```
Program Tesina4;      { algoritmo lift-to-front }

uses crt;

const dim = 'z';

type matr  = array ['a'..dim,'a'..dim] of integer;
     vet   = array ['a'..dim] of integer;
     link  = ^nodo;
     nodo  = record inf:char; cap:integer; prec,succ:link; end;
     lista = array ['a'..dim] of link;

var  u,v,vm,sor,poz : char;
     m,n,alt,fmax   : integer;                          { n=|V|, m=|E|, alt='old-height' }
     e,h            : vet;                              { e=flusso in eccesso, h=altezza }
     cap,f          : matr;                             { cap=matrice di adiac, f=matrice dei flussi netti }
     cur,vic,head   : lista;                            { lista dei vicini: head='head[N]', cur='current' }
     p,L            : link;                             { L=lista concatenata dei vertici interni }


  Function min(x,y:integer):integer;
    begin
      if x<y then min:=x else min:=y;
    end;

  Procedure creamatr(var g:matr; m:integer; vm:char);   { crea una matrice di adiacenza, in base alle capacità }
    var i,j:char;  k:integer;
    begin
      for i:='a' to vm do
      for j:='a' to vm do g[i,j]:=0;
      for k:=1 to m do
        begin
          writeln('  > ',k,'^ spigolo.  ');
          write  ('      - 1^ nodo : ');  readln(i);
          write  ('      - 2^ nodo : ');  readln(j);
          write  ('        -> cap. : ');  readln(g[i,j]);
          writeln;
        end;
    end;

  Procedure TrovaEstremi(var s,t:char; g:matr);         { determina il vertice sorgente e il vertice pozzo,  }
    var a,b:boolean; i,j:char;                          { memorizzandoli in variabili apposite }
    begin
      for i:='a' to vm do
        begin
          a:=true; b:=true;
          for j:='a' to vm do
            begin
              if g[i,j]<>0 then b:=false;
              if g[j,i]<>0 then a:=false;
            end;
          if a then s:=i;
          if b then t:=i;
        end;
    end;
```

```
Procedure CreaTesta(g:matr; var n:lista);                          { crea la lista dei vicini "head" }
  var u,v:char; p,top:link;
  begin
    for u:='a' to vm do
      begin
        n[u]:=nil;
        for v:='a' to vm do if (g[u,v]>0) or (g[v,u]>0) then
          begin
            new(p); p^.inf:=v;
            if (n[u]=nil) then n[u]:=p else top^.succ:=p;
            top:=p;
          end;
        top^.succ:=nil;
      end;
  end;

Procedure InizPreflusso(c:matr; s:char);                           { inizializza il preflusso, le altezze, e il flusso in eccesso }
  var u,v:char;
  begin
    for u:='a' to vm do  begin h[u]:=0; e[u]:=0; end;
    for u:='a' to vm do for v:='a' to vm do  begin f[u,v]:=0; f[v,u]:=0; end;
    h[s]:=n;
    for u:='a' to vm do if c[s,u]>0 then
      begin f[s,u]:=c[s,u]; f[u,s]:=-c[s,u]; e[u]:=c[s,u]; end;
  end;

Procedure CreaCatena(var L:link);                                  { crea una lista contenente I vertici interni della rete }
  var u:char; p,top:link;
  begin
    l:=nil;
    for u:='a' to vm do if (u<>sor) and (u<>poz) then
      begin
        new(p); p^.inf:=u;
        if (l=nil) then l:=p else begin top^.succ:=p; p^.prec:=top; end;
        top:=p;
      end;
    l^.prec:=nil; top^.succ:=nil;
  end;

Procedure Push(u,v:char; cf:integer);                              { operzione base di invio }
  var d:integer;
  begin
    d:=min(e[u],cf);
    f[u,v]:=f[u,v]+d; f[v,u]:=-f[u,v];
    e[u]:=e[u]-d;     e[v]:=e[v]+d;
  end;

Procedure Lift(u:char);                                            { operazione base di innalzamento }
  var hmin:integer; v:char;
  begin
    hmin:=999;
    for v:='a' to vm do if (cap[u,v]>f[u,v]) and (h[v]<hmin) then hmin:=h[v];
    h[u]:=1+hmin;
  end;

Procedure Scarica(u:char);                                         { procedura fondamentale che scarica tutto il flusso in eccesso di un vertice }
```

```
    var v:char; cf:integer; q:link;
    begin
      while (e[u]>0) do
        begin
          q:=cur[u];
          if (q=nil) then begin Lift(u); cur[u]:=head[u]; end else
            begin
              v:=q^.inf; cf:=cap[u,v]-f[u,v];
              if (cf>0) and (h[u]=h[v]+1) then Push(u,v,cf) else cur[u]:=cur[u]^.succ;
            end;
        end;
    end;

  Procedure Intesta(p:link);                                    { procedura che porta il puntatore p in testa alla lista L }
    begin
      p^.prec^.succ:=p^.succ;  p^.succ^.prec:=p^.prec;
      p^.succ:=L;              p^.prec:=nil;
      L^.prec:=p;              L:=p;
    end;

Begin                                                           { inizio del programma principale }
  Clrscr;  writeln;
  writeln('      << TESINA  n^3 >>');  writeln;
  writeln('Programma : Algoritmo lift-to-front per determinare il flusso massimo');  writeln;
  writeln(' - dato un grafo orientato, con spigoli "pesati", avente un`unica');
  writeln('   sorgente s e un unico pozzo t, si vuole calcolare il massimo ');
  writeln('   valore che può assumere un flusso da s a t -');  writeln;
  writeln('Inserimento del grafo.');
  write  (' * inserisci il numero dei vertici   : ');   readln(n);  vm:=chr(n+ord('a')-1);
  write  (' * inserisci il numero degli spigoli : ');   readln(m);
  writeln(' * inserisci gli spigoli e le rispettive capacità :');
  writeln('     (chiamare i vertici con nomi da "a" ad "',vm,'")');
  Creamatr(cap,m,vm);
  TrovaEstremi(sor,poz,cap);
  CreaTesta(cap,head);
  InizPreflusso(cap,sor);
  Creacatena(L);
  for u:='a' to vm do if (u<>sor) and (u<>poz) then cur[u]:=head[u];
  p:=L;
  while (p<>nil) do
    begin
      u:=p^.inf;
      alt:=h[u];
      scarica(u);
      if (h[u]>alt) and (p<>L) then intesta(p);
      p:=p^.succ;
    end;
  readln;
  fmax:=0;  p:=head[sor];
  while (p<>nil) do begin u:=p^.inf; fmax:=fmax+f[sor,u]; p:=p^.succ; end;
  writeln('  Flusso massimo : ',fmax); readln;
End.
```