TEAM 4

544 - Heavy Cargo

100703013 洪詠威
100703014 張傢凱
100703044 江岱蓉

Problem

- *Big Johnsson Trucks Inc.* is a company specialized in manufacturing big trucks. Their latest model, the *Godzilla V12*, is so big that the amount of cargo you can transport with it is never limited by the truck itself. It is only limited by the weight restrictions that apply for the roads along the path you want to drive.
- Given start and destination city, your job is to determine the maximum load of the *Godzilla V12* so that there still exists a path between the two specified cities.

Input

- The input file will contain one or more test cases. The first line of each test case will contain two integers: the number of cities *n* (2~200) and the number of road segments *r* (1~19900) making up the street network.
- Then *r* lines will follow, each one describing one road segment by naming the two cities connected by the segment and giving the weight limit for trucks that use this segment. Weight limits are integers in the range 0 - 10000. Roads can always be travelled in both directions.
- The last line of the test case contains two city names: start and destination.
- Input will be terminated by two values of 0 for *n* and *r*.

Output

For each test case, print three lines:

- a line saying ``Scenario #*x*" where *x* is the number of the test case
- a line saying ``*y* tons" where *y* is the maximum possible load
- a blank line

Sample Input/Output

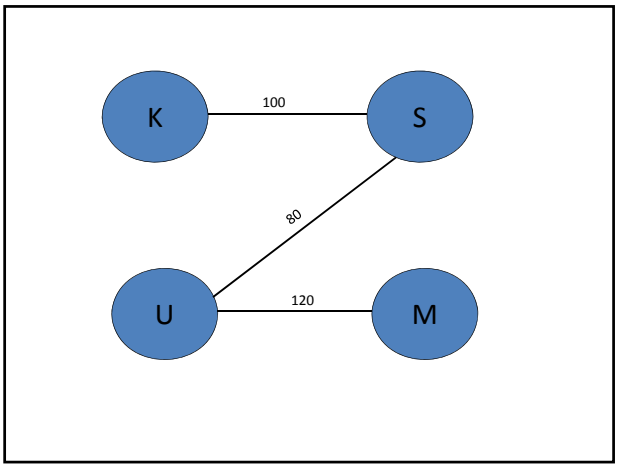
- 4 3
- Karlsruhe Stuttgart 100
- Stuttgart Ulm 80
- Ulm Muenchen 120
- Karlsruhe Muenchen
- 5 5
- Karlsruhe Stuttgart 100
- Stuttgart Ulm 80
- Ulm Muenchen 120
- Karlsruhe Hamburg 220
- Hamburg Muenchen 170
- Muenchen Karlsruhe
- 0 0

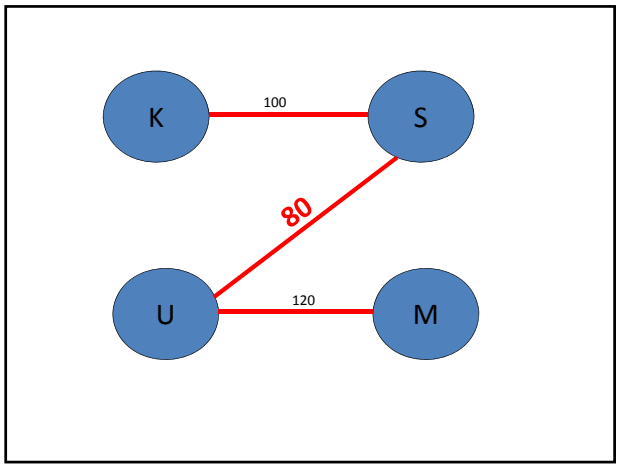
Sample Input/Output

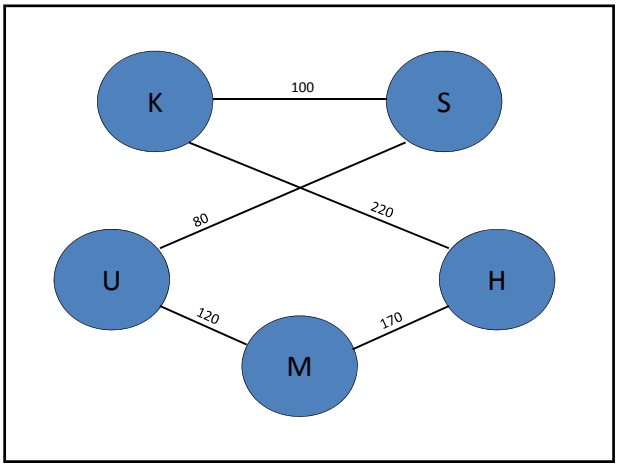
- 4 3
- Karlsruhe Stuttgart 100
- Stuttgart Ulm 80
- Ulm Muenchen 120
- Karlsruhe Muenchen
- 5 5
- Karlsruhe Stuttgart 100
- Stuttgart Ulm 80
- Ulm Muenchen 120
- Karlsruhe Hamburg 220
- Hamburg Muenchen 170
- Muenchen Karlsruhe
- 0 0

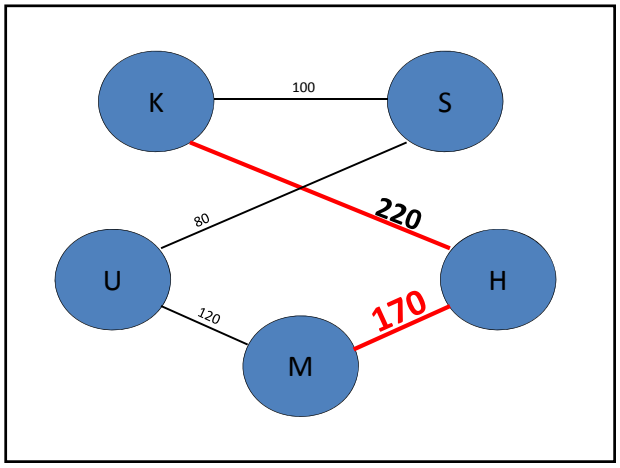
- Scenario #1
- 80 tons

- Scenario #2
- 170 tons









Solution 1

```
while ( scanf ("%d %d", &n, &r) ) {
        if ( n == 0 && r == 0 ) break;

        map <string, int> cityIndex;
        string first, second;
        int cost;
        int index = 1;
        int d [200] [200];
```

1. map <string, int> cityIndex的意思是cityIndex[string]=int

   (例: cityIndex[Taipei]=6;

   cityIndex[Taichung]=7;

   代表Taipei是輸入的第6個程式，Taichung是第7個)

```
while ( scanf ("%d %d", &n, &r) ) {
        if ( n == 0 && r == 0 ) break;

        map <string, int> cityIndex;
        string first, second;
        int cost;
        int index = 1;
        int d [200] [200];
```

2. 宣告string first和string second來存輸入的城市，cost為兩個程式間重量最大承受量的input，d[200][200]來存城市到城市間的重量最大承受量

假設

cityIndex[Miami]=1;

cityIndex[Berlin]=2;

cityIndex[London]=3;

cityIndex[Tokyo]=4;

cityIndex[Paris]=5;

cityIndex[Taipei]=6;

d[200][200]

	1	2	3	4	5	6
1	Miami ↓ Miami	Miami ↓ Berlin	Miami ↓ London	Miami ↓ Tokyo	Miami ↓ Paris	Miami ↓ Taipei
2	Berlin ↓ Miami	Berlin ↓ Berlin	Berlin ↓ London	Berlin ↓ Tokyo	Berlin ↓ Paris	Berlin ↓ Taipei
3	London ↓ Miami	London ↓ Berlin	London ↓ London	London ↓ Tokyo	London ↓ Paris	London ↓ Taipei
4	Tokyo ↓ Miami	Tokyo ↓ Berlin	Tokyo ↓ London	Tokyo ↓ Tokyo	Tokyo ↓ Paris	Tokyo ↓ Taipei
5	Paris ↓ Miami	Paris ↓ Berlin	Paris ↓ London	Paris ↓ Tokyo	Paris ↓ Paris	Paris ↓ Taipei
6	Taipei ↓ Miami	Taipei ↓ Berlin	Taipei ↓ London	Taipei ↓ Tokyo	Taipei ↓ Paris	Taipei ↓ Taipei

```
for ( int i = 0; i < 200; i++ ) {
    for ( int j = 0; j < 200; j++ ) {
        d [i] [j] = -1;
        d [i] [i] = 0;
    }
}
```

3. 先把所有最大承受量設成-1

自己和自己之間為0

	1	2	3	4	5	6
1	0	-1	-1	-1	-1	-1
2	-1	0	-1	-1	-1	-1
3	-1	-1	0	-1	-1	-1
4	-1	-1	-1	0	-1	-1
5	-1	-1	-1	-1	0	-1
6	-1	-1	-1	-1	-1	0

```
for ( int i = 0; i < r; i++ ) {
    cin >> first >> second >> cost;
    if ( !cityIndex [first] )
        cityIndex [first] = index++;
    if ( !cityIndex [second] )
        cityIndex [second] = index++;

    d [cityIndex [first]] [cityIndex [second]] = cost;
    d [cityIndex [second]] [cityIndex [first]] = cost;
}
```

4. 判斷輸入的城市之前有沒有出現過，若沒有則將程式列入排序

(例: cityIndex[Taipei]=6, cityIndex[Taichung]=7)

5. 將輸入的重量最大承受量cost存入d[200][200]裡面

整個程式最重點的部分!!!

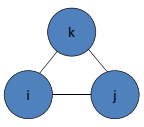
6. 計算i j兩個程式之間的重量最大承受量

```
for ( int k = 1; k <= n; k++ ) {
    for ( int i = 1; i <= n; i++ ) {
        for ( int j = 1; j <= n; j++ ) {
            d [i] [j] = d [j] [i] = max (d [i] [j], min (d [i] [k], d [k] [j]));
        }
    }
}
```

整個程式最重點的部分!!!

6.計算i j兩個程式之間的重量最大承受量

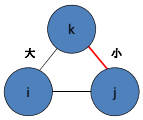
```
for ( int k = 1; k <= n; k++ ) {
    for ( int i = 1; i <= n; i++ ) {
        for ( int j = 1; j <= n; j++ ) {
            d [i] [j] = d [j] [i] = max (d [i] [j], min (d [i] [k], d [k] [j]));
        }
    }
}
```



整個程式最重點的部分!!!

6.計算i j兩個程式之間的重量最大承受量

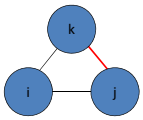
```
for ( int k = 1; k <= n; k++ ) {
    for ( int i = 1; i <= n; i++ )  {
        for ( int j = 1; j <= n; j++ ) {
            d [i] [j] = d [j] [i] = max (d [i] [j], min (d [i] [k], d [k] [j]));
        }
    }
}
```



整個程式最重點的部分!!!

6.計算i j兩個程式之間的重量最大承受量

```
for ( int k = 1; k <= n; k++ ) {
    for ( int i = 1; i <= n; i++ )  {
        for ( int j = 1; j <= n; j++ ) {
            d [i] [j] = d [j] [i] = max (d [i] [j], min (d [i] [k], d [k] [j]));
        }
    }
}
```



整個程式最重點的部分!!!

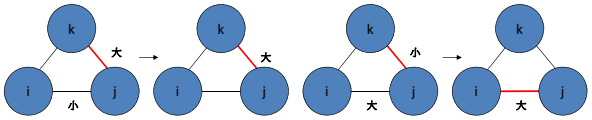
6.計算i j兩個程式之間的重量最大承受量

```
for ( int k = 1; k <= n; k++ ) {
    for ( int i = 1; i <= n; i++ )  {
        for ( int j = 1; j <= n; j++ ) {
            d [i] [j] = d [j] [i] = max (d [i] [j], min (d [i] [k], d [k] [j]));
        }
    }
}
```

整個程式最重點的部分!!!

6.計算i j兩個程式之間的重量最大承受量

```
for ( int k = 1; k <= n; k++ ) {
    for ( int i = 1; i <= n; i++ )  {
        for ( int j = 1; j <= n; j++ ) {
            d [i] [j] = d [j] [i] = max (d [i] [j], min (d [i] [k], d [k] [j]));
        }
    }
}
```



整個程式最重點的部分!!!

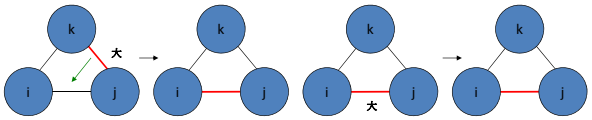
6.計算i j兩個程式之間的重量最大承受量

```
for ( int k = 1; k <= n; k++ ) {
    for ( int i = 1; i <= n; i++ )  {
        for ( int j = 1; j <= n; j++ ) {
            d [i] [j] = d [j] [i] = max (d [i] [j], min (d [i] [k], d [k] [j]));
        }
    }
}
```

整個程式最重點的部分!!!

6.計算i j兩個程式之間的重量最大承受量

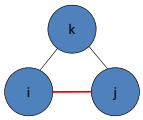
```
for ( int k = 1; k <= n; k++ ) {
    for ( int i = 1; i <= n; i++ )  {
        for ( int j = 1; j <= n; j++ ) {
            d [i] [j] = d [j] [i] = max (d [i] [j], min (d [i] [k], d [k] [j]));
        }
    }
}
```

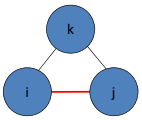


整個程式最重點的部分!!!

6.計算i j兩個程式之間的重量最大承受量

```
for ( int k = 1; k <= n; k++ ) {
    for ( int i = 1; i <= n; i++ )  {
        for ( int j = 1; j <= n; j++ ) {
            d [i] [j] = d [j] [i] = max (d [i] [j], min (d [i] [k], d [k] [j]));
        }
    }
}
```






```
string source, dest;
cin >> source >> dest;
printf ("Scenario #%d\n", ++cases);
printf ("%d tons\n\n", d [cityIndex[source]] [cityIndex [dest]]);
```

7. 宣告string source來存起點，string dest存終點
8. 輸出兩城市間重量最大承受量➔答案!

Solution 2

1.把每座城市編上編號
2.把輸入的城市用linked-list存好
3.並把linked-list指向struct，struct裡存通往的城市及最大負載重量

```
while(r--) {
    cin >> x >> y >> v;
    if(record[x] == 0) {
        encode ++;
        record[x] = encode;
    }
    if(record[y] == 0) {
        encode++;
        record[y] = encode;
    }
    nx = record[x], ny = record[y];
    Arc arc;
    arc.to = ny, arc.v = v;
    Link[nx].push_back(arc);
    arc.to = nx, arc.v = v;
    Link[ny].push_back(arc);
}
```

4.輸入起點與目的地
5.把起點與目的地所代表的編號傳到SPFA

```
cin >> x >> y;
printf("Scenario #%d\n",++Case);
nx = record[x], ny = record[y];
printf("%d tons\n", SPFA(nx, ny));
```

SPFA

```
queue<int> Q;
 int dis[201];
 int used[201];
 memset(dis, 0, sizeof(dis));
 memset(used, 0, sizeof(used));
 dis[st] = 0xffffffff;
  for(it i = Link[st].begin();
i !=Link[st].end(); i++) {
      if(dis[i->to] < Min(dis[st], i->v)) {
          dis[i->to] = Min(dis[st], i->v);
          if(used[i->to] == 0) {
              used[i->to] = 1;
              Q.push(i->to);
          }
      }
  }
```

```
int tv;
while(!Q.empty()) {
    tv = Q.front();
    Q.pop();
    used[tv] = 0;
    for(it i = Link[tv].begin(); i !=
Link[tv].end(); i++) {
        if(dis[i->to] < Min(dis[tv], i->v)) {
            dis[i->to] = Min(dis[tv], i->v);
            if(used[i->to] == 0) {
                used[i->to] = 1;
                Q.push(i->to);
            }
        }
    }
}
return dis[ed];
```

- 4 3
- Karlsruhe Stuttgart 100
- Stuttgart Ulm 80
- Ulm Muenchen 120
- Karlsruhe Muenchen

LINK[1]	2	100		
LINK[2]	1	100	3	80
LINK[3]	2	80	4	120
LINK[4]	3	120		

QUEUE | 2 | | | | | | |

```
for(it i = Link[st].begin();
i !=Link[st].end(); i++) {
      if(dis[i->to] < Min(dis[st], i->v)) {
          dis[i->to] = Min(dis[st], i->v);
          if(used[i->to] == 0) {
              used[i->to] = 1;
              Q.push(i->to);
           }
       }
   }
```

把起點通往的城市丟到queue
用dis[]紀錄最大負載重量
dis[2]<Min(dis[st],i->v)=100
dis[2]=100
用used[]判斷是否在queue裡

QUEUE | 4 | | | | | | |

```
while(!Q.empty()) {
      tv = Q.front();
      Q.pop();
      used[tv] = 0;
      for(it i = Link[tv].begin(); i != Link[tv].end();
i++) {
            if(dis[i->to] < Min(dis[tv], i->v)) {
                dis[i->to] = Min(dis[tv], i->v);
                if(used[i->to] == 0) {
                      used[i->to] = 1;
                      Q.push(i->to);
                }
            }
       }
}
```

把queue的2pop出來
Linked[2]連接到1號城市及3號城市

因為1號城市是起點，dis[1]=無限大
dis[1]>Min(dis[2],i->v)=100
不會進入queue

dis[3]=0<Min(dis[2],i->v)=80
dis[3]=80
把3號丟進queue裡

把queue的3pop出來
Linked[3]連接到2號城市及4號城市

dis[2]=100>Min(dis[3],i->v)=80
不會進入queue
->代表直接從1號到2號的最大負載重量會大於從1號經過2號和3號再回到2號

時間複雜度 Solution1 v.s. Solution2

Solution1	Solution2
$O(V^3)$	O(VE)

因為solution1是用array去存，浪費空間，浪費時間跑不存在的東西
Solution2是用linked-list去存，只會存需要用到的東西

Running Time Solution1 v.s. Solution2