Metoda
Programării Dinamice

Prezentarea metodei

Cadru general

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

Cadru general

- Fie A şi B două mulţimi oarecare (B = **N**, **Z**, **R**, {0,1} …)

  Fiecărui $x \in A$ **urmează să i se asocieze** o valoare $v(x) \in B$.

$$\mathbf{x \in A \longrightarrow v(x) \in B}$$

Cadru general

- Fie A şi B două mulţimi oarecare (B = **N**, **Z**, **R**, {0,1} …)

  Fiecărui $x \in A$ **urmează să i se asocieze** o valoare $v(x) \in B$.

$$\mathbf{x \in A \longrightarrow v(x) \in B}$$

  - v este **cunoscută** doar pe submulţimea $\mathbf{X \subset A}$

Cadru general

- Fie A şi B două mulţimi oarecare (B = **N**, **Z**, **R**, {0,1} ...)

  Fiecărui $x \in A$ **urmează să i se asocieze** o valoare $v(x) \in B$.

$$x \in A \longrightarrow v(x) \in B$$

  - v este **cunoscută** doar pe submulţimea $X \subset A$

  - Pentru fiecare $x \in A \setminus X$ avem relația

$$v(x) = f_x(v(a_1),\ldots,v(a_k))$$

    unde

$$A_x = \{a_1,\ldots,a_k\}$$

    este mulţimea elementelor din A **de a căror valoare depinde (direct) v(x)**

Cadru general

- Dat $z \in A$, se cere să se calculeze, **dacă este posibil**, valoarea $v(z)$ – **eficient**

Cadru general

- Putem reprezenta problema pe un ***graf de dependenţe***. Vârfurile corespund elementelor din A, iar descendenţii unui vârf x sunt vârfurile din A_x.

$$a_i \longrightarrow x$$

valoarea lui x depinde

direct de $v(a_i)$

$$(v(x) = f_x(v(a_1),...,v(a_k)))$$

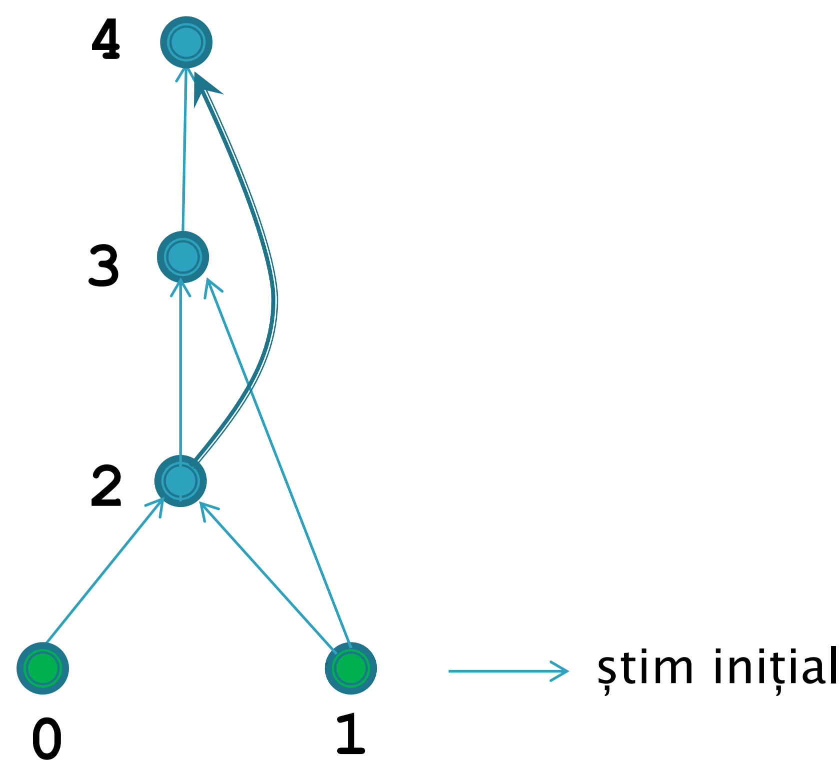
- Problema are soluție numai dacă în graful de dependențe **nu există circuite accesibile din z**

Graf de dependențe

- Calcul număr Fibonacci F_n

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

$\Longrightarrow$ **X ={0, 1}**



Graf de dependenţe

- **Alt exemplu**

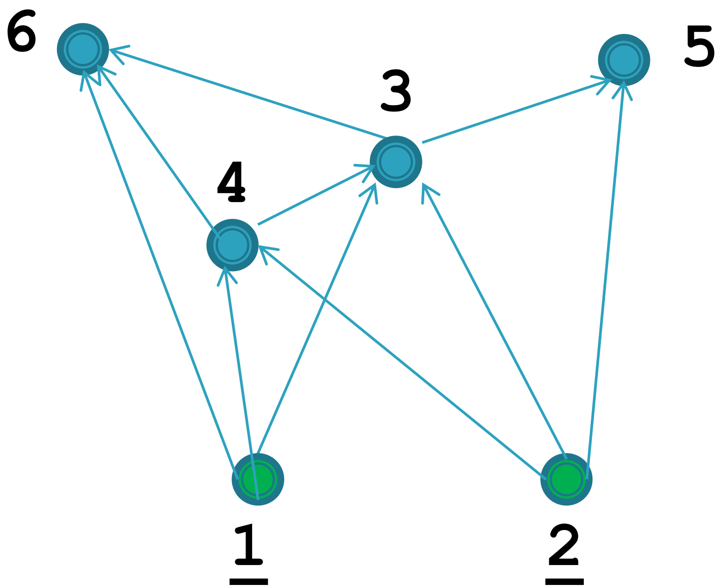
```
A = {1,2,...,5,6};
v(1) = v(2) = 1
v(3) = v(1) + v(2) + v(4)
v(4) = v(1) + v(2)
v(5) = v(2) + v(3)
v(6) = v(1) + v(3) + v(4)
```

v(6) = ?

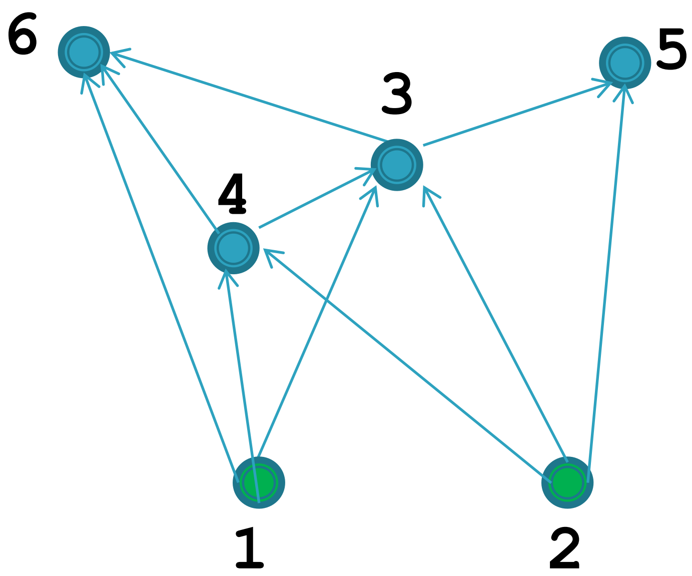
Graf de dependenţe

- **Alt exemplu**

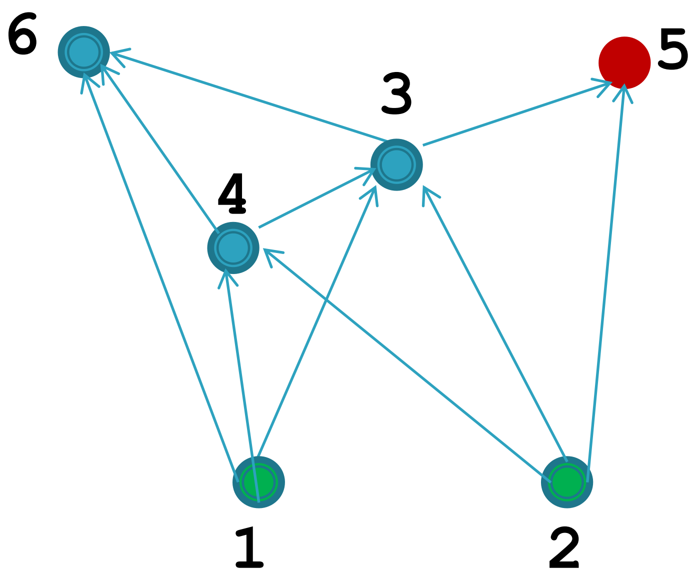
```
A = {1,2,...,5,6};
v(1) = v(2) = 1      ⟹  X = {1, 2}
v(3) = v(1) + v(2) + v(4)
v(4) = v(1) + v(2)
v(5) = v(2) + v(3)
v(6) = v(1) + v(3) + v(4)
```



$v(6) = ?$



v(6) = ?



nu intervine în calculul lui v(6)

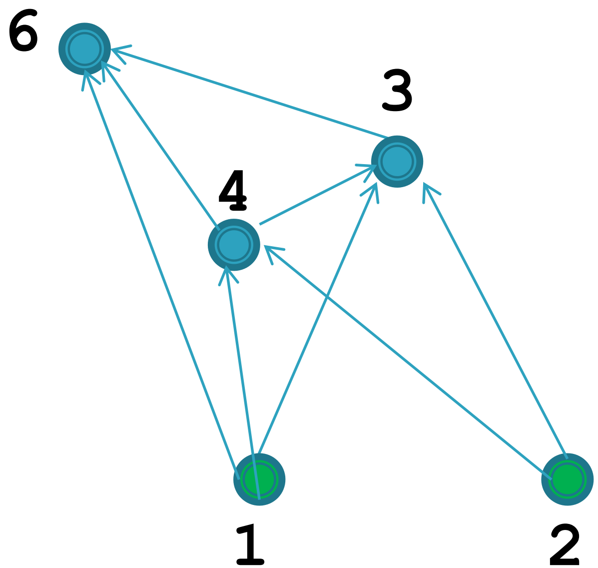
Cadru general

- Fie G_z graful indus de mulțimea vârfurilor **y de a căror valoare depinde v(z)** = pentru care există un drum de la y la z = **vârfuri observabile** din z

  G_z = **graful vârfurilor observabile** din z

- Problema presupune o **parcurgere a grafului G_z**

v(6) = ?



graful vârfurilor observabile din 6

Cadru general

- Ar fi bine dacă
  - am cunoaşte de la început G_z
  - forma acestui graf ar permite o **parcurgere mai simplă**, care să conducă la calcularea valorii $v(z)$

Cadru general

- Încercare de rezolvare cu metoda Divide et Impera

```
procedure DivImp(x)
    for y∈Ax\X  {y de care x depinde direct}
       DivImp(y)
    calculează v(x) conform funcţiei fx
end;
```

  - algoritmul nu se termină pentru grafuri ciclice
  - valoarea unui vârf poate fi calculată de mai multe ori

Cadru general

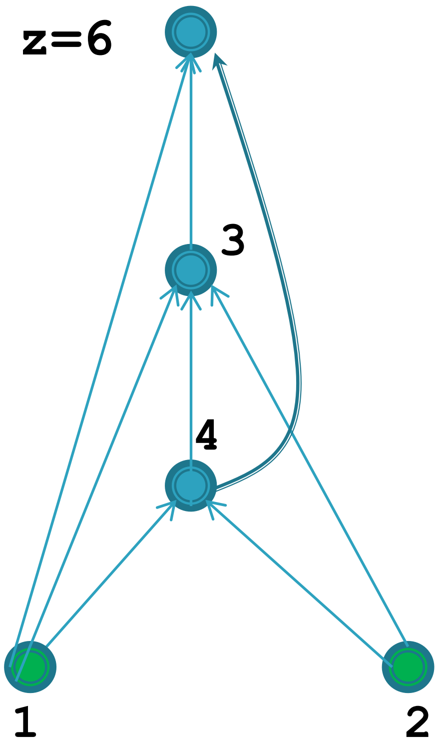
- **Soluție** – sortarea topologică pentru G_z ⟶ ordinea în care se calculează valorile v ale vârfurilor
  - **Ar fi mai bine dacă forma grafului ar permite o parcurgere mai simplă.**

Metoda programării dinamice

- **Metoda programării dinamice constă în următoarele**:
  - Se asociază problemei un **graf de dependenţe, corespunzător relațiilor de recurență**

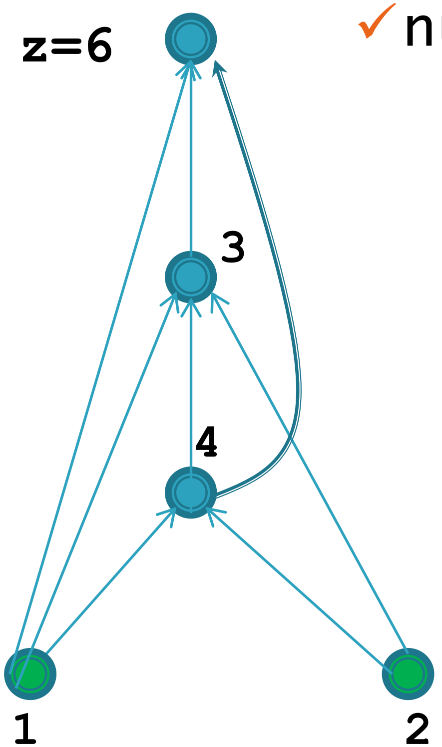
Metoda programării dinamice

- **Metoda programării dinamice constă în următoarele**:
  - Se asociază problemei un **graf de dependenţe**
  - În graf este pus în evidenţă un graf de vârfuri **observabile** din z, numit **PD-arbore de rădăcină z**, cu proprietățile



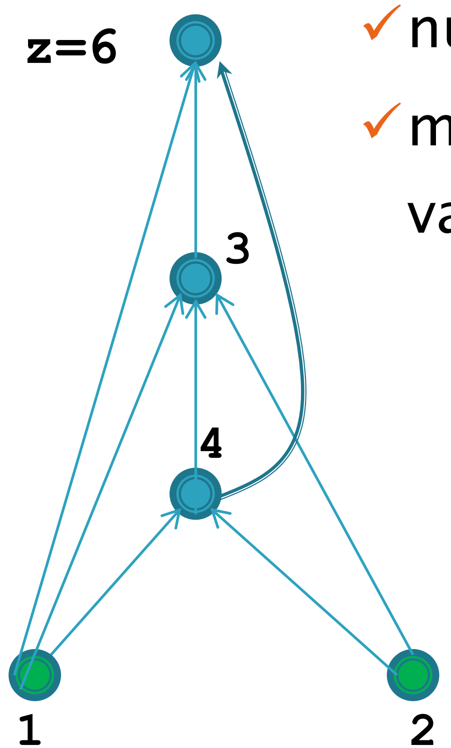
Metoda programării dinamice

- **Metoda programării dinamice constă în următoarele**:
  - Se asociază problemei un **graf de dependenţe**
  - În graf este pus în evidenţă un graf de vârfuri **observabile** din z, numit **PD-arbore de rădăcină z**, cu proprietățile
    - ✓ nu conţine circuite



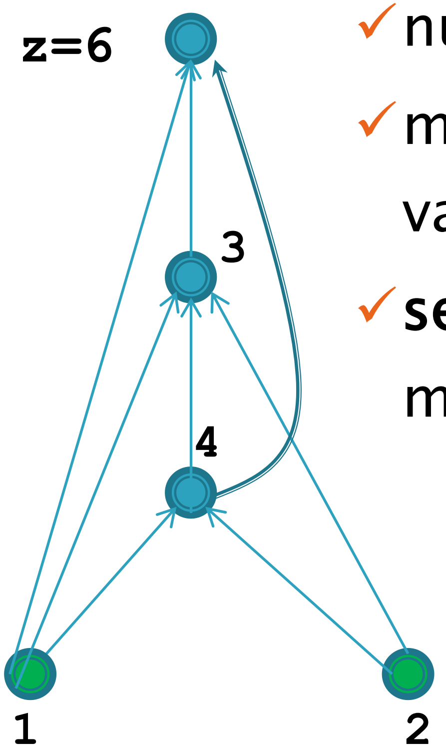
Metoda programării dinamice

- **Metoda programării dinamice constă în următoarele**:
  - Se asociază problemei un **graf de dependenţe**
  - În graf este pus în evidenţă un graf de vârfuri **observabile** din z, numit **PD-arbore de rădăcină z**, cu proprietățile
    - ✓ nu conţine circuite
    - ✓ mulțimea vârfurilor cu gradul intern 0 (ale căror valori nu depind de o altă valoare) este inclusă în X



Metoda programării dinamice

- **Metoda programării dinamice constă în următoarele**:
  - Se asociază problemei un **graf de dependenţe**
  - În graf este pus în evidenţă un graf de vârfuri **observabile** din z, numit **PD-arbore de rădăcină z**, cu proprietățile



✓ nu conţine circuite

✓ mulțimea vârfurilor cu gradul intern 0 (ale căror valori nu depind de o altă valoare) este inclusă în X

✓ **se poate așeza pe niveluri** (nivelul lui y = lungimea maximă a unui drum de la y la z)

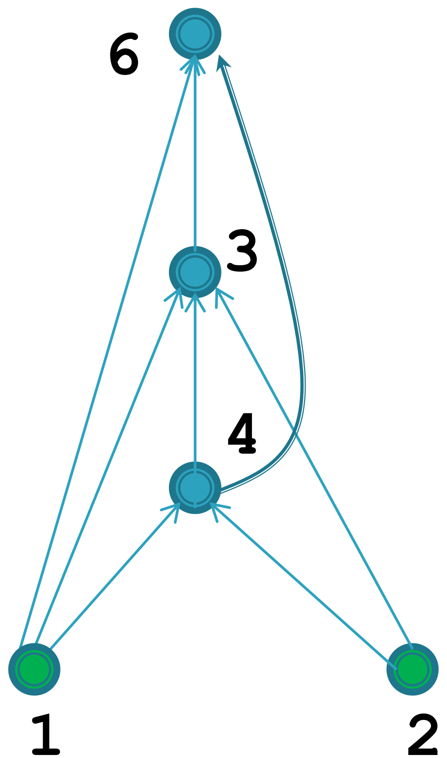
Metoda programării dinamice

- **Metoda programării dinamice constă în următoarele**:
  - Se asociază problemei un **graf de dependenţe**
  - În graf este pus în evidenţă un graf de vârfuri **observabile** din z, numit **PD-arbore de rădăcină z**
  - Se parcurge PD-arborele în **postordine generalizată** (fără a parcurge noduri deja vizitate = fără a rezolva din nou subprobleme deja rezolvate)

```
procedure postoprd(x)
    for j∈A_x
        if viz[j]=false {diferența față de DI}
            postord(j)
    calculează v(x) conform funcţiei f_x;
    viz[x]←true
end
```

- Apel postord(z)

v(6) = ?



- Prin parcurgerea în postordine generalizată, vârfurile vor fi **sortate topologic: 1, 2, 4, 3, 6**

Metoda programării dinamice

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

- Este util să căutăm în PD-arbore regularități care să evite memorarea valorilor tuturor vârfurilor şi/sau să simplifice parcurgerea în postordine (generalizată).

Exemplu – Fibonacci

- A = {0,...,n}, B = **N**
- **X = {0,1}** (ştim F_0=0; F_1=1)
- v(k) = F_k , deci
  - **v(k) = v(k-1)+ v(k-2)**

Exemplu – Fibonacci

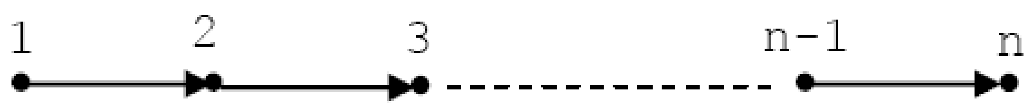
- $A = \{0,...,n\}$, $B = \mathbf{N}$
- $\mathbf{X = \{0,1\}}$ (ştim $F_0=0$; $F_1=1$)
- $v(k) = F_k$, deci
  - $\mathbf{v(k) = v(k-1)+ v(k-2)}$
  - $A_k = \{k-1,k-2\}$, $\forall k \geq 2$
  - $f_k(a,b) = a + b$, $\forall k \geq 2$

Exemplu – Fibonacci , varianta 2

- $A = \{1,...,n\}$, $B = \mathbf{N} \times \mathbf{N}$
- $\mathbf{v(k) = (F_{k-1}, F_k)}$
- $v(1) = (0, 1)$

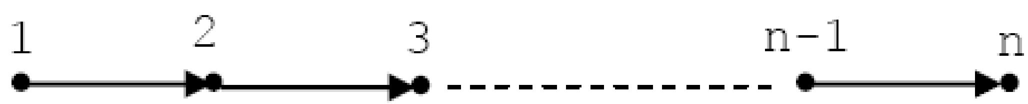
Exemplu – Fibonacci , varianta 2

- $A = \{1, ..., n\}, \quad B = \mathbf{N} \times \mathbf{N}$
- $\mathbf{v(k) = (F_{k-1}, F_k)}$
- $v(1) = (0, 1)$
  - $A_k = \{k-1\}, \ \forall k \geq 2$
  - $\mathbf{f_k(a,b) = (b, a+b)} \ \forall k \geq 2$
  - $\mathbf{v(k) = f_k(v(k-1))}$



Exemplu – Fibonacci , varianta 2

- $A = \{1, \ldots, n\}$, $B = \mathbf{N} \times \mathbf{N}$
- $\mathbf{v(k) = (F_{k-1}, F_k)}$
- $v(1) = (0, 1)$
  - $A_k = \{k-1\}$, $\forall k \geq 2$
  - $\mathbf{f_k(a,b) = (b, a+b)}$ $\forall k \geq 2$
  - $\mathbf{v(k) = f_k(v(k-1))}$




```
a←0; b←1
for i=2,n
    (a,b)←(b,a+b)
write(b)
```

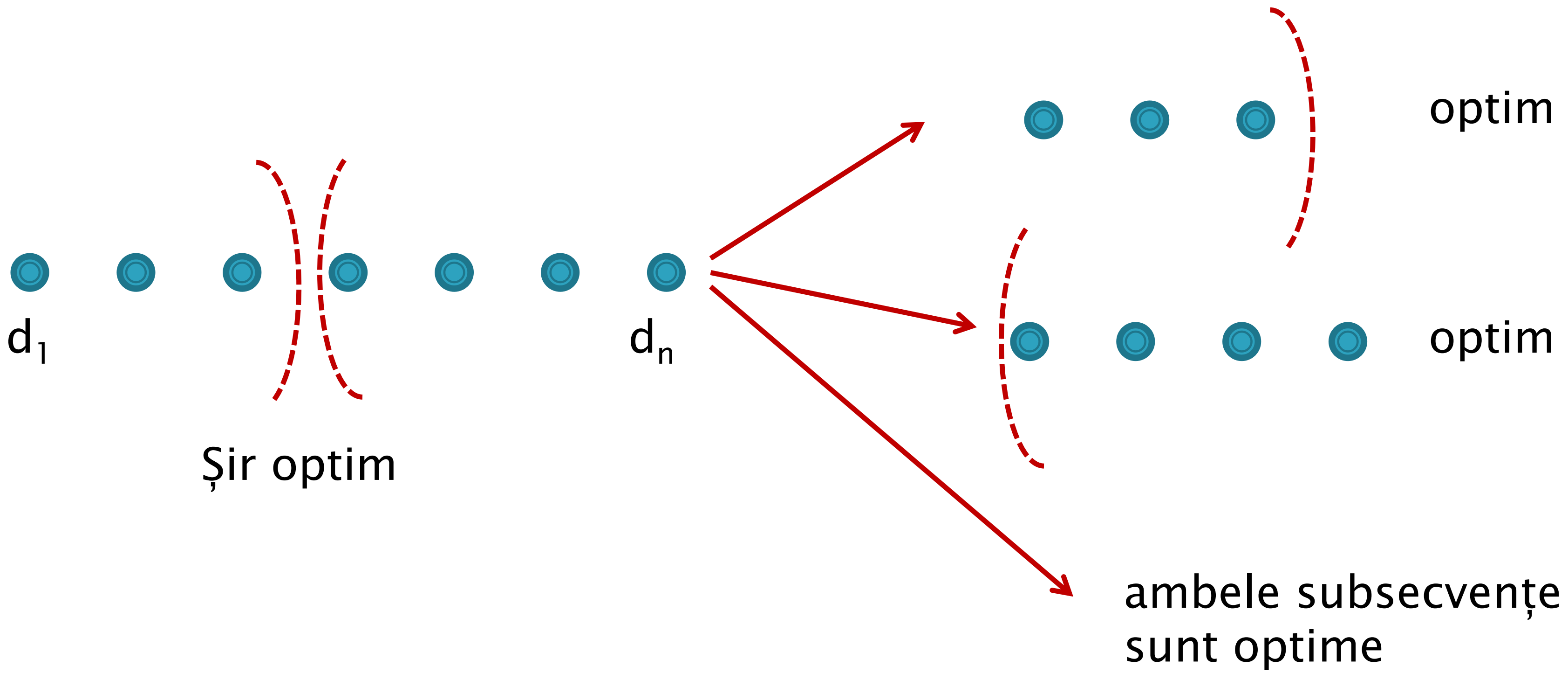
Metoda programării dinamice

- Se poate utiliza în **problemele de optim** – care verifică un **principiu de optimalitate**, din care se obţin relaţiile de calcul

Metoda programării dinamice

Fie soluția optimă $d_1, \ldots, d_n$

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



Metoda programării dinamice

Fie soluția optimă $d_1, \ldots, d_n$

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

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

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

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

si

$d_k, \ldots, d_n$ 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 (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

<u>Exemplu</u>

Pentru

a = (8, 3, 1, 4, 6, 5, 11)

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

3, 4, 6, 11

Subşir crescător de lungime maximă

- **Longest Increasing Subsequence**
- **Înrudită cu problema determinării celui mai lung subșir comun a două șiruri** (Longest Common Subsequence)
- **Aplicații**
  - cautarea de tiparuri (patterns):

    baze de date mari

    bioinformatica
  - similitudini în genetică (ADN)

    sequence alignment

- Lavanya, B., Murugan, A.: Discovery of longest increasing subsequences and its variants using DNA operations. International Journal of Engineering and Technology 5(2), 1169-1177 (2013)

Subşir crescător de lungime maximă

Principiu de optimalitate:

Dacă

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

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

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

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

Mai general

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

este un subşir optim care începe pe poziţia `ik`

(pentru $k \le p$)

Subşir crescător de lungime maximă

Principiu de optimalitate:

Dacă $a_{i1}, a_{i2}, \dots, a_{ip}$,

este un subşir optim care începe pe poziţia i1, atunci:

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

este un subşir optim care începe pe poziţia i2;

Observație Este incorectă concluzia că $a_{i2}, \dots, a_{ip}$ este un subșir optim pentru subvectorul $a[i_1+1\dots n]$ (format cu elementele de după a_{i1}) sau $a[i_2\dots n]$, este obligatoriu să impunem subșirului și condiția să înceapă cu elementul a_{i2} (mai exact cu elementul de pe poziția i2)

Exemplu Pentru a = (8, 3, 4, 1, 2, 6, 5, 11)

subșirul 3 4 6 11 este subșir optim, dar

subșirul 4 6 11 nu este subșir optim pentru subvectorul (4, 1, 2, 6, 5, 11) (subșirul 1 2 6 11 este mai lung), ci este optim cu proprietatea că începe cu elementul 4 (de pe poziția 3)

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

- Subprobleme

- Soluție problemă inițială

- Ce subprobleme știm să rezolvăm direct

- Relații de recurență

- Ordinea de rezolvare a recurenţelor (parcurgerea PD-arborelui)

Subşir crescător de lungime maximă

- **Subprobleme:**

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

Subşir crescător de lungime maximă

- **Subprobleme:**

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

- **Soluție problemă:**

Subşir crescător de lungime maximă

- **Subprobleme:**

  `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

- **Subprobleme:**

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

- **Știm direct**

- **Relație de recurență**

- **Ordinea de rezolvare a recurenţelor (parcurgerea PD–arborelui)**

Subşir crescător de lungime maximă

- Subprobleme:

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

- Știm direct `lung[n] = 1`

- Relație de recurență

- Ordinea de rezolvare a recurenţelor (parcurgerea PD-arborelui)

Subşir crescător de lungime maximă

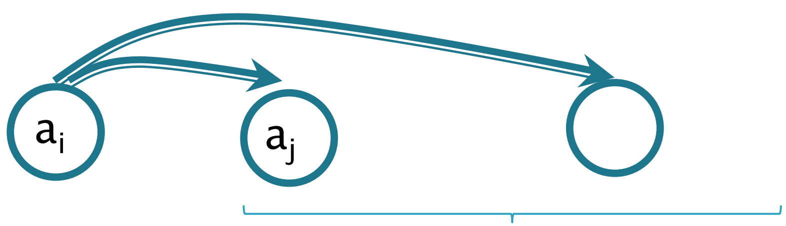
- **Subprobleme:**

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

- **Ştim direct** `lung[n] = 1`

- **Relație de recurență**

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



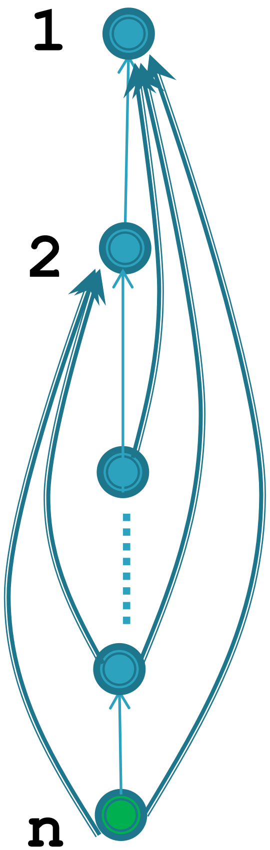
au lung[j] **deja calculat**

- următorul element după a_i în subșirul maxim care începe pe poziția i va fi acel a_j cu $i<j$, $a_i<a_j$ având lung[j] maxim

(aleg un **subşir de lungime maximă care începe pe o poziție j> i cu $a_j>a_i$**)

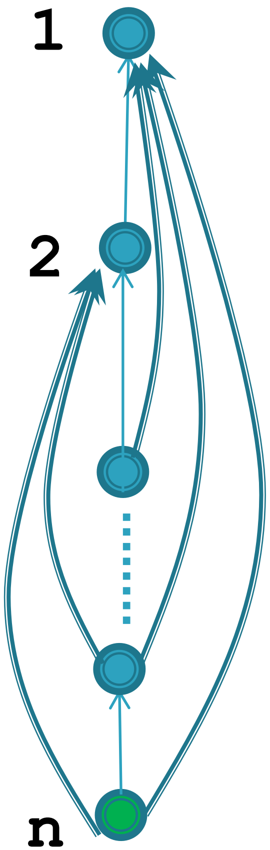
Subşir crescător de lungime maximă

- Graful de dependenţe



Subşir crescător de lungime maximă

- Graful de dependenţe



- Ordinea de rezolvare a recurenţelor (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	3	1	4	6	5	11
	1	2	3	4	5	6	7
lung :							
succ :							

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

Subşir crescător de lungime maximă

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

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

Subşir crescător de lungime maximă

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

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

Subşir crescător de lungime maximă

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

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

Subşir crescător de lungime maximă

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

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

Subşir crescător de lungime maximă

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

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

Subşir crescător de lungime maximă

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

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

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

Subşir crescător de lungime maximă

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

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

Subşir crescător de lungime maximă

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

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

Subşir crescător de lungime maximă

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

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

Subşir crescător de lungime maximă

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

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

Subşir crescător de lungime maximă

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

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

Subşir crescător de lungime maximă

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

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

Subşir crescător de lungime maximă

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

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

Subşir crescător de lungime maximă

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

Soluţie: lung = 4

Subşir crescător de lungime maximă

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

Subşir: 3,

Subşir crescător de lungime maximă

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

Subşir: 3,

Subşir crescător de lungime maximă

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

Subşir: 3, 4,

Subşir crescător de lungime maximă

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

Subşir: 3, 4, 6

Subşir crescător de lungime maximă

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

Subşir: 3, 4, 6, 11

Subşir crescător de lungime maximă

Algoritm

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;









}
```

```
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;
      }
    }

}
```

```
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++){
    cout<<a[poz]<<" ";
    poz = succ[poz];
}
```

```java
//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

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 acei indici j pentru cu proprietatea că a_j este **succesor** al lui a_i **într-un subșir de lungime maximă** care începe cu a_i:
  - **j>i, a_i<a_j** pentru care **`lung[i] = lung[j] + 1`**

(adică acei j pentru care se atinge max în relaţia de recurenţă

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

Subşir crescător de lungime maximă

- Subprobleme

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

- Soluție problema inițială

- Ştim să rezolvăm direct

- Relații de recurență

- Ordinea de rezolvare a recurenţelor **`i=n,…,1`**

Subşir crescător de lungime maximă

- **Subprobleme**

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

- **Soluție problema inițială** $$\sum_{\substack{poz=1,\ldots,n \\ lung[poz]=\text{lmax}}} nr[poz]$$

- **Ştim să rezolvăm direct**

- **Relații de recurență**

- **Ordinea de rezolvare a recurenţelor** **`i=n,…,1`**

Subşir crescător de lungime maximă

- **Subprobleme**

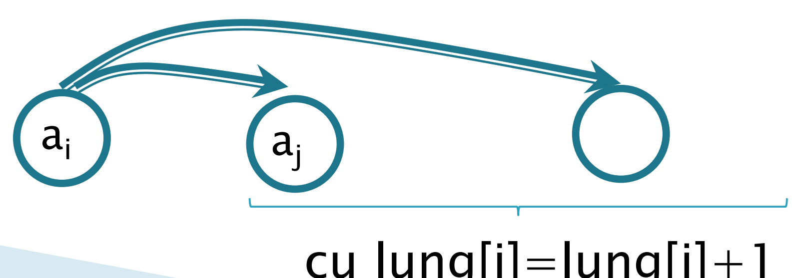
  `nr[i]` = numărul de subşiruri crescătoare de lungime maximă care încep pe poziţia i

- **Soluție problema inițială** $$\sum_{\substack{poz=1,\dots,n \\ lung[poz]=\text{lmax}}} nr[poz]$$

- **Ştim să rezolvăm direct** `nr[n]` $= 1$

- **Relații de recurență**

$$nr[i] = \begin{cases} \displaystyle\sum_{\substack{i<j,\, a_j>a_i \\ lung[i]=lung[j]+1}} nr[j], & \text{dacă există j>i cu } a_j > a_i \\ 1, & \text{altfel} \end{cases}$$



Subşir crescător de lungime maximă

- **Subprobleme**

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

- **Soluție problema inițială** $\displaystyle\sum_{\substack{poz=1,\dots,n \\ lung[poz]=\mathrm{lmax}}} nr[poz]$

- **Ştim să rezolvăm direct** `nr[n]` $= 1$

- **Relații de recurență**

$$nr[i] = \begin{cases} \displaystyle\sum_{\substack{i<j,\, a_j>a_i \\ lung[i]=lung[j]+1}} nr[j], & \text{dacă există j>i cu } a_j > a_i \\ 1, & \text{altfel} \end{cases}$$

- **Ordinea de rezolvare a recurenţelor** `i=n,…,1`

Subşir crescător de lungime maximă

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

nr:							1

Subşir crescător de lungime maximă

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

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

Subşir crescător de lungime maximă

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

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

Subşir crescător de lungime maximă

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

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

Subşir crescător de lungime maximă

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

```
nr[3] = nr[4]
```

Subşir crescător de lungime maximă

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

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

Subşir crescător de lungime maximă

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

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

Subşir crescător de lungime maximă

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

Subşir crescător de lungime maximă

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

Soluție: nr[2]+nr[3]=4

Altă soluție

Principiu de optimalitate:

Dacă

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

este un subșir optim care se termină pe poziția `ip` și $k<p$ arbitrar, atunci

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

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

<u>Altă soluție</u>

Principiu de optimalitate:

Dacă

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

este un subșir optim care se termină pe poziția `ip` și k<p arbitrar, 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 <u>**se termină**</u> **pe poziția i**

Subşir crescător de lungime maximă

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

Subşir crescător de lungime maximă

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

Subşir crescător de lungime maximă

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

Subşir crescător de lungime maximă

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

Subşir crescător de lungime maximă

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

Subşir crescător de lungime maximă

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

Subşir crescător de lungime maximă

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

Subşir crescător de lungime maximă

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

Subşir crescător de lungime maximă

Algoritm O(n log n)– Indicații

1. pentru i=1,n

Pentru fiecare lungime j=1..,n reținem

m[j] = poziția ultimului element dintr-un **subșir de lungime j** (în subvectorul a[1..i])

Pentru a[i] – **căutăm cea mai mare lungime j** cu proprietatea

a[m[j]] ≤ a[i]

(ultimul element din subșirul de lungime j este ≤ a[i])

Subşir crescător de lungime maximă

Algoritm O(n log n)– Indicații

1. pentru i=1,n

Pentru fiecare lungime j=1..,n reținem

m[j] = poziția ultimului (?!) element dintr-un **subșir crescător de lungime j** (în subvectorul a[1..i])

Pentru a[i] – **căutăm cea mai mare lungime j** cu proprietatea

`a[m[j]] < a[i]`

(ultimul element din subșirul de lungime j este ≤ a[i])

Dacă sunt mai multe subșiruri crescătoare de lungime j, ce memorăm ca ultim termen în m[j]?

Subşir crescător de lungime maximă

Algoritm O(n log n)– Indicații

1. pentru i=1,n

Pentru fiecare lungime j=1..,n reținem

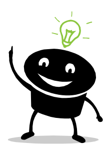
m[j] = poziția ultimului (?!) element dintr-un **subşir crescător de lungime j** (în subvectorul a[1..i])

Pentru a[i] – **căutăm cea mai mare lungime j** cu proprietatea

`a[m[j]] < a[i]`

(ultimul element din subșirul de lungime j este ≤ a[i])

Dacă sunt mai multe subșiruri crescătoare de lungime j, ce memorăm ca ultim termen în m[j]?



cel mai mic

Subşir crescător de lungime maximă

Algoritm O(n log n)– Indicații

1. pentru i=1,n

   Pentru fiecare lungime j=1..,n reținem

   m[j] = poziția celui mai mic element din șir (din subvectorul a[1..i]) cu proprietatea că există un **subșir** de lungime j **care se termină cu el**

   - `a[m[1]] ≤ a[m[2]] ≤ … ≤ a[m[n]]`
   - La pasul i – căutăm <u>**binar**</u> cea mai mare lungime j cu

     `a[m[j]] < a[i]`

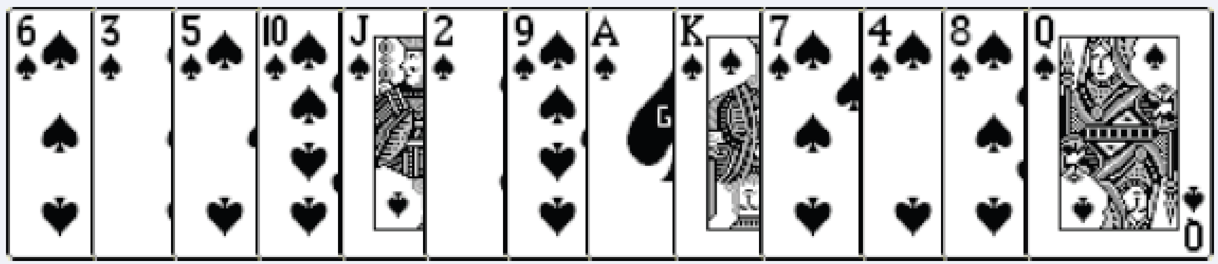
     – dacă găsim j se actualizează m[j+1], altfel se actualizează m[1]

Subşir crescător de lungime maximă

Patience solitaire / patience sort

Amintim problema Greedy: Să se descompună un șir într-un număr minim de subșiruri descrescătoare (monoton)

prima carte

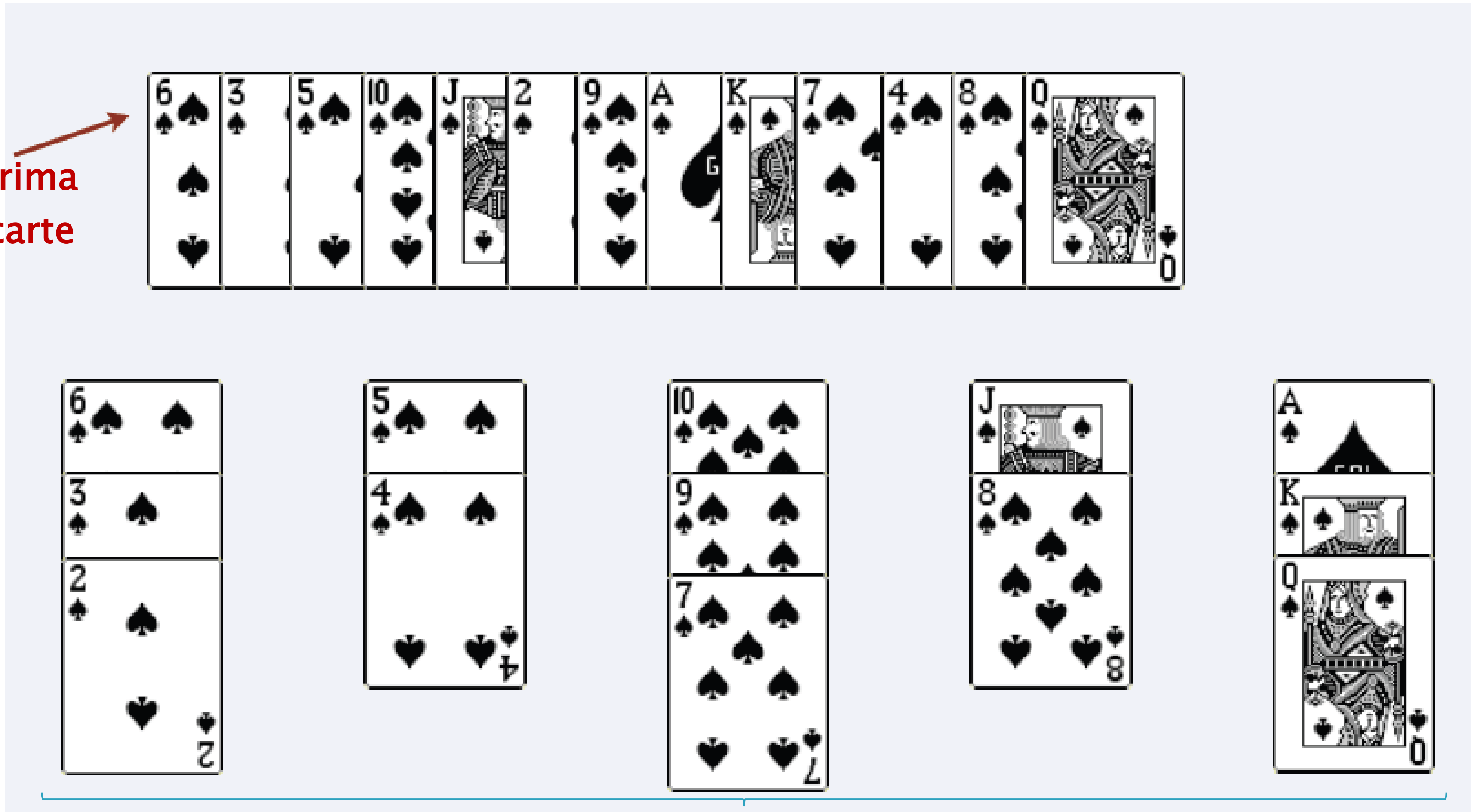


https://www.cs.princeton.edu/courses/archive/spring13/cos423/lectures/LongestIncreasingSubsequence.pdf

- **Vom numi subşirurile şi** teancuri sau stive (similitudine cu Solitaire)

Subşir crescător de lungime maximă

Patience solitaire / patience sort



Număr minim de teancuri

https://www.cs.princeton.edu/courses/archive/spring13/cos423/lectures/LongestIncreasingSubsequence.pdf

Subşir crescător de lungime maximă

Patience solitaire

Algoritm: Greedy – cartea curentă este adăugată **la cel mai din stânga teanc pe care se potrivește**

− La fiecare pas, **cărțile din topul fiecărui teanc formează un șir crescător (! care nu este însă neapărat subșir al șirului inițial)**

- Determinarea celui mai din stânga teanc pe care se potrivește cartea – cu **căutare binară**

– **O(n log n)**

Subşir crescător de lungime maximă

Patience solitaire / patience sort

6, 3, 5, 10, 12, 2, 9, 15, 14, 7, 4,8,13

Subşir crescător de lungime maximă

Patience solitaire / patience sort

3, 5, 10, 12, 2, 9, 15, 14, 7, 4,8,13

6

Subşir crescător de lungime maximă

Patience solitaire / patience sort

5, 10, 12, 2, 9, 15, 14, 7, 4,8,13

6

3

Subşir crescător de lungime maximă

Patience solitaire / patience sort

10, 12, 2, 9, 15, 14, 7, 4,8,13

6 5
3

Subşir crescător de lungime maximă

Patience solitaire / patience sort

12, 2, 9, 15, 14, 7, 4,8,13

6	5	10
3		

Subşir crescător de lungime maximă

Patience solitaire / patience sort

2, 9, 15, 14, 7, 4,8,13

6	5	10	12
3			

Subşir crescător de lungime maximă

Patience solitaire / patience sort

9, 15, 14, 7, 4,8,13

6	5	10	12
3			
2			

Subşir crescător de lungime maximă

Patience solitaire / patience sort

15, 14, 7, 4,8,13

6	5	10	12
3		9	
2			

Subşir crescător de lungime maximă

Patience solitaire / patience sort

14, 7, 4,8,13

6	5	10	12	15
3		9		
2				

Subşir crescător de lungime maximă

Patience solitaire / patience sort

7, 4,8,13

6	5	10	12	15
3		9		14
2				

Subşir crescător de lungime maximă

Patience solitaire / patience sort

4,8,13

6	5	10	12	15
3		9		14
2		7		

Subşir crescător de lungime maximă

Patience solitaire / patience sort

8,13

6	5	10	12	15
3	4	9		14
2		7		

Subşir crescător de lungime maximă

Patience solitaire / patience sort

13

6	5	10	12	15
3	4	9	8	14
2		7		

Subşir crescător de lungime maximă

Patience solitaire / patience sort

6	5	10	12	15
3	4	9	8	14
2		7		13

Subşir crescător de lungime maximă

Patience solitaire / patience sort

Evident:

numărul minim de subșiruri (teancuri) descrescătoare în care se poate descompune un șir $\geq$

lungimea maximă a unui subșir crescător

Subşir crescător de lungime maximă

Patience solitaire / patience sort

Evident:

numărul minim de subșiruri (teancuri) descrescătoare în care se poate descompune un șir $\geq$ **lungimea maximă a unui subșir crescător**

(două elemente dintr-un subșir crescător nu pot fi în același teanc)

Subşir crescător de lungime maximă

Patience solitaire / patience sort

Proprietate:

numărul minim de subșiruri descrescătoare în care se poate descompune un șir =

lungimea maximă a unui subșir crescător

Subşir crescător de lungime maximă

Patience solitaire / patience sort

Proprietate:

numărul minim de subșiruri descrescătoare în care se poate descompune un șir =

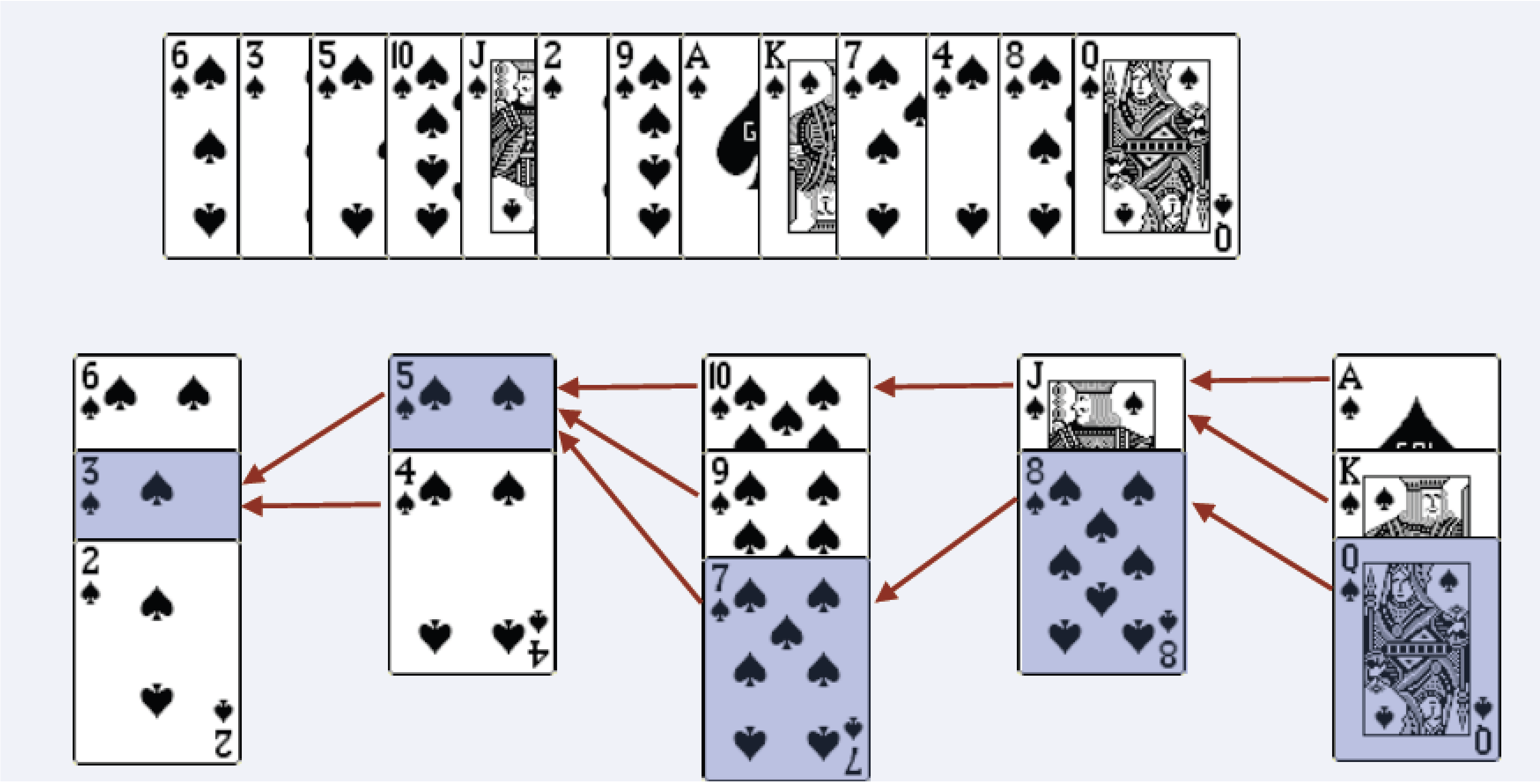
lungimea maximă a unui subșir crescător

- Pentru a memora un subșir crescător, memorăm la fiecare pas al algoritmului Greedy, pentru cartea curent adăugată o **legătură de tip predecesor către vârful teancului anterior** celui în care a fost adăugată

- **Un subșirul crescător de lungime maximă al șirului inițial se obține pornind de la o carte din <u>ultimul teanc</u> și urmând legătura predecesor ⇒ algoritm O(n log n)**

Subşir crescător de lungime maximă

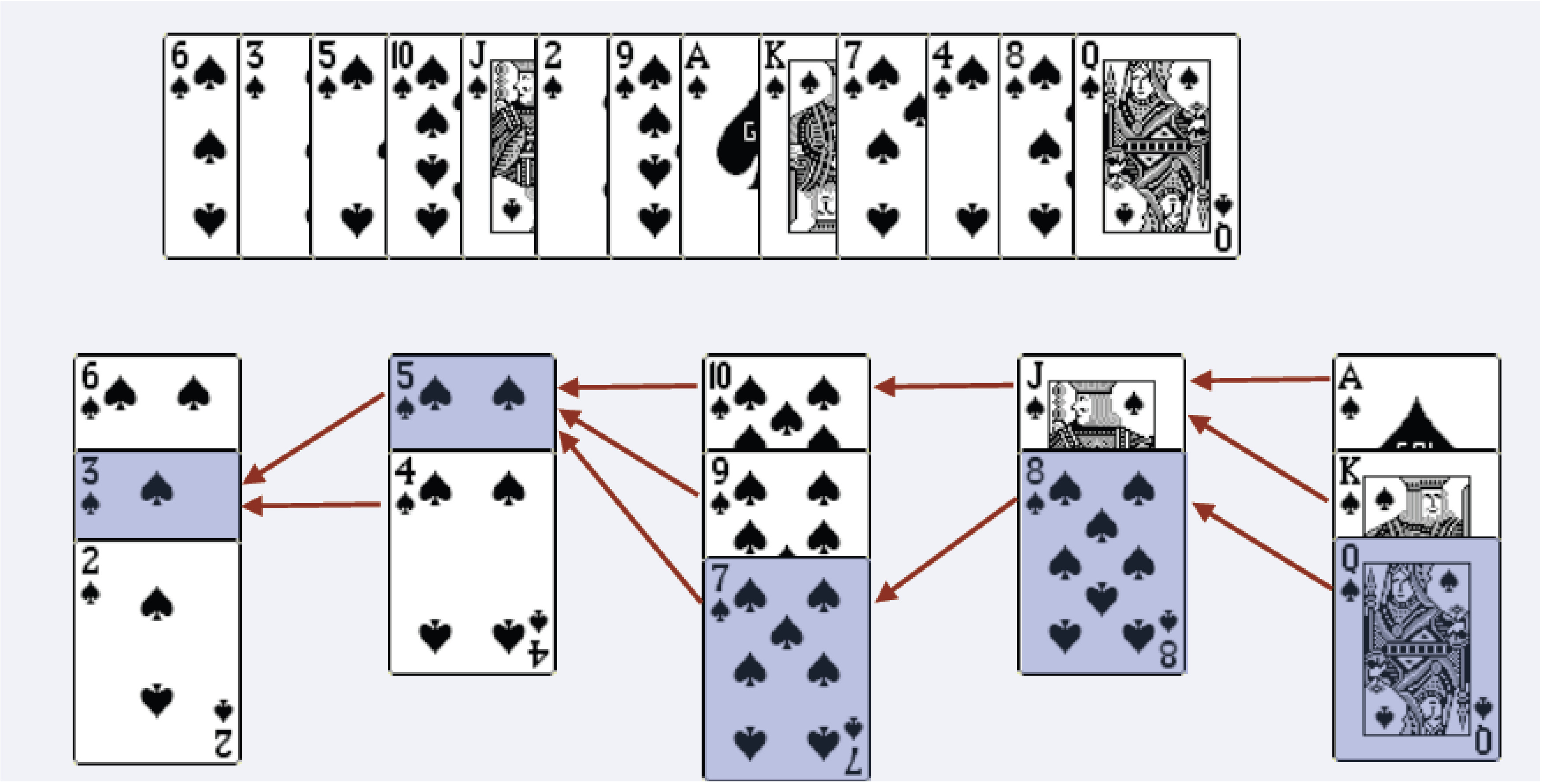
Patience solitaire / patience sort



https://www.cs.princeton.edu/courses/archive/spring13/cos423/lectures/LongestIncreasingSubsequence.pdf

Subşir crescător de lungime maximă

Patience solitaire / patience sort



Vârful teancului j = cel mai mic element cu care se termină un subșir crescător de lungime j al șirului inițial

(<u>aceeași idee ca prima, vârful teancului j = a[m[j]]</u>)

Subşir crescător de lungime maximă

Descompunerea în subşiruri descrescătoare

3, 5, 10, 12, 2, 9, 15, 14, 7, 4,8,13

6

Subşir crescător de lungime maximă

Descompunerea în subşiruri descrescătoare

5, 10, 12, 2, 9, 15, 14, 7, 4,8,13

6

3

Subşir crescător de lungime maximă

Descompunerea în subşiruri descrescătoare

10, 12, 2, 9, 15, 14, 7, 4,8,13

6 5

3

Subşir crescător de lungime maximă

Descompunerea în subşiruri descrescătoare

12, 2, 9, 15, 14, 7, 4,8,13

6 5 10

3

Subşir crescător de lungime maximă

Descompunerea în subşiruri descrescătoare

2, 9, 15, 14, 7, 4,8,13

6   5 ← 10 ← 12

3

Subşir crescător de lungime maximă

Descompunerea în subşiruri descrescătoare

9, 15, 14, 7, 4,8,13

6	5 ←	10 ←	12
3			
2			

Subşir crescător de lungime maximă

Descompunerea în subşiruri descrescătoare

15, 14, 7, 4,8,13

6	5	10	12
3		9	
2			

Subşir crescător de lungime maximă

Descompunerea în subşiruri descrescătoare

14, 7, 4,8,13

6 5 ← 10 ← 12 ← 15

3 9

2

Subşir crescător de lungime maximă

Descompunerea în subşiruri descrescătoare

7, 4,8,13

6 5 10 12 15

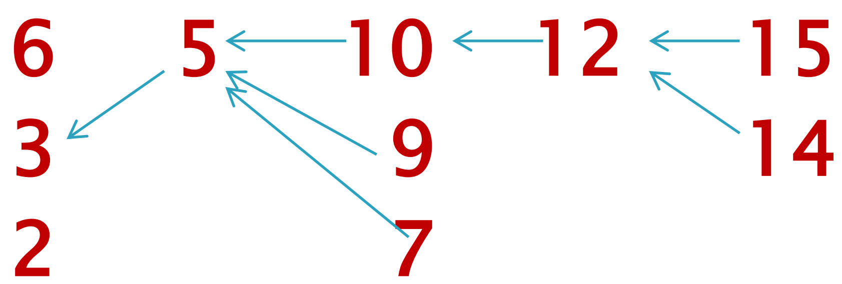
3 9 14

2

Subşir crescător de lungime maximă

Descompunerea în subşiruri descrescătoare

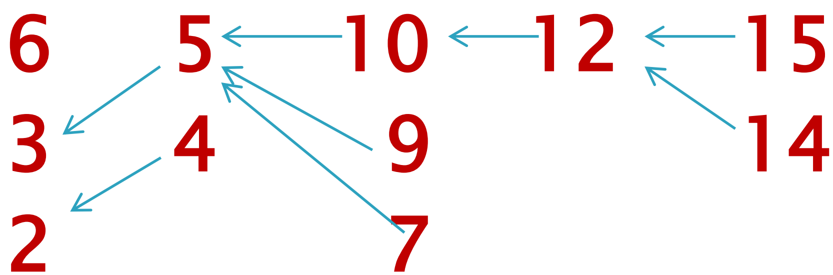
4,8,13



Subşir crescător de lungime maximă

Descompunerea în subşiruri descrescătoare

8,13



Subşir crescător de lungime maximă

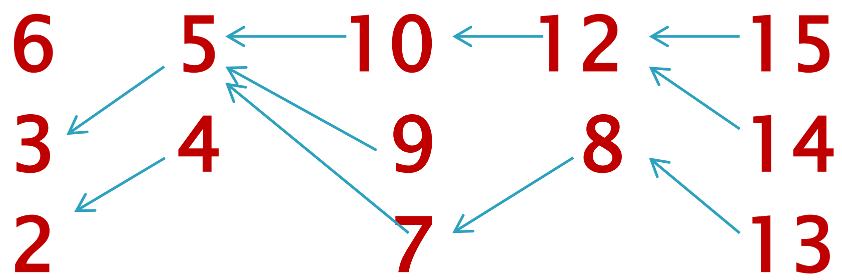
Descompunerea în subşiruri descrescătoare

13

6	5	10	12	15
3	4	9	8	14
2		7		

Subşir crescător de lungime maximă

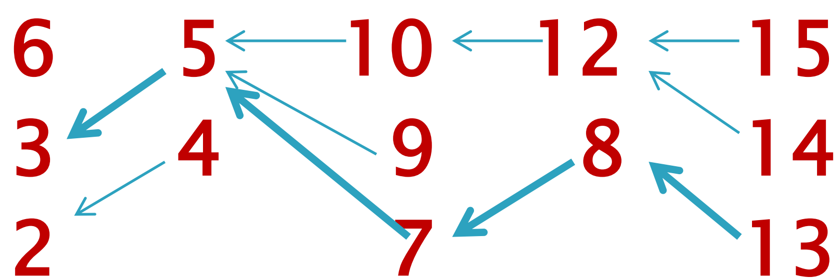
Descompunerea în subşiruri descrescătoare



Subşir crescător de lungime maximă

Descompunerea în subşiruri descrescătoare

6, 3, 5, 10, 12, 2, 9, 15, 14, 7, 4,8,13

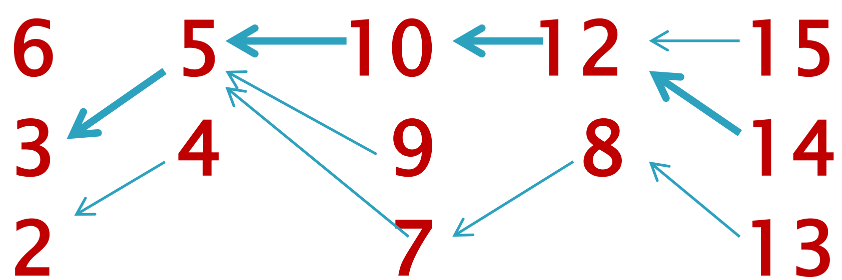


3, 5, 7, 8,13

Subşir crescător de lungime maximă

Descompunerea în subşiruri descrescătoare

6, 3, 5, 10, 12, 2, 9, 15, 14, 7, 4,8,13

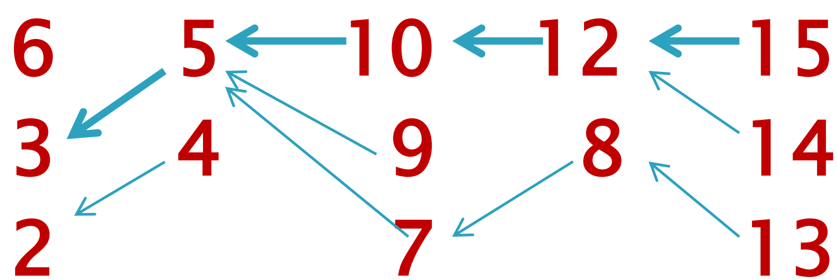


3, 5, 10, 12,14

Subşir crescător de lungime maximă

Descompunerea în subşiruri descrescătoare

6, 3, 5, 10, 12, 2, 9, 15, 14, 7, 4,8,13



3, 5, 10, 12,15