1. **METODA BACKTRACKING**
	1. **SCOPUL LUCRĂRII**

Lucrarea urmăreşte familiarizarea studentului cu aplicarea unei metode puternice de programare – metoda backtracking – prin prezentarea unei variante de implementare nerecursive, caracterizate prin gradul mare de generalitate, în care atenţia studentului poate să se axeze pe ceea ce diferenţiază două probleme de backtracking (impunerea restricţiilor specifice)

* 1. **BREVIAR TEORETIC**

 Metoda Backtraking se aplică problemelor în care soluția poate fi reprezentată sub forma unui vector

**x** = (x1, x2, x3, …xst,… xn) $\in $ S,

unde S este mulțimea soluțiilor problemei și S = S1 x S2 x… x Sn, și Si sunt mulțimi finite, având câte mi elemente.

 Pentru fiecare problemă, se dau relații între componentele vectorului x, care sunt numite ***condiții interne***; soluțiile posibile (care satisfac condițiile interne) se numesc ***soluții rezultat***. Metoda de generare a tuturor soluțiilor posibile și apoi de determinare a soluțiilor rezultat, prin verificarea îndeplinirii condițiilor interne, necesită foarte mult timp.

 Metoda Backtraking evită această generare și este mai eficientă. Elementele vectorului **x**, primesc pe rând valori în ordinea crescătoare a indicilor; x[st] va primi o valoare numai dacă au fost atribuite valori elementelor x1.. x[st-1]. La atribuirea valorii lui x[st] se verifică îndeplinirea unor condiții de continuare referitoare la x[1] … x[st-1]. Dacă aceste condiții nu sunt îndeplinite, la pasul ***st***, acest lucru înseamnă că orice valori i-am atribui lui x[st+1], x[st+1], .. x[n] nu se va ajunge la o soluție rezultat.

Metoda Backtraking construieşte un vector soluţie în mod progresiv, începând cu prima componentă a vectorului, şi mergând spre ultima, cu eventuale reveniri asupra atribuirilor anterioare.

 Metoda se aplică astfel :

1. se alege prima valoare din S1 şi i se atribuie lui x[1] ;
2. se presupun generate elementele x[1]…x[st-1], cu valori din S1..Sst-1; pentru generarea lui x[st] se alege primul element din Sst disponibil şi, pentru valoarea aleasă, se testează îndeplinirea condiţiilor de continuare.

Pot apărea urmatoarele situaţii :

a) x[st] îndeplineşte condiţiile de continuare. Dacă s-a ajuns la soluţia finală (st = n), atunci se afişează soluţia obţinută. Dacă nu s-a ajuns la soluţia finală, se trece la generarea elementului următor, x [st+1];

b) x[st] nu îndeplineşte condiţiile de continuare. Se încearcă următoarea valoare disponibila din Sst. Dacă nu se găseşte nici o valoare în Sst care să îndeplinească condiţiile de continuare, se revine la elementul x[st-1] şi se reia algoritmul pentru o nouă valoare a acestuia. Algoritmul se încheie când au fost luate în considerare toate elementele lui S1.

Metoda Backtraking este o tehnică de programare aplicabilă algoritmilor care oferă mai multe soluții și are ca rezultat obținerea tuturor soluțiilor problemei. Fiecare soluție se memorează într-o structură de date de tip stivă implementată cu ajutorul unui vector. Deci fiecare soluție poate fi pusă sub forma unui vector.

Într-un algoritm Backtraking ne interesează toate soluțiile posibile. Pentru a obține fiecare soluție finală se completează stiva nivel cu nivel, trecând astfel prin niște ***soluții parțiale***. Pentru a fi luate în considerare, atât soluțiile finale, cât și cele parțiale, trebuie să îndeplinească anumite condiții numite condiții de validare. O soluție care îndeplinește o astfel de condiție se numește ***soluție validă***.

Toate configurațiile stivei ce reprezintă soluții finale sunt alcătuite din elementele aceleiași mulțimi bine definite pe care o numim ***mulțimea soluțiilor***. Fiecare nouă soluție parțială se obține prin completarea soluției parțiale precedente cu încă o nivel pe stivă. La fiecare nivel se pun valori din mulțimea soluțiilor care nu au fost încercate, până când se obține o soluție validă. În acest moment, se trece la nivelul următor în stivă, pentru a completa mai departe soluția, reluând încercările pe noul nivel.

La un moment dat, pe un anumit nivel nu mai există nici o valoare neîncercată din mulțimea valorilor problemei. În acest caz, se face un pas înapoi în stivă, la nivelul anterior, și se reia căutarea cu valorile rămase neîncercate pe acest nivel anterior.

Respectivul nivel a mai fost vizitat, dar l-am abandonat după ce am pus o valoare care a generat o soluție validă. Deci, este posibil să fi rămas aici valori neîncercate. Dacă nici pe acest nivel nu mai avem valori neîncercate, mai facem un pas înapoi în stivă. Mecanismul revenirilor a determinat denumirea de metodă ***Backtraking***.

 Plecând de la nivelul 1 și repetând algoritmul până când pe toate nivelele au fost încercate toate valorile din mulțimea valorilor, se obțin soluții finale care se tipăresc.

 Vom implementa metoda Backtraking iterativ, folosind o funcţie unică, aplicabilă oricărei probleme, care determină dacă soluţia este acceptabilă din punctul de vedere al problemei concrete – am numit-o funcţia ***bun***.

 Sarcina rezolvitorului este să scrie explicit - pentru fiecare problemă - această funcție.

 O posibilă implementare este următoarea:

*#include <conio.h>*

*#include <iostream.h>*

*int n,x[10],m[10],i,j,p;*

*unsigned long int nrs=0;*

*/\* n - dimensiunea problemei*

 *in cazul general este numarul de multimi din produsul cartezian*

 *numarul de "straturi"*

 *numarul de elemente dintr-un vector solutie*

 *in cazul particular - dimensiunea tablei de sah*

 *x[i] - unul dintre posibilii vectori solutie*

 *are lungimea n*

 *m[i] - cardinalul multimii Ai*

 *cate elemente are multimea Ai*

 *pana la cat pot merge pe stratul i*

 *caz particular 1: e posibil ca m[1]=m[2]=...=m[n]=m<>n*

 *caz particular 2: e posibil ca m[1]=m[2]=...=m[n]=n*

 *i, j - contoare*

 *p - fanion - echivalent cu gasit*

*\*/*

*int* ***bun****(int st) {*

*// poate exista un* ***if*** *sau mai multe: if(???) return 0;*

*// mai jos sunt doar* ***exemple***

*// if(st>1)if(x[st]==x[st-1]) return 0;*

*// int k; for(k=1;k<=st-1;k++) if(x[k]==x[st]) return 0;*

*// if(st>1)if(x[st]>x[st-1]) return 0;*

*// if(st>1)if(x[st]>=x[st-1]) return 0;*

*// if(x[st]%2==1) return 0;*

 *return 1;*

*}*

*void* ***main****() {*

 *clrscr();*

 *int st; // stratul*

 *cout<<"n=";cin>>n;*

 *for(i=1;i<=n;i++){*

 *cout<<"pana la cat merge x["<<i<<"] ? ";*

 *cin>>m[i];*

 *}*

 *st=1;x[st]=0;*

 *while(st){*

 *p=0;*

 *while(!p && x[st]<m[st]){*

 *x[st]++;*

 *if(bun(st)==1) p=1;*

 *}*

 *if(p==0)st--; //bt*

 *else if(st<n){*

 *st++;*

 *x[st]=0;*

 *}*

 *else{*

 *nrs++;*

 *for(j=1;j<=n;j++)cout<<x[j]<<" ";*

 *cout<<endl;*

 *}*

 *}*

 *cout<<endl<<"am gasit "<<nrs<<" solutii";*

 *getch();*

*}*

 Faptul că, în lipsa unor restricţii concrete (definite prin ***if*** – urile urmate de ***return 0***) funcţia ***bun*** întoarce totdeauna valoarea 1, face ca programul să genereze produsul cartezian .

 Orice problemă concretă se rezolvă, atunci, prin simpla modificare (de obicei uşoară şi intuitivă) a funcţiei ***bun***, de aşa natură încât aceasta să întoarcă valoarea 0 pentru soluţiile inacceptabile pentru cazul concret, ***fără nici o modificare a programului principal***.

 Există cazuri în care toate mulţimile Si au acelaşi număr de elemente

m1 = m2 = … = mn = m,

caz în care funcţia main() se modifică uşor, pentru a evita citirea repetată a aceleiaşi valori

*void* ***main****() {*

 *clrscr();*

 *int st;*

 *cout<<"n=";cin>>n;*

 *cout<<”numarul de elemente: ”; cin>>m;*

 *st=1;x[st]=0;*

 *while(st){*

 *p=0;*

 *while(!p && x[st]<m){*

 *x[st]++;*

 *if(bun(st)==1) p=1;*

 *}*

 *if(p==0)st--; //bt*

 *else if(st<n){*

 *st++;*

 *x[st]=0;*

 *}*

 *else{*

 *nrs++;*

 *for(j=1;j<=n;j++)cout<<x[j]<<" ";*

 *cout<<endl;*

 *}*

 *}*

 *cout<<endl<<"am gasit "<<nrs<<" solutii";*

 *getch();*

*}*

Mai mult, xistă cazuri în care toate mulţimile Si au acelaşi număr de elemente, şi anume n (egal cu dimensiunea problemei)

m1 = m2 = … = mn = n,

caz în care funcţia main() se poate simplifica şi mai mult

*void* ***main****() {*

 *clrscr();*

 *int st;*

 *cout<<"n=";cin>>n;*

 *st=1;x[st]=0;*

 *while(st){*

 *p=0;*

 *while(!p && x[st]<n){*

 *x[st]++;*

 *if(bun(st)==1) p=1;*

 *}*

 *if(p==0)st--; //bt*

 *else if(st<n){*

 *st++;*

 *x[st]=0;*

 *}*

 *else{*

 *nrs++;*

 *for(j=1;j<=n;j++)cout<<x[j]<<" ";*

 *cout<<endl;*

 *}*

 *}*

 *cout<<endl<<"am gasit "<<nrs<<" solutii";*

 *getch();*

*}*

După cum am mai menţionat, metoda backtracking furnizează ***toate*** soluţiile unei probleme.

Dacă este necesară o singură soluţie (oarecare), algoritmul poate fi oprit după prima afişare.

Pentru a determina ***soluţia optimă*** (dintr-un anumit punct de vedere, precizat de problemă), vom genera cu metoda backtracking toate soluţiile şi vom reţine dintre acestea doar soluţia cea mai bună (problemă clasică de tip maxim). Aceasta presupune să nu afişăm fiecare soluţie ci, în momentul obţinerii unei noi soluţii, o vom compara cu soluţia precedentă (din punctul de vedere cerut în problema concretă); în cazul în care soluţia curentă este mai bună,vom reţine această soluţie.

* 1. **EXEMPLE**
1. ***Generarea produsului cartezian***

Nici o modificare nu este necesară în funcţia bun; programul construieşte în mod nativ produsul cartezian.

1. ***Permutările unei mulţimi***

Suntem în situaţia particulară *m[1] = m[2] = … = m[n] = n*;

Funcţia ***bun*** se modifică după cum urmează:

*int bun(int x[], int st){*

 *int i;*

 *for(i=1; i<=st-1; i++) if(x[i]==x[st]) return 0;*

 *return 1;*

*}*

1. ***Aranjamente de n luate câte k***

În problema noastră, n joacă rolul lui k, iar *m[1] = m[2] = … = m[n]* joacă rolul lui n

Funcţia bun este

*int bun(int x[], int st){*

 *int i;*

 *for(i=1; i<=st-1; i++) if(x[i]==x[st]) return 0;*

 *return 1;*

*}*

1. ***Combinări de n luate câte k***

În problema noastră, n joacă rolul lui k, iar *m[1] = m[2] = … = m[n]* joacă rolul lui n

Funcţia bun este

*int bun(int x[], int st){*

 *int i;*

 *for(i=1; i<=st-1; i++) if(x[st]<=x[i]) return 0;*

 *return 1;*

*}*

1. ***Problema turnurilor de pe o tablă de şah***

Se cere să se aşeze n turnuri pe o tablă de şah de dimensiune (*n X n)*, fără ca acestea să se atace reciproc.

Suntem în situaţia particulară *m[1] = m[2] = … = m[n] = n*;

Funcţia ***bun*** se modifică după cum urmează:

*int bun(int x[], int st){*

 *int i;*

 *for(i=1; i<=st-1; i++) if(x[i]==x[st]) return 0;*

 *return 1;*

*}*

şi putem observa uşor că avem aceeaşi funcţie bun ca în cazul generării permutărilor!

1. ***Problema comis-voiajorului***

Un  comis voiajor trebuie să viziteze un număr de ***n*** oraşe. Nu între oricare două oraşe există legături directe. Iniţial, comis voiajorul află într-unul dintre cele n oraşe, notat cu ***1***. El doreşte să nu treacă de 2 ori prin acelaş oraş, iar la întoarcere să ajungă din nou în oraşul 1.

Cunoscînd legăturile existente între oraşe, se cer toate traseele posibile de urmat, respectând cerinţele date.

(Generalizare: să se găsească toate traseele posibile, în cazul în care oricare din oraşe poate fi ales ca punct de plecare – evident, comis-voiajorul trebuie să se întoarcă în oraşul din care a plecat!)

Exemplu de legături între oraşe:



Trebuie găsită, mai întâi, o modalitate de a comunica programului “harta legăturilor între oraşe”. Cea mai intuitivă metodă este să reţinem aceste legături într-o matrice ***ma[ ][ ]***, de dimensiuni n X n, ale cărei elemene ma[i][j] să fie unt sau zero, în funcţie de faptul dacă există sau nu legătură directă între oraşul i şi oraşul j.

$$ma\left[i\right][j]=\left\{\begin{matrix}0&daca nu exista legatura directa intre orasul i si orasul j\\1&daca exista legatura directa intre orasul i si orasul j\end{matrix}\right.$$

 După cum vom vedea în cursurile următoare, o astfel de matrice poartă numele de ***matrice de adiacenţă*** şi joacă un rol central în studiul grafurilor (căci, fără ca aici să precizăm, figura de mai sus este un ***graf*** – mai exact, un ***graf neorientat***).

 Pentru exemplul de mai sus, matricea de adiacenţă (reţinută, pentru uşurinţă într-un fişier pe disc) este



(pe prima linie este trecută “dimensiunea” matricei, în cazul de faţă 6).

Rezolvarea în cazul (uzual) în care oraşul de plecare este precizat, ar putea fi

*#include <conio.h>*

*#include <iostream.h>*

*#include <stdio.h>*

*#include <fstream.h>*

*int x[10], drum[10][10],n;*

*int bun(int st){*

 *int i;*

 *if((st>1)&&(!drum[x[st-1]][x[st]]))return 0;*

 *for(i=1;i<st;i++)if(x[i]==x[st])return 0;*

 *if((st==n)&&(!drum[x[st]][1]))return 0;*

 *return 1;*

*}*

*void main(){*

 *clrscr();*

 *int i,j,st,p,s=0;*

 *char numefis[30];*

 *cout<<"numele fisierului de intrare: "; cin>>numefis;*

 *ifstream fin(numefis);*

 *fin>>n;*

 *for(i=1;i<=n;i++)*

 *for(j=1;j<=n;j++)*

 *fin>>drum[i][j];*

 *cout<<endl<<"Harta legaturilor :"<<endl<<endl;*

 *for(i=1;i<=n;i++){*

 *for(j=1;j<=n;j++)*

 *cout<<drum[i][j]<<" ";*

 *cout<<endl;*

 *}*

 *cout<<endl<<endl;*

 *x[1]=1;*

 *st=2;x[2]=0;*

 *while(st>1){*

 *p=0;*

 *while((!p)&&(x[st]<n)){*

 *x[st]++;*

 *if(bun(st))p=1;*

 *}*

 *if(!p)st--;*

 *else if(st<n){*

 *st++;*

 *x[st]=0;*

 *}*

 *else{*

 *for(j=1;j<=n;j++) cout<<x[j]<<" ";*

 *cout<<endl;*

 *s++;*

 *}*

 *}*

 *cout<<endl<<"am gasit "<<s<<" solutii";*

 *getch();*

*}*

În urma rulării acestui program, obţinem, pentru fişierul de intrare de mai sus:



Rezolvarea, în cazul în care se poate lua pentru plecare oricare dintre cele n oraşe, ar putea fi

*#include <conio.h>*

*#include <iostream.h>*

*#include <stdio.h>*

*#include <fstream.h>*

*int x[10], drum[10][10],n;*

*int bun(int st){*

 *int i;*

 *if((st>1)&&(!drum[x[st-1]][x[st]]))return 0;*

 *for(i=1;i<st;i++)if(x[i]==x[st])return 0;*

 *if((st==n)&&(!drum[x[st]][x[1]]))return 0;*

 *return 1;*

*}*

*void main(){*

 *int i,j,st,p,s=0;*

 *char numefis[30];*

 *cout<<"numele fisierului de intrare: "; cin>>numefis;*

 *ifstream fin(numefis);*

 *fin>>n;*

 *for(i=1;i<=n;i++)*

 *for(j=1;j<=n;j++)*

 *fin>>drum[i][j];*

 *cout<<endl<<"Harta drumurilor :"<<endl<<endl;*

 *for(i=1;i<=n;i++){*

 *for(j=1;j<=n;j++)*

 *cout<<drum[i][j]<<" ";*

 *cout<<endl;*

 *}*

 *cout<<endl<<endl;*

 *st=1;x[1]=0;*

 *while(st){*

 *p=0;*

 *while((!p)&&(x[st]<n)){*

 *x[st]++;*

 *if(bun(st))p=1;*

 *}*

 *if(!p)st--;*

 *else if(st<n){*

 *st++;*

 *x[st]=0;*

 *}*

 *else{*

 *for(j=1;j<=n;j++) cout<<x[j]<<" ";*

 *cout<<endl;*

 *s++;*

 *}*

 *}*

 *cout<<endl<<"am gasit "<<s<<" solutii";*

 *getch();*

*}*

iar rularea sa duce, în mod evident, la găsirea mai multor soluţii



* 1. **TEME DE LABORATOR**
1. Din 10 persoane, dintre care 6 bărbaţi şi 4 femei se formează o delegaţie alcătuită din 5 persoane dintre care cel puţin doua femei. Afişaţi toate modalităţile în care se poate forma aceasta delegaţie.
2. La un concurs de dans iau parte 7 fete si 8 baieţi. La un anumit dans trebuie să se formeze 4 perechi. În câte moduri se pot forma aceste perechi? Afişaţi toate soluţiile.
3. ***Problema drapelelor***

Având la dispoziţie ***c*** culori, să se determine toate drapelele tricolore care se pot realize, astfel încât să nu avem două culori successive identice

1. ***Problema nebunilor de pe o tablă de şah***

Se cere să se aşeze ***n*** nebuni pe o tablă de şah de dimensiune (*n X n)*, fără ca aceştia să se atace reciproc.

1. ***Problema reginelor de pe o tablă de şah***

Se cere să se aşeze ***n*** regine pe o tablă de şah de dimensiune (*n X n)*, fără ca acestea să se atace reciproc (se va ţine cont că o regină se comportă, simultan, ca o tură şi ca un nebun).

1. ***Problema colorării hărţilor***

Pentru o hartă cu ***n*** ţări, se cer să se găsească toate posibilităţile de colorare a hărţii, utilizând maximum patru culori, astfel încât oricare două ţări care au o frontieră comună sa fie colorate diferit.

Harta este prezentată sub forma unui tablou bidimensional, care codifică dacă o ţară I se învecinează sau nu cu o ţară j prin valorile 1 sau 0 (matricea de adiacenţă):



1. ***Problema rucsacului***

Într-un rucsac se poate transporta o anumită greutate maximă G.

O persoană dispune de n obiecte. Pentru fiecare obiect se cunoaşre greutatea şi câştigul pe care persoana îl poate obţine transportând acelst obiect.

Ce obiecte trebuie să transporte persoana respectivă pentru a obţine un câştig maxim.

Datele de intrare se citesc din fişierul RUCSAC.IN astfel:

* + linia 1: n G -unde n este numărul de obiecte şi G greutatea maximă admisă de rucsac
	+ linia i: nume[i] g[i] c[i]

unde: -nume – este numele obiectului

-g – este greutatea obiectului

- c –este caştigul obţinut pentru acel obiect cu i=1,n

1. ***Numărarea în baza B***

Fiind dat un număr natural ***B*** (1 > ***B*** > 10), să se numere în baza ***B***, pe ***n*** cifre.

Exemplu: pentru B = 3, n = 2 vom număra

00

01

02

10

11

12

20

21

22

**BONUS:**

Fiind dat un număr natural ***B*** (1 > ***B*** > 20), să se numere în baza **B**, pe ***n*** cifre (în exemplul următor, B = 16, n = 2).

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

10 11 12 … 19 1A 1B 1C 1D 1E 1F

20 21 22 … 2F

30 … 3F

40 … 4F

…

90 91 92 … 99 9A 9B 9C 9D 9E 9F

A0 A1 A2 … A9 AA AB AC AD AE AF

B0 … BF

…

F0 … FF