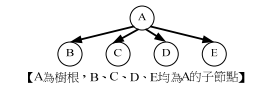
北一女中 2014
資訊選手培訓營
0818-0822

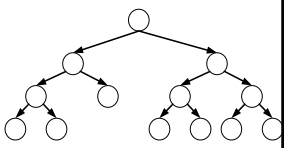
何謂樹狀結構？

- 定義：

樹(tree)是一種特殊的資料結構，它可以用來描述有分支的結構，是由一個或一個以上的節點所組成的有限集合， 且具有下列特質：

  - 存在一個特殊的節點，稱為樹根(root)。
  - 其餘的節點分為n ≥ 0 個互斥的集合，$T_1, T_2, T_3 \ldots T_n$，且每個集合稱為子樹。





4

Agenda

- 8/18(一) 樹狀結構與圖論 (怡芬老師)
- 8/19(二) 排序演算法 (怡芬老師)
- 8/20(三) 網路概論 (致平老師)
- 8/21(四) 動態規畫法 (致平老師)
- 8/22(五) 加解密與壓縮演算法 (怡芬老師)

何謂樹狀結構？

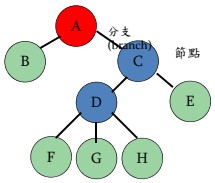
- 樹由數個節點（node)與將節點連接起來的分支（branch)所組成。
- 節點分支出來的節點稱為「子節點」，其上層的節點稱為「父節點」。
- 樹的最上面節點稱為「根」（root)。
- 底下沒有子節點的節點稱為樹葉（leaf)

樹狀結構

- 樹狀結構是一種日常生活中應用相當廣泛的非線性結構。
- 樹狀結構的衍生運用：
  - 企業內的組織架構
  - 家族內的族譜關係
  - 電腦領域中的作業系統
  - 資料庫管理系統

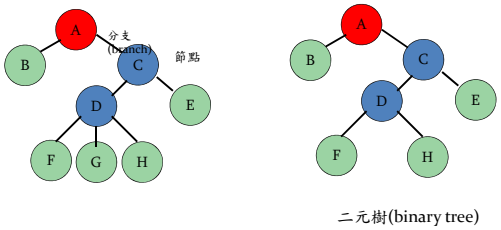
Tree 的深度與高度

- Depth（深度）：由根到某個節點間所通過的分支數，稱為該節點的深度。
- Height(高度）：由根到最深節點的深度，稱為樹的高度。
- 問：
  1. 節點B、G、E的深度分別為？
  2. 樹的高度為？
  3. 節點A(根）的深度為？



二元樹（binary tree)

- 樹的各節點分支如在 2 個以下（含 2 個），則稱為二元樹（binary tree)



二元樹(binary tree)

二元搜尋樹 – 練習題 1

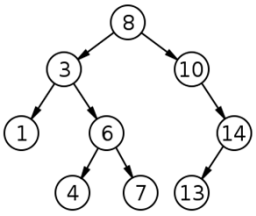
下列何種順序所建造的二元搜尋樹(Binary Search Tree)最平衡(Balanced)?

(a) 30,20,50,5,25,41,80
(b) 5,20,25,30,41,50,80
(c) 80,50,41,30,25,20,5
(d) 50,80,41,30,25,20,5

二元搜尋樹（Binary search tree)

二元搜尋樹是一種二元樹，它可以為空，若不為空，則必須要滿足以下條件：

1. 若左子樹不為空，則左子樹的鍵值均須要小於樹根的鍵值。
2. 若右子樹不為空，則右子樹的鍵值均須要大於樹根的鍵值。
3. 左子樹與右子樹必須也要保持二元搜尋樹。



二元搜尋樹 – 練習題 2

如果依序輸入六筆資料，下列何者所建立的二元搜尋樹(Binary Search Tree)層數最少？

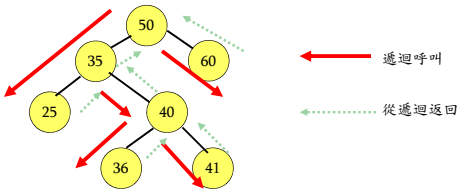
(a) 100, 200, 300, 400, 500, 600
(b) 300, 200, 500, 400, 100, 600
(c) 600, 500, 400, 300, 200, 100
(d) 400, 100, 500, 100, 200, 600

建立二元搜尋樹

1. 依據原始資料輸入順序來建立
2. 第一個輸入的資料(數)當作樹根
3. 接下來輸入的數從樹根的節點資料開始比較
4. 如果比樹根大，就再和其他右子樹相比
   - 如果沒有右子樹，就將新輸入的數當作其右子樹）
   - 如果比樹根小，就再和其他左子樹相比（如果沒有左子樹，就將新輸入的數當作其左子樹）
5. 遞迴執行前二步驟直到位置確定
6. 重複前四步驟直到資料輸入結束

二元搜尋樹的追蹤（traversal)

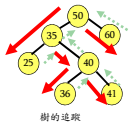
- 以一定的步驟巡視樹的所有節點，稱為樹的追蹤（traversal)。也有人稱為尋訪。



樹的追蹤

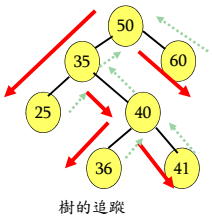
二元搜尋樹的追蹤（traversal)

- 樹的追蹤過程中，「巡視過的節點顯示處理」方式分為：
  - 前序追蹤（preorder traversal)
  - 中序追蹤（inorder traversal)
  - 後序追蹤（postorder traversal)



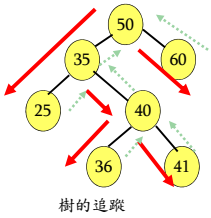
後序追蹤（postorder traversal)

- 左右中
  1. 巡視左側樹的遞迴呼叫
  2. 巡視右側樹的遞迴呼叫
  3. 顯示節點
- 資料顯示順序：
  - 25 36 41 40 35 60 50



前序追蹤（preorder traversal）

- 中左右
  1. 顯示節點
  2. 巡視左側樹的遞迴呼叫
  3. 巡視右側樹的遞迴呼叫
- 資料顯示順序：
  - 50 35 25 40 36 41 60

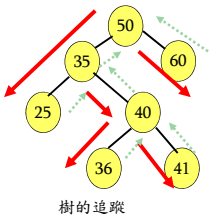


計算式樹

- 有一計算式樹，三種traversal方式分別如下：
  - 前序：-*ab+/cde
  - 中序：a*b-c+d/e
  - 後序：ab*cd+e/-
- 請畫出此樹。

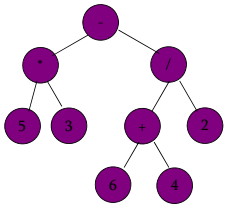
中序追蹤（inorder traversal)

- 左中右
  1. 巡視左側樹的遞迴呼叫
  2. 顯示節點
  3. 巡視右側樹的遞迴呼叫
- 資料顯示順序：
  - 25 35 36 40 41 50 60



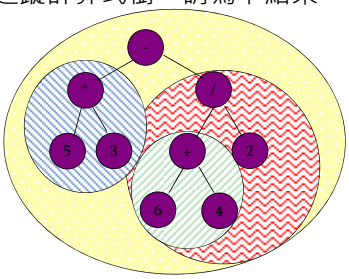
計算式樹的計算

- 以後序追蹤計算式樹，請寫下結果。

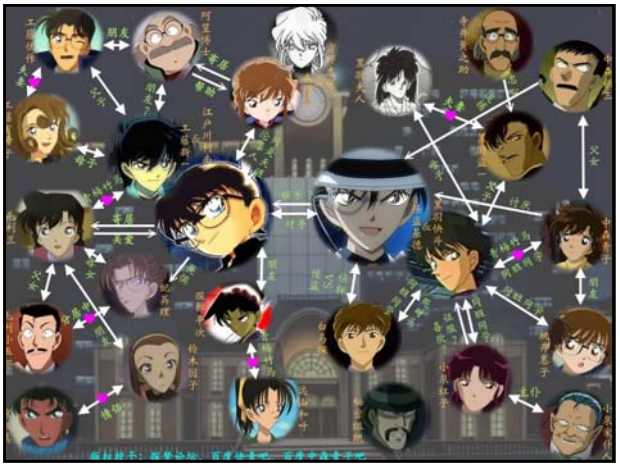


計算式樹的計算

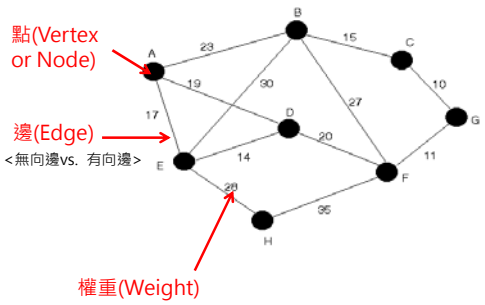
- 以後序追蹤計算式樹，請寫下結果。



圖論



圖形（Graph)

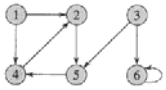


圖形（Graph)

- 圖形(Graph)是指以邊（Edge)將節點（node, Vertex)連接起來的物件。
- G=(V, E)
  - V = vertex set
  - E = edge set

Undirected Graph

- 圖形表示法：
  - Adjacency list
  - Adjacency matrix



Directed Graph

圖形的搜尋

- **Breadth-first search (BFS)**
- **Depth-first search (DFS)**
  - Topological sort
  - Strongly connected components

無向圖（undirected Graph）表示法

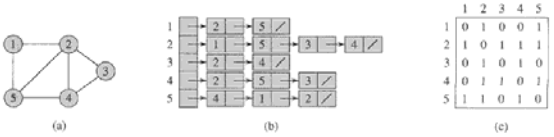


Figure 22.1 Two representations of an undirected graph. **(a)** An undirected graph G having five vertices and seven edges. **(b)** An adjacency-list representation of G. **(c)** The adjacency-matrix representation of G.

Breadth-First Search (BFS)

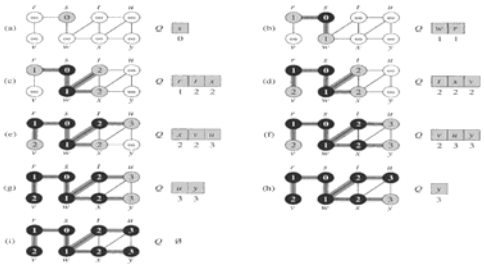


Figure 22.3 The operation of BFS on an undirected graph. Tree edges are shown shaded as they are produced by BFS. Within each vertex u is shown $d[u]$. The queue Q is shown at the beginning of each iteration of the **while** loop of lines 10–18. Vertex distances are shown next to vertices in the queue.

有向圖（directed Graph）表示法

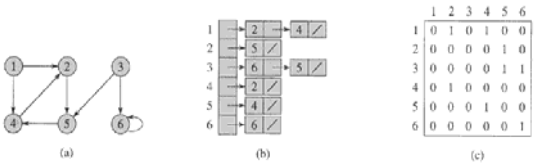


Figure 22.2 Two representations of a directed graph. **(a)** A directed graph G having six vertices and eight edges. **(b)** An adjacency-list representation of G. **(c)** The adjacency-matrix representation of G.

Depth-first search (DFS)

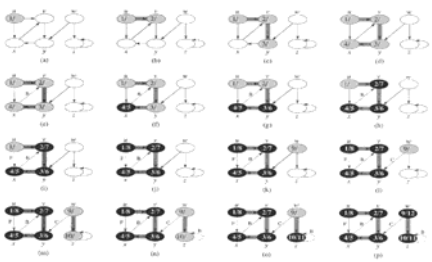
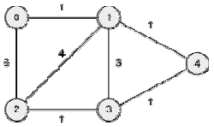


Figure 22.4 The progress of the depth-first-search algorithm DFS on a directed graph. As edges are explored by the algorithm, they are shown as either shaded (if they are tree edges) or dashed (otherwise). Nontree edges are labeled B, C, or F according to whether they are back, cross, or forward edges. Vertices are timestamped by discovery time/finishing time.

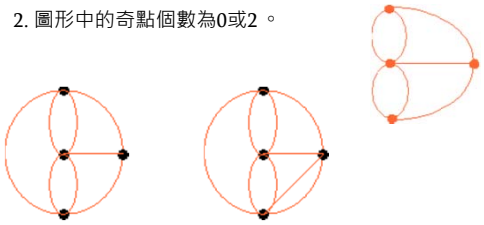
補充：串接的方式

```
#include <stdlib.h>
#define NEW(T, N) (T)malloc((N) * sizeof(T) )

struct edge {
    int id;
    int weight;
    struct edge *next;
};

int main(){
    struct edge *map[5];
    struct edge *ptr = 0;
    ptr = NEW(struct edge, 1);
    ptr->id = 1;
    ptr->weight = 1;
    ptr->next = NULL;
    map[0] = ptr;

    ptr = NEW(struct edge, 1);
    ptr->id = 2;
    ptr->weight = 6;
    ptr->next = NULL;
    map[0]->next = ptr;
//…
}
```



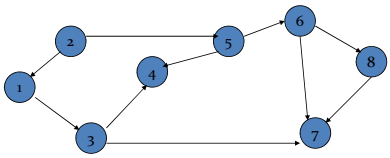
Euler的一筆畫

- 1736年，尤拉發表了他的「一筆劃定理」，大致如下：
  一個圖形要能一筆劃完成必須符合兩個狀況：
  1. 圖形是封閉連通的；
  2. 圖形中的奇點個數為0或2。

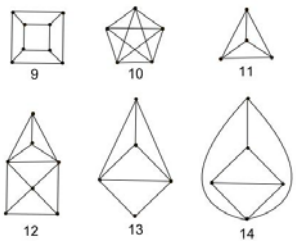


拓樸排序（Topological sort)

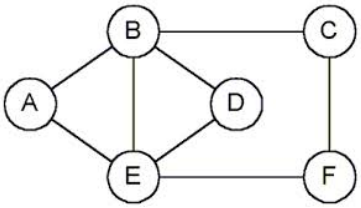
- 拓樸排序是指以某種規則將一有向圖形連接的節點排列成一列的情形。
- 方法不是唯一。



Euler的一筆畫

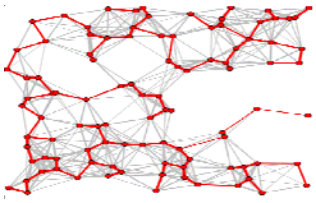


一筆畫?



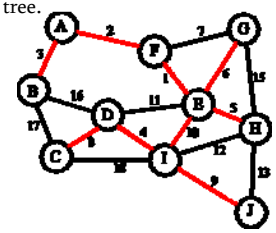
Spanning Tree(展開樹)

- A *spanning tree* of a graph is just a subgraph that contains all the vertices and is a tree.
- A graph may have many spanning trees;
- for instance the complete graph on four vertices

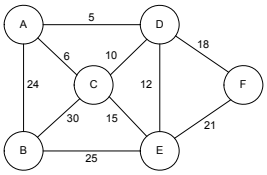


Minimum spanning tree

- The weight of a tree is just the sum of weights of its edges.
- Lemma: Let X be any subset of the vertices of G, and let edge e be the smallest edge connecting X to G-X. Then e is part of the minimum spanning tree.



那個邊（edge）存在於下圖的最小成本生成樹（minimum-cost spanning trees）中？



（a）AB
（b）CD
（c）CE
（d）EF

Kruskal's algorithm

- 最易理解，也最易以手算的方法。

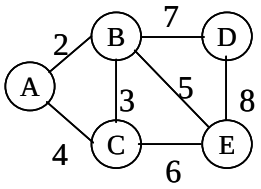
Kruskal's algorithm:

sort the edges of G in increasing order by length
keep a subgraph S of G, initially empty
for each edge e in sorted order
    if the endpoints of e are disconnected in S
    add e to S
return S

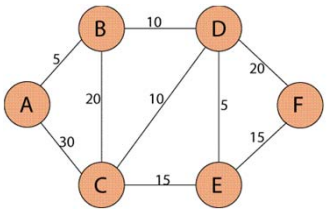
- 這是一個Greedy method (貪心演算法)

Minimum cost spanning tree

下圖中的最小成本擴張樹(Minimum cost spanning tree)的成本為
（a）17
（b）20
（c）22
（d）14



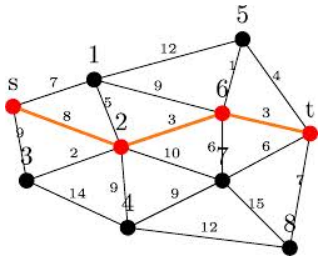
Minimum spanning tree
Kruskal's algorithm



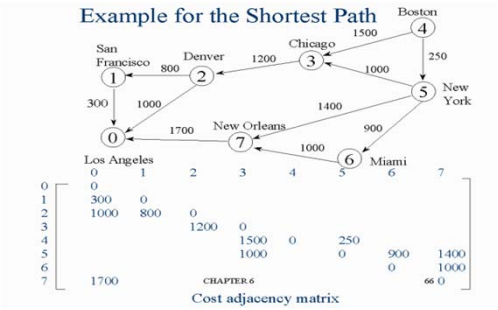
D - 5 - E
A - 5 - B
B - 10 - D
C - 10 - D
C - 15 - E
E - 15 - F
D - 20 - F
B - 20 - C
A - 30 - C

最短路徑(Shortest Path)

最短路徑(Shortest Path)



最短路徑(Shortest Path)



最短路徑(Shortest Path)

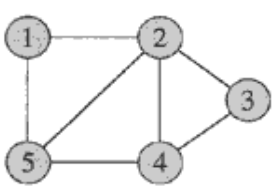
Single Source All Destinations

```
void shortestpath(int v, int
  cost[][MAX_VERTICES], int distance[], int n,
  short int found[])
{
  int i, u, w;
  for (i=0; i<n; i++) {
    found[i] = FALSE;
    distance[i] = cost[v][i];
  }
  found[v] = TRUE;
  distance[v] = 0;

 for (i=0; i<n-2; i++) {determine n-1 paths from v
   u = choose(distance, n, found);
   found[u] = TRUE;
   for (w=0; w<n; w++)
     if (!found[w])
       if (distance[u]+cost[u][w]<distance[w])
         distance[w] = distance[u]+cost[u][w];
 }
}
```

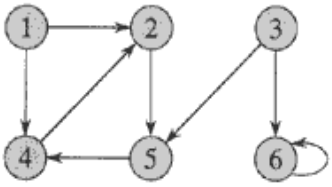
O(n)

圖論(Graph Theory)

- 圖形(Graph)是指以邊（Edge)將節點（node, Vertex)連接起來的物件。
- G=(V, E)
  - V = vertex set
  - E = edge set
- 圖形表示法：
  - Adjacency list
  - Adjacency matrix



Undirected

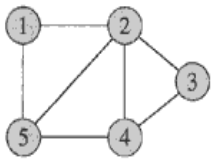


Directed

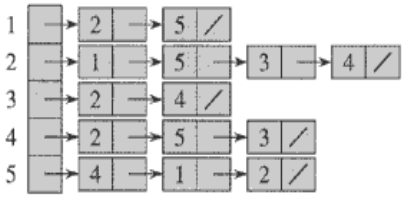
圖形搜尋

- Breadth-first search (BFS)
- Depth-first search (DFS)
  - Topological sort
  - Strongly connected components

無向圖（undirected Graph）表示法



(a)



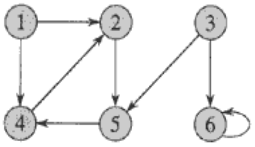
(b)

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

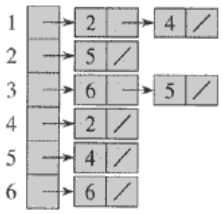
(c)

Figure 22.1 Two representations of an undirected graph. **(a)** An undirected graph G having five vertices and seven edges. **(b)** An adjacency-list representation of G. **(c)** The adjacency-matrix representation of G.

有向圖（directed Graph）表示法



(a)



(b)

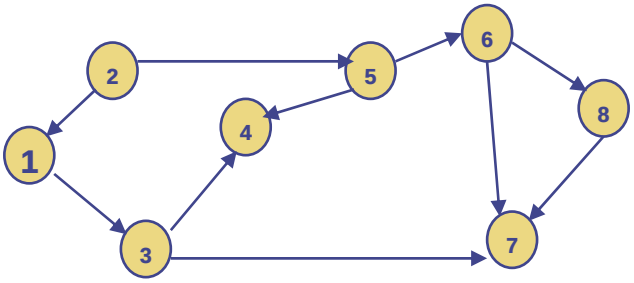
	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

(c)

Figure 22.2 Two representations of a directed graph. **(a)** A directed graph G having six vertices and eight edges. **(b)** An adjacency-list representation of G. **(c)** The adjacency-matrix representation of G.

拓樸排序（Topological sort)

- 拓樸排序是指以某種規則將一有向圖形連接的節點排列成一列的情形。
- 方法不是唯一。



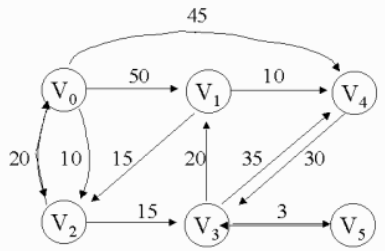

```
/* ------------------------------
 * 拓樸排序   *
 * ------------------------------*/
#include <stdio.h>

#define N 8                             /* 節點數量 */

int a[N+1][N+1]={{0,0,0,0,0,0,0,0,0},   /* 相鄰矩陣 */
                 {0,0,0,1,0,0,0,0,0},
                 {0,1,0,0,0,1,0,0,0},
                 {0,0,0,0,1,0,0,1,0},
                 {0,0,0,0,0,0,0,0,0},
                 {0,0,0,0,1,0,1,0,0},
                 {0,0,0,0,0,0,0,1,1},
                 {0,0,0,0,0,0,0,0,0},
                 {0,0,0,0,0,0,0,1,0}};
int v[N+1];                             /* 查訪旗標 */

void visit(int);

void main(void)
{
    int i;

    for (i=1;i<=N;i++)
       v[i]=0;
    for (i=1;i<=N;i++)
       if (v[i]==0)
          visit(i);
}
void visit(int i)
{
    int j;
    v[i]=1;
    for (j=1;j<=N;j++){
       if (a[i][j]==1 && v[j]==0)
          visit(j);
    }
    printf("%d ",i);
}
```

最短路徑(Shortest Path)

Single Source All Destinations

Determine the shortest paths from v0 to all the remaining vertices.

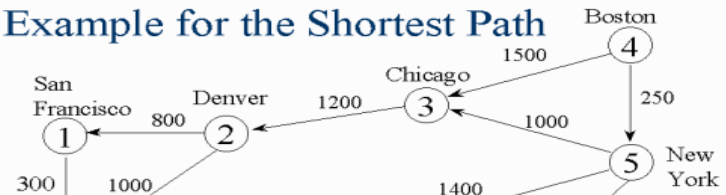
***Figure 6.29:** Graph and shortest paths from v_0 (p.293)*



path	length
1) v0 v2	10
2) v0 v2 v3	25
3) v0 v2 v3 v1	45
4) v0 v4	45

(a) CHAPTER 6 (b) 65

Example for the Shortest Path



	0	1	2	3	4	5	6	7
0	0							
1	300	0						
2	1000	800	0					
3			1200	0				
4				1500	0	250		
5				1000		0	900	1400
6							0	1000
7	1700							0

CHAPTER 6 66

Cost adjacency matrix

Dijkstra 法：從起點的周邊開始，一個一個地確定到各節點的最短路徑，隨著範圍的不斷擴大，最終求到所有各點的最短路徑。

演算法如下：

1. 對於與起點有連接的節點，求出從起點到各節點的距離，確定一個具有最小值的節點，並作上標記。
2. 對於與作上標記的節點有連接的節點，求出它們之間的距離，確定在該點尚未計算(未作標記)的節點中具有最小距離的節點，並作上標記。
3. 重覆上述過程，直到所有節點都作上標記為止。

```
/* ---------------------------------------
 * 最短路徑問題（戴克斯特拉法）   *
 * ---------------------------------------*/

#include <stdio.h>

#define N 8        /* 節點數量 */
#define M 9999

int a[N+1][N+1]={{0,0,0,0,0,0,0,0,0}, /* 相鄰矩陣 */
                 {0,0,1,7,2,M,M,M,M},
                 {0,1,0,M,M,2,4,M,M},
                 {0,7,M,0,M,M,2,3,M},
                 {0,2,M,M,0,M,M,5,M},
                 {0,M,2,M,M,0,1,M,M},
                 {0,M,4,2,M,1,0,M,6},
                 {0,M,M,3,5,M,M,0,2},
                 {0,M,M,M,M,M,6,2,0}};
int main(void)
{
   int j,k,p,start,min,
      leng[N+1],            /* 至節點的距離 */
      v[N+1];               /* 確定旗標 */

   printf("起點");scanf("%d",&start);
   for (k=1;k<=N;k++){
      leng[k]=M;v[k]=0;
   }
   leng[start]=0;

   for (j=1;j<=N;j++){
      min=M;         /* 搜尋最小的節點 */
      for (k=1;k<=N;k++){
         if (v[k]==0 && leng[k]<min){
            p=k; min=leng[k];
         }
      }
      v[p]=1;           /* 確定最小的節點 */

      if (min==M){
         printf("圖形沒有連接\n");
         return 1;
      }

      /* 經由 p 至 k 的距離若比當時最短的距離小的話，便會執行更新作業 */
      for (k=1;k<=N;k++){
         if((leng[p]+a[p][k])<leng[k])
            leng[k]=leng[p]+a[p][k];
      }
   }
   for (j=1;j<=N;j++)
      printf("%d -> %d : %d\n",start,j,leng[j]);
}
```

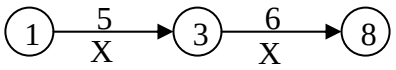
範例：大眾運輸系統　87 年資訊能力競賽全國決賽題

某城市的大眾運輸系統包括公車與捷運。各路線間，無論是公車或捷運，彼此之間都可能有交會點，乘客可以藉此轉乘公車或捷運。
各路線所經過的車站及車站間所需的行車時間為已知，假設轉乘公車的等待時間為 5 分鐘，轉乘捷運的等待時間為 10 分鐘，(例如：公車轉乘捷運需等待 10 分鐘，捷運轉乘捷運也需等待 10 分鐘)，試求從起站 1 號站到其他各站費時最少的乘車方式。

輸入：

公車代號以大寫英文字母代表，捷運代號以小寫英文字母代表。車站代號以數字代表。各路線分別有兩行資料
第一行：路線代號
第二行：有 M 筆資料(M 代表經過的車站數目)，每筆資料包括車站代號與到下一站所需時間，中間以空隔隔開。其中每筆資料所代表的均為單向，而非雙向。當到達下一站的時間為 0 時，表示已是終點站。

例如：




```
X
1 5 3 6 8 0
```

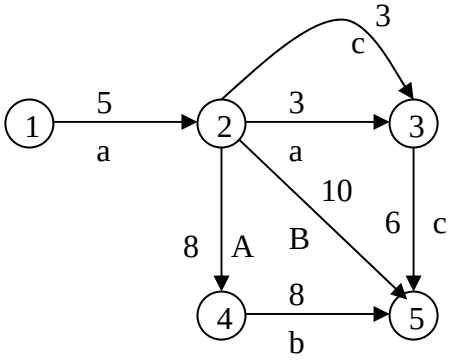
代表 X 號公車的行車路線，起站是 1 號站，經過 5 分鐘到達 3 號站，再經過 6 分鐘到 8 號站，也就是終點站。

輸出：

第一行：站名代號，轉乘次數，所需時間
第二行至第 N＋1 行:
路線經過的各路段，路線代號及所需時間，
其中如果有轉乘，必需加入轉乘的兩個路線代號及轉乘所需時間。

輸入範例：

```
a
1 5 2 3 3 0
b
4 8 5 0
c
2 3 3 6 5 0
A
2 8 4 0
B
2 10 5 0
```



輸出範例：

```
====================
1=>2 (0 transfer, 5 min.)
---------------------------------
```

```
1-> 2 (a, 5 min.)
====================
1=>3 (0 transfer, 8 min.)
---------------------------------
1->2 (a, 5 min.)
2->3 (a, 3 min.)
====================
1=>4 (1 transfer, 18 min.)
---------------------------------
1->2 (a, 5 min.)
a->A (5 min.)
2->4 (A, 8 min.)
====================
1=>5 (1 transfer, 20 min.)
---------------------------------
1->2 (a, 5 min.)
a->B (5 min.)
2->5 (B, 10 min.)
====================
```

資料結構與演算法複習試題(出自：全國資訊競賽 89, 91、IOI 2002, 2003)

Tree

1. 有一種資料結構其為樹狀結構，且在任何位置中其父節點的值恆大於子節點的值?
   (a) 二元樹(binary tree)
   (b) 二元搜尋樹(binary search tree)
   (c) 堆(heap)
   (d) 堆疊(stack)

2. 設T為一個m元樹，也就是T中的每一個節點之分支度小於或等於m。若T中共有n個節點，其中內部節點數為i，葉節點數為j，且共有k個分枝。則以下何者不恆為真?
   (1) $k = i + j - 1$ (2) $j \fallingdotseq (m-1) \times i$ (3) $n \fallingdotseq m \times i + 1$ (4) $i \equiv k/m$

3. 4 個節點所能排出的二元樹之個數有多少？
   (a) 10
   (b) 14
   (c) 20
   (d) 24

4. 樹的深度(depth)為葉子(leaves)到根(root)最長路徑之長度。試問一個深度為h的完整二元樹(complete binary tree)共有幾個節點？
   (a) 2^{h-1}
   (b) $2^{h-1}-1$
   (c) 2^{h+1}
   (d) $2^{h+1}-1$

5. 在二元樹中，根節點屬於第1 層，其子節點屬於第2 層，第2 層節點之子節點屬於第3 層，依此類推。給一個二元樹，樹的深度為k (k≡4)。樹中的每一個節點存有一筆不同值的資料，且對於每個位於奇數層的節點O，O 的資料為以O 為根節點之子樹中的最小值，對於每個位於偶數層的節點E，E 的資料為以E 為根節點之子樹中的最大值。請問整棵樹中最大的資料會出現在此樹的第幾層?
   (a) 第1 層
   (b) 第2 層
   (c) 第3 層
   (d) 第4 層

6. 延續上題，請問整棵樹中第二小的資料會出現在此樹的第幾層?
   (a) 第2 層
   (b) 第1 層或第2 層
   (c) 第2 層或第3 層
   (d) 第3 層

Tree –Traversal

7. 下列何者為中置式 (Infix Expression) (A+B)*C-D/E 的後置式 (Postfix Expression)?
   (a) AB+C*DE/-
   (b) ABC*+DE-/
   (c) AB+C*D-E/
   (d) ABCDE+*-/

8. A + (B + C) 之前置表示法(Pre-Order)為
   (a) A+B+C (b) ABC++
   (c) +A+BC (d) ++ABC

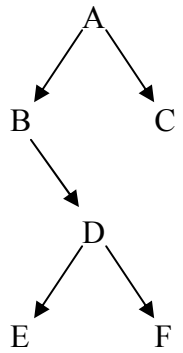
9. 下列何者為A*B+C/(D-E) 的後續式(postfix) 表示法?
   (a) DE-C/B+A* (b) DE-CB/+A*
   (c) AB*CDE-/+ (d) BCDE-/*A+

10. 已知在一棵二元樹T 中包含7個節點，7個節點分別存放A, B, C, D, E, F, G，且資料不重複。今由根節點開始，以前序(preorder traversal)來追蹤此二元樹，且每走到一個節點便印出節點中的資料，得到BDFAGEC的結果，以後序(postorder traversal)來追蹤這棵二元樹，得到AFECGDB的結果，則下列何者不可能為由根節點開始以中序(inorder traversal)來追蹤這棵二元樹的節點順序?
    (1) BAFDEGC (2) FADEGCB (3) BDFGAEC (4) AFDEGCB

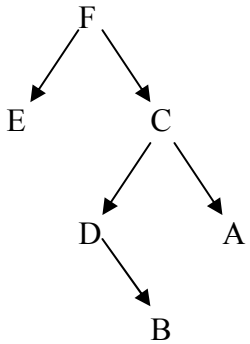
11. 假設一二元樹（binary tree）經前序（Preorder）追蹤可得一次序為ABCDEFGH，經中序（Inorder）追蹤可得一次序為CDBAFEHG，則此樹經後序（Postorder）追蹤後的次序為？
    （a）CDBAEFGH （b）DCBFHGEA
    （c）HGFEABCD （d）ABECFGDH

12. 將中序（infix）的運算式 A/B-C+D*E-A 轉換成後序（postfix）的運算式將是？
    （a）A B C D E A / - + * - （b）A B / + C * D E A –
    （c）A B / C - D E *+ A – （d）A B / C D - E * +A –

13. 下列 A 與 B 兩樹分別用什麼樣的追蹤方式會得到相同的結果



A 樹

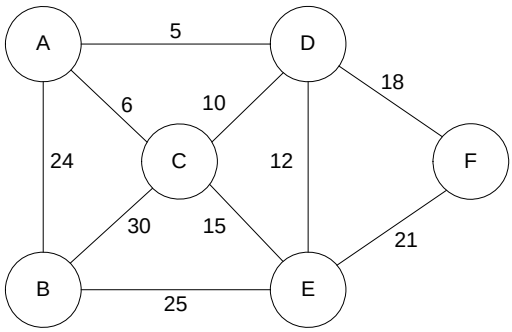


B樹

（a）A 用後序追蹤 B 用前序追蹤（b）A 前序追蹤 B 用中序追蹤
（c）A 後序追蹤 B 用中序追蹤（d）A 用中序追蹤 B 用後序追蹤

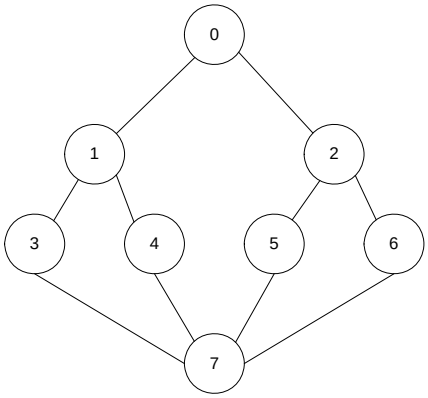
Graph

14. 那個邊（edge）存在於下圖的最小成本生成樹（minimum-cost spanning trees）中？



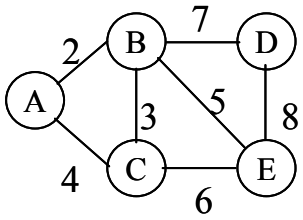
（a）AB
（b）CD
（c）CE
（d）EF

15. 從頂點0開始，利用depth-first search的方法走訪下圖，則所有點會以何種順序被走過？



（a）0,1,2,3,4,5,6,7
（b）0,1,3,4,2,5,6,7
（c）0,1,3,4,7,2,5,6
（d）0,1,3,7,4,5,2,6

16. 下圖中的最小成本擴張樹(Minimum cost spanning tree)的成本為



（a）17
（b）20
（c）22
（d）14

1.	a	2.	4	3.	4	4.	3	5.	3
6.	3	7.	4	8.	d	9.	c	10.	d
11.	a	12.	d	13.	c	14.	d	15.	2
16.	c								