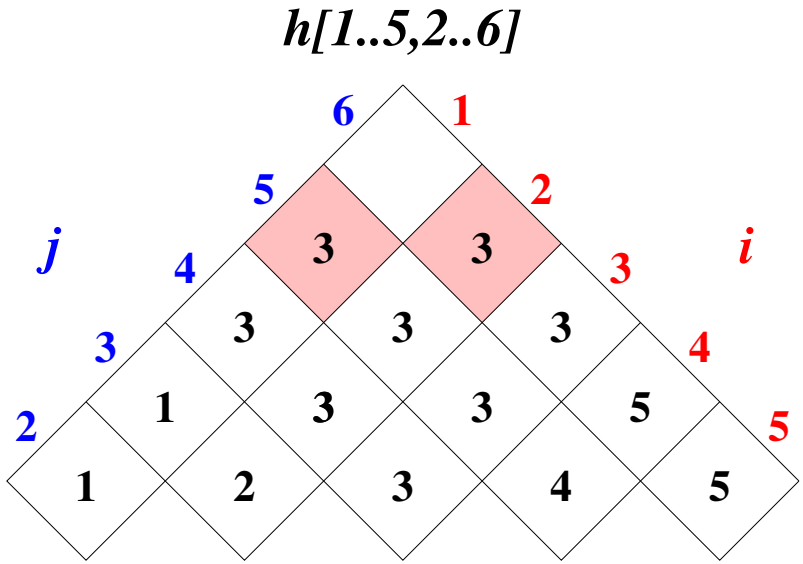




A_1	A_2	A_3	A_4	A_5	A_6
30×35	35×15	15×5	5×10	10×20	20×25

Příklad běhu BottomUpRNM$(6, d, m, h)$

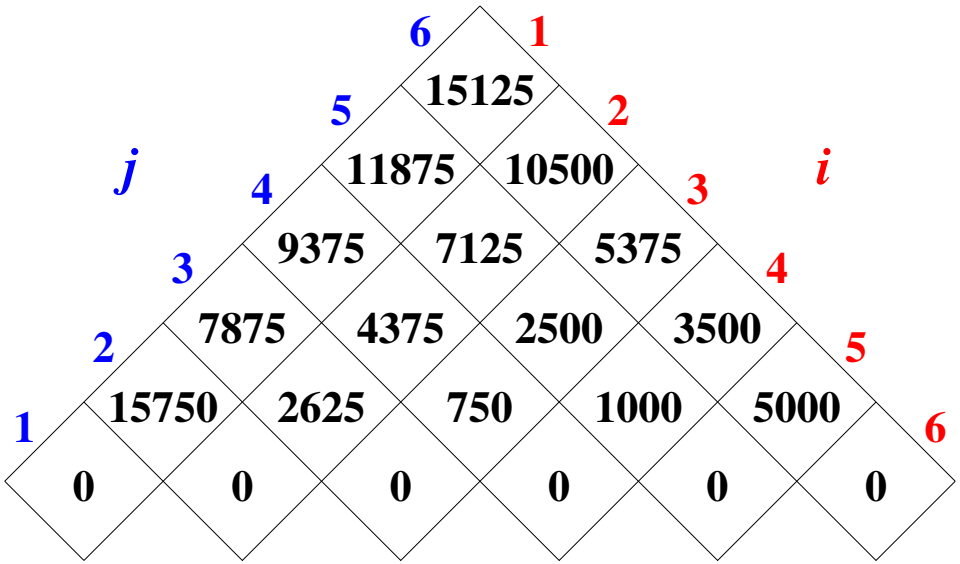




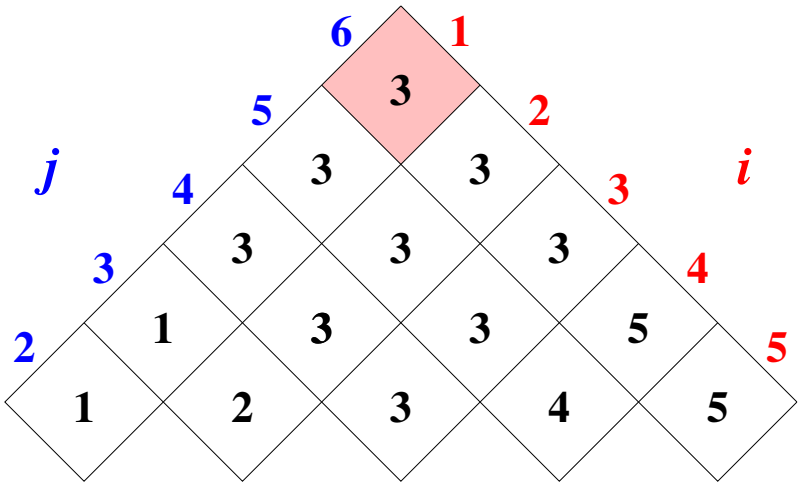
A_1	A_2	A_3	A_4	A_5	A_6
30×35	35×15	15×5	5×10	10×20	20×25

Příklad běhu BottomUpRNM$(6, d, m, h)$

m[1..6,1..6]



h[1..5,2..6]



A_1	A_2	A_3	A_4	A_5	A_6
30×35	35×15	15×5	5×10	10×20	20×25

Třetí krok DP algoritmu: Varianta 2
Rekurzivní algoritmus (shora dolů) výpočtu ceny optim. závorkování s tabelací

Princip tabelace (anglické označení je *memoization*):

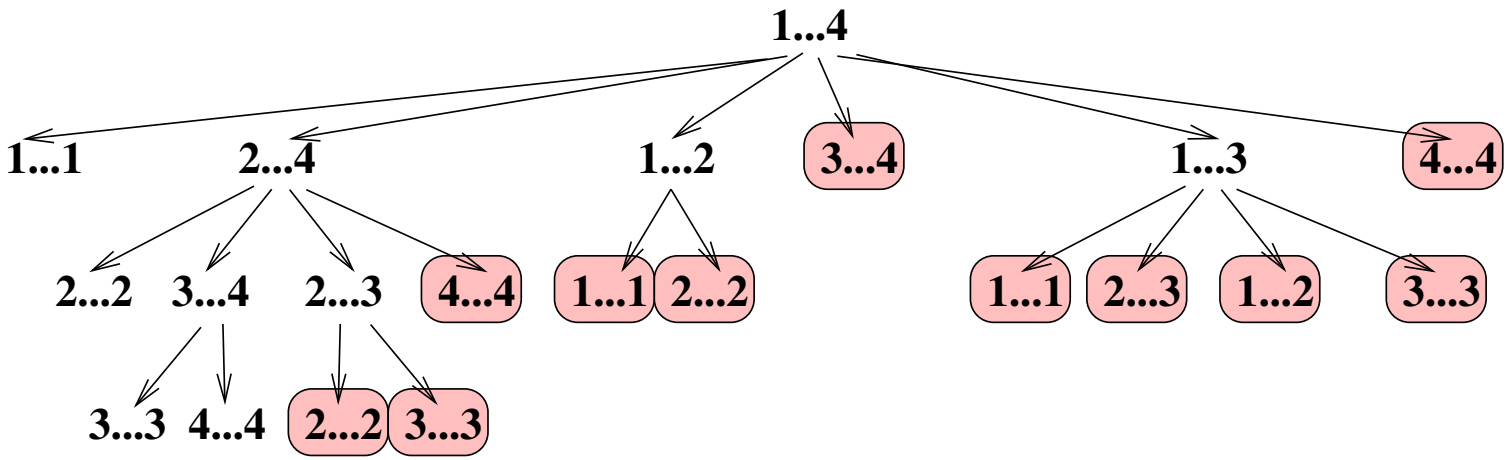
- Algoritmus také vytváří tabulku s hodnotami řešení podúloh.
- Každá položka tabulky je také inicializována speciální hodnotou `Neurčeno`.
- Řídící struktura pro zaplňování však není záplavové plnění zdola nahoru jako ve variantě 1, ale pořadí vyvolávané rekurzivním algoritmem.
- První rekurzivně vyvolané řešení dané podúlohy provede výpočet a výsledek uloží do příslušného místa v tabulce.
- Následné opakované volání řešení téže podúlohy pouze přečte tuto hodnotu z tabulky.

Rekurzivní algoritmus s tabelací

procedure MemRecRNM(n, d) $\{$
(1) **for** $(i \leftarrow 1$ **to** $n)$
(2) **for** $(j \leftarrow i$ **to** $n)$ **do** $m[i, j] \leftarrow$ `Neurceno`.
(3) **return**(LookUpRNM$(d, 1, n)$)$\}$

procedure LookUpRNM(d, i, j) $\{$
(1) **if** $(m[i, j] \neq$ `Neurceno`$)$
(2) **then return**$(m[i, j])$;
(3) **if** $(i = j)$
(4) **then** $m[i, j] \leftarrow 0$
(5) **else for** $(k \leftarrow i$ **to** $j - 1)$
(6) **do** $\{q \leftarrow$ LookUpRNM(d, i, k)
(7) $+$LookUpRNM$(d, k + 1, j) + d[i - 1] \cdot d[k] \cdot d[j]$;
(8) **if** $(q < m[i, j])$ **then** $\{m[i, j] \leftarrow q;\ h[i, j] \leftarrow k\}\}$;
(9) **return**$(m[i, j])\}$

Strom rekurzivních volání MemRecRNM



- Zakroužkované uzly stromu rekurzivních volání se nepočítají, ale pouze načtou z tabulky.

Složitost a srovnání MemRecRNM a BottomUpRNM

Lemma

Oba algoritmy mají složitost $O(n^3)$.

Srovnání:

- Algoritmus MemRecRNM je svojí podstatou metoda shora dolů, kdežto BottomUpRNM je metoda zdola nahoru.
- Pokud logika výpočtu vyžaduje, aby každá podúloha byla řešena aspoň jednou, pak BottomUpRNM je rychlejší o multiplikativní konstantu.
- Pravidelnost přístupu do tabulky pak uspoří čas i díky menším výpadkům skrytých pamětí a efektivnějším tokům dat mezi pamětí a procesorem.
- Pokud rekurzivní sestup umožní, že některé podúlohy se vůbec nevyvolají, pak MemRecRNM může být rychlejší.

Čtvrtý krok DP algoritmu: Konstrukce optimálního řešení úlohy ŘNM

- Tabulka $h[1 \ldots n-1, 2 \ldots n]$ obsahuje hodnoty hranic pro optimální závorkování při ŘNM.
- Díky Lemmatu na Slajdu 7 je konstrukce optimálního řešení triviální rekurzivním sestupem.
- Na nejvyšší úrovni to je $h[1, n]$: Nejprve rekurzivně optimálně vypočteme $A_1 \cdots A_{h[1,n]}$, pak $A_{h[1,n]+1} \cdots A_n$ a výsledné matice nakonec vynásobíme.

Čtvrtý krok DP algoritmu: Konstrukce optimálního řešení úlohy ŘNM

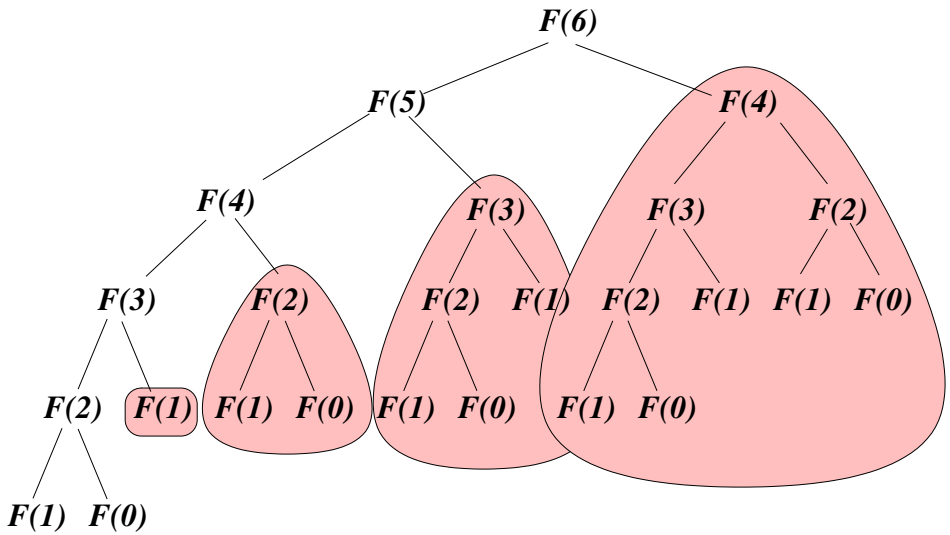
- Úloha ŘNM je pak vyřešena voláním $\mathrm{RNM}(A, n, h, 1, n)$.
- Vstupní data: posloupnost matic $A_1, A_2, \ldots, A_n$ a tabulka hranic $h[1 \ldots n-1, 2 \ldots n]$.

procedure $\mathrm{RNM}(A, n, h, i, j)$
{
(1) **if** $(i < j)$
(2) **then** { $X \leftarrow \mathrm{RNM}(A, n, h, i, h[i, j])$;
(3) $Y \leftarrow \mathrm{RNM}(A, n, h, h[i, j] + 1, j)$;
(4) **return**(XY)}
(5) **else return**(A_i);
}

Výpočet Fibonacciho posloupnosti

Naivní rekurzivní algoritmus:

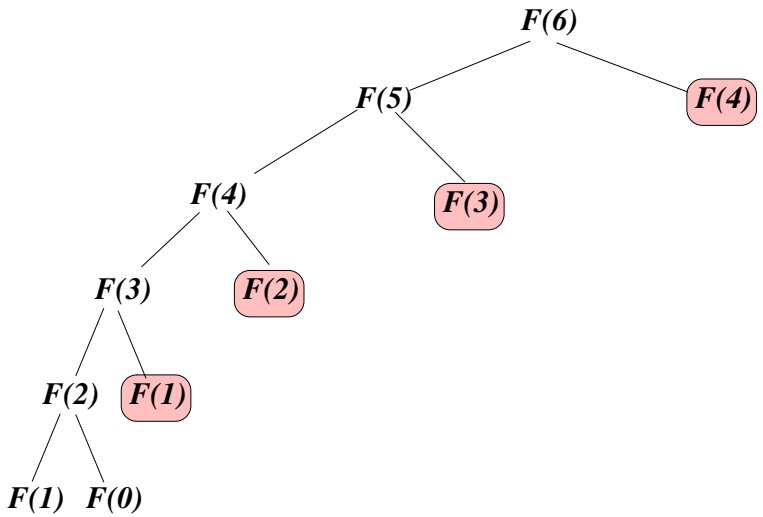
procedure $\textsc{RecFib}(n)$
$\{$
(1) **if** $(n \leq 1)$ **then return**(1);
(2) **return**$(\textsc{RecFib}(n-1) + \textsc{RecFib}(n-2))\}$



- SRV naivního rekurzivního algoritmu má **exponenciální** velikost ($O(\Phi^{n+1})$, viz Slajdy 25 až 33 v Přednášce 8).

DP: Tabelizace při výpočtu Fibonacciho posloupnosti

- Efekt tabelizace DP je ještě razantnější než v případě "2-rozměrného" ŘNM.
- Zakroužkované části se nemusí volat, pokud se příslušná hodnota Fibonacciho čísla při rekurzivním sestupu zapamatuje.
- Znovuvýpočet každého z těchto zakroužkovaných znovuvyvolaných podstromů je nahrazen přečtením 1 čísla z tabulky v $O(1)$ čase!!!
- Exponenciální složitost se změní na lineární!!!



DP: Metoda shora dolů

procedure MemRecFib(n) {
(1) **for** ($i \leftarrow 1$ **to** n) **do** $F[i] \leftarrow$ `Neurceno`.
(2) **return**(LookUpFib(n))}

procedure LookUpFib(i) {
(1) **if** ($F[i] \neq$ `Neurceno`)
(2) **then return**($F[i]$);
(3) **if** ($i \leq 1$)
(4) **then** $F[i] \leftarrow 1$
(5) **else** $F[i] \leftarrow$ LookUpFib($i-1$) + LookUpFib($i-2$);
(6) **return**($F[i]$)}

Nejefektivnější algoritmus: iterační výpočet zdola nahoru

procedure ITERFIB(n) {
(1) **if** $(n \leq 1)$
(2) **then** $F[i] \leftarrow 1$
(3) **else for** $(i \leftarrow 2$ **to** $n)$
(4) **do** $F[i] \leftarrow F[i-1] + F[i-2];$
(5) **return**$(F[1 \ldots n])\}$

Shrnutí principů algoritmů DP

DP se hodí na řešení optimalizačních úloh s vysokou (superpolynomiální) složitostí, pokud mají následující 2 vlastnosti:

1. Úloha vykazuje **optimální substrukturu**: Optimální řešení celé úlohy lze poskládat z optimálních řešení podúloh.
2. Podúlohy se výrazně **překrývají**, t.j. ve stromu rekurzivních volání se mnohé podstromy vyskytují vícenásobně.

Příklady úloh vhodných pro řešení metodou DP

1. Floydův algoritmus hledání nejkratších cest v grafech.
2. Optimální plánování výpočetních úloh, zadaných časovými intervaly, v kterých mohou běžet.
3. Výpočet Levenshteinovy vzdálenosti mezi 2 textovými řetězci.
4. Určení způsobu, jak nejlépe daný řetězec vygenerovat pomocí dané bezkontextové gramatiky.
5. Hledání nejdelšího společného podřetězce dvou řetězců.
6. Optimální triangularizace konvexního mnohoúhelníku.
7. Problém obchodního cestujícího:
   - Obecný případ v exponenciálním, ale $o(n!)$ čase.
   - Spec. případy v polynomiálním čase.
8. ...