一、 程式摘要

1. 邏輯/原理

「一個 directed 的 graph 為 G，u 及 v 為其 vertex，對於所有 u→v 存在至多一條 simple path，則代表存在 Singly Connected。如何決定 G 是否為 Singly Connected」，我們以下列方式實作。

✓ 判斷是否存在 forward edge 或 cross edge

因為 Singly Connected 表所有 u→v 至多只能有一條 simple path，因此只要 vertex 間不存在 forward edge（所有指向 descendant 但不是 tree edge 的 edge）及 cross edge（兩個 vertex 不在同一 Depth-First tree 上或在同一 Depth-First tree 但沒有 ancestor descendant 的關係），G 就會是 Singly Connected。我們以一張 color table 來決定每個 vertex 被訪問的狀態（白色表尚未發現；灰色表已發現；黑色表結束），再以一張 adjacency table 記錄 edge 的 relation，接著針對 G 的每一個 vertex 跑 DFS（Depth-First Search），若以 v 為起始點往下走訪並發現走訪過的 u 又再度被拜訪（color 為黑），則代表 v 與 u 間有其他 path 存在，連結兩 vertex 的 edge 可能是 forward edge 或是 cross edge，我們即可判斷 G 為非 Singly Connected，反之則是。

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>
#include <string.h>

// 定義常數
#define MAX_VERTICES_NUMBER 1000
#define OUPTPUT_CHAR_SIZE 3
#define WHITE 0
#define GRAY 1
#define BLACK 2
#define TRUE 1
#define FALSE 0

const char YES[] = "YES";
const char NO[] = "NO";

// 裝填多個 vertex 的結構
```

```
struct vertices {
	// 記錄 vertex 走訪的狀態
	// 以白(未被發現)、灰(已發現)、黑(結束)表示
	int color[MAX_VERTICES_NUMBER];
	// 記錄 edge 關係
	int adjacency[MAX_VERTICES_NUMBER][MAX_VERTICES_NUMBER];
};

// 裝填結果的結構
struct sc_result {
	int number;
	char is_sc[OUPTPUT_CHAR_SIZE];
};

// 宣告全域變數
struct vertices vs;
char is_sc[OUPTPUT_CHAR_SIZE];

// 復原 vertex 的 function
void clear_vertices(int i) {
	vs.color[i] = WHITE;
}

// 復原 edge 的 function
void clear_edges(int size) {
	for (int i = 0; i < size; i++) {
		for (int j = 0; j < size; j++) {
			vs.adjacency[i][j] = FALSE;
		}
	}
}

// 建立 edge 的 function
void add_edge(int from, int to) {
	// duplicate edge
	if (vs.adjacency[from][to] == TRUE) {
		strcpy(is_sc, NO);
	} else {
		vs.adjacency[from][to] = TRUE;
	}
}
```

```
// 實際以 dfs 走訪各 vertex 的 function
void dfs_visit(int size, int vertex) {
        vs.color[vertex] = GRAY;
        for (int i = 0; i < size; i++) {
                if (vs.adjacency[vertex][i] == TRUE) {
                        if (vs.color[i] == WHITE) {
                                dfs_visit(size, i);
                        } else if (vs.color[i] == BLACK) {
                                strcpy(is_sc, NO);
                                break;
                        }
                }
        }
        vs.color[vertex] = BLACK;
}

// dfs function
void dfs(int size) {
      for (int i = 0; i < size; i++) {

                if (strcmp(is_sc, NO) == 0) {
                        break;
                }

                if (vs.color[i] == WHITE) {
                        dfs_visit(size, i);
                }

                for (int j = 0; j < size; j++) {
                        clear_vertices(j);
                }
        }
}

// main function
int main() {
        // 取得測資筆數
        int input_num;
        scanf("%i", &input_num);

        // 初始化結果結構
        struct sc_result result[input_num];
```

```
    // 開始運行測資
    for (int i = 0; i < input_num; i++) {
        // 初始化記錄是否為 singly connected 的變數
        strcpy(is_sc, YES);

        // 取得 vertex 個數
        int vertex_num;
        scanf("%i", &vertex_num);

        // 取得 edge 個數
        int edge_num;
        scanf("%i", &edge_num);

        // 開始建立 edge
        for (int j = 0; j < edge_num; j++) {
            int from;
            int to;
            scanf("%d %d", &from, &to);
            add_edge(from, to);
        }
        // 呼叫 dfs
        dfs(vertex_num);

        // 裝填結果
        result[i].number = i + 1;
        strcpy(result[i].is_sc, is_sc);

        // 清空 edge
        clear_edges(vertex_num);
    }

    // 輸出結果
    for (int i = 0; i < input_num; i++) {
        printf("%d %s\n", result[i].number, result[i].is_sc);
    }
}
```

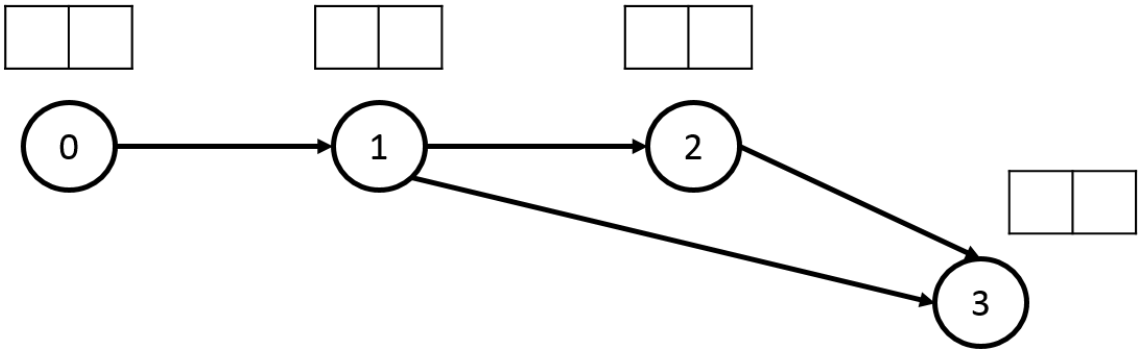
2. 圖解

✓ **找 forward edge**

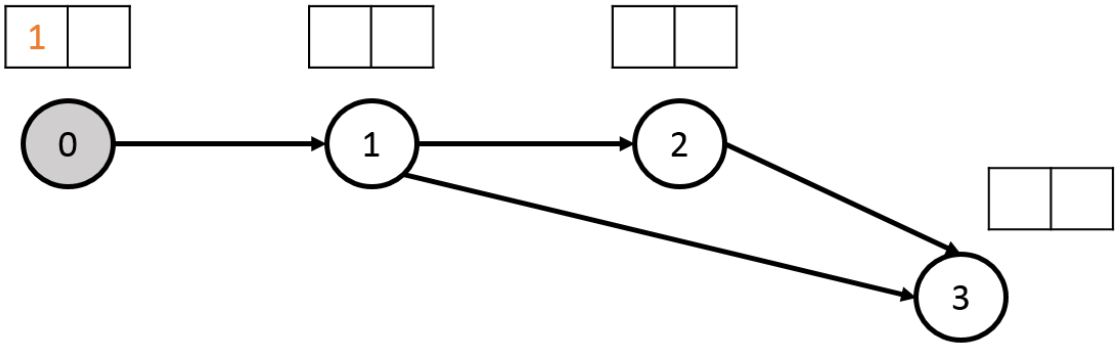
以 vertices = {0, 1, 2, 3} 及 adjacency = {{0, 1}, {1, 2}, {2, 3}, {1, 3}} 為例，以 DFS

走訪找出 forward edge，並判斷為非 Singly Connected。

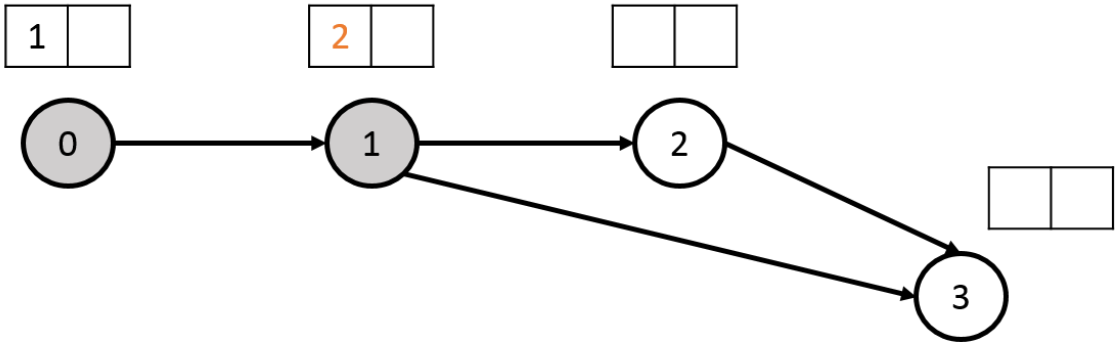
(a)



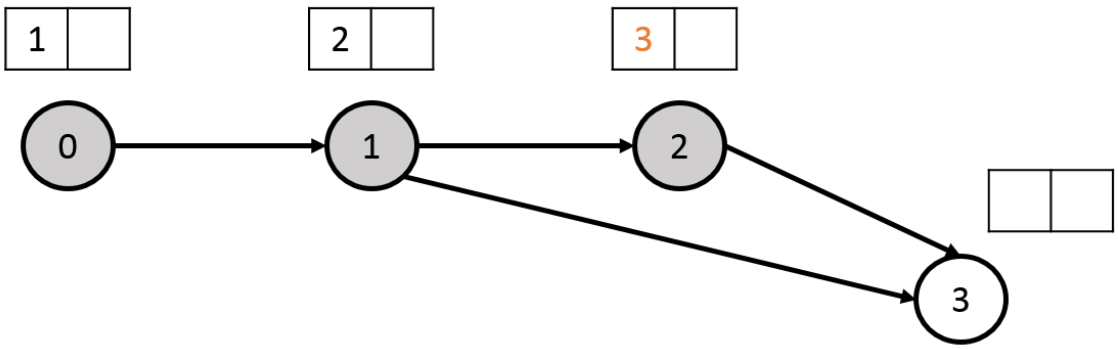
(b)



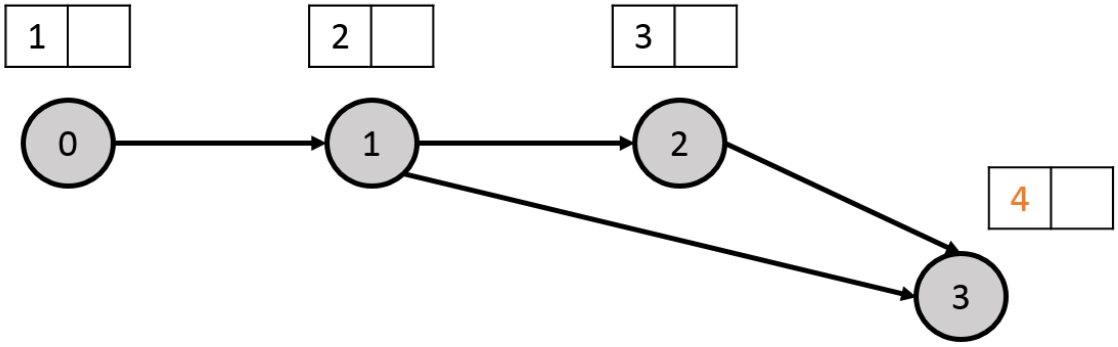
(c)



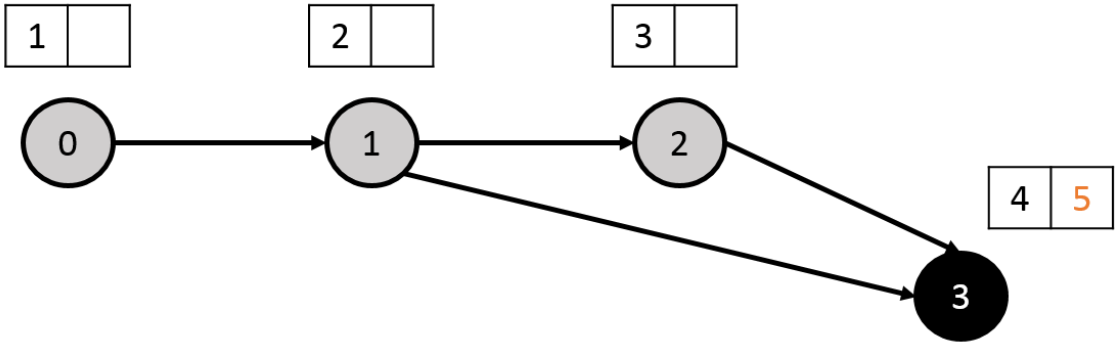
(d)



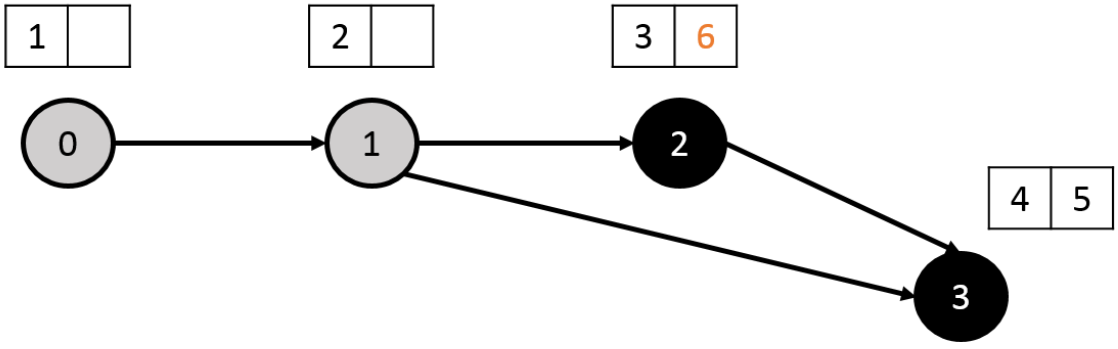
(e)



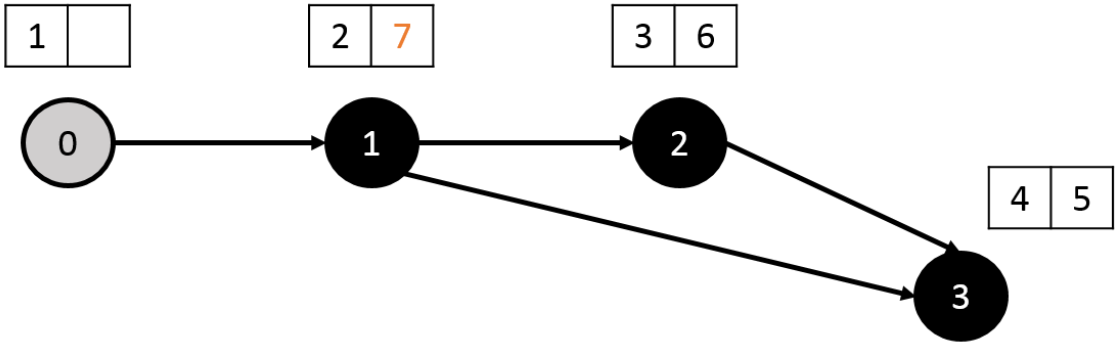
(f)



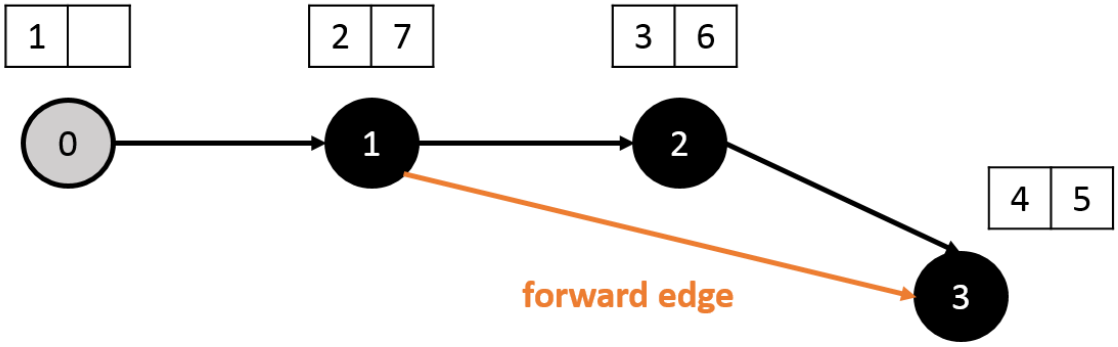
(g)



(h)



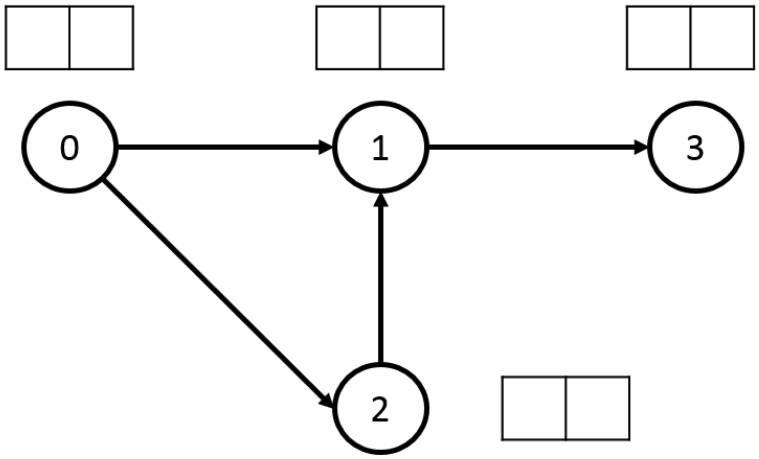
(i)



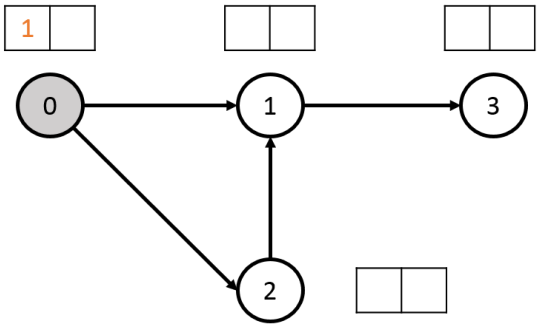
✓ **找 cross edge**

以 vertices = {0, 1, 2, 3} 及 adjacency = {{0, 1}, {0, 2}, {1, 3}, {2, 1}} 為例，以 DFS 走訪找出 cross edge，並判斷為非 Singly Connected。

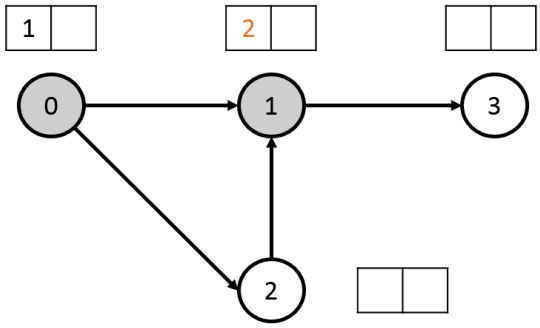
(a)



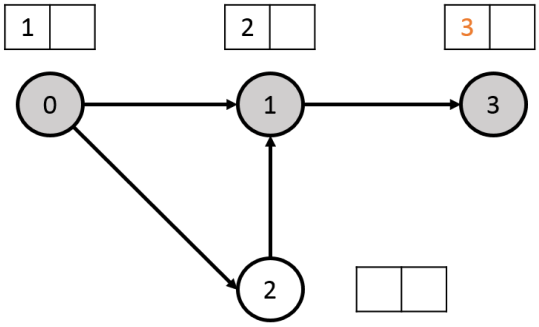
(b)



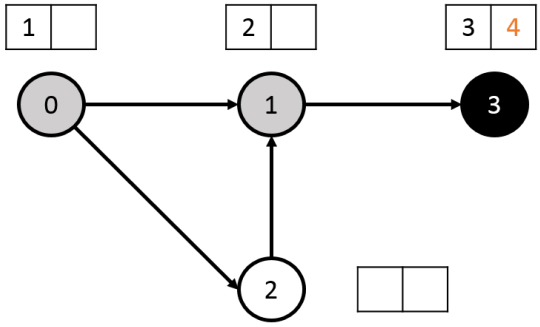
(c)



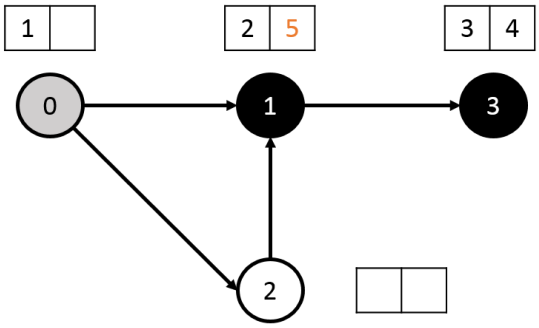
(d)



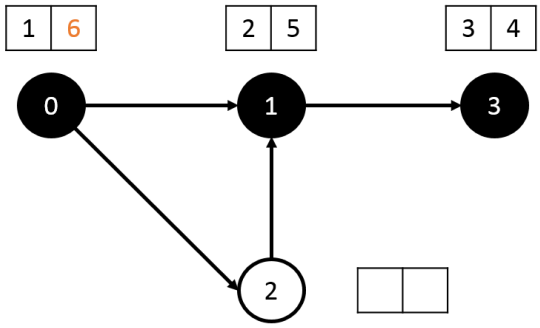
(e)



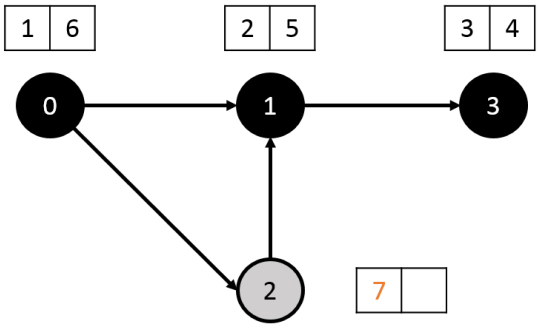
(f)



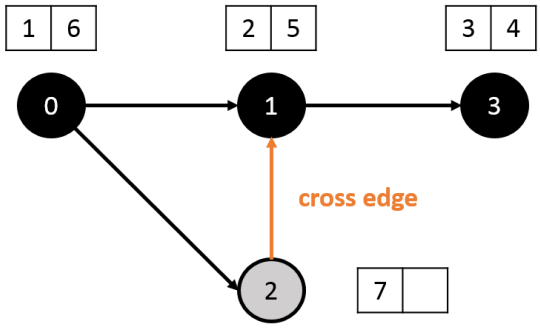
(g)



(h)



(i)



3. 虛擬碼

```
DFS_VISIT(u)
    // 開始走訪
    // 顏色標示為灰
    u.color = GRAY
```

```
        // 從 u 的鄰邊開始判斷
        for each vertex v adjacent to u
            // 顏色為白
            // 尚未走訪過
            if (v.color == WHITE)
                // 以 v 為頂點，再往下走訪
                DFS_VISIT(v);
            // 顏色為黑
            // 已走訪過
            // 代表存在 forward edge 或是 cross edge
            else if (v.color == BLACK)
                // 代表不會是 singly connected
                is_sc = "NO"
                // 跳出迴圈
                break
        // 走訪結束
        // 顏色標示為黑
        u.color = BLACK


DFS()
    // 記錄是否為 singly connected
    is_sc = "YES"
    // 走訪每個 vertex
    for each vertex u in vertices
            // 已判斷不是 singly connected，不用再繼續執行
            if is_sc == "NO"
                break
            // 顏色為白
            // 未走訪過
            if (u.color == WHITE)
                // 以 u 為頂點 開始往下走訪
                DFS_VISIT(u)
```