

TRẦN ĐỖ HÙNG

BÀI TẬP

TIN HỌC CHỌN LỌC

Tập II



NHÀ XUẤT BẢN GIÁO DỤC VIỆT NAM

Lời nói đầu

Tin học cũng như Toán học là những môn học có nhiều điều kiện rèn luyện tư duy. Ngày nay trong nền công nghiệp hiện đại và sự phát triển của công nghệ thông tin, tư duy của con người thêm nhiều cách thức mới. Con người đã chế tạo máy tính và các thiết bị truyền thông và con người “dạy” máy làm thay mình nhiều công việc quan trọng với hiệu suất cao. Trong quá trình “dạy” máy này, con người lại khám phá ra chính trong bản thân mình còn nhiều cách tư duy mới, chỉ bộc lộ ra trong quá trình “dạy máy”.

Cùng giải một bài toán, Tin học và Toán học có thể có cách giải khác nhau hoàn toàn. Tin học có thể thực hiện các thuật toán không đề cập trong Toán học vì các thuật toán này chỉ cần thiết cho máy và chỉ có máy mới “biết” thực hiện và có khả năng thực hiện. Tuy nhiên những kiến thức Toán học là rất cần thiết khi học Tin học. Kết hợp Toán học vào các bài tập Tin học thường cho lời giải ngắn gọn, hiệu suất cao. Ngược lại, Toán học có thể được kiểm nghiệm chân lý đúng sai qua thử nghiệm Tin học.

Bộ sách “Bài tập tin học” tập hợp một số bài tập lập trình chọn lọc dành cho học sinh và sinh viên, giới thiệu những tư duy khi giải chúng. Bộ sách chia thành ba tập:

Tập I gồm 90 bài tập về tư duy chọn lọc, lập, vét cạn, đệ quy, chia để trị, sắp xếp, duyệt tuần tự, loang, tìm kiếm nhị phân, thuật toán số học, nguyên lý tham và cách tổ chức dữ liệu trừu tượng.

Tập II gồm 60 bài tập trong chuyên đề thuật toán đồ thị.

Tập III gồm 70 bài tập thuộc các chuyên đề: thuật toán hình học, lập trình động, lập trình tuyến tính, mã hóa, trò chơi và một số thuật toán khác.

Qua phân tích và giới thiệu lời giải, chương trình, tác giả hy vọng giới thiệu được một số tư duy cơ bản trong Tin học.

Tác giả xin chân thành cảm ơn Nhà xuất bản Giáo dục Việt Nam đã cho xuất bản cuốn sách này.

Hà Nội , tháng 02 năm 2012

1. Phương pháp tìm kiếm theo chiều sâu

Để thăm các đỉnh của đồ thị, ngoài phương pháp tìm kiếm theo chiều rộng (BFS – Breadth first search) như đã nêu ở Tập I, vấn đề 7 (Loang), còn dùng phương pháp tìm kiếm theo chiều sâu.

Tìm kiếm theo chiều sâu (DFS - depth first search) có thể mô tả qua hai thủ tục sau:

```
Procedure DFS(x: integer); {Thăm bắt đầu từ đỉnh x chưa thăm}
var y: integer;
begin
    write(x, ' '); {Thông báo đã thăm đỉnh x}
    for y:=1 to n do {Duyệt mọi đỉnh y}
        if (a[x,y]=1) and (trace[y]=0) then begin {y kề với x và y chưa thăm}
            trace[y]:=x; {lưu vết: x được thăm trước y}
            DFS(y); {thăm tiếp từ y}
        end;
    end;
```

```
Procedure duyet; {Duyệt theo chiều sâu thăm tất cả các đỉnh, mỗi đỉnh một lần}
var s: integer;
begin
    fillchar(trace, sizeof(trace), 0);
    for s:=1 to n do
        if trace[s]=0 then DFS(s);
end;
```

2. Tìm số thành phần liên thông.

Ta có thể hiểu khái niệm “liên thông” trên đồ thị vô hướng theo nghĩa đời thường: hai thành phố gọi là liên thông với nhau nếu có đường đi hai chiều giữa chúng (có thể qua các thành phố trung gian). Một tập hợp A gồm các thành phố tạo thành một vùng liên thông cực đại nếu hai thành phố bất kỳ trong A là liên thông và mọi thành phố không thuộc A đều không liên thông với bất kỳ thành phố nào của A.

Trên đồ thị có hướng, nếu thay cung bởi cạnh có đồ thị vô hướng liên thông thì đồ thị có hướng này gọi là *liên thông yếu*.

Trên đồ thị có hướng, nếu giữa hai đỉnh bất kỳ đều có đường đi một chiều nối chúng thì đồ thị này gọi là *liên thông mạnh*.

Để kiểm tra đồ thị có hướng có liên thông mạnh hay không, dùng thuật toán Tarjan (xem bài tập 9 – Kiểm tra dữ liệu) hoặc thuật toán tìm tập thông trị và bị thông trị nhỏ nhất (xem bài tập 14 – Mạng trường học).

Sau đây nói về thuật toán tìm vùng liên thông trên đồ thị vô hướng:

Để tìm các vùng liên thông cực đại (sau đây gọi là vùng liên thông) trên một đồ thị vô hướng, có thể dùng thuật toán loang hoặc thuật toán hợp nhất dần các vùng liên thông để có các vùng liên thông cực đại. Phương pháp hợp nhất cây để tìm vùng liên thông dưới đây cũng là một thuật toán có hiệu suất:

Mỗi cây được đại diện bởi gốc của nó (vì khi biết gốc, có thể duyệt tìm các nút trong cây). Quy ước gốc của cây được gán với một trọng số âm là w mà $|w|$ bằng số nút trong cây, còn mỗi nút khác gốc trong cây được gán với một trọng số dương là số hiệu đỉnh của nút gốc.

Ban đầu mỗi đỉnh được coi là một cây có duy nhất một nút, đó là đỉnh này và nó làm gốc với trọng số là $w(i) = -1, \forall i=1, 2, \dots, n$. Sau đó thực hiện:

Vòng lặp duyệt lần lượt mọi cạnh (x, y) .

Nếu 2 đỉnh x, y của cạnh thuộc hai cây có gốc khác nhau là r_1 và r_2 (x thuộc cây có gốc r_1 , y thuộc cây có gốc r_2) thì hợp nhất hai cây này như sau :

+ Đặt $s = w(r_1) + w(r_2)$.

+ Nếu $|w(r_1)| < |w(r_2)|$ (số nút trong cây r_1 ít hơn số nút trong cây r_2) thì gán cho các nút trong cây có gốc r_1 trọng số là r_2 , gán cho r_2 trọng số là s (r_2 là gốc của cây sau hợp nhất);

+ Nếu $|w(r_1)| > |w(r_2)|$ gán cho các nút trong cây gốc r_2 trọng số r_1 , gán cho r_1 trọng số s (r_1 là gốc của cây sau hợp nhất)

Lặp cho đến khi duyệt hết các cạnh.

Ví dụ. Chương trình sau đây cho biết số vùng liên thông trong đồ thị vô hướng và liệt kê các đỉnh trong từng vùng.

Dữ liệu vào lấy từ tệp stplt.in: Dòng đầu là số đỉnh của đồ thị. Các dòng sau, mỗi dòng hai số i và j thể hiện có cạnh $(i; j)$.

Kết quả ra ghi vào tệp stplt.out: Dòng đầu tiên ghi số sv là vùng liên thông. sv dòng sau, mỗi dòng ghi các đỉnh của cùng một vùng liên thông khác nhau.

Ví dụ

Stplt.in	Stplt.out
1 2	2
1 4	1 2 3 4
2 3	5 6 7
3 4	
5 6	
6 7	

```

uses crt;
const max      = 5000;
      fileinp   = 'stplt.in';
      fileout   = 'stplt.out';
var   n        : integer;
      w        : array[1..max] of integer;
      sv       : integer;

function find_root(x : integer) : integer; {tìm gốc của cây chứa x}
var i : integer;
begin
  i:=x;
  while w[i]>0 do {i có trọng số dương nên chưa phải là gốc}
    i:=w[i];
  find_root:=i;
end;

procedure change(x : integer; y: integer); {các đỉnh nằm trong cây có
gốc là x được xác nhận thuộc cây có gốc mới là y}
var i : integer;
begin
  for i:=1 to n do
    if w[i]= x then begin
      w[i]:= y;
    end;

```

```

end;
procedure union(x,y:integer); {hợp nhất hai cây có gốc là x và y}
var temp:integer;
begin
    temp := w[x]+ w[y]; {trọng số của gốc của cây hợp nhất}
    if w[x] > w[y] then begin {cây có gốc x có ít nút hơn cây có gốc là y}
        w[x] := y; {xác nhận x thuộc cây có gốc là y}
        change(x, y); {các nút thuộc cây có gốc x được xác nhận vào cây có gốc y}
        w[y] := temp; {ghi nhận trọng số của gốc cây hợp nhất}
    end
    else {cây có gốc x có số nút không ít hơn số nút trong cây gốc y}
    begin
        w[y] := x; {xác nhận y thuộc cây có gốc là x}
        change(y, x); {các nút thuộc cây có gốc y được xác nhận vào cây có gốc x}
        w[x] := temp; {ghi nhận trọng số của gốc cây hợp nhất}
    end;
end;

procedure creatnet; {hợp nhất các cây, mỗi cây xác nhận một vùng liên thông}
var f          : text;
    i          : longint;
    x,y,r1,r2  : integer;
begin
    assign(f,fileinp); reset(f);
    readln(f,n); {đọc số đỉnh của đồ thị vô hướng}
    for i:=1 to n do w[i]:=-1; {Khởi trị trọng số các gốc của n cây}
    while not seekeof(f) do begin {duyệt mọi cạnh}
        readln(f,x,y); {đọc một cạnh có hai đầu là x và y}
        r1 := find_root(x); {r1 là gốc của cây chứa đỉnh x}
        r2 := find_root(y); {r2 là gốc của cây chứa đỉnh y}
        if r1<>r2 then begin {x và y thuộc hai cây có gốc khác nhau r1 và r2}
            union(r1,r2); {hợp nhất hai cây có gốc là r1 và r2}
        end;
    end;
end;

procedure write_out;
var f : text;
    i,j,r1,r2 : integer;
begin
    assign(f,fileout);
    rewrite(f);
    sv := 0; {Khởi trị số vùng liên thông}
    for i:=1 to n do
        if w[i]<0 then
            inc(sv); {đếm số vùng liên thông bằng đếm đỉnh có trọng số âm}

```

```

writeln(f,sv) ; {ghi kết quả số vùng liên thông}
for i:=1 to n do
if w[i]<0 then begin {Lần lượt tìm các đỉnh i là gốc}
    write(f,i,' '); {ghi đỉnh i}
    for j:=1 to n do {ghi các đỉnh thuộc vùng liên thông có chứa i}
        if w[j]=i then write(f,j,' ');
    writeln(f);
end;
close(f);
end;
BEGIN
    creatnet;
    write_out;
END.

```

Lưu ý. Nếu không cần liệt kê các đỉnh thuộc cùng một vùng liên thông (chỉ yêu cầu tìm số vùng liên thông) thì không cần thủ tục change(); khi đó thuật toán hợp nhất cây để tìm số vùng liên thông sẽ hiệu suất hơn nhiều thuật toán khác cùng giải vấn đề này.

3. Tìm cây khung, cây khung ngắn nhất

a. Cây. Cây là đồ thị vô hướng, hữu hạn, liên thông và không có chu trình đơn (chu trình đi qua mỗi cạnh trong nó đúng một lần).

a) Định lý:

Nếu H là đồ thị vô hướng, n đỉnh ($n > 1$) và H có một trong sáu tính chất sau thì H là cây:

- (1) H liên thông và không có chu trình đơn.
- (2) H không có chu trình đơn và có $n - 1$ cạnh.
- (3) H có $n - 1$ cạnh và liên thông.
- (4) H không có chu trình đơn và nếu thêm một cạnh nối giữa hai đỉnh bất kỳ không kề nhau thì đồ thị có chu trình đơn (và chỉ một chu trình mà thôi).
- (5) H liên thông và nếu bớt một cạnh bất kỳ thì H không liên thông.
- (6) Mọi cặp đỉnh của H đều được nối với nhau bằng một đường đi đơn duy nhất.

b. Cây khung.

Cho đồ thị vô hướng, hữu hạn và liên thông G . Tập hợp các đỉnh của G cùng một số cạnh của nó tạo thành một cây được gọi là một cây khung của G .

Bài toán tìm cây khung cũng là một bài toán có nhiều ý nghĩa trong thực tế. Ví dụ: Trong mạng giao thông của N thành phố giữa một số cặp hai thành phố nào đó có đúng một đường đi hai chiều trực tiếp nối với nhau sao cho hai thành phố bất kỳ trong mạng giao thông đều có đường đi tới nhau, hãy bỏ bớt một số đường đi trực tiếp giữa một số cặp hai thành phố nào đó sao cho vẫn bảo đảm hai thành phố bất kỳ trong mạng giao thông vẫn có đường đi tới nhau.

Sau đây là một số thuật toán tìm cây khung trên đồ thị liên thông:

Thuật toán 1 (duyệt thăm các đỉnh, mỗi đỉnh 1 lần) : Vì đồ thị liên thông, nên thăm được đủ n đỉnh (qua $n-1$ cạnh) được cây khung. Có thể thăm bằng DFS hoặc BFS.

Thuật toán 2 (hợp nhất dần các vùng liên thông) :

Mỗi lần hợp nhất 2 vùng liên thông khác nhau bằng một cạnh nối 2 vùng này thì cạnh đó được nạp vào cây khung đang hình thành. Quá trình chấm dứt khi nạp đủ $n-1$ cạnh.

Thuật toán 3 (hợp nhất dần các cây) :

Mỗi lần hợp nhất 2 cây có gốc khác nhau bằng một cạnh của đồ thị (nối 2 đỉnh thuộc 2 cây này) thì cạnh đó được xác nhận là một cạnh của cây khung đang hình thành. Quá trình chấm dứt khi nạp đủ $n-1$ cạnh của cây khung.

Chương trình (thuật toán 3)

```
const max      = 5000;
      fileinp   = 'ck.in';
      fileout   = 'ck.out';
var    n        : integer;
      w,d,c     : array[1..max] of integer;
      sl        : integer;
```

```
function find_root(x:integer):integer; {tìm gốc của cây có chứa đỉnh x}
var i    : integer;
```



```

begin
    i:=x;
    while w[i]>0 do i:=w[i];
    find_root:=i;
end;

Procedure union(x,y:integer); {hợp nhất 2 cây có gốc là x và y}
var temp:integer;
begin
    temp := w[x]+w[y];
    if w[x]>w[y] then begin {cây chứa đỉnh x có ít nút hơn}
        w[x] := y; {xác nhận x thuộc cây có gốc là y}
        w[y] := temp; {xác nhận trọng số của gốc y - gốc mới của cây hợp nhất}
    end
    else begin {cây chứa đỉnh y có ít nút hơn}
        w[y] := x; {xác nhận y thuộc cây có gốc là x}
        w[x] := temp; {xác nhận trọng số của gốc x - gốc mới của cây hợp nhất}
    end;
end;

end;

Procedure unify;    {hợp nhất các cây}
var f                : text;
    i                : longint;
    x,y,r1,r2        : integer;
begin
    assign(f,fileinp); reset(f);
    readln(f,n);
    for i:=1 to n do w[i]:=-1;
    sl := 0; {số cạnh của cây khung}
    while not seekeof(f) do begin {duyet các cạnh của đồ thị}
        if sl=n-1 then exit; {điều kiện đủ để hoàn thành cây khung}
        readln(f,x,y); {đọc lấy một cạnh của đồ thị}
        r1:=find_root(x); {gốc của cây có chứa đỉnh x}
        r2:=find_root(y); {gốc của cây có chứa đỉnh y}
        if r1<>r2 then begin
            inc(sl); {số cạnh đã nạp vào cây khung}
            {d và c: 2 mảng lưu cạnh nạp vào cây khung}
            d[sl] := x; {cạnh thứ sl của cây khung có đầu thứ nhất là x}
            c[sl] := y; {cạnh thứ sl của cây khung có đầu thứ hai là y}
        end;
    end;
end;

```

```

        union(r1, r2); {hợp nhất 2 cây khác nhau}
    end;
end;
end;
Procedure write_out;
var f : text;
    i, j, r1, r2 : integer;
begin
    assign(f, fileout);
    rewrite(f);
    if sl < n-1 then write(f, 'không tạo thành cây khung') else
    for i:=1 to n-1 do writeln(f, d[i], ' ', c[i]);
    close(f);
end;
BEGIN
    unify;
    write_out;
END.

```

c. Cây khung ngắn nhất

Cho đồ thị G vô hướng, liên thông và có trọng số không âm. Cây khung ngắn nhất của đồ thị G là cây khung có tổng trọng số trên các cạnh của nó là nhỏ nhất (gọi là trọng số của cây khung ngắn nhất). Có thể tìm cây khung ngắn nhất bằng các thuật toán Prim, Kruskal hoặc dùng phương pháp gần nhần.

Nội dung của thuật toán Kruskal là: nạp dần các cạnh ngắn nhất vào cây khung nếu cạnh ấy không tạo thành chu trình với các cạnh đã nạp.

Thực hiện cụ thể gồm các bước sau :

(a) Sắp xếp các cạnh tăng dần theo trọng số;

(b) Lần lượt kết nạp các cạnh có trọng số nhỏ nhất trong các cạnh còn lại vào cây nếu sau khi kết nạp cạnh này không tạo thành chu trình. Để thực hiện được những yêu cầu này, ta dựa vào thuật toán hợp nhất cây nêu trên. Quá trình này dừng khi hợp nhất được một cây có đủ $n-1$ cạnh.

Nội dung của thuật toán Prim là: Nạp dần các đỉnh vào cây khung. Mỗi lần chọn một đỉnh chưa nạp là đỉnh kề và gần các đỉnh đã nạp nhất.

Thực hiện cụ thể :

(a) Nạp một đỉnh đầu tiên vào cây khung (nếu đầu bài không yêu cầu nhất thiết phải là đỉnh nào thì chọn đỉnh tùy ý thường là đỉnh 1).

(b) Lần lượt nạp $n-1$ đỉnh còn lại (tương ứng là $n-1$ cạnh) vào cây khung bằng cách: mỗi lần chọn một cạnh có trọng số nhỏ nhất mà một đầu của cạnh đã thuộc cây, đầu kia chưa thuộc cây (nghĩa là chọn một đỉnh gần các đỉnh đã nạp vào cây nhất).

Thuật toán Prim còn được thực hiện bởi kỹ thuật gán nhãn như sau:

Mỗi đỉnh i của đồ thị được gán cho một nhãn là cặp số (i_0, v) với ý nghĩa : trong các đỉnh đã nạp vào cây khung thì i_0 kề và gần i nhất, v là độ dài cạnh (i_0, i) . Nhãn v gọi là nhãn “khoảng cách ngắn nhất” giữa đỉnh i tới các đỉnh của cây khung đang hình thành (gọi tắt là “khoảng cách” từ i tới cây). Nhãn i_0 gọi là nhãn “đỉnh trước” của i . Các bước cụ thể:

(a) Khởi trị và gán nhãn ban đầu: Lấy một đỉnh i tùy ý (nếu không yêu cầu gì về đỉnh đầu tiên này) đưa vào tập đỉnh của cây khung ngắn nhất (gọi đỉnh này là i_0). Khi đó tập đỉnh của cây khung ngắn nhất đang hình thành là $D = \{i_0\}$ có duy nhất một đỉnh được nạp là i_0 . Tập cạnh của cây khung ngắn nhất là $C = \emptyset$ (tập rỗng, chưa có cạnh nào). Với mỗi đỉnh i kề i_0 ta gán cho đỉnh i một nhãn là cặp số (i_0, v) với v là khoảng cách giữa i_0 và i . Các đỉnh i không kề i_0 thì gán cho i nhãn $(0, \infty)$.

(b) Thực hiện vòng lặp cho đến khi chọn đủ $n-1$ cạnh vào cây khung (hoặc đã duyệt hết các đỉnh)

Vòng lặp thực hiện các thao tác sau :

- **Kết nạp** : Chọn đỉnh i có nhãn (i_0, v) ngoài tập D mà nhãn v nhỏ nhất, kết nạp i vào D . Vậy $D = D + \{i\}$. Kết nạp cạnh (i_0, i) vào tập cạnh C . Vậy $C = C + \{(i_0, i)\}$.
- Nếu số cạnh bằng $n-1$ thì kết thúc vòng lặp.
- **Sửa nhãn** : Sau khi đỉnh i được nạp vào cây khung ta gọi nó là i_0 , nó tạo cho một số đỉnh kề với nó có thể sẽ gần cây khung hơn, nên cần phải sửa lại nhãn của các đỉnh này. Cụ thể : xét các đỉnh j chưa thuộc

Đ, kề với i_0 , giả sử j đang có nhãn là (k, v) . Đỉnh j sẽ được sửa nhãn nếu độ dài cung (i_0, j) là e mà $e < v$ thì đỉnh j có nhãn mới là (i_0, e) vì khoảng cách từ j đến cây khung bây giờ là khoảng cách e .

Chương trình

```

const fi  = 'prim.in';
      fo  = 'prim.out';
      max = 100;
type m1 = array[1..max,1..max] of integer;
      m2 = array[1..max] of integer;
var   a    : m1; {ma trận trọng số}
      d    : m2; {đánh dấu đỉnh đã nạp vào cây khung}
      dl,d2 : m2; {quản lý các cạnh nạp vào cây khung}
      v,t   : m2; {v: nhãn khoảng cách, t: lưu vết}
      n,i0,dem : integer;
Procedure docf;
var f : text; i,j,x : integer;
begin
  assign(f,fi); reset(f);
  readln(f,n);
  while not seekeof(f) do begin
    read(f,i,j,x); a[i,j] := x; a[j,i] := x;
  end;
  close(f);
end;
Procedure khoitri;
var i : integer;
begin
  for i:=1 to n do d[i] := 0;
  d[1] := 1; {đánh dấu nạp đỉnh 1 vào cây khung ngắn nhất}
  i0 := 1; {đỉnh vừa nạp}
  v[1] := 0; {nhãn khoảng cách của đỉnh 1 tới cây khung}
  t[1] := 0; {lưu vết trước đỉnh 1 là đỉnh 0}
  for i:=2 to n do {Khởi trị nhãn các đỉnh còn lại}
    if a[1,i]=0 then begin v[i] := maxint; t[i] := 0;
    end else begin v[i] := a[1,i]; t[i] := 1;
    end;
end;
Procedure ketnap; {Kết nạp thêm một đỉnh}
var min,i,li : integer;
begin
  {Tìm đỉnh i chưa nạp, có nhãn nhỏ nhất}
  min := maxint;
  for i:=1 to n do

```

```

if d[i]=0 then
if v[i]<min then begin min := v[i]; li := i; end;
i0 := li; {i0 là đỉnh chưa nạp, có nhãn nhỏ nhất}
d[i0] := 1; {đánh dấu đã nạp đỉnh i0 vào cây khung}
inc(dem); {tăng biến đếm số cạnh đã nạp vào cây khung}
{hru cạnh vừa nạp}
d1[dem] := t[i0];
d2[dem] := i0;
end;
Procedure suanhan; {Sửa nhãn các đỉnh i chưa nạp, kề với đỉnh i0 vừa nạp}
var i,j : integer;
begin
for i:=1 to n do
if (a[i0,i]>0)and (d[i]=0) then
if v[i]>a[i0,i] then begin
v[i] := a[i0,i]; t[i] := i0;
end;
end;
Procedure gan_nhan;
begin
dem := 0;
khoitri;
repeat
ketnap;
suanhan;
until dem=n-1;
end;
Procedure ghif;
var f : text; i,tong : integer;
begin
tong := 0;
assign(f,fo); rewrite(f);
for i:=1 to n-1 do begin
writeln(f,d1[i], ' ',d2[i]);
tong := tong + a[d1[i],d2[i]];
end;
writeln(f,tong);
close(f);
end;
BEGIN
docf;
gan_nhan;
ghif;
END.

```

4. Tìm đường đi ngắn nhất trên đồ thị có trọng số

a) Thuật toán Ford Bellman

Bài toán. Tìm đường đi ngắn nhất từ đỉnh xuất phát x đến đỉnh đích y trên đồ thị đơn, có hướng, có trọng số và không có chu trình âm (là chu trình có tổng trọng số trên các cung của nó bằng số âm).

Thuật toán Ford Bellman.

Thuật toán được thực hiện bằng kỹ thuật gán nhãn cho các đỉnh: Giả sử ma trận trọng số là $A(N,N)$, có $A[i,j]$ là độ dài (coi là trọng số) cung nối trực tiếp đỉnh i và đỉnh j . Mỗi đỉnh i ($1 \leq i \leq N$) được tạo một nhãn là $L(i)$ thể hiện là độ dài

đường đi ngắn nhất từ đỉnh xuất phát x tới i . Đương nhiên $L(x)=0$. Nếu có cung nối trực tiếp từ i tới j với trọng số là $A[i,j]$ và $L[j] > L[i] + A[i,j]$ thì sửa lại nhãn của j là $L[j] = L[i] + A[i,j]$ thể hiện đường đi ngắn nhất từ x tới j bây giờ dài là $L[j]$, và xác nhận vết $T[j]=i$ thể hiện trước khi tới j phải tới i .

Có thể chứng minh được rằng: sau $N-1$ lần thực hiện sửa nhãn cho thì $L[y]$ sẽ là độ dài đường đi ngắn nhất từ đỉnh x đến đỉnh y . Các bước cụ thể thực hiện thuật toán là:

Bước 1. Khởi trị mảng trọng số $A[i,i]=0$ với mọi i ($1 \leq i \leq N$), Đọc $A[i,j]$ từ tệp dữ liệu vào. $A[i,j]=+\infty$ với $i \neq j$.

Khởi trị mảng nhãn: $L[i]=+\infty$ với mọi i ($1 \leq i \leq N, i \neq x$) và $L[x]=0$.

Khởi trị mảng ghi vết đường đi ngắn nhất: $T[i]=0$ với mọi i ($1 \leq i \leq N$).

Bước 2. Thực hiện $N-1$ lần sửa nhãn để tối ưu dần nhãn của các đỉnh:

For $k:=1$ to $n-1$ do

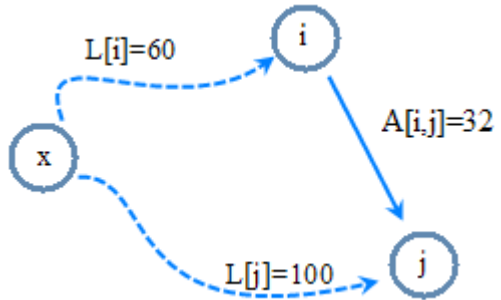
For $i:=1$ to n do

For $j:=1$ to n do

If $A[i,j] > 0$ then

If $(j \neq x)$ and $(L[j] > L[i] + A[i,j])$ then begin

$L[j] := L[i] + A[i,j];$



$T[j] := i;$

End;

Bước 3. Dựa vào mảng vết T , tìm đường đi ngược từ y về x . Hiện đường đi từ x tới y .

Lưu ý 1. Khi áp dụng với đồ thị vô hướng thì mỗi cạnh (i,j) được coi là hai cung (i,j) và (j,i) cùng trọng số.

Lưu ý 2. Nếu sau $N-1$ lần sửa nhãn, vẫn có $L[y] = +\infty$ thì không tồn tại đường đi ngắn nhất từ x đến y .

Lưu ý 3. Thuật toán còn có thể dùng tìm đường đi dài nhất (thay điều kiện $L[j] > L[i] + A[i,j]$ thành $L[j] < L[i] + A[i,j]$)

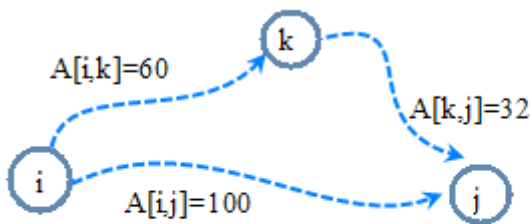
b) Thuật toán Floyd

Bài toán. Tìm đường đi ngắn nhất giữa **mọi cặp đỉnh** trên đồ thị đơn, có hướng, có trọng số và không có chu trình âm.

Thuật toán Floyd. Cho ma trận trọng số $A(N,N)$ với $A[i,j]$ là độ dài cung nối trực tiếp từ i tới j ; sẽ xây dựng $A[i,j]$ thành nhãn độ dài đường đi ngắn nhất từ i tới j bằng công thức:

$$A[i,j]_{\text{mới}} = \text{Min}\{A[i,j]_{\text{cũ}}, A[i,k] + A[k,j], k=1, 2, \dots, N\} \quad \forall i, j=1, 2, \dots, N$$

Và ghi nhận lại đỉnh k là đỉnh trung gian trên đường đi ngắn nhất từ i tới j bằng cách gán vết $T[i,j]=k$.



Thay lại nhãn $A[i,j]=92$, lưu vết $T[i,j]=k$

Các bước cụ thể thực hiện thuật toán là:

Bước 1. Khởi trị: Mảng $A(N,N)$ với $A[i,i]=0$ với mọi $i = 1, 2, \dots, N$ và Đọc $A[i,j]$ từ tệp dữ liệu vào. $A[i,j] = +\infty$ với $i \neq j$.

Khởi trị mảng vết $T(N,N)$ mà $T[i,j] := j$ với mọi $i, j = 1, 2, \dots, N$.

Bước 2. Từ mảng trọng số $A(N,N)$ xây dựng thành mảng nhãn đường đi ngắn nhất $A(N,N)$:

```
For k:=1 to n do
  For i:=1 to n do if  $A[i,k] < +\infty$  then
    For j:=1 to n do if  $A[k,j] < +\infty$  then
      If  $A[i,k] + A[k,j] < A[i,j]$  then begin
         $A[i,j] := A[i,k] + A[k,j]$ ;
         $T[i,j] := k$ ;
      end;
    end;
  end;
```

Bước 3. Dựa vào vết $T(N,N)$ tìm ra đường đi ngắn nhất từ một xuất phát x tới đích y , với (x,y) tùy ý nếu $A[x,y] < +\infty$. Các đỉnh trên đường đi ngắn nhất từ x đến y được lưu vào mảng $PATH$ có thể tìm bằng thủ tục đệ quy sau (trong đó biến *count* dùng để đếm số đỉnh trên hành trình):

```
Procedure find(x,y : integer);
  var k : integer;
  begin
    k := t[x,y];
    if k=0 then begin
      if (count=0) then
        inc(count); path[count] := x;
      end;
      inc(count); path[count] := y;
    end
    else begin
      find(x, k); find(k, y);
    end;
  end;
```

c) Thuật toán Dijkstra

Bài toán: Tìm đường đi ngắn nhất từ đỉnh s đến đỉnh t trên đồ thị có hướng với trọng số cung (i,j) là $C[i,j] \geq 0$ (gọi là trọng số không âm).

Thuật toán Dijkstra gồm các bước thực hiện sau:

Bước 1. Khởi trị. Khởi trị nhãn đường đi ngắn nhất từ đỉnh s tới đỉnh i là $d[i] := C[s,i]$ (nếu không có đường đi trực tiếp từ s đến i thì $C[s,i]$ bằng vô cùng. Gán $d[s] = 0$).

Lưu đỉnh trước khi tới i trên hành trình ngắn nhất là $Tr[i] := s$.

Đánh dấu mọi đỉnh i là tự do (nhãn $d[i]$ chưa tối ưu): $DX[i]:=false$

Bước 2. (vòng lặp vô hạn)

Tìm đỉnh i_0 là đỉnh tự do có nhãn $d[i_0]$ nhỏ nhất.

Nếu không tìm được i_0 (nghĩa là $i_0 = 0$) hoặc $i_0 = t$ thì thoát khỏi vòng lặp, còn không thì:

+ Đánh dấu i_0 đã được cố định nhãn $DX[i_0]:=True$ (gọi i_0 là đỉnh được cố định nhãn)

+ Sửa nhãn cho các đỉnh j tự do kề với i_0 theo công thức $d[j] = \min\{d[j], d[i_0]+C[i_0, j]\}$ và ghi lưu lại đỉnh trước j là i_0 : $Tr[j] := i_0$.

Bước 3. Tìm và ghi kết quả. Dựa vào giá trị $d[t]$ và mảng $Tr(N)$ để kết luận thích hợp.

d) Tìm đường đi ngắn nhất trên đồ thị không có chu trình

Trên đồ thị có hướng, không chu trình có thể đánh lại số hiệu các đỉnh sao cho mỗi cung nối từ đỉnh có số hiệu nhỏ đến đỉnh có số hiệu lớn (gọi là sắp lại thứ tự bộ phận, hay là sắp tô pô). Công việc dễ dàng tiến hành bằng tìm kiếm theo chiều rộng, đỉnh nào được thăm trước có số hiệu nhỏ hơn.

Sau đó sửa nhãn đỉnh theo số hiệu đỉnh tăng dần. (xem bài 18 – xếp công việc).

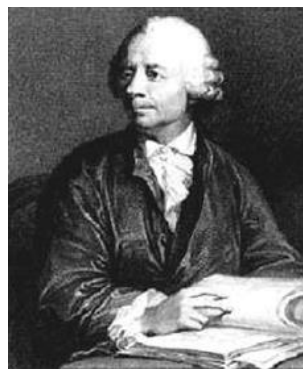
Việc sắp tô pô các đỉnh cũng có thể tiến hành bằng tìm kiếm theo chiều sâu trên đồ thị đảo chiều và đánh số các đỉnh theo thứ tự duyệt.

5. Chu trình

Xét đồ thị vô hướng $G=(V,E)$. Một đường đi từ một đỉnh thuộc V lại trở về đỉnh đó tạo thành một chu trình. Trên chu trình, mỗi cạnh chỉ xuất hiện một lần gọi là chu trình đơn.

Tập chu trình cơ sở của G là tập nhiều nhất các chu trình thỏa mãn: mỗi chu trình có một cạnh riêng không thuộc chu trình khác.

Bài toán “Bảy chiếc cầu trên dòng sông Pregel của thành phố Königsberg” được Euler đề



Leonhard Euler [1707-1783]

xuất năm 1735 trong lý thuyết đồ thị là bài toán tìm chu trình đi qua mọi cạnh của đồ thị, mỗi cạnh chỉ qua một lần (bài toán bảy chiếc cầu này là vô nghiệm).

Chu trình đơn đi qua tất cả các cạnh thuộc E , mỗi cạnh đúng một lần gọi là chu trình Euler. Đường đi đơn qua tất cả các cạnh thuộc E , mỗi cạnh đúng một lần gọi là đường đi Euler.

Chu trình đơn đi qua tất cả các đỉnh thuộc V , mỗi đỉnh một lần gọi là chu trình Haminton. Đường đi đơn đi qua tất cả các đỉnh thuộc V , mỗi đỉnh một lần gọi là đường đi Haminton.

Định lý 1 (Euler). Một đồ thị vô hướng liên thông $G=(V,E)$ có chu trình Euler khi và chỉ khi mọi đỉnh của nó đều có bậc chẵn.

Hệ quả. Một đồ thị vô hướng liên thông $G=(V,E)$ có đường đi Euler khi và chỉ khi nó có đúng hai đỉnh bậc lẻ.

Định lý 1. Một đồ thị có hướng liên thông yếu có chu trình Euler khi và chỉ khi mọi đỉnh có số cung vào bằng số cung ra.

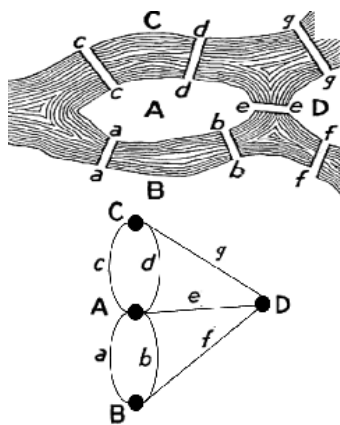
Hệ quả 1. Đồ thị có hướng liên thông yếu nếu có chu trình Euler thì liên thông mạnh.

Hệ quả 2. Đồ thị có hướng liên thông yếu có đường đi Euler nhưng không có chu trình Euler khi và chỉ khi có hai đỉnh i và j sao cho: (số cung vào i) – (số cung ra khỏi i) = (số cung ra khỏi j) – (số cung vào j).

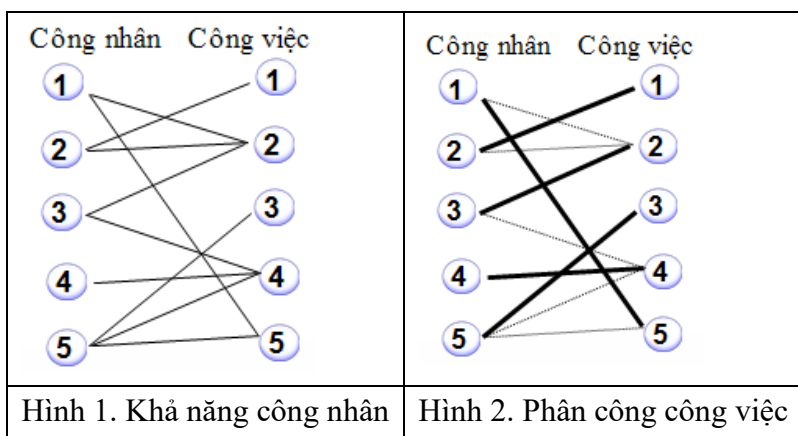
6. Bộ ghép trên đồ thị hai phía

Trước hết chúng ta xét bài toán sau:

Bài toán 1. Có N công nhân có số hiệu từ 1 đến N và N công việc cũng đánh số từ 1 đến N . Mỗi công nhân có thể làm được một số công việc nào đó trong N công việc này. Nhưng mỗi công nhân chỉ được giao làm nhiều nhất là một việc. Hãy lập lịch bố trí công việc sao cho nhiều công nhân được nhận việc.



Ví dụ: Khả năng của công nhân được cho như hình 1 dưới đây:



Công nhân 1 có thể làm được các công việc 2 và 5, công nhân 2 có thể làm các việc 1 và 2, công nhân 3 có thể làm được các việc 2 và 4, công nhân 4 chỉ làm được công việc 4, công nhân 5 có thể làm được các việc 3, 4 và 5. Ta có thể giao việc cho công nhân như hình 2: Công nhân 1 làm việc 5, công nhân 2 làm việc 1, công nhân 3 làm việc 2, công nhân 4 làm việc 4, công nhân 5 làm việc 3.

Bài toán này được giải nhờ thuật toán tìm bộ ghép cực đại trên đồ thị hai phía. Để tìm hiểu thuật toán này, chúng ta sơ lược nêu một số khái niệm cần thiết sau đây:

- Đồ thị hai phía là gì?
- Bộ ghép là gì?
- Bộ ghép cực đại là gì?

Đồ thị hai phía là đồ thị vô hướng $G(V,E)$. Trong đó tập đỉnh là V được chia thành hai tập không giao nhau là X và Y . Tập cạnh E chỉ gồm các cạnh $e=(i,j)$ với $i \in X, j \in Y$.

Bộ ghép trên đồ thị hai phía là tập M gồm một số cạnh thuộc E mà các cạnh này không có đỉnh chung.

Mỗi cạnh thuộc bộ ghép được gọi là cạnh đậm (cạnh đã ghép), hai đầu cạnh đậm được gọi là đỉnh đậm (đỉnh đã ghép). Các cạnh còn lại là cạnh nhạt (cạnh chưa ghép), các đỉnh còn lại là đỉnh nhạt (đỉnh chưa ghép).

Bộ ghép cực đại là bộ ghép gồm nhiều cạnh nhạt.

Một đường đi theo hướng bắt đầu từ đỉnh nhạt bên X , xen kẽ cạnh nhạt và cạnh đậm và kết thúc tại đỉnh nhạt bên Y được gọi là một **đường mở**.

Rõ ràng trên một đường mở, số cạnh nhạt nhiều hơn số cạnh đậm là 1. Do đó nếu trên đường mở ta đổi màu các cạnh thì số cạnh đậm được tăng thêm 1, nghĩa là bộ ghép được tăng thêm một cạnh.

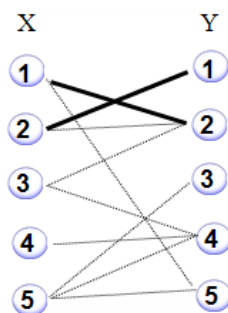
a) Thuật toán tìm bộ ghép cực đại:

Vòng lặp: Duyệt lần lượt các đỉnh nhạt bên X :

- **Tìm đường mở từ mỗi đỉnh nhạt**
- **Nếu tìm được đường mở thì đổi màu các cạnh trên đường mở**

Để thực hiện thuật toán trên, chúng ta dùng một vài kỹ năng lập trình như sau: Dùng mảng một chiều $A(N)$ để xác nhận các đỉnh đậm và nhạt bên X và mảng một chiều $B(N)$ để xác nhận các đỉnh đậm và nhạt bên Y . Giả sử có cạnh đậm $e=(i,j)$ với đỉnh i thuộc X với đỉnh j thuộc Y thì ta ghi nhận hai đỉnh i và j là đỉnh đậm bằng cách gán $A[i]:=j$ (dương), $B[j]:=i$ (dương). Còn khi $A[i]=0$ thì đỉnh i là đỉnh nhạt bên X hoặc $B[j]=0$ thì đỉnh j là đỉnh nhạt bên Y ; khi đó các cạnh có một đầu là $i \in X$ hoặc một đầu là $j \in Y$ sẽ là cạnh nhạt (vì không thỏa mãn $A[i]=j$ và $B[j]=i$).

Ví dụ: trong hình vẽ bên có $A[2]=1$, $B[1]=2$ cho biết cạnh $(2,1)$ là cạnh đậm có đỉnh 2 bên X và đỉnh 1 bên Y là đỉnh đậm. Còn biết $A[3]=0$, $B[2]=1$, vậy cạnh $(3,2)$ là cạnh nhạt có đỉnh 3 bên X là đỉnh nhạt và đỉnh 2 bên Y là đỉnh đậm...



Một đường mở gồm các cạnh liên tiếp là (3,2), (1,2), (1,5) (số thứ nhất là đỉnh bên X, số thứ hai là đỉnh bên Y). Trong đó chỉ có cạnh (1,2) là cạnh đậm. Bắt đầu đường mở là đỉnh nhạ 3 bên X, kết thúc đường mở là đỉnh nhạ 5 bên Y.

Nếu đổi màu các cạnh trên đường mở này thì các cạnh (3,2) và (1,5) thành cạnh đậm còn (1,2) đổi thành cạnh nhạ. Bộ ghép được tăng thêm một cạnh.

Chương trình giải bài toán 1 (Tìm bộ ghép cực đại).

```
const max      = 100;    maxq = 10000;
    fi        = 'boghep1.in'; fo      = 'boghep1.out';
var  c        : array[1..max,1..max] of byte;
    q         : array[1..maxq] of integer;
    a,        {đánh dấu các đỉnh là đậm, nhạ bên X}
    b,        {đánh dấu các đỉnh là đậm, nhạ bên Y}
    t : array[1..max] of byte; {vết trên đường mở}
    n : byte;
```

Procedure khai_tri;

```
begin
    fillchar(c,sizeof(c),0);
    fillchar(a,sizeof(a),0);
    fillchar(b,sizeof(b),0);
end;
```

Procedure doc_file;

```
var i,j : byte;  f : text;
begin
    assign(f,fi); reset(f);
    readln(f,n); {số công nhân, số công việc}
    repeat
        readln(f,i,j); {công nhân i có thể làm được công việc j}
        if i<>0 then c[i,j] := 1;
    until i=0; {Kết thúc tệp input bởi số 0}
    close(f);
end;
```

Procedure tang_canh(j:byte); {đổi màu các cạnh từ đỉnh cuối đường mở là j}

```
var i,p : byte;
begin
    repeat
        i := t[j]; {cạnh cuối cùng của đường mở là cạnh (i,j), i thuộc X, j thuộc Y}
        p := a[i]; {đỉnh bên phải của cạnh đậm cuối cùng trên đường mở}
        a[i]:= j; {chuyển cạnh nhạ cuối cùng thành cạnh đậm}
        b[j]:= i; {như dòng chú thích trên}
```

```

    j := p;    {chuyển lên đỉnh nhật tiếp theo bên Y}
until j=0;
end;
Procedure tim_duong_di(i : byte); {Tìm đường mở từ đỉnh nhật i bên X}
var j : byte;
    dau, cuoi : integer;
begin
    fillchar(t,sizeof(t),0);
    dau := 0;
    cuoi := 1;
    q[cuoi] := i;
    repeat
        inc(dau);
        i := q[dau];
        for j:=1 to n do {duyet các đỉnh j bên Y}
            if (t[j]=0) and (c[i,j]=1) then begin {điều kiện chọn j}
                t[j] := i; {xác nhận i là đỉnh kề trước của j trên đường mở}
                if b[j]=0 then begin {nếu j là đỉnh nhật bên Y thì kết thúc đường mở}
                    tang_canh(j); {đổi màu trên đường mở để tăng cạnh được ghép}
                    exit;
                end
            else begin {b[j]>0: j là đỉnh đậm, còn có thể kéo dài đường mở}
                inc(cuoi);
                q[cuoi]:=b[j]; { nạp đỉnh b[j] là đỉnh đậm bên X vào cuối hàng đợi}
            end;
        end;
    until dau>cuoi; {Kết thúc tìm một đường mở}
end;
Procedure thuc_hien;
var i : byte;
begin
    for i:= 1 to n do
        if a[i]=0 then
            tim_duong_di(i); {Tìm đường mở bắt đầu từ đỉnh nhật a[i] bên X}
    end;
Procedure ghi_file;
var i : byte;
begin
    assign(f,fo); rewrite(f);
    for i:=1 to n do
        if a[i]<>0 then writeln(f,i,' ',a[i]);
    close(f);
end;
BEGIN
    khoi_tri; doc_file; thuc_hien; ghi_file;

```

END.

Ngoài thuật toán trên, còn có thuật toán Hopcroft – Karp với ý tưởng tìm đồng thời tìm nhiều đường mở không có đỉnh chung, đạt hiệu quả hơn (xem bài tập 50 - Minimum path cover).

b) Thuật toán tìm bộ ghép có tổng trọng số lớn nhất.

Bài toán 2. Có N công nhân có số hiệu từ 1 đến N và M máy đánh số từ 1 đến M . Mỗi công nhân có thể biết sử dụng một số máy nào đó trong M máy này. Máy i giao cho công nhân j điều khiển sẽ tạo lợi nhuận là C_{ij} . Mỗi công nhân chỉ được giao nhiều nhất là một máy. Hãy lập lịch bố trí máy sao cho lợi nhuận tạo được nhiều nhất.

Đây là bài toán tìm bộ ghép có tổng trọng số trên các cạnh của bộ ghép là lớn nhất, bài toán được giải dựa trên thuật toán gồm các bước sau:

Bước 1: Khởi trị tạo bộ ghép ban đầu

- Xét đồ thị hai phía (X là phía máy có M đỉnh, Y là phía thợ có N đỉnh). Cạnh (i, j) có trọng số $C[i, j]$ là lợi nhuận khi máy i giao cho thợ j . Sử dụng hai mảng một chiều là FX và FY để tạo nhãn cho mỗi đỉnh. $FX[i]$ thể hiện khả năng còn tạo lợi nhuận khi dùng máy i . $FY[j]$ thể hiện khả năng lợi nhuận khi dùng công nhân j . Luôn điều chỉnh nhãn sao cho chúng thỏa mãn điều kiện:

$$\forall i: 1 \leq i \leq M, 1 \leq j \leq N: FX[i] + FY[j] \geq C[i, j] \quad (1).$$

Máy i được giao cho thợ j chỉ khi bất đẳng thức (1) thành đẳng thức.

Khởi trị FX và FY như sau:

$\forall i: 1 \leq i \leq M, FX[i] = \max\{C[i, j] \mid \forall j: 1 \leq j \leq N\}$ và $FY[j] = 0 \quad \forall j: 1 \leq j \leq N$, (nghĩa là ban đầu máy i còn khả năng tạo lợi nhuận cao nhất và mọi thợ chưa nhận máy nên chưa tạo được lợi nhuận nào).

Nếu $FX[i] + FY[j] = C[i, j]$ (2) Đẳng thức (2) thể hiện giao máy i cho thợ j là hoàn toàn hợp lý. Việc giao nhận này được ghi nhận bằng cách cho cạnh (i, j) là đậm: $A[i] = j$ và $B[j] = i$.

Vậy bằng cách tạo nhãn ban đầu như trên chúng ta có thể ghép được một số máy với thợ tạo lợi nhuận nhiều nhất trên mỗi máy đó, tạo ra bộ ghép M ban đầu (nói chung chưa tối ưu).

Bước 2: Tối ưu hóa tổng lợi nhuận bằng cách thay dần các bộ ghép ngày càng tốt hơn:

*Duyệt <với mỗi đỉnh nhật i của X > (còn máy chưa giao cho thợ nào) thì:
Thực hiện vòng lặp*

- **Tìm đường mở** xen kẽ cạnh nhật và cạnh đậm bắt đầu từ đỉnh nhật của X , kết thúc tại đỉnh nhật của Y .
- *Nếu <không tìm được đường mở> thì <sửa nhãn> (nhằm mục đích giảm dần khả năng còn lại của máy, tăng dần khả năng tạo ra lợi nhuận của thợ và đồng thời tạo khả năng sinh đường mở);*
còn không thì <tăng số cạnh bộ ghép trên đường mở>;

Cho đến khi tìm thấy đường mở bắt đầu từ đỉnh i ;

Cho đến khi duyệt xong mọi đỉnh nhật trên X ;

Trong thuật toán trên phải thực hiện các thao tác: tìm đường mở, sửa nhãn, tăng số cạnh bộ ghép trên đường mở:

Tìm đường mở: Có thể tìm kiếm theo chiều sâu hoặc tìm kiếm theo chiều rộng. Yêu cầu đường mở xuất phát từ một đỉnh nhật của X , kết thúc bởi một đỉnh nhật của Y và có các cạnh nhật, đậm liên tiếp xen kẽ nhau.

Sửa nhãn: Thao tác này được thực hiện khi tìm đường mở không thành công, chỉ tạo ra dây chuyền xen kẽ cạnh đậm và nhật bắt đầu từ đỉnh nhật bên X kết thúc tại đỉnh đậm cùng bên X (còn gọi là đường pha). Trên các cạnh nối mỗi đỉnh $i \in X$ thuộc dây chuyền này tới các đỉnh $j \in Y$ chưa thuộc dây chuyền, tìm $\min = \text{Min}\{FX[i] + FY[j] - C[i,j]\}$ làm lượng sửa nhãn (ký hiệu là \min). Việc sửa nhãn như sau: Nhãn các đỉnh của X thuộc dây chuyền sẽ giảm đi một lượng là \min , nhãn các đỉnh của Y thuộc dây chuyền sẽ tăng thêm lượng \min để luôn bảo đảm $FX[i] + FY[j] - C[i,j] \geq 0$. Sau khi các đỉnh thuộc dây chuyền được sửa nhãn thì xuất hiện khả năng có thể tạo một đường mở kết thúc tại đỉnh nhật bên Y (vì sẽ xuất hiện những cạnh (i,j) mới mà $FX[i] + FY[j] = C[i,j]$).

Tăng bộ ghép: Khi tìm được đường mở, thực hiện đổi màu các cạnh của đường mở này, bắt đầu từ cạnh nhật cuối cùng của đường mở đổi ngược

dẫn về cạnh nhạt đầu tiên của đường mở. Việc thực hiện này bộ ghép tăng thêm một cạnh, đồng thời tổng trọng số trên các cạnh của bộ ghép cũng tăng thêm (tự chứng minh, dựa vào (1) và (2))

Lưu ý: Bài toán tìm bộ ghép có tổng trọng số nhỏ nhất được giải tương tự, chỉ khác biệt ở chỗ ban đầu phải đổi dấu $C[i,j]$ và cuối cùng đổi dấu tổng.

Hoặc giải theo cách sau:

- Khởi trị $C[i,j]=\infty$ nếu thợ j không có khả năng điều khiển máy i ,
- Nhãn ban đầu $\forall i \in X: FX[i]=\text{Min}\{C[i,j] \mid \forall j \in Y\}$ và $FY[j]=0 \forall j \in Y$.
- Lượng sửa nhãn $m = \text{Min}[C[i,j]-FX[i]-F[j]]$

Chương trình (tìm bộ ghép có tổng trọng số lớn nhất)

```
uses      crt;
const    fi = 'capghep2.in';
          fo = 'capghep2.out';
          mn = 100;
type     m1 = array[1..mn,1..mn] of integer;
          m2 = array[1..mn] of integer;
var       c  : m1; {mảng trọng số}
          a,b,      {2 mảng dùng xác nhận đỉnh đậm, đỉnh nhạt}
          fx,fy,    {2 mảng ghi nhãn các đỉnh}
          dx,dy,t  : m2; {3 mảng phục vụ tìm dây chuyển xen kẽ cạnh đậm, nhạt}
          m,n,      {m máy, n công nhân}
          lj : integer; {lưu lại đỉnh nhạt cuối cùng của đường mở}
          ok      : boolean; {cờ báo tìm được/không được một đường mở}
```

Procedure read_init;

```
var f : text;
    max,i,j : integer;
begin
  assign(f,fi); reset(f);
  readln(f,m,n); {đọc từ tệp input: m máy, n công nhân}
  for i:=1 to m do begin
    max := -maxint; {khởi trị lợi nhuận cao nhất của máy i}
    for j:=1 to n do begin {đọc ma trận trọng số từ tệp input}
      read(f,c[i,j]);
      if c[i,j]>max then max := c[i,j];
    end;
    readln(f);
    fx[i] := max; {Khởi trị nhãn cho máy i}
  end;
```

```

close(f);
for j:=1 to n do fy[j] := 0; {Khởi trị nhãn cho các công nhân}
end;
Procedure init1; {Khởi trị các mảng a và b: các đỉnh đều nhạt}
begin
  fillchar(a,sizeof(a),0);
  fillchar(b,sizeof(b),0);
end;
Procedure init2; {Khởi trị các mảng dx, dy, t, biến ok trước khi tìm đường mở}
begin
  fillchar(dx,sizeof(dx),0);
  fillchar(dy,sizeof(dy),0);
  fillchar(t,sizeof(t),0);
  ok := false;
end;
Procedure find(i: integer); {Tìm đường mở bắt đầu từ đỉnh nhạt i bên X}
var j : integer;
begin
  if ok then exit; {Tìm được đường mở thì thoát}
  dx[i] := 1; {Đánh dấu đỉnh i đã nạp vào dây chuyền tìm đường mở}
  for j:=1 to n do {Đề cử công nhân j ghép với máy i}
  if dy[j]=0 then {Điều kiện: công nhân j chưa nạp vào dây chuyền này}
  if (fx[i]+fy[j]=c[i,j]) then begin {và thỏa mãn đẳng thức (2)}
    t[j] := i; {xác nhận trên dây chuyền: trước đỉnh j là đỉnh i}
    if b[j]=0 then begin {Điều kiện kết thúc đường mở: đỉnh j là nhạt}
      lj := j; {Lưu lại đỉnh cuối của đường mở}
      ok := true; {xác nhận đã tìm thấy đường mở}
      exit; {Thoát đệ quy}
    end else begin {chưa kết thúc được dây chuyền thì}
      dy[j] := 1; {đánh dấu j đã nạp vào dây chuyền}
      find(b[j]); {đệ quy, kéo dài dây chuyền}
    end;
  end;
end;
end;
function find_min : integer; {Tìm lượng sửa nhãn các đỉnh trên dây chuyền}
var p,i,j : Integer;
begin
  p := maxint; {Khởi trị lượng sửa nhãn}
  for i:=1 to m do {Đề cử mọi đỉnh i trên dây chuyền}
  if dx[i]=1 then
  for j:=1 to n do {Đề cử mọi đỉnh j chưa thuộc dây chuyền}
  if dy[j]=0 then
    if fx[i]+fy[j]-c[i,j]<p then {tìm fx[i]+fy[j]-c[i,j] nhỏ nhất}
      p := fx[i]+fy[j]-c[i,j]; {ghi nhận vào p}

```

```

    find_min := p; {trả lại lượng sửa nhẵn}
end;
Procedure repair; {sửa nhẵn}
var i,j, min : integer;
begin
    min := find_min; {lượng sửa nhẵn}
    for i:=1 to m do {giảm khả năng còn lại của máy i trên dây chuyền}
        if dx[i]=1 then dec(fx[i],min);
    for j:=1 to n do {tăng khả năng của thợ trên dây chuyền}
        if dy[j]=1 then inc(fy[j],min);
    end;
Procedure inc_edge(j : integer); {Tăng cạnh bộ ghép trên đường mở kết
thức tại đỉnh nhật j bên Y}
var i,p : integer;
begin
    repeat
        i := t[j]; {i là đỉnh của cạnh đậm cuối cùng}
        p := a[i]; {p: lưu lại số hiệu đỉnh bên Y của cạnh đậm cuối cùng}
        a[i] := j; {đổi màu cạnh nhạt cuối cùng}
        b[j] := i;
        j := p; {chuyển lên cạnh nhạt ngay phía trên}
    until j=0; {Khi j=0 thì đã qua đỉnh i là đỉnh nhạt đầu tiên của đường mở}
end;
Procedure solution; {Thực hiện thuật toán tìm bộ ghép có tổng trọng số max}
var i : integer;
begin
    init1; {Khởi trị các mảng A và B}
    for i:=1 to m do
        if a[i]=0 then {Tìm thấy đỉnh nhạt bên X}
            repeat {Thực hiện vòng lặp cho đến khi tìm được đường mở}
                init2; {Khởi trị các mảng phục vụ tìm đường mở}
                find(i); {Tìm đường mở bắt đầu từ đỉnh nhạt i trên X}
                if not ok then
                    repair {Nếu chưa tìm thấy đường mở thì sửa nhẵn các đỉnh trên dây chuyền}
                else inc_edge(1j); {nếu có đường mở thì tăng cạnh bộ ghép}
            until ok; {Kết thúc lặp khi đã tìm thấy đường mở}
        end;
Procedure write_out;
var i,tong : integer; f : text;
begin
    tong := 0; {Khởi trị tổng trọng số trên bộ ghép}
    for i:=1 to m do {Đề cử duyệt các máy}
        if a[i]>0 then {máy đã được giao cho công nhân a[i]}
            tong := tong + c[i,a[i]]; {cộng thêm trọng số cạnh (i, A[i])}

```

```

assign(f,fo);  rewrite(f);
writeln(f,tong); {ghi tổng trọng số vào tệp output}
for i:=1 to m do {ghi phân công máy i cho thợ j là A[i]}
if a[i]>0 then writeln(f,i,' ',a[i]);
close(f);
end;
BEGIN
  read_init;
  solution;
  write_out;
END.

```

7. Luồng cực đại

a) Một vài khái niệm

Bài toán: Hãy tìm cách vận chuyển được nhiều hàng hoá nhất ra khỏi địa điểm s đến được địa điểm t trên mạng giao thông, cho biết điều kiện vận chuyển cho phép tối đa trên mỗi cung đường (gọi là sức cho phép thông qua tối đa của cung) trên mạng giao thông này.

Lượng hàng hóa vận chuyển được trên các cung đường của mạng giao thông đã cho tạo thành luồng (dương) trên mạng giao thông này. Điểm s gọi là đỉnh phát, điểm t gọi là điểm thu của mạng. Bài toán tìm cách vận chuyển được nhiều hàng hóa nhất từ điểm phát tới điểm thu gọi là bài toán tìm luồng cực đại trên mạng.

Một số khái niệm trong bài toán tìm luồng cực đại là:

Mạng. Mạng là bộ năm $G=(V, E, c, s, t)$ trong đó V và E lần lượt là tập đỉnh và tập cung của một đa đồ thị có hướng không khuyên. Đỉnh phát s và đỉnh thu t thuộc V . Hàm c xác định trên tập cung E : $c(e)$ là số không âm cho biết sức thông qua tối đa trên cung e thuộc E .

Luồng dương. Luồng dương là một hàm f xác định trên tập E thỏa mãn:

- $0 \leq f(e) \leq c(e) \quad \forall \text{ cung } e \in E$ (nghĩa là giá trị luồng trên mỗi cung không vượt quá sức thông qua tối đa của mỗi cung ấy)
- Với mọi đỉnh v khác s và t thì tổng luồng dương trên các cung đi vào v bằng tổng luồng dương trên các cung đi ra khỏi v (trong bài toán thực tế

nêu trên, điều này có nghĩa là hàng hóa vận chuyển không đọng lại tại các điểm trung gian).

Giá trị luồng. Giá trị luồng f trên mạng G là tổng giá trị luồng trên các cung đi ra khỏi đỉnh phát. *Luồng cực đại* là luồng có giá trị lớn nhất.

Lát cắt s-t. Lát cắt s - t trên mạng $G(V, E, c, s, t)$ là một phân hoạch V thành hai tập đỉnh X và Y mà $s \in X$ và $t \in Y$. Lưu lượng qua lát cắt này là tổng trọng số trên các cung (u,v) mà $u \in X, v \in Y$, kí hiệu lưu lượng qua lát cắt (X,Y) là $c(X,Y)$, tổng luồng qua lát cắt này là $f(X,Y)$. Lát cắt s - t có lưu lượng nhỏ nhất gọi là *lát cắt hẹp nhất*.

Định lý Ford-Fulkerson: “Giá trị luồng cực đại trên mạng $G(V, E, c, s, t)$ bằng lưu lượng thông qua lát cắt hẹp nhất”.

Khi luồng đạt giá trị cực đại thì $f(X,Y)=c(X,Y)$. Tất nhiên đẳng thức này cũng đúng với lát cắt hẹp nhất.

b) Thuật toán tìm luồng cực đại.

Quá trình chứng minh định lí Ford Fulkerson, cho một thuật toán tìm luồng cực đại trên mạng $G=(V, E, c, s, t)$ gọi là thuật toán Ford-Fulkerson: Các bước cụ thể như sau:

Bước 1- Tạo một luồng ban đầu (thông thường là $F(e)=0, \forall e \in E$).

Bước 2- Sau khi đã có một luồng trên G , xây dựng thêm đồ thị $G_f = (V, E_f)$ như sau:

+ Nếu $e=(i, j) \in E$ mà $F(e)<A(e)$ thì trên G_f có cung $e_1=(i, j)$ với trọng số bằng $A(e)-F(e)$ gọi là cung thuận. Cung thuận của G_f thể hiện lượng còn có thể tăng thêm luồng trên cung (i, j) của G . Nếu $F(e)=A(e)$ thì trên G_f không có cung (i,j) .

+ Nếu $F(e)>0$ thì trên G_f có cung $e_2 = (j, i)$ với trọng số $F(e)$ gọi là cung ngược. Cung ngược trên G_f thể hiện lượng còn có thể giảm luồng trên cung $e=(i,j)$ của G .

Bước 3- Tìm một đường đi trên G_f từ đỉnh phát s đến đỉnh thu t . Đường đi này gọi là đường tăng luồng. Nếu không tìm được đường tăng luồng thì về bước 6

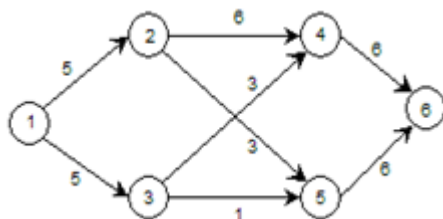
Bước 4. Tìm trọng số nhỏ nhất trên các cung thuộc đường tăng luồng. Số này gọi là lượng sửa nhân, kí hiệu là m .

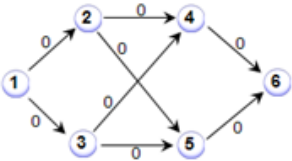
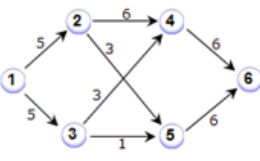
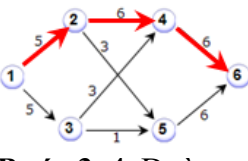
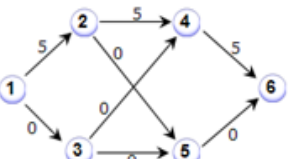
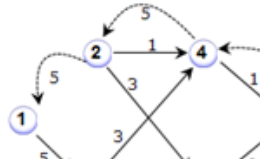
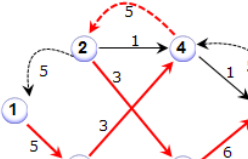
Bước 5- Tăng luồng trên đồ thị G bằng cách: trên G_f , dọc theo đường tăng luồng gặp cung thuận (i,j) thì tăng thêm luồng trên cung (i,j) của G một lượng m , nếu gặp cung ngược (j,i) trên G_f thì giảm luồng trên cung (i,j) của G một lượng m .

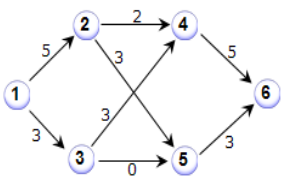
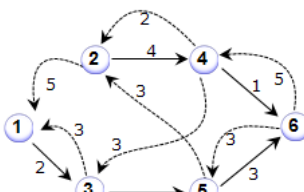
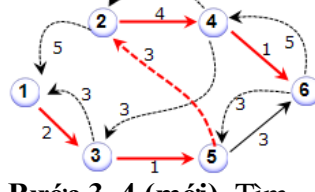
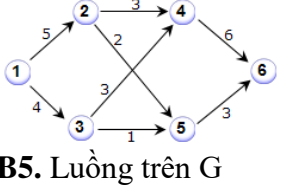
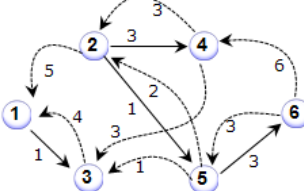
Bước 6- Thoát.

Vậy thuật toán Ford Fulkerson còn có thể phát biểu tóm tắt là: “Luồng cực đại trên G chỉ đạt được khi không còn đường tăng luồng trên G_f ”.

Ví dụ. Xét mạng nêu ở Hình bên
Ban đầu khởi trị luồng $F(e) = 0 \forall e \in E$.



 <p>Bước 1. Luồng ban đầu trên mạng G</p>	 <p>Bước 2. Xây dựng G_f</p>	 <p>Bước 3, 4. Đường tăng luồng trên G_f: 1,2,4,6, lượng sửa nhân $m=5$</p>
 <p>Bước 5. Luồng trên G</p>	 <p>Bước 6 về bước 2 mới. Xây dựng G_f</p>	 <p>Bước 3, 4 (mới). Tìm đường tăng luồng trên G_f: 1,3,4,2,5,6, $m=3$</p>

 <p>Bước 5 (mới). Luồng trên G</p>	 <p>Bước 6 về bước 2 mới. Xây dựng G_f</p>	 <p>Bước 3, 4 (mới). Tìm đường tăng luồng trên G_f: 1,3,5,2,4,6, $m=1$</p>
 <p>B5. Luồng trên G</p>	 <p>G_f không còn đường tăng luồng.</p>	<p>Kết luận: Giá trị luồng cực đại trên mạng G là $W=f((1,2))+f((1,3))=5+4=9$</p>

Nhận thấy giá trị luồng cực đại đúng bằng khả năng thông qua của lát cắt hẹp nhất. Lát cắt hẹp nhất trong trường hợp này là phân hoạch (X, Y) trong đó X là tập đỉnh $\{1,3\}$, Y là tập đỉnh $V \setminus X$.

Trong thực tế lập trình tìm luồng cực đại, thường chỉ sử dụng một đồ thị G , không xây dựng trực tiếp đồ thị G_f , do đó thường sử dụng một số kỹ năng sau :

a) Định nghĩa nhãn của một đỉnh: Nhãn của đỉnh i trên G là một cặp số: Số thứ nhất gọi là nhãn lưu vết đỉnh trước của đỉnh i , *nhãn thứ hai* gọi là nhãn báo khả năng còn có thể thay đổi thông lượng luồng qua đỉnh i . Nếu nhãn của đỉnh i là cặp số $(+j, v)$ thì giá trị luồng trên cung (j, i) không được phép tăng thêm một lượng vượt quá v . Nhãn của đỉnh i là cặp số $(-j, v)$ thì giá trị của luồng trên cung (i, j) không được phép giảm nhiều hơn v .

b) Đường tăng luồng : Là dãy liên tiếp các cung thuộc E , nối từ đỉnh s tới đỉnh t . Những cung này trên G gồm 2 loại: cung thuận và cung ngược. Trên đường tăng luồng, cung e là cung thuận nếu có chiều như chiều từ s tới t và bảo đảm điều kiện $F(e) < A(e)$; cung e là cung ngược nếu có chiều ngược từ t tới s và bảo đảm điều kiện $F(e) > 0$.

c) Các thao tác : Trong quá trình thực hiện thuật toán, ta sử dụng 3 thao tác : Khởi tri, sửa nhãn, điều chỉnh luồng (tăng luồng).

Khởi tri : Tạo một luồng ban đầu trên mạng (có thể chọn luồng tầm thường $F: F(e)=0 \forall e$. Giá trị của luồng là $W=0$.

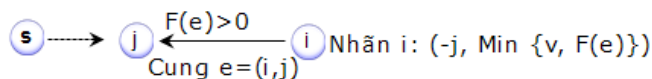
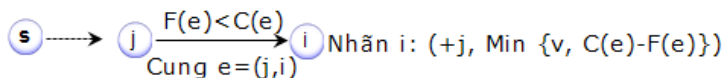
Tất cả các đỉnh chưa có nhãn, và đánh dấu là chưa xét.

Gán nhãn $s(+s, \infty)$. Nạp đỉnh s vào stack.

Sửa nhãn: Quá trình sửa nhãn cho các đỉnh được tiến hành đồng thời với quá trình tìm các *đường tăng luồng*. Lấy đỉnh j ở đỉnh của stack để thêm cung liên thuộc với j vào đường tăng luồng đang xây dựng, đồng thời sửa nhãn cho các đỉnh i chưa đánh dấu, kề với j . Giả sử nhãn đỉnh j đang có nhãn là $(+k, v)$ hoặc $(-k, v)$.

+ Nếu $e=(j,i) \in E$ là cung thuận mà $F(e) < C(e)$ thì nhãn mới của i là $(+j, v_0)$ với $v_0 = \min\{v, C(e)-F(e)\}$.

+ Nếu $e=(i,j) \in E$ là cung ngược mà $F(e) > 0$ thì nhãn mới của i là $(-j, v_0)$ với $v_0 = \min\{v, F(e)\}$



Sửa xong nhãn thì nạp đỉnh i vào stack.

Cuối cùng, sau khi không tìm được đỉnh i nào nữa, ta đánh dấu đỉnh j là đã được dùng để sửa nhãn cho các đỉnh i kề với j .

Điều chỉnh luồng: Sau khi đỉnh thu t được sửa nhãn, t có nhãn thứ hai là v thì bắt đầu điều chỉnh luồng với lượng điều chỉnh là v bằng cách lần ngược đường tăng luồng từ đỉnh thu t về đỉnh phát s :


```

+ Gán  $j := t$ )
+ Vòng lặp
     $i := j$ ;
     $j :=$  nhãn thứ nhất của  $i$ ;
    Nếu  $j > 0$  thì luồng  $F[j, i]$  tăng thêm một lượng  $v$ ;
    Nếu  $j < 0$  thì  $F[i, -j]$  giảm một lượng  $v$  ;
     $j := \text{Abs}(j)$ ;
    Lặp cho đến khi  $j = s$  ;

```

d) Dàn bài tìm luồng cực đại

Khởi trị luồng 0;

Repeat

 Khởi trị nhãn và mảng đánh dấu;

 While *<Stack khác rỗng>* do begin

 Lấy j ở đỉnh Stack;

 If *<còn đỉnh i chưa thăm, kề với j >* then *<Sửa nhãn i >*

 If *< t đã được đánh dấu>* then begin

 Điều_chỉnh_luồng ;

 Break; {Thoát khỏi vòng While}

 end;

 end;

Until *<không thể đánh dấu đỉnh t >* ;

Chú ý:

Trong lần cuối cùng tìm đường tăng luồng (không thành công), các đỉnh đã được đánh dấu tạo thành tập X , các đỉnh còn lại chưa được đánh dấu tạo thành tập Y thì lát cắt (X, Y) là lát cắt hẹp nhất.

Chương trình tìm luồng cực đại

```
uses crt;
```

```

const max      = 101;
      fi       = 'luong.inp';
      fo       = 'luong.out';
type mang      = array[0..max,0..max] of integer;
      mang2    = array[0..max] of integer;
      rec      = record
                      v,tr:integer;
                  end;
      mang3    = array[0..max] of rec;
var   a,F      : mang;
      nh       : mang3;
      stack,dx : mang2;
      n,top,s,t,m : integer;
Procedure doc_dulieu;
var i,j, k      :integer;
      f        : text;
begin
  assign(f,fi); reset(f);
  readln(f,n,m, s,t); {n đỉnh, m cung, đỉnh phát s và đỉnh thu t}
  for k:=1 to m do {cung (i,j) có thông lượng tối đa là a[i,j]}
    readln(f,i,j, a[i,j]);
  close(f);
end;
Procedure nap(p:integer); { nạp đỉnh p vào stack}
begin
  inc(top);
  stack[top]:=p;
end;
function lay:integer; {Lấy đỉnh ở đỉnh stack}
begin
  lay:=stack[top];
  dec(top);
end;
function min(a,b:integer):integer;
begin
  if a<b then min:=a  else min:=b;
end;

```

Procedure init; *{Khởi trị mảng đánh dấu dx, ngăn xếp stack, nhãn đỉnh s}*

var i : integer;

begin

for i:=1 to n do begin

nh[i].v:=maxint; nh[i].tr:=0;

end;

fillchar(dx,sizeof(dx),0);

top:=1; stack[top]:=s;

nh[s].v:=maxint; nh[s].tr:=s;

end;

Procedure suanhan(j:integer);

var i:integer;

begin

for i:=1 to n do *{xét các cung liên thuộc với đỉnh j}*

if dx[i]=0 then begin

if (a[j,i]<>0)and(f[j,i]<a[j,i]) then begin *{(j,i) là cung thuận}*

dx[i]:=1;

nh[i].tr:=+j;

nh[i].v:= min(nh[j].v,a[j,i]-f[j,i]);

nap(i);

end else

if (a[i,j]<>0) and (f[i,j]>0) then begin *{(i,j) là cung ngược}*

dx[i]:=1;

nh[i].tr:=-j;

nh[i].v:=min(nh[j].v,f[i,j]);

nap(i);

end

end;

end;

Procedure tangluong;

var i, j:integer;

begin

j:=t;

repeat

i := j;

j := nh[i].tr;

if j>0 then inc(F[j,i],nh[t].v)

else dec(F[i,-j],nh[t].v);

```

        j:=abs(j);
until j=s;
end;
Procedure ghi_ketqua;
var i,j,gt : integer;
    g      : text;
begin
    gt := 0;
    for i:=1 to n do
        if i<>s then
            if f[s,i]>0 then gt:=gt+f[s,i];
            assign(g,fo); rewrite(g);
            writeln(g,gt); {Giá trị luồng cực đại}
            for i:=1 to n do {Hiện luồng trên các cung}
                for j:=1 to n do
                    if f[i,j]>0 then
                        writeln(g,i,' ',j,' ',f[i,j]);
                    close(g);
                end;
            end;
Procedure timluong; {Thuật toán tìm luồng cực đại}
var v:integer;
begin
    repeat
        init;
        while top>0 do begin
            v:=lay;
            suanhan(v);
            if dx[t]<>0 then begin
                tangluong;
                break; {lệnh thoát while}
            end;
        end;
    until dx[t]=0;
end;
BEGIN
    doc_dulieu;
    timluong;
    ghi_ketqua;

```

END.

Khi mạng có số cung tương đối lớn, nên tổ chức danh sách liên thuộc để quản lý các cung:

```
const maxN = 1000;
      maxM = 1000;
      maxC = 10000;
      fi   = 'bvtp.in';
      fo   = 'bvtp.out';

type t_edge = record {cấu trúc một cung}
  x,y : integer; {cung ra khỏi x, vào y}
  c,f : integer; {trọng số cung (x,y) và luồng trên cung này}
end;

t_queue = record {cấu trúc hàng đợi dùng tìm kiếm đường tăng luồng}
  items:array[1..maxN] of integer; {các phần tử trong hàng đợi}
  front:integer; {biến đầu hàng đợi}
  rear :integer; {biến cuối hàng đợi}
end;

var e :array[-maxM..maxM] of t_edge; {quản lý các cung}
    link:array[-maxM..maxM] of integer; {liên kết với cung tiếp theo}
    head:array[1..maxN] of integer; {head[i] chỉ số cung đầu ra khỏi i}
    trace: array[1..maxN] of integer; {vết đường đi khi tìm kiếm}
    n,m,s,t : integer;
    flow_value : integer; {giá trị luồng trên mạng}
    queue : t_queue; {hàng đợi}
Procedure enter; {nhập dữ liệu và tổ chức danh sách liên thuộc}
var i,u,v,w : integer;
    f : text;
begin
  assign(f,fi); reset(f);
  readln(f,n,m,s,t); {số đỉnh, số cung, đỉnh phát và đỉnh thu}
  fillchar(head[1], n*sizeof(head[1]), 0);
  for i:=1 to m do begin {đọc các cung và tổ chức danh sách liên thuộc}
    readln(f,u,v,w);
    with e[i] do begin {tổ chức cung thứ i là cung (u, v) có trọng số w}
      x := u;
      y := v;
      c := w;
      link[i] := head[u];
      head[u] := i; {cung đầu tiên ra khỏi đỉnh u là cung thứ i đọc được}
    end;
    with e[-i] do begin {tổ chức thêm cung ngược (v,u) có trọng số 0, qui ước
là cung thứ -i}
      x := v;
```

```

    y := u;
    c := 0;
    link[-i] := head[v];
    head[v] := -i;
end;
end;
end;
Procedure init_flow; {Khởi trị luồng 0}
var i : integer;
begin
    for i:=-m to m do e[i].f := 0;
    flow_value := 0;
end;
function find_path: boolean; {tìm đường tăng luồng}
var u,v,i : integer;
begin
    find_path := true;
    fillchar(trace[1],n*sizeof(trace[1]),0);
    trace[s] := 1; {đánh dấu vết của đỉnh phát}
    with queue do begin
        items[1] := s; {phần tử thứ nhất trong hàng đợi là đỉnh phát s}
        front := 1; {Khởi trị biến đầu hàng đợi}
        rear := 1; {Khởi trị biến cuối hàng đợi}
        repeat {thực hiện tìm kiếm cho đến khi hàng đợi rỗng}
            u := items[front]; {lấy đỉnh u ở đầu hàng đợi}
            inc(front);
            i := head[u]; {i là số hiệu cung đầu tiên ra khỏi u}
            while i<>0 do begin {duyệt các cung ra khỏi u}
                v := e[i].y; {v kề với u, chưa thăm, luồng trên cung (u,v) chưa bão hòa}
                if (trace[v]=0) and (e[i].f<e[i].c) then begin
                    trace[v] := i; {vết: theo cung thứ i đi vào v}
                    if v=t then exit; {gặp đỉnh thu, tìm xong đường tăng luồng}
                    inc(rear);
                    items[rear] := v; {nạp v vào hàng đợi}
                end;
                i := link[i]; {móc nối sang cung tiếp theo (cũng xuất phát từ u)}
            end;
        until front>rear; {kết thúc lặp khi hàng đợi rỗng}
        find_path := false; {cờ báo không tìm thấy đường tăng luồng}
    end;
end;
end;
Procedure augment_flow; {điều chỉnh luồng}
var delta, v, i : integer;
begin
    v := t; {lần ngược từ đỉnh thu t}

```

```

{tìm lượng sửa nhân là min của sức còn chứa được trên các cung của đường tăng luồng}
delta := maxC;
repeat
    i := trace[v];
    if e[i].c-e[i].f<delta then
        delta :=e[i].c-e[i].f;
    v := e[i].x;
until v=s;
{điều chỉnh luồng, }
v := t;
repeat
    i := trace[v];
    inc(e[i].f, delta); {tăng luồng trên cung thuận}
    dec(e[-i].f, delta); {giảm luồng trên cung ngược}
    v := e[i].x;
until v=s;
inc(flow_value,delta); {cộng thêm lượng vừa tăng vào giá trị luồng}
end;
Procedure print_result;
var i : integer; g : text;
begin
    assign(g,fo); rewrite(g);
    writeln(g,flow_value); {ghi giá trị luồng}
    for i:=1 to m do begin
        with e[i] do begin {ghi luồng trên các cung của mạng}
            if f>0 then writeln(g,x,' ', y,' ', c,' ',f);
        end;
    end;
    close(g);
end;
BEGIN
    enter;
    init_flow;
    while find_path do augment_flow; {Thuật toán Ford-Fulkerson}
    print_result;
END.

```

8. Bài tập minh họa

Bài 1. Mạng vòng

Một số mạng cục bộ được tổ chức theo mô hình phân cấp bộ phận: máy chủ quản lý một máy trạm hoặc không quản lý máy trạm nào, thông tin điều khiển được phép truyền một chiều từ máy chủ tới máy trạm (nếu có).

Các máy trạm này có thể là máy chủ của một số trạm cục bộ có thể bao gồm nhiều trạm cục bộ nhỏ hơn.

Để tận dụng tối đa tài nguyên thông tin, người ta quyết định hợp nhất tất cả các mạng cục bộ thành một mạng thông tin thống nhất, trong đó các máy đều bình quyền với cấu hình mạng vòng tròn (token ring). Mỗi máy có và chỉ có quyền truyền thông tin điều khiển tới đúng một máy khác. Việc cắt bớt đường truyền cũ được thực hiện dễ dàng coi như là không kể tới nhưng việc thiết lập thêm đường truyền mới đòi hỏi phải cài đặt lại hệ thống ở máy truyền.

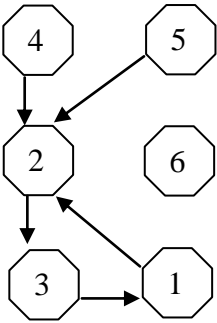
Yêu cầu: hãy chỉ ra cách chỉnh lý mạng hiện có tạo thành mạng thông tin thống nhất theo yêu cầu trên mà số đường truyền mới cần thiết lập là ít nhất.

Dữ liệu: Vào từ file văn bản NET.INP: Dòng đầu chứa một số nguyên N là số máy trong mạng ($1 < N \leq 4000$). Các dòng sau, mỗi dòng chứa hai số nguyên i, j cho biết thông tin được truyền từ máy i sang máy j. Dòng cuối cùng chứa 2 số 0.

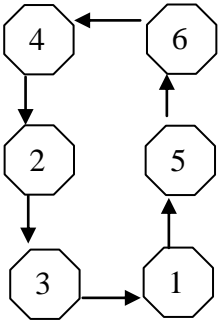
Kết quả: ghi ra file văn bản NET.OUT: Dòng đầu tiên ghi số nguyên K là số máy cần phải cài đặt lại hệ thống. K dòng sau: mỗi dòng chứa ba số nguyên P, Q, R cho biết phải bỏ đường truyền từ máy P sang máy Q và thiết lập đường truyền từ máy P sang máy R. Nếu P không có đường truyền sang máy khác thì Q=0.

Ví dụ:

NET . INP	NET . OUT
6	3
1 2	1 2 5
2 3	5 2 6
3 1	6 0 4
4 2	
5 2	
0 0	



Hệ thống cũ



Hệ thống mới

Gợi ý. Bằng tìm kiếm chiều sâu hoặc chiều rộng, tìm các cung liên tiếp nhau tạo thành một dây chuyền hoặc chu trình. Đặc biệt: đỉnh không có cung vào và cung ra (đỉnh độc lập) coi như một dây chuyền có đỉnh cuối của dây chuyền là đỉnh đó và có cung đi tới đỉnh 0 (qui ước trong đề). Mỗi dây chuyền bắt đầu từ đỉnh không có cung vào (gọi là đỉnh xuất phát), kết thúc tại một đỉnh có cung đi tới một đỉnh đã thăm hoặc tới đỉnh 0. Trong hình trên có 3 dây chuyền là: $4 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 2$, $5 \rightarrow 2$ và $6 \rightarrow 0$.

Trước hết tìm các dây chuyền có đỉnh xuất phát không có cung vào, sau đó tìm đến các chu trình (điểm cuối có cung đi tới đỉnh đầu) và các dây chuyền còn lại (đỉnh xuất phát có cung vào).

Cần bỏ cung đi ra từ điểm cuối mỗi dây chuyền hoặc chu trình và thay nó bằng cung nối điểm cuối này với điểm xuất phát của dây chuyền hoặc chu trình tiếp theo.

Chương trình.

```
const fi    = 'NET.IN';
      fo    = 'NET.OUT';
      maxn  = 4001;

type mang1 = array[1..maxn] of integer;
      mang2 = array[1..maxn] of byte;

var  f, g   : text;
      N     : integer;
      next  : mang1; {next[u] là đỉnh mà u đi tới}
      bacvao: mang1; {cho số cung đi vào mỗi đỉnh}
      visit : mang2; {đánh dấu các đỉnh đã thăm khi tìm dây chuyền, chu trình}
      count : integer; {đếm số dây chuyền và chu trình}
      dau   : mang1; {quản lý điểm đầu các dây chuyền và chu trình}
      cuoi  : mang1; {quản lý điểm cuối các dây chuyền và chu trình}
```

Procedure doc_dulieu;

```
var i, u, v: integer;
begin
  assign(f, fi); reset(f);
  read(f, n); {số đỉnh - số máy}
  for i:=1 to n do next[i]:=0; {khởi trị mảng next}
  for i:=1 to n do bacvao[i]:=0; {khởi trị mảng bacvao}
  while not seekeof(f) do begin
    read(f, u, v); {đọc cung (u,v)}
    if (u=0) or (v=0) then break;
    next[u]:=v; {xác nhận v là đỉnh có cung đi tới từ u}
```

```

    inc(bacvao[v]); {tăng số cung đi vào v}
end;
close(f);
end;
Procedure DFS(xp:integer); {tìm chu trình hoặc dây chuyền bắt đầu từ đỉnh xp}
var u,v: integer;
begin
    dau[count]:=xp; {xác nhận chu trình/dây chuyền thứ count có điểm đầu là xp}
    u:=xp;
    repeat
        visit[u]:=1; {đã thăm u}
        v:=u; {giữ lại u để phục vụ mảng cuoi}
        u:=next[u];
    until (u=0) or (visit[u]=1);
    cuoi[count]:=v; {đỉnh cuối của chu trình/dây chuyền thứ count}
end;
Procedure find; {tìm các chu trình và dây chuyền}
var i: integer;
begin
    for i:=1 to n do visit[i]:=0; {khởi trị mảng visit}
    count:=0; {số chu trình, dây chuyền}
    for i:=1 to n do
        if bacvao[i]=0 then begin {không có cung vào, nên là đỉnh xuất phát của
dây chuyền}
            inc(count);
            DFS(i);
        end;
    for i:=1 to n do {tìm các chu trình, dây chuyền còn lại}
        if visit[i]=0 then begin {đỉnh i chưa được thăm, có bậc vào dương}
            inc(count);
            DFS(i);
        end;
    end;
end;
Procedure ghi_ketqua;
var i: integer;
begin
    assign(g,fo); rewrite(g);
    dau[count+1]:=dau[1]; {chuẩn bị cho nối điểm cuối của dây chuyền/chu trình
thứ count với điểm đầu của dây chuyền/chu trình thứ 1}
    writeln(g,count) {ghi số máy phải cài đặt lại hệ thống}
    for i:=1 to count do {ghi các cài đặt lại theo qui cách đề bài yêu cầu}
        writeln(g,cuoi[i],#32,next[cuoi[i]],#32,dau[i+1]);
    close(g);
end;
BEGIN

```

```
doc_dulieu;  
find;  
ghi_ketqua;  
END.
```

Bài 2. Khám phá mê cung

Một mê cung gồm N phòng và một số hành lang nối các phòng ($1 \leq N \leq 200$). Giữa hai phòng bất kì có không quá một hành lang nối chúng, các hành lang có thể đi được theo hai chiều.

Một robot khám phá mê cung, xuất phát từ một phòng và đi thăm tất cả các hành lang sao cho mỗi hành lang được đi qua đúng một lần, rồi trở về phòng xuất phát. Khi qua mỗi hành lang, robot sẽ nạp một nguồn năng lượng là c (c có thể âm hoặc dương). Robot xuất phát với năng lượng bằng 0. Trong quá trình di chuyển, robot có thể bị ngừng hoạt động nếu sau khi đi hết hành lang nào đó chỉ còn năng lượng âm.

Yêu cầu tìm một hành trình an toàn cho robot thoả mãn các yêu cầu nêu trên.

Dữ liệu vào từ file MECUNG.IN chứa các thông tin sau :

Dòng đầu ghi hai số N, M

M dòng sau, dòng thứ i mô tả hành lang i gồm 3 số u, v, c với ý nghĩa có hành lang giữa hai phòng u và v với giá trị năng lượng là c ($|c| \leq 10000$).

Kết quả ghi ra file MECUNG.OUT chứa các thông tin sau :

Dòng đầu ghi số 1 hoặc 0 tùy theo có nghiệm hay không.

Nếu có nghiệm thì $M+1$ dòng tiếp theo ghi lần lượt các số hiệu phòng mà robot đi qua : từ phòng xuất phát, ..., rồi trở về phòng xuất phát.

Gợi ý.

Thực hiện các bước sau:

Kiểm tra đồ thị có liên thông hay không. Nếu không liên thông thì ghi vào file output số 0, dừng chương trình.

Kiểm tra có chu trình Euler hay không. Nếu không có thì ghi vào file output số 0, dừng chương trình.

Nếu có thì :

+ Tìm chu trình Euler. Giả sử tìm được chu trình là $\{s[1], s[2], \dots, s[m], s[1]\}$

+ Xây dựng mảng một chiều Total với ý nghĩa : Total[i] là tổng các trọng số của các cạnh mà rô bắt đi qua dọc theo chu trình tìm được từ $s[1]$ đến $s[i]$.

+Tìm điểm i_0 mà Total[i_0] nhận giá trị nhỏ nhất. Điểm i_0 chính là phòng bắt đầu xuất phát của rô bắt. Hành trình cần tìm sẽ là $\{s[i_0], \dots, s[m], s[1], s[2], \dots, s[i_0]\}$

+ Hiện kết quả vào file.

Chương trình

```
const fi    = 'mecung.in';
      fo    = 'mecung.out';
      max   = 100;
      maxc  = 20000;
      maxe  = 5000;
type  tarr  = array[1.. max, 1.. max] of integer;
var    a      : tarr;
      b      : ^tarr;
      v,deg   : array[1..max] of integer;

      stack  : array[1.. maxe] of integer;
      path   : array[1.. maxe] of integer;
      d      : array[1.. maxe] of longint;
      sol     : boolean;
      n, m,
      sv,top, count      : integer;
      f                  : text;
      total               : longint;
procedure read_input;
var i, u, v: integer;
begin
  assign(f, fi); reset(f);
  readln(f, n, m);
  for u:= 1 to n do
    for v:= 1 to n do a[u, v]:= -maxc;
  fillchar(deg,sizeof(deg),0);
  total  := 0;
  for i:= 1 to m do
    begin
      readln(f, u, v, a[u, v]);
      a[v, u] := a[u, v];
```

```

        inc(deg[u]);
        inc(deg[v]);
        total := total + a[u,v];
    end;
close(f);
sol := false; { bài toán vô nghiệm}
{nếu có đỉnh cô lập hoặc đỉnh bậc lẻ thì không tồn tại chu trình Euler}
for i:=1 to n do
    if (deg[i]=0) or (deg[i] mod 2 =1) then exit;
    {điều kiện cần để bài toán có nghiệm là total>=0}
    if total<0 then exit;
    sol := true; { bài toán có nghiệm}
end;
procedure DFS(i : integer); {tìm kiếm theo chiều sâu một thành phần liên
thông chứa đỉnh i}
var j : integer;
begin
    for j:=1 to n do
        if v[j]=0 then
            if a[i,j]=1 then
                begin
                    v[j] := sv;
                    DFS(j);
                end;
            end;
    end;
end;
procedure stplt; {tìm số thành phần liên thông}
var s : integer;
begin
    fillchar(v,sizeof(v),0);
    sv := 0;
    for s:=1 to n do
        if v[s]=0 then {đỉnh s bắt đầu một vùng liên thông mới}
            begin
                inc(sv);
                v[s] := sv;
                DFS(s);
            end;
    end;
end;
procedure init; {khởi trị}
var i : integer;
begin
    new(b);
    b^ := a;
    top := 1;
    stack[1] := 1;
    count := 0;

```

```

end;
procedure euler; {tìm chu trình Euler}
var u, v : integer;
begin
    repeat
        u:= stack[top];
        for v:= 1 to n do
            if a[u, v] <> -maxc then
                begin
                    inc(top);
                    stack[top] := v;
                    a[u, v] := -maxc;
                    a[v, u] := -maxc;
                    break;
                end;
            if u = stack[top] then
                begin
                    inc(count);
                    path[count]:= u;
                    dec(top);
                end;
        until top <= 0;
end;
procedure solve;
var i, p : integer;
    vmin : longint;
begin
    if not sol then {vô nghiệm}
        begin
            assign(f, fo); rewrite(f);
            writeln(f, 0);
            close(f);
            exit;
        end
    else {có chu trình Euler}
        begin
            d[1]:= 0; {d[i] là tổng năng lượng khi robot đi từ phòng 1 đến phòng i}
            for i:= 2 to count do
                d[i]:= d[i - 1] + b^[path[i - 1], path[i]];

            {tìm điểm i có d[i] nhỏ nhất}
            vmin := maxlongint;
            for i:= 1 to count do
                if vmin > d[i] then
                    begin
                        vmin := d[i];

```

```

        p      := i;
    end;
    {ghi file output. Robot đi từ phòng i (tìm đo ở trên, theo chu trình Euler tìm
đo }
    assign(f, fo); rewrite(f);
    writeln(f, 1);
    for i:= p to count do writeln(f, path[i]);
    for i:= 2 to p do writeln(f, path[i]);
    close(f);
end;
end;
BEGIN
    read_input; {đọc dữ liệu vào từ file input}
    init;
    stplt; {tìm số thành phần liên thông}
    if sv>1 then {không liên thông, bài toán vô nghiệm}
    begin
        assign(f, fo); rewrite(f);
        writeln(f, 0);
        close(f);
        exit;
    end
    else
    begin
        euler; {tìm một chu trình Euler}
        solve; {tìm ra thành phố xuất phát, hiện kết quả vào file output}
    end;
END.

```

Bài 3. Bin – Các thùng nước

Có N thùng nước được đánh số từ 1 đến N, giữa 2 thùng bất kỳ đều có một ống nối có một van có thể khóa hoặc mở. Ở trạng thái ban đầu tất cả các van đều đóng.

Bạn được cho một số yêu cầu, trong đó mỗi yêu cầu có 2 dạng:

Dạng X Y 1 có ý nghĩa là bạn cần mở van nối giữa 2 thùng X và Y.

Dạng X Y 2 có ý nghĩa là bạn cần cho biết với trạng thái các van đang mở / khóa như hiện tại thì 2 thùng X và Y có thuộc cùng một nhóm bình thông nhau hay không? Hai thùng được coi là thuộc cùng một nhóm bình thông nhau nếu nước từ bình này có thể chảy đến được bình kia qua một số ống có van đang mở.

Input: BIN.INP. Dòng đầu tiên ghi một số nguyên dương P là số yêu cầu. Trong P dòng tiếp theo, mỗi dòng ghi ba số nguyên dương X, Y, Z với ý nghĩa có yêu cầu loại Z với 2 thùng X và Y.

Output: BIN.OUT. Với mỗi yêu cầu dạng X Y 2 (với $Z = 2$) bạn cần ghi ra số 0 hoặc 1 trên 1 dòng tùy thuộc 2 thùng X và Y không thuộc hoặc thuộc cùng một nhóm bình. Giới hạn: $1 \leq N \leq 10000$, $1 \leq P \leq 50000$. Thời gian: 1 s/test. Bộ nhớ: 1 MB

Ví dụ:

BIN . INP	BIN . OUT
9	0
1 2 2	0
1 2 1	1
3 7 2	0
2 3 1	1
1 3 2	0
2 4 2	
1 4 1	
3 4 2	
1 7 2	

Gợi ý. Đây là bài toán tìm vùng liên thông. Chương trình sau dùng phương pháp hợp nhất cây.

Chương trình

```

const fi      = 'BIN.IN0';
      fo      = 'BIN.OU0';
      N      = 10000;
type mang1    = array[1..N] of integer;
var   f, g    : text;
      father  : mang1;
Procedure MoVan(u,v: integer);
var i,j: integer;
begin
  i:=u; j:=v;
  while father[i]>0 do i:=father[i]; {tìm gốc của cây chứa u}
  while father[j]>0 do j:=father[j]; {Tìm gốc của cây chứa v}
  if i<>j then begin {Hai cây khác gốc}
    if father[i]>father[j] then begin {Cây chứa u ít nút hơn}
      father[j]:=father[j]+father[i]; {Xác nhận gốc cây hợp nhất}
      father[i]:=j; {Xác nhận gốc của cây chứa u là con của gốc cây chứa v}
    end
  end

```



```

        else begin {Cây chứa v ít nút hơn}
            father[i]:=father[i]+father[j]; {Xác nhận gốc cây hợp nhất}
            father[j]:=i; {Xác nhận gốc cây chứa v là con của gốc cây chứa u}
        end;
    end;
end;
Procedure Xuat(u,v: integer);
var i,j: integer;
begin
    i:=u; j:=v;
    while father[i]>0 do i:=father[i]; {Tìm gốc của cây chứa u}
    while father[j]>0 do j:=father[j]; {Tìm gốc của cây chứa v}
    if i=j then writeln(fo,1) {u và v cùng thuộc một cây}
    else writeln(fo,0); {u và v thuộc hai cây khác nhau}
end;
Procedure Main;
var P,i: longint;
    u,v,k: integer;
begin
    assign(f,fi); reset(f);
    assign(g,fo); rewrite(g);
    for i:=1 to N do father[i]:= -1; {Mỗi đỉnh là một cây (chỉ có 1 nút)}
    readln(f,P); {Đọc số lượng lệnh}
    for i:=1 to P do begin
        readln(f,u,v,k); {Đọc một lệnh}
        if k=1 then MoVan(u,v) {Xử lý lệnh loại 1}
        else Xuat(u,v); {Xử lý lệnh loại 2}
    end;
    close(f); close(g);
end;
BEGIN
    Main;
END.

```

Bài 4. Trong vườn trẻ

Trong vườn trẻ, có các nhóm trẻ đang chơi ở những sân chơi khác nhau cùng với cô giáo hướng dẫn (mỗi sân chơi chỉ cho một nhóm, mỗi nhóm chỉ có một cô giáo hướng dẫn). Mỗi sân chơi có lối đi nối một hay nhiều sân chơi khác. Một hay nhiều sân chơi có lối đi nối với sân chơi chính. Một số sân chơi không có học sinh. Có tổng cộng p lối đi.

Có đúng 26 sân chơi được đặt tên từ 'a' đến 'z', sân chơi chính tên là 'z' và theo qui định chung không có nhóm trẻ nào được chơi ở đó. Trên bảng

theo dõi ở sân chơi chính, sân nào hiện đang có 1 nhóm trẻ đang chơi được gán tên là chữ in hoa (ví dụ, sân chơi có tên là 'y' hiện đang có 1 nhóm trẻ đang chơi thì trên bảng theo dõi sẽ được gán tên là 'Y')

Khi có thông báo đến giờ nghỉ của vườn trẻ, các nhóm trẻ sẽ theo cô giáo hướng dẫn của mình đi nhanh về sân chính bằng các lối đi rồi mới được bố mẹ đón về - các cô giáo luôn tìm ra con đường ngắn nhất để dẫn các cháu đi (tất nhiên, đường đi mà các cô giáo chọn có thể đi ngang qua một số sân chơi khác). Giả sử tốc độ đi của các nhóm là như nhau và các lối đi đủ rộng để không có nhóm nào phải tạm dừng lại (tránh nhau) khi đi - Người ta cần biết sau khi phát thông báo thì nhóm trẻ hiện đang chơi ở sân chơi nào về tới sân chính trước nhất. Hãy viết chương trình giải quyết yêu cầu trên.

Dữ liệu vào từ file văn bản VUONTRE . IN:

Dòng đầu tiên chỉ một số nguyên dương là giá trị số p ($p \leq 10000$)

Từ dòng thứ hai đến dòng thứ $p+1$ mỗi dòng mô tả một lối đi gồm tên của hai sân chơi và độ dài của lối đi nối chúng, ba nội dung trên một dòng được viết cách nhau ít nhất một dấu cách (độ dài mỗi lối đi không vượt quá 1000)

Kết quả ghi ra file văn bản VUONTRE . OUT một dòng duy nhất gồm:

Một ký tự in hoa là tên của sân chơi có nhóm trẻ về tới sân chính sớm nhất

Tiếp theo là dấu cách và ghi độ dài đường đi ngắn nhất của nhóm trẻ đó.

Ví dụ:

VUONTRE . IN	VUONTRE . OUT
5	B 9
A d 6	
B d 3	
C e 9	
d z 8	
e z 3	
B d 1	

Gợi ý. Tìm đường đi ngắn nhất từ sân chính ‘z’ đến các sân có học sinh bằng thuật toán gán nhãn (trên đa đồ thị vô hướng). Chú ý trong các cạnh (lối đi) nối trực tiếp hai đỉnh (sân chơi) chỉ cần giữ lại cạnh có trọng số nhỏ nhất. Các đỉnh có nhãn được nạp vào hàng đợi. Mỗi lần cần chọn một đỉnh có nhãn nhỏ nhất trong các đỉnh chưa cố định nhãn để sửa nhãn cho các đỉnh kề có nhãn chưa cố định thì duyệt tìm trong hàng đợi. Do số phần tử trong hàng đợi không vượt quá 26 nên cũng không cần tổ chức hàng đợi ưu tiên (Heap).

Chương trình

```
const fi='VUONTRE.IN';
      fo='VUONTRE.OUT';
var   a: array[1..26,1..26] of integer;
      co: array[1..26] of byte;
      f, g: text;
      Q: array[1..26] of integer; // Hàng đợi
      qn: integer; // số phần tử còn trong hàng đợi
      kc: array[1..26] of longint; // độ dài các lối đi
      loai: array[1..26] of byte; // loại đỉnh trong thuật toán gán nhãn
procedure init; // Khởi trị hàng đợi
begin
    qn:=0;
end;
procedure put(u: integer); // Nạp đỉnh u vào hàng đợi
begin
    inc(qn);
    q[qn]:=u;
end;
function get: integer; // Lấy một đỉnh có nhãn nhỏ nhất ra khỏi hàng đợi
var u,i: integer;
begin
    u:=1; // Duyệt các đỉnh có trong hàng đợi để chọn đỉnh có nhãn nhỏ nhất
    for i:=2 to qn do
        if kc[q[i]]<kc[q[u]] then u:=i;
    get:=q[u]; // phần tử được lấy ra
    q[u]:=q[qn]; // phần tử cuối hàng đợi được thay vào chỗ của phần tử lấy ra
    dec(qn); // giảm số phần tử trong hàng đợi
end;
function empty: boolean; // Kiểm tra hàng đợi rỗng
begin
    empty:=(qn=0);
end;
```

```

function order(ch: char): integer; // tạo mã số thứ tự cho 26 sân chơi
begin
    case ch of
        'a'..'z': order:=ord(ch)-96;
        'A'..'Z': order:=ord(ch)-64;
    end;
end;

procedure read_input; // Đọc tệp input
var p,i: integer;
    ch1, ch2: char;
    L, i1, i2: integer;
begin
    fillchar(a,sizeof(a),0);
    fillchar(co,sizeof(co),0); // sân chơi có học sinh hay không
    assign(f,fi); reset(f);
    readln(f,p); // số lối đi
    for i:=1 to P do begin // đọc các lối đi
        repeat read(f,ch1) until ch1<>#32; // đọc sân chơi ở một đầu lối đi
        repeat read(f,ch2) until ch2<>#32; // đọc sân chơi ở đầu kia lối đi
        readln(f,L); // đọc độ dài lối đi
        i1:= order(ch1); // i1 là mã số sân chơi ch1
        i2:= order(ch2); // i2 là mã số sân chơi ch2
        if ch1 in ['A'..'Z'] then co[i1]:=1; // ch1 có học sinh không
        if ch2 in ['A'..'Z'] then co[i2]:=1; // ch2 có học sinh không
        if (a[i1,i2]=0) or (a[i1,i2]>L) then begin
            a[i1,i2]:=L; // lưu độ dài lối đi ngắn nhất giữa hai sân chơi i1 và i2
            a[i2,i1]:=L;
        end;
    end;
    close(f);
end;
```

```

procedure find; // Thuật toán gán nhãn tìm đường đi ngắn nhất
var i, u, v: integer;
begin
    init; // Khởi trị hàng đợi
    for i:=1 to 26 do Loai[i]:=0; // các đỉnh chưa thăm
    kc[26]:=0; // sân chính (là 'z') có nhãn là 0
    loai[26]:=1; // xác nhận sân chơi z nhãn
    put(26); // nạp sân chơi z vào hàng đợi
    repeat
        u:= get; // lấy đỉnh u có nhãn nhỏ nhất trong các đỉnh chưa cố định nhãn
        loai[u]:=2; // cố định nhãn đỉnh u
        for v:=1 to 26 do // duyệt các đỉnh kề với u
```

```

if a[u,v]>0 then begin //v kề với u
    if (loai[v]=1) then //v có nhãn chưa cố định
    if kc[v]>kc[u]+a[u,v] then //nhãn của v chưa tối ưu thì
    kc[v]:=kc[u]+a[u,v]; //sửa nhãn cho v
    if loai[v]=0 then begin //nếu v chưa thăm thì
        kc[v]:=kc[u]+a[u,v]; //gán nhãn cho v
        loai[v]:=1; //xác nhận v đã có nhãn
        put(v); // nạp v vào hàng đợi
    end;
end;
until empty;
end;

```

```

procedure write_out;
var i, min, li: integer;
begin
    min:=maxint;
    for i:=1 to 26 do //Tìm sân chơi có học sinh và có nhãn đường đi nhỏ nhất
    if (co[i]=1) and (kc[i]<min) then begin
        min:=kc[i];
        li:=i;
    end;
    assign(g,fo); rewrite(g);
    writeln(g,chr(64+li),#32,min);
    close(g);
end;
BEGIN
    read_input;
    find;
    write_out;
END.

```

Bài 5. Tô màu bản đồ

Bản đồ của một đất nước có dạng hình chữ nhật $M \times N$ ô. Đất nước này được chia làm nhiều vùng, một vùng là tập các ô có chung cạnh liên thông với nhau và được đánh cùng một số trên bản đồ.

Biết rằng số vùng không vượt quá 200. Ví dụ: Bản đồ có kích thước 3×4 và chia làm 4 vùng như hình vẽ bên:

1	1	9	9
2	1	9	2
2	1	9	2

Hãy tìm cách tô màu bản đồ sao cho mỗi vùng có một màu và hai vùng tiếp giáp nhau (hai vùng có ít nhất một cạnh chung) không được tô cùng màu. Tìm cách tô với số màu ít nhất.

Dữ liệu vào trong file BANDO.IN có dạng:

Dòng đầu là 2 số nguyên M, N ($M, N \leq 50$);

M dòng tiếp theo, mỗi dòng N số thể hiện bản đồ (các số thuộc kiểu integer).

Kết quả: ra file BANDO.OUT có dạng:

Dòng đầu ghi K là số màu tô ít nhất tìm được;

M dòng sau, mỗi dòng N số thể hiện cách tô màu bản đồ (dùng các màu từ 1 đến K)

Ví dụ:

BANDO.IN	BANDO.OUT
3 4	2
1 1 9 9	1 1 2 2
2 1 9 2	2 1 2 1
2 1 9 2	2 1 2 1

Gợi ý. Trước hết tìm các vùng liên thông. Sau đó xây dựng đồ thị có mỗi đỉnh là một vùng liên thông. Hai đỉnh có cạnh nối với nhau nếu hai vùng liên thông tương ứng kề nhau (có ô vuông vùng này chung cạnh với ô vuông của vùng kia). Sau đó thực hiện thuật toán tô màu.

Chương trình

```
const
    max      = 200;
    maxx     = 50;
    chon      = '9';
    fi        = 'BANDO.in' + chon;
    fo        = 'BANDO.ou' + chon;
    tx        : array[1..4] of shortint=(0,-1,0,1);
    ty        : array[1..4] of shortint=(-1,0,1,0);
var
    a         : array[1..maxx,1..maxx] of integer;
    d         : array[1..maxx,1..maxx] of byte;
    w         : array[1..max,1..max] of byte;
    ten,bac,back : array[1..max] of byte;
    mau,lmau    : array[1..max] of byte;
    m,n,sv      : integer;
    sm,lsm      : integer;
```

```
Procedure docf;
var f      :text;
```

```

    i,j :integer;
begin
    assign(f,fi); reset(f);
    readln(f,m,n); {Đọc kích thước bản đồ hình chữ nhật}
    for i:=1 to m do
        for j:=1 to n do read(f,a[i,j]); {Đọc các số trên bản đồ}
    close(f);
end;

Procedure loang(x,y:byte); {Dùng DFS tìm các vùng liên thông}
var k,u,v :byte;
begin
    d[x,y]:=sv;
    for k:=1 to 4 do begin
        u:=x+tx[k];v:=y+ty[k];
        if (u>0)and(v>0)and(u<=m)and(v<=n)then
            if (d[u,v]=0)and(a[u,v]=a[x,y])then loang(u,v);
    end;
end;

Procedure trao(var u,v:byte);
var coc :byte;
begin
    coc:=u;u:=v;v:=coc;
end;

Procedure init;
var i,j,k,l:byte;
    bb,bmax:shortint;
begin
    sv:=0;
    fillchar(d,sizeof(d),0); {đánh dấu các ô đã thăm, chưa thăm}
    fillchar(w,sizeof(w),0); {Khởi trị mảng trọng số các cạnh của đồ thị}
    for i:=1 to m do
        for j:=1 to n do
            if d[i,j]=0 then begin {Tìm và đánh số các vùng liên thông}
                inc(sv);
                loang(i,j);
            end;
            for k:=1 to sv do ten[k]:=k; {Khởi trị tên vùng, cũng là tên đỉnh}
            {Xây dựng cạnh của đồ thị: hai vùng liên thông kề nhau thì có cạnh nối 2 đỉnh}
            for i:=1 to m do
                for j:=1 to n-1 do
                    if d[i,j]<>d[i,j+1] then begin
                        w[d[i,j],d[i,j+1]]:=1;
                        w[d[i,j+1],d[i,j]]:=1;
                    end;
            for i:=1 to m-1 do
                for j:=1 to n do

```

```

if d[i,j]<>d[i+1,j] then begin
    w[d[i,j],d[i+1,j]]:=1;
    w[d[i+1,j],d[i,j]]:=1;
end;
{Xây dựng bậc các đỉnh của đồ thị}
fillchar(bac,sizeof(bac),0);
for i:=1 to sv-1 do
for j:=i+1 to sv do
if w[i,j]=1 then begin inc(bac[i]);inc(bac[j]);end;
{Xếp lại tên các đỉnh: đỉnh i có bậc lớn hơn thì ten[i] nhỏ hơn. Đây là một ý tưởng thực
hiện thuật toán tham: đỉnh có bậc cao được tô màu trước }
for i:=1 to sv-1 do
for j:=i+1 to sv do
if bac[ten[i]]<bac[ten[j]] then trao(ten[i],ten[j]);
{Cải tiến thêm ý tưởng thực hiện tham}
for k:=2 to sv-1 do begin
    bmax:=-1;
    for i:=k+1 to sv do begin
        bb:=0;
        for j:=1 to k-1 do inc(bb,w[ten[i],ten[j]]);
        if bb>bmax then begin bmax:=bb;l:=i;end;
    end;
    trao(ten[k],ten[l]);
end;
lsm:=5; {Khởi trị số màu tối đa}
sm:=1; {Khởi trị số màu cần dùng khi bắt đầu xây dựng một cách tô màu}
mau[1]:=1; {tô màu đỉnh 1 là màu 1}
end;

```

function check(i,j:byte):boolean; {Kiểm tra đỉnh thứ i có tên là ten[i] có thể gán cho màu j được hay không }

```

var k      :byte;
begin
    check:=false;
    for k:=1 to i-1 do {xét các đỉnh đã tô màu trước đỉnh thứ i}
        if (w[ten[k],ten[i]]=1)and(mau[k]=j) then exit;
    check:=true;
end;

```

Procedure toiuu; {ghi lại phương án tối ưu}

```

begin
    if sm<lsm then begin
        lsm:=sm;lmau:=mau;
    end;
end;

```

Procedure try(i:byte); {Duyệt tìm màu tô cho đỉnh thứ i}

```

var j      :byte;

```



```

begin
  if sm>=lsm then exit; {phương án hiện tại không tối ưu hơn cũ thì bỏ}
  for j:=1 to sm do {Tìm màu j trong các màu đã tô}
  if check(i,j) then begin {nếu tô được thì}
    mau[i]:=j; {tô đỉnh thứ i bằng màu j}
    if i<sv then try(i+1) {nếu chưa tô hết các đỉnh thì duyệt tiếp}
    else toiuu; {còn không thì so tìm nghiệm tối ưu}
  end;
  if sm<lsm-1 then begin {đ/ kiện: số màu không tối hơn phương án tối ưu}
    inc(sm); {màu mới}
    mau[i]:=sm; {tô đỉnh thứ i bằng màu sm}
    if i<sv then try(i+1)
    else toiuu;
    dec(sm); {quay lui}
  end;
end;
Procedure ghif; {Ghi kết quả vào tệp output}
var f      :text;
    i,j,k  :byte;
begin
  for k:=1 to sv do back[ten[k]]:=k; {đỉnh (vùng) có tên là ten[k]}
  assign(f,fo); rewrite(f);
  writeln(f,lsm);
  for i:=1 to m do begin
    for j:=1 to n do
      write(f,lmau[na[d[i,j]]],#32); {ghi màu lên ô vuông}
    writeln(f);
  end;
  close(f);
end;

BEGIN
  docf;
  init;
  if sv>1 then try(2)
  else begin lsm:=sm;lmau:=mau;end;
  ghif;
END.

```

Bài 6. Comnet (Đề thi chọn HSG giỏi 12/3/2003, Bài 1)

Tổng công ty MeKa có N máy tính được đánh số từ 1 đến N. Người ta đã lắp đặt được M kênh trực tiếp truyền tin hai chiều giữa một số cặp máy. Hai máy tính u và v trong Tổng công ty được coi là có thể truyền tin được cho nhau nếu chúng được nối với nhau bằng một kênh nối trực tiếp hoặc tồn

tại một dãy các máy tính $u=m_0, m_1, \dots, m_k=v$ sao cho giữa hai máy m_{i-1} và m_i bất kỳ ($i=1..k$) đều có ít nhất một kênh nối trực tiếp. Nhằm triển khai thực hiện chủ trương cải cách hành chính theo tinh thần chỉ thị 58 của Bộ chính trị, Ban giám đốc đã cho triển khai dự án hoàn thiện mạng máy tính để nâng cao hiệu quả hoạt động của Tổng công ty.

Yêu cầu: Hãy giúp Ban giám đốc xác định số lượng ít nhất kênh trực tiếp truyền tin cần lắp đặt thêm, sao cho hai máy tính bất kỳ có thể truyền tin cho nhau.

Dữ liệu: vào từ file văn bản COMNET.INP, trong đó:

- Dòng đầu tiên chứa hai số nguyên dương N, M ($0 < N \leq 5000; M \leq 200000$);
- M dòng tiếp theo chứa thông tin về M kênh truyền tin đã được lắp đặt: Mỗi dòng chứa hai số nguyên dương xác định hai máy tính được nối bởi kênh trực tiếp truyền tin đã lắp đặt.

Kết quả: ghi ra file văn bản COMNET.OUT, trong đó:

- Dòng đầu tiên ghi số k là số lượng kênh truyền tin cần lắp đặt thêm
- K dòng tiếp theo mô tả thông tin về K kênh tìm được: Mỗi dòng chứa hai số nguyên xác định hai máy tính cần lắp đặt thêm kênh nối giữa chúng.

Trong trường hợp có nhiều hơn một cách nối chỉ cần đưa ra một cách.

Trong file **Dữ liệu vào** và trong file kết quả các số trên một dòng cách nhau ít nhất một dấu cách.

Ví dụ

COMNET . INP		COMNET . OUT	
8	6	2	
1	2	1	5
3	1	1	7
3	4		
4	1		
5	6		
7	8		

Gợi ý. Đây là bài toán xét đồ thị vô hướng G có N đỉnh, cần bổ sung số cạnh ít nhất để đồ thị liên thông. Trước hết tìm số vùng liên thông. Số cạnh ít nhất cần bổ sung bằng số vùng liên thông trừ đi 1.

Chương trình.

```
uses crt;
const max = 5000;
      fi = 'comnet.in';
      fo = 'comnet.out';
var  n,m : longint;
      w : array[1..max] of longint;
      sv : integer;
function find_root(x : integer) : longint;
var i : longint;
begin
  i:=x;
  while w[i]>0 do i:=w[i];
  find_root:=i;
end;
Procedure change(x : longint; y: longint);
var i : longint;
begin
  for i:=1 to n do
    if w[i]= x then w[i]:= y;
  end;
Procedure union(x,y:longint);
var temp:longint;
begin
  temp := w[x]+ w[y];
  if w[x] > w[y] then begin
    w[x] := y;
    change(x,y);
    w[y] := temp;
  end
  else begin
    w[y] := x;
    change(y,x);
    w[x] := temp;
  end;
end;
Procedure creatnet;
var f : text;
      i,x,y,r1,r2: longint;
begin
  assign(f,fi); reset(f);
```

```

readln(f,n,m);
for i:=1 to n do w[i]:=-1;
for i:=1 to m do begin
    readln(f,x,y);
    r1 := find_root(x);
    r2 := find_root(y);
    if r1<>r2 then union(r1,r2);
end;
close(f);
end;
Procedure write_out;
var g : text;
    i,j,k : longint;
begin
    assign(g,fo); rewrite(g);
    sv := 0;
    for i:=1 to n do
        if w[i]<0 then inc(sv); {đếm số vùng liên thông}
    writeln(g,sv-1); {số cạnh nối thêm}
    sv := 0;
    for i:=1 to n do begin
        if w[i]<0 then begin
            if sv=0 then begin
                k:=i; inc(sv) {k là gốc của vùng liên thông thứ nhất}
            end else {i là gốc của các vùng liên thông còn lại}
                writeln(g,k, ' ',i);
        end;
    end;
    close(g);
end;
BEGIN
    creatnet;
    write_out;
END.

```

Tạm gọi đỉnh đầu tiên nạp vào một vùng liên thông là “ngọn” của vùng. Có thể lập trình đơn giản hơn như sau:

```

Const fi = 'COMNET.INP';
    Fo = 'COMNET.OUT';
    maxN = 5000;
var f,g : text;
    N : integer;
    LT : array[1..maxN] of integer;
Procedure doc_dulieu;
var M,i : longint;
    u,v : integer;
begin

```

```

assign(f,fi); reset(f);
readln(f,N,M);
fillchar(LT,sizeof(LT),0);
for i:=1 to M do begin
    readln(f,u,v);
    while LT[u]<>0 do u:=LT[u]; {tìm ngọn của vùng chứa u đọc được}
    while LT[v]<>0 do v:=LT[v]; {tìm ngọn của vùng chứa v đọc được}
    if u<>v then LT[v]:=u; {hợp nhất hai vùng}
end;
close(f);
end;
Procedure in_ketqua;
var S: integer;
    i,j: integer;
begin
    S:=0; {Khởi trị số vùng liên thông}
    for i:=1 to N do
        if LT[i]=0 then inc(S); {gặp ngọn của một vùng thì tăng số vùng}
        assign(g,fo); rewrite(g);
        writeln(g,S-1); {số cạnh cần nối thêm}
        if S>1 then begin
            i:=0;
            repeat inc(i) until LT[i]=0; {ngọn của vùng thứ nhất}
            for j:=i+1 to N do {tìm ngọn của các vùng khác}
                if LT[j]=0 then writeln(g,i,' ',j); {cạnh được nối thêm}
            end;
            close(g);
        end;
    end;
BEGIN
    doc_dulieu;
    in_ketqua;
END.

```

Bài 7. HIGHWAY - Hệ thống đường cao tốc

Hệ thống đường cao tốc hiện tại ở thành phố A mới đảm bảo đi lại giữa một số nút giao thông trọng điểm và còn nhiều nút giao thông trọng điểm chưa có đường cao tốc qua nó. Để giải toả tình trạng ách tắc giao thông trong thành phố, chính quyền thành phố quyết định phát triển hệ thống đường cao tốc của thành phố sao cho có thể đi lại giữa hai nút giao thông trọng điểm bất kỳ. Có N nút giao thông trọng điểm, được đánh số từ 1 đến N. Nút giao thông i được cho bởi toạ độ (x_i, y_i) trong hệ toạ độ Đề các. Mỗi tuyến đường cao tốc nối hai nút giao thông trọng điểm. Tất cả các tuyến

đường hiện có cũng như sẽ được phát triển được xây dựng theo đường thẳng nối chúng, vì thế độ dài của mỗi tuyến đường chính là khoảng cách giữa hai điểm tương ứng với hai nút giao thông trọng điểm. Tất cả các tuyến đường cao tốc là hai chiều. Các tuyến đường có thể cắt nhau nhưng người sử dụng phương tiện giao thông chỉ được đổi tuyến đi ở các nút giao thông là đầu mút của các tuyến đường.

Chính quyền thành phố muốn tìm cách xây dựng bổ sung một số tuyến đường cao tốc nối các nút giao thông trọng điểm với chi phí nhỏ nhất đảm bảo sự đi lại giữa mọi nút giao thông trọng điểm. Chi phí xây dựng tỷ lệ thuận với độ dài của tuyến đường, vì vậy bạn có thể tính chi phí xây dựng các tuyến đường bổ sung như là tổng độ dài của các tuyến đường cần xây dựng.

Dữ liệu: Vào từ file văn bản `HIGHWAY.INP`:

Dòng đầu tiên chứa số N ($N \leq 750$);

Dòng thứ i trong số N dòng tiếp theo chứa toạ độ (x_i, y_i) của nút giao thông trọng điểm i ;

Dòng tiếp theo chứa M là số tuyến đường cao tốc hiện có ($0 \leq M \leq 1000$);

Dòng thứ j trong số M dòng tiếp theo chứa hai chỉ số của hai nút giao thông là đầu mút của tuyến đường j .

Kết quả: Ghi ra file `HIGHWAY.OUT`

Dòng đầu tiên chứa số K là số lượng tuyến đường cần xây dựng bổ sung;

Dòng thứ i trong số K dòng tiếp theo chứa hai chỉ số của hai nút giao thông là đầu mút của tuyến đường cần xây dựng bổ sung thứ i .

Ví dụ:

<code>HIGHWAY.INP</code>	<code>HIGHWAY.OUT</code>
9	5
1 5	1 6
0 0	3 7
3 2	4 9
4 5	5 7

5	1	8	3
0	4		
5	2		
1	2		
5	3		
4			
1	3		
9	7		
1	2		
2	3		

Gợi ý. Đây là bài toán tìm cây khung ngắn nhất trên đồ thị vô hướng. Chỉ lưu ý một điều là: khi hai đỉnh cùng thuộc một vùng liên thông thì cạnh nối chúng coi là bằng 0 (nghĩa là không cần bỏ sung cạnh mới nối chúng). Chương trình sau đây giải bài toán bằng thuật toán Prim (kết nạp dần các đỉnh vào cây khung ngắn nhất) sử dụng kỹ thuật gán nhãn. Để sửa nhãn được nhanh, các đỉnh chưa thăm được nạp vào một mảng coi như một “hàng đợi”. Mỗi lần lấy một đỉnh có nhãn nhỏ nhất trong “hàng đợi” cần phải duyệt toàn bộ hàng đợi, đó là hạn chế của chương trình dưới đây. Để cải tiến có thể tổ chức Heap để công việc sửa nhãn thực sự nhanh hơn.

Chương trình.

```

const
  fi='HIGHWAY.IN';
  fo='HIGHWAY.OUT';
  maxN=750;
var
  f, g : text;
  n    : longint;
  x,y  : array[1..maxN] of real;
  m    : longint;
  lt   : array[1..maxN] of longint; {quản lý các vùng liên thông}
  Q    : array[1..maxN] of longint; {coi như “hàng đợi”}
  top  : longint; {số phần tử trong “hàng đợi”}
  kc   : array[1..maxN] of real; {quản lý nhãn tổng độ dài các cạnh đã
  nạp vào cây khung, gọi là nhãn khoảng cách}
  tr   : array[1..maxN] of longint; {mảng vết và đánh dấu}
Procedure init; {Khởi trị “hàng đợi”}
begin
  top:=0;
end;
Procedure put(u: longint); {nạp đỉnh u vào “hàng đợi”}
begin

```

```

    inc(top);
    q[top]:=u;
end;
function get: longint; {Lấy khỏi “hàng đợi” đỉnh có nhãn nhỏ nhất}
var u,i: longint;
begin
    u:=1;
    for i:=2 to top do
        if kc[q[i]]<kc[q[u]] then u:=i;
    get:=q[u];
    q[u]:=q[top];
    dec(top);
end;
function empty: boolean; {Hàm kiểm tra “hàng đợi” rỗng}
begin
    empty:=(top=0);
end;
function khoangcach(u,v: longint): real; {Tính khoảng cách 2 đỉnh u, v}
var i, j: longint;
begin
    i:=u;
    while lt[i]<>0 do i:=lt[i]; {tìm “gốc” của vùng liên thông chứa đỉnh i}
    j:=v;
    while lt[j]<>0 do j:=lt[j]; {tìm “gốc” của vùng liên thông chứa đỉnh j}
    if i=j then {u và v cùng thuộc một vùng liên thông}
        khoangcach:=0.0 {thì khoảng cách u và v bằng 0}
    else {u và v không cùng thuộc một vùng liên thông thì khoảng cách là độ dài đoạn thẳng nối u và v}
        khoangcach:=sqrt (sqr (x[u]-x[v])+sqr (y[u]-y[v]));
end;
Procedure doc_dulieu;
var k,u,v,i,j: longint;
begin
    assign(f,fi); reset(f);
    read(f,N); {đọc số đỉnh}
    for k:=1 to N do read(f,x[k],y[k]); {tọa độ đỉnh k}
    read(f,M); {đọc các cạnh đã có}
    fillchar(lt,sizeof(lt),0); {khởi trị mảng theo dõi các vùng liên thông}
    for k:=1 to M do begin
        read(f,u,v); {đọc một cạnh đã có}
        i:=u;
        while lt[i]<>0 do i:=lt[i]; {tìm “gốc” của vùng liên thông chứa u}
        j:=v;
        while lt[j]<>0 do j:=lt[j]; {tìm “gốc” của vùng liên thông chứa v}
        if i<>j then lt[i]:= j; {Hai vùng liên thông khác nhau thì hợp nhất}
    end;
end;

```



```

end;
close(f);
end;
Procedure Prim; {Thuật toán Prim xây dựng cây khung ngắn nhất}
var u,v: longint;
    l: real;
begin
    init; {Khởi trị “hàng đợi” chứa các đỉnh có nhãn nhưng nhãn còn tự do}
    for u:=1 to n do tr[u]:=0; {mảng vết, cũng là mảng cho biết đỉnh thuộc loại
    nào. Nếu tr[u]=0 thì u là đỉnh chưa được thăm, tr[u]<0: đỉnh u đã thăm và có nhãn nhưng
    nhãn chưa cố định, tr[u]>0 đỉnh u đã có nhãn cố định (ngoài ra trong trường hợp này
    tr[u] còn là đỉnh được nạp vào cây khung ngắn nhất ngay trước khi nạp u)}
    tr[1]:=-(n+1); {Khởi trị vết của đỉnh 1}
    put(1); {nạp đỉnh 1 vào “hàng đợi”}
    repeat
        u:=get; {lấy ra khỏi “hàng đợi” một đỉnh đã thăm, có nhãn tự do nhỏ nhất}
        tr[u]:= -tr[u]; {cố định nhãn của u, lưu vết của u}
        for v:=1 to n do begin {duyệt các đỉnh v}
            l:=khoangcach(u,v); {tính khoảng cách giữa u và v}
            if (tr[v]<0)and(kc[v]>l)then begin {v có nhãn tự do chưa tối ưu}
                kc[v]:=l; {sửa lại nhãn khoảng cách của v}
                tr[v]:=-u; {xác nhận v có nhãn tự do}
            end;
            if tr[v]=0 then begin {nếu v chưa thăm}
                put(v); {thì nạp v vào “hàng đợi q”}
                tr[v]:=-u; {xác nhận đã thăm v}
                kc[v]:=l; {xác nhận nhãn khoảng cách của v}
            end;
        end;
    until empty; {cho đến khi các đỉnh đều được thăm}
end;
Procedure in_ketqua;
var sol,i: longint;
begin
    sol:=0; {khởi trị tổng độ dài các đường mới cần bổ sung}
    for i:=2 to n do {cộng dồn các độ dài các đường mới cần bổ sung}
        if khoangcach(i,tr[i])>0 then inc(sol);
    assign(g,fo); rewrite(g);
    writeln(g,Sol); {ghi kết quả: tổng độ dài các đường mới cần bổ sung}
    for i:=2 to n do
        if khoangcach(i,tr[i])>0 then
            writeln(g,i,' ',tr[i]); {ghi một con đường mới}
    close(g);
end;

```

```
BEGIN
    doc_dulieu;
    Prim;
    in_ketqua;
END.
```

Bài 8. Olymnet - Nối mạng máy tính

Để chuẩn bị cho cuộc thi đồng đội trong kỳ thi Olympic tin học sinh viên toàn quốc, Ban tổ chức giải quyết định thực hiện việc nối mạng N máy tính (được đánh số từ 1 đến N) để có thể thực hiện việc chấm bài theo yêu cầu của các đội dự giải. Một máy tính ở một trại nào đó được gọi là đã được hoà mạng nếu như nó được nối trực tiếp với máy chủ của Ban tổ chức (máy được đánh số 1 trong số N máy tính nói trên) hoặc được nối với một máy đã được hoà mạng của một trại khác. Ta gọi cách nối mạng là việc thực hiện các kênh nối giữa một số cặp máy sao cho mỗi máy đều được hoà mạng. Do nhiều lý do khác nhau, chỉ có một số máy có thể nối trực tiếp với máy chủ và cũng chỉ có một số máy tính có thể nối trực tiếp được với nhau. Biết chi phí thực hiện các kênh nối này, Ban tổ chức giải cần chọn ra hai cách nối mạng với tổng chi phí thực hiện là tiết kiệm nhất.

Yêu cầu: Xác định các chi phí của hai cách nối mạng với tổng chi phí thực hiện là tiết kiệm nhất.

Dữ liệu: Vào từ file văn bản OLYMNET.INP: Dòng đầu tiên chứa hai số nguyên N, M được cách nhau bởi dấu cách trong đó N ($3 \leq N \leq 1000$) là số lượng máy tính nối mạng, M là số kênh nối có thể thực hiện được giữa một số cặp máy trong số chúng ($N \leq M \leq 10000$). Mỗi dòng trong số M dòng tiếp theo chứa ba số nguyên dương a_i, b_i, c_i được ghi cách nhau bởi dấu cách, trong đó c_i cho biết chi phí để thực hiện kênh nối máy a_i với máy b_i , $c_i \leq 30000$ ($i=1,2,...,M$). Giả thiết dữ liệu đảm bảo luôn có cách nối mạng.

Kết quả: Ghi ra file văn bản OLYMNET.OUT hai số S_1, S_2 (cách nhau bởi dấu cách) là hai chi phí của hai cách nối mạng tiết kiệm nhất, trong đó $S_1 \leq S_2$, $S_1 = S_2$ khi và chỉ khi có ít nhất hai cách nối mạng có chung chi phí nhỏ nhất.

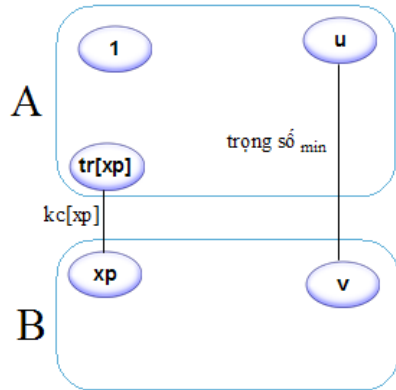
Ví dụ:

OLYMNET.INP	OLYMNET.OUT
5 8	110 121
1 3 75	
3 4 51	
2 4 19	
3 2 95	
2 5 42	
5 4 31	
1 2 9	
3 5 66	

Gợi ý.

Đây là bài toán tìm cây khung ngắn nhất trên đồ thị vô hướng. Đề bài yêu cầu tìm 2 cây khung ngắn nhất (có chứa máy số 1). Sau khi tìm được cây khung thứ nhất, để tìm cây khung thứ hai chúng ta thay một cạnh của cây khung thứ nhất bởi một cạnh ngắn nhất chưa thuộc cây khung mà không tạo ra chu kì trong mạng. Cụ thể: Lần lượt chọn một đỉnh bất kỳ khác đỉnh 1

là gốc cây mới (gọi là xp). Phân chia tập đỉnh thành 2 tập liên thông theo các cạnh của cây khung thứ nhất: Tập A gồm các đỉnh đi về 1 không qua đỉnh xp. Tập B là tập đỉnh còn lại (chứa cả đỉnh xp). Tìm cạnh có trọng số nhỏ nhất trong các cạnh (u,v) với u thuộc A, v thuộc B mà (u,v) không trùng với cạnh (xp, tr[xp]), sau đó thay cạnh (xp, tr[xp]) bởi cạnh (u,v) này sẽ được cây khung thứ hai.



Chương trình.

```

const fi      = 'OLIMNET.IN';
      fo      = 'OLIMNET.OUT';
      maxN    = 1001;
      maxM    = 20001;
type  arr1    = array[1..maxM] of integer;
      arr2    = array[1..maxN] of integer;
var   f, g    : text;
      N,M     : integer;
      tro     : arr2;

```

```

    kel, ke2      : ^arr1;
    Q             : arr2;
    top           : integer;
    kc, kcl       : arr2;
    trl           : arr2;
    min1, min2    : longint;
    goc1          : arr2;
Procedure init;
begin
    top:=0;
end;
Procedure put(u: integer);
begin
    inc(top);
    q[top]:=u;
end;
function get: integer; {Lấy ra khỏi “hàng đợi” đỉnh có nhãn nhỏ nhất}
var u,i: integer;
begin
    u:=1;
    for i:=2 to top do
        if kc[q[i]]<kc[q[u]] then u:=i;
    get:=q[u];
    q[u]:=q[top];
    dec(top);
end;
function empty: boolean;
begin
    empty:=(top=0);
end;
Procedure doc_dulieu;
var i, u, v, L: integer;
    tam: array[1..maxN] of integer;
begin
    assign(f,fi); reset(f);
    readln(f,N,M); {số đỉnh, số cạnh}
    {Tạo danh sách kề theo kiểu Forward Star}
    for i:=1 to N do tro[i]:=0;
    for i:=1 to M do begin
        readln(f, u,v,L);
        inc(tro[u]);
        inc(tro[v]);
    end;
    close(f);
    v:=1;
    for i:=1 to N do begin

```

```

    u:=tro[i];
    tro[i]:=v;
    v:=v+u;
end;
tro[N+1]:=v;
move(tro,tam,sizeof(tro));
reset(f);
readln(f,N,M);
for i:=1 to M do begin
    readln(f,u,v,l);
    ke1^[tam[u]]:=v; ke2^[tam[u]]:=L;
    ke1^[tam[v]]:=u; ke2^[tam[v]]:=L;
    inc(tam[u]);
    inc(tam[v]);
end;
close(f);
end;

```

Procedure ck1; {Xây dựng cây khung có tổng trọng số trên các cạnh nhỏ nhất}

var i, u, v, ts: integer;

begin

 init; {Khởi trị “hàng đợi” rỗng}

 for i:=1 to N do trl[i]:=0; {Khởi trị mảng vết}

 put(1); {Nạp máy 1 vào “hàng đợi”}

 trl[1]:=-(N+1); {vết của máy 1}

 kc[1]:=0; {Nhãn khoảng cách của máy 1}

 repeat {Xây dựng theo Prim}

 u:=get; {Lấy u là đỉnh có nhãn khoảng cách nhỏ nhất ra khỏi “hàng đợi”}

 trl[u]:= -trl[u]; {xác nhận nhãn của u đã cố định}

 for i:=tro[u] to tro[u+1]-1 do begin {tìm các đỉnh kề với u}

 v:=ke1^[i]; {v kề với u}

 ts:=ke2^[i]; {độ dài cạnh (u,v)}

 if (trl[v]<0) and (kc[v]>ts) then begin {điều kiện : nhãn của v chưa cố định và chưa tối ưu}

 trl[v]:=-u; {Xác nhận vết của v: trước v là u, và chưa cố định nhãn của v}

 kc[v]:=ts; {xác nhận nhãn mới của v}

 end;

 if trl[v]=0 then begin {v là đỉnh chưa thăm thì nạp v vào “hàng đợi”}

 put(v); {nạp v vào “hàng đợi”}

 trl[v]:=-u; {vết của v}

 kc[v]:=ts; {nhãn của v}

 end;

 end;

until empty; {xây dựng xong cây khung khi “hàng đợi” rỗng}

min1:=0; {Khởi trị tổng trọng số các cạnh trên cây khung ngắn nhất}

```

for i:=2 to N do
    min1:=min1+kc[i]; {Tính tổng trọng số các cạnh trên cây khung ngắn nhất}
    move(kc, kcl, sizeof(kc)); {sao chép kc sang kcl phục vụ tìm cây khung 2}
end;
Procedure ck2(xp: integer);
var i,j,u,v,lmoi,ts: integer;
begin
    {Phân chia tập đỉnh thành hai tập A và B. Tập A gồm các đỉnh i mà đường đi ra khỏi i
    về đỉnh 1 không qua đỉnh xuất phát (goc1[i]=1), tập B là tập còn lại (goc1[i]=0)}
    fillchar(goc1, sizeof(goc1), 0);
    for i:=1 to N do begin
        j:=i;
        while (j<>xp) and (j<>N+1) do j:=tr1[j];
        if j=N+1 then goc1[i]:=1;
    end;
    {Tính min2 là tổng trọng số các cạnh của cây khung 2}
    lmoi:=maxint;
    {Tìm đỉnh u thuộc tập A}
    for u:=1 to N do
        if goc1[u]=1 then {đường từ đỉnh u về gốc 1 không qua đỉnh xp}
        {Tìm đỉnh v kề với u mà v thuộc tập B}
        for i:=tro[u] to tro[u+1]-1 do begin {xét các đỉnh v kề với u}
            v:=kel^[i]; {v: đỉnh kề với đỉnh u}
            ts:=ke2^[i]; {trọng số cạnh (u,v)}
            if (goc1[v]=0) then {đường từ v về gốc 1 đi qua đỉnh xp}
            if ((u<>tr1[xp]) or (v<>xp)) then {cạnh (u,v) khác cạnh (xp, tr[xp])}
            if (ts<lmoi) then lmoi:=ts; {lưu lại độ dài cạnh (u,v) ngắn nhất}
        end;
    if min2>min1-kcl[xp]+lmoi then
        min2:=min1-kcl[xp]+lmoi; {xác nhận tối ưu của min2}
end;
Procedure xaydung;
var i: integer;
begin
    ck1; {Xây dựng cây khung thứ nhất}
    min2:=maxlongint;
    for i:=2 to N do ck2(i); {Tìm tổng các cạnh của cây khung thứ hai}
end;
Procedure in_ketqua;
begin
    assign(g, fo); rewrite(g);
    writeln(g, min1, ' ', min2);
    close(g);
end;
Procedure capphat;

```

```

begin
    new(ke1);    new(ke2);
end;
BEGIN
    capphat;
    doc_dulieu;
    xaydung;
    in_ketqua;
END.

```

Bài 9. Kiểm tra dữ liệu

Giáo sư địa chất học Smith trong một chuyến khảo sát nghiên cứu đã thu thập được N hòn đá để phân tích ($1 \leq N \leq 1000$). Các hòn đá được đánh số từ 1 tới N . Vấn đề đầu tiên mà giáo sư quan tâm là mối quan hệ về trọng lượng giữa chúng với nhau. Thật không may là xe chở dụng cụ đo đặc bị tai nạn giao thông và các thiết bị nghiên cứu đều hỏng hết. Không chịu bó tay, bằng các vật liệu hiện có, giáo sư Smith đã thiết kế một chiếc cân bàn, tuy thô sơ nhưng cũng giúp được ông khả dĩ có thể so sánh trọng lượng của 2 hòn đá bất kỳ. Các hòn đá khá nặng, nên mỗi lần cân, ở mỗi bên chỉ đặt được đúng một hòn đá. Ở sườn núi, nơi giáo sư làm việc gió thổi khá mạnh và làm sai lệch kết quả đo đạc. Khi hai hòn đá có trọng lượng lệch nhau nhiều, chiếc cân sẽ cho biết chính xác hòn nào nặng hơn, còn khi trọng lượng gần giống nhau hoặc giống nhau - kết quả không đáng tin cậy. Hòn đá a có thể nặng hơn hòn b , nhưng cân có thể cho kết quả là b nặng hơn a . Không có lần cân nào cho kết quả hai hòn đá có cùng trọng lượng. Ta gọi mỗi lần cân là một lần đo. Giáo sư ghi lại kết quả đo dưới dạng cặp số (i, j) , cho biết hòn i nặng hơn hòn j . Không có số liệu dạng (i, i) . Sau lần đo thứ m ($1 \leq m \leq 10000$), cân bị gãy, không thể sửa chữa được. Sau khi trở về phòng thí nghiệm, giáo sư rà soát lại các số liệu đo đạc, loại bỏ những số liệu không đáng tin cậy, dựa trên cơ sở là nếu hòn đá a_1 nặng hơn hòn đá a_2 và hòn đá a_2 nặng hơn hòn đá a_3 thì a_1 sẽ nặng hơn a_3 . Số liệu (i, j) gọi là không đáng tin cậy, nếu tồn tại $k > 0$, sao cho có dãy $H = \{(j, b_1), (b_1, b_2), \dots, (b_{k-1}, b_k), (b_k, i)\}$.



Yêu cầu: Cho n , m và các số đo (i, j) . Hãy xác định số lượng các số đo không đáng tin cậy.

Dữ liệu vào: Tập DATA.INP gồm nhiều bộ dữ liệu, mỗi bộ dữ liệu gồm: Dòng thứ nhất chứa số nguyên n . Dòng thứ 2 chứa số nguyên m . M dòng sau: mỗi dòng chứa 2 số nguyên i, j . Kết thúc bằng dòng chứa số 0.

Kết quả: file văn bản DATA.OUT, với mỗi bộ dữ liệu đưa ra trên một dòng số nguyên p - số lượng số đo không tin cậy.

Ví dụ:

DATA . INP	DATA . OUT
3	3
3	0
1 2	
2 3	
3 1	
2	
1	
1 2	
0	

Gợi ý. Đây là bài toán về liên thông mạnh trên đồ thị có hướng. Sử dụng thuật toán Tarjan. Coi mỗi viên đá là một đỉnh của đồ thị. Khi viên đá a nặng hơn viên đá b thì có cung (a,b) . Phép đo (a, b) là sai khi đỉnh a và đỉnh b cùng thuộc một vùng liên thông mạnh vì khi đó sẽ có đường đi một chiều từ j tới i , nghĩa là hòn đá j nặng hơn hòn đá i (mâu thuẫn)

Chương trình.

```
const fi      = 'data.inp' ;
      fo      = 'data.out' ;
type  link    = ^node;
      node    = record
                s      : word;
                next   : link;
      end;
tree     = array [0..1000] of link;
ml       = array [0..1000] of word;
var  sv,      {Số vùng liên thông mạnh}
     id,      {số thứ tự thăm của đỉnh trên cây tìm kiếm DFS}
     n,m,     {số đỉnh, số cạnh}
     top : word; {đỉnh của stack}
     a,      {mảng theo dõi thứ tự thăm của các đỉnh}
```


b , $\{b[i]\}$ là số nhỏ nhất trong các số thứ tự thăm của các đỉnh có thể tới được từ các đỉnh thuộc cây con có gốc là i trong cây tìm kiếm DFS, dùng phát hiện các chốt. Đỉnh i là chốt khi $a[i]=b[i]$. Các đỉnh thuộc cây con có gốc là chốt i sẽ tạo thành một vùng liên thông mạnh}

p , $\{p[i]\}$ là số hiệu vùng liên thông mạnh của đỉnh i

s : ml; {stack chứa các đỉnh đã thăm khi xây dựng cây tìm kiếm DFS}

dsk : tree; {danh sách kề, tổ chức theo kiểu danh sách liên kết một chiều}

f, g : text;

function min(u, v : word) : word;

begin

if $u < v$ then min:= u else min:= v ;

end;

Procedure dfs(i : word); {Tìm kiếm theo chiều sâu từ đỉnh i }

var j : word;

t : link;

begin

inc(id); {số thứ tự thăm của đỉnh i }

$a[i]$:= id; {xác nhận số thứ tự thăm của đỉnh i }

$b[i]$:= id; {khởi trị $b[i]$ tạm thời bằng $a[i]$ }

t := $dsk[i]$; {danh sách các đỉnh mà i có thể đi tới}

inc(top); {tăng chiều cao stack thêm một đơn vị}

$s[top]$:= i ; {nhập i vào stack: vì đã thăm tới i }

while ($t > nil$) do begin {duyệt mọi đỉnh kề với i (có cùng đi tới từ i)}

j := $t.s$; { j : đỉnh kề với i }

if $p[j]=0$ then { j chưa thuộc vùng liên thông nào (coi như chưa bị loại)}

if $a[j]=0$ then begin {nếu đỉnh j chưa được thăm}

dfs(j); {gọi đệ quy thăm j }

$b[i]$:= min($b[i], b[j]$); {sau đệ qui có giá trị $b[j]$, nên sửa lại $b[i]$ }

end else {nếu j đã được thăm trước i }

$b[i]$:= min($b[i], a[j]$); {sửa lại $b[i]$ vì có cùng (i, j)}

t := $t.next$; {chuẩn bị tới đỉnh kế khác của i }

end;

if $b[i]=a[i]$ then begin {điều kiện để đỉnh i là đỉnh chốt}

inc(sv); {tăng số thành phần liên thông mạnh}

{lấy ra khỏi stack các đỉnh thuộc cây con có gốc là chốt i , chúng cùng thuộc một vùng liên thông mạnh }

repeat

j := $s[top]$;

dec(top);

$p[j]$:= sv; {xác nhận đỉnh j thuộc vùng sv, cũng là dấu hiệu loại j }

until $i=j$;

end;

end;

Procedure init; {Khởi trị một số biến trước khi xử lý một test}

```

begin
  fillchar(a,sizeof(a),0);  fillchar(b,sizeof(b),0);
  fillchar(p,sizeof(p),0);  fillchar(s,sizeof(s),0);
  top := 0;  sv := 0;  id := 0;
end;
Procedure find; {xây dựng cây tìm kiếm DFS}
var i : word;
begin
  for i:=1 to n do
    if a[i]=0 then dfs(i);
  end;
var i,j,u,v,
  ds : word; {Kết quả của từng test}
  t : link;
BEGIN
  assign(g,fo); rewrite(g);
  assign(f,fi); reset(f);
  while not seekeof(f) do begin
    readln(f,n); {số đỉnh}
    if n=0 then break;
    for i:=1 to n do begin {Khởi trị các danh sách kề của các đỉnh}
      new(dsk[i]);
      dsk[i]:= nil;
    end;
    readln(f,m); {Số lần thí nghiệm (số cung)}
    init;
    for i:=1 to m do begin {tạo các danh sách kề}
      readln(f,u,v); {đọc cung (u,v):viên đá u nặng hơn viên đá v}
      {thêm đỉnh v vào danh sách kề của đỉnh u bằng một danh sách liên kết}
      new(t);
      t^.s:=v;
      t^.next:=dsk[u];
      dsk[u]:=t;
    end;
    find; {Thuật toán Tarjan tìm các vùng liên thông mạnh}
    {Tìm đáp số: là số lượng các cung (i,j) mà đỉnh i và đỉnh j cùng thuộc một vùng liên thông mạnh}
    ds := 0;
    for i:=1 to n do begin
      t := dsk[i];
      while t<>nil do begin
        j := t^.s;
        if p[i]=p[j] then inc(ds);
        t := t^.next;
      end;
    end;
  end;
end;

```

```
writeln(g, ds) ; {ghi đáp số của một test}
end;
close(f) ; close(g) ;
END.
```

Bài 10. Mạng máy tính (Thi HSG Quốc Gia 2006 - Bảng A)

Một hệ thống n máy tính (các máy tính được đánh số từ 1 đến n) được nối lại thành một mạng bởi m kênh nối, mỗi kênh nối hai máy nào đó và cho phép ta truyền tin một chiều từ máy này đến máy kia. Giả sử s và t là hai máy tính trong mạng. Ta gọi đường truyền tin từ máy s đến máy t là một dãy các máy tính và các kênh nối chúng có dạng:

$$s = u_1, e_1, u_2, \dots, u_i, e_i, u_{i+1}, \dots, u_{k-1}, e_{k-1}, u_k = t,$$

trong đó u_1, u_2, \dots, u_k là các máy tính trong mạng, e_i – kênh truyền tin từ máy u_i đến máy u_{i+1} ($i = 1, 2, \dots, k-1$).

Mạng máy tính được gọi là thông suốt nếu như đối với hai máy u, v bất kỳ ta luôn có đường truyền tin từ u đến v và đường truyền tin từ v đến u . Mạng máy tính được gọi là hầu như thông suốt nếu như đối với hai máy u, v bất kỳ, hoặc là có đường truyền tin từ u đến v hoặc là có đường truyền tin từ v đến u .

Biết rằng mạng máy tính đã cho là hầu như thông suốt nhưng không thông suốt.

Yêu cầu: Hãy xác định xem có thể bổ sung đúng một kênh truyền tin để biến mạng đã cho trở thành thông suốt được không?

Dữ liệu: Vào từ file văn bản ONEARC.INP: Dòng đầu tiên ghi hai số nguyên n và m . Dòng thứ i trong số m dòng tiếp theo mô tả kênh nối thứ i bao gồm hai số nguyên dương u_i và v_i cho biết kênh nối thứ i cho phép truyền tin từ máy u_i đến máy v_i , $i=1,2,\dots,m$. Các số trên cùng một dòng được ghi cách nhau bởi dấu cách.

Kết quả: Ghi ra file văn bản ONEARC.OUT: Dòng đầu tiên ghi ‘YES’ nếu câu trả lời là khẳng định, ghi ‘NO’ nếu câu trả lời là phủ định. Nếu câu trả lời là khẳng định thì dòng thứ hai ghi hai số nguyên dương u, v cách nhau bởi dấu cách cho biết cần bổ sung kênh truyền tin từ máy u đến máy v để biến mạng thành thông suốt.

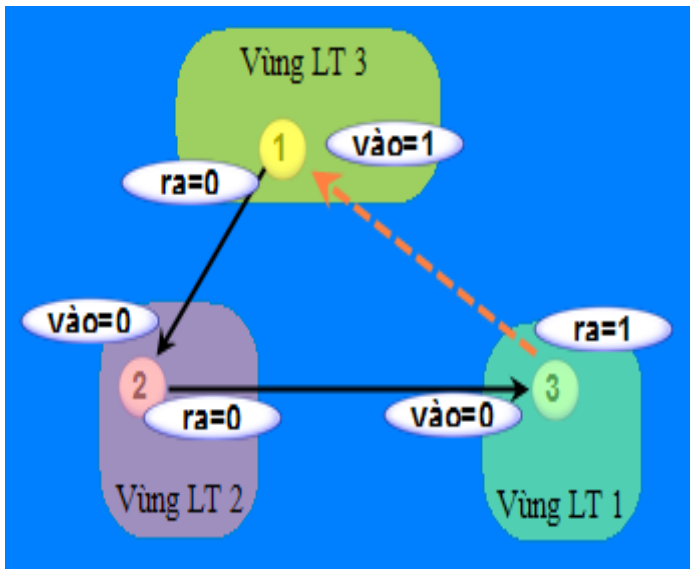
Ví dụ:

ONEARC . INP	ONEARC . OUT
3 2	YES
1 2	3 1
2 3	

Hạn chế: Trong tất cả các test: $n \leq 2000$, $m \leq 30000$. Có 50% số lượng test với $n \leq 200$.

Gợi ý. Tìm các thành phần liên thông mạnh trên đồ thị có hướng. Do đồ thị “hầu như thông suốt” nhưng “chưa thông suốt” nên chỉ có duy nhất 2 vùng liên thông mạnh không có cung nối với nhau. Cung cần bổ xung là cung nối 2 vùng liên thông mạnh này. Thực hiện cụ thể như sau:

Mỗi vùng liên thông mạnh tạm đặt có một đỉnh đi vào nó và một đỉnh ra khỏi nó. Ban đầu đánh dấu đỉnh vào và đỉnh ra của các vùng liên thông mạnh bởi số 1. Duyệt mọi cung (u,v) . Nếu đỉnh u và đỉnh v thuộc hai vùng liên thông $Lt[u]$ và $Lt[v]$ khác nhau thì đỉnh vào vùng liên thông của v bằng 0 và đỉnh ra của vùng liên thông của u bằng 0. Cuối cùng nối cung từ đỉnh i thuộc vùng liên thông có đỉnh ra bằng 1 tới đỉnh j thuộc vùng liên thông có đỉnh vào bằng 1.



Chương trình.

```
const fi      = 'ONEARC.IN';
      fo      = 'ONEARC.OUT';
      maxM    = 30001;
      maxN    = 2001;
type mang1    = array[1..maxM] of integer;
      mang2    = array[1..maxN] of integer;
      mang3    = array[1..maxN] of byte;
var f, g      : text;
      d, c,
      ke      : ^mang1;
      A, B,
      tro,
      S ,
      LT      : mang2;
      visit   : mang3;
      N, M,
      id, top,
      solt,
      xp, kt   : integer;
      vao, ra  : mang3;
```

Procedure allocate;

begin

new(d); new(c); new(ke);

end;

Procedure read_input; *{Đọc tệp input}*

var i: integer;

begin

assign(f, fi); reset(f);

read(f, N, M); *{số đỉnh, số cung}*

for i:=1 to M do

read(f, d^[i], c^[i]); *{đỉnh đầu và đỉnh cuối một cung}*

close(f);

end;

Procedure init; *{tạo danh sách đỉnh kề theo kiểu Forward Star}*

var i, u, v: integer;

begin

for i:=1 to n do tro[i]:=0;

for i:=1 to m do inc(tro[d^[i]]);

v:=0;

for i:=1 to n do begin

u:=tro[i];

tro[i]:=v+1;

v:=v+u;

```

end;
tro[n+1]:=v+1;
for i:=1 to m do begin
    u:=d^[i];
    v:=c^[i];
    ke^[tro[u]]:=v;
    inc(tro[u]);
end;
for i:=n downto 2 do tro[i]:=tro[i-1];
tro[1]:=1;
end;
Procedure DFS(u: integer); {tìm miền liên thông mạnh chứa u}
var i,v: integer;
begin
    inc(id); {số thứ tự thăm đỉnh}
    A[u]:=id;
    B[u]:=A[u]; {B[i]: min của các A[j] mà j có thể tới được từ một đỉnh thuộc nhánh có gốc i}
    visit[u]:=1; {đã thăm u}
    inc(top); {tăng chiều cao Stack}
    s[top]:=u; {nhập u vào Stack}
    for i:=tro[u] to tro[u+1]-1 do begin
        v:=ke^[i]; {v: một đỉnh kề của u}
        if LT[v]=0 then begin {v chưa thuộc vùng liên thông nào}
            if (visit[v]=1) and (A[v]<B[u]) then {v đã thăm trước u}
                B[u]:=A[v]; {sửa lại B[u] cho phù hợp}
            if visit[v]=0 then begin {v chưa thăm}
                DFS(v); {thì thăm v}
                if B[v]<B[u] then B[u]:=B[v]; {sau đó sửa lại B[u] cho phù hợp}
            end;
        end;
    end;
    if A[u]=B[u] then begin {u là đỉnh chốt}
        inc(solt); {số hiệu thành phần liên thông mạnh mới tìm được}
        repeat {đánh dấu các đỉnh thuộc vùng liên thông mạnh này và loại trừ chúng}
            v:=s[top];
            dec(top);
            LT[v]:=solt; {đánh dấu đỉnh v thuộc vùng liên thông mạnh solt}
        until v=u;
    end;
end;
Procedure find; {Thuật toán tìm cung cần bổ sung}
var i, j, u,v: integer;
begin
    for i:=1 to n do visit[i]:=0; {Khởi trị mảng visit}

```

```

for i:=1 to n do LT[i]:=0; {Khởi trị mảng đánh dấu đỉnh thuộc vùng liên
thông nào}
solt:=0; {Khởi trị số vùng liên thông mạnh}
id:=0;
top:=0;
for i:=1 to n do {Thuật toán Tarjan tìm các thành phần liên thông mạnh}
  if visit[i]=0 then DFS(i);
{Khởi trị đầu vào và đầu ra của các vùng liên thông mạnh}
For i:=1 to solt do begin
  vao[i]:=1; ra[i]:=1;
end;
for i:=1 to m do begin {sửa lại giá trị đầu vào và đầu ra của các vùng ltm}
  u:=d^[i]; v:=c^[i]; {có cung (u,v)}
  if LT[v]<>LT[u] then begin {hai vùng liên thông chứa u và v khác nhau}
    vao[LT[v]]:=0; {đầu vào của v, đầu ra của u gán lại bằng 0}
    ra[LT[u]]:=0;
  end;
end;
{Tìm cung (i,j) cần bổ xung: điểm đầu là đỉnh i thuộc vùng liên thông có đầu ra bằng 1,
điểm cuối là đỉnh j thuộc vùng liên thông có đầu vào bằng 1}
for i:=1 to n do
  if (ra[LT[i]]=1) then begin xp:=i; break; end;
for j:=1 to n do
  if (vao[LT[j]]=1) then begin kt:=j; break; end;
end;
Procedure write_out;
begin
  assign(g,fo); rewrite(g);
  writeln(g,'YES');
  writeln(g,xp,#32,kt);
  close(g);
end;
BEGIN
  allocate;
  read_input;
  init;
  find;
  write_out;
END.

```

Bài 11.Sói và cừu

Có một số con cừu trong trại chăn nuôi của Mickey. Trong khi Mickey đang ngủ say, những con sói đói đã vào trại và tấn công đàn cừu.

Trại có dạng hình chữ nhật gồm các ô tổ chức thành hàng và cột. Kí tự dấu chấm ‘.’ là ô trống, kí tự ‘#’ là hàng rào, kí tự ‘o’ là cừu và kí tự ‘v’ là chó sói. Chúng ta coi 2 ô là cùng một miền nếu có thể chuyển từ ô nọ tới ô kia bằng đường đi chỉ gồm các đường đi theo chiều ngang hoặc thẳng đứng không vướng hàng rào. Các ô mà từ chúng có thể thoát khỏi sân không được xem là một phần của bất kì miền nào.

May thay, những con cừu biết tự vệ. Chúng có thể chiến đấu với những con sói trong miền (húc chết sói) nếu số lượng cừu lớn hơn số lượng sói trong cùng một miền. Ngược lại những con sói sẽ ăn hết các con cừu trong cùng một miền. Ban đầu các con cừu và các con sói đã được xác định trong các miền của trại.

Viết một chương trình tính số lượng cừu và số lượng sói còn lại trong sáng hôm đó.

Dữ liệu vào lấy từ file văn bản SOICUU.IN: Dòng đầu tiên chứa hai số nguyên R và C, $3 \leq R, C \leq 250$, là số hàng và số cột của trại. Mỗi dòng trong R dòng sau gồm C kí tự. Tất cả các kí tự này biểu diễn các vị trí có hàng rào, cừu và chó sói trong trại.

Kết quả ra file văn bản SOICUU: Chỉ một dòng gồm 2 con số : số cừu và số sói còn lại trong trại.

Ví dụ

SOICUU.IN	SOICUU.IN	SOICUU.IN
6 6 ...#.. .#v#.. #v.#.# #.o#.. .###.. ...###	8 8 .#####. #..o...# #.##### #.#v.#.# #.#.o#o# #o.###.. #.v...v.# .#####.	9 12 .###.#####.. #.oo#...#v#.. #..o#.#.#.#.. #..#o#...#.. #.#v#o####.. #...#v#...#.. #...#v#v####.. .#####.#vv.o#####.
SOICUU.OUT	SOICUU.OUT	SOICUU.OUT
0 2	3 1	3 5

Gợi ý. Sử dụng thuật toán tìm kiếm các miền liên thông. Trong mỗi miền liên thông đếm số cừu và số sói trong đó. Nếu số cừu lớn hơn số sói thì coi như số sói còn lại trong miền này bằng 0, ngược lại thì số cừu còn lại trong miền này bằng 0. Khi loang tới ô nào thì xoá ô đó bằng gán kí tự '#' trên ô đó.

Chương trình

```
const fi    = 'soicuu.in';
      fo    = 'soicuu.out';
      maxn  = 250;
      maxm  = maxn;
      maxq  = maxn*maxm;
type mang  = array[0..maxn+1,0..maxm +
1] of char;
      mangxy = array[0..maxq] of byte;

var a      : ^mang;
    n, m,
    c, s,      {số cừu và số sói trong cùng một miền liên thông}
    tc, ts    : longint; {tổng số cừu còn lại, tổng số sói còn lại}
    qx        : ^mangxy; {hàng đợi : toạ độ dòng một ô được nạp}
    qy        : ^mangxy; {hàng đợi : toạ độ cột một ô được nạp}
    dau, cuoi : longint; {đầu và cuối hàng đợi}
```

Procedure read_input;

```
var f      : text;
    i, j   : byte;
begin
    assign(f, fi); reset(f);
    readln(f, n, m); {kích thước trang trại}
    {làm hàng rào xung quanh trang trại}
    for i:=0 to n+1 do a^[i, 0]:='#';
    for i:=0 to n+1 do a^[i, m+1]:='#';
    for i:=0 to m+1 do a^[0, i]:='#';
    for i:=0 to m+1 do a^[n+1, i]:='#';
    for i:=1 to n do begin
        for j:=1 to m do read(f, a^[i, j]); {đọc từng ô của trang trại}
        readln(f);
    end;
    close(f);
end;
```

Procedure push(x, y : byte); {thao tác nạp một ô vào hàng đợi}

begin



```

    qx^[dau]:=x;
    qy^[dau]:=y;
    if dau<maxq then
        inc(dau)
    else dau:=0;
end;
Procedure pop(var x, y : byte); {thao tác lấy một ô khỏi hàng đợi}
begin
    x:=qx^[cuoi];
    y:=qy^[cuoi];
    if cuoi<maxq then inc(cuoi)
    else cuoi := 0;
end;
Procedure xet(x, y : longint);
begin
    if a^[x, y]<>'#' then begin {ô (x,y) thuộc miền liên thông đang xét}
        push(x, y); {nhập ô (x,y) vào hàng đợi}
        if a^[x, y]='v' then s:=s+1 {tăng biến đếm số sói}
        else if a^[x, y]='o' then c:=c+1; {tăng biến đếm số cừu}
        a^[x, y]:='#'; {xóa ô đã loang qua}
    end;
end;
Procedure loang; {tìm kiếm theo chiều rộng để tìm một miền liên thông}
var i, j, x, y : byte;
begin
    dau:=0; {khởi trị đầu hàng đợi}
    cuoi:=0; {khởi trị cuối hàng đợi}
    tc:=0; {tổng số cừu còn lại}
    ts:=0; {tổng số sói còn lại}
    for i:=1 to n do
        for j:=1 to m do {duyệt các ô trong trang trại}
            if (a^[i, j]='v') or (a^[i, j]='o') then begin
                c:=0; {khởi trị số cừu trong miền liên thông mới}
                s:=0; {khởi trị số sói trong miền liên thông mới}
                xet(i, j); {loang tới ô (i,j) là ô đầu tiên của một vùng liên thông mới}
                while dau<>cuoi do begin
                    pop(x, y); {lấy một ô ở đầu hàng đợi, loang tiếp ra 4 ô xung quanh}
                    xet(x+1, y);
                    xet(x, y+1);
                    xet(x-1, y);
                    xet(x, y-1);
                end;
                if c>s then s:=0 {cừu giết hết sói}
                else c:=0; {sói ăn hết cừu}
                tc := tc + c; {cập nhật tổng số cừu còn lại}
            end;
        end;
    end;

```

```

    ts := ts + s; {cập nhật tổng số sói còn lại}
end;
end;
Procedure write_output;
var g : text;
begin
    assign(g,fo); rewrite(g);
    writeln(g,tc, ' ', ts);
    close(g);
end;
BEGIN
    new(a); new(qx); new(qy);
    read_input;
    loang;
    write_output;
END.

```

Bài 12. Thỏ và cà rốt

Trong một mảnh vườn hình chữ nhật có kích thước cạnh là m và n người ta trồng cà rốt trong những ô vuông đơn vị có cạnh bằng 1 đơn vị. Trong mảnh vườn này có một chú thỏ ở trong một hang chiếm diện tích 1 ô vuông đơn vị, chú thỏ này cần xác định miền người ta đã trồng cà rốt có diện tích lớn nhất trong mảnh vườn để đào một đường hầm ngắn nhất theo phương dọc hoặc phương ngang từ hang đến phần diện tích lớn nhất đó. (Hai miền cà rốt là khác nhau nếu không có một cạnh ô vuông nào chung).

Dữ liệu vào từ file văn bản THO.IN.

Dòng đầu tiên ghi 4 số M, N, x, y với x, y là hàng và cột của hang thỏ trong mảnh vườn ($1 \leq M, N \leq 100$).

Trong M dòng tiếp theo, mỗi dòng thứ i có N số 0 hoặc 1 thể hiện hàng thứ i của mảnh vườn với ý nghĩa 0 là không trồng cà rốt, 1 là có trồng cà rốt.

Kết quả: Ghi ra file văn bản THO.OUT:

Dòng đầu ghi S là chiều dài của đường hầm ($S=0$ nếu hang thỏ đang ở trên phần trồng cà rốt có diện tích lớn nhất).



Nếu $S > 0$ thì trong các dòng tiếp theo lần lượt ghi hàng và cột của các ô trên đường hàm bắt đầu từ hàng thỏ đến vùng diện tích lớn nhất trông cà rốt.

Ví dụ

CAROT.IN	CAROT.OUT
6 6 1 1	4
0 0 0 0 1 1	1 1
0 0 0 0 1 1	1 2
0 0 0 0 1 1	1 3
0 0 0 0 1 1	1 4
0 0 0 0 1 1	1 5
1 1 1 0 0 0	

Gợi ý. Dùng thuật toán loang tìm các miền liên thông có cà rốt, sẽ tìm ra miền cà rốt rộng nhất. Sau đó dùng tìm kiếm theo chiều rộng để tìm đường hàm của thỏ qua ít ô nhất từ hàng thỏ tới miền cà rốt rộng nhất.

```
const
  fi                      = 'CAROT.IN';
  fo                      = 'CAROT.OUT';
  maxN                   = 101;
  dh: array[1..4] of integer = (0,-1,0,1);
  dc: array[1..4] of integer = (1,0,-1,0);
type
  mang                    = array[1..maxN*maxN] of
integer;
var
  f, g                    : text;
  M,N                     : integer;
  hxp, cxp                : integer;
  a                        : array[0..maxN,0..maxN] of byte;
  Smax,S                  : integer;
  imax,id                 : integer;
  lt                      : array[0..maxN,0..maxN] of integer;
  kq                      : integer;
  way                     : array[1..2,1..2*maxN] of integer;
  trace                   : array[1..maxN,1..maxN] of byte;
  hkt, ckt                : integer;
  Q                       : array[1..2] of ^mang;
  first,last              : integer;
```

Procedure CapPhat;

```

begin new(q[1]); new(q[2]);end;
Procedure Dondep;
begin dispose(q[1]); dispose(q[2]);end;
Procedure InitQ;
begin first:=1; last:=1;end;
Procedure Put(u,v: integer);
begin q[1]^[last]:=u; q[2]^[last]:=v; inc(last);end;
Procedure Get(var u,v: integer);
begin u:=q[1]^[first]; v:=q[2]^[first]; inc(first);end;
function Qempty: boolean;
begin Qempty:=(first=last);end;
Procedure doc_dulieu;
var i,j: integer;
begin
    fillchar(a,sizeof(a),0);
    assign(f,fi); reset(f);
    readln(f,M,N,hxp,cxp);
    for i:=1 to M do begin
        for j:=1 to N do read(f,a[i,j]);
        readln(f);
    end;
    close(f);
end;
Procedure mien_lienthong(i,j: integer);
var k, il, jl: integer;
begin
    InitQ;
    Put(i,j);
    LT[i,j]:=id; S:=S+1;
    repeat
        Get(i,j);
        for k:=1 to 4 do begin
            il:=i+dh[k]; jl:=j+dc[k];
            if (a[il,jl]=1) and (LT[il,jl]=0) then begin
                Put(il,jl);
                LT[il,jl]:=id;
                S:=S+1;
            end;
        end;
    until Qempty;
end;
Procedure duong_ngan_nhat;
var h,c, k, h1, c1: integer;
begin
    if lt[hxp,cxp]=imax then begin
        kq:=0;

```

```

    exit;
end;
InitQ;
fillchar(Trace,sizeof(Trace),0);
Put(hxp,cxp); Trace[hxp,cxp]:=5;
repeat
    Get(h,c);
    for k:=1 to 4 do begin
        hl:=h+dh[k];
        cl:=c+dc[k];
        if (hl>=1) and (hl<=M) and (cl>=1) and (cl<=N)
            and (Trace[hl,cl]=0) then begin
            Put(hl,cl);
            Trace[hl,cl]:=k;
            if LT[hl,cl]=imax then begin
                kq:=-1;
                hkt:=hl;
                ckt:=cl;
            end;
        end;
    end;
until Qempty;
end;
Procedure Tim_hanhtrinh;
var h,c,k: integer;
begin
    h:=hkt;
    c:=ckt;
    kq:=0;
    repeat
        inc(kq);
        way[1,kq]:=h; way[2,kq]:=c;
        k:=Trace[h,c];
        h:=h+dh[1+(1+ k mod 4) mod 4];
        c:=c+dc[1+(1+ k mod 4) mod 4];
    until Trace[h,c]=5;
end;
Procedure thuat_giai;
var i,j: byte;
begin
    Smax:=0;
    imax:=0;
    id:=0;
    fillchar(LT,sizeof(LT),0);
    for i:=1 to M do

```

```

for j:=1 to M do
  if (LT[i,j]=0) and (a[i,j]=1) then begin
    S:=0;
    inc(id);
    mien_lienthong(i,j);
    if S>Smax then begin
      Smax:=S;
      imax:=id;
    end;
  end;
end;
duong_ngan_nhat;
if kq=-1 then Tim_hanhtrinh;
end;
Procedure inkq;
var i: integer;
begin
  assign(g,fo); rewrite(g);
  writeln(g,Kq);
  if kq>0 then begin
    writeln(fo,hxp, ' ',cxp);
    for i:=kq downto 1 do writeln(g,way[1,i], ' ',way[2,i]);
  end;
  close(g);
end;
BEGIN
  capphat;
  doc_dulieu;
  thuat_giai;
  inkq;
  dondep;
END.

```

Bài 13. Công chúa kén chồng (ZAM)

Megachip IV, Vua của Byteotia, có ý định cho cô con gái xinh đẹp của mình là công chúa Ada kén chồng. Ông ta hỏi con gái rằng cô ta muốn người chồng như thế nào. Công chúa trả lời là cô hy vọng người chồng tương lai của cô sẽ thông minh không bủn xỉn mà cũng không lãng phí quá. Nhà vua suy nghĩ tìm một cuộc thử tài các chàng trai xin cầu hôn công chúa để có thể xếp hạng họ nhằm chọn người hợp nhất cho con gái mình. Sau khi ngẫm nghĩ khá lâu ông nhận thấy rằng nên sử dụng lâu đài (ông đã xây dựng cho dân cư trong vùng Byteotia tiêu khiển) làm đề tài thử sức. Lâu đài gồm một số lượng lớn các phòng, đó là nơi trưng bày những tài sản quý giá của

vương quốc, các thần dân có thể tới chiêm ngưỡng các đồ vật quý giá trưng bày tại đó. Các phòng nối với nhau bởi các hành lang. Để thăm một phòng phải trả một số đồng bytealer (bytealer là đơn vị tiền tệ của Byteotia). Một cuộc tham quan lâu đài được bắt đầu từ phòng vào lâu đài. Nhà vua đã trao cho mỗi chàng trai đến cầu hôn một túi tiền. Ông yêu cầu mỗi chàng trai hãy chọn cách thăm một số phòng (có thể lặp lại) từ phòng vào và cuối cùng là phòng công chúa và cần phải chi tiêu cho hành trình của mình đúng bằng số tiền trong túi của vua ban cho. Lãng phí tiêu quá nhiều tiền trên hành trình thì không tới được phòng công chúa. Ngược lại nếu chàng trai nào quá keo kiệt khi xuất hiện trước công chúa với túi tiền còn thừa thì sẽ bị công chúa yêu cầu tiếp tục thăm lâu đài cho đến khi hết tiền trong túi tiền của họ. Đáng tiếc, không một chàng trai nào thực hiện được yêu cầu của nhà vua. Công chúa vẫn đang chờ đợi chàng trai thích hợp của cô ta. Vì sao bạn không muốn thử giải bài toán này? Bạn có thể viết một chương trình giúp đỡ một chàng trai đang sống ở Byteotia ấy chứ !

Hãy viết một chương trình sao cho:

Đọc từ file văn bản ZAM.IN dữ liệu miêu tả lâu đài, số hiệu phòng công chúa đang ở, tổng số tiền trong mỗi túi; tìm dãy các phòng đi qua theo thứ tự từ phòng vào cho đến phòng công chúa sao cho tiêu đúng hết túi tiền. Ghi hành trình tìm được vào file văn bản ZAM.OUT. Bạn có thể giả sử rằng với mỗi **Dữ liệu vào** luôn tồn tại hành trình. Nếu tìm được nhiều hơn một hành trình bạn chọn một hành trình tùy ý trong chúng.

Dữ liệu vào. Dòng đầu tiên của file văn bản ZAM.IN có 5 số nguyên dương n, m, e, p, b mà $1 \leq n \leq 100, 1 \leq m \leq 4950, 1 \leq e, p \leq n, 1 \leq b \leq 1000$, cách nhau bởi đúng một dấu cách. Số n là số lượng phòng, và m là số hành lang. Các phòng được đánh số từ 1 đến n . Số e là số hiệu của phòng vào lâu đài, còn p là số hiệu phòng có công chúa. Số b là tổng số tiền trong mỗi túi tiền vua ban cho. Trong dòng thứ hai có n số nguyên dương $c_1, c_2, \dots, c_n, 1 \leq c_i \leq 1000$, cách nhau bởi dấu cách. Số c_i là lệ phí mỗi lần vào thăm cho phòng i . Trong m dòng tiếp theo có từng cặp số nguyên dương $x, y, (x < y, 1 \leq x, y \leq n)$; mỗi cặp một dòng, các

số cách nhau một dấu trống. Mỗi cặp biểu thị một hành lang nối phòng x với phòng y.

Kết quả ra. Chương trình của bạn ghi trên dòng thứ nhất (và chỉ một dòng) của file ZAM.OUT dãy các số nguyên dương cách nhau bởi đúng một dấu trống. Dãy này biểu thị các số hiệu của các phòng liên tiếp được thăm bắt đầu từ phòng vào và cuối cùng là phòng công chúa, để tiêu đúng hết túi tiền. Ví dụ.

Ví dụ 1		Ví dụ 2	
ZAM.IN	ZAM.OUT	ZAM.IN	ZAM.OUT
5 6 3 4 9 1 2 3 4 5 2 4 5 4 1 5 1 2 2 3 3 1	3 2 4	5 6 3 4 18 1 2 3 4 5 2 4 5 4 1 5 1 2 2 3 3 1	3 2 4 5 4

Gợi ý. Thực hiện loang (bằng tìm kiếm chiều sâu chẳng hạn), từ trạng thái xuất phát tới trạng thái đích. Kí hiệu mỗi trạng thái là cặp số (t, u) với t là số tiền còn sau khi thăm phòng u. Trạng thái xuất phát là (b-c[e], e). Trạng thái đích là (0, p).

Về tổ chức dữ liệu: để lưu vết hành trình dùng mảng $pre[1..b]^{[0..N]}$ với ý nghĩa: $pre[t, u]$ là phòng đã thăm trước phòng u; khởi trị $pre[t, u] = \infty$.

```
{ $M 50000, 0, 10000 }
const inp    = 'zam.in';
      out    = 'zam.out';
      maxn   = 100;
      maxb   = 1000;

type mang    = array[1..maxn] of integer;
var  a       : array[1..maxn,1..maxn] of boolean;
      pre    : array[0..maxb] of ^mang;
      c      : array[1..maxn] of integer;
      g      : text;
      m,n,e,p,b : integer;

Procedure nhap;
var i,u,v : integer;
begin
```

```

assign(g,inp);reset(g);
fillchar(a,sizeof(a),false);
readln(g,n,m,e,p,b); {số phòng, số hành lang, phòng vào, phòng công chúa,
tiền vua ban}
for i:=1 to n do read(g,c[i]); {lệ phí thăm các phòng}
readln(g);
for i:=1 to m do begin
    readln(g,u,v); {hai phòng u và v có hành lang nối}
    a[u,v]:=true;
    a[v,u]:=true;
end;
close(g);
end;
Procedure khoitri;
var i,j:integer;
begin
    for i:=0 to b do begin {Khởi trị mảng pre}
        new(pre[i]);
        for j:=1 to maxn do pre[i]^j:=maxint;
    end;
    pre[b-c[e]]^e:=0; {sau khi thăm phòng vào còn b-c[e] đồng, sau phòng e là
phòng chưa rõ số hiệu, tạm gán là 0}
end;
Procedure duyet(tien, u:integer); {từ phòng u ra, còn số tiền là tien, tìm
phòng tiếp theo sau u}
var i:integer;
begin
    if (u=p) and (tien=0) then exit; {Dữ liệu vào bảo đảm có trạng thái này}
    for i:=1 to n do {đề cử các phòng i sau phòng u}
    if a[u,i] {có hành lang thông u và i}
    and (tien-c[i]>=0) then begin {thừa hoặc đủ tiền vào thăm phòng i}
        pre[tien-c[i]]^i:=u; {xác nhận u là phòng trước khi thăm phòng i}
        duyet(tien-c[i],i); {đệ qui tiếp}
    end;
end;
Procedure thuchien;
var ph:array[1..16000]of byte;
    tien,i,j,luutien:integer;
begin
    duyet(b-c[e],e); {duyet từ phòng vào}
    j:=0; {bắt đầu lần ngược hành trình, số phòng đã xét qua được khởi trị là 0}
    tien:=0; {tại phòng công chúa vừa vận hết tiền}
    repeat
        inc(j); {số lượng phòng đi qua}
        ph[j]:=p; {lưu lại hành trình ngược bắt đầu là phòng công chúa}

```

```

luutien := tien; {tiền có sau khi thăm phòng p}
tien    := tien+c[p]; {tiền có trước khi vào phòng p}
p       := pre[luutien]^[p]; {phòng trước phòng p}
until tien=b; {kết thúc khi tiền có trước hành trình bằng tiền vua ban cho }
assign(g,out);rewrite(g);
for i:=j downto 1 do write(g,ph[i], ' '); {ghi hành trình vào output}
close(g);
end;
BEGIN
    nhap;
    khoitri;
    thuchien;
END.

```

Bài 14. Mạng trường học (IOI 1996 - Network of Schools)

Một số trường học được nối với nhau bởi một mạng máy tính. Có một sự thoả thuận giữa các trường học này: Mỗi trường học có một danh sách các trường học (gọi là danh sách các “trường nhận”) và mỗi trường khi nhận được một phần mềm từ một trường khác trong mạng hoặc từ bên ngoài cần phải chuyển phần mềm nhận được cho các trường trong danh sách các trường nhận của nó. Cần chú ý rằng nếu B thuộc danh sách các trường nhận của A thì A không nhất thiết phải xuất hiện trong danh sách các trường nhận của B.

Người ta muốn gửi một phần mềm đến tất cả các trường học trong mạng. Bạn cần viết chương trình tính số ít nhất các trường học cần gửi bản sao của phần mềm này để cho phần mềm đó có thể chuyển tới tất cả các trường học trong mạng theo thoả thuận trên (câu a). Ta muốn chắc chắn rằng khi bản sao phần mềm được gửi đến một trường học bất kì, phần mềm này sẽ được chuyển tới tất cả các trường học trong mạng. Để đạt được mục đích này, ta có thể mở rộng các danh sách các trường nhận bằng cách thêm vào các trường mới. Tính số ít nhất các mở rộng cần thực hiện sao cho khi ta gửi một phần mềm mới đến một trường bất kì trong mạng, phần mềm này sẽ được chuyển đến tất cả các trường khác (câu b). Ta hiểu một mở rộng là việc thêm một trường mới vào danh sách các trường nhận của một trường học nào đó.

Dữ liệu vào : Input.txt

Dòng đầu tiên chứa số nguyên N là số trường học trong mạng ($2 \leq N \leq 100$). Các trường được đánh số bởi N số nguyên dương đầu tiên. Mỗi một trong N dòng tiếp theo mô tả một danh sách các trường nhận. Dòng thứ $i+1$ chứa số hiệu các trường nhận của trường i . Mỗi danh sách kết thúc bởi số 0. Dòng tương ứng với danh sách rỗng chỉ chứa một số 0.

Kết quả ra : Output.txt

Chương trình của bạn ghi ra 2 dòng: Dòng thứ nhất ghi một số nguyên dương là lời giải của câu a, dòng thứ hai ghi lời giải của câu b.

Ví dụ

INPUT.TXT	OUTPUT.TXT
5	1
2 4 3 0	2
4 5 0	
0	
0	
1 0	

Gợi ý.

Theo <http://www.ioinformatics.org/locations/ioi96/contest/ioi96n.pas>

Mạng trường học có thể biểu diễn bằng một đồ thị có hướng $G(V, E)$ mà tập đỉnh V của nó là các trường học, và tập cung E gồm các cung (A, B) là cung nếu trường B nằm trong danh sách các trường nhận của trường A . Kí hiệu $p \rightarrow q$ nếu có một đường đi một chiều từ p đến q trong đồ thị G . Một tập các đỉnh $D \subset V$, được gọi là tập *thống trị* của G nếu mọi đỉnh $q \in V$ đều có một đỉnh $p \in D$ mà $p \rightarrow q$.

Câu a) là tìm một tập thống trị với ít phần tử nhất.

Tập CD gồm một số đỉnh trong G được gọi là tập *bị thống trị* của G nếu mọi đỉnh p đều có một đỉnh $q \in CD$ sao cho $p \rightarrow q$.

Đồ thị G được gọi là liên thông mạnh nếu với mọi cặp đỉnh p và q đều có đường $p \rightarrow q$ và $q \rightarrow p$ trong G .

Lời giải của câu b) là số nhỏ nhất các cạnh mới cần thêm vào để G liên thông mạnh.

Giả sử D là tập thống trị nhỏ nhất và CD là tập bị thống trị nhỏ nhất của G . Đáp án của câu b) là 0 nếu G liên thông mạnh và bằng $\max(|D|, |CD|)$ trong trường hợp ngược lại.

Sự chứng minh suy từ các bổ đề 1 và 2 dưới đây.

Không mất tổng quát chúng ta giả sử rằng $|D| \leq |CD|$

Bổ đề 1. Nếu D là tập chỉ có một phần tử là p và CD gồm các phần tử q_1, \dots, q_k thì đưa vào các cung mới là $(q_1, p), \dots, (q_k, p)$ tạo cho G liên thông mạnh.

Thật vậy: Giả sử u và v là 2 đỉnh tùy ý của G . Do có phần tử $q_i \in CD$ mà $u \rightarrow q_i$ nên $u \rightarrow q_i \rightarrow p \rightarrow v$ là đường đi từ u tới v .

Bổ đề 2. Nếu $|D| > 1$ có đỉnh $p \in D$ và đỉnh $q \in CD$ sao cho khi đưa vào cung mới (q, p) trong G sẽ tạo ra $D - \{p\}$ là tập thống trị mới và $CD - \{q\}$ là tập bị thống trị mới của G .

Thật vậy: Do $|D| > 1$ nên có 2 đỉnh phân biệt p_1 và p_2 trong D và có 2 đỉnh phân biệt q_1 và q_2 trong CD sao cho $p_1 \rightarrow q_1$ và $p_2 \rightarrow q_2$. Do đó cung mới sẽ là cung (q_1, p_2) . Khi đó một đỉnh u bất kỳ có thể tới được từ p_2 bằng đường đi $p_2 \rightarrow u$ nên tới được từ p_1 bằng đường đi $p_1 \rightarrow q_1 \rightarrow p_2 \rightarrow u$ và một đường bất kỳ $v \rightarrow q_1$ sẽ nối thành $v \rightarrow q_1 \rightarrow p_2 \rightarrow q_2$ trong đồ thị mới. Nghĩa là p_2 có thể loại khỏi tập thống trị cũng như q_1 có thể loại khỏi tập bị thống trị.

Mặt khác chú ý rằng: Tập bị thống trị trong G chính là tập thống trị trong đồ thị chuyển vị GT của G và ngược lại. Do đó chúng ta có thể tính tập bị thống trị nhỏ nhất của G bằng cách tạo đồ thị GT là đồ thị chuyển vị G sau đó tính tập thống trị nhỏ nhất trong đồ thị chuyển vị GT.

Thuật toán tính tập thống trị nhỏ nhất như sau:

Dominated := [] ; // Tập các đỉnh đã có đỉnh của tập thống trị đi tới

D := [] ; // Tập thống trị nhỏ nhất

While <có p không thuộc Dominated> Begin

Search(p) ; // đặt các đỉnh có thể tới từ p trong tập Reachable

Dominated := Dominated + Reachable ; // cập nhật Dominated

D := D - Reachable ; // loạ các đỉnh trong Reachable khỏi D

```

Include p in D; // nạp p vào D
End;

```

Hiển nhiên tập D sinh ra bằng thuật toán này là tập thống trị.

Giả sử rằng D gồm các đỉnh p_1, \dots, p_k và D và chưa nhỏ nhất nghĩa là có tập thống trị Q nhỏ nhất gồm các đỉnh q_1, \dots, q_h mà $h < k$. Do D là tập thống trị và Q là tập thống trị nhỏ nhất nên với mỗi $q_i \in Q$ có duy nhất p_i thuộc D mà q_i có thể tới được từ p_i bằng đường đi $p_i \rightarrow q_i$. Nhưng mọi đỉnh đều có thể tới được từ Q do đó p_k cũng tới được từ một đỉnh nào đó trong Q, nói cách khác có $q_i \rightarrow p_k$. Vậy có đường đi từ p_i tới p_k : $p_i \rightarrow q_i \rightarrow p_k$. Thuật toán thực hiện hai lời gọi: Search(p_i) và Search(p_k). Hoặc Search(p_i) hoặc Search(p_k) được thực hiện trước, đỉnh p_k bị loại trừ khỏi D vì p_k có thể tới được từ p_i . Điều này mâu thuẫn với giả thiết D không là nhỏ nhất.

Chương trình { Scientific Committee IOI'96 }

```

Const MaxN=200; // Số tối đa trường học
Type GraphType=Array[1..MaxN,0..MaxN] Of 0..MaxN;
VertexSet=Set Of 1..MaxN;
Var OutFile: Text;
N :Word; // Số trường học
G: GraphType; // Đồ thị biểu diễn mạng trường học
// G[p,0] là số cung đi ra từ p, cung đi ra từ p là (p,G[p,i]), 1 ≤ i ≤ G[p,0]
Domin, // Tập thống trị
CoDomin : VertexSet; // Tập bị thống trị
NoDoms, // Số phần tử của tập thống trị
NoCoDoms : 0..MaxN; // Số phần tử của tập bị thống trị
AnswerB : 0..MaxN; // Trả lời của câu b)
P : 0..MaxN;

```

Procedure ReadInput; // Đọc tệp input, xây dựng G

```

Var InFile : Text;
i,p : Word;
Begin
Assign(InFile, 'input.txt'); Reset(InFile);
ReadLn(InFile,N);
For i:=1 To N Do G[i,0]:=0; // Khởi trị số đỉnh đi tới từ i
For i:=1 To N Do Begin
Read(InFile, p); // đỉnh p tới được từ i
While p>0 Do Begin
Inc(G[i,0]);
G[i,G[i,0]]:=p; // xác nhận p là đỉnh tới được từ i

```

```

        Read(InFile, p);
    End;
    ReadLn(InFile);
End;
Close(InFile);
End { ReadInput };
Procedure ComputeDomin(Const G: GraphType; Var D: VertexSet);
// Tạo tập thống trị nhỏ nhất D của G
Var    Dominated, Reachable: Set of 1..MaxN;
        p: 1..MaxN;
Procedure Search(p:Word); // Tìm các đỉnh có thể tới từ p
Var    i: Word;
Begin
    Exclude(D, p); // loại bỏ p khỏi D
    Include(Dominated, p); // nạp p vào tập thống trị
    For i:= 1 To G[p,0] Do // duyệt các đỉnh kề với p là  $p_i = G[p,i]$ 
        If Not (G[p,i] in Reachable) Then begin
            // nếu chưa có  $p_i$  trong tập có thể tới được từ p là Reachable thì
            Include(Reachable, G[p,i]); // nạp  $p_i$  vào Reachable
            Search(G[p,i]); // tìm tiếp từ  $p_i$ 
        End;
    End { Search };
Begin { ComputeDomin }
    D:=[]; // Khởi trị tập thống trị nhỏ nhất là rỗng
    Dominated:=[]; // Khởi trị tập thống trị
    For p:=1 To N Do
        If Not (p In Dominated) Then Begin
            // nếu p chưa trong tập thống trị
            Reachable:=[p]; // thì xác nhận vào tập có thể tới
            Search(p); // tìm tiếp các đỉnh có thể tới từ p
            Include(D, p); // nạp p vào tập thống trị nhỏ nhất
        End;
    End { ComputeDomin };
Procedure ComputeCoDomin(Const G: GraphType; Var CD:
VertexSet);
// Tìm tập bị thống trị nhỏ nhất của G
Var    GT      : GraphType;    // Đồ thị chuyển vị của G
        p,q      : 1..MaxN;
        i        : Word;
Begin { ComputeCoDomin }
    For p:=1 To N Do GT[p,0]:=0;
    For p:=1 To N Do // Thực hiện xây dựng đồ thị chuyển vị GT
        For i:=1 To G[p,0] Do Begin
            q:=G[p,i];
            Inc(GT[q,0]); GT[q,GT[q,0]]:=p;

```

```

End;
ComputeDomin (GT, CD) // Tìm tập CD là tập thống trị trong GT
End; { ComputeCoDomin }
Begin { Program }
ReadInput;
ComputeDomin (G, Domin) ; // Xây dựng tập thống trị nhỏ nhất trong G
ComputeCoDomin (G, CoDomin) ; // Tập bị thống trị nhỏ nhất trong G
NoDoms:=0; // Khởi trị số phần tử của tập thống trị nhỏ nhất
For p:=1 To N Do // Đếm số phần tử trong tập thống trị nhỏ nhất
If p In Domin Then Inc (NoDoms) ;
NoCoDoms:=0; // Khởi trị số phần tử của tập bị thống trị nhỏ nhất
For p:=1 To N Do // Đếm số phần tử trong tập bị thống trị nhỏ nhất
If p In CoDomin Then Inc (NoCoDoms) ;
If (Domin=[1]) And (CoDomin=[1]) Then // Liên thông mạnh
AnswerB:=0 // Không cần thêm cung nào
Else
If NoDoms > NoCoDoms Then
AnswerB:=NoDoms // Lực lượng tập thống trị lớn hơn thì nó là kết quả
Else
AnswerB:=NoCoDoms; // ngược lại, kết quả là lực lượng tập bị thống trị
Assign (OutFile, 'output.txt'); Rewrite (OutFile);
WriteLn (OutFile, NoDoms) ; // Kết quả câu a)
WriteLn (OutFile, AnswerB) ; // Kết quả câu b)
Close (OutFile) ;
End.

```

Bài 15. Kênh xung yếu

Một hệ thống n máy tính (các máy tính được đánh số từ 1 đến n) được nối thành một mạng bởi m kênh nối, mỗi kênh nối hai máy nào đó và cho phép ta truyền tin một chiều từ máy này đến máy kia. Ta gọi một mạch vòng của mạng đã cho là một dãy các máy tính và các kênh nối chúng có dạng:

$$u_1, e_1, u_2, \dots, u_i, e_i, u_{i+1}, \dots, u_{k-1}, e_{k-1}, u_k, e_k, u_1$$

trong đó u_1, u_2, \dots, u_k là các máy tính khác nhau trong mạng, e_i – kênh truyền tin từ máy u_i đến máy u_{i+1} ($i=1, 2, \dots, k-1$), e_k là kênh truyền tin từ máy u_k đến máy u_1 . Một kênh truyền tin trong mạng được gọi là kênh xung yếu nếu như bất cứ mạch vòng nào của mạng cũng đều chứa nó.

Yêu cầu: Hãy xác định tất cả các kênh xung yếu của mạng đã cho.

Dữ liệu: Vào từ file văn bản CIRARC.INP:

Dòng đầu tiên ghi hai số nguyên dương n và m .

Dòng thứ i trong số m dòng tiếp theo mô tả kênh nối thứ i bao gồm hai số nguyên dương u_i và v_i cho biết kênh nối thứ i cho phép truyền tin từ máy u_i đến máy v_i .

Các số trên cùng một dòng được ghi cách nhau bởi dấu cách.

Kết quả: Ghi ra file văn bản CIRARC.OUT:

Dòng đầu tiên ghi số nguyên k là số lượng kênh xung yếu trong mạng đã cho. Ghi $k=-1$ nếu mạng không chứa kênh xung yếu.

Nếu $k>0$ thì mỗi dòng trong số k dòng tiếp theo ghi thông tin về một kênh xung yếu tìm được theo quy cách mô tả giống như trong file dữ liệu vào.

Ví dụ:

CIRARC.IN	CIRARC.OUT	CIRARC.IN	CIRARC.OUT
2 2	2	3 3	-1
1 2	1 2	1 2	
2 1	2 1	2 3	
		1 3	

Gợi ý. Bằng tìm kiếm theo chiều rộng thăm các đỉnh để tìm mọi chu trình (khi gặp lại một đỉnh đã thăm thì được một chu trình). Trong quá trình thăm, qua cung nào thì tăng số lần thăm cung đó lên một đơn vị. Kết cục, cung nào có số lần thăm bằng tổng số các chu trình thì đó là kênh xung yếu.

Chương trình.

```
const      fi      = 'cirarc.inp';
           fo      = 'cirarc.out';
           nmax    = 1000;
           mmax    = 20000;

type      m1      = array [1 .. mmax] of word;
           m2      = array [0 .. nmax] of integer;

var        n,m      : longint;
           ke       : m1;
           luot     : ^m1;
           idx      : m2;
           ct       : integer;
           q        : m2;
           pre      : m2;
           cung     : m2;
           f,g      : text;
```

```

                tb      : longint;
procedure readfile;
var i,j,k : longint;
    t      : m2;
begin
    fillchar(idx,sizeof(idx),0);
    readln(f,n,m); {số đỉnh, số cung}
    for k:=1 to m do {tổ chức danh sách kề trên đồ thị có hướng theo Forward star}
    begin
        readln(f,i,j);
        inc(idx[i]);
    end;
    for i:=1 to n do idx[i]:=idx[i-1]+idx[i];
    close(f);reset(f);
    readln(f,n,m);
    t := idx;
    for k:=1 to m do begin
        readln(f,i,ke[t[i]]);
        dec(t[i]);
    end;
end;

procedure bfs(s:integer); {thăm từ đỉnh s bằng BFS}
var dau,cuoi : integer;
    i,j,k, u : integer;
begin
    fillchar(pre,sizeof(pre),0);
    pre[s] := -1;
    dau    := 1;
    cuoi    := 1;
    q[dau] := s;
    while dau<=cuoi do begin
        i:=q[dau];
        inc(dau);
        for k:=idx[i-1]+1 to idx[i] do begin {đọc đỉnh kề với đỉnh i}
            j := ke[k]; {j là một đỉnh mà có cung (i,j)}
            if pre[j]=0 then begin {chưa thăm j}
                pre[j] := i; {xác nhận đã thăm j sau khi thăm i}
                cung[j] := k; {xác nhận cung đi tới j là cung thứ k}
                inc(cuoi); {nhập j vào hàng đợi}
                q[cuoi] := j;
            end else
                if pre[j]=-1 then begin {j đã thăm trước, nên có một chu trình}
                    inc(ct); {tăng số lượng chu trình}
                    u := i; {tăng số lượt thăm các cung trên chu trình vừa tìm được}
                    while pre[u]>0 do begin

```

```

        inc(luot^[cung[u]]);
        u:=pre[u];
    end;
    inc(luot^[k]);
end;
end;
end;
end;
procedure progress;
var i,j : longint;
begin
    fillchar(luot^, sizeof(luot^), 0);
    ct := 0; {tổng số các chu trình, kể cả chu trình lặp lại}
    for i:=1 to n do bfs(i); {duyet thăm các đỉnh}
end;
procedure writefile;
var i,j,t : longint;
begin
    t:=0;
    for i:=1 to m do {tính số lượng các kênh xung yếu}
        t := t + ord(luot^[i]=ct); {kênh thứ i là xung yếu nếu số lần thăm nó bằng
tổng số chu trình}
        if (t=0) or (ct=0) then writeln(g,-1) {vô nghiệm}
        else begin
            writeln(g,t); {số lượng kênh xung yếu}
            for i:=1 to m do
                for j:=idx[i-1]+1 to idx[i] do {hiện các kênh xung yếu}
                    if luot^[j]=ct then
                        writeln(g,i, ' ', ke[j]);
                end;
            end;
        end;
end;
BEGIN
    assign(f, fi); reset(f);
    assign(g, fo); rewrite(g);
    new(luot);
    readfile;
    progress;
    writefile;
    dispose(luot);
    close(f); close(g);
END.

```

Bài 16. Khám phá mê cung

Một mê cung gồm N phòng và một số hành lang nối các phòng ($1 \leq N \leq 200$). Giữa hai phòng bất kì có không quá một hành lang nối chúng, các hành lang có thể đi được theo hai chiều.

Một robot khám phá mê cung, xuất phát từ một phòng và đi thăm tất cả các hành lang sao cho mỗi hành lang được đi qua đúng một lần, rồi trở về phòng xuất phát. Khi qua mỗi hành lang, robot sẽ nạp một nguồn năng lượng là c (c có thể âm hoặc dương). Robot xuất phát với năng lượng bằng 0. Trong quá trình di chuyển, robot có thể bị ngừng hoạt động nếu sau khi đi hết hành lang nào đó chỉ còn năng lượng âm.

Yêu cầu tìm một hành trình an toàn cho robot thoả mãn các yêu cầu nêu trên.

Dữ liệu vào từ file MECUNG.IN chứa các thông tin sau :

Dòng đầu ghi hai số N, M

M dòng sau, dòng thứ i mô tả hành lang i gồm 3 số u, v, c với ý nghĩa có hành lang giữa hai phòng u và v với giá trị năng lượng là c ($|c| \leq 10000$).

Kết quả ghi ra file MECUNG.OUT chứa các thông tin sau :

Dòng đầu ghi số 1 hoặc 0 tùy theo có nghiệm hay không.

Nếu có nghiệm thì $M+1$ dòng tiếp theo ghi lần lượt các số hiệu phòng mà robot đi qua : từ phòng xuất phát, ..., rồi trở về phòng xuất phát.

Ví dụ

MECUNG.IN	MECUNG.OUT
6 8	1
1 3 18	5
1 2 -5	3
2 3 -10	4
3 4 20	1
4 1 -5	3
1 6 9	2
3 5 12	1
5 6 -38	6
	5

Gợi ý. Thực hiện các bước sau :

Kiểm tra đồ thị có liên thông hay không. Nếu không liên thông thì ghi vào file output số 0, dừng chương trình.

Kiểm tra có chu trình Euler hay không. Nếu không có thì ghi vào file output số 0, dừng chương trình.

Nếu có thì :

+ Tìm chu trình Euler. Giả sử tìm được chu trình là $\{s[1], s[2], \dots, s[m], s[1]\}$

+ Xây dựng mảng một chiều Total với ý nghĩa : Total[i] là tổng các trọng số của các cạnh mà rô bắt đi qua dọc theo chu trình tìm được từ $s[1]$ đến $s[i]$.

+ Tìm điểm i_0 mà Total[i₀] nhận giá trị nhỏ nhất. Đỉnh i_0 chính là phòng bắt đầu xuất phát của rô bắt. Hành trình cần tìm sẽ là $\{s[i_0], \dots, s[m], s[1], s[2], \dots, s[i_0]\}$

+ Hiện kết quả vào file.

Chương trình.

```
const fi    = 'mecung.in';
      fo    = 'mecung.out';
      max   = 100;
      maxc  = 20000;
      maxe  = 5000;
type  tarr  = array[1.. max, 1.. max] of integer;
var    a      : tarr;
      b      : ^tarr;
      v,deg   : array[1..max] of integer;

      stack  : array[1.. maxe] of integer;
      path   : array[1.. maxe] of integer;
      d      : array[1.. maxe] of longint;
      sol    : boolean;
      n, m,
      sv,top, count : integer;
      f, g      : text;
      total     : longint;
```

```
procedure read_input;
var i, u, v: integer;
begin
  assign(f, fi); reset(f);
```

```

readln(f, n, m);
for u:= 1 to n do
for v:= 1 to n do  a[u, v]:= -maxc;
fillchar(deg,sizeof(deg),0);
total  := 0;
for i:= 1 to m do begin
    readln(f, u, v, a[u, v]);
    a[v, u] := a[u, v]; {Năng lượng trên cạnh (u,v)}
    inc(deg[u]);        {tăng bậc của u}
    inc(deg[v]);        {tăng bậc của v}
    total := total + a[u,v];
end;
close(f);
sol  := false; {khởi trị cờ báo bài toán vô nghiệm}
           {nếu có đỉnh cô lập hoặc đỉnh bậc lẻ thì không tồn tại chu trình Euler}
for i:=1 to n do
if (deg[i]=0) or (deg[i] mod 2 =1) then exit;
    {điều kiện cần để bài toán có nghiệm là total>=0}
if total<0 then exit;
sol  := true; {bài toán có nghiệm}
end;
procedure DFS(i : integer); {tìm kiếm theo chiều sâu một thành phần liên
thông chứa đỉnh i}
var j : integer;
begin
    for j:=1 to n do
    if v[j]=0 then
    if a[i,j]<>-maxc then begin {có cạnh (i,j)}
        v[j] := sv;
        DFS(j);
    end;
end;
end;
procedure stplt; {tìm số thành phần liên thông trên đồ thị vô hướng}
var s : integer;
begin
    fillchar(v,sizeof(v),0);
    sv  := 0;
    for s:=1 to n do
    if v[s]=0 then begin {đỉnh s bắt đầu một vùng liên thông mới}
        inc(sv);
        v[s] := sv;
        DFS(s);
    end;
end;
end;
procedure init; {khởi trị}

```

```

var i : integer;
begin
    new(b);  b^    := a;
    top  := 1;
    stack[1] := 1;
    count  := 0;
end;

procedure euler; {tìm chu trình Euler trên đồ thị vô hướng}
var u, v : integer;
begin
    repeat
        u:= stack[top];
        for v:= 1 to n do
            if a[u, v] <> -maxc then begin
                inc(top);
                stack[top] := v;
                a[u, v]    := -maxc; {xác nhận đã thăm cạnh (u,v)}
                a[v, u]    := -maxc;
                break;
            end;
        if u = stack[top] then begin {không còn đỉnh kề với u chưa thăm}
            inc(count);
            path[count] := u; {ghi nhận u vào chu trình Euler đang tìm}
            dec(top);      {xóa cạnh tới u}
        end;
    until top <= 0;
end;

procedure solve;
var i, p : integer;
    vmin : longint;
begin
    if not sol then begin {vô nghiệm}
        write(g, 0);
        exit;
    end else begin {có chu trình Euler}
        d[1]:= 0; {d[i] là tổng năng lượng khi robot đi từ phòng 1 đến phòng i}
        for i:= 2 to count do
            d[i]:= d[i - 1] + b^[path[i - 1], path[i]];
        {tìm điểm i có d[i] nhỏ nhất}
        vmin := maxlongint;
        for i:= 1 to count do
            if vmin > d[i] then begin
                vmin := d[i];
                p     := i;
            end;
        {ghi file output. Robot đi từ phòng i (tìm được ở trên, theo chu trình Euler tìm được)}
    end;

```

```

        writeln(g, 1);
        for i:= p to count do writeln(g, path[i]);
        for i:= 2 to p do writeln(g, path[i]);
    end;
end;
BEGIN
    assign(g, fo); rewrite(g);
    read_input; {đọc dữ liệu vào từ file input}
    init;
    stplt; {tìm số thành phần liên thông}
    if sv>1 then begin {không liên thông, bài toán vô nghiệm}
        writeln(g, 0);
        close(g);
        exit;
    end
    else begin
        euler; {tìm một chu trình Euler}
        solve; {tìm ra thành phố xuất phát, hiện kết quả vào file output}
    end;
    close(g);
END.

```

Bài 17. Roboco

Công ty phát triển phần mềm tự động hoá Roboco vừa cho xuất xưởng một mô hình Robot mới. Đặc điểm của Robot mới này là nó có thể làm việc theo chương trình soạn cho nó. Trong những chương trình như vậy có các lệnh: thực hiện một bước di chuyển về phía Đông, phía Tây, phía Nam hoặc phía Bắc. Robot thực hiện các lệnh chương trình một cách tuần tự và dừng lại khi gặp dấu hiệu kết thúc chương trình. Các chuyên viên của công ty Roboco muốn xác định xem có bao nhiêu chương trình khác nhau gồm K câu lệnh điều khiển Robot di chuyển từ gốc tọa độ đến điểm có tọa độ (x, y). Giả thiết là 1 đơn vị trên trục tọa độ tương ứng với độ dài của một bước di chuyển của Robot.

Dữ liệu vào từ tệp văn bản ROBOCO.INP gồm nhiều dòng, mỗi dòng chứa 3 số nguyên K, x, y ($0 \leq K \leq 16$, $|x|, |y| \leq 16$) được ghi cách nhau bởi dấu cách.

Kết quả ghi ra tệp văn bản ROBOCO.OUT mỗi dòng là số lượng chương trình tìm được kết quả của một dòng tương ứng trong tệp ROBOCO.INP

Ví dụ

ROBOCO.INP	ROBOCO.OUT
5 2 3	10

Gợi ý. Dùng phương pháp sửa nhãn. Tổ chức dữ liệu: Dùng mảng 3 chiều $D[-16..16, -16..16, 0..k-1]$ of longint. $D[i, j, b]$ có ý nghĩa là số lượng đường đi dài là b từ gốc tọa độ đến điểm (i, j) . Nếu $D[i, j, b] > 0$ thì robot đã tới điểm (i, j) sau b bước. Xét bước tiếp theo là $b+1$, nếu ở bước này robot tới được ô (i', j') - là một trong 4 ô xung quanh ô (i, j) , thì số đường đi tới ô (i', j') sẽ thêm được một lượng là tổng số lượng các đường đi tới ô (i, j) ở bước b . Vậy sau k lần sửa nhãn sẽ có tổng số cách đi từ gốc tọa độ đến ô (x, y) mà độ dài các đường đi bằng k .

Chương trình

```
const inp = 'roboco.inp';
      out = 'roboco.out';
      nmax = 16 ;
      kmax = 16 ;
      di: array[1..4] of integer = (-1, 0, 1, 0) ;
      dj: array[1..4] of integer = (0, 1, 0, -1) ;
type mang = array[ -nmax..nmax, 0.. kmax] of longint;
var f, g:text ;
    k, x, y, i: integer ;
    d: array[-nmax.. nmax] of ^mang;
```

Procedure open_files;

begin

assign(f, inp); reset(f);

assign(g, out); rewrite(g);

end;

Procedure close_files;

begin

close(f); close(g); halt;

end;

Procedure nhap;

begin readln(f, k, x, y); *{Đọc tệp input lấy giá trị k, x, y}* end;

Procedure xuly;

var i, j, b, huong, ih, jh: integer;

begin

for i:=-nmax to nmax do

fillchar(d[i]^, sizeof(d[i]^), 0); *{Khởi trị mảng d}*

d[0]^ [0,0] := 1; *{coi như đứng yên là 1 cách di chuyển}*

for b := 0 to k-1 do *{số lần sửa nhân, sau mỗi bước đi lại sửa nhân}*

for i := -k to k do *{phạm vi tung độ}*

for j := -k to k do *{phạm vi hoành độ}*

for huong := 1 to 4 do *{các hướng di chuyển của robot}*

if d[i]^ [j,b] > 0 then begin *{sau b bước, robot đã tới (i,j)}*

ih := i + di[huong]; *{tung độ ô mới}*

jh := j + dj[huong]; *{hoành độ ô mới}*

inc(d[ih]^ [jh,b+1], d[i]^ [j,b]); *{thêm số đường dài b+1 tới ô (ih,jh)}*

end;

writeln(g, d[x]^ [y, k]); *{ghi kết quả vào tệp output}*

end;

BEGIN

for i:=-nmax to nmax do begin *{xin cấp phát bộ nhớ, khởi trị d[i]}*

new(d[i]);

fillchar(d[i]^, sizeof(d[i]^), 0);

end;

open_files ; *{Mở các tệp input và output}*

while not seekeof(f) do begin

nhap; xuly;

end;

close_files; *{Đóng các tệp input và output}*

END.

Bài 18. Xếp công việc.

Cho đồ thị có hướng, N đỉnh, không chu trình, có trọng số. Mỗi đỉnh thể hiện một công việc trong một dự án. Mỗi cung (i,j) thể hiện công việc i phải làm trước công việc j. Trọng số cung (i, j) thể hiện thời gian hoàn thành công việc j nếu sau công việc i sẽ làm ngay công việc j. Tìm thời gian hoàn

thành sớm nhất của từng công việc. Những đỉnh không có cung vào được coi là công việc đã xong tại mốc thời gian 0, những đỉnh không có cung ra được coi là các công việc kết thúc của một phần dự án.

Dữ liệu vào từ file XEPCV.IN: Dòng đầu là số n và m tương ứng là số đỉnh và số cung; M dòng tiếp theo mỗi dòng 3 số i, j, x thể hiện trọng số cung (i, j) là x .

Kết quả ghi ra file XEPCV.OUT. Dòng đầu là thời gian sớm nhất hoàn thành toàn bộ dự án. Dòng thứ hai ghi các thời điểm hoàn thành sớm nhất của từng công việc lần lượt từ công việc 1 đến công việc n .

Gợi ý. Đây là bài toán tìm đường đi ngắn nhất trên đồ thị có hướng, có trọng số, không chu trình.

Chương trình.

```
const    fi      = 'xepcv.in';
         fo      = 'xepcv.out';
         maxn    = 1000;
         maxm    = 10000;
type     mang1   = array[1..maxn] of integer;
         mang2   = array[1..maxm] of integer;
var      f, g    : text;
         n, m    : integer;
         v,l,t,p,id, kq ,lv: mang1;
         a,c     : mang2;
procedure read_input;
var f      : text;
    sc, i,j : integer;
    sl, x   : integer;
begin
    assign(f,fi); reset(f);
    fillchar(p,sizeof(p),0);
    fillchar(a,sizeof(a),0);
    fillchar(c,sizeof(c),0);
    {tổ chức danh sách kề trên đồ thị có h□ớng}
    readln(f,n,m); {số đỉnh, số cung}
    for sl:=1 to m do begin
        readln(f,i,j,x);
        if x>0 then begin
            inc(v[j]); {số cung vào đỉnh j}
            inc(p[i]); {dùng trong danh sách kề kiểu Forward star}
        end;
    end;
end;
```

```

lv := v;
for j:=2 to n do p[j] := p[j-1] + p[j];
close(f);
reset(f);
readln(f,n,m);
for sl:=1 to m do begin
    readln(f,i,j,x);
    if x>0 then begin
        a[p[i]] := j; {p[i] trở vào đỉnh j có cung từ i tới j}
        c[p[i]] := x; {cung (i, a[p[i]]) có trọng số là x}
        dec(p[i]);
    end;
end;
close(f);
p[n+1] := m;
end;

procedure topo; {Đánh lại chỉ số các đỉnh. Trong tìm kiếm BFS đỉnh nào thăm trước
sẽ có chỉ số nhỏ hơn}
var    q : mangl;
        i, j, sh ,dau, cuoi, u : integer;
begin
    fillchar(q,sizeof(q),0); {Khởi trị hàng đợi}
    dau := 0;    cuoi := 0;
    for i:=1 to n do { nạp các đỉnh không có cung vào (bậc vào =0) vào q }
    if v[i]=0 then begin
        inc(cuoi);
        q[cuoi] := i;
    end;
    sh := 0; {khởi trị số hiệu thăm}
    while dau<cuoi do begin
        inc(dau);
        u := q[dau]; {lấy u khỏi hàng đợi}
        inc(sh);
        id[u] := sh; {đánh chỉ số cho u}
        for j:=p[u]+1 to p[u+1] do begin {xét các đỉnh a[j] mà u đi tới}
            dec(v[a[j]]); {loại u khỏi đồ thị nên giảm bậc vào các đỉnh u đi tới}
            if v[a[j]]=0 then begin {nếu đỉnh a[j] không còn cung vào}
                inc(cuoi);
                q[cuoi] := a[j]; {thì nạp đỉnh a[j] vào hàng đợi q}
            end;
        end;
    end;
end;

procedure repair; {Sửa chữa nhãn cho các đỉnh}
var    i, j, u : integer;

```

```

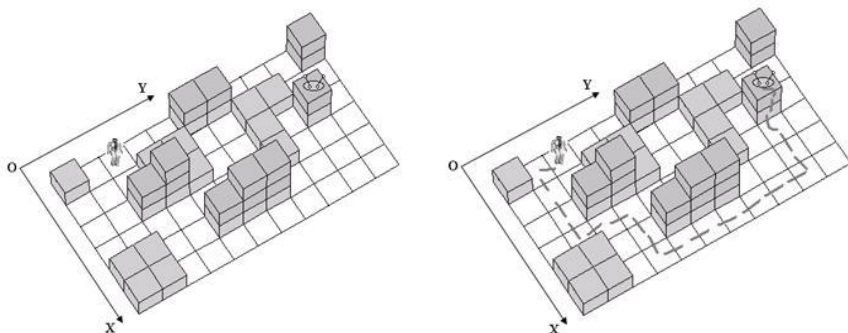
begin
  for i:=1 to n do l[i] := maxint div 2; {Khởi trị nhãn các đỉnh}
  fillchar(t, sizeof(t), 0); {mảng vết}
  {gán nhãn cho các đỉnh có bậc vào=0}
  for i:=1 to n do
    if lv[i]=0 then l[i]:=0;
  {sửa nhãn cho các đỉnh còn lại dựa vào nhãn đỉnh u có chỉ số i tăng dần}
  for i:=1 to n-1 do
    for u:= 1 to n do
      if id[u]=i then begin
        for j:=p[u]+1 to p[u+1] do begin {xét các đỉnh kề với u là a[j]}
          if l[a[j]]>l[u]+c[j] then begin {nhãn đỉnh a[j] chưa tối ưu}
            l[a[j]] := l[u] + c[j]; {sửa nhãn đỉnh a[j]}
            t[a[j]] := u; {vết: trước a[j] là u}
          end;
        end;
        break; {thoát vòng for u}
      end;
    end;
  end;
procedure write_output;
var f : text;
    i, j, li : integer;
    maxl : integer;
begin
  maxl := 0;
  for i:=1 to n do {tìm thời hạn xong toàn bộ dự án}
    if l[i]>maxl then begin
      li := i;
      maxl := l[i];
    end;
  assign(f,fo); rewrite(f);
  writeln(f,maxl); {thời gian xong toàn bộ dự án}
  for i:=1 to n do write(f,l[i], ' '); {th.gian sớm nhất xong công việc i}
  close(f);
end;
BEGIN
  read_input;
  topo;
  repair;
  write_output;
END.

```

Bài 19. Thi robocon.

Trong một cuộc thi Robocon người ta tổ chức phần thi “leo cột” – phần thi thể hiện tốc độ tư duy và kỹ năng, kỹ xảo của các chú Robot. Một

sa bàn kích thước $M \times N$ ($0 < M, N < 101$) được chia thành các lưới ô vuông đơn vị, kích thước 1×1 gắn với trục tọa độ OXY như hình vẽ bên dưới. Kích thước M tính theo đơn vị trục OX.



Trên các ô vuông người ta xây các cột hình khối chữ nhật có mặt cắt 1×1 và chiều cao là một số nguyên dương không vượt quá 100, các cột có chiều cao bằng nhau mà giáp nhau thì sẽ tạo thành một mặt bằng.

Khi đến phần thi của mình, mỗi chú Robot được đặt vào một ô vuông tọa độ (x_1, y_1) – đó có thể là bề mặt sa bàn hoặc đỉnh của một cột. nào đó. Tại một vị trí (x_2, y_2) khác người ta đặt một thiết bị phát sóng, các Robot có khả năng bắt sóng và tính ra được vị trí đặt thiết bị này. Từ vị trí (x_1, y_1) hiện tại Robot phải đi đến vị trí (x_2, y_2) , có thể phải leo lên hay leo xuống các cột trên sa bàn và đường mà Robot đi phải là con đường tốn ít năng lượng nhất. Mỗi bước đi của Robot luôn bắt đầu từ tâm một ô vuông này đến tâm một ô vuông khác có chung đúng một cạnh với nó nhưng có thể có 2 khả năng:

- Hai ô vuông thuộc cùng một mặt phẳng: Robot tốn năng lượng là 1.
- Hai ô vuông không thuộc cùng một mặt phẳng – khi đó chúng sẽ vuông góc với nhau: Robot tốn năng lượng là 2 (do phải sử dụng kỹ năng di chuyển phức tạp hơn).

Nhiệm vụ của bạn là phải lập trình tìm ra con đường này. Đối với các chuyên gia Robot việc cài đặt một chương trình như vậy cho Robot không phải khó khăn và bạn có thể hình dung ra trong thực tế những Robot này sẽ có ích như thế nào.

Dữ liệu vào từ file ROBOT.INP:

- Dòng đầu tiên ghi M, N cách nhau bởi một dấu cách.

- M dòng tiếp theo, mỗi dòng ghi N số. Số thứ j trên dòng i thể hiện độ cao của cột có tọa độ (i,j) trên sa bàn. Độ cao 0 thể hiện rằng không có cột nào tại vị trí đó. Dòng M+2 ghi 4 số x1, y1, x2, y2, các số cách nhau bởi một dấu cách.

Dữ liệu ra file ROBOT.OUT: Ghi một số duy nhất là năng lượng tối thiểu của con đường tìm được cho Robot.

Ví dụ:

ROBOT.INP	ROBOT.OUT
6 11 1 0 0 0 0 2 2 0 0 0 2 0 0 0 1 1 0 0 1 1 0 0 0 0 2 3 1 0 0 1 0 2 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 2 3 3 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 3 2 10	21

Hai hình vẽ trên thể hiện vị trí ban đầu của Robot và một con đường Robot đi tốn năng lượng 21.

Gợi ý. Thực hiện thuật toán loang kết hợp gán nhãn tìm đường đi ngắn nhất từ điểm xuất phát tới điểm đích. Tổ chức dữ liệu dùng Heap là hàng đợi ưu tiên để chọn đỉnh có nhãn tự do nhỏ nhất dùng sửa nhãn cho các đỉnh kề.

Chương trình.

```
{ $M 16384,0,655360 }  
Const fi          = 'robot.inp';  
fo                = 'robot.out';  
maxN              = 101;  
dh: array[1..4] of integer = (0,-1,0,1);  
dc: array[1..4] of integer = (1,0,-1,0);  
type arr1         = array[0..maxN] of integer;  
arr2              = array[0..maxN*maxN] of integer;  
mang1             = array[0..maxN] of ^arr1;  
mang2             = ^arr2;
```

```

    mang3      = array[0..maxN*maxN] of byte;
var   f, g    : text;
    M,N      : integer; {Kích thước sa bàn}
    a        : mang1; {Quản lý sa bàn}
    xp, kt    : integer; {vị trí xuất phát và kết thúc}
    heap      : mang2; {Hàng đợi ưu tiên: Heap }
    nh        : integer; {số phần tử của Heap - số hiệu phần tử cuối của Heap}
    vt        : mang2; {quản lý số hiệu nút trên Heap}
    kc        : mang2; {quản lý nhãn khoảng cách trong khi loang}
    color     : mang3; {đánh dấu thuộc tính của các đỉnh đồ thị}

```

Procedure capphat;

```
var i :integer;
```

```
begin
```

```
    for i:=1 to maxN do new(a[i]);
```

```
    new(heap); new(vt); new(kc);
```

```
end;
```

Procedure down(k: integer); {Phép down heap: chuyển nút k xuống phía lá}

```
var v,l: integer;
```

```
label l1;
```

```
begin
```

```
    v:=heap[k];
```

```
    while 2*k<=nh do begin
```

```
        l:=2*k;
```

```
        if (l<nh) and (kc[heap[l]]]>kc[heap[l+1]]]) then
```

```
            inc(l);
```

```
        if kc[v]<=kc[heap[l]]] then goto l1;
```

```
        heap[k]:=heap[l];
```

```
        vt[heap[k]]]:=k;
```

```
        k:=l;
```

```
    end;
```

```
    l1:
```

```
    heap[k]:=v;
```

```
    vt[v]:=k;
```

```
end;
```

Procedure up(k: integer); {Phép Up heap, chuyển nút k về phía gốc}

```
var v: integer;
```

```
begin
```

```
    v:=heap[k];
```

```
    while kc[v]<kc[heap[k div 2]]] do begin
```

```
        heap[k]:=heap[k div 2];
```

```
        vt[heap[k]]]:=k;
```

```
        k:=k div 2;
```

```
    end;
```

```
    heap[k]:=v;
```

```
    vt[v]:=k;
```

```
end;
```


Procedure init; *{Khởi trị Heap}*

begin

nh:=0;

heap^[0]:=0;

vt^[0]:=0;

kc^[0]:=-1;

end;

Procedure put(u,p: integer); *{Nạp một phần tử u có nhãn p vào heap}*

begin

kc^[u]:=p;

inc(nh);

heap^[nh]:=u;

vt^[u]:=nh;

up(nh);

end;

function get: integer; *{Lấy phần tử gốc Heap ra khỏi Heap}*

begin

get:=heap^[1];

heap^[1]:=heap^[nh];

vt^[heap^[1]]:=1;

dec(nh);

down(1);

end;

Procedure update(u,p: integer); *{cập nhật phần tử u, có nhãn mới là p}*

var k: integer;

begin

kc^[u]:=p;

k:=vt^[u];

up(k);

end;

function empty: boolean; *{Hàm trả về Heap là rỗng hay chưa}*

begin

empty:=(nh=0);

end;

function doiso(u,v: integer): integer; *{Đổi tọa độ (u,v) thành một số}*

begin

doiso:=(u-1)*N+v;

end;

Procedure doidinh(p: integer; var u,v: integer);

{Đổi số là một vị trí thành tọa độ của vị trí đó}

begin

u:=(p-1) div N+1;

v:=(p-1) mod N+1;

end;

Procedure doc_dulieu;

var i,j,x1,y1,x2,y2: integer;

```

begin
  assign(f,fi); reset(f);
  readln(f,M,N); {Đọc kích thước sa bàn}
  for i:=1 to m do begin
    for j:=1 to n do read(f,a[i]^j); {Đọc một ô của sa bàn}
    readln(f);
  end;
  readln(f,x1,y1,x2,y2); {Đọc tọa độ xuất phát và kết thúc}
  close(f);
  xp:=Doiso(x1,y1); {quy đổi tọa độ xuất phát thành một số là xp}
  kt:=Doiso(x2,y2); {quy đổi tọa độ kết thúc thành một số là kt}
end;
Procedure ke(u,k: integer; var v,Ts: integer);
{Tìm vị trí kề với vị trí u theo hướng k}
var i,j,im,jm: integer;
begin
  Doidinh(u,i,j); {quy đổi vị trí u thành ô (i,j)}
  im:=i+dh[k]; {ô (i,jm) kề cạnh với (i,j) theo hướng k}
  jm:=j+dc[k];
  v:=0;
  if (im<1) or (im>m) or (jm<1) or (jm>n) then exit;
  v:=Doiso(im,jm); {quy đổi ô (im,jm) thành vị trí v}
  if a[i]^j=a[im]^jm then ts:=1 {hai ô cùng độ cao thì chi phí là 1}
  else
    ts:=2*abs(a[i]^j-a[im]^jm)+1; {chi phí trong trường hợp độ cao
chênh nhau }
end;
Procedure xuly;
var i,u,v,ts: integer;
begin
  init;
  for i:=1 to M*N do color[i]:=0; {Khởi trị các vị trí là chưa thăm}
  put(xp,0); {gấp vị trí xuất phát với nhãn khoảng cách (chi phí) bằng 0}
  color[xp]:=1; {xác nhận đã thăm vị trí xuất phát}
  repeat {Thực hiện loang và gán nhãn}
    u:=get; {Lấy ra phần tử ở gốc Heap}
    color[u]:=2; {xác nhận cố định nhãn của u}
    if u=kt then exit; {điều kiện thoát: tới vị trí kết thúc}
    for i:=1 to 4 do begin {đề cử mọi đỉnh kề với u, theo 4 hướng i}
      ke(u,i,v,ts); {tìm được vị trí v kề với u, chi phí từ u sang v là ts}
      if (v>0) then begin {v là một vị trí trên sa bàn}
        if (color[v]=1) and (kc[v]>kc[u]+ts) then
          {nhãn v chưa cố định và chưa tối ưu thì sửa nhãn cho v}
          update(v,kc[u]+ts); {sửa nhãn cho v}
      end;
    end;
  until false;
end;

```

```

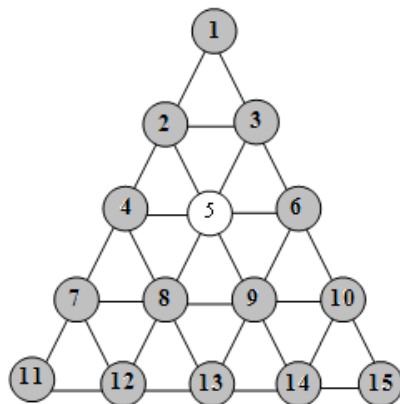
        if color[v]=0 then begin {nếu v chưa thăm thì nạp v vào Heap}
            put(v,kc^[u]+ts);
            color[v]:=1; {xác nhận v là có nhãn tự do}
        end;
    end;
end;
until empty;
end;
Procedure Ghi_ketqua;
begin
    assign(g,fo); rewrite(g);
    writeln(g,kc^[kt]); {ghi vào tệp output, chi phí ít nhất}
    close(g);
end;
BEGIN
    capphat;
    doc_dulieu;
    xuly;
    Ghi_ketqua;
END.

```

Bài 20. Trò chơi bi nhảy

Cho 15 viên bi đặt trong một lưới có 16 ô. Ví dụ như hình bên, tất cả các ô đen là ô đặt bi, ô trắng (chứa số 5) là ô rỗng.

Ta có thể cho một viên bi nhảy qua một hoặc nhiều ô trên một đường thẳng đến ô rỗng gần nhất. Những viên bi nhảy qua sẽ bị lấy ra khỏi lưới. Ví dụ: chúng ta có thể nhảy từ ô 12 đến ô 5 và lấy viên 8 ra khỏi mảng hoặc nhảy từ ô 14 đến ô 5 và lấy ra viên 9.



Hãy tìm cách nhảy sao cho chỉ còn lại đúng một viên và số bước nhảy là ít nhất.

Dữ liệu vào: file Bi_nhay.IN gồm 1 số nguyên trong phạm vi từ 1 đến 15 thể hiện cho ô rỗng (không có bi).

Kết quả: Ghi ra file Bi_nhay.OUT:

Dòng đầu ghi p là số bước nhảy

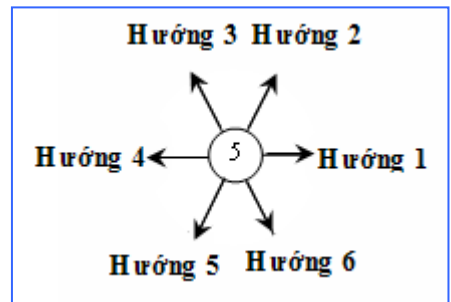
Dòng kế tiếp gồm p cặp số thể hiện cho p bước nhảy

Ví dụ:

Bi_nhay.IN	Bi_nhay.OUT
5	8
	12 5
	10 8
	1 10
	11 1
	14 12
	12 3
	1 6
	15 3

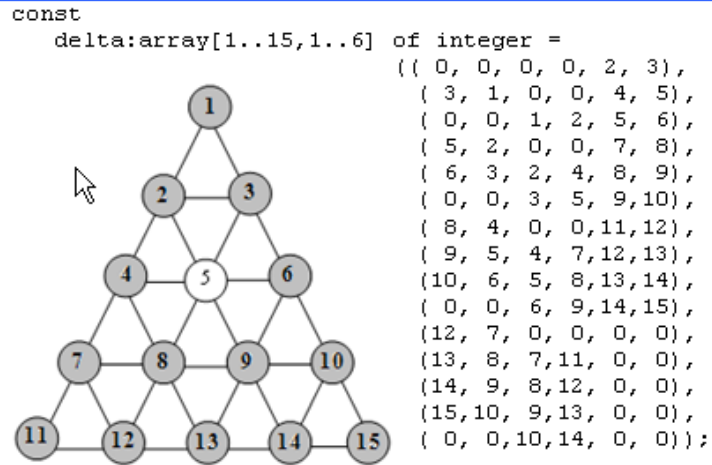
Gợi ý. Xét đồ thị mà mỗi đỉnh là số tương ứng với một trạng thái của trò chơi. Mỗi trạng thái của trò chơi tương ứng với một số nguyên 16 bit, mà bit $15-i$ bằng 1 nếu ô i còn chứa số. Ví dụ trong hình vẽ nêu ở đề bài, ban đầu ô số 5 rỗng, nên trạng thái bàn cờ lúc này là số nguyên viết dưới dạng nhị phân có bit 10 bằng 0 đó là số: 0111101111111111 (số thập phân là 16383). Chúng ta sẽ dùng thuật toán loang cùng tổ chức dữ liệu kiểu hàng đợi để tìm đường đi ngắn nhất từ trạng thái ban đầu đến trạng thái đích (là số chỉ có đúng một bit bằng 1, các bit khác bằng 0).

Mặt khác, cần tổ chức dữ liệu sao cho trên tam giác lưới dễ tìm được các ô kề với một ô nào đó. Có thể tổ chức như sau: xung quanh mỗi ô có tối đa 6 ô kề với nó, ta qui định ô kề nằm ở hướng Đông là hướng 1, tiếp theo theo chiều quay ngược kim đồng hồ lần lượt là các ô kề có hướng 2, 3, 4, 5, 6. Trong



chương trình có thể tạo sẵn mảng hằng delta là mảng hai chiều với ý nghĩa: $\text{delta}[i, k]$ là ô kề với ô i theo hướng k . Ví dụ: Kề với ô 6 theo hướng 4 là ô 5 nên $\text{delta}[6, 4]=5$, kề với ô 6 theo hướng 5 là ô 9 nên $\text{delta}[6, 5]=9$. Ngoài ra qui định $\text{delta}[i, k]=0$ nếu không có ô kề với ô số i theo hướng k . Ví dụ: $\text{delta}[6, 1]=\text{delta}[6, 2]=0$.

Hình vẽ sau minh họa các giá trị của mảng delta :



Chương trình

```
const
  fi                                ='Bi_nhay.IN';
  fo                                ='Bi_nhay.OUT';
  delta:array[1..15,1..6] of integer=(( 0, 0, 0, 0, 2, 3),
                                         ( 3, 1, 0, 0, 4, 5),
                                         ( 0, 0, 1, 2, 5, 6),
                                         ( 5, 2, 0, 0, 7, 8),
                                         ( 6, 3, 2, 4, 8, 9),
                                         ( 0, 0, 3, 5, 9,10),
                                         ( 8, 4, 0, 0,11,12),
                                         ( 9, 5, 4, 7,12,13),
                                         (10, 6, 5, 8,13,14),
                                         ( 0, 0, 6, 9,14,15),
                                         (12, 7, 0, 0, 0, 0),
                                         (13, 8, 7,11, 0, 0),
                                         (14, 9, 8,12, 0, 0),
                                         (15,10, 9,13, 0, 0),
                                         ( 0, 0,10,14, 0, 0));

type
  Dinh                                =    array[1..15] of byte;
  mangI                                =    array[1..32767] of integer;
  CapSo                                =    record c1, c2: byte end;
  mangC                                =    array[1..32767] of CapSo;

var
  f,g                                  :    text;
  a                                    :    Dinh;      {Quản lý một trạng thái bàn cờ}
```

xp	:	integer; {Trạng thái (đỉnh) xuất phát}
ok	:	boolean;
Q,	:	\wedge mangI; {Hàng đợi}
qf, ql	:	integer; {Biến đầu và cuối hàng đợi}
Tr	:	\wedge mangI; {Lưu vết hành trình}
kt	:	integer; {Trạng thái (đỉnh) kết thúc}
c	:	\wedge MangC;
slx	:	integer;

Procedure cap_phat; {Xin cấp phát bộ nhớ}
begin New(q); New(Tr); New(c); end;

Procedure initQ; {Khởi trị hàng đợi}
begin qf:=1; ql:=1; end;

Procedure put(u: integer); {Nạp đỉnh u vào hàng đợi}
begin q $^$ [ql]:=u; inc(ql); end;

function get: integer; {Hàm lấy đỉnh ở cuối hàng đợi}
begin Get:=q $^$ [qf]; inc(qf); end;

function Qempty: boolean; {Hàm kiểm tra Hàng đợi đã rỗng chưa}
begin Qempty:=(qf=ql); end;

function doiso(var a:Dinh):integer; {Hàm đổi trạng thái a ra đỉnh (số)}
var u, i: integer;
begin
 u:=0;
 for i:=1 to 15 do u:=u*2+a[i];
 doiso:=u;
end;

Procedure doidinh(u:integer;var a:Dinh); {Đổi đỉnh u ra trạng thái a}
var i: integer;
begin
 for i:=15 downto 1 do begin
 a[i]:=u mod 2;
 u:=u div 2;
 end;
end;

Procedure doc_input; {Đọc trạng thái ban đầu từ tệp input}
var k, i: integer;
begin
 assign(f,fi); reset(f);

```

readln(f,k); {Ô k là ô rỗng}
close(f);
for i:=1 to 15 do a[i]:=1; {Tạo mảng a quản lý trạng thái: các ô đều có số}
a[k]:=0; {riêng ô k không chứa số}
xp:=DoiSo(a); {Chuyển trạng thái ban đầu thành đỉnh xuất phát là số xp}
end;
{Tìm đỉnh v kề với đỉnh u. Đỉnh u ứng với trạng thái a có ô i chứa số, đỉnh v nhận được
sau khi di chuyển ô i theo hướng k tới ô rỗng}

```

```

Procedure Ke(u,i,k: integer; var v: integer; var r: byte);
var j,j1: integer;
begin
  v:=0; {Khởi trị v}
  DoiDinh(u,a); {Chuyển đỉnh u thành trạng thái a}
  if a[i]=0 then exit; {nếu ô i không chứa số thì không di chuyển được, thoát}
  j:=delta[i,k]; {j là ô liền kề ô i theo hướng k}
  if (j=0) or (a[j]=0) then exit; {ô j ngoài lưới hoặc là ô rỗng thì thoát}
  while (j>0) and (a[j]=1) do j:=delta[j,k]; {cho tới ô rỗng}
  if j>0 then begin {có ô j rỗng thì thực hiện di chuyển}
    j1:=i;
    while j1<>j do begin
      a[j1]:=0; {loại các số ở các ô đi qua, kể từ ô i}
      j1:=delta[j1,k];
    end;
    a[j1]:=1; {xác nhận ô di chuyển tới là có số}
    r:=j1; {r là ô có số mới tạo sau di chuyển}
    v:=DoiSo(a); {chuyển trạng thái a sau di chuyển thành đỉnh là số v}
    exit;
  end;
end;

function KetThuc(v: integer): boolean; {Kiểm tra đỉnh v có là đỉnh kết
thức hay chưa}
var dem: integer;
begin
  KetThuc:=false;
  if v=0 then exit; {v=0 không là đỉnh kết thúc}
  dem:=0; {đếm số lượng bit 1 của số v}
  while v>0 do begin
    dem:=dem+v mod 2;
    v:=v div 2;
    if dem>1 then exit; {hơn một ô còn chứa số nên v không là đỉnh kết thúc}
  end;
  KetThuc:=true; {có đúng một ô chứa số nên v là đỉnh kết thúc}
end;

Procedure Loang;

```

```

var i,k,u,v: integer;
    l: byte;
begin
    ok:=true;
    InitQ; {Khởi trị hàng đợi}
    for i:=1 to 32727 do Tr^[i]:=0; {Khởi trị mảng vết}
    Put (xp); {Nạp đỉnh xuất phát vào cuối hàng đợi}
    Tr^[xp]:=-1; {Đỉnh trước đỉnh xuất phát coi là -1}
    repeat
        u:=Get; {Lấy đỉnh ở đầu hàng đợi}
        for i:=1 to 15 do {Đề cử mọi ô i}
            for k:=1 to 6 do begin {Di chuyển theo mọi hướng k}
                Ke(u,i,k,v,l); {Bước nhảy ô i theo hướng k tới ô l, biến đỉnh u thành v}
                if v>0 then {Nếu đỉnh v là một đỉnh của đồ thị}
                    if Tr^[v]=0 then begin {và đỉnh v chưa thăm thì}
                        Put (v); {Nạp v vào cuối hàng đợi}
                        Tr^[v]:=u; {Lưu vết hành trình: trước đỉnh v là đỉnh u}
                        C^[v].c1:=i; {Lưu bước nhảy tạo ra đỉnh v vào C^[v]}
                        C^[v].c2:=l;
                        If KetThuc(v) then begin {Nếu v là đỉnh kết thúc thì}
                            kt:=v; {Lưu lại đỉnh kết thúc và thoát, khi đó OK=True}
                            exit;
                        end;
                    end;
                end;
            until Qempty; {Cho đến khi hàng đợi rỗng}
            ok:=false; {Xác nhận không thể đi tới đỉnh kết thúc}
        end;
Procedure tim_duong; {Tìm hành trình của các bước nhảy liên tiếp}
var v: integer;
begin
    slx:=0; {Khởi trị số lượng bước nhảy}
    v:=kt; {Lần ngược hành trình từ đỉnh kết thúc về đỉnh xuất phát}
    repeat
        inc(slx); {Tăng thêm một bước nhảy}
        q^[slx]:=v; {Lưu đỉnh v}
        v:=tr^[v]; {về đỉnh liền trước}
    until v=-1; {về tới -1 coi là đỉnh trước đỉnh xuất phát}
end;
Procedure ghi_out;
var i,v: integer;
begin
    assign(g,fo); rewrite(g);
    if not ok then writeln(g,-1) {Nếu không tới đích thì ghi -1}

```



```

else begin {còn không thì}
    tìm_duong; {Tìm hành trình}
    writeln(g, slx-1); {Ghi vào tệp output số bước nhảy}
    for i:=slx-1 downto 1 do begin {ghi theo hành trình thuận}
        v:=q^[i]; {v là đỉnh nhảy tới}
        writeln(g, c^[v].c1, ' ', c^[v].c2); {ghi bước nhảy tương ứng}
    end;
end;
close(g);
end;
BEGIN
    Cap_phat;
    doc_input;
    loang;
    ghi_out;
END.

```

Bài 21. Brick

Có một loại gạch hình vuông, mỗi viên được phân đôi theo đường chéo tạo thành một nửa màu đen còn nửa kia màu trắng. Người ta lát một diện tích hình vuông kích thước $N \times N$ bởi loại gạch nói trên. Các hình được tạo bởi các tam giác cùng màu kề cạnh và xung quanh có màu khác. Ví dụ, trên Hình 1 có 10 hình màu đen, trong đó có 1 hình vuông, 2 hình bình hành, 1 hình thang, 4 hình tam giác và 2 hình khác.



Biểu diễn diện tích lát gạch bởi một mảng $A[i,j]$ ($i=1, \dots, N; j=1, \dots, N$) các hàng được đánh số từ trên xuống dưới bắt đầu từ 1, các cột được đánh số từ trái qua phải bắt đầu từ 1, trong đó $A[i,j]$ có giá trị là 0,1,2,3 nếu viên gạch ở hàng i , cột j có màu đen tương ứng nằm ở nửa trên đường chéo

chính, nửa dưới đường chéo chính, nửa trên đường chéo phụ, nửa dưới đường chéo phụ (Hình 2). Chẳng hạn ma trận A biểu diễn sân ở hình 1 là:

```

3 1 1 3 1
0 2 3 2 2
3 0 1 0 1
0 3 0 3 1
1 2 0 3 2

```

Hãy đếm số lượng các hình màu đen có dạng hình tam giác, hình vuông, hình bình hành, hình thang và các hình khác.

Dữ liệu vào: trong file văn bản BRICK.IN:

Dòng đầu tiên ghi số nguyên dương N ($N \leq 50$)

Dòng thứ i trong N dòng tiếp theo ghi N số $A[i,1], \dots, A[i,N]$. Các số trên một dòng cách nhau ít nhất một dấu cách.

Kết quả: Đưa ra file văn bản BRICK.OUT gồm 6 dòng lần lượt là: Số lượng các hình màu đen, số lượng các tam giác màu đen, số lượng các hình vuông màu đen, số lượng các hình bình hành màu đen, số lượng các hình thang màu đen, số lượng các hình khác màu đen.

Ví dụ:

BRICK.INP	BRICK.OUT
5	10
3 1 1 3 1	4
0 2 3 2 2	1
3 0 1 0 1	2
0 3 0 3 1	1
1 2 0 3 2	2

Gợi ý. Qui định cạnh phía Đông, phía Bắc, phía Tây, phía Nam của mỗi viên gạch lần lượt gọi là cạnh thuộc hướng 1, 2, 3 và 4. Chúng ta xây dựng mảng $mau[0..3, 1..4]$ với ý nghĩa: $mau[l, h] = 1$ nếu cạnh ở hướng h của gạch loại l thuộc nửa trắng của gạch, $mau[l, h] = 0$ nếu cạnh ở hướng h của gạch loại l thuộc nửa đen của gạch. Vậy màu các cạnh có thể đưa vào mảng hằng hai chiều:

`const`

```

mau array[0..3, 1..4] of integer = ((0, 0, 1, 1),
                                     (1, 1, 0, 0),
                                     (1, 0, 0, 1),
                                     (0, 1, 1, 0));

```



Ta cần tìm các tam giác đen kề cạnh tạo thành các vùng liên thông. Sau đó căn cứ vào đặc điểm của mỗi vùng đen này mà khẳng định đó là hình loại gì. Rõ ràng hai viên gạch có thể ghép (nửa đen) với nhau khi màu các cạnh trên cùng một hướng của hai viên gạch này phải khác nhau; nói cách khác nếu chúng cùng màu thì không coi là kề nhau trên cùng một vùng liên thông..

Nhận thấy mỗi vùng đen hình vuông, hình bình hành, hình thang, hình tam giác chỉ có tối đa là 4 viên gạch, do đó có thể chọn 1 số nguyên 4 bit đại diện cho mỗi vùng. Nếu có loại gạch số i ($i=0,1,2,3$) tham gia thì bit i trong số nguyên này nhận giá trị 1. Ví dụ: vùng hình vuông có đủ 4 loại gạch tham gia nên số nguyên đại diện vùng này viết dưới dạng nhị phân là $1111_{(2)}$ (dạng thập phân là 15). Vùng hình bình hành chỉ có hai loại gạch (số 0 ghép với số 1, hoặc số 2 ghép với số 3) nên được đại diện bởi số nguyên 3 hoặc 12. Vùng hình thang cũng chỉ có hai loại gạch (hai viên số 0 và một viên số 1, hoặc một viên số 2 và hai viên số 3) nên cũng được đại diện bởi số nguyên 3 hoặc 12.

Mặt khác, còn phân biệt hình dáng các vùng bởi số lượng gạch tạo ra chúng. Vùng tam giác chỉ là 1 hoặc 2, vùng hình thang có 3, hình bình hành có 2 (viên gạch).

Ngoài ra để xác định chính xác hình vuông, còn thêm dấu hiệu: cột biên phải và cột biên trái của vùng chỉ chênh nhau 1, dòng biên dưới và dòng biên trên của vùng chỉ chênh nhau 1. Ví dụ vùng nêu ở hình vẽ bên cũng có số đại diện là 15 (vì đủ 4 loại gạch) nhưng không có đặc điểm các cột biên, dòng biên chênh nhau 1.



Chương trình

```
const
  fi='BRICK.INP';
  fo='BRICK.OUT';
  maxN=50;
  mau: array[0..3,1..4] of integer = ((0,0,1,1), {màu cạnh}
                                         (1,1,0,0),
                                         (1,0,0,1),
```

```

                                (0,1,1,0));
dh: array[1..4] of integer=(0,-1,0,1); {số gia về dòng}
dc: array[1..4] of integer=(1,0,-1,0); {số gia về cột}
var
  f, g: text;
  N: integer; {Kích thước sân hình vuông}
  a: array[1..maxN,1..maxN] of byte; {Mảng sân lát gạch}
  lt: array[1..maxN,1..maxN] of integer;
  loai,      {Loại của một vùng đen liên thông}
  Sd,        {Số gạch dùng trong một vùng đen}
  tren, duoi, {Số hiệu dòng trên, dưới của vùng đen}
  trai, phai : {Số hiệu cột trái và phải của vùng đen}
                                array[1..maxN*maxN] of integer;
  solt: integer; {Số vùng liên thông}
  Q: array[1..2,1..maxN*maxN] of integer; {Hàng đợi}
  qf,ql: integer; {Biến đầu và biến cuối hàng đợi}
  tamgiac,hinhbh,hinhthang,hinhvuong:integer; {Số lượng các hình}
Procedure read_input;
var i,j: integer;
begin
  assign(f,fi); reset(f);
  readln(f,N); {Kích thước sân vuông}
  for i:=1 to N do
    for j:=1 to N do read(f,a[i,j]); {Đọc các viên gạch lát trên sân}
  close(f);
end;
Procedure initQ; {Khởi trị hàng đợi}
begin  qf:=1;  ql:=1; end;
Procedure put(i,j: integer); {Nạp viên gạch (i,j) vào hàng đợi}
begin
  q[1,ql]:=i; {tọa độ dòng của viên gạch}
  q[2,ql]:=j; {tọa độ cột của viên gạch}
  inc(ql);
end;
Procedure get(var i,j: integer); {Lấy một viên gạch ở đầu hàng đợi}
begin
  i:=q[1,qf];  j:=q[2,qf];  inc(qf); end;
function Qempty: boolean; {Hàm kiểm tra Hàng đợi rỗng}
begin  Qempty:=(qf=ql); end;
Procedure Ke(i,j,k: integer; var u,v: integer; var ok:
boolean); {Tìm gạch (u,v) có phần đen kề với gạch (i,j) theo hướng k, tìm được thì OK}
begin
  ok:=false;
  if Mau[a[i,j],k]=1 then exit; {theo hướng k, cạnh trắng thì thoát}

```

```

    {ô (u,v) là một bên cạnh theo hướng k}
    u:=i+dh[k];
    v:=j+dc[k];
    if (u<1) or (u>N) or (v<1) or (v>N) then exit; {ngoài sân}
    if Mau[a[u,v], 1+(1+k mod 4) mod 4]=1 {theo hướng đối của hướng k
gạch (u,v) có cạnh màu trắng nên hai nửa đen của gạch (i,j) và gạch (u,v) không kề cạnh}
    then exit;
    ok:=true;
end;
Procedure Loang(i,j: integer); {Loang bắt đầu từ ô (i,j)}
var k,u,v: integer;
    ok: boolean;
begin
    initQ; {Khởi trị hàng đợi}
    put(i,j); {Nạp phần tử (i,j) vào hàng đợi}
    lt[i,j]:=solt; {đánh số vùng liên thông cho gạch (i,j)}
    repeat {Thực hiện lặp tìm các phần đen kề cạnh}
        get(i,j); {Lấy phần tử đầu hàng đợi là gạch (i,j)}
        for k:=1 to 4 do begin {Đề cử 4 hướng}
            ke(i,j,k,u,v,ok); {Tìm gạch (u,v) kề bên theo hướng k}
            if ok and (lt[u,v]=0) then begin {(u,v) có phần đen kề với miền
đang xét và (u,v) chưa thuộc vùng liên thông nào}
                put(u,v); {Nạp (u,v) vào hàng đợi}
                lt[u,v]:=solt; {đánh số vùng liên thông cho gạch (u,v)}
                if v<Trai[solt] then Trai[solt]:=v; {biên trái của vùng}
                if v>Phai[solt] then Phai[solt]:=v; {biên phải của vùng}
                if u<Tren[solt] then Tren[solt]:=u; {biên trên của vùng}
                if u>Duoai[solt] then Duoai[solt]:=u; {biên dưới của vùng}
                SD[solt]:=SD[solt]+1; {số lượng gạch tham gia vào vùng}
                loai[solt]:=loai[solt] or (1 shl a[u,v]);
                {thêm loại gạch góp phần tạo ra đặc điểm của loại vùng}
            end;
        end;
    until Qempty; {Hàng đợi rỗng}
end;
Procedure TimTPLT; {Tìm số thành phần liên thông}
var i,j: integer;
begin
    fillchar(LT, sizeof(LT), 0); {Khởi trị mảng đánh số các vùng liên thông}
    solt:=0; {Khởi trị số thành phần liên thông}
    for i:=1 to N do {Duyệt các ô (i,j)}
        for j:=1 to N do
            if LT[i,j]=0 then begin {(ô (i,j) chưa tham gia vùng liên thông nào}
                inc(solt); {tăng số lượng vùng liên thông}
            end;
        end;
    end;
end;

```

```

    trai[solt]:=j; phai[solt]:=j; {Biên trái và phải là cột j}
    tren[solt]:=i; duoi[solt]:=i; {Biên trên và biên dưới là dòng i}
    SD[solt]:=1; {Khởi trị số gạch tham gia vào vùng liên thông này}
    loai[solt]:=loai[solt] or (1 shl a[i,j]); {bổ sung vào loại vùng}
    loang(i,j); {Tìm các phần đen khác kề với phần đen của (i,j)}
end;
end;
Procedure DemHinh; {Dựa vào đặc điểm của loại vùng mà suy ra hình dáng của nó}
var i: integer;
begin
    tamgiac:=0; {Khởi trị số lượng tam giác}
    hinhvuong:=0; {Khởi trị số lượng hình vuông}
    hinhbh:=0; {Khởi trị số lượng hình bình hành}
    hinhthang:=0; {Khởi trị số lượng hình thang}
    for i:=1 to solt do begin {xét lần lượt các loại vùng liên thông}
        if (loai[i]=3) or (loai[i]=12) then begin {hình thang, h bình hành}
            if odd(SD[i]) then hinhthang:=hinhthang+1 {3 gạch: thang}
            else hinhbh:=hinhbh+1; {2 gạch: h bình hành}
            continue;
        end;
        if (SD[i]=1) or (SD[i]=2) then begin {còn lại 1, 2 gạch: tam giác}
            tamgiac:=tamgiac+1;
            continue;
        end;
        if (loai[i]=15) and (phai[i]-trai[i]=1)
        and (duoi[i]-tren[i]=1) then begin {hình vuông}
            hinhvuong:=hinhvuong+1;
            continue
        end;
    end;
end;
Procedure write_output;
begin
    assign(g,fo); rewrite(g);
    writeln(g,solt);
    writeln(g,tamgiac);
    writeln(g,hinhvuong);
    writeln(g,hinhbh);
    writeln(g,hinhthang);
    writeln(g,solt-(tamgiac+hinhvuong+hinhbh+hinhthang));
    close(g);
end;
BEGIN
    read_input;
    timTPLT;

```

```
demhinh;  
write_output;  
END.
```

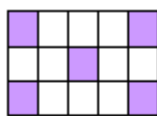
Bài 22. Bridges - Xây dựng cầu

Hội đồng thành phố New Altonville đặt kế hoạch xây dựng những chiếc cầu nối liền tất cả các khu đã xây dựng của thành phố với nhau sao cho mọi người có thể đi từ khu này tới khu khác mà không phải ra ngoài mảnh đất của New Altonville. Bạn hãy viết một chương trình tìm ra một mô hình xây cầu tối ưu.

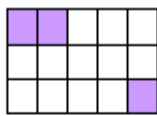
Mảnh đất New Altonville nằm trên một lưới các ô vuông. Mỗi khu đã xây dựng chiếm một hoặc nhiều ô vuông. Hai ô vuông có đỉnh chung được xem là cùng một khu và không cần cầu. Các cầu chỉ xây trên các đường lưới dọc các cạnh của các ô vuông và mỗi cầu đều phải xây trên một đường thẳng nối đúng hai khu với nhau.

Cho tập các khu đã xây dựng, bạn cần tìm số cầu ít nhất nối tất cả các khu. Nếu không tìm được, hãy tìm nghiệm sao cho *số các khu không nối được với nhau nữa là nhỏ nhất*. Trong các nghiệm có cùng số lượng cầu, chọn nghiệm có tổng độ dài các cầu nhỏ nhất, tổng này là một bội số của kích thước ô lưới. Hai cầu có thể ngang qua nhau, nhưng trong trường hợp này chúng được xem là tách rời nhau không có thể nối cầu này sang cầu kia.

Hình vẽ sau đây minh họa 4 mô hình cầu của 4 thành phố. Thành phố 1 gồm 5 khu, có thể nối bằng 4 cầu, tổng độ dài bằng 4. Thành phố 2 không có phương án xây cầu do không có khu nào nằm trên cùng một đường lưới. Thành phố 3 không cần cầu vì chỉ có 1 khu. Thành phố 4, phương án tốt nhất là chỉ cần 1 cầu nối hai khu, thành phố tách thành hai nhóm (một nhóm chứa hai khu, nhóm còn lại chỉ chứa 1 khu).



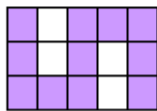
Thành phố 1



Thành phố 2



Thành phố 4



Thành phố 3

Dữ liệu vào từ tệp BRIDGES.IN mô tả một vài hình chữ nhật (thành phố). Mỗi thành phố được miêu tả bởi một dòng chứa hai số nguyên r và c , tương ứng là kích thước thành phố theo hướng trục Bắc-Nam và Đông-Tây đo theo độ dài của lưới ($1 \leq r \leq 50$ và $1 \leq c \leq 50$). Tiếp theo đúng n dòng mỗi dòng gồm có dấu thăng (“#”) và dấu chấm (“.”). Mỗi ký tự tương ứng với một ô vuông của lưới. Dấu thăng là đất thuộc khu đã xây dựng, dấu chấm là đất chưa xây dựng. File **Dữ liệu vào** kết thúc bằng một dòng chứa hai số 0.

Kết quả ghi ra file BRIDGES.OUT

Với mỗi thành phố được miêu tả, in ra hai hoặc ba dòng như sau: Dòng thứ nhất là tên thành phố (City 1 hoặc City 2,...). Nếu thành phố có ít hơn hai khu thì in ra câu “No bridge are needed” (Không cần cầu). Nếu thành phố có hơn hai khu nhưng không thể nối chúng bằng các cầu thì in câu “No bridges are possible” (Không thể nối cầu thành một khu). Mặt khác dòng thứ hai là “N bridges of total length L” (N cầu nối dài là L); N là số cầu và L là tổng độ dài của các cầu trong nghiệm tốt nhất. (Nếu $N=1$ thì thay từ “bridges” thành “bridge”). Nếu nghiệm chia thành phố thành nhiều nhóm thì in trong dòng thứ ba số nhóm phân chia: in câu “M disconnected groups” (M nhóm không nối được với nhau nữa). Giữa hai test là dòng trống.

BRIDGES.IN	BRIDGES.OUT
3 5 #...# ..#.. #...#	City 1 4 bridges of total length 4
3 5	City 2 No bridgers are possible

##...	2 disconnected groups
.....	
....#	City 3
3 5	No bridgers are needed
#####	
###.##	City 4
####.	1 bridge of total length 1
3 5	2 disconnected groups
###..	
.....	
....#	
0 0	

Gợi ý. Dùng kỹ thuật tìm kiếm theo chiều rộng để gộp các ô vuông liên thông nhau (có chung 1 đỉnh ô vuông) thành một nhóm. Sau đó, coi mỗi nhóm là một đỉnh của đồ thị vô hướng đưa về bài toán tìm cây khung ngắn nhất trên đồ thị này.

Trọng số trên các cạnh của đồ thị này (mỗi cạnh nối hai đỉnh đồ thị tương ứng với hai nhóm) chính là độ dài cầu ngắn nhất nối hai nhóm. Tuy nhiên không cần xây dựng trọng số cụ thể cho mọi cạnh.

Trong quá trình xây dựng cây khung ngắn nhất, chỉ khi xây dựng nhãn cho các đỉnh đồ thị chúng ta mới cần tìm độ dài cầu ngắn nhất nối một đỉnh đã thuộc cây khung đang hình thành tới một đỉnh khác chưa thuộc cây khung này. Lưu ý: mọi ô vuông thuộc cùng một nhóm thì có cùng một nhãn. Các ô vuông nào của đỉnh đã thuộc cây khung thì được đánh dấu là đã xét, các ô vuông nào của đỉnh chưa thuộc cây khung thì được đánh dấu là chưa xét.

Chương trình.

```
const  fi   = 'bridges.in';
        fo   = 'bridges.out';
        maxn = 52;

var     f,g   : text;
        r,c   : byte;
        a     : array[0..maxn+1,0..maxn+1] of char;
        cx    : array[1..maxn+1,1..maxn+1] of boolean;
        qd,qc : array[1..maxn*maxn] of byte;
        kc    : array[1..maxn+1,1..maxn+1] of word; {nhãn đỉnh đồ thị}
        dau, cuoi,
                                {hai biến đầu và cuối hàng đợi}
```

```

sm,          {số nhóm- số đỉnh của đồ thị}
ts,          {tổng độ dài các cầu cần bắc}
dem,
scau :integer; {số lượng cầu đã bắc}

```

Procedure mofile;

begin

```
    assign(f,fi);reset(f);
```

```
    assign(g,fo);rewrite(g);
```

end;

Procedure dongfile;

begin

```
    close(f); close(g);
```

end;

Procedure nhap;

```
var i,j:byte;
```

begin

```
    readln(f,r,c); {đọc hàng và cột của hình chữ nhật lưới ô vuông}
```

```
    fillchar(a,sizeof(a),0);
```

```
    for i:=1 to r do begin
```

```
        for j:=1 to c do read(f,a[i,j]); {đọc các ô vuông trong lưới}
```

```
        readln(f);
```

```
    end;
```

end;

Procedure bfs(i,j:byte); {Loang tìm các ô cùng nhóm với ô (i,j)}

```
var ldd,ldc,il,jl:integer;
```

begin

```
    qd[1]:=i;qc[1]:=j; { nạp (i,j) vào đầu hàng đợi}
```

```
    dau:=1;cuoi:=1; {khởi trị biến đầu và cuối hàng đợi}
```

```
    cx[i,j]:=false; {xác nhận đã xét ô (i,j)}
```

```
    while dau<=cuoi do begin {điều kiện còn loang}
```

```
        i:=qd[dau]; {lấy ô (i,j) ở đầu hàng đợi}
```

```
        j:=qc[dau];
```

```
        inc(dau); {dịch chuyển đầu hàng đợi lên vị trí mới}
```

```
        for ldd:=-1 to 1 do {duyệt 8 ô (il,jl) chung đỉnh với (i,j)}
```

```
            for ldc:=-1 to 1 do begin
```

```
                il:=i+ldd; jl:=j+ldc;
```

```
                if (a[il,jl]='#') and(cx[il,jl]) then begin {điều kiện ô (il,jl)
```

cùng nhóm với ô (i,j)}

```
                    inc(cuoi); {dịch chuyển cuối hàng đợi lên vị trí mới}
```

```
                    qd[il,jl]:=il; { nạp ô (il,jl) vào cuối hàng đợi}
```

```
                    qc[il,jl]:=jl;
```

```
                    cx[il,jl]:=false; {xác nhận đã xét ô (il,jl)}
```

```
                    kc[il,jl]:=kc[i,j]; {coi các ô cùng nhóm có cùng một nhãn}
```

```
                end;
```

```
            end;
```

```

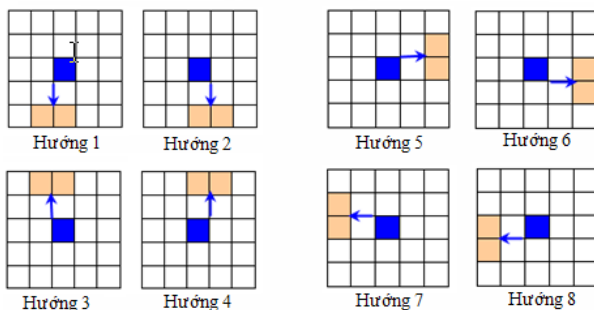
end;
end;
Procedure timmin(var i1,j1:byte); {tìm một ô vuông (i1,j1) thuộc đỉnh chưa nạp vào cây khung và nhãn nhỏ nhất}
var i,j:byte;
    nn:longint;
begin
    nn:=65535;i1:=0;
    for i:=1 to r do
        for j:=1 to c do
            if (a[i,j]='#') and cx[i,j] and (kc[i,j]<nn) then begin
                nn:=kc[i,j];
                i1:=i; j1:=j;
            end;
        end;
    if i1>0 then inc(ts,nn); {cầu mới dài là nn, nên tổng độ dài cầu thêm nn}
end;
Procedure di(i,j,ldd,ldc,oked,okec:integer); {tìm độ dài của cầu theo một hướng đi. Đồng thời sửa lại nhãn của các ô vuông thuộc đỉnh kề (cầu lao tới) chưa thuộc cây khung}
var dem:integer;
begin
    dem:=-1;
    repeat
        inc(dem); {độ dài của một cầu đang lao theo hướng này}
        i:=i+ldd; j:=j+ldc;
    until (i<1) or (j<1) or (i>r) or (j>c) {vượt qua biên hình chữ nhật}
        or (a[i,j]='#') {tới ô vuông đất đã xây dựng ở một bên của hướng lao cầu}
        or (a[i+oked,j+okec]='#') {tới ô ở bên còn lại của hướng lao cầu}
    if (a[i,j]='#') and cx[i,j] and (kc[i,j]>dem) then
        kc[i,j]:=dem; {sửa lại nhãn của ô(i,j) thuộc đỉnh kề chưa xét}
        i:=i+oked; j:=j+okec;
    if (a[i,j]='#') and cx[i,j] and (kc[i,j]>dem) then
        kc[i,j]:=dem; {sửa lại nhãn của ô(i,j) thuộc đỉnh kề chưa xét}
end;
Procedure thucau; {Thử tìm cầu ngắn nhất nối nhóm có (i,j) với nhóm khác}
var tt:integer;
begin
    for tt:=1 to cuoi do begin
        di(qd[tt],qc[tt], 1, 0, 0,-1); {lao cầu theo hướng 1}
        di(qd[tt],qc[tt], 1, 0, 0, 1); {lao cầu theo hướng 2}
        di(qd[tt],qc[tt],-1, 0, 0,-1); {lao cầu theo hướng 3}
        di(qd[tt],qc[tt],-1, 0, 0, 1); {lao cầu theo hướng 4}
        di(qd[tt],qc[tt], 0, 1,-1, 0); {lao cầu theo hướng 5}
        di(qd[tt],qc[tt], 0, 1, 1, 0); {lao cầu theo hướng 6}
        di(qd[tt],qc[tt], 0,-1,-1, 0); {lao cầu theo hướng 7}
    end;
end;

```

```

    di(qd[tt],qc[tt], 0,-1, 1, 0); {lao cầu theo hướng 8}
end;
end;

```



Procedure xuli; {Áp dụng thuật toán tìm cây khung ngắn nhất}

```

var i,j,il,jl:byte;

```

```

begin

```

```

    fillchar(cx,sizeof(cx),true); {các đỉnh chưa kết nạp vào cây khung}

```

```

    fillchar(kc,sizeof(kc),255); {khởi trị nhãn các đỉnh coi là vô cùng}

```

```

    ts:=0;scau:=0; {khởi trị tổng độ dài cầu, số cầu }

```

```

    sm:=0; {khởi trị số nhóm – số đỉnh đồ thị}

```

```

    for i:=1 to r do

```

```

        for j:=1 to c do

```

```

            if cx[i,j] and(a[i,j]='#') then begin

```

```

                inc(sm); {tăng số đỉnh}

```

```

                il:=i;jl:=j; {ô (il,jl) thuộc đỉnh vừa thấy}

```

```

                repeat

```

```

                    bfs(il,jl); {Loang tìm các ô vuông cùng nhóm}

```

```

                    thucau; {tìm nhãn sửa và sửa nhãn các đỉnh chưa vào cây khung}

```

```

                    timmin(il,jl); {tìm ô (il,jl) thuộc đỉnh kề gần cây khung nhất}

```

```

                    if il>0 then inc(scau); {tăng số cầu}

```

```

                until il=0; {cho đến khi không còn ô vuông thuộc đỉnh nào}

```

```

            end;

```

```

    end;

```

Procedure xuat; {ghi kết quả vào tệp output}

```

begin

```

```

    if dem>1 then writeln(g);

```

```

    writeln(g,'City ',dem);

```

```

    if (sm=1)and(ts=0) then writeln(g,'No bridges are needed')

```

```

    else begin

```

```

        if(ts=0) then writeln(g,'No bridges are possible')

```

```

        else begin

```

```

            if scau>1 then writeln(g,scau,' bridges of total length ',ts)

```

```

            else writeln(g,scau,' bridge of total length ',ts);

```

```

        end;

```

```

        if sm>1 then writeln(g,sm,' disconnected groups');

```

```

end;
end;
BEGIN
  mofile;
  dem:=0; {đếm số test trong tệp input}
  while not seekeof(f) do begin
    inc(dem); nhập;
    if (r=0)and(c=0) then break; { hết Dữ liệu vào, thoát}
    xuli; xuất;
  end;
  dongfile;
END.

```

Bài 23. Đặt trạm cứu hỏa.

Một mạng giao thông có N nút đánh số từ 1 đến N , giữa một số cặp nút có đường đi hai chiều. Mạng này liên thông. Hiện nay toàn bộ hệ thống đường rất xấu.

Cần chọn một nút đặt trạm cứu hỏa và một số đoạn đường để nâng cấp sao cho với hệ thống chỉ gồm những đoạn đường được nâng cấp, từ trạm cứu hỏa đến mỗi nút có đúng một đường đi và khoảng cách từ nút xa trạm nhất đến trạm là nhỏ nhất có thể được.

Dữ liệu vào được cho bởi file văn bản **CH.INP** trong đó dòng thứ nhất ghi số N ($N \leq 200$), trong một số dòng tiếp theo, mỗi dòng ghi ba số nguyên U, V, W với ý nghĩa có đường đi hai chiều nối nút U với nút V dài là W ($W \leq 10000$)

Kết quả ghi ra file văn bản **CH.OUT** như sau : dòng thứ nhất ghi tên nút đặt trạm cứu hỏa, dòng thứ hai ghi khoảng cách từ nút xa nhất đến trạm, tiếp theo là một số dòng, mỗi dòng ghi hai nút đầu mút của một đoạn đường nâng cấp.

Ví dụ.

CH . INP	CH . OUT
5	4
1 2 50	25
1 3 30	1 5
1 4 100	2 4
1 5 10	3 2

2 3 5	5 4
2 4 20	
3 4 50	
4 5 10	

Gợi ý.

a) Thực hiện N lần, mỗi lần thực hiện các bước sau :

a1. Lần lượt chọn một đỉnh xuất phát là i ($i=1,2, \dots, N$).

a2. Thực hiện tương tự Dijkstra, nhưng sửa nhãn N lần, mỗi lần đi xa đỉnh xuất phát thêm một đỉnh theo con đường ngắn nhất tới các đỉnh chưa thăm. Do vậy nhãn của đỉnh chưa thăm cuối cùng chính là khoảng cách xa trạm xuất phát nhất, đi theo con đường ngắn nhất.

a3. So nhãn của đỉnh chưa thăm cuối cùng với phương án tối ưu để xác nhận lại phương án tối ưu (trạm xuất phát là trạm nào? độ dài con đường ngắn nhất từ trạm xuất phát tới thành phố xa nhất là bao nhiêu? Vết đi của con đường ngắn nhất qua N thành phố).

b) Dựa vào phương án tối ưu đã lưu, xuất ra kết quả.

Chương trình

```
const fi      = 'ch.inp';
      fo      = 'ch.out' ;
      vc = 10000000000;
type  dd      = array[0..200] of byte;
      dong    = array[1..200] of longint;
var    f       : text;
       w       : array[1..200] of ^dong; {ma trận trọng số}
      chưa    : array[1..200] of boolean; {đánh dấu đỉnh còn tự do}
      tr,     {lưu vết đường đi ngắn nhất}
      truoc   : dd; {lưu vết đường đi ngắn nhất trong phương án tối ưu}
      max,    {độ dài đường đi ngắn nhất qua N đỉnh}
      mm: longint; {độ dài nhất trong các độ dài đường đi ngắn nhất qua N đỉnh}
      d       : array[1..200] of longint; {nhãn các đỉnh}
      m,n,ch,
      chon    : byte; {đỉnh được chọn đặt trạm cứu hỏa}
```

```
Procedure nhap_input;
var  i,j:byte;x:longint;
begin
```

```

assign(f,fi);reset(f);
readln(f,n);
for i:=1 to n do
for j:=1 to n do w[i]^j:=vc; {Khởi trị mảng trọng số}
while not(seekeof(f)) do begin {đọc mảng trọng số từ tệp input}
    readln(f,i,j,x);
    w[i]^j:=x;
    w[j]^i:=x;
end;
close(f);
end;
Procedure Dijkstra(ch : byte); {Thực hiện tương tự Dijkstra, từ đỉnh ch,
sửa nhãn đủ N lần để xây dựng nhãn các đỉnh}
var dem,last,j:byte;min:longint;
begin
    for j:=1 to n do begin {Khởi trị mảng nhãn d và mảng đánh dấu chua}
        chua[j]:=true;
        d[j]:=vc;
    end;
    chua[ch]:=false; {Đỉnh ch đã cố định nhãn}
    d[ch]:=0; {Nhãn đỉnh ch là 0}
    last:=ch; {Đỉnh cuối cùng của đường đi ngắn nhất bây giờ chính là đỉnh ch}
    dem:=1; {số đỉnh đã cố định nhãn}
    fillchar(tr, sizeof(tr), 0); {Khởi trị mảng lưu vết đường đi}
    while dem<n do begin {Còn thực hiện khi chưa đủ N đỉnh được cố định nhãn}
        for j:=1 to n do {Sửa nhãn các đỉnh sau khi đỉnh last được cố định nhãn}
            if chua[j] and (w[last]^j<vc)
            and (d[j]>d[last] + w[last]^j) then begin
                d[j]:=d[last]+w[last]^j;
                tr[j]:= last;
            end;
        min:=1500000000; {Tìm đỉnh còn tự do có nhãn nhỏ nhất gán cho last}
        for j:=1 to n do
            if chua[j] and (d[j]<min) then begin
                min:=d[j];last:=j;
            end;
        chua[last]:=false; {Cố định đỉnh last là đỉnh tự do có nhãn nhỏ nhất}
        inc(dem); {số đỉnh đã được cố định nhãn}
        if d[last]>mm then exit; {điều kiện cần để kéo dài tiếp đường đi}
    end;
    max:=d[last]; {độ xa nhất trong các đường đi ngắn nhất bắt đầu từ đỉnh ch}
    if max<mm then begin {so và lưu phương án tối ưu}
        mm:=max; chon:=ch;
        truoc := tr;
    end;
end;

```

```

end;
Procedure ghi_out;
var i : byte;
begin
    assign(f,fo);rewrite(f);
    writeln(f,chon); {ghi đỉnh được chọn đặt trạm cứu hỏa}
    writeln(f,mm); {ghi độ dài xa nhất đi tới trạm cứu hỏa}
    for i:=1 to n do {ghi lại các con đường cần nâng cấp}
        if i<>chon then writeln(f,i:5,truoc[i]:5);
    close(f);
end;
Procedure xuly;
var h : byte;
begin
    mm:=2000000000;
    for h:=1 to n do Dijkstra(h); {Xử lý tại N đỉnh khác nhau}
end;
BEGIN
    for m:=1 to 200 do new(w[m]);
    nhap_input; xuly; ghi_out;
    for m:=1 to 200 do dispose(w[m]);
END.

```

Bài 24. Đường mòn và những con bò (IOI 2003 , Trail Maintenance)

Những con bò của anh nông dân John muốn di chuyển tự do trong N ($1 \leq N \leq 200$) cánh đồng (được đánh số từ 1 đến N) của trang trại, tuy nhiên những cánh đồng được ngăn cách nhau bởi rừng. Những con bò muốn duy trì những đường mòn giữa từng cặp hai cánh đồng sao cho chúng có thể di chuyển từ một cánh đồng này tới một cánh đồng khác bằng cách sử dụng các con đường mòn đó. Những con bò có thể di chuyển trên những con đường mòn theo cả hai hướng.

Những con bò không thể tự tạo ra các con đường mòn, thay cho điều này, chúng theo vết những động vật hoang dã mà chúng đã phát hiện ra. Mỗi tuần lễ, chúng có thể duy trì bất kỳ hoặc tất cả các đường mòn do động vật hoang dã tạo ra mà chúng biết.

Luôn tò mò, những con bò thường xuyên phát hiện ra một động vật hoang dã mới tại ngày đầu mỗi tuần. Khi đó chúng có thể quyết định chọn một tập các đường mòn được duy trì cho tuần đó sao cho chúng có thể di

chuyển từ một cánh đồng bất kỳ tới một cánh đồng bất kỳ khác. Những con bò chỉ sử dụng những con đường mòn đang được duy trì.

Những con bò luôn muốn tổng độ dài các đường mòn mà chúng duy trì là nhỏ nhất. Những con bò có thể chọn duy trì một tập con của các đường mòn do các động vật hoang dã tạo ra mà chúng biết kể cả một số đường mòn được duy trì ở tuần lễ trước.

Các đường mòn của động vật hoang dã (cả khi được duy trì) là không thẳng. Hai đường cùng nối tới hai cánh đồng có thể có độ dài khác nhau. Khi hai con đường mòn vắt ngang qua nhau, con bò sẽ không chuyển đường mòn trừ khi chúng ở trong cánh đồng.

Tại ngày đầu của mỗi tuần lễ những con bò sẽ miêu tả những con đường mòn do động vật hoang dã tạo ra mà chúng phát hiện thấy.

Chương trình của bạn cần xuất ra tổng độ dài nhỏ nhất của các đường mòn mà những con bò cần duy trì trong tuần lễ ấy sao cho chúng có thể di chuyển từ một cánh đồng bất kỳ sang một cánh đồng bất kỳ khác, nếu tồn tại tập các đường mòn như vậy.

Input: input chuẩn

- Dòng đầu tiên chứa hai số nguyên cách nhau bởi dấu cách, N và W . W là số tuần lễ mà chương trình sẽ đáp ứng. ($1 \leq W \leq 6000$)
- Với mỗi tuần lễ, đọc đúng 1 dòng về đường mòn do động vật hoang dã tạo ra đã phát hiện được. Dòng này chứa 3 số nguyên cách nhau dấu cách: các điểm đầu mút (số hiệu hai cánh đồng) và số nguyên là độ dài của đường mòn ($1..10000$). Không có đường mòn nào có hai đầu mút là cùng một cánh đồng.

Output: output chuẩn

Ngay sau khi chương trình của bạn biết được đường mòn mới phát hiện, nó cần xuất ra một dòng cho biết tổng độ dài nhỏ nhất của các đường mòn mà những con bò muốn duy trì sao cho chúng có thể di chuyển từ cánh đồng này sang cánh đồng khác. Nếu không có tập đường mòn để các con bò có thể di chuyển từ cánh đồng này sang cánh đồng khác thì ghi “-1”.

Chương trình của bạn cần thoát sau khi xuất câu trả lời cho tuần lễ cuối cùng.

Ví dụ:

Input	Output	Giải thích
4 6		
1 2 10	-1	Không có đường mòn nối 4 với các cánh đồng còn lại
1 3 8	-1	Không có đường mòn nối 4 với các cánh đồng còn lại
3 2 3	-1	Không có đường mòn nối 4 với các cánh đồng còn lại
1 4 3	14	Duy trì 1 4 3, 1 3 8 và 3 2 3
1 3 6	12	Duy trì 1 4 3, 1 3 6 và 3 2 3
2 1 2	8	Duy trì 1 4 3, 2 1 2 và 3 2 3

Yêu cầu: Thời gian chạy chương trình 1 giây của CPU. Bộ nhớ 64MB

Cách cho điểm: Được đầy đủ điểm mỗi test nếu output đúng, không lấy điểm từng phần với mọi test.

Gợi ý. Mỗi lần đọc được một cánh mới ta nạp cánh mới đó vào danh sách các cánh đã có sao cho danh sách luôn luôn được sắp tăng. Sau đó dùng thuật toán Kruskal với cách tổ chức dữ liệu là hợp nhất dần các cây con sẽ được cây khung có tổng trọng số bé nhất.

Chương trình.

```
uses crt;
const
    max    = 200;
    maxc   = 2000001;
    fi     = '';
    fo     = '';
var
    f, g      : text;
    n,sc      : integer;
    a,b       : array[1..6000] of byte;
    l         : array[0..6000] of longint;
    father    : array[1..max] of integer;
    w,t,count,sum,long:longint;

procedure init;
begin
    count := 0;
    l[0]  := 0;
    long  := 10001;
```

```

end;
procedure union(r1,r2:integer);
var   x : integer;
begin
  x:=father[r1]+father[r2];
  if father[r1]>father[r2] then begin
    father[r1]:=r2; father[r2]:=x;
  end
  else begin
    father[r2]:=r1; father[r1]:=x;
  end;
end;
procedure swap(var i,j:longint);
var temp:integer;
begin
  temp:=i; i:=j; j:=temp;
end;
procedure swap1(var i,j:byte);
var   temp      : integer;
begin
  temp:=i; i:=j; j:=temp;
end;
function root(i:integer):integer;
begin
  while father[i]>0 do i:=father[i];
  root:=i;
end;
procedure solve;
var i,u,v,r1,r2,x : integer;
begin
  inc(count); {đọc thêm 1 dòng mới}
  readln(f,a[count],b[count],l[count]); {l[u vào a, b, l]}
  x:=count;
  if l[x]>=long then begin {để phòng file dữ liệu cho dữ liệu sai}
    dec(count);
    exit;
  end;
  while l[x]<l[x-1] do begin {chèn l[x] sao cho mảng l vẫn sắp tăng}
                                {để có thể thực hiện Kruskal}
    swap1(a[x],a[x-1]);
    swap1(b[x],b[x-1]);
    swap(l[x],l[x-1]);
    dec(x);
  end;

```

{Tạo cây khung bằng thuật toán Kruskal: chọn cạnh ngắn nhất trong các cạnh còn ở ngoài cây hợp vào cây. Điều kiện hợp được là 2 đầu của cạnh này thuộc hai cây con khác nhau ($r1 <> r2$)}

```

for i:=1 to n do
    father[i]:=-1;
sc:=0; i:=1; sum:=0;
repeat
    u:=a[i]; v:=b[i];
    r1:=root(u); r2:=root(v);
    if r1<>r2 then begin
        union(r1,r2);
        inc(sc);
        inc(sum,l[i]);
        long:=l[i];
    end;
    inc(i);
    if i>count then exit;
until sc=n-1;
end;
procedure printresult;
var i: integer;
begin
    if sc<n-1 then begin
        writeln(g,-1); long:=10001;
        exit;
    end;
    writeln(g,sum);
end;
BEGIN
    assign(f,fi); reset(f);
    assign(g,fo); rewrite(g);
    readln(f,n,w);
    init;
    for t:=1 to w do begin
        solve;
        printresult;
    end;
    close(f); close(g);
END.

```

Bài 25. Joinst - Ghép xâu ký tự

Cho hai tập hợp $A=\{a_1, a_2, \dots, a_n\}$ $B=\{b_1, b_2, \dots, b_n\}$ chứa các xâu.

Một chuỗi S được gọi là nhận được từ A nếu S có thể được tạo thành bằng cách ghép các chuỗi trong A. Tương tự chuỗi S được gọi là nhận được từ B nếu S có thể tạo thành được bằng cách ghép các chuỗi tương ứng trong B.

Ví dụ: $A=\{'12', '43', '5'\}$, $B=\{'124', '31', '25'\}$. Chuỗi $S='1254312'$ là chuỗi có thể nhận được từ A nhưng không thể nhận được từ B. Chuỗi $S='12425'$ là chuỗi có thể nhận được từ B nhưng không thể nhận được từ A.

Hãy tìm chuỗi S nhận được từ A và B với độ dài ngắn nhất có thể.

Dữ liệu: vào từ file văn bản JOINST.INP

Dòng đầu tiên chứa N, M là số lượng các chuỗi ký tự của tập hợp A và tập hợp B. ($1 \leq N, M \leq 100$)

N dòng tiếp theo, mỗi dòng chứa một chuỗi ký tự của tập hợp A.

M dòng cuối, mỗi dòng chứa một chuỗi ký tự của tập hợp B

Độ dài của các chuỗi ký tự trong các tập hợp A và B không quá 20.

Kết quả: Ghi ra file văn bản JOINST.OUT:

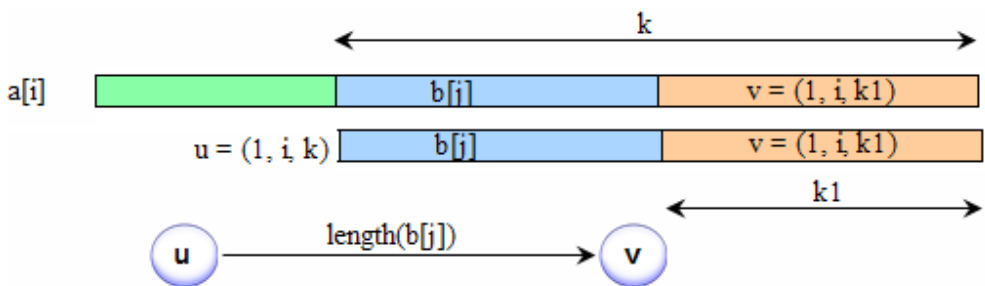
Dòng đầu ghi P là độ dài của chuỗi tìm được, ghi -1 nếu không tìm được chuỗi chung ngắn nhất. Dòng thứ 2 ghi chuỗi tìm được (nếu dòng đầu khác -1)

Ví dụ:

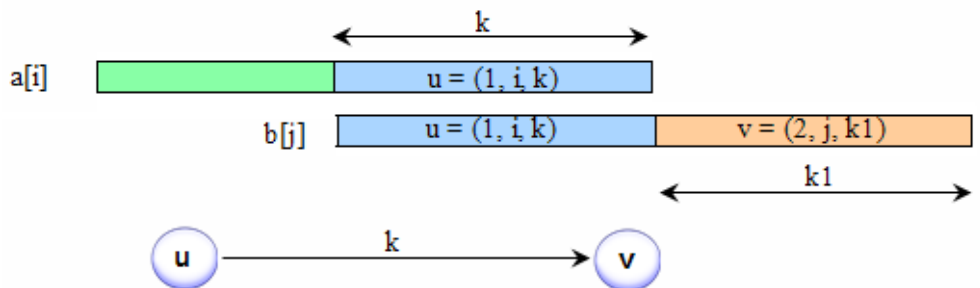
JOINST.INP	JOINST.OUT
3 3	7
12	1243125
43	
5	
124	
31	
25	

Gợi ý. Xây dựng đồ thị mà mỗi đỉnh là một bộ ba (p, i, k) trong đó $p=1$ hoặc 2. Bộ $(1, i, k)$ tương ứng với chuỗi con có k ký tự cuối cùng của chuỗi $a[i]$ và được nén thành một số nguyên $u = (i-1)*21 + (k+1)$. Bộ $(2, i, k)$ tương ứng với chuỗi con có k ký tự cuối của chuỗi $b[i]$ và được nén thành số nguyên $u = MN*21 + (i-1)*21 + (k+1)$. Sau đây là 4 trường hợp sẽ có cung đi từ đỉnh u sang đỉnh v:

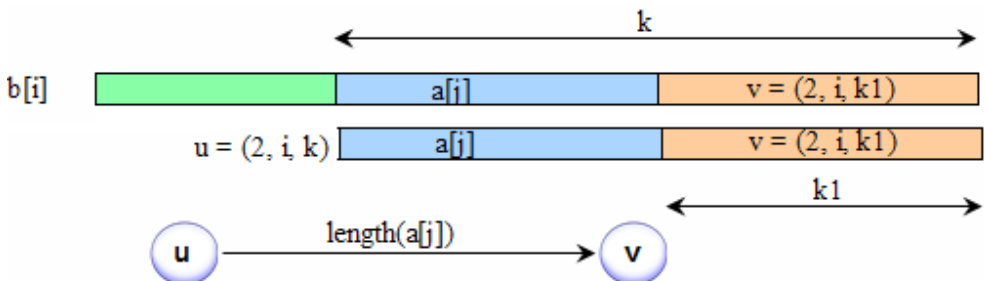
Trường hợp 1a. Chuỗi $b[j]$ nằm hoàn toàn trong đoạn cuối của chuỗi $a[i]$



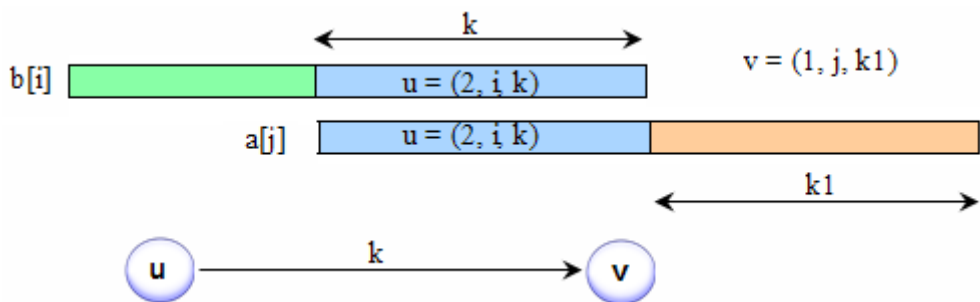
Trường hợp 1b. Đoạn cuối cùng của $a[i]$ là đoạn đầu của $b[j]$



Trường hợp 2a. Xâu $a[j]$ nằm hoàn toàn trong đoạn cuối của xâu $b[i]$



Trường hợp 2b. Đoạn cuối cùng của $b[i]$ là đoạn đầu của $a[j]$



Chương trình.

```

const fi      = 'JOINST.IN';
      fo      = 'JOINST.OUT';
      MN      = 100;
      maxQ    = 5000;
type  xau20   = string[20];
var   f, g    : text;
      N, M    : integer;
      a, b    : array[1..MN] of xau20;
      ok      : boolean;
      Q       : array[1..maxQ] of integer;
      qf,ql   : integer;
      tr      : array[1..maxQ] of integer;
      kc      : array[1..maxQ] of longint;
      kt      : integer;
      slx     : integer;
      x       : array[1..maxQ] of integer absolute q;
      Lmin    : Longint;

function doiso(p,i,k: integer): integer; {nén bộ (p,i,k) thành một số}
begin
  doiso:=(p-1)*MN*21+(i-1)*21+(k+1);
end;

Procedure doidinh(u: integer; var p,i,k: integer);
begin  {đổi số u thành bộ số (p, i, k)}
  p:=(u-1) div (21*MN)+1;
  u:=(u-1) mod (21*MN)+1;
  i:=(u-1) div 21+1;
  k:=(u-1) mod 21;
end;

Procedure init; {Khởi trị hàng đợi}
begin
  qf:=1;  ql:=1;
end;

Procedure put(u: integer); {Nạp đỉnh (số u) vào hàng đợi}
begin
  q[ql]:=u;  inc(ql);
end;

function get: integer; {Lấy khỏi hàng đợi một đỉnh ở đầu hàng đợi}
begin
  get:=q[qf];  inc(qf);
end;

function empty: boolean; {Hàm kiểm tra hàng đợi rỗng hay không}
begin
  empty:=(qf=ql);
end;

Procedure doc_dulieu; {Đọc dữ liệu input}
var i: integer;

```

```

begin
  assign(f,fi); reset(f);
  readln(f,N,M); {số các xâu của tập A và của tập B}
  for i:=1 to N do readln(f,a[i]); {đọc các xâu thuộc tập A}
  for i:=1 to M do readln(f,b[i]); {đọc các xâu thuộc tập B}
  close(f);
end;
Procedure update(u,v,ts: integer); {dựa vào nhãn của u sửa nhãn v, biết
trong số cung (u, v) là ts}
begin
  if (tr[v]<0) and (kc[v]>kc[u]+Ts) then begin {nhãn v chưa tối ưu}
    tr[v]:=-u; {vết: trước v là u}
    kc[v]:=kc[u]+ts; {nhãn mới của v}
  end;
  if tr[v]=0 then begin {v chưa thăm (chưa có nhãn)}
    put(v); { nạp v vào hàng đợi}
    tr[v]:=-u; {vết: trước v là u}
    kc[v]:=kc[u]+ts; {nhãn của v}
  end;
end;
Procedure kela(i,k: integer); {tìm và sửa nhãn cho v kề với u=(1, i, k) theo
trường hợp 1a}
var u,j,k1,ts,v: integer;
    s,s1: xau20;
begin
  if length(a[i])<k then exit;
  u:=doiSo(1,i,k); {đổi bộ (1,i,k) thành số nguyên u}
  s:=copy(a[i],length(a[i])-k+1,k); {định u ứng với xâu s}
  for j:=1 to M do {duyệt các xâu b[j]}
  if pos(b[j],s)=1 then begin {xâu b[j] ở vị trí đầu xâu s}
    {thì tìm phần cuối a[i] ứng với đỉnh v=(1,i, length(s)-length(b[j]))}
    k1:=length(s)-length(b[j]);
    ts:=length(b[j]);
    v:=doiSo(1,i,k1);
    update(u,v,ts); {sửa lại nhãn đỉnh v, nhờ nhãn đỉnh u}
  end;
end;
Procedure kelb(i,k: integer); {tìm và sửa nhãn cho v kề với u=(1, i, k) theo
trường hợp 1b}
var u,j,k1,ts,v: integer;
    s: xau20;
begin
  if length(a[i])<k then exit;
  u:=doiso(1,i,k);
  s:=copy(a[i],length(a[i])-k+1,k);

```



```

for j:=1 to M do
if (s='') or (pos(s,b[j])=1) then begin
    k1:=length(b[j])-length(s);
    ts:=length(s);
    v:=doiso(2,j,k1);
    update(u,v,ts);
end;
end;
Procedure ke2a(i,k: integer); {tìm và sửa nhân cho v kề với u=(2, i, k) theo trường hợp 2a}
var u,j,k1,ts,v: integer;
    s: xau20;
begin
    if length(b[i])<k then exit;
    u:=doiso(2,i,k);
    s:=copy(b[i],length(b[i])-k+1,k);
    for j:=1 to N do
    if pos(a[j],s)=1 then begin
        k1:=length(s)-length(a[j]);
        ts:=length(a[j]);
        v:=doiso(2,i,k1);
        update(u,v,Ts);
    end;
end;
Procedure ke2b(i,k: integer); {tìm và sửa nhân cho v kề với u=(2, i,k) theo trường hợp 2b}
var u,j,k1,ts,v: integer;
    s: xau20;
begin
    if length(b[i])<k then exit;
    u:=doiso(2,i,k);
    s:=copy(b[i],length(b[i])-k+1,k);
    for j:=1 to N do
    if (s='') or (pos(s,a[j])=1) then begin
        k1:=length(a[j])-length(s);
        ts:=length(s);
        v:=doiso(1,j,k1);
        update(u,v,Ts);
    end;
end;
Procedure process; {Thuật toán sửa nhân nhiều lần cho đến khi không thể}
var i,k,p,u: integer;
begin
    ok:=true; {Báo hiệu nhận được xâu ghép từ A và ghép từ B}
    init; {Khởi trị hàng đợi}
    for i:=1 to maxQ do tr[i]:=0; {Khởi trị mảng vết và đánh dấu loại đỉnh}

```

```

for i:=1 to N do begin
  {Nạp các đỉnh xuất phát (là các  xâu rỗng được tạo từ các xâu của tập A) vào hàng đợi}
  u:=doiso(1,i,0);
  put(u);
  tr[u]:=(maxQ+1); {vết của các đỉnh xuất phát}
  kc[u]:=0;
end;
while not empty do begin {Sửa nhãn các đỉnh cho đến khi hàng đợi rỗng}
  u:=get; {Lấy đỉnh u từ đầu hàng đợi}
  tr[u]:=-tr[u]; {xác nhận đỉnh u đã được cố định nhãn}
  doidinh(u,p,i,k); {đổi đỉnh u thành bộ số (p, i, k) để sau này tiện dùng}
  if (k=0) and (p=2) then begin {điều kiện kết thúc: u là đỉnh rỗng tạo từ
một xâu thuộc tập B}
    kt:=u; {Lưu lại đỉnh kết thúc}
    exit; {Thoát lặp}
  end;
  case p of {Nếu không bị thoát thì lựa chọn xử lý tiếp theo giá trị của p}
    1: begin {u là đỉnh tương ứng với phần cuối của xâu a[i]}
      kela(i,k); {sửa nhãn cho đỉnh v kề u theo trường hợp 1a}
      kelb(i,k); {sửa nhãn cho đỉnh v kề u theo trường hợp 1b}
    end;
    2: begin {u là đỉnh tương ứng phần cuối của xâu b[i]}
      ke2a(i,k); {sửa nhãn cho đỉnh v kề u theo trường hợp 2a}
      ke2b(i,k); {sửa nhãn cho đỉnh v kề u theo trường hợp 2b}
    end;
  end;
  end;
  ok:=false; {Không tìm được hành trình (không có xâu ghép từ A và từ B)}
end;
Procedure find; {Tìm trả lời}
var v: integer;
begin
  if not ok then exit; {Không tìm được xâu chung thì thoát}
  slx:=0; {số lượng đỉnh trên hành trình ngắn nhất}
  Lmin:=Kc[kt]; {Độ dài của hành trình cũng là độ dài của xâu ghép từ A và từ B}
  v:=kt; {Lần ngược hành trình từ đỉnh kết thúc}
  repeat
    inc(slx);
    x[slx]:=v;
    v:=tr[v];
  until v=maxQ+1; {gặp vết của đỉnh xuất phát}
end;
Procedure ghi(u,v: integer); {Ghi một cung (u, v)}
var i,k,p,il,kl,pl: integer;

```

```

    s: xau20;
begin
    doidinh(u,p,i,k); {đổi u=(p, i, k)}
    doidinh(v,p1,i1,k1); {đổi v=(p1, i1, k1)}
    if p=p1 then begin {v kề với u theo trường hợp 1a hoặc 2a}
        if p=1 then s:=a[i] {trường hợp 1a: thì cũng ghép thêm được lấy từ a[i]}
        else s:=b[i]; {còn không được lấy từ b[i]}
        s:=copy(s,length(s)-k+1,k); {tạo ra đoạn ghép thêm vào hành trình}
        s:=copy(s,1,k-k1);
    end
else begin {v kề với u theo trường hợp 1b hoặc 2b}
    if p=1 then s:=b[i1] else s:=a[i1];
    s:=copy(s,1,k);
end;
write(g,s); {ghi đoạn ghép thêm vào tệp output}
end;
Procedure in_ketqua;
var i: integer;
begin
    assign(g,fo); rewrite(g);
    if not ok then writeln(g,-1) {ghi vô nghiệm}
    else begin {ghi nghiệm}
        writeln(g,Lmin); {độ dài của xâu được ghép từ A và cũng được ghép từ B}
        for i:=slx downto 2 do
            ghi(x[i],x[i-1]); {ghi liên tiếp các đoạn ghép}
        end;
        close(g);
    end;
end;
BEGIN
    doc_dulieu;
    process;
    find;
    in_ketqua;
END.

```

Bài 26. SightseeingTrip

Trong thị trấn Adelton thuộc đảo Zanzibar có một chi nhánh du lịch. Chi nhánh này quyết định mời chào khách hàng bằng một chuyến thăm quan miễn phí và nhiều điều hấp dẫn khác. Để thu được càng nhiều lợi nhuận càng tốt chi nhánh đưa ra một quyết định tinh ranh là: Cần tìm một hành trình ngắn nhất mà xuất phát và kết thúc tại cùng một địa điểm. Nhiệm vụ của bạn là viết chương trình tìm hành trình ngắn nhất này.

Trong thị trấn có N nút giao thông đánh số từ 1 đến N và M đoạn đường phố hai chiều đánh số từ 1 đến M . Hai nút giao thông có thể nối với nhau bởi nhiều đoạn đường phố, nhưng không có đoạn đường nào nối một nút giao thông với chính nút ấy. Mỗi hành trình tham quan là dãy gồm một số đoạn đường y_1, y_2, \dots, y_k với $k \geq 2$. Đoạn đường y_i ($1 \leq i \leq k-1$) nối nút x_i và x_{i+1} , đoạn đường y_k nối nút x_k với x_1 . Tất cả các số hiệu x_1, x_2, \dots, x_k là khác nhau. Độ dài của hành trình là tổng độ dài của tất cả các đoạn đường thuộc hành trình, nghĩa là bằng $L(y_1) + L(y_2) + \dots + L(y_k)$ với $L(y_i)$ là độ dài đoạn đường y_i ($1 \leq i \leq k-1$). Chương trình của bạn cần tìm hành trình mà tổng độ dài của nó là nhỏ nhất hoặc khẳng định không tìm được vì không có hành trình tham quan trong thị trấn.

Input Dòng thứ nhất của file TRIP.IN chứa 2 số nguyên dương: Số nút $N \leq 100$ và số đoạn đường $M \leq 10000$. Mỗi dòng trong M dòng tiếp theo mô tả một đoạn đường. Nó chứa 3 số nguyên dương: số hiệu nút đầu, nút cuối và độ dài của đoạn đường (độ dài không quá 500).

Output Chỉ có một dòng trong file TRIP.OUT. Nó chứa hoặc là xâu “No solution.” trong trường hợp không có hành trình tham quan; hoặc chứa các số hiệu các **nút** trên hành trình ngắn nhất theo thứ tự đi qua x_1, x_2, \dots, x_k , cách nhau dấu trống. Nếu có nhiều hành trình ngắn nhất cùng độ dài thì bạn có thể nêu một hành trình tùy ý trong các hành trình này.

Ví dụ 1

Ví dụ 1	
TRIP.IN	TRIP.OUT
5 7	1 3 5 2
1 4 1	
1 3 300	
3 1 10	
1 2 16	
2 3 100	
2 5 15	
5 3 20	

Ví dụ 2	
TRIP.IN	TRIP.OUT
4 3	No solution.
1 2 10	
1 3 20	
1 4 30	

Gợi ý.

Giả sử cạnh đầu tiên của chu trình là (i,j) có độ dài là h . Dùng thuật toán Dijkstra tìm đường đi ngắn nhất từ j về i không đi lại cạnh (i,j) bằng cách xoá bỏ cung (j,i) . Rõ ràng, nếu không có xử lý đặc biệt thì với thuật toán trên, độ phức tạp tính toán trung bình là $O(n^4)$.

Gọi độ dài chu trình tối ưu đã tìm được là *best* (ban đầu chưa có chu trình nào thì *best* là vô cùng), độ dài cạnh (k_1, k_2) là $a[k_1,k_2]$, nhãn đường đi ngắn nhất của đỉnh k là $L[k]$. Cần lưu ý một số điểm sau trong thực hiện thuật toán Dijkstra:

+ Đỉnh tự do k_1 có nhãn nhỏ nhất $L[k_1]$ phải thỏa mãn: $L[k_1]+h < best$.

+ Đỉnh k_2 kề với đỉnh k_1 chỉ được sửa nhãn khi thỏa mãn điều kiện: $a[k_1,k_2]+L[k_1]+h < best$ (1) và $a[k_1,k_2] + L[k_1] < L[k_2]$ (2)

Chương trình

```
uses      crt;
const     max      = 100;
          limit     = 50001;
          fi        = 'trip.in';
          fo        = 'trip.out';
var        a        : array[1..max,1..max] of word;
          l,tr,d,tr0 : array[1..max] of word;
          best,h     : word;
          li,n       : byte;

Procedure input;
var  f      : text;
     i,j    : byte;
     k      : integer;
     m,w    : integer;
begin
  for i:=1 to max do
    for j:=1 to max do a[i,j]:=limit; {không có đường đi từ i tới j}
  assign(f,fi);      reset(f);
  readln(f,n,m); {số đỉnh, số cạnh}
  for k:=1 to m do begin
    readln(f,i,j,w); {đọc một cạnh}
    if w<a[i,j] then begin {sơ giản dữ liệu, chỉ giữ lại dữ liệu cần thiết}
      a[i,j] := w;
      a[j,i] := w;
    end;
  end;
  close(f);
end;
```

```

function find(var k1 : byte) : boolean;
var k    : byte;
    min  : word;
begin
    min:=best-h; {Khởi trị nhãn nhỏ nhất}
    for k:=1 to n do {chọn đỉnh tự do có nhãn nhỏ nhất}
        if (d[k]=0) and (min>l[k]) then begin
            min:=l[k];
            k1:=k;
        end;
    find:=(l[k1]<best-h) ; {điều kiện cần của đỉnh k1}
end;

Procedure repair(k1 : byte);
var k2    : byte;
begin
    for k2:=1 to n do
        if ((a[k1,k2]+l[k1]+h)<best) then {điều kiện (1)}
        if (a[k1,k2]+l[k1]<l[k2]) then begin {điều kiện (2)}
            l[k2]:=a[k1,k2]+l[k1]; {sửa lại nhãn đỉnh k2}
            tr[k2]:=k1; {vết hành trình}
        end;
    end;

Procedure work(i,j : byte);
var k1    : byte;
    ok     : boolean;
    f      : text;
begin
    fillchar(tr,sizeof(tr),0);
    fillchar(d,sizeof(d),0);
    for k:=1 to n do l[k]:=limit; {khởi trị nhãn mọi đỉnh}
    l[j]:=0; {khởi trị nhãn đỉnh j- của hành trình i→j→...k1→k2...→i}
    repeat
        ok:=find(k1); {chọn đỉnh tự do có nhãn nhỏ nhất}
        if ok then begin {nếu tìm được}
            if k1=i then begin {được chu trình mới}
                if l[k1]+ h < best then begin {chu trình mới tối ưu hơn thì}
                    best:= l[k1] + h; {ghi nhận độ dài chu trình tối ưu hơn}
                    tr0:=tr; {lưu lại mảng vết}
                    li:=i; {lưu lại đỉnh i}
                end;
                exit;
            end;
        end;
        repair(k1); {chưa thành chu trình thì sửa nhãn các đỉnh kề với k1}
        d[k1] :=1; {cố định nhãn đỉnh k1}
    end;

```

```

until ok=false; {không còn tìm được đỉnh tự do có nhãn nhỏ nhất}
end;
Procedure solution;
var i,j : byte;
    nho : integer;
begin
    best := limit; {khởi trị độ dài chu trình tối ưu}
    for i:=1 to n-1 do
        for j:=i+1 to n do
            if a[i,j]<best then begin {điều kiện chọn cạnh đầu tiên của chu trình}
                h      := a[i,j]; {độ dài cạnh đầu tiên trên chu trình}
                nho     := a[j,i]; {lưu lại a[j,i]}
                a[j,i] := limit; {xóa chiều từ j về i}
                work(i,j); {tìm đường ngắn nhất từ j về i}
                a[j,i] := nho; {trả lại giá trị của a[j,i] để tìm kiếm chu trình khác}
            end;
        end;
    end;
Procedure output;
var f : text;
    q : array[1..max] of byte;
    t,k : byte;
begin
    assign(f,fo); rewrite(f);
    if best<limit then begin {có nghiệm}
        t:=0; {đếm}
        k:=li;
        while k>0 do begin
            inc(t);
            q[t]:=k;
            k:=tr0[k];
        end;
        for k:=t downto 1 do write(f,q[k],#32);
    end
    else writeln(f,'No solution. ');
    close(f);
end;
BEGIN
    input;
    solution;
    output;
    writeln(best);
END.

```

Bài 27. Đường truyền tin

Một mạng truyền tin gồm N máy được đánh số từ 1 đến N . Có M kênh truyền tin được đánh số từ 1 đến M . Mỗi kênh truyền tin thứ i được mô tả bởi cặp số (u_i, v_i) , $u_i \neq v_i$ cho biết có kênh truyền tin i kết nối trực tiếp hai máy u_i và v_i . Mỗi kênh truyền tin i được gán với 2 số nguyên dương c_i , d_i trong đó c_i là khả năng thông qua còn d_i là độ trễ, $i=1,2,\dots,n$. Giả sử s và t là 2 máy nào đó của mạng. Ta gọi một đường truyền tin trên mạng là một dãy $z_0, z_1, z_2, \dots, z_{k-1}, z_k$ trong đó $z_0=s$, $z_k=t$ (z_{i-1}, z_i) là kênh truyền tin của mạng, $i=1, 2, \dots, k$. Thời gian để truyền W đơn vị thông tin từ máy s đến máy t theo đường truyền nói trên là:

$$d(z_0, z_1) + d(z_1, z_2) + \dots + d(z_{k-1}, z_k) + W/c_{\min}$$

trong đó $d(z_{i-1}, z_i)$ là độ trễ của kênh truyền tin (z_{i-1}, z_i) và c_{\min} là giá trị nhỏ nhất trong số các khả năng thông qua của các kênh trên đường truyền tin. Giả thiết giữa hai máy bất kỳ trong mạng luôn có kênh truyền tin.

Yêu cầu: Cho trước 2 máy truyền tin s và t . Hãy tìm đường truyền tin để truyền W đơn vị thông tin từ s đến t với thời gian truyền tin nhỏ nhất.

Dữ liệu vào: file văn bản COMM.IN:

Dòng đầu tiên chứa hai số nguyên N, M ($N \leq 150$)

Dòng thứ hai ghi s, t, W (W là số nguyên dương không vượt quá 10000)

Dòng thứ i trong số M dòng tiếp theo mô tả thông tin về kênh truyền tin thứ i bao gồm 4 số nguyên dương u_i, v_i, c_i, d_i cho biết kênh truyền tin từ máy u_i đến máy v_i và kênh này có khả năng thông qua là c_i và độ trễ d_i , $i=1, 2, \dots, M$.

Kết quả: Ghi ra file văn bản COMM.OUT:

Dòng đầu tiên ghi thời gian truyền tin theo đường truyền tin tìm được (làm tròn đến hai chữ số sau dấu thập phân)

Dòng thứ hai ghi dãy các chỉ số máy biểu diễn đường truyền tin tìm được.

Các giá trị số trên cùng dòng được ghi cách nhau ít nhất một dấu cách.

Ví dụ:

	COMM.IN	COMM.OUT
4 5		13.17
1 4 25		1 3 4
1 2 2 1		
1 3 8 4		
2 3 1 2		
2 4 6 5		
3 4 6 5		

Gợi ý.

Dùng thuật toán Dijkstra tìm đường đi ngắn nhất từ s đến t (ở đây trọng số cạnh là thời gian truyền).

Chú ý: Trong quá trình thực hiện thuật toán có thể dùng một mảng *like_heap* để lưu các đỉnh đã có nhãn nhưng chưa được cố định nhãn. Mỗi lần lấy ra khỏi mảng này một đỉnh có nhãn bé nhất để cố định nhãn và dùng sửa nhãn cho các đỉnh đã có nhãn hoặc chưa có nhãn.

Chương trình.

```
{ $A+, B-, D+, E+, F-, G-, I+, L+, N+, O-, P-, Q+, R+, S+, T-, V+, X+ }
{ $M 16384, 0, 655360 }
Const fi    = 'COMM.IN';
      fo    = 'COMM.OUT';
      maxN = 150;
type mang=array[1..maxN] of longint;
var  f, g      : text;
      n, m      : longint; {số máy, số kênh truyền tin}
      s, t, w    : longint; {xuất phát, đích, lượng thông tin cần chuyển}
      c          : array[1..maxN] of ^mang; {khả năng thông qua}
      d          : array[1..maxN] of ^mang; {độ trễ}
      like_heap  : array[1..maxN] of longint; {hàng đợi chứa các đỉnh
đã có nhãn nhưng chưa cố định nhãn}
      nh        : longint; {số phần tử có trong hàng đợi}
      time      : array[1..maxN] of real; {nhãn thời gian ngắn nhất}
      tr        : array[1..maxN] of longint; {lưu vết}
      min_c,    : array[1..maxN] of longint; {nhãn khả năng thông qua nhỏ nhất}
      sum_d     : array[1..maxN] of longint; {nhãn độ trễ}
      x         : array[1..maxN] of longint; {đường truyền ngắn nhất}
      slx       : longint; {số máy của đường truyền}

Procedure capphat;
var i: longint;
begin
```

```

    for i:=1 to maxN do new(c[i]);
    for i:=1 to maxn do new(d[i]);
end;
Procedure giaiphong;
var i: longint;
begin
    for i:=1 to maxN do dispose(c[i]);
    for i:=1 to maxn do dispose(d[i]);
end;
Procedure init; {Khởi trị hàng đợi rỗng}
begin
    nh:=0;
end;
Procedure put(u: longint); { nạp đỉnh u vào hàng đợi}
begin
    inc(nh);
    like_heap[nh]:=u;
end;
function get: longint; {lấy khỏi hàng đợi đỉnh u có nhãn thời gian nhỏ nhất}
var u,i:longint;
begin
    u:=1;
    for i:=2 to nh do
        if time[like_heap[i]]<time[like_heap[u]] then u:=i;
    get:=like_heap[u]; {lấy ra đỉnh ở vị trí u}
    like_heap[u]:= like_heap[nh]; {lấp đỉnh tại vị trí nh vào vị trí u đã lấy ra}
    dec(nh); {giảm số đỉnh có nhãn nằm trong hàng đợi}
end;
function empty: boolean; {hàm kiểm tra hàng đợi rỗng}
begin
    empty:=(nh=0);
end;
Procedure doc_dulieu;
var u,v,i: longint;
begin
    assign(f,fi); reset(f);
    readln(f,n,m); {số máy, số kênh truyền}
    readln(f, s, t, w); {máy xuất phát, máy đích và lượng tin cần truyền đi}
    for u:=1 to N do
    for v:=1 to N do begin {Khởi trị các mảng c và d}
        c[u]^v:=0;
        d[u]^v:=0;
    end;
    for i:=1 to m do begin {khả năng truyền tin và độ trễ của các kênh (u,v)}
        read(f,u,v);

```

```

    readln(f,c[u]^[v],d[u]^[v]);
end;
close(f);
end;
function min(u,v: longint): longint;
begin
    if u<v then min:=u else min:=v;
end;
function thoigian(u,v: longint): real; {tính thời gian truyền từ u đến v}
begin
    thoigian:=sum_d[u]+d[u]^[v]+w/min(min_c[u],c[u]^[v]);
end;
Procedure xaydung; {Thực hiện thuật toán Dijkstra}
var i,u,v: longint;
    ll: real;
begin
    init; {Khởi trị hàng đợi rỗng}
    for i:=1 to N do tr[i]:=0; {Khởi trị mảng vết}
    put(s); { nạp đỉnh xuất phát}
    tr[s]:=-(N+1); {gói ước vết đỉnh trước của đỉnh xuất phát}
    sum_d[s]:=0; {khởi trị tổng độ trễ}
    min_c[s]:=maxlongint; {Khởi trị khả năng truyền nhỏ nhất}
    time[s]:=0; {Khởi trị thời gian truyền ngắn nhất}
    repeat
        u:=get; {lấy ra đỉnh có nhãn thời gian nhỏ nhất trong các đỉnh chưa cố định nhãn}
        tr[u]:=-tr[u]; { đánh dấu u đã cố định nhãn, không được sửa nhãn nữa}
        for v:=1 to N do {Tìm các đỉnh v kề với u (có kênh truyền tin từ u đến v)}
            if c[u]^[v]>0 then begin
                ll:= thoigian(u,v); {thời gian truyền từ u đến v}
                if (tr[v]<0) and (time[v]>ll) then begin
                    {v có nhãn,v nằm trong hàng đợi, nhãn thời gian truyền của v chưa tối ưu thì }
                    time[v]:=ll; {sửa nhãn thời gian truyền}
                    min_c[v]:=min(min_c[u],c[u]^[v]); {sửa nhãn khả năng truyền}
                    sum_d[v]:=sum_d[u]+d[u]^[v]; {sửa nhãn tổng độ trễ}
                    tr[v]:=-u; {lưu vết bằng số âm mới là -u: trước đỉnh v là đỉnh u}
                end;
            if tr[v]=0 then begin {v chưa có nhãn (còn ở ngoài hàng đợi)}
                time[v]:=ll; {xác nhận nhãn thời gian truyền của v}
                min_c[v]:=min(min_c[u],c[u]^[v]); {nhãn khả năng truyền}
                sum_d[v]:=sum_d[u]+d[u]^[v]; {nhãn tổng độ trễ}
                tr[v]:=-u; {xác nhận vết của v}
                put(v); { nạp v vào hàng đợi}
            end;
        end;
    end;
end;

```

```

until empty or (tr[t]>0); {hàng đợi rỗng hoặc t đã được cố định nhãn}
slx:=0; {số máy trên đường truyền tối ưu từ s đến t}
u:= t; {lần ngược đường truyền từ t về s}
repeat
    inc(slx);
    x[slx]:=u;
    u:=tr[u];
until u=N+1; {đã gặp đỉnh xuất phát s (vì vết đỉnh trước của s sau khi cố định
nhãn của s là N+1)}
end;
Procedure in_ketqua;
var i: longint;
begin
    assign(g,fo); rewrite(g);
    writeln(g,time[t]:0:2); {thời gian truyền nhanh nhất}
    for i:=slx downto 1 do
        write(g,x[i], ' '); {hiện các máy trên đường truyền tối ưu}
    close(g);
end;
BEGIN
    capphat;
    doc_dulieu;
    xaydung;
    in_ketqua;
    giaiphong;
END.

```

Bài 28. D'Artagnan

D'Artagnan là một lính ngự lâm cự phách, nhân vật chính trong cuốn tiểu thuyết “Ba người lính ngự lâm” của nhà văn A. Duma. Thời đó nước Pháp có N thành phố, $N \leq 50$ với các tên từ 1 đến N . Paris mang tên 1 còn thành phố quê hương của D'Artagnan mang tên N .

Có hai lực lượng bảo vệ kinh địch nhau: lính ngự lâm và lính cảnh vệ. Với mỗi con đường từ thành phố A đến thành phố B, tại lối ra-vào hai thành phố đầu mút có hai vọng gác, một trong hai vọng gác đó do lính ngự lâm đảm nhiệm và vọng gác kia do lính cảnh vệ.



Luật pháp thời đó qui định khi vào một thành phố bất kỳ bằng một vọng gác do một loại lính nào đó đảm nhiệm, lúc ra phải đi qua vọng gác của cùng loại lính đó.

Thông tin về mỗi con đường nối hai thành phố được cho bởi một dòng như sau:

U V W1 W2 L

với ý nghĩa có đường trực tiếp từ thành phố U đến thành phố V, thời gian đi từ U đến V bằng W1, thời gian đi từ V đến U bằng W2, L là ký tự M nếu vọng gác đầu U của đoạn đường đó do lính ngự lâm hoặc là ký tự G nếu lính cảnh vệ đảm nhiệm (theo qui định khi đó vọng gác đầu V tương ứng do lính cận vệ/ngự lâm đảm nhiệm), giữa hai mục liên tiếp trên dòng cách nhau đúng một dấu trống. Các số W1, W2 không lớn hơn 60000.

Một lần D'Artagnan cần đi rất gấp từ thành phố quê hương đến Paris để hỗ trợ cho các bạn của mình. D' Artagnan đầu kiếm rất giỏi nhưng tính toán hơi yếu. Hãy giúp D' Artagnan cần tìm một con đường để đi nhanh nhất từ quê hương đến Paris theo đúng các qui định về vọng gác.

Dữ liệu vào được cho bởi file DART.INP trong đó dòng thứ nhất ghi số N, tiếp theo là một nhóm không quá 1000 dòng, dòng thứ i ghi thông tin về đoạn đường thứ i theo qui cách nêu trên. D' Artagnan luôn có thể đi từ quê hương lên Paris

Kết quả ghi ra file DART.OUT như sau: dòng thứ nhất ghi độ dài đường đi, dòng thứ hai ghi tên các thành phố lần lượt đi trên hành trình từ thành phố N đến thành phố 1. Dòng thứ 3 ghi các số hiệu các đoạn đường lần lượt trên hành trình.

DART . INP	DART . OUT
5	5
1 2 3 3 M	5 4 1
1 4 4 4 G	6 2
1 3 2 2 G	
2 5 3 3 G	
2 4 10 10 G	
5 4 1 1 G	
4 3 1 1 M	

Gợi ý. Tương tự bài “Đường truyền tin”. Dùng thuật toán Dijkstra tìm đường đi ngắn nhất trên đồ thị có hướng. Mỗi đỉnh i của đồ thị là cặp thành phố và cổng, đó là cặp số (u, L) trong đó u là số hiệu thành phố, L là cổng vào hoặc ra. Ta sẽ dùng cách viết $i=(u, L)$ để hiểu theo nghĩa trên. Nếu L là cổng do lính ngự lâm gác thì $L=1$, L là cổng do cảnh vệ thì $L=2$. Mỗi đỉnh $i=(u,L)$ của đồ thị là một số nguyên tính theo công thức: $i=(L-1) * N + u$.

Khi đọc được một dòng sau đây trong tệp input: $U \ V \ W1 \ W2 \ M$ thì chúng ta tạo ra 2 đỉnh đồ thị là $i=(u, 1)$ và $j=(v, 2)$.

Khi đọc được một dòng sau đây trong tệp input: $U \ V \ W1 \ W2 \ G$ thì chúng ta tạo ra 2 đỉnh đồ thị là $i=(u, 2)$ và $j=(v, 1)$.

Đỉnh xuất phát có 2 đỉnh là 2 số nguyên tương ứng với các cặp số $(N,1)$ và $(N,2)$. Đỉnh kết thúc là số nguyên tương ứng với cặp số $(1,1)$ hoặc $(1,2)$.

Chương trình.

```
const
  fi    = 'DART.IN';
  fo    = 'DART.OUT';
  maxN  = 100;
var
  Gr     : array[1..maxN,1..maxN] of longint;
  a      : array[1..maxN,1..maxN] of integer;
  N      : integer;
  f, g   : text;
  H      : array[1..maxN] of longint;
  nh     : longint;
  kc     : array[1..maxN] of longint;
  tr     : array[1..maxN] of longint;
  Smin   : longint;
  slx    : longint;
  x      : array[1..maxN] of longint;
Procedure Init;
begin
  nh:=0;
end;
Procedure Put(u: longint);
begin
  inc(nh);  H[nh]:=u;
end;
function get: longint; {Lấy ra khỏi mảng H một đỉnh có nhãn kc nhỏ nhất}
```

```

var u,i: longint;
begin
  u:=1;
  for i:=2 to nh do
    if kc[H[i]]<kc[H[u]] then u:=i;
  Get:=H[u];
  H[u]:=H[nh];
  dec(nh);
end;
function empty: boolean;
begin
  empty:=(nh=0);
end;
function doiso(u,L: longint): longint; {đổi cặp số thành 1 số nguyên}
begin
  doiso:=(l-1)*N+u;
end;
Procedure doidinh(u: longint; var i,L: longint);
begin {Phân tích một đỉnh u thành cặp số (i, L)}
  l:=(u-1) div N+1;
  i:=(u-1) mod N+1;
end;
Procedure doc_dulieu;
var m,u,v,w1,w2,l,i,j: longint;
    ch: char;
begin
  fillchar(Gr,sizeof(Gr),0); {Khởi trị ma trận kề}
  assign(f,fi); reset(f);
  readln(f,N); {số thành phố}
  m:=0; {khởi trị số hiệu con đường}
  while not seekeof(f) do begin
    read(f,u,v,w1,w2); {thời gian từ u đến v là w1; từ v đến u là w2}
    repeat read(f,ch) until ch<>' '; {cho đến khi đọc được L}
    readln(f);
    inc(m);
    if ch='M' then l:=1 else l:=2; {lính ngự lâm gác: L=1, cảnh vệ: L=2}
    i:=doiso(u,l); {tạo đỉnh thứ nhất: thành phố u, có cổng ra lính ngự lâm gác}
    j:=doiso(v,3-l); {đỉnh thứ hai: thành phố v có cổng vào lính cảnh vệ gác}
    if (Gr[i,j]=0) or (Gr[i,j]>w1) then begin
      Gr[i,j]:=w1; {xác nhận cung (i,j) có trọng số w1}
      a[i,j]:=m; {xác nhận cung (i,j) thuộc con đường có số hiệu }
    end;
    if (Gr[j,i]=0) or (Gr[j,i]>w2) then begin
      Gr[j,i]:=w2; {xác nhận cung (j,i) có trọng số w2}
      a[j,i]:=m; {xác nhận cung (j,i) thuộc con đường m}
    end;
  end;
end;

```

```

    end;
end;
close(f);
end;
Procedure Dijkstra;
var L,u,v,i,j: longint;
begin
    init; {khởi trị mảng H}
    fillchar(Tr,sizeof(Tr),0); {khởi trị mảng lưu vết hành trình}
    for L:=1 to 2 do begin {Tạo các đỉnh xuất phát, nạp chúng vào H}
        u:=Doiso(N,L);
        Put(u);
        Tr[u]:=(2*N+1); {vết của đỉnh xuất phát, đỉnh này chưa cố định (vết âm)}
        kc[u]:=0; {khởi trị nhãn của đỉnh xuất phát}
    end;
    repeat
        u:=Get; {Lấy đỉnh u có nhãn nhỏ nhất trong các đỉnh có nhãn chưa cố định}
        tr[u]:=-tr[u]; {cố định nhãn của u bằng vết dương}
        doidinh(u,i,L); {đổi u thành cặp số (i,L): i là thành phố, L là loại cổng}
        for j:=1 to N do begin {Tìm các đỉnh v=(j, 3-L) kề với u}
            v:=doiso(j,3-l);
            if (Gr[u,v]>0) then begin {có cung đi từ u tới v}
                {v có nhãn chưa cố định (vì vết âm) và nhãn chưa tối ưu thì sửa lại nhãn của v}
                if (Tr[v]<0) and (kc[v]>kc[u]+Gr[u,v]) then begin
                    kc[v]:=kc[u]+Gr[u,v];
                    Tr[v]:=-u; {xác nhận vết mới của v: trước v là u}
                end;
                if Tr[v]=0 then begin {nếu v chưa có nhãn thì}
                    kc[v]:=kc[u]+Gr[u,v]; {tạo nhãn cho v}
                    Tr[v]:=-u; {vết của v, cũng là xác nhận nhãn của v chưa cố định}
                    Put(v); {nạp v vào H}
                end;
            end;
        end;
    until empty;
end;
Procedure timDuong;
var l,u,kt: longint;
begin
    Smin:=maxlongint;
    for l:=1 to 2 do begin put(v); {Tìm cách kết thúc tốt hơn trong 2 cách}
        u:=Doiso(l,L); {vào Paris qua cổng có lính loại nào gác?}
        if (Tr[u]>0) and (Smin>kc[u]) then {u đã cố định nhãn, và tối ưu hơn}
            begin
                kt:=u;

```



```

    Smin:=kc[u];
end;
end;
{nếu tồn tại hành trình tối ưu thì lần ngược vết để tìm hành trình ngược}
if Smin<maxlongint then begin
    u:=kt; {bắt đầu lần ngược từ Paris}
    slx:=0; {khởi trị số thành phố đi qua}
    repeat
        inc(slx);
        x[slx]:=u; {ghi nhận thành phố đi qua}
        u:=tr[u]; {lần ngược tiếp}
    until u=2*N+1; {gặp được quê hương của D'artagnan}
end;
end;
Procedure in_ketqua;
var u,i,l: longint;
begin
    assign(g,fo); rewrite(g);
    if Smin=maxlongint then {không tồn tại hành trình thì ghi -1}
        writeln(g,-1)
    else begin {tồn tại hành trình tối ưu thì}
        writeln(g,Smin); {ghi thời gian của hành trình tối ưu}
        for u:=slx downto 1 do begin {ghi hành trình}
            doidinh(x[u],i,l); {ra khỏi thành phố i qua cổng của loại L}
            write(g,i,' '); {ghi thành phố đi qua}
        end;
        writeln(g);
        for u:=slx downto 2 do {ghi số hiệu các con đường đi qua}
            write(g,a[x[u],x[u-1]],' ');
        end;
        close(g);
    end;
end;
BEGIN
    doc_dulieu;
    Dijsktra;
    timDuong;
    in_ketqua;
END.

```

Bài 29. Network - Mạng truyền tin

Một mạng truyền tin có N trạm và đường truyền tin hai chiều nối trực tiếp giữa các trạm. Chi phí truyền tin giữa hai trạm U và V là W . Một bức thông điệp cần phát đi từ trạm S đến trạm T thông qua các đường truyền trực tiếp. Để bảo đảm tính an toàn, phải truyền thông tin đi theo hai đường khác

nhau (hai đường được gọi là khác nhau nếu chúng không có đường truyền chung). Bài toán đặt ra là hãy tìm 2 đường đi với tổng chi phí thấp nhất.

Input. Dòng đầu tiên của tệp **Dữ liệu vào** chứa 4 số nguyên: N, M, S và T. Trong đó N là số trạm ($N \leq 200$), M là số đường truyền trực tiếp ($1 \leq M \leq N(N-1)/2$), S là trạm phát, T là trạm thu.

M dòng tiếp theo miêu tả các đường truyền trực tiếp gồm 3 số nguyên U, V và W thể hiện chi phí truyền giữa U và V là W ($W \leq 30000$).

Output. Dòng đầu tiên của tệp dữ liệu ra là tổng chi phí thấp nhất tìm được. Dòng thứ hai là đường đi thứ nhất, dòng thứ ba là đường đi thứ hai. Mỗi đường đi bắt đầu bằng S và kết thúc bằng T.

Ví dụ.

NETWORK . IN	NETWORK . OUT
6 8 1 2	25
1 4 10	1 4 6 2
1 5 1	1 5 3 2
1 2 100	
4 6 2	
6 2 1	
5 3 4	
3 2 7	
5 6 1	

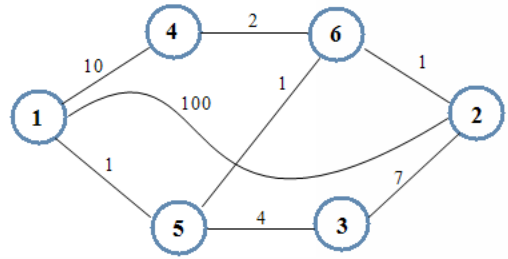
Gợi ý. Trước hết xây dựng nhãn đường đi ngắn nhất từ đỉnh S tới đỉnh T, theo thuật toán Dijkstra. Giả sử tìm được đường đi đó là:

$$S = d_1 \rightarrow d_2 \rightarrow \dots \rightarrow d_k = T \quad (1)$$

Sau đó tiếp tục xây dựng nhãn đường đi ngắn nhất ngược từ T về S. Trước khi thực hiện quá trình này cần lưu ý phải xóa các cạnh (d_i, d_{i+1}) ($i=1, 2, \dots, k-1$) trên đường đi (1) (để hai đường đi không có cạnh nào chung) và bổ sung thêm các cung (d_{i+1}, d_i) có trọng số bằng số đối của trọng số cạnh (d_i, d_{i+1}) để từ nhãn của d_{i+1} có được nhãn của d_i vẫn như cũ (khi xây dựng đường (1)). Quá trình xây dựng nhãn từ T về S phải thực hiện bằng thuật toán Ford_Bellman (hoặc thuật toán Floyd) vì đồ thị có trọng số âm.

Việc thêm các cung có trọng số âm như trên không sinh ra chu trình âm vì đường (1) là đường ngắn nhất.

Chú ý: Trong ví dụ nêu ở đề bài, kết quả là hai đường đi không chung cạnh, cả hai đường có thể đều không phải là đường ngắn nhất nối S và T, nhưng tổng độ dài của chúng là nhỏ nhất.



Chương trình.

```
const fi      = 'network.in';
      fo      = 'network.out';
      maxn    = 200;
type arr1     = array[1..maxn] of integer;
      arr2    = array[1..maxn, 1..maxn] of byte;
      arr3    = array[1..maxn] of longint;
      arr4    = array[1..maxn] of boolean;
      arr5    = array[1..maxn] of ^arr1;
var  f,g      : text;
      n,m,s,t : longint;
      father  : arr1;
      b       : arr2;
      d       : arr3;
      visit   : arr4;
      a       : arr5; {ma trận kề}
      ans     : longint;
```

```
Procedure init;
var i : longint;
begin
  for i:=1 to maxn do begin
    new(a[i]);
    fillchar(a[i]^, sizeof(a[i]^), 0);
  end;
end;
```

```
Procedure done;
var i : longint;
begin
  for i:=1 to maxn do dispose(a[i]);
end;
```

```
Procedure read_input;
var i, u, v, w : longint;
begin
  assign(f, fi); reset(f);
  readln(f, n, m, s, t); {số đỉnh, số cạnh, xuất phát và đích}
  for i:=1 to m do begin
    readln(f, u, v, w); {cạnh (u,v) có trọng số w}
```

```

    a[u]^[v] := w;
    a[v]^[u] := w;
end;
close(f);
end;
Procedure Dijsktra; {Thực hiện thuật toán Dijsktra tìm đường đi ngắn thứ nhất}
var i,j,min,last : longint;
begin
    for i:=1 to n do begin
        d[i] := maxlongint; {Khởi trị mảng d: nhãn đường đi}
        visit[i] := false; {Khởi trị mảng visit: đánh dấu chưa cố định nhãn }
    end;
    d[s] := 0; {Khởi trị nhãn đường đi của đỉnh xuất phát}
    visit[s] := true; {Đánh dấu cố định nhãn của s}
    last := s;
    while not visit[t] do begin {trong khi t tự do (chưa cố định nhãn) thì}
        for i:=1 to n do
            if (not visit[i]) and (a[last]^[i]<>0) {chọn i tự do kề với last}
            and (d[i]>d[last]+a[last]^[i]) then begin {nhãn i chưa tối ưu}
                d[i] := d[last]+a[last]^[i]; {thì sửa nhãn cho i}
                father[i] := last; {lưu vết: trước i là last}
            end;
        min := maxlongint; {chọn trong các đỉnh tự do, có nhãn nhỏ nhất là last}
        for i:=1 to n do
            if (not visit[i]) and (d[i]<min) then begin
                min := d[i];
                last := i;
            end;
        visit[last] := true; {cố định last}
    end;
    ans := d[t]; {độ dài đường thứ nhất}
end;
Procedure tracel; {Tìm hành trình thứ nhất bằng lần vết ngược từ t về s}
var i,j : longint;
begin
    fillchar(b, sizeof(b),0);
    i := t;
    while i<>s do begin
        j := father[i];
        a[i]^[j] := -a[i]^[j]; {xây dựng cung đối cho việc tìm hành trình sau}
        a[j]^[i] := 0; {xóa cung thuận trên hành trình thuận từ s tới t}
        b[j,i] := 1; {xác nhận có cung (j,i) trên hành trình thứ nhất}
        i := j;
    end;
end;
end;
```

Procedure Ford_Bellman; *{Thực hiện thuật toán Ford Bellman tìm đường đi ngắn nhất từ s tới t.}*

var i,j,k : longint;

begin

for i:=1 to n do begin *{Khởi trị mảng nhãn đường đi ngắn nhất, mảng vết}*

d[i] := maxlongint;

father[i] := 0;

end;

d[s] := 0; *{Khởi trị nhãn của đỉnh xuất phát}*

for k:=1 to n-1 do

for i:=1 to n do

for j:=1 to n do

if (a[i]^j<>0) and (d[i]<>maxlongint)

and (d[j]>d[i] + a[i]^j) then begin

d[j] := d[i] + a[i]^j;

father[j] := i;

end;

ans := ans + d[t]; *{tổng độ dài tối ưu của hai hành trình}*

end;

Procedure trace2; *{Tìm hành trình thứ hai}*

var i,j : longint;

begin

i := t;

while i<>s do begin

j := father[i];

b[j,i] := 1; *{xác nhận cung thuộc hành trình 2}*

if a[j]^i<0 then begin *{xóa bỏ cạnh đã có trong hành trình 1}*

b[j,i] := 0;

b[i,j] := 0;

end;

i := j;

end;

end;

Procedure find_path; *{ghi vào tệp output 2 hành trình}*

var i,j : longint;

begin

i := s;

while i<>t do begin

write(g,i,' ');

for j:=1 to n do

if b[i,j]=1 then break;

b[i,j] := 0;

i := j;

end;

writeln(g,t);

end;

```

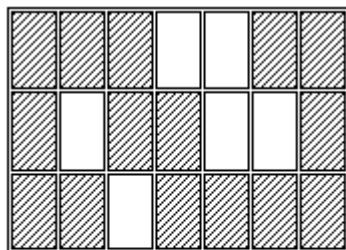
BEGIN
  assign(g,fo); rewrite(g);
  init;
  read_input;
  Dijkstra;
  tracel;
  Ford_Bellman;
  trace2;
  writeln(g,ans);
  find_path;
  find_path;
  close(g);
  done;

```

END.

Bài 30. MOVILAB (mê cung chuyển dịch)

Để cất giữ báu vật, quốc vương Sadama đã cho xây dựng một mê cung kỳ ảo. Muốn đến kho cất giữ báu vật cần phải đi qua mê cung này. Mê cung là một cung điện gồm N tầng được xây dựng ngầm dưới đất. Mỗi tầng là một dãy gồm M phòng. Có một số phòng chứa cầu thang và cửa thông phòng. Những phòng như vậy gọi là phòng nối. Các phòng còn lại được gọi là phòng cô lập. Có thể di chuyển theo chiều từ trên xuống dưới giữa các phòng nối có cầu thang hoặc sang phòng nối kề bên nếu có chung tường bên trái hoặc phải, đặc biệt không thể di chuyển từ tầng dưới lên tầng trên.

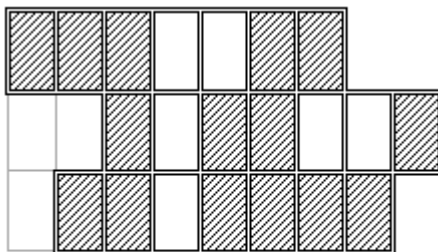


Hình 1. Kho báu vật

=====

Chẳng hạn, xét mê cung gồm 3 tầng, mỗi tầng gồm 7 phòng ($N=3$, $M=7$) như chỉ ra trong hình 1. Bạn dễ dàng nhận thấy là không thể đi qua mê cung này.

Tuy nhiên bí mật của mê cung là đối với mỗi tầng đều có cơ quan thực hiện việc dịch chuyển nó. Khi ấn nút cơ quan của một tầng, cả tầng này sẽ dịch chuyển đi



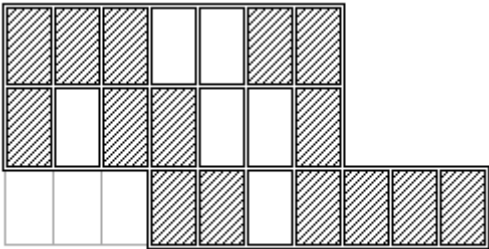
Hình 2. Dịch tầng một sang trái 1 đv
Dịch chuyển tầng 2 sang phải 1 đv

một khoảng cách (đơn vị tính là 1 phòng) nào đó theo chiều ngang sang trái hoặc sang phải. Khi dịch chuyển, các phòng giữ nguyên tính chất của nó. Do đó bằng việc dịch chuyển các tầng một cách thích hợp có thể tạo ra lối đi đến kho cất giữ báu vật.

Ví dụ: Đối với mê cung trong hình 1, ta có thể thực hiện việc dịch chuyển các tầng để đến kho báu vật (xem hình 2, hình 3).

Điều quan trọng cuối cùng là phải thực hiện việc dịch chuyển sao cho tổng khoảng cách dịch chuyển là nhỏ nhất, nếu không toàn bộ hệ thống sẽ bị phá hủy.

Dữ liệu: Vào từ file văn bản MVL.IN: Dòng đầu tiên chứa hai số nguyên N, M ($0 < N < 100$, $0 < M < 50$); N dòng tiếp theo mô tả các tầng của mê cung. Mỗi tầng được mô tả bởi M ký tự 'X' và '.' Ký tự 'X' cho biết phòng tương ứng là phòng cô lập còn ký tự '.' cho biết phòng tương ứng là phòng nối.



Hình 3. Dịch tầng ba sang phải 3 đv

Kết quả: Ghi ra file văn bản MVL.OUT tổng khoảng cách dịch chuyển nhỏ nhất tìm được. Ghi -1 nếu không có cách thực hiện dịch chuyển để có lối đi đến kho báu.

Ví dụ

MVL . IN	MVL . OUT	MVL . IN	MVL . OUT	MVL . IN	MVL . OUT
3 7 xxx . .xx x .xx . .x xx .xxxxx	2	5 5 xxx .x x .xxx xx . .x x .x .x xx .xx	3	5 5 xxx .x xxxxxx xx . .x x .x .x xx .xx	

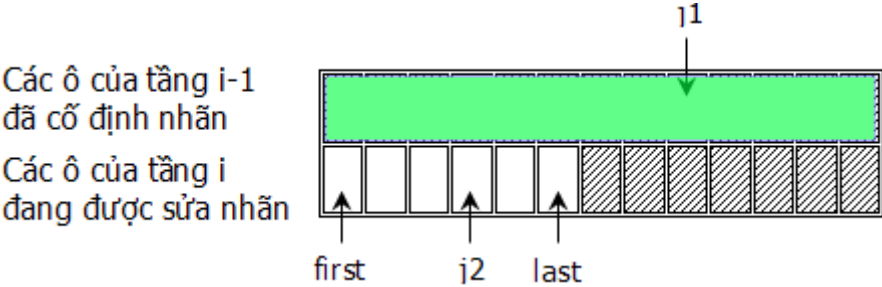
Gợi ý. Mỗi tọa độ (i,j) là một vị trí có thể tới của phòng j trên tầng i, khi tầng đó dịch chuyển sang trái hoặc sang phải thì $-m \leq j \leq 2m$. Coi mỗi tọa độ này là một đỉnh của đồ thị. Độ dài các cung từ (i-1,j1) đến (i,j2) chính là đoạn di chuyển tầng i sao cho có thể đi từ (i-1,j1) đến (i,j2). Rõ ràng, chúng

ta chỉ di chuyển tầng i sao cho một phòng nối nào đó của tầng i trùng một đoạn các “phòng nối” của tầng trên nó.

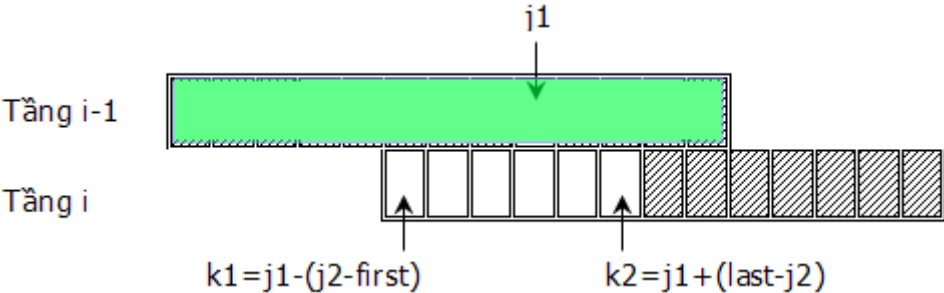
Sử dụng thuật toán tìm đường đi ngắn nhất trên đồ thị có hướng, không chu trình bằng cách gán nhãn và sửa nhãn các đỉnh là các tọa độ (i,j) lần lượt từ tầng $i=1$ đến tầng n (nhãn của các tọa độ tầng trên sửa nhãn cho các tọa độ tầng ngay dưới).

Ví dụ: Tầng i có các phòng nối liên tiếp từ cột $first$ đến cột $last$ di chuyển sao cho ô (i,j_2) đến (i,j_1) (với $j_2 \in [first, last]$) thì: Đặt $k_1=j_1-(j_2-first)$ và $k_2=j_1+(last-j_2)$. (xem hình 4 và hình 5 dưới đây). Từ các ô $(i-1,k_1)$ đến $(i-1,k_2)$ nếu đã có nhãn khác vô cùng đều có đường đi tới ô (i,j_1) . Do đó nhãn của (i,j_1) được sửa nhờ nhãn của các ô $(i-1,k)$ đã có nhãn khác vô cùng thuộc đoạn $[k_1,k_2]$ của tầng $i-1$.

Ngoài ra chú ý phải khởi trị nhãn ban đầu của các ô tầng 0 từ 1 đến m bằng 0, nhãn các ô của các tầng còn lại là vô cùng (các ô này nằm từ cột $-m$ đến cột $2m$).



Hình 4. Tầng i trước khi di chuyển



Hình 5. Tầng i sau khi di chuyển cho ô (i,j2) đến (i,j1)

Chương trình.

```
const fi = 'mvl.in';
      fo = 'mvl.out' ;
      maxm = 51 ;
      maxn = 101 ;
      maxc = 2000 ;
type tarr = array[1..maxm * maxn] of integer ;
var   c   : array[1..maxn , -maxm..2*maxm] of char ;
      d : array[0..maxn , -maxm..2*maxm] of word ;
      queue : array[1..2] of ^tarr ;
      n , m : integer ;
      f , g : text ;

procedure getmem ; {xin cấp phát bộ nhớ}
var u : integer ;
begin
  for u := 1 to 2 do begin
    new(queue[u]) ;
    fillchar(queue[u]^ , sizeof(queue[u]^) , 0) ;
  end ;
end ;

procedure enter ;
var i , j : integer ;
begin
  assign(f , fi) ; reset(f) ;
  assign(g , fo) ; rewrite(g) ;
  readln(f , n , m) ;
  fillchar(c , sizeof(c) , 'x') ; {Khởi trị các phòng đều là cô lập}
  for i := 1 to n do begin
    for j := 1 to m do
      read(f , c[i,j]) ; {Đọc đặc điểm phòng thứ j ở tầng i}
    readln(f) ;
  end ;
end ;

procedure init ;
var i , j : integer ;
begin
  for j := 1 to m do d[0,j] := 0 ; {Khởi trị nhân các đỉnh của tầng 0}
  for i := 1 to maxn do {Khởi trị nhân các đỉnh của các tầng từ 1 đến N là ∞}
    for j := -maxm to 2*maxm do d[i,j] := maxc ;
end ;

procedure solve ;
var i , j , first , last , j1, j2 : integer ;
    k                               : integer ;
begin
```

```

for i := 1 to n do begin {sửa nhãn dần từ tầng 1 đến tầng N}
  j := 1 ;
  repeat
    while (c[i,j] <>'.' ) and (j <= m) do inc(j) ;
    if j <= m then begin
      first := j ;
      while c[i,j] = '.' do inc(j) ;
      last := j-1 ;
      {Tầng i có đoạn các phòng nối từ cột first đến cột last}
      for j1 := -m to m*2 do begin
        for j2 := first to last do begin { di chuyển (i,j2) đến (i,j1)}
          for k:=j1-(j2-first) to j1+(last-j2) do
            if (-m<=k) and (k<=2*m) then
              if d[i-1,k]<>maxc then
                if d[i,j1]>d[i-1,k]+abs(j1-j2) then
                  d[i,j1]:=d[i-1,k]+abs(j1-j2)
            end ;
          end ;
          j:=last+1; {để lên đầu vòng lặp tìm đoạn phòng nối khác của tầng i}
        end ;
      until j > m ; {đến kho báu vật}
    end ;
  end ;
end ;
procedure result ;
var best : integer ;
    i : integer ;
begin
  best := maxc ;
  for i := -m to 2*m do {Tìm nhãn tối ưu nhất trong các ô tầng n}
    if d[n,i] < best then
      best := d[n,i] ;
    if best = maxc then writeln(g, -1)
    else writeln(g, best) ;
  close(g) ;
end ;
BEGIN
  getmem ;
  enter ;
  loang ;
  init ;
  solve ;
  result ;
END.

```

Bài 31. Thành phố trên sao hoả

Đầu thế kỷ 21, người ta thành lập một dự án xây dựng một thành phố trên sao Hoả để thế kỷ 22 con người có thể sống và sinh hoạt ở đó. Giả sử rằng trong thế kỷ 22, phương tiện giao thông chủ yếu sẽ là các phương tiện giao thông công cộng nên để đi lại giữa hai điểm bất kỳ trong thành phố người ta có thể yên tâm chọn đường đi ngắn nhất mà không sợ bị trễ giờ do kẹt xe. Khi mô hình thành phố được chuyển lên Internet, có rất nhiều ý kiến phản nản về tính hợp lý của nó. Đặc biệt, tất cả các ý kiến đều cho rằng hệ thống đường phố như vậy là quá nhiều, làm tăng chi phí xây dựng cũng như bảo trì.

Hãy bỏ đi một số đường trong dự án xây dựng thành phố thoả mãn:

- Nếu giữa hai địa điểm bất kỳ trong dự án ban đầu có ít nhất một đường đi thì sự sửa đổi này không làm ảnh hưởng tới độ dài đường đi ngắn nhất giữa hai địa điểm đó.
- Tổng độ dài của những đường phố được giữ lại là ngắn nhất có thể

Dữ liệu vào: file văn bản CITY.IN, chứa bản đồ dự án:

- Dòng thứ nhất ghi số địa điểm N và số đường phố m (giữa hai địa điểm bất kỳ có nhiều nhất là một đường phố nối chúng, $n \leq 200$; $0 \leq m \leq \frac{n(n-1)}{2}$);
- m dòng tiếp theo, mỗi dòng ghi ba số nguyên dương u, v, c cho biết có đường hai chiều nối giữa hai địa điểm u, v và độ dài của con đường đó là c ($c \leq 10000$).

Kết quả: file văn bản CITY.OUT, chứa kết quả sau khi sửa đổi:

- Dòng thứ nhất ghi hai số k, d . Trong đó k là số đường phố còn lại còn d là tổng độ dài của các con đường phố còn lại.
- k dòng tiếp theo, mỗi dòng ghi hai số nguyên dương p, q cho biết cần phải giữ lại con đường nối địa điểm p với địa điểm q .

Các số trên một dòng của các file CITY.IN, CITY.OUT được ghi cách nhau ít nhất một dấu cách.

Ví dụ:

CITY.IN	CITY.OUT
10 13	10 22
1 2 1	1 2
1 5 2	1 5
2 6 7	3 4
3 4 1	3 7
3 7 2	5 6
4 8 8	5 10
5 6 3	6 7
5 10 1	6 9
6 7 1	7 8
6 9 2	9 10
7 8 5	
7 10 8	
9 10 4	

Gợi ý:

Dùng thuật toán Floy xây dựng nhãn đường đi ngắn nhất giữa hai thành phố i, j bất kỳ. Sau đó đánh dấu loại bỏ con đường nối trực tiếp i và j nếu đường đi ngắn nhất giữa i và j không phải là con đường nối trực tiếp i và j . Sau đó, nếu không có dấu loại bỏ này thì đường đi ngắn nhất của các cặp thành phố i, j chính là độ dài con đường trực tiếp nối i và j , do đó tổng các nhãn này bằng tổng các con đường được giữ lại.

Chương trình:

```
{ $A+, B-, D+, E+, F-, G-, I+, L+, N+, O-, P-, Q-, R-, S+, T-, V+, X+ }
{ $M 16384, 0, 655360 }
const
  fi      = 'CITY.IN';
  fo      = 'CITY.OUT';
  maxN    = 200;
  vc      = 20000000;
type
  mangB = array[1..maxN] of byte;
  mangL = array[1..maxN] of longInt;
var
  f, g    : text;
  N, M    : longInt;
  a       : array[1..maxN] of ^mangL;
  b       : array[1..maxN] of ^mangB;
  tr      : array[1..maxN, 1..maxN] of byte;
  S, D    : longInt;
Procedure capphat;
var i: integer;
```

```

begin
    for i:=1 to maxN do new(a[i]);
    for i:=1 to maxN do new(b[i]);
end;
Procedure giaiphong;
var i: integer;
begin
    for i:=1 to maxN do Dispose(a[i]);
    for i:=1 to maxN do Dispose(b[i]);
end;
Procedure doc_dulieu;
var i,j,u,v,l: longInt;
begin
    assign(f,fi); reset(f);
    readln(f,N,M);
    for i:=1 to N do
        for j:=1 to N do a[i]^j:=vc; {Khởi trị mảng nhãn đường đi ngắn nhất}
    for i:=1 to N do
        for j:=1 to N do b[i]^j:=0; {Khởi trị mảng ma trận kề}
    for i:=1 to M do begin
        readln(f,u,v,L);
        a[u]^v:=L; a[v]^u:=L;
        b[u]^v:=1; b[v]^u:=1;
    end;
    close(f);
end;
Procedure floy; {Thực hiện thuật toán Floyd tìm nhãn đường đi ngắn nhất giữa 2
đỉnh bất kỳ}
var k,i,j: integer;
begin
    fillchar(tr,sizeof(tr),0); {Khởi trị mảng vết đường đi}
    for k:=1 to N do
        for i:=1 to N do
            for j:=1 to N do
                if a[i]^j>=a[i]^k+a[k]^j then begin
                    a[i]^j:=a[i]^k+a[k]^j;
                    tr[i,j]:=k;
                end;
    end;
end;
Procedure solve;
var i,j: longInt;
begin
    {Tìm các đường đi ngắn nhất giữa i và j thỏa điều kiện không là đường trực tiếp nối i và j}
    for i:=1 to N do
        for j:=1 to N do
            if (b[i]^j=1) and (tr[i,j]>0) then begin {thỏa điều kiện}

```

```

    b[i]^[j]:=0; {xóa bỏ đường nối trực tiếp i và j}
    b[j]^[i]:=0;
end;
S:=0;
D:=0;
{Tìm các đường đi trực tiếp còn lại và tổng độ dài của chúng}
for i:=1 to N-1 do
  for j:=i+1 to N do
    if b[i]^[j]=1 then begin {với dấu xóa bỏ b[i]^[j]=1, thì a[i]^[j] chính là độ
dài con đường trực tiếp nối i và j}
      S:=S+a[i]^[j];
      D:=D+1;
    end;
  end;
end;
Procedure in_ketqua;
var i,j: LongInt;
begin
  assign(g,fo); rewrite(g);
  writeln(g, D, ' ', S); {ghi: số con đường còn lại và tổng độ dài của chúng}
  for i:=1 to N-1 do {ghi các con đường giữ lại}
    for j:=i+1 to N do
      if b[i]^[j]=1 then writeln(g,i, ' ', j);
    close(g);
  end;
BEGIN
  capphat;
  doc_dulieu;
  floy;
  solve;
  in_ketqua;
  giaiphong;
END.

```

Bài 32. JOURN - Hành trình chẵn lẻ

Một mạng giao thông gồm N thành phố và các tuyến đường một chiều nối chúng. Các thành phố được đánh số từ 1 đến n ($1 < n \leq 100$). Sau mỗi ngày, mỗi tuyến đường của hệ thống đường một chiều này lại đổi chiều, trong các ngày lẻ ta có thể đi theo một chiều nào đó, còn trong các ngày chẵn ta lại chỉ được đi theo chiều ngược lại. Thời gian đi theo một tuyến đường được cho bởi một số nguyên dương (đơn vị thời gian tính bằng giờ).

Cần xác định hành trình nhanh nhất đi từ thành phố A đến thành phố B. Ngày đầu tiên của hành trình được coi là ngày lẻ. Hành trình của một

ngày không được kéo dài quá 12 giờ. Chỉ có thể nghỉ đêm ở các thành phố. Hành trình có thể tiếp tục ở ngày kế tiếp.

Dữ liệu: vào từ file JOURN.INP:

Dòng đầu tiên chứa hai số nguyên dương theo thứ tự là chỉ số của hai thành phố A và B;

Dòng tiếp theo chứa hai số n, k là số thành phố và số tuyến đường giữa các thành phố ($1 < k \leq 1000$);

Dòng thứ i trong số k dòng tiếp theo chứa ba số nguyên dương theo thứ tự là hai đầu mút và thời gian đi lại của tuyến đường i. Chiều đi lại trong ngày lẻ hướng từ thành phố thứ nhất đến thành phố thứ hai.

Kết quả: ghi ra file văn bản JOURN.OUT đường đi tìm được theo khuôn dạng sau:

Dòng đầu tiên ghi độ dài của hành trình tìm được;

Mỗi dòng tiếp theo chứa thông tin của một chặng đường bao gồm 4 số: thành phố xuất phát , thành phố kết thúc, chỉ số của ngày và độ dài của tuyến đường (các chặng được liệt kê liên tiếp theo hành trình).

Ví dụ:

JOURN . INP	JOURN . OUT	
1 3	28	
6 7	1 5 1 10	
1 2 9	5 4 1 1	
1 6 2	4 3 3 4	
1 5 10		
5 4 1		
4 6 2		
4 3 4		
2 3 5		

Gợi ý. Đây là bài toán tìm đường đi ngắn nhất trên đồ thị có hướng.

Lưu ý:

- Thời gian đi trong một ngày không vượt quá 12 giờ.

- Nếu là ngày lẻ, các đỉnh j kề với i nếu có cung (j,i) . Nếu là ngày chẵn, các đỉnh j kề với i nếu có cung (i,j) .
- Khi đến đỉnh i , để sửa nhãn cho các đỉnh j có thể chọn một trong 3 phương án: đi tiếp, nghỉ lại 1 ngày (vì thời gian còn lại của ngày hiện tại không đủ đi hết cung tiếp theo, nghỉ lại 2 ngày (chờ ngày thông đường trên cung ngược).

Chương trình.

```

const      fi                = 'journ.in1';
           fo                = 'journ.oul';
           max               = 101;
var        a,b,n,k          : integer;
           c                 : array[0..max,0..max] of integer;
           nh ,tr ,cx       : array[0..max] of integer;

```

Procedure read_input;

```

var        f                : text;
           i,x,y ,z         : integer;

```

begin

```
    assign(f,fi);reset(f);
```

```
    readln(f, a, b);{đỉnh xuất phát và kết thúc}
```

```
    readln(f, n, k);{số đỉnh, số cung}
```

```
    for i:= 1 to k do begin {đọc các cung}
```

```
        readln(f,x,y,z);
```

```
        if c[x,y]=0 then c[x,y] := z
```

```
        else
```

```
            if (c[x,y]>z) then c[x,y] := z
```

```
    end;
```

```
    close(f);
```

end;

function findmin : integer; {Tìm đỉnh có nhãn tự do nhỏ nhất}

```
var        i ,li            : integer;
```

```
           lval             : integer;
```

begin

```
    lval := maxint;
```

```
    li := 0;
```

```
    for i:= 1 to n do
```

```
        if cx[i] = 0 then {đỉnh i tự do}
```

```
            if nh[i] < lval then begin
```

```
                lval := nh[i];
```

```
                li := i;
```

```
        end;
```

```
    findmin := li;
```

end;


```

Procedure movenext(i : integer; ngay,gio : integer);
var      j : integer;
begin
  if odd(ngay) then begin {đang ngày lẻ}
    for j := 1 to N do
      if c[j,i] > 0 then {có đường đi từ j sang i}
        if gio + c[j,i] <= 12 then {điều kiện thời gian đi trong ngày }
          if ngay*12+gio+c[j,i]<nh[j] then begin {nhãn j chưa tối ưu}
            nh[j] := ngay*12 + gio + c[j,i]; {sửa nhãn cho j}
            tr[j] := i; {hư vết: trước j là i}
          end;
        end;
      end;
    else begin {đang ngày chẵn}
      for j := 1 to N do
        if c[i,j] > 0 then {đi được từ j sang i (vì đang ngày chẵn)}
          if gio + c[i,j] <= 12 then {còn đủ thời gian đi }
            if ngay*12+gio+c[i,j]<nh[j] then begin {nhãn j chưa tối ưu}
              nh[j] := ngay*12 + gio + c[i,j]; {sửa nhãn cho j}
              tr[j] := i; {hư vết: trước j là i}
            end;
          end;
        end;
      end;
    end;
  end;
Procedure solve(i : integer);
begin
  movenext(i, nh[i] div 12, nh[i] mod 12); {tới i rồi đi luôn}
  movenext(i, nh[i] div 12 + 1, 0); {tới i rồi nghỉ lại 1 ngày}
  movenext(i, nh[i] div 12 + 2, 0); {tới i rồi nghỉ lại 2 ngày}
end;
Procedure dijkstra; {Thực hiện thuật toán Dijkstra}
var      i : integer;
begin
  for i := 1 to n do nh[i] := maxint;
  i      := a;
  nh[i]  := 0; {Khởi trị nhãn đỉnh xuất phát}
  cx[a]  := 1; {Cố định nhãn đỉnh xuất phát}
  repeat
    solve(i); {sửa nhãn cho các đỉnh có thể đi tới i}
    i := findmin; {chọn đỉnh có nhãn tự do nhỏ nhất}
    if i = 0 then break
    else cx[i] := 1; {cố định nhãn của i}
  until cx[b] = 1; {cho đến khi đỉnh kết thúc có nhãn cố định}
end;
Procedure write_output;
var      i ,x,y,top : integer;

```

```

        cc          : integer;
        kq          : array[1..101] of integer;
        g          : text;
begin
    assign(g,fo); rewrite(g);
    for x := 1 to N do
    for   y := 1 to N do
    if c[x,y] = 0 then c[x,y] := c[y,x];
    if cx[b] = 0 then begin {vô nghiệm}
        write(g,0);
        close(g);
        halt;
    end;
    i := b; {Lần ngược vết hành trình }
    top := 0;
    while i > 0 do begin
        inc(top);
        kq[top] := i;
        i := tr[i];
    end;
    for i := top downto 2 do begin {ghi hành trình thuận}
        x := kq[i];
        y := kq[i-1];
        if odd((nh[y]-1) div 12) then cc :=c[y,x]
        else cc := c[x,y];
        writeln(g,x,' ',y,' ',(nh[y]-1) div 12+1,' ',cc);
    end;
    close(g);
end;
BEGIN
    clrscr;
    read_input;
    dijkstra;
    write_output;
END.

```

Bài 33. Toll - Lệ phí thấp nhất

Ngày xưa chàng thương nhân Sinbad thường phải chuyển hàng từ một địa điểm xuất phát tới một địa điểm đích nào đó, chàng phải chọn đi qua một số địa điểm trung gian. Các địa điểm này (kể cả địa điểm



xuất phát và địa điểm đích) là một thị trấn hoặc một làng quê. Khi vào thị trấn, cứ 20 đơn vị hàng phải nộp lệ phí là một đơn vị hàng; nếu chưa đủ 20 đơn vị hàng vẫn phải nộp 1 đơn vị hàng. Khi vào một làng quê, bất kể chuyển bao nhiêu đơn vị hàng, chỉ phải nộp 1 đơn vị hàng. Hãy viết một chương trình giúp Sinbad tìm đường đi tối ưu trả ít lệ phí nhất sao cho số hàng chuyển tới địa điểm đích còn đáp ứng được nhu cầu mua của địa điểm đích. Lưu ý: Khi ra khỏi một địa điểm không phải nộp lệ phí.

Input

Dữ liệu vào từ tệp SINBAD.IN chứa nhiều bộ test. Mỗi bộ test bao gồm hai phần: bản đồ đường đi và thông tin chi tiết về việc vận chuyển hàng (địa điểm xuất phát, địa điểm đích và số đơn vị hàng là nhu cầu mua của địa điểm đích). Với mỗi bộ test:

Dòng đầu tiên chứa số nguyên N không âm, là số lượng đường đi trên bản đồ.

N dòng tiếp theo, mỗi dòng chứa đúng 2 chữ cái tượng trưng cho 2 địa điểm có đường đi hai chiều trực tiếp nối với nhau; chữ cái hoa là thị trấn, chữ cái thường là làng quê. Bản đồ luôn đảm bảo có đường vận chuyển.

Dòng cuối cùng chứa số nguyên p ($0 < p \leq 1000$), là số đơn vị hàng cần chuyển tới địa điểm đích đáp ứng được nhu cầu mua, tiếp theo là hai chữ cái mô tả địa điểm xuất phát và địa điểm đích. Kết thúc tệp là dòng chứa số -1.

Output

Tệp ghi kết quả ra là SINBAD.OUT chứa nhiều dòng, mỗi dòng tương ứng với mỗi bộ test và có dạng:”Case x : q” trong đó x là số hiệu bộ test, q là số đơn vị hàng tối thiểu cần mang đi từ địa điểm xuất phát chuyển theo đường đi tối ưu để khi đến địa điểm đích còn đủ đơn vị hàng đáp ứng nhu cầu. Ví dụ

SINBAD.IN	SINBAD.OUT
1	CASE 1 : 20
aZ	CASE 2 : 44
19 aZ	
5	
AD	

DX	
Ab	
bc	
cX	
39 AX	
-1	

Gợi ý. Bài toán này thực chất là tìm đường đi ngắn nhất trên đồ thị có trọng số không âm, có thể dùng thuật toán Dijkstra. Tuy nhiên trọng số cạnh (i, j) chưa có sẵn, nó chính là lệ phí trả cho địa điểm j. Lệ phí này phụ thuộc vào lượng hàng đã chuyển vào i và loại địa điểm của j.

Ngoài ra một khó khăn nhỏ nữa là tên các đỉnh thuộc tập chữ cái ['a'..'z'] hoặc ['A'..'Z'] vì vậy cần mã số các đỉnh này.

Khó khăn lớn nhất khi giải bài toán này là phải dự kiến số đơn vị hàng bắt đầu mang đi từ địa điểm xuất phát sao cho trên hành trình, sau khi trả các lệ phí vẫn còn đáp ứng được nhu cầu mua của địa điểm đích. Có thể dùng phương pháp tìm kiếm nhị phân để tìm được số đơn vị hàng bắt đầu mang đi từ địa điểm xuất phát.

```
const fi = 'sinbad.in';
      fo = 'sinbad.out';
var    f, g : text;
      n : integer;
      a : array[1..52,1..52] of longint; {ma trận kề}
      p : longint; {số đơn vị hàng vào tới địa điểm đích}
      start, finish : byte; {địa điểm xuất phát và địa điểm đích}
      d : array[1..52] of longint; {nhãn đường đi tối ưu}
      t : array[1..52] of byte; {lưu vết đường đi tối ưu}
      visit : array[1..52] of byte; {đánh dấu các đỉnh}
      ntest : integer; {số test trong tệp input}
function id(x : char) : byte; {tạo số hiệu đỉnh x}
begin
    if (x>='a') and (x<='z') then id := ord(x)-96;
    if (x>='A') and (x<='Z') then id := ord(x)-64+26;
end;
Procedure find(pp: longint; s,f : byte);
{Tìm đường đi tối ưu khi chuyển pp đơn vị hàng từ địa điểm s tới địa điểm f}
var i,j, last : byte;
    min,c : longint;
    function cost(x: longint; i : byte): longint;
    {Hàm cho biết lệ phí cần trả khi mang x đơn vị hàng vào địa điểm i}
begin
```

```

    if (i>0) and (i<27) then cost := 1 {i là làng quê}
    else
    if (i>26) and (i<53) then {i là thị trấn}
    if x mod 20 =0 then cost := x div 20
    else cost := x div 20 + 1;
end;
begin
    {Thực hiện thuật toán Dijkstra tìm đường đi tối ưu khi mang pp đơn vị hàng từ địa điểm
s vào tới địa điểm f}
    last := s; {Khởi trị địa điểm cuối vừa nạp vào đường đi tối ưu}
    fillchar(visit, sizeof(visit), 0); {Khởi trị: các đỉnh còn tự do}
    fillchar(t, sizeof(t), 0); {Khởi trị vết}
    for i:=1 to 52 do d[i] := maxlongint; {Khởi trị nhãn độ dài tối ưu}
    visit[last] := 1; {cố định đỉnh last}
    t[last] := 0; {đỉnh trước của last coi là 0}
    d[last] := 0; {độ dài đường đi tối ưu đến giờ bằng 0}
    while visit[f]=0 do begin {thực hiện đến khi địa điểm đích cố định nhãn}
        for i:=1 to 52 do {sửa nhãn cho các đỉnh i còn tự do kề với đỉnh last}
        if (a[last,i]=1) and (visit[i]=0) then begin
            c := cost(pp, i); {lệ phí trả khi vào địa điểm i}
            if d[i]>d[last]+c then begin {điều kiện sửa nhãn cho đỉnh i}
                d[i] := d[last]+c;
                t[i] := last;
            end;
        end;
    end;
    {tìm đỉnh còn tự do có nhãn nhỏ nhất}
    min := maxlongint;
    for i:=1 to 52 do
    if visit[i]=0 then
    if d[i]<min then begin
        min := d[i];
        last := i;
    end;
    visit[last] := 1; {cố định nhãn đỉnh tìm được}
end;
end;
Procedure process; {Sử dụng phương pháp tìm kiếm nhị phân để tìm giá trị pp}
var pmax, pmin, pp, xp : longint;
begin
    pmin := p; {giá trị tối thiểu của pp}
    pmax := p + 26*((p div 20)+1) + 26; {giá trị tối đa của pp}
    repeat
        pp := (pmax + pmin) div 2; {giá trị giữa}
        find(pp, start, finish); {Tìm đường chuyển pp đ vị từ start vào finish}
        xp := pp-d[finish]; {hàng chuyển vào được finish}

```

```

    if (xp>p) then pmax:= pp {nếu đáp ứng nhu cầu mua thì giảm pmax}
    else pmin:= pp; {chưa đáp ứng thì tăng pmin}
until xp=p; {cho đến khi đáp ứng nhu cầu tối thiểu là p}
end;
Procedure work;
var i, j : integer; x,y,z : char;
begin
    assign(g,fo); rewrite(g);
    ntest := 1; {Khởi trị số test là 1}
    assign(f,fi); reset(f);
    readln(f,n); {số con đường trong một test}
    while n>0 do begin {đọc các con đường trong một test}
        for i:=1 to 52 do {Khởi trị ma trận kề}
            for j:=1 to 52 do a[i,j] := maxlongint;
            for i:=1 to n do begin
                readln(f,x,y); {Đọc một con đường}
                a[id(x),id(y)] := 1; a[id(y),id(x)] := 1;
            end;
            readln(f,p,z,x,y); {Đọc thông tin chi tiết về việc vận chuyển hàng. Chú ý
biến z dùng để lấy dấu cách giữa số p và kí tự x}
            start := id(x); finish := id(y);
            process; {Tìm kiếm nhị phân cho lượng hàng cần mang đi từ điểm xuất phát}
            writeln(g,'CASE ',ntest,' : ',p+d[finish]); {Ghi kết quả một test}
            readln(f,n); inc(ntest); {sang test tiếp theo}
        end;
        close(f); close(g);
    end;
BEGIN
    work;
END.

```

Bài 34. Hai thang máy

Trong trung tâm thương mại cao 101 tầng (đánh số từ 0 đến 100) khách hàng có thể sử dụng hai loại thang máy:

Thang loại I: cho phép di chuyển đến bất kì tầng nào với thời gian di chuyển một tầng là E_1 giây.

Thang máy loại II: chỉ dừng ở các tầng có chỉ số chia hết cho 10, thực hiện di chuyển qua 10 tầng với thời gian E_2 giây.

Bất kể thang máy ở tầng nào, thời gian chờ đợi thang máy loại I và loại II tương ứng là W_1 và W_2 giây (để chuyển thang máy hoặc chờ vào thang máy). Ngoài ra tại mỗi tầng, khách hàng còn có thể di chuyển từ tầng

này lên tầng trên hoặc xuống tầng dưới theo cầu thang cố định với thời gian là S giây.

Yêu cầu: Xác định thời gian nhỏ nhất T cần thiết để một khách hàng có thể di chuyển từ tầng X đến tầng Y. Giả thiết là $1 \leq E_1, E_2, W_1, W_2, S \leq 1000$.

Dữ liệu vào từ file văn bản TM.IN:

Dòng đầu tiên chứa hai số E_1, W_1 ;

Dòng thứ hai chứa hai số E_2, W_2 ;

Dòng thứ ba chứa số S;

Dòng thứ tư chứa hai số X và Y.

Kết quả ra file văn bản TM.OUT: thời gian T tìm được.

Các số trên cùng một dòng của TM.IN được ghi cách nhau ít nhất một dấu cách.

Ví dụ :

TM. IN	TM. OUT
2 25	96
4 15	
10	
85 43	

Cách di chuyển tối ưu trong ví dụ trên như sau :

Đang ở tầng 85, chờ thang máy loại I		: 25 giây
Xuống tầng 80 :	2giây x 5	: 10 giây
Chờ thang máy loại II		: 15 giây
Xuống tầng 40	4 giây x 4	: 16 giây
Di chuyển theo cầu thang lên tầng 43	10 giây x 3	: 30 giây
Tổng cộng		: 96 giây

Gợi ý. Xây dựng đồ thị có hướng, mỗi đỉnh i là cặp số $(\text{floor}_i, \text{lift}_i)$ với ý nghĩa: đang ở tầng có số hiệu floor_i và vừa đi đến bằng loại thang lift_i . Để biết trọng số đi từ đỉnh i $(\text{floor}_i, \text{lift}_i)$ đến đỉnh j $(\text{floor}_j, \text{lift}_j)$ tốn bao nhiêu

thời gian, cần biết tầng floor_i và floor_j thuộc loại tầng nào ($0 \leq \text{floor}_i, \text{floor}_j \leq 100$) và chọn thang l là loại thang nào ($0 \leq l \leq 2$) để tới floor_j .

Kí hiệu loại thang: thang cố định là $l=0$, thang máy loại một là $l=1$, thang máy loại hai là $l=2$.

Đỉnh xuất phát có thể chọn là $(x,0)$ trong đó x là tầng xuất phát, loại thang đi tới tầng xuất phát có thể chọn là loại tùy ý không ảnh hưởng gì tới trọng số các đỉnh kề với đỉnh xuất phát vậy chọn là $l=0$. Đỉnh đích là (y, lift) trong đó lift có thể là 0, 1 hoặc 2.

Trong quá trình thực hiện thuật toán Dijkstra, dựa vào giả thiết của bài toán chúng ta có thể xét dần các đỉnh thích hợp, tính trọng số các cạnh cần thiết và xây dựng nhãn đường đi ngắn nhất của các đỉnh như chương trình dưới đây:

```
const fi    = 'tm.in';
      fo    = 'tm.out';
      n     = 100;
      maxd  = 10000000;
var e, w: array[0..2] of real;
    t   : real;
    d   : array[-10..n + 10, 0..2] of real; {nhãn đường đi ngắn nhất}
    tudo: array[0..n, 0..2] of boolean; {đỉnh tự do hay đã cố định nhãn}
    x, y : integer; {số hiệu tầng xuất phát và tầng đích}
Procedure read_input;
var f : text;
begin
  assign(f, fi); reset(f);
  readln(f, e[1], w[1]); {thời gian di chuyển và chờ vào thang máy loại 1}
  readln(f, e[2], w[2]); {thời gian di chuyển và chờ vào thang máy loại 2}
  readln(f, e[0]); {thời gian di chuyển theo thang cố định}
  w[0] := 0; {thời gian chờ tại thang cố định coi là 0}
  readln(f, x, y); {tầng xuất phát và tầng đích}
  close(f);
end;
Procedure init;
var i, j : integer;
begin
  fillchar(tudo, sizeof(tudo), true); {đánh dấu các đỉnh còn tự do}
  for i := 0 to n do
    for j := 0 to 2 do d[i, j] := maxd; {Khởi trị nhãn đường đi}
  d[x, 0] := 0; {nhãn đường đi của đỉnh xuất phát (x,0) của đồ thị coi bằng 0}
```


end;

Procedure tim_min(var tang, thang: integer); *{tìm lượng sửa nhĩn}*

var i, j : integer;

mind : real;

begin

mind := maxd; *{Khởi trị lượng sửa nhĩn}*

for i:=0 to n do *{Duyệt các đỉnh (i,j), i: tầng, j: loại thang vừa dùng để tới i}*

for j:=0 to 2 do

if tudo[i, j] and (d[i, j] < mind) then begin *{đỉnh tự do, nhĩn nhỏ hơn thì ghi nhận lưu vào hai biến: tang và thang}*

mind := d[i, j];

tang := i;

thang := j;

end;

end;

function min(a, b : real) : real;

begin if a < b then min := a else min := b;end;

Procedure sua_nhan(tang, thang: integer); *{sửa nhĩn cho các đỉnh tự do kề với đỉnh (tang, thang)}*

var L : integer;

begin

for L := 0 to 2 do *{chọn loại thang đi tiếp là l}*

if tudo[tang, L] then *{đỉnh kề với (tang, thang) còn là đỉnh tự do}*

{trước hết thêm thời gian chờ đợi khi chuyển loại thang}

d[tang, L] := min(d[tang, L], d[tang, thang] + w[L]);

{sau đó thêm thời gian di chuyển trên cùng một loại thang máy}

if thang < 2 then begin *{đi trên cầu thang hoặc thang máy loại 1}*

d[tang-1, thang] := min(d[tang-1, thang], d[tang, thang] + e[thang]);

d[tang+1, thang] := min(d[tang+1, thang], d[tang, thang] + e[thang]);

end

else *{đi trên thang máy loại 2}*

if tang mod 10 = 0 then begin

d[tang-10, 2] := min(d[tang-10, 2], d[tang, 2] + e[2]);

d[tang+10, 2] := min(d[tang+10, 2], d[tang, 2] + e[2]);

end;

end;

Procedure dijkstra;

var tang, thang : integer;

begin

repeat

tim_min(tang, thang);

if tang = y then break; *{đã tới đỉnh đích là (y, thang)}*

tudo[tang, thang] := false;

sua_nhan(tang, thang);

until false;

t := d[tang, thang]; *{thời gian ngắn nhất}*

```

end;
Procedure result;
var f : text;
begin
    assign(f, fo); rewrite(f);
    if frac(t) = 0 then writeln(f, t:1:0)
    else writeln(f, t);
    close(f);
end;
BEGIN
    read_input;
    init;
    dijkstra;
    result;
END.

```

Bài 35. Trains

Có N thành phố và M đường ray hai chiều giữa một số cặp thành phố nào đó, các đường ray được quản lý bởi 16 hãng đường sắt. Các thành phố được đánh số từ 1 đến N ($N \leq 100$) và các hãng được đánh số từ 1 tới 16.

Được biết chi phí đi tàu trực tiếp giữa hai thành phố i, j bất kì (nếu có đường ray) là C. Nếu đang đi tàu của một hãng đến nhà ga nào đó rồi chuyển sang tàu của hãng khác thì sẽ phải mất thêm một khoản phụ phí A.

Yêu cầu : Cho trước hai thành phố S và F, hãy tìm hành trình đi tàu từ thành phố S đến thành phố F với chi phí ít nhất. Với giả thiết rằng luôn luôn tồn tại cách đi tàu từ S tới F.

Dữ liệu vào từ file văn bản TRAINS.INP:

Dòng thứ nhất ghi sáu số nguyên dương N, M, C, A, S, F ($1 \leq A, C \leq 100$);

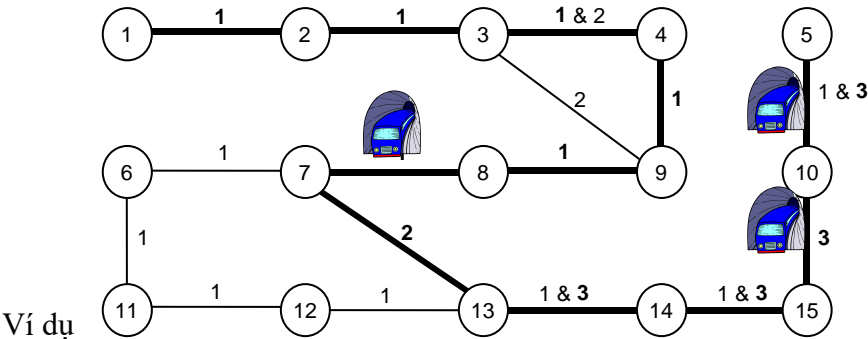
M dòng tiếp theo, mỗi dòng có dạng u v k1 k2 ... cho biết rằng giữa thành phố u và thành phố v có đường ray và k1, k2, ... là số hiệu các hãng sở hữu đường ray đó.

Kết quả ghi ra file văn bản TRAINS.OUT:

Dòng thứ nhất ghi chi phí tối thiểu phải trả;

Các dòng tiếp theo, mỗi dòng ghi bộ ba i, j, k. thể hiện tại bước đó sẽ đi từ thành phố i đến thành phố j bởi tàu của hãng k. Thứ tự các dòng phải theo đúng thứ tự đi trong hành trình.

Các số trên một dòng của các file input và output ghi cách nhau ít nhất một dấu cách.



TRAINS . INP						TRAINS . OUT		
15	16	3	2	1	5	37		
1	2			1		1	2	1
2	3			1		2	3	1
3	4			1	2	3	4	1
3	9			2		4	9	1
4	9			1		9	8	1
5	10			1	3	8	7	1
6	7			1		7	13	2
6	11			1		13	14	3
7	8			1		14	15	3
7	13			2		15	10	3
8	9			1		10	5	3
10	15			3				
11	12			1				
12	13			1				
13	14			1	3			
14	15			1	3			

Gợi ý. Xét đồ thị có hướng, mỗi đỉnh là cặp số (city, company) thể hiện vừa đi tàu của hãng *company* đến thành phố *city*. Các đoạn đường được quản lý bởi mảng *road* gồm các bản ghi. Mỗi bản ghi có bốn trường: *road.x* và *road.y* là hai thành phố ở hai đầu con đường, *road.num* là số lượng hãng quản lý con đường này, *road.company* là mảng chứa số hiệu các hãng quản lý con đường này.

Các đỉnh xuất phát là (start, j) trong đó *start* là thành phố xuất phát, j là hãng có tàu vừa đi đến thành phố *start* (có thể khởi trị j nhận mọi giá trị từ 0 đến 16). Dùng thuật toán Dijkstra tìm đường đi ngắn nhất từ đỉnh xuất phát tới đỉnh kết thúc. Chú ý kỹ thuật tìm đỉnh kề với một đỉnh cho trước bằng sử dụng mảng *road*.

```

const fi      = 'trains.inp';
      fo      = 'trains.out';
type  int     = integer;
      arr1    = array[1..16] of int;
      typeroad = record
                                x,y      : int;
                                num      : int;
                                company   : arr1;
      end;
var   n,m,c,a,start,finish: int; {n,m,c,a,s,f với ý nghĩa như trong đề bài}
      road : array[1..1000] of typeroad; {quản lý các cạnh của đồ thị}
      l,    {nhân đường đi ngắn nhất của các đỉnh}
      tracecity, {lưu vết thành phố trên đường đi ngắn nhất}
      tracecompany : array[1..100,1..16] of int; {lưu vết chọn
hãng tàu trên đường đi ngắn nhất}
      visit: array[1..100,1..16] of 0..1; {đánh dấu đỉnh tự do}
      companyfinish: int; {hãng tàu được chọn trên đoạn đường cuối cùng}
Procedure read_input;
var   f      : text; i      : int;
begin
  assign(f,fi);reset(f);
  readln(f,n,m,c,a,start,finish); {đọc từ tệp input các giá trị n,m,c,a,s,f}
  for i:=1 to m do begin {đọc các con đường và các hãng quản lý chúng}
    read(f,road[i].x, road[i].y); {hai thành phố được nối }
    road[i].num:=0; {khởi trị số lượng hãng quản lý con đường}
    while not seekeoln(f) do begin {đọc các hãng quản lý con đường}
      inc(road[i].num); {đếm số lượng hãng}
      read(f, road[i].company[road[i].num]); {đọc số hiệu của hãng}
    end;
    readln(f);
  end;
  close(f);
end;
Procedure khoitao;
var   i,j    : int;
begin
  for i:=1 to n do

```

```

for j:=1 to 16 do l[i,j]:=maxint; {Khởi trị nhãn đường đi ngắn nhất}
{Khởi trị nhãn đường đi ngắn nhất của các đỉnh xuất phát}
for j:=1 to 16 do begin
    l[start,j]:=0;
    tracecity[start,j]:=start;
    tracecompany[start,j]:=j;
end;
fillchar(visit,sizeof(visit),0); {Khởi trị đánh dấu các đỉnh tự do}
end;
Procedure timtp(var city,company:int); {Tìm đỉnh tự do có nhãn nhỏ nhất}
var i,j,min,xcity,xcompany :int;
begin
    min:=maxint; {Khởi trị số min để tìm nhãn nhỏ nhất}
    for i:=1 to n do {Duyệt các đỉnh (i,j): i là thành phố, j là hãng có tàu tới i}
        for j:=1 to 16 do
            if visit[i,j]=0 then {đỉnh (i,j) là đỉnh tự do}
                if l[i,j]<min then {có nhãn nhỏ hơn min thì lưu lại giá trị mới của min}
                    begin
                        min:=l[i,j]; {Lưu min}
                        xcity :=i; {Lưu đỉnh (i,j) có nhãn nhỏ hơn các đỉnh xét trước}
                        xcompany:=j;
                    end;
                city:=xcity; {lưu lại đỉnh tự do có nhãn nhỏ nhất là (city, company)}
                company:=xcompany;
            end;
Procedure suanhan(xcity, xcompany : int);
var cost,i,t,tp : int;
begin
    cost:=l[xcity,xcompany]; {nhãn của đỉnh (xcity,xcompany)}
    {Tìm mọi đỉnh tự do kề với đỉnh (xcity, xcompany)}
    for i:=1 to m do with road[i] do {xét mọi đoạn (road[i].x, road[i].y)}
        if (x=xcity) or (y=xcity) then begin {có một đầu là xcity}
            if x<>xcity then tp:=x else tp:=y; {và đầu kia là tp }
            for t:=1 to num do {duyet mọi hãng quản lý đoạn ( road[i].x,road[i].y)}
                if visit[tp,company[t]]=0 then {đỉnh (tp, company[t]) là tự do }
                    if (company[t]<>xcompany) {chọn chuyển hãng tàu }
                        and (cost+a+c<l[tp,company[t]]) then begin {tối ưu hơn}
                            l[tp,company[t]]:=cost+a+c; {thì sửa nhãn cho đỉnh tự do này}
                            tracecity[tp,company[t]]:=xcity; {vết thành phố trước tp là city}
                            tracecompany[tp,company[t]]:=xcompany; {hãng trước: xcompany}
                        end
                    else
                        if (company[t]=xcompany) {không chuyển hãng tàu}

```

```

        and (cost+c<l[tp,company[t]]) then begin {tối ưu hơn}
        l[tp,company[t]] := cost + c;
        tracecity[tp,company[t]] := xcity;
        tracecompany[tp,company[t]] := xcompany;
    end;
end;
end;
Procedure dijkstra; {Thực hiện thuật toán tìm đường đi ngắn nhất}
var city, company: int;
begin
    repeat {Thực hiện vòng lặp cho tới khi gặp thành phố đích}
        timtp(city,company); {Tìm đỉnh (city, company) tự do có nhãn nhỏ nhất}
        if city=0 then break; {Nếu không tìm được thì thoát}
        suanhan(city,company); {Sửa nhãn cho các đỉnh kề với (city, company)}
        visit[city,company] := 1; {Cố định nhãn của đỉnh (city,company)}
    until city = finish; {kết thúc vòng lặp}
    companyfinish := company; {ghi nhận lại hãng tàu trên đoạn cuối cùng}
end;

Procedure write_output;
var g : text;
    city,company,count,i : int;
    city1,city2,arrcompany : array[1..100] of int;
begin
    assign(g,fo);rewrite(g);
    writeln(g,l[finish,companyfinish]); {ghi chi phí tối thiểu phải trả}
    {Quá trình lần ngược hành trình từ thành phố đích về thành phố xuất phát, lưu vào các mảng city1,city2,arrcompany}
    city := finish;
    company := result;
    count := 0;
    repeat
        inc(count); {đếm các đoạn đường (i,j) của hành trình ngắn nhất}
        city1[count] := tracecity[city,company]; {i}
        city2[count] := city; {j}
        arrcompany[count] := company; {hãng tàu trên đoạn đường (i,j)}
        {phục vụ chuyển sang đoạn đường trước}
        company := tracecompany[city,company];
        city := city1[count];
    until city=start;
    for i:=count downto 1 do {ghi lại hành trình có chi phí tối thiểu}
        writeln(g,city1[i], ' ', city2[i], ' ', arrcompany[i]);
    close(g);
end;
BEGIN

```

```

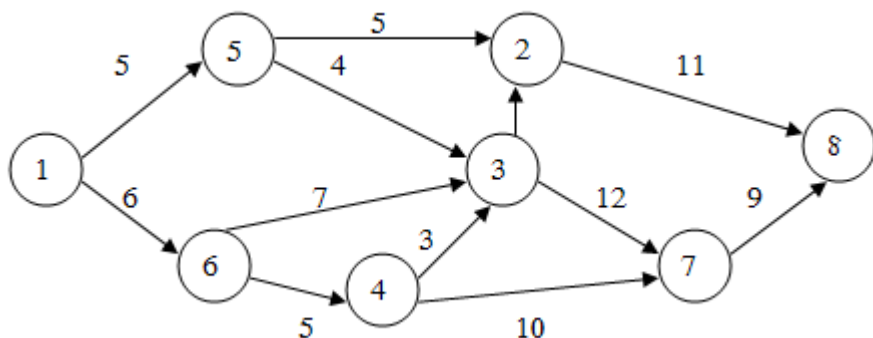
read_input;
khoitao;
dijkstra;
write_output;
END.

```

Bài 36. Hike - Cuộc hành quân dã ngoại

Một đoàn học sinh tổ chức hành quân dã ngoại. Có N địa điểm ($2 \leq N \leq 100$) và thời gian đi lại giữa hai địa điểm là một số nguyên dương không quá 100 được thông báo cho mọi học sinh trong đoàn. Các địa điểm được đánh số từ 1 đến N . Sơ đồ dưới đây cho một ví dụ với $N=8$.

Mỗi học sinh trong đoàn đều cùng xuất phát từ địa điểm 1 và đi theo một con đường nào đó đến địa điểm N . Tại mỗi giao lộ, đoàn học sinh lại phải phân ra thành các nhóm, mỗi nhóm đi theo một tuyến đường rẽ nhánh. Chẳng hạn ở địa điểm 1, nhóm được phân thành hai nhóm con, một nhóm đi theo tuyến đường dẫn đến địa điểm 5, còn một nhóm tiến đến địa điểm 6. Nhóm học sinh đến địa điểm 5 lại chia thành hai nhóm con một nhóm đến địa điểm 2 còn nhóm kia đến địa điểm 3... Nghĩa là, tất cả các tuyến đường đều được khảo sát.



Do số lượng tuyến đường là cố định nên ta giả thiết là tại mỗi giao lộ có đủ học sinh để phân chia thành các nhóm con đòi hỏi để tiếp tục khảo sát. Để bảo đảm an toàn, nếu nhóm đến một địa điểm nào đó sớm hơn thì nhóm đó cần phải chờ tất cả các nhóm khác đến đông đủ rồi mới phân chia thành nhóm con tiếp tục hành trình. Chẳng hạn, giả sử thời điểm bắt đầu cuộc

hành quân là 0, nhóm đầu tiên đến địa điểm 3 vào thời điểm 9 (nhóm đi qua địa điểm 5), nhóm này phải chờ hai nhóm đến từ địa điểm 6 và 4. Khi ba nhóm hội đủ, họ mới phân thành hai nhóm tiếp tục đi đến địa điểm 2 và 7.

Chú ý là chỉ có duy nhất một địa điểm xuất phát đó là địa điểm 1 và chỉ có duy nhất một địa điểm đích là N. Từ địa điểm xuất phát luôn có đường đi đến bất cứ địa điểm nào trong số các địa điểm còn lại, đồng thời từ một địa điểm bất kỳ luôn có đường đi đến địa điểm đích. Giả thiết là không có chu trình.

Yêu cầu: Cần tính thời điểm sớm nhất T khi nhóm cuối cùng về đích N, giả thiết thời điểm bắt đầu cuộc hành quân là 0. Trong ví dụ trên $T=35$, nghĩa là thời điểm sớm nhất để nhóm cuối cùng về đến đích là 35.

Do mỗi nhóm khi đến một địa điểm nào đó phải chờ một số nhóm khác đến đông đủ mới có thể tiếp tục cuộc hành trình, nên ở đây xuất hiện thời gian chờ đợi. Ta gọi thời gian chờ đợi ở mỗi địa điểm là khoảng thời gian từ thời điểm nhóm đầu tiên đến địa điểm này đến thời điểm nhóm cuối cùng đến địa điểm này. Ví dụ tại địa điểm 3 nhóm đầu tiên (nhóm đi qua 5) đến địa điểm 3 tại thời điểm 9, còn nhóm cuối cùng đến địa điểm 3 tại thời điểm 14 (nhóm đi qua 6 và 4). Do đó thời gian chờ đợi của địa điểm 3 là 5. Tương tự như vậy, thời gian chờ đợi của địa điểm 2 là 13.

Bạn cần tính tổng thời gian chờ của tất cả các địa điểm. Trong ví dụ đang xét, tổng thời gian chờ đợi là 24.

Tại một số địa điểm, các nhóm học sinh khi đã hội hợp đông đủ cũng không nhất thiết phải tiếp tục hành trình ngay tức khắc. Họ có thể cùng nhau vui chơi một thời gian ở địa điểm này mà vẫn có thể đến đích không muộn hơn thời điểm T . Ví dụ, tại địa điểm 5, các nhóm hội đủ ở đây vào thời điểm 5, và có thể vui chơi tại đó cho đến thời điểm 10 mới xuất phát mà vẫn đến đích 8 không muộn hơn thời điểm 35-thời điểm sớm nhất mà nhóm cuối cùng đến đích. Tương tự, khi các nhóm đã hội đủ tại địa điểm 2, họ có thể vui chơi trong 1 đơn vị thời gian rồi tiếp tục hành trình. Tuy nhiên tại tất cả các địa điểm còn lại, các nhóm khi đã tập hợp đông đủ thì phải ngay lập tức phân nhóm tiếp tục hành trình. Như vậy, trong ví dụ đang xét chỉ có hai địa

điểm mà tại đó các nhóm học sinh có thể nghỉ ngơi trước khi tiếp tục hành trình.

Bạn cần xác định số lượng địa điểm mà tại đó các nhóm học sinh có thể nghỉ ngơi trước khi tiếp tục hành trình (gọi là điểm dừng).

Dữ liệu vào từ file văn bản HIKE.INP:

Dòng đầu tiên chứa số nguyên N ($2 \leq N \leq 100$) là số địa điểm, và số nguyên M ($1 \leq M \leq 1000$) là số cung đường;

Dòng thứ i trong số M dòng tiếp theo chứa 3 số nguyên: địa điểm đầu, địa điểm cuối và thời gian đi từ địa điểm đầu đến địa điểm cuối. Các giá trị số trên một dòng được ghi cách nhau bởi dấu trống. Thời điểm bắt đầu cuộc hành quân là 0. Địa điểm xuất phát là 1, địa điểm kết thúc là N .

Kết quả ghi trên một dòng của file văn bản HIKE.OUT 3 số nguyên theo thứ tự là thời điểm sớm nhất mà nhóm cuối cùng có thể đến đích, tổng thời gian chờ đợi của các địa điểm, số lượng điểm dừng.

Ví dụ:

HIKE.INP	HIKE.OUT
8 12	35 24 2
3 7 12	
5 2 5	
6 3 7	
1 6 6	
4 7 10	
2 8 11	
1 5 5	
5 3 4	
6 4 5	
7 8 9	
4 3 3	
3 2 9	

Gợi ý.

Thời gian của nhóm học sinh đến địa điểm i muộn nhất kí hiệu là $lmax[i]$. Có thể áp dụng tương tự thuật toán Ford Bellman tìm nhãn thời gian dài nhất đi từ điểm xuất phát tới mỗi điểm.

Căn cứ vào nhãn thời gian dài nhất cho các đỉnh, có thể tìm nhãn thời gian ngắn nhất cho các đỉnh (là thời gian đến đỉnh này của nhóm học sinh đến sớm nhất).

Thời gian sớm nhất tất cả học sinh có mặt tại địa điểm N là $l_{\max}[N]$.

Tổng thời gian chờ tại các địa điểm là tổng: $\sum_{i=1}^n (l_{\max}[i] - l_{\min}[i])$

Số lượng các điểm dừng bằng $N-k$, với k là số lượng các điểm “không là điểm dừng” (gọi tắt là điểm không dừng). Có thể dùng thuật toán loang và tổ chức hàng đợi, tìm các điểm không dừng như sau: từ điểm không dừng cuối cùng là điểm N, tìm được các điểm không dừng ngay trước đó. Giả sử i là một điểm không dừng đã nạp vào hàng đợi và được lấy ra để tìm các điểm không dừng j trước nó thì quan hệ giữa i và j như sau :

$$L_{\max}[j] + a[j,i] = L_{\max}[i] \quad (a[j,i] \text{ là thời gian đi từ } j \text{ tới } i).$$

Chương trình.

```
uses crt;
const max      = 100;
      fi       = 'hike.inp';
      fo       = 'hike.out';
type m1        = array[1..max] of integer;
      m2        = array[1..max,1..max] of integer;
var a          : m2;
    lmin, lmax  : m1;
    n           : byte;
Procedure input;
var f          : text;
    i,j,num    : integer;
    k,m        : integer;
begin
  fillchar(a,sizeof(a),0);
  assign(f,fi); reset(f);
  readln(f,n,m); {Số đỉnh, số cạnh}
  for k:=1 to m do begin
    readln(f,i,j,num); {thời gian đi trên cung (i,j) là num}
    a[i,j]:=num;
  end;
  close(f);
end;
Procedure creat_max; {Tạo nhãn thời gian dài nhất từ đỉnh xuất phát tới các đỉnh}
var k,i,j      : integer;
```

begin

```
lmax[1] := 0; {Khởi trị nhãn của đỉnh xuất phát}
for i:=2 to n do lmax[i]:=-maxint; {Khởi trị nhãn các đỉnh còn lại}
for k:=1 to n-1 do {Sửa nhãn n-1 lần tương tự thuật toán Ford Bellman}
for i:=1 to n do
for j:=1 to n do
if a[i,j]>0 then
if lmax[i]+a[i,j]>lmax[j] then
lmax[j] := lmax[i]+a[i,j]; {Sửa lại nhãn của j}
```

end;

Procedure creat_min; {Tạo nhãn thời gian ngắn nhất từ đỉnh xuất phát tới các đỉnh j, phụ thuộc vào nhãn thời gian dài nhất của các đỉnh i có cung (i,j)}

var i,j : integer;

begin

```
lmin[1] := 0; {Khởi trị nhãn của đỉnh xuất phát}
for i:=2 to n do lmin[i] := maxint; {Khởi trị nhãn các đỉnh còn lại}
for i:=1 to n do begin {Duyệt các cung (i,j) xuất phát từ i}
for j:=1 to n do
if a[i,j]>0 then
if lmax[i] + a[i,j] < lmin[j] then
lmin[j] := lmax[i] + a[i,j]; {thời gian đến j sớm nhất bằng thời gian
đến i muộn nhất (khởi hành ngay), cộng với thời gian đi trên cung (i,j)}
end;
```

end;

Procedure solution;

var k : integer;

f : text;

sum1,sum2 : integer;

q,d : ml;

dau,cuoi : integer;

begin

creat_max;

creat_min;

sum1:=0;

for k:=1 to n do

sum1:=sum1+(lmax[k]-lmin[k]); {Tổng thời gian chờ nhau ở các địa điểm}
{Thực hiện loang tìm các điểm không dừng}

fillchar(q,sizeof(q),0); {Khởi trị hàng đợi}

fillchar(d,sizeof(d),0); {Khởi trị mảng đánh dấu đỉnh nạp vào hàng đợi}

dau:=1; {biến đầu hàng đợi}

cuoi:=1; {biến cuối hàng đợi}

q[1]:=n; {Nạp đỉnh N vào hàng đợi}

d[n]:=1; {đánh dấu đỉnh N là đỉnh đầu tiên được nạp vào hàng đợi}

while dau<=cuoi do begin

for k:=1 to n do {Đề cử các đỉnh k}

```

if (d[k]=0) and (a[k, q[dau]]>0) {có cung (k, q[dau]) và k chưa được nạp}
and (lmax[k]+a[k, q[dau]]=lmax[q[dau]]) then begin
{dòng trên là điều kiện để k là điểm không dừng}
    inc (cuoi) ;
    q[cuoi] :=k; {thì nạp k vào hàng đợi}
    d[k] :=1; {đánh dấu đã nạp k}
end;
inc (dau) ; {chuẩn bị cho lấy phần tử đầu hàng đợi để loang tiếp}
end;
sum2:=n-cuoi; {sum2 là số điểm dừng, cuoi bằng số điểm không dừng}
assign (f, fo) ; rewrite (f) ;
writeln (f, lmax[n] , #32, sum1 , #32, sum2) ; {Ghi kết quả ra tệp output}
close (f) ;
end;
BEGIN
    input;
    solution;
END.

```

Bài 37. Build - Xây dựng đường

Vua Peaceful vừa khai hoang một vùng đất để lập ra đất nước Peace, lúc đầu chỉ có N thành phố (được đánh số từ 1 đến N) và không có con đường nào. Vua Peace chọn ra 4 thành phố đặc biệt để làm trung tâm kinh tế và 4 thành phố này phải được liên thông với nhau. Chi phí xây dựng các con đường không phải nhỏ vì thế nhà vua muốn sử dụng chi phí ít nhất để xây dựng các con đường sao cho 4 thành phố đặc biệt đó vẫn liên thông. Bạn được biết chi phí ước tính để xây dựng một số con đường và bạn hãy chọn một số con đường để xây dựng để theo đúng ý nhà vua biết rằng luôn tồn tại ít nhất một phương án xây dựng đường sao cho 4 thành phố đặc biệt liên thông.

Input: BUILD.INP

Dòng đầu tiên ghi số nguyên dương N là số lượng các thành phố.

Dòng thứ hai ghi 4 số nguyên là số hiệu của 4 thành phố đặc biệt.

Trong một số dòng tiếp theo, mỗi dòng ghi 3 số nguyên u, v và c với ý nghĩa muốn xây dựng một con đường hai chiều nối trực tiếp giữa 2 thành phố u và v thì chi phí là c.

Output: BUILD.OUT

Dòng đầu tiên tổng chi phí nhỏ nhất để xây dựng hệ thống đường.

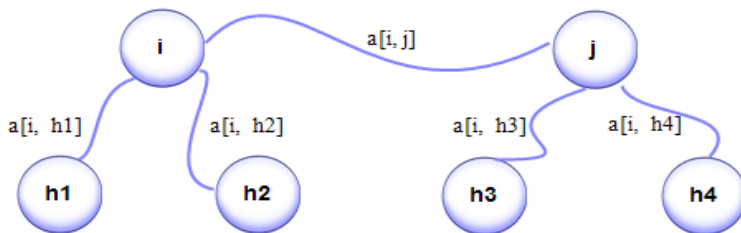
Trong một số dòng tiếp theo, mỗi dòng ghi 2 số nguyên u và v với ý nghĩa cần xây dựng con đường 2 chiều nối giữa 2 thành phố u và v .

Giới hạn: $1 \leq N \leq 100$; $1 \leq c \leq 5000$; Thời gian: 0.5 s/test

Bộ nhớ: 1 MB. Ví dụ:

BUILD.INP	BUILD.OUT
5	5
2 3 4 1	1 5
1 2 10	5 2
1 5 1	3 2
5 2 1	4 1
1 4 1	
4 3 3	
3 2 2	

Gợi ý. Có thể dùng thuật toán Floyd tìm đường đi ngắn nhất giữa hai đỉnh bất kỳ. Giả sử $a[i,j]$ là đường đi ngắn nhất giữa đỉnh i và đỉnh j . Gọi một hoán vị của 4 đỉnh đặc biệt là $h1, h2, h3, h4$. Với mọi cặp đỉnh (i,j) chọn ra cặp có tổng $a[i, j] + a[i, h1] + a[i, h2] + a[j, h3] + a[j, h4]$ nhỏ nhất.



Xét đủ 6 hoán vị của 4 đỉnh đặc biệt sẽ tìm ra các đỉnh trong hệ thống đường thỏa mãn đề bài.

Chương trình.

```

const
  fi          = 'BUILD.INP';
  fo          = 'BUILD.OUT';
  maxN        = 101;
  vc: longint = 600000;
  {Mảng H lưu 6 hoán vị của 1,2,3,4 }
  H: array[1..6,1..4] of integer= ((1,2,3,4),
                                     (1,3,2,4),
                                     (1,4,2,3),
  
```

```

(2, 3, 1, 4),
(2, 4, 1, 3),
(3, 4, 1, 2));

```

```

type
  mang1=array[1..maxN,1..maxN] of longint;
  mang2=array[1..maxN,1..maxN] of integer;
var
  f, g: text;
  N: integer; {Số đỉnh}
  t: array[1..4] of integer; {Lưu tên 4 đỉnh đặc biệt}
  a: mang1; {Mảng tính đường đi ngắn nhất giữa hai đỉnh bất kỳ}
  trace: mang2; {Lưu vết đường đi ngắn nhất}
  min: longint;
  sl: integer; {Số lượng cạnh trong hệ thống thỏa mãn đề bài}
  way: array[1..2,1..maxN] of integer; {Lưu các cạnh kết quả}

```

```

Procedure read_input;
var i,j,u,v,c: integer;
begin
  assign(f,fi); reset(f);
  readln(f,N);
  readln(f,t[1],t[2],t[3],t[4]); {Đọc 4 đỉnh đặc biệt lưu vào t}
  for i:=1 to n do
    for j:=1 to n do a[i,j]:= vc; {Khởi trị mảng a}
  while not seekeof(f) do begin
    readln(f,u,v,c); {Đọc độ dài cạnh (u,v) là c}
    a[u,v]:=c;
    a[v,u]:=c;
  end;
  close(f);
end;

```

```

Procedure Floy; {Thuật toán tìm đường đi ngắn nhất giữa mọi cặp đỉnh}
var k,i,j: integer;
begin
  fillchar(trace,sizeof(trace),0);
  for k:=1 to n do
    for i:=1 to n do
      for j:=1 to n do
        if a[i,j]>a[i,k]+a[k,j] then begin
          a[i,j]:=a[i,k]+a[k,j];
          trace[i,j]:=k; {Lưu vết đường đi ngắn nhất}
        end;
    for i:=1 to n do a[i,i]:=0; {Gán trị cho trường hợp đặc biệt}
  end;
end;

```

```

Procedure Tim(u,v: integer);
{Tìm hành trình ngắn nhất giữa hai đỉnh u và v, dựa vào vết đã lưu}

```

```

begin
    if trace[u,v]=0 then begin {Điểm dừng của thủ tục đệ quy}
        inc(sl); {Tăng số lượng cạnh trên hành trình}
        way[1,sl]:=u; {Lưu hai đầu cạnh}
        way[2,sl]:=v;
        exit;
    end;
    Tim(u,trace[u,v]); {Đệ quy tìm hành trình giữa u và trace[u,v]}
    Tim(trace[u,v],v); {Đệ quy tìm hành trình giữa trace[u,v] và v}
end;
Procedure GhiNhan(i,j,k: integer); {Ghi nhận các cạnh của toàn bộ hệ thống đường khi chọn hai đỉnh i, j cùng với hoán vị thứ k của 4 đỉnh đặc biệt}
begin
    sl:=0; {Khởi trị số cạnh của toàn bộ hệ thống đường}
    if i<>t[H[k,1]] then Tim(i,t[H[k,1]]); {Hành trình giữa i và t[H[k,1]]}
    if i<>t[H[k,2]] then Tim(i,t[H[k,2]]); {Hành trình giữa i và t[H[k,2]]}
    if i<>j then Tim(i,j); {Hành trình giữa i và j}
    if j<>t[H[k,3]] then Tim(j,t[H[k,3]]); {Hành trình giữa j và t[H[k,3]]}
    if j<>t[H[k,4]] then Tim(j,t[H[k,4]]); {Hành trình giữa j và t[H[k,4]]}
end;
Procedure Solve; {Thuật giải như gợi ý nêu trên}
var i,j,k, i0,j0,k0: integer;
    tong: longint;
begin
    min:= vc;
    for i:=1 to n do
        for j:=1 to n do
            if (a[i,j]<vc) then
                for k:=1 to 6 do
                    if (a[i,t[H[k,1]]]<vc) and (a[i,t[H[k,2]]]<vc)
                    and (a[j,t[H[k,3]]]<vc) and (a[j,t[H[k,4]]]<vc) then begin
                        tong:=a[i,j]+a[i,t[H[k,1]]]+a[i,t[H[k,2]]]+
                            a[j,t[H[k,3]]]+a[j,t[H[k,4]]];
                        if tong<min then begin
                            min:=Tong;
                            i0 := i; j0 := j; k0 := k;
                        end;
                    end;
                end;
            ghinhan(i0,j0,k0);
        end;
    end;
Procedure write_output;
var i: integer;
begin
    assign(g,fo); rewrite(g);
    writeln(g,Min);
    for i:=1 to sl do writeln(g,way[1,i],#32,way[2,i]);

```

```

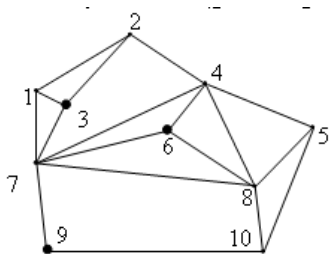
close(g);
end;
BEGIN
  read_input;
  Floyd;
  solve;
  write_output;
END.

```

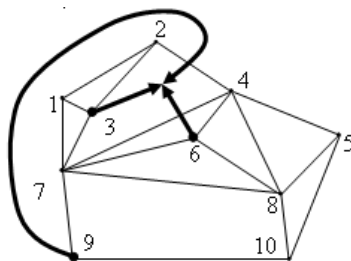
Bài 38. Walls (IOI 2000)

Ở một đất nước nọ, các đại lộ được xây dựng sao cho mỗi đại lộ nối đúng hai thành phố. Các đại lộ này không cắt nhau. Do đó đất nước này bị chia thành các miền mà muốn chuyển từ miền này sang miền khác cần phải tới một thành phố và xuyên qua một đại lộ. Với mọi cặp thành phố A và B cho trước, có nhiều nhất một đại lộ mà hai đầu là A và B, hơn nữa có thể đi từ A đến B theo các thành phố hoặc dọc theo đại lộ. Tập input đảm bảo điều này. Một Câu lạc bộ có các hội viên sống ở các thành phố. Mỗi thành phố có

nhiều nhất một hội viên. Các hội viên muốn gặp gỡ tại một địa điểm ngoại thành. Mọi hội viên sẽ đi du lịch bằng xe đạp. Họ không muốn vào bất kỳ thành phố



Hình 1



Hình 2

nào vì điều kiện giao thông và muốn qua ít đại lộ nhất nếu không gặp trở ngại. Để tới nơi hẹn, mỗi hội viên cần qua một số (có thể bằng 0) đại lộ. Họ muốn tìm một địa điểm tốt nhất sao cho tổng của những con số này là bé nhất (gọi là tổng cắt đại lộ). Các thành phố có số hiệu từ 1 đến N, N là số thị trấn. Trong hình 1, các nút đã đánh số là các thành phố và các đường nối các nút là các đại lộ. Giả sử rằng có 3 hội viên, họ sống ở các thành phố 3, 6 và 9. Do đó địa điểm gặp gỡ tốt nhất như hình 2. Tổng cắt đại lộ bằng 2: hội viên ở thành phố 9 đi ngang qua đại lộ nối thành phố 2 và thành phố 4, hội viên ở thành phố 6 đi ngang qua đại lộ nối thành phố 4 và thành phố 7. Bạn

hãy viết một chương trình nhận các thành phố, các miền, thành phố có hội viên câu lạc bộ, tính miền tốt nhất để tổng giao đại lộ là nhỏ nhất.

Dữ liệu vào. Tên tệp input là WALLS.IN. Dòng thứ nhất lần lượt chứa ba số nguyên M, N và L: M là số miền, $2 \leq M \leq 200$, N là số thành phố, $3 \leq N \leq 250$, L là số hội viên Câu lạc bộ, $1 \leq L \leq 30$, $L \leq N$. Dòng thứ hai chứa L số nguyên theo thứ tự tăng là số hiệu thành phố các hội viên sống. Sau đó, tệp input chứa 2M dòng, mà cứ hai dòng mô tả một miền. Với mỗi cặp, dòng thứ nhất cho biết số I là số thành phố trên biên của miền, dòng thứ hai chứa I số nguyên là các nhãn của I thành phố trên biên tính theo chiều kim đồng hồ. Miền cuối cùng là miền bao quanh tất cả các thành phố và các miền kia, thì nhãn các thành phố cho theo thứ tự ngược chiều kim đồng hồ. Thứ tự các miền có nhãn nguyên: miền thứ nhất có nhãn là 1, miền thứ hai có nhãn là 2,.. Chú ý tệp input chứa tất cả các miền tạo bởi các thành phố và đại lộ, kể cả miền ngoài cùng.

Kết quả ra. Tệp output có tên là WALLS.OUT. Dòng đầu chứa một số nguyên: là tổng giao đại lộ nhỏ nhất. Dòng thứ hai chứa một số nguyên là nhãn của miền gặp mặt tốt nhất. Có thể có nhiều nghiệm khác nhau, bạn chỉ cần nêu một nghiệm trong số đó.

Ví dụ input và output (phù hợp hình vẽ 1 và 2)

WALLS . IN	WALLS . OUT
10	2
10	
3	
3 6 9	
3	
1 2 3	
3	
1 3 7	
4	
2 4 7 3	
3	
4 6 7	
3	
4 8 6	
3	
6 8 7	

3	
4 5 8	
4	
7 8 10 9	
3	
5 10 8	
7	
7 9 10 5 4 2 1	

Gợi ý. Một cách giải chân phương là dùng đồ thị đối ngẫu với bản đồ các thành phố và đại lộ. Bản đồ đối ngẫu này đạt được khi xem mỗi miền là một đỉnh đồ thị, các cạnh nối chúng là các đại lộ (có thể là đa đồ thị), mỗi thành phố có thể thuộc nhiều miền khác nhau nên từ mỗi thành phố có thể chọn miền bắt đầu xuất phát là một trong các miền chứa thành phố này. Đi trên một cung của đồ thị đối ngẫu tương ứng với đi ngang qua một đại lộ. Vậy để tìm tổng giao đại lộ nhỏ nhất cần sử dụng thuật toán tìm đường đi ngắn nhất trên đồ thị đối ngẫu. Cách tiếp cận thô sơ nhất là thử cho mỗi miền (trong M miền) miền là nơi gặp gỡ của các hội viên và tìm kiếm đường đi tốt nhất cho mỗi thành viên (trong L thành viên) bằng cách thử với mỗi miền làm đỉnh xuất phát ($\leq N$). Áp dụng thuật toán Floyd tìm tất cả các cặp đường đi ngắn nhất trên đồ thị đối ngẫu sẽ cho thông tin cần thiết. Thuật toán này có độ phức tạp trung bình là $O(N^3 + M * L * N)$.

Ngoài ra cần lưu ý: Chiều quy định khi duyệt các thành phố trên biên của một vùng là: trên biên của miền ngoài cùng là chiều ngược kim đồng hồ, trên biên các miền còn lại là cùng chiều kim đồng hồ. Dùng chiều quy định này, khi xây dựng đồ thị đối ngẫu có thể xây dựng đúng các cạnh có độ dài bằng 1 và độ dài bằng 0. Cạnh (a, b) có độ dài bằng 1 khi 2 miền a và b cùng chứa đại lộ (i,j) nối thành phố i và thành phố j với nhau mà khi đi từ i đến j trên biên giới của a và của b thấy ngược chiều nhau. Ngược lại, cạnh (a,b) dài bằng 0 (vì từ miền kề với miền ngoài cùng vào miền ngoài cùng không đi cắt ngang qua đại lộ nào).

Chương trình.

```
{ $R-, Q- }
const  fi      = 'walls.in';
       fo      = 'walls.out';
```

```

maxM    = 200;
maxN    = 250;
type    arrN    = array[1..maxN] of byte;
        arrM    = array[1..maxM] of integer;
var      m,n,L  : integer;
        wall    : array[1..maxN] of ^arrN; {ma trận kề của đồ thị đối
ngẫu}
        belong  : array[1..30] of set of byte; {tập các đỉnh xuất phát
của từng hội viên}
        mark    : array[1..maxN] of byte; {mark[j]: hội viên thành phố j}
        dy      : array[1..maxM] of ^arrM; {nhân khoảng cách}
        ansRegion : integer; {miền gặp tốt nhất}
        ansDis   : longint; {tổng giao đại lộ ít nhất}
procedure init;
var f : text;
    i, j, k, a, b : integer;
    circle : array[1..maxN+1] of integer;
begin
    assign(f, fi); reset(f);
    readln(f, M, N, L); {số miền, số thành phố, số hội viên}
    fillchar(belong, sizeof(belong), 0); {Khởi trị mảng belong}
    fillchar(mark, sizeof(mark), 0); {Khởi trị mảng mark}
    for i := 1 to L do begin {đọc nơi ở của các hội viên}
        read(f, j);
        mark[j] := i; {Hội viên i ở thành phố j}
    end;
    for i := 1 to N do begin {Khởi trị mảng wall}
        new(wall[i]);
        fillchar(wall[i]^, sizeof(wall[i]^), 0);
    end;
    for i := 1 to M do begin {Khởi trị mảng new}
        new(dy[i]);
        fillchar(dy[i]^, sizeof(dy[i]^), 0);
    end;
    for i := 1 to M do begin {Đọc mô tả các miền}
        readln(f, j); {Số thành phố trên biên của miền}
        for k := 1 to j do
            read(f, circle[k]); {đọc thành phố thứ k trên biên giới miền i}
        circle[j+1] := circle[1]; {tạo mảng vòng các thành phố biên giới}
        for k:=1 to j do begin {lưu chiều từ thành phố a tới b trên biên miền i}
            a := circle[k];
            b := circle[k+1];
            if wall[a]^b = 0 then
                wall[a]^b := i {dành cho các miền trong}
            else wall[b]^a := i; {dành cho miền ngoài cùng}
        end;
    end;
end;

```

```

    if mark[a] > 0 then {có hội viên ở thành phố a}
    include(belong[ mark[a] ], i); {kết nạp thêm miền i vào tập các
                                đỉnh xuất phát của hội viên sống ở thành phố a}

    end;
end;
{Xây dựng ma trận trọng số của đồ thị đối ngẫu}
for i := 1 to N do
for j := 1 to N do
if (wall[i]^[j]>0)and (wall[j]^[i]>0) then begin
    dy[ wall[i]^[j] ]^[ wall[j]^[i] ] := 1;
    dy[ wall[j]^[i] ]^[ wall[i]^[j] ] := 1;
end;
close(f);
end;
procedure main;
var i, j, k, min : integer;
    thisDis      : longint;
begin
    ansDis := maxLongInt; {Khởi trị tổng giao đại lộ ít nhất}
    {Hai đỉnh không có cạnh nối bằng 0, gán lại trọng số bằng -1}
    for i := 1 to M do
    for j := 1 to M do
    if (i <> j) and (Dy[i]^[j] = 0) then
        Dy[i]^[j] := -1;
    {Floyd}
    for k := 1 to M do
    for i := 1 to M do
        if dy[i]^[k] <> -1 then
    for j := 1 to M do
        if dy[k]^[j] <> -1 then
            if (dy[i]^[j]=-1) or (dy[i]^[j]>dy[i]^[k]+dy[k]^[j]) then
                dy[i]^[j]:=dy[i]^[k]+dy[k]^[j];
    for i := 1 to M do begin {đề cử nơi họp là i}
        thisDis := 0; {Khởi trị giá trị tổng giao đại lộ ít nhất của phương án này}
        for j := 1 to L do begin{Xét đường đi ngắn nhất của từng hội viên j}
            min := maxint;
            for k := 1 to M do {chọn đỉnh xuất phát (miền k) cho hội viên j}
                if (k in belong[j]) and (dy[k]^[i]<min) then
                    min := Dy[k]^[i]; {sao cho từ đỉnh xuất phát tới nơi họp ngắn nhất}
            inc(thisDis, min); {cộng đường đi của hội viên j vào giá trị ph/án này}
        end;
        if thisDis<ansDis then begin {chọn phương án tốt nhất}
            ansDis := thisDis; {giá trị tổng giao đại lộ ít nhất}
            ansRegion := i;     {nơi họp tốt nhất}
        end;
    end;
end;

```

```

end;
end;
procedure out;
var g : text;
begin
    assign(g, fo); rewrite(g);
    writeln(g, ansDis);
    writeln(g, ansRegion);
    close(g);
end;
BEGIN
    init;
    main;
    out;
END.

```

Bài 39. Hội chợ

Khu gian hàng của một hội chợ có hình đa giác lồi $2N$ cạnh chia thành $2N-2$ gian hàng hình tam giác, đỉnh của chúng trùng với đỉnh của đa giác. Không có bất cứ một gian hàng nào liên hệ với đúng hai gian hàng khác (nghĩa là một gian hàng hoặc là liền kề với đúng một gian hàng khác hoặc đúng 3 gian hàng khác). Từ đó có N gian hàng (gọi là gian hàng ngoài), mỗi gian liền kề đúng với một gian hàng khác và có $N-2$ gian hàng (gọi là gian hàng trong), mỗi gian liền kề với đúng 3 gian hàng khác. Các gian hàng ngoài được đánh số từ 1 đến N . Các gian hàng trong được đánh số từ $N+1$ đến $2N-2$. Đi tham quan từ một gian hàng này đến một gian hàng khác thì phải mua vé. Trên mỗi cạnh chung giữa hai gian hàng có một cửa bán vé và chỉ bán vé cùng giá để từ gian hàng này sang gian hàng kia hoặc ngược lại. Giá vé tại các cửa khác nhau có thể khác nhau. Với mỗi cặp gian hàng ngoài biết được tổng giá vé phải trả trên đường đi từ gian hàng này đến gian hàng kia khi đi theo con đường qua ít cửa bán vé nhất. Sau một thời gian hoạt động, để khuyến mại Ban quản lý hội chợ quyết định giảm đồng loạt giá vé qua mỗi cửa, giá mỗi vé giảm bớt 1.

Yêu cầu: Hãy tính lại giá vé phải trả trên đường đi từ gian hàng ngoài này đến gian hàng ngoài khác khi đi theo con đường qua ít cửa bán vé nhất và thực hiện quyết định khuyến mại của Ban quản lý.

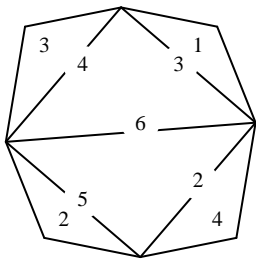
Dữ liệu vào từ file văn bản BL4.INP:

Dòng đầu tiên chứa số nguyên dương N ($4 \leq N \leq 100$)

Dòng thứ i trong N dòng tiếp theo chứa N số nguyên không âm $c_{i1}, c_{i2}, \dots, c_{iN}$, trong đó c_{ij} là tổng giá vé phải trả trên đường đi qua ít cửa bán vé nhất từ gian hàng ngoài i đến gian hàng ngoài j . Giả thiết giá vé nằm trong đoạn $[2..100]$, $c_{ij}=c_{ji}$ và $c_{ii}=0$.

Kết quả ghi ra file BL4.OUT gồm N dòng, trong đó dòng thứ i chứa N số nguyên không âm $C_{i1}, C_{i2}, \dots, C_{iN}$, trong đó C_{ij} là tổng giá vé phải trả trên đường đi qua ít cửa bán vé nhất từ gian hàng ngoài i đến gian hàng ngoài j khi thực hiện quyết định khuyến mại của Ban quản lý.

Ví dụ

BL4 . INP	BL4 . OUT	
<pre> 4 0 14 7 11 14 0 15 7 7 15 0 12 11 7 12 0 </pre>	<pre> 0 11 5 8 11 0 12 5 5 12 0 9 8 5 9 0 </pre>	

Gợi ý.

Bước 1. Thực hiện tính giá vé các cửa (lưu vào mảng B):

Tìm hai đỉnh u và v là phòng ngoài, cùng kề với phòng trong t , đó là phòng thoả mãn: với mọi phòng ngoài j ($j \neq u, j \neq v$) hiệu $\text{delta} = C_{ju} - C_{jv}$ không thay đổi (luôn bằng $x - y$, trong đó x là giá vé cửa nối phòng trong t và phòng ngoài u , y là giá vé cửa nối phòng trong t và phòng ngoài v). Chi phí từ u sang v là $C_{uv} = x + y$ vì đường đi qua ít cửa nhất từ u sang v chỉ qua 2 cửa này. Vậy có hệ phương trình :

$$\begin{cases} x + y = C(u, v) \\ x - y = \text{delta} \end{cases}$$

Giải hệ này tính được giá vé các cửa x và y nối phòng trong t với hai phòng ngoài u và v .

Xoá bỏ 2 phòng ngoài u và v , xác nhận t thành phòng ngoài.

Sửa lại chi phí đường đi qua ít cửa nhất từ các phòng ngoài còn lại tới phòng ngoài mới t .

Quá trình trên lặp $n-2$ lần đến khi chỉ còn hai phòng có một cửa duy nhất sang nhau. Rõ ràng giá vé cửa này cũng chính là cước phí đi từ phòng n tới phòng k ia. Vậy ta đã xây dựng được giá các cửa (lưu vào mảng B)

Bước 2: Dựa vào mảng B, xây dựng ma trận quan hệ của đồ thị sau: coi mỗi phòng là một đỉnh đồ thị, hai phòng có cửa thông nhau coi là hai đỉnh có cạnh dài bằng 1 nối chúng. Dựa vào mảng C, tìm đường đi qua ít cửa nhất giữa hai phòng ngoài bất kì.

Chi phí mới (giá khuyến mại) từ phòng ngoài u sang phòng ngoài v bằng chi phí cũ trừ đi số cửa trên đường đi ít cửa nhất từ u sang v .

```
const    fi    = 'bl4.inp';
         fo    = 'bl4.out';
         nmax  = 100;
         mmax  = 2*nmax-2;

type     mang  = array[1..mmax] of integer;
var      f      : text;
         a      : array[1..nmax,1..nmax] of integer; {mảng chi phí
trước khi khuyến mại từ phòng ngoài này tới phòng ngoài kia, mảng này không thay đổi}
         b      : array[1..mmax,1..mmax] of byte; {mảng giá các cửa đi
từ phòng nọ sang phòng kia}
         c      : array[1..mmax] of ^mang; {ban đầu gán là mảng a, giai
đoạn sau đóng vai trò ma trận quan hệ của đồ thị có đỉnh là các phòng và cạnh là cửa}
         ngoai, {đánh dấu phòng nào là phòng ngoài}
         dx : array[1..mmax] of boolean; {đánh dấu phòng đã xét}
         n,m, {n : số phòng ngoài, m : tổng số phòng trong và ngoài}
         u,v : integer; {2 phòng ngoài cùng kề một phòng trong}
         t, {phòng trong kề với u và v}
         x,y, {giá cửa từ phòng tr sang phòng u và sang phòng v}
         delta : integer; {hiệu x-y}

Procedure nhap;
var i,j : integer;
begin
    assign(f,fi); reset(f);
```

```

readln(f,n); {số phòng ngoài}
m:=2*n-2; {tổng số phòng trong và ngoài}
for i:=1 to n do
for j:=1 to n do begin
    read(f,a[i,j]); {cước phí qua ít cửa nhất từ phòng ngoài i đến phòng ngoài j}
    c[i]^j:=a[i,j]; {tạm thời gán c=a}
end;
close(f);
end;
Procedure khai_tao;
var i : integer;
begin
    fillchar(b,sizeof(b),0); {khởi tạo mảng b : mảng giá các cửa}
    fillchar(dx,sizeof(dx),false); {các đỉnh đều chưa xét}
    fillchar(ngoai,sizeof(ngoai),false); {tạm cho các phòng đều là
phòng ngoài}
    for i:=1 to n do ngoai[i]:=true; {xác nhận lại, chỉ có các phòng 1..n
là phòng ngoài}
end;
function thoa_man(u,v:integer):boolean; {kiểm tra 2 phòng ngoài u và v
có là hai phòng kề nhau không?}
var i : integer;
begin
    thoa_man := false;
    delta := maxint;
    for i:=1 to m do {xét mọi phòng ngoài i khác u và v,  $C[u,i]-C[v,i]=const?$  }
    if (ngoai[i]) and (i<>u) and (i<>v) then
    begin
        if delta=maxint then delta:=c[u]^i-c[v]^i
        else
            if delta<>c[u]^i-c[v]^i then exit;
    end;
    thoa_man:=true;
end;
Procedure tim_uv; {tìm 2 phòng ngoài u và v cùng kề một phòng trong}
var i,j : integer;
begin

```



```

for i:=1 to m do
  if ngoai[i] then
    for j:=1 to m do
      if (j<>i) and (ngoai[j]) then
        if thoa_man(i,j) then begin
          u:=i; v:=j;
          exit;
        end;
      end;
    end;
  end;
end;

Procedure tinh_giave; { giá vé 2 cửa nối phòng trong với 2 phòng ngoài u và v}
begin
  x := (c[u]^v+delta) div 2;
  y := (c[u]^v-delta) div 2;
end;

Procedure tao_phong_trong_ke_uv; {gán phòng trong tr chưa xét nào đó được
kề với hai phòng ngoài u và v}
var i : integer;
begin
  for i:=n+1 to m do
    if not dx[i] then begin
      t := i;
      exit;
    end;
  end;
end;

Procedure them_cua; {xác nhận giá vé 2 cửa nối phòng trong t với u và v}
begin
  b[u,t]:=x;b[t,u]:=x;
  b[v,t]:=y;b[t,v]:=y;
end;

Procedure sua_mang_c; {sau khi loại trừ 2 phòng ngoài u và v, phòng t được
thành phòng ngoài, do đó cần xây dựng cước phí từ các phòng ngoài tới phòng t}
var i : integer;
begin
  for i:=1 to m do
    if (i<>t) and (ngoai[i]) then begin
      c[t]^i := c[u]^i-x;
      c[i]^t := c[t]^i;
    end;
  end;
end;

```

end;

Procedure danh dau; *{đánh dấu đã xét các phòng u, v, t, đánh dấu loại trừ 2 phòng ngoài u và v, đánh dấu phòng tr được thành phòng ngoài}*

begin

dx[t] := true; dx[u] := true; dx[v] := true;

ngoai[u] := false; ngoai[v] := false;

ngoai[t] := true;

end;

Procedure xaydung_cua; *{xây dựng giá 2 cửa vào u và v, một số thao tác khác liên quan}*

begin

tim_uv;

tinh_giave;

tao_phong_trong_ke_uv;

them_cua;

danh dau;

sua_mang_c;

end;

Procedure xaydung_cua_cuoicung;

var i, j : integer;

begin

{tìm 2 phòng ngoài cuối cùng là u}

for i:=1 to m do if ngoai[i] then begin u:=i; break; end;

for j:=i+1 to m do if ngoai[j] then begin v:=j; break; end;

{xác nhận giá cửa cuối cùng là cửa thông 2 phòng u và v}

b[u,v] := c[u]^v; b[v,u] := b[u,v];

end;

Procedure tao_mang_gia_cua; *{tạo mảng giá các cửa}*

var i : integer;

begin

for i:=1 to n-2 do xaydung_cua;

xaydung_cua_cuoicung;

end;

Procedure chuanbi;

begin

khởi_tạo;

tao_mang_gia_cua;

end;

Procedure tao_mang_cua; *{tạo mảng c : ma trận quan hệ của đồ thị mà mỗi phòng là một đỉnh, có cửa thông hai phòng coi là có cạnh dài bằng 1: $c[i]^j=1$ là có cửa, $c[i]^j=0$ là không có cửa}*

```

var i,j,k : integer;
begin
  for i:=1 to m do
    for j:=1 to m do
      if b[i,j]>0 then c[i]^j:=1
      else
        if i<>j then c[i]^j:=maxint div 2
        else c[i]^j:=0;
        {tìm đường đi ngắn nhất (qua ít cửa nhất) bằng thuật toán Floyd}
        for k:=1 to m do
          for i:=1 to m do
            for j:=1 to m do
              if c[i]^j>c[i]^k+c[k]^j then
                c[i]^j:=c[i]^k+c[k]^j;
            end;
          end;
        end;
      end;
    end;
  end;
Procedure ghikq;
var i,j : integer;
begin
  assign(f,fo); rewrite(f);
  for i:=1 to n do begin
    for j:=1 to n do
      write(f,a[i,j]-c[i]^j:6); {cước phí khuyến mại}
      writeln(f);
    end;
  end;
  close(f);
end;
var i : integer;
BEGIN
  for i:=1 to mmax do begin
    new(c[i]);
    fillchar(c[i]^,sizeof(c[i]^),0);
  end;
  nhap;
  chuanbi;
  tao_mang_cua;

```

```

ghikq;
for i:=1 to mmax do dispose(c[i]);
END.

```

Bài 40. Mê cung (Thi HSG Quốc Gia 2004 - Bảng A)

Một hãng tư nhân ở thành phố X vừa khánh thành một mê cung rất hấp dẫn khách du lịch. Mê cung có N phòng được đánh số từ 1 đến N và có M cầu thang, mỗi cầu thang nối trực tiếp hai phòng với nhau. Việc đi lại giữa các phòng chỉ có thể thực hiện thông qua các cầu thang. Phòng 1 là lối ra-vào duy nhất của mê cung. Các cầu thang không cắt nhau. Giữa hai phòng bất kỳ có không quá 1 cầu thang nối chúng. Không có cầu thang nối một phòng với chính nó. Thời gian đi theo mỗi một trong hai chiều cầu thang là cho trước.

Nhân dịp khai trương mê cung, chủ hãng treo giải thưởng lớn cho du khách nào có cách thâm nhập vào mê cung theo một tuyến đường nào đó trong mê cung và sau đó tìm cách ra ngoài với thời gian ít nhất. Tuyến đường phải bắt đầu từ phòng số 1, đi qua ít nhất một phòng (không kể phòng số 1) rồi quay lại phòng số 1, và thoả mãn yêu cầu: Mỗi cầu thang, mỗi phòng (không kể phòng số 1) đi không quá 1 lần.

Yêu cầu: Cho biết sơ đồ của mê cung, hãy tìm cách thâm nhập với thời gian ít nhất thoả mãn các điều kiện nêu trên.

Dữ liệu: Vào từ file văn bản MAZE.INP:

Dòng thứ nhất chứa số nguyên N ($3 \leq N \leq 5000$) và M ($3 \leq M \leq 10000$)

M dòng tiếp theo chứa thông tin về các cầu thang: Mỗi dòng chứa 4 số nguyên a, b, c, d được ghi cách nhau bởi dấu cách, cho biết phòng a và phòng b được nối với nhau bởi cầu thang, và theo cầu thang này: thời gian đi từ a đến b là c còn thời gian đi từ b đến a là d ($1 \leq c, d \leq 10000$)

Kết quả: Ghi ra file văn bản MAZE.OUT thời gian ít nhất của cách thâm nhập tìm được.

Ví dụ:

MAZE.INP	MAZE.OUT
3 3	3

```

1 2 1 10
1 3 5 1
2 3 1 9

```

Gợi ý. Sử dụng thuật toán tìm đường đi ngắn nhất từ phòng xuất phát là phòng 1 tới các phòng kết thúc là các phòng có thang xuống phòng 1. Lưu ý cần cộng thêm thời gian đi trên thang cuối cùng này về phòng 1. So sánh các phương án, sẽ được phương án tối ưu.

Chương trình.

```

{B-,R-,Q-}
const
    fi      = 'maze.inp';
    fo      = 'maze.out';
    maxN    = 5001;
    maxM    = 20001;
    vc      = maxlongint;

type
    mangI   = array[1..maxM] of integer;
var
    f, g    : text;
    N,M     : integer;
    Tro     : array[1..maxN] of integer;
    ke1,ke2 : ^mangI;
    min     : longint;
    Q       : array[1..maxN] of integer;
    top     : integer;
    kc      : array[1..maxN] of longint;

Procedure init;
begin top:=0;end;
Procedure put(u: integer);
begin
    inc(top); q[top]:=u;
end;
function get: integer;
var u,i: integer;
begin
    u:=1;
    for i:=2 to top do
        if kc[q[i]]>kc[q[u]] then u:=i;
    get:=q[u];
    q[u]:=q[top]; dec(top);
end;
function empty: boolean;
begin
    empty:=(top=0);

```

```

end;
Procedure doc_dulieu;
var i,u,v,l1,l2: integer;
    tam: array[1..maxN] of integer;
begin
    assign(f,fi); reset(f);
    readln(f,N,M); {Số phòng, số cầu thang}
    {tạo danh sách kề theo kiểu Forward Star}
    for i:=1 to N do tro[i]:=0;
    for i:=1 to M do begin
        readln(f,u,v,l1,l2);
        inc(tro[u]);inc(tro[v]);
    end;
    close(f);
    v:=0;
    for i:=1 to N do begin
        u:=tro[i];
        tro[i]:=v+1;
        v:=v+u;
    end;
    tro[N+1]:=v+1;
    {đọc file input lần thứ hai để tạo hai mảng ke1 và ke2 với ý nghĩa: ke1 là tên đỉnh kề,
ke2 là thời gian trên cầu thang tới đỉnh kề}
    move(tro,tam,sizeof(tro));
    reset(f);
    readln(f,N,M);
    for i:=1 to M do begin
        readln(f,u,v,l1,l2);
        ke1^[tam[u]]:=v; ke2^[tam[u]]:=l1; inc(tam[u]);
        ke1^[tam[v]]:=u; ke2^[tam[v]]:=l2; inc(tam[v]);
    end;
end;
Procedure dijkstra(xp,kt,lkt: integer; var dmin: longint);
{Thực hiện Dijkstra từ đỉnh xp tới đỉnh kt, cộng thêm độ dài từ kt về phòng 1 là lkt, trả về
độ dài ngắn nhất của hành trình này là dmin}
var i,u,v,lv: integer;
begin
    dmin:= vc;
    init;
    for i:=1 to N do kc[i]:= vc;
    kc[xp]:=0;
    for i:=tro[xp] to tro[xp+1]-1 do {i là vị trí các phòng kề với phòng 1}
    {nếu chưa là phòng có thể về phòng 1, và độ dài hành trình chưa tối ưu thì }
    if (ke1^[i]<>kt) and (ke2^[i]+lkt<dmin) then begin
        put(ke1^[i]); {nạp đỉnh kề này vào "hàng đợi"}
        kc[ke1^[i]]:=-ke2^[i]; {xác nhận nhãn tự do của đỉnh kề với phòng 1}
    end;
end;

```

```

end;
while not empty do begin
    u:=get; {lấy một đỉnh có nhãn tự do nhỏ nhất}
    kc[u]:=-kc[u]; {xác nhận đã cố định nhãn}
    if u=kt then begin {kết thúc khi gặp phòng kt có thang về phòng 1}
        dmin:=kc[kt]+lkt; {xác nhận độ dài ngắn nhất của hành trình}
        exit;
    end;
    for i:=tro[u] to tro[u+1]-1 do begin {i là vị trí các phòng kề với u}
        v:=ke1[i]; {v: phòng kề với u}
        lv:=ke2[i]; {lv: thời gian đi từ u tới v}
        {nếu v đã có nhãn tự do và nhãn này chưa tối ưu}
        if (kc[v]<0) and (-kc[v]>kc[u]+lv) then
            kc[v]:=-(kc[u]+lv); {sửa nhãn cho v}
        {nếu v chưa có nhãn và hành trình tới v nhờ qua u tốt hơn các hành trình đã có thì }
        if (kc[v]=vc) and (kc[u]+lv+lkt<min) then begin
            put(v); { nạp v vào “hàng đợi” gồm các đỉnh có nhãn tự do}
            kc[v]:=-(kc[u]+lv); {xác nhận nhãn tự do của v}
        end;
    end;
end;
end;
end;
Procedure solve;
var i,j,u: integer;
    dmin: longint;
begin
    min:= vc;
    for i:=tro[1] to tro[2]-1 do begin {tìm các vị trí của đỉnh kề với 1}
        u:=ke1[i]; {u: phòng đầu tiên kề với phòng 1}
        j:=tro[u]; {j là vị trí của u trong danh sách kề}
        while ke1[j]<>1 do inc(j) {duyet mọi phòng kt có đường về phòng 1}
        dijkstra(1,ke1[i],ke2[j],dmin);
        if dmin<min then min:=dmin;
    end;
end;
end;
Procedure in_ketqua;
begin
    assign(g,fo); rewrite(g);
    writeln(g,Min);
    close(g);
end;
Procedure capphat;
begin
    new(ke1); new(ke2);
end;

```

```

BEGIN
    capphat;
    doc_dulieu;
    solve;
    in_ketqua
END.

```

Bài 41. Roads - Chia tay

N thành phố được đánh số từ 1 đến N nối với nhau bằng các đường một chiều. Mỗi con đường có hai thông số: độ dài con đường và lệ phí cần thiết phải trả (tính bằng số đồng tiền xu). Bob và cô bạn gái Alice cùng sống với nhau trong thành phố 1. Sau khi nhận thấy Alice đã lừa dối mình trong các công việc làm ăn, Bob quyết định chia tay cô ta và muốn về thành phố N. Bob muốn ra đi càng nhanh càng tốt nhưng còn rất ít tiền. Chúng ta hãy giúp đỡ Bob tìm con đường ngắn nhất từ thành phố 1 đến thành phố N mà Bob có đủ khả năng trả lệ phí cho những con đường sẽ đi qua.

Dữ liệu vào từ file văn bản ROADS.IN chứa số nguyên K ($0 \leq K \leq 10000$) là số tối đa đồng xu mà Bob còn có thể chi cho lệ phí đi đường. Dòng thứ hai chứa số nguyên N ($2 \leq N \leq 100$) là số thành phố. Dòng thứ ba chứa số nguyên R ($1 \leq R \leq 10000$) là số các con đường. Mỗi dòng trong R dòng tiếp theo xác định một con đường bởi các số nguyên S, D, L và T cách nhau dấu cách.

S là thành phố bắt đầu của con đường, $1 \leq S \leq N$

D là thành phố tới của con đường $1 \leq D \leq N$

L là độ dài con đường, $1 \leq L \leq 100$

T là lệ phí đi qua con đường tính bằng xu, $0 \leq T \leq 100$

Kết quả ghi ra file văn bản ROADS.OUT chỉ gồm 1 dòng duy nhất ghi độ dài đường đi ngắn nhất từ thành phố 1 đến thành phố N mà tổng lệ phí không vượt quá K xu. Nếu không tồn tại đường đi nào thì ghi -1.

Ví dụ 1		Ví dụ 2		Ví dụ 3	
ROADS.IN	ROADS.OUT	ROADS.IN	ROADS.OUT	ROADS.IN	ROADS.OUT
5	11	0	-1	5	5
6		4		6	
7		4		8	

1 2 2 3		1 4 5 2		1 2 8 3	
2 4 3 3		1 2 1 0		1 3 1 1	
3 4 2 4		2 3 1 1		2 4 1 1	
1 3 4 1		3 4 1 0		2 6 1 1	
4 6 2 1				3 4 3 4	
3 5 2 0				3 5 1 1	
5 4 3 2				4 6 4 1	
				5 4 1 1	

Gợi ý. Trước hết sử dụng thuật toán Dijkstra tìm đường đi ngắn nhất (về khoảng cách) ngược từ đỉnh N về các đỉnh khác, lưu vào mảng *mindist*, sau đó một lần nữa sử dụng thuật toán Dijkstra tìm đường đi ngắn nhất (về chi phí tiền) ngược từ đỉnh N về các đỉnh khác lưu vào mảng *mincost*. Hai mảng *mindist* và *mincost* được dùng làm cận cho quá trình duyệt tìm hành trình từ đỉnh 1 đến đỉnh N sao cho đường đi ngắn nhất (về độ dài) và đủ tiền thực hiện hành trình:

Xây dựng thủ tục Duyệt(i, d, t) với ý nghĩa: đã duyệt tới đỉnh i, đã đi được quãng đường là d và số tiền đã chi là t, hãy tìm hành trình tiếp theo đỉnh i. Ngay đầu thủ tục Duyệt(i, d, t) chúng ta lưu ý đặt hai cận :

- Nếu $(d + \text{mindist}[i] \geq \text{đường đi của phương án tốt nhất})$ thì không cần duyệt tiếp phương án hiện thời nữa.
- Nếu $(t + \text{mincost}[i] > k \text{ (số tiền có của Bob)})$ thì không cần duyệt tiếp phương án hiện thời nữa.

Ngoài ra cũng nên tổ chức các đỉnh theo danh sách kề để quá trình tìm các đỉnh kề với một đỉnh nào đó được nhanh chóng và thuận tiện.

Chương trình.

```

const    fi                = 'roads.in';
         fo                = 'roads.out';
         maxn              = 100;
         vc                = 20000;
         maxtime           = 180;
type     pt                = ^tnode;
         tnode             = record
                                   v                : byte;
                                   l, t            : byte;
                                   next            : pt;
         end;

```

```

        m1                = array[1..maxn] of word;
        m2                = array[1..maxn, 1..maxn] of word;
var
    list                  : array[1..maxn] of pt;
    dd                    : array[1..maxn] of boolean;
    cost, dist            : m2;
    mincost, mindist      : m1;
    k                     : word;
    n                     : byte;
    best                  : word;
    f                     : text;
Procedure init;
var i, r, u, v, l, t : word;
    tmp                : pt;
begin
    assign(f, fi); reset(f);
    readln(f, k);      {số tiền của Bob}
    readln(f, n);      {số thành phố}
    readln(f, r);      {số con đường}
    for u:=1 to n do {khởi trị nhãn : giá tiền min, khoảng cách min}
    for v:=1 to n do begin
        cost[u, v]:= vc; dist[u, v]:= vc;
    end;
    { danh sách liên kết một chiều list[i] là danh sách các đỉnh kề đỉnh i}
    for i:=1 to n do list[i]:=nil; {khởi trị từng dslk list[i]}
    for i:=1 to r do begin {đọc từng con đường}
        readln(f, u, v, l, t); {điểm đầu, điểm cuối, độ dài và cước phí con đường}
        {thêm một nút nối vào đầu dslk : list[u]}
        new(tmp); tmp^.v:=v; tmp^.l:=l; tmp^.t:=t;
        tmp^.next:=list[u]; list[u]:=tmp;
        new(tmp); tmp^.v:=u; tmp^.l:=l; tmp^.t:=t;
        tmp^.next:=list[v]; list[v]:=tmp;
        cost[u,v] := t; cost[v,u] := t;
        dist[u,v] := l; dist[u,vu] := l;
    end;
    close(f);
end;
Procedure dijkstra(var a : m2; var nhan : m1);
{Thuật toán Dijkstra tìm đường đi tối ưu xuất phát từ thành phố N}
var tudo                : array[1..maxn] of boolean;
    min                  : word;
    i, j, last          : byte;
begin
    fillchar(tudo, sizeof(tudo), true); {mảng đánh dấu đỉnh tự do}
    for i:=1 to n do nhan[i] := vc; {khởi trị mảng nhãn}

```

```

nhan[n] := 0; {khởi trị nhãn đỉnh N}
tudo[n] := false; {khởi trị đỉnh N đã được cố định nhãn}
last := n; {last : đỉnh được chọn để sửa nhãn cho các đỉnh tự do}
for i:=1 to n-1 do begin {sửa nhãn N-1 lần các đỉnh đều được cố định
nhãn}
    {sửa nhãn cho các đỉnh tự do, kể với đỉnh last}
    for j:=1 to n do
        if tudo[j] and (a[j,last]+nhan[last]<nhan[j]) then
            nhan[j] := nhan[last] + a[j, last];
    {tìm đỉnh tự do có nhãn nhỏ nhất}
    min := vc + 1;
    for j:=1 to n do
        if tudo[j] and (nhan[j]<min) then begin
            min := nhan[j];
            last := j;
        end;
    tudo[last]:=false; {cố định đỉnh last}
end;
end;
Procedure try(last : byte; l, t : word);
{Duyệt đường đi tiếp khi Bob đã tới thành phố last, đã đi đoạn đường là l và đã chi t xu}
var tmp : pt;
begin
    if (l+mindist[last]>=best) {điều kiện cần về tối ưu độ dài đường đi}
    or (t+mincost[last]>k) then {điều kiện cần về tối ưu tiền chi phí}
        exit;
    if last = n then begin {tới đích : điểm dừng của đệ qui}
        best:=l; exit;
    end;
    tmp := list[last]; {tmp : danh sách các thành phố kề với thành phố đích}
    while tmp<>nil do begin {duyet đề cử các thành phố kề thành phố last}
        if not dd[tmp^.v] then begin {thành phố tmp^.v chưa qua}
            dd[tmp^.v]:=true; {đánh dấu thành phố tmp^.v đã qua}
            try(tmp^.v, l+tmp^.l, t+tmp^.t); {duyet tiếp từ tmp^.v}
            dd[tmp^.v]:=false; {quay lui}
        end;
        tmp:=tmp^.next; {tạo cho đề cử thành phố tiếp theo}
    end;
end;
end;
Procedure process;
begin
    {xây dựng mảng cost và dist để làm cận phục vụ duyệt}
    dijkstra(cost, mincost);
    dijkstra(dist, mindist);
    best := vc; {khởi trị độ dài đường đi của phương án tối ưu}

```

```

fillchar(dd, sizeof(dd), false); {khởi trị mảng đánh dấu đỉnh chưa duyệt}
try(1, 0, 0); {Duyệt từ thành phố 1, đường đã đi là 0, tiền đã tiêu là 0}
end;
Procedure done;
begin
    assign(f, fo); rewrite(f);
    if best = vc then writeln(f, -1)
        else writeln(f, best);
    close(f);
end;
BEGIN
    init;
    process;
    done;
END.

```

Bài 42. Cưỡi lạc đà thăm K thành phố

Một hoang đảo trên sa mạc có N thành phố, $2 \leq N \leq 1000000$. Hai thành phố bất kì có thể được nối trực tiếp với nhau bởi đúng một con đường, mỗi thành phố có không quá 5 con đường nối trực tiếp tới các thành phố khác.

An nhận được danh sách K thành phố cần đến thăm bằng lạc đà, $2 \leq K \leq 8$. Các thành phố này có số hiệu là a_1, \dots, a_k và lạc đà không thể đi quá L con đường mỗi ngày, $1 \leq L \leq 10$. An bắt đầu hành trình từ thành phố a_1 sau đó thăm và nghỉ đêm tại từng thành phố trong K-1 thành phố cần thăm trong đúng K ngày. An có thể thăm theo một thứ tự tùy ý, nhưng phải trở về thành phố a_1 khi kết thúc hành trình. An không thăm bất kì thành phố nào hơn một lần. Đáng tiếc là An không có bản đồ của hoang đảo, do đó không biết được những thành phố nào là được nối hay không được nối với nhau bởi các con đường trực tiếp. May thay, An có điện thoại di động gọi sự giúp đỡ của nhà cung cấp dịch vụ SMS địa phương. Mỗi lần gửi tin nhắn cho SMS, An nhận được bản tin phản hồi của SMS thông báo cho biết tất cả các thành phố có đường nối trực tiếp tới một thành phố nào đó mà An hỏi. Bởi vì tài khoản máy di động có hạn nên An không thể gửi quá 5000 tin nhắn tới SMS.

Viết chương trình giúp An tìm ra hành trình thoả mãn những điều kiện đã nêu.

Thư viện. Để giải bài toán, bạn được sử dụng một thư viện tên là NHANTIN gồm 4 chương trình con sau:

- **KHOITRI** : Chỉ được gọi đúng một lần và sử dụng trước mọi thủ tục khác trong thư viện. Thủ tục này trả về số N và L qua các tham biến.

procedure khoitri(var n, L : longint);

- **DANHSACH** : Thủ tục trả về số thành phố (K) mà An cần thăm và danh sách các thành phố này (mảng một chiều LISTC) tương ứng có số hiệu a_1, \dots, a_k .

procedure danhsach(var k:longint;var listc:array of longint);

- **PHANHOI** : Thủ tục đóng vai trò như một bản tin phản hồi của SMS. Nó cung cấp số lượng (howmany) và danh sách các thành phố (mảng một chiều neighbours) có đường nối trực tiếp với thành phố đang quan tâm (city). Có thể gọi thủ tục này không quá 5000 lần.

procedure phanhoi(city : longint; var howmany : longint; var neighbours : array of longint);

- **GHIKQ** : Thủ tục được gọi khi bạn muốn nhờ nó ghi kết quả vào file output và kết thúc thực hiện chương trình của bạn. Tham số của nó là số lượng (tourlength) và số hiệu các thành phố An đã thăm theo đúng thứ tự (mảng một chiều tour), thành phố a_1 được kể hai lần. Thành phố đầu tiên và cuối cùng của hành trình phải là a_1 . (thủ tục này sẽ tự ghi kết quả hành trình do chương trình của bạn tìm được vào file output là C:\BT\nhantin.o01)

procedure ghikq(tourlength : longint; var tour : array of longint);

Bạn được cung cấp thư viện nêu trên từ file C:\BT\NHANTIN.PAS và một file input làm ví dụ là C:\BT\nhantin.i01

```
unit nhantin;
interface
const fi      = 'nhantin.i01';
      fo      = 'nhantin.o01';
const MAXN    = 1000002; {1000000;}
      MAXQ    = 2000002;
type kieu0    = array[0..7] of longint;
      kieu11   = array[0..4] of longint;
      kieu5    = array[0..MAXN] of kieu11;
```

```

        kieu8 = array[0..99] of longint;
var f : text;

procedure khoitri( var n, m: longint );
procedure danhhsach( var k: longint; var dstham : kieu0 );
procedure phanhoi(a : longint; var x : longint; var ke :
kieu11);
procedure ghikq( sothanhpho : longint; var manght : kieu8 );

implementation
procedure khoitri( var n, m : longint );
begin
    assign(f,fi);
    reset(f);
    readln(f,n,m);
    close(f);
end;
procedure danhhsach( var k: longint; var dstham : kieu0 );
var i : byte;
begin
    assign(f,fi);
    reset(f);
    readln(f);
    readln(f,k);
    for i:=0 to k-1 do read(f,dstham[i]);
    close(f);
end;
procedure phanhoi(a: longint; var x : longint; var ke :
kieu11 );
var i,j,k : longint;
begin
    assign(f,fi);
    reset(f);
    readln(f);
    readln(f);
    readln(f);
    for i:=1 to a-1 do readln(f);
    while not seekeof(f) do
    begin
        read(f, x);
        for j:=0 to x-1 do
            read(f, ke[j]);
        break;
    end;
    close(f);
end;
end;

```

```

procedure ghikq( sothanhpho : longint; var manght : kieu8 );
var g : text;
    i : longint;
begin
    assign(g,fo);
    rewrite(g);
    writeln(g,sothanhpho);
    for i:=0 to sothanhpho - 1 do
        if i=0 then write(g,manght[0])
        else write(g,' ',manght[i]);
    close(g);
    halt;
end;
END.

```

NHANTIN.I01

```

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

```

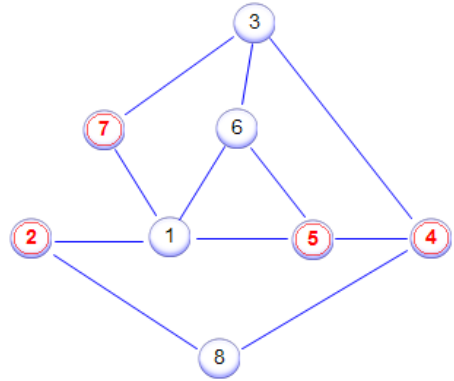
Gợi ý. Trước hết, dùng Free Pascal từ tệp NHANTIN.PAS được cung cấp, bạn cần tạo tệp mã lệnh của máy có phần mở rộng của tên tệp là *PPW). Các hằng và các kiểu được định nghĩa trong NHANTIN.PPW cần được bảo lưu dùng trong chương trình của bạn.

Phần thứ nhất trong lời giải là tìm xem những cặp thành phố nào được có thể đi tới nhau bằng một đường đi gồm không quá L con đường trực tiếp. Thực hiện công việc này nếu sử dụng tiếp cận BFS một cách chân phương thì có thể phải dùng hơn $8 \cdot 4^{10}$ câu hỏi (hàng triệu câu hỏi).

Tuy nhiên, chúng ta dùng cách chỉ tiếp cận đến thành phố ở giữa mỗi đường đi (meet-in-the-middle) để giảm số lượng câu hỏi xuống. Trước hết chúng ta thực hiện tìm kiếm theo chiều rộng (BFS) bắt đầu tại từng thành phố để tìm tất cả các thành phố (trong K thành phố cần thăm) có thể tới bằng đường đi không quá $L/2$ con đường. Nếu chúng ta có thể tới thành phố

C từ thành phố A bằng đường đi không quá $L/2$ con đường, và có thể tới thành phố C từ thành phố B cũng không quá $L/2$ con đường thì từ thành phố A có thể tới B không quá L con đường. Số câu hỏi yêu cầu trong cách tiếp cận này xấp xỉ bằng căn bậc hai của số câu hỏi trong cách tiếp cận ban đầu. Thực ra, chúng ta không cần sử dụng nhiều hơn $8*(1+5+5*4+5*4^2+5*4^3)=3408$ câu hỏi.

Phần thứ hai của lời giải là tìm chu trình Haminton trên đồ thị không quá 8 đỉnh bằng cách tạo ra mọi hoán vị của 8 đỉnh rồi kiểm tra xem hoán vị nào chấp nhận được. Chúng ta có thể thực hiện 8! khả năng trong thời gian cho phép.



Với input NHANTIN.i01, kết quả tệp output NHANTIN.o01 là:
10
7 1 2 8 4 3 6 5 1 7

Chương trình.

```
uses nhantin;
const MAXN = 1000002;
      MAXQ = 2000002;

type
    kieu1 = array[0..MAXN] of boolean;
    kieu2 = array[0..7, 0..MAXN] of shortint;
    kieu3 = array[0..7, 0..MAXN] of longint;
    kieu4 = array[0..MAXN] of longint;
    kieu6 = array[0..7] of boolean;
    kieu7 = array[0..8] of longint;
    kieu9 = array[0..7, 0..7] of longint;
    kieu10 = array[0..MAXQ] of longint;

var n, m, k, i, j : longint;
    list : kieu0; {danh sách các thành phố cần thăm}
    visited : kieu1; {đánh dấu đã thăm}
    kc : kieu2; {độ dài giữa hai thành phố tính bằng số con đường}
    truoc : kieu3; {theo dõi vết các đường đi}
    slke : kieu4; {số lượng các thành phố kề với một thành phố nào đó}
    dske : kieu5; {quản lý danh sách các thành phố kề với một thành phố}
    dd : kieu6; {đánh dấu một hoán vị của các thành phố cần thăm}
    hoanvi : kieu7; {một hoán vị của các thành phố cần thăm}
```



```

ht      : kieu8; {hành trình của An}
giua    : kieu9; {giua[i,j]=a thể hiện có thể đi từ list[i] tới list[j] bằng cách
qua thành phố là a, nếu giua[i,j]=0 thể hiện không có đường đi nào từ list[i] tới list[j] phù
hợp sức lực đà}
q       : kieu10; {hàng đợi dùng trong BFS}
ke      : kieu11; {danh sách kề với thành phố nào đó}
sl      : longint; {số lượng thành phố đi qua trong hành trình của An}
cuoi,dau: longint; {biến theo dõi đầu và cuối hàng đợi}
procedure push(a : longint); {thủ tục nạp a vào hàng đợi}
begin
  q[cuoi]:=a;
  if cuoi<MAXQ then inc(cuoi) else cuoi:=0;
end;
function pop : longint; {thủ tục lấy phần tử ở đầu hàng đợi}
begin
  pop:=q[dau];
  if dau<MAXQ then inc(dau) else dau:=0;
end;
procedure bfs(j : longint); {Tìm đường đi từ thành phố thứ j trong danh sách
các thành phố cần thăm qua các thành phố khác, chú ý từ thành phố này đến thành phố kia
dài không quá một nửa sức đi của lạc đà trong một ngày. Quá trình tìm đường đi này cũng
là quá trình định ra khoảng cách giữa hai thành phố liên tiếp trên đường đi}
var a, b, i : longint;
begin
  kc[j,list[j]]:=0; {Khởi trị nhãn khoảng cách từ thành phố thứ j tới chính nó}
  cuoi:=0; {cuối hàng đợi}
  dau:=0; {đầu hàng đợi}
  push(list[j]); {nạp thành phố list[j] cần thăm vào hàng đợi}
  while cuoi<> dau do begin
    a := pop; {lấy phần tử đầu hàng đợi là thành phố a}
    if kc[j,a]>=(m+1) shr 1 then break; {bỏ qua a nếu khoảng cách từ
      thành phố list[j] tới a vượt quá 1/2 khả năng đi của lạc đà trong một ngày}
    if slke[a]=-1 then {chưa có tin phản hồi về các con đường ra khỏi a}
      phanhoi(a, slke[a], dske[a]); {thì gọi SMS giúp đỡ}
    for i:=0 to slke[a]-1 do begin {duyet các thành phố kề a theo SMS}
      b := dske[a, i]; {thành phố b là thành phố thứ i trong dske kề với a}
      if kc[j,b]<=kc[j,a]+1 then
        continue; {đi từ list[j] tới b không xa hơn a quá 1 con đường thì bỏ qua b}
      kc[j,b]:=kc[j,a]+1; {vì từ j qua a tới b thêm 1 con đường nối từ a tới b}
      truoc[j, b] := a; {khi xét đường đi từ list[j] thì trước b là a}
      if not visited[b] then push(b); {loang tiếp từ b nếu b chưa qua}
    end;
  end;
end;

```

```

procedure creat_g; {Tạo đồ thị chỉ gồm các đỉnh là các thành phố trong danh sách cần thăm}
var i, j, a : longint;
begin
  for i := 0 to k-2 do {chọn thành phố thứ i trong danh sách}
  for j:=i+1 to k-1 do {chọn thành phố j trong danh sách (j<>i)}
  for a:=1 to n do begin { a giữa list[i] và list[j] }
    if kc[i,a]+kc[j,a]>m then continue; {quá sức lực đà, bỏ qua}
    if (a<>list[i]) and (a<>list[j]) and visited[a] then
      continue; {a là thành phố trung gian đã thăm, thì bỏ qua}
    giữa[j,i]:=a; {ghi nhận thành phố “giữa” con đường từ list[i] đến list[j] là a}
    giữa[i,j]:=a; {ghi nhận thành phố “giữa” con đường từ list[j] đến list[i] là a}
    break;
  end;
end;

function haminton(a, i : longint) : boolean;
{xây dựng từ phần tử thứ i của hoán vị là vị trí thứ a trong danh sách (list) cần thăm}
var b : longint;
begin
  hoanvi[i]:=a; { phần tử thứ i của hoán vị ứng với phần tử ở vị trí a của list}
  if i = k then begin {xây dựng xong một hoán vị có k phần tử}
    haminton:=(a=0); {phần tử cuối hoán vị là phần tử đầu hoán vị, có chu trình}
    exit;
  end;
  for b:=0 to k-1 do begin {đề cử phần tử tiếp theo của hoán vị là list[b]}
    if giữa[a,b]=0 then
      continue; {bỏ qua vì lực đà không đi được}
    if dd[b] then continue; {bỏ qua vì list[b] đã vào hoán vị rồi}
    dd[b] := true; {xác nhận list[b] vào hoán vị}
    {chú ý đoạn in đậm là một cách duyệt hiệu suất: bước sau duyệt thành công thì kết luận bước hiện tại duyệt mới thành công, do đó nó tự động cắt bỏ đi các nhánh có các bước sau không thành công}
    if haminton(b,i+1) then begin { nếu xây dựng tiếp vị trí thứ i+1 của hoán vị từ phần tử thứ b của list thành công thì}
      haminton:=true; { thì xây dựng vị trí thứ i của hoán vị từ phần tử thứ a của list cũng thành công }
      exit;
    end;
    dd[b] := false; {quay lui}
  end;
  haminton:=false; {không thoát được ở trên thì bước hiện tại không thành công}
end;

procedure timduong1(i, a : longint);
{đệ qui, phục hồi lại đường đi từ hoanvi[i+1], biết thành phố a nằm “giữa” hoanvi[i] và hoanvi[i+1]}

```

```

begin
    if a=-1 then exit;
    if ht[sl-1] <> a then begin
        ht[sl] := a;
        inc(sl);
    end;
    timduong1(i,truoc[i,a]);
end;
procedure timduong2(i, a : longint);
{đệ qui, phục hồi lại đường đi từ hoanvi[i], biết thành phố a nằm “giữa” hoanvi[i] và
hoanvi[i+1]}
begin
    if a = -1 then exit;
    timduong2( i, truoc[i, a] );
    if ht[sl-1] <> a then begin
        ht[sl] := a;
        inc(sl);
    end;
end;
BEGIN
    khoitri(n,m); {số thành phố, M sức đi của lạc đà}
    danhsach(k,list); {số thành phố cần thăm và danh sách các thành phố này}
    {Khởi trị một số mảng}
    for i:=0 to 7 do for j:=0 to MAXN do kc[i, j] := 63;
    for i:=0 to 7 do for j:=0 to MAXN do truoc[i, j] := -1;
    for i:=0 to MAXN do slke[i] := -1;
    for i:=0 to k-1 do visited[list[i]] := true;
    for i:=0 to k-1 do bfs(i); {tìm các đường đi từ list[i]}
    creat_g; {tạo ma trận giữa cho biết có đường đi vừa sức lạc đà giữa các thành
phố trong danh sách các thành phố cần thăm hay không}
    haminton(0,0); {tìm một hoán vị thể hiện thứ tự thăm bắt đầu từ list[0]}
    { phục hồi lại vết của hành trình}
    ht[sl] := list[0];
    inc(sl);
    for i:=0 to k-1 do begin
        timduong2( hoanvi[i], giữa[hoanvi[i], hoanvi[i+1]] );
        timduong1( hoanvi[i+1], giữa[hoanvi[i], hoanvi[i+1]] );
    end;
    ghikq(sl,ht); {nhờ thư viện, ghi nhận kết quả}
END.

```

Bài 43. Robot cứu hỏa (Thi HSG Quốc gia 2007)

Mạng lưới giao thông gồm n nút, giữa một số nút có đường nối 2 chiều. Đoạn đường chứa 2 thông tin $t[i,j]$ là thời gian đi đoạn đường, $c[i,j]$ là năng lượng để đi hết đoạn đường. Robot chỉ đi qua được một đoạn đường

(i,j) khi năng lượng còn lại trong bình không ít hơn $c[i,j]$. Trên một số nút có trạm tiếp năng lượng, khi robot đến nút này thì được nạp đầy năng lượng. Thời gian nạp năng lượng coi như không đáng kể. Robot xuất phát tại nút 1 với bình năng lượng w đến cứu hỏa tại nút n .

Yêu cầu: Xác định giá trị w nhỏ nhất để robot đi được trên một đường đi từ nút 1 đến nút n trong thời gian ít nhất

Input: Cho trong file ROBOT.INP

Dòng đầu tiên ghi số nguyên dương N ($2 \leq N \leq 500$)

Dòng thứ hai ghi N số 1 hoặc 0 thể hiện ở trạm thứ i có hoặc không có trạm tiếp năng lượng

Dòng thứ ba ghi M là số tuyến đường ($M \leq 30000$)

Dòng thứ k trong M dòng tiếp ghi thông tin về một tuyến đường gồm 4 số $i, j, t[i,j], c[i,j]$ ($t[i,j], c[i,j] \leq 104$)

Output: Ghi ra file văn bản ROBOT.OUT số w tìm được

Ví dụ:

ROBOT.INP	ROBOT.OUT
4	3
0 1 1 0	
5	
1 2 5 4	
1 3 4 3	
1 4 9 4	
2 4 4 1	
3 4 5 2	

Gợi ý.

Sử dụng thuật toán tìm đường đi ngắn nhất Dijkstra kết hợp tổ chức dữ liệu Heap. Đồng thời có thể sử dụng phương pháp tìm kiếm nhị phân để tìm lượng chi phí w nhỏ nhất bảo đảm robot đi được tới đích.

Chương trình.

```
uses crt;
const fi = 'robot.in';
      fo = 'robot.out';
      maxn = 502;
      maxm = 300002;
```

```

    vc      = 2000000000;
type  arr1 = array[0..maxm] of integer;
      arr2 = array[0..maxn] of longint;
      arr3 = array[0..2*maxm] of integer;
var    n,m   : integer;
      dau, cuoi, t, c : ^arr1;
      x      : arr2;
      tro    : arr2;
      ke     : arr3;
      qn,ds,sl : integer;
      q,vt,kc,nl,prev,tt : arr2;
      f,g     : text;
Procedure doc_dulieu;
var i : integer;
begin
    new(dau); new(cuoi); new(t); new(c);
    assign(f,fi); reset(f);
    read(f,n);
    for i:=1 to n do read(f,x[i]);
    read(f,m);
    for i:=1 to m do
        read(f,dau^[i], cuoi^[i], t^[i] , c^[i]);
    close(f);
end;
Procedure danhsachke;
var u,v : longint; i : integer;
begin
    for i:=1 to m do begin
        inc(tro[dau^[i]]); inc(tro[cuoi^[i]]);
    end;
    v := 0;
    for i:=1 to n do begin
        u := tro[i];
        tro[i] := v+1;
        v := v + u;
    end;
    tro[n+1] := v+1;
    for i:=1 to m do begin
        ke[tro[dau^[i]]] := i;
        ke[tro[cuoi^[i]]] := i;
        inc(tro[dau^[i]]);
        inc(tro[cuoi^[i]]);
    end;
    for i:=n downto 2 do tro[i] := tro[i-1];
    tro[1] := 1;
end;

```

```

Procedure up (k : integer);
var v : integer;
begin
  v := q[k];
  while (kc[q[k div 2]] > kc[v]) do begin
    q[k] := q[k div 2];
    vt[q[k]] := k;
    k := k div 2;
  end;
  q[k] := v;
  vt[q[k]] := k;
end;

Procedure down(k : integer);
var v, l : integer;
begin
  v := q[k];
  while (2*k < qn) do begin
    l := 2*k;
    if ((l < qn) and (kc[q[l+1]] < kc[q[l]])) then inc(l);
    if (kc[q[l]] > kc[v]) then break;
    q[k] := q[l];
    vt[q[k]] := k;
    k := l;
  end;
  q[k] := v;
  vt[q[k]] := k;
end;

Procedure init;
begin
  qn := 0;
  vt[0] := 0;
  kc[0] := -1;
end;

Procedure put(u : integer);
begin
  inc(qn); q[qn] := u; vt[u] := qn; up(qn);
end;

function get : integer;
var kq : integer;
begin
  kq := q[1];
  q[1] := q[qn]; dec(qn);
  vt[q[1]] := 1;
  down(1);
  get := kq;
end;

```

```

Procedure update(u : integer);
var k : integer;
begin
    k := vt[u];
    up(k);
end;
function empty: boolean;
begin
    empty := (qn=0);
end;
Procedure Dijkstra;
var u, v : integer; i : longint;
begin
    init;
    kc[1] := 0;
    prev[1] := -(n+1);
    put(1);
    while not empty do begin
        u := get;
        prev[u] := -prev[u];
        inc(sl); tt[sl] := u;
        for i:= tro[u] to tro[u+1]-1 do begin
            if dau^[ke[i]]<>u then v := dau^[ke[i]]
            else v := cuoi^[ke[i]];
            if ((prev[v]<0) and(kc[v]>kc[u]+t^[ke[i]])) then begin
                kc[v] := kc[u]+t^[ke[i]];
                prev[v] := -u;
                update(v);
            end;
            if (prev[v]=0) then begin
                kc[v] := kc[u]+t^[ke[i]];
                prev[v] := -u;
                put(v);
            end;
        end;
    end;
end;
function diduoc(w : longint): boolean;
var i,k,u,v,tg,cp,cl : longint;
begin
    for i:=1 to n do nl[i] := -vc;
    nl[1] := w;
    for k:=1 to sl do begin
        u := tt[k];
        if nl[u]>0 then
            for i:=tro[u] to tro[u+1]-1 do begin

```

```

    if dau^[ke[i]]<>u then v := dau^[ke[i]]
    else v := cuoi^[ke[i]];
    tg := t^[ke[i]];
    cp := c^[ke[i]];
    if (kc[v]=kc[u]+tg) and (nl[u]>=cp) then begin
        if x[v]=1 then cl := w else cl := nl[u]-cp;
        if cl>nl[v] then nl[v] := cl;
    end;
end;
end;
diduoc := (nl[n]>=0);
end;
Procedure solve;
var first, last, lim : longint;

begin
    first :=0; last := 1;
    while not diduoc(last) do begin
        first := last; last := last*2;
    end;
    while (last-first>1) do begin
        lim := (first + last) div 2;
        if diduoc(lim) then last := lim
        else first := lim;
    end;
    ds := last;
end;

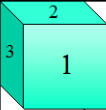
Procedure ghi_ketqua;
begin
    assign(g,fo); rewrite(g);
    write(g,ds);
    close(g);
end;
BEGIN
    doc_dulieu;
    danhsachke;
    dijkstra;
    solve;
    ghi_ketqua;
END.

```

Bài 44. Lăn xúc xắc

Cho một bàn cờ lưới ô vuông đơn vị kích thước $m \times n$, trên mỗi ô ghi một số tự nhiên ≤ 7 . Có một con xúc xắc (hình lập phương cạnh 1 đơn vị)

nằm tại một ô (x, y) mang số 7. Các mặt con xúc xắc được ghi các số nguyên dương từ 1 đến 6: mặt trên mang số 1, mặt bên hướng về mép trên của lưới mang số 2, mặt bên hướng về mép trái của lưới mang số 3, tổng hai số ghi trên hai mặt đối diện bất kỳ luôn bằng 7. (Xem hình vẽ)

1	2	3	4
3		4	1
6	6	6	6
3	4	1	2

Cho phép lăn con xúc xắc sang một trong 4 ô kề cạnh. Sau mỗi phép lăn như vậy, mặt trên của xúc xắc sẽ trở thành mặt bên tương ứng với hướng di chuyển và mặt bên theo hướng di chuyển sẽ trở thành mặt đáy. Một phép lăn được gọi là hợp lệ nếu nó luôn đảm bảo số ghi ở ô xúc xắc đang đứng hoặc bằng 7, hoặc bằng với số ghi ở mặt đáy của xúc xắc. Như ví dụ trên, ta có thể lăn lên trên, sang phải hay sang trái nhưng không thể lăn xuống dưới.

Yêu cầu

Hãy chỉ ra một số hữu hạn các phép lăn hợp lệ để lăn con xúc xắc ra một ô biên của lưới, nếu có nhiều phương án thực hiện thì chỉ ra phương án mà tổng các số ghi ở mặt trên của xúc xắc sau mỗi bước di chuyển là cực tiểu.

Dữ liệu vào từ file văn bản `ROLL.INP`

Dòng 1: Chứa 4 số m, n, x, y ($1 < x < m \leq 300$; $1 < y < n \leq 300$). M dòng tiếp theo, dòng thứ i chứa n số mà số thứ j là số ghi tại ô (i, j) của lưới.

Kết quả ghi ra file văn bản `ROLL.OUT`

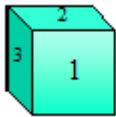
Gồm một dòng chứa dãy liên tiếp các ký tự, ký tự thứ k có thể là L, R, U hoặc D tương ứng với phép lăn tại bước thứ k là lăn sang trái, lăn sang phải, lăn lên trên hay lăn xuống dưới.

Ví dụ

ROLL. INP	ROLL. OUT
9 6 3 3	URDDLULL
0 0 0 0 0 0	
0 0 2 4 0 0	
1 4 7 6 6 6	
0 0 2 3 0 0	
0 0 0 1 0 0	
0 0 0 4 0 0	
0 0 0 6 0 0	
0 0 0 3 0 0	
0 0 0 1 0 0	

Gợi ý.

Trước hết xây dựng bảng mã của mặt trái của quân xúc xắc khi biết mã mặt đáy và mã mặt bên ở phía trên lưu vào mảng leftcode[1..6,1..6] có 24 giá trị khác 0 như dưới đây:



0	4	2	5	3	0
3	0	6	1	0	4
5	1	0	0	6	2
2	6	0	0	1	5
4	0	1	6	0	3
0	3	5	2	4	0

Ví dụ leftcode[6; 2]=3 vì mặt dưới là 6, mặt bên phía trên là 2 thì mặt bên phía trái là 3. Leftcode[4,5]=1 vì mặt dưới là 4 và mặt bên phía trên là 5 thì mặt bên phía trái là 1. Một quân xúc xắc hoàn toàn xác định khi biết số ở mặt dưới là B, số ở mặt bên phía trên là U (khi đó số ở mặt bên phía trái là L=Leftcode[B,U]). Vì vậy mỗi bộ ba (B, U, L) xác định một trạng thái duy nhất của quân xúc xắc.

Khi đặt một quân xúc xắc lên bàn cờ, một trạng thái của trò chơi được tổ hợp bởi trạng thái quân xúc xắc và toạ độ ô quân xúc xắc đang đứng. Mỗi trạng thái trò chơi cũng được mã hoá thành một số. Khi quân xúc xắc đứng

ở một ô biên, tạo thành một trạng thái đích của bàn cờ, vì vậy có quá nhiều trạng thái đích.

Dùng thuật toán Dijkstra, tổ chức trên Heap, tìm đường đi có trọng số nhỏ nhất từ trạng thái đích ngược về trạng thái ban đầu của trò chơi. Sau đó lại dùng thuật toán Dijkstra để tìm lại đường đi thuận từ trạng thái ban đầu đi về một trạng thái đích tìm được trong quá trình tìm ngược đã tiến hành trước.

Chương trình.

```
const    fi  = 'ROLL.IN1';
         fo  = 'ROLL.OUT';
         max = 300;
         maxlong = 10000000;
         leftcode: array[1..6, 1..6] of Byte =
         (
           (0, 4, 2, 5, 3, 0),
           (3, 0, 6, 1, 0, 4),
           (5, 1, 0, 0, 6, 2),
           (2, 6, 0, 0, 1, 5),
           (4, 0, 1, 6, 0, 3),
           (0, 3, 5, 2, 4, 0)
         );
type     t_heap = record
         x, y, code: Integer;
       end;
         t_xucxac = record
         B, U, L: Integer;
       end;
var      map: array[0..max + 1, 0..max + 1] of byte;
         trace: array[1..max, 1..max, 1..24] of byte;
         d: array[1..max, 1..max, 1..24] of longint;
         pos: array[1..max, 1..max, 1..24] of Integer;
         heap: array[1..max * max * 24] of t_heap;
         index: array[1..6, 1..6, 1..6] of integer;
         state: array[1..24] of t_xucxac;
         m, n, xstart, ystart, nh: word;
```

procedure read_input;

```
var      f: text;
         i, j: word;
begin
  fillchar(map, sizeof(map), 0);
  assign(f, fi); reset(f);
```

```

    readln(f, m, n, xstart, ystart); // số dòng, số cột, toạ độ xuất phát
    for i := 1 to m do begin
        for j := 1 to n do read(f, map[i, j]); // đọc bàn cờ
        readln(f);
    end;
    close(f);
end;
procedure getindex; // mã hoá các trạng thái của xúc xắc
var bottom, up, count: integer;
begin
    count := 0;
    for bottom := 1 to 6 do
        for up := 1 to 6 do
            if leftcode[bottom, up] <> 0 then begin
                inc(count);
                with state[count] do begin
                    B:=bottom; U:=up; L:=leftcode[bottom, up];
                    index[B, U, L] := count; // mã số của một trạng thái
                end;
            end;
        end;
    end;
end;
function less(p1, p2: t_heap): boolean;
// so sánh của hai trạng thái trò chơi: nhãn của p1 nhỏ hơn nhãn của p2
begin
    less := d[p1.x, p1.y, p1.code] < d[p2.x, p2.y, p2.code];
end;
function less_equal(p1, p2: t_heap): boolean;
begin
    less_equal := d[p1.x, p1.y, p1.code] <= d[p2.x, p2.y, p2.code];
end;
procedure setheap(id: integer; value: t_heap);
begin
    heap[id] := value;
    with value do
        pos[x, y, code] := id;
end;
function getd(p: t_heap): longint; // trả về nhãn của trạng thái p
begin
    with p do
        getd := d[x, y, code];
end;
procedure push(p: t_heap); // nạp trạng thái p vào heap
var c, r: word;
begin
    c := pos[p.x, p.y, p.code]; // số hiệu trong heap của trạng thái p

```

```

if c = 0 then begin
    inc(nh); // tăng số nút của heap
    c := nh;
end;
// tìm ngược về gốc đến vị trí thích hợp
r := c div 2;
while r <> 0 do begin
    if less_equal(heap[r], p) then break; // đã đến chỗ thích hợp
    setheap(c, heap[r]); // đặt lại heap[r] vào vị trí c
    c := r;
    r := c div 2;
end;
setheap(c, p); // đặt p vào vị trí c của heap
end;
procedure pop(var p: t_heap); // Lấy phần tử 1 khỏi heap, vun lại heap
var r, c: word;
    tmp: t_heap;
begin
    p := heap[1];
    tmp := heap[nh];
    dec(nh);
    r := 1;
    while r * 2 <= nh do begin
        c := r * 2;
        if (c < nh) and (less(heap[c + 1], heap[c])) then
            inc(c); // chuyển sang nút con có giá nhỏ hơn
        if less_equal(tmp, heap[c]) then break;
        setheap(r, heap[c]); // đặt nút heap[c] vào vị trí r
        r := c;
    end;
    setheap(r, tmp); // đặt tmp vào vị trí r
end;
procedure Dijkstra1;
var tmp: t_heap; x, y, c: integer;
procedure check(tmp: t_heap);
begin
    with tmp do
        if (map[x,y]=7) or (map[x,y]=state[code].B) then begin
            d[x, y, code]:=7-state[code].B; // nhãn là số mặt trên
            push(tmp); // đi được tới tmp nên nạp tmp vào heap
        end;
    end;
begin
    // Khởi trị mảng nhãn là vô cùng
    for x := 1 to m do

```

```

for y := 1 to n do
for c := 1 to 24 do
    d[x, y, c] := maxlong;
nh := 0; // khởi trị số nút của heap
// nạp các trạng thái có quân xúc xắc đứng ở các ô biên của bàn cờ vào heap
for c := 1 to 24 do begin
    for x := 1 to m do begin
        tmp.x := x; tmp.y := 1; tmp.code := c;
        check(tmp);
    end;;
    for x := 1 to m do begin
        tmp.x := x; tmp.y := n; tmp.code := c;
        check(tmp);
    end;
    for y := 2 to n-1 do begin
        tmp.x := 1; tmp.y := y; tmp.code := c;
        check(tmp);
    end;
    for y := 2 to n-1 do begin
        tmp.x := m; tmp.y := y; tmp.code := c;
        check(tmp);
    end;
end;
fillchar(trace, sizeof(trace), 0); // Khởi trị mảng vết
end;
function changestate(currentstate: integer; direction:
integer): integer;
var xucxac, res: t_xucxac;
begin
    xucxac := state[currentstate]; // trạng thái hiện thời
    res := xucxac; // khởi trị trạng thái mới bằng trạng thái hiện thời
    case direction of // theo hướng di chuyển, xác nhận trạng thái mới
        1: {UP}
            begin
                res.B := xucxac.U; res.U := 7 - xucxac.B;
            end;
        2: {DOWN}
            begin
                res.B := 7 - xucxac.U; res.U := xucxac.B;
            end;
        3: {LEFT}
            begin
                res.L := 7 - xucxac.B; res.B := xucxac.L;
            end;
        4: {RIGHT}
            begin

```

```

        res.L := xucxac.B; res.B := 7 - xucxac.L;
    end;
end;
changestate := index[res.B, res.U, res.L]; //mã số trạng thái mới
end;
function canmove(var p, q: t_heap; direction: integer):
boolean; // trả về khả năng có thể di chuyển trạng thái p theo hướng direction
const dx: array[1..4] of integer = (-1, 1, 0, 0);
      dy: array[1..4] of integer = (0, 0, -1, 1);
var   newstate: integer;
begin
    newstate := changestate(p.code, direction); //trạng thái mới
    q.x := p.x + dx[direction]; // tọa độ ô xúc xắc đứng ở trạng thái mới
    q.y := p.y + dy[direction];
    q.code := newstate; // mã số thể hiện kiểu đứng của xúc xắc
    with q do
        canMove:=(map[x,y]=7) or (map[x,y]=state[code].B);
        // có thể chuyển khi ô tới có số 7 hoặc số như mặt đáy của xúc xắc
    end;
procedure update(p: t_heap); // cập nhật p vào heap khi xúc xắc di chuyển
var   q: t_heap;
      dp: longint;
      di: integer;
begin
    dp := getd(p); //nhận trạng thái p
    for di := 1 to 4 do //xét 4 hướng di chuyển
    if canmove(p, q, di) and // có thể di chuyển được
        (getd(q) > dp + 7 - state[q.code].B) then begin //tối ưu hơn
            with q do begin //di chuyển và cập nhật lại mảng nhãn và mảng lưu vết
                d[x, y, code] := dp + 7 - state[q.code].B;
                trace[x, y, code] := di;
            end;
            push(q); // nạp trạng thái mới vào heap
        end;
    end;
procedure Dijkstra2;
// từ các trạng thái xúc xắc ở ô biên đã nạp, nạp thêm cho đến khi gặp ô xuất phát
var p: t_heap;
    endcode: Integer;
begin
    endcode := index[6, 2, 3];
    repeat
        pop(p); //lấy đỉnh heap
        if (p.code=endcode) and (p.x=xstart) and (p.y=ystart)
        then break; // gặp ô xuất phát thì dừng

```

```

    update(p); // nạp thêm ô mới do từ trạng thái p di chuyển tiếp
until nh = 0; // số nút trong heap bằng 0
end;
procedure write_out; // Lần theo vết, tìm kết quả
const back: array[1..4] of char = 'DURL';
      backD: array[1..4] of integer = (2, 1, 4, 3);
var g: text;
    d: word;
    p, q: t_heap;
begin
    assign(g, fo); rewrite(g);
    with p do begin // từ trạng thái xuất phát
        x := xstart; y := ystart; code := index[6, 2, 3];
    end;
    repeat // tìm vết hành trình
        with p do d := trace[x, y, code]; // vết của hướng đi
        write(g, back[d]); // ghi hướng đi vào tệp output
        canmove(p, q, backD[d]);
        p := q;
    until trace[p.x, p.y, p.code] = 0; // không đi được nữa: hướng=0
    close(g);
end;
BEGIN
    read_input;
    getindex;
    dijkstral;
    dijkstra2;
    write_out
END.

```

Bài 45. Supersch - Đội tuyển có thành tích cao nhất

Có N trường học lập đội tuyển thi chạy việt dã. Mỗi đội gồm k người. Mỗi lần thi đấu mỗi trường cử một người ra chạy. Như vậy có k lần thi tất cả. Mỗi lần thi như vậy, người chạy nhanh nhất được nhận huy chương. Trường nào nhận được nhiều huy chương nhất thì thắng cuộc. Trường phổ thông "Siêu đẳng" không khá về thể thao lắm nên tìm cách thi có lợi nhất. Do Ban tổ chức sẽ lần lượt gọi danh sách theo thứ tự đăng ký mà các trường gửi lên, nên trường "Siêu đẳng" đã thu thập thành tích của tất cả các vận động viên của các đội bạn (tính theo thời gian chạy 1000m). Đợi cho tất cả các trường khác đăng ký xong họ tìm cách lấy danh sách đó để sắp xếp số

thứ tự cho đội mình (dĩ nhiên, họ nắm rõ trình độ đội nhà). Cho biết nếu nhiều vận động viên về đích cùng lúc thì đều giành được huy chương.

Bạn hãy giúp trường "Siêu đẳng" sắp xếp danh sách đội tuyển sao cho số huy chương đạt được là nhiều nhất.

Dữ liệu vào từ file văn bản SUPERSCH.INP:

Dòng đầu tiên chứa hai số n, k ($N \leq 1000, k \leq 100$)

Tiếp theo là N dòng, dòng thứ i ghi thành tích của k vận động viên trường i . Trường "Siêu đẳng" là trường N , trừ trường "Siêu đẳng" tất cả các trường khác danh sách k vận động viên chính là danh sách mà Ban tổ chức sẽ căn cứ vào đó để gọi vận động viên ra thi đấu.

Kết quả ghi ra file SUPERSCH.OUT:

Dòng đầu ghi số huy chương mà trường "Siêu đẳng" nhận được

Dòng thứ hai chứa số thứ tự của vận động viên (số thứ tự căn cứ theo thứ tự của vận động viên trong dữ liệu nhập vào)

SUPERSCH.INP	SUPERSCH.OUT
3 4	2
2 3 5 6	3 4 1 2
6 7 4 5	
9 8 1 2	

Gợi ý.

Dùng mảng một chiều $\text{Super}(k)$ lưu thành tích k vận động viên trường "Siêu đẳng" và mảng một chiều $\text{Max}(k)$ lưu điểm cao nhất của $N-1$ đội "không siêu đẳng" trong mỗi trận đấu: $\text{Max}[j]$ là điểm cao nhất của $N-1$ đội này trong trận thứ j ($1 \leq j \leq k$). Xây dựng đồ thị hai phía $G=(V, E)$, trong đó tập đỉnh $V=A \cup B$ mà các đỉnh bên A là số hiệu vận động viên trường "Siêu đẳng", các đỉnh bên B là số hiệu trận đấu. Có cạnh nối $A[i]$ với $B[j]$ nếu $\text{super}[i] \geq \text{max}[j]$ (điều kiện để vận động viên i của trường siêu đẳng sẽ giành được huy chương nếu cho đấu trận j). Sau đó sử dụng thuật toán tìm "bộ ghép cực đại" trên G .

Chương trình.

```
const fi = 'Supersch.INP';
```

```

fo      = 'Supersch.OUT';
maxK    = 100;
type    arr1 = array[1..maxK] of longint;
        arr2 = array[1..maxK] of integer;
var      f, g      : text;
        N, K      : integer;
        max, Super : arr1;
        A, B      : arr2;
        Q        : arr2;
        first, last : integer;
        Trace     : arr2;
        lu, lv    : integer;
        sol       : integer;
Procedure init; {Khởi trị hàng đợi}
begin
    first:=1;    last:=1;
end;
Procedure put(u: integer); {Nạp u vào cuối hàng đợi}
begin
    q[last]:=u; inc(last);
end;
function get: integer; {Lấy phần tử ở đầu hàng đợi ra khỏi hàng đợi}
begin
    get:=q[first]; inc(first);
end;
function empty: boolean; {Hàm kiểm tra hàng đợi rỗng}
begin
    empty:=(first=last);
end;
Procedure doc_dulieu;
var i, j : integer;
    u    : longint;
begin
    assign(f, fi); reset(f);
    readln(f, N, K); {Đọc số trường, số môn thi đầu}
    {Xây dựng mảng điểm vận động viên giỏi nhất môn j của N-1 trường kia}
    for j:=1 to K do max[j]:= -MaxLongint;
    for i:=1 to N-1 do begin
        for j:=1 to K do begin
            read(f, u);
            if u>max[j] then max[j]:=u;
        end;
        readln(f);
    end;
    for i:=1 to K do
        read(f, Super[i]); {Đọc điểm k vận động viên trường Super}

```

```

    close(f);
end;
Procedure khoitri; {Khởi trị mảng A và B trong thuật toán tìm bộ ghép cực đại}
var i: integer;
begin
    for i:=1 to K do A[i]:=0;
    for i:=1 to K do B[i]:=0;
end;
Procedure tim_duongmo(var ok: boolean); {Tìm đường mở}
var i,u,v,w: integer;
begin
    ok:=true;
    init; {Khởi trị hàng đợi}
    for i:=1 to K do Trace[i]:=0; {Khởi trị mảng vết}
    for i:=1 to K do {Nạp đồng thời các điểm đầu của các đường mở}
    if A[i]=0 then begin
        trace[i]:=-1;
        put(i);
    end;
    {Tìm đồng thời các đường mở cho tới khi có một đường mở}
    while Not empty do begin
        u:=Get; {Lấy một đỉnh bên A (bên trường Super)}
        for v:=1 to K do {Chọn môn v cho vận động viên u}
        if Super[u]>=max[v] then begin {u chắc thắng}
            if B[v]=0 then begin {v là đỉnh nhạt bên B thì tìm xong một đường mở}
                lu:=u; {lưu lại đỉnh cuối cùng bên A của đường mở}
                lv:=v; {lưu lại đỉnh cuối cùng bên B của đường mở}
                exit; {Thoát, ok bằng true: báo tìm được đường mở}
            end;
            w:=B[v]; {w là đỉnh bên A đã ghép với v bên B}
            if Trace[w]=0 then begin {nếu w chưa thuộc đường mở thì}
                put(w); {nạp w vào hàng đợi để kéo dài thêm đường mở}
                trace[w]:=u; {xác nhận đỉnh u là đỉnh trước (bên A) của w trên đường mở}
            end;
        end;
    end;
    ok:=false; {không tìm được đường mở}
end;
Procedure tang_canh; {Đổi màu các cạnh trên đường mở để tăng cạnh ghép}
var u,v,v1: integer;
begin
    v:= lv;
    u:= lu;
    repeat
        v1:=A[u];

```

```

    A[u]:=v;
    B[v]:=u;
    u:=trace[u];
    v:=v1;
until u=-1;
end;
Procedure Solve; {Thực hiện thuật toán tìm bộ ghép cực đại}
var ok: boolean;
    i,j: integer;
begin
    khoitri; {Khởi trị mảng A, B}
    repeat
        tim_duongmo(ok); {Tìm đường mở}
        if ok then tang_canh; {nếu thấy thì tăng cạnh trên đường mở}
until not ok; {lặp cho đến khi không tìm được đường mở nào nữa}
{iếm số huy chương (số cạnh của bộ ghép cực đại)}
sol:=0;
for j:=1 to K do
    if B[j]<>0 then inc(sol);
{i bổ trí môn đấu cho các vận động viên còn lại (không được huy chương)}
j:=0;
for i:=1 to K do
    if A[i]=0 then begin
        repeat inc(j) until B[j]=0;
        A[i]:=j;
        B[j]:=i003B
    end;
end;
Procedure in_ketqua;
var i: integer;
begin
    assign(g,fo); rewrite(g);
    writeln(g,sol); {số huy chương của trường Super}
    for i:=1 to K do write(g,A[i], ' '); {cách bố trí các môn thi của các
vận động viên trường Super từ 1 đến k}
    close(g);
end;
BEGIN
    doc_dulieu;
    solve;
    in_ketqua;
END.

```

	1	2	3	4
1				
2				
3				
4				

Bài 46. Thi bắn súng (Sho)

Chào mừng bạn đến với cuộc thi bắn súng hàng năm của Byteland. Địch bắn là một hình chữ nhật gồm $r \times c$ ô vuông tạo thành r dòng và c cột. Các ô vuông có màu đen hoặc trắng. Mỗi cột có đúng 2 ô trắng và $r-2$ ô đen. Các dòng được đánh số liên tục từ 1 đến r theo chiều từ trên xuống và các cột được đánh số liên tục từ 1 đến c theo chiều từ trái qua phải. Mỗi đầu thủ có c viên đạn. Một loạt bắn c viên đạn sẽ được công nhận là đạt yêu cầu nếu trên mỗi cột có đúng một ô trắng bị bắn và không có hàng nào không có ô trắng bị bắn. Hãy giúp đầu thủ bắn một loạt c viên đạn đạt yêu cầu.

Ví dụ. Hãy quan sát địch bắn ở hình bên: Loạt đạn trúng các ô ở các dòng 2, 3, 1, 4 thuộc các cột liên tiếp 1, 2, 3, 4 là đạt yêu cầu.

Hãy viết một chương trình sao cho: Đọc số khối dữ liệu từ file văn bản SHO.IN. Mỗi khối chứa một tập đầy đủ dữ liệu giải bài toán.

Với mỗi khối: Đọc kích thước hình chữ nhật (địch bắn) và mô tả các ô trắng của nó. Kiểm tra xem có tồn tại loạt bắn đạt yêu cầu hay không, nếu có thì chỉ ra một loạt bắn. Ghi kết quả vào file văn bản SHO.OUT

Dữ liệu vào. Dòng đầu tiên của file SHO.IN chứa số khối dữ liệu là số X ($1 \leq X \leq 5$). Các dòng theo sau mô tả X khối. Khối thứ nhất bắt đầu từ dòng thứ hai của file input, mỗi khối tiếp theo được bắt đầu liền ngay sau khối trước nó. Dòng thứ nhất của mỗi khối gồm 2 số nguyên r và c cách nhau dấu cách, $2 \leq r \leq c \leq 1000$, tương ứng là số dòng và số cột của địch bắn. Mỗi dòng trong c dòng tiếp theo của khối chứa hai số nguyên cách nhau dấu cách. Các số nguyên trong dòng $i+1$ của khối ($1 \leq i \leq c$) là số hiệu dòng của các ô trắng thuộc cột i .

Kết quả ra. Với khối thứ i , $1 \leq i \leq X$, chương trình sẽ ghi vào dòng thứ i của file SHO.OUT hoặc là dãy gồm c số hiệu dòng (cách nhau dấu cách) của các ô trắng thuộc các cột liên tiếp 1, 2,... c bị trúng đạn trong loạt bắn đạt yêu cầu, hoặc từ NO nếu không thể tồn tại loạt bắn nào đạt yêu cầu.

Ví dụ

Sho.in	Sho.out
2	2 3 1 4
4 4	NO
2 4	
3 4	
1 3	
1 4	
5 5	
1 5	
2 4	
3 4	
2 4	
2 3	

Gợi ý. Đây là bài toán tìm bộ ghép cực đại trên đồ thị hai phía. Tập X là tập gồm các số hiệu hàng từ 1 đến r. Tập Y là tập gồm số hiệu các cột từ 1 đến c. Mỗi ô trắng (i,j) ở hàng i, cột j trên hình chữ nhật $r \times c$ tương ứng là một cạnh của đồ thị hai phía, nối đỉnh $i \in X$ đến đỉnh $j \in Y$. Một cạnh đậm (i,j) thuộc bộ ghép thể hiện có một viên đạn bắn vào ô (i,j). Tuy nhiên do số hàng không vượt quá số cột ($r \leq c$) nên $|X| \leq |Y|$, vì vậy sau khi đã có bộ ghép cực đại (hàng nào cũng có một viên đạn) có thể vẫn còn $c-r$ cột chưa thỏa mãn yêu cầu mỗi cột có đúng một viên đạn. Để hoàn thành đúng yêu cầu này, chỉ việc tìm những cột chưa có viên đạn nào bắn vào, rồi chọn một ô trắng tùy ý ở cột đó để bắn.

Chương trình.

```

const fi      = 'SHO.in';
      fo      = 'SHO.out';
      maxn    = 1000 + 1;
type  mlint   = array[0..maxn] of integer;
      m2int   = array[0..maxn, 1..2] of integer;
var    f, g    : text;
      b, a    : mlint; {B: quản lý đỉnh bên Y (số hiệu cột), A: bên X (số
hiệu dòng)}
      truoc, q : mlint; {truoc: mảng vết, q: hàng đợi dùng trong tìm kiếm}
      w        : m2int; {quản lý các ô trắng của các cột}
      r, c     : longint; {số hàng, số cột}
      socap    : longint; {số cạnh của bộ ghép}
      i, sotest : longint; {sotest: số bộ test trong tệp input}

```

```

Procedure nhap;
var i : integer;

```

```

begin
    read(f, r, c); {đọc số hàng, số cột của hình chữ nhật}
    for i := 1 to c do
        read(f, w[i, 1], w[i, 2]); {đọc hai dòng của ô trắng thuộc cột i}
    end;
Procedure tangcap(j: integer); {Tăng thêm một cạnh đậm trên đường mở, lần
ngược đường mở từ đỉnh nhạt cuối cùng bên A là đỉnh j}
var i, jnext: integer;
begin
    while j <> 0 do begin
        i := truoc[j]; {đỉnh đậm bên B của cạnh đậm cuối cùng của đường mở}
        jnext := b[i]; {đỉnh đậm kia (bên A) của cạnh đậm cuối cùng}
        b[i] := j; {đổi màu cạnh nhạt cuối cùng}
        a[j] := i; {đổi màu cạnh nhạt cuối cùng}
        j := jnext; {chuẩn bị cho quá trình lần ngược được tiếp tục}
    end;
    inc(socap); {số cạnh bộ ghép được tăng thêm một}
end;
Procedure BFS(i: integer); {tìm đường mở, bắt đầu từ đỉnh nhạt i bên B}
var j, p, dau, cuoi: integer;
begin
    fillchar(truoc, sizeof(truoc), 0); {Khởi trị mảng vết}
    dau := 0; {Khởi trị biến đầu hàng đợi}
    cuoi := 1; {Khởi trị biến cuối hàng đợi}
    q[cuoi] := i; {nhập i vào cuối hàng đợi}
    repeat
        inc(dau); {tăng biến đầu hàng đợi}
        i := q[dau]; {lấy i ở đầu hàng đợi}
        for p := 1 to 2 do begin {duyệt 2 đỉnh j bên A, mà có cạnh (i,j)}
            j := w[i, p];
            if truoc[j] = 0 then begin {j chưa thuộc đường mở}
                truoc[j] := i; {thì nhập j vào đường mở, lưu lại vết này}
                if a[j] = 0 then begin {j là đỉnh nhạt bên A thì xong đường mở}
                    tangcap(j); {mở rộng số cạnh đậm trên đường mở}
                    exit;
                end;
            end;
            inc(cuoi); {còn không thì tiếp tục xây dựng đường mở}
            q[cuoi] := a[j]; {nhập đỉnh đậm A[j] của cạnh đậm (j, A[j]) vào q}
        end;
    end;
    until dau = cuoi; {hàng đợi rỗng}
end;
Procedure xuly; {Thuật toán tìm bộ ghép cực đại}
var i: longint;

```

```

begin
  fillchar(b, sizeof(b), 0);
  fillchar(a, sizeof(a), 0);
  socap := 0;
  for i := 1 to c do
    if b[i] = 0 then BFS(i);
  end;
Procedure ketqua;
var i: longint;
begin
  if socap < r then writeln(g, 'NO') {số cạnh của bộ ghép cực đại nhỏ hơn số hàng nên không thể thực hiện yêu cầu của đề bài: vậy bài toán vô nghiệm}
  else begin
    for i := 1 to c do
      if b[i] = 0 then write(g, w[i, 1], ' ') {cột chưa có đạn thì ghi một dòng tùy ý trong hai dòng có ô trống của cột này}
      else write(g, b[i], ' '); {cột đã có đạn thì ghi dòng ghép với cột này}
    writeln(g);
  end;
end;

BEGIN
  assign(f, fi); reset(f);
  assign(g, fo); rewrite(g);
  readln(f, satest);
  for i := 1 to satest do begin
    nhap;
    xuly;
    ketqua;
  end;
  close(f); close(g);
END.

```

Bài 47. Guards (Bố trí các lính gác)

Một lâu đài hình chữ nhật được chia thành $M \times N$ ô vuông đơn vị. Một vài ô là tường, các ô còn lại là trống. Mỗi ô trống gọi là 1 phòng. Một tên vua tham lam và độc ác quyết định xây các hầm bí mật chôn dấu của cải quý báu ở dưới một số phòng trống và sắp đặt một số lính canh trong một số phòng của lâu đài. Tên vua này hạ lệnh cho lính canh nếu họ trông thấy ai là bắt liền vì thế không bố trí hai lính canh ở hai vị trí nhìn thấy nhau, ngoài ra tên vua cũng qui định không bố trí lính tại các phòng có hầm của cải ở dưới và mỗi phòng chỉ bố trí không quá 1 lính canh. Tên vua đã sắp xếp nhiều

lính canh nhất trong lâu đài. Rô -bin - hút cùng các người bạn khởi nghĩa “Rừng xanh” của mình muốn tấn công lâu đài nên cần phải biết trước các cách sắp xếp nhiều lính canh nhất trong lâu đài. Bạn có thể giúp Rô -bin - hút được không?

Dữ liệu vào từ file văn bản GUARDS.IN: Dòng đầu chứa hai số M, N ($1 \leq M, N \leq 150$) là kích thước lâu đài. Dòng thứ i trong M dòng tiếp theo chứa N số $A_{i1}, A_{i2}, \dots, A_{iN}$. Trong đó $A_{ij}=0$ thì ô $[i, j]$ trống, $A_{ij}=1$ thì ô $[i, j]$ là hầm, $A_{ij}=2$ thì ô $[i, j]$ là tường. Toạ độ đầu là dòng, toạ độ sau là cột.

Kết quả ghi ra file văn bản GUARDS.OUT một cách sắp xếp nhiều lính canh nhất: Dòng đầu chứa số K là số lớn nhất các lính canh có thể sắp được trong lâu đài. K dòng tiếp theo chứa các toạ độ ô có lính gác. Mỗi dòng 2 số nguyên: số dòng và số cột của phòng có lính canh.

Ví dụ

GUARDS . IN	GUARDS . OUT
3 4	2
2 0 0 0	1 2
2 2 2 1	3 3
0 1 0 2	

Gợi ý. Ta gọi đoạn dòng là một dãy các ô cùng một dòng, liên tiếp nhau, không chứa tường (chỉ gồm có giá trị 0 hoặc 1) và không thể mở rộng sang trái hoặc sang phải. Đặc biệt, đoạn dòng có thể chỉ là một ô. Đại diện cho mỗi đoạn dòng là toạ độ ô trái nhất của nó và độ dài của đoạn dòng.

Tương tự, gọi đoạn cột là dãy các ô cùng một cột, liên tiếp nhau, không chứa tường và không thể mở rộng lên trên hoặc xuống dưới. Đặc biệt đoạn cột có thể chỉ là một ô. Đại diện cho mỗi đoạn cột là toạ độ ô trên cùng của nó.

Mỗi đoạn dòng hoặc mỗi đoạn cột chỉ có thể chứa nhiều nhất 1 lính gác. Một lính gác cần bố trí tại một ô trống ($=0$) là giao điểm của một đoạn dòng và một đoạn cột mà trong đoạn dòng và đoạn cột này không có lính gác khác.

Xây dựng đồ thị 2 phía $G=(X,Y,E)$: phía X gồm các đỉnh là các đoạn cột (đặc trưng bởi toạ độ ô cao nhất của đoạn cột này), phía Y gồm các đỉnh

là các đoạn dòng (đặc trưng bởi tọa độ ô trái nhất của đoạn dòng này). Hai đỉnh kề nhau nếu đoạn dòng và đoạn cột tương ứng có ô trống chung (giá trị bằng 0).

Tìm cặp ghép cực đại trên đồ thị hai phía này sẽ cho biết số lính canh nhiều nhất. Cảnh đậm trên đường tăng cặp ghép thể hiện bố trí được 1 lính gác đứng tại giao điểm của đoạn dòng và đoạn cột tương ứng với hai đỉnh là hai đầu cạnh đậm này của đường mở trên đồ thị hai phía.

Chương trình.

```
const    mn    = 150;
         fi    = 'guards.in';
         fo    = 'guards.out';
type     m1    = array[0..mn,0..mn] of byte;
         m2    = array[1..mn*mn] of byte;
var      c, ytren, ngang      : ^m1;
         m, n                : byte;
         ax, ay, bx, by, tx, ty : ^m1;
         dem                 : integer;
         qx, qy              : ^m2;
         dau, cuoi           : integer;
```

Procedure read_input;

var f : text;

 i, j : byte;

begin

 assign(f, fi); reset(f);

 readln(f, m, n); *{số dòng và số cột}*

 new(c);

 for i:=1 to m do

 for j:=1 to n do c^[i, j] := 0;

 for i:=1 to m do begin

 for j:=1 to n do read(f, c^[i, j]); *{giá trị ô (i,j): 0/1/2}*

 readln(f);

 end;

 close(f);

end;

function count(i, k : byte): byte;

*{Hàm trả lại tung độ điểm cao nhất trên đoạn cột chứa ô (i, k)}*begin

```

while (c^[i,k]<>2) and (i>=1) do dec(i);
inc(i);
count := i;
end;

Procedure creat_init;
{xây dựng 2 mảng ngang và ytren : ngang[i,j] cho biết độ dài một đoạn dòng có đầu trái là (i,j). Còn ytren[i,k] cho biết tung độ điểm cao nhất trên đoạn cột chứa ô (i,k) }
var i,j,k,lj : byte;
begin
  new(ytren); fillchar(ytren^, sizeof(ytren^), 0);
  new(ngang); fillchar(ngang^, sizeof(ngang^), 0);
  for i:=1 to m do begin {xét từng dòng}
    j := 1;
    repeat {tìm các đoạn dòng nằm trên dòng i}
      while (c^[i,j]<>0) and (j<=n) do inc(j);
      if j<=n then begin {bắt đầu một đoạn dòng là ô (i, j)}
        lj := j;
        k := j;
        while (c^[i,k]<>2) and (k<=n) do begin
          if c^[i,k]=0 then {(i,k) là một ô trống trên đoạn dòng đang xét}
            {tung độ điểm cao nhất trên đoạn cột chứa (i,k)}
            ytren^[i,k] := count(i,k);
            inc(k); {tăng k để sang ô (i, k) tiếp theo bên phải}
          end;
          j := k; {là hoành độ điểm bên phải nhất của đoạn dòng}
          ngang^[i,lj] := j-lj; {độ dài đoạn dòng có ô trái nhất là (i, lj)}
        end;
      until j>=n; {hết một dòng}
    end;
  end;

Procedure init;
begin
  new(ax); new(ay); new(bx); new(by);
  fillchar(ax^, sizeof(ax^), 0); fillchar(ay^, sizeof(ay^), 0);
  fillchar(bx^, sizeof(bx^), 0); fillchar(by^, sizeof(by^), 0);
end;

Procedure inc_edg(jx, jy : byte);

```

```

{tăng cạnh đậm trên đường mở có đỉnh nhạt cuối cùng là (jx,jy)}
var ix,iy,ljx,ljy : byte;
begin
  repeat
    {(ix,iy) là đỉnh đậm cuối cùng bên A trên đường mở}
    ix := tx^[jx,jy];
    iy := ty^[jx,jy];
    {(ljx,ljy) là đỉnh đậm bên B được ghép với đỉnh đậm (ix,iy) bên A}
    ljx := ax^[ix,iy];
    ljy := ay^[ix,iy];
    {đổi màu cạnh nhạt nối đỉnh (ix,iy) với đỉnh (jx,jy)}
    ax^[ix,iy] := jx;
    ay^[ix,iy] := jy;
    bx^[jx,jy] := ix;
    by^[jx,jy] := iy;
    {vị trí mới của đỉnh cuối bên B trong quá trình lần ngược đường mở}
    jx := ljx;
    jy := ljy;
  until (jx=0) {and (jy=0)};
end;

Procedure find_path(ix,iy: byte); {tìm đường mở bằng BFS}
var k : byte;
begin
  dau := 0; {biến chỉ vào đầu hàng đợi}
  cuoi := 1; {biến chỉ vào cuối hàng đợi}
  fillchar(qx^, sizeof(qx^), 0); {hàng đợi quản lý tọa độ dòng của một ô}
  fillchar(qy^, sizeof(qy^), 0); {hàng đợi quản lý tọa độ cột của một ô}
  qx^[cuoi] := ix; { nạp phần tử đầu tiên vào hàng đợi }
  qy^[cuoi] := iy;
  repeat
    inc(dau);
    ix := qx^[dau];
    iy := qy^[dau];
    for k:= iy to iy+ngang^[ix,iy]-1 do {xét các ô trên đoạn dòng}
      begin
        if c^[ix,k]=0 then {(ix,k) là ô trống}
          if tx^[ytren^[ix,k],k]=0 then begin

```

```

    {đỉnh (ytren^[ix,k], k) chưa thăm}
    tx^[ytren^[ix,k], k] := ix; {lưu vết đã thăm trong tìm kiếm}
    ty^[ytren^[ix,k], k] := iy;
    if (bx^[ytren^[ix,k], k]=0) then {(ytren^[ix,k],k) là đỉnh nhậ}
    begin
        inc_edg(ytren^[ix,k], k); {tăng cạnh bộ ghép}
        inc(dem); {thêm một linh canh gác}
        exit;
    end
else begin {tiếp tục tìm kiếm đường mở bằng BFS}
    inc(cuoi);
    qx^[cuoi] := bx^[ytren^[ix,k], k];
    qy^[cuoi] := by^[ytren^[ix,k], k];
end;
end;
end;
until dau=cuoi;
end;

Procedure process; {Thực hiện thuật toán tìm bộ ghép cực đại}
var i, j : byte;
begin
    dem := 0; init;
    new(tx); new(ty); new(qx); new(qy);
    for i:=1 to m do
    for j:=1 to n do
    if ytren^[i,j]>0 then begin {đỉnh nhậ bên X}
        fillchar(tx^, sizeof(tx^), 0); fillchar(ty^, sizeof(ty^), 0);
        find_path(i, j); {tìm đường mở, bắt đầu từ ô (ytren^[i,j], j)}
    end;
    dispose(tx); dispose(ty);
end;

Procedure write_output;
var f : text; i, j : byte;
begin
    assign(f, fo); rewrite(f);
    writeln(f, dem);
    for i:=1 to m do

```

```

for j:=1 to n do
if (ax^[i,j]>0) and (ay^[i,j]>0) then begin
    writeln(f,i,' ',ay^[i,j]);
end;
close(f);
end;
BEGIN
    read_input;
    creat_HV;
    process;
    write_output;
END.

```

Bài 48. Sắp xếp hoa trong triển lãm

Bratislava có truyền thống lâu đời tổ chức triển lãm hoa quốc tế Flora. Giáo sư Andrây rất yêu hoa và thăm Flora trong năm nay. Ông hài lòng về những bông hoa đẹp và cách cắm hoa của mọi người. Dù các loại hoa: Hồng, lan, mộc lan, xương rồng..., tất cả những bông hoa đã được trưng bày rất khéo. Tuy nhiên, cách sắp xếp những bông hoa đã lỗi cuốn ông ta nhất (ít nhất là về phương diện toán học) đó là cách sắp xếp nhiều loại hoa trên một lưới hình chữ nhật sao cho:

Trên mỗi hàng của lưới mỗi loại hoa xuất hiện đúng 1 lần

Trên mỗi cột của lưới mỗi loại hoa xuất hiện nhiều nhất là 1 lần

Giáo sư Andrây là một nhà toán học giỏi, ông đã sớm nhận thấy số cột của lưới bằng số loại hoa khác nhau của cách sắp xếp. Nhưng ông cũng tìm thấy một bài toán mà ông chưa kịp giải quyết: Giáo sư muốn thêm vào cách sắp xếp này nhiều hàng hoa nữa mà không vi phạm quy luật nêu trên. (Chú ý rằng không thay đổi những hàng đã có và cũng không dùng loại hoa mới trong hàng mới). Hãy giúp giáo sư thêm càng nhiều hàng mới càng tốt.

Dữ liệu vào từ file A1.IN miêu tả một vài cách sắp xếp, dòng thứ nhất cho số cách sắp xếp. Mô tả mỗi cách sắp xếp bắt đầu bằng 2 số nguyên dương N và M là số cột và dòng của lưới. Các loại hoa khác nhau được đánh số là 1, 2, 3, ..., N. M dòng tiếp theo mỗi dòng chứa N số nguyên, mỗi số mô tả một loại hoa có trong một hàng của cách sắp xếp.

Kết quả ghi ra file A1.OUT: Với mỗi cách sắp xếp ghi số K là số dòng lớn nhất có thể thêm vào cách sắp xếp này. Sau đó in K dòng, mỗi dòng chứa N số mô tả dòng thêm vào. Các số trong file output có thể cách nhau một số tùy ý dấu trống.

A1.IN	A1.OUT
2	1
	2 1 3
3 2	
3 2 1	2
1 3 2	3 2 1 4
	4 3 2 1
4 2	
1 4 3 2	
2 1 4 3	

Gợi ý. Rõ ràng cần phải thêm N-M dòng hoa mới. Xây dựng đồ thị 2 gồm tập đỉnh X và tập đỉnh Y. Các đỉnh thuộc X tương ứng với các cột của lưới. Các đỉnh thuộc Y tương ứng với các loại hoa. Mảng A quản các đỉnh bên X, mảng B quản các đỉnh bên Y. Một cột j thuộc X và một loại hoa i thuộc Y được nối bởi một cạnh nếu cột j chưa có loại hoa i. Bộ ghép đầy đủ của đồ thị này sẽ cho một hàng hoa mới thêm vào lưới và ngược lại.

Chương trình.

```
uses crt;
const fi  = 'a1.in';
      fo  = 'a1.out';
      maxn = 100;
var a,b,trace,q : array[0..maxn] of integer;
    e : array[1..maxn,1..maxn] of byte;
    test : integer;
    n : integer;
    m : integer;
    f, g : text;
    i,j ,ii,t,k,h : integer;
Procedure tangcanh(j : integer); {Tăng thêm cạnh trên đường mở}
var i, jnext : integer;
begin
  while j<>0 do begin
    i := trace[j];
    jnext := b[i];
    b[i] := j;
```

```

    a[j] := i;
    j := jnext;
end;
end;
Procedure BFS(i: integer); {Tìm đường mở bắt đầu từ đỉnh nhạ i bên B (hoa)}
var k, j, p, dau, cuoi : integer;
begin
    for k:=1 to n do trace[k] := 0;
    dau := 0; cuoi := 1;
    q[cuoi] := i;
    repeat
        inc(dau);
        i := q[dau];
        for j:=1 to n do
            if e[i,j]=1 then {loại hoa j còn vắng mặt trên cột i}
            if trace[j]=0 then begin {j chưa thuộc đường mở}
                trace[j] := i; { nạp j vào đường mở}
                if a[j]=0 then begin {j là đỉnh nhạ thì kết thúc đường mở}
                    tangcanh(j); {tăng thêm cạnh đậm trên đường mở}
                    exit; {thoát}
                end; {nếu j là đỉnh đậm nối với đỉnh đậm a[j] thì tiếp tục kéo dài đường mở}
                inc(cuoi);
                q[cuoi] := a[j];
            end;
        until dau=cuoi;
    end;
Procedure capghiep;
var i : integer;
begin
    for i:=1 to n do {Thuật toán tìm bộ ghép đầy đủ}
        if b[i]=0 then BFS(i);
    end;
Procedure ghi; {ghi một hàng mới xếp hoa phù hợp đề bài}
var i : integer;
begin
    for i:=1 to n do {duyet các cột}
        if b[i]>0 then write(g,b[i], ' '); {ghi hoa b[i] vào cột i}
        writeln(g);
    end;
Procedure capnhathat_e;
var i: integer;
begin
    for i:=1 to n do {xóa các cạnh do có hàng mới tạo ra}
        if b[i]>0 then e[i,b[i]]:= 0;
    end;

```



```

BEGIN
    assign(f,fi); reset(f);
    assign(g,fo); rewrite(g);
    readln(f,test);
    for t := 1 to test do begin {đọc và thực hiện từng test}
        readln(f,n,m);
        for i:=1 to n do
            for j:=1 to n do e[i,j] := 1; {tạm cho cột i có thể xếp hoa j}
            for k:=1 to m do
                for j:=1 to n do begin
                    read(f,h);
                    e[j,h] := 0; {cột j không thể xếp hoa h}
                end;
            writeln(g,n-m); {số hàng mới cần tạo thêm}
            for ii:=1 to n-m do begin {Lần lượt tìm từng hàng mới}
                for k:=1 to n do begin
                    a[k]:= 0; b[k]:=0;
                end;
                capghep;
                ghi;
                capnhat_e;
            end;
        end;
        close(f); close(g);
    END.

```

Bài 49. Minimum path cover (Tập phủ đường tối thiểu)

Cho một đồ thị có hướng, không chu trình là $G(V, E)$, trong đó tập đỉnh V có N đỉnh đánh từ 1 đến N và E là tập cung gồm M cung.

Người ta định nghĩa một đường đi là một dãy các đỉnh x_1, x_2, \dots, x_k thoả mãn: có các cung (x_i, x_{i+1}) ($i=1, 2, \dots, k-1$) và các đỉnh của dãy đôi một khác nhau. Đường đi có độ dài bằng 0 hiểu là đường đi chỉ gồm 1 đỉnh. Một bộ phủ đường của đồ thị là một bộ các đường đi sao cho mỗi đỉnh của đồ thị thuộc đúng một đường đi của bộ đó.

Hạn chế kỹ thuật : $N \leq 100$.

Hãy tìm bộ phủ đường tối thiểu (có số đường đi ít nhất).

Dữ liệu vào từ file văn bản PHU.INP tổ chức như sau:

Dòng đầu là hai số nguyên N, M .

M dòng tiếp theo: mỗi dòng 2 số i, j là hai đỉnh đầu và cuối của một cung (i, j) .

Kết quả ra ghi vào file văn bản PHU.OUT tổ chức như sau:

Dòng đầu ghi số s là số lượng đường đi của bộ phủ nhỏ nhất.

S dòng sau : mỗi dòng một đường đi (dãy các đỉnh liên tiếp của đường đi này).

Ví dụ.

PHU . IN	PHU . OUT
5	2
1 2	1 2 5 4
1 4	3
2 5	
5 4	
1 3	
3 5	
0 0□	

Gợi ý. Từ đồ thị $G=(V, E)$, xây dựng đồ thị hai phía $G'=(X,Y,E')$ trong đó tập $X=\{x_1, x_2, \dots, x_N\}$ và $Y=\{y_1, y_2, \dots, y_N\}$, với mỗi cung $(i,j) \in E$, xây dựng cung $(x_i, y_j) \in E'$.

Để thấy bộ phủ đường ban đầu là tập V (mỗi đỉnh là một đường đi có độ dài bằng 0). Để tìm bộ phủ đường tối thiểu, ta thực hiện:

+ Tìm bộ ghép cực đại trên đồ thị hai phía G' . Sau đó chấp x_i và y_i (vì x_i và y_i tương ứng ở hai tập X và Y chỉ đại diện cho một đỉnh là đỉnh i) ta sẽ nối được một số cạnh của bộ ghép thành một đường đi. Khi nối hết các cạnh của bộ ghép cực đại thì được các đường đi tạo thành bộ phủ đường tối thiểu.

Chương trình sau đây thực hiện tìm bộ ghép cực đại với ý tưởng tìm đồng thời nhiều đường mở.

Chương trình.

```
const maxn = 101;
      maxm = 5000;
      fi   = 'bophu.in';
      fo   = 'bophu.out';
type   arr1 = array[1..maxm] of integer;
      arr2 = array[1..maxn] of integer;
      arr3 = array[1..maxn] of boolean;
```

```

var    n, m, nList : integer;
      adj, link    : arr1;
      head, A, B,
      list         : arr2;
      avail        : arr3;

```

Procedure doc_dulieu;

```

var i,x,y : integer; f : text;

```

```

begin

```

```

    assign(f,fi); reset(f);

```

```

    readln(f,n,m); {Số đỉnh, số cung}

```

```

    {tổ chức danh sách các cung theo kiểu liên kết}

```

```

    for i:=1 to n do head[i] := 0;

```

```

    for i:=1 to m do begin

```

```

        readln(f,x,y); {đọc cung (x,y)}

```

```

        adj[i]:=y; {đỉnh cuối của cung thứ i là y}

```

```

        link[i] := head[x]; {liên kết của cung thứ i}

```

```

        head[x] := i; {x là đỉnh xuất phát của cung thứ i }

```

```

    end;

```

```

    close(f);

```

```

end;

```

Procedure init;

```

var i: integer;

```

```

begin

```

```

    for i:=1 to n do begin{Khởi trị mảng A và B quản lý màu các đỉnh là nhạt}

```

```

        A[i] := 0; B[i] := 0;

```

```

    end;

```

```

    for i:=1 to n do list[i] := i; {Tập đường phủ ban đầu}

```

```

    nlist := n; {số đường trong bộ phủ này}

```

```

end;

```

Procedure cacduongmo; {Xây dựng các đường mở}

```

var found : boolean;

```

```

    old, i : integer;

```

Procedure visit(x: integer); {thăm từ đỉnh x}

```

var i, y : integer;

```

```

begin

```

```

    i := head[x]; {x là đỉnh đầu của cung thứ i}

```

```

    while i<>0 do begin {duyet mọi cung xuất phát từ x}

```

```

        y := adj[i]; {y là điểm cuối của một cung xuất phát từ x}

```

```

        if avail[y] then begin {y chưa thăm}

```

```

            avail[y] := false; {đánh dấu đã thăm y}

```

```

            if B[y]=0 then found:= true {y là đỉnh nhạt: có một đường mở}

```

```

            else visit(B[y]); {còn không thì thăm tiếp từ đỉnh đã ghép với y}

```

```

            if found then begin {nếu có đường mở thì ghép x với y và thoát}

```

```

                B[y] := x;

```

```

                A[x] := y;

```

```

        exit;
    end;
end;
    i := link[i]; {còn không thì chuyển sang cung khác xuất phát từ x}
end;
end;
begin {chương trình tìm đồng thời các đường mở}
    repeat {Thực hiện lặp}
        old := nList; {số đỉnh hiện tại còn có thể là xuất phát của một đường mở}
        for i:=1 to n do avail[i] := true; {các đỉnh chưa thăm}
        {tìm các đường mở từ mọi đỉnh trong danh sách các đỉnh hiện tại}
        for i:=nList downto 1 do begin
            found := false;
            visit(list[i]); {thăm một đỉnh trong danh sách}
            if found then begin {có đường mở thì loại bỏ đỉnh i trong danh sách}
                list[i] := list[nlist];
                dec(nlist);
            end;
        end;
    until old=nlist; {số đỉnh trong danh sách không thể giảm được nữa}
end;
```

```

Procedure in_ketqua;
var i, k, dem      : integer;
    g              : text;
begin
    assign(g,fo); rewrite(g);
    dem := 0;
    for i:=1 to n do
        if B[i]=0 then inc(dem);
    writeln(g,dem);

    for k:=1 to n do begin
        i := k;
        if B[i]=0 then begin
            if A[i]=0 then writeln(g,i)
            else begin
                write(g,i,' ');
                repeat
                    write(g,A[i], ' ');
                    i := A[i];
                until A[i]=0;
                writeln(g);
            end;
        end;
    end;
end;
```

```

close(g);
end;

BEGIN
    doc_dulieu;
    init;
    cacduongmo;
    in_ketqua
END.

```

Bài 50. Minimum vertex cover (Tập phủ đỉnh tối thiểu)

Maze – Mê cung. Một mê cung có dạng một hình chữ nhật $M \times N$ chia thành các ô vuông đơn vị. Một số ô có đặt vật cản, các ô còn lại thì không. Người ta đặt các camera tại một số ô để theo dõi mê cung. Mỗi camera được lắp để kiểm soát theo một trong các hướng trên-dưới hoặc trái-phải theo các cạnh của mê cung. Khi đặt camera tại một ô nào đó thì các ô dọc theo hướng quay của camera chừng nào vẫn chưa bị chắn bởi vật cản sẽ theo dõi được (kể cả ô đặt camera).

Yêu cầu: Tìm cách đặt một số ít nhất các camera tại các ô trống sao cho tất cả các ô không có vật cản đều nằm trong tầm kiểm soát.

Dữ liệu vào: Tập Maze.in

Dòng đầu ghi hai số nguyên dương M, N ($M, N \leq 50$).

M dòng tiếp theo, dòng thứ i ghi trạng thái của các ô trên hàng i của ma trận, gồm N số 0/1 viết liên tiếp thể hiện ô tương ứng không/có vật cản.

Kết quả ra: Tập Maze.out

Dòng đầu ghi k là số nhỏ nhất các camera cần thiết. K dòng tiếp theo, mỗi dòng mô tả một camera gồm ba số là chỉ số hàng, chỉ số cột của ô đặt và 0/1 tùy theo hướng trên xuống (dọc theo cột)/ trái phải (dọc theo hàng).

Ví dụ

Maze.in	Maze.out
5 7	6
0000000	1 1 1
1000111	2 2 1
0000000	3 1 1
0111100	4 1 0
0111110	4 6 1

Gợi ý.

Nếu coi mỗi đoạn nằm ngang hoặc nằm dọc gồm các ô rỗng liên tiếp là một đỉnh đồ thị (gọi tương ứng là đoạn dòng hoặc đoạn cột). Mỗi đỉnh là đoạn dòng đặc trưng bởi tọa độ ô bên trái nhất của nó, mỗi đỉnh là đoạn cột được đặc trưng bởi ô bên trên nhất của nó. Cần đặt camera tại một số ô đặc trưng này sao cho số ô được chọn là ít nhất. Khi đoạn dòng và đoạn cột có ô rỗng chung thì có cạnh nối hai đỉnh tương ứng với đoạn dòng và đoạn cột này. Vì các camera cần quan sát hết các ô trống nên dẫn tới bài toán phủ đỉnh tối thiểu (Minimum Vertex Cover) trên đồ thị hai phía $G=(X, Y, E)$: Nghĩa là cần tìm tập C gồm ít đỉnh nhất sao cho mọi cạnh của đồ thị đều liên thuộc với ít nhất một đỉnh của C .

Giải bài toán phủ đỉnh tối thiểu dựa trên việc tìm bộ ghép cực đại M được tiến hành như sau: Sau khi tìm được M , không còn đường mở nhưng còn các đường pha. Đường pha là đường xuất phát từ đỉnh nhạ x bên X , xen kẽ cạnh nhạ và cạnh đậm và kết thúc tại cạnh đậm (vì không còn cạnh nhạ tiếp theo nữa). Tiếp theo, tìm một lát cắt hẹp nhất (X^*, Y^*) theo quy tắc:

$$Y^* = \{y \in Y: \text{có đường pha đến } y\}$$

$$X^* = \{x \in X: x \text{ là đỉnh đậm không ghép với đỉnh } y \in Y^*\}$$

$$\text{Tập đỉnh phủ tối thiểu là } C = X^* \cup Y^*.$$

Chương trình.

```
const  fi    = 'maze.in';
       fo    = 'maze.out';
       maxs  = 51;
       max   = 1300;
type   arr1  = array[0..maxs, 0..maxs] of integer;
       arr2  = array[1..max] of Integer;
var map    : arr1; {ma trận mê cung}
    queue  : arr2; {hàng đợi dùng trong tìm kiếm bộ ghép cực đại}
    trace, {lưu vết trong tìm kiếm đường mở}
    A, B,  {quản lý các đỉnh đậm/nhạ bên X và bên Y}
    dx, cx, {tọa độ dòng và cột của ô đầu của một đoạn dòng trên bản đồ}
    dy, cy, {tọa độ dòng và cột của ô đầu của một đoạn cột trên bản đồ}
    ddx, ddy : arr2; {đánh dấu đỉnh đã xét trên X và trên Y}
    E, P: array[1..2*max] of integer; {tổ chức danh sách kề forward star}
```

```

first, last,  {biến đầu và cuối hàng đợi}
finish : integer; {biến cho biết đã tìm được đường mở hay chưa}
m, n      : integer; {kích thước ma trận mê cung: số dòng và số cột}
nx, ny    : integer; {số lượng đoạn dòng, số lượng đoạn cột}
g         : text;

```

Procedure doc_dulieu; {đọc ma trận bản đồ mê cung}

```

var f : text;  t : char;  i, j : Integer;
begin
  assign(f, fi); reset(f);
  readln(f, m, n);
  fillchar(map, sizeof(map), 0);
  for i := 1 to m do begin{i: tọa độ dòng}
    for j := 1 to n do begin {j: tọa độ cột}
      read(f, t);
      if t = '0' then map[i, j] := 1; {ô (i,j) rỗng}
    end;
    readln(f);
  end;
  close(f);
end;

```

Procedure chuanbi; {tìm các đoạn dòng, đoạn cột}

```

var i, j, pre : integer;
    top      : integer;
begin
  ny := 0; {số lượng đoạn cột}
  pre := 0; {biến phụ dùng để báo bắt đầu vào ô đầu tiên của một đoạn cột }
  for j := 0 to n do
    for i := 0 to m do begin
      if (pre = 0) and (map[i, j] <> 0) then begin
        inc(ny);
        dy[ny] := i; cy[ny] := j; {lưu tọa độ ô đầu tiên đoạn cột}
      end;
      if map[i, j] <> 0 then map[i, j] := ny; {phủ ô bằng số hiệu đoạn cột}
      pre := map[i, j];
    end;
  nx := 0; {số lượng đoạn dòng}
  top := 1;
  pre := 0; {biến phụ dùng để báo bắt đầu vào ô đầu tiên của một đoạn dòng}
  for i := 0 to m do
    for j := 0 to n do begin
      if (pre = 0) and (map[i, j] <> 0) then begin
        inc(nx); {thêm một đoạn dòng mới}
        P[nx] := top; {top: trở vị trí của ô hiện tại thuộc đoạn dòng thứ nx}
        dx[nx] := i; {lưu tọa độ dòng của ô đầu tiên thuộc đoạn dòng thứ nx}
        cx[nx] := j; {lưu tọa độ cột của ô đầu tiên thuộc đoạn dòng thứ nx}
      end;
    end;
  end;

```

```

    end;
    if map[i, j] <> 0 then begin
        E[top] := map[i, j]; {số hiệu đoạn cột chứa ô thứ top của đoạn dòng}
        inc(top);
    end;
    pre := Map[i, j];
end;
P[nx+1] := top; {phục vụ khi duyệt các ô của đoạn dòng thứ nx}
end;
Procedure ghiep_bandau; {tạo bộ ghép ban đầu}
var
    x, i, y : Integer;
begin
    fillChar(A, sizeof(A), 0);
    fillChar(B, sizeof(B), 0);
    for x := 1 to nx do {duyet các đoạn dòng}
        for i := P[x] to P[x+1]-1 do begin {trở vào các ô của đoạn dòng x}
            y := E[i]; {y: đoạn cột có chứa ô đang xét của đoạn dòng x}
            if B[y] = 0 then begin {y là đỉnh nhật thì ghép}
                A[x] := y; {ghép x và y}
                B[y] := x;
                trace[y] := x; {lưu vết}
                break;
            end;
        end;
    end;
end;
Procedure push(x : Integer);
begin
    inc(last);
    queue[last] := x;
end;
function get : integer;
begin
    inc(first);
    get := queue[first];
end;
Procedure nap_bandau; {Nạp các đỉnh nhật bên X, là các đỉnh xuất phát của các
đường mở}
var i : Integer;
begin
    first := 0;
    last := 0;
    finish := 0;
    fillchar(trace, sizeof(trace), 0);
    for i := 1 to nx do

```



```

    if A[i] = 0 then push(i);
end;

```

Procedure tim_duongmo; *{Tìm đường mở}*

var x, i, y : integer;

begin

if first = last then exit;

repeat *{Thực hiện lặp}*

x := Get; *{lấy một đỉnh nhạ bên X}*

for i := P[x] to P[x+1]-1 do begin

y := E[i]; *{lần lượt xét các đỉnh y kề với x}*

if trace[y] = 0 then begin *{y chưa thăm}*

trace[y] := x; *{đánh dấu đã thăm y}*

if B[y] = 0 then begin *{y là đỉnh nhạ thì kết thúc một đường mở}*

finish := y; *{lưu lại đỉnh kết thúc của đường mở}*

exit;

end else push(B[y]); *{nạp đỉnh x bên X đã ghép với y vào hàng đợi để tiếp tục kéo dài đường mở}*

end;

end;

until first = last;

end;

Procedure tang_canh; *{tăng cạnh ghép trên một đường mở vừa tìm được}*

var x, y, next : Integer;

begin

y := finish; *{đỉnh cuối đường mở bên Y}*

repeat

x := trace[y];

next := A[x];

A[x] := y;

B[y] := x;

y := next;

until y = 0;

end;

Procedure xuly; *{tìm các đường mở để hình thành bộ ghép cực đại}*

begin

chuanbi; *{tạo các đoạn dòng, đoạn cột làm các đỉnh của đồ thị hai phía G(X,Y,E)}*

ghép_bandau; *{tạo các cạnh ghép ban đầu}*

repeat *{lặp cho đến khi không còn tìm được đường ở nào nữa}*

nạp_bandau; *{nạp các đỉnh nhạ bên X vào hàng đợi}*

tim_duongmo; *{thực hiện lần lượt tìm các đường mở}*

if finish<>0 then tang_canh; *{có một đường mở thì tăng cạnh ghép}*

until finish = 0; *{cho đến khi không còn đường mở}*

end;

Procedure tim_duongpha(x : integer);

```

{sau khi có bộ ghép cực đại, tìm các đường pha xuất phát từ đỉnh nhạc x thuộc X}
var i, j : integer;
begin
    if P[x+1]-1 < P[x] then exit; {điều kiện: đỉnh x không có đỉnh kế}
    for i := P[x] to P[x+1]-1 do begin {xét các vị trí trở vào đỉnh kế}
        j := E[i]; {j là một đỉnh kế}
        {j là một đoạn cột, là đỉnh đậm bên Y, j chưa được xét thì;}
        if (j > 0) and (B[j] > 0) and (ddy[j] = 0) then begin
            ddx[B[j]] := 1; {đánh dấu đã xét đỉnh B[j] bên X (đỉnh ghép với j)}
            writeln(g, dy[j], ' ', cy[j], ' ', 0); {tọa độ ô đầu của đoạn cột j}
            ddy[j] := 1; {đánh dấu đã xét j}
            tìm_duongpha(B[j]); {đệ quy kéo dài đường pha}
        end;
    end;
end;
end;
Procedure in_ketqua;
var c, i, j: integer;
begin
    assign(g, fo); rewrite(g);
    c := 0;
    for i:=1 to nx do {đếm số cạnh của bộ ghép cực đại, cũng là số đỉnh của tập
phủ định tối thiểu}
        if A[i] > 0 then inc(c);
        writeln(g, c); {ghi số đỉnh của tập phủ định tối thiểu}
        fillchar(ddx, sizeof(ddx), 0);
        fillchar(ddy, sizeof(ddy), 0);
        for i:=1 to nx do begin {duyệt các đỉnh bên A tìm đường pha}
            if A[i] = 0 then tìm_duongpha(i); {ghi các đỉnh của tập Y*}
        end;
        for i:=1 to nx do {tìm ghi các đỉnh của tập X*}
            if (A[i] > 0) and (ddx[i] = 0) then
                writeln(g, dx[i], ' ', cx[i], ' ', 1);
        close(g);
    end;
BEGIN
    doc_dulieu;
    xuly;
    in_ketqua;
END.

```

Bài 51. Maximum independent set (Tập độc lập cực đại)

Knight. Cho một bàn cờ kích thước $N \times N$ đã bỏ đi một vài ô. Nhiệm vụ là tìm số lớn nhất các quân mã có thể đặt trên các ô còn lại của bàn cờ sao cho không có hai quân mã nào khống chế được nhau.

Hình bên: Quân mã đặt tại ô S có thể không chế các ô được đánh dấu x.

	x		x	
x				x
		S		
x				x
	x		x	

Nhiệm vụ. Viết chương trình thực hiện các yêu cầu sau: Đọc miêu tả về bàn cờ với một vài ô bỏ đi từ file input KNI.IN. Tìm số lớn nhất các quân mã có thể đặt trên bàn cờ sao cho không có hai quân mã nào khống chế nhau. Ghi kết quả vào file output KNI.OUT

Input. Dòng đầu tiên của file input KNI.IN chứa hai số nguyên N và M, cách nhau một dấu cách, $1 \leq N \leq 200, 0 \leq M < N^2$; N là kích thước bàn cờ, M là số ô bị bỏ đi. Mỗi một dòng trong M dòng tiếp theo chứa hai số nguyên x và y ($1 \leq x, y \leq N$) cách nhau bởi một dấu cách, là toạ độ của những ô bị bỏ đi. Toạ độ của góc trái trên của bàn cờ là (1, 1) và góc phải dưới là (N, N). Danh sách các ô bỏ đi không chứa ô lặp lại.

Output. File output KNI.OUT chỉ chứa một số nguyên (trên dòng đầu tiên và là dòng duy nhất của file). Đó là số lớn nhất các quân mã có thể đặt trên bàn cờ đã cho sao cho chúng không thể khống chế lẫn nhau.

Ví dụ

KNI.IN	KNI.OUT
3 2	5
1 1	
3 3	

Gợi ý.

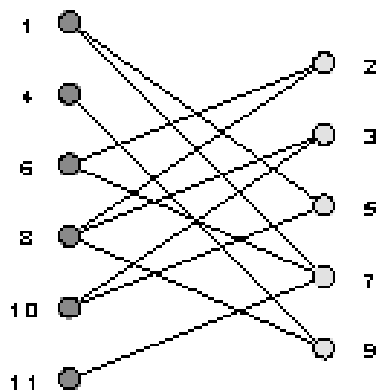
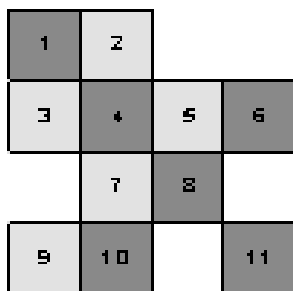
Trước hết chúng ta nêu một vài khái niệm :

Cho đồ thị vô hướng $G=(V, E)$ có tập đỉnh là V và tập cạnh là E.

- a. Tập độc lập là tập con A của tập đỉnh V sao cho không có hai phần tử nào của A được nối với nhau bằng một cạnh. *Tập độc lập cực đại* là tập độc lập có lực lượng lớn nhất.
- b. Tập đỉnh phủ các cạnh là tập con B của tập đỉnh V sao cho bất kỳ một cạnh nào thuộc E cũng đều có ít nhất một đầu thuộc B. Tập đỉnh phủ các cạnh cực tiểu (gọi là *tập phủ đỉnh cực tiểu*) là tập đỉnh có lực lượng nhỏ nhất phủ các cạnh.

Trong bài toán này, xây dựng đồ thị gồm các đỉnh là các ô của bàn cờ. Hai đỉnh có cạnh nối nếu quân mã từ đỉnh này có thể nhảy tới đỉnh kia. Vậy bài toán của chúng ta là tính số phần tử của tập độc lập cực đại.

Do một quân mã chỉ nhảy từ ô trắng đến ô đen theo một trong 8 hướng và ngược lại, do đó cần xây dựng đồ thị hai phía : một phía gồm các ô đen, phía còn lại gồm các ô trắng.



Ví dụ bàn cờ 4x4 có một vài ô bị bỏ, đồ thị hai phía với các bước chuyển của mã

Bài toán tìm tập độc lập cực đại là bài toán sinh đôi với bài toán tìm phủ các cạnh cực tiểu vì $|A|=N-|B|$, với A là tập độc lập cực đại và B là tập phủ các cạnh cực tiểu, N là số đỉnh của đồ thị.

Trong đồ thị hai phía, có một quan hệ giữa bộ ghép cực đại M và tập phủ đỉnh cực tiểu B; đó là số phần tử của bộ ghép cực đại đúng bằng số phần tử của tập phủ đỉnh cực tiểu: $|M|=|B|$, suy ra $|A|=|V|-|M|$.

Vậy bài toán của chúng ta được tập trung chủ yếu vào công việc tìm lực lượng của bộ ghép cực đại.

Tuy nhiên, tổ chức dữ liệu cũng là một khó khăn của bài toán này. Với kích thước bàn cờ 200x200, đồ thị hai phía sẽ có 20000 đỉnh mỗi phía.

Sau đây để tiện trình bày, các đỉnh bên A tương ứng với ô đen bàn cờ thuộc dòng d_1 , cột c_1 sẽ kí hiệu là $A(d_1, c_1)$; các đỉnh bên B tương ứng với các ô trắng bàn cờ thuộc dòng d_2 , cột c_2 sẽ kí hiệu là $B(d_2, c_2)$. Có thể dùng các mảng sau:

Quản lí các ô bàn cờ bằng mảng $A[1..200, 1..200]$. Ô bỏ đi được cho giá trị 2, các ô còn lại cho giá trị 0.

Khi tìm kiếm đường mở bằng BFS, dùng 2 mảng là $qd[1..20000]$ of byte và $qc[1..20000]$ of byte đóng vai trò hàng đợi để lưu lại toạ độ dòng và toạ độ cột của các đỉnh bên A thuộc đường mở trong quá trình tìm kiếm.

Để theo dõi vết của đường mở, dùng mảng $tr[1..200, 1..200]$ of byte. Giả sử, dọc theo đường mở, từ đỉnh $A(d_1, c_1)$ mà theo hướng h nhảy sang đỉnh $B(d_2, c_2)$ thì gán $tr[d_2, c_2] := h$.

Để thể hiện các đỉnh đậm dùng mảng $d[1..200, 1..200]$ of byte. Cụ thể: Nếu đỉnh $A(d_1, c_1)$ được ghép với $B(d_2, c_2)$ mà từ $A(d_1, c_1)$ mà phải nhảy theo hướng h mới tới được $B(d_2, c_2)$ thì gán $d[d_1, c_1] := h$ và $d[d_2, c_2] := h$.

Ngoài phương pháp trên, có thể dùng luồng cực đại và lát cắt hẹp nhất để giải bài toán này (vì bản chất bài toán là tìm tập độc lập cực đại).

Chương trình.

```
uses crt;
const fi='kni.in';
      fo='kni.out';
      dx:array[1..8] of shortint = (-2,-2,-1, 1,-1, 1, 2, 2);
      dy:array[1..8] of shortint = (-1, 1, 2, 2,-2,-2,-1, 1);
      max = 202;
      max1 = 20001;
type mang = array[-1..max,-1..max] of byte;
      mang1 = array[0..max1] of byte;
var a, {bàn cờ}
    d, {quản lý các đỉnh đậm}
    tr: ^mang; {theo dõi vết của đường mở}
```

```

qd, qc: ^mang1; {hàng đợi dùng trong tìm kiếm đường mở}
n, m : longint; {kích thước bàn cờ, số ô bị bỏ đi}
sl : longint; {số cạnh của bộ ghép}
f, g : text;

```

Procedure capphat;

```

begin
    new(a);
    new(d); fillchar(d^, sizeof(d^), 0);
    new(tr); new(qd); new(qc);
end;

```

Procedure doc_dulieu;

```

var i, j, k : integer;
begin
    assign(f, fi); reset(f);
    readln(f, n, m);
    {Tạo hàng rào xung quanh bàn cờ bởi các ô số 2}
    for i:=-1 to n+2 do
    for j:=-1 to n+2 do a^[i, j] := 2;
    {Tạo bàn cờ: ô đen là 1, ô trắng là 0}
    for i:=1 to n do
    for j:=1 to n do
        if (i+j) and 1 = 1 then a^[i, j] := 1
        else a^[i, j] := 0;
    {Đọc các ô bỏ đi: gán cho các ô này số 2}
    for k:=1 to m do begin
        readln(f, i, j); a^[i, j] := 2;
    end;
    close(f);
end;

```

Procedure tangcanh(lx, ly : integer); {đổi màu các cạnh trên đường mở có điểm kết thúc là đỉnh nhạ (lx, ly) bên B}

```

var k, h, x, y : integer;
begin
    inc(sl); {tăng thêm số cạnh trên bộ ghép}
    h:=tr^[lx, ly]; {hướng ra khỏi ô (x,y) bên A đến ô (lx, ly) bên B}
    while h<>0 do begin {Tìm ô (x,y) là ô mã đứng trước khi nhảy theo hướng h}
        x:= lx - dx[h]; y:= ly - dy[h];
    end;

```

```

k:= d^[x,y]; {hướng nhảy của hai ô cuối cùng ghép với nhau trên đường mở}
if k=0 then exit;
{đổi màu cạnh nhạt cuối cùng}
d^[lx,ly] := h; d^[x,y] := h;
{xác nhận vị trí mới của ô (Lx, Ly)}
lx := x + dx[k]; ly := y + dy[k];
h := tr^[lx,ly]; {chuẩn bị cho tìm cạnh tiếp theo}
end;
end;

Procedure findpath(x,y : integer); {Tìm đường mở}
var i,j,u,v,k,dau, cuoi : integer;
begin
  fillchar(tr^, sizeof(tr^), 0);
  fillchar(qd^, sizeof(qd^), 0);
  fillchar(qc^, sizeof(qc^), 0);
  dau := 0; {biến: đầu hàng đợi}
  cuoi := 1; {biến: cuối hàng đợi}
  { nạp ô (x,y) vào hàng đợi}
  qd^[1] := x; qc^[1] := y;
  while dau < cuoi do begin
    inc(dau); {Lấy ô (i,j) ở đầu hàng đợi}
    i:=qd^[dau];
    j:=qc^[dau];
    {xét các ô (u,v) mã nhảy từ (i,j) tới theo hướng k}
    for k:=1 to 8 do begin
      u := i + dx[k]; v := j + dy[k];
      if (tr^[u,v] = 0) then {ô (u,v) chưa thăm}
      if (a^[u,v]=0) then begin {ô (u,v) là ô trống}
        tr^[u,v] := k; {xác nhận theo hướng k, từ ô (i,j) mã tới ô (u,v)}
        if d^[u,v]=0 then begin {ô (u,v) là đỉnh nhạt bên B thì}
          tangcanh(u,v); {đổi màu các cạnh trên đường mở}
          exit; {thoát tìm kiếm đường mở}
        end
      else begin {ô (u,v) là đỉnh đậm thì nạp ô bên A ghép với nó vào h đợi}
        inc(cuoi);
        qd^[cuoi] := u - dx[d^[u,v]];
        qc^[cuoi] := v - dy[d^[u,v]];
      end
    end
  end
end;

```

```

        end;
    end;
end;
end;
end;
Procedure bo_ghep_max; {Thuật toán tìm bộ ghép cực đại trên đồ thị hai phía}
var i,j,x,y : integer;
begin
    for i:=1 to n do
        for j:=1 to n do
            if (d^[i,j]=0) then {xét các ô(i,j) là đỉnh nhạt}
            if (a^[i,j]=1) then {ô (i,j) bên A (là tập các ô đen)}
                findpath(i,j); {tìm đường mở}
            end;
Procedure ghi_ketqua;
begin
    assign(g,fo); rewrite(g);
    sl:=n*n-sl-m; {Theo định lý Hung ga ri: Lực lượng tập độc lập cực đại bằng số
đỉnh của đồ thị hai phía trừ đi lực lượng bộ ghép cực đại}
    writeln(g,sl);
    close(g);
end;
Procedure ghep_bandau; {Thực hiện ghép ban đầu}
var i,j, u,v ,h : integer;
begin
    sl := 0;
    for i:=1 to n do
        for j:=1 to n do
            if d^[i,j]=0 then {(i,j) là đỉnh nhạt}
            if a^[i,j]=1 then {(i,j) là ô bên A}
                for h:=1 to 8 do begin {xét các ô trắng bên B mà mã từ (i,j) có thể nhảy tới}
                    u := i + dx[h]; v := j + dy[h];
                    if d^[u,v]=0 then {(u,v) cũng là đỉnh nhạt – là đỉnh chưa ghép}
                    if a^[u,v]=0 then begin {(u,v) là ô trắng thì ghép với (i,j)}
                        d^[i,j] := h; d^[u,v] := h; {xác nhận ghép}
                        inc(sl); {tăng số cạnh bộ ghép}
                        break;

```



```

end;
end;
end;
BEGIN
    capphat;
    doc_dulieu;
    ghép_bandau;
    bo_ghép_max;
    ghi_ketqua;
END.

```

Bài 52. Dự đoán

Trong một cuộc thi vui dự đoán về giải điền kinh có N vận động viên tham gia, Ban tổ chức đã thu được M câu trả lời. Mỗi câu trả lời có 2 loại:

“Vận động viên i không về đích thứ j ”. Câu trả lời loại này được đánh số là loại 0

“Vận động viên i về đích thứ j ”. Câu trả lời loại này được đánh số là 1.

Hỏi thứ tự về đích của các vận động viên như thế nào thì số câu trả lời đúng là nhiều nhất.

Dữ liệu vào từ file văn bản DUDOAN.INP gồm: Dòng đầu là hai số N và M ($1 < N \leq 200$, $1 \leq M \leq 32767$). M dòng tiếp theo mỗi dòng là một câu trả lời gồm 3 số X, Y, C . X . Nếu $C=0$ thì X không về thứ Y , nếu $C=1$ thì X về thứ Y .

Kết quả ghi ra file văn bản DUDOAN.OUT ghi N số là thứ tự về đích của các vận động viên theo thứ tự tính từ vận động viên 1 đến N

Ví dụ

DUDOAN . INP	DUDOAN . OUT
2 3 1 1 1 2 2 0 1 2 0	1 2

Gợi ý. Đây là bài toán có thể giải bằng thuật toán tìm bộ ghép có tổng trọng số trên các cạnh ghép đạt lớn nhất trên đồ thị hai phía. Phía X gồm các đỉnh là N vận động viên số hiệu từ 1 đến N , phía Y gồm N đỉnh là thứ tự về

đích từ 1 đến N. Cần xây dựng trọng số cho cạnh (i,j) với $i \in X$ và $j \in Y$ MÀ $c[i,j]$ là số hằng định vận động viên i về đích thứ j . Khi có câu trả lời “ $i \rightarrow j$ 1” thì tăng $c[i,j]$ một đơn vị, Khi gặp câu hỏi “ $i \rightarrow j$ 0” thì tăng $c[i,j_0]$ lên một đơn vị ($\forall j_0 \neq j$) vì vận động viên i không về đích thứ j thì có thể về các đích khác j .

Chương trình.

```
const fi      = 'dudoan.in';
      fo      = 'dudoan.out';
      max     = 201;
type  int     = integer;
      arr1    = array[0..max] of int;
      arr2    = array[0..max] of ^arr1;
var    n,m    : int;
      c      : arr2;
      a,b,t  : arr1;
      dd     : arr1;
      fx,fy  : arr1;
      dx,dy  : arr1;
      ok     : boolean;
      lj     : int;
Procedure   chuanbi;
var  i,j     :int;
begin
  for i:=1 to n do begin
    new(c[i]);
    fillchar(c[i]^, sizeof(c[i]^), 0);
  end;
  fillchar(fx, sizeof(fx), 0);
  fillchar(fy, sizeof(fy), 0);
end;
Procedure   input;
var  f      :text;
      i,x,y,z,y0,j,max :int;
begin
  assign(f,fi); reset(f);
  readln(f,n,m);
  chuanbi;
  for i:=1 to m do begin
    readln(f,x,y,z);
    case z of
      0: begin
          for y0:=1 to n do
            if y0<>y then inc(c[x]^[y0]);
```

```

        end;
        1: inc(c[x]^[y]);
    end;
end;
for i:=1 to n do begin
    max:=-maxint;
    for j:=1 to n do
        if c[i]^[j]>max then max:=c[i]^[j];
        fx[i]:=max;
    end;
    close(f);
end;
Procedure    init1;
begin
    fillchar(a,sizeof(a),0);
    fillchar(b,sizeof(b),0);
end;
Procedure    init2;
begin
    fillchar(dx,sizeof(dx),0);
    fillchar(dy,sizeof(dy),0);
    fillchar(t,sizeof(t),0);
    ok:=false;
end;
Procedure tim_duongmo(i:int);
var    j        :int;
begin
    if ok then exit;
    dx[i]:=1;
    for j:=1 to n do
        if (dy[j]=0) then
            if (fx[i]+fy[j]=c[i]^[j]) then begin
                t[j]:=i;
                if b[j]=0 then begin
                    lj:=j;
                    ok:=true;
                    exit;
                end
            else begin dy[j]:=1; tim_duongmo(b[j]); end;
        end;
    end;
end;
Procedure tang_canh(j:int);
var    i,p : int;
begin
    repeat
        i:=t[j];

```

```

    p:=a[i];
    a[i]:=j;
    b[j]:=i;
    j:=p;
until (p=0);
end;
function    timmin:int;
var  i,j,min    :int;
begin
    min:=maxint;
    for i:=1 to n do
        if dx[i]=1 then
            for j:=1 to n do
                if dy[j]=0 then
                    if fx[i]+fy[j]-c[i]^j<min then
                        min:=fx[i]+fy[j]-c[i]^j;
                        timmin:=min;
end;
Procedure    suanhan;
var  i,j,min    :int;
begin
    min:=timmin;
    for i:=1 to n do
        if dx[i]=1 then dec(fx[i],min);
        for j:=1 to n do
            if dy[j]=1 then inc(fy[j],min);
end;
Procedure    xuly;
var  i          :int;
begin
    init1;
    for i:=1 to n do
        if a[i]=0 then
            repeat
                init2;
                tim_duongmo(i);
                if not ok then suanhan
                else tang_canh(1j);
            until ok;
end;
Procedure output;
var  g          : text; i,j : int;
begin
    assign(g,fo);rewrite(g);
    for i:=1 to n do write(g,a[i], ' ');
    close(g);

```

```

for i:=1 to n do
    dispose(c[i]);
end;
BEGIN
    input;
    xuly;
    output;
END.

```

Bài 53. Tiếp thị

Cho N thành phố. Khoảng cách giữa 2 thành phố là $C[i,j]$. Có K nhân viên tiếp thị hiện đang ở tại K thành phố trong N thành phố nói trên. Hãy chuyển K nhân viên tiếp thị này đến K thành phố mới trong N thành phố này sao cho tổng khoảng cách di chuyển là nhỏ nhất (mỗi thành phố mới có 1 nhân viên tiếp thị tới).

Dữ liệu vào từ file văn bản TIEP THI.INP tổ chức như sau: Dòng đầu là 2 số N và K . N dòng tiếp theo cho ma trận khoảng cách C : số ở dòng i , cột j là độ dài đường đi một chiều từ thành phố i sang thành phố j . Hai dòng tiếp theo: Dòng đầu gồm K số hiệu của các thành phố cũ tương ứng với nhân viên tiếp thị 1, 2..., K đang ở. Dòng thứ hai gồm K số hiệu của các thành phố mới mà các nhân viên tiếp thị sẽ đến.

Kết quả ghi ra file văn bản TIEP THI.OUT theo yêu cầu sau: Dòng đầu là tổng độ dài quãng đường di chuyển của K nhân viên tiếp thị. K dòng tiếp theo: mỗi dòng là hành trình của từng nhân viên gồm các số hiệu được liệt kê liên tiếp theo thứ tự hành trình, bắt đầu hành trình là số hiệu thành phố cũ, kết thúc hành trình là thành phố mới.

Ví dụ :

TIEP THI . INP	TIEP THI . OUT
10 4	5
0 7 7 1 2 1 1 5 1 3	1 7
2 0 1 1 1 1 5 4 1 7	2 9
1 1 0 1 1 1 3 7 2 4	3 4 10
5 2 4 0 2 4 10 1 7 1	4 8
7 1 3 7 0 10 2 4 1 1	
10 1 1 2 1 0 1 4 2 1	
1 1 4 1 1 3 0 1 10 1	
7 1 7 1 1 3 4 0 1 1	

7	7	1	2	1	1	4	2	0	10
1	3	4	1	2	4	1	1	1	0
1	2	3	4						
10	9	8	7						

Gợi ý. Trước hết, có thể dùng Floyd để tính chi phí ít nhất từ một thành phố cũ của tiếp thị viên đến một trong các thành phố mới của tiếp thị viên. Tổ chức đồ thị hai phía $G(X,Y,E)$: các đỉnh phía X là k thành phố cũ do mảng A và FX quản lý, các đỉnh phía Y là k thành phố mới do mảng B và FY quản lý. Trọng số của cạnh nối một đỉnh i của phía A với một đỉnh j của phía B là chi phí ít nhất di chuyển từ i tới j. Bài toán trở thành tìm bộ ghép có tổng trọng số trên các cạnh của bộ ghép là *nhỏ nhất*.

Chương trình.

```
uses      crt;
const    max      = 100;
         fi       = 'tiepthi.in';
         fo       = 'tiepthi.out';
type     mang1 = array[1..max,1..max] of integer;
         mang2 = array[1..max] of integer;
         mang3 = array[1..2*max] of integer;
var       tr      : mang3;
         n,k,lj   : integer;
         ok       : boolean;
         a,c,t    : mang1;
         fx,fy,old,new,dx,dy,r,l : mang2;
Procedure doc_dulieu;
var       f      : text;
         i,j     : integer;
begin
  assign(f,fi);
  reset(f);
  readln(f,n,k);
  for i:=1 to n do begin
    for j:=1 to n do begin
      read(f,c[i,j]);
      if c[i,j]=0 then c[i,j] := 10000;
    end;
    readln(f);
  end;
  for i:=1 to n do c[i,i]:=0;
  for i:=1 to k do read(f,old[i]); {k thành phố đang ở}
  readln(f);
  for i:=1 to k do read(f,new[i]); {k thành phố cần di chuyển tới}
```

```

    close(f);
end;
Procedure tinh_cp; {Thuật toán Floyd tính chi phí nhỏ nhất giữa hai thành phố}
var i,j,m : integer;
begin
    fillchar(t,sizeof(t),0);
    for m:=1 to n do
        for i:=1 to n do
            for j:=1 to n do
                if c[i,m]+ c[m,j]< c[i,j] then begin
                    c[i,j] := c[i,m]+ c[m,j];
                    t[i,j] := m;
                end;
            end;
        end;
    end;
Procedure tao_bandau;
var i,j,p : integer;
begin
    for i:=1 to k do begin
        p := -maxint;
        for j:=1 to k do begin
            a[i,j] := -c[old[i],new[j]];
            if a[i,j] > p then p := a[i,j];
        end;
        fx[i] := p; {Tạo nhãn ban đầu cho các đỉnh bên X}
    end;
    fillchar(fy,sizeof(fy),0); {Tạo nhãn ban đầu cho các đỉnh bên Y}
end;
Procedure tang_canh(j : integer); {Tăng thêm cạnh đậm trên đường mở}
var i,p : integer;
begin
    repeat
        i := tr[j];
        p := r[i];
        r[i] := j;
        l[j] := i;
        j := p;
    until j=0;
end;
Procedure tim_duongmo(i : integer); {đường mở bắt đầu từ đỉnh i của X}
var j : integer;
begin
    if ok then exit;
    dx[i] := 1; {đánh dấu đã thăm i, đồng thời xác nhận i thuộc đường mở}
    for j:= 1 to k do {xét các đỉnh j bên Y}
        if dy[j]=0 then {j là đỉnh chưa thăm, chưa thuộc đường mở}
            if fx[i]+fy[j]=a[i,j] then begin {điều kiện ghép i và j}

```

```

tr[j] := i; {xác nhận vết trên đường mở: trước j là i}
if l[j]=0 then begin {j là đỉnh nhạt}
    lj := j; {lưu lại đỉnh cuối của đường ở}
    ok := true; {báo đã tìm xong đường mở}
    exit;
end
else begin {j là đỉnh đậm}
    dy[j] := 1; {đánh dấu đã thăm j, xác nhận j thuộc đường mở}
    tìm_duongmo(l[j]); {tìm tiếp, kéo dài đường mở}
end;
end;
end;
function min : integer; {tìm lượng sửa nhãn}
var i,j,ph : integer;
begin
    ph := maxint;
    for i:=1 to k do {xét các đỉnh i bên X đã thuộc đường mở}
        if dx[i]=1 then
            for j:=1 to k do {xét các đỉnh j bên Y chưa thuộc đường mở}
                if dy[j]=0 then
                    if fx[i]+fy[j]-a[i,j]<ph then {tìm Min của (fx[i]+fy[j]-a[i,j])}
                        ph := fx[i]+fy[j]-a[i,j];
                    min := ph;
            end;
        Procedure sua_nhan; {Thực hiện sửa nhãn}
        var i, d : integer;
        begin
            d := min;
            for i:=1 to k do {nhãn các đỉnh i bên X thuộc đường mở bị giảm một lượng d}
                if dx[i]=1 then dec(fx[i],d);
            for i:=1 to k do {nhãn các đỉnh j bên Y thuộc đường mở tăng một lượng d}
                if dy[i]=1 then inc(fy[i],d);
            end;
        Procedure thuc_hien; {Thuật toán tìm bộ ghép có tổng trọng số các cạnh nhỏ nhất}
        var i : integer;
        begin
            fillchar(l,sizeof(l),0);
            fillchar(r,sizeof(r),0);
            for i:=1 to k do {tìm các đỉnh i là đỉnh nhạt bên X}
                if r[i]=0 then
                    repeat
                        ok := false;
                        fillchar(dx,sizeof(dx),0);
                        fillchar(dy,sizeof(dy),0);
                        fillchar(tr,sizeof(tr),0);

```



```

    tim_duongmo(i) ; {tìm đường mở từ đỉnh i}
    if not ok then sua_nhan {nếu không tìm được đường mở thì sửa nhãn}
        else tang_canh(lj) ; {còn không thì tăng cạnh của bộ ghép}
    until ok;
end;
Procedure ghi_ketqua;
var    f          : text;
       i,j,dem,tong : integer;
       ng         : mang2;
Procedure find(i,j : integer);
{Dựa vào kết quả của Floyd, dùng đệ quy tìm hành trình từ thành phố i đến thành phố j}
var k : integer;
begin
    k := t[i,j];
    if k=0 then begin
        if (dem=0) then
            inc(dem); ng[dem] := i;
        end;
        inc(dem); ng[dem] := j;
    end
    else begin
        find(i, k); find(k, j);
    end;
end;
begin
    tong := 0;
    for i:=1 to k do tong := tong + a[i,r[i]];
    assign(f,fo); rewrite(f);
    writeln(f,-tong); {ghi tổng chi phí nhỏ nhất}
    for i:=1 to k do begin
        dem:=0;
        find(old[i],new[r[i]]); {tìm hành trình của nhân viên i}
        for j:=1 to dem do write(f,ng[j], ' '); {ghi hành trình}
        writeln(f);
    end;
close(f);
end;
BEGIN
    doc_dulieu;
    tinh_cp;
    tao_bandau;
    thuc_hien;
    ghi_ketqua;
END.

```

Bài 54. Phỏng vấn

Ngày hội phỏng vấn việc làm được tổ chức để người xin việc (các ứng viên) và nơi cần người (các công ty) có dịp trực tiếp gặp gỡ nhau.

Giả sử có N công ty (đánh số từ 1 đến N) và M ứng viên (đánh số từ 1 đến M) tham gia. Tập hợp các yêu cầu tuyển người được mã hoá bằng bảng chữ cái in tiếng Anh (như vậy có không quá 26 yêu cầu). Mỗi công ty và mỗi ứng viên tham gia, đều khai báo cho Ban tổ chức biết các yêu cầu (đối với các công ty) và các khả năng (đối với các ứng viên) của mình. Căn cứ vào những thông tin này, Ban tổ chức cần sắp xếp các ứng viên vào các nơi tương ứng để phỏng vấn. Mỗi ứng viên chỉ được xếp phỏng vấn ở một công ty tại đó mình đáp ứng được tất cả các yêu cầu của công ty.

Thời gian bắt đầu phỏng vấn từ 7 giờ và kéo dài đến không quá 17 giờ trong ngày. Mỗi công ty được bố trí riêng một bàn và phỏng vấn liên tục (người này ra, người kia vào). Thời gian phỏng vấn mỗi người (đối với mọi công ty) là 10 phút.

Bạn hãy viết một chương trình giúp Ban tổ chức tìm phương án sao cho có tối đa các ứng viên được phỏng vấn với thời gian hoàn thành sớm nhất.

Dữ liệu vào cho trong file văn bản có tên là `PV.IN`, gồm: Dòng đầu ghi số N (số công ty). Dòng thứ i trong số N dòng kế tiếp ghi thông tin yêu cầu của công ty thứ i gồm các chữ cái in viết liền nhau mô tả các yêu cầu tuyển người của công ty. Dòng tiếp ghi số M (số ứng viên). Dòng thứ i trong số M dòng kế tiếp ghi khả năng thực hiện các yêu cầu của ứng viên thứ i bao gồm các chữ cái in viết liền nhau.

Giả thiết dữ liệu vào là đúng đắn, không cần kiểm tra.

Kết quả cần ghi ra file văn bản có tên là `PV.OUT`, gồm: Dòng đầu ghi số K là số công ty có ứng viên đến phỏng vấn. K dòng tiếp, mỗi dòng ghi lịch phỏng vấn của một công ty, bắt đầu là số hiệu công ty, kế đó là số ứng viên được phỏng vấn tại công ty và cuối cùng là các số hiệu của các ứng viên này. Các số trên cùng một dòng ghi cách nhau ít nhất một dấu trắng.

Hạn chế kích thước: $M \leq 1500$, $N \leq 30$.

Thí dụ

PV . IN
3
DBA
CB
CBD
5
DBAC
BADC
DCAB
CBDA
ABC

PV . OUT
3
1 2 1 2
2 2 4 5
3 1 3

Chương trình

```

const      maxn      = 30;
           maxm      = 1500;
           limit     = 60; //số cuộc phỏng vấn tối đa của 1 công ty
           fi        = 'pv.in';
           fo        = 'pv.out';

type       m1        = array[1..maxn,1..maxm] of byte;
           m2        = array[1..maxn] of string;
           m3        = array[1..maxm] of integer;
           m4        = array[1..maxm] of byte;
           mleft     = array[0..maxn] of integer;
           mright    = array[1..maxm] of byte;

var        f,g       : text;
           a         : m1;
           bac       : m3;
           tr,dx     : m4;
           dem,left  : mleft;
           th,right  : mright;
           lj,count,m,n: integer;
           ok        : boolean;

procedure open_file;
begin
    assign(f,fi); reset(f);
    assign(g,fo); rewrite(g);
end;

procedure close_file;
begin
    close(f);  close(g);
end;

procedure read_input;
var    i,j    : integer;  x      : string;          cty      : m2;
procedure rutgon(var x :string);

```

```

begin
    while x[1]=' ' do delete(x,1,1);
    while x[length(x)]=' ' do delete(x,length(x),1);
end;
function thoaman(x1,x2 :string):boolean;
// Kiểm tra các khả năng x2 đã đáp ứng được yêu cầu x1 hay chưa
var i : byte;
begin
    thoaman:=false;
    for i:=1 to length(x1) do
        if pos(x1[i],x2)=0 then exit;
        thoaman:=true;
    end;
begin
    readln(f,n);
    for i:=1 to n do begin
        readln(f,x); rutgon(x);
        cty[i]:=x; // Yêu cầu đòi đáp ứng của công ty i
    end;
    fillchar(bac,sizeof(bac),0);
    readln(f,m);
    for j:=1 to m do
        begin
            readln(f,x); rutgon(x); // khả năng của ứng cử viên j
            for i:=1 to n do
                if thoaman(cty[i],x) then // ứng cử viên j đáp ứng yêu cầu công ty i
                begin
                    a[i,j]:=1; // xác nhận công ty i có thể cho j được phỏng vấn
                    inc(bac[j]); // số công ty mà j có thể tham gia phỏng vấn
                end
                else a[i,j]:=0; // i không được phỏng vấn tại công ty j
            end;
        end;
end;
procedure find(i: byte); // tìm đường tăng cặp ghép
// chọn ứng cử viên j đến phỏng vấn tại công ty i
var j : integer;
begin
    for j:=1 to m do begin
        if (dx[j]=0) and (th[j]=0) and (a[i,j]>0) then begin
            dx[j]:=1;
            tr[j]:=i; // lưu vết trước j là i trên đường tăng cặp ghép
            if right[j]=0 then begin // Kết thúc đường tăng cặp ghép
                ok:=true;
                lj:=j; // lưu đỉnh cuối cùng của đường tăng cặp ghép
            end;
            exit;
        end;
    end;
end;

```

```

        end else    find (right[j]); // tìm tiếp
    end;
end;
end;
procedure add_edge; // đổi màu cạnh trên đường tăng cặp ghép để tăng cạnh
var    i,j,p : integer;
begin
    inc(count); // tăng thêm một cạnh
    j:=lj;
    repeat
        i:=tr[j];
        p:=left[i];
        left[i]:=j;
        right[j]:=i;
        j:=p;
    until    j=0;
end;
procedure    match; // thuật toán tìm cặp ghép cực đại
var    i        :byte;
begin
    count:=0; // lực lượng của cặp ghép
    fillchar(left,sizeof(left),0); // các đỉnh bên trái (người) đều nhạt
    fillchar(right,sizeof(right),0); // các đỉnh bên phải (công ty) đều nhạt
    for i:=1 to n do
        if (dem[i]<limit) then // công ty i chưa đủ 60 cuộc phỏng vấn
            if (left[i]=0) then begin // đỉnh i là đỉnh nhạt
                ok:=false;
                fillchar(dx,sizeof(dx),0);
                fillchar(tr,sizeof(tr),0);
                find(i); // tìm đường tăng cặp ghép từ i
                if ok then add_edge; // tìm được đường tăng cặp ghép thì tăng cạnh
            end;
        end;
    end;
procedure    update;
var    j: integer;
begin
    for j:=1 to m do
        if th[j]=0 then begin // nếu j chưa tham gia phỏng vấn tại công ty nào
            th[j]:=right[j]; // thì tham gia tại công ty right[j]
            inc(dem[ th[j] ]); // tăng thêm số lượng người đến công ty này
        end;
    end;
end;
procedure    process;
begin
    fillchar(th,sizeof(th),0);

```

```

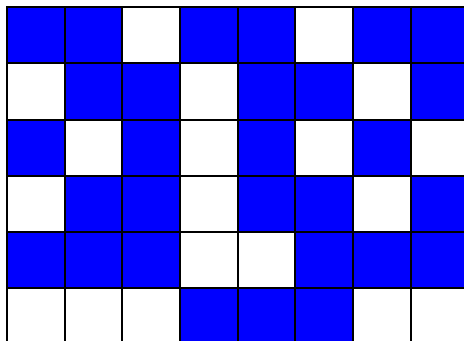
fillchar(dem,sizeof(dem),0);
count:=1;
while count>0 do begin
    match; // tìm cặp ghép
    if count>0 then update; // tìm được thì cập nhật thêm kết quả ghép này
end;
end;
procedure write_out;
var i,j,sol : integer;
begin
    sol:=0;
    for i:=1 to n do
        if dem[i]>0 then inc(sol); // số công ty có người đến tham gia phỏng vấn
        writeln(g,sol);
        for i:=1 to n do
            if dem[i]>0 then begin // công ty i có người đến tham gia phỏng vấn
                write(g,i,' ',dem[i]); // số hiệu công ty, số người đến được phỏng vấn
                for j:=1 to m do
                    if th[j]=i then write(g,' ',j); // số hiệu người được phỏng vấn
                writeln(g);
            end;
        end;
    BEGIN
        open_file;
        read_input;
        process;
        write_out;
        close_file;
    END.

```

Bài 55. Hai ô đen (Đề thi HSG năm 1995)

Cho một hình chữ nhật kích thước $m \times n$ ô vuông, trong đó có một số ô tô màu đen (gọi là ô đen), các ô còn lại tô màu trắng (gọi là ô trắng). Giả thiết rằng mỗi dòng có ít nhất hai ô đen và $m, n < 100$.

Yêu cầu : Cần phải chọn ra $2 \times m$ ô đen của lưới sao cho :



Trong mỗi dòng của lưới có đúng hai ô đen được chọn;

Số ô được chọn trong cột có nhiều ô được chọn nhất là nhỏ nhất.

Dữ liệu vào được cho trong file văn bản là BL1_95.INP, trong đó dòng đầu tiên ghi các giá trị m, n (cách nhau bởi dấu cách); m dòng tiếp theo, mỗi dòng ghi thông tin về màu các ô trong một dòng của lưới dưới dạng một xâu nhị phân độ dài n (các kí tự 0,1 viết liền nhau), với quy ước kí tự 1 ghi nhận màu đen, kí tự 0 ghi nhận màu trắng.

Ví dụ thông tin về màu của các ô trong lưới 6×8 cho trong hình bên được mô tả trong file văn bản có tên BL1_95.INP có nội dung như sau :

```
6 8
11011011
01101101
10101010
01101101
11100111
00011100
```

Kết quả đưa ra file văn bản có tên là BL1_95.OUT , trong đó dòng đầu tiên ghi số ô được chọn trong cột có nhiều ô được chọn nhất, m dòng tiếp theo, mỗi dòng thứ i ghi một xâu độ dài n gồm các ký tự 0, 1 viết liền nhau với quy ước ký tự 1 ghi nhận vị trí ô được chọn, kí tự 0 ghi nhận vị trí ô không được chọn ($i=1, 2, \dots, m$)

Chẳng hạn file BL1_95.OUT có thể có nội dung sau :

```
2
10000001
01000001
00101000
01000100
00100100
00011000
```

Gợi ý.

Đây là bài toán tối ưu được giải quyết bằng tìm luồng cực đại.

Trước hết xây dựng mạng : Các đỉnh từ 1 đến m thể hiện số hiệu các dòng, các đỉnh từ $m+1$ đến $m+n$ thể hiện số hiệu các cột. Thêm đỉnh phát là s có số hiệu là $m+n+2$, đỉnh thu là t có số hiệu là $m+n+1$. Trọng số trên các cung như sau : Các cung (s, i) với $i=1, 2, \dots, m$ có trọng số bằng 2 (do mỗi dòng chọn đúng 2 ô đen). Nếu trên hình chữ nhật ô (i, j) thuộc dòng i cột j là ô đen thì trên mạng cho cung $(i, j+m)$ trọng số 1. Cuối cùng, các cung (i, t) với $i=m+1, m+2, m+n$ thì cho trọng số là d (tăng dần từ 1). Trọng số d của các cung này thể hiện số ô đen của cột có nhiều ô đen nhất.

Với mỗi giá trị d , thực hiện tìm luồng cực đại trên mạng vừa xây dựng. Nếu tìm được luồng, cần kiểm tra lại giá trị luồng có đúng bằng $2*m$ hay không. Nếu đủ, tiếp tục tăng d . Quá trình dừng khi luồng không thỏa mãn có giá trị bằng $2*m$

Chương trình

```
const fi = 'b11_95.inp';
      fo = 'b11_95.out';
type arr = array[0..202,0..202] of byte;
      bg = record toi,delta : integer;
           end;

var   a,f      : ^arr;
      stack,dx : array[0..202] of byte;
      nh       : array[0..202] of bg;
      m,n,sn,s,top,t,d : integer;
```

Procedure doc_dulieu; {Đọc dữ liệu và tạo mạng}

```
var i,j,sl : integer;
    ch      : char;
    f       : text;

begin
  assign(f,fi); reset(f);
  readln(f,m,n);
  sn:=m+n+1;
  s:=sn+1; {đỉnh phát s=m+n+2}
  t:=sn;   {đỉnh thu t=m+n+1}
  for i:=1 to m do begin
    sl:=0; {số lượng ô đen trên dòng i}
    for j:=1 to n do begin
```



```

        read(f,ch);
        if ch='1' then a^[i,m+j]:=1;
    end;
    readln(f);
    a^[s,i]:=2; {trọng số trên cung nối đỉnh phát với các đỉnh là số hiệu dòng}
end;
close(f);
end;
Procedure capphat;
begin
    new(a); fillchar(a^,sizeof(a^),0);
    new(f); fillchar(f^,sizeof(f^),0);
end;
Procedure nap(p:integer);
begin
    inc(top); stack[top]:=p;
end;
function lay:integer;
begin
    lay:=stack[top]; dec(top);
end;
function min(a,b:integer):integer;
begin
    if a<b then min:=a else min:=b;
end;
Procedure init;
begin
    fillchar(dx,sizeof(dx),0);
    top:=1;
    stack[top]:=s;
    nh[s].delta:=maxint;
    nh[s].toi:=s;
end;
Procedure suanhan(j:integer);
var i:integer;
begin
    for i:=1 to sn+1 do
        if dx[i]=0 then begin {đỉnh i chưa đánh dấu đã xét}

```

```

    if (a^[j,i]<>0) and (f^[j,i]<a^[j,i]) then begin {cung (j,i)
thuận}
        dx[i]:=1; {đánh dấu đã xét đỉnh i}
        nh[i].toi:=+j;
        nh[i].delta := min(nh[j].delta,a^[j,i]-f^[j,i]);
        nap(i); {nap i vào đường tăng luồng}
    end else
    if (a^[i,j]<>0) and (f^[i,j]>0) then begin {cung (i,j) nghịch}
        dx[i]:=1;
        nh[i].toi:=-j;
        nh[i].delta := min(nh[j].delta,f^[i,j]);
        nap(i);
    end
end;
end;
Procedure tangluong;
var z,x : integer;
begin
    z:=t; {lần ngược theo đường tăng luồng, bắt đầu từ đỉnh thu t}
    repeat
        x := z;
        z := nh[x].toi;
        if z>0 then inc(f^[z,x],nh[t].delta) {tăng luồng trên cung thuận}
        else f^[x,-z]:=f^[x,-z]-nh[t].delta; {giảm luồng trên cung n}
        z:=abs(z);
    until z=s;
end;
Procedure timluong;
var v : integer;
begin
    repeat
        init;
        while top>0 do begin
            v:=lay; {lấy đỉnh v tại đỉnh của stack}
            suanhan(v); {sửa nhãn cho các đỉnh kề với đỉnh v}
            if dx[t]<>0 then begin {có đường tăng luồng}
                tangluong; {tăng luồng}
            end
        end
    until top=0;
end;

```

```

        break;
    end;
end;
until dx[t]=0; {không còn đường tăng luồng nào nữa}
end;
Procedure gan(k : integer);
var i : integer;
begin
    for i:=m+1 to m+n do a^[i,sn]:=k;
end;
Procedure solve;
var i : integer;
    ok : boolean;
begin
    d:=0; {d : trọng số cung nối các đỉnh là số hiệu cột với đỉnh thu. Số ô đen ở cột có
    nhiều ô đen nhất không thể vượt qua trọng số d }
    repeat
        d := d+1; {tăng dần trọng số d}
        gan(d); {gán lại trọng số cho các cung nối các đỉnh là số hiệu cột với đỉnh thu}
        timluong;
        ok:=true;
        for i:=1 to m do {kiểm tra xem mỗi dòng có đúng 2 ô đen hay chưa}
            if f^[s,i]<>2 then begin
                ok:=false;
                break;
            end;
        if ok then break; {thỏa mãn mỗi dòng có đúng 2 ô đen thì thoát ngay để bảo
        đảm số ô đen ở cột có nhiều ô đen nhất là ít nhất}
        until false;
    end;
Procedure ghi_ketqua;
var g : text;
    i,j : integer;
begin
    assign(g,fo); rewrite(g);
    writeln(g,d); {d : số ô đen ở cột có nhiều ô đen nhất}

```

```

        {Ô đen (i,j) được chọn nếu có luồng trên cung (i, m+j) }   for i:=1 to m do
begin
    for j:=1 to n do
        if f^[i,m+j]>0 then write(g,1)
        else write(g,0);
        writeln(g);
    end;
    close(g);
end;
BEGIN
    capphat;
    doc_dulieu;
    solve;
    ghi_ketqua;
END.

```

Bài 56. Kho xăng

Khu vực đặt các bể xăng của một Tổng đại lý xăng dầu có dạng một hình chữ nhật được chia thành $M \times N$ ô vuông. Các ô vuông đánh toạ độ từ trên xuống dưới, từ trái qua phải theo thứ tự hàng, cột. Tại K ô của lưới có đặt các bể xăng. Người ta cần xây dựng một hệ thống đèn pha chiếu dọc theo hàng hoặc là cột của lưới ô vuông sao cho mỗi bể chứa phải được chiếu sáng bởi ít nhất một đèn pha chiếu dọc theo hàng hoặc theo cột chứa nó. Biết:

- A_i là chi phí xây dựng đèn chiếu sáng dọc theo hàng i
- B_j là chi phí xây dựng đèn chiếu sáng dọc theo cột j

Yêu cầu: Tìm cách xây dựng hệ thống đèn chiếu với tổng chi phí xây dựng là nhỏ nhất.

Dữ liệu vào: từ file văn bản Khoxang.inp: Dòng đầu tiên chứa ba số nguyên dương M, N, K ($M, N < 100$); Dòng thứ hai chứa m số nguyên dương A_1, A_2, \dots, A_M ; Dòng thứ ba chứa n số nguyên dương: B_1, B_2, \dots, B_N ; Dòng thứ i trong K dòng còn lại chứa toạ độ của bể xăng thứ i ($i=1, 2, \dots, K$).

Kết quả ra: file văn bản Khoxang.out ghi như sau: Dòng đầu tiên ghi tổng chi phí theo cách xây dựng tìm được. Dòng thứ hai ghi hai số P và Q

theo thứ tự là số lượng đèn chiếu dọc theo hàng và cột (ghi số 0 nếu không có). P+Q dòng tiếp theo lần lượt các số hiệu của các hàng rồi đến các cột có đặt đèn chiếu, mỗi dòng ghi 1 số.

Ví dụ:

KHOXANG . INP	KHOXANG . OUT
2 3 4	11
10 5	1 2
12 4 2	2
1 2	2
1 3	3
2 1	
2 3	

Gợi ý. Đây là bài toán tìm lát cắt hẹp nhất trên mạng. Có thể giải bằng tìm luồng cực đại trên mạng.

Trước hết xây dựng mạng như sau : Các đỉnh từ 2 đến m+1 tương ứng với các dòng từ 1 đến m, các đỉnh từ m+2 đến m+n+1 tương ứng với các cột từ 1 đến n. Thêm 2 đỉnh là đỉnh s=1 và đỉnh t=m+n+2. Xây dựng các cung như sau: Các cung nối đỉnh s với đỉnh i+1 (i=1,2,..m) có trọng số tương ứng là chi phí xây dựng đèn chiếu sáng hàng i. Các cung nối đỉnh (m+1)+j (j=1,2,...n) với đỉnh t tương ứng có trọng số là chi phí xây dựng đèn chiếu sáng cột j. Nếu có đèn tại hàng u cột v thì cung nối đỉnh u+1 đến đỉnh v+m+1 có trọng số là vô cùng. Sau đó tìm luồng cực đại trên mạng này (đỉnh phát là s, đỉnh thu là t). Đến khi không tìm được đường tăng luồng đi tới đỉnh t nữa thì hình thành tập X gồm các đỉnh i không có đường tăng luồng đang xây dựng nào đi tới nó và tập Y gồm các đỉnh còn lại (các đường tăng luồng đang xây dựng đi tới được nó). Nghĩa là phân hoạch (X, Y) tạo thành lát cắt hẹp nhất.

Kết quả là: xây dựng các đèn chiếu sáng dọc theo những hàng ứng với các đỉnh thuộc tập X trên mạng không có đường tăng luồng cuối cùng đi tới và xây dựng các đèn chiếu sáng dọc theo những cột ứng với các đỉnh thuộc

tập Y trên mạng có đường tăng luồng cuối cùng đi tới. Chi phí ít nhất chính là thông lượng qua lát cắt hẹp nhất này.

Chương trình.

```
uses crt;
const max      = 101;
      fi       = 'khoxang.inp';
      fo       = 'khoxang.out';
type mang      = array[0..2*max] of integer;
      mang1    = array[0..2*max] of ^mang;
var  c,f       : mang1;
      v,
      trace,
      kq,dx,
      stack   : mang;
      n, m, k, top, t,
      n_row, n_column : integer;
function min(a,b:integer):integer;
begin
  if a<b then min:=a  else min:=b;
end;
Procedure doc_dulieu; {Đọc dữ liệu, xây dựng mạng như phần gợi ý}
var i,j,u,v : integer;
      g      : text;
begin
  assign(g,fi); reset(g);
  readln(g,m,n,k);
  t := m + n + 2;
  for i:= 0 to t do begin
    new(c[i]);
    fillchar(c[i]^,sizeof(c[i]^),0);
    new(f[i]);
  end;
  for i:=1 to m do read(g,c[1]^ [i+1]);
  readln(g);
  for j:=1 to n do read(g,c[m+1+j]^ [t]);
  readln(g);
  for i:=1 to k do begin
    readln(g,u,v);
    c[u+1]^ [v+m+1]:= maxint;
  end;
  close(g);
end;
{Bài toán luồng cực đại}
Procedure nap(p:integer);
begin
```

```

    inc(top); stack[top]:=p;
end;
function lay:integer;
begin
    lay:=stack[top]; dec(top);
end;
Procedure init;
var i : integer;
begin
    fillchar(dx,sizeof(dx),0);
    for i:=1 to t do trace[i] := 0;
    top      := 1;
    stack[top] := 1;
    trace[1]  := 1000;
    dx[1]     := 1;
    v[1]      := maxint;
end;
Procedure suanhan(j:integer);
var i:integer;
begin
    for i:=1 to t do
        if dx[i]=0 then begin
            if (f[j]^i<c[j]^i) then begin
                trace[i] := +j;
                v[i] := min(v[j],c[j]^i-f[j]^i);
                dx[i] := 1;
                nap(i);
            end;
            if (c[i]^j>0) and (f[i]^j>0) then begin
                trace[i] := -j;
                v[i] := min(v[j],f[i]^j);
                dx[i] := 1;
                nap(i);
            end;
        end;
    end;
end;
Procedure tangluong;
var i,j : integer;
begin
    j:=t;
    repeat
        i := j;
        j := trace[i];
        if j>0 then inc(f[j]^i,v[t]);
        if j<0 then dec(f[i]^[-j],v[t]);
        j:=abs(j);

```

```

    until j = 1000;
end;
Procedure timluong;
var v:integer;
begin
    for v:=1 to t do begin
        new(f[v]);
        fillchar(f[v]^, sizeof(f[v]^), 0);
    end;
    repeat
        init;
        while top>0 do begin
            v:=lay;
            suanhan(v);
            if trace[t]<>0 then begin
                tangluong;
                break;
            end;
        end;
    until trace[t]= 0;
end;
{tìm lát cắt hợp nhất, ghi kết quả vào tệp output}
Procedure ghi_ketqua;
var i,j,cost, count : integer;
    g      : text;
begin
    cost := 0; {Khởi trị chi phí xây dựng}
    n_row := 0; {số hàng chọn xây đèn pha}
    fillchar(kq, sizeof(kq), 0); {lưu các hàng, các cột được xây đèn pha}
    for i:=2 to m+1 do {Tìm các phần tử thuộc tập X (chỉ cần bên hàng)}
    if trace[i] = 0 then begin
        inc(n_row); {tăng số hàng có đèn pha}
        inc(cost, c[1]^ [i]); {thêm chi phí xây đèn cho hàng này}
        kq[n_row] := i-1; {lưu số hiệu hàng này vào kết quả}
    end;
    count := n_row; {count sẽ cho tổng số hàng và cột có đèn pha}
    n_column := 0; {Khởi trị số cột có đèn pha}
    for i:= m+2 to t-1 do {Tìm các phần tử thuộc tập Y (chỉ cần bên cột)}
    if trace[i] <> 0 then begin
        inc(n_column); {tăng số cột có đèn pha}
        inc(cost, c[i]^ [t]); {thêm chi phí xây đèn pha cho cột này}
        inc(count);
        kq[count] := i-(m+1); {lưu số hiệu cột này vào kết quả}
    end;
    assign(g, fo); rewrite(g);

```



```

writeln(g, cost); {ghi tổng chi phí}
writeln(g,n_row,' ',n_column); {ghi số hàng, số cột có đèn pha}
for i:=1 to count do
    writeln(g,kq[i]); {ghi số hiệu các hàng, cột có đèn pha}
close(g);
end;
BEGIN
    doc_dulieu;
    tim_luong;
    ghi_ketqua;
END.

```

Bài 57. Numway (Di chuyển quân tốt)

Cho một bảng hình chữ nhật được chia làm $N \times M$ ô vuông (N dòng, M cột), một con tốt sau một nước đi có thể di chuyển từ một ô ở cột này sang một ô ở cột kế tiếp. Đối với mỗi ô vuông, biết số hiệu của các ô trong cột kế tiếp theo mà con tốt có thể đến được sau một nước đi. Con tốt không được di chuyển đến ô mà nó đi qua trước đó. Thoạt đầu, con tốt được đặt tại một ô nào đó của cột thứ nhất. Sau đó con tốt di chuyển về phía cột cuối cùng. Khi con tốt đạt đến cột cuối cùng, người ta lại đặt nó vào một ô nào đó ở cột đầu tiên mà trước đó chưa hề đặt nó và lại tiếp tục thực hiện di chuyển. Trò chơi kết thúc khi không thể thực hiện nước đi tại chạng cuối cùng để con tốt có thể đến cột M . Yêu cầu xác định xem nhiều nhất có thể thực hiện bao nhiêu lần di chuyển con tốt từ cột đầu tiên đến cột cuối cùng.

Dữ liệu vào từ file văn bản NUMWAYS.INP gồm:

Dòng đầu là 2 số nguyên dương N, M ($N \leq 50, 2 \leq M \leq 10$)

Tiếp theo là $M-1$ nhóm dòng, mỗi nhóm gồm N dòng, mô tả khả năng di chuyển của con tốt từ mỗi ô của bảng. Dòng thứ i của nhóm dòng j mô tả khả năng di chuyển của con tốt từ ô ở dòng i và cột j bao gồm: số đầu tiên cho biết số khả năng di chuyển, tiếp theo là tọa độ dòng của các ô trong cột kế tiếp mà con tốt có thể di chuyển sang (các tọa độ được liệt kê theo thứ tự tăng dần).

Kết quả ghi ra file văn bản NUMWAYS.OUT số lượng đường đi tìm được

Ví dụ:

NUMWAYS . INP	NUMWAYS . OUT
4 3	3
2 1 3	
3 1 2 4	
0	
2 2 3	
1 2	
1 2	
1 3	
2 2 4	

Gợi ý.

Thuật toán : tìm luồng cực đại

Xây dựng mạng với trọng số trên các cung không vượt quá 1. Xét bàn cờ có n dòng, m cột gồm $n*m$ ô. Từ mỗi ô (i, j) thuộc dòng i , cột j của bàn cờ tạo ra 2 đỉnh của mạng : đỉnh $(i, 2*j)$ và đỉnh $(i, 2*j+1)$. Cung nối hai đỉnh này có trọng số bằng 1 để bảo đảm quân tốt không đi lại ô cũ trên bàn cờ. Khi tốt có khả năng đi từ ô (x, y) của bàn cờ đến ô $(t, y+1)$ của bàn cờ thì tạo một cung trên mạng nối đỉnh $(x, 2*y+1)$ tới đỉnh $(t, 2*y+2)$ với trọng số bằng 1. Ngoài ra, mạng còn có :

Một đỉnh phát là đỉnh $s=(1,0)$; nối đỉnh này với các đỉnh $(i,1) \mid \forall i=1..n$ bằng cung có trọng số 1.

Một đỉnh thu là đỉnh $t=(1, m+1)$, nối các đỉnh $(i, 2*m+1) \mid \forall i=1..n$, với đỉnh t bằng cung có trọng số bằng 1.

Chương trình dưới đây, dùng mảng A ba chiều để mô tả mạng: Từ mỗi ô (i, j) thuộc dòng i , cột j của bàn cờ tạo ra một cung của mạng $a[i,j]^j:=1$ thể hiện nối đỉnh $(i, 2*j)$ và đỉnh $(i, 2*j+1)$ của mạng. Khi tốt có khả năng đi từ ô (x,y) của bàn cờ đến ô $(t,y+1)$ của bàn cờ thì tạo ra cung $a[x,2*y+1]^t:=1$ (nghĩa là không cần quản lý cột kề của một cột đã biết trong mạng: cột kề của $2*y+1$ tất nhiên là $2*y+2$)

Khi thực hiện bài toán luồng cực đại chú ý rằng luồng nếu có trên các cung chỉ có thể bằng 1 do đó nhãn các đỉnh báo hiệu khả năng luồng thông

qua đỉnh chỉ có thể là 0 hoặc 1. Do đó lượng tăng luồng (nếu có thể tăng) thì chỉ có thể bằng 1.

Trong quá trình tìm kiếm đường tăng luồng, khi xét cung có luồng bằng 0 thì có thể tăng luồng nên nó là cung thuận, cung có luồng bằng 1 (không thể tăng được nữa) nên có thể tạo ra cung ngược (chỉ cần nhận biết có cung này, không cần tạo ra thực sự).

Cuối cùng, sau khi đã tìm được luồng cực đại thì giá trị luồng cực đại chính là số lần nhiều nhất có thể di chuyển tốt về cột 1.

Chương trình

```
const fi    = 'numways.in';
      fo    = 'numways.out';
      maxm  = 21;
      maxn  = 51;

type arr    = array[0..maxn] of shortint;
var  a,      {ma trận trọng số trên mạng, chú ý mỗi cột của bàn cờ được nhân thành hai
      cột: mỗi ô bàn cờ tạo thành 2 đỉnh trên mạng, chúng nối với nhau bằng cung có trọng số
      bằng 1}
      f      : array[0..maxn,0..maxm] of ^arr; // luồng
      delta,
      pre    : array[0..maxn,0..maxm] of shortint;
              // theo dõi đỉnh trước trên đường tăng luồng
      vs     : array[0..maxn,0..maxm] of boolean;
              {đánh dấu đỉnh đã thăm trong quá trình tìm đường tăng luồng}
      n,m,   // số dòng, số cột của bàn cờ
      sn,    // số cột của mạng
      sd,sc, // số hiệu dòng và cột của đỉnh phát
      td,tc  : integer; // số hiệu dòng và cột của đỉnh thu
      pathfound : boolean; // cờ báo tìm thấy đường tăng luồng
      ff      : text;

procedure read_input;
var i,j,k,l,ii,jj : integer;
begin
    // xin cấp phát vùng nhớ, khởi trị các mảng f (luồng) và a (mạng)
    for i:=0 to maxn do
    for j:=0 to maxm do begin
```

```

    new(a[i,j]);
    new(f[i,j]);
    fillchar(a[i,j]^, sizeof(a[i,j]^), 0);
    fillchar(f[i,j]^, sizeof(f[i,j]^), 0);
end;
assign(ff, fi); reset(ff);
readln(ff, n, m); // n là số dòng, m là số cột của bàn cờ
k := 1; // số hiệu cột của mạng bắt đầu là 1
for i:=1 to m-1 do begin // đọc m-1 nhóm dữ liệu ứng với m-1 cột bàn cờ
for j:=1 to n do begin // đọc một nhóm (n dòng của bàn cờ)
    a[j,k]^ [j] := 1; // trọng số cung nối đỉnh (j,k) đến đỉnh (j,k+1) trong mạng
    read(ff, l); // số lượng ô đi tới từ ô (j,k) trên bàn cờ
    for ii:=1 to l do begin // đọc các ô đi tới từ ô (j,k)
        read(ff, jj); // tọa độ dòng của ô đi tới trên cột k+1 trên bàn cờ
        a[j, k+1]^ [jj] := 1; // tạo trọng số cung nối (j,k+1) tới (jj,k+2) trên mạng
    end;
    readln(ff);
end;
k:=k+2; // tạo xong cột k và k+1 trong mạng, sang cột k+2
end;
sn:=k+1; // số hiệu cột cuối cùng
sd:=1; // số hiệu dòng của đỉnh phát
sc:=0; // số hiệu cột của đỉnh phát
td:=1; // số hiệu dòng của đỉnh thu
tc:=sn; // số hiệu cột của đỉnh thu sn=2*m+1
for i:=1 to n do begin
    a[1,0]^ [i] := 1; // tạo trọng số các cung nối đỉnh phát tới các đỉnh thuộc cột 1
    a[i, sn-1]^ [1] := 1; // trọng số các cung nối các đỉnh cột sn-1 tới đỉnh thu
end;
close(ff);
end;
function min(a,b : integer) : integer;
begin
    if a<b then min:=a else min:=b;
end;
procedure findpath; // tìm đường tăng luồng

```

```

var i,j,dq,cq,x,y : integer;
    qd,qc          : array[0..1000] of integer;
begin
    pathfound := true;
    fillchar(vs,sizeof(vs),false);
    fillchar(delta,sizeof(delta),0);
    fillchar(pre,sizeof(pre),0);
    dq := 1; // biến đầu hàng đợi
    cq := 1; // biến cuối hàng đợi
    qd[1] := sd; // nạp số hiệu dòng của đỉnh phát vào đầu hàng đợi
    qc[1] := sc; // nạp số hiệu cột của đỉnh phát vào đầu hàng đợi
    delta[sd,sc] := 1; // gán nhãn cho đỉnh phát
    pre[sd,sc] := sd; // lưu vết: đỉnh trước của đỉnh phát là chính nó
    vs[sd,sc] := true; // đánh dấu đã thăm đỉnh phát
    while dq <= cq do begin
        // lấy phần tử ở đầu hàng đợi: dòng là x, cột là y
        x := qd[dq]; y := qc[dq];
        inc(dq); // tăng giá trị biến đầu hàng đợi
        // tạo các cung thuận trên đường tăng luồng
        if y < tc then // đường tăng luồng chưa tới điểm thu
        for j := 1 to n do begin // xét các đỉnh ở cột kề với cột y
            if (not vs[j,y+1]) // đỉnh chưa thăm
            and (a[x,y]^ [j]=1) // có cung nối đỉnh (x,y) tới đỉnh (j,y+1)
            and (f[x,y]^ [j]=0) then begin // có khả năng tăng luồng trên cung này
                pre[j,y+1] := x; // xác nhận x là dòng của đỉnh trước đỉnh (j,y+1)
                vs[j,y+1] := true; // đánh dấu đã thăm đỉnh (j,y+1)
                // sửa lại nhãn của đỉnh (j,y+1)
                if delta[x,y]=0 then delta[j,y+1] := 0
                else delta[j,y+1] := 1;
                inc(cq); // nạp (j,y+1) vào cuối hàng đợi
                qd[cq] := j;
                qc[cq] := y+1;
                if (y+1=tc) and (j=1) then // đến được đỉnh thu
                    exit; // thoát để tăng luồng
            end;
        end;
    end;
end;

```

```

// tạo các cung ngược
if y>0 then // chưa ngược về tới đỉnh phát thì
for j:=1 to n do begin//xét các đỉnh thuộc cột kề bên trái
if (not vs[j,y-1]) //đỉnh (j,y-1) chưa thăm
and (a[j,y-1]^ [x]=1) //có cung nối từ đỉnh (j,y-1) tới đỉnh (x,y)
and (f[j,y-1]^ [x]=1) then begin // có luồng trên cung thuận
    pre[j,y-1] := -x; // đánh dấu đỉnh trước trên cung ngược
    vs[j,y-1] := true; //đánh dấu đã thăm (j,y-1)
    //sửa nhãn đỉnh (j,y-1)
    if delta[x,y]=0 then delta[j,y-1]:=0
    else delta[j,y-1]:=1;
    //nhập (j,y-1) vào hàng đợi
    inc(cq);
    qd[cq] := j;
    qc[cq] := y-1;
    end;
end;
end;
pathfound:=false;
end;
procedure tangluong; //tăng luồng
var x,y,z,tang : integer;
begin
    tang := delta[td,tc]; //lượng tấn luồng là nhãn của đỉnh thu
    if tang>0 then begin //tăng được
        //bắt đầu từ đỉnh thu, theo đường tăng luồng lần ngược về đỉnh phát
        x:=td; y:=tc;
        repeat
            z:=pre[x,y]; //cho dòng của đỉnh trước
            if z>0 then begin //tăng luồng trên cung thuận
                f[z,y-1]^ [x]:=f[z,y-1]^ [x] + tang;
                x := z;
                y := y-1;
            end
            else begin //giảm luồng trên cung ngược
                z := -z;
                f[x,y]^ [z]:=f[x,y]^ [z] - tang;
            end
        until x=td;
    end;
end;

```

```

        x := z;
        y := y+1;
    end;
    until (x=sd) and (y=sc) ; //tới đỉnh phát
end;
end;
procedure solve; //thuật toán tìm luồng cực đại
begin
    repeat
        findpath; //tìm đường tăng luồng
        if pathfound then //nếu tìm thấy đường tăng luồng
            tangluong; //thì thực hiện tăng luồng
        until not pathfound; //cho đến khi không còn đường tăng luồng
    end;
procedure result;
var i,kq : integer;
begin
    kq:=0;
    for i:=1 to n do
        if f[1,0]^[i]=1 then inc(kq) ; //số lần đặt tốt tại cột i
        assign(ff,fo) ; rewrite(ff) ;
        writeln(ff,kq) ; //xuất kết quả vào tệp output
        close(ff) ;
    end;
BEGIN
    read_input;
    solve;
    result;
END.

```

Bài 58. Bảo vệ thành phố

Một hệ thống giao thông gồm N thành phố đánh số từ 1 đến N . Giữa một số cặp thành phố có đoạn đường một chiều nối với nhau. Quân địch đang ở thành phố N và định tiến về thành phố 1. Với hai thành phố U và V có đường theo hướng từ U đến V , bộ Tham mưu có thể tính được cần tối thiểu bao nhiêu lính thì có thể ngăn không cho quân địch tiến từ U về V theo

đoạn đường đó. Hãy cho biết để bảo vệ thành phố 1, cần tối thiểu bao nhiêu lính và số lính này bố trí trên những đoạn đường nào.

Dữ liệu vào được cho bởi file BVTP.IN trong đó dòng thứ nhất ghi số $N \leq 5000$, tiếp theo là không quá 10000 dòng, mỗi dòng ghi ba số U, V, W, $W \leq 60000$ với ý nghĩa có đường một chiều từ U đến V và cần ít nhất W lính phòng thủ trên đoạn đường đó. Chú ý rằng có thể có hai thành phố U và V với nhiều đường từ U đến V.

Kết quả ra file BVTP.OUT như sau: dòng thứ nhất ghi số M là tổng số ít nhất lính cần bố trí, tiếp theo là một số dòng, mỗi dòng ghi ba số X, Y, Z với ý nghĩa cần bố trí Z lính phòng thủ trên đoạn đường từ X đến Y.

BVTP.IN	BVTP.OUT
3 1 7	23
6 1 47	3 1 7
7 2 12	4 6 16
7 2 20	
2 4 11	
7 3 15	
5 3 20	
7 4 10	
4 2 21	
4 6 16	
6 5 23	

Gợi ý. Xét mạng với tập đỉnh V gồm các thành phố, các cung là các đoạn đường với khả năng thông qua bằng số quân cần bố trí trên đoạn đường đó. Tổng số quân cần bố trí ngăn cản địch phải bằng khả năng tràn đến lớn nhất của địch nghĩa là bằng giá trị của luồng cực đại trên mạng. Gọi lát cắt hẹp nhất của mạng là phân hoạch (X,Y). Các đoạn đường cần bố trí quân chính là các cung đi từ đỉnh thuộc X đến đỉnh thuộc Y.

Chương trình. Danh sách các đỉnh kẻ được tổ chức theo kiểu liên kết.

```
uses crt;
const fi      = 'bvtp.in';
      fo      = 'bvtp.out';
      maxn    = 5000;
      inf     = 65530;
type link     = ^node;
```



```

node      = record
                ke      : integer;
                w        : word;
                f        : word;
                next     : link;
        end;
tmin      = array[1..maxn] of longint;
var f,g    : text;
n          : longint;
a          : array[1..maxn] of link; {danh sách kề tổ chức bằng liên kết}
found      : boolean; {cờ báo tìm thấy đường tăng luồng}
qu         : array[1..maxn] of integer; {hàng đợi, tìm đường tăng luồng}
dau, cuoi  : longint; {đầu và cuối hàng đợi}
truoc      : array[1..maxn, 1..2] of word;
        {truoc[i,1]: đỉnh tới đỉnh i, truoc[i,2]: độ tăng luồng trên cung tới i, dựa vào
        nhãn này sẽ biết được cung tới i là cung thuận hay cung ngược }
min        : ^tmin; {nhãn báo khả năng luồng cho phép qua mỗi đỉnh}
sum        : longint; {tổng số quân cần bố trí}

```

Procedure init;

var i : longint;

begin

for i := 1 to maxn do a[i]:=nil; {khởi trị các danh sách kề của đỉnh i}
new(min);

end;

Procedure add(u,v : integer; w,lu : word);

{thêm đỉnh v vào danh sách kề của đỉnh u và ghi nhận trọng số cung (u,v) là w, luồng trên cung (u,v) là lu}

var l : link;

begin

new(l);

l^.ke := v;

l^.w := w;

l^.f := lu;

l^.next := a[u];

a[u] := l;

end;

Procedure nhap;

```

var u,v   : integer;
    w     : word;
begin
    assign(f,fi); reset(f);
    readln(f,n); {số đỉnh của đồ thị}
    while not seekeof(f) do begin
        readln(f,u,v,w); {đọc cung (u,v) có trọng số w}
        add(u,v,w,0); {cung (u,v) có thông lượng tối đa là w, luồng trên cung là 0}
        add(v,u,inf,inf); {tạo cung nghịch (v,u) thông lượng và luồng tối đa}
    end;
    close(f);
end;

function nhonhat( s1,s2 : longint ) : longint;
begin
    if s1 < s2 then nhonhat := s1 else nhonhat := s2;
end;

Procedure findpath; {tìm đường tăng luồng bằng BFS}
var i,j : longint;
    l    : link;
    p    : link;
begin
    for i := 1 to n do
        min^[i] := maxlongint; {khởi trị khả năng luồng thông qua các đỉnh i}
        found := false; {khởi trị : chưa tìm được đường tăng luồng}
        fillchar(truoc,sizeof(truoc),0);
        truoc[n][1] := n; {khởi trị đỉnh tới đỉnh phát n là chính đỉnh n}
        truoc[n][2] := 0; {khởi trị độ tăng luồng trên cung (n,n) }
        dau := 1;
        cuoi := 1;
        qu[cuoi] := n; { nạp đỉnh n vào hàng đợi}
        while dau <= cuoi do begin
            i := qu[dau]; {lấy đỉnh i ở đầu hàng đợi}
            inc(dau);
            l := a[i]; {l : danh sách các đỉnh kề đỉnh i}
            while l <> nil do begin
                j := l^.ke; {j đỉnh kề với đỉnh i}
                if (truoc[j][1]=0) then begin {j chưa thuộc đường tăng luồng}

```

```

if ( $l^{\wedge}.f < l^{\wedge}.w$ ) then begin {luồng nhỏ hơn thông lượng max}
    truoc[j][1] := i; {trên đường tăng luồng có đỉnh i trước đỉnh j}
    truoc[j][2] :=  $l^{\wedge}.w - l^{\wedge}.f$ ; {xác nhận độ tăng luồng trên (i,j)}
    min^[j] := nhonhat(min^[i],  $l^{\wedge}.w - l^{\wedge}.f$ ); {sửa nhãn đỉnh j}
    { nạp thêm đỉnh j vào hàng đợi}
    inc(cuoi); qu[cuoi] := j;
    if j=1 then begin {đường tăng luồng tới đỉnh thu (đỉnh 1)}
        found := true; {xác nhận đã tìm được đường tăng luồng}
        exit;
    end;
end else
if ( $l^{\wedge}.w = \text{inf}$ ) then begin {cung (i, j) là cung ngược}
    {chọn một cung thuận có luồng chảy từ j về i}
    p := a[j];
    while p <> nil do begin
        if ( $p^{\wedge}.ke = i$ ) and ( $p^{\wedge}.w <> \text{inf}$ ) then
            break;
        p := p^{\wedge}.next;
    end;
    {tìm được cung thuận (j, i) có luồng chảy tới i là  $p^{\wedge}.f$  thì kết nạp cung
    nghịch (i, j) vào đường tăng luồng và xác nhận lại nhãn của j}
    if ( $p^{\wedge}.f > 0$ ) then begin
        truoc[j][1] := i; {xác nhận trước đỉnh j là đỉnh i}
        truoc[j][2] :=  $p^{\wedge}.f$ ; {độ giảm luồng trên cung ngược (i, j)}
        {sửa nhãn về độ giảm luồng qua đỉnh j}
        min^[j] := nhonhat(min^[i],  $p^{\wedge}.f$ ); {luồng từ j về i}
        { nạp thêm đỉnh j vào hàng đợi}
        inc(cuoi);
        qu[cuoi] := j;
        if j=1 then begin {đường tăng luồng tới đỉnh thu (đỉnh 1)}
            found := true; {xác nhận đã tìm được đường tăng luồng}
            exit;
        end;
    end;
end;
end;
end;
l := l^{\wedge}.next; {thực hiện tìm kiếm theo BFS}

```

```

        end;
    end;
end;
Procedure incflow;
var i,j    : longint;
    l      : link;
    flag   : boolean;
begin
    i := 1;
    while i <> n do begin
        j:=truoc[i][1]; {điều chỉnh luồng từ đỉnh thu (đỉnh 1) ngược về đỉnh n}
        flag := false;
        l := a[j]; {xét các đỉnh có thể đi tới từ j để tìm ra cung thuận (j, i)}
        while l <> nil do begin
            {cung (j, i) là cung thuận}
            if (l^.ke=i) and (l^.w-l^.f=truoc[i][2]) then begin
                flag := true;
                l^.f := l^.f + min^[1]; {tăng luồng trên cung thuận}
                break;
            end;
            l := l^.next; {tìm kiếm trong danh sách kề của đỉnh j cho đến khi tìm được}
        end;
        {nếu(j, i) không là cung thuận thì tìm cung ngược (i, j)}
        if not flag then begin
            l := a[i]; {xét các đỉnh có thể đi tới từ i}
            while l <> nil do begin
                {cung (i, j) là cung ngược}
                if (l^.ke=j) and (l^.f = truoc[i][2]) then begin
                    l^.f := l^.f - min^[1]; {giảm luồng trên cung ngược (i, j)}
                    break;
                end;
                l := l^.next; {tìm kiếm trong danh sách kề của đỉnh i}
            end;
        end;
        i := j; {chuyển tới đỉnh i khác theo hướng tiến dần về phía đỉnh phát (đỉnh n)}
    end;
end;
end;

```

Procedure xuly; *{Thực hiện thuật toán tìm luồng cực đại}*

```
begin
    sum := 0;
    repeat
        findpath;
        if found then begin
            incflow;
            sum := sum + min^[1]; {tính giá trị luồng cực đại}
        end;
    until not found;
end;
```

Procedure ghi;

```
var i,j : longint;
    l : link;
```

```
begin
    assign(g,fo); rewrite(g);
    writeln(g,sum); {ghi số quân lính cần bố trí}
    {tìm lát cắt hẹp nhất, sẽ bố trí quân trên các cạnh qua lát cắt hẹp nhất}
    for i := 1 to cuoi do begin
        L := a[qu[i]]; {Tập X gồm các đỉnh là qu[i] với i=1..cuoi là tập đỉnh còn
        nằm trong hàng đợi trong lần tìm đường tăng luồng không thành công (lần cuối cùng). L :
        danh sách các đỉnh có thể đi tới từ đỉnh qu[i]. Cần bố trí quân trên các cung nối từ qu[i]
        tới các đỉnh thuộc L không có đường tăng luồng qua các đỉnh này thuộc tập  $Y=V\setminus X$ }
        while l <> nil do begin
            if (l^.w<>inf) and (truoc[l^.ke][1]=0) then
                writeln(g,qu[i], ' ', l^.ke, ' ', l^.w);
            l := l^.next;
        end;
    end;
    close(g);
end;
```

BEGIN

```
    init;
    nhap;
    xuly;
    ghi;
    dispose(min);
```

END.

Chương trình 2 (Free Pascal). Các cung được tổ chức theo danh sách liên thuộc.

```
const maxN = 5001;
      maxM = 10001;
      maxC = 60001;
      fi   = 'bvtp.inp';
      fo   = 'bvtp.out';
type t_edge = record
      x,y : integer;
      c,f : integer;
end;
t_queue = record
      items : array[1..maxN] of integer;
      front : integer;
      rear  : integer;
end;
var e      : array[-maxM..maxM] of t_edge;
    link : array[-maxM..maxM] of integer;
    head : array[1..maxN] of integer;
    trace,
    x : array[1..maxN] of integer; {Quản lý tập X trong lát cắt hẹp nhất}
    n,m,s,t      : longint;
    flow_value : longint;
    queue : t_queue;
Procedure enter;
var u,v,w : longint;
    f : text;
begin
    assign(f,fi); reset(f);
    readln(f,n);
    s := n;
    t := 1;
    fillchar(head[1], n*sizeof(head[1]),0);
    m := 0;
    while not eof(f) do begin
        inc(m);
        readln(f,u,v,w);
        with e[m] do begin
            x := u;
            y := v;
            c := w;
            link[m] := head[u];
            head[u] := m;
        end;
    end;
```

```

with e[-m] do begin
    x := v;
    y := u;
    c := 0;
    link[-m] := head[v];
    head[v] := -m;
end;
end;
end;
Procedure init_flow;
var i : integer;
begin
    for i:=-m to m do e[i].f := 0;
    flow_value := 0;
end;
function find_path: boolean;
var u,v,i : integer;
begin
    fillchar(x, sizeof(x),0);
    find_path := true;
    fillchar(trace[1],n*sizeof(trace[1]),0);
    trace[s] := 1;
    with queue do begin
        items[1] := s;
        front := 1;
        rear := 1;
        repeat
            u := items[front];
            inc(front);
            i := head[u];
            while i<>0 do begin
                v := e[i].y;
                if (trace[v]=0) and (e[i].f<e[i].c) then begin
                    trace[v] := i;
                    if e[i].c-e[i].f>0 then begin
                        x[v] := 1; {trong lần tăng luồng cuối, đánh dấu v thuộc tập X}
                    end;
                    if v=t then exit;
                    inc(rear);
                    items[rear] := v;
                end;
                i := link[i];
            end;
        until front>rear;
        find_path := false;
    end;
end;

```

```

end;
Procedure augment_flow;
var delta, v, i : longint;
begin
  v := t;
  delta := maxC;
  repeat
    i := trace[v];
    if e[i].c-e[i].f<delta then
      delta :=e[i].c-e[i].f;
    v := e[i].x;
  until v=s;
  v := t;
  repeat
    i := trace[v];
    inc(e[i].f, delta);
    dec(e[-i].f, delta);
    v := e[i].x;
  until v=s;
  inc(flow_value,delta);
end;
Procedure print_result;
var i,k,j : integer; g : text;
begin
  assign(g,fo); rewrite(g);
  writeln(g,flow_value);
  for i:=1 to n do
    for j:=1 to n do
      if (x[i]=1) and (x[j]=0) then {i thuộc tập X, j thuộc tập Y}
        for k:=1 to m do {tìm cung k là cung (i,j) qua lát cắt hẹp nhất}
          if (e[k].x= i) and (e[k].y=j) and (e[k].f>0) then
            writeln(g,e[k].x, ' ',e[k].y, ' ',e[k].c);
        close(g);
      end;
    end;
  BEGIN
    enter;
    init_flow;
    while find_path do augment_flow;
    print_result;
  END.

```

Bài 59. Patrol (Tuần tra)

Tại Nha trang, ngay trong thành phố có một bể tắm nước khoáng nóng tự nhiên, được dẫn từ trên núi xuống. Từ nguồn, nước được dẫn theo các đoạn ống nước, qua các bể nước trung gian để xử lý trước khi đổ vào bể

tắm. Các bể nước khác nhau đều ở các độ cao khác nhau và được đánh số thứ tự bắt đầu từ 1 (là nguồn nước) đến N (là bể tắm). Nước chảy từ bể ở cao hơn (có số hiệu nhỏ hơn) xuống bể ở thấp hơn (có số hiệu lớn hơn) nếu có đoạn ống nước nối trực tiếp chúng. Như vậy nước từ nguồn có thể chảy theo các đoạn ống nước khác nhau tạo thành một tuyến nước đổ vào bể tắm. Mỗi bể nước có ít nhất một đoạn ống nước dẫn nước vào và một đoạn ống nước dẫn nước ra.

Vì là nước khoáng quý hiếm nên các đoạn ống nước phải được tuần tra thường xuyên. Hàng ngày, Ban quản lý phải điều động cán bộ đi tuần tra và trả tiền bồi dưỡng cho họ. Một lượt đi tuần tra phải xuất phát từ nguồn theo các đoạn ống nước của cùng một tuyến nước nào đó và kết thúc tại bể tắm. Ban quản lý phải trả cho một lượt tuần tra một khoản tiền là: $T \times (\text{số đoạn ống nước trên tuyến tuần tra})$ đồng.

Yêu cầu: Hãy giúp Ban quản lý tổ chức tuần tra sao cho mỗi đoạn ống nước phải được tuần tra ít nhất hai lượt và tổng số tiền phải trả là ít nhất.

Dữ liệu vào: từ file văn bản PATROL.INP

Dòng đầu chứa 2 số nguyên dương N, T tương ứng là số lượng các bể nước và số tiền bồi dưỡng cho một lượt người tuần tra đi qua một đoạn ống nước ($1 < N, T \leq 100$)

Mỗi dòng trong N-1 dòng tiếp theo ghi dãy các số nguyên. Các số trong dòng (i+1) xác định số hiệu các bể nước có đoạn ống dẫn nước trực tiếp từ bể thứ i vào chúng. Số thứ nhất trong dòng là k đó là số lượng các bể nước đó và k số tiếp theo là số hiệu của chúng.

Hai số liên tiếp trên cùng một dòng cách nhau một dấu cách

Kết quả: ghi ra file văn bản PATROL.OUT:

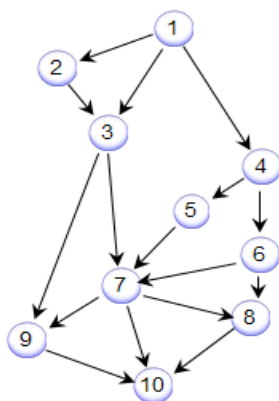
Dòng đầu tiên là tổng số tiền phải trả ít nhất.

Các dòng sau, mỗi dòng nêu một hành trình, bắt đầu từ bể 1 đến bể N.

Ví dụ

PATROL . INP	PATROL . OUT
--------------	--------------

10	2				84				
3	3	2	4		1	2	3	7	8
1	3				1	2	3	7	8
2	9	7			1	3	9	10	
2	5	6			1	3	9	10	
1	7				1	4	5	7	9
2	7	8			1	4	5	7	9
3	9	10	8		1	4	6	7	10
1	10				1	4	6	7	10
1	10				1	4	6	8	10
					1	4	6	8	10



Gợi ý. Đây là bài toán tìm luồng có giá trị nhỏ nhất thỏa mãn yêu cầu.

Khi xây dựng mạng, các đỉnh có số cung vào bằng số cung ra có thể bỏ qua (gọi là sơ giản mạng). Vậy các đỉnh từ 2 đến $N-1$ chia làm hai loại đỉnh: Loại A là những đỉnh có số cung vào lớn hơn số cung ra, loại B là những đỉnh có số cung vào nhỏ hơn số cung ra. Đồng thời thêm hai đỉnh là 0 và $N+1$. Xây dựng mạng có đỉnh phát s là đỉnh 0, đỉnh thu t là đỉnh $N+1$ với trọng số các cung (khả năng thông qua) như sau:

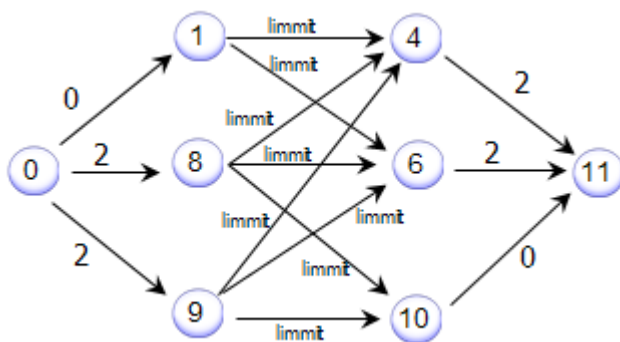
Với đỉnh i thuộc loại A thì: trọng số cung $(0,i)$ gán bằng $2 \cdot (\text{bậc vào } i - \text{bậc ra } i)$, và trọng số cung (i,n) gán bằng vô cùng (limmit).

Với đỉnh j thuộc loại B thì: trọng số cung $(j, n+1)$ gán bằng $2 \cdot (\text{bậc ra } j - \text{bậc vào } j)$, trọng số cung $(n,n+1)$ bằng vô cùng (limmit)

Gọi tổng bậc vào của các đỉnh loại A là tv , tổng các bậc ra của các đỉnh loại B là tr thì: nếu $tv > tr$ sẽ gán trọng số cung $(0, 1)$ bằng 0, trọng số cung $(n, n+1)$ bằng $tv - tr$. Ngược lại nếu $tv \leq tr$ thì: trọng số cung $(0,1)$ bằng $tr - tv$, trọng số cung $(n, n+1)$ bằng 0, trọng số cung $(n, n+1)$ bằng $(tv - tr)$

Trọng số các cung (i,j) với i thuộc A, j thuộc B được gán như sau: Nếu có các ống nước từ bể i đến bể j thì trọng số cung (i,j) là vô cùng.

Trọng số cung $(1,n)$ cũng bằng vô cùng.



Mỗi lần tìm luồng không thành công thì điều chỉnh dần thêm một lượng là *mid* vào trọng số cung (0,1) và cung (n, n+1). Nếu sau điều chỉnh tìm được luồng thì lại giảm lượng *mid*. Cách điều chỉnh này có thể tiến hành bằng tìm kiếm nhị phân giá trị của *mid* trong khoảng (0, vô cùng).

Ngoài ra, có thể dùng thuật toán tìm đường đi ngắn nhất để biết đường đi nối bể *i* và bể *j* có số ống nước ít nhất.

Chương trình.

```

const  fi      = 'patrol.in7654321';
       fo      = 'patrol.ou7654321';
       nmax    = 101;
       vc      = 120;
       limit   = 30000;
type   mang    = array[0..nmax, 0..nmax] of byte;
       mang1   = array[0..nmax,0..nmax] of integer;
       mang2   = array[0..nmax] of longint;
var f,g       : text;
    a,
    kc        : mang;
    tg        : ^mang;
    c,
    flow      : mang1;
    bv,br,
    d,
    tr        : mang2;
    n,t,
    best,
    min,cost,
    s,
    num       : longint;
    found     : boolean;
  
```

```

procedure nhap;
var i,j,u,bac : longint;
begin
  assign(f,fi); reset(f);
  read(f,n,t); {Số bể, số tiền trả công tuần tra 1 lượt cho 1 ống dẫn nước}
  fillchar(a,sizeof(a),0); {Khởi trị ma trận kề (đỉnh là các bể nước)}
  fillchar(bv,sizeof(bv),0); {Khởi trị bậc vào của các đỉnh}
  fillchar(br,sizeof(br),0); {Khởi trị bậc ra của các đỉnh}
  for i:=1 to n-1 do begin
    read(f,bac); {số đỉnh kề với i}
    for u:=1 to bac do begin
      read(f,j); {đỉnh j kề với i}
      a[i,j]:=1; {xác nhận có cung (i,j)}
      inc(br[i]); {tăng bậc ra của đỉnh i}
      inc(bv[j]); {tăng bậc vào của đỉnh j}
    end;
  end;
  close(f);
end;

procedure dfs(i : integer); {Tìm tiếp hành trình từ i}
var j : integer;
begin
  for j:=i+1 to n do
    if (tr[j]=0) {đỉnh j chưa thăm}
    and(a[i,j]=1) {có ống nước nối trực tiếp từ i tới j}
    and(c[i,j]>0) then begin {chưa hết lượt tuần tra}
      tr[j]:=i; {xác nhận vết: trước đỉnh j là đỉnh i}
      dfs(j); {gọi đệ quy thăm tiếp từ j}
    end;
  end;

procedure find_path; {Tìm đường của một lượt tuần tra}
begin
  fillchar(tr,sizeof(tr),0); {Khởi trị mảng vết}
  tr[1]:=n+1;
  found := false; {Khởi trị cờ báo có tìm thấy hay không}
  dfs(1); {Tìm đường một lần tuần tra bắt đầu từ bể 1}
  found:=(tr[n]<>0); {Tìm được khi tuần tra tới bể N}
end;

procedure lannguoc(j:longint); {Lần ngược hành trình (phục vụ ghi output)}
begin
  if j>1 then begin
    dec(c[tr[j],j]); {giảm số lần còn có thể tuần tra trên cung (tr[j],j)}
    lannguoc(tr[j]); {đệ quy để tiếp tục lần ngược hành trình}
  end;
end;

```

```

    write(g,j, ' '); {chú ý : do cơ chế stack, hành trình sẽ được ghi từ 1 về phía N}
end;
procedure write_path; {lần ngược hành trình một lượt tuần tra để ghi hành trình
này vào tệp output}
begin
    lannguc(n);
    writeln(g);
end;
procedure tim_hanh_trinh; {Tìm các lượt hành trình tuần tra và ghi các hành
trình này vào tệp output}
begin
    fillchar(flow,sizeof(flow),0);
    repeat
        find_path;
        if found then write_path;
    until not found;
end;
procedure ghikq; {Ghi toàn bộ kết quả ra tệp output}
begin
    assign(g,fo); rewrite(g);
    writeln(g,cost*t);
    tim_hanh_trinh;
    close(g);
end;
procedure tinh_mang_kc; {Floyd tính số đoạn ống ít nhất giữa hai bể i và j, lưu
vào mảng khoảng cách kc(N)}
var k,i,j : longint;
begin
    fillchar(kc,sizeof(kc),0); {Khởi trị mảng kc}
    for i:=1 to n do
        for j:=1 to n do
            if a[i,j]=1 then begin {có ống nối trực tiếp bể i và bể j}
                kc[i,j]:=1; {thì khoảng cách i và j bằng 1}
                tg^[i,j]:=i; {vết : bể trung gian giữa i và j coi là i}
            end else kc[i,j]:=vc; {Khởi trị khoảng cách bằng vô cùng khi chưa có đoạn
ống nối trực tiếp i và j}{}
            for k:=1 to n do {Tính nhân khoảng cách ngắn nhất}
                for i:=1 to n do
                    for j:=1 to n do
                        if kc[i,j]>kc[i,k]+kc[k,j] then begin
                            kc[i,j]:=kc[i,k]+kc[k,j];
                            tg^[i,j]:=tg^[k,j]; {lưu vết đỉnh trung gian}
                        end;
                    end;
                end;
            end;
        end;
    end;
procedure khoi_tao_mang_c; {Tạo trọng số các cung cho mạng để tìm luồng}

```

```

var i,j,tv,tr : longint;
begin
    tv:=0; tr:=0; {Khởi trị tổng luồng vào và tổng luồng ra}
    for i:=2 to n-1 do begin {xét các đỉnh có bậc vào lớn hơn bậc ra}
        if bv[i]>br[i] then begin
            inc(tv,2*(bv[i]-br[i])); {tính tổng luồng vào, đã sơ giản bởi lượng
khả năng đi ra khỏi đỉnh i}
            c[0,i]:=2*(bv[i]-br[i]); {trọng số cung (0,i)}
            c[i,n]:=limit; {trọng số cung (i,n)}
        end;
        if bv[i]<br[i] then begin {xét các đỉnh có bậc vào nhỏ hơn bậc ra}
            inc(tr,2*(br[i]-bv[i])); {tính tổng luồng ra, đã sơ giản bởi lượng khả
năng đi vào đỉnh i}
            c[i,n+1]:=2*(br[i]-bv[i]); {trọng số cung (i,n+1)}
            c[1,i]:=limit; {trọng số cung (1,i)}
        end;
    end;
    {xây dựng trọng số cung (0,1) và cung (n,n+1)}
    if tv>tr then c[0,1]:=0
    else c[0,1]:=tr-tv;
    c[n,n+1]:=tv+c[0,1]-tr;
    {xây dựng trọng số các cung (i,j) mà i ở phía các đỉnh có bậc vào lớn hơn, j ở phía các
đỉnh có bậc ra lớn hơn}
    for i:=2 to n-1 do
        if bv[i]>br[i] then
            for j:=2 to n-1 do
                if (bv[j]<br[j]) and (kc[i,j]<> vc) then
                    c[i,j]:=limit;
            c[1,n]:=limit;
            s:=tv+c[0,1]; {thêm c[0,1] vào tổng luồng vào}
        end;
procedure chuanbi; {tạo mảng nhãn khoảng cách và mạng phục vụ tìm luồng}
begin
    tinh_mang_kc;
    khoi_tao_mang_c;
end;
procedure khoi_tao; {Khởi trị luồng 0}
begin
    fillchar(flow,sizeof(flow),0);
    num:=0;
end;
procedure chbitim; {Khởi trị các mảng nhãn d (khả năng cho luồng qua từng đỉnh),
mảng vết tr phục vụ tìm đường tăng luồng}
var i : longint;
begin

```

```

for i:=0 to n+1 do
    begin d[i]:=limit; tr[i]:=limit; end;
d[0]:=0; tr[0]:=0;
end;
procedure find_path_augment_flow; {Tìm đường tăng luồng}
var
    k,i,j,ts : longint;
    stop : boolean;
begin
    repeat
        stop:=true;
        for i:=0 to n do
            for j:=i+1 to n+1 do
                if (c[i,j]>0)and(flow[i,j]<c[i,j])then begin {(i,j): cung thuận}
                    ts:=kc[i,j];
                    if (d[j]>d[i]+ts)and(abs(tr[i])<>j) then begin
                        d[j]:=d[i]+ts; {tối ưu lại nhãn j}
                        tr[j]:=i; {ghi vết}
                        stop:=false;
                    end;
                end;
            end;
        for i:=1 to n+1 do
            for j:=0 to i-1 do
                if (c[j,i]>0)and(flow[j,i]>0) then begin {(j,i): cung ngược}
                    ts:=-kc[j,i];
                    if (d[j]>d[i]+ts)and(abs(tr[i])<>j) then begin
                        d[j]:=d[i]+ts; {tối ưu lại nhãn j}
                        tr[j]:=-i;
                        stop:=false;
                    end;
                end;
            end;
        until stop;
    end;
procedure find_min; {Tìm lượng điều chỉnh luồng}
var i,j : longint;
begin
    min:=limit; j:=n+1;
    repeat
        i:=tr[j];
        if i>=0 then begin
            if min>c[i,j]-flow[i,j] then
                min:=c[i,j]-flow[i,j];
        end else begin
            i:=-i;
            if min>flow[j,i] then min:=flow[j,i];
        end;
    end;

```

```

    j:=i;
until j=0;
if min+num>s then {giá trị luồng sau khi điều chỉnh không vượt quá dự định s}
    min:=s-num; {nếu vượt quá thì lượng tăng luồng lần này phải điều chỉnh như
lệnh bên để giá trị luồng đạt được bằng s}
end;
procedure augment_flow; {Tăng luồng}
var i,j : longint;
begin
    inc(num,min); {num là giá trị luồng thêm lượng min}
    j:=n+1;
    repeat
        i:=tr[j];
        if i>=0 then inc(flow[i,j],min) {tăng luồng trên cung thuận}
        else begin
            i:=-i;
            dec(flow[j,i],min); {giảm luồng trên cung ngược}
        end;
        j:=i;
    until j=0;
end;
procedure ford_fulkerson; {Thực hiện thuật toán tìm luồng có giá bằng s}
begin
    repeat
        chbitim;
        find_path_augment_flow;
        if d[n+1]=limit then break;
        find_min;
        if min=0 then break;
        augment_flow;
    until num=s;
end;
procedure tinh_tcp; {Tính tổng chi phí phải trả}
var i,j : longint;
begin
    cost:=0; {Khởi trị tổng chi phí (số ống kiểm tra) chưa nhân với đơn giá t}
    for i:=1 to n do inc(cost,2*br[i]); {Số ống coi như đã kiểm tra do ban
đầu sơ giản mạng}
    for i:=1 to n-1 do
        for j:=i+1 to n do
            if (flow[i,j]>0) then
                inc(cost,kc[i,j]*flow[i,j]); {số ống nhân với số lần kiểm tra đoạn i tới j}
        end;
procedure xet(i,j,lt : longint);
{chương trình con phục vụ tính số lần tuần tra trên đoạn (i,j)}

```



```

var u,v : longint;
begin
  v:=j;
  repeat
    u:=tg^[i,v];
    inc(c[u,v],lt);
    v:=u;
  until v=i;
end;
procedure tinh_kq;
var i,j : longint;
begin
  fillchar(c, sizeof(c), 0); {Xây dựng lại mảng C, c[i,j] là tổng số lần kiểm tra trên các ống thuộc đoạn (i,j)}
  {Số lần tối thiểu tuần tra mỗi đoạn ống là 2. Chú ý do sơ giản mạng ban đầu nên kết quả luồng trên mạng sơ giản đã bỏ qua, nên phải có đoạn lệnh này}
  for i:=1 to n do
    for j:=i+1 to n do
      if a[i,j]=1 then c[i,j]:=2;
      {Tinh thêm số lần kiểm tra khác, dựa trên kết quả luồng trên mạng đã sơ giản }
      for i:=1 to n-1 do
        for j:=i+1 to n do
          if flow[i,j]>0 then xet(i,j,flow[i,j]);
end;
procedure dieukhien;
var l,r,mid : longint;
begin
  l:=0; r:=limit; best:=limit;
  repeat {Dùng tìm kiếm nhị phân để tìm lượng tốt nhất điều chỉnh giá trị luồng}
    writeln(l, ' ', r);
    mid:=(l+r) div 2;
    inc(s,mid); inc(c[0,1],mid); inc(c[n,n+1],mid);
    khoi_tao;
    ford_fulkerson;
    if num=s then begin {có luồng đạt được giá trị s thì}
      if mid<best then best:=mid; {nếu mid tốt hơn thì gán lại best}
      r:=mid-1; {và giảm biên phải để lần sau mid nhỏ hơn}
    end
    else l:=mid+1; {không có luồng đạt giá trị s thì tăng biên trái để lần sau mid lớn hơn}
    dec(s,mid); dec(c[0,1],mid); dec(c[n,n+1],mid);
  until l>r;
  {lượng tốt nhất để thêm vào giá trị luồng s là best}
  inc(s,best); inc(c[0,1],best); inc(c[n,n+1],best);
  khoi_tao;
  ford_fulkerson;

```

```

    tinh_tcp;
    tinh_kq;
end;
BEGIN
    new(tg);
    nhap;
    chuanbi;
    dieukhien;
    ghikq;
    dispose(tg);
END.

```

Bài 60. Chia cắt địch

Cuối năm 1944, quân Nga phản công. Trong vùng X của nước Nga còn bị phát xít Đức chiếm đóng có N thành phố. Dọc các con đường giữa hai thành phố của vùng này đều có một lực lượng quân Đức canh phòng. Bộ tham mưu quân sự Nga vùng X chỉ đạo một chiến dịch phản công tiêu diệt địch. Trận phản công đầu tiên của chiến dịch nhằm chia cắt địch trong vùng X thành hai vùng tách biệt không thể liên hệ được với nhau chuẩn bị cho những trận phản công tiếp theo. Tuy nhiên, cần tính toán nên đánh địch trên những con đường nào là lợi nhất (chỉ điều động lực lượng quân ít nhất đủ giành thắng lợi cho trận phản công đầu tiên này). Biết rằng muốn giành thắng lợi chiếm lại một con đường, quân đội Nga cần phải bố trí lực lượng ít nhất là bằng lực lượng quân Đức chiếm đóng trên con đường ấy.

Hãy viết chương trình giúp Bộ tham mưu quân sự Nga vùng X thực hiện kế hoạch chia cắt địch mà chỉ cần điều động ít quân nhất.

Dữ liệu vào từ file văn bản CHIACAT.IN :

Dòng đầu là hai số nguyên dương N và M, trong đó N ($2 \leq N \leq 100$) là số thành phố trong vùng X, M là số con đường nối các cặp hai thành phố trong vùng X.

M dòng tiếp theo, mỗi dòng 3 số nguyên dương i, j và w thể hiện trên con đường nối thành phố i và thành phố j hiện có w lính Đức canh phòng ($1 \leq w \leq 100$).

Kết quả ra file văn bản CHIACAT.OUT :

Dòng đầu là số lính Nga cần điều động vào kế hoạch chia cắt địch.

Các dòng sau, mỗi dòng hai số u và v thể hiện một con đường (u, v) mà quân Nga cần chiếm lại trong trận phản công đầu tiên này.

Ví dụ

CHIACAT . IN	CHIACAT . OUT
10 18	10
1 2 8	1 7
1 4 4	3 6
1 7 1	4 5
2 3 5	6 10
2 4 5	8 10
2 9 9	
3 4 4	
3 6 2	
3 9 7	
4 5 5	
5 6 6	
5 7 5	
5 8 3	
6 8 7	
6 10 1	
7 8 5	
8 10 1	
9 10 9	

Gợi ý.

Định nghĩa : Cho đồ thị $G(V, E)$ vô hướng, liên thông, các cạnh có trọng số dương. Lát cắt tổng quát hẹp nhất trên G là lát cắt có giá trị nhỏ nhất trong mọi lát cắt hẹp nhất tách các cặp đỉnh phát và thu (s, t) . Kí hiệu giá trị lát cắt hẹp nhất tách hai đỉnh phân biệt s và t là $\text{Cut}(G, s, t)$ và kí hiệu giá trị lát cắt tổng quát hẹp nhất là $\text{MinCut}(G)$ thì :

$$\text{MinCut}(G) = \text{Min}\{\text{Cut}(G, s, t) \mid \forall (s, t) s \neq t, s, t \in V\}$$

Tìm lát cắt tổng quát hẹp nhất có thể dựa trên các định lý sau:

Định lý 1 : Cho đồ thị $G(V, E)$ vô hướng, liên thông, N đỉnh, các cạnh $(i, j) \in E$ có trọng số dương là $C[i, j]$. Gọi G_1 là đồ thị G sau khi đã ghép đỉnh s và t thành một đỉnh u , và gán trọng số các cạnh (i, u) là $C[i, u] = C[i, s] + C[i, t]$. Với mọi cặp đỉnh s và t phân biệt thì :

$$\text{MinCut}(G) = \text{Min}\{\text{Cut}(G, s, t), \text{MinCut}(G_1)\}$$

Định lý 2 : Cho đồ thị $G(V, E)$ vô hướng, liên thông, N đỉnh, các cạnh $(i, j) \in E$ có trọng số dương là $C[i, j]$. Chọn một đỉnh bất kỳ $u \in V$, khởi tạo tập $A = \{u\}$. Ta sẽ mở rộng dần tập A bằng cách lần lượt thêm vào A từng đỉnh gắn chặt với A (một đỉnh i thuộc $V \setminus A$ được gọi là gắn chặt với A nhất nếu tổng trọng số các cạnh nối đỉnh i với các đỉnh thuộc A là lớn nhất). Gọi t là đỉnh cuối cùng chưa nạp vào A và s là đỉnh cuối cùng vừa nạp vào A thì lát cắt $(V \setminus \{t\}, \{t\})$ là lát cắt hẹp nhất.

Vậy có thể không cần tìm luồng cực đại, vẫn tìm được lát cắt hẹp nhất của hai đỉnh đặc biệt s và t nêu trên (t là đỉnh cuối cùng chưa nạp vào A và s là đỉnh cuối cùng vừa nạp vào A).

Dựa vào hai định lý nêu trên, có thể tìm lát cắt tổng quát hẹp nhất theo thuật toán sau :

- Khởi tạo $\text{MinCut}(G)$ bằng vô cùng
- Vòng lặp thực hiện $n-1$ lần :

Bước 1. Khởi tạo tập A rỗng. Tìm một đỉnh tùy ý nạp vào A .

Bước 2. Tìm s và t bằng cách nạp dần vào A các đỉnh gắn chặt với A (t là đỉnh cuối cùng chưa nạp vào A và s là đỉnh cuối cùng vừa nạp vào A)

Tính $\text{Cut}(G, s, t)$, so tối ưu với $\text{MinCut}(G)$ để xác nhận lại $\text{MinCut}(G)$.

Bước 3. Xây dựng đồ thị G_1 (chập s và t thành một đỉnh u , gán $C[i, u] = C[i, s] + C[i, t]$ với mọi đỉnh i). Nếu G_1 chỉ còn 1 đỉnh thì thoát khỏi vòng lặp còn không thì coi G là G_1 và trở về bước 1.

Về tổ chức dữ liệu : Sử dụng cấu trúc cây để dễ biểu diễn các đỉnh chập. Ban đầu các đỉnh chưa chập có nhãn $L[i] := -1$. Khi chập hai đỉnh s và t thực hiện phép Union hợp nhất hai cây (cây có gốc r_1 chứa s và cây có gốc r_2 chứa t). Chú ý rằng các đỉnh cùng cây với đỉnh t sẽ thuộc cùng một miền trong phân hoạch của lát cắt hẹp nhất (s, t) .

Bài tập 60 là bài toán tìm lát cắt tổng quát nhỏ nhất. Sau đây là chương trình giải sử dụng phương pháp hợp nhất đỉnh, không tìm luồng cực đại.

Chương trình

```
const    fi          = 'chiacat.inp';
         fo          = 'chiacat.out';
```

```

        max          = 100;
var      c,lc        : array[1..max, 1..max] of integer;
        n, m         : integer;
        l,ll         : array[1..max] of integer;
        best         : longint;
        r            : array[1..max] of 0..1;
        ok           : boolean;
        lt           : integer;

procedure read_input;
var      f : text;
        i, u, v : integer;
begin
    assign(f,fi); reset(f);
    readln(f,n,m); // Số thành phố, số con đường
    fillchar(c,sizeof(c),0);
    for i:=1 to m do begin
        readln(f,u,v,c[u,v]); // con đường (u, v) có trọng số c[u,v]
        c[v,u] := c[u,v];
    end;
    close(f);
    lc := c; // lc: lưu mảng trọng số c
end;

procedure init;
var      i : integer;
begin
    for i:=1 to n do l[i] := -1; // Nhãn các đỉnh chưa chấp
    best := maxint; // Khởi trị giá trị của lát cắt tổng quát hẹp nhất
end;

function getroot(i:integer):integer; // Tìm gốc của cây chứa đỉnh i
begin
    while l[i]>0 do i := l[i];
    getroot := i;
end;

procedure union(s,t:integer); // hợp nhất hai cây chứa đỉnh s và t
var      x, i, r1,r2 : integer;
begin
    r1 := getroot(s); // gốc của cây chứa s
    r2 := getroot(t); // gốc của cây chứa t
    x := l[r1] + l[r2];
    if l[r1]<l[r2] then begin // cây hợp nhất có gốc là r1(chứa s)
        l[r2] := r1; // nhãn mới của r2
        l[r1] := x; // nhãn mới của r1
        for i:=1 to n do // sửa lại trọng số các cung nối tới đỉnh chấp s
            if (l[i]=-1) and (i<>r1) and (i<>r2) then begin
                c[s,i] := c[s,i] + c[t,i];
            end;
        end;
    end;
end;

```

```

        c[i,s] := c[i,s] + c[i,t];
    end;
end
else begin {cây hợp nhất có gốc là r2 (cây chứa t)}
    l[r1] := r2;
    l[r2] := x;
    for i:=1 to n do{sửa lại trọng số các cung nối tới đỉnh chấp t}
        if (l[i]=-1) and (i<>r1) and (i<>r2) then begin
            c[t,i] := c[t,i] + c[s,i];
            c[i,t] := c[i,t] + c[i,s];
        end;
    end;
end;
end;

function find : integer; // Tìm một đỉnh còn lại (chưa bị chấp)
var i : integer;
begin
    for i:=1 to n do
        if l[i] < 0 then begin
            find := i;
            exit;
        end;
    end;
end;

procedure expand(i0 : integer; var s, t : integer);
//mở rộng tập A
var i,j , maxd : integer;
    d          : array[1..max] of integer;
    free       : array[1..max] of boolean;
begin
    for i:=1 to n do //các đỉnh còn lại (là các gốc các cây) chưa thuộc tập A
        free[i] := l[i]<0;
    free[i0] := false; //xác nhận đỉnh i0 thuộc tập A
    for i:=1 to n do //sửa nhãn (độ liên kết của các đỉnh còn lại với tập A)
        if free[i] then d[i] := c[i0,i];
    t := i0; //xác nhận đỉnh t là đỉnh vừa được kết nạp vào A
    repeat//kết nạp các đỉnh còn lại vào A cho đến khi chỉ còn một đỉnh ngoài A
        // Tìm đỉnh i có độ gắn kết cao nhất với tập A
        maxd := 0;
        for j:=1 to n do
            if free[j] and (d[j]>maxd) then begin
                maxd := d[j];
                i := j;
            end; // tìm được đỉnh i nếu maxd>0
        if maxd = 0 then break; //không tìm được i, thì thoát, còn không thì
        free[i] := false; //kết nạp đỉnh i vào tập A
        s := t; //xác nhận lại đỉnh s là đỉnh được kết nạp ngay trước đỉnh t
    until false;
end;

```

```

    t := i; //xác nhận lại đỉnh t là đỉnh vừa được kết nạp vào A
    //sửa lại nhãn gắn kết của các đỉnh còn lại với tập A
    for j:=1 to n do
        if free[j] then d[j] := d[j] + c[i,j];
    until false;
end;
procedure prefer (t : integer);
//Tìm trọng số của lát cắt hẹp nhất  $(V \setminus \{t\}, \{t\})$ 
var cost : longint; i,j,k : integer;
begin
    cost := 0;
    for i:=1 to n do begin
        if (l[i]=-1) and (i<>t) then begin //i chưa bị chấp và khác t
            for k:=1 to n do //tìm các đỉnh k đã bị chấp vào t
                if k<>i then
                    if lc[i,k]>0 then
                        if getroot(k)=t then
                            cost:=cost+lc[i,k]; //cộng trọng số các cạnh (i,k) vào cost
            end
        else //i là đỉnh chấp
            if (l[i]<0) and (i<>t) then begin
                for j:=1 to n do //tìm các đỉnh j đã chấp vào i
                    if getroot(j)=i then
                        for k:=1 to n do //tìm các đỉnh k đã chấp vào t, mà k khác j
                            if k<>j then
                                if lc[j,k]>0 then
                                    if getroot(k)=t then
                                        cost:=cost+lc[j,k]; //cộng thêm trọng số các cạnh (j,k) vào cost
                        end;
            end;
        if cost<best then begin //cost tối ưu hơn
            fillchar(r,sizeof(r),0);
            best := cost; //xác nhận lại tối ưu
            for i:=1 to n do //xác nhận các đỉnh i chấp vào t
                r[i] := ord(getroot(i)=t);
            end;
        end;
    end;
end;
procedure solve;
var i          : integer;
    i0, s, t   : integer;
begin
    fillchar(r,sizeof(r),false);
    for i:=n downto 2 do begin
        i0 := find; //tìm một đỉnh là gốc của một cây chấp, xây dựng  $A=\{i0\}$ 
        expand(i0, s, t); //nap dần các đỉnh vào A, hai đỉnh cuối cùng là s, t
    end;
end;

```

```

    prefer(t); // tìm trọng số của lát cắt hợp nhất ( $\bigvee \{t\}, \{t\}$ )
    union(s, t); // hợp nhất hai cây chập chứa đỉnh s và đỉnh t
end;
end;
procedure write_output;
var f      : text;
    i, u,v : integer;
begin
    assign(f, fo); rewrite(f);
    writeln(f, best);
    for u:=1 to n do
    for v := u+1 to n do
        if (lc[u,v]>0) and ((r[u] xor r[v])=1) then
            //u và v so với t, do đó u và v thuộc 2 miền khác nhau của lát cắt
            writeln(f, u, ' ', v);
        close(f);
    end;
BEGIN
    read_input;
    init;
    solve;
    write_output;
END.

```

8. Bài tập tự giải

1. Sửa đường. Trong một thành phố có n nút giao thông và m đường phố hai chiều. Giữa hai nút giao thông có nhiều nhất là một đường phố nối chúng. Hệ thống giao thông đảm bảo sự đi lại giữa hai nút bất kỳ. Sau một thời gian dài, các đường phố xuống cấp nghiêm trọng đòi hỏi ban quản lý giao thông và công trình đô thị phải lên kế hoạch nâng cấp tất cả các đường phố. Khi một đường phố đang trong thời gian nâng cấp thì sự đi lại trên tuyến đường đó bị cấm. Xét về khả năng, với phương tiện kỹ thuật hiện đại và lực lượng nhân công dồi dào, người ta có thể tiến hành nâng cấp cùng lúc k đường phố, bất kể đường phố nào cũng chỉ cần sửa chữa trong một ngày. Tuy nhiên vì vẫn muốn đảm bảo sự đi lại giữa hai nút giao thông bất kỳ trong thời gian sửa chữa, người ta phải lên lịch thi công các tuyến đường một cách hợp lý.

Yêu cầu: Hãy xếp lịch thi công để thời gian nâng cấp toàn bộ các tuyến đường là ngắn nhất.

Dữ liệu: Vào từ file văn bản SCHEDULE . INP

Dòng đầu ghi ba số nguyên dương $n \ m \ k$ ($2 \leq n \leq 100$; $1 < m \leq n*(n - 1)/2$; $1 < k < 10$). m dòng tiếp theo, mỗi dòng có dạng $u \ v$ cho biết giữa hai nút giao thông u và v có một đường phố nối chúng.

Kết quả: Ghi ra file văn bản SCHEDULE . OUT

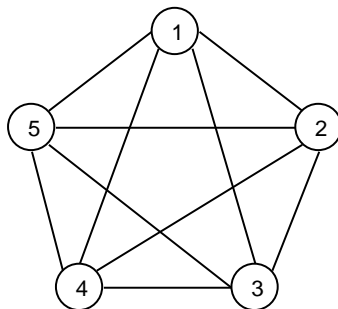
Dòng đầu ghi số ngày tối thiểu cần để thực hiện dự án sửa đường. Nếu không có phương án thì chỉ cần ghi số -1.

Nếu có phương án xếp lịch, m dòng tiếp theo, mỗi dòng có dạng $u \ v \ p$ cho biết sẽ phải tiến hành sửa chữa đoạn đường nối giữa nút u và nút v trong ngày thứ p của dự án. (Ngày khởi công dự án là ngày thứ nhất).

Các số trên một dòng của các tệp Input/Output được ghi cách nhau ít nhất một dấu cách.

Ví dụ

SCHEDULE . INP	SCHEDULE . OUT
5 10 5	2
1 2	1 2 1
1 3	1 3 2
1 4	1 4 2
1 5	1 5 2
2 3	2 3 1
2 4	2 4 2
2 5	2 5 1
3 4	3 4 1
3 5	3 5 2
4 5	4 5 1



2. Mạng giao thông

Theo thiết kế, một mạng giao thông gồm N nút có tên từ 1 đến N ($N \leq 100$). Chi phí để xây dựng một đường hai chiều trực tiếp nối từ nút i đến nút j bằng $a_{ij} = a_{ji} \geq 0$ với mọi i, j ($a_{ii} = 0$). Hai tuyến đường khác nhau không cắt nhau tại điểm không là đầu nút. Hiện đã xây dựng được K tuyến đường.

Bài toán đặt ra như sau: Hệ thống đường đã xây dựng đã đảm bảo sự đi lại giữa hai nút bất kỳ chưa? Nếu chưa, hãy chọn một số tuyến đường cần xây dựng thêm sao cho:

1. Các tuyến đường sẽ xây dựng sẽ cùng với các tuyến đường đã xây dựng phải bảo đảm sự đi lại giữa hai nút bất kỳ.
2. Tổng kinh phí để xây dựng các tuyến đường thêm là ít nhất.

Dữ liệu được cho bởi file văn bản MGT.DAT trong đó dòng thứ nhất ghi hai số N và K. Trong K dòng tiếp theo, mỗi dòng ghi hai số là tên của hai nút, đó là hai đầu của một tuyến đường đã xây dựng. Cuối cùng là N dòng, dòng thứ i ghi các số $a_{i1}, a_{i2}, \dots, a_{in}$.

Kết quả ghi ra file GP.OUT như sau: Dòng thứ nhất ghi số CP là chi phí xây dựng thêm. Nếu $CP > 0$, trong một số dòng tiếp theo, mỗi dòng ghi hai số là tên của hai nút, đó là hai đầu của một tuyến đường cần xây dựng thêm.

Ví dụ:

MGT . DAT	GP . OUT
5 4	1
1 2	3 4
2 3	
3 1	
4 5	
0 1 1 1 1	
1 0 1 1 1	
1 1 0 1 1	
1 1 1 0 1	
1 1 1 1 0	

3. Roads. Bộ giao thông vận tải ở Romania quyết định tiến hành nâng cấp các đoạn đường nối trực tiếp các thành phố với nhau. Mạng lưới giao thông của Romania có thể được mô tả dưới dạng một đồ thị vô hướng liên thông trong đó các đỉnh đại diện cho các thành phố và các cạnh đại diện cho các đoạn đường trực tiếp nối các thành phố. Quá trình nâng cấp sẽ được diễn ra tuần tự giữa các đoạn đường nghĩa là tại mỗi thời điểm chỉ có một đoạn đường được sửa và trong suốt thời gian sửa chữa, nâng cấp đoạn đường đó không được sử dụng. Do đó, trong suốt quá trình nâng cấp có thể xảy ra trường hợp tắc nghẽn giao thông giữa một số thành phố (không tồn tại tuyến

đường giao thông liên lạc giữa 2 thành phố nào đó). Để đảm bảo giao thông thông suốt, bộ giao thông vận tải quyết định sẽ xây dựng thêm một số đoạn đường phụ để hỗ trợ theo nguyên tắc sau:

Các đoạn đường phụ phải là những đoạn đường mới chưa xuất hiện trên bản đồ (nghĩa là nếu tồn tại đoạn đường nối trực tiếp 2 thành phố U, V thì không thể xây dựng thêm đoạn đường phụ nối U với V).

Vì lý do kinh tế nên tổng số đoạn đường phụ cần bổ sung phải là ít nhất có thể được (giả thiết chi phí xây dựng các đoạn đường là như nhau).

Input: ROADS.INP. Dòng đầu tiên ghi hai số N và M là số thành phố và số đoạn đường nối trực tiếp giữa chúng. Trong M dòng sau, mỗi dòng ghi hai số U và V thể hiện có đường nối trực tiếp giữa U và V.

Output: ROADS.OUT. Dòng thứ nhất ghi số K là số đoạn đường tối thiểu cần xây dựng thêm. K dòng sau, mỗi dòng ghi hai số X và Y thể hiện cần xây thêm đoạn đường phụ nối X và Y.

Giới hạn: $3 \leq N \leq 5000$; $2 \leq M \leq 30000$; Thời gian: 0.5s/test; bộ nhớ: 1MB.

Ví dụ:

ROADS . INP	ROADS . OUT
4 3	2
1 2	1 4
2 3	1 3
2 4	

3. Du lịch

Có N thành phố đánh số từ 1 đến N, $N \leq 100$. Giữa một số cặp thành phố có đường đi hai chiều nối trực tiếp. Cần chọn một tour du lịch đi qua ít nhất 3 thành phố khác nhau, mỗi thành phố đứng một lần trừ thành phố đầu tiên đi qua đứng hai lần (lần đầu tiên và lần cuối cùng) sao cho tổng chi phí của tour du lịch là ít nhất.

Dữ liệu: Vào từ file DULICH.INP trong đó dòng đầu tiên ghi hai số nguyên N, M với M là số đoạn đường nối trực tiếp hai thành phố. Tiếp theo là M dòng, mỗi dòng ghi ba số nguyên dương u, v, w với ý nghĩa là có đường đi hai chiều trực tiếp từ thành phố u đến thành phố v với chi phí w,

$w \leq 10000$. Chú ý rằng giữa hai thành phố có thể có nhiều đường nối trực tiếp.

Kết quả: ghi ra file văn bản DULICH.OUT như sau: dòng thứ nhất ghi 1/0 tùy theo có hay không có tour du lịch theo yêu cầu trên. Nếu dòng thứ nhất ghi số 1, dòng thứ hai ghi số C là tổng chi phí của tour được chọn, dòng thứ 3 ghi số k là số thành phố khác nhau trong tour, cuối cùng là k dòng, mỗi dòng ghi tên một thành phố theo trình tự lần lượt đi trong tour.

DULICH.INP	DULICH.OUT	DULICH.INP	DULICH.OUT
5 7	1	4 3	0
1 4 1	61	1 2 10	
1 3 300	4	1 3 20	
3 1 10	1	1 4 30	
1 2 16	3		
2 3 100	5		
2 5 15	2		
5 3 20			

4. Thăm hội chợ (Thi HSG Quốc Gia năm 2000 - Bảng A)

Sơ đồ của một hội chợ gồm N lối vào-ra (được đánh số từ 1 đến N) và các hành lang nối các cặp lối vào-ra. Người ta đã thiết kế sơ đồ này đảm bảo có thể bắt đầu từ một lối vào-ra nào đó du khách đi theo các hành lang có thể đến được một lối vào-ra bất kỳ trong số các lối vào-ra còn lại mà không phải đi qua một hành lang nào quá 1 lần, hơn thế nữa cách đi này là xác định duy nhất. Khi muốn thăm một hành lang hội chợ nào đó du khách có thể phải trả lệ phí nhưng đồng thời lại có thể nhận được một món quà khuyến mại có giá trị của các công ty có gian hàng giới thiệu sản phẩm đặt ở hành lang này.

Yêu cầu hãy xác định cách đi thăm bắt đầu từ một lối vào-ra nào đó và kết thúc cũng tại một lối vào-ra khác, mỗi hành lang không được đi qua quá 1 lần sao cho chi phí phải trả là nhỏ nhất (chi phí phải trả cho một cách đi thăm là hiệu của tổng lệ phí phải trả và tổng giá trị của các món quà khuyến mại nhận được)

Dữ liệu: vào từ file văn bản MINPATH.INP: Dòng đầu tiên chứa số nguyên dương N ($N \leq 1000$). Các dòng tiếp theo mỗi dòng chứa thông tin về một hành lang hội chợ gồm 4 số nguyên s, t, p, q trong đó s và t là các số

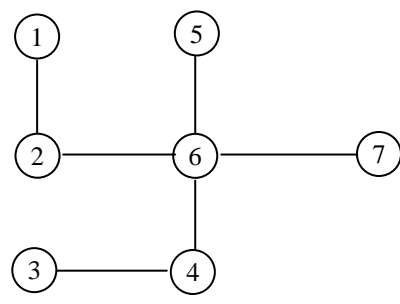
hiệu của các lối vào-ra là đầu mút của hành lang, p là lệ phí phải trả và q là giá trị của món quà nhận được khi đi qua hành lang này. Các giá trị số trên một dòng được ghi cách nhau ít nhất một dấu trắng.

Kết quả: ghi ra tệp văn bản MINPATH.OUT theo cấu trúc sau:

Dòng đầu tiên ghi chi phí theo cách thăm tìm được (chi phí có thể là số âm). Dòng thứ hai ghi cách thăm tìm được là dãy số hiệu của các lối vào-ra trên đường đi tìm được từ lối vào-ra bắt đầu đến lối vào-ra kết thúc. Các giá trị trên mỗi dòng ghi cách nhau ít nhất một dấu trắng.

Ví dụ: Sơ đồ hành lang như trong hình vẽ

MINPATH.INP	MINPATH.OUT
7	-15
1 2 5 20	1 2 6 4
2 6 20 10	
3 4 15 7	
4 6 9 19	
5 6 7 5	
6 7 8 11	



5. Hành trình ngắn nhất

Cho N thành phố (đánh số từ 1 đến N) và M con đường hai chiều nối chúng. Biết độ dài và chi phí đi ô tô của mỗi con đường. Minh đã quá mệt mỏi vì cuộc du lịch vừa trải qua, Minh đang ở thành phố 1 và muốn tìm một hành trình (đi ô tô) ngắn nhất từ thành phố 1 để nhanh chóng về nhà mình tại thành phố N (với cố gắng sao cho chi phí càng ít càng tốt). Bạn hãy giúp Minh tìm một hành trình như thế.

Dữ liệu vào từ file văn bản HTNN.IN

Dòng đầu ghi 2 số nguyên dương N và M ($2 \leq N \leq 100$; $1 \leq M \leq 1500$)

M dòng tiếp theo, mỗi dòng ghi 4 số i, j, L, c thể hiện có con đường nối thành phố i với thành phố j dài là L km và chi phí đi ô tô trên con đường này là c đồng (L và c là các số nguyên không quá 100).

Kết quả ra ghi vào file văn bản HTNN.OUT gồm một dãy liên tiếp số hiệu các thành phố của hành trình tìm được, bắt đầu là 1 và kết thúc là N.

Ví dụ

HTNN . IN		HTNN . OUT
4 4		1 3 4
1 2 3 4		
2 4 6 8		
1 3 5 7		
4 3 4 1		

6. Hai bạn tìm địa điểm gặp nhau

Trước kia Tuấn và Mai là hai bạn cùng lớp, còn bây giờ hai bạn học hai trường khác nhau. Cứ mỗi sáng, đúng 6 giờ cả hai đi từ nhà tới trường của mình theo con đường mất ít thời gian nhất (có thể có nhiều con đường đi mất thời gian bằng nhau và đều ít nhất). Nhưng hôm nay, hai bạn muốn gặp nhau để bàn việc họp lớp cũ nhân ngày 20-11.

Cho biết sơ đồ giao thông của thành phố gồm N nút giao thông được đánh số từ 1 đến N và M tuyến đường phố (mỗi đường phố nối 2 nút giao thông). Vị trí của nhà Mai và Tuấn cũng như trường của hai bạn đều nằm ở các nút giao thông. Cần xác định xem Mai và Tuấn có cách nào đi thoả mãn yêu cầu nêu ở trên, đồng thời họ lại có thể gặp nhau ở nút giao thông nào đó trên con đường tới trường hay không? (Ta nói Tuấn và Mai có thể gặp nhau tại một nút giao thông nào đó nếu họ đến nút giao thông này tại cùng một thời điểm). Nếu có nhiều phương án thì hãy chỉ ra phương án để Mai và Tuấn gặp nhau sớm nhất.

Dữ liệu vào từ file văn bản DOIBAN.INP :

- Dòng đầu tiên chứa 2 số nguyên dương N, M ($1 \leq N \leq 100$);
- Dòng tiếp theo chứa 4 số nguyên dương H_a, S_a, H_b, S_b lần lượt là số hiệu các nút giao thông tương ứng với nhà Tuấn, trường của Tuấn, nhà Mai, trường của Mai.
- Dòng thứ i trong số M dòng tiếp theo chứa 3 số nguyên dương A, B, T . Trong đó A và B là hai đầu của tuyến đường phố i . Còn T là thời gian (tính bằng giây không quá 1000) cần thiết để Tuấn (hoặc Mai) đi từ A đến B cũng như từ B đến A .

Giả thiết là sơ đồ giao thông trong thành phố đảm bảo để có thể đi từ một nút giao thông bất kỳ đến tất cả các nút còn lại.

Kết quả ra file văn bản DOIBAN.OUT

- Dòng thứ nhất ghi YES hoặc NO tùy theo có phương án giúp cho hai bạn gặp nhau hay không. Trong trường hợp có phương án :
- Dòng thứ hai ghi thời gian ít nhất để Tuấn tới trường
- Dòng thứ ba ghi các nút giao thông theo thứ tự Tuấn đi qua
- Dòng thứ tư ghi thời gian ít nhất để Mai tới trường
- Dòng thứ năm ghi các nút giao thông theo thứ tự Mai đi qua
- Dòng thứ sáu ghi số hiệu nút giao thông mà hai bạn gặp nhau
- Dòng thứ bảy ghi thời gian sớm nhất tính bằng giây kể từ 6 giờ sáng mà hai bạn có thể gặp nhau.

Các số trên một dòng của các file Input/output ghi cách nhau ít nhất một dấu cách.

Ví dụ. Với sơ đồ giao thông sau : (N=6, M=7, Ha=1, Sa=6, Hb=2, Sb=5)

DOIBAN.INP	DOIBAN.OUT	
6 7	YES	
1 6 2 5	25	
1 3 10	1 4 6	
1 4 10	30	
2 3 5	2 3 4 5	
3 4 5	4	
3 6 15	10	
4 5 20		
4 6 15		

7. K đường đi ngắn nhất

Vùng đất X có N thành phố ($4 \leq N \leq 800$) được đánh số từ 1 đến N. Giữa hai thành phố có thể có đường nối trực tiếp hoặc không. Các con đường này được đánh số từ 1 đến M ($1 \leq M \leq 4000$). Ban lãnh đạo thể dục thể thao vùng X tổ chức cuộc chạy đua tiếp sức “thông minh” theo quy luật sau :

- + Thành phố xuất phát là thành phố 1, thành phố đích là thành phố N.

+ Mỗi đội thi đấu có K người dự thi. Lần lượt từng người chạy từ thành phố 1 về thành phố N.

+ Khi người thứ nhất đến được thành phố N thì người thứ hai mới bắt đầu rời khỏi thành phố 1, khi người thứ hai đến được thành phố N thì người thứ ba mới bắt đầu rời khỏi thành phố 1, ..., người thứ i đến được thành phố N thì người thứ i+1 mới bắt đầu rời khỏi thành phố 1, ..., (i<K). Người thứ K về tới đích tại thời điểm nào thì thời điểm đó được coi là thời điểm về đích của toàn đội.

+ Đường chạy của các đội viên không được giống nhau hoàn toàn.

+ Có thể chạy lại đoạn đường đã chạy.

Hãy viết chương trình tính thời gian nhỏ nhất để một đội hoàn thành cuộc chạy đua tiếp sức nêu trên nếu các vận động viên có tốc độ chạy như nhau.

Dữ liệu vào từ file văn bản PATHK.IN

- Dòng đầu tiên ghi ba số nguyên K, N, M;
- M dòng tiếp theo, mỗi dòng chứa 3 số nguyên i, j, w thể hiện một đường đi trực tiếp giữa hai thành phố i và j mất thời gian chạy là w (đơn vị thời gian) $1 \leq w \leq 9500$.

Kết quả ra file văn bản PATHK.OUT :

- Dòng thứ nhất chứa một số nguyên duy nhất là thời gian chạy nhỏ nhất của một đội.
- K dòng tiếp theo, mỗi dòng thể hiện hành trình chạy của một vận động viên trong đội là dãy số hiệu các thành phố liên tiếp trên hành trình đó.

Ví dụ

PATHK.IN	PATHK.OUT	Giải thích
4 5 8	23	<i>Đường chạy toàn đội : 23</i>
1 2 1	23	<i>Vận động viên 1 : 1 3 2 5</i>
1 3 2	1 3 2 5	<i>Vận động viên 2 : 1 3 5</i>
1 4 2	1 3 5	<i>Vận động viên 3 : 1 2 1 2 5</i>
2 3 2	1 2 1 2 5	<i>Vận động viên 4 : 1 2 5</i>

2	5	3
3	4	3
3	5	4
4	5	6

1	2	5

8. Tín hiệu giao thông

Bản đồ một thành phố gồm các đường phố thẳng hai chiều :

M đường phố phương Tây-Đông là H_1, H_2, \dots, H_M theo thứ tự từ Bắc xuống Nam.

N đường phố phương Bắc-Nam là V_1, V_2, \dots, V_N theo thứ tự từ Tây sang Đông.

Hai đường phố vuông góc bất kì cắt nhau tạo thành một nút giao thông. Ngoại trừ hai nút giao thông nằm ở vị trí góc Đông-Nam và góc Tây-Bắc, những nút giao thông khác có thể gắn đèn tín hiệu giao thông hai trạng thái :

Trạng thái EW : Xanh cho phương Tây-Đông, Đỏ cho phương Bắc-Nam.

Trạng thái NS : Xanh cho phương Bắc-Nam, Đỏ cho phương Tây-Đông.

Mỗi đèn tín hiệu có một chu kì thời gian riêng, cứ sau mỗi chu kì thời gian đó, đèn đổi trạng thái một lần. Tại thời điểm 0, các đèn tín hiệu đều ở trạng thái 0 (EW).

Để giữ an toàn, luật giao thông qui định : Khi xe theo một phương nào đó tới một nút giao thông đúng vào thời điểm đèn tín hiệu cho phương đó đang Đỏ hoặc chuyển sang Đỏ thì buộc phải dừng lại, đúng vào thời điểm đèn tín hiệu cho phương đó đang Xanh hoặc chuyển sang Xanh thì có thể đi thẳng, rẽ phải hay rẽ trái tùy ý.

Trên một đường phố, thời gian xe đi giữa hai nút giao thông liên tiếp cố định là C đơn vị thời gian.

Yêu cầu : Biết sơ đồ giao thông và các đèn tín hiệu, có hai xe xuất phát cùng thời điểm S, xe thứ nhất xuất phát từ góc Tây-Bắc, xe thứ hai xuất phát từ góc Đông-Nam và hẹn cùng đến một nút giao thông nào đó. Hãy tìm

điểm hẹn và hành trình để hai xe gặp nhau sớm nhất có thể (Xe đến trước có thể chờ xe đến sau tại điểm hẹn).

Dữ liệu vào : từ file văn bản GT.IN: Dòng thứ nhất chứa 4 số tự nhiên : M, N, C, S ($1 \leq M, N, C \leq 100$; $0 \leq S \leq 10000$). M dòng tiếp theo, dòng thứ i chứa N số tự nhiên không quá 100, số thứ j là chu kì của đèn tín hiệu nằm ở giao điểm của đường H_i và V_j . (Qui ước rằng chu kì bằng 0 tương ứng với một nút giao thông không có đèn tín hiệu);

Các số trên một dòng được ghi cách nhau ít nhất một dấu cách.

Kết quả ra file văn GT.OUT: Dòng thứ nhất ghi thời điểm hẹn. Dòng thứ hai ghi một dãy kí tự gồm các kí tự thuộc tập {'E', 'W', 'S', 'N'}, kí tự thứ p cho biết hướng đi từ nút giao thông thứ p đến nút giao thông thứ p+1 trên hành trình của xe thứ nhất. Dòng thứ ba ghi một dãy kí tự gồm các kí tự thuộc tập {'E', 'W', 'S', 'N'}, kí tự thứ q cho biết hướng đi từ nút giao thông thứ q đến nút giao thông thứ q+1 trên hành trình của xe thứ hai.

Ví dụ :

GT . IN	GT . OUT	GT . IN	GT . OUT	
3 4 99 0	297	3 3 99 2	201	
0 1 2 1	SEE	0 1 2	EE	
2 1 2 0	WN	1 2 2	NN	
3 1 2 0		1 1 0		

9. N thang máy. Một toà nhà cao tầng có N thang máy. Mỗi thang máy nối liền đúng 2 tầng với nhau và không dừng lại những tầng nằm giữa hai tầng này. Vận tốc của các thang máy là như nhau : 5 giây qua một tầng.

Thời điểm bắt đầu, mỗi thang máy đều ở tầng thấp và chúng cùng bắt đầu di chuyển lên tầng trên. Sau khi tới tầng trên, ngay lập tức lại chuyển xuống tầng dưới, rồi lại lên tầng trên, cứ như thế ...

Mình đang ở tầng 1 (tầng thấp nhất) và muốn nhanh chóng lên tầng trên cùng của toà nhà. Anh ta thay đổi thang máy chỉ trên những tầng chung của hai thang máy và nếu thang máy kia tại thời điểm này cũng tới tầng này, việc chuyển thang máy khi đó coi như không tốn thời gian nào.

Yêu cầu : Hãy viết chương trình tính thời gian ít nhất mà Minh có thể lên tới tầng trên cùng của toà nhà.

Dữ liệu vào từ file văn bản LIFT.IN :

Dòng đầu tiên chứa hai số nguyên K và N, cách nhau dấu cách, là số tầng và số thang máy của toà nhà ($2 \leq K \leq 1000$, $1 \leq N \leq 50000$).

Trên mỗi một trong N dòng tiếp theo, mô tả một thang máy ghi hai số nguyên A và B, cách nhau dấu cách, ($1 \leq A < B \leq K$), nghĩa là thang máy này di chuyển giữa hai tầng A và B.

Không có hai thang máy nào khác nhau mà lại cùng di chuyển giữa hai tầng như nhau.

Chú ý : **Dữ liệu vào** bảo đảm luôn tồn tại nghiệm.

Kết quả ra file văn bản LIFT.OUT chỉ trên một dòng : ghi thời gian ít nhất (tính theo giây) mà Minh có thể lên tới tầng trên cùng của toà nhà.

Ví dụ .

LIFT . IN		LIFT . IN		LIFT . IN
10 4		10 3		20 5
1 5		1 5		1 7
5 10		3 5		7 20
5 7		3 10		4 7
7 10				4 10
				10 20
LIFT . OUT		LIFT . OUT		LIFT . OUT
45		105		150

10. Robot đi dạo (đề thi chọn HSG 2002-2003)

Công ty VinaBot là một trong các công ty nổi tiếng về chế tạo Rô bốt. Vừa qua, họ đã chế tạo một con Rô bốt mới và đang trong quá trình thử nghiệm. Rô bốt này có khả năng tự vận hành. Điểm đặc biệt là nó có khả năng tự tìm về vị trí xuất phát. Giai đoạn hiện nay Rô bốt đang được huấn luyện để tìm đường đi theo qui luật định sẵn. Sân thử nghiệm Rô bốt là hình

vuông có kích thước $N \times N$ ô, giới hạn bởi điểm có tọa độ $(0, 0)$ ở góc dưới trái và điểm tọa độ (N, N) ở góc trên phải.

Từ điểm thích hợp P (đỉnh xuất phát) thuộc sân thử nghiệm có tọa độ (i, j) ($0 \leq i, j \leq N$) rô bốt phải di chuyển theo một đường gấp khúc kín thoả mãn các điều kiện: Đường gấp khúc có đúng M đỉnh (kể cả đỉnh P). Đường nối giữa hai đỉnh liên tiếp là một đoạn thẳng và không có 3 đỉnh liên tiếp thẳng hàng. Các đỉnh của đường gấp khúc phải nằm trên sân thử nghiệm và có tọa độ nguyên. Kể từ đỉnh P , theo lộ trình chuyển động của Rô bốt, các đoạn thẳng của đường gấp khúc mà rô bốt đi qua phải nhận những giá trị cho trước d_1, d_2, \dots, d_M làm độ dài theo trình tự tương ứng. Rô bốt không đi qua một đỉnh nào của đường gấp khúc quá một lần, trừ đỉnh P nơi nó sẽ tới lần thứ hai khi quay về.

Ban đầu, rô bốt đứng tại điểm có tọa độ $(0, 0)$. Nếu không tồn tại đường gấp khúc khép kín đáp ứng yêu cầu đã nêu, Rô bốt nhấp nháy đèn đỏ và ở nguyên tại chỗ. Trong trường hợp ngược lại, rô bốt tiến thẳng tới đỉnh xuất phát P , bật đèn xanh báo hiệu bắt đầu thực hiện chương trình của mình và chuyển động theo đường gấp khúc tìm được.

Yêu cầu: Hãy giúp rô bốt xác định đỉnh xuất phát P cũng như các đỉnh khác của đường gấp khúc mà rô bốt phải đi theo yêu cầu đặt ra hoặc thông báo rằng không thể tìm được một đường như vậy.

Để đơn giản cho việc tính toán của rô bốt, thay vì cho chiều dài của các đoạn thẳng nối hai đỉnh, người ta sẽ cung cấp cho nó bình phương của các chiều dài này (vì đây sẽ là các giá trị nguyên).

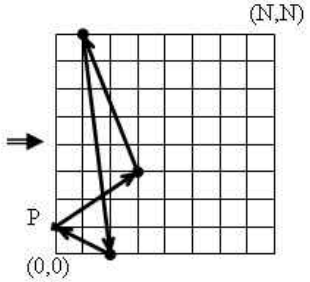
Dữ liệu: Vào từ file văn bản TOUR.INP chứa 2 dòng: Dòng đầu chứa 2 số nguyên N và M ($2 \leq N \leq 15$; $3 \leq M \leq 20$). Dòng thứ hai gồm M số nguyên dương cho biết bình phương chiều dài của các đoạn trên đường gấp khúc đúng theo trình tự mà Rô bốt cần đi qua.

Kết quả: ghi ra file văn bản TOUR.OUT gồm M dòng, mỗi dòng chứa 2 số nguyên là tọa độ các đỉnh của đường gấp khúc tìm được. Các tọa độ được nêu theo trình tự chuyển động của rô bốt, tính từ đỉnh xuất phát P .

Trong trường hợp có nhiều phương án, chỉ cần đưa đúng một trong số đó. Nếu không có lời giải, file sẽ chứa duy nhất số -1.

Trong file **Dữ liệu vào** và file kết quả, các số cách nhau ít nhất một dấu cách

TOUR. INP	TOUR. OUT
8 4 13 29 65 5	0 1 3 3 1 8 2 0
8 4 1 1 1 2	-1



11. Robot cứu hỏa (Đề thi chọn HSG Quốc gia 2007)

Mạng lưới giao thông gồm N nút, giữa một số nút có đường nối hai chiều. Đoạn đường chứa 2 thông tin $t[i,j]$ là thời gian đi đoạn đường, $c[i,j]$ là năng lượng để đi hết đoạn đường. Robot chỉ đi qua được một đoạn đường (i,j) khi năng lượng còn lại trong bình không ít hơn $c[i,j]$. Trên một số nút có trạm tiếp năng lượng, khi robot đến nút này thì được nạp đầy năng lượng. Thời gian nạp năng lượng coi như không đáng kể. Robot xuất phát tại nút 1 với bình năng lượng w đến cứu hỏa tại nút n .

Yêu cầu: Xác định giá trị w nhỏ nhất để robot đi được trên một đường đi từ nút 1 đến nút n trong thời gian ít nhất

Input: Cho trong file ROBOT.INP. Dòng đầu tiên ghi số nguyên dương N ($2 \leq N \leq 500$). Dòng thứ hai ghi N số 1 hoặc 0 thể hiện ở trạm thứ i có hoặc không có trạm tiếp năng lượng. Dòng thứ ba ghi M là số tuyến đường ($M \leq 30000$). Dòng thứ k trong M dòng tiếp ghi thông tin về một tuyến đường gồm 4 số $i, j, t[i,j], c[i,j]$ ($t[i,j], c[i,j] \leq 10^4$)

Output: Ghi ra file văn bản ROBOT.OUT số w tìm được

Ví dụ:

ROBOT. INP	ROBOT. OUT
4	3
0 1 1 0	

5
 1 2 5 4
 1 3 4 3
 1 4 9 4
 2 4 4 1
 3 4 5 2

12. Du lịch khứ hồi

Cho N thành phố đánh số từ 1 đến N . Một người muốn đi du lịch từ thành phố A đến thành phố B và sau đó quay lại A. Người đó muốn rằng trên đường từ B về A sẽ không quay lại những thành phố đã qua trên đường từ A đến B. Hãy tìm cho người đó một hành trình với chi phí ít nhất.

Dữ liệu vào được cho bởi file văn bản DULICH.IN :

Dòng thứ nhất ghi 3 số: Số nguyên dương $N \leq 100$ là số thành phố, hai số nguyên dương $A, B \leq N$ là số hiệu thành phố xuất phát và thành phố cần đến.

Các dòng tiếp theo mỗi dòng ghi 2 số nguyên dương i, j và một số thực k , tượng trưng cho thông tin: Giữa thành phố i và thành phố j có đường đi trực tiếp và chi phí đi quãng đường đó là k .

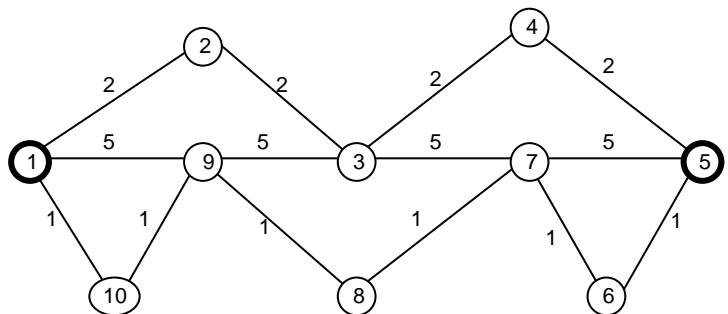
Kết quả ra file văn bản DULICH.OUT trong đó:

Dòng thứ nhất ghi chi phí tổng cộng trên hành trình tìm được;

Dòng thứ hai ghi các thành phố đi qua trên hành trình tìm được theo đúng thứ tự, cách nhau 1 dấu cách.

Nếu không có cách nào đi như vậy thì ghi thông báo 'NO SOLUTION'

Ví dụ



DULICH.IN	DULICH.OUT
10 1 5	14.0000
1 2 2	1 2 3 4 5 6 7 8 9 10 1
2 3 2	
3 4 2	
4 5 2	
5 6 1	
6 7 1	
7 8 1	
8 9 1	
9 10 1	
10 1 1	
1 9 5	
9 3 5	
3 7 5	
7 5 5	

13. Du lịch mỗi ngày M thành phố

Cho N thành phố được đánh số từ 1 đến N ($N \leq 15$). Đường đi một chiều trực tiếp từ i tới j có độ dài là số nguyên không âm D_{ij} , nếu $D_{ij} = 0$ thì coi như không tồn tại đường đi trực tiếp từ i đến j.

Một công ty du lịch ở thành phố 1 có nhiệm vụ đưa khách hàng đi thăm tất cả N thành phố trong P ngày. Mỗi một ngày sẽ xuất phát tại thành phố 1, đi thăm M thành phố ($M \leq 4$) và quay về thành phố 1. Trừ ngày cuối cùng có thể đi ít hơn M thành phố. (P là số nhỏ nhất mà $P \cdot M \geq N$)

Một lịch được gọi là hợp lý nếu như mỗi thành phố được thăm đúng 1 lần (trừ thành phố 1, rõ ràng có thể qua nhiều lần).

Yêu cầu: Câu 1: Chỉ ra một lịch hợp lý. Câu 2: Chỉ ra lịch hợp lý với tổng số quãng đường phải đi là ngắn nhất.

Dữ liệu vào từ file văn bản TP.INP gồm: Dòng đầu tiên là số N và số M. N dòng sau: dòng thứ i, số thứ j trên dòng thứ i đó là D_{ij} .

Kết quả ghi ra file văn bản TP.OUT theo quy định như sau:

Nếu không tồn tại lịch hợp lý thì ghi số 0 tại dòng một, trong trường hợp ngược lại thì ghi những dữ liệu sau: P dòng đầu tiên là lịch hợp lý cho câu 1, với mỗi dòng là lịch đi của một ngày, bắt đầu bằng thành phố 1 và kết

thúc cũng tại thành phố 1. P dòng tiếp theo là lịch hợp lý có tổng khoảng cách nhỏ nhất cho câu 2 như qui cách ghi P dòng trên. Dòng cuối cùng là tổng khoảng cách nhỏ nhất mà khách du lịch phải đi.

TP . INP	TP . OUT
5 2	1 2 3 1
0 1 1 1 1	1 4 5 1
1 0 1 0 1	1 2 3 1
1 1 0 1 0	1 4 5 1
1 0 1 1 1	6
1 0 0 1 1	

14. Olempic

Một đất nước nọ có N địa điểm dân cư đánh số từ 1 đến N và M con đường nối chúng. Các địa điểm dân cư có hai loại: làng và thành phố. Mỗi con đường nối liền hai địa điểm dân cư, và đi trên nó mất Ti phút. Người ta quyết định tiến hành Đại hội Olempic tại một trong các địa điểm dân cư (có thể là thành phố hoặc làng). Những người đưa thư cần chuyển tin từ Đại hội tới các thành phố (mỗi người tới một thành phố).

Hãy viết chương trình cho biết các thời gian của từng người đưa thư tới thành phố họ được phái đến theo thứ tự thời điểm tăng dần. Giả sử rằng trên đường đi, những người đưa thư không nghỉ.

Dữ liệu vào từ file Olemp.in: Dòng đầu tiên, trước hết là 3 số N, M, K tương ứng là số địa điểm dân cư, số con đường và số thành phố. ($2 \leq N \leq 1000$, $1 \leq M \leq 10000$, $1 \leq K \leq N$); tiếp theo là số C đó là số hiệu của địa điểm Đại hội ($1 \leq C \leq N$). Dòng thứ hai là K số là các số hiệu của các thành phố. Tiếp theo là M dòng, mỗi dòng miêu tả một con đường bởi 3 số Si, Ei, Ti tương ứng là số hiệu hai điểm dân cư được nối bởi con đường và thời gian đi trên con đường. ($1 \leq Ti \leq 100$). Bảo đảm từ địa điểm Đại hội luôn tồn tại con đường tới các thành phố (có thể qua một số địa điểm dân cư khác).

Kết quả ra ghi vào file OLEMP.OUT là K cặp số: mỗi cặp ghi số hiệu thành phố và thời gian ít nhất mà người đưa thư có thể tới thành phố này. Các cặp số được sắp tăng theo thời điểm.

OLEMP . IN	OLEMP . OUT
5 4 5 1	1 0
1 2 3 4 5	2 1
1 2 1	3 11
2 3 10	4 111
3 4 100	5 211
4 5 100	
5 5 3 1	2 1
2 4 5	5 1
2 1 1	4 101
2 3 10	
3 4 100	
4 5 100	
1 5 1	

15. Xây cầu vượt biển

Đất nước Delta là quốc đảo lớn trên thế giới. Đất nước gồm N đảo được đánh số từ 1 đến N. Việc đi lại giữa các đảo là rất khó khăn. Vì kinh tế còn rất kém phát triển, nhà nước phải khó khăn lắm mới mở được N – 1 tuyến phà biển để người dân có thể đi lại được giữa hai đảo bất kì. Cách đây không lâu, đất nước mới nhận được sự đầu tư lớn của các nước. Nhà vua quyết định xây mới K cây cầu vượt biển để thay thế cho K tuyến phà. Các cây cầu mới được xây dựng sẽ nối liền hai đảo mà trước đây có tuyến phà nối trực tiếp. Nhà vua muốn chọn K tuyến phà để thay thế bằng cầu sao cho tổng thời gian đi lại giữa mọi cặp đảo khác nhau là nhỏ nhất.

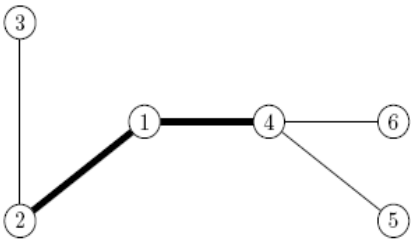
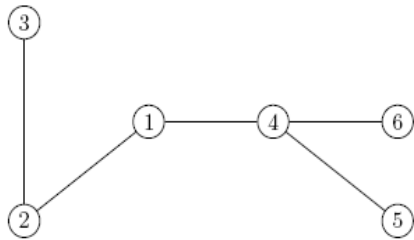
Bạn hãy chọn ra K trong số N – 1 tuyến phà để thay thế bằng cầu, thỏa mãn yêu cầu của nhà vua. Nếu có nhiều phương án tương đương, bạn chỉ cần đưa ra một phương án.

Input BRIDGES.INP: Dòng thứ nhất ghi 4 số nguyên N, K, VP, VC trong đó VP là vận tốc nếu đi bằng phà và VC là vận tốc nếu đi bằng cầu. VP và VC có đơn vị là m/s. N – 1 dòng tiếp theo, dòng thứ I ghi thông

tin về tuyến phà thứ I gồm 3 số U V L có nghĩa tuyến phà thứ I nối hai đảo U, V và khoảng cách giữa U và V là L mét.

Output BRIDGES.OUT: In ra K số là số hiệu của tuyến phà cần được thay thế bằng cầu.

BRIDGES . INP	BRIDGES . OUT
6 2 1 2	1
1 2 5	3
3 2 6	
1 4 4	
4 6 4	
4 5 5	



Các giới hạn:

$$1 \leq K < N \leq 10\,000$$

$$1 \leq V_P, V_C \leq 100\,000$$

$$1 \leq L_{UV} \leq 10^6$$

16. Dàn đèn

Cho một bảng vuông kích thước $m \times n$ được chia thành lưới ô vuông đơn vị, tại mỗi ô của bảng có một trong các ký hiệu:

"." : Ô trống

"+" : Ô có chứa một đèn chưa bật sáng

"*" : Ô có chứa một đèn đã bật sáng

Hai đèn đã bật sáng bất kỳ không nằm cùng hàng hoặc cùng cột.

Yêu cầu: Hãy bật sáng thêm một số nhiều nhất các đèn sao cho: số đèn sáng trên mỗi hàng cũng như trên mỗi cột của bảng tối đa là 1.

Dữ liệu: Vào từ file văn bản GRID.INP: Dòng 1: Chứa hai số m, n ($1 \leq m, n \leq 200$) cách nhau ít nhất một dấu cách; m dòng tiếp theo, dòng thứ i chứa n ký tự liên tiếp, ký tự thứ j là ký hiệu ô (i, j) của bảng.

Kết quả: Ghi ra file văn bản GRID.OUT: Dòng 1: Ghi số đèn có thể bật thêm. m dòng tiếp theo, dòng thứ i ghi n ký tự liên tiếp, ký tự thứ j là ký hiệu ô (i, j) của bảng sau khi đã bật sáng thêm các đèn.

Ví dụ:

GRID.INP	GRID.OUT
4 5	3
+..*.	+..*.
+++.+	*+.+.
.++..	.*+..
.++..	.+*..

17. Phân công hai việc. Có N thợ và M công việc ($N \leq 100, M \leq 200$). Một thợ có thể làm được nhiều việc và một việc có thể do nhiều thợ làm. Hãy phân công N thợ thực hiện M việc đó sao cho mỗi thợ làm đúng hai việc và số việc được thực hiện nhiều nhất, hoặc thông báo không có cách phân công thoả mãn yêu cầu này.

Dữ liệu vào cho trong file văn bản PV.INP :

Dòng đầu là hai số N và M

Dòng thứ i trong N dòng tiếp theo là danh sách việc mà người thợ thứ i thực hiện, các tên việc trong danh sách này cách nhau ít nhất một dấu cách

Kết quả ra file văn bản PV.OUT :

Nếu không có cách phân công thì ghi số 0

Nếu có cách phân công thì file gồm N dòng, mỗi dòng có 3 số t, u, v với ý nghĩa là người thợ t được phân công làm công việc u và v , tương ứng theo thứ tự này.

Dòng thứ $n+1$ của file ghi số K là số việc được thực hiện theo cách phân công này.

Ví dụ

PV . INP	PV . OUT
5 10	1 7
1 3 2 7	3 8
3 6 8	5 4
5 4 2 10	2 9
1 2 3 9	2 6
2 6	9

PV . INP	PV . OUT
5 6	0
3	
6 8	
3	
1 2 9	
2 6	

18. Domino. (Bài 2, IOI 2005).

Cho một bàn cờ kích thước $n \times m$ ô vuông. Có một vài đoạn thẳng kẻ trên nó. Các đoạn thẳng này kẻ song song với hai cạnh kề của bàn cờ. Bạn sẽ đặt những quân domino trên bàn cờ. Mỗi quân domino sẽ phủ hai ô vuông kề nhau. Bạn có thể đặt một quân domino lên 2 ô kề nhau chỉ khi chúng không bị ngăn cách bởi một đoạn thẳng đã kẻ nào. Nhiệm vụ của bạn là sắp đặt các quân domino trên bàn cờ sao cho mỗi ô vuông được phủ bởi đúng một quân domino. Bạn có thể giả thiết rằng mỗi **Dữ liệu vào** đều có nghiệm.

Nhiệm vụ

Mỗi test cần xuất một output. Điều đó có nghĩa là, bạn được cho các file input `dom1.in`, `dom2.in`, ..., `dom20.in` và chương trình của bạn cần xuất ra các file output `dom1.out`, `dom2.out`, ..., `dom20.out`. Bạn sẽ nộp một file nén kiểu zip hoặc tar-gzip lưu trữ tất cả các file output, không có thư mục con nào trong nó. Dạng của các file input và các file output do bạn xuất ra, được miêu tả ở dưới.

Input

Dòng đầu tiên của file input là 2 số nguyên n và m cách nhau đúng một dấu cách – n là số dòng và m là số cột của bàn cờ, $1 \leq n, m \leq 100$, $n \cdot m$ chẵn. Các ô vuông của bàn cờ được đánh số từ 1 đến $n \cdot m$. Ô thứ i (kể từ trái) thuộc dòng thứ j (kể từ trên xuống) có số hiệu là $(j-1) \cdot m + i$ ($1 \leq i \leq m$, $1 \leq j \leq n$).

Dòng thứ hai gồm một số nguyên dương L ($0 \leq L \leq 5000$) là số các đoạn thẳng kẻ trên bàn cờ. Mỗi dòng trong L dòng tiếp theo chứa 2 số nguyên cách nhau đúng một dấu cách. Dòng thứ $i+2$ chứa hai số nguyên p_i và q_i (với

Trên mặt phẳng toạ độ có N robot được đánh số liên tục từ 1 đến N , đứng tại các điểm tương ứng là $A_1(x_1, y_1)$, $A_2(x_2, y_2)$, ..., $A_N(x_N, y_N)$ và N hình chữ nhật H_1, H_2, \dots, H_N . Hình chữ nhật có số hiệu i là hình H_i ($i=1, 2, \dots, n$) với toạ độ đỉnh trái-dưới là (x_{i1}, y_{i1}) và toạ độ đỉnh phải-trên là (x_{i2}, y_{i2}) .

Yêu cầu điều khiển N robot đồng thời chuyển động, cùng tốc độ, theo các đường ngắn nhất, sao cho mỗi robot chiếm được một hình chữ nhật và thời điểm robot cuối cùng chiếm được hình chữ nhật là sớm nhất. Robot chiếm được hình chữ nhật nếu nó chuyển động được tới một điểm thuộc hình chữ nhật đó.

Dữ liệu vào từ file văn bản ROBOT.IN :

Dòng đầu là số N ($1 \leq N \leq 100$)

Dòng thứ i trong N dòng tiếp theo mô tả hình chữ nhật H_i gồm 4 số $x_{i1}, y_{i1}, x_{i2}, y_{i2}$ ($x_{i1} < x_{i2}$, $y_{i1} < y_{i2}$);

Dòng thứ j trong N dòng còn lại mô tả vị trí ban đầu của robot j đó là hai số x_j, y_j .

Kết quả ra file văn bản ROBOT.OUT : chỉ một dòng ghi N số, số thứ j là số hiệu hình chữ nhật mà robot j chiếm được.

Hạn chế kỹ thuật : Các toạ độ nêu trong file input có giá trị tuyệt đối không vượt quá 1000.

Ví dụ

ROBOT.IN

2

1 1 3 2

2 0 4 2

0 1.5

0 0

ROBOT.OUT

1 2

20. Tuyển đường mới

Đường tàu hoả Bắc-Nam nối Hà Nội với thành phố Hồ Chí Minh, đi qua một số ga ở các tỉnh, thành. Các ga đánh số hiệu lần lượt từ Bắc vào Nam, bắt đầu từ ga Hà Nội có số hiệu là 1.

Yêu cầu: Lập kế hoạch (nếu được) để xây dựng K tuyến đường ô tô mới cần bổ sung, mỗi tuyến đi qua 2 ga cho trước thoả mãn: Từng đôi một không cắt nhau và không có tuyến nào cắt đường tàu.

Dữ liệu vào cho trong file văn bản TUYENMOI.INP: Dòng đầu ghi N, K tương ứng là số lượng ga tàu và số lượng tuyến đường ô tô mới cần bổ sung ($1 \leq N, K \leq 20000$). Mỗi dòng trong K dòng tiếp theo ghi số hiệu hai ga cần mở tuyến đường nối trực tiếp.

Kết quả: Ghi ra file văn bản TUYENMOI.OUT:

Dòng đầu ghi 0 hoặc 1 tương ứng là không thể hoặc có thể thiết kế các tuyến đường mới cần bổ sung.

Với trường hợp dòng đầu là 1 thì mỗi dòng trong K dòng tiếp theo ghi hoặc W hoặc E chỉ tương ứng tuyến đường cần xây dựng ở phía Tây hoặc phía Đông của đường tàu hoả.

Ví dụ:

TUYENMOI . INP	TUYENMOI . OUT
8 7	1
1 2	E
1 3	E
2 3	W
5 7	W
4 8	W
7 8	E
6 8	E

21. Xếp lịch dạy học (Đề thi chọn đội tuyển Quốc gia năm 1995)

Trong một trường đại học có M thầy giáo, đánh số từ 1 đến M và N lớp học, đánh số từ 1 đến N. Với $1 \leq i \leq M$, $1 \leq j \leq N$, thầy giáo i phải dạy cho lớp học j là $P(i,j)$ ngày, $P(i,j)$ là số nguyên trong hoảng từ 0 đến 10. Trong mỗi ngày, mỗi thầy không dạy hơn một lớp và mỗi lớp không học hơn một thầy. Hãy xếp lịch cho các thầy giáo sao cho toàn bộ yêu cầu giảng dạy trên được hoàn thành trong ít ngày nhất. Các ngày trong lịch dạy được đánh số là 1, 2, 3,

Dữ liệu: Vào từ file văn bản TEACH.INP

Dòng đầu tiên ghi hai số M, N ($M \leq 20$, $N \leq 20$)

Dòng thứ $i+1$ ($1 \leq i \leq M$) lần lượt ghi các giá trị $P(i,1)$, $P(i,2)$, ..., $P(i,n)$

Kết quả: Ghi ra file văn bản TEACH.OUT

Dòng thứ nhất ghi K là tổng số ngày hoàn thành chương trình

Trong K dòng tiếp theo, dòng thứ i ghi N số là tên của thầy giáo dạy trong N lớp của ngày thứ i. Ghi số 0 nếu hôm đó lớp tương ứng nghỉ học

Ví dụ:

TEACH.INP	TEACH.OUT
5 4	4
2 0 0 0	4 2 3 0
0 1 1 0	1 4 2 0
1 0 1 0	3 0 4 5
1 1 1 1	1 0 0 4
0 0 0 1	

22. Stalker (Nhà thám hiểm)

Trong thành phố H có một khu không có bản đồ lãnh địa rõ ràng nên trở thành một vùng không bình thường. Tất cả các lối vào khu này đều bị che phủ (chặn, cắt) và chính vì thế người ta gọi nó là một mê cung. Trong khu có N ngôi nhà, chúng được nối liền bởi các con đường. Trên mỗi con đường có thể đi theo cả hai chiều. Một nhà thám hiểm tập sự được giao nhiệm vụ vào một nhà kho trong mê cung này. Anh ta dựa vào thư viện điện tử để tìm một số bản đồ của mê cung. Nhưng bản đồ được thành lập bởi nhiều người khác nhau nên trên mỗi bản đồ chỉ có thông tin về một số con đường trong mê cung. Một con đường cũng có thể tồn tại trên vài bản đồ. Trên đường đi, nhà thám hiểm có thể tải một bản đồ từ thư viện về điện thoại di động của mình. Khi tải về bản đồ mới, các bản đồ trước trong bộ nhớ điện thoại di động không còn tồn tại. Nhà thám hiểm chỉ có thể di chuyển trên những con đường đã đánh dấu trên bản đồ vừa tải trong thời điểm đã cho. Mỗi lần tải bản đồ phải trả 1 (nghìn đồng). Để chi phí ít nhất, cần chọn các con đường sao cho số lần tải bản đồ về ít nhất. Có thể tải về một và chỉ một bản đồ vài lần, nhưng phải trả tiền cho mỗi lần tải. Ban đầu trong bộ nhớ của máy di động không có một bản đồ nào. Cần viết một

chương trình tính tổng chi phí ít nhất cần thiết cho nhà thám hiểm đi từ lối vào mê cung đến nhà kho.

Dữ liệu vào từ file STALKER.IN

Dòng thứ nhất chứa hai số nguyên N và K ($2 \leq N \leq 2000$; $1 \leq K \leq 2000$) tương ứng là số toà nhà trong mê cung và số bản đồ. Lối vào mê cung là vị trí trong ngôi nhà số 1, nhà kho trong ngôi nhà số N . Trong các dòng tiếp theo là thông tin về các bản đồ. Dòng đầu tiên của bản đồ thứ i là số r_i là số con đường trên bản đồ này. Sau đó là r_i dòng, mỗi dòng chứa hai số tự nhiên a và b ($1 \leq a, b \leq N$; $a \neq b$). Tổng số lượng các con đường trên các bản đồ không quá 300000 ($r_1 + r_2 + \dots + r_K \leq 300000$).

Kết quả ra ghi vào file STALKER.OUT tổng chi phí nhỏ nhất cho hành trình của nhà thám hiểm. Trong trường hợp không thể tới được nhà kho thì ghi -1.

Ví dụ

STALKER.IN	STALKER.OUT		STALKER.IN	STALKER.OUT
5 3 1 3 4 3 1 2 1 3 2 4 1 4 5	2		5 3 2 3 2 4 5 1 2 1 2 1 3 5 4	-1

23. Matching bins. (BOI 2010) Có một số lớn các thùng rỗng trong một kho của nhà máy. Các thùng được xếp chỉ trên một hàng. Người quản lý kho muốn đặt một số thùng trong các thùng khác để tạo khoảng trống nào đó ở phía trái của kho. Các thùng có thể được chuyển bằng một rô bốt, nó chọn

thùng, mang về phía phải và đặt nó vào thùng rộng hơn. Dãy ba thao tác này là cách duy nhất cho phép chuyển các thùng.

Vì qui tắc an toàn, một thùng bất kỳ có thể chứa không quá một thùng khác. Người quản lý cũng muốn giữ các thùng đúp ở bên trái hàng kết quả để dễ dàng quản lý chúng.

Bạn viết một chương trình tính số K lớn nhất có thể mà K thùng bên trái nhất có thể đặt trong K thùng ngay sau theo một thứ tự nào đó.

Input. Dòng đầu tiên của tệp văn bản BINS.IN chứa hai số nguyên cách nhau một dấu trống: M ($1 \leq M \leq 1000$), là kích thước của thùng rộng nhất, và N ($1 \leq N \leq 20000$) là số lượng thùng. Dòng thứ hai chứa N số nguyên A_i ($1 \leq A_i \leq M$), chúng cách nhau bởi các dấu trống: là kích thước các thùng liệt kê từ trái qua phải.

Output. Dòng đầu tiên và cùng là dòng duy nhất của tệp BINS.OUT chứa đúng một số nguyên là số K lớn nhất mà rõ bất có thể đặt K thùng bên trái nhất đặt vào K thùng tiếp theo.

Ví dụ.

BINS.IN	BINS.OUT
5 10	4
2 2 1 4 3 2 5 4 2 3	

MỤC LỤC

1. Phương pháp tìm kiếm theo chiều sâu	3
2. Tìm số thành phần liên thông.	3
3. Tìm cây khung, cây khung ngắn nhất	7
4. Tìm đường đi ngắn nhất trên đồ thị có trọng số	14
5. Chu trình	17
6. Bộ ghép trên đồ thị hai phía.....	18
7. Luồng cực đại.....	28
8. Bài tập minh họa	39
Bài 1. Mạng vòng	39
Bài 2. Khám phá mê cung	43
Bài 3. Bin – Các thùng nước	47
Bài 4. Trong vườn trẻ	49
Bài 5. Tô màu bản đồ.....	53
Bài 6. Comnet (Đề thi chọn HSG giỏi 12/3/2003, Bài 1)	57
Bài 7. HIGHWAY - Hệ thống đường cao tốc	61
Bài 8. Olymnet - Nối mạng máy tính.....	66
Bài 9. Kiểm tra dữ liệu	71
Bài 10. Mạng máy tính (Thi HSG Quốc Gia 2006 - Bảng A)	75
Bài 11. Sói và cừu.....	79
Bài 12. Thỏ và cà rốt.....	83
Bài 13. Công chúa kén chồng (ZAM).....	87
Bài 14. Mạng trường học (IOI 1996 - Network of Schools)	91
Bài 15. Kênh xung yếu.....	96
Bài 16. Khám phá mê cung	100
Bài 17. Roboco	104
Bài 18. Xếp công việc.....	106
Bài 19. Thi robocon.	109
Bài 20. Trò chơi bi nhảy	115
Bài 21. Brick.....	121
Bài 22. Bridges - Xây dựng cầu	127
Bài 23. Đặt trạm cứu hỏa.	133
Bài 24. Đường mòn và những con bò (IOI 2003 , Trail Maintenance)	136
Bài 25. Jointst - Ghép khâu ký tự.....	140
Bài 26. SightseeingTrip	147
Bài 27. Đường truyền tin	152
Bài 28. D'Artagnan.....	156
Bài 29. Network - Mạng truyền tin	161
Bài 30. MOVILAB (mê cung chuyển dịch).....	166

Bài 31.	Thành phố trên sao hoả	170
Bài 32.	JOURN - Hành trình chẵn lẻ.....	174
Bài 33.	Toll - Lệ phí thấp nhất	178
Bài 34.	Hai thang máy	182
Bài 35.	Trains	186
Bài 36.	Hike - Cuộc hành quân dã ngoại.....	191
Bài 37.	Build - Xây dựng đường	196
Bài 38.	Walls (IOI 2000)	200
Bài 39.	Hội chợ	205
Bài 40.	Mê cung (Thi HSG Quốc Gia 2004 - Bảng A)	212
Bài 41.	Roads - Chia tay	216
Bài 42.	Cưỡi lạc đà thăm K thành phố.....	220
Bài 43.	Robot cứu hỏa (Thi HSG Quốc gia 2007)	227
Bài 44.	Lăn xúc xắc	232
Bài 45.	Supersch - Đội tuyển có thành tích cao nhất.....	240
Bài 46.	Thi bắn súng (Sho)	245
Bài 47.	Guards (Bố trí các lính gác)	248
Bài 48.	Sắp xếp hoa trong triển lãm	254
Bài 49.	Minimum path cover (Tập phủ đường tối thiểu).....	257
Bài 50.	Minimum vertex cover (Tập phủ đỉnh tối thiểu)	261
Bài 51.	Maximum independent set (Tập độc lập cực đại)	266
Bài 52.	Dự đoán	273
Bài 53.	Tiếp thị	277
Bài 54.	Phỏng vấn.....	282
Bài 55.	Hai ô đen (Đề thi HSG năm 1995)	286
Bài 56.	Kho xăng	292
Bài 57.	Numway (Di chuyển quân tốt)	297
Bài 58.	Bảo vệ thành phố	303
Bài 59.	Patrol (Tuần tra)	312
Bài 60.	Chia cắt dịch	322
8. Bài tập tự giải.....		328