LAB 10

有向無環圖上的路徑長度

長榮大學

資訊工程學系

班級：資工 2B

姓名：郭智榮

學號：109B30612

日期：2022/05/23

版本校定紀錄：

版本	更新紀錄	發佈日期
0.0.0.0	初版完成	2022/05/23

一. 實驗需求：

(一). 題目說明

描述：

輸入一個 DAG 及節點 s 與 t 兩點，計算 s 到 t 的最短與最長路徑的長度。

輸入說明：

第一行是兩個正整數 n 與 m，代表節點數與有向邊數，節點是以 0~n-1 編號。

第二行兩個整數 s 與 t。

接著有 m 行，每一行三個整數 u, v, w 代表一條有向邊(u,v)的長度是 w。

輸出說明：

第一行輸出最短路徑長度，第二行輸出最長路徑長度，如果不存在，兩者皆輸出「No path」。

(二). 演算法

i. 虛擬碼

定義 using namespace 型態的 std；//無此定義則 vector 宣告將報錯

定義 max 為 1000000000；
定義 min 為-1000000000；

int main(){
宣告兩個整數(int)型態變數 n, m，並用於儲存「測資節點數量、有向邊數量」；

輸出提示訊息「請輸入節點數及有向邊數量」；
輸入兩整數代表該筆測資節點數及有向邊數，並存入 n 和 m 中；

宣告 vector 型態變數 vertex 並存放兩個整數(int)型態資料，且大小為 n，用於存放節點可前往的節點及該邊長(權重)；

宣告 queue 型態變數 visit 並存放一個整數(int)型態資料，用於存放需要造訪的節點；

建立一個整數陣列 haveEntrance 用於記錄可通往該節點的有向邊數量，預設全為 0，陣列大小為 n；

建立兩個整數陣列 Long 及 Short 用於記錄起點至該點的最長及最短路徑，Long 預設全為 min，Short 預設全為 max，陣列大小皆為 n；

宣告兩個整數(int)型態變數 s, t，並用於儲存「題目要求的起點、終點」；

輸出提示訊息「請輸入起點與終點」；
輸入兩整數代表該筆測資起點與終點，並存入 s 和 t 中；

將 Long[s]及 Short[s]設為 0，使 s 被定為起點；

```
for i = 0 to i < m (for i++){
    宣告三個整數(int)型態變數 u, v, w，分別用於儲存「有向邊的起點、目的地及邊長(權重)」;

    輸出提示訊息「請輸入有向邊起點、終點及有向邊邊長」;
    輸入有向邊起點存入 s、終點存入 t 及有向邊邊長存入 w;

    將{v, w}存入 vertex[u]，代表可由 u 前往 v 且邊長為 w;
    將 haveEntrance[v]加 1;
}

for i = 0 to i < n (for i++){
    如果 haveEntrance[i]為 0，代表該節點只能做為起點{
        將 i 放入 visit 中，以利後續造訪 i;
    }
}
```

```
迴圈 (visit 不為空){
    宣告一個整數(int)型態變數 now，存入 visit 中最早放的值；
    將 visit 中最早放入的值取出(pop)；

    如果(Long[now]等於 min)或 now 等於 t{
        執行下一次迴圈(continue)；
    }

    for next : vertex[now]{
        // next.first 為 next 節點；next.second 為有向邊邊長
        //因句子過長會使 word 自動換行，故以下將省去 next.
        //原 next.first 寫為 first;原 next.second 寫為 second

        如果 Long[first]小於(Long[now] + second){
            Long[first]為(Long[now] + second)；
        }

        如果 Short[first]大於(Short[now] + second){
            Short[first]為(Short[now] + second)；
        }

        將 haveEntrance[first]減 1；
        如果 haveEntrance[first]為 0，代表需要造訪{
            將 first 放入 visit 中，以利後續造訪；
        }
    }
}
如果 Long[t]等於 min，代表 t 未被造訪到{
    因為終點 t 未被造訪，故輸出兩次「No path」;
}
否則{
    輸出 Long[t]；
    輸出 Short[t]；
}
return 0；
}
```

二. 完整程式碼：

```
#include <bits/stdc++.h>
using namespace std;
#define max 1000000000
#define min -1000000000

int main(){
    int n, m;
    // 宣告整數變數，分別用於儲存「測資節點數量、有向邊數量」
    printf("\n Input the vertex and edges : ");
    // 輸出提示訊息
    scanf("%d%d", &n, &m);
    // 輸入節點數量並存入 n ; 輸入有向邊數量並存入 m

    vector< pair<int,int> > vertex[n];
    // 建立一個 vector 型態陣列，內部儲存值為兩個整數值
    queue<int> visit;
    // 建立一個 queue 型態陣列，內部儲存值為整數型態，記錄要造訪的節點
    int haveEntrance[n] = {0}, Long[n], Short[n];
    // 建立一個整數陣列，紀錄其他點可以進入該點的數量，預設為 0
    // 建立兩個整數陣列，紀錄起點至該點時的最長(Long)及最短(Short)路徑

    for(int i = 0 ; i < n ; i++){
        Long[i] = min;
        // 將 Long 內所有節點的值設為最小值
        Short[i] = max;
        // 將 Short 內所有節點的值設為最大值
    }

    int s, t;
    // 宣告整數變數，分別用於儲存「題目要求的起點、終點」
    printf("\n Input the start and end : ");
    // 輸出提示訊息
    scanf("%d%d", &s, &t);
    // 輸入起點並存入 s ; 輸入終點並存入 t
```

```
Long[s] = Short[s] = 0;
// 將題目要求起點的 Long 及 Short 設為 0

for(int i  = 0 ; i < m ; i++){
    int u, v, w;
    // 宣告整數變數，分別儲存「該有向邊的起點、目的地及邊長(權重)」
    printf("\n Input the edge-%d and weight : ", i);
    // 輸出提示訊息
    scanf("%d%d%d", &u, &v, &w);
    // 輸入有向邊起點存入 s 、有向邊終點並存入 t 及有向邊長存入 w
    vertex[u].push_back({v, w});
    // 紀錄 u 節點可以通往 v 節點且邊長為 w
    haveEntrance[v] ++;
    // 將可進入 v 節點的有向邊數量 + 1
}
for(int i = 0 ; i < n ; i++){
    if((haveEntrance[i] == 0)){
    // 若節點 i 的入口為 0 代表無法在路徑中造訪，因此需先放入佇列中
        visit.push(i);
        // 將 i 放入需要造訪的節點佇列中
    }
}

while(!visit.empty()){
// 若佇列不為空則繼續執行 while 內的程式
    int now = visit.front();
    // 宣告整數變數 now，存入佇列中最先放入的值
    visit.pop();
    // 將佇列中最先放入的值取出

    if((Long[now] == min) || (now == t)){
    // 如果 now 的最長路徑為最小值，代表尚未到可進入該節點的位置
    // 如果目前執行的節點為終點，則終點後續不必執行
        continue;
        // 執行下一個 while
    }
```

```
            for(auto next : vertex[now]){
            // 將節點的下一個節點及邊長放入 next
                if(Long[next.first] < (Long[now] + next.second)){
                // 如果 next 的最長路徑小於 now 節點加上邊長，則進入 if 內
                    Long[next.first] = (Long[now] + next.second);
                    // 將 next 節點的最長路徑存入 now 節點加上邊長
                }
                if(Short[next.first] > (Short[now] + next.second)){
                // 如果 next 的最短路徑大於 now 節點加上邊長，則進入 if 內
                    Short[next.first] = (Short[now] + next.second);
                    // 將 next 節點的最短路徑存入 now 節點加上邊長
                }
                haveEntrance[next.first] --;
                // 將可進入 next 節點的有向邊數量 - 1
                if(haveEntrance[next.first] == 0){
                // 如果後續沒有可進入 next 節點的有向邊，則進入 if 內
                    visit.push(next.first);
                    // 將 next 節點放入待造訪的節點佇列中
                }
            }
        }

        if(Long[t] == min){
        // 若終點的最大長度為最小值，進入 if 內
            printf("\n No path.");
            printf("\n No path.");
            // 因沒有路徑達終點，故輸出「No path」的訊息
        }
        else{
            printf("\n The path length min : %d.", Short[t]);
            // 輸出最短路徑
            printf("\n The path length max : %d.", Long[t]);
            // 輸出最長路徑
        }
        return 0;
    }
```

三. 輸入及輸出結果：

```
Input the vertex and edges : 5 6

Input the start and end : 0 4

Input the edge-0 and weight : 0 2 3

Input the edge-1 and weight : 0 3 1

Input the edge-2 and weight : 2 1 -2

Input the edge-3 and weight : 3 4 0

Input the edge-4 and weight : 1 4 2

Input the edge-5 and weight : 2 4 3

The path length min : 1.
The path length max : 6.
```

```
Input the vertex and edges : 4 3

Input the start and end : 2 0

Input the edge-0 and weight : 2 1 5

Input the edge-1 and weight : 1 3 0

Input the edge-2 and weight : 0 1 1

No path.
No path.
```

四. 心得與討論：

此次實作計算 DAG 路徑長度，是延續 LAB9 的概念往下延伸，因此在實作前閱讀題目了解的速度快非常多，後續在程式碼的撰寫部分邏輯也相較上次更清晰、明確。

因為 LAB9 是第一次對 DAG 進行實作，藉由 LAB9 的實作經驗延伸出 LAB10，所以在整體難度上 LAB10 並沒有 LAB9 那麼困難。

也在網路上稍微多找了一些別人撰寫 DAG 的程式碼、虛擬碼，也能發現一些自己沒有想到寫法，因此整體收獲也是非常大。