Sorin Ifrim

ALGORITMI RECURSIVI

99 de probleme
rezolvate și explicate

Editura EUROCART
Iași 2002

ISBN 973 – 99340 – 6 - 4

PREFAȚĂ

Recursivitatea constituie o manieră interesantă de abordare a problemelor, deoarece trebuie puse în balanță avantajele și dezavantajele utilizării ei. Prin lucrarea de față am dorit să determin cititorii să îmbrățișeze această modalitate de rezolvare a problemelor, subliniind cu precădere avantajele ei.

Toate problemele rezolvate în această lucrare conțin, pe lângă programele Pascal, explicații amănunțite ale modului în care *"lucrează"* algoritmii respectivi, semnificația parametrilor și a variabilelor.

Capitolul I conține doar algoritmi meniți să familiarizeze cititorul cu tehnica recursivității, să-l determine să creadă în utilizarea recursivității prin descoperirea stilului direct și pragmatic de abordare a problemelor, **algoritmi *"didactici"***, fără aplicabilitate practică, nerecomandați,.

Capitolul al II-lea se dorește o pledoarie pentru algoritmii recursivi chemați să soluționeze o serie de probleme care suportă o abordare **"Divide et Impera"**. Majoritatea algoritmilor prezentați aici constituie rezolvări cu o lizibilitate deosebit de ridicată în comparație cu algoritmii iterativi corespunzători.

Capitolul al III-lea vrea să arunce o mănușă schemelor complicate și stufoase utilizate pe scară largă în prezentarea metodei **Backtracking**, care fac imposibilă descoperirea problemei a cărei rezolvare o constituie respectivul program, fără o atentă și sisifică muncă. Abordarea recursivă reduce considerabil dimensiunea programelor și le conferă acestora o oarecare *"personalitate"*.

În finalul acestui capitol sunt prezentate problemele de backtracking (cele mai bine punctate subiecte!) conținute de cele cinci variante de subiecte de la examenul de bacalaureat, sesiunea iunie-iulie 2001.

Se cuvine a menționa faptul că prezenta lucrare abordează doar aceste metode de programare deoarece ele sunt singurele prezente în toate programele cursurilor de specialitate ale liceelor și facultăților de profil.

Sunt interesat de propunerile, observațiile, comentariile și sugestiile cititorilor pe care le aștept pe adresa

sorinif@yahoo.com

Autorul

I. RECURSIVITATE

Un subprogram se numeşte recursiv dacă acesta conţine un apel către el însuşi. Un algoritm se numeşte recursiv dacă acesta conţine cel puţin un subprogram recursiv.

Recursivitatea este de două feluri:

- **directă** – în care caz există un subprogram ce conţine un apel direct către el însuşi;
- **mutuală** – în care caz avem de a face cu două subprograme ce se apelează reciproc.

Toţi algoritmii prezentaţi în această carte se înscriu în categoria recursivităţii directe.

La abordarea recursivă a unei probleme, aceasta se reduce, din aproape în aproape, la probleme de acelaşi tip, dar de un ordin de mărime mai mic, până când se ajunge la o problemă banală.

Apelul recursiv trebuie să se realizeze în mod obligatoriu condiţionat, în caz contrar având de a face cu un apel recursiv infinit (buclă infinită). Condiţia de realizare (sau nerealizare) a apelului recursiv va fi numită condiţie de adâncime şi pentru scrierea ei trebuie stabilit (dedus) momentul la care procesul respectiv poate fi descris simplu (altfel decât prin el însuşi).

Pentru a putea decide când şi dacă trebuie să recurgem la o abordare recursivă în detrimentul uneia iterative (nerecursive), trebuie să ştim care sunt avantajele şi dezavantajele utilizării recursivităţii.

AVANTAJE:

- dimensiune redusă (în comparaţie cu algoritmii iterativi corespunzători);
- algoritmii sunt clari, lizibili;
- odată stăpânită, recursivitatea conduce la diminuarea greşelilor iminente ce apar în rezolvarea unor probleme cu grad de dificultate mai mare decât "*banal*".

DEZAVANTAJE:

- viteză de execuţie mică;
- consum mare de memorie.

P1. n!

Fiind dat un număr natural n, să se calculeze și să se afișeze valoarea lui n!

Rezolvare:

n! se poate scrie:

$$n! = (1 \cdot 2 \cdot \ldots \cdot (n-1)) \cdot n = (n-1)!n$$

sau, ținând cont de faptul că 1!=1, avem:

$$n! = \begin{cases} n \cdot (n-1)! & , n > 1 \\ 1 & , n = 1 \end{cases}$$

Programul Pascal:

```
program factorial;
uses
   crt;
var
   n:byte;

function fact(x:byte):longint;
begin
   if x<=1 then
      fact:=1
   else
      fact:=fact(x-1)*x;
end;

begin
   clrscr;
   write('n=');
   readln(n);
   writeln(n,'!=',fact(n));
   readln;
end.
```

P2. Fibonacci

Fie n un număr natural. Să se determine și să se afișeze al n-lea termen din șirul lui Fibonacci.

Rezolvare:

Metoda este directă și constă doar în introducerea relației de definiție a șirului:

$$F_x = \begin{cases} F_{x-1} + F_{x-2} & , x > 2 \\ 1 & , x \le 2 \end{cases}$$

într-o funcție recursivă.

Programul Pascal:

```
program Fibonacci;
uses
   crt;
var
   n:byte;

function fibo(x:byte):longint;
begin
   if x<=2 then
      fibo:=1
   else
      fibo:=fibo(x-1)+fibo(x-2);
end;

begin
   clrscr;
   write('n=');
   readln(n);
   writeln('F(',n,')=',fibo(n));
   readln;
end.
```

P3. x^n

Să se calculeze și să se afișeze x^n, x și n fiind numere naturale date.

Rezolvare:

Ținând cont de faptul că x^n poate fi scris:

$$x^n = \begin{cases} x \cdot x^n & , n > 0 \\ 1 & , n = 0 \end{cases}$$

soluția este directă.

Programul Pascal:

```
program x_la_n;
uses
   crt;
var
   x,n:byte;
```

```
function putere(b,e:byte):longint;
begin
   if e=0 then
     putere:=1
   else
     putere:=b*putere(b,e-1);
end;

begin
   clrscr;
   write('x=');
   readln(x);
   write('n=');
   readln(n);
   writeln(x,'^',n,'=',putere(x,n));
   readln;
end.
```

P4. x^n (variantă)

Rezolvare:

Soluția este asemănătoare celei precedente, cu deosebirea că, de această dată, se observă că x^n poate fi scris:

$$x^n = \begin{cases} 1 & , n = 0 \\ x^{\frac{n}{2}} \cdot x^{\frac{n}{2}} & , n \text{ par} \\ x \cdot x^{n-1} & , n \text{ impar} \end{cases}$$

Spre deosebire de metoda precedentă, aceasta este mai rapidă.

Programul Pascal:

```
program x_la_n;
uses
   crt;
var
   x,n:byte;

function putere(b,e:byte):longint;
begin
   if e=0 then
     putere:=1
   else
     if e mod 2=0 then
       putere:=putere(b,e div 2)*putere(b,e div 2)
     else
       putere:=b*putere(b,e-1);
end;

begin
   clrscr;
   write('x=');
   readln(x);
   write('n=');
   readln(n);
   writeln(x,'^',n,'=',putere(x,n));
   readln;
end.
```

P5. Algoritmul lui Euclid

Să se aducă la forma ireductibilă fracția x/y.

Rezolvare:

Pentru a aduce fracția x/y (y≠0) la forma ireductibilă trebuie să împărțim atât numărătorul cât și numitorul prin cel mai mare divizor comun al celor două numere.

Calculul celui mai mare divizor comun se va face utilizând algoritmul lui Euclid care presupune împărțirea unui număr la celălalt, apoi a fostului împărțitor la restul obținut, și așa mai departe, până la obținerea restului 0. În acel moment algoritmul ia sfârșit, iar cel mai mare divizor comun este ultimul împărțitor (cel pentru care s-a obținut rest 0).

În abordarea recursivă se observă că de fapt, dacă inițial se încearcă să se calculeze cel mai mare divizor comun a două numere (x și y), dacă restul nu este 0 (`x mod y<>0`), la pasul următor se va calcula cel mai mare divizor comun dintre fostul împărțitor și restul obținut (y și `x mod y`).

Programul Pascal:

```
program fractie;
uses
   crt;
var
   x,y:word;
```

```
function cmmdc(x,y:word):word;
begin
  if x mod y=0 then
    cmmdc:=y
   else
    cmmdc:=cmmdc(y,x mod y);
end;

begin
  clrscr;
  write('x=');
  readln(x);
  write('y=');
  readln(y);
  if y<>0 then
    writeln(x,'/',y,'=',x div cmmdc(x,y),'/',
      y div cmmdc(x,y))
  else
    writeln('Eroare!!!');
  readln;
end.
```

P6. Calculul combinărilor.

Să se calculeze numărul combinărilor de n obiecte luate câte k.

$$C_n^k = \frac{n!}{k!(n-k)!} = \frac{n!(n-k+1)}{(k-1)!k(n-k+1)!} = \frac{(n-k+1)}{k} \cdot \frac{n!}{(k-1)!(n-(k-1))!}$$

deci $C_n^k = \frac{(n-k+1)}{k} \cdot C_n^{k-1}$

Rezolvare:

Pornind de la această relație și ținând cont de relația

$$C_n^0 = 1$$

funcția recursivă devine banală.

Observație: Atenție la modul în care se implementează împărțirea la k în relația de mai sus!

Programul Pascal:

```
program combinari;
uses
  crt;
var
  n,k:byte;

function comb(n,k:byte):longint;
  begin
    if k=0 then
      comb:=1
    else
      comb:=(n-k+1)*comb(n,k-1) div k;
  end;

begin
  clrscr;
  write('n=');
  readln(n);
  write('k=');
  readln(k);
  writeln('comb(',n,',',k,')=',comb(n,k));
  readln;
end.
```

P7. Funcția lui Ackermann

Să se calculeze valoarea funcției lui Ackermann ack: N×N→N:

$$ack(m,n) = \begin{cases} n+1 & ,m=0 \\ ack(m-1,1) & ,n=0 \\ ack(m-1,ack(m,n-1)) & .m\neq 0, n\neq 0 \end{cases}$$

Rezolvare:

Metoda este directă și constă în introducerea relației de definiție într-o funcție recursivă.

Programul Pascal:

```
program Ackermann;
uses
  crt;
var
  m,n:byte;

function ack(x,y:byte):longint;
```

```
begin
   if x=0 then
      ack:=y+1
   else
      if y=0 then
         ack:=ack(x-1,1)
      else
         ack:=ack(x-1,ack(x,y-1));
end;

egin
 clrscr;
 write('m=');
 readln(m);
 write('n=');
 readln(n);
 writeln('Ack(',m,',',n,')=',ack(m,n));
 readln;
id.
```

B. Polinoame Hermite

se calculeze valorile polinoamelor Hermite $H_n(x)$, știind că:

$H_0(x)=1$

$H_1(x)=2x$

$H_n(x)=2xH_{n-1}(x)-2(n-1)H_{n-2}(x)$ pentru n>1.

zolvare:

introduce relația de definiție într-o funcție recursivă.

ogramul Pascal:

```
ogram Hermite;
es
 crt;
r
n,i,x:byte;

ıction her(k,x:byte):longint;
begin
   if k=0 then
     her:=1
   else
     if k=1 then
       her:=2*x
```



```
       else
         her:=2*x*her(k-1,x)-2*(k-1)*her(k-2,x);
   end;

begin
   clrscr;
   write('n=');
   readln(n);
   write('x=');
   readln(x);
   writeln('Valorile sunt:');
   for i:=1 to n do
     writeln(her(i,x));
   readln;
end.
```

P9. Funcția Manna:

Fie funcția:

$$f(x)=\begin{cases}2x+b & ,x\geq b\\ f(f(x+5)) & ,x<b\end{cases}$$

Să se calculeze f(x), pentru x=1,n.

Rezolvare:

Funcția recursivă (mn) se apelează în cadrul programului principal în cadrul unui for de la 1 la n, primind ca parametru (x) valoarea contorului (i). Valoarea b este memorată într-o variabilă globală.

Funcția recursivă este directă, implementare a relației de definiție a funcției. Valorile returnate de către funcție se afișează direct.

Programul Pascal:

```
program manna;
uses crt;
var
  n,i,b:word;

function mn(x:word):word;
begin
   if x>=b then
     mn:=2*x+b
   else
     mn:=mn(mn(x+5));
end;
```

```
begin
   clrscr;
   write('n=');
   readln(n);
   write('b=');
   readln(b);
   for i:=1 to n do
     begin
       writeln(mn(i));
       if i mod 24=0 then
         begin
           write('Apasati <ENTER>
             pentru a continua!')
           readln;
         end;
     end;
  readln;
end.
```

P10. Suma puterilor rădăcinilor

Fie ecuația x^2-Sx+P=0 și x_1, x_2 rădăcinile ecuației. Să se calculeze sumele puterilor rădăcinilor $S_i=x_1^k+x_2^k$ pentru k∈ [0,n].

Rezolvare:

Se cunoaște faptul că
$S_k=x_1^k+x_2^k=S(x_1^{k-1}+x_2^{k-1}) - P(x_2^{k-2}+x_2^{k-2})=S*S_{k-1}-P*S_{k-2}$
Rezultă relația

$$S_k = \begin{cases} 2 & ,k=0 \\ S & ,k=1 \\ S*S_{k-1}-P*S_{k-2} & ,k>1 \end{cases}$$

Implementarea funcției recursive este directă. Apelul funcției s-a realizat într-un for de la 0 la n pentru afișarea sumelor puterilor din acest interval.

Programul Pascal:

```
program sum_p_rad;
uses
  crt;
var
  s,p:shortint;
  n,i:byte;
```



```
function suma(k:byte):longint;
begin
   if k=0 then
      suma:=2
    else
      if k=1 then
        suma:=s
       else
        suma:=s*suma(k-1)+p*suma(k-2);
end;

begin
   clrscr;
    write('S=');
    readln(s);
    write('P=');
    readln(p);
    write('n=');
    readln(n);
    for i:=0 to n do
      writeln('S',i,'=',suma(i));
    readln;
end.
```

P11. Numărul cifrelor

Fie n un număr natural dat. Să se determine și să se afișeze numărul de cifre din care este compus acest număr.

Rezolvare:

Se utilizează o funcție (nrcif) care primește ca parametru numărul (x). Dacă numărul nu este 0, atunci numărul de cifre este 1 (ultima cifră) plus numărul de cifre ale numărului obținut prin eliminarea ultimei cifre (`x div 10`). Dacă toate cifrele numărului s-au eliminat (x=0), atunci valoarea returnată de funcție este 0 (se inițializează suma).

Programul Pascal:

```
program numar_cifre;
uses
   crt;
var
   n:longint;
```

```
unction nrcif(x:longint):byte;
egin
  if x=0 then
    nrcif:=0
  else
    nrcif:=1+nrcif(x div 10);
nd;

egin
  clrscr;
  write('n=');
  readln(n);
  writeln(n,' are ',nrcif(n),' cifre');
  readln;
nd.
```

12. Suma cifrelor

e n un număr natural dat. Să se determine suma numerelor reprezentate de rele sale.

ezolvare:

S-a utilizat o funcție recursivă (sumcif) ce primește ca parametrul numărul) a cărui sumă a cifrelor o dorim calculată. Dacă numărul este zero, suma relor sale este zero, valoare returnată de funcție (se inițializează suma).

În caz contrar, suma cifrelor se calculează ca fiind suma dintre ultima cifră a mărului (x mod 10) și suma cifrelor numărului obținut prin eliminarea imei cifre (sumcif(x div 10)).

ogramul Pascal:

```
ogram suma_cifre;
es
 crt;
r
 n:longint;

nction sumcif(x:longint):byte;
gin
if x=0 then
   sumcif:=0
else
   sumcif:=x mod 10 + sumcif(x div 10);
d;
```



```
begin
   clrscr;
   write('n=');
   readln(n);
   writeln('Suma cifrelor lui ',n,' este ',sumcif(n));
   readln;
end.
```

P13. Suma elementelor unui şir

Fie a un șir cu n elemente întregi. Să se determine suma elementelor șirului.

Rezolvare:

S-a utilizat o funcție (suma) ce primește ca parametri șirul (s) și lungimea sa (l). Se verifică mai întâi dacă lungimea șirului este mai mare decât zero (l>0). Dacă această condiție este adevărată atunci se calculează suma elementelor ca fiind suma dintre ultimul element al șirului (s[l]) și suma elementelor din restul șirului (suma(s,l-1)). În cazul în care condiția este falsă, funcția va returna valoarea 0 (se inițializează suma).

Programul Pascal:

```
program suma_sir;
uses
  crt;
type
  sir=array [1..100]of integer;
var
  a:sir;
  i,n:byte;

function suma(s:sir;l:byte):longint;
  begin
    if l>0 then
       suma:=s[l]+suma(s,l-1)
    else
       suma:=0;
  end;

begin
  clrscr;
  write('n=');
  readln(n);
  for i:=1 to n do
```

```
   n,c:byte;
   a:sir;
   v:integer;

function apar(s:sir;x:integer;i:byte):byte;
begin
   if i>0 then
      if s[i]=x then
         apar:=1+apar(s,x,i-1)
      else
         apar:=apar(s,x,i-1)
   else
      apar:=0;
end;

begin
   clrscr;
   write('n=');
   readln(n);
   for c:=1 to n do
      begin
         write('a[',c,']=');
         readln(a[c]);
      end;
   write('valoarea cautata=');
   readln(v);
   writeln(c,' se gaseste in sir de ',apar(a,v,n),'
ori');
   readln;
end.
```

P16. Element egal cu poziția sa

Fie a un șir de n numere naturale. Să se determine dacă există sau nu cel puțin o valoare x astfel încât a[x]=x.

Rezolvare:

S-a utilizat o funcție booleană (pegval) ce are ca parametri șirul (s) și lungimea sa (l). Dacă lungimea șirului este strict pozitivă, se cercetează dacă ultimul element din șir este egal cu poziția sa (`s[l]=l`).

În caz afirmativ funcția va returna valoarea true iar în caz contrar se cercetează restul șirului (`pegval(s,l-1)`). Dacă șirul s-a terminat (`l=0`)

înseamnă că s-au întâlnit numai elemente diferite de pozițiile lor, deci funcția va returna valoarea de false.

Programul Pascal:

```
program val_eg_poz;
uses
   crt;
type
   sir=array[1..50]of byte;
var
   a:sir;
   n,i:byte;

function pegval(s:sir;l:byte):boolean ;
   begin
      if l>0 then
         if s[l]=l then
            pegval:=true
         else
            pegval:=pegval(s,l-1)
      else
         pegval:=false;
   end;

begin
   clrscr;
   write('n=');
   readln(n);
   for i:=1 to n do
      begin
         write('a[',i,']=');
         readln(a[i]);
      end;
   if pegval(a,n) then
      writeln('Exista cel putin un element egal cu
pozitia sa.')
   else
      writeln('Nu exista nici un element egal cu
pozitia sa.');
   readln;
end.
```

P17. Produs maxim

Să se determine produsul maxim a doi termeni consecutivi ai unui șir.

Rezolvare:

Programul utilizează o funcție recursivă (pmax) care primește ca parametri șirul (s) și lungimea sa (l). Se verifică mai întâi dacă șirul are mai mult de două elemente (l>2), caz în care produsul maxim va fi cel mai mare număr dintre produsul ultimilor doi termeni (s[l]*s[l-1]) și produsul maxim din restul șirului (pmax(s,l-1)). În cazul în care mai sunt doar doi termeni în șir, produsul maxim este chiar produsul celor doi termeni.

Programul Pascal:

```
program prodmax;
uses
  crt;
type
  sir=array[1..20] of integer;
var
  a:sir;
  n,i:byte;
function pmax(s:sir;l:byte):integer;
begin
  if l>2 then
    if s[l]*s[l-1]>pmax(s,l-1) then
      pmax:=s[l]*s[l-1]
    else
      pmax:=pmax(s,l-1)
  else
    pmax:=s[l]*s[l-1];
end;

begin
 clrscr;
 write('n=');
 readln(n);
 for i:=1 to n do
   begin
     write('a[',i,']=');
     readln(a[i]);
   end;
 writeln('produsul maxim: ',pmax(a,n));
 readln;
 end.
```


P18. Subșir terminal descrescător

Să se determine lungimea celui mai lung subșir terminal ordonat (strict) descrescător dintr-un șir de numere naturale.

Rezolvare:

S-a utilizat o funcție recursivă (coada) care primește ca parametri șirul (b) și lungimea sa (l). Funcția verifică dacă mai există un element înaintea ultimului element (l>1) și dacă ultimul element este mai mic decât elementul precedent (b[l]<b[l-1]). În caz afirmativ se adaugă 1 la valoarea căutată și se continuă căutarea pe un subșir mai mic cu o unitate (coada(b,l-1)). În caz contrar se inițializează valoarea căutată cu 1 (ultimul element formează un subșir terminal strict descrescător.

Programul Pascal:

```
program subsir_terminal;
uses
  crt;
type
  sir=array[1..100]of byte;
var
  a:sir;
  i,n:byte;

function coada(b:sir;l:byte):byte;
begin
  if (l>1) and (b[l]<b[l-1]) then
    coada:=coada(b,l-1)+1
  else
    coada:=1;
end;

begin
  clrscr;
  write('n=');
  readln(n);
  for i:=1 to n do
    begin
      write('a[',i,']=');
      readln(a[i]);
    end;
  writeln('Subsirul cautat are ',
```

```
    begin
      write('a[',i,']=');
      readln(a[i]);
    end;
  writeln('Suma este ',suma(a,n));
  readln;
end.
```

P14. Şirul conţine elemente pozitive

Fie a un şir cu n numere reale. Să se determine dacă acesta conţine sau nu cel puţin un element pozitiv.

Rezolvare:

S-a utilizat o funcţie booleană (pozit) care primeşte ca parametri şirul (vect) şi lungimea sa (i). Dacă şirul mai are elemente (i>0), se verifică dacă ultimul element este pozitiv (vect[i]>0).

În caz afirmativ funcţia returnează valoarea true, iar în caz contrar, se precizează că şirul conţine sau nu elemente pozitive după cum restul şirului conţine sau nu elemente pozitive (pozit:=pozit(vect,i-1)). Dacă şirul s-a terminat (i=0), înseamnă că s-au întâlnit numai valori negative şi funcţia returnează valoarea false.

Programul Pascal:

```
program pozitiv;
uses
  crt;
type
  sir=array[1..20] of real;
var
  a:sir;
  n,c:integer;

function pozit(vect:sir;i:byte):boolean;
begin
  if i>0 then
    if vect[i]>0 then
      pozit:=true
    else
      pozit:=pozit(vect,i-1)
  else
    pozit:=false;
end;
```



```
begin
  clrscr;
  write('n=');
  readln(n);
  for c:=1 to n do
    begin
      write('a[',c,']=');
      readln(a[c]);
    end;
  if pozit(a,n) then
    write('Sirul contine cel putin o
      valoare pozitiva.')
  else
    write('Sirul nu contine nici o
      valoare pozitiva.');
  readln;
end.
```

P15. Apariţii ale unei valori într-un şir

Fie a un şir cu n elemente întregi, şi v o valoare întreagă. Să se determine numărul de apariţii ale valorii v în şir.

Rezolvare:

Pentru rezolvare s-a utilizat o funcţie recursivă (apar) ce are ca parametri şirul (s), valoarea căutată (x) şi un indice care indică poziţia pe care se caută valoarea (i – iniţial n). Dacă nu s-a terminat şirul (i>0) atunci se analizează dacă pe poziţia i se află valoarea căutată (s[i]=x), caz în care numărul de apariţii este 1 (valoarea găsită pe poziţia i) plus numărul de apariţii din restul şirului (apar(s,x,i-1)), iar în caz contrar numărul de apariţii este egal cu numărul de apariţii din restul şirului.

În cazul în care s-a terminat şirul (i=0), numărul de apariţii este 0 (practic se iniţializează suma). În concluzie, se determină numărul de apariţii în subşiruri din ce în ce mai mici, de la întreg şirul, până la şirul vid.

Programul Pascal:

```
program aparitii;
uses
  crt;
type
  sir=array[1..20] of integer;
var
```

```
    coada(a,n),' elemente.');
  readln;
end.
```

P19. Maximul dintr-un şir.

Fie a un şir de numere naturale cu n elemente. Să se determine elementul maxim din şir.

Rezolvare:

Se utilizează o funcţie ce primeşte ca parametru şirul şi lungimea sa. Elementul maxim dintr-un şir a cărui lungime este mai mare decât 1 este cel mai mare element dintre ultimul element (x[l]) şi maximul din restul şirului (max(x,l-1)). Într-un şir cu un singur element (dacă l=1) acel element (x[1]) este maxim(!).

Programul Pascal:

```
program element_maxim;
uses
  crt;
type
  sir=array[1..10] of byte;
var
  a:sir;
  i,n:byte;

function max(x:sir;l:byte):byte;
begin
  if l>1 then
    if x[l]>=max(x,l-1) then
      max:=x[l]
    else
      max:=max(x,l-1)
  else
    max:=x[l];
end;

begin
  clrscr;
  write('n=');
  readln(n);
  for i:=1 to n do
    begin
```



```
      write('a[',i,']=');
      readln(a[i]);
    end;
  write('Elementul maxim este ',max(a,n));
  readln;
end.
```

P20. Minim şi maxim simultan

Să se determine simultan minimul şi a maximul dintr-un şir de n numere naturale.

Rezolvare:

Pentru determinarea simultană a maximului şi minimului dintr-un şir nu se utilizează o funcţie (aceasta ar trebui să returneze două valori), ci o procedură (minmax) care are ca parametri, pe lângă şir (a) şi lungimea şirului (n) încă doi parametri transmişi prin adresă (min şi max), prin intermediul cărora se vor *"returna"* cele două valori căutate.

Procedura are şi două variabile locale (minim şi maxim), utilizate pentru preluarea valorilor ca urmare a apelului recursiv. Dacă şirul conţine mai mult de un element (n>1), se apelează recursiv procedura pentru subşirul format din n-1elemente, iar apoi, valorile returnate prin intermediul parametrilor transmişi prin adresă sunt comparate cu ultimul element, şi, dacă este cazul se actualizează.

Dacă şirul are un singur element (n=1), atunci acesta este şi minim şi maxim, făcându-se iniţializarea variabilelor locale.

Programul principal va primi valorile căutate de la procedură prin intermediul a două variabile (mi şi mx) ale căror valori iniţiale nu contează.

Programul Pascal:

```
program minimmax;
uses
  crt;
type
  sir=array[1..100] of byte;
var
  x:sir;
  l,c:byte;
  mi,mx:byte;

procedure minmax(var min,max:byte;a:sir;n:byte);
var
  maxim,minim:byte;
```

```
begin
   if n>1 then
      begin
         minmax(minim,maxim,a,n-1);
         if a[n]>maxim then
            max:=a[n]
         else
            max:=maxim;
         if a[n]<minim then
            min:=a[n]
         else
            min:=minim;
      end
   else
      begin
         min:=a[1];
         max:=a[1];
      end;
end;

begin
   clrscr;
   write('l=');
   readln(l);
   for c:=1 to l do
      begin
         write('x[',c,']=');
         readln(x[c]);
      end;
   minmax(mi,mx,x,l);
   writeln('minimul : ',mi);
   writeln('maximul : ',mx);
   readln;
end.
```

P21. Progresie geometrică:

Să se determine dacă elementele unui șir de numere întregi formează sau nu o progresie geometrică.

Rezolvare:

Funcția booleană (ratie) utilizată verifică dacă raportul elementelor alăturate din șir se păstrează. Funcția are ca parametri șirul (x) și lungimea sa (n). Se verifică mai întâi dacă șirul are două elemente, caz în care funcția returnează valoarea de adevărat (oricare două valori formează o progresie geometrică).

Dacă șirul are mai mult de două elemente, se verifică dacă raportul dintre ultimul element al șirului și penultimul element este același cu raportul dintre penultimul element și antepenultimul (`x[n]/x[n-1]=x[n-1]/x[n-2]`). În caz afirmativ se face apel recursiv cu decrementarea lungimii șirului iar în caz contrar, raportul nefiind același, funcția returnează valoarea false.

Observație: Pentru evitarea erorilor atunci când există valori nule în șir păstrarea raportului se poate verifica prin compararea produselor:
`x[n-1]*x[n-1]=x[n-2]*x[n]`.

Programul Pascal:

```
program progresie;
uses
   crt;
type
   sir=array[1..20] of integer;
var
   a:sir;l,c:byte;

function ratie(x:sir;n:byte):boolean;
begin
   if n=2 then
      ratie:=true
   else
      if (x[n]/x[n-1])=(x[n-1]/x[n-2]) then
         ratie:=ratie(x,n-1)
      else
         ratie:=false;
end;

begin
   clrscr;
   write('l=');
   readln(l);
   for c:=1 to l do
      begin
         write('a[',c,']=');
         readln(a[c]);
      end;
```

```
   if ratie(a,1) then
     writeln('Sirul este o progresie geometrica.')
   else
   writeln('Sirul nu este o progresie geometrica.');
   readln;
end.
```

P22. Conversia în baza b

Să se determine şi să se afişeze reprezentarea unui număr natural dat într-o bază b, b<10.

Rezolvare:

Se ştie că reprezentarea unui număr într-o bază este constituită din resturile obţinute prin împărţii succesive ale numărului la bază, până când se obţine câtul 0, considerate în ordine inversă. Pentru rezolvarea acestei probleme se ţine cont de faptul că operaţiile plasate înaintea apelului recursiv se execută la pătrunderea în adâncime (deci în ordine directă), iar cele aflate după apelul recursiv se execută la revenire (deci în ordine inversă).

Practic, s-a utilizat o procedură recursivă ce are ca parametri numărul de convertit (x) şi baza (b). În procedură se testează mai întâi dacă nu s-a terminat conversia (x>0), caz în care se depune într-o variabilă locală restul (r:=x mod b), se realizează conversia câtului (conv(x div b,b)), iar la revenire se afişează restul păstrat în variabila locală.

Programul Pascal:

```
program convbaza;
uses
   crt;
var
   x:longint;
   b:byte;

procedure conv(x:longint;b:byte);
var r:byte;
begin
   if x>0 then
     begin
       r:=x mod b;
       conv(x div b,b);
       write(r);
     end;
end;
```



```
begin
   clrscr;
   write('x=');
   readln(x);
   write('b=');
   readln(b);
   writeln('Reprezentarea numarului ',x,' in
     baza ',b,' este:');
   conv(x,b);
   readln;
end.
```

P23. Conversia în baza b (variantă)

Rezolvare:

În această variantă s-a utilizat o funcţie recursivă (conv) ce returnează reprezentarea numărului x în baza b sub forma unui şir de caractere. Cifrele reprezentării se obţin ca rest al împărţirii numărului la bază, sunt convertite din numere în caractere (chr(ord('0')+r)) şi sunt adăugate (prin concatenare), în faţă, la şirul de caractere ce constituie reprezentarea câtului împărţirii precedente.

Dacă s-a obţinut câtul 0 se iniţializează reprezentarea cu şirul vid (conv:='').

Programul Pascal:

```
program convbaza;
uses
   crt;
var
   x:longint;
   b:byte;

function conv(x:longint;b:byte):string;
var r:byte;
begin
   if x>0 then
     begin
       r:=x mod b;
       conv:=chr(ord('0')+r)+conv(x div b,b);
     end
   else
```

```
    conv:='';
end;

begin
   clrscr;
   write('x=');
   readln(x);
   write('b=');
   readln(b);
   writeln('Reprezentarea numarului ',x,'
      in baza ',b,' este:');
   writeln(conv(x,b));
   readln;
end.
```

P24. Conversia în baza 10

Dându-se reprezentarea unui număr într-o bază b (b<10), să se determine și să se afișeze reprezentarea acelui număr în baza 10.

Rezolvare:

Se cunoaște relația de conversie a unui număr din baza b în baza 10:

$$\overline{c_{k-1}c_{k-2}...c_2c_1c_0}_{(b)} = c_{k-1}b^{k-1} + c_{k-2}\cdot b^{k-2} + ... + c_2b^2 + c_1b + c_0 =$$

$$= ((...(c_{k-1}b + c_{k-2})b + ... + c_2)b + c_1)b + c_0$$

Reprezentarea numărului în baza b se transmite ca parametru funcției recursive conv, sub forma unui string (s). Dacă șirul de caractere este nevid, se realizează conversia începând cu ultima cifră a reprezentării care se depune într-o variabilă locală, convertită din caracter în număr (`r:=ord(s[length(s)]) - ord('0')`). Pentru a se evita folosirea unui nou parametru, s-a optat pentru eliminarea cifrei extrase din șirul de caractere (`delete(s,length(s),1)`). La revenire este adăugată cifra la rezultatul conversiei (`conv:=conv(s)*b+r`), conform relației de mai sus.

În cazul în care șirul s-a terminat (`length(s)=0`) se inițializează cu 0 rezultatul conversiei.

Programul Pascal:

```
program convbaza;
uses
   crt;

var
   x:string;
```



```
   b:byte;

function conv(s:string):longint;
var
   r:byte;
begin
   if length(s)>0 then
      begin
         r:=ord(s[length(s)])-ord('0');
         delete(s,length(s),1);
         conv:=conv(s)*b+r;
      end
   else
      conv:=0;
end;

begin
   clrscr;
   write('x=');
   readln(x);
   write('b=');
   readln(b);
   writeln('Numarul in baza 10 este: ',conv(x));
   readln;
end.
```

P25. Şir afişat invers

Să se afișeze în ordine inversă un șir de caractere citit de la tastatură și care este terminat cu caracterul punct.

Rezolvare:

Ideea de rezolvare pornește de la faptul că, într-un subprogram recursiv, operațiile aflate înainte de apelul recursiv se execută la pătrunderea în adâncime (în ordine directă), iar cele situate după apelul recursiv se execută la revenire (în ordine inversă). Se utilizează o procedură ce citește un caracter de la tastatură într-o variabilă locală (c) de tip char și afișează acel caracter după apelul recursiv. Apelul se realizează condiționat, dacă respectivul caracter nu este caracterul punct.

Observație: Dacă afișarea caracterului se realizează în afara condiției (tot după apelul recursiv), se va afișa și caracterul punct.

Programul Pascal:

```
program invers;
uses
   crt;

procedure inv;
var
   c:char;
begin
   c:=readkey;
   write(c);
   if c<>'.' then
      begin
         inv;
         write(c);
      end
   else
      writeln;
end;

begin
   clrscr;
   inv;
   readln;
end.
```

P26. Inversarea unui şir

Să se inverseze un şir de caractere dat.

Rezolvare:

Pentru inversarea şirului de caractere se utilizează o procedură (inv) ce are ca parametri şirul de caractere (s – transmis prin adresă!), şi doi contori ce indică poziţia iniţială şi poziţia finală (pi, respectiv pf) a elementelor şirului care au valorile iniţiale 1, respectiv lungimea şirului.

Dacă între poziţiile indicate de cei doi contori se află mai mult de un element (`pf-pi>1`), se interschimbă cele două caractere indicate de contori (`s[pi]` şi `s[pf]`) şi se apelează procedura cu incrementarea contorului pi şi decrementarea contorului pf. În caz contrar procesul de inversare a şirului a luat sfârşit.

Programul Pascal:

```
program invers;
uses
   crt;
var
   sir:string;

procedure inv(var s:string;pi,pf:byte);
var
   x:char;
begin
   if pf-pi>0 then
      begin
         x:=s[pf];
         s[pf]:=s[pi];
         s[pi]:=x;
         inv(s,pi+1,pf-1);
      end;
end;

begin
   clrscr;
   write('introduceti sirul: ');
   readln(sir);
   inv(sir,1,length(sir));
   writeln('sirul invers este: ',sir);
   readln;
end.
```

P27. Palindrom

Să se verifice dacă un şir de caractere este sau nu palindrom (simetric).

Rezolvare:

Se utilizează o funcţie booleană ce primeşte ca parametru şirul de caractere (x) şi verifică dacă acesta conţine mai mult de un caracter (un şir vid sau cu un singur caracter este simetric!). Dacă şirul este de lungime mai mare decât 1 se verifică dacă primul şi ultimul caracter sunt egale. În caz afirmativ se elimină extremităţile şirului (caracterul de pe poziţia 1 şi cel de pe poziţia length(x)) şi se apelează funcţia pentru şirul rămas.

Dacă elementele situate la cele două extremităţi sunt diferite şirul nu este palindrom (funcţia returnează valoarea false). Dacă s-a ajuns la un şir de

lungime 1 sau 0 înseamnă că la extremități s-au aflat întotdeauna caractere identice, deci șirul este palindrom (funcția returnează valoarea true).

Programul Pascal:

```
program palindrom;
uses
   crt;
var
   s:string;

function palin(x:string):boolean;
begin
   if length(X)>1 then
      if x[1]=x[length(x)] then
         begin
            delete(x,1,1);
            delete(x,length(x),1);
            palin:=palin(x);
         end
      else
         palin:=false
   else
      palin:=true;
end;

begin
   clrscr;
   write('Introsuceti sirul : ');readln(s);
   if palin(s) then
      writeln('Sirul este palindrom.')
   else
      writeln('Sirul nu este palindrom.');
   readln;
end.
```

P28. Egalitatea a două șiruri

Să se determine dacă două șiruri de caractere sunt sau nu egale.

Rezolvare:

Pentru a putea fi egale, cele două șiruri de caractere trebuie să aibă aceeași lungime. Din acest motiv, lungimile au fost comparate în programul principal iar apelul funcției booleene de determinare a faptului că cele două șiruri sunt sau nu egale s-a făcut numai dacă lungimile sunt egale.

Funcția (egal) verifică egalitatea primelor caractere din cele două șiruri (`x[1]=y[1]`). Dacă această condiție este adevărată, se elimină cele două caractere din șir și se continuă evaluarea șirurilor rămase. În cazul în care primele caractere sunt diferite, funcția returnează valoarea false.

Această verificare are loc atunci când șirurile nu s-au terminat (`length(x)>0`). În caz contrar înseamnă că toate caracterele au fost găsite identice și funcția returnează valoarea true.

Programul Pascal:

```
program egalitate;
uses crt;
var
   a,b:string;
function egal(x,y:string):boolean;
begin
   if length(x)=0 then
      egal:=true
   else
      if x[1]=y[1] then
         begin
            delete(x,1,1);
            delete(y,1,1);
            egal:=egal(x,y);
         end
      else
         egal:=false;
end;
begin
   clrscr;
   writeln('Introduceti cele doua siruri.');
   readln(a);
   readln(b);
   if (length(a)=length(b)) and egal(a,b) then
      writeln(a,'=',b)
   else
      writeln(a,'<>',b);
   readln;
end.
```

P29. Păsărească

Să se transforme un text în *"limba păsărească"* (prin inserarea după fiecare vocală a unui grup de două litere: consoana *"p"* și vocala respectivă).

Rezolvare:

Şirul de caractere este prelucrat de la sfârşit către început (pentru a se evita prelucrarea caracterelor introduse în şir în procesul de prelucrare). Se utilizează o procedură (pasare) ce primeşte ca parametri şirul de caractere de prelucrat (a – transmis prin adresă!) şi un contor ce indică poziţia caracterului analizat (c – iniţial egal cu lungimea şirului de caractere).

Dacă nu s-a terminat de prelucrat şirul (c>0), se analizează caracterul de pe poziţia c. Dacă acesta este vocală se inserează în şir, pe poziţia c+1 grupul de caractere *'p'* şi vocala respectivă. Procesul continuă cu prelucrarea poziţiei anterioare din şir (pasare(a,c-1)).

Programul Pascal:

```
program pasareasca;
uses
  crt;
var
  s:string;

procedure pasare(var a:string;c:byte);
begin
  if c>0 then
    begin
      if a[c] in
      ['a','e','i','o','u','A','E','I','O','U'] then
        insert('p'+a[c],a,c+1);
      pasare(a,c-1);
    end;
end;

begin
  clrscr;
  write('Introduceti sirul : ');
  readln(s);
  pasare(s,length(s));
  writeln('Textul in "pasareasca" este:');
  writeln(s);
  readln;
end.
```


P30. Anagrama

Să se determine şi să se afişeze dacă două şiruri de caractere sunt sau nu unul anagrama celuilalt (au în componenţă aceleaşi caractere, eventual într-o altă ordine).

Rezolvare:

Pentru determinarea faptului că cele două şiruri sunt sau nu unul anagrama celuilalt s-a utilizat o funcţie booleană (ana) ce primeşte ca parametri cele două şiruri. În programul principal se verifică dacă cele două şiruri au aceeaşi lungime. În caz contrar nu se mai face apelul funcţiei, neexistând posibilitatea ca un şir să fie anagrama celuilalt.

În funcţia recursivă se verifică mai întâi dacă unul din şiruri este vid (deci ambele sunt vide), caz în care funcţia va returna valoarea true, deoarece din cele două şiruri se elimină numai caracterele unui şir ce se găsesc şi în celălalt şir, eliminându-se toate caracterele, înseamnă că toate caracterele unui şir au fost găsite şi în celălalt şir.

În cazul în care nu s-a terminat primul şir (x), se va cerceta dacă primul caracter din primul şir se găseşte în cel de-al doilea şir (pos(x[1],y)>0). Dacă este îndeplinită condiţia se elimină caracterul comun din cele două şiruri şi se apelează funcţia pentru şirurile rămase.

Dacă primul caracter din primul şir nu se găseşte în cel de-al doilea şir funcţia va returna valoarea false.

Programul Pascal:

```
program anagrama;
uses
  crt;
var
  a,b:string;

function ana(x,y:string):boolean;
begin
  if x='' then
    ana:=true
  else
    if pos(x[1],y)>0 then
      begin
        delete(y,pos(x[1],y),1);
        delete(x,1,1);
        ana:=ana(x,y);
      end
    else
      ana:=false;
```

```
end;

begin
  clrscr;
  write('a=');
  readln(a);
  write('b=');
  readln(b);
  if (length(a)=length(b)) and ana(a,b) then
    writeln(a,' este o anagrama a lui ',b)
  else
    writeln(a,' este nu o anagrama a lui ',b);
  readln;
end.
```

P31. Număr prim

Determinarea faptului ca un număr natural dat este sau nu prim.

Rezolvare:

Se utilizează o funcție booleană (prim) care are ca parametri numărul cercetat (x) și un contor (d) care inițial are valoarea 2. Dacă d nu depășește radicalul numărului (este un potențial divizor), atunci se verifică dacă acesta este divizor al numărului. În caz afirmativ numărul nu este prim iar funcția va returna valoarea false. În caz contrar se apelează funcția cu incrementarea contorului. Dacă valoarea contorului depășește radicalul numărului înseamnă că nu s-a găsit nici un divizor, numărul este prim, deci funcția returnează valoarea true.

Observație: Atribuirea `prim:=prim(x,d+1)` poate fi interpretată ca: *"x este sau nu prim după cum este sau nu prim cu d+1"*.

Programul Pascal:

```
program nr_prim;
uses
  crt;
var
  n:word;

function prim(x:word;d:byte):boolean;
begin
  if d>sqrt(x) then
    prim:=true
  else
    if x mod d=0 then
```



```
      prim:=false
    else
      prim:=prim(x,d+1);
end;
begin
  clrscr;
  write('n=');
  readln(n);
  if prim(n,2) then
    writeln(n,' este prim')
  else
    writeln(n,' nu este prim');
  readln;
end.
```

P32. Factori primi

Să se afișeze descompunerea unui număr natural în factori primi.

Rezolvare:

Se utilizează o procedură care afișează, pe rând, divizorii primi ai numărului. Procedura primește ca parametri numărul (x) și un contor (d) cu ajutorul căruia se caută divizorii numărului (inițial d are valoarea 2 – primul potențial divizor al lui x). Dacă descompunerea nu s-a terminat (x>1) se verifică dacă d este divizor al lui x. În caz afirmativ se afișează respectivul divizor, și se continuă descompunerea în factori primi a câtului împărțirii lui x la d (`desc(x div d,d)`). În caz contrar, se încearcă împărțirea lui x la d+1 (`desc(x,d+1)`).

Observații:

- Datorită faptului că, în cazul în care se găsește un divizor, apelul se realizează cu aceeași valoare a contorului d, nu se vor afișa decât divizorii primi.
- În programul principal s-a utilizat write(#8#32#8) pentru ștergerea ultimului caracter *.

Programul Pascal:

```
program desc_fact_primi;
uses
  crt;
var
  n:word;

procedure desc(x:word;d:word);
```

```
begin
if x>1 then
   if x mod d=0 then
      begin
         write(d,'*');
         desc((x div d),d)
      end
   else
      desc(x,d+1)
end;

begin
   clrscr;
   write('n=');
   readln(n);
   write(n,'=');
   desc(n,2);
   write(#8#32#8);
   readln;
end.
```

P33. Evoluția scorului

Un spectator întârziat ajunge la un meci de fotbal în momentul în care tabela de marcaj indică scorul x:y. Să se afișeze o posibilă evoluție a scorului.

Rezolvare:

Se utilizează o procedură ce primește ca parametri numărul de goluri marcate de cele două formații (gazde și oaspeți – inițial 0 și 0). Se afișează scorul și se verifică dacă nu s-a ajuns la scorul final. Dacă nu s-a ajuns la scorul final se verifică dacă gazdele mai au de marcat goluri (până a ajunge la numărul final – `gazde<x`) și, în acest caz, se apelează procedura cu incrementarea numărului de goluri ale gazdelor. În caz contrar, procedura se apelează cu incrementarea numărului de goluri ale oaspeților.

Prin această metodă gazdele înscriu, pe rând, cele x goluri, după care oaspeții înscriu, pe rând, cele y goluri.

Programul Pascal:

```
program evolutia_scorului;
uses
   crt;
var
   x,y:byte;
```



```
procedure scor(gazde,oaspeti:byte);
begin
   writeln(gazde,'-',oaspeti);
   if (gazde<>x) or (oaspeti<>y) then
      if gazde<x then
         scor(gazde+1,oaspeti)
      else
         scor(gazde,oaspeti+1);
end;

begin
   clrscr;
   write('x=');
   readln(x);
   write('y=');
   readln(y);
   scor(0,0);
   readln;
end.
```

P34. Polinom

Să se calculeze valoarea unui polinom într-un punct.

Rezolvare:

Un polinom de grad n are forma:

$$P_n(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_2 x^2 + a_1 x + a_0 =$$
$$= ((...(((a_n)x + a_{n-1})x + a_{n-2})x + ... + a_2)x + a_1)x + a_0$$

S-a utilizat o funcție (poli) ce are ca parametri șirul coeficienților (a), gradul polinomului (n), un contor (d), și valoarea lui x. Exploatarea șirului coeficienților se face de la poziția 0 (valoarea inițială a lui d). Mai întâi se testează dacă mai există coeficienți neprelucrați (`d<=n`). În caz afirmativ, se adaugă valoarea coeficientului curent la produsul dintre rezultatul returnat de funcție pentru o valoare mai mare cu o unitate a contorului și x (conform relației de mai sus), iar în caz contrar funcția returnează valoarea 0 (se inițializează suma).

Observație: Recomand testarea modului de funcționare a programului(!) în cazul în care se folosește atribuirea `poli:=a[d]+x*poli(a,n,d+1,x)` (se inversează ordinea termenilor sumei).

Programul Pascal:

```
program polinom;
uses
```

```
   crt;
type
   sir=array[0..100] of shortint;
var
   s:sir;
   l,c:byte;
   x:real;

function poli(a:sir;n:byte;d:byte;x:real):real;
begin
   if d<=n then
      poli:=x*poli(a,n,d+1,x)+a[d]
   else
      poli:=0;
end;

begin
   clrscr;
   write('n=');
   readln(l);
   for c:=l downto 0 do
      begin
         write('a[',c,']=');
         readln(s[c]);
      end;
   write('x=');
   readln(x);
   writeln('Valoarea este ',poli(s,l,0,x):5:3);
   readln;
end.
```

P35. Maimuța

Pe un vas se află trei marinari și o maimuță. Marinarii au o ladă ce conține un număr de banane. Noaptea, un marinar merge la ladă și împarte bananele în trei părți egale. Rămânând o banană i-o dă maimuței și ia o parte din cele trei, ascunzând-o. La fel procedează și ceilalți doi marinari. Dimineața, marinarii împart bananele rămase în trei părți egale și banana rămasă i-o dau maimuței. Să se determine toate numerele mai mici decât o valoare dată n care permit acest scenariu.

Rezolvare:

Funcția booleană recursivă utilizată (cauta) este apelată în programul principal în cadrul unei instrucțiuni for de la 1 până la n, verificându-se fiecare valoare din acest interval, dacă poate constitui o soluție.

Funcția cauta primește ca parametri numărul de banane (x – inițial are valoarea contorului) și numărul de *"operații"* de împărțire la care au fost *"supuse"* bananele (i – inițial are valoarea 0). Se testează mai întâi dacă s-a obținut o soluție (dacă au fost efectuate cele trei împărțiri succesive (`i=3`) și numărul de banane rămase (dimineața, la împărțirea efectuată de către întregul grup) are restul împărțirii la 3 egal cu 1 (banana pentru maimuță).

Dacă s-a obținut o soluție, funcția va returna valoarea true. În cazul în care nu s-au efectuat toate operațiile de împărțire se pune condiția dacă împărțirea curentă poate avea loc conform textului problemei (`x mod 3=1`), caz în care se realizează apelul recursiv pentru numărul de banane rămase (`x-x div 3 -1`) și următoarea împărțire (`i+1`).

Dacă împărțirea curentă nu poate avea loc (`x mod 3 <>1`) funcția va returna valoarea false.

Programul Pascal:

```
program maimuta;
uses
   crt;
var
   n,i:word;
   gasitsol:boolean;
   nrsol:word;

function cauta(x:word;i:byte):boolean;
begin
   if (i=3) and (x mod 3=1) then
      cauta:=true
   else
      if x mod 3=1 then
         cauta:=cauta(x-x div 3-1, i+1)
       else
         cauta:=false;
end;

begin
   clrscr;
   write('n=');
   readln(n);
```

```
  nrsol:=0;
  for i:=1 to n do
    if cauta(i,0) then
      begin
        writeln(i);
          nrsol:=nrsol+1;
       end;
  writeln;
  writeln('Nr. de solutii: ',nrsol);
  readln;
end.
```


II. Divide et Impera

Metoda constă în descompunerea problemei de rezolvat în una sau mai multe probleme de același tip, dar de ordin mai mic. Descompunerea în probleme tot mai simple se face succesiv, până când se obține o problemă a cărei rezolvare este banală. Practic se pleacă dimensiunea problemei și aceasta se reduce succesiv, până când problema *"dispare"*.

Deși modalitatea de lucru în cazul aplicării acestei metode ne poate duce cu gândul la recursivitate (acest lucru se întâmplă deoarece majoritatea algoritmilor din capitolul precedent sunt, de fapt, algoritmi *"Divide et Impera"*), nu trebuie să facem confuzie între implementarea recursivă și metoda de programare *"Divide et Impera"*.

"Divide et Impera" este o metodă de programare care poate fi aplicată (ca și celelalte metode de programare) atât iterativ cât și recursiv. Este adevărat, însă, că majoritatea implementărilor problemelor rezolvate prin metoda *"Divide et Impera"* sunt recursive, sau, dacă sunt iterative, în literatura de specialitate nu se amintește nimic de metoda folosită.

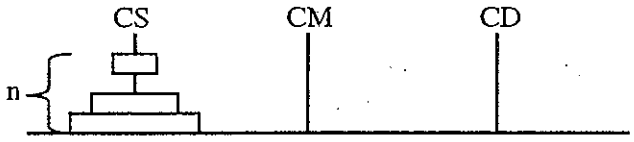
De exemplu, banalul algoritm de sortare prin selecție, este de fapt o rezolvare *"Divide et Impera"* (ca și altele) a problemei sortării. Prin selectarea minimului și aducerea sa în fața șirului (în cazul sortării crescătoare a uni șir prin metoda amintită), nu se face decât să se diminueze dimensiunea șirului pe care-l sortăm.

Practic sortarea prin selecție însemnă aducerea elementului minim (sau maxim în cazul sortării descrescătoare) pe prima poziție și sortarea restului șirului (de la al doilea element până la ultimul). Acest lucru se face până când subșirul rămas de sortat mai are un singur element (subșir care este sortat).

Algoritmii care se găsesc în literatura de specialitate ca fiind *"Divide et Impera"* sunt în număr relativ redus, doar acei care au un nivel de dificultate peste mediu sau cei destinați problemelor a căror abordare printr-o altă tehnică ar fi foarte dificilă.

P36: Turnurile din Hanoi.

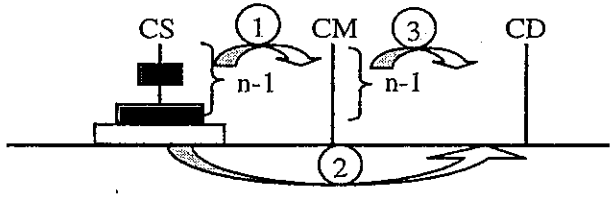
Pe o placă sunt fixate trei cuie: CS (cuiul sursă), CM (cuiul de manevră) și CD (cuiul destinație). Pe CS se află un număr de n discuri de diametre diferite, plasate în ordine descrescătoare a diametrelor de la bază către vârf.



Problema cere mutarea celor n discuri de pe CS pe CD, folosindu-ne de CM și respectând următoarele reguli:

- la o operație se mută un singur disc;
- nu este permis să avem, la nici un moment, un disc de diametru mai mare deasupra unui disc de diametru mai mic.

Rezolvare:



Prin Divide et Impera problema se rezolvă în trei pași:

1. se mută primele n-1 discuri de pe CS pe CM (prin intermediul lui CD)
2. se mută al n-lea disc de pe CS pe CD
3. se mută cele n-1 discuri (mutate la primul pas) de pe CM pe CD (prin intermediul lui CS)

Se pot face următoarele ***observații***:

a) Problema se reduce la două probleme de aceeași natură (a muta o stivă de discuri) dar de ordin mai mic (n-1), și o problemă banală (mutarea unui disc).

b) A muta n-1 discuri înseamnă de fapt mutarea a n-2 discuri, a unui disc și din nou a celor n-2 discuri, din aproape în aproape ajungându-se la mutarea unui singur disc (problemă banală).

c) Rolurile cuielor se schimbă în timp; astfel, de exemplu, la pasul 3 CM este cui sursă, CS este cui de manevră, iar CD este cui destinație.

Programul Pascal:

```
program hanoi;
uses
  crt;
var
  n:byte;


procedure han(x:byte;cs,cm,cd:char);
begin
  if x=1 then writeln('Se muta discul ',x,' de pe
cuiul ',cs,' pe cuiul ',cd)
  else
    begin
      han(x-1,cs,cd,cm);
      writeln('Se muta discul ',x,' de pe cuiul
',cs,' pe cuiul ',cd);
      han(x-1,cm,cs,cd);
    end;
end;

begin
  clrscr;
  write('n=');readln(n);
  han(n,'A','B','C');
  readln;
end.
```

P37. Algoritmul de căutare binară.

Problema cere să se analizeze dacă o anumită valoare se găsește sau nu printre elementele unui șir ordonat crescător.

Rezolvare:

Dacă șirul nu ar fi ordonat, singura posibilitate de rezolvare a problemei ar fi parcurgerea secvențială a șirului. În cazul nostru, șirul fiind sortat, vom proceda la căutarea valorii mai întâi în mijlocul șirului. Dacă limitele de început și de sfârșit ale șirului sunt `li` respectiv `ls`, vom căuta valoarea mai întâi pe poziția de mijloc `m=(li+ls) div 2`. Dacă valoarea căutată va fi găsită pe această poziție căutarea se termină iar în caz contrar ea va continua pe unul din subșirurile de poziții `[li,m-1]` sau `[m+1,ls]` după cum valoarea căutată este mai mică respectiv mai mare față de numărul aflat pe poziția `m`.

Cele două limite între care se face căutarea se apropie una de cealaltă astfel încât, dacă valoarea căutată nu se găsește în șir, acestea se depășesc (li>ls), caz în care căutarea de asemenea se oprește, dar de această dată fără succes.

Programul conține o funcție ce primește ca parametri șirul, valoarea căutată și limitele între care se face căutarea (inițial 1 și lungimea șirului) și returnează poziția pe care se află valoarea căutată sau 0 în cazul în care valoarea căutată nu se găsește în șir.

Programul Pascal:

```
program caubin;
uses
  crt;
type
  sir=array[1..30]of integer;
var
  n:integer;
  a:sir;
  l,i,p:byte;

function bin(b:sir;nr:integer;li,ls:byte):byte;
var
  m:byte;
begin
  m:=(li+ls) div 2;
  if li>ls then
    bin:=0
  else
    if nr<b[m] then
      bin:=bin(b,nr,li,m-1)
    else
      if nr>b[m] then
        bin:=bin(b,nr,m+1,ls)
      else
        bin:=m;
end;

begin
  clrscr;
  write('l=');
  readln(l);
  for i:=1 to l do
    readln(a[i]);
  write('n=');
```



```
  readln(n);
  p:=bin(a,n,1,l);
  if p>0 then
    writeln('Numrul ',n,' se gaseste in sir pe
pozitia ',p)
  else
    writeln('Numrul ',n,' nu se gaseste in sir.');
  readln;
end.
```

P38. Sortare prin interclasare.

Rezolvare:

Metoda pornește de la algoritmul de interclasare a două șiruri care creează, din două șiruri sortate, un șir care conține elementele celor două șiruri, de asemenea sortate.

Algoritmul de sortare prin interclasare operează astfel: pentru a sorta un șir acesta va fi împărțit în două subșiruri ce se vor sorta iar după aceea se vor interclasa.

Subșirurile se vor împărți la rândul lor în câte două subșiruri și așa mai departe. Lungimile subșirurilor vor scădea de la un pas la altul până când acestea vor ajunge la valoarea 1. Cum toate șirurile formate dintr-un singur element sunt sortate, de fapt se reduce problema sortării la problema interclasării.

Programul conține o procedură de sortare (sort) ce implementează algoritmul descris anterior și o procedură de interclasare (interc).

Procedura de sortare are ca parametri șirul (transmis prin adresă!) și limitele subșirului a cărui sortare se dorește (inițial 1 și lungimea șirului).

Procedura de interclasare utilizează trei contori, câte unul pentru fiecare subșir și unul pentru șirul rezultat. Interclasarea se realizează într-un șir intermediar (rez), iar la sfârșit șirul rezultat se transferă în șirul original.

Programul Pascal:

```
program interclasare;
uses
  crt;
type
  sir=array[1..30]of integer;
var
  a:sir;
  n,i:byte;
```

```
procedure interc(var b:sir;li,ls,mij:byte);
var
   i,j,k:byte;
   rez:sir;
begin
   i:=li;
   j:=mij;
   k:=li;
   while(i<=mij - 1)and(j<=ls) do
      begin
         if b[i]<b[j] then
            begin
               rez[k]:=b[i];
               inc(i);
            end
         else
            begin
               rez[k]:=b[j];
                  inc(j);
               end;
         inc(k);
      end;
   if j<=ls then
      while j<=ls do
         begin
               rez[k]:=b[j];
               inc(j);
               inc(k);
         end
   else
      while i<=mij-1 do
         begin
               rez[k]:=b[i];
               inc(i);
               inc(k);
         end;
   for k:=li to ls do
      b[k]:=rez[k];
end;

procedure sort(var b:sir;li,ls:byte);
begin
   if li<ls then
```



```
      begin
         sort(b,li,(li+ls)div 2);
         sort(b,(li+ls)div 2 + 1,ls);
         interc(b,li,ls,(li+ls)div 2 + 1);
      end;
end;

begin
   clrscr;
   write('n=');
   readln(n);
   for i:=1 to n do
      readln(a[i]);
   sort(a,1,n);
   writeln;
   writeln('Sirul sortat este:');
   for i:=1 to n do
      write(a[i],',');
   writeln(#8#32);
   readln;
end.
```

P39. Algoritmul de sortare rapidă (Quicksort)

Rezolvare:

Metoda Quicksort constă în următorii paşi: se pivotează un element (în implementarea noastră primul element) al şirului, adică, prin interschimburi succesive, va ajunge într-o poziţie astfel încât în stânga sa să se găsească numai elemente mai mici decât acesta, iar în dreapta sa să se găsească numai elemente mai mari decât acesta. În acest moment acel element (pivotul) este sortat (ocupă poziţia finală în şir), rămânând să sortăm subşirul aflat în stânga şi subşirul aflat în dreapta pivotului. Lungimile subşirurilor de sortat scad de la un pas la altul ajungând în final să aibă valoarea 1. Şirurile cu un singur element fiind sortate, se reduce problema sortării la problema pivotării.

Programul utilizează o funcţie (piv) ce realizează pivotarea primului element şi returnează poziţia pivotului şi o procedură (sort) ce implementează algoritmul descris.

Funcţia piv primeşte ca parametri şirul (transmis prin adresă!), precum şi limitele (x şi y). Pivotarea primului element se face prin utilizarea a doi contori (i şi j) care iniţial se poziţionează pe limitele subşirului (x, respectiv y). În continuare se compară elementele aflate pe poziţiile date de cei doi contori. Dacă se găseşte la stânga un element mai mare decât la dreapta (a[i]>a[j]), se

interschimbă cele două elemente. Indiferent dacă s-a realizat sau nu interschimbul, contorul ce nu indică pivotul (astfel încât la următoarea comparație să se verifice tot elementul propus spre pivotare) trebuie modificat (dacă este cel aflat în stânga – i – trebuie incrementat, iar dacă este cel aflat în dreapta – j – trebuie decrementat).

Pentru a memora care din cei doi contori indică poziția pivotului se utilizează o variabilă booleană (f) care inițial are valoarea true (pivotul este indicat de i) și care își modifică valoarea la fiecare interschimb (f:=not f). Tot acest proces se desfășoară atât timp cât i<j. Când i=j, elementul a fost pivotat, iar valoarea comună a celor doi contori indică pivotul. Această valoare comună este returnată de către funcție.

Programul Pascal:

```
program qsort;
uses
   crt;
type
   sir=array[1..30]of integer;
var
   n,i:byte;
   a:sir;

function piv(var b:sir;li,ls:byte):byte;
var
   i,j:byte;
   aux:integer;
   f:boolean;
begin
   i:=li;
   j:=ls;
   f:=true;
   while i<j do
      begin
         if b[i]>b[j] then
           begin
              f:=not f;
              aux:=b[i];
              b[i]:=b[j];
              b[j]:=aux;
           end;
         if f then
           j:=j-1
         else
```



```
           i:=i+1;
      end;
   piv:=i;
end;

procedure sort(var b:sir;li,ls:byte);
var
   k:byte;
begin
   if li<ls then
     begin
       k:=piv(b,li,ls);
       sort(b,li,k-1);
       sort(b,k+1,ls);
     end;
end;

begin
   clrscr;
   write('n=');
   readln(n);
   for i:=1 to n do
       begin
        write('a[',i,']=');
        readln(a[i]);
       end;
   sort(a,1,n);
   writeln('Sirul sortat este:');
   for i:=1 to n do
     write(a[i],' ');
   readln;
end.
```

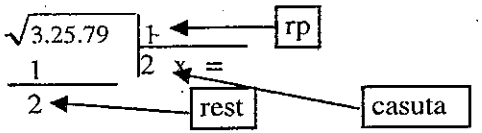
P40. Radicalul unui număr

Să se calculeze partea întreagă a radicalului unui număr natural dat.

Rezolvare:

Pentru a rezolva această problemă să ne aducem aminte de algoritmul de extragere a rădăcinii pătrate dintr-un număr: mai întâi cifrele se grupează câte două de la dreapta la stânga, iar exploatarea grupărilor se va face de la stânga spre dreapta; datorită acestui lucru, procedura de afișare a radicalului (rad) va avea o variabilă locală u2c în care, la intrare în procedură, se vor depune

ultimele două cifre ale numărului, urmând ca prelucrarea lor să se realizeze după apelul recursiv (pe revenire) – vezi P25.



Algoritmul utilizează două variabile globale rp și rest ce memorează radicalul parțial (primele cifre ale rezultatului, care au fost determinate până la un anumit moment), respectiv restul parțial (diferența obținută conform desenului). Pentru determinarea valorii ce poate fi înmulțită cu dublul radicalului parțial astfel încât produsul să nu depășească restul (completat cu ultimele două cifre) se folosește funcția casuta.

Funcția casuta determină cea mai mare cifră care poate ocupa locul în *"căsuță"* astfel încât produsul rezultat să nu depășească restul parțial. Această cifră se află pornind cu un contor de la valoarea 9 (cifra maximă a bazei 10) și decrementând contorul atât timp cât produsul depășește restul parțial (în funcție – prod).

Procedura rad (de extragere a radicalului și de afișare a acestuia, cifră cu cifră, pe măsură ce valoarea este determinată, extrage ultimele două cifre (în u2c) – dacă mai există cifre în număr - , iar după apelul recursiv coboară încă (precedentele) două cifre pe care le alipește la restul anterior (`rest:=rest*100+u2c`). Se determină cu ajutorul funcției casuta următoarea cifră a rezultatului (ce se păstrează în variabila locală cifra – care se și afișează), actualizând apoi restul (`rest:=rest-(10*rp*2+cifra)*cifra`) și se adaugă ultima cifră determinată la precedentul radical parțial (`rp:=rp*10+cifra`). După ce s-a prelucrat și ultima grupare de două cifre (extrasă înainte de apel – extragere ce se face până când nu mai există cifre – numărul devine 0), algoritmul ia sfârșit.

Programul Pascal:

```
program radical;
uses
   crt;
var
   rp,rest:word;x:longint;

function casuta(prod,prefix:longint):byte;
var
   c:byte;
begin
   c:=9;
   while (prefix*10+c)*c>prod do
```



```
      c:=c-1;
   casuta:=c;
end;

procedure rad(n:longint);
var
   u2c,cifra:byte;
begin
   if n>0then
     begin
          u2c:=n mod 100;
          rad(n div 100);
          rest:=rest*100+u2c;
          cifra:=casuta(rest,rp*2);
          write(cifra);
          rest:=rest-(10*rp*2+cifra)*cifra;
          rp:=rp*10+cifra;
      end
    else
      begin
        rp:=0;
        rest:=0
      end;
end;

begin
   clrscr;
   write('x=');
   readln(x);
   rad(x);
   readln;
end.
```

P41. Tăierea dreptunghiului

Dându-se două numere naturale, x și y, cu proprietatea că x<y și y-x<x, care reprezintă lungimile laturilor unui dreptunghi, prin decuparea din acest dreptunghi a unui pătrat de latură x se formează un nou dreptunghi de laturi y și y-x. Să se afișeze dimensiunile laturilor dreptunghiurilor ce se pot obține succesiv din cel inițial prin decuparea, la fiecare pas, a unui pătrat.

Rezolvare:

Metoda este directă: se utilizează o procedură ce primește ca parametri dimensiunile laturilor dreptunghiului (p respectiv q). La începutul procedurii se verifică dacă între dimensiunile actuale ale laturilor dreptunghiului există relațiile din enunț, iar în caz afirmativ afișează respectivele dimensiuni și se face apelul recursiv pentru noul dreptunghi obținut prin eliminarea pătratului (cu dimensiunile laturilor q-p, respectiv p).

Observație: Modificați tipul variabilelor (și a parametrilor) în longint sau chiar comp și introduceți valorile laturilor astfel încât raportul să fie cât mai aproape de valoarea

$$\frac{1+\sqrt{5}}{2}$$

Programul Pascal:

```
program patrat_aur;
uses
  crt;
var
  x,y:word;

procedure taie(p,q:word);
begin
  if (q-p<p)and(p<q) then
    begin
      writeln(p,' ',q);
      taie(q-p,p);
    end;
end;

begin
  clrscr;
  write('x=');
  readln(x);
  write('y=');
  readln(y);
  taie(x,y);
  readln;
end.
```

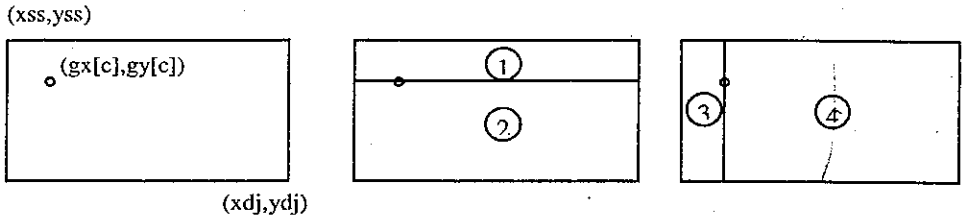

P42. Placa perforată

Dându-se o placă dreptunghiulară de dimensiuni cunoscute în care se află un număr de n găuri punctiforme a căror coordonate de asemenea se cunosc, să se determine porțiunea dreptunghiulară cu laturile paralele cu laturile plăcii, de arie maximă și fără găuri ce se poate decupa din placa inițială.

Rezolvare:

Considerăm x și y ca fiind dimensiunile dreptunghiului inițial și gx, gy două tablouri ce conțin coordonatele x respectiv y ale celor n găuri.

Funcția recursivă ariemax primește ca parametri coordonatele colțului din stânga sus (xss, yss) și cele ale colțului din dreapta jos (xdj,ydj) ale unui dreptunghi, și analizează dacă acesta conține sau nu gaură. Dacă aceasta nu are nici o gaură, aria suprafeței (care este și aria maximă ce se poate decupa din respectivul dreptunghi) este produsul celor două laturi. Dacă acesta are o gaură (c) ale cărei coordonate sunt gx[c] respectiv gy[c], atunci există două posibilități de a tăia dreptunghiul:



rezultând patru dreptunghiuri numerotate în desenul de mai sus. Problema se divide în patru probleme identice și ariile maxime ce se pot decupa din cele patru dreptunghiuri (determinate recursiv) se depun în tabloul s. Pentru calculul maximului din s se utilizează funcția max, iar valoarea returnată de către funcția max este și cea returnată de către funcția ariemax.

Pentru determinarea faptului că un anumit dreptunghi conține sau nu o gaură se utilizează funcția posgaura care returnează poziția în cadrul șirului a unei găuri (de indice maxim), dacă dreptunghiul conține o gaură și valoarea 0 în caz contrar. Acest lucru se realizează prin decrementarea unui contor (inițializat cu n) atât timp cât acesta este nenul și gaura indicată de el nu se află în dreptunghiul (ale cărui coordonate ale colțurilor sunt transmise funcției ca parametri).

Această rezolvare determină doar aria maximă; dacă se dorește determinarea și a coordonatelor colțurilor dreptunghiului de arie maximă în locul funcției ariemax se poate utiliza o procedură care se aibă, în plus, ca parametri (transmiși prin adresă) coordonatele colțurilor dreptunghiului de arie maximă și aria acestuia (cinci parametri suplimentari).

Programul Pascal:

```
program placa;
uses
  crt;
type
  sir=array[1..4]of word;
var
  gx,gy:array[1..30]of byte;
  x,y,i,j,n:byte;

function posgaura(xss,yss,xdj,ydj:byte):byte;
var
  i:byte;
begin
  i:=n;
  while (i>0)and((gx[i]<=xss)or(gx[i]>=xdj)
    or(gy[i]>=yss)or(gy[i]<=ydj)) do
      i:=i-1;
  posgaura:=i;
end;

function max(s:sir):byte;
var
  i,m:byte;
begin
  m:=1;
  for i:=2 to 4 do
    if s[i]>s[m] then
      m:=i;
  max:=m;
end;

function ariemax(xss,yss,xdj,ydj:byte):word;
var
  s:sir;
  c:byte;
begin
  c:=posgaura(xss,yss,xdj,ydj);
  if c=0 then
    ariemax:=(xdj-xss)*(yss-ydj)
  else
    begin
      s[1]:=ariemax(xss,yss,xdj,gy[c]);
```



```
      s[2]:=ariemax(xss,gy[c],xdj,ydj);
      s[3]:=ariemax(xss,yss,gx[c],ydj);
      s[4]:=ariemax(gx[c],yss,xdj,ydj);
      ariemax:=s[max(s)];
    end;
end;

begin
  clrscr;
  write('n=');
  readln(n);
  write('x=');
  readln(x);
  write('y=');
  readln(y);
  for i:=1 to n do
    begin
      write('gx[',i,']=');
      readln(gx[i]);
      write('gy[',i,']=');
      readln(gy[i]);
    end;
  write('aria=',ariemax(0,y,x,0));
  readln;
end.
```

P43: Generarea permutărilor

Să se determine toate anagramele unui șir de ce conține numai caractere distincte.

Rezolvare:

Pentru generarea permutărilor se pornește de la permutarea identică care se construiește în tabloul a de lungime egală cu numărul de caractere din șir. Procedura recursivă de generare a permutărilor, gen, are ca parametru dimensiunea șirului de permutat (inițial lungimea șirului de caractere) – i. Generarea permutărilor de i elemente poate fi rezolvată prin aducerea fiecărei valori pe poziția i în șir (chiar și a celei de pe poziția i!), prin interschimb, generarea permutărilor de i-1 elemente și refacerea stării inițiale (un nou interschimb).

Datorită faptului că apelul recursiv se realizează cu decrementarea parametrului, condiția de oprire este i=1 deoarece o mulțime formată dintr-un singur element nu mai poate genera permutări, ceea ce înseamnă că s-a obținut

o permutare a mulțimii originale, permutare care se afișează cu ajutorul procedurii afis. Pentru creșterea lizibilității procedurii gen, s-a utilizat procedura inters pentru realizarea interschimbului a două elemente ale șirului.

Programul Pascal:

```
program permutari_divide;
uses
   crt;
type
   sir=array[1..15]of byte;
var
   a:sir;
   s:string;
   k:byte;

procedure inters(var x,y:byte);
var
   tmp:byte;
begin
   tmp:=x;
   x:=y;
   y:=tmp;
end;

procedure  afis;
var
   j:byte;
begin
   for j:=1 to length(s) do
     write(s[a[j]]);
   writeln;
end;

procedure gen(i:byte);
var
   j:byte;
begin
   if i=1 then
     afis
   else
     for j:=1 to i do
       begin
         inters(a[i],a[j]);
         gen(i-1);
```



```
         inters(a[i],a[j]);
       end;
end;

begin
   clrscr;
   readln(s);
   for k:=1 to length(s) do
     a[k]:=k;
   gen(length(s));
   readln;
end.
```

III. Backtracking

Metoda Backtracking este o metodă de programare ce se poate aplica problemelor la care soluțiile sunt n-uple de forma $(x_1, x_2, ..., x_i, ..., x_n)$, în care fiecare element din soluție poate lua valori într-o anumită mulțime ($x_i \in A_i$). În plus, de regulă, se caută soluțiile ce îndeplinesc anumite criterii de performanță.

În abordarea acestor tipuri de probleme am putea acționa în mai multe moduri. O metodă de rezolvare ar putea fi **Metoda Forței Brute** care presupune generarea tuturor n-uplelor de forma de mai sus. Practic, se explorează o structură arborescentă, de adâncime n, în care fiecare nod aflat pe nivelul i are un număr de succesori egal cu numărul de elemente ale mulțimii în care poate lua valori (A_i).

Dintre toate șirurile generate doar o parte vor fi soluții ale problemei, urmând să se facă o triere a soluțiilor din această mulțime a pseudosoluțiilor. Datorită faptului că arborele mai sus menționat este foarte mare, această metodă nu este recomandată deoarece are o viteză de execuție foarte mică, fiind în același timp mare consumatoare de resurse.

La polul opus se poate opta pentru utilizarea **Metodei Greedy** (metoda lacomului) care presupune sortarea prealabilă a elementelor mulțimilor și alegerea directă a valorilor x_i (construirea directă a soluției), în ordine, astfel încât să se atingă criteriile de performanță impuse.

Metoda are avantajul unei viteze foarte mari dar are marele dezavantaj că nu furnizează întotdeauna soluția optimă, ba mai mult, în anumite cazuri, nu furnizează nici o soluție deși există soluție.

Metoda Backtracking elimină dezavantajele celor două metode prezentate anterior în sensul că, pe de o parte, nu explorează în întregime arborele mai sus amintit, iar, pe de altă parte, furnizează întotdeauna soluția (chiar toate soluțiile optime), dacă aceasta există.

Metoda construiește pas cu pas soluția pornind de la soluția vidă și adăugând câte o componentă, extinzând soluțiile parțiale succesive, până la obținerea unei soluții finale.

Dacă la un moment dat s-a obținut o soluție parțială de forma $(x_1, x_2, ..., x_{i-1})$ și se pune problema alegerii valorii x_i din soluție, se va încerca, pe rând, ca x_i să ia, toate valorile din mulțimea A_i, însă doar dacă acest lucru nu ar presupune obținerea unei false soluții sau obțincrea unei soluții mai slabe decât soluția obținută anterior.

Datorită faptului că, la alegerea valorii componentei x_i din soluție, se explorează în întregime mulțimea A_i, vom avea certitudinea că nu se pierde nici o soluție (se elimină dezavantajul Metodei Greedy).

Datorită faptului că atribuirea unei anumite valori din mulțimea Ai componentei xi se face condiționat (deci dacă acea valoare ar conduce către o non-soluție, se sare peste valoarea respectivă), din arborele posibilelor soluții se elimină subarbori întregi (înlăturându-se astfel dezavantajul Metodei Forței Brute).

Deși mă feresc de scheme și de șabloane, datorită particularităților problemelor, principiul mai sus explicat s-ar pute descrie schematic (dar trebuie ținut cont că este doar o descriere generală a etapelor realizării unui subprogram recursiv de extindere a unei soluții parțiale către o soluție finală) astfel:

```
procedure ext(i:byte; ...);
var x:..;
begin
   ....
   if <soluție> then
      <tratare soluție>
   else
      for (x∈Ai) do
         if <condiție de extindere> then
            begin
               sol[i]:=x;
               ...
               ext(i+1,...);
               ....
            end;
   ...
end;
```

Procedura recursivă de extindere a soluției parțiale (ext) are, în majoritatea cazurilor, printre parametri poziția ce trebuie completată din vectorul soluție (i). După eventuale operații de marcare a faptului că s-a ajuns la pasul curent (i) din soluție (...), se testează dacă s-a obținut o soluție finală (în multe cazuri condiție este `if i>n`). În cazul în care s-a obținut o soluție completă (finală), aceasta se *"tratează"* (se salvează vectorul soluție (eventual împreună cu alte caracteristici ale soluției), se afișează, etc.).

În cazul în care nu s-a obținut încă o soluție se parcurge mulțimea A_i (mulțimea în care poate lua valori componenta curentă a soluției - x_i). Pentru fiecare valoare din această mulțime se pune condiția dacă aceasta poate fi plasată pe poziția curentă în soluție.

De gradul de elaborare a acestei condiții depinde corecta și rapida funcționare a algoritmului, aceasta fiind cea care elimină subarbori întregi din spațiul potențialelor soluții. Practic, dacă verificarea valorilor componentelor soluțiilor nu se face înainte de considerarea lor ca și componente ale soluției, nu mai avem de a face cu Metoda Backtracking ci cu Metoda Forței Brute (verificarea făcându-se după ce soluția s-a completat).

În cazul în care valoarea curentă (x) poate apărea în soluție pe poziția curentă (i), se memorează acest lucru în vectorul soluție (`sol[i]:=x`), eventual și alte operații, după care se trece la completarea următoarei componente a soluției cu modificarea corespunzătoare a parametrilor (`ext(i+1,...)`).

Dacă înainte de apelul recursiv s-au făcut anumite modificări (de exemplu marcarea faptului că valoarea curentă a fost utilizată), acestea trebuie anulate după apelul recursiv (se reface starea anterioară, pentru ca acele modificări să nu influențeze negativ extinderile ulterioare ale soluției parțiale, în alte direcții).

De asemenea, pot apărea astfel de operații la intrarea în procedură, caz în care ele vor fi anulate prin operații complementare la sfârșitul procedurii.

P44. Problema investitorului:

Un investitor ce dispune de un anumit capital primește un număr de n oferte, fiecare ofertă presupunând investirea unei anumite sume și obținerea unui anumit profit. Capitalul nefiind suficient pentru acceptarea tuturor ofertelor se pune problema realizării unei selecții din ofertele disponibile astfel încât profitul obținut să fie maxim (eventual la același profit maxim se optează pentru soluția cu suma investită minimă).

Rezolvare:

Soluția optimă a acestei probleme se poate obține prin metoda backtracking. Dacă s-ar utiliza Metoda Forței Brute (generarea tuturor posibilităților și căutarea soluției optime în întreg spațiul soluțiilor), timpul de lucru ar fi mare datorită spațiului mare al soluțiilor (2^n). O abordare Greedy (sortarea ofertelor după un anumit criteriu – de exemplu după raportul profit obținut/sumă investită – și acceptarea, în ordine, a ofertelor în limita capitalului disponibil) nu ar oferi întotdeauna soluția optimă; de exemplu, în situația în care, prin respingerea unei oferte ce ar fi selectată prin această metodă, să fie posibilă acceptarea altor două sau mai multe oferte care, împreună, conduc la un profit mai mare.

Abordarea backtracking este implementată în procedura extindere, care are un parametru, i, ce indică oferta analizată la acel pas (inițial i=1). În cazul în care s-a ajuns la oferta n+1 (inexistentă) înseamnă că s-a obținut o soluție și se compară această soluție cu soluția optimă precedentă prin intermediul procedurii testoptim.

În caz contrar, când oferta de analizat există, aceasta va fi, pe rând, atât acceptată cât și respinsă – dacă acest lucru este posibil.

Pentru a verifica dacă o anumită ofertă, i, poate fi acceptată, s-a utilizat un parametru (sump) ce reține suma cheltuită prin acceptarea unor oferte din precedentele i-1 oferte deja analizate (parametru ce inițial – nefiind acceptată nici o ofertă – este 0), care crește la fiecare acceptare a unei oferte și nu se modifică la respingerea unei oferte. În concluzie, o ofertă poate fi acceptată dacă acest lucru nu presupune depășirea capitalului (`sump+s[i]<=cap`).

Pentru a verifica dacă o anumită ofertă, i, poate fi respinsă, s-a utilizat un alt parametru (prof) ce reține profitul potențial (profitul maxim ce mai poate fi atins), care inițial este suma profiturilor aferente tuturor ofertelor (nefiind respinsă nici o ofertă), și care scade la respingerea unei oferte. În concluzie, o ofertă poate fi respinsă dacă acest lucru nu presupune scăderea profitului potențial sub profitul optim obținut la o precedentă soluție (`prof-p[i]>=profopt`).

În cazul în care oferta poate fi acceptată se marchează în șirul de caractere sol acest lucru (`sol[i]:='a'`) și se trece la analiza următoarei oferte (`i+1`)

cu suma investită modificată corespunzător (sump+s[i]), și același profit potențial.

În cazul în care oferta poate fi respinsă se marchează în șirul de caractere sol acest lucru (sol[i]:='r') și se trece la analiza următoarei oferte (i+1) cu aceeași sumă investită, și profitul potențial diminuat corespunzător(prof-p[i]).

Procedura testoptim verifică soluția curentă dacă este mai bună sau la fel de bună cu soluția optimă anterioară. Dacă soluția este mai bună decât soluția optimă anterioară (aduce profit mai mare sau același profit la sumă investită mai mică), se inițializează cu 1 nrsol (numărul de soluții), iar în matricea soluțiilor optime se depune soluția curentă, actualizându-se sumopt și profopt (suma cheltuită respectiv profitul obținut în cazul soluției optime).

Dacă soluția curentă este la fel de bună ca soluția optimă se incrementează numărul de soluții și se depune în matricea soluțiilor optime soluția curentă.

Observație: Datorită faptului că soluțiile (curentă și optime) sunt string-uri în care modificările sunt efectuate la nivel de caracter, **este necesară** inițializarea acestora în scopul asigurării unei lungimi **efective** suficiente!

Programul Pascal:

```
program investitor;
uses
  crt;
var
  cap:word;
  sol:string[20];
  solopt:array[1..50] of string[20];
  j,n,nrsol:byte;
  sumopt,profopt:word;
  s,p:array[1..20] of word;
  sprof:word;

procedure testoptim(prof,sump:word);
begin
  if (prof>profopt) or (prof=profopt) and
  (sump<sumopt) then
    begin
      nrsol:=1;
      profopt:=prof;
      sumopt:=sump;
      solopt[nrsol]:=sol;
    end
  else
    if (prof=profopt) and (sump=sumopt) then
```



```
      begin
        nrsol:=nrsol+1;
        solopt[nrsol]:=sol;
      end;
  end;

procedure extindere(i:byte;sump,prof:word);
begin
  if i=n+1 then
    testoptim(prof,sump)
  else
    begin
      if sump+s[i]<=cap then
        begin
          sol[i]:='a';
          extindere(i+1,sump+s[i],prof);
        end;
      if prof-p[i]>=profopt then
        begin
          sol[i]:='r';
          extindere(i+1,sump,prof-p[i]);
        end;
    end;
end;

begin
  clrscr;
  write('n=');
  readln(n);
  sprof:=0;
  write('capital=');
  readln(cap);
  sol:='
  for j:=1 to n do
    begin
      writeln('Oferta ',j,':');
      write('suma investita=');
      readln(s[j]);
      write('profit=');
      readln(p[j]);
      sprof:=sprof+p[j];
      writeln;
      solopt[j]:=sol;
```

```
    end;
  extindere(1,0,sprof);
  for j:=1 to nrsol do
    writeln(solopt[j]);
  writeln('Suma investita: ',sumopt);
  writeln('Profitul: ',profopt);
  readln;
end.
```

P45. Mere şi pere

Într-un tren există n+1 vagoane. În fiecare din primele n vagoane există câte o grămadă de mere şi câte o grămadă de pere (grămezi cu număr de fructe cunoscut), cel de-al n+1-lea vagon fiind gol. O persoană cu un rucsac (suficient de mare, iniţial gol) merge din vagon în vagon şi trebuie să încarce în rucsac fie toate merele fie toate perele dintr-un vagon, şi să le transporte în vagonul următor. În ultimul vagon (iniţial gol) doar va goli rucsacul, în fiecare vagon rămânând un singur fel de fructe. Ştiind că la transportul unui fruct dintr-un vagon în următorul persoana consumă o calorie, să se determine ce fructe trebuie să transporte din fiecare vagon astfel încât numărul total de calorii consumate să fie minim.

Rezolvare:

Deşi, la prima vedere poate părea altfel, o abordare a acestei probleme Greedy (transportarea, la fiecare pas, a fructelor aflate în număr mai mic) nu conduce la obţinerea soluţiei optime. Pentru exemplificare se poate urmări exemplul:

Nr. fructe / Vagon	1	2	3
Mere	2	2	5
Pere	3	7	10

la care dacă se aplică metoda Greedy se obţine soluţia: mere – mere – mere cu un consum de 2+4+9=15 (calorii), soluţia optimă fiind: pere – mere – mere cu un consum de 3+2+7=12 (calorii).

Procedura recursivă de generare a soluţiei (mp) are ca parametri vagonul curent (v – iniţial 1), numărul de fructe aduse în camera curenta (nrm şi nrp – iniţial 0,0 iar pe parcurs unul nul şi unul nenul), precum şi numărul de calorii consumate până la acel moment (cal – iniţial 0).

La începutul procedurii se *"goleşte rucsacul"* adăugându-se cantităţile de fructe transportate (nrm şi nrp)la grămezile existente corespunzătoare (mere[v],pere[v]). Se verifică dacă s-a obţinut o soluţie (v=n+1) iar în caz afirmativ se salvează soluţia curentă (sol) în solopt, actualizându-se şi valoarea calopt (numărul de calorii consumate în cazul soluţiei optime).

În cazul în care nu s-a ajuns în vagonul n+1, se vor transporta în vagonul următor, pe rând, atât merele cât şi perele, dacă acest lucru nu presupune o depăşire valorii calopt (iniţial cât mai mare – numărul total de fructe). Soluţia se memorează într-un string, pe poziţia v depunându-se caracterul 'M' sau 'P', în funcţie de tipul fructelor transportate.

La sfârşitul procedurii trebuie în mod obligatoriu refăcută starea anterioară prin luarea fructelor depuse la sosirea în vagon (intrarea în procedură), pentru corectitudinea datelor la revenire (întoarcerea în camera anterioară – pentru încărcarea altui tip de fructe – se face cu transportarea în sens invers a fructelor, pentru a nu se modifica numărul fructelor existent iniţial în camere).

Programul Pascal:

```
program mere_pere;
uses
  crt;
type
  sir=string;
var
  sol,solopt:sir;
  mere,pere:array [1..50]of byte;
  calopt,caltot:word;
  n,i:byte;

procedure mp(v,nrm,nrp:byte;cal:word);
begin
  mere[v]:=mere[v]+nrm;
  pere[v]:=pere[v]+nrp;
  if v=n+1 then
    begin
      calopt:=cal;
      solopt:=sol;
    end
  else
    begin
      if cal+mere[v]<calopt then
        begin
          sol[v]:='M';
          mp(v+1,mere[v],0,cal+mere[v]);
        end;
      if cal+pere[v]<calopt then
        begin
          sol[v]:='P';
          mp(v+1,0,pere[v],cal+pere[v]);
```

```
            end;
        end;
    mere[v]:=mere[v]-nrm;
    pere[v]:=pere[v]-nrp;
end;

begin
    clrscr;
    write('Numarul de vagoane: ');
    readln(n);
    caltot:=0;
    for i:=1 to n do
        begin
            write('mere[',i,']=');
            readln(mere[i]);
            write('pere[',i,']=');
            readln(pere[i]);
            caltot:=caltot+mere[i]+pere[i];
        end;
    sol:='
    solopt:=sol;
    calopt:=caltot;
    mp(1,0,0,0);
    writeln(solopt);
    writeln('Calorii consumate: ',calopt);
    readln;
end.
```

P46. Paranteze

Fie n un număr natural par. Să se afișeze toate șirurile de n paranteze închise corect.

Rezolvare:

Procedura recursivă de extindere a soluției (ext) are ca parametri numărul parantezelor deschise și al celor închise folosite până la acel moment (d respectiv i). Se testează mai întâi dacă s-a obținut o soluție finală (s-au folosit n paranteze – d+i=n). În caz afirmativ se afișează soluția cu ajutorul procedurii afis.

În caz contrar, se încearcă pe rând, plasarea în cadrul soluției, pe următoarea poziție (d+i+1) atât a unei paranteze deschise cât și a unei paranteze închise, dacă acest lucru este posibil.

La un moment dat se poate plasa o paranteză deschisă dacă nu au fost utilizate deja toate parantezele deschise (d<n div 2) și se poate plasa o paranteză închisă dacă astfel nu se depășește numărul parantezelor deschise folosite până în acel moment (i<d).

În fiecare din cazuri se plasează paranteza respectivă în vectorul sol și se trece la următoarea poziție din soluție cu incrementarea corespunzătoare a unuia din cei doi parametri ai procedurii.

Programul Pascal:

```
program paranteze;
uses
    crt;
var
    n:byte;
    sol:array[1..30] of char;

procedure afis;
var
    j:byte;
begin
    for j:=1 to n do
        write (sol[j]);
    writeln;
end;

procedure ext(d,i:byte);
var
    j:byte;
begin
    if d+i=n then
        afis
    else
        begin
            if d<n div 2 then
                begin
                    sol[d+i+1]:='(';
                    ext(d+1,i);
                end;
            if i<d then
                begin
                    sol[d+i+1]:=')';
                    ext(d,i+1);
                end;
```

```
      end;
end;

begin
   clrscr;
   repeat
      write('n (nr. par)=');
      readln(n);
   until n mod 2=0;
   ext(0,0);
   readln;
end.
```

P 47. Săritura calului

Să se afișeze toate posibilitățile de acoperire a unei table pătrate nxn cu prin săritura calului de șah, care pleacă din colțul din stânga-sus.

Rezolvare:

Coordonatele relative (față de locația curentă) ale locației în care poate sări calul de șah sunt memorate în doi vectori, x și y. Procedura recursivă utilizată (ext) are ca parametri numărul de ordine al săriturii (i – inițial are valoarea 1) și coordonatele x și y ale locației curente (posx și pos y – ambele având valoarea inițială 1 deoarece se începe parcurgerea tablei din linia 1 și coloana 1).

Mai întâi se marchează în matricea soluție (tabla) numărul săriturii, iar apoi se verifică dacă s-au efectuat toate cele n*n-1 sărituri (i=n*n). În caz afirmativ s-a obținut o soluție, care se afișează cu ajutorul procedurii afis.

În caz contrar se încearcă, pe rând, săritura în fiecare din cele opt potențiale direcții (săritura se poate efectua dacă locația se află pe tablă și nu a fost vizitată anterior). Pentru creșterea lizibilității s-a utilizat funcția booleană cond. Dacă săritura este posibilă se face apelul recursiv cu incrementarea numărului săriturii și cu coordonatele locației actualizate.

La revenire (după apelul recursiv) se va demarca săritura (tabla [posx,posy] :=0), pentru ca locația respectivă să poată fi utilizată ulterior.

Programul Pascal:

```
program calul;
uses
   crt;
const
   x:array[1..8]of shortint=(-1,1,2,2,1,-1,-2,-2);
   y:array[1..8]of shortint=(-2,-2,-1,1,2,2,1,-1);
```



```
var
   tabla:array[1..10,1..10] of byte;
   n:byte;

function cond(j,posx,posy:byte):boolean;
begin
   if (posx+x[j]<=n)and(posx+x[j]>=1)and
      (posy+y[j]<=n)and(posy+y[j]>=1)and
      (tabla[posx+x[j],posy+y[j]]=0) then
      cond:=true
   else
      cond:=false;
end;

procedure afis;
var
   i,j:byte;
begin
   for i:=1 to n do
      begin
         for j:=1 to n do
            write(tabla[i,j]:3);
         writeln;
      end;
   writeln;
end;

procedure ext(i,posx,posy:byte);
var
   j:byte;
begin
   tabla[posx,posy]:=i;
   if i=n*n then
      afis
   else
      for j:=1 to 8 do
         if cond(j,posx,posy) then
            ext(i+1,posx+x[j],posy+y[j]);
   tabla[posx,posy]:=0;
end;

begin
   clrscr;
```

```
   write(' n : ');
   readln(n);
   ext(1,1,1);
   readln;
end.
```

P 48. Săritura calului (circular)

Variantă a problemei precedente cu deosebirea că se cer doar soluțiile ciclice (calul trebuie să poată sări din ultima locație în prima), fără soluțiile ce presupun parcurgerea unui anumit traseu în sens invers.

Rezolvare:

Rezolvarea ține seama că din locația (1,1) calul poate sări doar în locația (2,3) sau în locația (3,2). Dacă se forțează plecarea prin locația (2,3) prin considerarea locației (1,1), în programul principal, ca fiind primul pas și realizând apelul cal(2,2,3) (se pleacă de la a doua săritură) este suficient să testăm în procedură dacă s-a ajuns în locația (3,2). În momentul în care s-a ajuns la ultima săritură s-a găsit o soluție, trecerea anterioară prin locația (3,2) nefiind permisă de noua condiție de realizare a apelului recursiv.

Programul Pascal:

```
program cal_ciclic;
uses
   crt;
const
   dx:array[1..8]of shortint=(-1,1,2,2,1,-1,-2,-2);
   dy:array[1..8]of shortint=(-2,-2,-1,1,2,2,1,-1);
var
   tabla:array[1..10,1..10] of byte;
   n,i,j:byte;

procedure afis;
var
   i,j:byte;
begin
   for i:=1 to n do
     begin
        for j:=1 to n do
          write(tabla[i,j]:3);
        writeln;
     end;
   writeln;
```



```
end;

procedure cal(i,l,c:byte);
var
   m:byte;
begin
   tabla[l,c]:=i;
   if i=n*n then
      begin
        if (l=3)and(c=2) then
           afis;
      end
   else
      for m:=1 to 8 do
        if (l+dy[m]<>3)or(c+dx[m]<>2)or(i=n*n-1)then
           if(l+dy[m]>=1)and(l+dy[m]<=n)and
                (c+dx[m]>=1)and(c+dx[m]<=n)and
                (tabla[l+dy[m],c+dx[m]]=0) then
             cal(i+1,l+dy[m],c+dx[m]);
   tabla[l,c]:=0;
end;

begin
   clrscr;
   write('n=');
   readln(n);
   for i:=1 to n do
      for j:=1 to n do
        tabla[i,j]:=0;
    tabla[1,1]:=1;
   cal(2,2,3);
end.
```

P49. Labirint

Să se găsească ieșirea dintr-un labirint, pornind dintr-o locație ale cărei coordonate se cunosc.

Rezolvare:

Labirintul a fost memorat ca o matrice de caractere (cu semnificația prezentată în secțiunea const). Pentru simplitatea testării programului s-a utilizat pentru construirea algoritmului o procedură (initializare).

Procedura recursivă utilizată (labirintul) primește ca parametri linia și coloana locației curente (l, respectiv c). Mai întâi se marchează locația curentă ca fiind vizitată (`lab[l,c]:=trecut`), pentru a se evita o revenire în această locație dintr-una vecină. Se testează apoi dacă s-a ajuns la o margine a dreptunghiului, caz în care se marchează acest lucru prin intermediul unei variabile booleene (`gasitsol:=true` – variabilă inițializată cu valoarea false în programul principal).

Această marcare a faptului că s-a găsit o soluție este utilă pentru a se realiza traseul corespunzător soluției. Locațiile parcurse la pătrunderea în adâncime se marchează doar ca fiind parcurse (cu *"trecut"*), iar marcarea traseului corespunzător soluției se face diferit (cu *"drum"*) la revenire (după ce s-a găsit soluția).

Dacă nu s-a ajuns la o margine se încearcă, pe rând, deplasarea către cele patru direcții posibile (nord, sud, est vest), dacă acest lucru este posibil. Deplasarea într-o locație adiacentă se poate face dacă aceasta este marcată cu *"loc"*, la prima încercare de deplasare. La următoarele trei încercări apare și condiția suplimentară `not gasitsol`, pentru ca în momentul în care s-a găsit soluția să se producă revenirea, fără încercări suplimentare de deplasare.

În programul principal, după apelul procedurii recursive se afișează soluția (configurația labirintului cu marcajele corespunzătoare). Ilustrarea modului în care lucrează algoritmul este mai convingătoare dacă se optează pentru afișarea configurației la fiecare pas (apelul procedurii afisare în procedura labirintul, înainte de prima condiție).

Programul Pascal:

```
program labirint;
uses
   crt;
const
   zid='°';
   loc=' ';
   trecut='#';
   drum='*';

var
   lab:array [1..50,1..90] of char;
   m,n:byte;
   li,ci:byte;
   gasitsol:boolean;

procedure labirintul(l,c:byte);
begin
   lab[l,c]:=trecut;
```



```
   if (l=m) or (c=n) or (l=1) or (c=1) then
     gasitsol:=true
   else
     begin
       if lab[l-1,c]=loc then
         labirintul(l-1,c);
       if not gasitsol then
         if lab[l+1,c]=loc then
           labirintul(l+1,c);
       if not gasitsol then
         if lab[l,c+1]=loc then
           labirintul(l,c+1);
       if not gasitsol then
         if lab[l,c-1]=loc  then
           labirintul(l,c-1);
     end;
   if gasitsol then
     lab[l,c]:=drum;
end;

procedure initializare;
begin
   gasitsol:=false;
   m:=5;
   n:=5;
   li:=3;
   ci:=3;
   lab[1,1]:=zid;
   lab[1,2]:=zid;
   lab[1,3]:=zid;
   lab[1,4]:=zid;
   lab[1,5]:=zid;
   lab[2,1]:=loc;
   lab[2,2]:=loc;
   lab[2,3]:=loc;
   lab[2,4]:=zid;
   lab[2,5]:=zid;
   lab[3,1]:=zid;
   lab[3,2]:=loc;
   lab[3,3]:=loc;
   lab[3,4]:=loc;
   lab[3,5]:=zid;
   lab[4,1]:=zid;
```

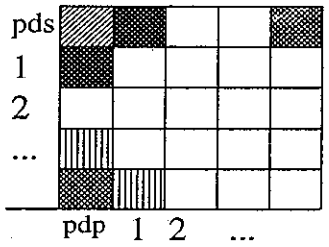
```
   lab[4,2]:=zid;
   lab[4,3]:=zid;
   lab[4,4]:=loc;
   lab[4,5]:=loc;
   lab[5,1]:=zid;
   lab[5,2]:=zid;
   lab[5,3]:=zid;
   lab[5,4]:=zid;
   lab[5,5]:=zid;
end;

procedure afisare;
var
   i,j:byte;
begin
   for i:=1 to m do
     begin
        for j:=1 to n do
          write(lab[i,j]);
        writeln;
     end;
end;

begin
   clrscr;
   initializare;
   labirintul(li,ci);
   afisare;
   readln;
end.
```

P50. Dame

Să se genereze toate posibilitățile de așezare a n dame de șah pe o tablă pătrată cu nxn locații, astfel încât acestea să nu se atace.

Rezolvare:

La amplasarea unei dame pe tablă într-o anumită locație (linia l și coloana c) se ocupă linia și coloana respectivă precum și două diagonale (o paralelă la diagonala principală și o paralelă la diagonala secundară). Paralelele la diagonala principală și paralelele la diagonala secundară vor fi notate de la 1 la 2n-1, de la stânga la dreapta, ca în desenul de mai jos.



Se poate constata că o locație (l,c) este situată pe paralela la diagonala principală notată cu n-l+c și pe paralela la diagonala secundară notată cu l+c-1.

Procedura recursivă utilizată (dame) are ca parametru numărul de ordine al damei ce urmează a fi așezată pe tablă, care coincide cu numărul liniei pe care va fi așezată dama respectivă (l – inițial are valoarea 1). Datorită acestei legături între linie și numărul de ordine al damei se va evita generarea unor soluții multiple.

Se verifică mai întâi dacă au fost așezate pe tablă toate cele n dame (`l>n`), caz în care se afișează soluția obținută, cu ajutorul procedurii afisare. În caz contrar, se încearcă depunerea damei l pe fiecare coloană de pe linia l, dacă acest lucru este posibil. O damă poate fi așezată într-o locație (l,c) dacă atât coloana cât și cele două paralele la diagonale sunt libere (`coloana[c] and pds[l+c-1] and pdp[n-l+c]`).

În cazul în care dama poate fi depusă în acea locație se marchează acest fapt în matricea soluție (`mat[l,c]:='d'`) și se ocupă coloana (`coloana[c]:=false`) și cele două paralele la diagonale (`pds[l+c-1]:=false` și `pdp[n-l+c]):=false`.

În continuare se trece la așezarea următoarei dame (`dame(l+1)`), având grijă ca la revenire să se elibereze coloana (`coloana[c]:=true`) și cele două paralele la diagonale (`pds[l+c-1]:=true` și `pdp[n-l+c]):=true`).

```
program dame_sah;
uses
   crt;
var
   n,x,y:byte;
   coloana,pdp,pds:array[1..19]of boolean;
   mat:array[1..10,1..10] of char;
   nrsol:word;

procedure afisare;
var
```

```
   i,j:byte;
begin
   nrsol:=nrsol+1;
   for i:=1 to n do
      begin
         for j:=1 to n do
            write(mat[i,j]:3);
         writeln;
      end;
   writeln('========================');
end;

procedure dame(l:byte);
var
   c:byte;
begin
   if l>n then
      afisare
   else
      for c:=1 to n do
         if coloana[c] and pds[l+c-1] and pdp[n-l+c]
then
            begin
               mat[l,c]:='d';
               coloana[c]:=false;
               pds[l+c-1]:=false;
               pdp[n-l+c]:=false;
               dame(l+1);
                  mat[l,c]:='■';
               coloana[c]:=true;
               pds[l+c-1]:=true;
               pdp[n-l+c]:=true;
               end;
end;
begin
   clrscr;
   write('n=');
   readln(n);
   for x:=1 to n do
      begin
         coloana[x]:=true;
         for y:=1 to n do
         mat[x,y]:='■';
```



```
      end;
   for x:=1 to 2*n-1 do
      begin
         pdp[x]:=true;
         pds[x]:=true;
      end;
   nrsol:=0;
   dame(1);
   writeln('Nr. de solutii: ',nrsol);
   readln;
end.
```

P51. Comisul voiajor

Cunoscându-se o rețea de drumuri (cu lungimile lor) existentă între un număr de n localități să se determine ce traseu trebuie să urmeze un comis voiajor care pleacă din prima localitate și trebuie să ajungă tot în prima localitate, vizitând toate localitățile, astfel încât lungimea totală a drumului parcurs să fie minimă.

Rezolvare:

Problema în discuție este o problemă de grafuri (se cere ciclul hamiltonian de lungime minimă). Memorarea drumurilor se face în matricea drum (drum[i,j] este lungimea drumului de la i la j dacă există drum și 0 în caz contrar.)

Procedura recursivă utilizată (ext) are ca parametri numărul de ordine al localității curente (i – inițial are valoarea 1), localitatea curentă (o – inițial are valoarea 1 – localitățile fiind codificate cu numere de la 1 la n) și distanța parcursă până în localitatea curentă (inițial 0).

La intrarea în procedură (sosirea într-o localitate) se marchează faptul că s-a vizitat localitatea respectivă (`trecut[o]:=true`) pentru a se evita vizitarea de mai multe ori a aceleiași localități și se înscrie localitatea vizitată în vectorul soluție (`sol[i]:=o`). Dacă s-a obținut o soluție, se actualizează soluția optimă existentă (`solopt:=sol`) și distanța parcursă în cadrul soluției optime (`dopt:=d`).

În caz contrar se încearcă deplasarea în fiecare din cele n localități (`for j:=1 to n`). Deplasarea într-o localitate se va face dacă acea localitate nu a fost vizitată (`not trecut[j]`), dacă există drum de la localitatea curentă la acea localitate (`drum[o,j]>0`) și deplasarea în localitatea respectivă ar însemna că ne îndreptăm spre o soluție mai bună decât soluția optimă actuală (`d+drum[o,j]<dopt`).

O tratare specială trebuie să aibă ultima deplasare (al n-lea drum(i=n), către localitatea inițială (j=1)– care a fost vizitată). Acesta se face numai dacă există

drum şi deplasarea ar conduce către o soluţie mai bună decât soluţia optimă actuală.

În cazul în care deplasarea se poate face, se realizează apelul recursiv cu incrementarea numărului de ordine al localităţii, către destinaţia testată şi cu modificarea corespunzătoare a distanţei parcurse (ext(i+1,j,d+drum[o,j])).

După apelul recursiv se demarchează localitatea curentă (ca fiind nevizitată) pentru a fi posibilă o vizitare ulterioară a acesteia.

Soluţia optimă (cu distanţa parcursă minimă) se va afişa în programul principal, după apelul procedurii ext (care generează această soluţie.)

Programul Pascal:

```
program comis;
uses
  crt;
type
  sir=array[1..10]of byte;
var
  drum:array[1..10,1..10]of byte;
  trecut:array[1..10]of boolean;
  dopt:word;
  n,i,j:byte;
  solopt,sol:sir;

procedure ext(i,o:byte;d:word);
var
  j:byte;
begin
  trecut[o]:=true;
  sol[i]:=o;
  if i=n+1 then
    begin
      dopt:=d;
      solopt:=sol;
    end
  else
    for j:=1 to n do
      if ((not trecut[j])and(drum[o,j]<>0)and
      (d+drum[o,j]<dopt))or((i=n)and(j=1)and
      (drum[o,j]<>0)and(d+drum[o,j]<dopt))then
        ext(i+1,j,d+drum[o,j]);
  trecut[o]:=false;
end;
```



```
begin
  clrscr;
  write('n : ');
  readln(n);
  dopt:=0;
  for i:=1 to n do
    begin
      trecut[i]:=false;
      drum[i,i]:=0;
      for j:=i+1 to n do
        begin
          write(i,' ',j,' : ');
          readln(drum[i,j]);
          drum[j,i]:=drum[i,j];
          dopt:=dopt+drum[i,j];
        end;
    end;
  ext(1,1,0);
  for i:=1 to n+1 do
    write(solopt[i],' ');
  readln;
end.
```

P52. Colorarea hărţii

Să se afişeze toate posibilităţile de colorare cu ajutorul a unui număr de r culori a unei hărţi, astfel încât două ţări alăturate să nu aibă aceeaşi culoare.

Rezolvare:

Pentru simplitatea rezolvării se vor numerota ţările cu valori de la 1 la n, iar culorile cu valori de la 1 la r. Starea de vecinătate a două ţări se va memora în matricea v (simetrică după diagonala principală – dacă ţara i este vecină cu ţara j atunci şi ţara j este vecină cu ţara i.

Procedura recursivă utilizată (ext) are ca parametru ţara ce urmează a fi colorată (t – iniţial are valoarea 1). Se va verifica, mai întâi, dacă s-a obţinut o soluţie (s-au colorat toate ţările – t>n), caz în care se afişează soluţia obţinută cu ajutorul procedurii afis (şirul cu culorile cu care este colorată fiecare ţară).

Dacă nu s-a obţinut încă o soluţie, pentru ţara curentă (t) se încercă, pe rând, atribuirea fiecărei culori (for cul:=1 to r), dacă acest lucru este posibil. Pentru verificarea faptului că o anumită ţară poate să primească o anumită culoare, s-a utilizat funcţia booleană se_poate ce verifică dacă nu mai există o altă ţară (c), vecină cu ţara curentă (x), ce este colorată cu aceeaşi culoare (cul).

Dacă țara curentă (t) poate fi colorată cu o anumită culoare (cul), se reține acest fapt în vectorul soluție (sol[t]:=cul) și se trece la țara următoare (ext(t+1)).

Programul Pascal:

```
program col_harta;
uses
   crt;
var
   v:array[1..20,1..20] of boolean;
   rasp:char;
   n,i,j,r:byte;
   sol:array[1..20] of byte;
   nrsol:word;

function se_poate(x,cul:byte):boolean;
var
   c:byte;
   f:boolean;
begin
   f:=true;
   for c:=1 to x-1 do
      if v[c,x] and (sol[c]=cul) then
         f:=false;
   se_poate:=f;
end;

procedure afis;
var
   g:byte;
begin
   nrsol:=nrsol+1;
   for q:=1 to n do
      write(sol[q]:3);
   writeln;
end;

procedure ext(t:byte);
var
   cul:byte;
begin
   if t=n+1 then
      afis
```



```
   else
      for cul:=1 to r do
         if se_poate(t,cul) then
            begin
               sol[t]:=cul;
               ext(t+1);
            end;
end;

begin
   clrscr;
   write('n=');
   readln(n);
   write('r=');
   readln(r);
   for i:=1 to n-1 do
      for j:=i+1 to n do
         begin
            write('Tara ',i,' este vecina cu tara'
                  ,j,'? (d/n)');
            rasp:=readkey;
            writeln(rasp);
            if upcase(rasp)='D' then
               begin
                  v[i,j]:=true;
                  v[j,i]:=true;
               end;
         end;
   nrsol:=0;
   ext(1);
   writeln('Nr. de solutii: ',nrsol);
   readln;
end.
```

P53. Aranjamente cu repetiție

Să se genereze toate aranjamentele cu repetiție de n obiecte luate câte k.

Rezolvare:

Cele n obiecte se codifică prin numere de la 1 la n. Aranjamentele de n luate câte k înseamnă toate grupurile de k numere în care fiecare număr trece prin toate valorile de la 1 la n (deci metoda utilizată este de fapt cea a forței brute).

Procedura recursivă utilizată (ext) are ca parametru poziția în cadrul grupului (i). Mai întâi se verifică dacă s-a completat soluția (i=k+1), caz în care se afișează soluția cu ajutorul procedurii afis.

În caz contrar se plasează, pe rând, pe poziția curentă (i), toate valorile posibile (for j:=1 to n) și se trece la poziția următoare (ext(i+1)).

Programul Pascal:

```
program gen_aranj_rep;
uses
   crt;
type
   sir=array[1..15] of byte;
var
   a:sir;
   k,n:byte;

procedure  afis;
var
   j:byte;
begin
   for j:=1 to k do
      write(a[j]);
   writeln;
end;

procedure ext(i:byte);
var
   j:byte;
begin
   if i=k+1 then
      afis
   else
      for j:=1 to n do
         begin
            a[i]:=j;
            ext(i+1);
         end;
end;

begin
  clrscr;
  write('n=');
  readln(n);
```



```
  write('k=');
  readln(k);
  ext(1);
  readln;
end.
```

P54. Aranjamente fără repetiție

Să se genereze toate aranjamentele fără repetiție de n obiecte luate câte k.

Rezolvare:

Problema este similară precedentei, numai că, pentru a se evita repetarea elementelor, s-a utilizat un șir de valori booleene (folosit – inițial toate elementele au valoarea false) pentru marcarea elementelor utilizate. Un element se va plasa în soluție numai dacă nu a fost plasat anterior (not folosit[j]). Înainte de apelul recursiv obiectul va fi marcat ca folosit (folosit[j]:=true) iar la revenire acesta va fi eliberat pentru o eventuală utilizare ulterioară (folosit[j]:=false).

Programul Pascal:

```
program gen_aranj;
uses
   crt;
type
   sir=array[1..15] of byte;
var
   a:sir;
   k,n:byte;
   folosit:array[1..15]of boolean;

procedure  afis;
var
   j:byte;
begin
   for j:=1 to k do
      write(a[j]);
   writeln;
end;

procedure ext(i:byte);
var
   j:byte;
begin
```

```
    if i=k+1 then
      afis
    else
      for j:=1 to n do
        if not folosit[j] then
          begin
            folosit[j]:=true;
            a[i]:=j;
            ext(i+1);
            folosit[j]:=false;
          end;
end;

begin
  clrscr;
  write('n=');
  readln(n);
  write('k=');
  readln(k);
  ext(1);
  readln;
end.
```

P55. Combinări cu repetiție

Să se genereze toate combinările cu repetiție de n obiecte luate câte k.

Rezolvare:

Metoda este asemănătoare cu cea utilizată în cazul generării aranjamentelor cu repetiție, doar că în acest caz nu trebuie să se țină cont de ordinea elementelor. Din acest motiv, fiecare valoare nu va mai lua toate valorile de la 1 ci va porni de la precedenta (`for j:=a[i-1] to n`). Astfel valorile vor fi în ordine crescătoare (nu neapărat strict crescătoare), eliminându-se dintre aranjamentele cu repetiție variantele cu aceleași elemente dar care apăreau în altă ordine.

Pentru a se putea demara procesul de generare a combinărilor (ținând cont de faptul că valoarea inițială a parametrului i este 1 și că în procedură se utilizează expresia `a[i-1]`), șirul utilizat la memorarea soluției, a, se folosește cu limita inferioară 0 și în programul principal se face atribuirea `a[0]:=1` (pentru ca prima valoare a lui a[1] să fie 1).

Programul Pascal:

```
program gen_comb_rep;
uses
  crt;
type
  sir=array[0..15] of byte;
var
  a:sir;
  k,n:byte;

procedure  afis;
var
  j:byte;
begin
  for j:=1 to k do
    write(a[j]);
  writeln;
end;

procedure ext(i:byte);
var
  j:byte;
begin
  if i=k+1 then
    afis
  else
    for j:=a[i-1] to n do
      begin
        a[i]:=j;
        ext(i+1);
      end;
end;

begin
  clrscr;
  write('n=');
  readln(n);
  write('k=');
  readln(k);
  a[0]:=1;
  ext(1);
  readln;
end.
```

P56. Combinări fără repetiție

Să se genereze toate combinările fără repetiție de n obiecte luate câte k.

Rezolvare:

Metoda este asemănătoare cu cea utilizată la generarea combinărilor cu repetiție, doar că de această dată valorile pe care le va avea elementul de pe o anumită poziție va porni de la valoarea următoare celei de pe poziția precedentă (`for j:=a[i-1]+1 to n`). De asemenea, inițializarea din programul principal va fi `a[0]:=0`, pentru ca a[1] să pornească de la valoarea 1.

Programul Pascal:

```
program gen_comb_f_r;
uses
  crt;
type
  sir=array[0..15] of byte;
var
  a:sir;
  k,n:byte;

procedure  afis;
var
  j:byte;
begin
  for j:=1 to k do
    write(a[j]);
  writeln;
end;

procedure ext(i:byte);
var
  j:byte;
begin
  if i=k+1 then
    afis
  else
    for j:=a[i-1]+1 to n do
      begin
        a[i]:=j;
        ext(i+1);
      end;
end;
```



```
begin
  clrscr;
  write('n=');
  readln(n);
  write('k=');
  readln(k);
  a[0]:=0;
  ext(1);
  readln;
end.
```

P57. Permutări

Să se genereze permutările unei mulțimi cu n obiecte.

Rezolvare:

Pentru simplitate vom genera permutările mulțimii {1,2,...,n}. În acest scop se utilizează o procedură recursivă (ext) ce are ca parametru poziția elementului curent din cadrul permutării. Se verifică mai întâi dacă s-a obținut o soluție (`i=n+1`), caz în care aceasta se afișează cu ajutorul procedurii afis.

În caz contrar, pe poziția curentă din permutare (i), se depun, pe rând, toate valorile (`for j:=1 to n`) care nu au fost depuse anterior (`if not folosit[j]`), se marchează valoarea respectivă ca fiind depusă (`folosit[j]:=true`), se trece la poziția următoare din permutare (`ext(i+1)`), având grijă ca la revenire să se elibereze valoarea folosită (`depus[i]:=false`), pentru a putea fi utilizată ulterior într-o altă poziție.

Programul Pascal:

```
program gen_perm;
uses
  crt;
type
  sir=array[1..15] of byte;
var
  a:sir;
  k,n:byte;
  folosit:array[1..15]of boolean;

procedure  afis;
var
  j:byte;
begin
  for j:=1 to n do
```

```
      write(a[j]);
   writeln;
end;

procedure ext(i:byte);
var
   j:byte;
begin
   if i=n+1 then
      afis
   else
      for j:=1 to n do
         if not folosit[j] then
            begin
               folosit[j]:=true;
               a[i]:=j;
               ext(i+1);
               folosit[j]:=false;
            end;
end;

begin
   clrscr;
   write('n=');
   readln(n);
   for k:=1 to n do
      folosit[k]:=false;
   ext(1);
   readln;
end.
```

P58. Anagrama (backtracking)

Să se genereze toate anagramele unui șir de caractere distincte dat.

Rezolvare:

Rezolvarea este similară celei utilizate la problema precedentă, doar că de data aceasta se citește mai întâi șirul de caractere (s) și lui n i se va atribui lungimea șirului de caractere (`n:=length(s)`). De asemenea, în procedura afis, se va afișa `s[sol[i]]` în loc de `sol[i]`.

Programul Pascal:

```
program anagrame;
uses
```



```
   crt;
var
   n,c:byte;s:string;
   sol:array[1..101]of byte;
   depus:array[1..101]of boolean;

procedure afis ;
var
   i:byte;
begin
   for i:=1 to n do
      write(s[sol[i]]);
   writeln;
end;

procedure extindere(k:byte);
var
   i:byte;
begin
   if k=n+1 then
      afis
   else
      for i:=1 to n do
         if not depus[i]then
            begin
               depus[i]:=true;
               sol[k]:=i;
               extindere(k+1);
               depus[i]:=false;
            end;
end;

begin
   clrscr;
   write('s=');
   readln(s);
   n:=length(s);
   for c:=1 to n do
      depus[c]:=false;
   extindere(1);
   readln;
end.
```

P59. Produsul cartezian

Să se determine produsul cartezian a n mulțimi: M_1, M_2, ..., M_n.

Rezolvare:

Pentru memorarea numărului de elemente ale celor n mulțimi s-a utilizat șirul lung, iar pentru memorarea elementelor mulțimilor s-a utilizat matricea m.

Produsului cartezian este format din **toate** n-uplele de forma $(x_1, x_2, ..., x_n)$ în care $x_i \in M_i$. Din modul de definire a produsului cartezian rezultă că metoda aplicată este de fapt metoda forței brute, doar modul de abordare fiind backtracking.

Procedura recursivă utilizată (extindere) are ca parametru poziția în cadrul n-uplei (x). Dacă s-a completat un element al produsului cartezian (x>n), rezultă că s-a obținut o soluție ce se afișează cu ajutorul procedurii afis. În caz contrar, pe poziția x din n-uplă se depune, pe rând, fiecare din cele lung[x] elemente ale mulțimii M_x și se trece la poziția următoare (extindere(x+1)).

Programul Pascal:

```
program prodcart;
uses
  crt;
var
  sol,lung:array[1..10]of byte;
  m:array[1..20,1..10]of byte;
  n,i,j:byte;

procedure afis;
var
  i:byte;
begin
  write('(');
  for i:=1 to n do
    write(sol[i],',');
  writeln(#8,')');
end;

procedure extindere(x:byte);
var
  c:byte;
begin
  if x>n then
    afis
```



```
  else
    for c:=1 to lung[x] do
      begin
        sol[x]:=m[x,c];
        extindere(x+1);
      end;
end;

begin
  clrscr;
  write('Nr. de multimi=');
  readln(n);
  for i:=1 to n do
    begin
      write('Nr. de elemente ale multimii ',i,'=');
      readln(lung[i]);
    end;
  for i:=1 to n do
    begin
      writeln('Introduceti elementele multimii ',i);
      for j:=1 to lung[i]do
        readln(m[i,j]);
    end;
  writeln('Produsul cartezian este format din:');
  extindere(1);
  readln;
end.
```

P60. Submulțimile unei mulțimi

Fie A o mulțime cu n elemente naturale. Să se genereze toate submulțimile mulțimii A.

Rezolvare:

Submulțimile mulțimii A pot avea între 0 și n elemente, motiv pentru care apelul procedurii recursive utilizate (ext) se realizează într-o instrucțiune for (for k:=0 to n). Procedura recursivă are ca parametru numărul de ordine al elementului din mulțime ce trebuie stabilit (i – inițial are valoarea 1).

Se testează, mai întâi, dacă s-a format întreaga submulțime (i=k+1), caz în care se afișează (cu ajutorul procedurii afis) soluția obținută. În cazul în care nu s-au completat toate elementele submulțimii, elementul curent (i) va primi, pe rând, toate valorile posibile, de la valoarea precedentului element plus o unitate până la n (în vectorul soluție – a – se memorează poziția elementului în

mulțimea inițială – memorată în vectorul m – și nu valoarea acestuia, având grijă ca afișarea să se realizeze corespunzător (m[a[j]]).

Fiecare din aceste valori se memorează în șirul soluție (a[i]:=j) și se trece la următorul element al submulțimii (ext(i+1)).

Programul Pascal:

```
program gen_submultimi;
uses
   crt;
type
   sir=array[0..15] of byte;
var
   a,m:sir;
   k,n:byte;

procedure  afis;
var
   j:byte;
begin
   write('{');
   for j:=1 to k do
      write(m[a[j]],',');
   if (k<>0) then
      write(#8);
   writeln('}');
end;

procedure ext(i:byte);
var
  j:byte;
begin
   if i=k+1 then
      afis
   else
      for j:=a[i-1]+1 to n do
         begin
            a[i]:=j;
            ext(i+1);
         end;
end;

begin
   clrscr;
```



```
   write('n=');
   readln(n);
   a[0]:=0;
   for k:=1 to n do
      begin
         write('A[',k,']=');
         readln(m[k]);
      end;
   for k:=0 to n do
      ext(1);
   readln;
end.
```

P61. Partițiile unei mulțimi

Fie A o mulțime cu n elemente. Să se afișeze toate partițiile mulțimii A.

Rezolvare:

Generarea soluțiilor se face prin marcarea într-un șir (sol), pe fiecare poziție i, submulțimea din care face parte elementul de pe poziția i din mulțimea inițială. Procedura recursivă utilizată (extindere) are ca parametri numărul de ordine al elementului din mulțimea A care urmează să fie plasat într-o submulțime (i – inițial are valoarea 1) și numărul de submulțimi (ns – inițial are valoarea 0.). Se verifică mai întâi dacă s-au plasat în submulțimi toate elementele mulțimii A (i=n+1), caz în care se afișează soluția cu ajutorul procedurii afis ce are ca parametru numărul de submulțimi din care este formată partiția respectivă (nrs).

În cazul în care mai sunt numere de plasat în submulțimi, acestea se vor plasa, pe rând, în fiecare din cele ns+1 submulțimi (prin adăugarea unui nou element la soluție, numărul de submulțimi poate crește cu 1). Se marchează în vectorul soluție faptul că elementul curent a fost repartizat într-o anumită submulțime (sol[i]:=j) și se trece la plasarea în submulțimi ale partiției a următorului element din mulțimea inițială în două moduri.

Dacă elementul a fost plasat în submulțimea ns+1, se incrementează și numărul de submulțimi, iar în caz contrar acesta rămâne la vechea valoare (doar dacă elementul curent a provocat creșterea – cu o unitate – a numărului de submulțimi din partiție, următorul element poate provoca creșterea cu încă o unitate a numărului de submulțimi).

Programul Pascal:

```
program par_mult;
uses
   crt;
```

```
var
   m:array[1..20]of byte;
   sol:array[1..20]of byte;
   n,i:byte;

procedure afis(nrs:byte);
var
   j,k:byte;
begin
   write('A=');
   for j:=1 to nrs do
      begin
         write('{');
         for k:=1 to n do
            if sol[k]=j then
               write(m[k],',');
         write(#8'}U');
      end;
   writeln(#8#32);
end;

procedure extindere(i,ns:byte);
var
   j:byte;
begin
   if i=n+1 then
      afis(ns)
   else
      for j:=1 to ns+1 do
         begin
            sol[i]:=j;
            if j=ns+1 then
               extindere(i+1,ns+1)
            else
               extindere(i+1,ns);
         end;
end;

begin
   clrscr;
   write('n=');
   readln(n);
   for i:=1 to n do
```



```
      begin
         write('A[',i,']=');
         readln(m[i]);
      end;
   extindere(1,0);
   readln;
end.
```

P62. Partițiile unui număr

Fie n un număr natural dat. Să se afișeze toate posibilitățile de scriere a acestui număr ca o sumă de numere naturale nenule.

Rezolvare:

Procedura recursivă utilizată (ext) are ca parametri numărul de ordine al termenului curent din suma parțială (i – inițial are valoarea 1) și suma parțială (s – inițial are valoarea 0). Se verifică mai întâi dacă s-a obținut o soluție (s=n), caz în care se afișează soluția cu ajutorul procedurii afis ce are ca parametru numărul de termeni ai soluției (l).

În cazul în care nu s-a obținut o soluție se încercă, pe rând, pentru termenul i din sumă, toate valorile, începând cu termenul precedent, (pentru a nu se obține aceeași soluție în care termenii să apară în altă ordine) până la cel mai mare care nu provoacă depășirea numărului inițial (`for j:=part[i-1] to n-s`).

Fiecare număr utilizat ca termen este memorat în vectorul soluție (`part[i]:=j`) și se trece la următorul termen, cu actualizarea corespunzătoare a sumei parțiale (`ext(i+1,s+j)`).

În programul principal se va face inițializarea `part[0]:=1` pentru ca la primul apel, când i are valoarea 1, contorul j să pornească de la valoarea 1.

Programul Pascal:

```
program partitii;
uses
   crt;
var
   part:array[0..20]of byte;
   n:byte;

procedure afis(l:byte);
var
   j:byte;
begin
   write(n,'=');
```

```
    for j:=1 to l do
      write(part[j],'+');
    writeln(#8#32);
  end;

  procedure ext(i,s:byte);
  var
    j:byte;
  begin
    if s=n then
      afis(i-1)
    else
      for j:=part[i-1] to n-s do
        begin
          part[i]:=j;
          ext(i+1,s+j);
        end;
  end;

  begin
    clrscr;
    part[0]:=1;
    write('n=');
    readln(n);
    ext(1,0);
    readln;
  end.
```

P63. Veşti bune şi veşti proaste

Un angajat trebuie să transmită şefului său un număr de veşti bune şi un număr de veşti proaste. Care sunt posibilităţile de a transmite cele două tipuri de veşti, ştiind că şeful nu suportă să cunoască la un moment dat mai multe veşti proaste decât veşti bune.

Rezolvare:

Fără a se face distincţie între veştile în sine ci doar între tipul lor (bune sau proaste), problema este o generalizare problemei P46 (numărul veştilor din cele două categorii nu trebuie să fie neapărat egale, deşi, pentru a exista soluţii, trebuie ca numărul de veşti proaste să nu depăşească numărul de veşti bune).

Similar cu metoda folosită la problema P46, procedura de extindere a soluţiei va avea doi parametri: numărul de veşti bune (x) şi numărul de veşti proaste (y). Condiţia de adâncime va fi în acest caz dacă au fost spuse toate

veştile – bune şi proaste – ((x=b)and(y=p)), în acest caz realizându-se afişarea soluţiei.

În caz contrar, se consideră, pe rând, veste bună şi veste proastă, dacă acest lucru este posibil. La un moment dat, angajatul poate da şefului o veste bună dacă mai are veşti bune şi poate da o veste proastă dacă numărul de veşti proaste deja date este mai mic decât numărul de veşti bune date.

Programul Pascal:

```
program vesti;
uses
  crt;
var
  b,p:byte;
  sol:array[1..30] of char;
  n:word;

procedure afis;
var
  j:byte;
begin
  for j:=1 to b+p do
    write (sol[j]);
  writeln;
  n:=n+1;
end;

procedure ext(x,y:byte);
begin
  if (x=b)and(y=p) then
    afis
  else
    begin
      if x<b then
        begin
          sol[x+y+1]:='B';
          ext(x+1,y);
        end;
      if (y<x)and(y<p) then
        begin
          sol[x+y+1]:='P';
          ext(x,y+1);
        end
    end;
```

```
end;

begin
   clrscr;
   write('Nr. de vesti bune : ');
   readln(b);
   write('Nr. de vesti proaste : ');
   readln(p);
   n:=0;
   ext(0,0);
   writeln('Numarul de solutii: ',n);
   readln;
end.
```

P64. Veşti bune şi veşti proaste (variantă)

Variantă a problemei precedente în care se ţine cont şi de conţinutul efectiv al veştilor, nu numai de tipul lor.

Rezolvare:

De data aceasta soluţia este un tablou de string-uri, veştile fiind memorate de asemenea într-un tablou de string-uri (pe primele b poziţii veştile bune, iar pe următoarele p poziţii veştile proaste).

După verificarea faptului că angajatul poate da o veste de o anumită categorie (bună sau proastă), se încearcă pe rând, plasarea fiecărei veşti din acea categorie, care nu a fost dată. Pentru a se reţine dacă o anumită veste a fost dată sau nu s-a utilizat un tablou de valori booleene (data). La plasarea unei veşti în soluţie se marchează acest lucru în şir(`data[j]:=true`), iar după apelul recursiv se elimină marcajul(`data[j]:=false`), pentru a putea utiliza vestea respectivă în alt context (pe altă poziţie din soluţie).

Programul Pascal:

```
program vesti;
uses
   crt;
var
   b,p,c:byte;
   v,sol:array[1..30] of string;
   data:array[1..30] of boolean;
   n:word;

procedure afis;
var
```



```
   j:byte;
begin
   n:=n+1;
   writeln('Solutia ',n);
   for j:=1 to b+p do
      writeln (sol[j]);
   writeln;
end;

procedure ext(x,y:byte);
var
   j:byte;
begin
   if (x=b)and(y=p) then
      afis
   else
      begin
         if x<b then
            for j:=1 to b do
               if not data[j] then
                  begin
                     data[j]:=true;
                     sol[x+y+1]:=v[j];
                     ext(x+1,y);
                     data[j]:=false;
                  end;
         if (y<x)and(y<p) then
            for j:=b+1 to b+p do
               if not data[j] then
                  begin
                     data[j]:=true;
                     sol[x+y+1]:=v[j];
                     ext(x,y+1);
                     data[j]:=false;
                  end
      end;
end;

begin
   clrscr;
   write('Nr. de vesti bune : ');
   readln(b);
   writeln;
```

```
    writeln('Vetile bune:');
    for c:=1 to b do
      begin
        write(c,'.:');
        readln(v[c]);
        data[c]:=false;
      end;
    write('Nr. de vesti proaste : ');
    readln(p);
    writeln;
    writeln('Vetile bune:');
    for c:=b+1 to b+p do
      begin
        write(c,'.:');
        readln(v[c]);
        data[c]:=false;
      end;
  writeln;
  n:=0;
  ext(0,0);
  readln;
end.
```

P65. Arena:

Un dresor trebuie să scoată în arenă un număr de m tigri și n lei. Care sunt posibilitățile astfel încât să nu scoată consecutiv doi tigri?

Rezolvare:

Se utilizează o procedură recursivă (arena) ce are trei parametri: numărul de ordine al animalului ce urmează a fi scos în arenă (i – inițial 1), numărul de tigri scoși anterior (t – inițial 0) și numărul de lei scoși anterior (l – inițial 0). (Unul din cei trei parametri poate lipsi, fiind introdus doar pentru creșterea lizibilității.) Mai întâi se verifică dacă au fost scoase toate animalele în arenă (i=m+n+1), caz în care se afișează soluția.

În caz contrar, se încearcă scoaterea animalului cu numărul de ordine i, pe rând, tigru și leu, dacă acest lucru este posibil. La un moment dat poate fi scos în arenă un tigru dacă este primul animal (i=1) sau precedentul animal nu scos nu a fost tigru (a[i-1]<>'T') și, bineînțeles, dacă nu au fost scoși deja toți tigrii (t<m).

Animalul cu numărul de ordine i poate fi un leu dacă nu au fost scoși toți leii (l<n) și dacă, prin scoaterea unui leu nu ar mai rămâne suficienți lei pentru separarea tigrilor (n - l >= m - t)!

Scoaterea unui anumit animal se marchează prin inserarea în soluție (a - care este de tip string), pe poziția i, a unui caracter corespunzător ('T' sau 'L'). În continuare se realizează apelul recursiv, iar la revenire se elimină caracterul inserat mai sus, refăcându-se starea.

Programul Pascal:

```
program tigri;
uses
  crt;
var
  a:string;
  l,n,m:byte;

procedure arena(i,t,l:byte);
begin
  if i=m+n+1 then
    writeln(a)
  else
    begin
      if(i=1)or(a[i-1]<>'T')and(t<m) then
        begin
          insert('T',a,i);
          arena(i+1,t+1,l);
          delete(a,i,1);
        end;
      if(l<n)and(n-l>=m-t) then
        begin
          insert('L',a,i);
          arena(i+1,t,l+1);
          delete(a,i,1);
        end;
    end;

begin
  clrscr;
  write('Nr. tigri: ');
  readln(m);
  write('Nr. lei: ');
  readln(n);
  arena(1,0,0);
  readln;
end.
```

P66. Armata

Un număr de 2n soldați de înălțimi diferite trebuie așezați pe două linii de câte n persoane, astfel încât, fiecare soldat să aibă în fața și în stânga sa numai soldați mai înalți decât el. Care sunt posibilitățile?

Rezolvare:

Pentru simplitate, vom codifica cei 2n soldați cu numere de la 1 la 2n, în ordine crescătoare a înălțimii. Soluția (sol) este o matrice cu două linii.

Procedura recursivă are un parametru ce reprezintă numărul de ordine (nu codificarea!) al soldatului ce urmează a fi pus în formație (inițial 1). La intrarea în procedură, acest număr de ordine este transformat în poziția pe care o va ocupa soldatul în formație (linia – l și coloana – c din matricea soluție. În continuare se verifică dacă au fost așezați toți soldații în formație (`i>2*n`), caz în care se face afișarea soluției obținute.

În caz contrar, se încearcă așezarea pe acea poziție, a tuturor soldaților. Un soldat poate fi așezat pe o anumită poziție dacă nu a fost așezat anterior pe o altă poziție (`not depus[j]`), dacă este pe prima linie sau are în fața sa un soldat mai înalt decât el (`((l=1)or(j<sol[l-1,c]))` și dacă este pe prima coloană sau are în stânga sa un soldat mai înalt decât el (`((c=1)or(j<sol[l,c-1]))`).

Dacă soldatul respectiv poate ocupa acea poziție, se marchează că acesta a fost introdus în formație pentru a nu fi introdus ulterior (`depus[j]:=true`), se marchează introducerea soldatului în formație (`sol[l,c]:=j`), și se trece la următoarea poziție din formație, după apel refăcându-se starea (`depus[j]:=false`).

Programul Pascal:

```
program soldat;
uses
   crt;
var
   c,n:byte;
   sol:array[1..2,1..25]of byte;
   depus:array[1..50]of boolean;

procedure afis;
var
   i,j:byte;
begin
   for i:=1 to 2 do
     begin
```



```
       for j:=1 to n do
         write(sol[i,j]:3);
       writeln;
     end;
  writeln;
end;

procedure armata(i:byte);
var
   j,l,c:byte;
begin
   l:=(i-1) mod 2+1;
   c:=(i-1) div 2+1;
   if i>2*n then
     afis
   else
     for j:=1 to 2*n do
       if (not depus[j])then
         if ((l=1)or(j<sol[l-1,c]))
           and((c=1)or(j<sol[l,c-1])) then
           begin
             depus[j]:=true;
             sol[l,c]:=j;
             armata(i+1);
             depus[j]:=false;
           end;
end;

begin
   clrscr;
   write('n=');
   readln(n);
   for c:=1 to 2*n do
     depus[c]:=false;
   armata(1);
   readln;
end.
```

P67. Biletul de tramvai

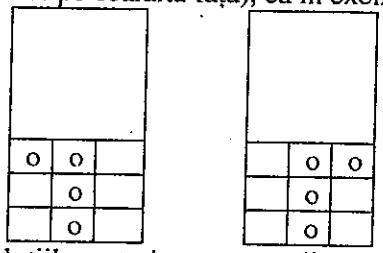
Să se afișeze toate posibilitățile în care poate fi perforat un bilet de tramvai. Se știe că perforatorul are nouă pini dispuși matriceal iar biletul poate fi introdus în aparat cu o față sau cu alte spre interior.

Rezolvare:

Metoda folosită este de fapt metoda forței brute, doar stilul de abordare fiind backtracking.

Procedura recursivă folosită (extindere) are ca parametru numărul de ordine al pinului (i). Dacă i=10, fiind doar 9 pini înseamnă că s-a obținut o soluție finală, care trebuie analizată. În caz contrar, pinul curent va lua pe rând valorile false și true (non perforație, respectiv perforație), valoare ce se va marca în matricea sol, transformându-se numărul pinului în poziții în matrice: pinul i va fi pe linia `(i - 1) div 3 +1` și pe coloana `(i - 1) mod 3+1` și se va trece la pinul următor.

Dacă s-a obținut o soluție, aceasta este mai întâi testată cu ajutorul unei proceduri (test). Practic aceasta soluție este testată cu toate cele nrsol soluții găsite anterior dacă nu cumva este o soluție anterioară *"în oglindă"* (obținută prin introducerea biletului pe cealaltă față), ca în exemplul de mai jos:



Compararea cu soluțiile anterioare se realizează cu ajutorul unei funcții booleene (ogl), care returnează valoarea true dacă soluția curentă este o imagine *"în oglindă"* a altei soluții și false în caz contrar. Dacă soluția nu este imaginea *"în oglindă"* a nici unei alte soluții, se incrementează numărul de soluții și se depune în șirul soluțiilor (sirsol). De asemenea, în acest caz, soluția obținută se afișează cu ajutorul unei proceduri (afis).

Programul Pascal:

```
program bil_tram;
uses
   crt;
type
   mat =array[1..3,1..3]of boolean;
var
   sol:mat;
   sirsol:array[1..300]of mat;
   n,i,j:byte;
   nrsol:word;

procedure afis;
var
```



```
   i,j:byte;
begin
   for i:=1 to 3 do
      begin
         for j:=1 to 3 do
            if  sol[i,j] then
               write(' o ')
            else
               write(' . ');
         writeln;
      end;
   writeln(nrsol);
   if nrsol mod 6=0 then
      begin
         write('Apsati Enter pentru a continua.');
         readln;
      end;
end;

function ogl(s,ss:mat):boolean;
var
   i,j:byte;
   f:boolean;
begin
   f:=true;
   for i:=1 to 3 do
      for j:=1 to 3 do
         if (s[i,j]<>ss[i,3-j+1])then
            f:=false;
   ogl:=f;
end;

procedure test;
var
   i:word;
   f:boolean;
begin
   f:=true;
   for i:=1 to nrsol do
      if ogl(sirsol[i],sol) then
         f:=false;
   if f then
      begin
```

```
        nrsol:=nrsol+1;
        sirsol[nrsol]:=sol;
        afis;
      end;
end;

procedure extindere(i:byte);
var
   j:boolean;
begin
   if i=10 then
     test
   else
     for j:=false to true do
       begin
         sol[(i-1)div 3+1,(i-1)mod 3 +1]:=j;
         extindere(i+1);
       end;
end;

begin
   clrscr;
   nrsol:=0;
   extindere(1);
   readln;
end.
```

P67. Cămilele

Un beduin ce călătorește prin deșert cu o caravană formată din n cămile se hotărăște să schimbe ordinea cămilelor astfel încât nici o cămilă să nu mai vadă în fața ei aceeași cămilă de până atunci. Care sunt posibilitățile?

Rezolvare:

Se consideră o codificare cu valori de la 1 la n a cămilelor, după poziția ocupată inițial în caravană. Procedura recursivă utilizată (ext) are ca parametru poziția în caravană (i).

Dacă s-a completat caravana (`i>n`) atunci înseamnă că s-a obținut o soluție și aceasta se afișează cu ajutorul procedurii afis. În caz contrar se încearcă așezarea în caravană pe poziția i a tuturor cămilelor (`for j:=1 to n`).

O cămilă oarecare (j) poate fi introdusă în caravană pe poziția i dacă nu a fost introdusă anterior (`not ales[j]`) și dacă este prima din caravană (`i=1`) sau în fața ei existase o altă cămilă (`sol[i-1]<>j-1`). Dacă acea cămilă

poate fi introdusă în caravană se marchează acest lucru în vectorul ce memorează soluția (`sir[i]:=j`), se marchează introducerea cămilei în șir pentru a nu mai putea fi introdusă ulterior (`ales[j]:=true`) și se trece la următoarea poziție din caravană (`ext(i+1)`).

După apelul recursiv se reface starea prin demarcarea introducerii cămilei respective în caravană (`ales[j]:=false`), pentru a o putea introduce ulterior pe altă poziție.

Programul Pascal:

```
program camila;
uses
   crt;
var
   sir:array[1..20]of byte;
   ales:array[1..20]of boolean;
   n,i:byte;
   nsol:word;

procedure afis;
var
   j:byte;
begin
   nsol:=nsol+1;
   for j:=1 to n do
     write(sir[j],' ');
   writeln;
end;

procedure ext(i:byte);
var
   j:byte;
begin
   if i=n+1 then
     afis
   else
     for j:=1 to n do
       if not ales[j] and ((i=1) or
       (sir[i-1]<>j-1)) then
         begin
           sir[i]:=j;
           ales[j]:=true;
           ext(i+1);
```

```
            ales[j]:=false;
          end;
end;

begin
   clrscr;
   write('n=');
   readln(n);
   for i:=1 to n do
     ales[i]:=false;
   nsol:=0;
   ext(1);
   writeln('Nr. de solutii: ',nsol);
   readln;
end.
```

P68. Numere cu cifrele unui număr

Să se afișeze toate numerele ce se pot forma cu cifrele unui număr natural dat x și care au numărul de cifre cel mult egal cu numărul de cifre ale numărului inițial.

Rezolvare:

În programul principal se citește numărul x și se extrag cifrele acestuia, marcându-se în vectorul boolean b faptul că o anumită cifră (de la 0 la 9) face parte din numărul inițial, concomitent cu numărarea cifrelor cu ajutorul contorului n.

Procedura recursivă de generare a numerelor (gen) se va apela în programul principal într-o structură repetitivă (for k:=1 to n), generându-se pe rând numere formate dintr-o cifră, din două cifre, până la numerele formate din n cifre.

Procedura gen are ca parametru poziția cifrei în cadrul numărului generat (i). Dacă s-a generat un număr de k cifre (i=k+1) s-a obținut o soluție și se afișează cu ajutorul procedurii afis.

În caz contrar se încearcă depunerea pe poziția curentă (i) a tuturor cifrelor bazei 10 (de la 0 la 9). O cifră a bazei 10 (j) poate fi depusă dacă aceasta face parte din numărul dat (b[j]) și dacă aceasta nu este un 0 nesemnificativ (poate fi 0 doar dacă nu este prima cifră (i>1)sau dacă este unica cifră (k=1)).

În cazul în care cifra poate fi folosită pe poziția respectivă aceasta se memorează în șirul soluție (s[i]:=j) și se trece la următoarea cifră a numărului generat (gen(i+1)).

Programul Pascal:

```
program cifre;
uses
   crt;
var
   b:array[0..9] of boolean;
   s:array[1..5] of byte;
   x:word;
   l,k,n:byte;

procedure afis;
var
   j:byte;
begin
   for j:=1 to k do
     write(s[j]);
   writeln;
end;

procedure gen(i:byte);
var
   j:byte;
begin
   if i=k+1 then
     afis
   else
     for j:=0 to 9 do
       if (b[j])and((k=1)or(i>1)or(j<>0))
         then
           begin
             s[i]:=j;
             gen(i+1);
           end;
   end;

begin
   clrscr;
   write('Nr.: ');
   readln(x);
   for l:=0 to 9 do
     b[l]:=false;
   n:=0;
   while x>0 do
```

```
      begin
        n:=n+1;
        b[x mod 10]:= true;
        x:=x div 10;
      end;
   for k:=1 to n do
     gen(1);
   readln;
end.
```

P69. Obiecte în cutii

Având la dispoziție un număr de n obiecte și k cutii, să se determine care sunt posibilitățile de a depune n_1 obiecte în prima cutie, n_2 obiecte în a două cutie, ..., n_k obiecte în cutia cu numărul k ($n_1+n_2+...+n_k=n$).

Rezolvare:

Soluția va fi reținută într-un vector (sol) de lungime egală cu numărul de obiecte (n), marcând pentru fiecare obiect cutia în care a fost depus.

Procedura recursivă utilizată (extindere) are ca parametru numărul obiectului curent (i). Dacă s-au depus toate obiectele (i=n+1) s-a obținut o soluție, care se afișează cu ajutorul procedurii afis.

În caz contrar se încearcă plasarea obiectului i, pe rând, în fiecare din cele k cutii (for j:=1 to k), dacă acest lucru este posibil. Un obiect (i) poate fi plasat într-o cutie (j) dacă numărul de obiecte existente deja în cutia respectivă (cut[j]) este mai mic decât numărul de obiecte ce trebuie plasate în cutia respectivă (nr[j]). Dacă se poate plasa obiectul în cutie se marchează acest lucru în vectorul soluție (sol[i]:=j), se incrementează numărul de obiecte aflate în cutia respectivă (cut[j]:=cut[j]+1) și se trece la următorul obiect (extindere(i+1)). La revenire se decrementează numărul de obiecte din cutie, pentru refacerea stării anterioare apelului.

Programul Pascal:

```
program cutii;
uses
  crt;
var
  sol,cut,nr:array[1..20]of byte;
  n,i,k:byte;

procedure afis;
var
  i:byte;
```



```
begin
  for i:=1 to n do
    write(sol[i],' ');
  writeln;
end;

procedure extindere(i:byte);
var
  j:byte;
begin
  if i=n+1 then
    afis
  else
    for j:=1 to k do
      if cut[j]<nr[j] then
        begin
          sol[i]:=j;
          cut[j]:=cut[j]+1;
          extindere(i+1);
          cut[j]:=cut[j]-1;
          end;
end;
begin
  clrscr;
  write('nr. obiecte = ');
  readln(n);
  write('nr. cutii = ');
  readln(k);
  for i:=1 to k do
    begin
      write('nr. de obiecte in cutia ',i,'=');
      readln(nr[i]);
      cut[i]:=0;
    end;
  extindere(1);
  readln;
end.
```

P70. Număr ca sumă

Fie n un număr natural dat. Să se determine toate posibilitățile de a scrie acest număr ca o sumă de k numere distincte.

Rezolvare:

S-a utilizat o procedură recursivă (desc) ce are ca parametri numărul de ordine al termenului sumei (i) și suma parțială (s – obținută din precedenții termeni). Se verifică mai întâi dacă suma parțială are k termeni (i>k) și dacă suma este cea dorită (s=n) caz în care s-a obținut o soluție ce se afișează cu ajutorul procedurii afis.

În caz contrar, se utilizează pe rând, ca termeni ai sumei, numerele mai mari decât termenul precedent ce pot apărea în sumă (for c:=sol[i-1]+1 to n-s) care se înscriu în vectorul soluție și se trece la termenul următor.

Afișarea soluției nu se face pe ecran ci în fișierul de tip text descomp.txt

Observație: Datorită modului în care s-au ales valorile termenilor (până la n-s), nu mai este nevoie de o condiție suplimentară pentru a ne asigura că nu se depășește numărul dat (s+c<n).

Programul Pascal:

```
program descompunere;
uses
   crt;
var
   x:text;
   n,k:byte;
   sol:array[0..20]of byte;

procedure afis;
var
   j:byte ;
begin
   for j:=1 to k do
      write(x,sol[j],' ');
   writeln(x);
end;

procedure desc(i,s:byte);
var
   c:byte;
begin
   if i>k then
     begin
        if s=n then
          afis
     end
   else
     for c:=sol[i-1]+1 to n-s do
```



```
        begin
          sol[i]:=c;
          desc(i+1,s+c);
        end;
end;
begin
    clrscr;
  write('n=');
  readln(n);
  write('k=');
  readln(k);
  assign(x,'descomp.txt');
  rewrite(x);
  sol[0]:=0;
  desc(1,0);
  close(x);
end.
```

P71. Discoteca

Într-o discotecă se află un număr de n băieți și m fete (m>=n). Să se afișeze care sunt toate posibilitățile în care băieții își pot alege fetele pentru dans (astfel încât toți băieții să danseze).

Rezolvare:

Numărul de fete fiind mai mare sau egal cu numărul de băieți vor fi n cupluri (vor dansa toți băieții și n din cele m fete). Vectorul soluție va memora pe fiecare poziție i (din cele n) numărul de ordine fetei cu care dansează băiatul cu numărul de ordine i (un număr de la 1 la m).

Procedura recursivă (discot) are ca parametru numărul de ordine al băiatului curent (i – inițial are valoarea 1). Se testează mai întâi dacă s-a obținut o soluție (i=n+1), caz în care aceasta se afișează cu ajutorul procedurii afis.

În caz contrar se încearcă, pe rând, atribuirea ca pereche de dans pentru băiatul i a tuturor celor m fete (for j:=1 to m). O fată poate dansa cu băiatul i dacă nu a fost plasată anterior într-o altă pereche (not al[j]). În acest caz se marchează fata ca fiind într-o pereche de dans (al[j]:=true), se trece la următorul băiat (discot(i+1)), urmând ca la revenire să se marcheze faptul că fata respectivă nu face parte din nici o echipă (al[j]:=false).

Programul Pascal:

```
program disco;
uses
```

```
   crt;
var
   b:array[0..20]  of byte;
   al:array[0..30] of boolean;
   n,m:byte;

procedure afis(j:byte);
var
   k:byte;
begin
   for k:=1 to n do
      write(b[k]);
   writeln;
end;

procedure discot(i:byte);
var
   j:byte;
begin
   if i=n+1 then
      afis(i)
   else
      for j:=1 to m do
         if not(al[j]) then
            begin
               b[i]:=j;
               al[j]:=true;
               discot(i+1);
               al[j]:=false;
            end;
end;

begin
   clrscr;
   write('nr. de baieti : ');
   readln(n);
   repeat
      write('nr. de fete : ');
      readln(m);
   until m>=n;
   discot(1);
   readln;
end.
```


P72. Discoteca la înălțime

Într-o discotecă se află un număr ne m băieți de înălțimi diferite și n fete de înălțimi diferite (m<=n). Să se afișeze toate modurile în care se pot forma cele m perechi de dans, astfel încât orice băiat x mai înalt decât alt băiat y să danseze cu o fată mai mare decât fata cu care dansează y.

Rezolvare:

Cei m băieți pot fi notați, în ordinea crescătoare a înălțimilor, de la 1 la m, iar cele n fete pot fi notate, tot în ordine crescătoare a înălțimilor de la 1 la n. Procedura recursivă utilizată (extindere) are ca parametri numărul de ordine al perechii (care este egal cu numărul de ordine al băiatului – i).

Mai întâi se testează dacă s-a obținut o soluție (`i=m+1`), caz în care aceasta se afișează cu ajutorul procedurii afis. În caz contrar se încercă, pe rând, atribuirea ca pereche de dans pentru băiatul i a tuturor celor n fete (`for j:=1 to n`). O fată poate dansa cu băiatul i dacă nu a fost plasată anterior într-o altă pereche (`not dans[j]`) și dacă este respectată condiția din enunț, implementată de funcția cond.

Funcția booleană cond verifică dacă este satisfăcută condiția prin comparație cu toate perechile anterior fixate.

În cazul în care fata j poate face pereche cu băiatul i se înscrie alegerea făcută în vectorul soluție (`sol[i]:=j`) , se marchează fata ca fiind într-o pereche de dans (`dans[j]:=true`) și se trece la următoarea pereche (`extindere(i+1)`). La revenire să se marcheze faptul că fata respectivă nu face parte din nici o echipă (`dans[j]:=false`), pentru a putea fi ulterior plasată într-o altă echipă.

Programul Pascal:

```
program hbhf;
uses
   crt;
var
   sol:array[1..20]of byte;
   dans:array[1..20]of boolean;
   n,m,i:byte;

procedure afis;
var
   i:byte;
begin
   for i:=1 to m do
      write('(B',i,'<->','F',sol[i],') ');
   writeln;
```

```
end;

function cond(b,f:byte):boolean;
var i:byte;
   fl:boolean;
begin
   fl:=true;
   for i:=1 to b-1 do
      if ((i<b)and(sol[i]>f))or((i>b)and(sol[i]<f))
then
      fl:=false;
   cond:=fl;
end;

procedure extindere(i:byte);
var j:byte;
begin
   if i=m+1 then
      afis
   else
      for j:=1 to n do
         if not dans[j] then
            if cond(i,j) then
               begin
                  sol[i]:=j;
                  dans[j]:=true;
                  extindere(i+1);
                  dans[j]:=false;
               end;
end;

begin
   clrscr;
   write('nr. baieti: ');
   readln(m);
    write('nr. fete: ');
   readln(n);
    for i:=1 to n do
      dans[i]:=false;
   extindere(1);
   readln;
end.
```


P73. Gândacul

Se dă o suprafață dreptunghiulară de dimensiune m x n în care fiecare locație are o anumită înălțime. Un gândac aflat într-o anumită poziție pe această suprafață, dorește să o părăsească. Să se afișeze toate soluțiile de părăsire a suprafeței, știind că gândacul nu poate urca (nu se poate deplasa dintr-o locație într-una vecină cu înălțime mai mare).

Rezolvare:

Înălțimile locațiilor s-au reținut în matricea h. Procedura recursivă utilizată (ext) are ca parametri linia (l) și coloana (c) pe care se află gândacul și numărul de pași parcurși în drumul spre ieșire (i – inițial are valoarea 0), folosit și pentru marcarea drumului parcurs.

La intrarea în procedură se marchează în matricea soluție locația curentă cu litera corespunzătoare pasului la care ne aflăm (punctul de plecare se notează cu 'A', locația următoare cu 'B', ș. a. m. d.: `sol[l,c]:= chr(ord('A') + i))`).

Se testează dacă s-a ajuns la o margine, caz în care înseamnă că s-a găsit o soluție care se afișează cu ajutorul procedurii afis. Dacă nu s-a ajuns la margine, se va încerca, pe rând deplasarea în fiecare din cele patru direcții. Deplasarea într-o anumită direcție este posibilă dacă înălțimea locației curente este mai mare sau egală decât înălțimea locației de destinație și dacă locația de destinație nu a fost vizitată anterior (are marcajul inițial – '*').

Dacă deplasarea într-o anumită direcție este posibilă se realizează apelul recursiv cu actualizarea corespunzătoare a poziției (actualizarea liniei și a coloanei) și incrementarea numărului de pași. La ieșirea din procedură se va demarca locația curentă marcată la intrarea în procedură pentru ca, după revenirea în locația precedentă, locația curentă să mai poată fi vizitată ulterior, dintr-o altă locație.

Programul Pascal:

```
program gandac;
uses
   crt;
var
   i,j,m,n,li,ci,nrsol:byte;
   h:array[1..20,1..20] of byte;
   sol:array[1..20,1..20] of char;

procedure afis;
var
   p,q:byte;
begin
   nrsol:=nrsol+1;
```

```
   for p:=1 to m do
     for q:=1 to n do
       begin
         gotoxy(4*q,p+12);
         write(sol[p,q]);
       end;
   writeln;
   write('Apasa <ENTER> pentru a continua ...');
   readln;
end;

procedure ext(l,c,i:byte);
begin
   sol[l,c]:=chr(ord('A')+i);
   if (l=1) or (c=1) or (l=m) or (c=n) then
     afis
   else
     begin
       if (h[l,c]>=h[l-1,c]) and
       (sol[l-1,c]='*') then
         ext(l-1,c,i+1);
       if (h[l,c]>=h[l+1,c])  and
       (sol[l+1,c]='*') then
         ext(l+1,c,i+1);
       if (h[l,c]>=h[l,c+1])  and
       (sol[l,c+1]='*') then
         ext(l,c+1,i+1);
       if (h[l,c]>=h[l,c-1])  and
       (sol[l,c-1]='*') then
           ext(l,c-1,i+1);
     end;
    sol[l,c]:='*';
end;

begin
   clrscr;
   write('m=');
   readln(m);
   write('n=');
   readln(n);
   for i:=1 to m do
     for j:=1 to n do
       begin
```



```
         sol[i,j]:='*';
         gotoxy(4*j,i+3);
         readln(h[i,j]);
       end;
   write('li=');
   readln(li);
   write('ci=');
   readln(ci);
   ext(li,ci,0);
   writeln('Nr. solutii: ',nrsol);
   readln;
end.
```

P74. Litere duble

Se dă un şir cu n litere distincte. Să se construiască toate şirurile de 2*n caractere ce utilizează de câte două ori literele din şirul dat, astfel încât să nu existe două litere identice consecutive.

Rezolvare:

Se utilizează o procedură recursivă (aranj) ce primeşte ca parametru poziţia curentă în şirul generat (i – iniţial are valoarea 1). Se verifică mai întâi dacă s-a creat întregul şir (`i=2*n+1`), caz în care se afişează soluţia (sol).

În cazul în care şirul nu este complet se încercă plasarea, pe rând, a tuturor celor n caractere din şirul iniţial. Un caracter poate fi plasat în şir dacă anterior nu a fost plasat de două ori (`fol[k]<2`) şi dacă este primul în şir (`i=1`) sau caracterul ce-l precede nu este identic (`sol[i-1]<>sir[k]`).

Dacă respectivul caracter îndeplineşte această condiţie el este plasat în şir (`sol[i]:=sir[k]`), este incrementat numărul de utilizări ale caracterului respectiv (`fol[k]:=fol[k]+1`), se trece la următoarea poziţie din şir (`aranj(i+1)`) iar la revenire se decrementează numărul de utilizări ale caracterului respectiv.

Programul Pascal:

```
program litere;
uses
   crt;
var
   sir,sol:string;
   j,n:byte;
```

```
  fol:array[1..10] of byte;
  nrsol:word;

procedure aranj(i:byte);
var
  k:byte;
begin
  if i=2*n+1 then
    begin
      writeln(sol);
      nrsol:=nrsol+1;
    end
  else
    for k:=1 to n do
      if (fol[k]<2)and((i=1)or(sol[i-1]<>sir[k]))
      then
        begin
          fol[k]:=fol[k]+1;
          sol[i]:=sir[k];
          aranj(i+1);
          fol[k]:=fol[k]-1;
        end;
end;

begin
  clrscr;
  write('Sirul de caractere: ');
  readln(sir);
  n:=length(sir);
  sol:='';
  for j:=1 to 2*n do
    begin
      fol[j]:=0;
      sol:=sol+' ';
    end;
  nrsol:=0;
  aranj(1);
  write('Nr. de solutii: ',nrsol);
  readln;
end.
```


P75. Moştenirea

Doi fraţi moştenesc o avere ce constă într-un număr de n monede de valori distincte. O clauză testamentară prevede ca cei doi să primească averea, dacă vor reuşi să împartă în mod egal monedele, iar în caz contrar banii să fie donaţi unei societăţi caritabile. Ce soluţii de împărţire a averii au cei doi fraţi?

Rezolvare:

În funcţie de paritatea sumei valorile monedelor şi de însăşi valorile acestora, pot exista sau nu soluţii de împărţire a sumei moştenite în două părţi egale. Valorile celor n monede se păstrează în şirul v.

Procedura recursivă utilizată are ca parametri moneda curentă (i – iniţial are valoarea 1) şi sumele deja primite de cei doi fraţi (s1 şi s2 – iniţial au valoarea 0).

Se testează dacă s-au împărţit toate cele n monede (i=n+1), caz în care se afişează soluţia. În caz contrar se va încerca atribuirea monedei i, pe rând, celor doi fraţi. Fiecare din fraţi poate primi o monedă dacă astfel nu ar depăşi jumătate din întreaga sumă moştenită.

În cazul în care moneda se poate atribui unuia dintre fraţi se marchează acest lucru în vectorul sol şi se trece la următoarea monedă, cu modificarea corespunzătoare a sumei primită de respectivul frate.

Programul Pascal:

```
program monede;
uses
  crt;
var
  f:boolean;
  n,c:byte;
  s:word;
  v,sol:array[1..100] of byte;

procedure afis;
var
  x,y:byte;
begin
  f:=true;
  for x:=1 to 2 do
    begin
      write('frate ',x,': ');
      for y:=1 to n do
        if sol[y]=x then
          write(v[y],'+');
      writeln(#8#32);
```

```
   end;
   writeln;
end;

procedure mon(i:byte;s1,s2:word);
begin
   if i=n+1 then
     afis
   else
     begin
       if s1+v[i]<=s div 2 then
         begin
           sol[i]:=1;
           mon(i+1,s1+v[i],s2);
         end;
       if s2+v[i]<=s div 2 then
         begin
           sol[i]:=2;
           mon(i+1,s1,s2+v[i]);
         end;
     end;
end;

begin
   clrscr;
   write('n=');
   readln(n);
   s:=0;
   for c:=1 to n do
     begin
       write('v[',c,']=');
       readln(v[c]);
       s:=s+v[c];
     end;
   f:=false;
   if s mod 2 =0 then
     mon(1,0,0);
   if not f  then
     writeln('Banii merg la societate!!');
end.
```


P76. Moştenirea (generalizare)

Problema precedentă generalizată pentru un număr de f fraţi.

Rezolvare:

De această dată procedura recursivă utilizată (extindere) are ca parametri, în loc de sumele primite de cei n fraţi un vector ce conţine sumele deja primite de cei f fraţi (sf). În locul celor două instrucţiuni if ce verifică dacă fiecare din cei doi fraţi poate primi o anumită monedă, se va utiliza o instrucţiune for (pentru toţi fraţii) ce va conţine o instrucţiune if ce verifică dacă un anumit frate poate primi moneda curentă.

Diferenţe faţă de rezolvarea problemei precedente apar şi în cadrul procedurii de afişare (afis), în acest caz parcurgându-se şirul fraţilor cu testarea fiecărei monede dacă a ajuns în posesia acelui frate.

Programul Pascal:

```
program mostenire;
uses
   crt;
type
   suma=array[1..20] of word;
var
   mon:array[1..20] of byte;
   sol:array[1..20] of byte;
   n,i,k,f:byte;
   s:word;
   w:suma;

procedure afis;
var
   i,j:byte;
begin
   k:=k+1;
   for j:=1 to f do
     begin
       write(j,': ');
       for i:=1 to n do
         if sol[i]=j then
           write(mon[i],'+');
       writeln(#8,'=',s div f);
     end;
   writeln;
end;
```

```
procedure extindere(sf:suma;i:byte);
var
   j:byte;
begin
   if i=n+1 then
      afis
   else
      for j:=1 to f do
         if mon[i]+sf[j]<=s div f then
            begin
               sol[i]:=j;
               sf[j]:=sf[j]+mon[i];
               extindere(sf,i+1);
               sf[j]:=sf[j]-mon[i];
            end;
end;

begin
   clrscr;
   write('n=');
   readln(n);
   s:=0;
   for i:=1 to n do
      begin
         write('mon[',i,']=');
         readln(mon[i]);
         s:=s+mon[i];
         w[i]:=0;
      end;
   write('Nr. frati este ');
   readln(f);
   k:=0;
   if (s mod f=0)and(n>=f) then
      extindere(w,1);
   if k=0 then
      writeln('Banii au fost donati !!!')
   else
      writeln('Nr. de solutii este ',k);
   readln;
end.
```


P77. Matrice 01

Să se genereze toate matricele pătratice n x n care au proprietatea că pe fiecare linie și pe fiecare coloană există un singur element de valoare 1, celelalte elemente fiind 0.

Rezolvare:

Deoarece pe fiecare linie există un singur element de valoare 1, o soluție presupune stabilirea coloanelor pe care se află elementele de 1 de pe fiecare linie (un șir). Procedura recursivă utilizată (ext) are ca parametru linia din matrice pe care urmează să se plaseze o valoare de 1 (l – inițial are valoarea 1). Se verifică mai întâi dacă s-au completat toate liniile (`l>n`), caz în care s-a obținut o soluție ce se afișează cu ajutorul procedurii afis.

În cazul în care nu s-au completat toate liniile, se încearcă depunerea pe linia curentă a valorii de 1, pe rând, pe fiecare din coloane (`for c:=1 to n`). Valoarea de 1 poate fi plasată pe o anumită coloană (c) dacă nu a fost plasată anterior pe coloana respectivă o valoare de 1 (`col[c]`).

În acest caz se memorează coloana ce conține valoarea de 1 în vectorul soluție (`sol[l]:=c`), se marchează coloana respectivă ca având o valoare de 1 (`col[c]:=false`) și se trece la următoarea linie (`ext(l+1)`). La revenire se va elibera coloana ocupată anterior (`col[c]:=true`).

Programul Pascal:

```
program mat01;
uses
   crt;
var
   n,x,y:byte;
   col:array[1..10]of boolean;
   sol:array[1..10] of 1..10;
   nrsol:word;

procedure afisare;
var
   i,j:byte;
begin
   nrsol:=nrsol+1;
   for i:=1 to n do
      begin
         for j:=1 to n do
            if sol[i]=j then
               write('1  ')
            else
```

```
            write('0  ');
        writeln;
      end;
   writeln;
end;

procedure ext(l:byte);
var
   c:byte;
begin
   if l>n then
      afisare
   else
      for c:=1 to n do
         if col[c] then
            begin
               sol[l]:=c;
               col[c]:=false;
               ext(l+1);
               col[c]:=true;
            end;
end;
begin
   clrscr;
   write('n=');
   readln(n);
   for x:=1 to n do
      col[x]:=true;
   nrsol:=0;
   ext(1);
   writeln('Nr. de solutii: ',nrsol);
   readln;
end.
```

P78. Plata cu număr minim de monede

Trebuie achitată o sumă s de unități monetare și se dispune de cantități suficiente de monede de un anumit număr de valori nominale. Care este modalitatea de plată astfel încât numărul de monede utilizat să fie minim?

Rezolvare:

Procedura recursivă utilizată (ext) are ca parametri suma restantă (sr – inițial este întreaga sumă) și numărul de monede utilizate (x – inițial are valoarea 0). Se testează mai întâi dacă suma restantă este egală cu zero, caz în care înseamnă că s-a găsit o (nouă) soluție și aceasta se salvează într-o altă variabilă (sol:=solt), cu actualizarea numărului de monede utilizate în cazul acestei soluții (no:=x-1).

În cazul în care nu s-a obținut încă o soluție se încercă, pe rând, plasarea pe poziția x din soluție a fiecărui tip de monedă, dacă astfel nu s-ar depăși suma restantă (sr-mon[j]>=0) și dacă s-ar tinde spre o soluție cu număr de monede mai mic decât cel din cadrul soluției optime curente (x+1<no).

Dacă se poate plasa o monedă de o anumită valoare nominală se marchează acest lucru în vectorul ce memorează soluția curentă (sol[x]:=mon[j]) și se trece la următoarea monedă, cu actualizarea sumei restante (ext(sr-mon[j],x+1)). Soluția optimă se va afișa în programul principal, după apelul procedurii ext.

Programul Pascal:

```
program money;
uses
   crt;
type
   sir=array[1..60] of byte;
var
   solt,sol,mon:sir;
   ntm,no,i:byte;
   s:word;

procedure ext(sr:longint;x:byte);
var
   j:byte;
begin
   if sr=0 then
      begin
         sol:=solt;
         no:=x-1;
      end
   else
      for j:=1 to ntm do
         if (sr-mon[j]>=0)and(x+1<no) then
            begin
               solt[x]:=mon[j];
```

```
            ext(sr-mon[j],x+1);
          end;
end;

begin
   clrscr;
   write('s : ');
   readln(s);
   write('Nr tipuri monede : ');
   readln(ntm);
   for i:=1 to ntm do
      readln(mon[i]);
   no:=200;
   ext(s,1);
   write(s,'=');
   for i:=1 to no do
      write(sol[i],'+');
   writeln(#8#32);
   readln;
end.
```

P79. Număr impar de 1

Să se genereze toate matricele de dimensiune m x n care au elemente din mulțimea {0, 1} și care au proprietatea că pe fiecare linie și pe fiecare coloană există un număr impar de valori de 1.

Rezolvare:

Procedura recursivă utilizată (ext) are ca parametru numărul de ordine al locației din matrice ce urmează a fi completată (i – inițial are valoarea 1). La intrarea în procedură se convertește poziția liniară (i) în poziția în cadrul matricei (`l:=(i-1) div n + 1`, respectiv `c:=(i-1) mod n + 1`).

Se verifică apoi dacă s-a completat toată matricea (`i=m*n`), caz în care s-a obținut o soluție ce se va afișa cu ajutorul procedurii afis. Dacă nu s-a completat întreaga matrice, se va încerca completarea locației curente, pe rând, cu cele două valori (`for j:=0 to 1`).

O anumită valoare poate fi depusă într-o locație dacă locația nu este situată pe ultima linie (`l<>m`) sau, dacă este situată pe ultima linie, dacă nu ar produce un număr par de valori de 1 pe coloana respectivă (`(scol[c]+j) mod 2 =1`). O condiție similară trebuie îndeplinită și referitor la coloană: fie coloana locației nu este ultima coloană (`c<>n`), fie nu s-ar produce un număr par de valori de 1 pe linia curentă (`(slin[l]+j) mod 2 =1`).

Dacă o anumită valoare poate fi depusă în locația curentă se memorează acest lucru în matricea soluție (`sol[l,c]:=j`), se actualizează numărul valorilor de 1 de pe linia și coloana curentă (`slin[l]:=slin[l]+j`, respectiv `scol[c]:=scol[c]+j`) – datorită faptului că în matrice există doar valori de 0 și 1, numărul valorilor de 1 de pe o linie sau de pe o coloană este egal cu suma elementelor de pe linia sau coloana respectivă.

În continuare se trece la completarea următoarei locații (`ext(i+1)`), la revenire refăcându-se starea anterioară prin refacerea celor două sume (`slin[l]:=slin[l]-j`, respectiv `scol[c]:=scol[c]-j`).

Programul Pascal:

```
program impar1;
uses
   crt;
var
   sol:array[1..13,1..13] of byte;
   slin,scol:array[1..13] of byte;
   m,n:1..13;
   p:byte;
   nsol:word;

procedure afis;
var
   x,y:byte;
begin
   for x:=1 to m do
      begin
         for y:=1 to n do
            write(sol[x,y]:3);
         writeln;
      end;
   writeln;
   nsol:=nsol+1;
   readln;
end;

procedure ext(i:byte);
var
   l,c,j:byte;
begin
   l:=(i-1) div n+1;
   c:=(i-1) mod n+1;
   if i>m*n then
```

```
      afis
   else
      for j:=0 to 1 do
         if ((l<>m) or ((scol[c]+j) mod 2 =1)) and
            ((c<>n) or ((slin[l]+j) mod 2 =1)) then
               begin
                  sol[l,c]:=j;
                  slin[l]:=slin[l]+j;
                  scol[c]:=scol[c]+j;
                  ext(i+1);
                  slin[l]:=slin[l]-j;
                  scol[c]:=scol[c]-j;
               end;
end;

begin
   clrscr;
   write('m=');
   readln(m);
   write('n=');
   readln(n);
   for p:=1 to m do
      slin[p]:=0;
   for p:=1 to n do
      scol[p]:=0;
   nnsol:=0
   ext(1);
   writeln('Nr. solutii: ',nsol);
   readln;
end.
```

P80. Plata unei sume

Să se determine toate posibilitățile în care se poate plăti o sumă având la dispoziție câte un număr de monede din fiecare din cale n tipuri (valori nominale).

Rezolvare:

Se utilizează o procedură recursivă (bktr) ce are ca parametri numărul de ordine al monedei curente (i – inițial are valoarea 1) și suma parțială (suma deja plătită cu cele i-1 monede anterioare – sp – inițial are valoarea 0.

Se testează mai întâi dacă s-a obținut o soluție (sp=suma), caz în care aceasta se afișează cu ajutorul procedurii afis. În caz contrar se încearcă

depunerea pe poziția i din cadrul soluției, pe rând, a câte unei monede din fiecare tip, dacă acest lucru este posibil. Un anumit tip de monedă poate ocupa poziția curentă (i) în soluție dacă utilizarea ei nu ar presupune depășirea sumei de plată (sp+val[k]<=s) și dacă mai există monede de tipul respectiv (nrm[k]>0).

În cazul în care această condiție este îndeplinită se marchează plasarea respectivului tip de monedă în vectorul soluție (sol[i]:=k), se decrementează numărul monedelor de acel tip (nrm[k]:=nrm[k]-1) și se trece la următoarea monedă cu actualizarea sumei parțiale (bktr(i+1,sp+val[k])), având grijă ca la revenire să se refacă starea (nrm[k]:=nrm[k]+1).

Programul Pascal:

```
program moned;
uses
   crt;
var
   n,k:byte;
   sol,val,nrm:array[0..100] of byte;
   suma:word;

procedure afis(p:byte);
var
   i:byte;
begin
   write(suma,'=');
   for i:=1 to p do
      write(val[sol[i]],'+');
   writeln(#8#32);
end;

procedure bktr(i:byte;sp:word);
var
   k:byte;
begin
   if sp=suma then
      afis(i-1)
   else
      for k:=sol[i-1] to n do
         if (sp+val[k]<=suma) and (nrm[k]>0) then
            begin
               sol[i]:=k;
               nrm[k]:=nrm[k]-1;
```

```
                bktr(i+1,sp+val[k]);
                nrm[k]:=nrm[k]+1;
            end;
end;

begin
   clrscr;
   write('suma=');
   readln(suma);
   write('nr. tipuri monede=');
   readln(n);
   for k:=1 to n do
      begin
         write('val[',k,']=');
         readln(val[k]);
         write('nrm[',k,']=');
         readln(nrm[k]);
      end;
   sol[0]:=1;
   bktr(1,0);
   readln;
end.
```

P81. Bara

Dintr-o bară metalică de lungime cunoscută trebuie tăiate repere de n lungimi cunoscute, astfel încât bucata rămasă să aibă lungime minimă.

Rezolvare:

Procedura recursivă utilizată (ext) are ca parametri numărul reperului ce urmează a fi tăiat (i – inițial are valoarea 1) și lungimea actuală a barei (lr – inițial are lungimea totală a barei). Pentru a se putea determina momentul în care se găsește o soluție, se utilizează o variabilă booleană locală (f) căreia i se atribuie la intrarea în procedură valoarea true.

Se va încerca, pe rând, tăierea câte unui reper din cele n (`for j:=1 to n`). Un anumit reper poate fi tăiat din bară dacă lungimea reperului nu depășește lungimea barei (`lr>=l[j]`). Dacă din bară poate fi decupat un anumit reper, se modifică valoarea variabilei booleene (`f:=false`), se memorează în vectorul soluție tipul reperului tăiat (`sol[i]:=j`) și se trece la tăierea următorului reper (`i+1`), cu actualizarea lungimii barei (`lr-l[j]`).

După terminarea for-ului se va cerceta starea variabilei f. Dacă aceasta are valoare true (valoarea inițială) înseamnă că nici un reper nu a putut fi tăiat din

bară (toate aveau lungime mai mare decât lungimea barei), și deci s-a obținut o soluție.

Soluția obținută se compară cu soluția optimă anterioară prin lungimea de bară rămasă. Dacă lungimea barei rămase este mai mică decât lungimea barei rămase la soluția optimă anterioară (`lr<difmin`) înseamnă că s-a obținut o soluție mai bună decât soluția optimă anterioară motiv pentru care se reinițializează numărul de soluții optime (`nrsol:=1`), se salvează soluția curentă în matricea soluțiilor (`solopt[nrsol]:=sol`), se actualizează lungimea de bară rămasă în cazul soluției optime (`difmin:=lr`) și se memorează lungimea soluției curente într-un vector (`lsol[nrsol]:=i-1`).

Dacă soluția nu este mai bună decât soluția optimă anterioară, se verifică dacă este la fel de bună (`lr=difmin`). În cazul în care soluția este la fel de bună ca soluția optimă anterioară se incrementează numărul de soluții optime (`nrsol:=nrsol+1`), se salvează soluția curentă în matricea soluțiilor optime (`solopt[nrsol]:=sol`) și se reține lungimea soluției curente (`lsol[nrsol]:=i-1`).

Soluțiile vor fi afișate în programul principal, după apelul procedurii recursive, prin explorarea matricei soluțiilor optime, folosindu-ne de vectorul ce memorează lungimile soluțiilor optime (lsol).

Programul Pascal:

```
program bara;
uses
   crt;
type
   sir=array[1..20]of byte;
var
   m,n,c,d,nrsol:byte;
   l,sol,lsol:sir;
   solopt:array[1..100] of sir;
   lb,ltot:word;
   difmin:byte;

procedure ext(i:byte;lr:word);
var
   j:byte;
   f:boolean;
begin
   f:=true;
   for j:=1 to n do
      if lr>=l[j] then
         begin
            f:=false;
```

```
            sol[i]:=j;
            ext(i+1,lr-l[j]);
          end;
    if f then
      if (lr<difmin) then
        begin
          nrsol:=1;
          solopt[nrsol]:=sol;
          difmin:=lr;
          lsol[nrsol]:=i-1;
        end
      else
        if (lr=difmin) then
          begin
            nrsol:=nrsol+1;
            solopt[nrsol]:=sol;
            lsol[nrsol]:=i-1;
          end;
end;

begin
  clrscr;
  write('lungime bara: ');
  readln(lb);
  write('n=');
  readln(n);
  for c:=1 to n do
    begin
      write('l[',c,']=');
      readln(l[c]);
    end;
  difmin:=l[1];
  nrsol:=0;
  ext(1,lb);
  for c:=1 to nrsol do
    begin
      for d:=1 to lsol[c] do
        write(solopt[c,d]:3);
      writeln;
    end;
  writeln('Rebut: ',difmin);
  write('Nr. de solutii: ',nrsol);
  readln;
```



```
end.
```

P82. Numere cu cifrele unei baze

Fie b un număr natural dat (1<b<10). Să se genereze și să se afișeze toate numerele ce se pot forma în baza b și care utilizează în scrierea lor cifrele bazei b cel mult o dată.

Rezolvare:

Procedura recursivă folosită (num) are ca parametru numărul de cifre ale numărului. Această procedură se apelează în programul principal în cadrul unui for în care contorul (k) ia valori între 1 (numărul minim de cifre ale unui număr) și b (numărul maxim de cifre distincte ale unui număr în baza b).

În cadrul procedurii se verifică mai întâi dacă s-a obținut un număr de k cifre, caz în care acesta se afișează cu ajutorul procedurii afis. În caz contrar se încearcă depunerea, pe rând, a tuturor cifrelor bazei b (de la 0 la b-1). O anumită cifră (c) poate fi depusă dacă aceasta nu a fost depusă anterior (`not depus[c]`) și dacă aceasta nu este un 0 nesemnificativ (`(k=1) or (i>1) or (c>0)`).

În cazul în care cifra poate fi depusă se marchează depunerea ei (`depus[c]:=true`), se memorează depunerea în vectorul soluție (sol[i]:=c), și se trece la următoarea cifră (`num(i)+1`), având grijă ca la revenire să se demarcheze utilizarea respectivei cifre (`depus[c]:=false`).

Programul Pascal:

```
program numere;
uses
  crt;
type
  sir=array[1..10] of byte;
var
  b,k:byte;
  sol:sir;
  depus:array[0..9] of boolean;

procedure afis;
var
  i:byte;
begin
  for i:=1 to k do
    write(sol[i]);
  writeln;
end;
```

```
procedure num(i:byte);
var
  c:byte;
begin
  if i>k then
    afis
  else
    for c:=0 to b-1 do
      if (not depus[c]) then
        if (k=1)or (i>1) or (c>0)then
          begin
            depus[c]:=true;
            sol[i]:=c;
            num(i+1);
            depus[c]:=false;
          end;
end;

begin
  clrscr;
  write('b=');
  readln(b);
  for k:=0 to b-1 do
    depus[k]:=false;
  for k:=1 to b do
    num(1);
  readln;
end.
```

P83. Numere divizibile cu 2^n

Să se genereze toate numerele de n cifre care se pot forma utilizând doar cifre 1 și 2 și care sunt divizibile cu 2^n.

Rezolvare:

Pentru calculul lui 2^n se utilizează o funcție (putere), iar valoarea se păstrează în variabila doilan. Procedura recursivă utilizată (ext) are ca parametri numărul de ordine al cifrei ce urmează a se stabili (ncif – inițial are valoarea 1) și numărul parțial format din cele ncif-1 cifre anterioare (nrp – inițial are valoarea 0).

Mai întâi se testează dacă s-au completat toate cele n cifre (`ncif>n`), caz în care, dacă numărul este divizibil cu doilan, acesta se afișează.

Dacă nu s-au completat toate cifrele, se depun, pe rând, ca cifră curentă, cele două valori – 1 și 2 – (`for c:=1 to 2`) și se trece la cifra următoare cu depunerea cifrei curente în numărul parțial (`ext(ncif+1,nrp*10+c)`).

Programul Pascal:

```
program div2lan;
uses
  crt;
var
  n:byte;
  doilan:longint;

function putere(b,e:byte):longint;
begin
  if e=0 then
    putere:=1
  else
    putere:=putere(b,e-1)*b;
end;

procedure ext(ncif:byte;nrp:longint);
var
  c:byte;
begin
  if ncif>n then
    begin
      if nrp mod doilan =0then
        writeln(nrp);
    end
  else
    for c:=1 to 2 do
      ext(ncif+1,nrp*10+c);
end;

begin
  clrscr;
  write('n=');
  readln(n);
  doilan:=putere(2,n);
  ext(1,0);
  readln;
end.
```

P84. Sesiune

Un student are de susținut un număr de n examene. El promovează dacă acumulează cel puțin m puncte. Care sunt posibilitățile de reușită?

Rezolvare:

Procedura recursivă utilizată (examen) are ca parametri numărul examenului pe care studentul urmează să-l susțină (i – inițial are valoarea 1) și numărul de puncte acumulate la examenele anterioare (inițial 0). Se verifică mai întâi dacă s-au terminat examenele (i=n+1), caz în care se afișează soluția obținută, cu ajutorul procedurii afis.

În cazul în care mai sunt examene, se dau toate valorile posibile (de la 10 la 1) notei de la examenul i, dacă acest lucru este posibil. La un anumit examen, i, studentul poate lua un anumită notă, c, doar dacă mai poate atinge punctajul de promovare (dacă luând nota maximă la cele n-i examene rămase poate ajunge la punctajul total m cerut (if p+c+10*(n-i)>=m).

În cazul în care studentul poate primi nota respectivă, se reține nota în vectorul soluție și se trece la următorul examen, cu creșterea punctajului cu valoare notei obținute (examen(i+1,p+c)).

Programul Pascal:

```
program studentu;
uses
  crt;
var
  n,m:byte;
  sol:array[1..100] of byte;

procedure  afis(pt:byte);
var
  x:byte;
begin
  for x:=1 to n do
    write (sol[x],' ');
   writeln;
   writeln ('punctaj = ',pt);
end;

procedure examen(i,p:byte);
var
  c:byte;
begin
  if i=n+1 then
    afis(p)
```



```
  else
    for c:=10 downto 1 do
      if p+c+10*(n-i)>=m then
        begin
          sol[i]:=c;
          examen(i+1,p+c);
        end;
end;

begin
  clrscr;
  write('Nr. de examene: ');
  readln(n);
  write('Nr. minim de puncte: ');
  readln(m);
  examen(1,0);
  readln;
end.
```

P85. Obezii

Un număr de n persoane, de greutăți diferite, merg la restaurant. La restaurant sunt disponibile un număr de m (m>n) meniuri, fiecare meniu având un anumit număr de calorii. Să se determine toate posibilitățile în care persoanele își pot alege meniurile, astfel încât să nu existe o persoană cu greutate mai mare decât alta și care să consume un meniu mai bogat în calorii decât meniul celeilalte persoane.

Rezolvare:

Procedura recursivă folosită (ext) are ca parametru numărul de ordine al persoanei ce își alege meniul (i – inițial are valoarea 1). Se testează mai întâi dacă toate persoanele și-au ales meniul (i=n+1), caz în care s-a obținut o soluție, ce va fi afișată cu ajutorul procedurii afis.

În cazul în care mai există persoane ce trebuie să-și aleagă meniul, persoana curentă va primi, pe rând, fiecare meniu, dacă acest lucru este posibil. O anumită persoană (i) poate primi un meniu (c) dacă acesta nu a fost dat altei persoane (not dat[c]) și dacă acesta respectă regula din enunț.

Pentru creșterea lizibilității, la implementarea condiției ca fiecare persoană cu greutate mai mare decât alta să consume un meniu mai slab în calorii decât al acesteia, s-a utilizat o funcție booleană (sepoate) ce verifică respectarea relației între persoana curentă (c) și toate persoanele ce și-au ales anterior meniul (de la 1 la i-1).

În cazul în care o persoană poate primi un anumit meniu, se reține meniul primit în vectorul soluție (sol[i]:=c), se marchează meniul ca fiind dat (dat[c]:=true) și se trece la următoarea persoană (ext(i+1)), având grijă ca la revenire să se elibereze meniul respectiv (dat[c]:=false), pentru a putea fi dat altei persoane.

Programul Pascal:

```
program restaurant;
uses
   crt;
type
   sir=array[1..20] of word;
var
   n,m,x:byte;
   g,cal,sol:sir;
   dat:array[1..20] of boolean;

function sepoate(i,c:byte):boolean;
var
   k:byte;
   f:boolean;
begin
   f:=true;
   for k:=1 to i-1  do
      if ((g[i]>g[k]) and (cal[c]>cal[sol[k]])) or
         ((g[i]<g[k]) and (cal[c]<cal[sol[k]])) then
         f:=false;
   sepoate:=f;
end;

procedure afis;
var
   i:byte;
begin
   for i:=1 to n do
      write(cal[sol[i]],' ');
   writeln;
end;

procedure ext(i:byte);
var
   c:byte;
begin
```



```
   if i=n+1 then
      afis
   else
      for c:=1 to m do
         if not dat[c] and sepoate(i,c) then
            begin
               sol[i]:=c;
               dat[c]:=true;
               ext(i+1);
               dat[c]:=false;
            end;
end;

begin
   clrscr;
   write('n=');
   readln(n);
   write('m=');
   readln(m);
   for x:=1 to n do
      begin
         write(x,' g=');
         readln(g[x]);
      end;
   for x:=1 to m do
      begin
         write(x,' cal=');
         readln(cal[x]);
      end;
   ext(1);
   readln;
end.
```

P86. Colaboratori

Cunoscându-se relațiile de tipul *"x a colaborat în trecut cu y"* între fiecare din cei 2n angajați ai unei firme, să se genereze n echipe de câte doi angajați, astfel încât numărul echipelor în care lucrează persoane ce nu au colaborat în trecut să fie minim.

Rezolvare:

Relațiile de colaborare dintre cele 2n persoane se reține într-o matrice pătratică (n x n), simetrică față de diagonala principală (dacă persoana i a colaborat cu persoana j atunci și persoana j a colaborat cu persoana i).

Procedura recursivă utilizată (extindere) va încerca plasarea fiecărei persoane în fiecare din cele n echipe. Aceasta primește ca parametri numărul de ordine al persoanei ce urmează să fie plasată într-o echipă (i – inițial are valoarea 1), numărul echipelor complete formate din persoane care nu au mai colaborat (nenc – inițial are valoarea 0).

Se verifică, mai întâi, dacă toate persoanele au fost plasate în echipe (`i=2*n+1`), caz în care înseamnă că s-a obținut o nouă soluție optimă și se înlocuiește soluția optimă precedentă (`solopt:=sol`) precum și numărul de echipe formate din persoane care nu au colaborat (`nencopt:=nenc`).

În continuare, se încearcă plasarea persoanei i în fiecare din cele n echipe (`for k:=1 to n`). O persoană poate fi plasată într-o echipă dacă echipa nu este completă. În vectorul ec se reține numărul de persoane din fiecare echipă (inițial toate elementele șirului au valoarea 0).

Se verifică dacă echipa curentă este formată din 0 persoane (`ec[k]=0`), caz în care se mărește numărul de membri ai echipei curente (`ec[k]:=1`), se marchează apartenența persoanei la echipă (`sol[1,k]:=i`) și se trece la persoana următoare cu păstrarea numărului de echipe de persoane care nu au mai colaborat deoarece nu s-a completat încă o echipă (`extindere(i+1, nenc)`). După apelul recursiv se va reface starea anterioară (`ec[k]:=0`).

În cazul în care persoana curentă ar fi a doua în echipă (`ec[k]=1`), sunt două situații posibile: persoana curentă să fi colaborat sau să nu fi colaborat în trecut cu persoana existentă deja în echipa curentă. Dacă în trecut a existat colaborare între cele două persoane (`a[sol[1,k],i]=1`) atunci se poate plasa persoana curentă în echipă, efectuându-se operațiile de actualizare evidențiate mai sus.

Dacă persoanele nu au mai colaborat, persoana curentă nu va putea fi plasată în echipa curentă decât dacă astfel s-ar ajunge la un număr de echipe formate din persoane ce nu au mai colaborat egal mai mic decât cel corespunzător soluției optime din acel moment (`nenc+1<nencopt`).

De asemenea, și în acest caz se vor face actualizările necesare având grijă ca după apelul recursiv să se refacă starea anterioară.

Programul Pascal:

```
program colaboratori;
uses
   crt;
type
   solutie=array[1..2,1..20]of byte;
```



```
var
   nencopt,n,i,j:byte;
   sol,solopt:solutie;
   ec:array[1..20]of byte;
   a:array[1..40,1..40]of 0..1;

procedure extindere(i,nenc:byte);
var
   k:byte;
begin
   if i=2*n+1 then
      begin
         solopt:=sol;
         nencopt:=nenc;
      end
   else
      for k:=1 to n do
         if ec[k]=0 then
            begin
               ec[k]:=1;
               sol[1,k]:=i;
               extindere(i+1,nenc);
               ec[k]:=0;
            end
         else
            if ec[k]=1 then
               if a[sol[1,k],i]=1 then
                  begin
                     ec[k]:=2;
                     sol[2,k]:=i;
                     extindere(i+1,nenc);
                     ec[k]:=1;
                  end
               else
                  if nenc+1<nencopt then
                     begin
                        ec[k]:=2;
                        sol[2,k]:=i;
                        extindere(i+1,nenc+1);
                        ec[k]:=1;
                     end;
end;
```

```
begin
   clrscr;
   write('n=');
   readln(n);
   for i:=1 to 2*n-1 do
      for j:=i+1 to 2*n do
         begin
            write('a[',i,',',j,']=');
            readln(a[i,j]);
            a[j,i]:=a[i,j];
         end;
   for i:=1 to n do
      ec[i]:=0;
   nencopt:=n;
   extindere(1,0);
   for i:=1 to n do
      writeln('Echipa ',i,': ',solopt[1,i],
         ' si ',solopt[2,i]);
   writeln('Numrul de echipe de ne fosti
      colaboratori: ',nencopt);
   readln;
end.
```

P87. Săritura broaştei

Pe un lac de dimensiune m x n se află, în fiecare locație se află câte o frunză iar în colțul de nord-vest, pe o piatră, se află o broască. Broasca trebuie să se deplaseze, prin sărituri similare cu cele ale unui cal pe tabla de șah, în colțul opus (m,n) și să revină în punctul de plecare, fără să treacă de două ori prin același punct (la trecere frunzele se scufundă). Se cer toate soluțiile cu număr minim de sărituri.

Rezolvare:

Procedura de extindere a soluției (ext) primește ca parametri numărul săriturii (i – inițial are valoarea 1) și coordonatele locației curente (posx și posy – inițial au valori 1 și 1).

Inițial se marchează locația curentă cu numărul săriturii (`lac[posx, posy]:=i`).

Se verifică apoi dacă s-a obținut o soluție (dacă s-a trecut prin colțul opus (`lac[m,n]>0`) și s-a ajuns în punctul de plecare (`(posx=1) and (posy= 1)`), caz în care se evaluează soluția obținută cu ajutorul procedurii testopt.

Procedura testopt verifică dacă soluția curentă este mai bună decât soluția optimă precedentă (`j<pasopt`), caz în care se actualizează numărul de pași pentru soluția optimă (`pasopt:=j`), se reinițializează numărul de soluții optime (`nrsol:=1`) și se salvează soluția obținută (`sol[nrsol]:=lac`).

În caz contrar, dacă s-a obținut o soluție la fel de bună ca soluția optimă precedentă (`j=pasopt`), se incrementează numărul de soluții optime (`inc(nrsol)`) și se memorează soluția curentă (`sol[nrsol]:=lac`).

În cazul în care nu s-a obținut o soluție se încearcă, pe rând, deplasarea în toate cele opt direcții posibile (cu deplasările pe cele două direcții date de valorile memorate în șirurile x și y). Dacă deplasarea într-o anumită direcție este posibilă (condiție verificată în cadrul funcției cond ce verifică dacă locația de destinație este pe lac și nu a fost anterior vizitată), se trece în locația respectivă (`ext(i+1,posx+x[j],posy+y[j])`).

La sfârșitul procedurii se va demarca, pentru revenire, locația curentă (`lac[posx,posy]:=-1`).

În cadrul programului principal se vor face citirile datelor de intrare și inițializările necesare. Variabila ce reține numărul de sărituri ale soluției optime (pasopt) se inițializează cu valoarea maximă (m x n). Pentru memorarea faptului că s-a găsit sau nu o soluție se utilizează variabila booleană gasit.

Programul Pascal:

```
program broasca;
uses
   crt;
type
   mat=array[1..10,1..10]of shortint;
const
   x:array[1..8]of shortint=(-1,1,2,2,1,-1,-2,-2);
   y:array[1..8]of shortint=(-2,-2,-1,1,2,2,1,-1);
var
   lac:mat;
   sol:array[1..30] of mat;
   cont1,cont2,m,n,pasopt,nrsol:byte;
   gasit:boolean;

function cond(j,posx,posy:byte):boolean;
begin
   if (posx+x[j]<=m) and (posx+x[j]>=1) and
         (posy+y[j]<=n) and (posy+y[j]>=1) and
         (lac[posx+x[j],posy+y[j]]=-1) then
      cond:=true
   else
      cond:=false;
```

```
end;

procedure afis;
var
  k,i,j:byte;
begin
  for k:=1 to nrsol do
    begin
      for i:=1 to m do
        begin
          for j:=1 to n do
            write(sol[k,i,j]:6);
          writeln;
        end;
      writeln;
    end;
end;

procedure testopt(j:byte);
begin
  gasit:=true;
  if j<pasopt then
    begin
      pasopt:=j;
      nrsol:=1;
      sol[nrsol]:=lac;
    end
  else
    if j=pasopt then
      begin
        inc(nrsol);
        sol[nrsol]:=lac;
      end;
end;

procedure ext(i,posx,posy:byte);
var
  j:byte;
begin
  lac[posx,posy]:=i;
  if (lac[m,n]>0) and (posx=1) and (posy=1) then
    testopt(i)
  else
```



```
    if i+1<=pasopt then
      for j:=1 to 8 do
        if cond(j,posx,posy) then
          ext(i+1,posx+x[j],posy+y[j]);
  lac[posx,posy]:=-1;
end;

begin
  clrscr;
  write('n=');
  readln(n);
  write('m=');
  readln(m);
  gasit:=false;
  pasopt:=m*n;
  for cont1:=1 to m do
    for cont2:=1 to n do
      lac[cont1,cont2]:=-1;
  if m>2 then
    ext(2,3,2);
  if gasit then
    afis
  else
    writeln('Nu exista solutie!!!');
  readln;
end.
```

P88. Câmpul minat

Un câmp minat de suprafață m x n trebuie parcurs de către un soldat din colțul de nord-vest către colțul de sud-est. Cunoscând poziția minelor, să se determine toate posibilitățile de parcurgere a câmpului.

Rezolvare:

Metoda de rezolvare este asemănătoare celei utilizată la problema P73. Poziția minelor se reține în matricea t. Procedura recursivă folosită (ext) primește ca parametri linia (l) și coloana (c) a locației curente, precum și numărul de ordine al locației curente în cadrul traseului (i – inițial are valoarea 1), care se utilizează și la marcarea traseului.

La intrarea în procedură se marchează cu litera corespunzătoare locația curentă (`t[l,c]:=chr(ord('A')+i-1)`). Se verifică apoi dacă s-a ajuns în colțul de sud-est (`(l=m) and (c=n)`), caz în care s-a obținut o soluție ce se afișează cu ajutorul procedurii afis.

În cazul în care nu s-a ajuns la destinație se încearcă deplasarea, pe rând, în fiecare din cele patru direcții, dacă acest lucru este posibil. Soldatul se poate deplasa într-o locație vecină dacă locația respectivă este în suprafața dreptunghiulară dată și dacă este marcată cu `'liber'` (nu conține mină și nu a fost vizitată anterior).

Dacă este posibilă deplasarea într-o locație vecină se realizează apelul recursiv cu actualizarea liniei, coloanei și a numărului de pași. La ieșirea din procedură se va demarca locația curentă (se va reface starea existentă la intrarea în procedură).

Programul Pascal:

```
program teren_minat;
uses
   crt;
const
   mina='°';
   liber=' ';
var
   i,j,m,n,nrsol:byte;
   t:array[1..20,1..20] of char;
   rasp:char;

procedure afis;
var
   p,q:byte;
begin
   for p:=1 to m do
     begin
        for q:=1 to n do
          write(t[p,q]);
        writeln;
     end;
   writeln;
   nrsol:=nrsol+1;
   readln;
end;

procedure ext(l,c,i:byte);
begin
   t[l,c]:=chr(ord('A')+i-1);
   if (l=m) and (c=n) then
     afis
   else
```



```
     begin
        if (l>1) and (t[l-1,c]=liber) then
             ext(l-1,c,i+1);
        if (l<m) and (t[l+1,c]=liber) then
             ext(l+1,c,i+1);
        if (c<n) and (t[l,c+1]=liber) then
             ext(l,c+1,i+1);
        if (c>1) and (t[l,c-1]=liber) then
             ext(l,c-1,i+1);
     end;
   t[l,c]:=liber;
end;

begin
   clrscr;
   write('m=');
   readln(m);
   write('n=');
   readln(n);
   for i:=1 to m do
     for j:=1 to n do
        begin
          write('t[',i,',',j,']=mina (d/n)');
          rasp:=readkey;
          if upcase(rasp)='D' then
            t[i,j]:=mina
          else
            t[i,j]:=liber;
          writeln;
        end;
   ext(1,1,1);
   writeln('Nr. solutii: ',nrsol);
   readln;
end.
```

P89. La curtea regelui Arthur

Regele Arthur trebuie să-i așeze la o masă rotundă pe cei de n cavaleri ce se află în vizită la acesta. Care sunt posibilitățile, știind că fiecare cavaler are dușmani printre ceilalți cavaleri, astfel încât nici unul dintre cavaleri să nu stea la masă alături de un dușman al său?

Rezolvare:

Procedura recursivă utilizată (ext) are ca parametru poziția locului la masă (i), care, pentru evitarea soluțiilor în oglindă și a celor obținute prin rotirea cavalerilor la masă, are ca valoare inițială 2 (se consideră, în programul principal, că pe locul numărul 1 stă cavalerul cu numărul 1 (`sol[1]:=1`).

La intrarea în procedură se verifică dacă s-a obținut o soluție (`i>n`), caz în care aceasta se va afișa cu ajutorul procedurii afis.

În caz contrar, se va încerca așezarea pe locul curent (i) a tuturor cavalerilor (cu excepția cavalerului numărul 1 care stă pe locul cu numărul 1), dacă acest lucru este posibil.

Un anumit cavaler (c) poate ocupa un anumit loc (i) numai dacă acesta nu a ocupat anterior alt loc (`not pus[c]`), dacă acesta nu este ultimul (`i<>n`) sau nu este dușman cu primul cavaler de la masă (`not d[sol[1],c]`), nu este ultimul (`i<>n`) sau are numărul de ordine mai mare decât al celui ce ocupă poziția numărul 2 la masă (`sol[2]<c`) – (pentru evitarea soluțiilor cu aceiași cavaleri așezați în ordine inversă), și dacă nu este dușman cu cavalerul ce ocupă poziția precedentă la masă (`not d[sol[i-1],c]`).

În cazul în care cavalerul poate ocupa locul respectiv se marchează faptul că respectivul cavaler a fost așezat la masă (`pus[c]:=true`), se memorează în vectorul soluție așezarea respectivului cavaler pe acea poziție (`sol[i]:=c`) și se trece la completarea următorului loc de la masă (`ext(i+1)`), având grijă ca la revenire să se demarcheze așezarea cavalerului respectiv la masă (`pus[c]:=false`).

Programul Pascal:

```
program Arthur;
uses
  crt;
var
  sol:array[1..20] of byte;
  d:array[1..20,1..20] of boolean;
  pus:array[1..20] of boolean;
  n,i,j:byte;
  rasp:char;

procedure afis;
var
  j:byte;
begin
  for j:=1 to n do
    write(sol[j]:3);
  writeln;
```



```
end;

procedure ext(i:byte);
var
  c:byte;
begin
  if i>n then
    afis
  else
    for c:=2 to n do
      if not pus[c] and ((i<>n) or
           (not d[sol[1],c])) and((i<>n)
           or (sol[2]<c)) then
        if not d[sol[i-1],c] then
          begin
            pus[c]:=true;
            sol[i]:=c;
            ext(i+1);
            pus[c]:=false;
          end;
end;

begin
  clrscr;
  write('n=');
  readln(n);
  for i:=1 to n do
    begin
      for j:=1 to n do
        d[i,j]:=false;
      pus[i]:=false;
    end;
  for i:=1 to n-1 do
    for j:=i+1 to n do
      begin
        write(i,' este dusman cu ',j,'? (d/n)');
        rasp:=readkey;
        writeln;
        if upcase(rasp)='D' then
          d[i,j]:=true;
      end;
  sol[1]:=1;
  ext(2);
```

```
  readln;
end.
```

P90. Sumă de numere consecutive

Să se afișeze toate posibilitățile de scriere a unui număr natural dat, n, ca o sumă de numere consecutive.

Rezolvare:

Procedura recursivă utilizată (sum) are ca parametri numărul de ordine al termenului curent din sumă (nt – inițial are valoarea 1), suma parțială (sp – inițial are valoarea 0) și termenul curent al sumei (x – inițial are valori cuprinse între 1 și n div 2). Procedura se apelează într-un for de la 1 la n div 2, contorul fiind termenul de început al sumei.

Atribuirea ca valoare a primului termen al sumei toate numerele de la 1 la n div 2 asigură găsirea tuturor soluțiilor de descompunere a numărului dat într-o sumă de (minim 2) numere consecutive.

La intrarea în procedură se verifică dacă s-a obținut o soluție (dacă suma parțială este egală cu numărul inițial – sp=n), caz în care aceasta se afișează cu ajutorul procedurii afis.

Dacă nu s-a obținut încă o soluție, dacă astfel nu s-ar depăși numărul (sp+c<=n) , se memorează termenul curent în vectorul soluție (sol[nt]:= c) și se trece la următorul termen al sumei (nt+1), cu actualizarea sumei parțiale (sp+c) și a valorii termenului (c+1).

Programul Pascal:

```
program sum_consec;
uses
   crt;
var
   n,x:longint;
   sol:array[1..1000] of longint;
   nrsol:word;

procedure afis(l:word);
var
   j:word;
begin
   for j:=1 to l do
      write(sol[j],'+');
   writeln(#8#32);
   writeln;
   nrsol:=nrsol+1;
```



```
end;

procedure sum(nt:word;sp,c:longint);
begin
   if sp=n then
      afis(nt-1)
   else
      if sp+c<=n then
         begin
            sol[nt]:=c;
            sum(nt+1,sp+c,c+1);
         end;
end;

begin
   clrscr;
   write('n=');
   readln(n);
   nrsol:=0;
   for x:=1 to n div 2 do
      sum(1,0,x);
   writeln('Nr. solutii: ',nrsol);
   readln;
end.
```

P91. Cost angajări

Se dau n locuri de muncă și n persoane. Cunoscându-se costul angajării fiecărei persoane la fiecare din cele n locuri de muncă, să se determine pe ce post va fi angajată fiecare persoană, astfel încât costul total al angajărilor să fie minim.

Rezolvare:

Procedura recursivă utilizată (ext) primește ca parametri numărul de ordine al persoanei ce urmează să fie angajate (i – inițial are valoarea 1) și suma cheltuită anterior (cp – inițial are valoarea 0).

Se verifică, mai întâi, dacă s-a obținut o soluție (toate persoanele au fost angajate – i>n), caz în care se înlocuiește soluția optimă curentă (solopt:= sol) și se actualizează cheltuiala totală aferentă soluției optime (copt:=cp). Din condiția de acceptare a unei anumite angajări se asigură faptul că soluția obținută este mai bună decât soluția optimă anterioară.

Dacă nu s-a obținut, încă, o soluție, se va încerca, pe rând, plasarea persoanei curente (i) în fiecare din cele n posturi (for x:=1 to n), dacă acest lucru este posibil. O persoană poate ocupa un anumit loc de muncă dacă

acesta nu a fost ocupat anterior (not ocupat[x]) și dacă astfel s-ar tinde către o soluție cu cheltuială totală mai mică decât suma cheltuită în cazul soluției optime curente (cp+c[i,x]<copt).

În cazul în care persoana poate ocupa respectivul loc de muncă se marchează postul respectiv ca fiind ocupat (ocupat[x]:=true), se memorează angajarea persoanei pe postul respectiv în vectorul soluție (sol[i]:=x) și se trece la următoarea persoană cu actualizarea corespunzătoare a costului parțial al angajării (ext(i+1,cp+c[i,j])), având grijă ca la revenire să se elibereze respectivul loc de muncă (ocupat[x]:=false).

Soluția optimă (cu cost total minim) generată de către procedura ext și costul total al angajărilor se vor afișa în programul principal, după apelul procedurii recursive.

Programul Pascal:

```
program cost_angajare;
uses
  crt;
type
  sir=array[1..20] of byte;
var
  n,i,j:byte;
  c:array[1..20,1..20] of word;
  ocupat:array[1..20] of boolean;
  sol,solopt:sir;
  copt:longint;

procedure ext(i:byte;cp:longint);
var
  x:byte;
begin
  if i>n then
    begin
      solopt:=sol;
      copt:=cp;
    end
  else
    for x:=1 to n do
      if (not ocupat[x]) and (cp+c[i,x]<copt) then
        begin
          ocupat[x]:=true;
          sol[i]:=x;
          ext(i+1,cp+c[i,x]);
```



```
          ocupat[x]:=false;
        end;
end;

begin
  clrscr;
  write('n=');
  readln(n);
  copt:=0;
  for i:=1 to n do
    for j:=1 to n do
      begin
        write('Costul angajarii persoanei '
          ,i,' n postul ',j,': ');
        readln(c[i,j]);
        copt:=copt+c[i,j];
      end;
  for i:=1 to n do
    ocupat[i]:=false;
  ext(1,0);
  for i:=1 to n do
    write(solopt[i]:3);
  writeln;
  writeln('costul total: ',copt);
  readln;
end.
```

P92. Domino

Fiind date n piese de domino, să se afișeze toate modalitățile corecte de formare a unui lanț cu cele n piese.

Rezolvare:

Piesele de domino conțin câte două valori de la 1 la 6. Ele pot fi poziționate în două moduri. Un lanț se consideră corect dacă valorile alăturate din oricare două piese vecine sunt identice.

Pentru reținere pieselor de domino s-a utilizat un șir cu elemente de tip record (pdom - având drept componente cele două valori – v1 și v2). Pentru întoarcerea unei piese (interschimbul celor două valori) s-a utilizat o procedură (inter).

Procedura recursivă utilizată (ext) primește ca parametrul numărul de ordine al piesei (poziția în lanț) ce urmează a fi plasate (i – inițial are valoarea 1). Se

verifică mai întâi dacă s-a completat lanțul (i>n), caz în care s-a obținut o soluție ce se va afișa cu ajutorul procedurii afis.

În cazul în care nu s-au depus toate piesele în lanț, se va încerca, pe rând, plasarea pe poziția curentă a tuturor pieselor (for j:=1 to n). Testarea fiecărei piese dacă poate ocupa poziția curentă se face într-un for de la 1 la 2, pentru fiecare din pozițiile în care se poate depune piesa respectivă. La fiecare pas al acestui for piesa va fi întoarsă.

O anumită piesă, j, (într-una din cele două poziții) poate fi plasată pe poziția curentă, i, dacă piesa nu a fost depusă anterior (not depus[j]) și dacă piesa este prima din lanț (i=1), sau, dacă nu este prima, dacă prima valoare a acestei piese este egală cu valoarea a doua a piesei precedente (a[j].v1=a[sol[i-1]].v2).

Dacă piesa curentă poate fi depusă se marchează acea piesă ca fiind folosită (depus[j]:=true), se marchează alegerea făcută prin memorarea în vectorul soluție a numărului de ordine al piesei depuse (sol[i]:=j), și se trece la următoarea poziție din lanț (ext(i+1)). La revenire se va elibera piesa (depus[j]:=false), pentru ca aceasta să poată fi depusă ulterior pe altă poziție în cadrul lanțului.

Procramul Pascal:

```
program domino;
uses
   crt;
type
   pdom=record
      v1,v2:1..6;
    end;
var
   a:array[1..30] of pdom;
   sol:array[1..30] of byte;
   depus:array[1..30] of boolean;
   n,c:byte;
   nrsol:word;

procedure afis;
var
   x:byte;
begin
   for x:=1 to n do
      write(a[sol[x]].v1,a[sol[x]].v2,' ');
   writeln;
   nrsol:=nrsol+1;
end;
```



```
procedure inter(var pp:byte);
var
   aux:1..6;
begin
   aux:=a[pp].v1;
   a[pp].v1:=a[pp].v2;
   a[pp].v2:=aux;
end;

procedure ext(i:byte);
var
   j,k:byte;
begin
  if i=n+1 then
     afis
   else
     for j:=1 to n do
       begin
          for k:=1 to 2 do
             begin
                if (not depus[j]) then
                   if (i=1) or (a[j].v1=a[sol[i-1]].v2)
                   then
                      begin
                         depus[j]:=true;
                         sol[i]:=j;
                         ext(i+1);
                         depus[j]:=false;
                      end;
                inter(j);
             end;
       end;
end;

begin
   clrscr;
   write('n=');
   readln(n);
   for c:=1 to n do
     begin
        writeln('Piesa ',c,': ');
        write('v1: ');
```

```
      readln(a[c].v1);
      write('v2: ');
      readln(a[c].v2);
      writeln;
      depus[c]:=false;
    end;
  nrsol:=0;
  ext(1);
  writeln('Nr. solutii: ',nrsol);
  readln;
end.
```

P93. Pătrat magic

Să se găsească toate soluțiile de umplere a unui pătrat de dimensiune nxn (n număr impar) cu numerele 1..n^2, astfel încât suma pe fiecare linie, coloană și pe cele două diagonale să fie aceeași.

Rezolvare:

Se poate observa că sumele pe linii, coloane și diagonale trebuie să fie n*((n^2+1) div 2) (adică de n ori valoarea medie între cele două extreme – 1 și n^2). Se mai poate observa, de asemenea, că toate soluțiile au în centru valoarea medie a extremelor ((n^2+1) div 2).

Procedura recursivă utilizată (ext) are drept parametru numărul de ordine al locației din matrice ce urmează a fi completat (i – inițial are valoarea 1). La intrarea în procedură se va converti poziția (liniară) în poziție matriceală (linie și coloană – l și c).

În continuare se verifică dacă s-a completat toată matricea (`i>n*n`), iar în caz afirmativ se verifică dacă soluția obținută este corectă (dacă suma pe diagonala principală – sdp – și suma pe diagonala secundară – sds – sunt egale cu suma calculată anterior - nrm). În cazul în care soluția este corectă, aceasta se afișează cu ajutorul procedurii afis.

În cazul în care nu s-a obținut încă o soluție, se va încerca, pe rând, plasarea în locația curentă a tuturor valorilor (`for j:=1 to n*n`). O anumită valoare, j, poate fi plasată pe linia l și coloana c dacă aceasta nu a fost plasată anterior pe o altă poziție (`not depus[j]`), dacă nu s-ar depăși suma țintă pe linia curentă (`slin[l]+j<=nrm`) și dacă linia nu este ultima (`l<>n`) sau suma obținută este chiar suma dorită (`slin[l]+j=nrm`).

Aceleași condiții trebuie îndeplinite și de coloană: suma pe coloana curentă să nu depășească suma țintă (`scol[c]+j<=nrm`) și coloana să nu fie ultima (`c<>n`) sau, dacă linia este ultima, suma să fie chiar cea căutată (`scol[c]+j=nrm`). De asemenea, trebuie pusă condiția că fie locația curentă

nu se este centrul matricei (`(l<>n div 2 +1) or (c<> n div 2 +1)`), fie numărul este chiar cel ce trebuie să fie în mijloc (`j=((n*n+1) div 2`).

Dacă numărul j poate fi depus în acea locație, se va marca acest lucru în matricea soluție (`sol[l,c]:=j`), se va marca valoarea curentă ca fiind folosită (`depus[j]:=true`), se vor actualiza sumele pe linia curentă (`slin[l]:=slin[l]+j`), pe coloana curentă (`scol[c]:=scol[c]+j`) și – dacă este cazul – pe diagonale, după care se va trece la locația următoare (`ext(i+1)`). La revenire se va demarca valoarea curentă pentru a putea fi folosită ulterior într-o altă locație (`depus[j]:=false`) și se va reface starea existentă înainte de actualizările menționate mai sus.

Programul Pascal:

```
program patrat_magic;
uses
  crt;
var
  sol:array[1..13,1..13] of byte;
  depus:array[1..169] of boolean;
  slin,scol:array[1..13] of byte;
  sdp,sds:byte;
  n:1..13;
  p:byte;
  nsol,nrm:word;

procedure afis;
var
  x,y:byte;
begin
  for x:=1 to n do
    begin
      for y:=1 to n do
        write(sol[x,y]:4);
      writeln;
    end;
  writeln;
  nsol:=nsol+1;
  readln;
end;

procedure ext(i:byte);
var
  l,c,j:byte;
```

```
begin
   l:=(i-1) div n+1;
   c:=(i-1) mod n+1;
   if i>n*n then
      begin
         if (sdp=nrm) and(sds=nrm) then
            afis;
      end
   else
      for j:=1 to n*n do
         if not depus[j] then
            if (slin[l]+j<=nrm) and ((l<>n)or
                  (scol[c]+j=nrm)) and (scol[c]+j<=nrm)
                  and ((c<>n) or (slin[l]+j=nrm))
                  and ((l<>c) or (sdp+j<=nrm)) and
                  ((l+c<>n+1) or (sds+j<=nrm)) then
                  if (l<>n div 2 +1) or (c<>n div 2 +1)
                     or (j=(n*n+1)div 2) then
                        begin
                           sol[l,c]:=j;
                           slin[l]:=slin[l]+j;
                           scol[c]:=scol[c]+j;
                           depus[j]:=true;
                           if l=c then
                              sdp:=sdp+j;
                           if l+c=n+1 then
                              sds:=sds+j;
                           ext(i+1);
                           slin[l]:=slin[l]-j;
                           scol[c]:=scol[c]-j;
                           depus[j]:=false;
                           if l=c then
                              sdp:=sdp-j;
                           if l+c=n+1 then
                              sds:=sds-j;
                        end;
end;

begin
   clrscr;
   repeat
      write('n=');
      readln(n);
```



```
   until n mod 2 =1;
   for p:=1 to n*n do
      depus[p]:=false;
   for p:=1 to n do
      begin
         slin[p]:=0;
         scol[p]:=0;
      end;
   sdp:=0;
   sds:=0;
   nsol:=0;
   nrm:=n*((n*n+1)div 2);
   ext(1);
   writeln('Nr. solutii: ',nsol);
   readln;
end.
```

P95. Bac2001 Varianta 1 S IV

Să se genereze toate şirurile strict crescătoare formate din numere naturale cu proprietatea că primul element din şir este n, iar ultimul element al şirului este n+k. Numerele naturale n şi k (0<n<20, 0<k<16) sunt citite de la tastatură. Fiecare şir generat va fi scris pe o linie, elementele unui şir fiind separate prin câte un spaţiu.

De exemplu, pentru n=7 şi k=3, se vor afişa (nu neapărat în această ordine) şirurile:

7 8 9 10
7 8 10
7 9 10
7 10

(20 puncte)

Rezolvare:

Deoarece primul element al oricărei soluţii trebuie să fie n, se fixează acest termen în programul principal (`sol[1]:=n`), iar procedura recursivă se apelează pentru construcţia soluţiei de la al doilea termen (`ext(2)`).

Procedura recursivă (ext) are ca parametru numărul de ordine al termenului şirului soluţie ce urmează a se fixa (i – iniţial are valoarea 2). Iniţial se verifică dacă s-a obţinut o soluţie (dacă anterior element – `sol[i-1]` – este n+k. În cazul în care s-a obţinut o soluţie aceasta se va afişa cu ajutorul procedurii afis.

În caz contrar se vor considera, pe rând, toate valorile de la valoarea următoare precedentului termen (`sol[i-1]+1`), până la ultima valoare ce trebuie să apară în şir (n+k). Pentru fiecare din aceste valori se memorează în

vectorul soluție alegerea făcută (sol[i]:=x) și se trece la următorul element al șirului soluție (ext(i+1)).

Programul Pascal:

```
program sir_cresc;
uses
   crt;
var
   sol:array[1..20] of byte;
   n,k:byte;

procedure afis(l:byte);
var
   p:byte;
begin
   for p:=1 to l do
      write(sol[p],' ');
   writeln;
end;

procedure ext(i:byte);
var
   x:byte;
begin
   if sol[i-1]=n+k then
      afis(i-1)
   else
      for x:=sol[i-1]+1 to n+k do
         begin
            sol[i]:=x;
            ext(i+1);
         end;
end;

begin
   clrscr;
   write('n=');
   readln(n);
   write('k=');
   readln(k);
   sol[1]:=n;
   ext(2);
   readln;
```



```
end.
```

P96. Bac2001 Varianta 2 S IV

Să se genereze și să se afișeze toate numerele prime de n cifre (n<10) format numai cu ajutorul cifrelor 2, 0 și 9. Valoarea numărului natural n se citește de la tastatură.
Descrieți sumar metoda folosită.
De exemplu, pentru n=3, se vor afișa (nu neapărat în această ordine) numerele:
229
929

(20 puncte)

Rezolvare:

Procedura recursivă utilizată are ca parametri poziția cifrei ce urmează a fi stabilite în cadrul numărului (i – inițial are valoarea 1) și numărul parțial obținut prin stabilirea precedentelor i-1 cifre (np – inițial are valoarea 0).

Se verifică mai întâi dacă s-au fixat toate cele n cifre (i>n), caz în care se verifică dacă numărul obținut este prim, cu ajutorul funcției booleene prim. Dacă numărul obținut este prim acesta se afișează.

Dacă numărul nu are n cifre, se procedează la considerarea fiecărei cifre din cele trei (memorate în vectorul cif), care poate ocupa poziția curentă în număr. Se observă faptul că cifra 2 și cifra 0 nu pot ocupa ultima poziție în număr (numărul ar fi par), iar cifra 0 nu poate ocupa prima poziție în număr.

O cifră poate ocupa o anumită poziție dacă aceasta nu este prima(i<>1) sau ultima (i<>n) sau cifra nu este 0 (cif[x]<>0) și, de asemenea, dacă cifra nu este ultima (i<>n) sau nu este 2 (cif[x]<>2). Dacă cifra poate ocupa poziția respectivă se trece la următoare cifră (i+1) cu actualizarea numărului prin adăugarea cifrei curente la sfârșitul său (np*10+cif[x]).

Programul Pascal:

```
program nr_n_cifre;
uses
   crt;
const
   cif:array[1..3] of byte=(2,0,9);
var
   n:byte;

function prim(nr:longint):boolean;
var
   c:longint;
begin
```

```
  c:=2;
  while (c<sqrt(nr)) and (nr mod c<>0) do
    c:=c+1;
  if c>sqrt(nr) then
    prim:=true
  else
    prim:=false;
end;

procedure ext(i:byte;np:longint);
var
  x:byte;
begin
  if i>n then
    begin
      if prim(np) then
        writeln(np);
    end
  else
    for x:=1 to 3 do
      if (i<>1) and (i<>n) or (cif[x]<>0) then
        if (i<>n) or (cif[x]<>2) then
          ext(i+1,np*10+cif[x]);
end;

begin
  clrscr;
  write('n=');
  readln(n);
  ext(1,0);
  readln;
end.
```

P97. Bac2001 Varianta 3 S IV

Pentru două numere naturale a și b date de le tastatură (0<a<b<1 000 000 000), să se afișeze toate numerele naturale mai mari sau egale cu a și mai mici sau egale cu b formate din cifre identice.

De exemplu, pentru a=51 și b=317 să se afișeze (nu neapărat în această ordine): 55, 66, 77, 88, 99, 111 și 222.

Realizați o scurtă descriere a metodei aplicate.

(15 puncte)

Rezolvare:

Procedura recursivă folosită primește ca parametri cifra curentă (c - inițial are valoarea 1) și numărul parțial format (nr – inițial are valoarea 0). Se verifică mai întâi dacă cifra este validă (c<10), caz în care se verifică dacă numărul format este mai mic decât b.

În cazul în care numărul format este mai mic decât b, dacă este mai mare decât a acesta se afișează și, indiferent cum este numărul față de a, i se adaugă o nouă cifră c (ext(c,nr*10+c)). Dacă numărul format este mai mare decât b înseamnă că s-au format toate numerele posibile cu cifra curentă și se va încerca formarea de noi numere cu următoarea cifră, începând de la numere formate dintr-o singură cifră (ext(c+1,0)).

În cazul în care c=10 procesul de formare a numerelor ia sfârșit, deoarece înseamnă că s-au utilizat toate cifrele și deci s-au format toate numerele cerute.

Programul Pascal:

```
program cif_ident;
uses
  crt;
var
  a,b:longint;

procedure ext(c:byte;nr:longint);
begin
  if c<10 then
    if nr<b then
      begin
        if nr>a then
          writeln(nr);
        ext(c,nr*10+c)
      end
    else
      ext(c+1,0);
end;

begin
  clrscr;
  write('a=');
  readln(a);
  write('b=');
  readln(b);
  ext(1,0);
  readln;
end.
```

P98. Bac2001 Varianta 4 S IV

Să se afișeze toate permutările unei mulțimi formate din n (n<10) numere naturale, a_1, a_2, ..., a_n ($0<a_i<1000$), date de la tastatură. Dacă cele n elemente nu formează mulțime (nu sunt distincte), se va afișa mesajul EROARE. În caz contrar, se vor afișa, câte una pe linie, permutările numerelor date, cu câte un spațiu între elementele permutării.

De exemplu, dacă se citesc n=4 și apoi valorile 7, 9, 51, 9, atunci se va afișa EROARE.

Dacă se citesc n=3 și apoi valorile 5, 11, 2, atunci se vor afișa (nu neapărat în această ordine) permutările:

5 11 2
5 2 11
11 5 2
11 2 5
2 11 5
2 5 11

(20 puncte)

Rezolvare:

Verificarea faptului că elementele șirului sunt distincte s-a efectuat în programul principal, în momentul introducerii datelor. Astfel fiecare valoare introdusă s-a comparat cu toate valorile introduse anterior. Dacă s-a constatat existența unui element ce se repetă se afișează mesajul de eroare și numai în caz contrar se apelează procedura de generare a permutărilor.

Pentru simplitate, procedura recursivă va genera permutările mulțimii {1, 2, ..., n} iar la afișare se va afișa, în loc de valoare din permutare, valoarea aflată în șirul inițial pe poziția respectivă (se vor genera permutările pozițiilor și se vor afișa permutările valorilor).

Procedura recursivă folosită (ext) are ca parametru poziția curentă din permutare ce urmează a fi completată (i – inițial are valoarea 1). Se verifică mai întâi dacă s-a obținut o permutare (`i>n`), caz în care aceasta se afișează cu ajutorul procedurii afis.

În caz contrar, se încearcă depunerea pe poziția i a tuturor valorilor (`for x:=1 to n`), dacă acest lucru este posibil. O anumită valoare poate fi depusă pe poziția curentă dacă nu a fost depusă anterior (`not depus[x]`).

Dacă valoarea nu a fost depusă anterior se marchează depunerea ei în vectorul soluție (`sol[i]:=x`), se marchează faptul că valoarea respectivă este depusă (`depus[x]:=true`) și se trece la următoarea poziție din permutare (`ext(i+1)`). La revenire se eliberează valoarea (`depus[x]:=false`), pentru a putea fi utilizată ulterior pe o altă poziție.

Programul Pascal:

```
program perm_sir;
uses
   crt;
var
   sir:array[1..10] of 1..1000;
   sol:array[1..10] of 1..10;
   n,c,j:1..10;
   f:boolean;
   depus:array[1..10] of boolean;

procedure afis;
var
   j:1..10;
begin
   for j:=1 to n do
      write(sir[sol[j]],' ');
   writeln;
end;

procedure ext(i:byte);
var
   x:byte;
begin
   if i>n then
      afis
   else
      for x:=1 to n do
         if not depus[x] then
            begin
               sol[i]:=x;
               depus[x]:=true;
               ext(i+1);
               depus[x]:=false;
            end;
end;

begin
   clrscr;
   write('n=');
   readln(n);
   f:=true;
   for c:=1 to n do
```

```
    begin
      write('sir[',c,']=');
      readln(sir[c]);
      for j:=1 to c-1 do
        if sir[c]=sir[j] then
          f:=false;
      depus[c]:=false;
    end;
  if f then
    ext(1)
  else
    writeln('EROARE');
  readln;
end.
```

P99. Bac2001 Varianta 5 S IV

Se citesc de la tastatură: două numere n ($0<n<15$) și s ($0<s<10^6$) și apoi n valori întregi distincte, fiecare valoare aparținând intervalului [-1000, 1000]. Să se determine toate mulțimile de numere dintre cele date, fiecare mulțime având proprietatea că suma elementelor ei este egală cu s. Fiecare mulțime se va afișa pe o linie, elementele ei fiind scrise în ordine crescătoare, despărțite prin câte un spațiu sau câte o virgulă.
De exemplu, pentru n=7, s=61 și valorile 12, 61, 22, 57, 10, 4, 23, se vor afișa, pe linii, următoarele mulțimi:
4, 12, 22, 23
4, 57
61

(20 puncte)

Rezolvare:

Pentru a se evita sortarea elementelor din fiecare mulțime, se generează mulțimile cu elemente deja sortate. În acest scop se sortează mai întâi șirul inițial de valori (v) cu ajutorul unei proceduri (sort).

Procedura recursivă utilizată (ext) are ca parametri poziția ce urmează a fi completată în mulțimea soluție (i – inițial are valoarea 1) și suma parțială – suma primelor i-1 elemente, deja stabilite – (sp – inițial are valoarea 0).

Se verifică, mai întâi, dacă s-a obținut o soluție (s=sp), caz în care aceasta se afișează cu ajutorul unei proceduri (afis). Afișarea se va face ținând cont că în vectorul sol se reține pozițiile valorilor și că este necesară afișarea valorilor (`write(v[sol[j]])`).

În cazul în care nu s-a completat încă o soluție, se vor considera, pe rând toate elementele de după precedentul element din permutare (`for j:=sol[i-1]+1 to n`). Pentru simplitate, în vectorul sol nu se reține valorile din șir ci pozițiile acestora. Șirul de valori fiind sortat, valoarea curentă va fi mai mare decât valorile anterioare din mulțimea creată.

Pentru fiecare din valori se verifică dacă acestea pot fi introduse în mulțime, adică dacă prin includerea lor nu s-ar depăși suma dată (`sp+v[j]<=s`). În cazul în care valoarea poate fi inclusă în mulțime se reține în vectorul soluție poziția ei în șirul de valori (`sol[i]:=j`) și se trece la poziția următoare din mulțimea soluție (`i+1`), cu actualizarea sumei parțiale (`sp+v[j]`).

Programul Pascal:

```
program mult_sum;
uses
  crt;
type
  sir=array[1..15] of integer;
var
  n,c:byte;
  s:longint;
  v:sir;
  sol:array[0..15] of byte;

procedure sort(var s:sir;l:byte);
var
  f:boolean;
  aux:integer;
  j:byte;
begin
  repeat
    f:=true;
      for j:=1 to l-1 do
        if s[j]>s[j+1] then
          begin
            f:=false;
            aux:=s[j];
            s[j]:=s[j+1];
            s[j+1]:=aux;
          end;
  until f;
end;

procedure afis(l:byte);
var
  j:byte;
```

```
begin
   write('{');
   for j:=1 to l do
      write(v[sol[j]],',');
   writeln(#8,'}');
end;

procedure ext(i:byte;sp:longint);
var
   j:byte;
begin
   if s=sp then
      afis(i-1)
   else
      for j:=sol[i-1]+1 to n do
         if sp+v[j]<=s then
            begin
               sol[i]:=j;
               ext(i+1,sp+v[j]);
            end;
end;

begin
   clrscr;
   write('n=');
   readln(n);
   for c:=1 to n do
      begin
         write('v[',c,']=');
         readln(v[c]);
      end;
   write('s=');
   readln(s);
   sort(v,n);
   sol[0]:=0;
   ext(1,0);
   readln;
end.
```


Bibliografie

1. *Draga – Maria Balan, George Balan* – **Limbajul BASIC – culegere de probleme pentru concursuri**, Ed. Licurici, Suceava, 1992

2. *Valentin Cristea, Irina Athanasiu, Eugenia Kalisz, Valeriu Iorga* – **Tehnici de programare**, Ed. Teora, București, 1993

3. *Valentin Cristea* – **Borland Pascal 7.0 pentru Windows**, Ed. Teora, 1994

4. *Cornelia Ivașc, Mona Prună* – **Bazele informaticii**, Ed. Petrion, 1995

5. *Mihai Mocanu, Gheorghe Marin, Costin Bădică, Carmen Bădică* – **333 probleme de programare**, Ed. Teora, București, 1993

6. *Florin Munteanu & co* – **Programarea calculatoarelor** – Ed. Didactică și pedagogică, București, 1993

7. *Ioan Odăgescu & co* – **Metode și tehnici de programare**, Ed. Intact, București, 1994

8. *Mihai Oltean* – **Proiectarea și implementarea algoritmilor**, Ed. Agora, 1999

9. *Octavian Pătrășcoiu, Gheorghe Marian, Nicolae Mitroi* – **Elemente de combinatorică. Metode, algoritmi și probleme**, Ed. All, București, 1994

10. *Kovacs Sandor*, **Turbo Pascal 6.0 – Ghid de utilizare**, Ed. Microinformatica, Cluj-Napoca, 1992

CUPRINS