

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!= $\Phi$  )
```

```
    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 $\pi$  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;

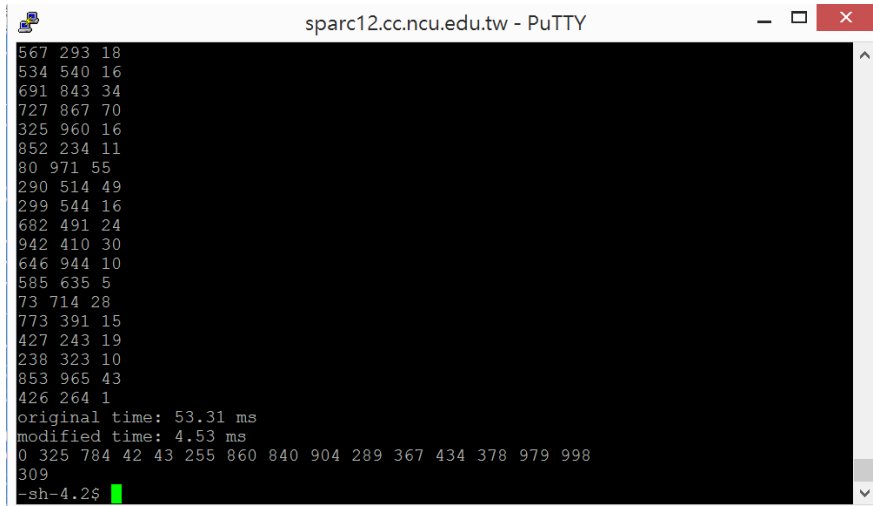
```

二、加速版時間複雜度分析

從 **Pseudo code** 中觀察，第七行的 `while(Q!= Φ)` 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

使用助教提供的1000個頂點與2000條邊的測資。

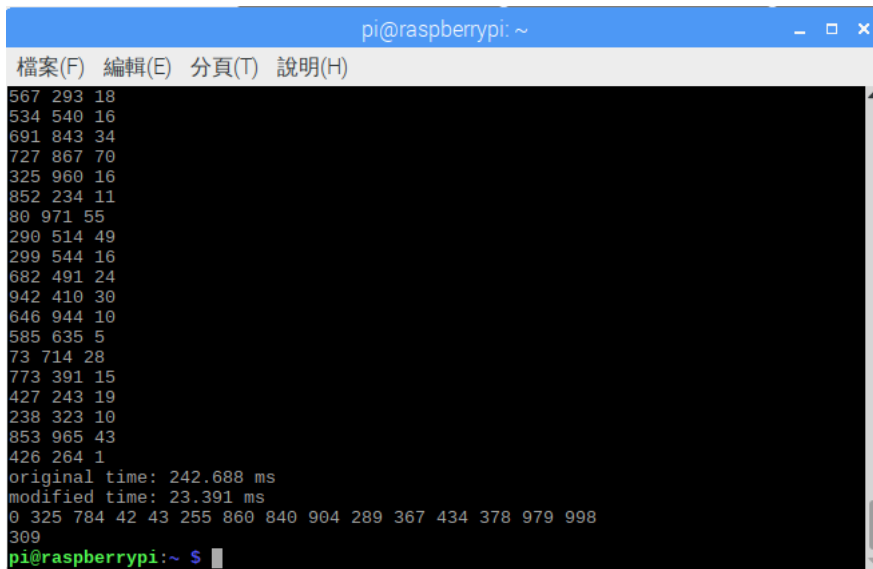


```
567 293 18
534 540 16
691 843 34
727 867 70
325 960 16
852 234 11
80 971 55
290 514 49
299 544 16
682 491 24
942 410 30
646 944 10
585 635 5
73 714 28
773 391 15
427 243 19
238 323 10
853 965 43
426 264 1
original time: 53.31 ms
modified time: 4.53 ms
0 325 784 42 43 255 860 840 904 289 367 434 378 979 998
309
-sh-4.2$
```

測試環境：中央大學電算中心SPARC工作站

original time:53.31 ms

modified time:4.53 ms



```
567 293 18
534 540 16
691 843 34
727 867 70
325 960 16
852 234 11
80 971 55
290 514 49
299 544 16
682 491 24
942 410 30
646 944 10
585 635 5
73 714 28
773 391 15
427 243 19
238 323 10
853 965 43
426 264 1
original time: 242.688 ms
modified time: 23.391 ms
0 325 784 42 43 255 860 840 904 289 367 434 378 979 998
309
pi@raspberrypi:~ $
```

測試環境：樹莓派3B+

original time:242.688 ms

modified time:23.391 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)
```

```
         $v = u; u = \pi[v];$ 
```

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

```
    AnsPath.push(u); //此時負環已經在 stack 的頂端
```

1
3
2
1
.
.
.

輸出 1 3 2