一、 程式摘要

1. 邏輯/原理

「給定一串指令並依照指令 MAKE-TREE、FIND-DEPTH、GRAFT、END 分別依序處理執行建樹、找深度、合併樹或結束」，我們以下列方式實作。

✓ 建樹

我們只簡單宣告一個資料型態為 integer 的全域陣列，用來記錄 node 與其 parent，索引即代表 node 編號；對應的值即代表 parent 編號。而建樹則是將自己與 parent node 設為一樣的編號。

✓ 合併樹

將 child node 的 parent 設為新的 parent 即可。

✓ 找深度

判斷欲找深度的 node 編號與其 parent node 的編號是否一致，不一致則利用遞迴繼續往上層找，並將深度累加；反之代表找到 root，可直接回傳深度。

2. 語言

以 C 語言實作。

二、 程式內容說明

1. 程式註解

```c
#include <stdio.h>

// 最大值常數化
#define MAX_NUMBER 1001

// 輸出結果用的結構
struct dd_result {
    int number;
    int depth;
};

// 宣告全域陣列，用來儲存各個 node 的 parent
int forest[MAX_NUMBER] = {0};
```

```
// 建樹 function
void make_tree(int v) {
    forest[v] = v;
}

// 找深度 function
int find_depth(int v, int depth) {
    if (v == forest[v]) {
            return depth;
    }
    return find_depth(forest[v], depth + 1);
}

// 合併樹 function
void graft(int r, int v) {
        forest[r] = v;
}

// main funciton
int main() {

    // 宣告裝填結果的結構
        struct dd_result results[MAX_NUMBER];
        // 宣告用來累計輸出結果個數的變數
    int count = 0;
    // 宣告用來判斷是否終止迴圈的變數
    int stop = 0;
    while(stop == 0) {
        // 輸入指令
            char operation;
            scanf("%s", &operation);
        // 判斷指令為何
                switch (operation) {
                        // 建樹
                    case 'M': {
                // 輸入 node 編號
                        int node_number = 0;
                        scanf("%d", &node_number);
                // 呼叫建樹的 function
                    make_tree(node_number);
                            break;
            }
```

```
        // 合併樹
            case 'G': {
            // 輸入 node 及 parent node 的編號
                int child_number = 0;
            int parent_number = 0;
                        scanf("%d %d", &child_number, &parent_number);
            // 呼叫合併樹的 function
            graft(child_number, parent_number);
                    break;
        }
        // 找深度
            case 'F': {
            // 輸入欲找深度的 node 編號
                    int node_number = 0;
                    scanf("%d", &node_number);
            // 呼叫找深度的 function
            int depth = find_depth(node_number, 0);
                        // 裝填輸出結果
                        struct dd_result result;
            result.number = node_number;
            result.depth = depth;
            results[count] = result;
            count++;
                    break;
        }
        // 結束
        // 輸出結果
            default: {
             for(int i = 0; i < count; i++) {
                struct dd_result result = results[i];
                printf("%d %d\n", result.number, result.depth);
             }
             // 終止迴圈
             stop = 1;
                break;
        }
        }
    }
}
```

2. 圖解

✓ **建樹**

我們宣告一個資料型態為 integer 的陣列，以下圖說明建樹過程

(a)

forest

index (node number)	0	1	2	...
parent node number	0	0	0	...

(b)

forest

index (node number)	0	1	2	...
parent node number	0	1	2	...

MAKE_TREE(0) → forest[0] = 0
MAKE_TREE(1) → forest[1] = 1
MAKE_TREE(2) → forest[2] = 2
...

✓ **合併樹**

承上述範例，以下圖說明合併執行後於陣列中的變化

forest

index (node number)	0	1	2	3	...
parent node number	1	1	1	2	...

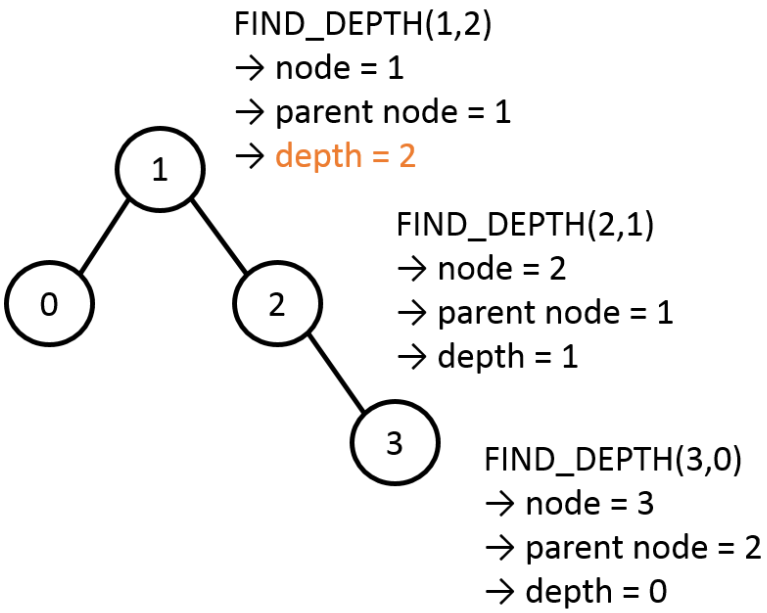
GRAFT(0,1) → forest[0] = 1
GRAFT(2,1) → forest[2] = 1
GRAFT(3,2) → forest[3] = 2

✓ **找深度**

承上述範例，以下圖說明尋找深度的過程

forest

index (node number)	0	1	2	3	...
parent node number	1	1	1	2	...



3. 虛擬碼

```
// 建樹
MAKE_TREE(int node)
    // 以全域陣列 forest 儲存各個 node 的 parent
    // 索引代表 node 編號
    // 其對應到的值即為 parent node 的編號
    // 建樹即是先將自己與 parent node 設為一樣的編號
    forest[node] = node

// 找深度
FIND_DEPTH(int node, int depth)
    // 如果自己與 parent node 的編號一樣，即代表找到 root
    // 直接回傳深度
    if (v == forest[v])
            return depth
    // 若編號不同，則利用遞迴繼續往上層找，並將深度累加
    return FIND_DEPTH(forest[v], depth + 1)

// 合併樹
GRAFT(int child, int parent)
    // 合併樹即是將 child 的 parent 設為新的 parent
       forest[child] = parent
```