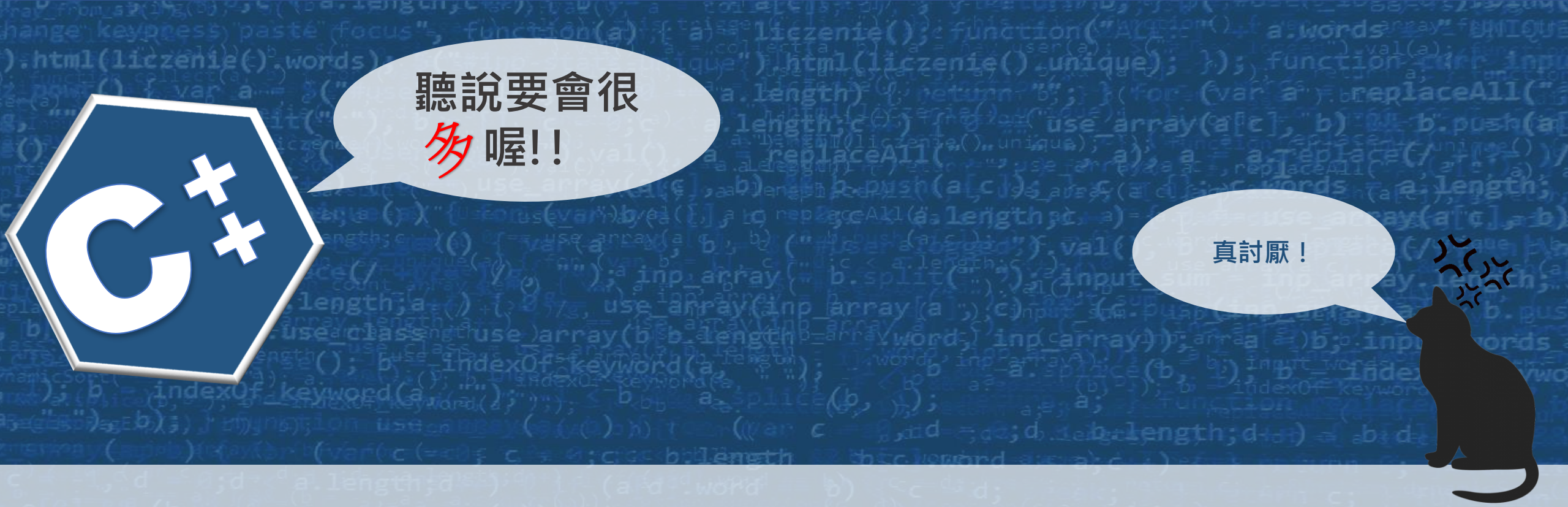


推廣教育資料結構與演算法
Topic 7 圖論

Kuan-Teng Liao (廖冠登)

2021/06/12

大綱

概論	BFS& DFS	最短距離	循環偵測	最小生成樹
		貪婪法		

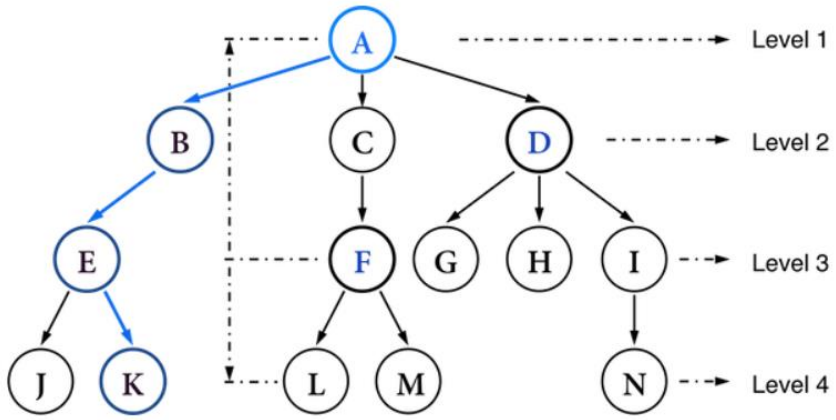
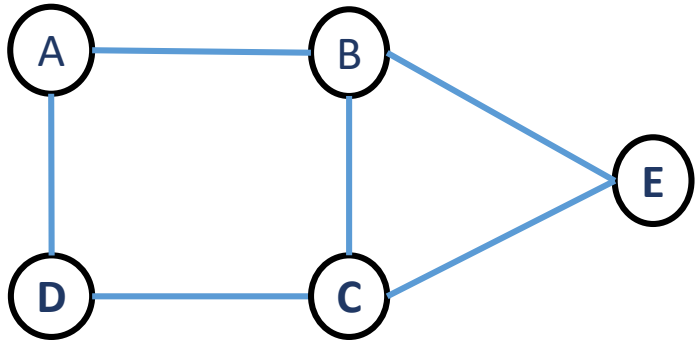
圖論(Graph) (1)

- 何謂圖(graph)
  - ✓定義
    - 為頂/端點V(vertex)與邊線E(edge)所形成的集合，因此圖可以表示成G(V,E)
  - ✓特色
    - 跟樹很像，在圖論中，強調的是點與點之間的連通(~~不是人與人之間的連結~~)，因此跟樹比較，圖論不一定會有與可能會有部分如下表所示

樹	圖論
• 必有根結點 • 每一個節點下都可以由彼此互斥(disjoint)的子樹與節點構成，因此不允許循環產生 • 只允許子節點來源只有一個父節點	• 不一定有根結點 • 邊與點允許構成循環(cycle) • 允許子端點於來源，可以來自不同父節點

圖論(Graph) (2)

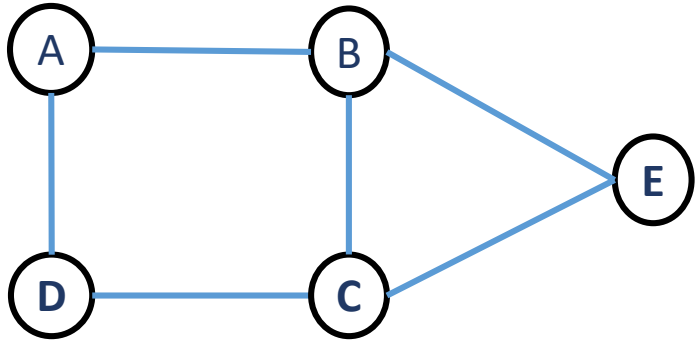
- E.g.,

樹	圖論
	

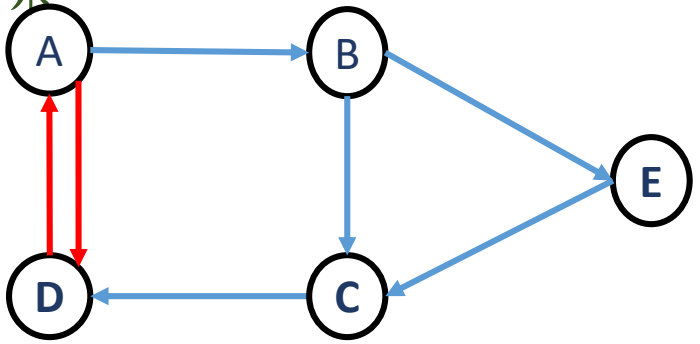
- ABCD為循環
- $G = \{\{V_A, V_B, V_C, V_D, V_E\}, \{E_{AB}, E_{BC}, E_{CD}, E_{AD}, E_{BE}, E_{CE}\}\}$
- 到E可以來自B或 C點

圖論(Graph) (3)

- 一般來說，圖的種類有
  - ✓無向圖
    - 即邊可代表兩端點可以互相連通



  - ✓有向圖
    - 即邊具有方向性代表只能從一端點至另一端點，而另一端點無法回至原端點；該邊會以箭號表示

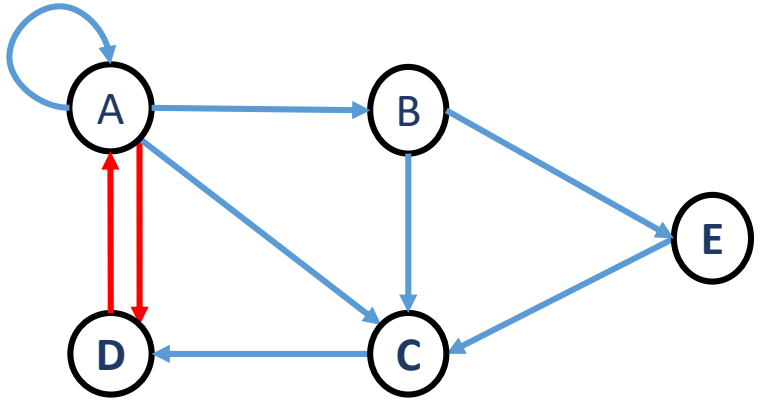


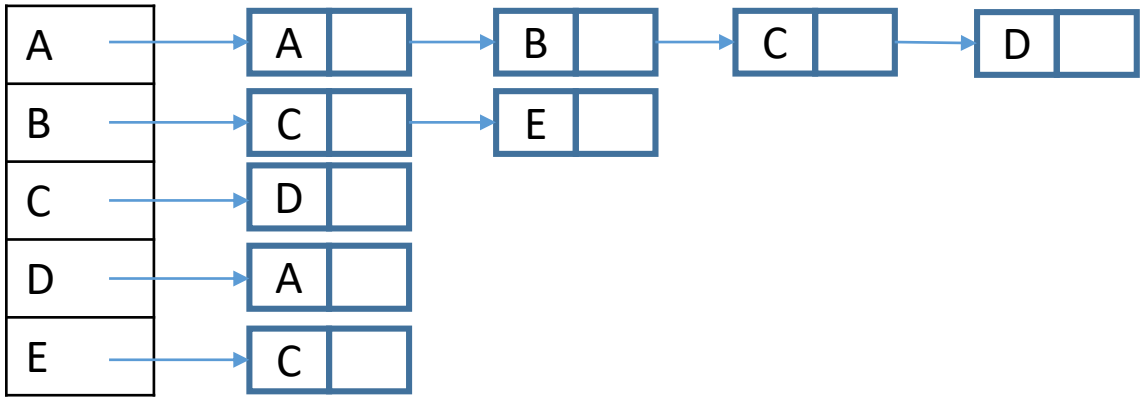
圖論(Graph) (4)

- 學圖論時需要了解以下術語
  - ✓相鄰(**adjacent**)：vertex(A)與vertex(B)之間有一條edge;不論方向性
  - ✓路徑(**path**)：從vertex(A)到vertex(B)，藉由edge所經過的所有vertex的**edge**
  - ✓長度/距離(length/distance)：path中，edge的數目
  - ✓簡單路徑(simple path)：若一條path中，除了起點vertex與終點vertex之外，**沒有vertex被重複經過**，則稱這條path為簡單路徑
  - ✓循環(cycle)：若有一條「簡單路徑」的起點vertex與終點vertex相同
  - ✓權重(weight)：於邊上可以註記述值，作為兩點之間的距離或是其他意義
  - ✓自我迴圈(self-loop)：若有edge從vertex(A)指向vertex(A)，即稱為self-edge或是self-loop

圖論(Graph) (2)

- 圖學vs.資料結構
  - ✓實作方法
    - 鄰近串列(Adjacency list)





圖論(Graph) (3)

main.cpp

```cpp
#include <algorithm>
#include <iostream>

struct Vet{
    public: char c_Name;
    public: Vet* clsp_Ptr;
};

struct AdList{
    public: Vet** cl2p_Ptr;
    public: char* cp_Name;
    public: int i_Size;

    public: AdList(char* cp_Name, int i_Size){
        cl2p_Ptr= new Vet*[i_Size];
        std::fill(cl2p_Ptr, cl2p_Ptr+ i_Size, nullptr);
        this->cp_Name = cp_Name;
        this->i_Size = i_Size;
    }
```

main.cpp

```cpp
    public: bool fn_InsVet(char c_NameA, char c_NameB){
        int i_Ind = -1;
        for(int i_Ct= 0; i_Ct < i_Size; i_Ct++){
            if(c_NameA == cp_Name[i_Ct]){
                i_Ind = i_Ct;
                break;
            }
        }
        if(i_Ind == -1){
            return false;
        }

        Vet* o_Ele  = new Vet();
        o_Ele->c_Name = c_NameB;
        o_Ele->clsp_Ptr = cl2p_Ptr[i_Ind];
        cl2p_Ptr[i_Ind] = o_Ele;

    }
```

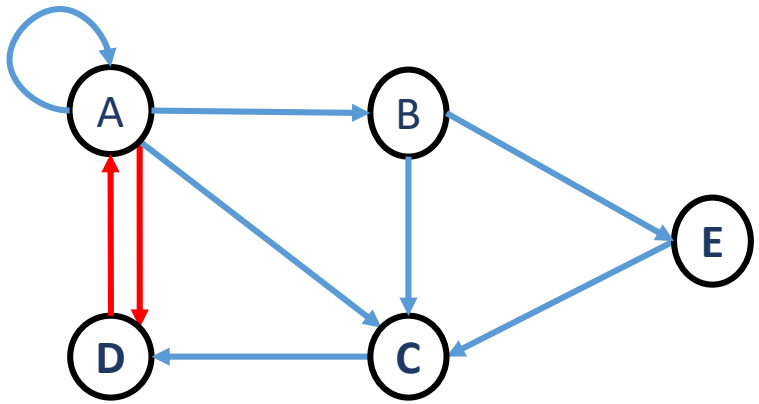
圖論(Graph) (4)

main.cpp

```
40      public: ~AdList(){
41          for(int i_Ct =0; i_Ct< i_Size; i_Ct++){
42              for(Vet* o_Ele = cl2p_Ptr[i_Ct];
43                  o_Ele != nullptr;){
44                  Vet* o_Ele2  = o_Ele->clsp_Ptr;
45                  delete o_Ele;
46                  o_Ele = o_Ele2;
47              }
48          }
49          delete [] cl2p_Ptr;
50      }
51  };
```

main.cpp

```
53  int main(){
54      char c_Arr[] = {'A', 'B', 'C', 'D', 'E'};
55      int i_Size  = sizeof(c_Arr)/sizeof(char);
56      AdList o_List = AdList(c_Arr, i_Size);
57      o_List.fn_InsVet('A', 'A');
58      o_List.fn_InsVet('A', 'B');
59      o_List.fn_InsVet('A', 'C');
60      o_List.fn_InsVet('A', 'D');
61
62      for(Vet* o_Ele = (o_List.cl2p_Ptr)[0];
63                  o_Ele!= nullptr; o_Ele = o_Ele->clsp_Ptr){
64
65          std::cout<< o_Ele->c_Name << " ";
66      }
67
68      return 0;
69  }
```

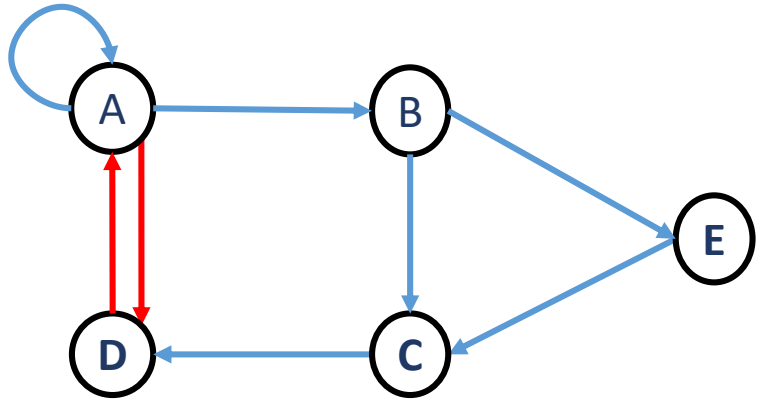



```
E:\CodeWorkShop\CodeBlock\ProjectDsAlg\GraphAdList\bin\De
D C B A
Process returned 0 (0x0)   execution time : 0.035 s
Press any key to continue.
```

程式碼網址：https://github.com/altoliaw2/GraphAdList

圖論(Graph) (5)

- 圖學vs.資料結構
  - ✓實作方法
    - 鄰近陣列(Adjacency array)
      - 於有序圖記錄有序的相鄰情況
      - 於無序圖則需雙邊記錄相鄰情況



	A	B	C	D	E
A	1	1	0	1	0
B	0	0	1	0	1
C	0	0	0	1	0
D	1	0	0	0	0
E	0	0	1	0	0

圖論(Graph) (6)

main.cpp

```cpp
#include <iostream>

struct AdList{
    public: int** cl2p_Ptr;
    public: char* cp_Name;
    public: int i_Size;

    public: AdList(char* cp_Name, int i_Size){
        cl2p_Ptr= new int*[i_Size];
        for(int i_Ct=0; i_Ct< i_Size; i_Ct++){
            cl2p_Ptr[i_Ct] = new int[i_Size]{};
        }
        this->cp_Name = cp_Name;
        this->i_Size = i_Size;
    }
```

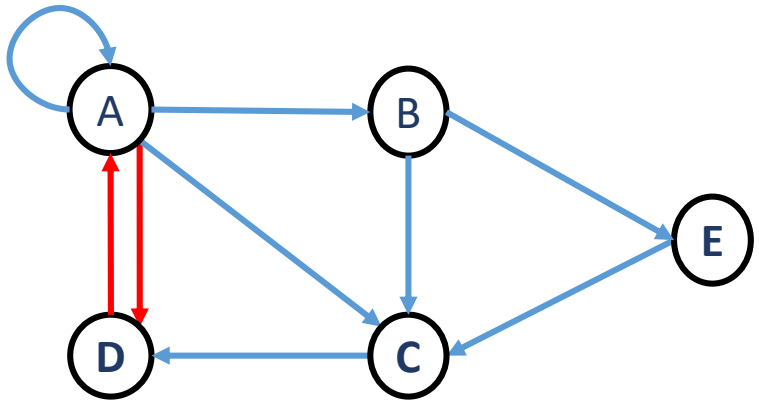
main.cpp

```cpp
    public: bool fn_InsVet(char c_NameA, char c_NameB){
        int i_RInd = -1;
        int i_CInd = -1;
        for(int i_Ct= 0; i_Ct < i_Size; i_Ct++){
            if(c_NameA == cp_Name[i_Ct]){
                i_RInd = i_Ct;
                break;
            }
        }
        for(int i_Ct= 0; i_Ct < i_Size; i_Ct++){
            if(c_NameB == cp_Name[i_Ct]){
                i_CInd = i_Ct;
                break;
            }
        }
        if(i_RInd == -1 || i_CInd == -1){
            return false;
        }
        cl2p_Ptr[i_RInd][i_CInd] = 1;
    }
```

圖論(Graph) (7)

main.cpp

```
38      public: ~AdList(){
39          for(int i_Ct =0; i_Ct< i_Size; i_Ct++){
40              delete [] cl2p_Ptr[i_Ct];
41          }
42          delete [] cl2p_Ptr;
43      }
44  };
```



程式碼網址：https://github.com/altoliaw2/GraphAdArray

main.cpp

```
46  int main(){
47      char c_Arr[] = {'A', 'B', 'C', 'D', 'E'};
48      int i_Size  = sizeof(c_Arr)/sizeof(char);
49      AdList o_List = AdList(c_Arr, i_Size);
50      o_List.fn_InsVet('A', 'A');
51      o_List.fn_InsVet('A', 'B');
52      o_List.fn_InsVet('A', 'C');
53      o_List.fn_InsVet('A', 'D');
54
55      for(int i_Ct= 0; i_Ct < i_Size; i_Ct++){
56          for(int i_Ct2= 0; i_Ct2 < i_Size; i_Ct2++){
57              std::cout<< o_List.cl2p_Ptr[i_Ct][i_Ct2] << " ";
58          }
59          std::cout<< "\n";
60      }
61
62      return 0;
63  }
```

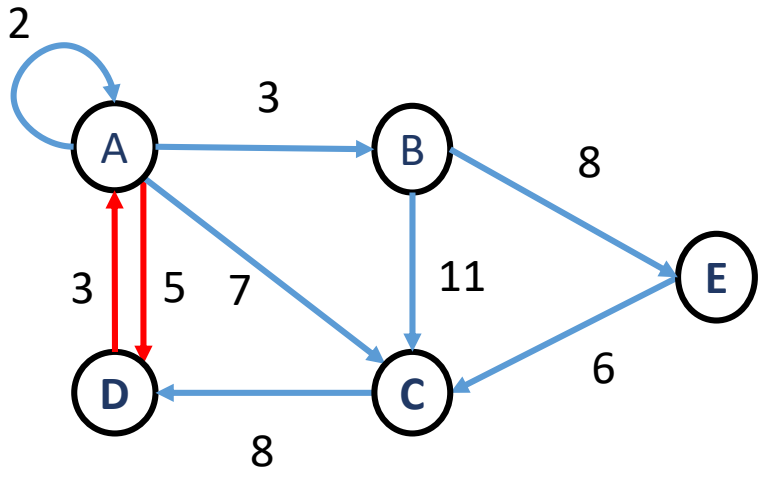
```
E:\CodeWorkShop\CodeBlock\ProjectDsAlg\GraphAdArray\bin\D
1 1 1 1 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0

Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

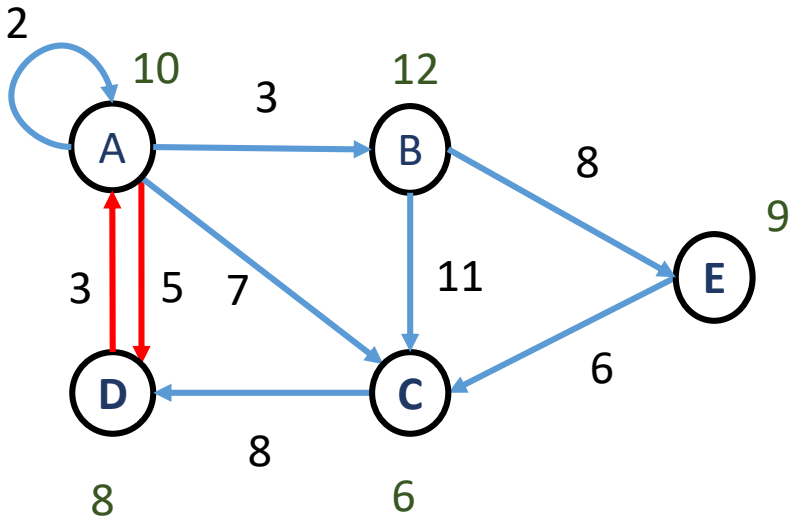
圖論(Graph) (8)

- 相鄰串列與相鄰陣列法比較

	相鄰串列	相鄰陣列
差異點	• 使用結構配合一維陣列再搭配串列可生成 • 不會發生稀疏矩陣的狀態 • 時間複雜度為 $O(V+E)$，V 為圖中點的數目，E為圖中邊線的個數 • 若每條邊有其權重，可以於結構內設計欄位(p. 14左圖) • 若每個點有其權重，可以於結構內設計欄位(p. 14右圖)	• 使用二維陣列即可生成 • 會出現稀疏矩陣的狀況 • 時間複雜度為 $O(V^2)$，V 為圖中點的數目 • 若每條邊有其權重，可以於陣列內內儲存邊的權重(p. 14左圖) • 若每個點有其權重，需變形或另外產生其他儲存的方法(p. 14右圖)

圖論(Graph) (8)





練習 7-1(1)

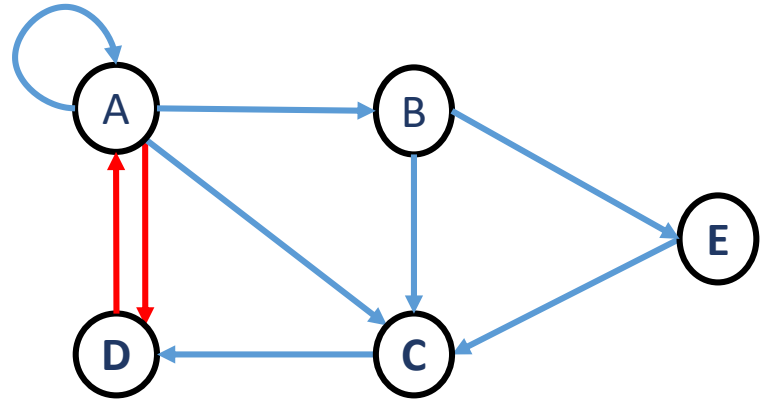
- 有一個小鎮，需要找出該小鎮是否有法官。假設今天有N個人編號為1至N，裡面若有法官則回傳人員編號否則回傳-1，法官與非法官的條件如下
  - ✓法官不會相信任何人
  - ✓除了法官本人，非法官的普通人會相信法官
  - ✓最多只會有一個法官滿足上述條件
- 輸入，共*m+1*列
  - ✓第一列有兩個輸入分別為N個人，接一空白後輸入代表接下來有*m*條關聯
  - ✓每條關聯皆有兩個輸入，輸入間以空白隔開
    - E.g., 1 2 代表 1號人信任2號人
- 輸出
  - ✓輸出是法官的人員編號

練習 7-1(2)

輸入	輸出
2 1 1 2	2
3 2 1 3 2 3	3
3 3 1 3 2 3 3 1	-1

圖論議題 (1)

- 尋訪(Traversal)
  - ✓BFS
    - 寬度尋訪
      - A D C B E
  - ✓DFS
    - 深度尋訪
      - D C E B A



圖論議題 (2)

- 尋訪(Traversal) (cont'd)
  - ✓BFS (相鄰串列法)

main.cpp

```
53
54      public: void fn_BFS(){
55          bool b_IsVis[i_Size];
56          std::fill(b_IsVis, b_IsVis+ i_Size, false);
57
58          std::queue<char> o_Qu;
59
60          for(int i_Ct =0; i_Ct < i_Size; i_Ct++){
61              int i_Ind = fn_GetIndex(cp_Name[i_Ct]);
62
63              if(i_Ind >=0 && i_Ind < i_Size
64                 && b_IsVis[i_Ind] == false){
65                  o_Qu.push(cp_Name[i_Ct]);
66                  b_IsVis [i_Ind] = true;
67
68                  for(;o_Qu.empty() == false;){
69                      std::cout<< o_Qu.front() << " ";
70                      o_Qu.pop();
71                  }
72              }
73
74              i_Ind = fn_GetIndex(cl2p_Ptr[i_Ct]->c_Name);
```

main.cpp

```
75              if(i_Ind >=0 && i_Ind < i_Size
76                 && b_IsVis[i_Ind] == false){
77
78                  o_Qu.push(cl2p_Ptr[i_Ct]->c_Name);
79                  b_IsVis [i_Ind] = true;
80
81                  for(;o_Qu.empty() == false;){
82                      std::cout<< o_Qu.front() << " ";
83                      o_Qu.pop();
84
85                      for(Vet* o_Tmp = cl2p_Ptr[i_Ct]->clsp_Ptr;
86                          o_Tmp != nullptr;
87                          o_Tmp = o_Tmp->clsp_Ptr){
88
89                          int i_TmpInd = fn_GetIndex(o_Tmp->c_Name);
90                          if(i_TmpInd >=0 && i_TmpInd < i_Size
91                             && b_IsVis[i_TmpInd] == false){
92
93                              o_Qu.push(o_Tmp->c_Name);
94                              b_IsVis [i_TmpInd] = true;
95                          }
96                      }
97                  }
98              }
99          }
100     }
```

圖論議題 (3)

- 尋訪(Traversal) (cont'd)
  - ✓BFS (相鄰串列法)

main.cpp

```
102     private: int fn_GetIndex(char c_Name){
103         int i_Ind = -1;
104         for(int i_Ct =0; i_Ct < i_Size; i_Ct++){
105                 if(cp_Name[i_Ct] == c_Name){
106                     i_Ind= i_Ct;
107                     break;
108                 }
109         }
110         return i_Ind;
111     }
```

main.cpp

```
114 int main(){
115     char c_Arr[] = {'A', 'B', 'C', 'D', 'E'};
116     int i_Size  = sizeof(c_Arr)/sizeof(char);
117     AdList o_List = AdList(c_Arr, i_Size);
118     o_List.fn_InsVet('A', 'A');
119     o_List.fn_InsVet('A', 'B');
120     o_List.fn_InsVet('A', 'C');
121     o_List.fn_InsVet('A', 'D');
122
123     o_List.fn_InsVet('B', 'C');
124     o_List.fn_InsVet('B', 'E');
125
126     o_List.fn_InsVet('C', 'D');
127
128     o_List.fn_InsVet('D', 'A');
129
130     o_List.fn_InsVet('E', 'C');
131
132     o_List.fn_BFS();
133
134     return 0;
135 }
```

```
E:\CodeWorkShop\CodeBlock\ProjectDsAlg\GraphAdList\bin\De
A D C B E
Process returned 0 (0x0)   execution time : 0.065 s
Press any key to continue.
```

程式碼網址：https://github.com/altoliaw2/GraphAdList/tree/BFS

圖論議題 (4)

- 尋訪(Traversal)
  - ✓DFS (相鄰串列法)

main.cpp

```
52      public: void fn_DFS(){
53          bool b_IsVis[i_Size];
54          std::fill(b_IsVis, b_IsVis+ i_Size, false);
55
56          for(int i_Ct =0; i_Ct < i_Size; i_Ct++){
57              int i_Ind = fn_GetIndex(cp_Name[i_Ct]);
58
59              if(i_Ind >=0 && i_Ind < i_Size
60                 && b_IsVis[i_Ind] == false){
61                  b_IsVis [i_Ind] = true;
62                  fn_DFSLink(b_IsVis, i_Size, i_Ct);
63                  std::cout<< cp_Name[i_Ct] << " ";
64              }
65          }
66      }
67
68      public: void fn_DFSLink(bool* bp_IsVis, int i_Size,
69                              int i_Curr){
70          for(Vet* op_Tmp = cl2p_Ptr[i_Curr]; op_Tmp != nullptr;
71              op_Tmp= op_Tmp->clsp_Ptr){
72              int i_Ind = fn_GetIndex(op_Tmp->c_Name);
73              if(i_Ind >=0 && i_Ind < i_Size
74                 && bp_IsVis[i_Ind] == false){
75                  bp_IsVis [i_Ind] = true;
76                  fn_DFSLink(bp_IsVis, i_Size, i_Ind);
77                  std::cout<< op_Tmp->c_Name << " ";
78              }
79          }
80      }
```

main.cpp

```
81      private: int fn_GetIndex(char c_Name){
82          int i_Ind = -1;
83          for(int i_Ct =0; i_Ct < i_Size; i_Ct++){
84                  if(cp_Name[i_Ct] == c_Name){
85                      i_Ind= i_Ct;
86                      break;
87                  }
88          }
89          return i_Ind;
90      }
```

圖論議題 (4)

- 尋訪(Traversal) (cont'd)
  - ✓DFS (相鄰串列法)

main.cpp

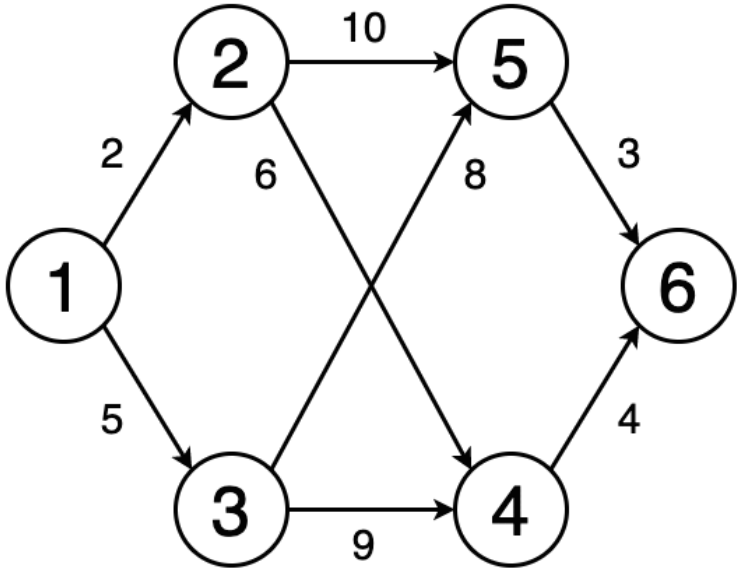
```
94  int main(){
95  int main(){
96      char c_Arr[] = {'A', 'B', 'C', 'D', 'E'};
97      int i_Size  = sizeof(c_Arr)/sizeof(char);
98      AdList o_List = AdList(c_Arr, i_Size);
99      o_List.fn_InsVet('A', 'A');
100     o_List.fn_InsVet('A', 'B');
101     o_List.fn_InsVet('A', 'C');
102     o_List.fn_InsVet('A', 'D');
103
104     o_List.fn_InsVet('B', 'C');
105     o_List.fn_InsVet('B', 'E');
106
107     o_List.fn_InsVet('C', 'D');
108
109     o_List.fn_InsVet('D', 'A');
110
111     o_List.fn_InsVet('E', 'C');
112
113     o_List.fn_DFS();
114
115     return 0;
116 }
```

```
E:\CodeWorkShop\CodeBlock\ProjectDsAlg\GraphAdList\bin\Debug\Gra
D C E B A
Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

程式碼網址：https://github.com/altoliaw2/GraphAdList/tree/DFS

圖論—最短距離議題 (1)

- 最短距離
  - ✓如下圖，有6個點，分別為1至6，每個點之間存在最少一條路且該條路上標有兩點相距之距離；除此之外，點與點間之路有行進方向性，假設從點1開始走，走至點6，請找出最短的距離。



	1	2	3	4	5	6
1	0	2	5	0	0	0
2	0	0	0	6	10	0
3	0	0	0	9	8	0
4	0	0	0	0	0	4
5	0	0	0	0	0	3
6	0	0	0	0	0	0

圖論—最短距離議題 (2)

- 最短距離(cont'd)
  - ✓窮舉法
    - 謝謝
  - ✓不過上過DP的看到窮舉法可以嘗試用DP去解

	1	2	3	4	5	6
1	0	2	5	0	0	0
2	0	0	0	6	10	0
3	0	0	0	9	8	0
4	0	0	0	0	0	4
5	0	0	0	0	0	3
6	0	0	0	0	0	0

	1	2	3	4	5	6
	0	max	max	max	max	max

	1	2	3	4	5	6
	0	2	5	max	max	max

圖論—最短距離議題 (3)

- 最短距離(cont'd)
  - ✓窮舉法
    - 謝謝
  - ✓不過上過DP的看到窮舉法可以嘗試用DP去解

	1	2	3	4	5	6
1	0	2	5	0	0	0
2	0	0	0	6	10	0
3	0	0	0	9	8	0
4	0	0	0	0	0	4
5	0	0	0	0	0	3
6	0	0	0	0	0	0

	1	2	3	4	5	6
	0	2	5	max	max	max

	1	2	3	4	5	6
	0	2	5	8	12	max

圖論—最短距離議題 (4)

- 最短距離(cont'd)
  - ✓窮舉法
    - 謝謝
  - ✓不過上過DP的看到窮舉法可以嘗試用DP去解

	1	2	3	4	5	6
1	0	2	5	0	0	0
2	0	0	0	6	10	0
3	0	0	0	9	8	0
4	0	0	0	0	0	4
5	0	0	0	0	0	3
6	0	0	0	0	0	0

	1	2	3	4	5	6
	0	2	5	8	12	max

	1	2	3	4	5	6
	0	2	5	8	12	max

圖論—最短距離議題 (5)

- 最短距離(cont'd)
  - ✓窮舉法
    - 謝謝
  - ✓不過上過DP的看到窮舉法可以嘗試用DP去解

	1	2	3	4	5	6
1	0	2	5	0	0	0
2	0	0	0	6	10	0
3	0	0	0	9	8	0
4	0	0	0	0	0	4
5	0	0	0	0	0	3
6	0	0	0	0	0	0

	1	2	3	4	5	6
	0	2	5	8	12	max

	1	2	3	4	5	6
	0	2	5	8	12	12

圖論—最短距離議題 (6)

- 最短距離(cont'd)
  - ✓窮舉法
    - 謝謝
  - ✓不過上過DP的看到窮舉法可以嘗試用DP去解

	1	2	3	4	5	6
1	0	2	5	0	0	0
2	0	0	0	6	10	0
3	0	0	0	9	8	0
4	0	0	0	0	0	4
5	0	0	0	0	0	3
6	0	0	0	0	0	0

	1	2	3	4	5	6
	0	2	5	8	12	max

	1	2	3	4	5	6
	0	2	5	8	12	12

圖論—最短距離議題 (7)

main.cpp

```cpp
#include <algorithm>
#include <iostream>
#include <limits>

int fn_MinDis(int* ip_Dist, bool* bp_SptSet, int i_Size){
    int i_Min = std::numeric_limits<int>::max();
    int i_MinInd = 0;

    for (int i_Ct = 0; i_Ct < i_Size; i_Ct++){
        if (bp_SptSet[i_Ct] == false &&
            ip_Dist[i_Ct] <= i_Min){

            i_Min = ip_Dist[i_Ct];
            i_MinInd = i_Ct;
        }
    }
    return i_MinInd;
}
```

main.cpp

```cpp
void fn_ShowSol(int* ip_Dist, int i_Size){
    //std::cout<< ip_Dist[i_Size-1];
    for(int i_Ct=0; i_Ct< i_Size; i_Ct++){
        std::cout<< ip_Dist[i_Ct] << " ";
    }
}

void fn_Dijkstra(int** i2p_2DMap, int i_SrcInd, int i_Size){
    int ia_Dist[i_Size];
    bool ba_SptSet[i_Size];

    std::fill(ia_Dist, ia_Dist+ i_Size, std::numeric_limits<int>::max());
    std::fill(ba_SptSet, ba_SptSet+ i_Size, false);

    ia_Dist[i_SrcInd] = 0;
    fn_ShowSol(ia_Dist, i_Size);
    std::cout<< "\n";
    for (int i_Ct= 0; i_Ct < i_Size - 1; i_Ct++) {
        int i_MinInd = fn_MinDis(ia_Dist, ba_SptSet, i_Size);
        ba_SptSet[i_MinInd] = true;

        for (int i_Ct2 = 0; i_Ct2 < i_Size; i_Ct2++){
            if (!ba_SptSet[i_Ct2] && i2p_2DMap[i_MinInd][i_Ct2] > 0 &&
                 ia_Dist[i_MinInd] != std::numeric_limits<int>::max()
                && ia_Dist[i_MinInd] + i2p_2DMap[i_MinInd][i_Ct2] < ia_Dist[i_Ct2]){
                    ia_Dist[i_Ct2] = ia_Dist[i_MinInd] + i2p_2DMap[i_MinInd][i_Ct2];
            }
        }
        std::cout<< "\n";
        fn_ShowSol(ia_Dist, i_Size);
    }
}
```

圖論—最短距離議題 (8)

main.cpp

```
53  int main(){
54      int i_Size = 6;
55      int** i2p_2DMap =new int*[i_Size];
56      for(int i_Ct=0; i_Ct< i_Size ; i_Ct++){
57          i2p_2DMap[i_Ct] =new int[i_Size];
58          std::fill(i2p_2DMap[i_Ct],
59                    i2p_2DMap[i_Ct]+ i_Size , 0);
60      }
61
62      i2p_2DMap[0][1] = 2;
63      i2p_2DMap[0][2] = 5;
64
65      i2p_2DMap[1][3] = 6;
66      i2p_2DMap[1][4] = 10;
67
68      i2p_2DMap[2][3] = 9;
69      i2p_2DMap[2][4] = 8;
70
71      i2p_2DMap[3][5] = 4;
72
73      i2p_2DMap[4][5] = 3;
74
75
76
77      fn_Dijkstra(i2p_2DMap, 0,  i_Size);
78
79      for(int i_Ct=0; i_Ct< i_Size; i_Ct++){
80          delete [] i2p_2DMap[i_Ct];
81      }
82      delete [] i2p_2DMap;
83
84      return 0;
85  }
```

```
E:\CodeWorkShop\CodeBlock\ProjectDsAlg\GraphDijk\bin\Debug\Graph
0 2147483647 2147483647 2147483647 2147483647 2147483647

0 2 5 2147483647 2147483647 2147483647
0 2 5 8 12 2147483647
0 2 5 8 12 2147483647
0 2 5 8 12 12
0 2 5 8 12 12
Process returned 0 (0x0)   execution time : 0.065 s
Press any key to continue.
```

程式碼網址：https://github.com/altoliaw2/GraphDijk

圖論—最短距離議題 (9)

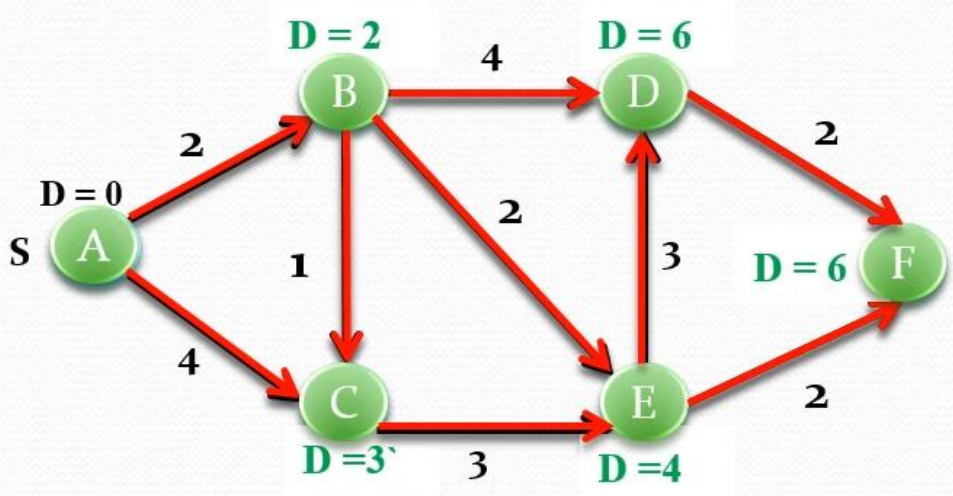
- 最短距離(cont'd)
  - ✓又稱Dijkstra's algorithm(找出最短距離方法)
  - ✓時間複雜度，為$O(V^2)$
- Dijkstra's algorithm法，真正解法
  - ✓可參考https://www.youtube.com/watch?v=Lfb8qkXzHY0
  - ✓注意
    - **邊的權重不可有負值**
    - 屬於貪婪法(greedy)內的一種，因為每次計算都在算在當下最好的情況距離

圖論—最短距離議題 (10)

Dijkstra's Rules

Rule 1: Make sure there is no negative edges. Set distance to source vertex as zero and set all other distances to infinity.
Rule 2: Relax all vertices adjacent to the current vertex.
Rule 3: Choose the closest vertex as next current vertex.
Rule 4: Repeat Rule2 and Rule 3 until the queue or reach the destination.

If (D[C] + D[AdjEdge]} < D [Adj]) {Update Adj's D with new shortest path}



Q <= V	A	B	C	D	E	F
A	0^A	2^A	4^A	∞^A	∞^A	∞^A
B	0^A	2^A	3^B	6^B	4^B	∞^A
C	0^A	2^A	3^B	6^B	4^B	∞^A
E	0^A	2^A	3^B	6^B	4^B	6^E
D	0^A	2^A	3^B	6^B	4^B	6^E
F	0^A	2^A	3^B	6^B	4^B	6^E

圖論—最短距離議題 (11)

- Dijkstra‘s algorithm法，時間複雜度
  - ✓ $O(n^2)$

貪婪法(Greedy) (1)

- 是一種在每一步選擇中都採取在當前狀態下最好或最佳（即最有利）的選擇，從而希望導致結果是最好或最佳的演算法
- 但整體來看，利用貪婪法的結果「**不一定為最好的結果**」
- 貪婪演算法與動態規劃的不同在於它對每個子問題的解決方案都做出選擇，不能回退。動態規劃則會儲存以前的運算結果，並根據以前的結果對當前進行選擇，有回退功能。
- 思路
  - ✓在每一步選擇中都採取在當前狀態下最好或最佳
    - E.g., 最短路徑中，每次都選最短的路徑的點來走
    - E.g, 換硬幣使的硬幣樹換完後最少，一定從幣值大的開始換

貪婪法(Greedy) (2)

✓E.g.,

- 要把 *n* 個 1 元硬幣兌換成 50 元, 10 元, 5 元, 1 元硬幣, 如何兌換可以讓最終的硬幣數最少?
  - 使用貪婪法一定先從能兌換面額最多的開始兌換，這樣使的硬幣數最少
  - 若*n*= 41，50 * 0 + 10 * 4 + 1 * 1
- 同樣問題把幣額改成25 元, 18 元, 5 元, 1 元硬幣
  - 若*n*= 41，貪婪法會得到： 25 *1 + 18 * 0 + 5 * 3 + 1 * 1
  - 若*n*= 41，但是最佳解是： 25 *0 + 18 * 2 + 5 * 1 + 1 * 1

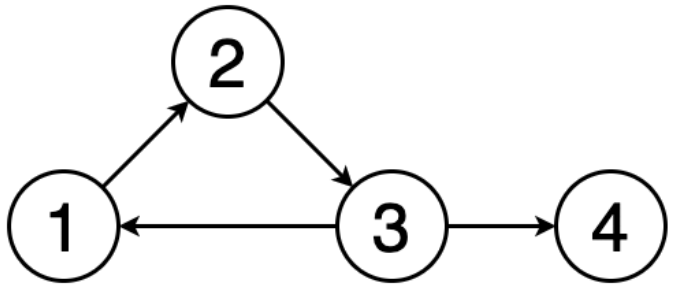
- 結論

✓所以當只問最佳解，並不管整體結果時，單用貪婪法會比較好

✓若要達到整體最佳解，則需要要用動態規劃

圖論—循環檢測議題 (1)

- 循環檢測 (Cycle detection)
  - ✓若為單純的Linked List
    - 可使用Floyd's Algorithm
    - 又名「龜兔賽跑法」Tortoise and Hare Algorithm
    - 用來檢驗一張圖裝是否存在循環
      - 使用方法：兩個指標，一個移動1一個移動2
  - ✓若為圖
    - 使用DFS方法進行檢測



圖論—循環檢測議題 (2)

- 循環檢測 (Cycle detection) (cont'd)

main.cpp

```
1  #include <algorithm>
2  #include <iostream>
3
4  struct Vet{
5      public: char c_Name;
6      public: Vet* clsp_Ptr;
7  };
8
9  struct AdList{
10     public: Vet** cl2p_Ptr;
11     public: char* cp_Name;
12     public: int i_Size;
13
14     public: AdList(char* cp_Name, int i_Size){
15         cl2p_Ptr= new Vet*[i_Size];
16         std::fill(cl2p_Ptr, cl2p_Ptr+ i_Size, nullptr);
17         this->cp_Name = cp_Name;
18         this->i_Size = i_Size;
19     }
```

main.cpp

```
21     public: bool fn_InsVet(char c_NameA, char c_NameB){
22         int i_Ind = -1;
23         for(int i_Ct= 0; i_Ct < i_Size; i_Ct++){
24             if(c_NameA == cp_Name[i_Ct]){
25                 i_Ind = i_Ct;
26                 break;
27             }
28         }
29         if(i_Ind == -1){
30             return false;
31         }
32
33         Vet* o_Ele  = new Vet();
34         o_Ele->c_Name = c_NameB;
35         o_Ele->clsp_Ptr = cl2p_Ptr[i_Ind];
36         cl2p_Ptr[i_Ind] = o_Ele;
37     }
38
39     public: ~AdList(){
40         for(int i_Ct =0; i_Ct< i_Size; i_Ct++){
41             for(Vet* o_Ele = cl2p_Ptr[i_Ct];
42                 o_Ele != nullptr;){
43                 Vet* o_Ele2  = o_Ele->clsp_Ptr;
44                 delete o_Ele;
45                 o_Ele = o_Ele2;
46             }
47         }
48         delete [] cl2p_Ptr;
49     }
```

圖論—循環檢測議題 (3)

- 循環檢測 (Cycle detection) (cont'd)

main.cpp

```
51      public: bool fn_isCyclic(){
52          bool* bp_Visited = new bool[i_Size];
53          bool* bp_RecStack = new bool[i_Size];
54          std::fill(bp_Visited, bp_Visited+ i_Size, false);
55          std::fill(bp_RecStack, bp_RecStack+ i_Size, false);
56
57          for(int i_Ct = 0; i_Ct < i_Size; i_Ct++){
58              if (fn_IsCyclic(i_Ct, bp_Visited, bp_RecStack)){
59                  return true;
60              }
61          }
62
63          delete [] bp_Visited;
64          delete [] bp_RecStack;
65          return false;
66      }
```

main.cpp

```
68      private:bool fn_IsCyclic(int i_Ind, bool* bp_Visited,
69                               bool* bp_RecStack){
70
71          if(bp_Visited[i_Ind] == false){
72              bp_Visited[i_Ind] = true;
73              bp_RecStack[i_Ind] = true;
74
75              for(Vet* op_Tmp= cl2p_Ptr[i_Ind]; op_Tmp!= nullptr;
76                  op_Tmp= op_Tmp->clsp_Ptr){
77
78                  int i_Tmp = fn_GetIndex(op_Tmp->c_Name);
79                  if ( !bp_Visited[i_Tmp] &&
80                      fn_IsCyclic(i_Tmp, bp_Visited, bp_RecStack)
81                      ){
82                      return true;
83                  }
84                  else if (bp_RecStack[i_Tmp]){
85                      return true;
86                  }
87              }
88          }
89          bp_RecStack[i_Ind] = false;
90          return false;
91      }
```

圖論—循環檢測議題 (4)

- 循環檢測 (Cycle detection) (cont'd)

main.cpp

```
93      private: int fn_GetIndex(char c_Name){
94          int i_Ind = -1;
95          for(int i_Ct =0; i_Ct < i_Size; i_Ct++){
96                  if(cp_Name[i_Ct] == c_Name){
97                      i_Ind= i_Ct;
98                      break;
99                  }
100         }
101         return i_Ind;
102     }
103 };
```

main.cpp

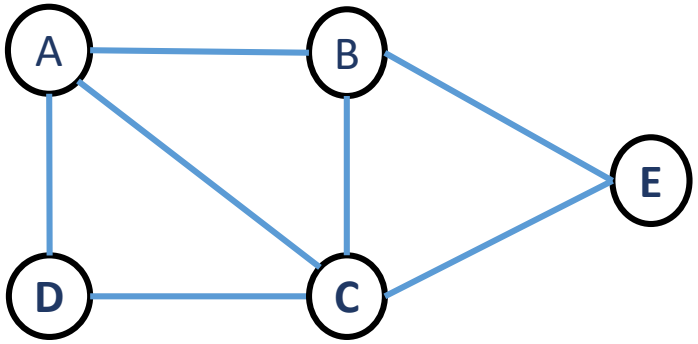
```
105 int main(){
106     char c_Arr[] = {'1', '2', '3', '4'};
107     int i_Size  = sizeof(c_Arr)/sizeof(char);
108     AdList o_List = AdList(c_Arr, i_Size);
109
110     o_List.fn_InsVet('1', '2');
111     o_List.fn_InsVet('2', '3');
112     o_List.fn_InsVet('3', '4');
113     o_List.fn_InsVet('3', '1');
114
115     std::cout<< (o_List.fn_isCyclic() == true ? "TRUE" : "FALSE");
116     return 0;
117 }
```

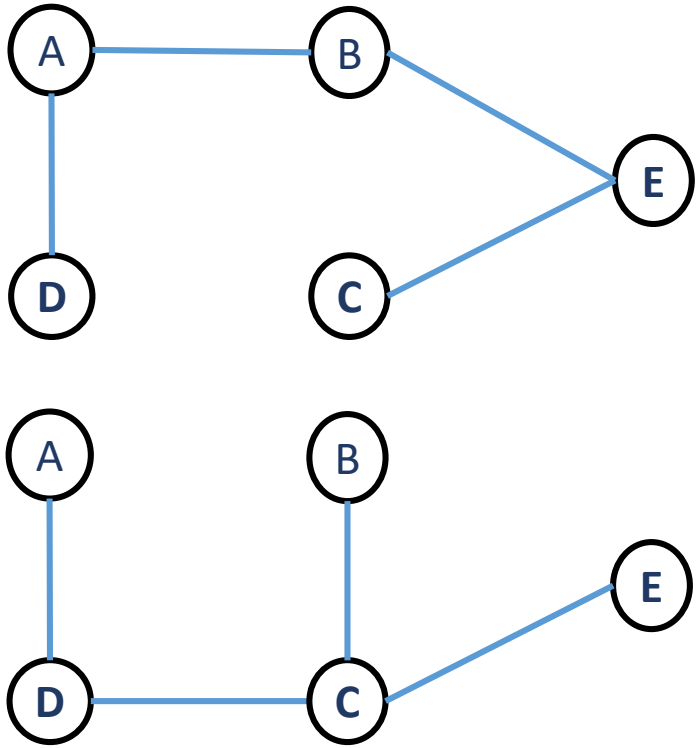
```
E:\CodeWorkShop\CodeBlock\ProjectDsAlg\GraphAdListCycleChecking
TRUE
Process returned 0 (0x0)   execution time : 0.061 s
Press any key to continue.
```

程式碼網址：https://github.com/altoliaw2/GraphAdListCycleChecking

圖論—最小生成樹議題 (1)

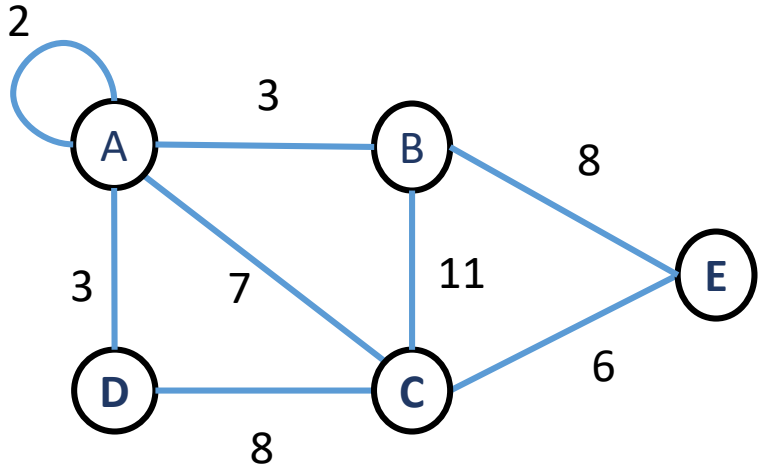
- 生成樹Spanning Tree
  - ✓從一張圖取出一棵樹
    - 有許多種可能
      - 左邊為原圖，右邊為從圖取出的樹





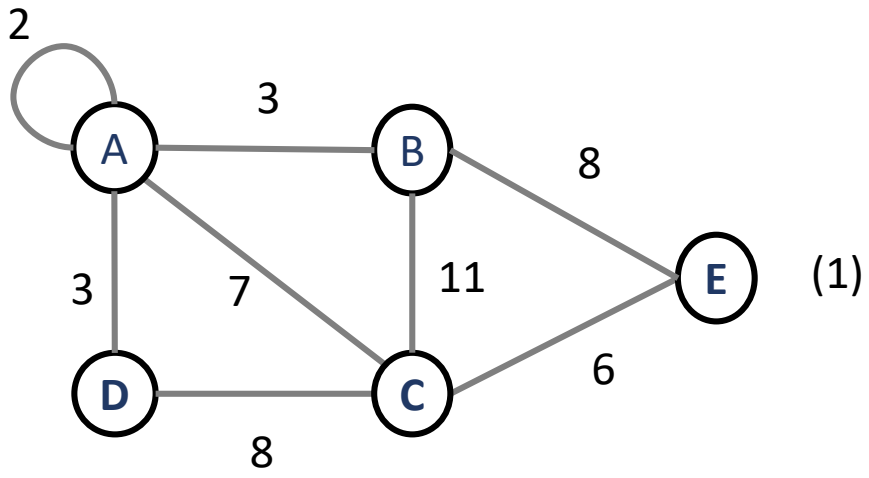
圖論—最小生成樹議題 (2)

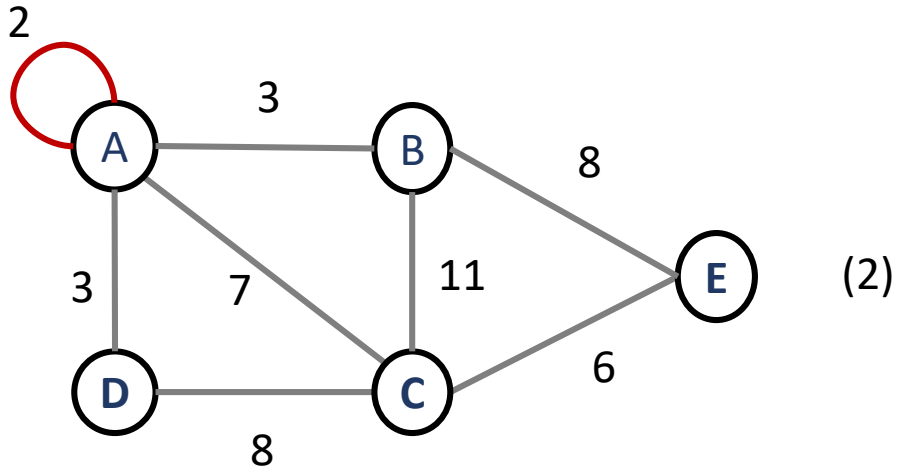
- 最小生成樹Minimum Spanning Tree
  - ✓一般來說在圖論中，除了描述關係外，也會呈現每個關係的權重(如左圖)
    - 假設左圖為公司網路連線所需耗費的電力，找出最小生成樹即代表，找出最少耗費並可連接所有點之狀樹狀架構，此架構將會耗費最少產能。
    - 因此找出公司網路穩定所需最少的電力
  - ✓一般來說，為求出經過所有點所需最小的權重



圖論—最小生成樹議題 (2)

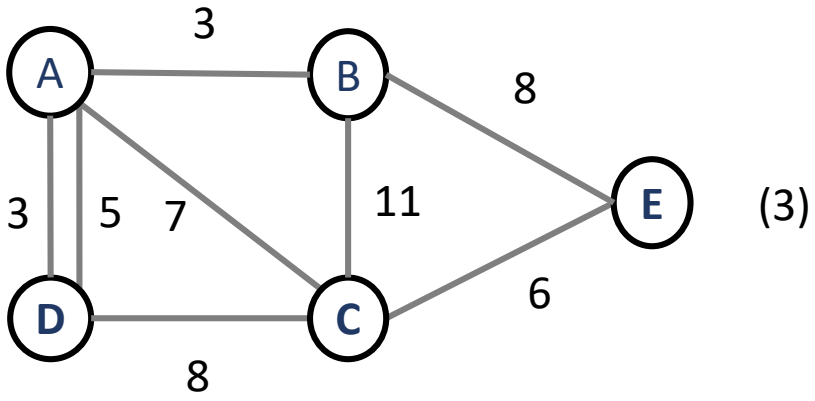
- 最小生成樹Minimum Spanning Tree (cont'd)
  - ✓想法(Kruskal's Algorithm)
    - 將所有權重由小到大排序
    - 依序加入，但每次加入後不可行成「環(cycle)」
      - 若形成環，則放棄該條加入



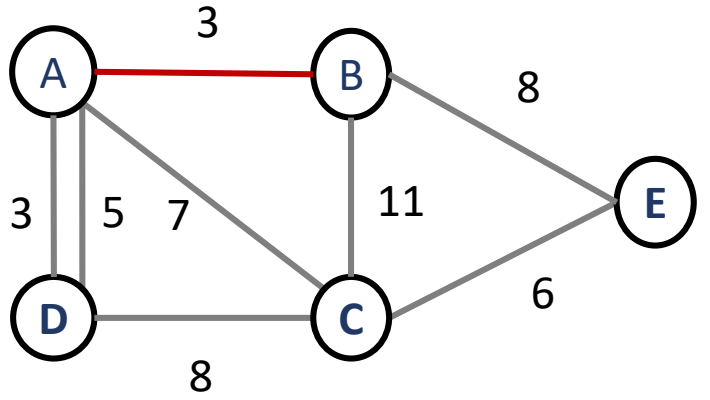


圖論—最小生成樹議題 (3)

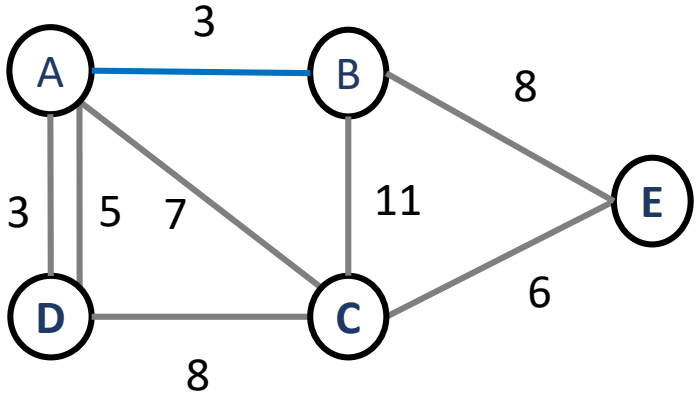
- 最小生成樹Minimum Spanning Tree (cont'd)



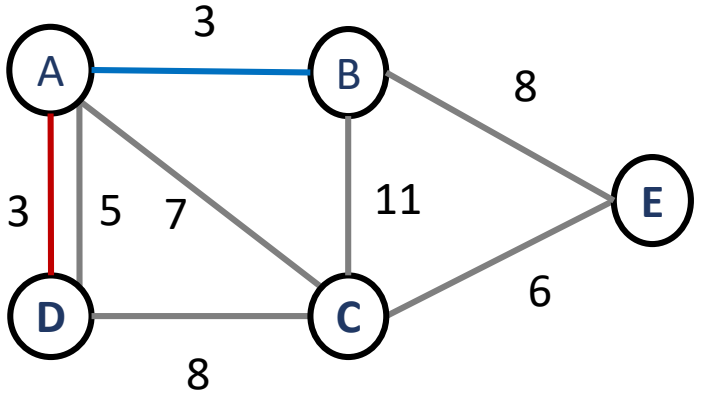
(3)



(4)



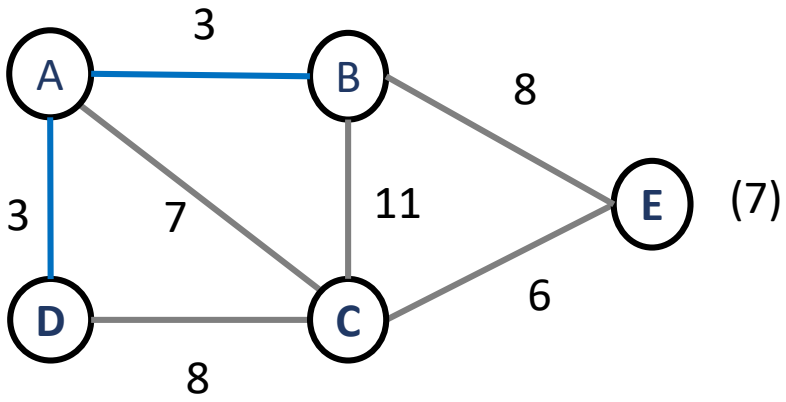
(5)



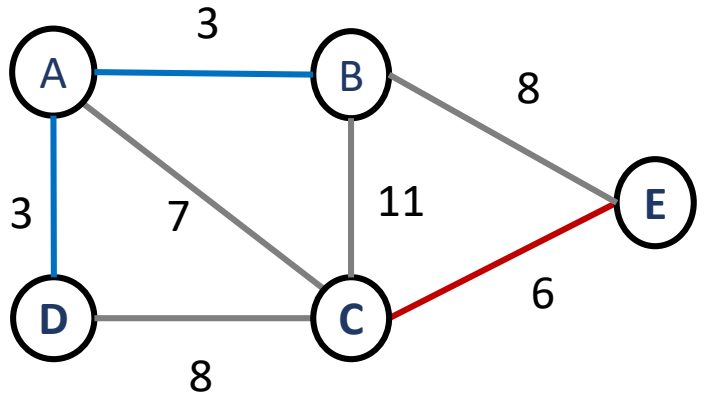
(6)

圖論—最小生成樹議題 (4)

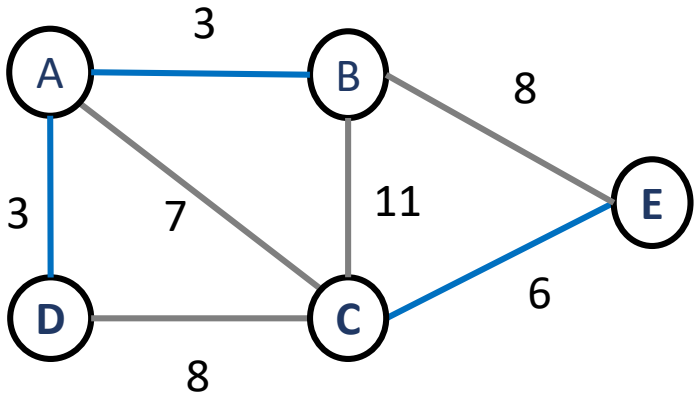
- 最小生成樹Minimum Spanning Tree (cont'd)



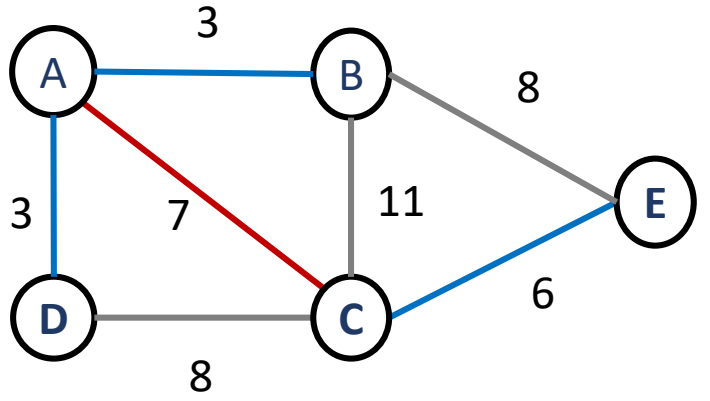
(7)



(8)



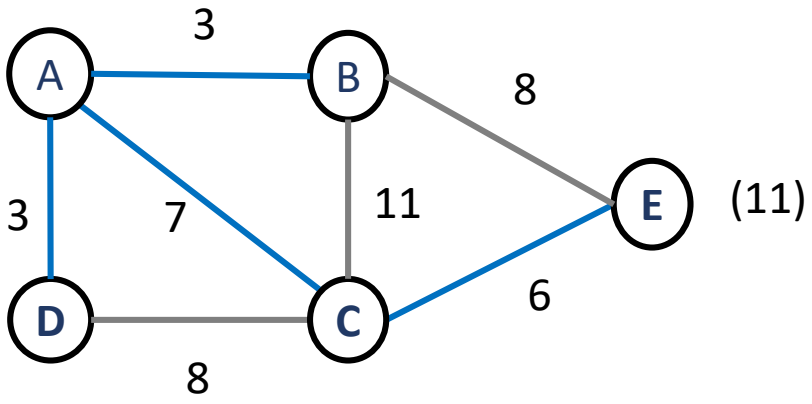
(9)



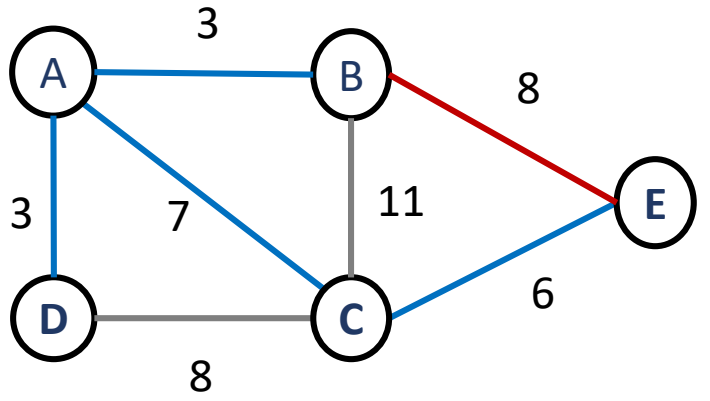
(10)

圖論—最小生成樹議題 (5)

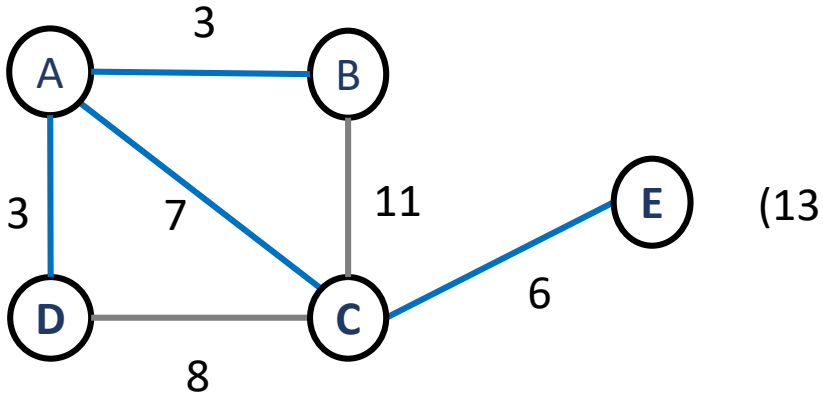
- 最小生成樹Minimum Spanning Tree (cont'd)



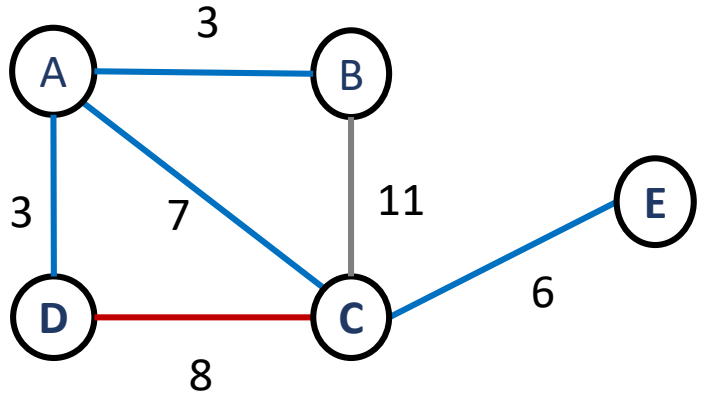
(11)



(12)



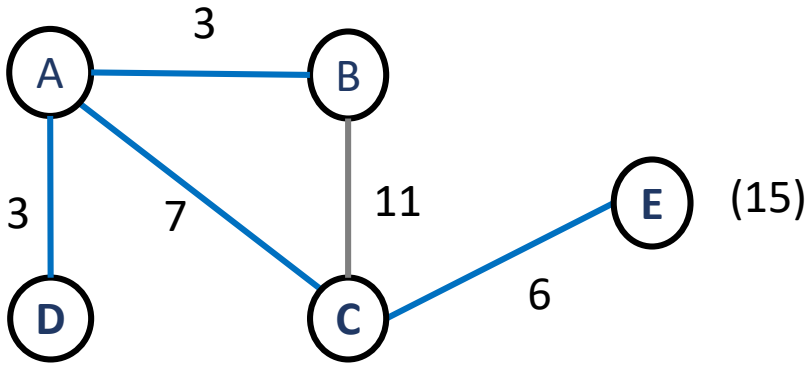
(13)



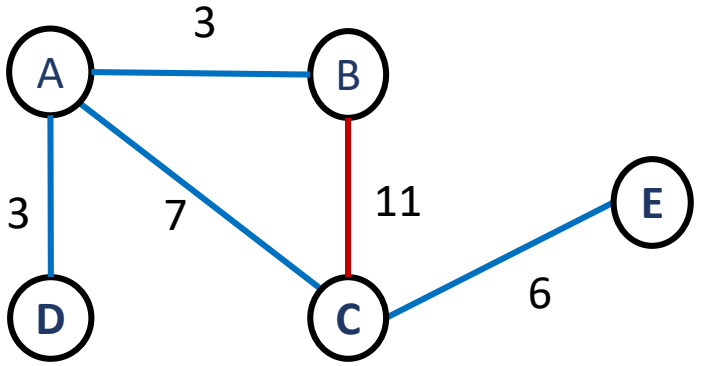
(14)

圖論—最小生成樹議題 (6)

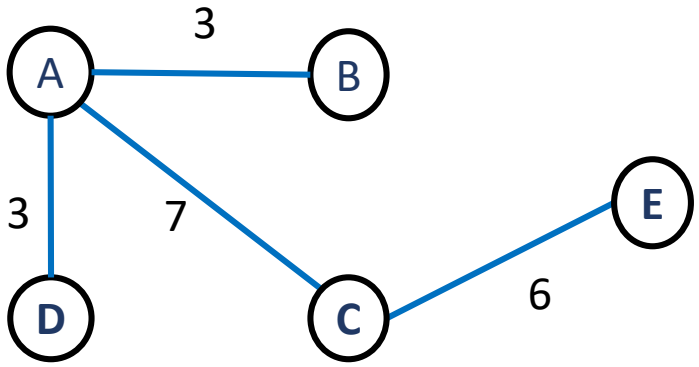
- 最小生成樹Minimum Spanning Tree (cont'd)



(15)



(16)



(17)

圖論—最小生成樹議題 (7)

- 最小生成樹Minimum Spanning Tree (cont'd)

main.cpp

```
#include <algorithm>
#include <iostream>
#include <limits>

struct AdList{
    public: int** cl2p_Ptr;
    public: char* cp_Name;
    public: int i_Size;

    private: int* ip_Parent;

    public: AdList(char* cp_Name, int i_Size){
        cl2p_Ptr= new int*[i_Size];
        for(int i_Ct=0; i_Ct< i_Size; i_Ct++){
            cl2p_Ptr[i_Ct] = new int[i_Size];
            std::fill(cl2p_Ptr[i_Ct], cl2p_Ptr[i_Ct]+ i_Size,
                      std::numeric_limits<int>::max());
        }
        this->cp_Name = cp_Name;
        this->i_Size = i_Size;

        ip_Parent = new int[i_Size];
    }
```

main.cpp

```
    public: bool fn_InsVet(char c_NameA, char c_NameB,
                           int i_Weight){
        int i_RInd = -1;
        int i_CInd = -1;
        for(int i_Ct= 0; i_Ct < i_Size; i_Ct++){
            if(c_NameA == cp_Name[i_Ct]){
                i_RInd = i_Ct;
                break;
            }
        }
        for(int i_Ct= 0; i_Ct < i_Size; i_Ct++){
            if(c_NameB == cp_Name[i_Ct]){
                i_CInd = i_Ct;
                break;
            }
        }
        if(i_RInd == -1 || i_CInd == -1){
            return false;
        }
        cl2p_Ptr[i_RInd][i_CInd] = i_Weight;
        cl2p_Ptr[i_CInd][i_RInd] = i_Weight;
    }
```

圖論—最小生成樹議題 (8)

- 最小生成樹Minimum Spanning Tree (cont'd)

main.cpp

```
48      public: ~AdList(){
49          for(int i_Ct =0; i_Ct< i_Size; i_Ct++){
50              delete [] cl2p_Ptr[i_Ct];
51          }
52          delete [] cl2p_Ptr;
53          delete [] ip_Parent;
54      }
```

main.cpp

```
56      public: void fn_GetMSTByKruskal(int** i2p_Cost){
57          int i_MinCost = 0;
58
59          for (int i_Ct = 0; i_Ct < i_Size; i_Ct++){
60              ip_Parent[i_Ct] = i_Ct;
61          }
62
63          for (int i_EdgeCount = 0; i_EdgeCount < i_Size - 1;
64                  i_EdgeCount++) {
65              int i_MinC = std::numeric_limits<int>::max();
66              int i_IInd = -1;
67              int i_JInd = -1;
68              for (int i_TmpIInd = 0; i_TmpIInd < i_Size; i_TmpIInd++) {
69                  for (int i_TmpJInd = 0; i_TmpJInd < i_Size; i_TmpJInd++) {
70                      if (fn_FindInd(i_TmpIInd) != fn_FindInd(i_TmpJInd) &&
71                          i2p_Cost[i_TmpIInd][i_TmpJInd] < i_MinC) {
72
73                              i_MinC = i2p_Cost[i_TmpIInd][i_TmpJInd];
74                              i_IInd = i_TmpIInd;
75                              i_JInd = i_TmpJInd;
76                      }
77                  }
78              }
```

圖論—最小生成樹議題 (9)

- 最小生成樹Minimum Spanning Tree (cont'd)

main.cpp

```
80              fn_Union(i_IInd, i_JInd);
81              for(int i_Ct = 0; i_Ct < i_Size; i_Ct++){
82                  std::cout <<ip_Parent[i_Ct] << "\t";
83              }
84              std::cout<< "\n\n";
85              std::cout<<"Edge "<< i_EdgeCount << ":("<<
86                      cp_Name[i_IInd]<< ", "<< cp_Name[i_JInd] << ") cost:"<<
87                      i_MinC<<"\n";
88              i_MinCost += i_MinC;
89          }
90          std::cout<< "\nMinimum cost= "<< i_MinCost<< "\n";
91      }
92
93
94      private: int fn_FindInd(int i_Ind){
95          for(;ip_Parent[i_Ind] != i_Ind;){
96              i_Ind = ip_Parent[i_Ind];
97          }
98          return i_Ind;
99      }
```

main.cpp

```
101     private: void fn_Union(int i_TmpIInd,
102                            int i_TmpJInd){
103         int i_IInd = fn_FindInd(i_TmpIInd);
104         int i_JInd = fn_FindInd(i_TmpJInd);
105         ip_Parent[i_IInd] = i_JInd;
106     }
107 };
```

main.cpp

```
109 int main(){
110     char c_Arr[] = {'A', 'B', 'C', 'D', 'E'};
111     int i_Size  = sizeof(c_Arr)/sizeof(char);
112     AdList o_List = AdList(c_Arr, i_Size);
113     o_List.fn_InsVet('A', 'A', 2);
114     o_List.fn_InsVet('A', 'B', 3);
115     o_List.fn_InsVet('A', 'C', 7);
116     o_List.fn_InsVet('A', 'D', 3);
117
118     o_List.fn_InsVet('B', 'C', 11);
119     o_List.fn_InsVet('B', 'E', 8);
120
121     o_List.fn_InsVet('C', 'D', 8);
122     o_List.fn_InsVet('C', 'E', 6);
```

圖論—最小生成樹議題 (10)

- 最小生成樹Minimum Spanning Tree (cont'd)

main.cpp

```
124     for(int i_Ct= 0; i_Ct < i_Size; i_Ct++){
125         for(int i_Ct2= 0; i_Ct2 < i_Size; i_Ct2++){
126             std::cout<< o_List.cl2p_Ptr[i_Ct][i_Ct2] << " ";
127         }
128         std::cout<< "\n";
129     }
130
131     std::cout<< "\n";
132
133     o_List.fn_GetMSTByKruskal(o_List.cl2p_Ptr);
134
135     return 0;
136 }
```