LAB 11

運用優先權佇列之 Dijkstra 最短路徑演算法

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：2022/05/30

版本校定紀錄：

版本	更新紀錄	發佈日期
0.0.0.0	初版完成	2022/05/30

一. 實驗需求：

(一). 題目說明

描述：

輸入一個無向圖 G=(V, E, w)，其中點以 0 ~ n-1 編號，而邊長是非負整數。運用 Dijkstra 最短路徑演算法計算頂點 0 到其他點的最短路徑長度。兩點之間可能有多個邊。

輸入說明：

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

接著有 m 行，每一行三個整數 u, v, w 代表一條無向邊(u,v)的長度是 w。其中 n 不超過 10 的 4 次方，m 不超過 10 的 5 次方，w 的絕對值不超過 10 的 4 次方。

輸出說明：

第一行輸出 0 到各點之最短路徑長度中的最大值,也就是在可以到達的點中，最短距離最大的是多少。

第二行輸出有多少點無法抵達。

(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 10000000；

int main(){

宣告兩個整數(int)型態變數 n, m，並用於儲存「頂點數量及無向邊數量」;

輸出提示訊息「請輸入頂點數量及無向邊數量」;

輸入兩整數代表測資頂點數量及無向邊數量，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)；

建立一個整數陣列 distance 用於紀錄由 0 到某頂點的距離，預設全為 max，陣列大小為 n；
將 distance[0]設為 0，因 0 到自己的距離為 0；

建立一個布林陣列 final 用於紀錄某頂點是否已完成，預設全為 false，陣列大小為 n；

```
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「該條無向邊的兩端及邊長(權重)」;

    輸出提示訊息「請輸入該條無向邊的兩端及邊長」;
    輸入無向邊的一端存入 u,另一端存入 v 及無向邊邊長存入 w;

    宣告布林變數 check 並設為 false,用於紀錄所輸入的兩端是否已有連結的邊;

    for j = 0 to j < vertex[u].size() (for j++){
        如果 vertex[u][j].first 等於 v，則 u 已與 v 有無向邊{
            將 check 改為 true;
            如果(vertext[u][j].second 大於 w{
            // 因原本的邊長大於目前輸入的 w，故改為 w
                vertext[u][j].second 設為 w;
                break;
            }
        }
    }
    for j = 0 to j < vertex[v].size() (for j++){
        如果 vertex[v][j].first 等於 u，則 v 已與 u 有無向邊{
            如果(vertext[v][j].second 大於 w{
            // 因原本的邊長大於目前輸入的 w，故改為 w
                vertext[v][j].second 設為 w;
                break;
```

```
                }
            }
        }

        如果 check 不為 true，代表 u 與 v 間目前沒有無向邊連結{
            將{v,w}放到 vertex[u]內的最後；
            將{u,w}放到 vertex[v]內的最後；
        }

    }


宣告優先佇列變數 queue 儲存兩個整數，用於紀錄要造訪的頂點；
將 distance[0] 及 0 放入 queue 中；

迴圈 (queue 不為空){
    宣告整數變數 now_vertex 並放入 queue 中最先的頂點；
    因已取出最先的資料，故將 queue 用 pop 丟出最先資料；

    如果 final[now_vertex]為 true，代表已被訪問完成{
        continue；
    }
    將 final[now_vertex]設為 true；

    for auto next 依序存入 vertex[now_vertex] {
        宣告整數 a 存入(distance[now_vertex] + next.second)
        如果 distance[next.first]大於 a{
            distance[next.first]設為 a；
            queue 存入{ distance[next.first], next.first };
        }
    }
}

宣告整數變數 max_path 及 no_visit 儲存「由 0 到某點的最長距
離及未造訪到的頂點數」;
```

```
    for i = 0 to i < n (for i ++){
        如果 max_path 小於 distance[i]{
            如果 distance[i]等於 max{
                將 no_visit 加 1；
            }
            否則{
                將 max_depth 設為 distance[i]；
            }
        }
    }

    輸出「最長距離為 max_depth」;
    輸出「未被造訪的頂點數量為 no_visit」;

    return 0；
}
```

二. 完整程式碼：

```
#include<bits/stdc++.h>
using namespace std;
#define max 10000000
// 定義通用的最大距離 max 為 10000000
int main(){
    int n, m;
    // 宣告整數 n, m，分別用於儲存「頂點數量及無向邊數量」
    printf("\n Input the vertex and edges : ");
    // 輸出提示訊息
    scanf("%d%d", &n, &m);
    // 將輸入的頂點數量存入 n, 無向邊數量存入 m
    vector< pair<int, int> > vertex[n];
    // 宣告 vector 型態變數 vertex 儲存資料為兩個整數

    int distance[n];
    // 宣告一個大小為 n 的整數陣列，用於儲存各點與頂點 0 的距離
    for(int i = 0 ; i < n ; i++){
        distance[i] = max;
        // 將 distance 內所有節點的值設為 max
    }
    distance[0] = 0;
    // 將頂點 0 與自己的距離設為 0

    bool final[n];
    // 建立大小為 n 的布林矩陣，紀錄該頂點是否已完成
    for(int i = 0 ; i < n ; i++){
        final[i] = false;
        // 將 final 內所有節點的值設為 false
    }

    for(int i  = 0 ; i < m ; i++){
        int u, v, w;
        // 宣告整數變數，分別用於儲存「該條無向邊的兩端及邊長(權重)」
        printf("\n Input the edge-%d and weight : ", i);
```

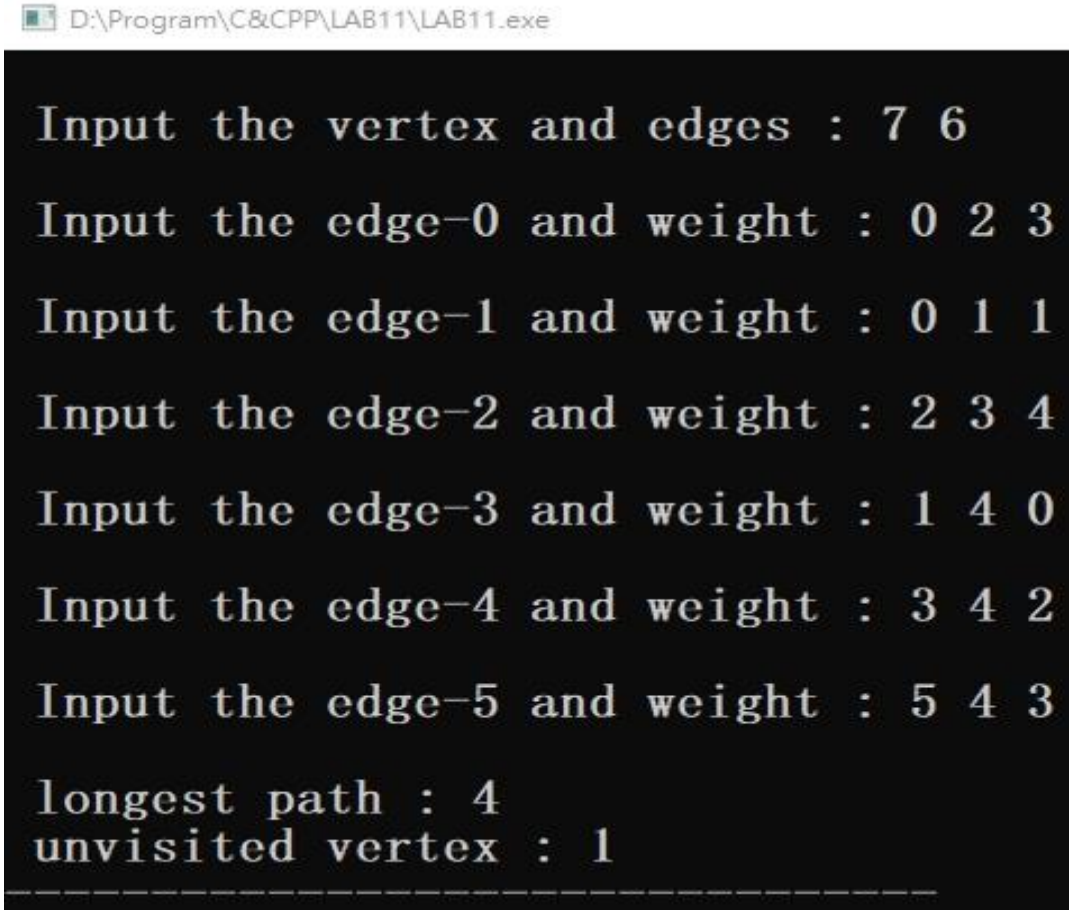
```
// 輸出提示訊息
scanf("%d%d%d", &u, &v, &w);
// 輸入有向邊起點存入 s 、有向邊終點並存入 t 及有向邊長存入 w
bool check = false;
// 宣告布林變數 check 紀錄所輸入的兩端是否已有連結的邊

for(int j = 0 ; j < vertex[u].size() ; j++){
// 搜尋 vertex[u] 內的所有資料
    if (vertex[u][j].first == v){
    // 若 vertex[u][j] 為 v，代表該條邊連結 u 與 v
        check = true;
        // 將兩端已有連結線變更為 true
        if(vertex[u][j].second > w){
        // 若 u 與 v 的無向邊邊長大於當前輸入的邊長則進入 if 內
            vertex[u][j].second = w;
            // 將 u 與 v 的邊長改為 w
            break;
            // 離開迴圈
        }
    }
}

for(int j = 0 ; j < vertex[v].size() ; j++){
// 搜尋 vertex[v] 內的所有資料
    if (vertex[v][j].first == u){
    // 若 vertex[v][j] 為 u，代表該條邊連結 v 與 u
        if(vertex[v][j].second > w){
        // 若 u 與 v 的無向邊邊長大於當前輸入的邊長則進入 if 內
            vertex[v][j].second = w;
            // 將 u 與 v 的邊長改為 w
            break;
            // 離開迴圈
        }
    }
}
```

```
    if(!check){
    // 若 u 與 v 並不存在無向邊則進入 if 內
        vertex[u].push_back({v, w});
        // 紀錄 u 節點可以通往 v 節點且邊長為 w
        vertex[v].push_back({u, w});
        // 紀錄 v 節點可以通往 u 節點且邊長為 w
    }
}

priority_queue< pair<int, int> > queue;
// 宣告優先佇列變數 queue 儲存兩個整數，用於紀錄要造訪的頂點
queue.push({distance[0], 0});
// 將 distance[0] 及 0 放入 queue 中

while(!queue.empty()){
// 若 queue 不為空則進入 while 內
    int now_vertex = queue.top().second;
    // 宣告整數變數 now_vertex 並放入 queue 中儲存最前的頂點
    queue.pop();
    // 因取出最前的資料，故將其用 pop 丟出
    if(final[now_vertex]){
    // 如果 now_vertex 已被訪問完成，則進入 if 內
        continue;
    }
    final[now_vertex] = true;
    // 將目前造訪的頂點設為 true 以記錄該頂點已完成

    for(auto next : vertex[now_vertex]){
    // 將當前造訪頂點所連結的資料依序存入 next
        int a = (distance[now_vertex] + next.second);
        // 因 word 會跑版，故宣告整數變數 a
        // 將 a 存入由 0 到目前頂點並延伸至下個頂點的距離
        if(distance[next.first] > a){
        // 如果原先由 0 到下個頂點的距離大於 a
            distance[next.first] = a;
            // 將由 0 到下個頂點的距離改為 a
```

```
                queue.push({(distance[next.first] * -1), next.first});
                // 將由 0 到下個頂點的距離及下個頂點傳入 queue 中
                // 在 queue 中同頂點的資料，若距離較大會排在前，故需*-1
讓較短距離排在前
            }
        }
    }

    int max_path = 0, no_visit = 0;
    // 宣告整數變數 max_path 及 no_visit 儲存「由 0 到某點的最長距離及
未造訪到的頂點數」

    for(int i = 0 ; i < n ; i++){
        if(max_path < distance[i]){
        // 如果 max_path 比由 0 到頂點 i 的距離短，進入 if 內
            if(distance[i] == max){
            //  由 0 到頂點 i 的距離為 max 則進入 if 內
                no_visit ++;
                // 由 0 到頂點 i 的距離與 max 相同代表沒有被造訪到
            }
            else{
                max_path = distance[i];
                // 若頂點 i 有被造訪，則 max_path 為 0 到頂點 i 的距離
            }
        }
    }

    printf("\n longest path : %d", max_path);
    // 輸出由 0 到某頂點的最長距離
    printf("\n unvisited vertex : %d", no_visit);
    // 輸出未被造訪的頂點數量

    return 0;
}
```

三. 輸入及輸出結果：


D:\Program\C&CPP\LAB11\LAB11.exe
Input the vertex and edges : 3 3
Input the edge-0 and weight : 2 1 5
Input the edge-1 and weight : 1 0 0
Input the edge-2 and weight : 0 1 1
longest path : 5
unvisited vertex : 0


D:\Program\C&CPP\LAB11\LAB11.exe
Input the vertex and edges : 7 6
Input the edge-0 and weight : 0 2 3
Input the edge-1 and weight : 0 1 1
Input the edge-2 and weight : 2 3 4
Input the edge-3 and weight : 1 4 0
Input the edge-4 and weight : 3 4 2
Input the edge-5 and weight : 5 4 3
longest path : 4
unvisited vertex : 1

四. 心得與討論：

本次實作的 Dijkstra 最短路徑演算法，在這學期素玲老師授課的「演算法」中正好有學到，而且只在幾周前，因此對 Dijkstra 演算法的步驟還蠻有印象的；因此實作過程中看了題目便將程式成功撰寫出來，加上前兩周的實作也提升對 STL 的熟練度，因此整體難度算適中偏易，不會太難卻也能學到在演算法課程中無法學到的演算法實作。

不過在實作過程也發生了一些小烏龍，因為一開始瞄了一眼標題後，便直接看題目描述開始實作，導致沒有發現題目要使用優先佇列，也因此做了一個沒有優先佇列的版本，就順便在心得的後方補上該程式的演算法及程式碼，並將兩者不同的地方用紅標標出(上方優先佇列的程式碼，在下方補充程式中有部分被刪除)。

五. 補充程式(無使用優先佇列)：

(一). 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 10000000；

int main(){

宣告兩個整數(int)型態變數 n, m，並用於儲存「頂點數量及無向邊數量」;

輸出提示訊息「請輸入頂點數量及無向邊數量」;
輸入兩整數代表測資頂點數量及無向邊數量，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)；

建立一個整數陣列 distance 用於紀錄由 0 到某頂點的距離，預設全為 max，陣列大小為 n；
將 distance[0]設為 0，因 0 到自己的距離為 0；

建立一個布林陣列 final 用於紀錄某頂點是否已完成，預設全為 false，陣列大小為 n；
將 final[0]設為 true，因 0 不會被其他點造訪故設為已完成；

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「該條無向邊的兩端及邊長(權重)」;

輸出提示訊息「請輸入該條無向邊的兩端及邊長」;
輸入無向邊的一端存入 u，另一端存入 v 及無向邊邊長存入 w；

宣告布林變數 check 並設為 false，用於紀錄所輸入的兩端是否已有連結的邊；

```
        for j = 0 to j < vertex[u].size() (for j++){
            如果 vertex[u][j].first 等於 v，則 u 已與 v 有無向邊{
                將 check 改為 true；
                如果(vertext[u][j].second 大於 w{
                // 因原本的邊長大於目前輸入的 w，故改為 w
                    vertext[u][j].second 設為 w；
                    break；
                }
            }
        }
        for j = 0 to j < vertex[v].size() (for j++){
            如果 vertex[v][j].first 等於 u，則 v 已與 u 有無向邊{
                如果(vertext[v][j].second 大於 w{
                // 因原本的邊長大於目前輸入的 w，故改為 w
                    vertext[v][j].second 設為 w；
                    break；
                }
            }
        }

        如果 check 不為 true，代表 u 與 v 間目前沒有無向邊連結{
            將{v,w}放到 vertex[u]內的最後；
            將{u,w}放到 vertex[v]內的最後；
        }
    }

    宣告整數變數 next_vertex 並設為 0，紀錄下一個要造訪的頂點；
    宣告布林變數 have_next 並設為 true，紀錄是否還能繼續造訪；

    迴圈 (have_next 為 true){
        宣告整數變數 now_vertex 設為 next_vertex，代表造訪頂點；
        宣告整數變數 min 為 max，紀錄目前頂點往下最小的路徑；

        for auto next 依序存入 vertex[now_vertex] {
            distance[next.first]設為「distance[next.first]與
            (distance[now_vertex]加 next.second)」兩者較小的值；
```

```
                如果 distance[next.first]小於 min{
                    如果 final[next.first]不為真{
                        將 min 設為 distance[next.first]；
                        將 next_vertex 設為 next.first；
                    }
                }
            }

            將 final[next_vertex]設為 true，紀錄下個造訪頂點為完成；

            如果 min 等於 max，代表沒有下個要造訪的頂點{
                將 have_next 設為 false；
            }
        }

        宣告整數變數 max_path 及 no_visit 儲存「由 0 到某點的最長距
        離及未造訪到的頂點數」；

        for i = 0 to i < n (for i ++){
            如果 max_path 小於 distance[i]{
                如果 distance[i]等於 max{
                    將 no_visit 加 1；
                }
                否則{
                    將 max_depth 設為 distance[i]；
                }
            }
        }

        輸出「最長距離為 max_depth」;
        輸出「未被造訪的頂點數量為 no_visit」;

        return 0；
    }
```

六. 補充程式碼(無使用優先佇列)：

```
#include<bits/stdc++.h>
using namespace std;
#define max 10000000 // 定義通用的最大距離 max 為 10000000
int main(){
    int n, m;
    // 宣告整數 n, m，分別用於儲存「頂點數量及無向邊數量」
    printf("\n Input the vertex and edges : ");
    // 輸出提示訊息
    scanf("%d%d", &n, &m);
    // 將輸入的頂點數量存入 n, 無向邊數量存入 m
    vector< pair<int, int> > vertex[n];
    // 宣告 vector 型態變數 vertex 儲存資料為兩個整數

    int distance[n];
    // 宣告一個大小為 n 的整數陣列，用於儲存各點與頂點 0 的距離
    for(int i = 0 ; i < n ; i++){
        distance[i] = max;
        // 將 distance 內所有節點的值設為 max
    }
    distance[0] = 0;
    // 將頂點 0 與自己的距離設為 0

    bool final[n];
    // 建立大小為 n 的布林矩陣，紀錄該頂點是否已完成
    for(int i = 0 ; i < n ; i++){
        final[i] = false;
        // 將 final 內所有節點的值設為 false
    }
    final[0] = true;
    // 將頂點 0 設為 true

    for(int i  = 0 ; i < m ; i++){
        int u, v, w;
        // 宣告整數變數，分別用於儲存「該條無向邊的兩端及邊長(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
bool check = false;
// 宣告布林變數 check 紀錄所輸入的兩端是否已有連結的邊

for(int j = 0 ; j < vertex[u].size() ; j++){
// 搜尋 vertex[u] 內的所有資料
    if (vertex[u][j].first == v){
    // 若 vertex[u][j] 為 v，代表該條邊連結 u 與 v
        check = true;
        // 將兩端已有連結線變更為 true
        if(vertex[u][j].second > w){
        // 若 u 與 v 的無向邊長大於當前輸入的邊長則進入 if 內
            vertex[u][j].second = w;
            // 將 u 與 v 的邊長改為 w
            break;
            // 離開迴圈
        }
    }
}

for(int j = 0 ; j < vertex[v].size() ; j++){
// 搜尋 vertex[v] 內的所有資料
    if (vertex[v][j].first == u){
    // 若 vertex[v][j] 為 u，代表該條邊連結 v 與 u
        if(vertex[v][j].second > w){
        // 若 u 與 v 的無向邊長大於當前輸入的邊長則進入 if 內
            vertex[v][j].second = w;
            // 將 u 與 v 的邊長改為 w
            break;
            // 離開迴圈
        }
    }
}
```

```
        if(!check){
        // 若 u 與 v 並不存在無向邊則進入 if 內
            vertex[u].push_back({v, w});
            // 紀錄 u 節點可以通往 v 節點且邊長為 w
            vertex[v].push_back({u, w});
            // 紀錄 v 節點可以通往 u 節點且邊長為 w
        }
    }
    int next_vertex = 0;
    // 宣告整數變數 next_vertex 紀錄下一個要造訪的頂點
    bool have_next = true;
    // 宣告布林變數 have_next 紀錄是否還能繼續造訪

    while(have_next){
        int now_vertex = next_vertex;
        // 宣告整數變數 now_vertex 紀錄當前要造訪的頂點
        int min = max;
        // 宣告整數變數 min 紀錄目前頂點往下最小的路徑

        for(auto next : vertex[now_vertex]){
        // 將當前造訪頂點所連結的資料依序存入 next
            distance[next.first] = std::min((distance[now_vertex] +
next.second),(distance[next.first]));
            // 將由 0 到下個頂點的距離與 0 到目前頂點延伸至下個頂點的
距離比較，並傳回較小值儲存到由 0 到下個頂點的距離
            if(distance[next.first] < min){
            // 如果由 0 到下個頂點的距離小於 min 則進入 if 內
                if(!final[next.first]){
                // 如果下個頂點尚未完成，則進入 if 內
                    min = distance[next.first];
                    // 最小值為 0 到下個頂點的距離
                    next_vertex = next.first;
                    // 紀錄下個造訪的頂點為當前 next 紀錄的頂點
                }
            }
        }
```

```
            final[next_vertex] = true;
            // 將下個要造訪的頂點設為 true 以記錄該頂點已完成

            if(min == max){
            // 若 min 為 max 代表目前頂點沒有下一個能造訪的頂點
                have_next = false;
                // 將 have_next 設為 false
            }
        }

        int max_path = 0, no_visit = 0;
        // 宣告整數變數 max_path 及 no_visit 儲存「由 0 到某點的最長距離及未造訪到的頂點數」

        for(int i = 0 ; i < n ; i++){
            if(max_path < distance[i]){
            // 如果 max_path 比由 0 到頂點 i 的距離短，進入 if 內
                if(distance[i] == max){
                //  由 0 到頂點 i 的距離為 max 則進入 if 內
                    no_visit ++;
                    // 由 0 到頂點 i 的距離與 max 相同代表沒有被造訪到
                }
                else{
                    max_path = distance[i];
                    // 若頂點 i 有被造訪，則 max_path 為 0 到頂點 i 的距離
                }
            }
        }

        printf("\n longest path : %d", max_path);
        // 輸出由 0 到某頂點的最長距離
        printf("\n unvisited vertex : %d", no_visit);
        // 輸出未被造訪的頂點數量

        return 0;
}
```