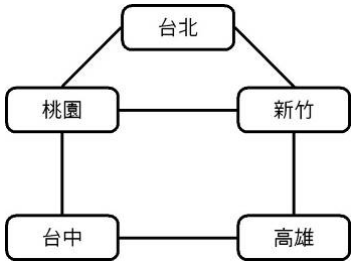
資料結構的圖形結構(Graphs)

資訊科技系
林偉川

圖形的基本觀念

- 在日常生活中，我們常常將複雜觀念或問題使用圖形來表達，例如：在進行系統分析、電路分析、電話佈線和企劃分析等。因為圖形化可以讓人更容易了解，所以「圖形」（Graph）是資料結構一種十分重要的結構。例如：城市之間的公路圖，如下圖所示：



2

圖形的基本定義

- 圖形是由有限的點和邊線集合所組成，其定義如下所示：

  定義 8.1：圖形G是由V和E兩個集合組成，寫成：

  G = (V, E)

  V：點(Vertices)組成的有限非空集合。

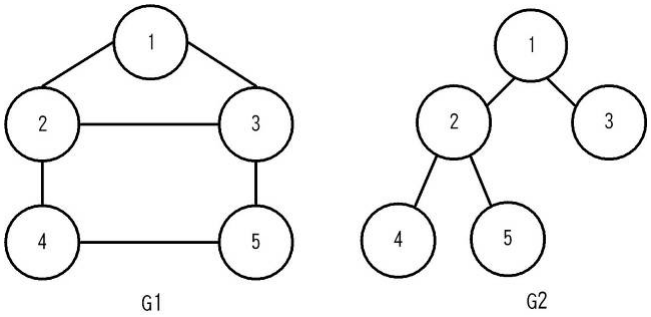
  E：邊線(Edges)組成的有限集合，這是成對的點集合。

- 圖形通常使用圓圈代表點，點之間的連線是邊線。

3

圖形的基本圖例

- 圖形通常使用圓圈代表點，點之間的連線是邊線。例如：上述公路圖繪成的圖形G1和另一個樹狀圖形G2，如下圖所示：



4

圖形的基本表示法

- 圖形一共擁有5個點V1、V2、...、V5，V(G1)是圖形G1的點集合，V(G2)是圖形G2，如下所示：

V(G1) = { V1, V2, V3, V4, V5 }

V(G2) = { V1, V2, V3, V4, V5 }

- 圖形G1點和點之間的邊線有6條，G2有4條，E(G1)是圖形G1的邊線集合，E(G2)是圖形G2，如下所示：

E(G1) = { (V1,V2),(V1,V3),(V2,V3),(V2,V4),(V3,V5),(V4,V5) }

E(G2) = { (V1,V2),(V1,V3),(V2,V4),(V2,V5) }

- 上述邊線是使用括號括起的兩個點，例如：(V1,V2)表示從點V1到V2存在一條邊線。

5

圖形的基本圖形種類

- 圖形是由點和邊線所組成，依邊線集合E(G)中點是否擁有順序性，可以分為兩種，如下所示：
  - **無方向性圖形（Undirected Graph）**：圖形的邊線沒有標示方向的箭頭，邊線只代表點間是相連的。例如：圖形G1和G2是無方向性圖形，所以(V1,V2)和(V2,V1)代表同一條邊線。
  - **方向性圖形（Directed Graph）**：在圖形的邊線加上箭號標示點間的順序性。

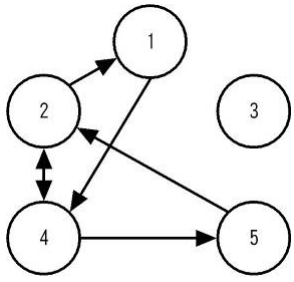
6

方向性圖形的基本觀念

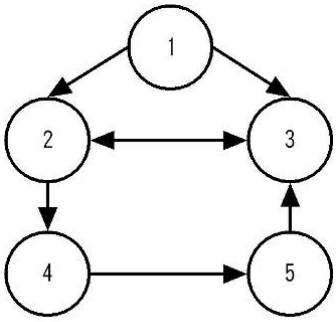
- 圖形G3是方向性圖形，G3圖形的點和邊線集合V(G3)、E(G3)，如下所示：

V(G3)={ V1, V2, V3, V4, V5 }

E(G3)={ <V1,V2>,<V1,V3>,<V2,V3>,<V3,V2>,<V2,V4>,<V4,V5>,<V5,V3> }



G7

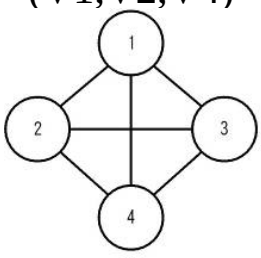


G3

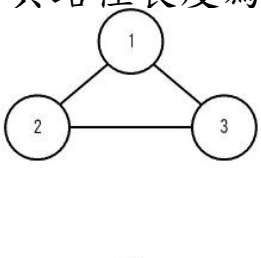
7

圖形術語

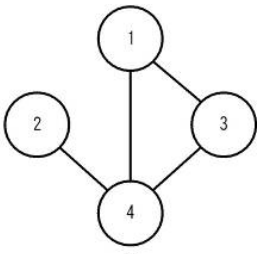
- 路徑：為連接兩點間的邊。例如：圖形G3中V1經由<V1,V2>及<V2,V4>的邊到達V4，則其路徑為(V1,V2,V4)
- 路徑長度：兩點間其路徑上邊的個數。例如：圖形G3中V1到V4的路徑為(V1,V2,V4)，其路徑長度為2



G4



G5



G6

8

圖形術語

- 簡單路徑（Simple Directed Path）：除了第1個和最後1個點可以相同外，其它位在路徑上的點都不相同。例如：圖形G7的路徑<V5,V2>、<V2,V1>，寫成：5,2,1是簡單路徑，路徑5,2,1,4,2,1就不是簡單路徑。
- 循環（Cycle）：屬於簡單路徑的一種特例，也就是第1個和最後1個點是同一個點的路徑。例如：圖形G7的路徑5,2,4,5，第1個和最後1個點都是5。

9

圖形術語

- 相連圖形（Connected Graph）：圖形內任何兩個點都有路徑相連結。例如：圖形G1、G2、G3、G4、G5和G6是相連圖形。
- 不相連圖形（Disconnected Graph）：圖形內至少有兩個點間是沒有路徑相連的。例如：圖形G7的點3。
- 完整圖形（Complete Graph）：一個n點的無方向性圖形擁有n(n-1)/2條邊線，例如：4點的完整圖形G4擁有6條邊線
- 子圖（Subgraph）：圖形G的子圖H，是指G的點包含或等於H的點，G的邊線包含或等於H的邊線，例如：G5和G6是G4的子圖

10

圖形術語

- 入支度(In-degree)：有向圖中連向某點箭頭的邊線數。例如：圖形G7點1的入支度是1(2)，點2的入支度是2 (4,5)。
- 出支度(Out-degree)：與內分支度相反，指某點連向其他點的邊線數。例如：圖形G3點2出支度是2(3,4)。
- 分支度(Degree)：若為無向圖，則分支度表示附著在點的邊數，若為有向圖，則分支度為入支度與出支度的總和。例如：圖形G3點2分支度是4。
- 鄰接(Adjacent)：如果兩個點間擁有一條邊線連結，則這兩個點稱為鄰接。

11

圖形的表示法

- 圖形結構可以使用多種方法來實作，常用的方法有二種，如下所示：
  - 鄰接矩陣表示法（Adjacency Matrix）。
  - 鄰接串列表示法（Adjacency Lists）。

12

鄰接矩陣表示法

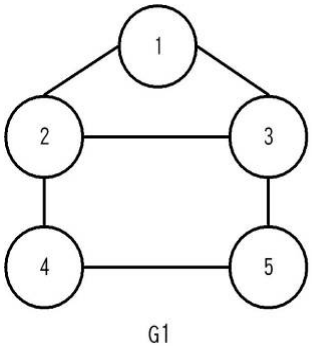
- 圖形G = (V, E)是一個包含n個點的圖形，可以使用一個n * n矩陣來儲存圖形，在C語言是宣告一個n * n的二維陣列，索引值代表點，陣列元素值0表示沒有邊線，1表示有邊線。陣列A中若圖之點Vi與點Vj相鄰，即存在邊線(Vi，Vj) ，則A[i][j]=1，否則A[i][j]=0。

$$A[i][j]=\begin{cases}1, 若(V_i, V_j)\in E(G)\\ 0, 若(V_i, V_j)\notin E(G)\end{cases}$$

13

鄰接矩陣表示法-圖例

- 無方向圖形G1的鄰接矩陣表示法使用5*5的二維陣列儲存矩陣，如果點V1和V2鄰接，陣列元素(V1,V2)和(V2,V1)的值是1，表示圖形包含點V1到V2和點V2到V1的路徑，如下圖所示：



G1

	第1欄	第2欄	第3欄	第4欄	第5欄
第1列	0	1	1	0	0
第2列	1	0	1	1	0
第3列	1	1	0	0	1
第4列	0	1	0	0	1
第5列	0	0	1	1	0

14

鄰接矩陣表示法

- 由鄰接矩陣很容易判斷是否有一邊線連接兩點，對無向圖而言，任一點i的列和為分支度。對有向圖而言，為出支度與各行和為入支度，分支度為列與行的和。

15

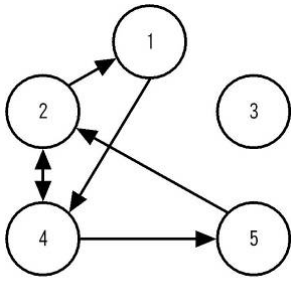
鄰接矩陣表示法

- 圖形中不能有自己循環，即點Vi至Vi不可能有邊的存在，所以鄰接矩陣之A[i][i]=0，即對角線元素皆為0
- 若為無向圖，邊(Vi,Vj)存在表示A[i][j]=A[j][i]=1，亦即無向圖之鄰接矩陣為一對稱矩陣，故只需保存上三角或下三角部份即可，大約可節省一半以上的空間。對於有向圖則不一定是對稱的

16

鄰接矩陣表示法-標頭檔

```
02: #define MAX_VERTICES   6
03: int graph[MAX_VERTICES][MAX_VERTICES];
04: /* 抽象資料型態的操作函數宣告 */
05: extern void createGraph(int len, int *edge);
06: extern void printGraph();
```



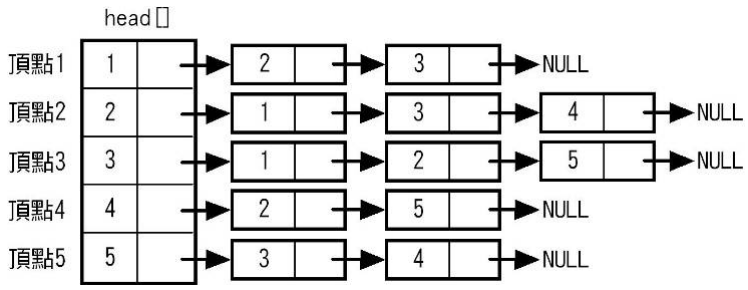
G7

	1	2	3	4	5
1	0	0	0	1	0
2	1	0	0	1	0
3	0	0	0	0	0
4	0	1	0	0	1
5	0	1	0	0	0

17

鄰接串列表示法

- 鄰接串列表示法的圖形是使用單向串列來鏈結每個點的鄰接點，使用一個點的結構陣列指標指向各點的鄰接點串列。例如：無方向圖形G1的鄰接串列表示法，如下圖所示：



18

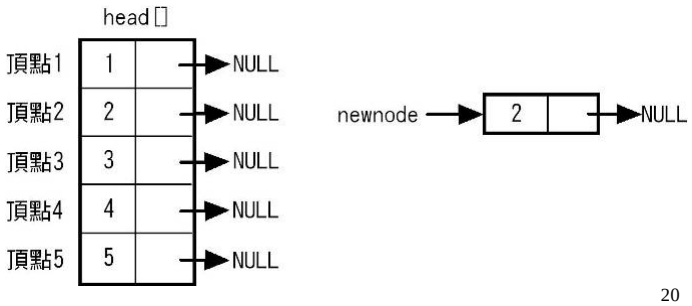
鄰接串列表示法-標頭檔

```
02: #define MAX_VERTICES   10 /* 圖形的最大點數 */
03: struct Vertex {              /* 圖形點結構宣告 */
04:    int data;                 /* 點資料 */
05:    struct Vertex *next;    /* 指下一個點的指標 */
06: };
07: typedef struct Vertex *Graph;   /* 圖形的新型態 */
08: struct Vertex head[MAX_VERTICES];
09: /* 抽象資料型態的操作函數宣告 */
10: extern void createGraph(int len, int *edge);
11: extern void printGraph();
12: extern void dfs(int vertex);
13: extern void bfs(int vertex);
```

19

鄰接串列表示法來建立圖形

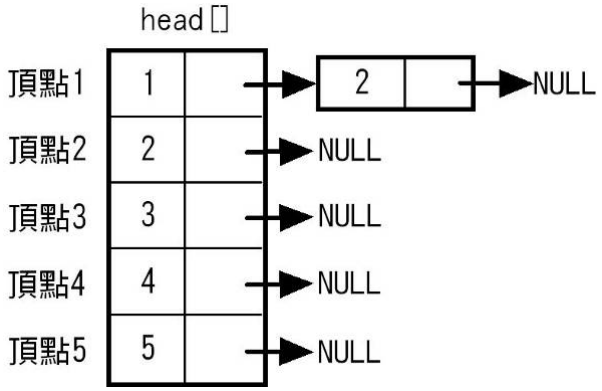
- 函數createGraph()首先使用for迴圈初始點結構陣列head[]，next指標都是指向NULL，讀入的第一條邊線是(1, 2)，從點1連到點2，所以建立結尾點2的節點指標newnode，如下圖所示：



20

鄰接串列表示法來建立圖形

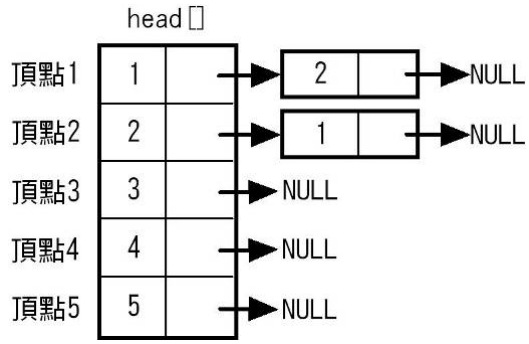
- 然後將節點插入結構陣列head[]索引1（即點1）的串列最後，如下圖所示：



21

鄰接串列表示法來建立圖形

- 繼續讀入的邊線是(2, 1)，從點2連到點1，插入的是結構陣列head[]索引值2的串列最後，如下圖所示：



等到讀完所有邊線，就可以完成圖形G1的鄰接串列表示法。

22

鄰接串列表示法來顯示圖形

函數printGraph()：顯示圖形

- 函數printGraph()使用迴圈走訪head[]陣列顯示鄰接串列表示法，在取得每一個點串列的串列指標後，使用while迴圈走訪串列的每一個節點，如下所示：

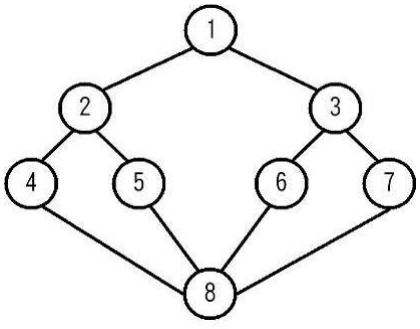
```
while ( ptr != NULL ) {
  printf("V%d ", ptr->data);
  ptr = ptr->next;
}
```

23

圖形的走訪

- 圖形結構與之前的二元樹和鏈結串列一樣，都擁有特定的走訪方式。例如：一個無方向圖形G9，如下圖所示：

	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	0	1	1	0	0	0
3	1	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0	1
5	0	1	0	0	0	0	0	1
6	0	0	1	0	0	0	0	1
7	0	0	1	0	0	0	0	1
8	0	0	0	1	1	0	0	0

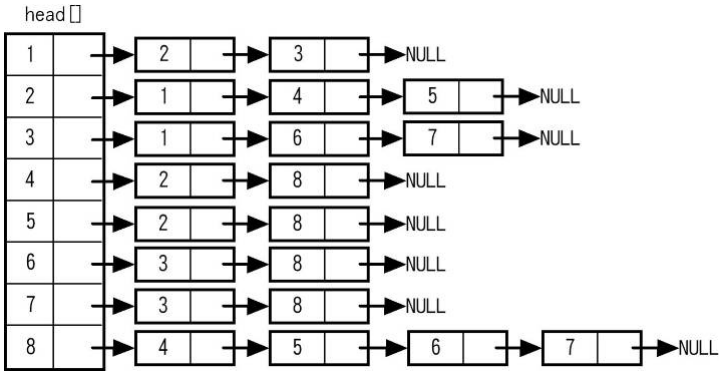


G9

24

圖形的走訪

- 圖形G9的鄰接串列表示法，如下圖所示：



25

圖形的走訪-種類

- 圖形G9的走訪可以分為偏向直的深度或橫的寬度兩種搜尋法，如下所示：
  - **深度優先搜尋法（Depth-first Search, DFS）**：當在鄰接串列表示法走訪某一點後，優先找尋點在結構陣列中的鄰接點。
  - **寬度優先搜尋法（Breadth-first Search, BFS）**：當在鄰接串列表示法走訪某一點後，優先找尋點串列的所有鄰接點。

26

深度優先搜尋法DFS

- 圖形G9的深度優先搜尋法是從點1開始以深度優先方式來走訪圖形，首先走訪點1，找尋結構陣列索引值1的鄰接點，結果找到未走訪的點2，點2從深度方向往下走訪，即走訪結構陣列索引值2的鄰接點，找到未走訪過的點4，繼續再往下搜尋結構陣列索引值4的鄰接點，可以找到未走訪過的點8。
- 從點8走訪結構陣列索引值8的鄰接點，找到未走訪的點5，因為點5的鄰接點2和8已經走訪過，所以再回到點8，繼續找到點6，接著從結構陣列索引值6找到鄰接點3，最後找到點7，可以得圖形走訪的點順序，如下所示：1→2→4→8→5→6→3→7

27

深度優先搜尋法DFS-演算法

- 深度優先搜尋法的遞迴函數dfs(V)的操作步驟，如下所示：
  - Step 1：設定點V已走訪過，1為走訪過。

    visited[vertex] = 1;
  - Step 2：如果點V存在有鄰接點W未被走訪過，遞迴呼叫函數dfs(W)。

```
while ( ptr != NULL ) {
  if ( visited[ptr->data] == 0 )  dfs(ptr->data);
  ptr = ptr->next;
}
```

28

寬度優先搜尋法BFS

- 圖形G9的寬度優先搜尋法與深度優先搜尋法走訪圖形的差別在於走訪點1後，接著是走訪點1的所有鄰接點，即點2和3，然後才從點2和3開始走訪所有鄰接且未走訪過的點，點2走訪點4和5，點3走訪點6和7，最後走訪點8。
- 整個寬度優先搜尋法走訪圖形的順序。如下所示：1→2→3→4→5→6→7→8

29

寬度優先搜尋法BFS演算法

- 寬度優先搜尋法的函數bfs(V)的操作步驟，如下所示：
  - Step 1：設定點V已經走訪過，1為走訪過。
    visited[vertex] = 1;
  - Step 2：將點V存入佇列。
    enqueue(vertex);

30

寬度優先搜尋法BFS演算法

– Step 3：執行迴圈直到佇列空了為止，如下：

while (!isQueueEmpty()) {

- (1) 取出佇列的點V。

```
vertice = dequeue();
ptr = head[vertex].next;
```

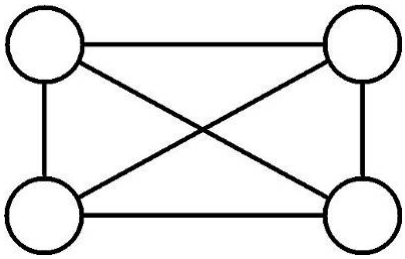
- (2) 將點V所有鄰接且未走訪的點W存入佇列，且設定已走訪。

```
while ( ptr != NULL ) {
  if ( visited[ptr->data]==0 ) {
    enqueue(ptr->data);
    visited[ptr->data] = 1;
    printf("[V%d] ", ptr->data);
  }
  ptr = ptr->next;
 }
}
```

31

擴張樹

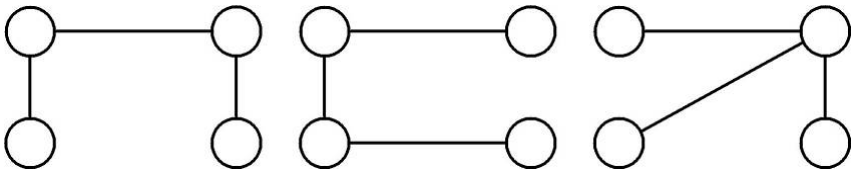
- 擴張樹(Spanning Trees)是將無方向性圖形的所有點使用邊線連接起來，但邊線並不會形成迴圈，擴張樹的邊線數將比點少1，因為再多一條邊線，圖形就會形成迴圈。例如：



32

擴張樹範例

- 一個擁有四個點六條邊線的圖形，依擴張樹的定義，可以得三棵不同的擴張樹，如下圖所示：



33

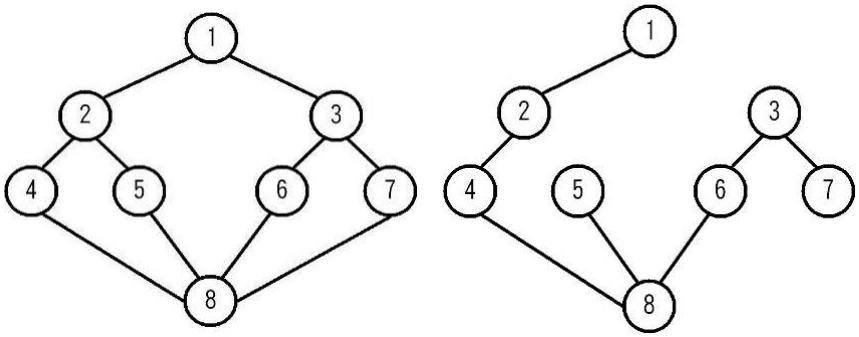
圖形走訪建立擴張樹

- 擴張樹可以使用走訪的搜尋法來建立，因為只需將圖形走訪過的點順序，使用邊線一一連接起來，就可以建立成擴張樹，依照搜尋法的不同，分成二種擴張樹，如下所示：
  - 深度優先擴張樹（DFS Spanning Trees）。
  - 寬度優先擴張樹（BFS Spanning Trees）。

34

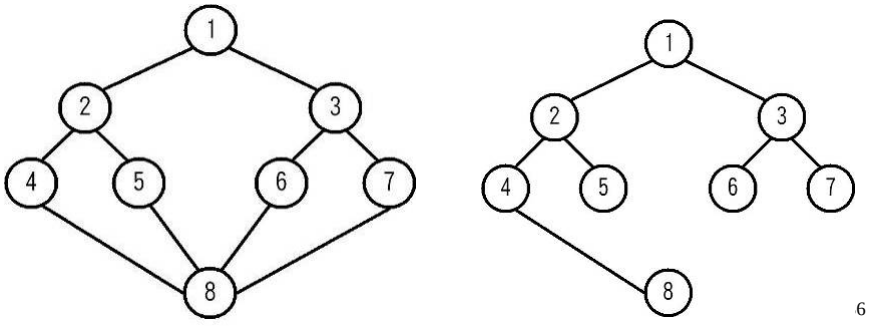
深度優先擴張樹

- 深度優先搜尋走訪圖形G9的點順序，如下：
  1,2,4,8,5,6,3,7
- 現在只需將走訪經過的邊線保留下來，就可以建立深度優先擴張樹，如下圖所示：



寬度優先擴張樹

- 寬度優先搜尋走訪圖形G9的點順序，如下：
  1,2,3,4,5,6,7,8
- 保留點1走訪到點2,3的邊線，點2走訪到4,5，點3走訪點6,7，點4走訪到點8，可以建立走訪的寬度優先擴張樹，如下圖所示：



6

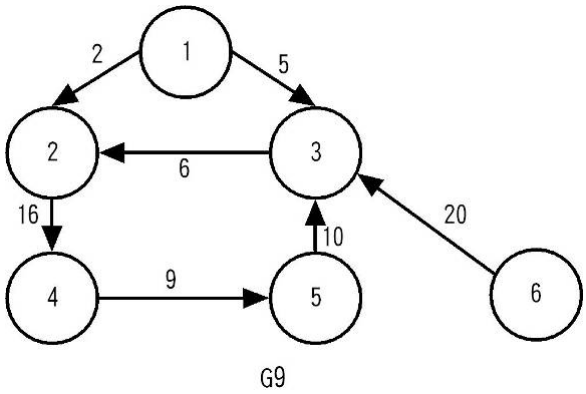
加權圖形表示法

- 圖形在解決問題時通常需要替邊線加上一個數值，這個數值稱為「權值」（Weights），常見的權值有：時間、成本或長度，擁有權值的圖形稱為加權圖形，一樣可以分別使用鄰接矩陣和鄰接串列來表示。

37

加權圖形表示法

- 例如：一個方向性圖形G9，如下圖所示：



G9

38

加權圖形鄰接矩列表示法

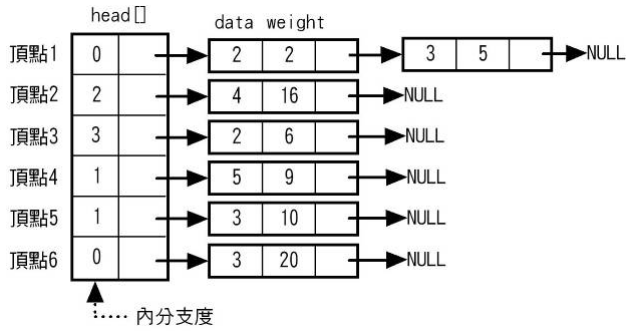
- 以鄰接矩陣的加權圖形來說，只需將原來儲存1和0的陣列元素換成各點間的權值。如果點間無邊線連接，就使用無窮大∞表示。所以，圖形G9的加權鄰接矩陣表示法，如下圖所示：

	第1欄	第2欄	第3欄	第4欄	第5欄	第6欄
第1列	0	2	5	∞	∞	∞
第2列	∞	0	∞	16	∞	∞
第3列	∞	6	0	∞	∞	∞
第4列	∞	∞	∞	0	9	∞
第5列	∞	∞	10	∞	0	∞
第6列	∞	∞	20	∞	∞	0

39

加權圖形鄰接串列表示法

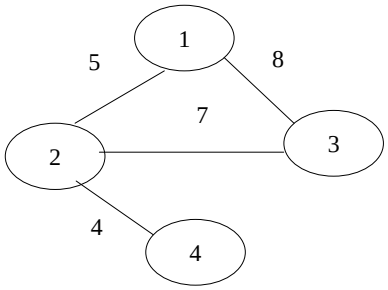
- 鄰接串列表示法的加權圖形只是在點結構新增成員變數weight儲存權值，圖形G9的加權鄰接串列表示法，如下圖所示：



40

加權圖表示法

- 圖形每一邊都附加一個數值，此數值為此邊之加權，則稱此圖為加權圖



	1	2	3	4
1	0	5	8	∞
2	5	0	7	4
3	8	7	0	∞
4	∞	4	∞	0

41

最低成本擴張樹

- 從加權圖形建立的擴張樹因為邊線擁有權值，所以可以計算邊線的權值和，換句話說，圖形建立的擴張樹會因連接的邊線權值不同，而建立出不同成本的擴張樹，因為各種建立方法會產生不同成本，所以如何找出「最低成本擴張樹」（Minimum-cost Spanning Trees）就成為一個重要的問題。

42

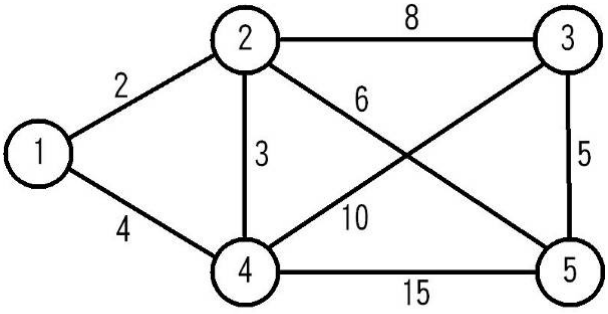
最低成本擴張樹

- 樹狀結構是圖形的特例，樹不允許循環，且n個點的樹狀結構，其邊數為n-1
- 一個圖形的擴張樹為其本身子圖，且點皆相同
- 如果一加權圖，其邊線上的加權和稱為成本，則擴 張樹中其總加權值最低者，是其最低成本擴張樹

43

最低成本擴張樹圖例

- 例如：一個擁有權值的無方向性加權圖形G10找出最低成本擴張樹，如下圖所示：



G10

44

最低成本擴張樹-克魯斯卡演算法

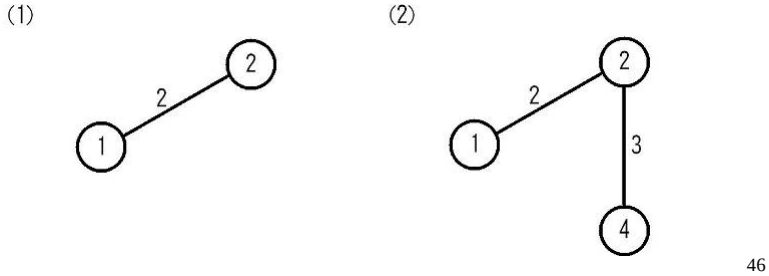
- 克魯斯卡（Kruskal）提出建立最低成本擴張樹的演算法，首先將各邊線依權值從小到大排列，如下表所示：

邊線	權值
(1, 2)	2
(2, 4)	3
(1, 4)	4
(3, 5)	5
(2, 5)	6
(2, 3)	8
(3, 4)	10
(4, 5)	15

45

最低成本擴張樹 -克魯斯卡演算法

- 接著從權值最低的一條邊線開始建立最低成本擴張樹，其步驟如下所示：
- Step 1：選擇第一條最低權值的邊線(1, 2)加入擴張樹，如下圖所示：

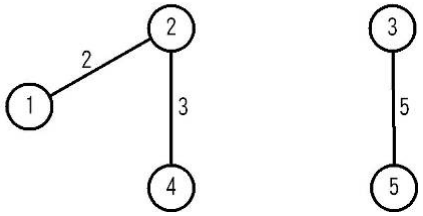


46

最低成本擴張樹 -克魯斯卡演算法

- Step 2：選擇第二低權值的邊線(2, 4)加入擴張樹，只需加入的邊線不會形成迴圈，以此例邊線(2, 4)並不會形成迴圈。
- Step 3：如果加入第三低權值的邊線(1, 4)，因為會形成迴圈，所以選擇下一條不會形成迴圈的低權值邊線(3, 5)，如下圖所示：

(3)

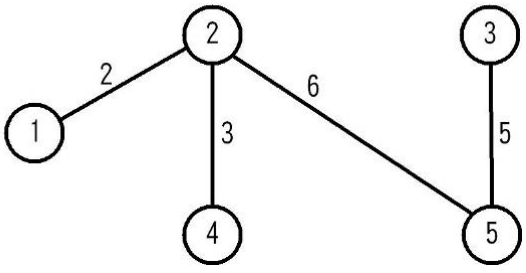


47

最低成本擴張樹 -克魯斯卡演算法

- Step 4：邊線(2, 5)也不會形成迴圈，將邊線(2, 5)加入擴張樹，現在所有點都已經連接，建立的就是一棵最低成本擴張樹，如下圖所示：

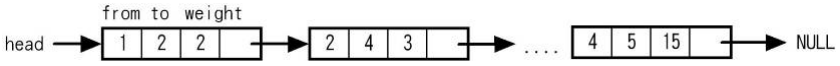
(4)



48

最低成本擴張樹-實作

- 最低成本擴張樹的實作是以權值從小到大使用單向鏈結串列儲存圖形的邊線點和權值，如下圖所示：



最低成本擴張樹-標頭檔

```
02: #define MAX_VERTICES   6      /* 最大點數加一 */
03: struct Edge {                 /* 圖形邊線結構宣告 */
04:    int from;                  /* 開始點 */
05:    int to;                    /* 終點點 */
06:    int weight;                /* 權值 */
07:    struct Edge *next;         /* 指下一條邊線 */
08: };
09: typedef struct Edge *EdgeList;  /* 邊線的結構新型態 */
10: EdgeList first = NULL;          /* 邊線串列開始指標 */
11: int vertex[MAX_VERTICES];  /* 檢查迴圈的點陣列 */
12: /* 抽象資料型態的操作函數宣告 */
13: extern void createEdge(int len, int *edge);
14: extern void minSpanTree();
15: extern void addSet(int from, int to);
16: extern int isSameSet(int from, int to);
```

最低成本擴張樹-演算法

- 函數minSpanTree()使用邊線的單向鏈結串列找出最低成本擴張樹，其操作步驟如下所示：
  - Step 1：走訪邊線的單向串列直到串列的結尾，如下：
    - (1) 如果邊線不會形成迴圈且成本最小，就將邊線加入擴張樹。

      ```
      if ( !isSameSet(ptr->from,ptr->to) )
        addSet(ptr->from,ptr->to);
      ```

    - (2) 移動指標到串列的下一條邊線節點。

      ```
      ptr = ptr->next;
      ```

51

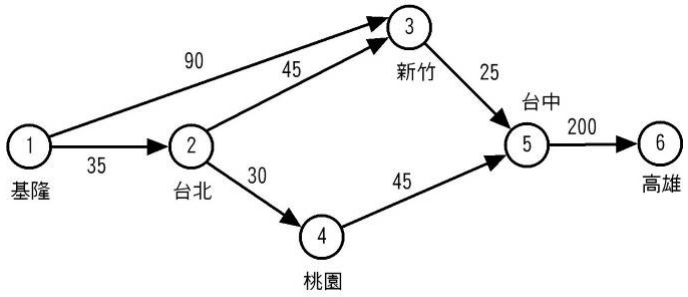
圖形的應用 - 最短路徑

- 最短路徑問題（Shortest Paths Problems）是指一個加權圖形G = (V, E)，邊線的權值不是負值且擁有一個點為來源（Source），問題是找出來源點到其它點的各條路徑中，各邊線權值和為最低的路徑，也就是最短路徑。最短路徑求法的常用方法有二種，如下所示：
  - 一個點到多點的求法（Dijkstra演算法）。
  - 各點至其它點的求法（Floyd演算法）。

52

一個點到多點的最短路徑

- 從一個點到多點最短路徑的求法中，最著名的是Dijkstra演算法。例如：從基隆到高雄，中間經過台北、新竹、台中或桃園，各城市間的距離，如下圖所示：



G11

53

一個點到多點的最短路徑

- 點1基隆到各點城市間的距離，很明顯的！可以發現基隆到新竹的最短路徑，也是基隆到台中或高雄最短路徑的一部分，也就是說，已知的最短路徑點，可能就是其它點最短路徑上經過的點，如下表所示：

開始頂點	結束頂點	路徑	距離
基隆-V1	台北-V2	1, 2	35
基隆-V1	新竹-V3	1, 3	90
基隆-V1	新竹-V3	1, 2, 3	35+45 = 80
基隆-V1	桃園-V4	1, 2, 4	35+30 = 65
基隆-V1	台中-V5	1, 3, 5	90+25 = 115
基隆-V1	台中-V5	1, 2, 3, 5	35+45+25 = 105
基隆-V1	台中-V5	1, 2, 4, 5	35+30+45 = 110
基隆-V1	高雄-V6	1, 2, 3, 5, 6	35+45+25+200 = 305
基隆-V1	高雄-V6	1, 2, 4, 5, 6	35+30+45+200 = 310

54

Dijkstra演算法的最短路徑

- Dijkstra演算法是依據上述最短路徑的特性，提出解決「單來源最短路徑的問題」（Single-source Shortest Paths）的演算法，Dijkstra演算法使用鄰接矩陣表示法建立圖形。例如：圖形G11的鄰接矩陣表示法，如下圖所示：

	第1欄	第2欄	第3欄	第4欄	第5欄	第6欄
第1列	0	35	90	∞	∞	∞
第2列	∞	0	45	30	∞	∞
第3列	∞	∞	0	∞	25	∞
第4列	∞	∞	∞	0	45	∞
第5列	∞	∞	∞	∞	0	200
第6列	∞	∞	∞	∞	∞	0

55

Dijkstra演算法最短路徑的步驟

- Dijkstra演算法的完整操作步驟，如下所示：
  - Step 1：初始相關陣列的內容：
    - (1) 將graph[source][i]來源點複製到一維陣列dist[i]。
    - (2) 設為selected[i]和pi[i]的陣列元素初值。
  - Step 2：執行點總數減1次的迴圈來計算最短路徑，如下所示：
    - (1) 走訪陣列dist[]找出最短距離的點W，且此點沒有選過。
    - (2) 將點W設為選過，即selected[W] = 1。
    - (3) 走訪陣列dist[]，如果有未選過的點X，則：
      - 1) 比較dist[X]和graph[W][X]+dist[W]的距離大小：
        - » a. 如果graph[W][X]+dist[W]小，存入dist[X]。
        - » b. 如果更改距離，指定點X的前點為W，pi[X] = W。

56

Dijkstra演算法過程

- Dijkstra演算法計算圖形G11來源點1到其它各點最短路徑的過程，如下表所示：

迴圈	選擇頂點	dist[]						pi[]						selected[]					
		1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	4	5	6
初值	V1	0	35	90	∞	∞	∞	0	0	0	0	0	0	1	0	0	0	0	0
1	V2	0	35	80	65	∞	∞	0	1	2	2	0	0	1	1	0	0	0	0
2	V4	0	35	80	65	110	∞	0	1	2	2	4	0	1	1	0	1	0	0
3	V3	0	35	80	65	105	∞	0	1	2	2	3	0	1	1	1	1	0	0
4	V5	0	35	80	65	105	305	0	1	2	2	3	5	1	1	1	1	1	0
5	V6	0	35	80	65	105	305	0	1	2	2	3	5	1	1	1	1	1	1

57

最短路徑的第一次迴圈

- 第一次迴圈搜尋陣列dist[]的最短距離點W是點2的35，然後走訪陣列dist[]，將未選過的點X：3、4、5、6，依序計算下列公式，如下所示：

  MIN (dist(X),dist(W)+鄰接矩陣(W,X))
- MIN()函數可以傳回兩個參數距離的最短距離。如果dist(W)+鄰接矩陣(W,X)比較短，例如：點1到點3的距離是dist[3] = 90，透過點2到點3的距離，如下所示：

  dist(2) + 鄰接矩陣(2, 3) = dist[2] + graph[2][3] = 35 + 45 = 80
- 距離80小於dist[3] = 90，所以更新dist[3] = 80，同時，將pi[3]更新為2，表示點3最短路徑的前一個點是2。同理，點4更新為65。

58

最短路徑的第二次迴圈

- 第2次迴圈搜尋陣列dist[]的最短距離點W是點4的65，然後走訪陣列dist[]，將未選過的點X：3、5、6，其中點5可由點4到達，其距離為：

  dist[4] + graph[4, 5] = 65 + 45 = 110
- 更新dist[5] = 110，同時，將pi[5]更新為4，表示點5最短路徑的前一個點是4。
- 重複上述操作，執行點數減一次迴圈，就可以完成點1到其它點的最短路徑計算。

59

找出最短路徑

- 如何知道最短路徑經過哪些點？可以從pi[]陣列得知。
- 例如：點1到點6的路徑，從pi[6]開始，前一個點是5，pi[5] = 3，表示點5的前一個點是3，點3的前一個點是2，所以得到路徑：1,2,3,5,6。

60

各點至其它點的最短路徑

- R. W. Floyd教授的演算法可以計算出各點至其它點的最短路徑。例如：以上一節的加權圖形G11為例，Floyd演算法是將鄰接矩陣表示法的內容複製到二維陣列dist[][]，如下圖所示：

	第1欄	第2欄	第3欄	第4欄	第5欄	第6欄
第1列	0	35	90	∞	∞	∞
第2列	∞	0	45	30	∞	∞
第3列	∞	∞	0	∞	25	∞
第4列	∞	∞	∞	0	45	∞
第5列	∞	∞	∞	∞	0	200
第6列	∞	∞	∞	∞	∞	0

61

各點至其它點的最短路徑

- Floyd演算法使用迴圈重覆執行n次的點數，(i, j)是矩陣座標，走訪二維陣列的每一個元素來計算下列公式，如下所示：

  $dist_n(i,j)=MIN(dist_{n-1}(i,j),dist_{n-1}(i,n)+dist_{n-1}(n,j))$

- 上述n值是第n次執行迴圈MIN()函數，函數找出兩個參數的最小值。當計算出$dist_{n-1}$後，在計算distn(i,j)時一共有兩種情況，如下所示：
  - **沒有經過點n**：最短距離仍然為$dist_{n-1}(i,j)$。
  - **如果有經過點n**：最短距離是比較$dist_{n-1}(i,n)+dist_{n-1}(n,j)$和$dist_{n-1}(i,j)$找出最小值，經過點n的路徑就是從點i到n和點n到j。

62

各點至其它點之第一次迴圈

- 現在執行第一次公式的計算，如下所示：
  $dist_1(i,j) = MIN(dist_0(i,j), dist_0(i,1)+dist_0(1,j))$
- 上述dist0是初始的二維陣列。在計算每一個元素後的陣列內容，如下圖所示：

	第1欄	第2欄	第3欄	第4欄	第5欄	第6欄
第1列	0	35	90	∞	∞	∞
第2列	35	0	45	30	∞	∞
第3列	90	45	0	∞	25	∞
第4列	∞	30	∞	0	45	∞
第5列	∞	∞	25	45	0	200
第6列	∞	∞	∞	∞	200	0

63

各點至其它點之第二次迴圈

- 繼續第二次公式的計算，如下所示：
  $dist_2(i,j) = MIN(dist_1(i,j), dist_1(i,2)+dist_1(2,j))$
- 計算後的陣列內容，如下圖所示：

	第1欄	第2欄	第3欄	第4欄	第5欄	第6欄
第1列	0	35	80	65	∞	∞
第2列	35	0	45	30	∞	∞
第3列	80	45	0	75	25	∞
第4列	65	30	75	0	45	∞
第5列	∞	∞	25	45	0	200
第6列	∞	∞	∞	∞	200	0

64

各點至其它點的執行完迴圈

- 等到執行完全部迴圈一共是點數6次後，就可以得到最後的二維陣列內容，如下圖所示：

	第1欄	第2欄	第3欄	第4欄	第5欄	第6欄
第1列	0	35	80	65	105	305
第2列	35	0	45	30	70	270
第3列	80	45	0	70	25	225
第4列	65	30	70	0	45	245
第5列	105	70	25	45	0	200
第6列	305	270	225	245	200	0

65

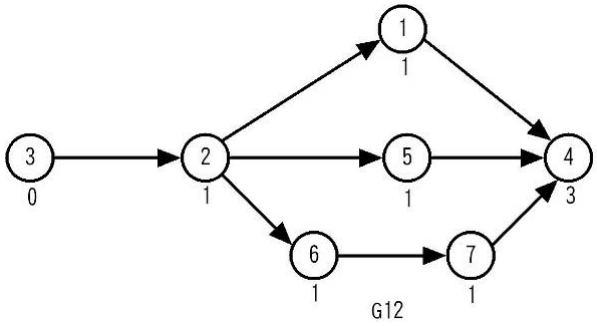
拓樸排序(AOV網路與拓樸排序)

- 工作或作業計劃都需要分割成數個小計劃，如果使用方向性圖形來表示各小計劃間的前後關連，這種圖形稱為「點工作網路」（Activity On Vertex Network），簡稱AOV網路。
- 不過問題是每個小計劃或工作間都擁有關連，所以準備進行某個小計劃前，需要等到其它小計劃先行完成。
  - 例如：主修資訊科學的學生，在選修資料結構這門課前，需要先修過C語言課程，如果沒有修過C語言，就不能選修資料結構。
- 換句話說，因為課程有先後順序，所以如何找出修課的先後順序，而且不會造成先修課程尚未修過的擋修問題，就是「拓樸排序」（Topological Sort）。

66

拓樸排序(AOV網路圖例)

- 例如：一個方向性圖形G12表示課程間的先後關連，如下圖所示：



67

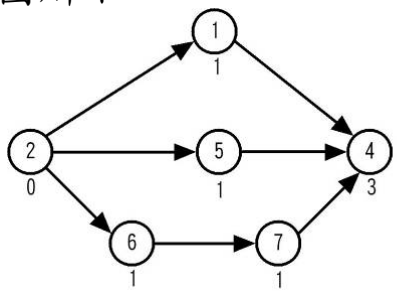
拓樸排序(AOV網路圖例)

- 圖形G12是一個AOV網路，其點代表課程，點1需要等到讀完點3和2的課程後，才可以選修，點3是其它點的「先行者」（Predecessor），點2是點1、5和6的「立即先行者」（Immediate Predecessor）。
- 點4的課程需要等到點1,5,7的課程都讀完，它是圖形其它點的「後繼者」（Successor），點1、5和7的「立即後繼者」（Immediate Successor）。

68

拓樸排序(拓樸排序演算法)

- 拓樸排序的圖形需要計算點的內分支度，內分支度為0的點表示沒有先修課程，所以是課程的開始。
- 拓樸排序就是從內分支度為0的點開始處理，先輸出此點，接著將點相連接的邊線刪除，如下圖所示：

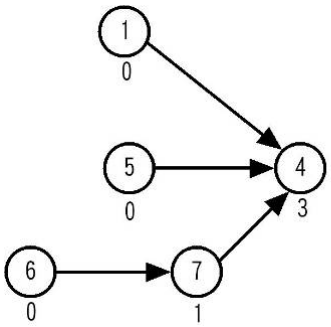


輸出：3

69

拓樸排序(拓樸排序演算法)

- 點2因為邊線已經刪除，內分支度也成為0，輸出點2。繼續將點2連接的三條邊線刪除，如下圖所示：

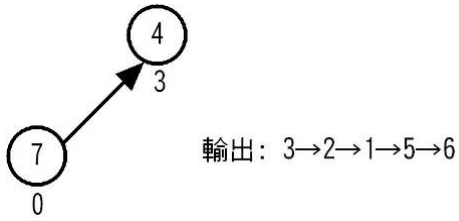


輸出：3→2

70

拓樸排序(拓樸排序演算法)

- 現在點1、5和6的內分支度都是0，只需刪除這三個點和邊線，輸出點1、5和6後，如下圖所示：



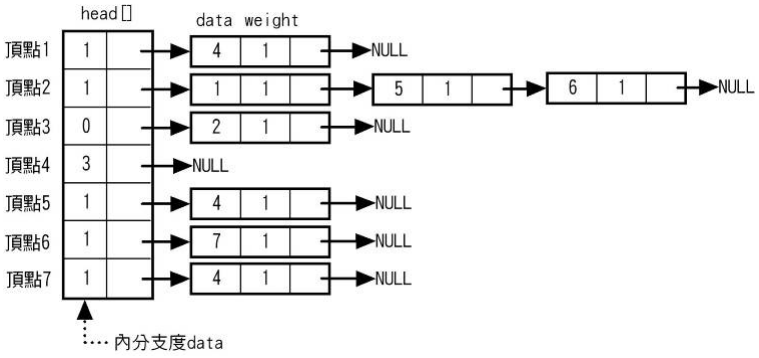
•繼續上述操作，刪除點7，最後輸出點7和4，就可以得到拓樸排序的結果，如下所示：

3→2→1→5→6→7→4

71

拓樸排序(加權鄰接串列表示法)

- 圖形G12的加權鄰接串列表示法，如下圖所示：



72

拓樸排序(拓樸排序演算法)

- Step 1：走訪head[]陣列將內分支度為0的點存入佇列。

```
for ( i = 1; i <= num; i++ )
  if ( head[i].data == 0 )  enqueue(i);
```

- Step 2：從佇列取出點，直到佇列空了為止，如下：

```
while ( !isQueueEmpty() ) {
```

– (1) 取出佇列元素，輸出拓樸排序的點。

```
vertex = dequeue();
printf(" %d ", vertex);
```

– (2) 走訪點的鄰接串列，將所有鄰接點的內分支度減1。

```
ptr = head[vertex].next;
while ( ptr != NULL ) {
vertex = ptr->data;
head[vertex].data--;
```

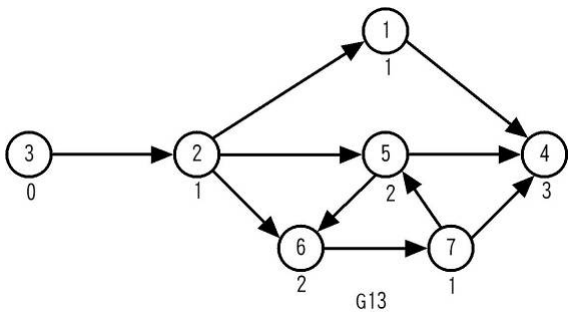
– (3) 如果點減1後的內分支度為0，將此點存入佇列。

```
if ( head[vertex].data == 0  )  enqueue(vertex);
  ptr = ptr->next;
  }
}
```

73

拓樸排序(擁有迴圈的AOV網路)

- AOV網路輸出拓樸排序的條件是點間沒有循環路徑的迴圈，如果AOV網路擁有迴圈，就沒有辦法找出拓樸排序，因為每一個計劃都在等待其它計劃的執行。例如：在圖形G12加上2條邊線，如下圖所示：

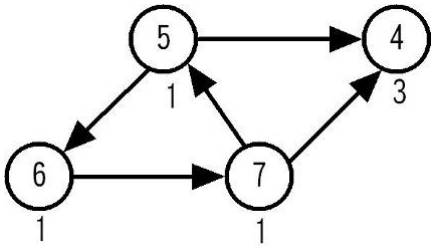


G13

74

拓樸排序(擁有迴圈的AOV網路)

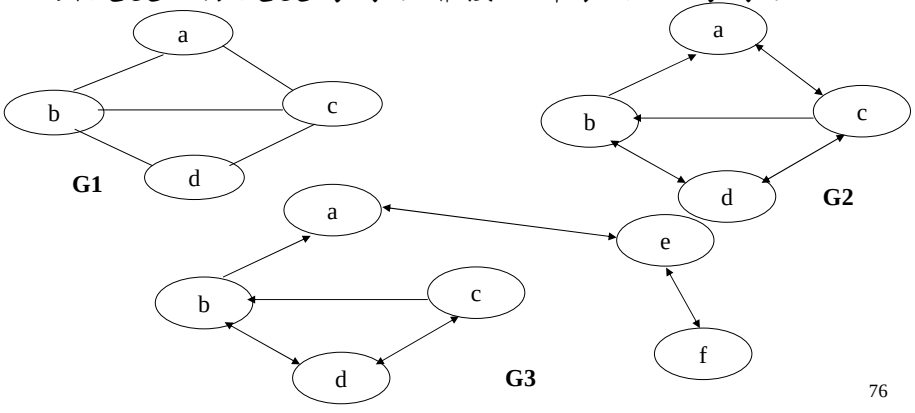
- 圖形擁有迴圈5→6→7→5，拓樸排序無法輸出圖形的所有節點，在輸出3→2→1後留下的圖形，點6等待點5，點7等待點6，點5等待點7，循環等待其它計劃的完成，換句話說，這個計劃將無法完成，如下圖所示：



75

請寫出答案

1. 現有圖形如下三圖，請將之以V和E兩個集合表示？請寫出G2圖之a-d之簡單路徑？路徑長為多少？請寫出G2之循環為何？請寫出G2、G3圖之各點入支度、出支度、分支度為何？鄰接矩陣表示法為何？



76

請寫出答案

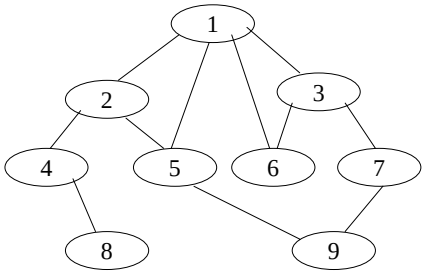
2. 現有一個圖形之鄰接矩陣如下，請畫出此圖？並將之以V和E兩個集合表示？請寫出右圖之各點入支度、出支度、分支度為何？此圖共有幾邊？

	1	2	3	4	5
1	0	1	1	1	0
2	0	0	1	1	0
3	1	0	0	1	0
4	0	1	0	0	1
5	1	0	0	0	0

77

請寫出答案

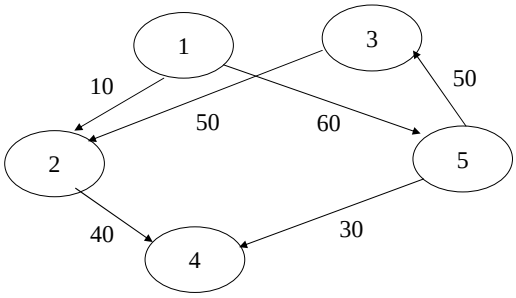
1. 現有一個圖形如下，鄰接矩陣表示法為何？ 深度優先搜尋法之次序為何？ 寬度優先搜尋法之次序為何？{124859736}
2. 深度優先搜尋法之擴張樹為何？寬度優先搜尋法之擴張樹為何？



78

請寫出答案

現有一個加權圖如下，並將之以鄰接矩陣表示？



79