Algoritmi exponenţiali

NP-Completitudine

Metode de elaborare a algoritmilor

- **Greedy** – probleme de optim
  - corectitudine (v. curs moodle actualizat)
  - 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 acelaşi tip
  - **construirea dinamică a unui arbore** (prin împărţirea în subprobleme) urmată de **parcurgerea în postordine a arborelui** (prin asamblarea rezultatelor parţiale).
  - 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 acelaşi tip
  - **construirea dinamică a unui arbore** (prin împărţirea în subprobleme) urmată de **parcurgerea în postordine a arborelui** (prin asamblarea rezultatelor parţiale).
  - algoritmi polinomiali

- **Programare dinamică** – rezolvare de recurenţe->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 esenţial.
- **Un algoritm este considerat "acceptabil" numai dacă timpul său de executare este polinomial**

Metode de elaborare a algoritmilor

Nu ştim algoritm polinomial – problemă grea?

Metode de elaborare a algoritmilor

Nu ştim algoritm polinomial – problemă grea?

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

Metode de elaborare a algoritmilor

Nu ştim algoritm polinomial – problemă grea?

NP

– există algoritm polinomial pentru a testa o soluţie candidat dacă este soluţie posibilă (verificator polinomial)

⇒ o problemă NP poate fi rezolvată în timp exponenţial (considerând toate soluţiile candidat)

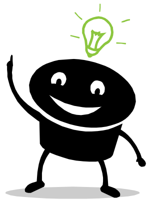
Metode de elaborare a algoritmilor

Nu ştim 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

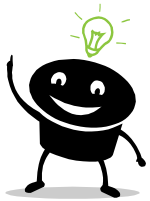


Nu ştim algoritm polinomial

- **Demonstrăm** NP - dificilă

Soluţii:

Metode de elaborare a algoritmilor



Nu ştim algoritm polinomial

- **Demonstrăm** NP – dificilă

Soluţii:

- algoritmi exponenţiali mai rapizi decât cei exhaustivi (brute force) de căutare în spaţiul soluţiilor: **Backtracking, Branch & Bound**

- **Compromis**: algoritmi mai rapizi care produc soluţii care nu sunt optime – algoritmi **euristici, aleatorii, genetici**…

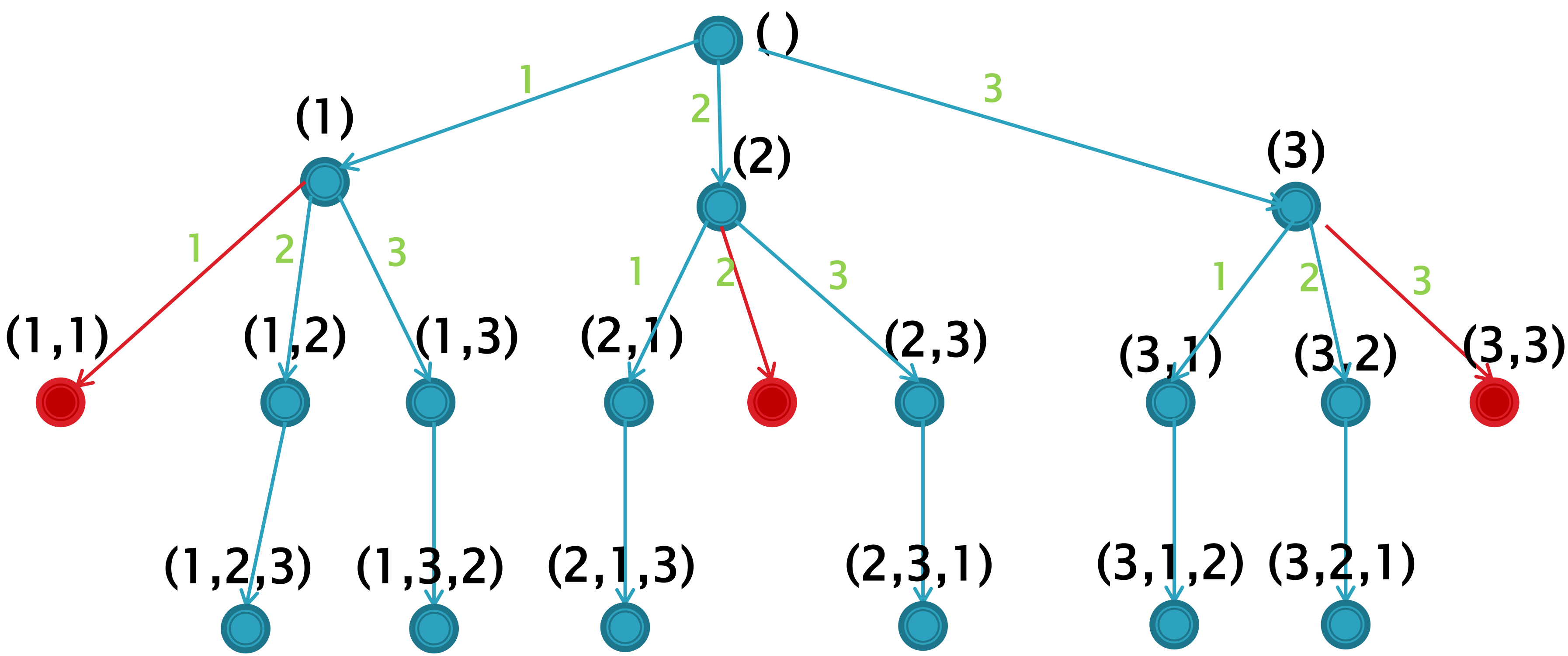
Backtracking, Branch and Bound

Cadru comun

- Căutare mai “inteligentă” în spaţiul în care se găsesc soluţiile posibile (printre soluţii candidat)
- **Configuraţiile prin care se trece în procesul de căutare – structură arborescentă**

Cadru comun

- Permutări `{1, 2, 3}`



Cadru comun

- Parcurgerea completă a arborelui ⇒ algoritm exhaustiv (brute force), care consideră toate soluţiile candidat ⇒ lent

Cadru comun

- Parcurgerea completă a arborelui ⇒ algoritm exhaustiv (brute force), care consideră toate soluţiile candidat
- Mai rapid – **limitarea parcurgerii** arborelui prin determinarea de configuraţii care **nu mai trebuie explorate (nu pot conduce către soluţii** dorite)
- **Diferenţe**

– modul în care este parcurs arborele (şi în care se fac limitările) = criteriul după care este ales nodul curent

– tipuri de probleme la care se pretează

Metoda Backtracking

Metoda Backtracking

- Soluțiile se construiesc element cu element, trecând prin configurații corespunzătoare soluțiilor parțiale (“incomplete”)
- Aceste configuraţii 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 $x_1, \ldots, x_n$
  - domeniul de valori pentru fiecare variabilă
  - constrângerea φ

Cadru posibil

- $X = X_1 \times \ldots \times X_n$ = **spaţiul soluţiilor candidat**
- $\varphi : X \to \{0,1\}$ este o **proprietate** definită pe X
- **Căutăm un element $x \in X$ cu proprietatea $\varphi(x)$, numite** <u>condiţie internă (finală)</u> pentru x

Metoda Backtracking

Condiţii de continuare pentru soluția parțială $y=x_1 \ldots x_k$ notate $cont_k(x)$ = condiții de continuare a parcurgerii subarborelui de rădăcină x

- **Condiţiile de continuare**
  - **rezultă de obicei din condiţiile interne**
  - sunt **strict necesare**, **ideal fiind să fie şi suficiente**
  - sunt importante pentru micşorarea timpului de executare

Metoda Backtracking

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

Metoda Backtracking

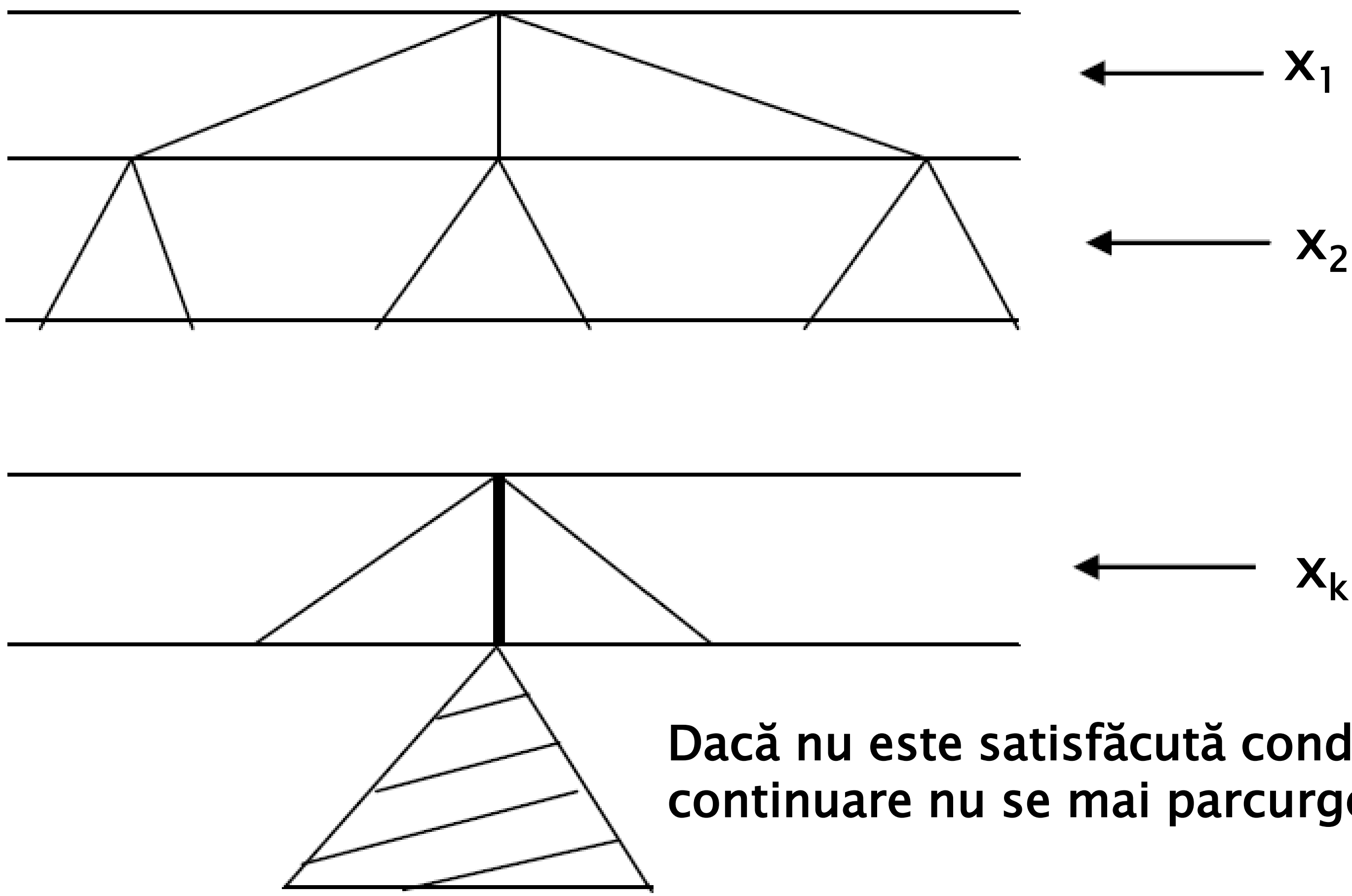
- Vectorul soluție $x = (x_1, x_2, \ldots, x_n) \in X$ este **construit progresiv**, începând cu prima componentă.
- Se avansează cu o valoare pentru x_k dacă este satisfăcută **condiţia de continuare** $cont_k(x_1,\ldots,x_k)$

Metoda Backtracking

- Vectorul soluție $x = (x_1, x_2,\ldots,x_n) \in X$ este **construit progresiv**, începând cu prima componentă.
- Se avansează cu o valoare pentru x_k dacă este satisfăcută **condiţia de continuare** $cont_k(x_1,\ldots,x_k)$

Metoda Backtracking

- Backtracking = parcurgerea <u>limitată</u> (de condiţiile de continuare) <u>în adâncime</u> a unui arbore



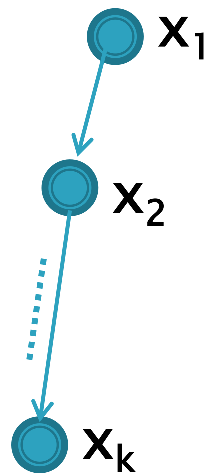
Dacă nu este satisfăcută condiția de continuare nu se mai parcurge subarborele

Metoda Backtracking

- Probleme computaţionale + cu constrângeri (constraint satisfaction problems) - puzzle
- Combinatorică
- Căutare IA

Metoda Backtracking

- Cazuri posibile pentru x_k:

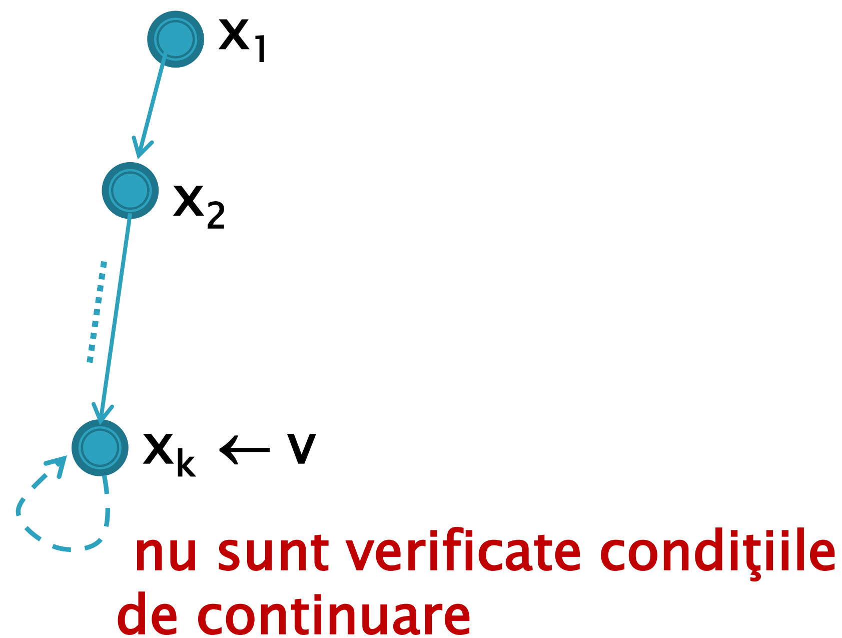


Metoda Backtracking

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

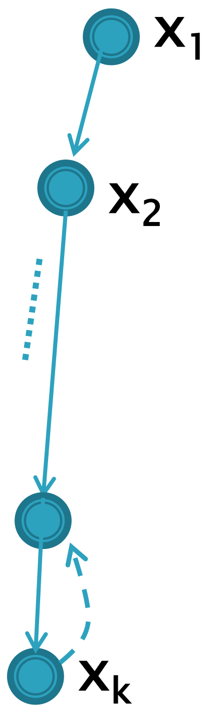
Metoda Backtracking

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



Metoda Backtracking

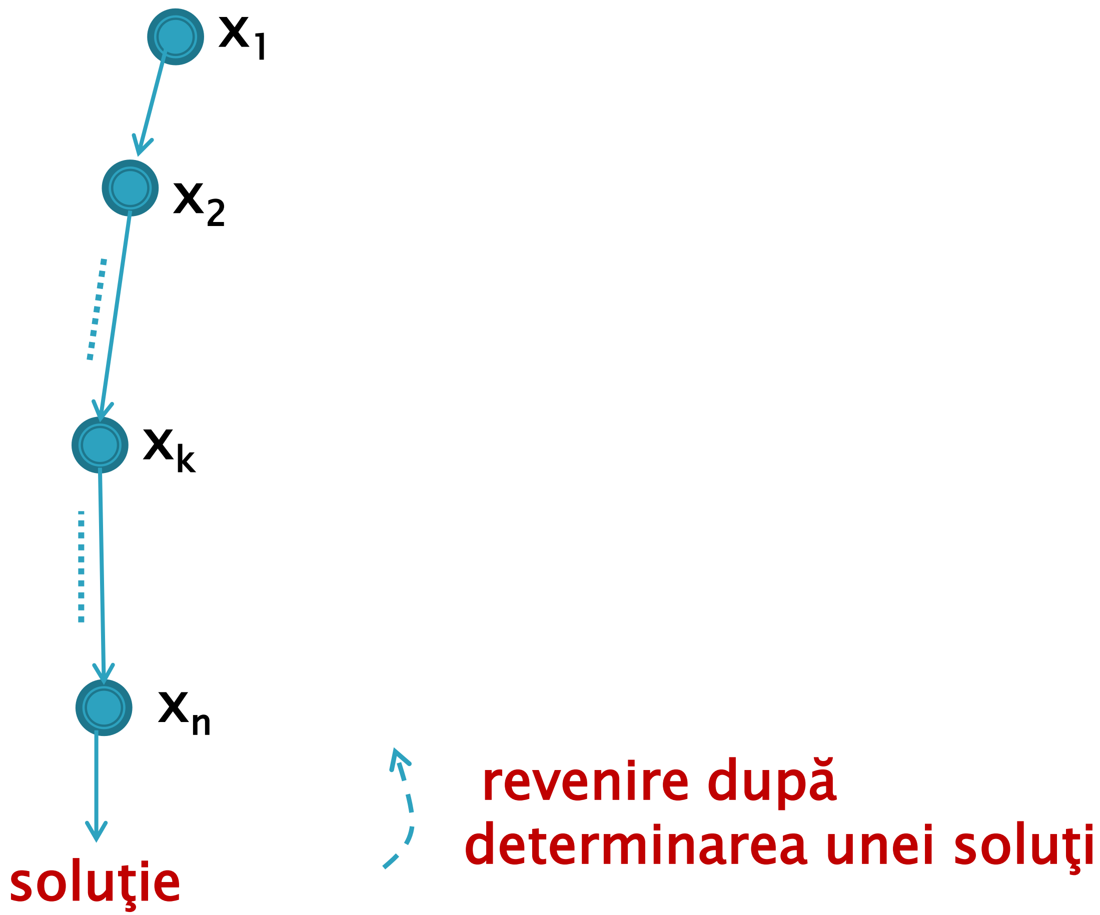
- Revenire – nu mai există valori $v \in X_k$ neconsiderate



nu mai există valori pentru x_k neconsiderate

Metoda Backtracking

- Revenire după determinarea unei soluţii



Varianta nerecursivă – pseudocod

– pentru soluţii cu lungime fixă–

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

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

$k \leftarrow 1;$

```
xi←pi-1, ∀i=1,...,n
k←1;
while k>0
   if k=n+1
       retsol(x); k←k-1;{revenire după o sol.}
   else
```

```
x_i←p_i-1, ∀i=1,...,n
k←1;
while k>0
   if k=n+1
       retsol(x); k←k-1;{revenire după o sol.}
   else
       if x_k<u_k  {mai sunt valori în X_k}
           x_k←x_k+1;
```

```
xi←pi-1, ∀i=1,...,n
k←1;
while k>0
   if k=n+1
        retsol(x); k←k-1;{revenire după o sol.}
   else
        if xk<uk  {mai sunt valori în Xk}
             xk←xk+1;
             if cont(x1,…,xk)
                  k←k+1;   { atribuie şi avansează }
             else              { încercare eşuată }
```

```
xi←pi-1, ∀i=1,...,n
k←1;
while k>0
   if k=n+1
       retsol(x); k←k-1;{revenire după o sol.}
   else
       if xk<uk  {mai sunt valori în Xk}
           xk←xk+1;
           if cont(x1,…,xk)
               k←k+1;   { atribuie şi avansează }
           else         { încercare eşuată }
       else xk←pk-1; k←k-1;   { revenire }
```

Varianta recursivă

- $X_i = \{p_i, p_i+1, \dots, u_i\}$
- Apelul iniţial este: **`back(1)`**

```
procedure back(k)
  if k=n+1
        retsol(x) {revenire dupa solutie}
  else









end.
```

- $X_i = \{p_i, p_i+1, \ldots, u_i\}$
- Apelul iniţial 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.
```

- $X_i = \{p_i, p_i+1, \ldots, u_i\}$
- Apelul iniţial 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
      Xk ← mulţimea valorilor posibile
      for (i ∈ Xk) {valori posibile}
         xk←i;               {atribuie}
         if cont(x1,…,xk)
            back(k+1);       {avanseaza}
        {revenire din recursivitate}
end.
```

Exemple

Exemple – de ştiut

- Permutări, combinări, aranjamente
- Colorarea hărţilor
- Toate subşirurile crescătoare de lungime maximă
- Partiţiile unui număr n
- Labirint
- Problema ciclului hamiltonian minim – lent
  - V. moodle (slide curs exemple backtracking+surse)

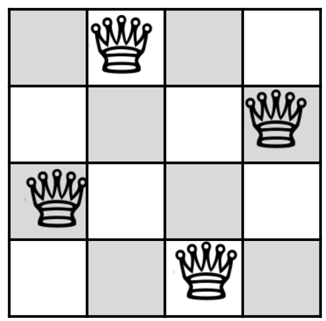
Exemple

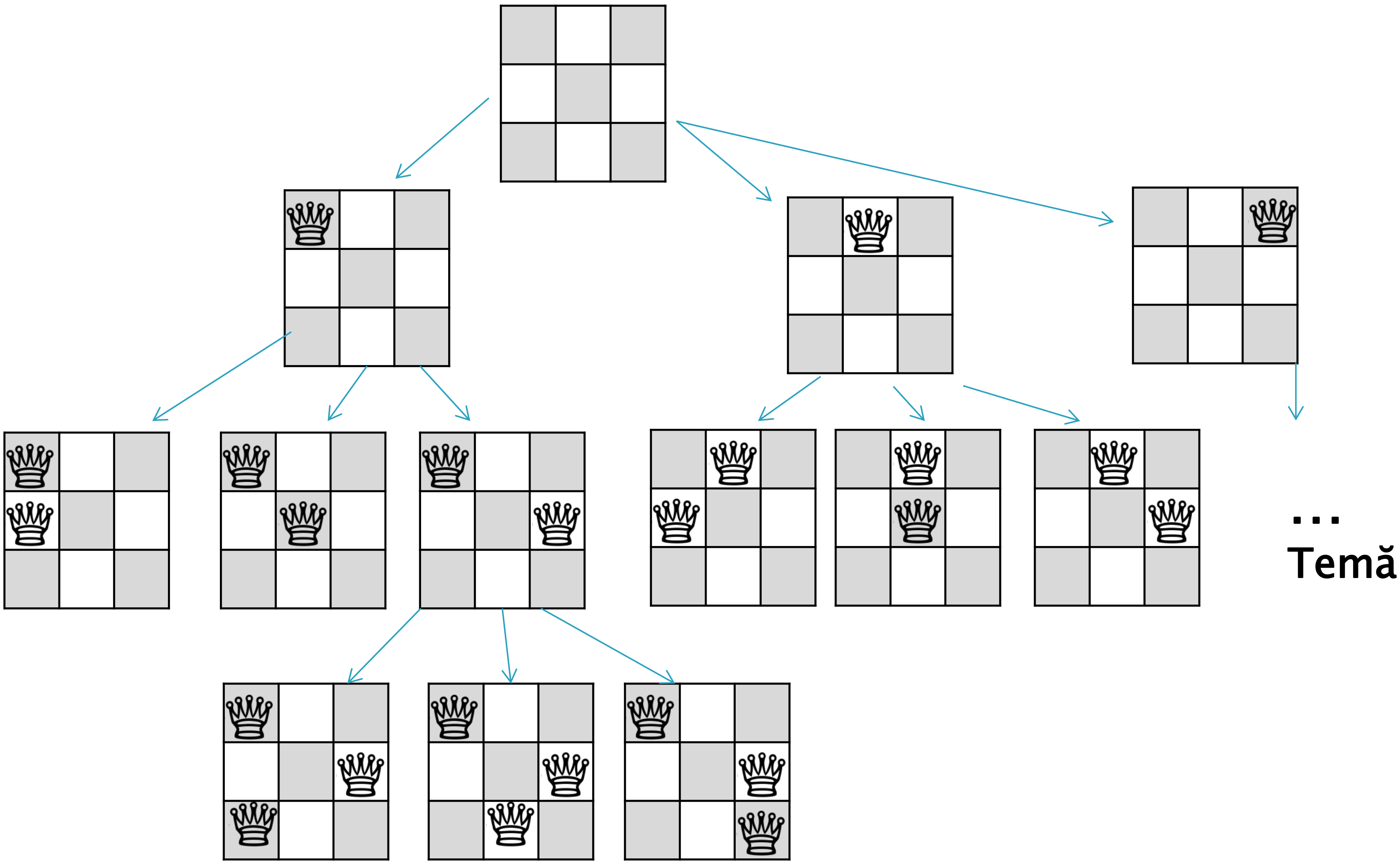
Pentru a testa condiţiile de continuare $\texttt{cont}_k(\texttt{x}_1, \ldots, \texttt{x}_k)$ vom folosi funcţia `cont`

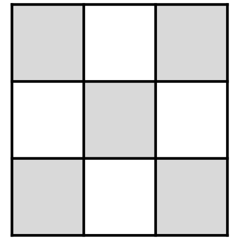
Problema celor n dame

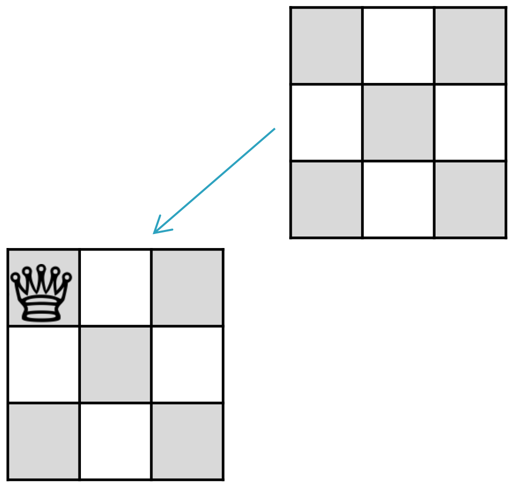
- Se consideră un caroiaj n×n.

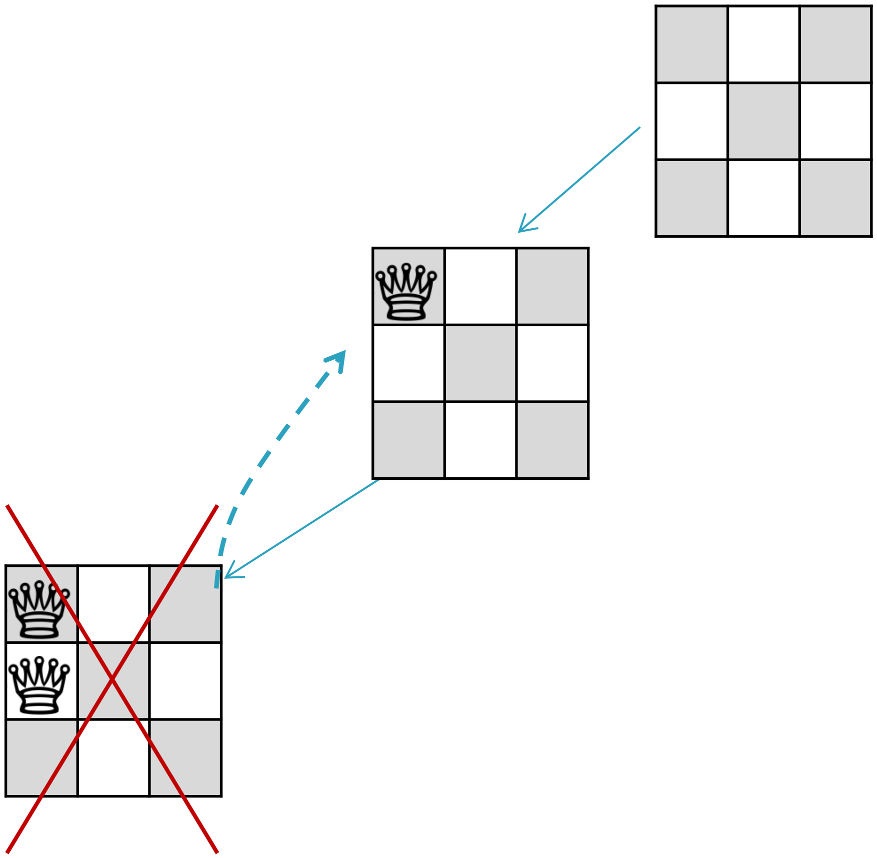
Prin analogie cu o tablă de şah (n=8), se doreşte plasarea a n dame pe pătrăţelele caroiajului, astfel încât să nu existe două dame una în bătaia celeilalte

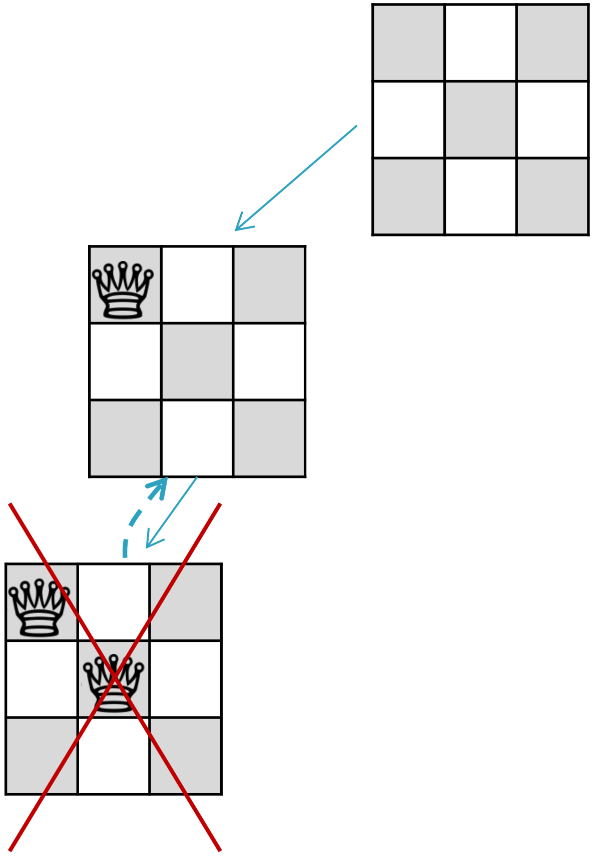


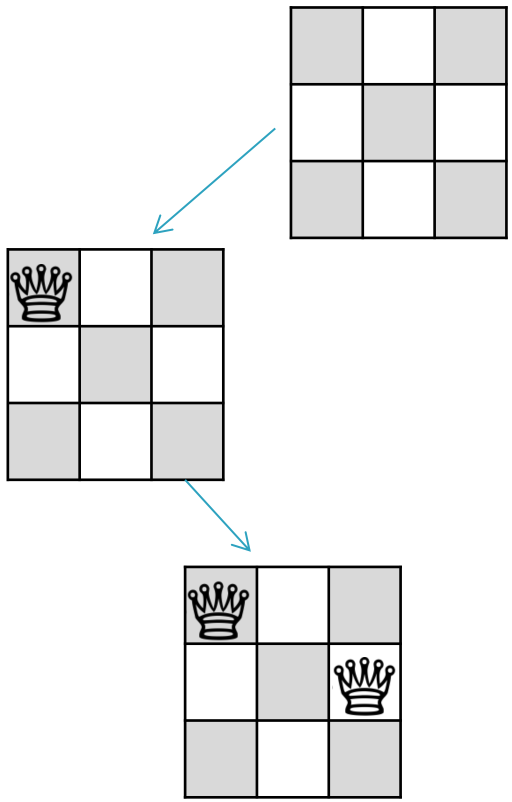

...
Temă

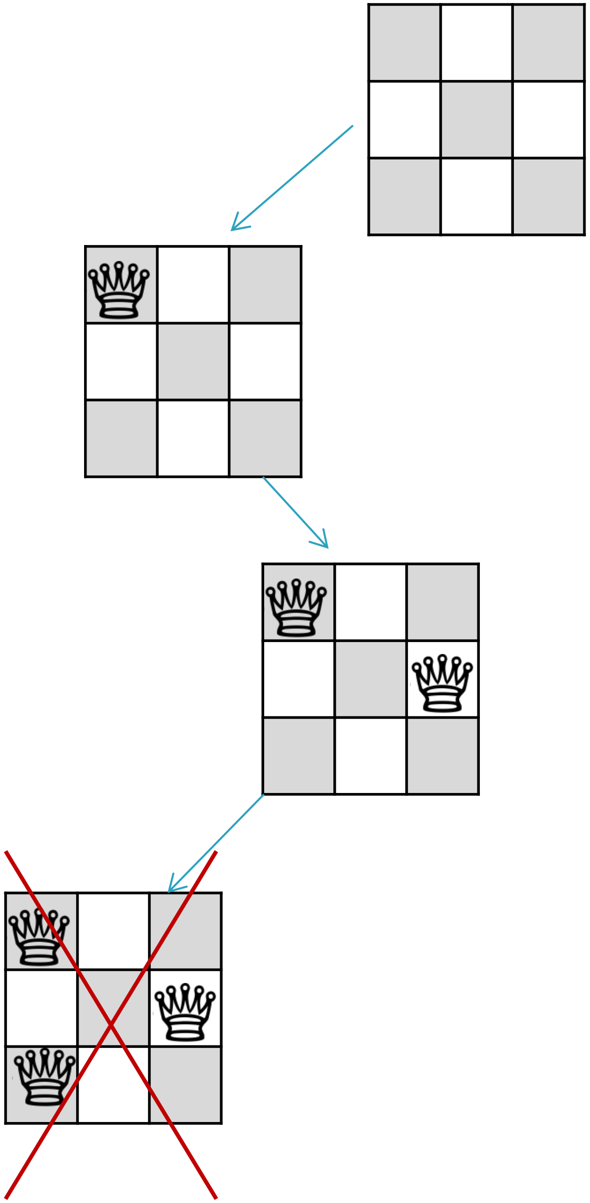


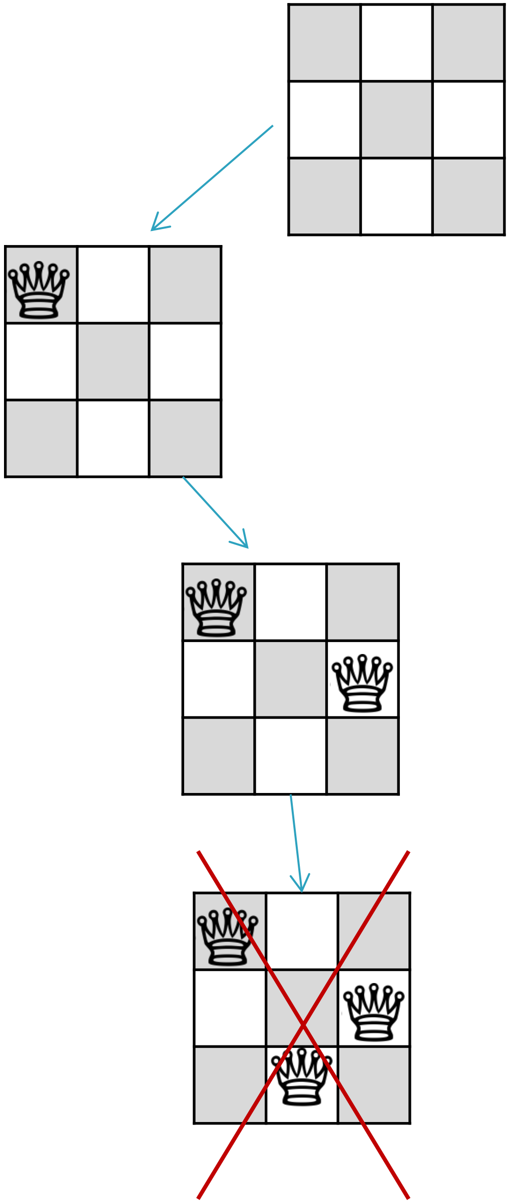


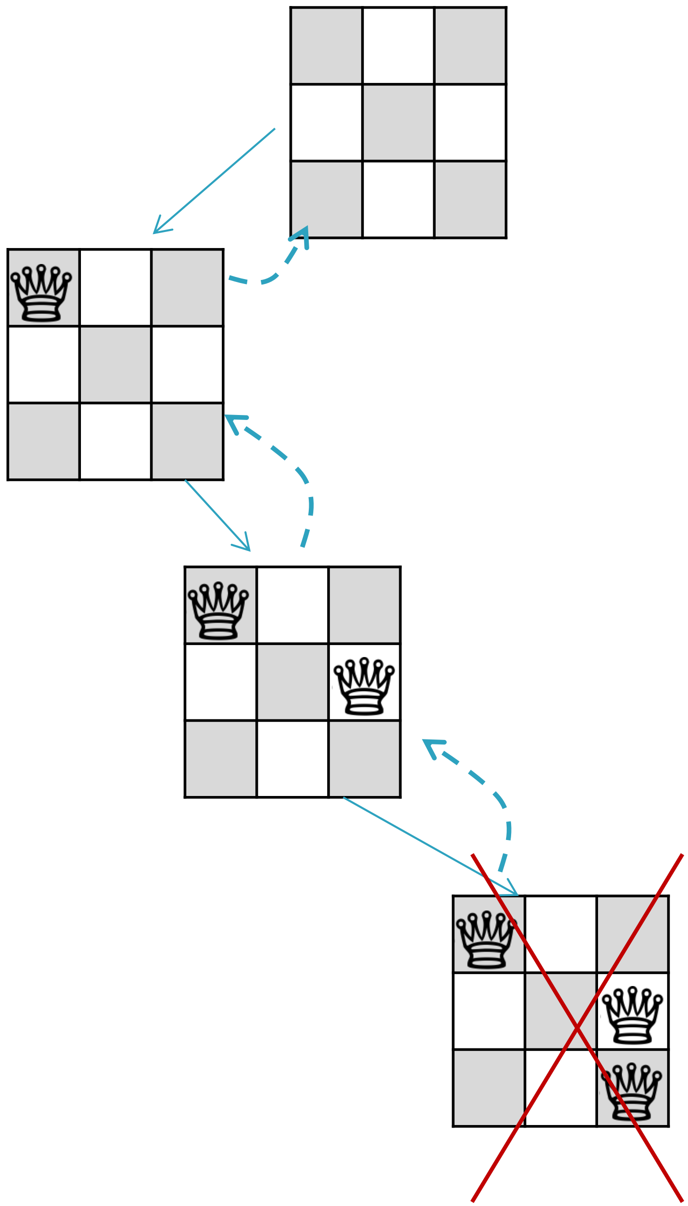


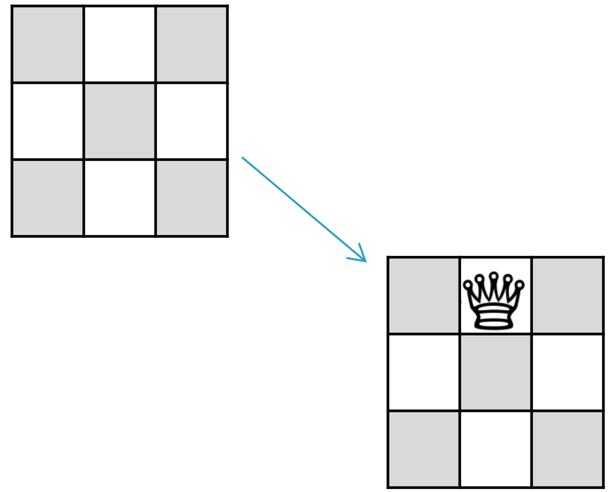


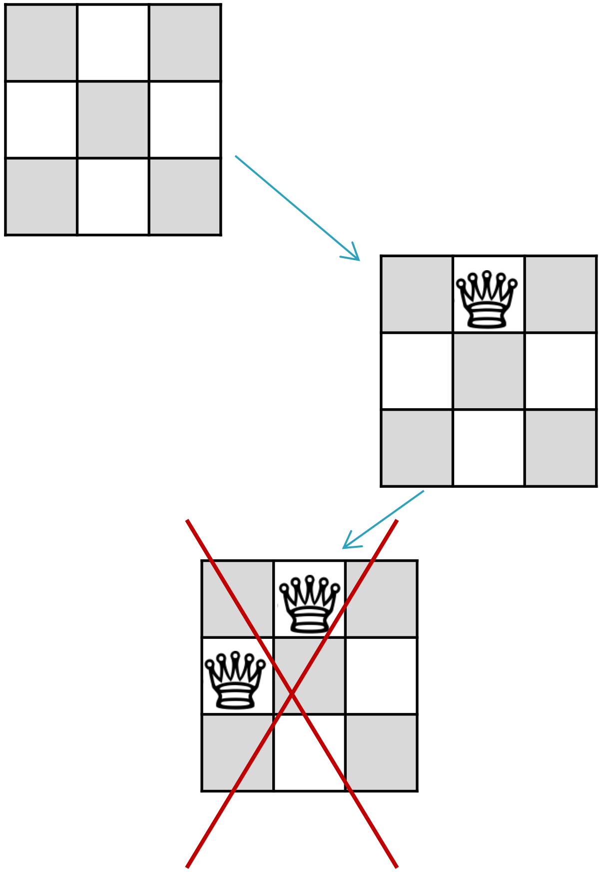


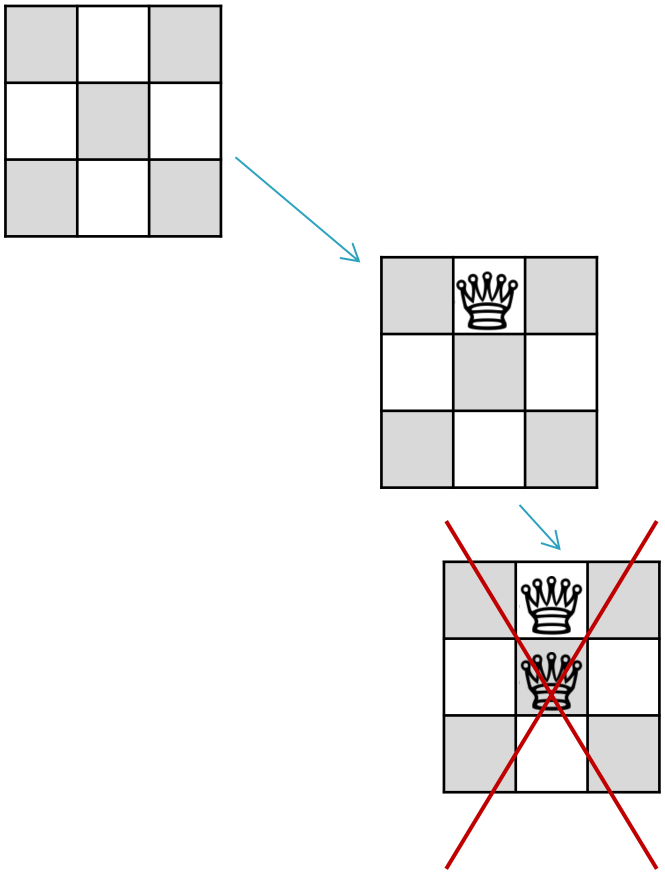


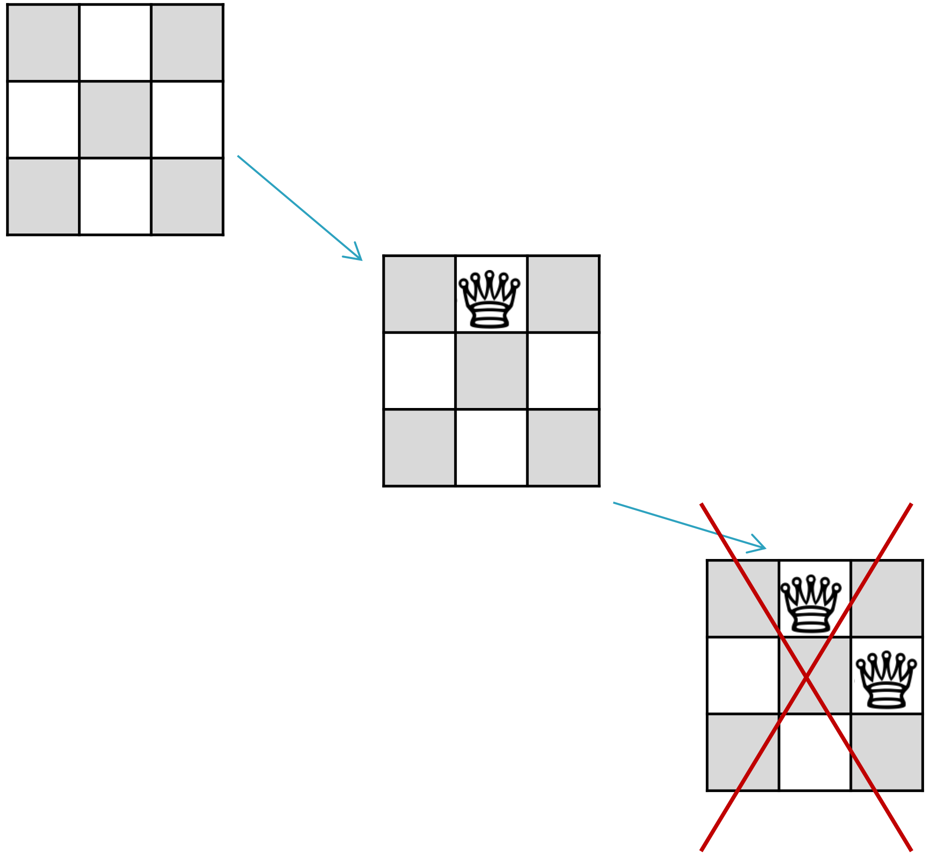


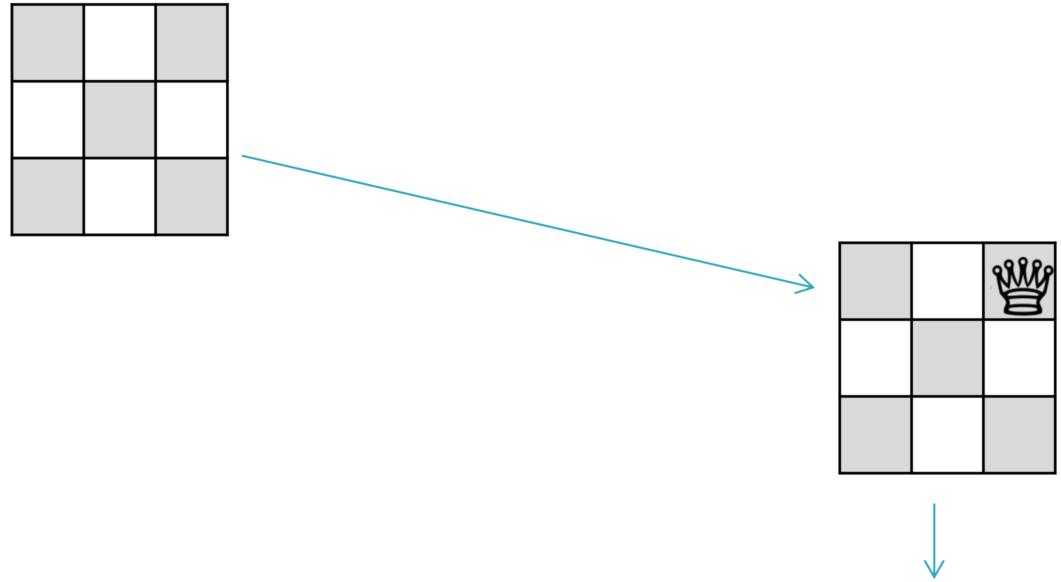












...

Temă

Problema celor n dame

- Reprezentarea soluţiei

- Condiţii interne (finale)

- Condiţii de continuare

Problema celor n dame

- **Reprezentarea soluţiei**

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

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

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

- **Condiţii interne (finale)**

- **Condiţii de continuare**

Problema celor n dame

- **Reprezentarea soluţiei**

  $x = \{x_1, x_2, \ldots, x_n\}$, unde

  x_k = coloana pe care este plasată dama de pe linia k

  $x_k \in \{1, 2, \ldots, n\}$ $(p_k = 1,\ u_k = n)$

- **Condiţii interne (finale)**

  pentru orice $i \neq j$: $x_i \neq x_j$ și $|x_i - x_j| \neq |j - i|$

- **Condiţii de continuare**

Problema celor n dame

- **Reprezentarea soluţiei**

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

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

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

- **Condiţii interne (finale)**

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

- **Condiţii de continuare** – pentru x_k

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

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 şirurile de n paranteze ce se închid corect (n par)

Șiruri corecte de paranteze

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

  - Numărul de şiruri corecte = $C_{n/2}$

    (numerele lui Catalan – v. PD)

Șiruri corecte de paranteze

- **Reprezentarea soluţiei**

  $x = \{x_1, x_2, \ldots, x_n\}$, unde

  $x_k \in \{'(', ')'\}$

- **Condiţii interne (finale)**

  Notăm $dif = nr_(- nr_)$

  $dif = 0$

  $dif \geq 0$ pentru orice secvenţă $\{x_1, x_2, \ldots, x_k\}$

- **Condiţii de continuare**

Șiruri corecte de paranteze

- **Reprezentarea soluţiei**

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

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

- **Condiţii interne (finale)**

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

  $\mathbf{dif} = 0$

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

- **Condiţii de continuare**

  $\mathbf{dif} \geq 0$ -> doar necesar

Șiruri corecte de paranteze

- **Reprezentarea soluţiei**

  $x = \{x_1, x_2, \ldots, x_n\}$, unde

  $x_k \in \{'(', ')'\}$

- **Condiţii interne (finale)**

  Notăm $dif = nr_(- nr_)$

  $dif = 0$

  $dif \geq 0$ pentru orice secvenţă $\{x_1, x_2, \ldots, x_k\}$

- **Condiţii de continuare**

  $dif \geq 0$ -> doar necesar

  $dif \leq n-k$ -> şi 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

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

pentru secvența $x_1 \ldots x_k$ (v. şi problema generării partiţiilor unui număr n)

Metoda Backtracking

- Fie vectorul $a=(a_1,...,a_n)$. Să se determine toate subşirurile crescătoare de lungime maximă –
- **Temă (v. exemple backtracking moodle)**

Metoda Backtracking – SAT

Problema satisfiabilităţii SAT

Considerăm o expresie logică în care apar variabilele $\{x_1, x_2, \ldots, x_n\}$ şi negaţiile lor $\overline{x_i}$. Ştiind că expresia este de forma

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

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)

$$E = (x_1 \vee \overline{x_2}) \wedge (x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (\overline{x_1} \vee x_3) \wedge (\overline{x_4} \vee \overline{x_3})$$

Metoda Backtracking – SAT

- **Problema satisfiabilităţii SAT**

Considerăm o expresie logică în care apar variabilele $\{x_1, x_2, \ldots, x_n\}$ şi negaţiile lor $\overline{x_i}$. Ştiind că expresia este de forma

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

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 $\overline{x_i}$

FNC – formă normală conjunctivă

Metoda Backtracking – SAT

- **Problema satisfiabilităţii SAT**

Considerăm o expresie logică în care apar variabilele $\{x_1, x_2, \ldots, x_n\}$ şi negaţiile lor $\overline{x_i}$. Ştiind că expresia este de forma

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

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 satisfiabilităţii SAT**

$$E = (x_1 \vee \overline{x_2}) \wedge (x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (\overline{x_1} \vee x_3) \wedge (\overline{x_4} \vee \overline{x_3})$$

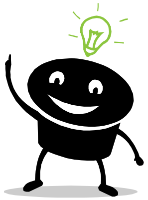
adevărată pentru $x_1 = x_2 = x_3 = 1, \; x_4 = 0$

- 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 şirurile binare $x_1x_2\ldots x_n$ (0 = false, 1 = true) şi verificăm pentru fiecare astfel de şir înlocuind în expresia E dacă E devine adevărată**
- Dacă găsim un şir pentru care expresia este adevărată oprim generarea

Metoda Backtracking – SAT

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

- **Complexitate?**

Metoda Backtracking – SAT

- **Generăm toate şirurile binare $x_1x_2\dots x_n$ (0 = false, 1 = true) şi verificăm pentru fiecare astfel de şir înlocuind în expresia E dacă E devine adevărată**
- Dacă găsim un şir pentru care expresia este adevărată oprim generarea

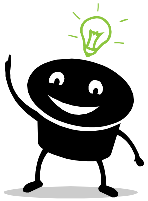
- Dacă expresia nu poate fi satisfăcută, atunci se vor genera şi testa toate şirurile binare de lungime n

$O(2^n m)$

Metoda Backtracking – SAT

- Putem face verificări pe parcurs ⇒ condiţii de continuare?
- Contează pentru performanţă ordinea în care dăm valori variabilelor? ⇒ euristici

Metoda Backtracking – SAT



- ## Condiţii 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 situaţiile:

– devine 1 $\Rightarrow$

– devine 0 $\Rightarrow$

Metoda Backtracking – SAT

- **Condiţii 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 $\overline{x}_k$.

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

– 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

- **Condiţii 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 $\overline{x}_k$.

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

– 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

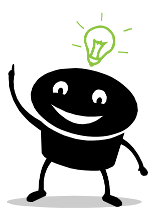
- **Condiţii de continuare**

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

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

– 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 eşuată

Metoda Backtracking - SAT

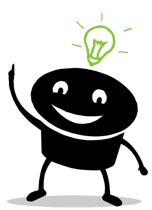
- **Condiţii de continuare**

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

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

- 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 soluţie (restul variabilelor pot primi orice valoare)

Metoda Backtracking – SAT

$$\boxed{(x_2 \vee \overline{x_3}) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2})}$$

x_1:

0

Metoda Backtracking – SAT

$$(x_2 \lor \overline{x_3}) \land (x_1 \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2})$$

x_1: 0

$$(x_2 \lor \overline{x_3}) \land (x_2 \lor x_3) \land (\overline{x_2})$$

x_2: 0

Metoda Backtracking – SAT

$$(x_2 \vee \overline{x_3}) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2})$$

x_1: 0

$$(x_2 \vee \overline{x_3}) \wedge (x_2 \vee x_3) \wedge (\overline{x_2})$$

x_2: 0

$$(\overline{x_3}) \wedge (x_3)$$

Metoda Backtracking – SAT

$$(x_2 \vee \overline{x_3}) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2})$$

x_1: 0

$$(x_2 \vee \overline{x_3}) \wedge (x_2 \vee x_3) \wedge (\overline{x_2})$$

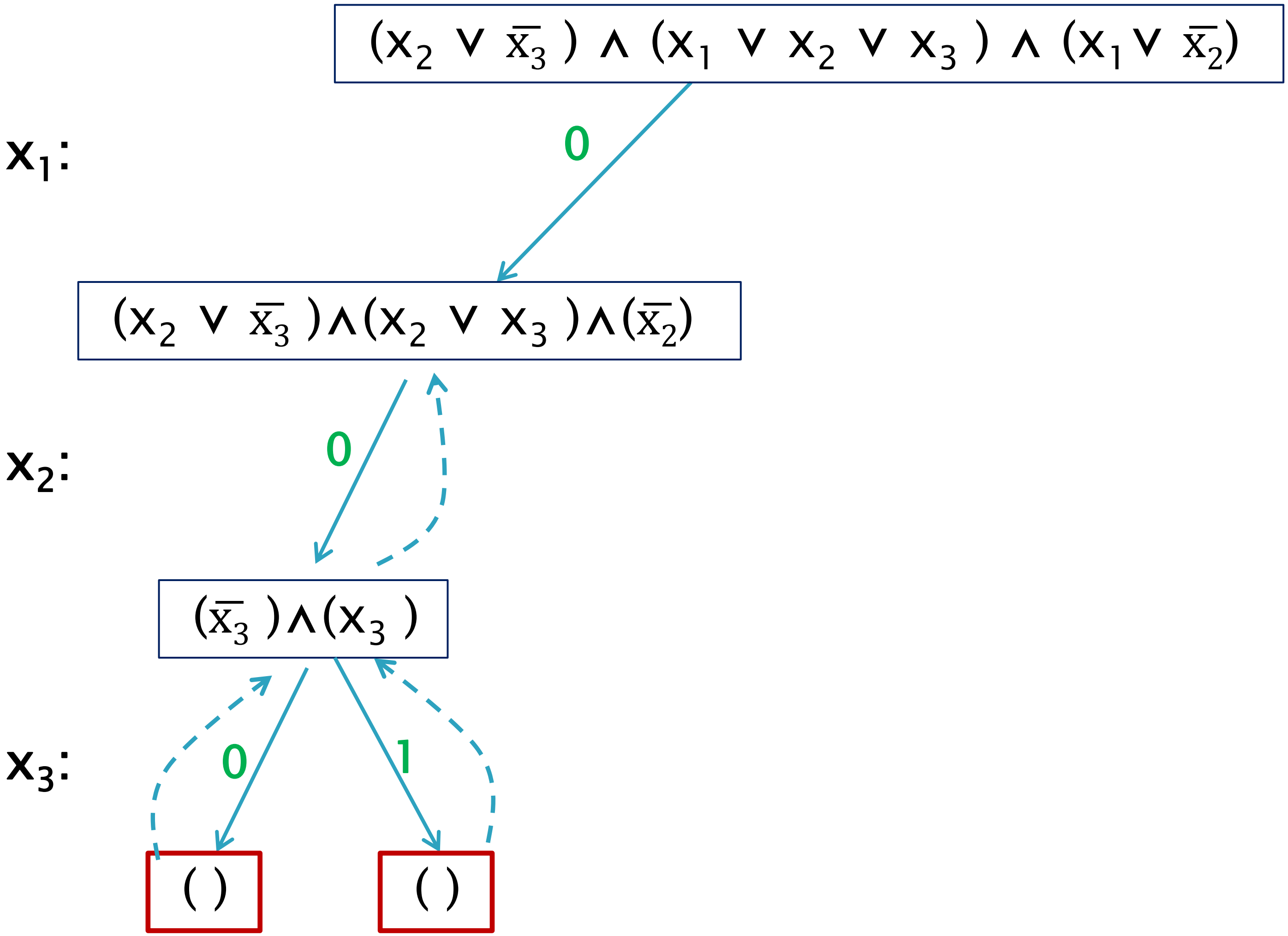
x_2: 0

$$(\overline{x_3}) \wedge (x_3)$$

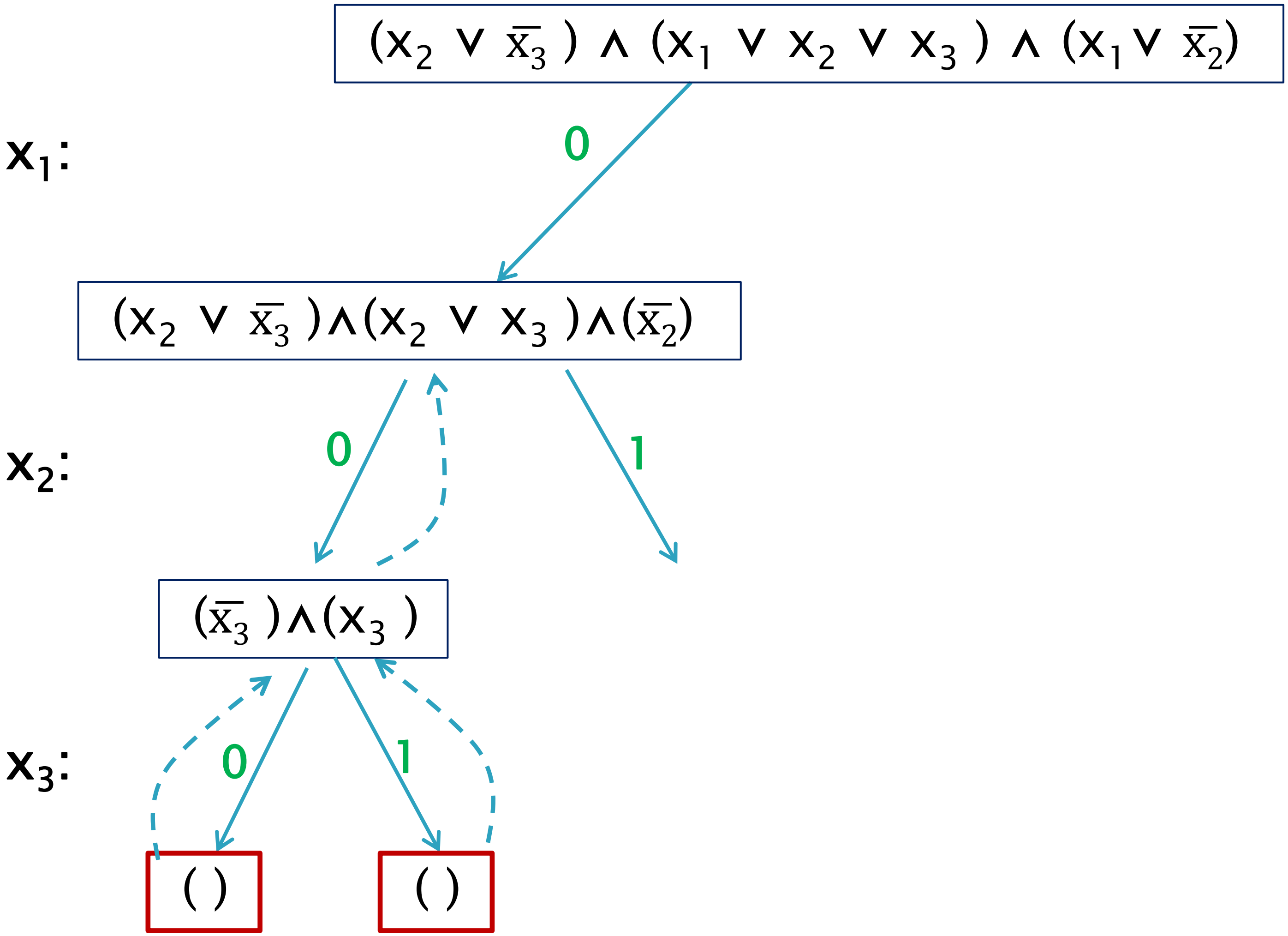
x_3: 0

$$(\ )$$

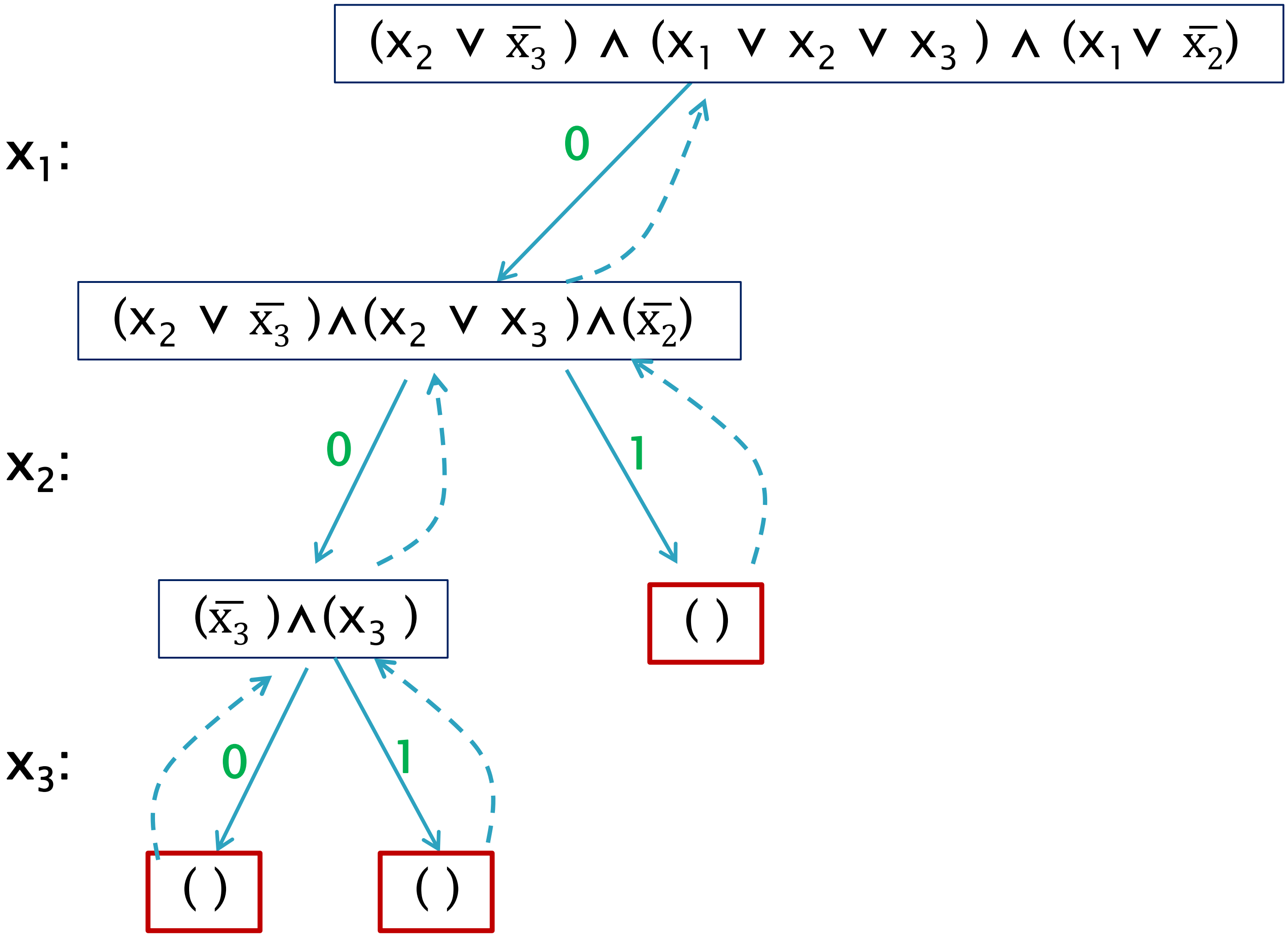
Metoda Backtracking – SAT



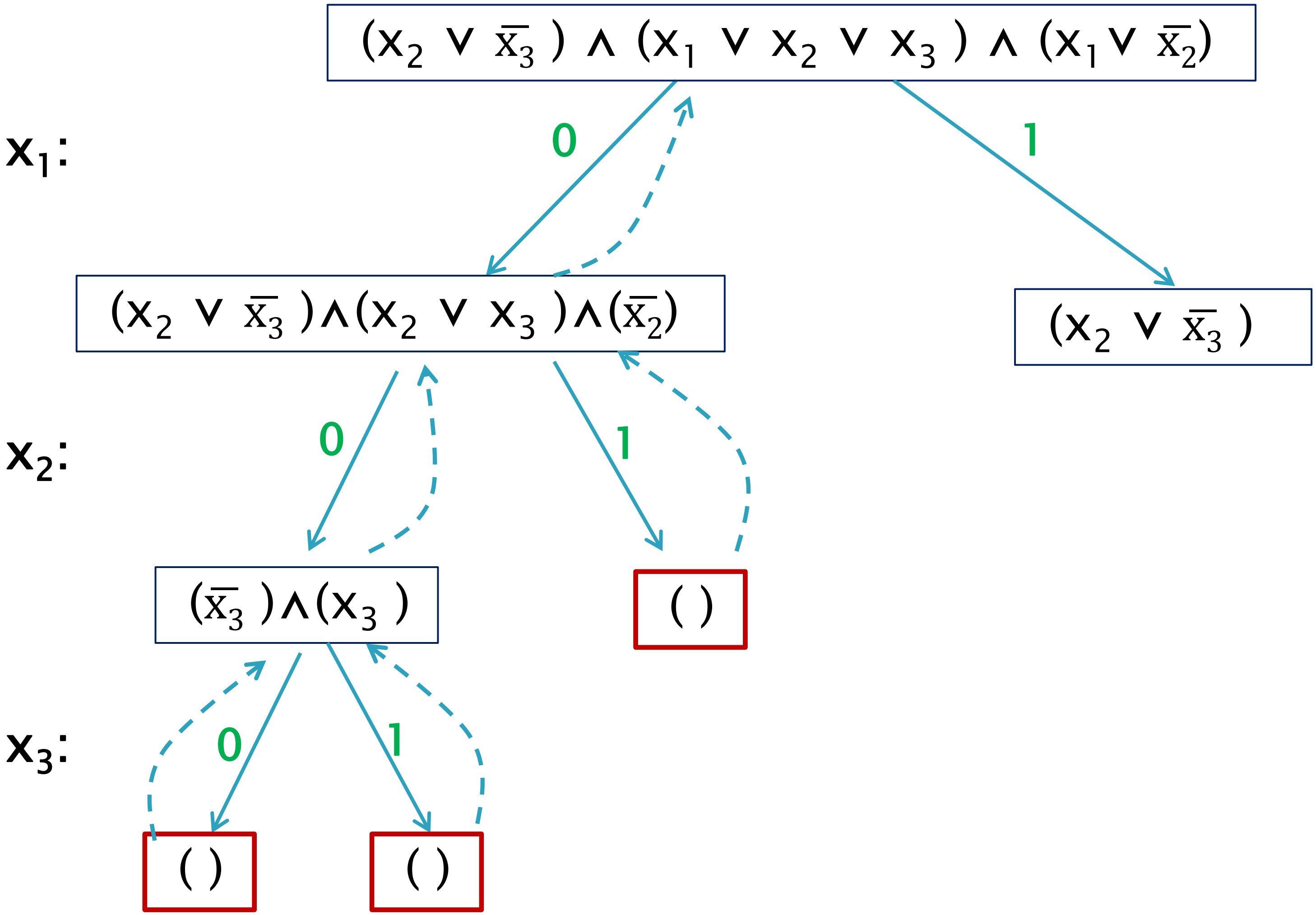
Metoda Backtracking – SAT



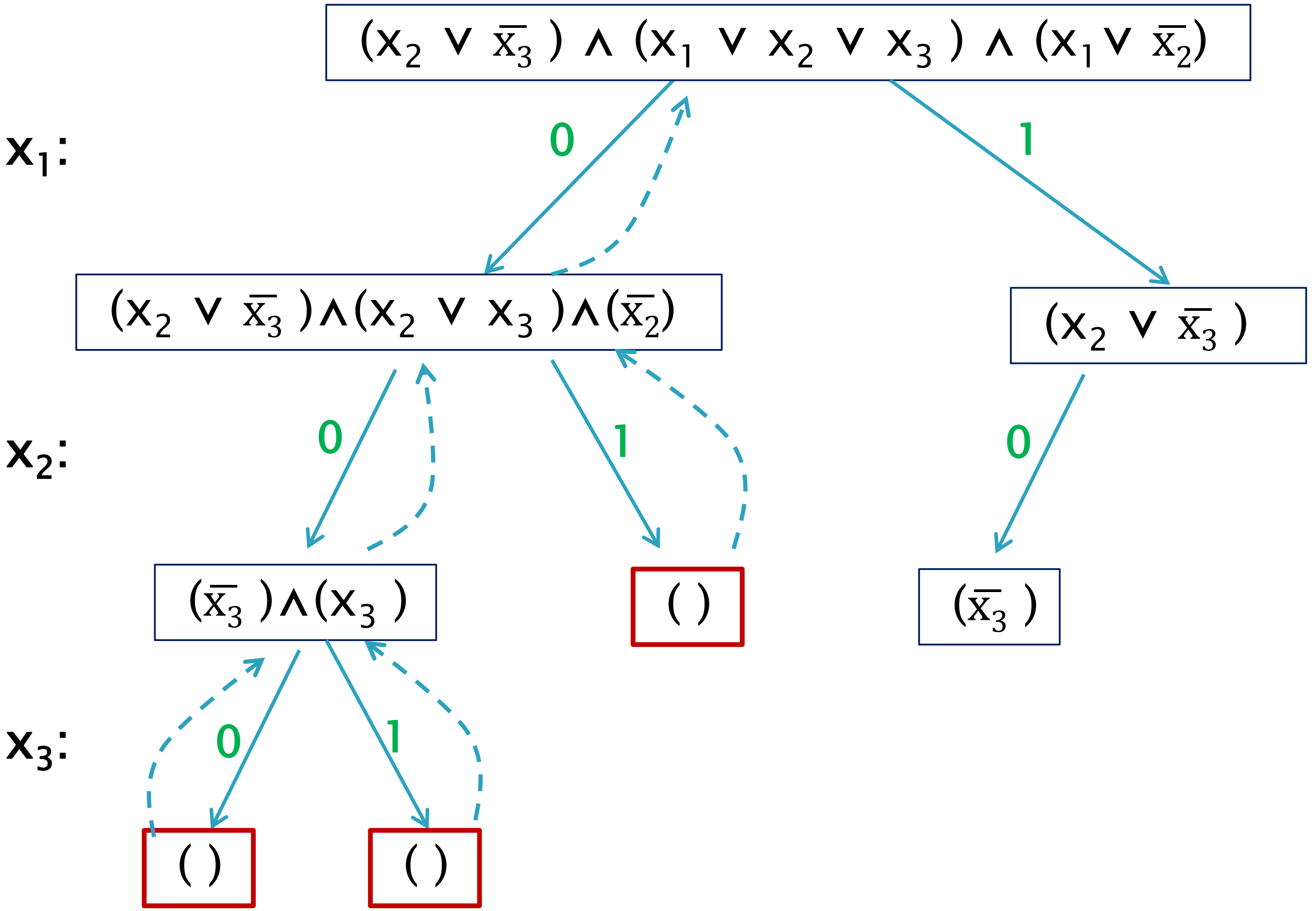
Metoda Backtracking – SAT



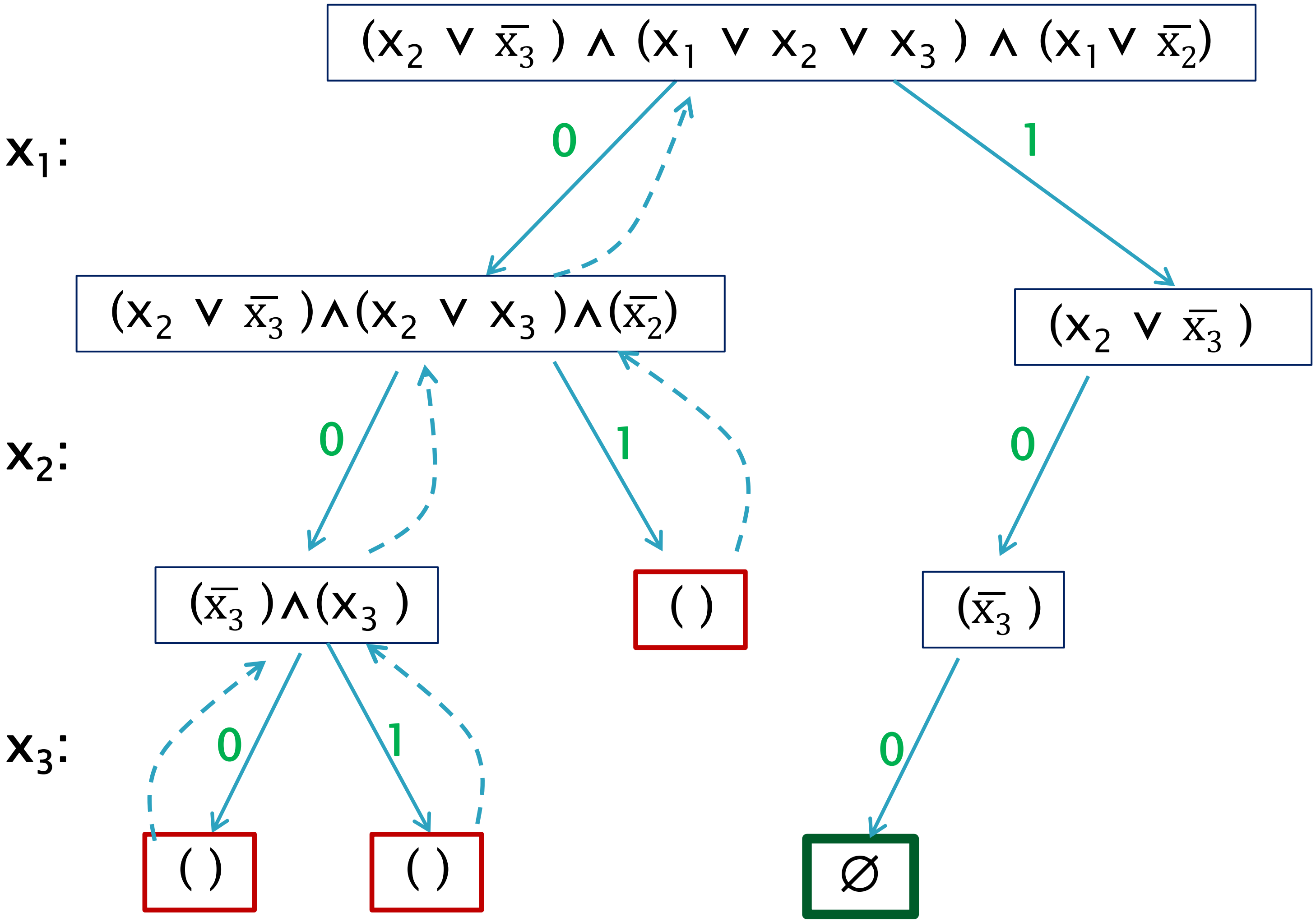
Metoda Backtracking – SAT



Metoda Backtracking – SAT



Metoda Backtracking – SAT



E= $\varnothing$ $\Rightarrow$ Soluţie + STOP

Metoda Backtracking - SAT

- Temă – backtracking pentru SAT /3-SAT

```
procedure back(k, E, V)
    if E = ∅
         scrie true, x STOP
    x_k←false;
    for (C clauza în E care contine x_k)
        daca x̄_k ∈ C atunci
             elimina(E, C)
             back(k+1,E)
             adauga(E, C)
```

```
back(1)

scrie false
```

```
procedure back(k, E, V)
    if E = ∅
         scrie true, x STOP
    x_k ← false;
    for (C clauza în E care contine x_k)
         daca x̄_k ∈ C atunci
              elimina(E, C)
              back(k+1,E)
              adauga(E, C)
         altfel
              reducem(C, k) – eliminam x_k din C
              if C ≠ ∅
                  back(k+1,E)
              restauram(C, k) – reintroducem x_k in C
```

```
back(1)

scrie false
```

```
procedure back(k, E, V)
    if E = ∅
        scrie true, x STOP
    x_k←false;
    for (C clauza în E care contine x_k)
        daca x̄_k ∈ C atunci
            elimina(E, C)
            back(k+1,E)
            adauga(E, C)
        altfel
            reducem(C, k) - eliminam x_k din C
            if C ≠ ∅
                back(k+1,E)
            restauram(C, k) - reintroducem x_k in C
    x_k←true;
```

… SIMILAR

```
back(1)

scrie false
```

Metoda Backtracking - SAT

Posibile îmbunătăţiri:

- Ordinea în care se dau valori variabilelor
- Prelucrări+ formule de logică
- Învăţare din rezultatele deja obţinute

Metoda Backtracking - SAT

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

$\Rightarrow$ 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 = ∅
        scrie true, x STOP
    t ← alegeVariabila(V)
    x_t←false;
    for (C clauza în E care contine x_t)
        daca x̄_t ∈ C atunci
            elimina(E, C)
            back(k+1,E, V-{t})
            adauga(E, C)
        altfel
            reducem(C, t)
            if C ≠ ∅
                back(k+1,E, V-{t})
            restauram(C, t)
    x_t←true;
    … SIMILAR
```

```
back(1, E, {1,2,…,n})

scrie false
```

Metoda Backtracking – SAT

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

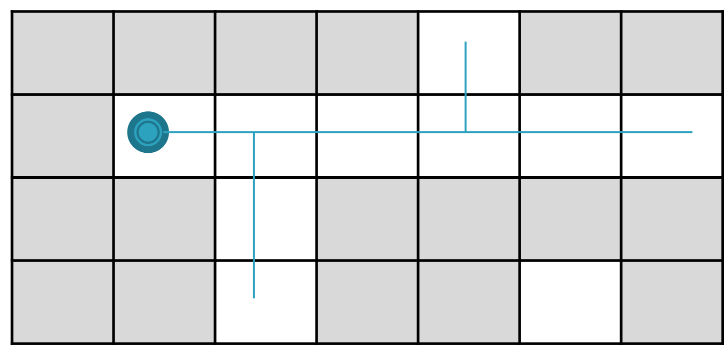
- Algoritmul Davis-Putnam

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

Backtracking în plan

- **Labirint.** Se consideră un caroiaj (matrice) A cu m linii şi n coloane. Poziţiile pot fi:
  - libere: $a_{ij}=0$;
  - ocupate: $a_{ij}=1$.

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



Variante:

- drumul maxim
- drumul minim

Backtracking –trasee în plan

- **Labirint.** Se consideră un caroiaj (matrice) A cu m linii şi n coloane. Poziţiile pot fi:
  - libere: $a_{ij}=0$;
  - ocupate: $a_{ij}=1$.

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

Variante:

- drumul maxim
- drumul minim – **NU Backtracking**

Backtracking –trasee în plan

- **Labirint.** Se consideră un caroiaj (matrice) A cu m linii şi n coloane. Poziţiile pot fi:
  - libere: $a_{ij}=0$;
  - ocupate: $a_{ij}=1$.

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

Variante:

- drumul maxim
- drumul minim – NU Backtracking

  -> parcurgerea în lăţime

Backtracking în plan

Indicaţii

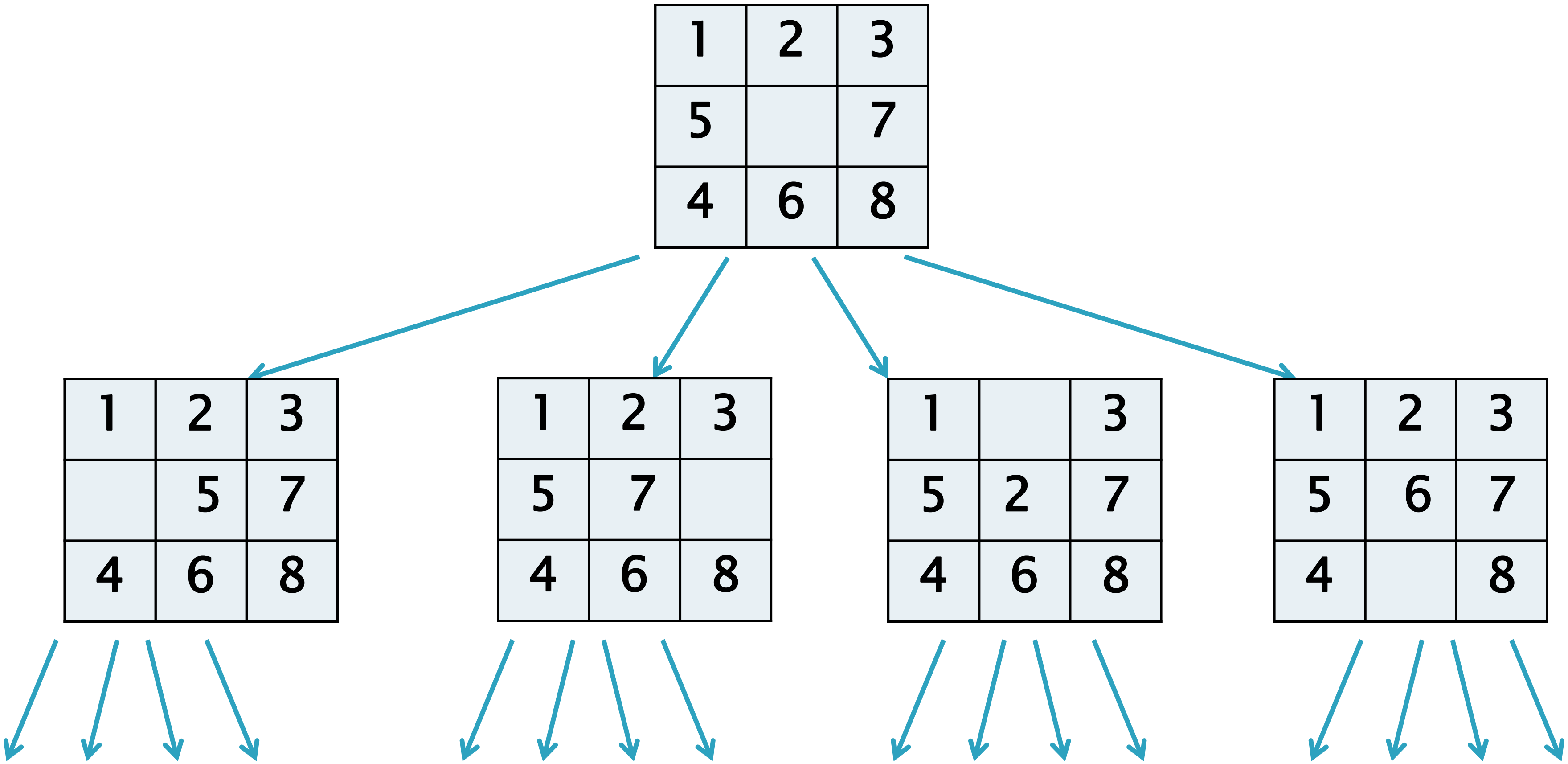
- Matricea deplasărilor `depl` cu două linii şi `ndepl` coloane :

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

- **dacă poziţia este liberă şi putem continua o marcăm şi** continuăm
- **repunem a_{ij}=0 la întoarcere** (din recursivitate)
- Bordăm matricea cu 2 (pentru a testa mai uşor ieşirea)
  - **V. moodle (slide curs exemple backtracking+surse)**

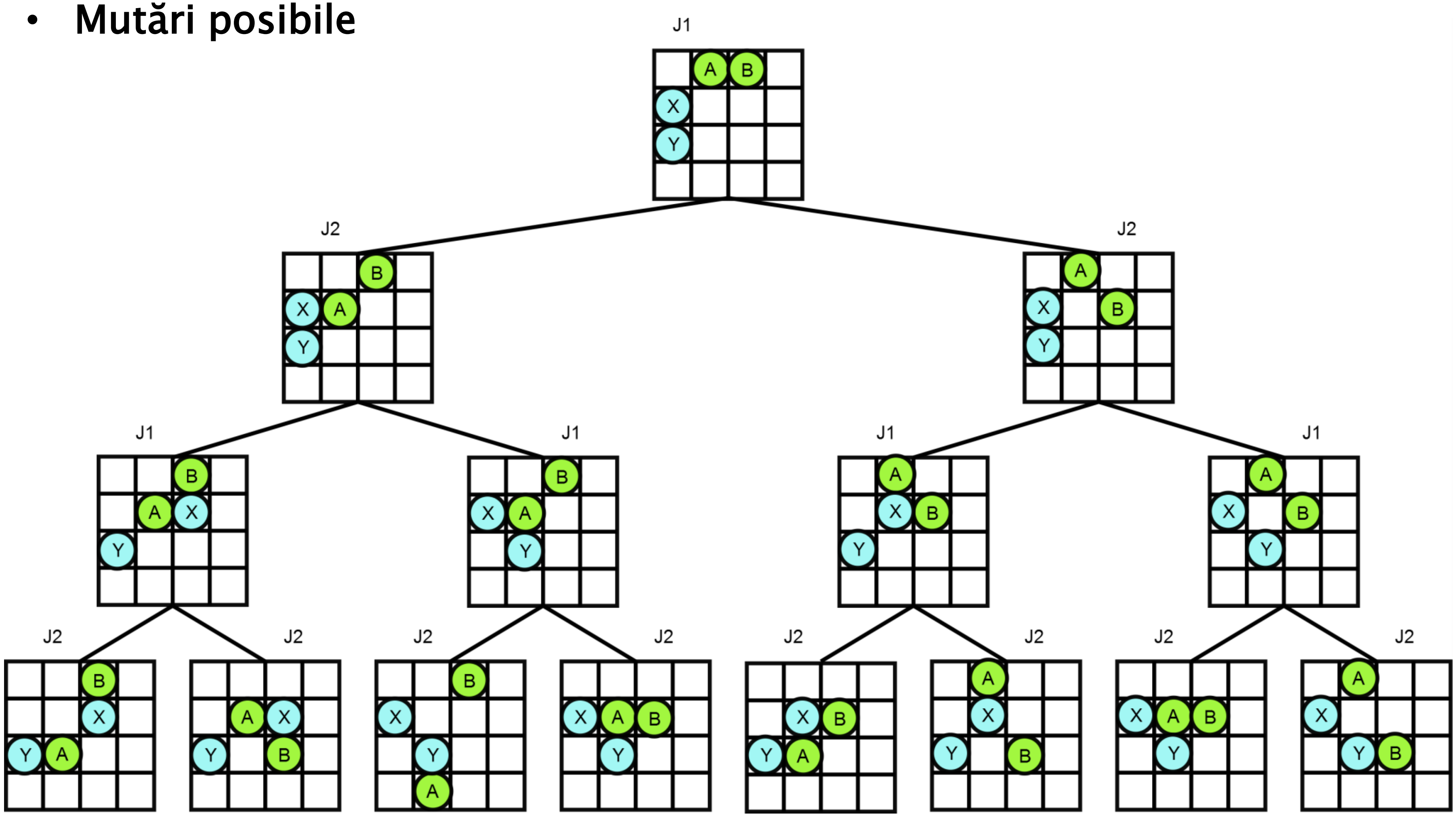
Backtracking

- **Arbori asociaţi mutărilor într-un joc**
  - Pentru dimensiuni mici
  - În multe cazuri arborele poate deveni de dimensiune mare şi un algoritm backtracking va fi lent
  - **Trebuie alte strategii de generare şi 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 asociaţi mutărilor într-un joc**
  - **Temă (suplimentar)** – Determinaţi o soluţie (la un joc de o persoană) sau o strategie de câştig (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