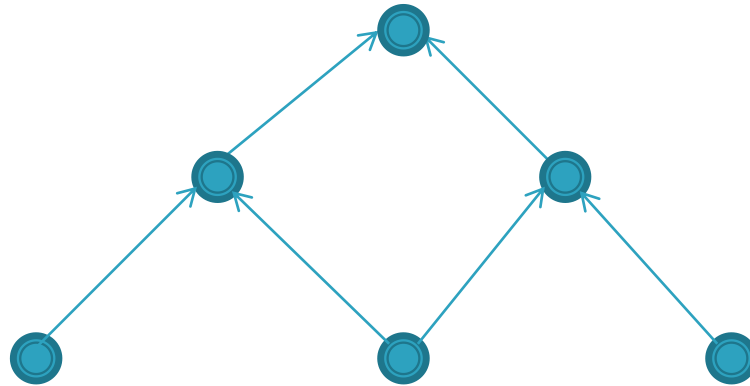


# Metoda Programării Dinamice

# Dezavantaje ale metodelor deja studiate

- Greedy – nu furnizează mereu soluția optimă
- Divide et Impera – inefficientă dacă subproblemele se repetă / se suprapun



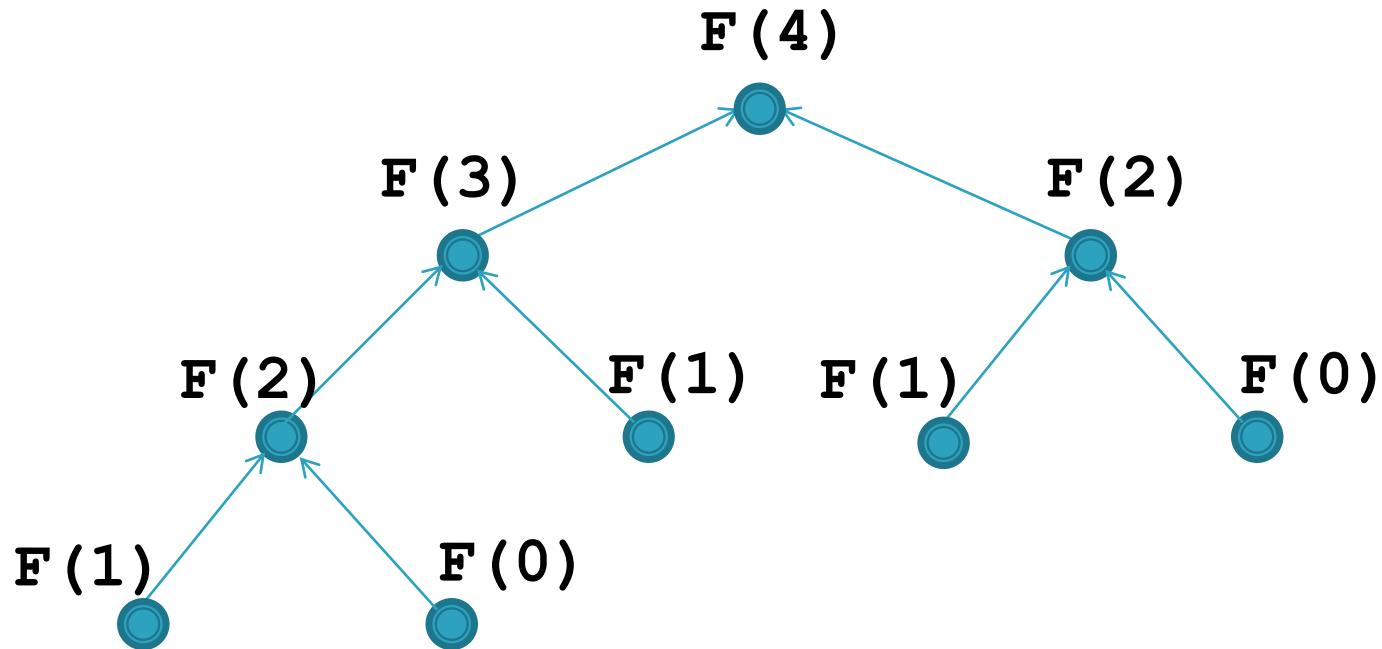
# Dezavantaje ale metodelor deja studiate

- Exemplu – Calculăm numărul Fibonacci  $F(n)$

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

$$F(0) = F(1) = 1$$

- ▶  $F(4)$



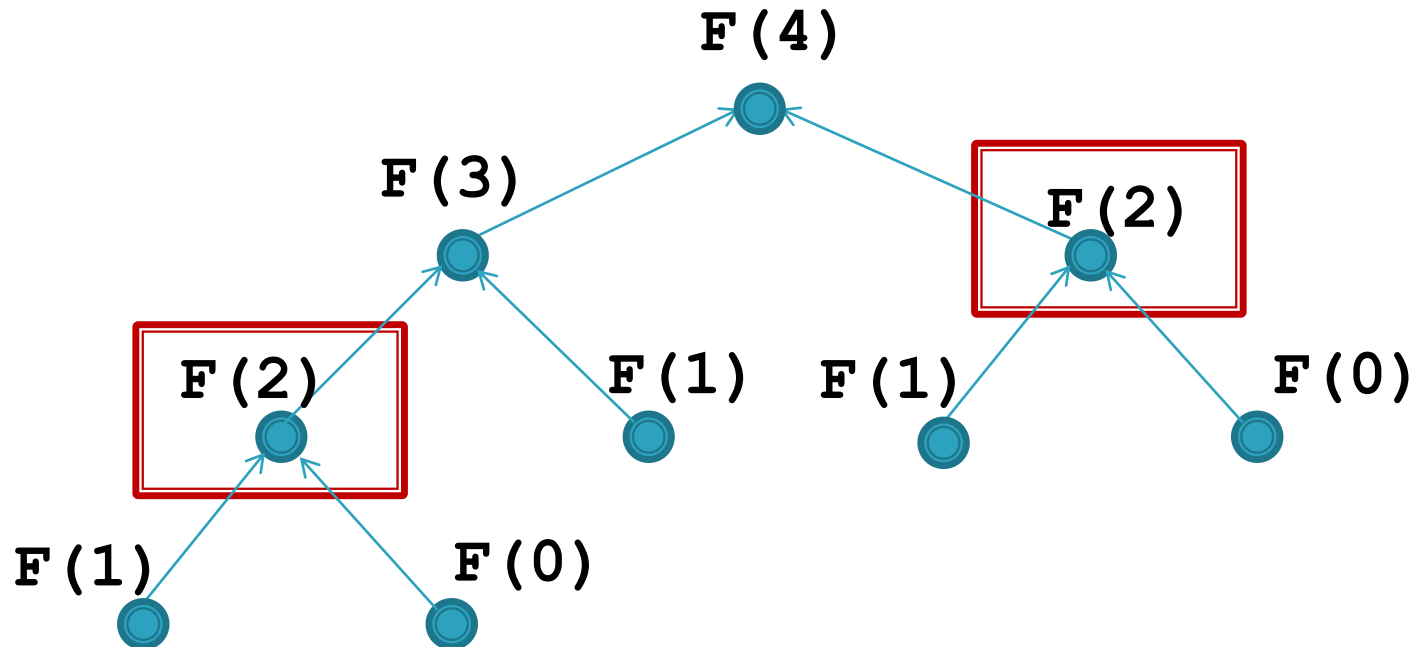
# Dezavantaje ale metodelor deja studiate

- 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 (**care se suprapun**) + determinarea de relații de recurență
  - rezolvarea eficientă a subproblemelor (recurențelor), cu **memoizare** = memorarea soluțiilor subproblemelor deja rezolvate (**pentru a nu le recalcula/rezolva din nou**)



# Metoda programării dinamice



Cum putem obține relații de recurență?

# Metoda programării dinamice



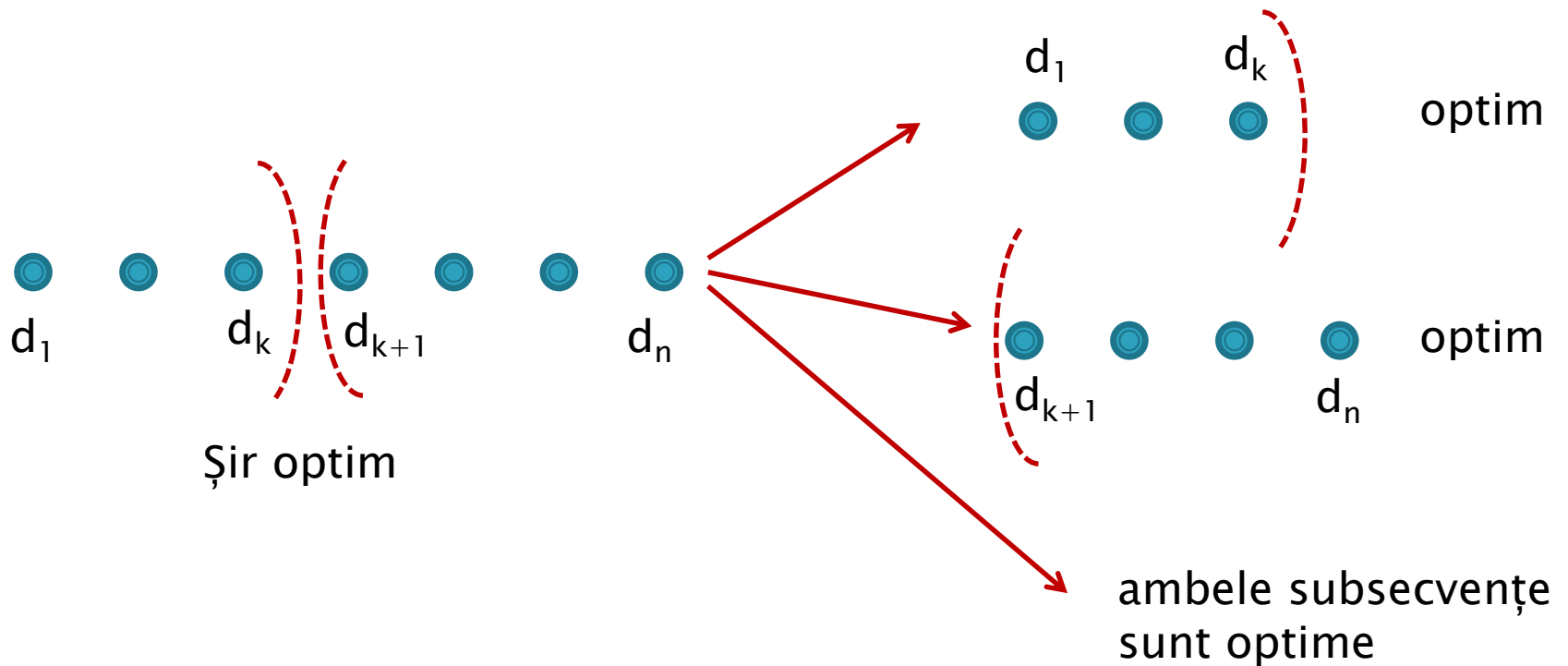
Cum putem obține relații de recurență?

- ▶ Exemplu: În **problemele de optim** – care verifică un principiu de optimalitate, din care se obțin relațiile de calcul

# Metoda programării dinamice

Fie soluția optimă  $d_1, \dots, d_n$

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



# Metoda programării dinamice

Fie soluția optimă  $d_1, \dots, d_n$

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

(1)  $d_1, d_2, \dots, d_n$  optim  $\Rightarrow d_k, \dots, d_n$  optim pentru subproblema corespunzătoare,  $\forall 1 \leq k \leq n$

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

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

și

$d_{k+1}, \dots, d_n$  optim,  $\forall 1 \leq k \leq n$

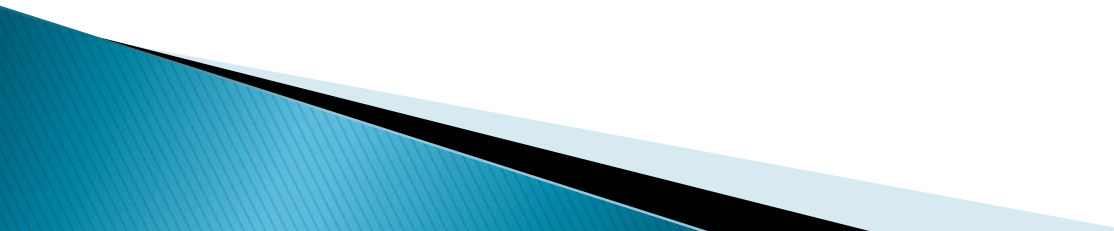
# Etape

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

# Etape

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

# Etape

- ▶ **Stabilirea subproblemelor utile** (de exemplu din principiul de optimalitate)
  - ▶ **Cum putem rezolva problema inițială 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 inițială folosind subproblemele**
  - ▶ **Care subprobleme le putem rezolva direct**
  - ▶ **Relațiile de recurență**
  - ▶ **Ordinea de rezolvare a recurențelor**
- 