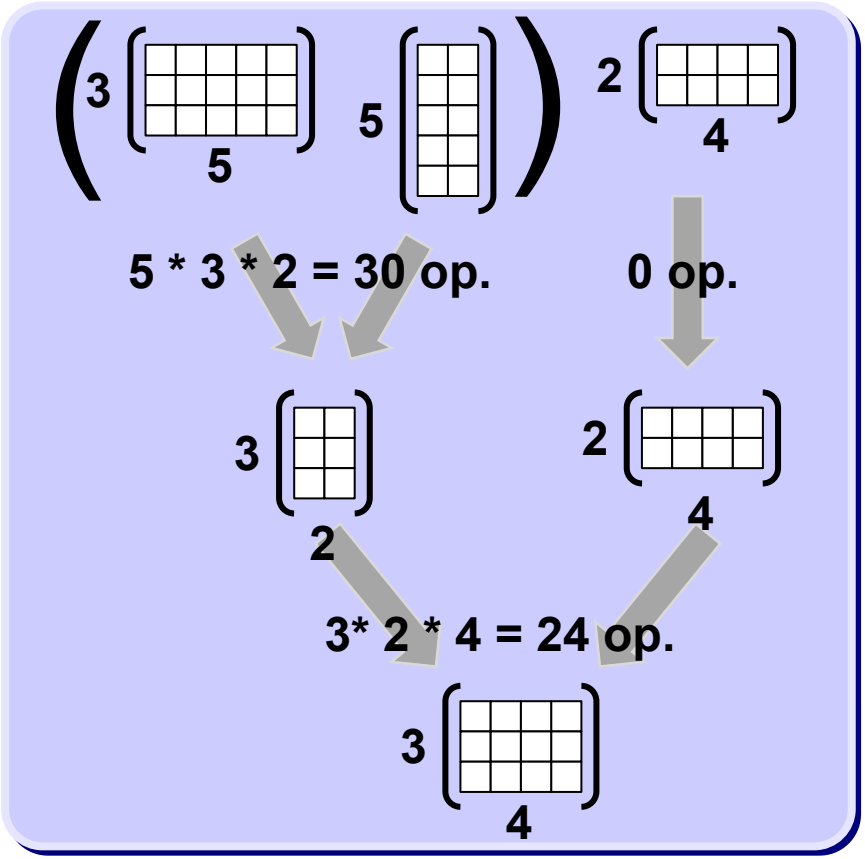
Algoritmizace

Daniel Průša, Robert Pěnička

2025

Přehled

- Dynamické programování (DP)
  - Nejdelší rostoucí podposloupnost
  - Optimální pořadí násobení matic
  - Nejdelší společná podposloupnost



Join at slido.com
#7719040

Nejdelší rostoucí podposloupnost

Z dané posloupnosti vyberte co nejdelší rostoucí podposloupnost.

5 4 9 11 5 3 2 10 0 8 6 1 7

Řešení: 4 5 6 7

Jiné možné varianty

Vlastnosti hledané podposloupnosti:
Klesající, nerostoucí, neklesající, aritmetická,
s omezenou rychlostí růstu, s váhami prvků, ... atd., ...

zde neprobírané

Koncepční přístup I

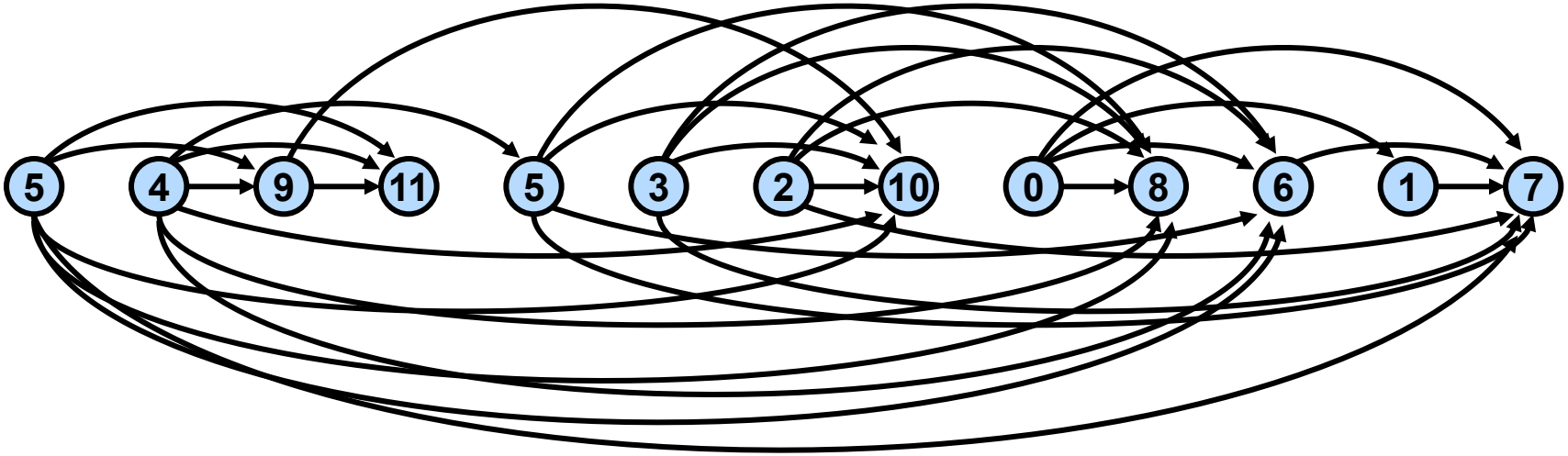
Převeď na známou úlohu, definuj vhodný DAG podle daných vlastností podposloupnosti, v DAG hledej nejdelší cestu.

Nejdelší rostoucí podposloupnost

Koncepční přístup I | **Transformace na známou úlohu**

Prvky posloupnosti budou uzly DAG, který je již topologicky uspořádán, pořadí v posloupnosti = pořadí v top. uspořádání.
Hrana x --> y existuje právě tehdy,
když x je v posloupnosti dříve než y a navíc x<y.

V tomto DAG hledáme nejdelší cestu.



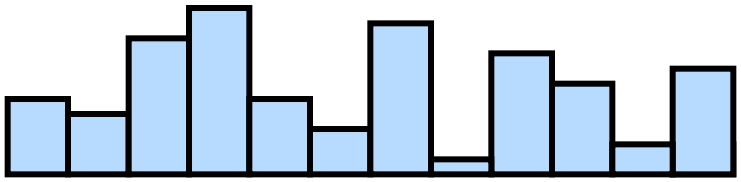
Algoritmus je znám, má složitost $\Theta(N+M)$, tedy $O(N^2)$.
Např. pro rostoucí posloupnost má složitost až $\Theta(N^2)$.

Nejdelší rostoucí podposloupnost

Koncepční přístup II - sestav nezávislý a rychlejší algoritmus

Registrujme optimální podposloupnosti (OPP) všech možných délek.
OPP každé délky má co nejmenší možný poslední prvek.
To ji dává potenciál budoucího růstu.
Postupně metodou DP aktualizujme tyto optimální podposloupnosti.

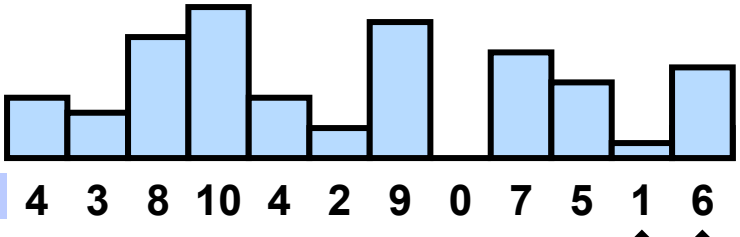
k	1	2	3	4	5	6	7	8	9	10	11	12
V	5	4	9	11	5	3	10	1	8	6	2	7
p	--											
iL	--											



k .. index prvku
V[k] .. hodnota prvku
p[k] .. index předchůdce prvku
iL[d] .. index posledního prvku v rostoucí podposloupnosti délky d = 1, 2, ..., N.

Pro každý index k:
Nechť d je index <u>největšího</u> prvku, pro který platí V[iL[d]] < V[k].
Potom iL[d+1] := k, p[k] = iL[d], pokud d existuje.
Jinak iL[1] := k, p[k] = null.
V[iL[d]], d = 1..N je neklesající, lze v ní hledat v čase O(log N).

Nejdelší rostoucí podposloupnost



k	1	2	3	4	5	6	7	8	9	10	11	12
V	4	3	8	10	4	2	9	0	7	5	1	6

k = 11

p	--	--	2	3	2	--	5	--	5	5	8	
iL	8	11	10									
V[iL]	0	1	5									

k = 12

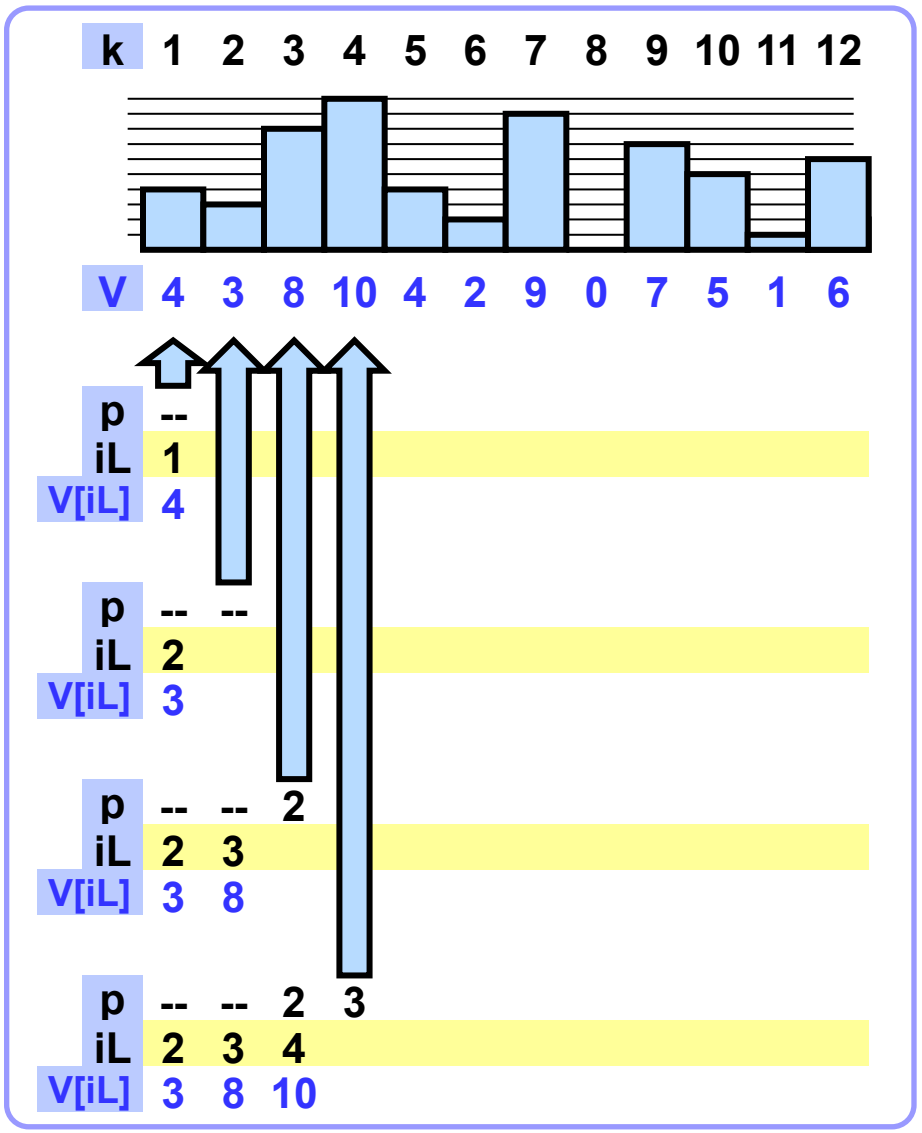
p	--	--	2	3	2	--	5	--	5	5	8	10
iL	8	11	10	12								
V[iL]	0	1	5	6								

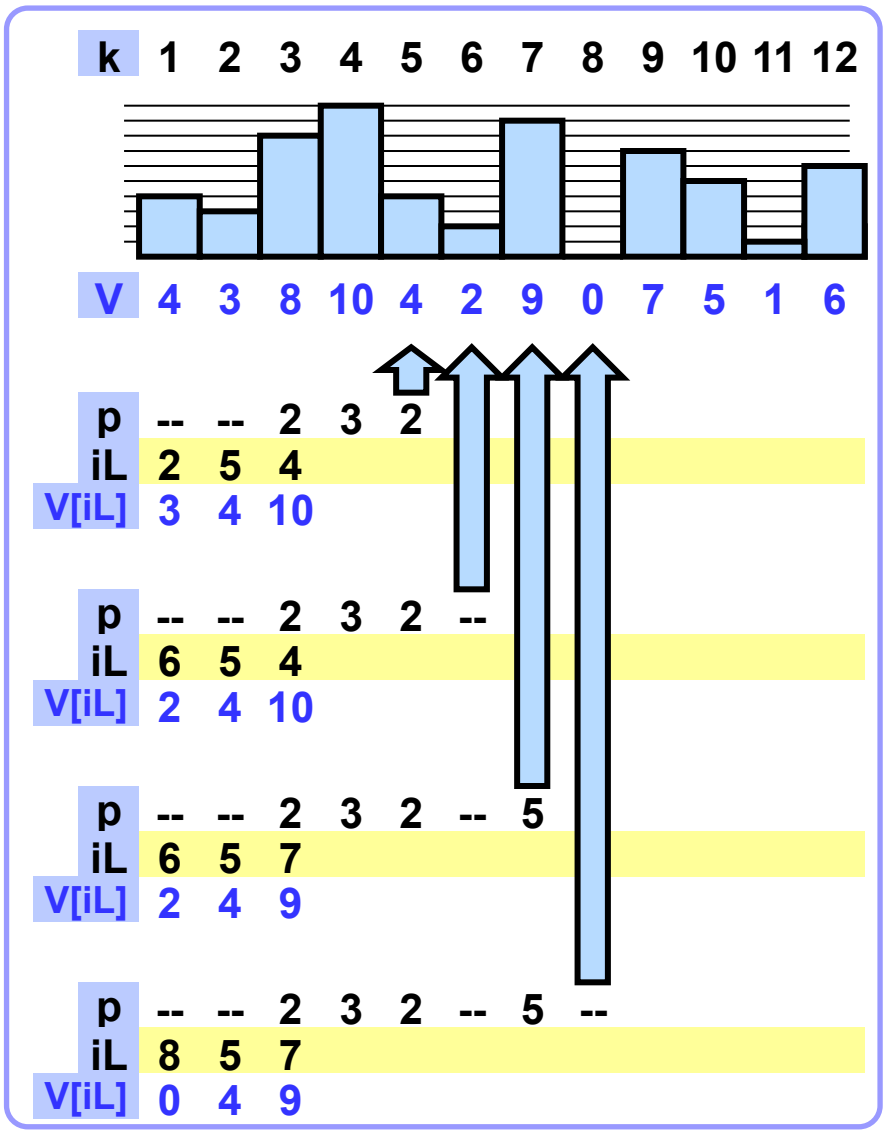
k .. index prvku
V[k] .. hodnota prvku
p[k] .. index předchůdce prvku
iL[d] .. index posledního prvku v rostoucí podposloupnosti délky d = 1, 2, ..., N.

Pro každý index k:
Nechť d je index max. prvku, pro který platí
V[iL[d]] < V[k].
Potom iL[d+1] := k, p[k] = iL[d], pokud d existuje.
Jinak iL[1] := k, p[k] = null.

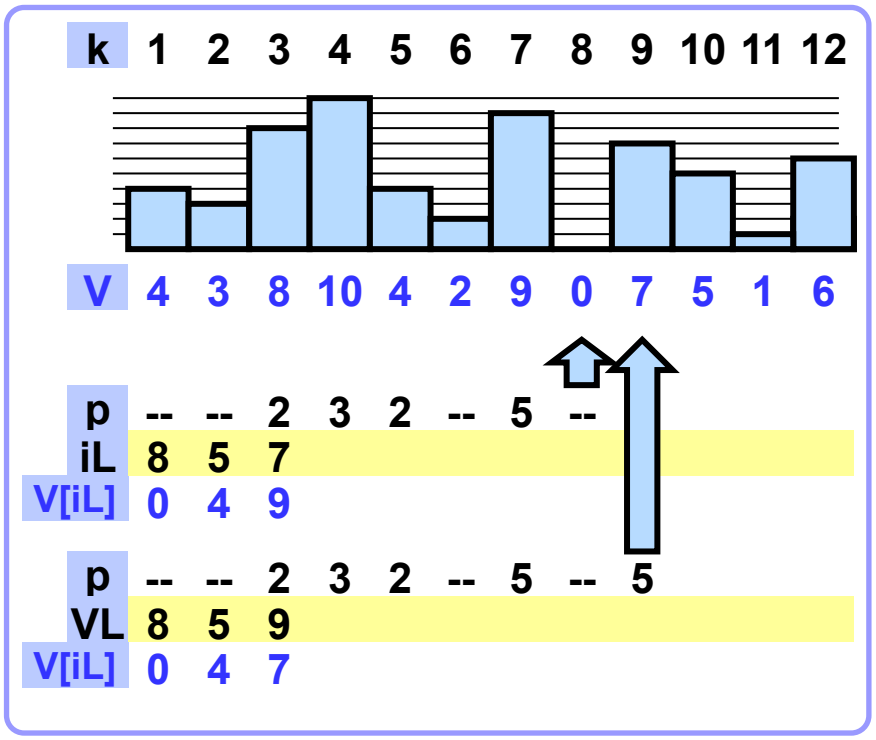
Pro každé k nalezneme d v čase O(log N) půlením intervalu.
Aktualizace iL a p proběhne v konstantním čase.
Celkem je složitost O(N log (N)).

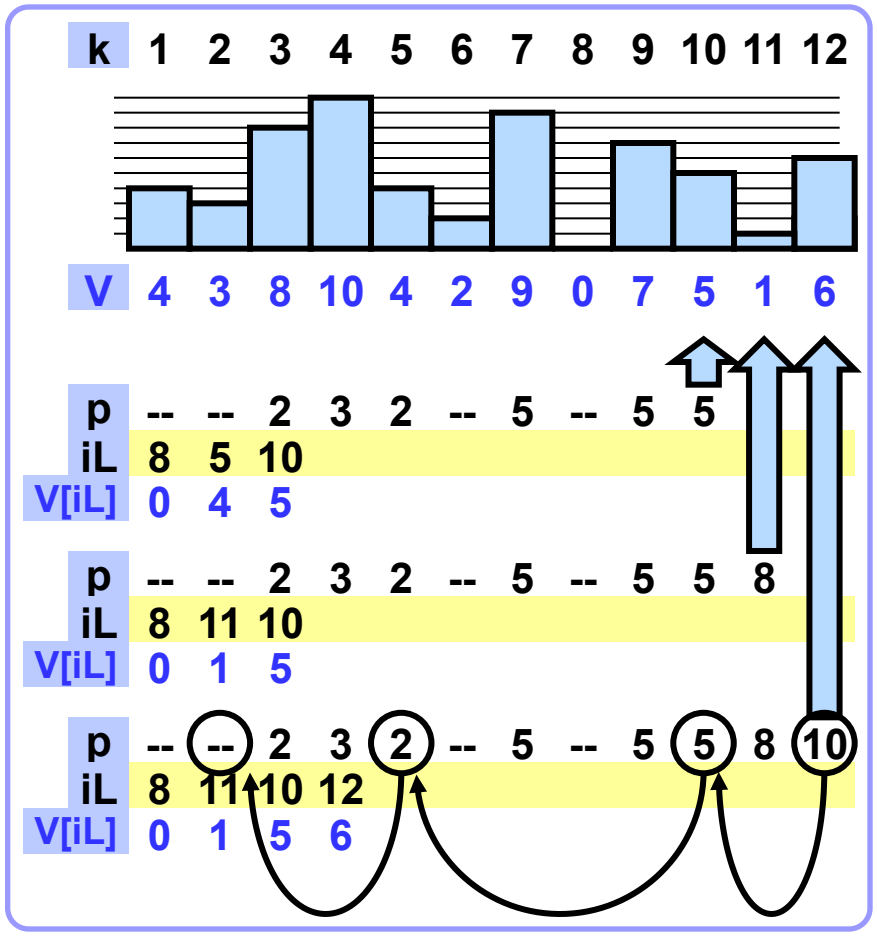
Nejdelší rostoucí podposloupnost





Nejdelší rostoucí podposloupnost





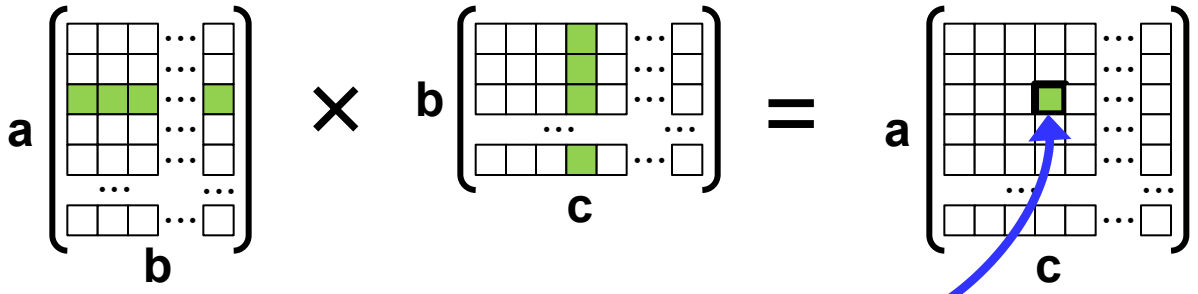
Rekonstrukce optimální cesty

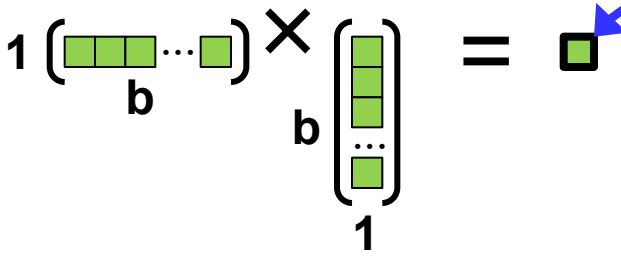
Poslední definovaný prvek v iL je indexem posledního prvku jedné z optimálních podposloupností celé posloupnosti.
Pole p určuje pomocí předchůdců tuto podposloupnost.

Audience Q&A

Optimální pořadí násobení matic

Počet operací v násobení dvou matic





b operací násobení pro výpočet jednoho prvku výsledné matice

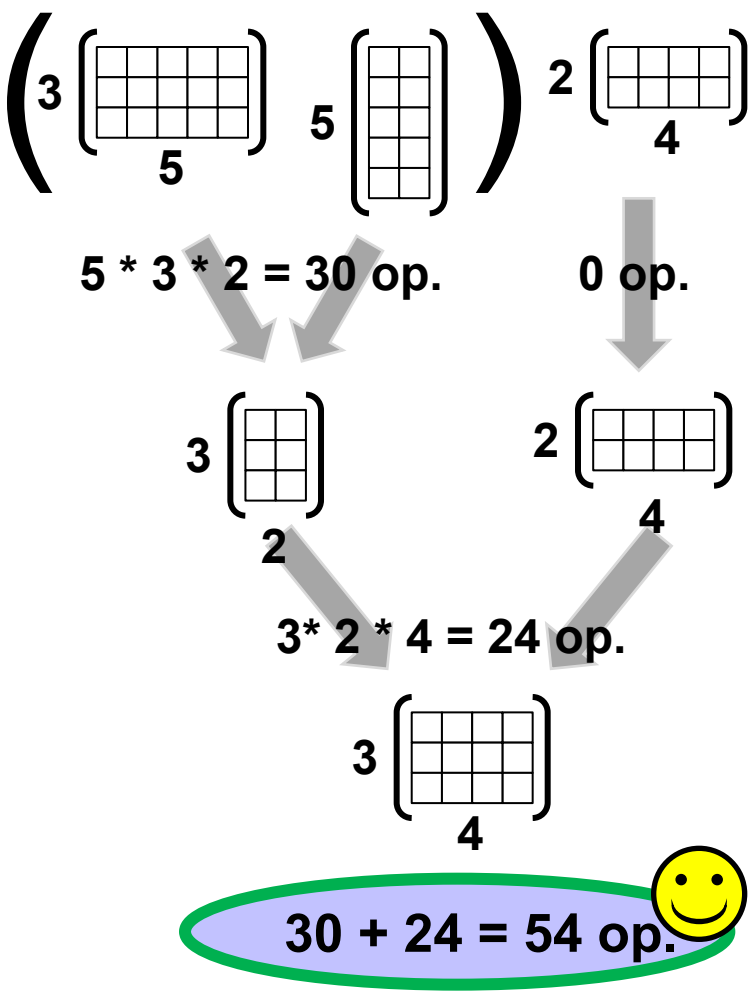
a * c prvků ve výsledné matici

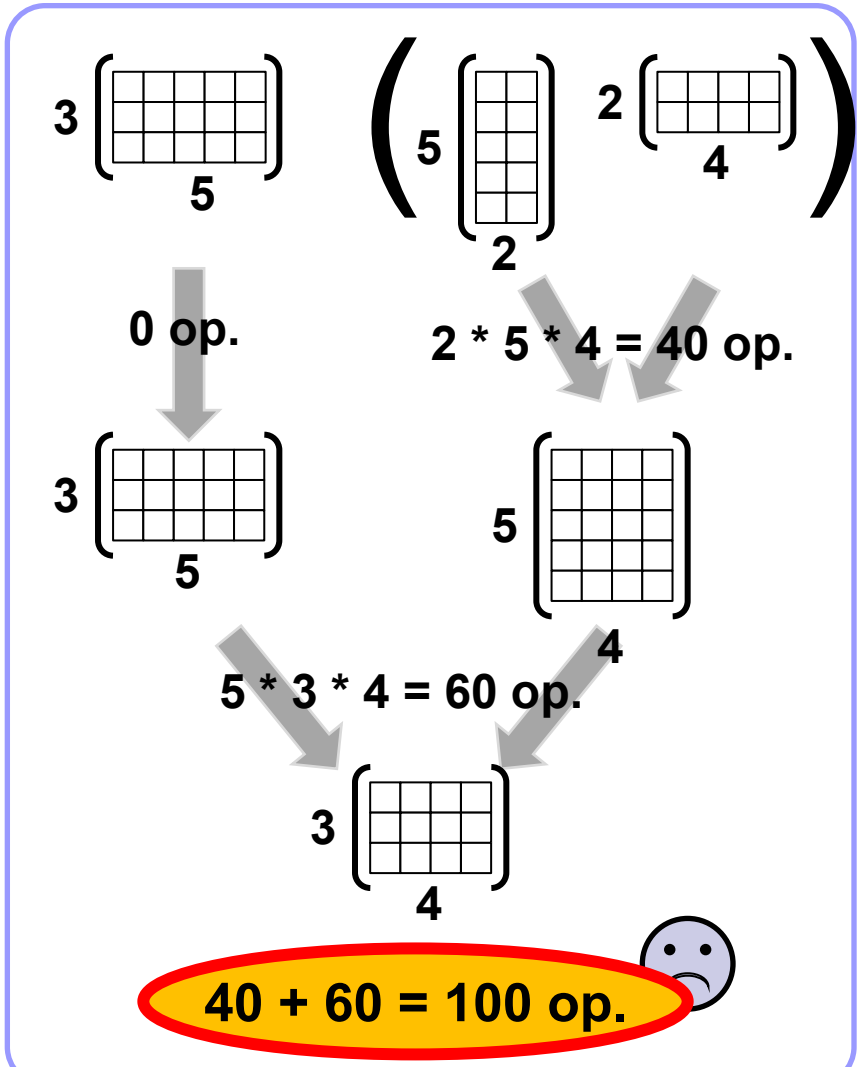
Vynásobení dvou matic o rozměrech a×b a b×c vyžaduje celkem a * b * c operací násobení dvou prvků (čísel).

Sčítání zde neuvažujeme, lze pro něj vyvinout analogický postup.

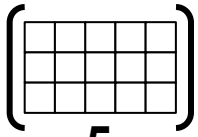
Optimální pořadí násobení matic

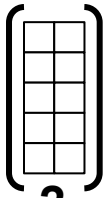
Příklad násobení více matic

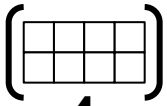




Optimální pořadí násobení matic

A_1 = 3 × 5 

A_2 = 5 × 2 

A_3 = 2 × 4 

Součin $(A_1 \times A_2) \times A_3$ vyžaduje 54 operace násobení .

Součin $A_1 \times (A_2 \times A_3)$ vyžaduje 100 operací násobení.

Evidentně, na způsobu uzávorkování záleží .

Catalanova čísla C_N

Součin $A_1 \times A_2 \times A_3 \times \ldots \times A_N$ lze uzávorkovat

C_N = Comb(2N, N) / (N+1) způsoby.

$C_1, C_2, \ldots, C_7, \ldots$ = 1, 1, 2, 5, 14, 42, 132, ... $C_N > 2^N$ pro N > 7.

V obecném případě by mělo vyzkoušení všech uzávorkování exponenciální složitost.

Optimální pořadí násobení matic

- **14 různých způsobů uzávorkování součinu 5 činitelů**

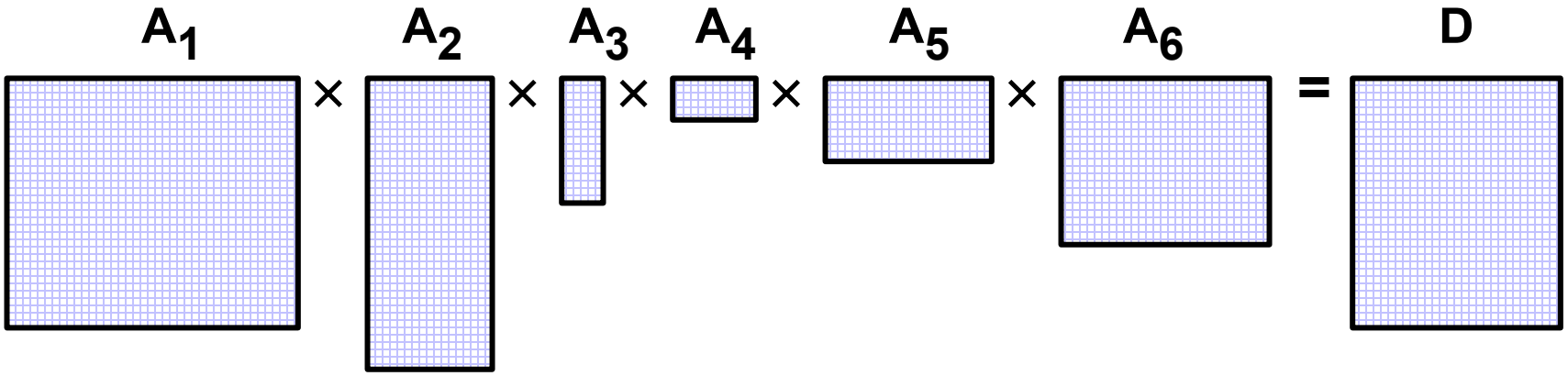
$$A_1 \times (A_2 \times (A_3 \times (A_4 \times A_5)))$$
$$A_1 \times (A_2 \times ((A_3 \times A_4) \times A_5))$$
$$A_1 \times ((A_2 \times A_3) \times (A_4 \times A_5))$$
$$A_1 \times ((A_2 \times (A_3 \times A_4)) \times A_5)$$
$$A_1 \times (((A_2 \times A_3) \times A_4) \times A_5)$$
$$(A_1 \times A_2) \times (A_3 \times (A_4 \times A_5))$$
$$(A_1 \times A_2) \times ((A_3 \times A_4) \times A_5)$$
$$(A_1 \times (A_2 \times A_3)) \times (A_4 \times A_5)$$
$$((A_1 \times A_2) \times A_3) \times (A_4 \times A_5)$$
$$(A_1 \times (A_2 \times (A_3 \times A_4))) \times A_5$$
$$(A_1 \times ((A_2 \times A_3) \times A_4)) \times A_5$$
$$((A_1 \times A_2) \times (A_3 \times A_4)) \times A_5$$
$$((A_1 \times (A_2 \times A_3)) \times A_4) \times A_5$$
$$(((A_1 \times A_2) \times A_3) \times A_4) \times A_5$$

Optimální pořadí násobení matic

Instance úlohy

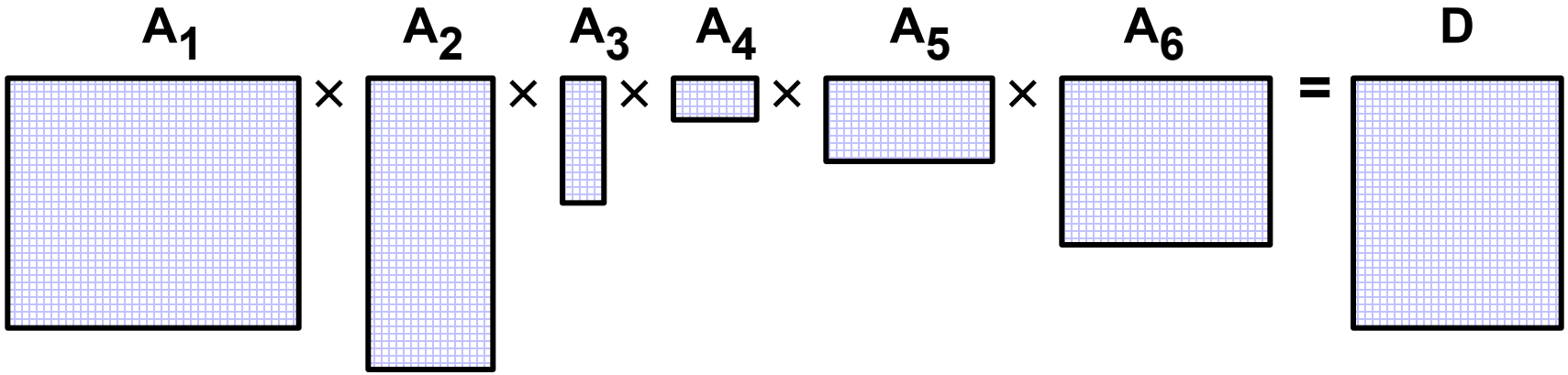
Máme spočítat co nejefektivněji součin reálných matic
$A_1 \times A_2 \times A_3 \times A_4 \times A_5 \times A_6$,
kde rozměry jednotlivých matic jsou po řadě
30×35, 35×15, 15×5, 5×10, 10×20, 20×25.
(Výsledná matice D má rozměr 30×20).

Grafická podoba (dimenze matic ve správném poměru)



Instance převzata z [CLRS], kap. 15.

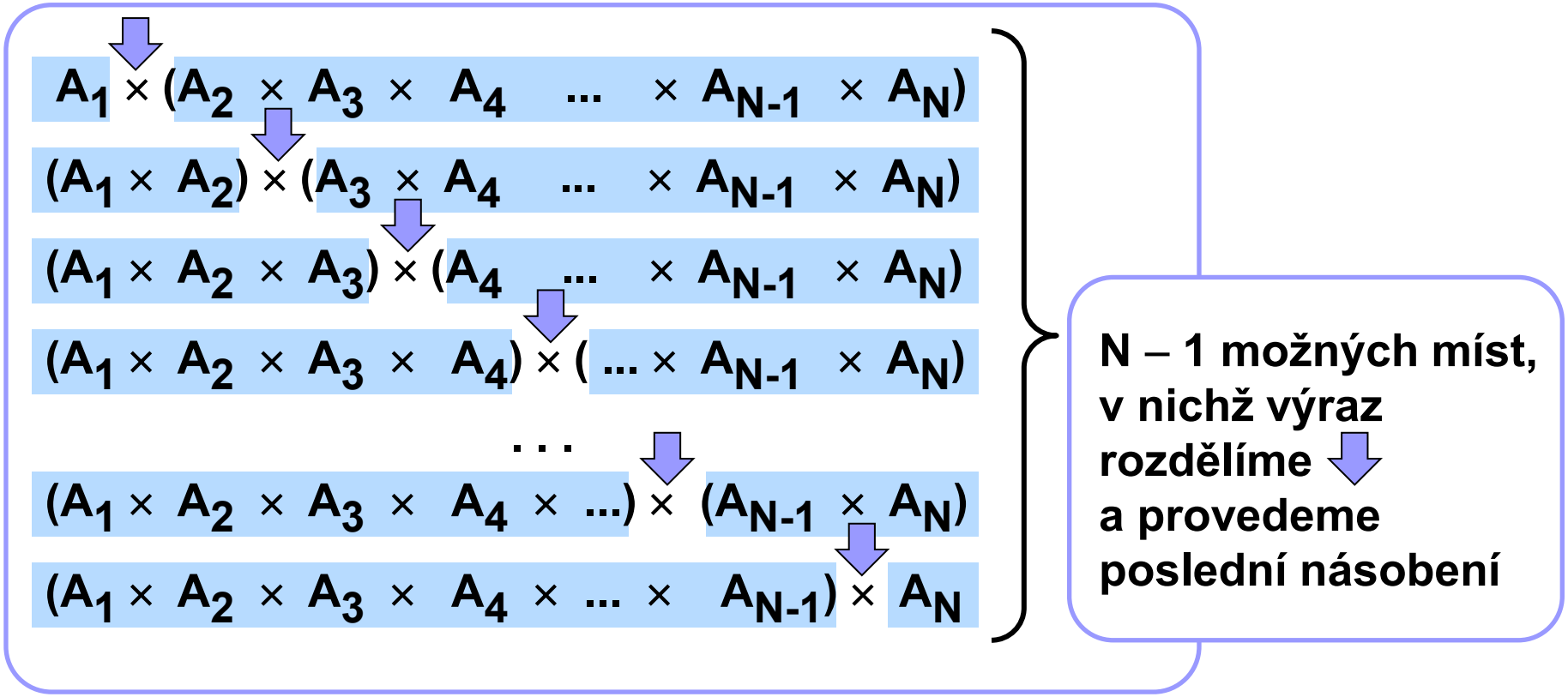
Optimální pořadí násobení matic



Sledujeme jen počet operací součinu dvou reálných čísel.
Uvažujeme různé možnosti uzávorkování a tím i pořadí výpočtu.

metoda	Výraz	Počet operací
zleva doprava	$((((A_1 \times A_2) \times A_3) \times A_4) \times A_5) \times A_6$	43 500
zprava doleva	$A_1 \times (A_2 \times (A_3 \times (A_4 \times (A_5 \times A_6))))$	47 500
nejhorší	$A_1 \times ((A_2 \times ((A_3 \times A_4) \times A_5)) \times A_6)$	58 000
nejlepší	$(A_1 \times (A_2 \times A_3)) \times ((A_4 \times A_5) \times A_6)$	15 125

Optimální pořadí násobení matic



Předpokládejme, že máme předpočítáno optimální uzávorkování pro každý modrý úsek celkového výrazu.

Optimální pořadí násobení matic

$$A_1 \times (A_2 \times A_3 \times A_4 \ \ldots \ \times A_{N-1} \times A_N) = B[1,1] \times B[2,N]$$

$$(A_1 \times A_2) \times (A_3 \times A_4 \ \ldots \ \times A_{N-1} \times A_N) = B[1,2] \times B[3,N]$$

$$(A_1 \times A_2 \times A_3) \times (A_4 \ \ldots \ \times A_{N-1} \times A_N) = B[1,3] \times B[4,N]$$

$$(A_1 \times A_2 \times A_3 \times A_4) \times (\ldots \times A_{N-1} \times A_N) = B[1,4] \times B[5,N]$$

$$\ldots \qquad\qquad \ldots$$

$$(A_1 \times A_2 \times A_3 \times A_4 \times \ldots) \times (A_{N-1} \times A_N) = B[1,N-2] \times B[N-1,N]$$

$$(A_1 \times A_2 \times A_3 \times A_4 \times \ldots \times A_{N-1}) \times A_N = B[1,N-1] \times B[N,N]$$

Matice B[i, j] představuje výsledek vynásobení odpovídajícího úseku.

Nechť r(X) resp. s(X) představují počet řádků resp sloupců matice X. Podle pravidel násobení matic platí
$r(B[i, j]) = r(A_i)$, $s(B[i, j]) = s(A_j)$, pro $1 \le i \le j \le N$.

Optimální pořadí násobení matic

Nechť MO[i, j] představuje minimální počet operací potřebných k výpočtu matice B[i, j], tj. minimální počet operací potřebných k výpočtu matice $A_i \times A_{i+1} \times \ldots \times A_{j-1} \times A_j$.

B[1,1]	× B[2,N]		MO[1,1] + $r(A_1)*s(A_1)*s(A_N)$ + MO[2, N]
B[1,2]	× B[3,N]		MO[1,2] + $r(A_1)*s(A_2)*s(A_N)$ + MO[3, N]
B[1,3]	× B[4,N]		MO[1,3] + $r(A_1)*s(A_3)*s(A_N)$ + MO[4, N]
	...		
B[1,N-2]	× B[N-1,N]		MO[1,N-2] + $r(A_1)*s(A_{N-2})*s(A_N)$ + MO[N-1, N]
B[1,N-1]	× B[N,N]		MO[1,N-1] + $r(A_1)*s(A_{N-1})*s(A_N)$ + MO[N, N]

operací v levém úseku

operací při násobení B[1,.] × B[.,N]

operací v pravém úseku

Celkem dostáváme MO[1,N]:

$MO[1,N] = \min \{MO[1,k] + r(A_1)*s(A_k)*s(A_N) + MO[k+1, N] \mid k = 1..N-1\}$

Optimální pořadí násobení matic

$$MO[1,N] = \min \{MO[1,k] + r(A_1)*s(A_k)*s(A_N) + MO[k+1, N] \mid k = 1..N-1\}$$

Za předpokladu znalosti MO[i, j] pro úseky kratší než [1, N],
lze řešení celé úlohy, tj. hodnotu MO[1, N], spočíst v čase $\Theta(N)$. (*)

Rekurentní využití řešení menších podúloh

Identické úvahy, jaké jsme provedli pro celý výraz
$A_1 \times A_2 \times A_3 \times \ldots \times A_N$,
provedeme rovněž pro každý jeho souvislý úsek
$\ldots A_L \times A_{L+1} \times \ldots \times A_{R-1} \times A_R \ldots$, $1 \le L \le R \le N$.

Počet těchto souvislých úseků je stejný jako počet dvojic indexů (L, R), kde $1 \le L \le R \le N$. Ten je roven $Comb(N, 2) \in \Theta(N^2)$.
Podúlohu na úseku (L, R) lze spočíst podle (*) v čase O(N),
celou úlohu tak lze vyřešit v čase $O(N^3)$.

Optimální pořadí násobení matic

$MO[L,R] = \min \{MO[L,k] + r(A_L)*s(A_k)*s(A_R) + MO[k+1,R] \mid k = L..R-1\}$ (*)

Hodnoty MO[L,R] ukládáme do 2D pole na pozici s indexy [L][R].

Při výpočtu MO[L,R] podle (*) používáme vesměs hodnoty MO[x,y], kde rozdíl y - x (odpovídající délce podvýrazu) je menší než rozdíl R - L.
Tabulku DP proto vyplňujeme v pořadí rostoucích rozdílů R - L.

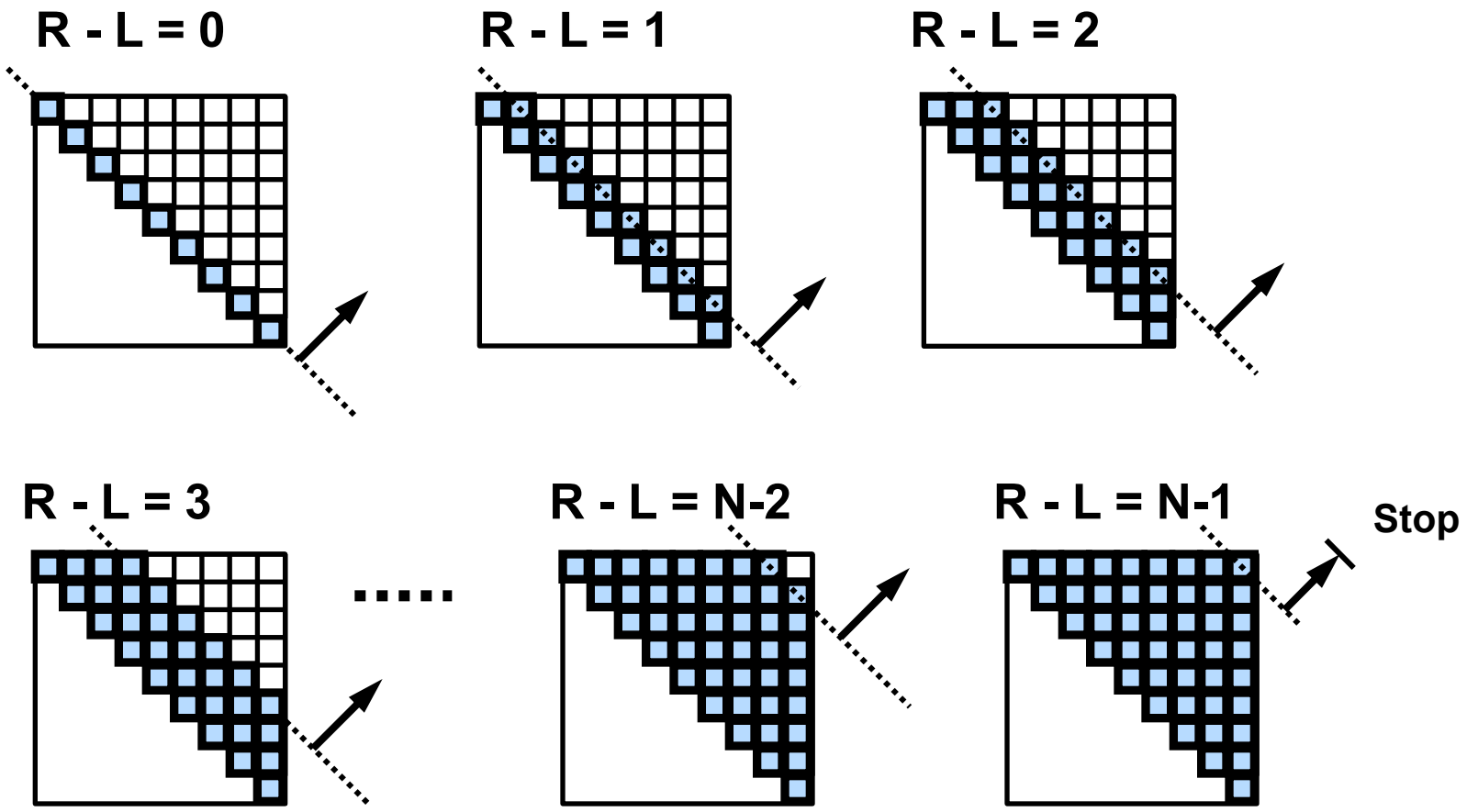
0. Vyplníme prvky s indexy[L][R], kde R-L = 0, to je hlavní diagonála.
1. Vyplníme prvky s indexy[L][R], kde R-L = 1, to je diagonála těsně nad hlavní diagonálou.
2. Vyplníme prvky s indexy[L][R], kde R-L = 2, to je diagonála těsně nad předchozí diagonálou.

...

N-1. Vyplníme prvek s indexem [L][R], kde R-L = N-1, to je pravý horní roh tabulky.

Optimální pořadí násobení matic

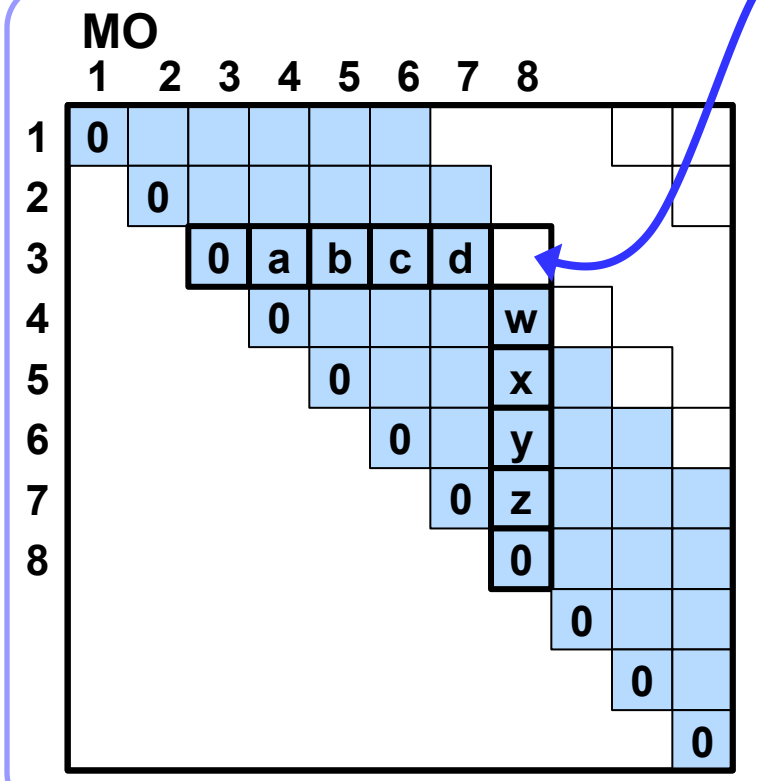
Schéma postupu výpočtu



Optimální pořadí násobení matic

$$MO[L,R] = \min \{MO[L,k] + r(A_L)*s(A_k)*s(A_R) + MO[k+1,R] \mid k = L..R-1\}$$

Ukázka postupu výpočtu



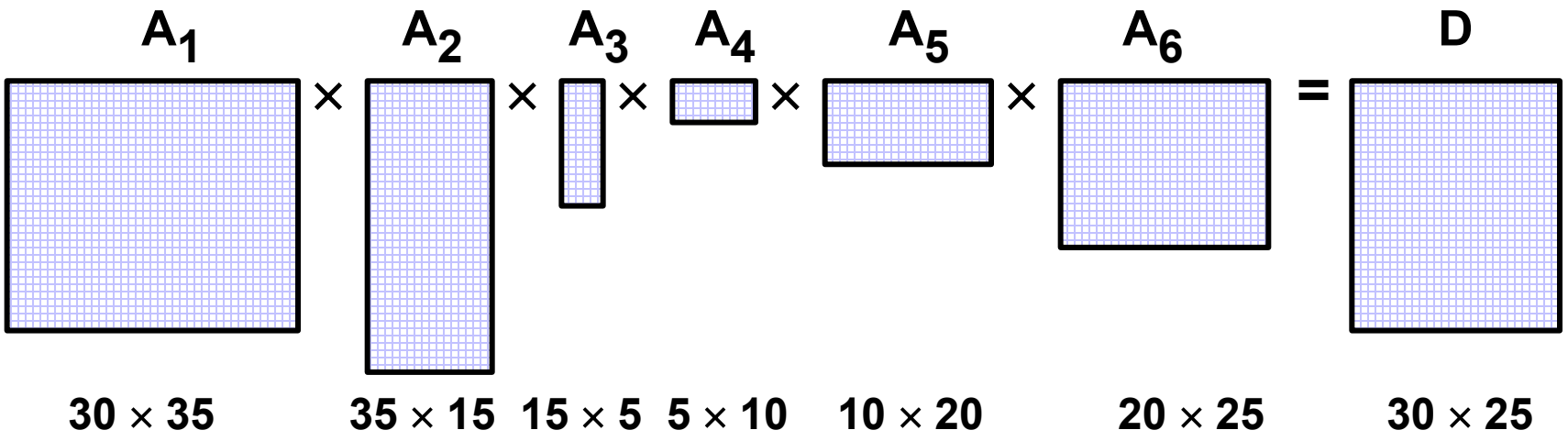
$MO[3,8] = \min \{$
$MO[3,3] + r(A_3)*s(A_3)*s(A_8) + MO[4,8],$
$MO[3,4] + r(A_3)*s(A_4)*s(A_8) + MO[5,8],$
$MO[3,5] + r(A_3)*s(A_5)*s(A_8) + MO[6,8],$
$MO[3,6] + r(A_3)*s(A_6)*s(A_8) + MO[7,8],$
$MO[3,7] + r(A_3)*s(A_7)*s(A_8) + MO[8,8]\}$

Označme $P[L, R] := r(A_L)*s(A_R)$. Potom

$MO[3,8] = \min \{$
$0 + s(A_3)*P[3,8] + w,$
$a + s(A_4)*P[3,8] + x,$
$b + s(A_5)*P[3,8] + y,$
$c + s(A_6)*P[3,8] + z,$
$d + s(A_7)*P[3,8] + 0\}.$

Optimální pořadí násobení matic

Instance úlohy



MO

	1	2	3	4	5	6
1	0	15750	7875	9375	11875	15125
2	0	0	2625	4375	7125	10500
3	0	0	0	750	2500	5375
4	0	0	0	0	1000	3500
5	0	0	0	0	0	5000
6	0	0	0	0	0	0

optimum

Optimální pořadí násobení matic

Rekonstrukce uzávorkování

$MO[L,R] = \min^{*} \{MO[L,k] + r(A_L)*s(A_k)*s(A_R) + MO[k+1,R] \mid k = L..R-1\}$

Při určení MO[L,R] do rekonstrukční tabulky RT stejné velikosti jako MO zaneseme na pozici [L][R] hodnotu k, v níž minimum * nastalo.
Hodnota k určuje optimální rozdělení výrazu
$(A_L \times A_{L+1} \times \ldots \times A_R)$

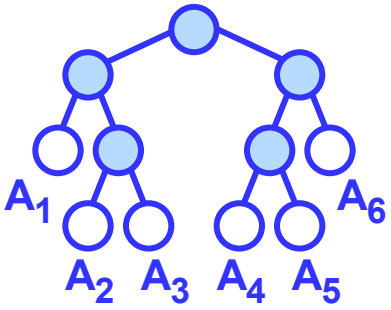
na dva menší optimálně uzávorkované výrazy
$(A_L \times A_{L+1} \times \ldots \times A_k) \times (A_{k+1} \times A_{k+2} \times \ldots \times A_R)$

Hodnota RT[1, N] určuje optimální rozdělení celého výrazu
$A_1 \times A_2 \times \ldots \times A_N$
na první dva menší optimálně uzávorkované výrazy
$(A_1 \times A_2 \times \ldots \times A_k) \times (A_{k+1} \times A_{k+2} \times \ldots \times A_N)$.
Dále rekonstrukce optimálního uzávorkování pokračuje rekurzivně analogicky pro výraz $(A_1 \times A_2 \times \ldots \times A_k)$ a pro výraz $(A_{k+1} \times A_{k+2} \times \ldots \times A_N)$
a dále pro jejich podvýrazy atd.

Optimální pořadí násobení matic

RT

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



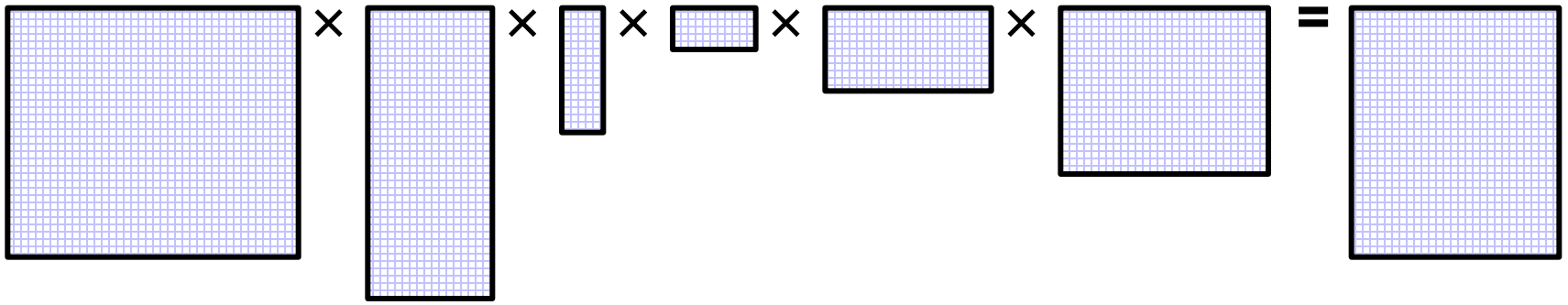
$A_1 \leftrightarrow A_6$

$A_1 \leftrightarrow A_3 \times A_4 \leftrightarrow A_6$

$A_1 \leftrightarrow A_1 \times A_2 \leftrightarrow A_3 \quad A_4 \leftrightarrow A_5 \times A_6 \leftrightarrow A_6$

A_1

$(A_1 \times (A_2 \times A_3)) \times ((A_4 \times A_5) \times A_6) = D$



Optimální pořadí násobení matic

Odvození asymptotické složitosti

index řádku	Řádkové součty
k = N–1	1/2 * (N–1) * N
k = N–2	1/2 * (N–2) * (N–1)
k = N–3	1/2 * (N–3) * (N–2)
k = k	1/2 * k * (k+1)
k = 3	1/2 * 3 * 4
k = 2	1/2 * 2 * 3
k = 1	1/2 * 1 * 2

Počet buněk, z nichž je počítán obsah dané buňky v DP tabulce, je úměrný složitosti výpočtu obsahu této buňky.

1	2	3		N–3	N–2	N–1
	1	2		N–4	N–3	N–2
		1		N–5	N–4	N–3
			1	N–k–2	N–k–1	N–k
				1	2	3
					1	2
						1

Celkový součet

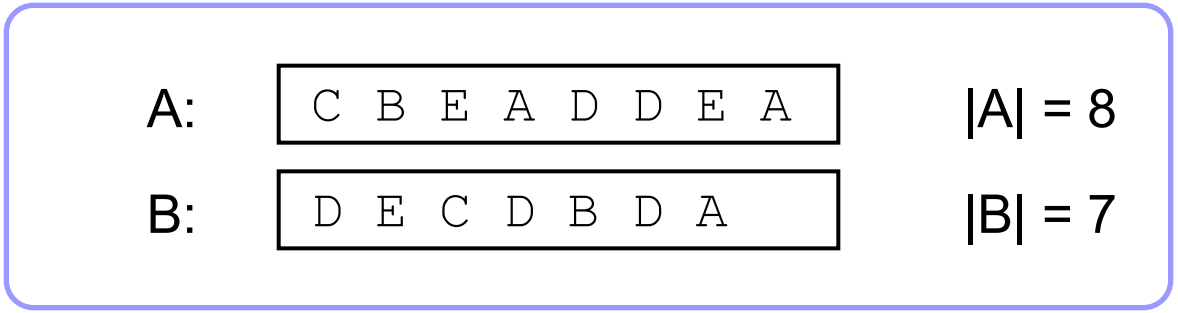
$$\frac{1}{2} * \sum_{k=1}^{N-1} k * (k+1) = \frac{1}{2} * \sum_{k=1}^{N-1} k^2 + \frac{1}{2} * \sum_{k=1}^{N-1} k$$

$$= \frac{1}{2} * (N-1) * N * (2N-1)/6 + 1/2 * (N-1) * N/2 \in \Theta(N^3)$$

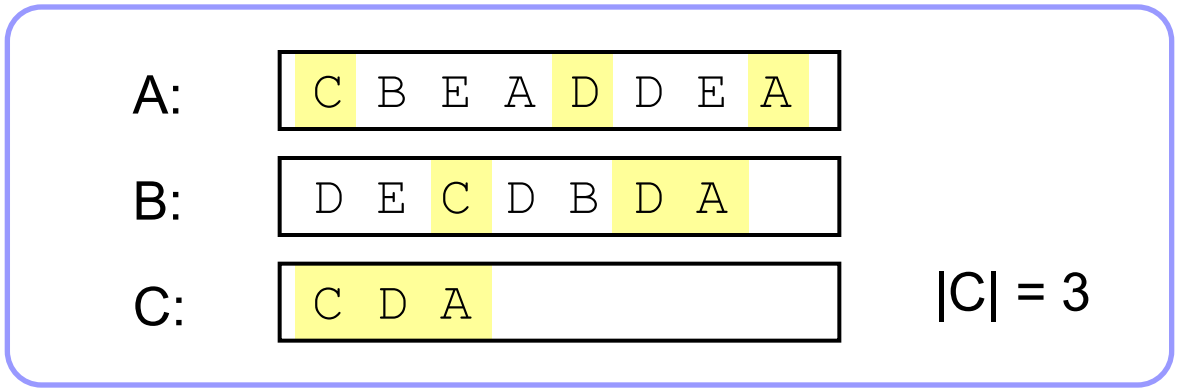
Audience Q&A

Nejdelší společná podposloupnost

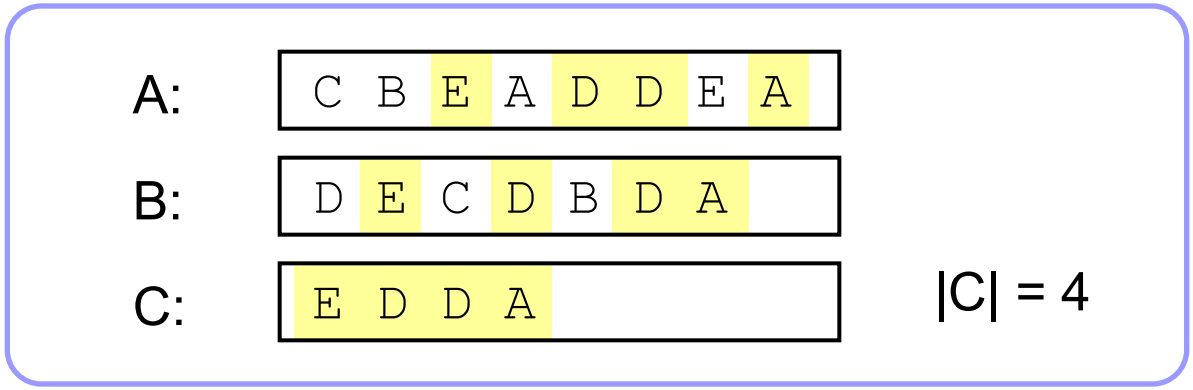
Dvě posloupnosti



Společná podposloupnost



Nejdelší společná podposloupnost (NSP)



Nejdelší společná podposloupnost

A_n: $(a_1, a_2, ..., a_n)$

B_m: $(b_1, b_2, ..., b_m)$

C_k: $(c_1, c_2, ..., c_k)$

.....

$C_k = LCS(A_n, B_m)$

	1	2	3	4	5	6	7	8
A_8:	C	B	E	A	D	D	E	A
B_7:	D	E	C	D	B	D	A	
C_4:	E	D	D	A				

Rekurzivní pravidla:

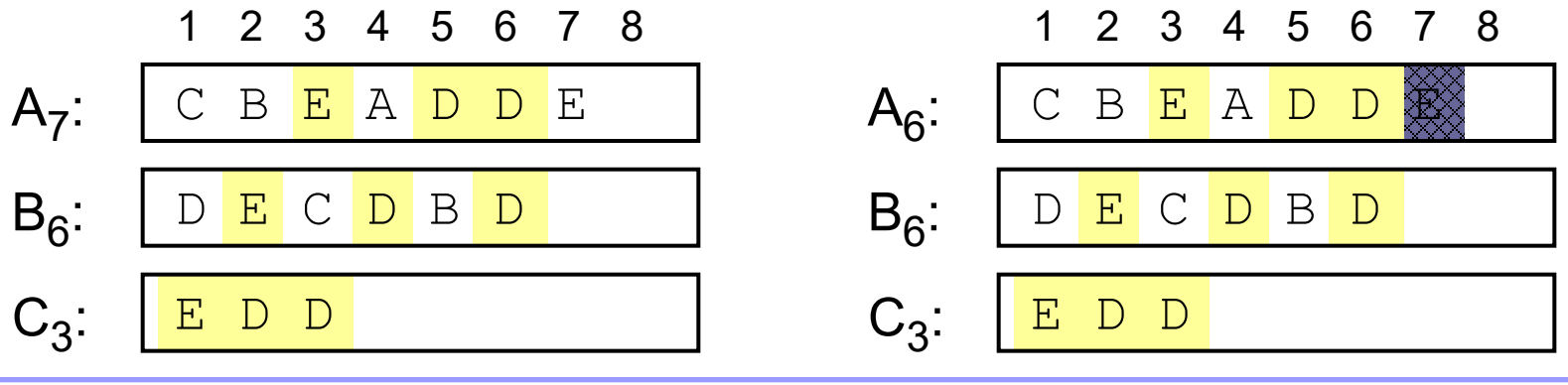
$(a_n = b_m) \Rightarrow (c_k = a_n = b_m) \ \& \ (C_{k-1} = LCS (A_{n-1}, B_{m-1}))$

	1	2	3	4	5	6	7	8
A_8:	C	B	E	A	D	D	E	A
B_7:	D	E	C	D	B	D	A	
C_4:	E	D	D	A				

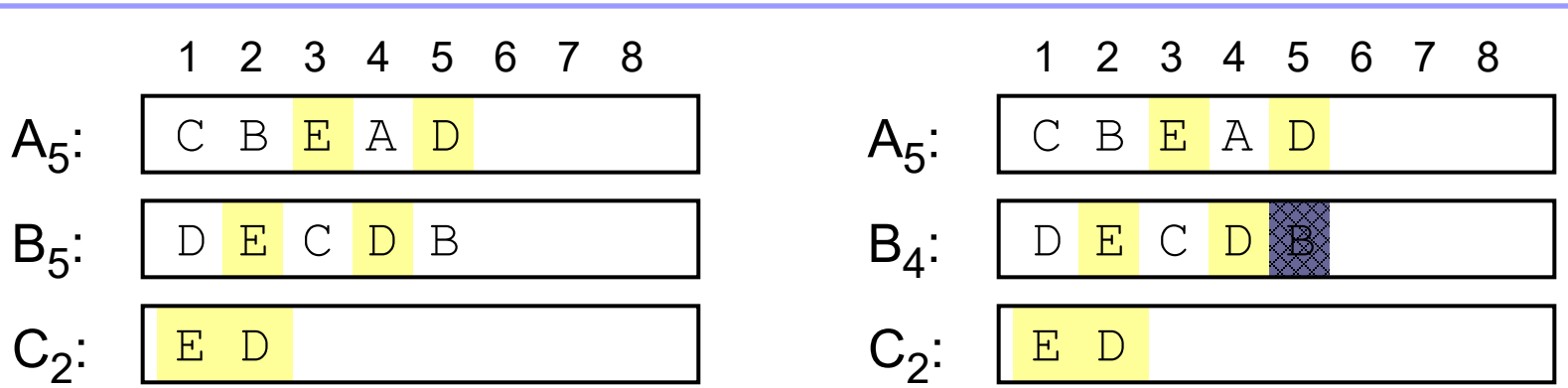
	1	2	3	4	5	6	7	8
A_7:	C	B	E	A	D	D	E	A
B_6:	D	E	C	D	B	D	A	
C_3:	E	D	D	A				

Nejdelší společná podposloupnost

$$(a_n != b_m) \ \& \ (c_k != a_n) \ ==> \ (C_k = LCS (A_{n-1}, B_m))$$



$$(a_n != b_m) \ \& \ (c_k != b_m) \ ==> \ (C_k = LCS (A_n, B_{m-1}))$$

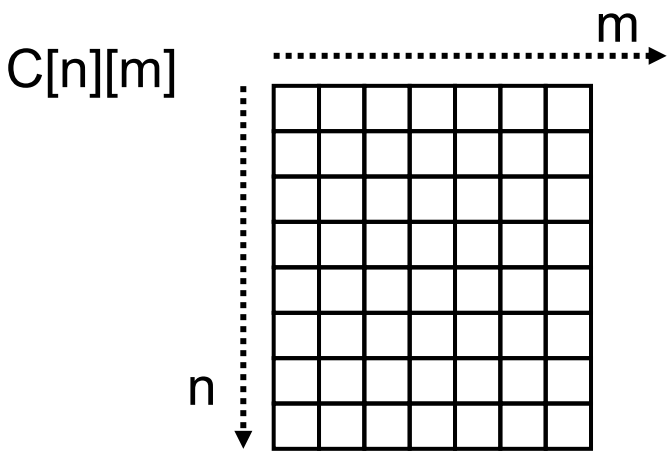


Nejdelší společná podposloupnost

Rekurzivní funkce – délka LCS

$$C(n,m) = \begin{cases} 0 & n = 0 \text{ or } m = 0 \\ C(n-1, m-1) + 1 & n > 0, m > 0, a_n = b_m \\ \max\{ C(n-1, m), C(n, m-1) \} & n > 0, m > 0, a_n \neq b_m \end{cases}$$

Strategie dynamického programování




```
for( a=1; a<=n; a++ )
  for( b=1; b<=m; b++ )
    C[a][b] = .... ;
}
```

Nejdelší společná podposloupnost

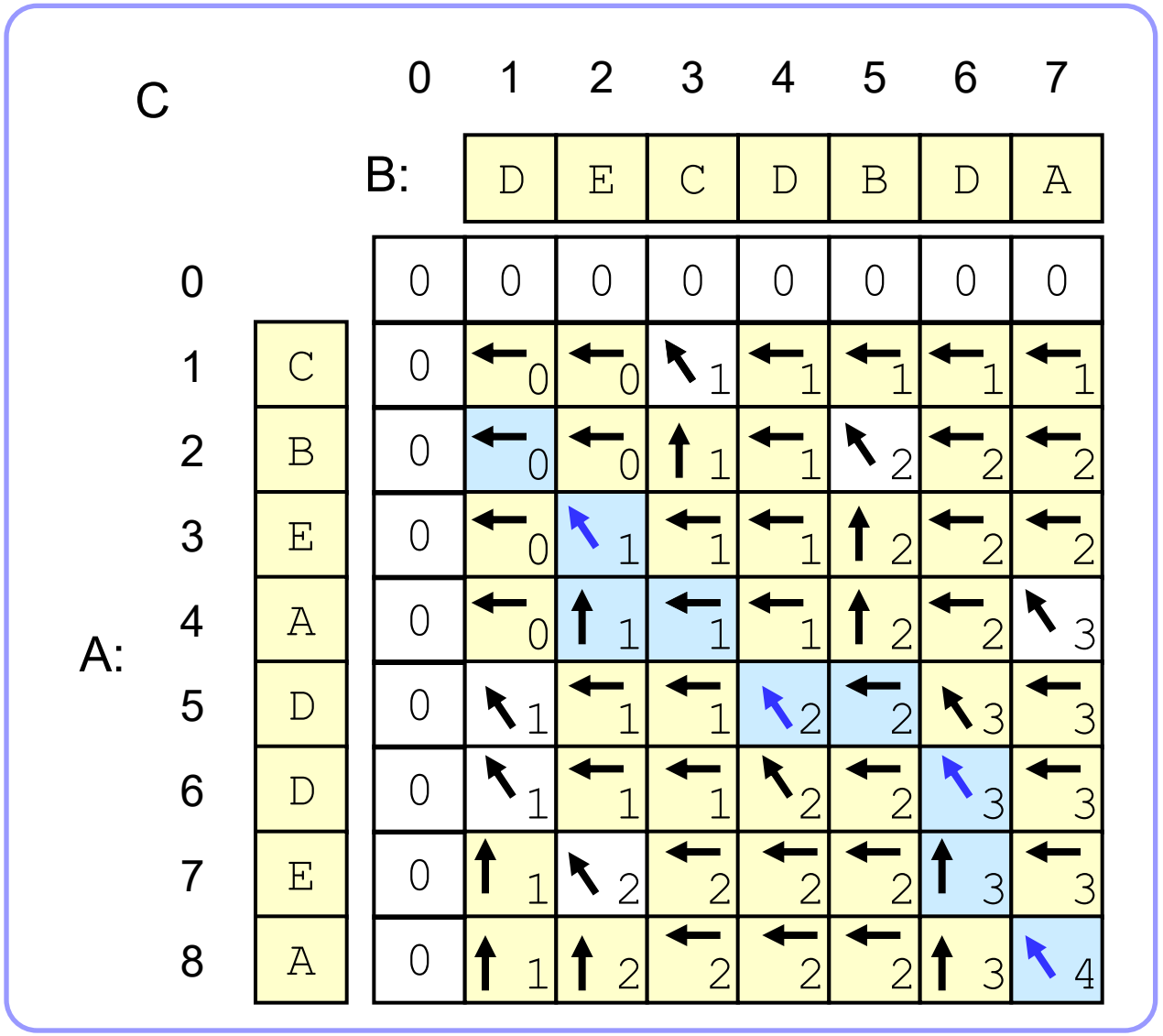
Konstrukce DP tabulek pro LCS

```
void findLCS() {
  for( int a=1; a<=n; a++ )
    for( int b=1; b<=m; b++ )
      if( A[a] == B[b] ) {
         C[a][b] = C[a-1][b-1]+1;
         arrows[a][b] = DIAG;
         }
      else
        if( C[a-1][b] > C[a][b-1] ) {
           C[a][b] = C[a-1][b];
           arrows[a][b] = UP;
        }
        else {
          C[a][b] = C[a][b-1];
          arrows[a][b] = LEFT;
 }      }
```

Nejdelší společná podposloupnost

Pole NSP pro

“CBEADDEA” a “DECDBDA”



Nejdelší společná podposloupnost

Výpis NSP -- rekurzivně :)

```
void outLCS( int a, int b ) {
if( a ==0 || b == 0 ) return;

if( arrows[a][b]  == DIAG ) {
    outLCS(a-1, b-1);       // recursion ...
    print(A[a]);            // ... reverses the sequence!
    }
else
    if( arrows[a][b] == UP )
        outLCS(a-1,b);
    else
        outLCS(a,b-1);
}
```

Audience Q&A