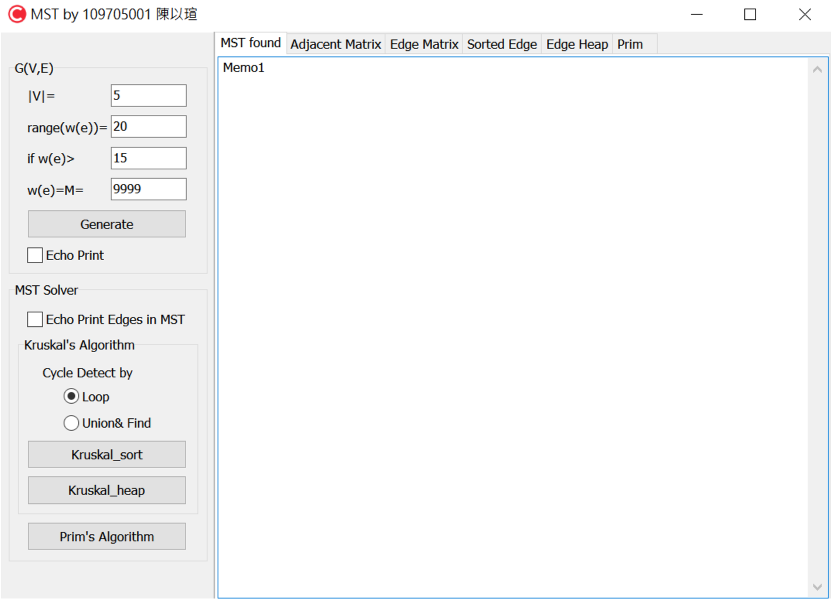
Bonus04_MST 作業報告

學號：109705001 姓名：陳以瑄

※Kruskal 與 Prim 的演算法執行時間比較圖檔 在第 6 頁

操作介面



完成的基本要求

1. **亂數產生圖 G 的相鄰矩陣，其濃密或稀疏程度可調整，使用者可選擇是否要印出**
   程式碼: MST.cpp 中第 44 行至第 137 行的部分

```
    //---------------------------------------------------------------------------
    //Generate Data
    void __fastcall TForm2::Button1Click(TObject *Sender)[...]
50  void Generate()[...]
    void EdgeMatrix()[...]
108 void PrintInGrid1(int n)[...]
    void PrintInGrid2(int n)[...]
    //
```

操作說明:

在 G(V,E)的 GroupBox 中的四個 Edit 欄位分別輸入 (1)節點個數 (2)邊的可能最大值 (3)(4)若權重大於多少就視為沒有邊相連並用 M 取代(調整疏密程度)

按下”Generate”按鈕會執行第 44 行的 Button1Click

->呼叫第 50 行的 Generate function 用亂數產生相鄰矩陣

->如果有勾選”Echo Print”會呼叫第 108 行的 PrintGrid1 function 在右側”Adjacent Matrix”頁面印出相鄰矩陣

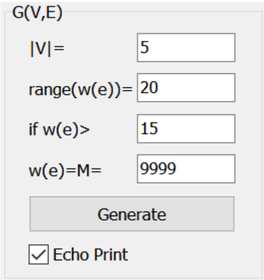
->為了之後方便做 Kruskal Algorithm 呼叫第 83 行的 EdgeMatrix function 將 |V|*|V|的相鄰矩陣轉成|E|*3 的邊矩陣

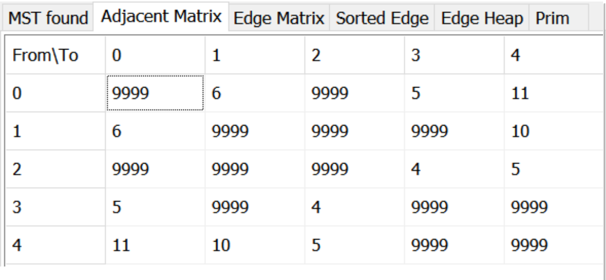
->如果有勾選”Echo Print”會呼叫第 123 行的 PrintGrid2 function 在右側”Edge Matrix”頁面印出邊矩陣

限制：|V|請大於100時請注意不要勾選”Echo Print”

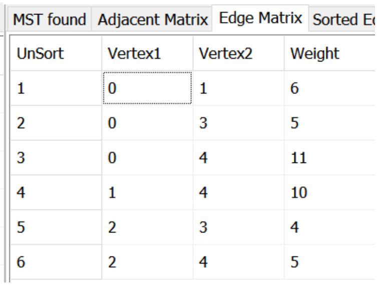
執行結果：

輸入 Adjacent Matrix Edge Ma trix





From\To	0	1	2	3	4
0	9999	6	9999	5	11
1	6	9999	9999	9999	10
2	9999	9999	9999	4	5
3	5	9999	4	9999	9999
4	11	10	5	9999	9999



UnSort	Vertex1	Vertex2	Weight
1	0	1	6
2	0	3	5
3	0	4	11
4	1	4	10
5	2	3	4
6	2	4	5

2. 利用Kruskal演算法找出G的最小延展樹，印出執行時間，此最小延展樹可讓使用者選定是否要印出

程式碼：MST.cpp中第140行至第408行的部分

```
//---------------------------------------------------------------------------
//Heap & HeapSort
140 int** CopyData...
void Restore(int ** data,int s, int r)...
void Heap()...
void HeapSort()...
void PrintInGrid3(int n)...
void PrintInGrid4(int n)...
//Cycle Detect(大號變小號)
void ResetMask()...
void CycleDetect(int min)...
//Union&Find
struc...
struct node ** NodeList;
void ResetNode()...
270 struct node * Find(struct node * ...
void Union(struct node * x,struct node *y ).. .
void UnionAndFind(int min)...
void KruskalMin(int flag)...
void KruskalHeap(int flag)...
340 //Kruskal Min
void __fastcall TForm2::Button2Click(TObject *Sender)...
//Kruskal Heap
void __fastcall TForm2::Button3Click(TObject *Sender)...
//---------------------------------------------------------------------------
```

操作說明：

生成G後

(1)按下”Kruskal_sort”按鈕會執行第341行的Button2Click

->**沒有生成G或|E|=0時**會輸出”#edges is zero .There's no MST of G.

->其他情況

->執行第179行HeapSort function將所有的邊排序

->如果有勾選”Echo Print Edge in MST” 會呼叫第194行的PrintGrid3 function在右側”Sorted Edge” 頁面印出**排序好的**邊矩陣(由小到大)

->接著呼叫第298行的KruskalMin function 執行Kruskal演算法

->(i)如果選取”loop” 會使用第234行的大號變小號cycle偵測

->(ii)如果選取”Union&Find” 會使用第 283 行的 Union&Find cycle 偵測

->如果有勾選”Echo Print Edge in MST” 且有找到 MST，會在右側”MST found” 頁面依序印出形成 MST 的邊以及偵測會形成 Cycle 的邊的數量

->在”MST found” 頁面印出 “Kruskal:(節點數,疏密程度,MST 總權重,耗時) 偵測會形成 Cycle 的邊佔全部邊的比例” **如果找不到 MST 總權重會顯示 INFINITE**

(2)按下”Kruskal_heap”按鈕會執行第 376 行的 Button3Click

->**沒有生成 G 或|E|=0 時**會輸出”#edges is zero .There's no MST of G.

->其他情況

->執行第 167 行 HeapSort function 將所有的邊排序

->如果有勾選”Echo Print Edge in MST” 會呼叫第 211 行的 PrintGrid4 function 在右側” Edge Heap”頁面印出**排序好的**邊矩陣(由小到大)

->接著呼叫第 317 行的 KruskalHeap function 執行 Kruskal 演算法

->(i)如果選取”loop” 會使用第 234 行的大號變小號 cycle 偵測

->(ii)如果選取”Union&Find” 會使用第 283 行的 Union&Find cycle 偵測

->如果有勾選”Echo Print Edge in MST” 且有找到 MST，會在右側”MST found” 頁面依序印出形成 MST 的邊以及偵測會形成 Cycle 的邊的數量

->在”MST found” 頁面印出 “Kruskal_H:(節點數,疏密程度,MST 總權重,耗時) 偵測會形成 Cycle 的邊佔全部邊的比例” **如果找不到 MST 總權重會顯示 INFINITE**

限制: |V|請大於 100 時請注意不要勾選” Echo Print Edge in MST”

使用”Kruskal_sort” 時|V|請小於 5000，如果要重複執行|V|請小於 3000，不然很可能會空間不夠

使用”Kruskal_heap” 時|V|請小於 8000，如果要重複執行|V|請小於 4000，不然很可能會空間不夠

執行結果:(有勾選”Echo Print Edge in MST”)

(1)Kruskal_sort

Edge Ma trix

MST found | Adjacent Matrix | Edge Matrix | Sorted E

UnSort	Vertex1	Vertex2	Weight
1	0	1	6
2	0	3	5
3	0	4	11
4	1	4	10
5	2	3	4
6	2	4	5

Sorted Edge

MST found | Adjacent Matrix | Edge Matrix | Sorted Edge

Sort	Vertex1	Vertex2	Weight
1	2	3	4
2	0	3	5
3	2	4	5
4	0	1	6
5	1	4	10
6	0	4	11

MST found

edge0:(2,3) [4]
edge1:(0,3) [5]
edge2:(2,4) [5]
edge3:(0,1) [6]
edges incurring cycles:0
Kruskal : (5 ,0.75 ,20 ,0(sec.)) #cycle_e/#e = 0/6~0%

(2)Kruskal_heap

Edge Ma trix

MST found | Adjacent Matrix | Edge Matrix | Sorted E

UnSort	Vertex1	Vertex2	Weight
1	0	1	6
2	0	3	5
3	0	4	11
4	1	4	10
5	2	3	4
6	2	4	5

Edge Heap

MST found | Adjacent Matrix | Edge Matrix | Sorted Edge | Edge Heap

Heap	Vertex1	Vertex2	Weight
1	2	3	4
2	0	3	5
3	2	4	5
4	1	4	10
5	0	1	6
6	0	4	11

MST found

```
edge0:(2,3) [4]
edge1:(0,3) [5]
edge2:(2,4) [5]
edge3:(0,1) [6]
# edges incurring cycles:0
Kruskal_H: (5 ,0.75 ,20 ,0.0149999996647239(sec.)) #cycle_e/#e = 0/6~0%
```

3. 利用 Prim 演算法找出 G 的最小延展樹，印出執行時間，此最小延展樹可讓使用者選定是否要印出

程式碼: MST.cpp 中第 410 行至第 570 行的部分

```
    //---------------------------------------------------------------------------
410 //Prim
    int ** color;
    void PrepareGrid5()[...]
    void Prim()[...]
530 void __fastcall TForm2::StringGrid5DrawCell(TObject *Sender, int ACol, int ARow,
546 void __fastcall TForm2::Button4Click(TObject *Sender)[...]
    //---------------------------------------------------------------------------
```

操作說明:

生成 G 後

按下”Prim’ s Algorithm”按鈕會執行第 546 行的 Button4Click

->**沒有生成 G 或|E|=0 時**會輸出”#edges is zero .There’s no MST of G.

->其他情況

->呼叫第 439 行 Prim function 執行 Prim 演算法

->如果有勾選”Echo Print Edge in MST”且有找到 MST，會在右側”MST found”頁面依序印出形成 MST 的邊

->在”MST found”頁面印出“Kruskal:(節點數,疏密程度,MST 總權重,耗時)”**如果找不到 MST 總權重會顯示 INFINITE**

限制: |V|請大於 100 時請注意不要勾選” Echo Print Edge in MST”

使用” Prim’ s Algorithm”時|V|請小於 10000，如果要重複執行|V|請小於 6000，不然很可能會空間不夠

執行結果:(有勾選”Echo Print Edge in MST”)

Adjacent Matrix

MST found | Adjacent Matrix | Edge Matrix | Sorted Edge | Edge Heap | Prim

From\To	0	1	2	3	4
0	9999	6	9999	5	11
1	6	9999	9999	9999	10
2	9999	9999	9999	4	5
3	5	9999	4	9999	9999
4	11	10	5	9999	9999

Prim

MST found | Adjacent Matrix | Edge Matrix | Sorted Edge | Edge Heap | Prim

Prim	0	1	2	3	4	Last
0	0	0	0	0	0	0
1	6	6	6	6	6	0
2	9999	4	4	4	4	3
3	5	5	5	5	5	0
4	11	11	5	5	5	2
Nearest	0	3	2	4	1	

(右圖紅色為起始點 藍色為該 iteration 的非 MST 節點最小權重 last 紀錄 MST 相鄰點)

MST found

edge1:(0,3) [5]
edge2:(2,3) [4]
edge3:(2,4) [5]
edge4:(0,1) [6]
* starting from vertex:0
Prim : (5 ,0.75 ,20 ,0.0149999996647239(sec.))

4. Kruskal 與 Prim 的演算法執行時間比較圖檔

執行資料

(1)密度為 1 (range=bound=100000)(因為空間有限所以|V|=6000,7000,8000 是另外再分次執行的)

[Nodes: 1000]
Kruskal_H: (1000 ,1 ,10689 ,0.0160000007599592(sec.)) #cycle_e/#e = 2270/499500~0.454454454454454%
Prim : (1000 ,1 ,10689 ,0(sec.))
[Nodes: 2000]
Kruskal_H: (2000 ,1 ,10761 ,0.0619999998986721(sec.)) #cycle_e/#e = 5461/1999000~0.273186593296648%
Prim : (2000 ,1 ,10761 ,0.0309999994933605(sec.))
[Nodes: 3000]
Kruskal_H: (3000 ,1 ,11158 ,0.157000005245209(sec.)) #cycle_e/#e = 9625/4498500~0.213960208958542%
Prim : (3000 ,1 ,11158 ,0.0469999983906746(sec.))
[Nodes: 4000]
Kruskal_H: (4000 ,1 ,11865 ,0.233999997377396(sec.)) #cycle_e/#e = 12667/7998000~0.158377094273568%
Prim : (4000 ,1 ,11865 ,0.0780000016093254(sec.))
[Nodes: 5000]
Kruskal_H: (5000 ,1 ,12478 ,0.375(sec.)) #cycle_e/#e = 22024/12497500~0.176227244544909%
Prim : (5000 ,1 ,12478 ,0.125(sec.))

[Nodes: 6000]
Kruskal_H: (6000 ,1 ,12950 ,0.531000018119812(sec.)) #cycle_e/#e = 24766/17997000~0.137611824192921%
Prim : (6000 ,1 ,12950 ,0.202999994158745(sec.))

[Nodes: 7000]
Kruskal_H: (7000 ,1 ,13622 ,0.718999981880188(sec.)) #cycle_e/#e = 24878/24496500~0.101557365337905%
Prim : (7000 ,1 ,13622 ,0.28099998831749(sec.))

[Nodes: 8000]
Kruskal_H: (8000 ,1 ,14130 ,0.938000023365021(sec.)) #cycle_e/#e = 29435/31996000~0.0919958744843105%
Prim : (8000 ,1 ,14130 ,0.360000014305115(sec.))

(2)密度為 0.01 (range=10000 bound=100)

[Nodes: 1000]
Kruskal_H: (1000 ,0.01 ,10501 ,0(sec.)) #cycle_e/#e = 2878/6103~47.1571358348353%
Prim : (1000 ,0.01 ,10501 ,0(sec.))
[Nodes: 2000]
Kruskal_H: (2000 ,0.01 ,11061 ,0(sec.)) #cycle_e/#e = 4134/24527~16.8548946059445%
Prim : (2000 ,0.01 ,11061 ,0.0160000007599592(sec.))
[Nodes: 3000]
Kruskal_H: (3000 ,0.01 ,10957 ,0(sec.)) #cycle_e/#e = 8403/54893~15.3079627639225%
Prim : (3000 ,0.01 ,10957 ,0.0469999983906746(sec.))
[Nodes: 4000]
Kruskal_H: (4000 ,0.01 ,12243 ,0.0309999994933605(sec.)) #cycle_e/#e = 15987/97535~16.3910391141641%
Prim : (4000 ,0.01 ,12243 ,0.0780000016093254(sec.))
[Nodes: 5000]
Kruskal_H: (5000 ,0.01 ,12416 ,0.0469999983906746(sec.)) #cycle_e/#e = 17558/152235~11.5334844155418%
Prim : (5000 ,0.01 ,12416 ,0.141000002622604(sec.))
[Nodes: 6000]
Kruskal_H: (6000 ,0.01 ,13096 ,0.0469999983906746(sec.)) #cycle_e/#e = 26043/220298~11.8217142234609%
Prim : (6000 ,0.01 ,13096 ,0.187999993562698(sec.))
[Nodes: 7000]
Kruskal_H: (7000 ,0.01 ,13612 ,0.0780000016093254(sec.)) #cycle_e/#e = 31414/298687~10.517364331223%
Prim : (7000 ,0.01 ,13612 ,0.25(sec.))
[Nodes: 8000]
Kruskal_H: (8000 ,0.01 ,14241 ,0.0939999967813492(sec.)) #cycle_e/#e = 30625/390252~7.84749341451165%
Prim : (8000 ,0.01 ,14241 ,0.375(sec.))

比較圖