**PROIECT IDIVIDUAL**

**LA INFORMATICĂ**

**TEMA: TEHNICA RELUĂRII**

**(BACKTRACKING)**

A REALIZAT: Duda Vladislav, clasa a XI-a ”C”  
A VERIFICAT: Maria Guțu  
IPLT ‘Spiru Haret’

**INFORMAȚIE**

BACKTRACKING ESTE NUMELE UNUI [ALGORITM](https://ro.wikipedia.org/wiki/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.

**AVANTAJE**

-Elimină atribuiri care nu sunt necesare prin verificarea condițiilor de continuare

-Utilizează puțin timp de utilizare

-Utilizează puțină memorie

**DEZAVANTAJE**

-Necesită condiții bune de continuare, nu există condiții universal

- Tehnica backtracking se poate aplica doar pentru probleme ce admit conceptul de *„candidat parțial de soluție”*

-Depinde de iscusința programatorului

**CURIOZITĂȚI**

BACKTRACKINGUL ESTE UTIL LA REZOLVAREA UNOR PROBLEME DE SATISFACERE A CONSTRÂNGERILOR, CUM AR FI CUVINTELE ÎNCRUCIȘATE, JOCURI DE [SUDOKU](https://ro.wikipedia.org/wiki/Sudoku) ȘI ALTE PROBLEME SIMILARE. EA STĂ LA BAZA UNEI SERII DE LIMBAJE DE PROGRAMARE LOGICĂ, CUM AR FI [ICON](https://ro.wikipedia.org/w/index.php?title=Limbajul_de_programare_Icon&action=edit&redlink=1), [PLANNER](https://ro.wikipedia.org/w/index.php?title=Limbajul_de_programare_Planner&action=edit&redlink=1) ȘI [PROLOG](https://ro.wikipedia.org/wiki/Prolog_(limbaj_de_programare)).

TERMENUL „BACKTRACK” A FOST INVENTAT DE MATEMATICIANUL AMERICAN [D. H. LEHMER](https://ro.wikipedia.org/w/index.php?title=Derrick_Henry_Lehmer&action=edit&redlink=1) ÎN ANII 1950.

**EXEMPLE DE PROGRAME**

**program** permutari;

**var** st:**array**[1..25] **of** integer;i,n,p:integer;

**procedure** init;

**var** i:integer;

**begin**

write('N='); readln(n);**for** i:=1 **to** 25 **do** st[i]:=0;**end**;

**function** valid(p:integer):boolean;

**var** i:integer;

**begin** valid:=true;**for** i:=1 **to** p-1 **do if** st[i]=st[p] **then** valid:=false;**end**;

**procedure** tipar(p:integer);

**var** i:integer; **begin for** i:=1 **to** p **do** writeln(st[i],' ');**end**;

**procedure** back(p:integer);

**begin** p:=1;

{plecam de la primul nivel }

st[p]:=0;

{initializam nivelul cu 0}

**while** p>0 **do**

{cat timp stiva nu este vida}

**begin if** st[p]<n **then**

{mai exista valori neincercate pe nivelul p}

**begin** st[p]:=st[p]+1;

{st[p]<-<o noua valoare din multimea valorilor posibile>}

**if** valid(p) **then if** p=n **then** tipar(p)

{solutia este finala}

**else begin** p:=p+1;

{trecem la nivelul urmator}

st[p]:=0;

{initializam valoarea de pe nivel cu 0}

**end**;**end else**

p:=p-1; {pas inapoi}

**end**;**end**;

**begin**

init;back(1);**end**.

**Program** permutari\_fisier;

**type** TSolutie=**array**[0..20] **of** Byte;

**var** v:TSolutie;

solutii:**array**[1..100] **of** TSolutie;

nrsol:integer;

n:Byte;

**procedure** Citire;

**var** f:text;

**begin**

assign(f,'perm.in');

reset(f);

readln(f,n);

Close(f);

**end**;

**procedure** Afisare;

**var** i,k:byte;

**begin**

**for** k:=1 **to** nrsol **do**

**begin**

**for** i:=1 **to** n **do**

**begin**

write(solutii[k,i],',');

**end**;

writeln;

**end**

**end**;

**function** LungimeData(k:byte):boolean;

**begin**

LungimeData:=k=n;

**end**;

**function** final(k:byte):boolean;

**begin**

final:=LungimeData(k);

**end**;

**function** Continuare(pozitie:integer; element:Byte):boolean;

**var** c:byte;

**begin**

Continuare:=true;

**for** c:=1 **to** pozitie-1 **do**

**if** v[c]=element

**then begin**

Continuare:=false;

**exit**;

**end**;

**end**;

**procedure** back(k:integer);

**var** element:Byte;

y0:byte;

**begin**

**if** final(k-1)

**then**

**begin**

inc(nrsol);

solutii[nrsol]:=v;

**end**

**else begin**

**for** y0:=1 **to** n **do**

**begin**

element:=y0;

**if** Continuare(k,y0)

**then begin**

v[k]:=element;

back(k+1);

**end**;**end**;

**end**

**end**;

**procedure** Initializeaza;

**begin**

nrsol:=0;

**end**;**begin**

Citire;

Initializeaza;

back(1);

Afisare;

**end**.

**program** factorial\_si\_nr\_de\_combinatii;

**var** st:**array**[1..25] **of** integer;i,n,p:integer;

**procedure** init;

**var** i:integer;

**begin**

write('N='); readln(n);**for** i:=1 **to** 25 **do** st[i]:=0;**end**;

**function** valid(p:integer):boolean;

**var** i:integer;

**begin** valid:=true;**for** i:=1 **to** p-1 **do if** st[i]=st[p] **then** valid:=false;**end**;

**procedure** tipar(p,f:integer);

**var** i:integer; **begin for** i:=1 **to** p **do** f:=f\*st[i]; writeln(f);**end**;

**procedure** back(p:integer);

**begin** p:=1;

{plecam de la primul nivel }

st[p]:=0;

{initializam nivelul cu 0}

**while** p>0 **do**

{cat timp stiva nu este vida}

**begin if** st[p]<n **then**

{mai exista valori neincercate pe nivelul p}

**begin** st[p]:=st[p]+1;

{st[p]<-<o noua valoare din multimea valorilor posibile>}

**if** valid(p) **then if** p=n **then** tipar(p,1)

{solutia este finala}

**else begin** p:=p+1;

{trecem la nivelul urmator}

st[p]:=0;

{initializam valoarea de pe nivel cu 0}

**end**;**end else**

p:=p-1; {pas inapoi}

**end**;**end**;

**begin**

init;back(1);

**end**.

**program** suma\_si\_nr\_de\_combinatii;

**var** st:**array**[1..25] **of** integer;i,n,p:integer;

**procedure** init;

**var** i:integer;

**begin**

write('N='); readln(n);**for** i:=1 **to** 25 **do** st[i]:=0;**end**;

**function** valid(p:integer):boolean;

**var** i:integer;

**begin** valid:=true;**for** i:=1 **to** p-1 **do if** st[i]=st[p] **then** valid:=false;**end**;

**procedure** tipar(p,s:integer);

**var** i:integer; **begin for** i:=1 **to** p **do** s:=s+st[i]; writeln(s);**end**;

**procedure** back(p:integer);

**begin** p:=1;

{plecam de la primul nivel }

st[p]:=0;

{initializam nivelul cu 0}

**while** p>0 **do**

{cat timp stiva nu este vida}

**begin if** st[p]<n **then**

{mai exista valori neincercate pe nivelul p}

**begin** st[p]:=st[p]+1;

{st[p]<-<o noua valoare din multimea valorilor posibile>}

**if** valid(p) **then if** p=n **then** tipar(p,0)

{solutia este finala}

**else begin** p:=p+1;

{trecem la nivelul urmator}

st[p]:=0;

{initializam valoarea de pe nivel cu 0}

**end**;**end else**

p:=p-1; {pas inapoi}

**end**;**end**;

**begin**

init;back(1);**end**.

**program** suma\_impare\_si\_nr\_de\_combinatii;

**var** st:**array**[1..25] **of** integer;i,n,p:integer;

**procedure** init;

**var** i:integer;

**begin**

write('N='); readln(n);**for** i:=1 **to** 25 **do** st[i]:=0;**end**;

**function** valid(p:integer):boolean;

**var** i:integer;

**begin** valid:=true;**for** i:=1 **to** p-1 **do if** st[i]=st[p] **then** valid:=false;**end**;

**procedure** tipar(p,s:integer);

**var** i:integer; **begin for** i:=1 **to** p **do if** st[i]**mod** 2>0 **then** s:=s+st[i]; writeln(s);**end**;

**procedure** back(p:integer);

**begin** p:=1;

{plecam de la primul nivel }

st[p]:=0;

{initializam nivelul cu 0}

**while** p>0 **do**

{cat timp stiva nu este vida}

**begin if** st[p]<n **then**

{mai exista valori neincercate pe nivelul p}

**begin** st[p]:=st[p]+1;

{st[p]<-<o noua valoare din multimea valorilor posibile>}

**if** valid(p) **then if** p=n **then** tipar(p,0)

{solutia este finala}

**else begin** p:=p+1;

{trecem la nivelul urmator}

st[p]:=0;

{initializam valoarea de pe nivel cu 0}

**end**;**end else**

p:=p-1; {pas inapoi}

**end**;**end**;

**begin**

init;back(1);**end**.

**program** suma\_pare\_si\_nr\_de\_combinatii;

**var** st:**array**[1..25] **of** integer;i,n,p:integer;

**procedure** init;

**var** i:integer;

**begin**

write('N='); readln(n);**for** i:=1 **to** 25 **do** st[i]:=0;**end**;

**function** valid(p:integer):boolean;

**var** i:integer;

**begin** valid:=true;**for** i:=1 **to** p-1 **do if** st[i]=st[p] **then** valid:=false;**end**;

**procedure** tipar(p,s:integer);

**var** i:integer; **begin for** i:=1 **to** p **do if** st[i]**mod** 2=0 **then** s:=s+st[i]; writeln(s);**end**;

**procedure** back(p:integer);

**begin** p:=1;

{plecam de la primul nivel }

st[p]:=0;

{initializam nivelul cu 0}

**while** p>0 **do**

{cat timp stiva nu este vida}

**begin if** st[p]<n **then**

{mai exista valori neincercate pe nivelul p}

**begin** st[p]:=st[p]+1;

{st[p]<-<o noua valoare din multimea valorilor posibile>}

**if** valid(p) **then if** p=n **then** tipar(p,0)

{solutia este finala}

**else begin** p:=p+1;

{trecem la nivelul urmator}

st[p]:=0;

{initializam valoarea de pe nivel cu 0}

**end**;**end else**

p:=p-1; {pas inapoi}

**end**;**end**;

**begin**

init;back(1);**end**.

**CONCLUZIE**

METODA RELUĂRII (BACKTRACKING) ESTE O TEMĂ COMPLICATĂ ȘI ÎN ACELAȘI TIMP O APTITUDINE NECESARĂ UNUI PROGRAMATOR BUN. NECESITĂ O GÂNDIRE LOGICĂ BUNĂ PENTRU ALCĂTUIREA CONDIȚIILOR DE VERIFICARE. ACEASTĂ METODĂ ESTE AVANTAJOASĂ DIN MOTIVUL ECONOMIEI DE MEMORIE UTILIZATĂ ȘI TIMPUL CU MULT MAI SCURT DE EFECTUARE A PROBLEMEI ȘI GĂSIREA SOLUȚIEI.

**BIBLIOGRAFIE**

<https://ro.wikipedia.org/wiki/Backtracking>

MANUAL CLASA XI-a EDITURA Știința

<https://www.scribd.com/document/13396582/Limbajul-Pascal-Metoda-Backtracking-Permutari>