**★★★★☆**

**題組：Problem Set Archive** **with Online Judge**

**題號：12821: Double Shortest Paths**

**解題者：盧彥凱**

**解題日期：2024年5月16日**

**題意：**

有兩人想從洞穴1走到洞穴n。 通道一旦通過會使再次通過變得更困難。 每個通道都有一個困難度，表示第一次通過的難度，和一個額外困難度，第二次通過難度會是困難度加上額外困難度。 每個測試案例提供了洞穴數量n，通道數量m，和每個通道的困難度。 此題要求找出兩條（可能相同的）路線，使得兩人通過的總困難度最小化。

**題意範例：**

輸入: 4 4

1 2 5 10

2 4 6 10

1 3 4 10

3 4 9 10

第一個人:

1->2->4=5+6=11

第二個人:

1->3->4=4+9=13

Total:11+13=23

輸出:

23

**解法：**

1. 用addEdge()為測試案例上的每一條邊創造一條正向邊(容量為1)、一條有加上額外cost的正向邊(容量為0)，和一條反向邊(其cost為-1\*該邊的cost(不包含額外)，且容量為0)。

2.利用minCost()(使用兩次Dijkstra’s algorithm)求解最小cost:

(1) 初始化距離數組dis[]為無限大，起點到自己的距離為0。

(2) 使用deque<int> Q來實作(這樣可以從front或back加入節點)，一開始會

 放入1節點。

(3) 當佇列不為空時，不斷從佇列中取出節點，接著查看從此節點開始的

 每一條邊，如果1節點到此節點的距離加正在查看的邊cost小於dis[]

 中紀錄的可到達節點的距離，則更新距離，並將其加入Q。

(4) 若佇列為空，則跳出循環

(5) 根據prev[]回溯路徑，更新邊的容量，並更新mincost(初始值是

 0)加上dis[n]

(6) 回到(1)在做一次

**解法範例：**

**例1:**



**例2:**

****

****

****

****

**討論：**

當解不是連兩次走當時的最短路徑時(上面的例2)，比較正確解跟程式解:

正確解

第一個人:1->3->6->7

第二個人:1->2->4->5->7

程式解:

第一個人:1->3->4->5->7 (從4後是第二條)

第二個人:1->2->4->3->6->7 (從3後是第一條)

可以從這個比較明白，設立反向邊的原因是因為要在走第二個路徑時讓第一個人可以反悔，因為正確答案不一定是最短路徑在加上之後的最短路徑。

**程式：**

#include<iostream>

#include<string>

#include<sstream>

#include<cstring>

#include<deque>

#include<fstream>

using namespace std;

#define INF 0x3f3f3f3f

class Edge

{

public:

    int x,y;

    int cap;

    int cost;

    int next;

    Edge(int a,int b,int capacity,int c,int n):x(a),y(b),cap(capacity),cost(c),next(n)

 {}

    Edge() {}

};

Edge edges[9000];

int head[505], prev\_node[505], inq[505],record[505],dis[505],e;

void addEdge(int x,int y,int cap,int cost)

{

    Edge e1(x,y,cap,cost,head[x]),e2(y,x,0,-cost,head[y]);

    head[x]=e;

    edges[e++]=e1;

    head[y]=e;

    edges[e++]=e2;

}

int minCost(int s,int t)

{

    int mncost=0,flow=0,totalflow=0;

    int x,y;

    while(1)

    {

        memset(dis,INF,sizeof(dis));

        int oo=dis[0];

        dis[s]=0;

        deque<int> Q;

        Q.push\_front(s);

        while(!Q.empty())

        {

            x=Q.front(); Q.pop\_front();

            inq[x]=0;

            for(int i=head[x];i!=-1;i=edges[i].next)

            {

                y=edges[i].y;

                if(edges[i].cap>0 && dis[y]>dis[x]+edges[i].cost)

                {

                    dis[y]=dis[x]+edges[i].cost;

                    prev\_node[y]=x;

                    record[y]=i;

                    if(inq[y]==0)

                    {

                        inq[y]=1;

                        if(Q.size() && dis[Q.front()]>dis[y])

                            Q.push\_front(y);

                        else

                            Q.push\_back(y);

                    }

                }

            }

        }

        if(dis[t]==oo)

            break;

        for(int x=t;x!=s;x=prev\_node[x])

        {

            int ri=record[x];

            edges[ri].cap-=1;

            edges[ri^1].cap+=1;

            edges[ri+2].cap+=1;

            edges[ri^1].cost=edges[ri].cost\*-1;

        }

        mncost+=dis[t];

    }

    return mncost;

}

int main()

{

    int n,m,count=1;

    string info;

    int x,y,a,b;

    while(getline(cin,info) && info.size()>0)

    {

        stringstream ss(info);

        ss >> n >> m ;

        e=0;

        memset(head,-1,sizeof(head));

        int source=n+1,sink=n+2;

        for(int i=0;i<m;i++)

        {

            cin >> x >> y >> a >> b ;

            addEdge(x,y,1,a);

            addEdge(x,y,0,a+b);

        }

        cin.ignore();

        addEdge(source,1,2,0);

        addEdge(n,sink,2,0);

        cout << "Case " << count++ << ": " << minCost(source,sink) << endl ;

    }

    return 0;

}