Metoda Reluarii (Backtrack)

Backtracking este numele unui algoritm general de descoperire a tuturor soluțiilor unei probleme de calcul, algoritm ce se bazează pe construirea incrementală de *soluții-candidat*, abandonând fiecare candidat parțial imediat ce devine clar că acesta nu are șanse să devină o soluție validă.

Avantajele acestei metode sunt ca putem la sigur, gasi toate solutiile unei problem. De asemenea, memoria ocupata de vectorul solutiilor nu este mare pe parcursul rularii programului, deoarece, odata ce este evident ca un candidat nu poate nici de cum sa devina o solutie, el este eliminat.

Timpul de procesare nu este mare si pseudocodul nu este complicat de inteles.

Dezavantajele acestei metode sunt ca ocupa multa memorie, intrucat este necesara memorarea fiecarei modificari alcatuita de program, la care ulterior este posibila revenirea.

Concluzia este ca aceasta metoda este foarte utila daca nu avem un set predefinit the solutii, si este necesara procesarea fiecarui candidat pentru al valida.

In secolul 21, cumca avem foarte multa memorie RAM si Cache, metoda nu are chiar asa de mari dezavantaje, necesarul de memorie foare greu poate depasi cantitatea maxima de memorie atribuita unui program. Pentru competitii insa, unde necesarul de memorie este pus la minim, metoda backtracking-ului nu prea se recomanda.

Aplicatiile. Acest algoritm este folosit de foarte multe ori pentru generarea, in timp real sau din prealabil, a unor structuri de date, care satisfac unele conditii. Cele mai simple exemple sunt generarea unui labirint sau a unui fisier foto ce defineste o textura. In game development, acest proces se numeste **procedural generation**.

Problema 1: De la tastatura se introduce un numar **n** si un arbore binary de integer. Cu ajutorul metodei reluarii, sa se gaseasca toate numerele **n** din arbore, si sa se afiseze calea, de la nodul radacina, pana la nodul cu solutia.

Programul este prezentat pe urmatoarea pagina

```
type nod = record
            st:^nod;
            dr:^nod;
            info:integer;
            passed:= false;
            registered:= false;
           end;

var s:^nod;
    origin:^nod;
    path:array of ^nod;
    solutii: array of array of ^nod;
    findValue,i,h:integer;

procedure InsertNod(e:^nod;nivel:integer);
var ans1:char;
    ans2:char;
begin
    write('valoarea in nod (nivel '+nivel+'): ');
    readln(e^.info);
    inc(nivel);

    write('are stang (A/F): ');
    readln(ans1);
    ans1:= UpperCase(ans1);

    write('are drept (A/F): ');
    readln(ans2);
    ans2:= UpperCase(ans2);
    if (ans1.equals('A')) then
    begin
        new(e^.st);
        InsertNod(e^.st,nivel);
    end;
    if (ans2.equals('A')) then
    begin
        new(e^.dr);
        InsertNod(e^.dr,nivel);
    end;
end;


procedure InsertNod(e:^nod);
begin
    InsertNod(e,0);
end;

function HasPath():boolean;
var ret:boolean;
begin
    ret:= false;
    if path[path.length-1]^.st <> nil then
        if path[path.length-1]^.st^.passed = false then
            ret:= true;
    if path[path.length-1]^.dr <> nil then
        if path[path.length-1]^.dr^.passed = false then
            ret:= true;

    hasPath:= ret;
end;

procedure RegisterSolution();
begin
    setlength(solutii,solutii.length+1);
    solutii[solutii.length-1]:= path;
end;

procedure Back();
var ar:array of ^nod;
    i:integer;
begin
    setlength(ar,path.Length-1);
    for i:= 0 to (path.Length-2) do
        ar[i]:= path[i];
    path[path.Length-1]^.passed:= true;
    path:= ar;
end;
```

```
procedure Next(n:^nod);
begin
    setlength(path,path.length+1);
    path[path.length-1]:= n;
end;

begin
    write('Introduceti valoarea pe care doriti sa o cautati: ');
    readln(findValue);
    new(s);
    InsertNod(s);
    setlength(solutii,0);
    setlength(path,1);
    path[0]:= s;

    while path.length > 0 do
    begin
        origin:= path[path.length-1];
        if ((origin^.info = findValue) and (origin^.registered = false)) then
        begin
            RegisterSolution();
            origin^.registered:= true;
        end;

        if HasPath() then
        begin
            if (origin^.st <> nil) and (origin^.st^.passed = false) then
            begin
                Next(origin^.st);
            end
            else if (origin^.dr <> nil) and (origin^.dr^.passed = false) then
            begin
                Next(origin^.dr);
            end;
        end
        else
        begin
            Back();
        end;
    end;

    writeln;
    if solutii.length > 0 then
    begin
        writeln('Solutiile: ');
        for i:= 0 to (solutii.length-1) do
        begin
            for h:= 0 to (solutii[i].Length-1) do
            begin
                write(solutii[i][h]^.info,' ');
            end;
            writeln;
        end;
    end
    else
    begin
        writeln('Nu exista solutii!');
    end;
end.
```

Initial introducem de la tastatura numarul **n** pe care dorim sa il gasim, acesta este salvat in variabila **findValue**.

Incepand cu procedure **InsertNod()**, introducem noduri prin intermediul variabilelor dinamice, si totodata introducem valorile de tip integer in acestea., dupa care ne intreaba daca acest nod are drept sau stand. Introducand **A/F** suntem deacord sau nu.

Dupa ce am creeat arborele binar, avem nevoie de o variabila care v-a stoca toate miscarile pe care le facem, aceasta este **path** si nu este altceva decat un array de noduri, fiecare nod reprezentand pasul la care am ajuns.

Totodata, avem nevoie de o variabila in care o sa inregistram solutiile pe care le-am gasit, aceasta este numita **solutii** si nu este altceva decat un array bidimensional de noduri.

Trebuie sa rectific ca fiecare nod mai are inca 2 variabile, acestea fiind **registered** si **passed**.

Registered inseamna ca daca valoarea pe care am gasito este deja inregistrata in array-ul de solutii.

Passed inseamna ca nodul deja a fost parcurs de program, si in cazul in care facem backtrack trebuie sa il ignoram, si sa trecem prin nodurile copil care nu au variabila **passed** setata cu true.

Initial, array-ul **path** are pe pozitia 0, pus nodul de start, din considerentul ca noi initial incepem de la acesta.

In **while loop** nu e nimic altceva decat parcurgereal arborelui binar, initial preluam nodul final din array-ul **path**, dupa care vedem daca el este solutie. Daca el este solutie, il introducem cu ajutorul procedurii **RegisterSolution()** in array-ul bidimensional **solutii**. Dupa, vedem daca are stand sau drept. In cazul in care exista macar unul (stand sau drept) **si** el nu este parcurs, atunci il setam ca origine prin indexarea acestuia la capatul array-ului **path** cu ajutorul procedurii **Next()** , in caz contrar noi eliminam nodul final din **path** cu ajutorul procedurii **Back()**. Procedura **Back()** si este elementul care facem backtrack.

Dupa ce am preluat toate solutiile, cu ajutorul unui **if statement** verificam daca array-ul bidimensional are o lungime mai mare decat 0. Daca lungimea e mai mare ca 0, cu ajutorul unui **for loop** dublu, afisam toate solutiile cu ajutorul procedurii **write()** si **writeln()**. In caz contrar, daca lungimea e egal cu 0, afisam tot cu ajutorul procedurii **writeln()** ca acest arbore binar nu are solutii