# Proiectarea Algoritmilor

## Curs 3a – Programare dinamică

# BIbliografie


- Cormen – Introducere în Algoritmi cap. 16
- Giumale – Introducere în Analiza Algoritmilor cap 4.5

# Programare dinamică

- Programare dinamică
  - Descriere generală
  - Algoritm generic
  - Caracteristici
- Exemplificare: Înmulțirea matricilor
- Exemplificare: Arbori optimi la căutare (AOC)
  - Definiții
  - Construcția AOC

# Programare dinamică

- Descriere generală
  - Soluții optime construite iterativ asamblând soluții optime ale unor probleme similare de dimensiuni mai mici.

- Algoritmi "clasici"
  - Înmulţirea unui şir de matrice
  - AOC
  - Algoritmul Floyd-Warshall care determină drumurile de cost minim dintre toate perechile de noduri ale unui graf.
  - Numere catalane
  - Viterbi

# Algoritm generic

- Programare dinamică (crit_optim, problema)
  - // fie $problema_0$ $problema_1$ … $problema_n$ astfel încât
  - // $problema_n$ = problema; $problema_i$ mai simplă decât $problema_{i+1}$
  - 1. Sol = soluții_inițiale(crit_optim, $problema_0$);
  - 2. **Pentru** *i* **de la** 1 **la** *n* // construcție soluții pentru
    // $problema_i$ folosind soluțiile problemelor precedente
  - 3. $Sol_i$ = calcul_soluții(Sol, Crit_optim, $Problema_i$);
    // determin soluția $problemei_i$
  - 4. Sol = Sol U $Sol_i$;
    // noile soluții se adaugă pentru a fi refolosite pe viitor
  - 5. s = soluție_pentru_$problema_n$(Sol);
    // selecție / construcție soluție finală
  - 6. **Întoarce** s;

# Caracteristici

- O soluție optimă a unei probleme conține soluții optime ale subproblemelor.
- Decompozabilitatea recursivă a problemei P în subprobleme similare $P = P_n, P_{n-1}, \ldots P_0$ care acceptă soluții din ce în ce mai simple.
- Suprapunerea problemelor (soluția unei probleme $P_i$ participă în procesul de construcție a soluțiilor mai multor probleme $P_k$ de talie mai mare $k > i$) – memoizare (se foloseşte un tablou pentru salvarea soluţiilor subproblemelor cu scopul de a nu le recalcula).
- În general se folosește o abordare bottom-up, de la subprobleme la probleme.

# Diferențe Greedy – Programare dinamică

*Programare lacomă*

- Sunt menținute doar soluțiile parțiale curente din care evoluează soluțiile parțiale următoare
- Soluțiile parțiale anterioare sunt eliminate
- Se poate obține o soluție neoptimă. (trebuie demonstrat că se poate aplica).

*Programare dinamică*

- La construcția unei soluții noi poate contribui orice altă soluție parțială generată anterior
- Se păstrează toate soluțiile parțiale
- Se obține soluția optimă.

# Diferenţe divide et impera – programare dinamică

*Divide et impera*

- abordare top-down – problema este descompusă în subprobleme care sunt rezolvate independent
- putem rezolva aceeaşi problemă de mai multe ori (dezavantaj potenţial foarte mare)

*Programare dinamică*

- abordare bottom-up - se porneşte de la sub-soluţii elementare şi se combină sub-soluţiile mai simple în sub-soluţii mai complicate, pe baza criteriului de optim
- se evită calculul repetat al aceleiaşi subprobleme prin memorarea rezultatelor intermediare (memoizare)

# Exemplu: Parantezarea matricelor (Chain Matrix Multiplication)

- Se dă un şir de matrice: $A_1, A_2, ..., A_n$.
- Care este numărul minim de înmulţiri de scalari pentru a calcula produsul:

$$A_1 \times A_2 \times ... \times A_n \ ?$$

- Să se determine una dintre parantezările care minimizează numărul de înmulţiri de scalari.

# Înmulţirea matricelor

- A(p, q) x B (q, r) => pqr înmulţiri de scalari.
- Dar înmulţirea matricelor este asociativă (deşi nu este comutativă).
- A(p, q) x B (q, r) x C(r, s)
  - (AB)C => pqr + prs  înmulţiri
  - A(BC) => qrs + pqs  înmulţiri
- Ex: p = 5, q = 4, r = 6, s = 2
  - (AB)C => 180  înmulţiri
  - A(BC) => 88  înmulţiri
- Concluzie: Parantezarea este foarte importantă!

# Soluţia banală

- Matrice: $A_1$, $A_2$, ..., $A_n$.
- Vector de dimensiuni: $p_0$, $p_1$, $p_2$, ... , $p_n$.
- $A_i(p_{i-1}, p_i) \rightarrow A_1(p_0, p_1), A_2(p_1, p_2), \ldots$
- Dacă folosim căutare exhaustivă şi vrem să construim toate parantezările posibile pentru a determina minimul: $\Omega(4^n / n^{3/2})$.
- Vrem o soluţie polinomială folosind P.D.

# Descompunere în subprobleme

- Încercăm să definim subprobleme identice cu problema originală, dar de dimensiune mai mică.
- $\forall\ 1 \le i \le j \le n$:
  - Notăm $A_{i,j} = A_i \times \ldots \times A_j$. $A_{i,j}$ are $p_{i-1}$ linii și $p_j$ coloane: $A_{i,j}(p_{i-1}, p_j)$
  - m[i, j] = numărul optim de înmulţiri pentru a rezolva subproblema $A_{i,j}$
  - s[i, j] = poziţia primei paranteze pentru subproblema $A_{i,j}$
  - Care e parantezarea optimă pentru $A_{i,j}$?
- Problema iniţială: $A_{1,n}$

# Combinarea subproblemelor

- Pentru a rezolva $A_{i,j}$
  - Trebuie găsit acel indice $i \leq k < j$ care asigură parantezarea optimă:

$$A_{i,\,j} = (A_i \times \ldots \times A_k) \times (A_{k+1} \times \ldots \times A_j)$$

$$A_{i,\,j} = A_{i,\,k} \times A_{k+1,\,j}$$

# Alegerea optimală

- Căutăm optimul dintre toate variantele posibile de alegere ($i \le k < j$)

- Pentru aceasta, trebuie însă ca şi subproblemele folosite să aibă soluţie optimală (adică $A_{i, k}$ şi $A_{k+1, j}$ să aibă soluție optimă).

# Substructura optimală

- Dacă ştim că alegerea optimală a soluţiei pentru problema $A_{i,j}$ implică folosirea subproblemelor ($A_{i,k}$ şi $A_{k+1,j}$) şi soluţia pentru $A_{i,j}$ este optimală, atunci şi soluţiile subproblemelor $A_{i,k}$ şi $A_{k+1,j}$ trebuie să fie optimale!

- Demonstraţie: Folosind metoda cut-and-paste (metodă standard de demonstrare a substructurii optimale pentru problemele de programare dinamică).

- Observație: Nu toate problemele de optim posedă această proprietate! Ex: drumul maxim dintr-un graf orientat.

# Definirea recursivă

- Folosind descompunerea în subprobleme, combinarea subproblemelor, alegerea optimală şi substructura optimală putem să rezolvăm problema prin programare dinamică.
- Următorul pas este să definim recursiv soluţia unei subprobleme.
- Vrem să găsim o formulă recursivă pentru m[i, j] şi s[i, j].

# Definirea recursivă (II)

$$m[i,j] = \begin{cases} 0 & i = j, \\ \min_{i \le k < j} (m[i,k] + m[k+1,j] + p_{i-1}p_k p_j) & i < j \end{cases}$$

- Cazurile de bază sunt m[i, i]
- Noi vrem să calculăm m[1, n]
- Cum alegem s[i, j] ?
- Bottom-up de la cele mai mici subprobleme la cea iniţială.

# Rezolvare bottom-up

$m[1,2], m[2,3], m[3,4], \ldots, m[n-3, n-2], m[n-2, n-1], m[n-1, n]$

$m[1,3], m[2,4], m[3,5], \ldots, m[n-3, n-1], m[n-2, n]$

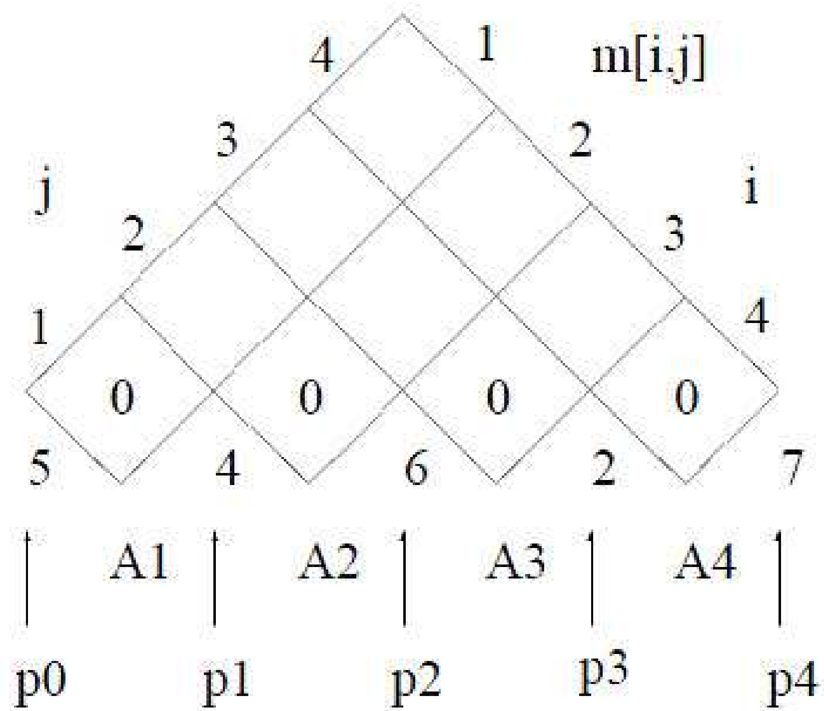
$m[1,4], m[2,5], m[3,6], \ldots, m[n-3, n]$

$\vdots$

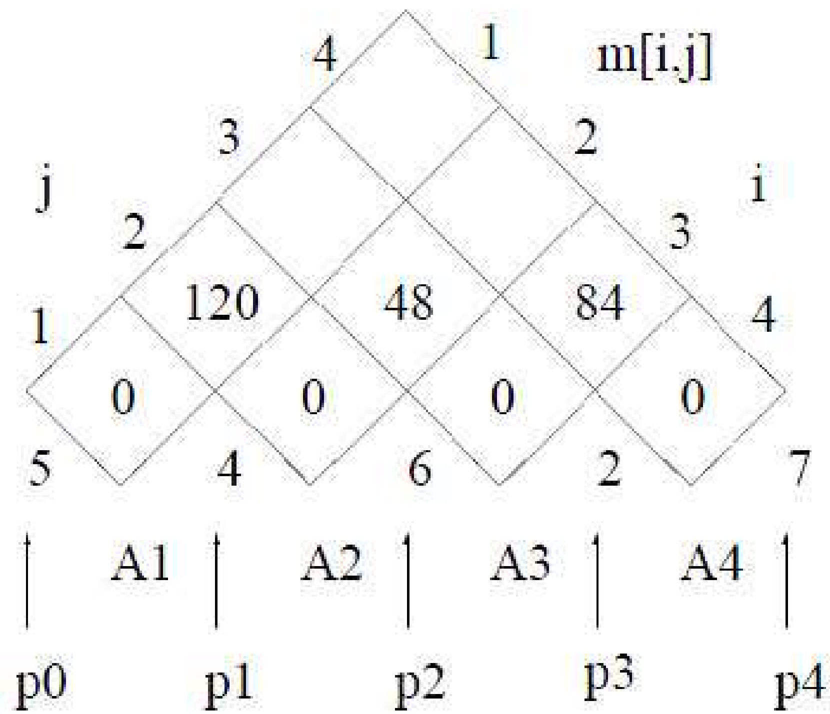
$m[1, n-1], m[2, n]$

$m[1, n]$

# Rezolvare - iniţializare



# Rezolvare – pas intermediar (I)

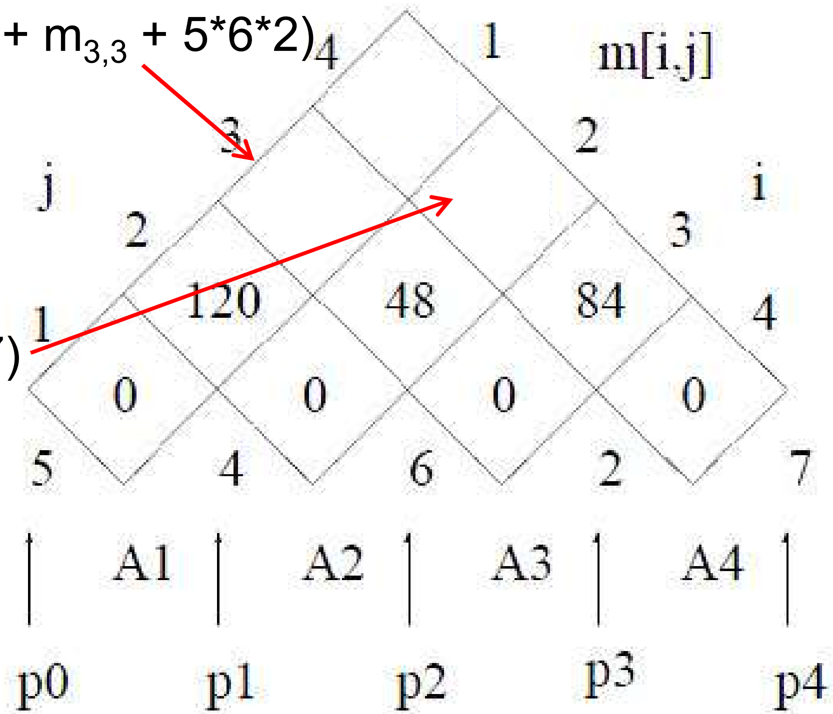


# Rezolvare – pas intermediar (II)

$$m[i,j] = \begin{cases} 0 & i = j, \\ \min_{i \le k < j} (m[i,k] + m[k+1,j] + p_{i-1}p_k p_j) & i < j \end{cases}$$

$A_{1,3} = \{A_1 * A_{2,3}, A_{1,2} * A_3\}$
$m_{1,3} = \min(m_{1,1} + m_{2,3} + 5*4*2, m_{1,2} + m_{3,3} + 5*6*2)$
$s_{1,3} = 1$

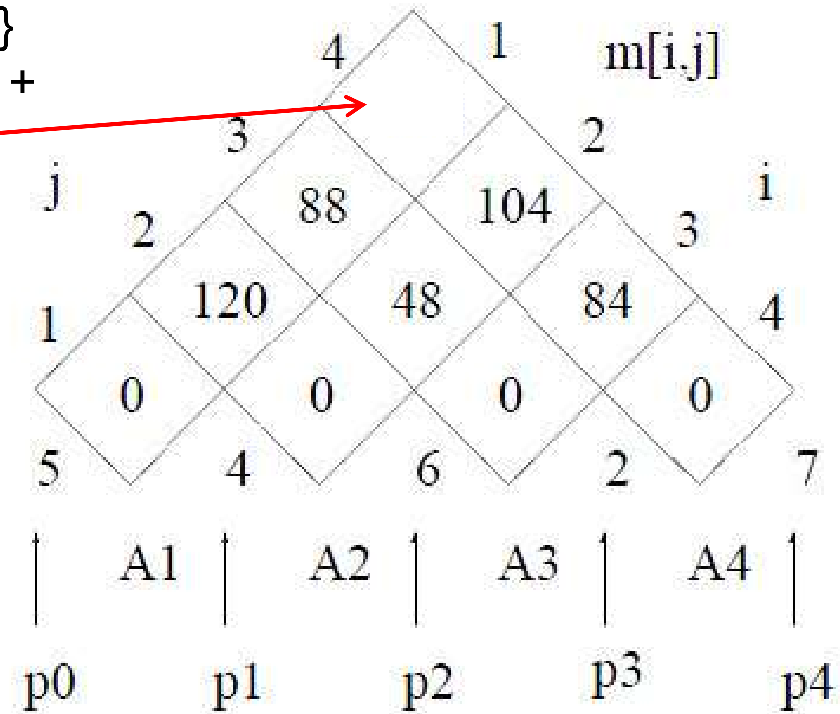
$A_{2,4} = \{A_2 * A_{3,4} , A_{2,3} * A_4\}$
$m_{2,4} = \min(m_{3,4} + 4*6*7, m_{2,3} + 4*2*7)$
$s_{2,4} = 3$



# Rezolvare – final

$$m[i,j] = \begin{cases} 0 & i = j, \\ \min_{i \le k < j} (m[i,k] + m[k+1,j] + p_{i-1}p_k p_j) & i < j \end{cases}$$

$A_{1,4} = \{A_1 * A_{2,4}, A_{1,2} * A_{3,4}, A_{1,3} * A_4\}$
$m_{1,4} = \min(m_{2,4} + 5*4*7,\ m_{1,2} + m_{3,4} + 5*6*7,\ m_{1,3} + 5*2*7) = 158$
$s_{1,4} = 3$




Parantezarea optimă este:

$(A_1(A_2A_3))A_4$

Numărul optim de operații

este: 158

# Pseudocod

- Înmulțire_matrice (p, n)
  - **Pentru** *i* **de la** 1 **la** *n* // inițializare
    - m[i, i] = s[i, i] = 0
  - **Pentru** **/** **de la** 2 **la** *n* // dimensiune problema
    - **Pentru** *i* **de la 1 la** *n – l + 1* // indice stânga
      - j = i + l – 1 // indice dreapta
      - m[i, j] = ∞ // pentru determinare minim
      - **Pentru** *k* **de la** *i* **la** *j – 1*
        - q = m[i, k] + m[k + 1, j] + p[i - 1] * p[k] * p[j]
        - **Dacă** q < m[i, j]
        - m[i, j] = q
        - s[i, j] = k
  - **Întoarce** *m* și *s*

# Complexitate

- Spaţială: $\Theta(n^2)$
  - Pentru memorarea soluţiilor subproblemelor
- Temporală: $O(n^3)$
  - Ns: Număr total de subprobleme: $O(n^2)$
  - Na: Număr total de alegeri la fiecare pas: $O(n)$
  - Complexitatea: $O(n^3)$ este de obicei egală cu Ns x Na

# ÎNTREBĂRI?