基礎圖論

2021/06/27

by林品安

圖的介紹

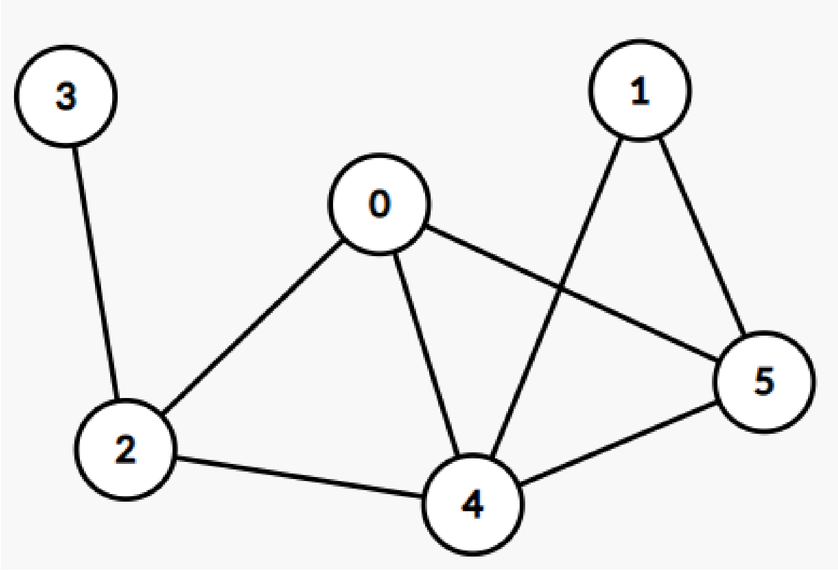
有向圖、無向圖、度(degree)..

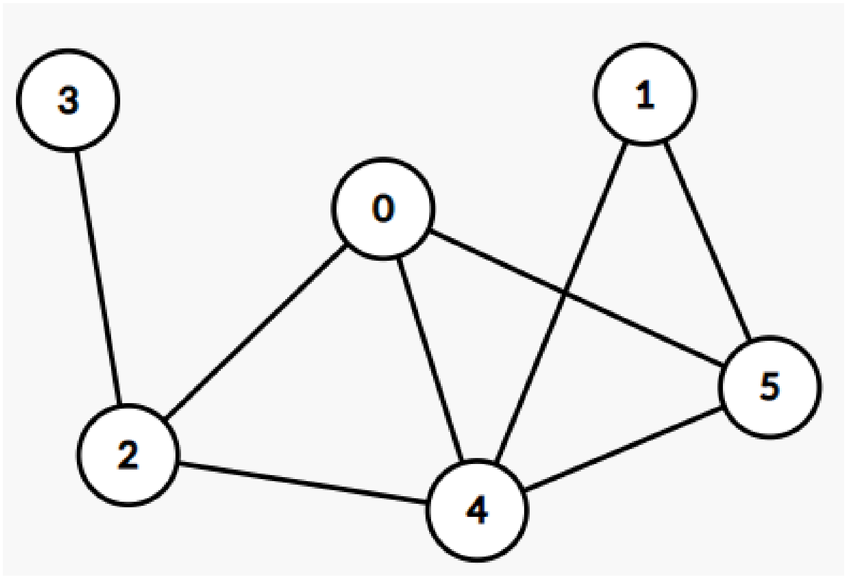
圖論有許多專有名詞需要記，
以後如果忘記了要將講義拿出來看喔。

一個資訊上的圖由 2 個部分構成 ：

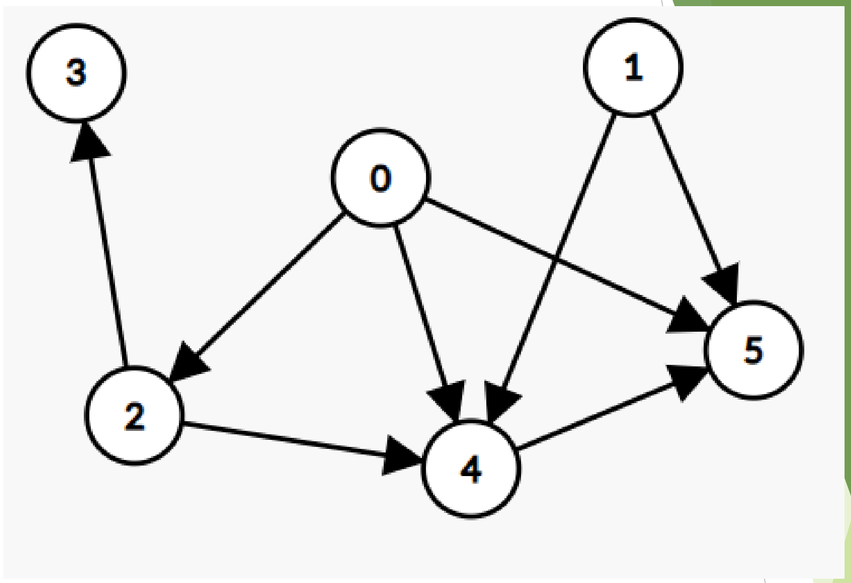
1. 點（V ， $Vertex$）
2. 邊（E ， $Edge$）。

有時候邊或點會有權重(weight)，代表過路費或是距離等等..



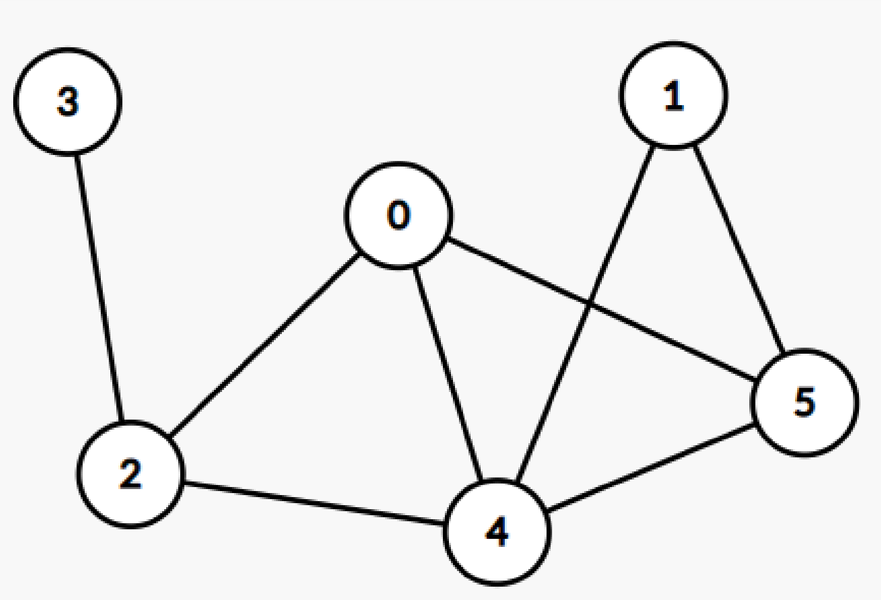


無向圖 (Undirected graph)

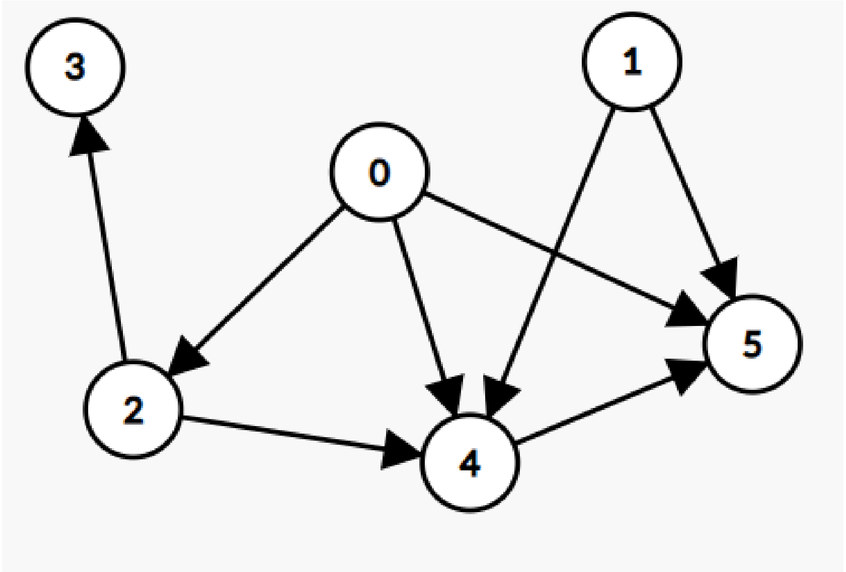


有向圖 (Directed graph)

- 而圖論題目的圖大致上可分為兩大類：分別是有向圖、無向圖
- 而有向圖意味著每條路都是單行道



- 再來介紹的是度（degree），也就是一個點連接的邊的數量
- 如上圖： 點4的度是4；點5的度是3。



- 而有向圖的度又分為入度跟出度。
- 入度就是進來的邊；出度就是出去的邊。
- 如上圖：點 0 →（出度 = 3，入度 = 0）；點 2 →（出度 = 2，入度 = 1）

講了這麼多，要怎麼存圖呢?

一堆點點、一堆線，程式怎麼畫出來???

#用二維陣列直接儲存圖的長相:

題目往往會先給你地圖的長、寬，

然後再給一堆符號，代表空地、障礙物、起點終點…等等

SMAPLE 1:

```
5 5
XXXXX
XXJJX
XJJJX
XXXXX
XJXXX
```

SMAPLE 2:

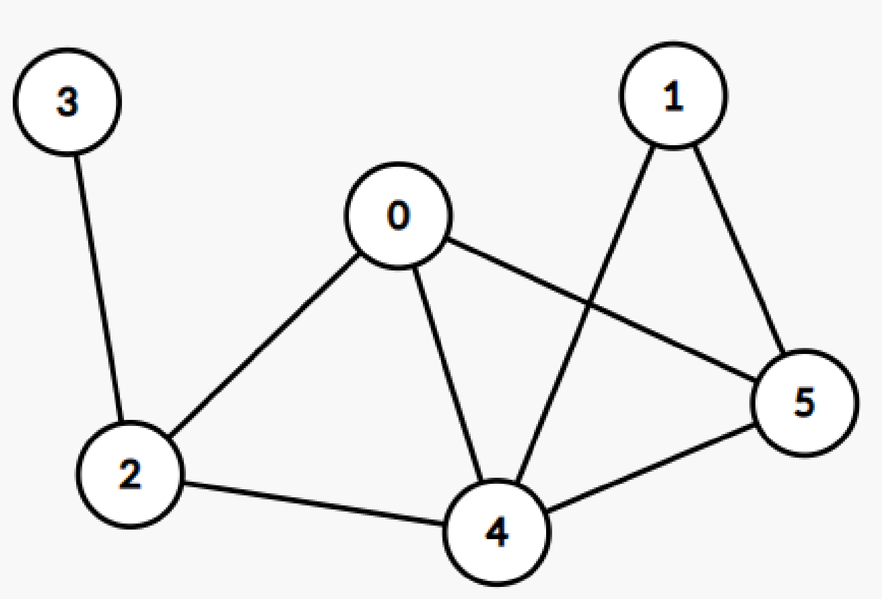
```
3 5
@.a.#
###.#
b.A.B
```

```
char Graph[1005][1005];
//假設要輸入 n*m 大小的地圖
for(int i=0 ; i<n ; ++i){
    for(int j=0 ; j<m ; ++j){
        cin >> Graph[i][j];
    }
}
```

#鄰接矩陣:

假設有 N 個點，開一個 $N * N$ 的二維陣列 G，

如果 $G_{i,j} = 1$ 代表可以從 i 走到 j 。

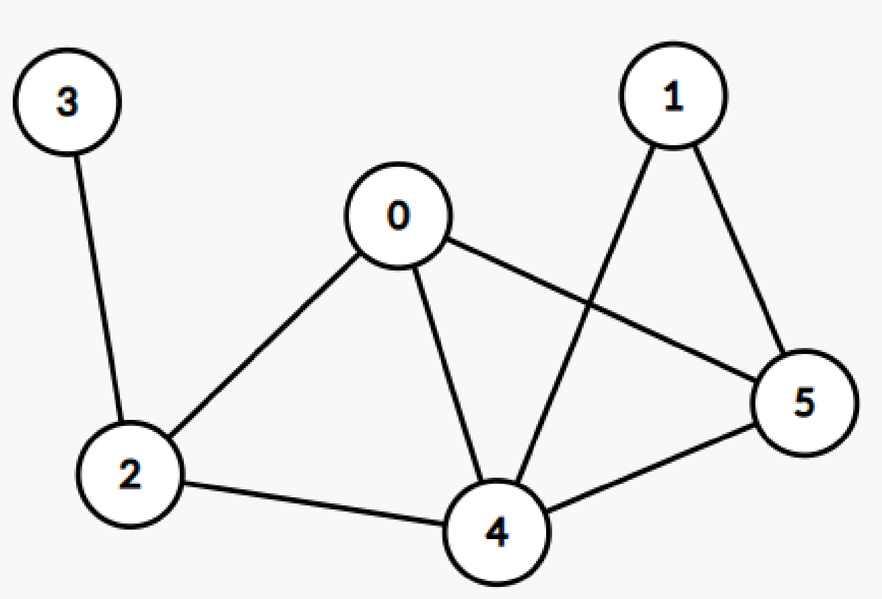


	0	1	2	3	4	5
0	0	0	1	0	1	1
1	0	0	0	0	1	1
2	1	0	0	1	1	0
3	0	0	1	0	0	0
4	1	1	1	0	0	1
5	1	1	0	0	1	0

#鄰接矩陣:

假設有 N 個點，開一個 $N * N$ 的二維陣列 G，

如果 $G_{i,j} = 1$ 代表可以從 i 走到 j 。

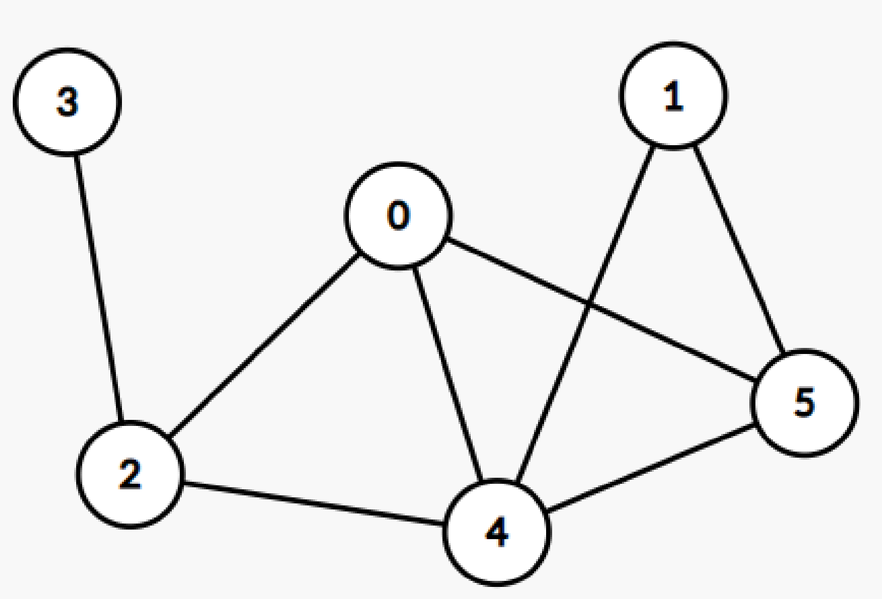



```cpp
int Graph[1005][1005];
//假設要輸入m條邊
for(int i=0 ; i<m ; ++i){
    int a,b; cin >> a >> b;
    //一條 (a,b) 的無向邊
    Graph[a][b] = Graph[b][a] = 1;
}
```

#鄰接串列:

假設有 N 個點，開 N 個 vector，

對於v[i]中的每點j，代表i可以走到j。

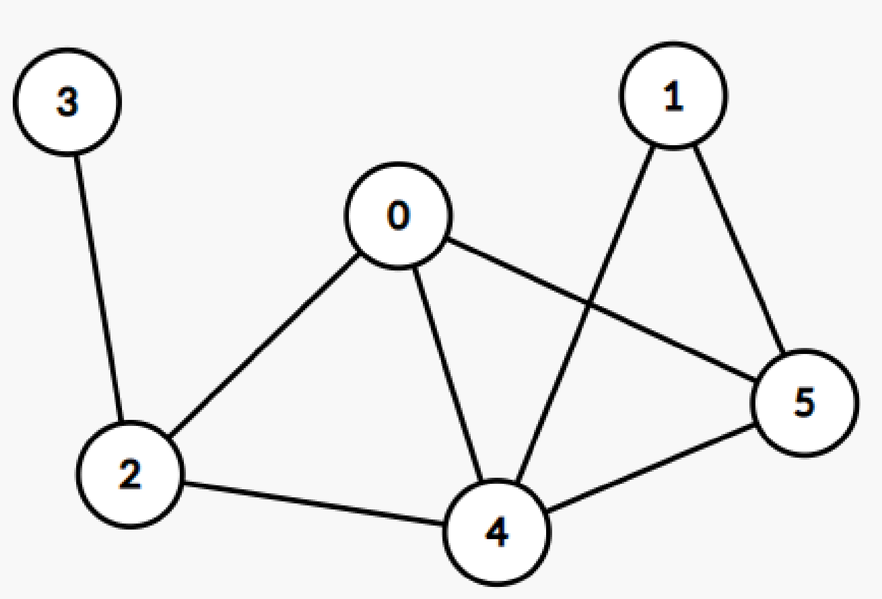


v[0] : 2、4、5
v[1] : 4、5
v[2] : 3、0、4
v[3] : 2
v[4] : 2、0、1、5
v[5] : 0、4、1

#鄰接串列:

假設有 N 個點，開 N 個 vector，

對於v[i]中的每點j，代表i可以走到j。




```cpp
vector<int> Graph[100050];
//假設要輸入m條邊
for(int i=0 ; i<m ; ++i){
    int a,b; cin >> a >> b;
    //一條 (a,b) 的無向邊
    Graph[a].push_back(b);
    Graph[b].push_back(a);
}
```

#優劣勢比較:

- 用二維陣列存圖
  1. 如果用這種方式存圖，題目比較直觀，比較容易思考~

- 鄰接矩陣
  1. 有時候要確認兩點(a, b)是否有邊的時候這種圖比鄰接串列快~
  2. 花費的記憶體很多，要 $N * N$ 的空間

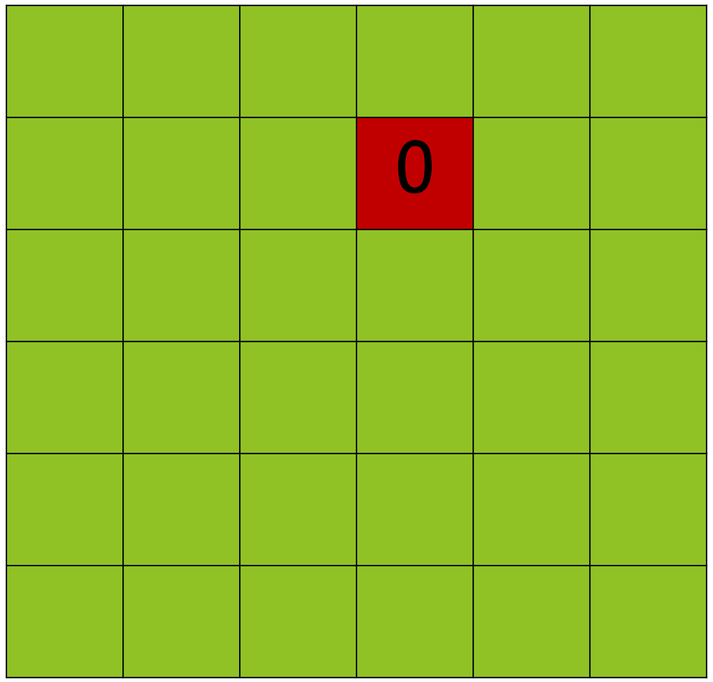
- 鄰接串列
  1. 不存在的邊就不儲存，所以如果有 M 條邊就只會花 M 的空間喔~
  2. 大多數比較難的圖論題目都是用鄰接串列

每個題目都有適合的存圖方式，所以都很重要喔!

圖的遍歷– BFS

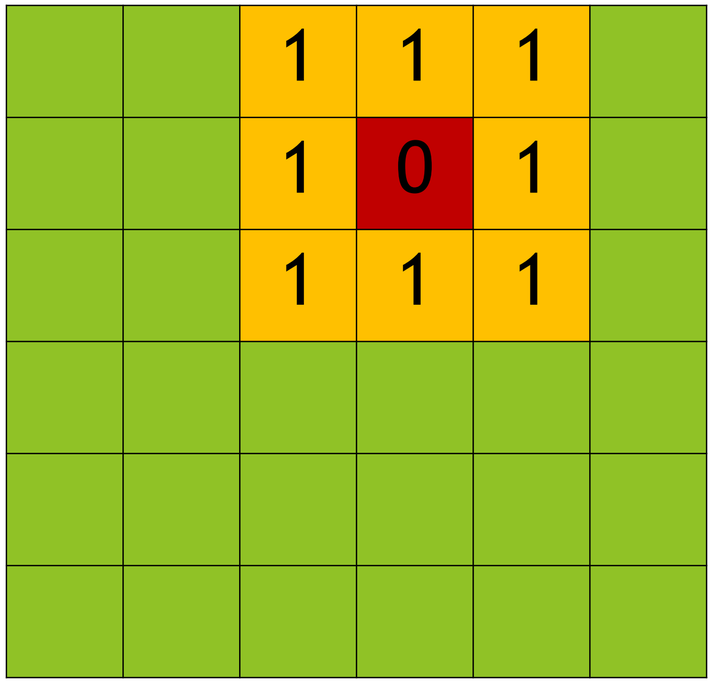
廣度優先搜尋

BFS



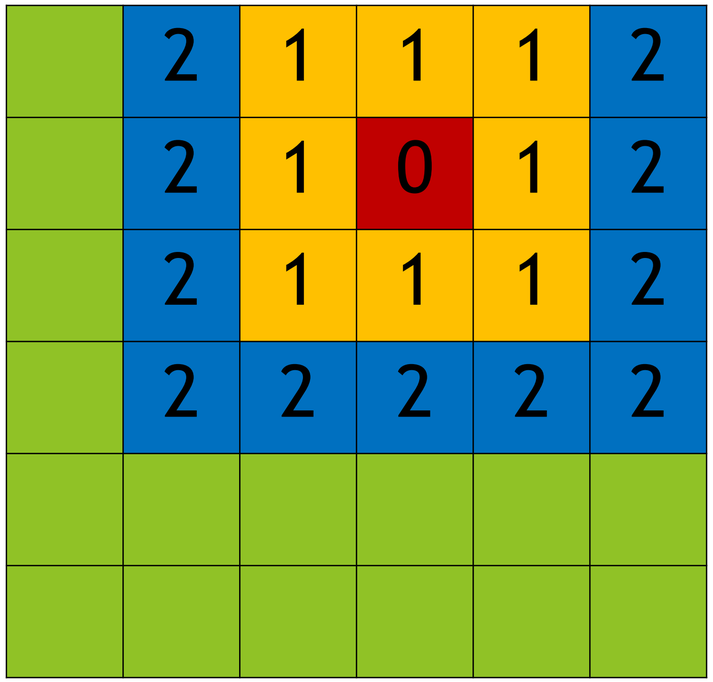
假設紅色方塊是起點

BFS



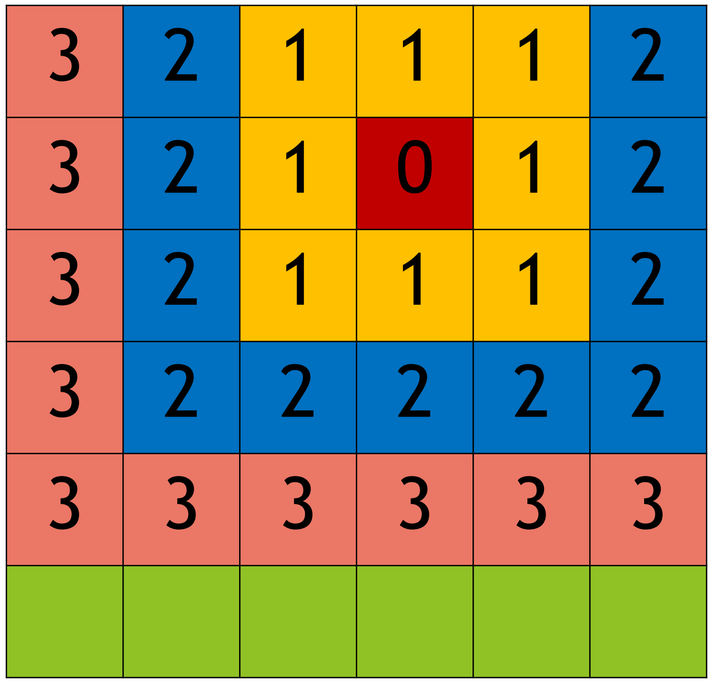
擴散紅色周圍的，並且標記起來

BFS



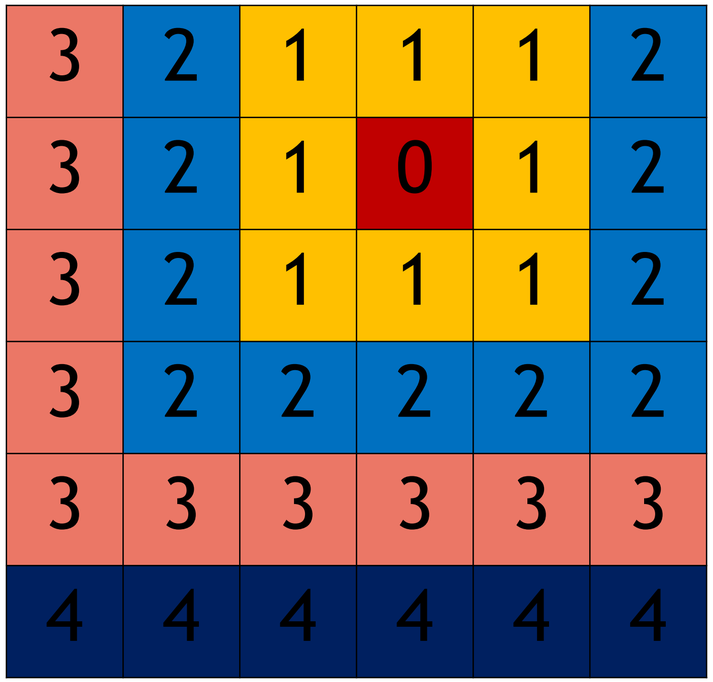
擴散紅色周圍的，並且標記起來

BFS



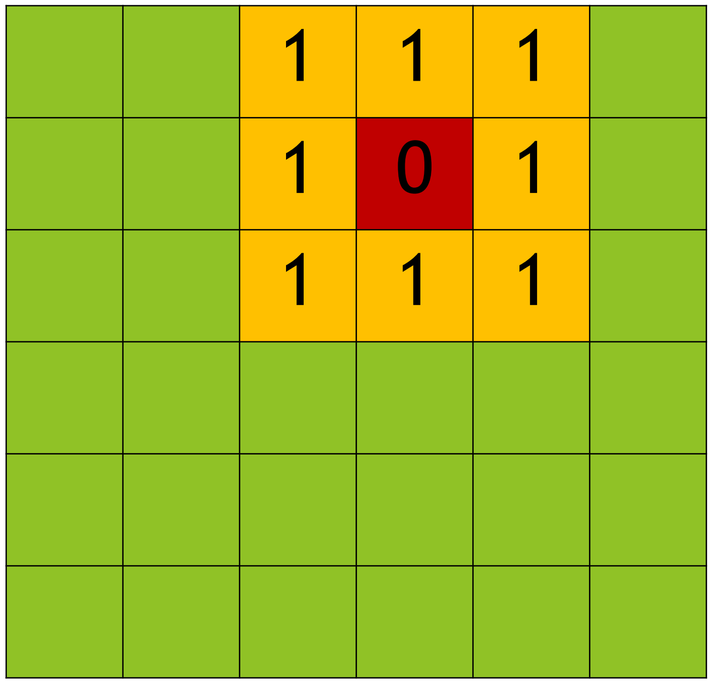
擴散紅色周圍的，並且標記起來

BFS



擴散紅色周圍的，並且標記起來

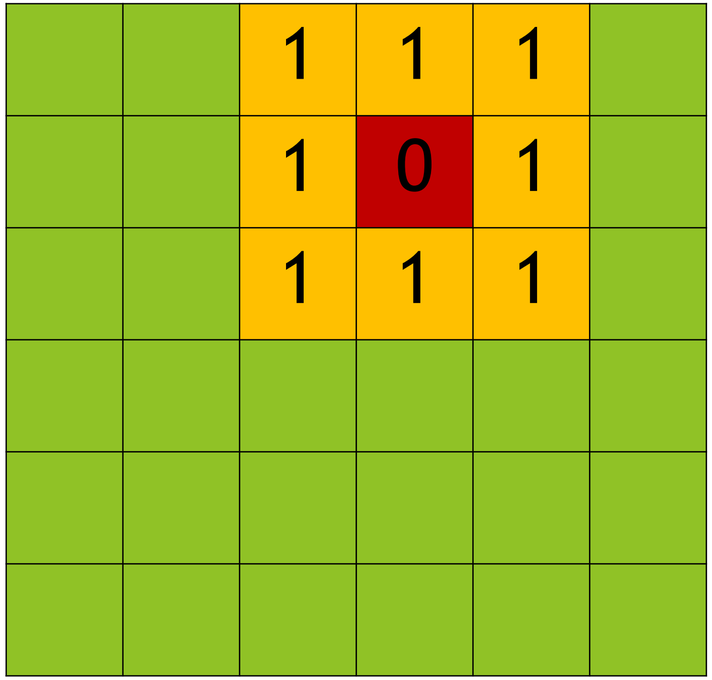
BFS



我們發現x, y的變化量都是1, 0, -1，
預先把變化量存進表格，就可以用迴圈完成囉！

```
if(x-1 >= 0) //visit (x-1 , y+0)
if(y-1 >= 0) //visit (x+0 , y-1)
if(x+1 <  n) //visit (x+1 , y+0)
if(y+1 <  n) //visit (x+0 , y+1)
```

BFS

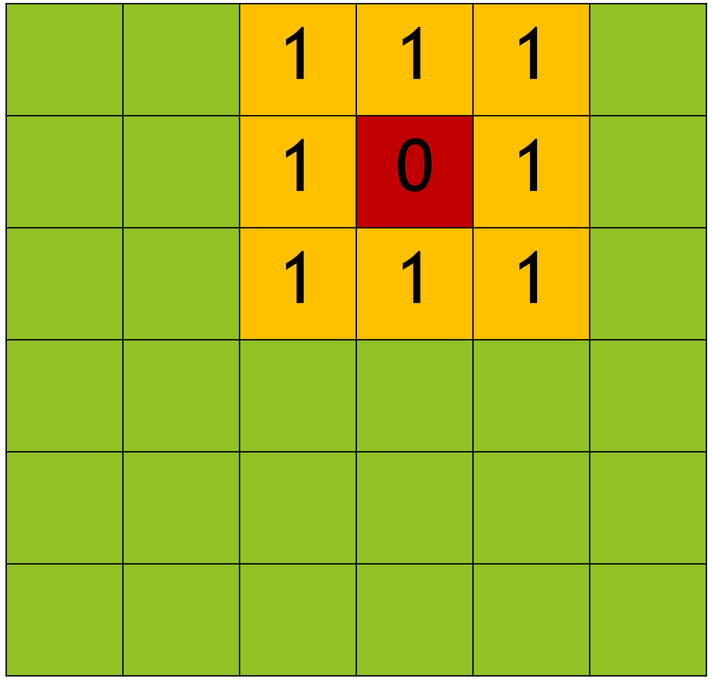


我們發現x, y的變化量都是1, 0, -1，
預先把變化量存進表格，就可以用迴圈完成囉！

```
if(x-1 >= 0) //visit (x-1 , y+0)
if(y-1 >= 0) //visit (x+0 , y-1)
if(x+1 <  n) //visit (x+1 , y+0)
if(y+1 <  n) //visit (x+0 , y+1)
```

```
int dx[]={-1,0,1,0};
int dy[]={0,-1,0,1};
```

BFS



我們發現x, y的變化量都是1, 0, -1，
預先把變化量存進表格，就可以用迴圈完成囉！

```
if(x-1 >= 0) //visit (x-1 , y+0)
if(y-1 >= 0) //visit (x+0 , y-1)
if(x+1 <  n) //visit (x+1 , y+0)
if(y+1 <  n) //visit (x+0 , y+1)
```

```
int dx[]={-1,0,1,0};
int dy[]={0,-1,0,1};
for(int i=0 ; i<4 ; ++i){
    int new_x = x + dx[i];//新的x座標
    int new_y = y + dy[i];//新的y座標
}
```

BFS

code : https://ideone.com/nTMGLf

```cpp
char table[5][5];//表格
bool vis[5][5];//看有沒有走過
bool check(int x , int y){
    if(x >= 0 && x < n && y >= 0 && y < m) return true;
    else return false;
}
void BFS(int stx , int sty){//(stx,sty)->起點
    queue< pair<int,int> > q;//代辦清單
    q.push({stx , sty});//把起點放進代辦清單
    vis[stx][sty] = true;//注意, 起點一開始就要記錄已經走過
    while(q.size()){//如果代辦清單還有東西
        int x = q.front().first;
        int y = q.front().second;
        q.pop();//從代辦清單取出後要pop
        for(int i=0 ; i<4 ; ++i){
            int xx = x + dx[i];
            int yy = y + dy[i];
            if(check(xx , yy) && vis[xx][yy]){
                q.push({xx,yy});
                vis[xx][yy] = true;
                //放進代辦清單就要當作走過, 才不會重複把某個位置放入代辦清單
            }
        }
    }
}
```

試題演練

BFS 用途:

- 連通塊 ： 處理大小、連通性
- 最短距離 ： 無權圖的最短距離
- 搜索 ： 對未知的圖進行搜索、探勘

試題演練

https://leetcode.com/problems/max-area-of-island/

695. Max Area of Island

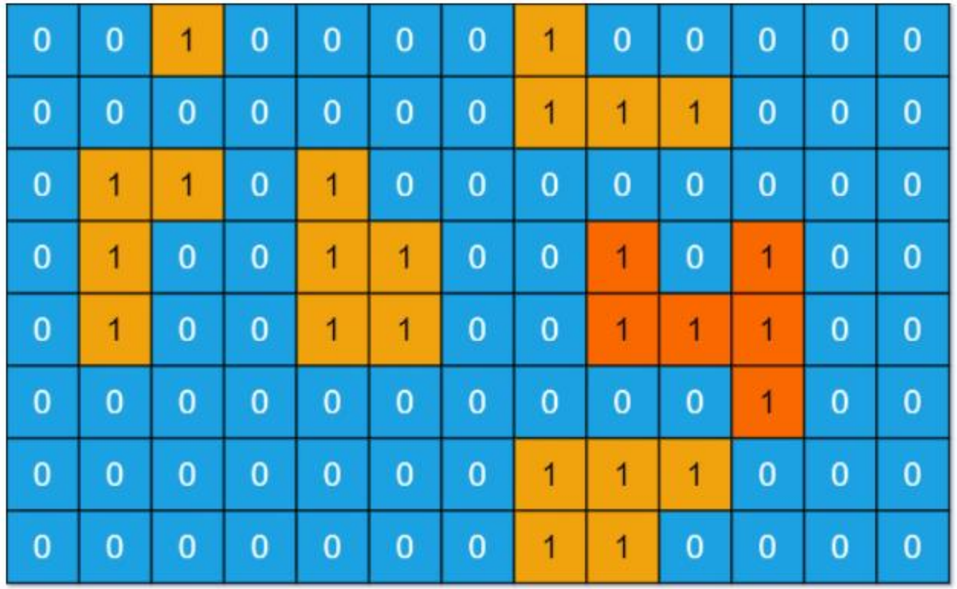
Medium 3683 114 Add to List Share

You are given an `m x n` binary matrix `grid`. An island is a group of `1`'s (representing land) connected **4-directionally** (horizontal or vertical.) You may assume all four edges of the grid are surrounded by water.

The **area** of an island is the number of cells with a value `1` in the island.

Return *the maximum **area** of an island in* `grid`. If there is no island, return `0`.

Example 1:



試題演練

https://ideone.com/ud17uT

```
class Solution {
public:
    #define pii pair<int,int>
    #define F first
    #define S second
    int maxAreaOfIsland(vector<vector<int>>& grid) {
        int dx[]={0,0,1,-1};
        int dy[]={1,-1,0,0};
        const int n = grid.size();
        const int m = grid[0].size();
        vector< vector< bool > > vis(n , vector<bool>(m , false));
        int ans=0;
        for(int i=0 ; i<n ; ++i){
            for(int j=0 ; j<m ; ++j){
                if(!vis[i][j] && grid[i][j]){
                    int sx = i , sy = j;
                    queue<pii> q;
                    q.push({sx , sy});
                    vis[sx][sy] = true;
                    int cnt = 0 ;
                    while(q.size()){
                        int x = q.front().F;
                        int y = q.front().S;
                        q.pop();
                        ++cnt;
                        for(int i=0 ; i<4 ; ++i){
                            int xx = x + dx[i];
                            int yy = y + dy[i];
                            if(xx >= 0 && xx < n  && yy >= 0 && yy < m && !vis[xx][yy] && grid[xx][yy]){
                                q.push({xx,yy});
                                vis[xx][yy] = true;
                            }
                        }
                    }
                    ans = max(ans , cnt);
                }
            }
        }
        return ans;
    }
};
```

試題演練

- 這題就是在考BFS的連通塊處理、計算大小。
- 還有很多類似的題目值得練習，像是:
  - 連通塊的數量(有幾塊連通塊）)->易
  - 連通塊的周長(幾個方塊在連通塊的邊緣）)->中
  - 連通塊的轉角數量(連通塊的外圍有幾個九十度的轉角)->難

試題演練

1091. Shortest Path in Binary Matrix

Medium 1325 82 Add to List Share

Given an `n x n` binary matrix `grid`, return *the length of the shortest **clear path** in the matrix*. If there is no clear path, return `-1`.

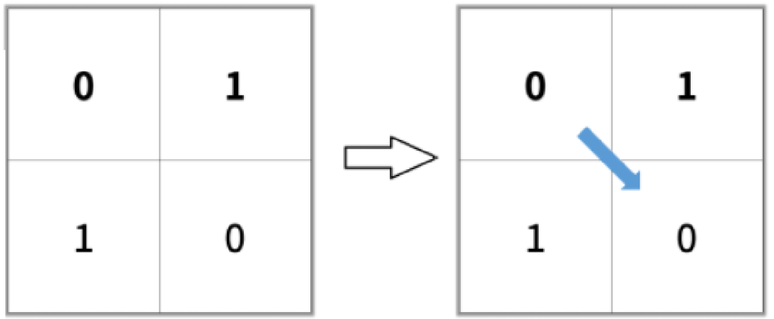
A **clear path** in a binary matrix is a path from the **top-left** cell (i.e., `(0, 0)`) to the **bottom-right** cell (i.e., `(n - 1, n - 1)`) such that:

- All the visited cells of the path are `0`.
- All the adjacent cells of the path are **8-directionally** connected (i.e., they are different and they share an edge or a corner).

The **length of a clear path** is the number of visited cells of this path.

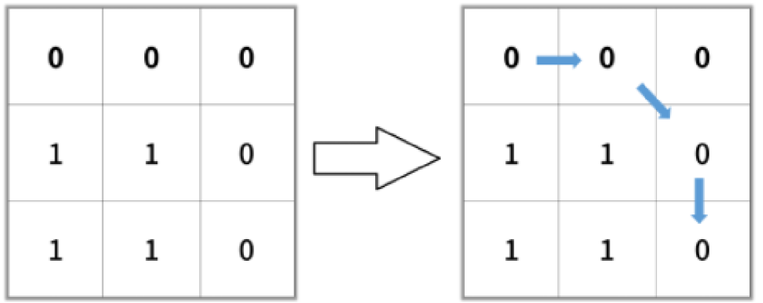
https://leetcode.com/problems/shortest-path-in-binary-matrix/

Example 1:




```
Input: grid = [[0,1],[1,0]]
Output: 2
```

ample 2:




```
Input: grid = [[0,0,0],[1,1,0],[1,1,0]]
Output: 4
```

試題演練

```cpp
class Solution {
public:
    #define pii pair<int,int>
    #define F first
    #define S second
    int shortestPathBinaryMatrix(vector<vector<int>>& grid) {
        int dx[]={1,1,1,0,-1,-1,-1,0};
        int dy[]={1,0,-1,-1,-1,0,1,1};
        const int n = grid.size();
        const int m = grid[0].size();
        vector< vector< bool > > vis(n , vector<bool>(m , false));
        vector< vector< int > > dis(n , vector<int>(m,-1));
        queue<pii> q;
        if(grid[0][0] == 0) q.push({0,0}) , dis[0][0] = vis[0][0] = 1;
        while(q.size()){
            int x = q.front().F;
            int y = q.front().S;
            q.pop();
            for(int i=0 ; i<8 ; ++i){
                int new_x = x + dx[i];
                int new_y = y + dy[i];
                if(new_x >= 0 && new_x < n && new_y >= 0 && new_y < m && !vis[new_x][new_y] && grid[new_x][new_y] == 0){
                    q.push({new_x , new_y});
                    dis[new_x][new_y] = dis[x][y] + 1;
                    vis[new_x][new_y] = true;
                }
            }
        }
        return dis[n-1][m-1];
    }
};
```

https://ideone.com/QCfb9x

試題演練

- 這題則是在考BFS求無權圖的最短路徑
- 如果邊有權重的話，要用較難的演算法來求最短路
  - $Dijkstra$、$Floyd-Warshall$

試題演練

841. Keys and Rooms

Medium 1994 138 Add to List Share

There are `N` rooms and you start in room `0`. Each room has a distinct number in `0, 1, 2, ..., N-1`, and each room may have some keys to access the next room.

Formally, each room `i` has a list of keys `rooms[i]`, and each key `rooms[i][j]` is an integer in `[0, 1, ..., N-1]` where `N = rooms.length`. A key `rooms[i][j] = v` opens the room with number `v`.

Initially, all the rooms start locked (except for room `0`).

You can walk back and forth between rooms freely.

Return `true` if and only if you can enter every room.

Example 1:

```
Input: [[1],[2],[3],[]]
Output: true
Explanation:
We start in room 0, and pick up key 1.
We then go to room 1, and pick up key 2.
We then go to room 2, and pick up key 3.
We then go to room 3.  Since we were able to go to every room, we return true.
```

https://leetcode.com/problems/keys-and-rooms/

試題演練

```cpp
class Solution {
public:
    bool canVisitAllRooms(vector<vector<int>>& rooms) {
        const int n = rooms.size();
        vector<bool> vis(n,false);
        queue<int> q;
        q.push(0);
        vis[0] = 1;
        int cnt = 0 ;
        while(q.size()){
            int now = q.front() ; q.pop();
            ++cnt;
            for(int &key : rooms[now]){
                if(!vis[key]){
                    q.push(key);
                    vis[key] = 1;
                }
            }
        }
        return cnt == n;
    }
};
```

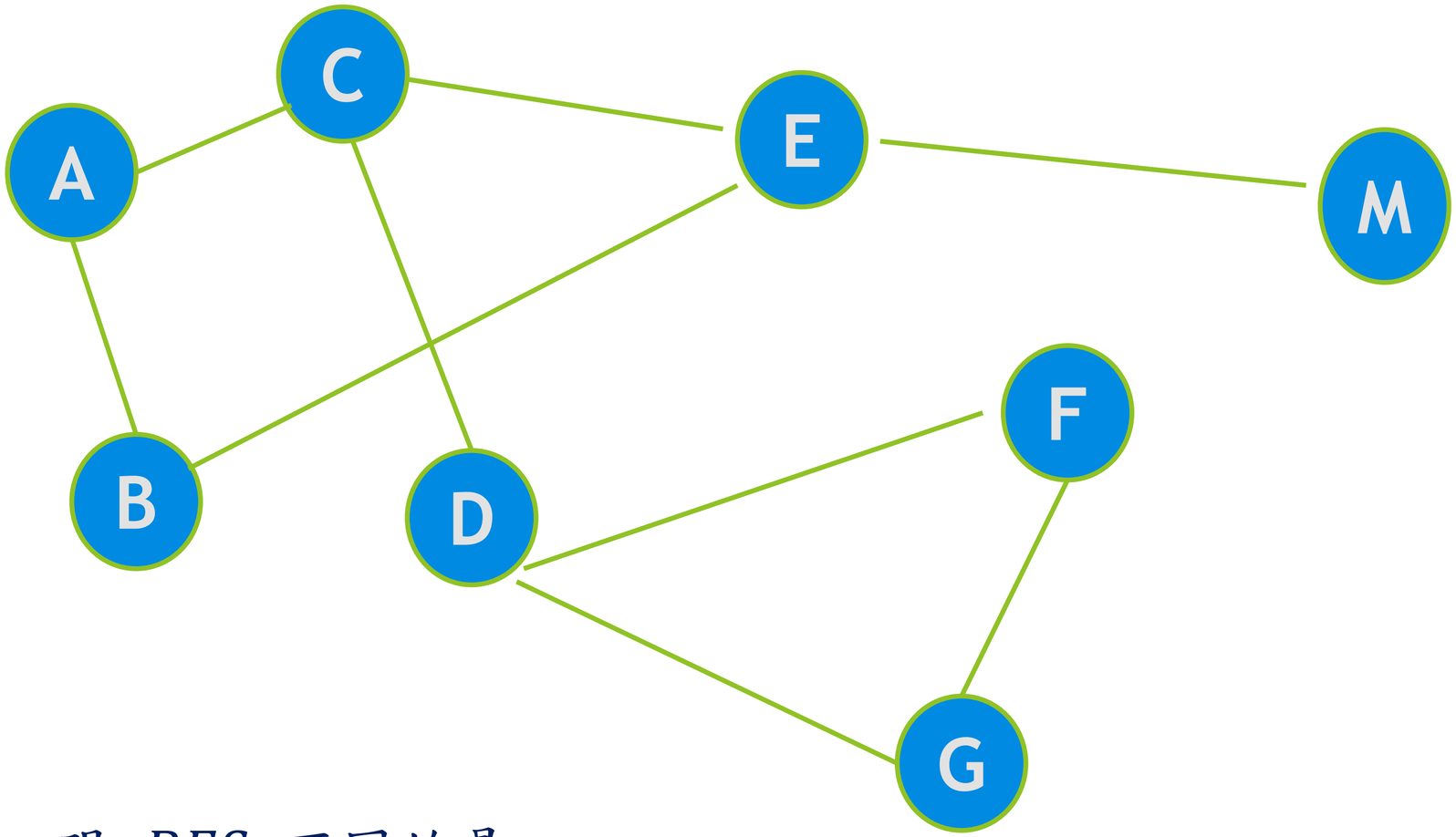
試題演練

- 這題在考隱式圖、未知圖的搜索、遍歷
- 所以我們可以知道，圖論的題目不一定會給圖
- 但也通常是比較難的題目了…

圖的遍歷– *DFS*

深度優先搜尋

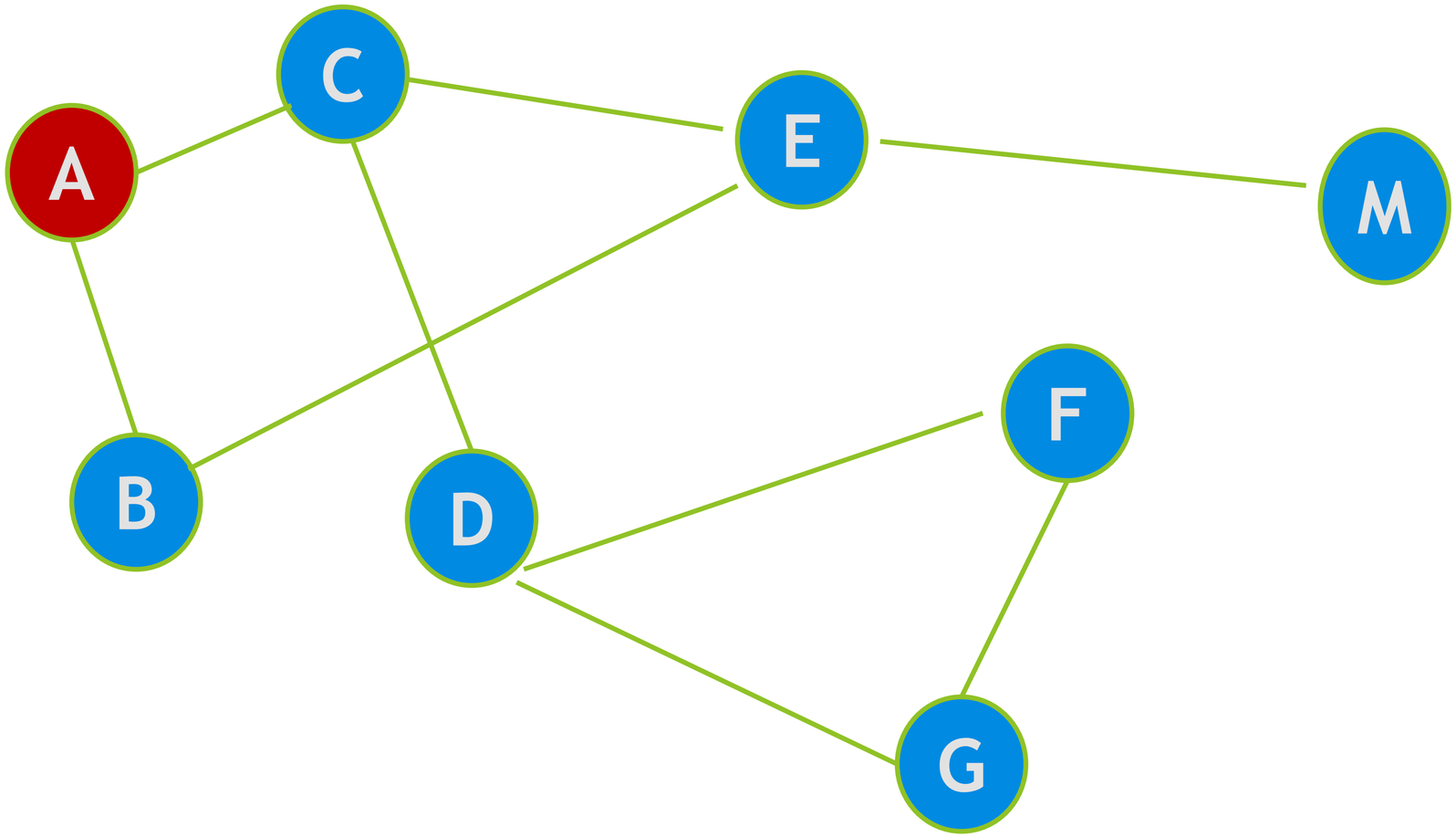
DFS



跟 BFS 不同的是，
BFS 先把周圍的路記起來，然後往下走，
DFS 則是看到路就先往下走。

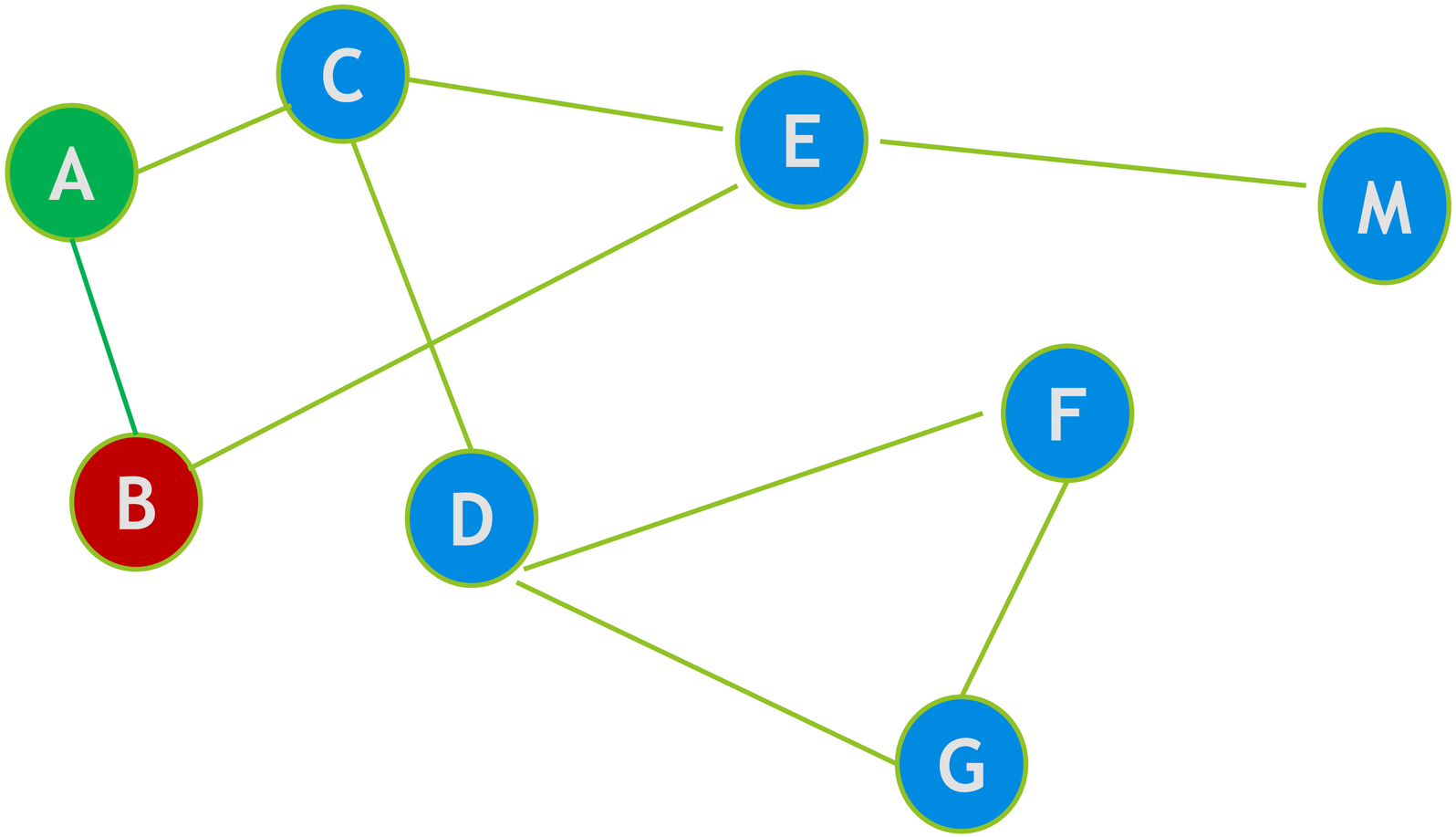
DFS

假設 A 是起點。



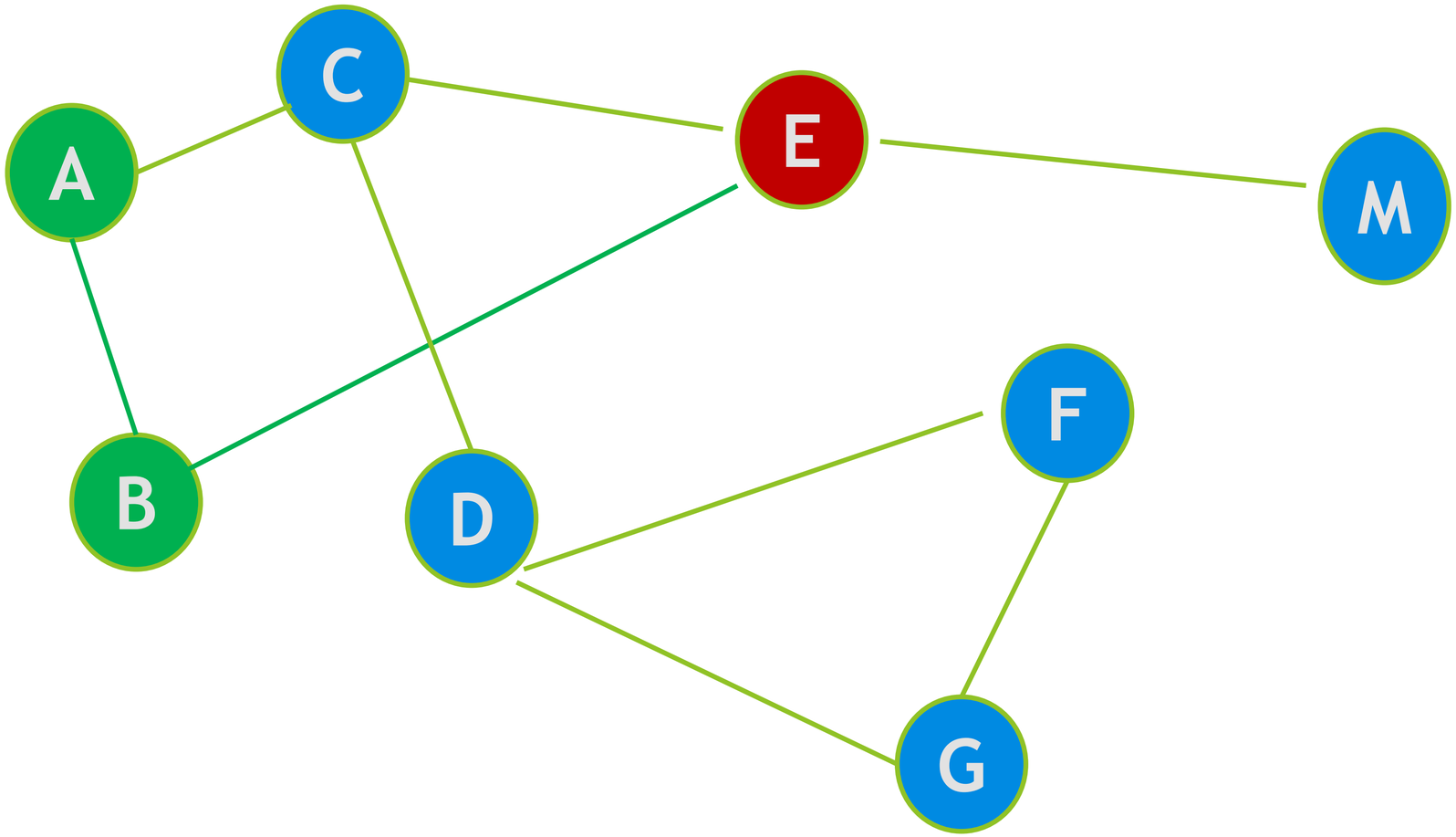
DFS

看到 B 先往 B 走



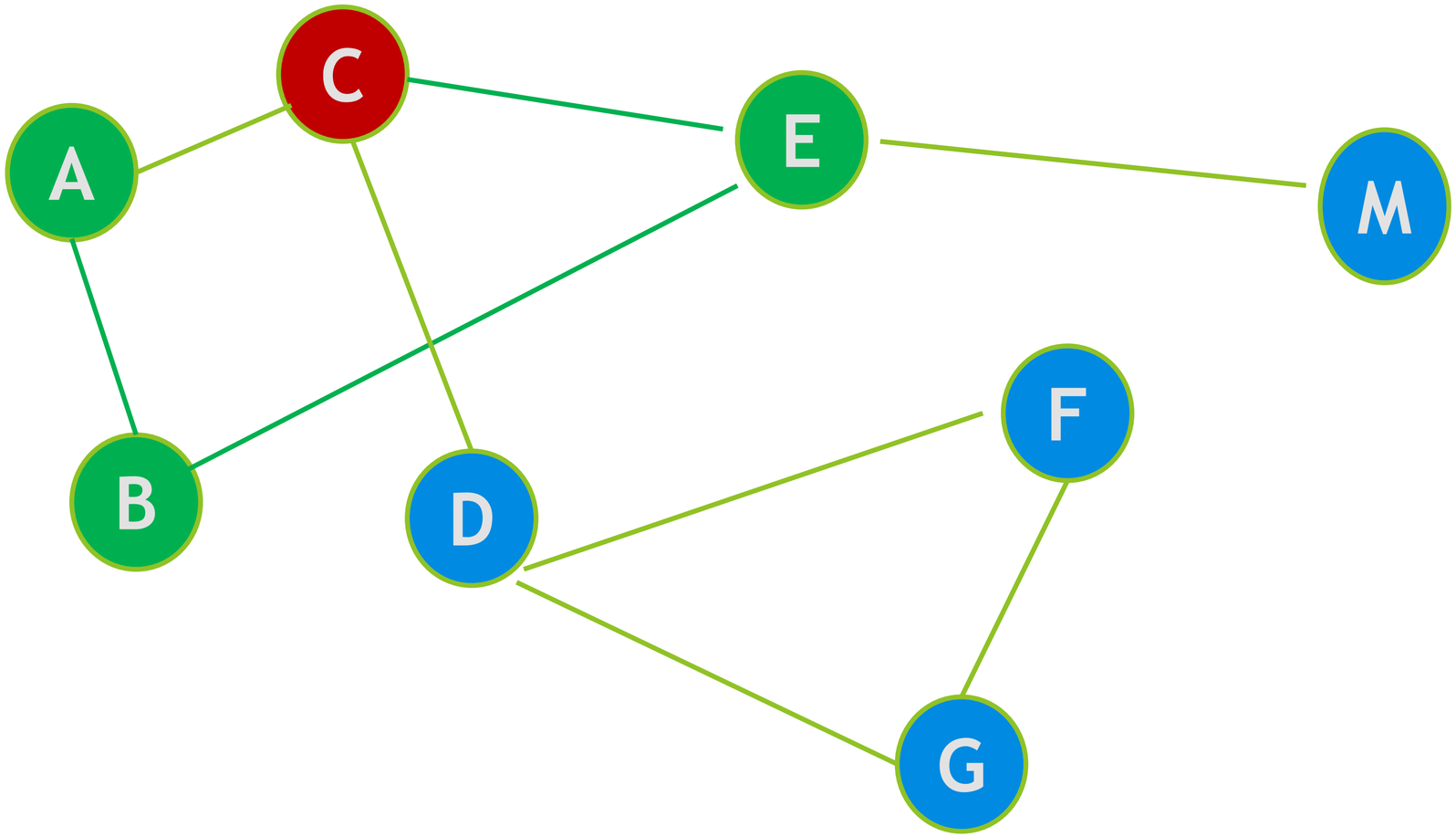
DFS

不斷往下走



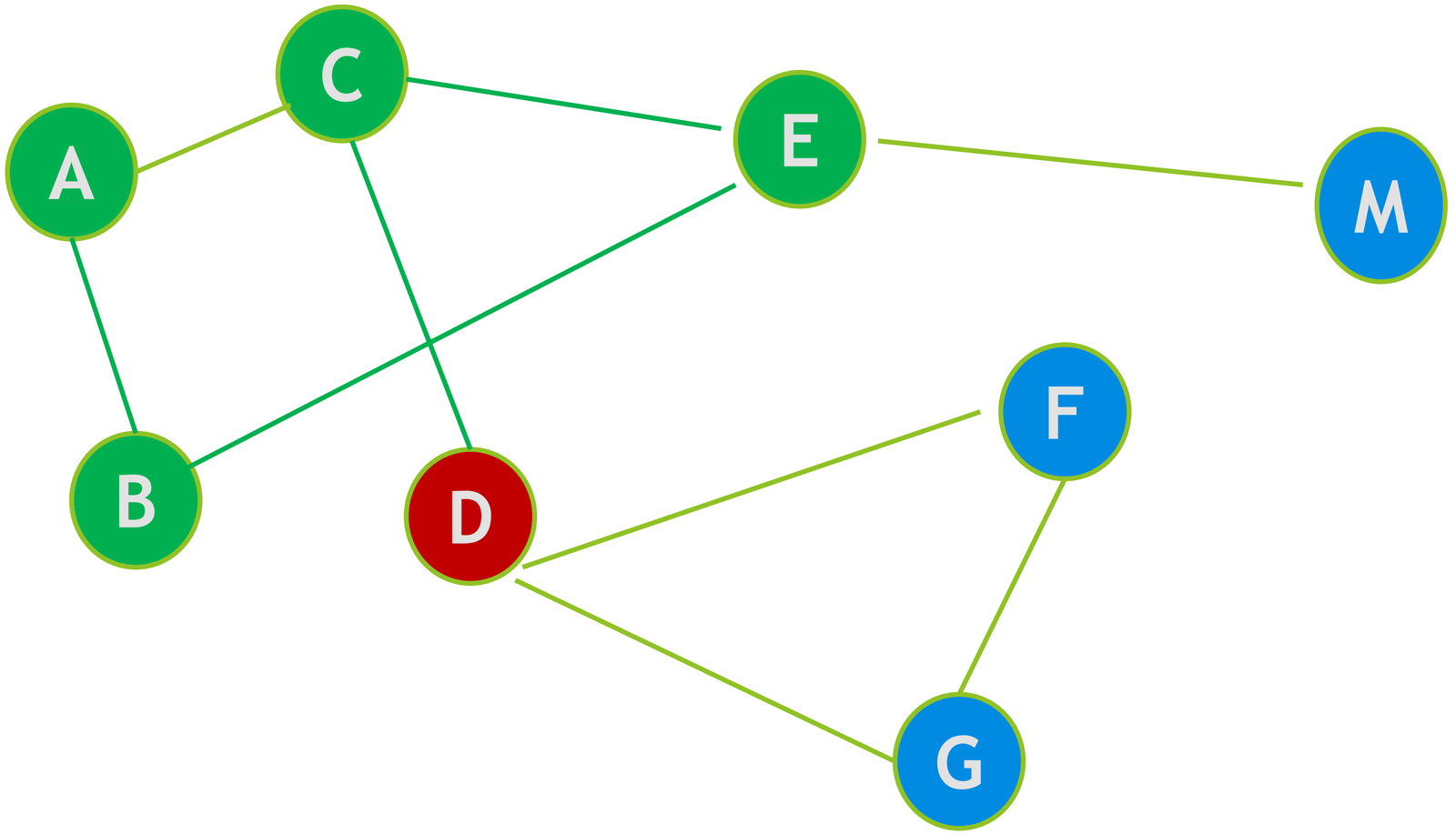
DFS

不斷往下走



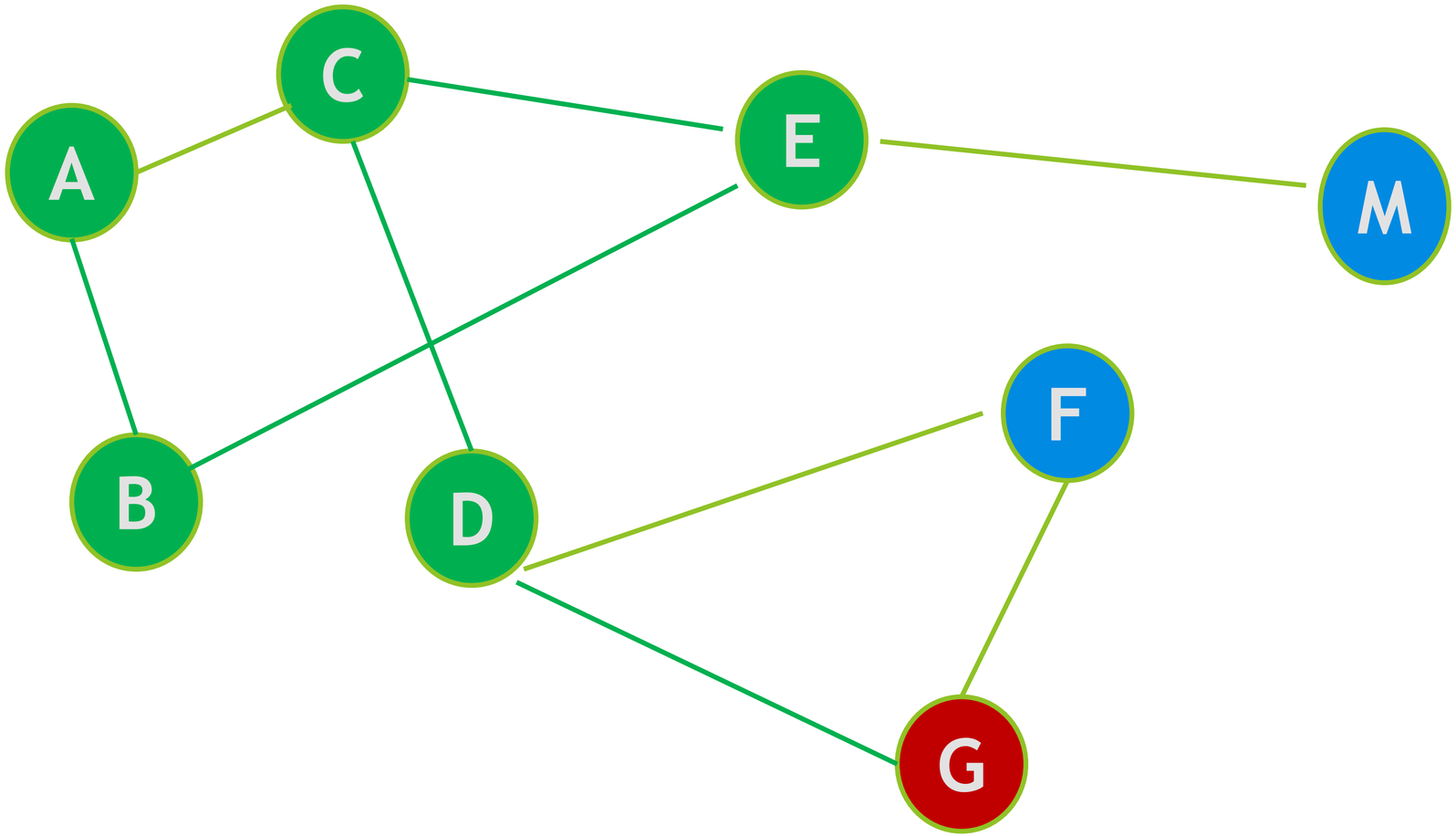
DFS

不斷往下走



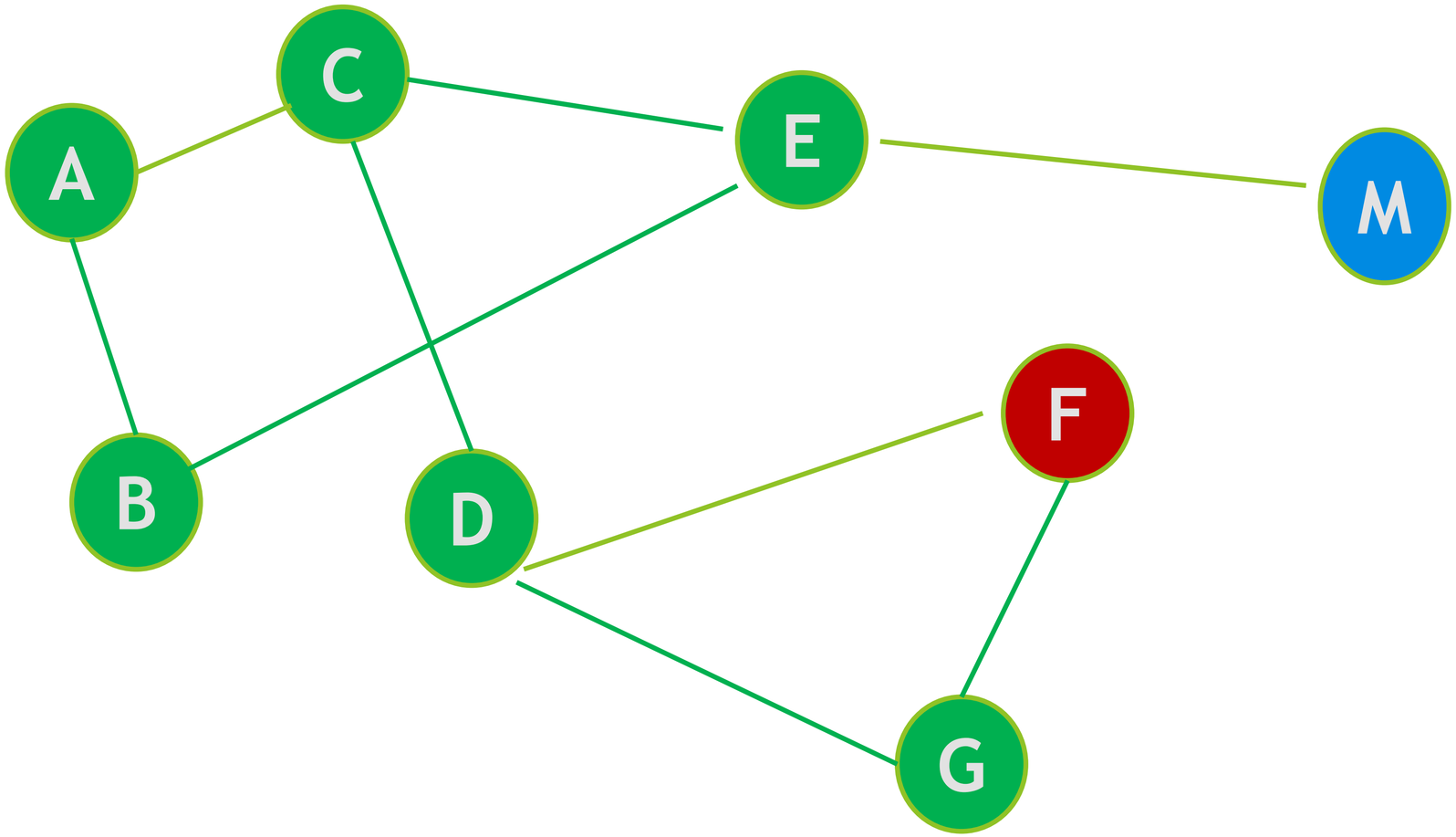
DFS

不斷往下走



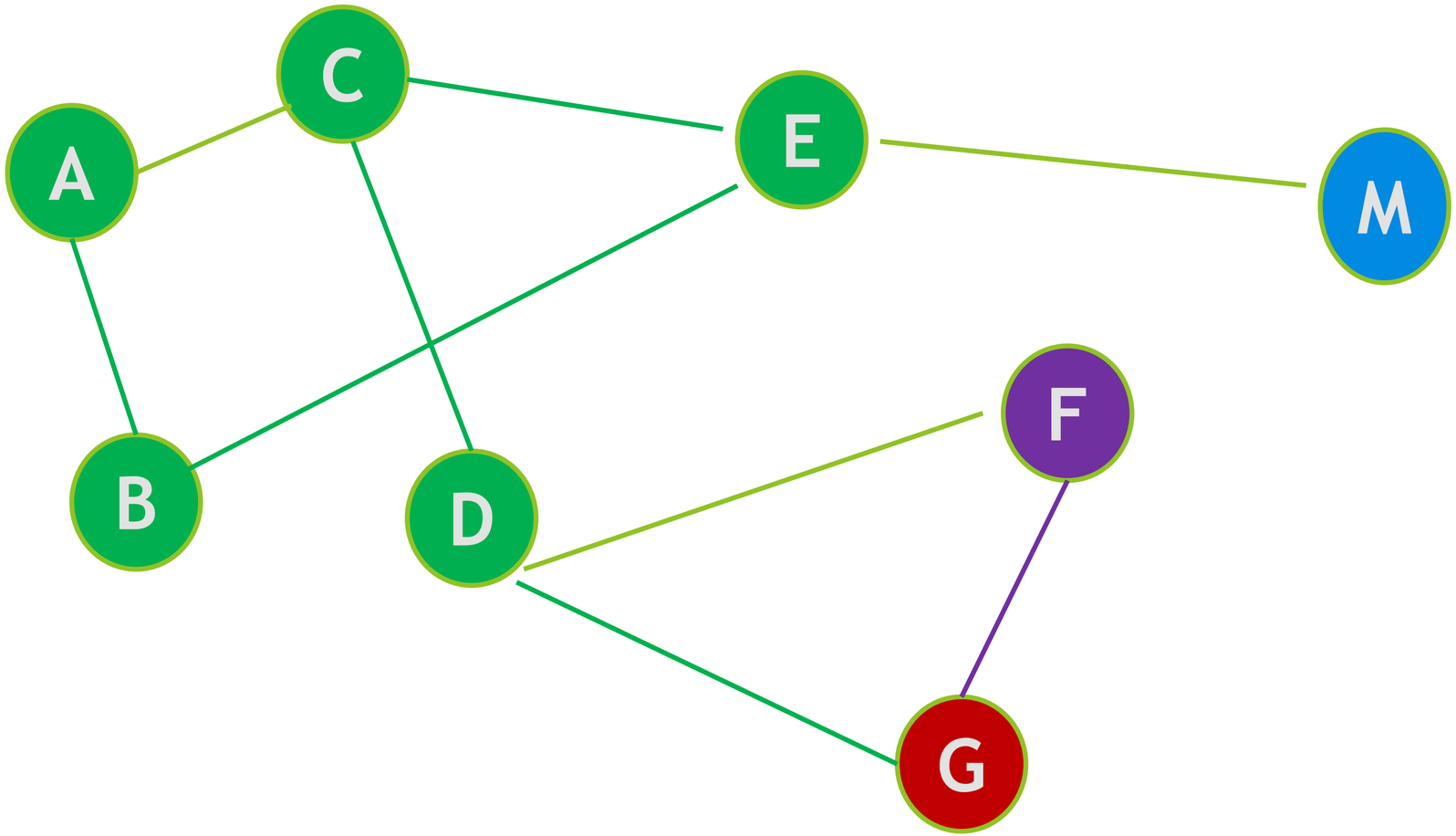
DFS

不斷往下走



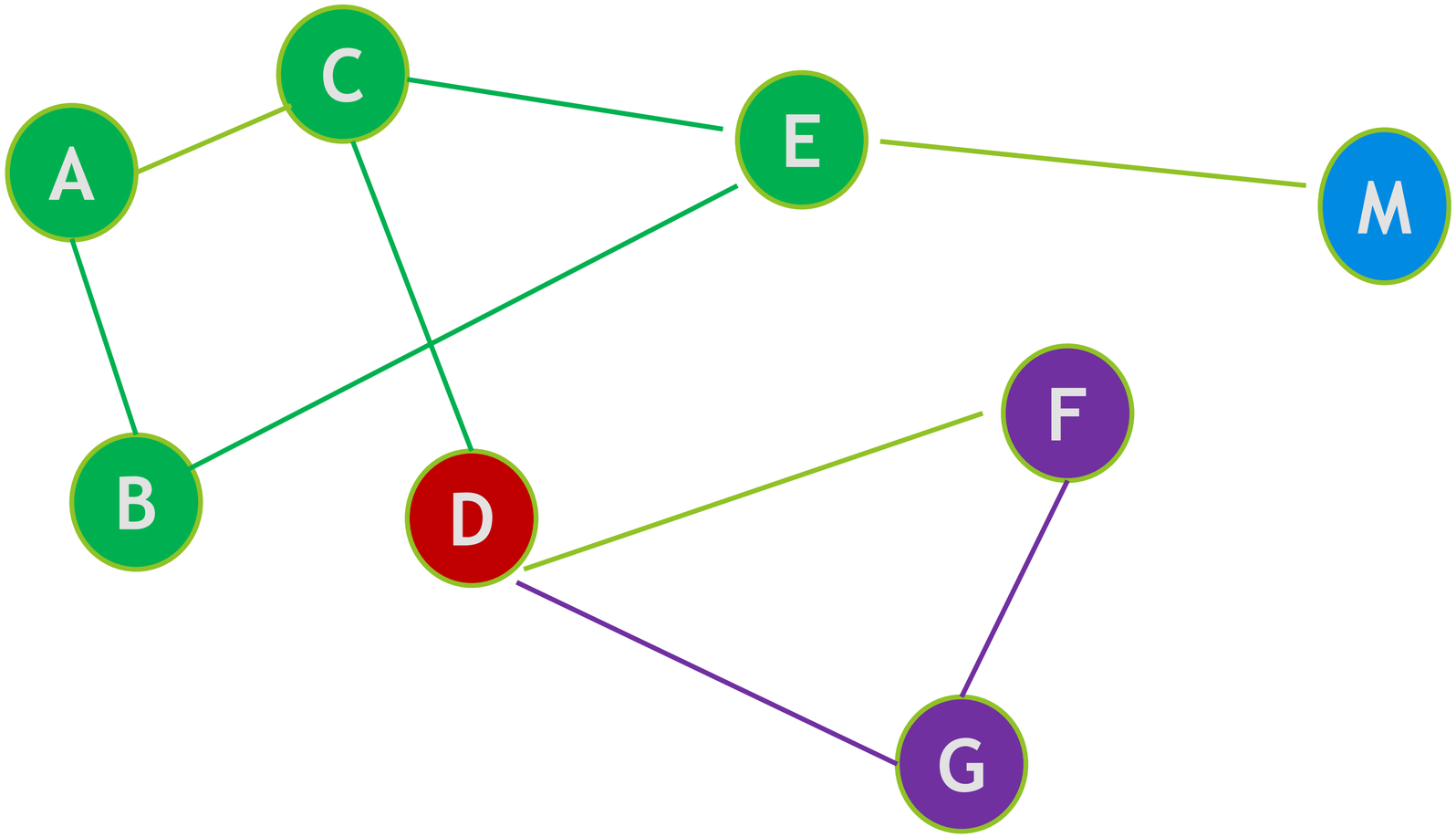
DFS

無路可走，返回 G



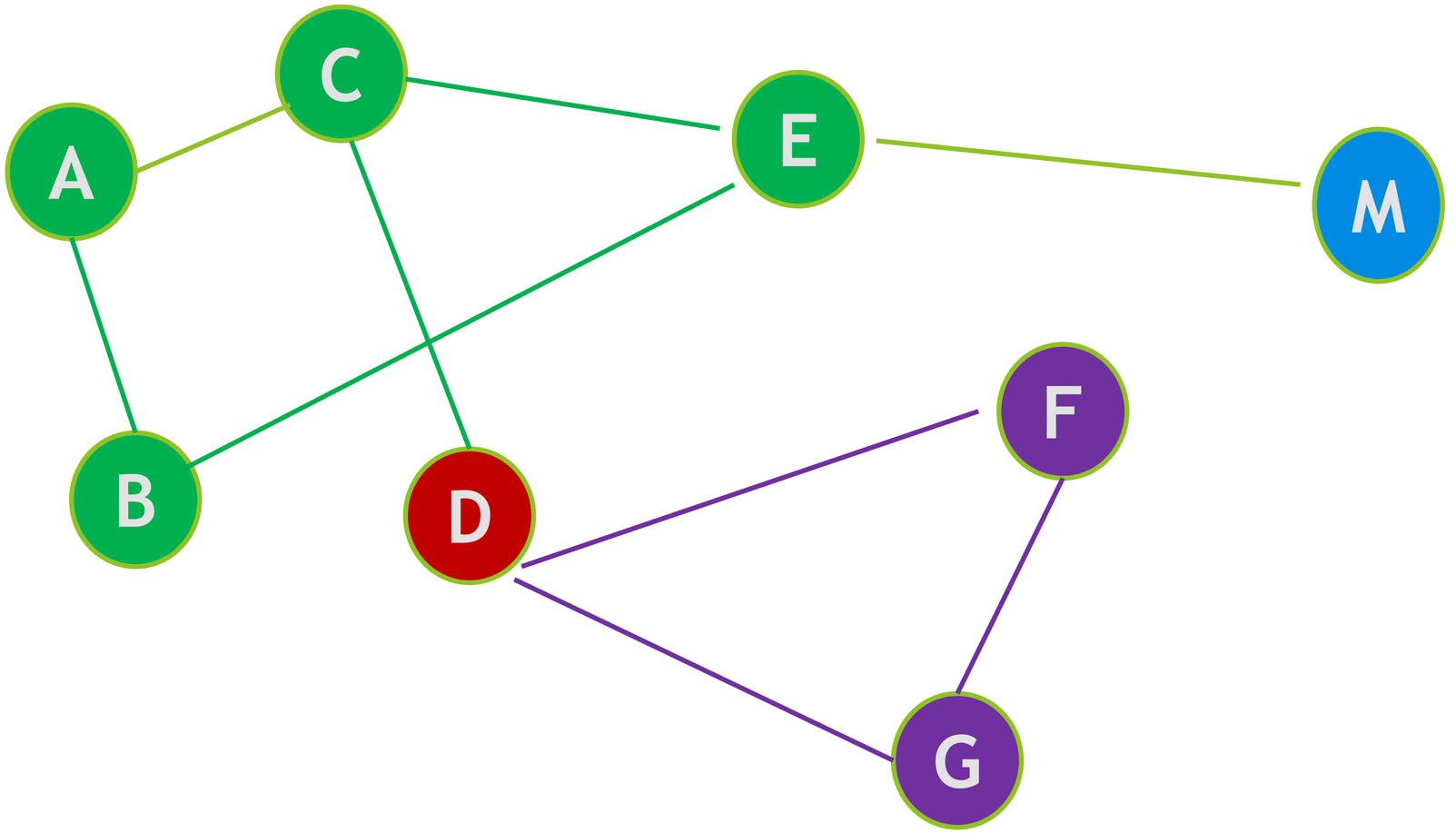
DFS

無路可走，返回 D



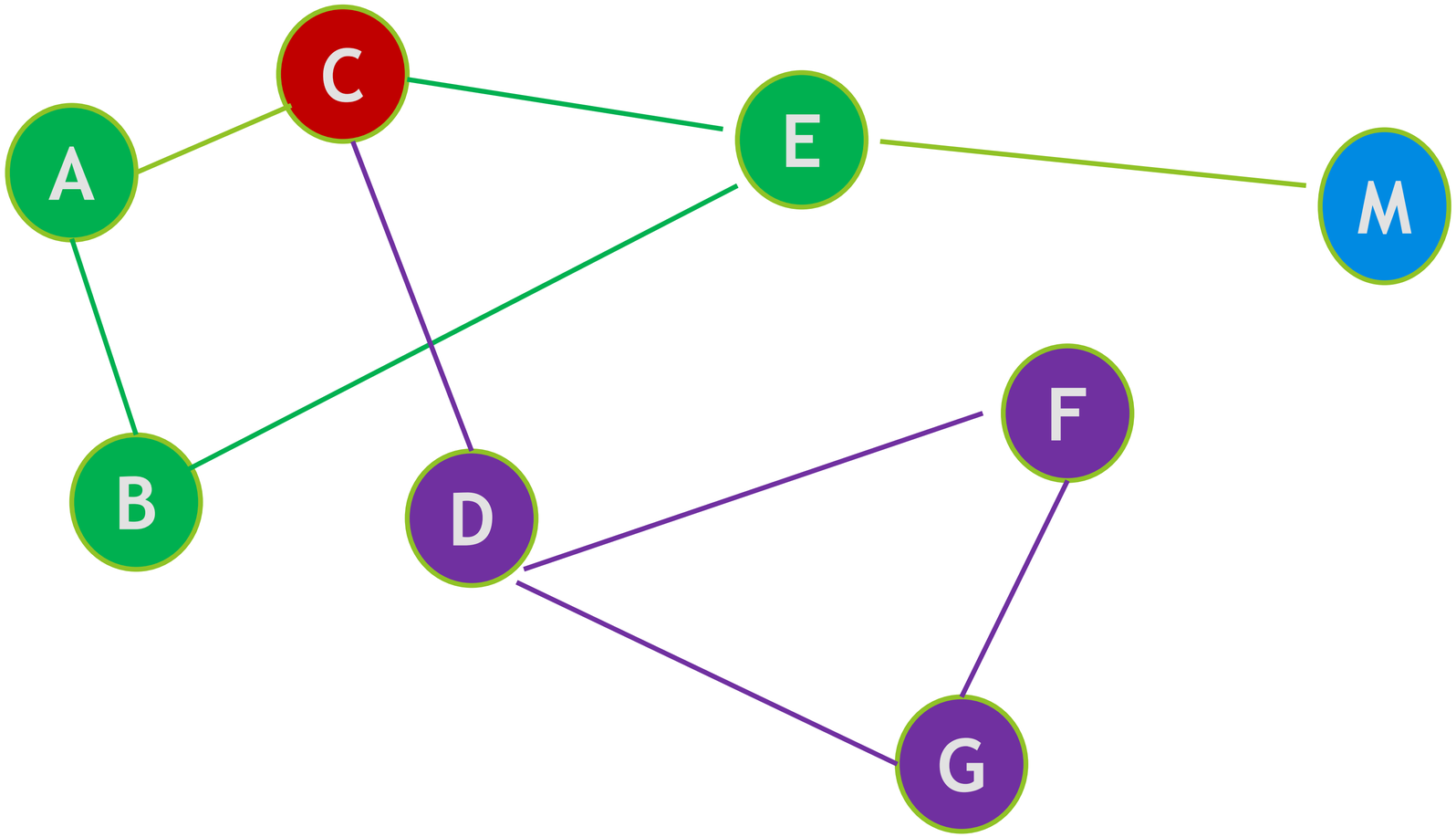
DFS

試圖往F走，但已經走過



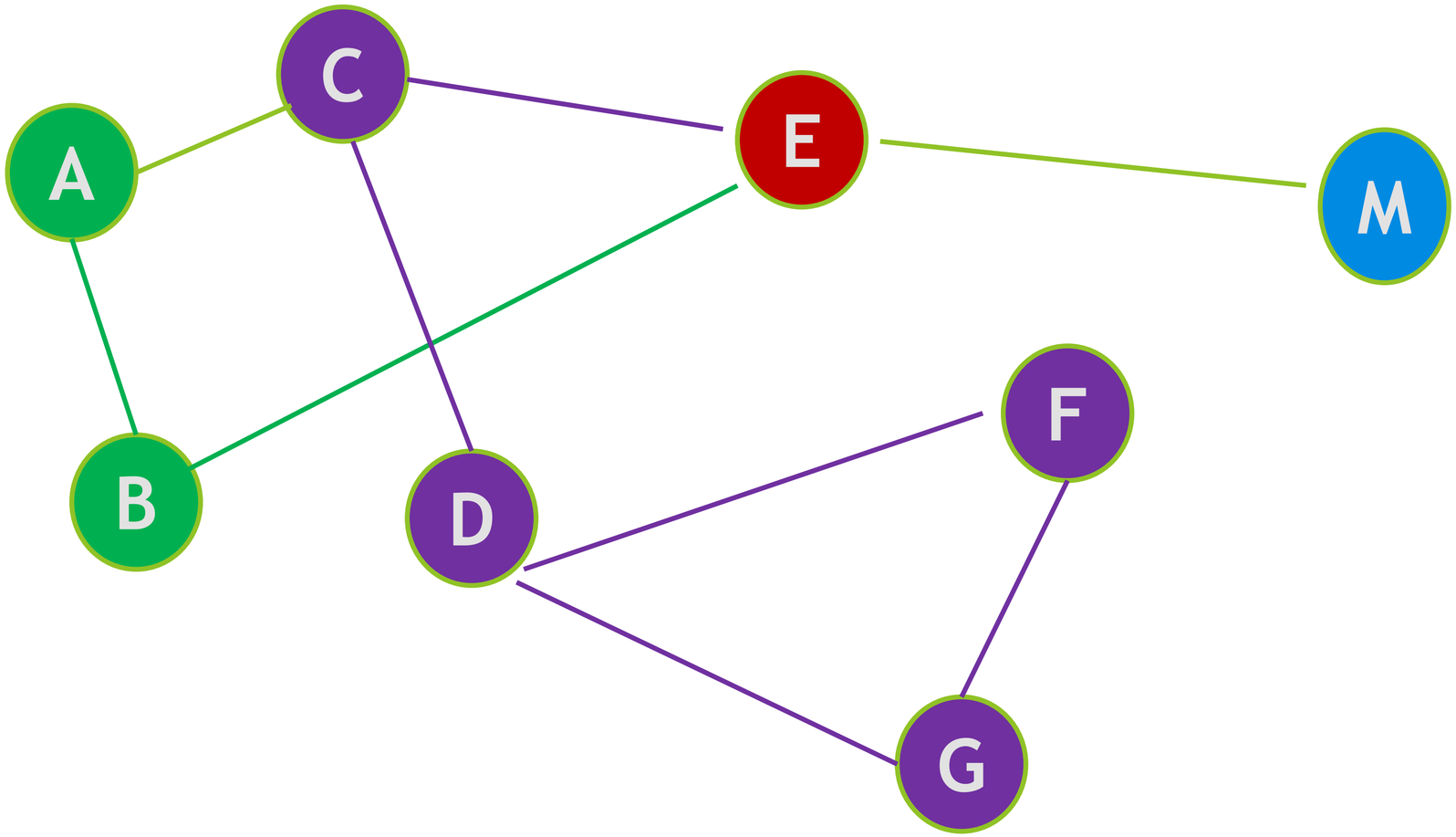
DFS

無路可走，返回 C



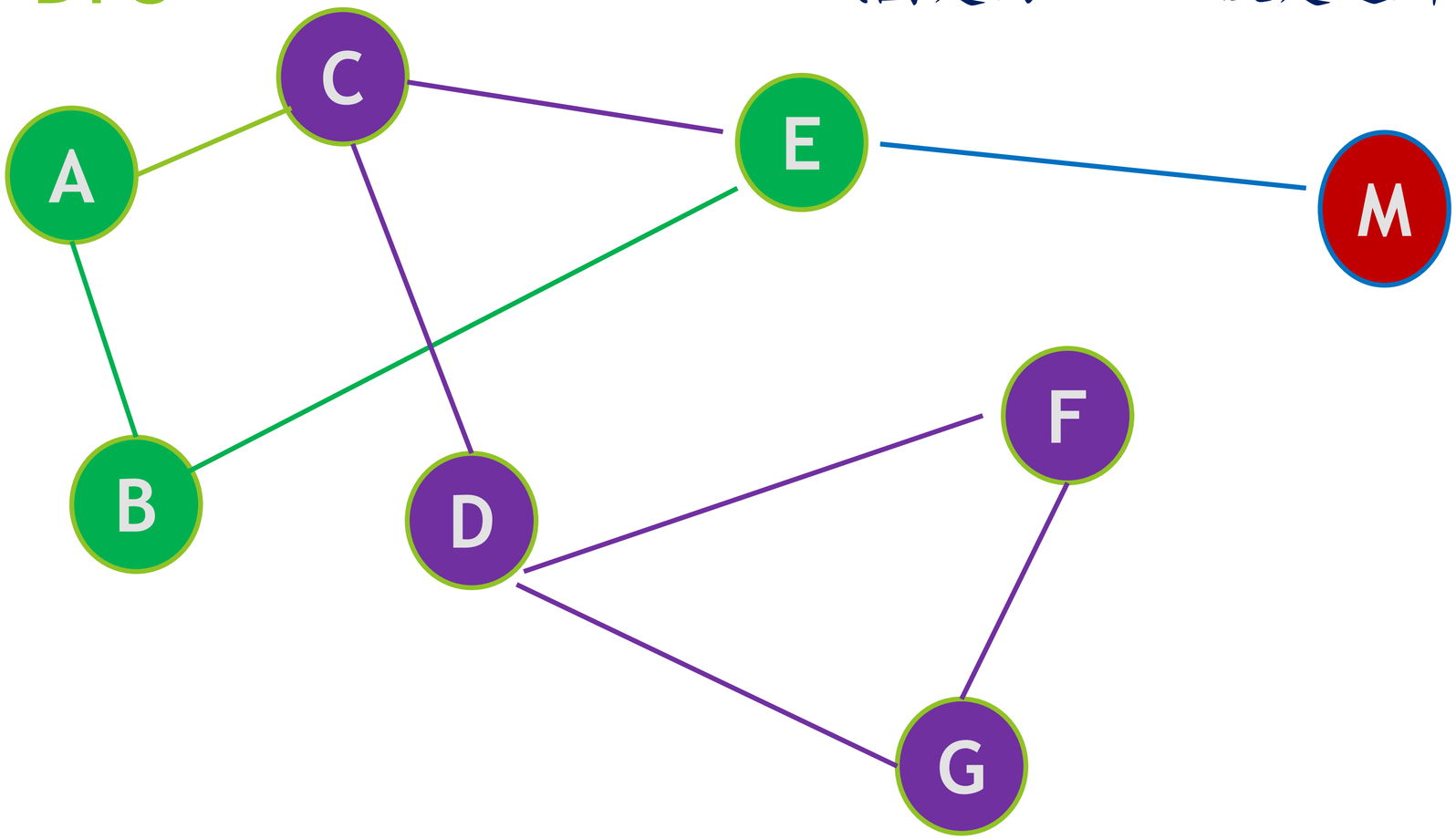
DFS

無路可走，返回 E



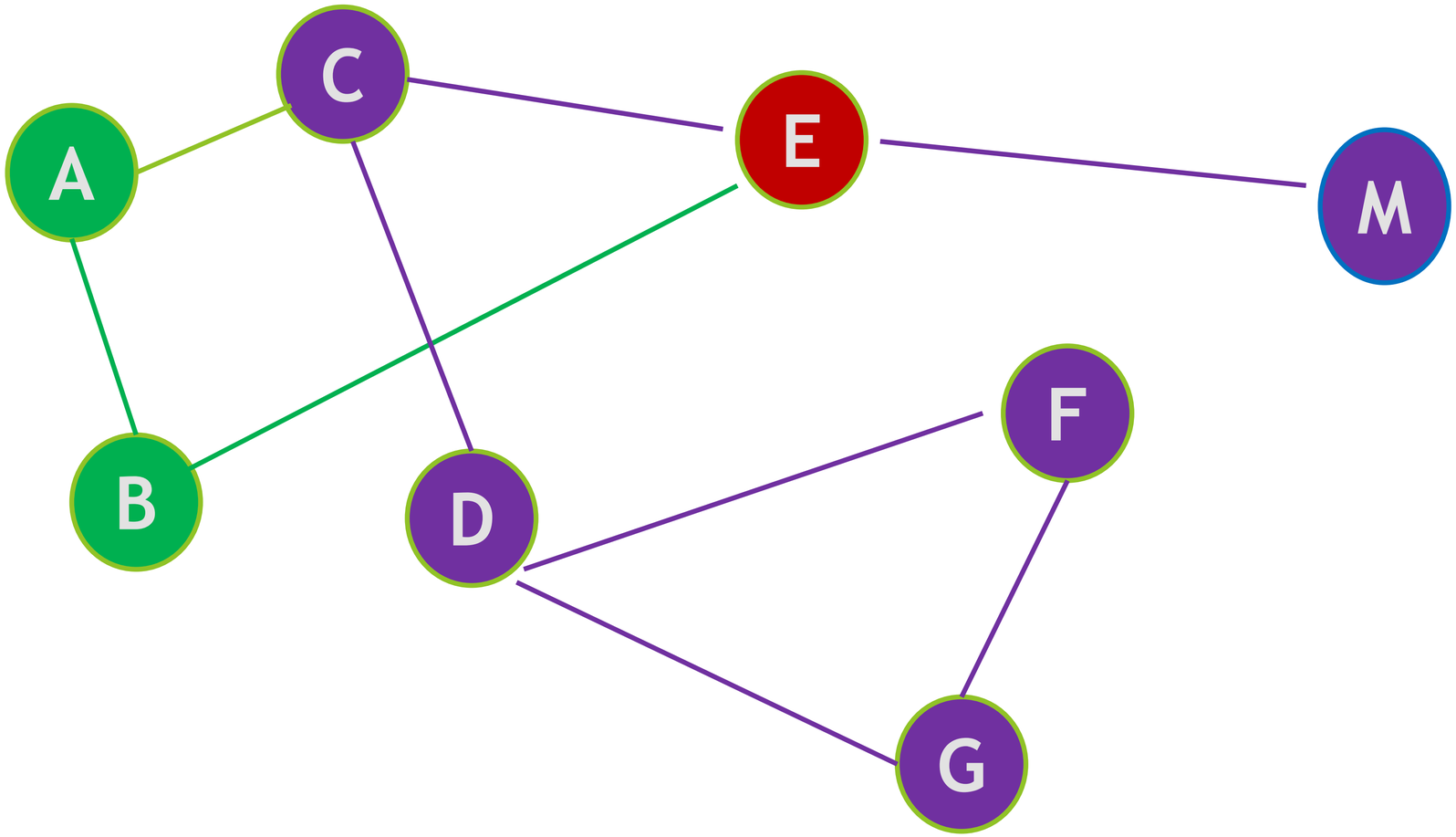
DFS

試圖走向 M ，沒走過所以往M走



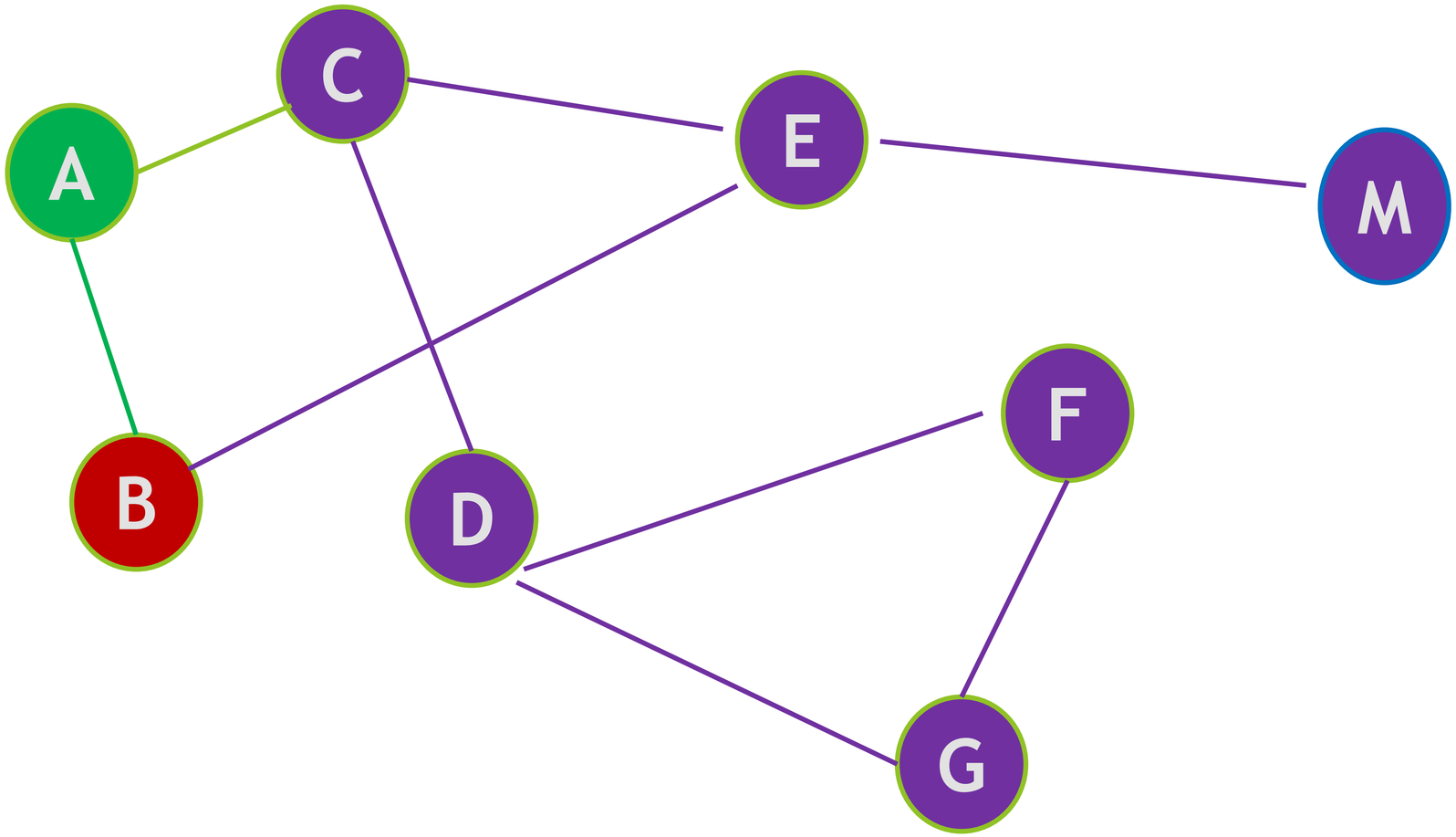
DFS

無路可走，返回 E



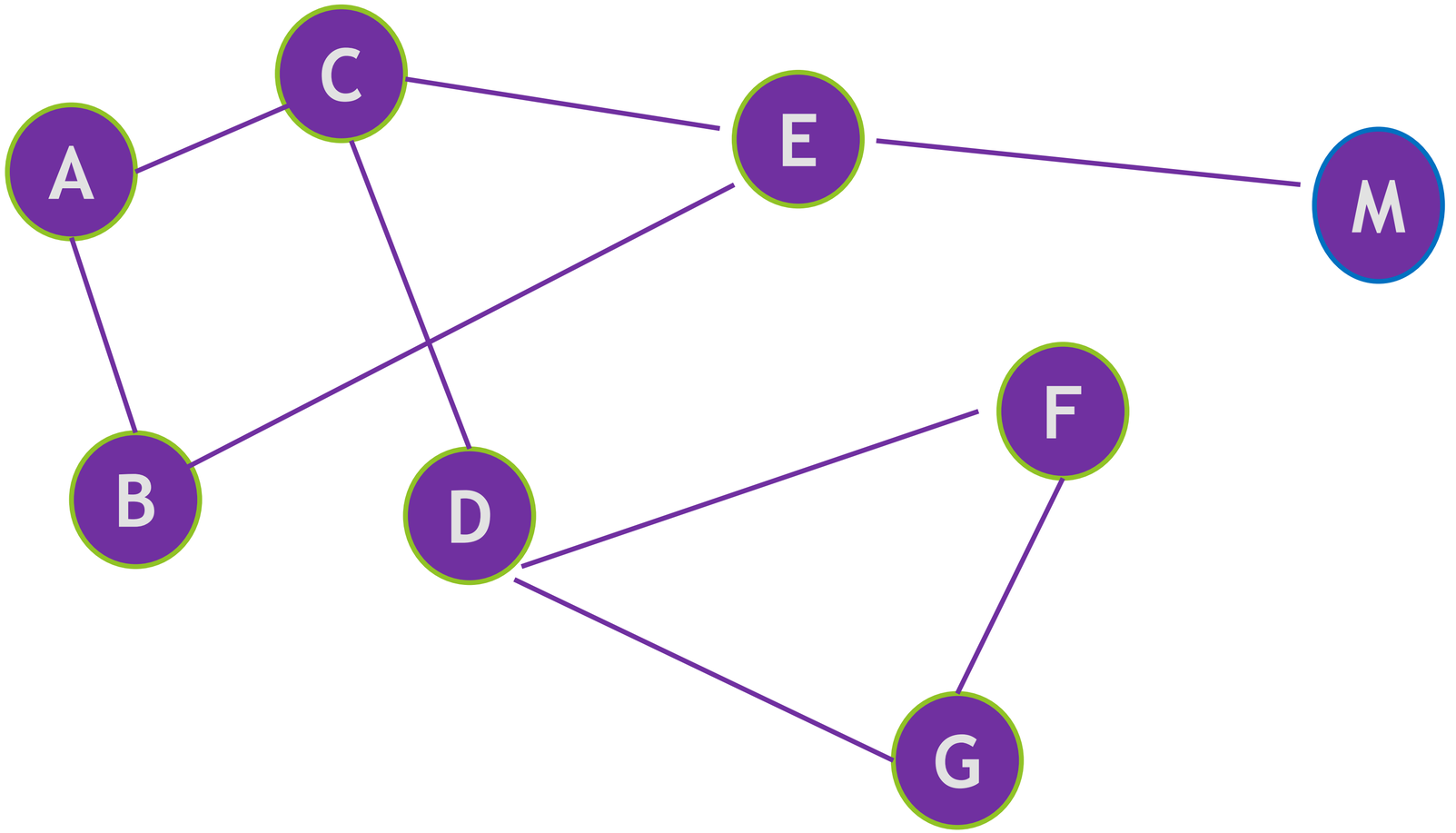
DFS

無路可走，返回 B



DFS

試圖往C走，但已經走過

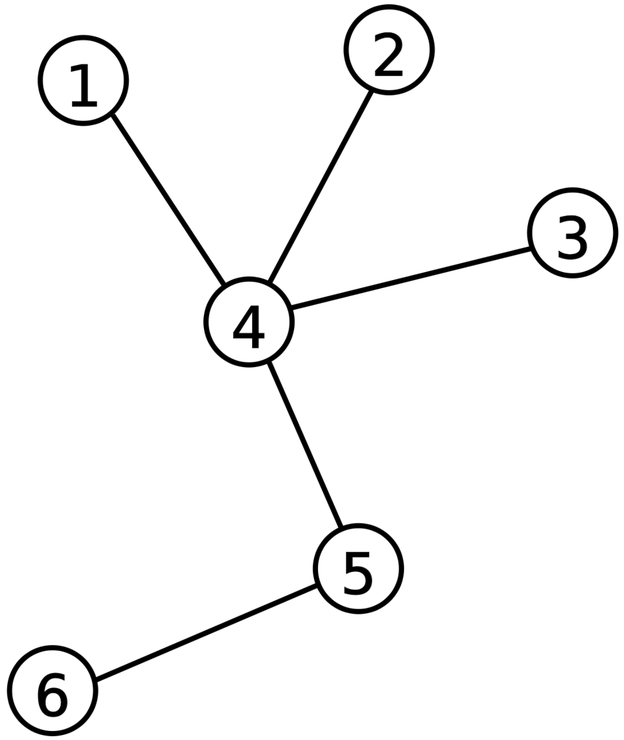


不過，DFS 通常不會用在一般的圖，

通常是用在“樹”這種圖上。

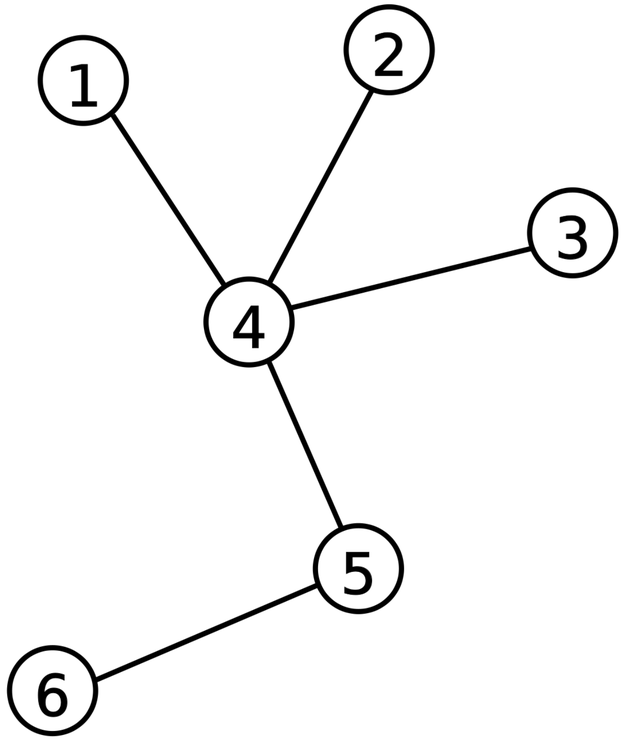
“樹”（Tree），
是一種沒有環的連通圖

如圖所示：



樹的小常識:

1. 沒有環
2. 總共 N-1 條邊
3. 通常會有一個根(root)
4. 深度 : 與根的距離
5. 葉節點 : 沒有小孩的節點



試題演練

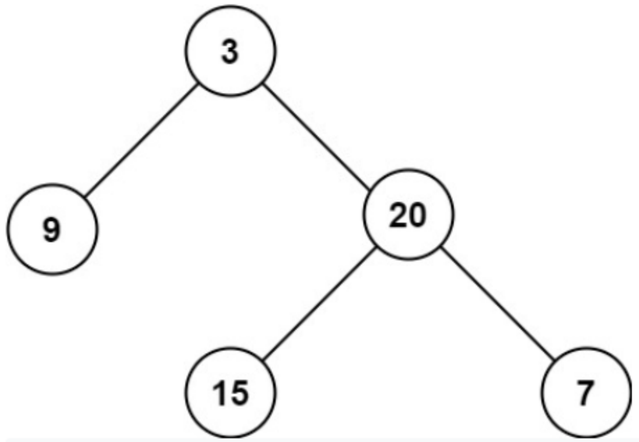
104. Maximum Depth of Binary Tree

Easy 4220 100 Add to List Share

Given the `root` of a binary tree, return *its maximum depth*.

A binary tree's **maximum depth** is the number of nodes along the longest path from the root node down to the farthest leaf node.

Example 1:




```
Input: root = [3,9,20,null,null,15,7]
Output: 3
```

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
```

https://leetcode.com/problems/maximum-depth-of-binary-tree

試題演練

```
class Solution {
public:
    int ans = 0;
    void dfs(TreeNode *&now , int dep=1){
        if(now == nullptr) return;
        if(now->left == nullptr && now->right == nullptr){
            ans = max(ans , dep);
        }
        dfs(now->left,dep+1);
        dfs(now->right,dep+1);
    }
    int maxDepth(TreeNode* root) {
        dfs(root);
        return ans;
    }
};
```

試題演練

- 簡單的二元樹 DFS

試題演練

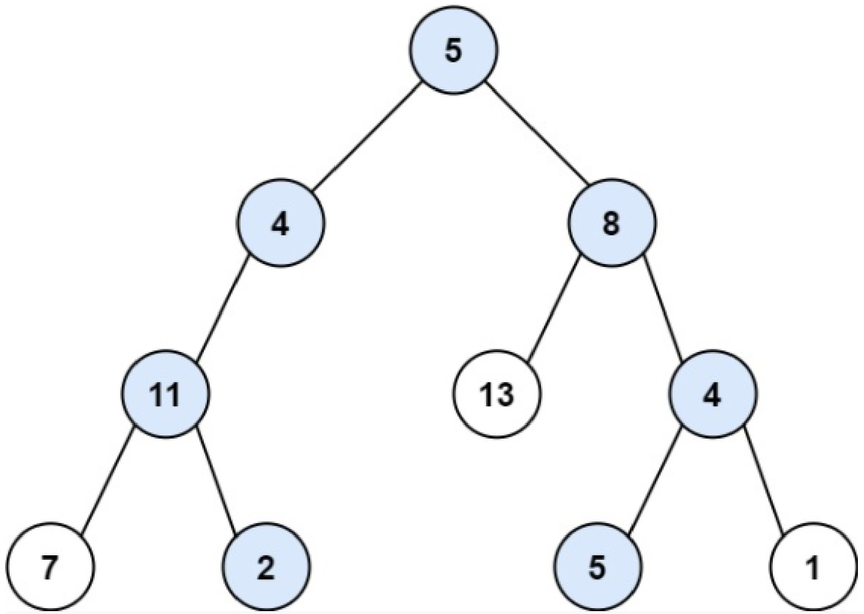
113. Path Sum II

Medium 2988 85 Add to List Share

Given the `root` of a binary tree and an integer `targetSum`, return all **root-to-leaf** paths where each path's sum equals `targetSum`.

A **leaf** is a node with no children.

Example 1:




```
Input: root = [5,4,8,11,null,13,4,7,2,null,null,5,1], targetSum = 22
Output: [[5,4,11,2],[5,8,4,5]]
```

https://leetcode.com/problems/path-sum-ii/

試題演練

```
12  class Solution {
13  public:
14      vector< vector<int> > ans;
15      vector<int> temp;
16      int target;
17      void dfs(TreeNode *now  , int sum){
18          temp.push_back(now->val);
19          sum += now->val;
20          if(now->left == nullptr && now->right == nullptr && sum == target){
21              ans.push_back(temp);
22          }
23          if(now->left != nullptr){
24              dfs(now->left , sum);
25          }
26          if(now->right != nullptr){
27              dfs(now->right , sum);
28          }
29          sum -= now->val;
30          temp.pop_back();
31      };
32      vector<vector<int>> pathSum(TreeNode* root, int targetSum) {
33          if(root == nullptr) return ans;
34          target = targetSum;
35          dfs(root , 0);
36          return ans;
37      }
38  };
```

試題演練

- 一樣是二元樹 DFS，但增加了一點參數、數值的維護
- 透徹的了解遞迴，才是學好 DFS 的關鍵

試題演練

886. Possible Bipartition

Medium 1545 38 Add to List Share

Given a set of `n` people (numbered `1, 2, ..., n`), we would like to split everyone into two groups of **any** size.

Each person may dislike some other people, and they should not go into the same group.

Formally, if `dislikes[i] = [a, b]`, it means it is not allowed to put the people numbered `a` and `b` into the same group.

Return `true` if and only if it is possible to split everyone into two groups in this way.

Example 1:

```
Input: n = 4, dislikes = [[1,2],[1,3],[2,4]]
Output: true
Explanation: group1 [1,4], group2 [2,3]
```

Example 2:

```
Input: n = 3, dislikes = [[1,2],[1,3],[2,3]]
Output: false
```

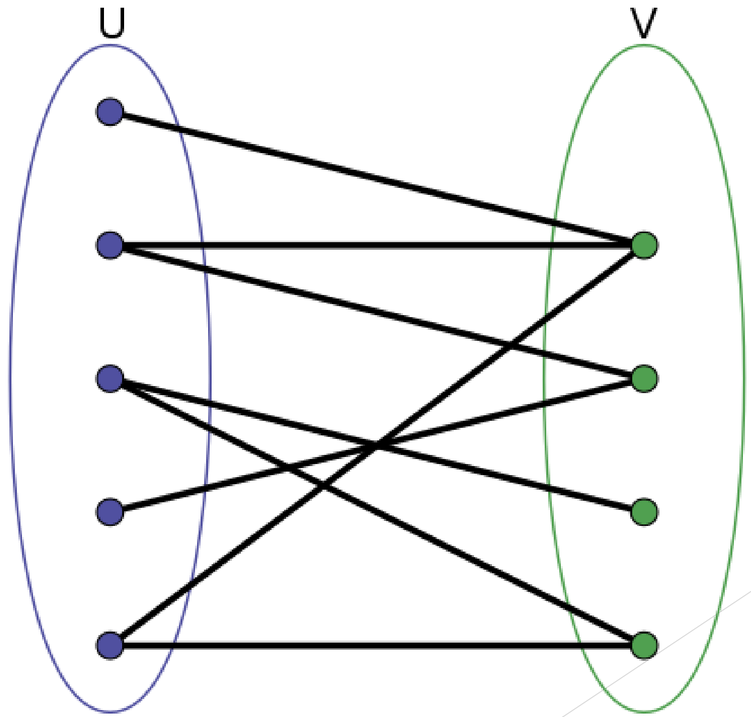
Example 3:

```
Input: n = 5, dislikes = [[1,2],[2,3],[3,4],[4,5],[1,5]]
Output: false
```

https://leetcode.com/problems/possible-bipartition/

試題演練

- “兩個互相討厭的人”是一種點跟點之間的關係
- 要確認這張圖是不是二分圖，即相鄰的邊不能同色(邊)



試題演練

```
class Solution {
public:
    bool ok = true;
    vector<int> G[2052];
    int color[2050];
    void dfs(int now){
        for(int &c : G[now]){
            if(color[c] == color[now]) ok = false;
            else if(color[c] == 0){
                color[c] = 3-color[now];
                dfs(c);
            }
        }
    }
    bool possibleBipartition(int n, vector<vector<int>>& dislikes) {
        for(auto edges : dislikes){
            G[edges[0]].push_back(edges[1]);
            G[edges[1]].push_back(edges[0]);
        }
        for(int i=1 ; i<=n ; ++i){
            if(!color[i])
                color[i] = 1 , dfs(i);
        }
        return ok;
    }
};
```

試題演練

- 只有在同一個連通塊的點互相影響
- 找出連通塊一樣可以用 DFS（當然 BFS 也行但沒必要）
- 對於每個連通塊，

  只要我們對其中一個點塗色後，其他點的顏色就確定了!

課後練習

- https://leetcode.com/problems/surrounded-regions/
- https://zerojudge.tw/ShowProblem?problemid=b059
- https://leetcode.com/problems/island-perimeter/
- https://leetcode.com/problems/jump-game-iii/
- https://leetcode.com/problems/maximum-depth-of-n-ary-tree/