Metoda
Programării Dinamice

Exemplu

Mulțime independentă de pondere maximă

Se consideră vectorul $w = (w_1,\dots, w_n)$.

Să se determine un subşir al lui w care nu conține elemente consecutive în w și are suma elementelor maximă

<u>**Exemplu**</u>

Pentru

$$w = (1, 4, 7, 5)$$

soluția este

$$(4, 5)$$

Mulțime independentă de pondere maximă

- **Problemă echivalentă**

Se consideră un graf de tip lanț cu $V = \{v_1,\dots, v_n\}$.

Vârfurile grafului au asociate ponderile $w_1,\dots,$ respectiv w_n.

Să se determine o mulțime independentă de vârfuri (neadiacente) de pondere maximă

- ponderea unei mulțimi de vârfuri = suma ponderilor vârfurilor

Mulțime independentă de pondere maximă

- **Aplicații –** probleme de alocare de resurse cu evitarea interferenței (indicată prin muchii -> graf de conflicte)

  Ponderea asociată vârfurilor poate reprezenta cantitatea de date pe care stația trebuie să o transmită

  Problemă – transmiterea cantității maxime de date fără interferențe

- **Maximum Weighted Independent Set**

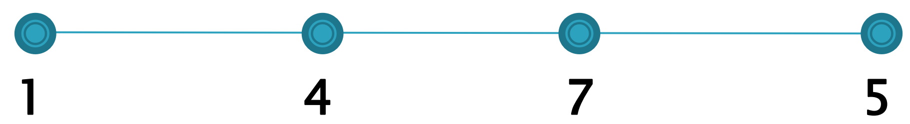
Mulțime independentă de pondere maximă

- **Maximum Weighted Independent Set**
  - pentru **graful lanț** – curs O(|V|)
  - pentru **arbori** – seminar/laborator O(|V|)
  - pentru **grafuri de intervale** (asociate intersecției intervalelor) = problema spectacolelor (planificării activităților) cu ponderi – tema laborator O(|V| log(|V|))

  - nu se cunosc algoritmi polinomiali în **cazul general** (NP-completitudine)

Mulțime independentă de pondere maximă

- **Abordare cu metoda Greedy**

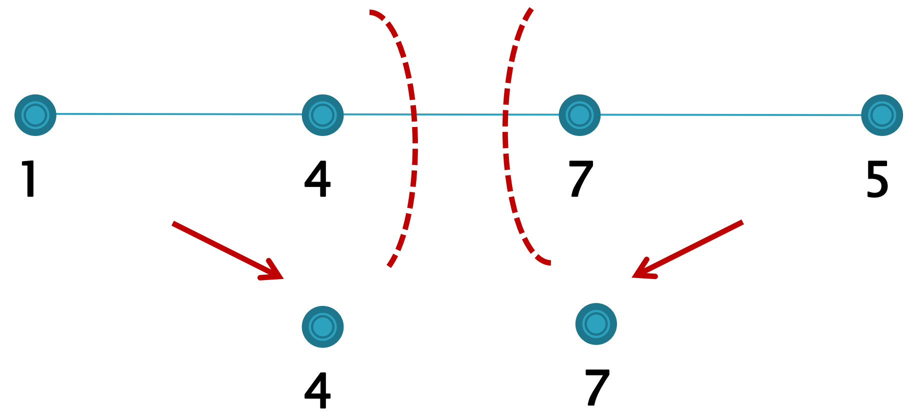


La un pas – este adăugat în soluție vârful de pondere maximă neadiacent cu cele deja selectate

Soluția: $\{v_3, v_1\}$ cu ponderile $\{7, 1\}$ – **nu este optimă**

Mulțime independentă de pondere maximă

- **Abordare cu metoda Divide et Impera**



Incorect – reuniunea soluțiilor subproblemelor nu este o soluție posibilă (corectă) pentru problema inițială

Mulțime independentă de pondere maximă

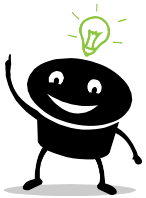
- **Căutarea exhaustivă a soluției optime**

Generarea tuturor soluțiilor posibile și determinarea celei optime – algoritm exponențial

Mulțime independentă de pondere maximă

Problema nu se poate rezolva folosind metode deja studiate

Mulțime independentă de pondere maximă



Analizăm structura unei soluții optime, evidențiind un element (primul/ultimul) al acesteia, pentru a determina subprobleme utile și relații de recurență

Mulțime independentă de pondere maximă

Fie $S \subseteq V = \{v_1, \ldots, v_n\}$ o soluție optimă

- Dacă $v_n \in S \Rightarrow S - \{v_n\}$ este soluție optimă pentru

  $G - \{v_n, v_{n-1}\}$

Mulțime independentă de pondere maximă

Fie $S \subseteq V = \{v_1,..., v_n\}$ o soluție optimă

- Dacă $v_n \in S \Rightarrow S-\{v_n\}$ este soluție optimă pentru

  $G-\{v_n, v_{n-1}\}$

- Dacă $v_n \notin S \Rightarrow S$ este soluție optimă pentru

  $G-\{v_n\}$

Mulțime independentă de pondere maximă

Fie $S \subseteq V = \{v_1, \ldots, v_n\}$ o soluție optimă

- Dacă $v_n \in S \Rightarrow S - \{v_n\}$ este soluție optimă pentru

  $G - \{v_n, v_{n-1}\}$

- Dacă $v_n \notin S \Rightarrow S$ este soluție optimă pentru

  $G - \{v_n\}$

- **Dacă am ști deja soluțiile pentru grafurile $G - \{v_n, v_{n-1}\}$ și $G - \{v_n\}$, am putea determina S alegând dintre cele două situații cazul în care se obține soluția optimă**

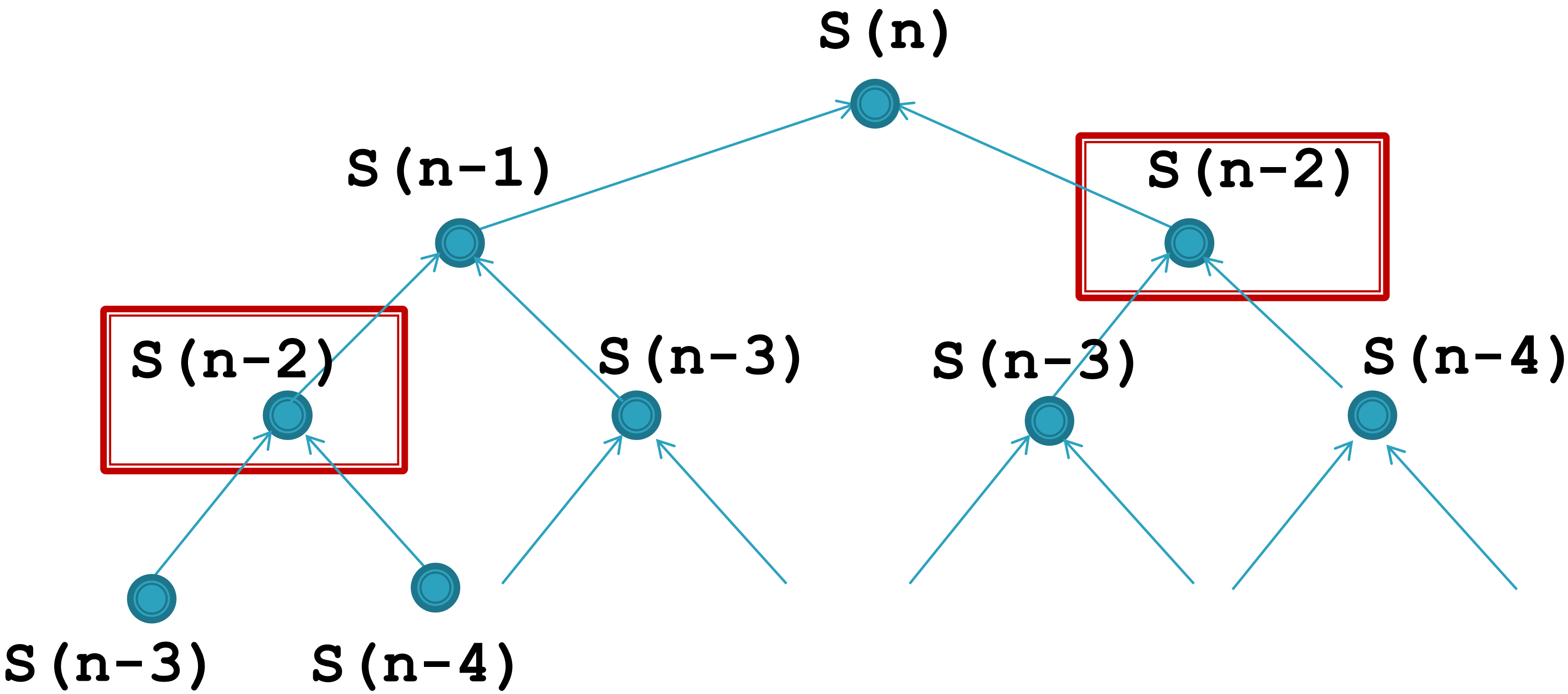
Mulțime independentă de pondere maximă

Recurență:

- Notăm $S(i)$ = ponderea maximă a unei mulțimi independete în graful indus de vârfurile $\{v_1,\ldots, v_i\}$

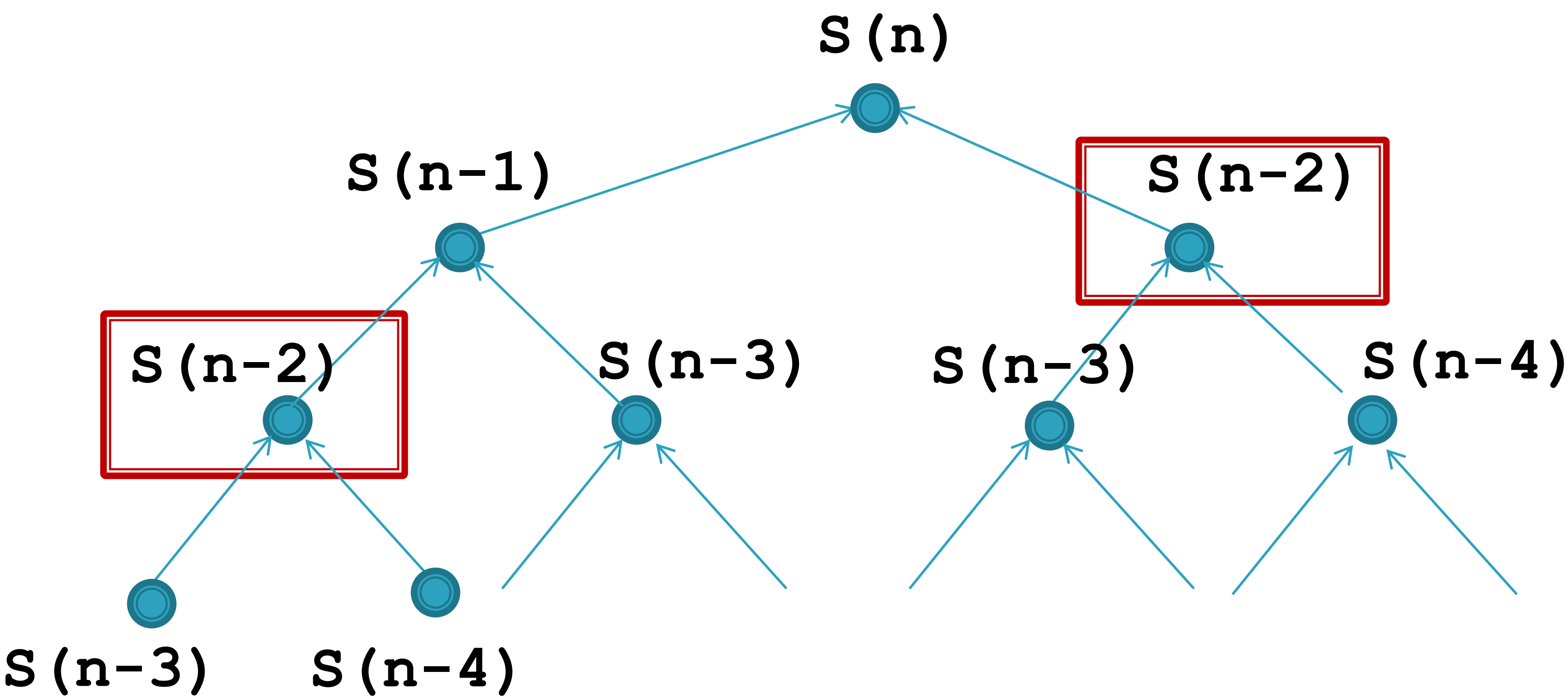
- **$S(n) = \max\{ S(n-2)+w_n, S(n-1)\}$**
- **$S(1) = w_1$, $S(0) = 0$**

Mulțime independentă de pondere maximă



Subproblemele se repetă – algoritm exponențial

Mulțime independentă de pondere maximă



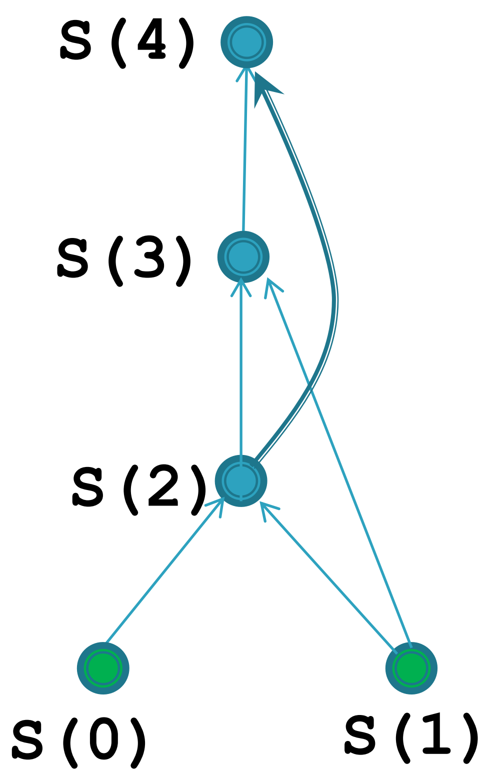
Putem evita rezolvarea unei subprobleme de mai multe ori?

Mulțime independentă de pondere maximă

Recurență:

- **$S(n)= \max\{ S(n-2)+w_n, S(n-1)\}$**
- **Memorăm într-un vector rezultatele subproblemelor deja rezolvate (memoizare)** $\Rightarrow$ o subproblemă va fi rezolvată o singură dată – algoritm O(n)

Mulțime independentă de pondere maximă



Mulțime independentă de pondere maximă

- **Variantă**– implementare iterativă a recurenței
  (bottom-up)

Implementare recursivă – memoizare

```
void sol(int n){
    if(n==0) {s[0]=0;return;}
    if(n==1) {s[1]=w[1]; return ;}
    if (s[n-1]==0) //nerezolvata
          sol(n-1);
    if (s[n-2]==0) //nerezolvata
          sol(n-2);
    s[n]= max(s[n-2]+w[n],s[n-1]);
}
```

```
for(int i=0;i<=n;i++)
     s[i]=0; //initial-nerezolvate pt n>1
```

Implementare iterativă

```
int SolNerec(int n){
        s[0] = 0;
        s[1] = w[1];
        for(int i=2;i<=n;i++)
                s[i] = max(s[i-2]+w[i],s[i-1]);
        return s[n];
 }

 int solNerecFaraVector(int n){ //similar Fibonacci
        if(n==0) return 0;
        int i,s0,s1,si;
        s0 = 0;  s1 = w[1];
        for(i=2;i<=n;i++){
                si = max(s0+w[i],s1);
                s0 = s1;  s1 = si;
        }
        return s1;
  }
```

Mulțime independentă de pondere maximă

Cum putem determina și o submulțime optimă, nu doar ponderea?

Mulțime independentă de pondere maximă

Cum putem determina și o submulțime optimă, nu doar ponderea?

- Din relația de recurență putem deduce ce vârfuri au fost selectate în soluție

$$s[n]= \max\{ s[n-2]+w[n],\ s[n-1]\}$$

Mulțime independentă de pondere maximă

Cum putem determina și o submulțime optimă, nu doar ponderea?

- Din relația de recurență putem deduce ce vârfuri au fost selectate în soluție

$$s[n]= \max\{ s[n-2]+w[n],\ s[n-1]\}$$

  - Dacă $s[n] = s[n-2]+w[n]$, vârful n se adaugă în soluție și problema se reduce la primele n-2 vârfuri
  - Dacă $s[n] = s[n-1]$, nu se adaugă nici un vârf la soluție și problema se reduce la primele n-1 vârfuri

```
void afisRec(int s[],int n){
    if(n==0) return;
    if(n==1){
          cout<<n<<" de pondere "<<w[n]<<endl;
          return;
    }
    if(s[n]==s[n-2]+w[n]){
          afisRec(s,n-2);
           cout<< n<<" de pondere "<w[n];
    }
    else
          afisRec(s,n-1);
}
```

Mulțime independentă de pondere maximă

TEMĂ – rezolvați problema pentru un arbore oarecare O(n)

Concluzii

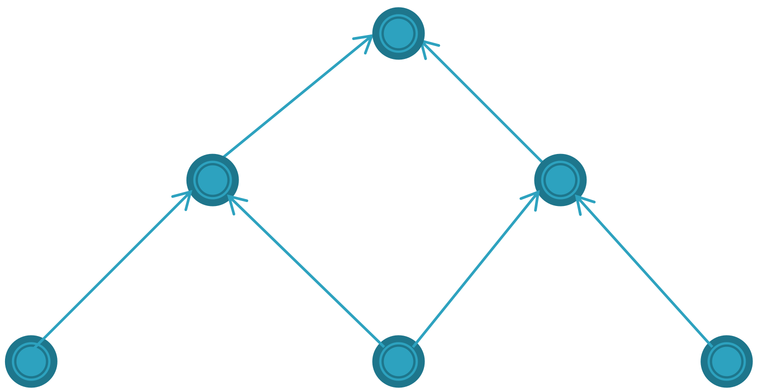
Dezavantaje ale metodelor deja studiate

- Greedy – nu furnizează mereu soluția optimă

- Alte exemple:
  - Problema rucsacului, cazul discret
  - Problema monedelor, cazul general

Dezavantaje ale metodelor deja studiate

- Divide et Impera – ineficientă dacă subproblemele se repetă



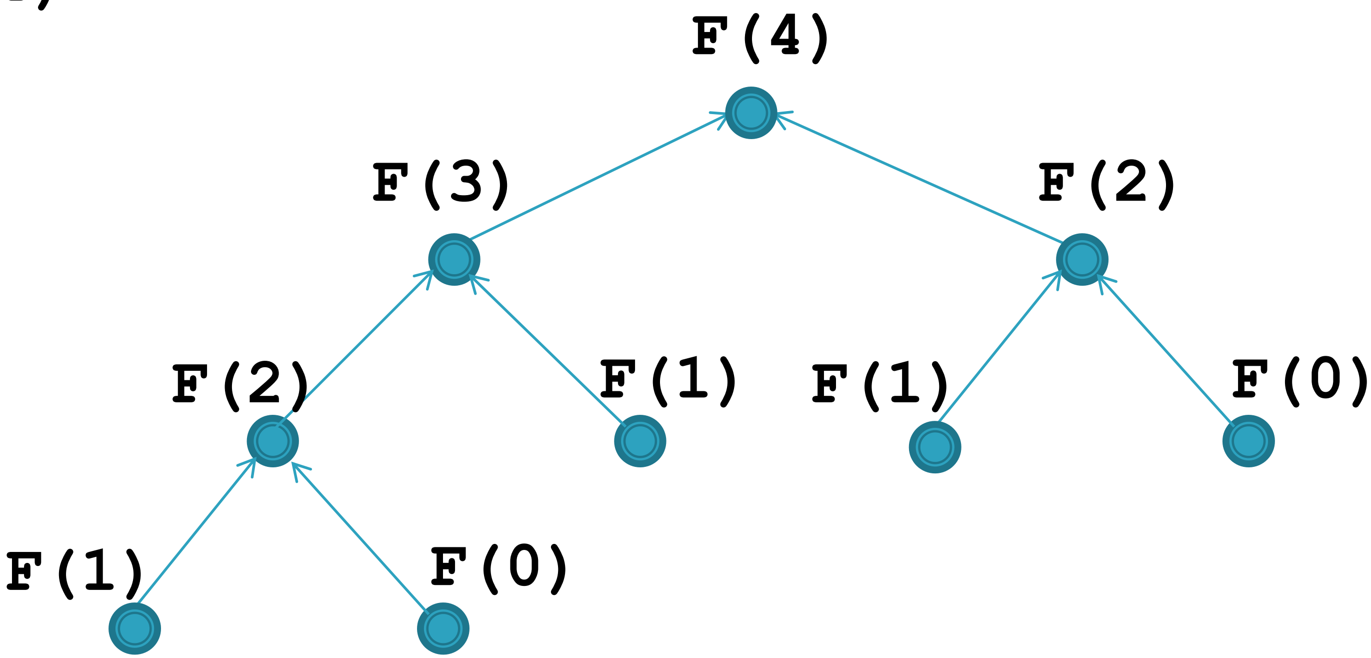
Dezavantaje ale metodelor deja studiate

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

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

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

- **`F(4)`**



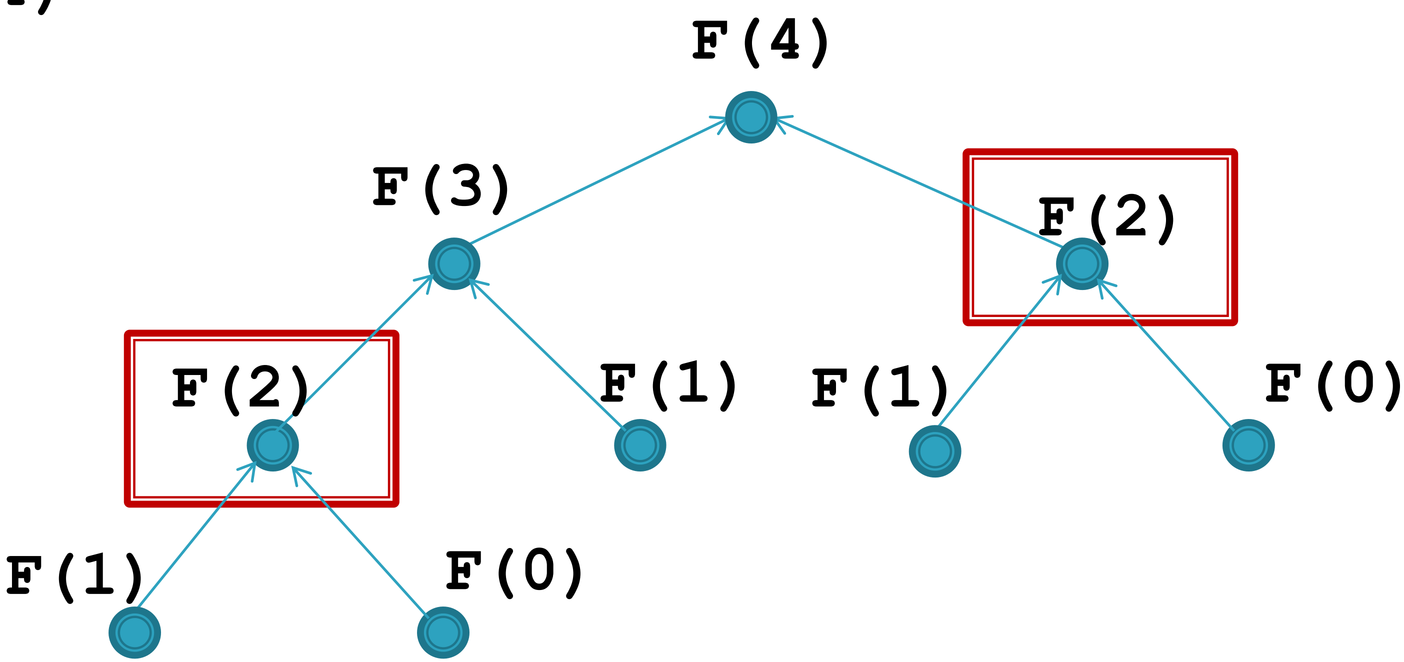
Dezavantaje ale metodelor deja studiate

- **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

- **Soluții**
  - **reducere la subprobleme utile + relații de recurență**
  - **rezolvarea eficientă a subproblemelor**
    - recursiv cu memoizare (salvarea rezultatelor subproblemelor deja rezolvate)
    - algoritmi iterativi buttom-up

↓

Metoda programării dinamice