**Programming Assignment 4**

Implement the Bellman-Ford algorithm

104303206 黃筱晴

一、加速版Bellman–Ford algorithm

原本Bellman–Ford演算法直接對所有edge進行Relaxation，總共做V-1次。加速版用Breadth-First-Search的方式，從起點開始，對鄰邊進行Relaxation，一層一層向外擴展，直到修正所有鄰點的最短路徑長度。

### 參考維基百科Bellman–Ford algorithm條目中提到的佇列最佳化來實作加速版。

### modified\_BellmanFord(G,s,d)

### Initialize();

queue Q;// 準備一個佇列來操作Relaxation向外擴展的順序

bool color[V]// 準備一個陣列紀錄哪些vertex已經被放進去過了

Q.push(s);//把起點放進Q

color[s]=1//紀錄起點已經放進去過了

while( Q!=⏀)

u=Q.front();//從Q中拿出一點

for (所有u的鄰邊uv)

Relaxation();

if( color[v]!=1 )

Q.push(v);//把v放進Q

color[v]=1//紀錄v已經放進去過了

再做一次以上步驟，若又找到可修正的邊，則return False//存在負環

//確定無負環後，利用π回推路徑，並使用一個stack來裝

stack AnsPath; int AnsWeight=0;

now=d;//從終點開始往回找

last=π[now];

while(now!=s)

AnsWeight += w[last][now];

AnsPath.push(now);

now=last;

last=π[now];

AnsPath.push(s);//此時stack中就是從s走到d的完整最短路徑

return True;

二、加速版時間複雜度分析

從Pseudo code中觀察，第七行的while( Q!=⏀)總共會拿出V個頂點；每拿出一個頂點，第九行的for迴圈最多有E個鄰邊可以Relaxation()。最差情形的時間複雜度為O(VE)和原版相同。但是通常每拿出一個頂點，它應該不會有那麼多鄰邊。最差情形發生在超密集的圖上(每個頂點都相鄰)。如果是稀疏一點的圖，加速版可以快許多。

三、原版與加速版的時間比較

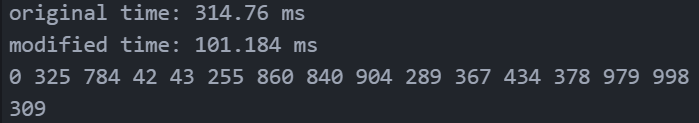
測試環境：ASUS UX303L筆電

處理器：Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz 2.19GHz

RAM：8.00GB

系統類型：64位元作業系統 x64型處理器

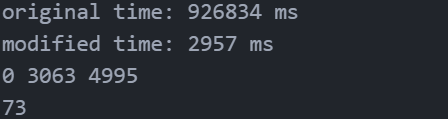
(1) 1000點 2000條邊。



original time:314.76 ms

modified time:101.184 ms

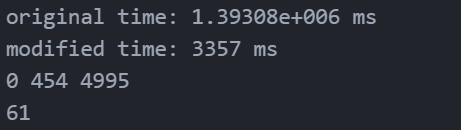
(2) 5000點 100萬條邊



original time:926834 ms

modified time:2957 ms

(3) 5000點 150萬條邊



original time:1.39308\*106  ms

modified time:3357 ms

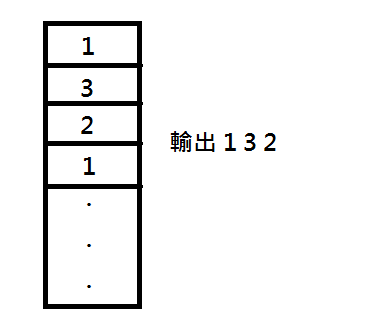
四、輸出Negative weight cycle的方式

再度發現可以修正的邊uv時，從u開始，利用π往回推，直到遇到v即是一個負環。

if d[v]>d[u]+w(u,v)

stack Ans;

int HaveFinded[n];//紀錄1:已經被放進stack 0:未放進stack

 Ans.push(v);

HaveFinded[v]=1;

while(HaveFinded[u]!=1)

v=u; u=π[v];

Ans.push(v); HaveFinded[v]=1;

AnsPath.push(u);//此時負環已經在stack的頂端