Domino

În jocul Domino se utilizează 28 de piese distincte plate, dreptunghiulare. Fiecare piesă se identifică prin două numere, simbolizate prin puncte (*figura 1*). Însemnarea acestor numere pe fiecare placă se efectuează prin divizarea feţei respective în două părţi egale, fiecare număr fiind din mulţimea {0, 1, 2, 3, 4, 5, 6}.

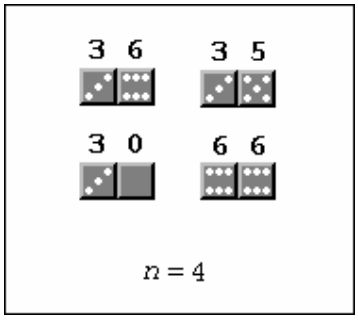

Fig. 1

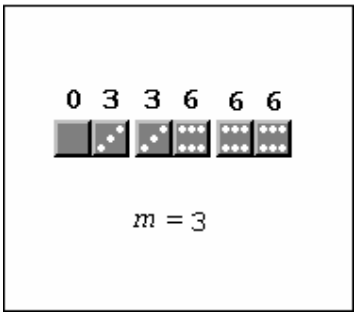

Fig. 2

Utilizînd *n* piese distincte de domino în calitate de vagoane imaginare, pe masă de joc poate fi construit un "tren" cu proprietatea ca cele două numere de pe părţile vecine ale oricăror două piese adiacente sînt egale între ele (*figura 2*). Lungimea trenului se defineşte prin numărul de piese *m* din componenţa lui. Evident, $m \le n$.

Elaboraţi un program care calculează numărul de plăci într-un "tren" de lungime maximă, ce poate fi construit din *n* plăci distincte de domino.

Date de intrare.

Fişierul text `DOMINO.IN` conţine pe prima linie numărul de piese *n*. Următoarele *n* linii ale fişierului de intrare conţin cîte două numere întregi separate prin spaţiu, fiecare linie reprezentînd o piesă de domino.

Date de ieşire.

Fişierul text `DOMINO.OUT` va conţine pe prima linie numărul de piese *m* într-un tren de lungime maximă. Următoarele *m* linii ale fişierului de ieşire vor conţine cîte două numere întregi separate prin spaţiu, fiecare linie reprezentînd o piesă de domino. Piesele vor fi listate în ordinea apariţiei lor în tren, astfel încît cele două numere de pe părţile vecine ale oricăror două piese adiacente sînt egale între ele.

Exemplu.

`DOMINO.IN`

```
4
3 6
3 5
3 0
6 6
```

`DOMINO.OUT`

```
3
0 3
3 6
6 6
```

Restricții. $1 \le n \le 10$. Timpul de execuție nu va depăşi 5 secunde. Fişierul sursă va avea denumirea `DOMINO.PAS`, `DOMINO.C` sau `DOMINO.CPP`.

Rezolvare

Pentru a calcula numărul de piese din componența unui tren de lungime maximă, vom examina consecutiv toate trenurile posibile ce pot fi construite din piesele date. În acest scop vom simula pe calculator acțiunile jucătorului în domino. Introducem în studiu două mulțimi ordonate:

C_0 – mulțimea pieselor disponibile;

X – trenul în curs de construcție.

Se observă că mulțimea X poate fi construită pas cu pas, alegînd din mulțimea pieselor disponibile cîte una care satisface condițiile problemei:

1) Inițial includem în mulțimea X prima piesă din mulțimea C_0: $X := (x_1)$. Evident, această piesă trebuie exclusă din mulțimea pieselor disponibile: $C_1 := C_0 \setminus \{x_1\}$.

2) Alegem din mulțimea pieselor rămase C_1 prima piesă care satisface condițiile problemei şi stabilim: $X := (x_1, x_2)$; $C_2 := C_1 \setminus \{x_2\}$.

3) Presupunem că acest proces a ajuns la pasul k, în mulțimea X fiind deja incluse k-1 piese:

$$X = (x_1, x_2, ..., x_{k-1}).$$

În continuare sînt posibile următoarele cazuri:

A) În mulțimea pieselor rămase C_{k-1} mai există piese care satisfac condițiile problemei. În astfel de cazuri stabilim

$$X := (x_1, x_2, ..., x_{k-1}, x_k), \qquad C_k := C_{k-1} \setminus \{x_k\}$$

şi continuăm calculele.

B) În mulțimea C_{k-1} nu mai există piese care satisfac condițiile problemei. Evident, am obținut un tren care nu mai putem ataşa alte vagoane. Memorăm lungimea acestui tren şi revenim la pasul precedent, unde încercăm să alegem din mulțimea C_{k-2} altă piese x_{k-1}. Dacă în mulțimea C_{k-2} astfel de piese nu mai există, revenim la mulțimea C_{k-3} ş.a.m.d.

Evident, algoritmul care simulează acțiunile jucătorului poate fi realizat prin **metoda reluării**, utilizînd în acest scop recursia.

În programul ce urmează, fiecare piesă de domino este reprezentată cu ajutorul tipului de date

```
Piesa = record
          a, b : 0..6;
        end;
```

iar mulțimile C şi X – cu ajutorul structurilor

```
MultimePiese = record
                 k : integer;
                 p : array[1..28] of Piesa;
               end;
Tren=MultimePiese;
```

Metoda reluării este realizată în procedura cu acelaşi nume, iar trenul de lungime maximă este memorat în variabila `Xmax`.

```
Program Domino;
 { Clasele 10-12 }
type Piesa=record
            a, b : 0..6;
          end;
     MultimePiese=record
                    k : integer;
                    p : array[1..28] of Piesa;
                  end;
     Tren=MultimePiese;
var n : integer;
    C : MultimePiese;
    X, Xmax : Tren;

procedure Citeste(var C : MultimePiese; var n : integer);
 { Citirea datelor din fisierul de intrare }
var i : integer;
    Intrare : text;
begin
   assign(Intrare, 'DOMINO.IN');
   reset(Intrare);
   readln(Intrare, n);
   C.k:=n;
   for i:=1 to n do
      readln(Intrare, C.p[i].a, C.p[i].b);
   close(Intrare);
end; { Citeste }

procedure Scrie(X : Tren);
 { Scrierea datelor in fisierul de iesire }
var i : integer;
    Iesire : text;
begin
   assign(Iesire, 'DOMINO.OUT');
   rewrite(Iesire);
   writeln(Iesire, X.k);
   for i:=1 to X.k do
     writeln(Iesire, X.p[i].a, ' ', X.p[i].b);
   close(Iesire);
end; { Scrie }

procedure AdaugaVagonul(var X : Tren; Q : Piesa);
 { Adauga vagonul Q la trenul X }
begin
   X.k:=X.k+1;
   X.p[X.k]:=Q;
```

```
end;

procedure ScoateVagonul(var X : Tren);
 { Scoate din trenul X ultimul vagon }
begin
  X.k:=X.k-1;
end; { ScoateVagonul}


procedure ExcludePiesa(var Z : MultimePiese; i : integer);
 { Exclude din multimea Z piesa i }
var j : integer;
begin
   for j:=i to Z.k-1 do
     Z.p[j]:=Z.p[j+1];
   Z.k:=Z.k-1;
end; { ExcludePiesa }

function Corespunde(X : MultimePiese; var Q : Piesa) : boolean;
 { Returneaza TRUE daca vagonul Q poate fi atasat la trenul X }
 { In caz de necesitate, roteste piesa Q }
label 1;
var i : integer;
begin
  if X.k=0 then
    begin
      Corespunde:=true;
      goto 1;
    end; { if }
  Corespunde:=false;
  { pozitia initiala a placii Q }
  if X.p[X.k].b=Q.a then
     begin
        Corespunde:=true;
        goto 1;
     end;
  if X.p[X.k].b=Q.b then
     { rotim placa Q }
     begin
        Corespunde:=true;
        i:=Q.a;
        Q.a:=Q.b;
        Q.b:=i;
     end;
1: end; { Corespunde }

procedure Reluare(C : MultimePiese; var X : Tren);
var C1 : MultimePiese;
    Q : Piesa;
```

```
    i : integer;
begin
  for i:=1 to C.k do
    begin
      Q:=C.p[i];
      if Corespunde(X, Q) then
         begin
           AdaugaVagonul(X, Q);
           if X.k>Xmax.k then Xmax:=X;
           C1:=C;
           ExcludePiesa(C1, i);
           Reluare(C1, X);
           ScoateVagonul(X);
         end; { if }
    end; { for }
end; { Reluare }

procedure TrenMaxim(C : MultimePiese; X :Tren);
 { Cauta trenul de lungime maxima }
var i, j : integer;
    Xmax : Tren;
begin
  Xmax.k:=0;
  X.k:=0;
  Reluare(C, X);
  { rotim piesele cu care poate incepe trenul }
  for i:=1 to C.k do
   begin
     j:=C.p[i].b; C.p[i].b:=C.p[i].a; C.p[i].a:=j;
   end;
   X.k:=0;
   Reluare(C, X);
end; { Tren }

begin
   Citeste(C, n);
   TrenMaxim(C, X);
   Scrie(Xmax);
end.
```

Complexitatea temporală a programului `Domino` poate fi estimată luînd în considerație faptul că numărul permutărilor posibile ale unei mulțimi C cu n elemente se determină ca $P_n = n!$. Conform restricțiilor problemei, $n = 10$, deci $P! \approx 4 \cdot 10^6$, mărime care este de acelaşi ordin ca şi capacitatea de prelucrare a calculatoarelor personale. Prin urmare, timpul de execuție va fi de ordinul unei scunde. Întrucît în metoda reluării nu se examinează în mod obligatoriu toate permutările posibile, timpul de execuție al programului `Domino` va fi cu mult mai mic.