Day5: 資料結構基礎 07/12

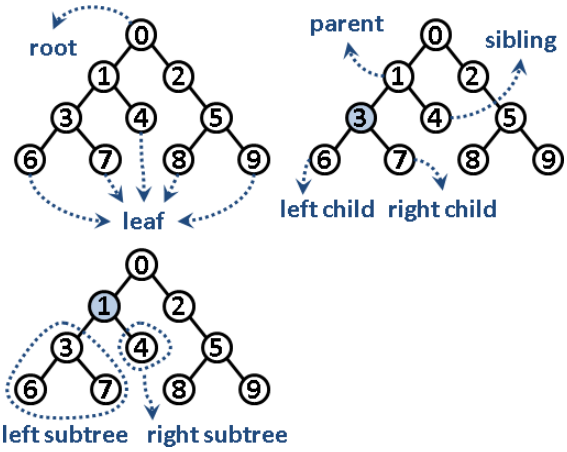
- Stack, queue, 運算式解析
- 二元樹與走訪
- 圖的 DFS 與 BFS
- 拓撲排序演算法、尤拉迴路
- Uva 514, 10305

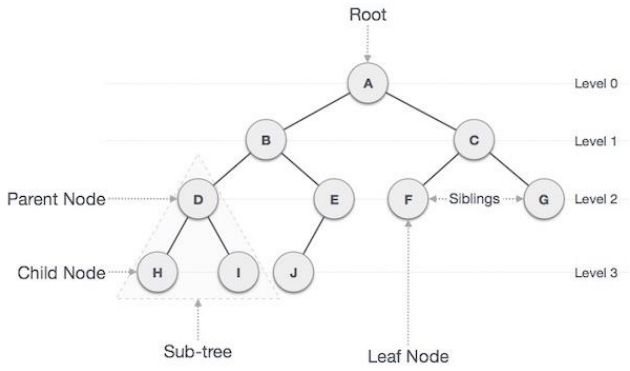
樹狀結構

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…T_n，且每個集合稱為子樹





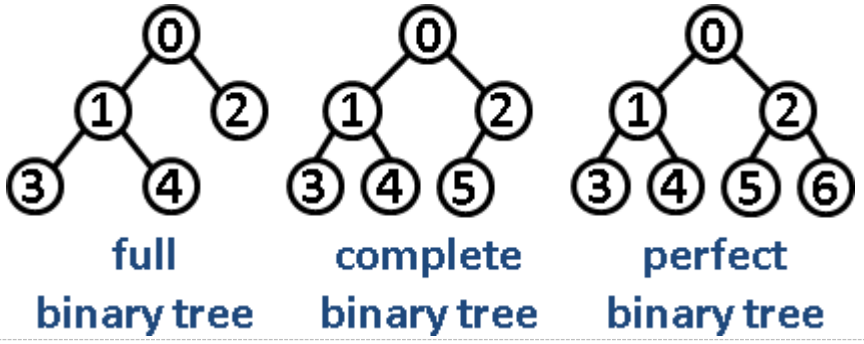
- 樹由數個節點（node)與將節點連接起來的分支（branch)所組成。
- 節點分支出來的節點稱為「子節點」，其上層的節點稱為「父節點」。
- 樹的最上面節點稱為「根」(root)。
- 底下沒有子節點的節點稱為樹葉（leaf)

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. 求節點 B、G、E 的深度
  2. 求樹的高度
  3. 求節點 A(根）的深度

二元樹（binary tree)

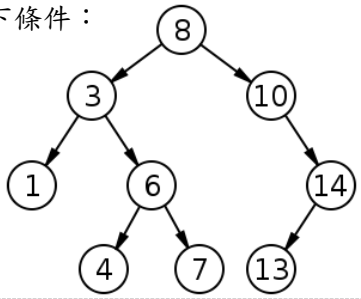
- 樹的各節點分支如在2個以下（含2個），則稱為**二元樹（binary tree)**



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。



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

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

二元搜尋樹的追蹤（traversal)

- 前序追蹤（preorder traversal)
- 中序追蹤（inorder traversal)
- 後序追蹤（postorder traversal)

前序追蹤（preorder traversal）

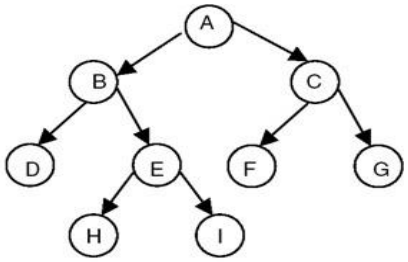
- 中左右
  1. 顯示節點
  2. 巡視左側樹的遞迴呼叫
  3. 巡視右側樹的遞迴呼叫
- 資料顯示順序：
  - 50 35 25 40 36 41 60

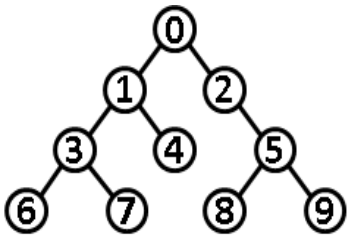
中序追蹤（inorder traversal)

- 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

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





Inorder:

Preorder:

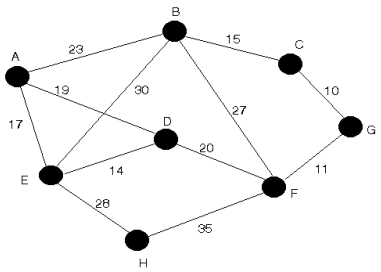
Postorder:

計算式樹

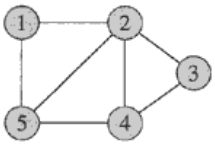
- 有一計算式樹，三種 traversal 方式分別如下：
  - 前序：-*ab/+cde
  - 中序：a*b-c+d/e
  - 後序：ab*cd+e/-
- 請畫出此樹。

圖形（Graph)

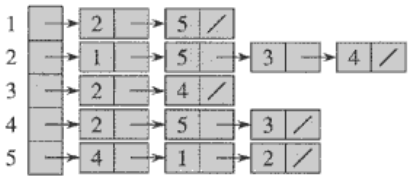
- 圖形(Graph)是指以邊（Edge)將節點（node, Vertex)連接起來的物件。
- G=(V, E)
  - V = vertex set
  - E = edge set
- 圖形表示法：
  - Adjacency list
  - Adjacency matrix



無向圖（undirected Graph）表示法



(a)

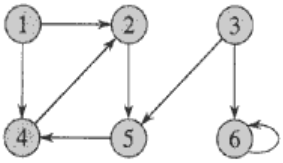


(b)

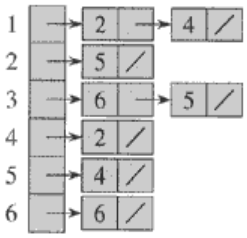
	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

(c)

有向圖（directed Graph）表示法



(a)

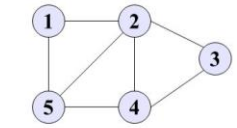


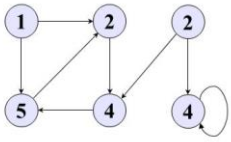
(b)

	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

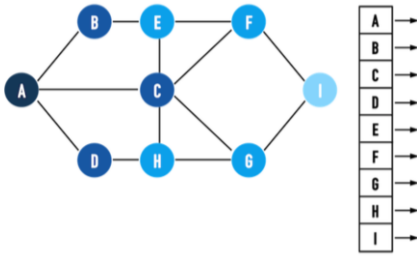
(c)

Adjacency matrix





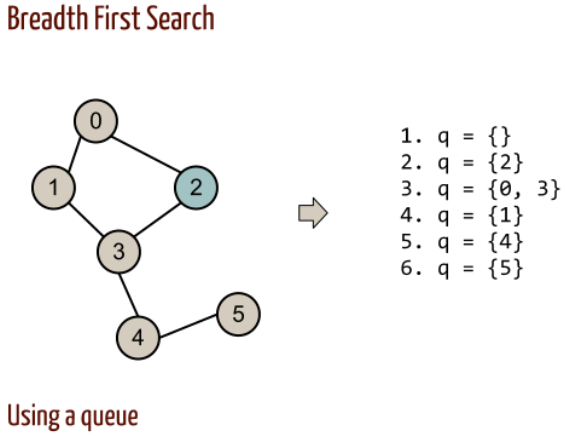
Adjacency List



圖形的搜尋

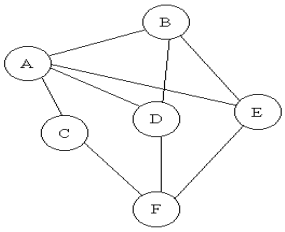
- Breadth-first search (BFS)
- Depth-first search (DFS)
  - Topological sort
  - Strongly connected components

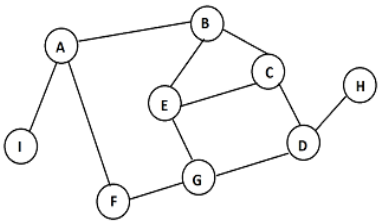
Breadth-First Search (BFS)

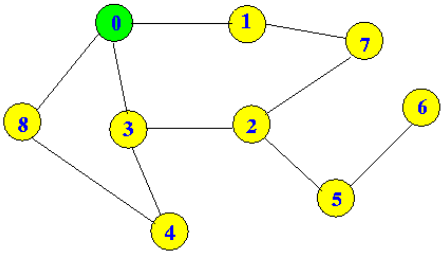


```
#include <string.h>
#define N 7
int visited[N]; // visited array
int graph[N][N] = {
       { 0, 1, 1, 0, 0, 0, 0 },
         { 0, 0, 0, 0, 0, 0, 1 },
         { 0, 0, 0, 1, 0, 0, 1 },
         { 0, 0, 0, 0, 1, 0, 0 },
         { 0, 0, 0, 0, 0, 0, 1 },
         { 6, 0, 0, 1, 0, 0, 0 },
         { 0, 0, 0, 0, 0, 0, 0 } };
int front = 0;
int rear = 0;
int q[N] = { 0 };

void bfs(int v);
int main() {
    // make all vertex unvisited
    memset(visited, 0, sizeof(visited));
     // run bfs from 0th vertex
    bfs(0);
    return 0;
}
void bfs(int v) {
      visited[v] = 1; // make vertex v visited
    // enqueue vertex v
    q[rear] = v; // insert at rear
    rear++; // increment rear
     while (rear != front) // condition for empty queue
    {
         // dequeue
        int u = q[front];
        printf("%d ", u);
        front++;
        // check adjacent nodes from u
         for (int i = 0; i < N; i++) {
            // if there is adjacent vertex enqueue it
            if (!visited[i] && graph[u][i]) {
                q[rear] = i;
                rear++;
                visited[i] = 1;
            }
        }
    }
    printf("\n");
}
```

畫出這個圖：





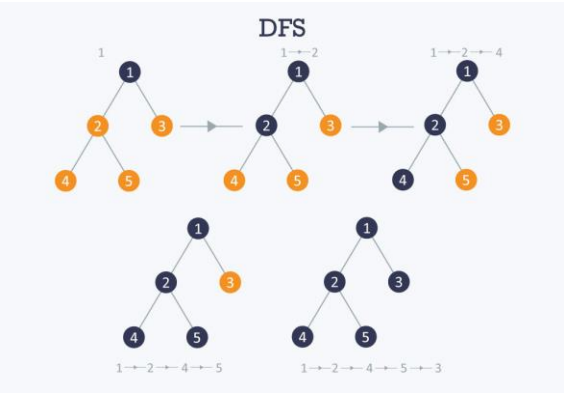


BFS：

BFS：

BFS：

Depth-first search (DFS)



思考：依何種方式走訪儲存資料？

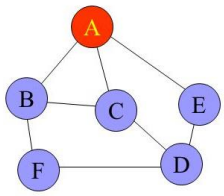
DFS Algorithm (Pseudo Code)

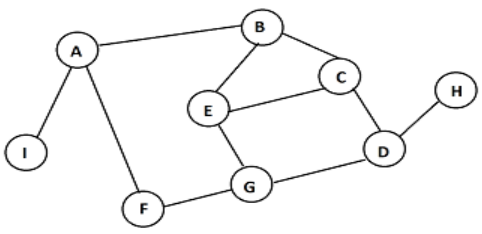
```
n ← number of nodes
Initialize visited[ ] to false (0)
for(i=0;i<n;i++)
    visited[i] = 0;

void DFS(vertex i) [DFS starting from i]
{
    visited[i]=1;
    for each w adjacent to i
        if(!visited[w])
            DFS(w);
}
```

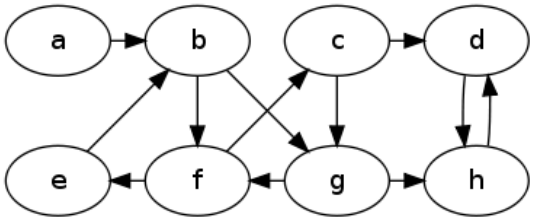
DFS Example

```
#include<stdio.h>

void DFS(int);
int G[10][10],visited[10],n;
//n is no of vertices and graph is sorted in array G[10][10]
void main()
{
    int i,j;
    scanf("%d",&n);
    //read the adjecency matrix
    for(i=0;i<n;i++)
       for(j=0;j<n;j++)
            scanf("%d",&G[i][j]);

   //visited is initialized to zero
   for(i=0;i<n;i++)
        visited[i]=0;

   DFS(0);
}
void DFS(int i)
{
    int j;
    printf("\n%d",i);
    visited[i]=1;

    for(j=0;j<n;j++)
       if(!visited[j]&&G[i][j]==1)
            DFS(j);
}
```

- Probing List is implemented as stack (LIFO)
- Example
  - A's neighbor: B, C, E
  - B's neighbor: A, C, F
  - C's neighbor: A, B, D
  - D's neighbor: E, C, F
  - E's neighbor: A, D
  - F's neighbor: B, D
  - start from vertex A







DFS:

UVA 572 Oil Deposits 油田（DFS 求連通塊）

Description

Due to recent rains, water has pooled in various places in Farmer John's field, which is represented by a rectangle of N x M (1 <= N <= 100; 1 <= M <= 100) squares. Each square contains either water ('W') or dry land ('.'). Farmer John would like to figure out how many ponds have formed in his field. A pond is a connected set of squares with water in them, where a square is considered adjacent to all eight of its neighbors.

Given a diagram of Farmer John's field, determine how many ponds he has.

Input

* Line 1: Two space-separated integers: N and M
* Lines 2..N+1: M characters per line representing one row of Farmer John's field. Each character is either 'W' or '.'. The characters do not have spaces between them.

Output

* Line 1: The number of ponds in Farmer John's field.

Sample Input

10 12
W........WW.
.WWW.....WWW
....WW...WW.
.........WW.
.........W..
..W......W..
.W.W.....WW.
W.W.W.....W.
.W.W......W.
..W.......W.

Sample Output

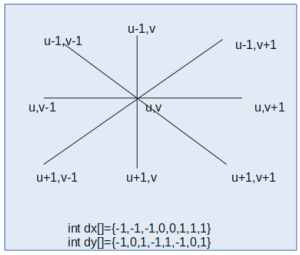
3

解析：

- 一般用 DFS 找連通塊。
- 從每個 W 開始出發，遞迴走訪周圍的「W」格子。
- 每次存取一個格子，就寫一個「連通分量編號」idx
- 這樣就可以在存取前檢查它是否已有編號
- 避免同一個格式存取多次。
- 用雙重迴圈找目前格子相鄰 8 個格子。

floodfill

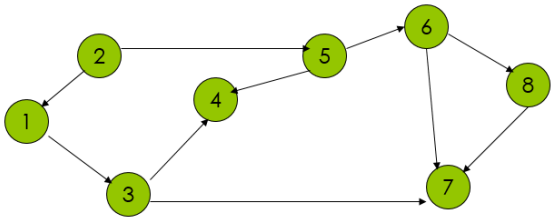
```
#include<cstdio>
#include<cstring>
const int maxn=1000+5;
char tu[maxn][maxn];  //輸入圖的陣列
int m,n,idx[maxn][maxn]; //標記陣列

void dfs(int r,int c,int id)
{
    if(r<0||r>=m||c<0||c>=n)
        return;
    if(idx[r][c]>0||tu[r][c]!='W')
        return;
    idx[r][c]=id;
    for(int dr=-1; dr<=1; dr++)
        for(int dc=-1; dc<=1; dc++)    // 尋找周圍八塊
            if(dr!=0||dc!=0)
                dfs(r+dr,c+dc,id);
}
int main()
{
    int i,j;
    while(scanf("%d%d",&m,&n)==2&&m&&n)
    {
        for(i =0; i<m; i++)
            scanf("%s",tu[i]);
        memset(idx,0,sizeof(idx));
        int q=0;
```



```
        for(i=0; i<m; i++)
            for(j=0; j<n; j++)
                if(idx[i][j]==0&&tu[i][j]=='W')
                    dfs(i,j,++q);
        printf("%d\n",q);
    }
    return 0;
}
```

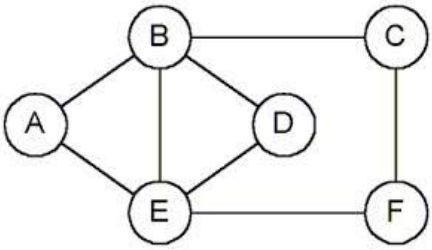
拓樸排序（Topological sort)

- 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。



Ans:

一筆畫? Eular Path

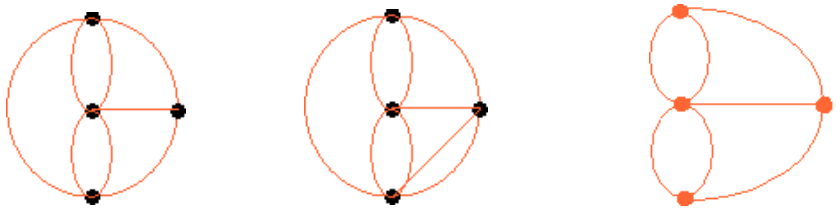


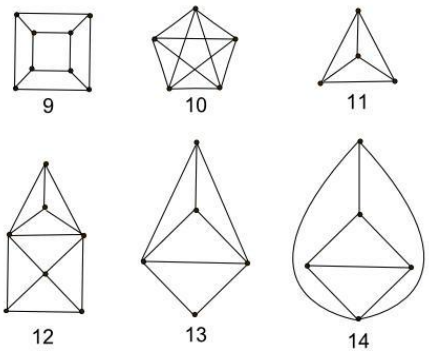

```
void euler(int u)
{
    for (int v=0; v<n; v++)
      if (G[u][v] && !vis[u][v]){
         vis[u][v] = vis[v][u] = 1;
         euler(v);
         printf("%d %d\n", u, v);
    }
}
```

- **1736 年，尤拉發表了他的「一筆劃定理」，大致如下：**

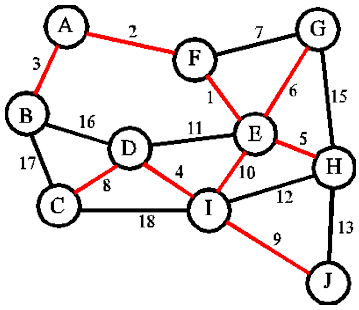
  一個圖形要能一筆劃完成必須符合兩個狀況：

  1. 圖形是封閉連通的；
  2. 圖形中的奇點個數為 0 或 2。





Minimum spanning tree

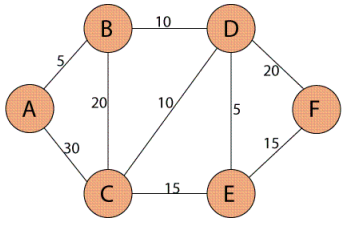


Kruskal's algorithm

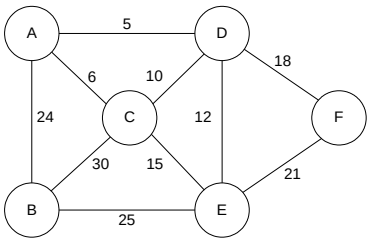
- 最易理解，也最易以手算的方法。

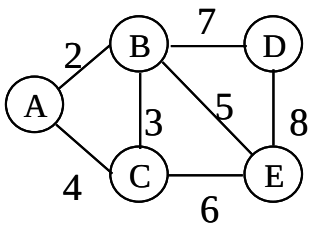
```
Kruskal's algorithm:
sort the edges of G in increasing order by length
keep a subgraph S of G, initially empty
for each edge e in sorted order
        if the endpoints of e are disconnected in S
        add e to S
return S
```

- 這是一個 Greedy method （貪心演算法）

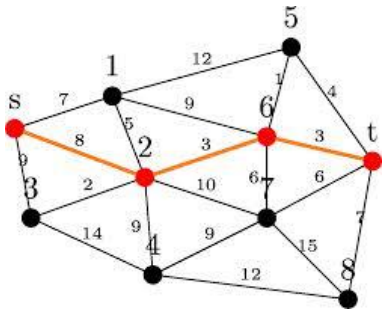


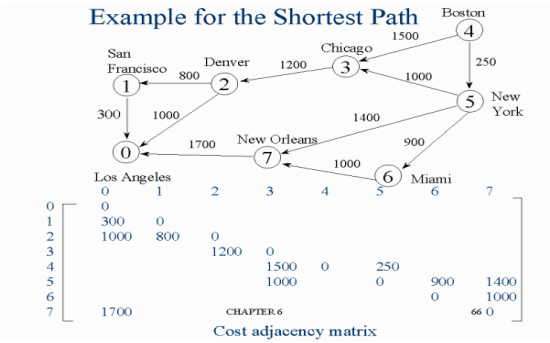
D - 5 - E
A - 5 - B
B - 10 - D
C - 10 - D
C - 15 - E
E - 15 - F
D - 20 - F
B - 20 - C
A - 30 - C





最短路徑(Shortest Path)





Dijkstra's Algorithm

Single Source All Destinations

```
void shortestpath(int v, int
  cost[][MAX_ERXTICES], 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;
```

O(n)

```
  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]) 與u相連的端點w
        if (distance[u]+cost[u][w]<distance[w])
          distance[w] = distance[u]+cost[u][w];
  }
}
```