Capitolul 9. Metoda Backtracking

În practică întâlnim foarte multe probleme în care o soluţie se reprezintă sub forma unui vector **x=(x_1, x_2, ..., x_n)** unde fiecare componentă **x_i** ia valori dintr-o mulţime **S_i** cu **i=1,2,…,n**. De regulă există mai multe astfel de soluţii. Teoretic am putea să generăm într-un mod oarecare toate elementele produsului cartezian **$S_1 x S_2 x$ ... x S_n** şi dintre acestea alegem pe cele care satisfac cerinţele (condiţiile) enunţate în problemă. Ne întrebăm însă dacă este eficientă o astfel de abordare. Un răspuns la această întrebare avem dacă vom considera cea mai simplă situaţie în care toate mulţimile **S_1, S_2, ..., S_n** au fiecare doar câte două elemente. Atunci există **2^n** elemente care se pot genera pentru produsul cartezian. Să analizăm cât ar dura să le generăm pe toate dacă avem la dispoziţie un calculator care execută un miliard (adică **10^9**) operaţii pe secundă. Presupunând că generarea unei singure posibilităţi înseamnă o singură operaţie, avem:

n	Timpul de execuţie
10	$2^n/10^9=2^{10}/10^9$=**0,000001024 secunde**
20	$2^n/10^9=2^{20}/10^9$=**0,001 secunde**
30	$2^n/10^9=2^{30}/10^9$=**1,074 secunde**
40	$2^n/10^9=2^{40}/10^9$=**1100 secunde**
50	$2^n/10^9=2^{50}/10^9$=**313 ore**
100	$2^n/10^9=2^{100}/10^9$=**40.196.936.841.331 ani**

Aţi aştepta atâta timp ca să generaţi toate elementele produsului cartezian şi apoi aţi mai avea răbdarea necesară să alegeţi dintre acestea doar pe cele care vă interesează, adică pe cele care îndeplinesc condiţiile cerute de o anumită problemă?

Această modalitate de a rezolva o problemă de acest tip mai poartă numele de *metoda forţei brute*.

Aşa cum vom vedea în continuare, metoda Backtracking[1] este cea care ne ajută ca pentru astfel de probleme să nu generăm toate elementele produsului cartezian, *construind* soluţiile problemei pas cu pas, evitând generarea a foarte multe elemente ale produsului cartezian care nu pot fi niciodată soluţii ale problemei.

9. 1. Prezentare generală

Putem să spunem că metoda Backtracking se aplică la probleme în care soluţia se poate reprezenta sub forma unui vector

$$x=(x_1,x_2,...,x_n)\in S_1\times S_2\times...\times S_n,$$

unde mulţimile **S_1, S_2, ..., S_n** sunt mulţimi finite având **s_1**, **s_2**, ..., respectiv **s_n** elemente (unde am notat cu **s_i** cardinalul mulţimii **S_i**, adică numărul de elemente pe care le conţine mulţimea **S_i**).

---

[1] Backtracking s-ar putea traduce "**urmărire înapoi**" ceea ce sugerează faptul că orice vector soluţie este construit *progresiv*, începând cu prima componentă şi mergând către ultima, cu eventuale ***reveniri*** (urmăriri) asupra valorilor atribuite anterior (cu unul sau mai mulţi paşi ***înapoi***). Termenul „backtrack" a fost inventat de matematicianul american D. H. Lehmer în anii 1950. Backtrackingul este util la rezolvarea unor probleme de satisfacere a constrângerilor, cum ar fi cuvintele încrucişate, jocuri de sudoku şi alte probleme similare. Ea stă la baza unei serii de limbaje de programare logică, cum ar fi Icon, Planner şi Prolog.

Pentru fiecare problemă concretă sunt date anumite relaţii între componentele vectorului **x**, numite *condiţii interne*.

Mulţimea finită $\mathbf{S_1xS_2x\ ...\ xS_n}$ (în matematică fiind numită produs cartezian, pe care să-l notăm cu **S**) se numeşte *spaţiul soluţiilor posibile*.

Soluţiile posibile care satisfac condiţiile interne se numesc *soluţii rezultat*.

Ceea ce ne propunem este de a determina toate soluţiile rezultat (cu scopul de a le lista sau de a alege dintre ele pe cele care maximizează sau minimizează o eventuală funcţie obiectiv dată, adică *soluţiile optime*). Sunt şi probleme în care se cere o singură soluție, iar în acest caz ne vom opri imediat ce am determinat una dintre ele.

Aşa cum am văzut, o metodă simplă de determinare a soluţiilor rezultat constă în a genera într-un mod oarecare *toate* soluţiile posibile şi, dintre acestea, de a verifica şi a le alege doar pe cele satisfac condiţiile interne.

Să luăm două exemple pentru a înţelege mai bine la ce tipuri de probleme se aplică metoda Backtracking, dar vom încerca, mai întâi, să le rezolvăm prin metoda forţei brute.

Problema 9.1. (*Mulțimi de litere*) Fie două mulţimi de litere $\mathbf{S_1=\{A,B,C\}}$ şi $\mathbf{S_2=\{M,N\}}$. Se cere să se determine acele perechi **(x1,x2)** cu $\mathbf{x_1 \in S_1}$ şi $\mathbf{x_2 \in S_2}$ cu proprietatea că dacă $\mathbf{x_1}$ este **A** sau **B**, atunci $\mathbf{x_2}$ nu poate fi **N**.

Soluţie. Pentru această problemă sunt *posibile* următoarele şase soluţii:

(A,M), (A,N), (B,M), (B,N), (C,M), (C,N).

Acestea fiind toate elementele produsului cartezian $\mathbf{S_1xS_2}$ (care am văzut că se numeşte *spaţiul soluţiilor posibile*).

Dintre acestea numai patru îndeplinesc *condiţiile* din enunţul problemei:

(A,M), (B,M), (C,M), (C,N)

acestea fiind *soluţii rezultat*.

Problema 9.2. *(Pronosport).* Fie un joc de tip pronosport la care, pentru simplitate, vom considera că sunt în discuţie doar **4** meciuri. Să presupunem că o persoană doreşte să participe la acest joc, plecând cu convingerea că în variantele câştigătoare nu pot exista mai mult de două meciuri nule (cu pronostic **X**), iar în ultimul meci gazdele nu câştigă (pronosticul nu poate fi **1**) şi vrea să găsească toate variantele care îndeplinesc aceste condiţii.

Soluţie. Prin urmare, problema cere determinarea unor vectori în care valorile componentelor sunt pronosticuri (**X**, **1** sau **2**) pentru cele **4** meciuri. Rezultă că pentru această problemă spaţiul soluţiilor posibile este produsul cartezian $\mathbf{S_1x\ S_2xS_3xS_4}$, unde $\mathbf{S_1=S_2=S_3=S_4=\{1,X,2\}}$. Pentru a fi soluţie rezultat, un asemenea vector trebuie să îndeplinească următoarele două condiţii:

- în vector nu pot să apară mai mult de două pronosticuri **X**;
- ultima componentă poate fi doar **X** sau **2**.

Desigur, există mai mulţi vectori care îndeplinesc aceste condiţii, ca de exemplu: **(X,1,2,X), (1,2,1,2)** etc.

Utilizând metoda forţei brute, se generează pe rând toate cele $\mathbf{3^4=81}$ soluţii posibile şi dintre ele se aleg cele care satisfac condiţiile din enunţ. Să observăm însă că dacă în loc de **4** meciuri vom considera **13** meciuri, aşa cum se întâmplă în realitate, vor exista $\mathbf{3^{13}=1.594.323}$ variante, adică un număr foarte mare de soluţii posibile. Mai precis,

numărul soluţiilor posibile, deci şi timpul de lucru, depind exponenţial de lungimea **n** a vectorului **x.**

Metoda Backtracking încearcă să evite generarea tuturor soluţiilor posibile.

Descrierea metodei

Pentru a evita generarea tuturor soluţiilor posibile, metoda Backtracking atribuie pe rând valori elementelor vectorului $\mathbf{x=(x_1, x_2, ..., x_n)}$ în ordinea crescătoare a indicilor, începând cu $\mathbf{x_1}$. Astfel, lui $\mathbf{x_k}$ i se atribuie o valoare numai dacă au fost atribuite deja valori lui $\mathbf{x_1, x_2, ..., x_{k-1}}$. Mai mult, odată stabilită o valoare pentru $\mathbf{x_k}$, nu se trece direct la atribuirea de valori lui $\mathbf{x_{k+1}}$, ci se verifică nişte *condiţii de continuare* referitoare la $\mathbf{x_1, x_2, ..., x_k}$. Aceste condiţii stabilesc situaţiile în care are sens să trecem la calculul lui $\mathbf{x_{k+1}}$, neîndeplinirea lor exprimând faptul că oricum am alege $\mathbf{x_{k+1},...,x_n}$ nu vom putea ajunge la o soluţie rezultat, adică o soluţie pentru care condiţiile interne să fie satisfăcute. Deci, verificarea condiţiilor de continuare este *strict necesară* pentru obţinerea unei soluţii.

De exemplu, dacă în problema 9.2 componentele $\mathbf{x_1}$ şi $\mathbf{x_2}$ au primit amândouă valoarea **X**, atunci lui $\mathbf{x_3}$ nu i se mai poate atribui valoarea **X** (pentru a nu încălca restricţia ca numărul maxim de pronosticuri **X** să fie cel mult **2,** aceasta însemnând că nu sunt îndeplinite condiţiile de continuare).

Evident că în cazul neîndeplinirii condiţiilor de continuare va trebui să facem o altă alegere pentru $\mathbf{x_k}$ sau, dacă $\mathbf{S_k}$ a fost epuizată, se merge înapoi la mulţimea $\mathbf{S_{k-1}}$ şi se încearcă să se facă o nouă alegere pentru componenta precedentă $\mathbf{x_{k-1}}$ a vectorului (ceea ce înseamnă să micşorăm pe **k** cu o unitate) după care înaintăm la $\mathbf{S_k}$ şi facem o nouă alegere pentru $\mathbf{x_k}$ (reconsiderând toate valorile din $\mathbf{S_k}$). Această revenire (prin micşorarea lui **k**) dă numele metodei, ilustrând faptul că atunci când nu putem avansa, revenim la componenta anterioară din soluţie.

Este evident că între condiţiile de continuare şi condiţiile interne există o strânsă legătură.

O bună alegere pentru condiţiile de continuare are ca efect o reducere a numărului de calcule. Faptul că valorile curente $\mathbf{v_1, v_2, ..., v_{k-1}}$ ale lui $\mathbf{x_1, x_2, ...,}$ respectiv $\mathbf{x_{k-1}}$ satisfac condiţiile de continuare nu este suficient pentru a garanta că se va obţine o soluţie ale cărei prime **k-1** componente coincid cu valorile $\mathbf{v_1, v_2, ..., v_{k-1}}$.

De exemplu, în cazul problemei 9.2, chiar dacă valorile curente pentru $\mathbf{x_1}$ şi $\mathbf{x_2}$ sunt **X**, iar pentru $\mathbf{x_3}$ este **1** (deci aceste valori îndeplinesc condiţiile de continuare), totuşi **(X, X, 1, 1)** nu este soluţie.

De obicei condiţiile de continuare reprezintă restricţia condiţiilor interne pentru primele **k** componente ale vectorului. Este evident că în cazul **k=n** condiţiile de continuare sunt chiar condiţiile interne.

Prin metoda Backtracking, orice vector soluţie este construit *progresiv*, începând cu prima componentă şi mergând către ultima, cu eventuale *reveniri* asupra valorilor atribuite anterior (cu unul sau mai mulţi paşi *înapoi*, atâta timp cât e posibilă revenirea). Prin atribuiri sau încercări de atribuire eşuate din cauza nerespectării condiţiilor de continuare, anumite valori sunt "consumate" (reamintim că $\mathbf{x_1, x_2, ..., x_n}$ primesc valori

în mulţimile finite $S_1, S_2, ..., S_n$). Vom nota cu C_k mulţimea valorilor consumate deja la momentul curent pentru componenta x_k (evident $C_k \subseteq S_k$).

Dacă ne imaginăm că vectorul **x** mai are o componentă suplimentară x_{n+1} (care nu poate lua nici o valoare, adică $S_{n+1}=\Phi$) şi încercăm să dăm valori componentei **k=n+1**, atunci putem spune că vectorul $v=(v_1, v_2, ..., v_n)$ a îndeplinit condiţiile de continuare (ce coincid cu condiţiile interne) şi prin urmare **v** este o soluţie rezulat. În practică, această condiţie este cea utilizată de obicei pentru a sesiza obţinerea unei soluţii.

O problemă importantă este aceea a determinării momentului în care se încheie procesul de găsire a tuturor soluţiilor. Având în vedere că mulţimile $S_1, S_2, ..., S_n$ sunt mulţimi finite, iar prin reveniri succesive nu se ajunge de două ori la aceeaşi soluţie parţială, rezultă că, la un moment dat, tot revenind, se va ajunge la momentul în care şi pentru x_1 se epuizează toate valorile din S_1, în această situaţie putând să ne imaginăm că vectorul **x** mai are o componentă suplimentară x_0 (care nu poate lua nici o valoare, adică $S_0=\Phi$) şi în acest caz înseamnă c-am epuizat toate situaţiile de a obţine o soluţie rezultat. În practică, această condiţie (**k=0**) este cea utilizată de obicei pentru a sesiza obţinerea tuturor soluţiilor rezultat.

Prin urmare, algoritmul general pentru metoda Backtracking poate fi sintetizat astfel:

```
Iniţializează mulţimile de valori S1, S2, ..., Sn
C1, C2, ..., Cn← Φ {Iniţial vectorul soluţie nu are valori}
k←1; {Se porneşte de la prima componentă}
cât timp k>0 execută {Nu s-au construit toate soluţiile}
  dacă k=n+1 atunci {S-a construit o soluţie}
  Reţine/afişează soluţia v=(v1, v2, ..., vn)
  k←k-1 {Revine după construirea unei soluţii}
  altfel
   dacă Ck<>Sk atunci {Mai există valori neconsumate}
   Alege o valoare vk din Sk\Ck; Ck←Ck ∪ {vk };
    dacă v=(v1,v2,...,vk) satisfac condiţiile de continuare atunci {Atribuie şi avansează}
    xk←vk; k←k+1
    altfel {Încercare eşuată}
   altfel {Revenire}
   Ck←Φ; k←k-1
```

Programarea algoritmului de mai sus se simplifică mult, fără a restrânge generalitatea sa, dacă se consideră că mulţimile de valori $S_1, S_2, ..., S_n$ sunt de forma:

$S_1=\{1,2,...,s_1\}$, $S_2=\{1,2,...,s_2\}$, ..., $S_n=\{1,2,...,s_n\}$

deci fiecare mulţime S_k este formată din primele s_k numere naturale, începând cu **1**. Prin urmare, mulţimea S_k poate fi reprezentată simplu prin numărul său de elemente s_k. Se presupune de asemenea că valorile pentru fiecare componentă x_k vor fi alese în ordine crescătoare.

În această situaţie, fiecare mulţime de valori consumate $\mathbf{C_k}$ este de forma **{1,2,...,v_k}** şi drept consecinţă poate fi reprezentată doar prin valoarea $\mathbf{v_k}$. Cum $\mathbf{v_k}$ este numărul elementelor mulţimii $\mathbf{C_k}$, această mulţime este vidă dacă şi numai dacă $\mathbf{v_k=0}$. De exemplu, în problema 9.1, vom conveni să reprezentăm pe **A, B, C** respectiv prin **1, 2, 3** iar pe **M** şi **N** prin **1** şi **2**. Prin urmare, mulţimea $\mathbf{S_1}$ va fi reprezentată prin numărul **3**, iar $\mathbf{S_2}$ prin numărul **2**. Se observă că deşi **A** şi **M** sunt codificate prin acelaşi număr, ele se disting prin poziţia pe care o ocupă în vectorul soluţie $\mathbf{x=(x_1,x_2)}$. Astfel, **x=(1,1)** reprezintă soluţia **x=(A,M)**.

În foarte multe probleme, mulţimile în care pot lua valori componentele vectorului sunt toate identice cu o mulţime finită oarecare **S** cu **s** elemente, codificată astfel:

S={1,2,...,s}.

În acest caz, algoritmul corespunzător este:

```
Întregi k,i;
 pentru i=1,n execută x[i]←0; {Soluţia iniţială}
 ■
k←1;
 cât timp k> 0 execută
  dacă k=n+1 atunci { S-a construit o soluţie}
  retsol; {Reţne/afişează soluţia}
  k←k-1; {Revenire după construirea unei soluţii}
  altfel
   dacă x[k]<s atunci {Mai există valori neconsumate pentru x[k]}
   x[k]←x[k]+1; {Se alege valoarea următoare}
    dacă continuare(k) atunci k←k+1; {Avansează}
    ■
   altfel {Revenire}
   x[k]←0; k←k-1
   ■
  ■
 ■
```

Subprogramul *Backtracking* apelează:

- subrutina *retsol* care reţine soluţia, adică valorile vectorului ***x*** (în mod concret, aceasta poate însemna listarea soluţiei, compararea ei cu soluţiile precedente etc.);

- funcţia *continuare*(*k*) verifică dacă valorile primelor **k** componente ale vectorului ***x*** satisfac condiţiile de continuare; în caz afirmativ este întoarsă valoarea *adevărat*, iar în caz contrar valoarea *fals*.

Un exemplu de aplicare a metodei

De exemplu, programul de mai jos generează soluţiile jocului de pronosport prezentat în problema 9.2. Pentru a folosi doar valori numerice elementele mulţimii **S** au fost notate cu **1,2,** şi **3** în loc de **X**, **1**, respectiv **2**. Programul este următorul:

```
#include <iostream>
using namespace std;
const int N=4; // Numarul de meciuri
const int S=3; // Nr. de valori {1,x,2}
int x[N+1], nrv=0; // x=vectorul solutie si nrv=nr. variantei
void init(int k)
{
    x[k]=0;
}
```

```
void retsol()
 {
     int i;
     nrv++; cout<<"Varianta "<<nrv<<" : ";
     for (i=1;i<=N;i++)
        switch (x[i])
        {
            case 1 : cout<<"X "; break;
            case 2 : cout<<"1 "; break;
            case 3 : cout<<"2 "; break;
        }
    cout<<"\n";
}
int continuare(int k)
{
    int nx=0,i; // nx este nr. pronosticuri X
    for (i=1;i<=k;i++)
        if (x[i]==1) nx++;
        if ((nx<=2)&&(x[N]!=2))
            return 1;
        else return 0;
}
void backtracking()
{
    int k=1;init(1);
    while (k>0)
        if (k==N+1)
            {retsol(); k--;}
        else
            if (x[k]<S)
            {
                x[k]++;
                if (continuare(k)) k++;
            }
            else {init(k); k--;};
}
int main()
{
    backtracking();
    return 0;
}
```

Caracteristici ale metodei

Metoda Backtracking este o metodă *constructivă* (în sensul că soluţia se construieşte începând cu primul element al vectorului **x** la care adăugăm pas cu pas următoarele elemente, numai că, dacă nu se mai poate înainta, se revine la pasul anterior, reconsiderându-se toate elementele). În acest fel pot fi determinate toate soluţiile rezultat. În funcţie de cerinţele problemei, ne putem mulţumi cu o singură soluţie sau le determinăm pe toate. Sunt probleme (de optimizare) în care ni se cere ca dintre toate soluţiile rezultat să alegem, pe baza unor anumite criterii, doar pe cele care le satisfac (acele soluţii care maximizează sau minimizează o anumită funcţie, numită *funcţie de optim*).

9.2. Probleme de generare. Oportunitatea utilizării metodei

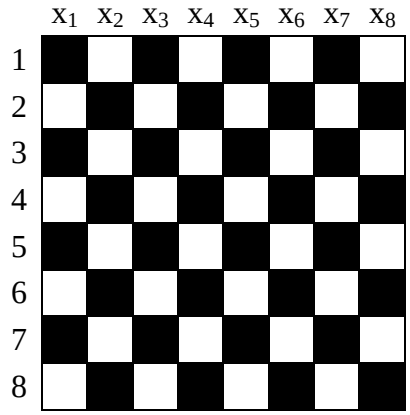
Metoda Backtracking este una destul de *costisitoare*, în sensul că ea consumă mult timp şi multă memorie, dar sunt probleme practice pentru care nu avem alte metode mai bune de rezolvare. Unele dintre ele vor fi rezolvate în acest capitol. Chiar dacă e o metodă costisitoare, pentru anumite probleme ea nu poate fi evitată, neexistând alte metode cu o eficienţă mai bună. Ea se dovedeşte utilă în probleme şi jocuri de logică (cum ar fi jocul de şah, jocul sudoku, foarte multe jocuri de tip puzzle, rebus şi altele), în unele aplicaţii din domeniul matematicii cum ar fi: generarea permutărilor, a aranjamentelor, combinări, partiţii ale unei mulţimi, etc. De asemenea, ea stă la baza unei serii de limbaje de programare logica, cum ar fi Icon, Planner şi Prolog.

Practic, metoda Backtracking este utilizează în aplicaţii din foarte multe domenii unde întâlnim probleme pentru rezolvarea cărora putem reprezenta soluţia sub forma unui vector, generând soluţiile rezultat ţinând cont de anumite constrângeri (condiţiile interne). De cele mai multe ori nu este evident faptul că soluţia se reprezintă sub forma unui vector, aşa cum vom vedea şi din exemplele care urmează.

Problema 9.3. ***(Problema celor n dame).*** Fiind dată o tablă de şah, de dimensiune **n**x**n**, se cer toate soluţiile de aranjare a **n** dame, astfel încât două dame oarecare să nu se afle pe aceeaşi linie, coloană sau diagonală (adică damele să nu se atace reciproc).

Soluţie. Evident că pe fiecare linie va fi plasată o singură damă (dacă ar fi plasate două dame pe aceeaşi linie atunci ele s-ar ataca). Prin urmare, o soluţie posibilă se poate reprezenta sub forma unui vector **x** cu **n** componente, unde pentru fiecare **k** de la **1** la **n**, $\mathbf{x_k}$ va preciza care este coloana pe care este plasată dama de pe linia **k**.

Figura următoare prezintă notaţiile pentru tabla de şah obişnuită (cazul **n=8**).



Notaţiile pentru tabla de şah

În cazul acestei probleme avem $\mathbf{s_i=n}$, numărul soluţiilor posibile fiind $\mathbf{n^n}$, ceea ce face ca timpul determinării tuturor soluţiilor posibile să fie foarte mare. Pentru a pune în evidenţă condiţiile de continuare prin aplicarea metodei Backtracking vom presupune că pe tabla de şah sunt plasate corect damele de pe primele **k-1** linii. Atunci dama de pe linia **k** poate fi plasată pe linia **k** dacă sunt îndeplinite următoarele condiţii:

1) pe coloana $\mathbf{x_k}$ să nu mai fi fost plasată nici o damă $\left(\text{adică } \mathbf{x_k \neq x_i \quad \forall i < k}\right)$.

2) pe diagonala ce trece prin căsuţa $\mathbf{(k,x_k)}$ să nu mai fi fost plasată nici o damă $\left(\text{adică } \mathbf{|x_k - x_i| \neq |k - i|, \ \forall i < k}\right)$.

Programul este următorul:

```
#include <iostream>
#include <algorithm>
using namespace std;
#define NR 20
int n, x[NR];
int continuare(int k)
{
    int i,b;
    b=1; i=1;
    while (b && (i<k))
        if ((x[k]==x[i]) || (abs(x[k]-x[i])==k-i)) b=0;
        else i++;
    return b;
}
void retsol()
{
    int i;
    for (i=1;i<=n;i++) cout<<x[i]<<" ";
    cout<<"\n";
}
void backtracking()
{
    int k=1,i;
    for (i=1;i<=n;i++) x[i]=0;
    while (k!=0)
        if (k==n+1)
        {
            retsol(); k--;
        }
        else
            if (x[k]<n)
            {
                x[k]++;
                if (continuare(k))
                    k++;
            }
            else
            {
                x[k]=0;
                k--;
            }
}
int main()
{
    cout<<"Dimensiunea tablei de sah n= ";
    cin>>n;
    backtracking();
}
```

Problema 9.4. ***(Generarea partiţiilor unui număr natural).*** Se citeşte un număr natural **n**. Se cere să se tipărească toate modurile de descompunere a sa ca sumă de numere naturale.

Exemplu. Pentru **n=4** avem: **1111, 112, 121, 13, 211, 22, 31, 4**.

Observaţie. Ordinea numerelor din sumă este importantă. Astfel, se tipăreşte **112** dar şi **211** precum şi **121**.

Soluție. Vecorul **x** conține numerele naturale în care poate fi descompus **n.** Observăm că el are lungimi diferite, în funcție de soluția determinată de algoritm. Mai observăm că lungimea maximă a unei soluții este chiar **n**, deoarece acesta poate fi descompus astfel: **1+1+…+1** (**n** valori, toate egale cu **1**), aceasta fiind prima soluție.

Mulțimile S_1, S_2 , …, S_n sunt toate egale cu mutimea **{1,2, …,n}**, prin urmare, pentru ca x_k să aibă succesor, e suficient ca $x_k<n$.

Condiția de continuare este ca suma $s=x_1+x_2+\ldots+x_k<n$. Se ajunge la o soluție atunci când **s=n**. Programul e următorul:

```
#include <iostream>
using namespace std;
const int NR=30;
int x[NR],k,n,i;
void init(int k)
{
    x[k]=0;
}
int succesor(int k)
{
    if ( x[k]<n ) return 1;
    else return 0;
}
int continuare(int k)
{
    int s=0;
    for(i=1;i<k;i++) s+=x[i];
    if(s<n) return 1;
    else return 0;
}
int solutie(int k)
{
    int s=0;
    for(i=1;i<k;i++) s+=x[i];
    if(s==n) return 1;
    else return 0;
}
void afisare()
{
    int i;
    cout<<x[1];
    for(i=2;i<k;i++) cout<<'+'<<x[i];
    cout<<'\n';
}
void back()
{
    k=1; init(1);
    while (k!=0)
        {
            if (solutie(k)) {afisare();k--;}
            else
                if (succesor(k))
                {
                    x[k]++;
                    if (continuare(k)) k++;
                }
```

```
                else {init(k); k--;}
        }
}
int main(void)
{
    cout<<"Dati n = "; cin>>n;
    back();
    return 0;
}
```

Metoda backtracking poate fi reprezentată uşor, pe un arbore construit astfel:

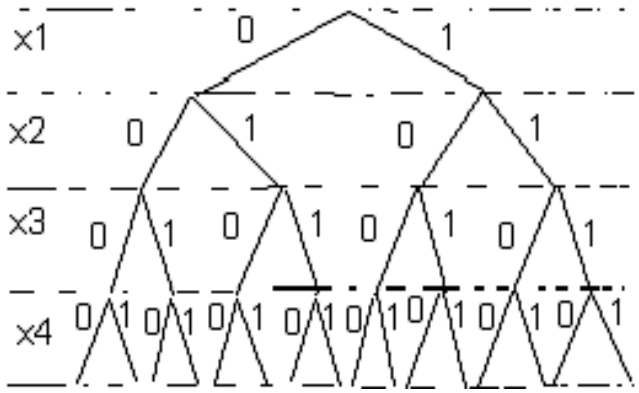
- nivelul 1 conţine rădăcina;

- din orice vârf de pe nivelul **k** pleacă s_k muchii spre nivelul k+1 etichetaţi cu cele s_k muchii ale lui S_k.

Nivelul **n+1** va conţine $s_1 \times s_2 \times ... \times s_n$ vârfuri. Pentru fiecare vârf de pe nivelul **n+1**, etichetele muchiilor conţinute pe drumul ce leagă rădăcina de acest vârf reprezintă o soluţie posibilă. Să luăm şi un exemplu.

Problema 9.5. (*Submulţimi de sumă dată*) Fie $A=(a_1, a_2, ..., a_n)$ cu $a_i>0$, $i \in \{1,2,\ldots,n\}$. Fie $M \in R+$. Se caută toate submulţimile **B** ale lui **A** pentru care suma elementelor este **M**. (Bacalaureat, sesiunea iunie-iulie 2001, variant 5, subiectul IV)

Pentru a putea rezolva problema prin metoda backtracking vom reprezenta soluţia sub forma $x=(x_1, x_2, ..., x_n)$ unde $x_i=1$ dacă $a_i \in B$ şi $x_i=0$ în caz contrar. Pentru **n=4** arborele ataşat metodei backtracking este următorul:



Dacă într-un vârf condiţiile de continuare nu mai sunt verificate, se va renunţa la parcurgerea subarborelui care are ca rădăcină acel vârf, în acest fel eliminând foarte multe dintre soluțiile posibile ce nu pot fi soluții rezultat.

Diferitele variante ale metodei backtracking nu schimbă esenţa ei care constă în faptul că poate fi reprezentată pe un arbore care este parcurs "coborând" numai dacă există şanse de a ajunge la o soluţie rezultat (această metodă de parcurgere a arborilor fiind cunoscută sub numele **DF** (**D**epth **F**irst – „în adâncime”) şi a fost propusă de Trémaux în secolul al XIX-lea ca tehnică de rezolvare a problemelor legate de parcurgerea labirintelor).

Programul e următorul:

```
#include <iostream>
using namespace std;
int x[20],a[20],k,m,i,n,nrv=0;
void sortare()
```

```
{
    int schimb=1,t;
    while(schimb)
    {
        schimb=0;
        for(i=1;i<n;i++)
            if(a[i]>a[i+1])
                {t=a[i];a[i]=a[i+1];a[i+1]=t;schimb=1;}
    }
}
void init()
{
    x[k]=0;
}
int succesor()
{
    int s=0;
    for(i=1;i<=k;i++)s+=a[x[i]];
    if(s<m){x[k]++;return 1;}
    else return 0;
}
int continuare()
{
    for(i=1;i<k;i++)
        if(x[i]>=x[i+1])return 0;
    return 1;
}
int solutie()
{
    int s=0;
    for(i=1;i<=k;i++)s+=a[x[i]];
    return (s==m);
}
void afiseaza()
{
    nrv++;cout<<'B'<<nrv<<"={";
    for(i=1;i<=k;i++)
        cout<<a[x[i]]<<' ';
    cout<<"}\n";
}
void backtracking()
{
    int as;
    k=1;init();
    while(k>0)
    {
        do {} while ((as=succesor())&&(!continuare()));
        if(as)
            if(solutie()) afiseaza();
            else {k++;init();}
        else k--;
    }
}
int main()
{
    cout << "suma= ";cin>>m;
    cout<<"n= ";cin>>n;
```

```
    for(i=1;i<=n;i++)
    {
        cout<<"a["<<i<<"]= ";cin>>a[i];
    }
    sortare();
    backtracking();
    if(!nrv)cout<<"Nu exista solutie";
    return 0;
}
```

Pentru a spori eficienţa algoritmului mai întâi am sortat elementele mulţimii şi am actualizat pe parcurs suma.

Complexitatea algoritmilor Backtracking

Dacă mulţimile $S_1, S_2, ..., S_n$ au acelaşi număr **s** de elemente, timpul necesar este $O(s^n)$, deci exponenţial. Dacă mulţimile $S_1, S_2, ..., S_n$ nu au acelaşi număr de elemente, atunci putem să luăm $\min\{s_1, s_2, ..., s_n\}$ şi să notăm această valoare cu **m** iar $\max\{s_1, s_2, ..., s_n\}$ să o notăm cu **M**. Atunci, timpul necesar este cuprins între $O(m^n)$ şi $O(M^n)$, prin urmare tot exponenţial.

În general, metoda Backtracking este ineficientă având complexitate exponenţială. Ea se utilizează totuşi în situaţiile în care se pot face optimizări în procesul de căutare, evitând, într-o etapă cât mai timpurie a analizei, căile care nu duc la soluţie. Există şi situaţii când rezolvarea prin metoda Backtracking nu poate fi înlocuită prin alte metode mai eficiente cum ar fi, de exemplu, problemele în care se cer *toate soluţiile acceptabile*.

Deci, având un timp de execuţie exponenţial, utilizarea metodei Backtracking se justifică numai atunci când nu cunoaştem o altă metodă cu eficienţă mai mare.

Backtracking recursiv

Până aici am studiat numai varianta iterativă pentru metoda Backtracking. În continuare vom aborda modalitatea de implementare recursivă a metodei Backtracking.

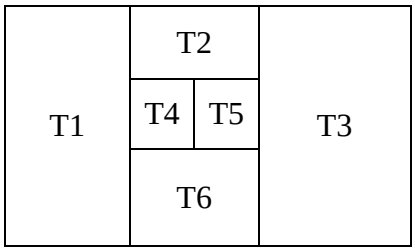
Presupunem că toate cele **n** mulţimi în care pot lua valori componentele vectorului sunt identice cu mulţimea finită **S={1,2,...,s}**. În această situaţie, varianta recursivă a metodei Backtracking este următoarea:

```
backtracking(întreg k);
  întreg i ;
┌dacă k = n+1 atunci retsol {Reţine soluţia}
│altfel
│     ┌ pentru i = 1, s execută
│     │     x[k] := i;
│     │  ┌ dacă continuare(k)
│     │  │              atunci Backtracking(k+1)
│     │  └■
│     └■
└■
```

Parametrul **k** din această rutină reprezintă numărul de ordine al componentei lui **x** ce urmează să primească o valoare din **S**. Pentru iniţierea procesului de generare a soluţiilor, în unitatea de program care iniţiază procesul backtracking se va utiliza apelul:

backtracking(1)

Problema 9.6. ***(Colorarea hărţilor).*** Fie dată o hartă ca cea din figura de mai jos, în care sunt prezentate schematic **6** ţări $\mathbf{T_1, T_2, T_3, T_4, T_5}$ și $\mathbf{T_6}$, dintre care unele au graniţe comune



O hartă cu 6 ţări

Presupunând că dispunem doar de trei culori (galben, albastru şi roşu), se cere să se determine toate variantele de colorare a hărţii, care să respecte condiţia ca orice două ţări vecine (care au frontieră comună) să fie colorate diferit.

Soluţie. Pentru a memora relaţia de vecinătate dintre ţări este folosită matricea *vecin* definită prin:

$$\mathbf{vecin[i,j]} = \begin{cases} \mathbf{1} & \text{dacă ţările } \mathbf{Ti} \text{ şi } \mathbf{Tj} \text{ sunt vecine} \\ \mathbf{0} & \text{dacă ţările } \mathbf{Ti} \text{ şi } \mathbf{Tj} \text{ nu sunt vecine} \end{cases}$$

Pentru harta din figura de mai sus, matricea **vecin**[2] corespunzătoare este următoarea:

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Problema poate fi generalizată uşor la o hartă cu **n** ţări ce trebuie colorată (cu restricţia menţionată) folosind un număr dat de **s** culori.

Programul este următorul:

```
#include <iostream>
using namespace std;
const int N=6;
const int S=3;
int nv=0; // Numarul variantei de colorare
int vecin[N+1][N+1],x[N];
int retsol()
{
    int i; nv++;
    cout<<"Varianta "<<nv<<" este: ";
    for (i=1;i<=N;i++)
        switch (x[i])
        {
            case 1 : cout<<" g ";break;
            case 2 : cout<<" a ";break;
```

[2]Matricea vecin mai poartă numele şi de *matrice de adiaceţă*.

```
            case 3 : cout<<" r ";break;
        };
    cout<<"\n";
    return 0;
}
int continuare(int k)
{
    int i=1,b=1;
    while (b&&(i<k))
        {
            b=!(vecin[k][i]&&(x[k]==x[i]));
            i++;
        }
    return b;
}
int citeste()
{
    int i,j;
    for (i=1;i<=N;i++)
        for (j=1;j<=N;j++) vecin[i][j]=0;
    for (i=1;i<=N;i++)
        {
            cout<<"Tara "<<i<<" vecina cu (0 pt. finalizare): ";
            cin>>j;
            while (j!=0)
                {
                    vecin[i][j]=1;vecin[j][i]=1;
                    cin>>j;
                }
        }
    cout<<"\n";
    return 0;
}
int backtracking(int k)
{
    int i;
    if (k==N+1) retsol();
    else
        for (i=1;i<=S;i++)
        {
            x[k]=i;
            if (continuare(k)) backtracking(k+1);
        }
    return 0;
}
int main()
{
    citeste();
    cout<<"Tara:             1  2  3  4  5  6"<<'\n';
    backtracking(1);
    return 0;
}
```

Problema 9.7. ***(Plata unei sume folosind monede de valori date)*** Se dau n tipuri de monede având valori de $\mathbf{v_1, v_2, \ldots, v_n}$. Se cer toate modalităţile de plată a sumei s utilizând aceste monede. Se presupune că fiecare monedă există într-un număr nelimitat de exemplare.

Soluţie. Valorile celor **n** monede sunt reţinute de vectorul **v**. Astfel, **v[1]** va reţine valoarea monedei de tipul **1**, **v[2]** valoarea monedei de tipul **2**, ş.a.m.d. Numărul de monede din fiecare tip va fi reţinut de vectorul **sol**. Astfel, **sol[1]** reţine numărul de monede de tipul **1**, **sol[2]** reţine numărul de monede de tipul **2**, ş.a.m.d.

Orice soluţie are exact **n** componente (pentru monedele care nu sunt luate în calcul în vectorul **sol** se reţine **0**, din acest motiv la început, fiecare componentă a vectorului sol va fi iniţializată cu valoarea **-1**).

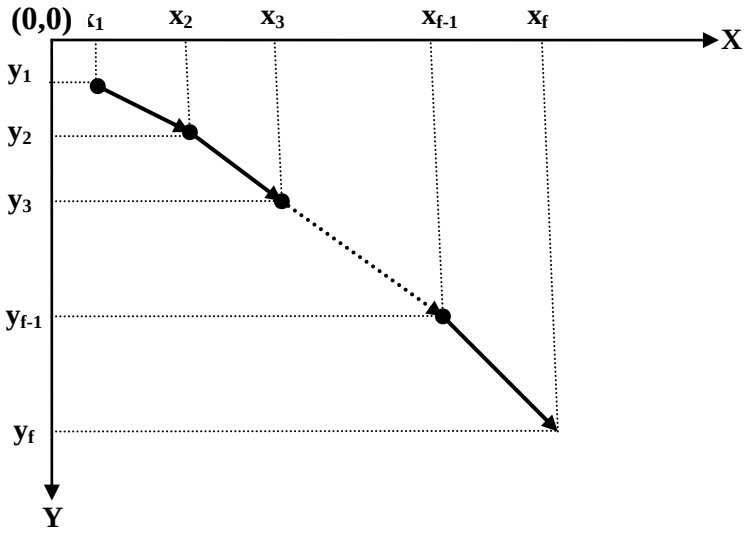
La pasul **k** avem la dispoziţie suma obţinută:
s=v[1]*sol[1]+v[2]*sol[2]+…+v[k-1]*sol[k-1].
O soluţie avem numai dacă:
s=v[1]*sol[1]+v[2]*sol[2]+…+v[n]*sol[n]=Suma
Programul este următorul:

```
#include <iostream>
using namespace std;
int v[100],sol[100],n,i,s,suma;
void backtracking(int k)
{
    if (s==suma)
    {
        cout<<"Solutie\n";
        for (i=1;i<=k-1;i++)
            if (sol[i]!=0) cout<<sol[i]<<" monede de " <<v[i]<<"\n";
        cout<<"\n";
    }
    else
    {
        sol[k]=-1;
        while (((s+sol[k]*v[k])<suma)&&(k<n+1))
        {
            sol[k]++;
            s=s+sol[k]*v[k];
            backtracking(k+1);
            s=s-sol[k]*v[k];
        }
    }
}
int main()
{
    cout<<"Suma = "; cin>>suma;
    cout<<"\n n= "; cin>>n;
    for (i=1;i<=n;i++)
    {
        cout<<"v["<<i<<"]= ";
        cin>>v[i];
    }
    backtracking(1);
}
```

Vă rămâne ca exerciţiu rezolvarea problemei în situaţia în care numărul de monede este limitat pentru fiecare tip de monedă în parte. Astfel, numărul de monede de valoare $\mathbf{v_i}$ de tipul **i** este reţinut în componenta $\mathbf{b_i}$ din vectorul **b**. Astfel, pentru **s=100**, **v=(2,5,50)**, **b=(10,9,4)**, varianta **s=10x5+1x50** nu corespunde cerinţei deoarece nu avem la dispoziţie **10** monede de **5**, ci doar **9**.

Backtracking generalizat (în plan)

În rezolvarea problemelor care necesită determinarea unor trasee de deplasare a unor obiecte în plan în zone codificate matriceal se poate utiliza o generalizare a metodei Backtracking. Întrucât soluţiile unor astfel de probleme nu se mai reprezintă printr-un singur vector ci prin doi vectori $\mathbf{x=(x_1,x_2,\ldots,x_f)}$ şi $\mathbf{y=(y_1,y_2,\ldots,y_f)}$ în care $\mathbf{(x_i,y_i)}$ reprezintă indicii elementelor matricii ce constituie traseul de deplasare a obiectului, aceşti indici putem să spunem că reprezintă coordonatele unor puncte din planul **XOY** reprezentând traseul de deplasare.

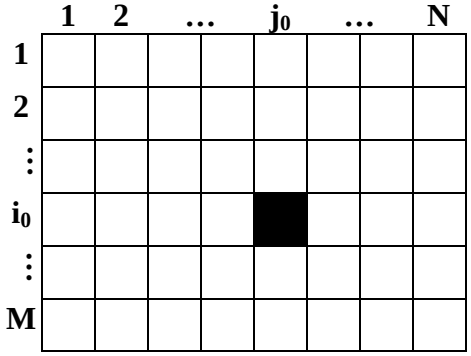


Mergând şi mai departe cu generalizarea, putem să aplicăm metoda într-un spaţiu cu **n** dimensiuni (soluţia fiind reprezentată prin **n** vectori).

În unele situaţii problemele nu cer afişarea traseului de deplasare ci se cere să se determine doar dacă există sau nu un asemenea traseu.

Să studiem în continuare la modul general o problemă în spaţiul cu două dimensiuni (în plan).

Fie un caroiaj având **m** linii şi **n** coloane ca cel din figura următoare.



Exemplu de caroiaj

Să presupunem că un mobil (piesă de şah, robot etc.) pleacă din punctul (pătratul) iniţial **(i0,j0)**, unde **i0** reprezintă numărul liniei, iar **j0** reprezintă numărul coloanei şi că el se deplasează conform unor reguli sintetizate (memorate) în doi vectori **di** şi **dj**

având o dimensiune **d** dată, cu următoarea semnificaţie: dacă mobilul se află în punctul de coordonate **(i, j)**, el se poate deplasa doar într-unul dintre punctele **(i+di[k],j+dj[k])**, **k=1,2,...,d**, bineînţeles cu condiţia ca să nu iasă din caroiaj.
De exemplu, să presupunem că mobilul se poate deplasa doar astfel:
- cu o poziţie la dreapta pe aceeaşi linie;
- cu o poziţie la stânga pe aceeaşi linie;
- cu o poziţie în sus pe aceeaşi coloană;
- cu o poziţie în jos pe aceeaşi coloană.

Ca urmare a acestor posibilităţi de mişcare a mobilului, vom avea: **di=(0,0,1,-1)** şi **dj=(1,-1,0,0)**, astfel că, de exemplu, pentru **k=2** vom avea **di[k]=0** şi **dj[k]=-1** ceea ce are următoarea semnificaţie: mobilul se deplasează zero linii şi o coloană în jos (semnul minus indicând mişcarea mobilului la stânga şi în jos iar semnul plus indicând mişcarea la dreapta şi în sus).

În practică apar de nenumărate ori astfel de situaţii şi, mai mult, în unele probleme, se consideră că anumite pătrate ale caroiajului sunt "ocupate" (ceea ce înseamnă că mobilul nu poate fi plasat prin deplasări succesive într-un astfel de pătrat). Bineînţeles că pătratul iniţial se presupune că nu este ocupat.

Frecvent se întâlnesc probleme ce constau în:

- simularea mişcării mobilului;
- determinarea pătratelor în care mobilul poate să ajungă prin deplasări permise;
- determinarea unei succesiuni de deplasări în urma cărora mobilul poate să ajungă într-un punct **(if,jf)** dat, dacă o astfel de succesiune există;
- determinarea celei mai scurte succesiuni de deplasări în urma cărora mobilul poate să ajungă într-un punct **(if,jf)** dat, accesibil din punctul iniţial (traseul optim).

Astfel de probleme pot fi rezolvate şi prin aplicarea metodei Backtracking (dar în unele situaţii sunt mult mai eficiente alte metode cum ar fi, de exemplu, metoda Branch and Bound).

Pentru rezolvarea unor astfel de probleme cu metoda Backtracking, vectorului **x** întâlnit până acum în acest capitol va avea în continuare o nouă semnificaţie. Elementele sale vor lua valori în mulţimea **{1,2,...,d}** iar semnificaţia sa este următoarea: dacă după **k-1** mutări mobilul a ajuns în poziţia **(i,j)**, atunci, după următoarea sa mutare, mobilul va ajunge în poziţia **(i+di[x[k]],j+dj[x[k]])**. Cu alte cuvinte, vectorul **x** va conţine numărul de ordine al deplasărilor permise (adică va conţine indicele la care s-a ajuns în tablourile **di** şi **dj**).

Funcţia *continuare* va fi tot o funcţie booleană ce permite să determinăm dacă o anumită încercare de deplasare este permisă, adică nu se ajunge la vreuna dintre eventualele poziţii ocupate şi nu se iese din caroiaj.

Rutina Backtracking este asemănătoare cu cele utilizate anterior în acest capitol numai că vor apărea în plus în ea variabilele **i** şi **j** cu semnificaţia că **(i,j)** este poziţia curentă a mobilului.

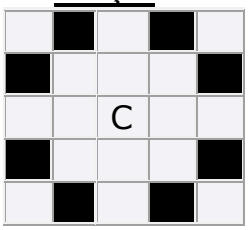
Să aplicăm toate acestea la o problemă concretă.

Problema 9.8. ***(Săritura calului[3]).*** Fie dată o "tablă de şah", de dimensiune **n**x**n**. Se pleacă cu un cal din poziţia **(1,1)**, adică din colţul din stânga sus. Nefiind interzisă nici o poziţie, se cere să se efectueze cu calul, conform regulilor jocului de şah, o

[3]Această problemă a fost enunţată explicit pentru prima dată de către Leonhard Euler.

succesiune de mutări astfel încât calul să treacă o dată şi numai o dată prin fiecare pătrat al tablei de şah.

Soluţie. Ca urmare a posibilităţilor mişcării calului notat în figura de mai jos cu C,



pătrăţelele negre reprezentând locurile unde poate fi mutat calul, vom avea:

di=(1,2,2,1,-1,-2,-2,-1) şi

dj=(2,1,-1,-2,-2,-1,1,2).

Se va utiliza o matrice **a**. Pentru ca verificările efectuate de funcţia continuare să fie mai uşoare, vom "borda" matricea **a** cu două prime linii, cu două ultime linii, cu două prime coloane şi cu două ultime coloane, pătratele nou introduse prin bordare intrând în categoria celor ocupate (interzise). Bordarea s-a făcut cu câte **2+2** linii şi **2+2** coloane deoarece calul poate să sară cu câte două pătrăţele la dreapta, sus, stânga sau jos. Drept urmare, liniile şi coloanele vor fi numerotate cu **-1,0,1,...,n,n+1,n+2**. Ne propunem ca în final matricea *a* să ilustreze traseul calului, deci să conţină elementele **1,2,...,n^2** astfel încât o săritură a calului de pe o poziţie pe poziţia următoare să corespundă la numere succesive în matrice. Poziţiile interzise vor primi valoarea **-1**, iar cele libere (poziţiile efective ale tablei de şah) vor primi valoarea **0** (aceste iniţializări fiind efectuate de rutina *iniţializare*). Prin urmare, calul poate să treacă în poziţia **(i,j)** dacă şi numai dacă **a[i,j]=0**.

Pentru a evita determinarea de soluţii simetrice, vom pune **a[2,3]=2** şi vom începe cu **k=2**, **a[1,1]=1** şi **a[2,3]=2**. De altfel vom urmări determinarea unei singure soluţii pentru că vom vedea că şi aşa timpul de execuţie va fi destul de mare chiar pentru valori mici ale lui **n**. Deficienţa acestui algoritm constă în faptul că ordinea de deplasare a calului plecând din poziţia iniţială este prestabilită prin vectorii **di** şi **dj**, fixaţi de la început.

Rutina *backtracking()* se termină când au fost ocupate toate cele n^2 câmpuri ale tablei de şah. Programul este următorul:

```
#include <iostream>
using namespace std;
int di[8]={1,2,2,1,-1,-2,-2,-1},
    dj[8]={2,1,-1,-2,-2,-1,1,2},
    a[13][13],x[100],
    i,n,n2;
int initializare()
{
    int i,j;
    for (i=0;i<=n+3;i++)
        {
            a[0][i]=1; a[1][i]=1;
            a[n+2][i]=1; a[n+3][i]=1;
            a[i][0]=1; a[i][1]=1;
            a[i][n+2]=1; a[i][n+3]=1;
        }
    for (i=2;i<=n+1;i++)
```

```
        for (j=2;j<=n+1;j++) a[i][j]=0;
    return 0;
}
int continuare(int i,int j)
{
    if (a[i][j]==0) return 1;
    else return 0;
};
int retsol()
{
    int i,j;
    for (i=2;i<=n+1;i++)
        {
            for (j=2;j<=n+1;j++)
                if (a[i][j]>9) cout<<a[i][j]<<" ";
                else cout<<" "<<a[i][j]<<" ";
            cout<<"\n";
        }
    return 0;
}
int backtracking()
{
    int k,i,j;
        a[2][2]=1;a[3][4]=2;
        i=3;j=4;k=2;x[k]=-1;
    while (k>1)
        if (k==n2) k=0;
        else
            if (x[k]<7)
            {
                x[k]++;
                if (continuare(i+di[x[k]],j+dj[x[k]]))
                {
                    i=i+di[x[k]];j=j+dj[x[k]];
                    k++;a[i][j]=k;x[k]=-1;
                }
            }
            else
            {
                a[i][j]=0;k--;
                i=i-di[x[k]];j=j-dj[x[k]];
            }
    return 0;
}
int main()
{
    cout<<"n= ";cin>>n;cout<<"\n";
    initializare(); n2=n*n;
    backtracking();
    retsol();
    return 0;
}
```

Problema săriturii calului poate fi rezolvată folosind şi alte metode de programare (ca, de exemplu, metoda Branch and bound sau algoritmii probabilistici cum ar fi algoritmii Las Vegas) ce permit obţinerea soluţiei într-un timp mult mai scurt.

Variante ale metodei Backtracking

Până acum am prezentat metoda Backtracking în forma sa standard. În practică se întâlnesc nenumărate "abateri" de la această formă standard a metodei Backtracking, dintre care cele mai uzuale sunt următoarele:

- soluţia **x** poate avea un număr variabil de componente (deci nu neapărat **n**);
- dintre soluţii trebuie aleasă una care să maximizeze (minimizeze) o funcţie de cost dată iar această soluţie se va numi *soluţie optimă*.

Problema care urmează ilustrează ambele aspecte menţionate mai sus.

Problema 9.9. ***(Subşir maxim).*** Fie dat un vector **a** cu **n** componente întregi. Să se determine un subşir al lui **a** (ale cărui elemente sunt luate dintre componentele vectorului **a** nu neapărat unul imediat după celălalt) de lungime maximă, subşir ale cărui elemente sunt în ordine crescătoare. Mai precis, se caută cea mai mare valoare **k** pentru care există indicii **x[1]<x[2]< ... <x[k]** astfel încât **a[x[1]]<a[x[2]]< ... <a[x[k]]**.

Exemplu. Pentru **n=8** şi **a=(1,2,5,3,9,4,7,8)**, se obţine **k=6**, o secvenţă de indici satisfăcând condiţiile date fiind **1,2,4,6,7,8**, căreia îi corespunde subşirul **(1,2,3,4,7,8)** format din elemente ale vectorului **a** care sunt în ordine crescătoare (în vectorul luat ca exemplu ele sunt subliniate).

Soluţie. O soluţie va fi un vector **x** ce conţine secvenţa de indici ai tabloului **a** cu restricţiile menţionate în problemă şi care satisface în plus condiţia că el nu mai poate fi suplimentat în plus cu încă o componentă. Astfel, pentru exemplul dat, cazul particular **x=(1,2,3,7,8)** este considerat soluţie (soluţia în acest caz fiind subşirul **(1,2,5,7,8)** corespunzător indicilor daţi de **x**); în schimb **x=(1,2,3)** nu este soluţie, deoarece poate fi suplimentat, de exemplu, cu componenta **7**. Lungimea efectivă va fi memorată în variabila **k**.

Funcţia de *cost* ataşează fiecărei soluţii lungimea sa.

În variabila **kf** va fi memorată lungimea unei soluţii optime, iar soluţia optimă va fi memorată în vectorul **xf**. Vom pune iniţial **kf=0**, urmând ca de fiecare dată când determinăm o soluţie **x** de lungime **k** să verificăm dacă **k>kf**, în caz afirmativ actualizând pe **xf** şi **kf** (acest lucru realizându-l rutina *retsol*).

Cerinţa problemei ca indicii să fie în ordine crescătoare impune ca pentru **k>1**, iniţializarea pentru **x[k]** să fie:

x[k]:=x[k-1]

Pentru a uşura verificările realizate de funcţia *continuare*, vom introduce două componente suplimentare: **a[0]** care va fi iniţializată cu cea mai mică valoare întreagă şi, respectiv **a[n+1]** care va fi iniţializată cu cea mai mare valoare întreagă; în acest mod putând identifica uşor faptul că o secvenţă crescătoare de indici **x[1]<x[2]<...<x[k]** nu poate fi suplimentată, verificând condiţia **x[k]<n+1**.

Programul este următorul:

```
#include <iostream>
using namespace std;
#define MAXINT 32767
int a[100],x[100],xf[100],n,kf,i;
int continuare(int k)
{
    if (a[x[k-1]]<a[x[k]]) return 1;
    else return 0;
}
void retsol(int k)
{
```

```
    int i;
    if (k>kf)
    {
        kf=k;
        for (i=1;i<=kf;i++) xf[i]=x[i];
    }
}
void backtracking()
{
    int k=1;
    x[0]=0; x[1]=0;
    while (k>0)
        if (x[k]<n+1)
        {
            x[k]++;
            if (continuare(k))
                {
                    if (x[k]==n) { retsol(k); k--; }
                    else { k++; x[k]=x[k-1]; }
                }
        }
        else k--;
}
int main()
{
    cout<<"n= "; cin>>n;cout<<'\n';
    cout<<"Elementele vectorului\n";
    for (i=1;i<=n;i++) cin>>a[i];
    a[0]=-MAXINT; n++; a[n]=MAXINT;
    kf=0; backtracking();
    cout<<"O secventa crescatoare de lg. max. este:\n";
    for (i=1;i<=kf-1;i++)
        cout<<a[xf[i]]<<" ";
    cout<<"\n";
    cout<<" si are lungimea "<<kf-1;
    return 0;
}
```

Teste grilă

1. Utilizând metoda backtracking se generează în ordine lexicografică cuvintele de câte patru litere din mulţimea **A={a,b,c,d,e}**, cuvinte care nu conţin două vocale alăturate. Primele opt cuvinte generate sunt, în ordine: **abab**, **abac**, **abad**, **abba**, **abbb**, **abbc**, **abbd**, **abbe**. Câte dintre cuvintele generate încep cu litera **b** şi se termină cu litera **e**?

a. 9 **b. 15** **c. 12** **d. 20**

(Bacalaureat 2009, varianta 1, III, 1)

2. Utilizând metoda backtracking se generează în ordine lexicografică cuvintele de câte patru litere din mulţimea **A={a,b,c,d,e}**, cuvinte care nu conţin două vocale alăturate. Primele opt cuvinte generate sunt, în ordine: **abab**, **abac**, **abad**, **abba**, **abbb**, **abbc**, **abbd**, **abbe**. Care este ultimul cuvânt generat?

a. edcb **b. eeee** **c. edde** **d. eded**

(Bacalaureat 2009, varianta 2, III, 1)

3. Utilizând metoda backtracking se generează în ordine lexicografică cuvintele de câte patru litere din mulţimea **A={a,b,c,d,e}**, cuvinte care nu conţin două vocale alăturate. Primele opt cuvinte generate sunt, în ordine: **abab**, **abac**, **abad**, **abba**, **abbb**, **abbc**,

abbd, **abbe**. Care este penultimul cuvânt generat?

a. edec **b. eded** **c. edde** **d. edcb**

(Bacalaureat 2009, varianta 3, III, 1)

4. Utilizând metoda backtracking se generează în ordine lexicografică cuvintele de câte patru litere din mulţimea **A={a,b,c,d,e}**, cuvinte care nu conţin două vocale alăturate. Primele opt cuvinte generate sunt, în ordine: **abab**, **abac**, **abad**, **abba**, **abbb**, **abbc**, **abbd**, **abbe**. Care este antepenultimul cuvânt generat?

a. edde **b. eddb** **c. edeb** **d. edcb**

(Bacalaureat 2009, varianta 4, III, 1)

5. Folosind modelul combinărilor se generează numerele naturale cu câte trei cifre distincte din mulţimea **{1,2,3,7}**, numere cu cifrele în ordine strict crescătoare, obţinându-se, în ordine: **123**, **127**, **137**, **237**. Dacă se utilizează exact aceeaşi metodă pentru a genera numerele naturale cu patru cifre distincte din mulţimea **{1,2,3,4,5,6,7,8}**, câte dintre numerele generate au prima cifră **2** şi ultima cifră **7**?

a. 8 **b. 3** **c. 4** **d. 6**

(Bacalaureat 2009, varianta 5, III, 1)

6. Utilizând metoda backtracking sunt generate numerele de **3** cifre, având toate cifrele distincte şi cu proprietatea că cifrele aflate pe poziţii consecutive sunt de paritate diferită. Ştiind că primele şase soluţii generate sunt, în această ordine, **103**, **105**, **107**, **109**, **123**, **125**, care este a zecea soluţie generată?

a. 145 **b. 147** **c. 230** **d. 149**

(Bacalaureat 2009, varianta 6, III, 1)

7. Folosind tehnica bactracking un elev a scris un program care generează toate numerele de câte **n** cifre (**0<n≤9**), cifrele fiind în ordine strict crescătoare. Dacă **n** este egal cu **5**, scrieți în ordine crescătoare toate numerele având cifra unităților **6**, care vor fi generate de program.

(Bacalaureat 2009, varianta 7, III, 2)

8. Utilizând metoda backtracking sunt generate numerele de **3** cifre care au cifrele în ordine crescătoare, iar cifrele aflate pe poziţii consecutive sunt de paritate diferită. Ştiind că primele cinci soluţii generate sunt, în această ordine, **123**, **125**, **127**, **129**, **145**, care este cel de al **8**-lea număr generat?

a. 169 **b. 149** **c. 167** **d. 147**

(Bacalaureat 2009, varianta 8, III, 1)

9. Utilizând metoda backtracking, sunt generate în ordine crescătoare toate numerele de **3** cifre, astfel încât cifrele sunt în ordine crescătoare, iar cifrele aflate pe poziţii consecutive sunt de paritate diferită. Ştiind că primele trei soluţii generate sunt, în această ordine, **123**, **125**, **127**, scrieţi toate numerele generate care au suma cifrelor egală cu **12.**

(Bacalaureat 2009, varianta 9, III, 2)

10. Un elev a scris un program care, folosind metoda backtracking, generează toate numerele de câte **5** cifre, cifrele fiind în ordine strict crescătoare. Scrieţi toate numerele generate de program care au prima cifră **5.**

(Bacalaureat 2009, varianta 10, III, 2)

11. Un algoritm de tip backtracking generează, în ordine lexicografică, toate şirurile de **5** cifre **0** şi **1** cu proprietatea că nu există mai mult de două cifre **0** pe poziţii consecutive. Primele **7** soluţii generate sunt: **00100**, **00101**, **00110**, **00111**, **01001**, **01010**, **01011**. Care este a **8**-a soluţie generată de acest algoritm?

a. 01110 **b. 01100** **c. 01011** **d. 01101**

(Bacalaureat 2009, varianta 11, III, 1)

12. Pentru a scrie valoarea **10** ca sumă de numere prime se foloseşte metoda backtracking şi se generează, în această ordine, sumele distincte: **2+2+2+2+2**, **2+2+3+3**, **2+3+5**, **3+7**, **5+5**. Folosind exact aceeaşi metodă, se scrie valoarea **9** ca sumă de numere prime. Care sunt primele trei soluţii, în ordinea generării lor?

(Bacalaureat 2009, varianta 12, III, 2)

13. Trei băieţi, **Alin**, **Bogdan** şi **Ciprian**, şi trei fete, **Delia**, **Elena** şi **Felicia**, trebuie să formeze o echipă de **3** copii, care să participe la un concurs. Echipa trebuie să fie mixtă(adică să conţină cel puţin o fată şi cel puţin un băiat). Ordinea copiilor în echipă este importantă deoarece aceasta va fi ordinea de intrare a copiilor în concurs (de exemplu echipa **Alin**, **Bogdan**, **Delia** este diferită de echipa **Bogdan**, **Alin**, **Delia**).

- Câte echipe se pot forma, astfel încât din ele să facă parte simultan **Alin** şi **Bogdan**?
- Daţi exemplu de o echipă corect formată din care să nu facă parte nici **Alin** şi nici **Bogdan**.

(Bacalaureat 2009, varianta 13, III, 2)

14. Utilizând metoda backtracking se generează permutările cuvântului **info**. Dacă primele trei soluţii generate sunt: **fino**, **fion**, **fnio** care este cea de-a cincea soluţie?

a. foin **b. fnoi** **c. foni** **d. ifon**

(Bacalaureat 2009, varianta 14, III, 1)

15. Câte numere cu exact două cifre pot fi construite folosind doar cifre pare distincte?

a. 12 **b. 16** **c. 20** **d. 25**

(Bacalaureat 2009, varianta 15, III, 1)

16. Un algoritm generează în ordine crescătoare toate numerele de **n** cifre, folosind doar cifrele **3**, **5** şi **7**. Dacă pentru **n=5**, primele cinci soluţii generate sunt **33333**, **33335**, **33337**, **33353**, **33355**, precizaţi care sunt ultimele **trei** soluţii generate, în ordinea generării.

(Bacalaureat 2009, varianta 16, III, 2)

17. Un algoritm generează în ordine descrescătoare toate numerele de **5** cifre, fiecare dintre ele având cifrele în ordine strict crescătoare. Ştiind că primele cinci soluţii generate sunt **5678946789**, **45789**, **45689**, **45679**, precizaţi care sunt ultimele **trei** soluţii generate, în ordinea generării.

(Bacalaureat 2009, varianta 17, III, 2)

18. Un algoritm generează, în ordine lexicografică, toate şirurile alcătuite din câte **n** cifre binare(**0** şi **1**). Ştiind că pentru **n=5**, primele patru soluţii generate sunt **00000**, **00001**, **00010**, **00011**, precizaţi care sunt ultimele **trei** soluţii generate, în ordinea obţinerii lor.

(Bacalaureat 2009, varianta 18, III, 2)

19. Un algoritm generează în ordine crescătoare, toate numerele de **n** cifre (**n<9**), cu cifre distincte, care nu au două cifre pare alăturate. Dacă pentru **n=5**, primele cinci soluţii generate sunt **10325**, **10327**, **10329**, **10345**, **10347**, precizaţi care sunt următoarele **trei** soluţii generate, în ordinea obţinerii lor.

(Bacalaureat 2009, varianta 19, III, 2)

20. Un algoritm generează în ordine descrescătoare, toate numerele de **n** cifre (**n<9**), cu cifrele în ordine strict crescătoare, care nu au două cifre pare alăturate. Dacă pentru **n=5**, primele cinci soluţii generate sunt **56789**, **45789**, **45679**, **45678**, **36789**, precizaţi

care sunt următoarele **trei** soluţii generate, în ordinea obţinerii lor.

(Bacalaureat 2009, varianta 20, III, 2)

21. In timpul procesului de generare a permutărilor mulţimii **{1,2,…,n}** prin metoda backtracking, în tabloul unidimensional **x** este plasat un element x_k ($1 \leq k \leq n$). Acesta este considerat valid dacă este îndeplinită condiţia:

a. $x_k \notin \{x_1, x_2, \ldots, x_{k-1}\}$ **b.** $x_k \neq x_{k-1}$

c. $x_k \notin \{x_1, x_2, \ldots, x_n\}$ **d.** $x_k \neq x_{k-1}$ şi $x_k \neq x_{k+1}$

(Bacalaureat 2009, varianta 22, III, 1)

22. Generând şirurile de maximum **3** caractere distincte din mulţimea **{A,B,C,D,E}**, ordonate lexicografic, obţinem succesiv: **A**, **AB**, **ABC**, **ABD**, **...** . Ce şir va fi generat imediat după **BAE**?

a. BCA **b. CAB** **c. BC** **d. BEA**

(Bacalaureat 2009, varianta 24, III, 1)

23. Un program citeşte o valoare naturală nenulă impară pentru **n** şi apoi generează şi afişează în ordine crescătoare lexicografic toate combinaţiile formate din **n** cifre care îndeplinesc următoarele proprietăţi:

- încep şi se termină cu **0**;
- modulul diferenţei între oricare două cifre alăturate dintr-o combinaţie este **1**.

Astfel, pentru **n=5**, combinaţiile afişate sunt, în ordine, următoarele: **01010**, **01210**. Dacă se rulează acest program şi se citeşte pentru **n** valoarea **7**, imediat după combinaţia **0101210** va fi afişată combinaţia:

a. 0121210 **b. 0123210** **c. 0111210** **d. 0121010**

(Bacalaureat 2009, varianta 25, III, 1)

24. Pentru generarea numerelor cu **n** cifre formate cu elementele mulţimii **{0,2,9}** se utilizează un algoritm backtracking care, pentru **n=2**, generează, în ordine, numerele **20,22,29,90,92,99**. Dacă **n=4** şi se utilizează acelaşi algoritm, care este numărul generat imediat după numărul **2009**?

a. 2002 **b. 2020** **c. 2090** **d. 2010**

(Bacalaureat 2009, varianta 26, III, 1)

25. Pentru generarea în ordine crescătoare a numerelor cu **n** cifre formate cu elementele mulţimii **{0,2,8}** se utilizează un algoritm backtracking care, pentru **n=2**, generează, în ordine, numerele **20,22,28,80,82,88**. Dacă **n=4** şi se utilizează acelaşi algoritm, precizaţi câte numere generate sunt divizibile cu **100**?

a. 8 **b. 90** **c. 6** **d. 10**

(Bacalaureat 2009, varianta 27, III, 1)

26. În câte dintre permutările elementelor mulţimii **{'I','N','F','O'}** vocalele apar pe poziţii consecutive?

a. 24 **b. 6** **c. 12** **d. 4**

(Bacalaureat 2009, varianta 29, III, 1)

27. Pentru generarea numerelor cu **n** cifre formate cu elementele mulţimii **{0,4,8}** se utilizează un algoritm backtracking care, pentru **n=2**, generează, în ordine, numerele **40,44,48,80,84,88**. Dacă **n=4** şi se utilizează acelaşi algoritm, care este numărul generat imediat după numărul **4008**?

a. 4040 **b. 4004** **c. 4080** **d. 8004**

(Bacalaureat 2009, varianta 30, III, 1)

28. Având la dispoziţie cifrele **0**, **1** şi **2** putem genera, în ordine crescătoare, numere care au suma cifrelor egală cu **2** astfel încât primele **6** numere generate sunt, în această

ordine: **2**, **11**, **20**, **101**, **110**, **200**. Folosind acelaşi algoritm se generează numere cu cifrele **0**, **1**, **2** şi **3** care au suma cifrelor egală cu **4**. Care va fi al **7**-lea număr din această generare ?

a. 103 **b. 301** **c. 220** **d. 130**

(Bacalaureat 2009, varianta 31, III, 1)

29. In vederea participării la un concurs, elevii de la liceul sportiv au dat o probă de selecţie, în urma căreia primii **6** au obţinut punctaje egale. În câte moduri poate fi formată echipa selecţionată ştiind că poate avea doar **4** membri, aleşi dintre cei **6**, şi că ordinea acestora în cadrul echipei nu contează?

a. 24 **b. 30** **c. 15** **d. 4**

(Bacalaureat 2009, varianta 32, III, 1)

30. Folosind un algoritm de generare putem obţine numere naturale de **k** cifre care au suma cifrelor egală cu un număr natural **s**. Astfel, pentru valorile **k=2** şi **s=6** se generează, în ordine, numerele: **15**, **24**, **33**, **42**, **51**, **60**. Care va fi al treilea număr generat pentru **k=4** şi **s=5**?

a. 1301 **b. 1022** **c. 2201** **d. 1031**

(Bacalaureat 2009, varianta 33, III, 1)

31. Completarea unui bilet de LOTO presupune colorarea a **6** numere dintre cele **49**, înscrise pe bilet. O situaţie statistică pe o anumită perioadă de timp arată că cele mai frecvente numere care au fost extrase la LOTO sunt: **2, 20, 18, 38, 36, 42, 46, 48**. Câte bilete de **6** numere se pot completa folosind doar aceste valori, ştiind că numărul **42** va fi colorat pe fiecare bilet?

a. 21 **b. 6!** **c. 42** **d. 56**

(Bacalaureat 2009, varianta 34, III, 1)

32. Utilizăm metoda backtracking pentru generarea tuturor modalităţilor de a scrie numărul **9** ca sumă a cel puţin două numere naturale nenule distincte. Termenii fiecărei sume sunt în ordine strict crescătoare. Soluţiile se generează în ordinea: **1+2+6**, **1+3+5**, **1+8**, **2+3+4**, **2+7**, **3+6** şi **4+5.** Se aplică exact aceeaşi metodă pentru scrierea lui **12**. Scrieţi, în ordinea generării, toate soluţiile de forma **2+...**

(Bacalaureat 2009, varianta 36, III, 2)

33. Se utilizează un algoritm pentru a genera în ordine lexicografică inversă toate permutările mulţimii {**1**,**2**,**3**,**4**,**5**}. Primele patru permutări generate sunt: **54321**, **54312**, **54231**, **54213.** A cincea permutare este:

a. 53421 **b. 54321** **c. 54132** **d. 54123**

(Bacalaureat 2009, varianta 37, III, 1)

34. Utilizăm metoda backtracking pentru generarea tuturor modalităţilor de a scrie numărul **9** ca sumă a cel puţin două numere naturale nenule distincte. Termenii fiecărei sume sunt în ordine strict crescătoare. Soluţiile se generează în ordinea: **1+2+6**, **1+3+5**, **1+8**, **2+3+4**, **2+7**, **3+6** şi**4+5**. Se aplică exact aceeaşi metodă pentru scrierea lui **8**. Câte soluţii vor fi generate?

a. 3 **b. 4** **c. 6** **d. 5**

(Bacalaureat 2009, varianta 38, III, 1)

35. Utilizăm metoda backtracking pentru generarea tuturor modalităţilor de a scrie numărul **6** ca sumă a cel puţin două numere naturale nenule. Termenii fiecărei sume sunt în ordine crescătoare. Soluţiile se generează în ordinea: **1+1+1+1+1+1**, **1+1+1+1+2**, **1+1+1+3**, **1+1+4**, **1+5**, **2+2+2**, **2+4** şi **3+3**. Se aplică exact aceeaşi metodă pentru scrierea lui **9.** Care este penultima soluţie?

a. 3+3+3 **b. 3+6** **c. 4+5** **d. 2+7**

(Bacalaureat 2009, varianta 39, III, 1)

36. Utilizăm metoda backtracking pentru generarea tuturor modalităţilor de a scrie numărul **6** ca sumă a cel puţin două numere naturale nenule. Termenii fiecărei sume sunt în ordine crescătoare. Soluţiile se generează în ordinea: **1+1+1+1+1+1**, **1+1+1+1+2**, **1+1+1+3**, **1+1+4**, **1+5**, **2+2+2**, **2+4** şi **3+3**. Se aplică exact aceeaşi metodă pentru scrierea lui **9**. Câte soluţii de forma **2+...** vor fi generate?

a. 2 **b. 3** **c. 4** **d. 5**

(Bacalaureat 2009, varianta 40, III, 1)

37. Utilizând metoda backtracking se generează toate permutările mulţimii **{1,2,3,4}**. Dacă primele trei permutări generate sunt, în acestă ordine: **1234**, **1243**, **1324** precizaţi care este permutarea generată imediat după **3412**.

a. 3214 **b. 3413** **c. 4123** **d. 3421**

(Bacalaureat 2009, varianta 42, III, 1)

38. Utilizând metoda backtracking se generează numerele formate din câte **3** cifre distincte din mulţimea **{1,3,5,7}**. Dacă primele trei numere generate sunt, în acestă ordine: **135**, **137**, **153** care este cel de-al patrulea număr generat?

a. 315 **b. 173** **c. 157** **d. 357**

(Bacalaureat 2009, varianta 43, III, 1)

39. Utilizând metoda backtracking se generează toate cuvintele de câte **3** litere din mulţimea **{a,b,c}**. Dacă primele patru cuvinte generate sunt, în acestă ordine: **aaa**, **aab**, **aac**, **aba**, care este cel de-al optulea cuvânt generat?

a. acb **b. acc** **c. aca** **d. bca**

(Bacalaureat 2009, varianta 45, III, 1)

40. Un program generează, în ordine crescătoare, numerele naturale de exact **5** cifre din mulţimea {**1**, **2**, **3**, **4**, **5}**. Fiecare dintre numerele generate are cifrele distincte două câte două. Primele **3** numere astfel generate sunt: **12345**, **12354**, **12435**. Care este numărul generat imediat după **12543**?

a. 15342 **b. 12534** **c. 13245** **d. 13452**

(Bacalaureat 2009, varianta 46, III, 1)

41. Într-un penar sunt opt creioane: trei sunt roşii, două albastre şi trei negre. Dacă scoatem din penar cinci creioane, câte posibilităţi există ca cel puţin două dintre ele să fie roşii?

a. 6 **b. 12** **c. 15** **d. 3**

(Bacalaureat 2009, varianta 47, III, 1)

42. Se generează prin metoda backtracking mulţimile distincte ale căror elemente sunt numere naturale nenule şi care au proprietatea că suma elementelor fiecărei mulţimi este egală cu **7**. Astfel, sunt generate, în această ordine, mulţimile: **{1,2,4}**, **{1,6}**, **{2,5}**, **{3,4}**, **{7}**. Folosind aceeaşi metodă pentru a genera mulţimile distincte ale căror elemente sunt numere naturale nenule şi care au proprietatea că suma elementelor fiecărei mulţimi este egală cu **9**, stabiliţi în ce ordine sunt generate următoarele mulţimi: **M1={2,3,4}**; **M2={3,6}**; **M3={2,7}**; **M4={4,5}**.

(Bacalaureat 2009, varianta 48, III, 2)

43. Se generează în ordine strict crescătoare numerele de câte şase cifre care conţin: cifra **1** o singură dată, cifra **2** de două ori şi cifra **3** de trei ori. Se obţin, în această ordine, numerele: **122333**, **123233**, **123323**, …, **333221**. Câte numere generate prin această metodă au prima cifră **1** şi ultima cifră **2**?

a. 1 **b. 3** **c. 4** **d. 8**

(Bacalaureat 2009, varianta 49, III, 1)

44. Se generează în ordine strict crescătoare toate numerele de câte şase cifre care conţin: cifra **1** o singură dată, cifra **2** de două ori şi cifra **3** de trei ori. Se obţin, în această ordine, numerele: **122333**, **123233**, **123323**, …, **333221**. Ce număr se află imediat înaintea şi ce număr se află imediat după numărul **332312** în şirul numerelor generate?

(Bacalaureat 2009, varianta 50, III, 2)

45. Utilizând metoda backtracking, se generează în ordine lexicografică toate anagramele cuvântului **caiet** (cuvinte formate din aceleaşi litere, eventual în altă ordine). Câte cuvinte care încep cu litera **t** vor fi generate?

a. 1 **b. 6** **c. 12** **d. 24**

(Bacalaureat 2009, varianta 52, III, 1)

46. Utilizând metoda backtracking se generează în ordine lexicografică toate anagramele cuvântului **caiet** (cuvinte formate din aceleaşi litere, eventual în altă ordine). Care este a **şasea** soluţie?

a. catei **b. actie** **c. actei** **d. catie**

(Bacalaureat 2009, varianta 54, III, 1)

47. Utilizând metoda backtracking se generează toate matricele pătratice de ordinul **4** ale căror elemente aparţin mulţimii **{0,1}**, cu proprietatea că pe fiecare linie şi pe fiecare coloană există **o singură** valoare **1**. Primele **4** soluţii generate sunt, în această ordine:

```
1 0 0 0     1 0 0 0     1 0 0 0     1 0 0 0
0 1 0 0     0 1 0 0     0 0 1 0     0 0 1 0
0 0 1 0     0 0 0 1     0 1 0 0     0 0 0 1
0 0 0 1     0 0 1 0     0 0 0 1     0 1 0 0
```

Care este a **opta** soluţie?

a.
```
0 1 0 0
1 0 0 0
0 0 0 1
0 0 1 0
```
b.
```
0 1 0 0
1 0 0 0
0 0 1 0
0 0 0 1
```
c.
```
0 1 0 0
0 0 1 0
1 0 0 0
0 0 0 1
```
d.
```
0 0 10
1 0 0 0
0 1 0 0
0 0 0 1
```

(Bacalaureat 2009, varianta 55, III, 1)

48. Se utilizează metoda backtracking pentru a genera în ordine lexicografică toate cuvintele de câte patru litere din mulţimea **{d,a,n,s}**, astfel încât în niciun cuvânt să nu existe două litere alăturate identice. Ştiind că primele trei cuvinte generate sunt, în ordine, **adad**, **adan** şi **adas**, care va fi ultimul cuvânt obţinut?

a. snns **b. nsns** **c. snsn** **d. dans**

(Bacalaureat 2009, varianta 57, III, 1)

49. Se utilizează metoda backtracking pentru a genera **în ordine lexicografică** toate cuvintele de câte trei litere distincte din mulţimea **{d,a,n,s}**. Care este cel de-al treilea cuvânt obţinut?

a. ads **b. ans** **c. dan** **d. and**

(Bacalaureat 2009, varianta 58, III, 1)

50. Se utilizează metoda backtracking pentru a genera **în ordine lexicografică** toate cuvintele care conţin toate literele din mulţimea **{a,m,i,c}**, astfel încât fiecare literă să apară exact o dată într-un cuvânt. Câte soluţii sunt generate după cuvântul **amic** şi înainte de cuvântul **cami**?

a. 6 **b. 4** **c. 1** **d. 3**

(Bacalaureat 2009, varianta 59, III, 1)

51. Se utilizează metoda backtracking pentru a genera toate cuvintele care conţin toate literele din mulţimea **{i,n,f,o}**, astfel încât fiecare literă să apară exact o dată într-un cuvânt şi literele **n** şi **o** să nu se afle pe poziţii vecine. Ştiind că primul cuvânt generat este **info**, iar al treilea, al patrulea şi al cincilea sunt **nifo, niof, nfio** care este cel de-al doilea cuvânt obţinut?

a. iofn **b. inof** **c. ionf** **d. niof**

(Bacalaureat 2009, varianta 60, III, 1)

52. Utilizând metoda backtracking pentru afişarea tuturor modalităţilor de descompunere a unui număr natural ca o sumă de numere naturale nenule, pentru **n=3** se obţin, în ordine, soluţiile: **1+1+1**; **1+2**; **2+1**; **3**. Ordinea de scriere a termenilor dintr-o descompunere este semnificativă. Folosind aceeaşi metodă pentru **n=10**, care este soluţia generată imediat după **1+1+3+5**?

a. 1+1+4+1+1+1+1 **b. 1+1+7+1** **c. 1+2+7** **d. 1+1+4+4**

(Bacalaureat 2009, varianta 62, III, 1)

53. Se generează, prin metoda backtracking, toate partiţiile mulţimii **A={1,2,3}** obţinându-se următoarele soluţii: **{1}{2}{3}**; **{1}{2,3}**; **{1,3}{2}**; **{1,2}{3}**; **{1,2,3}**. Se observă că dintre acestea, prima soluţie e alcătuită din exact trei submulţimi. Dacă se foloseşte aceeaşi metodă pentru a genera partiţiile mulţimii **{1,2,3,4}** stabiliţi câte dintre soluţiile generate vor fi alcătuite din exact trei submulţimi.

a. 3 **b. 12** **c. 6** **d. 5**

(Bacalaureat 2009, varianta 63, III, 1)

54. Se generează, prin metoda backtracking, toate modalităţile de aşezare a numerelor naturale de la **1** la **5**, astfel încât oricare **2** numere consecutive să nu se afle pe poziţii alăturate. Dacă primele două soluţii sunt: (**1,3,5,2,4)** şi **(1,4,2,5,3)**, care este prima soluţie generată în care primul număr este **4**?

a. (4, 1, 3, 2, 5) **b. (4,2,5,1, 3)** **c. (4, 3, 5, 3, 1)** **d. (4, 1, 3, 5, 2)**

(Bacalaureat 2009, varianta 64, III, 1)

55. Se generează, prin metoda backtracking, toate modalităţile de aşezare a numerelor naturale de la **1** la **5** astfel încât oricare două numere consecutive să nu se afle pe poziţii alăturate. Dacă primele două soluţii sunt: (**1,3,5,2,4)** şi **(1,4,2,5,3)**, care este prima soluţie generată care începe cu **2**?

a. (2, 4, 1, 3, 5) **b. (2, 5, 4, 3, 1)** **c. (2, 4, 1, 3, 1)** **d. (2, 3, 5, 4, 1)**

(Bacalaureat 2009, varianta 65, III, 1)

56. Se generează în ordine crescătoare, toate numerele naturale de **5** cifre distincte, care se pot forma cu cifrele **2**,**3**,**4**,**5** şi **6.** Să se precizeze numărul generat imediat înaintea şi numărul generat imediat după secvenţa următoare : **34256**, **34265**, **34526**, **34562**

a. 32645 şi **34625** **b. 32654** şi **34655**

c. 32654 şi 34625 **d. 32645 şi 34655**

(Bacalaureat 2009, varianta 66, III, 1)

57. Se generează în ordine crescătoare, toate numerele naturale de **5** cifre distincte, care se pot forma cu cifrele **5**,**6**,**7**,**8** şi **9.** Să se precizeze numărul generat imediat înaintea şi numărul generat imediat după secvenţa următoare: **67589**,**67598**,**67859**,**67895**.

a. 65987 şi **67958** **b. 65978** şi **67988**

c. 65978 şi **67958** **d. 65987** şi **67988**

(Bacalaureat 2009, varianta 67, III, 1)

58. Construim anagramele unui cuvânt $c_1c_2c_3c_4$ prin generarea în ordine lexicografică

a permutărilor indicilor literelor cuvântului şi obţinem $c_1c_2c_3c_4$ $c_1c_2c_4c_3$ $c_1c_3c_2c_4$ ... $c_4c_3c_1c_2$ $c_4c_3c_2c_1$. Pentru anagramele cuvântului **pateu**, după şirul **paetu**, **paeut**, **paute** cuvintele imediat următoare sunt:

a. pauet şi **ptaeu**
b. ptaeu şi **ptaue**
c. pauet şi **ptaue**
d. ptaeu şi **patue**

(Bacalaureat 2009, varianta 69, III, 1)

59. Pentru rezolvarea cărei probleme dintre cele enumerate mai jos se poate utiliza metoda backtracking ?

a. determinarea reuniunii a **3** mulţimi
b. determinarea tuturor divizorilor unui număr din **3** cifre
c. determinarea tuturor elementelor mai mici decât **30000** din şirul lui Fibonacci
d. determinarea tuturor variantelor în care se pot genera steagurile cu 3 culori (din mulţimea: ”roşu”, ”galben”, ”albastru” şi ”alb”), având la mijloc culoarea ”galben”

(Bacalaureat 2009, varianta 70, III, 1)

60. Se generează în ordine crescătoare toate numerele de exact **4** cifre care se pot forma cu elementele mulţimii **{0,1,2,3,4}**. Primele **8** soluţii generate sunt, în ordine: **1000**, **1001**, **1002**, **1003**, **1004**, **1010**, **1011**, **1012**. Care sunt primele trei numere ce se vor genera imediat după numărul **3443**?

a. 4000,4001,4002
b. 3444,4443,4444
c. 3444,4444,4000
d. 3444,4000,4001

(Bacalaureat 2009, varianta 71, III, 1)

61. Se generează în ordine crescătoare toate numerele de **4** cifre, cu cifre distincte, astfel încât diferenţa în valoare absolută dintre prima şi ultima, respectiv a doua şi a treia cifră este egală cu **2**. Primele **11** soluţii generate sunt, în ordine: **1023**, **1203**, **1243**, **1423**, **1463**, **1573**, **1643**, **1683**, **1753**, **1793**, **1863**. Care dintre următoarele numere se va genera imediat înaintea numărului **9317**?

a. 9247 **b. 9357** **c. 9207** **d. 8976**

(Bacalaureat 2009, varianta 72, III, 1)

62. Se generează în ordine crescătoare toate numerele de **4** cifre, cu cifre distincte, astfel încât diferenţa în valoare absolută dintre ultimele două cifre ale fiecărui număr generat este egală cu **2**. Primele opt soluţii generate sunt, în ordine: **1024**, **1035**, **1042**, **1046**, **1053**, **1057**, **1064**, **1068**. Care dintre următoarele numere se va genera imediat după numărul **8975**?

a. 8979 **b. 9013** **c. 8957** **d. 9024**

(Bacalaureat 2009, varianta 73, III, 1)

63. Prin metoda backtracking se generează toate anagramele (cuvintele obţinute prin permutarea literelor) unui cuvânt dat. Ştiind că se aplică această metodă pentru cuvântul **solar**, precizaţi câte cuvinte se vor genera astfel încât prima şi ultima literă din fiecare cuvânt generat să fie **vocală** (sunt considerate vocale caracterele **a**, **e**, **i** , **o**, **u**)?

a. 24 **b. 6** **c. 10** **d. 12**

(Bacalaureat 2009, varianta 74, III, 1)

64. Dacă se utilizează metoda backtracking pentru a genera toate permutările de **4** obiecte şi primele **5** permutări generate sunt, în această ordine, **4 3 2 1**, **4 3 1 2**, **4 2 3 1**, **4 2 1 3**, **4 1 3 2**, atunci a **6**-a permutare este:

a. 3 2 1 4 **b. 3 4 2 1** **c. 1 4 3 2** **d. 4 1 2 3**

(Bacalaureat 2009, varianta 76, III, 1)

65. Folosind cifrele **{1,2,3}** se generează, în ordinea crescătoare a valorii, toate numerele pare formate din trei cifre distincte. Astfel, se obţin în ordine, numerele: **132**, **312**. Folosind aceeaşi metodă, se generează numerele pare formate din patru cifre distincte din mulţimea **{1,2,3,4}**. Care va fi al **4**-lea număr generat?

a. 2134 **b. 1432** **c. 2314** **d. 1423**

(Bacalaureat 2009, varianta 81, III, 1)

66. Folosind cifrele **{2,3,4}** se generează, în ordinea crescătoare a valorii, toate numerele impare formate din trei cifre distincte. Astfel se obţin, în ordine, numerele: **243**, **423**. Folosind aceeaşi metodă, se generează numerele pare formate din patru cifre distincte din mulţimea **{2,3,4,5}.** Care va fi al **5**-lea număr generat?

a. 3452 **b. 3524** **c. 2534** **d. 3542**

(Bacalaureat 2009, varianta 82, III, 1)

67. Folosind cifrele **{1,2,3}** se generează, în ordinea crescătoare a valorii, toate numerele formate din exact trei cifre, în care cifrele alăturate au valori consecutive. Astfel se obţin în ordine, numerele: **121, 123, 212, 232, 321** şi **323**. Folosind aceeaşi metodă se generează numere de patru cifre din mulţimea **{1,2,3,4}** care îndeplinesc aceeaşi condiţie. Care va fi al **5**-lea număr generat?

a. 2121 **b. 2123** **c. 3121** **d. 2323**

(Bacalaureat 2009, varianta 83, III, 1)

68. Folosind cifrele **{3,4,5}** se generează, în ordinea crescătoare a valorii, toate numerele impare formate din trei cifre distincte. Astfel se obţin, în ordine, numerele: **345**, **435**, **453**, **543**. Folosind aceeaşi metodă, se generează numerele impare formate din patru cifre distincte din mulţimea **{2,3,4,5}**. Care va fi al**5**-lea număr generat?

a. 3425 **b. 2534** **c. 4235** **d. 3245**

(Bacalaureat 2009, varianta 84, III, 1)

69. Folosind cifrele **{1,2,3}** se generează, în ordinea crescătoare a valorii, toate numerele impare formate din trei cifre distincte. Astfel se obţin, în ordine, numerele: **123**, **213**, **231**, **321**. Folosind aceeaşi metodă, se generează numerele impare formate din patru cifre distincte din mulţimea **{1,2,3,4}**. Care va fi al **5**-lea număr generat ?

a. 2413 **b. 1423** **c. 2431** **d. 3241**

(Bacalaureat 2009, varianta 85, III, 1)

70. Având la dispoziţie cifrele **0**, **1** şi **2** se pot genera, în ordine crescătoare, numere care au suma cifrelor egală cu **2**. Astfel, primele **6** soluţii sunt **2**, **11**, **20**, **101**, **110**, **200**. Folosind acelaşi algoritm, se generează numere cu cifrele **0**, **1**, **2** şi **3** care au suma cifrelor egală cu **4.** Care va fi al **7**-lea număr din această generare?

a. 130 **b. 301** **c. 220** **d. 103**

(Bacalaureat 2009, varianta 92, III, 1)

71. Un elev realizează un program care citeşte o valoare naturală pentru o variabilă **n** şi apoi afişează în fişierul **permut.txt**, pe prima linie, valoarea lui **n**, apoi toate permutările mulţimii **{1,2,...,n}**, câte o permutare pe câte o linie a fişierului. Rulând programul pentru **n=3**, fişierul va conţine cele **7** linii alăturate.

Dacă va rula din nou programul pentru **n=5**, ce va conţine a **8**-a linie din fişier?

```
3
3 2 1
3 1 2
2 3 1
2 1 3
1 3 2
1 2 3
```

a. 2134 **b. 2143** **c. 3421** **d. 3412**

(Bacalaureat 2009, varianta 94, III, 1)

72. Un program citeşte o valoare naturală nenulă pentru **n** şi apoi generează şi afişează,

în ordine crescătoare lexicografic, toate combinaţiile formate din **n** cifre care aparţin mulţimii **{0,1}**. Astfel, pentru **n=2**, combinaţiile sunt afişate în următoarea ordine: **00**, **01**, **10**, **11**. Dacă se rulează acest program şi se citeşte pentru **n** valoarea **9**, imediat după combinaţia **011011011** va fi afişată combinaţia:

a. 011100100 **b. 011011100** **c. 011011011** **d. 011100000**

(Bacalaureat 2009, varianta 95, III, 1)

73. Un program citeşte o valoare naturală nenulă pentru **n** şi apoi generează şi afişează, în ordine descrescătoare lexicografic, toate combinaţiile de **n** cifre care aparţin mulţimii **{0,1}**. Astfel, pentru **n=2**, combinaţiile sunt afişate în următoarea ordine: **11**, **10**, **01**, **00**. Dacă se rulează acest program şi se citeşte pentru **n** valoarea **8**, imediat după combinaţia **10101000** va fi afişată combinaţia:

a. 01010111 **b. 10100111** **c. 10101001** **d. 10100100**

(Bacalaureat 2009, varianta 96, III, 1)

74. Se generează, utilizând metoda bactracking, cuvintele cu exact **3** litere din mulţimea **{a,x,c,f,g}**. Dacă primele patru cuvinte generate sunt, în ordine, **aaa**, **aax**, **aac**, **aaf**, scrieţi ultimele trei cuvinte care încep cu litera **a**, în ordinea în care vor fi generate.

(Bacalaureat 2009, varianta 97, III, 2)

75. Utilizând metoda backtracking se generează toate submuţimile nevide ale mulţimii **{3,6,2,5}**. Primele şase submulţimi generate sunt, în ordine: **{3}**, **{3,6}**, **{3,6,2}**, **{3,6,2,5}**, **{3,6,5}**, **{3,2}**. Care sunt, în ordinea obţinerii, ultimele trei submulţimi, generate după această regulă?

(Bacalaureat 2009, varianta 98, III, 2)

76. Se utilizează metoda backtracking pentru a genera toate cuvintele formate din două litere distincte din muţimea **{w,x,z,y}** astfel încât niciun cuvânt să nu înceapă cu litera **x** şi niciun cuvânt să nu conţină litera **w** lângă litera **z**. Cuvintele vor fi generate în ordinea **wx**, **wy**, **zx**, **zy**, **yw**, **yx**, **yz**. Folosind aceeaşi metodă se generează toate cuvintele de două litere distincte din mulţimea **{w,x,z,y,t}** astfel încât niciun cuvânt să nu înceapă cu litera **x** şi niciun cuvânt să nu conţină litera **w** lângă litera **z**. Care sunt a treia şi a patra soluţie generată?

(Bacalaureat 2009, varianta 99, III, 2)

77. Aplicând metoda backtracking pentru a genera toate permutările celor **n** elemente ale unei mulţimi, o soluţie se memorează sub forma unui tablou unidimensional $x_1,x_2,\ldots,x_n$. Dacă sunt deja generate valori pentru componentele $x_1,x_2,\ldots,x_{k-1}$, iar pentru componenta curentă, x_k ($1<k<n$), a fost găsită o valoare convenabilă, atunci se încearcă alegerea

a. unei noi valori pentru componenta x_{k-1} **b.** unei valori pentru componenta x_{k+1}

c. unei noi valori pentru componenta x_k **d.** unei noi valori pentru componenta x_1

(Bacalaureat 2009, varianta 100, III, 1)

Probleme propuse

Problema 9.10. ***(Metagrama).*** Să se scrie un program care, citind un cuvânt şi un număr natural cuprins între **1** şi lungimea cuvântului, să afişeze toate anagramările obţinute din cuvânt, după eliminarea literei de pe poziţia citită.

Problema 9.11. ***(Paranteze[4]).*** Se dă numărul natural par **n>0**. Să se determine toate şirurile de **n** paranteze care se închid corect.

Exemplu: **n=6 ((( ))), ( )( )( ) ,(( )( )), ( )(( )), (( ))( ).**

Problema 9.12. Fiind dat un număr natural pozitiv **n**, se cere să se afişeze toate descompunerile sale ca sumă de **p** numere naturale nenule.

Problema 9.13. Fiind dat un număr natural pozitiv **n**, se cere să se afişeze toate descompunerile sale ca sumă de numere prime.

Problema 9.14. Să se genereze toate numerele de **n** cifre (**n≤6**) care nu conțin trei cifre pare sau trei cifre impare alăturate.

Problema 9.15. Să se genereze toate numerele de **n** cifre (**n<6**) care nu conțin trei cifre consecutive pe poziții alăturate și ale căror cifre se află în ordine strict descrescătoare.

Problema 9.16. Fiind dat un șir de **n** numere naturale mai mici decât **30000**, să se genereze toate subșirurile crescătoare de **k** numere care se pot forma începând cu primul element din șir. De exemplu, pentru **n=5**, **k=3** și numerele **4, 8, 5, 7, 6, 3** se vor afișa numerele: **4, 5, 6**; **4, 5, 7**; **4, 5, 8**.

Problema 9.17. Fiind dat un număr natural **n** și un șir de **m** numere naturale, să se afișeze toate modalitățile de scriere a numărului **n** ca sumă de numere din șirul dat.

Problema 9.18. Fiind dat un șir de **n** numere naturale, să se genereze submulțimile de **k** numere din șir în care se găsesc cel puțin trei numere care pot reprezenta laturile unui triunghi.

Problema 9.19. Să se genereze toate numerele de **n** cifre egale cu de **k** ori produsul cifrelor (1**≤k≤9,** 1**≤n≤10000**).

Problema 9.20. Fiind dat un alfabet ce conține **v** vocale și **c** consoane se cere să se genereze toate cuvintele de lungime **n** (**3≤n≤15**) care nu conțin trei vocale sau trei consoane alăturate.

Problema 9.21. ***(Problema turnurilor[5])*** Fiind dată o tablă de şah, de dimensiune **n**x**n**, se cer toate soluţiile de aranjare a **n** turnuri, astfel încât două turnuri oarecare să nu se afle pe aceeaşi linie sau coloană (adică turnurile să nu se atace reciproc).

Problema 9.22. Un grup de **n** (**n<=10**) persoane numerotate de la **1** la **n** sunt aşezate pe un rând de scaune, dar între oricare două persoane vecine s-au ivit conflicte. Scrieţi un program care afişează toate modurile posibile de reaşezare a persoanelor, astfel încât între oricare două persoane aflate în conflict să stea una sau cel mult două persoane. De exemplu, pentru **n=4** programul trebuia să afişeze: **2, 4, 1, 3** şi **3, 1, 4, 2**.

Problema 9.23. ***(Drapele).*** Avem la dispoziţie **6** culori: **alb**, **galben**, **roşu**, **verde**, **albastru**, **negru**. Să se precizeze toate drapelele tricolore care se pot proiecta, ştiind că trebuie respectate următoarele reguli:

- orice drapel are culoarea din mijloc **galben** sau **verde**;
- cele trei culori de pe drapel sunt distincte.

---

[4]Problema a fost propusă la faza finală a Olimpiadei Naţionale de Informatică 1993 la clasa a X-a.

[5]Problema este echivalentă cu generarea tuturor matricelor pătratice binare de ordin n, care au proprietatea că atât pe fiecare linie cât și pe fiecare coloană conțin exact un element egal cu 1.

Problema 9.24. ***(Colorarea hărţilor).*** Fiind dată o hartă cu **n** ţări, se cer toate soluţiile de colorare a hărţii, utilizând cel mult **4** culori[6], astfel încât două ţări cu frontieră comună să fie colorate diferit.

Problema 9.25. ***(Umplerea unei suprafeţe închise[7]).*** Se dă o matrice binară (elementele ei au numai valorile **0** sau **1**). Valorile **1** delimitează o anumită suprafaţă închisă în cadrul matricei (elementele aparţinând acestei suprafeţe sunt marcate cu **0**). Se dau de asemenea coordonatele **x** şi **y** ale unui element al matricei semnificând un punct din interiorul acestei suprafeţe. Se cere schimbarea valorilor **0** din suprafaţa închisă cu o altă valoare (colorarea suprafeţei închise).

Problema 9.26. ***(Problema fotografiei).*** O fotografie alb-negru este prezentată sub forma unei matrice binare. Ea înfăţişează unul sau mai multe obiecte. Porţiunile corespunzătoare obiectului (sau obiectelor) în matrice au valoarea **1**. Se cere să se determine dacă fotografia reprezintă unul sau mai multe obiecte.

Problema 9.27. ***(Problema bilei)*** Fiind dat un teren sub formă de matrice cu m linii și n coloane, unde fiecare element al matricei reprezintă anumite zonele cu diferite altitudini, fiecare dintre acestea fiind date de valoarea reținută în acel element și considerând că în punctul de coordonate (lin, col) se găsește o bilă, aceasta putându-se deplasa în orice porțiune de teren aflata la nord, est, sud sau vest, doar dacă are altitudinea inferioraă punctului în care se află bila. Să se găsească toate posibilitățile ca bila să părăsească terenul.

Problema 9.28. ***(Problema discretă a rucsacului).*** Cu ajutorul unui rucsac de greutate maximă admisibilă **G** trebuie să se transporte o serie de obiecte din **n** disponibile, având greutăţile $g_1, g_2, ..., g_n$, aceste obiecte fiind de utilităţile $c_1, c_2, ..., c_n$. Presupunând că obiectele nu pot fi luate decât în întregime (neputându-se diviza), să se găsească o modalitate de a umple optim rucsacul.

Problema 9.29. **** (Labirint).*** Se dă un labirint sub forma unei matrice binare în care unităţile corespund spaţiilor pe unde se poate trece, iar zerourile corespund zidurilor. Un şoricel pus într-o anumită căsuţă a labirintului va trebui să ajungă într-o altă căsuţă a labirintului, unde se află o bucăţică de caşcaval. El se poate mişca doar ortogonal, nu şi diagonal. Să se determine toate posibilităţile şoricelului de a ajunge la caşcaval, fără a trece de mai multe ori prin acelaşi loc.

Problema 9.30. **** (Problema canibalilor şi misionarilor).*** Pe malul unei ape se găsesc **c** canibali şi **m** misionari. Ei urmează să treacă apa, având la dispoziţie o barcă cu două locuri. Se ştie că, dacă atât pe un mal cât şi pe celălalt, avem mai mulţi canibali decât misionari, misionarii sunt mâncaţi de canibali. Se cere să se scrie un program care să furnizeze toate variantele de trecere a apei, în care misionarii să nu fie mâncaţi.
De exemplu, pentru c=3 şi m=3 avem:

Malul stâng		**Malul drept**	
Canibali	Misionari	Canibali	Misionari
3	**3**	**0**	**0**

[6] Faptul că sunt suficiente numai **4** culori pentru ca orice hartă să poată fi colorată a fost demonstrat de W. Haken şi K. Appel în 1976 cu ajutorul uinui algoritm implementat pe calculator ce a lucrat aproape 1200 ore, iar în 1977 F. Allaise, folosind o altă procedură, a reuşit să obţină demonstraţia teoremei în numai 50 de ore de dialog om-calculator.

[7] Algoritmii pentru rezolvarea acestei probleme mai poartă numele de *algoritmi **FILL***.

1	3	2	0
2	3	1	0
0	3	3	0
1	3	2	0
1	1	2	2
2	2	1	1
2	0	1	3
3	0	0	3
1	0	2	3
2	0	1	3
0	0	3	3

Problema 9.31. * ***(Puncte albe şi negre).*** Se dau **n** puncte **albe** şi **n** puncte **negre** în plan, de coordonate întregi. Fiecare punct **alb** se uneşte cu câte un punct **negru**, astfel încât din fiecare punct, fie el **alb** sau **negru**, pleacă exact un segment. Să se determine o astfel de configuraţie de segmente aşa încât oricare două segmente să nu se intersecteze. Se citesc **n** perechi de coordonate corespunzând punctelor **albe** şi **n** perechi de coordonate corespunzând punctelor **negre**.

Problema 9.32. * ***(Attila şi regele).*** Un cal şi un rege se află pe o tablă de şah. Unele câmpuri sunt "arse", poziţiile lor fiind cunoscute. Calul nu poate călca pe câmpuri "arse", iar orice mişcare a calului face ca respectivul câmp să devină "ars". Să se afle dacă există o succesiune de mutări permise (cu restricţiile de mai sus) prin care calul să poată ajunge la rege şi să revină la poziţia iniţială. Poziţia iniţială a calului precum şi poziţia regelui sunt considerate "nearse".

Problema 9.33. * ***(Problema căsătoriilor stabile).*** Se consideră **n** fete care urmează să se căsătorească cu **n** băieţi. Fetele şi băieţii îşi exprimă preferinţele unul faţă de altul prin numerele reale din intervalul **[0,1]**. Preferinţa fetei **i** pentru băiatul **j** este dată de **fb[i,j]**, iar preferinţa băiatului **i** pentru fata **j** este dată de **bf[i,j]**; elementele matricilor **fb** şi **bf** sunt numere reale din intervalul **[0,1]**. Băiatul ales de fata **i** are numărul **x[i]**. Bineînţeles, vectorul **x** va fi un vector permutare. Costul căsătoriei fetei **i** cu băiatul **x[i]** este **fb[i,x[i]]*bf[x[i],i]**, iar costul general (care trebuie minimizat) este suma după **i** a acestor valori. Mai mult, se cere ca cele **n** căsătorii să fie stabile, adică să nu existe **(i,j)** cu **i≠j** astfel încât fata **i** să prefere băiatul **x[j]** băiatului **x[i]**, iar băiatul **x[j]** să prefere fata **i** fetei **j**.

Problema 9.34. * Hercule trebuie să străbată un labirint cu capcane reprezentat de o matrice **n**x**n**. Pentru fiecare celulă a labirintului, se cunoaşte timpul în minute după care celula respectivă devine capcană. După ce o celulă devine capcană, Hercule moare dacă intră în acea celulă. Hercule porneşte din colţul stânga-sus al labirintului şi trebuie să ajungă în colţul dreapta jos. El are nevoie de un minut ca să treacă dintr-o celulă intr-una vecină şi se poate deplasa în sus, în jos, spre stanga sau spre dreapta. Să se afişeze timpul minim în care poate Hercule să străbată labirintul, numărul de drumuri de timp minim, precum şi toate drumurile minime pe care le poate urma Hercule prin labirint de la intrare la ieşire, astfel încât Hercule să nu moară. Drumurile vor fi afişate ca matrici în care sunt indicaţi paşii lui Hercule.

Răspunsuri la teste

1) b; **2)** d; **3)** a; 4) c; **5)** d; **6)** a; **7)** 12346, 12356, 12456, 13456, 23456; **8)** c; **9)** 129, 147, 345; **10)** 56789; **11)** b; **12)** 2+2+2+3, 2+2+5, 2+2+7; **13)** 18; 1**4)** a; **15)** a; **16)**

(7,7,7,7,3), (7,7,7,7,5), (7,7,7,7,7); **17)** 12347, 12346, 12345; **18)** 11101, 11110, 11111; **19)** 10349, 10352, 10354; **20)** 35789, 35679, 35678; **21)** a; **22)** a; **23)** c; **24)** b; **25)** c; **26)** c; **27)** a; **28)** c; **29)** c; **30)** b; **31)** a; **32)** 2+3+7, 2+4+6, 2+10; **33)** c; **34)** d; **35)** b; **36)** c; **37)** d; **38)** c; **39)** a; **40)** c); **41)** a; **42)** M1,M3,M2,M4; **43)** c; **44)** 332231 și 332321; **45)** d; **46)** b; **47)** a (Este de fapt problema turnurilor (9.21) pentru o tablă de șah de 4x4); **48)** a; **49)** c; **50)** c; **51)** a; **52)** a; **53)** c; **54)** d; **55)** a; **56)** c; **57)** a; **58)** a; **59)** d; **60)** d; **61)** a; **62)** a; **63)** b; **64)** d; **65)** b; **66)** b; **67)** b; **68)** d; **69)** a; **70)** a; **71)** c; **72)** b; **73)** b; **74)** agc,agf,agg; **75)** {2}, {2,5},{5}; **76)** wt și zy; **77)** b.

Indicații pentru unele probleme propuse

9.10) Fie lungimea cuvântului **n**. Se vor genera permutările[8] mulțimii **{1**, **2**, …, **n-1}** iar soluțiile vor fi afișate ținând cont că valorile **1**, **2**, …, **n-1** sunt indicii vectorului care reține cuvântul după eliminarea literei.

9.11) Avem câte n/2 paranteze de fiecare fel. Codificăm paranteza deschisă cu 0 și cu 1 pe cea închisă. Condițiile de continuare sunt ca numarul de paranteze deschise/închise să nu fie mai mare ca n/2 si la fiecare pas numărul celor închise să fie mai mic sau egal cu numărul celor deschise.

9.12) E asemănătoare cu problema pentru generarea partiţiilor unui număr natural, doar că sunt soluții rezultat doar cele care au exact p numere.

9.13) Mai intâi determinăm mulțimea numerelor prime mai mici decât n și apoi generăm toate submulțimile acestei mulțimi care au suma n.

9.14) Generăm toate permutările mulțimii {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} fără cifra 0 pe prima poziție și care nu conțin trei cifre pare sau trei cifre impare alăturate.

9.15) Generăm toate permutările mulțimii {9, 8, 7, 6, 5, 4, 3, 2, 1, 0} care nu conțin trei cifre consecutive pe poziții alăturate.

9.16) Asemănătoare cu problema "Subșir maxim" doar că aici lungimea e k.

9.17) Generăm toate submulțimile mulțimii celor m numere care au suma n.

9.18) Sunt generate aranjamente de n luate de câte k și sunt soluții doar cele care conțin cel puțin 3 numere ce pot fi laturi ale unui triunghi.

9.19) Generăm toate permutările mulțimii {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} fără cifra 0 pe prima poziție care verifică proprietatea din enunț.

9.20) Se generează permutările celor c+v litere care verifică proprietatea din enunț.

9.21) Asemănătoare cu problema damelor, doar că turnurile nu se atacă și pe diagonală.

9.22) Se generează permutările ținând cont de restricții.

9.23) Se generează permutările celor 6 culori ținând cont de restricții.

9.24) Asemănătoare cu problema colorării hărților cu 3 culori.

9.25) FILL

9.29) Asemănătoare cu problema sărituri calului, însă ținînd cont de mișcările pe care le poate face șoricelul și că trebuie să ajungă la caşcaval.

9.30) Codificăm traversările astfel:

1 trec doi canibali; **2** trec doi misionari; **3** trec un canibal şi un misionar; **4** trece un canibal; **5** trece un misionar. Pentru rezolvare recursivă se va folosi o stiva multiplă, pe fiecare nivel reţinem:

- traversarea care urmează a se face;
- numărul de canibali de pe malul stâng;

---

[8] Problemele permutărilor, aranjamentelor, cobinărilor și altele le găsiți rezolvate în capitolul următor.

- numărul de misionari de pe malul stâng;
- numărul de canibali de pe malul drept;
- numărul de misionari de pe malul drept.

9.32) Să notăm cu **t** tabla de şah şi cu **st** stiva (st[k][0], st[k][1] reprezintă coordonatele calului la mutarea cu numărul k, respectiv linia şi coloana). Condiţiile de continuare sunt date de menţinerea calului pe tabla de şah, precum şi plasarea acestuia pe câmpuri nearse. Soluţia este gasită în momentul în care calul ajunge în punctul de unde a plecat, de coordonate xc, yc, iar coordonatele regelui (xr, yr) se regăsesc în stivă. Când se ajunge la o soluţie, se afişează conţinutul stivei unde este memorat traseul calului şi se opreşte execuţia programului.

Bibliografie

1. *Tehnici de programare,* Octavian Aspru, Editura Adias, 1997, Rm. Vâlcea;
2. *Fundamentele programării – culegere de probleme pentru clasele IX-XI*, Dana Lica, Radu Boriga, Doru Popescu Anastasiu, Dan Pracsiu, Mariana Ciobanu, Radu Vișinescu, Mihaela Stan, Editura L&S Soft, București;
3. *Informatică – Manual pentru clasa a XI-a, varianta Pascal şi C++,* Tudor Sorin, Vlad Huţanu, Editura L&S Infomat, 2006, Bucureşti;
4. *Bac 2009: subiecte posibile*, Vlad Giorgie Daniel, Marcu Daniela ș. a., Editura CYGNUS, 2009, Suceava;
5. *Colecția revistei Gazeta de informatică.*