Central European Olympiad in Informatics

28 July – 4 August 2005 Sárospatak, Hungary
http://ceoi.inf.elte.hu

Depot Rearrangement

Let n be the number of shops and m be the number of products.
Consider the directed multi-graph $G = (V, E)$, where

$$V = \{p_1, \ldots, p_n\} \cup \{q_1, \ldots, q_m\}$$

For each i and for each j the edge set E of G contains (p_i, q_j) k-times iff the number of containers labeled by j in the container positions $[m*(i-1)+1, m*i]$ is $k+1$ and $k > 0$.
A pair (q_j, p_i) is in E iff there is no container labeled by j in the container positions $[m*(i-1)+1, m*i]$. There is no other edge in G.
It is clear that for each node the number of incoming edges is equal to the number of outgoing edges. Consequently, each component of G has an Eulerian circuit. The number of moves is at least $\sum_{j=1}^{m} Outdegree(q_j)$. Moreover, for every component of the graph at least one container must be moved to the free position. Therefore the minimal number of moves is at least

$$c + \sum_{j=1}^{m} Outdegree(q_j)$$

where c is the number of components of G. The following algorithm uses this number of moves by performing the moves according to the Eulerian circuits.

Complexity:
Time: $O(|E| + n*m)$
Memory: $O(|E| + n*m)$
Implementation

```
program Depot;
Const
  MaxN=1000;       {max # shops}
  MaxM=1000;       {max # products}
  MaxNM=MaxN*MaxM;{max input length}
Type
  List=^Cell;
  Cell=record elem:integer; link:List end;

Var
  n,               {# shops}
  m:longint;       {# products}
  Pl:array[1..MaxNM+1] of longint;
  Poz:array[1..MaxN,1..MaxM] of longint;
  G,G0:array[boolean, 1..MaxN] of List;
  fromp,top,freep,i:longint;
  Nmove:longint;
  outFile:Text;

procedure ReadIn;
Var
  inFile:Text;
```

```
  x,i,j:longint;
  node:List;
begin
  assign(inFile, 'depot.in'); reset(inFile);
  readln(inFile, n, m);
  for i:=1 to MaxN do begin
    G[true,i]:=nil;
    G[false,i]:=nil;
  end;
  for i:=1 to n*m do Pl[i]:=0;
  for i:=1 to n do begin
    for j:=1 to m do
      Poz[i,j]:=0;
  end;
  for i:=1 to n*m do begin
    read(inFile, x);
    j:=(i-1) div m+1;
    if Poz[j,x]=0 then
      Poz[j,x]:=i
    else begin{add x to list of j}
      new(node);
      Pl[i]:=Poz[j,x];
      Poz[j,x]:=i;
      node^.elem:=x;
      node^.link:=G[true,j];
      G[true,j]:=node;
    end;
  end{for i};

  for i:=1 to n do
    for j:=1 to m do
      if Poz[i,j]=0 then begin{add i to list of j}
        inc(Nmove);
        new(node);
        node^.elem:=i;
        node^.link:=G[false,j];
        G[false,j]:=node;
      end;
  close(inFile);
end{ReadIn};

function FromPoz(v,t:longint):longint;
  Var p:longint;
begin
  p:=Poz[v,t];
  Poz[v,t]:=Pl[p];
  FromPoz:=p;
```

```
end{FromPoz};

Procedure DFS(t:boolean; pre,p:integer; domove:boolean);
  Var q:integer;
begin{DFS}
  While G[t,p]<>Nil Do Begin{visiting all edge p->q }
    q:=G[t,p]^.elem;
    G[t,p]:=G[t,p]^.link;
    DFS(not t,p,q, domove);
  End{while};
  if domove and not t then begin
     fromp:=FromPoz(pre,p);
     writeln(outFile, fromp,' ',top);
     top:=fromp;
  end;
end{DFS};

begin{program}
  Nmove:=0;
  ReadIn;
  assign(outFile, 'depot.out'); rewrite(outFile);
  G0:=G;
  for i:=1 to n do {increase NMove by the number of components}
    if G[true,i]<>nil then begin
      inc(NMove);
      DFS(true,G[true,i]^.elem,i, false);
    end;
  writeln(outFile, NMove);

  G:=G0;
  freep:=n*m+1;
  for i:=1 to n do
    if G[true,i]<>nil then begin
      top:=freep;
      DFS(true,G[true,i]^.elem,i, true);
      writeln(outFile, freep,' ',fromp);
    end;
  close(outfile);
end.
```