Metoda
Programării Dinamice
-continuare-

Metoda programării dinamice

- Probleme care presupun rezolvarea de relații de recurență
- De obicei aceste relații se obțin din respectarea unui principiu de optimalitate (subprobleme optime)

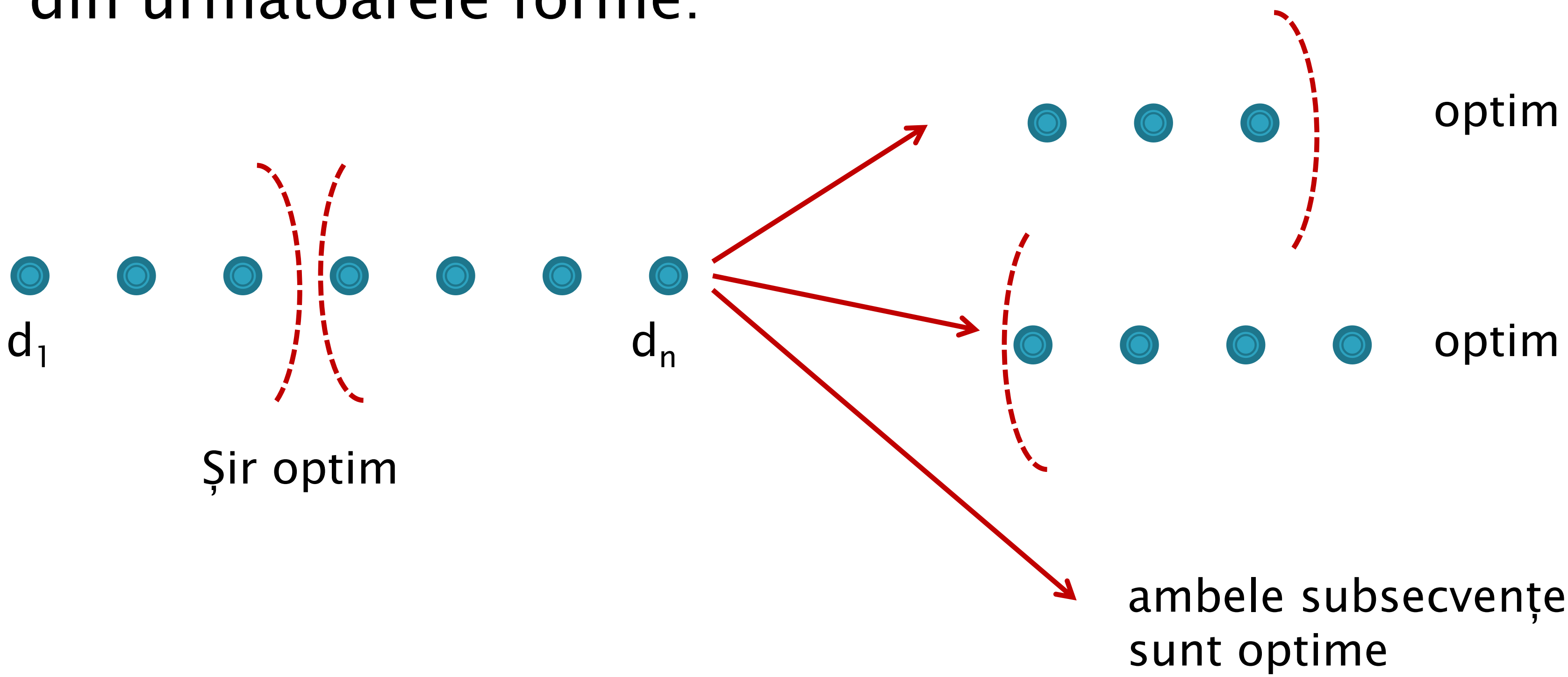
Metoda programării dinamice

- **Generalizează metoda Divide et Impera** – dependenţele nu au forma unui arbore, ci a unui PD-arbore.

Metoda programării dinamice

Fie soluția optimă $d_1, \ldots, d_n$ (șir finit de decizii, fiecare decizie depinzând de cele anterioare)

Principiul de optimalitate poate fi satisfăcut sub una din următoarele forme:



Metoda programării dinamice

Fie soluția optimă d_1, ..., d_n

Principiul de optimalitate poate fi satisfăcut sub una din următoarele forme:

(1) $d_1, d_2, ..., d_n$ optim $\Rightarrow$ $d_k, ..., d_n$ optim pentru subproblema corespunzatoare, $\forall\ 1 \le k \le n$

(2) $d_1, d_2, ..., d_n$ optim $\Rightarrow$ $d_1, ..., d_k$ optim, $\forall\ 1 \le k \le n$

(3) $d_1, d_2, ..., d_n$ optim $\Rightarrow$ $d_1, ..., d_k$ optim, $\forall\ 1 \le k \le n$

si

$d_k, ..., d_n$ optim, $\forall\ 1 \le k \le n$

Etape

- Stabilirea subproblemelor utile (de exemplu din principiul de optimalitate)
- Cum putem rezolva problema iniţială folosind subproblemele
- Care subprobleme le putem rezolva direct
- Relațiile de recurență
- Ordinea de calcul (parcurgerea PD-arborelui)

Exemple

Subşir crescător de lungime maximă

Se consideră vectorul $a = (a_1,..., a_n)$.

Să se determine lungimea maximă a unui subşir crescător din a şi un astfel de subşir de lungime maximă

Exemplu

Pentru

$$a = (8, 1, 7, 4, 6, 5, 11)$$

lungimea maximă este **4**, un subșir fiind

1, 4, 6, 11

Subşir crescător de lungime maximă

Principiu de optimalitate:

Dacă

$$a_{i1},\ a_{i2},\ \dots,\ a_{ip},$$

este un subșir optim care începe pe poziția `i1`, atunci:

$$a_{i2},\ \dots,\ a_{ip}$$

este un subșir optim care începe pe poziția `i2`;

Mai general

$$a_{ik},\ \dots,\ a_{ip}$$

este un subșir optim care începe pe poziția `ik`.

Subşir crescător de lungime maximă

Principiu de optimalitate

⬇

Subprobleme:

Calculăm pentru fiecare poziție i lungimea maximă a unui subşir crescător ce <u>**începe**</u> **pe poziția i** (cu elementul a_i)

Subşir crescător de lungime maximă

- **Subproblemă:**

  `lung[i]` = lungimea maximă a unui subşir crescător ce începe pe poziția i

- **Soluție problemă:**

$$\texttt{lmax} = \max\{\texttt{lung[i]} \mid \texttt{i} = 1,2,\ldots,\texttt{n}\}$$

Subşir crescător de lungime maximă

- **Subproblemă:**

  `lung[i]` = lungimea maximă a unui subşir crescător ce începe pe poziția i

- **Știm direct** `lung[n] = 1`

- **Relație de recurență**

  `lung[i] = 1 + max{lung[j] | j>i,` $a_i<a_j$`}`

- **Ordinea de calcul (parcurgerea PD-arborelui)**

  `i = n, n-1, … , 1`

Subşir crescător de lungime maximă

- Cum determinăm un subşir maxim?

Subşir crescător de lungime maximă

- Pentru a determina și **un subșir optim** putem memora în plus

`succ[i]` = indicele următorului element dintr-un subșir optim care începe pe poziția i (**n+1** dacă nu există)

= **indicele pentru care se realizează maximul în relaţia de recurenţă**

Subşir crescător de lungime maximă

a:	8	1	7	4	6	5	11
	1	2	3	4	5	6	7
lung :							
succ :							

Subşir crescător de lungime maximă

a:	8	1	7	4	6	5	11
	1	2	3	4	5	6	7
lung :							1
succ :							8

Subşir crescător de lungime maximă

a:	8	1	7	4	6	5	11
	1	2	3	4	5	6	7
lung :							1
succ :							8

Subşir crescător de lungime maximă

a:	8	1	7	4	6	5	11
	1	2	3	4	5	6	7
lung :						2	1
succ :						7	8

Subşir crescător de lungime maximă

a:	8	1	7	4	6	5	11
	1	2	3	4	5	6	7
lung :						2	1
succ :						7	8

Subşir crescător de lungime maximă

a:	8	1	7	4	6	5	11
	1	2	3	4	5	6	7
lung :					2	2	1
succ :					7	7	8

Subşir crescător de lungime maximă

a:	8	1	7	4	6	5	11
	1	2	3	4	5	6	7
lung :					2	2	1
succ :					7	7	8

Subşir crescător de lungime maximă

a:	8	1	7	4	6	5	11
	1	2	3	4	5	6	7
lung :				3	2	2	1
succ :				5	7	7	8

Subşir crescător de lungime maximă

a:	8	1	7	4	6	5	11
	1	2	3	4	5	6	7
lung :				3	2	2	1
succ :				5	7	7	8

Subşir crescător de lungime maximă

a:	8	1	7	4	6	5	11
	1	2	3	4	5	6	7
lung :			2	3	2	2	1
succ :			7	5	7	7	8

Subşir crescător de lungime maximă

a:	8	1	7	4	6	5	11
	1	2	3	4	5	6	7
lung :			2	3	2	2	1
succ :			7	5	7	7	8

Subşir crescător de lungime maximă

a:	8	1	7	4	6	5	11
	1	2	3	4	5	6	7
lung :		4	2	3	2	2	1
succ :		4	7	5	7	7	8

Subşir crescător de lungime maximă

a:	8	1	7	4	6	5	11
	1	2	3	4	5	6	7
lung :		4	2	3	2	2	1
succ :		4	7	5	7	7	8

Subşir crescător de lungime maximă

a:	8	1	7	4	6	5	11
	1	2	3	4	5	6	7
lung :	2	4	2	3	2	2	1
succ :	7	4	7	5	7	7	8

Subşir crescător de lungime maximă

a:	8	1	7	4	6	5	11
	1	2	3	4	5	6	7
lung :	2	4	2	3	2	2	1
succ :	7	4	7	5	7	7	8

Soluţie: lung = 4

Subşir crescător de lungime maximă

a:	8	1	7	4	6	5	11
	1	2	3	4	5	6	7
lung :	2	4	2	3	2	2	1
succ :	7	4	7	5	7	7	8

Subşir: 1,

Subşir crescător de lungime maximă

	1	2	3	4	5	6	7
a:	8	1	7	4	6	5	11
lung :	2	4	2	3	2	2	1
succ :	7	4	7	5	7	7	8

Subşir: 1,

Subşir crescător de lungime maximă

	1	2	3	4	5	6	7
a:	8	1	7	4	6	5	11
lung :	2	4	2	3	2	2	1
succ :	7	4	7	5	7	7	8

Subşir: 1, 4,

Subşir crescător de lungime maximă

	1	2	3	4	5	6	7
a:	8	1	7	4	6	5	11
lung :	2	4	2	3	2	2	1
succ :	7	4	7	5	7	7	8

Subşir: 1, 4, 6

Subşir crescător de lungime maximă

a:	8	1	7	4	6	5	11
	1	2	3	4	5	6	7
lung :	2	4	2	3	2	2	1
succ :	7	4	7	5	7	7	8

Subşir: 1, 4, 6, 11

Subşir crescător de lungime maximă

Algoritm

```
lmax= 1;
poz = n; //poz de inceput a sirului maxim
lung[n] = 1; succ[n] = n+1; //stim
for (int i=n-1;i>=1;i--){//ordine de calcul
    succ[i]= n+1; lung[i]=1;
    //formula de recurenta
    for(int j=i+1;j<=n; j++){
      if((a[i]<a[j]) && (1+lung[j]>lung[i])){
            lung[i]= 1 + lung[j];
            succ[i] = j;
      }
    }
    if(lung[i]> lmax){lmax = lung[i]; poz = i
}
```

```
//afisare subsir
for (int i=1;i<=lmax;i++){
    System.out.print(a[poz]+" ");
    poz = succ[poz];
}
```

```
//afisare subsir
for (int i=1;i<=lmax;i++){
    System.out.print(a[poz]+" ");
    poz = succ[poz];
}
```

- **Complexitate – O(n^2)**

- **Temă O(n log n)**

Subşir crescător de lungime maximă

- Numărul de subşiruri crescătoare de lungime maximă ale şirului

Subşir crescător de lungime maximă

- **`nr[i]`** = numărul de subşiruri crescătoare de lungime maximă care încep pe poziţia i

- În calculul lui **`nr[i]`** intervin doar indicii j cu **`j>i`**, $a_i<a_j$ pentru care **`lung[i] = lung[j] + 1`** (pentru care se obţine egalitate în relaţia de recurenţă pentru **`lung[i]`**, adică acei **`j`** care sunt succesori posibili ai lui **`i`**)

Subşir crescător de lungime maximă

a:	8	1	7	4	6	5	11
	1	2	3	4	5	6	7
lung :	2	**4**	2	3	2	2	1
succ :	7	4	7	5	7	7	8
nr:							1

Subşir crescător de lungime maximă

	1	2	3	4	5	6	7
a:	8	1	7	4	6	5	11
lung :	2	4	2	3	2	2	1
succ :	7	4	7	5	7	7	8
nr:						1	1

```
nr[6] = nr[7]
```

Subşir crescător de lungime maximă

	1	2	3	4	5	6	7
a:	8	1	7	4	6	5	11
lung :	2	4	2	3	2	2	1
succ :	7	4	7	5	7	7	8
nr:					1	1	1

```
nr[5] = nr[7]
```

Subşir crescător de lungime maximă

	1	2	3	4	5	6	7
a:	8	1	7	4	6	5	11
lung :	2	4	2	3	2	2	1
succ :	7	4	7	5	7	7	8
nr:				2	1	1	1

`nr[4] = nr[5] + nr[6]`

Subşir crescător de lungime maximă

	1	2	3	4	5	6	7
a:	8	1	7	4	6	5	11
lung :	2	4	2	3	2	2	1
succ :	7	4	7	5	7	7	8
nr:			1	2	1	1	1

```
nr[3] = nr[7]
```

Subşir crescător de lungime maximă

a:	8	1	7	4	6	5	11
	1	2	3	4	5	6	7
lung :	2	4	2	3	2	2	1
succ :	7	4	7	5	7	7	8
nr:		2	1	2	1	1	1

```
nr[2] = nr[4]
```

Subşir crescător de lungime maximă

a:	8	1	7	4	6	5	11
	1	2	3	4	5	6	7
lung :	2	4	2	3	2	2	1
succ :	7	4	7	5	7	7	8
nr:		2	1	2	1	1	1

```
nr[2] = nr[4]
```

Subşir crescător de lungime maximă

a:	8	1	7	4	6	5	11
	1	2	3	4	5	6	7
lung :	2	4	2	3	2	2	1
succ :	7	4	7	5	7	7	8
nr:	1	2	1	2	1	1	1

```
nr[1] = nr[7]
```

Subşir crescător de lungime maximă

	1	2	3	4	5	6	7
a:	8	1	7	4	6	5	11
lung :	2	4	2	3	2	2	1
succ :	7	4	7	5	7	7	8
nr:	1	2	1	2	1	1	1

`nr[poz]`

Altă soluție

Principiu de optimalitate:

Dacă

$$a_{i1},\ a_{i2},\ \ldots,\ a_{ip},$$

este un subșir optim care se termină pe poziția `ip`, atunci

$$a_{i1},\ \ldots,\ a_{ik}$$

este un subșir optim care se termină pe poziția `ik`.

Subproblemă:

Calculăm pentru fiecare poziție i lungimea maximă a unui subşir crescător ce **se termină pe poziția i**

Subşir crescător de lungime maximă

a:	8	1	7	4	6	5	11
	1	2	3	4	5	6	7
lung :							
pred :							

Subşir crescător de lungime maximă

a:	8	1	7	4	6	5	11
	1	2	3	4	5	6	7
lung :	1						
pred :	0						

Subşir crescător de lungime maximă

a:	8	1	7	4	6	5	11
	1	2	3	4	5	6	7
lung :	1	1					
pred :	0	0					

Subşir crescător de lungime maximă

a:	8	1	7	4	6	5	11
	1	2	3	4	5	6	7
lung :	1	1	2				
pred :	0	0	2				

Subşir crescător de lungime maximă

a:	8	1	7	4	6	5	11
	1	2	3	4	5	6	7
lung :	1	1	2	2			
pred :	0	0	2	2			

Subşir crescător de lungime maximă

a:	8	1	7	4	6	5	11
	1	2	3	4	5	6	7
lung :	1	1	2	2	3		
pred :	0	0	2	2	4		

Subşir crescător de lungime maximă

a:	8	1	7	4	6	5	11
	1	2	3	4	5	6	7
lung :	1	1	2	2	3	3	
pred :	0	0	2	2	4	4	

Subşir crescător de lungime maximă

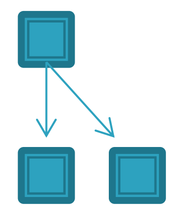
a:	8	1	7	4	6	5	11
	1	2	3	4	5	6	7
lung :	1	1	2	2	3	3	4
pred :	0	0	2	2	4	4	6

Trasee de sumă maximă

- Se consideră un triunghi de numere naturale **t** cu n linii.

Să se determine cea mai mare sumă pe care o putem forma dacă ne deplasăm în triunghi și adunăm numerele din celulele de pe traseu, regulile de deplasare fiind următoarele:

- pornim de la numărul de pe prima linie
- din celula (i,j) putem merge doar în (i+1,j) sau (i+1,j+1).



Să se indice și un traseu de sumă maximă

Trasee de sumă maximă

- **Exemplu**

1

6 **2**

1 2 **10**

3 4 **7** 2

Câte astfel de trasee există?

Trasee de sumă maximă

- **Exemplu**

```
1
6 2
1 2 10
3 4  7 2
```

1
6 **2**
1 2 **10**
3 4 **7** 2

- **Greedy** – nu obţinem soluţia optimă

1
6 2
1 **2** 10
3 4 **7** 2

Trasee de sumă maximă

- **Principiu de optimalitate:**

  Dacă

  $$(i_1 = 1,\ j_1 = 1),\ (i_2,\ j_2),\ \dots\ ,(i_n,\ j_n)$$

  este un traseu optim

Trasee de sumă maximă

- **Subproblemă:**

  `s[i][j]` = suma maximă pe care o putem obține dacă **pornim din celula (i, j)**

- **Soluție problemă**
- **Ştim direct**
- **Relație de recurență**
- **Ordinea de rezolvare a recurenţelor (parcurgerea PD-arborelui)**

Trasee de sumă maximă

- Pentru a memora și un traseu

  `u[i][j]` = coloana pe care ne deplasăm din celula (i, j) pe linia i+1 într-un traseu optim

sau

reconstituim traseul folosind relația de recurență = ne deplasăm mereu în celula permisă de pe linia următoare cu s (!!nu t) maxim

Trasee de sumă maximă

- **Exemplu**

```
1
6 2
1 2 10
3 4  7 2                  3   4   7   2
    t                         s
```

Trasee de sumă maximă

- **Exemplu**

1								
6	2							
1	2	10			5			
3	4	7	2		3	4	7	2

t **s**

Trasee de sumă maximă

- **Exemplu**

```
1
6 2
1 2 10
3 4  7 2
    t
```

```
5   9
3   4   7   2
   s
```

Trasee de sumă maximă

- **Exemplu**

```
1
6  2
1  2  10                          5   9  17
3  4   7  2                       3   4   7   2
      t                               s
```

Trasee de sumă maximă

- **Exemplu**

1							
6	2			15	19		
1	2	10		5	9	17	
3	4	7	2	3	4	7	2

t **s**

Trasee de sumă maximă

- **Exemplu**

1					20			
6	2				15	19		
1	2	10			5	9	17	
3	4	7	2		3	4	7	2
	t					**s**		

Trasee de sumă maximă

- **traseu**

1				**20**			
6	2			15	19		
1	2	10		5	9	17	
3	4	7	2	3	4	7	2

t **s**

Trasee de sumă maximă

- **traseu**

```
1                      20
6 2                    15 19
1 2 10                 5  9 17
3 4  7 2               3  4  7  2
    t                     s
```

Trasee de sumă maximă

- **traseu**

1					**20**			
6	2				15	**19**		
1	2	10			5	9	**17**	
3	4	7	2		3	4	7	2

t **s**

Trasee de sumă maximă

- **traseu**

1			
6	2		
1	2	10	
3	4	7	2

t

20			
15	**19**		
5	9	**17**	
3	4	**7**	2

s

Trasee de sumă maximă

Algoritm

```
//stim
for(int i=0;i<n;i++) {//ultima linie n-1
    s[n-1][i]=t[n-1][i]; //u[n-1][i]=n+1;
}
```

```
//stim
for(int i=0;i<n;i++) {//ultima linie n-1
    s[n-1][i]=t[n-1][i]; //u[n-1][i]=n+1;
}
//ordine de calcul
for(int i=n-2;i>=0;i--)
  for(int j=0;j<=i;j++)
```

```
//stim
for(int i=0;i<n;i++) {//ultima linie n-1
    s[n-1][i]=t[n-1][i]; //u[n-1][i]=n+1;
}
//ordine de calcul
for(int i=n-2;i>=0;i--)
  for(int j=0;j<=i;j++)
     if(s[i+1][j]>s[i+1][j+1])  {
       s[i][j]=t[i][j]+s[i+1][j];
       //u[i][j]=j; //coloana pe care ne deplasam
     }
```

```
//stim
for(int i=0;i<n;i++) {//ultima linie n-1
    s[n-1][i]=t[n-1][i]; //u[n-1][i]=n+1;
}
//ordine de calcul
for(int i=n-2;i>=0;i--)
  for(int j=0;j<=i;j++)
     if(s[i+1][j]>s[i+1][j+1])  {
       s[i][j]=t[i][j]+s[i+1][j];
       //u[i][j]=j; //coloana pe care ne deplasam
     }
     else{
        s[i][j]=t[i][j]+s[i+1][j+1];
        //u[i][j]=j+1;//col. pe care ne deplasam
     }
```

```
//traseu reconstituit din relatia de recurenta
j=0;
for(int i=0;i<n-1;i++){
    out.println(" ("+(i+1)+" "+(j+1)+")");
    if(s[i][j] == t[i][j]+s[i+1][j+1])
        j++;
    //altfel coloana ramane j
}
out.println(" ("+n+" "+(j+1)+")");//ultima linie
```

```
//traseu reconstituit din relatia de recurenta
j=0;
for(int i=0;i<n-1;i++){
    out.println(" ("+(i+1)+" "+(j+1)+")");
    if(s[i][j] == t[i][j]+s[i+1][j+1])
        j++;
    //altfel coloana ramane j
}
out.println(" ("+n+" "+(j+1)+")");//ultima linie

//traseu reconstituit folosind u
j=0;
for(int i=0;i<=n-1;i++){//pe fiecare linie
  out.println(" ("+(i+1)+" "+(j+1)+")");//transl. de la 1
  j=u[i][j]; //coloana urmatoare
}
```

Trasee de sumă maximă

- **Complexitate – $O(n^2)$**

Trasee de sumă maximă

- Principiu de optimalitate – <u>Altă variantă</u>

Trasee de sumă maximă

- **Principiu de optimalitate – Altă variantă**

  Dacă

  $$(i_1=1,\ j_1=1),\ (i_2,\ j_2),\ \ldots\ ,(i_n,\ j_n)$$

  este un traseu optim atunci

  $$(i_1,\ j_1),\ \ldots\ ,(i_k,\ j_k)$$

  este un traseu optim **pentru a ajunge** în celula $(i_k,\ j_k)$ pornind din $(i_1,\ j_1)$

Trasee de sumă maximă

- **Principiu de optimalitate – <u>Altă variantă</u>**

  Dacă

  $$(i_1=1, j_1=1), (i_2, j_2), \ldots ,(i_n, j_n)$$

  este un traseu optim atunci

  $$(i_1, j_1), \ldots ,(i_k, j_k)$$

  este un traseu optim **pentru a ajunge** în celula (i_k, j_k) pornind din (i_1, j_1)

- **Subproblemă:**

  Calculăm pentru o poziție (i, j) suma maximă pe care o putem obține dacă **ajungem în celula (i, j) pornind din (1,1)**

Trasee de sumă maximă

- **Subproblemă:**

  `s[i][j]` = suma maximă pe care o putem obține **pornind din (1,1)** și ajungând în **celula (i, j)**

- **Soluție problemă**
- **Știm direct**
- **Relație de recurență**
- **Ordinea de parcurgere a PD-arborelui (ordinea de calcul)**

Trasee de sumă maximă

Numărul de trasee optime – Temă

Înmulţirea optimă a unui şir de matrice

- Avem de calculat produsul de matrice

  $A_1 \cdot A_2 \cdot \ldots \cdot A_n$,

  unde dimensiunile lor sunt respectiv

  $(d_1,d_2), (d_2,d_3),\ldots, (d_n,d_{n+1})$.

  Ştiind că înmulţirea matricelor este **asociativă**, se pune problema **ordinii** în care trebuie înmulţite matricele astfel încât numărul de înmulţiri **elementare** să fie **minim.**

- Produsul matricelor A(m,n) şi B(n,p) necesită **m·n·p înmulţiri elementare.**

Înmulţirea optimă a unui şir de matrice

- **Exemplu**

  $A_1(100,1)$, $A_2(1,100)$, $A_3(100,1)$, $A_4(1,100)$

  - $(A_1 \cdot A_2) \cdot (A_3 \cdot A_4) \longrightarrow$ 1.020.000
  - $(A_1 \cdot (A_2 \cdot A_3)) \cdot A_4 \longrightarrow$ 10.200

Înmulţirea optimă a unui şir de matrice

- **Principiu de optimalitate**

ultima operaţie

$$(A_i \cdot \ldots \cdot A_k) \cdot (A_{k+1} \cdot \ldots \cdot A_j) \text{ optim } \Rightarrow$$

Înmulţirea optimă a unui şir de matrice

- **Principiu de optimalitate**

ultima operaţie

$$(A_i \cdot \ldots \cdot A_k) \cdot (A_{k+1} \cdot \ldots \cdot A_j) \text{ optim } \Rightarrow$$

$$A_i \cdot \ldots \cdot A_k \text{ și } A_{k+1} \cdot \ldots \cdot A_j \text{ parantezate optim}$$

Înmulţirea optimă a unui şir de matrice

- **Subproblemă:**

  `cost[i][j]` = numărul minim de înmulţiri elementare pentru calculul produsului $A_i \cdot \ldots \cdot A_j$

- **Soluție**

- **Ştim direct**

- **Relație de recurență**

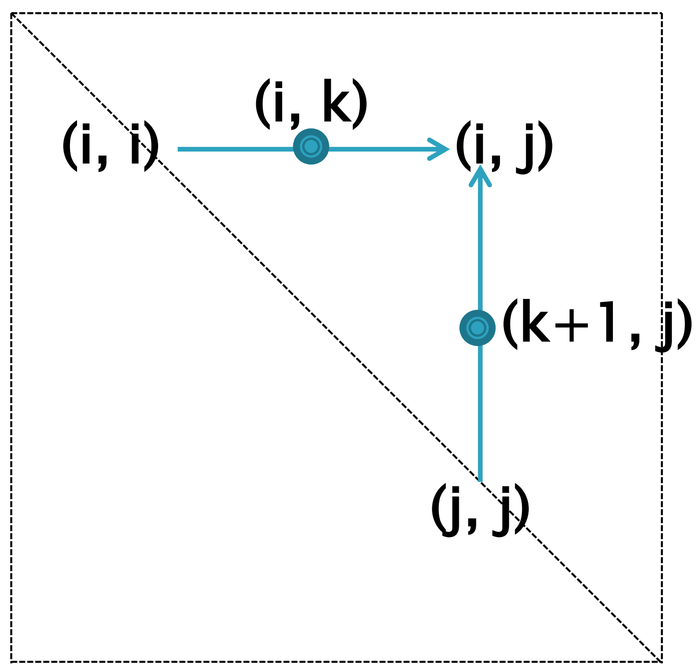
- **Ordinea de parcurgere a PD-arborelui (ordinea de calcul)**

Înmulţirea optimă a unui şir de matrice

- **Relație de recurență**

`cost[i][j]= min{cost[i][k]+cost[k+1][j]+`$d_i \cdot d_{k+1} \cdot d_{j+1}$
`| i≤k<j}.`

- **Graf de dependenţe**



Înmulţirea optimă a unui şir de matrice

- **Ordinea de parcurgere (ordinea de calcul)**

  - **Varianta 1:** Parcurgem în ordine coloanele 2,...,n, iar pe fiecare coloană j mergem în sus de la diagonală până la prima linie

Înmulţirea optimă a unui şir de matrice

- **Ordinea de parcurgere (ordinea de calcul)**

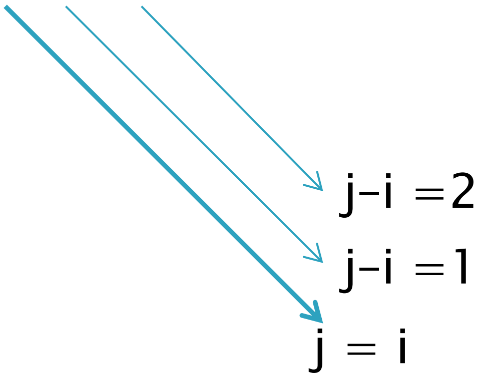
  - **Varianta 1:** Parcurgem în ordine coloanele 2,...,n, iar pe fiecare coloană j mergem în sus de la diagonală până la prima linie

```
for (j=2; j<=n; j++)
  for (i=j-1; i>=1; i--)
     calculeaza cost[i][j] dupa relatia de recurenta
     fie k valoarea pentru care se realizează minimul
     cost[j][i]=k

scrie cost[1][n]
```

Înmulţirea optimă a unui şir de matrice

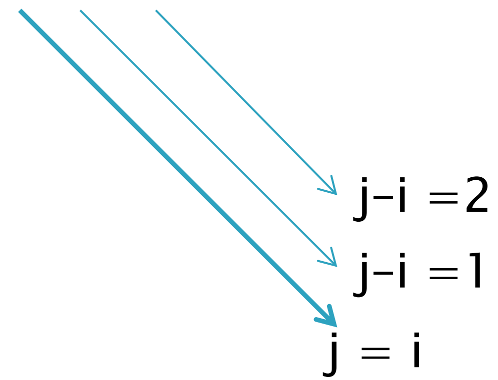
- **Ordinea de parcurgere (ordinea de calcul)**
  - **Varianta 2:** Parcurgem indicii în ordinea modulului diferenței j - i = **paralel cu diagonala principală.**



Înmulţirea optimă a unui şir de matrice

- **Ordinea de parcurgere (ordinea de calcul)**

  ◦ **Varianta 2:** Parcurgem indicii în ordinea modulului diferenței j – i = **paralel cu diagonala principală.**




```
for (dif=1; dif<=n-1; dif++)
 for (i=1; i<=n-dif; i++)
    j = i+dif
    calculeaza cost[i][j] dupa relatia de recurenta
    fie k valoarea pentru care se realizează minimul
    cost[j][i]=k

scrie cost[1][n]
```

Înmulţirea optimă a unui şir de matrice

- **Exemplu**

$A_1(100,1)$, $A_2(1,100)$, $A_3(100,1)$, $A_4(1,100)$

cost[i][j]= min{cost[i][k]+cost[k+1][j]+$d_i \cdot d_{k+1} \cdot d_{j+1}$ | i≤k<j}.

cost:

Înmulţirea optimă a unui şir de matrice

- **Exemplu**

$A_1(100,1)$, $A_2(1,100)$, $A_3(100,1)$, $A_4(1,100)$

cost[i][j]= min{cost[i][k]+cost[k+1][j]+$d_i \cdot d_{k+1} \cdot d_{j+1}$ | i≤k<j}.

cost:

0			
	0		
		0	
			0

Înmulţirea optimă a unui şir de matrice

Exemplu

$A_1(100,1)$, $A_2(1,100)$, $A_3(100,1)$, $A_4(1,100)$

cost[i][j]= min{cost[i][k]+cost[k+1][j]+$d_i \cdot d_{k+1} \cdot d_{j+1}$ | i≤k<j}.

cost:

0			
	0		
		0	
			0

cost[i][i+1]= costul optim pentru $A_i \cdot A_{i+1}$ =

= cost[i][i]+cost[i+1][i+1]+$d_i \cdot d_{i+1} \cdot d_{i+2}$ = $d_i \cdot d_{i+1} \cdot d_{i+2}$

cost[i+1][i] = k = i

Înmulţirea optimă a unui şir de matrice

Exemplu

$A_1(100,1)$, $A_2(1,100)$, $A_3(100,1)$, $A_4(1,100)$

cost[i][j]= min{cost[i][k]+cost[k+1][j]+$d_i \cdot d_{k+1} \cdot d_{j+1}$ | i≤k<j}.

cost:

0	10000		
1	0	100	
	2	0	10000
		3	0

A_1A_2: cost[1][2]= $d_1 \cdot d_2 \cdot d_3$ = 100 · 1 · 100

A_2A_3: cost[2][3]= $d_2 \cdot d_3 \cdot d_4$ = 1 · 100 · 1

A_3A_4: cost[3][4]= $d_3 \cdot d_4 \cdot d_5$ = 100 · 1 · 100

Înmulţirea optimă a unui şir de matrice

- **Exemplu**

$A_1(100,1)$, $A_2(1,100)$, $A_3(100,1)$, $A_4(1,100)$

cost[i][j]= min{cost[i][k]+cost[k+1][j]+$d_i \cdot d_{k+1} \cdot d_{j+1}$ | i≤k<j}.

cost:

0	10000		
1	0	100	
	2	0	10000
		3	0

cost[i][i+2]= costul optim pentru $A_i A_{i+1} A_{i+2}$=

Înmulţirea optimă a unui şir de matrice

- **Exemplu**

$A_1(100,1)$, $A_2(1,100)$, $A_3(100,1)$, $A_4(1,100)$

cost[i][j]= min{cost[i][k]+cost[k+1][j]+$d_i \cdot d_{k+1} \cdot d_{j+1}$ | i≤k<j}.

cost:

0	10000		
1	0	100	
	2	0	10000
		3	0

cost[i][i+2]= costul optim pentru $A_iA_{i+1}A_{i+2}$=

= min {cost $A_i(A_{i+1}A_{i+2})$, cost $(A_iA_{i+1})A_{i+2}$ }=

=

Înmulţirea optimă a unui şir de matrice

- **Exemplu**

$A_1(100,1)$, $A_2(1,100)$, $A_3(100,1)$, $A_4(1,100)$

cost[i][j]= min{cost[i][k]+cost[k+1][j]+$d_i \cdot d_{k+1} \cdot d_{j+1}$ | i≤k<j}.

cost:

0	10000		
1	0	100	
	2	0	10000
		3	0

cost[i][i+2]= costul optim pentru $A_iA_{i+1}A_{i+2}$=

= min {cost $A_i(A_{i+1}A_{i+2})$, cost $(A_iA_{i+1})A_{i+2}$ }=

= min{cost[i][i]+ cost[i+1][i+2]+ $d_i \cdot d_{i+1} \cdot d_{i+3}$,

cost[i][i+1]+ cost[i+2][i+2]+ $d_i \cdot d_{i+2} \cdot d_{i+3}$}

Înmulţirea optimă a unui şir de matrice

- **Exemplu**

$A_1(100,1)$, $A_2(1,100)$, $A_3(100,1)$, $A_4(1,100)$

cost[i][j]= min{cost[i][k]+cost[k+1][j]+$d_i \cdot d_{k+1} \cdot d_{j+1}$ | i≤k<j}.

cost:

0	10000	200	
1	0	100	
1	2	0	10000
		3	0

$A_1A_2A_3$ = min {cost $A_1(A_2\ A_3)$, cost $(A_1A_2)A_3$ } =

= min{cost[1][1]+cost[2][3]+ $d_1 \cdot d_2 \cdot d_4$,

cost[1][2]+ cost[3][3]+ $d_1 \cdot d_3 \cdot d_4$}

= min{0+ 100+ 100 · 1· 1 , 10000+ 0 + 100· 100· 1} = 200

pentru k = 1

Înmulţirea optimă a unui şir de matrice

Exemplu

$A_1(100,1)$, $A_2(1,100)$, $A_3(100,1)$, $A_4(1,100)$

cost[i][j]= min{cost[i][k]+cost[k+1][j]+$d_i \cdot d_{k+1} \cdot d_{j+1}$ | i≤k<j}.

cost:

0	10000	200	
1	0	100	200
1	2	0	10000
	3	3	0

$A_2A_3A_4$ = min {cost $A_2(A_3 A_4)$, cost $(A_2A_3)A_4$ } =

= min{cost[2][2]+ cost[3][4]+ $d_2 \cdot d_3 \cdot d_5$,

cost[2][3]+ cost[4][4]+ $d_2 \cdot d_4 \cdot d_5$}

= min{0+ 10000+ 1· 100· 1 , 100+ 0 + 1· 1· 100} = 200

pentru k = 3

Înmulţirea optimă a unui şir de matrice

Exemplu

$A_1(100,1)$, $A_2(1,100)$, $A_3(100,1)$, $A_4(1,100)$

cost[i][j]= min{cost[i][k]+cost[k+1][j]+$d_i \cdot d_{k+1} \cdot d_{j+1}$ | i≤k<j}.

cost:

0	10000	200	
1	0	100	200
1	2	0	10000
	3	3	0

$$A_1A_2A_3A_4 = \min \{A_1(A_2A_3A_4),\ (A_1A_2)(A_3A_4),\ (A_1A_2A_3)A_4\}$$

Înmulţirea optimă a unui şir de matrice

- **Exemplu**

$A_1(100,1)$, $A_2(1,100)$, $A_3(100,1)$, $A_4(1,100)$

cost[i][j]= min{cost[i][k]+cost[k+1][j]+$d_i \cdot d_{k+1} \cdot d_{j+1}$ | i≤k<j}.

cost:

0	10000	200	
1	0	100	200
1	2	0	10000
	3	3	0

$$A_1A_2A_3A_4 = \min\{A_1(A_2A_3A_4),\ (A_1A_2)(A_3\ A_4),\ (A_1A_2A_3)A_4\}$$

cost[1][4]= min{cost[1][1]+ cost[2][4]+ $d_1 \cdot d_2 \cdot d_5$,

cost[1][2]+ cost[3][4]+ $d_1 \cdot d_3 \cdot d_5$,

cost[1][3]+ cost[4][4]+ $d_1 \cdot d_4 \cdot d_5$

}

Înmulţirea optimă a unui şir de matrice

Exemplu

$A_1(100,1)$, $A_2(1,100)$, $A_3(100,1)$, $A_4(1,100)$

$$cost[i][j] = \min\{cost[i][k]+cost[k+1][j]+d_i \cdot d_{k+1} \cdot d_{j+1} \mid i \le k < j\}.$$

cost:

0	10000	200	10200
1	0	100	200
1	2	0	10000
1	3	3	0

$A_1A_2A_3A_4 = \min\{A_1(A_2A_3A_4),\ (A_1A_2)(A_3\ A_4),\ (A_1A_2A_3)A_4\}$

$cost[1][4] = \min\{0 + 200 + 100 \cdot 1 \cdot 100,$

$10000 + 10000 + 100 \cdot 100 \cdot 100,$

$200 + 0 + 100 \cdot 1 \cdot 100$

$\}$ se obţine pentru k = 1

Înmulţirea optimă a unui şir de matrice

- **Exemplu**

$A_1(100,1)$, $A_2(1,100)$, $A_3(100,1)$, $A_4(1,100)$

cost[i][j]= min{cost[i][k]+cost[k+1][j]+$d_i \cdot d_{k+1} \cdot d_{j+1}$ | $i \le k < j$}.

cost:

0	10000	200	10200
1	0	100	200
1	2	0	10000
1	3	3	0

Înmulţirea optimă a unui şir de matrice

- **Afișarea unei parantezări optime**

Înmulţirea optimă a unui şir de matrice

- **Afişarea unei parantezări optime –** recursiv din relaţia de recurenţă, folosind indicele k memorat

Înmulţirea optimă a unui şir de matrice

- **Afişarea unei parantezări optime –** recursiv din relaţia de recurenţă, folosind indicele k memorat

```
void sol(int p,int u){
    if (p==u)
        out.print(p);
    else {








    }
}
```

Înmulţirea optimă a unui şir de matrice

- **Afişarea unei parantezări optime –** recursiv din relaţia de recurenţă, folosind indicele k memorat

```
void sol(int p,int u){
   if (p==u)
        out.print(p);
   else {
        k = cost[u][p];



  }
}
```

Înmulţirea optimă a unui şir de matrice

- **Afişarea unei parantezări optime –** recursiv din relaţia de recurenţă, folosind indicele k memorat

```
void sol(int p,int u){
    if (p==u)
        out.print(p);
    else {
        k = cost[u][p];
        out.print ('(');
        sol(p,k);




    }
}
```

Înmulţirea optimă a unui şir de matrice

- **Afişarea unei parantezări optime –** recursiv din relaţia de recurenţă, folosind indicele k memorat

```
void sol(int p,int u){
    if (p==u)
        out.print(p);
    else {
        k = cost[u][p];
        out.print ('(');
        sol(p,k);
        out.print (',');




    }
}
```

Înmulţirea optimă a unui şir de matrice

- **Afişarea unei parantezări optime –** recursiv din relaţia de recurenţă, folosind indicele k memorat

```
void sol(int p,int u){
    if (p==u)
        out.print(p);
    else {
        k = cost[u][p];
        out.print ('(');
        sol(p,k);
        out.print (',');
        sol(k+1,u);
        out.print(')');
    }
}
```

Înmulţirea optimă a unui şir de matrice

- **Exemplu**

$A_1(100,1)$, $A_2(1,100)$, $A_3(100,1)$, $A_4(1,100)$

cost[i][j]= min{cost[i][k]+cost[k+1][j]+$d_i \cdot d_{k+1} \cdot d_{j+1}$ | i≤k<j}.

cost:

0	10000	200	10200
1	0	100	200
1	2	0	10000
1	3	3	0

$A_1A_2A_3A_4 = A_1(A_2A_3A_4)$ deoarece k=cost[4][1]=1

Înmulţirea optimă a unui şir de matrice

Exemplu

$A_1(100,1)$, $A_2(1,100)$, $A_3(100,1)$, $A_4(1,100)$

cost[i][j]= min{cost[i][k]+cost[k+1][j]+$d_i \cdot d_{k+1} \cdot d_{j+1}$ | i≤k<j}.

cost:

0	10000	200	10200
1	0	100	200
1	2	0	10000
1	3	3	0

$A_1A_2A_3A_4 = A_1(A_2A_3A_4)$ deoarece k=cost[4][1]=1

A_1

$A_2A_3A_4 = (A_2A_3)A_4$ deoarece k=cost[4][2]=3

Înmulţirea optimă a unui şir de matrice

Exemplu

$A_1(100,1)$, $A_2(1,100)$, $A_3(100,1)$, $A_4(1,100)$

cost[i][j]= min{cost[i][k]+cost[k+1][j]+$d_i \cdot d_{k+1} \cdot d_{j+1}$ | i≤k<j}.

cost:

0	10000	200	10200
1	0	100	200
1	2	0	10000
1	3	3	0

$A_1A_2A_3A_4 = A_1(A_2A_3A_4)$ deoarece k=cost[4][1]=1

A_1

$A_2A_3A_4 = (A_2A_3)A_4$ deoarece k=cost[4][2]=3

$A_2A_3 = (A_2)(A_3)$ deoarece k=cost[3][2]=2

A_4

Parantezare optima: $A_1A_2A_3A_4 = A_1((A_2A_3)A_4)$

Înmulţirea optimă a unui şir de matrice

- **Număr de comparații – $O(n^3)$**

$$\sum_{j=2}^{n}\sum_{i=1}^{j-1}(j-i+1)=\sum_{j=2}^{n}\left[j(j-1)-\frac{(j-1)(j-2)}{2}\right]$$

Înmulţirea optimă a unui şir de matrice

- **Numărul de parantezări optime ale produsului**

$$A_1 \cdot A_2 \cdot \ldots \cdot A_n$$

- **Numărul de parantezări ale produsului**

$$A_1 \cdot A_2 \cdot \ldots \cdot A_n$$

Înmulţirea optimă a unui şir de matrice

- Numărul de parantezări ale produsului

$$A_1 \cdot A_2 \cdot \ldots \cdot A_n$$

$$(A_1 \ldots A_k)\ (A_{k+1} \ldots A_n)$$

- **Subprobleme**

  P[i] = numărul de parantezări pentru o secvenţă de lungime i

- **Relaţii de recurenţă**

$$P[i] = \sum_{k=1}^{i-1} P[k]P[i-k]$$

Înmulţirea optimă a unui şir de matrice

- **Numărul de parantezări ale produsului**

$$A_1 \cdot A_2 \cdot \ldots \cdot A_n$$

$$P[n] = C_{n-1}$$

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$ –numărul lui Catalan

$$C_{n+1} = \frac{2(2n+1)}{n+2} C_n$$

https://en.wikipedia.org/wiki/Catalan_number