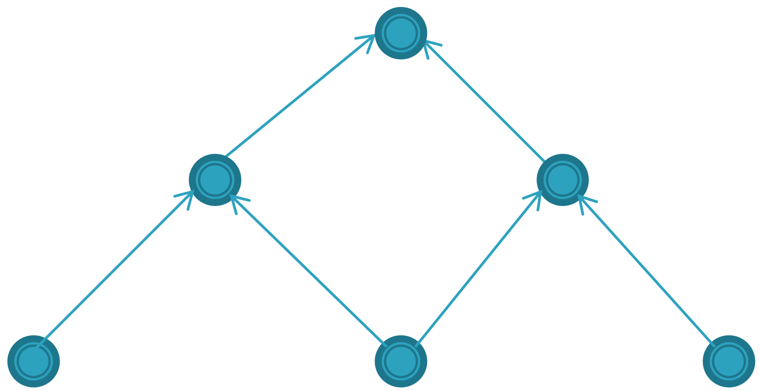
Metoda
Programării Dinamice

Dezavantaje ale metodelor deja studiate

- Greedy – nu furnizează mereu soluția optimă

- Divide et Impera – ineficientă dacă subproblemele se repetă



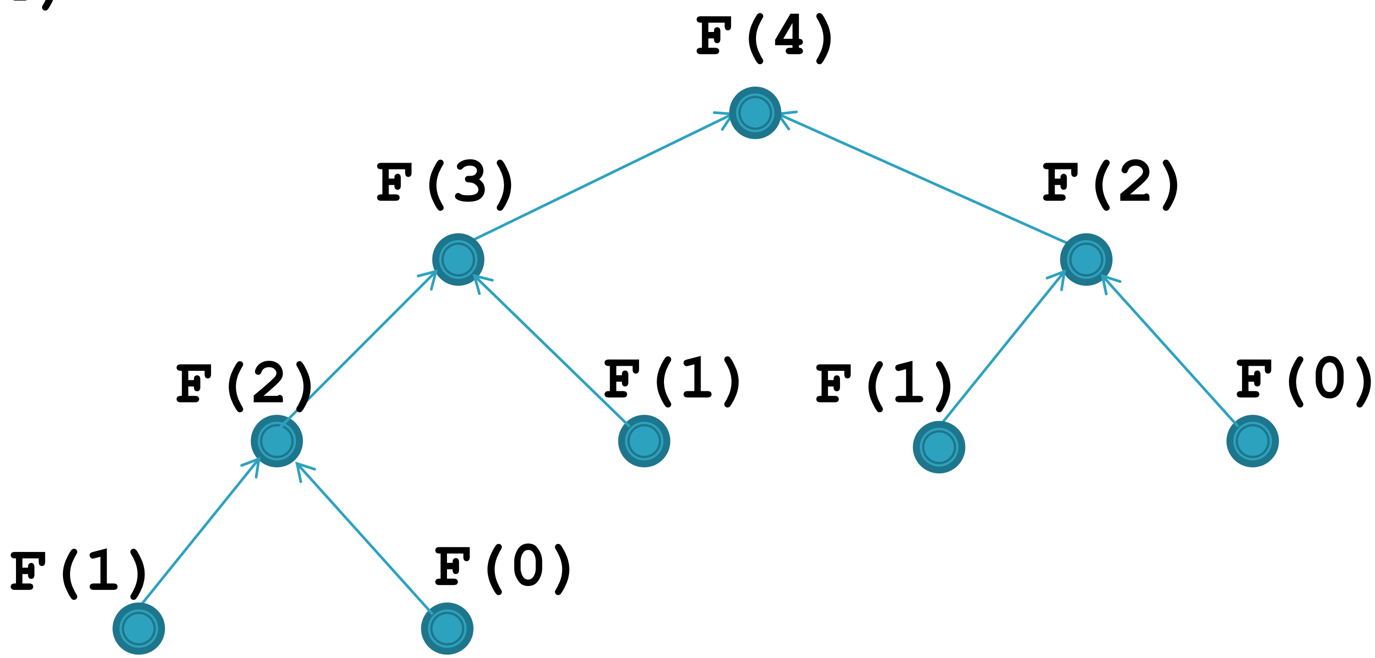
Dezavantaje ale metodelor deja studiate

- **Exemplu –** Calculăm numărul Fibonacci `F(n)`

  `F(n) = F(n-1)+F(n-2)`

  `F(0) = F(1) = 1`

- **`F(4)`**



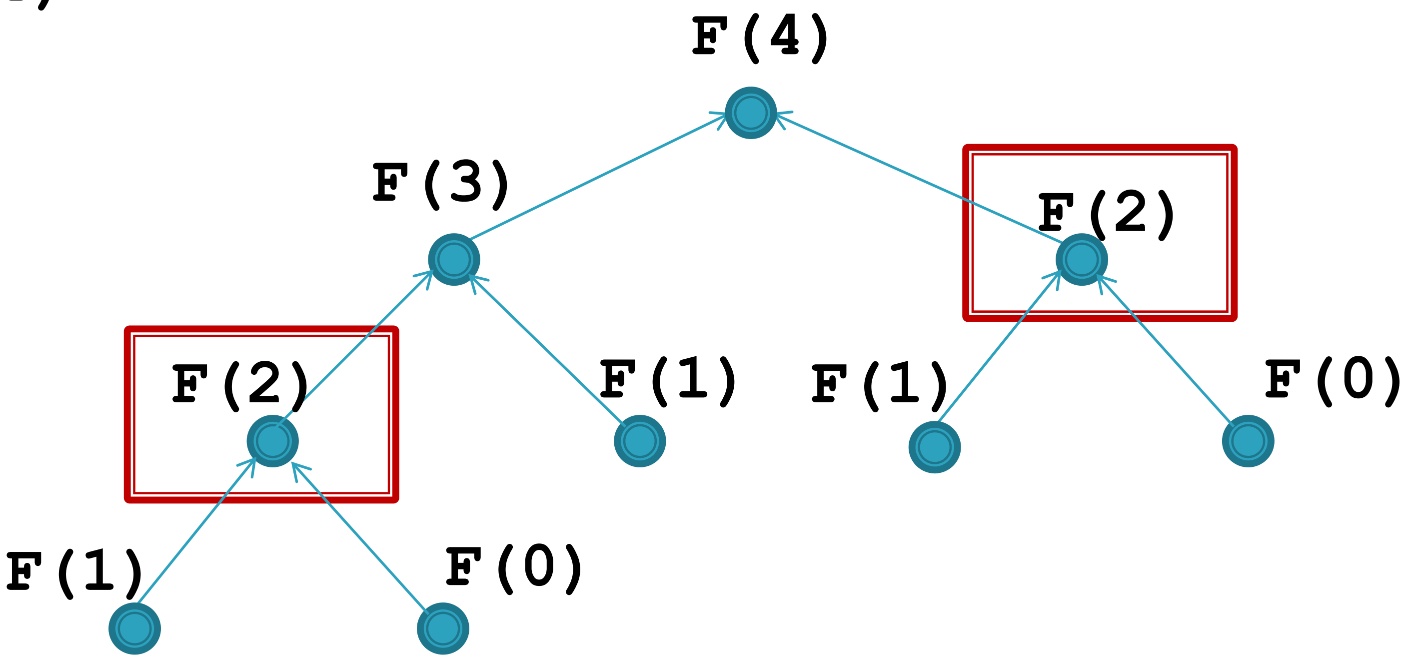
Dezavantaje ale metodelor deja studiate

- **Exemplu** – Calculăm numărul Fibonacci `F(n)`

  `F(n) = F(n-1)+F(n-2)`

  `F(0) = F(1) = 1`

- **`F(4)`**



Dezavantaje ale metodelor deja studiate

- **Exemplu –** Calculăm numărul Fibonacci `F(n)`

```
F(n) = F(n-1)+F(n-2)
F(0) = F(1) = 1
```

- Subproblemele se repetă => memorez termenii deja calculați (pe cei necesari)

```
F[0] = 1; F[1] = 1
for i in range(2,n+1):
    F[i] = F[i-1] + F[i-2]
```

Dezavantaje ale metodelor deja studiate

- **Exemplu –** Calculăm numărul Fibonacci `F(n)`

```
F(n) = F(n-1)+F(n-2)
F(0) = F(1) = 1
```

- Subproblemele se repetă => memorez termenii deja calculați (pe cei necesari)

```
F[0] = 1; F[1] = 1
for i in range(2,n+1):
    F[i] = F[i-1] + F[i-2]
```

Observație – suficient să memorăm doar doi termeni

```
F0, F1 = 1, 1
for i in range(2,n+1):
    F0, F1 = F1, F0 + F1
```

Metoda Programării Dinamice

Metoda programării dinamice

- **Metoda programării dinamice constă în**
  - **Reducerea problemei la subprobleme utile + determinarea de relații de recurență**
  - **rezolvarea eficientă a subproblemelor (recurențelor), cu memoizare = memorarea soluțiilor subproblemelor deja rezolvate** (pentru a nu le recalcula)

Metoda programării dinamice

Cum putem obține relații de recurență?

Metoda programării dinamice

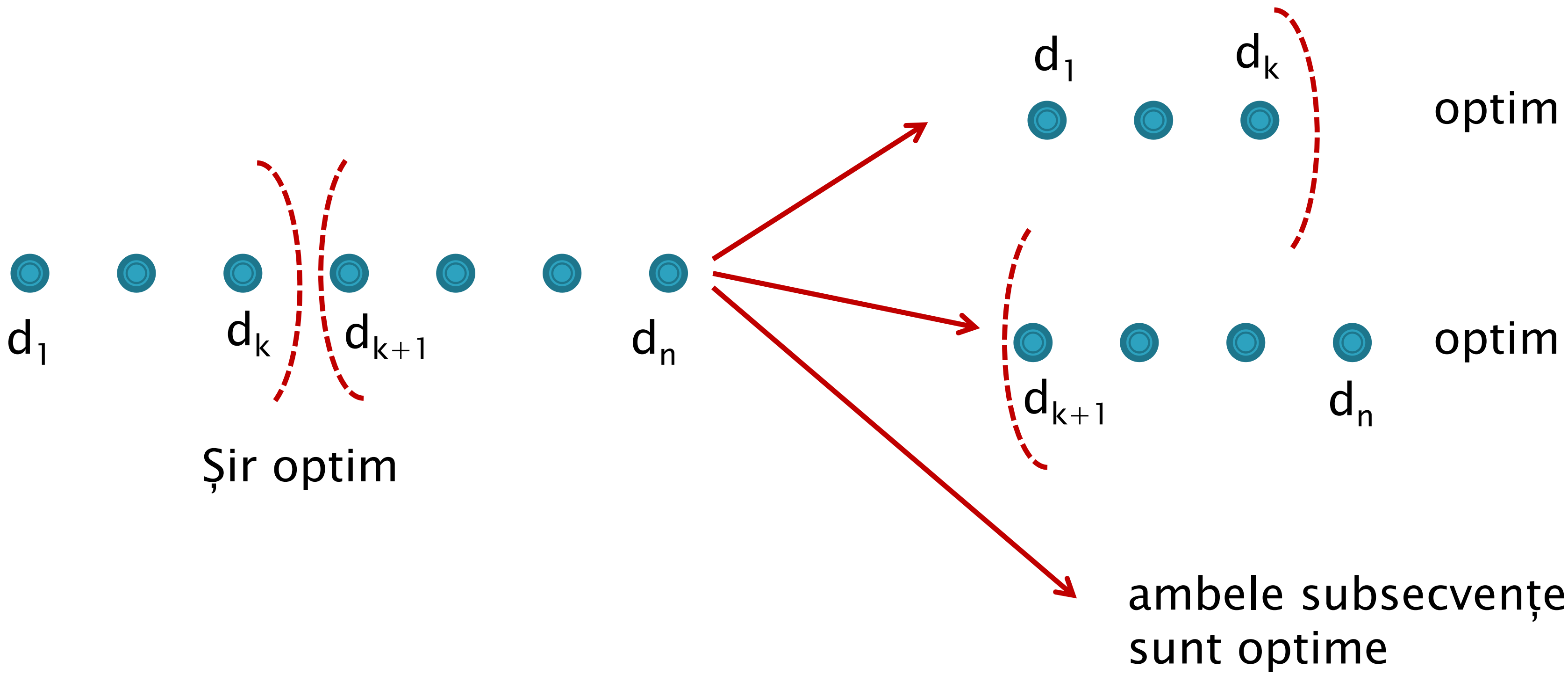
Cum putem obține relații de recurență?

- Exemplu: În **problemele de optim** – care verifică un **principiu de optimalitate**, din care se obţin relaţiile de calcul

Metoda programării dinamice

Fie soluția optimă $d_1, ..., d_n$

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:

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

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

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

si

$$d_{k+1}, ..., d_n \text{ optim, } \forall\ 1 \le k \le n$$

Etape

- **Stabilirea subproblemelor utile** (de exemplu din principiul de optimalitate)

Etape

- **Stabilirea subproblemelor utile** (de exemplu din principiul de optimalitate)
- **Cum putem rezolva problema iniţială folosind subproblemele**

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**

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 rezolvare a recurenţelor**

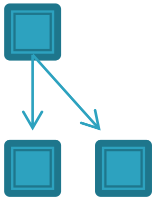
Exemple

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
```

- Se pot construi în total 2^{n-1} astfel de trasee

Trasee de sumă maximă

- **Exemplu**

  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,

atunci

$$(i_2,\ j_2),\ \dots\ ,(i_n,\ j_n)$$

$$(i_k,\ j_k),\ \dots\ ,(i_n,\ j_n) \quad \Longrightarrow$$

Trasee de sumă maximă

- **Principiu de optimalitate:**

Dacă

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

este un traseu optim care **începe** din celula (i_1, j_1), atunci

$$(i_2, j_2), \dots ,(i_n, j_n)$$

este un traseu optim **dacă pornim din celula (i_2, j_2)**

Trasee de sumă maximă

- **Principiu de optimalitate:**

  Dacă

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

  este un traseu optim care **începe** din celula (i_1, j_1), atunci

  $(i_2, j_2), \ldots ,(i_n, j_n)$

  este un traseu optim **dacă pornim din celula (i_2, j_2)**

- **Subproblemă:**

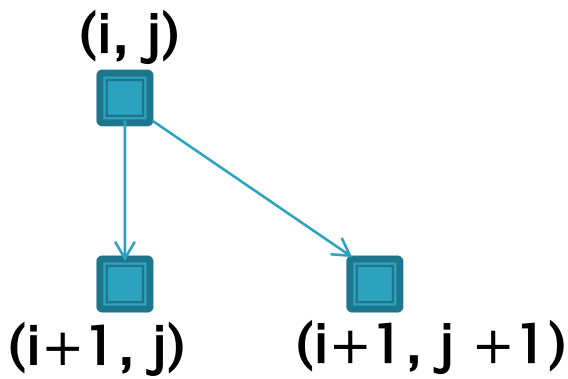
  Calculăm pentru o poziție (i, j) suma maximă pe care o putem obține **dacă pornim din celula (i, j)**

Trasee de sumă maximă

- **Subproblemă:**

  Calculăm pentru o poziție (i, j) suma maximă pe care o putem obține **dacă pornim din celula (i, j)**

  suma(i,j) = t[i][j] + max(suma(i+1, j), suma(i+1, j+1))



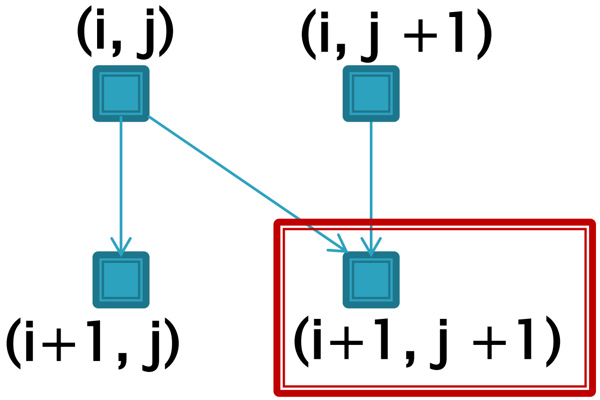
Trasee de sumă maximă

- **Subproblemă:**

Calculăm pentru o poziție (i, j) suma maximă pe care o putem obține **dacă pornim din celula (i, j)**

$$suma(i,j) = t[i][j] + \max(suma(i+1, j), suma(i+1, j+1))$$

Subproblemele se suprapun, o soluție recursivă fără memiozare (tip "Divide et Impera") este ineficientă



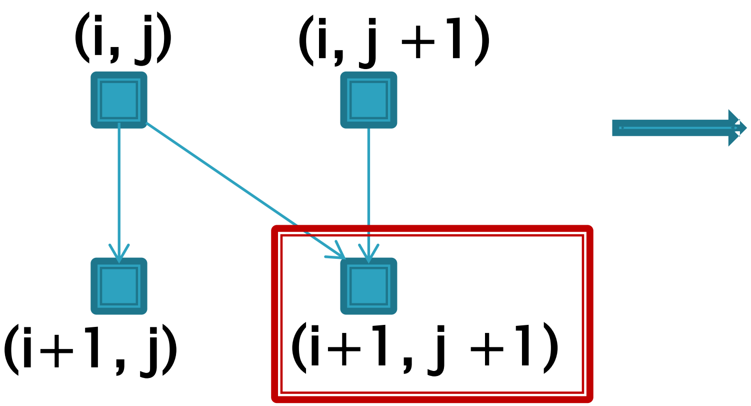
Trasee de sumă maximă

- **Subproblemă:**

  Calculăm pentru o poziție (i, j) suma maximă pe care o putem obține **dacă pornim din celula (i, j)**

  suma(i,j) = t[i][j] + max(suma(i+1, j), suma(i+1, j+1))

  Subproblemele se suprapun, o soluție recursivă fără memiozare (tip "Divide et Impera") este ineficientă



Memorăm rezultatele subproblemelor într-o matrice

Trasee de sumă maximă

- **Subproblemă:**

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

Trasee de sumă maximă

- **Subproblemă:**

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

- **Soluție problemă**

Trasee de sumă maximă

- **Subproblemă:**

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

- **Soluție problemă** `s[1][1]`

Trasee de sumă maximă

- **Subproblemă:**

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

- **Soluție problemă** `s[1][1]`
- **Știm direct**

Trasee de sumă maximă

- **Subproblemă:**

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

- **Soluție problemă** `s[1][1]`

- **Știm direct** `s[n][j] = t[n][j], j=1,2,…,n`

Trasee de sumă maximă

- **Subproblemă:**

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

- **Soluție problemă** `s[1][1]`
- **Știm direct** `s[n][j] = t[n][j], j=1,2,…,n`
- **Relație de recurență**

`s[i][j] = t[i][j]+ max{s[i+1][j],s[i+1][j+1]}`

Trasee de sumă maximă

- **Subproblemă:**

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

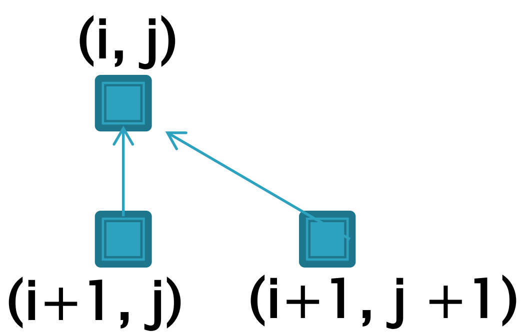
- **Soluție problemă** `s[1][1]`

- **Ştim direct** `s[n][j] = t[n][j], j=1,2,…,n`

- **Relație de recurență**

```
s[i][j] = t[i][j]+ max{s[i+1][j],s[i+1][j+1]}
```

- **Ordinea de rezolvare a recurenţelor**



Trasee de sumă maximă

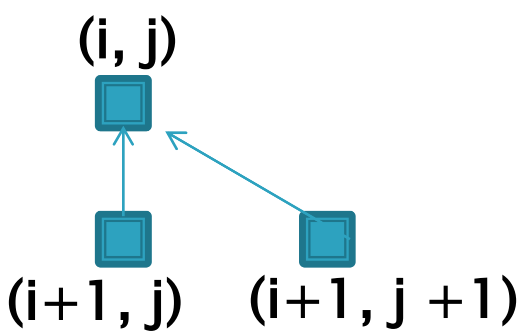
- **Subproblemă:**

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

- **Soluție problemă** `s[1][1]`
- **Știm direct** `s[n][j] = t[n][j], j=1,2,…,n`
- **Relație de recurență**

`s[i][j] = t[i][j]+ max{s[i+1][j],s[i+1][j+1]}`

- **Ordinea de rezolvare a recurenţelor**



de la ultima linie către prima

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              5   9
3  4   7  2           3   4   7   2
      t                   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
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

- **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

- **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

Implementare

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

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

este un traseu optim atunci

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

este un traseu optim ...

Trasee de sumă maximă

- **Principiu de optimalitate – Altă variantă**

  Dacă

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

  este un traseu optim atunci

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

  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 calcul**

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

<u>Exemplu</u>

Pentru

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

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

$$1, \quad 4, 6, \quad 11$$

Subşir crescător de lungime maximă

Principiu de optimalitate:

Dacă

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

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

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

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

Mai general

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

este un subşir 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:**

  lmax = max{lung[i] | i = 1,2,…,n}

Subşir crescător de lungime maximă

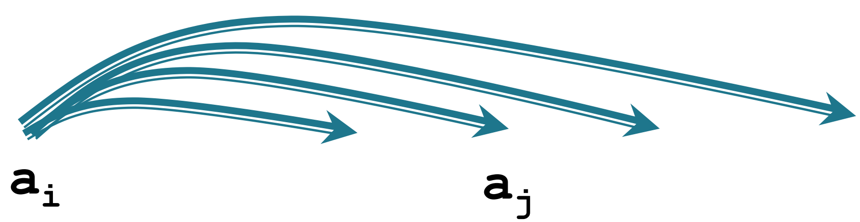
- **Subproblemă:**

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

- **Știm direct** `lung[n] = 1`

- **Relație de recurență**

  $$\texttt{lung[i]} = 1 + \max\{\texttt{lung[j]} \mid j>i,\ a_i<a_j\}$$



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**

  $\texttt{lung[i]} = 1 + \max\{\texttt{lung[j]} \mid j>i,\ a_i<a_j\}$

- **Ordinea de calcul**

  `i = n, n-1, … , 1`

Subşir crescător de lungime maximă

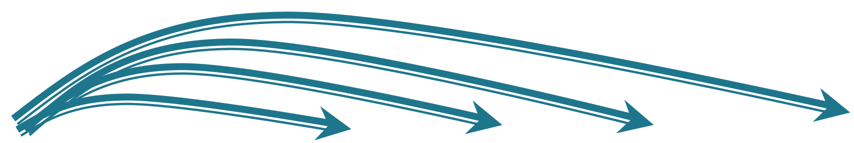
Cum determinăm un subşir maxim?

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**



$\mathbf{a_i}$ $\mathbf{a_j}$

`succ[i]=j`

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

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

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,

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

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

Altă soluție

Principiu de optimalitate:

Dacă

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

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

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

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