Metoda Backtracking - Prezentarea metodei

Metoda **Backtracking**, numită şi metoda **„căutării cu revenire”** se utilizează pentru determinarea unei submulţimi a unui produs cartezian $S_1 \times S_2 \times ... \times S_n$, unde S_i sunt mulţimi finite. Fiecare element al submulţimii poate fi văzut ca o soluţie a unei probleme concrete. O soluţie este de forma unui vector $x=(x_1, x_2 ,..., x_n) \in S_1 \times S_2 \times ... \times S_n$, unde $| S_i | = s_i$ elemente.

În majoritatea cazurilor nu orice element al produsului cartezian este soluţie, ci doar cele care satisfac anumite restricţii sau relaţii între componentele vectorului x, numite ***condiţii interne***.

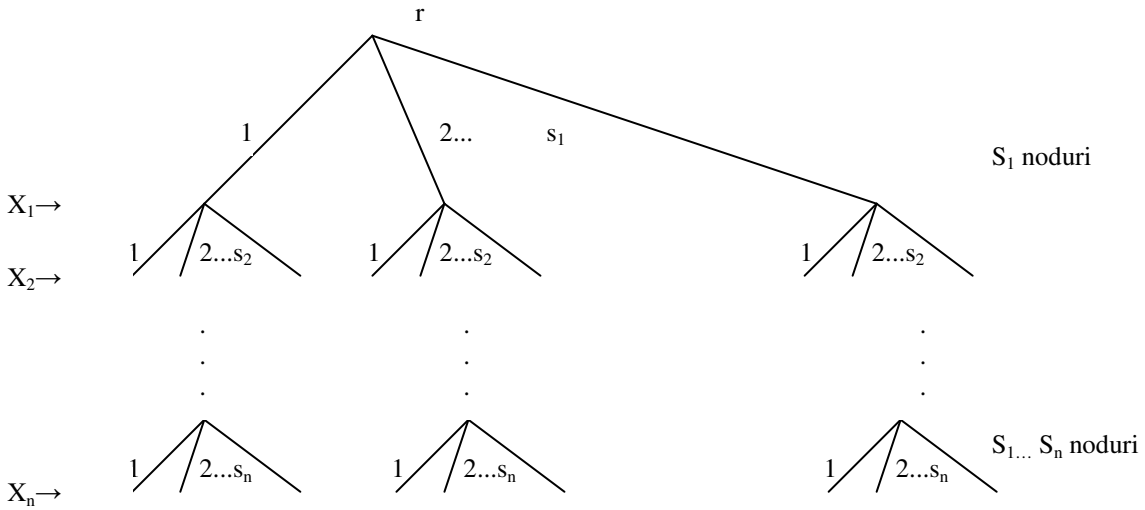
Mulţimea $S = \prod_{i=1}^{n} S_i$ se numeşte ***spaţiul soluţiilor posibile***. Soluţiile posibile care verifică şi condiţiile interne ale problemei se numesc ***soluţii rezultat***.

Metoda **Backtracking** urmăreşte fie determinarea tuturor soluţiilor rezultat, fie obţinerea unei soluţii oarecare. Dacă pentru obţinerea soluţiilor rezultat se generează toate soluţiile posibile şi pentru fiecare soluţie se verifică dacă sunt satisfăcute condiţiile interne timpul necesar generării lor va fi exponenţial.

Elementele vectorului x primesc valori în ordinea crescătoare a indicilor, deci lui x_k nu i se atribuie o valoare decât după ce $x_1, x_2...,x_{k-1}$ au primit valori. După ce x_k a primit o valoare se verifică dacă sunt îndeplinite condiţiile de continuare referitoare la $x_1, x_2...,x_k$. Neîndeplinirea condiţiilor de continuare conduce la neobţinerea unei soluţii şi astfel se face o altă alegere pentru x_k dacă este posibil (S_k nu s-a terminat) sau se micşorează k şi se face o altă alegere pentru x_k. Această micşorare a lui k dă numele metodei, deoarece atunci când nu putem avansa urmărim înapoi secvenţa curentă din soluţie.

Putem spune că metoda foloseşte un arbore virtual construit astfel:

- nivelul 0 conţine rădăcina virtuală r;
- nivelul 1 conţine ca noduri elementele mulţimii S_1, în care componenta x_1 ia valori legăturile mulţimii etichetate cu $1,2, ..., s_1$;
- nivelul $k \geq 2$, conţine ca noduri elementele mulţimii S_k , astfel ca din orice nod de pe nivelul $k-1$ pleacă exact s_k muchii; nivelul conţine $s_1... s_k$ noduri;
- nivelul n conţine $s_1... s_n$ noduri terminale;



Arborele ataşat spaţiului soluţiilor posibile

Metoda Backtracking – varianta iterativă

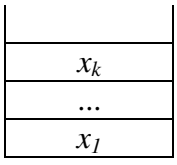
Metoda Backtracking în varianta iterativă, constă în construirea soluției x completând succesiv componentele x_k , $k = \overline{1,n}$.

Astfel:

- soluțiile sunt construite succesiv, la fiecare etapă fiind completată câte o componentă (similar ca la metoda Greedy, însă ulterior se poate reveni asupra alegerii unei componenete);
- alegerea unei valori pentru o componentă se face într-o anumită ordine (se presupune că pe mulțimile S_k există o relație de ordine şi se asigură o parcurgere sistematică a spațiului $S_1 \times S_2 \times ... \times S_n$);
- la completarea componentei x_k se verifică dacă soluția parțială (x_1, x_2,..., x_k) verifică condițiile de continuare impuse de problemă;
- dacă au fost încercate toate valorile pentru componenta x_k şi nu a fost găsită o soluție sau se doreşte determinarea unei noi soluții se revine la componenta anterioară *k-1* şi se încearcă următoarea valoare corespunzătoare acesteia;
- procesul de căutare şi revenire este continuat fie până când este găsită o soluție sau până au fost determinate toate soluțiile posibile;

Pentru implementarea metodei, la generarea soluției se poate folosi o stivă (LIFO), unde pentru a completa fiecare nivel $k = \overline{1,n}$ cu o valoare x_k putem avea următoarele cazuri:

1. există valori neanalizate în S_k şi valoare x_k din S_k satisface φ_k, caz în care k se măreşte;
2. există valori neanalizate în S_k şi valoarea x_k nu satisface φ_k, caz în care se face altă alegere pentru x_k;
3. nu mai există valori neanalizate în S_k caz în care k descreşte;
4. s-a ajuns la soluție;



stiva

Dacă $S_k = \{ \alpha_k^1, \alpha_k^2, ..., \alpha_k^{s_k} \}$, $|S_k| = s_k$, $k = \overline{1,n}$.

Presupunem $\alpha_k^0 \notin S_k$ *şi* $succesor(\alpha_k^0) = \alpha_k^{\wedge}$

Algoritmul Backtracking- varianta iterativă, în cazul general este următorul:

$k \leftarrow 1$
$x_k \leftarrow \alpha_k^0$ *{valoarea de inițializare}*

```
cât timp k>0 execută
    dacă k=n+1 atunci                {s-a găsit o soluţie}
        scrie (x1, x2,..., xn)
        k←k-1
    altfel  {altă soluţie}
        dacă xk < α_k^{s_k} atunci     {caut altă valoare pentru xk }
            xk ← succesor(xk )
            dacă φk(x1, x2,..., xk) atunci k←k+1
        altfel {s-a epuizat Sk }
            xk ← α_k^0       {valoarea de iniţializare}
            k←k-1
```

În cazul particular în care mulţimile:

$$S_1 = S_2 = ... = S_n = \{1, 2, ..., s\},$$

algoritmul devine:

$k \leftarrow 1$

$x_k \leftarrow 0 \quad , k = \overline{1,n}$ {valoarea de iniţializare}

```
cât timp k>0 execută
    dacă k=n+1 atunci            {s-a găsit o soluţie}
        scrie (x1, x2,..., xn)
        k←k-1
    altfel  {altă soluţie}
        dacă xk <s atunci  {caut altă valoare pentru xk }
            xk ← xk +1
            dacă φk(x1, x2,..., xk) atunci k←k+1
        altfel {s-a epuiza          Sk }
            xk ← 0    {valoarea de iniţializare}
            k←k-1
```

Pentru implementarea metodei la clasă, pentru uşurarea înțelegerii metodei vom prezenta o rutină standardizată (aplicabilă oricărei probleme), care foloseşte structura de stivă. Rutina va aplela subprograme care au întotdeuna acelaşi nume, şi care, din punct de vedere al metodei , realizează acelaşi lucru.

Elevului îi revine sarcina de a scrie explicit, pentru fiecare problemă în parte, subprogramele respective:

- Inițializarea unui nivel (oarecare) se face cu procedura de inițializare ***Init (st, k)***
- Găsirea următorului element al mulțimii S_k (element care a fost netestat) se face cu ajutorul subprogramului ***Succesor(st, k, as)***. Parametrul *as* (am succesor) este o variabilă booleană. În situația în care am găsit elementul, acesta este pus în stivă şi *as* ia valoarea *true*, contrar (nu a rămas nici un element netestat) *as* ia valoarea *false*.
- După alegerea unui element x_k, trebuie văzut dacă acesta îndeplineşte condițiile de continuare (altfel spus, dacă elementul este valid). Acest test se face cu ajutorul subprogramului ***Valid(st, k, ev).***
- Testul dacă s-a ajuns sau nu la soluția finală se face cu ajutorul funcției ***Soluție(k).***
- O soluție se tipăreşte cu ajutorul procedurii ***Tipar()***.

- Rutina ***Backtracking***:

```
k←1
Init(st,k)
cât timp k>0 do
        repetă
            succesor (st,k,as)
            dacă as atunci valid(st,k,ev)
        până când (not as) sau (as and ev)

        dacă as atunci
            dacă solutie(k) atunci tipar
                altfel
                    k←k+1
                    init (st,k)

        altfel k←k-1
```