

PROIECTAREA SI IMPLEMENTAREA ALGORITMULUI OR

Mihai Oltean
Institutul Politehnic
din Iasi

Mihai Oltean



AGORA

I. METODA PROGRAMARII DINAMICE

Termenul de *programare dinamică* desemnează o metodă de rezolvare a problemelor a căror soluție se construiește dinamic în timp. Cuvântul *programare* din această denumire nu are nimic de a face cu calculatoarele, ci este un termen pur matematic, derivând dintr-o categorie mai largă de probleme, și anume *programarea matematică*. Si, de fapt era normal să fie aşa, deoarece inițiatorul acestei metode este un *matematician american*, pe nume *Richard Bellman*, profesor și cercetător la mai multe universități de prestigiu: Princeton, University of Southern California etc. Prima sa carte despre această metodă se numește *Dynamic Programming* și a fost publicată în 1957 la universitatea Princeton, din New Jersey. Din acel moment, Bellman, a mai publicat câteva zeci de cărți și articole despre această metodă, până la moartea sa în anul 1982.

Preocupările lui Bellman au fost mult mai vaste, iar în ultimii ani ai vieții atenția i-a fost atrasă, ca și mulțor alțor informaticieni și matematicieni geniali, de domeniul inteligenței artificiale. În anul 1978 a publicat carteă *An Introduction to Artificial Intelligence: Can Computers Think?*, în care își exprimă opinia personală asupra acestui domeniu. În această carte, Bellman dă o definiție celebră matematică și în special teoriei deciziei: *dacă ne întrebăm pe noi, înșine care este cea mai importantă valoare pe care o are matematica în lumea noastră, răspunsul este că ea [matematica] e o călărie prin care se poate evita responsabilitatea. Dacă avem de luat o decizie serioasă, putem spune: "facem asta pe baza statisticilor", sau "facem asta pe baza teoriilor matematice". Apoi, dacă lucrurile merg rău, sau dacă cineva vine și se plângă "Dă ce ai luat această decizie proastă?", atunci spunem "Am făcut-o în deplină concordanță cu teoria."*. Tot în această carte Bellman spunea că *lumea noastră merge bine nu pentru că se luă deciziile cele mai bune, ci pentru că majoritatea deciziilor sunt de mică importanță*.

O altă carte importantă scrisă de Bellman împreună cu *S. E. Dreyfus*, și pe care o vom cita de mai multe ori, este: *Programarea dinamică aplicată*. Ea a fost tradusă în

limba română și a fost editată de *Editura Tehnică*. Vă recomandăm însă, să citiți originalul, deoarece autori au fost mult mai atenți decât traducătorii la indicii din formule. În această lucrare, se găsesc multe exemple reprezentative pentru domeniile și direcțiile în care această metodă are aplicabilitate.

Să revenim acum la prezentarea metodei. O problemă rezolvabilă prin metoda programării dinamice trebuie adusă mai întâi la o formă discretă în timp. Decizii care se iau pentru a obține un rezultat trebuie să se poată lua pas cu pas. De asemenea, foarte importantă este ordinea în care acestea se iau. Programarea dinamică este (și nu luă atestă rânduri ca pe o definiție) în esență un proces decizional în mai multe etape: în starea inițială a problemei luăm prima decizie, care determină o nouă stare a problemei în care luăm o nouă decizie... termenul dinamic se referă chiar la acest lucru: problema este rezolvată în etape dependente de timp. Variabilele, sau funcțiile care descriu fiecare etapă trebuie să fie în aşa fel definite încât să descrie complet un proces, deci pentru acest lucru va trebui să răspundem la două întrebări:

- care este etapa *inițială* (caz în care avem de a face cu un proces decizional descendenter) sau care este etapa *finală* (caz în care avem de a face cu un proces decizional ascendent)?
- care este *regula* după care trecem dintr-o etapă în alta? De obicei această regulă este exprimată printr-o *recurență*.

Deoarece, avem de a face cu o problemă care se rezolvă în mai multe etape, nu ne mai rămâne decât să vedem cum luăm decizii dintr-o etapă în alta. Nu mă refer aici la relația de recurență de care am vorbit mai sus, ci la faptul că foarte probabil apare posibilitatea că la un anumit moment să putem alege din mai multe decizii. De exemplu, problema calculului *numerelor lui Fibonacci* se încadrează în categoria programării dinamice deoarece:

- este un proces în etape,
- fiecarei etape k îi corespunde calculul celui de al k -lea număr Fibonacci,
- există o singură decizie pentru a trece la o etapă superioară,
- însă determinarea unui drum ce leagă două orașe A și B și care trece printr-un număr minim de alte orașe este tot programare dinamică deoarece:
 - este tot un proces în etape,
 - fiecarei etape k îi corespunde determinarea unui drum de lungime k ce pleacă din orașul A ,
 - dar există mai multe decizii pentru trecerea la drumul de lungime $k + 1$.

Deci, să vedem ce decizie luăm! În cele ce urmează prin strategie înțelegem un set de decizii. Conform principiului lui Bellman, numit *principiul optimalității* avem:

O strategie are proprietatea că oricare ar fi starea inițială și decizia inițială, decizii rămase trebuie să constituie o strategie optimă privitoare la starea care rezultă din decizia anterioară.

Rutherford Aris descrie principiul în termeni mai conversaționali:

Dacă nu faci cel mai bine cu ceea ce s-a întâmplat să ai, atunci nu vei face niciodată cel mai bine cu ceea ce ar fi trebuit să ai.

Să vedem cum se aplică practic acest principiu. Vrem să maximizăm o funcție $R(x_1, x_2, \dots, x_N)$, unde domeniul este format din acelle N -uple (x_1, x_2, \dots, x_N) care îndeplinesc condițiile $x_i \geq 0$ ($i=1, \dots, N$) și $x_1 + x_2 + \dots + x_N = x$. Presupunem de asemenea că R se poate scrie sub formă $R = g_1(x_1) + g_2(x_2) + g_3(x_3) + \dots + g_N(x_N)$. Pentru a fi mai clar, putem spune că x este o resursă (timp, apă, pâine etc.), care trebuie distribuită la N activități. De pe urma distribuirii cantității x_i la activitatea i se obține un anumit profit $g_i(x_i)$. Se presupune că valorile returnate de funcțiile g sunt situate pe aceeași scară de măsură, în caz contrar trebuie făcută o conversie prealabilă.

În această situație împunem ca deciziile să se poată lua pe rând, deci vom avea de a face cu un proces temporal. Se cere să se determine o strategie de alocare a resursei x , la procesele g_1, \dots, g_N astfel încât să se maximizeze valoarea funcției R .

De asemenea se presupune că funcțiile g_1, \dots, g_N nu sunt monotone (crescător sau descrescător), deoarece în caz contrar, întreaga resursă x se poate aloca unui singur proces (să zicem g_k) care dă profitul maxim. De asemenea există cazuri particulare, când funcțiile g_k sunt continue și derivabile, în această situație putându-se aplica metodele cunoscute din analiză. Însă aici vom avea neplăcerea de a determina uneori, în loc de extreame globale, anumite extreme locale. Programarea dinamică înălță toate aceste neajunsuri.

Din cauza că $x_1 + x_2 + \dots + x_N = x$ va apărea un conflict de interes între procesele (activitățile) g_1, \dots, g_N care fac ca problema să nu fie trivială. Să notăm cu $f_k(x)$ venitul maxim obținut de la o alocare a cantității x pentru N activități.

Avem $f_N(0) = 0$ și $f_1(x) = g_1(x)$.

Trebuie găsită o relație între $f_N(x)$ și $f_{N-1}(x)$. Fie x_N , $0 \leq x_N \leq x$, alocarea pentru a N -a activitate (proces). Atunci, conform principiului optimalității, indiferent de valoarea precisă a lui x_N , noi știm că restul $x - x_N$ va fi optim pentru celelalte $N-1$ activități.

Principiul optimalității spune că după ce am ales o valoare inițială x_N , nu mai suntem obligați să examinăm toate strategiile pe care le implică această alegeră particulară a lui x_N , ci numai pe aceleia care sunt optime pentru un proces cu $N-1$ pași și cu resturile $x - x_N$.

Venitul total va fi deci $g_N(x_N) + f_{N-1}(x - x_N)$.

Pe noi ne interesează să maximizăm venitul total (valoarea funcției R), deci avem:

$$f_N(x) = \max_{0 \leq x_N \leq x} [g_N(x_N) + f_{N-1}(x - x_N)]$$

Pentru demonstrarea corectitudinii acestei formule observăm că:

$$\max_{x_1 + x_2 + \dots + x_N = x} = \max_{0 \leq x_N \leq x} \left[\max_{x_1 + x_2 + \dots + x_{N-1} = x - x_N} \right], \text{ deci putem scrie:}$$

$$\begin{aligned}
 f_N(x) &= \max_{x_1 + \dots + x_N = x} [g_N(x_N) + \dots + g_1(x_1)] = \\
 &= \max_{0 \leq x_N \leq x} \left[\max_{x_1 + \dots + x_{N-1} = x - x_N} [g_N(x_N) + \dots + g_1(x_1)] \right] = \\
 &= \max_{0 \leq x_N \leq x} [g_N(x_N) + \max_{x_1 + \dots + x_{N-1} = x - x_N} [g_N(x_N) + \dots + g_1(x_1)]] = \\
 &= \max_{0 \leq x_N \leq x} [g_N(x_N) + f_{N-1}(x - x_N)];
 \end{aligned}$$

Demonstrarea corectitudinii unui algoritm de programare dinamică se face, așa cum rezultă și din principiul optimalității, prin *inducție matematică*. Exemplul de mai sus a confirmat acest lucru.

Rămâne la latitudinea dumneavoastră implementarea acestei rezolvări. Numerele x_1, x_2, \dots, x_N, x sunt întregi.

Aceeași metodă se aplică și proceselor multidimensionale. De exemplu, să presupunem că avem de maximizat aceeași funcție R care de dată aceasta se scrie sub forma $R(x_1, \dots, x_N, y_1, \dots, y_N) = g_1(x_1, y_1) + g_2(x_2, y_2) + \dots + g_N(x_N, y_N)$. Resursele sunt supuse restricțiilor:

$$\begin{aligned}
 x_1 + x_2 + \dots + x_N &= x, \\
 y_1 + y_2 + \dots + y_N &= y, \\
 x_i &\geq 0, y_i \geq 0, \text{ iar}
 \end{aligned}$$

$g_i(x_i, y_i)$ este venitul obținut în urma celei de a i -a activități, dacă pentru ea se alocă x_i, y_i .

Notăm cu $f_N(x, y)$ venitul maxim obținut corespunzător alocării cantităților x și y pentru N activități.

Averm $f_i(x, y) = g_i(x, y)$.

Trebuie găsită o relație între $f_N(x, y)$ și $f_{N-1}(x, y)$. Fie (x_N, y_N) , $(0 \leq x_N \leq x, 0 \leq y_N \leq y)$ alocarea pentru a N -a activitate (proces); atunci, conform principiului optimalității, indiferent de valoarea precisă a lui (x_N, y_N) , știm că restul de $(x - x_N, y - y_N)$ va fi optim pentru celelalte $N - 1$ activități.

Venitul total va fi deci $g_N(x_N, y_N) + f_{N-1}(x - x_N, y - y_N)$.

Ne interesează să maximizăm venitul total (valoarea funcției R), deci avem:

$$f_N(x, y) = \max_{0 \leq x_N \leq x} \max_{0 \leq y_N \leq y} [g_N(x_N, y_N) + f_{N-1}(x - x_N, y - y_N)]$$

Există probleme în care apare și un element aleator, introdus de factori independenți de voința noastră. De exemplu, putem avea de a face cu probabilitatea ca moneda să cadă pe față cu stema, sau cu probabilitatea ca un anumit produs să fie vândut etc. Să exemplificăm și această situație.

Enunț

Un vapor poate fi încărcat cu produse de N tipuri. Notăm cu:

w_i = greutatea unei piese de tipul i ,

s_i = volumul unei piese de tipul i ,

d_i = numărul de piese disponibile pentru încărcare din tipul i ,

p_{zi} = probabilitatea ca la destinație să se ceară z piese de tipul i ,

c_i = penalizarea care trebuie plătită dacă nu se acoperă cererea de piese de tipul i la destinația vaporului.

Să cere să se determine încărcatura care minimizează costul total probabil, știind că vaporul poate transporta o greutate maximă w și un volum maxim de marfă s .

Soluție

Deoarece nu știm exact care este cererea de produse, ci doar o anumită probabilitate cu care este cerut un anumit tip de produse, spunem că avem de a face cu un proces stochastic.

Să notăm cu x_i numărul de piese de tipul i încărcate pe vapor. Atunci costul probabil datorat pentru cererea neacoperită de piese de tipul i este:

$$c_i \cdot \sum_{z=x_i+1}^{d_i} (z - x_i)p_{zi}; \text{ rezultă costul total probabil:}$$

$$E_N = \sum_{i=1}^N \left[c_i \cdot \sum_{z=x_i+1}^{d_i} (z - x_i)p_{zi} \right],$$

unde x_i sunt supuși restricțiilor:

$$\begin{aligned}
 x_i &= 0, 1, 2, \dots, d_i \\
 x_1 w_1 + x_2 w_2 + \dots + x_N w_N &\leq w \\
 x_1 s_1 + x_2 s_2 + \dots + x_N s_N &\leq s
 \end{aligned}$$

Notăm cu $f_k(w', s')$ costul asociat unei alegeri optime a pieselor din primele k tipuri, unde restricția de încărcare a vaporului pentru greutate este w' , iar pentru volum s' , $0 \leq w' \leq w$, $0 \leq s' \leq s$.

$$\begin{aligned}
 f_k(w', s') &= \min_{x_k} \left[c_k \cdot \sum_{z=x_k+1}^{d_k} (z - x_k)p_{zk} + f_{k-1}(w' - x_k w_k, s' - x_k s_k) \right], \text{ unde} \\
 0 \leq x_k &\leq \min \left\{ \left\lfloor \frac{w'}{w_k} \right\rfloor, \left\lfloor \frac{s'}{s_k} \right\rfloor \right\}.
 \end{aligned}$$

Definiție

Fie P problema care trebuie rezolvată. Simbolul P codifică problema inițială împreună cu dimensiunea datelor de intrare. O *subproblemă* Q_i (care este rezolvată la etapa i) are aceeași formă ca P , dar datele de intrare pe care le prelucrază sunt mai mici în comparație cu cele cu care lucrează P . Cuvântul *dinamic* vrea să sugereze tocmai acest lucru: uniformitatea subproblemelor și rezolvarea lor în mod ascendent. Pe scurt Q_i se obține din P printr-o restricționare a datelor de intrare. Din această definiție rezultă că zisa dependență (mai precis inclusivă) dintre subprobleme. O subproblemă Q_i conține alte subprobleme dacă datele asupra cărora operează Q_i sunt mai mari decât datele asupra cărora operează subproblemele conținute, deci nu este vorba doar de o simplă dependență între subprobleme, acestea fiind *incluse* una în celalătă. (De exemplu, problema rezolvării ecuației de gradul II, de asemenea se poate descompune în subprobleme, dar acestea nu sunt uniforme.)

Există două tipuri de subprobleme: *directe* care rezultă din relația de recurență și subprobleme *indirecte* care de fapt sunt sub-sub-subprobleme ale problemei inițiale. De exemplu, în cazul numerelor lui Fibonacci, pentru determinarea termenului $F(5)$ subprobleme directe sunt $F(4)$ și $F(3)$, iar $F(2)$, $F(1)$ și $F(0)$ sunt subprobleme indirecte.

Definirea relațiilor dintre etape se face recursiv. Prin prisma unui matematician acest lucru nu este inconvenient, dar orice informatician știe că recursivitatea înseamnă resurse (timp și memorie) consumate. Presupun că eroarea *Stack overflow* a apărut de multe ori pe ecranele dumneavoastră. Pe lângă inconveniențele legate de apelurile recursive, o subproblemă este calculată de mai multe ori. Aceste lucruri pot fi evitate dacă subproblemele se calculează începând de jos în sus (adică de la cea mai "mică", către cea mai "mare") și se rețin rezultatele obținute. În cazul numerelor Fibonacci, cea mai mică subproblemă este calcularea lui $Fibonacci(0)$ și a lui $Fibonacci(1)$, iar cea mai mare este calcularea lui $Fibonacci(N)$, unde N este un număr dat.

Deci, pentru rezolvarea unei probleme de programare dinamică sunt necesare următoarele etape:

1. Aranjarea problemei astfel încât să rezulte caracterul de proces cu mai multe etape (mai mulți pași).
2. Găsirea unei relații de recurență între etape. Acest lucru va însemna un algoritm recursiv (inefficient).
3. Eliminația recurenței prin calcularea și memorarea rezultatelor subproblemelor. Va rezulta astfel o ordine de calcul al subproblemelor.
4. Algoritmul rezultat anterior poate fi îmbunătățit printr-o re-arranjare a ordinii de calcul a subproblemelor.
5. Dacă se dorește să se afișeze și etapele care au intervenit în calculul soluției, va fi necesară, memorarea la fiecare etapă, a subproblemelor care au generat-o.

Să exemplificăm toți pașii pe problema *submulțimii de sumă dată*.

Enunț

Se dă un sir de numere naturale nenule x_1, x_2, \dots, x_N și un număr T . Se cere să se determine dacă există o submulțime de numere din sirul x a căror sumă este T .

Soluție
O subproblemă (o etapă) poate fi considerată ca fiind calcularea unei sume j , folosind primele i numere din sirul x . Este evident că avem de a face cu un proces decizional în mai multe etape.

Suma dată T se poate obține fie folosind valoarea x_N , fie nefolosind-o. În cazul în care o folosim, suma $T - x_N$ trebuie obținută din numerele x_1, \dots, x_{N-1} , iar în caz contrar suma T trebuie obținută din numerele x_1, \dots, x_{N-1} . Deci un algoritm recursiv care rezolvă această problemă este:

```
function Partitie(N, T:Word):Boolean;
begin
  if T=0 then Partitie:=true
  else Partitie:=Partitie(N-1, T) or Partitie(N-1, T-x[N])
end;
```

Complexitatea funcției de mai sus este 2^N . Să încercăm acum să reținem soluțiile subproblemelor. Construim o matrice $M_{N,T}$ care poate lua valorile 1 (true), 0 (false) și -1 (nedefinit). Elementul $M_{N,T}$ va fi true dacă T se poate scrie ca sumă de numere din sirul x_1, x_2, \dots, x_N . Se presupune că inițial totul este setat la nedefinit. Am folosit câteva conversii de tip pentru a trece de la valori booleene (true și false) la valori întregi (1 și 0).

```
function Partitie(N, T:Integer):Boolean;
begin
  if T<0 then Partitie:=false
  else
    if T=0 then Partitie:=true
    else
      begin
        if M[N, T]=-1 then M[N, T]:=Ord(Partitie(N-1, T) or Partitie(N-1, T-x[N]));
        Partitie:=Boolean(M[N, T])
      end
end;
```

Potem elimina recursivitatea obținând o programare dinamică care operează de jos în sus de la subprobleme mici la subprobleme mai mari:

```

function Partitie(N,T:Integer):Boolean;
begin
  M[0,0]:=true;
  for j:=1 to N do
  begin
    M[j,0]:=true;
    for i:=1 to T do
      if x[j]>i then M[j,i]:=M[j-1,i]
      else M[j,i]:=M[j-1,i] or M[j-1,i-x[j]];
  end;
  Partitie:=M[N,T];
end;

```

Pentru a simplifica funcția de mai sus observăm că nu este necesar să reținem dacă suma i s-a putut descompune folosind numerele de la x_1 până la x_j , ci trebuie să reținem doar dacă suma i se poate scrie ca sumă, folosind numerele de la x_1 la x_N . În acest fel "scăpăm" de indicele j , iar matricea M se reduce la un sir. Acum M_i este true, dacă i se poate scrie ca sumă de numere din sirul x_1, \dots, x_N . În plus, parcursarea vectorului M se face descrescător.

Algoritmul de mai jos funcționează astfel:

- La pasul 0, inițializăm M_0 cu valoarea true, deoarece numărul 0 se poate scrie (folosind 0 termeni) ca sumă de numere dintr-un sir dat.
- La pasul 1 setăm valoarea M_{x_1} la true.
- La pasul 2 setăm valorile M_{x_2} și $M_{x_1+x_2}$ la true.
- La pasul j vom avea true pe acele poziții care diferă prin x_j de alte poziții marcate la pași anteriori cu true.

```

function Partitie(N,T:Integer):Boolean;
var max:Word;
begin
  M[0]:=true;
  max:=0;
  for j:=1 to N do
  begin
    for i:=max downto 0 do
      if M[i] then M[i+x[j]]:=true;
    max:=max+x[j];
  end;
  Partitie:=M[T];
end;

```

Pentru a determina și submulțimea care a generat suma T , procedăm în felul următor: definim încă un vector de lungime T care pe poziția i va reține proveniența sumei i , știind că aceasta s-a scris sub forma $i = j + x_k$. Rămâne să se decidă ce se va memoră: x_k sau j . Noi am ales varianța cu j .

Reconstituirea soluției se face printr-o procedură recursivă (pentru a afișa numerele în ordinea crescătoare a indicilor; dacă nu se dorea acest lucru se folosea doar o strucțură repetitivă de tip while).

```

function Partitie(N,T:Integer):Boolean;
var max:Word;
begin
  M[0]:=true; max:=0;
  for j:=1 to N do
  begin
    for i:=max downto 0 do
      if M[i] then begin M[i+x[j]]:=true; d[i+x[j]]:=j end;
      max:=max+x[j];
    end;
    Partitie:=M[T];
  end;
  procedure Reconstituie(T:Word);
  begin
    if T>0
    then begin Reconstituie(d[T]);
            Writeln('Am folosit valoarea:',T-d[T])
          end;
  end;

```

Unele probleme sunt dotate natural cu o structură în mai multe etape. Altele pot fi rezolvate artificial cu o astfel de structură.

Vom exemplifica cele afirmate prin rezolvarea problemei ciclului hamiltonian cu cost minim (sau problema comis-voiajorului).

Enunț

Un comis-voiajor trebuie să viziteze o singură dată fiecare din cele $N+1$ orașe plecând dintr-un oraș oarecare și întorcându-se tot în acel oraș. Între oricare două orașe i și j există o legătură directă care are atașat un cost pozitiv d_{ij} . Care drum minimă realizează distanța totală parcursă de comis-voiajor?

Soluție

Fără a pierde din generalitate, din moment ce fiecare tur este ciclic, putem fixa originea de exemplu în orașul 0. Presupunem că în un anumit pas al unui ciclu optim care începe cu orașul 0, s-a ajuns în orașul i și au mai rămas k orașe j_1, j_2, \dots, j_k nevizitate înainte de a se întoarce în orașul 0. Este clar că, deoarece turul este optim, drumul între i și 0 (trecând prin orașele j_1, j_2, \dots, j_k) trebuie să fie optim (să aibă lungime minimă), deoarece în caz contrar întregul ciclu nu ar mai fi optim (s-ar putea alege o cale mai scurtă între i și 0 astfel încât să se treacă prin orașele j_1, j_2, \dots, j_k).

Notăm cu c_{i,j_1,j_2,\dots,j_k} lungimea unui drum minim, care pornește din orașul i , ajunge în orașul 0 și trece o singură dată prin fiecare din orașele nevizitate j_1, j_2, \dots, j_k .

Soluția problemei se va afla în c_{0,j_1,j_2,\dots,j_k} .

Să notăm cu $d_{i,j}$ distanța directă dintre două orașe i și j . Atunci, ca o consecință a observației anterioare avem:

$$c_{i,j_1,j_2,\dots,j_k} = \min_{1 \leq m \leq k} \{ d_{i,j_m} + c_{j_m,j_1,j_2,\dots,j_{m-1},j_{m+1},\dots,j_k} \}.$$

Această formulă este o aplicație directă a principiului optimalității.

Pentru calcularea acestei formule începem cu $c_{i,j} = d_{i,j} + d_{j,0}$, iar apoi calculăm pe c_{i,j_1,j_2} , apoi pe c_{i,j_1,j_2,j_3} etc., iar în final pe c_{0,j_1,j_2,\dots,j_k} .

Selecția de valori m care minimizează expresia de mai sus, definește drumul minim.

Să vedem acum care este complexitatea acestui algoritm. După cum se observă și din formula optimalității, numărul de parametri ai funcției c este mare (în cel mai de favorabil caz $N + 1$). Ar fi fatal pentru un programator în Turbo Pascal (sau în orice alt limbaj) să definească un tablou $N + 1$ dimensional, fiecare dimensiune fiind de lungime N . Aceasta ar însemna $(N + 1)^N$ spațiu de memorare. Pentru $N = 5$ sunt necesari 7776 octeți, iar pentru $N = 6$ sunt necesari 117649 octeți (în cazul în care distanța totală se poate memora pe un octet). Deci, soluția ar fi inadmisibilă.

Dar, din fericire, există o soluție care reduce considerabil volumul de memorie folosit. Acest lucru este posibil datorită faptului că numerele j_1, j_2, \dots, j_k nu trebuie stocate în ordine, deci pentru reținerea lor nu este nevoie de un sir de lungime k , ci de o mulțime de cardinal k . În Pascal mulțimile sunt reprezentate pe 8, 16, 24 sau 32 bytes, în funcție de numărul de elemente, deci în loc să avem o reprezentare a numerelor j_1, j_2, \dots, j_k pe k octeți, vom avea nevoie doar de $([k \text{ div } 64] + 1) * 8$ octeți. Observăm de asemenea că nu este necesară reținerea tututor numerelor c_{i,j_1,j_2,\dots,j_k} ($k = 1, \dots, N$), deoarece c_{i,j_1,j_2,\dots,j_k} depind doar de numerele $c_{i,j_1,\dots,j_m,\dots,j_k}$ ($1 \leq m \leq k$) adică doar de submulțimile de cardinal $j - j$ ale mulțimii numerelor j_1, j_2, \dots, j_k .

Nici codificarea sub formă de mulțimi nu este foarte economică din punct de vedere al memoriei ocupate. Să etichetăm orașele cu numere din mulțimea $\{0, \dots, N\}$. La pasul 1 (inițial) trebuie să calculăm cele N numere c_{i,j_i} . La pasul 2 trebuie să calculăm cele C_N^2 numere c_{i,j_1,j_2} , și așa în continuare, la pasul k trebuie să calculăm cele C_N^k numere c_{i,j_1,j_2,\dots,j_k} pentru un i fixat. Submulțimile j_1, j_2, \dots, j_k ale unei mulțimi de N elemente sunt în număr de C_N^k . Acestea se pot genera în ordine lexicografică. Mai mult decât atât, având o submulțime de cardinal k a unei mulțimi de cardinal N se poate obține direct numărul ei de ordine lexicografică. De exemplu, submulțimea $\{1, 2, \dots, k\}$ are numărul de ordine lexicografică 1, iar submulțimea $\{N - k + 1, \dots, N\}$ are numărul de ordine lexicografică C_N^k . Procedeul este valabil și invers, adică, având un număr de ordine lexicografică se poate genera direct (printr-un algoritm de complexitate $O(k)$) submulțimea de cardinalitate k corespunzătoare.

Revenind la problema noastră, nu este nevoie să reținem întreaga submulțime, ci doar un singur număr care să indice numărul de ordine al submulțimii în ordine lexicografică. Cel mai mare astfel de număr de ordine C_N^k este acela pentru care k este jumătate din N . Să presupunem că N este par. Avem următoarele valori:

N	$C_N^{[N/2]}$
10	252
12	924
14	3432
16	12870
18	48620

Până la valoarea $N = 16$ (deci 17 orașe) putem folosi mediul Turbo Pascal. Bineînțeles că stocarea unui vector de lungime 16×12870 se va face în Heap, adică accesul se va face în genul $c[i]^j$. Lungimea drumului poate să fie un număr de tip Word, adică trebuie să fie mai mică de 65535 care credem că este mai mult decât necesar. Pentru $N = 18$ lungimea drumului trebuie să încapă pe un octet, adică să fie mai mică decât 255. Valori mai mari pentru N se pot implementa în limbiile care folosesc registri pe 32 de biți, cum ar fi mediul Borland Delphi.

Complexitatea va fi în acest caz de ordinul $O(N^2 \cdot 2^N)$, dacă facem abstracție de determinarea numărului de ordine lexicografică. Dacă am fi rezolvat problema în stil "brut" prin generarea tuturor celor $N!$ drumuri, numărul de operații necesare ar fi fost mult mai mare. Următorul tabel demonstrează acest lucru.

N	$N!$	$N^2 2^{N-1}$
1	1	2
2	2	16
3	6	72
4	24	256
5	120	800
6	720	2312
7	5040	6272
8	40320	14336
9	362880	41472
10	3628800	102400
11	39916800	247808
12	479001600	589824
13	6227020800	1384448
14	87178291200	3211264
15	1307674368000	7372800
16	2092278988000	16777216

Observăm că pentru $N < 8$ este mai eficientă generarea celor $N!$ drumuri, dar pentru $N = 10$ algoritmul de programare dinamică este de 35 de ori mai rapid, iar pentru $N = 16$ de 1247095 mai rapid decât generarea tuturor drumurilor. Însă, dacă generarea tuturor drumurilor nu se face brut, ci prin aplicarea metodei backtracking cu optimizări bine alese, atunci raportul de mai sus se reduce considerabil.

Se presupune că nu între toate orașele există legături directe. Determinați în acest caz formulele de calcul care calculează numerele $c_{i,j}, \dots, c_{j,k}$ și implementați într-un program ceea ce ati obținut.

O problemă de programare dinamică se poate reprezenta sub forma unui graf orientat. Fiecare nod îi corespunde o etapă (sau o subproblemă), iar din relațiile de recurență se deduce modul de adăugare a arcelor. Mai precis, vom adăuga un arc de la starea (etapa) i la starea (etapa) j dacă starea j depinde direct de starea i . Dependența directă dintre etape este dată de relațiile de recurență.

Construirea unui astfel de graf este echivalentă cu rezolvarea problemei. Determinarea șirului de decizii care au dus la soluție se reduce la o problemă de drum în grauri. Să exemplificăm acest lucru pentru problema prezentată la început, și anume *submulțime de sumă dată*.

Vom construi un graf cu $T + 1$ noduri (în care T este suma care trebuie obținută). Etichetăm nodurile cu numere distincte din mulțimea $\{0, \dots, T\}$. Primul nod este nodul 0. La pasul 0, marcăm acest nod (nodul 0). La pasul k vom marca nodurile ale căror numere de etichetare sunt mai mari decât x_k decât numerele de etichetare ale unor noduri deja marcate. Problema are soluție dacă vom reuși să marcăm nodul T .

```

var a:array[0..MaxT, 0..MaxT] of Word;
function Partitie(N,T:Integer):Boolean;
  var max,i,j:Word;
begin
  a[0,0]:=1; max:=0;
  for i:=1 to N do
    begin
      for j:=max downto 0 do
        if a[j,j]=1
        then { Un nod j se consideră marcat dacă a[j,j]=1 }
          begin a[j,j+x[i]]:=1; a[j+x[i],j+x[i]]:=1 end;
        max:=max+x[i];
    end;
  Partitie:=a[T,T]=1;
end;
```

Reconstituirea soluției se reduce la determinarea unui drum în graf.

Pentru rezolvarea problemelor de programare dinamică se poate da și un algoritm general. Avem nevoie însă în prealabil de câteva considerații.

Metoda programării dinamice

Notăm cu P , problema pe care dorim să o rezolvăm. Înțelesul pe care il dăm lui include și dimensiunea datelor de intrare. De exemplu, în cazul problemei submulțim de sumă dată, P este caracterizată de (T, x_1, \dots, x_N) .

Am spus anterior că o etapă, sau o subproblemă are aceeași formă ca și problema de rezolvat la care sunt adăugate câteva restricții. Aici o subproblemă este descompunerea lui i ca sumă de numere din vectorul x_1, \dots, x_N .

Pentru ca aceste probleme să poată fi calculate trebuie stabilită o ordine în care vor fi prelucrate. Considerând reprezentarea sub formă unui graf (nodurile corespunzătoare etapelor, muchiile – deciziilor), va trebui să efectuăm o sortare topologică asupra nodurilor grafului. Fie Q_0, Q_1, \dots, Q_N ordinea rezultată în urma unei astfel de sortări. Prin Q_i am notat o subproblemă (Q_0 este subproblema cea mai mică). În cazul submulțim de sumă dată, Q_i este chiar descompunerea lui i în sumă de numere din vectorul x . Fie S_i soluția problemei Q_i . Soluția subproblemei Q_N este soluția problemei P .

Algoritmul general este:

```

procedure Rezolva (P)
begin
  initializeaza(S0)
  for i := 1 to N do
    progreseaza(S0, ..., Si-1, Si);
  { Aceasta procedura calculeaza solutia problemei Qi pe baza
  { solutiilor problemelor Q0, ..., Qi-1, cu o recurenta inapoi.
  Afiseaza(SN)
end;
```

Demonstrarea corectitudinii unui astfel de algoritm se face prin inducție după i .

Referitor la complexitatea unui algoritm de programare dinamică, putem spune că aceasta depinde de mai mulți factori: numărul de stări, numărul de decizii cu care se poate trece într-o stare, complexitatea subproblemei inițiale (Q_0)... Ceea ce apare înălțat în toate problemele, este numărul de stări (etape). Deci complexitatea va avea forma $O(N^k)$.

Noțiunea de complexitate pseudo-polynomială, de asemenea este legată uneori de *complexitatea unui algoritm de programare dinamică*. Fie D mulțimea tuturor intrărilor corecte pentru o problemă dată. Definim două funcții:

Max: $D \rightarrow Z^+$ și

Lungime: $D \rightarrow Z^+$.

Funcția *Max* indică valoarea maximă a datelor de intrare, iar funcția *Lungime* indică lungimea lor.

Un algoritm este numit *algoritm pseudo-polynomial*, dacă funcția sa de complexitate este mărginită superior de o funcție polinomială în două variabile: *Max[I]* și *Lungime[I]*. Prin definiție, orice algoritm polinomial este și pseudo-polynomial, deoarece se execută într-un timp mărginit de un polinom în *Lungime[I]*, dar nu toți algorit-

mii pseudo-polinomiali sunt polinomiali (vezi problema submulțimii de sumă dată). Pentru problemele care au proprietatea că $\text{Max}[I]$ este mărginită de o funcție polinomială în variabila $\text{Lungime}[I]$ nu există nici o distincție între algoritmii polinomiali și cei pseudo-polynomiali. Să vedem când se poate și când nu se poate construi un algoritm pseudo-polynomial? Spunem despre o problemă că este *number-problem* dacă nu există nici o funcție polinomială p , astfel încât $\text{Max}[I] \leq p(\text{Lungime}[I])$ pentru orice intrare I corectă. Problema submulțimii de sumă dată este o astfel de problemă (*number problem*). Singurele probleme NP-complete care sunt candidate pentru a fi rezolvate prin algoritmi pseudo-polynomiali se află în această clasă de probleme.

Să vedem de ce totuși problema submulțimii de sumă dată are complexitatea pseudo-polynomială și nu polinomială. Reamintim definiția complexității:

Spunem despre o funcție $f(n)$ că are complexitatea $O(g(n))$, dacă există o constantă c astfel încât $|f(n)| \leq c \cdot g(n)$, pentru toate valorile $n \geq 0$. Funcția de complexitate a unui algoritm polinomial este $O(p(n))$, unde p este o funcție polinomială, iar n reprezintă lungimea intrării (lungimea datelor de intrare).

Să revenim la dilema *polynomial* versus *pseudo-polynomial* a problemei submulțimii de sumă dată. Dificultatea apare la codificarea intrării. Pentru a reprezenta un număr x_1 avem nevoie de $\log_{10}(x_1)$ cifre din baza 10. Astfel pentru problema noastră, lungimea intrării este $n \cdot \log_{10}(T)$. Complexitatea algoritmului este $O(n \cdot T)$. Din definiția complexității polinomiale ar fi trebuit să avem că $|O(n \cdot T)| \leq c \cdot |O(p(n \cdot \log_{10}))|$, unde p este o funcție polinomială. Dar este evident că un astfel de polinom p nu poate exista. Din această cauză rezolvarea submulțimii de sumă dată este pseudo-polynomială și nu simplu polinomială.

Există mai multe clasificări ale problemelor de programare dinamică:

1. După natura deciziilor pot fi *deterministe* sau *nedeterministe*. La cele nedeterministe, în scrierea relațiilor de recurență intervine și o probabilitate (probabilitatea ca produsul să fie vândut, ca individul x să ajungă într-un punct dat, ca moneda să cădă pe partea cu stema, ca o mașină să se strice după ce a produs un număr de piese.).
2. După valoarea deciziilor pot fi *de optimizare* (determinarea drumurilor de cost minim între două noduri ale unui graf), *de neoptimizare* (determinarea numerelor Fibonacci).
3. După numărul de dimensiuni (sau numărul de parametri ai funcțiilor de recurență) pot fi *unidimensionale* (precum numerele lui Fibonacci), sau *multidimensionale* (precum primele implementări de la submulțimea de sumă dată).
4. După numărul deciziilor care se pot lua la un pas pot fi *unidecizionale* (ca numerele lui Fibonacci), sau *multidecizionale* (ca drumul de lungime minimă între două noduri ale unui graf). Problemele unidecizionale sunt de obicei de neoptimizare.
5. După direcția deciziilor putem avea programare dinamică *înainte* în care starea i se calculează în funcție de stările $i+1, i+2, \dots$, sau programare dinamică *înapoi* în care starea i se calculează în funcție de stările $i-1, i-2, \dots$.

PROBLEME

P1. Problema agricultorului

Cantitatea de porumb recoltată depinde de distanța dintre tulipinile a doi tulei de porumb succesivi. Dacă această valoare este mai mică decât o valoare dată x , atunci recolta nu va mai fi maximă. De aceea, agricultorul trebuie să eliminate o parte din porumbii semănați (dintr-acei care au răsărit) astfel încât distanța dintre oricare două tulipini succesișe să fie mai mare sau egală cu valoarea dată x . Pentru simplificare se presupune că porumbul a fost semănat de-a lungul unei singure drepte. Dându-se distanța x , precum și coordonata fiecărui porumb, se cere să se afișeze numărul minim de tulipini care au fost eliminate, astfel încât să se respecte condiția de mai sus. Se știe că primul porumb nu se elimină.

Restricții

Datele sunt citite dintr-un fișier text (PORUMB.IN) cu următoarea structură:

- pe prima linie se găsește distanța x (dată în cm);
- pe a doua linie se află numărul de porumbi semănați (toti porumbii semănați au răsărit) $n \leq 1000$;
- pe a treia linie se află coordonata a_i a fiecărui porumb (dată în cm); coordonatele sunt date în ordine crescătoare ([1]).

Soluție

Să urmărim pașii prezențați în partea teoretică:

1) *Definirea structurii unei soluții optime*: fie un vector c , având lungimea egală cu numărul de porumbi plantați. În acest vector c_i reprezintă numărul maxim de porumbi rămași de la sfârșitul rândului până la al i -lea inclusiv, fără ca cel de pe poziția i să fie eliminat.

2) *Găsirea unei relații de recurență* între termenii sirului:

$$c_n = 1,$$

$$c_i = 1 + \max\{c_k / i < k \text{ și } d(k, i) \geq x\},$$

unde $d(k, i)$ este distanța dintre porumbul i și porumbul k . Pe parcursul calculării valorilor c_i , indicele k variază între $i+1$ și n . Această limită este prețințărgă. E suficient ca valoarea lui k să nu depășească indicele de proveniență a maximului lui c_{i+1} .

3) *Calculul elementelor vectorului* se face pe baza formulei de mai sus.

4) Folosim vectorul poz în care păstrăm poziția de proveniență a maximului lui c_i . Reconstituirea soluției se face recursiv.

Propoziție

Numeralele c_i , $i = 1, \dots, n$ formează un sir descrescător.

Demonstrație

Presupunem prin reducere la absurd că există două numere $c_i < c_j$ cu $j > i$.

Dacă $a_j - a_i \geq x$, atunci presupunerea este falsă din cauza modului de calcul al numerelor c_i .

Dacă $a_j - a_i < x$, atunci fie k poziția de proveniență a maximului lui c_j , adică $c_j = 1 + c_k$. Deoarece $j > i$, avem că $k > i$ și $a_k - a_i \geq x$. Deci c_i va fi cel puțin $1 + c_k = c_j$, deci $c_i \geq c_j$. Contradicție; propoziția este astfel demonstrată.

Consecință

Porumbul 1 nu va trebui niciodată eliminat, deci soluția optimă va fi stocată întotdeauna în c_1 . Următorul exemplu dovedește că eliminarea primului porumb nu duce întotdeauna la soluție optimă:

3
4
2 4 6

Dacă se elimină porumbul 1 (aflat la coordonata 2), atunci în soluția optimă va mai rămâne doar porumbul 4 (sau 6), iar dacă se elimină porumbul 2 (aflat la coordonata 4) atunci soluția optimă este formată din porumbii 1 și 3.

```
Program ProblemaAgricultorului;
const MaxN=1000;
var a,c,poz:array[1..MaxN] of Word;
    x,i,n,ultim:Word; f:Text;

procedure Citire;
begin
  Assign(f,'PORUMB.IN'); Reset(f);
  Readln(f,n); Readln(f,x);
  for i:=1 to n do
    if not SeekEoln(f) then Read(f,a[i]);
  Close(f);
end;

procedure Calcul;
begin
  c[n]:=1; { ultim retine ultima pozitie de provenienta }
  ultim:=n; { a maximului numerelor c[i] }
  for i:=n-1 downto 1 do
    begin
      c[i]:=1;
      while (a[ultim-1]-a[i]>=x) do Dec(ultim);
      if a[ultim]-a[i]>=x
      then begin c[i]:=1+c[ultim]; poz[i]:=ultim end
    end
  end;
end;
```

```
procedure Reconstituie(p:Byte);
begin
  Write(p, ' ');
  if poz[p]<>0 then Reconstituie(poz[p])
end;

Begin
  'Citire;
  Calcul;
  Writeln('Numarul maxim de porumbi ramasi este:',c[1]);
  Write('Numerele de ordine ale porumbilor ramasi sunt:');
  Reconstituie(1)
End.
```

Intrare	Ieșire
5	3
3	1 3 5
1 3 4 6 9	

Propoziția demonstrată anterior conduce la un algoritm *greedy*, care constă în parcugerea sirului porumbilor de la început spre sfârșit și eliminarea acestor porumbi care se află la o distanță mai mică decât x de porumbii deja parcursi. Variabila *ramasi* reține numărul tuleilor rămași. Afisarea numerelor de ordine ale tulipinilor rămase este lăsată pe seama cititorului.

```
Program ProblemaAgricultorului;
const MaxN=1000;
var a:array[1..MaxN] of Word;
    x,i,n,ultim,ramasi:Word;
    f:Text;

procedure Citire;
begin
  Assign(f,'PORUMB.IN'); Reset(f);
  Readln(f,n); Readln(f,x);
  for i:=1 to n do
    if not SeekEoln(f) then Read(f,a[i]);
  Close(f);
end;

procedure Calcul;
begin
  ramsasi:=1;
  ultim:=1;
  for i:=2 to n do
    if a[i]-a[ultim+1]>=x then Inc(ramasi);
    while a[i]-a[ultim+1]>=x do Inc(ultim)
  end;
end;
```

```

Begin
    Citire;
    Calcul;
    Writeln('Numarul maxim de porumbi ramasi este:', ramasi)
End.

```

Discuție

Problema este echivalentă cu *determinarea unei componente intern stabile* în graf (arborele) ale cărui noduri sunt tuleii de porumb, iar muchii există doar între doi tulei între care se află o distanță mai mică decât x . Deoarece tuleii sunt în ordine, (pe un segment de dreaptă) ne aflăm în fața unui proces decizional pas cu pas, deci putem aplica metoda programării dinamice.

Pentru cazul în care fiecărui tuleu de porumb îi se atribuie o valoare pozitivă (numără productivitate) și se cere să se maximizeze productivitatea totală a tuleilor rămași, rezolvarea este asemănătoare: construim un vector c , de dimensiune n , în care pe poziția i reținem profitul maxim pe care îl obținem din porumbii rămași de la sfârșitul rândului până la al i -lea, inclusiv, dacă nu-l eliminăm pe cel de pe poziția i . Relațiile de recurență în acest caz sunt:

$$c_n = \text{profit}_n$$

$c_i = \text{profit}_i + \max\{c_k \mid i < k \text{ și } d(k, i) \geq x\}$, unde prin profit_i am notat profitul obținut de pe urma porumbului i .

În acest caz propoziția demonstrată mai sus nu mai este valabilă. Motivul este următorul: spre deosebire de cazul anterior, aici contează care dintre cei doi porumbi, aflați la o distanță mai mică decât x , este eliminat. Astfel este posibil ca soluția optimă să includă eliminarea primului tuleu, aşa cum se vede din următorul exemplu (datele de intrare sunt asemănătoare cu cele anterioare, doar că pe lângă coordonata unui porumb se mai specifică și profitul obținut de pe urma lui):

```

2
2
3 2
4 10

```

În mod asemănător unui tuleu i se poate atribui o pondere invers proporțională cu distanța dintre el și cel mai apropiat tuleu. Acest lucru se întâmplă de fapt și în realitate (cu cât două rădăcini de porumb sunt mai apropiate, cu atât aceștia se dezvoltă mai greu). În acest caz nu se cere eliminarea tuleilor care se află la o distanță mai mică decât x , ci se cere eliminarea tuleilor astfel încât să obținem o productivitate maximă.

Dacă porumbul s-a semănat pe o suprafață dreptunghiulară (și astfel tuleii nu sunt amplasati de-a lungul unui segment) atunci singura soluție corectă ar fi aplicarea metodei *backtracking*, deoarece determinarea unei componente intern stabile într-un graf planar este o problemă NP-completă ([14]).

P2. Pieze rotite

Se consideră un sir de pieze de *domino*. Fiecare piesă poate fi rotită în jurul centru-lui ei cu 180° . Să se determine secvența de pieze de lungime maximă în care oricare două pieze alăturate au inscrise pe ele același număr: al doilea număr de pe prima piesă coincide cu primul număr inscris pe cea de-a doua.

Restricții

Datele sunt citite din fișierul text DOMINO.IN, având următoarea structură:

- pe prima linie se află numărul de pieze de *domino* (n);
- pe a doua linie se află numerele inscrise pe pieze.

Soluția se va da sub formă unei perechi (i, j) , în care i reprezintă lungimea celei mai mari secvențe, iar j este poziția de început a acestei secvențe ([1]).

Soluție

Vom aplica metoda *programării dinamice*. Introducem următoarele notății:

- (decizia) 0: reprezintă faptul că piesa nu este rotită;
- (decizia) 1: reprezintă faptul că piesa este rotită.

Să urmărim pașii care descriu un algoritm de programare dinamică:

1) *Definirea structurii unei soluții optime*: vom construi o matrice c cu n coloane și două linii în care $c_{i,j}$ reprezintă dimensiunea secvenței de lungime maximă, având ca ultim element piesa i ; aceasta se obține dacă luăm pentru această piesă decizia $j, j = 0, 1$.

2) *Găsirea unei relații de recurență* între termenii matricei:

$$c_{1,0} = c_{1,1} = 1$$

$c_{i,j} = \max\{c_{i-1,k} + d(a_{i,j}, a_{i-1,1-k}), k = 0 \text{ sau } k = 1\}$, unde:

- $a_{i,0}$ este primul număr inscris pe piesa i ,
- $a_{i,1}$ este al doilea număr inscris pe piesa i ,
- prin $d(a_{i,j}, a_{i-1,1-k})$, am notat distanța dintre un număr aflat pe piesa i și un număr aflat pe piesa j definit în felul următor: $d(a_{i,j}, a_{i-1,1-k}) = 1$ dacă cele două numere sunt egale și $d(a_{i,j}, a_{i-1,1-k}) = 0$ în caz contrar.

Pasul 3) îl vom implementa direct în program.

Observație

Dacă prin adăugarea piesei i nu se mărește secvența anterioară, atunci $c_{i,j}$ va primi valoarea 1;

Program PiezeRotitel;

```

const MaxN=1000;
var a,c:array[1..MaxN,0..1] of Word;
    f:Text;
    n,i,j,k,max,poz:Word;

```

```

function d(q,w:Word):Word;
begin
  if q=w then d:=1
  else d:=0
end;

procedure citire;
begin
  Assign(f, 'DOMINO.IN'); Reset(f); Readln(f,n);
  for i:=1 to n do Read(f,a[i,0],a[i,1]);
  Close(f)
end;

procedure calcul;
begin
  c[1,0]:=1;
  c[1,1]:=1;
  for i:=2 to n do
    for j:=0 to 1 do
      begin
        max:=0;
        for k:=0 to 1 do
          if max< c[i-1,k]+d(a[i,j],a[i-1,1-k])
          then max:=c[i-1,k]+d(a[i,j],a[i-1,1-k]);
        if (d(a[i,j],a[i-1,0])=1) or (d(a[i,j],a[i-1,1])=1)
        then c[i,j]:=max
        else c[i,j]:=1
      end
    end;
  begin
    citire;
    calcul;
    max:=1; poz:=1;
    for i:=2 to n do
      for j:=0 to 1 do
        if c[i,j]>max
        then begin max:=c[i,j]; poz:=i end;
    Writeln(max,' ',poz-max+1)
  end;
end...

```

Intrare

5
1 2,3 4,5 3,2 5,6 2.

adică secvența maximă va începe din poziția 3 și va avea lungimea 3:

1 2,3 4,2 5,5 2,2 6.

Ieșire

3,3

Discuție

O variantă a acestei probleme ar putea să ceară în locul secvenței maximale, subșirul maximal.

Soluție

Rezolvarea este asemănătoare cu precedenta și de aceea vom prezenta doar diferențele care apar:

- $c_{i,j}$ este dimensiunea subșirului de lungime maximă având ca ultim element piesa i și care se formează dacă luăm pentru aceasta decizia $j, j = 0,1$.
- $c_{i,j} = \max\{c_{l,k} + d(a_{l,j}, a_{l,k}); 0 \leq k \leq l; 1 \leq l \leq i-1\}$.

```

Program PiezeRotite2;
uses Crt;
const MaxN=1000;
var a,c:array[1..MaxN,0..1] of Word;
  f:Text;
  n,i,j,k,L,max,poz,distanta:Word;

function d(q,w:Word):Word;
begin
  if q=w then d:=1 else d:=0
end;

procedure citire;
begin
  Assign(f, 'DOMINO.IN'); Reset(f); Readln(f,n);
  for i:=1 to n do Read(f,a[i,0],a[i,1]);
  Close(f)
end;

procedure calcul;
begin
  c[1,0]:=1; c[1,1]:=1;
  for i:=2 to n do
    for j:=0 to 1 do
      begin
        max:=1;
        for L:=1 to i-1 do
          for k:=0 to 1 do
            begin
              distanta:=d(a[i,j],a[L,1-k]);
              if distanta=1
              then
                if max< c[L,k]+distanta then max:=c[L,k]+distanta
            end;
        c[i,j]:=max
      end;
    end;
end;

```

```

Begin
  Clrscr;
  citire;
  calcul;
  max:=1; poz:=1;
  for i:=2 to n do
    for j:=0 to 1 do
      if c[i,j]>max then begin max:=c[i,j]; poz:=i end;
  Writeln(max, ' ', poz);
  Readln
End.

```

Intrare Ieșire

3 2 1
1 2 4 5 6 1

Discuție

Rezolvați problema în cazul în care piesele nu se amplasează în sir, ci pe un cerc: după ultima urmează prima.

P3. Tir

La un concurs de *tir*, țintă este alcătuită din n cercuri concentrice, numerotate din exterior spre interior. Fiecare sector determinat de două cercuri succesive îl este atașată o valoare strict pozitivă, reprezentând numărul de puncte pe care le poate primi un concurrent în cazul în care va lovi acest sector.

Se cere să se determine numărul minim de lovitură pe care trebuie să le execute un concurrent pentru a obține exact k puncte.

Restricții

- $n \leq 10$; $k \leq 1000$;
- Exteriorul celui mai mare cerc nu are nici o valoare.

Datele sunt citite din fișierul **TIR.IN**, având următoarea structură:

- pe prima linie este scris numărul de cercuri;
- pe a doua linie este scris numărul de puncte ce urmează a fi obținute de concurrent (k);
- pe a treia linie sunt scrise cele n valori atașate sectoarelor ([1]).

Soluție

Valorile regiunilor le vom păstra în vectorul *valori* ordonate crescător. Trebuie să găsim o modalitate de a-l scrie pe k sub forma unei sume având număr minim de termeni (nu neapărat distincți) din sirul *valori*.

Notăm cu c_i numărul minim de elemente din sirul *valori*, având suma i . Soluția

problemei se obține în c_k . O subproblemă constă în calcularea lui c_i , $i \leq k$.

Aveam următoarele relații:

$c_0=0$,

iar $c_i = 1 + \min \{c_j \mid i = j + valori_k\}$, $k = 1, 2, \dots, n$, $c_j \geq 0$.

Pentru a nu lucra cu numere negative, algoritmul de mai jos pornește marcarea cu valoarea 1, adică $c_0 = 1$. Dacă $c_i = 0$, atunci se consideră că i este nemarcat.

Numerelor c_i sunt calculate astfel: la pasul 0, se marchează $c_0 = 1$; apoi se marchează toți multiplii elementului *valori₁*, urmând ca la pasul j să se marcheze toate numerele care sunt mai mari cu *valori_j* decât valorile marcate la pasul anterior.

```

Program Tir;
  const MaxN=50;
  var valori:array[1..MaxN] of Byte;
  c:array[0..MaxN] of Byte;
  d:array[0..MaxN] of Byte;
  { poziția de provenienta a minimului }
  n,i,j,k:Byte;
  operatii:Boolean;
  f:Text;
procedure Citire;
begin
  Assign(f,'TIR1.PAS'); Reset(f);
  Readln(f,n); Readln(f,k);
  for i:=1 to n do
    if not SeekEoln(f) then Read(f,valori[i]);
  Close(f);
end;
procedure Calcul;
begin
  c[0]:=1;
  for j:=1 to n do
    for i:=0 to k-valori[j] do
      if c[i]>0
      then
        if (c[i+valori[j]]>c[i]+1) or (c[i+valori[j]]=0)
        then begin c[i+valori[j]]:=1+c[i]; d[i+valori[j]]:=i
        end;
end;
procedure Reconstituie(p:Integer);
begin
  if p<>0
  then begin Reconstituie(d[p]);
           Writeln('Folosești valoare:',p-d[p])
  end
end;

```

```

Begin
  Citire;
  Calcul;
  if c[k]=0 then Writeln('Imposibil!')
    else
      begin
        Writeln('Numarul minim de operatii:',c[k]-1);
        Reconstituie(k)
      end
End.

```

O programare dinamică care se bazează pe algoritmul lui Lee este următoarea: La pasul 1 marcăm toate pozițiile $c[\text{valori}[j]]$, $j=1, \dots, N$. La pasul $t+1$ marcăm toate pozițiile care sunt mai mari cu $\text{valori}[j]$, $(j=1, \dots, N)$ decât elementele marcate la pasul t . Variabila t folosită mai sus, indică numărul minim de operații necesare. Noua procedură Calcul este:

```

procedure Calcul;
begin
  c1[0]:=1; c2[0]:=1; max:=0;
  for t:=0 to k do
  begin
    for i:=0 to max do
      if c1[i]>0
      then
        for j:=1 to n do
          if (c1[i+valori[j]]=0) and (i+valori[j]<=k)
          then
            begin
              c2[i+valori[j]]:=1+c1[i];
              d[i+valori[j]]:=i
            end;
    max:=max+valori[n];
    c1:=c2
  end
end;

```

Intrare

5
23
2 3 5 6 8

Ieșire

Numarul minim de operatii: 4
Folosesc valoarea 2
Folosesc valoarea 5
Folosesc valoarea 8
Folosesc valoarea 8

Observăm că pentru testul de mai sus există mai multe soluții. O altă ieșire ar fi: $23 = 5 + 6 + 6 + 6$. Încercați să rezolvați problema astfel încât rezultatul furnizat să conțină numerele cele mai mari pe care le permit datele de intrare.

Cazul în care fiecare sector de cerc poate fi atins cel mult o dată se rezolvă printr-un algoritm de complexitate mai redusă. Pentru simplificare presupunem că ne interesează doar dacă numărul k poate fi obținut ca sumă din numerele valori_i . În cazul acestei verificări vom construi un sir de valori booleene.

Notăm prin c_{ij} faptul că suma i se obține folosind doar primele j elemente din vectorul valori . Este evident că rezultatul final va fi stocat în $c_{k,n}$. Avem $c_{0,j} = \text{true}$, iar $c_{ij} = \text{true} \Leftrightarrow c_{i-\text{valori}_{j-1}} = \text{true}$.

În formula de mai sus se observă că valorile de pe o linie nu depind decât de valorile de pe linia precedentă. Deci nu este nevoie să memorăm întreaga matrice c , ci doar doi vectori reprezentând două linii consecutive din matrice.

Algoritmul de mai jos funcționează astfel: la primul pas se marchează poziția 0, adică vom avea $c_0 = \text{true}$. La al doilea pas se marchează poziția c_{valori_1} cu true . La următorul pas se marchează valorile c_{valori_2} și $c_{\text{valori}_1 + \text{valori}_2}$. La pasul j se marchează toate pozițiile mai mari cu exact valori_j decât toate elementele de pe pozițiile deja marcate.

Reconstituirea soluției o lăsăm în seama cititorului. Pentru aceasta va fi nevoie de memorarea întregii matrice c , sau de o nouă matrice.

```

Program Tir;
const MaxN=50;
var valori:array[1..MaxN] of Byte;
    c1,c2:array[0..MaxN] of Boolean;
    n,i,j,k:Byte;
    operatii:Boolean;
    f:Text;

procedure citire;
begin
  Assign(f,'TIR.IN'); Reset(f);
  Readln(f,n); Readln(f,k);
  for i:=1 to n do
    if not SeekEof(f) then Read(f,valori[i]);
  Close(f)
end;

procedure calcul;
begin
  c1[0]:=true;
  c2[0]:=true;
  for j:=1 to n do
  begin
    for i:=0 to k-valori[j] do
      if c1[i] then c2[i+valori[j]]:=true;
    c1:=c2
  end;
end;

```

```

Begin
  citire;
  calcul;
  if not c1[k] then Writeln('Imposibil!')
    else Writeln('Posibil.')
End.

```

Discuție

Sunt utilizati doi vectori pentru a evita folosirea de mai multe ori a unui oarecare de $valori_i$ într-o descompunere. Complexitatea acestui algoritm este $k \times n$. Dacă numerele din vectorul $valori$ sunt mari, (dar apropriate între ele) putem construi un vector c având ca domeniu intervalul $\min(valori_i), \dots, \sum_i [valori_i - \min(valori_k)]$ (în locul intervalului $0, \dots, \sum_i valori_i$).

Cazul în care fiecare sector k poate fi atins de maxim a_k ori se rezolvă într-un mod asemănător:

```

Program TIR;
const MaxN=50;
var valori,a:array[1..MaxN] of Byte;
  c1,c2:array[0..MaxN] of Boolean;
  n,i,j,k,rep:Byte;
  operatii:Boolean;
  f:Text;

procedure citire;
begin
  Assign(f,'TIR.IN'); Reset(f);
  Readln(f,n); Readln(f,k);
  for i:=1 to n do
    if not SeekEoln(f) then Read(f,valori[i]);
  Close(f)
end;

procedure calcul;
begin
  c1[0]:=true; c2[0]:=true;
  for j:=1 to n do
    for rep:=1 to a[j] do
      begin
        for i:=0 to k-valori[j] do
          if c1[i] then c2[i+valori[j]]:=true;
        c1:=c2;
      end
end;

```

```

Begin
  citire;
  calcul;
  if not c1[k] then Writeln('Imposibil!')
    else Writeln('Posibil.')
End.

```

P4. Transformă numere

Se dau două numere naturale A și B și un vector V , ordonat crescător, având lungimea n , care conține de asemenea numere naturale. Se cere să se determine dacă se poate trece din A în B , știind că singurele operații permise sunt:

- Adunarea la A a oricărui număr din vectorul V .
- Scăderea din A a oricărui număr din vectorul V .

Fiecare număr poate fi adunat, respectiv scăzut de mai multe ori.

Dacă răspunsul la întrebare este afirmativ, se cere numărul minim de operații prin care se poate trece din A în B .

Restricții

Datele sunt citite din fișierul **NUMERE.IN** care are următoarea structură:

- pe prima linie sunt scrise numerele n, A, B ; ($A, B \leq 100$; $n < 100$).
- pe a doua linie sunt scrise elementele vectorului în ordine crescătoare ([1]).

Soluție

Pentru a determina dacă se poate trece din A în B , putem folosi un algoritm de o complexitate mai redusă, și anume:

notăm cu

$$c_i = \begin{cases} \text{true, dacă numărul } i \text{ se poate obține din numărul 0 aplicând operații de tipul} \\ \text{celor descrise} \\ \text{false, în caz contrar} \end{cases}$$

Setăm domeniul vectorului c de la $[-\text{Max}..+\text{Max}]$, unde Max este o valoare suficient de mare.

Atribuim inițial lui c_0 valoarea **true**, apoi $c_j = \text{true} \Leftrightarrow c_i = \text{true} \text{ și } j = i \pm v_k$
Soluția se găsește în c_{b-a} .

Variabila booleană **operatii** este **true** cât timp s-au mai efectuat noi atribuirile elementelor din domeniul $-\text{Max}..+\text{Max}$ al vectorului c . Dacă am fi siguri că avem soluție, condiția de oprire ar fi marcarea elementului c_{b-a} .

Pentru determinarea unei soluții se folosesc încă un vector care reține poziția care a generat marcarea elementului curent, iar afișarea sirului operațiilor necesare se realizează cu procedură recursivă **Reconstituie**:

```

Program TransformaNumere;
const MaxN=9; Max=1000;
var valori:array[1..MaxN] of Byte;
    c:array[-Max..Max] of Boolean;
    d:array[-Max..Max] of Integer;
    n,i,j,a,b:Integer;
    operatii:Boolean;
    f:Text;

procedure Citire;
begin
  Assign(f,'TRANS.IN'); Reset(f);
  Readln(f,n,a,b);
  for i:=1 to n do
    if not(Seekeoln(f)) then Read(f,valori[i]);
  Close(f);
end;

procedure Calcul;
begin
  c[0]:=true;
  operatii:=true;
  while operatii or (not c[b-a]) do
  begin
    operatii:=false;
    for i:=-Max to Max do
      for j:=1 to n do
        begin
          if c[i] and not c[i+valori[j]] and (i+valori[j]<=Max)
          then
            begin
              c[i+valori[j]]:=true;
              d[i+valori[j]]:=i;
              operatii:=true
            end;
          if c[i] and not c[i-valori[j]] and (i-valori[j]>=-Max)
          then
            begin
              c[i-valori[j]]:=true;
              d[i-valori[j]]:=i;
              operatii:=true
            end;
        end;
    end;
  end;
end;

```

```

procedure Reconstituie(p:Integer);
begin
  if p<>0
  then
    begin Reconstituie(d[p]);
      Writeln('Folosesc valoare:',p-d[p])
    end
  end;

Begin
  Citire;
  Calcul;
  if c[b-a]
  then begin Writeln('DA'); Reconstituie(b-a) end
  else Writeln('NU')
End.

```

Răvenim la rezolvarea problemei din enunț, respectiv ne propunem să obținem soluția optimă. Semnificația vectorului c se schimbă: c_i reprezintă numărul minim de operații de adunare și scădere efectuate cu elemente din vectorul $valori$ pentru a se obține valoarea i .

Relațiile de recurență se schimbă astfel:

$$c_0 = 1$$

$$c_i = 1 + \min \{c_{i \pm valori_j}\}$$

De data aceasta bucla principală a programului este executată atât timp cât mai sunt posibile operații de optimizare pe domeniul $-Max \dots +Max$ al vectorului c .

```

Program TransformaNumere;
const MaxN=9;
      Max=1000;
var valori:array[1..MaxN] of Byte;
    c:array[-Max..Max] of Word;
    d:array[-Max..Max] of Integer;
    n,i,j,a,b:Integer;
    f:Text;
    operatii:Boolean;

procedure Citire;
begin
  Assign(f,'TRANS.IN'); Reset(f);
  Readln(f,n,a,b);
  for i:=1 to n do
    if not(Seekeoln(f)) then Read(f,valori[i]);
  Close(f);
end;

```

```

procedure Calcul;
begin
  c[0]:=1; operatii:=true;
  while operatii do
  begin
    operatii:=false;
    for i:=-Max to Max do
      if c[i]>0
      then
        for j:=1 to n do
        begin
          if (i+valori[j]<=Max) and ((c[i+valori[j]]=0) or
            (c[i+valori[j]]>c[i] +1))
          then
            begin
              c[i+valori[j]]:=c[i]+1;
              d[i+valori[j]]:=i; operatii:=true
            end;
          if (i-valori[j]>=-Max) and ((c[i-valori[j]]=0)
            or (c[i-valori[j]]>c[i] +1))
          then
            begin
              c[i-valori[j]]:=c[i]+1;
              d[i-valori[j]]:=i;
              operatii:=true
            end
          end
        end
      end;
  end;
procedure Reconstituie(p:Integer);
begin
  if p<>0
  then
  begin
    Reconstituie(d[p]);
    Writeln('Folosesc valoare:',p-d[p])
  end
  end;
begin
  Citire;
  Calcul;
  if c[b-a]>0
  then begin Writeln('DA'); Reconstituie(b-a) end
  else Writeln('NU')
end.

```

Intrare

2 0 3
2 7

Ieșire

DA
Folosesc valoare:-2
Folosesc valoare:-2
Folosesc valoare:7

Discuție

Era posibilă și o rezolvare prin programare dinamică, bazată pe algoritmul lui Lee: se marchează toate pozițiile la care se poate ajunge printr-o operație, apoi toate pozițiile la care se poate ajunge prin două operații etc. În acest caz condiția de oprire ar fi fost tot una euristică: numărul de iterații să nu depășească o valoare fixă.

Varianta de rezolvare care folosește fiecare număr o singură dată, respectiv algoritmul care determină dacă trecerea de la A la B este posibilă, este tot de programare dinamică și constă în calcularea unei matrice c, unde c_{ij} este true dacă valoarea i poate fi obținută folosind doar numerele v_1, \dots, v_j . Soluția finală se găsește în $c_{b-a,n}$.

Următorul algoritm folosește doar doi vectori (c1 și c2) în locul matricei c:

```

Program TransformaNumere;
const MaxN=9;
var valori:array[1..MaxN] of Byte;
    c1,c2:array[-MaxN..MaxN] of Boolean;
    d:array[-MaxN..MaxN] of Integer;
    n,i,j,max,a,b:Integer;
    f:Text; operatii:Boolean;
procedure Citire;
begin
  Assign(f,'TRANS.IN'); Reset(f); Readln(f,n,a,b);
  for i:=1 to n do
    if not(Seekolein(f)) then Read(f,valori[i]);
  Close(f);
end;
procedure Calcul;
begin
  max:=0; c1[0]:=true; c2:=c1;
  for j:=1 to n do
  begin
    for i:=-max to max do
    begin
      if c1[i] and (not c2[i+valori[j]])
      then
        begin
          c2[i+valori[j]]:=true;
          d[i+valori[j]]:=i;
          operatii:=true
        end;
    end;
  end;
  if operatii then
    begin
      if c2[b-a]
      then Writeln('DA')
      else Writeln('NU')
    end;
end;

```

```

if c1[i] and not c2[i-valori[j]] and (i-valori[j]>=max)
then
begin
  c2[i-valori[j]]:=true;
  d[i-valori[j]]:=i;
  operatii:=true
end
end;
c1:=c2; max:=max+valori[n]
end
end;

procedure Reconstituie(p:Integer);
begin
  if p<>0
  then
  begin
    Reconstituie(d[p]);
    Writeln('Folosesc valoare:',p-d[p])
  end
  end;
begin
  Citire;
  Calcul;
  if c1[b-a] then begin Writeln('DA'); Reconstituie(b-a) end
  else Writeln('NU');
end.

```

Discuție

Pe lângă operațiile de adunare și scădere se pot introduce și alte tipuri de operații, de exemplu adunarea la suma curentă a unei submulțimi de numere. În acest caz se cere tot numărul minim de operații care să conducă la rezultat.

Problema s-ar mai putea generaliza dacă fiecare operație i se atașează un cost și se cere să se realizeze o trecere din A în B cu cost minim.

O altă formă a enunțului de mai sus este următoarea: avem la dispoziție n tipuri de segmente (un tip de segment este specificat prin lungimea sa); din fiecare tip de segment avem la dispoziție suficiente bucăți. Trebuie să se construiască un drum între două puncte A și B aflate pe aceeași dreaptă. La fiecare pas se pune un segment cu un capăt obligatoriu la capătul drumului construit până în prezent, iar celălalt pe dreapta determinată de AB . Astfel, segmentele se pot suprapune, iar capetele lor pot ajunge în afara segmentului AB (dar tot pe dreapta determinată de AB).

Tinând cont de faptul că $cmmdc(x,y) = cmmdc(x,y-x) = cmmdc(x,x+y)$, se mai observă că A și B nu pot fi unite decât dacă distanța dintre ele este multiplu al celui mai mare divizor comun al celor n numere. Condiția este necesară dar nu și suficientă.

Dacă A și B nu se află pe o aceeași dreaptă, atunci problema se rezolvă asemănător, mărind însă numărul de dimensiuni. Pentru determinarea existenței soluției vom lucra pe o matrice de dimensiuni $[-Max..Max, -Max..Max]$, adică pe pătratul de latură $2*Max$, centrat pe $(0,0)$.

În G.M. nr.3/1989, Florin Vulpescu-Jalea propune următoarea problemă: fie d un număr natural strict pozitiv care se poate scrie în mod unic sub forma $d = \sqrt{a^2 + b^2}$. Să se arate că oricare două puncte din plan pot fi unite printr-o linie poligonală având toate vârfurile de coordonate întregi și laturile de lungime d , dacă și numai dacă $cmmdc(a,b) = 1$. Găsiți un algoritm care determină o astfel de linie poligonală.

Să considerăm o variantă a problemei în plan în care segmentele se înlocuiesc cu *corniere* (un cornier este alcătuit din două segmente de dreaptă care sunt unite între ele, astfel încât la un capăt formează un unghi drept). Dându-se două puncte A și B și un set de corniere (specificate prin lungimile celor două segmente care intră în alcătuire) se cere să se determine numărul minim de corniere necesare pentru a uni cele două puncte.

O aplicație a acestei probleme poate fi următoarea: să se realizeze instalația de apă a unei case, folosind țevi (de lungimi date) și corniere (având ambele dimensiuni date). Dându-se poziția punctului prin care intră țeava cu apă în curte și poziția unde se va instala un robinet, se cere lungimea minimă de țeavă și cornier folosite pentru a lega intrarea de robinet. Se știe în plus că țevile nu au voie să străbată nici o cameră (care se va da sub forma unui dreptunghi).

Problema devine mai complexă dacă adăugăm constrângeri de tipul:

- țevile nu au voie să fie tăiate;
- țevile trebuie să aibă un număr minim de tăieturi;
- în cazul în care se cere construirea mai multor robinete (deci în anumite puncte ale rețelei cu apă vor trebui create bifurcații) se cere lungimea minimă de țeavă necesară pentru a lega toate robinetele. Această problemă admite o soluție asemănătoare, realizată de asemenea cu metoda programării dinamice.

Problema în care segmentul este generalizat la dreptunghi și se cere numărul minim de dreptunghiuri necesare pentru a acoperi exact o suprafață dată, este NP-completă.

P5. Stive de monede

Fie un sir de n stive de monede. Fiecare stivă i are inițial a_i monede. Singura operație permisă este mutarea unei monede de pe o stivă pe o altă stivă. Să se treacă (dacă este posibil) într-o configurație în care diferența dintre numărul de monede de pe oricare două stive consecutive este $+1$ sau -1 . Se cere o soluție cu număr minim de mutări.

Restricții

- $n \leq 10$; $a_i \leq 1000$, pentru $i \geq 1$ și $a_1 \geq n$;
- pe prima stivă nu se pot pune monede;
- de pe prima stivă nu se pot lua monede.

Datele sunt citite din fișierul MONEDA.IN având următoarea structură:

- pe prima linie este scris numărul de stive (n);
- pe a două linie sunt scrise numere a_i care reprezintă numărul de monede din fiecare stivă ([1]).

Soluție

În acest caz, necesitatea utilizării metodei programării dinamice nu este evidentă, deci, în prealabil, sunt necesare câteva prelucrări. Observăm mai întâi că dacă prima stivă rămâne neschimbată, atunci pe a doua stivă vom putea avea doar $a_1 - 1$ sau $a_1 + 1$ monede, pe a treia stivă vom putea avea doar $a_2 - 1$ sau $a_2 + 1$ monede etc. Acest lucru ne sugerează construirea unui arbore binar având n niveluri în care informația atașată fiecărui nod de pe nivelul k (considerând că nivelul 1 este cel mai de sus) reprezintă numărul de monede care trebuie puse sau luate de pe stiva k , astfel încât să fie îndeplinită condiția din enunț. Prin convenție, informația atașată unui nod este negativă dacă de pe stiva respectivă se iau monede și pozitivă dacă pe stiva respectivă se pun monede.

Construcția unui astfel de arbore se realizează în felul următor: pe nivelul 1 vom avea un singur nod corespondător stivei 1. Informația atașată acestui nod este numărul 0, deoarece pe stiva 1 nu vom pune și nu vom lua monede. Apoi, pentru orice alt nivel i , numărul atașat unui nod de pe acel nivel va fi $a_{i-1} + c_{i-1,j} - 1 - a$; sau $a_{i-1} + c_{i-1,j} + 1 - a$, unde j este numărul de ordine al nodului de proveniență în cadrul nivelului $i - 1$.

Arboarele nu îl vom reprezenta în *Heap*, ci într-o matrice cu n linii și 2^n coloane (pe prima linie vom avea nodul rădăcină, pe a doua linie nodurile adjacente cu nodul de pe linia 1 și.a.m.d.). Astfel, problema inițială se reduce la determinarea unui drum de cost minim între rădăcină și o frunză, drum pe care îl vom afla *parcugând arborele de jos în sus*. Vom reține într-o matrice c (cu n linii și 2^n coloane) pentru fiecare nod, costul minim al unui drum ce unește nodul respectiv cu una din frunze. Costul minim al oricărui nod va fi minimul dintre costurile atașate nodurilor adjacente lui (și aflate mai jos în arbore) la care se adună numărul de monede care trebuie luate (așezate) de pe (pe) stiva respectivă.

În final, prima poziție a matricei va conține dublul numărului de mutări care trebuie efectuate, deoarece în momentul în care am calculat valoarea drumului am adunat și monedele care trebuie așezate și cele care trebuie luate.

Condiția de existență a unei soluții: pe un drum care unește rădăcina cu una din frunze, numărul de monede luate trebuie să fie egal cu numărul de monede puse; de aceea, în momentul construirii matricei vom păstra această informație în variabila $c[i,j].ok$.

```
Program StiveDeMonede;
const MaxN=10;
type art=record
  as,suma:Integer;
  ok:Boolean
end;
```

```
var a,b:array[1..MaxN] of Word;
  c:array[1..MaxN,1..1024] of art;
  n,i,j,p:Word;
procedure citeste;
  var f:Text;
begin
  Assign(f,'MONEDA.IN'); Reset(f); Readln(f,n);
  for i:=1 to n do if not SeekEoln(f) then Read(f,a[i]);
  Close(f);
end;
procedure ConstruiesteMatrice;
begin
  c[1,1].as:=0; c[1,1].suma:=0; p:=1;
  for i:=2 to n do
    begin
      for j:=1 to p do
        begin
          c[i,2*j-1].as:=a[i-1]+c[i-1,j].as-1-a[i];
          c[i,2*j].as:=a[i-1]+c[i-1,j].as+1-a[i];
          c[i,2*j-1].suma:=c[i-1,j].suma+c[i,2*j-1].as;
          c[i,2*j].suma:=c[i-1,j].suma+c[i,2*j].as;
        end;
      p:=2*p;
    end;
  for j:=1 to p do c[i,j].ok:=c[i,j].suma=0;
  p:=p div 2;
  for i:=n-1 downto 1 do
    begin
      for j:=1 to p do
        if c[i+1,2*j-1].ok
        then
          if c[i+1,2*j].ok
          then
            if Abs(c[i+1,2*j-1].as)<Abs(c[i+1,2*j].as)
            then
              begin
                c[i,j].ok:=true;
                c[i,j].as:=Abs(c[i,j].as)+Abs(c[i+1,2*j-1].as);
              end
            else
              begin
                c[i,j].ok:=true;
                c[i,j].as:=Abs(c[i,j].as)+Abs(c[i+1,2*j].as);
              end
            else

```

```

begin
  c[i,j].ok:=true;
  c[i,j].as:=Abs(c[i,j].as)+Abs(c[i+1,2*j-1].as)
end
else
  if c[i+1,2*j].ok
  then
    begin
      c[i,j].ok:=true;
      c[i,j].as:=Abs(c[i,j].as)+Abs(c[i+1,2*j].as)
    end;
  p:=p div 2
end
end;

procedure Reconstruie;
begin
  b[1]:=a[1]; j:=1;
  for i:=2 to n do
    if c[i,2*j-1].ok
    then
      if c[i,2*j].ok
      then
        if c[i,2*j-1].as< c[i,2*j].as
        then begin j:=2*j-1; b[i]:=b[i-1]-1 end
        else begin j:=2*j; b[i]:=b[i-1]+1 end
      else begin j:=2*j-1; b[i]:=b[i-1]-1 end
    else
      if c[i,2*j].ok
      then
        begin j:=2*j; b[i]:=b[i-1]+1 end
  end;

procedure AfiseazaMutari;
begin
  for i:=2 to n do
    while a[i]>b[i] do
    begin
      j:=2;
      while (j<=n) and (b[j]<=a[j]) do Inc(j);
      Dec(a[i]); Inc(a[j]);
      Writeln('De pe stiva ',i,' pe stiva ',j)
    end
  end;
begin
  citeste;
  ConstruiesteMatrice;

```

```

if not c[1,1].ok
then Writeln('Nu exista solutie.')
else
begin
  Writeln('Numarul minim de mutari este:',c[1,1].as div 2);
  Reconstituie;
  AfiseazaMutari
end
End.

```

Intrare

10
12 15 13 14 17 10 12 10 12 10

Iesire

Numarul minim de mutari: 7
 De pe stiva 2 pe stiva 3
 De pe stiva 2 pe stiva 6
 De pe stiva 4 pe stiva 6
 De pe stiva 5 pe stiva 6
 De pe stiva 5 pe stiva 8
 De pe stiva 5 pe stiva 10
 De pe stiva 9 pe stiva 10

Discuție

Încercați să optimizați programul de mai sus ținând cont de faptul că numerele $c[i,j].as$ pot fi calculate direct fără a fi necesară construirea matricei c . Rezolvați problema în cazul în care se pot muta oricătre monede de pe o stivă pe alta.

**P6. Segmente intersectante**

Se dă n segmente aflate pe o dreaptă. Să se determine cardinalul maxim al unei submulțimi de segmente care are proprietățile:

- 1) oricare două segmente din submulțime nu se intersectează;
- 2) submulțimea conține primul segment.

Restricții

Datele se citesc din fișierul SEGMENTE.IN, având următoarea structură:

- pe prima linie este scris numărul de segmente ($n \leq 1000$);
- pe următoarele linii se dau perechi de numere, reprezentând coordonatele punctelor de început și de sfârșit ale segmentelor. Segmentele sunt ordonate crescător după coordonata de început ([1]).

Soluție

Dacă reținem în c_i cardinalul maxim al unei submulțimi de segmente care conține segmentul i și în plus conține doar segmente aflate la dreapta acestui segment, atunci în c_i vom avea soluția problemei. Formula de calcul a numerelor c_i este:

$$c_i = 1 + \max\{c_j \mid j > i \text{ și segmentul } i \text{ nu intersectează segmentul } j\}.$$

În cazul în care segmentul respectiv intersectează toate segmentele aflate în dreapta lui, cardinalul maxim este 1.

Program SegmenteIntersectate;

```

const MaxN=1000;
type segment=record
    inceput,sfarsit:Word
end;
var a:array[1..MaxN] of segment;
    c:array[1..MaxN] of Word;
    max,n,i,j:Word;
procedure citeste;
var f:Text;
begin
    Assign(f,'SEGMENTE.IN'); Reset(f); Readln(f,n);
    for i:=n downto 1 do Readln(f,a[i].inceput,a[i].sfarsit);
    Close(f);
end;
function intersect(g,h:Word):Boolean;
begin
    intersect:=(a[g].sfarsit>=a[h].inceput)
end;
procedure calcul;
begin
    for i:=n downto 1 do
    begin
        max:=1;
        for j:=i+1 to n do
            if not intersect(i,j)
            then
                if (max=1) or (max<1+c[j]) then max:=1+c[j];
        c[i]:=max
    end
end;
Begin
    citeste;
    calcul;
    Writeln('Numarul maxim de segmente este:',c[1])
End.

```

Intrare Ieșire

3	2
4	6
3	5
1	3

Discuție

Dacă se poate elimina și primul segment, problema se rezolvă identic, iar soluția se găsește în c_k , unde $c_k = \max c_j$.

O altă variantă constă în atribuirea unei ponderi fiecărui segment și se cere o soluție de pondere maximă.

Ponderea unei soluții este definită ca fiind suma ponderilor segmentelor care o alcătuiesc. Aici avem două situații:

1) dintre toate soluțiile cu număr maxim de segmente trebuie să alegem una și de pondere maximă. În acest caz pe lângă vectorul c trebuie să construim un vector p , în care p_i reprezintă ponderea maximă a unei submulțimi care conține c_i segmente. Numerele p sunt calculate astfel:

$$p_i = \text{pondere}_i + \max\{ p_j \mid j > i, \text{ segmentul } i \text{ nu se intersectează cu segmentul } j \text{ și } c_j = 1 + c_i \}$$

2) trebuie să determinăm o soluție de pondere maximă indiferent dacă aceasta are sau nu un număr maxim de segmente. În acest caz vom lucra cu un singur vector c , în care pe poziția k reținem ponderea maximă a unei submulțimi de segmente ce include segmentul k și segmente aflate la dreapta acestui segment. Modul de calcul al numerelor c este următorul:

$$c_n = \text{pondere}_n,$$

$$c_i = \text{pondere}_i + \max\{ c_j \mid j > i \text{ și segmentul } i \text{ nu intersectează segmentul } j \}$$

P7. Treceri de pietoni

Într-un oraș există un sistem de străzi cu *structură arborescentă*. Astfel există o singură stradă în *centrul* orașului de la care pleacă străzile, ramificându-se până la periferia orașului. Să se determine numărul minim de treceri de pietoni care trebuie amplasate astfel încât pe cel puțin una din oricare două străzi consecutive (adiacente) să existe o trecere de pietoni. Pe strada din centru nu trebuie amplasată nici o trecere de pietoni.

Restricții

Datele se citesc dintr-un fișier text, având următoarea structură:

- pe prima linie este scris numărul de străzi ($n \leq 100$);
- pe următoarele n linii se găsește structura arborelui (sistemu de străzi) a cărui rădăcină este nodul 1; pe fiecare linie $i+1$ sunt scrise străzile vecine cu strada i , situate pe nivelul următor ([1]).

Solutie

Dinamica nu este evidentă în acest caz, de aceea vom reformula enunțul problemei în felul următor: să se determine o submulțime maximală de străzi pe care nu se vor amplasa treceri de pietoni. Această submulțime are următoarele proprietăți:

- conține stradă din centrul orașului;
- oricare două străzi din submulțime nu sunt adiacente.

Cu alte cuvinte, trebuie să determinăm o *componentă intern stabilă maximală* (în raport cu inclusiunea) care are un număr maxim de străzi și care conține strada din centrul orașului (nodul rădăcină al arborelui).

S-a precizat la începutul acestui capitol că problemele de programare dinamică conțin subprobleme comune și în plus, aceste subprobleme seamănă cu problema principală. În cazul problemei de față, subproblemă se definește (pentru fiecare subarbore) ca fiind determinarea unei componente interne stable maxime (indusă de subarborele respectiv) care conține nodul rădăcină (al acelui subarbore).

În prima fază vom ordona arborele pe niveluri. Structura arborelui pe niveluri o vom reține în matricea *niv* (pe linia *i* se află toate nodurile de pe nivelul *i*), apoi vom defini o matrice *c* în care *c_{ij}* este soluția unei subprobleme, adică în *c_{ij}* vom reține cardinalul maxim al unei componente interne stabile maxime indușă de subarborele care are ca rădăcină nodul *niv_{i,j}*. Vom calcula aceste numere în felul următor: dacă nodul *niv_{i,j}* este terminal, atunci *c_{ij}* = 1, iar în caz contrar *c_{ij}* = 1 + numărul nodurilor terminale

$$\text{de pe nivelul } (i+2) + \sum_{\substack{niv_{i+2,h} \\ \text{neterminal}}} \max \left\{ c_{i+2,h}, \sum_{\substack{niv_{i+3,g} \\ \text{fiu al lui } niv_{i+2,h}}} c_{i+3,g} \right\}.$$

Formula de mai sus are următoarea semnificație: componenta intern stabilă într-un arbore conține nodul rădăcină. Deci nici unul din nodurile aflate pe următorul nivel (*i* + 1) nu vor apartine submulțimii cerute. Se adaugă apoi toate nodurile terminale de pe nivelul *i* + 2. Deoarece într-un arbore nu se știe dacă componenta intern stabilă maximală conține sau nu nodul rădăcină, vom adăuga la submulțimea construită până în prezent suma maximelor dintre *c_{i+2,h}* și suma numerelor *c_{i+3,g}*, unde nodurile *niv_{i+3,g}* sunt fiile nodului *niv_{i+2,h}*.

Program Trecere_De_Pietoni;

```

const MaxN=125;

var niv,c:array[1..MaxN,1..MaxN] of -1..MaxN;
    a:array[1..MaxN] of set of 1..MaxN;
    L:array[1..MaxN] of 0..MaxN;
    { numarul de strazi de pe fiecare nivel }
    n,i,j,k,max,g,h,q,nrniv,v,pers,ma,s:Byte;
    niv2,niv3:set of 1..MaxN;

```

```

procedure citeste;
var f:Text;
    q:1..MaxN;
begin
    Assign(f,'PIETONI.IN'); Reset(f); Readln(f,n);
    for i:=1 to n do
    begin
        a[i]:=[];
        while not SeekEoln(f) do
        begin Read(f,q); a[i]:=a[i]+[q] end;
        Readln(f)
    end;
    Close(f)
end;

procedure Ordonare_pe_niveluri;
begin
    L[1]:=1;
    niv[1,1]:=1;
    pers:=1;
    nrniv:=1;
    while pers<>n do
    begin
        Inc(nrniv); L[nrniv]:=0;
        for i:=1 to L[nrniv-1] do
            for j:=1 to n do
                if j=niv[nrniv-1,i]
                then
                    for k:=1 to n do
                        if k in a[j]
                        then
                            begin
                                Inc(L[nrniv]);
                                niv[nrniv,L[nrniv]]:=k;
                                Inc(pers)
                            end
                end
    end;
end;

procedure calcul;
begin
    for i:=nrniv downto 1 do
        for j:=1 to L[i] do
        begin
            { calcularea pentru fiecare strada de pe nivelul i }
            { a maximului de strazi neadiacente din subarborele }
            { generat de strada niv[i,j] }
        end;
end;

```

```

max:=1; niv2:=[];
{ in variabila niv2 se retin strazile de pe nivelul actual+2 }
for k:=1 to n do
  if k in a[niv[i,j]] then niv2:=niv2+a[k];
  niv3:=[]; { in variabila niv3 se retin strazile de pe }
             { nivelul actual+3 }

for k:=1 to n do
  if k in niv2 then niv3:=niv3+a[k];
for k:=1 to n do
  if k in niv2
  then
    if a[k]=[]
    then Inc(max)
    else
    begin
      v:=1;
      while niv[i+2,v]<>k do Inc(v);
      ma:=c[i+2,v]; s:=0;
      for g:=1..to n do
        if g in a[niv[i+2,v]]
        then
        begin
          h:=1;
          while g<>niv[i+3,h] do Inc(h);
          s:=s+c[i+3,h];
        end;
        if ma>s then max:=max+ma else max:=max+s
      end;
      c[i,j]:=max
    end;
  end;
begin.
  citeste;
  Ordonare_pe_niveluri;
  calcul;
  Writeln('Numarul minim de treceri de pietoni este:',n-c[1,1])
end.

```

Intrare

```

13
2 3 4
5 6 7
8
9 10 11
12
13

```

Ieșire

```

Numarul minim de treceri de pietoni este:5

```

P8. Problema polițiștilor

Într-un oraș există un sistem de străzi cu *structură arborescentă*. Astfel există o singură stradă în *centrul* orașului de la care pleacă străzile, ramificându-se până la periferia orașului.

Pentru a preveni delictele, Poliția orașului decide să amplaseze pe străzi un număr de polițiști, astfel încât fiecare stradă să poată fi *supravegheată* de cel puțin un polițist. Sunem că un polițist amplasat pe strada i supraveghetă strada j , dacă $i=j$ sau dacă de pe strada j se poate trece direct pe strada i (nodurile i și j sunt adiacente în arbore). Se cere să se determine numărul minim de polițiști necesari, știind că pe strada din centrul orașului a fost amplasat un singur polițist.

Restricții

Datele se citesc dintr-un fișier text, având următoarea structură:

- pe prima linie este scris numărul de străzi ($n \leq 100$);
- pe următoarele n linii se găsește structura arborelui (sistemu de străzi) a căruia rădăcină este nodul 1: pe fiecare linie $i+1$ sunt scrise străzile vecine cu strada i , situate pe nivelul următor ([1]).

Soluție

Ideea de rezolvare este asemănătoare cu cea de la problema P7. Subproblemele în acest caz sunt: calcularea pentru fiecare subarbore a unei submulțimi de noduri care are următoarele proprietăți:

- conține nodul rădăcină al subarborelui respectiv;
- are număr minimum de noduri;
- orice nod al subarborelui respectiv care nu face parte din acea submulțime este adjacente cu cel puțin un nod din acea submulțime.

În prima fază ordonăm (ca și la problema P7) străzile pe niveluri. Apoi vom construi o matrice c în care c_{ij} este cardinalul submulțimii de noduri (care are proprietățile descrise anterior) a subarborelui având rădăcină nodul $niv_{i,j}$.

Aveam:

$$c_{ij} = 1 + \sum_{niv_{i,k,h} \text{ neterminal}} \min \left\{ c_{i+1,h}, \sum_{niv_{i+2,k} \text{ fiu al lui } niv_{i+1,h}} c_{i+2,g} \right\}.$$

Program Problema_Politistilor;

```

const MaxN=125;
var niv,c:array[1..MaxN,1..MaxN] of -1..MaxN;
a:array[1..MaxN] of set of 1..MaxN;
L:array[1..MaxN] of 0..MaxN;
niv2,niv3:set of 1..MaxN;
min,s,pers,nrniv,i,j,k,n,h,g:Byte;

```

```

procedure citeste;
var f:Text;
q:1..MaxN;
begin
Assign(f,'POLITIST.IN'); Reset(f); Readln(f,n);
for i:=1 to n do
begin
a[i]:=[];
while not SeekEoln(f) do
begin Read(f,q); a[i]:=a[i]+[q] end;
Readln(f);
end;
Close(f);
end;

procedure Ordonare_pe_niveluri;
begin
L[1]:=1; niv[1,1]:=1; pers:=1; nrniv:=1;
while pers<>n do
begin
Inc(nrniv); L[nrniv]:=0;
for i:=1 to L[nrniv-1] do
for j:=1 to n do
if j=niv[nrniv-1,i]
then
for k:=1 to n do
if k in a[j]
then
begin
Inc(L[nrniv]);
niv[nrniv,L[nrniv]]:=k;
Inc(pers)
end
end
end;
procedure calcul;
begin
for i:=nrniv downto 1 do
for j:=1 to L[i] do
begin
min:=1;
for k:=1 to n do
if (k in a[niv[i,j]]) and (a[k]<>[])
then
begin
niv2:=a[k];

```

```

s:=0;
for h:=1 to n do
if h in niv2
then
begin
g:=1;
while niv[i+2,g]<>h do Inc(g);
Inc(s,c[i+2,g])
end;
g:=1;
while niv[i+1,g]<>k do Inc(g);
if s<c[i+1,g]
then min:=min+s
else min:=min+c[i+1,g]
end;
c[i,j]:=min
end
end;

Begin
citeste;
ordonare_pe_niveluri;
Calcul;
Writeln('Numărul minim de politisti necesari:',c[1,1])
End.

```

Intrare

Ieșire

Numarul minim de politisti necesari:5

9

2 3

4 5

9

6 7

8

P9. Reconstituie fraza

Se dă o frază în care lipsesc spațiile dintre cuvinte. Se cere să se reconstituie fraza pe baza unui dicționar de cuvinte. În cazul în care există mai multe soluții se cere cea cu număr minim de cuvinte.

Restrictii

Datele se citesc din fișierul text FRAZA.IN, având următoarea structură:

- pe prima linie este scris numărul de litere din frază (n);
- pe a doua linie este scrisă fraza din care s-au pierdut spațiile;
- pe următoarele n linii se găsesc cuvintele din dicționar (câte unul pe linie) ([1]).

Soluție

Construim un vector c având $n+1$ elemente cu următoarea semnificație: c_i este numărul minim de cuvinte al subfrazei care începe cu caracterul de pe poziția 1 și se termină cu caracterul de pe poziția i . În cazul în care această subfrază nu conține un număr întreg de cuvinte (adică există litere care nu formează un cuvânt) vom avea $c_i = -1$. În final în c_n vom avea numărul minim de cuvinte al frazei citite de la intrare.

Modul de calcul al vectorului c este următorul: $c_0 = 0$, apoi $c_i = 1 + \min\{c_j, j = 1, \dots, i-1\}$, iar $c_i \neq -1$ și literele începând cu poziția $j+1$ până la poziția i formează un cuvânt din dicționar.

În scopul de a simplifica afișarea soluției vom reține într-un vector poz poziția de proveniență a minimului elementelor c_i .

Program ReconstituieFraza;

```

const MaxN=10;
var c, poz:array[0..MaxN] of -1..MaxN;
    dic:array[1..MaxN] of string;
    n:1..MaxN;
    fraza:string;
    min, i, j:Integer;

procedure citeste;
    var f:Text;
begin
    Assign(f,'DICTION.IN'); Reset(f); Readln(f,n);
    Readln(f,fraza);
    for i:=1 to n do Readln(f,dic[i]);
    Close(f)
end;

function cuvant(st,sf:Integer):Boolean;
    var s:string;
        v:1..MaxN;
begin
    s:=Copy(fraza,st,sf-st+1);
    for v:=1 to n do
        if dic[v]=s
        then
            begin
                cuvant:=true; Exit
            end;
        cuvant:=false
end;

```

```

procedure calcul;
begin
    c[0]:=0;
    poz[0]:=-1;
    for i:=1 to n do
begin
    min:=-1;
    poz[i]:=0;
    for j:=0 to i-1 do
        if cuvant(j+1,i)
        then
            if ((min=-1) or (min>i+c[j])) and (c[j]<>-1)
            then
                begin
                    min:=i+c[j];
                    poz[i]:=j
                end;
            c[i]:=min
        end
end;
procedure Afiseaza(k:Byte);
    var t:Integer;
begin
    t:=poz[k];
    if t<>-1
    then
        begin
            Afiseaza(t);
            for i:=t+1 to k do
                Write(fraza[i]);
            Write(' ')
        end
    end;
end;
Begin
    citeste;
    calcul;
    if c[n]=-1
    then Writeln('Nu există soluție!!')
    else
        begin
            Writeln('Numarul minim de cuvinte este:',c[n]);
            Write('Fraza este următoarea:');
            Afiseaza(n)
        end
End.

```

Intrare

```

10
afostodata
q
w
ert
a
de
tare
fost
el
odata
asd

```

Ieșire

Numarul minim de cuvinte este:3
Fraza este urmatoarea: a fost odata.

P10. Plata datorilor

O întreprindere se restructurează după un plan, având n etape. La fiecare etapă i , ($1 \leq i \leq n$) întreprinderea împrumută de la o bancă suma $a[i]$. Odată făcute cele n împrumuturi, întreprinderea trebuie să restituie sumele împrumutate. După restituirea primului împrumut conducerea întreprinderii își dă seama că nu mai poate restitui toate sumele împrumutate, ci doar aceleia care nu au fost împrumutate în etape succesive. Să se determine suma maximă pe care o poate recupera banca.

Restricții

Datele se citesc din fișierul text BANCA.IN, având următoarea structură:

- pe prima linie este scris numărul n , ($n \leq 1000$);
- pe a doua linie sunt scrise sumele $a[i]$ despărțite prin spații ([]).

Soluție

În cazul în care $a_i = 1$, ($1 \leq i \leq n$) această problemă este un caz particular al problemei determinării unei *componente intern stabile maximale* (care conține nodul rădăcină) într-un graf linie și se rezolvă prin metoda *greedy* aşa cum este descrisă în discuția de final.

Pentru cazul de față vom da o soluție bazată pe programare dinamică, și anume: vom defini un vector c cu n componente în care c_i reprezintă suma maximă de bani pe care banca o poate recupera în etape, având numărul de ordine mai mare decât i (respectând condiția de neconsecutivitate) și care conține suma din etapa i . În final în c_1 vom avea soluția problemei.

Componentele vectorului c se calculează pe baza formulei:

$$c_i = a_i + \max\{c_j, j > i+1\}.$$

Se vor afișa și etapele care au generat suma maximă plătită de întreprindere. Pentru a ușura afișarea etapelor care au generat maximul, se va reține în vectorul pozitie poziția de proveniență a maximului numerelor c_i .

```

Program PlataDatorilor;
const MaxN=1000;
var a,pozitie:array[1..MaxN] of Word;
    c:array[1..MaxN] of Longint;
    n,i,j,poz:Word; max:Longint;
    f:Text;

procedure citire;
begin
  Assign(f,'DATORIE.IN'); Reset(f); Readln(f,n);
  for i:=1 to n do
    if not SeekEoln(f) then Read(f,a[i]);
  Close(f);
end;

procedure calcul;
begin
  c[1]:=a[1]; pozitie[1]:=0;
  for i:=2 to n do
    begin
      max:=0;
      for j:=1 to i-2 do
        if max<c[j] then begin max:=c[j]; poz:=j end;
      if max>>0 then begin c[i]:=max+a[i]; pozitie[i]:=poz end
        else c[i]:=0
    end;
end;

procedure Afiseaza(h:Word);
var q:Word;
begin
  q:=pozitie[h];
  if q<>0 then Afiseaza(q);
  Write(h, ' ')
end;

Begin
  citire;
  calcul;
  max:=c[1]; poz:=1;
  for i:=2 to n do
    if c[i]>max
    then
      begin max:=c[i]; poz:=i end;
  Writeln('Suma maxima este:',max,'.');
  Afiseaza(poz);
  Readln
End.

```

Intrare

6
1 3 6 2 4 3

Ieșire

Suma maxima este 11.

Discuție

La prima vedere am fi tentați să dăm o soluție prin metoda *greedy*, care constă în plăta datorilor de ordin impar. Exemplul de mai sus este elovent în acest sens, dar pentru intrarea

4
2 3 4 100

metoda *greedy* nu ar mai furniza soluția optimă.

P11. Deplasează robot

Se dă un labirint reprezentat printr-o matrice de dimensiuni $n \times m$, având elemente în mulțimea $\{0, 1\}$ (0 =cameră, 1 =perete). Un robot se poate deplasa în acest labirint doar pe orizontală sau verticală. La fiecare mutare, acest robot poate trece într-o poziție vecină cu cea actuală pe orizontală sau verticală și în plus, poate sparge peretei. Timpul necesar deplasării într-o poziție liberă este 1, iar timpul necesar deplasării într-o poziție ocupată de perete este 2 (robotul fiind nevoit să spargă peretele). Se cere să se determine timpul minim necesar robotului pentru a se deplasa dintr-o poziție inițială dată, într-o finală.

Restricții

Datele se citesc dintr-un fișier text ROBOT.IN, având următoarea structură:

- pe prima linie sunt scrise numerele n și m ;
- pe următoarele două linii, coordonatele poziției inițiale și finale ale robotului;
- pe următoarele n linii se află o matrice având elemente în mulțimea $\{0, 1\}$, reprezentând configurația labirintului ([1]).

Soluție

Construim o matrice c de dimensiuni $n \times m$ în care $c_{i,j}$ este timpul minim necesar robotului pentru a ajunge din poziția inițială, până în poziția (i,j) .

Observăm că dacă poziția (i,j) este cameră, atunci

$$c_{i,j} = 1 + \min\{c_{i-1,j}, c_{i+1,j}, c_{i,j+1}, c_{i,j-1}\}$$

$$c_{i,j} = 2 + \min\{c_{i-1,j}, c_{i+1,j}, c_{i,j+1}, c_{i,j-1}\}$$

Modul de calcul al elementelor matricei c este următorul: determinăm toate pozițiile în care se poate ajunge în timp 1, apoi toate pozițiile în care se poate ajunge în timp 2, s.a.m.d., până în momentul în care am marcat și poziția finală a robotului.

Program DeplaseazaRobot;

```
const MaxN=6;
var a:array[1..MaxN] of string[MaxN];
    c:array[1..MaxN,1..MaxN] of -1..2*MaxN*MaxN;
    n,m,i,j,k,pil,pic,pfl,pfc:1..MaxN;
```

```
procedure citeste;
var f:Text;
begin
  Assign(f,'ROBOT.IN'); Reset(f);
  Readln(f,n,m); Readln(f,pil,pic); Readln(f,pfl,pfc);
  for i:=1 to n do
  begin
    for j:=1 to m do Read(f,a[i]); Readln(f)
  end;
  Close(f)
end;

function Inside(h:Integer):Boolean;
begin Inside:=(h>0) and (h<n+1) end;

procedure calcul;
begin
  for i:=1 to n do
    for j:=1 to m do c[i,j]:=-1;
  k:=1; c[pil,pic]:=0;
  while c[pfl,pfc]=-1 do
  begin
    for i:=1 to n do
      for j:=1 to m do
        if a[i,j]=0 then
          if c[i,j]=-1 then
            if ((Inside(i-1)) and (c[i-1,j]=k-1)) or
               ((Inside(i+1)) and (c[i+1,j]=k-1)) or
               ((Inside(j-1)) and (c[i,j-1]=k-1)) or
               ((Inside(j+1)) and (c[i,j+1]=k-1))
            then c[i,j]:=k
            else
            else
            if k>1
            then
              if c[i,j]=-1 then
                if ((Inside(i-1)) and (c[i-1,j]=k-2)) or
                   ((Inside(i+1)) and (c[i+1,j]=k-2)) or
                   ((Inside(j-1)) and (c[i,j-1]=k-2)) or
                   ((Inside(j+1)) and (c[i,j+1]=k-2))
                then c[i,j]:=k;
            k:=k+1
  end
end;
```

```

Begin
  citeste;
  calcul;
  Writeln('Timpul minim este:',c[pfl,pfc])
End.

```

Intrare

6 6
1 1
1 6
011110
000000
000000
000000
000000
000000

Ieșire**Timpul minim este:7****P12. Câini și pisici**

Într-un oraș se află inițial n pisici și m câini. Datorită faptului că întâlnirea dintre un grup de câini și un grup de pisici duce inevitabil la ciocniri săngeroase soldate cu victime, numărul animalelor din acel oraș este în continuă scădere. După un timp oarecare în oraș au mai rămas k pisici și L câini. Se cere să se determine numărul minim de întâlniri dintre câini și pisici care au avut loc, precum și numărul animalelor din fiecare grup în momentul fiecărei întâlniri. Se știe că întâlnirea unui grup de i pisici cu un grup de j câini va determina o încăierare din care vor supraviețui p pisici și c câini.

Restricții

Datele se citesc din fișierul text ANIMALE.IN, având următoarea structură:

- pe prima linie este scrisă perechea n, m ($n, m \leq 15$);
- pe a doua linie este scrisă perechea k, L ;
- pe următoarele linii sunt scrise quadruple (câte unul pe fiecare linie) de forma: $i \ j \ p \ c$ cu semnificația din enunț ([1]).

Soluție

Generăm toate perechile de numere care reprezintă numărul de animale care pot exista la un moment dat, adică toate perechile (i,j) în care i reprezintă numărul de pisici, iar j reprezintă numărul de câini. În total există $n \times m$ astfel de perechi. Fiecarei perechi (i,j) îi vom ataşa un număr de ordine care se calculează în felul următor: $(i - 1) * m + j$. Observăm că plecând de la un număr de ordine (z) putem reconstituîn mod unic perechea care are acel număr de ordine, și anume: $i = z \text{ div } m; j = z \text{ mod } m$;

Dacă $j = 0$, atunci $j = m$ și îl decrementăm pe i cu o unitate.

Aceste perechi vor fi nodurile unui graf orientat, ale căruia arce au următoarea semnificație:

- între perechea (i,j) și perechea (q,w) există arc, dacă și numai dacă în urma unei singure întâlniri (dintre cele posibile) numărul pisicilor scăde de la i la q și al câinilor de la j la w .

Odată construit acest graf, va trebui să determinăm un drum de lungime minimă de la perechea (n,m) la perechea (k,L) . Lungimea acestui drum va exprima numărul minim de întâlniri dintre câini și pisici. Determinarea acestui drum se va face prin programare dinamică, și anume:

- inițial vom marca nodul (n,m) , apoi la fiecare pas vom marca acele noduri (nemarcate) de la care pleacă arce spre nodurile deja marcate. Vom repeta această operație până când am reușit să marcăm nodul (k,L) sau nu mai putem marca nici un alt nod. Fiecare nod va fi marcat cu un număr care reprezintă lungimea minimă a drumului de la el la nodul (n,m) . Pentru a simplifica afișarea, vom reține pentru fiecare nod (în vectorul *tata*) nodul care a generat drumul minim.

```

Program CainiPisici;
const MaxN=225;
var a:array[1..MaxN,1..MaxN] of 0..1;
  patr:array[1..MaxN,1..4] of 1..MaxN;
  drum,tata:array[1..MaxN] of Integer;
  n,i,j,k,m,L,nrp,cainil,caini2,pisicil,pisici2,p:Byte;
  sw,blocaj:Boolean;
procedure citeste;
  var f:Text;
begin
  Assign(f,'ANIMALE.IN'); Reset(f); Read(f,n,m);
  Readln(f,k,L);
  nrp:=0;
  while not SeekEOF(f) do
  begin
    Inc(nrp);
    for i:=1 to 4 do
      if not SeekEOLN(f) then Read(f,patr[nrp,i])
    end;
  Close(f)
end;
procedure calcul;
begin
  for i:=1 to n*m-1 do
  begin
    a[i,i]:=0;
    for j:=i+1 to n*m do
    begin
      pisicil:=i div m+1; cainil:=i mod m;

```

```

if cainil=0
then
begin cainil:=m; Dec(pisicil) end;
pisici2:=j div m+1;
caini2:=j mod m;
if caini2=0
then
begin caini2:=m; Dec(pisici2) end;
if (pisicil<=pisici2) and (cainil<=caini2)
then
begin
sw:=false; p:=1;
while not sw and (p<=nrp) do
begin
if (pisici2>=patr[p,1]) and (caini2>=patr[p,2])
and (pisici2-patr[p,1]+patr[p,3]=pisicil)
and (caini2-patr[p,2]+patr[p,4]=cainil)
then sw:=true;
p:=p+1
end;
if sw
then begin a[j,i]:=1; a[i,j]:=0 end
else begin a[i,j]:=0; a[j,i]:=0 end
end
else
if (pisici2<=pisicil) and (caini2<cainil)
then
begin
sw:=false; p:=1;
while not sw and (p<=nrp) do
begin
if (pisicil>=patr[p,1]) and (cainil>=patr[p,2])
and (pisicil-patr[p,1]+patr[p,3]=pisici2)
and (cainil-patr[p,2]+patr[p,4]=caini2)
then sw:=true;
p:=p+1
end;
if sw then
begin a[i,j]:=1; a[j,i]:=0 end
else
begin a[i,j]:=0; a[j,i]:=0 end
end
else begin a[i,j]:=0; a[j,i]:=0 end
end
end;
end;

```

```

procedure Afiseaza(w:Byte);
var v,p1,c1,p2,c2:Byte;
begin
v:=tata[w];
if v<>0
then
begin
Afiseaza(v);
p1:=v div m+1; c1:=v mod m;
if c1=0 then begin c1:=m; Dec(p1) end;
p2:=w div m+1; c2:=w mod m;
if c2=0 then begin c2:=m; Dec(p2) end;
p:=0;
repeat
Inc(p);
until (patr[p,1]-patr[p,3]=p1-p2) and
(patr[p,2]-patr[p,4]=c1-c2) and
(patr[p,1]<=p1) and (patr[p,3]<=p2) and
(patr[p,2]<=c1) and (patr[p,4]<=c2);
Write('Se intalnesc ',patr[p,1]);
Writeln(' pisici cu ',patr[p,2],' caini.');
end
end;

Begin
citere;
calcul;
for i:=1 to n*m-1 do
begin drum[i]:=-1; tata[i]:=0 end;
drum[n*m]:=0; tata[n*m]:=0;
blocaj:=false;
while (drum[(L-1)*m+k]=-1) and not blocaj do
begin
blocaj:=true;
for i:=1 to n*m do
if drum[i]=-1
then
begin
j:=1;
while ((a[j,i]=0) or (drum[j]=-1)) and (j<=m*n) do
Inc(j);
if j<=n*m
then
begin drum[i]:=1+drum[j]; tata[i]:=j; blocaj:=false end
end
end;

```

```

if (n=k) and (m=L)
then Writeln('Nu a murit nici un animal!')
else
  if drum[(L-1)*m+k]=0
  then Writeln('Nu exista solutie!')
  else
    begin
      Write('Numarul minim de intalniri este:');
      Writeln(drum[(k-1)*m+L]);
      Afiseaza((k-1)*m+L)
    end
End.

```

Intrare

10 8
5 1
2 1 2 0
1 2 0 2
2 2 1 1

Ieșire

Numarul minim de intalniri este: 7
 Se intalnesc 2 pisici cu 2 caini.
 Se intalnesc 2 pisici cu 1 caini.
 Se intalnesc 2 pisici cu 1 caini.

P13. Formații

În vederea realizării unui spectacol de televiziune este nevoie de alcătuirea unui grup muzical din care să facă parte:

- v vocaliști ($0 \leq v \leq 1$); - c chitariști ($0 \leq c \leq 4$);
- p pianisti ($0 \leq p \leq 4$); - b bateriști ($0 \leq b \leq 4$).

Ca urmare a anunțului dat de realizatorul spectacolului (dar și datorită mizei mari) la televiziune se prezintă foarte multe formații, toate fiind la fel de valoroase. Se știe că există n tipuri de formații, din fiecare tip existând suficiente formații, iar fiecare tip i de formație are în componență v_i vocaliști, c_i chitariști, p_i pianisti și b_i bateriști. Se cere să se determine numărul minim de formații pe care le folosește realizatorul spectacolului pentru a-și alcătui grupul.

Restrictii

Datele se citesc dintr-un fișier text, având următoarea structură:

- pe prima linie este scrisă componența grupului pe care o dorește realizatorul spectacolului (dată sub forma quadruplului $v \ c \ p \ b$);
- pe următoarele linii este scrisă componența fiecărui tip de formație care se prezintă la televiziune ([1]).

Soluție

Pentru simplificare generăm toate quadruplurile (v, c, p, b) care sunt în număr de 250. Acestea vor reprezenta nodurile unui graf. Între două noduri i și j vom trasa un arc doar dacă există un tip de formație care adunată la trupa i deja formată va genera trupa j . Odată graful construit, va trebui să determinăm un drum de lungime minimă de la quadruplu $(0,0,0,0)$ la quadruplu (v, c, p, b) .

Program Formatii;

```

const MaxN=250;
type formație=array[1..4] of 0..5;
var c:array[0..MaxN,0..MaxN] of 0..1;
  drum,tata:array[0..MaxN] of shortint;
  form:array[1..10] of formație;
  sp:formație;
  i,j,k,nrf,v1,v2,c1,c2,b1,b2,p1,p2,p,r:Byte;
  gasit,blōcaj:Boolean;
procedure citește;
  var f:Text;
begin
  Assign(f,'FORMATIE.IN'); Reset(f);
  for i:=1 to 4 do
    if not (Seekerln(f)) then Read(f,sp[i]);
  Readln(f);
  nrf:=0;
  while not Seekerof(f) do
  begin
    Inc(nrf);
    for i:=1 to 4 do
      if not Seekerln(f) then Read(f,form[nrf,i]);
    end;
    Close(f)
  end;
procedure calcul;
begin
  for i:=0 to 249 do
    for j:=i+1 to 250 do
    begin
      v1:=i div 125; r:=i mod 125;
      c1:=r div 25; r:=r mod 5;
      p1:=r div 5; b1:=r mod 5;
      v2:=j div 125; r:=j mod 125;
      c2:=r div 25; r:=r mod 25;
      p2:=r div 5; b2:=r mod 5;
      if v1=c1+p1+b1 then c[j]:=1;
    end;
  end;
end;

```

```

if (v1<=v2) and (c1<=c2) and (p1<=p2) and (v1<=v2)
then
begin
  gasit:=false;
  p:=1;
  while not gasit and (p<=nrf) do
  begin
    if (v1+form[p,1]=v2) and (c1+form[p,2]=c2) and
       (p1+form[p,3]=p2) and (b1+form[p,4]=b2)
    then gasit:=true;
    p:=p+1
  end;
  if gasit then begin c[i,j]:=1; c[j,i]:=0 end
  else begin c[i,j]:=0; c[j,i]:=0 end
end
else
begin
  if (v2<=v1) and (c2<=c1) and (p2<=p1) and (v2<=v1)
  then
  begin
    gasit:=false;
    p:=1;
    while not gasit and (p<=nrf) do
    begin
      if (v2+form[p,1]=v1) and (c2+form[p,2]=c1)
         and (p2+form[p,3]=p1) and (b2+form[p,4]=b1)
      then gasit:=true;
      p:=p+1
    end;
    if gasit then begin c[i,j]:=0; c[j,i]:=1 end
    else begin c[i,j]:=0; c[j,i]:=0 end
  end
end
end;
begin
  citeste;
  calcul;
  for i:=0 to 250 do
  begin
    drum[i]:=-1; tata[i]:=0
  end;
  drum[125*sp[1]+25*sp[2]+5*sp[3]+sp[4]]:=0;
  blocaj:=false;
  tata[125*sp[1]+25*sp[2]+5*sp[3]+sp[4]]:=0;
  while (drum[0]=-1) and not blocaj do
  begin
    blocaj:=true;

```

```

for i:=0 to 250 do
  if drum[i]=-1
  then
  begin
    j:=0;
    while ((c[i,j]=0) or (drum[j]=-1)) and (j<=250) do
      Inc(j);
    if j<=250
    then
    begin
      drum[i]:=1+drum[j];
      tata[i]:=j;
      blocaj:=false
    end
  end;
  if drum[0]=-1
  then Writeln('Nu exista solutie!')
  else
  begin
    Write('Numarul minim de formari ');
    Writeln('folosite este:',drum[0])
  end
End.

```

Intrare Iesire

```

2 2 1 0      2
1 1 0 0
1 2 1 0
2 1 0 0

```

P14. Adaugă o unitate

Fie o matrice de dimensiuni $n \times m$ care poate conține doar elemente din multimea $\{0, 1\}$. Î fiecare linie și coloană reprezintă scrierea binară a unui număr natural (în cadrul unei linii, numărul se citește de la dreapta la stânga, adică cifra cea mai din dreapta este cea mai semnificativă, iar în cadrul unei coloane numărul se citește de jos în sus, adică cifra cea mai de jos este cea mai semnificativă). Singura operație care poate fi efectuată asupra acestei matrice este adăugarea (în baza 2) unei unități la unul din numerele binare de pe linii sau coloane.

Se cere să se determine numărul minim de unități care trebuie adăugate pentru a se trece într-o matrice care conține doar cifre de 1.

Restriții

Datele sunt citite dintr-un fișier, având următoarea structură:

- pe prima linie este scrisă perechea $n, m \leq 100$;
- pe următoarele n linii este scrisă matricea inițială ([1]).

Soluție

Construim un tablou c de dimensiuni $n \times m$ în care poziția curentă (i,j) reprezintă numărul minim de unități care trebuie adăugate pentru ca toate elementele matricei inițiale care se găsesc la intersecția liniilor $1, \dots, i$ cu coloanele $1, \dots, j$ să fie egale cu 1, adică submatricea, al cărei colț stânga sus este poziția $(1,1)$ a matricei inițiale, și al cărei colț dreapta jos este poziția (i,j) a matricei inițiale, să conțină doar elemente de unu.

Modul de calcul al elementelor matricei c este următorul:

- elementele de pe prima linie se calculează pe baza formulei:
 $c_{1,1} = 1 - a_{1,1}$, iar pentru $i \geq 2$ avem $c_{1,i} = 1 - a_{1,i} + c_{1,i-1}$, deoarece dacă $a_{1,i} = 0$, este suficient să adăugăm o unitate la numărul binar de pe coloana i .
- elementele de pe prima coloană se calculează asemănător cu cele de pe prima linie:
 $c_{i,1} = 1 - a_{i,1}$, iar pentru $i \geq 2$ avem $c_{i,1} = 1 - a_{i,1} + c_{i-1,1}$.
- celelalte elemente se calculează în felul următor:
 - 1) dacă $a_{ij} = 1$, atunci nu mai trebuie să adăugăm nici o unitate pentru a obține cifra 1 pe poziția (i,j) , deci $c_{ij} = c_{i,j-1} + c_{i-1,j} - c_{i-1,j-1}$.
 - 2) dacă $a_{ij} = 0$, atunci trebuie adăugate un număr minim de unități pentru a obține cifra 1 pe poziția (i,j) . Să notăm cu k maximul dintre i și j . Atunci:
 $c_{ij} = c_{i-1,j} + c_{i,j-1} - c_{i-1,j-1} + 1 + 2^{k-1} - 1$, deoarece trebuie să adăugăm o unitate (la o linie sau la o coloană binară) pentru a obține valoarea 1 la intersecția liniei i cu coloana j . Astfel linia sau coloana respectivă va arăta în genul 1000... (totul fiind citit de jos în sus și de la dreapta la stânga). Apoi trebuie să adăugăm valoarea $2^{k-1} - 1$ care în binar are $k - 1$ cifre de 1, unde k este minimul dintre numărul de linie și numărul de coloană la care am ajuns.

Program Adăuga_0_Unitate;

```

const MaxN=100;
var a:array[1..MaxN,1..MaxN] of 0..1;
    c:array[1..MaxN,1..MaxN] of Word;
    n,m,i,j:Byte;
procedure citeste;
  var f:Text;
begin
  Assign(f,'UNITATE.IN'); Reset(f); Readln(f,n,m);
  for i:=1 to n do
    begin
      for j:=1 to m do Read(f,a[i,j]); Readln(f)
    end;
  Close(f)
end;

```

```

procedure calcul;
begin
  if a[1,1]=0 then c[1,1]:=1
  else c[1,1]:=0;
  for i:=2 to n do
    if a[i,1]=0 then c[i,1]:=1+c[i-1,1]
    else c[i,1]:=c[i-1,1];
  for i:=2 to m do
    if a[1,i]=0 then c[1,i]:=1+c[1,i-1]
    else c[1,i]:=c[1,i-1];
  for i:=2 to n do
    for j:=2 to m do
      if a[i,j]=1 then c[i,j]:=c[i-1,j]+c[i,j-1]-c[i-1,j-1]
      else
        begin
          c[i,j]:=c[i-1,j]+c[i,j-1]-c[i-1,j-1]+1;
          if i>j then c[i,j]:=c[i,j]+1 shl (j-1)-1
          else c[i,j]:=c[i,j]+1 shl (i-1)-1
        end;
  end;
begin
  citeste;
  calcul;
  Writeln('Numarul minim de unitati este:',c[n,m])
end.

```

Intrare

```

3 3
1 0 0
0 0 1
1 0 1

```

Ieșire

```

Numarul minim de unitati este:7

```

P15. Distanță cursor

Un editor de texte poate efectua (asupra unui text) următoarele operații:

- deplasarea cursorului la stânga, sau dreapta (1);
- inserarea unei litere în poziția curentă a cursorului; în urma acestei operații cursorul este deplasat automat cu o poziție spre dreapta (2);
- ștergerea literei de deasupra cursorului; în urma acestei operații cursorul rămâne pe poziția lui inițială (3);
- modificarea literei de deasupra cursorului; în acest caz cursorul își păstrează poziția (4).

O persoană care utilizează acest editor, scrie la un moment dat un cuvânt, dar pe care ulterior dorește să îl modifice. Pentru aceasta, inițial poziționează cursorul sub prima literă a cuvântului, apoi începe modificarea acestuia.

Dându-se cuvântul inițial și cel final, se cere:

- numărul minim de operații necesare pentru a se trece din cuvântul inițial în cel final.
- numărul minim de operații de deplasare a cursorului (operație de tipul 1), necesare pentru a se trece din cuvântul inițial în cel final ([1]).

Soluție

Să identificăm și în cazul de față subproblemele comune din problema dată. Să notăm cuvântul inițial cu $A = a_1a_2...a_n$, iar cuvântul final cu $B = b_1b_2...b_m$. În aceste condiții o subproblemă poate fi definită ca fiind numărul minim de operații necesare pentru a trece din subcuvântul $a_1a_2...a_i$ în subcuvântul $b_1b_2...b_j$.

Rezolvarea problemei se face în felul următor:

- construim o matrice c cu n linii și m coloane în care c_{ij} este numărul minim de operații necesare pentru a trece din subcuvântul $a_1a_2...a_i$ în $b_1b_2...b_j$.

Modul de calcul al numerelor c_{ij} este următorul:

- calculăm prima linie a matricei determinând poziția în cuvântul final pe care se găsește prima literă din cuvântul inițial (să notăm cu k această poziție). Până la acea poziție avem $c_{1,1} = 1$, $c_{1,i} = 1 + c_{1,i-1}$, iar $c_{1,k} = c_{1,k-1}$ (în cazul $k = 1$, avem $c_{1,1} = 0$), apoi $c_{1,k+1} = c_{1,k} + 2$, iar $c_{1,i} = 1 + c_{1,i-1}$ pentru $i > k + 1$.
- se calculează prima coloană a matricei în mod asemănător cu prima linie;
- pentru a calcula și celelalte elemente trebuie să observăm mai întâi faptul că dacă o subsecvență din sirul A (care începe, de exemplu, cu poziția q și se termină cu poziția w), este identică cu o subsecvență (care începe cu poziția r și se termină cu poziția y) din cuvântul B , atunci pentru a calcula $c_{w+1,y+1}$, este necesar ca toată secvența care a fost comună celor două siruri să fie parcursă de cursor. De aceea mai construim o matrice p cu n linii și m coloane, în care p_{ij} reprezintă lungimea celei mai mari subsecvențe comune care se termină pe poziția i a lui A , respectiv pe poziția j a lui B .

Restul elementelor matricei c se calculează cu formula:

$$c_{ij} = 1 + \min\{c_{i-1,j} + p_{i-1,j}, c_{i,j-1} + p_{i-1,j-1}, c_{i-1,j-1} + p_{i-1,j-1}\}.$$

Numărul minim de poziții cu care se deplasează cursorul se obține scăzând din $c_{n,m}$ (calculat anterior) numărul minim de operații de tipul: *inserare caracter, stergere caracter, modificare caracter*, necesare pentru a trece din cuvântul A în cuvântul B .

Cele două proceduri care calculează distanța dintre cuvinte sunt: *Distanta* (calculează distanța dintre două cuvinte, fără a se pune problema deplasării cursorului), respectiv *DistantaCursor* (calculează distanța dintre cuvintele A și B dacă se introduce și cursorul).

Lăsăm în seama cititorului afișarea operațiilor necesare pentru a trece dintr-un cuvânt inițial într-un cuvânt final.

```
Program TransformaSiruri;
var c,p:array[1..100,1..100] of Word;
sir1,sir2:string;
i,j,poz,k:Byte; dis1,dis2:Word;
```

```
function Distanta(s1,s2:string):Word;
var nr:Byte; gasit:Boolean;
begin
  gasit:=false; nr:=0;
  for i:=1 to Length(s2) do
  begin
    if (s1[1]=s2[i]) and not gasit then gasit:=true
    else Inc(nr);
    c[1,i]:=nr
  end;
  gasit:=false; nr:=0;
  for i:=1 to Length(s1) do
  begin
    if (s2[1]=s1[i]) and not gasit then gasit:=true
    else Inc(nr);
    c[i,1]:=nr
  end;
  for i:=2 to Length(s1) do
    for j:=2 to Length(s2) do
      if s1[i]=s2[j]
      then c[i,j]:=c[i-1,j-1]
      else
        if c[i-1,j]<c[i,j-1]
        then
          if c[i-1,j]<c[i-1,j-1] then c[i,j]:=1+c[i-1,j]
          else c[i,j]:=1+c[i-1,j-1]
        else
          if c[i,j-1]<c[i-1,j-1] then c[i,j]:=1+c[i,j-1]
          else c[i,j]:=1+c[i-1,j-1];
  Distanta:=c[Length(s1),Length(s2)]
end;

function DistantaCursor(s1,s2:string):Word;
var nr:Word; gasit,ok:Boolean;
begin
  for i:=1 to Length(s1) do
    for j:=1 to Length(s2) do p[i,j]:=0;
  i:=1; nr:=0;
  while (i<=Length(s2)) and (s1[1]<>s2[i]) do
  begin Inc(nr); c[1,i]:=nr; Inc(i); end;
  if i<Length(s2)
  then
    begin
      c[1,i]:=nr; Inc(nr); Inc(i);
      while i<=Length(s2) do
        begin Inc(nr); c[1,i]:=nr; Inc(i) end
    end
  end;
```

```

else if i=Length(s2) then c[1,Length(s2)]:=nr;
i:=1; nr:=0;
while (i<=Length(s1)) and (s1[i]<>s2[i]) do
begin
  Inc(nr); c[i,1]:=nr; Inc(i)
end;
if i<Length(s1)
then
begin
  c[i,1]:=nr; Inc(nr); Inc(i);
  while i<Length(s1) do
  begin
    Inc(nr); c[i,1]:=nr; Inc(i)
  end
  end
else
  if i=Length(s1)
  then c[Length(s1),1]:=nr;
for i:=2 to Length(s1) do
for j:=2 to Length(s2) do
  if s1[i]=s2[j]
  then
    begin c[i,j]:=c[i-1,j-1]; p[i,j]:=1+p[i-1,j-1] end
  else
    if c[i-1,j]+p[i-1,j]<c[i,j-1]+p[i,j-1]
    then
      if c[i-1,j]+p[i-1,j]<c[i-1,j-1]+p[i-1,j-1]
      then c[i,j]:=2+c[i-1,j]+p[i-1,j]
      else c[i,j]:=2+c[i-1,j-1]+p[i-1,j-1]
    else
      if c[i,j-1]+p[i,j-1]<c[i-1,j-1]+p[i-1,j-1]
      then c[i,j]:=2+c[i,j-1]+p[i,j-1]
      else c[i,j]:=2+c[i-1,j-1]+p[i-1,j-1];
  DistanțaCursor:=c[Length(s1),Length(s2)]
end;
Begin
  Write('Primul cuvant:'); Readln(sir1);
  Write('Al doilea cuvant:'); Readln(sir2);
  dis1:=Distanța(sir1,sir2);
  dis2:=DistanțaCursor(sir1,sir2);
  Writeln('Punctul a)');
  Writeln(dis2);
  Writeln('Punctul b)');
  Write('Numarul minim de pozitii cu care ');
  Writeln('se deplasează cursorul:',dis2-dis1)
End.

```

Intrare

A=abc
B=adfA=abc
B=dbe

Ieșire

Punctul a)
4
Punctul b)
2Punctul a)
4
Punctul b)
2

Discuție

O variantă a problemei ar fi aceea în care cursorul nu s-ar afla inițial pe poziția 1 ci undeva, pe o poziție oarecare în sir. Este evident că în acest caz soluția optimă nu constă în aducerea cursorului pe prima poziție și apoi aplicarea algoritmului de mai sus. O altă generalizare a problemei constă în introducerea de noi operații ale editorului: de exemplu, cursorul să poată fi deplasat la sfârșitul sirului (tasta END), sau la începutul lui (tasta HOME) etc. Fiecare din aceste operații va avea atașat un cost fix.

P16. Depozite și centre de distribuire

Într-un oraș există două fabrici de pâine, n centre de distribuire a pâinii și o singură mașină de transport. Prima fabrică are un stoc de x_1 pâini, iar cea de-a două are un stoc de x_2 pâini. Cererile celor n centre de distribuire sunt date în sirul cerere, iar suma cererilor este egală cu cantitatea de pâine aflată la cele două fabrici. Cunoscând costul transportului unei pâini de la fabrica i la centrul j să se determine câte pâini sunt transportate de la fiecare fabrică la fiecare centru de distribuire, astfel încât să se satisfacă toate cererile cu un cost al transportului minim.

Restricții

Datele de intrare se citesc dintr-un fișier text având următoarea structură:

- pe prima linie se află numărul centrelor de distribuire: x_1 și x_2 ;
- pe următoarea linie se dau cererile celor n centre de distribuire;
- pe următoarele două linii se dau costurile transportului unei pâini de la prima fabrică, respectiv de la a două, la cele n centre ([3]).

Soluția 1

Pentru a putea include această problemă în cadrul metodei programării dinamice, trebuie să determinăm dacă ne aflăm sau nu, în cazul unui proces decizional pas cu pas, cu alte cuvinte trebuie să determinăm care sunt subproblemele și relațiile dintre ele. Într-adevăr, deciziile le putem lua pas cu pas, o asemenea decizie fiind satisfăcătoare unui centru de distribuire a pâinii. Astfel, la primul pas satisfacem primul centru de distribuire, la al doilea pas al doilea centru, ... etc.

Să notăm cu $a_k[i, j]$ costul minim necesar satisfacerii primelor k depozite cu i pâini din primul depozit și j pâini din al doilea depozit. În aceste condiții avem:

$$a_1[i, j] = cost_{1,1} \cdot i + cost_{2,1} \cdot j; (i + j = cerere_1). Apoi:$$

$$a_k[i, j] = \min_{\substack{h_1+h_2=cerere_k \\ h_1, h_2 \geq 0}} \{b_i \cdot cost_{1,k} + b_j \cdot cost_{2,k} + a_{k-1}[i-h_1, j-h_2]\} \text{ prin } h_1, \text{ respectiv } h_2$$

am notat numărul pâinilor duse de la prima fabrică, respectiv de la a doua, la centrul k .

Soluția se obține în $a_n[x_1, x_2]$.

În implementare nu este necesar să reținem tot sirul de matrice a_k , deoarece relația de recurență depinde doar de termenul predecesor, deci este suficient să lucrăm doar pe două matrice ale căror valori le vom tot schimba între ele.

Program DouaDepozite;

```

const MaxV=100;
type TMatrice=array[0..MaxV, 0..MaxV] of Word;
var a,b:TMatrice;
    cost:array[1..2, 1..100] of Byte;
    cerere:array[1..100] of Byte;
    f:Text;
    n,x1,x2,h1,h2:Word; i,j,k:Byte;

procedure Citire;
begin
  Assign(f,'DEPOZIT.IN'); Reset(f); Readln(f,n,x1,x2);
  for i:=1 to n do Read(f,cerere[i]);
  for i:=1 to 2 do
  begin
    for j:=1 to N do Read(f,cost[i,j]); Readln(f)
  end;
  Close(f);
end;

function min(a,b:Word):Word;
begin
  if a>b then min:=b else min:=a
end;

procedure calcul;
begin
  for i:=0 to x1 do
    for j:=0 to x2 do
      if i+j= cerere[1] then a[i,j]:=cost[1,1]*i+cost[2,1]*j;
  for k:=2 to N do
  begin
    for i:=0 to x1 do
      for j:=0 to x2 do
        begin
          b[i,j]:=65535;

```

```

        for h1:=0 to i do
          for h2:=0 to j do
            if h1+h2=cerere[k]
            then b[i,j]:=min(b[i,j],cost[1,k]*h1+
                               +cost[2,k]*h2+a[i-h1,j-h2])
        end;
        a:=b
      end;
    end;

Begin
  Citire;
  calcul;
  Writeln('Costul minim este:',a[x1,x2])
End.

```

Intrare Ieșire

```

3 5 6      29
3 4 4
5 2 3
5 3 4

```

Soluția 2

Este evident că soluția de mai sus poate fi mult îmbunătățită. În primul rând, deoarece $h_1 + h_2 = cerere_k$, este evident că nu mai avem nevoie și de h_2 , acesta fiind calculat cu formula $h_2 = cerere_k - h_1$.

De asemenea, nu mai avem nevoie nici de indicele j din sirul $a_k[i, j]$, deoarece stim că $i + j = \sum_{p=1}^{k-1} cerere_p$.

Notăm cu $a_k[i]$ costul minim al satisfacerii primelor k depozite cu i pâini de la prima fabrică. Avem: $a_1[i] = cost_{1,1} * i + cost_{2,1} * (cerere_1 - i)$. Apoi: $a_k[i] = \min_{\substack{h_1 \geq 0 \\ h_1+h_2=i}} \{a_{k-1}[i-h_1] + cost_{1,k} * h_1 + cost_{2,k} * (cerere_k - h_1)\}$.

Restricțiile la care sunt supuse h_1 și i sunt următoarele:

$cerere_k - h_1 \leq \sum_{p=1}^{k-1} cerere_p - i$ sau, cu alte cuvinte $h_2 \leq j$ și $i \geq \sum_{p=1}^k cerere_p - x_2$ deoarece există situații în care doar prima fabrică nu poate satisface singură cererile primelor k centre de distribuire. Soluția problemei se obține în $a_n[x_1]$.

Program DouaDepoziteRedus;

```

const MaxV=100;
type TVector=array[0..MaxV] of Integer;
var a,b:TVector;
    cost:array[1..2, 1..100] of Byte;
    cerere:array[1..100] of Byte;

```

```

f:Text;
n,x1,x2,h1,suma:Integer;
i,j,k:Byte;

procedure Citire;
begin
  Assign(f,'DEPOZIT.IN'); Reset(f); Readln(f,n,x1,x2);
  for i:=1 to n do Read(f,cerere[i]);
  for i:=1 to 2 do
    begin
      for j:=1 to n do Read(f,cost[i,j]);
      Readln(f)
    end;
  Close(f)
end;

function min(a,b:Integer):Integer;
begin
  if a>b then min:=b else min:=a
end;

function max(a,b:Integer):Integer;
begin
  if a<b then max:=b else max:=a
end;

procedure calcul;
begin
  for i:=0 to min(cerere[1],x1) do
    a[i]:=cost[1,1]*i+cost[2,1]*(cerere[1]-i);
  suma:=cerere[1];
  for k:=2 to N do
    begin
      for i:=max(0,suma+cerere[k]-x2) to min(suma+cerere[k],x1) do
        begin
          b[i]:=32000;
          for h1:=max(i-suma,0) to min(i,cerere[k]) do
            b[i]:=min(b[i],cost[1,k]*h1+cost[2,k]*
              (cerere[k]-h1)+a[i-h1])
        end;
      a:=b; suma:=suma+cerere[k]
    end
  end;
end;

Begin
  Citire;
  calcul;
  Writeln('Costul minim este:',a[x1])
End.

```

Intrare	Ieșire
3 5 6	29
3 4 4	
5 2 3	
5 3 4	

Discuție

Problema se rezolvă într-un mod asemănător și pentru cazul în care există trei fabrici și n centre de distribuire. Acest caz a fost propus la proba de baraj, a lotului olimpic de informatică, desfășurată la Cluj-Napoca, 1997.

P17. La cules de mere

Pe o creangă de măr, se află n mere, fiecare măr fiind caracterizat prin distanță de la pământ (număr întreg, dat în cm) la care se află și prin greutatea sa (număr întreg, dat în grame). Un culegător, dorește să culeagă o cantitate (greutate) cât mai mare de mere. După ce este cules un măr, întreaga creangă devine mai ușoară și se ridică în sus cu x cm. Culegătorul ajunge doar la merele aflate la o înălțime mai mică sau egală cu d . Se cere să se determine greutatea maximă de mere care poate fi culeasă și ordinea în care sunt culese merele care au această greutate.

Restricții

Datele de intrare se citesc dintr-un fișier text, având următoarea structură:

- pe prima linie se află înscrise trei numere întregi: n , x , d ;
- pe următoarele n linii se dau greutatea, respectiv înălțimea inițială față de sol, la care se află cele n mere.

(Baraj pentru lotul lărgit, Oradea 1998)

Soluție

Culegerea unui măr conduce la ridicarea întregii crengi cu o distanță x dată și din această cauză este posibil ca unele mere să se ridice la înălțimi de peste d (la care culegătorul nu poate ajunge). Apare astfel un "conflict de interes" între merele care trebuie culese și deci este esențială ordinea de culegere a lor.

Algoritmul *greedy* care culege la fiecare pas cel mai greu măr nu funcționează optim pentru următorul caz:

2 1 3
1 3
5 2

Algoritmul *greedy* care culege la fiecare pas cel mai de sus măr nu funcționează optim pentru următorul caz:

2 2 3
1 3
5 2

Soluția corectă se obține prin programare dinamică.

- Într-o primă etapă sortăm merele descrescător după înălțime.
- Notăm cu c_i greutatea maximă de mere pe care o putem obține culegând ne-apărat mărul având numărul de ordine k .

Aveam:

$$\begin{aligned} c_i &= \text{greutate}_i, \\ c_i &= \max_j \{c_j + \text{greutate}_j | \text{inaltime}_j + x * \text{NrMere}_j \leq d\}, \end{aligned}$$

unde cu greutate_i am notat greutatea mărului i , prin inaltime_i ; am notat înălțimea mărului i și prin NrMere_i am notat numărul merelor culese până la mărul i inclusiv.

Program CulegeMere;

```
const MaxN=10;
var greutate, inaltime, c, pred, NrMere:array[1..MaxN] of Word;
  x, d, n, i, j, max, nr:Word;
  f:Text;

procedure Citire;
begin
  Assign(f, 'MERE.IN'); Reset(f);
  Readln(f, n, x, d);
  for i:=1 to n do Readln(f, greutate[i], inaltime[i]);
  Close(f)
end;

procedure sort(l,r:Integer);
var i, j, x, y:Integer;
begin
  i:=l; j:=r;
  x:=inaltime[(l+r) div 2];
  repeat
    while inaltime[i]>x do i:=i+1;
    while x>inaltime[j] do j:=j-1;
    if i<=j
    then
      begin
        y:=inaltime[i];
        inaltime[i]:=inaltime[j]; inaltime[j]:=y;
        y:=greutate[i];
        greutate[i]:=greutate[j]; greutate[j]:=y;
        i:=i+1; j:=j-1
      end;
  until i>j;
  if l<j then sort(l,j);
  if i<r then sort(i,r)
end;
```

```
procedure OrdeneazaDescrescatorDupaInaltime;
begin
  sort(1,n)
end;

procedure Reconstituie(nr:Word);
begin
  if nr>0
  then
    begin
      Reconstituie(pred[nr]);
      Writeln('Culeg marul de greutate ', greutate[nr], ' si inaltime ', inaltime[nr]);
    end
  end;

procedure calcul;
begin
  max:=0;
  for i:=1 to n do
  begin
    c[i]:=greutate[i]; NrMere[i]:=1;
    for j:=1 to i-1 do
      if (inaltime[i]+NrMere[j]*x <=d )
      and (c[i] <= greutate[i]+c[j])
      then
        begin
          c[i]:=c[j]+greutate[i];
          NrMere[i]:=1+NrMere[j];
          pred[i]:=j;
        end;
    if c[i]>max then begin max:=c[i]; nr:=i end
  end;
end;

Begin
  Citire;
  OrdeneazaDescrescatorDupaInaltime;
  calcul;
  Writeln('Greutatea maxima de mere este:',max);
  Reconstituie(nr)
End.
```

Intrare

4 2 6
1 2
1 4
1 2
1 1

Ieșire

Greutatea maxima de mere este:3
Culeg marul de greutate 1 si inaltime 4

Discuție.

O rezolvare asemănătoare se aplică și în cazul în care greutatea este constantă, iar x este variabilă în funcție de măr, adică avem de a face cu x_i . În acest caz culegerea unei greutăți maxime de mere înseamnă culegerea unui număr maxim de mere.

Este interesant de cercetat cazul în care și greutatea mărului și distanța cu care se ridică creanga sunt variabile în funcție de mărul care este cules. Rezolvăm și acest caz tot prin programare dinamică. La soluția de mai sus aveam nevoie pe lângă greutate și de numărul merelor culese până la un moment dat. În acest fel puteam calcula distanța cu care s-a înălțat creanga ca fiind produsul dintre numărul merelor culese și distanța cu care se ridică creanga după culegerea unui măr. Acum reținem direct distanța cu care a urcat întreaga creangă. Fie $dist_i$ distanța cu care s-a ridicat creanga după culegerea unei multimi de mere de greutate maximă și care îl include și pe mărul i . Avenim:

$$dist_0 = 0,$$

$$dist_i = x_i + dist_j, \text{ unde } j \text{ este valoarea de proveniență a maximului numărului } c_i,$$

$$c_i = \max_j \{c_j + \text{greutate}_i | \text{înaltime}_i + x * dist_j \leq d\}$$

Dacă dorim ca problema să fie cât se poate de naturală, trebuie să avem în vedere faptul că nu toate crengile se ridică cu o aceeași înălțime atunci când este cules un măr. În acest caz, ca date de intrare se dau greutatea fiecărui măr (g_i) și înălțimea cu care se ridică mărul j atunci când este cules mărul i (x_{ij}).

P18. Arbori stricți

Fiecărui nod terminal al unui arbore binar strict i se asociază un cost întreg, pozitiv. Ponderea unui nod terminal este egală cu produsul dintre cost și nivelul pe care se află nodul (rădăcina se consideră pe nivelul 0). Ponderea unui arbore este definită ca fiind suma ponderilor nodurilor terminale. Dându-se costurile asociate fiecărui nod terminal (aceste noduri se consideră numerotate de la stânga la dreapta în arbore) se cere să se determine un arbore de pondere minimă.

Restricții

Datele de intrare se citesc dintr-un fișier text, având următoarea structură:

- pe prima linie se află numărul nodurilor terminale ale arborelui,
- pe a doua linie se dau costurile asociate nodurilor terminale.

Soluție

Arboarele fiind binar strict, este evident că nu putem atașa drumuri de lungime maximă nodurilor terminale de pondere minimă și drumuri de lungime minimă nodurilor terminale de pondere maximă. Dacă am fi putut efectua acest lucru, atunci problema ar fi admis o soluție *greedy*. Din cauza arborelui binar se poate întâmpla ca un drum să aibă ca frunză un nod de pondere mică, iar un alt drum vecin să aibă un nod terminal

cu pondere mare. În aceste condiții nu se poate face asignarea amintită, deci avem de a face cu un proces în care există un conflict de interes în distribuirea aceleiași resurse. De aceea apelăm la programarea dinamică: notăm cu $c_{i,j}$ costul satisfacerii a j noduri (de la stânga la dreapta) începând cu nodul i . Avem $c_{i,i} = a_i$ (adică ponderea nodului i),

$$\text{iar apoi } c_{i,j} = \min_k \{c_{i,k} + c_{k+i,j-k} + \sum_{g=i}^{i+k-1} a_g + \sum_{g=i+k}^{j-k} a_g\}.$$

Explicația acestei formule este următoarea: dorim să calculăm ponderea unui (sub)-arbore care are j noduri terminale, primul dintre ele fiind i . Avem calculate valorile pentru subarborei cu k noduri terminale, primul fiind i (subarborele A) și $j - k$ noduri terminale, primul fiind $k + i$ (subarborele B). Combinăm acești doi subarbore prin adăugarea unui nou nod care va servi drept rădăcină și care va avea ca subarbore stâng pe A și ca subarbore drept pe B . Calculăm costul noului (sub)arbore ca sumă a costurilor subarborelor A și B la care se adaugă costul rezultat prin deplasarea nodurilor lui A și B cu un nivel mai jos.

Program ArboriStricți;

```

const MaxN=100;
var c:array[1..MaxN,1..MaxN] of Longint;
    a:array[1..MaxN] of Word;
    n,i,j,k:Byte;
    f:Text;
    suma:Longint;

procedure Citire;
begin
  Assign(f,'ARB.IN'); Reset(f); Readln(f,n);
  for i:=1 to n do Read(f,a[i]);
  Close(f)
end;

function SumaCosturi(start,numar:Byte):Longint;
var g:Byte;
    s:Longint;
begin
  s:=0;
  for g:=start to start+numar-1 do Inc(s,a[g]);
  SumaCosturi:=s;
end;

procedure Calcul;
begin
  for j:=2 to n do
    for i:=1 to n-j+1 do
      begin
        c[i,j]:=MaxLongint;

```

```

for k:=1 to j-1 do
begin
  suma:=c[i,k]+c[i+k,j-k]+SumaCosturi(i,k)+  

         +SumaCosturi(i+k,j-k);
  if suma<c[i,j] then c[i,j]:=suma
end
end;
Begin
  Citire;
  Calcul;
  Writeln(c[1,n]);
End.

Intrare           Ieșire
4                  20
3 2 4 1

```

P19. Colorează cu patru culori

Se dau coordonatele a n puncte în plan. Între aceste puncte se trasează segmente astfel încât orice punct să fie legat de cel mult alte trei puncte.

Să se determine o colorare cu cel mult patru culori ale punctelor date astfel încât orice două puncte legate printr-un segment să fie colorate diferit.

Restricții

Datele se citesc dintr-un fișier text, având următoarea structură:

- pe prima linie este scris numărul de puncte $n \leq 200$;
- pe următoarele n linii sunt date coordonatele punctelor în ordinea crescătoare a coordonatei de pe axa Ox ; coordonatele sunt numere întregi nenegative, iar sistemul de referință este ales astfel încât coordonatele de pe axa Ox să fie distințe două câte două;
- pe următoarele n linii pentru fiecare punct sunt date punctele cu care acesta este legat ([9],[1]).

Soluție

Problema colorării nodurilor unui graf planar cu un număr minim de culori, astfel încât oricare două noduri adiacente să fie colorate diferit, necesită un algoritm având timp de execuție exponențial. De asemenea, se știe că numărul minim de culori necesare este patru.

Și totuși, această problemă, datorită unor particularități, nu necesită un algoritm exponențial. Motivele sunt:

- 1) gradul fiecărui nod este mai mic sau egal cu trei;

- 2) între puncte există o relație de ordine totală, determinată de aranjarea acestor puncte în planul xOy .

Colorarea punctelor se face în ordinea crescătoare a coordonatelor pe axa Ox .

Datorită faptului că fiecare punct este legat de cel mult alte trei puncte, sunt suficiente patru culori.

```

Program ColoreazaCu4Culori;
const MaxN=200;
type punct=record
  x,y:Word;
  color:1..4
end;
var a:array[1..MaxN] of punct;
  legate:array[1..MaxN] of set of 1..MaxN;
  n,i,j:Word;
  posibil:set of 1..4;
procedure citeste;
var f:Text;
  c:1..MaxN;
begin
  Assign(f,'PUNCTE.IN'); Reset(f); Readln(f,n);
  for i:=1 to n do Readln(f,a[i].x,a[i].y);
  for i:=1 to n do
  begin
    legate[i]:=[];
    while not Seekolein(f) do
    begin
      Read(f,c); legate[i]:=legate[i]+[c]
    end;
    Readln(f)
  end;
  Close(f)
end;
procedure calcul;
begin
  for i:=1 to n do
  begin
    posibil:=[1..4];
    for j:=1 to i-1 do
      if j in legate[i] then posibil:=posibil-[a[j].color];
    j:=1;
    while not (j in posibil) do Inc(j);
    a[i].color:=j
  end
end;

```

```

Begin
  citeste;
  calcul;
  for i:=1 to n do
  begin
    Write('Punctul de coordonate (',a[i].x,',',a[i].y);
    Writeln(') are culoarea ',a[i].color,'')
  end
End.

```

Intrare Ieșire

```

4 Punctul de coordonate (1,1) are culoarea 1.
1 1 Punctul de coordonate (2,2) are culoarea 1.
2 2 Punctul de coordonate (3,3) are culoarea 1.
3 3 Punctul de coordonate (4,4) are culoarea 2.
4 4
4
4
1 2 3

```

P20. Buget de călătorie

O persoană dorește să călătorească cu mașina proprie prin N orașe. În fiecare oraș există o stație de benzină care vinde carburant la pret₁ \$/litru. La plecare, persoana umple rezervorul mașinii, plătind x \$. Pe parcursul călătoriei, automobilistul oprește la o stație de benzină doar dacă are mai puțin de jumătate de rezervor plin, sau dacă benzina pe care o are nu îi ajunge până în următorul oraș. Se cunoaște numărul de litri de benzină consumată de mașină pe distanță de 1 km și distanțele între orașe. La fiecare oprire la o stație de benzină automobilistul cheltuiește încă 2\$ pe mâncare.

Să se determine o strategie pentru opririle automobilistului astfel încât acesta să ajungă în ultimul oraș cu un cost total minim.

Restricții

Datele de intrare se citesc dintr-un fișier text care are următoarea structură:

- pe prima linie este scrisă capacitatea rezervorului;
- pe a doua linie se precizează numărul de litri de benzină consumată pe 1 km;
- pe a treia linie se dă costul umplerii rezervorului în orașul de plecare;
- pe următoarele linii se dau perechi de numere reprezentând distanțele orașelor față de orașul de start și prețul unui litru de benzină în orașul respectiv.

(ACM)

Soluție

Construim un sir *cost*, în care pe poziția *k* reținem costul minim necesar pentru a se ajunge în orașul *k* plus costul aprovizionării cu benzină din orașul *k*.

Relațiile de recurență sunt:

*cost*₀ = costul aprovizionării din orașul de plecare, iar

*cost*_{*i*} = min_{*j*} {*cost*_{*j*} + *d*(*j*, *i*) * LitriPerKm * pret_{*i*} + 2}, *j* ≤ *i*, iar prin *d*(*j*, *i*) am notat

distanța dintre orașul *j* și orașul *i*.

Mai trebuie să ținem cont, bineînțeles, de restricțiile din enunț.

Program BugetDeCalatorie;

```

const MaxN=10;

var dist, pret:array[0..MaxN] of Word;
    cost:array[0..MaxN] of Longint;
    n, i, j, Capacitate, LitriPerKm, st:Longint;
    f:Text;

procedure Citire;
begin
  Assign(f, 'BENZINA.IN'); Reset(f);
  Readln(f, Capacitate);
  Readln(f, LitriPerKm);
  Readln(f, cost[0]);
  n:=0;
  while not SeekEOF(f) do
  begin Inc(n); Readln(f, dist[n], pret[n]) end;
  Close(f)
end;

function d(j, i:Byte):Longint;
begin d:=dist[i]-dist[j] end;

procedure Calcul;
begin
  for i:=1 to n-1 do
  begin
    cost[i]:=MaxLongint;
    for j:=0 to i-1 do
      if (Capacitate-d(j, i)*LitriPerKm>0) and
         ((Capacitate-d(j, i)*LitriPerKm<=Capacitate div 2) or
          (Capacitate-d(j, i+1)*LitriPerKm<0))
      then
        if cost[j]+d(j, i)*LitriPerKm*pret[i]+2<cost[i]
        then cost[i]:=cost[j]+d(j, i)*LitriPerKm*pret[i]+2
    end
  end;
end;

```

```

Begin
    Citire;
    Calcul;
    cost[n]:=MaxLongint;
    for j:=n-1 downto 1 do
        if (cost[j]<cost[n]) and (Capacitate-d(j,n)*LitriPerKm>=0)
        then cost[n]:=cost[j];
    if cost[n]>MaxLongint
    then Writeln('Costul minim este:',cost[n])
    else Writeln('Nu se poate.')
End.

```

Intrare

Ieșire

```

10          Costul minim este:57
2
5
3 2
6 3
8 4
13 0

```

P21. Înlocuirea utilajului

Una din problemele fundamentale ale societății noastre industriale este aceea a înlocuirii mașinilor vechi cu altele noi, a uneltele perimate cu aparate moderne. Întrucât, cu timpul, utilajul își pierde valoarea, fie prin uzură, fie în raport cu randamentul investițiilor și perfecționărilor mai recente, există un moment în care investițiile inițiale mari care se fac pentru reînnoirea utilajului, pierderea provocată de închiderea lucrului și costul recalificării personalului sunt, toate la un loc, compenate prin creșterea productivității și scăderea costului de producție.

Pentru a simplifica discuția, vom presupune că avem de a face cu o singură mașină, care oferă un anumit venit anual și necesită un anumit grad de îngrijire. În plus, în orice moment, mașina poate fi vândută și poate fi cumpărată una nouă. Considerăm că inițial cumpărarea unei mașini a necesitat *PretMasina \$*. Notăm că *venit_i* profitul obținut într-un an, de o mașină în vîrstă de *i* ani. Costul întreținerii (pe perioada unui an) a unei mașini vechi de *i* ani este *intretinere_i*. Înlocuirea unei mașini de *i* ani costă *inlocuire_i*. Se presupune că înlocuirea se poate face cel mult o dată într-un an. Să se determine strategia care trebuie urmată în fiecare an (înlocuirea mașinii sau întreținerea ei) astfel încât după o perioadă de *N* ani să avem un profit maxim.

Restricții

Datele de intrare se citesc dintr-un fișier text având următoarea structură:

- pe prima linie se află numărul *N*;
- pe a doua linie se precizează *PretMasina*;
- pe următoarele *n* linii avem câte un triplet:
venit_i, intretinere_i, inlocuire_i (*i*=0, ..., *n*+1) ([3]).

Soluție

Aveam din nou o problemă de programare dinamică unidimensională. Trebuie să stabilim anii în care se va face înlocuirea mașinii. În toți ceilalți ani mașina va fi doar întreținută.

Notăm cu *profit_i* profitul maxim care poate fi obținut după *i* ani, știind că în anul *i* s-a înlocuit mașina. Avem $profit_0 = venit_0 - PretMasina - intretinere_0$, adică profitul obținut în curs de un an de pe urma unei mașini de 0 ani este egal cu venitul adus de acea mașină din care se scade prețul de cumpărare al ei și costul de întreținere pe durata unui an.

Dacă în anul *i* se va face o înlocuire a unei mașini vechi de *j* ani, profitul va fi:

$$profit_i = profit_j + venit_0 - intretinere_0 - inlocuire_{i,j} + ProfitPerioada_{j,i-1}$$
 unde prin *ProfitPerioada(x,y)* am notat profitul care se obține de pe urma unei mașini schimbată în anul *x*, până în anul *y*.

```

Program InlocuireMasina;
const MaxN=100;
var venit,intretinere,inlocuire:array[0..MaxN] of Integer;
    profit:array[1..MaxN] of Integer;
    max,PretMasina:Integer;
    n,i,j:Byte;
    f:Text;

procedure Citire;
begin
    Assign(f,'INLOC.IN'); Reset(f);
    Readln(f,n);
    Readln(f,PretMasina);
    for i:=0 to n-1 do
        Readln(f,venit[i],intretinere[i],inlocuire[i]);
    Close(f);
end;

function ProfitPerioada(j,i:Byte):Integer;
var k:Byte;
    p:Integer;
begin
    p:=0;
    for k:=j+1 to i do
        p:=p+venit[k]-intretinere[k];
    ProfitPerioada:=p;
end;

```

```

procedure Calcul;
  var max:Integer;
begin
  profit[1]:=venit[0]-PretMasina-intretinere[0];
  for i:=2 to n do
    begin
      max:=-MaxInt;
      for j:=1 to i-1 do
        if max<profit[j]+venit[0]-intretinere[0]-
           inlocuire[i-j]+ProfitPerioada(j,i-1)
        then max:=profit[j]+venit[0]-intretinere[0]-
             inlocuire[i-j]+ProfitPerioada(j,i-1);
      profit[i]:=max;
    end;
end;

Begin
  Citire;
  Calcul;
  max:=profit[n];
  for j:=1 to n-1 do
    if max<profit[j]+ProfitPerioada(j,n)
    then max:=profit[j]+ProfitPerioada(j,n);
  Writeln('Profitul maxim dupa ',n,' ani este:',max)
End.

```

Intrare**Ieșire**

```

5          Profitul maxim dupa 5 ani este:2
10
5 1 0
4 2 2
4 3 2
3 3 4
2 4 4

```

Discuție

În enunțul de mai sus nu s-a ținut cont de creșterea productivității și a randamentului, respectiv de scăderea costurilor de producție. E normal ca o mașină produsă în anii '90 să fie mai ieftină și mai productivă decât una produsă în anii '80. Pentru a ține cont și de acest lucru ne mai trebuie date de intrare. Pentru ca acestea să nu fie prea mari, putem presupune că productivitatea unei mașini crește liniar, sau crește cu o constantă în fiecare nou an. De asemenea, prețul de producție al unei mașini ar putea scădea în fiecare an.

În practică mai intervine încă un parametru foarte important, și anume probabilitatea ca o mașină de vârstă k să nu se strice până în anul $k+1$. Această probabilitate este un număr real cuprins între 0 și 1. Este evident că dacă probabilitatea este 1, atunci utilajul

utilajul nu se va strica în decursul aceluia an, iar dacă probabilitatea este 0, atunci utilajul se va strica sigur în acel an. Pentru simplificare, presupunem că un utilaj odată stricat nu mai poate fi reparat, deci el va trebui înlocuit. Mai trebuie precizat că, totuși, costul de întreținere este valabil și la această problemă, deoarece această operație nu constă în reparații capitale, ci doar revizii, alimentări, înlocuire de mici piese uzate etc.

Să se determine o strategie astfel încât după N ani, profitul mediu probabil să fie maxim.

Soluție

Relațiile de recurență anterioare sunt aproape identice și, în acest caz, mai trebuie adăugată probabilitatea de a obține venitul pe anul respectiv.

```

Program InlocuireUtilaj;
  const MaxN=10;
  var inlocuire,intretinere,venit:array[0..MaxN] of Integer;
    profit:array[1..MaxN] of Real;
    probabil:array[0..MaxN] of Real;
    max,PretUtilaj:Real;
    f:Text;
    n,i,j:Byte;

  procedure Citire;
  begin
    Assign(f,'INLOC2.IN'); Reset(f);
    Readln(f,n);
    Readln(f,PretUtilaj);
    for i:=0 to n-1 do
      Readln(f,venit[i],intretinere[i],inlocuire[i],probabil[i]);
    Close(f);
  end;

  function ProfitPerioada(j,i:Byte):Real;
    var k:Byte;
    p:Real;
  begin
    p:=0;
    for k:=j+1 to i do
      p:=p+probabil[k-j]*venit[k-j]-intretinere[k-j];
    ProfitPerioada:=p;
  end;

  procedure Calcul;
    var max:Real;
  begin
    profit[1]:=probabil[0]*venit[0]-PretUtilaj-intretinere[0];
    for i:=2 to n do
      begin
        max:=-MaxInt;
        for j:=1 to i-1 do
          if max<profit[j]+venit[0]-intretinere[0]-
             inlocuire[i-j]+ProfitPerioada(j,i-1)
          then max:=profit[j]+venit[0]-intretinere[0]-
               inlocuire[i-j]+ProfitPerioada(j,i-1);
        profit[i]:=max;
      end;
  end;

```

```

for j:=1 to i-1 do
  if max<profit[j]+probabil[0]*venit[0]-intretinere[0]-
    inlocuire[i-j]+ProfitPerioada(j,i-1)
  then max:=profit[j]+probabil[0]*venit[0]-intretinere[0]-
    inlocuire[i-j]+ProfitPerioada(j,i-1);
  profit[i]:=max
end;
begin
  Citire;
  Calcul;
  max:=profit[n];
  for j:=1 to n-1 do
    if max<profit[j]+ProfitPerioada(j,n)
    then max:=profit[j]+ProfitPerioada(j,n);
  Writeln('Profitul maxim dupa ',n,' ani este:',max)
end.

```

Intrare	Ieșire
5	0.4
10	
5 1 0 0.9	
4 2 2 1	
4 3 2 0.8	
3 3 4 0.6	
2 4 4 0.5	

P22. Intercalare siruri

Se dau două siruri de numere întregi. Se cere să se afișeze subșirul crescător de lungime maximă al sirului obținut prin intercalarea, în orice mod, a sirurilor date.

Restricții

Datele de intrare se citesc dintr-un fișier text având următoarea structură:

- pe prima linie se află lungimile celor două siruri (n, m);
- pe a doua linie se află numerele primului sir, despărțite prin spațiu;
- pe a treia linie se află numerele celui de al doilea sir.

Soluție

Determinăm subșirurile crescătoare ale sirurilor inițiale și interclasăm rezultatele.

Procedura SubSirCrescator returnează subșirul crescător de lungime maximă al sirului dat ca prim parametru.

Procedura Interclasare realizează interclasarea a două siruri, date ca parametri. Nu am implementat-o aici, deoarece ea face subiectul mai multor probleme din capitolele următoare.

```

program SirCrescator;
const MaxN=100;
type TSir=array[1..MaxN] of Integer;
var a,b,c,s1,s2,w,pw:TSir;
  n,m,l1,l2,lc,i,j:Byte;
  f:Text;
procedure Citire;
begin
  Assign(f,'SIR.IN'); Reset(f); Readln(f,n,m);
  for i:=1 to n do Read(f,a[i]); Readln(f);
  for j:=1 to m do Read(f,b[j]);
  Close(f);
end;
procedure SubSirCrescator(x:TSir; lung:Byte;
                           var sx:TSir; var slung:Byte);
var pmax:Byte;
procedure Reconst(p:Byte);
begin
  if p>0
  then
    begin Reconst(pw[p]); Inc(slung); sx[slung]:=x[p] end
  end;
begin
  for i:=1 to lung do w[i]:=0;
  w[1]:=1; pmax:=1;
  for i:=2 to lung do
  begin
    w[i]:=1;
    for j:=1 to i-1 do
      if x[i]>=x[j]
      then
        if w[i]<w[j]+1 then begin w[i]:=w[j]+1; pw[i]:=j end;
        if w[i]>w[pmax] then pmax:=i
      end;
    slung:=0;
    Reconst(pmax)
  end;
end;
procedure Interclasare(s1:TSir; l1:Byte; s2:TSir; L2:Byte;
                       var c:TSir; var lc:Byte);
begin
  { aici se introduce cod pentru interclasarea a două siruri }
  { s1 de lungime l1 și s2 de lungime L2 }
  { rezultatul va fi depus în sirul c de lungime lc }
end;

```

```

Begin
    Citire;
    SubSirCrescator(a,n,s1,l1);
    SubSirCrescator(b,m,s2,l2);
    Interclasare(s1,l1,s2,l2,c,lc)
End.

```

Intrare	Ieșire
3 4	1 2 2 4 5
3 1 4	
2 2 1 5	

P23. Ploaie în luna lui Marte

Ploua infernal și... Gigel dorea să ajungă acasă. Se afla la școală și își făcea planuri privind modul în care trebuie să străbată drumul până acasă, udându-se cât mai puțin. Știa că trebuie să ajungă acasă în T secunde, și că dacă o ia la fugă, poate ajunge în M secunde ($M \leq T$). Pe drumul spre casă se află $N + 1$ magazine unde se poate adăposti de ploaie. Adăpostul 0 este școală, iar adăpostul N este chiar casa lui. De asemenea, el cunoaște (de la Serviciul Meteo) cantitățile de apă care vor cădea asupra orașului în fiecare secundă. Este evident că Gigel dorește să își planifice adăpostiri sub acoperișurile magazinelor, astfel încât cantitatea de apă care îi va uda hainele să fie cât mai mică. Scrieți un program care îl ajută pe Gigel să își planifice drumul până acasă.

Restricții

Datele de intrare se citesc dintr-un fișier text având următoarea structură:

- pe prima linie se află numerele N și T ;
- pe a doua linie se dau distanțele la care se află magazinele față de școală (care este punctul de plecare); ultima valoare din acest șir este distanța între școală și casă; distanțele se dau în metri, iar viteza cu care băiatul parcurge aceste distanțe este aceeași: 1m/s.
- pe a treia linie se dau cantitățile de apă care cad în fiecare secundă începând cu momentul 0 și până la momentul T .

Soluție

O subproblemă va consta din determinarea cantității minime de ploaie pe care o acumulează hainele băiatului dacă la momentul j intră sub adăpostul i . Este evident că momentul j este cel puțin egal cu distanța de la școală la adăpostul i și cel mult egal cu $T - dist_n + dist_i$ (deoarece în timpul care a mai rămas trebuie să ajungă acasă). Să notăm cu $cost_{i,j}$ aceste numere. Este evident că soluția se află undeva pe linia lui N , adică în $c_{N,k}$, unde k este mai mare decât $dist_n$ (deoarece Gigel parcurge distanțele cu viteza de 1m/s). Pentru determinare relațiilor de recurență trebuie să ne "legăm" doar de magazinul în care Gigel s-a adăpostit anterior. Acest magazin va avea numărul de ordine

k (între 0 și $i-1$). Între magazinul k și magazinul i Gigel nu s-a mai adăpostit niciodată. În magazinul k el a intrat într-un moment $q \leq j$.

Modul de calcul al numerelor $cost_{i,j}$ este:

$$cost_{i,j} = \min_{k,q} \left\{ c_{k,q} + \sum_{c=j-dist_i+dist_k}^{dist_j-dist_k} intensitate_c \right\}$$

Suma din formula de mai sus reprezintă cantitatea de ploaie care cade asupra lui Gigel între magazinele k și i .

Reconstituirea și afișarea soluției se face recursiv cu procedura Reconstituie.

O altă modalitate de a defini o subproblemă este următoarea: $cost_{i,j}$ este cantitatea minimă de ploaie care cade asupra băiatului, știind că în momentul j a plecat de sub adăpostul i . Relațiile de recurență pentru calculul acestor numere sunt asemănătoare cu cele precedente.

```

Program PloaieInLunaLuiMarte;
const MaxN=10; MaxT=10;
type TProvenienta=record { se retine pozitia de provenienta }
                           moment,loc:Byte           { a minimului }
                           end;
var dist:array[0..MaxN] of Word;
    intensitate:array[0..MaxT] of Byte;
    cost:array[0..MaxN,0..MaxT] of Word;
    d:array[0..MaxT,0..MaxN] of TProvenienta;
    T,n,i,j,prov:Byte;
    min:Word;
    f:Text;
procedure Citire;
begin
    Assign(f,'PLOAIE.IN'); Reset(f);
    Readln(f,n,t);
    for i:=1 to n do Read(f,dist[i]);
    Readln(f);
    for i:=0 to T do Read(f,intensitate[i]);
    Close(f)
end;
function PloaieEfectiva(start,lung:Word):Word;
var pl,c:Word;
begin
    pl:=0;
    for c:=start to start+lung-1 do
        Inc(pl,intensitate[c]);
    PloaieEfectiva:=pl
end;

```

```

procedure Calcul;
var q,k:Byte;
begin
  for i:=1 to N do
    for j:=dist[i] to T-dist[n]+dist[i] do
    begin
      cost[i,j]:=MaxInt;
      for k:=0 to i-1 do
        for q:=dist[k] to j-dist[i]+dist[k] do
          if cost[k,q]+
            PloaieEfectiva(j-dist[i]+dist[k],dist[i]-dist[k])
              < cost[i,j]
          then
          begin
            cost[i,j]:=cost[k,q]+
            PloaieEfectiva(j-dist[i]+dist[k],dist[i]-dist[k]);
            d[i,j].moment:=q; d[i,j].loc:=k
          end
        end
      end;
    end;

procedure Reconstituie(loc,moment:Byte);
begin
  if moment >0
  then
  begin
    Reconstituie(d[loc,moment].loc,d[loc,moment].moment);
    Writeln('A plecat de sub adapostul ',d[loc,moment].loc,
    ' la momentul ',moment-dist[loc]-dist[d[loc,moment].loc]);
    Writeln('A intrat sub adaptostul ',loc,' la momentul ',moment)
  end
end;

Begin
  Citire;
  Calcul;
  min:=MaxInt; prov:=0;
  for i:=1 to T do
    if (min>cost[N,i]) and (cost[N,i]>0)
    then begin prov:=i; min:=cost[N,i] end;
  Writeln('Cantitatea minima de ploaie acumulata:',min);
  Reconstituie(N,prov)
End.

```

Intrare

3 8
2 5 7
7 1 1 1 1 1 1 1 1

Ieșire

Cantitatea minima de ploaie acumulata: 7
A plecat de sub adaptostul 0 la momentul 1
A intrat sub adaptostul 3 la momentul 7

Discuție

În cazul în care există mai multe soluții, determinați soluția cu număr minim de adăpostiri.

Gigel, în drum spre casă trebuie să se opreasă la anumite magazine date pentru a face cumpărături. Se cere o soluție a problemei care să satisfacă și aceste cerințe.

Probleme propuse**1. Al k-lea cel mai scurt drum**

Se dă un graf neorientat; fiecare muchie având atașat un cost. Să se determine al k-lea cel mai scurt drum ([3]).

2. Problema furnicii

Se consideră un caroaj având dimensiunea $n \times n$. O furnică se află inițial în căsuță de coordonate (0,0); la fiecare pas ea se poate deplasa doar într-o căsuță din dreapta cu o probabilitate p, sau într-o căsuță de deasupra, cu o probabilitate q ($p + q = 1$). Să se calculeze probabilitatea ca furnica să ajungă în căsuța (i, j).

3. Problema directorului de restaurant

Directorul unui restaurant își planifică activitatea pe următoarele n zile. Pentru fiecare zi j , ($j=1,n$), cunoaște necesarul de șervețe (r_j) de masă curate. În localitate există două spălătorii. La una termenul de prestare a serviciului este de p zile și costul spălării unei șervețe este de b lei; la cealaltă termenul este de q zile și costul de c lei. Pornind fără nici un șerveț utilizable (acestea sunt murdare sau sunt la spălătorie) directorul trebuie să cumpere șervețe noi cu a lei bucata. Cum trebuie să-și planifice achiziționarea șervețelor noi și spălarea șervețelor existente, astfel încât costul total pe perioada de n zile să fie minim?

Ipoteză simplificatoare: încercați într-o primă etapă să rezolvați problema știind că au fost cumpărate x șervețele ([3]).

4. Stoc de produse

Un magazin particular se ocupă cu vânzarea unui singur produs (de exemplu, biciclete). Prețul de achiziționare, precum și prețul de vânzare al bicicletelor variază în fiecare lună. Se mai cunoaște cererea (numărul de biciclete) pe fiecare lună. Magazinul are un depozit în care încap cel mult m biciclete. Pentru stocarea unei biciclete în depozit se percepe o taxă de x lei în fiecare lună. Să se determine o strategie de cumpărare și stocare a bicicletelor astfel încât profitul obținut după N luni să fie maxim.

Datele de intrare se citesc dintr-un fișier text având următoarea structură:

- pe prima linie se află numerele N, M și x;
- pe următoarele N linii sunt scrise triplete, reprezentând prețul de achiziție, prețul de vânzare și cererea de biciclete corespunzătoare celor N luni ([3]).

5. Prelucrare de materiale

Considerăm N articole diferite care trebuie să treacă prin două faze de prelucrare (două mașini), una după alta. Pe fiecare mașină, într-un moment poate fi prelucrat un singur articol și presupunem că odată ce prelucrarea a început, ordinea articolelor ce se prelucrează nu se mai poate schimba. Fiind dat timpul necesar pentru trecerea fiecărui articol prin fiecare fază, în ce ordine trebuie să fie introduse articolele în prima fază, astfel încât timpul necesar pentru prelucrarea tuturor articolelor să fie minim ([3])?

6. Problema taximetristului

Să presupunem că un taximetrist lucrează într-un sector, format din trei orașe. Dacă el se află în orașul 1 atunci are trei alternative:

- Poate face o cursă cu speranță că va găsi pe drum un pasager.
- Poate merge la cea mai apropiată stație de taxi și să aștepte.
- Poate staționa undeva, așteptând o chemare prin radio.

În orașul 3 are aceleași alternative, însă în orașul 2 a treia alternativă lipsește, deoarece în acest oraș nu există serviciu radio pentru chemarea taximetrelor. Pentru fiecare oraș dat și fiecare alternativă dată în cadrul acestui oraș există câte o probabilitate ca următoarea ursă să fie angajată pentru oricare dintre orașele 1, 2, 3, cu un venit corespunzător. Probabilitățile și veniturile depind de alternativă deoarece fiecărei alternative îi corespunde o populație de clienți diferită.

Să se determine o strategie, astfel încât, după ce a servit n clienți, taximetristul să obțină un profit maxim ([3]).

7. Tastatură optimă

Se consideră o tastatură specială: caracterele sunt amplasate doar pe o singură linie. Ordinea caracterelor se poate schimba în funcție de textul introdus, însă această schimbare nu se poate face în timpul tastării, ci doar înainte de a începe introducerea unui text.

Dându-se un text, să se stabilească ordinea literelor de pe tastatură, astfel încât deplasarea degetelor pe ea să fie minimă.

8. Drum în arbore

Se dă un arbore cu n noduri. Fiecare nod îi este atașată o valoare pozitivă. Să se determine un drum de cost minim care pleacă de la rădăcină la o frunză oarecare.

9. Medicamente pe rețetă

O persoană trebuie să procure n medicamente, având la dispoziție trei tipuri de rețete: gratuite, rețete doar pe jumătate compensate și rețete necompensate. Dacă persoana dorește să cumpere un anumit medicament (sau mai multe) de pe o rețetă, va trebui să cumpere toate medicamentele care sunt prescrise pe acea rețetă. Se cere suma minimă pe care trebuie să o plătească persoana pentru a cumpăra cele n medicamente, fără a cumpăra nici un alt medicament în afara celor de care are nevoie.

Datele de intrare se citesc dintr-un fișier text având următoarea structură:

- pe prima linie este scris numărul de rețete (m);

- pe următoarele n linii se află medicamentele înscrise pe rețeta respectivă și tipul acestora (gratuită, compensată sau necompensată);
- pe ultima linie se dau cele n medicamente pe care le dorește persoana în cauză.

10. Treceri de pietoni

Într-un oraș există un sistem de străzi cu *structură arborescentă*. Astfel există o singură stradă în *centrul orașului* de la care pleacă străzile, ramificându-se până la periferia orașului.

Să se determine numărul minim de treceri de pietoni care trebuie amplasate astfel încât pe cel puțin una din oricare trei (apoi generalizat la k) străzi consecutive (adiacente) să existe o trecere de pietoni, și în plus, pe strada din centru să nu fie amplasată nici una.

11. Problema monedelor în n dimensiuni

Se dă o mulțime A având m elemente de forma $x = (x_1, x_2, \dots, x_n)$ și un număr k având aceeași structură. Să se scrie k sub formă unei sume de numeră din A .

12. Problema culegătorului de cartofi

Fie un teren pe care se cultivă cartofi. Se dau coordonatele în planul xOy a cuburilor de cartofi. În fiecare cub se află o anumită cantitate dată de cartofi. Un hoț planifică să culeagă cât mai mulți cartofi, parcurgând terenul semănat de jos în sus. Fiecare cub de cartofi cules are coordonata pe axa Oy strict mai mare decât anteriorul cub cules. Determinați ordinea în care trebuie culese cuburile astfel încât cantitatea de cartofi culeasă să fie maximă. Dacă există mai multe soluții se cere cea pentru care distanța parcursă de culegător este minimă.

- Considerați cazul în care hoțul mai merge încă o dată la furat pe același teren. Să în această situație dorește cantitate maximă de cartofi, iar distanța totală parcursă cu cele două ocazii să fie minimă.
- Se cere o soluție în care distanța dintre două cuburi culese succesiv să fie mai mică decât o valoare dată x .
- Determinați o soluție în care distanța totală parcursă de hoț este mai mică decât o valoare dată D .

13. Expresii regulate

Expresiile regulate sunt siruri de caractere de forma $ba((na+bo)^*)^n$. Ele sunt interpretate ca fiind tipare sau *modele* care generează siruri de lungime mai mare, folosind următoarele trei operații:

Concatenare: dacă $e1$ este o expresie regulară pentru sirul $s1$ iar $e2$ este una pentru sirul $s2$, atunci $e1e2$ este expresie regulară pentru $s1s2$.

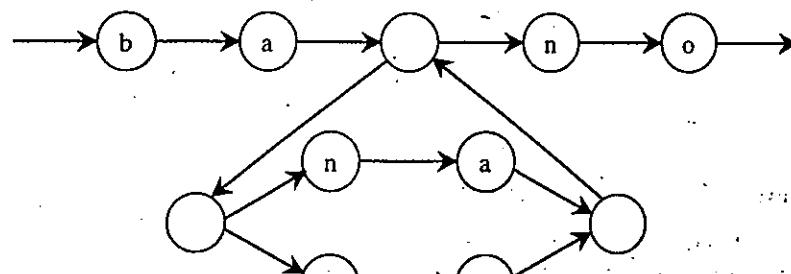
Iterații: dacă e este o expresie regulară pentru sirul s , atunci e^* este o potrivire pentru orice număr de copii ale lui s , inclusiv 0.

Alternăție: $e1 + e2$ este o potrivire pentru $s1$ sau pentru $s2$.

Dându-se o expresie regulară și un sir, să se verifice dacă expresia regulară este un model pentru sirul dat.

Indicație:

Construim un graf al expresiei regulate în care nodurile conțin caracterele expresiei, iar muchiile codifică cele trei operații. De exemplu, pentru expresia regulară $ba((na+bo)^*)no$ graful este:



Determinăm apoi un drum care coincide cu sirul dat.

14. Condorra

Republica Condorra are o hidrocentrală care, atunci când funcționează, produce x kWh putere. Întreținerea generatorului în stare de funcționare costă y dolari pe oră. De asemenea Condorra poate importa energie electrică cu z dolari/kWh, sau poate vinde cu s dolari/kWh. Pornirea generatorului costă g dolari, dacă el a fost oprit. Pentru următoarele N ore se dă cererile de curent din țară. Determinați cum și cât trebuie să funcționeze centrala, astfel încât să satisfacă cererile, iar profitul obținut să fie maxim (sau pierderile să fie minime).

15. Aliniatul secvențelor

Se dau două siruri de caractere A și B.

Se cere să se introducă în cele două siruri spații (caractere vide), astfel încât mulțimea $M = \{k \mid A_k = B_k\}$ să fie de cardinal maxim.

Se cere să se introducă în cele două siruri spații (caractere vide), astfel încât cardinalul mulțimii $M = \{k \mid A_k = B_k\}$ din care se scad numărul de spații introduse, să fie maxim.

16. Plăti incerte

Un lanț format din n magazine primește o cantitate de L litri de lapte. Se știe că cererea și probabilitatea de cerere este a_i , pentru fiecare magazin i și j litri de lapte. Se cere să se distribuie cei L litri de lapte la cele N magazine astfel încât profitul obținut să fie maxim, știind că un litru de lapte se vinde cu z lei, iar laptele nevândut se penalizează cu x lei/litru.

II. METODA BACKTRACKING

P1. Cuvinte

Se consideră un sir de n litere mici distințe.

Să se genereze toate cuvintele care au următoarele proprietăți:

- conțin exact m litere din sirul celor n litere date;
- literele din care sunt formate, apar în ordine lexicografică (crescătoare).

Datele de intrare se citesc de la tastatură. Cuvintele generate vor fi scrise în ordine lexicografică în fișierul de ieșire OUTPUT.TXT, un cuvânt pe fiecare linie.

Exemplu:

Intrare

$n=4$

$m=2$

Sirul de litere:

a

b

c

d

Ieșire

ab

ac

ad

bc

bd

cd

Soluție

Deoarece se cere determinarea tuturor soluțiilor, vom utiliza *metoda backtracking*. Vom prezenta o implementare nerecursivă, pentru o mai ușoară înțelegere a ei. Literele se memorează în vectorul a , având componente de tip caracter. Stiva este reprezentată prin vectorul st , având componente de tip întreg, deoarece va reține indicii literelor din vectorul a . Domeniul în care iau valori elementele vectorului st este intervalul $[0, n]$. Un nou nivel pe stivă se inițializează prin depunerea valoții 0. Un nou element pe un nivel pe stivă se determină prin incrementarea valorii elementului curent. Un element depus pe stivă se validează prin verificarea condiției de ordine lexicografică a literelor din cadrul cuvântului. Am obținut o soluție în momentul în care nivelul stivei este egal cu numărul m impus pentru dimensiunea cuvintelor.

```

Program cuvinte;
type stiva=array[1..100] of Integer;
var st:stiva;
    a:array[1..1000] of Char;
    i,j,n,k,m:Integer;
    as,ev:Boolean;
    g:Text;
procedure succ(k:Integer; var st:stiva; var as:Boolean);
begin
    if st[k]<n then begin Inc(st[k]); as:=true end
    else as:=false
end;
procedure valid(k:Integer; st:stiva; var ev:Boolean);
begin
    ev:=true;
    if k>1
    then
        if a[st[k-1]]>=a[st[k]] then ev:=false
end;
Begin
    Write('n='); Readln(n);
    Write('m='); Readln(m);
    Writeln('Sirul de caractere:');
    for i:=1 to n do Readln(a[i]);
    Assign(g,'OUTPUT.TXT'); ReWrite(g);
    k:=1; st[k]:=0;
    while k>0 do
    begin
        repeat
            succ(k,st,as);
            if as then valid(k,st,ev)
        until not as or (as and ev);
        if as
        then
            if k=m
            then
                begin
                    for i:=1 to m do Write(g,a[st[i]]);
                    Writeln(g)
                end
            else
                begin Inc(k); st[k]:=0 end
        else Dec(k)
    end;
    Close(g)
End.

```

P2. Problema celor 8 regine

Să se creeze un fișier cu tip, numit REGINE care va cuprinde soluțiile problemei celor 8 regine. Se cere determinarea tuturor modalităților de așezare a 8 regine pe tabla de șah, astfel încât ele să nu se atace între ele. Din fișierul standard de intrare se vor citi numerele de ordine ale soluțiilor care trebuie afișate. Se va afișa fiecare soluție cerută, accesând fișierul REGINE în mod direct.

Soluție

Deoarece se cere determinarea tuturor soluțiilor, vom utiliza *metoda backtracking*.

Tabloul unidimensional *r* se utilizează pentru a păstra în elementele sale (*r_i*) coloana în care se placează a *i*-a regină.

Condițiile de continuare constituie verifică dacă regina curentă atacă sau nu reginele așezate deja. Se știe că două regine se atacă dacă se află pe aceeași linie, sau pe aceeași coloană, respectiv pe aceeași diagonală. Din modul în care se dau valori elementelor șirului *r* rezultă că nu trebuie făcute verificări pentru atacul pe linie, deoarece la pasul *i* se placează o singură regină (a *i*-a) pe a *i*-a linie. Rezultă că în funcția *nu_se_ataca* se verifică doar atacul pe coloane și diagonale. Obținem o soluție în momentul în care s-a plasat și a 8-a regină. Căutarea soluțiilor se oprește atunci când ar trebui căutată o poziție nouă pentru regina de pe linia 1 dincolo de coloana 8.

```

Program regine;
type sol=array[1..8] of Byte;
var f:file of sol;
    r:sol; { r[i]=coloana in care se afla regina din linia i }
    n:Longint; { numarul de ordine al solutiei cerute }
function nu_se_ataca(i:Byte):Boolean;
var j:Byte;
begin
    nu_se_ataca:=true;
    for j:=1 to i-1 do
        if (r[i]=r[j]) or (i-j=Abs(r[i]-r[j])) then begin nu_se_ataca:=false; Exit end
end;
procedure regina(i:Byte);
var j:Byte;
begin
    for j:=1 to 8 do
    begin
        r[i]:=j;
        if nu_se_ataca(i) then if i=8 then Write(f,r) { scrie solutia in fisier }
                                else regina(i+1)
        end
    end;
end;

```

P9. Problema pionului

Fie o tablă dreptunghiulară, împărțită în $n \times m$ căsuțe identice. Inițial, într-o din căsuțe se află un pion care se poate deplasa pe orizontală sau verticală fără a putea ieși din cadrul tablei.

Dându-se poziția inițială și cea finală a pionului, precum și de câte ori a trecut acesta prin fiecare poziție, se cere să se reconstituie traseul parcurs de pion.

Restrictii

Datele se citesc dintr-un fișier text PION.IN, având următoarea structură:

- pe prima linie este scrisă perechea n, m ;
- pe a doua linie sunt scrise coordonatele poziției inițiale;
- pe a treia linie sunt scrise coordonatele poziției finale;
- în continuare se dau, sub forma unei matrice ($a[1..n, 1..m]$), numărul de treceri ale pionului prin fiecare poziție.

Soluție

Traseul pionului este reținut într-un vector (v), având lungimea egală cu suma numărului de treceri ale pionului prin fiecare poziție. Condițiile de continuare sunt următoarele:

- poziția în care se trece trebuie să fie în cadrul tablei (funcția `inside`);
- poziția nouă (i, j) , în care se trece, trebuie să aibă elementul corespunzător în matricea a mai mare decât zero;
- numărul total de mutări efectuate până în acel moment trebuie să fie mai mic decât suma (calculată inițial) a numărului de treceri ale pionului prin fiecare poziție.

```
Program Pion;
const MaxN=10;
var a:array[1..MaxN,1..MaxN] of Word;
    v:array[1..MaxN*MaxN,1..2] of 1..MaxN;
    n,m,nrm,i,j,pil,pic,pfl,pfc:Integer;

procedure citeste;
  var f:Text;
begin
  Assign(f,'PION.IN'); Reset(f);
  Read(f,n);
  nrm:=0;
  if not Seekoln(f) then Readln(f,m);
  Read(f,pil);
  if not Seekoln(f) then Readln(f,pic);
  Read(f,pfl);
  if not Seekoln(f) then Readln(f,pfc);
```

```
for i:=1 to n do
begin
  for j:=1 to m do
    if not Seekoln(f)
      then begin Read(f,a[i,j]); nrm:=nrm+a[i,j] end;
    Readln(f)
  end;
  Close(f)
end;

function inside(q,w:Integer):Boolean;
begin
  inside:=(q>0) and (q<=n) and (w>0) and (w<=m+1);
end;

procedure back(g,h,k:Integer);
begin
  Dec(a[g,h]);
  v[nrm-k,1]:=g; v[nrm-k,2]:=h;
  if (k=0) and (g=pfl) and (h=pfc) then
  begin
    Write('Pozițiile de pe traseul pionului:');
    for i:=1 to nrm do Write('('','v[i,1]',','','v[i,2]',')');
    Readln; Halt
  end
  else
    if k>0 then
    begin
      if (inside(g-1,h)) and (a[g-1,h]>0) then back(g-1,h,k-1);
      if (inside(g+1,h)) and (a[g+1,h]>0) then back(g+1,h,k-1);
      if (inside(g,h-1)) and (a[g,h-1]>0) then back(g,h-1,k-1);
      if (inside(g,h+1)) and (a[g,h+1]>0) then back(g,h+1,k-1)
    end;
    Inc(a[g,h]);
  end;
end;

Begin
  citeste;
  back(pil,pic,nrm-1);
  Writeln('Nu există soluție!');
End.
```

Intrare

5 5	Ieșire
1 1	Pozițiile de pe traseul pionului:
1 3	(1,1) (1,2) (2,2) (1,2) (2,3) (3,3) (4,3) (4,4)
1 2 1 0 0	(3,4) (2,4) (2,3) (1,3)
0 2 2 1 0	
0 0 1 1 0	
0 0 1 1 0	
0 0 0 0 0	

P10. Generează cuvinte

Se dă un alfabet care conține v vocale și c consoane. Se cere să se genereze toate cuvintele de lungime n care nu conțin trei vocale sau trei consoane alăturate.

Restricții

Datele se citesc dintr-un fișier text, având următoarea structură:

- pe prima linie este scrisă lungimea cuvintelor (n);
- pe a doua linie este scris numărul vocalelor din alfabet;
- pe a treia linie sunt scrise vocalele alfabetului, despărțite prin spațiu;
- pe a patra linie este scris numărul consoanelor din alfabet;
- pe a cincia linie sunt scrise consoanele alfabetului, despărțite prin spațiu.

Soluție

Construim (prin backtracking) un vector a de lungime n care pe fiecare poziție are un caracter (vocală sau consoană) cu condiția să nu existe trei vocale sau consoane alăturate.

Procedura back va genera cuvintele în felul următor:

- pe primele două poziții va pune orice literă din alfabet, fără nici o restricție;
- pentru următoarele poziții va testa dacă ultimele două litere puse sunt de același fel (consoane sau vocale); în caz afirmativ pe poziția curentă va pune o literă apartinând unui tip diferit față de cele două anterioare, altfel vom putea pune orice literă.

Program GenereazaCuvintele;

```

const MaxN=10;
var a,vocale,consoane:array[1..MaxN] of Char;
    mc,mv:set of Char;
    n,c,v,i:Byte;
procedure citeste;
    var f:Text;
begin
    Assign(f,'ALFABET.IN'); Reset(f);
    Readln(f,n); Readln(f,v);
    mv:=[];
    for i:=1 to v do
        if not Seekoln(f)
            then begin Read(f,vocale[i]); mv:=mv+[vocale[i]] end;
    Readln(f); Readln(f,c);
    mc:=[];
    for i:=1 to c do
        if not Seekoln(f)
            then begin Read(f,consoane[i]); mc:=mc+[consoane[i]] end;
    Readln(f); Close(f)
end;

```

```

procedure back(k:Byte);
var g:Byte;
begin
    if k=n+1
    then
    begin
        for g:=1 to n do Write(a[g]);
        Readln
    end
    else
        if k in [1,2]
        then
        begin
            for g:=1 to v do
                begin a[k]:=vocale[g]; back(k+1) end;
            for g:=1 to c do
                begin a[k]:=consoane[g]; back(k+1) end
        end
        else
            if (a[k-1] in mc) and (a[k-2] in mc)
            then
                for g:=1 to v do
                    begin a[k]:=vocale[g]; back(k+1) end
            else
                if (a[k-1] in mv) and (a[k-2] in mv)
                then
                    for g:=1 to c do
                        begin a[k]:=consoane[g]; back(k+1) end
                else
                    begin
                        for g:=1 to v do
                            begin a[k]:=vocale[g]; back(k+1) end;
                        for g:=1 to c do
                            begin a[k]:=consoane[g]; back(k+1) end
                    end
    end;

```

Begin
citeste;
back(1)

End.

Intrare Ieșire

3	aab
1	aba
a	abb
1	baa
b	bab
	bba

```

until not as or (as and ev);
if as
then
begin
  solutie(kk,esol);
  if not esol then begin Inc(kk); st[kk]:=0 end
end
else Dec(kk)
end;
Assign(g, 'OUTPUT.TXT'); ReWrite(g);
for i:=1 to ki do Write(g,a[st1[i]]);
Close(g)
End.

```

P8. Elimină litere

Se dă un cuvânt și un dicționar. Prin operația de ștergere a unei litere din cuvântul dat se poate obține alt cuvânt. La fiecare pas se șterge o singură literă din cuvântul dat. Operația poate continua cât timp cuvântul nou (obținut prin ștergerea literei respective) aparține dicționarului.

Se cere să se determine cuvântul de lungime minimă (care aparține dicționarului) și care se poate obține din cel inițial după aplicarea operației descrise anterior.

Restricții

Datele se citesc dintr-un fișier text DICTIO.IN, având următoarea structură:

- pe prima linie este scris numărul de cuvinte din dicționar (n);
- pe a doua linie este scris cuvântul inițial (care trebuie să aparțină dicționarului);
- pe următoarele n linii se dau cuvintele dicționarului, câte unul pe o linie.

Soluție

Chiar dacă nu se cere afișarea tuturor cuvintelor, determinarea celui de lungime minimă o vom realiza prin *backtracking*, aplicând operația de ștergere în toate modurile posibile. Condiția de continuare este clară, cuvântul nou trebuie să aparțină dicționarului (funcția CuvantInDicționar). În momentul în care am obținut un cuvânt de lungime mai mică decât cel reținut în variabila *cmin*, această variabilă va fi actualizată.

Program Elimină litere;

```

uses crt;
var dicționar:array[1..100] of string;
  cuv,cmin:string;
  n,i,lmin:Byte;

```

```

procedure citeste;
  var f:Text;
begin
  Assign(f, 'DICTIO.IN'); Reset(f);
  Readln(f,n); Readln(f,cuv);
  for i:=1 to n do Readln(f,dicționar[i]);
  Close(f)
end;

function CuvantInDicționar(sir:string):Boolean;
begin
  for i:=1 to n do
    if sir=dicționar[i]
    then
      begin CuvantInDicționar:=true; Exit end;
  CuvantInDicționar:=false
end;

procedure back(cuvant:string);
  var v:Byte;
    s:string;
begin
  if Length(cuvant)<lmin
  then
    begin
      cmin:=cuvant;
      lmin:=Length(cuvant)
    end;
  for v:=1 to Length(cuvant) do
  begin
    s:=Copy(cuvant,v,1);
    Delete(cuvant,v,1);
    if CuvantInDicționar(cuvant) then back(cuvant);
    Insert(s,cuvant,v)
  end
end;

Begin
  Clrscr;
  citeste; lmin:=255;
  back(cuv);
  Writeln('Cuvântul de lungime minima este:',cmin);
End.

```

Intrare

```

3
abc
abc
ab
b

```

Ieșire

Cuvântul de lungime minima este: b

Cuvintele se vor păstra în vectorul *a*, având componente de tip **string**. Stiva este reprezentată prin vectorul *st* cu componente de tip întreg, deoarece vor reține indicii cuvintelor din vectorul *a*, (domeniul în care iau valori elementele vectorului *st* este intervalul $[0, n]$).

Literele în cadrul cuvântului format se vor păstra în vectorul *nr* având indici definiți pe subdomeștiul 'a'... 'z', iar elementele sunt de tip întreg.

Un nou nivel pe stivă se initializează prin depunerea valorii 0.

Depunerea unui nou element pe un nivel în stivă se realizează în felul următor:

- se decrementează numărul de apariție al fiecărei litere din cuvântul găsit de către pe acest nivel;
- se incrementează valoarea elementului curent dacă este strict mai mică decât *n*;
- se incrementează numărul de apariție al fiecărei litere din noul cuvânt de către pe acest nivel.

Validarea unui element depus în stivă necesită verificarea următoarelor condiții:

- nici o literă nu va avea numărul de apariție mai mare decât numărul *k*;
- de asemenea se impune condiția ca elementele din *st* să fie distințe;
- pentru evitarea permutării cuvintelor în cadrul cuvântului format, se impune ca elementele din *st* să apară în ordine crescătoare.

Am obținut o soluție dacă nivelul stivei este egal cu *n*. Evident, dacă cele *n* cuvinte pot fi alipite în condițiile impuse, atunci sirul format are lungime maximă. Dacă nu există această posibilitate, variabila *max* va fi actualizată mereu cu cel mai lung sir obținut, iar vectorul *st* va păstra configurația stivei care a condus la această lungime. Rezultatele se scriu în fișierul OUTPUT.TXT.

```
Program sir_cuvinte;
  type stiva=array[1..100] of Integer;
    num=array['a'..'z'] of Integer;
  var st,st1:stiva;
    a:array[1..1000] of string[30];
    i,j,n,kk,m,k,ki,max:Integer;
    r:Char;
    nr:num;
    as,ev,esol:Boolean;
    g,f:Text;

  procedure succ(k:Integer; var st:stiva; var nr:num;
                           var as:Boolean);
  begin
    var c:string[30]; i:Integer;
    begin
      if st[k]<n
      then
        begin
          if st[k]<>0
```

```
      then
        begin
          c:=a[st[k]];
          for i:=1 to Length(c) do Dec(nr[c[i]]);
        end;
        Inc(st[k]);
        c:=a[st[k]];
        for i:=1 to Length(c) do Inc(nr[c[i]]);
        as:=true
      end
      else as:=false
    end;

  procedure valid(k:Integer; st:stiva; nr:num; var ev:Boolean);
  begin
    ev:=true;
    for r:='a' to 'z' do if nr[r]>k then ev:=false;
    for i:=1 to k-1 do if st[i]=st[k] then ev:=false;
    if st[k-1]>st[k] then ev:=false
  end;

  procedure solutie(k:Integer; var esol:Boolean);
  var ni:Integer;
  begin
    ni:=0;
    for i:=1 to k do ni:=ni+Length(a[st[i]]);
    if ni>max
    then
      begin
        for i:=1 to k do st1[i]:=st[i];
        max:=ni; ki:=k
      end;
    esol:=k=n
  end;

Begin
  Assign(f,'INPUT.TXT'); Reset(f);
  Readln(f,k);
  n:=0;
  while not Eof(f) do
  begin Inc(n); Readln(f,a[n]) end;
  for r:='a' to 'z' do nr[r]:=0;
  max:=0;
  kk:=1;
  st[kk]:=0;
  while kk>0 do
  begin
    repeat
      succ(kk,st,nr,as);
      if as then valid(kk,st,nr,ev)
```

P6. Generare de şiruri

Să se determine toate şirurile de şase numere distincte mai mici decât un număr dat ($n \leq 100$) a căror sumă este egală cu numărul s ($s \leq 50$). Cele două numere (n și s) se citesc de la tastatură. Se cere, de asemenea și numărul acestor şiruri.

Pe fiecare linie a fișierului de ieșire OUTPUT.TXT se va scrie câte o combinație de şase numere care au suma egală cu s . Pe ultima linie se va scrie numărul şirurilor.

Exemplu:

Intrare

$n=50$ $s=24$

Ieșire

```
1 2 3 4 5 9
1 2 3 4 6 8
1 2 3 5 6 7
3
```

Soluție

Deoarece se cere determinarea tuturor soluțiilor, vom utiliza *metoda backtracking*. Prezentăm o implementare recursivă.

Procedura *back* are doi parametri (i de tipul Byte și s de tipul Integer) reprezentând indicele elementului curent din şirul care se generează, respectiv suma care trebuie acoperită în pașii care urmează.

Pentru a nu genera de două ori un anumit număr ca termen în şir, un termen nou trebuie să fie cel puțin cu 1 mai mare decât termenul anterior. Păstrăm în sp valoarea rămasă până la s după însumarea celor i termeni din şir. Valorile sp se depun pe stivă, deoarece fac parte din contextul apelului.

Am obținut o soluție atunci când valoarea lui sp este 0 și avem 6 termeni, deoarece în acest moment suma celor 6 elemente ale şirului este egală cu s .

```
Program siruri;
var sol:array[0..6] of Byte;
    s:Integer;
    n,nr_sol:Byte;
    g:Text;

procedure scrie;
    var i:Byte;
begin
    Inc(nr_sol);
    for i:=1 to 6 do
        Write(g,sol[i], ' ');
    Writeln(g)
end;

procedure back(i:Byte; s:Integer);
    var j:Byte;
    sp:Integer;
begin
```

```
begin
    for j:=sol[i-1]+1 to n do
    begin
        sp:=s-j; sol[i]:=j;
        if (i=6) and (sp=0)
        then scrie
        else
            if (sp>0) and (i<6) then back(i+1,sp)
    end;
end;
```

Begin

```
Write('n= '); Readln(n);
Write('s= '); Readln(s);
Assign(g,'OUTPUT.TXT');
Rewrite(g);
nr_sol:=0; sol[0]:=0;
back(1,s);
Writeln(g,nr_sol); Close(g)
End.
```

P7. Concatenare de cuvinte

Se consideră n cuvinte și un număr natural (k). Să se determine cel mai lung cuvânt format prin alipirea cuvintelor date care respectă condiția: nici o literă nu apare în cadrul cuvântului de lungime maximă de mai mult de k ori.

Datele de intrare se găsesc în fișierul INPUT.TXT.

Pe prima linie a fișierului este scris numărul k , iar pe următoarele n linii sunt scrise cele n cuvinte. Cuvântul obținut ca rezultat va fi scris în fișierul OUTPUT.TXT.

Se va afișa o singură soluție.

Exemplu:

Intrare

```
3
cand
poate
frunza
pica
toamna
```

Ieșire

candpoatefrunza

Soluție

Ordinea literelor în cadrul unui cuvânt nu este importantă. De aceea putem codifica un cuvânt printr-un şir de numere care reprezintă de câte ori apare o literă oarecare în cadrul acelui cuvânt. În acest fel această problema devine asemănătoare cu determinarea unei submulțimi de sumă dată și având un număr maxim de elemente. Putem aplica metoda *programării dinamice*, dar pentru seturi mari de date este preferată *metoda backtracking*; prezentăm o implementare nerecursivă.

```

procedure solutie(k:Integer; var esol:Boolean);
var su:Integer;
begin
  su:=0;
  for i:=1 to k do su:=su+a[st[i]];
  esol:=su=s;
end;
procedure tipar(k:Integer; st:stiva);
var t:Integer;
begin
  for t:=1 to k do Write(g,a[st[t]],' '); Writeln(g);
  Inc(nr)
end;
Begin
  Write('n='); Readln(n);
  Write('s='); Readln(s);
  Assign(g,'OUTPUT.TXT'); ReWrite(g);
  Write('Sirul de numere: ');
  for i:=1 to n do Read(a[i]);
  k:=1; st[k]:=0; nr:=0;
  while k>0 do
    begin
      repeat
        succ(k,st,as);
        if as then valid(k,st,ev)
      until not as or (as and ev);
      if as
      then
        begin
          solutie(k,esol);
          if esol then tipar(k,st)
          else begin Inc(k); st[k]:=0 end
        end
        else Dec(k)
      end;
      Write(g,nr); Close(g)
    End.

```

P5. Generare de funcții

Din fișierul standard de intrare se citesc două mulțimi de numere întregi A și B (date fiecare pe câte o linie).

Să se genereze toate funcțiile $f:A \rightarrow B$ și să se memoreze în fișierul text FUNCTII (o funcție pe o linie).

Să se afișeze fișierul creat.

Soluție

Deoarece se cer toate funcțiile, vom aplica metoda backtracking, implementată iterativ. Observăm că funcțiile cerute sunt determinate de sirul imaginilor elementelor a_1, a_2, \dots, a_n . Pornim cu configurația: b_1, b_1, \dots, b_1 , apoi mărim ultimul indice rând pe rând la valorile 2, 3, ..., m , după care se mărește penultimul indice cu 1; ultimul indice va lua din nou toate valorile de la 1 până la m . Apoi se mărește din nou penultimul indice, și a.m.d., generarea terminându-se după obținerea configurației b_m, b_m, \dots, b_m .

```

Program generare_functii;
var a,b:array[1..20] of Integer;
f:array[1..10] of Byte;
n,m,i,j:Byte;
fo:Text;
Begin
  Assign(fo,'FUNCTII'); ReWrite(fo);
  n:=0;
  while not Eoln do begin Inc(n); Read(a[n]) end; Readln;
  m:=0;
  while not Eoln do begin Inc(m); Read(b[m]) end;
  for i:=1 to n do
    begin Write(fo,a[i]:3); f[i]:=1 end;
  for i:=1 to n do Write(fo,b[i]:3); Writeln(fo);
  i:=n;
  repeat
    while f[i]<m do
      begin
        f[i]:=f[i]+1;
        for j:=1 to n do Write(fo,a[j]:3);
        for j:=1 to n do Write(fo,b[f[j]]:3);
        i:=n; Writeln(fo)
      end;
      f[i]:=1; i:=i-1
    until i=0;
  Close(fo);
  Reset(fo);
  while not Eof(fo) do
    begin
      Writeln; Write(' x:');
      for i:=1 to n do Read(fo,a[i]);
      for i:=1 to n do Write(a[i]:3); Writeln;
      Write('f(x):');
      for i:=1 to n do Read(fo,b[i]);
      Readln(fo);
      for i:=1 to n do Write(b[i]:3)
    end;
  Writeln; Readln
End.

```

```

    then
begin
  sol[i,j]:=pas;
  if (i=xf) and (j=yf)
  then scriere(f2)
  else
begin
  labirint(i-1,j,pas+1);
  labirint(i+1,j,pas+1);
  labirint(i,j-1,pas+1);
  labirint(i,j+1,pas+1)
end;
  sol[i,j]:=0
end;
end;

Begin
Assign(f1,'LABIRINT.TXT'); Reset(f1);
Assign(f2,'TRASEE.TXT'); ReWrite(f2);
Readln(f1,m,n); Citire(f1,lab);
Close(f1);
repeat
  Write('Coord, initiale si finale:');
  Readln(xi,yi,xf,yf)
until ([xi,xf]<=[1..m]) and ([yi,yf]<=[1..n]);
labirint(xi,yi,1);
Close(f2);
Reset(f2);
while not Eof(f2) do
begin
  citire(f2,scl);
  Inc(k);
  Writeln('solutia ',k,':');
  scriere(Output)
end;
Close(f2)
End.

```

P4. Submulțimi

- Se consideră o mulțime formată din n elemente numere întregi.
- Să se genereze toate submulțimile acestei mulțimi având proprietatea că suma elementelor lor este egală cu s .
- Datele de intrare se citesc de la tastatură.
- Elementele submulțimilor vor fi scrise pe câte o linie în fișierul de tip text, denumit OUTPUT.TXT; ultima linie va conține numărul de submulțimi.

Exemplu:

Intrare

$n=5$
 $s=4$
Sirul de numere: 1 3 2 -1 5

Ieșire

1 3
3 2 -1
-1 5
3

Soluție

Deoarece se cer toate submulțimile vom aplica *metoda backtracking*. Prezentăm o implementare nerecursivă. Numerele date se vor păstra în vectorul a . Stiva este reprezentată prin vectorul st , având componente de tip întreg. Deoarece acestea sunt indicii elementelor din vectorul a , ele iau valori din intervalul $[0, n]$.

Un nou nivel pe stivă se initializează prin depunerea valorii 0. Un nou element pe un nivel pe stivă se determină prin incrementarea valorii elementului curent. Validarea unui element depus în stivă constă în verificarea condiției ca un element să nu fi fost depus anterior. De asemenea, pentru evitarea repetării soluțiilor (datorită comutativității adunării) numerele se vor depune pe stivă în ordine crescătoare.

Considerăm că am obținut o soluție dacă suma valorilor depuse pe stivă este egală cu numărul s dat la intrare.

```

Program submultimi;
type stiva=array[1..100] of Integer;
var st,a:stiva;
  i,j,n,k,s,nr:Integer;
  as,ev,esol:Boolean;
  g:Text;
procedure succ(k:Integer; var st:stiva; var as:Boolean);
begin
  if st[k]<n
  then
  begin
    st[k]:=st[k]+1; as:=true
  end
  else as:=false
end;
procedure valid(k:Integer; st:stiva; var ev:Boolean);
begin
  ev:=true;
  if k>1
  then
    if st[k]<=st[k-1] then ev:=false;
  for i:=1 to k-1 do
    if st[k]=st[i] then ev:=false
end;

```

```

procedure scrie;
var i,j:Byte;
begin
  for i:=1 to 8 do
  begin
    for j:=1 to 8 do
      if r[i]=j then Write(' R.')
      else Write(' .');
    Writeln;
  end;
end;
Begin
Assign(f,'REGINE'); Rewrite(f);
regina(1);
Close(f);
Reset(f);
Write('Numarul de ordine al solutiei(1-92):');
While not Eof do
begin
  Readln(n);
  if n in [1..92]
  then
  begin
    Seek(f,n-1); Read(f,r); scrie
  end
  else Writeln('Numarul de ordine trebuie sa fie in [1..92]');
  Write('Numarul de ordine al solutiei(1-92):')
end;
Close(f)
End.

```

P3. Labirint

Fișierul text LABIRINT.TXT conține pe prima linie dimensiunile m și n ale labirintului, iar pe următoarele linii sunt elementele tabloului bidimensional corespunzător labirintului. Culoarele labirintului sunt codificate prin valori egale cu 1, zidurile cu 0. O persoană trebuie să ajungă din punctul corespunzător coordonatelor (x_i, y_i) în poziția (x_f, y_f) , traversând labirintul, deplasarea făcându-se pe orizontală și pe verticală.

Din fișierul standard de intrare se vor citi coordonatele punctului inițial (x_i, y_i) și coordonatele punctului final (x_f, y_f) ale traseului.

Să se creeze un fișier text TRASEE.TXT cu toate traseele prin care se poate ajunge din punctul (x_i, y_i) în (x_f, y_f) . Un traseu nu va conține aceeași poziție de două ori.

Să se afișeze conținutul fișierului pe ecran.

Soluție

Labirintul se păstrează în tabloul lab , iar soluțiile cu traseele din labirint se scriu în tabloul sol . Deoarece se cer toate drumurile vom aplica metoda *backtracking*. Prezentăm o implementare recursivă.

Tabloul sol se inițializează cu 0 și pe parcursul căutării traseului din labirint acesta se marchează cu valoarea pasului (păstrată în variabila pas). Pentru a decide dacă un "pas" din traseu este corect ales sau nu, trebuie verificată valoarea din tabloul lab (o valoare diferită de 1 semnifică faptul că în poziția respectivă este zid) și valoarea din tabloul sol (o valoare diferită de 0 semnificând faptul că poziția respectivă deja face parte din traseu). În cazul îndeplinirii condițiilor de continuare se efectuează pasul și se verifică dacă s-a ajuns în poziția finală, caz în care se poate afișa o soluție.

Dintr-o anumită poziție (i, j) , traseul se poate continua în patru direcții: în linile $i - 1$ și $i + 1$, respectiv coloanele $j - 1$ și $j + 1$.

După tipărire unei soluții, respectiv abandonarea unei ramificații, elementul curent din tabloul sol se reinițializează cu 0 (se șterge din traseu).

```

Program trasee_in_labirint;
type tablou=array[0..10,0..20] of Byte;
var f1,f2:Text;
  sol,lab:tablou;
  xi,xf,yi,yf,m,n,k:Byte;
procedure citire(var f:Text; var x:tablou);
var i,j:Byte;
begin
  for i:=1 to m do
  begin
    for j:=1 to n do
      Read(f,x[i,j]);
    Readln(f)
  end;
end;
procedure scriere(var f:Text);
var i,j:Byte;
begin
  for i:=1 to m do
  begin
    for j:=1 to n do
      Write(f,sol[i,j]:3);
    Writeln(f)
  end;
end;
procedure labirint(i,j,pas:Byte);
begin
  if (lab[i,j]=1) and (sol[i,j]=0)

```

Capitolul 1**Backtracking****Problema 1**

Enunț. Avem la dispoziție 6 culori: alb, galben, roșu, verde, albastru și negru. Să se precizeze toate drapelele tricolore care se pot proiecta, știind că trebuie respectate următoarele reguli:

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

Exemple: "alb galben roșu", "roșu verde galben"

Rezolvare. Folosim o stivă cu 3 nivele, reprezentând cele trei culori de pe drapel și codificăm culorile prin numere:

- 1 - alb
- 2 - galben
- 3 - roșu
- 4 - verde
- 5 - albastru
- 6 - negru

Folosim un vector auxiliar fol de tip boolean:

fol[i] = TRUE, dacă culoarea i a fost folosită în drapel deja;
FALSE, dacă nu a fost folosită.

Condițiile care trebuie îndeplinite pentru așezarea unei anumite culori c pe un nivel al stivei sunt:

- fol[c]=FALSE, adică să nu fi folosită deja culoarea respectivă în drapel;
- dacă nivelul stivei este 2 (adică culoarea din mijloc), atunci culoarea trebuie să fie 2 sau 4 (galben sau verde).

program Drapel;

```
const Culoare:array [1..6] of string[10]=('alb','galben',
                                           'roșu','verde','albastru','negru');
```

```
var ST :array [1..3] of integer;
```

```

Fol      :array [1..6] of boolean;
k, i      :integer;

procedure Printsol;
var i:integer;
begin
  for i:=1 to 3 do write(Culoare[ST[i]], ' ');
  writeln;
end;

function Valid:boolean;
begin
  if Fol[ST[k]] then Valid:=false
  else
    if k>2 then Valid:=true
    else Valid:=(ST[k] in [2,4])
end;

begin
  for i:=1 to 6 do Fol[i]:=false;
  k:=1;
  ST[k]:=0;
  while k>0 do
    begin
      repeat
        ST[k]:=ST[k]+1
      until (ST[k]>6) or Valid;
      if ST[k]<=6 then
        if k=3 then Printsol
        else
          begin
            Fol[ST[k]]:=true;
            k:=k+1;
            ST[k]:=0
          end
        else
          begin
            k:=k-1;
            if k>0 then Fol[ST[k]]:=false
          end
    end
  end.

```

Problema 2

Enunț. Să se descompună un număr natural N , în toate modurile posibile, ca sumă de P numere naturale ($P \leq N$).

Rezolvare. Folosim o stivă cu P nivele, unde fiecare nivel ia valoarea unui termen din sumă. Variabila S reține suma primelor k nivele pentru a putea testa validitatea elementului așezat pe poziția curentă, fără a mai face o parcurgere suplimentară a stivei. Condiția de validitate devine: $S+ST[k] \leq N$.

Pentru a evita afișarea descompunerilor identice, cu ordinea termenilor schimbăță, folăm ordinea crescătoare (nu strict crescătoare) a termenilor din sumă, prin initializarea unui nivel din stivă cu valoarea nivelului anterior.

```

program Descompuneri;
var n, p, k, S, i           :integer;
    ST                      :array [0..1000] of integer;

Procedure Printsol;
var i:integer;
begin
  for i:=1 to p do write(ST[i], ' ');
  writeln;
end;

begin
  write('N= '); readln(n);
  write('P= '); readln(p);
  S:=0;
  k:=1;
  fillchar(ST,sizeof(ST),0);
  while k>0 do
    begin
      ST[k]:=ST[k]+1;
      if S+ST[k]<=N then
        if (k=p) and (S+ST[k]=N) then Printsol
        else
          begin
            S:=S+ST[k];
            k:=k+1;
            ST[k]:=ST[k-1]+1
          end
        else
          begin
            k:=k-1;
            S:=S-ST[k]
          end
    end
  end.

```

Problema 3

Enunț. Dintron un grup de N persoane, dintre care P femei, trebuie formată o delegație de C persoane, dintre care L femei. Să se precizeze toate delegațiile care se pot forma.

Rezolvare. vom nota femeile cu numerele de la 1 la P , și bărbații de la $P+1$ la N . Pentru rezolvarea problemei folosim o stivă cu C nivele, unde nivelele $1 \dots L$ conțin indicii femeilor din delegația curentă, și nivelele $L+1 \dots C$ indicii bărbaților.

$st[i]$ poate lua valori între $st[i-1]$ și MAX , unde:

$MAX=P$, pentru $i \leq L$;
 $MAX=N$, pentru $i > L$.

$st[1]$ poate lua valori între p și n .

Motivul pentru care $st[i]$ se initializează cu $st[i-1]$ este evitarea duplicării delegațiilor, prin forțarea ordinii strict crescătoare în fiecare delegație.

Limita până la care se pot majora elementele de pe un nivel este P pentru nivelele $1 \dots L$ (femeile) și N pentru nivelele $L+1 \dots C$. De aceea, la urcarea și coborârea pe nivelul L , variabila MAX devine N , respectiv P .

$st[1]$ nu se initializează cu $st[1-1]$, ci cu P , pentru că este primul dintre bărbații și bărbații au indicii între $P+1$ și N .

program Delegatie;

```
var st :array [1..100] of integer;
      MAX, k, N, P, C, L :integer;
procedure Printsol;
var i:integer;
begin
  for i:=1 to C do write(st[i], ' ');
  writeln;
end;

begin
  write('N, P, C, L: ');
  readln(N,P,C,L);
  k:=1;
  st[1]:=0;
  MAX:=P;
  while k>0 do
    begin
```

```
      st[k]:=st[k]+1;
      if st[k]<=MAX then
        if k=C then Printsol
        else
          begin
            k:=k+1;
            if k>L+1 then st[k]:=st[k-1]
            else
              begin
                st[k]:=P;
                MAX:=N
              end
            end
          end
        else
          begin
            k:=k-1;
            if k=L then MAX:=P
          end
        end
      end.
```

Problema 4

Enunț. O persoană are la dispoziție un rucsac cu o capacitate de G unități de greutate și intenționează să efectueze un transport în urma căruia să obțină un câștig. Persoanei î se pun la dispoziție N obiecte. Pentru fiecare obiect se cunoaște greutatea (mai mică decât capacitatea rucsacului) și câștigul obținut în urma transportului său.

Ce obiecte trebuie să aleagă persoana pentru a-și maximiza câștigul și care este acesta? Se va avea în vedere și faptul că pentru același câștig persoana să transporte o greutate mai mică.

Datele de intrare se vor citi din fișierul input.txt. Pe primele două linii ale fișierului sunt N și G , numărul de obiecte și capacitatea rucsacului. Pe fiecare din următoarele N linii se află două numere întregi reprezentând greutatea și câștigul asociat obiectului respectiv.

Rezolvare. În vectorii gr și c avem greutatea și câștigul obținut în urma transportării fiecărui obiect. CG este câștigul obținut în urma transportării obiectelor așezate în stivă până la nivelul curent și s , greutatea acestor obiecte.

Condițiile care trebuie îndeplinite pentru așezarea unui obiect i pe nivelul k al stivei sunt:

1. $Pos[i]=\text{FALSE}$, adică să nu fi așezat deja obiectul respectiv în stivă.

2. $S + gr[ST[k]] \leq G$, adică greutatea obiectelor să fie mai mică sau cel mult egală cu greutatea maximă care poate fi transportată cu rucsacul.

În variabilele Cmax, Gmax și STmax (câștigul, greutatea și obiectele) reținem pe timpul execuției programului cea mai bună soluție găsită.

Notă: Rezolvarea optimă a acestei probleme poate fi găsită în manualul "Tehnici de programare" de Tudor Sonin la capitolul "Programare dinamică".

```

program Rucsac;

var i,k,n,G,CG,S,Cmax,Gmax,NOb :integer;
    c,gr,ST,STmax           :array [1..100] of integer;
    Pus                      :array [1..100] of boolean;

procedure Citire;
var f      :text;
    i      :integer;
begin
    assign(f,'input.txt');
    reset(f);
    readln(f,N);
    readln(f,G);
    for i:=1 to n do readln(f,gr[i],c[i]);
    close(f)
end;

function Valid:boolean;
begin
    if Pus[ST[k]] then Valid:=false
    else
        if S+gr[ST[k]]<=G then
            begin
                S:=S+gr[ST[k]];
                CG:=CG+c[ST[k]];
                Valid:=true
            end
        else Valid:=false
end;

begin
    Citire;
    k:=1;
    for i:=1 to n do Pus[i]:=false;
    ST[k]:=0;
    CG:=0;
    S:=0;
    Cmax:=0;

```

```

    while k>0 do
    begin
        repeat
            ST[k]:=ST[k]+1
        until (ST[k]>n) or Valid;
        if ST[k]<=n then
            begin
                if CG>Cmax then
                    begin
                        Cmax:=CG;
                        NOb:=k;
                        for i:=1 to k do STmax[i]:=ST[i];
                        Gmax:=S;
                    end;
                Pus[ST[k]]:=true;
                k:=k+1;
                ST[k]:=0
            end
        else
            begin
                k:=k-1;
                if k>0 then
                    begin
                        S:=S-gr[ST[k]];
                        CG:=CG-c[ST[k]];
                        Pus[ST[k]]:=false
                    end
            end
        end;
    end;

    writeln('Cistigul maxim: ',Cmax);
    writeln('Greutate transportata: ',Gmax);
    writeln('Obiectele transportate: ');
    for i:=1 to NOb do write(STmax[i],' ');
    writeln
end.

```

Problema 5

Enunț. Fiind dat un număr natural N, se cere să se afișeze toate descompunerile sale ca sumă de numere prime.

Rezolvare. Pentru eficiență, vom calcula mai întâi toate numerele prime mai mici ca N; acestea vor fi memorate în vectorul Numar.

Rutina de backtracking este similară cu cea din problema 2, cu excepția faptului că termenii pot lua valori din Numar[1..P] și că descompuneră nu trebuie să aibă un număr fix de termeni.

```

Program SumaPrime;

var ST, Numar
  n, K, P, S, i
    :array[0..1000] of integer;
    :integer;

Function Prim( X : integer):boolean;
var i:integer;
begin
  Prim:=true;
  i:=2;
  while i*i<=X do
  begin
    if X mod i=0 then
      begin
        Prim:=false;
        i:=X
      end;
    i:=i+1;
  end;
end;

procedure Printsol;
var i:integer;
begin
  for i:=1 to k do
    write(Numar[ST[i]], ' ');
  writeln
end;

begin
  fillchar(ST,sizeof(ST),0);
  fillchar(Numar,sizeof(Numar),0);
  write('N=');
  readln(N);
  P:=0;
  for i:=2 to N do
    if Prim(i) then
    begin
      inc(P);
      Numar[P]:=i
    end;
  k:=1;
  S:=0;
  while k>0 do

```

```

begin
  ST[k]:=ST[k]+1;
  if (ST[k]<=P) and (S+Numar[ST[k]]<=N) then
    if S+Numar[ST[k]]=N then Printsol
    else
      begin
        S:=S+Numar[ST[k]];
        k:=k+1;
        ST[k]:=ST[k-1]-1
      end
    else
      begin
        k:=k-1;
        S:=S-Numar[ST[k]]
      end
  end
end.

```

Problema 6

Enunț. ("Attila și regele") Un cal și un rege se află pe o tablă de șah. Unele câmpuri ale tablei sunt "arse", pozițiile lor fiind cunoscute. Calul nu poate călca pe câmpuri "arse", iar orice mișcare a calului face ca respectivul câmp să devină "ars". Să se afle dacă există o succesiune de mutări permise (cu restricțiile de mai sus), prin care calul să poată ajunge la rege și să revină la poziția inițială. Poziția inițială a calului, precum și poziția regelui sunt considerate "nearse".

Rezolvare. Vom folosi o stivă triplă cu elemente de tip pozitie: l,c - linia și coloana poziției curente, mut - mutarea făcută din această poziție.

Vectorul Mutari (constant) conține cele 8 mutări pe care le poate efectua un cal (modificările coordonatelor calului prin mutarea respectivă; vezi și problema 4).

Pentru început vom cîții datele de intrare cu procedura Citire. Fișierul de intrare input.txt trebuie să aibă forma:

N	// pe prima linie, n - numărul de linii (coloane) al tablei
lCal cCal	// pe următoarele două linii, poziția inițială a
lRege cRege	// calului și a regelui
a ₁₁ a ₁₂ a ₁₃ ... a _{1n}	// pe următoarele n linii, tabla de șah
...	// a ₁₁ = 0, dacă pătrățelul respectiv e ars
a _{n1} a _{n2} a _{n3} ... a _{nn}	// a _{nn} = 1, dacă nu este ars

Vom face o "bordare" a tablei de șah cu două rânduri de pătrățele "arse". De exemplu dacă tabla este 3x3 și notăm cu 0 pătrățele arse și cu 1 pe cele nearse:

```

0000000
0000000
0011100
0011100
0011100
0000000
0000000

```

Această bordare este utilă pentru că nu trebuie să mai verificăm dacă coordonatele rezultate în urma efectuării unei mutări sunt pe tablă de şah.

Astfel, presupunând că 1 și c sunt coordonatele rezultate în urma unei mutări, condiția de validitate a acelei mutări va fi:

```

if a[l,c]=1 then ... , în loc de,
if (l>=1) and (l<=n) and (c>=1) and (c<=n) and (a[l,c]=1)
then ...

```

Condiția de validitate a unei mutări este ca poziția rezultată în urma mutării să fie pe un pătrat "nears" și diferită de poziția inițială a calului dacă acesta nu a ajuns încă la rege. Dacă poziția rezultată în urma mutării este poziția inițială a calului și acesta a ajuns la rege atunci programul se termină cu afișarea soluției (variabila Terminat devine TRUE).

Dacă mutarea este corectă, se "arde" câmpul respectiv și se verifică dacă poziția rezultată în urma mutării este poziția regelui pentru dă valoarea TRUE variabilei Rege (care este TRUE dacă s-a ajuns la rege și FALSE, altfel). La trecerea pe un nivel superior al stivei, se initializează acest nivel cu poziția curentă.

La trecerea pe un nivel inferior al stivei, se marchează poziția respectivă ca "nears" și, dacă poziția de pe care se coboară în stivă era poziția regelui, variabila Rege devine FALSE.

```

program attila;

const Mutari:array [1..8,1..2] of integer=
      ((-2,1),(-1,2),(1,2),(2,1),(2,-1),(1,-2),(-1,-
2),(-2,-1));

type pozitie=record
           l,c,mut:integer;
         end;

var N, lCal, cCal, lRege, cRege, k, nlin, ncol, i :integer;
    T :array [-1..22,-1..22] of integer;
    ST :array [1..400] of pozitie;
    Rege, Terminat :boolean;

```

```

procedure citire;
var f:text;
    i,j:integer;
begin
  assign(f,'input.txt');
  reset(f);
  readln(f,N);
  readln(f,lCal,cCal);
  readln(f,lRege,cRege);
  for i:=1 to n do
    begin
      for j:=1 to n do
        read(f,T[i,j]);
      readln(f);
    end;
  close(f);
end;

procedure Printsol;
var i:integer;
begin
  for i:=1 to k do writeln(ST[i].l,' ',ST[i].c);
  writeln(lCal,' ',cCal)
end;

function Valid:boolean;
begin
  nlin:=ST[k].l+Mutari[ST[k].mut,1];
  ncol:=ST[k].c+Mutari[ST[k].mut,2];
  if (nlin=lCal) and (ncol=cCal) then
    if Rege then
      begin
        Terminat:=true;
        Valid:=true;
      end
    else Valid:=false
  else
    if T[nlin,ncol]=1 then Valid:=true
    else Valid:=false
  end;
begin
  Citire;
  for i:=-1 to N+2 do
    begin
      T[-1,i]:=0; T[0,i]:=0; T[n+1,i]:=0; T[n+2,i]:=0;
      T[i,-1]:=0; T[i,0]:=0; T[i,n+1]:=0; T[i,n+2]:=0
    end;
  k:=1;
end;

```

```

ST[1].mut:=0;
ST[1].l:=lcal;
ST[1].c:=ccal;
Rege:=false;
Terminat:=false;
while (k>0) and (not Terminat) do
begin
repeat
  ST[k].mut:=ST[k].mut+1
until (ST[k].mut>8) or Valid;
if ST[k].mut<=8 then
  if Terminat then Printsol
  else
  begin
    k:=k+1;
    ST[k].mut:=0;
    ST[k].l:=nlin;
    ST[k].c:=ncol;
    T[nlin,ncol]:=0;
    if (nlin=lRege) and (ncol=cRege) then Rege:=true
  end
  else
  begin
    T[ST[k].l,ST[k].c]:=1;
    if (ST[k].l=lRege) and (ST[k].c=cRege) then
      Rege:=false;
    k:=k-1
  end
end;
if k=0 then writeln('Nu exista solutie.')
end.

```

Problema 7

Enunț. (Proprietăți numerice) Numărul 123456789 are câteva caracteristici amuzante: Astfel, înmulțit cu 2,4,7 sau 8 dă ca rezultat tot un număr de nouă cifre distințe (în afară de 0):

$$123456789 \cdot 2 = 246913578$$

$$123456789 \cdot 4 = 493827156$$

$$123456789 \cdot 7 = 864197523$$

$$123456789 \cdot 8 = 987654312$$

Această proprietate nu funcționează pentru numerele 3,6 sau 9. Există totuși multe numere de nouă cifre distințe (fără 0) care înmulțite cu 3 au aceeași proprietate. Să se listeze toate aceste numere în care ultima cifră este 9.

Rezolvare. În stivă se vor reține cifrele numărului căutat. Pentru că ultima cifră este întotdeauna 9, stiva va avea numai 8 poziții. Condiția de "valid" este ca cifra selectată să nu fi fost folosită anterior (de fapt se generează toate permutările cifrelor 1..8).

Datorită faptului că la fiecare nivel în stivă se alege cea mai mică cifră nefolosită care nu a fost încă încercată, numerele vor fi generate în ordine crescătoare. Astfel, dacă înmulțind cu 3 numărul obținut la pasul curent se va obține un număr pe 10 cifre vom ști că și următoarele numere vor da un produs pe 10 cifre. Rezultă că procesul se oprește când întâlneste numărul 341256789 (cel mai mic număr cu cifrele distințe, mai mare decât 333333333).

```

program Numeric;

var ST,U :array[0..10] of integer;
  K,i,f :integer;
  e :double;
  t :set of char;
  s :string;

begin
  k:=1;
  fillchar(ST,sizeof(ST),0);
  fillchar(U,sizeof(U),0);
  ST[9]:=9;
  U[9]:=1;
  while k>0 do
    if k=9 then
    begin
      e:=0; { se validează o posibilă soluție }
      for f:=1 to 9 do e:=e*10+ST[f];
      e:=e*3; { se înmulțește numărul cu 3 }
      str(e:0:0,s); { și se verifică dacă cifrele }
      t:={}; { rezultatului sunt distințte }
      i:=0;
      if length(s)=10 then exit;
      for f:=1 to length(s) do
        if not (s[f] in t) then
        begin
          i:=i+1;
          t:=t+[s[f]]
        end;
      if i=9 then
      begin
        for f:=1 to 9 do write(ST[f]);
        writeln
      end;
    end;
  end;
end.

```

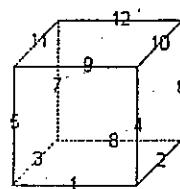
```

end;
k:=k-1
end
else
begin
  U[ST[k]]:=0;
repeat
  ST[k]:=ST[k]+1
until (U[ST[k]]=0) or (ST[k]>8);
if ST[k]>8 then
begin
  ST[k]:=0;
  k:=k-1
end
else
begin
  U[ST[k]]:=1;
  k:=k+1
end
end
end.

```

Problema 8

Enunț. Să se dispună pe cele 12 muchii ale unui cub toate numerele de la 1 la 12, astfel încât suma numerelor aflate pe muchiile unei fețe să fie aceeași pentru toate fețele.



Considerăm muchiile numerotate ca în figură:

Ieșirea va fi într-un fișier text având numele `solcub.txt` care va conține căte o soluție pe fiecare linie sub forma:

1 m₂ m₃ m₄ m₅ m₆ m₇ m₈ m₉ m₁₀ m₁₁ m₁₂

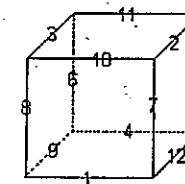
...

1 m₂ m₃ m₄ m₅ m₆ m₇ m₈ m₉ m₁₀ m₁₁ m₁₂

unde m_i este valoarea din mulțimea {2, ..., 12} plasată pe muchia i, i=1..12, iar muchia 1 conține valoarea 1 în toate soluțiile.

O soluție corectă este cea de mai jos, ilustrată și în figură:

1 12 9 6 8 5 7 4 11 3 2 10



Baraj Sibiu, 1996

Rezolvare. Notăm cu F₁, ..., F₆ suma numerelor de pe muchiile fețelor cubului.

$$F_1 = F_2 = \dots = F_6 = F$$

F₁+F₂+F₃+F₄+F₅+F₆ = 2(m₁+m₂+...+m₁₂), pentru că fiecare muchie apare în două fețe ale cubului.

Deci, 6*F=2*(m₁+m₂+...+m₁₂),
dar m₁+m₂+...+m₁₂ = 1+2+...+12 = 78 => 6*F=2*78=156 => F=26.

Deci suma muchiilor de pe fiecare față trebuie să fie 26.

Observăm că este suficient să folosim doar 6 nivele ale stivei, celelalte valori deducându-se din acestea 6. Astfel, m₁=1 întotdeauna. După ce așezăm m₂ și m₃, m₈ se deduce că fiind 26-m₁-m₂-m₃. Analog, după ce așezăm m₄ și m₅ se deduce m₉=26-m₄-m₅. La fel, se deduc m₁₀=26-m₂-m₄, m₁₁=26-m₅-m₇-m₉ și m₁₂=26-m₃-m₁₀-m₁₁.

Deci numai 6 muchii variază independent una de celelalte: m₂, ..., m₇.

Vectorul auxiliar Pus este folosit pentru a ști ce numere au fost deja așezate în stivă.

În concurs, la această problemă a fost impus un timp de execuție de maxim 10 secunde. Programul face aranjamente de 11 luate câte 6, timpul de calcul fiind O(A₁₁⁶)≈55000, adică sub o secundă. Dacă se utiliza soluția banală, adică permutări de 11, timpul de calcul era O(11!)≈39 milioane și timpul de execuție era de aproximativ 30 de secunde.

```

program cub_magic;

var ST: array [2..12] of integer;
    Pus: array [1..26] of boolean;
    k, i: integer;

function Valid:boolean;
begin
  if Pus[ST[k]] then Valid:=false
  else
    case k of
      2:begin
        Pus[ST[2]]:=true;
        Valid:=true
      end;
      3:begin
        Pus[ST[3]]:=true;
        ST[8]:=26-ST[2]-ST[3]-1;
        if not Pus[ST[8]] then
          begin
            Pus[ST[8]]:=true;
            Valid:=true
          end
        else
          begin
            Pus[ST[3]]:=false;
            Valid:=false
          end
      end;
      4:begin
        Pus[ST[4]]:=true;
        Valid:=true
      end;
      5:begin
        Pus[ST[5]]:=true;
        ST[9]:=26-ST[5]-ST[4]-1;
        if not Pus[ST[9]] then
          begin
            Pus[ST[9]]:=true;
            Valid:=true
          end
        else
          begin
            Pus[ST[5]]:=false;
            Valid:=false
          end
      end;
      6:begin
        Pus[ST[6]]:=true;
      end;
    end;
end;

```

```

ST[10]:=26-ST[4]-ST[6]-ST[2];
if (ST[10]>0) and (not Pus[ST[10]]) then
  begin
    Pus[ST[10]]:=true;
    Valid:=true
  end
else
  begin
    Pus[ST[6]]:=false;
    Valid:=false
  end;
end;
7:begin
  Pus[ST[7]]:=true;
  ST[11]:=26-ST[5]-ST[7]-ST[3];
  ST[12]:=26-ST[6]-ST[7]-ST[8];
  if (ST[11]>0) and (ST[12]>0) and (ST[11]<>ST[12]) and
    (not Pus[ST[11]]) and (not Pus[ST[12]]) and
    (ST[9]+ST[10]+ST[11]+ST[12]=26) then
    begin
      Pus[ST[7]]:=false;
      Valid:=true
    end
  else
    begin
      Pus[ST[7]]:=false;
      Valid:=false
    end;
  end;
procedure Printsol;
begin
  write('1 ');
  for i:=2 to 12 do write(ST[i], ' ');
  writeln
end;

begin
  k:=2; ST[k]:=0;
  Pus[1]:=true;
  for i:=2 to 26 do Pus[i]:=false;
  while k>1 do
    begin
      repeat
        ST[k]:=ST[k]+1
      until (ST[k]>12) or Valid;
      if ST[k]<=12 then

```

```

if k=7 then Printsol
else
begin
  k:=k+1;
  ST[k]:=0
end
else
begin
  k:=k-1;
  case k of
    2:Pus[ST[2]]:=false;
    3:begin
      Pus[ST[3]]:=false;
      Pus[ST[8]]:=false
    end;
    4:Pus[ST[4]]:=false;
    5:begin
      Pus[ST[5]]:=false;
      Pus[ST[9]]:=false
    end;
    6:begin
      Pus[ST[6]]:=false;
      Pus[ST[10]]:=false
    end
  end
end
end.

```

Problema 9

Enunț. Fie $A_{m,n}$ o matrice binară. Se cunosc coordonatele (i, j) ale unui element, i aparținând $1 \dots m$, j aparținând $1 \dots n$, care are valoarea 1. Să se găsească toate ieșirile din matrice mergând numai pe elementele cu valoarea 1.

Rezolvare. Vom proceda ca la problema 4: reținem un vector cu deplasările relative ale mișcărilor posibile; se subînțelege că dintr-o căsuță se poate merge doar în cele patru vecine cu ea:

$(i-1, j) \quad (i+1, j) \quad (i, j+1) \quad (i, j-1)$

Condiția pentru o soluție o reprezintă verificarea atingerii marginilor matricei; deci avem patru cazuri, pentru fiecare din cele patru margini: $i=1$ sau $i=M$ sau $j=1$ sau $j=N$.

O mutare este considerată validă dacă:

este în matrice;

poziția respectivă este marcată cu 1.

Pozitările deja parcurse sunt marcate cu 2 pentru a nu fi considerate încă o dată în soluție.

Program Matrice;

const

```

Muta : array[0..4,1..2] of integer=
  (0,0), (-1,0), (1,0), (0,1), (0,-1));

```

var

```

N,M,k,i,j,oi,oj :integer;
A :array[0..100,0..100] of integer;
ST :array[0..1000] of integer;

```

Procedure Sol;

var x,y,f :integer;
begin

x:=oi; y:=oj;

for f:=1 to k do

begin

write('(',x,',',y,')');

x:=x+Muta[ST[f],1];

y:=y+Muta[ST[f],2];

end;

writeln;

dec(k)

end;

Function Valid:boolean;

var pi,pj :integer;

begin

pi:=i+Muta[ST[k],1];

pj:=j+Muta[ST[k],2];

Valid:=(pi>0) and (pj>0) and (pi<=M)
and (pj<=N) and (A[pi,pj]=1)

end;

begin

write('M,N:');

readln(M,N);

for i:=1 to M do

for j:=1 to N do

begin

write('A[',i,',',j,']=');

readln(A[i,j]);

```

    end;
    write('i,j:');
    readln(i,j);
    oj:=j;
    oi:=i;
    A[i,j]:=-1;
    fillchar(ST,sizeof(ST),0);
    k:=1;
    while k>0 do
    begin
        if (i=1) or (j=1) or (i=M) or (j=N) then Sol;
        if A[i,j]=2 then A[i,j]:=1;
        i:=i-Muta[ST[k],1];
        j:=j-Muta[ST[k],2];
        repeat
            inc(ST[k])
        until (ST[k]>4) or valid;
        if ST[k]=5 then
        begin
            ST[k]:=0;
            dec(k)
        end else
        begin
            A[i,j]:=2;
            i:=i+Muta[ST[k],1];
            j:=j+Muta[ST[k],2];
            inc(k)
        end
    end
end.

```

Problema 10

Enunț. Se dă un număr natural par N. Să se afișeze toate sirurile de N paranteze care se închid corect.

Rezolvare. Un sir de N paranteze care se închid corect, este un sir în care fiecare paranteză deschisă îi corespunde o paranteză închisă la o poziție ulterioară în sir.

Exemple de siruri corecte: ()()(); ((())) ; ()()()

Exemple de siruri incorrecte: ()()(); ((())) ;)()()

Deducem din propoziția de mai sus o primă condiție necesară pentru ca un sir de paranteze să se închidă corect și anume că nu trebuie să deschidem mai mult de $N/2$ paranteze. După cum se vede însă din aceste exemple (incorrecte), această condiție nu este suficientă: ((())(;))((();)()()).

A doua condiție este să nu închidem mai multe paranteze decât am deschis.

Pentru formalizarea condițiilor notăm cu $N_d(k)$ și $N_i(k)$ numărul de paranteze deschise, respectiv închise, până la poziția k a sirului, inclusiv. Cele două condiții sunt:

1. $N_d(k) \leq N/2, \forall 1 \leq k \leq n$
2. $N_d(k) \geq N_i(k), \forall 1 \leq k \leq n$

Pentru implementare folosim o stivă cu N nivele:

st[i] = 1, dacă paranteza de pe poziția i este deschisă
 2, dacă paranteza de pe poziția i este închisă.

În variabilele N_d și N_i avem numărul de paranteze de fiecare fel așezate în sir până la poziția curentă, k.

Pentru a așeza o paranteză deschisă pe poziția curentă trebuie să fi deschis mai puțin de $N/2$ paranteze, adică $N_d < N/2$.

Pentru a așeza o paranteză închisă pe poziția curentă trebuie să mai avem o paranteză pe care să o închidem, adică $N_d > N_i$.

La urcarea și coborârea în stivă se modifică N_d sau N_i în funcție de tipul parantezei de pe nivelul respectiv,

program Paranteze;

```

var ST :array [0..100] of integer;
    k, N, Nd, Ni :integer;

procedure Printsol;
var i:integer;
begin
    for i:=1 to N do
        if ST[i]=1 then write('(')
                    else write(')');
    writeln;
end;

function valid:boolean;
begin
    if ST[k]=1 then
        if Nd<N div 2 then valid:=true
                           else valid:=false
        else
            if Nd>Ni then valid:=true

```

```

        else valid:=false
end;

begin
  write('N= ');
  readln(N);
  Nd:=0;
  Ni:=0;
  k:=1;
  ST[k]:=0;
  while k>0 do
    begin
      repeat
        ST[k]:=ST[k]+1
      until (ST[k]>2) or valid;
      if ST[k]<=2 then
        if k=N then Printsol
        else
          begin
            if ST[k]=1 then Nd:=Nd+1
            else Ni:=Ni+1;
            k:=k+1;
            ST[k]:=0
          end;
      else
        begin
          k:=k-1;
          if ST[k]=1 then Nd:=Nd-1
          else Ni:=Ni-1
        end;
    end;
end.

```

Problema 11

Enunț. Se consideră o mulțime de N elemente și un număr natural K nenul. Să se calculeze câte submulțimi cu K elemente satisfac pe rând condițiile de mai jos și să se afișeze aceste submulțimi:

- conțin p obiecte date;
- nă conțin nici unul din q obiecte date
- conțin exact un obiect dat, dar nu conțin un altul
- conțin exact un obiect din p obiecte date
- conțin cel puțin un obiect din p obiecte date
- conțin r obiecte din p obiecte date, dar nu conțin alte q obiecte date.

Rezolvare. Pentru calcularea și generarea soluțiilor se pot folosi doar combinațiile. Pentru simplitatea programului se poate considera, fără a restrângere

generalitatea, că cele p , respectiv q obiecte date au numerele de ordine $1, 2, 3, \dots, p$, respectiv $p+1, p+2, \dots, p+q$ (în cazul c) avem $p=q=1$).

Fiecare nivel al stivei îi este asociată o mulțime de elemente care pot ocupa acel loc în soluție. Pentru a generaliza, vom considera că fiecare mulțime este de forma $\{L_1, L_{i+1}, L_{i+2}, \dots, L_s\}$. Anume: pentru fiecare nivel i al stivei vom reține doi vectori $L_i[i]$ și $L_s[i]$ care indică limita inferioară, respectiv superioară a elementului ce poate fi ales pe nivelul i (deci pentru $L_i[i]=2$ și $L_s[i]=5$, pe nivelul i vom putea avea doar elementele 2, 3, 4, 5).

Acum să vedem cu ce valori vom inițializa vectorii L_i și L_s pentru fiecare caz:

- a) practic trebuie să alegem întotdeauna elementele $1, 2, 3, \dots, p$ și apoi încă oricare $k-p$ din cele rămase ($p+1, p+2, \dots, N$). Pentru a forța alegerea primelor p elemente vom considera $L_i[i]=L_s[i]=i$.

Deci vectorii vor arăta astfel:

I	1	2	3	...	p	p+1	p+2	...	N
Li	1	2	3	...	p	p+1	p+2	...	p+1
Ls	1	2	3	...	p	N	N	...	N

- b) aici considerăm $p=0$: deci obiectele care nu trebuie selectate vor fi $1, 2, 3, \dots, q$. Avem $L_i[i]=q+1$ și $L_s[i]=N$ pentru oricare i .

- c) $p=1$ și $q=1$
 $L_i[1]=1$ și $L_s[1]=1$ obiectul 1 trebuie ales mereu;
 $L_i[i]=3$ și $L_s[i]=N$, pentru $i > 1$ cel cu numărul 2 trebuie evitat.

- d) $L_i[1]=1$ și $L_s[1]=p$ alegem un obiect între 1 și p ;
 $L_i[i]=p+1$ și $L_s[i]=N$, pentru $i > 1$ și restul între $p+1$ și N .

- e) $L_i[1]=1$ și $L_s[1]=p$ un obiect între 1 și p ;
 $L_i[i]=1$ și $L_s[i]=N$, pentru $i > 1$ restul pot lua orice valori.

- f) $L_i[i]=1$ și $L_s[i]=p$, pentru $1 \leq i \leq R$ pe primele r poziții alegem obiecte din primele p ;
 $L_i[i]=p+q+1$ și $L_s[i]=N$, pentru $R < i \leq K$ restul trebuie să nu fie printre cele p sau q .

Deci vom scrie o singură rutină backtracking, care, în funcție de inițializările vectorilor L_i și L_s , va furniza răspunsul la oricare din cele șase cerințe. Programul va mai folosi și vectorul u care va indica dacă elementul i a fost sau nu utilizat în soluție până la pasul curent. Pentru că variabila k face parte din datele de intrare, nivelul curent în stivă va fi memorat în variabila i .

```

program submultimi;
var
  ST, Li, Ls, u :array[0..1000] of integer;
  N, P, Q, R, K, i, f :integer;
  sol :longint;
  ch :char;
begin
  write('N, K, P, Q:');
  readln(N, K, P, Q);
  write('subpunctul (A, B, C, D, E sau F):');
  readln(ch);
  Case UpCase(ch) of
    'A':begin
      for i:= 1 to P do
        begin Li[i]:=i; Ls[i]:=i end;
      for i:=P+1 to K do
        begin Li[i]:=P+1; Ls[i]:=N end
    end;
    'B': for i:= 1 to K do
      begin Li[i]:=Q+1; Ls[i]:=N end;
    'C':begin
      for i:= 2 to K do
        begin Li[i]:=3; Ls[i]:=N end;
      Li[1]:=1; Ls[1]:=1
    end;
    'D':begin
      for i:= 2 to K do
        begin Li[i]:=P+1; Ls[i]:=N end;
      Li[1]:=1; Ls[1]:=P
    end;
    'E':begin
      for i:= 2 to K do
        begin Li[i]:=1; Ls[i]:=N end;
      Li[1]:=1; Ls[1]:=P
    end;
    'F':begin
      write('R=');
      readln(R);
      for i:= 1 to R do
        begin Li[i]:=i; Ls[i]:=P end;
      for i:=R+1 to K do
        begin Li[i]:=P+Q+1; Ls[i]:=N end
    end;
  end;
  i:=1;
  fillchar(ST,sizeof(ST),0);
  fillchar(u,sizeof(u),0);
  ST[1]:=Li[1]-1;

```

```

  sol:=0;
  while i>0 do
    if i=k+1 then
      begin
        sol:=sol+1;
        for f:=1 to K do write(ST[f], ' ');
        writeln;
        i:=i-1
      end else
      begin
        U[ST[i]]:=0;
        repeat
          ST[i]:=ST[i]+1
        until (ST[i]>LS[i]) or (U[ST[i]]=0);
        if ST[i]>LS[i] then
          begin
            ST[i]:=0;
            i:=i-1
          end else
          begin
            U[ST[i]]:=1;
            i:=i+1;
            if ST[i-1]<Li[i] then ST[i]:=Li[i]-1
            else ST[i]:=ST[i-1]-1
          end
        end;
      end;
  writeln('Numar solutii= ',sol)
end.

```

Problema 12

Enunț. Să se genereze toate permutările de N cu proprietatea că oricare ar fi $2 \leq i \leq N$, există $1 \leq j \leq i$ astfel încât $|V(i) - V(j)| = 1$.

Exemplu: pentru $N=4$, permutările cu proprietatea de mai sus sunt:
2134, 2314, 3214, 3241, 3421, 4321, 1234

Rezolvare. Pentru generarea permutărilor folosim o stivă cu N nivele și un vector auxiliar:

Pus[i]=TRUE, dacă numărul i a fost așezat deja în permutare;
FALSE, altfel.

Condițiile care trebuie îndeplinite pentru așezarea unui număr c pe nivelul k al stivei sunt:

- Pus[c]=FALSE, adică să nu fi așezat deja numărul c în permutare;

- cel puțin unul din numerele așezate pe pozițiile $1..k-1$, respectiv $st[1]..st[k-1]$, să aibă diferență absolută față de c egală cu 1.

```

program Permutari;

var ST           :array [1..20] of integer;
    k, i, N     :integer;
    Pus         :array [1..20] of boolean;

function Valid:boolean;
var tmp:boolean;
begin
  if k=1 then Valid:=true
  else
  begin
    tmp:=false;
    if not Pus[ST[k]] then
      for i:=1 to k-1 do
        if abs(ST[k]-ST[i])=1 then tmp:=true;
    Valid:=tmp
  end
end;

procedure Printsol;
var i:integer;
begin
  for i:=1 to N do write(st[i], ' ');
  writeln
end;

begin
  write('N= ');
  readln(N);
  for i:=1 to N do Pus[i]:=false;
  k:=1;
  ST[k]:=0;
  while k>0 do
    begin
      repeat
        ST[k]:=ST[k]+1
      until (ST[k]>N) or Valid;
      if ST[k]<=N then
        if k=N then Printsol
        else
          begin
            Pus[ST(k)]:=true;
            k:=k+1;
            ST[k]:=0
          end
    end
end.

```

```

  end
else
begin
  k:=k-1;
  Pus[ST[k]]:=false
end
end.

```

Problema 13

Enunț. Se dau N puncte albe și N puncte negre în plan, de coordonate întregi. Fiecare punct alb se unește cu câte un punct negru, astfel încât din fiecare punct, fie el alb sau negru, pleacă exact un segment. Să se determine o astfel de configurație de segmente încât oricare două segmente să nu se intersecteze. Se citeșc $2N$ perechi de coordonate corespunzând punctelor.

Rezolvare. Vom reține în vectorii X și Y coordonatele punctelor, așezând pe primele N poziții punctele albe și pe pozițiile $N+1, \dots, 2N$ punctele negre.

Soluția problemei este de fapt o permutare. Aceasta va fi generată în vectorul ST : $ST[i]$ reprezintă punctul negru cu care va fi conectat punctul alb i .

Funcția Intersect întoarce o valoare booleană care indică dacă segmentul determinat de punctul alb i și punctul negru $ST[i]$ se intersectează cu cel determinat de j și respectiv $ST[j]$.

Condiția de validitate pentru un segment este, evident, ca acesta să nu se intersecteze cu nici unul din cele construite anterior.

```

program Puncte;

var N,i,k           :integer;
    X,Y             :array[1..1000] of real;
    ST,u             :array[0..1000] of integer;

function Intersect(i,j:integer):boolean;
var a1,a2,b1,b2:real;
begin
  if x[i]=x[st[i]] then a1:=9e10
  else a1:=(y[st[i]]-y[i])/(x[st[i]]-x[i]);
  if x[j]=x[st[j]] then a2:=9e10
  else a2:=(y[st[j]]-y[j])/(x[st[j]]-x[j]);
  b1:=y[i]-a1*x[i];
  b2:=y[j]-a2*x[j];
  if (b1>a2) and (b1<a1) and (b2>a2) and (b2<a1) then
    Intersect:=true
  else
    Intersect:=false
end;

```

```

Intersect:=
  ((x[i]*a2+b2-y[i])*(x[st[i]]*a2+b2-y[st[i]])<=0) and
  ((x[j]*a1+b1-y[j])*(x[st[j]]*a1+b1-y[st[j]])<=0)
end;

function Valid:boolean;
var i:integer;
  r:boolean;
begin
  r:=(U[ST[k]]=0);
  i:=1;
  while r and (i<k) do
  begin
    r:=r and (not Intersect(k,i));
    i:=i+1
  end;
  valid:=r
end;

procedure Printsol;
var i:integer;
begin
  for i:=1 to N do write(ST[i], ' ');
  writeln;
  k:=0;
end;

begin
  write('N=');
  readln(N);
  writeln('punctele albe:');
  for i:=1 to N do
  begin
    write(i:3, ':X, Y=');
    readln(X[i], Y[i]);
  end;
  writeln('punctele negre:');
  for i:=N+1 to 2*N do
  begin
    write(i:3, ':X, Y=');
    readln(X[i], Y[i]);
  end;

  k:=1;
  fillchar(ST,sizeof(ST),0);
  fillchar(U,sizeof(U),0);
  ST[1]:=N;
  while k>0 do
  begin
    repeat

```

```

      ST[k]:=ST[k]+1
      until (ST[k]>2*N) or valid;
      if ST[k]<=2*N then
        if k=N then Printsol
        else ...
        begin
          U[ST[k]]:=1;
          k:=k+1;
          ST[k]:=N
        end
      else
        begin
          k:=k-1;
          U[ST[k]]:=0
        end
      end
    end.

```

Problema 14

Enunț: Se consideră n puncte în plan, de coordonate reale, (X_1, Y_1) , (X_2, Y_2) , (X_3, Y_3) , ..., (X_n, Y_n) . Elaborează un program care selectează din aceste puncte vârfurile unui pătrat, ce conține numărul maxim de puncte din cele date. Afisează coordonatele punctelor selectate și numărul de puncte incluse în pătratul respectiv.

Notă: Punctele care aparțin laturilor păratului se consideră incluse în pătrat (inclusiv colțurile păratului).

Datele de intrare se vor cîti din fișierul `input.txt`, astfel:

- pe prima linie n , numărul de puncte
- pe următoarele n linii coordonatele punctelor.

Rezolvare. Coordonatele punctelor se vor cîti în vectorul `Pct`.

Stîvă are 4 nivele reprezentând îndicii celor 4 puncte care formează pătratul. Dacă se găsește un astfel de pătrat, se numără câte puncte sunt incluse în acest pătrat. Se păstrează cea mai bună soluție în vectorul `MaxP`.

Condiția pentru că 3 puncte să poată face parte dintr-un pătrat este să formeze un triunghi dreptunghic isoscel. Acest lucru se verifică prin relația între distanțele dintre puncte (d_{12}, d_{23}, d_{13}) . Două dintre acestea trebuie să fie egale și a treia egală cu una din primele două înmulțită cu $\sqrt{2}$.

Atenție! În geometria analitică aplicată, testul de egalitate a două variabile reale a și b nu trebuie să fie "dacă $a=b$ ", ci "dacă $\text{abs}(a-b)<1e-5$ ", adică dacă diferența absolută dintre a și b este mai mică decât 10^{-5} . Acest test evită erorile de

aproximație (în programul nostru acestea pot apărea, de exemplu, la operația $\sqrt{2}$). Din acest motiv, am folosit funcția booleană Egal care face testul de egalitate prezentat mai sus.

Dacă cele 3 puncte formează un triunghi dreptunghic isoscel, se păstrează în variabila colt indicele vârfului triunghiului (unghiul drept) și în vectorul P (pe primele 3 poziții) indicii celor 3 puncte în ordinea în care apar în pătrat. Astfel, P[2] este vârful triunghiului dreptunghic isoscel și P[1], P[3], celelalte două vârfuri.

Pe nivelul 4, condiția de validitate este ca punctul așezat să aibă distanțele la P[1] și P[3], egale cu distanțele de la P[2] la aceste două puncte.

Funcția Dist(p1, p2) întoarce distanța între Pct[p1] și Pct[p2].

Funcția booleană Inclus(i) întoarce TRUE dacă punctul Pct[i] este inclus în pătratul P[1]P[2]P[3]P[4] și FALSE, altfel. Pentru a fi inclus în pătrat, punctul Pct[i] trebuie să se afle între dreptele suport ale segmentelor P[1]P[2] și P[4]P[3], și de asemenea între dreptele suport ale lui P[2]P[3] și P[1]P[4]. Pentru a se afla între două drepte paralele, de același sens, un punct trebuie să se afle în semiplane de semne opuse față de acele drepte.

```
program puncte;
type punct=record
    x,y:real
end;
var n, k, colt, max, i :integer;
    Pct :array [1..20] of punct;
    ST :array [1..4] of integer;
    P,maxp :array [1..4] of punct;
procedure Citire;
var i :integer;
    f, :text;
begin
    assign(f,'input.txt');
    reset(f);
    readln(f, n);
    for i:=1 to n do readln(f, Pct[i].x, Pct[i].y);
    close(f)
end;

function Egal(val1,val2:real):boolean;
begin
    Egal:=(abs(val1-val2)<=1e-5)
```

```
end;

function Dist(p1,p2:integer):real;
begin
    Dist:=sqrt(sqr(pct[p1].x-pct[p2].x)+sqr(pct[p1].y-pct[p2].y))
end;

function Valid:boolean;
var d12,d23,d13,d1,d2,d3:real;
begin
    if k<=2 then valid:=true
    else if k=3 then
        begin
            d12:=Dist(ST[1],ST[2]);
            d23:=Dist(ST[2],ST[3]);
            d13:=Dist(ST[1],ST[3]);
            if Egal(d12,d23) then
                if Egal(d13,d12+sqrt(2)) then
                    begin
                        colt:=2;
                        P[1]:=Pct[ST[1]]; P[2]:=Pct[ST[2]];
                        P[3]:=Pct[ST[3]];
                        Valid:=true
                    end
                else Valid:=false
            else if Egal(d12,d13) then
                if Egal(d23,d12+sqrt(2)) then
                    begin
                        colt:=1;
                        P[1]:=Pct[ST[2]]; P[2]:=Pct[ST[1]];
                        P[3]:=Pct[ST[3]];
                        Valid:=true
                    end
                else Valid:=false
            else if Egal(d13,d23) and Egal(d12,d23*sqrt(2)) then
                begin
                    colt:=3;
                    P[1]:=Pct[ST[1]]; P[2]:=Pct[ST[3]];
                    P[3]:=Pct[ST[2]];
                    Valid:=true
                end
            else Valid:=false
        end
    else begin
        case colt of
            1:begin
                d1:=Dist(ST[2],ST[4]);
                d2:=Dist(ST[3],ST[4]);
                d3:=Dist(ST[1],ST[2]);
            end;
```

III. Metoda Branch and Bound

Metoda *Branch and Bound* (*Ramifică și mărginește*) este înrudită cu metoda *backtracking*, diferențele constau în ordinea de parcursere a arborelui spațiului stăriilor și în modul în care se elimină subarborii care nu pot duce la rezultat.

Pentru început vom stabili cadrul în jurul căruia se va desfășura explicația: avem o stare *initială* (să o numim s_0) și o stare *finală* (să o numim s_f). Într-o stare se poate trece într-o altă stare prin luarea unei *decizii*. Este posibil ca într-o stare să se poată trece mai multe decizii, ceea ce implică faptul că dintr-o stare se poate trece în mai multe alte stări. Problema care se pune este determinarea șirului de decizii (dacă există) care trebuie luate pentru a se trece din starea *initială* în starea *finală*.

Pentru a simplifica problema vom presupune din start că întotdeauna există soluție, în caz contrar aplicarea acestei metode va duce doar la epuizarea resurselor calculatorului.

Din cele amintite rezultă că trebuie să construim un graf ale căruia noduri vor corespunde stăriilor, iar muchiile (mai precis arcele care sunt orientate de la un nod s_{iota} spre un alt nod s_{fina} , indicând faptul că decizia respectivă a transformat starea s_{iota} în starea s_{fina}) reprezintă decizii. Dacă permitem ca în graf să existe noduri ce reprezintă aceeași stare, vom avea de a face cu un graf infinit (din punct de vedere al numărului nodurilor). Dacă, în schimb, vom introduce restricția că un nod să apară o singură dată, atunci graful se va transforma într-un arbore (într-un nod nu vor ajunge niciodată două arce, deoarece arborele construindu-se nod cu nod, nu vom lua niciodată o decizie care să ne ducă într-o stare deja existentă). Arboarele se generează până la prima apariție a nodului final, deoarece în majoritatea cazurilor cardinalul spațiului stăriilor depășește cu mult posibilitățile unui calculator. Să ne gândim doar la problema jocului căsuțelor *Perspicco* al cărui număr de stări este $16! = 20.922.789.888.000$.

Să analizăm puțin două dintre modalitățile principale de rezolvare a problemei, respectiv principalele modalități de construire (parcursere) ale arborelui stăriilor:

1. Parcurserea **DF** (**Depth-First**) este, așa cum spune și titlul o parcursere în adâncime a grafului (arborelui) stăriilor. Procedura **DF(stare)** se aplică astfel: se pleacă din starea *initială* s_0 și se apelează **DF(s_0)**. Vor rezulta nodurile fii ale nodului s_0 pentru care s-a apelat procedura. Pentru fiecare nod rezultat (și care nu a fost încă vizitat) se apelează din nou procedura **DF**. Ne oprim în momentul în care vizităm nodul care conține starea *finală*. Deza-

vantajul principal al acestei metode este faptul că nu va furniza soluția întotdeauna cu număr minim de decizii. Acest lucru este remediat de către parcurgerea **BF**.

2. Parcurgerea **BF** (**Breadth-First**) este o parcursere în lățime a arborelui (grafului) stăriilor. Pentru acest algoritm construim un șir de mulțimi de stări. În prima mulțime vom introduce doar starea *initială*. În a doua mulțime vom introduce stările fii ale stării *initială*. Apoi, la fiecare pas în mulțimea k vom introduce stările fii ale stăriilor din mulțimea $k - 1$. Ne vom opri în momentul în care s-a generat mulțimea ce conține starea *finală*. Prin această metodă vom avea întotdeauna soluția în număr minim de decizii, în schimb, trebuie parcursă relativ multe stări pentru a ajunge la rezultatul final.

Dé fapt, ambele metode (DF și BF) au marele dezavantaj de a parcurge multe stări care nu sunt necesare. Putem afirma, fără temă de a greși, că spațiul stăriilor este parcurs *orbește*, fără a se ține cont de starea în care trebuie să ajungem. Scopul principal al acestei metode de programare este tocmai acesta, de a ne îndrepta căt mai mult către starea *finală*. În acest fel, multe dintre stările care ne îndepărtează de scopul final vor fi eliminate (sau cel puțin vor fi luate în considerare mai târziu).

Aceste considerente conduc la ideea introducerii unei *funcții de cost a unei stări*. Această funcție trebuie să respecte următoarele două condiții:

- *costul stării finale trebuie să fie minim* (în raport cu celelalte stări),
- *costurile stăriilor*, care se află pe un drum minim (de lungime minimă) de la starea *initială* la starea *finală*, trebuie să fie *descrescătoare*.

Este evident că, în mod ideal, la fiecare pas ar trebui să alegem decizia care să ne ducă într-o stare având cost mai mic. Însă, în cazul real, apar câteva dificultăți. Considerăm că această funcție nu este optimă (nu duce direct la soluție) datorită următoarelor două cauze principale:

- pentru o valoare dată a costului, există mai multe stări care au aceeași valoare,
- dintr-o stare nu putem trece întotdeauna în alta, luând în considerare numai costul (dacă ar fi fost posibil, am fi ales metoda *greedy*).

Există mai multe modalități pentru a calcula costul unei configurații. Această funcție de cost se definește astfel:

$$\text{cost}(s_{final}) \leftarrow 0;$$

$$\text{cost}(s) \leftarrow 1 + \min \{ \text{cost}(v) \mid v \text{ este deja calculat și există o decizie care trece } s \text{ în } v \}.$$

Dar, din păcate, această funcție nu o putem cunoaște decât după ce am construit deja soluția. De aceea vom lucra cu aproximări euristică ale costului. Denumirea de "euristică" vine de la faptul că vor fi explorate și stări care nu conduc la soluția optimă (sau care nu duc la nici un fel de soluție).

P11. Amplasează melodii

Fie n melodii care trebuie înregistrate pe o casetă audio (cu două fețe: față A și față B) astfel încât diferența (în valoare absolută) dintre lungimea feței A și lungimea feței B să fie minimă.

Restricții

Datele se citesc din fișierul MELODII.IN care are următoarea structură:

- pe prima linie este scris numărul de melodii;
- pe a doua linie sunt scrise lungimile melodiilor.

Soluție

Problema se reduce la a determina o submulțime de melodii a căror lungime este cât mai aproape de jumătatea sumei lungimilor tuturor melodiilor. Această problemă este cunoscută în literatura de specialitate ca fiind NP-completă. Ea admite totuși o soluție de complexitate pseudo-polinomială, adică mărginită de lungimea intrării și dimensiunea maximă a datelor de intrare. Dacă datele sunt mari, atunci se preferă o soluție *backtracking*.

Vom copia recursiv fiecare melodie pe *FataA*, apoi pe *FataB* până când vom epuiza toate posibilitățile de așezare (care sunt în număr de 2^n). Soluția optimă se reține în doi vectori *FA* și *FB*, având lungimile *mA* respectiv *mB*.

```
Program AmplaseazaMelodii;
const MaxN=100;
type fata=array[1..MaxN] of Word;
var FataA,FataB,FA,FB,l:fata;
    mA,mB,n,nrA,nrB,i,min:Word;
procedure citeste;
var f:Text;
begin
  Assign(f,'MELODII.IN'); Reset(f);
  Readln(f,n);
  for i:=1 to n do
    if not SeekEoln(f) then Read(f,l[i]);
  Close(f)
end;
function CalculeazaDiferenta:Word;
var d,c:Integer;
begin
  d:=0;
  for c:=1 to nrA do d:=d+FataA[c];
  for c:=1 to nrB do d:=d-FataB[c];
  CalculeazaDiferenta:=Abs(d)
end;
```

```
procedure Amplaseaza(g:Word);
var v,dif:Word;
begin
  if g=n+1 then
  begin
    dif:=CalculeazaDiferenta;
    if dif<min then
      begin
        FA:=FataA; mA:=nrA; FB:=FataB; mB:=nrB; min:=dif
      end
    end
  else
    begin
      Inc(nrA); FataA[nrA]:=l[g];
      dif:=CalculeazaDiferenta;
      if dif<min then Amplaseaza(g+1);
      Dec(nrA); Inc(nrB); FataB[nrB]:=l[g];
      dif:=CalculeazaDiferenta;
      if dif<min then Amplaseaza(g+1);
      Dec(nrB)
    end
  end;
procedure AfiseazaSolutie;
begin
  Write('Fata A:');
  for i:=1 to mA do Write(FA[i], ' ');
  Writeln;
  Write('Fata B:');
  for i:=1 to mB do Write(FB[i], ' ');
  Writeln
end;
Begin
  citeste;
  nrA:=0;
  nrB:=0;
  min:=65535;
  Amplaseaza(1);
  AfiseazaSolutie;
End.
```

Intrare	Iesire
4 1 3 5 3	Fata A:1 5 Fata B:3 3

Mai întâi introducem noțiunea de *distanță* dintre două stări care, dacă este aleasă corespunzător, poate duce la scăderea drastică a numărului de configurații vizitate. Această funcție, de asemenea, trebuie să îndeplinească câteva condiții:

Distanța(*A*, *A*) $\leftarrow 0$,

Distanța(*A*, *s_{final}*) \geq *Distanța*(*B*, *s_{final}*), dacă există o decizie care trece *A* în *B*.

Funcția *distanță* nu poate fi generalizată pentru orice problemă, adică nu există o formulă adecvată pentru toate situațiile. De alegerea ei depinde numărul de configurații pe care le vom vizita până ajungem la configurația finală. Se poate propune o alegere independentă de tipul unei stări și anume consideră reprezentarea în memorie a unei stări ca fiind o succesiune de octeți (lucru real în practică) și calculează numărul de poziții prin care către două zone de memorie diferă. Există cazuri când această modalitate este destul de bună, de exemplu în cazul jocului *Perspicco*, în care o stare este reprezentată sub forma unei matrice de dimensiuni 4x4 (cu elemente în mulțimea {0, ..., 15}). Dacă, în schimb avem stările sub formă unor multimi, compararea ar trebui să se facă la nivel de bit și nu de octet. Rezultă că în acest caz folosirea distanței ca mai sus, nu ar conduce la o aproximare mulțumitoare.

Revenind la cost, dintre euristicile mai des întâlnite vom amânta două:

1. *cost* \leftarrow nivelul curent + distanța dintre configurația curentă și configurația finală;
2. *cost* \leftarrow costul stării părinte + distanța dintre configurația curentă și configurația finală (prin nivel se înțelege numărul mutărilor prin care s-a ajuns la configurația curentă).

Cea de a doua modalitate de cost duce la o parcurgere în lățime a arborelui stărilor. În unele situații limită, în care implementarea unei funcții de distanță este dificilă, se poate cere ca o configurație să aibă un cost fix. Aici prin *fix* se înțelege: independent de alte stări, dar dependent de anumite variabile legate efectiv de stare.

Trecem la descrierea metodei și a principaliilor ei pași. Pentru simplificare, vom lucea cu varianta standard a problemei, respectiv: *se dă o stare inițială, o stare finală și se cer decizile*.

Structura unui element al listei cu stările are câmpurile:

inf – în care se reține o stare,

cost – în care se reține costul unei stări,

urm, *pred* – legăturile spre următoarea stare din listă, respectiv spre starea părinte.

Creează_capul_listei; { initializază câmpul *inf* cù starea inițială și *costul* }
{ conform formulei definite va rezulta variabila *cap* }

while *cap*<>StareaFinală do

begin

Expandează(*cap*) { aplică lui *cap* toate deciziile posibile pentru fiecare }
{ stare rezultată din expandare îi calculează costul și o adaugă unde îi este locul }
end;

Metoda branch and bound

PROBLEME

P1. Jocul Perspicco

Se consideră o matrice având dimensiunea 4×4 ale cărei celule conțin numerele de la 1 la 15, cu excepția uneia dintre ele care este liberă (conține cifra 0). Trebuie obținută o configurație finală dintr-o configurație inițială a matricei date, (ambele de tipul descris mai sus) folosind următoarea regulă de transformare: orice număr poate fi deplasat orizontal sau vertical într-o celulă liberă alăturată, celula care a conținut acest număr devinând liberă. Se cere să se determine numărul minim de mutări care transformă configurația dată în cea finală.

Discuție

O stare a jocului este caracterizată printr-o matrice de dimensiuni 4×4 . Un element al listei este deci alcătuit astfel:

```
type lista=^articol;
  articol=record
    inf:array[1..4,1..4] of 0..15;
    cost:Word;
    urm,pred:lista
  end;
```

având următoarea semnificație:

- *articol^.inf* reține o configurație;
- *articol^.cost* este costul configurației reținute în *articol^.inf*.

Vom amplasa aceste elemente (articole) într-o listă ordonată crescător după valoarea *articol^.cost*. Inițial, în listă vom avea doar articoul referitor la configurația inițială. La fiecare pas vom extrage din listă configurația cu costul cel mai mic (pentru a ne apropiă cât mai mult de configurația finală) și o vom expanda, adică vom aplica cele 4 mutări permise (dacă sunt posibile) obținând astfel 4 noi configurații pe care le vom insera în listă. Repetăm aceste operații până în momentul în care ajungem la configurația finală.

Procedura *AdaugaNod* vă insera în listă ordonată crescător, nodul transmis ca parametru.

Funcția *Costul* va returna costul unei configurații. În primul program costul va fi egal cu suma dintre costul părintelui și costul distanței (*Manhattan*) a stării curente față de cea finală.

Pentru a simplifica algoritmul, am presupus că avem următoarea configurație finală:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	0

Soluție

Reconstituirea soluției se realizează recursiv, prin procedura Reconstituie.

Program Branch_and_Bound;

```

type TConfiguratie=array[1..4,1..4] of 0..15;
  lista^:art;
  art=record
    cost:Word;
    inf:TConfiguratie;
    urm,pred:lista
  end;

var i,k,zl,zc:Byte;
  ConfIni:TConfiguratie;
  r,cap:lista;

procedure Citire;
  var f:Text;
begin
  Assign(f,'INPUT.IN');
  Reset(f);
  for i:=1 to 4 do
  begin
    for k:=1 to 4 do Read(f,ConfIni[i,k]);
    Readln(f)
  end;
  Close(f)
end;

procedure CautaZero(x:TConfiguratie; var zLinie,zColoana:Byte);
  var i,k:Byte;
begin
  { cauta casuta libera in matricea x }
  for i:=1 to 4 do
    for k:=1 to 4 do
      if x[i,k]=0
      then
        begin
          zLinie:=i; { se returneaza linia si }
          zColoana:=k; { coloana casutei libere }
        end
end;

```

```

function Distanța(conf:TConfiguratie):Byte;
  var cst,e,r:Byte;
begin
  { calculeaza distanța dintre }
  { starea curentă și cea finală }
  { cu ajutorul distantei Manhattan }

  cst:=0;
  for i:=1 to 4 do
    for k:=1 to 4 do
    begin
      e:=conf[i,k] div 4+1;
      r:=conf[i,k] mod 4;
      if r=0
      then
        begin r:=4; e:=e-1.end;
      if conf[i,k]=0
      then cst:=cst+Abs(i-4)+Abs(k-4)
      else cst:=cst+Abs(i-e)+Abs(k-r)
    end;
  Distanța:=cst
end;

function Costul(p:TConfiguratie; costParinte:Word):Word;
begin
  Costul:=Distanța(p)+costParinte
end;

procedure AdaugaNod(p:lista);
  var t,u:lista;
begin
  t:=cap^.urm;
  u:=cap;
  while (p^.cost>=t^.cost) and (t<>nil) do
  begin
    u:=t;
    t:=t^.urm
  end;
  p^.urm:=t;
  u^.urm:=p
end;

procedure AfiseazaMatrice(x:TConfiguratie);
  var v,b:Byte;
begin
  for v:=1 to 4 do
  begin
    for b:=1 to 4 do Write(x[v,b],' ');
    Writeln
  end;
  Writeln
end;

```

```

procedure Reconstituie(nod:lista);
begin
  if nod<>nil
  then
    begin
      Reconstituie(nod^.pred);
      AfiseazaMatrice(nod^.inf)
    end
  end;
end;

procedure AfisareSolutie(p:lista);
begin
  Writeln('ConfiguratieInitiala:');
  AfiseazaMatrice(ConfIni);
  Writeln('ConfiguratieFinala:');
  AfiseazaMatrice(cap^.inf);
  Writeln('Mutarile:');
  Reconstituie(cap)
end;

procedure AplicaMutare(var w: TConfiguratie; mut:Char);
begin
  case mut of
    's':begin w[zl,zc]:=w[zl,zc-1]; w[zl,zc-1]:=0 end;
    'd':begin w[zl,zc]:=w[zl,zc+1]; w[zl,zc+1]:=0 end;
    'u':begin w[zl,zc]:=w[zl-1,zc]; w[zl-1,zc]:=0 end;
    'j':begin w[zl,zc]:=w[zl+1,zc]; w[zl+1,zc]:=0 end
  end
end;
end;

Begin
  Citire;
  New(cap);
  cap^.pred:=nil;
  cap^.inf:=ConfIni;
  cap^.urm:=nil;
  cap^.cost:=Costul(cap^.inf, 0);
repeat
  { se schimba casuta libera cu cea din stanga }
  CautaZero(cap^.inf, zl, zc);
  if zc>1 { casuta libera nu se afla pe prima coloana }
  then
    begin
      New(r);
      r^.inf:=cap^.inf;
      AplicaMutare(r^.inf, 's');
      r^.cost:=Costul(r^.inf, cap^.cost);
      r^.pred:=cap;
      r^.urm:=nil;
      AdaugaNod(r)
    end;
  { se schimba casuta libera cu cea de jos }
end;

```

```

if zl<4 { casuta libera nu se afla pe ultima linie }
then
begin
  New(r);
  r^.inf:=cap^.inf;
  AplicaMutare(r^.inf, 'j');
  r^.cost:=Costul(r^.inf, cap^.cost);
  r^.pred:=cap;
  r^.urm:=nil;
  AdaugaNod(r)
end;
{ se schimba casuta libera cu cea de sus }
if zl>1 { casuta libera nu se afla pe prima linie }
then
begin
  New(r);
  r^.inf:=cap^.inf;
  AplicaMutare(r^.inf, 'u');
  r^.cost:=Costul(r^.inf, cap^.cost);
  r^.pred:=cap;
  r^.urm:=nil;
  AdaugaNod(r)
end;
{ se schimba casuta libera cu cea din dreapta }
if zc<4 { casuta libera nu se afla pe ultima coloana }
then
begin
  New(r);
  r^.inf:=cap^.inf;
  AplicaMutare(r^.inf, 'd');
  r^.cost:=Costul(r^.inf, cap^.cost);
  r^.pred:=cap;
  r^.urm:=nil;
  AdaugaNod(r)
end;
cap:=cap^.urm;
until cap^.cost=cap^.pred^.cost;
AfisareSolutie(cap);
End.

```

- Costul unei configurații poate fi calculat și ca sumă a distanței dintre configurația curentă și cea finală și nivelul (din arborele mutărilor) pe care se află nodul părinte. Se consideră că nodul inițial (cel care conține configurația inițială) se află pe nivelul zero. Avem nevoie de încă un câmp (*niv*) care conține nivelul din arborele mutărilor pe care se află nodul curent (starea curentă). În rest programul este asemănător cu precedentul. Observăm că lista nu va mai conține nodurile ordonate crescător, și de aceea testul de final se va face prin compararea cu zero a costului unei configurații.

```

Program Branch_and_Bound;
type TConfiguratie=array[1..4,1..4] of 0..15;
  lista^=art;
  art=record
    cost:Word;
    inf:TConfiguratie;
    niv:Byte;
    urm,pred:lista
  end;
var i,k,zl,zc:Byte;
  ConfIni:TConfiguratie;
  r,cap:lista;
procedure Citire;
  var f:Text;
begin
  Assign(f,'INPUT.IN');
  Reset(f);
  for i:=1 to 4 do
  begin
    for k:=1 to 4 do Read(f,ConfIni[i,k]);
    Readln(f)
  end;
  Close(f)
end;
procedure CautaZero(x:TConfiguratie; var zLinie,zColoana:Byte);
  var i,k:Byte;
begin
  for i:=1 to 4 do
  begin
    for k:=1 to 4 do
      if x[i,k]=0
      then
        begin
          zLinie:=i;
          zColoana:=k
        end
    end;
  end;
  function Distaanta(conf:TConfiguratie):Byte;
    var cst,e,r:Byte;
begin
  cst:=0;
  for i:=1 to 4 do
    for k:=1 to 4 do
      begin
        e:=conf[i,k] div 4+1; r:=conf[i,k] mod 4;
        if r=0
        then begin r:=4; e:=e-1 end;
        if conf[i,k]=0 then cst:=cst+Abs(i-4)+Abs(k-4)
        else cst:=cst+Abs(i-e)+Abs(k-r)
      end;
  end;
  function Costul(p:TConfiguratie; nivelParinte:Word):Word;
begin
  Costul:=Distaanta(p)+nivelParinte
end;
procedure AdaugaNod(p:lista);
  var t,u:lista;
begin
  t:=cap^.urm; u:=cap;
  while (p^.cost>=t^.cost) and (t<>nil) do
  begin
    u:=t; t:=t^.urm
  end;
  p^.urm:=t; u^.urm:=p
end;
procedure AfiseazaMatrice(x:TConfiguratie);
  var v,b:Byte;
begin
  for v:=1 to 4 do
  begin
    for b:=1 to 4 do Write(x[v,b],' ');
    Writeln
  end;
  Writeln
end;
procedure Reconstituie(nod:lista);
begin
  if nod<>nil
  then
    begin
      Reconstituie(nod^.pred);
      AfiseazaMatrice(nod^.inf)
    end
  end;
procedure AfisareSolutie(p:lista);
begin
  Writeln('ConfiguratieInitiala:');
  AfiseazaMatrice(ConfIni);
  Writeln('ConfiguratieFinala:');
  AfiseazaMatrice(cap^.inf);
  Writeln('Mutarile:');
  Reconstituie(cap)
end;

```

```

Begin
  Citire;
  New(cap);
  cap^.pred:=nil;
  cap^.inf:=ConfIni;
  cap^.urm:=nil;
  cap^.niv :=0;
  cap^.cost:=Costul(cap^.inf,0);
repeat      { se schimba casuta libera cu cea din stanga }
  CautaZero(cap^.inf,zl,zc);
  if zc>1    { casuta libera nu se afla pe prima coloana }
  then
  begin
    New(r);
    r^.inf:=cap^.inf;
    r^.inf[zl,zc]:=r^.inf[zl,zc-1];
    r^.inf[zl,zc-1]:=0;
    r^.cost:=Costul(r^.inf,cap^.niv);
    r^.pred:=cap;
    r^.urm:=nil;
    r^.niv:=cap^.niv+1;
    AdaugaNod(r)
  end;       { se schimba casuta libera cu cea de jos }
  if zl<4    { casuta libera nu se afla pe ultima linie }
  then
  begin
    New(r);
    r^.inf:=cap^.inf;
    r^.inf[zl,zc]:=r^.inf[zl+1,zc];
    r^.inf[zl+1,zc]:=0;
    r^.cost:=Costul(r^.inf,cap^.niv);
    r^.pred:=cap;
    r^.urm:=nil;
    r^.niv:=cap^.niv+1;
    AdaugaNod(r)
  end;       { se schimba casuta libera cu cea de sus }
  if zl>1    { casuta libera nu se afla pe prima linie }
  then
  begin
    New(r);
    r^.inf:=cap^.inf;
    r^.inf[zl,zc]:=r^.inf[zl-1,zc];
    r^.inf[zl-1,zc]:=0;
    r^.cost:=Costul(r^.inf,cap^.niv);
    r^.pred:=cap;
    r^.urm:=nil;
    r^.niv:=cap^.niv+1;
    AdaugaNod(r)
  end;       { se schimba casuta libera cu cea din dreapta }
end;

```

Metoda branch and bound

```

if zc<4      { casuta libera nu se afla pe ultima coloana }
then
begin
  New(r);
  r^.inf:=cap^.inf;
  r^.inf[zl,zc]:=r^.inf[zl,zc+1];
  r^.inf[zl,zc+1]:=0;
  r^.cost:=Costul(r^.inf,cap^.niv);
  r^.pred:=cap;
  r^.urm:=nil;
  r^.niv:=cap^.niv+1;
  AdaugaNod(r)
end;
  cap:=cap^.urm;
until Costul(cap^.inf,0)=0;
AfisareSolutie(cap)
End.

```

În cazul în care nu dorim să reținem efectiv starea curentă (un motiv ar fi lipsa spațiului), ci doar șirul mutărilor care a condus la ea, vom opera câteva modificări asupra programelor de mai sus. În primul rând informația dintr-un element al listei nu va fi alcătuită dintr-o matrice a stărilor, ci dintr-un șir (string) de mutări. Acestea sunt codificate după cum urmează:

- *s* schimbă conținutul căsuței libere cu conținutul căsuței aflate la stânga căsuței libere;
- *d* schimbă conținutul căsuței libere cu conținutul căsuței aflate la dreapta căsuței libere;
- *u* schimbă conținutul căsuței libere cu conținutul căsuței aflate deasupra căsuței libere;
- *j* schimbă conținutul căsuței liberă cu conținutul căsuței aflate sub căsuța liberă.

Program BranchAndBound;

```

type lista=^art;
art=record
  cost:Word;
  inf:string;
  urm,pred:lista
end;
TConfiguratie=array[1..4,1..4] of 0..15;
var i,k,zLinie,zColoana,zl,zc:Byte;
ConfIni,w:TConfiguratie;
r, cap:lista;
procedure Citire;
  var f:Text;

```

```

begin
  Assign(f,'INPUT.IN'); Reset(f);
  for i:=1 to 4 do
    begin
      for k:=1 to 4 do Read(f,ConfIni[i,k]);
      Readln(f)
    end;
  Close(f);
end;

procedure CautaZero;
begin
  for i:=1 to 4 do
    for k:=1 to 4 do
      if(ConfIni[i,k]=0)
      then
        begin zLinie:=i; zColoana:=k end
  end;

procedure AplicaMutare(var w: TConfiguratie; mut:Char);
begin
  case mut of
    's':begin zc:=zc-1; w[zl,zc+1]:=w[zl,zc]; w[zl,zc]:=0 end;
    'd':begin zc:=zc+1; w[zl,zc-1]:=w[zl,zc]; w[zl,zc]:=0 end;
    'u':begin zl:=zl-1; w[zl+1,zc]:=w[zl,zc]; w[zl,zc]:=0 end;
    'j':begin zl:=zl+1; w[zl-1,zc]:=w[zl,zc]; w[zl,zc]:=0 end
  end;
end;

function Distanță(conf:TConfiguratie):Byte;
var cst,e,r:Byte;
begin
  { calculează distanța dintre }
  { starea curentă și cea finală }
  { cu ajutorul distantei Manhattan }
  cst:=0;
  for i:=1 to 4 do
    for k:=1 to 4 do
      begin
        e:=conf[i,k] div 4+i; r:=conf[i,k] mod 4;
        if r=0
        then begin r:=4; e:=e-1 end;
        if conf[i,k]=0 then cst:=cst+Abs(i-4)+Abs(k-4)
        else cst:=cst+Abs(i-e)+Abs(k-r)
      end;
  Distanță:=cst
end;

function Costul(p:TConfiguratie; costParinte:Word):Word;
begin
  Costul:=Distanță(p)+costParinte
end;

```

```

procedure AdaugaNod(p:lista);
var t,u:lista;
begin
  t:=cap^.urm; u:=cap;
  while (p^.cost>=t^.cost) and (t<>nil) do
    begin
      u:=t;
      t:=t^.urm
    end;
  p^.urm:=t; u^.urm:=p
end;

procedure AfiseazaMatrice(x:TConfiguratie);
var v,b:Byte;
begin
  for v:=1 to 4 do
    begin
      for b:=1 to 4 do Write(x[v,b], ' ');
      Writeln
    end;
  Writeln
end;

procedure AfisareSolutie(sirMutari:string);
var w:TConfiguratie;
l,c:Word;
begin
  Writeln('Mutarile:');
  w:=ConfIni;
  zl:=zLinie;
  zc:=zColoana;
  AfiseazaMatrice(w);
  for i:=1 to Length(sirMutari). do
    begin
      AplicaMutare(w,sirMutari[i]);
      AfiseazaMatrice(w)
    end;
end;

Begin
  Citire;
  CautaZero;
  New(cap);
  cap^.pred:=nil;
  cap^.inf:='';
  cap^.urm:=nil;
  cap^.cost:=Costul(ConfIni,0);
repeat
  zl:=zLinie;
  zc:=zColoana; w:=ConfIni;

```

```

for i:=1 to Length(cap^.inf) do
  AplicaMutare(w, cap^.inf[i]);
  { se schimba casuta libera cu cea din stanga }
  if zc>1 then { casuta libera nu se afla pe prima coloana }
  then
    begin
      New(r); r^.inf:=cap^.inf+'s';
      w[zl,zc]:=w[zl,zc-1]; w[zl,zc-1]:=0;
      r^.cost:=Costul(w, cap^.cost); r^.pred:=cap; r^.urm:=nil;
      AdaugaNod(r);
      w[zl,zc-1]:=w[zl,zc]; w[zl,zc]:=0
    end;
    { se schimba casuta libera cu cea de jos }
    if zl<4 then { casuta libera nu se afla pe ultima linie }
    then
      begin
        New(r); r^.inf:=cap^.inf+'j';
        w[zl,zc]:=w[zl+1,zc]; w[zl+1,zc]:=0;
        r^.cost:=Costul(w, cap^.cost); r^.pred:=cap; r^.urm:=nil;
        AdaugaNod(r);
        w[zl+1,zc]:=w[zl,zc]; w[zl,zc]:=0
      end;
      { se schimba casuta libera cu cea de sus }
      if zl>1 then { casuta libera nu se afla pe prima linie }
      then
        begin
          New(r); r^.inf:=cap^.inf+'u';
          w[zl,zc]:=w[zl-1,zc];
          w[zl-1,zc]:=0;
          r^.cost:=Costul(w, cap^.cost);
          r^.pred:=cap;
          r^.urm:=nil;
          AdaugaNod(r);
          w[zl-1,zc]:=w[zl,zc];
          w[zl,zc]:=0
        end;
        { se schimba casuta libera cu cea din dreapta }
        if zc<4 then { casuta libera nu se afla pe ultima coloana }
        then
          begin
            New(r);
            r^.inf:=cap^.inf+'d';
            w[zl,zc]:=w[zl,zc+1]; w[zl,zc+1]:=0;
            r^.cost:=Costul(w, cap^.cost); r^.pred:=cap; r^.urm:=nil;
            AdaugaNod(r);
            w[zl,zc+1]:=w[zl,zc];
            w[zl,zc]:=0
          end;
          cap:=cap^.urm;
        until cap^.cost=cap^.pred^.cost;
        AfisareSolutie(cap^.inf);
      End.

```

Datorită faptului că memoria calculatorului are dimensiuni relativ mici, nu ne vom permite să examinăm oricără de multe configurații. Rezultă că trebuie să căutăm posibilități de optimizare. Numărul de octeți ocupati de prima structură definită anterior este de 22. Dacă în locul matricei am reține doar mutarea prin care se ajunge de la configurația anterioară la cea actuală, am putea economisi o cantitate apreciabilă de memorie. Structura va arăta în felul următor:

```

type lista=^articol;
  articol=record
    inf:Char;
    cost:Word;
    urm,pred:lista
  end;

```

iar numărul de octeți ocupati va fi 11.

Funcția `ReturneazaSirulMutarilor` va reconstituî mutările din care se obține configurația actuală, plecând de la cea inițială.

Program Branch_And_Bound;

```

type lista=^art;
  art=record
    cost:Word;
    inf:Char;
    urm,pred:lista
  end;
  TConfiguratie=array[1..4,1..4] of 0..15;
var i,k,zLinie,zColoana,zl,zc:Byte;
  ConfIni,w:TConfiguratie;
  r,cap:lista;
  sir:string;
procedure Citire;
  var f:Text;
begin
  Assign(f,'INPUT.IN'); Reset(f);
  for i:=1 to 4 do
    begin
      for k:=1 to 4 do Read(f,ConfIni[i,k]); Readln(f)
    end;
  Close(f)
end;
procedure CautaZero;
begin
  for i:=1 to 4 do
    for k:=1 to 4 do
      if ConfIni[i,k]=0 then begin zLinie:=i; zColoana:=k end
end;

```

```

function ReturneazaSirulMutarilor(q:lista):string;
  var s,g:string;
    t:lista;
begin
  s:='';
  t:=q;
  g:='';
  while t^.inf<>'' do
  begin
    g:=g+t^.inf;
    Insert(g,s,1);
    t:=t^.pred
  end;
  ReturneazaSirulMutarilor:=s
end;

procedure AplicaMutare(var w:TConfiguratie; mut:Char);
begin
  case mut of
    's':begin zc:=zc-1; w[zl,zc+1]:=w[zl,zc]; w[zl,zc]:=0 end;
    'd':begin zc:=zc+1; w[zl,zc-1]:=w[zl,zc]; w[zl,zc]:=0 end;
    'u':begin zl:=zl-1; w[zl+1,zc]:=w[zl,zc]; w[zl,zc]:=0 end;
    'j':begin zl:=zl+1; w[zl-1,zc]:=w[zl,zc]; w[zl,zc]:=0 end
  end
end;

function Distanță(conf:TConfiguratie):Byte;
  var cst,e,z:Byte;
begin
  { calculează distanța dintre starea curentă și }
  { cea finală cu ajutorul distantei Manhattan }
  cst:=0;
  for i:=1 to 4 do
    for k:=1 to 4 do
      begin
        e:=conf[i,k] div 4+1;
        r:=conf[i,k] mod 4;
        if r=0
        then
          begin r:=4; e:=e-1 end;
        if conf[i,k]=0
        then cst:=cst+Abs(i-4)+Abs(k-4)
        else cst:=cst+Abs(i-e)+Abs(k-r)
      end;
  Distanță:=cst
end;

function Costul(p:TConfiguratie; costParinte:Word):Word;
begin
  Costul:=Distanță(p)+costParinte
end;

```

```

procedure AdaugaNod(p:lista);
  var t,u:lista;
begin
  t:=cap^.urm; u:=cap;
  while (p^.cost>=t^.cost) and (t<>nil) do
  begin
    u:=t; t:=t^.urm
  end;
  p^.urm:=t; u^.urm:=p
end;

procedure AfiseazaMatrice(x:TConfiguratie);
  var v,b:Byte;
begin
  for v:=1 to 4 do
  begin
    for b:=1 to 4 do Write(x[v,b], ' ');
    Writeln
  end;
  Writeln
end;

procedure AfisareSolutie(nod:lista);
  var sirMutari:string;
begin
  Writeln('Mutările:');
  sirMutari:=ReturneazaSirulMutarilor(nod);
  w:=Confini; zl:=zLinie; zc:=zColoana;
  AfiseazaMatrice(w);
  for i:=1 to Length(sirMutari) do
  begin
    AplicaMutare(w,sirMutari[i]);
    AfiseazaMatrice(w)
  end;
end;

Begin
  Citire;
  CautaZero;
  New(cap);
  cap^.pred:=nil;
  cap^.inf:='-';
  cap^.urm:=nil;
  cap^.cost:=Costul(Confini,0);
repeat
  zl:=zLinie;
  zc:=zColoana;
  w:=Confini;
  sir:=ReturneazaSirulMutarilor(cap);
  for i:=1 to Length(sir) do AplicaMutare(w,sir[i]);

```

```

if zc>1 { se schimba casuta libera cu cea din stanga };
then
begin
  New(r);
  r^.inf:='s';
  w[zl,zc]:=w[zl,zc-1]; w[zl,zc-1]:=0;
  r^.cost:=Costul(w,cap^.cost);
  r^.pred:=cap; r^.urm:=nil;
  AdaugaNod(r);
  w[zl,zc-1]:=w[zl,zc]; w[zl,zc]:=0
end; { se schimba casuta libera cu cea de jos };
if zl<4 { casuta libera nu se afla pe ultima linie };
then
begin
  New(r);
  r^.inf:='j';
  w[zl,zc]:=w[zl+1,zc]; w[zl+1,zc]:=0;
  r^.cost:=Costul(w,cap^.cost);
  r^.pred:=cap; r^.urm:=nil;
  AdaugaNod(r);
  w[zl+1,zc]:=w[zl,zc]; w[zl,zc]:=0
end; { se schimba casuta libera cu cea de sus };
if zl>1 { casuta libera nu se afla pe prima linie };
then
begin
  New(r); r^.inf:='u';
  w[zl,zc]:=w[zl-1,zc]; w[zl-1,zc]:=0;
  r^.cost:=Costul(w,cap^.cost);
  r^.pred:=cap; r^.urm:=nil;
  AdaugaNod(r);
  w[zl-1,zc]:=w[zl,zc]; w[zl,zc]:=0
end; { se schimba casuta libera cu cea din dreapta };
if zc<4 { casuta libera nu se afla pe ultima coloana };
then
begin
  New(r); r^.inf:='d';
  w[zl,zc]:=w[zl,zc+1];
  w[zl,zc+1]:=0;
  r^.cost:=Costul(w,cap^.cost);
  r^.pred:=cap;
  r^.urm:=nil;
  AdaugaNod(r);
  w[zl,zc+1]:=w[zl,zc]; w[zl,zc]:=0
end;
  cap:=cap^.urm;
until cap^.cost=cap^.pred^.cost;
AfisareSolutie(cap);
End.

```

Intrare

Configuratia initiala:

2	0	3	4	2	0	3	4
1	6	7	8	1	6	7	8
5	10	11	12	5	10	11	12
9	13	14	15	9	13	14	15

Iesire

Configuratia finala:

1	2	3	4	1	2	3	4
5	6	7	8	5	6	7	8
0	10	11	12	0	10	11	12
9	13	14	15	9	13	14	15

Iesire (continuare)

stanga jos jos jos dreapta dreapta dreapta

Mutarile:

2	0	3	4	0	2	3	4	1	2	3	4
1	6	7	8	1	6	7	8	0	6	7	8
5	10	11	12	5	10	11	12	5	10	11	12
9	13	14	15	9	13	14	15	9	13	14	15
1	2	3	4	1	2	3	4	1	2	3	4
5	6	7	8	5	6	7	8	5	6	7	8
9	10	11	12	9	10	11	12	9	10	11	12
0	13	14	15	13	0	14	15	13	14	0	15

Discuție

Studiați optimalitatea algoritmilor descriși mai sus. Determinați pentru fiecare în parte timpul de execuție, numărul de stări generate și memoria ocupată.

Unele stări sunt generate de două ori. De exemplu se aplică o mutare *s*, apoi una *d*, rezultând aceeași stare. Încercați să eliminați acest inconvenient.

P2. Problema macaralei

Pe o linie de cale ferată se află n vagoane (așezate unul după altul), numerotate cu valori distincte din mulțimea {1..n}. O macara specială poate lua oricâte vagoane de pe linie și le poate așeza în partea dreaptă, la sfârșitul sirului de vagoane. Datorită faptului că linia de cale ferată se află pe un plan înclinat (având partea mai ridicată în dreapta macaralei) vagoanele vor ajunge, prin alunecare (de la dreapta spre stânga), din nou unul lângă altul, în noua ordine creată după operațiile efectuate de macara.

Dându-se ordinea inițială a vagoanelor, se cere să se determine (dacă este posibil) numărul minim de operații pe care trebuie să le efectueze macaraua pentru ca în final vagoanele să se afle în ordine crescătoare conform numerotării lor inițiale.

Restricții

Datele se citesc din fișierul text MACARA.IN, având următoarea structură:

- pe prima linie este scris numărul de vagoane;
- pe a doua linie este scrisă ordinea inițială a vagoanelor.

Soluție

Vom aplică metoda *branch and bound*. Structura atașată fiecărei configurații de vagoane este următoarea:

```
type lista=^art;
  art=record
    inf:Byte;
    cost:Word;
    urm,pred:lista
  end;
```

unde *art^.inf* este poziția de început a secvenței de vagoane, obținută la mutarea anterioară. Această mutare a generat configurația actuală.

Funcția de cost este definită în felul următor: din numărul total de vagoane se scad lungimea celei mai mari secvențe de vagoane care sunt numerotate cu valori consecutive.

Datele sunt afișate sub forma unui sir, fiecare element al sirului reprezentând poziția de început a unei secvențe de vagoane mutate.

Program ProblemaMacaralei;

```
const MaxN=5;

type configuratie=array[1..MaxN] of 1..MaxN;
  lista=^art;
  art=record
    inf:Byte;
    cost:Word;
    urm,pred:lista
  end;
var ConfIni:configuratie;
  stare:pointer;
  cap,r:lista;
  n,i,k,l,:MaxN;

procedure citeste;
  var f:Text;
begin
  Assign(f,'MACARA.IN'); Reset(f);
  Readln(f,n); Readln(f,k);
  for i:=1 to n do
    if not SeekEoln(f) then Read(f,ConfIni[i]);
  Close(f)
end;

procedure ReconstituieMutari(var mutari:configuratie;
                           var nrmut:Byte);
  var t:lista;
  x,sch:Byte;
```

Metoda branch and bound

```
begin
  t:=cap; nrmut:=0;
  while t^.pred<>nil do
  begin
    Inc(nrmut);
    mutari[nrmut]:=t^.inf;
    t:=t^.pred
  end;
  for x:=1 to nrmut div 2 do
  begin
    sch:=mutari[x];
    mutari[x]:=mutari[nrmut-x+1];
    mutari[nrmut-x+1]:=sch
  end;
end;

function Costul(q:lista; c:Word):Word;
  var m,w,e:configuratie;
  L,v,y,sw,Lm,Lmax:Byte;
begin
  ReconstituieMutari(m,L);
  m[L+1]:=c; L:=L+1; w:=ConfIni;
  for v:=1 to L do
  begin
    for y:=m[v] to m[v]+k-1 do e[y-m[v]+1]:=w[y];
    for y:=m[v]+k to n do w[y-k]:=w[y];
    for y:=1 to k do w[n-k+y]:=e[y]
  end;
  Lm:=1; Lmax:=1;
  for v:=1 to n-1 do
    if w[v+1]=1+w[v]
      then begin Inc(Lm);
            if Lm>Lmax then Lmax:=Lm
          end
    else begin if Lmax<Lm then Lmax:=Lm;
            Lm:=1
          end;
  Costul:=n-Lmax
end;

procedure AdaugaNod(p:lista);
  var t,u:lista;
begin
  t:=cap^.urm; u:=cap;
  while (p^.cost>=t^.cost) and (t<>nil) do
    begin u:=t; t:=t^.urm end;
  p^.urm:=t;
  u^.urm:=p
end;
```

```

procedure AfiseazaMutari(p:lista);
var e:configuratie;
nrm:Byte;
begin
  nrm:=0;
  while p^.pred<>nil do
    begin Inc(nrm); e[nrm]:=p^.inf; p:=p^.pred end;
    for i:=nrm downto 1 do Write(e[i], ' ')
  end;

begin
  citeste;
  mark(stare);
  New(cap);
  cap^.inf:=0; cap^.cost:=0;
  cap^.pred:=nil; cap^.urm:=nil;
  while cap^.cost<>cap^.pred^.cost do
    begin
      for i:=1 to n-k do
      begin
        New(r); r^.pred:=cap; r^.inf:=i;
        r^.cost:=cap^.cost+Costul(cap,r^.inf);
        AdaugaNod(r);
      end;
      cap:=cap^.urm;
    end;
  AfiseazaMutari(cap);
  Release(stare);
  Readln;
end.

```

Intrare

7 3 1 7 2 3 5 6 4

Ieșire

2 4 4 4 2

adică la prima mutare se ridică 3 vagoane începând cu al doilea, rezultând configurația:

1 5 6 4 7 2 3

apoi la a doua mutare se ridică 3 vagoane începând cu al 4-lea, rezultând configurația:

1 5 6 3 4 7 2

apoi din nou se ridică 3 vagoane începând cu al 4-lea, rezultând configurația:

1 5 6 2 3 4 7

și din nou se ridică 3 vagoane începând cu al 4-lea, rezultând:

1 5 6 7 2 3 4

pentru ca în final să se ridice 3 vagoane începând cu al doilea, rezultând configurația finală: 1 2 3 4 5 6 7.

P3. Ciclu eulerian

Se consideră un graf neorientat conex. Să se adauge la acest graf un număr minim de muchii astfel încât să apară (dacă este posibil) un ciclu eulerian.

Restricții

Datele se citesc dintr-un fișier text, având următoarea structură:

- pe prima linie este scris numărul de noduri din graf;
- pe următoarele n linii se dă matricea de adiacență a grafului.

Soluție

Stim că un graf neorientat admite un drum eulerian doar dacă este conex și gradul fiecărui nod este un număr par. Să se determine între ce perechi de noduri trebuie adăugată câte o muchie, astfel încât să apară cât mai multe vârfuri având grade impare. Avem trei cazuri:

1. Ambele noduri au grade impare (și nodurile nu sunt adjacente). În acest caz putem adăuga o muchie între cele două noduri, mărinind astfel gradele celor două noduri cu 1, deci, având în vedere că inițial ele au fost impare, acum vor deveni pare.

2. Ambele noduri au grade pare. Si în acest caz s-ar putea să fie necesar să adăugăm o muchie între cele două noduri. De exemplu, să presupunem că avem un graf care are gradele tuturor nodurilor (cu excepția a două dintre ele) numere pare, și în plus, cele două vârfuri cu gradele impare sunt legate între ele printr-o muchie. Atunci, prin adăugarea unei muchii între două noduri având gradele pare, și care nu sunt legate cu cele două noduri având gradele impare, se vor obține încă două noduri cu grade impare, în acest caz putându-se realiza legăturile, astfel încât să se obțină un ciclu eulerian.

3. Dintre noduri numai unul are gradul impar (și nodurile nu sunt legate între ele). Si în acest caz s-ar putea ca unirea celor două vârfuri să ducă la apariția unui ciclu eulerian. De exemplu, să presupunem că avem un graf în care nodurile 1 și 2 au gradele impare, iar nodul 3 are gradul par. Se mai știe că nodul 1 este legat de nodul 2. Atunci prin adăugarea muchiei (1, 3), iar apoi a muchiei (2, 3) se obține un graf în care nodurile 1, 2, 3 au grade pare.

Vom elabora un algoritm care la fiecare pas va adăuga grafului acea muchie care conduce la un graf cu cât mai multe noduri cu grade pare.

Fiecărui graf îi atașăm următoarea înregistrare:

```

lista:^art;
art=record
  inf:pereche;
  cost:Word;
  urm,pred:lista
end;

```

unde inf este perechea de noduri între care se adaugă o muchie la pasul actual, celelalte variabile având semnificațiile cunoscute din rezolvările problemelor precedente.

Funcția Costul va determina costul configurației curente care este egal cu costul configurației de la care a provenit cea curentă, adunat cu numărul nodurilor având gradele impare, după adăugarea unei muchii la pasul curent. Această funcție are următorii parametri:

1. un articol din listă, după care se vor reconstitui toate muchiile adăugate până la pasul curent;

2. muchia care se adaugă la pasul curent;

3. o variabilă booleană care va returna false dacă muchia curentă există deja în graf.

La începutul algoritmului se testează dacă graful nu este deja eulerian.

Program CicluEulerian;

```

const MaxN=10;
type pereche=record
  x,y:0..MaxN
end;
SirPerechi=array[1..MaxN*MaxN] of pereche;
matrice=array[1..MaxN,1..MaxN] of 0..1;
lista:^art;
art=record
  inf:pereche;
  cost:Word;
  urm,pred:lista
end;
var a:matrice;
per:SirPerechi;
grad:array[1..MaxN] of 0..MaxN;
nrp,n,i,j:Byte;
cap,r:lista;
posibil:Boolean;
procedure citeste;
  var f:Text;
begin
  Assign(f,'GRAF.IN'); Reset(f); Readln(f,n);
  for i:=1 to n do
    begin
      for j:=1 to n do
        if not SeekEoln(f) then Read(f,a[i,j]);
      Readln(f)
    end;
  Close(f)
end;

```

```

procedure DeterminaGradeleVarfurilor(q:matrice);
  var cl,c2:Byte;
begin
  for cl:=1 to n do
    begin
      grad[cl]:=0;
      for c2:=1 to n do grad[cl]:=grad[cl]+q[cl,c2]
    end;
end;

procedure AdaugaNod(p:lista);
  var t,u:lista;
begin
  t:=cap^.urm; u:=cap;
  while (p^.cost>=t^.cost) and (t<>nil) do
    begin u:=t; t:=t^.urm end;
  p^.urm:=t; u^.urm:=p
end;

procedure ReconstituieMutari
  (capul:lista; var recons:SirPerechi; var num:Byte);
  var sch:pereche;
begin
  num:=0;
  while capul^.inf.x<>0 do
    begin
      Inc(num);
      recons[num]:=capul^.inf;
      capul:=capul^.pred
    end
end;

function Costul(c:lista;m:pereche; var ok:Boolean):Byte;
  var rec:SirPerechi;
  w:matrice;
  nri,nr,k:Byte;
begin
  ReconstituieMutari(c,rec,nr);
  w:=a;
  for k:=1 to nr do
    begin
      w[rec[k].x,rec[k].y]:=1; w[rec[k].y,rec[k].x]:=1
    end;
  if w[m.x,m.y]=1 then begin ok:=false; Exit end;
  w[m.x,m.y]:=1; w[m.y,m.x]:=1;
  DeterminaGradeleVarfurilor(w);
  nri:=0;
  for k:=1 to n do if odd(grad[k]) then Inc(nri);
  Costul:=nri; ok:=true
end;

```

```

procedure AfiseazaPerechi(capul:lista);
  var rec:SirPerechi;
    nr:Byte;
begin
  ReconstituieMutari(capul,rec,nr);
  for i:=1 to nr do Write('(',rec[i].x,',',rec[i].y,') ')
end;

Begin
  citeste;
  DeterminaGradeleVarfurilor(a);
  posibil:=true;
  for i:=1 to n do
    if odd(grad[i]) then posibil:=false;
  if posibil
    then
      begin
        Writeln('Graful este deja eulerian!'); Readln; Exit;
      end;
  nrp:=0;
  for i:=1 to n-1 do
    for j:=i+1 to n do
      begin Inc(nrp); per[nrp].x:=i; per[nrp].y:=j end;
  New(cap); cap^.cost:=1; cap^.urm:=nil; cap^.pred:=nil;
  New(r); r^.urm:=nil; r^.pred:=cap; cap^.urm:=r;
  r^.cost:=0; r^.inf.x:=0; r^.inf.y:=0;
  cap:=cap^.urm;
  while cap^.cost<>cap^.pred^.cost do
begin
  for i:=1 to nrp do
  begin
    New(r); r^.inf:=per[i]; r^.pred:=cap;
    r^.cost:=Costul(cap,r^.inf,posibil)+cap^.cost;
    if posibil then AdaugaNod(r)
  end;
  cap:=cap^.urm
end;
AfiseazaPerechi(cap)
End.

```

Intrare

6	(1,5) (1,2) (3,6)
0 0 1 0 0 1	
0 0 1 0 1 1	
1 1 0 0 1 0	
0 0 0 0 1 1	
0 1 1 1 0 0	
1 1 0 1 0 0	

Ieșire

Discuție

O altă problemă asemănătoare este problema *poștașului chinez* formulată pentru prima oară în 1962 de către *Mei-Ko Kwan*:

Determinați un ciclu de lungime minimă care trece prin fiecare nod al unui graf neorientat ponderat cel puțin o dată.

În cazul în care graful este eulerian este evident că soluția optimă constă din parcugerea o singură dată a fiecărei muchii. În caz contrar aplicăm următorul algoritm, bazat pe:

Propoziția 1

Intr-un graf neorientat numărul nodurilor de grad impar este par.

Determinăm un drum de cost minim între oricare două noduri de grad impar din graful G . Costul unui astfel de drum este egal cu suma ponderilor muchiilor aflate în componența lui. Construim un graf G' care are ca noduri, nodurile grafului inițial, iar muchii avem doar între nodurile de grad impar (din graful inițial). Ponderea unei muchii este egală cu costul drumului minim dintre cele două noduri extremități (considerate în graful inițial). Determinăm în graful G' un cupaj maxim de cost minim, care va reprezenta soluția problemei.

P4. Drumul pionului

Fie o tablă dreptunghiulară împărțită în $n \times m$ căsuțe identice. Fiecărei căsuțe i-s-a atașat un număr natural pozitiv mai mic sau egal cu $n \times m$ (numerele atașate căsuțelor sunt distințe două câte două). Inițial în una din aceste căsuțe se află un pion care se poate deplasa doar pe orizontală sau pe verticală.

Efectul deplasării pionului constă în interschimbarea conținutului liniei și coloanei de pe care este mutat pionul, cu linia, respectiv coloana pe care este mutat.

Dându-se poziția inițială a pionului, precum și numerele atașate inițial căsuțelor, se cere să se determine numărul minim de deplasări ale pionului, necesare pentru a se trece într-o configurație finală de formă:

1	2	...	m
m+1	m+2	...	2m

$(n-1) \times m + 1$	$(n-1) \times m + 2$...	$n \times m$
----------------------	----------------------	-----	--------------

Restricții

Datele se citesc dintr-un fișier text, având următoarea structură:

- pe prima linie este scrisă perechea n, m ;

- pe a doua linie sunt scrise coordonatele inițiale ale pionului;
- pe următoarele n linii sunt scrise numerele atașate căsuțelor.

Soluție

Vom ataşa fiecărui matrice, următoarea structură:

```
lista^=art;
art=record
  inf:pozitie;
  cost:Word;
  urm,pred:lista
end;
```

unde inf reprezintă ultima poziție în care s-a deplasat pionul pentru a genera configurația curentă, iar celelalte variabile au aceeași semnificație ca și în programele precedente.

Reconstituirea drumului pionului se face de la poziția finală spre ceea inițială (prin intermediul procedurii ReconstituieMutari). De aceea, șirul generat (care are pe prima poziție ultima mutare a pionului) va fi parcurs de la coadă spre început.

Program DrumulPionului;

```
const MaxN=10;
type pozitie=record
  x,y:Byte
end;
lista^=art;
art=record
  inf:pozitie;
  cost:Word;
  urm,pred:lista
end;
configuratie=array[1..MaxN,1..MaxN] of 1..MaxN*MaxN;
vector=array[1..100] of pozitie;
var ConfIni:configuratie;
n,m,i,j,i0,j0:Shortint;
cap,r:lista;
procedure citeste;
var f:Text;
begin
  Assign(f,'TABLA.IN'); Reset(f); Read(f,n,m); Read(f,i0,j0);
  for i:=1 to n do
    begin
      for j:=1 to m do
        if not SeekoIn(f) then Read(f,ConfIni[i,j]); Readln(f)
    end;
  Close(f)
end;
```

```
function Inside(g,h:Shortint):Boolean;
begin
  Inside:=(g>0) and (g<n+1) and (h>0) and (h<m+1)
end;
procedure AdaugaNod(p:lista);
var t,u:lista;
begin
  t:=cap^.urm;
  u:=cap;
  while (p^.cost>=t^.cost) and (t<>nil) do
    begin
      u:=t; t:=t^.urm
    end;
  p^.urm:=t;
  u^.urm:=p
end;
procedure ReconstituieMutari(capul:lista; var mut:vector;
var nrmut:Word);
begin
  nrmut:=1;
  while capul^.inf.x<>0 do
    begin
      Inc(nrmut); mut[nrmut]:=capul^.inf; capul:=capul^.pred
    end;
end;
function Costul(ca:lista; x0,y0:Byte):Word;
var sc,cst,nrm,c,b:Word;
  mutari:vector;
  w:configuratie;
begin
  ReconstituieMutari(ca,mutari,nrm);
  mutari[1].x:=x0; mutari[1].y:=y0; w:=ConfIni;
  for c:=2 to nrm do
    begin
      for b:=1 to m do
        begin
          sc:=w[mutari[c].x,b];
          w[mutari[c].x,b]:=w[mutari[c-1].x,b];
          w[mutari[c-1].x,b]:=sc
        end;
      for b:=1 to n do
        begin
          sc:=w[b,mutari[c].y];
          w[b,mutari[c].y]:=w[b,mutari[c-1].y];
          w[b,mutari[c-1].y]:=sc
        end
    end;
end;
```

IV. DIVERSE

P1. Sortare liniară

Se cere un algoritm de complexitate liniară care să sorteze un sir de numere naturale aparținând intervalului [0..255] știind că nu există în sir mai mult de 250 de elemente egale.

Soluție

Algoritmul care urmează, este cunoscut în literatura de specialitate cu numele de *MathSort*. Are complexitate liniară, dar din păcate nu poate fi folosit pentru orice sir de numere. În schimb poate fi generalizat pentru multe siruri particolare (cum ar fi de exemplu, cel descris în enunț).

Construim un vector b de 256 numere, aparținând intervalului 0, ..., 255, astfel: valoarea elementului de pe poziția k a vectorului b va fi numărul elementelor din sirul dat a având valoarea k . Acest lucru se poate face printr-o singură parcurgere a vectorului a : pentru fiecare element a_i incrementăm cu o unitate numărul b_{a_i} . Afisarea vectorului b necesită cel mult 2^*N operații, unde N este lungimea sirului dat.

Program SortareLiniara;

```

var b:array[0..255] of Byte;
    a:array[1..62025] of Byte;
    n,i,j:Word;
    f:Text;

procedure Citire;
begin
  Assign(f,'SIR.IN'); Reset(f);
  Readln(f,n);
  for i:=1 to n do Read(f,a[i]);
  Close(f)
end;

Begin
  Citire;
  for i:=1 to n do Inc(b[a[i]]);
  for i:=0 to 255 do
    for j:=1 to b[i] do Write(i,' ')
End.

```

P2. Numere complexe

Se consideră un fișier text care conține un sir de numere complexe. Să se scrie un program care realizează următoarele operații:

- determină și afișează sirul modulelor numerelor complexe, ordonat crescător, utilizând un sir de indici;
- afișează forma trigonometrică (modulul și argumentul) pentru fiecare element al sirului de numere complexe, precizând cadranul în care se găsește argumentul ([2]).

Soluție

```

Program hr_complex;
type complex=record
  re,im,modul,argument:Real;
  codran:Byte
end;
mat=array[1..10] of complex;
var z:mat;
  n,i:Byte; { n-dimensiunea sirului numerelor complexe }
  maxim:Real;
  f:Text; cale:string;
procedure citire;
begin
  Write('Dati numele fisierului:'); Readln(cale);
  Assign(f,cale); Reset(f); Readln(f,n);
  for i:=1 to n do
  begin Readln(f,z[i].re,z[i].im);
    z[i].modul:=Sqrt(Sqr(z[i].re)+Sqr(z[i].im))
  end;
  Close(f)
end;

procedure ordonare;
var sw:Boolean; aux:complex;
begin
  repeat
    sw:=true;
    for i:=1 to n-1 do
      if z[i].modul>z[i+1].modul
      then
        begin
          aux:=z[i]; z[i]:=z[i+1]; z[i+1]:=aux; sw:=false
        end
    until sw;
  Writeln('Sirul modulelor:');
  for i:=1 to n do Write(z[i].modul:5:2,' ');
  Writeln
end;

```

```

cst:=0;
for c:=1 to n do
  for b:=1 to b do
    if (c-1)*m+b<>w[c,b] then Inc(cst);
  Costul:=cst
end;

procedure AfiseazaSolutie(ca:lista);
  var mutari:vector;
  nrm,c:Word;
begin
  ReconstituieMutari(ca,mutari,nrm);
  for c:=nrm downto 3 do
  begin
    Write('Pionul trece de pe pozitia ());
    Write(mutari[c].x,',',mutari[c].y,') pe pozitia ());
    Writeln(mutari[c-1].x,',',mutari[c-1].y,)');
  end;
end;

Begin
  citeste;
  New(r);
  r^.cost:=0;
  r^.pred:=nil;
  r^.inf.x:=0;
  r^.inf.y:=0;
  New(cap);
  cap^.inf.x:=i0;
  cap^.inf.y:=j0;
  cap^.pred:=r;
  r^.urm:=cap;
  cap^.urm:=nil;
  cap^.cost:=Costul(cap,i0,j0);
  while cap^.cost<>cap^.pred^.cost do
  begin
    i:=cap^.inf.x+1; j:=cap^.inf.y;
    if Inside(i,j)
    then
    begin
      New(r);
      r^.inf.x:=i;
      r^.inf.y:=j;
      r^.pred:=cap;
      r^.cost:=Costul(cap,i,j)+cap^.cost;
      AdaugaNod(r);
    end;
    i:=cap^.inf.x-1;
    j:=cap^.inf.y;
    if Inside(i,j)
  end;
end;

```

```

  then
begin
  New(r);
  r^.inf.x:=i;
  r^.inf.y:=j;
  r^.pred:=cap;
  r^.cost:=Costul(cap,i,j)+cap^.cost;
  AdaugaNod(r);
end;
i:=cap^.inf.x; j:=cap^.inf.y+1;
if Inside(i,j)
then
begin
  New(r);
  r^.inf.x:=i; r^.inf.y:=j;
  r^.pred:=cap;
  r^.cost:=Costul(cap,i,j)+cap^.cost;
  AdaugaNod(r);
end;
i:=cap^.inf.x; j:=cap^.inf.y-1;
if Inside(i,j)
then
begin
  New(r);
  r^.inf.x:=i;
  r^.inf.y:=j;
  r^.pred:=cap;
  r^.cost:=Costul(cap,i,j)+cap^.cost;
  AdaugaNod(r);
end;
  cap:=cap^.urm
end;
AfiseazaSolutie(cap)
End.

```

Intrare

3 3
1 1
5 4 6
2 1 3
8 7 9

Ieșire

Pionul de pe pozitia (1,1) se deplaseaza pe pozitia (2,1).
Pionul de pe pozitia (2,1) se deplaseaza pe pozitia (2,2).

Elaborați un algoritm polinomial care rezolvă această problemă.

```

procedure afisare(x:pol; dim:Byte);
begin
  Write(' ');
  for i:=0 to dim do
    if x[i]<>0
    then
      begin
        if x[i]<0 then Write(#8#8,'- ');
        if ((x[i]<>1) and (x[i]<>-1)) or (i=0)
        then Write(Abs(x[i]):3:2,'*');
        if i=0 then Write(#8,'+')
        else
          begin
            Write('x');
            if i>1 then Write('^',i,'+')
            else Write ('+')
          end
      end;
  Writeln(#8#8,' ');
end;

procedure suma;
begin
  Writeln('Suma polinoamelor:');
  afisare(a,na);
  afisare(b,nb);
  Writeln('este:');
  if na>nb
  then
    begin
      for i:=0 to nb do result[i]:=a[i]+b[i];
      for i:=nb+1 to na do result[i]:=a[i];
      afisare(result,na)
    end
    else
    begin
      for i:=0 to na do result[i]:=a[i]+b[i];
      for i:=na+1 to nb do result[i]:=b[i];
      afisare(result,nb)
    end
  end;
end;

procedure diferente;
begin
  Writeln('diferenta polinoamelor:');
  afisare(a,na); afisare(b,nb);
  Writeln(' este:');
  if na>nb
  then

```

```

begin
  for i:=0 to nb do result[i]:=a[i]-b[i];
  for i:=nb+1 to na do result[i]:=a[i];
  afisare(result,na)
end;
else
begin
  for i:=0 to na do result[i]:=a[i]-b[i];
  for i:=na+1 to nb do result[i]:=-b[i];
  afisare(result,nb)
end;
end;

procedure produs;
begin
  Writeln('produsul polinoamelor:');
  afisare(a,na); afisare(b,nb);
  Writeln('este:');
  for i:=0 to na+nb do result[i]:=0;
  for i:=0 to na do
    for j:=0 to nb do result[i+j]:=result[i+j]+a[i]*b[j];
  afisare(result,na+nb)
end;

procedure derivata(x:pol; nx:Byte);
begin
  Writeln('Derivata polinomului:');
  afisare(x,nx);
  Writeln('este:');
  for i:=1 to nx do result[i-1]:=x[i]*i;
  afisare(result,nx-1)
end;

procedure deriv;
var n:Char;
begin
  repeat
    Write('pentru care dintre polinoame doriti sa calculati');
    Writeln(' derivata (1,2)?'); Readln(n);
    if n='1' then derivata(a,na)
    else if n='2' then derivata(b,nb)
  until n in ['1','2']
end;

procedure ev(x:pol; nx:Byte);
var pct,putere,valoare:Real;
begin
  Write('Punctul dat? '); Readln(pct);
  putere:=1;
  valoare:=0;

```

```

procedure cadranul;
begin
  if z[i].im>=0
  then if z[i].re>=0 then z[i].cadran:=1
       else z[i].cadran:=2
  else if z[i].re<0 then z[i].cadran:=3
       else z[i].cadran:=4
end;

procedure trig;
begin
  for i:=1 to n do
  begin
    if z[i].re<>0
    then
      begin
        z[i].argument:=Arctan(z[i].im/z[i].re)*(180/PI);
        cadranul;
      end
    else Writeln('Impartire imposibila!')
  end;
end;

procedure scrie;
begin
  for i:=1 to n do
  begin
    Writeln;
    Writeln(z[i].re:5:2,'+',z[i].im:5:2,'*i');
    Write('modulul:',z[i].modul:5:2);
    Writeln('argument:',z[i].argument:5:2,'cadran:',z[i].cadran)
  end;
end;

Begin
  citire;
  ordonare;
  trig;
  scrie
End.

```

P3. Polinoame

Se consideră două polinoame de forma:

$$P(X) = a_0 X^n + a_1 X^{n-1} + \dots + a_n \text{ și } Q(X) = b_0 X^m + b_1 X^{m-1} + \dots + b_m.$$

Să se scrie un program care realizează următoarele operații cu polinoame:

- adunarea și scăderea a două polinoame;
- înmulțirea a două polinoame;

- c) calculul derivatei unui polinom;
- d) calculul valorii polinomului într-un punct dat.

Datele de intrare se vor citi dintr-un fișier text:

- pe prima linie este scris gradul primului polinom;
- pe linia a doua sunt scrise coeficienții primului polinom, delimitați printr-un spatiu sau mai multe spații, primul coeficient fiind termenul liber;
- pe linia a treia este scris gradul celui de-al doilea polinom;
- pe linia următoare sunt scrise coeficienții celui de-al doilea polinom în mod similar cu primul polinom ([2]).

Soluție

Algoritmul pe care îl vom folosi în calculul valorii unui polinom într-un punct dat necesită efectuarea a $2n - 1$ înmulțiri și n adunări. Din fericire există algoritmi având complexitate mai scăzută. De exemplu metoda lui Horner necesită efectuarea a n înmulțiri și n adunări. În acest caz polinomul se va scrie sub forma:

$$P(x) = [(...(a_n x + a_{n-1})x + a_{n-2})x + \dots + a_1] x + a_0, \text{ iar algoritmul este urmatorul:}$$

```

valoare:=an
for i:=n-1 downto 0 do
  valoare:=valoare*x+ai { în punctul x se calculează valoarea }

```

Pentru un polinom am definit un tip pol ca fiind un vector de coeficienți. Gradele celor două polinoame sunt reținute în variabilele na, nb. În acest caz, reprezentarea sugerată este suficient de structurată. Definiți un tip de date care, pe lângă coeficienți să conțină tipul polinoomului pentru cazul în care există n polinoame. Se recomandă să nu se reprezinte mai multe polinoame sub forma unei matrice de coeficienți, respectiv a unui vector de grade, ci sub forma unor structuri care permit prelucrări mai eficiente.

Program polinoame;

```

type pol=array[0..100] of Real;
var a,b,resultat:pol;
na,nb,nz,i,j:Byte; opt:Char;
fi:Text;
fis:string;

```

```

procedure cit(var x:pol; var dim:Byte);
begin
  Readln(fi,dim);
  for i:=0 to dim do Read(fi,x[i])
end;

```

```

procedure citire;
begin
  Writeln('numele fisierului de intrare'); Readln(fis);
  Assign(fi,fis); Reset(fi);
  cit(a,na); cit(b,nb)
end;

```

```

for i:=0 to nx do
begin
  valoare:=valoare+x[i]*putere;
  putere:=putere*pct
end;
Writeln('Valoarea polinomului:'); afisare(x,nx);
Writeln(' In punctul ',pct,' este ',valoare)
end;

procedure eval;
var n:Char;
begin
repeat
  Writeln('Care dintre polinoame doriti sa-l evaluati (1,2)?');
  Readln(n);
  if n='1' then ev(a,na)
  else if n='2' then ev(b,nb)
until n in ['1','2'];
end;

Begin
repeat
  Writeln('1.. citire');
  Writeln('2.. suma polinoamelor');
  Writeln('3.. diferența polinoamelor');
  Writeln('4.. inmultirea polinoamelor');
  Writeln('5.. derivarea unui polinom');
  Writeln('6.. evaluarea unui polinom-intr-un punct');
  Writeln('7.. Exit'); Readln(opt);
  case opt of
    '1': citire;
    '2': suma;
    '3': diferența;
    '4': produs;
    '5': deriv;
    '6': eval;
    '7': ;
    else Write('Optiunea trebuie sa fie in intervalul 1-8')
  end
until opt='7';
End.

```

P4. Ordonare în matrice

Se consideră o matrice a de dimensiune $n \times m$. Să se rearanjeze elementele în matrice astfel încât ele să apară în ordine crescătoare atât pe linii cât și pe coloane. Matricea a se găsește în fișierul de intrare INPUT.TXT, fiecare linie a matricei pe câte o linie din fișier.

Nu este specificată valoarea lui n și a lui m .

Matricea obținută se va scrie în fișierul OUTPUT.TXT, în mod similar cu cea de intrare.

În cazul când există mai multe soluții se va scrie în fișier numai una ([2]).

Soluție

Pentru rezolvare, elementele din matrice sunt ordonate crescător ca într-un șir de $n*m$ elemente și puse în grupuri de câte m elemente pe fiecare linie. În același timp se realizează și corespondența între indicii elementelor din matrice și numărul de ordine al acestora în matricea liniarizată.

```

Program ordine;
var a:array[1..100,1..100] of Integer;
  n,m:Integer;
  f,g:Text;
procedure citeste;
var i,j:Integer;
begin
  Assign(f,'INPUT.TXT'); Reset(f);
  i:=0;
  while not Eof(f) do
  begin
    Inc(i); j:=0;
    while not Eoln(f) do
    begin Inc(j); Read(f,a[i,j]) end;
    Readln(f)
  end;
  n:=i; m:=j
end;

procedure scrie;
var i,j:Integer;
begin
  Assign(g,'OUTPUT.TXT'); Rewrite(g);
  for i:=1 to n do
  begin for j:=1 to m do Write(g,a[i,j], ' '); Writeln(g) end;
  Close(g)
end;

procedure ordonare;
var i1,j1,i2,j2,k,t,c:Integer;
begin
  for k:=1 to n*m-1 do
  begin
    for t:=k+1 to m*n do
    begin
      if k mod m>0 then i1:=k div m+1
      else i1:=k div m;
      j1:=k-(i1-1)*m;
      if a[i1,j1]>a[t,m] then
        begin
          a[i1,j1]:=a[t,m];
          a[t,m]:=a[i1,j1];
        end;
    end;
  end;
end;

```

```

if t mod m>0 then i2:=t div m+1
else i2:=t div m;
j2:=t-(i2-1)*m;
if a[i1,j1]>a[i2,j2]
then
begin
  c:=a[i1,j1];
  a[i1,j1]:=a[i2,j2];
  a[i2,j2]:=c
end
end;
Begin
  citeste;
  ordonare;
  scrie
End.

```

Exemplu:

Intrare

```

3 2 5 44 3 1
6 7 55 4 3 2
4 10 4 2 4 5
34 5 6 8 2 4
5 3 2 4 41 3

```

Ieșire

```

1 2 2 2 2 2
3 3 3 3 3 4
4 4 4 4 4 5
5 5 5 6 6 7
8 10 34 41 44 55

```

P5. Produs cartezian

Se citește din fișierul standard de intrare un număr $n \in N^*$ și o mulțime M de litere mici din alfabetul englezesc.

Să se determine și să se scrie în fișierul text **CART** elementele produsului cartezian

$$\underbrace{M \times M \times \dots \times M}_{\text{de } n \text{ ori}} = M^n$$

de n ori

Să se listeze conținutul acestui fișier ([2]).

Soluție

Program produs_carteziian;

```

var n:Byte;
  m:string;
  x:array[1..20] of Byte;
  f:Text;

```

```

procedure scrie;
var i:Byte;
begin
  for i:=1 to n do Write(f,m[x[i]]);
  Writeln(f)
end;

procedure cart(i:Byte);
var j:Byte;
begin
  for j:=1 to Length(m) do
  begin
    x[i]:=j;
    if i=n then scrie
    else cart(i+1)
  end
end;

Begin
  Write('n='); Readln(n);
  Write('multimea M:'); Readln(m);
  Assign(f,'CART'); Rewrite(f);
  cart(1);
  Close(f);
  Reset(f);
  while not Eof(f) do
  begin
    Readln(f,m);
    Write(m); Readln
  end;
  Close(f)
End.

```

P6. Submulțimi

Se citește o linie din fișierul standard de intrare care cuprinde o mulțime de numere întregi. Să se genereze submulțimile acestei mulțimi, memorându-le în câte o linie a fișierului text **SUBM.TXT**.

Să se afișeze fișierul creat ([2]).

Soluție

Program submultimi;

```

var n:Byte;
  m:array[1..20] of Integer;
  x:array[0..20] of Byte;
  f:Text;

```

```

procedure scrie(i:Byte);
  var j:Byte;
begin
  for j:=1 to i do Write(f, ' ', m[x[j]]);
  Writeln(f);
end;

procedure subm(i:Byte);
  var j:Byte;
begin
  for j:=x[i-1]+1 to n do
  begin
    x[i]:=j;
    scrie(i);
    subm(i+1);
  end;
end;

Begin
  n:=0;
  Write('Multimea M:');
  while not Eoln do
  begin Inc(n); Read(m[n]) end;
  Readln;
  Assign(f,'SUBM.TXT'); Rewrite(f);
  subm(1);
  Close(f);
  Reset(f);
  Writeln('Submultimile:');
  Writeln(Chr(237):3);
  while not Eof(f) do
  begin
    while not Eoln(f) do
    begin
      Read(f,n);
      Write(n:3)
    end;
    Readln;
    Readln(f)
  end;
  Close(f)
End.

```

Discuție

În continuare vă supunem atenției încă o modalitate de a genera submulțimile unei mulțimi date: se poate stabili o bijecție între mulțimea numerelor $0, \dots, 2^n - 1$ și submulțimile unei mulțimi cu n elemente. Generarea submulțimilor se face folosind un ciclu repetitiv în $2^n - 1$ pași.

```

program Submultimi;
  var i,nr,p,n:Longint;
begin
  Write('n:'); Readln(n);
  for i:=1 to (1 shl n)-1 do
  begin
    Write('Submultimea ',i,':');
    p:=Trunc(Ln(i)/Ln(2));
    if i=1 shl (p+1) then Inc(p);
    nr:=i; { se salveaza valoarea lui i }
    while p>=0 do
    begin
      Write(p+1,' ');
      nr:=nr-1 shl p;
      while 1 shl p>nr do Dec(p)
    end;
    Writeln(' ')
  end;
End.

```

P7. Numere mari

Se consideră două numere naturale n și m care au cel puțin 200 de cifre. Să se determine produsul celor două numere.

Pe prima linie a fișierului de intrare INPUT.TXT este scris numărul n , având cifrele despărțite prin câte un spatiu; pe a doua linie este scris numărul m , în mod similar.

În fișierul OUTPUT.TXT se va scrie pe fiecare linie câte o cifră din rezultat ([2]).

Soluție

Problema se rezolvă simulând înmulțirea prin calcul pe vectori de cifre. Numerele sunt memorate în vectorii a și b în care fiecare element constituie o cifră. Produsul se determină în vectorul c a cărui dimensiune maximă este dată de constanta $llimita$.

Cifrele lui c se determină începând cu $c_{llimita}$ și terminând cu c_1 prin înmulțirea cifrelor corespunzătoare din a , respectiv din b . Evident, înmulțirea celor două cifre poate produce o cifră de transport, care trebuie gestionată corespunzător. La afișare se vor ignora zerourile nesemnificative:

```

Program nr_mari;
const limita=10000;
      llimita=20000;
var a,b:array [1..llimita] of Byte;
     c:array [1..llimita] of Byte;
     f,g:Text;
     rez,adun,n,m,poz,pz,i,j,ra,ri,v:Integer;

```

```

Begin
  Assign(f, 'INPUT.TXT'); Reset(f);
  Assign(g, 'OUTPUT.TXT'); Rewrite(g);
  n:=0;
  while not SeekEoln(f) do
    begin n:=n+1; Read(f,a[n]) end;
  Readln(f); m:=0;
  while not SeekEoln(f) do
    begin m:=m+1; Read(f,b[m]) end;
  Close(f);
  for i:=1 to llimita do c[i]:=0;
  poz:=llimita;
  for i:=m downto 1 do
    begin
      pz:=poz;
      ra:=0; ri:=0;
      for j:=n downto 1 do
        begin
          v:=b[i]*a[j]+ri;
          adun:=v mod 10;
          ri:=v div 10;
          rez:=adun+c[pz]+ra;
          ra:=rez div 10;
          c[pz]:=rez mod 10; pz:=pz-1
        end;
      ri:=ri+ra;
      while ri>0 do
        begin
          adun:=c[pz]+ri;
          c[pz]:=adun mod 10;
          ri:=adun div 10
        end;
      poz:=poz-1
    end;
  poz:=1;
  while c[poz]=0 do poz:=poz+1;
  for i:=poz to llimita do Writeln(g,c[i]);
  Close(g)
End.

```

P8. Conjectura lui Goldbach

Se consideră un număr $n > 6$, par. Să se determine toate reprezentările lui n ca sumă de numere prime, sumă cu număr minim de termeni.

Rezultatele vor fi scrise în fișierul OUTPUT.TXT, fiecare linie conținând toți termenii dintr-o reprezentare ([2]).

Soluție

Rezolvarea problemei se bazează pe faptul că reprezentarea unui număr par ca sumă formată dintr-un număr minim de termeni primi este dată de conjectura Goldbach: numărul se poate reprezenta ca sumă de două numere prime. În aceste condiții se verifică numerele i (de la 1 la jumătatea numărului dat n), în scopul de a determina numerele care sunt prime, apoi se verifică dacă și numărul $n-i$ este prim.

```

Program termeni;
var, n:Integer; sw:Boolean; g:Text;
procedure prime(a:Integer; var sw:Boolean);
  var i,x:Integer;
begin
  sw:=true; i:=2;
  x:=Trunc(Sqr(a)); { daca doriti viteza mai mare la }
  while sw and (i<=x) do { algoritm atunci folositi i*i<=x }
    if a mod i=0 then sw:=false
    else Inc(i)
end;
procedure Goldbach;
  var i:Integer;
begin
  for i:=1 to Trunc(n/2) do
    begin
      prime(i,sw);
      if sw then prime(n-i,sw);
      if sw then Writeln(g,i,' ',n-i)
    end;
  Close(g)
end;
Begin
  Readln(n); Assign(g, 'OUTPUT.TXT'); Rewrite(g);
  Goldbach
End.

```

Exemplu:

Intrare

n=8

Ieșire

1 7

3 5

P9. Bridge (I)

Într-un joc de bridge există patru jucători, doi câte doi formând o echipă sau o axă. Celor patru jucători li se împart în mod egal toate cărțile. Cărțile de joc le vom codifica în felul următor: 2..10, J, D, K, A. Jucătorii sunt numerotați de la 1 la 4. Axele sunt formate din jucătorii numărul 1 și 3, respectiv jucătorii numărul 2 și 4.

În mod analog se calculează numărul de onoruri al fiecărui jucător din fiecare culoare. În matricea O de dimensiune 4×4 elementul $O_{i,j}$ reprezintă numărul de onoruri din culoarea i al jucătorului j . În final se însumează elementele din cele două matrice pe coloanele 1 și 3 respectiv 2 și 4 și se calculează maximele.

Program bridge;

```

const onor=['A','D','J','K'];
var L,O:array[1..4,1..4] of Byte;
    a1,a2,m1,m2,c1,c2,ax,c,max,i,v,e,j,k,on:Integer;
    b:string[50];
    f:Text;

Begin
  for i:=1 to 4 do
    for j:=1 to 4 do begin L[i,j]:=0; O[i,j]:=0 end;
  Assign(f,'INPUT.TXT'); Reset(f);
  for j:=1 to 4 do
    for c:=1 to 4 do
      begin
        Readln(f,b);
        if b[2]<>'F'
        then
          begin
            Val(b[1],v,e);
            if Pos('10',b)<>0 then L[v,j]:=Length(b)-2
              else L[v,j]:=Length(b)-1;
            for k:=2 to Length(b) do
              if b[k] in onor then Inc(O[v,j]);
          end
        end;
      Close(f);
      a1:=0; a2:=0;
      for i:=1 to 4 do
        begin
          m1:=L[i,1]+L[i,3];
          if m1>a1 then begin a1:=m1; c1:=i end
        end;
      for i:=1 to 4 do
        begin
          m2:=L[i,2]+L[i,4];
          if m2>a2 then begin a2:=m2; c2:=i end
        end;
        if a1>a2 then begin max:=a1; c:=c1; ax:=1 end
          else begin max:=a2; c:=c2; ax:=2 end;
        on:=O[c,ax]+O[c,ax+2];
        Writeln('Axa jucatorilor ',ax,'-',ax+2,' are din culoarea ',c);
        Writeln(' un numar de ',max,' carti cu ',on,' onoruri')
      End.

```

Intrare

1345678
2DKA
3F
4JDKA
12A
2234J
323410JD
4234
1910D
278910
3KA
4567
1JK
256
356789
48910

Ieșire

Axa jucatorilor 2-4 au din culoarea 4
un numar de 11 carti cu 2 onoruri

P11. Jumătate

Se consideră o mulțime de n numere întregi.

Să se particioneze această mulțime în două submulțimi astfel încât oricare element al primei submulțimi să fie mai mic decât b (o valoare întreagă cunoscută) iar elementele celei de a doua submulțimi să fie mai mari sau cel mult egale cu acest număr b .

Să se realizeze un program care rezolvă problema, folosind tehnica *Divide et Impera*.

Pe prima linie a fișierului INPUT.TXT sunt scrise numerele n și b ; pe a doua linie este scris sirul celor n numere despărțite prin spațiu. Cele două submulțimi se vor afișa pe două linii diferite în fișierul OUTPUT.TXT ([2]).

Exemplu:

Intrare

7 5
1 3 2 -1 6 -11 9

Ieșire

-11 -1 1 2 3
6 9

Soluție

În rezolvarea problemei se vor efectua doi pași:

- ordonarea vectorului s ;
- determinarea indicelui elementului din vectorul s care reprezintă ultimul element al primei submulțimi. În subprogramul recursiv care realizează acest al doilea paș, indicele căutat este identificat prin înjumătățirea intervalului de căutare. Inițial intervalul de căutare este $[1, n]$.

Fiecare carte considerată *onor* are un punctaj care nu depinde de culoare. Astfel: pentru o carte codificată cu valoarea J se obține 1 punct, pentru una cu valoarea D se obțin 2 puncte, pentru K se obțin 3 puncte, iar pentru A 4 puncte.

Dându-se o repartiție de cărți pentru cei 4 jucători, să se determine care este axa cu cele mai multe puncte și câte puncte au împreună cei doi jucători.

Fișierul de intrare INPUT.TXT conține 16 linii. Fiecare grup de 4 linii conține cărțile primite, în ordine, de fiecare jucător (de la 1 la 4). Pe fiecare linie din cele patru sunt scrise numai cărțile de o singură culoare. Dacă un jucător nu primește nici o carte de o anumită culoare, atunci pe una din cele patru linii care-i corespund va fi trecut caracterul F ([2]).

Soluție

Datele din fișierul de intrare se citesc într-o variabilă de tip **string**. Odată cu această citire se calculează punctele fiecărui jucător, punctajele fiind memorate într-un vector de dimensiune 4. Se calculează, și se afișează punctajele pe axa *a1* și axa *a2*.

Program bridge;

```

var a:array[1..4] of Byte;
  a1,a2,i,j,k:Integer;
  b:string[50]; f:Text;
Begin
  Assign(f,'INPUT.TXT'); Reset(f);
  for i:=1 to 4 do
    begin
      a[i]:=0;
      for j:=1 to 4 do
        begin
          Readln(f,b);
          if b<>'F' then
            for k:=1 to Length(b) do
              case b[k] of
                'J':a[i]:=a[i]+1;
                'D':a[i]:=a[i]+2;
                'K':a[i]:=a[i]+3;
                'A':a[i]:=a[i]+4
              end
        end;
      Close(f);
      a1:=a[1]+a[3]; a2:=a[2]+a[4];
      if a1>a2 then Writeln('Axa jucatorilor 1,3 are punctajul ',a1)
      else if a1<a2 then Writeln('Axa jucatorilor 2,4 are punctajul ',a2)
      else Writeln('Axele au acelasi punctaj ',a1)
    End.
End.

```

Intrare

345678

DKA

E

JDKA

2A

234J

23410JD

234

910D

78910

KA

567

JK

56

56789

8910

Iesire

Axa jucatorilor 1,3 are punctajul 28

P10. Bridge (II)

Intr-un joc de bridge există patru jucători, doi către doi jucând împreună pe o axă. Le sunt împărțite în mod egal toate cărțile. 2, 10, J, D, K, A sunt codurile pentru valourile cărților. Jucătorii sunt numerotati de la 1 la 4. Axele sunt formate din jucătorii având numărul 1 și 3, respectiv jucătorii cu numărul 2 și 4. Onoruri sunt considerate cărțile J, D, K, A. Cărțile pot avea patru culori, codificate prin caracterele: 1, 2, 3, 4.

Dându-se o repartiție de cărți pentru cei patru jucători, să se determine: axa cu cele mai multe cărți de aceeași culoare și culoarea respectivă; câte cărți de această culoare are această axă; câte onoruri are axa din această culoare.

Fișierul de intrare INPUT.TXT conține 16 linii. Fiecare grup de 4 linii conține cărțile primite în ordine de fiecare jucător (de la 1 la 4).

Pe fiecare linie din cele patru sunt scrise numai cărțile de o singură culoare. Codul culorii este primul caracter de pe acea linie. Dacă un jucător nu primește nici o carte de o anumită culoare, atunci pe una din cele patru linii, după codul culorii respective, va fi trecut caracterul 'F'. Rezultatele vor fi afișate pe ecran.

Soluție

Se determină (pe cele două axe de jucători) numărul de cărți din fiecare culoare. În acest scop se folosește o matrice *L* de dimensiune 4x4 în care elementul *L_{i,j}* are semnificația: din culoarea *i* jucătorul *j* are *L_{i,j}* cărți.

Datele din fișierul de intrare se citesc într-o variabilă *b* de tip **string**.

Elementul *L_{i,j}* este egal cu numărul de caractere al variabilei *b*, mai puțin primul caracter, care reprezintă codul culorii, iar dacă în sirul *b* apare cartea de valoare 10 atunci *L_{i,j}* va fi egal cu lungimea lui *b* minus 2.

```

for k:=1 to p do b[k]:=a[1,i+k-1];
m:=0;
for k:=1 to Trunc(p/2) do
  if b[k]<>b[p-k+1] then m:=1;
if m=0
then
  if maxl<p then begin maxl:=p; lin:=1; pozli:=i end
end
end;
maxc:=0;
for c:=1 to n do
begin
  for i:=1 to n-1 do
  begin
    for j:=i+1 to n do
    begin
      p:=j-i+1;
      for k:=1 to p do b[k]:=a[i+k-1,c];
      m:=0;
      for k:=1 to Trunc(p/2) do
        if b[k]<>b[p-k+1] then m:=1;
      if m=0
      then
        if maxc<p then begin maxc:=p; col:=c; pozci:=i end
      end
    end
  end;
  if maxl>maxc
  then
    for i:=1 to maxl do Write(a[lin,pozli+i-1])
  else
    for i:=1 to maxc do Write(a[pozci+i-1,col]);
End.

```

Exemplu:

Intrare
acnzzr
moazic
ijdpas
cokoad
tclimic
rpimtu

Ieșire
cojoc

Discuție

Încercați să rezolvați această problemă și în cazul în care cuvântul palindromic nu se află doar pe o linie sau pe o coloană, ci el este înscris într-o succesiune de căsuțe învecinate pe orizontală sau verticală.

P13. Sortare prin interclasare

Se dă un sir (a_n) , $1 < n < 100$, cu elemente numere reale. Să se ordeneze descrescător acest sir, folosind algoritmul sortării prin interclasare (metoda "divide et impera").

Datele de intrare se citesc dintr-un fișier text SIR.TXT cu structura:
 $a_1 \ a_2 \dots \ a_n$

Elementele sirului sunt separate printr-un spațiu. Sirul sortat se va memora într-un fișier ORDONAT.TXT, pe o singură linie, elementele fiind separate printr-un spațiu ([2]).

Exemplu:

Intrare

2 1 3 7 5.2 6.123

Ieșire

7 6.123 5.2 3 2 1

Soluție

Se va utiliza metoda "divide et impera" care presupune împărțirea sirului inițial în două subșiruri de lungimi aproximativ egale. Vom repeta împărțirea în subșiruri până când subșirurile obținute vor fi de lungime 1. Un subșir de lungime 1 este ordonat (crescător sau descrescător). Subșirurile ordonate vor fi interclasate, constituind noi subșiruri ordonate. Procedeul se repetă până când se obține soluția problemei inițiale.

```

Program sortare_prin_interclasare;
var n,i:Byte; el:Real;
  a:array[1..100] of Real;
  f,g:Text;

procedure sortintc(u,v:Byte);
  var m:Byte;

procedure intercl; { procedura de interclasare }
  var b:array[1..100] of Real;
  i,j,k,m:Byte;
begin
  m:=(u+v) div 2; { Se imparte sirul in doua subsiruri }
  i:=u;
  j:=m+1;
  k:=u-1;
  repeat { Se alege elementul mai mare si se }
    k:=k+1; { memoraza in sirul b. Se trece la }
    if a[i]>a[j] { elementul urmator in subsirul din }
    then { care s-a preluat elementul, pana }
    begin { la terminarea unuia dintre cele }
      b[k]:=a[i];
      i:=i+1
    end { doua subsiruri }
    else begin b[k]:=a[j]; j:=j+1 end;
  until (i>m) or (j>v);

```

```

Program divimp;
var s:array[1..100] of Integer;
    m,k,n,b1,i,c:Integer; f,g:Text;
procedure sortare;
var i,j:Integer;
begin
  for i:=1 to n-1 do
    for j:=i+1 to n do
      if s[i]>s[j] then begin c:=s[i]; s[i]:=s[j]; s[j]:=c end
end;
procedure recurs(a,b:Integer);
begin
  if (a=b) or ((b-a)=1) then k:=a
  else begin
    m:=Trunc((a+b)/2);
    if s[m]<b1 then recurs(m,b)
    else recurs(a,m)
  end
end;
procedure citeste;
var i:Integer;
begin
  Assign(f,'INPUT.TXT'); Reset(f); Read(f,n,b1);
  for i:=1 to n do Read(f,s[i]);
  Close(f)
end;
Begin
  citeste;
  sortare;
  recurs(1,n);
  Assign(g,'OUTPUT.TXT'); Rewrite(g);
  for i:= 1 to k do Write(g,s[i],' ');
  Writeln(g);
  for i:= k+1 to n do Write(g,s[i],' ');
  Close(g)
End.

```

P12. Scrabble

O tablă de scrabble de dimensiuni $n \times n$ este completată cu n^2 litere mici ale alfabetului englez. Jack trebuie să descopere cel mai lung cuvânt cu proprietatea palindromului (citindu-l de la dreapta la stânga sau de la stânga la dreapta se obține același cuvânt).

Cuvintele în scrabble se formează începând cu orice poziție pe linii sau coloane. Scrieți un program care determină palindromul de lungime maximă.

Datele de intrare se găsesc în fișierul INPUT.TXT care conține în linii; fiecare linie conține n litere între care nu există nici un separator.

Răspunsul va fi afișat pe ecran. În cazul în care există mai multe soluții se vor afișa toate ([2]).

Soluție

Pentru rezolvarea problemei se efectuează următoarele calcule:

- se caută dimensiunea maximă $maxl$ a palindromelor pe cele n linii;
- în variabila lin se rețină indicele liniei pe care se găsește palindromul de lungime maximă și în variabila $pozli$ indicele coloanei în care se găsește prima literă a acestuia.
- se caută dimensiunea maximă $maxc$ a palindromelor pe cele n coloane;
- în variabila col se rețină indicele coloanei pe care se găsește palindromul de lungime maximă și în variabila $pozci$ indicele liniei pe care se găsește prima literă a acestuia;
- se afișează cel mai lung palindrom dintre cele două găsite.

Vectorul b este folosit pentru o mai ușoară verificare a proprietății de palindrom a sirului curent.

```

Program palindrom;
var a:array[1..100,1..100] of Char;
    b:array[1..100] of Char;
    n,i,j,l,c,k,m,p,col,lin,pozli,pozci,maxl,maxc:Integer;
    f:Text;

Begin
  Assign(f,'INPUT.TXT'); Reset(f);
  n:=0;
  while not Eof(f) do
  begin
    Inc(n); j:=0;
    while not Eoln(f) do
    begin Inc(j); Read(f,a[n,j]) end;
    Readln(f)
  end;
  Close(f); maxl:=0;
  for l:=1 to n do
  begin
    for i:=1 to n-1 do
    begin
      for j:=i+1 to n do
      begin
        begin
          p:=j-i+1;
        end;
      end;
    end;
  end;

```

```

for j:=1 to nrc do
  if (x in clasa[j]) or (y in clasa[j])
  then begin k:=k+1; a[k]:=j end;
  case k of
    { x,y formeaza o noua clasa }
    0:begin nrc:=nrc+1; clasa[nrc]:=[x,y] end;
    { x sau y apartine clasei cu indicele a[1] }
    1:clasa[a[1]]:=clasa[a[1]]+[x,y];
    { x,y aparțin unor clase diferite cu indicele a[1],a[2] }
    2:begin
      clasa[a[1]]:=clasa[a[1]]+clasa[a[2]];
      if a[2]=nrc then nrc:=nrc-1
      else clasa[a[2]]:=[];
    end
  end;
end;
k:=0;
for i:=1 to nrc do
  if clasa[i]<>[]
  then
  begin
    k:=k+1;
    Write('Clasa',k:3,' :');
    for x:='a' to 'z' do
      if x in clasa[i] then Write(x:2);
    Writeln
  end;
End.

```

Varianta 2

- fiecare pereche citită va forma inițial câte o clasă;
- clasele cu elemente comune se reunesc în clasa cu indice mai mare, cealaltă transformându-se în mulțimea vidă; rămân, în final, doar clase disjuncte care reprezintă chiar clasele de echivalență și "clasele" vide.

Program Clase_de_echivalenta_solutia_2;

```

var clasa:array[1..40] of set of 'a'..'z';
x,y:'a'..'z';
r:Char;
n,i,j,k:0..40;
Begin
  Write('Introduceți numarul de perechi n='); Readln(n);
  for i:=1 to n do
  begin
    Readln(x,r,y);
    clasa[i]:=[x,y]
  end;

```

```

for i:=1 to n-1 do
begin
  for j:=i+1 to n do
    if clasa[i]*clasa[j]<>[] { clasele nedisjuncte se reunesc }
    then
    begin
      clasa[j]:=clasa[j]+clasa[i];
      clasa[i]:=[];
      Break { devine vida }
    end
  end;
  k:=0;
  for i:=1 to n do
    if clasa[i]<>[]
    then
    begin
      k:=k+1;
      Write('Clasa',k:3,' :');
      for x:='a' to 'z' do if x in clasa[i] then Write(x:2);
      Writeln
    end;
End.

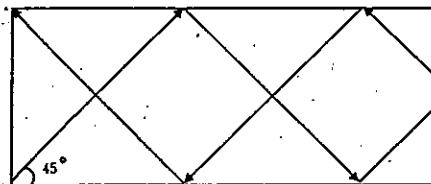
```

P15. Raza de lumină

Fie o matrice A cu m linii și n coloane ($m, n \leq 256$). O rază pornește din colțul stânga jos al matricei, formând un unghi de 45° cu prima coloană și ultima linie. Raza se reflectă când ajunge pe prima sau ultima linie, respectiv prima sau ultima coloană. Să se afișeze toate pozițiile atinse de rază, până când ea ajunge într-un colț al acesteia ([2]).

Soluție

Indiceii elementelor din matrice reprezintă pozițiile prin care trece raza. Prin urmare avem nevoie de matrice numai pentru a găsi problema.



Când raza:

- 1) urcă – indicele de linie scade
- 2) coboară – indicele de linie crește
- 3) se deplasează spre dreapta – indicele de coloană crește
- 4) se deplasează spre stânga – indicele de coloană scade

a) reflexia de linie are loc atunci când se întâlnește linia 1 sau m

– determină:

- creșterea liniei (dacă anterior a scăzut)
- scăderea liniei (dacă anterior a crescut)

```

if i>m           { se trec in b elementele ramase din }
then             { subsirul neîntransferat complet in b }
  for i:=j to v do
    begin k:=k+1; b[k]:=a[i] end
  else
    for j:=i to m do
      begin
        k:=k+1; b[k]:=a[j]
      end;
        { se copiaza subsirul rezultat din b in a }
      for i:=u to v do a[i]:=b[i]
    end;

begin           { corpul procedurii sortintc }
if u<v
then
begin
  m:=(u+v) div 2;       { impartire in subsiruri }
  sortintc(u,m);        { sortarea primului subsir }
  sortintc(m+1,v);      { sortarea subsirului al doilea }
  intercl               { inteclasarea sirurilor ordonate }
end
end;

Begin
Assign(f,'SIR.TXT'); Reset(f);
i:=0;             { citire sir din fisierul de intrare }
repeat
  i:=i+1;
  Read(f,a[i]);
until Eofn(f);
Close(f);
n:=i;
Write(' Sirul initial:');           { afisare sir initial }
for i:=1 to n do Write(a[i]:1:3,' ');
WriteLn;
sortintc(1,n);           { apel procedura de sortare }
Assign(g,'ORDONAT.TXT');     { memorare sir sortat in }
Rewrite(g);                { fisierul de iesire }
for i:=1 to n-1 do Write(g,a[i]:1:3,' ');
Write(g,a[n]:1:3);
Close(g);
Reset(g);
Write(' Sirul sortat:');
repeat           { afisare sir din fisierul de iesire }
  Read(g,e1);
  Write(e1:1:3,' ');
until Eofn(g);
Close(g)
End.

```

P14. Relații de echivalență

Se citesc n perechi de elemente notate cu litere mici din alfabetul latin, care se găsesc într-o relație de echivalență R . Perechile sunt separate prin caracterul 'R'. Se cere să se afișeze clasele de echivalență ale relației R ([2]).

Exemplu:

n=6
aRb
cRd
eRf
bRf
gRg
cRh

Ieșire
Clasa 1:a b e f
Clasa 2:c d h
Clasa 3:g

Soluție

Findând dată o mulțime M , spunem că R este o relație de echivalență, dacă au loc proprietățile de reflexivitate, simetrie și tranzitivitate. O clasă de echivalență este cea mai mare submulțime din M care cuprinde elemente echivalente între ele. Clasele de echivalență sunt disjuncte, iar reuniunea lor este mulțimea M ([2]).

Programul este realizat în două variante:

Varianta 1

- clasele se construiesc treptat, pe măsură ce se citesc perechile de elemente x, y ;
- se caută clasa de echivalență căreia îi aparține perechea citită x, y , adăugând această pereche clasei respective;
- dacă x aparține unei clase, iar y nu aparține claselor existente, se adaugă y clasei lui x ;
- dacă x aparține unei clase, iar y aparține altei clase, se reunesc cele două clase în prima, iar a doua se transformă în mulțimea vidă;
- dacă x, y nu aparțin nici unei clase dintre cele construite, vor forma o nouă clasă.

Program Clase_de_echivalenta_solutia_1;

```

var clasa:array[1..40] of set of 'a'..'z';
x,y:'a'..'z';
r:Char;
n,nrc,i,j,k:Integer; { n=nr numar perechi, nrc=nr numar clase }
a:array[1..2] of 1..40;
{ indicii claselor carora le apartin x si y }

```

Begin

```

Write('Introduceti numarul de perechi n='); Readln(n);
nrc:=0; { initial nici o clasă }
for i:=1 to n do
begin
  Write('Perechea x,y:'); Readln(x,r,y);
  k:=0;

```

- b) reflexia de coloană are loc dacă se întâlnește coloana i sau n
- determină:

- creșterea coloanei (dacă anterior a scăzut)
- scăderea coloanei (dacă anterior a crescut)

Creșterea, respectiv scăderea liniei/coloanei se realizează prin adunarea valorii +1 sau -1 la indicele de linie/coloană. Rezultă că simularea fenomenului de reflexie se va realiza prin schimbarea semnului valorii +1/-1 care se adună indicelui de linie sau de coloană.

```
Program Raza_de_lumina;
var m,n,i,j:Byte;      { rl=simuleaza deplasarea pe linie }
rl,rc:Shortint;        { rc=simuleaza deplasarea pe coloana }
np:0..MaxLongint;     { np=Numarul de pozitii ale razei }
Begin
  Write('m,n='); Readln(m,n);
  i:=m; j:=1;           { raza pleaca din coltul stanga jos }
  rl:=-1;
  rc:=1;
  np:=0;
  Writeln('Pozitiile prin care trece raza de lumina:');
  repeat
    Write(i,'-',j,' ');
    i:=i+rl;
    j:=j+rc;
    if i in [1,m] then rl:=-rl; { raza se reflecta de linie }
    if j in [1,n] then rc:=-rc; { raza se reflecta de coloana }
    np:=np+1;
    if np mod 10=0 then Writeln;
  until (i in [1,m]) and (j in [1,n]); { pana ajunge intr-un colt }
  Writeln(i,'-',j);
End.
```

P16. Carte

O carte pentru copii are paginile împărțite în trei părți, fiecare parte putând fi răsfoită separat. Fiecare din cele trei părți ale unei pagini cuprinde una din cele trei silabe ale unui nume de animal tipărit pe ambele fețe ale paginii (de exemplu: cro-co-dil, e-le-fant etc.). Răsfoirea separată a celor trei părți ale cărții produce, spre deliciul copiilor, nume de animale imaginare (de exemplu: cro-li-fant, e-co-dil etc.). Să se afișeze toate cuvintele care se pot citi prin această răsfoire ciudată a cărții. Se introduc: numărul de pagini ale cărții și cuvintele despărțite în silabe de pe fiecare pagină. Datele de intrare se presupun corecte ([2], Gazeta de informatică 1/91).

Soluție

Cartea se poate codifica printr-o matrice pag[3][n] care cuprinde silabele cuvântului fără linioarele de despărțire în silabe pentru a afișa cuvântul.

Programul utilizează funcția Pos pentru a depista poziția primei "-", preia cu funcția Copy prima silabă și o memorează în matrice, apoi o șterge din string-ul citit din linia de intrare. Procedează în continuare la fel și pentru restul silabelor. (Pentru aceasta a fost nevoie să se adauge o "-" la sfârșitul cuvântului.)

Program Jocul_silabelor;

```
var pag:array[1..3,1..30] of string;
  cuv:string;
  i,j,k,n,lg_silabei:Byte;
Begin
  Write('nr.pagini='); Readln(n);
  for j:=1 to n do
  begin
    Write('Cuvantul despartit in cele 3 silabe:');
    Readln(cuv);
    cuv:=cuv+'-';
    for i:=1 to 3 do
    begin
      lg_silabei:=Pos('-',cuv)-1;
      pag[i,j]:=Copy(cuv,1,lg_silabei);
      Delete(cuv,1,lg_silabei+1);
    end
  end;
  Writeln('Cuvintele care se pot forma sunt:');
  for i:=1 to n do
  begin
    for j:=1 to n do
      for k:=1 to n do
        Write(pag[1,i]+pag[2,j]+pag[3,k]+' ');
    Writeln
  end;
End.
```

P17. Caractere 'AB'

Fie un sir de $2n$ caractere cuprindând două steluțe alăturate, $n-1$ caractere egale cu A și $n-1$ caractere egale cu B. Să se afișeze toate mutările prin care cele două steluțe alăturate se înlocuiesc cu două caractere successive din sir în scopul aducerii șirului la forma cu toate A-urile la început și toate B-urile la sfârșit, fără să conțeze poziția steluțelor. Dacă problema nu are soluție, se va afișa mesajul 'Imposibil' ([2]).

Soluție

Problema nu are soluție decât dacă valorile lui n sunt mici, deoarece, în aceste cazuri, posibilitățile de mutare sunt mult mai reduse.

Exemplu:

BA**BA	Imposibil
BAB**ABA	Stelutele se mută din 4 în 1
	Stelutele se mută din 1 în 5
	Stelutele se mută din 5 în 3
	Stelutele se mută din 3 în 7
	Stelutele se mută din 7 în 5
	Stelutele se mută din 5 în 2

Algoritmul încearcă să mute caracterele A pe primele $n - 1$ poziții din sir; la fiecare pas se transferă un A , doi A sau eventual nici unul (dacă poziția respectivă este deja ocupată de A). Pentru a aduce un A pe o anumită poziție, se caută în dreapta acestea primul A . Dacă poziția sa nu coincide cu locul pe care trebuie adus, atunci, pentru a fi posibil transferul, vor trebui aduse întâi steluțele în acel loc (dacă ele nu se găsesc deja acolo). O steluță se poate aduce pe poziția respectivă printr-o singură mutare, dacă distanța dintre această poziție și aceea a steluțelor este mai mare decât 1, sau prin două mutări: mutarea lor cu două poziții la dreapta și apoi cu trei poziții la stânga, în caz contrar.

Mutarea steluțelor este realizată în procedura **Muta**; determinarea poziției steluțelor și a primului A care trebuie adus se face cu funcția **Pos**.

Sirul **poz** cuprinde pozițiile succesive ale steluțelor; cu ajutorul lui se face afișarea mutărilor, dacă problema are soluție. Evident, dacă un A nu se poate aduce pe locul său, problema nu are soluție și programul se oprește forțat. Acest caz este testat în procedura **Muta**.

Programul cuprinde și funcția booleană **Sir_ordonat** care verifică dacă sirul citit satisfac cerințele problemei, caz în care nu mai trebuie făcută nici o mutare. Această verificare este necesară, deoarece algoritmul aduce A -urile pe primele $n - 1$ poziții, dar s-ar putea ca A -urile să fie la începutul sirului, fără să ocupe primele $n - 1$ poziții.

Program Sir_A_B;

```

var a:string[50];           { a este sirul initial }
  poz:array[0..50] of Byte;
  n,n2,i,k,p:Byte;

function Sir_ordonat:Boolean;
  var b:Boolean; { sir neordonat daca dupa un B apare un A }
begin
  Sir_ordonat:=true;
  b:=false;
  { N-a aparut nici un B }

```

```

for i:=1 to n2 do
  case a[i] of
    'A':if b then begin Sir_ordonat:=false; i:=n2 end;
    'B':b:=true
  end;
end;

procedure Muta(i,j:Byte); { Se mută steluțele din poziția i }
{ in poziția j }
begin
  if (Abs(i-j)>1) and (j in [1..n2-1]) then
    { necesara în apelurile artificiului }
    begin
      if k=0 then poz[k]:=i;
      k:=k+1;
      poz[k]:=j;
      a[i]:=a[j];
      a[i+1]:=a[j+1];
      a[j]:='*';
      a[j+1]:='*'
    end
  else begin Writeln('Imposibil'); Halt end
end;

Begin
  Writeln('Dati sirul cu n-1 A-uri, n-1 B-uri ');
  Writeln(' si 2 stelute consecutive:'); Readln(a);
  n2:=Length(a); n:=n2 div 2;
  if Sir_ordonat then Write('Sir satisfac conditiile problemei-0 mutari')
  else
  begin
    k:=0;
    for p:=1 to n-1 do { p=poziția pe care trebuie adus A }
      if a[p]<>'A' then
        begin
          i:=Pos('*',Copy(a,p,50))+p-1;
          if i=p then begin Muta(i,i+2); Muta(i+2,p) end
          else
            if i>p then Muta(i,p);
            { steluțele sunt pe poziția p }
            i:=Pos('A',Copy(a,p+1,50))+p;
            if i<n2 then Muta(p,i)
            else
              begin
                Muta(p,n2-1);
                Muta(n2-1,n+1);
                Muta(n+1,n-2)
              end
        end;
    end;
  end;
end;

```

```

for i:=0 to k-1 do
begin
  Write('Stelutele se muta din ', poz[i]);
  Writeln(' in ', poz[i+1])
end;
end;
End.

```

P18. Calcule modulo 10

Se consideră fișierul INPUT.TXT care conține un număr de n linii, pe fiecare linie fiind scrise câte 7 cifre. Aceste cifre sunt elementele unui tablou bidimensional $a_{n \times 7}$.

Se cere crearea fișierului text OUTPUT.TXT care va conține tabloul bidimensional $b_{n \times 7}$ ale căruia elemente sunt determinate astfel: un element $b_{i,j}$ se calculează ca rest modulo 10 al sumei elementelor din matricea a situate pe coloana j și pe linii mai mari sau egale cu i ([2]).

Soluție

Odată cu citirea tabloului din fișierul de intrare se calculează și numărul de linii (n) ale acestuia. Algoritmul de rezolvare a problemei necesită existența unei variabile b de tip întreg în care se va calcula noua valoare a elementului din tabloul a . Rezultatul va fi afișat (după transformări) în fișierul de ieșire OUTPUT.TXT.

```

Program modulò;
var n,b:Integer;
  a:array[1..1000,1..7] of Byte;
  f,g:Text;
procedure citeste;
  var i,j:Integer;
begin
  Assign(f,'INPUT.TXT');
  Reset(f);
  i:=0;
  while not Eof(f) do
  begin
    Inc(i);
    for j:=1 to 7 do
      Read(f,a[i,j]);
    Readln(f)
  end;
  n:=i;
  Close(f)
end;
procedure determin;
  var i,j,k:Integer;

```

Diverse

```

begin
  for i:=1 to n do
    for j:=1 to 7 do
    begin
      b:=0;
      for k:=i to n do b:=b+a[k,j];
      a[i,j]:=b mod 10
    end;
  end;

procedure scrie;
  var i,j:Integer;
begin
  Assign(g,'OUTPUT.TXT'); Rewrite(g);
  for i:=1 to n do
  begin
    for j:=1 to 7 do Write(g,a[i,j], ' ');
    Writeln(g)
  end;
  Close(g)
end;

Begin
  citeste;
  determin;
  scrie
End.

```

Exemplu:

Intrare	Ieșire
2 3 7 2 1 5 4	7 8 9 0 3 2 8
1 7 2 5 5 2 1	5 5 2 8 2 7 4
3 5 6 8 9 4 1	4 8 0 3 7 5 3
1 3 4 5 8 1 2	1 3 4 5 8 1 2

P19. Ordonare

Se consideră un tablou bidimensional pătratic de dimensiune $n \times n$ de numere întregi.

Se cere să se inverseze liniile și coloanele acestuia astfel încât tabloul obținut să conțină elementele de pe diagonala principală în ordine crescătoare. Datele de intrare se citesc din fișierul de intrare INPUT.TXT sub forma următoare: pe prima linie este scris numărul n , reprezentând numărul de linii și coloane.

Pe următoarele n linii se găsesc elementele tabloului linie după linie.

Rezultatul va fi scris în fișierul OUTPUT.TXT. Pe fiecare linie a fișierului se va scrie o linie a tabloului obținut ([2]).

Exemplu:

Intrare

```
3
2 4 3
1 5 6
-1 7 8
```

Ieșire

```
-1 8 7
2 3 4
1 6 5
```

Soluție

Tabloul care trebuie ordonat este a_{ii} , $i=1..n$. Singura complicație este că nu putem interschimba numai elementele a_{ii} și $a_{i+1,i+1}$, ci trebuie să interschimbăm linia i cu linia $i+1$ și coloana i cu coloana $i+1$.

Variabila a s-a declarat ca fiind tablou cu elemente de tip *linie*, deoarece trebuie ca liniile a_i și a_{i+1} să aibă același tip cu *linie_aux*, pentru ca atribuirile să fie corecte.

Program Ordonarea_elementelor_de_pe_diagonala_principala;

```
type linie=array[1..10] of Integer;
var a:array[1..10] of linie;
    n,i,j:1..10;
    ordonat:Boolean;
    linie_aux:linie; aux:Integer;
begin
  Write('Introduceti n:'); Read(n);
  Writeln('Introduceti matricea a :');
  for i:=1 to n do { citirea matricei A }
  begin
    Write(' Linia ',i,':');
    for j:=1 to n do Read(a[i,j])
  end;
  repeat
    ordonat:=true;
    for i:=1 to n-1 do
      if a[i,i]>a[i+1,i+1]
      then
        begin
          linie_aux:=a[i];
          a[i]:=a[i+1];
          a[i+1]:=linie_aux;
          for j:=1 to n do
            begin
              aux:=a[j,i];
              a[j,i]:=a[j,i+1];
              a[j,i+1]:=aux
            end;
          ordonat:=false
        end
    until ordonat;
```

```
Write('Matricea A cu diagonala principală');
Writeln('ordonată crescător :');
for i:=1 to n do { scrierea matricei A }
begin
  for j:=1 to n do Write(a[i,j]:3); Writeln
end;
End.
```

P20. K de minus

Fie o matrice de dimensiuni $n \times m$ care poate conține doar numerele +1 sau -1. O prelucrare (mutare) aplicată acestei matrice constă în schimbarea semnului elementelor de pe o linie sau o coloană. Știind că inițial matricea conține doar elemente egale cu +1, se cere:

- a) să se determine dacă se poate trece într-o configurație care conține k elemente egale cu -1;
- b) în caz afirmativ se cer mutările necesare pentru a realiza acest lucru ([11], [1]).

Restricții

$n, m \leq 1000$; $k \leq 1000000$

Soluție

În primul rând observăm că prelucrarea de două ori a aceleiași linii sau coloane nu are nici un efect, deci este suficient să prelucrăm fiecare linie și coloană cel mult o dată.

Elementul -1 poate să apară într-o poziție oarecare (i, j) dacă linia i a fost prelucrată și coloana j nu a fost prelucrată, sau dacă linia i nu a fost prelucrată și coloana j a fost prelucrată.

Să notăm cu x numărul de linii prelucrate și cu y numărul de coloane prelucrate. Atunci numărul total de -1 va fi $x(m-y) + y(n-x)$. Dări în total trebuie să existe k elemente egale cu -1, deci numerele x, y trebuie să satisfacă ecuația $x(m-y) + y(n-x) = k$.

Problema se reduce la rezolvarea ecuației precedente având necunoscutele x și y . Dacă ecuația nu are soluție, atunci nu se poate trece într-o matrice care să conțină k elemente egale cu -1, în caz contrar vom prelucra la întâmplare x linii și y coloane.

Program KdeMinus;

```
var x,y,n,m,k:Word;
    gasit:Boolean;
```

Begin

```
Write('Introduceti numarul de linii:'); Readln(n);
Write('Introduceti numarul de coloane:'); Readln(m);
Write('Introduceti k:'); Readln(k);
gasit:=false;
x:=0;
```

```

while not gasit and (x<=n) do
begin
  y:=0;
  while not gasit and (y<=m) do
  begin
    if x*(m-y)+y*(n-x)=k then gasit:=true;
    y:=y+1
  end;
  x:=x+1
end;
if gasit
then
begin
  x:=x-1; y:=y-1;
  Writeln('Problema are solutie!');
  Write('Se actioneaza oricare ', x);
  Writeln(' linii si oricare ', y, ' coloane.')
end
else Writeln('Problema nu are solutie!');
End.

```

Intrare

n=10, m=10, k=38

Ieșire

Se actionează oricare 2 linii și oricare 3 coloane.

P21. Problema gospodinei

O gospodină trebuie să prăjească n plăcinte, având la dispoziție o tigaie în care încap cel mult k plăcinte. Știind că prăjirea unei plăcinte pe o față necesită un minut (o plăcintă are două fețe), se cere să se determine modul în care va proceda gospodina pentru a prăji toate plăcintele în timpul cel mai scurt ([1]).

Soluție

Cât timp numărul plăcintelor care au mai rămas de prăjit este mai mare sau egal cu $2*k$, vom pune în tigaie câte k plăcinte (pe prima față, iar apoi le vom întoarce pe a doua față). Dacă numărul plăcintelor rămase este între $k + 1$ (inclusiv) și $2*k$, vom aplica următoarea strategie de prăjire:

- numerotăm plăcintele rămase cu numere aparținând mulțimii $\{1, \dots, g\}$ unde g este numărul plăcintelor neprăjite;
- vom prăji plăcintele $1, \dots, k$ pe față 1;
- vom prăji plăcintele $k + 1, k + 2, \dots, g$ pe față 1 și plăcintele $1, \dots, 2*k - g$ pe față 2 (în total k plăcinte);
- vom prăji plăcintele $k + 1, k + 2, \dots, g, 2*k - g + 1, 2*k - g + 2, \dots, k$ pe față 2.

Aplicând această strategie, gospodina va reuși să prăjească ultimele plăcinte în trei minute (în loc de patru).

```

Program ProblemaGospodinei;
var n,k,i,etapa,placinte,TimpMinim:Integer;
Begin
  Write('Introduceti numarul placintelor:'); Readln(n);
  Write('Introduceti numarul maxim de placinte ');
  Write('care incap intr-o tigaie:'); Readln(k);
  if n mod k=0
  then TimpMinim:=2*(n div k)
  else
    if n<k then TimpMinim:=2
    else TimpMinim:=2*(n div k)+1;
  Writeln('Timpul minim este:',TimpMinim);
  if n<k then k:=n;
  etapa:=1;
  placinte:=0;
  while not (n>k) and (n<2*k) and (n>0) do
  begin
    Writeln('Etapa ',etapa);
    for i:=placinte+1 to placinte+k do
      Writeln(' Se prajeste placinta ',i,' pe fata 1');
    etapa:=etapa+1;
    Writeln('Etapa ',etapa);
    for i:=placinte+1 to placinte+k do
      Writeln(' Se prajeste placinta ',i,' pe fata 2');
    placinte:=placinte+k;
    etapa:=etapa+1;
    n:=n-k
  end;
  if n>0
  then
  begin
    Writeln('Etapa ',etapa);
    for i:=placinte+1 to placinte+n do
      Writeln(' Se prajeste placinta ',i,' pe fata 1');
    etapa:=etapa+1;
    Writeln('Etapa ',etapa);
    for i:=placinte+k+1 to placinte+n do
      Writeln(' Se prajeste placinta ',i,' pe fata 2');
    etapa:=etapa+1; Writeln('Etapa ',etapa);
    for i:=placinte+k+1 to placinte+n do
      Writeln(' Se prajeste placinta ',i,' pe fata 2')
  end
End.

```

Intrare

n=3 k=2

Ieșire

Timpul minim este:3 minute

Etapa 1

Se prăjeste placinta 1 pe fata 1
 Se prăjeste placinta 2 pe fata 1

Etapa 2

Se prăjeste placinta 3 pe fata 1
 Se prăjeste placinta 1 pe fata 2

Etapa 3

Se prăjeste placinta 2 pe fata 2
 Se prăjeste placinta 3 pe fata 2

P22. Trasează drum

Să se traseze (dacă este posibil) un drum de lungime n între punctul de coordonate $(0, 0)$ și punctul de coordonate (a, b) unde a și b sunt două numere naturale ($(a, b) \neq (0, 0)$) știind că oricare două puncte succesive $(x_i, y_i), (x_{i+1}, y_{i+1})$ de pe acest drum satisfac relația: $|x_i - x_{i+1}| + |y_i - y_{i+1}| = 1$.

În caz afirmativ se cer coordonatele punctelor de pe traseu ([1]).

Restricție

Drumul trece cel mult o dată prin fiecare punct.

Soluție

În cazul $a + b > n$ nu avem soluție. Dacă $a + b = n$, atunci drumul cerut se poate trasa în două etape: de la punctul $(0, 0)$ până la $(a, 0)$, iar apoi de la $(a, 0)$ până la (a, b) .

Dacă $a + b < n$, atunci există un drum care îndeplinește condițiile din enunț doar dacă $n - a - b$ este număr par. Această afirmație se demonstrează observând că numărul de segmente cu care ne îndepărăm de la drumul $(0, 0), (1, 0), \dots, (a, 0), (a, 1), \dots, (a, b)$ trebuie să fie egal cu numărul de segmente necesare pentru a reveni la acel drum.

Program TraseazaDrum;
 var a,b,n,i:Integer;

Begin

```
Write('Introduceti coordonatele punctului de sosire:');
Readln(a,b);
Write('Introduceti lungimea drumului:'); Readln(n);
if Odd(Abs(a+b-n)) or (n<a+b) then
begin
  Write('Nu exista drum care sa satisfaca conditiile ');
  Writeln('din enunt!');
end
```

```
else
begin
  Writeln('Exista drum.');
  Write('Punctele prin care trece:');
  if b=0
  then
    begin
      Write('(0,0)');
      for i:=1 to Abs(a-n) div 2 do Write('(',0,',',i,',') ');
      for i:=Abs(a-n) div 2 downto 0 do Write('(',1,',',i,',') ');
      for i:=2 to a do Write('(',i,',',0,',') ')
    end
  else
    if a=0
    then
      begin
        Write('(0,0)');
        for i:=1 to Abs(b-n) div 2 do Write('(',i,',',0,',') ');
        for i:=Abs(b-n) div 2 downto 0 do Write('(',i,',',1,',') ');
        for i:=2 to b do Write('(',0,',',i,',') ')
      end
    else
      begin
        Write('(0,0)');
        for i:=1 to a do Write('(',i,',',0,',') ');
        for i:=a+1 to a+(n-a-b) div 2 do Write('(',i,',',0,',') ');
        for i:=1 to b do Write('(',a+(n-a-b) div 2,',',i,',') ');
        for i:=a+1+(n-a-b) div 2 downto a do Write('(',i,',',b,',') ')
      end
    end;
  End.
```

Intrare

$(a, b) = (1, 2)$
 $n=5$

Ieșire

Exista drum.

Punctele prin care trece: (0,0) (1,0) (2,0) (2,1) (2,2) (1,2)

P23. Problema ciorilor

Într-un parc se află $n+1$ copaci pe care se află n ciori astfel încât în fiecare copac se găsește cel mult o cioară. În fiecare moment o cioară poate zbura de pe copacul pe care stă pe unul liber. Dându-se ordinea inițială a ciorilor să se determine ordinea în care vor zbura ciorile de pe un copac pe altul, pentru a trece într-o configurație finală, dată.

Restricții

Datele se citesc dintr-un fișier text, având următoarea structură:

- pe prima linie este scris numărul de ciori;
- pe următoarele două linii sunt scrise cele două configurații (cea inițială și cea finală) ale ciorilor ([!3], [1]).

Soluție

Vom repeta următorii pași până în momentul în care toate ciorile vor ajunge pe copaci aşa cum cere configurația finală:

- determinăm copacul pe care se află o cioară al cărei număr de ordine este diferit de cel al ciorii care se află pe același copac în configurația finală;
- căutăm ciocanul care trebuie să fie mutată în acest copac (să o notăm cu k);
- mutăm ciocanul din acest copac pe copacul liber;
- mutăm ciocanul k la locul ei pe copacul eliberat anterior.

```
Program ProblemaCiorilor;
const MaxN=1000;
type Configuratie=array[1..MaxN+1] of 0..MaxN;
var ConfigIni,ConfigFin:Configuratie;
    n,i,liber,copac:Word;
    operatie:Boolean;

procedure citeste;
    var f:Text;
begin
    Assign(f,'CIORI.IN'); Reset(f); Readln(f,n);
    for i:=1 to n+1 do Readln(f,ConfigIni[i]);
    for i:=1 to n+1 do Readln(f,ConfigFin[i]);
    Close(f)
end;

procedure ParLiber;
begin
    i:=1;
    while ConfigIni[i]<>0 do Inc(i);
    liber:=i;
end;

function DeterminaCopac(cioara:Word):Word;
    var j:Word;
begin
    j:=1;
    while ConfigIni[j]<>cioara do Inc(j);
    DeterminaCopac:=j
end;

Begin
    citeste;
    ParLiber;

```

```
operatie:=true;
while operatie do
begin
    operatie:=false;
    for i:=1 to n do
        if (ConfigIni[i]<>ConfigFin[i]) and (ConfigFin[i]<>0)
        then
            begin
                copac:=DeterminaCopac(ConfigFin[i]);
                if ConfigIni[i]<>0
                then
                    begin
                        Write('Ciocanul ',ConfigIni[i],' zboara de pe ');
                        Writeln('copacul ',i,' pe copacul ',liber);
                        ConfigIni[liber]:=ConfigIni[i];
                        ConfigIni[i]:=0;
                        liber:=i;
                        operatie:=true
                    end;
                Write('Ciocanul ',ConfigIni[copac],' zboara de pe ');
                Writeln('copacul ',copac,' pe copacul ',liber);
                ConfigIni[liber]:=ConfigIni[copac];
                ConfigIni[copac]:=0;
                liber:=copac;
                operatie:=true
            end;
        end;
    End.
```

Intrare

```
5
1 2 0 3 4
2 1 4 3 0
```

Ieșire

```
Ciocanul 1 zboara de pe copacul 1 pe copacul 3
Ciocanul 2 zboara de pe copacul 2 pe copacul 1
Ciocanul 1 zboara de pe copacul 3 pe copacul 2
Ciocanul 4 zboara de pe copacul 5 pe copacul 3
```

P24. Configurații

Fie un vector de lungime n care conține elemente din mulțimea {0, 1}. Acționarea unei poziții i are ca efect schimbarea conținutului ($0 \leftrightarrow 1$) tuturor pozițiilor ale căror valoare este multiplu de i (inclusiv poziția i).

Dându-se două configurații, una inițială și alta finală, să se determine dacă se poate trece din una în alta, după acționarea unui număr nespecificat de poziții. În caz afirmativ să se afișeze pozițiile acționate ([1]).

Restricții

Datele se citesc dintr-un fișier de intrare având următoarea structură:

- pe prima linie este scris numărul $n \leq 10000$;
- pe următoarele două linii sunt scrise configurația inițială și cea finală (elementele vectorilor fiind despărțite prin spații).

Soluție

Observăm că acționarea unei poziții i nu afectează starea pozițiilor având numere de ordine mai mici decât i . Din această observație rezultă că din oricare configurație se poate trece în orice altă configurație.

Mutările care trebuie efectuate pentru a se trece dintr-o configurație în alta se determină parcursând sirurile de la poziția 1 spre poziția n și acționând pozițiile prin care configurația curentă este diferită de cea finală.

```
Program Multiplii;
const MaxN=1000;
var ConfIni,ConfFin:array[1..MaxN] of 0..1;
n,i,j:Word;

procedure citeste;
  var f:Text;
begin
  Assign(f,'VECTOR.IN'); Reset(f); Readln(f,n);
  for i:=1 to n do Readln(f,ConfIni[i]);
  for i:=1 to n do Readln(f,ConfFin[i]);
  Close(f);
end;

Begin
  citeste;
  for i:=1 to n do
    if ConfIni[i]<>ConfFin[i]
    then
      begin
        Writeln('Se actioneaza pozitia ',i);
        j:=i;
        while (j<=n) do
          begin
            ConfIni[j]:=1-ConfIni[j];
            j:=j+1
          end
      end;
end;
End.
```

Intrare

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

Ieșire

```
- Se actioneaza pozitia 2
  Se actioneaza pozitia 4
  Se actioneaza pozitia 5
  Se actioneaza pozitia 6
  Se actioneaza pozitia 7
```

P25. Cod de lungime minimă

Fiecarei număr natural n îi atașăm un sir de caractere format doar cu elemente din mulțimea $\{i, d, m\}$, acestea reprezentând o operație care se realizează asupra unui număr curent (în sensul de intermediar pe calea obținerii lui n din 0):

- mărirea numărului curent cu o unitate (acestei operații i se atașează caracterul i);
- decrementarea numărului cu o unitate (acestei operații i se atașează caracterul d);
- înmulțirea cu 10 a numărului curent (acestei operații i se atașează caracterul m).

Dându-se un număr natural n ; să se genereze cel mai scurt cod care poate fi atașat acestui număr. Inițial se pleacă de la numărul 0 și trebuie să se obțină numărul n , singurele operații permise fiind cele specificate.

Exemplu:

Numărului 16 îi este atașat codul $dddmii$, (codul îl citim de la coadă spre început):

S-a pornit de la 0, s-au adăugat două unități (secvența ii), s-a înmulțit cu 10 (m), în final s-au scăzut patru unități (secvența $ddad$) ($[I]$).

Soluție

Codul atașat fiecarui număr îl construim plecând de la cifra cea mai semnificativă spre cele mai nesemnificative.

Dacă pe poziția curentă este o cifră mai mică decât 5 și dacă poziția curentă în numărul dat este diferită de 1, atunci inserăm pe poziția 1 a codului curent caracterul m , apoi inserăm în dreapta lui o secvență formată doar din caractere i de lungime egală cu cifra înscrisă pe poziția curentă, în caz contrar inserăm pe poziția 1 o secvență formată doar din caractere i de lungime egală cu cifra înscrisă pe poziția curentă.

Dacă pe poziția curentă este o cifră mai mare sau egală cu 5, atunci este mai eficient să adăugăm o unitate la numărul curent, să-l înmulțim cu 10, apoi să scădem din el un număr de unități egal cu $10 - \text{cifra curentă}$ (exemplu: pentru $n=16$ se observă că este mai eficient să obținem numărul 20 și apoi să scădem din el 4 unități, decât să obținem 10 la care să adăugăm 6 unități).

Rezultatul îl citim de la dreapta spre stânga.

```
Program SirDeLungimeMinima;
var s,x:string;
n,i,j:Longint;
Begin
  Write('n='); Readln(n);
  s:=''; Str(n,x);
  for i:=1 to Length(x) do
    if x[i]<'5'
    then
      begin
        if i<>1 then Insert('m',s,i);
        i:=i+1
      end;
    end;
  Writeln(s);
End.
```

```

    for j:=1 to Ord(x[i])-Ord('0') do Insert('i',s,1)
  end;
else
begin
  Insert('i',s,1); Insert('m',s,1);
  for j:=9 downto Ord(x[i])-Ord('0')+1 do Insert('d',s,1);
end;
Write('Codul asociat:',s)
End.

```

Intrare

n=123
n=639

Ieșire

Codul asociat: iiimiiimi
Codul asociat: dmiiiiimddddmi**P26. Problema macaralei**

Pe o linie de cale ferată se află n vagoane (dispuse unul după altul), numerotate cu valori distincte din mulțimea $\{1 \dots n\}$. O macara specială poate ridica oricâte vagoane, apoi le aşează (în partea dreaptă) la sfârșitul șirului de vagoane.

Datorită faptului că linia de cale ferată se află pe un plan înclinat, având parteă mai înaltă în dreapta macaralei, vagoanele, prin alunecare de la dreapta spre stânga, în nouă ordine existentă după operația executată de macara, vor ajunge din nou unul lângă altul.

Dându-se ordinea inițială a vagoanelor să se determine (dacă este posibil) numărul minim de operații pe care trebuie să le efectueze macaraua, pentru ca în final vagoanele să fie în ordine crescătoare după numărul lor de ordine.

Restrictions.

Datele se citesc dintr-un fișier text MACARA.IN, având următoarea structură:

- pe prima linie este scris numărul de vagoane;
- pe a doua linie este precizată ordinea inițială a vagoanelor ([1]).

Soluție

Pe mulțimea tuturor permutărilor introducem o relație de ordine totală, și anume relația de *ordine lexicografică*. În aceste condiții permutarea $(1, 2, 3, \dots, n)$ va avea numărul de ordine 1, iar permutarea $(n, n-1, \dots, 3, 2, 1)$ va avea numărul de ordine $n!$. De asemenea, oricare două permutări distincte vor avea atașate numere de ordine distincte, iar oricărui număr de ordine din mulțimea $\{1, \dots, n!\}$ îi este atașată o singură permutare.

Considerăm ordinea inițială a vagoanelor ca fiind o permutare a mulțimii $\{1, \dots, n\}$. Trebuie să determinăm numărul minim de mutări necesare pentru a trece din permutarea indușă de ordinea inițială a vagoanelor, în permutarea identică.

Observăm că din orice permutare se poate trece într-o altă permutare având numărul de ordine mai mic. Următorul algoritm se bazează pe această observație și pe restricția: la fiecare pas se va executa o mutare care determină o permutare având numărul de ordine cel mai mic posibil.

Dacă secvența care începe cu vagonul căruia îi este atașat numărul 1 și se termină cu al n -lea vagon este o permutare, atunci vom ordona mai întâi această permutare, apoi vom trece la permutarea curentă.

Exemplu:

Având permutarea $(5 \ 1 \ 2 \ 4 \ 3)$, vom ordona mai întâi permutarea $(1 \ 2 \ 4 \ 3)$, apoi îl vom trece și pe cel de al cincilea vagon la sfârșit.

```

Program ProblemaMacaralei;
const MaxN=10;
var a,mutate:array[1..MaxN] of 1..MaxN;
  n,i,j,incep,poz,poz1:Word;
  mult:set of 1..MaxN;

procedure citeste;
  var f:Text;
begin
  Assign(f,'MACARA.IN'); Reset(f); Readln(f,n);
  for i:=1 to n do Read(f,a[i]);
  Close(f);
end;

function Caută(vagon:Byte):Byte;
  var c:Byte;
begin
  c:=1; while a[c]<>vagon do Inc(c);
  Caută:=c
end;

Begin
  citeste;
  for i:=2 to n do
  begin
    poz:=Caută(i); poz1:=Caută(1); mult:=[];
    for j:=poz1 to n do mult:=mult+[a[j]];
    if (mult=[1..n-poz1+1]) and (poz>poz1)
    then
    begin
      incep:=Caută(i-1);
      if poz-incep>1
      then
      begin
        Write('Se mută ',poz-incep-1,' vagoane;');
        Writeln(' începând cu poziția ',incep+1)
      end;
    end;
  end;
end.

```

```

for j:=incep+1 to poz-1 do mutate[j-incep]:=a[j];
for j:=poz to n do a[incep+j-poz+1]:=a[j];
for j:=1 to poz-incep-1 do a[n-poz+incep+j+1]:=mutate[j];
end
else
  if a[1]<>1
  then
    begin
      Write('Se mută ',poz1-1,' vagoane;');
      Writeln(' incepând cu poziția 1');
      for j:=1 to poz1-1 do mutate[j]:=a[j];
      for j:=poz1 to n do a[j-poz1+1]:=a[j];
      for j:=1 to poz1-1 do a[n-poz1+j+1]:=mutate[j];
      i:=i-1
    end
  end
End.

```

Intrare

7
7 6 1 4 3 2 5

Ieșire

Se mută 2 vagoane incepând cu poziția 4
 Se mută 2 vagoane incepând cu poziția 5
 Se mută 1 vagoane incepând cu poziția 6
 Se mută 2 vagoane incepând cu poziția 1
 Se mută 1 vagoane incepând cu poziția 6

P27. Ocupă suprafață

O suprafață dreptunghiulară cu aria $n \times m$ este împărțită în $n \times m$ pătrate de latură 1. Să se acopere această suprafață cu figuri de forma:



astfel încât fiecare pătrat al dreptunghiului să fie acoperit de exact un pătrătel.

Restriții

n și m sunt multipli de 4 ([1]).

Soluție

Datorită faptului că n și m sunt multipli de 4, vom împărți suprafața inițială în pătrate având latura de dimensiune 4. Apoi fiecare astfel de pătrat este acoperit de figuri în felul următor:

1 1 1 4
 2 1 4 4
 2 2 3 4
 → 2 3 3 3

```

Program OcupaSuprafata;
const MaxN=200;
var a:array[1..MaxN..1..MaxN] of 1..4;
n,m,i,j:Byte;
f:Text;
Begin
repeat
  Write('n:'); Readln(n);
until n mod 4=0;
repeat
  Write('m:'); Readln(m);
until m mod 4=0;

for i:=1 to n do
  for j:=1 to m do
    if i mod 4=0
    then
      if j mod 4=1 then a[i,j]:=2
                    else a[i,j]:=3
    else
      if i mod 4=1
      then
        if j mod 4=0 then a[i,j]:=4
                      else a[i,j]:=1
      else
        if i mod 4=2
        then
          if j mod 4=1
          then a[i,j]:=2
          else
            if j mod 4=2 then a[i,j]:=1
                          else a[i,j]:=4
        else
          if j mod 4=0
          then a[i,j]:=4
          else
            if j mod 4=3 then a[i,j]:=3
                          else a[i,j]:=2;
Assign(f,'supraf.out'); Rewrite(f);
for i:=1 to n do
begin
  for j:=1 to m do Write(f,a[i,j],',');
  Writeln(f)
end;
Close(f);
Write('Datele au fost depuse in fisierul de ieșire.');
End.

```

Intrare

n=8, m=12

Ieșire

```

1 1 1 4 1 1 1 4 1 1 1 4
2 1 4 4 2 1 4 4 2 1 4 4
2 2 3 4 2 2 3 4 2 2 3 4
2 3 3 3 2 3 3 3 2 3 3 3
1 1 1 4 1 1 1 4 1 1 1 4
2 1 4 4 2 1 4 4 2 1 4 4
2 2 3 4 2 2 3 4 2 2 3 4
2 3 3 3 2 3 3 3 2 3 3 3

```

*** P28. Grafuri idempotente**

Să se calculeze numărul grafurilor orientate cu n vârfuri care au următoarele proprietăți:

- într-un vârf poate să nu vină și poate să nu plece nici un arc;
- dintr-un vârf poate pleca cel mult un arc;
- dacă dintr-un vârf pleacă un arc atunci în acel vârf nu mai poate veni nici un arc;
- dacă într-un vârf vine cel puțin un arc atunci din acel vârf nu mai poate pleca nici un arc. ([1]).

Soluție

Dacă reprezentăm graful sub forma unui vector f în care $(i, f(i))$ este muchie, atunci problema este echivalentă cu determinarea numărului *funcțiilor idempotente* definite pe o mulțime $\{1, \dots, n\}$ și cu valori în mulțimea $\{1, \dots, n\}$. Reamintim faptul că o funcție este idempotentă dacă $f(f(x)) = f(x)$ oricare ar fi x din domeniu.

Să calculăm numărul acestor funcții. Să notăm cu \mathcal{M} mulțimea elementelor i , cu proprietatea că $f(i) = i$. Pentru ca o funcție să fie idempotentă trebuie ca \mathcal{M} să aibă cel puțin un element, iar pentru orice element j care nu aparține lui \mathcal{M} trebuie ca $f(j)$ să aparțină lui \mathcal{M} . În aceste condiții pentru $\text{card}(\mathcal{M}) = k$ unde $1 \leq k \leq n$, avem $C_n^k \times k^{n-k}$, funcții idempotente, deoarece cele k elemente le putem alege în C_n^k moduri distincte, iar pe celelalte $n - k$ poziții trebuie să distribuim cel mult k numere distincte, ceea ce este echivalent cu determinarea numărului funcțiilor definite pe un co-domeniu cu k elemente și cu domeniul având $n - k$ elemente.

Program Grafuri Idempotente;

```

const MaxN=13;
var n,L,suma:Longint;
function Combinari(m,k:Byte):Longint;
var c:array[1..MaxN,0..MaxN] of Longint;
i,j:Byte;

```

```

begin
  c[1,0]:=1; c[1,1]:=1;
  for i:=2 to n do
    begin
      c[i,0]:=1;
      for j:=1 to n-1 do c[i,j]:=c[i-1,j]+c[i-1,j-1];
      c[i,i]:=1;
    end;
  Combinari:=c[m,k]
end;

Begin
  Write('n='); Readln(n);
  suma:=0;
  for L:=1 to n do
    suma:=suma+Combinari(n,L)*Trunc(exp((n-L)*Ln(L)));
  Writeln('Numarul grafurilor este:', suma)
End.

```

Intrare

n=5

Ieșire

196

*** P29. Problema secvenței**

Se dă un număr natural x care are doar cifre distincte. Se cere să se determine numărul de apariții ale lui x ca subsecvență de cifre în numerele din intervalul $(b, a]$, unde a și b sunt două numere naturale de formă $10...0$ ([1]).

Soluție

Vom determina de câte ori apare x ca subsecvență în numerele de la 1 la a (inclusiv), apoi din acest număr vom scădea de câte ori apare x ca subsecvență în numerele de la 1 la b (inclusiv).

Să construim un vector v , în care pe poziția k vom reține numărul de apariții ale secvenței x în numerele din intervalul $19\dots9$ (unde cifra 9 apare de k ori). Modul de calcul al elementelor acestui vector este:

- pentru k mai mic decât lungimea lui x , evident $v(k) = 0$;
- $v(k) = 9 * v(k - 1) + 10^{k-1-lungimea lui x} + v(k - lungimea lui x)$.

Formula de mai sus se deduce astfel: $9 * v(k - 1)$ reprezintă numărul de apariții ale secvenței x în numerele de formă $100\dots0, 200\dots0, 300\dots0, \dots, 900\dots0$, mai puțin numărul care are prima cifră identică cu prima cifră a lui x , iar partea a doua a formulei reprezintă numărul de apariții ale secvenței x în numărul care începe cu prima cifră a lui x .

În cazul în care x este de formă $10\dots0$ vom mai adăuga o unitate la numărul de apariții ale secvenței x .

```

Program ProblemaSubsecventei;
var a,b,x:Longint;
v:Longint;
coder,i:Integer;
function Numarari(q:Longint):Longint;
var nr:Longint;
numar,coder:Integer;
v:array[0..10] of Longint;
sirl,sir2:string;
begin
for i:=0 to Trunc(Ln(x)/Ln(10)) do v[i]:=0;
for i:=Trunc(Ln(x+1)/Ln(10))+1 to Trunc(Ln(q+1)/Ln(10)) do
v[i]:=9*v[i-1]+Trunc(exp((i-1-Trunc(Ln(x+1)/
Ln(10)))*Ln(10)))+v[i-Trunc(Ln(x+1)/Ln(10))];
nr:=v[Trunc(Ln(q+1)/Ln(10))];
Str(q,sirl); Str(x,sir2);
while Pos(sir2,sirl)<>0 do
begin
Val(copy(sirl,Pos(sir2,sirl)+Trunc(Ln(x+1)/
Ln(10)),255),numar,coder);
nr:=nr+numar*10;
Delete(sirl,Pos(sir2,sirl),Trunc(Ln(x+1)/Ln(10)));
end;
Numarari:=nr;
end;
Begin
Write('a='); Readln(a);
Write('b='); Readln(b);
Write('x='); Readln(x);
Writeln('Numarul de aparitii=',Numarari(a)-Numarari(b));
End.

```

Intrare
a=100, b=1, x=12

Ieșire
Numarul de aparitii:1

P30. Matrice Fibonacci

Se dă două matrice pătratice A și B, având următoarele forme:

$$A = \begin{pmatrix} 5 & 8 \\ 5 & 8 \end{pmatrix} \quad B = \begin{pmatrix} 2 & 3 \\ 3 & 5 \end{pmatrix}$$

Se repetă următoarele operații de k ori:

$$C = A \times B$$

$$A = C$$

Să se determine forma finală a matricei A, fără a efectua operații de înmulțire a matricelor ([1]).

Soluție

Mai întâi observăm că matricele A și B conțin termeni ai șirului Fibonacci, definit prin $F(0) = F(1) = 1, F(n) = F(n - 1) + F(n - 2)$.

Deci înmulțind matricea A cu matricea B, obținem o matrice

$$C = \begin{pmatrix} F(4) \times F(2) + F(5) \times F(3) & F(4) \times F(3) + F(5) \times F(4) \\ F(4) \times F(2) + F(5) \times F(3) & F(4) \times F(3) + F(5) \times F(4) \end{pmatrix}$$

Mai știm, de asemenea, că numerele Fibonacci satisfac relația :

$$F(n+m) = F(n-1) \times F(m) + F(n) \times F(m+1) \quad (1)$$

Matricea A fiind inițial de forma:

$$A = \begin{pmatrix} F(n) & F(n+1) \\ F(n) & F(n+1) \end{pmatrix}, \text{ unde } n = 5, \text{ iar matricea } B \text{ fiind inițial de forma:}$$

$$B = \begin{pmatrix} F(m) & F(m+1) \\ F(m+1) & F(m+2) \end{pmatrix}, \text{ unde } m = 2. \text{ Rezultă (înănd cont de relația (1)) că matricea}$$

C după o operație va fi de forma:

$$C = \begin{pmatrix} F(n+m) & F(n+m+1) \\ F(n+m) & F(n+m+1) \end{pmatrix}, \text{ iar după } k \text{ operații va fi de forma:}$$

$$C = \begin{pmatrix} F(n+k \times m) & F(n+k \times m+1) \\ F(n+k \times m) & F(n+k \times m+1) \end{pmatrix}$$

Program Fibonacci;
const n=5; m=2;
var f:array[0..10000] of Longint;
poz,k,i:Word;

Begin
Write('Introduceti k:'); Readln(k);
poz:=n+k*m+1; f[0]:=1; f[1]:=1;
for i:=2 to poz do f[i]:=f[i-1]+f[i-2];
Writeln('Matricea A devine:');
Write(f[poz-1]:10);
Writeln(f[poz]:10);
Write(f[poz-1]:10);
Writeln(f[poz]:10)
End.

Intrare Ieșire
k=5 987 1597
987 1597

P31. Numerele lui Stirling

Un concurs este câștigat de n persoane. Sponsorii pun la dispoziția organizatorilor premii de k tipuri (din fiecare tip existând suficiente premii). Toate premiile au aceeași valoare. Fiecare persoană primește un singur premiu (un singur tip de premiu din cele k existente), iar din fiecare tip de premiu este premiată cel puțin o persoană.

Se cere să se determine numărul de posibilități de a premia cele n persoane ([1]).

Soluție

Observăm că cele k tipuri de premii determină o partitie cu k clase a mulțimii câștigătorilor. Două persoane fac parte din aceeași clasă dacă și numai dacă au primit același tip de premiu.

Numărul partitiilor unei mulțimi cu n elemente în k clase poartă denumirea de *numerele lui Stirling de speță a doua* (și se notează cu $S(n, k)$).

Aceste numere se calculează astfel:

- În primul rând observăm că dacă $k > n$ atunci $S(n, 1) = 1$, și $S(n, k) = 0$;
- dacă pornim de la mulțimea celor $S(n - 1, k - 1)$ partiții ale unei mulțimi cu $n - 1$ elemente în $k - 1$ clase, vom putea obține $S(n - 1, k - 1)$ partiții ale unei mulțimi cu n elemente în k clase, adăugând fiecărei partiții o nouă clasă alcăuită din al n -lea element. Totodată, dacă luăm în considerare o partiție a unei mulțimi din $n - 1$ elemente în k clase, putem adăuga al n -lea element la clasele deja existente în k moduri.

Din aceste două moduri de a obține $S(n, k)$, deducem că

$$S(n, k) = S(n - 1, k) + k \times S(n - 1, k).$$

Vom construi o matrice S în care $S(i, j)$ reprezintă numărul partiților unei mulțimi cu i elemente în j clase.

```
Program NumereleStirling;
const MaxN=100;
var s:array[1..MaxN,1..MaxN] of Longint;
    n,k,i,j:Byte;
begin
  Write('Introduceti numarul de persoane:'); Readln(n);
  Write('Introduceti numarul de tipuri de premii:');
  Readln(k); s[1,1]:=1;
  for i:=2 to k do s[1,i]:=0;
  for i:=2 to n do s[i,1]:=1;
  for i:=2 to n do
    for j:=2 to k do s[i,j]:=s[i-1,j-1]+j*s[i-1,j];
  Write('Numarul de posibilitati de impartire a premiilor:');
  Writeln(s[n,k])
end.
```

Intrare Iesire

$n=5, k=3$ Numarul de posibilitati de impartire a premiilor:25

P32. Funcții surjective

La un magazin există k tipuri de produse. O persoană cumpără n produse astfel încât din fiecare tip de produs a luat cel puțin un exemplar. Să se determine numărul de moduri în care și poate efectua cumpărăturile ([1]).

Soluție

Problema se reduce la determinarea numărului funcțiilor surjective definite pe domeniul având cardinalitate n și cu valori în codomeniul de cardinalitate k . Numărul acestor funcții este: $k^n - C_k^1 \times (k-1)^n + C_k^2 \times (k-2)^n - \dots - (-1)^{k-1} C_k^{k-1}$.

Nu vom insistă asupra demonstrării formulei de calcul a acestui număr, deoarece este binecunoscută.

O altă metodă de calcul a numărului funcțiilor surjective este legată de *numerele lui Stirling de speță a doua*, numere prezentate în problema precedentă, și anume: $S(n, k) = (1/k!) \times s(n, k)$, unde prin $s(n, k)$ am notat numărul funcțiilor surjective.

Demonstrarea acestei afirmații se face pe baza a două observații:

- 1) pentru orice funcție surjectivă $f: X \rightarrow Y$ există o partiție a mulțimii X , astfel: $(f^{-1}(y_1)) \cup (f^{-1}(y_2)) \cup \dots$
- 2) într-o partiție nu contează ordinea termenilor și de aceea $k!$ funcții vor genera aceeași partiție.

```
Program FunctiiSurjective;
const MaxN=100;
var c:array[0..MaxN] of Longint;
    n,k,i,j:Byte;
    nr:Longint;

procedure Combinari(g:Byte);
begin
  c[0]:=1;
  for i:=1 to g do c[i]:=((g-i+1)*c[i-1]) div i
end;

function putere(g,h:Byte):Longint;
var p:Longint;
begin
  p:=1;
  for j:=1 to h do p:=p*g;
  putere:=p
end;

begin
  Write('Introduceti numarul de produse:'); Readln(n);
  Write('Introduceti numarul tipurilor de produse existente:');
  Readln(k);
  combinari(k);
end.
```

```

nr:=0;
for i:=0 to k do
  if odd(i)
    then nr:=nr-c[i]*putere(k-i,n)
    else nr:=nr+c[i]*putere(k-i,n);
Write('Numarul de moduri in care pot fi cumparate cele ',n,' produse: ',nr)
End.

```

Intrare

n=3 k=2

Ieșire

Numarul de moduri in care pot fi cumparate cele 3 produse:6

P33. Problema secvenței

Fie un vector de lungime n , care inițial are toate elementele nule. Vom mări cu o unitate valoarea elementelor unei secvențe de lungime k , operație care se va repeta de un număr nespecificat de ori.

Dându-se un vector care are componentele nenegative, să se determine dacă este posibil sau nu ca acesta să fie obținut dintr-un vector inițial nul, după aplicarea unui număr nespecificat de operații descrise anterior.

Restricții

Datele se citesc dintr-un fișier text, având următoarea structură:

- pe prima linie este scrisă lungimea vectorului (n);
- pe a doua linie este scrisă lungimea secvenței (k);
- pe a treia linie este dat sirul de elemente nenegative ([1]).

Soluție

Observăm că o condiție necesară ca problema să admită soluție este ca suma elementelor vectorului dat să fie un multiplu de k . Această condiție nu este însă și suficientă. De aceea, vom aplica următorii pași până în momentul în care toate elementele vectorului vor deveni nule sau nu mai există o secvență de lungime k având toate elementele pozitive: determinăm cel mai din stânga element pozitiv al vectorului. Pentru ca problema să aibă soluție trebuie să existe o secvență de lungime k (cu elemente pozitive) care începe cu poziția respectivă. În cazul în care o astfel de secvență nu există, se va afișa mesajul: 'Nu există soluție' și executarea programului va fi întreruptă. În caz contrar efectuăm operația inversă celei din enunț, iar dacă la un moment dat toate elementele vectorului au devenit nule, se va afișa mesajul: 'Există soluție'.

```

Program No;
const MaxN=1000;
var a:array[1..MaxN] of Word;
n,k,i,nenul:Word;
f:Text;

Begin
  Assign(f,'SECVENTA.IN'); Reset(f);
  Readln(f,n); Readln(f,k);
  for i:=1 to n do Read(f,a[i]);
  Close(f);
  nenul:=1;
  while nenul<=n+1 do
    begin
      while (a[nenul]=0) and (nenul<=n+1) do Inc(nenul);
      if nenul>=n+1
        then
          for i:=0 to k-1 do
            if a[i+nenul]>0
              then Dec(a[i+nenul])
            else
              begin Writeln('Nu există soluție.'); Readln; Halt end
        end;
      Writeln('Există soluție.');
    end.
End.

```

Intrare

5 Există soluție.

3
0 1 1 2 1 1**Intrare**5 Nu există soluție.
3
0 1 1 2 1 0**P34. Sir de lungime maximă**

Se dă un număr natural n . Spunem că două numere naturale x și y se află în relația x dacă și numai dacă cel puțin unul dintre numerele x și y este divizor al lui n .

Să se construiască un sir a de lungime maximă având următoarele proprietăți:

- $a_1=1$, iar ultimul termen al sirului este n ;
- oricare doi termeni consecutivi ai sirului trebuie să se găsească în relația x ;
- sirul trebuie să aibă lungime maximă.

Soluție

```

Begin
    citeste;
    DeterminaComponente;
    if not Corect
        then Writeln('Nu se poate trece!')
    else
        begin
            for i:=1 to n-1 do
                for j:=i+1 to n do
                    if ConfFin[i,j] and not ConfIni[i,j]
                    then Writeln('Adauga muchia:',i,',',j);
            for i:=1 to n-1 do
                for j:=i+1 to n do
                    if ConfIni[i,j] and not ConfFin[i,j]
                    then Writeln('Elimina muchia:',i,',',j)
        end
End.

```

Intrare

7
0 1 0 0 0 0 0
1 0 1 0 0 0 0
0 1 0 1 0 0 0
0 0 1 0 1 0 0
0 0 0 1 0 1 0
0 0 0 0 1 0 1
0 0 0 0 0 1 0
0 0 1 0 0 0 0
0 0 0 1 0 0 1
1 0 0 0 1 0 0
0 1 0 0 0 1 0
0 0 1 0 0 0 1
0 0 0 1 0 0 0
0 1 0 0 1 0 0

Ieșire

Adauga muchia:1,3
Adauga muchia:2,4
Adauga muchia:2,7
Adauga muchia:3,5
Adauga muchia:4,6
Adauga muchia:4,7
Elimina muchia:1,2
Elimina muchia:2,3
Elimina muchia:3,4
Elimina muchia:4,5
Elimina muchia:5,6
Elimina muchia:6,7

V. EXPRESII ARITMETICE

P1. Expresie generalizată

Se dă o expresie aritmetică, care conține operatori de diferite priorități. Aceștia sunt identificați prin caracterele speciale (+, -, *, &, ^, %, \$, #, @, !, ~, |, \). Fiecărui astfel de operator îi se atașează o prioritate care este un număr întreg cuprins între 0 și 255. Operanții sunt litere mici. Expresia poate conține și paranteze. Să se construiască arborele atașat expresiei date, apoi să se afișeze acesta în inordine.

Restriții

Datele de intrare se citesc dintr-un fișier text cu următoarea structură:

- pe prima linie se află numărul de operatori conținuți în expresie;
- pe următoarele linii se află perechi de forma operator prioritate;
- pe ultima linie se află expresia.

Soluție

Luând cazul particular al expresiilor care conțin doar paranteze și operatorii +, -, *, /, observăm că o astfel de expresie este de fapt o sumă de termeni. Un termen la rândul lui poate fi privit ca fiind format dintr-un produs de factori. Factorul, în cazul nostru este o literă mică.

Vom începe parcurgerea expresiei de la stânga la dreapta și o vom împărți într-o sumă (diferență) de termeni. Apoi fiecare termen, îl vom parcurge de la stânga la dreapta și îl vom descompune într-un produs (sau raport) de factori. În final avem o procedură factor care construiește efectiv un nod terminal al arborelui.

Dacă avem de-a face cu o expresie generalizată, atunci într-o primă etapă o vom descompune într-o mulțime de subexpresii între care există doar operatori de prioritate minimă. Apoi fiecare subexpresie rezultată o vom descompune, la rândul ei, în alte subexpresii între care există doar operatori de prioritate mai mare decât operatorii din subexpresiile inițiale. Vom continua acest procedeu până când chtinem descompunerea în subexpresii între care există doar operatori de prioritate maximă.

Analizând posibilitățile de implementare, observăm că este destul de dificil și probabil costisitor să memorăm toate subexpresiile între care există doar un anumit tip de operatori. De aceea vom construi arborele printr-o singură parcurgere a expresiei. Pro-

```

Program JoculComponentelorConexe;
const MaxN=10;
type componenta=set of 1..MaxN;
matrice=array[1..MaxN,1..MaxN] of Boolean;
var componente:array[1..MaxN] of componenta;
ConfIni,ConfFin,a:matrice;
card:array[1..MaxN] of 1..MaxN;
n,i,j,k,nrc,e:Byte;
toate:componenta;

procedure citeste;
var f:Text;
begin
Assign(f,'CONEX.IN'); Reset(f); Readln(f,n);
for i:=1 to n do
begin
for j:=1 to n do
if not SeekEoln(f)
then begin Read(f,e); ConfIni[i,j]:=e=1 end;
Readln(f);
end;
Readln(f);
for i:=1 to n do
begin
for j:=1 to n do
if not SeekEoln(f)
then
begin
Read(f,e); ConfFin[i,j]:=e=1;
end;
Readln(f);
end;
Close(f);
end;

procedure InchidereTranzitiva;
begin
for k:=1 to n do
for i:=1 to n do
for j:=1 to n do
a[i,j]:=(a[i,j]) or ((a[i,k]) and (a[k,j]));
end;

procedure DeterminaComponente;
begin
a:=ConfIni;
InchidereTranzitiva;
nrc:=0;
i:=1;

```

```

toate:=[1..n];
while toate<>[] do
begin
Inc(nrc);
componente[nrc]:=[i];
toate:=toate-[i];
for j:=i+1 to n do
if a[i,j]
then
begin
componente[nrc]:=componente[nrc]+[j];
toate:=toate-[j];
end;
i:=2;
while not (i in toate) and (i<=n) do i:=i+1;
end;
function Cardinal(q:Byte):Byte;
var nr:Byte;
begin
nr:=0;
for k:=1 to n do
if k in componente[q] then nr:=nr+1;
Cardinal:=nr;
end;

procedure DeterminaCardinal;
begin
for i:=1 to nrc do card[i]:=Cardinal(i);
end;

function AceeasiComponenta(q,w:Byte):Boolean;
begin
for k:=1 to nrc do
AceeasiComponenta:=(q in componente[k]) and
(w in componente[k]) and (card[k]>2);
end;

function Corect:Boolean;
begin
DeterminaCardinal;
for i:=1 to n do
for j:=1 to n do
if (ConfIni[i,j] and not ConfFin[i,j]) or
(ConfFin[i,j] and not ConfIni[i,j]) and
not AceeasiComponenta(i,j);
then begin Corect:=false; Exit end;
Corect:=true
end;

```

$L_i = j$ dacă nodul i face parte din componenta numărului j . Apoi se determină primele două componente cu număr maxim de noduri. Muchia dorită se obține unind două noduri din cele două componente determinate.

Program conex;

```

var a:array[1..2,1..100] of Integer;
    L:array[1..100] of Integer;
    i,j,n,m,c,cl:Integer;
    f:Text;

procedure citeste;
begin
    Assign(f,'INPUT.TXT'); Reset(f); Readln(f,n); m:=0;
    while not Eof(f) do
        begin Readln(f,i,j); Inc(m); a[1,m]:=i; a[2,m]:=j end
    end;

procedure conexitate;
var p,s,q,max,ind,nr,k,v:Integer;
begin
    for i:=1 to n do L[i]:=i;
    for i:=1 to m do
        begin
            p:=a[1,i]; q:=a[2,i];
            if L[p]<L[q]
            then
                for j:=1 to n do
                    if L[j]=L[q]
                    then L[j]:=L[p]
                    else
                        for j:=1 to n do
                            if L[j]=L[p] then L[j]:=L[q]
            end;
            for k:=1 to 2 do
                begin
                    max:=0;
                    for i:=1 to n do
                        begin
                            if L[i]<>0
                            then
                                begin
                                    nr:=0; p:=L[i];
                                    for j:=1 to n do
                                        if L[j]=p then Inc(nr);
                                    if max<nr
                                    then begin max:=nr; ind:=i; v:=L[i] end
                                end
                            end
                        end;
        end;
end;

```

```

Write(ind,' ');
for i:=1 to n do
    if L[i]=v then L[i]:=0;
end;
end;

Begin
    citeste;
    conexitate
End.

```

P56. Jocul componentelor conexe

Se dă două grafuri A și B, având același număr de noduri. Să se obțină graful B din graful A, utilizând următoarele operații:

- adăugând o muchie între două vârfuri care sunt legate printr-un drum de lungime mai mare decât 1;
- eliminând o muchie ale cărei extremități sunt legate printr-un drum de lungime mai mare decât 1.

Restricții

Datele sunt citite dintr-un fișier text CONEX.IN, având următoarea structură:

- pe prima linie este scris numărul nodurilor (n);
- pe următoarele $2 \times n$ linii sunt scrise matricele de adiacență ale celor două grafuri (1 dacă nodurile respective sunt legate printr-o muchie, 0 în caz contrar) ([1]).

Soluție*

Din cele două condiții din enunț deducem că o muchie poate fi adăugată sau eliminată doar dacă nodurile care reprezintă extremitățile acesteia fac parte din aceeași componentă conexă. Deci, vom împărți graful inițial în componente conexe. În cazul în care extremitățile unei muchii care aparțin grafului inițial și care nu aparțin grafului final (sau invers) se găsesc în componente conexe diferite, atunci problema nu are soluție.

Determinarea componentelor conexe se face în felul următor:

- calculăm închiderea tranzitivă a grafului respectiv;
- această închidere o vom memora în matricea a ;
- două noduri i și j vor face parte din aceeași componentă conexă dacă $a_{ij} = a_{ji} = 1$.

* Ideea soluției îi aparține lui Radu Lupșa (preparator la Univ. "Babeș-Bolyai").

Pe prima linie a fișierului INPUT.TXT este scris numărul de centre comerciale (n); pe următoarele linii sunt scrise perechi de numere: $i \ j$ – reprezentând strada între centrele i și j cu sensul de parcurgere de la i la j .

Rezultatul se va afișa pe ecran; în cazul în care există mai multe soluții se vor afișa toate [2].

Exemplu:

Intrare Ieșire

```
7
1 2
2 7
1 7
7 3
5 6
5 4
6 4
4 3
```

Soluție

Pentru reprezentarea centrelor comerciale și a străzilor folosim un graf orientat, reprezentat prin matricea de adiacență a_{nxn} . Centrele care respectă condițiile problemei se identifică în modul următor: se determină cu ajutorul algoritmului Roy-Warshall, matricea drumurilor (închiderea tranzitivă). Apoi, însumând elemente pe fiecare coloană j , vom determina numărul de noduri prin care se poate ajunge în nodul j . Evident, soluția se constituie din acele noduri care au suma pe coloană egală cu $n - 1$ (fără elementul $a_{j,j}$).

```
Program graf;
var a:array[1..100,1..100] of Integer;
i,j,n,k:Integer;
f:Text;

procedure citeste;
begin
  for i:=1 to n do for j:=1 to n do a[i,j]:=0;
  Assign(f,'INPUT.TXT'); Reset(f); Readln(f,n);
  while not Eof(f) do
    begin Readln(f,i,j); a[i,j]:=1 end
end;

procedure Roy_Warshall;
begin
  for k:=1 to n do
    for i:=1 to n do
      for j:=1 to n do
        if a[i,j]=0 then a[i,j]:=a[i,k]*a[k,j]
end;
```

```
procedure sumă;
var s:Integer;
begin
  for j:=1 to n do
    begin
      s:=0;
      for i:=1 to n do
        if i<>j then s:=s+a[i,j];
      if s=n-1 then Writeln(j)
    end;
end;

Begin
  citeste;
  Roy_Warshall;
  sumă
End.
```

P55. Maximizări

Se consideră un graf $G=(X, U)$ neorientat, neconex.

Vom considera că două noduri sunt *conectate* dacă există un lanț care le unește.

Să se determine o pereche de noduri între care nu există muchie, astfel încât, prin adăugarea muchiei corespunzătoare numărul de noduri conectate să fie maxim.

Pe prima linie a fișierului INPUT.TXT este scris numărul de noduri (n); pe următoarele linii sunt scrise perechi de numere: $i \ j$ – reprezentând perechi de noduri adiacente. Rezultatul se va afișa pe ecran; în cazul în care există mai multe soluții se va afișa una singură [2].

Exemplu:

Intrare Ieșire

```
9
1 2
1 3
2 4
3 4
5 6
5 7
6 7
8 9
```

Soluție

Reprezentăm graful neorientat prin matricea a_{2xn} a muchiilor. Numărul m al muchiilor este determinat la citire. Se identifică muchia care trebuie adăugată determinând toate componentele conexe (procedura conexitate) care va reține în vectorul

P53. Problema acvariului

Intr-un acvariu se află n pești carnivori. Știind care pești se pot mâncă unii pe alții, se cere să se determine ordinea în care aceștia se mănâncă între ei, astfel încât în final să rămână un număr minim de pești [!].

Observație

Un pește poate fi mâncat doar de un pește mai mare decât el.

Restricții

- $n \leq 200$;

- Datele se citesc dintr-un fișier de intrare având următoarea structură:
- pe prima linie este scris numărul de pești;
- pe următoarele n linii (sub forma unei matrice) sunt scrise relațiile dintre pești (pe poziția (i, j) vom avea numărul 1 dacă peștele i poate mâncă peștele j, 0 în caz contrar).

Soluție

Reprezentăm relațiile dintre pești sub formă unui graf orientat în care arcul (i, j) are semnificația că peștele i poate mâncă peștele j . Datorită faptului că un pește mai mic nu poate mâncă un pește mai mare, graful atașat relațiilor dintre pești nu va avea cicluri.

Ordinea în care se vor mâncă peștii între ei se determină în felul următor: la fiecare pas vom găsi un pește care nu mai poate mâncă nici un alt pește, dar care poate fi mâncat de alți pești, deci îl vom elimina. Această operație corespunde cu găsirea unui nod din care nu pleacă nici un arc, dar în care vin arce, și cu eliminarea lui împreună cu arcele incidente.

Vom repeta pasul de mai sus până în momentul în care nu mai există nici un arc în graf.

```
Program ProblemaPestilor;
const MaxN=200;
var a:array[1..MaxN,1..MaxN] of 0..1;
n,i,j,s:Byte;
morti:set of Byte;
f:Text;
blocaj,gasit:Boolean;

Begin
  Assign(f,'PESTI.IN'); Reset(f); Readln(f,n);
  for i:=1 to n do
    begin
      for j:=1 to n do
        if not Seekoleln(f) then Read(f,a[i,j]);
      Readln(f)
    end;
  end;
```

```
Close(f);
morti:=[];
blocaj:=false;
while not blocaj do
begin
  blocaj:=true; gasit:=true;
  for i:=1 to n do
    begin
      gasit:=false;
      if not (i in morti) then
        begin
          s:=0;
          for j:=1 to n do s:=s+a[i,j];
          if s=0 then
            begin
              j:=1;
              while (a[j,i]=0) and (j<=n) do Inc(j);
              if j<=n then
                begin
                  Writeln('Pestele ',j,' mananca pestele ',i);
                  for j:=1 to n do a[j,i]:=0;
                  morti:=morti+[i]; gasit:=true
                end
              end;
            end;
        end;
      end;
    end;
  if gasit then blocaj:=false
end;
End.
```

Intrare

```
5
0 1 0 0 1
0 0 1 0 0
0 0 0 1 1
0 0 0 0 0
0 0 0 0 0
```

Ieșire

```
Pestele 3 mananca pestele 4
Pestele 1 mananca pestele 5
Pestele 2 mananca pestele 3
Pestele 1 mananca pestele 2
```

P54. Străzi

Intr-un oraș există n centre comerciale unite printr-o rețea stradală, toate fiind cu circulație în sens unic.

Se cere determinarea unui centru comercial în care se poate ajunge din oricare alt centru comercial.

```

procedure ComponenteInternStabile(a:graf; nr:Byte;
    var comp:componente; var nrc:Word);
var v,g,h,nrcl:Word;
    vecini:set of 1..MaxN;
begin
    nrc:=1; comp[1]:=[1];
    for v:=2 to nr do
    begin
        vecini:=[];
        for h:=1 to nr do
            if (a[h,v]=1) then vecini:=vecini+[h];
        nrcl:=nrc;
        for g:=1 to nrc do
            if (comp[g]*vecini)=[]
            then
                begin
                    Inc(nrcl); comp[nrcl]:=comp[g]+[v];
                end
            else
                begin
                    Inc(nrcl); comp[nrcl]:=comp[g]-vecini+[v];
                end;
        nrc:=nrcl; { vom elimina componente intern stabile }
        { care nu sunt maximale in raport cu incluziunea }
        for h:=1 to nrcl-1 do
            for g:=1 to nrc do
                if (g<>h) and (comp[g]<=comp[h]) then comp[g]:=[];
        for h:=1 to nrc do
            if comp[h]<>[]
            then
                begin
                    g:=h;
                    while (g>1) and (comp[g-1]=[]) do Dec(g);
                    comp[g]:=comp[h];
                    if h>g then comp[h]:=[];
                    nrcl:=nrcl-(h-g)
                end
        end;
    end;
begin
    Assign(f,'AMENZI.IN'); Reset(f); Readln(f,n);
    for i:=1 to n do
    begin
        for j:=1 to n do
            if not Seekof(f) then Read(f,b[i,j]);
        Readln(f);
    end;
    ComponenteInternStabile(b,n,compon,nrcomp);

```

```

max:=0;
for k:=1 to nrcomp do
begin
    s:=0;
    for i:=1 to n do
        if i in compon[k] then Inc(s);
    if max<s then max:=s;
end;
Write('Numarul maxim de restrictii care pot fi puse :');
Writeln('simultan este:',max); Readln;
if not Seekeof(f) then
    begin
        { Se citesc amenzile care }
        { nu pot fi incalcate simultan. }
        begin
            for i:=1 to n do
            begin
                for j:=1 to n do
                    if not Seekof(f) then Read(f,b[i,j]);
                Readln(f);
            end;
            for i:=1 to n do
                if not Seekof(f) then Read(f,cost[i]);
            Close(f);
            ComponenteInternStabile(b,n,compon,nrcomp); max:=0;
            for k:=1 to nrcomp do
            begin
                s:=0;
                for i:=1 to n do
                    if i in compon[k] then s:=s+cost[i];
                if s>max then max:=s;
            end;
            Writeln('Amenda maxima este:',max);
        end;
    end;

```

Intrare

```

3
0 1 0
1 0 1
0 1 0
0 1 0
1 0 1
0 1 0
1 4 2

```

Iesire

Numarul maxim de restrictii care pot fi puse simultan este:2
Amenda maxima este:4

```

while gasit and not marcat[final] do
begin
  gasit:=false;
  for j:=1 to m do
    if marcat[j]
    then
      for k:=1 to m do
        if not marcat[k] and (saturatB[k] or (k=final))
        then
          for i:=1 to n do
            if (a[i,j]=1) and (a[i,k]=1) and(saturatA[i])
            then
              if InCuplaj(i, j) and not InCuplaj(i, k)
              then
                begin
                  gasit:=true;
                  marcat[k]:=true;
                  predA[k]:=i;
                  predB[k]:=j
                end
            end;
            if marcat[final]
            then
              begin
                ReconstituieLant(final);
                DeterminaLantAlternantCrescator:=true;
                saturatA[start]:=true;
                saturatB[final]:=true
              end
            . else DeterminaLantAlternantCrescator:=false
          end;
        procedure TransferDeLungime(lant:TLant; lung:Byte);
        var CuplajNou:TCuplaj;
          NrKNou:Byte;
          gasit:Boolean;
        begin
          NrKNou:=0;
          i:=1;
          while i<=lung-1 do
            begin
              Inc(NrKNou);
              CuplajNou[NrKNou].u:=lant[i];
              CuplajNou[NrKNou].v:=lant[i+1];
              Inc(i,2);
            end;

```

```

for j:=1 to nrK do
begin
  j:=2;
  gasit:=false;
  while (j<=lung-2) and not gasit do
begin
  if (Cuplaj[i].u=lant[j+1]) and (Cuplaj[i].v=lant[j])
  then gasit:=true
  else Inc(j,2)
end;
if not(gasit)
then
begin
  Inc(NrKNou);
  CuplajNou[NrKNou]:=Cuplaj[i]
end;
Cuplaj:=CuplajNou;
nrK:=nrKNou
end;

Begin
  Citire;
  nrK:=0; { numarul muchiilor din cuplaj este 0 }
  operatii:=true;
  while operatii do
  begin
    operatii:=false;
    for i:=1 to n do
      if not saturatA[i]
      then
        begin
          for j:=1 to m do
            if not saturatB[j]
            then
              if DeterminaLantAlternantCrescator(i, j, lant, lung)
              then begin operatii:=true; Break end;
              if operatii then Break
            end;
            if operatii
            then
              begin
                TransferDeLungime(lant, lung);
                Fillchar(marcat,m,false)
              end;
            end;
          for i:=1 to NrK do
            Writeln(Cuplaj[i].u, ' ', Cuplaj[i].v)
        End.

```

Intrare	Ieșire
4 5	4 2
0 0 0 0 1	2 3
1 1 1 1 0	3 1
1 0 0 1 0	1 5
0 1 0 1 1	

Discuție

Problema cuplajului poate să se ascundă în enunțurile unor probleme similare cu cele de mai jos:

La o discotecă se află n fete și m băieți. Fiecare fată are o listă de preferințe cu băieți pe care i-ar dori ca parteneri de dans. Se cere să se determine partenerul fiecărei fete, astfel încât să danseze un număr maxim de fete.

O altă variantă a enunțului de mai sus, constă în existența și a unor preferințe a băieților. Determinați în aceste condiții o soluție în care un număr maxim de perechi să danseze fără ca nimeni să nu facă compromisuri.

La un concurs există n calculatoare și m concurenți. Fiecare concurent are anumite preferințe. Distribuiți concurenților calculatoare, astfel încât numărul de preferințe satisfăcute să fie maxim.

P52. Amenzi de circulație

Pe unele străzi dintr-un oraș se vor stabili restricții de circulație. Știind că există perechi de restricții care nu pot fi puse simultan pe aceeași stradă, se cere:

- să se determine numărul maxim de restricții care pot fi stabilite simultan pe aceeași stradă;
- știind că încălcarea restricției i este penalizată cu cost $[i]$ lei și că există restricții care nu pot fi încălcate simultan, se cere să se determine amenda maximă pe care o poate primi un conducător auto; (o restricție poate fi încălcată cel mult o dată);
- dându-se valoarea unei amenzi, se cere să se determine restricțiiile încălcate.

Restricții

Datele se citesc dintr-un fișier text, având următoarea structură:

- pe prima linie este scris numărul de restricții (n);
- pe următoarele n linii sunt date (sub forma unei matrice) restricțiiile care nu pot fi puse simultan (1 – dacă două restricții nu pot fi puse simultan, 0 în caz contrar);
- urmează o linie vidă;
- pe următoarele n linii sunt date restricțiiile care nu pot fi încălcate simultan (1 dacă două restricții nu pot fi încălcate simultan, 0 în caz contrar);
- pe următoarea linie se dau amenzile pentru fiecare restricție încălcată [1].

Soluție

Dacă reprezentăm restricțiile și interdependențele dintre ele sub formă unui graf, ale căruia noduri sunt chiar restricțiile de circulație, iar între două noduri i și j există muchie doar dacă restricția i nu poate fi amplasată în același timp cu restricția j , atunci va trebui să determinăm o componentă intern stabilită maximală (față de incluziune) care conține un număr maxim de elemente (restricții). Problema determinării componentelor interne stable maximale intr-un graf oarecare face parte din clasa problemelor *NP-complete*, deci are soluție doar în timp de execuțare exponențial.

Această problemă ar putea fi rezolvată prin metoda *backtracking*, dar am preferat să căutăm un algoritm mult mai simplu de programat. Se numește *algoritmul Bednarik-Tauble* și construiește recursiv (prin inducție) componente interne stable maximale. Trebuie să găsim toate componentele interne stable maximale ale unui graf cu n noduri.

Pentru subgraful format cu nodul 1 avem o singură componentă intern stabilită maximală. Să presupunem că am determinat toate componentele interne stable maximale ale unui subgraf cu k noduri ($k < n$) și le avem într-un vector de mulțimi (fiecare mulțime este o componentă intern stabilită maximală). Pentru subgraful cu $k+1$ noduri, componentele interne stable maximale se determină în felul următor: fie X o componentă intern stabilită maximală a subgrafului cu k noduri. Dacă X intersectată cu mulțimea vecinilor vârfului $k+1$ este mulțimea vidă atunci mulțimea $X \setminus \{k+1\}$ care se adaugă vârfului $\{k+1\}$ este componentă intern stabilită a subgrafului cu $k+1$ noduri. Dacă X intersectată cu mulțimea vecinilor vârfului $k+1$ nu este mulțimea vidă atunci $X \setminus \{k+1\}$ va fi de asemenea o componentă intern stabilită a subgrafului cu $k+1$ noduri.

Procedura ComponenteInternStabile implementează acest algoritm; va determina toate componentele interne stable maximale ale grafului reprezentat prin matricea de adiacență a . La prima vedere algoritmul pare de complexitate liniară, dar la o analiză mai atentă se observă că numărul componentelor interne stable maximale crește exponențial cu numărul de noduri ale grafului.

Rezolvarea punctului c) o lăsăm în seama cititorului.

Program AmenziDeCirculatie;

```
const MaxN=50;
type graf=array[1..MaxN,1..MaxN] of 0..1;
    mult=set of 1..MaxN;
    componente=array[1..5000] of mult;
var cost:array[1..MaxN] of Word;
    n,i,j:Byte;
    nrcomp,k,max,s:Word;
    compon:componente;
    b:graf;
    f:Text;
```

fel: în primul rând se observă că deoarece lanțul alternant este crescător (adică are vârfurile nesaturate) lungimea lui va fi impară (adică va avea un număr impar de muchii). Rezultă că un nod nesaturat, fie acesta V_1 , va fi într-o mulțime a grafului bipartit, iar nodul V_2 va fi în cealaltă mulțime). Algoritmul constă în marcarea la primul pas a tuturor nodurilor nemarcate și care sunt legate cu V_1 . Dacă V_2 se află în mulțimea marcată atunci algoritmul se termină. În caz contrar se marchează toate nodurile (din mulțimea din care face parte V_1) care sunt legate prin muchii saturate cu nodurile deja marcate. La un pas de ordin par se marchează nodurile legate prin muchii din K de celelalte noduri marcate, iar la un pas de ordin impar se marchează vârfurile legate de cele marcate prin muchii din $G - K$.

Pentru reconstituirea unui lanț alternant crescător, se folosesc vectorii $predA$, $predB$ care rețin de unde a provenit marcajul nodului curent.

Implementarea care urmează are o complexitate mai mare din cauza unor considerente didactice de claritate. Un algoritm optim are complexitatea $O(N^3)$.

```
Program CuplajBipartit;
const MaxN=200;
type TLant=array[1..2*MaxN] of Byte;
    muchie=record
        u,v:Byte
    end;
    TCuplaj=array[1..MaxN] of muchie;
var a:array[1..MaxN,1..MaxN] of Byte;
    saturatA,saturatB:array[1..MaxN] of Boolean;
    Cuplaj:TCuplaj; { cuplajul }
    nrK:Byte; { numarul muchiilor din cuplaj }
    n,m,i,j:Byte;
    f:Text;
    operatii:Boolean;
    lant:TLant; { un lant alternant crescator }
    lung:Byte; { lungimea lantului }
    marcat:array[1..MaxN] of Boolean;
    predA,predB:array[1..MaxN] of Byte;

procedure Citire;
begin
    Assign(f,'CUPLAJ.IN');
    Reset(f);
    Readln(f,n,m);
    for i:=1 to n do
    begin
        for j:=1 to m do Read(f,a[i,j]);
        Readln(f)
    end;
    Close(f);
end;
```

```
function InCuplaj(i,j:Byte):Boolean;
var k:Byte;
begin
    for k:=1 to nrK do
        if (Cuplaj[k].u=i) and (Cuplaj[k].v=j)
        then
            begin
                InCuplaj:=true;
                Exit
            end;
    InCuplaj:=false
end;

function DeterminaLantAlternantCrescator(start,final:Byte;
                                             var lant:TLant; var Lung:Byte):Boolean;
var gasit:Boolean;
    k,j,i:Byte;

procedure ReconstituieLant(fin:Byte);
begin
    if predA[fin]<>start
    then
        begin
            ReconstituieLant(predB[fin]);
            Inc(lung);
            lant[lung]:=predA[fin];
            Inc(lung);
            lant[lung]:=fin
        end
    else
        if predA[fin]=start
        then
            begin
                lung:=1;
                lant[lung]:=start;
                Inc(lung);
                lant[lung]:=fin
            end
end;

begin { function DeterminaLantAlternantCrescator }
    gasit:=false;
    for j:=1 to m do
        if (a[start,j]=1) and (saturatB[j] or (j=final))
        then
            begin
                marcat[j]:=true;
                gasit:=true;
                predA[j]:=start
            end;
end;
```

P50. Joc

Să se simuleze, utilizând alocare dinamică, următorul joc: n copii sunt în cerc. Copiii efectuează o numărătoare și fiecare al k-lea copil ieșe din cerc. Câștigă jocul acel copil care rămâne ultimul în cerc. Numerele n și k se citesc de la tastatură. Numele copiilor se citesc de pe căte o linie din fișierul JOC.TXT.

Se cere să se afișeze numele copiilor în ordinea ieșirii lor din joc ([2]).

Soluție

Așezarea copiilor în cerc va fi simulață printr-o listă circulară care se realizează prin legarea ultimului element din listă de primul. Prin codificarea în acest fel a jocului, ieșirea unui copil din cerc înseamnă stergerea elementului corespunzător din listă.

Program joc;

```

type reper^=element;
  element=record
    nume:string[20];
    urm:reper;
  end;
var f:Text;
  k,i,n:Byte;
  p,q,sant,ultim:reper;
begin
  Assign(f,'JOC.TXT'); Reset(f);
  New(sant); sant^.urm:=nil; ultim:=sant;
  Write('n='); Readln(n);
  for i:=1 to n do
  begin
    New(p);
    Readln(f,p^.nume);
    p^.urm:=nil;
    ultim^.urm:=p;
    ultim:=p;
  end;
  Close(f);
  Write('k='); Readln(k);
  ultim^.urm:=sant^.urm; { după ultimul copil urmează primul }
  p:=sant^.urm; { numaratoarea incepe de la primul copil }
  while p<>p^.urm do
  begin
    for i:=1 to k-1 do p:=p^.urm;
    Writeln('Ieșe din cerc ',p^.nume);
    q:=p^.urm; p^:=q^;
    Dispose(q);
  end;
  Writeln('A câștigat ',p^.nume);
end.

```

P51. Cuplajul în grafuri bipartite

Se dă un graf bipartit. Se cere să se determine o (sub)mulțime maximală de muchii cu proprietatea că oricare două muchii nu au aceeași extremitate.

Restricții

Datele de intrare se citesc dintr-un fișier text având următoarea structură:

- pe prima linie se dau numerele n și m reprezentând cardinalele celor două mulțimi de vârfuri care formează graful bipartit;
- pe următoarele n linii se dă o matrice de dimensiuni $n \times m$ reprezentând legăturile dintre noduri (1 dacă nodurile respective sunt legate; 0 în caz contrar); elementele matricei sunt despărțite printr-un spațiu.

Soluție

Avem nevoie pentru început de câteva definiții:

1. Se numește **cuplaj** într-un graf, o submulțime de muchii, cu proprietatea că oricare două nu au o extremitate comună. Cuplajul este maximal dacă numărul de muchii care îl formează este maxim (orice muchie am adăuga ar avea un capăt deja atins de o altă muchie din cuplaj).
2. Se numește **vârf saturat** al unui graf, în raport cu un cuplaj, acel vârf care este extremitate pentru o muchie din cuplaj. Celelalte vârfuri se numesc **nesaturate**.
3. Se numește **lanț alternant** al unui graf G în raport cu un cuplaj K , un drum ale căruia muchii alterneză în K și în $G - K$. Un lanț alternant se numește **crescător**, dacă extremitățile lui sunt nesaturate.
4. Fie K un cuplaj și L un lanț alternant crescător. Diferența simetrică $D = (K - L) \cup (L - K)$ poartă numele de **transfer de lungime**. Mulțimea de muchii D este tot un cuplaj, însă de cardinal cu unu mai mare decât K .

Algoritmul de determinare a unui cuplaj într-un graf bipartit este următorul:

Se pornește de la un cuplaj aleator K (poate să fie și vid sau format dintr-o singură muchie).

Cât_timp există L lanț alternant crescător execută

Transfer de Lungime între L și K

Rezultatul este nouj cuplaj.

Sfârșit_cât_timp

Acest algoritm nu este valabil și pentru cazul general, din cauză că într-un graf oricare nu se poate determina un lanț alternant crescător în timp polinomial. Pentru un graf oarecare există un alt algoritm care în loc de graf alternant, construiește un arbore alternant.

În graful bipartit determinarea unui lanț alternant crescător relativ la K se face astfel:

```

Begin
  Assign(f,'INTER.TXT'); Reset(f);
  New(L1); New(L2); { crearea santinelelor }
  L1^.leg:=nil; L2^.leg:=nil; { listeau santinele initiale }
  L1^.info:=maxint; { presupun ca numerele sunt strict mai }
  L2^.info:=maxint; { mici decat maxint }
  { citirea si crearea listelor }
  { prima lista-prima linie din fisier }

  while not Eoln(f) do
    begin Read(f,nr); adaug(L1,nr) end;
  Readln(f); { a doua lista-a doua linie din fisier }
  while not Eoln(f) do
    begin
      Read(f,nr);
      adaug(L2,nr);
    end;
  Writeln; Writeln('Prima lista:');
  afisare(L1);
  Writeln; Writeln('A doua lista:');
  afisare(L2);
  intersectie(L1,L2,L);
  Writeln;
  Writeln('Intersectia:');
  afisare(L);
  Close(f);
End.

```

P48. Interclasare

Se citesc din fișierul text SIRURI.TXT două linii care conțin două siruri neordonate de numere întregi.

Să se creeze din fiecare sir câte o listă de numere ordonate crescător. Să se construiască o a treia listă care se obține prin interclasarea celor două și să se afișeze toate cele trei liste ([2]).

Soluție

```

Program interclasare;

type list^=^nod;
  nod=record
    nr:Integer;
    urm:list;
  end;
var cap1,cap2,cap3:list;
  n:Integer;
  f:Text;

```

```

procedure inter(cap1,cap2:list; var cap3:list);
  { interclasarea propriu-zisă }
  { cap1,cap2 : santinelele listelor initiale }
  { cap3: santineala liste rezultate din interclasare }
  var p,q:list;
begin
  New(cap3);
  cap3^.urm:=nil;
  cap3^.nr:=-maxint; { crearea santinelei liste rezultate }
  cap1:=cap1^.urm;
  cap2:=cap2^.urm;
  p:=cap3;
  while (cap1<>nil) and (cap2<>nil) do
    begin
      New(p^.urm); p:=p^.urm;
      p^.urm:=nil;
      if cap1^.nr<cap2^.nr { adaugarea unui nou nod in lista 3 }
      then begin p^.nr:=cap1^.nr; cap1:=cap1^.urm end
      else begin p^.nr:=cap2^.nr; cap2:=cap2^.urm end
    end;
  while cap1<>nil do
    begin
      New(p^.urm);
      p:=p^.urm;
      p^.urm:=nil;
      p^.nr:=cap1^.nr;
      cap1:=cap1^.urm
    end;
  while cap2<>nil do
    begin
      New(p^.urm);
      p:=p^.urm;
      p^.urm:=nil;
      p^.nr:=cap2^.nr;
      cap2:=cap2^.urm
    end;
end;

procedure adaug(n:Integer; cap:list);
  var p,q:list; { adauga un element, pastrand lista ordonata }
begin
  p:=cap; { se cauta primul nod cu informatie mai }
  { mare sau egala, sau sfirsitul listei }
  while (p^.urm<>nil) and (p^.urm^.nr<n) do p:=p^.urm;
  New(q);
  q^.nr:=n;
  q^.urm:=p^.urm;
  p^.urm:=q
end;

```

```

procedure afisare(c:list); { afiseaza o lista cu santinela c }
begin
  c:=c^.urm;
  while c<>nil do
  begin Write(c^.nr:4); c:=c^.urm end;
  Writeln;
end;

Begin
  Assign(f, 'SIRURI.TXT'); Reset(f);
  New(cap1);
  cap1^.urm:=nil; cap1^.nr:=-maxint; { creare santinele }
  New(cap2);
  cap2^.urm:=nil; cap2^.nr:=-maxint;
  while not Eoln(f) do { citire si adaugare in liste }
  begin Read(f,n); adaug(n,cap1) end;
  Readln(f);
  while not Eoln(f) do begin Read(f,n); adaug(n,cap2) end;
  Writeln('Lista 1:');
  afisare(cap1);
  Writeln('Lista 2:');
  afisare(cap2);
  Writeln('Lista rezultata din interclasare:');
  inter(cap1,cap2,cap3);
  afisare(cap3);
  Close(f)
End.

```

P49. Arboare

Să se citească (din fișierul standard de intrare) un număr necunoscut de cuvinte (nu neapărat distincte) separate prin caractere albe (spațiu, Enter). Să se creeze un arbore binar de căutare care să cuprindă numai cuvintele distincte.

Să se afișeze cuvintele din arbore, parcurs în preordine (cu indentare) ([2]).

Soluție

```

Program arbore;
type cuvant=string[20];
arb:^nod;
nod=record
  cuv:cuvant;
  st,dr:arb
end;
var cap:arb;
f:Text;
s,s1:cuvant;
i:Byte;
{ radacina arborelui }

```

```

function adaug(s:cuvant; p:arb):arb;
{ adauga in arbore doar cand cuvantul nu a mai fost introdus }
{ functie recursiva care returneaza pointer spre nodul creat }
begin
  if p=nil
  then begin New(p); p^.cuv:=s; p^.st:=nil; p^.dr:=nil end
  else
    if p^.cuv<s then p^.dr:=adaug(s,p^.dr)
    else
      if p^.cuv>s then p^.st:=adaug(s,p^.st);
  adaug:=p
end;

procedure afisare(c:arb; i:Integer);
{ procedura afiseaza arborele in preordine, arbore nenul }
{ i asigura indentarea -numarul de spatii din fata numarului }
begin
  Write(' ':i); Writeln(c^.cuv);
  if c^.st<>nil then afisare(c^.st,i+1);
  if c^.dr<>nil then afisare(c^.dr,i+1)
end;

Begin
  Assign(f, 'CUV.TXT'); Reset(f); cap:=nil;
  while not Eof(f) do
  begin
    Readln(f,s); { se elimină spațiile de la începutul liniei }
    while s[1]=' ' do Delete(s,1,1);
    if s<>''
    then
      repeat
        i:=Pos(' ',s);
        if i>0
        then
          begin
            s1:=Copy(s,1,i-1);
            cap:=adaug(s1,cap);
            Delete(s,1,i); { cuvantul extras se elimină din linie }
            while s[1]=' ' do Delete(s,1,1)
          end
        else cap:=adaug(s,cap)
        until i=0
      end;
    if cap<>nil
    then afisare(cap,1)
    else Writeln('Arbore vid');
  Close(f)
End.

```

```

if cuv<> ''
then
begin
  p:=sant1^.urm;
  while p^.cuvant<cuv do p:=p^.urm;
  if p^.cuvant>cuv { daca cuvantul cuv nu mai exista }
  then { in lista va fi introdus in fata }
  begin { celui care-i urmeaza alfabetic }
    New(q);
    q^:=p^;
    p^.cuvant:=cuv;
    p^.urm:=q;
    if p=sant2
    then sant2:=q
  end
end;
Close(f);
p:=sant1^.urm;
while p>sant2 do
begin
  Writeln(p^.cuvant);
  p:=p^.urm
end;
End.

```

P47. Intersecție

Se citesc două linii din fișierul text INTER.TXT care cuprinde numere întregi.
Să se creeze din fiecare linie câte o listă alocată dinamic, formată din numerele distincte. Să se creeze și să se afișeze o nouă listă formată din intersecția celor două liste ([2]).

Soluție

```

Program inter;
type lista=^nod;
nod=record
  info:Integer;
  leg:lista
end;
var f:Text;
L1,L2,L:lista;
nr:Integer;

```

```

procedure afisare(L:lista);
{ afisează lista care incepe cu sântinela L }
var p:lista;
begin
  p:=L^.leg;
  while p<>nil do
    begin Write(p^.info:6); p:=p^.leg end
  end;

procedure intersectie(L1,L2:lista; var L:lista);
{ creează lista L, formată din intersecția listelor L1 și L2 }
var p1,p2,p:lista;
begin
  New(L);
  L^.leg:=nil;
  p1:=L1^.leg;
  p:=L;
  while p1<>nil do
    begin { se parcurge prima listă }
      p2:=L2^.leg;
      while p2<>nil do { se verifică dacă elementul din }
      begin { prima listă este și în a două }
        if p1^.info = p2^.info
        then
          begin
            New(p^.leg); { se adaugă un nou nod în listă nouă }
            p:=p^.leg;
            p^.leg:=nil;
            p^.info:=p1^.info;
            p2:=nil
          end
        else p2:=p2^.leg
      end;
      p1:=p1^.leg
    end;
  end;

procedure adaug(L:lista; n:Integer);
var p:lista;
begin
  p:=L; { verifică dacă numarul este deja în lista }
  while (p^.leg<>nil) and (p^.info<>n) do p:=p^.leg;
  if p^.info<>n
  then
    begin
      New(p^.leg);
      p:=p^.leg; p^.info:=n; p^.leg:=nil
    end
  end;
end;

```

```

begin
  pi:=Trunc(Int(Abs(n))); { contine partea intreaga a lui n}
  i:=1;
repeat
  s[i]:=a[pi mod 16];
  pi:=pi div 16; { s contine cifrele reprezentarii partii}
  i:=i+1; { intregi a numarului n in baza 16 in s}
until pi=0; { ordinea inversa celei dorite}
i:=i-1; j:=1;
if n<0:
then
begin
  x[1]:=-'; j:=2
end;
while i>0 do
begin
  x[j]:=s[i]; i:=i-1; j:=j+1
end;
x[j]:='.'; { x contine cifrele reprezentarii partii}
j:=j+1; { intregi a numarului n in baza 16 in x}
{ ordinea dorita}
pr:=Abs(n)-Int(Abs(n));
{ pr contine partea fractionara a numarului n}
i:=1;
while (i<6) and (Frac(pr)<>0) do
begin
  x[j]:=a[Trunc(Int(pr*16))];
  j:=j+1;
  pr:=pr*16;
  pr:=pr-Int(pr);
  i:=i+1
end;
x[0]:=Chr(j-1); { se modifica lungimea sirului x}
hexa:=x
end;
Begin
  New(sant1); New(sant2);
  sant1^.urm:=sant2; sant2^.pred:=sant1;
  Write('Numarul:');
  while not Eof(f) do
begin
  Readln(nr);
  p:=sant2;
  p^.nr_hexa:=hexa(nr);
  New(sant2);
  sant2^.pred:=p;
  p^.urm:=sant2;
  Write('Numarul:')
end;

```

```

  p:=sant1^.urm;
  if p=sant2
  then Writeln('Lista vida')
  else
    while p>>sant2 do
      begin Writeln(p^.nr_hexa); p:=p^.urm end
End.

```

P46. Cuvinte distincte

Se citesc cuvinte din fișierul text CUV.TXT în care pe o linie pot fi mai multe cuvinte separate prin spații.

Să se creeze o listă înlățuită, ordonată alfabetic care să cuprindă cuvintele distincte din fișier.

Să se afișeze lista creată ([2]).

Soluție

```

Program cuvinte_ordonate_alfabetic;
  type reper^=element;
    element=record
      cuvant:string[20];
      urm:reper
    end;
  var f:Text;
  cuv:string[20];
  linie:string;
  p,q,sant1,sant2:reper;
  i:Integer;
begin
  New(sant1); New(sant2);
  sant1^.urm:=sant2; sant2^.cuvant:='zzzzzzzzzzzzzzzz';
  { pentru ca si primul cuvant din lista sa fie introdus }
  { inaintea unui cuvant care-i urmeaza alfabetic, campul }
  { cuvant din sant2 se initializeaza cu 'zz...zz' }
  Assign(f,'CUV.TXT'); Reset(f);
  while not Eof(f) do
begin
  Readln(f,linie);
  i:=1;
  while i<=Length(linie). do
begin
  cuv:='';
  while (i<=Length(linie)) and (linie[i]<>' ') do
  begin cuv:=cuv+linie[i]; i:=i+1 end;
  while (i<=Length(linie)) and (linie[i]=' ') do i:=i+1;
  if cuv<>' ' then
    begin
      p:=sant1;
      while p^.cuvant>=cuv do p:=p^.urm;
      q:=New(element);
      q^.cuvant:=cuv;
      if p=sant1 then
        begin
          q^.pred:=nil;
          q^.urm:=sant1;
          sant1^.pred:=q;
          sant1:=q;
        end
      else
        begin
          q^.pred:=p^.pred;
          q^.urm:=p;
          p^.pred^.urm:=q;
          p^.pred:=q;
        end;
    end;
end;

```

```

procedure intrare;
begin
  if prim=nil then
    begin
      coada e vida, se adauga primul element
      New(prim);
      Write('introduceti codul '); Readln(prim^.cod);
      prim^.urm:=nil;
      ultim:=prim
    end
  else
    begin
      New(p);
      Write('introduceti codul ');
      Readln(p^.cod);
      p^.urm:=nil;
      ultim^.urm:=p; { legarea elementului la coada }
      ultim:=p { elementul adaugat devine ultimul }
    end
  end;
procedure stergere;
begin
  if prim=nil
  then Writeln('Coada estevida.')
  else
    begin
      Writeln('Ies nodul cu codul: ',prim^.cod,'');
      p:=prim;
      prim:=prim^.urm;
      Dispose(p)
    end;
  Readkey
end;
procedure listare;
begin
  if prim=nil
  then Writeln('Coada e vida.')
  else
    begin
      p:=prim;
      Writeln('Nodurile cozii: ');
      repeat
        Write(p^.cod,' ');
        p:=p^.urm
      until p=nil
    end;
  Readkey
end;

```

```

Begin
  prim:=nil;
  repeat
    Clrsr;
    Writeln('1: creare/adaugare ');
    Writeln('2: stergere nod ');
    Writeln('3: vizualizare coada ');
    Writeln('4: terminare ');
    Writeln('alege optiunea: ');
    Readln(w);
    case w of
      '1':intrare;
      '2':stergere;
      '3':listare
    end
  until w='4';
  Readkey
End.

```

P45. Reprezentare în baza 16

Să se creeze o listă dublu înlățuită având în elemente reprezentările în baza 16 a numerelor citite din fișierul standard de intrare. Să se afișeze lista creată ([2]).

Soluție

Lista dublu,înlățuită creată are două săntinile: *sant1* și *sant2*. Numerele se citesc pe rând din fișierul standard de intrare în variabila *nr* de tip real. Funcția *hexa* are un parametru de tip real. Transformă numărul primit ca parametru într-un sir care conține reprezentarea sa în baza 16. Funcția are două părți distincte: transformarea părții întregi și transformarea părții fraționare. Partea fraționară va avea cel mult 5 zecimale exacte:

```

Program reprezentare_in_baza_16;
const a:array[0..15] of Char=(0,'1','2','3','4','5','6','7',
                           '8','9','A','B','C','D','E','F');
type reper=^element;
  element=record
    nr_hexa:string[20];
    pred,urm:reper
  end;
var nr,x:Real;
  p,sant1,sant2:reper;
function hexa(n:Real):string;
  var s,x:string[20];
  pi:Word; pr:Real; i,j:Byte;

```

```

procedure valoare;
begin
  p:=nod;
  val:=0;
  while p<>nil do
  begin
    val:=val+p^.c*(Exp(p^.e*Ln(a)));
    p:=p^.leg
  end;
  Writeln(val:10:2)
end;

Begin
  New(nod);
  with nod^ do
  begin
    Write('Coeficient:'); Read(c);
    Write('Gradul:'); Read(e);
    leg:=nil
  end;
  creare(nod^.e);
  Read(a);
  valoare
End.

```

P43. Stive

Să se simuleze crearea unui vraf de farfurii (despachetate dintr-o cutie). Farfurii sunt desenate – pe partea interioară – cu diferite modele. După crearea stivei corespunzătoare vrafului, să se caute o anumită farfurie (având un anumit model), “așezând” totodată farfurii puse deoparte într-un vraf nou. Să se afișeze un mesaj corespunzător în urma căutării([2]).

Soluție

```

Program farfurii;
type vraf^=farfurie;
  farfurie=record
    model:string[20];
    urm:vraf
  end;
var cap,p,q,cap_nou:vraf;
  model_cautat:string[20];
  raspuns:Char;
  gasit:Boolean;
Begin
  cap:=nil;
  raspuns:='d';

```

```

while Upcase(raspuns)<>'N' do
begin
  New(p);
  with p^ do
  begin
    Write('model:'); Readln(model);
    urm:=cap;
  end;
  Write('Mai doriti sa introduceti (d/n)? ');
  Readln(raspuns);
  cap:=p;
end;
Write('Modelul cautat:'); Readln(model_cautat);
gasit:=false;
cap_nou:=nil;
while not gasit and (cap<>nil) do { cautare farfurie }
begin
  if cap^.model=model_cautat
  then gasit:=true
  else
  begin
    New(p);
    p^.model:=cap^.model;
    p^.urm:=cap_nou;
    cap_nou:=p;
    q:=cap; cap:=cap^.urm; Dispose(q)
  end
end;
if gasit then Writeln('Farfurie cautata nu s-a spart!')
  else Writeln('Farfuriă cautata s-a spart!')
End.

```

P44. Cozi

Să se realizeze operațiile de creare și vizualizare a unei cozi implementate dinamic, precum și eliminarea unui element din coadă ([2]).

Soluție

```

Program coada;
uses Crt;
type reper^=el;
  el=record
    cod:Word;
    urm:reper
  end;
var ultim,prim,p:reper;
  w:Char;

```

```

Begin
  Citeste;
  k:=0;
  for i:=1 to n-p+1 do
    for j:=1 to m-q+1 do
      if a[i][j]=0
      then
        begin
          Inc(k);
          mut[k].ii:=i; mut[k].jj:=j;
          for kl:=1 to p do
            for k2:=1 to q do
              a[i+kl-1][j+k2-1]:=1-a[i+kl-1][j+k2-1]
        end;
  Assign(f,'PANCU.OUT'); Rewrite(f);
  for i:=1 to n do
    for j:=1 to m do
      if a[i][j]=0
      then begin Writeln(f,'NU'); Close(f); Halt end;
  Writeln(f,'DA');
  for i:=1 to k do Writeln(f,mut[i].ii,' ',mut[i].jj);
  Close(f)
End.

```

Intrare

```

6 6
2 2
+++++
++++++
+---+-
++++++
+---+-
+---+-

```

Ieșire

```

DA
1 2
2 2
3 5
4 5
5 2

```

Discuție

Generalizați problema astfel încât să existe și căsuțe neutre (care nu au inițial nici un semn atașat), dar la prima operație efectuată asupra lor să li se atribuie automat semnul aflat sub colțul din stânga sus al submatricei.

P42. Polinoame

Se consideră un polinom $P(x)$, de grad n , reprezentat cu ajutorul unei liste simplu înlanțuite. Fiecare element al listei codifică un monom al polinomului păstrând coeficientul (număr real) și gradul monomului (număr întreg).

Să se scrie un program care creează lista necesară reprezentării polinomului $P(x)$ și care, pentru un număr real a , calculează valoarea $P(a)$.

Datele de intrare se introduc de la tastatură, iar valoarea $P(a)$ se va afișa pe ecran ([2]).

Exemplu:

Pentru polinomul $2x^3+x^2-2x+1$ și $a=1.0$ datele vor fi introduse astfel:

2 3 - monomul de grad 3

1 2 - monomul de grad 2

-2 1 - monomul de grad 1

1 0 - monomul de grad 0

1.0 - valoarea reală a

Răspunsul va fi: 2.0

Soluție

Rezolvarea problemei necesită parcurgerea a două etape: crearea listei care reține polinomul și calculul valorii $P(a)$.

Primul pas este realizat în procedura `creare`, primul monom fiind alocat însă în programul principal odată cu reținerea adresei acestuia în variabila `nod`.

Introducerea monoamelor este întreruptă când gradul monomului introdus este 0 deci s-a introdus inclusiv termenul liber. Calculul valorii polinomului $P(a)$ este realizat de procedura `valoarea` prin însumarea valorilor fiecărui monom în parte.

```

Program polinom;
type lista=^ inr;
  inr=record
    c,e:Integer;
    leg:lista
  end;
var p,q,nod:lista;
  val,a:Real;
procedure creare(g:Integer);
begin
  p:=nod;
  while g<>0 do
  begin
    New(q);
    with q^ do
    begin
      Write('Coeficient:'); Read(c);
      Write('Grad:'); Read(e);
      leg:=nil
    end;
    p^.leg:=q;
    p:=q; g:=q^.e
  end
end;

```

```

Begin
  Assign(f, 'SUME.IN'); Reset(f); Readln(f,n,m); Read(f, nr);
  s:=0;
  while (s+nr<m) and (nr<=l+s) and not Eof(f) do
    begin
      s:=s+nr;
      if s<0
      then
        begin
          Close(f);
          Assign(f, 'sume.out'); Rewrite(f);
          Writeln(f, 'DA');
          Close(f)
        end;
      Read(f, hr);
    end;
  Close(f);
  Assign(f, 'sume.out'); Rewrite(f);
  if nr>l+s then begin Writeln(f, 'NU'); Writeln(f, s+1) end
  else
    if s+nr<m then begin Writeln(f, 'NU'); Writeln(f, s+nr+1) end
    else Writeln(f, 'DA');
  Close(f);
End.

```

Exemple:

SUME.IN	SUME.OUT
4	DA

1	
2	

4	
5	

11	
----	--

SUME.IN	SUME.OUT
4	NU

1	8
---	---

2	
---	--

4	
---	--

10	
----	--

11	
----	--

P41. Panou

Se consideră un panou (o matrice) formată din $n \times m$ căsuțe în care se află înscrise inițial semnele + și - (exact un semn în fiecare căsuță). O mutare efectuată asupra acestui panou constă în schimbarea tuturor semnelor ($+$ \leftrightarrow $-$) căsuțelor aflate în interiorul unui dreptunghi de dimensiuni $p \times q$ (p și q sunt fixe și sunt citite de la intrare).

Să se determine dacă matricea dată poate fi transformată într-o matrice care conține doar semnul +. În caz afirmativ se cere un sir de mutări prin care se poate realiza acest lucru.

Restricții

Datele de intrare se citesc dintr-un fișier text cu următoarea structură:

- pe prima linie sunt scrise numerele n, m, p, q .
- pe următoarele n linii se dă conținutul panoului (câte m semne + și - pe fiecare linie, despartite prin spațiu).

Datele de ieșire se scriu într-un fișier text, sub forma unui sir de perechi (câte o perieche pe fiecare linie) reprezentând colțul din stânga sus (în cadrul panoului $n \times m$) unde este pus dreptunghiul $p \times q$ (Olimpiada judeteană Cluj, clasa a X-a, 1999).

Soluție

Parcurgem matricea pe linii, până la întâlnirea unui semn -. În acea poziție aplicăm submatricea $p \times q$. Acest lucru are ca efect eliminarea unor semne - și deplasarea spre dreapta-jos a tablei al altor semne -. Aplicăm operația de mai sus până în momentul în care nu mai există semne -. sau toate aceste semne au migrat în partea dreaptă sau de jos a matricei, unde submatricea $p \times q$ nu poate ajunge (deoarece acoperirea parțială a panoului nu este permisă).

```

Program Panou;
  type poz=record
    ii,jj:Byte
    end;
  var a:array[1..100,1..100] of Byte;
  mut:array[1..100] of poz;
  i,j,n,m,p,q,k1,k2,k:Byte;
  f:Text;

procedure Citeste;
  var c:Char;
begin
  Assign(f, 'PANOU.IN'); Reset(f); Readln(f,n,m,p,q);
  for i:=1 to n do
    begin
      for j:=1 to m do
        begin
          Read(f,c);
          if c='+' then a[i][j]:=1
          else a[i][j]:=0
        end;
      Readln(f)
    end;
  Close(f)
end;

```

```

procedure Move(F:TVector; start,stop:Word);
var y:Word;
begin
  for y:=start to stop do
  begin
    c[p]:=F[y];
    Dec(p)
  end
end;

procedure BinMerge(F:TVector; var v:Word; G:TVector;
                    var w:Word);
var k,j:Word;
begin
  k:=1 shl Trunc(Ln(v/w)/Ln(10));
  if g[w]<=f[v-k+1]
  then
  begin
    Move(F,v-k+1,v);
    v:=v-k
  end
  else
  begin
    BinSearch(G[w],F,v-k+2,v,j);
    Move(F,j+1,v);
    c[p]:=g[w];
    v:=j;
    w:=w-1;
    p:=p-1
  end
end;

procedure Afisare;
begin
  for i:=1 to lung do Write(c[i], ' ')
end;

begin
  citire;
  p:=n+m;
  lung:=p;
  while (n>0) and (m>0) do
    if n>m then BinMerge(a,n,b,m)
              else BinMerge(b,m,a,n);
  if n=0 then Move(b,1,m)
            else Move(a,1,n);
  Afisare
end.

```



P40. Sume

Se dă un sir strict crescător de n numere naturale ($1 < n \leq 1.000.000$). Să se decidă dacă orice număr natural $x \in \{1, 2, \dots, m\}$ se poate scrie ca sumă de termeni distincți din sirul dat.

Dacă descompunerea solicitată nu este posibilă, să se determine cel mai mare număr mai mic decât m , care nu poate fi scris ca sumă de termeni distincți din sirul dat.

Datele de intrare se vor citi din fișierul de tip text SUME.IN. Pe prima linie a fișierului sunt scrise numerele n și m . Pe următoarele n linii sunt scrise elementele sirului strict crescător ($m \leq 2.000.000.000$).

În cazul în care descompunerea este posibilă, în fișierul SUME.OUT se va scrie DA. În caz contrar, prima linie a fișierului va conține cuvântul NU, iar pe următoarea linie se va scrie cel mai mic număr mai mare decât m care nu poate fi scris ca sumă de termeni distincți din sirul dat. (Olimpiada județeană Cluj, clasa a XII-a, 1999)

Soluție.

Din cauza numărului mare de elemente ale sirului dat, nu se poate aplica programarea dinamică. În primul rând se observă că, dacă m este mai mare decât suma termenilor sirului dat, atunci nu putem obține descompunerea cerută. Iar dacă m este mai mic sau egal cu suma termenilor sirului atunci soluția se bazează pe următoarea propoziție:

Propoziție

Condițiile necesare și suficiente ca un număr natural m să se poată scrie ca sumă de numere dintr-un sir dat, sunt:

1. numărul 1 să aparțină sirului dat
2. fiecare număr mai mic decât m trebuie să fie cel mult egal cu suma celorlalte din fața lui, minus 1.

Demonstrarea propoziției o lăsăm pe seama cititorului.

Algoritmul constă în calcularea sumelor parțiale $\sum_{i=1}^k a_i$, $k = 1, \dots, N$. Dacă la un mo-

ment dat avem o valoare m mai mică decât suma parțială curentă, atunci m se poate scrie ca sumă de numere din sirul dat. Dacă ajungem la o sumă parțială care este mai mică decât următorul termen din sir minus unu, atunci nu avem soluție afirmativă și primul număr care nu se poate scrie ca sumă de termeni din sirul dat este egal cu suma parțială curentă plus unu.

Dacă un număr se poate scrie ca sumă de termeni din sirul dat, atunci se poate determina și care sunt aceștia.

```

Program sir_de_numere;
var n,m,nr,s:Longint;
f:Text;

```

```

for i:=1 to n do
begin
  for j:=1 to n do Read(f,a[i,j]);
  Readln(f);
end;
Close(f);
end;

Begin
  Citire;
  prod:=1;
  ProdTotal:=1;
  for k:=1 to n-1 do
begin
  l:=k;
  while (a[k,l]=0) and (l<=n) do Inc(l);
  if (l<=n) and (l>k)
  then
    for j := k to n do
    begin
      sch:=a[k,j]; a[k,j]:=a[l,j]; a[l,j]:=sch;
    end;
  for i:=k+1 to n do
    for j:=k+1 to n do
      a[i,j]:=a[i,k] * a[i,j]-a[k,j] * a[i,k];
  ProdTotal:=ProdTotal * ProdTotal;
  ProdTotal:=ProdTotal * a[k,k];
  prod:=prod * a[k,k];
end;
  ProdTotal:=ProdTotal div prod;
  Writeln('Determinantul este:',a[n,n] div ProdTotal);
End.

```

Intrare	Ieșire
4 2 2 1 4 1 2 1 3 3 1 2 1 1 2 3 2	9

P39. Interclasare binară

Să se interclaseze două siruri de numere întregi ordonate crescător.

Soluție

Principalul dezavantaj al metodelor de interclasare folosite mai sus este numărul mare ($m + n - 1$) de comparații efectuate (unde m și n sunt lungimile celor două siruri).

Să considerăm, de exemplu, cazul în care $m = 1$ și n este foarte mare. Atunci algoritmul de mai sus efectuează de fapt o căutare liniară a unui număr într-un sir de lungime n . Dacă numărul căutat este mai mare decât ultimul număr din sirul de lungime n , atunci avem de efectuat n comparații. Se știe însă că această căutare poate fi efectuată mai rapid prin *algoritmul de căutare binară*, care efectuează cel mult $\log(n)$ operații. Revenind la cazul general, să presupunem că $m < n$. Dacă cele m numere sunt distribuite uniform printre cele n , atunci algoritmul de interclasare binară efectuează aproximativ $m \cdot \log(n/m)$ comparații.

Să notăm cu A sirul de lungime n și cu B sirul de lungime m . Vrem să determinăm dacă b_1 aparține la a i -a secvență de lungime m din A . Începem prin a compara b_1 cu $a_{1,m}$. Dacă b_1 este mai mic decât acest element, atunci o căutare binară în primele m numere ale lui A va determina poziția lui b_1 . Dacă b_1 este mai mare sau egal decât $a_{m,m}$, atunci știm că primele m numere din A pot fi mutate în vectorul final. Problema s-a redus la interclasarea unui sir de dimensiune m cu un sir de dimensiune $n - m$.

```

Program InterclasareBinara;
const MaxN=100;
type TVector=array[1..MaxN] of Integer;
var a,b,c:TVector;
  n,m,p,i,j,lung:Word;
  f:Text;
procedure Citire;
begin
  Assign(f,'MERGE.IN'); Reset(f); Readln(f,n,m);
  for i:=1 to n do Read(f,a[i]);
  Readln(f);
  for j:=1 to m do Read(f,b[j]);
  Close(f)
end;
procedure BinSearch(x:Integer; F:TVector; start,stop:Word;
                     var j:Word);
begin
  j:=0;
  while start<=stop do
  begin
    j:=(start+stop) div 2;
    if x=F[j]
    then Exit
    else
      if x<F[j] then stop:=j-1
      else start:=j+1
  end;
  if j=0 then j:=stop
end;

```

```

Begin
    citeste;
    min:=65535;
    for i:=1 to n-k+1 do
        for j:=1 to m-L+1 do
            begin
                nr:=0;
                for c1:=i to i+k-1 do
                    for c2:=j to j+L-1 do nr:=nr+a[c1,c2];
                if min>nr then min:=nr
            end;
        for i:=1 to n-L+1 do
            for j:=1 to m-k+1 do
                begin
                    nr:=0;
                    for c1:=i to i+L-1 do
                        for c2:=j to j+k-1 do nr:=nr+a[c1,c2];
                    if min>nr then min:=nr
                end;
    Writeln('Nr. minim de metri cubi de gheata dislocati:',min)
End.

```

Intrare

Iesire

5 6
2 3
1 4 5 3 8 1
4 6 2 6 8 5
3 6 8 3 4 7
1 1 2 3 5 8
8 1 1 1 3 3

Nr. minim de metri cubi de gheata dislocati:9

P38. Calculul unui determinant

Să se calculeze valoarea unui determinant de ordinul n .

Restricții

Datele se citesc dintr-un fișier text având următoarea structură:

- pe prima linie este scris ordinul determinantului (n);
- pe următoarele n linii se găsesc valorile elementelor de pe fiecare linie a determinantului ([12]).

Soluție

Există mai multe metode de a calcula valoarea unui determinant. Dintre acestea amintim:

- dezvoltare cu ajutorul minorilor de ordin $n - 1, n - 2, \dots, 3, 2$. În acest caz numărul operațiilor elementare pentru a obține valoarea determinantului este:

$$N_1(n) = n! S - 1, \text{ unde}$$

$$S = 1 + \frac{1}{1!} + \frac{1}{2!} + \dots + \frac{1}{(n-1)!}$$

b) transformarea determinantului dat într-un determinant triunghiular (cu elementele de pe diagonala principală egale cu zero). În acest caz numărul de operații elementare necesare este:

$$N_2(n) = \frac{1}{3}(n-1)[2n(2n-1)+3]$$

c) aplicând regula dreptunghiului. Acest procedeu necesită:

$$N_3(n) = (n-1)^2(n+1)$$

operații, deci este cel mai eficient dintre toți algoritmii prezenți mai sus. De aceea noi o să îl prezentăm pe acesta.

Fie determinantul:

$$\Delta_n = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix}$$

Efectuăm $n - 1$ pași. La pasul k , un element oarecare se va calcula după formula:

$$a_{ij}^{(k)} = a_{kk}^{(k-1)} \cdot a_{ij}^{(k-1)} - a_{ik}^{(k-1)} \cdot a_{kj}^{(k-1)}$$

După efectuarea acestor pași, valoarea determinantului se va calcula într-un mod asemănător cu cel de la punctul b), adică se înmulțesc elementele de pe diagonala principală, iar rezultatul se împarte la următorul produs:

$$a_{11}^{(n-1)} \cdot [a_{22}^{(1)}]^{n-2} \cdot \dots \cdot [a_{n-2,n-2}^{(n-3)}]^2 \cdot a_{n-1,n-1}^{(n-2)}$$

După simplificări valoarea determinantului va fi:

$$\Delta_n = \frac{a_{nn}^{(n-1)}}{a_{11}^{(n-2)} \cdot [a_{22}^{(1)}]^{n-3} \cdot \dots \cdot a_{n-2,n-2}^{(n-3)}}$$

Program CalculDeterminant;

```

const MaxN=10;
var n,i,j,k,l:Byte;
    a:array[1..MaxN,1..MaxN] of Longint;
    f:Text;
    prod,ProdTotal,sch:Longint;
procedure Citire;
begin
    Assign(f,'DETERMIN.IN'); Reset(f);
    Readln(f,n);

```

```

while not final and not blocaj do
begin
  blocaj:=true;
  for i:=1 to n do
    for j:=1 to m do
      if a[i,j]=0
      then
        if a[i,j+1]+a[i,j-1]+a[i-1,j]+a[i+1,j]>=2
        then
          begin
            b[i,j]:=1;
            blocaj:=false
          end;
      if blocaj
      then
      begin
        Write('Desertul nu poate fi acoperit cu paduri!');
        Readln;
        Exit
      end;
    for i:=1 to n do
      for j:=1 to m do
        if b[i,j]=1
        then
        begin
          a[i,j]:=1;
          b[i,j]:=0
        end;
      Inc(nr)
    end;
  Write('Numarul minim de zile după care desertul ');
  Writeln('va fi impadurit este:',nr)
End.

```

Intrare

```

5 6
1 0 1 0 1 1
1 0 0 0 0 0
1 0 0 0 0 0
0 1 0 0 0 0
0 0 1 0 0 0

```

Ieșire

Numarul minim de zile după care desertul
va fi impadurit este:9

Intrare

```

3 3
1 0 0
0 0 0
1 0 0

```

Ieșire

Desertul nu poate fi acoperit cu paduri!

Desertul**P37. Nautilus**

La întoarcerea din călătoria efectuată la polul Sud, căpitanul *Nemo* trebuia să străbată cu submarinul *Nautilus* întreaga întindere de gheăță care îl despărțea de oceanul inghețat, bineînteles acest lucru făcându-se sub banchiza de gheăță. După cum relatează *Jules Verne*, la un moment dat submarinul a fost prins într-un fel de grotă de gheăță, grotă care datorită temperaturii scăzute se transformase într-o veritabilă închisoare pentru cei din submarin. De aceea ei s-au decis să spargă gheăța de sub submarin pentru a putea ieși din capcană.

Dacă reprezentăm fundul grotei sub forma unui dreptunghi de $N \times M$ m², iar submarinul sub forma unui dreptunghi de $K \times L$ m², și cunoscând grosimea ghetii fiecărui metru pătrat din suprafața fundului grotei, se cere să se determine numărul minim de metri cubi de gheăță dislocați de echipajul submarinului pentru a ieși din capcană.

Observație

Gheăța dislocată formează un dreptunghi congruent cu cel al submarinului și are laturile paralele cu laturile grotei ([1]).

Restricții

Datele se citesc dintr-un fișier text, având următoarea structură:

- pe prima linie sunt scrise dimensiunile grotei;
- pe a doua linie sunt scrise dimensiunile submarinului;
- pe următoarele n linii este dată o matrice reprezentând grosimea ghetii.

Soluție

Calculăm numărul de metri cubi de gheăță care trebuie dislocați din fiecare dreptunghi de dimensiuni $K \times L$ (dimensiunile submarinului), dreptunghi care este paralel cu laturile fundului grotei. Cantitatea minimă de gheăță dislocată este reținută în variabila min.

Program Nautilus;

```

const MaxN=100;
var a:array[1..MaxN,1..MaxN] of Word;
  n,m,i,j,k,L,c1,c2:Byte;
  nr,min:Word;
procedure citeste;
  var f:Text;
begin
  Assign(f,'GHEATA.IN'); Reset(f); Read(f,n,m); Read(f,k,L);
  for i:=1 to n do
    for j:=1 to m do Read(f,a[i,j]);
  Close(f)
end;

```

```

Inc(NrSubMultimi);
NrRamase:=NrRamase-max;
if di[poz]>cul[poz]
then
  for i:=1 to n do
    if a[i].diametru=poz
    then
      begin Dec(di[poz]); Dec(cul[a[i].culoare]) end
    else
  end;
  for i:=1 to n do
    if a[i].culoare=poz
    then
      begin Dec(cul[poz]); Dec(di[a[i].diametru]) end
  end;
Writeln('Numarul minim de submultimi este:',NrSubMultimi);
End.

```

Intrare Ieșire

```

4           Numarul minim de submultimi este:2
1 1
1 2
1 2
3 1

```

P36. Copaci în deșert

Datorită faptului că deșertul *Sahara*, se extinde tot mai mult în fiecare an, statele lumii s-au hotărât să reducă acest proces. Chiar mai mult, specialiști în geodezie au demonstrat că la câțiva metri sub stratul de nisip se ascund zăcăminte importante de apă și au hotărât împădurirea acestui deșert.

Pentru simplificare, să considerăm reprezentarea hărții deșertului ca fiind o matrice de dimensiuni $n \times m$. În fiecare căsuță a matricei poate fi plantat un singur copac. Se mai știe că datorită condițiilor climatice extreme un copac plantat nu rezistă decât dacă este înconjurat de cel puțin doi copaci plantați anterior (în zilele anterioare). Știind că în fiecare zi sunt plantați copaci (respectând condiția anterioară) să se determine dacă deșertul poate fi împădurit sau nu. În caz afirmativ, se cere numărul minim de zile în care se poate realiza împădurirea.

Observație

- deșertul conține oaze;
- în fiecare căsuță a matricei există doar un singur copac;
- două poziții se numesc vecine dacă ele sunt învecinate pe orizontală sau verticală.

Restricții

Datele se citesc dintr-un fișier de intrare care are următoarea structură:

- pe prima linie sunt scrise dimensiunile matricei;
- pe următoarele n linii este scrisă matricea inițială (0 reprezintă poziție neocupată de copaci, 1 poziție ocupată de copaci în oaze) ([1]).

Soluție

Determinăm, pentru fiecare zi, copaci care pot fi plantați astfel încât fiecare să fie înconjurat de cel puțin doi copaci plantați în zilele anterioare (sau existenți în oaze). Dacă într-o zi nu este posibil să se planteze nici un copac, deșertul nu poate fi împădurit.

```

Program CopaciInDesert;
const MaxN=100;
var a,b:array[0..MaxN+1,0..MaxN+1] of 0..1;
n,m,i,j,nr:Byte;
blocaj:Boolean;
procedure citeste;
  var f:Text;
begin
  Assign(f,'DESSERT.IN'); Reset(f); Readln(f,n,m);
  for i:=1 to n do
  begin
    for j:=1 to m do
      begin Read(f,a[i,j]); b[i,j]:=0 end;
      Readln(f)
    end;
    Close(f)
  end;
function final:Boolean;
  var c1,c2:Byte;
begin
  for c1:=1 to n do
    for c2:=1 to m do
      if a[c1,c2]=0
      then
        begin
          final:=false; Exit
        end;
      final:=true
  end;
begin
  citeste;
  for i:=0 to n+1 do begin a[i,0]:=0; a[i,m+1]:=0 end;
  for j:=0 to m+1 do begin a[0,j]:=0; a[n+1,j]:=0 end;
  nr:=0; blocaj:=false;

```

Observăm mai întâi că pentru a forma un sir de lungime maximă trebuie să includem în acel sir toți divizorii numărului n .

Apoi, între oricare doi divizori ai lui n (care nu sunt numere consecutive în sirul numerelor naturale) mai putem introduce încă un număr (care nu este divizor al lui n) deoarece în acest fel dintre oricare doi termeni consecutivi ai sirului cel puțin unul va fi divizor al lui n ([1]).

```
Program SirDeLungimeMaxima;
const MaxN=10000;
var a:array[1..MaxN] of Word;
    n,i,nr:Word;
begin
  Write('Introduceti n:'); Readln(n);
  a[1]:=1;
  nr:=1;
  for i:=2 to (n+1) div 2 do
    if n mod i=0 then begin Inc(nr); a[nr]:=i end
    else
      if n mod (i-1)=0 then begin Inc(nr); a[nr]:=i end;
    if not Odd(n) then begin Inc(nr); a[nr]:=n div 2+1 end;
    Inc(nr); a[nr]:=n;
    for i:=1 to nr do Write(a[i], ' ');
end.
```

Intrare Ieșire
n=35 1 2 5 6 7 8 35

P35. Multime de discuri

Se dă o mulțime de n discuri. Fiecare disc este caracterizat de două elemente: diametru (măsurat în cm) și culoare.

Să se împartă această mulțime într-un număr minim de submulțimi cu proprietatea că în fiecare submulțime se găsesc, fie discuri având aceeași culoare, fie discuri având același diametru ([1]).

Soluție

Vom determina la fiecare pas o submulțime maximală de discuri în raport cu inclusiunea și care are următoarele proprietăți:

- 1) conține doar discuri având același diametru sau având aceeași culoare;
- 2) este maximală din punct de vedere al numărului de elemente pe care le conține.

După ce această submulțime a fost determinată, ea se elimină din mulțimea de discuri și operația anterioară se repetă pentru mulțimea rămasă până în momentul în care mulțimea devine vidă.

```
program MultimeDeDiscuri;
const MaxN=1000;
type moneda=record
  diametru,culoare:Word
end;
var a:array[1..MaxN] of moneda;
  di,cul:array[1..MaxN] of Word;
  NrSubMultimi,poz,n,i,max,NrRamase:Word;
procedure citeste;
var f:Text;
begin
  Assign(f,'DISCURI.IN'); Reset(f); Readln(f,n);
  for i:=1 to n do begin di[i]:=0; cul[i]:=0 end;
  for i:=1 to n do
    begin
      Read(f,a[i].diametru,a[i].culoare);
      Inc(di[a[i].diametru]);
      Inc(cul[a[i].culoare]);
    end;
  Close(f);
end;
begin
  citeste;
  NrSubMultimi:=0;
  NrRamase:=n;
  while NrRamase>0 do
  begin
    max:=0; poz:=0;
    for i:=1 to n do
      if di[i]>0
      then
        if cul[i]>0
        then
          if di[i]>cul[i]
          then
            if max<di[i] then begin max:=di[i]; poz:=i end
            else
              if max<cul[i] then begin max:=cul[i]; poz:=i end
              else
                if max<di[i] then begin max:=di[i]; poz:=i end
                else
                  if cul[i]>0
                  then
                    if max<cul[i] then begin max:=cul[i]; poz:=i end;
  end;
```

Exemplu:

Intrare	Ieșire
2	$a+b-c*d$
+ 0	
- 0	
* 1	
a + (b - c) * d	

Discuție

Încercați să simplificați algoritmul de mai sus pentru expresii booleene.

În practică pe lângă operatorii binari (+, -, *, /, ^) mai există și operatori unari (și/sau funcții) precum Ln, Exp, Sqrt, Sin, Cos, ... Încercați să construiți arborele atașat unei astfel de expresii.

Generalizați expresia pentru operatori ternari, n -ari.

Să se calculeze derivata unei funcții de o singură variabilă. Încercați, pe baza arborelui atașat expresiei să calculați arborele atașat derivatei. De exemplu, pentru expresia $E = x^2 + \ln(x)$ programul trebuie să vă returneze derivata: $2x + 1/x$.

Dându-se o expresie, aduceți-o la formă canonica, adică fără paranteze. În acest scop, construiți arborele atașat ei, apoi parcurgeți-l în postordine pentru construirea (sub)expresiilor canonice atașate unui nod. Veți avea nevoie în fiecare nod al arborelui de un sir în care să rețineți (sub)expresia canonica atașată lui.

Încercați să simplificați expresia aritmetică dată, astfel încât ea să conțină un număr minim de operatori. De exemplu $a*c+b*c$ poate fi simplificată la $(a+b)*c$.

P2. Evaluare expresie

Se dă o expresie aritmetică care conține operatorii $-$, $+$, $*$, iar operanții sunt litere mici. Dându-se valorile operanților, să se evaluateze expresia.

Restricții

Datele de intrare se citesc dintr-un fișier text care are următoarea structură:

- pe prima linie se găsește expresia dată sub forma unui sir de caractere;
- pe următoarele linii se dau valorile operanților, sub forma unor perechi: operand valoare.

Soluție

Modul în care este construit arborele atașat expresiei este aproape identic cu cel de la problema precedentă. Ceea ce apare nou, este câmpul **val**, atașat unui nod, care conține valoarea (sub)expresiei care are ca rădăcină nodul respectiv. Este evident că în rădăcina arborelui vom avea valoarea întregii expresii.

Calculul valorilor câmpului **val** se face în postordine. Inițial doar nodurile terminale (care conțin operanți) au valori, apoi prin intermediul procedurii **Eval** (care realizează parcurgerea în postordine) se evaluatează mai întâi (sub)expresia din partea stângă a unui operand, apoi (sub)expresia din partea dreaptă și în final se combină cele două rezultate (în funcție de operator).

```
Program EvaluareExpresie;
type TArb=^art;
  art=record
    inf:Char;
    val:Integer;
    st,dr:TArb
  end;

var radacina:TArb;
  StareHeap:Pointer;
  sir:string;
  f:Text;
  i:Byte;
  c:Char;
  prioritate:array[Char] of Shortint;
  valoare:array[Char] of Integer;
  max:Integer;

procedure Citire;
  var v:Byte;
begin
  Fillchar(prioritate, 255, 255);
  Assign(f, 'EXPRESIE.IN'); Reset(f);
  max:=1;
  prioritate['+]:=0;
  prioritate['-]:=0;
  prioritate['*]:=1;
  Readln(f,sir);
  while not Seekeof(f) do
  begin
    Readln(f,c,v); valoare[c]:=v
  end;
  Close(f)
end;

procedure Expresie(prior:Byte; var rad:TArb); forward;

procedure UrmatorulCaracter;
begin
  c:=sir[i];
  Inc(i)
end;
```

cădura recursivă Expresie va returna arborele atașat unei expresii. Pe lângă referința la tipul TArb, această procedură mai conține un parametru prior care rezolvă elegant necesitatea existenței funcției pentru fiecare tip de operator. Procedura InOrdine afișează arborele construit în inordine (mai întâi subarborele stâng, apoi rădăcina urmată de subarborele drept).

```

Program ExpresieGeneralizata;
  type TArb=^art;
    art=record
      inf:Char;
      st, dr:TArb
    end;
  var radacina:TArb;
    sir:string;
    f:Text;
    i:Byte;
    c:Char;
    prioritate:array[Char] of Shortint;
    max:Integer;
  procedure Citire;
    var n, pr:Byte;
  begin
    Fillchar(prioritate,255,255);
    Assign(f,'EXPRESIE.IN'); Reset(f); Readln(f,n);
    max:=-1;
    for i:=1 to n do
    begin
      Readln(f,c,pr);
      prioritate[c]:=pr;
      if pr>max then max:=pr
    end;
    Readln(f,sir);
    Close(f)
  end;
  procedure Expresie(prior:Byte; var rad:TArb); forward;
  procedure UrmatorulCaracter;
  begin
    c:=sir[i];
    Inc(i)
  end;
  procedure factor(var rad:TArb);
  begin
    UrmatorulCaracter;
    if c='(' then Expresie(0,rad)

```

```

      else
      begin
        New(rad);
        rad^.st:=nil;
        rad^.dr:=nil;
        rad^.inf:=c
      end
    end;
  procedure Expresie(prior:Byte; var rad:TArb);
    var p:TArb;
  begin
    if prior=max+1
    then
    begin
      factor(rad);
      UrmatorulCaracter
    end
    else
    begin
      Expresie(prior+1,rad);
      p:=rad;
      while prioritare[c]=prior do
      begin
        New(rad); rad^.inf:=c; rad^.st:=p;
        expresie(prior+1,rad^.dr);
        p:=rad
      end
    end
  end;
  procedure InOrdine(rad:TArb);
  begin
    if rad<>nil
    then
    begin
      InOrdine(rad^.st);
      Write(rad^.inf);
      InOrdine(rad^.dr)
    end
  end;
  Begin
    Citire;
    i:=1;
    Expresie(0,radacina);
    InOrdine(radacina)
  End.

```

```

procedure factor(var rad:TArb);
begin
  UrmatorulCaracter;
  if c= '(' then Expresie(0,rad)
    else
      begin
        New(rad);
        rad^.st:=nil;
        rad^.dr:=nil;
        rad^.inf:=c;
        rad^.val:=valoare[c]
      end
end;

procedure Expresie(prior:Byte; var rad:TArb);
var p:TArb;
begin
  if prior=max+1
  then begin factor(rad); UrmatorulCaracter end
  else
    begin
      Expresie(prior+1,rad); p:=rad;
      while prioritata[c]=prior do
        begin
          New(rad);
          rad^.inf:=c;
          rad^.st:=p;
          expresie(prior+1,rad^.dr);
          p:=rad
        end
    end
end;
procedure Eval(rad:TArb);
begin
  if rad^.st<>nil
  then
    begin
      Eval(rad^.st);
      Eval(rad^.dr);
      case rad^.inf of
        '+':rad^.val:=rad^.st^.val+rad^.dr^.val;
        '-':rad^.val:=rad^.st^.val-rad^.dr^.val;
        '*':rad^.val:=rad^.st^.val*rad^.dr^.val
      end
    end
end;

```

```

Begin
  Citire; i:=1;
  Mark(stareHeap);
  Expresie(0,radacina);
  Eval(radacina);
  Writeln('Expresia are valoare:',radacina^.val);
  Release(stareHeap)
End.

```

Exemplu:

Intrare

a+(b-c)*d
a 1
b 5
c 2
d 3

Ieșire

Expresia are valoarea: 10

P3. Cod cu trei adrese

Pentru evaluarea unei expresii este nevoie de locații de memorie pentru stocarea unor rezultate parțiale. În primul rând este clar că este nevoie de memorie pentru fiecare operand din expresie. Apoi pentru calcularea unei (sub)expresii de forma $E_1 \otimes E_2$ (unde \otimes desemnează un operator oarecare) avem nevoie de o nouă locație de memorie unde vom stoca rezultatul obținut prin combinarea rezultatelor de la adresele unde sunt stocate valorile lui E_1 și E_2 .

Un cod cu trei adrese constă dintr-un cuadruplu:
adresa operator adresal adresa2,
unde adresa este adresa la care se stochează rezultatul obținut prin aplicarea operatorului operator asupra valorilor de la adresele adresal și adresa2. Ultimele două adrese pot fi și operanzi.

Soluție

Construim arborele atașat expresiei ca în cazul problemelor precedente, apoi îl parcugem în postordine (în procedura Afiseaza). Este evident că fiecărui nod trebuie să îi mai atașăm un câmp (adr) în care reținem adresa la care memorăm rezultatul (sub)expresiei atașate lui. Variabila globală adresa reține ultima adresa de memorie folosită (acestea sunt numerotate de la 1 la 255). Astfel, din păcate, unele adrese, nu vor fi disponibile, deși nu mai sunt utilizate. Încercați să optimizați algoritmul astfel încât să folosească un număr minim de adrese.

```

Program ExpresieGeneralizata;
type TArb=^art;
art=record
  inf:Char;
  adr:Integer;
  st,dr:TArb
end;
var radacina:TArb;
StateHeap:Pointer;
sir:string;
f:Text;
i,adresa:Byte;
c:Char;
prioritate:array[Char] of Shortint;
max:Integer;

procedure Citire;
var n,pr:Byte;
begin
  Fillchar(prioritate,255,255);
  Assign(f,'expresie.in'); Reset(f);
  max:=1;
  prioritate['+']:=0;
  prioritate['-']:=0;
  prioritate['*']:=1;
  Readln(f,sir);
  Close(f);
end;

procedure Expresie(prior:Byte; var rad:TArb); forward;

procedure UrmatorulCaracter;
begin c:=sir[i]; Inc(i) end;

procedure factor(var rad:TArb);
begin
  UrmatorulCaracter;
  if c='(' then Expresie(0,rad)
  else
    begin
      New(rad);
      rad^.st:=nil;
      rad^.dr:=nil;
      rad^.inf:=c;
      rad^.adr:=0
    end
end;

```

```

procedure Expresie(prior:Byte; var rad:TArb);
var p:TArb;
begin
  if prior=max+1
  then begin factor(rad); UrmatorulCaracter end
  else
    begin
      Expresie(prior+1,rad);
      p:=rad;
      while prioritate[c]=prior do
        begin
          New(rad);
          rad^.inf:=c;
          rad^.st:=p;
          expresie(prior+1,rad^.dr);
          p:=rad
        end
    end
end;

procedure Afiseaza(rad:TArb);
begin
  if rad^.st<>nil then Afiseaza(rad^.st);
  if rad^.dr<>nil then Afiseaza(rad^.dr);
  if rad^.st<>nil
  then
    begin
      Write(adresa+1,' ');
      Write(rad^.inf,' ');
      if rad^.st^.adr=0 then Write(rad^.st^.inf,' ')
      else Write(rad^.st^.adr,' ');
      if rad^.dr^.adr=0 then Writeln(rad^.dr^.inf,' ')
      else Writeln(rad^.dr^.adr,' ');
      Inc(adresa);
      rad^.adr:=adresa
    end
end;

Begin
  Citire; i:=1;
  Expresie(0,radacina);
  Afiseaza(radacina)
End.

Exemplu:
Intrare           Iesire
a+(b-c)*d       1 - c b
                  2 * 1 d
                  3 + a 2

```

VI. ALGORITMI EURISTICI

P1. Drum hamiltonian

Se consideră un graf neorientat, complet, cu n noduri. Fiecarei muchii i se atribuie un cost pozitiv, numit și distanță. Se cere să se determine un drum hamiltonian de cost minim, unde prin costul unui drum se înțelege suma costurilor muchiilor componente.

Restricții

Datele de intrare se citesc dintr-un fișier text având următoarea structură:

- pe prima linie este scris numărul de noduri;
- pe următoarele n linii sunt scrise distanțele dintre noduri, sub forma unei matrice de dimensiuni $n \times n$. Două elemente de pe aceeași linie sunt despărțite în fișier printr-un singur spațiu.

Soluție

Problema este cunoscută în literatura de specialitate ca fiind NP-completă. O demonstrație a acestei afirmații se găsește în [14]. Deci, cu alte cuvinte, nu se cunoaște un algoritm cu timp de lucru polinomial care rezolvă problema. În continuare prezentăm un algoritm euristic, care va furniza în multe situații (dar nu în toate) soluția optimă.

Începem prin a construi drumul hamiltonian, plecând dintr-un nod oarecare. La fiecare pas, alegem un nou nod pentru care distanța de la el până la ultimul nod din drumul deja construit este minimă.

```
Program hamiltonian;
const MaxN=10;
var dist:array[1..MaxN,1..MaxN] of Word;
    drum:array[1..MaxN] of 1..MaxN;
    f:Text;
    n,i,j,min,costMin,next:Word;
    mult:set of 1..MaxN;

procedure Citire;
begin
    Assign(f,'HAMILT.IN'); Reset(f);
    Readln(f,n);
    for i:=1 to n do
        for j:=1 to n do Read(f,dist[i,j])
    end;
```

Algoritmul euristic

```
procedure Calcul;
begin
    drum[1]:=1; mult:=[1];
    for i:=2 to N do
        begin
            min:=65535;
            for j:=1 to N do
                if (min>dist[drum[i-1],j]) and (not(j,in mult))
                then begin min:=dist[drum[i-1],j]; next:=j end;
            costMin:=costMin+min; drum[i]:=next; mult:=mult+[next];
        end;
end;

procedure Afiseaza;
begin
    Writeln('Costul drumului minim este:', costMin);
    Write('Drumul este:');
    for i:=1 to N do Write(drum[i], ' ');
end;
```

Begin

```
Citire;
Calcul;
Afiseaza
End.
```

Intrare

```
4
0 2 1 1
2 0 2 3
1 2 0 5
1 3 5 0
```

Ieșire

```
Costul drumului minim este:6
Drumul este: 1 3 2 4
```

Discuție

Un set de date pentru care algoritmul de mai sus nu funcționează este următorul:

```
4
0 1
```

Se pot face însă câteva optimizări:

- Se aleg, pe rând, toate vîrfurile ca vîrfuri de plecare. Acest lucru va mări complexitatea algoritmului de la $O(N^2)$ la $O(N^3)$.
- La un pas se adaugă nu doar căte un nod, ci lanțuri de căte două noduri cu suma distanțelor minimă. Astfel complexitatea algoritmului crește de la $O(N^2)$ la $O(N^3)$. Adăugarea lanțurilor de lungime k , în loc de lanțuri de lungime 1 va mări complexitatea la $O(N^{k+1})$, dar va îmbunătăți cu mult performanțele algoritmului. Este evident că un algoritm de complexitate $O(N^k)$ furnizează întotdeauna soluția optimă.

- c) Adăugarea nodurilor se poate face nu numai la un sigur capăt al drumului, ci la ambele capete. În această situație este mai convenabil ca soluția să se construiască într-o listă dinamică decât într-un vector.
- d) În [18] este prezentat un algoritm heuristic mai bun: drumul se construiește sub forma unei mulțimi de muchii. La fiecare pas se adaugă muchia de cost minim care nu generează un ciclu și care împreună cu alte două deja adăugate nu au un același vârf comun. Din păcate pentru testul :

6
 0 3 10 11 7 25
 3 0 6 12 8 26
 10 6 0 9 4 20
 11 12 9 0 5 15
 7 8 4 5 0 18
 25 26 20 15 18 0

rezultatul obținut este 1 2 3 4 5 6 care este mai mare decât 1 2 3 6 4 : 5.

P2. Colorarea grafurilor

Se dă un graf neorientat cu n noduri. Se cere să se determine numărul minim de culori necesare pentru a colora nodurile grafului dat, astfel încât oricare două vârfuri legate printr-o muchie să fie colorate cu culori diferite.

Restricții

Datele de intrare se citesc dintr-un fișier text având următoarea structură:

- pe prima linie se dau numărul de noduri;
- pe următoarele n linii se dau legăturile dintre noduri sub forma unei matrice $n \times n$ (1 dacă nodurile respective sunt legate, și 0 în caz contrar). Două elemente de pe aceeași linie sunt despărțite în fișier printr-un spațiu.

Soluție

Colorăm cu o culoare câte noduri putem, apoi alegem o altă culoare și repetăm procesul. Algoritmul se oprește când au fost colorate toate nodurile.

```
Program ColoreazaNoduri;
const MaxN=100;
var a:array[1..MaxN,1..MaxN] of Byte;
    culoare:array[1..MaxN] of Byte;
    { pe pozitia i avem culoarea nodului i }
    n,i,j,NrMin:Byte; f:Text;
procedure Citire;
begin
  Assign(f,'INTERN.IN'); Reset(f);
  Readln(f,N);
end;
```

```
for i:=1 to n do
  for j:=1 to n do Read(f,a[i,j])
end;

procedure Calcul;
  function NeLegat(v,c,start,stop:Byte):Boolean;
    var g:Byte;
    begin
      for g:=start to stop do
        if (culoare[g]=c) and (a[v,g]=1)
          then begin NeLegat:=false; Exit end;
      NeLegat:=true;
    end;
begin
  NrMin:=0;
  for i:=1 to n do
    if culoare[i]=0
    then
      begin
        Inc(NrMin); culcare[i]:=NrMin;
        for j:=i+1 to n do
          if (culoare[j]=0) and (a[i,j]=0)
            and NeLegat(j,NrMin,i+1,j-1) then culoare[j]:=NrMin
        end
      end;
end;

procedure Afisare;
begin
  Writeln('Numarul minim de culori este:',NrMin);
  for i:=1 to n do
    Writeln('Culoarea nodului ',i,' este:',culoare[i]);
end;
```

```
Begin
  Citire;
  Calcul;
  Afisare
End.
```

Intrare

```
5
0 1 1 0 0
1 0 0 1 0
1 0 0 0 1
0 1 0 0 0
0 0 1 0 0
```

Ieșire

```
Numarul minim de culori este:2
Culoarea nodului 1 este:1
Culoarea nodului 2 este:2
Culoarea nodului 3 este:2
Culoarea nodului 4 este:1
Culoarea nodului 5 este:1
```

Discuție

Algoritmul nu funcționează optim pentru cazul:

```
5
0 0 0 0 1
0 0 1 1 0
0 1 0 0 1
0 1 0 0 1
1 0 1 1 0
```

Deoarece nodurile 1 și 2 sunt colorate cu culoarea 1, nodurile 3 și 4 cu culoarea 2, iar nodul 5 cu culoarea 3, în timp ce soluția optimă ar fi constat din colorarea nodurilor 1, 3 și 4 cu culoarea 1, iar nodurile 2 și 5 cu culoarea 2.

P3. Intern stabilitate

Se dă un graf neorientat cu N noduri. Se cere să se determine o submulțime maximă de noduri cu proprietatea că oricare două noduri din ea nu sunt legate printr-o muchie.

Restricții

Dătele de intrare se citesc dintr-un fișier text având următoarea structură:

- pe prima linie se dau numărul de noduri;
- pe următoarele n linii se dau legăturile dintre noduri sub forma unei matrice $n \times n$ (1 dacă nodurile respective sunt legate, și 0 în caz contrar). Două elemente de pe aceeași linie sunt despărțite în fișier printr-un spațiu.

Soluție

Aveam de a face din nou cu o problemă NP-completă. Un algoritm care furnizează soluție optimă este cu timp de lucru exponențial și l-am implementat în capitolul Diverse. Un algoritm heuristic de complexitate $O(N^2)$ este următorul: eliminăm la fiecare pas nodul de grad maxim din graful curent. Evident, la urmă vor rămâne doar noduri de grad 0, care constituie o soluție a problemei.

Program InternStabilitate;

```
const MaxN=10;
var a: array[1..MaxN,1..MaxN] of Byte;
grad:array[1..MaxN] of Byte; { gradele nodurilor }
MCoIntStab:set of 1..MaxN;
{ Nodurile care nu fac parte din componenta intern stabila }
N,i,j,Nr:Byte;
f:Text;
procedure Citire;
begin
  Assign(f,'INTERN.IN'); Reset(f);
  Readln(f,N);
end;
```

```
for i:=1 to n do
  for j:=1 to n do
    begin Read(f,a[i,j]); Inc(grad[i],a[i,j]) end
  end;

function DeterminaMax:Byte;
{ Determina nodul (din cele ramase) care are gradul maxim }
begin
  max:=0; v:=0;
  for c:=1 to N do
    if not (c in MCoIntStab) and (max<grad[c])
    then begin max:=grad[c]; v:=c end;
  DeterminaMax:=v
end;

procedure Calcul;
var v:Byte;
begin
  Nr:=0; MCoIntStab:=[]; v:=DeterminaMax;
  while v<>0 do
  begin
    MCoIntStab:=MCoIntStab+[v];
    for i:=1 to N do
      if (not (i in MCoIntStab)) and (a[i,v]=1)
      then Dec(grad[i]);
    v:=DeterminaMax
  end;
end;

procedure Afisare;
begin
  Writeln('Nodurile din componenta intern stabila sunt:');
  for i:=1 to N do if not (i in MCoIntStab) then Write(i,' ');
end;

Begin
  Citire;
  Calcul;
  Afisare
End.
```

Intrare	Ieșire
5	Nodurile din componenta intern stabila sunt:
0 1 1 0 0	4,5
1 0 0 1 0	
1 0 0 0 1	
0 1 0 0 0	
0 0 1 0 0	

Discuție

Algoritmul nu funcționează optim pentru cazul de mai sus, în care o componentă intern stabilă maximală ar fi fost {1, 4, 5}.

P4. Problema rucsacului (varianta discretă)

Avem un rucsac de capacitate K, și care trebuie umplut cu obiecte având greutate și valoare dată. Să se determine o soluție în care valoarea obiectelor din sac este maximă.

Restricții

Datele de intrare se citesc dintr-un fișier text având următoarea structură:

- pe prima linie sunt scrise numărul de obiecte pe care le avem la dispoziție și capacitatea rucsacului (N, K);
- pe următoarele N linii sunt scrise perechi de numere reprezentând greutățile și valorile obiectelor.

Soluție

Ordonăm obiectele crescător în funcție de raportul valoare/greutate. Acest raport îl mai numim și utilitate sau eficiență. Apoi începem să încărcăm obiectele cu cea mai mare utilitate. Algoritmul se termină când nu mai începe nici un obiect în rucsac.

```
Program Rucsac;
const MaxN=100;
var greutate, valoare: array[1..MaxN] of Word;
    a: array[1..MaxN] of Word;
    eficienta: array[1..MaxN] of Real;
    N, i, K, Nr, GreutateInSac: Word;
    f: Text;

procedure Citire;
begin
  Assign(f, 'RUCSAC.IN'); Reset(f); Readln(f, n, k);
  for i:=1 to n do
  begin
    Readln(f, greutate[i], valoare[i]);
    eficienta[i]:=valoare[i]/greutate[i]
  end;
  Close(f)
end;

procedure Sort(l, r: Integer);
var i, j: Word;
    x, y: Real;
    yi: Word;
begin
  i:=l; j:=r;
```

```
x:=eficienta[(l+r) div 2];
repeat
  while eficienta[i]<x do i:=i+1;
  while x<eficienta[j] do j:=j-1;
  if i>=j
  then
  begin
    y:=eficienta[i];
    eficienta[i]:=eficienta[j]; eficienta[j]:=y;
    yi:=valoare[i];
    valoare[i]:=valoare[j]; valoare[j]:=yi;
    yi:=greutate[i];
    greutate[i]:=greutate[j]; greutate[j]:=yi;
    Inc(i); Dec(j)
  end;
until i>j;
if l<j then Sort(l, j);
if i<r then Sort(i, r)
end;

procedure Calcul;
begin
  Sort(1, N);
  Nr:=0; GreutateInSac:=0;
  for i:=N downto 1 do
    if greutate[i]+GreutateInSac<=k
    then
      begin Inc(Nr); Inc(GreutateInSac, greutate[i]); a[Nr]:=i end
  end;
```

```
procedure Afiseaza;
begin
```

```
  Writeln(k-GreutateInSac, ' kg nu au fost umplute in sac.');
  Writeln('Obiecte introduse au greut. si val. urmatoare:');
  for i:=1 to Nr do Writeln(greutate[a[i]], ', ', valoare[a[i]])
end;
```

```
Begin
  Citire;
  Calcul;
  Afiseaza
End.
```

Intrare

```
4 6
4 1
7 100
3 2
1 1
```

Ieșire

```
2 kg nu au fost umplute in sac.
Obiectele introduse au greut. si val. urmatoare:
```

```
1 1
```

Discuție

Algoritmul nu funcționează optim pentru cazul:
 În acest caz soluția optimă ar fi constat din adăugarea celor două obiecte de greutate 3.

O optimizare constă în adăugarea în rucsac a către două obiecte deodată.

3 6
 4 5
 3 3
 3 3

P5. Determină bile albe

Într-o cutie se află n bile dintre care k bile albe sunt numerotate cu numere distincte din mulțimea $\{1 \dots n\}$, restul sunt negre. Doi jucători participă la următorul joc: unul dintre ei trebuie să ghicească numerele de ordine ale bilelor albe, punând întrebări în felul următor: căte bile albe conține submulțimea $\{i_1, i_2, \dots, i_g\}$?

Jucătorul care cunoaște culorile bilelor trebuie să răspundă la aceste întrebări.

Să se elaboreze o strategie pentru jucătorul care trebuie să ghicească numerele de ordine ale bilelor albe, răspunsurile la întrebări fiind citite de la tastatură ([1], [11]).

Soluție

Se pună întrebarea: oare se poate rezolva această problemă după cel mult n întrebări de genul: $\{i\}?$ ($1 \leq i \leq n$). Răspunsul este: da, dar această strategie nu este eficientă; de aceea vom aplica metoda *divide et impera* (nici această metodă nu determină soluția optimă, dar este mai bună decât cea bazată pe întrebarea pusă de n ori).

La primul pas vom afișa o submulțime care are cardinalul egal cu jumătate din cardinalul mulțimii initiale, la pasul următor vom afișa o submulțime de bile inclusă în cea afișată la pasul anterior (având cardinalul tot jumătate din cel al mulțimii de la pasul precedent). În funcție de răspunsurile primite, vom afișa numerele de ordine ale bilelor albe.

Exemplu:

Să presupunem că avem 5 bile dintre care 3 sunt albe, având numerele de ordine: 1, 2, 5. La primul pas vom afișa submulțimea:

$\{1, 2, 3\}$? Răspuns: 2

Urmează pașii:

$\{1, 2\}$? Răspuns: 2

Rezultă că bilele 1 și 2 sunt albe.

$\{4\}$? Răspuns: 0

Rezultă că bila 5 este albă.

Astfel am reușit să determinăm 3 bile albe din 3 întrebări. Procedura care generează întrebările este Determină și are ca parametri de intrare poziția de început și cea de sfârșit a ultimei submulțimi afișate.

```

Program DeterminaBileleAlbe;
var n,k,i,h:Byte;
procedure Determina(st,sf,nr:Byte);
var mij,m:Byte;
begin
  if sf-st+1=nr
  then
    begin
      Write('Bilele ');
      for i:=st to sf do Write(i,' ');
      Writeln('sunt albe.')
    end
    else
      if nr<>0
      then
        begin
          mij:=(st+sf) div 2;
          Write('{');
          for i:=st to mij do Write(i,' ');
          Write('}')?'; Readln(m);
          Determina(st,mij,m);
          Determina(mij+1,sf,nr-m)
        end
      end;
end;
Begin
  Write('Introduceti numarul de bile:'); Readln(n);
  Write('Introduceti numarul bilelor albe:'); Readln(k);
  Write('{');
  for i:=1 to (n+1) div 2 do Write(i,' ');
  Write('}')?'; Readln(h);
  Determina(1,(n+1) div 2,h);
  Determina((n+1) div 2+1,n,k-h)
End.

```

Intrare

n=7 k=4
 1,2,4,6

{ 1 2 3 } ? 2
 { 1 2 } ? 2
 Bilele 1 2 sunt albe.
 { 4 5 } ? 1
 { 4 } ? 1
 Bilele 4 sunt albe.
 { 6 } ? 1
 Bilele 6 sunt albe.

Observație

Soluția nu este optimă.

P6. Problema submarinului

O hartă a unei mări este reprezentată sub forma unei matrice de dimensiuni $n \times m$. Pe această mare se află vapoare reprezentate pe hartă prin valoarea 1, atribuite pozițiilor unde se află un vapor.

Aceste vapoare trebuie distruse de submarine inamice. Un submarin poate distruge un vapor dacă se află pe aceeași linie sau coloană cu vaporul (dar nu pe aceeași pozitie) și dacă între vapor și respectivul submarin nu se mai află un alt vapor.

Dându-se poziția vapoarelor, se cere numărul minim de submarine (precum și poziția acestora) necesare pentru a distruge aceste vapoare.

Restricții

Datele se citesc dintr-un fișier text, având următoarea structură:

- pe prima linie este scrisă perechea n, m ;
- pe următoarele n linii este dată harta mării sub forma unei matrice în care elementele au valoarea 1 dacă pe poziția actuală se află un vapor, 0 în caz contrar ([1]).

Soluție

Următorii pași vor fi repetați până în momentul în care toate vapoarele pot fi distruse de un submarin:

- determinăm poziția de amplasare a unui submarin astfel încât numărul de vapoare pe care le poate distruge (și care nu pot fi distruse de un alt submarin) să fie maxim;
- marcăm toate vapoarele care pot fi distruse de submarinul amplasat pe poziția determinată anterior; nu le eliminăm, deoarece la pași următori ar putea fi încălcată una dintre restricțiile din enunț, și anume un submarin ar putea distruge un vapor, chiar dacă între ei s-ar afla un alt vapor.

Numărul minim de submarine necesare îl vom reține în variabila nrs .

```
Program ProblemaSubmarinului;
const MaxN=6;
var a:array[1..MaxN,1..MaxN] of 0..2;
    pozitii:array[1..MaxN*MaxN,1..2] of 1..MaxN;
    n,m,i,j,k,L,nrs,max,nrv:Byte;
procedure citeste;
    var f:Text;
begin
    Assign(f,'SUBMARIN.IN'); Reset(f); Read(f,n,m);
    for i:=1 to n do
        begin for j:=1 to m do Read(f,a[i,j]); Readln(f) end;
    Close(f);
end;
Begin
    citeste; nrs:=0;
```

```
repeat
    max:=0; nrs:=nrs+1;
    for i:=1 to n do
        for j:=1 to m do
            begin
                k:=i-1; L:=j; nrv:=0;
                while (k>0) and not (a[k,L] in [1..2]) do Dec(k);
                if (k>0) and (a[k,L]=1) then Inc(nrv);
                k:=i+1;
                while (k<=n) and not (a[k,L] in [1..2]) do Inc(k);
                if (k<=n) and (a[k,L]=1) then Inc(nrv);
                k:=i; L:=j-1;
                while (L>0) and not (a[k,L] in [1..2]) do dec(L);
                if (L>0) and (a[k,L]=1) then Inc(nrv);
                L:=j+1;
                while (L<=m) and not (a[k,L] in [1..2]) do Inc(L);
                if (L<=m) and (a[k,L]=1) then Inc(nrv);
                if max<nrv then
                    begin pozitii[nrs,1]:=i; pozitii[nrs,2]:=j; max:=nrv end
            end;
    k:=pozitii[nrs,1]-1; L:=pozitii[nrs,2];
    while (k>0) and not (a[k,L] in [1..2]) do Dec(k);
    if (k>0) and (a[k,L]=1) then a[k,L]:=2;
    k:=pozitii[nrs,1]+1;
    while (k<=n) and not (a[k,L] in [1..2]) do Inc(k);
    if (k<=n) and (a[k,L]=1) then a[k,L]:=2;
    k:=pozitii[nrs,1]; L:=pozitii[nrs,2]-1;
    while (L>0) and not (a[k,L] in [1..2]) do Dec(L);
    if (L>0) and (a[k,L]=1) then a[k,L]:=2;
    L:=pozitii[nrs,2]+1;
    while (L<=m) and not (a[k,L] in [1..2]) do Inc(L);
    if (L<=m) and (a[k,L]=1) then a[k,L]:=2;
until max=0;
Writeln('Numarul minim de submarine necesare:',nrs-1);
Writeln('Pozițiile pe care vor fi amplasate submarine:');
for i:=1 to nrs-1 do
    Write('(' ,pozitii[i,1],',',pozitii[i,2],') ')
```

End.

Intrare

```
6 6
0 1 0 0 1 0
1 0 1 0 1 0
0 1 0 0 1 0
0 0 0 1 0 1
1 0 1 0 1 0
0 1 0 0 0 0
```

Ieșire

```
Numarul minim de submarine necesare este:5
Pozițiile pe care vor fi amplasate submarine:
(2,2) (4,5) (5,2) (1,2) (2,5)
```

VII. ALGORITMI GENETICI

Introducere

Algoritmii genetici sunt o parte a calculului evolutiv care la rândul său este o parte în rapida dezvoltare a inteligenței artificiale. Algoritmii genetici sunt inspirați din teoriile lui *Ch. Darwin* despre evoluție. Simplu spus, soluția unei probleme rezolvate de către un algoritm genetic evoluează.

Ideea calculului evolutiv a fost introdusă în 1960 de către *I. Rechenberg* în lucrarea sa *Evolution strategies*. Ideea lui a fost dezvoltată de către alți cercetători. Algoritmii genetici au fost inventați de către *John Holland*. Acești și-a expus ideile în carte *Adaptation in Natural and Artificial Systems*, publicată în 1975.

În 1992 *John Koza* a folosit algoritmii genetici pentru a dezvolta programe destinate anumitor scopuri precise. Metoda folosită de el a primit denumirea de *programare genetică*.

Termeni biologici

Toate organismele vii sunt alcătuite din celule. În nucleul fiecărei celule există aceeași mulțime de cromozomi. Cromozomii sunt șiruri de ADN și servesc ca model pentru întregul organism. Un cromozom este alcătuit din gene, care sunt fragmente de ADN. Fiecare genă codifică o caracteristică (spre exemplu culoarea ochilor). Genele sunt situate pe un cromozom. Această poziție se numește *locus*.

Întregul material genetic (toți cromozomii) poartă numele de *genom*.

În timpul reproducерii în primul rând apare încrucișarea (engl. *crossover*). Astă înseamnă că gene din doi părinți (doi cromozomi) formează într-un fel un nou cromozom. Acesta la rândul lui poate fi schimbat. Schimbarea sau mutația (engl. *mutation*) înseamnă că elemente din cromozom au fost modificate. Mutarea apare de obicei din cauza erorii copierii de gene de la părinți.

Algoritmul general

Soluția pentru o problemă rezolvată prin algorimi genetici evoluează. Algoritmul începe cu o mulțime de soluții (fiecare soluție este reprezentată de către un cromozom) numită populație. Soluțiile dintr-o populație sunt luate și folosite pentru a forma o nouă populație. Acăt lucru este motivat de faptul că noua populație ar putea fi mai

bună decât cea veche. Combinarea soluțiilor se face prin *crossover* și *mutation*. Prin crossover doi cromozomi sunt desfăcuți și combinați pentru a forma alți doi noi cromozomi numiți și *urmași* (engl. *offspring*). Fiecare cromozom are atașată o valoare numită *speranță de viață* (engl. *fitness*). Cu cât fitness-ul unui cromozom este mai mare, cu atât șansa ca acesta să supraviețuiască în populația următoare este mai mare. Totodată, la încrucișare, cea mai mare șansă de a fi aleși ca părinți, sunt cromozomii cu fitness-ul cel mai mare.

Algoritmul de mai sus este repetat până când o condiție este satisfăcută. Condiția poate fi executarea de un număr de ori a ciclului, îmbunătățirea soluției cu un anumit grad etc.

Cele spuse până acum se concretizează în:

1. Generarea aleatoare a unei populații de n cromozomi (care reprezintă soluții posibile pentru problemă);
2. Calcularea fitness-ului fiecărui cromozom;
3. Crearea unei noi populații pe baza celei vechi. Următorii pași sunt repetați până când noua populație este completă:
 - 3.1: [Selecție] Selectează doi părinți cromozomi (cu cât fitness-ul lor este mai mare, cu atât trebuie să fie mai mare și șansa de a fi selectați);
 - 3.2: [Crossover] Încrucișează cei doi cromozomi peintru a forma doi cromozomi noi;
 - 3.3: [Mutare] Modifică noii cromozomi formați;
 - 3.4: Pune cei doi cromozomi formați în noua populație;
4. Noua populație formată o va înlocui pe cea veche;
5. Dacă se îndeplinește *condiția de final*, atunci se afișează cel mai bun cromozom din populație și *stop*;
6. Se revine la pasul 2.

Precum se poate observa, algoritmul de mai sus este foarte general. Pentru fiecare problemă particulară în parte trebuie să se răspundă la următoarele întrebări:

1. Cum codificăm cromozomii, și cum este reprezentată o soluție într-un cromozom?
2. Cum inițializăm cromozomii? Care este populația inițială? Cât de mare este ea?
3. Cine este fitness-ul unui cromozom?
4. Cum selecționăm părinții pentru încrucișare?
5. Cum se face încrucișarea?
6. Cum se face mutația?
7. Dacă noua populație este formată doar prin crossover, atunci este posibil să se piardă soluțiile cele mai bune din actuala populație. Cum evităm acest lucru?
8. Cât de des apare mutația? Dacă apare prea des, ar însemna că facem o căutare aleatoare în spațiul soluțiilor. Dacă nu apare nici o dată, atunci s-ar putea să cădem într-un extrem local.
9. Când se termină algoritmul?

La toate aceste întrebări și la multe altele vom răspunde în paragrafele ce urmează.

1) Codificarea cromozomilor

Există multe variante de codificare, totul depinzând de problema pe care dorim să o rezolvăm. Trebuie avută mare grijă în această etapă, deoarece o codificare bună poate duce la o implementare ușoară a operațiilor de **încrucișare și mutație**. Exemple:

a) Codificarea binară

Fiecare cromozom este reprezentat ca un sir de 1 și 0:

Cromozomul A	10110010110010101011100101
Cromozomul B	111111100000110000011111

Problema clasică care folosește acest tip de codare este problema rucsacului: se dau N obiecte și se cere să se umple cât mai mult un rucsac de capacitate C . Aici fiecare obiect este codificat cu 1 dacă este în sac și cu 0 dacă nu este în sac.

b) Codificarea sub formă de permutare

Fiecare cromozom este reprezentat sub forma unei permutări:

Cromozomul A	1 5 3 2 6 4 7 9 8
Cromozomul B	8 5 6 7 2 3 1 4 9

Această codificare se folosește în cazul problemei comis voiajorului. O permutare reprezintă ordinea în care comis-voiajorul vizitează orașele.

c) Codificarea sub formă de valori

Fiecare cromozom este un sir de valori:

Cromozomul A	1.2324 5.3243 0.4556 2.3293 2.4545
Cromozomul B	ABDJEIFJDHDIERJFDLDLFLFEGT
Cromozomul C	(back), (back), (right), (forward), (left)

2) Populația inițială

Populația este o mulțime de cromozomi care trebuie să fie suficient de măre pentru a se putea realiza un număr multumitor de combinații între cromozomii compoziți, dar nu prea mare, deoarece acest lucru va încetini algoritmul.

Populația inițială este una aleatoare. Spre exemplu, pentru determinarea unui drum hamiltonian de cost minim se poate începe cu generarea aleatoare a câtorva permutări. Pentru determinarea unei componente intern stabile maximale se poate începe cu generarea aleatoare a câtorva componente intern stabile. Acestea pot să fie formate și doar dintr-un singur nod.

De obicei populația inițială este aleatoare. Dar pentru o mai bună performanță, o parte din cromozomi ar putea fi generați prin metode euristică (cum ar fi de exemplu cele din capitolul despre algoritmi euristică). În acest fel s-ar putea ajunge la soluție mai repede.

3) Fitness-ul unui cromozom

Fitness-ul unui cromozom este speranța lui de viață. Pe noi ne interesează să obținem cromozomi cu un fitness cât mai mare, sau cât mai mic, în orice caz, cu un fitness la o extremitate.

În cazul problemei determinării unei componente intern stabile maximale, fitness-ul unui cromozom este cardinalul multumii de noduri pe care o codifică acel cromozom. Aici ne interesează să obținem un fitness cât mai mare.

În cazul problemei comis-voiajorului fitness-ul este lungimea drumului codificat de respectivul cromozom. Aici ne interesează să obținem un cromozom cu fitness-ul cât mai mic.

În cele ce urmează presupunem, pentru simplificare, că dorim să realizăm cromozomi cu fitness-ul cât mai mare.

4) Selectarea cromozomilor părinți

Cromozomii sunt selectați din populație pentru a fi părinți într-o încrucișare (cross-over). Conform teoriei lui Darwin, cei mai buni părinți supraviețuiesc și se înmulțesc creând noi urmași.

Dintre metodele cele mai des întâlnite de selecție amintim:

a) Selecție aleatoare

Doi părinți sunt aleși aleator din populație. Acest lucru se face generând două numere aleatoare între 1 și dimensiunea populației.

b) Selecția cu ajutorul rulelei

Părinții sunt selectați în conformitate cu fitness-ul lor. Cu cât sunt mai buni, cu atât sansa lor de a fi aleși este mai mare. Imagineați-vă o rulată în care sunt dispuși toți cromozomii din populație. Fiecare cromozom îi corespunde un sector al rulelei, direct proporțional, ca mărime, cu fitness-ul cromozomului respectiv. În acest fel cromozomii cu fitness mare au atașate sectoare mai mari, iar cei cu fitness mic au atașate sectoare mai mici. La aruncarea bilei pe rulată există mai multe șanse de alegere pentru cromozomii cu fitness mare.

Simularea roții de rulată se face în felul următor:

- se calculează suma tuturor fitness-urilor cromozomilor. Fie aceasta S .
- se generează un număr aleator între 1 și S . Fie acesta r .
- parcurem populația și calculăm suma fitness-urilor cromozomilor până ajungem la valoarea r . Acel cromozom îl alegem.

c) Selecția aranjată

Selecția anterioară nu este recomandată în cazul în care fitness-urile cromozomilor diferă foarte mult. De exemplu dacă unul din cromozomi are fitness-ul de 100 de ori mai mare decât suma fitness-urilor celorlalți cromozomi este clar care sunt șansele cromozomilor cu fitness mic de a fi aleși. Pentru a evita această situație, se ordonează cromozomii crescător după fitness. Apoi se renumează cu numere întregi din inter-

valul [1, ..., dimensiunea populației]. Cromozomul cel mai mic are numărul 1 etc., iar cel mai mare (cu fitness-ul cel mai mare) are numărul egal cu dimensiunea populației. Aceste numere sunt considerate fitness-uri și pe ele se aplică selecția roții de ruletă. Observăm că cromozomii cu fitness mare au tot cele mai mari șanse de a fi aleși, dar de data aceasta nu mai sunt așa de mari în comparație cu șansele celorlalți.

d) Selecție cu starea consolidată

Aceasta nu este o metodă particulară. Ideea esențială este că cromozomii nou generați vor înlocui, dacă sunt mai buni, o parte din cromozomii vechii populații. În acest fel nu mai este necesar să se lucreze cu mai multe populații deodată, ci doar cu una singură.

e) Elitism

Când creăm o nouă populație prin încrucișare și mutații, există o șansă mare de a pierde cei mai buni cromozomi. De aceea este ar trebui să copiem cei mai buni cromozomi în noua populație, fără să schimbăm nimic la ei. Elitismul va crește rapid performanțele algoritmilor genetici.

5) Încrucișarea (crossover)

Încrucișarea depinde de tipul de codificare a cromozomilor și de problema particulară pe care o rezolvăm. Să luăm câteva exemple:

Codificarea binară

a) Încrucișare într-un singur punct:

Un punct de încrucișare este ales. Din primul părinte este copiată secvența de la început până la punctul de încrucișare, iar din al doilea părinte este copiată secvența de la punctul de încrucișare până la final.

$$11001011 + 11011111 = 11001111$$

b) Încrucișarea în două puncte:

Sunt alese două puncte de încrucișare. Secvența dintre cele două puncte este aleasă dintr-unul din părinți, iar ceea ce a rămas din celălalt părinte:

$$11001011 + 11011111 = 11011111$$

c) Încrucișarea uniformă

Biții sunt copiați aleator din primul și al doilea părinte.

$$11001011 + 11011101 = 11011111$$

d) Încrucișarea aritmetică

Operații aritmetice sunt efectuate pentru a crea noi urmași.

$$11001011 + 11011111 = 11001001 \text{ (AND)}$$

Codificarea sub formă de permutare

Metodele folosite aici sunt asemănătoare cu cele de la codificarea binară. Trebuie avută grijă să se păstreze consistența permutterii, adică să nu apară un număr de două ori, iar altele nici o dată. $(1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9) + (4\ 5\ 3\ 6\ 8\ 9\ 7\ 2\ 1) = (1\ 2\ 3\ 4\ 5\ 6\ 8\ 9\ 7)$

Codificarea sub formă de valoare:

Metodele de la codificarea binară pot fi folosite și aici.

Observații

Probabilitatea de încrucișare indică cât de des trebuie să apară această operație într-o populație. De obicei are valoarea mare, cuprinsă între 60% și 90%.

Metodele de încrucișare prezentate mai sus, sunt destul de generale. De obicei sunt alese metode specifice problemei care se dorește a fi rezolvată și care generează o populație mai bună. Spre exemplu în cazul problemei componentei interne stabile maximale se aleg doi cromozomi care codifică două componente interne stabile. Aceste componente sunt de fapt mulțimi, reprezentate ca un sir de biți. În acest caz se poate folosi, de exemplu, încrucișarea cu un singur punct; sau cu două puncte, dar s-ar putea ca în acest caz componentele nou formate să fie mai mici decât părinții lor. De aceea putem lua unul din părinți și îi vom adăuga noduri din al doilea părinte, rezultând în acest fel un cromozom cu fitness-ul mai mare sau egal cu al unuia dintre părinți. Este repetată aceeași operație și pentru celălalt părinte. Acest tip de încrucișare se numește încrucișare inteligentă.

6) Mutația

Mutația apare în principal pentru a evita căderea soluțiilor problemei într-un optim local. Efectuarea de prea multe ori a acestei operații va transforma algoritmul într-o căutare aleatoare în spațiul soluțiilor posibile. De aceea probabilitatea de mutație trebuie să fie mică, circa 0,5%.

Codificarea binară

a) Inversarea biților

Biții selectați sunt inversați $1 \leftrightarrow 0$.

$$11001001 \Rightarrow 10001001$$

b) Interschimbarea biților

Se aleg doi biți și se schimbă valorile între ei.

$$11001001 \Rightarrow 10001101$$

Codificarea sub formă de permutare

a) Transpoziție

Este efectuată o transpoziție asupra permutării respective. Acest lucru este echivalent cu interschimbarea bițiilor de la codificarea binară.

$$(1\ 2\ 3\ 4\ 5\ 6\ 8\ 9\ 7) \Rightarrow (1\ 8\ 3\ 4\ 5\ 6\ 2\ 9\ 7)$$

Codificarea sub formă de valori

Un număr mic este adăugat sau scăzut din valorile selectate.

$$(1.29\ 5.68\ 2.86\ 4.11\ 5.55) \Rightarrow (1.29\ 5.68\ 2.73\ 4.22\ 5.55)$$

7) Când se termină algoritmul?

Algoritmii genetici se termină, de obicei, după ce s-au creat un număr predefinit de populații noi. În alte cazuri se poate determina dacă soluția s-a îmbunătățit de la o populație la alta. În caz afirmativ se continuă cu crearea de noi populații, iar în caz negativ se afișează cel mai bun cromozom din populația curentă.

Implementare

Algoritmii genetici au o structură foarte generală, care permite folosirea lor pentru majoritatea problemelor. Ceea ce se schimbă sunt următoarele:

- codificarea cromozomilor;
- inițializarea populației;
- încrucișarea;
- mutația.

Comentariile la nivel de cod sursă sunt suficiente pentru a înțelege ce se întâmplă.

```
Program AlgoritmGeneticGeneral;
const MaxPopulatie=100; { Numarul de cromozomi in populatie }
MaxIteratii=100; { Numarul de iteratii ale algoritmului }
MaxElitism=2; { Cati cromozomi sunt copiati prin elitism }
MaxCrossOver=20;
{ De cate ori apare incrusarea intr-o populatie }
MaxMutatie=1;
{ De cate ori apare mutatia intr-o populatie }
type TCromozom=record
    cromozom:Byte;
    { depinde de problema care este rezolvata }
    fitness:Byte
end;
TPopulatie=array[1..MaxPopulatie] of TCromozom;
{ tipul populatie }
var populatieVeche,populatieNoua:TPopulatie;
{ se lucreaza cu doua populatii }
```

```
i,j,N,contor,co,r1,r2,k:Byte;
NrPop:Byte;
{ contorizeaza numarul cromozomilor adaugati la noua populatie }
procedure Citire; { Citeste datele de intrare }
begin
end;

function CalculFitness(G:TCromozom):Byte;
{ Calculeaza fitness-ul unui cromozom }
begin
{ de obicei fitness-ul este calculat in momentul in care se }
{ creaza populatia sau in momentul in care se face crossover }
{ sau mutatie deci aceasta procedura nu este utila, decat }
{ atunci cand calcularea fitness-ului nu se poate face direct }
end;

procedure InitializeazaPopulatie; { initializarea populatiei }
begin
    for i:=1 to MaxPopulatie do
        begin
            { sunt initializati cromozomii din populatie }
        end
end;

procedure Elitism(NrMutati:Byte);
begin
    { Realizeaza copierea celor mai buni }
    { NrMutati cromozomi in noua populatie }
    for i:=1 to NrMutati do
        begin
            Inc(NrPop);
            populatieNoua[NrPop]:=populatieVeche[MaxPopulatie-i+1]
        end
end;

procedure Crossover(g1,g2:TCromozom;var rez1,rez2:TCromozom);
begin
    { incrusisarea parintilor cromozomi g1 si g2 }
    { vor rezulta doi pui cromozomi rez1 si rez2 }
end;

procedure Mutatie(var mutant:TCromozom);
begin
    { se efectueaza o mutatie asupra unui cromozom }
    { ales aleator, se returneaza cromozomul mutant }
end;

procedure MutaRestul(NrMutati:Byte); { Dupa ce s-a efectuat }
{ incrusisarea, mutatia si elitismul mai raman }
{ NrMutati pozitii libere in noua populatie. }
begin
end;
```

```

begin { Acestea se vor umple aleator cu }
  for i:=1 to NrMutati do { cromozomi din vechea populatie }
  begin
    Inc(NrPop);
    populatieNoua[NrPop]:=populatieVeche[1+ Random(MaxPopulatie-MaxElitism)]
  end
end;

procedure OrdeneazaDupaFitness(l,r:Byte);
{ Ordonarea cromozomilor dintr-o populatie }
var i,j,x:Integer; { crescator dupa fitness }
y:TCromozom;
begin
  i:=l; j:=r; x:=populatieVeche[(l+r) div 2].fitness;
repeat
  while populatieVeche[i].fitness<x do i:=i+1;
  while x<populatieVeche[j].fitness do j:=j-1;
  if i<=j
  then
  begin
    y:=populatieVeche[i];
    populatieVeche[i]:=populatieVeche[j];
    populatieVeche[j]:=y;
    Inc(i); Dec(j)
  end;
until i>j;
if l<j then OrdeneazaDupaFitness(l, j);
if i<r then OrdeneazaDupaFitness(i, r)
end;

procedure SelectieParinti(var r1,r2:Byte);
{ Selecteaza doi parinti din populatie }
begin { Se va folosi una din metodele descrise mai sus }
end;

procedure Afiseaza;
begin { se va afisa cel mai bun cromozom din populatie }
end; { programul principal }

Begin
  Citire;
  Randomize;
  InitializeazaPopulatie;
  OrdeneazaDupaFitness(1,MaxPopulatie);
  for contor:=1 to MaxIteratii do
  begin { Algoritmul se va termina dupa MaxIteratii }
    NrPop:=0;
    Elitism(MaxElitism);
  end;
end;

```

```

for co:=1 to MaxCrossover do
begin
  SelectieParinti(r1,r2);
  CrossOver(populatieVeche[r1],populatieVeche[r2],
             populatieNoua[NrPop+1],populatieNoua[NrPop+2]);
  Inc(NrPop,2);
end;
MutaRestul(MaxPopulatie-NrPop);
populatieVeche:=populatieNoua;
for co:=1 to MaxMutation do
  Mutatie(populatieNoua[1+Random(maxPopulatie)]);
  OrdeneazaDupaFitness(1,MaxPopulatie);
end;
Afiseaza;
End.

```

Să vedem cum implementăm *selecția*. Metodele cele mai importante pe care le-am prezentat mai sus sunt:

1. Selectia uniformă

```

procedure SelectieUniforma(var r1,r2:Byte);
begin
  r1:=1+Random(MaxPopulatie); r2:=1+Random(MaxPopulatie)
end;

```

2. Selectie folosind ruleta

```

procedure SelectieRuleta(var r1,r2:Byte);
var S,R:Integer;
begin
  S:=0;
  for i:=1 to MaxPopulatie do
    Inc(S,populatieVeche[i].fitness);
  R:=1+Random(S); { pentru primul parinte }
  r1:=0;
  while R>0 do
  begin
    Inc(r1); R:=R-populatieVeche[r1].fitness
  end;
  R:=1+Random(S); r2:=0; { pentru al doilea parinte }
  while R>0 do
  begin
    Inc(r2); R:=R-populatieVeche[r2].fitness
  end;
end;

```

3. Selectie aranjată

```

procedure SelectieAranjata(var r1,r2:Byte);
var S,R:Integer;
begin
  S:=MaxPopulatie*(1+MaxPopulatie) div 2;
  R:=1+Random(S); r1:=0; {pentru primul parinte}
  while R>0 do begin Inc(r1); R:=R-r1 end;
  R:=1+Random(S); r2:=0; {pentru al doilea parinte}
  while R>0 do begin Inc(r2); R:=R-r2 end;
end;

```

P1. Componentă intern stabilă maximală

Se dă un graf neorientat cu N noduri. Se cere să se determine o componentă intern stabilă maximală, adică o submulțime maximală de noduri cu proprietatea că oricare două noduri din submulțime nu sunt legate printr-un arc.

Soluție

Datele de intrare se citesc sub forma unei matrice, în maniera cunoscută.

Un cromozom va fi definit ca o mulțime de noduri (o componentă intern stabilă). Fitness-ul unui cromozom este egal cu cardinalul mulțimii de noduri pe care o codifică.

Inițializarea unui cromozom din populație se face prin următorul algoritm euristic: se pune în cromozom un nod de start. Populația trebuie să fie suficient de mare pentru ca fiecare nod să fie în câțiva cromozomi. Apoi se tot încercă adăugarea de noi noduri la componentă curentă. După ce s-au introdus în ea toate nodurile posibile se trece la construirea următorului cromozom.

Inițializarea cromozomilor se poate face și doar cu un singur nod. În acest caz însă vom avea nevoie de mai mulți pași pentru a obține soluția, în timp ce algoritmul euristic prezentat mai sus, poate ajunge la soluția dorită chiar după etapa de inițializare.

Pentru a realiza încrucișarea, primul cromozom fiu va fi considerat primul cromozom părinte la care se adaugă noduri din al doilea părinte. Al doilea cromozom pui va fi al doilea părinte la care se adaugă noduri din primul părinte.

Mutarea constă din eliminarea unui nod oarecare dintr-un cromozom ales la întâmpinare.

În continuare prezentăm procedurile care se vor adăuga la algoritmul genetic general de mai sus.

```

type TMultime=set of 1..MaxN;
TCromozom=record
  multime:TMultime;
  fitness:Byte;
end;

```

```

procedure Citire;
begin
  Assign(f,'graf.in'); Reset(f); Readln(f,n);
  for i:=1 to N do
    begin
      for j:=1 to N do Readln(f,a[i,j]);
      Readln(f)
    end;
  end;

procedure InitializeazaPopulatie;
var ok:Boolean;
begin
  for i:=1 to MaxPopulatie do
    begin
      populatieVeche[i].multime:=
        populatieVeche[i].multime+[i mod N+1];
      populatieVeche[i].fitness:=1;
      for j:=i mod N+2 to N do
        begin
          ok:=true;
          for k:=1 to N do
            if (k in populatieVeche[i].multime) and (a[k,j]=1)
              then ok:=false;
          if ok
            then
              begin
                populatieVeche[i].multime:=populatieVeche[i].multime+[j];
                Inc(populatieVeche[i].fitness)
              end
        end;
    end;
end;

procedure Crossover(g1,g2:TCromozom; var rez1,rez2:TCromozom);
begin
  rez1:=g1;
  for i:=1 to N do
    if i in g2.multime
      then
        begin
          j:=1;
          while (j<=N) and not((j in rez1.multime) and (a[i,j]=1)) do
            if j=N+1 then rez1.multime:=rez1.multime+[i];
            Inc(rez1.fitness)
          end;
  rez2:=g2;
end;

```

```

for i:=1 to N do
  if i in g1.multime
  then
    begin
      j:=1;
      while (j<=N) and not((j in rez2.multime) and (a[i,j]=1)) do
        if j=N+1 then rez2.multime:=rez2.multime+[i];
        Inc(rez2.fitness)
      end
    end;
procedure Mutatie(mutant:TCromozom);
  var v,r:Byte;
begin
  r:=1+Random(MaxPopulatie);           { un cromozom aleator }
  v:=1+Random(N);                     { un nod aleator       }
  mutant:=populatieVeche[r];
  if v in mutant.multime
  then mutant.multime:=mutant.multime-[v]
end;
procedure Afisare;
begin
  for i:=1 to N do
    if i in populatieVeche[MaxPopulatie].multime
    then Writeln(i,' ')
end;

```

P2. Drum de lungime maximă

Se dă un graf neorientat cu N noduri. Se cere să se determine un drum elementar care conține cât mai multe noduri.

Soluție

Un cromozom codifică un drum sub forma unui șir de noduri. Fitness-ul unui cromozom este egal cu lungimea drumului (numărul de noduri din componenta lui).

Pentru inițializare se folosește următorul algoritm euristic: se pleacă dintr-un vârf oarecare și se adaugă noduri cât timp mai există legătură între ele.

Încrucișarea se face în următorul mod: primul urmaș este primul părinte, căruia îl se adaugă la un capăt un lanț ce există în al doilea părinte. Al doilea urmaș este al doilea părinte la care îl se adaugă un lanț din primul părinte.

În final obținem cel mai lung drum elementar. Acesta ar putea fi chiar hamiltonian.

```

type TDrum=array[1..MaxN] of 0..MaxN;
TCromozom=record
  drum:TDrum;
  fitness:Byte
end;
procedure InitializeazaPopulatie;
var ok:Boolean;
begin
  for i:=1 to MaxPopulatie do
    begin
      populatieVeche[i].drum[1]:=i mod N+1;
      populatieVeche[i].fitness:=1;
      ok:=true;
      k:=1;
      while ok do
        begin
          ok:=false;
          for j:=populatieVeche[i].drum[k]+1 to N do
            if a[populatieVeche[i].drum[k],j]=1
            then
              begin
                Inc(k);
                populatieVeche[i].drum[k]:=j;
                Inc(populatieVeche[i].fitness);
                ok:=true;
                Break
              end
            end;
        end;
    end;
end;
procedure Crossover(g1,g2:TCromozom;var rez1,rez2:TCromozom);
var ok:Boolean;
function InDrum(Nr:Byte; dindr,dr:TCromozom):Boolean;
var c:Byte;
begin
  if (Nr=0) or (Nr>dindr.fitness)
  then begin InDrum:=true; Exit end;
  for c:=1 to dr.fitness do
    if dr.drum[c]=dindr.drum[Nr]
    then begin InDrum:=true; Exit end;
  InDrum:=false
end;
begin
  rez1:=g1; ok:=true;
{ crossover }

```

```

while ok do
begin
  ok:=false;
  for i:=1 to g2.fitness do
    if g2.drum[i]=rez1.drum[rez1.fitness]
    then
      if not InDrum(i-1,g2,rez1)
      then
        begin
          Inc(rez1.fitness); ok:=true;
          rez1.drum[rez1.fitness]:=g2.drum[i-1];
          Exit
        end
      else
        if not InDrum(i+1,g2,rez1)
        then
          begin
            Inc(rez1.fitness); ok:=true;
            rez1.drum[rez1.fitness]:=g2.drum[i+1];
            Exit
          end
        end;
    rez2:=g2; ok:=true;
  while ok do
  begin
    ok:=false;
    for i:=1 to g1.fitness do
      if g1.drum[i]=rez2.drum[rez2.fitness]
      then
        if not InDrum(i-1,g1,rez2)
        then
          begin
            Inc(rez2.fitness); ok:=true;
            rez2.drum[rez2.fitness]:=g1.drum[i-1];
            Exit
          end
        else
          if not InDrum(i+1,g1,rez2)
          then
            begin
              Inc(rez2.fitness); ok:=true;
              rez2.drum[rez2.fitness]:=g1.drum[i+1];
              Exit
            end
          end;
    end;
  end;
end;

```

P3. Rezolvarea ecuației de gradul 2

Fie ecuația $ax^2 + bx + c = 0$. Se cere să i se determine rădăcinile.

Soluție

Puteam bineînțeles aplica formulele cunoscute. Însă programul care urmează demonstrează cum această tehnică poate fi folosită și la ecuații de grad mai mare.

Inițial generăm o mulțime aleatoare de perechi (x_1, x_2) . Pentru ca soluția să fie găsită mai rapid, se presupune că rădăcinile se află în intervalul $-50, \dots, 50$.

Prin mutație se schimbă puțin valoarea uneia dintre rădăcini. Probabilitatea de mutație aici este mare.

Prin încrucișare, noi cromozomi preiau câte o rădăcină de la fiecare din părinți.

```

type TSolutie=record
  x1,x2:Integer
end;
TCromozom=record
  sol:TSolutie;
  fitness:Byte
end;
TPopulatie=array[1..MaxPopulatie] of TCromozom;
var populatieVeche,populatieNoua:TPopulatie;
  populatieIntermediara:TPopulatie;
  f:Text;
  i,j,contor,NrPop,co,k,r1,r2:Byte;
  a,b,c:Integer;
procedure Citire;
begin
  Assign(f,'ec.in'); Reset(f);
  Readln(f,a,b,c);
  Close(f);
end;
procedure InitializeazaPopulatie;
var ok:Boolean;
begin
  for i:=1 to MaxPopulatie do
    with populatieVeche[i] do
      begin
        sol.x1:=Random(101)-50;
        sol.x2:=Random(101)-50;
        fitness:=-Abs(a*sol.x1*sol.x1+b*sol.x1+c)+Abs(a*sol.x2*sol.x2+b*sol.x2+c)
      end;
end;

```

```

begin { Acestea se vor umple aleator cu }
  for i:=1 to NrMutati do { cromozomi din vechea populatie }
    begin
      Inc(NrPop);
      populatieNoua[NrPop]:=populatieVeche[1+
        Random(MaxPopulatie-MaxElitism)];
    end
  end;
procedure OrdeneazaDupaFitness(l,r:Byte);
  { Ordonarea cromozomilor dintr-o populatie }
  var i,j,x:Integer; { crescator dupa fitness }
  y:TCromozom;
begin
  i:=l; j:=r; x:=populatieVeche[(l+r) div 2].fitness;
  repeat
    while populatieVeche[i].fitness < x do i:=i+1;
    while x < populatieVeche[j].fitness do j:=j-1;
    if i <=j
    then
      begin
        y:=populatieVeche[i];
        populatieVeche[i]:=populatieVeche[j];
        populatieVeche[j]:=y;
        Inc(i); Dec(j);
      end;
    until i>j;
    if l < j then OrdeneazaDupaFitness(l, j);
    if i < r then OrdeneazaDupaFitness(i, r)
  end;
procedure SelectieParinti(var r1,r2:Byte);
  { Selecteaza doi parinti din populatie }
begin { Se va folosi una din metodele descrise mai sus }
end;
procedure Afiseaza;
begin { se va afisa cel mai bun cromozom din populatie }
begin { programul principal }
Begin
  Citire;
  Randomize;
  InitializeazaPopulatie;
  OrdeneazaDupaFitness(1,Maxpopulatie);
  for contor:=1 to MaxIteratii do
    begin { Algoritmul se va termina dupa MaxIteratii }
      NrPop:=0;
      Elitism(MaxElitism);

```

```

      for co:=1 to MaxCrossover do
        begin
          SelectieParinti(r1,r2);
          CrossOver(populatieVeche[r1],populatieVeche[r2],
                     populatieNoua[NrPop+1],populatieNoua[NrPop+2]);
          Inc(NrPop,2);
        end;
      MutaRestul(MaxPopulatie-NrPop);
      populatieVeche:=populatieNoua;
      for co:=1 to MaxMutatii do
        Mutatie(populatieNoua[1+Random(maxPopulatie)]);
      OrdeneazaDupaFitness(1,MaxPopulatie);
    end;
    Afiseaza;
  End.

```

Să vedem cum implementăm *selecția*. Metodele cele mai importante pe care le-am prezentat mai sus sunt:

1. Selectia uniformă

```

procedure SelectieUniforma(var r1,r2:Byte);
begin
  r1:=1+Random(MaxPopulatie); r2:=1+Random(MaxPopulatie)
end;

```

2. Selectie folosind ruleta

```

procedure SelectieRuleta(var r1,r2:Byte);
var S,R:Integer;
begin
  S:=0;
  for i:=1 to MaxPopulatie do
    Inc(S,populatieVeche[i].fitness);
  R:=1+Random(S); { pentru primul parinte }
  r1:=0;
  while R>0 do
    begin
      Inc(r1); R:=R-populatieVeche[r1].fitness
    end;
  R:=1+Random(S); r2:=0; { pentru al doilea parinte }
  while R>0 do
    begin
      Inc(r2); R:=R-populatieVeche[r2].fitness
    end;

```



```

procedure Crossover(g1,g2:TCromozom;var rez1,rez2:TCromozom);
begin
  rez1:=g1; rez1.sol.x2:=g2.sol.x2;
  rez2:=g2; rez2.sol.x1:=g1.sol.x1;
  with rez1 do
    fitness:=-Abs(a*sol.x1*sol.x1+b*sol.x1+c)+  

      Abs(a*sol.x2*sol.x2+b*sol.x2+c);

  with rez2 do
    fitness:=-Abs(a*sol.x1*sol.x1+b*sol.x1+c)+  

      Abs(a*sol.x2*sol.x2+b*sol.x2+c)
end;

procedure Mutatie(var mutant:TCromozom);
var i:Byte;
begin
  mutant:=populatieVeche[1+Random(MaxPopulatie)];
  with mutant do
    begin
      i:=Random(2); { "mutam" pe x1: }
      if i=0 { scadem }
      then
        begin
          i:=Random(2);
          if i=0 then sol.x1:=sol.x1-Random(3) { adunam }
          else sol.x1:=sol.x1+Random(3)
        end
      else
        begin
          i:=Random(2);
          if i=0 then sol.x2:=sol.x2-Random(3)
          else sol.x2:=sol.x2+Random(3)
        end;
      fitness:=-Abs(a*sol.x1*sol.x1+b*sol.x1+c)+  

        Abs(a*sol.x2*sol.x2+b*sol.x2+c);
    end;
end;

procedure Afiseaza;
begin
  with populatieVeche[maxPopulatie].sol do
    Writeln(x1,x2)
end;

```

Bibliografie

- [1]. Oltean Mihai
Culegere de probleme cu rezolvări în Pascal
Editura Libris, Cluj-Napoca, 1997
- [2]. *Culegere de probleme*
(Algoritmică, baze de date, mediu Windows, Microsoft Word și DOS)
Editura Libris, Cluj-Napoca, 1998
- [3]. Bellman R., Dreyfus
Programarea dinamică în aplicații
Editura Tehnică, București, 1967
- [4]. Knuth D. E.
The art of computer programming. Vol 1: Fundamental Algorithms
Addison Wesley, Reading, Mass, 1968
- [5]. Kovács Sándor
Turbo Pascal - ghid de utilizare
MicroInformatica, Cluj-Napoca, 1992
- [6]. Păun Gheorghe
Jocuri logice, competitive
Editura Sport – Turism, București, 1990
- [7]. Oltean Mihai
Probleme rezolvate în Pascal
Editura Albastră, Cluj-Napoca, 1995
- [8]. Rădulescu Valentin
Cutezanță minții
Editura Militară, București, 1988.
- [9]. Tomescu Ioan
Probleme de combinatorică și teoria grafurilor
Editura Didactică și Pedagogică, 1981
- [10]. Tomescu Ioan
Introducere în combinatorică
Editura Tehnică, București, 1972
- [11]. Probleme de matematică traduse din revista sovietică *Kvant*,
(Selectarea și traducerea: Horea Banea)
Editura Didactică și Pedagogică, 1983
- [12]. Cerchez Mihu
Metode numerice în algebra liniară
Editura Tehnică, București, 1977
- [13]. Titus Popescu
Matematica de vacanță
Editura Sport – Turism, București, 1986
- [14]. Johnson, G.
Computers and Intractability; A Guide to the Theory of NP-Completeness
W. H. Freeman & Co., 1979
- [15]. Teodorescu Nicolae și colectiv
Probleme din Gazeta Matematică
Editura Tehnică, București, 1984
- [16]. Berge, C.
Theorie des graphes et ses applications
Dunod, Paris, 1967
- [17]. Rancea Doina
Limbajul Pascal. Algoritmi fundamentali
Editura Computer Libris Agora, Cluj, 1999
- [18]. Andonie Răzvan, Gârbacea Ilie
Algoritmi fundamentali, O perspectivă C++
Editura Libris, Cluj-Napoca, 1995
- [19]. Bellman R.
Dynamic Programming
Princeton University Press, Princeton, New Jersey, 1957.
- [20]. Oltean Mihai
Programare automată,
Lucrare de diplomă (îndrumător prof. univ. Miltion Frențiu, Universitatea Babeș-Bolyai, Cluj-Napoca, 1999)
- [21]. Colecția *Gazetei de Informatică*
- [22]. Colecția *Gazetei de Matematică*
- [23]. Colecția *Journal of the ACM*
- [24]. Colecția *Acta Informatica*

CUPRINS

I. Metoda programării dinamice	5
II. Metoda backtracking	95
III. Metoda branch and bound	116
IV. Diverse	148
V. Expresii aritmetice	249
VI. Algoritmi euristică	258
VII. Algoritmi genetici	270
Bibliografie	287