CHUYÊN ĐỀ

ĐƯỜNG ĐI NGẮN NHẤT TRÊN ĐỒ THỊ

A, MỞ ĐẦU

1. Lý do chọn đề tài

Lý thuyết đồ thị là một lĩnh vực được phát triển từ rất lâu, được nhiều nhà khoa học quan tâm nghiên cứu nó có vai trò hết sức quan trọng trong nhiều lĩnh vực. Trong Tin học lý thuyết đồ thị được ứng dụng một cách rộng rãi có rất nhiều thuật toán được nghiên cứu và ứng dụng. Trong chương trình môn Tin học của THPT chuyên phần lý thuyết đồ thị nói chung và các thuật toán tìm đường đi ngắn nhất trên đồ thị là nội dung rất quan trọng, trong các kỳ thi học sinh giỏi xuất hiện rất nhiều các bài toán liên quan đến việc tìm đường đi ngắn nhất trên đồ thị. Tuy nhiên trong qua trình giảng dạy tôi thấy học sinh vẫn còn khó khăn trong việc phân tích bài toán để có thể áp dụng được thuật toán và cài đặt giải bài toán. Vì vậy tôi chọn chuyên đề này để giúp học sinh có cái nhìn tổng quan hơn về các thuật toán tìm đường đi ngắn nhất trên đồ thị.

2. Mục đích của đề tài

Về nội dung kiến thức của các thuật toán tìm kiếm trên đồ thị đã có rất nhiều tài liệu đề cập đến, trong chuyên đề này tôi chỉ tổng hợp lại các nội dung kiến thức đã có và đưa vào áp dụng để giải một số bài toán cụ thể, để làm tài liệu tham khảo cho học sinh và giáo viên trong quá trình học tập và giảng dạy.

A. NỘI DUNG

I, Giới thiệu bài toán đường đi ngắn nhất

- Trong thực tế có rất nhiều các bài toán chẳng hạn như trong mạng lưới giao thông nối giữa các Thành Phố với nhau, mạng lưới các đường bay nối các nước với nhau người ta không chỉ quan tâm tìm đường đi giữa các địa điểm với nhau mà phải lựa chọn một hành trình sao cho tiết kiệm chi phí nhất (chi phí có thể là thời gian, tiền bạc, khoảng cách…). Khi đó người ta gán cho mỗi cạnh của đồ thị một giá trị phản ánh chi phí đi qua cạnh đó và cố gắng tìm ra một hành trình đi qua các cạnh với tổng chi phí thấp nhất.

- Ta đi xét một đồ thị có hướng $G = (V, E)$ với các cung được gán trọng số (trọng số ở đây là chi phí). Nếu giữa hai đỉnh u, v không có cạnh nối thì ta có thể thêm vào cạnh "giả" với trọng số $a_{ij} = +\infty$. Khi đó đồ thị G có thể giả thiết là đồ thị đầy đủ.
- Nếu dãy $v_0, v_1, ..., v_p$ là một đường đi trên G thì độ dài của nó được định nghĩa là tổng các trọng số trên các cung của nó: $\sum_{i=1}^{p} a(v_{i-1}, v_i)$

- Bài toán đặt ra là tìm đường đi có độ dài nhỏ nhất từ một đỉnh xuất phát $s \in V$ đến đỉnh đích $t \in V$. Đường đi như vậy gọi là đường đi ngắn nhất từ s đến t và độ dài của nó ta còn gọi là khoảng cách từ s đến t, kí hiệu $d(s, t)$.

- Nhận xét:
  + Khoảng cách giữa hai đỉnh của đồ thị có thể là số âm.
  + Nếu như không tồn tại đường đi từ s đến t thì ta sẽ đặt $d(s, t) = +\infty$.
  + Nếu như trong đồ thị, mỗi chu trình đều có độ dài dương thì đường đi ngắn nhất sẽ không có đỉnh nào bị lặp lại. Đường đi không có đỉnh lặp lại gọi là *đường đi cơ bản*. Còn nếu trong đồ thị có chứa chu trình với độ dài âm (gọi là chu trình âm) thì khoảng cách giữa một số cặp đỉnh nào đó của đồ thị là không xác định, bởi vì bằng cách đi vòng theo chu trình này một số lần đủ lớn, ta có thể chỉ ra đường đi giữa các đỉnh này có độ dài nhỏ hơn bất kì số thực nào cho trước. Trong những trường hợp như vậy ta có thể đặt vấn đề tìm đường đi cơ bản ngắn nhất.
  + Trong thực tế, bài toán tìm đường đi ngắn nhất giữa hai đỉnh của một đồ thị liên thông có một ý nghĩa to lớn. Nhiều bài toán có thể dẫn về bài toán trên. Ví dụ bài toán chọn một hành trình tiết kiệm nhất (theo tiêu chuẩn khoảng cách hoặc thời gian hoặc chi phí) trên một mạng giao thông đường bộ, đường thuỷ hoặc đường không. Bài toán lập lịch thi công các công đoạn trong một công trình lớn, bài toán lựa chọn đường truyền tin với chi phí nhỏ nhất trong mạng thông tin, ... Hiện nay có rất nhiều phương pháp để giải bài toán trên. Trong bài này ta xét các giải thuật được xây dựng trên cơ sở lý thuyết đồ thị tỏ ra là hiệu quả cao nhất.

II, Đường đi ngắn nhất xuất phát từ một đỉnh

1, Bài toán tìm đường đi ngắn nhất xuất phát từ một đỉnh được phát biểu như sau : Cho đồ thị có trọng số G=(V,E,w) hãy tìm đường đi ngắn nhất từ đỉnh xuất phát s đến các đỉnh còn lại của đồ thị. Độ dài đường đi từ đỉnh s đến đỉnh t kí hiệu là δ(s,t) gọi là khoảng cách từ s tới t nếu như không tồn tại khoảng cách từ s tới t thì ta đặt khoảng cách đó là + ∞.

2,Giải thuật Ford-Bellman

- Thuật toán Ford – Bellman có thể dùng để tìm đường đi ngắn nhất xuất phát từ một đỉnh s thuộc V trong trường hợp đồ thị G=(V,E,w) không có chu trình âm thuật toán như sau:
  + Gọi $d[v]$ là khoảng cách từ đỉnh s đến đỉnh $v \in V$, $d[v]=0$, $d[t]=+\infty$. Sau đó ta thực hiện phép co tức là mỗi khi phát hiện $d[u] + a[u, v] < d[v]$ thì cận trên $d[v]$ sẽ được tốt lên $d[v] := d[u] + a[u, v]$. Quá trình trên kết thúc khi nào ta không thể làm tốt thêm được bất cứ cận trên nào . Khi đó giá trị của mỗi $d[v]$ sẽ cho khoảng cách từ s đến v. Nói riêng $d[t]$ là độ dài ngắn nhất giữa hai đỉnh s và t.

Cài đặt thuật toán

```
Procedure       Ford_Bellman ;

Begin

    For i := 1 to n do

    begin

            d [i]:=maxint ;

            tr[i]:=maxint ;

    end ;

    d[s]:=0;

    Repeat

            Ok:=true;

            For i:=1 to n do

             if d[i]<>maxint then

            for j:=1 to n do

            if (a[i,j]<>0)and(d[i]+a[i,j]<d[j]) then

            begin

                    ok:=false;

                    d[j]:=d[i]+a[i,j];

                    tr[j]:=i;

            end;

    until ok ;
```

Nhận xét:

- Việc chứng minh tính đúng đắn của giải thuật trên dựa trên cơ sở nguyên lý tối ưu của quy hoạch động.
- Độ phức tạp tính toán của giải thuật Ford-Bellman là $O(n^3)$.

- Thực chất của thuật toán này là thuật toán Quy Hoạch Động trong đó , d[i] là mảng độ dài ngắn nhất đi từ s đến i . vậy nếu t là đỉnh cần thiết thì d[t] là độ dài cần tìm . Còn nếu muốn lưu lại đường đi thì chúng ta dùng mảng Tr [i] để đi ngược lại .

3, Thuật toán Dijkstra

Trong trường hợp đồ thị G=(V,E,w) có trọng số trên các cung không âm thuật toán Dijkstra đề cập dưới đây hoạt động hiệu quả hơn nhiều so với thuật toán Ford – Bellman. Thuật toán Dijkstra như sau:

Bước 1: Khởi tạo

Với đỉnh $v \in V$, ta gọi d[v] là độ dài đường đi từ s tới v ban đầu khởi tạo d[v]=0, d[t]=+ ∞. $\forall v \neq s$. Nhãn của mỗi đỉnh có hai trạng thái tự do hay cố định, nhãn tự do có nghĩa là có thể tối ưu được nữa, nhãn cố định d[v] là đường đi ngắn nhất từ s tới v nên không thể tối ưu được nữa. Ta dùng thêm một mảng Free[] để đánh dấu nếu d[v] là tự do thì Free[v]=True, ngược lại Free[v]=Flase. Ban đầu các nhãn đều tự do.

Bước 2: Lặp

Bước lặp gồm hai thao tác :

- Cố định nhãn: chọn trong các đỉnh có nhãn tự do lấy ra đỉnh u có d[u] nhỏ nhất và cố định d[u]
- Sửa nhãn: dùng đỉnh u để xét tất cả các đỉnh v và sửa lại các nhãn d[v] theo công thức sau:

d[v]= min (d[v], d[u]+c[u,v])

Bước lặp sẽ kết thúc khi mà 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 đường đi ngắn nhất tìm được hoặc cho biết không tồn tại đường đi d[t]=+ ∞.

Cài đặt thuật toán:

```
Const
  MAX_N = 100;
  FI = 'dijkstra.inp';
  FO = 'dijkstra.out';
Var
  n, nU, s, t : integer;
  a : array[1..MAX_N, 1..MAX_N] of integer;
  d, tr, U : array[1..MAX_N] of integer;
  f : text;

Procedure Doc;
```

```
Var i, j : integer;
Begin
  assign(f, FI); reset(f);

  read(f, n, s, t);
  for i := 1 to n do
    for j := 1 to n do read(f, a[i, j]);
  close(f);
End;

Procedure Khoi_tao;
Var i : integer;
Begin
  for i := 1 to n do
    begin
      tr[i] := s;
      d[i] := a[s, i];
    end;
  for i := 1 to n do U[i] := i;
  U[s] := U[n];
  nU := n - 1;
End;
Function Co_dinh_nhan : integer;
Var i, j, p : integer;
Begin
  { Tim p }
  i := 1;
  for j := 2 to nU do
    if d[U[i]] > d[U[j]] then i := j;
  p := U[i];

  { Loai p ra khoi U }
  U[i] := U[nU];
  nU := nU - 1;
  Co_dinh_nhan := p;
End;
Procedure Sua_nhan(p : integer);
Var i, x : integer;
Begin
  for i := 1 to nU do
    begin
      x := U[i];
```

```
      if d[x] > d[p] + a[p, x] then
        begin
          tr[x] := p;
          d[x] := d[p] + a[p, x];
        end;
    end;
End;
Procedure Print(i : integer);
Begin
  if i = s then
    begin
      writeln(f, d[t]);
      write(f, s);
      exit;
    end;
  Print(tr[i]);
  write(f, ' ', i);
End;
Procedure Ghi;
Begin
  assign(f, FO); rewrite(f);
  Print(t);
  close(f);
End;
Procedure Dijkstra;
Var p : integer;
Begin
  Khoi_tao;
  repeat
    p := Co_dinh_nhan;
    Sua_nhan(p);
  until p = t;
End;
Begin
  Doc;
  Dijkstra;
  Ghi;
End.
```

4, Thuật toán Dijkstra với cấu trúc Heap

Cấu trúc Heap và một số phép xử lí trên Heap

* *Mô tả Heap:* Heap được mô tả như một cây nhị phân có cấu trúc sao cho giá trị

khoá ở mỗi nút không vượt quá giá trị khoá của hai nút con của nó (suy ra giá trị khoá tại gốc Heap là nhỏ nhất).

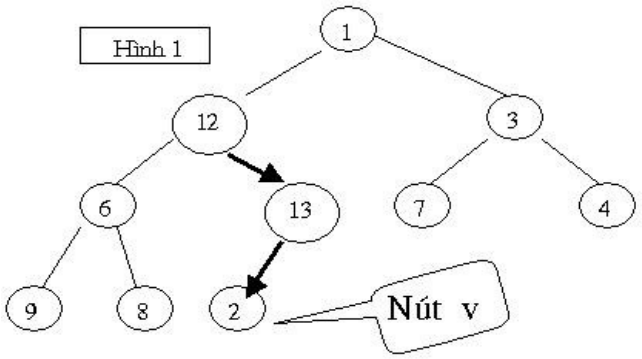
** Hai phép xử lí trên Heap*

- Phép cập nhật Heap

Vấn đề: Giả sử nút v có giá trị khoá nhỏ đi, cần chuyển nút v đến vị trí mới trên Heap để bảo toàn cấu trúc Heap

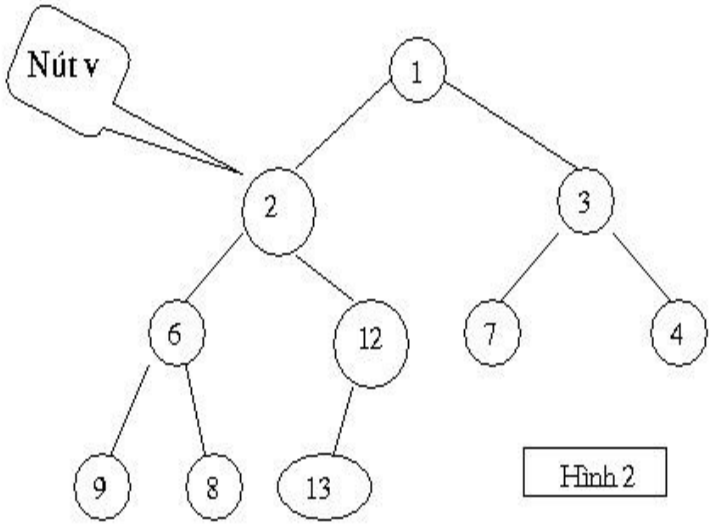
Giải quyết:

+ Nếu nút v chưa có trong Heap thì tạo thêm nút v thành nút cuối cùng của Heap (*hình 1*)



+ Chuyển nút v từ vị trí hiện tại đến vị trí thích hợp bằng cách tìm đường đi ngược từ vị trí hiện tại của v về phía gốc qua các nút cha có giá trị khoá lớn hơn giá trị khoá của v. Trên đường đi ấy dồn nút cha xuống nút con, nút cha cuối cùng chính là vị trí mới
của nút v (*hình 2*).

Chú ý: trên cây nhị phân, nếu đánh số các nút từ gốc đến lá và từ con trái sang con phải thì dễ thấy: khi biết số hiệu của nút cha là i có thể suy ra số hiệu hai nút con là 2*i và 2*i+1, ngược lại số hiệu nút con là j thì số hiệu nút cha là j div 2.



- Phép loại bỏ gốc của Heap

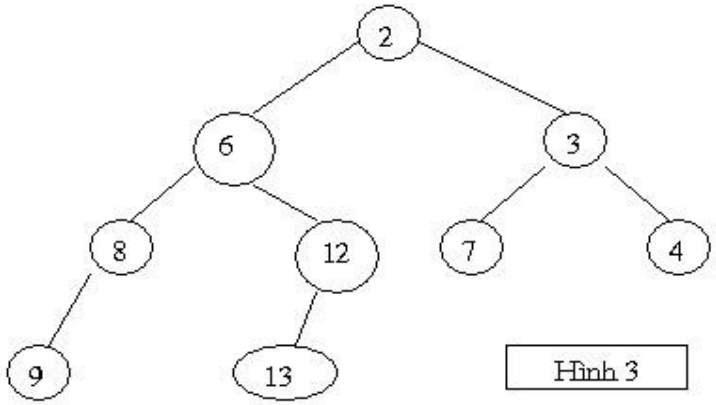
Vấn đề: Giả sử cần loại bỏ nút gốc khỏi Heap, hãy sắp xếp lại Heap (gọi là phép vun đống)

Giải quyết:

+ Tìm đường đi từ gốc về phía lá, đi qua các nút con có giá trị khoá nhỏ hơn trong hai nút con cho đến khi gặp lá.

+ Trên dọc đường đi ấy, kéo nút con lên vị trí nút cha của nó.

Ví dụ trong hình vẽ 2 nếu bỏ nút gốc có khoá bằng 1, ta sẽ kéo nút con lên vị trí nút cha trên đường đi qua các nút có giá trị khoá là 1, 2, 6, 8 và Heap mới như hình 3



Hình 3

Thuật toán Dijkstra tổ chức trên cấu trúc Heap (tạm kí hiệu là Dijkstra_Heap)

Tổ chức Heap: Heap gồm các nút là các đỉnh i tự do (chưa cố định nhãn đường đi ngắn nhất), với khoá là nhãn đường đi ngắn nhất từ s đến i là d[i]. Nút gốc chính là đỉnh tự do có nhãn d[i] nhỏ nhất. Mỗi lần lấy nút gốc ra để cố định nhãn của nó và sửa nhãn cho các đỉnh tự do khác thì phải thức hiện hai loại xử lí Heap đã nêu (phép cập nhật và phép loại bỏ gốc).

Vậy thuật toán Dijkstra tổ chức trên Heap như sau:

Cập nhật nút 1 của Heap (tương ứng với nút s có giá trị khoá bằng 0)

Vòng lặp cho đến khi Heap rỗng (không còn nút nào)

Begin

+ Lấy đỉnh u tại nút gốc của Heap (phép loại bỏ gốc Heap)

+ Nếu u= t thì thoát khỏi vòng lặp

+ Đánh dấu u là đỉnh đã được cố định nhãn

+ Duyệt danh sách cung kề tìm các cung có đỉnh đầu bằng u, đỉnh cuối là v

Nếu v là đỉnh tự do và d[v] > d[u] + khoảng cách (u,v) thì

Begin

Sửa nhãn cho v và ghi nhận đỉnh trước v là u

Trên Heap, cập nhật lại nút tương ứng với đỉnh v.

End;

End;

**** Đánh giá***

+ Thuật toán Dijkstra tổ chức như nêu ở mục 1. Có độ phức tạp thuật toán là $O(N^2)$, nên không thể thực hiện trên đồ thị có nhiều đỉnh.
+ Các phép xử lí Heap đã nêu (cập nhật Heap và loại bỏ gốc Heap) cần thực hiện không quá 2.lgM phép so sánh (nếu Heap có M nút). Số M tối đa là N (số đỉnh của đồ thị) và ngày càng nhỏ dần (tới 0). Ngoài ra, nếu đồ thị thưa (số cung ít) thì thao tác tìm đỉnh v kề với đỉnh u là không đáng kể khi ta tổ chức danh sách các cung kề này theo từng đoạn có đỉnh đầu giống nhau (dạng Forward Star). Do đó trên đồ thị thưa, độ phức tạp của Dijkstra_Heap có thể đạt tới O(N. k.lgN) trong đó k không đáng kể so với N
+ *Kết luận*: Trên đồ thị nhiều đỉnh ít cung thì Dijkstra_Heap là thực hiện được trong thời gian có thể chấp nhận.

III, Đường đi ngắn nhất giữa tất cả các cặp đỉnh - Thuật toán Floyd

Ta có thể giải bài toán tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh của đồ thị bằng cách sử dụng *n* lần giải thuật đã Ford –Bellman hoặc Dijkstra , trong đó ta sẽ chọn *s* lần lượt là các đỉnh của đồ thị. Khi đó ta sẽ thu được giải thuật với độ phức tạp là $O(n^4)$ (nếu sử dụng giải thuật Ford - Bellman) hoặc $O(n^3)$ đối với trường hợp đồ thị có trọng số không âm hoặc không có chu trình.
Trong trường hợp tổng quát việc sử dụng giải thuật Ford-Bellman *n* lần không phải là cách làm tốt nhất. Ở đây ta xét giải thuật Floyd giải bài toán trên với độ phức tạp tính toán $O(n^3)$.

Đầu vào là đồ thị cho bởi ma trận trọng số $a[i, j]$, $i, j = 1, 2, ..., n$.
Đầu ra : - Ma trận đường đi ngắn nhất giữa các cặp đỉnh: $d[i, j]$ $(i, j = 1, 2, ..., n)$.
- Ma trận ghi nhận đường đi $tr[i, j]$ $(i, j = 1, 2, ..., n)$ trong đó $tr[i, j]$ ghi nhận đỉnh đi trước đỉnh *j* trong đường đi ngắn nhất từ *i* đến *j*.

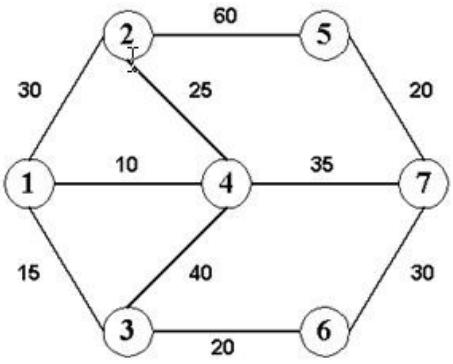
```
Procedure Floyd;
Var i, j, k : integer;
Begin
  { Khởi tạo }
  for i := 1 to n do
    for j := 1 to n do
     begin
       d[i, j] := a[i, j];
       tr[i, j] := i;
     end;

  { Bước lặp }
```

```
  for k := 1 to n do
    for i := 1 to n do
      for j := 1 to n do
        if d[i, j] > d[i, k] + d[k, j] then
          begin
            d[i, j] := d[i, k] + d[k, j];
            tr[i, j] := tr[k, j];
          end;
End;
```

6, Một số bài toán tìm đường đi ngắn nhất

Bài toán 1: 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 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**

- Tệp **Tourist.inp** sẽ chứa một hay nhiều trường hợp test.

- Dòng đầu tiên trong mỗi trường hợp test chứa hai số nguyên N ($N \leq 100$) và R mô tả lần lượt số thành phố và số đường đi giữa các thành phố.

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

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

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

```
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
0 0
```

Tourist.out

```
5
```

Lời giải :

Đây là một bài toán hay, đòi hỏi các bạn phải nắm vững về thuật toán Dijkstra. Bài toán này là bài toán biến thể của bài toán kinh điển tìm đường đi ngắn nhất. Với con đường (u,v) gọi C[u, v] là số người tối đa có thể đi trên con đường đó trong một lần. C[u,v]-1 sẽ là số khách tối đa có thể đi trên con đường đó trong một lần. C[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]. Một chú ý nữa là khi tính số lần đi, các bạn chỉ cần dùng các phép div, mod để tính.

Chương trình thể hiện thuật toán trên (độ phức tạp: n^2)

```
{$R+,Q+}
const
  INP = 'tourist.inp';
  OUT = 'tourist.out';
  maxn = 100;

var
  fi, fo: text;
  c: array [1..maxn, 1..maxn] of longint;
  d: array [1..maxn] of longint;
  chua: array [1..maxn] of boolean;
  n, m, s, t, w: longint;

procedure open_file;
begin
  assign(fi, INP);  reset(fi);
  assign(fo, OUT);  rewrite(fo);
end;

procedure close_file;
begin
  close(fi);
  close(fo);
end;

procedure read_data;
var i, u, v, x: longint;
begin
  readln(fi, n, m);
  for i := 1 to m do
    begin
```

```
    readln(fi, u, v, x);
    c[u, v] := x - 1;
    c[v, u] := x - 1;
  end;
 readln(fi, s, t, w);
end;

function min2(x, y: longint): longint;
begin
 if x > y then min2 := y else min2 := x;
end;

procedure process;
var
 i, max, last: longint;
begin
 fillchar(chua, sizeof(chua), true);
 fillchar(d, sizeof(d), 0);
 chua[s] := false;
 last := s;
 d[s] := maxlongint;
Khởi tạo d[s] = vô cùng hay tất cả mọi người đều có thể đến S cùng lúc
 while chua[t] do
  begin
   for i := 1 to n do
{Tìm các đỉnh i kề với last để cập nhật lại}
     if chua[i] and (d[i] < min2(c[last, i], d[last])) then
       d[i] := min2(c[last, i], d[last]);
   max := -1;
   for i := 1 to n do
     if chua[i] and (d[i] > max) then
       begin
         max := d[i];
         last := i;
       end;
   chua[last] := false;
  end;
```

```
end;

procedure write_result;
begin
  if w mod d[t] = 0 then
    writeln(fo, w div d[t])
  else
    writeln(fo, w div d[t] + 1);
end;

begin
  open_file;
  read_data;
  process;
  write_result;
  close_file;
end.
```

Bài 2: Chuyển Hàng

Bản đồ một kho hàng hình chữ nhật kích thước mxn được chia thành các ô vuông đơn vị (m hàng, n cột: các hàng đánh số từ trên xuống dưới, các cột được đánh số từ trái qua phải). Trên các ô vuông của bản đồ có một số ký hiệu:

- Các ký hiệu # đánh dấu các ô đã có một kiện hàng xếp sẵn.
- Một ký hiệu *: Đánh dấu ô đang có một rôbốt.
- Một ký hiệu $: Đánh dấu ô chứa kiện hàng cần xếp.
- Một ký hiệu @: Đánh dấu vị trí mà cần phải xếp kiện hàng $ vào ô đó.
- Các ký hiệu dấu chấm ".": Cho biết ô đó trống.

Tại môt thời điểm, rô bốt có thể thực hiện một trong số 6 động tác ký hiệu là:

- L, R, U, D: Tương ứng với phép di chuyển của rô bốt trên bản đồ: sang trái, sang phải, lên trên, xuống dưới. Thực hiện một phép di chuyển mất 1 công
- +, − : Chỉ thực hiện khi rôbốt đứng ở ô bên cạnh kiện hàng $. Khi thực hiện thao tác +, rôbốt đứng yên và đẩy kiện hàng $ làm kiện hàng này trượt theo hướng đẩy, đến khi chạm một kiện hàng khác hoặc tường nhà kho thì dừng lại. Khi thực hiện thao tác − , rô bốt kéo kiện hàng $ về phía mình và lùi lại 1 ô theo hướng kéo. Thực hiện thao tác đẩy hoặc kéo mất C công. Rô bốt chỉ được di chuyển vào ô không chứa kiện hàng của kho.

Hãy tìm cách hướng dẫn rôbốt thực hiện các thao tác để đưa kiện hàng $ về vị trí @ sao

cho số công phải dùng là ít nhất.

Dữ liệu: Vào từ file văn bản CARGO.INP

- Dòng 1: Ghi ba số nguyên dương m, n, C (m, n ≤ 100; C ≤ 100)
- m dòng tiếp theo, dòng thứ i ghi đủ n ký kiệu trên hàng i của bản đồ theo đúng thứ tự trái qua phải.

Các ký hiệu được ghi liền nhau.

Kết quả: Ghi ra file văn bản CARGO.OUT

- Dòng 1: Ghi số công cần thực hiện
- Dòng 2: Một dãy liên tiếp các ký tự thuộc {L, R, U, D, +, -} thể hiện các động tác cần thực hiện của rô bốt.

Rằng buộc: Luôn có phương án thực hiện yêu cầu đề bài.

Ví dụ:

CARGO.INP	CARGO.OUT
6 8 3 ###. .### *$ . . . .## #### .### #### . .## #@. . . .## ########	23 +RRRR-UR+DDDRD+

Phân tích:

Thuật toán: Ta sẽ dùng thuật toán Dijkstra để giải bài toán này.

*** Mô hình đồ thị:**

Mỗi đỉnh của đồ thị ở đây gồm 3 trường để phân biệt với các đỉnh khác:

- i: Tọa độ dòng của kiện hàng (i = 1..m)
- j: Tọa độ cột của kiện hàng (j = 1..n)
- h: Hướng của rô bốt đứng cạnh kiện hàng so với kiện hàng (h = 1..4: Bắc, Đông, Nam, Tây).

	1	
2	$	3
	4	

Bạn có thể quan niệm mỗi đỉnh là (i,j,u,v): trong đó i,j: tọa độ của kiện hàng; u,v: tọa độ của rôbốt đứng cạnh kiện hàng. Nhưng làm thế sẽ rất lãng phí bộ nhớ và không chạy hết được dữ liệu. Ta chỉ cần biết hướng h của rôbốt so với kiện hàng là có thể tính được tọa độ của rôbốt bằng cách dùng 2 hằng mảng lưu các số ra:

dx : array[1..4] of integer = (-1,0,1,0)

dy : array[1..4] of integer = (0,1,0,-1)
Khi đó, tọa độ(u,v) của rôbốt sẽ là : u := i + dx[h]; v := j + dy[h];

- Hai đỉnh (i1,j1,h1) và (i2,j2,h2) được gọi là kề nhau nếu qua 1 trong 2 thao tác + hoặc
- kiện hàng được rôbốt đẩy hoặc kéo từ ô (i1, j1) đến ô (i2, j2) và rôbốt có thể di chuyển được từ ô (u1,v1) đến ô (u2,v2) (u1 = i1+dx[h1]; v1=j1+dy[h1]; u2=i2+dx[h2]; v2= j2+dy[h2]). Tất nhiên các ô (i2,j2) và (u2,v2) phải đều không chứa kiện hàng.
- Trọng số giữa 2 đỉnh là C (số công mà rô bốt đẩy kiện hàng từ ô (i1,j1) đến ô (i2,j2)) cộng với công để rô bốt di chuyển từ ô (u1,v1) đến ô (u2,v2).

Giả sử kiện hàng cần xếp đang ở ô (is,js) và hướng của rôbốt đứng cạnh kiện hàng là hs và ô cần xếp kiện hàng vào là ô (ie, je). Khi đó, ta sẽ dùng thuật toán Dijkstra để tìm đường đi ngắn nhất từ đỉnh (is,js,hs) đến đỉnh (ie,je,he) với he thuộc {1..4}.
Mảng d sẽ là 1 mảng 3 chiều: d[i,j,h]: Độ dài đường đi ngắn nhất từ đỉnh xuất phát (is,js,hs) đến đỉnh (i,j,h). Kết quả của bài toán sẽ là d[ie,je,he] với he thuộc {1..4}.
Để ghi nhận phương án ta sẽ dùng 3 mảng 3 chiều tr1, tr2, tr3. Khi ta di từ đỉnh (i1,j1,h1) đến đỉnh (i2,j2,h2) thì ta sẽ gán: tr1[i2,j2,h2]:= i1; tr2[i2,j2,h2]:= j1; tr3[i2,j2,h2] := h1 để ghi nhận các thông tin: tọa độ dòng, cột, hướng của dỉnh trước đỉnh (i2,j2,h2). Từ 3 mảng này ta có thể dễ dàng lần lại đường đi.

Bài 3: 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 trưước đó, nhà Trời cho phép 2 "ông bà" đưược 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 ngưười có thể cùng đi qua một hành tinh nếu như họ đến hành tinh đó vào những thời điểm khác nhau.

Dữ liệu Trong file văn bản ONBANGAU.INP gồm
Dòng đầu là 4 số N M S T ($N \le 100$, $1 \le S \ne T \le N$), M là số cầu vồng. M dòng tiếp, mỗi dòng gồm hai số I J L thể hiện có cầu vồng nối giữa hai hành tinh I , J và cầu vồng đó có độ dài là L ($1 \le I \ne J \le N$, $0 < L \le 200$).

Kết quả Ra file văn bản ONBANGAU.OUT, do tính chất cầu vồng, mỗi năm một khác, nên nếu nhưu 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:

ONBANGAU.INP	ONBANGAU.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 đồ thị: Từ hành tinh S(nơi ông Ngâu ở) ta xây dựng bảng SP. Trong đó SP[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). SP[i] = 0 tức là không có đường đi từ hành tinh S đến hành tinh i. Tương tự ta sẽ xây dựng bảng TP, trong đó TP[i] là đường đi ngắn nhất từ hành tinh T đến hành tinh i. Và TP[i] = 0 tức là không có đường đi 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 sẽ tìm hành tinh h sao cho (h khác S và T) và(SP[h] = ST[h]) đạt giá trị nhỏ nhất khác 0. Nếu không có hành tinh h nào thoả mãn thì ta thông báo CRY

Để xây dựng mảng SP và ST ta có rất nhiều giải thuật khác nhau. ở đây ta chọn giải thuật Djkstra tìm đường đi ngắn nhất giữa 2 đỉnh đồ thị.

Chương trình như sau:

```
uses crt;
const MaxN = 101;
fi= 'ONBANGAU.inp';
fo= 'ONBANGAU.out';
var n,m,s,t,h:byte;
a:array[0..MaxN,0..MaxN] of byte;
SP,ST,B:array[0..MaxN] of integer;
f:text;
{*-------------*thnt*------------*}
procedure Init;
```

```
var i,u,v,ts:byte;
begin
fillchar(a,sizeof(a),0);
assign(f,fi);
reset(f);
readln(f,n,m,s,t);
for i:=1 to m do
begin
readln(f,u,v,ts);
a[u,v]:=ts;
a[v,u]:=ts;
end;
close(f);
end;
{*-------------*thnt*------------*}
procedure Build(s:byte);
var tt:array[0..maxN] of byte;
min,i,vtr:integer;
begin
fillchar(tt,sizeof(tt),0);
fillchar(b,sizeof(b),0);
for i:=1 to n do
b[i] := a[s,i];
tt[s]:=1;
min:=0;
while min <> maxint do
begin
min:=maxint; vtr:=0;
for i:=1 to n do
if tt[i] = 0 then
if (b[i] <>0) and (b[i]
begin min:=b[i]; vtr:=i; end;
if vtr <> 0 then tt[vtr]:=1;

for i:=1 to n do
if (tt[i] = 0) then
if a[vtr,i] <> 0 then
```

```
if (b[vtr] + a[vtr,i]
b[i]:=b[vtr] + a[vtr,i];
end;
end;
{*-------------*thnt*------------*}
procedure FindWay;
var i:integer;
begin
build(s); {xay dung mang SP }
SP:=B;
build(t); {xay dung mang ST}
ST:=B;
h:= 0;{hanh tinh can tim}
sp[0]:= Maxint;
for i:=1 to n do
if (SP[i] = ST[i]) then
if (SP[i]<>0) then
if (SP[i] < SP[h]) then
h:=i;
end;
{*-------------*thnt*------------*}
procedure ShowWay;
begin
assign(f,fo);
rewrite(f);
if h <> 0 then writeln(f,h)
else writeln(f,'CRY');
close(f);
end;
{*-------------*thnt*------------*}
begin
Init;
FindWay;
ShowWay;
end.
```

Bài 4: Máy chủ

Một Công ty muốn phát triển một hệ thống mạng máy tính lớn bao gồm các máy chủ cung cấp nhiều loại hình dịch vụ khác nhau.

Mạng dược kết nối từ n máy chủ bởi các kênh nối cho phép truyền tin hai chiều. Hai máy chủ có thể được nối trực tiếp với nhau bởi không quá một kênh nối. Mỗi máy chủ được nối trực tiếp với không quá 10 máy chủ khác và hai máy chủ bất kỳ luôn có thể kết nối được với nhau hoặc thông qua một kênh nối trực tiếp giữa chúng hoặc thông qua các máy chủ khác. Đối với kênh nối ta biết thời gian truyền thông được đo bằng mili giấy là một số dương. Khoảng cách (tính bằng mili giấy) d(u,v) giữa máy chủ u và máy chủ v được xác định như là độ dài của đường đi ngắn nhất (ứng với thời gian truyền thông trên cạnh) nối u và v trên mạng. Để tiện dùng, ta qui ước d(v,v)=0 với mọi v.

Có một số máy chủ cung cấp nhiều dịch vụ hơn các máy chủ khác. Vì thế mỗi máy chủ v được gán với một số tự nhiên r(v) gọi là hạng của nó. Máy chủ có hạng càng cao càng là máy chủ mạnh hơn. Tại mỗi máy chủ dữ liệu về các máy chủ gần nó thường được cất giữ. Tuy nhiên không phải máy chủ nào cũng là đáng quan tâm. Dữ liệu về các máy chủ lân cận với hạng thấp hơn không được cất giữ. Chính xác hơn, *máy chủ w là đáng quan tâm đối với máy chủ v* nếu với mọi máy chủ u sao cho d(v,u)≤d(v,w) ta có r(u)≤r(w). Chẳng hạn, tất các các máy chủ với hạng lớn nhất đều là đáng quan tâm đối với tất cả các máy chủ. Nếu máy chủ v có hạng lớn nhất thì rõ ràng chỉ có các máy chủ với hạng lớn nhất mới là đáng quan tâm đối với v.

Gọi B(v) là tập các máy chủ đáng quan tâm đối với máy chủ v. Ta gọi *kích thước dữ liệu về các máy chủ đáng quan tâm đối với máy chủ v là* $|B(v)|$.

Yêu cầu: Tính tổng kích thước dữ liệu về các máy chủ đáng quan tâm của tất cả các máy chủ trong toàn mạng. Biết rằng tổng này có giá trị không vượt quá 30n.

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

+ Dòng đầu chứa hai số n, m trong đó n là số máy chủ (1≤n≤3000) và m là số kênh nối (1≤m≤5n)

+ Dòng thứ i trong số n dòng tiếp theo chứa r_i (1≤r_i≤10) là hạng của máy chủ i.

+ Tiếp đến là m dòng, mỗi dòng chứa thông tin về một kênh nối bao gồm a, b, t (1≤t≤1000, 1≤a,b≤n, a≠b), trong đó a, b là chỉ số của hai máy chủ được nối bởi kênh đang xét còn t là thời gian truyền thông của kênh (đo bằng mili giây).

Kết quả: Ghi ra file văn bản SERVER.OUT một số nguyên duy nhất là tổng kích thước dữ liệu về các máy chủ đáng quan tâm của tất cả các máy chủ trong toàn mạng

Ví dụ:

SERVER.INP	SERVER.OUT	Giải thích
4 3 2 3	9	Ta có: B(1)={1,2} B(2)={2}

1 1 1 4 30 2 3 20 3 4 20		B(3)={2,3} B(4)={1,2,3,4}

Bài này ta dùng thuật toán Dijkstra để giải chương trình như sau

```
const
  tfi          =    'SERVER.INP';
  tfo          =    'SERVER.OUT';
  maxN         =    3000;
type
  mang1        =    array[1..10] of integer;
  mang2        =    array[1..maxN] of integer;
var
  fi,fo        :    text;
  N,M          :    longint;
  Sol          :    array[1..maxN] of byte;
  r            :    array[1..maxN] of byte;
  a            :    array[1..maxN] of ^mang1;
  d            :    array[1..maxN] of ^mang1;
  kq           :    longint;
  Q            :    array[1..maxN] of longint;
  vt           :    ^mang2;
  qn           :    longint;
  kc           :    array[1..maxN] of longint;
  loai         :    array[1..maxN] of byte;
  Rank         :    array[1..maxN] of byte;
  Q1           :    ^mang2;
  q1n          :    integer;
  kcold        :    longint;
  t1           :    longint;
  t2           :    longint absolute 0:$46c;
  rmax         :    byte;

procedure CapPhat;
var i: longint;
begin
```

```
  for i:=1 to maxN do new(a[i]);
  for i:=1 to maxN do new(d[i]);
  new(q1);
  new(vt);
end;
procedure InitQ;
begin
  qn:=0;
end;
procedure Put(u: longint);
begin
  inc(qn);
  q[qn]:=u;
end;
function Get: longint;
var i,u: longint;
begin
  u:=1;
  for i:=2 to qn do
    if kc[q[i]]<kc[q[u]] then u:=i;
    Get:=q[u];
    q[u]:=q[qn];
    dec(qn);
end;
function Qempty: boolean;
begin
  Qempty:=(qn=0);
end;
procedure Docdl;
var i,u,v,w:longint;
begin
  assign(fi,tfi); reset(fi);
  readln(fi,n,m);
  for i:=1 to N do sol[i]:=0;
  for i:=1 to N do readln(fi,r[i]);
  for i:=1 to M do
    begin
```

```
        readln(fi,u,v,w);
        inc(sol[u]); a[u]^[sol[u]]:=v; d[u]^[sol[u]]:=w;
        inc(sol[v]); a[v]^[sol[v]]:=u; d[v]^[sol[v]]:=w;
      end;
  close(fi);
  rmax:=0;
  for i:=1 to N do
    if rmax<r[i] then rmax:=r[i];
end;

procedure Dijstra(xp: longint);
var i,u,v,ll: longint;
   MaxRank: byte;
begin
  InitQ;
  kcold:=-1;
  for i:=1 to N do loai[i]:=0;
  for i:=1 to N do rank[i]:=rmax;
  MaxRank:=0;
  Put(xp); loai[xp]:=1; kc[xp]:=0;
  repeat
    u:=Get; loai[u]:=2;
    if MaxRank<r[u] then MaxRank:=r[u];
    if kc[u]=kcold then
      begin
        inc(q1n);
        q1^[q1n]:=u;
      end
    else
      begin
        q1n:=1;
        q1^[1]:=u;
      end;
    kcold:=kc[u];
    for i:=1 to q1n do
      Rank[q1^[i]]:=MaxRank;
    if (maxRank<rmax) then
```

```
      for i:=1 to sol[u] do
        begin
          v:=a[u]^[i]; ll:=d[u]^[i];
          if (loai[v]=1) and (kc[v]>kc[u]+ll) then kc[v]:=kc[u]+ll;
          if loai[v]=0 then
            begin
              Loai[v]:=1;
              kc[v]:=kc[u]+ll;
              Put(v);
            end;
        end;
    until Qempty;
end;

function Dem: longint;
var k,i: longint;
begin
   k:=0;
   for i:=1 to N do
     if (Rank[i]<=r[i]) then k:=k+1;
   Dem:=k;
end;

procedure Solve;
var i,k: longint;
begin
   kq:=0;
   for i:=1 to N do
     begin
       Dijstra(i);
       kq:=kq+Dem;
     end;
end;

procedure Inkq;
begin
   assign(fo,tfo); rewrite(fo);
```

```
    writeln(fo,kq);
    close(fo);
  end;

  BEGIN
    clrscr;
    t1:=t2;
    CapPhat;
    Docdl;
    Solve;
    Inkq;
    writeln('Total time =',(t2-t1)/18.3:0:4,' s');
    readln;
END.
```

Bài 5: Hành trình trên xe lửa

Lịch hoạt động của tuyến đường sắt trong một ngày bao gồm thông tin của từng chuyến tầu có trong ngày đó. Thông tin của mỗi chuyến tầu bao gồm:

- Số hiệu chuyến tầu (được đánh số từ 1 đến *M*),
- Danh sách các ga mà chuyến tầu đó dừng lại, mỗi ga bao gồm:
  + Số hiệu ga (các ga được đánh số từ 1 trở đi),
  + Giờ đến (số thực),
  + Giờ đi (số thực).

Các giá trị thời gian tính theo đơn vị giờ và viết dưới dạng thập phân (ví dụ 7.5 có nghĩa là 7 giờ 30 phút).

Một hành khách bất kỳ khi đi đến một ga nào đó (gọi là ga hiện tại) cho biết yêu cầu của mình gồm: thời điểm mà từ đó anh ta có thể đi được, số hiệu ga cần đến và thời gian tối thiểu cho mỗi lần chuyển tầu. Nhân viên nhà ga phải trả lời được là có đáp ứng được yêu cầu của khách không? Nếu đáp ứng được, nhân viên nhà ga phải đưa ra được hành trình cần đi cho khách.

Hãy giải bài toán trong 2 trường hợp:

a. Tìm hành trình đến ga cuối cùng sớm nhất.

b. Tìm hành trình ít phải chuyển tầu nhất. Nếu tồn tại nhiều phương án như vậy, hãy tìm phương án đến ga cuối cùng sớm nhất.

Dữ liệu: File vào gồm các dòng:

- Dòng 1: Ghi 4 số theo thứ tự: thời điểm đi, ga hiện tại, ga cần đến và thời gian tối đa cho mỗi lần chuyển tầu;
- Dòng 2: Ghi số nguyên dương M ($M \leq 50$);

- Dòng i+2 (i = 1, 2, ..., M): Ghi thông tin của chuyến tầu số hiệu i bao gồm: số lượng ga mà chuyến tầu đó dừng lại (≤ 20), danh sách các ga theo trình tự đi đến của chuyến tầu, trong đó mỗi ga được mô tả bởi 3 số theo thứ tự: số hiệu ga, giờ đến, giờ đi.

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

Kết quả: Trong trường hợp không tìm thấy hành trình thì ghi giá trị 0. Trái lại, ghi hành trình tìm được dưới dạng sau:

- Dòng đầu ghi S là số hiệu chuyến tầu mà khách bắt đầu đi,
- Dòng tiếp ghi T_1 là thời điểm đi của chuyến tầu này,
- Dòng tiếp ghi K là số lần khách phải chuyển tầu,
- K dòng tiếp, mỗi dòng ghi thông tin của một lần chuyển tầu gồm số hiệu ga mà khách phải chuyển tầu và số hiệu chuyến tầu cần đi tiếp (ghi cách nhau một dấu trắng),
- Dòng cuối ghi T_2 là thời điểm đến ga cuối cùng của hành trình.

Kết quả của câu a và câu b ghi cách nhau bởi 1 dòng trắng.

Ví dụ:

XELUA.INP	XELUA.OUT
6 4 3 1.5 6 3 1 7 7 2 8 9.1 3 9.5 9.5 2 4 6 6 2 7 7.5 2 2 7.5 7.5 5 8 8 3 6 8 8 5 9 9.5 3 10 10 3 4 6.5 6.5 7 9 9.5 3 11 11 2 4 7 7 3 12 12	26.02 2 3 5 4 10.0 5 6.5 0 11.0

Chương trình như sau

```
Const
  FI = 'xelua.inp';
  FO = 'xelua.out';
  MAX_VALUE = 999999999;
Var
  n, nU, ga_di, ga_den, dem : integer;
  t0, t_di, t_cho : real;
  tau, ga, tr, U : array[0..1001] of integer;
  d, gio_den, gio_di : array[0..1001] of real;
  f : text;
Procedure Doc;
```

```
Var m, i, j, k : integer;
Begin
  assign(f, FI); reset(f);
  read(f, t_di, ga_di, ga_den, t_cho, m);
  tau[0] := 0;
  ga[0] := ga_di;
  gio_den[0] := t_di;
  gio_di[0] := t_di;
  n := 0;
  for i := 1 to m do
    begin
      read(f, k);
      for j := 1 to k do
        begin
          n := n + 1;
          tau[n] := i;
          read(f, ga[n], gio_den[n], gio_di[n]);
        end;
    end;
  close(f);
End;
Function Khoang_cach(i, j : integer) : real;
Var t : real;
Begin
  if tau[i] = tau[j] then
    begin
      t := gio_di[i] - gio_den[i];
      if (j = i+1) and (t <= t_cho) then Khoang_cach := gio_den[j] - gio_den[i]
                                    else Khoang_cach := MAX_VALUE;
    end
  else
    if ga[i] = ga[j] then
      begin
        t := gio_di[j] - gio_den[i];
        if (t >= 0) and (t <= t_cho) then Khoang_cach := t + t0
                                     else Khoang_cach := MAX_VALUE;
      end
```

```
    else Khoang_cach := MAX_VALUE;
End;
Procedure Khoi_tao;
Var i : integer;
Begin
  for i := 0 to n do
    begin
      d[i] := Khoang_cach(0, i);
      tr[i] := 0;
    end;
  nU := n;
  for i := 1 to nU do U[i] := i;
End;

Function Co_dinh_nhan : integer;
Var i, j : integer;
Begin
  i := 1;
  for j := 2 to nU do
    if d[U[j]] < d[U[i]] then i := j;
  Co_dinh_nhan := U[i];
  U[i] := U[nU];
  nU := nU - 1;
End;
Procedure Sua_nhan(p : integer);
Var
  x, i : integer;
  kc : real;
Begin
  for i := 1 to nU do
    begin
      x := U[i];
      kc := Khoang_cach(p, x);
      if d[x] > d[p] + kc then
        begin
          d[x] := d[p] + kc;
          tr[x] := p;
```

```
        end;
    end;
End;
Procedure Print(i : integer);
Begin
  if tr[i] = 0 then
   begin
     writeln(f, tau[i]);
     writeln(f, gio_di[i] : 0 : 1);
     writeln(f, dem);
     exit;
   end;
  if tau[tr[i]] <> tau[i] then dem := dem + 1;
  Print(tr[i]);
  if tau[tr[i]] <> tau[i] then writeln(f, ga[i], ' ', tau[i]);
End;
Procedure Ghi;
Var
  dich, i : integer;
  som_nhat : real;
Begin
  som_nhat := MAX_VALUE;
  for i := 1 to n do
    if (ga[i] = ga_den) and (d[i] < som_nhat) then
      begin
        som_nhat := d[i];
        dich := i;
      end;
  if som_nhat = MAX_VALUE then writeln(f, 0)
  else
    begin
      dem := 0;
      Print(dich);
      writeln(f, gio_den[dich] : 0 : 1);
    end;
  writeln(f);
End;
```

```
Procedure Dijktra;
Var p : integer;
Begin
  Khoi_tao;
  while nU > 0 do
    begin
      p := Co_dinh_nhan;
      Sua_nhan(p);
    end;
  Ghi;
End;
Procedure Xu_ly;
Begin
  assign(f, fo); rewrite(f);
  { Cau a }
  t0 := 0;
  Dijktra;
  { Cau b }
  t0 := 9999;
  Dijktra;
  close(f);
End;
Begin
  Doc;
  Xu_ly;
End.
```

Bài 6: Hội thảo trực tuyến

Một trung tâm quản trị một mạng gồm N (≤ 100) cổng truy cập được đánh số từ 1 đến N. Giữa hai cổng có thể không có đường nối hoặc có đường nối trực tiếp và thông tin truyền hai chiều trên đường nối. Mạng có M đường nối trực tiếp giữa các cổng và nếu đường nối trực tiếp giữa hai cổng i, j được sử dụng thì chi phí truyền tin phải trả là c_{ij} ($\leq$ 32767).

Trung tâm nhận được hợp đồng tổ chức một cuộc hội thảo trực tuyến từ 3 địa điểm khác nhau truy cập vào mạng từ 3 cổng. Bạn hãy giúp công ty tổ chức sử dụng các đường nối truyền tin sao cho tổng chi phí là ít nhất có thể được.

Dữ liệu: File vào gồm các dòng:

- Dòng đầu tiên ghi hai số *N* và *M*;
- *M* dòng tiếp theo, mỗi dòng chứa 3 số nguyên dương trong đó 2 số đầu là chỉ số của hai cổng, số thứ 3 là chi phí khi truyền tin trên hai cổng đó;
- Dòng cuối cùng chứa 3 số nguyên dương theo thứ tự là chỉ số của 3 cổng tại 3 địa điểm của hội thảo.

Kết quả: File ra gồm:
- Dòng đầu ghi xâu 'No' nếu không thể tổ chức hội thảo trực tuyến được, ngược lại ghi 'Yes'.
- Nếu tìm được cách tổ chức thì dòng thứ hai ghi *S* là chi phí nhỏ nhất tìm được và dòng thứ ba ghi *P* là số đường nối cần sử dụng. *P* dòng tiếp theo mỗi dòng ghi hai số *i*, *j* thể hiện một đường nối giữa hai cổng *i* và *j* được sử dụng. Các số trên một dòng ghi cách nhau bởi một dấu cách.

Ví dụ:

NET.INP	NET.OUT
8 12 1 2 20 2 3 8 2 4 3 2 5 3 2 6 6 3 5 2 3 6 9 4 7 5 5 6 1 5 7 7 6 8 4 7 8 6 1 4 6	Yes 27 4 1 2 2 4 2 5 5 6

Lời giải: Chúng ta thấy rằng chắc chắn đoạn nối đó phải là một cây . Tức là sẽ có một cây đồ thị bao lấy ba địa điểm đó . Mà cây đó là cây có độ dài nhỏ nhất . Vì vậy tồn tại một điểm là trung gian T (có thể trùng với 1 trong ba địa điểm đó) . Thì tổng đường truyền từ T đến 3 đỉnh đó phải nhỏ nhất . Tức là ta sẽ dùng thuật toán Floyd . Sau đó tìm đỉnh nào có tổng khoảng cách nhỏ nhất đến ba đỉnh làn nhỏ nhất thì các đường nối đó chính là các đường nối thoả mãn .

Chương trình

```
Program Hoi_thao_truc_tuyen;
Uses crt;
Const
```

```
  FI = 'net.inp';
  FO = 'net.out';
  MAX_N = 100;
  MAX_VALUE = 999999999;
Var
  n, x, y, z, sum, so_canh : integer;
  c : array[1..MAX_N, 1..MAX_N] of longint;
  tr : array[1..MAX_N, 1..MAX_N] of byte;
  f : text;

Procedure Doc;
Var m, chi_phi, i, j, k : integer;
Begin
  assign(f, FI); reset(f);

  readln(f, n, m);
  for i := 1 to n do
    for j := 1 to n do c[i, j] := MAX_VALUE;

  for k := 1 to m do
    begin
      readln(f, i, j, chi_phi);
      c[i, j] := chi_phi;
      c[j, i] := chi_phi;
    end;
  readln(f, x, y, z);

  close(f);
End;

Procedure Floyd;
Var i, j, k : integer;
Begin
  for i := 1 to n do
    for j := 1 to n do tr[i, j] := i;

  for k := 1 to n do
```

```
    for i := 1 to n do
      for j := 1 to n do
        if c[i, j] > c[i, k] + c[k, j] then
          begin
            c[i, j] := c[i, k] + c[k, j];
            tr[i, j] := tr[k, j];
          end;
End;

Procedure Print(i, j : integer);
Begin
  if i = j then exit;
  if c[tr[i, j], j] <> -1 then
    begin
      so_canh := so_canh + 1;
      sum := sum + c[tr[i, j], j];
      c[tr[i, j], j] := -1;
      c[j, tr[i, j]] := -1;
    end;
  Print(i, tr[i, j]);
End;
Procedure Ghi;
Var min, t, i, j : longint;
Begin
  assign(f, FO); rewrite(f);
  min := MAX_VALUE;
  for i := 1 to n do
    if min > c[x, i] + c[y, i] + c[z, i] then
      begin
        t := i;
        min := c[x, i] + c[y, i] + c[z, i];
      end;
  if min = MAX_VALUE then write(f, 'No')
  else
    begin
      so_canh := 0;
      sum := 0;
```

```
      Print(x, t);
      Print(y, t);
      Print(z, t);
      writeln(f, 'Yes');
      writeln(f, sum);
      writeln(f, so_canh);
      for i := 1 to n do
        for j := i+1 to n do
          if c[i, j] = -1 then writeln(f, i, ' ', j);
    end;
  close(f);
End;
Begin
  Doc;
  Floyd;
  Ghi;
End.
```

Bài 7: Chợ trung tâm

Có N địa điểm dân cư đánh số từ 1 đến N. Giữa M cặp địa điểm trong số N địa điểm nói trên có tuyến đường nối chúng. Cần xây dựng một trung tâm dịch vụ tổng hợp tại một địa điểm trùng với một địa điểm dân cư, sao cho tổng khoảng cách từ trung tâm dịch vụ đến N địa điểm dân cư là nhỏ nhất. Ta gọi khoảng cách giữa hai địa điểm là độ dài đường đi ngắn nhất nối chúng. Giả sử N địa điểm trên là liên thông với nhau. Nếu có nhiều phương án thì đưa ra phương án đặt trung tâm dịch vụ tại địa điểm có số hiệu nhỏ nhất.

Dữ liệu: File vào gồm M+1 dòng:
- Dòng 1: Chứa hai số nguyên dương N và M ($N \le 100$);
- Dòng i+1 ($1 \le i \le M$): Chứa 3 số nguyên dương x, y, z, ở đó hai số đầu x, y là số hiệu của hai địa điểm dân cư được nối với nhau bởi tuyến đường này, còn số thứ ba z ($\le$ 32767) là độ dài của tuyến đường này.

Kết quả: File ra gồm 2 dòng:
- Dòng 1: Ghi vị trí trung tâm dịch vụ;
- Dòng 2: Ghi tổng khoảng cách từ trung tâm dịch vụ đến các địa điểm dân cư.

Ví dụ:

MARKET.INP	MARKET.OUT

5 7 1 2 9 2 3 4 1 4 2 4 5 5 5 3 1 5 1 5 3 1 4	3 15

```
Program Cho_trung_tam;
Uses crt;
Const
  FI = 'market.inp';
  FO = 'market.out';
  MAX_N = 100;
  MAX_VALUE = 999999999;
Var
  n, dia_diem, min : longint;
  d : array[1..MAX_N, 1..MAX_N] of longint;
  f : text;

Procedure Doc;
Var i, j, k, m : integer;
Begin
  assign(f, FI); reset(f);

  read(f, n, m);
  for i := 1 to n do
    begin
      d[i, i] := 0;
      for j := i+1 to n do
        begin
          d[i, j] := MAX_VALUE;
          d[j, i] := MAX_VALUE;
        end;
    end;
  for k := 1 to m do
    begin
```

```
      read(f, i, j);
      read(f, d[i, j]);
      d[j, i] := d[i, j];
    end;

  close(f);
End;

Procedure Floyd;
Var sum, i, j, k : longint;
Begin
  for k := 1 to n do
    for i := 1 to n do
      for j := 1 to n do
        if d[i, j] > d[i, k] + d[k, j] then d[i, j] := d[i, k] + d[k, j];

  min := MAX_VALUE;
  for i := 1 to n do
    begin
      sum := 0;
      for j := 1 to n do sum := sum + d[i, j];

      if sum < min then
        begin
          dia_diem := i;
          min := sum;
        end;
    end;
End;

Procedure Ghi;
Begin
  assign(f, FO); rewrite(f);

  writeln(f, dia_diem);
  write(f, min);
```

```
  close(f);
End;

Begin
  Doc;
  Floyd;
  Ghi;
End.
```

Bài 8: 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 phàn nàn 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 từ file văn bản CITY.INP, 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, $n \le 200$; $0 \le m \le n*(n-1)/2$)

+ 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 ($c \le 10000$)

Kết quả: Ghi ra 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.INP, CITY.OUT được ghi cách nhau ít nhất một dấu cách

Ví dụ:

CITY.INP

```
10 12
1 2 1
1 5 2
2 6 7
3 4 1
3 7 2
4 8 8
5 6 3
6 7 1
6 9 2
7 8 5
7 10 8
9 10 4
```

CITY.OUT

```
9 21
1 2
1 5
3 4
3 7
5 6
6 7
6 9
7 8
9 10
```

Chương trình

```
const
  tfi          =    'CITY.INP';
  tfo          =    'CITY.OUT';
  maxN         =    200;
  Unseen       =    2000000;
type
  mangB        =    array[1..maxN] of byte;
  mangL        =    array[1..maxN] of LongInt;
var
  fi,fo        :    text;
  N,M          :    LongInt;
  a            :    array[1..maxN] of ^mangL;
  Gr           :    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(Gr[i]);
end;
procedure GiaiPhong;
```

```
var i: integer;
begin
  for i:=1 to maxN do Dispose(a[i]);
  for i:=1 to maxN do Dispose(Gr[i]);
end;
procedure Docdl;
var i,j,u,v,l: LongInt;
begin
  assign(fi,tfi); reset(fi);
  readln(fi,N,M);
  for i:=1 to N do
    for j:=1 to N do a[i]^[j]:=Unseen;
  for i:=1 to N do
    for j:=1 to N do Gr[i]^[j]:=0;
  for i:=1 to M do
    begin
      readln(fi,u,v,l);
      a[u]^[v]:=l; a[v]^[u]:=l;
      Gr[u]^[v]:=1; Gr[v]^[u]:=1;
    end;
  close(fi);
end;
procedure Floyd;
var k,i,j: integer;
begin
  Fillchar(Tr,sizeof(Tr),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];
            Tr[i,j]:=k;
          end;
end;

procedure Solve;
```

```
var i,j: LongInt;
begin
  for i:=1 to N do
    for j:=1 to N do
      if (Gr[i]^[j]=1) and (Tr[i,j]>0) then
        begin
          Gr[i]^[j]:=0;
          Gr[j]^[i]:=0;
        end;
  S:=0;
  D:=0;
  for i:=1 to N-1 do
    for j:=i+1 to N do
      if Gr[i]^[j]=1 then
        begin
          S:=S+a[i]^[j];
          D:=D+1;
        end;
end;
procedure inkq;
var i,j: LongInt;
begin
  assign(fo,tfo); rewrite(fo);
  writeln(fo,d,' ',S);
  for i:=1 to N-1 do
    for j:=i+1 to N do
      if Gr[i]^[j]=1 then
        writeln(fo,i,' ',j);
  close(fo);
end;

BEGIN
  CapPhat;
  Docdl;
  Floyd;
  Solve;
  Inkq;
```

```
  GiaiPhong;
END.
```

B. KẾT LUẬN

Để tìm đường đi ngắn nhất trên đồ thị còn có nhiều thuật toán nữa và cũng còn nhiều cách để cài đặt các thuật toán trên hiệu quả hơn. Tuy nhiên trong chuyên đề này tôi chỉ đưa ra các cách cài đặt cơ bản nhất để từ đó học sinh tự nghiên cứu và phát triển thêm. Vì thời gian và trình độ có hạn nên chuyên đề này có thể còn nhiều hạn chế, thiếu sót mong các đồng nghiệp và các em học sinh góp ý.

C. TÀI LIỆU THAM KHẢO

1, Tài liệu chuyên tin quyển 2 – Hồ Sỹ Đàm

2, Giải thuật và lập trình – Lê Minh Hoàng

3, Một số tài liệu khác của các đồng nghiệp