CHUYÊN ĐỀ MÔN TIN HỌC
ĐƯỜNG ĐI NGẮN NHẤT TRÊN ĐỒ THỊ

Trong các bài toán về Tìm đường đi như đường đi Euler, đường đi Hamltion thì bài toán Tìm đường đi ngắn nhất giữa các cặp đỉnh trong đồ thị có nhiều ứng dụng trong thực tế cũng như trong các kỳ thi chọn học sinh giỏi hằng năm. Bài toán Tìm đường đi ngắn nhất có rất nhiều thuật toán để giải bài toán này, tuy nhiên có 3 thuật toán quen thuộc trong chương trình chuyên Tin: Thuật toán **Floyd** (Khoảng cách mọi cặp đỉnh), thuật toán **Ford-Bellman** (Khoảng cách từ một đỉnh đến các đỉnh còn lại), thuật toán **Dijkstra** (khoảng cách từ một đỉnh đến các đỉnh còn lại). Với thời gian, kiến thức và kinh nghiệm có hạn tôi chỉ nghiên cứu thuật toán **Dijkstra** (1959) Tìm đường đi ngắn nhất giữa hai cặp đỉnh trong đồ thị, được biên tập và tổng hợp từ nhiều nguồn tài liệu.

I. GIỚI THIỆU BÀI TOÁN

Bài toán: Cho **G=(V, E)** là đơn đồ thị gồm **n** đỉnh và **m** cạnh, trọng số trên các cạnh **không âm**. Yêu cầu tìm đường đi ngắn nhất từ đỉnh xuất phát **s ∈ V** đến đỉnh đích **t ∈ V**.

- Trong trường hợp đồ thị không có trọng số thì thuật toán **BFS** để tìm đường đi ngắn nhất cho ta thời gian tuyến tính **O(n+m)**. Nếu đồ thị dầy thuật toán **Dijkstra** biểu diễn đồ thị bằng ma trận kề thì thời gian thực hiện thuật toán **O(n^2)**, nếu đồ thị thưa thì thuật toán **Dijkstra** biểu diễn đồ thị bằng danh sách kề kết hợp với **Heap** nhị phân thì thời gian thực hiện thuật toán là **O((n+m)logn)**, **Dijkstra** với **Fibonacii Heap** thì thời gian thực hiện thuật toán là **O(nlogn+m)**. **Dijkstra** với **Fibonacii** có thời gian tiệm cận khá tốt tuy nhiên ít được dùng vì cài đặt phức tạp và tìm ẩn nhiều suy biến với các bộ dữ liệu lớn. Trong thuật toán **Dijkstra Heap** nhị phân ta có thể cải tiến kết hợp vớ **Heap** tam phân, tứ phân, ...Ý tưởng dùng heap **k** phân (**k= m div n**) được **Johson** đề xuất năm 1977 ta thu được thời gian thực hiện thuật toán **O(mlog$_k$n)**, với đồ thị thưa thì thời gian là **O(nlogn)**, với đồ thị dầy thì thời gian là **O(m)** (theo chỉ mục tài liệu tham khảo [5] và [6]).

- Thuật toán **Dijkstra** dựa trên nguyên lí tối ưu của **Bellman** (1953): "**P** là đường đi ngắn nhất từ **s** đến **t** (**s** <> **t**), **k** là một đỉnh trên **P** thì **sk**+**kt** cũng là đường đi ngắn nhất".

Với mục đích biên soạn tài liệu đi từ cơ bản đến mở rộng, các chương trình sau đây tôi xin trình bày và lựa chọn phương án cài đặt sao cho đơn giản, dễ hiểu nhằm nêu bật được bản chất của thuật toán.

II. THUẬT TOÁN DIJKSTRA:

- Ý tưởng: Tìm đường đi từ **s** đến **t**, nếu đi từ đỉnh **s** đến đỉnh **k** và từ đỉnh **k** đến đỉnh **t** ngắn hơn đường đi từ đỉnh **s** đến đỉnh **t** thì ta chọn đường đi qua đỉnh trung gian **k**. Trong các đỉnh trung gian thì chọn đỉnh trung gian có nhãn bé nhất và đánh dấu đỉnh này để lần sau không chọn nữa.

Với đỉnh **t** ∈ V, ta gọi D[t] là độ dài đường đi ngắn nhất từ đỉnh **s** đến đỉnh **t**, thuật toán dùng kĩ thuật đánh dấu, nếu đỉnh có giá trị **False** (đỉnh tự do) là đỉnh có thể tối ưu, có giá trị **True** (đỉnh cố định) là đỉnh không thể tối ưu, nếu sau khi thực thi thuật toán nhãn đỉnh **t** được cố định thì ta có đường đi ngắn nhất từ **s** tới **t**, ngược lại không có đường đi từ đỉnh **s** đến đỉnh đến đỉnh **t.**

- Cấu trúc dữ liệu thuật toán Dijkstra cổ điển:

+ Mảng hai chiều **A**: Biểu diễn đồ thị (biểu diễn đồ thị bằng ma trận kề)
+ Mảng một chiều **Ch**: Đánh dấu đỉnh tự do hay đỉnh cố định.
+ Mảng một chiều **D**: Mảng khoảng cách.
+ Mảng một chiều **Tr**: Lưu vết đường đi.

1. THUẬT TOÁN DIJKSTRA CỔ ĐIỂN:

Bước 1: Khởi tạo

- Khởi trị mảng khoảng cách **D[s]=0, D[i]=+** ∞ (∀**i≠ s**; i=1..n).

- Khởi trị mảng đánh dấu **Ch[i]=False** (∀i=1..n**).**

Bước 2: Lặp

Bước lặp gồm hai thao tác:

B2.1. Cố định nhãn: Tìm đỉnh **u** là nhãn tự do có **D[u]** nhỏ nhất và cố định đỉnh **u.**

B2.2. Sửa nhãn: Dùng đỉnh **u** tìm được để xét tất cả các đỉnh **v** tự do kề u và sửa lại các nhãn **D[v]** theo công thức sau:

D[v]= min (D[v], D[u]+A[u,v])

Lưu vết đường đi **Tr[v]=u**;

Bước lặp sẽ kết thúc khi đỉnh **t** (đỉnh đích) đã được cố định nhãn.

Bước 3: Kết hợp với lưu vết đường đi trên từng bước sửa nhãn, thông báo "độ dài", đường đi ngắn nhất tìm được hoặc cho biết không tồn tại đường đi (D[t]=+ ∞).

Mã Pascal tham khảo:

```
Procedure DTra(s, t: longint);
Var i,j, u, v, min: longint;
Begin
    //B1, Khoi tao
    For i:=1 To n Do
            Begin
                  D[i]:=MaxInt;
                  Ch[i]:=False;
            End;
    D[s]:=0;
    //B2. Lap
    For i:=1 To n Do // có thể dùng Repeat ..Until
      Begin
            //B2.1 Tim dinh u tự do co D[u] nho nhat
            Min:=MaxInt;
            For j:=1 To n Do
```

```
                If (D[j]< Min) And (Ch[j]=False) Then
                       Begin
                            Min:=D[j];
                            u:=j;
                       End;
            Ch[u]:=True; // if u=0 Then Break;
            If u = t Then Exit;
             // B2.2 Sua nhan;
             For v:=1 To n Do
                  If (ch[v]=False) And (a[u,v]<>0) And (D[v]> D[u]+A[u,v]) Then
                            Begin
                                  D[v]:=D[u]+A[u,v];
                                  Tr[v]:=u;
                            End;
      End;
End;
```

Truy vết đường đi từ đỉnh s đến đỉnh t: Dựa vào mảng lưu vết **Tr** ta truy vết ngược từ đỉnh **t** về đỉnh **s** và đưa vào ngăn xếp **P** mã Pascal như sau:

```
Procedure TruyVet;
Var top, k: LongInt;
       P: array[1..nmax] of LongInt;
Begin
         If D[t]=MaxInt Then Write(0)
         Else
              Begin
                  k:=t; top:=0;
                  While k<>0 Do
                        Begin
                             top:=top+1;
                             P[top]:=k;
                             k:=Tr[k];
                         End;
                  Writeln(D[t]);
                  For k:=top DownTo 1 Do Write(P[k],' ');
             End;
End;
```

- Để tìm đường đi dài nhất từ đỉnh **s** đến đỉnh **t** ta chỉ cần đổi dấu ma trận biểu diễn đồ thị thì đường đi dài nhất từ đỉnh **s** đến đỉnh **t** là ABS(D[t]).

- Thuật toán Dijkstra có thể áp dụng cho đa đồ thị có hướng hay vô hướng có trọng số không âm.

Bài 1: ÔNG NGÂU BÀ NGÂU

Hẳn các bạn đã biết ngày "ông Ngâu bà Ngâu" hàng năm, đó là một ngày đầy mưa và nước mắt. Tuy nhiên, một ngày trước đó, nhà Trời cho phép 2 "ông bà" được đoàn tụ. Trong vũ trụ vùng thiên hà nơi ông Ngâu bà Ngâu ngự trị có **n** hành tinh đánh số từ 1 đến **n**, ông ở hành tinh Adam (có số hiệu là s) và bà ở hành tinh Eva (có số hiệu là **t**). Họ cần tìm đến gặp nhau. **n** hành tinh được nối với nhau bởi một hệ thống cầu vồng, hai hành tinh bất kỳ chỉ có thể không có hoặc duy nhất một cầu vồng (hai chiều) nối giữa chúng. Họ luôn đi tới mục tiêu theo con đường ngắn nhất. Họ đi với tốc độ không đổi và nhanh hơn tốc độ ánh sáng. Điểm gặp mặt của họ chỉ có thể là tại một hành tinh thứ 3 nào đó.

Yêu cầu: Hãy tìm một hành tinh sao cho ông Ngâu và bà Ngâu cùng đến đó một lúc và thời gian đến là sớm nhất. Biết rằng, hai ngươi có thể cùng đi qua một hành tinh nếu như họ đến hành tinh đó vào những thời điểm khác nhau.

Dữ liệu vào: Trong tệp văn bản OBNGAU.INP gồm:

- Dòng thứ nhất là 4 số **n, m, s, t** ($2< n \le 10^3$, $1 \le s \ne t \le n$), m là số cầu vồng ($m \le 10^4$).

- Tiếp theo là **m** dòng, mỗi dòng gồm ba số i, j, l thể hiện có cầu vồng nối giữa hai hành tinh i , j có độ dài là l ($1 \le i \ne j \le n$, $0 < l \le 200$).

Kết quả: Ghi ra file văn bản OBNGAU.OUT, do tính chất cầu vồng, mỗi năm một khác, nên nếu như không tồn tại hành tinh nào thoả mãn yêu cầu thì ghi ra một dòng chữ CRY. Nếu có nhiều hành tinh thoả mãn thì ghi ra hành tinh có chỉ số nhỏ nhất.

Ví dụ:

OBNGAU.INP	OBNGAU.OUT
4 4 1 4 1 2 1 2 4 1 1 3 2 3 4 2	2

Tư tưởng thuật toán: *Chúng ta có một số nhận xét sau:*

+ Hai hành tinh bất kì chỉ được nối đến nhau bởi nhiều nhất một cầu vồng.

+ Ông Ngâu và bà Ngâu luôn đi tới mục tiêu theo con đường ngắn nhất.

+ Họ đi với vận tốc không đổi và nhanh hơn vận tốc ánh sáng.

Thực chất đây là một bài toán đơn đồ thị vô hướng, có số đỉnh là **n** (số hành tinh), số cạnh là **m** (số cầu vòng).

+ Từ hành tinh **s** (nơi ông Ngâu ở) ta xây dựng mảng khoảng cách **SD**. Trong đó SD[i] là đường đi ngắn nhất từ hành tinh **s** đến hành tinh i (do ông Ngâu luôn đi tới mục tiêu theo con đường ngắn nhất).

+ Tương tự ta sẽ xây dựng bảng T**D**, trong đó TD[i] là đường đi ngắn nhất từ hành tinh **t** đến hành tinh **i.**

Do yêu cầu của bài toán là tìm hành tinh khác S và T mà 2 ông bà Ngâu cùng đến một lúc và trong thời gian nhanh nhất. Tức là ta tìm hành tinh h sao cho (h khác S và T) và SD[h] = TD[h] đạt giá trị nhỏ nhất khác 0. Nếu không có hành tinh h nào thoả mãn thì thông báo CRY

Với ràng buộc dữ liệu đã cho đủ nhỏ để áp dụng thuật toán **Dijkstra** cổ điển để tìm đường đi ngắn nhất giữa 1 đỉnh đến các đỉnh của đồ thị. Bài giải sau đây áp dụng hoàn toàn giống thuật toán **Dijkstra** đã trình bày ở mục 2.1:

```
Program ongbangau;
const fi='OBNGAU.INP';
      fo='OBNGAN.OUT';
      nmax=Trunc(1e3);

Var a: array[1..nmax, 1..nmax] of longint;
```

```
    d, SD, TD: array[1..nmax] of longint;
    ch: array[1..nmax] of Boolean;
    n, m, u, v, s, t, res: longint;
    f: text;

Procedure doc;
Var i,x, y, z: longint;
Begin
     assign(f,fi); reset(f);
     read(f, n, m, s, t);
     for i:=1 to m do
        begin
           read(f, x, y, z);
           a[x,y]:=z;
           a[y,x]:=z;
        end;
     close(f);
end;
Procedure Dtra(s: longint);
Var i,j, u, v, min: longint;
Begin
    For i:=1 To n Do
        Begin
            D[i]:=MaxInt;
            Ch[i]:=False;
        End;

    D[s]:=0;
    For i:=1 To n Do
       Begin

           Min:=MaxInt;
           For j:=1 To n Do
               If (D[j]< Min) And (Ch[j]=False) Then
                  Begin
                       Min:=D[j];
                       u:=j;
                  End;

            Ch[u]:=True;

            For v:=1 To n Do
                If (ch[v]=false) and (a[u,v]<>0) And (d[v]>d[u]+a[u,v]) Then

                       D[v]:=D[u]+A[u,v];
           End;
End;
Procedure  TryVet;
Var mmin, k: LongInt;
Begin
      mmin:=MaxInt;

      For k:=1 to n do
           If (SD[k]=TD[k]) and (SD[k] <> 0) And (SD[k]< MMin) then
            Begin
               MMin:=SD[k];
               Res:=k;
            End;
```

```
End;

Begin
    Doc;
    DTra(s);
    SD:=D;
    DTra(t);
    TD:=D;
    TryVet;
    Assign(f, fo); Rewrite(f);
    If res=0 Then Write(f, 'CRY') Else Write(f, res);
    close(f);
End.
```

Bài 2: ĐÔI BẠN

Trước kia Tuấn và Mai là hai bạn cùng lớp còn bây giờ hai bạn học khác trường nhau. Cứ mỗi sáng, đúng 6 giờ cả hai đều 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 nhà của 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 FRIEND.INP

- Dòng đầu tiên chứa 2 số nguyên dương N, M ($1 \le N \le 100$);
- Dòng tiếp theo chứa 4 số nguyên dương Ha, Sa, Hb, Sb 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 ≤ 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.

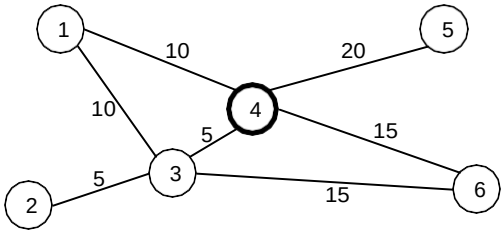
***Kết quả* :** ghi ra file văn bản FRIEND.OUT

- Dòng 1: Ghi từ YES hay NO tuỳ theo có phương án giúp cho hai bạn gặp nhau hay không. Trong trường hợp có phương án:
- Dòng 2: Ghi thời gian ít nhất để Tuấn tới trường
- Dòng 3: Ghi các nút giao thông theo thứ tự Tuấn đi qua
- Dòng 4: Ghi thời gian ít nhất để Mai tới trường
- Dòng 5: Ghi các nút giao thông theo thứ tự Mai đi qua
- Dòng 6: Ghi số hiệu nút giao thông mà hai bạn gặp nhau

- Dòng 7: Thời gian sớm nhất tính bằng giây kể từ 6 giờ sáng mà hai bạn có thể gặp nhau.

Ví dụ : Với sơ đồ giao thông sau: (N=6,M=7, Ha=1, Sa=6, Hb=2, Sb=5)

Dòng	FRIEND.INP	FRIEND.OUT
1	**6 7**	**YES**
2	**1 6 2 5**	**25**
3	**1 3 10**	**1 4 6**
4	**1 4 10**	**30**
5	**2 3 5**	**2 3 4 5**
6	**3 4 5**	**4**
7	**3 6 15**	**10**
8	**4 5 20**	
9	**4 6 15**	



Tư tưởng thuật toán: Sử dụng thuật toán Dijkstra, xây dựng thủ tục: DTra(s:integer, var d: mảng_khoảng cách); để xây dựng mảng d cho đường đi ngắn nhất từ điểm xuất phát s đến mọi đỉnh (có thể tới). Sau đó gọi thủ tục này 4 lần bằng các lời gọi:

- DTra(ha,d1); sẽ được d1 cho biết các đường đi ngắn nhất xuất phát từ nhà Tuấn
- DTra(sa,d2); sẽ được d2 cho biết các đường đi ngắn nhất xuất phát từ nhà Mai
- DTra(hb,d3); sẽ được d3 cho biết các đường đi ngắn nhất xuất phát từ trường Tuấn
- DTra(sb,d4); sẽ được d4 cho biết các đường đi ngắn nhất xuất phát từ trường Mai

Điểm hẹn là nút u cần thỏa mãn các điều kiện sau:

d1[u] + d3[u]=d1[sa] {thời gian Tuấn đi từ nhà tới điểm hẹn + Tuấn}

d2[u] + d4[u] = d2[sb] {thời gian Mai đi từ nhà tới điểm hẹn + từ điểm hẹn tới trường Mai}

d1[u] = d2[u] {thời gian đi từ nhà tới điểm hẹn của Tuấn và Mai bằng nhau}

d1[u] nhỏ nhất {thời gian Tuấn đi từ nhà tới điểm hẹn sớm nhất}

Để ghi kết quả vào file FRIENDS.OUT, cần gọi thủ tục DTra một lần nữa: DTra(u,d); sẽ được mảng D(n) cho biết nhãn đường đi ngắn nhất.

Bài 3: TẢI TRỌNG TUYẾN ĐƯỜNG (*Đề thi Olympia 30/4/2016 - Tin 10*)

Một hệ thống giao thông liên thông gồm **n** thành phố được đánh số từ **1** đến **n**. Hệ thống giao giao thông có **m** đoạn đường hai chiều nối giữa các thành phố. Mỗi đoạn đường có một tải trọng tối đa mà chỉ có các xe với tải trọng không lớn hơn mới lưu thông được.

Yêu cầu: Cho trước tải trọng của các đoạn đường trong hệ thống giao thông. Hãy tìm một hành trình từ thành phố **s** đến thành phố **t** sao cho tải trọng cho phép của các xe luu thông trên hành trình đó là lớn nhất có thể được.

Dữ liệu vào: Cho từ tệp văn bản TAITRONG.INP gồm:

- Dòng thứ nhất ghi 4 số nguyên **n, m, s, t** ($2 \le \mathbf{n} \le 10^3$; $1 \le \mathbf{m} \le 10^4$; $1 \le \mathbf{s, t} \le \mathbf{n}$; $\mathbf{s} \ne \mathbf{t}$)

- Tiếp theo là **m** dòng, mỗi dòng ghi ba số nguyên dương **x, y, z** với ý nghĩa có đoạn đường đi giữa thành phố **x** và thành phố **y** với tải trọng tối đa cho phép là **z** ($1 \le z \le 10^4$).

Các số ghi trên cùng một dòng cách nhau ít nhất một kí tự trắng.

Kết quả: Ghi vào tệp văn bản TAITRONG.OUT chỉ một dòng ghi số nguyên là tải trọng lớn nhất cần tìm.

Ví dụ:

TAITRONG.INP	TAITRONG.OUT
4 5 1 4 1 2 10 2 4 1 1 3 5 3 4 3 1 4 2	3

Ràng buộc: 50% số test ứng với 50% số điểm của bài có $2 \le n \le 100$.

Tư tưởng thuật toán: Xây dựng đơn đồ thị gồm n đỉnh, m cạnh. Mỗi đỉnh là một thành phố. Hai đỉnh x, y có cạnh nối nếu giữa chúng có đoạn đường với trọng số a[x,y]=a[y,x]=z. Áp dụng tư tưởng thuật toán Dijkstra. Gọi D[i] là tải trọng lớn nhất của xe từ s đến i thì D[t] là tải trọng lớn nhất của xe cần tìm.

- Ta cần tìm tải trọng lớn nhất của tuyến đường do đó bước khởi tạo: D[i]=0, với i=1..n; D[s]=Vô cực;

- Lặp: Tại mỗi bước ta tìm đỉnh u tự do có nhãn D[u] lớn nhất, cố định đỉnh u và cập nhật d[v] = Max(d[v] , min(d[u], a[u,v])), với mọi v là đỉnh kề u.

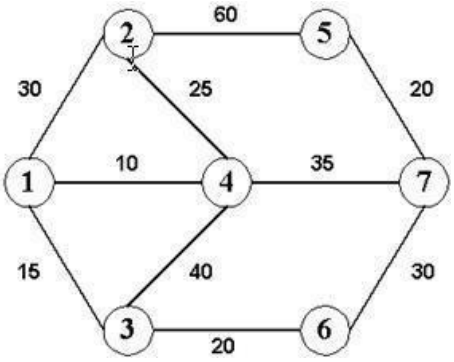
Cho đến khi u=t;

```
const fi='TAITRONG.INP';
      fo='TAITRONG.OUT';
      const nmax=1001;

Var  A:array[1..nmax, 1.. nmax] of LongInt;
     D: array[1..nmax] of LongInt;
     Ch: array[1..nmax] of Boolean;
     K, N, M, S, T, U, V: LongInt;
     F: text;

Procedure doc;
    Var i, x, y, z: LongInt;

Begin
     assign(f, fi); reset(f);
     read(f, n, m, s, t);
     for i:=1 to m do
         Begin
             readln(f, x, y, z);
             a[x,y]:=z;
             a[y,x]:=z;
         end;
```

```
       close(f);

end;
Function Min( x, y: LongInt): LongInt;
   Begin
      min:=x;
      if x> y then min:=y;
   end;

Procedure DTra(s: LongInt);
   Var i, j, max: LongInt;

   Begin
      For i:=1 To n Do D[i]:=0;
      D[s]:=MaxLongInt;
      For k:=1 to n do
            Begin
                  Max:=0;
                  For i:=1 to n do
                        if  (D[i]>Max) and (ch[i]=false) Then
                              Begin
                                    max:=d[i];
                                    u:=i;
                              end;
         Ch[u]:=True;
         For v:=1 to n do
            if (d[v]<min(d[u], a[u,v])) and (a[u,v]<>0) and (ch[v]=false) then
                  d[v]:=Min(d[u],a[u,v]);
        if u=t then exit;
    end;
end;

BEGIN
     Doc;
     Dtra(s);
     Assign(f, fo); rewrite(f);
     Writeln(f, D[t]);
     Close(f);
END.
```

Bài 4: HƯỚNG DẪN VIÊN DU LỊCH

Ông G. là một hướng dẫn viên du lịch. Công việc của ông ta là hướng dẫn một vài "tua" du lịch từ thành phố này đến thành phố khác. Trên các thành phố này, có một vài con đường hai chiều được nối giữa chúng. Mỗi cặp thành phố có đường kết nối đều có dịch vụ xe buýt chỉ chạy giữa hai thành phố này và chạy theo đường nối trực tiếp giữa chúng. Mỗi dịch vụ xe buýt đều có một giới hạn lớn nhất lượng khách mà xe buýt có thể trở được. Ông G có một tấm bản đồ chỉ các thành phố và những con đường nối giữa chúng. Ngoài ra, ông ta cũng có thông tin về mỗi dịch vụ xe buýt giữa các thành phố. Ông hiểu rằng ông không thể đưa tất cả các khách du lịch đến thành phố thăm quan trong cùng một chuyến đi. Lấy ví dụ:



Về bản đồ gồm 7 thành phố, mỗi cạnh được nối giữa các thành phố biểu thị những con đường và các số viết trên mỗi cạnh cho biết cho biết giới hạn lớn nhất hành khách của dịch vụ xe buýt chạy trên tuyến đường đó.

Bây giờ, nếu ông G muốn đưa 99 khách du lịch từ thành phố 1 đến thành phố 7. Ông ta sẽ phải yêu cầu ít nhất là 5 chuyến đi, và lộ trình ông ta nên đi là 1 – 2 – 4 – 7.

Nhưng, Ông G. nhận thấy là thật khó để tìm ra tất cả lộ trình tốt nhất để sao cho ông ta có thể đưa tất cả khách du lịch đến thành phố thăm quan với số chuyến đi là nhỏ nhất. Do vậy mà ông ta cần sự trợ giúp của các bạn.

Dữ liệu: Vào từ file **Tourist.inp**

- Dòng đầu tiên chứa hai số nguyên N ($2 \le N \le 1000$) và R mô tả lần lượt số thành phố và số đường đi giữa các thành phố ($1< R \le 10000$).

- R dòng tiếp theo, mỗi dòng chứa 3 số nguyên: C1, C2, P. C1, C2 mô tả lộ trình đường đi từ thành phố C1 đến thành phố C2 và P là giới hạn có thể phục vụ của dịch vụ xe buýt giữa hai thành phố ($1< P < 1000$).

- Các thành phố được đánh dấu bằng một số nguyên từ 1 đến N. Dòng thứ (R+1) chứa ba số nguyên S, T, L mô tả lần lượt thành phố khởi hành, thành phố cần đến và số khách du lịch được phục vụ. ($1< L<10000$).

Kết quả: Đưa ra file **Tourist.out**

Ghi ra số lộ trình nhỏ nhất cần phải đi qua các thành phố thỏa mãn yêu cầu đề bài.

Ví dụ:

Tourist.inp	Tourist.out
7 10 1 2 30 1 3 15 1 4 10 2 4 25 2 5 60 3 4 40 3 6 20 4 7 35 5 7 20 6 7 30 1 7 99	5

Tư tưởng thuật toán: Bài toán biến thể của bài toán kinh điển tìm đường đi ngắn nhất. Lời giải tương tự bài 3. Lưu ý: số khách đi tuyến đường < giới hạn. Với con đường (u,v) gọi A[u, v] là số người tối đa có thể đi trên con đường đó trong một lần. A[u,v]-1 sẽ là số khách tối đa có thể đi trên con đường đó trong một lần. A[u,v] = 0 tương đương với giữa u và v không có con đường nào. Gọi D[i] là số khách nhiều nhất có thể đi 1 lần từ điểm xuất phát đến i. Với mỗi đỉnh j kề với i, ta cập nhật lại D[j] = min(D[i], C[i, j]). Số khách có thể

đi cùng một lúc từ điểm xuất phát tới điểm kết thúc t là D[t]. Tính số lần đi, ta dùng các phép div, mod để tính.

2. DIJKSTRA HAI TẦNG:

Bài 5: CÁC ĐƯỜNG HẦM

Có **n** thành phố đánh số từ 1 đến **n**. Các thành phố được nối với nhau bởi các con đường đi qua một số các đường hầm. Mỗi con đường nối hai thành phố có một độ dài tương ứng và có một giới hạn độ cao cho các phương tiện đi qua. Một vài cặp thành phố có thể không có đường nối trực tiếp. Không có đường nào là một chiều cả.

Yêu cầu: Cần xác định đường đi từ thành phố X đến thành phố Y qua những con đường đi đạt được độ cao lớn nhất có thể của một phương tiện giao thong. Trong những đường đi này, cần tìm đường đi có tổng chiều dài nhỏ nhất.

Input: file TUNNEL.INP:

- Dòng đầu ghi 3 số N, X, Y. (N <=100)
- Dòng tiếp theo ghi số M là số cặp thành phố có đường nối.
- M dòng tiếp, mỗi dõng mô tả một đường nối hai thành phố gồm 4 số : i , j, H, D – tường ứng là có đường nối giữa hai thành phố i và j, được giới hạn chiều cao H, có độ dài D (H, D nguyên và <=10000)

Output: file TUNNEL.OUT

- Dòng đầu ghi số K là số thành phố mà con đường đi qua (kể cả X, Y)
- Dòng sau là K só hiệu thành phố tương ứng theo thứ tự trên đường đi, bắt đầu là X và kết thúc là Y.

Ví dụ:

TUNNEL.INP	TUNNEL.OUT
6 2 4 10 2 1 900 100 5 2 400 700 1 5 200 600 6 3 200 200 4 5 100 100 2 6 300 400 1 6 500 200 6 5 200 300 3 4 200 300 3 5 300 100	5 2 1 6 3 4

Tư tưởng thuật toán: Để đi từ thành phố X đến thành phố Y đạt được độ cao lớn nhất và độ dài đường đi càng ngắn càng tốt. Trong Dijkstra tiêu chí độ cao được ưu tiên trước.

Gọi H[i], L[i] là độ cao và chiều dài từ đỉnh xuất phát tới đỉnh i.

- Tìm độ cao lớn nhất của tuyến đường do đó bước khởi tạo: H[i]=0, với i=1..n; H[x]=Vô cực;

- Lặp: Tại mỗi bước ta tìm đỉnh v tự do kề u có nhãn H[u] lớn nhất, cố định đỉnh u và cập nhật H[v] = Max(H[v] , min(H[u], a[u-->v])), đồng thời cập nhật L[v] = (L[u] + b[u-->v])); với a[u->v] là độ cao con đừơng từ u đến v, b[u->v] là độ dài con đường từ đỉnh u đến đỉnh v. Cho đến khi u=y;

Trong mỗi bước sửa nhãn ta dùng mảng Tr lưu vết đường đi, dựa vào mảng Tr trả lời đáp số bài toán.

```
{$A+,B-,D+,E+,F+,G-,I+,L+,N+,O+,P+,Q+,R+,S+,T+,V+,X+}
{$M 16384,0,655360}

Const   fi   =      'Tunnel.inp';
        fo   =      'Tunnel.out';
        maxN =      100;
Type
        bb   =      byte;
        ii   =      integer;
        ss   =      shortint;
        ll   =      longint;
        km   =      array[0..maxN] of boolean;

Var     f1,f2    :      text;
        time     :      ll;
        n,m      :      ii;
        x,y      :      ss;
        f        :      array[1..maxN,1..maxN] of record h,l:ii;end;
        h,l                  :      array[1..maxN] of ll;
        tr                   :      array[1..maxN] of ss;
        max                  :      ii;

Procedure Open_files;
begin
 Time:=meml[0:$46C];
 assign(f1,fi);reset(f1);
 assign(f2,fo);rewrite(f2);
end;

Procedure Close_files;
begin
 close(f1); close(f2);
 writeln('  Time = ',(meml[0:$46C]-time)/18.2:0:5);
end;

Procedure Read_Data;
var a,b:ss;p,q,i:ii;
begin
   fillchar(f,sizeof(f),0);
   readln(f1,n,x,y);
   readln(f1,m);
   for i:=1 to m do
    begin
      readln(f1,a,b,p,q);
      f[a,b].l:=q;f[a,b].h:=p;
      f[b,a].l:=q;f[b,a].h:=p;
    end;
```

```
end;

Function min(a,b:ii):ii;
begin
  if a<b then min:=a else min:=b;
end;

Procedure Print_Data_1;
var kq:array[1..maxN] of ss;
    count,t:ss;
begin
   max:=h[y];
   fillchar(kq,sizeof(kq),0);
   tr[x]:=0;
   count:=1;t:=y;kq[count]:=y;
   repeat
     inc(count);
     t:=tr[t];
     kq[count]:=t;
   until tr[t]=0;
   writeln(f2,count);
   for t:=count downto 1 do write(f2,kq[t],' ');
end;

Procedure Print_Data_2;
var kq:array[1..maxN] of ss;
    count,t:ss;
begin
   fillchar(kq,sizeof(kq),0);
   tr[x]:=0;
   count:=1;t:=y;kq[count]:=y;
   repeat
     inc(count);
     t:=tr[t];
     kq[count]:=t;
   until tr[t]=0;
   if l[x]<max then
   begin
   close(f2);rewrite(f2);
   writeln(f2,count);
   for t:=1 to count do write(f2,kq[t],' ');
   end;
end;

Procedure Perform_1;
var i,j:ss;ok:boolean;
begin
    fillchar(tr,sizeof(tr),0);
    fillchar(h,sizeof(h),0);
    fillchar(l,sizeof(l),0);
    for i:=1 to n do
     if f[x,i].h<>0 then begin h[i]:=f[x,i].h;l[i]:=f[x,i].l;tr[i]:=x;end;
    h[x]:=10000;

    repeat
      ok:=true;
      for i:=1 to n do
       if i<>x then
```

```
       for j:=1 to n do
        if (h[j]>0) and (f[j,i].l>0) then
        begin
          if (h[j]>h[i]) and (f[j,i].h>h[i]) then
           begin
              h[i]:=min(h[j],f[j,i].h);
              l[i]:=l[j]+f[j,i].l;
              tr[i]:=j;
              ok:=false;
           end
           else
            if (h[j]>=h[i]) and (f[j,i].h>=h[i]) then
             if (l[i]>l[j]+f[j,i].l) or (l[i]=0) then
             begin
                h[i]:=min(h[j],f[j,i].h);
                l[i]:=l[j]+f[j,i].l;
                tr[i]:=j;
                ok:=false;
             end
        end;
    until ok;
    Print_Data_1;
end;

Procedure Perform_2;
var i,j,tg:ss;ok:boolean;
begin
    tg:=x;x:=y;y:=tg;
    fillchar(tr,sizeof(tr),0);
    fillchar(h,sizeof(h),0);
    fillchar(l,sizeof(l),0);
    for i:=1 to n do
     if f[x,i].h<>0 then begin h[i]:=f[x,i].h;l[i]:=f[x,i].l;tr[i]:=x;end;
    h[x]:=10000;

    repeat
      ok:=true;
      for i:=1 to n do
       if i<>x  then
       for j:=1 to n do
        if (h[j]>0) and (f[j,i].l>0) then
        begin
          if (h[j]>h[i]) and (f[j,i].h>h[i]) then
           begin
              h[i]:=min(h[j],f[j,i].h);
              l[i]:=l[j]+f[j,i].l;
              tr[i]:=j;
              ok:=false;
           end
           else
            if (h[j]>=h[i]) and (f[j,i].h>=h[i]) then
             if (l[i]>l[j]+f[j,i].l) or (l[i]=0) then
             begin
                h[i]:=min(h[j],f[j,i].h);
                l[i]:=l[j]+f[j,i].l;
                tr[i]:=j;
                ok:=false;
             end
```

```
        end;
    until ok;
    Print_Data_2;
end;

BEGIN
  Open_files;
  Read_Data;
  Perform_1;
  Perform_2;
  Close_files;
END.
```

Bài 6: ĐƯỜNG ĐI CHI PHÍ TIẾT KIỆM (Nguồn: vn.spoj.com)

Có **n** thành phố được đánh số từ 1 đến **n** nối với nhau bằng các con đường một chiều. Mỗi con đường có hai giá trị: độ dài và chi phí phải trả để đi qua con đường đó. Bé Sen ở thành phố **1.** Bạn hãy giúp bé Sen tìm đường đi từ thành phố 1 đến thành phố **n** mà chi phí không quá k.

Dữ liệu: vào từ file văn bản ROADS.INP

- Dòng đầu tiên ghi **t** là số test. Với mỗi test:

- Dòng đầu ghi số nguyên k ($0 \le K \le 10000$) là số tiền tối đa mà bé Sen còn có thể chi cho lệ phí đi đường.

- Dòng 2 ghi số nguyên n ($2 \le n \le 100$) là số thành phố.

- Dòng 3 ghi số nguyên m là số đường nối.

- Mỗi dòng trong m dòng tiếp theo ghi 4 số nguyên x, y, z, l mô tả một con đường nối giữa x và y với độ dài z ($1 \le z < 100$) và chi phí l ($0 \le l < 100$).

Kết quả: ghi ra file văn bản ROADS.OUT

Với mỗi test, in ra độ dài đường đi chi phí không quá k xuất phát từ 1 đến n. Nếu không tồn tại, in ra -1.

Ví dụ:

ROADS.INP	ROADS.OUT
2 5 6 7 1 2 2 3 2 4 3 3 3 4 2 4 1 3 4 1 4 6 2 1 3 5 2 0 5 4 3 2 0 4 4	11 -1

1 4 5 2 1 2 1 0 2 3 1 1 3 4 1 0	

Tư tưởng thuật toán: Sử dụng thuật toán **Dijkstra** hai tầng.

Gọi D1[i] nhãn chi phí, D2[i] nhãm khoảng cách.

- Tìm đường đi ngắn nhất (về chi phí tiền) từ đỉnh 1 đến các đỉnh khác để tính chi phí D1[i] tối ưu, nếu D1[n] >k thì bài toán không có nghiệm.

- Trong mỗi bước sửa nhãn ta cập nhật D1[v]=min(D1[u], D1[u]+a[u->v]); D2[v]=Min(D[u], D[u]+b[u->v]);

Chú ý: Để test trên vn.spoj.com cần biểu diễn đồ thị bằng danh sách kề.

3. THUẬT TOÁN DIJKSTRA KẾT HỢP HEAP NHỊ PHÂN.

Thuật toán Dijkstra cổ điển có độ phức tạp là $O(n^2)$. Hai thao tác tốn thời gian nhất của mỗi lần lặp là:

[1] Lấy đỉnh **u** tự do có giá trị **d[u]** nhỏ nhất ở bước 2.1

[2] Cập nhật nhãn **d[v]** của các đỉnh tự do kề **u** ở bước 2.2

Thời gian thực hiện thao tác [1] là **O(n**), thao tác [2] phụ thuộc vào bậc của đỉnh u **O(deg(u))**. Ta có thời gian thực hiện thuật toán: $\mathbf{O(n^2+m) = O(n^2)}$. Tuy nhiên, nếu biểu diễn đồ thị bằng danh sách kề kết hợp một **Heap** nhị phân, thao tác thứ [1] chính là lấy gốc của **min Heap** và thao tác thứ [2] là cập nhật khóa của **Heap**, mỗi thao tác mất **O(log(n))** thời gian để thực hiện thuật toán **Dijkstra** với Heap nhị phân: **O((n+m)logn).**

Lưu ý: Heap ta ghi nhận tập đỉnh của đồ thị chứa không phải ghi khóa, còn khóa của heap ta lưu trong mảng D. Như vậy khóa của đỉnh **i** là **D[heap[i]]** còn thứ tự của đỉnh **i** ta ghi trong mảng **pos**. Hay nói cách khác **pos[v]=i** khi và chỉ khi **Heap[i]=v**

Hai thao tác cập nhật và loại bỏ gốc của **Heap** nhị phân có mã Pascal như sau:

```
Procedure update(v: longint);
var cha,con: longint;
begin
      con:=pos[v];
      if con=0 then
      begin
            inc(nheap);
            con:=nheap;
      end;
            cha:=con div 2;
      while (cha>0) and (d[heap[cha]]>d[v]) do
      begin
            heap[con]:=heap[cha];
            pos[heap[con]]:=con;
            con:=cha;
            cha:=con div 2;
      end;
      heap[con]:=v;
```

```
      pos[v]:=con;
end;
```

```
Function pop: longint;
var v,cha,con: longint;
begin
      pop:=heap[1];
      v:=heap[nheap]; dec(nheap);
      cha:=1;
      while  2*cha<=nheap do
      begin
            con:=2*cha;
            if (con<nheap) and (d[heap[con]]>d[heap[con+1]]) then inc(con);
            if d[heap[con]]>=d[v] then break;
            heap[cha]:=heap[con];
            pos[heap[cha]]:=cha;
            cha:=con;
      end;
      heap[cha]:=v;
      pos[v]:=cha;
end;
```

Từ đó ta suy ra thuật toán Dijkstra với Heap nhị phân có gỉa mã:

```
Procedure DTraHeap(s, t: longint);
Var i, u, v: longint;
Begin
    //B1, Khoi tao
      D[i]<--+∞;∀i=1..n
      Ch[i]<-- False; ∀i=1..n

      D[s]<--0;
      Update(s);// cap nhat nut 1 cua heap

    //B2. Lap cho den khi heap = rong
      Repeat
          //B2.1 Lay dinh ke co D[u] nho nhat
           U<--pos;// Loai bo nut goc cua heap

           Ch[u]<-- True;// danh dau dinh u co dinh
           If u = t Then Exit;
           // B2.2 Sua nhan;
           Duyet tat ca cac đỉnh v tự do va ke u. Neu d[v]> d[u]+ a[u->v] thì
                 Begin
                       D[v]<--D[u]+A[u-->v];
                       Update(v);//cap nhat nut v vao heap
                 End;
      Until heap=rong;
End;
```

Bài 7. GIAO THÔNG:

Sống trong một thành phố lớn có mật độ dân số cao và giao thông phức tạp, ông Bình rất lo về nạn kẹt xe ảnh hưởng đến cuộc sống của mình. Mỗi sáng thức dậy công việc đầu tiên của ông Bình là nghe bản tin giao thông của thành phố. Ông Bình rất lo là con đường quen thuộc hàng ngày đi đến cơ quan bị kẹt xe thì ông không biết phải chọn con đường nào để thay thế sao cho ông có thể đến cơ quan sớm nhất có thể.

Bản đồ thành phố là gồm có **n** nút giao thông và **m** con đường hai chiều nối các nút giao thông này. Độ dài của mỗi con đường là một số nguyên dương. Nhà ông Binh ở nút giao thông 1, cơ quan ông bình ở nút giao thông **n**. Để thuận tiện cho việc đi lại, ông Bình đang tìm hiểu các con đường ngắn nhất dẫn từ nhà ông Bình đến cơ quan. Bạn hãy giúp ông Bình thực hiện công việc kho khăn này.

Input:

Dòng thứ nhất ghi hai số nguyên **n** và **m** ($1 \le \mathbf{n} \le 5000$, $1 \le \mathbf{m} \le 20000$)

m dòng tiếp theo, mỗi dòng ghi 3 số nguyên dương U, V, L (con đường nối U đến V với độ dài L, $L \le 32000$).

Output: Ghi hai số là độ dài đường đi ngắn nhấn và số lượng đường đi ngắn nhất.

Ví dụ:

Input	Output
4 5 1 2 1 1 3 2 1 4 3 2 3 1 3 4 1	3 3

Ý tưởng thuật toán: Dijkstra Heap nhị phân kết hợp quy hoạch động và xử lí số lớn:

Dễ thấy, đây là bài toán Tìm đường đi ngắn nhất từ đỉnh 1 đến đỉnh n. Tuy nhiên:

- Theo điều kiện đề bài n, m tương đối lớn ta cần dùng Dijkstra với cấu trúc heap nhị phân, và $L \le 32000$ cần xử lí số lớn.

- Theo yêu cầu xuất ra số lượng đường đi ngắn nhất, ta dùng quy hoạt động như sau:

+ Gọi Count[i] là số đường đi ngắn nhất từ đỉnh s đến đỉnh i.

Khởi tạo Count [i]=0 với mọi $i \ne s$ và Count [s]=1

Trong thuật toán Dijkstra: Khi tìm được đường đi mới có độ dài ngắn hơn (d[v]>d[u]+a[u->v]) ta tiến hành thay đổi d[v]:=d[u]+a[u-->v]) đồng thời Count[v]:=Count[u].

Mỗi khi tìm được 2 đường đi có độ dài bằng nhau (d[v]=d[u]+A[u->v]) ta thay đổi Count[v]:=Count[v]+Count[u]. Kết quả là Count[n].

Bài giải tham khảo:

```
Program GiaoThong;
const
      fi='CPATH.inp';
      fo='CPATH.out';
      maxn=10000;
      maxm=41000;
      vc=1000000000;
var
      n,m,nheap: longint;
      f: text;
      h,heap,d: array [-1..maxn+4] of longint;
            count: array [-1..maxn+4] of string;
            dd: array [-1..maxn+4] of boolean;
      adj,adjcost,pos: array [-1..maxm+4] of longint;
procedure doc;
var
      i: longint;
      dau,cuoi,cost:  array [-1..maxm+4] of longint;
begin
      assign(f,fi); reset(f);
      fillchar(h,sizeof(h),0);
      readln(f,n,m);
      for i:=1 to m do
            begin
                  readln(f,dau[i],cuoi[i],cost[i]);
                  inc(h[dau[i]]);
                  inc(h[cuoi[i]]);
            end;
      for i:=2 to n do h[i]:=h[i-1]+h[i];
      h[n+1]:=h[n];
      for i:=1 to m do
            begin
                  adj[h[dau[i]]]:=cuoi[i];
                  adjcost[h[dau[i]]]:=cost[i];
                  dec(h[dau[i]]);
                        adj[h[cuoi[i]]]:=dau[i];
                        adjcost[h[cuoi[i]]]:=cost[i];
                        dec(h[cuoi[i]]);
            end;
      close(f);
end;
procedure update(v: longint);
var cha,con: longint;
begin
      con:=pos[v];
      if con=0 then
      begin
            inc(nheap);
            con:=nheap;
      end;
            cha:=con div 2;
      while (cha>0) and (d[heap[cha]]>d[v]) do
      begin
            heap[con]:=heap[cha];
            pos[heap[con]]:=con;
            con:=cha;
```

```
            cha:=con div 2;
      end;
      heap[con]:=v;
      pos[v]:=con;
end;
function pop: longint;
var v,cha,con: longint;
begin
      pop:=heap[1];
      v:=heap[nheap]; dec(nheap);
      cha:=1;
      while  2*cha<=nheap do
      begin
            con:=2*cha;
            if (con<nheap) and (d[heap[con]]>d[heap[con+1]]) then inc(con);
            if d[heap[con]]>=d[v] then break;
            heap[cha]:=heap[con];
            pos[heap[cha]]:=cha;
            cha:=con;
      end;
      heap[cha]:=v;
      pos[v]:=cha;
end;
function cong(a,b: string): string;
var
      c: string;
      nho,x,y,z: byte;
      i: longint;
begin
      while length(a)<length(b) do a:='0'+a;
      while length(b)<length(a) do b:='0'+b;
      nho:=0; c:='';
      for i:=length(a) downto 1 do
      begin
            x:=ord(a[i])-48;
            y:=ord(b[i])-48;
            z:=x+y+nho;
            c:=chr((z mod 10)+48)+c;
            nho:=z div 10;
      end;
      if nho>0 then c:='1'+c;
      cong:=c;
end;
procedure DTra;
var i,u: longint;
begin
      fillchar(dd,sizeof(dd),false);
      fillchar(pos,sizeof(pos),0);
      fillchar(heap,sizeof(heap),0);
      for i:=1 to n do count[i]:='';
      count[1]:='1';
      for i:=1 to n do d[i]:=vc;
      d[1]:=0;
      nheap:=0;  update(1);
      repeat
            u:=pop;
            if u=0 then break;
            dd[u]:=true;
```

```
            for i:=h[u]+1 to h[u+1] do
                        if not(dd[adj[i]]) then
                              if (d[adj[i]]>d[u]+adjcost[i])  then
                                    begin
                                          d[adj[i]]:=d[u]+adjcost[i];
                                          update(adj[i]);
                                          count[adj[i]]:=count[u];
                                    end
                        else
if                        (d[adj[i]]=d[u]+adjcost[i])                        then
count[adj[i]]:=cong(count[adj[i]],count[u]);
      until (nheap=0) ;
end;
procedure ghi;
var i: longint;
begin
      assign(f,fo); rewrite(f);
      write(f,d[n],' ',count[n]);
      close(f);
end;
BEGIN
doc;
dtra;
ghi;
END.
```

Bài 8. GỬI THƯ

Một công ty bưu điện chịu trách nhiệm chuyển phát thư một cách nhanh chóng giữa các thành phố. Có tất cả N thành phố được đánh số từ 1..N, với M con đường hai chiều nối giữa các thành phố. Việc di chuyển trên mỗi con đường mất một khoảng thời gian là T. Bức thư từ thành phố A muốn gửi đến thành phố B phải đi theo một quy trình do công ty đặt ra. Đầu tiên nhân viên bưu điện mất một khoảng thời gian T1 vận chuyển thư từ thành phố A đến công ty bưu điện đặt tại ***thành phố 1*** để kiểm tra tính hợp lệ của các bức thư, sau đó mất thêm một khoảng thời gian T2 để đưa các bức thư từ công ty đến thành phố B. Như vậy thời gian tổng cộng để gửi 1 bức thư từ thành phố A đến thành phố B là T1+T2.

Yêu cầu: Biết rằng có rất nhiều yêu cầu gửi thư giữa các thành phố khác nhau. Hãy tính thời gian ngắn nhất để một bức thư được gửi từ thành phố A đến thành phố B.

Dữ liệu: Cho trong file văn bản Mail.inp

Dòng đầu tiên gồm 4 số nguyên N, M, A và B. M dòng tiếp theo, mỗi dòng ghi 3 số nguyên dương L, P, T biểu diễn cho việc di chuyển trên con đường hai chiều nối thành phố L và P, sẽ mất khoảng thời gian là T.

Kết quả: Ghi ra file văn bản Mail.out số nguyên duy nhất là tổng thời gian nhanh nhất để chuyển thư từ A đến B.

Ví dụ:

Mail.inp	**Mail.out**
4 5 2 4 1 2 3	6

2 4 4 1 3 1 3 4 2 1 4 7	

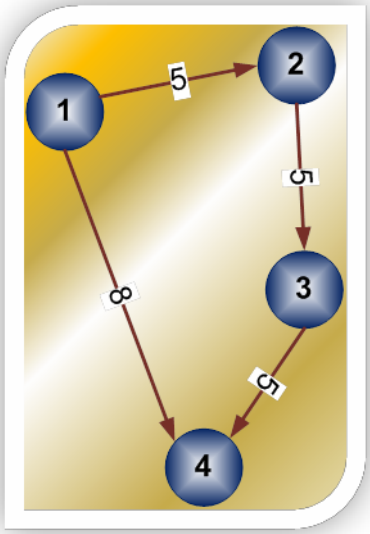
Giới hạn dữ liệu:

- $1 \le N, M \le 10^5$
- $1 \le L \ne P \le N, 1 \le T \le 10^6$

Tư tưởng của thuật toán: Dùng Dijkstra + Heap nhị phân

Bài 9: ĐƯỜNG ĐI

Trong khu vực được xét có ***n*** thành phố, đánh số từ 1 đến ***n***. Các thành phố được nối với nhau bằng ***m*** tuyến đường một chiều. Với mỗi tuyến đường người ta cho biết thành phố xuất phát, thành phố đích và độ dài của nó. Giữa hai thành phố có thể có nhiều tuyến đường nối.



Đường đi ngắn nhất từ ***A*** tới ***B*** là đường mà tổng độ dài các tuyến đi qua là nhỏ nhất.

Mỗi tuyến đường có thể thuộc một hoặc nhiều đường đi ngắn nhất giữa các cặp thành phố. Ví dụ, với mạng lưới giao thông ở hình bên, tuyến đường từ 1 tới 2 thuộc các đường đi ngắn nhất từ 1 tới 2 và từ 1 tới 3, còn tuyến đường từ 1 tới 4 chỉ thuộc một đường đi ngắn nhất từ 1 tới 4.

Yêu cầu: Cho ***n***, ***m*** và thông tin về mỗi tuyến đường. Với mỗi tuyến hãy xác định số lượng đường ngắn nhất mà tuyến đó tham gia. Số này có thể rất lớn nên bạn chỉ cần đưa ra số dư của kết quả tìm được khi chia cho 10^9+7.

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

- Dòng đầu tiên chứa 2 số nguyên ***n*** và ***m*** ($1 \le n \le 1\,500$, $1 \le m \le 5\,000$),
- Mỗi dòng trong ***m*** dòng sau chứa 3 số nguyên xác định điểm đầu, điểm cuối và độ dài con đường (độ dài không vượt quá 10 000).

Kết quả: Đưa ra file văn bản PATHS.OUT ***m*** dòng, mỗi dòng chứa một số nguyên, dòng thứ ***i*** xác định kết quả tìm được với tuyến đường ***i***.

Ví dụ:

PATHS.INP		PATHS.OUT
4 4 1 2 5 2 3 5		2 3 2

3 4 5 1 4 8

1

Ràng buộc:

30% số test có N<=15 và M<=30

Tư tưởng thuật toán:

Để tính được số đường đi ngắn nhất qua cạnh (u,v) ta tính được số đường đi ngắn nhất đến u (to[u]) và số đường đi ngắn nhất từ v đi ra (from[v])=>kết quả số đường đi ngắn nhất đi qua cạnh (u,v)=to[u]*from[v].

Sử dụng Dijkstra n lần, mỗi lần ta tính được to[u] và from[u]. Mỗi cạnh (u,v) ta tính tích lũy kết quả cho cạnh (u,v). Ta cài đặt đồ thị bằng danh sách kề và heap để giải bài toán.

Bài 10: VÉ XE MIỄN PHÍ (Duyên Hải 2015)

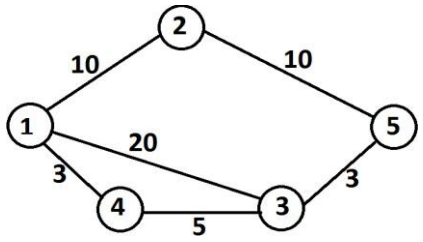
Tham gia trò chơi nhảy lò cò, thật may mắn, Khuê đã giành giải nhất của cuộc thi. Phần thưởng mà Khuê nhận được là *k* vé xe buýt miễn phí để đi thăm quan thành phố Hạ Long. Mỗi vé xe chỉ được sử dụng một lần và có thể sử dụng cho bất kỳ tuyến xe buýt nào trong thành phố. Thành phố có *n* nút giao thông được đánh số từ 1 đến *n* và *m* tuyến xe buýt hai chiều. Mỗi cặp nút giao thông *i, j* có không quá một tuyến xe buýt hai chiều, nếu có thì để đi từ nút *i* đến nút *j* (hoặc từ nút *j* đến nút *i*) với giá vé là $c_{ij} = c_{ji}$ đồng. Xuất phát từ nút giao thông *s*, Khuê muốn di chuyển đến nút giao thông *t* và anh luôn lựa chọn đường đi với chi phí ít nhất.

Ví dụ: thành phố có 5 nút giao thông và 6 tuyến xe buýt:

Tuyến 1: 1-2 giá vé 10 đồng; Tuyến 2: 2-5 giá vé 10 đồng;

Tuyến 3: 1-4 giá vé 3 đồng; Tuyến 4: 3-4 giá vé 5 đồng;

Tuyến 5: 3-5 giá vé 3 đồng; Tuyến 6: 1-3 giá vé 20 đồng.



Xuất phát từ nút 1 đến nút 5, đi theo hành trình 1→4→3→5 hết 11 đồng là đường đi với chi phí ít nhất. Tuy nhiên, nếu Khuê sử dụng 1 vé xe miễn phí thì đường đi 1→3→5 hết 3 đồng là ít nhất (vé xe miễn phí được sử dụng tại tuyến 1-3).

Yêu cầu: Cho biết các tuyến xe buýt với giá vé tương ứng và các giá trị *s, t, k*. Hãy tính chi phí ít nhất để đi từ nút giao thông *s* đến nút giao thông *t* mà không sử dụng quá *k* vé xe miễn phí.

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

- Dòng đầu tiên ghi năm số nguyên dương *n, m, k, s, t*;
- *m* dòng sau, mỗi dòng 3 số nguyên *i, j,* c_{ij} mô tả có tuyến xe buýt *i – j* hết c_{ij} đồng.

Kết quả: Đưa ra file văn bản FREEBUS.OUT một số duy nhất là chi phí ít nhất để đi từ nút giao thông *s* đến nút giao thông *t* mà không sử dụng quá *k* vé xe miễn phí.

Ví dụ:

FREEBUS.INP	FREEBUS.OUT
5 6 1 1 5 1 2 10 2 5 10 1 4 3 3 4 5 3 5 3 1 3 20	3

Ràng buộc dữ liệu:

- Có 40% số test ứng với 40% số điểm có $n \le 100$, $m \le 1000$ và $k = 1$;
- Có 20% số test ứng với 20% số điểm có $n \le 10^5$, $m \le 10^5$ và $k = 1$;
- Có 40% số test còn lại ứng với 40% số điểm có $n \le 10^5$, $m \le 10^5$ và $k \le 5$.