Tehnica backtracking

Backtracking s-ar putea traduce "drum înapoi", "cale întoarsă" sau "revenire" ceea ce sugerează faptul că orice vector soluție este construit progresiv, începând cu prima componentă şi mergând spre ultima, cu eventuale reveniri asupra valorilor atribuite anterior (revenire care presupune un pas sau chiar mai mulți paşi înapoi).

Pentru rezolvarea problemelor utilizând această metodă vom folosi noțiunea de stivă.
Stiva este acea formă de organizare a datelor (structură de date) cu proprietatea că operațiile de introducere şi extragere a datelor se fac la un singur capăt, respectiv în vârful ei.

Această metodă se foloseşte în rezolvarea problemelor care îndeplinesc simultan următoarele condiții:

- soluția poate fi pusă sub forma unui vector; $X=x_1\ x_2\ \ldots x_n$, cu $x_1 \in A_1, x_2 \in A_2, \ldots, x_n \in A_n$;
- Mulțimile $A_1, A_2, \ldots, A_n$ sunt mulțimi finite, elementele lor fiind într-o relație de ordine bine stabilită, de obicei reprezentând termenii unei progresii aritmetice; mulțimile A_i pot fi distincte sau nu;
- Cer găsirea unei soluții $x_1, x_2, \ldots, x_n$ care optimizează o funcție criteriu $P(x_1,x_2,\ldots,x_n)$;
- Nu se dispune de o altă metodă mai rapidă de rezolvare. Acestă ultimă condiție este necesar a fi respectată, cunoscut fiind timpul mare de execuție al algoritmilor ce se bazează pe această metodă.

Metoda poate fi implementată **iterativ şi recursiv**. În continuare prezentăm *varianta iterativă*.

Tehnica Backtracking are la bază următorul principiu:

- Se construieşte soluția pas cu pas: $x_1, x_2, \ldots, x_n$;
- Dacă se constată că, pentru o valoare aleasă, nu avem cum să ajungem la soluție, se renunță la acea valoare şi se reia căutarea din punctul în care am rămas;
- Pentru fiecare problemă dată se pun în evidență nişte *condiții interne* pe care trebuie să le îndeplinească componentele $x_1, x_2, \ldots, x_n$;

  n-uplele $x_1, x_2, \ldots, x_n$ care satisfac condițiile interne se numesc *soluții posibile*.

- Dacă s-au determinat componentele $x_1, x_2, \ldots, x_{k-1}$ ale unei soluții parțiale, un element $x_k \in A_k$ nu este ataşat imediat soluției, ci se verifică mai întâi dacă el satisface anumite *condiții de continuitate*. Dacă ele nu sunt satisfăcute, atunci x_k nu este adăugat soluției, încercând alt element $x_k \in A_k$, dacă există. Dacă A_k s-a epuizat, atunci se revine la elementul x_{k-1} pentru care se caută altă valoare din A_{k-1}, dacă există;

- Soluțiile posibile care satisfac condițiile de continuare se numesc *soluții rezultat*;

- Construirea unei soluții se face pe mai mulți paşi, rezultând că elementele vectorului X primesc valori pe rând: lui x_k îi atribuim o valoare din A_k numai după ce $x_1,\ldots,x_{k-1}$ au primit valori. După ce x_k a primit o valoare, se verifică condițiile de continuitate referitoare la $x_1,\ldots,x_k$ şi numai dacă acestea au dat răspuns afirmativ se trece la atribuirea unei valori pentru x_{k+1} din A_{k+1}.

- Neîndeplinirea condițiilor de continuitate exprimă faptul că oricum s-ar alege $x_{k+1},\ldots,x_n$ nu se va ajunge la o soluție rezultat. Condițiile de continuitate sunt absolut necesare. În cazul neîndeplinirii condițiilor de continuitatre pentru x_k, se face o nouă alegere pentru x_k din A_k (dacă mai există elemente în A_k). Dacă aceasta a fost epuizată, se micşorează k cu o unitate (se coboară în sivă) deci urmează să alegem o nouă valoare pentru x_{k-1} din spațiul A_{k-1}. Micşorarea lui k dă numele metodei şi semnifică faptul că atunci când nu putem avansa, vom urmări înapoi secvența curentă din soluție;

Interesează să se determine soluțiile rezultat fie pentru a le lista, fie pentru a le alege pe acelea care optimizează funcția criteriu P dată.

Metoda backtracking constă în următoarele:

1. se alege primul element $x_1 \in A_1$;
2. presupunând generate elementele $x_1,\ldots,x_k$ ale mulțimilor A_1, $A_2,\ldots$, A_k , se alege (dacă există) x_{k+1} , primul element disponibil din mulțimea A_{k+1} , apărând 2 situații:

   a. nu s-a găsit un astfel de element, caz în care se reia căutarea considerând generate elementele $x_1,\ldots,x_{k-1}$, luând următorul element al mulțimii A_k , rămas netestat

   b. a fost găsit, caz în care se testează dacă acesta îndeplineşte condițiile de continiutate, apărând 2 posibități:

   b1. le îndeplineşte, caz în care se testează dacă s-a ajuns la soluție şi apar din nou 2 situații:

   b11. s-a ajuns la soluție, se tipăreşte soluția şi se reia algoritmul considerând generate elementele $x_1,\ldots,x_k$ (se caută în continuare un element al mulțimii A_{k+1} rămas netestat)

   b12. nu s-a ajuns la soluție, caz în care se reia algortimul considerând generate elementele $x_1,\ldots,x_{k+1}$ şi se caută un element $x_{k+2} \in A_{k+2}$;

   b2. nu le îndeplineşte, caz în care se reia algoritmul considerând generate elementele $x_1,\ldots,x_k$, iar elementul x_{k+1} se caută între elementele mulțimii A_{k+1} rămase netestate.
3. algoritmul se încheie când au fost luate în considerare toate elementele mulțimii A_1.

Pentru rezolvarea problemelor cu metoda backtracking se utilizează o stivă în care la fiecare nivel k se află un element x_k din mulțimea A_k şi următoarele funcții:

- funcția *void Init();* - la urcarea în stivă, pe nivelul la care s-a ajuns pe pune o valoare care nu se află în mulțimea considerată, dar de la care, la pasul următor, se ajunge la primul element din acea mulțime

- funcția *int Am_Succesot();* - pe un anumit nivel găsirea elementului următor celui considerat, element netestat; funcția returnează 1, dacă există succesor acesta fiind pus în stivă şi 0 în caz contrar
- funcția *int Valid();* - verifică dacă elementul ales îndeplineşte sau nu condițiile de continuitate ale problemei, ea returnează 1 dacă le îndeplineşte şi 0 în caz contrar.
- funcția *int Solutie();* - testează dacă s-a ajuns sau nu la soluția finală, returnând 1, respectiv 0.
- funcția *void Tipar();* - tipăreşte soluția.

Cu aceste notații rutina backtracking este următoarea:

```
void back()
 {
 int As;
 k=1;
 Init();
 while (k>0)
        {
        do  { } while ((As=Am_Succesor()) && !Valid());
        if (As)
                if (Solutie()) Tipar();
                   else {
                        k++;
                        Init();
                        }
        else k--;
        }
}
```