**★★★☆☆**

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

**題號：10246: Asterix and Obelix**

**解題者：陳宥景**

**解題日期：2021年4月1日**

**題意：**

給定C個點，各點都有各自的權重，給定R條路，各條路都有各自的距離長度，給定Q個問題(起點和終點)，最後要求起點和終點間的距離加上路徑中的最大權重是最少的數字。

**題意範例：**

7 8 2 4 4 2

2 3 5 15 4 4 6 2 1 8 3

1 2 20 1 2 7

1 4 20 1 3 5

1 5 50 2 4 8

2 3 10 3 4 6

3 4 10 1 4 -> 18

3 5 10 2 3 -> 20

4 5 15 0 0 0

6 7 10

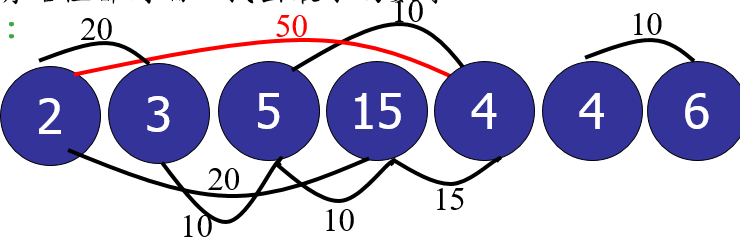
5 1 -> 45

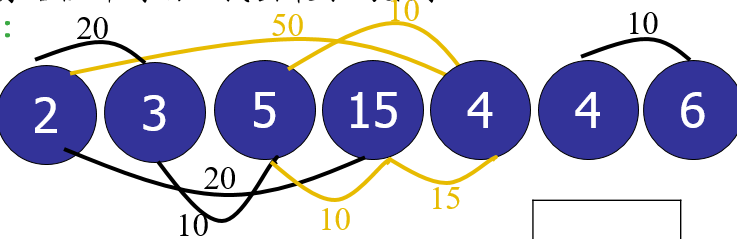
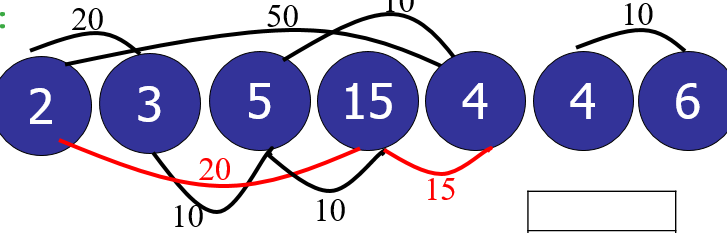
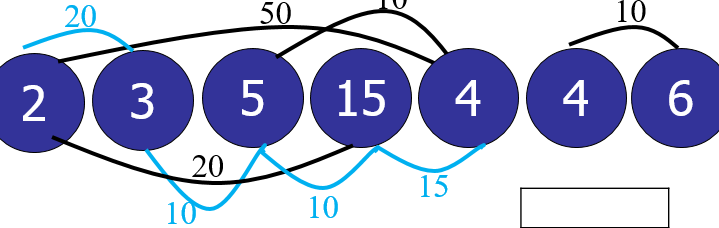
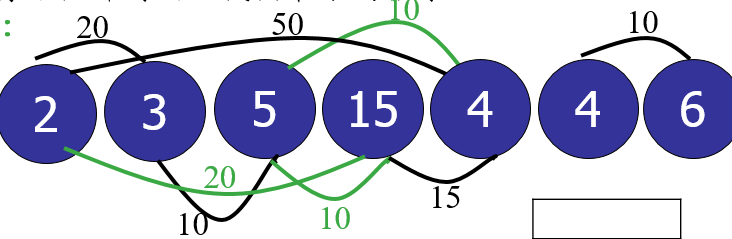
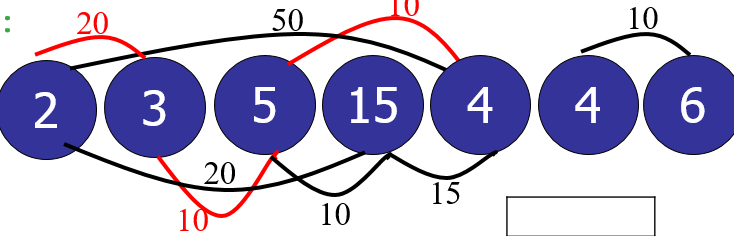
1 6 -> -1

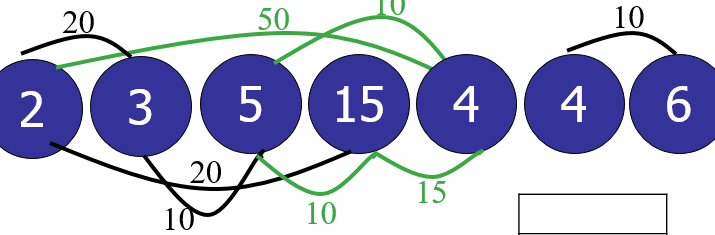
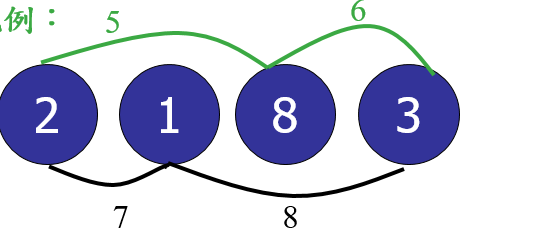
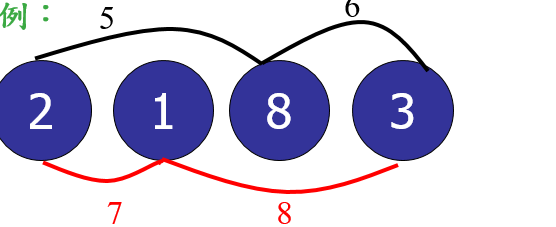
**解法：**

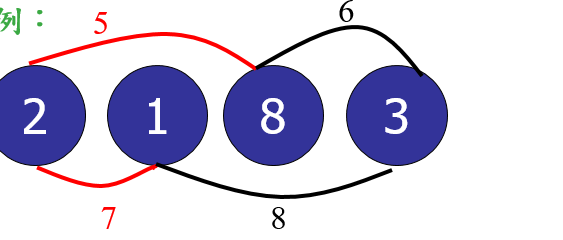
用二維陣列來儲存兩點之間的花費。每一個點都用一維陣列儲存最一開始題目給此點到其他點的距離先當作估計值，如果是自己到自己，就設為距離0，如果不通，就設為999999。開始判斷此點最近點的距離(也就是一維陣列裡的估計值)是不是比繞路還來的遠，如果距離比繞路短，估計值就保持不變，如果距離比繞路長，就更新一維陣列裡面的估計值成繞路的距離，保持此點到其他點的距離是最短的。最後一個一個對點假設此點的權重是此條路徑最大的權重，並找出起點到終點的距離加上假設點的權重是最小值的路徑，就是答案了。

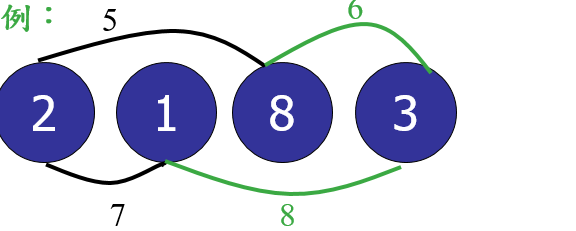
**解法範例：**

範例一 5到1



範例二 1到4

範例二 2到3



**討論：**

不能直接用求最短的路徑的演算法，因為點的權重也會影響答案

**程式：**

#include <iostream>

#include <algorithm>

#include <queue>

#include <vector>

#include <cstring>

using namespace std;

vector<int> weight;//點的權重

int cityNumber;

int roadNumber;

int queryNumber;

vector<int> estimateValue;

vector<int> leastCost[100];

vector<bool> test;

vector<pair<int,int> > dis[100];

class Connect

{

public :

int class\_c;

int class\_w;

bool operator < (const Connect& a) const

{

return class\_w > a.class\_w;

}

Connect(int c, int w);

};

Connect::Connect(int c, int w)

{

class\_c=c;

class\_w=w;

}

void DijkstraAlgorithm(int city);

int main()

{

int caseNumber = 0;

int temp;

int c1;

int c2;

int w;

int queryCity1;

int queryCity2;

while(1)//cityNumber 城市數 roadNumber 路數 queryNumber 詢問數

{

cin >> cityNumber >> roadNumber >> queryNumber;

if(cityNumber == 0 & roadNumber == 0 & queryNumber == 0)

{

break;

}

caseNumber++;

weight.clear();

weight.push\_back(0);

for(int i = 1; i <= cityNumber; i++)

{

cin >> temp;

weight.push\_back(temp);//輸入權重

dis[i].clear();//清空兩點之間距離的矩陣

}

for(int i = 0; i < roadNumber; i++)

{

cin >> c1 >> c2 >> w;//輸入c1點到c2點的w cost

dis[c1].push\_back(make\_pair(c2, w));//將兩點的距離 存到dis陣列裡

dis[c2].push\_back(make\_pair(c1, w));//因為雙向 所以兩邊都要存

}

for(int i = 0; i < 100; i++)//將點到點之間的成本 初始化設為無限大

{

leastCost[i].clear();

}

for(int i = 0; i < 100; i++)//將點到點之間的成本 初始化設為無限大

{

for(int j = 0; j < 100; j++)

{

leastCost[i].push\_back(999999);

}

}

for(int i = 1; i <= cityNumber; i++)

{

DijkstraAlgorithm(i);

}

if(caseNumber >= 2)//case結束 換行

{

printf("\n");

}

cout << "Case #" << caseNumber << endl;

for(int i = 0; i < queryNumber; i++)

{

cin >> queryCity1 >> queryCity2;

if (leastCost[queryCity1][queryCity2] != 999999)

{

cout << leastCost[queryCity1][queryCity2] << endl;//詢問的路最小成本路徑

}

else

{

cout << "-1" << endl;//詢問的路不通

}

}

}

return 0;

}

void DijkstraAlgorithm(int city)

{

priority\_queue<Connect> queue;

int c1;//起點

int c2;//終點

int w;//c1到c2的距離

test.clear();

for(int i = 0 ; i < 100; i++)//初始化所有估計值都沒試過

{

test.push\_back(false);

}

estimateValue.clear();

for(int i = 0; i < 100; i++)//點city到其他點的距離 初始化為無限大

{

estimateValue.push\_back(999999);

}

estimateValue[city] = 0;//自己的距離設為零

queue.push(Connect(city, estimateValue[city]));

while(!queue.empty())//直到再也沒有估計值為止

{

c1 = queue.top().class\_c;//能連到的點開始測試

queue.pop();//刪除正在測試的點

if (test[c1] == true)//試過 就下一個

{

continue;

}

test[c1] = true;//設為已試過

for(int i = 0; i < dis[c1].size(); i++)

{

c2 = dis[c1][i].first;

w = dis[c1][i].second;

if (weight[c2] > weight[city])//假設city的權重最大 所以點c2的權重較大 就不用比較了

{

continue;

}

if (estimateValue[c2] > estimateValue[c1] + w)//點city到點c2的距離 和 點city到點c1的距離再加上點c1到點c2的距離 的比較

{

estimateValue[c2] = estimateValue[c1] + w;//鬆弛

queue.push(Connect(c2, estimateValue[c2]));

}

}

}

for(int i = 1; i <= cityNumber; i++)

{

for(int j = 1; j <= cityNumber; j++)

{

leastCost[i][j] = min(leastCost[i][j], estimateValue[i] + estimateValue[j] + weight[city]);//選出最小成本路徑

}

}

}