

Metoda Backtracking

~~~~~

Se aplică pentru problemele în care soluția poate fi reprezentată sub forma unui vector  $\mathbf{x}=(x[1], x[2], \dots, x[k])$ .

$$X \in S[1] \times S[2] \times \dots \times S[n]$$

$S[1] \times S[2] \times \dots \times S[n]$  - se numește spațiul soluțiilor posibile.

$$|S[i]| = s[i], \quad S[i] = \{a[i,1], a[i,2], \dots, a[i,s[i]]\}$$

Între componentele vectorului  $X$  sunt date mai multe **condiții interne**, care depind de natura problemei date. **Soluțiile posibile** pentru care sunt verificate condițiile interne le numim **soluții rezultat**. În unele probleme se cere alegerea dintre soluțiile rezultat a **soluției optime** după un anumit criteriu.

### Observații:

Astfel de probleme pot fi rezolvate prin generarea tuturor elementelor produsului cartezian  $S[1] \times S[2] \times \dots \times S[n]$  (întreg spațiul soluțiilor), dar această metodă nu este eficientă. Această metodă se numește **metoda forței brute - brute-force**.

**Soluție parțială** = o soluție pentru care sunt indeplinite condițiile de continuare. Condițiile de continuare sau de validare se obțin din condițiile interne aplicate pentru o soluție parțială.

### Construirea soluției:

- \* Soluția se construiește sub forma unei stive, astfel încât la fiecare pas se alege o valoare pentru elementul curent  $x[k]$  (soluția se construiește progresiv). La început  $k=1$  și  $x[1]$  se inițializează **cu o valoare anterioară primei valori** din  $S[1]$ .
- \* Cât timp stiva nu este vidă, completăm soluția parțială cu noi componente. Elementului  $x[k]$  din vârful stivei i se atribuie următoarea valoare din mulțimea  $S[k]$  (dacă există). Pentru soluția parțială obținută  $(x[1], x[2], \dots, x[k])$  sunt verificate **condițiile de validare**. Apar astfel două posibilități:
  - 1) Condiția de continuare este **adevărată**:
    - în acest caz testăm egalitatea  $k==n$ , adică verificăm dacă avem valori pentru toate componentele vectorului soluție. În caz afirmativ, am obținut o soluție a problemei, pe care o reținem.
    - În cazul  $k < n$  incrementăm  $k$  cu 1 și inițializăm  $x[k]$  cu o valoare anterioară primei valori din mulțimea  $S[k]$ .
  - 2) În situația în care condiția de continuare este **falsă**, alegem pentru  $x[k]$  următoarea valoare posibilă din  $S[k]$  și verificăm din nou condițiile de validare. Dacă nu se poate face altă alegere, revenim pe stivă la nivelul anterior și verificăm următoarea valoare pentru  $x[k-1]$ .

Pentru orice problemă particulară care este rezolvată prin metoda backtracking trebuie precizate:

- (1) **spațiul soluțiilor**, adică mulțimile  $S[1], S[2], \dots, S[n]$ .  
Nu este necesar ca aceste mulțimi să aibă același număr de elemente.
- (2) **condițiile de validare** a unei soluții parțiale  $(x[1], x[2], \dots, x[k])$ .  
Alegerea acestor condiții influențează **decisiv eficiența metodei**.

### Procedura generală backtracking iterativă

```
1.  int x[31],n;
2.
3.  void back()
4.  {
5.    int k=1;
6.    x[k]=init(k);
7.    while (k>0)
8.    {
9.      while (există o valoare netestată în Sk)
10.     {
11.       x[k]=urmator(k);
12.       if valid(x[1],x[2],...,x[k])
13.         if (k==n) solutie(x[1],x[2],...,x[n])
14.         else {
15.           k=k+1;
16.           x[k]=init(k);
17.         }
18.     }
19.     k=k-1;
20. }
21. }
```

### Procedura generală backtracking recursivă

```
1.  int x[31],n;
2.
3.  void back(int k)
4.  {
5.    x[k]:=init(k);
6.    while (are_succesor(x[k]))
7.      if valid(k)
8.        if (este_solutie(k)) solutie();
9.        else back(k+1);
10. }
```

\* O variantă recursivă în cazul în care toate mulțimile  $S[k]$  sunt egale,  
 $S[k]=\{1, 2, \dots, n\}$   $k=1, n$  este următoarea:

```
1.  void back(int k)
2.  {
3.    int i;
4.    for(i=1;i<=n;i++)
5.    {
6.      x[k]=i;
7.      if (valid(k))
8.        if (este_solutie(k)) solutie();
9.        else back(k+1);
10. }
11. }
```

## **Probleme propuse:**

1. Generarea produsului cartezian a  $n$  mulțimi.
2. Generarea permutărilor de  $n$  elemente.
3. Problema damelor:  
Să se afișeze toate posibilitățile de așezare a 8 ( $n$ ) regine pe o tablă de șah  $8 \times 8$  ( $n \times n$ ) în așa fel încât să nu se atace reciproc.
4. Generarea combinărilor de  $n$  elemente luate câte  $k$ .
5. Generarea funcțiilor injective (aranjamente).  
Să se afișeze toate funcțiile injective  $f: A \rightarrow B$ , unde  $A$  și  $B$  sunt două mulțimi cu  $m$  respectiv  $n$  elemente.
6. Să se afișeze toate partițiile unui număr natural nenul  $n$ .  
Prin partiție a lui  $n$  se înțelege o descompunere a lui  $n$  ca sumă de numere naturale nenule.
7. Plata unei sume de bani cu bancnote de valori date  
Să se afișeze toate modalitățile de a plăti o sumă de bani  $S$  cu bancnote de valori  $b_1 > b_2 > \dots > b_n$ . Să se determine modalitățile de plată cu număr minim de bancnote.
8. Colorarea hărților.  
S-a demonstrat că orice hartă plană poate fi colorată cu maximum 4 culori. Pentru o hartă plană dată să se afișeze toate modalitățile de colorare a hărții cu maxim 4 culori.

Indicație: Relația de vecinătate a două țări de pe hartă se păstrează sub forma unei matrici pătratică  $T$ , ale cărei elemente au valorile 0 sau 1:

$$T_{i,j} = \begin{cases} 0, & \text{dacă țara } t_i \text{ nu este vecină cu țara } t_j \\ 1, & \text{țara } t_i \text{ nu este vecină cu țara } t_j \end{cases}$$

Observații:

- i) Matricea  $T$  este o matrice pătratică.
  - ii) Pe diagonala principală elementele au valoarea 0 (țara  $t_i$  nu este vecină cu ea însăși).
  - iii) Matricea  $T$  este simetrică față de diagonala principală.
9. Să se descompună numărul natural  $n$  în toate modurile posibile ca sumă de exact  $p$  numere naturale nenule ( $p \leq n$ ).
  10. Să se descompună numărul natural  $n$  în toate modurile posibile ca sumă de numere prime.
  11. Problema determinării unui subșir crescător maximal.  
Se dă un șir de numere întregi. Să se determine un subșir crescător de lungime maximă.

## 12. Problema comis-voiajorului

Un comis-voiajor trebuie să viziteze  $n$  orașe. Inițial, pornește din orașul 1, parcurge toate orașele fără să treacă de două ori prin același oraș astfel încât la întoarcere să ajungă înapoi în orașul 1.

Cunoscând drumurile de legătură existente între orașe să se tiparească toate rutele posibile pe care le poate urma comis-voiajorul.

$$A_{i,j} = \begin{cases} 0, & \text{dacă } i \text{ nu este vecin cu } j \\ 1, & \text{dacă } i \text{ este vecin cu } j \end{cases}$$

## Backtracking în plan

### 13. Problema labirintului

Se dă un labirint sub forma unei matrici de tip  $m \times n$  cu elemente 0 (liber) și 1 (zid). Să se afle toate drumurile de la un punct de intrare în labirint  $(i_s, j_s)$  la un punct de ieșire din labirint  $(i_e, j_e)$ , aflat la marginea labirintului - pe prima sau ultima linie/coloană.

### 14. Problema bilei

Un teren este reprezentat sub forma unei matrici  $T$  de tip  $m \times n$ .

Elementele  $T[i,j]$  ale matricii reprezintă cotele unor porțiuni din teren.

O bilă se găsește pe o anumită porțiune  $(i_b, j_b)$  cu o anumită cotă.

Să se determine toate traseele posibile ale bilei pentru a ieși de pe teren, știind că bila se poate deplasa pe orice porțiune de teren învecinată cu o cotă strict mai mică decât cota la care se află bila.

### 15. Problema calului

Pe o tablă de șah se așează un cal într-o căsuță a tablei. Să se precizeze toate drumurile pe care le poate urma calul prin săritură astfel încât să fie parcurse toate căsuțele tablei.