Metoda Backtracking

Metoda **Backtracking** este, alături de **Divide et Impera,** o metodă generală de elaborare a algoritmilor. Ea se aplică problemelor în care soluţia se poate reprezenta sub forma unui vector **X=(x1, x2,..., xn) ∈ A1xA2x…xAn**, unde **Ai,** cu **i ∈ 1,n**, sunt mulţimi finite, iar elementele lor se află într-o relaţie de ordine bine stabilită.

Descrierea metodei

Elementele vectorului x primesc pe rand valori, în sensul că lui x[k] i se atribuie o valoare numai dacă au fost deja atribuite valori elementelor x[1], x[2], ..., x[k-1]. Mai mult, după ce lui x[k] i s-a atribuit o valoare se verifică aşa numitele **condiţii de continuare** pentru x[1], x[2], ..., x[k] (neîndeplinirea lor exprimând faptul că oricum am alege x[k+1],... x[n] nu vom ajunge la o soluţie). Numai dacă aceste condiţii sunt îndeplinite se trece la alegerea lui x[k+1]. În caz contrar, fie i se dă lui x[k] următoarea valoare posibilă din mulţimea Ak, fie (dacă mulţimea Ak a fost epuizată) iese x[k] din stivă şi se revine la x[k-1], încercându-se o nouă alegere pentru acesta. De aici vine şi numele metodei (cănd nu se poate avansa în soluţia curentă se revine pe nivelul anterior).

Exemplu: Să se genereze toate permutările mulţimii {1,2, … , n}. Explicarea mecanismului metodei pentru n=3.

Obs. Metoda poate fi implementată iterativ sua recursiv.

Backtracking recursiv

1) Generarea permutărilor mulţimii {1,2,…n}.

```
int x[100], n; // variabile globale

void afis (int k)  // afiseaza solutia curenta, adică vectorul x cu k elemente
{    int i;
     for (i=1; i<=k; i++) cout <<x[i]<<" ";
     cout <<endl;
 }

 int valid (int k) // verifica conditiile de continuare pentru x[k]
{ int i;
   for (i=1; i<k; i++)
       if (x[k]==x[i]) return 0;
    return 1;
  }

 void bkt (int k)    // k = indicele elementului din vârful stivei
 {   int i;
     for (i=1; i<=n; i++)
       { x[k]=i;  // i se da lui x[k] urmatoarea valoare din multimea de valori posibila
          if (valid(k))  // daca x[k] satisface conditiile de continuare
               if (k==n) afis(k);  // daca s-a ajuns la o solutie se afiseaza
               else bkt(k+1);  // altfel, se trece la alegerea lui x[k+1] prin autoapel
        }
   }
```

Apel: bkt(1);

- **Modelarea problemei**

I) **x[k] ∈ {1,2,…,n} , oricare ar fi k=1,n** **// for (i=1; i<=n; i++)** din funcţia **bkt()**

II) **Condiţii de continuare pentru x[k]** **//** funcţia **valid()**
elemente distincte
x[k] ≠ x[i], oricare ar fi i=1,k-1

III) S-a ajuns la o soluţie dacă k=n **//** condiţia din functia **bkt()** pe care apelez **afis()**

Obs.
1) **Permutări pe un vector oarecare a=(a1, a2, ….an)**
X=(1,2,3) -> a1, a2, a3
X=(1,3,2) -> a1, a3, a2
….
X=(3,2,1) -> a3, a2, a1

În afis() : cout <<a[x[i]]; // se afiseaza elementele cu indicii permutati

2) **Numarare soluţii** – in variabila *nr* globala
nr++; **// în funcţia afis()**

Principiul de funcţionare corespunzător nivelului k al stivei este:

- i se da lui x[k] urmatoarea valoare din multimea de valori posibila (*x[k]=i*)
- se verifică dacă x[k] satisface condiţiile de continuare. În caz afirmativ, se verifică dacă s-a ajuns la o soluţie şi dacă da se afişează. Dacă nu s-a ajuns încă la o soluţie se trece la nivelul următor, adică la alegerea lui x[k+1], prin autoapel.
- dacă x[k] nu este valid (nu satisface condiţiile de continuare), fie primeşte următoarea valoare din mulţimea de valori posibilă (ia următoarea valoare din *for*), fie se revine pe nivelul anterior (nivelul *k-1*) prin mecanismul de descărcare al stivei specific recursivităţii.

Algoritmul se încheie când stiva devine vidă.

Observaţii asupra metodei

1) Metoda Backtracking generează **toate soluţiile** problemei. În cazul în care se doreşte o singură soluţie, se poate forţa oprirea atunci când a fost găsită, cu instrucţiunea *exit(0)*, care determină ieşirea forţată din program.

exit(0); **// în fişierul antet <cstdlib>**

2) **Soluţiile sunt generate în ordine lexicografică**.

3) **Condiţiile de continuare** pe care le stabilim derivă din condiţiile interne ale problemei.

4) **Algoritmul metodei este exponenţial - O(2^n), deci metoda este ineficientă din punct de vedere al timpului de executare**. De aceea nu o utilizăm decât atunci când nu avem la dispoziţie alt algoritm mai eficient, sau când dorim generarea tuturor soluţiilor unei probleme.