

Algoritmi esponenziali

NP-Completitudine

Metode de elaborare a algoritmilor

- ▶ **Greedy** – probleme de optim
 - corectitudine
 - algoritmi polinomiali
- ▶ **Divide et impera**
- ▶ **Programare dinamică**

Metode de elaborare a algoritmilor

- ▶ **Greedy** – probleme de optim
 - corectitudine
 - 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
 - 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 + **pseudopolinomiali**

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
 - $B \in NP$ a.î. $\forall A \in NP, A \leq_p B$ (reducere în timp polinomial)
 - Dacă pentru un B se găsește algoritm polinomial, atunci $P = NP$
 - SAT (Cook–Levin)
- 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:

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...

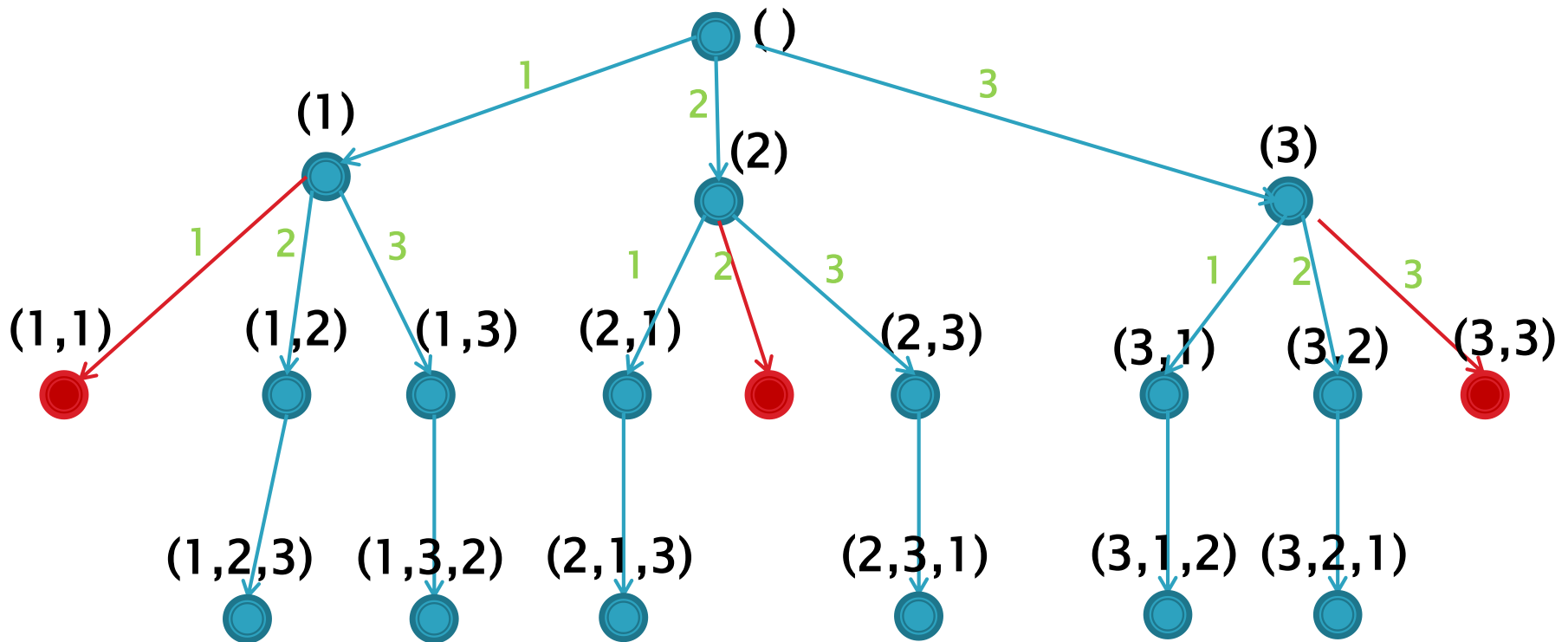
Backtracking, Branch and Bound

Cadru comun

- ▶ Căutare mai “inteligentă” în spațiul în care se găsesc soluțiile posibile (printre soluții candidat)
- ▶ **Configurațiile prin care se trece în procesul de căutare**
 - structură arborescentă

Cadru comun

- ▶ Permutări $\{1, 2, 3\}$



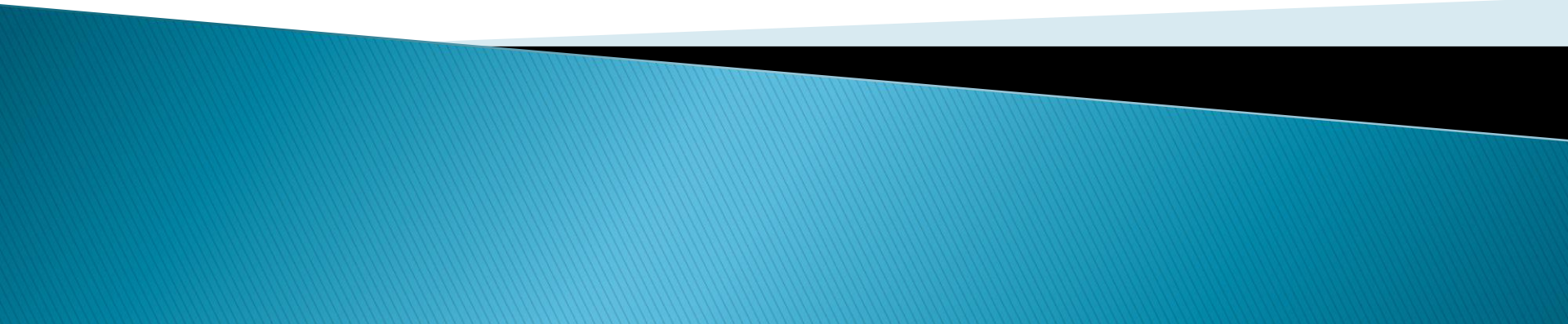
Cadru comun

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

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 mai trebuie explorate** (nu pot conduce către soluții dorite)
- ▶ 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



Metoda Backtracking

- ▶ Probleme computaționale + cu constrângeri (constraint satisfaction problems) – puzzle
- ▶ Combinatorică
- ▶ Căutare IA

Metoda Backtracking

- ▶ Soluțiile se construiesc element cu element, trecând prin configurații **corespunzătoare soluțiilor parțiale (“incomplete”)**
- ▶ Aceste configurații se pot **testa** dacă **nu** pot fi completate până la o soluție posibilă → **condiții de continuare**

Cadru posibil

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

Cadru posibil

- ▶ $X = X_1 \times \dots \times X_n =$ **spațiul soluțiilor candidat**
- ▶ $\varphi : X \rightarrow \{0, 1\}$ este o **proprietate** definită pe X
- ▶ **Căutăm un element $x \in X$ cu proprietatea $\varphi(x)$, numite condiție internă (finală) pentru x**
 - Limitarea parcurgerii arborelui de configurații – cu ajutorul condițiilor de continuare

Metoda Backtracking

Condiții de continuare pentru soluția parțială $y = x_1 \dots x_k$
notate $\text{cont}_k(x)$ = condiții de continuare a parcurgerii
subarborelui de rădăcină x

- Condițiile de continuare
 - rezultă de obicei din condițiile interne
 - sunt strict necesare, **ideal fiind să fie și suficiente**
 - sunt importante pentru micșorarea timpului de executare

Metoda Backtracking

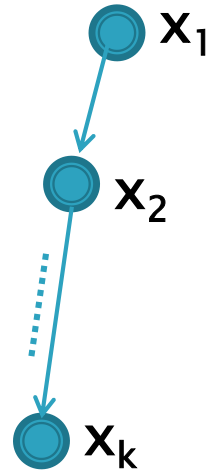
- ▶ Vectorul soluție $x = (x_1, x_2, \dots, x_n) \in X$ este **construit progresiv**, începând cu prima componentă.

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** $\text{cont}_k(x_1, \dots, x_k)$

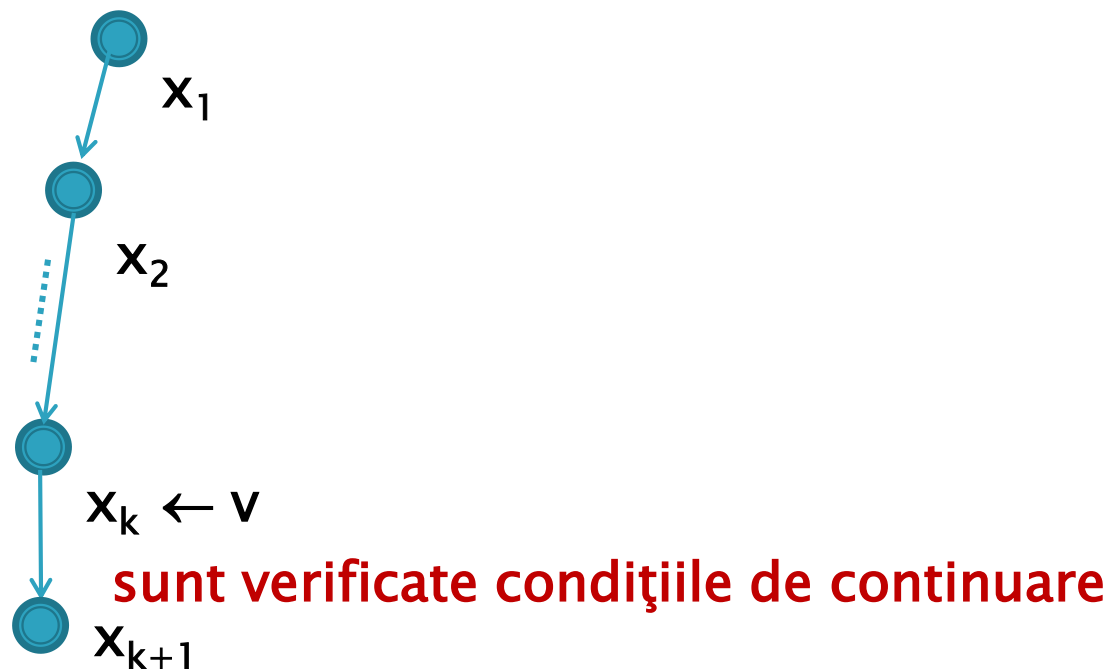
Metoda Backtracking

- Cazuri posibile pentru x_k :



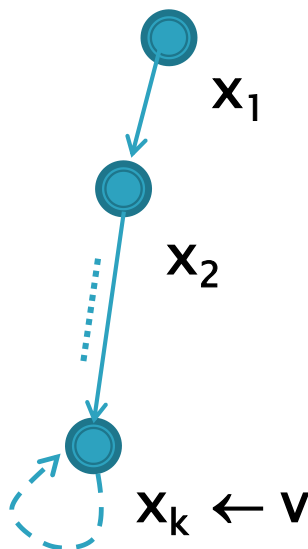
Metoda Backtracking

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



Metoda Backtracking

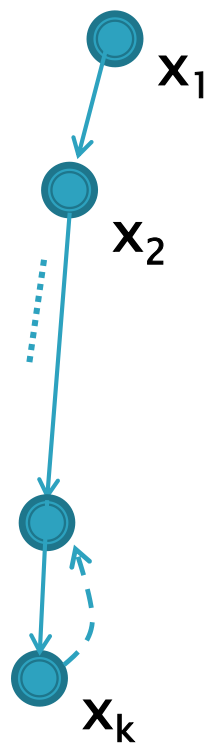
- ❑ Încercare eșuată (atribuie o valoare $v \in X_k$ lui x_k pentru care nu sunt verificate condițiile de continuare)



nu sunt verificate condițiile
de continuare

Metoda Backtracking

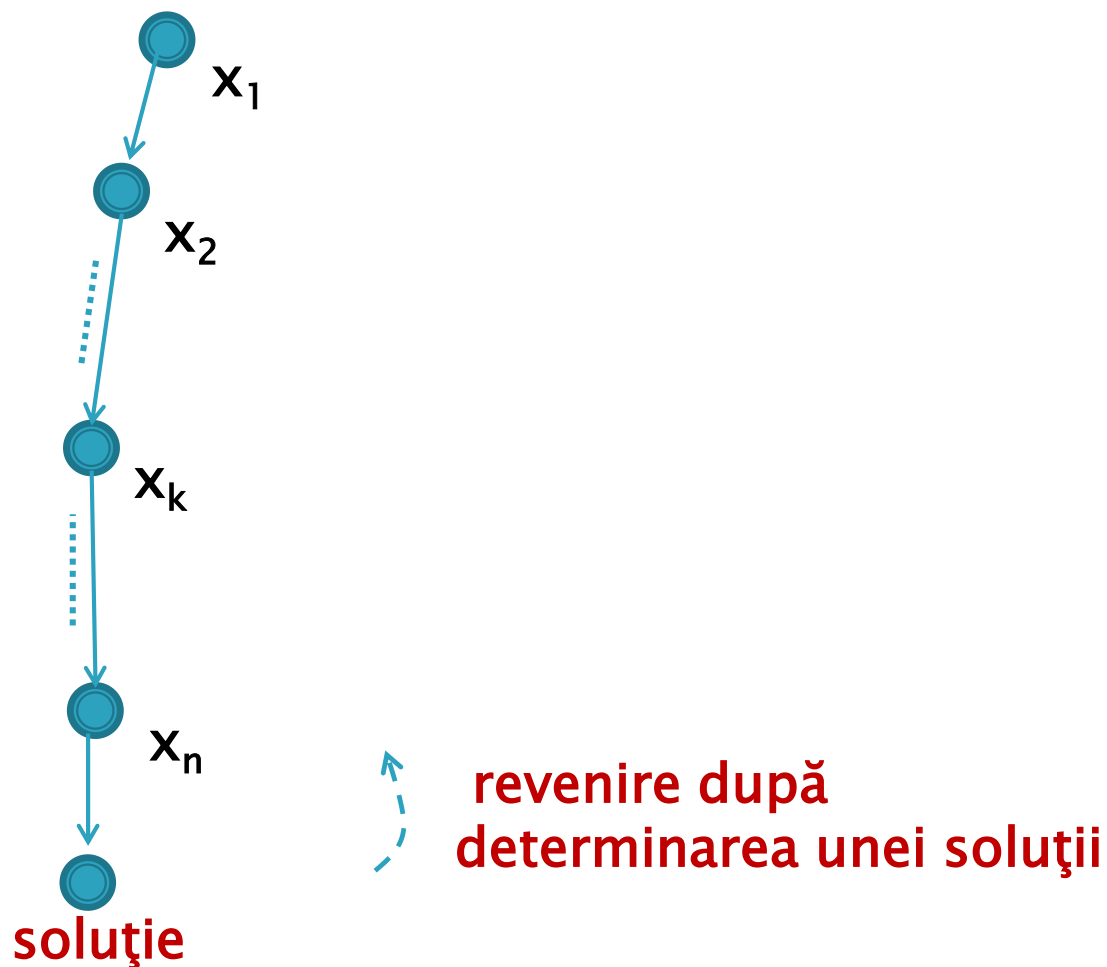
□ Revenire – nu mai există valori $v \in X_k$ neconsiderate



nu mai există valori pentru x_k
neconsiderate

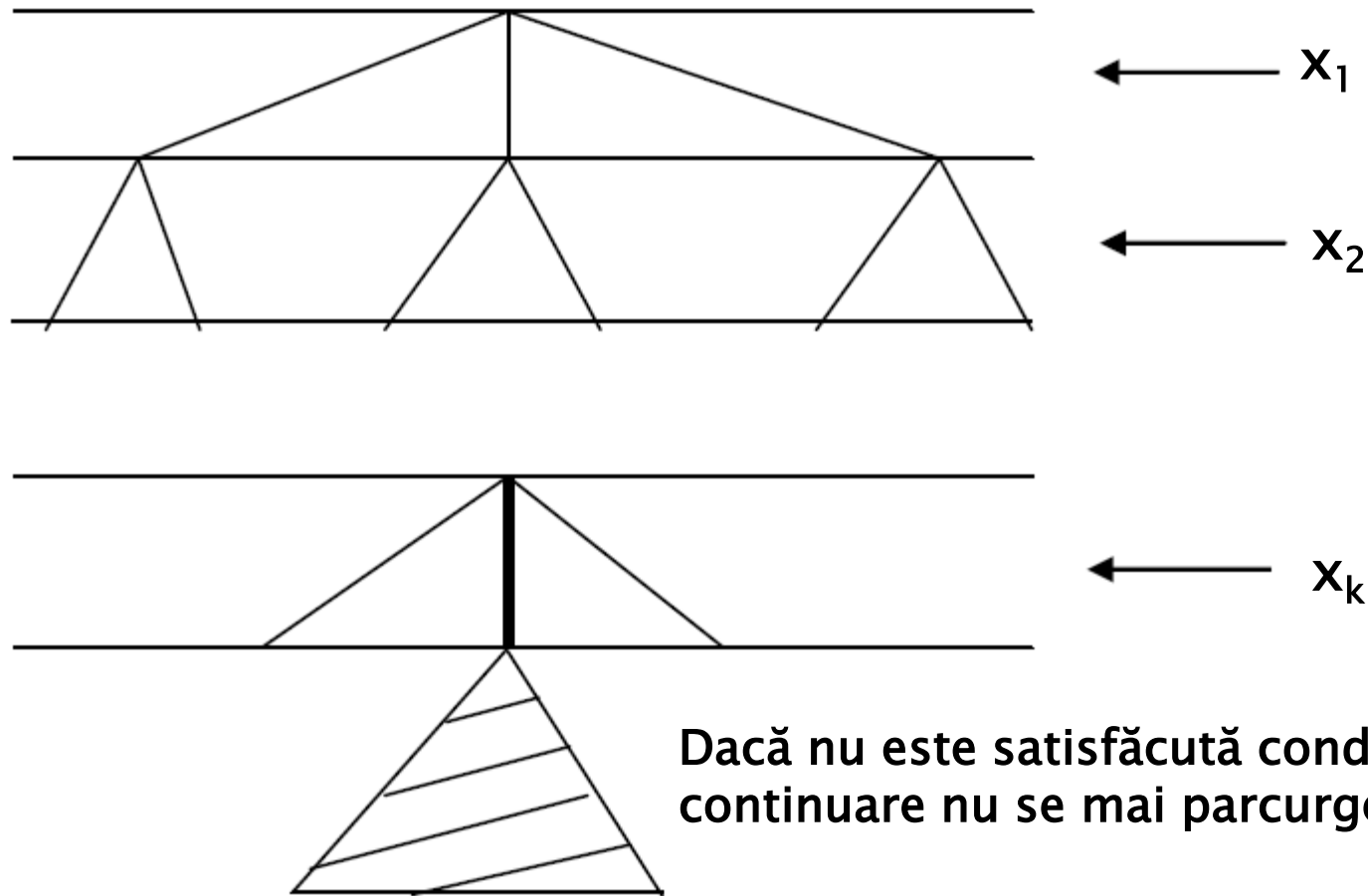
Metoda Backtracking

- Revenire după determinarea unei soluții



Metoda Backtracking

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



Varianta nerecursivă – pseudocod

- pentru soluții cu lungime fixă–

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

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

$k \leftarrow 1;$

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

$k \leftarrow 1;$

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 $\mathbf{x}_k < \mathbf{u}_k$ {**mai sunt valori în \mathbf{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 $k = n + 1$

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

 else

 if $x_k < u_k$ { **mai sunt valori în X_k** }

$x_k \leftarrow x_k + 1;$

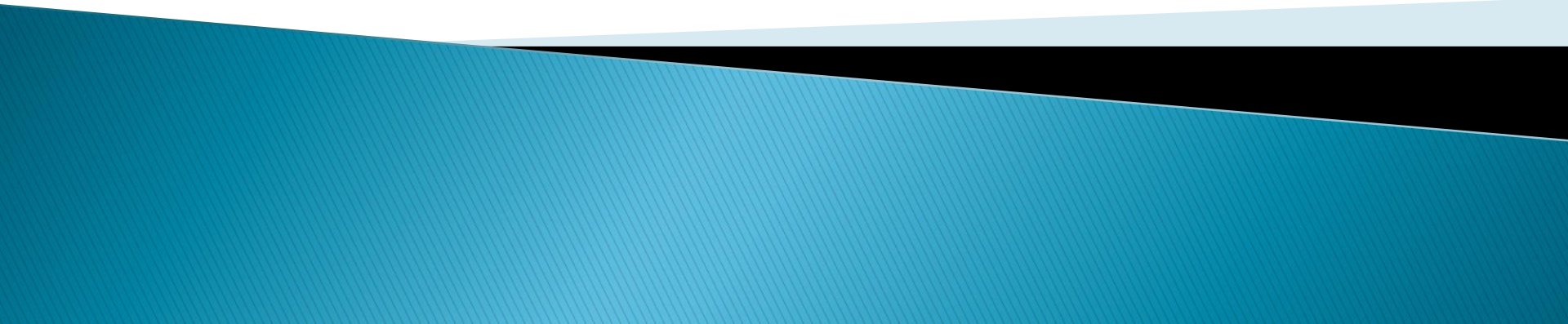
 if cont(x_1, \dots, x_k)

$k \leftarrow k + 1$; { **atribuie și avansează** }

 else { **încercare eșuată** }

 else $x_k \leftarrow p_k - 1$; $k \leftarrow k - 1$; { **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}
      xk ← i; {atribuie}
      if cont(x1, ..., xk)
        back(k+1); {avanseaza}
        {revenire din recursivitate}
end.
```

Cazul X_k oarecare

► **back(1)**

```
procedure back(k)
  if esteSolutie()
    retsol(x) {revenire dupa solutie}
  else
     $X_k \leftarrow$  mulțimea valorilor posibile
    for ( $i \in X_k$ ) {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
- ▶ Toate subșirurile crescătoare de lungime maximă
- ▶ Partițiile unui număr n
- ▶ Labirint
- ▶ Problema ciclului hamiltonian minim – lent
 - V. moodle (slide curs exemple backtracking)

Exemple

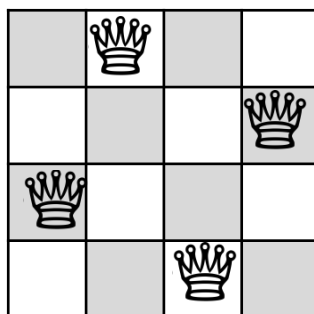
Pentru a testa condițiile de continuare $\text{cont}_k(x_1, \dots, x_k)$
vom folosi funcția `cont`

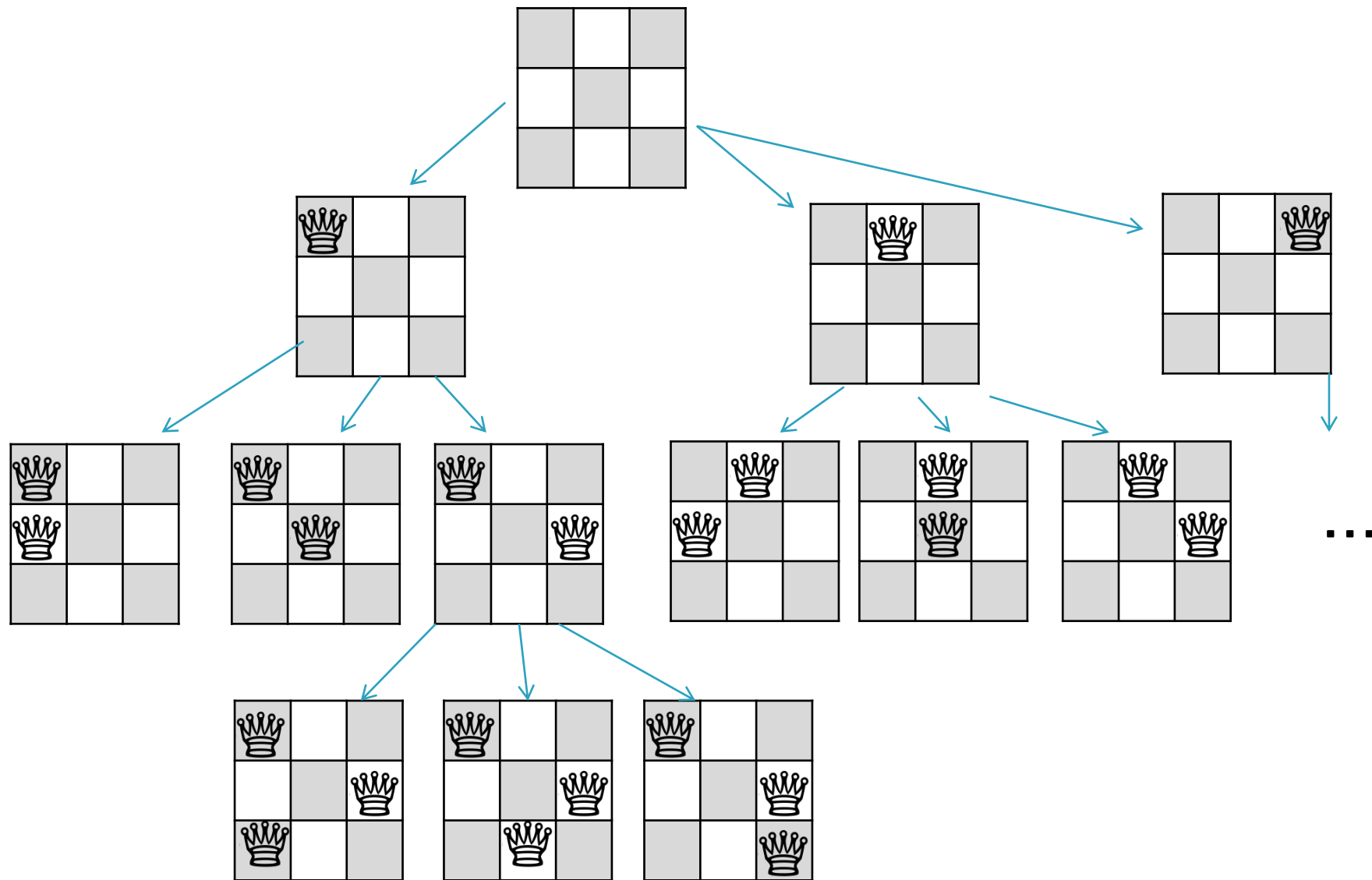
Problema celor n dame

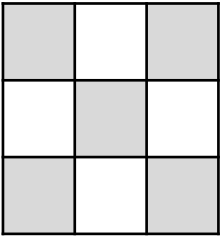


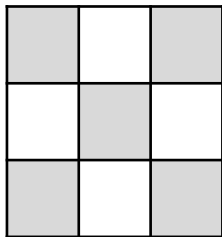
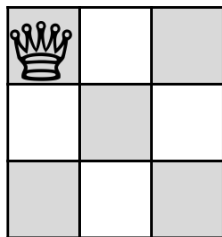
► Se consideră un caroiăj $n \times n$.

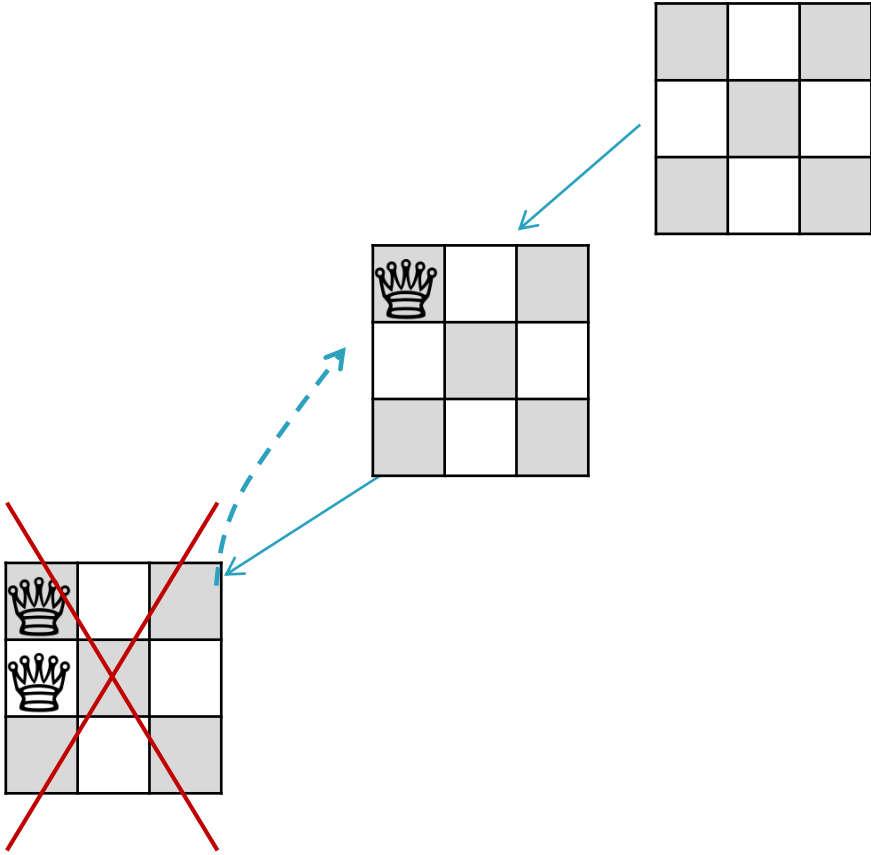
Prin analogie cu o tablă de șah ($n=8$), se dorește plasarea a n dame pe pătrățelele caroiăjului, astfel încât să nu existe două dame una în bătaia celeilalte

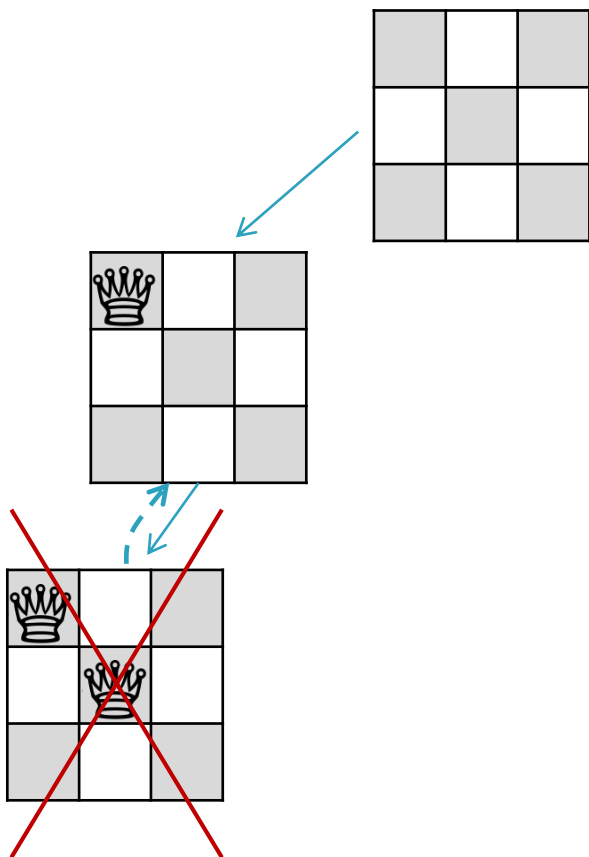


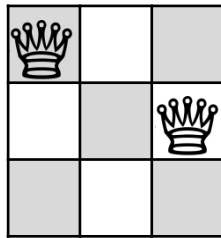
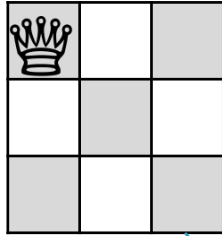
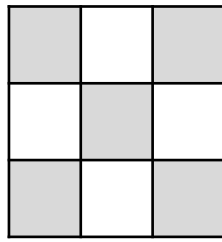


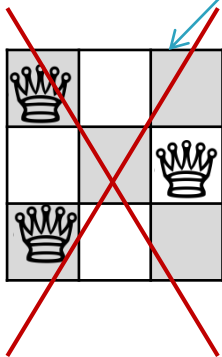
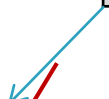
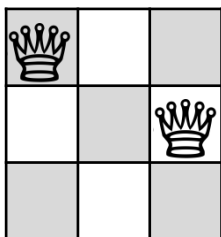
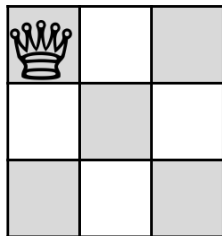
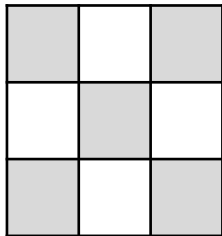


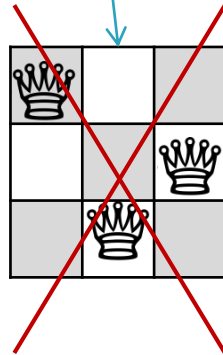
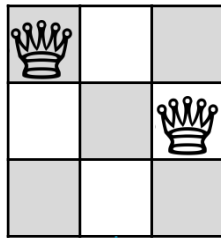
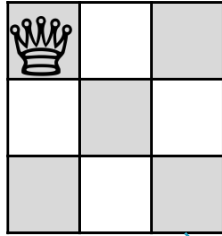
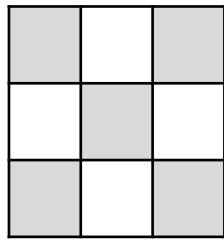


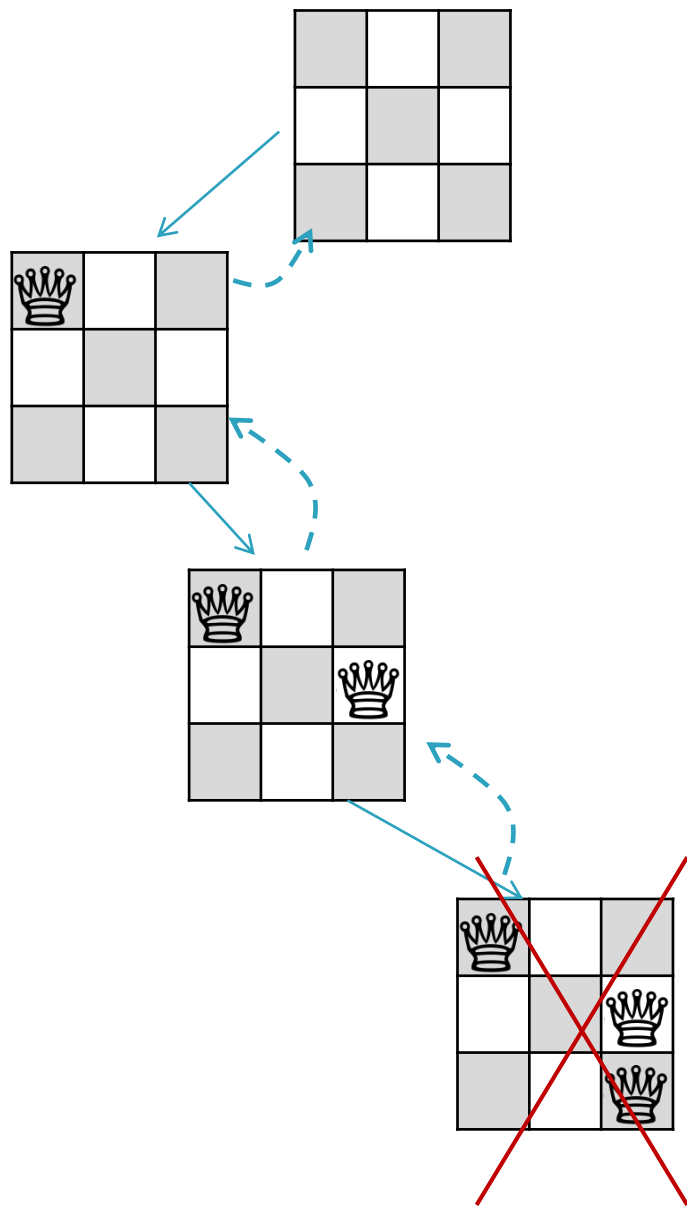


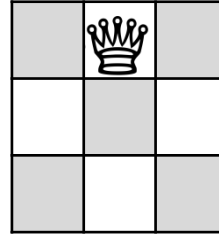
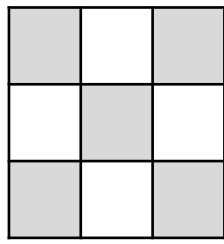


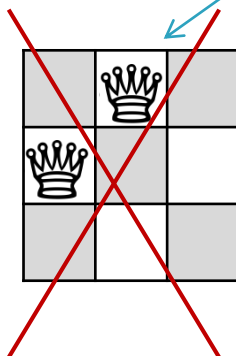
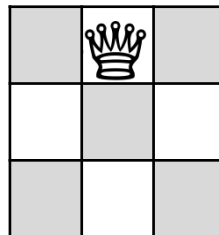
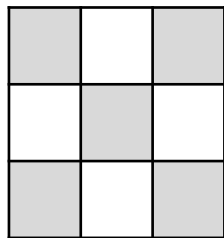


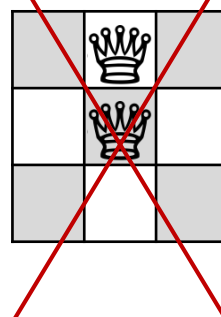
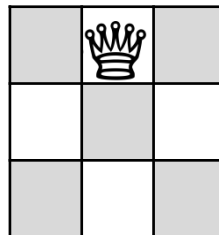
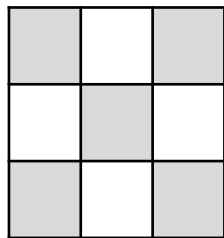


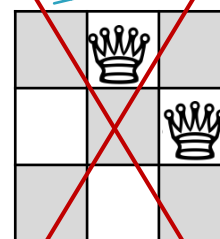
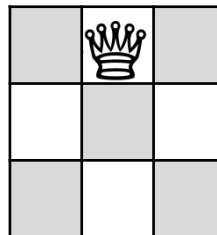
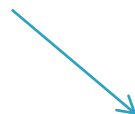
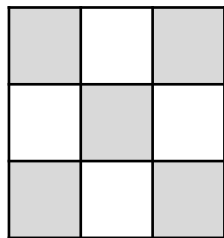


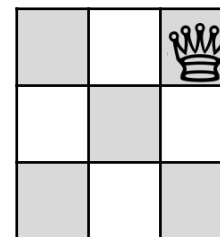
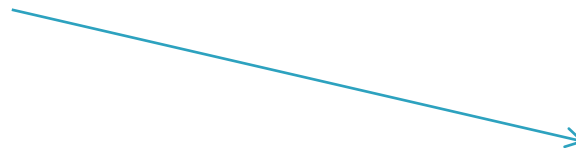
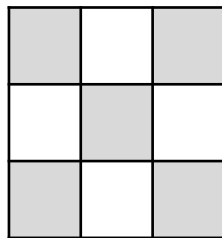






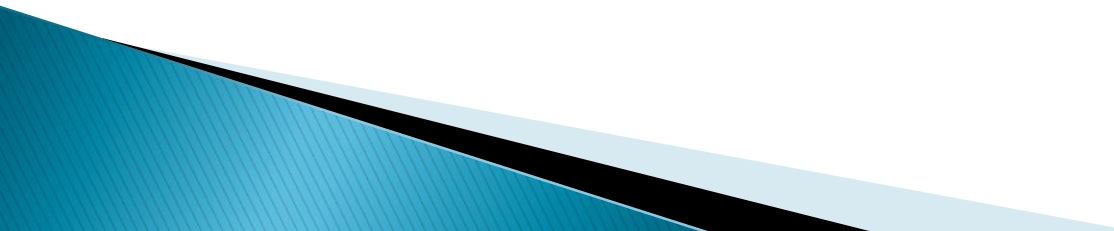






...
Temă

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

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

► 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 \mathbf{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]) || (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]) || (abs(x[k]-x[i])==k-i))  
            return false;  
    return true;  
}  
  
void retsol(int x[]){  
    for(int i=1;i<=n;i++)  
        cout<<" ("<<i<<", "<<x[i]<<" ) ";  
    cout<<endl;  
}
```

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**

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

$\mathbf{x}_k \in \{ ' (' , ') ' \}$

- ▶ **Condiții interne (finale)**

- ▶ **Condiții de continuare**

Șiruri corecte de paranteze

► Reprezentarea soluției

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

$\mathbf{x}_k \in \{ ' (' , ') ' \}$

► Condiții interne (finale)

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

$\text{dif} = 0$

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

► Condiții de continuare

Șiruri corecte de paranteze

► Reprezentarea soluției

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

$\mathbf{x}_k \in \{ ' (' , ') ' \}$

► Condiții interne (finale)

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

$\text{dif} = 0$

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

► Condiții de continuare

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

Șiruri corecte de paranteze

► Reprezentarea soluției

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

$\mathbf{x}_k \in \{ ' (' , ') ' \}$

► Condiții interne (finale)

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

$\text{dif} = 0$

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

► Condiții de continuare

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

$\text{dif} \leq n-k \quad \rightarrow$ 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
- ▶ **DE EVITAT** recalcularea lui dif la fiecare pas ca fiind

$$dif = nr_{(} - nr_{)}$$

pentru secvența $x_1 \dots x_k$ (v. și problema generării partițiilor unui număr n)

Metoda Backtracking



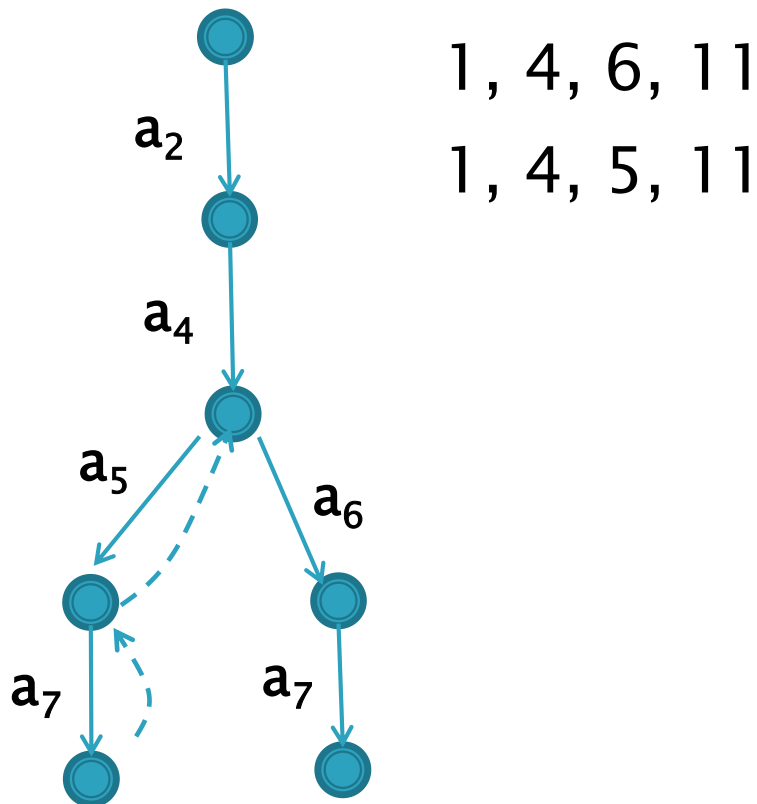
- ▶ Fie vectorul $a=(a_1,\dots,a_n)$. Să se determine **toate subșirurile crescătoare de lungime maximă**
- ▶ v. exemple backtracking moodle

Subșiruri crescătoare lungime maximă

- ▶ Un subșir crescător de lungime maximă începe cu o poziție i cu $lung[i] = l_{max}$
 - ▶ Următorul element după a_i este un a_j cu proprietățile
 - $i < j$
 - $a_j > a_i$
 - $lung[j] = lung[i] - 1$ (v.PD – numărarea subșirurilor)
- = **acei a_j posibili succesori ai lui i , pentru care se realizează egalitate în relația de recurență** (în PD se reține în $succ[i]$ un singur astfel de j)

Subșiruri crescătoare lungime maximă

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



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)

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

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 $x_1x_2\dots x_n$ (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 $x_1x_2\dots x_n$ (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 $x_1x_2\dots x_n$ (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

3SAT $O(2^n m)$

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

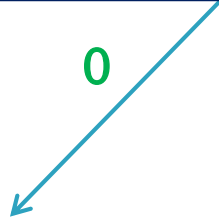
Dacă expresia E devine vidă atunci am găsit o soluție (restul variabilelor pot primi orice valoare)

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

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

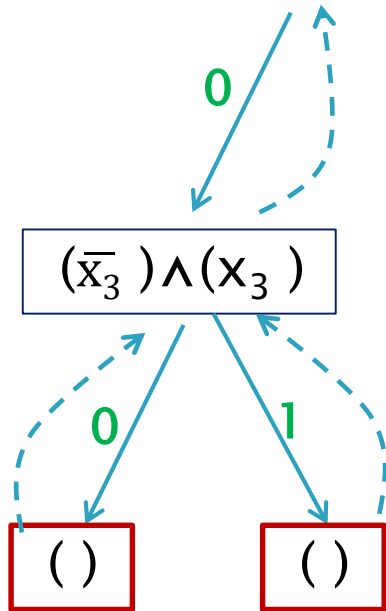
x_3 :

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 :

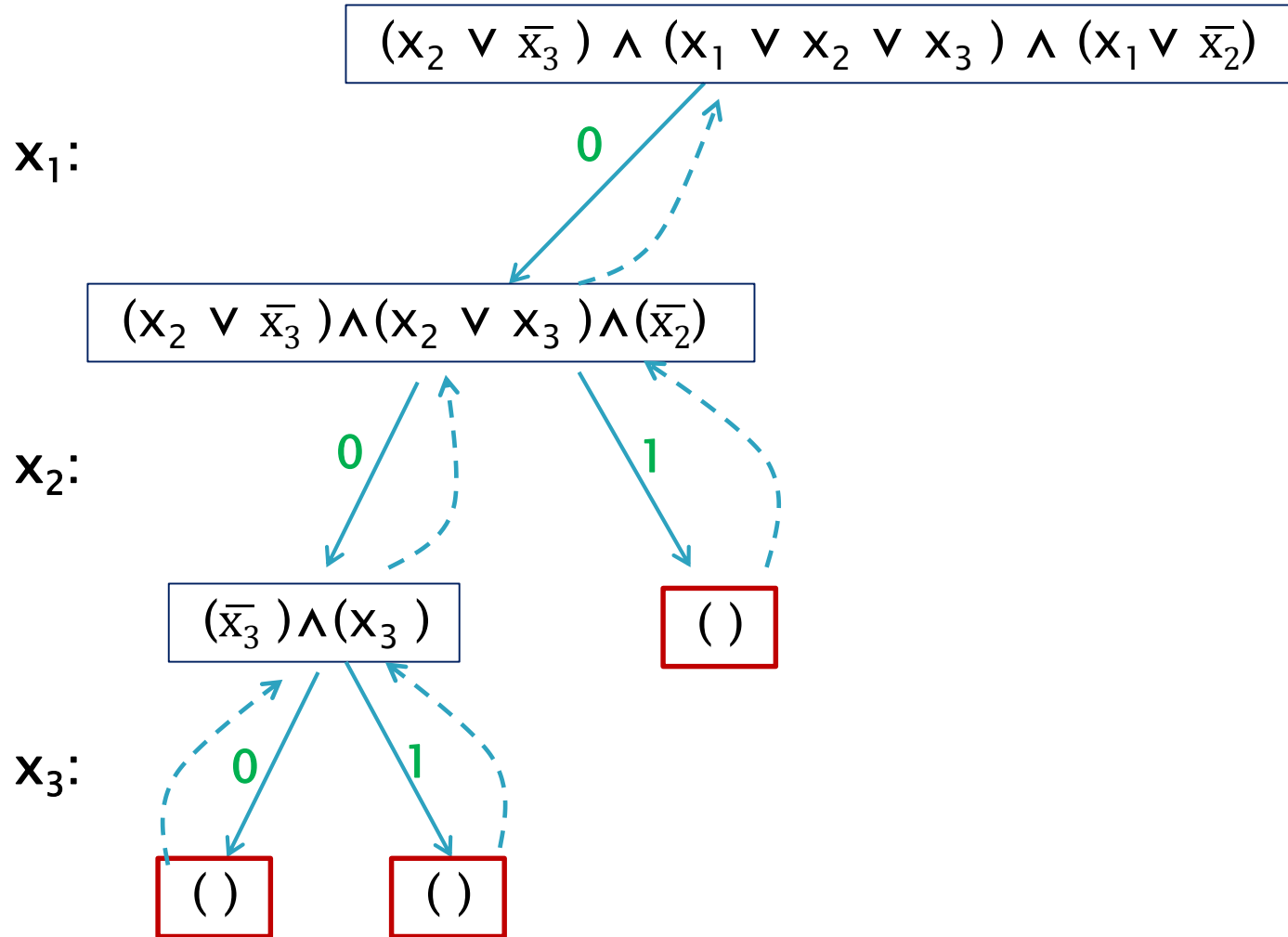
0

1

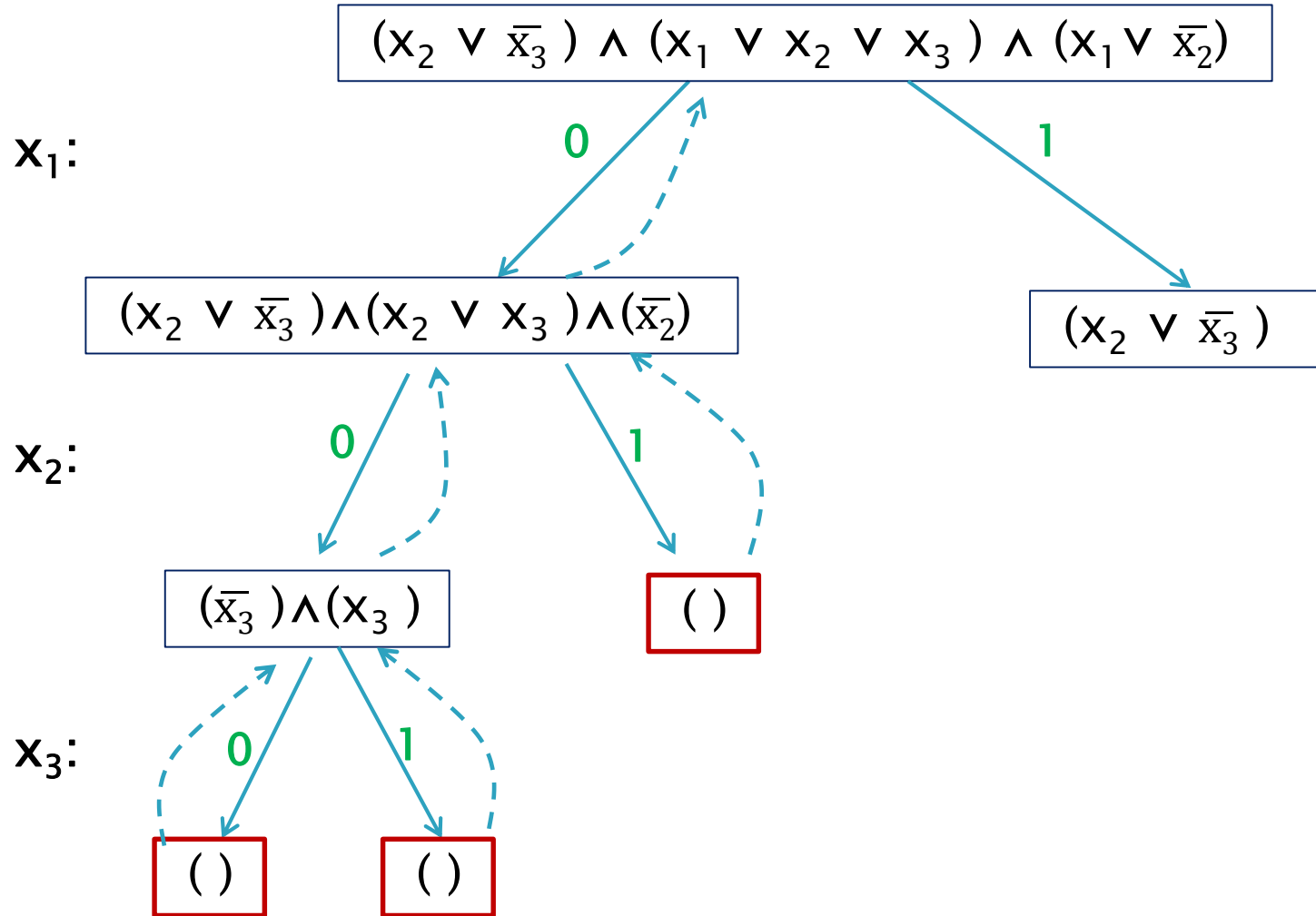
()

()

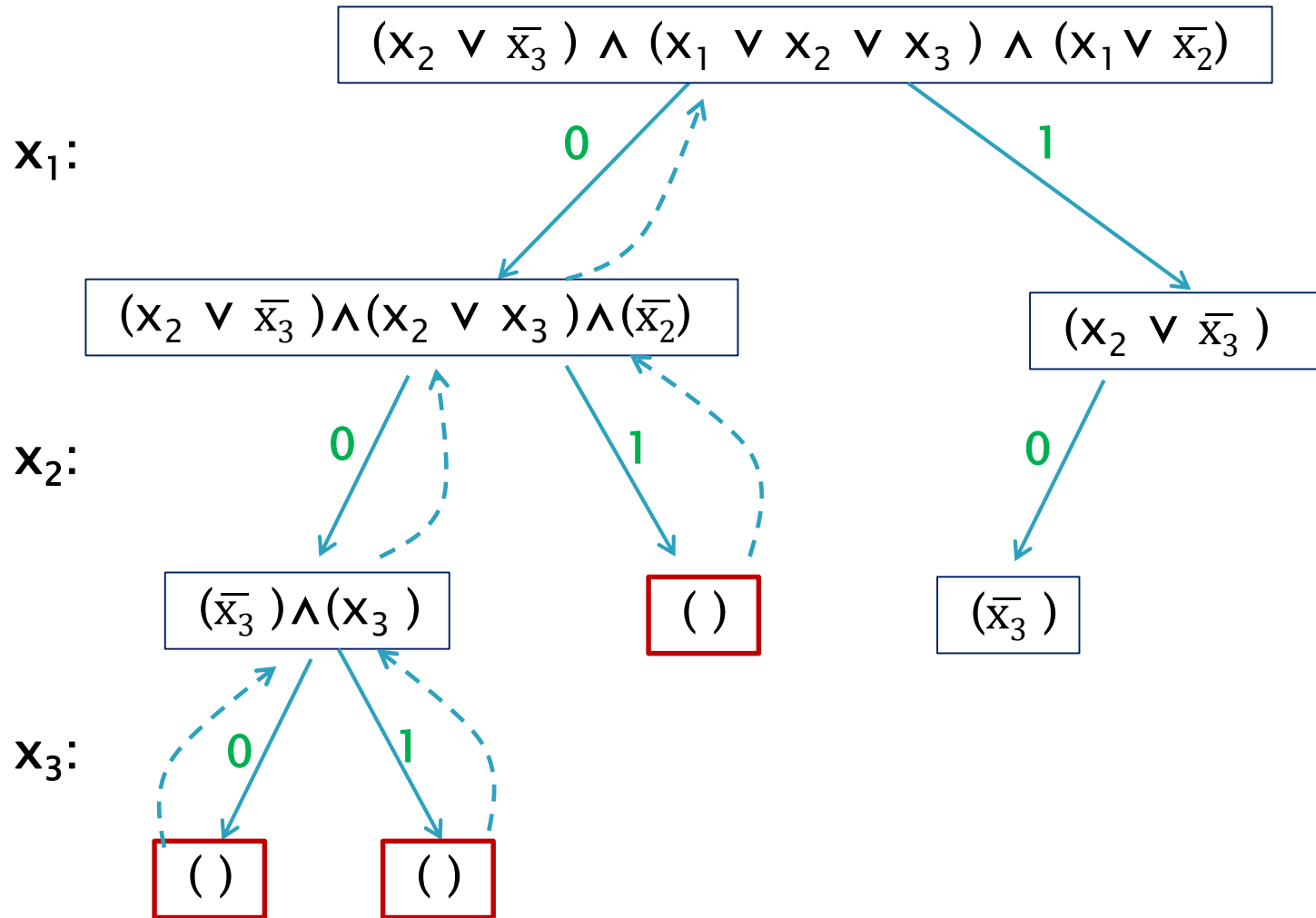
Metoda Backtracking – SAT



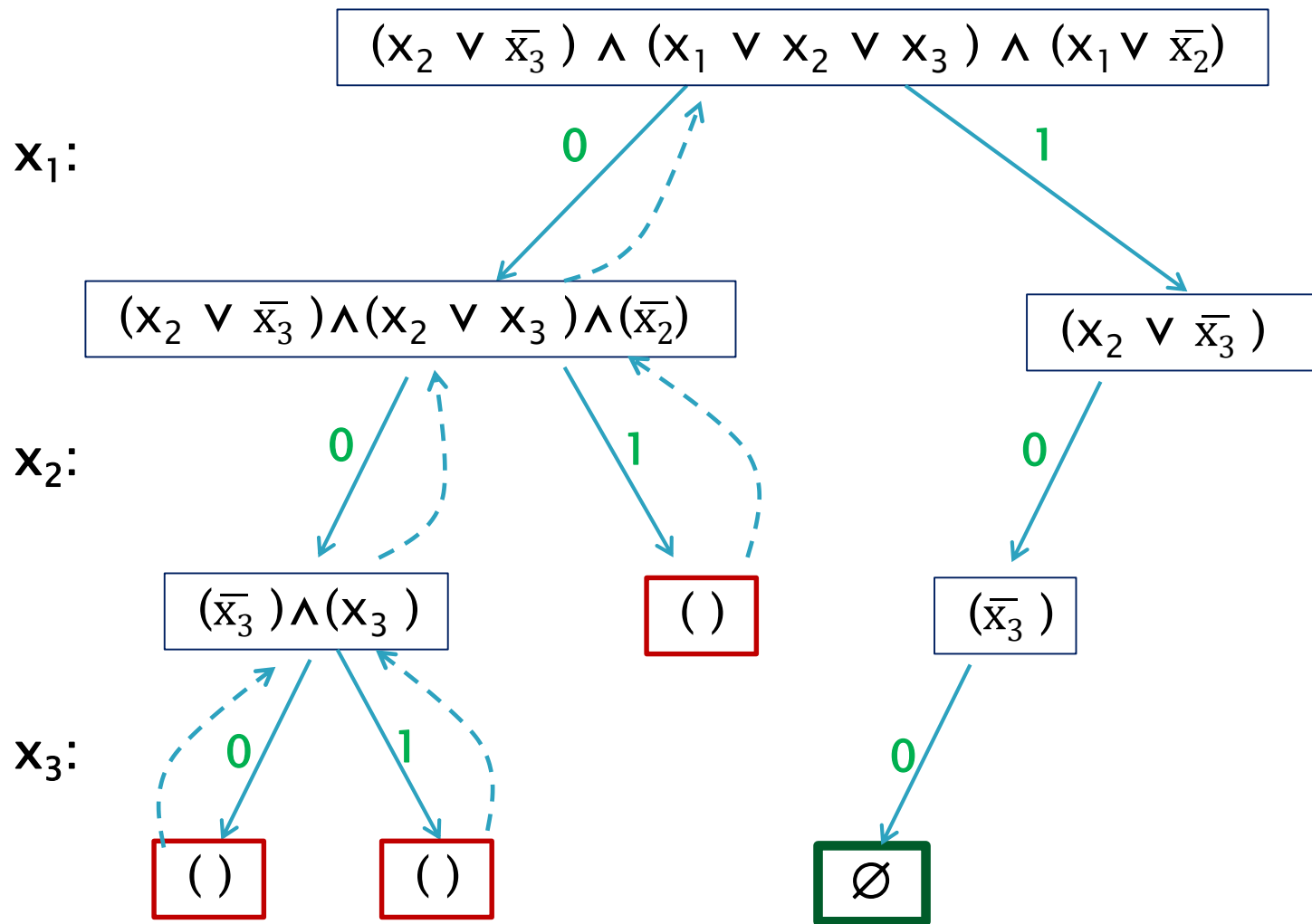
Metoda Backtracking – SAT



Metoda Backtracking – SAT



Metoda Backtracking – SAT



$E = \emptyset \Rightarrow$ Soluție + STOP

Metoda Backtracking – SAT

- ▶ Temă – backtracking pentru SAT /3-SAT

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

```

| |
|---|
| <pre> back(1) scrie false </pre> |
|---|

Metoda Backtracking – SAT

Posibile îmbunătățiri:

- ▶ Ordinea în care se dau valori variabilelor
- ▶ Prelucrări + formule de logică
- ▶ Învățare din rezultatele deja obținute

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

```

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

```

```

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

```

```

scrie false

```

Metoda Backtracking – SAT

▶ Prelucrări

- Clauze cu un literal \rightarrow determină unic valoarea variabilei din clauză
- Dacă un literal apare în formulă într-o singură formă (fie x , fie \bar{x}) \rightarrow atribuim variabilei corespunzătoare valoarea astfel încât literalul să devină adevărat

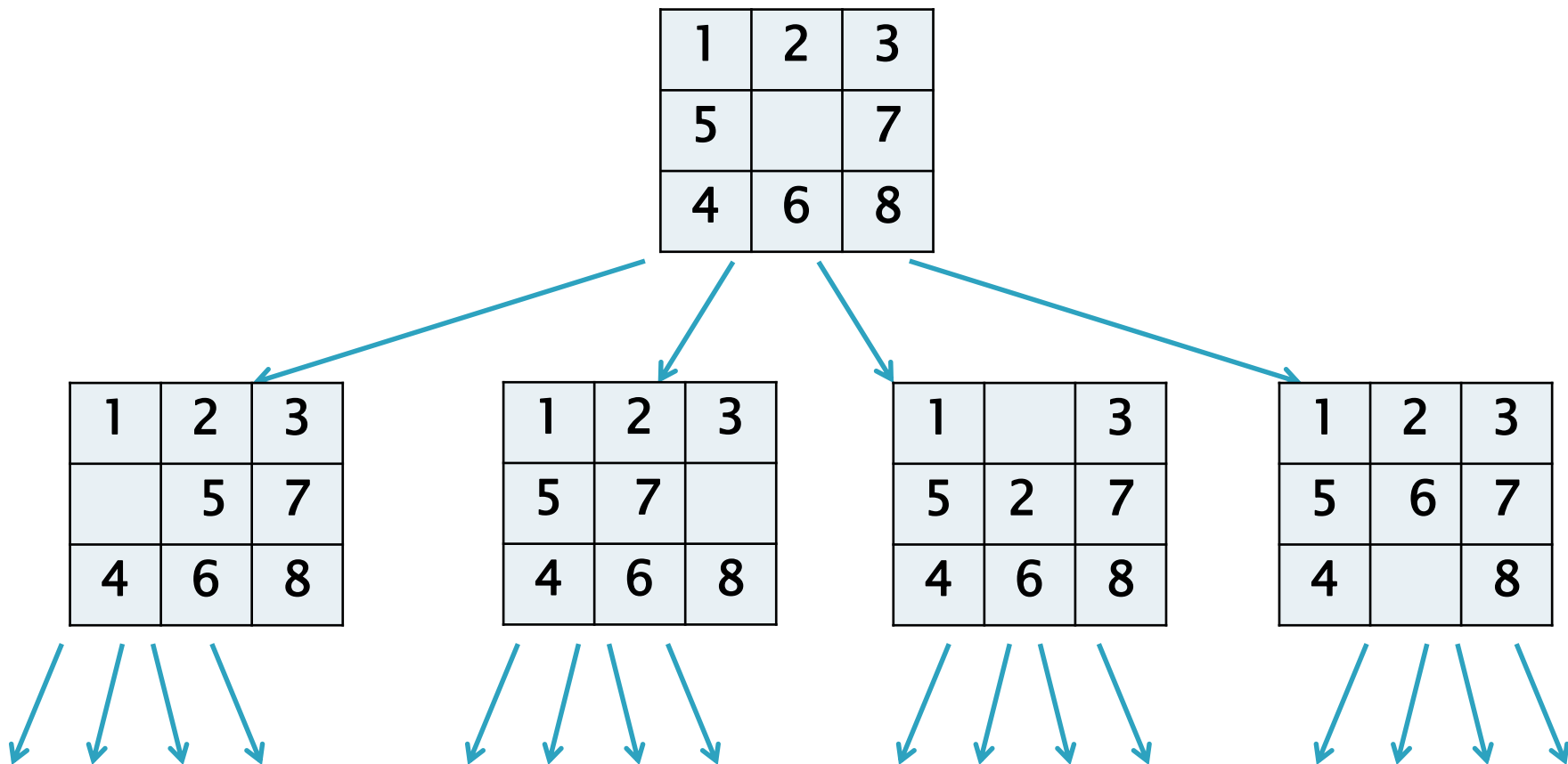
▶ Algoritmul Davis–Putnam

- ▶ C.P. Gomes, H. Kautz, A. Sabharwal, B. Selman, *Satisfiability Solvers*, Handbook of Knowledge Representation, Elsevier 2008.

Backtracking

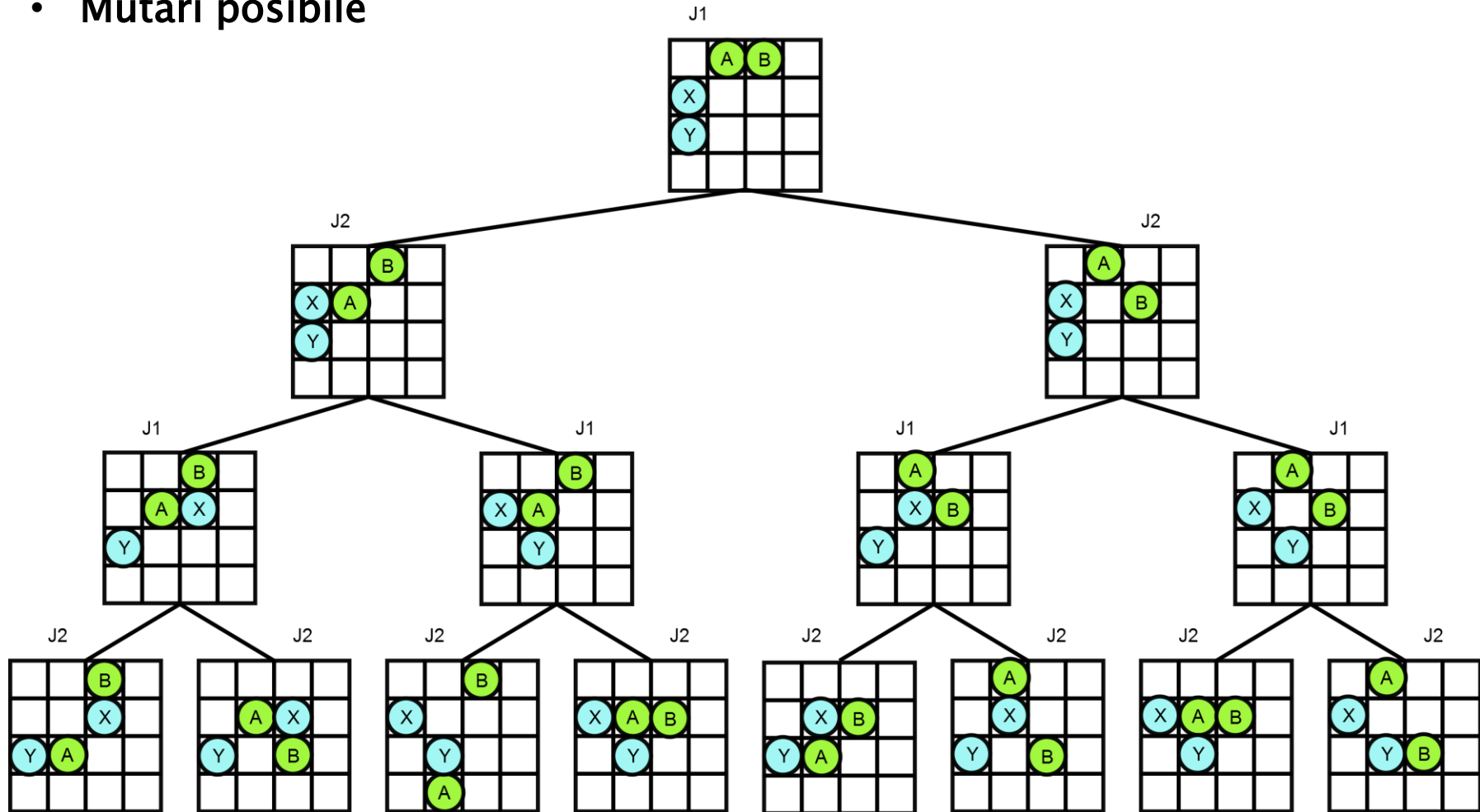
▶ **Arbori asociați mutărilor într-un joc**

- 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**



Perspico 3x3

- 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 (suplimentar) – 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

- Perspico

- 2x2 fake-sugar-packet game

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

- 2x2 Rubik's cube MIT

- <https://courses.csail.mit.edu/6.006/fall11/psets/ps6.pdf>

- <https://courses.csail.mit.edu/6.006/fall11/rec/rec16.pdf> etc

Backtracking în plan

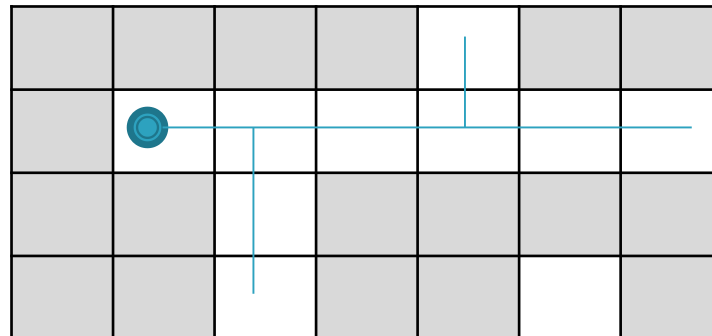
- ▶ **Labirint.** Se consideră un caroiaj (matrice) A cu m linii și n coloane. Pozițiile pot fi:

- 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 –trasee 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 – **NU Backtracking**

Backtracking –trasee 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 – NU Backtracking
 - > parcurgerea în lățime

Backtracking în plan

Indicații

- ▶ Matricea deplasărilor $dep1$ cu două linii și $ndep1$ coloane :

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

- ▶ dacă poziția este liberă și putem continua o marcă și continuăm
- ▶ **repunem $a_{ij}=0$ la întoarcere** (din recursivitate)
- ▶ Bordăm matricea cu 2 (pentru a testa mai ușor ieșirea)
 - **V. slide curs exemple backtracking**

```
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
                }
    }
}

```

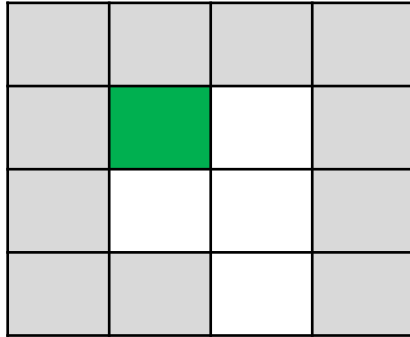
Apel:

$x_1 \leftarrow (i_0, j_0);$

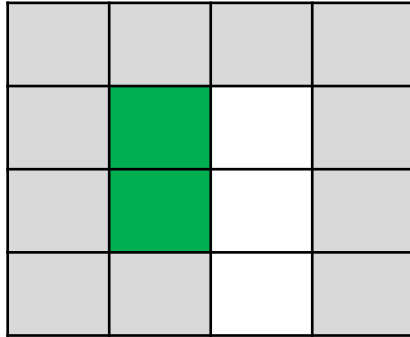
$k = 1;$

back(i_0, j_0)

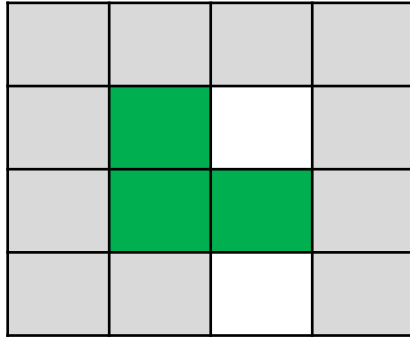
Backtracking in plan



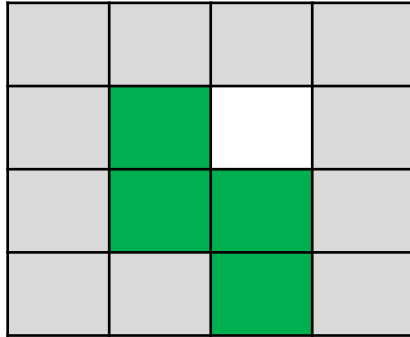
Backtracking in plan



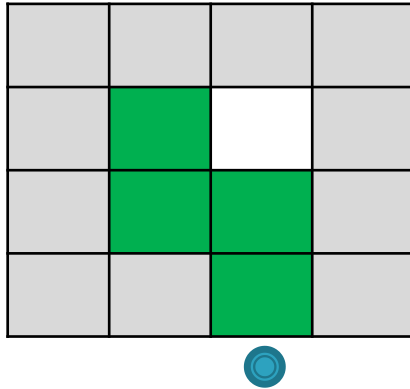
Backtracking in plan



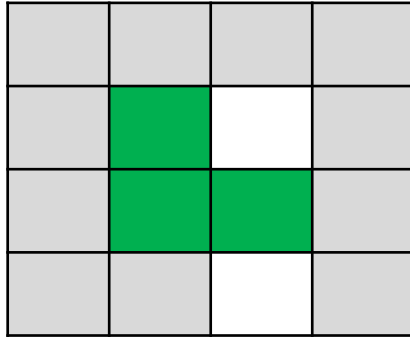
Backtracking in plan



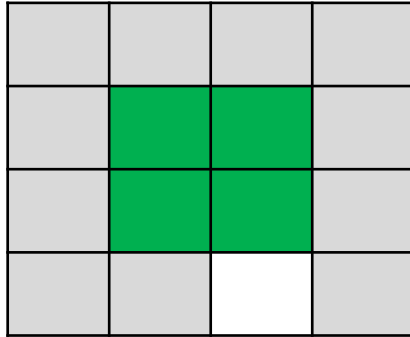
Backtracking in plan



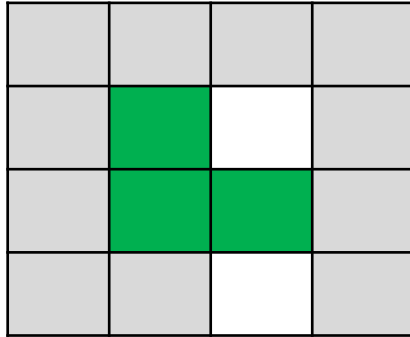
Backtracking in plan



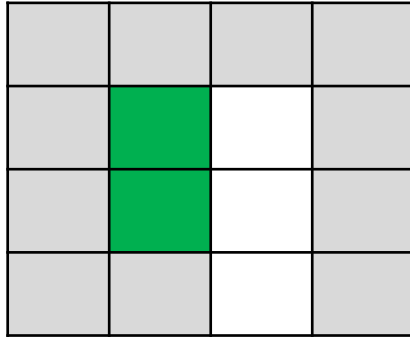
Backtracking in plan



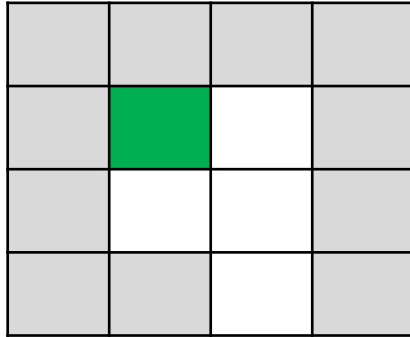
Backtracking in plan



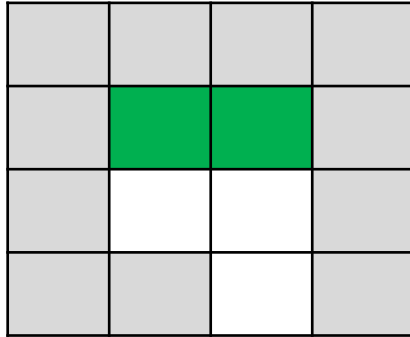
Backtracking in plan



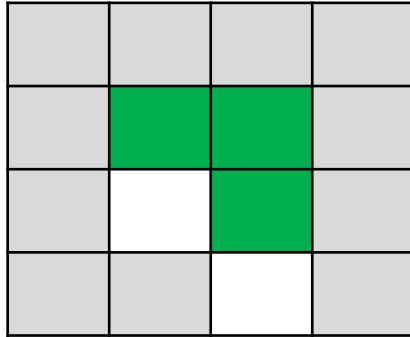
Backtracking in plan



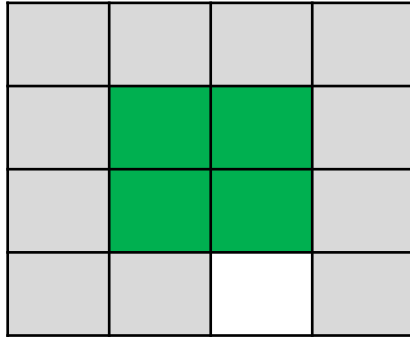
Backtracking in plan



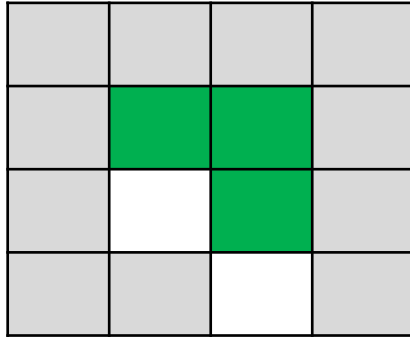
Backtracking in plan



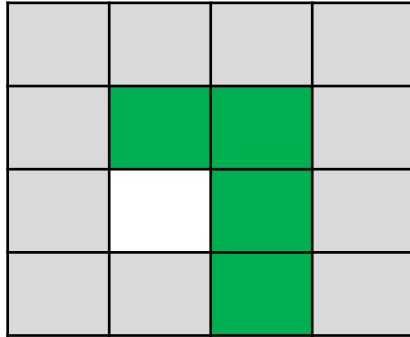
Backtracking in plan



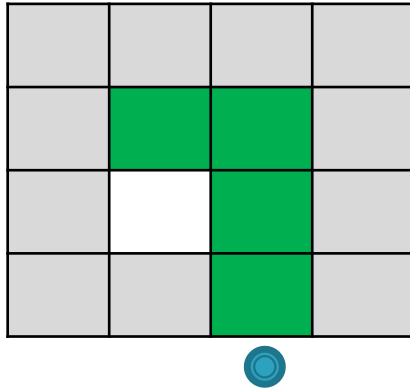
Backtracking in plan



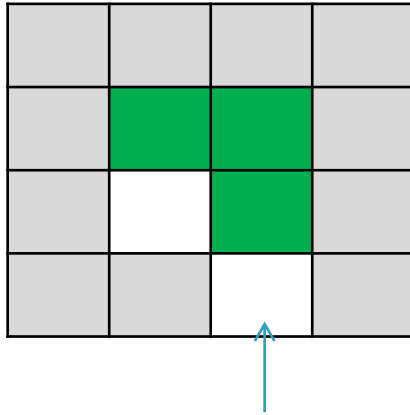
Backtracking în plan



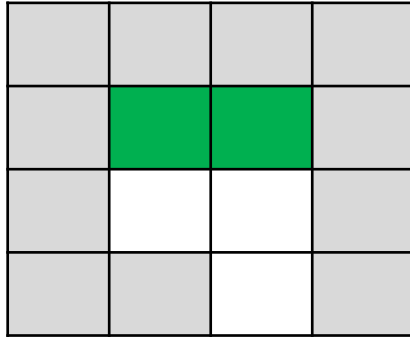
Backtracking in plan



Backtracking in plan



Backtracking in plan



Backtracking in plan

