

NP-Completitudine

Algoritmi esponenziali

Metode de elaborare a algoritmilor

- **Greedy** – probleme de optim
 - este necesară demonstrarea corectitudinii
 - algoritmi polinomiali
- **Divide et impera**
- **Programare dinamică**

Metode de elaborare a algoritmilor

- **Greedy** – probleme de optim
 - corectitudine (v. curs moodle actualizat)
 - algoritmi polinomiali
- **Divide et impera** – subprobleme de același tip
 - **construirea dinamică a unui arbore** (prin împărțirea în subprobleme) urmată de **parcurgerea în postordine a arborelui** (prin asamblarea rezultatelor parțiale).
 - algoritmi polinomiali
- **Programare dinamică**

Metode de elaborare a algoritmilor

- **Greedy** – probleme de optim
 - corectitudine (v. curs moodle actualizat)
 - algoritmi polinomiali
- **Divide et impera** – subprobleme de același tip
 - **construirea dinamică a unui arbore**
(prin împărțirea în subprobleme) urmată de **parcurgerea în postordine a arborelui** (prin asamblarea rezultatelor parțiale).
 - algoritmi polinomiali
- **Programare dinamică** – rezolvare de recurențe->PD-arbore
 - principiu de optimalitate
 - **parcurgerea în “postordine” generalizată a PD-arborelui**
 - algoritmi polinomiali + **pseudo** polinomiali

Metode de elaborare a algoritmilor

- Complexitatea în timp a algoritmilor joacă un rol esențial.
- **Un algoritm este considerat "acceptabil" numai dacă timpul său de executare este polinomial**

Metode de elaborare a algoritmilor

Nu știm algoritm polinomial – problemă grea?

Metode de elaborare a algoritmilor

Nu știm algoritm polinomial – problemă grea?

P = clasa problemelor pentru care există algoritmi polinomiali (determiniști)

Metode de elaborare a algoritmilor

Nu știm algoritm polinomial – problemă grea?

NP

- există algoritm polinomial pentru a testa o soluție candidat dacă este soluție posibilă

(verificator polinomial)

⇒ o problemă NP poate fi rezolvată în timp exponențial (considerând toate soluțiile candidat)

Metode de elaborare a algoritmilor

Nu știm algoritm polinomial – problemă grea?

NP

- $P \neq NP$?

- Probleme NP-complete (NP, NP-hard)

- $B \in NP$ a.î. $\forall A \in NP, A \leq_p B$.
- Dacă pentru una se găsește algoritm polinomial, atunci $P = NP$
- SAT

- Probleme NP-dificile (NP-hard)

- B a.î. $\forall A \in NP, A \leq_p B$.

Metode de elaborare a algoritmilor

Nu știm algoritm polinomial

- **Demonstrăm NP – dificilă**

Soluții:

- algoritmi exponențiali mai rapizi decât cei exhaustivi (brute force) de căutare în spațiul soluțiilor: **Backtracking, Branch & Bound**
- **Compromis:** algoritmi mai rapizi care produc soluții care nu sunt optime – algoritmi **euristici, aleatorii, genetici...**
- O euristică este o metodă de a clasifica alegerile posibile la un anumit pas (cu scopul de a determina o alegerea optimă a pasului următor) în explorarea spațiului soluțiilor unei probleme.

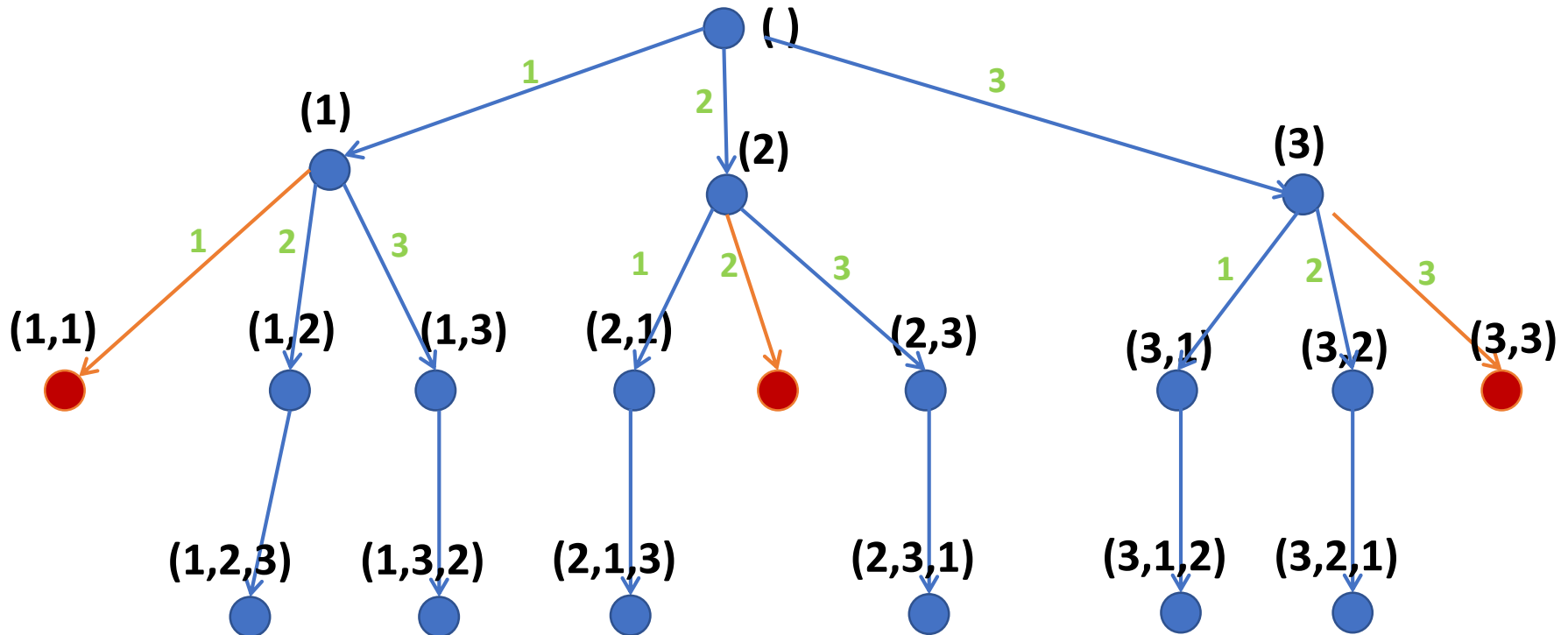
Backtracking, Branch and Bound

Cadru comun

- Căutare mai “inteligentă” în spațiul în care se găsesc soluțiile posibile, reprezentate de obicei cu ajutorul vectorilor
- Configurațiile prin care se trece în procesul de căutare - structură arborescentă

Cadru comun

- Structură arborescentă – permutări $\{1, 2, 3\}$



Cadru comun

- Parcurgerea completă a arborelui \Rightarrow algoritm exhaustiv (brute force), care consideră toate soluțiile candidat

Cadru comun

- Parcurgerea completă a arborelui \Rightarrow algoritm exhaustiv (brute force), care consideră toate soluțiile candidat
- Mai rapid – **limitarea parcurgerii** arborelui prin determinarea de configurații care **nu pot conduce către soluții** dorite, care nu mai sunt explorate

Cadru comun

- Parcurgerea completă a arborelui \Rightarrow algoritm exhaustiv (brute force), care consideră toate soluțiile candidat
 - Mai rapid – **limitarea parcurgerii** arborelui prin determinarea de configurații care **nu pot conduce către soluții** dorite, care nu mai sunt explorate
 - **Diferențe**
 - modul în care este parcurs arborele (și în care se fac limitările)
- = criteriul după care este ales nodul curent
- tipuri de probleme la care se pretează

Metoda Backtracking

Cadru

- **Probleme de satisfacere a unor constrângeri:**
 - variabile x_1, \dots, x_n
 - domeniul de valori pentru fiecare variabilă
 - constrângerea φ

Cadru

- $X = X_1 \times \dots \times X_n =$ **spațiul soluțiilor posibile**
- $\varphi : X \rightarrow \{0,1\}$ este o **proprietate** definită pe X
- **Căutăm un vector $x \in X$ cu proprietatea $\varphi(x)$**
 - condiții interne pentru x

Metoda Backtracking

Metoda backtracking încearcă micșorarea timpului de calcul
- prin **evitarea generării unor soluții care nu satisfac condițiile interne**

Metoda Backtracking

- Soluțiile se construiesc element cu element, existând configurații corespunzătoare soluțiilor parțiale (“incomplete”), pe care le putem **testa** dacă pot fi completate până la o soluție posibilă -> condiții de continuare
 - condiții de continuare pentru soluția parțială $y=x_1...x_k$
notate $cont_k(x)$ = condiții de continuare a parcurgerii subarborelui de rădăcină x

Metoda Backtracking

- Probleme computaționale + cu constrângeri (constraint satisfaction problems) - puzzle, colorarea hărților, problema damelor
- Combinatorică, probleme optimizări combinatoriale – rucsac
- Căutare IA

Metoda Backtracking

- Vectorul soluție $x = (x_1, x_2, \dots, x_n) \in X$ este **construit progresiv**, începând cu prima componentă.

Metoda Backtracking

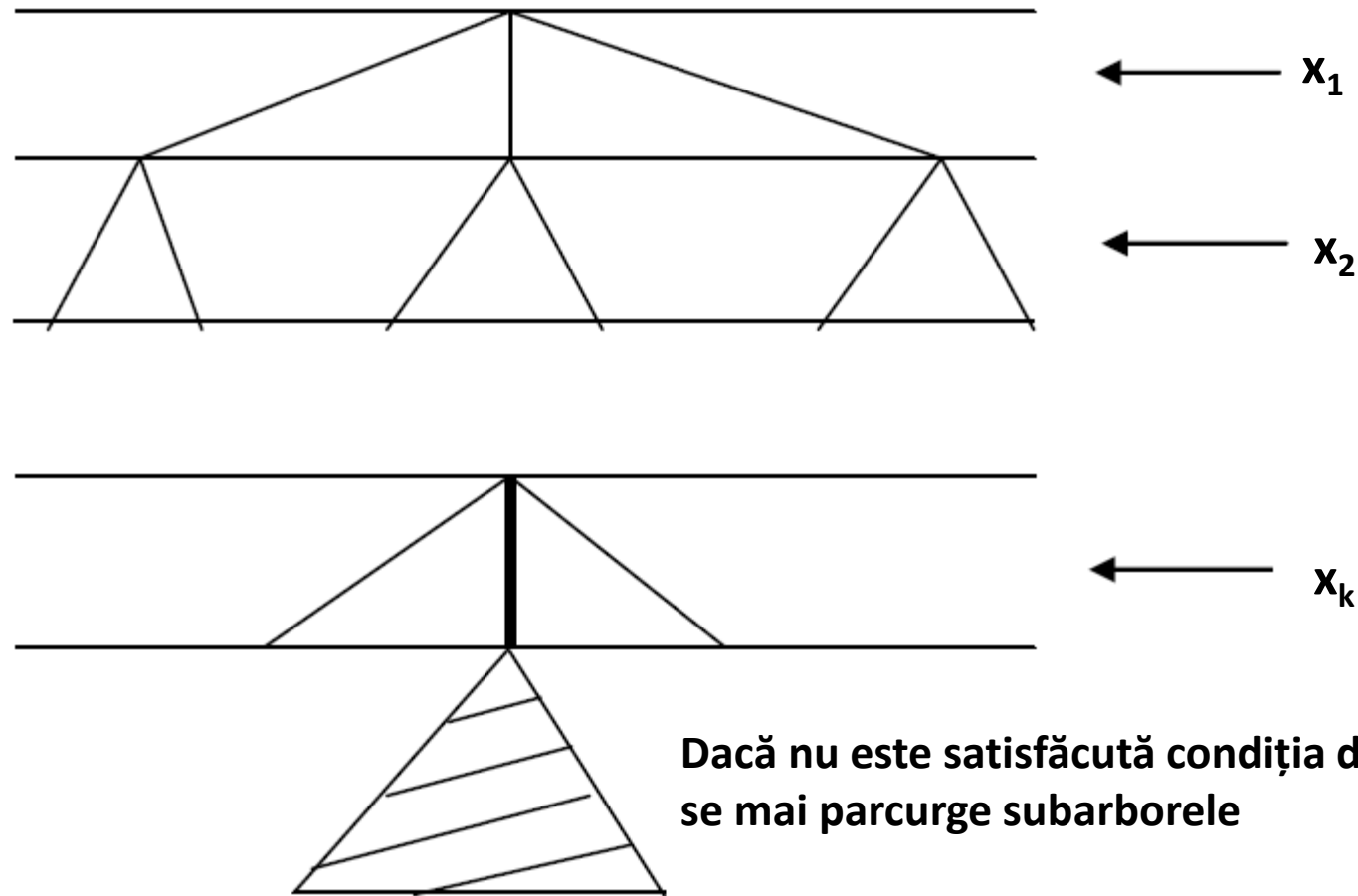
- Vectorul soluție $x = (x_1, x_2, \dots, x_n) \in X$ este **construit progresiv**, începând cu prima componentă.
- Se avansează cu o valoare pentru x_k dacă este satisfăcută **condiția de continuare** $\text{cont}_k(x_1, \dots, x_k)$

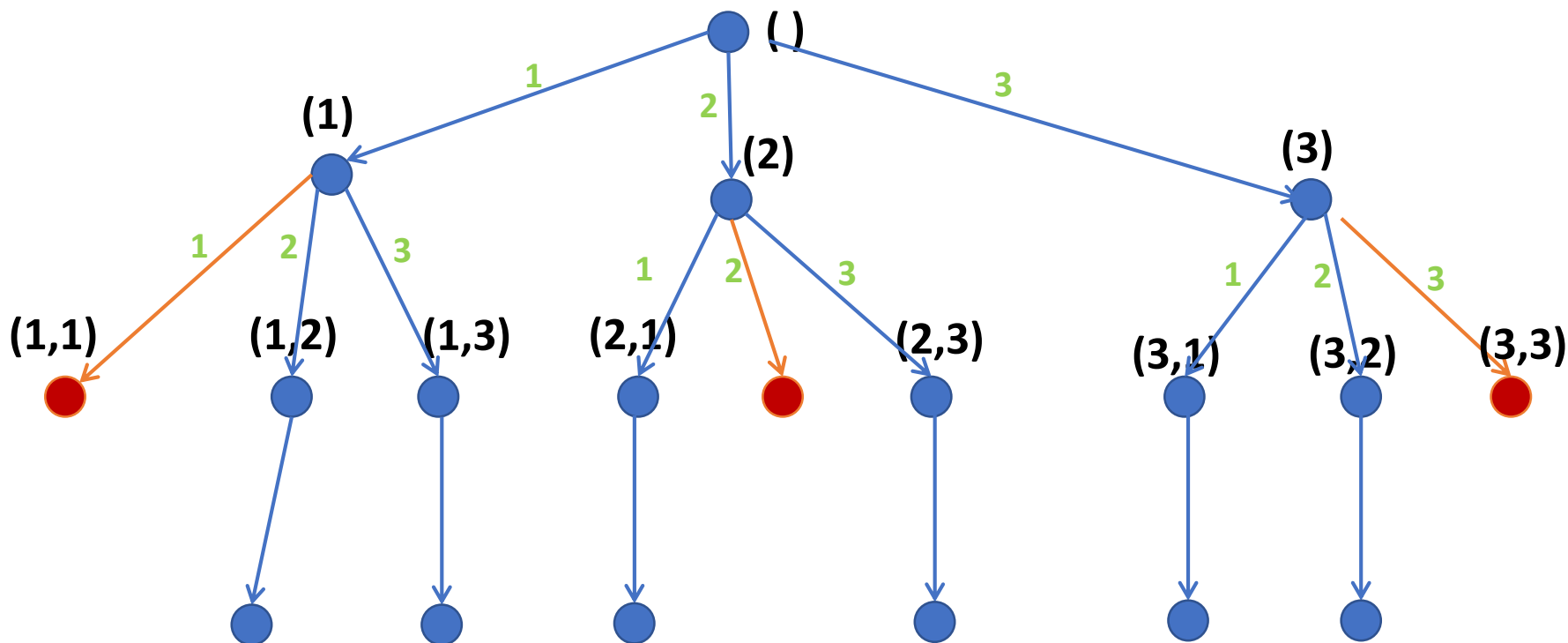
Metoda Backtracking

- Vectorul soluție $x = (x_1, x_2, \dots, x_n) \in X$ este **construit progresiv**, începând cu prima componentă.
- Se avansează cu o valoare pentru x_k dacă este satisfăcută **condiția de continuare** $\text{cont}_k(x_1, \dots, x_k)$
- Condițiile de continuare rezultă de obicei din φ . Ele sunt strict necesare, **ideal fiind să fie și suficiente**.

Metoda Backtracking

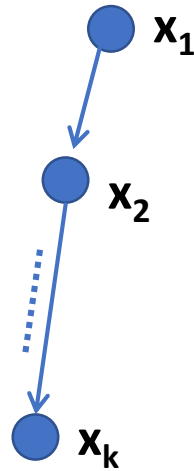
- Backtracking = parcurgerea **limitată** (de condițiile de continuare) **în adâncime** a unui arbore





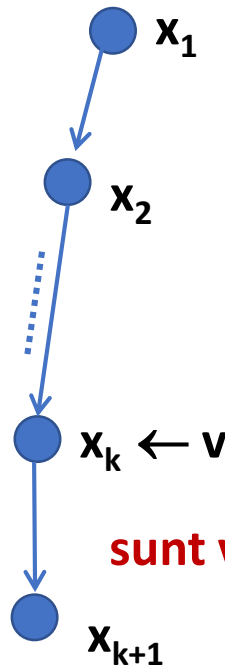
Metoda Backtracking

- Cazuri posibile la alegerea lui x_k :



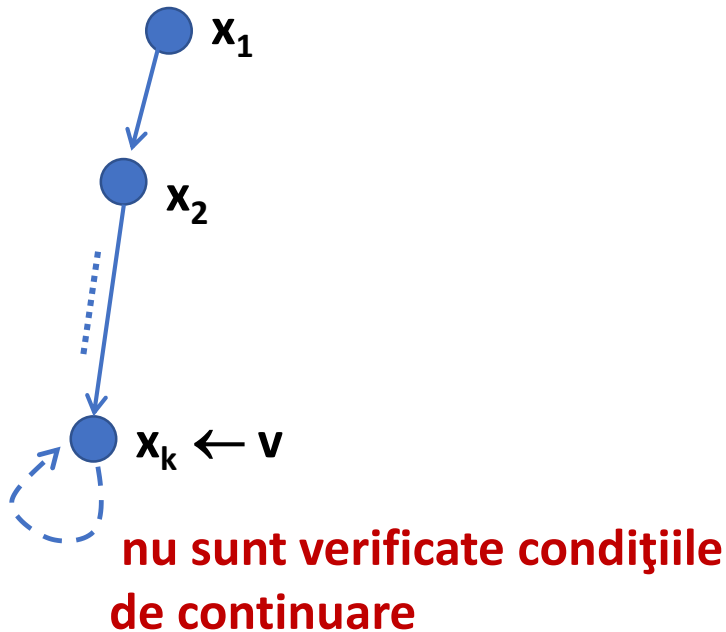
Metoda Backtracking

- Atribuie o valoare $v \in X_k$ lui x_k și avansează (sunt verificate condițiile de continuare)



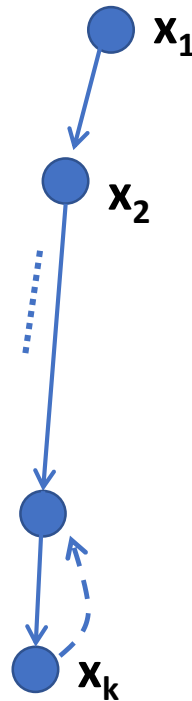
Metoda Backtracking

- ❑ Încercare eșuată (atribuie o valoare $v \in X_k$ lui x_k pentru care nu sunt verificate condițiile de continuare)



Metoda Backtracking

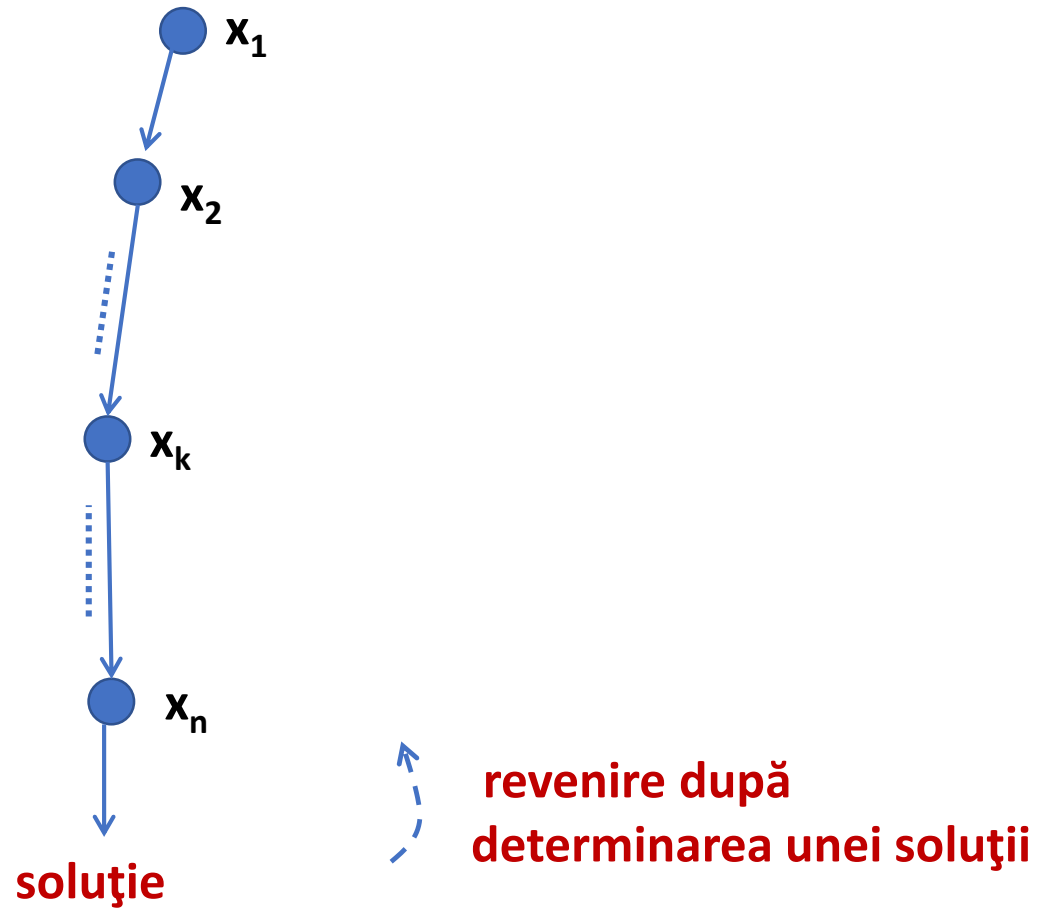
□ Revenire - nu mai există valori $v \in X_k$ neconsiderate



nu mai există valori pentru x_k
neconsiderate

Metoda Backtracking

□ Revenire după determinarea unei soluții



Varianta nerecursivă – pseudocod
– pentru soluții cu lungime fixă-

Cazul $X_i = \{p_i, p_i+1, \dots, u_i\}$

$x_i \leftarrow \mathbf{p_i - 1}, \quad \forall i = 1, \dots, n$

$k \leftarrow 1;$

$x_i \leftarrow p_i - 1, \quad \forall i=1, \dots, n$

$k \leftarrow 1;$

while $k > 0$

 if $k = n+1$

 retsol(x); $k \leftarrow k-1$; {**revenire după o sol.**}

 else

```

 $x_i \leftarrow p_i - 1, \quad \forall i=1, \dots, n$ 
 $k \leftarrow 1;$ 
while  $k > 0$ 
    if  $k = n+1$ 
        retsol(x);  $k \leftarrow k-1$ ; {revenire după o sol.}
    else
        if  $x_k < u_k$  {mai sunt valori în  $X_k$ }
             $x_k \leftarrow x_k + 1;$ 

```

```

 $x_i \leftarrow p_i - 1, \quad \forall i=1, \dots, n$ 
 $k \leftarrow 1;$ 
while  $k > 0$ 
    if  $k = n+1$ 
        retsol(x);  $k \leftarrow k-1$ ; { revenire după o sol. }
    else
        if  $\mathbf{x}_k < \mathbf{u}_k$  { mai sunt valori în  $X_k$  }
             $x_k \leftarrow x_k + 1;$ 
            if cont( $x_1, \dots, x_k$ )
                 $k \leftarrow k+1$ ; { atribuie și avansează }
            else { încercare eșuată }

```

```

 $x_i \leftarrow p_i - 1, \quad \forall i=1, \dots, n$ 
 $k \leftarrow 1;$ 
while  $k > 0$ 
    if  $k = n + 1$ 
         $\text{retsol}(x); k \leftarrow k - 1; \{ \text{revenire după o sol.} \}$ 
    else
        if  $x_k < u_k \{ \text{mai sunt valori în } X_k \}$ 
             $x_k \leftarrow x_k + 1;$ 
            if  $\text{cont}(x_1, \dots, x_k)$ 
                 $k \leftarrow k + 1; \{ \text{atribuie și avansează} \}$ 
            else
                 $\{ \text{încercare eșuată} \}$ 
        else  $x_k \leftarrow p_k - 1; k \leftarrow k - 1; \{ \text{revenire} \}$ 

```

Cazul general

- C_k = mulțimea valorilor consumate din X_k

$C_i \leftarrow \emptyset, \quad \forall i;$

$k \leftarrow 1;$

while $k > 0$

if **$k = n + 1$**

$\text{retsol}(x); \quad k \leftarrow k - 1; \{ \text{revenire după o soluție} \}$

else

 if $C_k \neq X_k$

alege $v \in X_k \setminus C_k; \quad C_k \leftarrow C_k \cup \{v\};$

 if $\text{cont}(x_1, \dots, x_{k-1}, v)$

$x_k \leftarrow v; \quad k \leftarrow k + 1; \{ \text{atribuie și avansează} \}$

 else $\{ \text{încercare eșuată} \}$

 else $C_k \leftarrow \emptyset; \quad k \leftarrow k - 1; \{ \text{revenire} \}$

Varianta recursivă

- $X_i = \{p_i, p_i+1, \dots, u_i\}$

- Apelul inițial este: **back (1)**

```
procedure back(k)
```

```
  if k=n+1
```

```
    retsol(x) {revenire dupa solutie}
```

```
  else
```

```
end.
```

- $X_i = \{p_i, p_i+1, \dots, u_i\}$

- Apelul inițial este: **back (1)**

```
procedure back(k)
```

```
  if k=n+1
```

```
    retsol(x) {revenire dupa solutie}
```

```
  else
```

```
    for (i=pk; i<=uk; i++) {valori posibile}
```

```
      xk ← i; {atribuie}
```

```
end.
```

- $X_i = \{p_i, p_i+1, \dots, u_i\}$

- Apelul inițial este: **back (1)**

```
procedure back(k)
```

```
  if k=n+1
```

```
    retsol(x) {revenire dupa solutie}
```

```
  else
```

```
    for (i=pk; i<=uk; i++) {valori posibile}
```

```
       $x_k \leftarrow i$ ; {atribuie}
```

```
      if cont( $x_1, \dots, x_k$ )
```

```
        back(k+1); {avanseaza}
```

```
        {revenire din recursivitate}
```

```
end.
```

Exemple

Exemple – de știut

- **Permutări, combinări, aranjamente**
- **Colorarea hărților**
- **Problema ciclului hamiltonian**

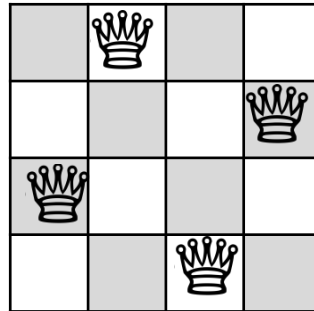
Pentru a testa condițiile de continuare $\varphi_k(x_1, \dots, x_k)$
vom folosi funcția `cont(k)`

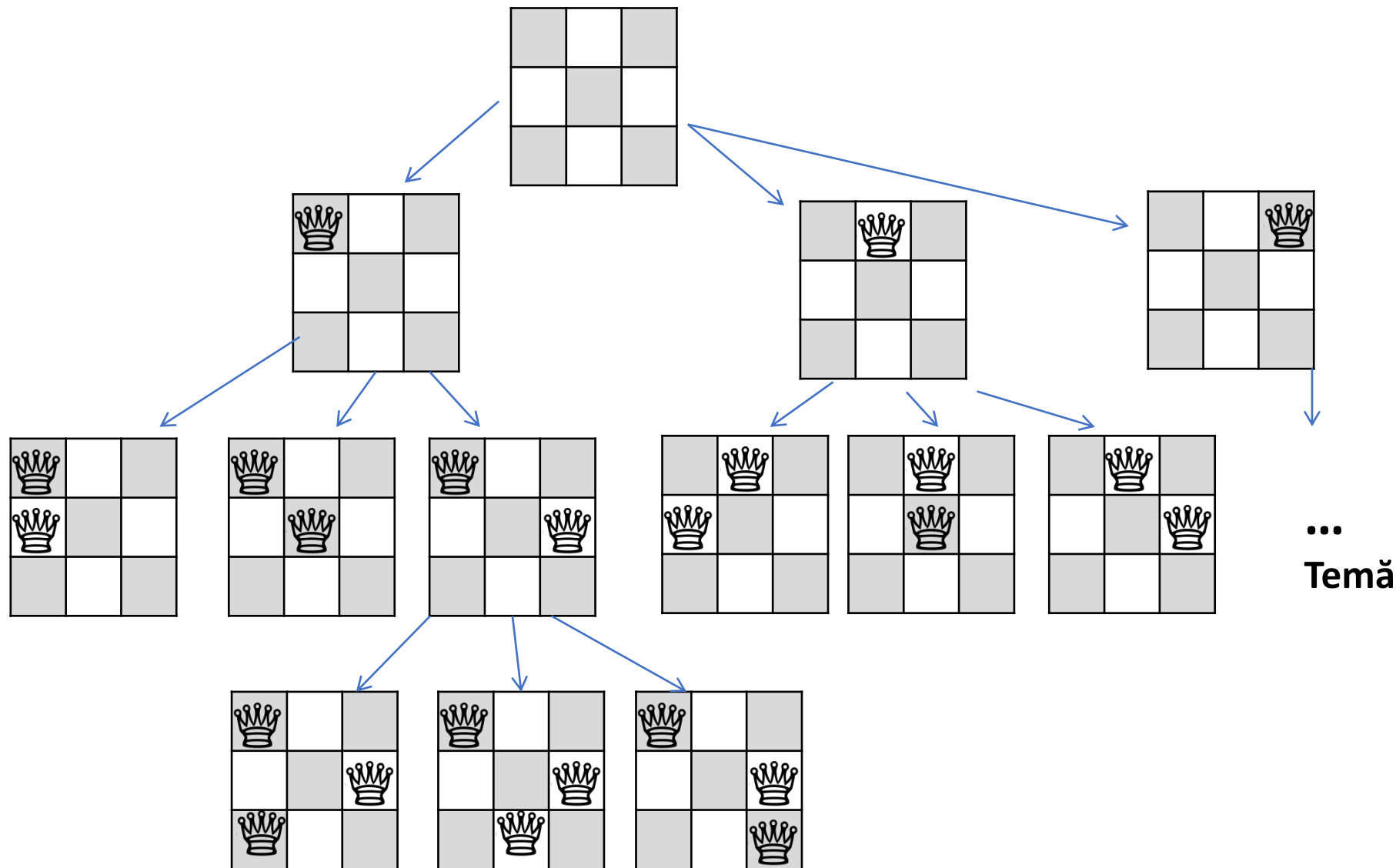
Problema celor n dame

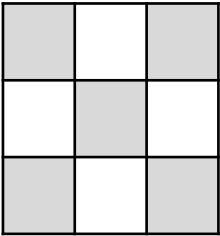


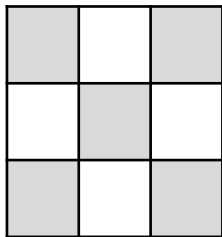
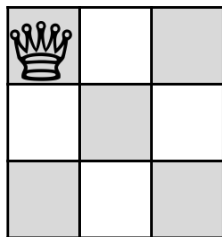
• Se consideră un caroiăj $n \times n$.

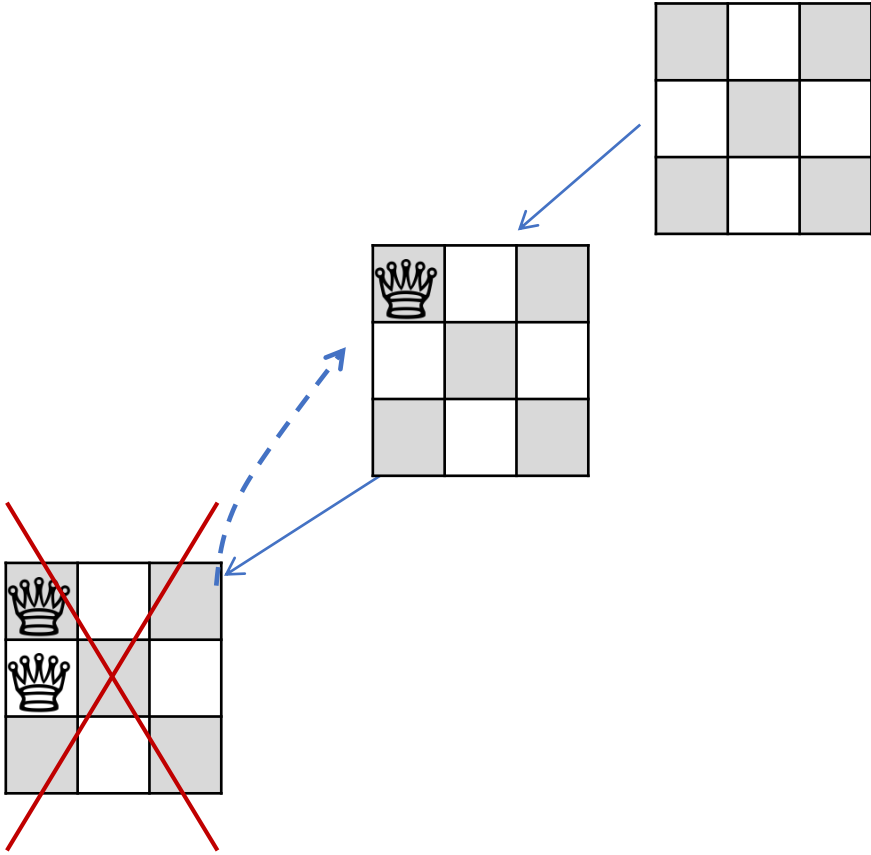
Prin analogie cu o tablă de șah ($n=8$), se dorește plasarea a n dame pe pătrățelele caroiăjului, astfel încât să nu existe două dame una în bătaia celeilalte

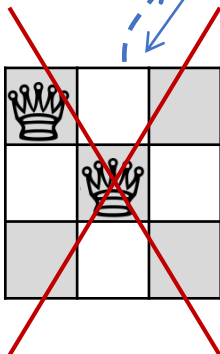
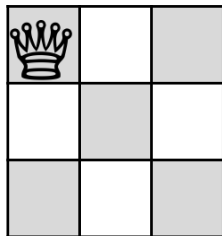
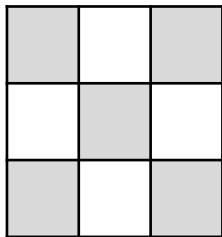


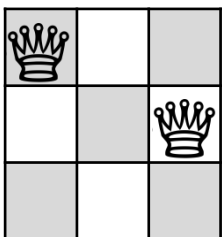
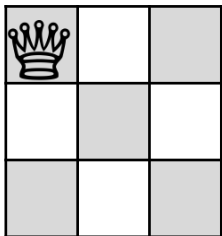
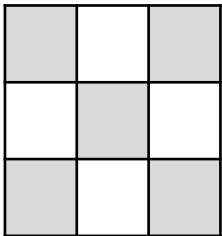


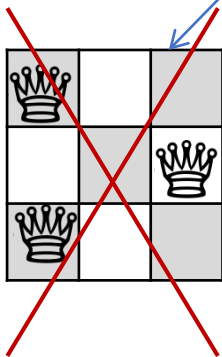
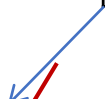
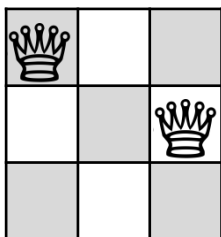
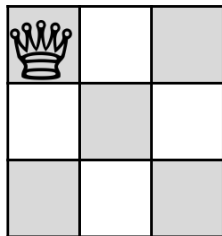
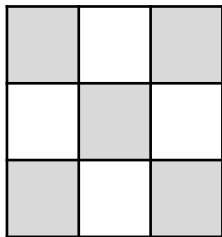


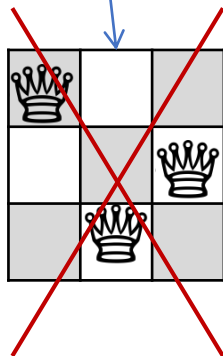
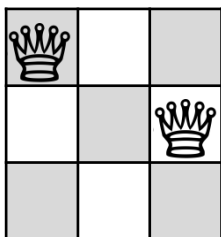
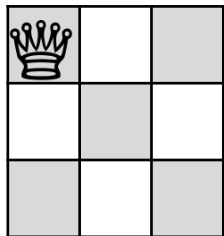
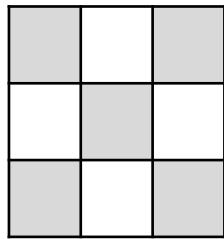


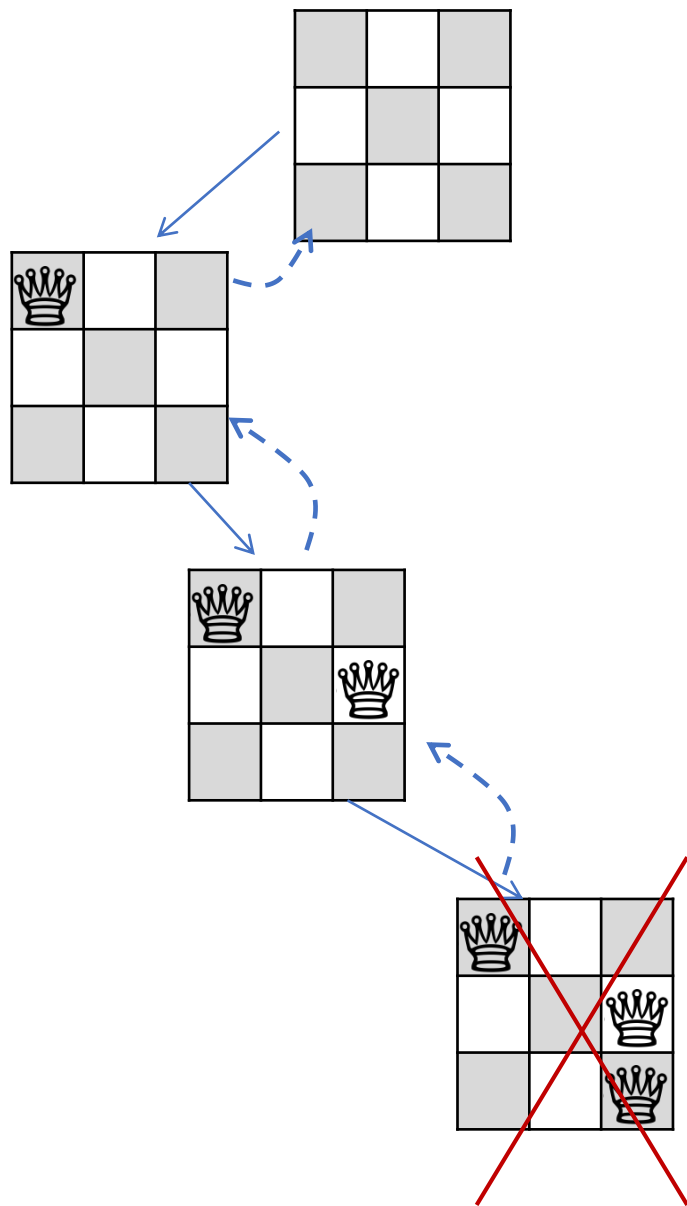


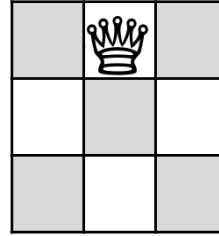
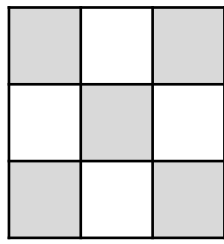


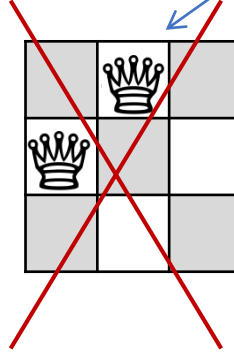
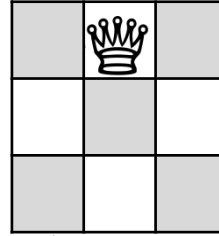
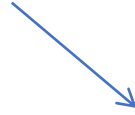
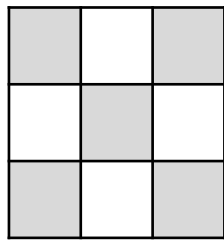


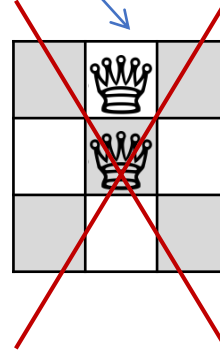
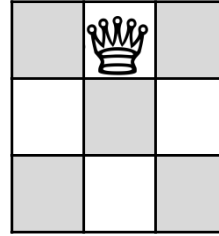
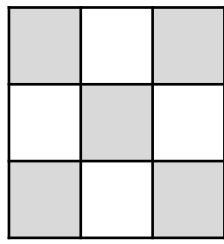


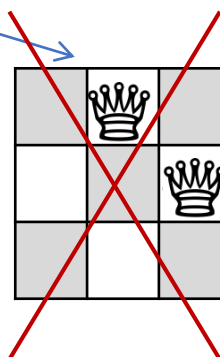
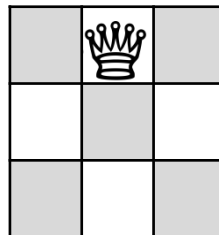
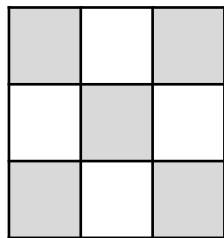


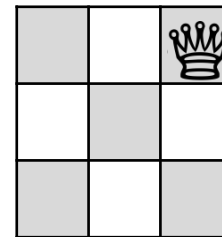
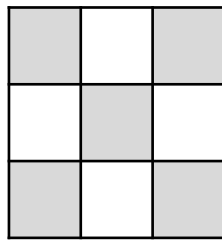












...

Temă

Problema celor n dame

- **Reprezentarea soluției**
- **Condiții interne (finale)**
- **Condiții de continuare**

Problema celor n dame

- **Reprezentarea soluției**

$\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, unde

\mathbf{x}_k = coloana pe care este plasată dama
de pe linia k

$$\mathbf{x}_k \in \{1, 2, \dots, n\} \quad (\mathbf{p}_k = 1, \mathbf{u}_k = n)$$

- **Condiții interne (finale)**

- **Condiții de continuare**

Problema celor n dame

- **Reprezentarea soluției**

$\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, unde

\mathbf{x}_k = coloana pe care este plasată dama
de pe linia k

$$\mathbf{x}_k \in \{1, 2, \dots, n\} \quad (p_k = 1, u_k = n)$$

- **Condiții interne (finale)**

pentru orice $i \neq j$: $\mathbf{x}_i \neq \mathbf{x}_j$ și $|\mathbf{x}_i - \mathbf{x}_j| \neq |j - i|$

- **Condiții de continuare**

Problema celor n dame

- **Reprezentarea soluției**

$\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, unde

\mathbf{x}_k = coloana pe care este plasată dama
de pe linia k

$$\mathbf{x}_k \in \{1, 2, \dots, n\} \quad (p_k = 1, u_k = n)$$

- **Condiții interne (finale)**

pentru orice $i \neq j$: $\mathbf{x}_i \neq \mathbf{x}_j$ și $|\mathbf{x}_i - \mathbf{x}_j| \neq |j - i|$

- **Condiții de continuare** – pentru x_k

pentru orice $i < k$: $\mathbf{x}_i \neq \mathbf{x}_k$ și $|\mathbf{x}_i - \mathbf{x}_k| \neq k - i$

Implementare - varianta recursivă

```
boolean cont(int k) {  
    for(int i=1; i<k; i++)  
        if((x[i]==x[k]) || (Math.abs(x[k]-x[i])==k-i))  
            return false;  
    return true;  
}
```

Implementare - varianta recursivă

```
boolean cont(int k){
    for(int i=1;i<k;i++)
        if((x[i]==x[k]) || (Math.abs(x[k]-x[i])==k-i))
            return false;
    return true;
}

void retsol(int[] x){
    for(int i=1;i<=n;i++)
        System.out.print("(" + i + ", " + x[i] + ") ");
    System.out.println();
}
```

Implementare - varianta recursivă

```
void backrec(int k) {  
    if (k==n+1)  
        retsol(x);  
    else  
        for (int i=1; i<=n ; i++) { // xk  
            x[k]=i;  
            if (cont(k))  
                backrec(k+1);  
        }  
}
```

Implementare - varianta nerecursivă

```
void back() {
    int k=1;
    x=new int[n+1];
    for(int i=1;i<=n;i++) x[i]=0;
    while(k>0) {
        if(k==n+1) { retsol(x); k--; } // revenire dupa sol
        else{
            if(x[k]<n) {

            }
            else{
                } // revenire
            }
        }
    }
}
```

Implementare - varianta nerecursivă

```
void back() {  
    int k=1;  
    x=new int[n+1];  
    for(int i=1;i<=n;i++) x[i]=0;  
    while(k>0) {  
        if(k==n+1) { retsol(x); k--; } // revenire dupa sol  
        else {  
            if(x[k]<n) {  
                x[k]++; // atribuie  
                if (cont(k)) k++; // si avanseaza  
            }  
            else { x[k]=0; k--; } // revenire  
        }  
    }  
}
```

Șiruri corecte de paranteze

- Să se genereze toate șirurile de n paranteze ce se închid corect (n par)

Șiruri corecte de paranteze

- Să se genereze toate șirurile de n paranteze ce se închid corect (n par)
- Numărul de șiruri corecte = $C_{n/2}$
(numerele lui Catalan – v. PD)

Șiruri corecte de paranteze

- **Reprezentarea soluției**
- **Condiții interne (finale)**
- **Condiții de continuare**

Șiruri corecte de paranteze

- **Reprezentarea soluției**

$$\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}, \text{ unde}$$
$$\mathbf{x}_k \in \{ ' (' , ') ' \}$$

- **Condiții interne (finale)**

- **Condiții de continuare**

Șiruri corecte de paranteze

- **Reprezentarea soluției**

$$\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}, \text{ unde}$$
$$\mathbf{x}_k \in \{ ' (' , ') ' \}$$

- **Condiții interne (finale)**

Notăm $\mathbf{dif} = \mathbf{nr}_\text{(')} - \mathbf{nr}_\text{('}$

$$\mathbf{dif} = 0$$

$\mathbf{dif} \geq 0$ pentru orice secvență $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k\}$

- **Condiții de continuare**

Șiruri corecte de paranteze

- **Reprezentarea soluției**

$$\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}, \text{ unde}$$
$$\mathbf{x}_k \in \{ ' (' , ') ' \}$$

- **Condiții interne (finale)**

$$\text{Notăm } \mathbf{dif} = \mathbf{nr}_\text{(')} - \mathbf{nr}_\text{(')}$$

$$\mathbf{dif} = 0$$

$$\mathbf{dif} \geq 0 \text{ pentru orice secvență } \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k\}$$

- **Condiții de continuare**

$$\mathbf{dif} \geq 0 \quad \rightarrow \text{doar necesar}$$

Șiruri corecte de paranteze

- **Reprezentarea soluției**

$$\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}, \text{ unde}$$
$$\mathbf{x}_k \in \{ ' (' , ') ' \}$$

- **Condiții interne (finale)**

Notăm $\mathbf{dif} = \mathbf{nr}_\text{(')} - \mathbf{nr}_\text{(')}$

$$\mathbf{dif} = 0$$

$\mathbf{dif} \geq 0$ pentru orice secvență $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k\}$

- **Condiții de continuare**

$\mathbf{dif} \geq 0 \quad \rightarrow$ doar necesar

$\mathbf{dif} \leq \mathbf{n-k} \quad \rightarrow$ și suficient

```
void back() {  
    dif=0;  
    back(1);  
}  
void back(int k) {  
    if(k==n+1)  
        retsol(x);  
    else{  
  
        }  
}
```

```

void back() {
    dif=0;
    back(1);
}
void back(int k) {
    if(k==n+1)
        retsol(x);
    else{
        x[k]='(';
        dif++;
        if (dif <= n-k)
            back(k+1);
        dif--;

    }
}

```



```

void back() {
    dif=0;
    back(1);
}
void back(int k) {
    if(k==n+1)
        retsol(x);
    else{
        x[k]='(';
        dif++;
        if (dif <= n-k)
            back(k+1);
        dif--;

        x[k]=')';
        dif--;
        if (dif >= 0)
            back(k+1);
        dif++;
    }
}

```

Șiruri corecte de paranteze

- **Implementare nerecursivă** -temă

Metoda Backtracking

- Fie vectorul $a=(a_1,\dots,a_n)$. Să se determine toate subșirurile crescătoare de lungime maximă.

Metoda Backtracking

- Fie vectorul $a=(a_1,\dots,a_n)$. Să se determine toate subșirurile crescătoare de lungime maximă.

- **Subproblemă:**

$\text{lung}[i]$ = lungimea maximă a unui subșir crescător ce începe pe poziția i

- **Soluție problemă:**

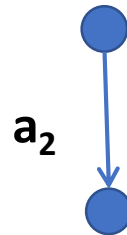
$$\text{lmax} = \max\{\text{lung}[i] \mid i = 1, 2, \dots, n\}$$

Subșirurile crescătoare maxime

a: 8 1 7 4 6 5 11

1 2 3 4 5 6 7

lung : 2 4 2 3 2 2 1

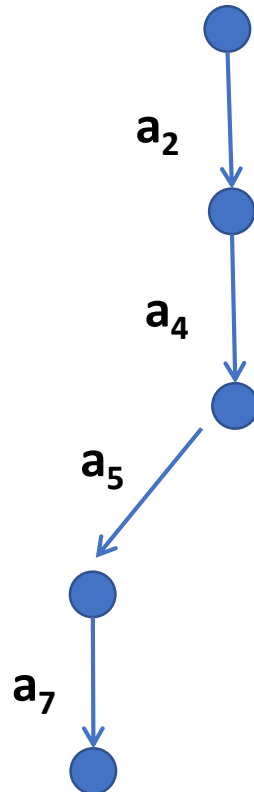


Subșirurile crescătoare maxime

a: 8 1 7 4 6 5 11

1 2 3 4 5 6 7

lung : 2 4 2 3 2 2 1

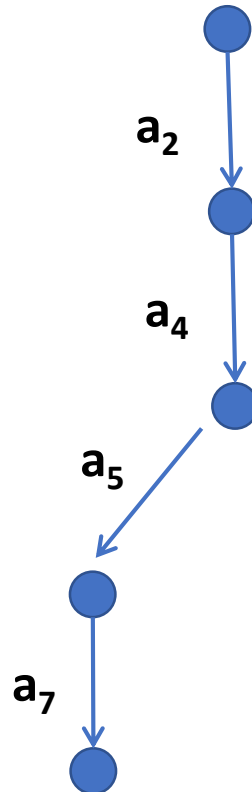


Subșirurile crescătoare maxime

a: 8 1 7 4 6 5 11

1 2 3 4 5 6 7

lung : 2 4 2 3 2 2 1

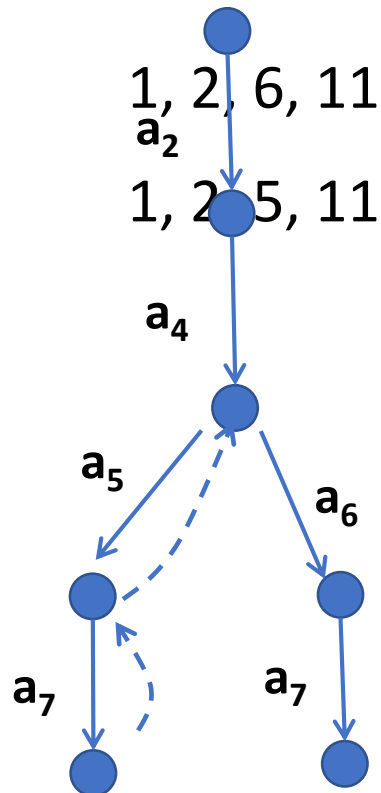


Metoda Backtracking

a: 8 1 7 4 6 5 11

1 2 3 4 5 6 7

lung : 2 4 2 3 2 2 1



Metoda Backtracking

- **Reprezentarea soluției**
- **Condiții interne (finale)**
- **Condiții de continuare**

Metoda Backtracking

- **Reprezentarea soluției**

$\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{1\max}\}$, unde
 $\mathbf{x}_k \in \{1, \dots, n\}$ poziție din vectorul a

- **Condiții interne (finale)**

$$a_{x_1} < a_{x_2} < \dots < a_{x_{1\max}}$$

- **Condiții de continuare**

Metoda Backtracking

- **Reprezentarea soluției**

$\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{l_{\max}}\}$, unde
 $\mathbf{x}_k \in \{1, \dots, n\}$ poziție din vectorul \mathbf{a}

- **Condiții interne (finale)**

$$a_{x_1} < a_{x_2} < \dots < a_{x_{l_{\max}}}$$

- **Condiții de continuare**

$$\text{lung}[\mathbf{a}[\mathbf{x}_1]] = l_{\max}$$

$$\mathbf{x}_{k-1} < \mathbf{x}_k, \quad a_{x_{k-1}} < a_{x_k}, \quad \text{lung}[\mathbf{x}_k] = \text{lung}[\mathbf{x}_{k-1}] - 1$$

```

void scribe(int k){
    if (k==lmax+1){    //retsol();
        for (int i=1;i<=lmax;i++)
            System.out.print(a[x[i]]+" ");
        System.out.println();
    }
    else
        for (int j=x[k-1]+1;j<=n;j++){
            x[k]=j;

        }
}

```

```

for(int i=1;i<=n;i++)
    if (lung[i]==lmax) {
        x[1]=i;
        scribe(2);
    }

```

```

void scribe(int k){
    if (k==lmax+1){    //retsol();
        for (int i=1;i<=lmax;i++)
            System.out.print(a[x[i]]+" ");
        System.out.println();
    }
    else
        for (int j=x[k-1]+1;j<=n;j++){
            x[k]=j;
            if ((a[x[k-1]]<a[x[k]]) &&
                (lung[x[k-1]]==1+lung[x[k]]))
                scribe(k+1);
        }
}

```

```

for(int i=1;i<=n;i++)
    if (lung[i]==lmax) {
        x[1]=i;
        scribe(2);
    }

```

Metoda Backtracking - SAT



• Problema satisfiabilității SAT

Considerăm o expresie logică în care apar variabilele $\{x_1, x_2, \dots, x_n\}$ și negațiile lor \bar{x}_i . Știind că expresia este de forma

$$E = C_1 \wedge C_2 \wedge \dots \wedge C_m$$

unde C_i sunt clauze disjunctive = în care apare doar operatorul \vee , să se verifice dacă se pot atribui valori variabilelor astfel încât valoarea expresiei să fie true (expresia să fie satisfăcută)

Metoda Backtracking - SAT

- **Problema satisfiabilității SAT**

Considerăm o expresie logică în care apar variabilele $\{x_1, x_2, \dots, x_n\}$ și negațiile lor \bar{x}_i . Știind că expresia este de forma

$$E = C_1 \wedge C_2 \wedge \dots \wedge C_m$$

unde C_i sunt clauze disjunctive = în care apare doar operatorul \vee , să se verifice dacă se pot atribui valori variabilelor astfel încât valoarea expresiei să fie true (expresia să fie satisfăcută)

Literal = x_i sau \bar{x}_i

FNC – formă normală conjunctivă

Metoda Backtracking - SAT

- **Problema satisfiabilității SAT**

Considerăm o expresie logică în care apar variabilele $\{x_1, x_2, \dots, x_n\}$ și negațiile lor \bar{x}_i . Știind că expresia este de forma

$$E = C_1 \wedge C_2 \wedge \dots \wedge C_m$$

unde C_i sunt clauze disjunctive = în care apare doar operatorul \vee , să se verifice dacă se pot atribui valori variabilelor astfel încât valoarea expresiei să fie true (expresia să fie satisfăcută)

- **k-SAT** – fiecare clauză are cel mult k literali

Metoda Backtracking - SAT

- **Problema satisfiabilității SAT**

$$E = (x_1 \vee \bar{x}_2) \wedge (x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (\bar{x}_1 \vee x_3) \wedge (\bar{x}_4 \vee \bar{x}_3)$$

adevărată pentru $x_1 = x_2 = x_3 = 1, x_4 = 0$

- k-SAT = fiecare clauză are cel mult k literali
- k = 2 - polinomial
- k = 3 – NP-completă

Metoda Backtracking - SAT

Idei de rezolvare?

Metoda Backtracking - SAT

- **Generăm toate șirurile binare $x_1x_2...x_n$ (0 = false, 1 = true) și verificăm pentru fiecare astfel de șir înlocuind în expresia E dacă E devine adevărată**
- Dacă găsim un șir pentru care expresia este adevărată oprim generarea

Metoda Backtracking - SAT

- **Generăm toate șirurile binare $x_1x_2...x_n$ (0 = false, 1 = true) și verificăm pentru fiecare astfel de șir înlocuind în expresia E dacă E devine adevărată**
- Dacă găsim un șir pentru care expresia este adevărată oprim generarea
- **Complexitate?**

Metoda Backtracking - SAT

- **Generăm toate șirurile binare $x_1x_2...x_n$ (0 = false, 1 = true) și verificăm pentru fiecare astfel de șir înlocuind în expresia E dacă E devine adevărată**
- Dacă găsim un șir pentru care expresia este adevărată oprim generarea
 - Dacă expresia nu poate fi satisfăcută, atunci se vor genera și testa toate șirurile binare de lungime n

$O(2^nm)$

Metoda Backtracking - SAT

- **Putem face verificări pe parcurs \Rightarrow condiții de continuare?**
- **Contează pentru performanță ordinea în care dăm valori variabilelor? \Rightarrow euristici**

Metoda Backtracking - SAT

- **Condiții de continuare**

Înlocuim valoarea v dată variabilei x_k la pasul k în fiecare clauză C_i din E în care apare x_k sau \bar{x}_k .

În clauză literalul corespunzător lui x_k este în una din situațiile:

- devine 1 \Rightarrow

- devine 0 \Rightarrow

Metoda Backtracking - SAT

- **Condiții de continuare**

Înlocuim valoarea v dată variabilei x_k la pasul k în fiecare clauză C_i din E în care apare x_k sau \bar{x}_k .

În clauză literalul corespunzător lui x_k este în una din situațiile:

- devine 1 \Rightarrow pot elimina clauza C_i din E
- devine 0 \Rightarrow pot elimina 0 (literalul corespunzător lui x_k) din C_i

Metoda Backtracking - SAT

- **Condiții de continuare**

Înlocuim valoarea v dată variabilei x_k la pasul k în fiecare clauză C_i din E în care apare x_k sau \bar{x}_k .

În clauză literalul corespunzător lui x_k este în una din situațiile:

- devine 1 \Rightarrow pot elimina clauza C_i din E
- devine 0 \Rightarrow pot elimina 0 (literalul corespunzător lui x_k) din C_i

Când nu mai are sens să continuăm (pentru că expresia nu poate fi satisfăcută cu valorile date deja variabilelor)?

Metoda Backtracking - SAT

- **Condiții de continuare**

Înlocuim valoarea v dată variabilei x_k la pasul k în fiecare clauză C_i din E în care apare x_k sau \bar{x}_k .

În clauză literalul corespunzător lui x_k este în una din situațiile:

- devine 1 \Rightarrow pot elimina clauza C_i din E
- devine 0 \Rightarrow pot elimina 0 (literalul corespunzător lui x_k) din C_i

**Dacă o clauză devine vidă, dar expresia nu este vidă,
atunci încercarea de a da valoarea v lui x_k este eșuată**

Metoda Backtracking - SAT

- **Condiții de continuare**

Înlocuim valoarea v dată variabilei x_k la pasul k în fiecare clauză C_i din E în care apare x_k sau \bar{x}_k .

În clauză literalul corespunzător lui x_k este în una din situațiile:

- devine 1 \Rightarrow pot elimina clauza C_i din E
- devine 0 \Rightarrow pot elimina 0 (literalul corespunzător lui x_k) din C_i

Dacă expresia E devine vidă atunci am găsit o soluție (restul variabilelor pot primi orice valoare)

Metoda Backtracking - SAT

- **Temă – backtracking pentru 3-SAT**

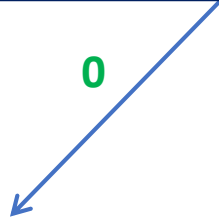
(n variabile, m clauze de lungime cel mult 3 = cu cel mult 3 literali)

Metoda Backtracking - SAT

$$(x_2 \vee \bar{x}_3) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2)$$

x_1 :

0



Metoda Backtracking - SAT

$$(x_2 \vee \bar{x}_3) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2)$$

x_1 :

0

$$(x_2 \vee \bar{x}_3) \wedge (x_2 \vee x_3) \wedge (\bar{x}_2)$$

x_2 :

0

Metoda Backtracking - SAT

$$(x_2 \vee \bar{x}_3) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2)$$

x_1 :

0

$$(x_2 \vee \bar{x}_3) \wedge (x_2 \vee x_3) \wedge (\bar{x}_2)$$

x_2 :

0

$$(\bar{x}_3) \wedge (x_3)$$

Metoda Backtracking - SAT

$$(x_2 \vee \bar{x}_3) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2)$$

x_1 :

0

$$(x_2 \vee \bar{x}_3) \wedge (x_2 \vee x_3) \wedge (\bar{x}_2)$$

x_2 :

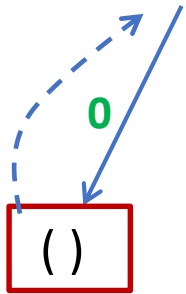
0

$$(\bar{x}_3) \wedge (x_3)$$

x_3 :

0

()



Metoda Backtracking - SAT

$$(x_2 \vee \bar{x}_3) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2)$$

x_1 :

0

$$(x_2 \vee \bar{x}_3) \wedge (x_2 \vee x_3) \wedge (\bar{x}_2)$$

x_2 :

0

$$(\bar{x}_3) \wedge (x_3)$$

x_3 :

0

1

()

()

Metoda Backtracking - SAT

$$(x_2 \vee \bar{x}_3) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2)$$

x_1 :

0

$$(x_2 \vee \bar{x}_3) \wedge (x_2 \vee x_3) \wedge (\bar{x}_2)$$

x_2 :

0

1

$$(\bar{x}_3) \wedge (x_3)$$

x_3 :

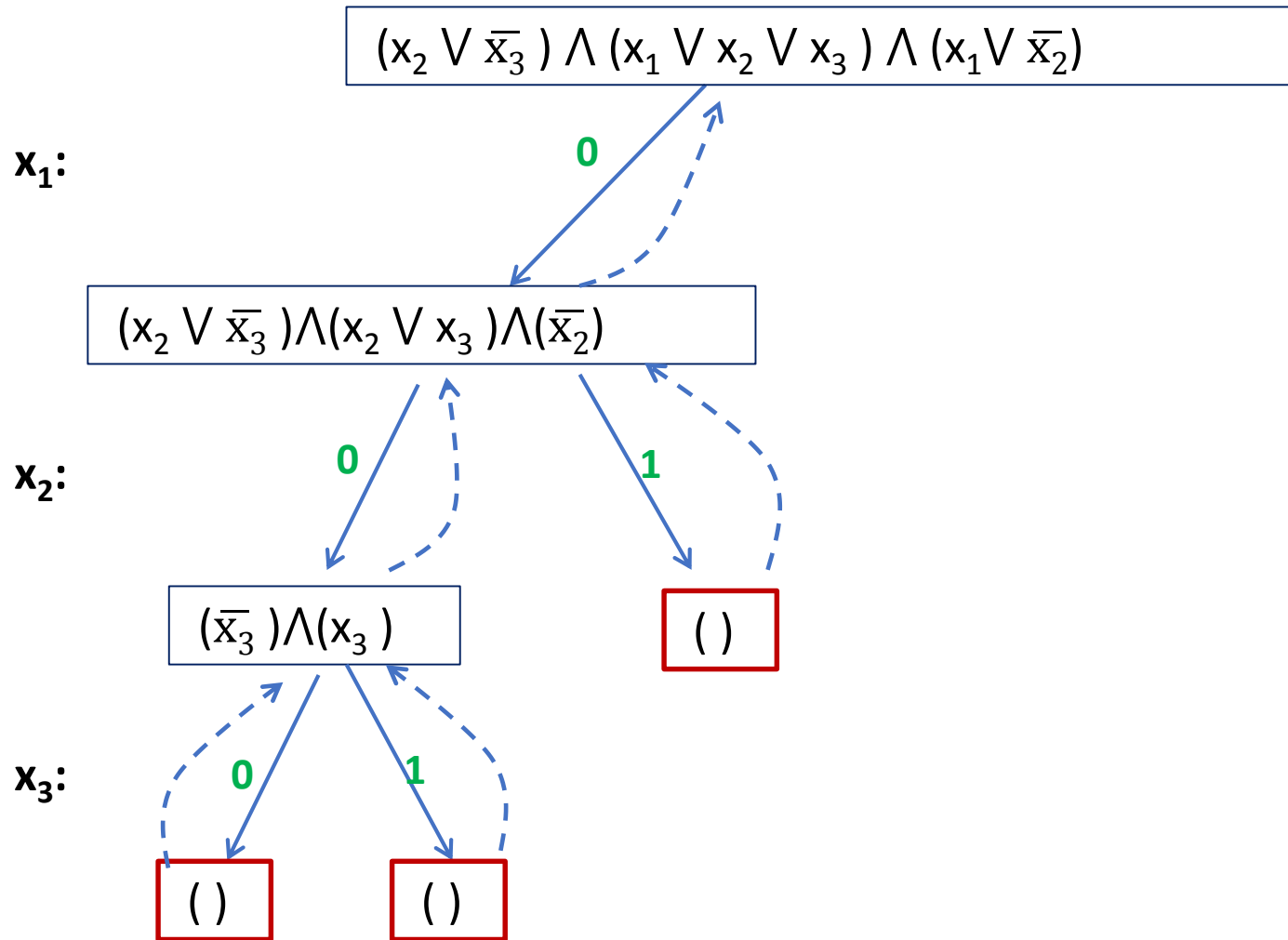
0

1

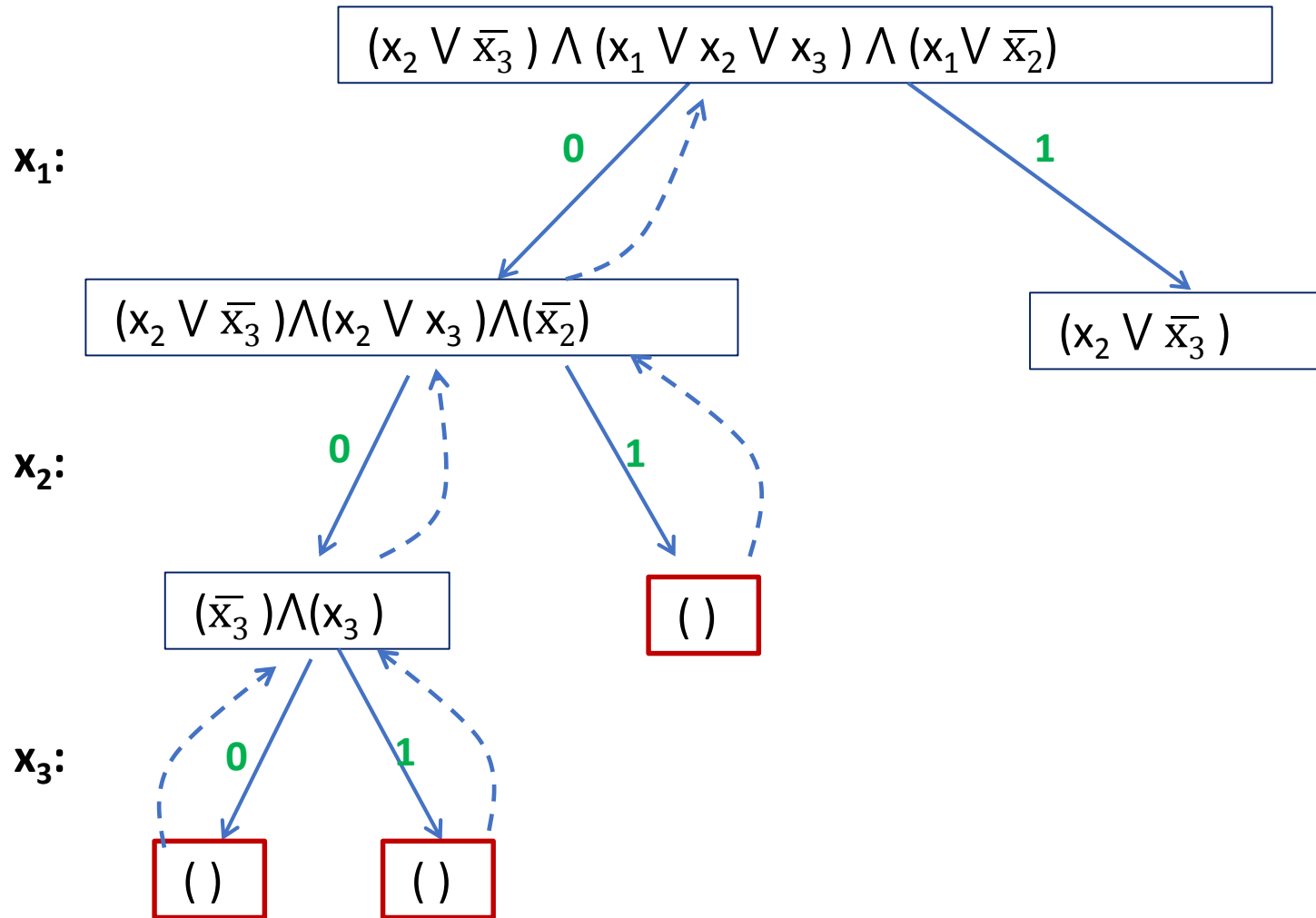
()

()

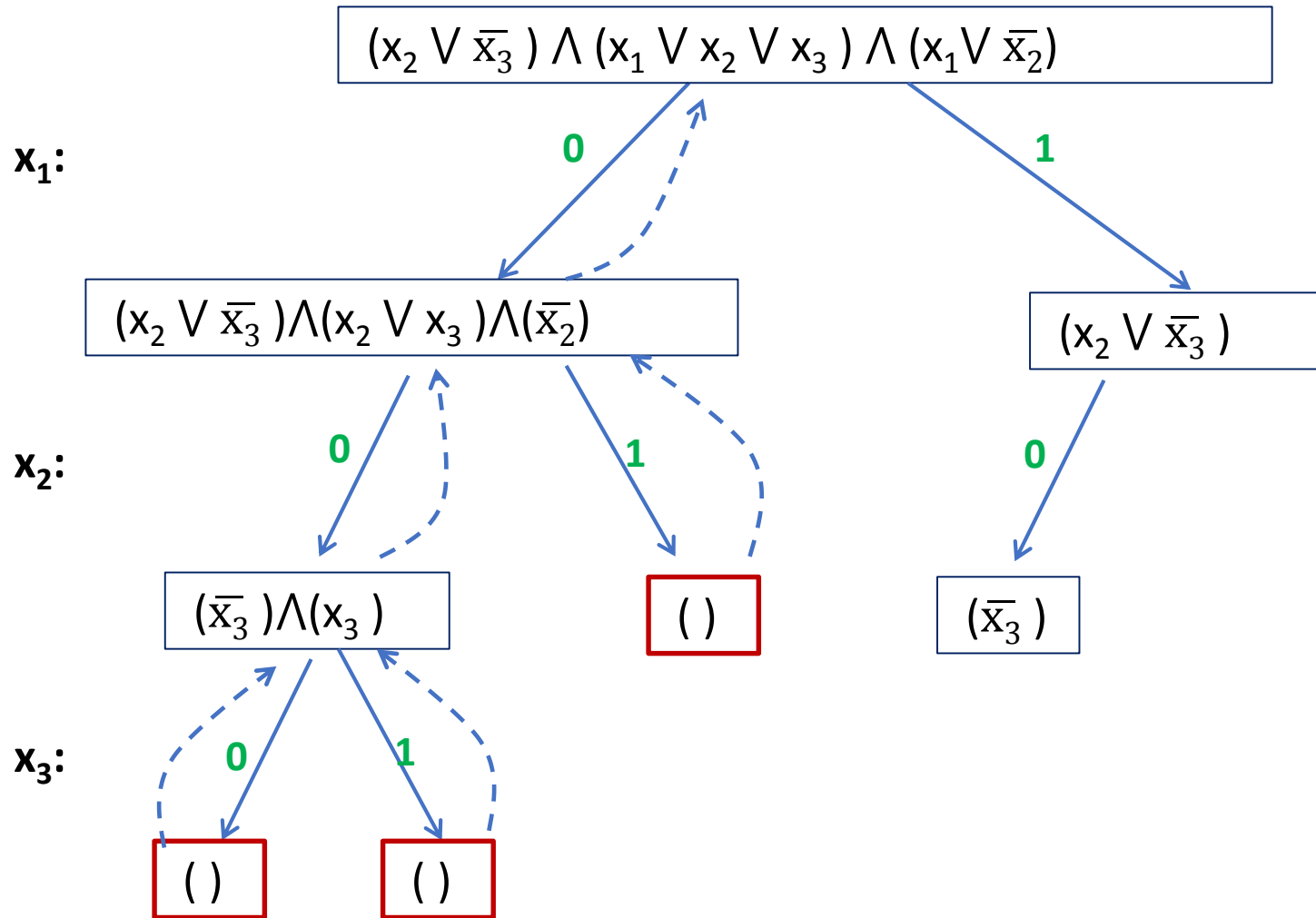
Metoda Backtracking - SAT



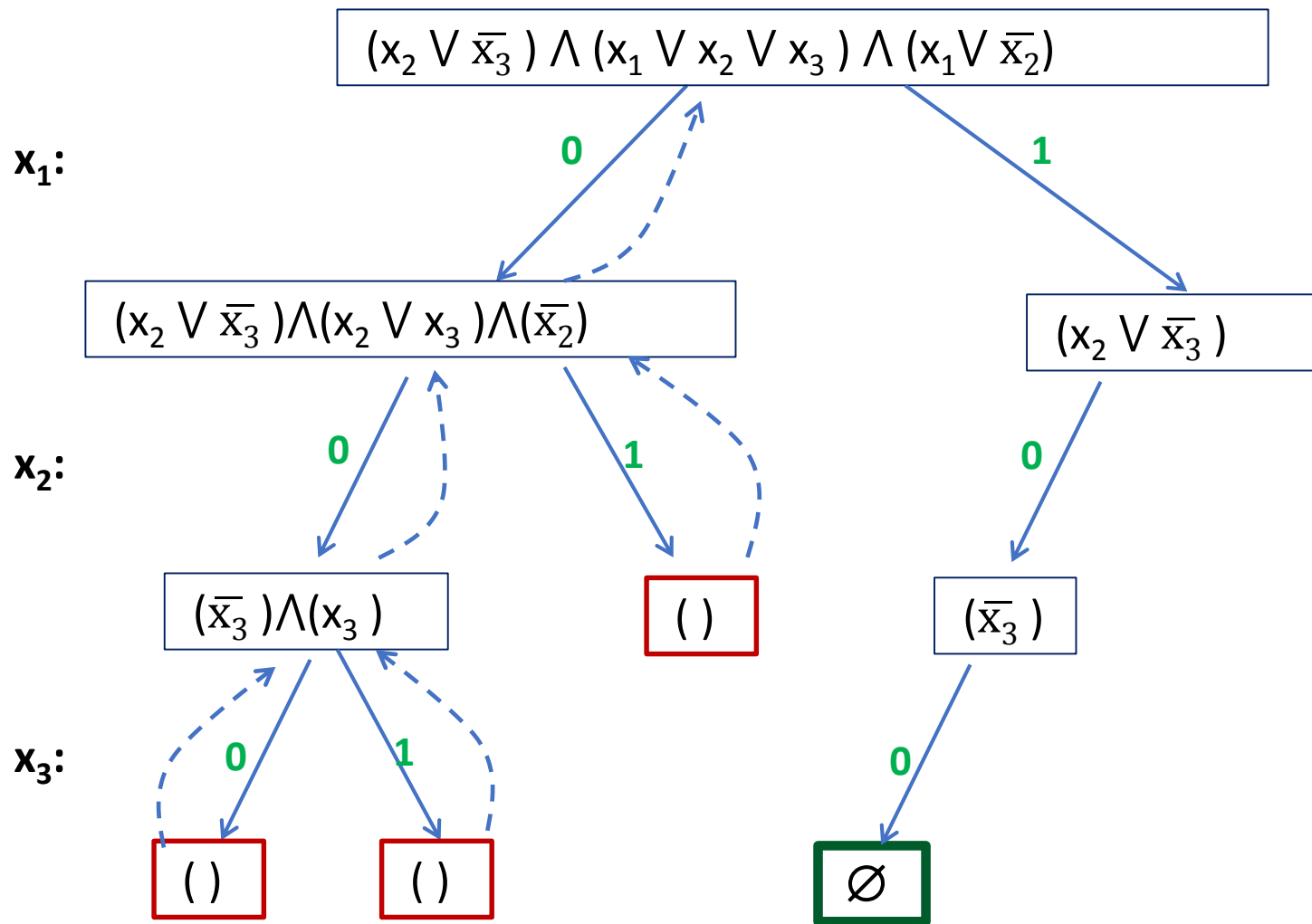
Metoda Backtracking - SAT



Metoda Backtracking - SAT



Metoda Backtracking - SAT



$E = \emptyset \Rightarrow \text{Soluție} + \text{STOP}$

```
procedure back(k, E, V)
  if E =  $\emptyset$ 
    scrie true, x STOP
   $x_k \leftarrow$  false;
  for (C clauza în E care contine  $x_k$ )
    daca  $\bar{x}_k \in C$  atunci
      elimina(E, C)
      back(k+1, E)
      adauga(E, C)
```

```
back(1)
```

```
scrie false
```

```

procedure back(k, E, V)
    if E =  $\emptyset$ 
        scrie true, x STOP
     $x_k \leftarrow$  false;
    for (C clauza în E care contine  $x_k$ )
        daca  $\bar{x}_k \in C$  atunci
            elimina(E, C)
            back(k+1, E)
            adauga(E, C)
        altfel
            reducem(C, k) - eliminam  $x_k$  din C
            if C  $\neq \emptyset$ 
                back(k+1, E)
            restauram(C, k) - reintroducem  $x_k$  in C

```

back(1)

scrie false


```

procedure back(k, E, V)
    if E =  $\emptyset$ 
        scrie true, x STOP
     $x_k \leftarrow$  false;
    for (C clauza în E care contine  $x_k$ )
        daca  $\bar{x}_k \in C$  atunci
            elimina(E, C)
            back(k+1, E)
            adauga(E, C)
        altfel
            reducem(C, k) - eliminam  $x_k$  din C
            if C  $\neq \emptyset$ 
                back(k+1, E)
            restauram(C, k) - reintroducem  $x_k$  in C
     $x_k \leftarrow$  true;
... SIMILAR

```

back(1)

scrie false

Metoda Backtracking - SAT

- **Ordinea în care se dau valori variabilelor**

⇒ euristici

Exemplu

– întâi dam valori **variabilelor care apar în clauze scurte**

- Greedy: literalul care satisface mai multe clauze

- **Detectarea de conflicte + formule de logică**

```

procedure back(k, E, V)  V-multimea variabilelor neselectate
    if E =  $\emptyset$ 
        scrie true, x STOP
    t  $\leftarrow$  alegeVariabila(V)
     $x_t \leftarrow$  false;
    for (C clauza în E care contine  $x_t$ )
        daca  $\bar{x}_t \in C$  atunci
            elimina(E, C)
            back(k+1, E)
            adauga(E, C)
        altfel
            reducem(C, k) - eliminam  $x_t$  din C
            if C  $\neq \emptyset$ 
                back(k+1, E)
            restauram(C, k) - reintroducem  $x_t$  in C
     $x_t \leftarrow$  true;
... SIMILAR

```

back(1, E, {1,2,...,n})

scrie false

Metoda Backtracking

- **Variantele** cele mai uzuale întâlnite în aplicarea metodei backtracking sunt următoarele:
 - soluția poate avea un număr variabil de componente *și/sau*
 - dintre ele alegem una care optimizează o funcție dată

Metoda Backtracking

- **Exemplu:** Dat un număr natural n , să se genereze toate partițiile lui n ca sumă de numere pozitive

Partiție a lui $n = \{x_1, x_2, \dots, x_k\}$ cu

$$x_1 + x_2 + \dots + x_k = n$$

$$4 = 1+1+1+1$$

$$4 = 1+1+2$$

$$4 = 1+3$$

$$4 = 2+2$$

Metoda Backtracking - Partiții

- **Reprezentarea soluției**

$$\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k\}, \text{ unde}$$
$$\mathbf{x}_i \in \{1, \dots, n\}$$

- **Condiții interne (finale)**

- **Condiții de continuare**

Metoda Backtracking - Partiții

- **Reprezentarea soluției**

$$\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k\}, \text{ unde} \\ \mathbf{x}_i \in \{1, \dots, n\}$$

- **Condiții interne (finale)**

$$\mathbf{x}_1 + \mathbf{x}_2 + \dots + \mathbf{x}_k = n$$

Pentru unicitate: $\mathbf{x}_1 \leq \mathbf{x}_2 \leq \dots \leq \mathbf{x}_k$

- **Condiții de continuare**

Metoda Backtracking - Partiții

- **Reprezentarea soluției**

$$\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k\}, \text{ unde}$$
$$\mathbf{x}_i \in \{1, \dots, n\}$$

- **Condiții interne (finale)**

$$\mathbf{x}_1 + \mathbf{x}_2 + \dots + \mathbf{x}_k = n$$

Pentru unicitate: $\mathbf{x}_1 \leq \mathbf{x}_2 \leq \dots \leq \mathbf{x}_k$

- **Condiții de continuare**

$$\mathbf{x}_{k-1} \leq \mathbf{x}_k \quad \longrightarrow \quad \mathbf{x}_k \in \{\mathbf{x}_{k-1}, \dots, n\} = X_k$$

$$\mathbf{x}_1 + \mathbf{x}_2 + \dots + \mathbf{x}_k \leq n$$

Implementare - varianta recursivă

```
void backrec(int k) {
    for(int i=x[k-1]; i<=n; i++) {
        x[k]=i;
        if(s+x[k]<=n) /// verif.cond.de cont

        // else return;
    }
}
```

```

void backrec(int k) {
    for(int i=x[k-1]; i<=n; i++) {
        x[k]=i;
        if(s+x[k]<=n) ////verif.cond.de cont
            if(s+x[k]==n) {//este solutie
                retsol(x, k);
                return;
            }

        // else return;
    }
}

```

```

void backrec(int k) {
    for(int i=x[k-1]; i<=n; i++) {
        x[k]=i;
        if (s+x[k]<=n) ////verif.cond.de cont
            if (s+x[k]==n) { //este solutie
                retsol(x, k);
                return;
            }
            else{
                s+=x[k];
                backrec(k+1);
                s-=x[k];
            }
        // else return;
    }
}

```

```
void retsol(int[] x,int k){  
    for(int i=1;i<=k;i++)  
        System.out.print(x[i]+" ");  
    System.out.println();  
}
```

```
void backrec(){  
    x=new int[n+1];  
    x[0]=1; //prima valoare pentru x[1]  
    s=0;  
    backrec(1);  
}
```

Implementare - varianta nerecursivă

```

void back() {
    int k=1, s=0; int x[]=new int[n+1];
    x[1]=0;
    while(k>=1) {
        if(x[k]<n) {
            x[k]++; s++;
            if(s<=n) { //cont - verific. conditiilor de cont

        }
    }
}

```

```

void back() {
    int k=1, s=0; int x[]=new int[n+1];
    x[1]=0;
    while(k>=1) {
        if(x[k]<n) {
            x[k]++; s++;
            if(s<=n) {//cont - verific. conditiilor de cont
                if(s==n) {//dc este sol
                    retsol(x, k);
                    s=s-x[k]; k--;//revenire dupa sol
                }
            }
        }
    }
}

```



```

void back() {
    int k=1, s=0; int x[]=new int[n+1];
    x[1]=0;
    while(k>=1) {
        if(x[k]<n) {
            x[k]++; s++;
            if(s<=n) { //cont - verific. conditiilor de cont
                if(s==n) { //dc este sol
                    retsol(x, k);
                    s=s-x[k]; k--; //revenire dupa sol
                }
            }
            else{ k++; x[k]=x[k-1]-1; s+=x[k]; //avansare
            }
        }
    }
}

```

```

void back() {
    int k=1, s=0; int x[]=new int[n+1];
    x[1]=0;
    while(k>=1) {
        if(x[k]<n) {
            x[k]++; s++;
            if(s<=n) { //cont - verific. conditiilor de cont
                if(s==n) { //dc este sol
                    retsol(x, k);
                    s=s-x[k]; k--; //revenire dupa sol
                }
            }
            else{ k++; x[k]=x[k-1]-1; s+=x[k]; //avansare
            }
        }
        else{ s=s-x[k]; k--; //revenire
        }
    }
}

```

Metoda Backtracking - Partiții

- **DE EVITAT** recalcularea lui s la fiecare pas ca fiind $s = x[1] + \dots + x[k]$

Backtracking în plan

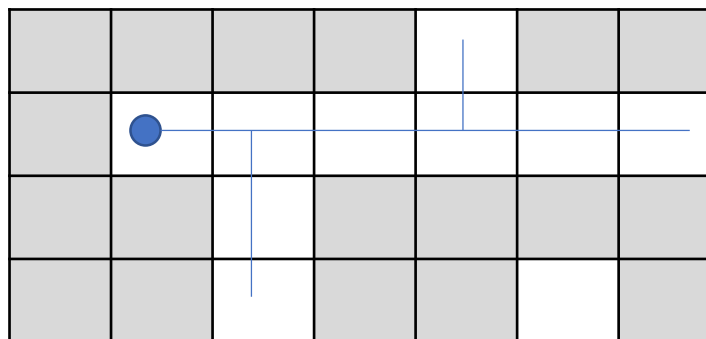
Backtracking în plan

- **Labirint.** Se consideră un caroiaj (matrice) A cu m linii și n coloane.

Pozițiile pot fi:

- libere: $a_{ij}=0$;
- ocupate: $a_{ij}=1$.

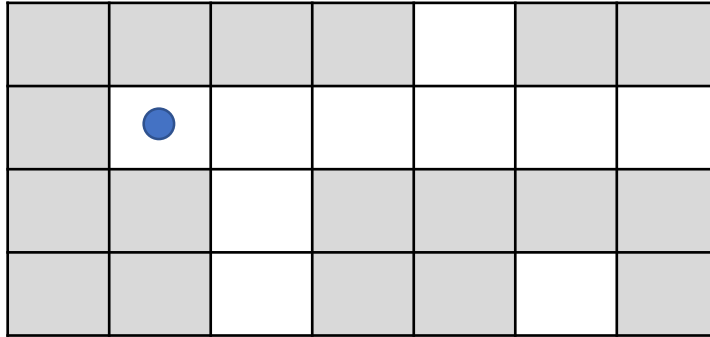
Se mai dă o poziție (i_0, j_0) . Se caută **toate** drumurile care ies în afara matricei, trecând numai prin poziții libere (fără a trece de două ori prin aceeași poziție).



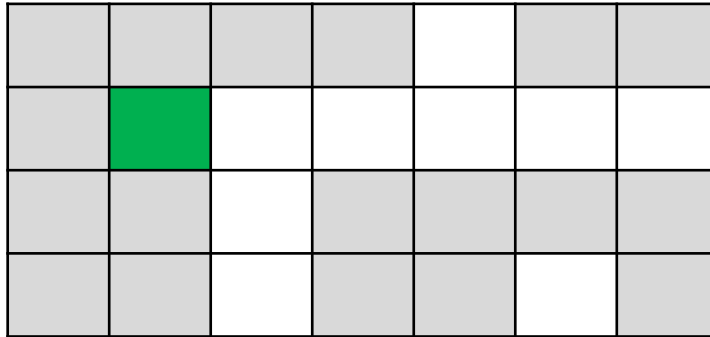
Variante:

- drumul maxim
- drumul minim

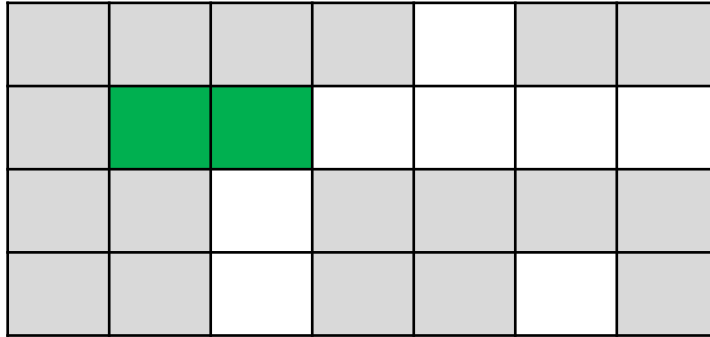
Backtracking în plan



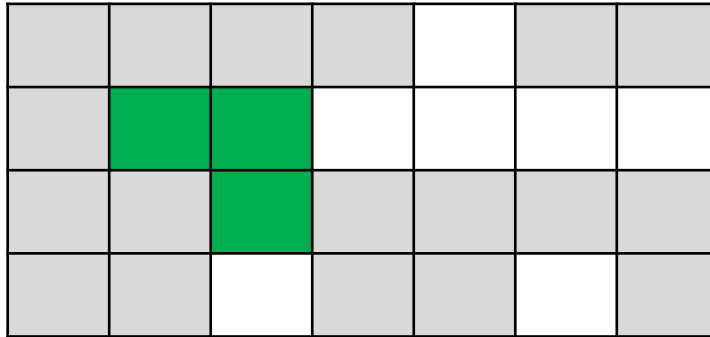
Backtracking în plan



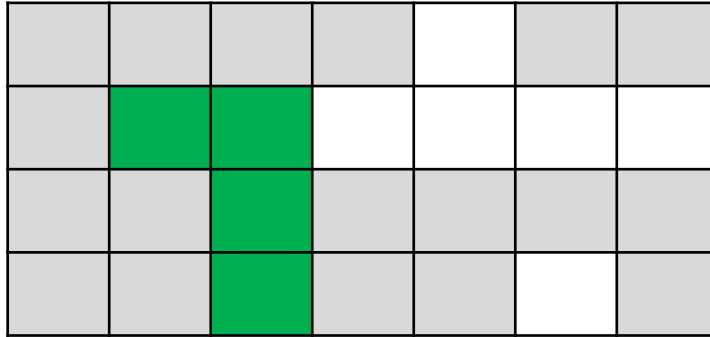
Backtracking în plan



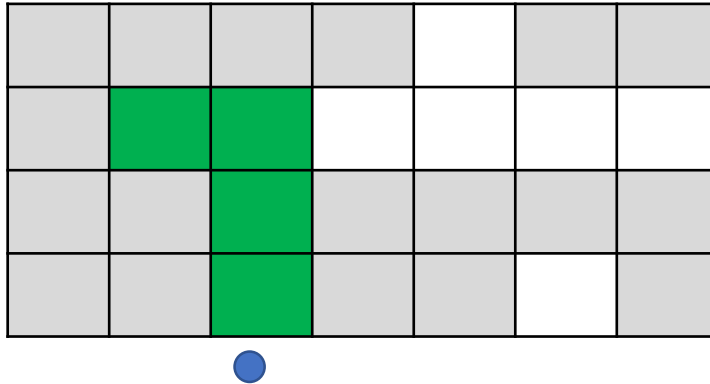
Backtracking în plan



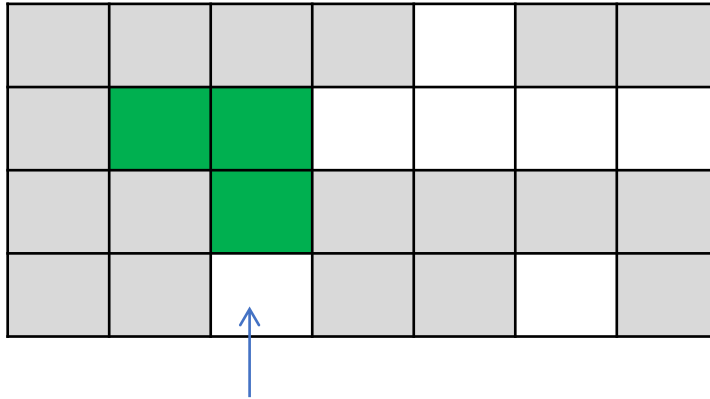
Backtracking în plan



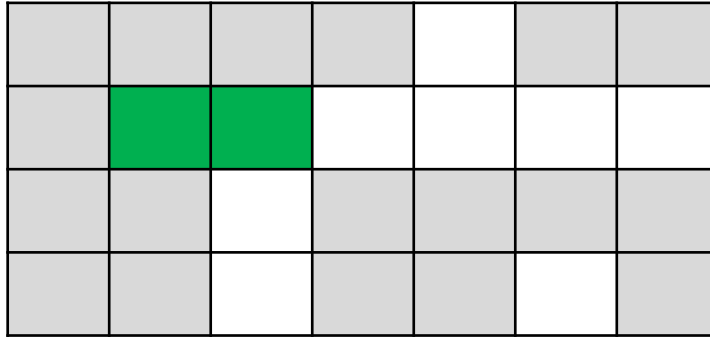
Backtracking in plan



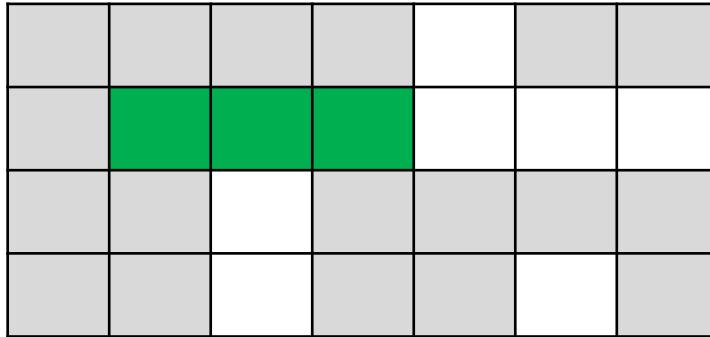
Backtracking in plan



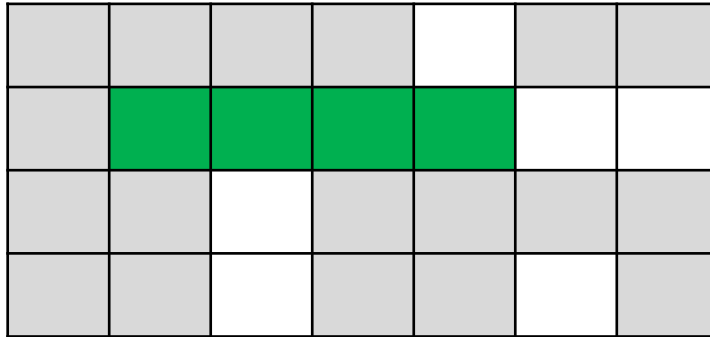
Backtracking în plan



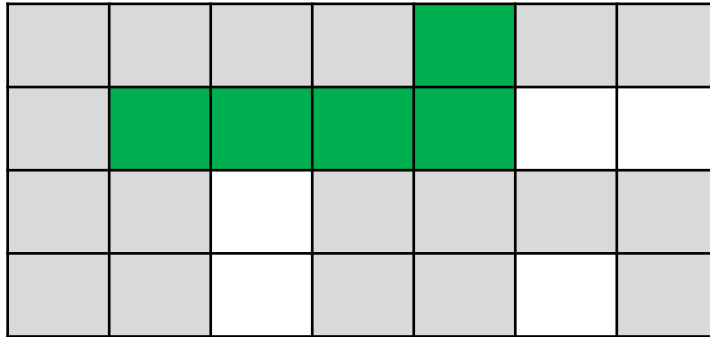
Backtracking în plan



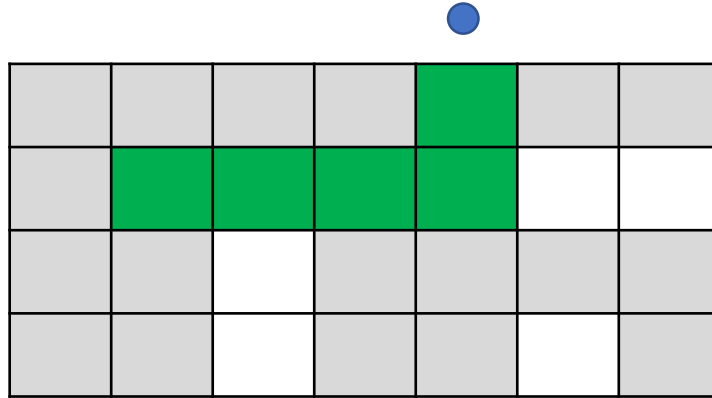
Backtracking în plan



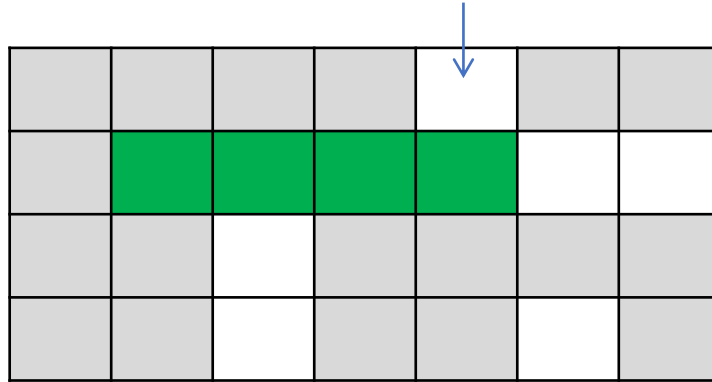
Backtracking în plan



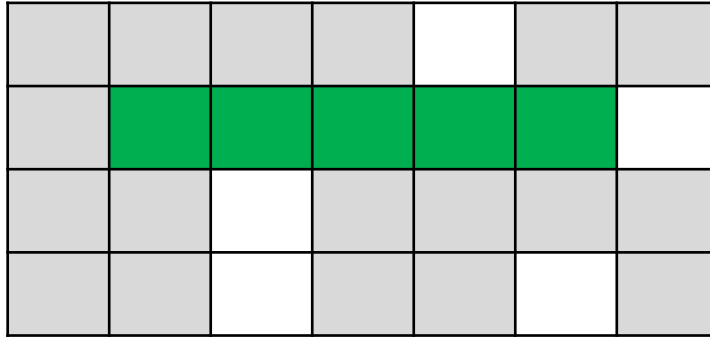
Backtracking în plan



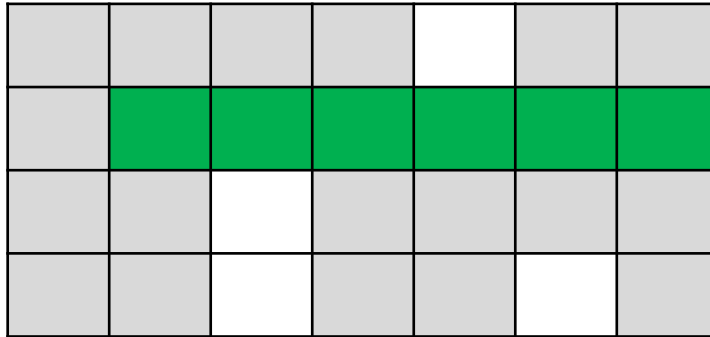
Backtracking în plan



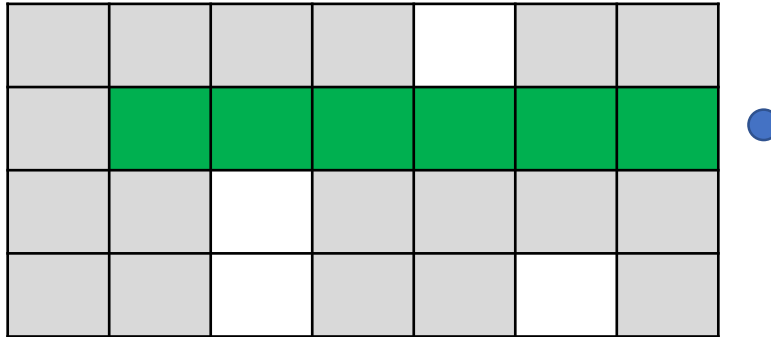
Backtracking în plan



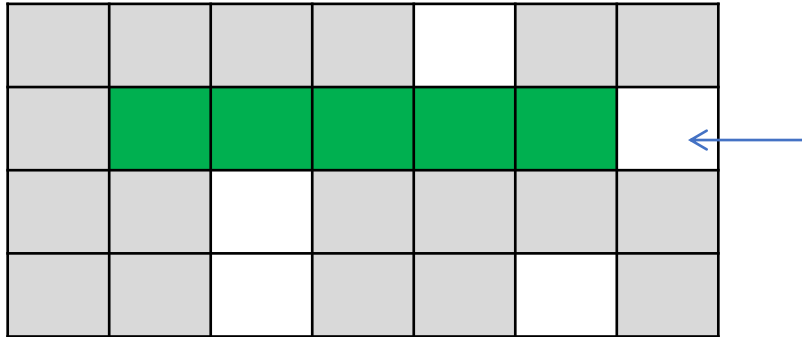
Backtracking în plan



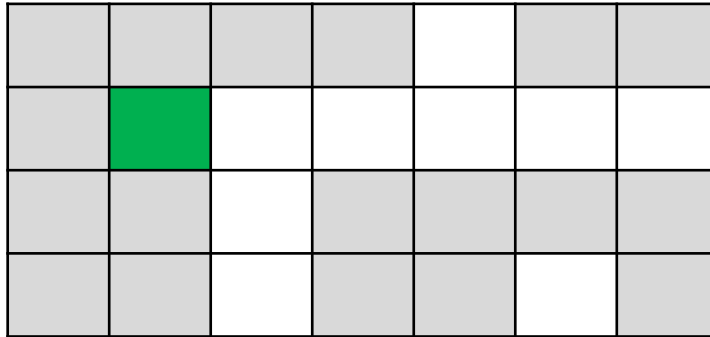
Backtracking in plan



Backtracking in plan



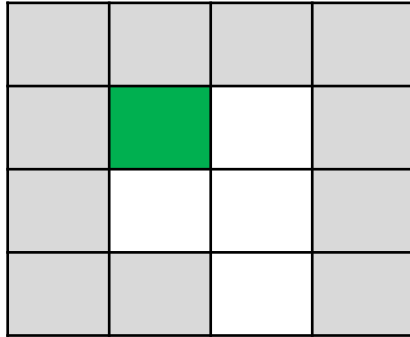
Backtracking în plan



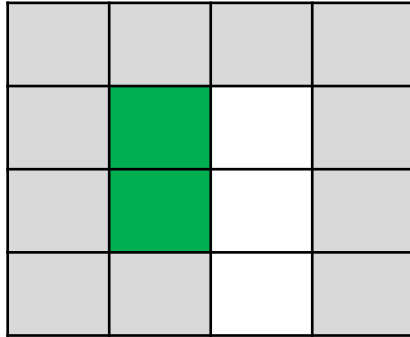
Backtracking în plan

- ▶ **Observație:** este important să demarcăm celulele când facem pasul înapoi

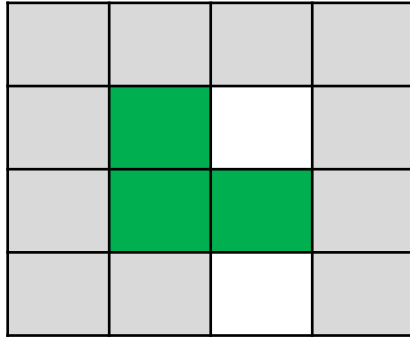
Backtracking în plan



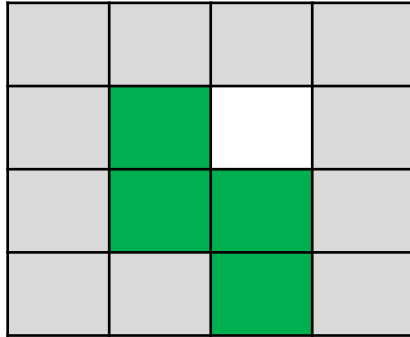
Backtracking în plan



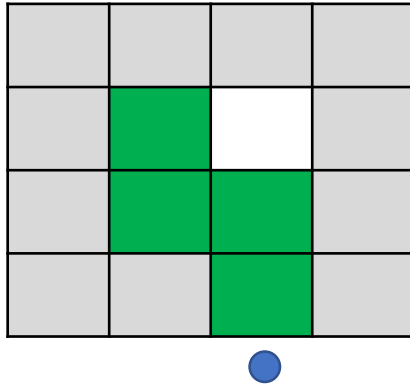
Backtracking în plan



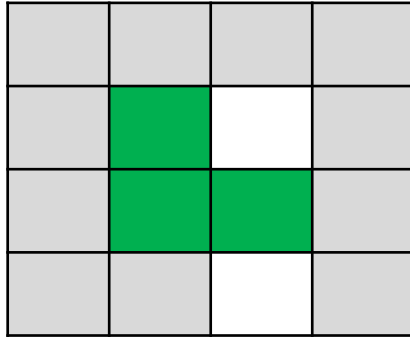
Backtracking în plan



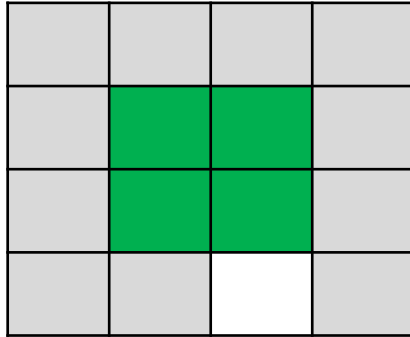
Backtracking în plan



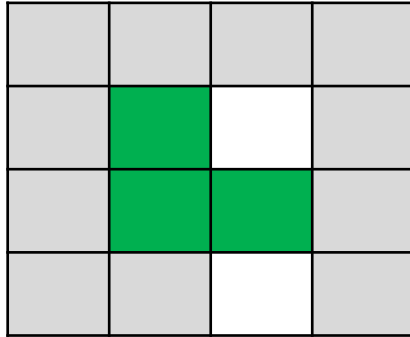
Backtracking în plan



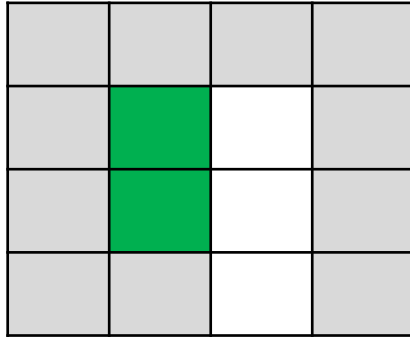
Backtracking în plan



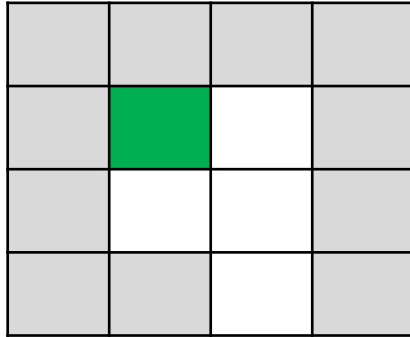
Backtracking în plan



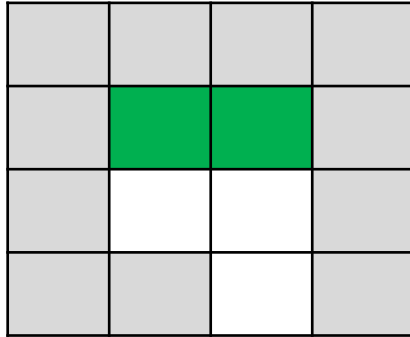
Backtracking în plan



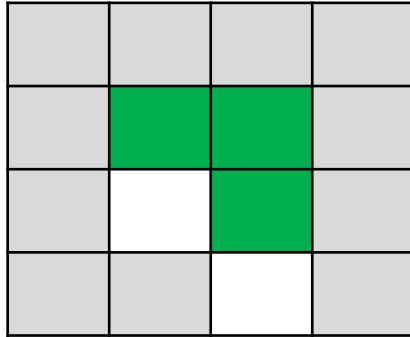
Backtracking în plan



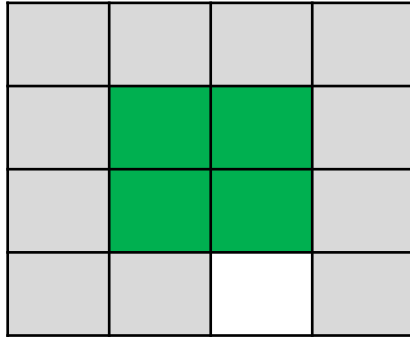
Backtracking în plan



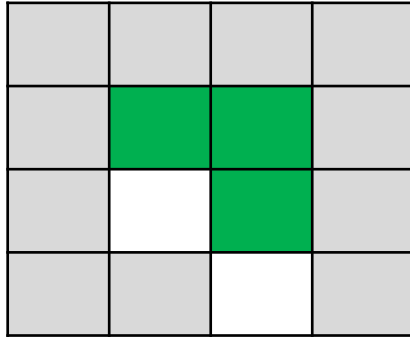
Backtracking în plan



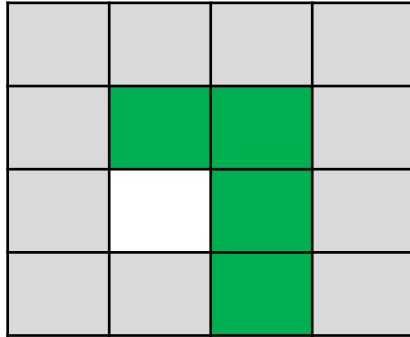
Backtracking în plan



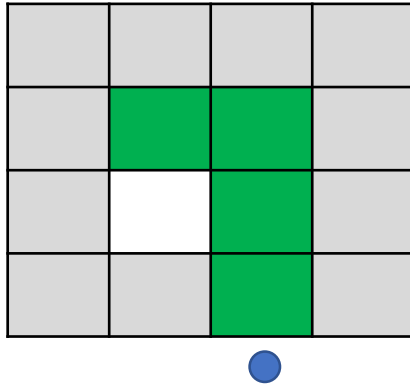
Backtracking în plan



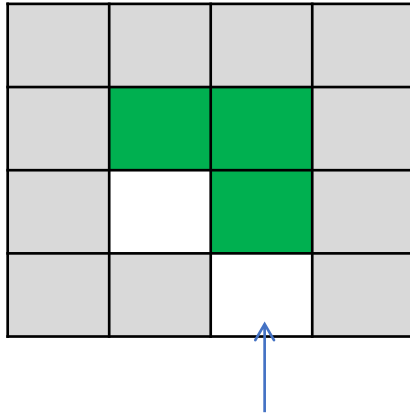
Backtracking în plan



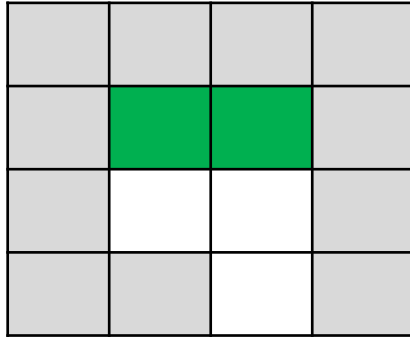
Backtracking in plan



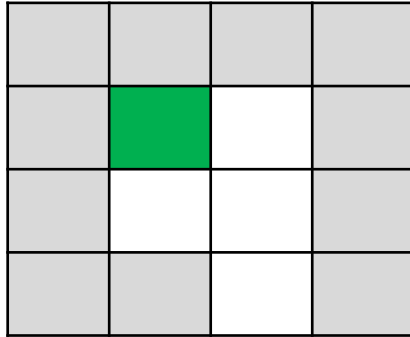
Backtracking in plan



Backtracking în plan



Backtracking în plan



Backtracking în plan

- Bordăm matricea cu 2 pentru a nu studia separat ieșirea din matrice.

- **Reprezentarea soluției**

$\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k\}$, unde
 \mathbf{x}_i = a i-a celulă din drum

- **Condiții interne (finale)**

- **Condiții de continuare**

- **Reprezentarea soluției**

$\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k\}$, unde

\mathbf{x}_i = a i-a celulă din drum

- **Condiții interne (finale)**

\mathbf{x}_k = celulă din afara matricei (marcată cu 2)

$\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{k-1}\}$ – celule libere (marcată cu 0)

- **Condiții de continuare**

\mathbf{x}_k celulă liberă prin care nu am mai trecut

Backtracking în plan

- **dacă poziția este liberă și putem continua, setăm $a_{ij}=-1$ (a fost atinsă), continuăm**
- repunem $a_{ij}=0$ la întoarcere (din recursivitate)

Backtracking în plan

- **dacă poziția este liberă și putem continua, setăm $a_{ij}=-1$ (a fost atinsă), continuăm**
- repunem $a_{ij}=0$ la întoarcere (din recursivitate)
- Matricea deplasărilor $dep1$ cu două linii și $ndep1$ coloane :

$$\begin{pmatrix} 1 & 0 & -1 & 0 \\ 0 & -1 & 0 & 1 \end{pmatrix}$$


```
void back(i, j){
    for (t = 1; t<=ndepl; t++){
        ii = i + depl[1][t]
        jj = j + depl[2][t];

    }
}
```

```

void back(i, j){
    for (t = 1; t<=ndepl; t++){
        ii = i + depl[1][t]
        jj = j + depl[2][t];
        if (a[ii][jj] == 1)
        else
            if (a[ii][jj] == 2)
                retsol(x,k);
            else
                if (a[ii][jj] == 0) {

                    }

            }
    }
}

```

```

void back(i, j){
    for (t = 1; t<=ndepl; t++){
        ii = i + depl[1][t]
        jj = j + depl[2][t];
        if (a[ii][jj] == 1)
        else
            if (a[ii][jj] == 2)
                retsol(x,k);
            else
                if (a[ii][jj] == 0) {
                    k = k+1;           //creste
                     $x_k \leftarrow (ii, jj);$ 
                    a[i][j] = -1; //marcam
                    back(ii, jj);
                }
    }
}

```

```

void back(i, j){
    for (t = 1; t<=ndepl; t++){
        ii = i + depl[1][t]
        jj = j + depl[2][t];
        if (a[ii][jj] == 1)
        else
            if (a[ii][jj] == 2)
                retsol(x,k);
            else
                if (a[ii][jj] == 0) {
                    k = k+1;           //creste
                     $x_k \leftarrow (ii, jj);$ 
                    a[i][j] = -1; //marcam
                    back(ii, jj);
                    a[i][j] = 0;  //demarcam
                    k = k-1 ;      //scade
                }
    }
}

```

Apel:

$x_1 \leftarrow (i_0, j_0);$

$k = 1;$

back(i_0, j_0)

Backtracking în plan

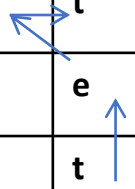
- **Cuvinte.** Se consideră un caroiaj (matrice) A cu m linii și n coloane cu litere și un cuvânt c.

Să se determine dacă c se poate regăsi în matrice pornind dintr-o celulă și deplasându-ne în oricare din celulele vecine pe orizontală, verticală sau diagonală fără a trece de două ori prin aceeași celulă –

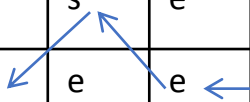
TEMĂ

c = test

s	s	e	s	t	t	a
b	a	s	e	e	e	t
b	t	e	e	t	e	e
a	a	a	e	c	e	e



s	s	e	s	t	t	a
b	a	s	e	e	e	t
b	t	e	e	t	e	e
a	a	a	e	c	e	e



Backtracking în plan

- **Cuvinte**

Indicații:

- similar cu Labirint
- punct de start poate fi orice celulă care conține prima literă din c
- printre condiții de continuare: litera la care am ajuns în matrice la pasul k trebuie să fie a k-a literă din c

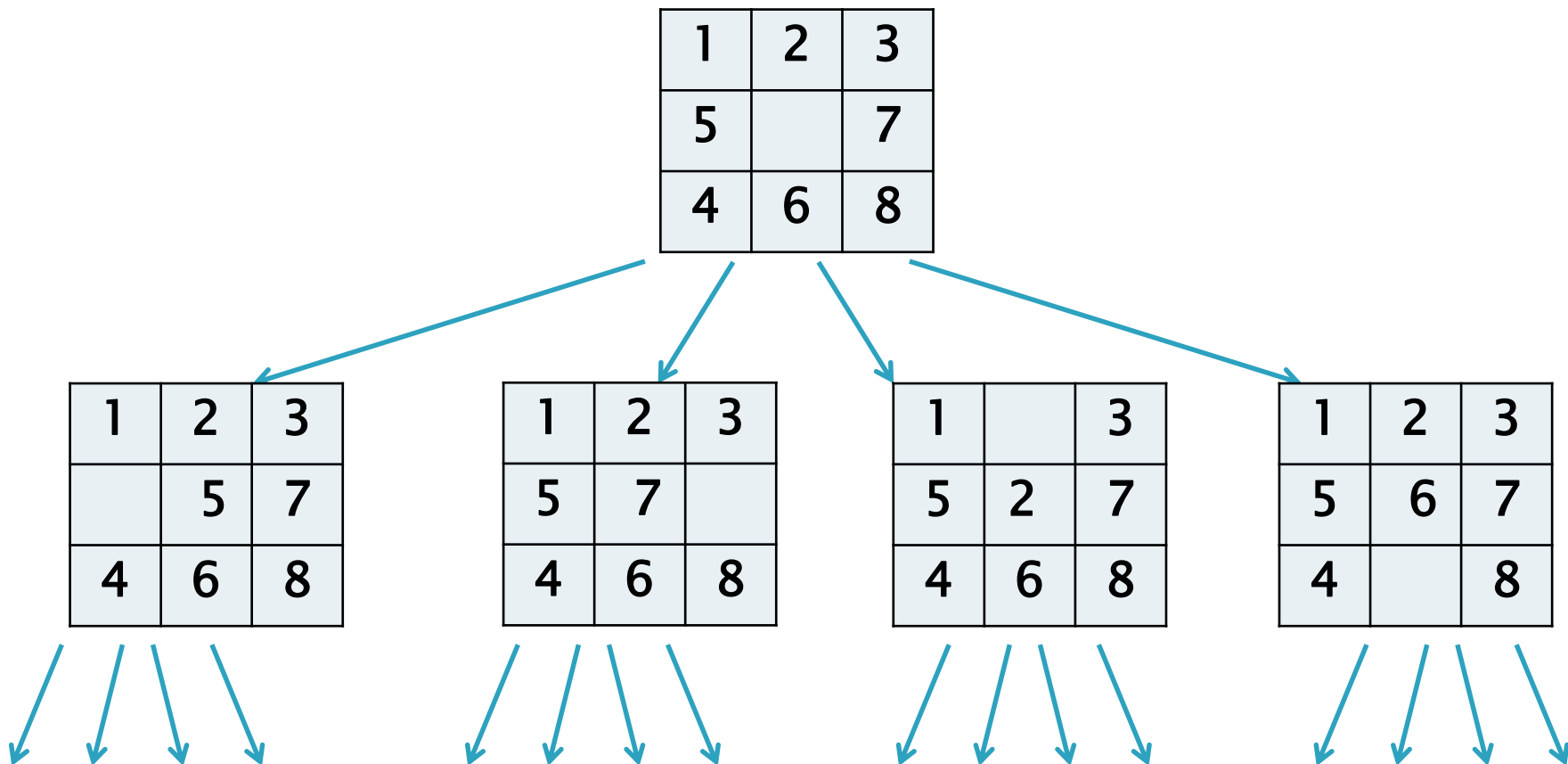
Backtracking în plan

- **Arbori asociați mutărilor într-un joc**
 - Pentru dimensiuni mici
 - În multe cazuri arborele poate deveni de dimensiune mare și un algoritm backtracking va fi lent
 - **Trebuie alte strategii de generare și parcurgere a arborelui decât parcurgerea în adâncime**

Backtracking în plan

- **Arbori asociați mutărilor într-un joc**

- Pentru dimensiuni mici (exemplu: Perspico, Peg solitaire)
https://en.wikipedia.org/wiki/Peg_solitaire
- În multe cazuri arborele poate deveni de dimensiune mare și un algoritm backtracking va fi lent
- **Trebuie alte strategii de generare și parcurgere a arborelui decât parcurgerea în adâncime**



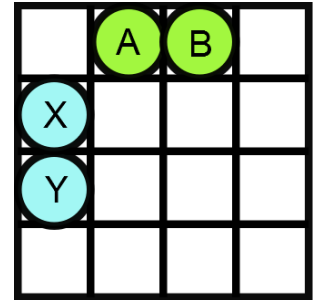
Perspico 3x3

▶ 2x2 fake-sugar-packet game

<http://jeffe.cs.illinois.edu/teaching/algorithms/notes/03-backtracking.pdf>

▶ Stare inițială – tablă nxn

- Jucatorul 1 – piese pe prima linie (A,B)
 - mută piese în jos
 - trebuie să aducă piesele pe ultima linie
- Jucatorul 2 – piese pe prima coloană (X,Y)
 - mută piese în dreapta
 - trebuie să aducă piesele pe ultima coloană

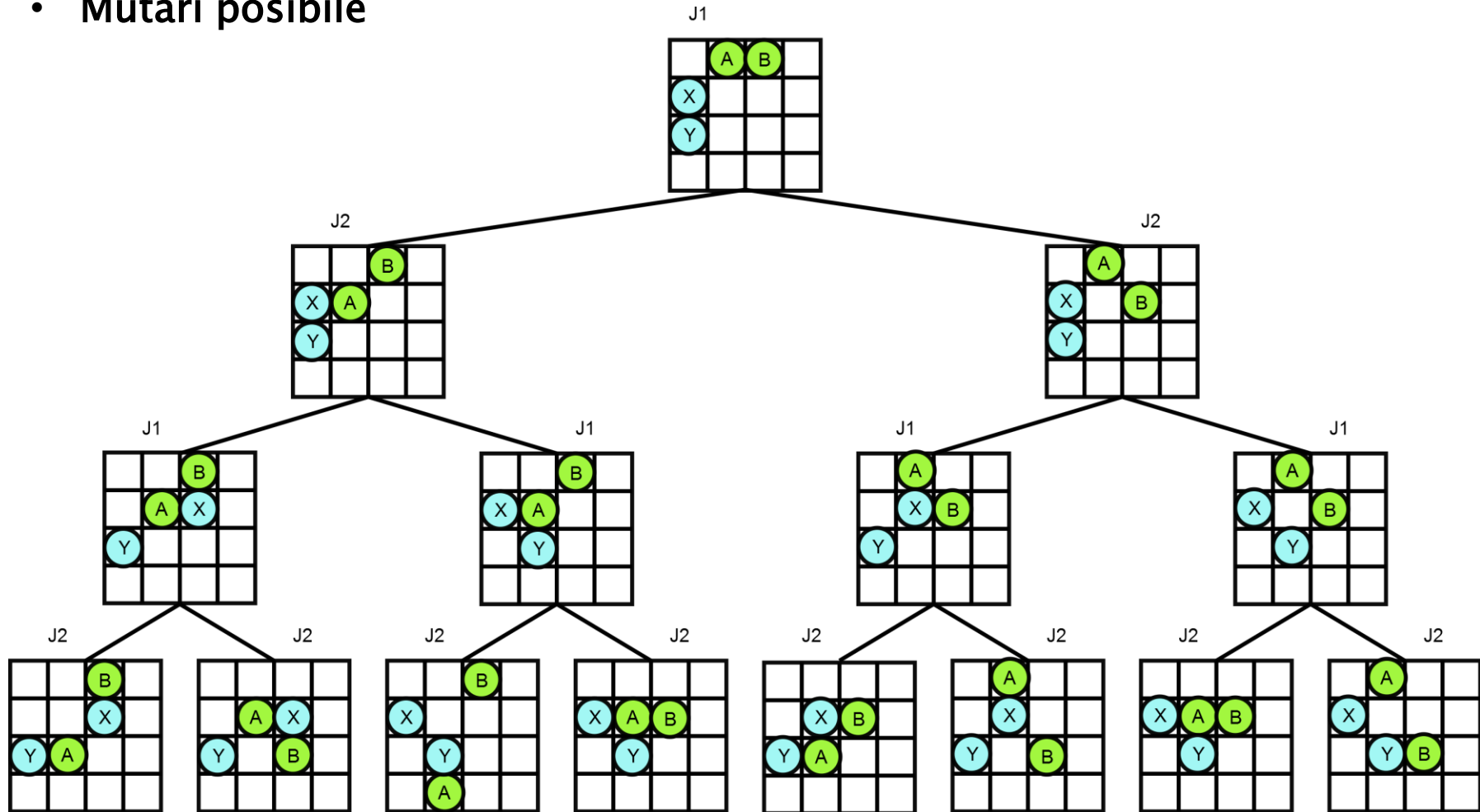


▶ Câștigă – primul care aduce toate piesele unde trebuie

▶ Mutări posibile

- piesa în celula următoare (dacă e liberă)
- piesa se mută 2 celule dacă sare o piesă a adversarului și celula aflată la distanța 2 este liberă

- Mutări posibile



2x2 fake-sugar-packet game

<http://jeffe.cs.illinois.edu/teaching/algorithms/notes/03-backtracking.pdf>

Backtracking în plan

▶ Arbori asociați mutărilor într-un joc

- **Temă** – Determinați o soluție (la un joc de o persoană) sau o strategie de câștig (pentru un joc de două persoane de dimensiune mică) folosind metoda backtracking

Metoda Branch and Bound

Metoda Branch and Bound

Asemănări cu backtracking

- Pentru o configurație se poate **estima** dacă **nu** poate fi completată până la o soluție (mai bună decât cea mai bună soluție determinată până la momentul curent, în cazul problemelor de optim) -> nu mai este explorată

nu se mai parcurge subarborele/ramificațiile care îl au ca rădăcină \Rightarrow branch and bound

Metoda Branch and Bound

Asemănări cu backtracking

- se aplică problemelor care pot fi reprezentate pe un arbore – la un pas avem de ales între mai multe variante
- Vârfurile arborelui (configurațiile) corespund stărilor posibile în dezvoltarea soluției (soluții parțiale)
- Pentru o configurație se poate **estima** dacă **nu** poate fi completată până la o soluție (mai bună decât cea mai bună soluție determinată până la momentul curent, în cazul problemelor de optim)

Metoda Branch and Bound

- **Diferențe față de backtracking**

- ordinea de parcurgere a arborelui (nu neapărat DF)
- modul în care sunt eliminați subarborii care nu pot conduce la o soluție
- arborele poate fi infinit
- util în probleme de optim

Metoda Branch and Bound

- Două tipuri de probleme la care se poate utiliza:
 - Se caută o anumită soluție = un anumit ***vârf rezultat*** (final, frunză)
 - Se caută o **soluție optimă** (există mai multe vârfuri finale)
 - probleme de optim – principalele aplicații

Exemplu - Jocul 15 (Perspico)

1	2	3	4
5		7	8
9	6	10	11
13	14	15	12

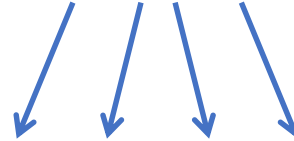
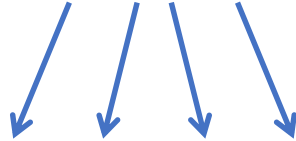
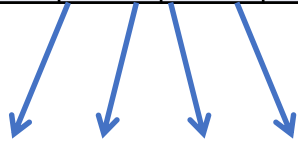


1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

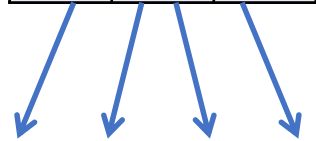
1	2	3	4
5		7	8
9	6	10	11
13	14	15	12



1	2	3	4
5	7		8
9	6	10	11
13	14	15	12



1	2	3	4
5	6	7	8
9		10	11
13	14	15	12



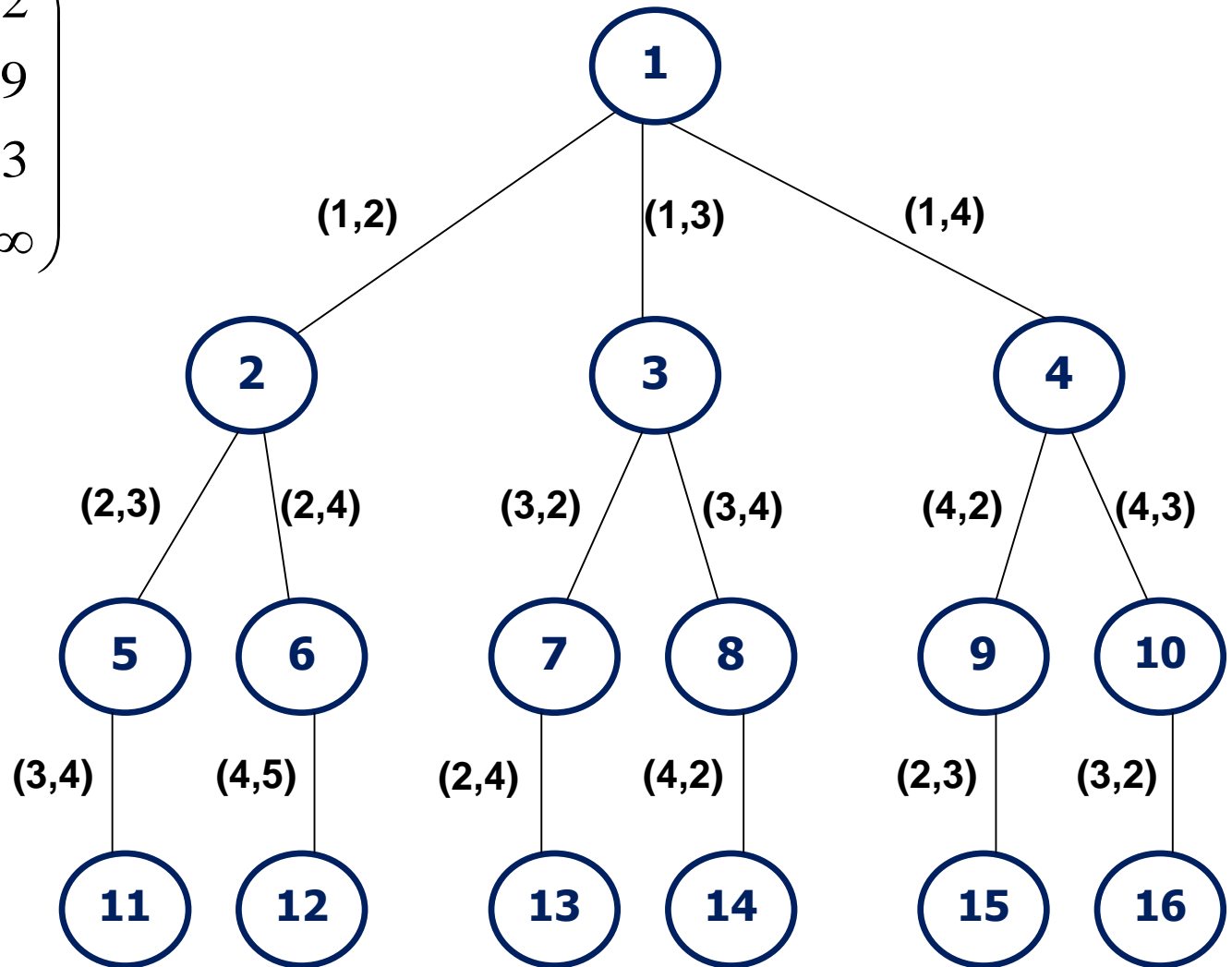
Exemplul 2 – circuit hamiltonian minim

Matricea costurilor:

$$C = \begin{pmatrix} \infty & 3 & 7 & 2 \\ 5 & \infty & 1 & 9 \\ 4 & 8 & \infty & 3 \\ 6 & 2 & 6 & \infty \end{pmatrix}$$

Exemplul 2 – circuit hamiltonian minim

$$C = \begin{pmatrix} \infty & 3 & 7 & 2 \\ 5 & \infty & 1 & 9 \\ 4 & 8 & \infty & 3 \\ 6 & 2 & 6 & \infty \end{pmatrix}$$



Metoda Branch and Bound

L – lista de vârfuri **active** din arbore (care mai pot fi explorate)

Schemă

- Se inserează în L vârful inițial
 - **Repetă**
 - Se alege un vârf din L care devine **curent**
 - Se generează fiii săi, care se adaugă în L
- până când vârful curent este final**

Metoda Branch and Bound



Cum se alege vârful curent?

Metoda Branch and Bound



Cum se alege vârful curent

- **DF**

- o parte arborele poate fi infinit
- soluția căutată poate fi de exemplu un fiu al rădăcinii diferit de primul fiu

Metoda Branch and Bound



Cum se alege vârful curent

- **DF**

- o parte arborele poate fi infinit
- soluția căutată poate fi de exemplu un fiu al rădăcinii diferit de primul fiu

- **BF**

- conduce totdeauna la soluție, dar poate fi ineficientă dacă vârfurile au mulți fii

Metoda Branch and Bound

- **Compromis:**

Vârf x – asociat un **cost** pozitiv $c(x)$

= măsură a gradului de "apropiere" a
vârfului de o soluție

Metoda Branch and Bound

- **Compromis:**

Vârf x – asociat un **cost** pozitiv $c(x)$

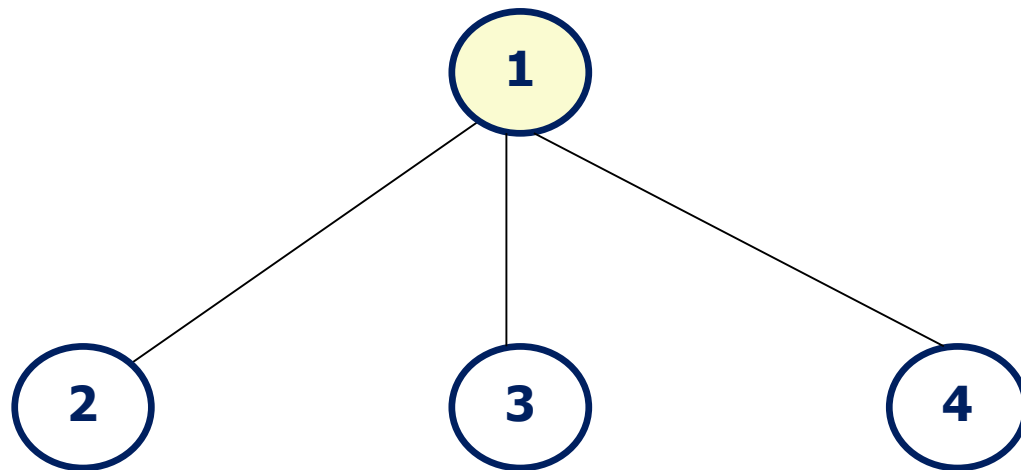
= măsură a gradului de "apropiere" a

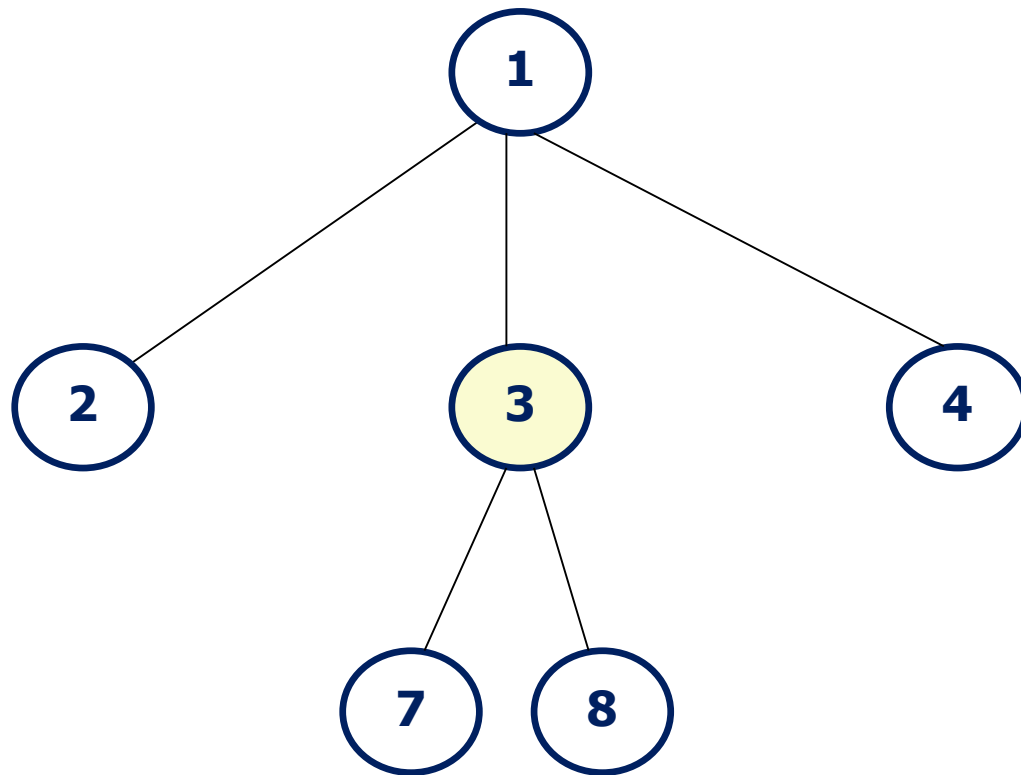
vârfului de o soluție

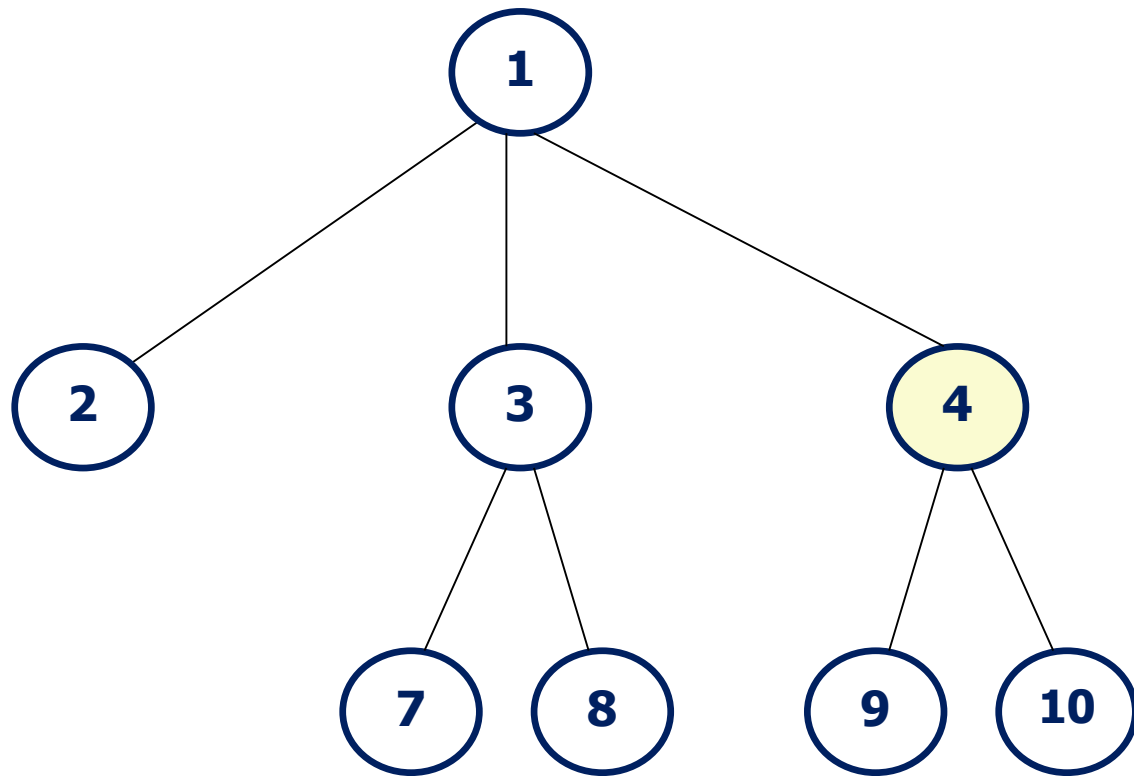
➤ Este ales vârful de cost minim (! care **nu este neapărat fiu al vârfului curent anterior**)

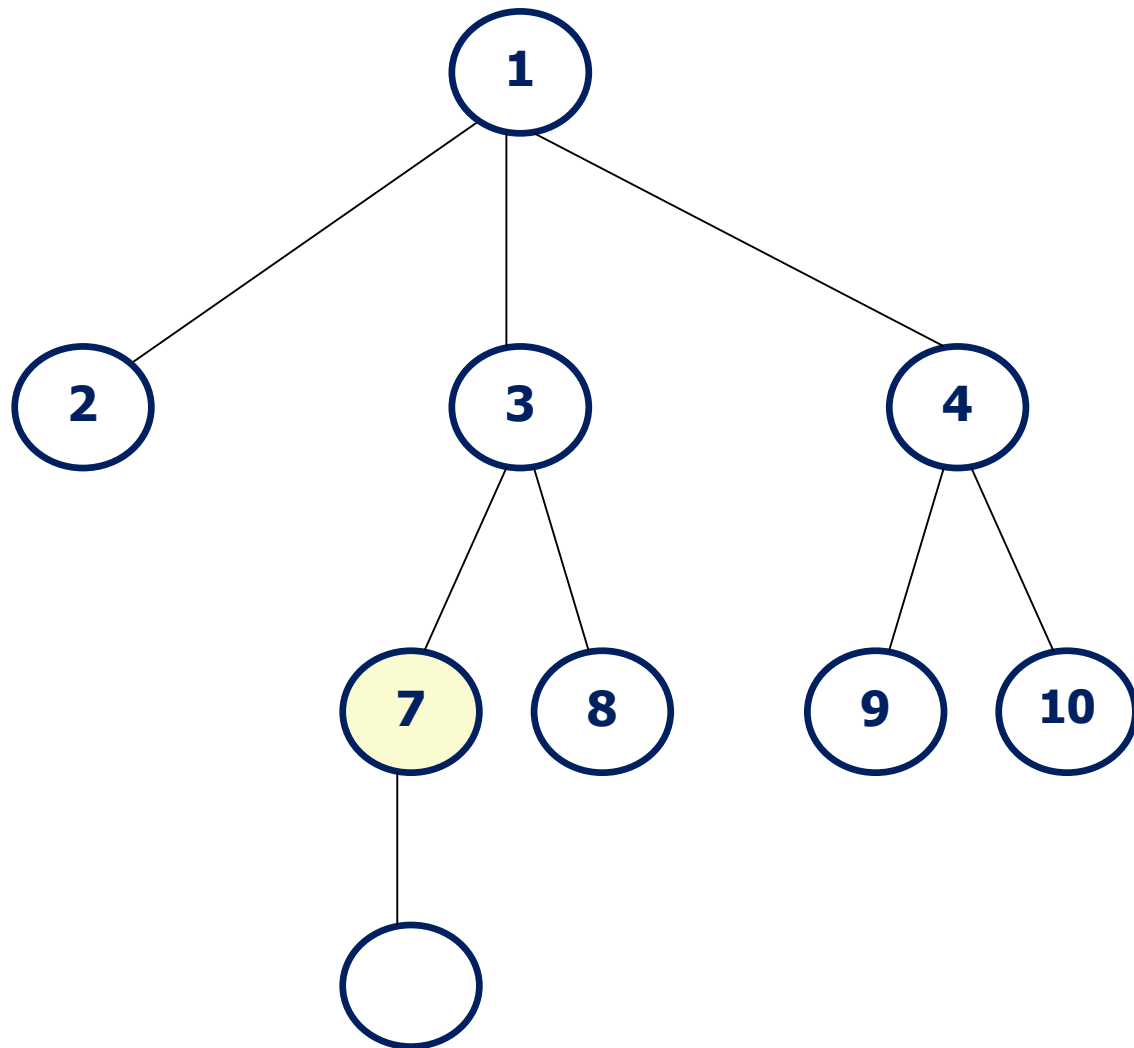
➤ $c(\text{varf}) \leq c(\text{fiu})$

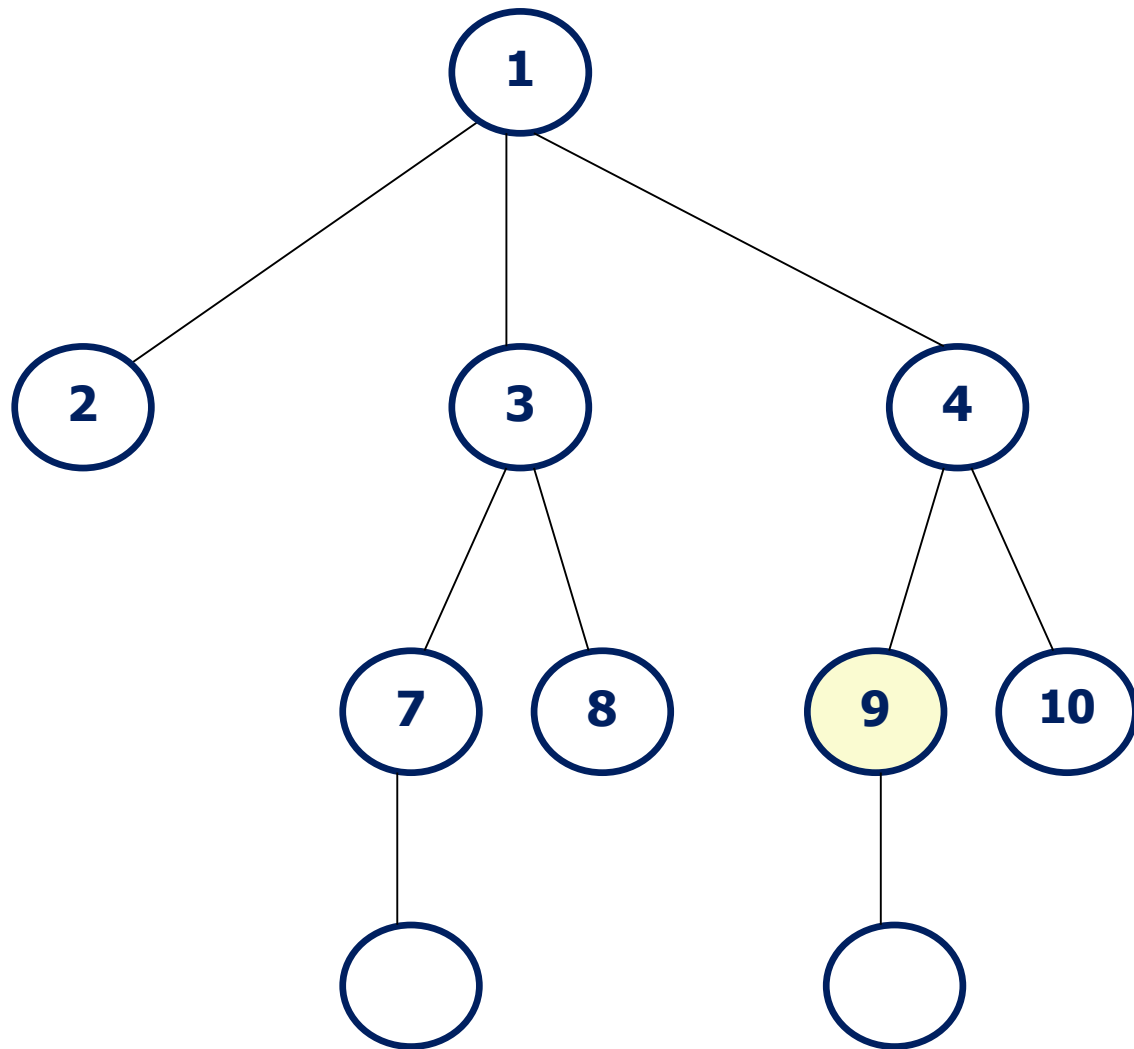
L va fi în general un min-ansamblu











Metoda Branch and Bound

- **Funcție de cost ideală**

$c(x) =$

- **nivel**(x), dacă x este vârf **rezultat**
- $+\infty$, dacă x este vârf **final**, diferit de
vârf rezultat
- **min** { $c(y)$ | y fiu al lui x },
dacă x nu este vârf final

Metoda Branch and Bound

- **Funcție de cost ideală**

$$c(x) =$$

- **nivel(x)**, dacă x este vârf **rezultat**
- $+\infty$, dacă x este vârf **final**, diferit de
vârf rezultat
- **min** { $c(y)$ | y fiu al lui x },
dacă x nu este vârf final
 - Nu se poate calcula pentru arbore infinit
 - Trebuie parcurs arborele în întregime pentru a o calcula

Metoda Branch and Bound

- **Aproximație f a lui c**

Metoda Branch and Bound

- **Aproximație f a lui c**

- $f(x)$ să poată fi calculată doar pe baza informațiilor din drumul de la rădăcină la x
- este indicat ca $\mathbf{f} \leq \mathbf{c}$ (aproximare optimistă)

Metoda Branch and Bound

- **Condiție compromis DF/BF:**

Există un k natural - pentru orice vârf x situat pe un nivel n_x și orice vârf y situat pe un nivel $n_y \geq n_x + k$, să avem

$$f(x) < f(y)$$

\Rightarrow dacă există soluție, ea va fi atinsă într-un timp finit

Metoda Branch and Bound

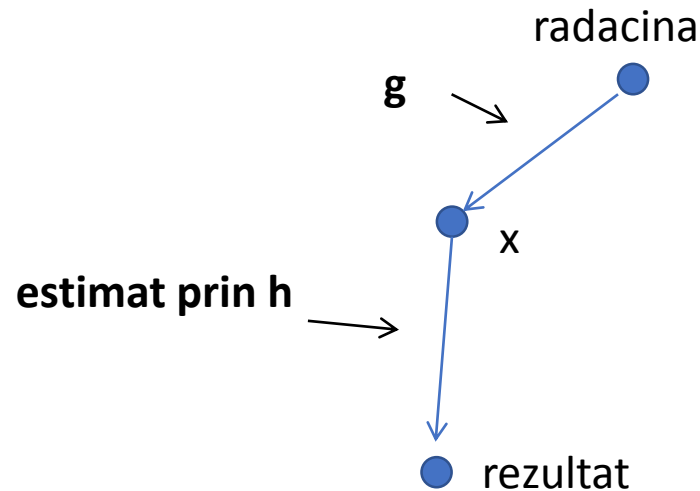
- **Cum definim f ?**

Metoda Branch and Bound

- Cum definim f ? O posibilitate ar fi:

$$f(x) = g(x) + h(x)$$

- $g(x)$ = distanța de la rădăcină la nodul x
- $h(x)$ = estimarea distanței de la x la vârful rezultat ("subestimare")



Exemplu - Jocul 15 (Perspico)

1	2	3	4
5		7	8
9	6	10	11
13	14	15	12



1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Exemplu - Jocul 15 (Perspico)

- **Condiție de existență**

- căsuța liberă **16**, pe poziția (l, c)

- pentru plăcuța etichetată cu i calculăm

$n(i)$ = numărul locașurilor care îi urmează și care
conțin o plăcuță cu etichetă
mai mică decât i

Există soluție \Leftrightarrow

$n(1)+n(2)+\dots+n(16)+(l+c) \% 2$ este par.

Exemplu - Jocul 15 (Perspico)

- Putem considera euristici precum:
 - $h(t)$ = numărul de plăcuțe care nu sunt la locul lor
 - $h(t)$ = **distanța Manhattan** = suma pentru fiecare căsuță a **numărului de mutări** pentru a o putea aduce în poziția finală
 - dacă o cifră este pe poziția (i,j) și trebuie adusă pe poziția (r,s) distanța este

$$|r - i| + |s - j|$$

Exemplu - Jocul 15 (Perspico)

- Putem considera euristici precum:
 - $h(t)$ = numărul de plăcuțe care nu sunt la locul lor
 - $h(t)$ = **distanța Manhattan** = suma pentru fiecare căsuță a **numărului de mutări** pentru a o putea aduce în poziția finală
- $h(t)$ = numărul de plăcuțe care nu sunt la locul lor

**h = câte plăcuțe nu
sunt la locul lor**

1	2	3	4
5		7	8
9	6	10	11
13	14	15	12

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

$g = 0$
 $h = 5$

1	2	3	4
5		7	8
9	6	10	11
13	14	15	12

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

$g=0, h=5$
 $f=5$

1	2	3	4
5		7	8
9	6	10	11
13	14	15	12

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

$g=1, h=6$
 $f=7$

1	2	3	4	4
	5	7	8	8
9	6	10	11	11
13	14	15	12	12

$g=1, h=6$
 $f=7$



$g=1, h=4$
 $f=5$



$g=1, h=6$
 $f=7$

1	2	3	4
5	7		8
9	6	10	11
13	14	15	12

$g=0, h=5$
 $f=5$

1	2	3	4
5		7	8
9	6	10	11
13	14	15	12

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

$g=1, h=6$
 $f=7$

1	2	3	4	4
	5	7	8	8
9	6	10	11	11
13	14	15	12	12

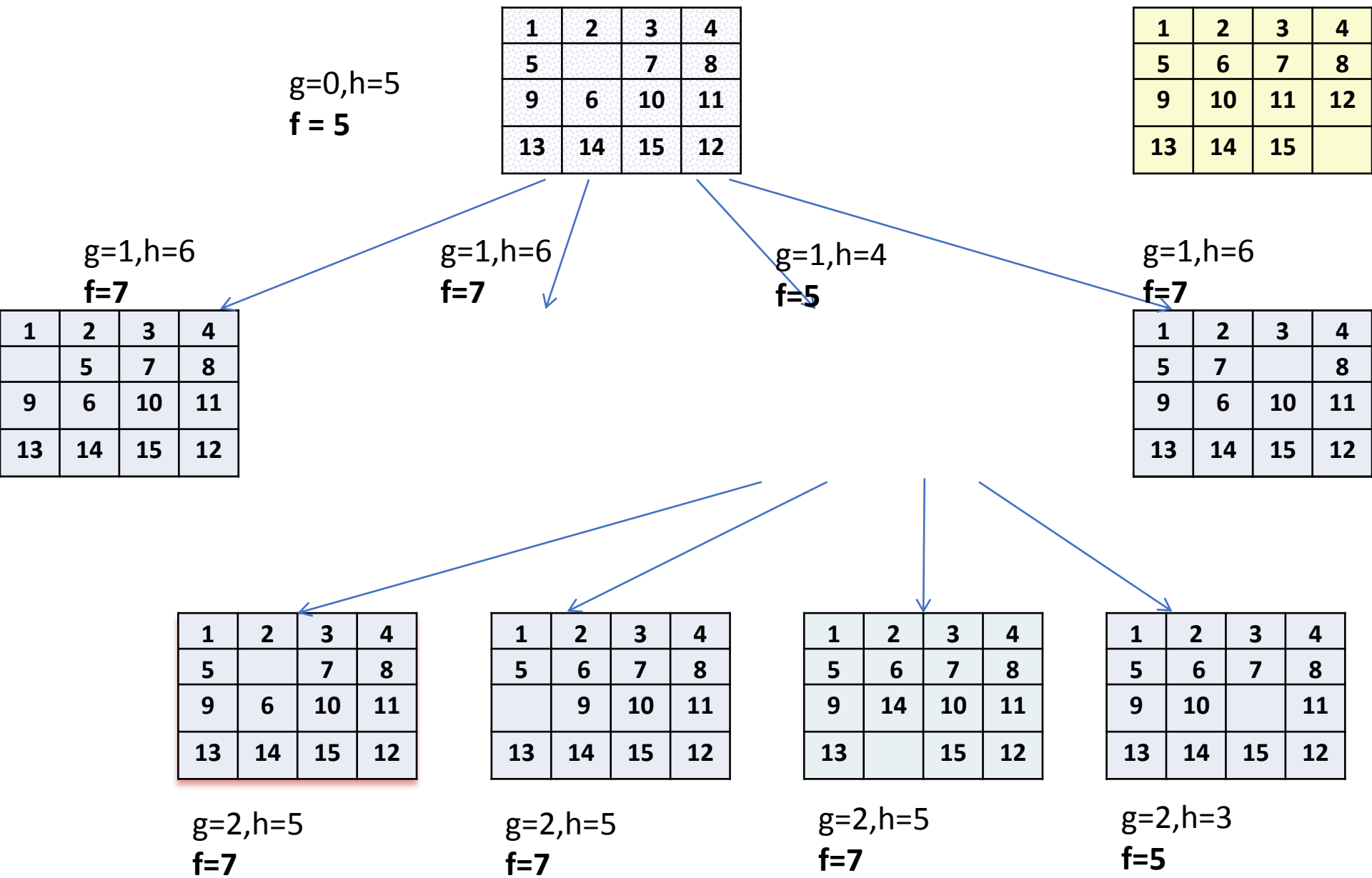
$g=1, h=6$
 $f=7$

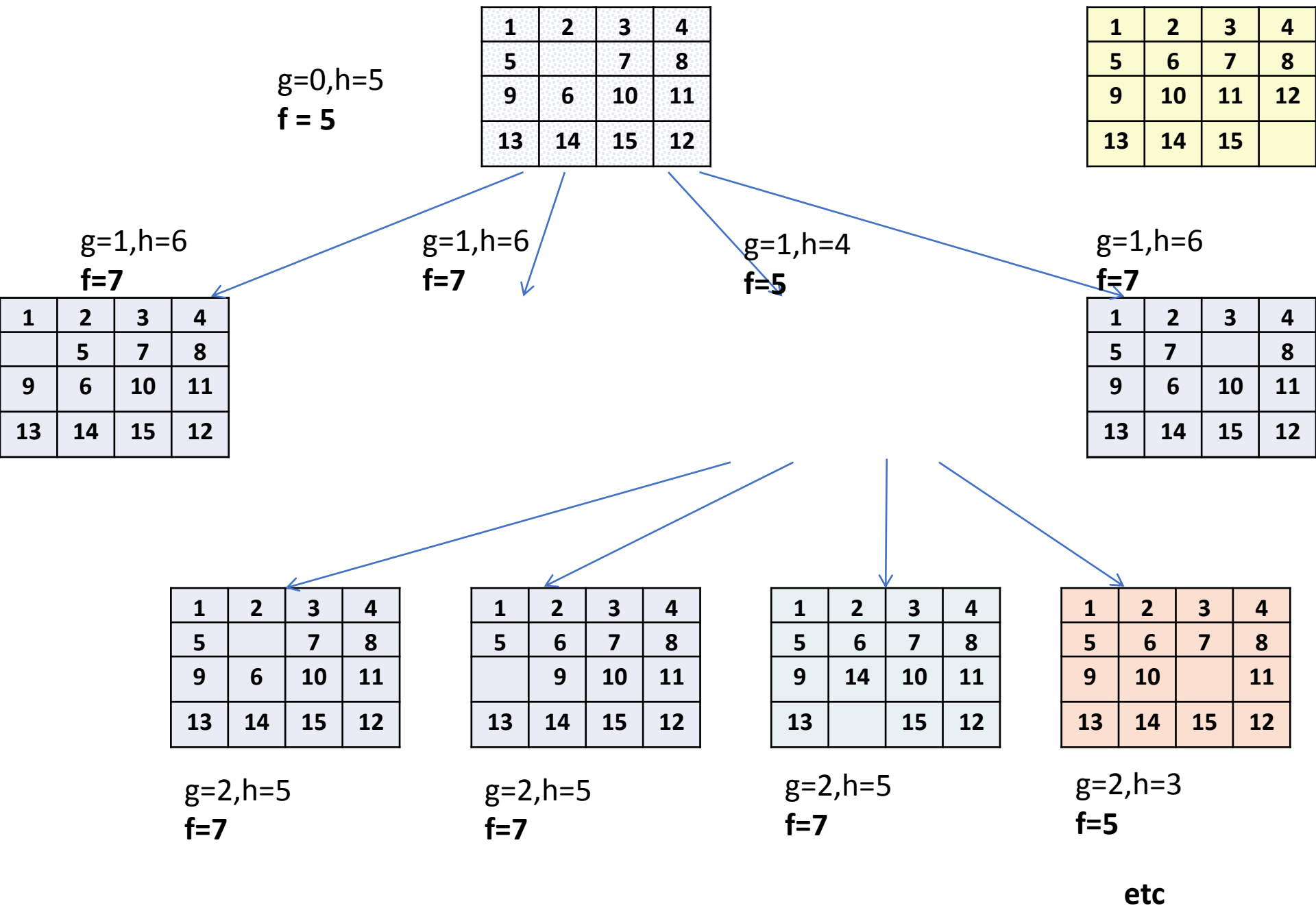


$g=1, h=4$
 $f=5$

$g=1, h=6$
 $f=7$

1	2	3	4
5	7		8
9	6	10	11
13	14	15	12





Metoda Branch and Bound

- **Pentru probleme de optim**

- **Vom considera probleme de minimizare**

- Maximizare – similar

(maximizare obiectiv $g \rightarrow$ minimizare obiectiv $-g$)

Metoda Branch and Bound

- **Pentru probleme de optim (minim)**

- Considerăm lim - **aproximație prin adaos** a minimului căutat, care se actualizează pe parcursul algoritmului = costul celei mai bune soluții găsite până la pasul curent

Metoda Branch and Bound

- **Pentru probleme de optim (minim)**

- Configurațiile cu costul estimat mai mare decât **lim** nu mai trebuie considerate:

$$\text{cost real} \geq \text{cost estimat} > \text{lim}$$

- Dacă o configurație finală (corespunzătoare unei soluții posibile) are costul estimat mai mic decât **lim**, **lim** se actualizează și sunt eliminate din lista vârfurilor active vârfurile cu cost mai mare decât noul lim

Metoda Branch and Bound

- **Pentru probleme de optim (minim)**

- Notăm rad – vârful corespunzător configurației inițiale

```
i ← rad; L ⇐ {i}; min ← lim;  
calculăm f(rad); tata(i) ← 0
```



```
i ← rad; L ← {i}; min ← lim;  
calculăm f(rad); tata(i) ← 0  
while L ≠ ∅  
    i ← L {scos vârful i cu f(i) minim din min-ansamblul L}  
    for toți j fii ai lui i
```

```
i ← rad; L ← {i}; min ← lim;  
calculăm f(rad); tata(i) ← 0  
while L ≠ ∅  
    i ← L {scos vârful i cu f(i) minim din min-ansamblul L}  
    for toți j fii ai lui i  
        calculăm f(j); calcule locale asupra lui j;  
        tata(j) ← i
```

```
i ← rad; L ← {i}; min ← lim;
calculăm f(rad); tata(i) ← 0
while L ≠ ∅
    i ← L {scos vârful i cu f(i) minim din min-ansamblul L}
    for toți j fii ai lui i
        calculăm f(j); calcule locale asupra lui j;
    tata(j) ← i
    if j este vârf final
        if f(j) < min
            min ← f(j); ifinal ← j
            elimină din L vârfurile k cu f(k) ≥ min
```

```

i ← rad; L ← {i}; min ← lim;
calculăm f(rad); tata(i) ← 0
while L ≠ ∅
    i ← L {scos vârful i cu f(i) minim din min-ansamblul L}
    for toți j fii ai lui i
        calculăm f(j); calcule locale asupra lui j;
        tata(j) ← i
        if j este vârf final
            if f(j) < min
                min ← f(j); ifinal ← j
                elimină din L vârfurile k cu f(k) ≥ min
    else
        if f(j) < min
            j ⇒ L

```

```

i ← rad; L ← {i}; min ← lim;
calculăm f(rad); tata(i) ← 0
while L ≠ ∅
    i ← L {scos vârful i cu f(i) minim din min-ansamblul L}
    for toți j fii ai lui i
        calculăm f(j); calcule locale asupra lui j;
        tata(j) ← i
        if j este vârf final
            if f(j) < min
                min ← f(j); ifinal ← j
                elimină din L vârfurile k cu f(k) ≥ min
        else
            if f(j) < min
                j ⇒ L

if min = lim
    write('Nu există soluție')
else writeln(min); i ← ifinal
    while i ≠ 0
        write(i); i ← tata(i)

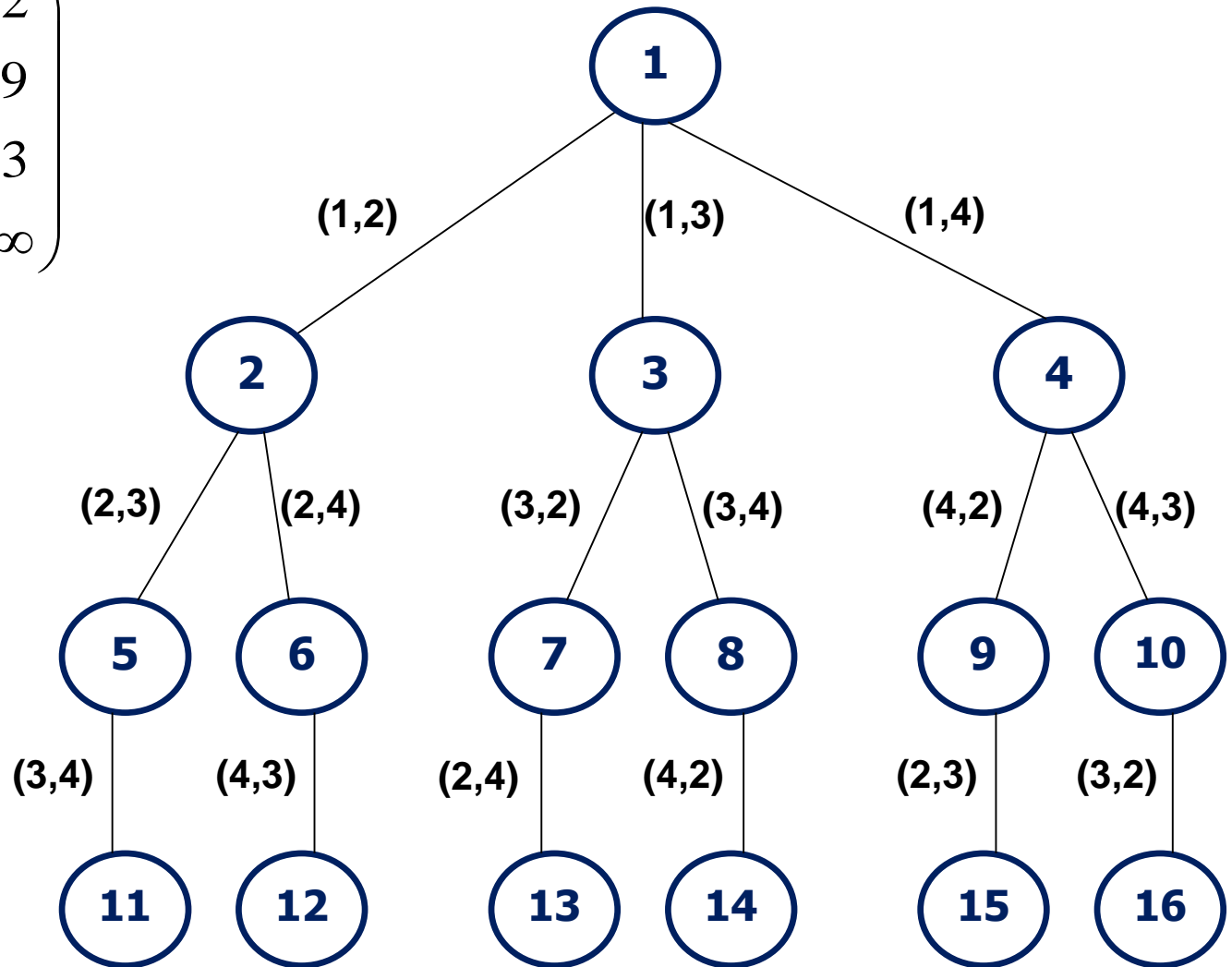
```

Exemplu – circuit hamiltonian minim TSP

$$C = \begin{pmatrix} \infty & 3 & 7 & 2 \\ 5 & \infty & 1 & 9 \\ 4 & 8 & \infty & 3 \\ 6 & 2 & 6 & \infty \end{pmatrix}$$

Exemplu – circuit hamiltonian minim TSP

$$C = \begin{pmatrix} \infty & 3 & 7 & 2 \\ 5 & \infty & 1 & 9 \\ 4 & 8 & \infty & 3 \\ 6 & 2 & 6 & \infty \end{pmatrix}$$



Exemplu – circuit hamiltonian minim TSP

Pentru un vârf x din arbore valoarea $c(x)$ dată de funcția de cost **ideală** este:

- lungimea circuitului corespunzător lui x dacă x este frunză
- $\min \{c(y) \mid y \text{ fiu al lui } x\}$ altfel.

Exemplu – circuit hamiltonian minim TSP



Cum determinăm limita inferioară $f(x)$ pentru circuitului minim corespunzător vârfului x din arborele Branch and Bound

Exemplu – circuit hamiltonian minim TSP

Cum determinăm limita inferioară $f(x)$ pentru circuitului minim corespunzător vârfului x din arborele Branch and Bound

Idee simplă - lungimea circuitului minim care corespunde lui x este \leq lungimea drumului de la rădăcină la x (arcelor alese pentru a ajunge la x)

$$f(x) = g(x) + h(x)$$

□ $g(x)$ = distanța de la rădăcină la nodul x

□ $h(x) = 0 \rightarrow$ este o "subestimare"

Exemplu – circuit hamiltonian minim TSP

Cum determinăm limita inferioară $f(x)$ pentru circuitului minim corespunzător vârfului x din arborele Branch and Bound

- ❑ $h(x) = 0 \rightarrow$ este o "subestimare"
- ❑ Problemă – cu cât estimarea este mai puțin precisă (mai depărtată de valoarea reală) sunt excluse mai puține vârfuri din parcurgere \Rightarrow algoritm mai lent

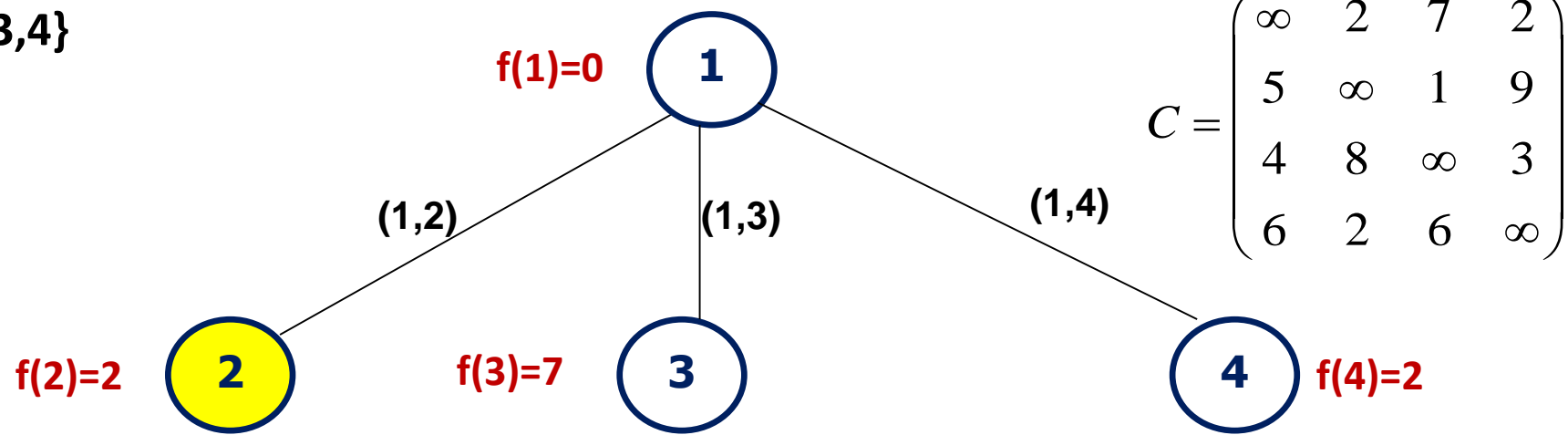
$$L = \{1\}$$

$$f(1)=0$$

$$\textcircled{1}$$

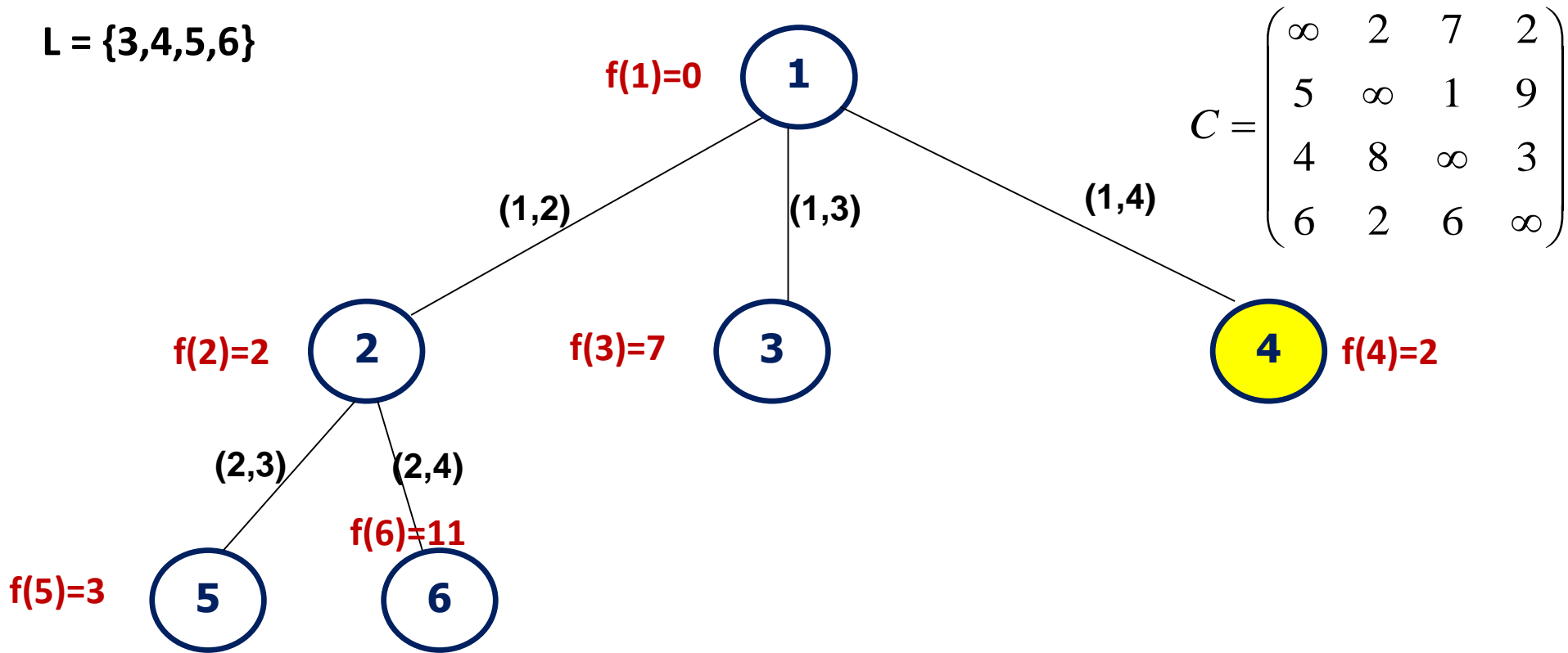
$$C = \begin{pmatrix} \infty & 2 & 7 & 2 \\ 5 & \infty & 1 & 9 \\ 4 & 8 & \infty & 3 \\ 6 & 2 & 6 & \infty \end{pmatrix}$$

$L = \{2,3,4\}$



Extragem din L vârful cu f minim și îi generăm fii

$L = \{3,4,5,6\}$



Extragem din L vârful cu f minim și îi generăm fii

$L = \{3,5,6,9,10\}$

$f(1)=0$

$$C = \begin{pmatrix} \infty & 2 & 7 & 2 \\ 5 & \infty & 1 & 9 \\ 4 & 8 & \infty & 3 \\ 6 & 2 & 6 & \infty \end{pmatrix}$$

$(1,2)$

$(1,3)$

$(1,4)$

$f(2)=2$

$f(3)=7$

$f(4)=2$

$(2,3)$

$(2,4)$

$f(6)=11$

$f(5)=3$

$(4,2)$

$(4,3)$

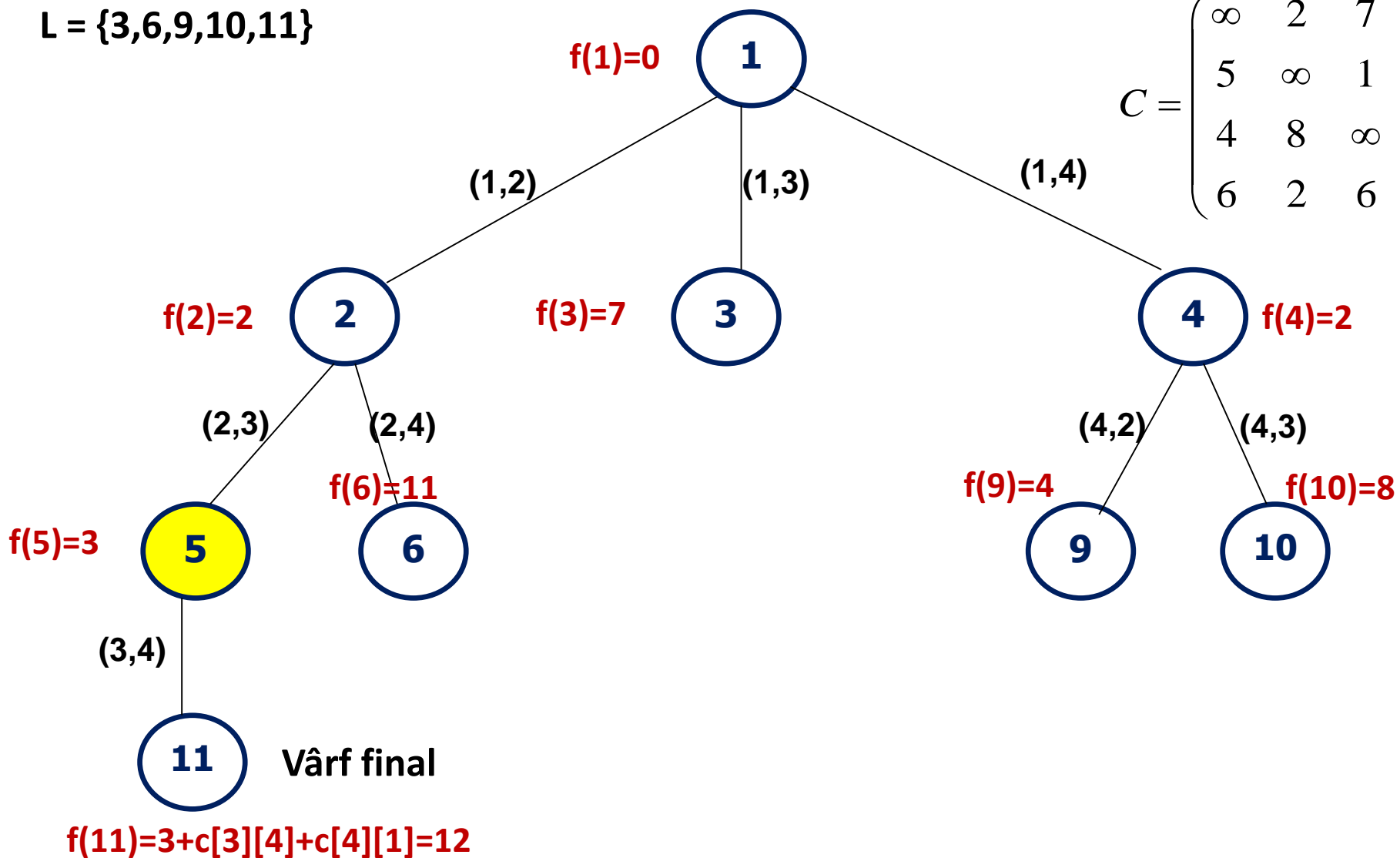
$f(9)=4$

$f(10)=8$



$L = \{3, 6, 9, 10, 11\}$

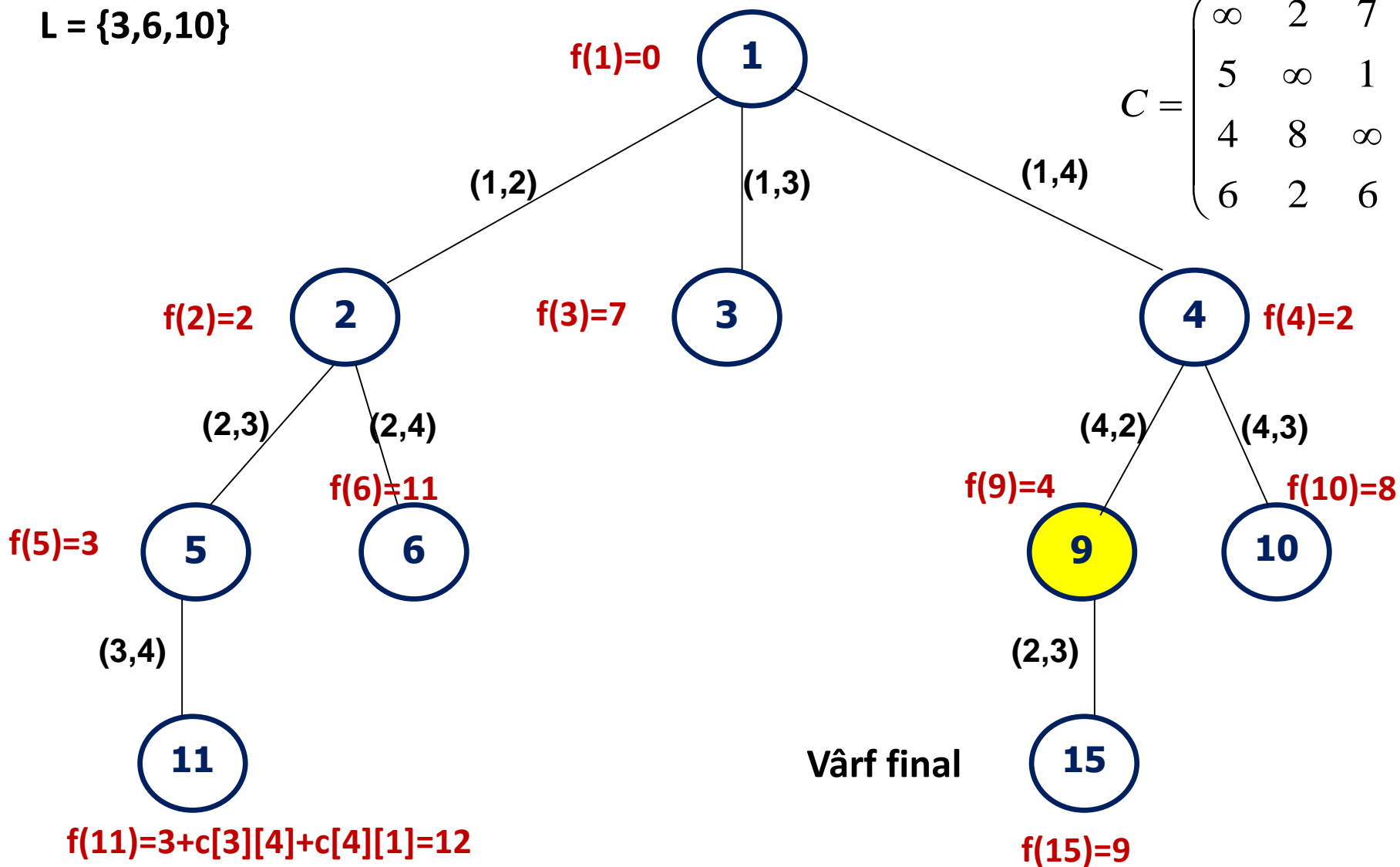
$$C = \begin{pmatrix} \infty & 2 & 7 & 2 \\ 5 & \infty & 1 & 9 \\ 4 & 8 & \infty & 3 \\ 6 & 2 & 6 & \infty \end{pmatrix}$$



lim=12 - > eliminăm vârfurile cu f mai mare decât 12

$L = \{3, 6, 10\}$

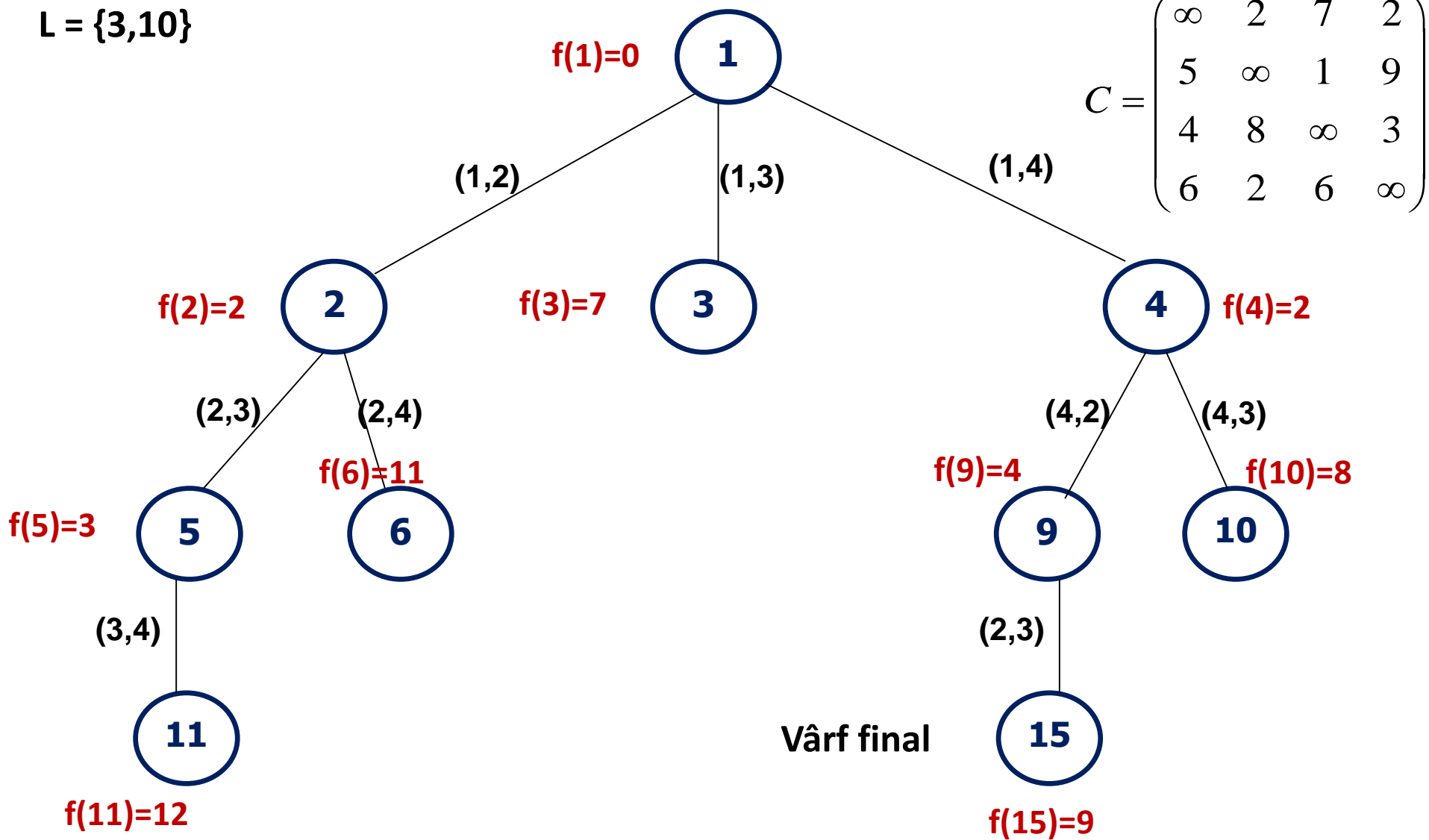
$$C = \begin{pmatrix} \infty & 2 & 7 & 2 \\ 5 & \infty & 1 & 9 \\ 4 & 8 & \infty & 3 \\ 6 & 2 & 6 & \infty \end{pmatrix}$$



$\lim = \min\{12, 9\} = 9 \rightarrow$ eliminăm vârfurile cu f mai mare decât 9

$L = \{3, 10\}$

$$C = \begin{pmatrix} \infty & 2 & 7 & 2 \\ 5 & \infty & 1 & 9 \\ 4 & 8 & \infty & 3 \\ 6 & 2 & 6 & \infty \end{pmatrix}$$

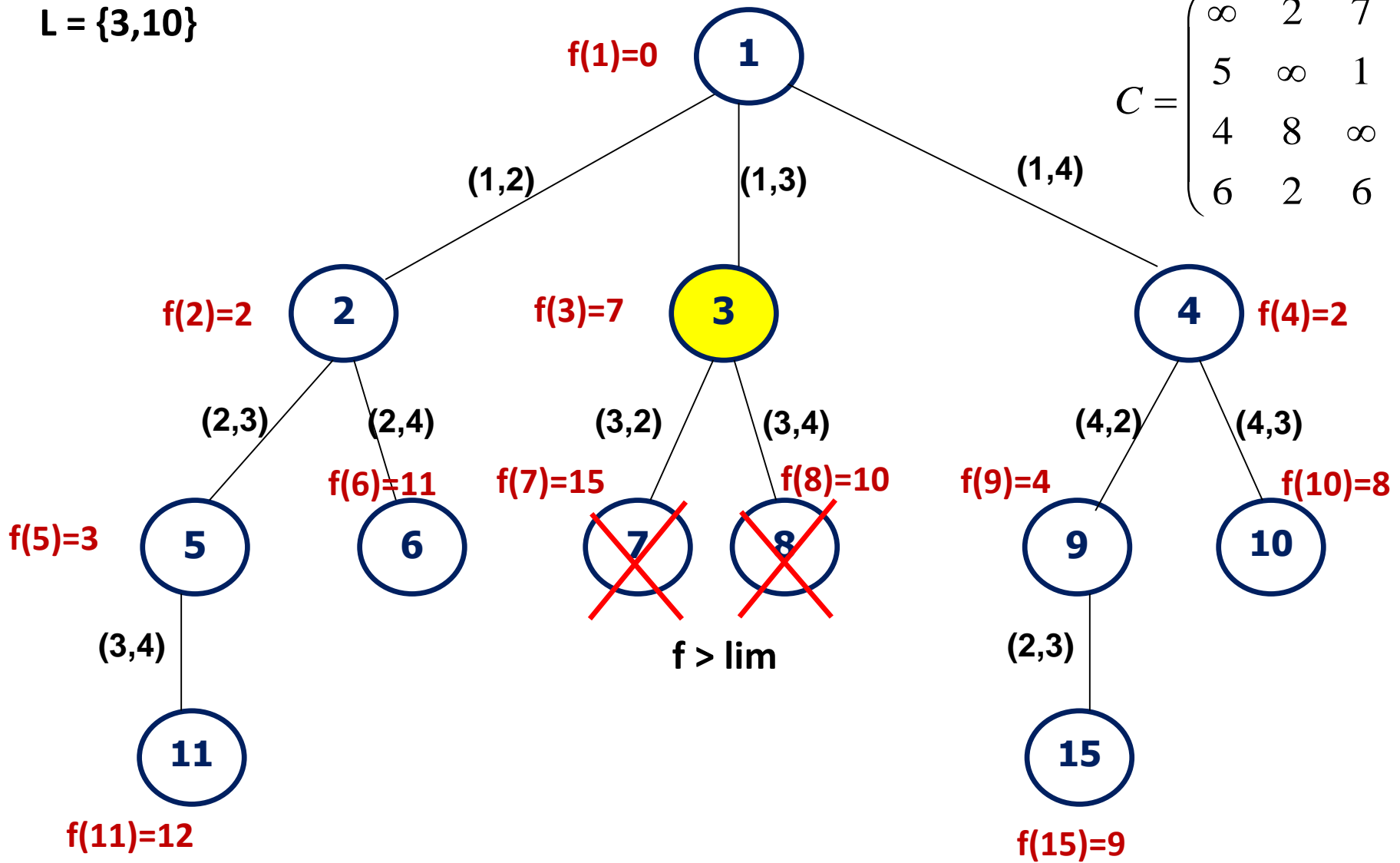


lim= 9

$L = \{3,10\}$

$C =$

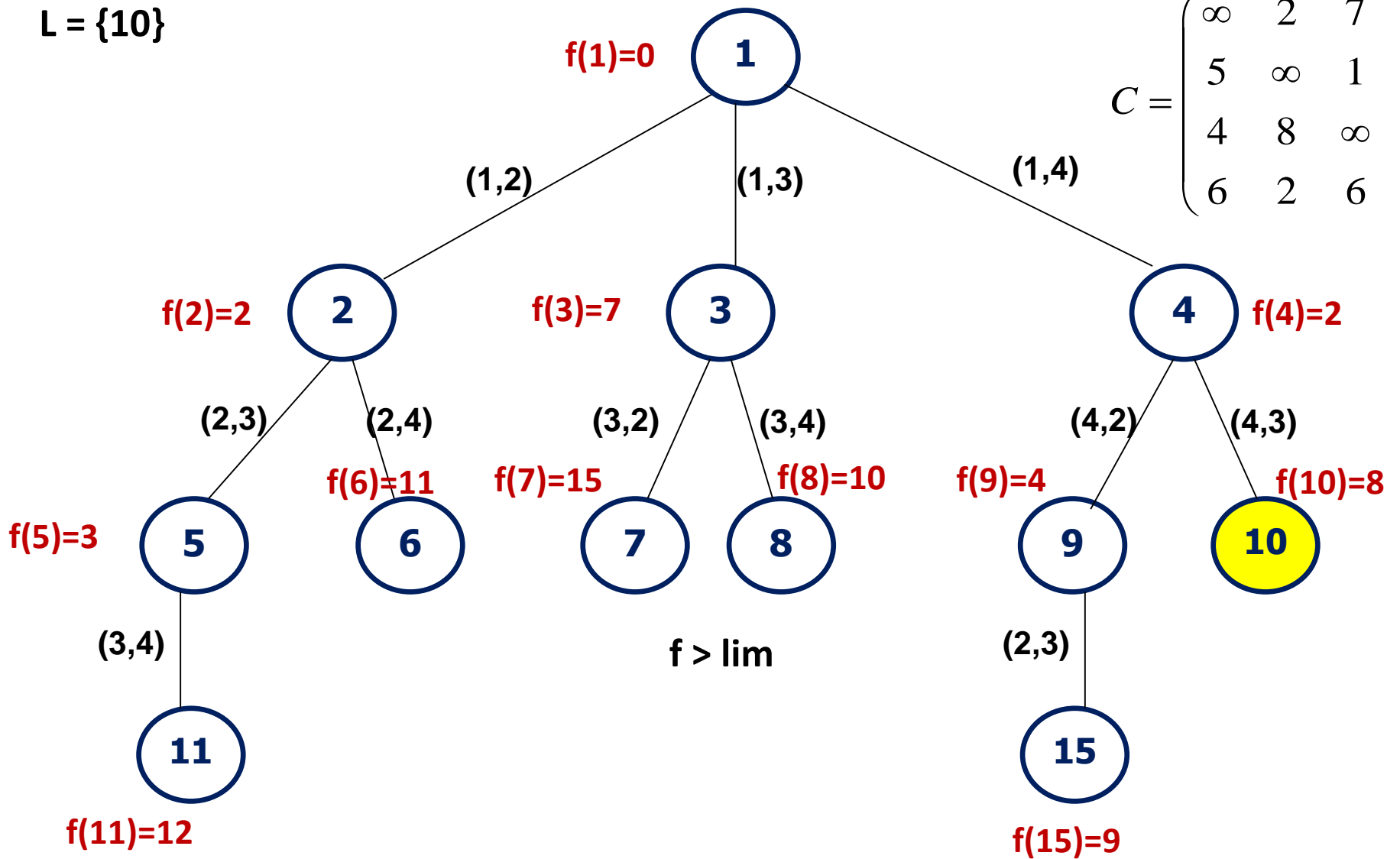
∞	2	7	2
5	∞	1	9
4	8	∞	3
6	2	6	∞



$f > \text{lim}$

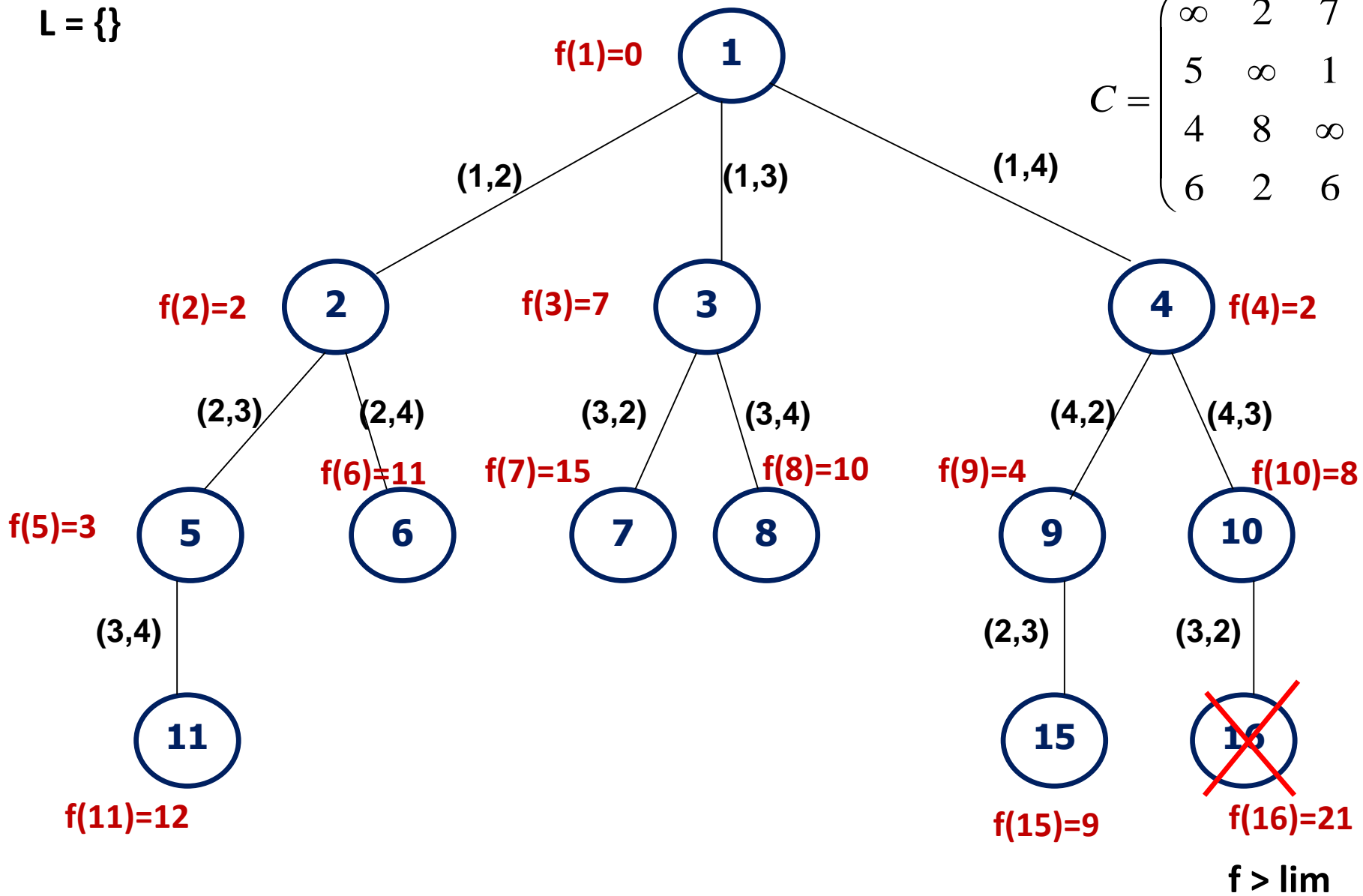
$L = \{10\}$

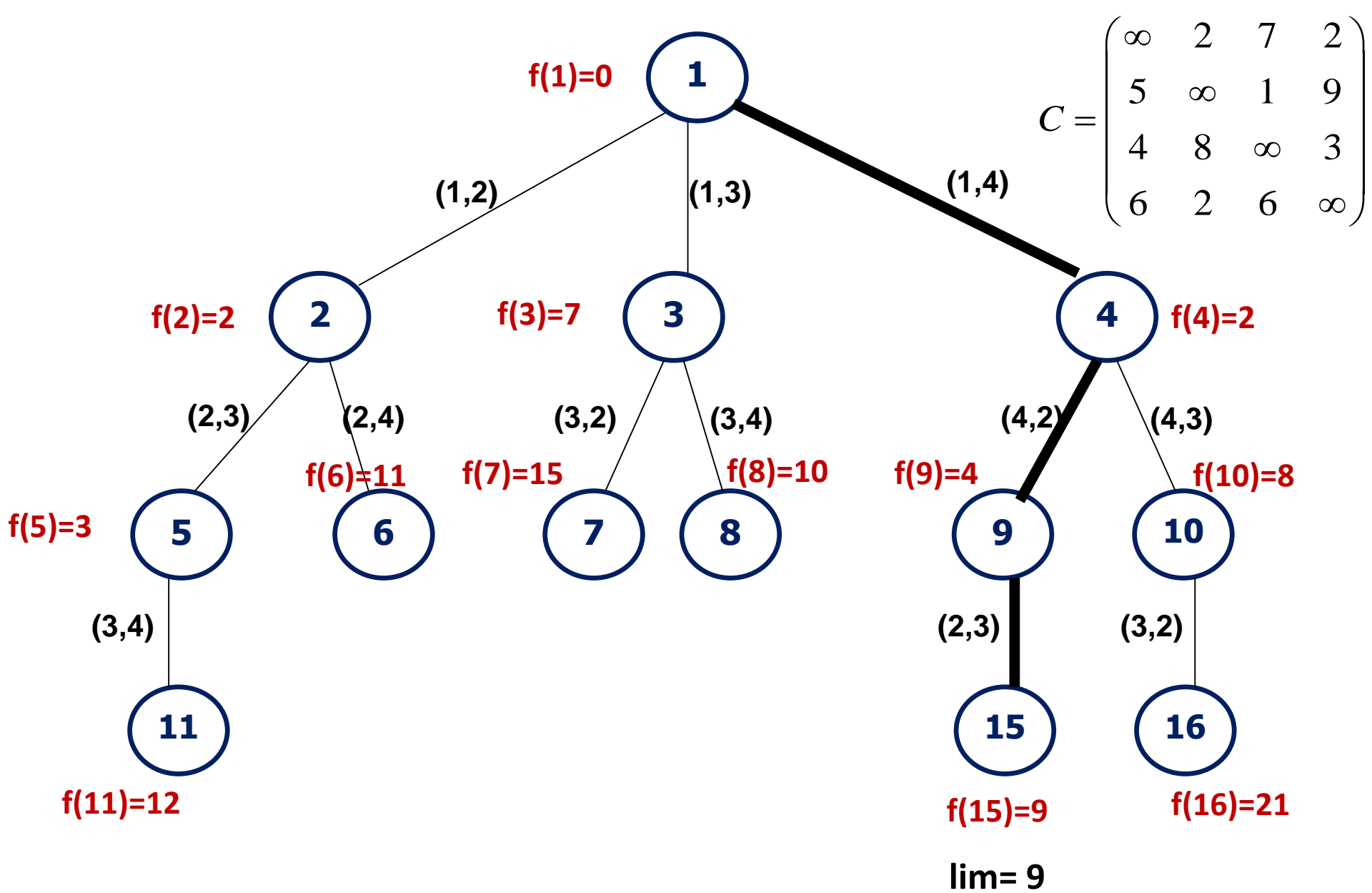
$$C = \begin{pmatrix} \infty & 2 & 7 & 2 \\ 5 & \infty & 1 & 9 \\ 4 & 8 & \infty & 3 \\ 6 & 2 & 6 & \infty \end{pmatrix}$$

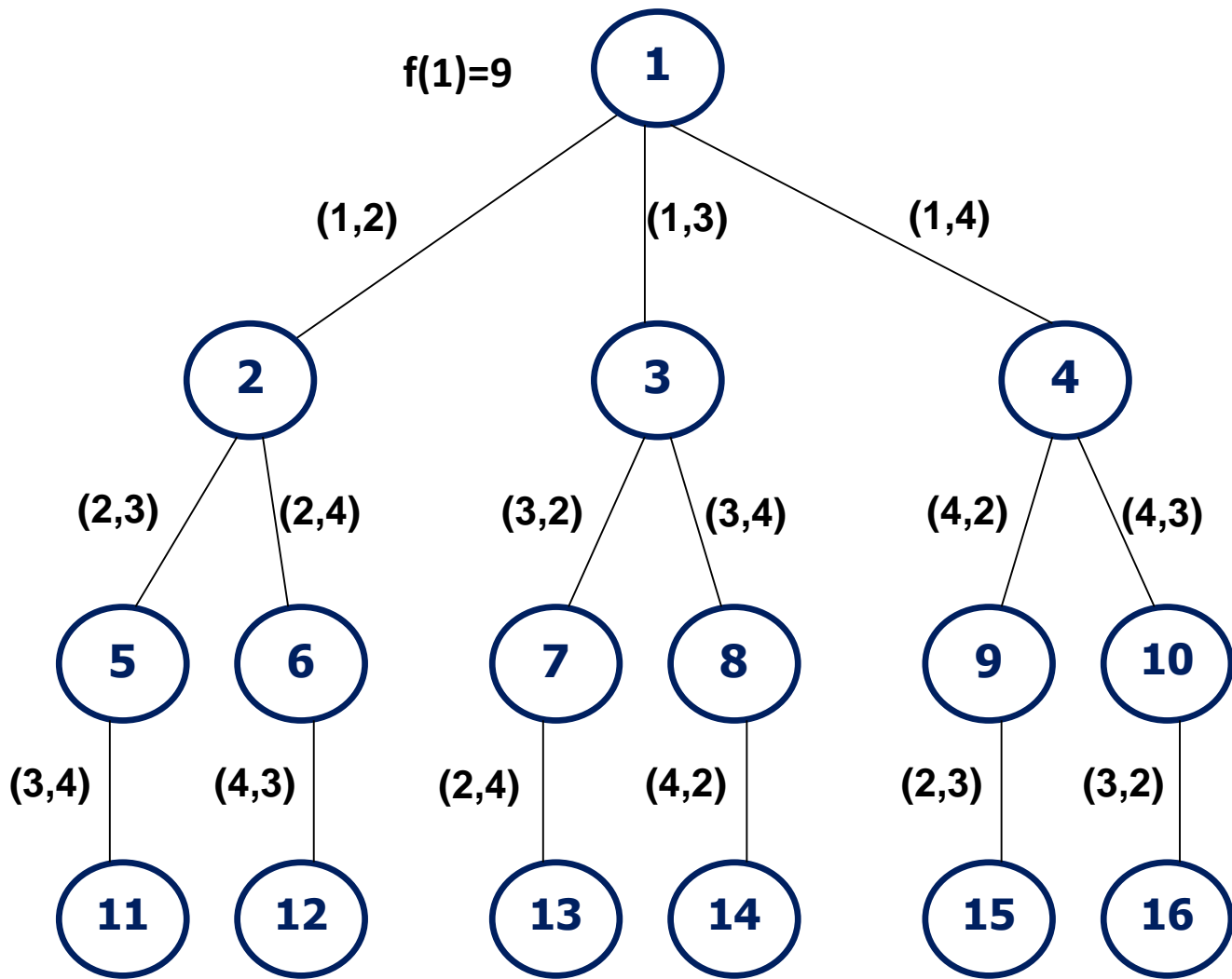


$L = \{\}$

$$C = \begin{pmatrix} \infty & 2 & 7 & 2 \\ 5 & \infty & 1 & 9 \\ 4 & 8 & \infty & 3 \\ 6 & 2 & 6 & \infty \end{pmatrix}$$







Au fost excluse puține vârfuri de la expandare

Exemplu – circuit hamiltonian minim TSP

Euristici h mai precise:

□ **cost (TSP) \geq**

$$\left(\frac{1}{2} \sum_{v \in V} \text{arc minim care intra in } v + \sum_{v \in V} \text{arc minim care iese din } v \right)$$

□ Pentru o configurație care deja conține anumite arce - se actualizează limita inferioară

□ Pentru gestionare – se modifică succesiv matricea costurilor

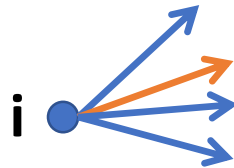
Circuit hamiltonian minim TSP

- **Observația 1.** Dacă micșorăm toate elementele unei linii i din matricea de costuri C cu α , orice circuit hamiltonian va avea costul micșorat cu α

De ce?

Circuit hamiltonian minim TSP

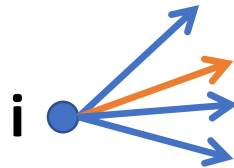
- **Observația 1.** Dacă micșorăm toate elementele unei linii i din matricea de costuri C cu α , orice circuit hamiltonian va avea costul micșorat cu α



Un circuit conține un unic arc care iese din i

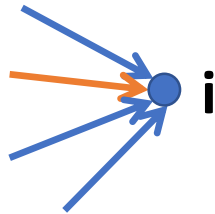
Circuit hamiltonian minim TSP

- **Observația 1.** Dacă micșorăm toate elementele unei linii i din matricea de costuri C cu α , orice circuit hamiltonian va avea costul micșorat cu α



Circuit hamiltonian minim TSP

- **Observația 2.** Dacă micșorăm toate elementele unei coloane din matricea de costuri C cu α , orice circuit hamiltonian va avea costul micșorat cu α



Circuit hamiltonian minim TSP

- **matrice de costuri reduse** = în care pe orice linie sau coloană apare cel puțin un zero, exceptând cazul când linia sau coloana conține numai ∞
- $f(\text{rad})$ = cantitatea (totală) cu care se reduce matricea de cost C
= limită inferioară pentru costul CHM

Circuit hamiltonian minim TSP

$$C = \begin{pmatrix} \infty & 2 & 7 & 2 \\ 5 & \infty & 1 & 9 \\ 4 & 8 & \infty & 3 \\ 6 & 2 & 6 & \infty \end{pmatrix}$$

Reducem linia 1 cu 2

$$\begin{pmatrix} \infty & 0 & 5 & 0 \\ 5 & \infty & 1 & 9 \\ 4 & 8 & \infty & 3 \\ 6 & 2 & 6 & \infty \end{pmatrix}$$

Reducem linia 1 cu 1

$$\begin{pmatrix} \infty & 0 & 5 & 0 \\ 4 & \infty & 0 & 8 \\ 4 & 8 & \infty & 3 \\ 6 & 2 & 6 & \infty \end{pmatrix}$$

Reducem linia 3 cu 3

$$\begin{pmatrix} \infty & 0 & 5 & 0 \\ 4 & \infty & 0 & 8 \\ 1 & 5 & \infty & 0 \\ 6 & 2 & 6 & \infty \end{pmatrix}$$

Reducem linia 4 cu 2

$$\begin{pmatrix} \infty & 0 & 5 & 0 \\ 4 & \infty & 0 & 8 \\ 1 & 5 & \infty & 0 \\ 4 & 0 & 4 & \infty \end{pmatrix}$$

Reducem coloana 1 cu 1

$$\begin{pmatrix} \infty & 0 & 5 & 0 \\ 3 & \infty & 0 & 8 \\ 0 & 5 & \infty & 0 \\ 3 & 0 & 4 & \infty \end{pmatrix}$$

Circuit hamiltonian minim TSP

$$C = \begin{pmatrix} \infty & 3 & 7 & 2 \\ 5 & \infty & 1 & 9 \\ 4 & 8 & \infty & 3 \\ 6 & 2 & 6 & \infty \end{pmatrix} \quad \text{matricea redusă} = \begin{pmatrix} \infty & 0 & 5 & 0 \\ 3 & \infty & 0 & 8 \\ 0 & 5 & \infty & 0 \\ 3 & 0 & 4 & \infty \end{pmatrix}$$

Cantitatea totală cu care s-a redus matricea =

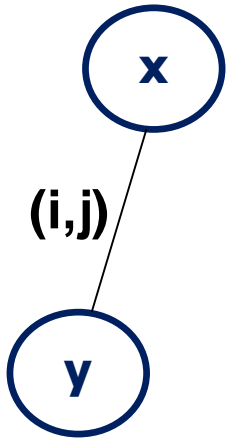
$$2 + 1 + 3 + 2 + 1 = 9$$

Algoritmul Branch and Bound pentru TSP

- ▶ $f(\text{rad})$ = cantitatea totală cu care s-a redus matricea C
- ▶ Unui vârf x îi asociem
 - o matrice de costuri redusă M_x (dacă nu este frunză)
 - o valoare $f(x)$ calculată după cum urmează

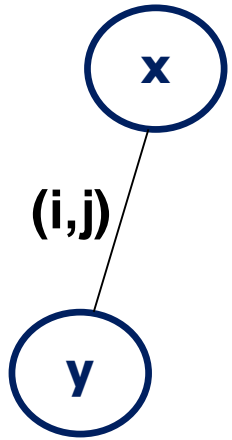
Algoritmul Branch and Bound pentru TSP

- ▶ Pentru un vârf y din arborele BB al cărui tată este x și muchia (x,y) este etichetată cu (i,j) modificăm M_x :



Algoritmul Branch and Bound pentru TSP

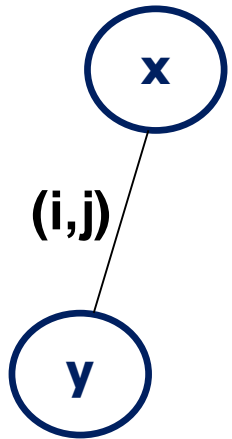
- ▶ Pentru un vârf y din arborele BB al cărui tată este x și muchia (x,y) este etichetată cu (i,j) modificăm M_x :



- elementele liniei i devin ∞ (mergem sigur către j)
- elementele coloanei j devin ∞ (am ajuns sigur în j)
- $M_x(j,1) \leftarrow \infty$ pentru a nu reveni prematur în rădăcina 1;

Algoritmul Branch and Bound pentru TSP

- ▶ Pentru un vârf y din arborele BB al cărui tată este x și muchia (x,y) este etichetată cu (i,j) modificăm M_x :



- elementele liniei i devin ∞ (mergem sigur către j)
- elementele coloanei j devin ∞ (am ajuns sigur în j)
- $M_x(j,1) \leftarrow \infty$ pentru a nu reveni prematur în rădăcina 1;
- reducem noua matrice M_x și obținem M_y ;
fie r cantitatea cu care s-a redus M_x .
- Luăm $f(y) \leftarrow f(x) + M_x(i,j) + r$.

Algoritmul Branch and Bound pentru TSP

- ▶ Avem

$$\mathbf{f}(\mathbf{x}) \leq \text{cost CHM corespunzător lui } \mathbf{x}$$

- ▶ Dacă \mathbf{x} este frunză avem

$$f(\mathbf{x}) = c(\mathbf{x}) = \text{cost CHM}$$

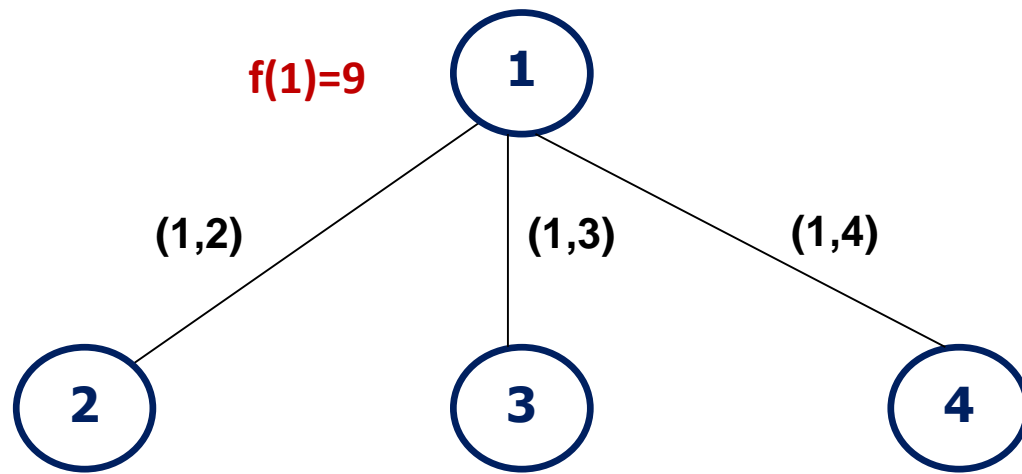
$$C = \begin{pmatrix} \infty & 2 & 7 & 2 \\ 5 & \infty & 1 & 9 \\ 4 & 8 & \infty & 3 \\ 6 & 2 & 6 & \infty \end{pmatrix}$$

$$M_1 = \begin{pmatrix} \infty & 0 & 5 & 0 \\ 3 & \infty & 0 & 8 \\ 0 & 5 & \infty & 0 \\ 3 & 0 & 4 & \infty \end{pmatrix}$$

$$f(1) = 2 + 1 + 3 + 2 + 1 = 9$$

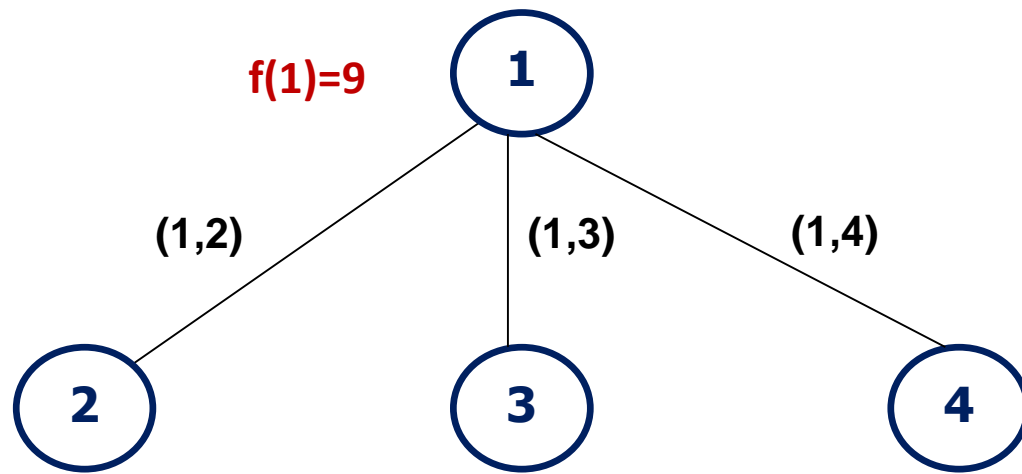
$$f(1)=9$$

1



$f(1)=9$

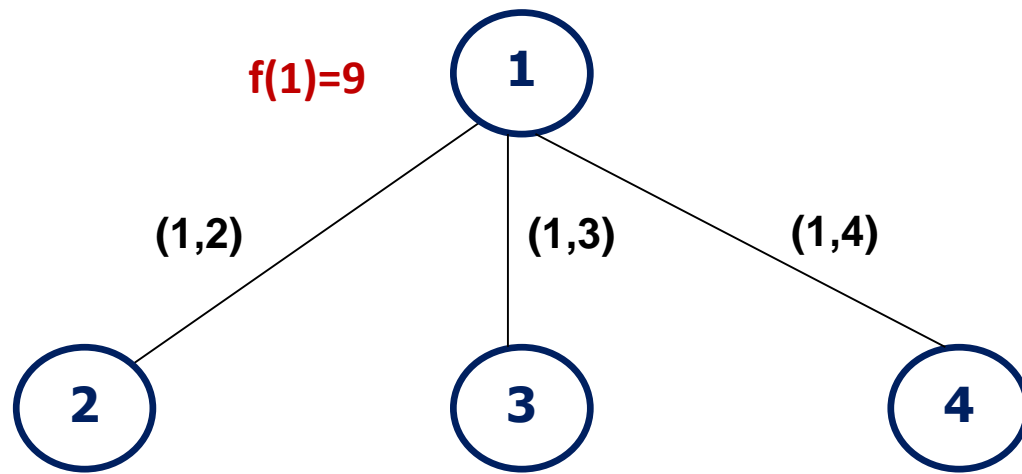
$L = \{2, 3, 4\}$



$L = \{2, 3, 4\}$

Calculăm $f(2)$

$$M_1 = \begin{pmatrix} \infty & 0 & 5 & 0 \\ 3 & \infty & 0 & 8 \\ 0 & 5 & \infty & 0 \\ 3 & 0 & 4 & \infty \end{pmatrix}$$

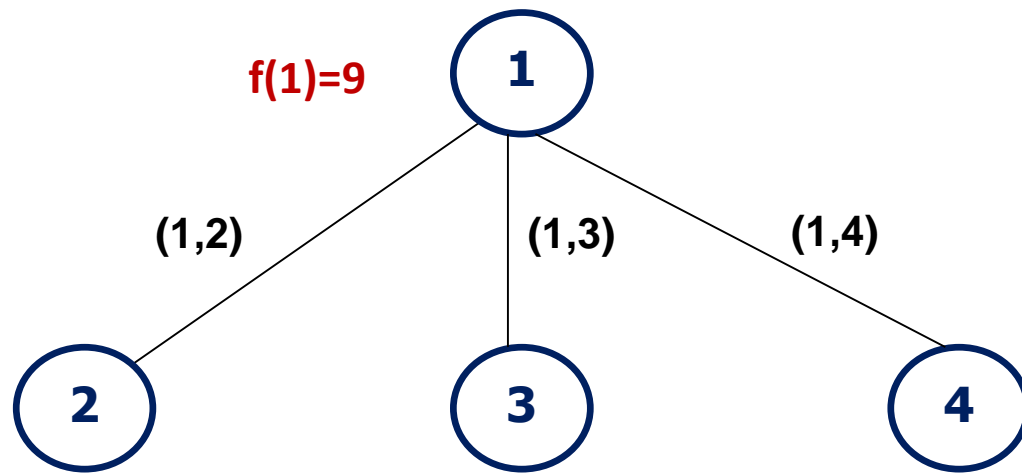


$L = \{2, 3, 4\}$

Calculăm $f(2)$

$$\begin{pmatrix} \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & 8 \\ 0 & \infty & \infty & 0 \\ 3 & \infty & 4 & \infty \end{pmatrix}$$

- Linia 1 și coloana 2 devin ∞
- Elementul (2, 1) devine ∞
- Reducem linia 4 cu 3

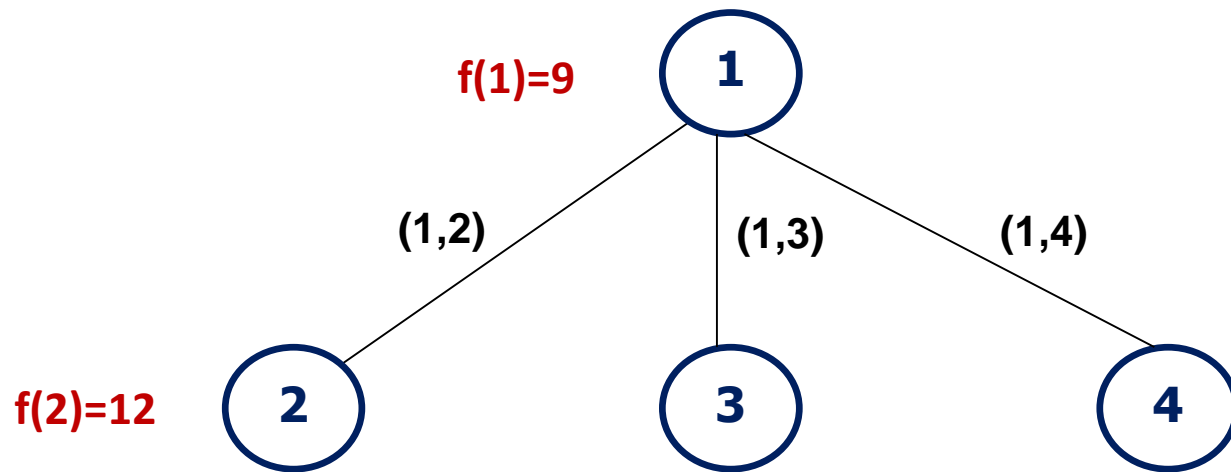


$L = \{2, 3, 4\}$

Calculăm $f(2)$

$$\begin{pmatrix} \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & 8 \\ 0 & \infty & \infty & 0 \\ 0 & \infty & 1 & \infty \end{pmatrix}$$

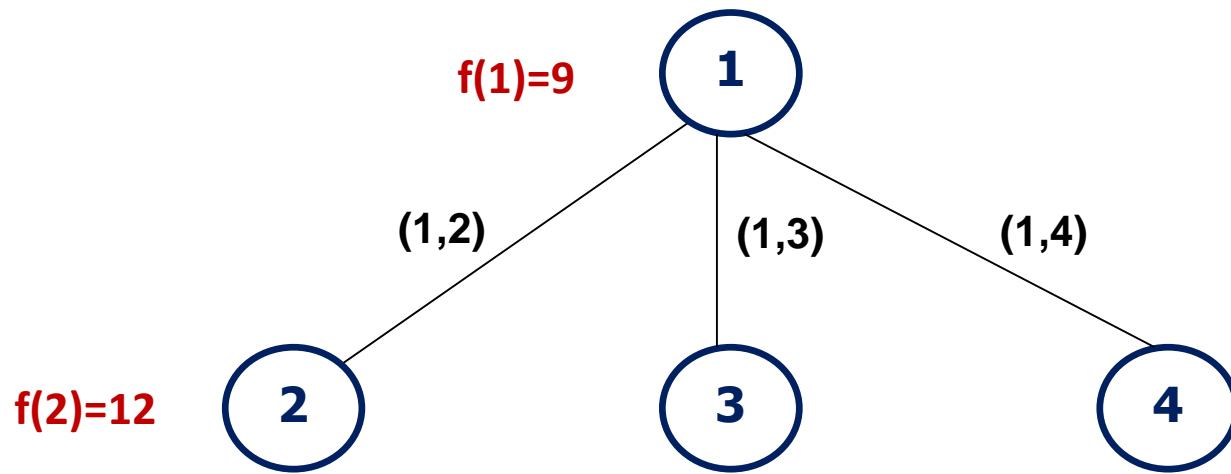
- Linia 1 și coloana 2 devin ∞
- Elementul (2, 1) devine ∞
- Reducem linia 4 cu 3



$L = \{2, 3, 4\}$

Calculăm $f(2)$

$$M_2 = \begin{pmatrix} \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & 8 \\ 0 & \infty & \infty & 0 \\ 0 & \infty & 1 & \infty \end{pmatrix} \begin{array}{l} - \text{Linia 1 și coloana 2 devin } \infty \\ - \text{Elementul (2, 1) devine } \infty \\ - \text{Reducem linia 4 cu 3} \\ - \text{Obținem } f(2) = f(1) + r + M_1(1,2) = 9 + 3 + 0 = 12 \end{array}$$

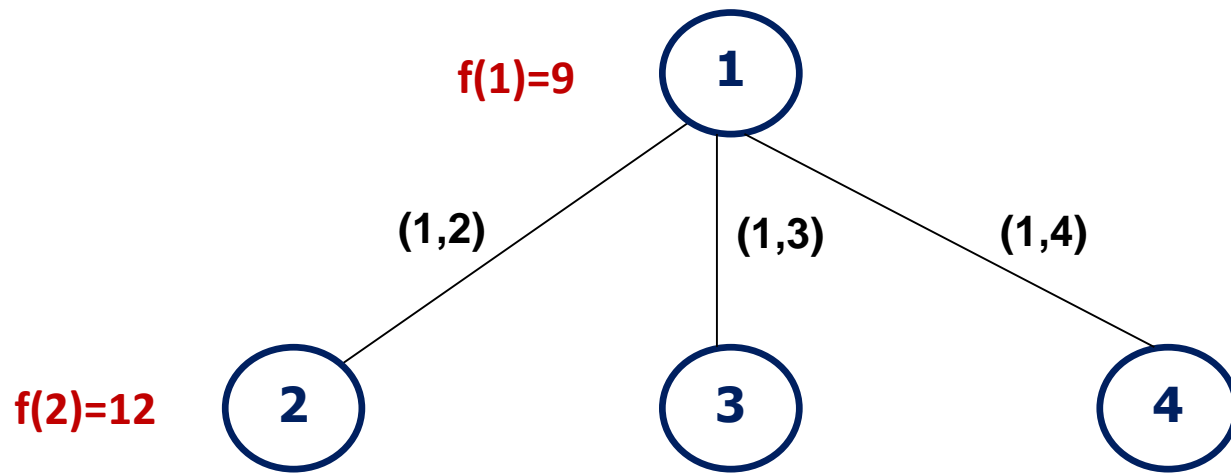


$L = \{2, 3, 4\}$

Calculăm $f(3)$

$$M_1 = \begin{pmatrix} \infty & 0 & 5 & 0 \\ 3 & \infty & 0 & 8 \\ 0 & 5 & \infty & 0 \\ 3 & 0 & 4 & \infty \end{pmatrix}$$

- Linia 1 și coloana 3 devin ∞
- Elementul (3, 1) devine ∞

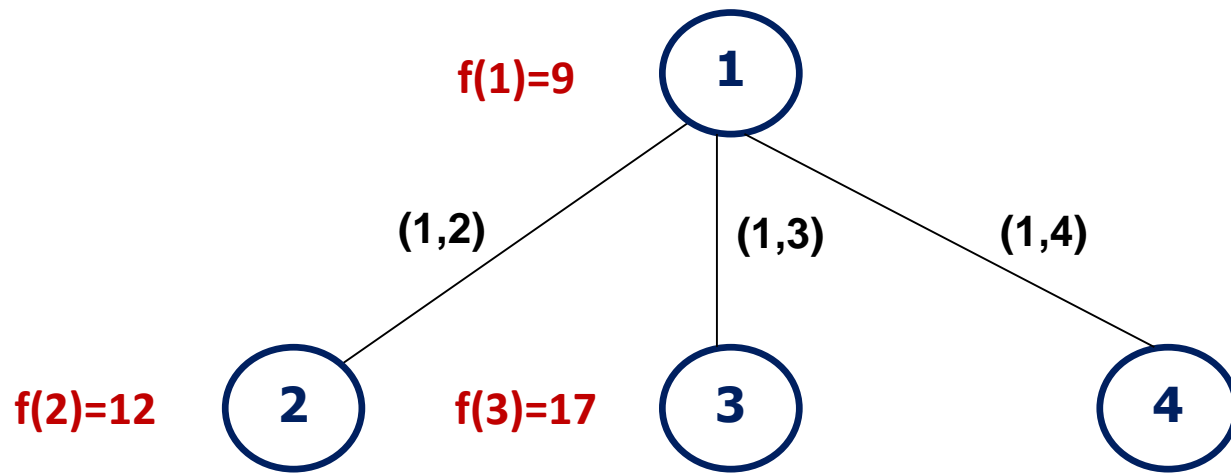


$L = \{2, 3, 4\}$

Calculăm $f(3)$

$$\begin{pmatrix} \infty & \infty & \infty & \infty \\ 3 & \infty & \infty & 8 \\ \infty & 5 & \infty & 0 \\ 3 & 0 & \infty & \infty \end{pmatrix}$$

- Linia 1 și coloana 3 devin ∞
- Elementul (3, 1) devine ∞
- Reducem linia 2 cu 3

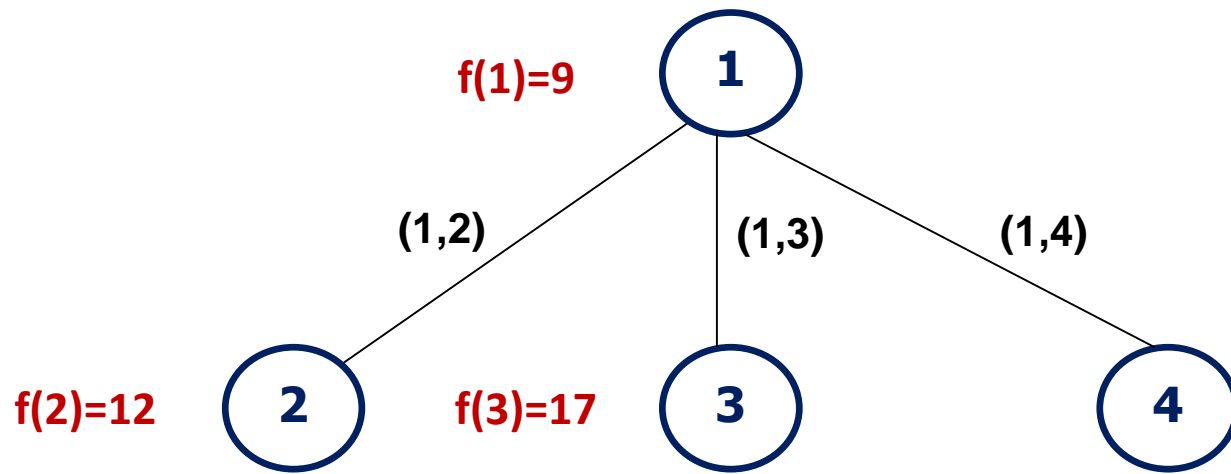


$L = \{2, 3, 4\}$

Calculăm $f(3)$

$$\begin{pmatrix} \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & 5 \\ \infty & 5 & \infty & 0 \\ 3 & 0 & \infty & \infty \end{pmatrix}$$

- Linia 1 și coloana 3 devin ∞
- Elementul (3, 1) devine ∞
- Reducem linia 2 cu 3
- Obținem $f(3) = f(1) + r + M_1(1,3) = 9 + 3 + 5 = 17$

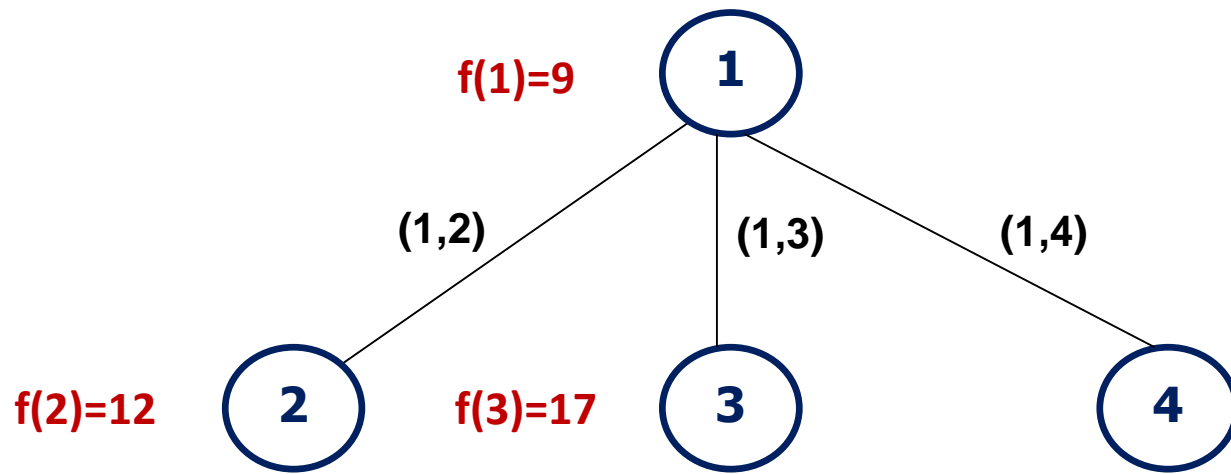


$L = \{2, 3, 4\}$

Calculăm $f(4)$

$$M_1 = \begin{pmatrix} \infty & 0 & 5 & 0 \\ 3 & \infty & 0 & 8 \\ 0 & 5 & \infty & 0 \\ 3 & 0 & 4 & \infty \end{pmatrix}$$

- Linia 1 și coloana 4 devin ∞
- Elementul (4, 1) devine ∞

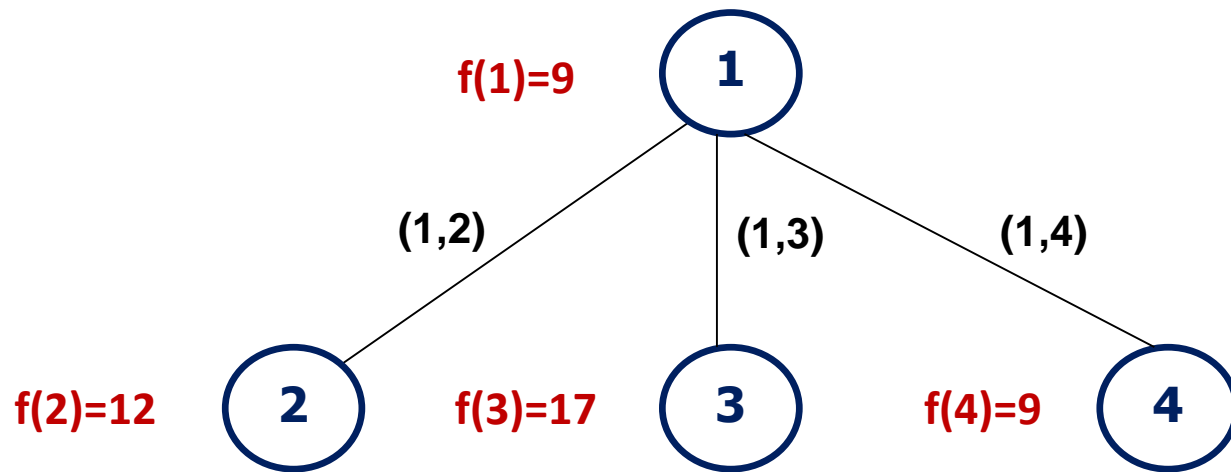


$L = \{2, 3, 4\}$

Calculăm $f(4)$

$$\begin{pmatrix} \infty & \infty & \infty & \infty \\ 3 & \infty & 0 & \infty \\ 0 & 5 & \infty & \infty \\ \infty & 0 & 4 & \infty \end{pmatrix}$$

- Linia 1 și coloana 4 devin ∞
- Elementul (4, 1) devine ∞

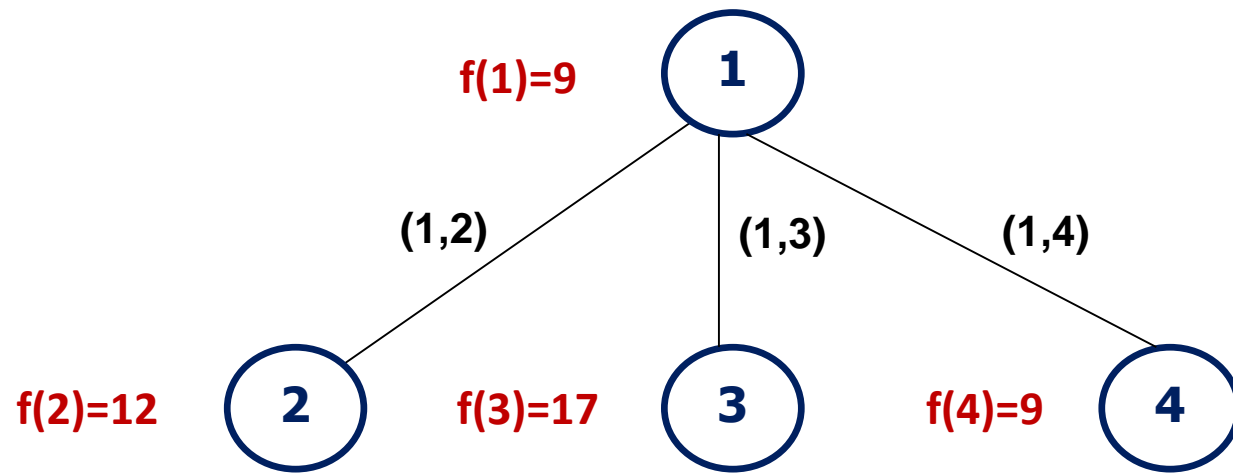


$L = \{2, 3, 4\}$

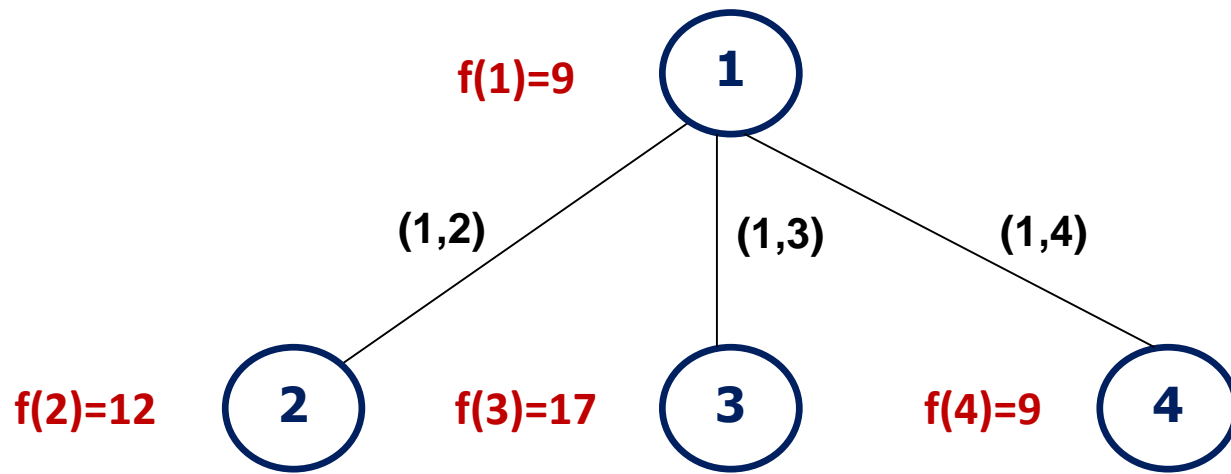
Calculăm $f(4)$

$$M_4 = \begin{pmatrix} \infty & \infty & \infty & \infty \\ 3 & \infty & 0 & \infty \\ 0 & 5 & \infty & \infty \\ \infty & 0 & 4 & \infty \end{pmatrix}$$

- Linia 1 și coloana 4 devin ∞
- Elementul (4, 1) devine ∞
- Nu sunt necesare reduceri
- Obținem $f(4) = f(1) + M_1(1,4) = 9 + 0 = 9$

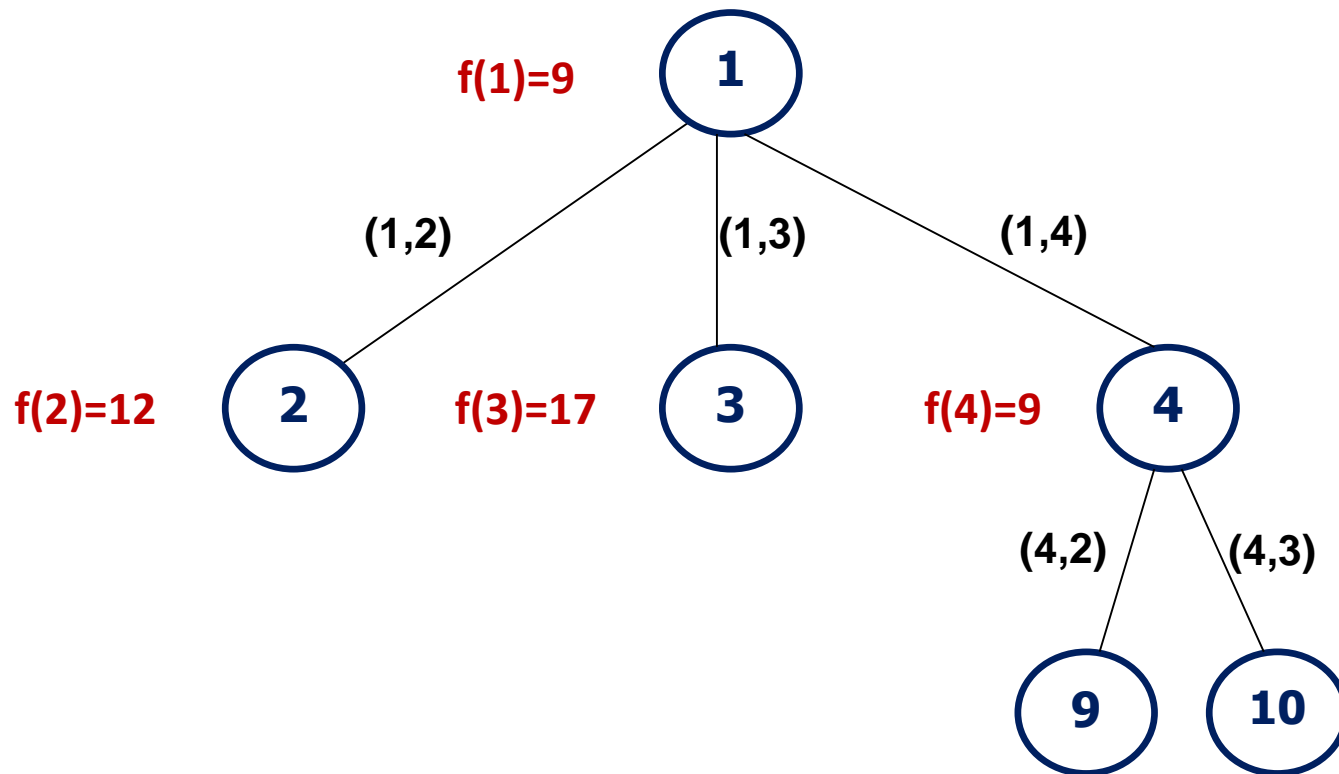


$L = \{2, 3, 4\}$



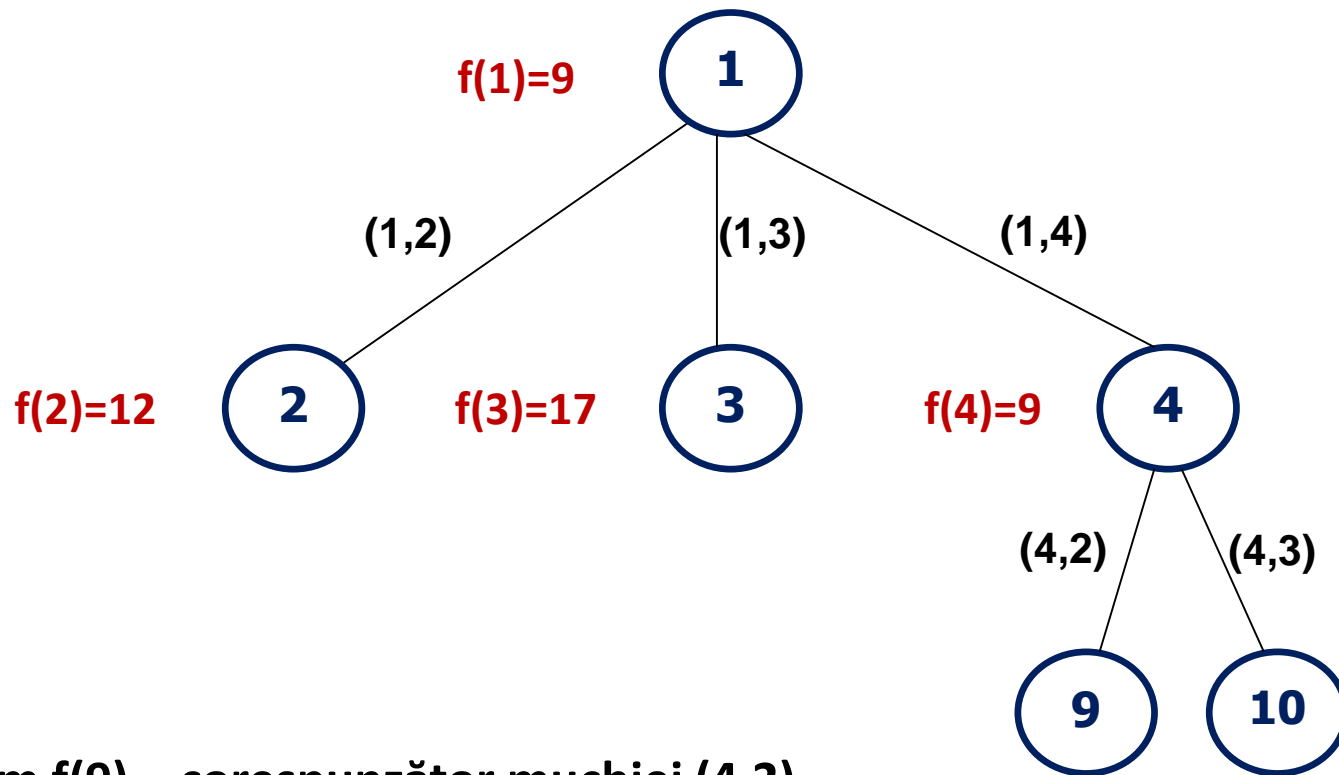
$L = \{2, 3, 4\}$

Extragem din L vârful cu f minim $\rightarrow 4$



$L = \{2, 3, 9, 10\}$

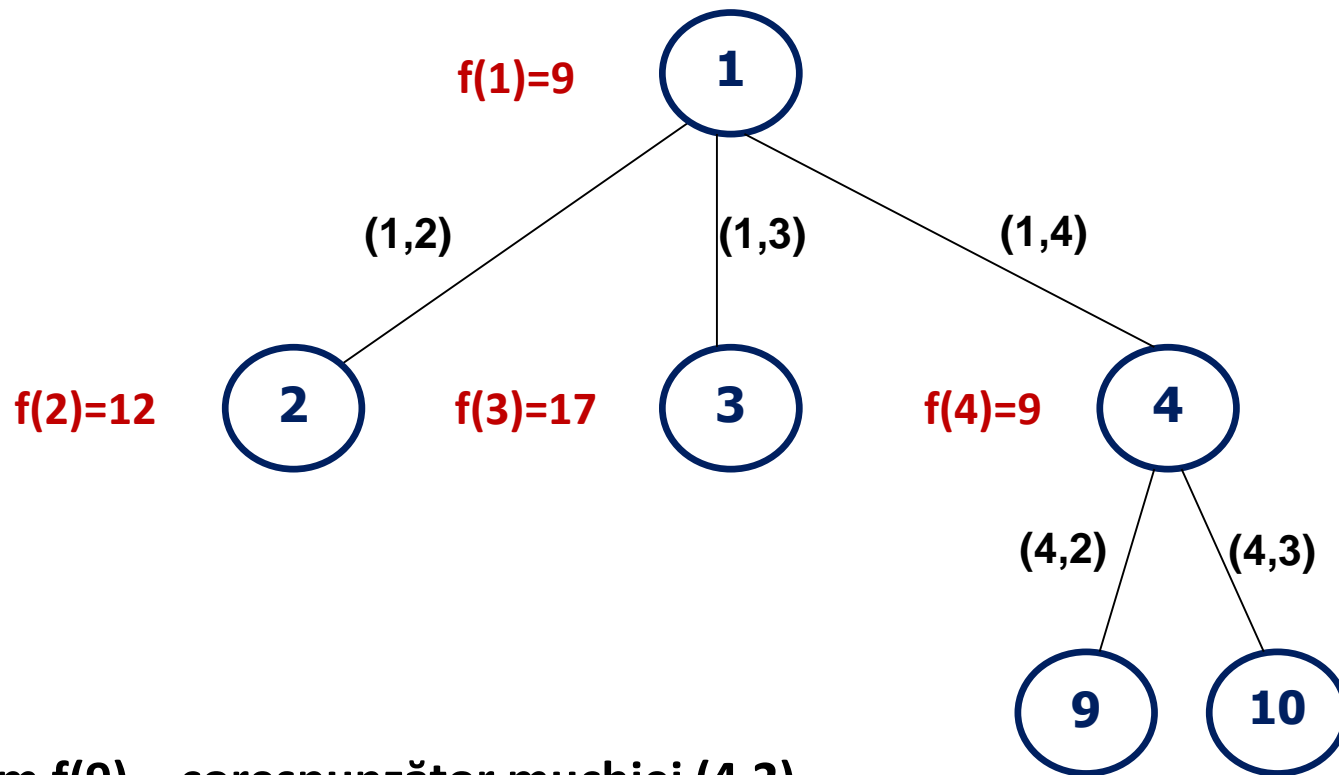
Calculăm $f(9)$ și $f(10)$ analog (pornind de la M_4)



Calculăm $f(9)$ – corespunzător muchiei (4,2)

$$M_4 = \begin{pmatrix} \infty & \infty & \infty & \infty \\ 3 & \infty & 0 & \infty \\ 0 & 5 & \infty & \infty \\ 3 & 0 & 4 & \infty \end{pmatrix}$$

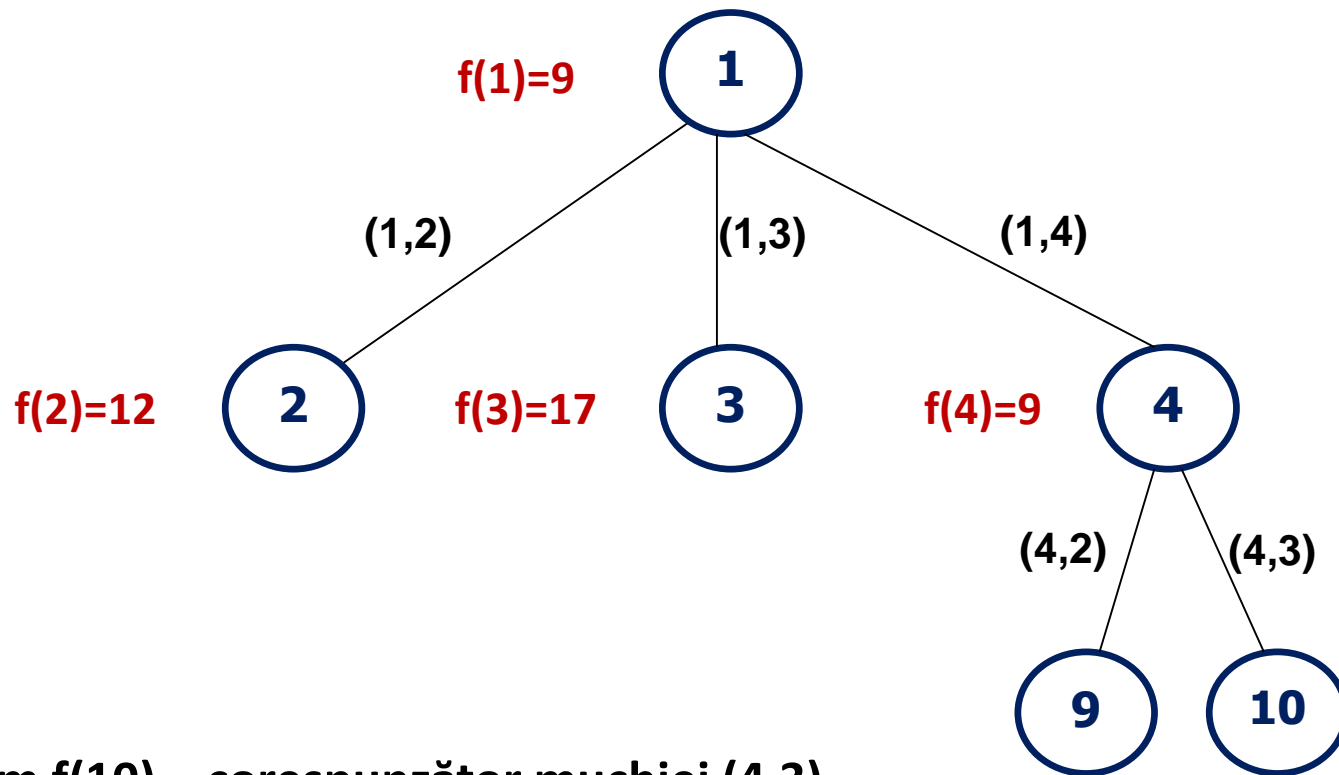
- Linia 4 și coloana 2 devin ∞
- Elementul (2, 1) devine ∞



Calculăm $f(9)$ – corespunzător muchiei (4,2)

$$M_9 = \begin{pmatrix} \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty \\ 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \end{pmatrix}$$

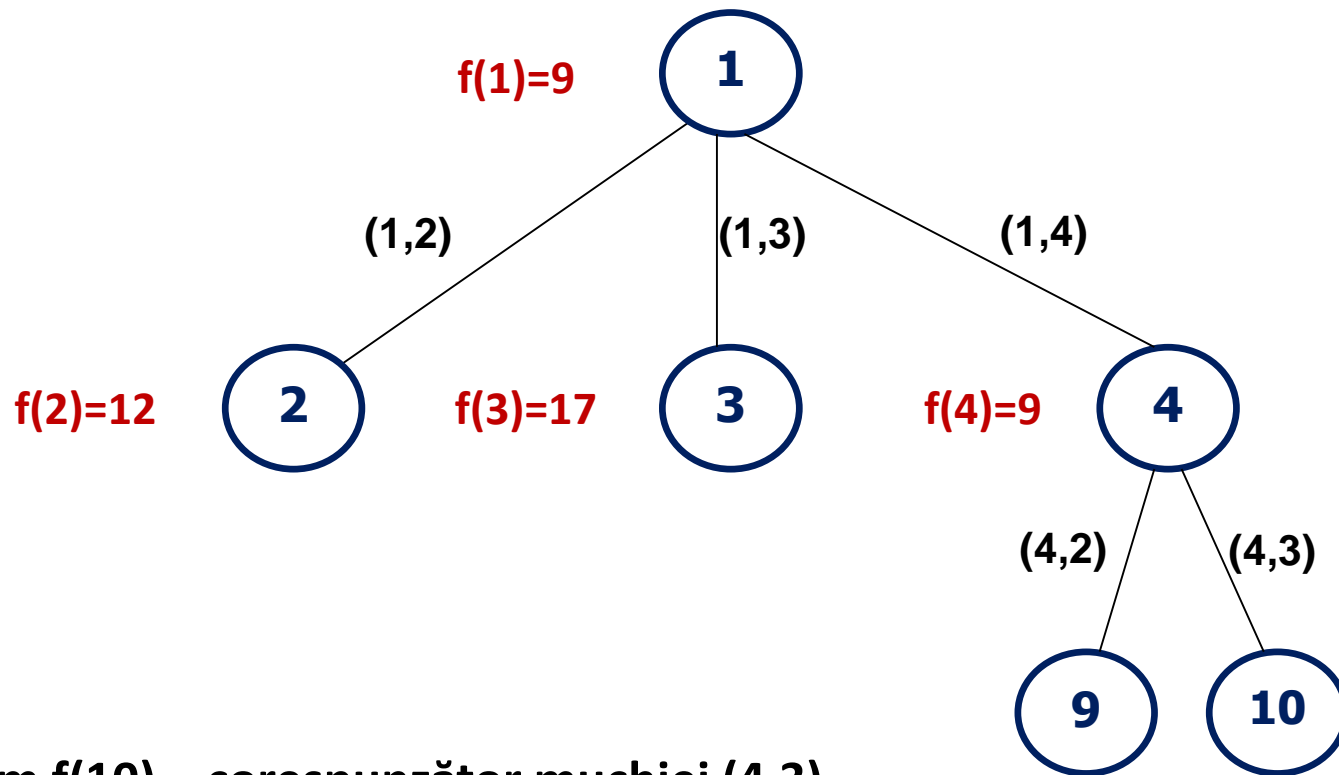
- Linia 4 și coloana 2 devin ∞
- Elementul (2, 1) devine ∞
- Nu sunt necesare reduceri
- $f(9) = f(4) + M_4(4,2) = 9 + 0 = 9$



Calculăm $f(10)$ – corespunzător muchiei (4,3)

$$M_4 = \begin{pmatrix} \infty & \infty & \infty & \infty \\ 3 & \infty & 0 & \infty \\ 0 & 5 & \infty & \infty \\ 3 & 0 & 4 & \infty \end{pmatrix}$$

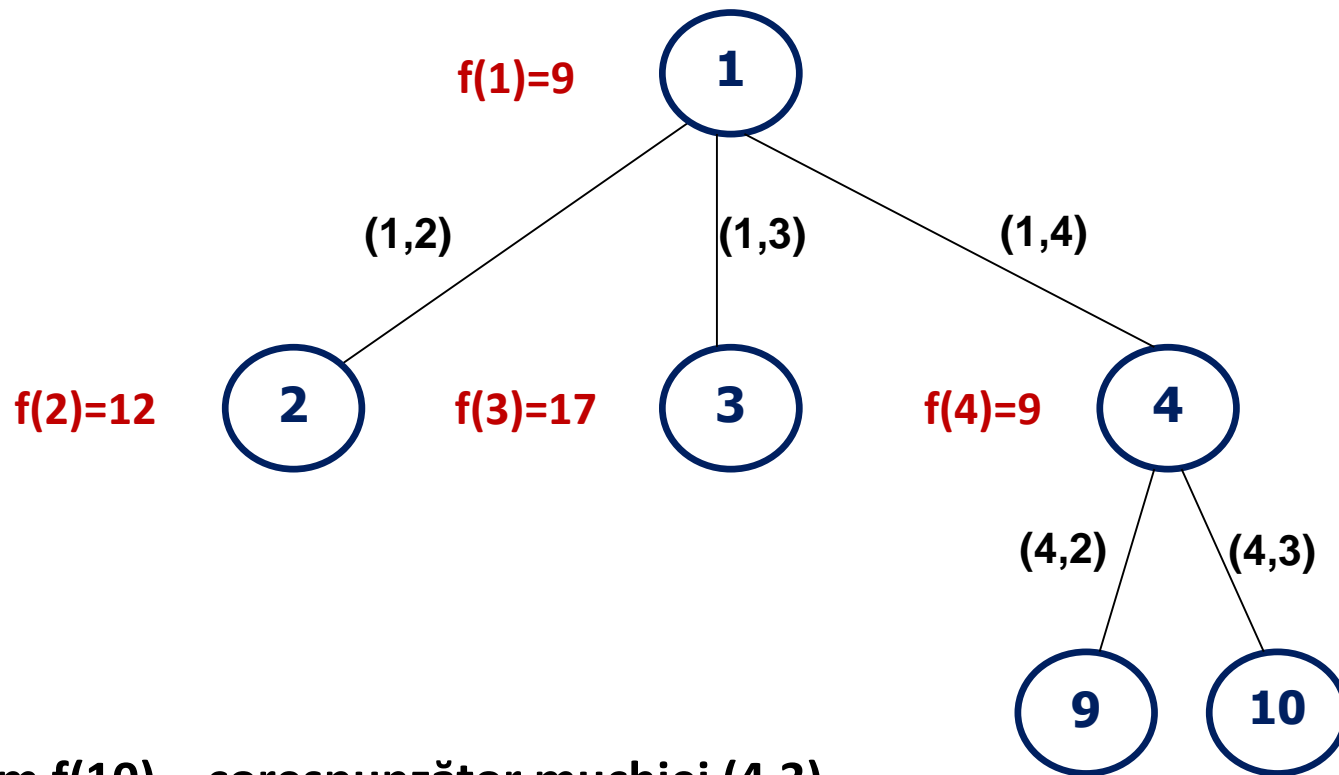
- Linia 4 și coloana 3 devin ∞
- Elementul (3, 1) devine ∞



Calculăm $f(10)$ – corespunzător muchiei (4,3)

$$\begin{pmatrix} \infty & \infty & \infty & \infty \\ 3 & \infty & \infty & \infty \\ \infty & 5 & \infty & \infty \\ \infty & \infty & \infty & \infty \end{pmatrix}$$

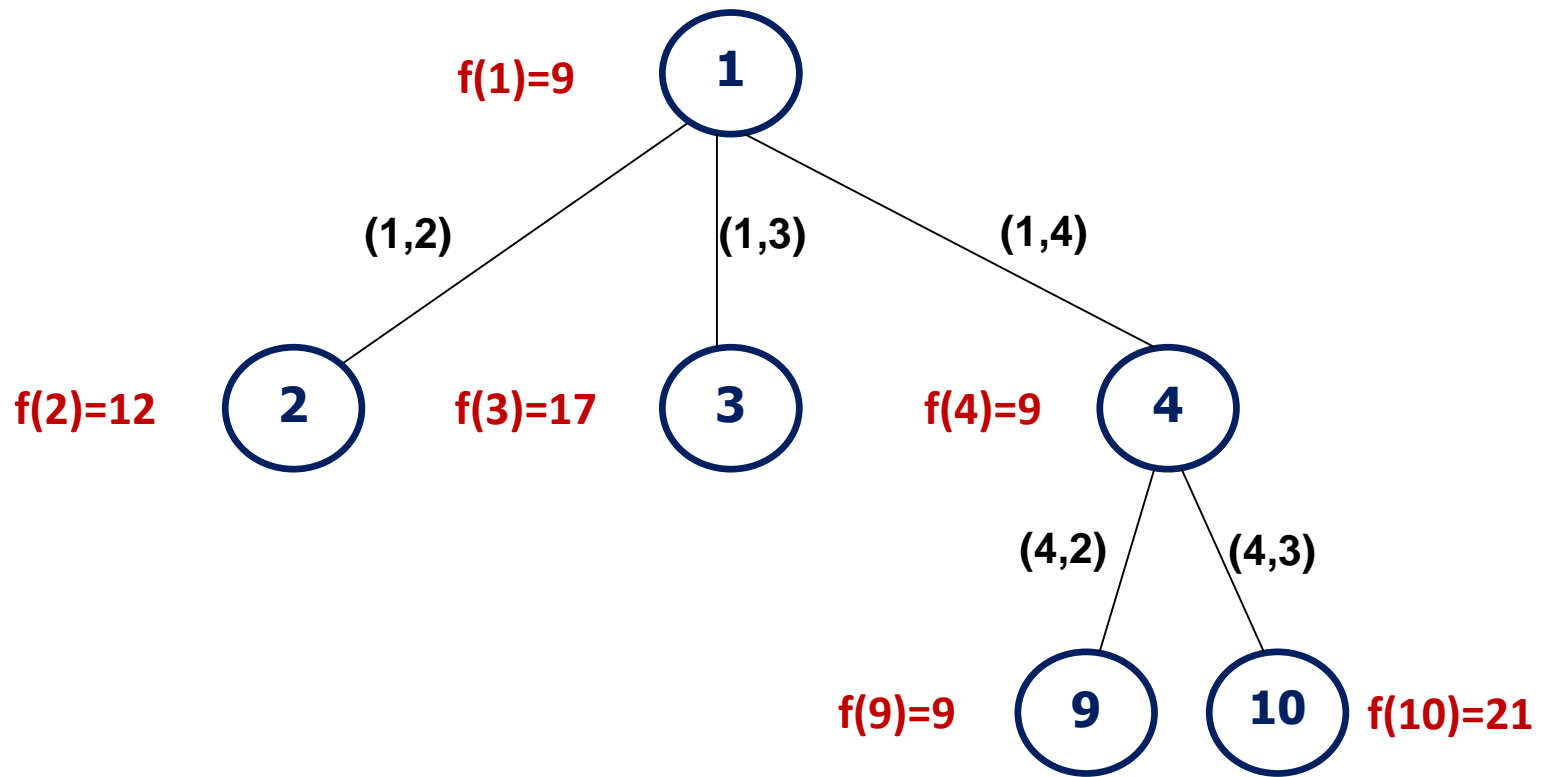
- Linia 4 și coloana 3 devin ∞
- Elementul (3, 1) devine ∞
- Reducem linia 2 cu 3 și linia 3 cu 5



Calculăm $f(10)$ – corespunzător muchiei (4,3)

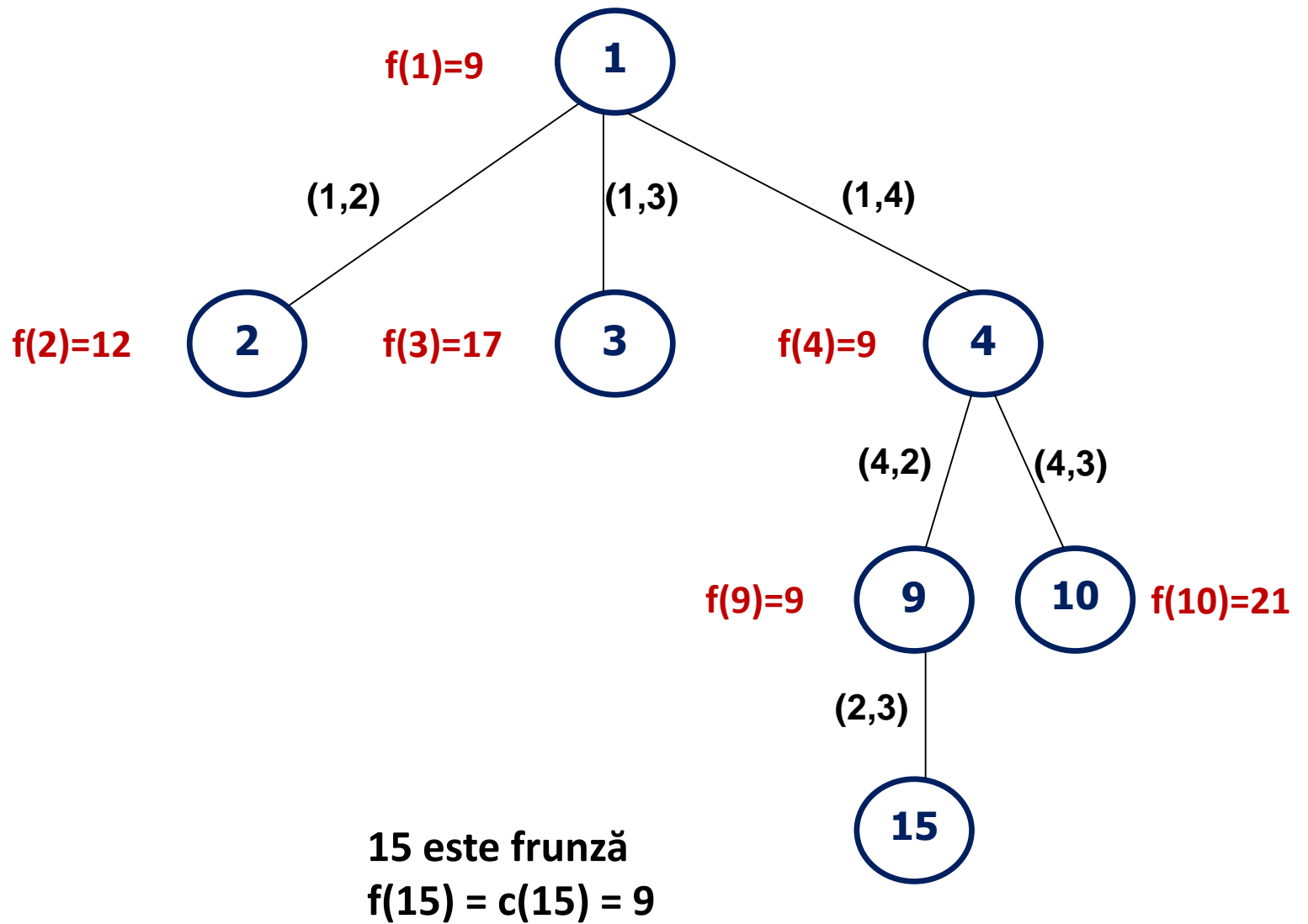
$$M_{10} = \begin{pmatrix} \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty \end{pmatrix}$$

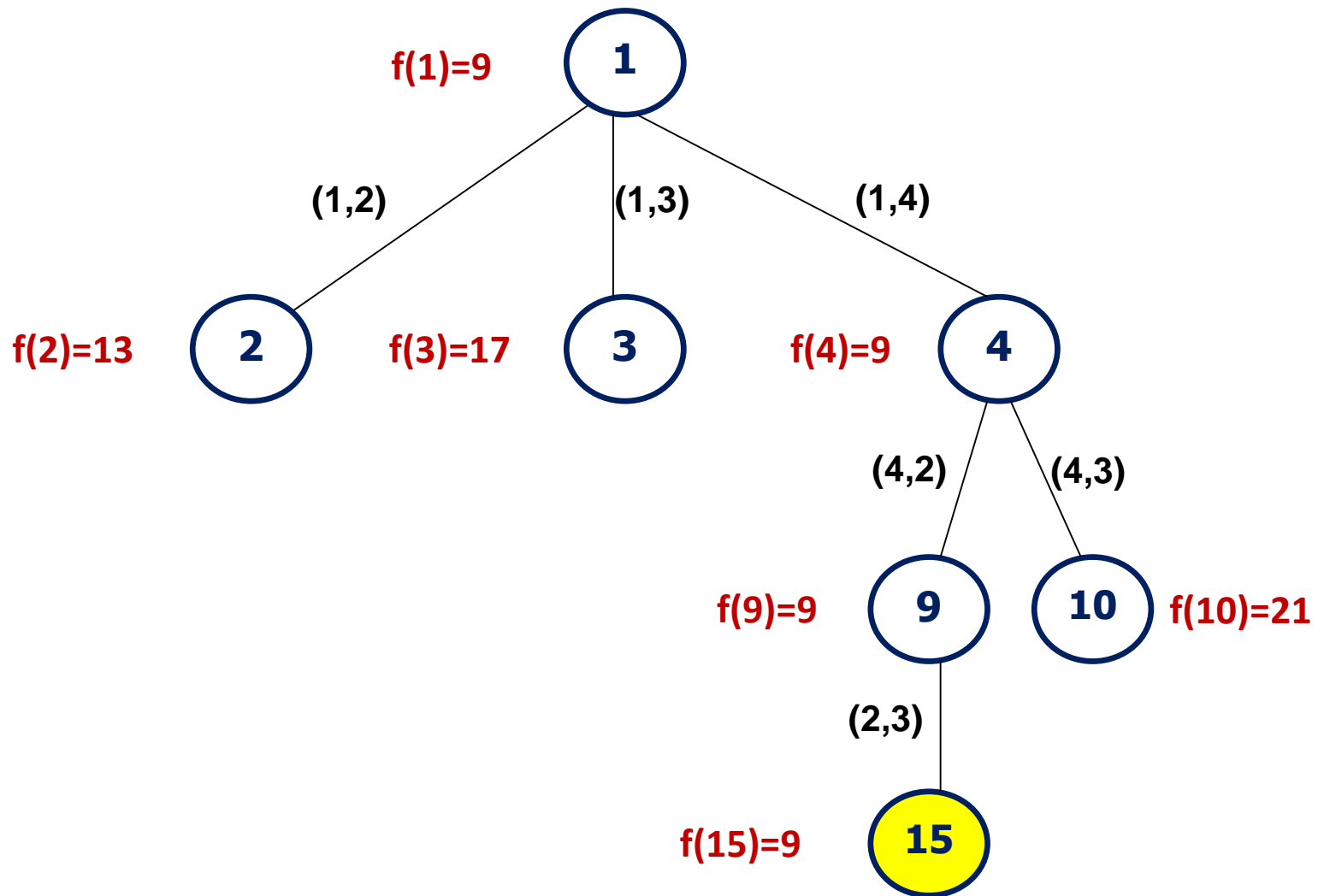
- Linia 4 și coloana 3 devin ∞
- Elementul (3, 1) devine ∞
- Reducem linia 2 cu 3 și linia 3 cu 5
- $f(10)=f(4)+r+M_4(4,3)=9+(3+5)+4=21$



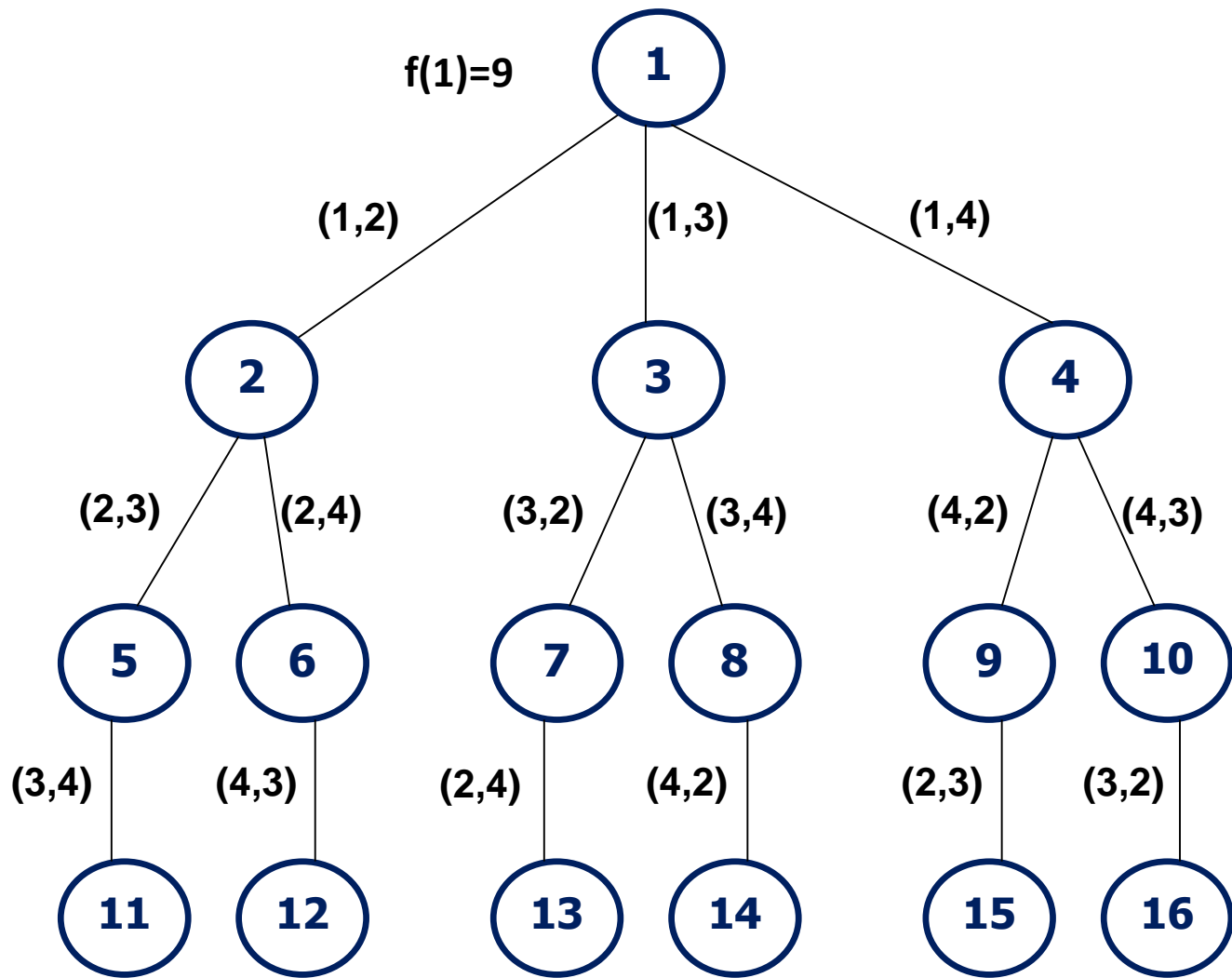
$L = \{2, 3, 9, 10\}$

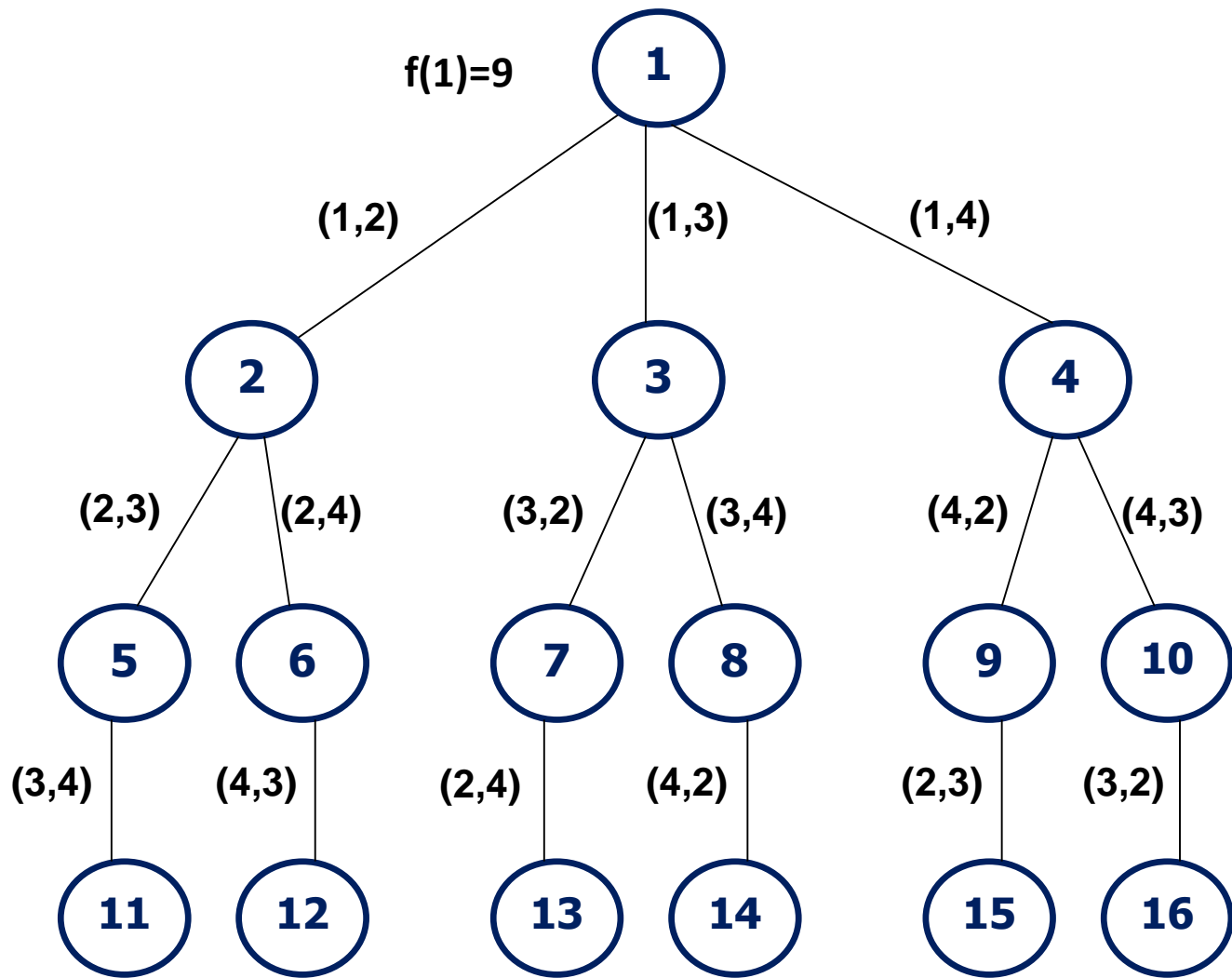
Extragem din L vârful cu f minim $\rightarrow 9$





- min devine 9
- se elimină din L toate vârfurile cu f mai mare decât 9
- L devine **vidă** -> STOP





Exemplu – circuit hamiltonian minim TSP

Alte euristici h :

□ **Pentru cazul neorientat:**

$\text{cost}(\text{TSP}) \geq \text{cost arbore parțial de cost minim}$

Exemplu – circuit hamiltonian minim TSP

Euristici h se pot obține cu algoritmi greedy.

Spre exemplu, pentru problema rucsacului.

Algoritmi Probabiliști

Algoritmi Probabiliști

- În timpul rezolvării unei probleme, putem ajunge la un moment dat în situația de a avea de ales între mai multe variante de continuare.
 - se alege aleator una dintre variante
- la executări diferite ale unui algoritm probabilist, rezultatele sunt în general diferite.

Algoritmi Probabiliști

- Categori
- Monte Carlo
- Las Vegas
- Algoritmi numerici (exemplu aproximarea lui π)

Algoritmi Monte Carlo

- Furnizează totdeauna un rezultat, care însă nu este neapărat corect.
- Probabilitatea ca rezultatul să fie corect crește pe măsură ce timpul disponibil crește.

Algoritmi Monte Carlo

- Se consideră un vector cu n elemente distincte. Să se determine un element al vectorului care să fie mai mare sau egal cu mediana a celor n numere din vector
 - n este foarte mare
 - timpul avut la dispoziție este mic

Algoritmi Monte Carlo

- $v = -\infty$
- Repetă fără a depăși timpul disponibil:
 - alegem aleatoriu x un element al vectorului
 - $v = \max(v, x)$ = cel mai mare element ales
- scrie v

Algoritmi Monte Carlo

- Care este probabilitatea ca un element ales x să fie mai mic decât mediana?
- Care este probabilitatea ca răspunsul să fie greșit după k încercări? (toate cele k elemente alese (în timpul de rulare avut la dispoziție) să fie mai mici decât mediana (deci $v < \text{mediana}$)?)

Algoritmi Monte Carlo

- Care este probabilitatea ca răspunsul să fie corect după k încercări?
 - $P = 1 - 1/2^k$
 - *pentru* $k = 20$ $p > 0,999999$

Algoritmi Monte Carlo

- Se consideră un vector cu n elemente. Să se determine dacă există un element majoritar în vector (cu frecvența $> n/2$)
- Problemă de decizie

Algoritmi Monte Carlo

- $v = -\infty$
 - Repetă fără a depăși timpul disponibil:
 - alegem aleator x un element al vectorului
 - Calculăm f = frecvența lui x
 - Dacă $f > n/2$ scrie DA; STOP
 - scrie NU
-
- Care este probabilitatea de a răspunde greșit NU după k pași?

Algoritmi Las Vegas

- Nu furnizează totdeauna un rezultat, dar dacă furnizează un rezultat atunci acesta este corect
- Probabilitatea ca algoritmul să se termine crește pe măsură ce timpul disponibil crește

Algoritmi Las Vegas

- Se dau n texte (n foarte mare) cu următoarele proprietăți:
 - există un unic text t_0 care apare de cel puțin 10% ori;
 - celelalte texte sunt distincte.
- Se cere determinarea textului t_0 .
- Generăm aleatoriu doi indici i și j și testăm dacă
 - $i \neq j$ și $t_i = t_j$
 - până când testul se încheie cu succes

Algoritmi Las Vegas

```
repeat
    if Text()
        stop
until false
```

```
Text()
     $i \leftarrow \text{random}(1..n); j \leftarrow \text{random}(1..n);$ 
    if  $i \neq j$  and  $t_i = t_j$ 
        write  $t_i$ ; return true
    else
        return false
```