

資料結構



演算法

圖的搜尋

Graph Search

圖的搜尋 Graph Search

[十八豆教育科技](#)

還記得一開始提的這幾個問題嗎？



如果想從紅色圈圈的 Bond Street 到達藍色圈圈的 Temple，要如果搭乘呢？



如果 Marshall 想認識 May 的話，可以透過誰介紹呢？

C 語言資料型態

char 文字

想學習 int 如何使用的話，要先了解哪些知識點呢？

數字

整數

int

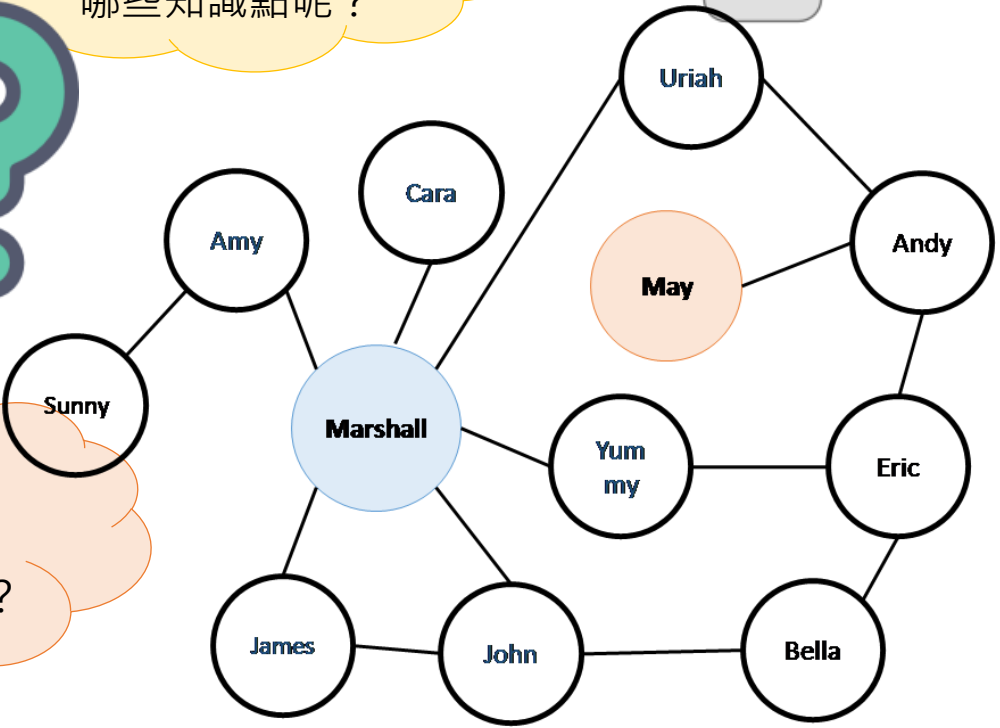
short

long

符點數

float

double

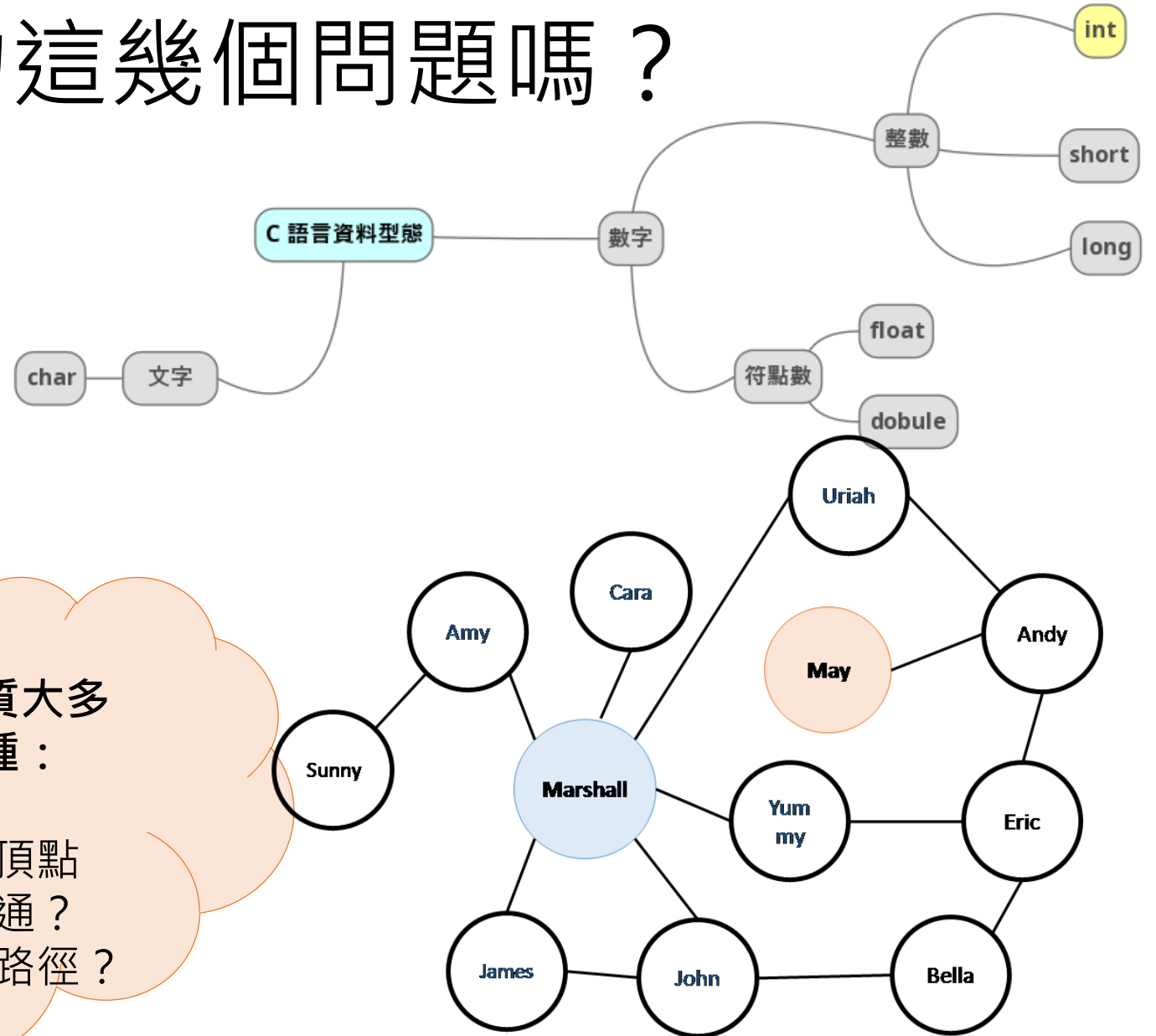


還記得一開始提的這幾個問題嗎？



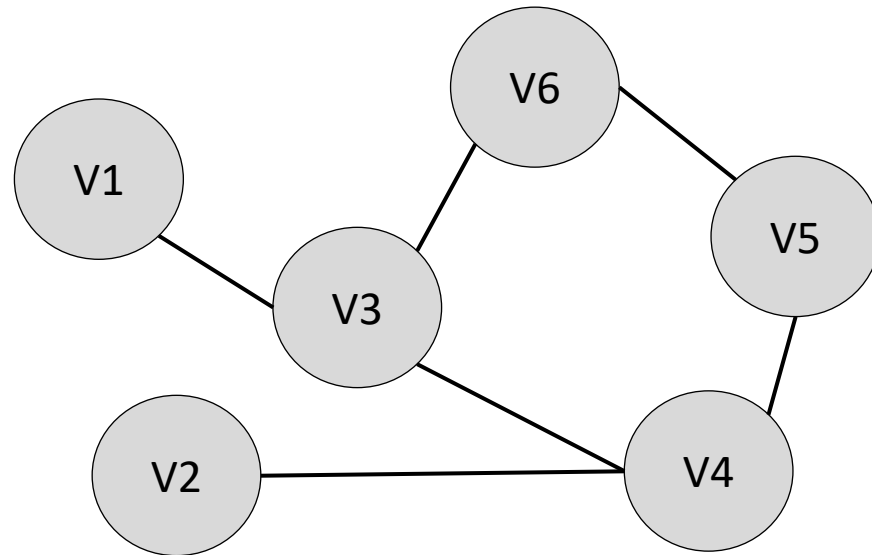
這些問題的本質大多圍繞在以下幾種：

- 列出圖內所有頂點
- 任兩點是否連通？
- 任兩點之間的路徑？



最基本的圖形問題？

- 列出圖內所有頂點
- 列出圖內所有邊
- 任兩點是否連通？
- 任兩點之間的路徑？
- 任兩點之間的最短路徑？



如何回答最基本的圖形問題？

- 列出圖內所有頂點
- 列出圖內所有邊

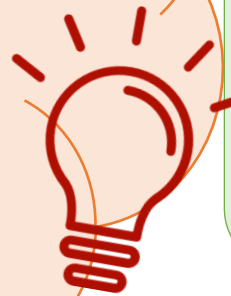


走過整個圖形內的邊與頂點，答案自然就出現！



問題是不是有點熟悉呢？

其實這就跟處理平面上的移動一樣，都可以使用**廣度優先搜尋(BFS)**與**深度優先搜尋(DFS)**解決喔！



- 任兩點是否連通？
- 任兩點之間的路徑？
- 任兩點之間的最短路徑？



就像迷宮一樣，從入口（起始點）出發，探險所有可以前進的路線，到達出口（結束點）就達成任務了！

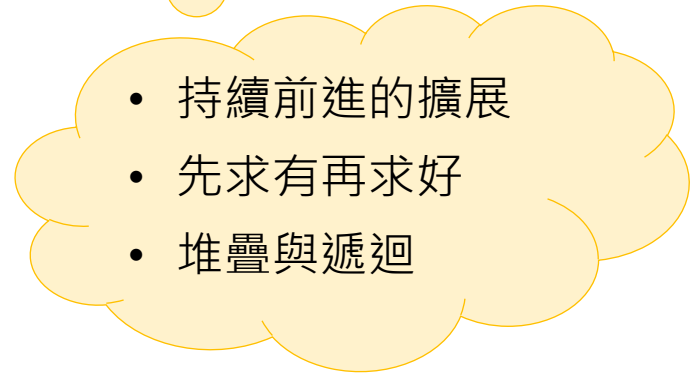
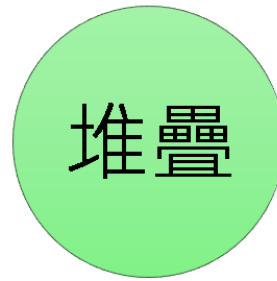
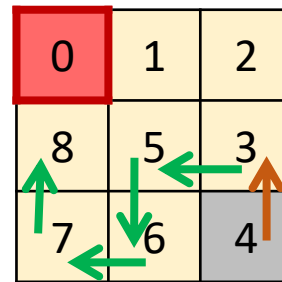
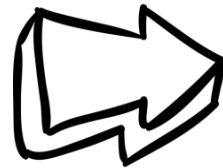
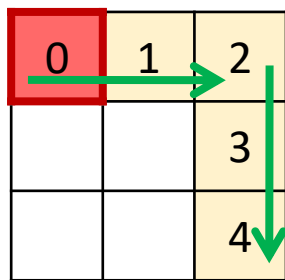
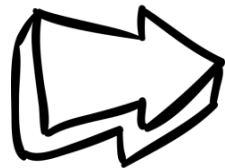
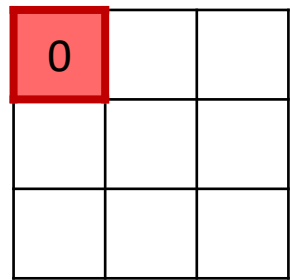
DFS

深度優先搜尋



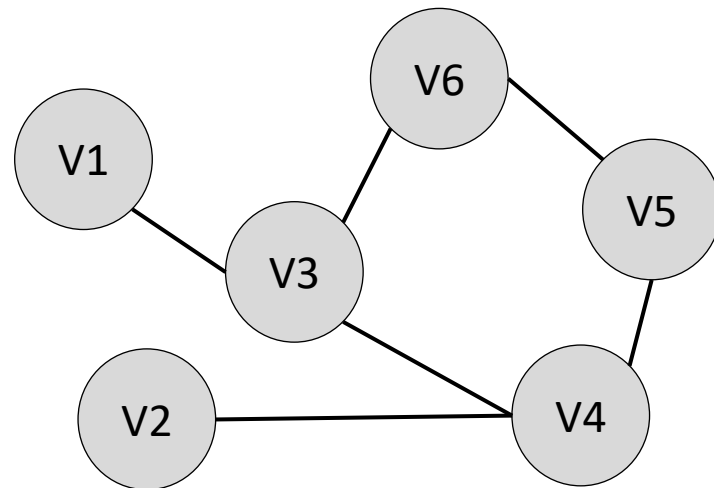
先招喚 DFS 回憶

- 只要旁邊還有沒拜訪過的格子，就**持續前進**！
- 如果沒有可前進的格子，就往回退一格，找尋可前進的點



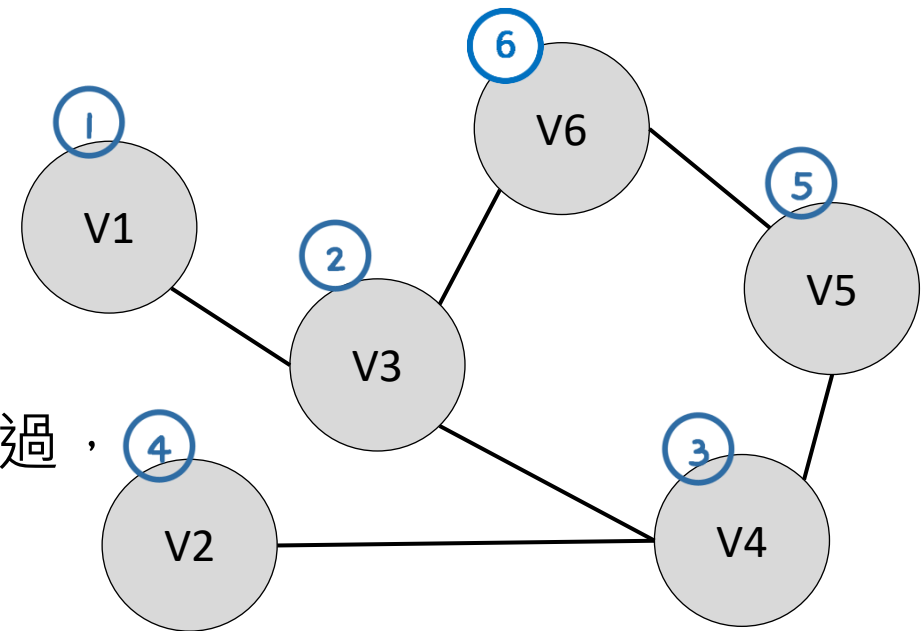
套用在圖形結構的DFS

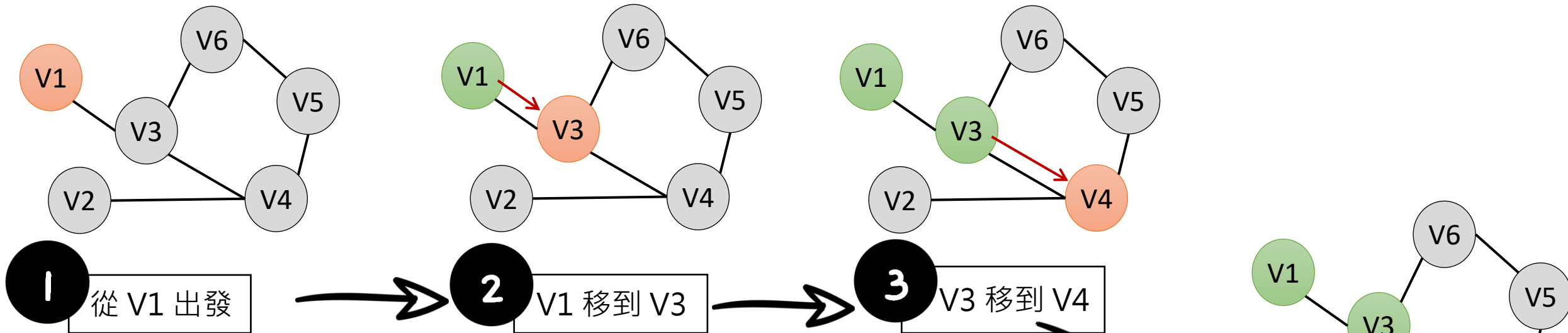
- 概念是一樣的！
- 先遇到的相鄰點，就先拜訪！
- 已經沒有沒拜訪過的相鄰點，就返回，找出前面走過的點中還沒拜訪過的！



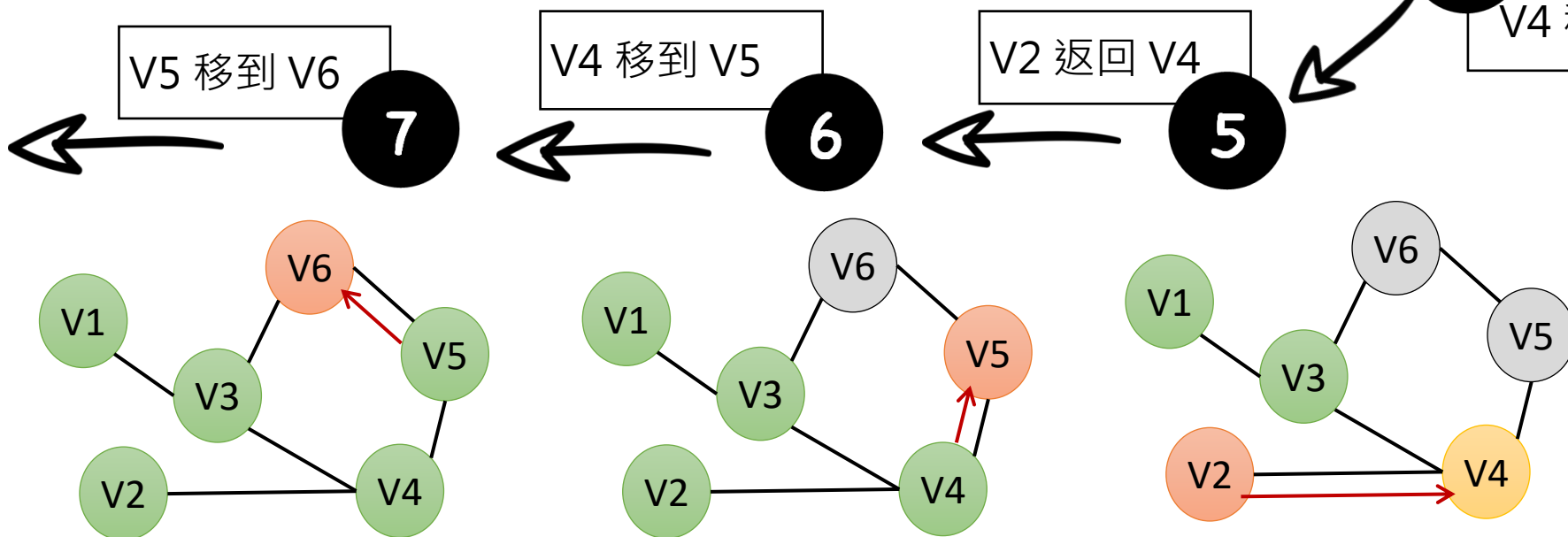
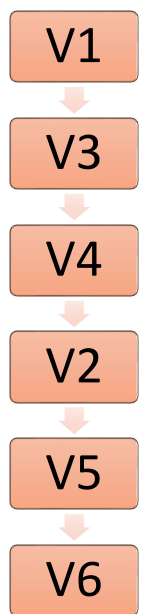
DFS 應用在圖形結構

- 將 V1 視為原點，從 V1 出發
- V1 只能到 V3，因此移到 V3
- V3 往前可以到達 V4 與 V6，先移到 V4
- V4 往前可以到達 V2 與 V5，先移到 V2
- V2 不能往前走了，因此返回 V4
- 與 V4 相鄰的點(V2、V3、V5)中，還有 V5 沒拜訪過，因此移到 V5
- V5 往前可以到達 V6，因此移到 V6
- 到達 V6 後，6 個頂點都已經拜訪過，DFS 也就完成了！





完成 DFS



實作深度優先搜尋 – 搭配堆疊

```
void DFS(int vertex, int verticesCount, int graph[][MAX_VERTICES], int visited[])  
{  
    int i, j, top=0, parent;  
    int ancestor[verticesCount+1], stack[verticesCount+1];  
  
    memset(ancestor, -1, sizeof(ancestor));  
    stack[top++] = vertex;  
    while(top >= 0) {  
        i = stack[top--];  
        while (visited[i]==1 && top >=0) {  
            i = stack[top--];  
        }  
  
        if (top < 0)  
            break;  
        printf("Visit v%d\n", i);  
        visited[i] = 1;  
  
        for(j = 0; j < verticesCount; j++) {  
            if(visited[j] == 0 && graph[i][j] == 1) {  
                top++;  
                stack[top] = j;  
                ancestor[j] = i;  
            }  
        }  
    }  
    printf("\n");  
}
```

找出還沒拜訪過的節點

全部都被拜訪過，就可以結束走訪

使用堆疊記錄曾經看到但未拜訪過的節點

實作深度優先搜尋 – 搭配遞迴

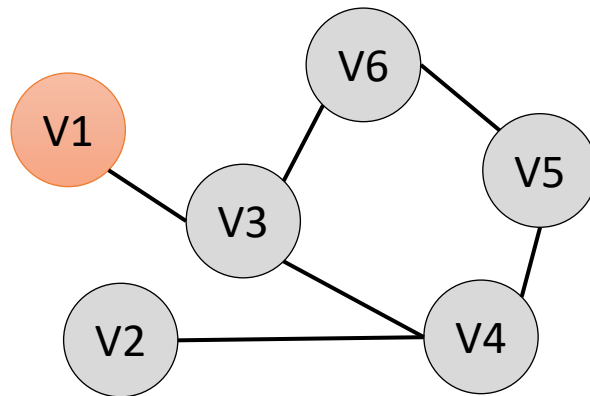
```
void DFS(int vertex, int verticesCount, int graph[][MAX_VERTICES], int visited[])  
{  
    int j;  
    printf("Visit v%d\n", vertex);  
    visited[vertex] = 1;  
    for(j=0; j<verticesCount; j++) {  
        if(!visited[j] && graph[vertex][j]==1) {  
            DFS(j, verticesCount, graph, visited);  
        }  
    }  
}
```

找出目前所在節點的所有相鄰頂點

使用遞迴方式拜訪尚未拜訪過的節點

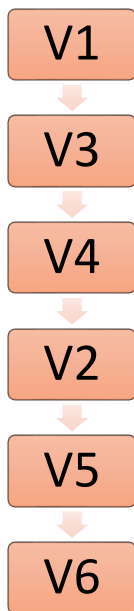


不同的路徑結果



每次都選編號比較小的先拜訪

每次都選編號比較大的先拜訪



DFS 提供的是搜尋演算法邏輯，但實作時有些細節規則並沒有絕對的答案，交由程式開發者自行決定喔。



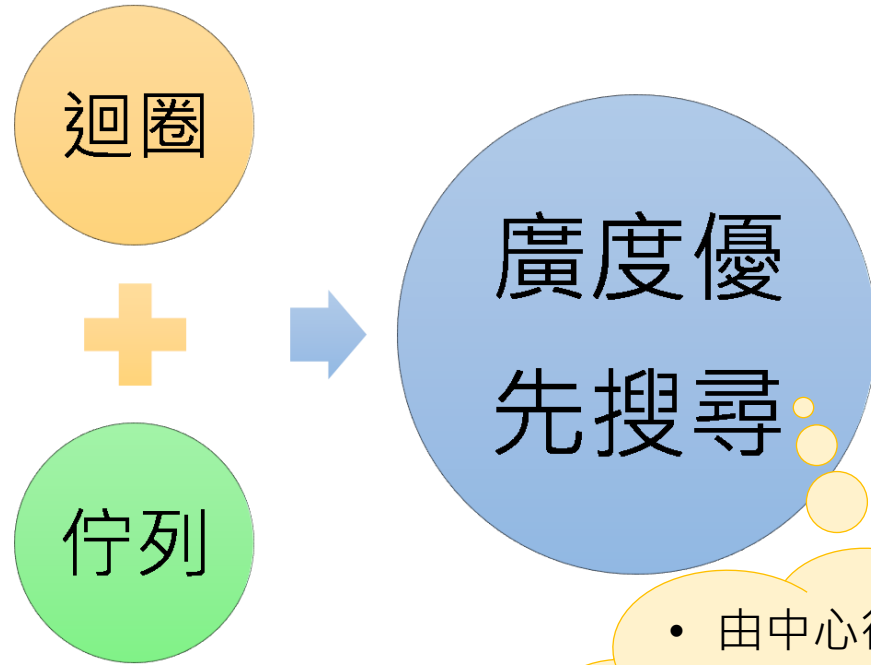
BFS

廣度優先搜尋

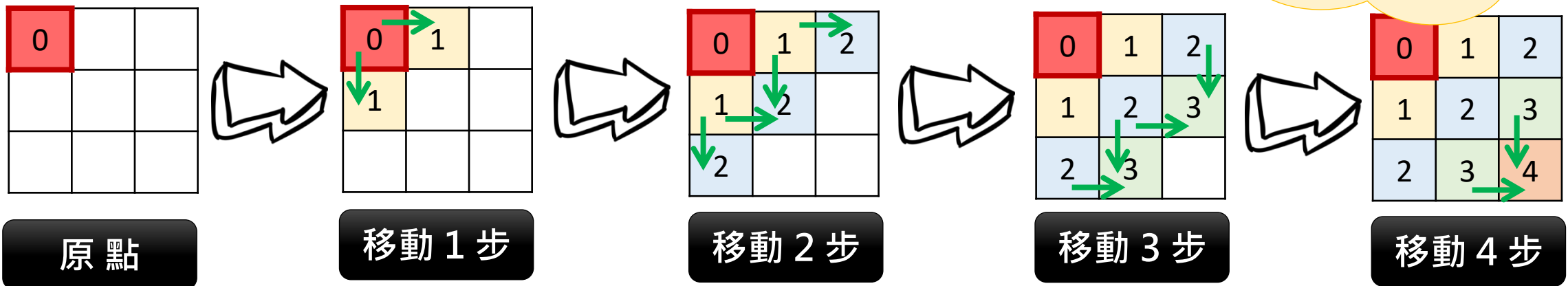


先招喚 BFS 回憶

- 先訪問從目前位置移動 1 步可到達的點
- 再訪問移動 2 步可到達的點，3 步可到達的點，以此類推

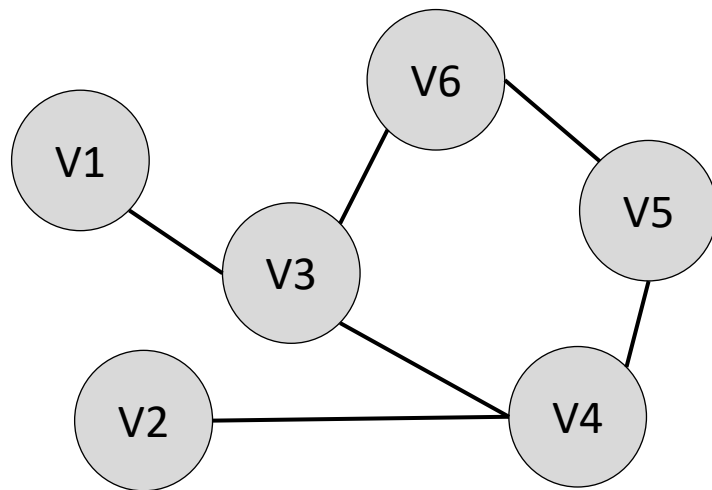


- 由中心往外的擴展
- 一次到位弄到好
- 佇列與迴圈



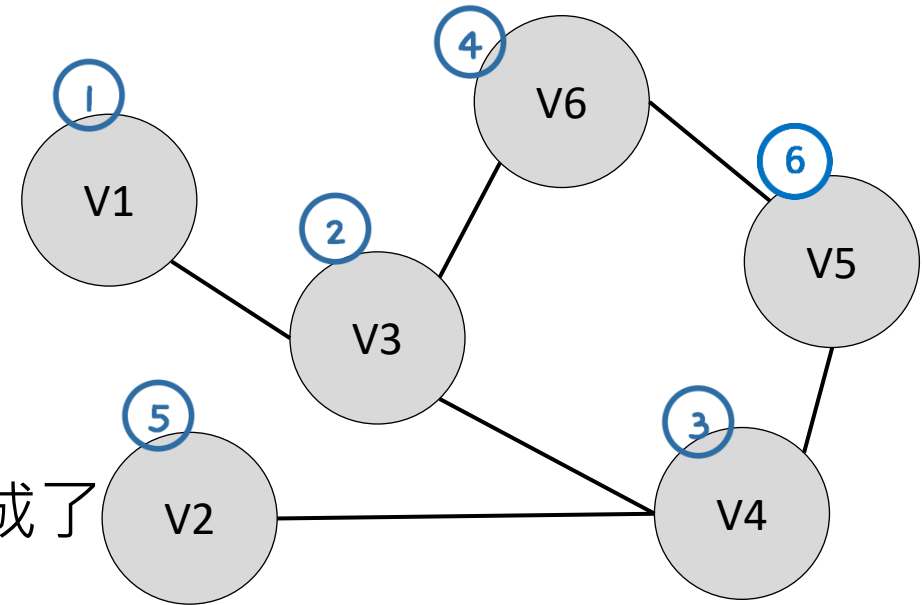
套用在圖形結構的BFS

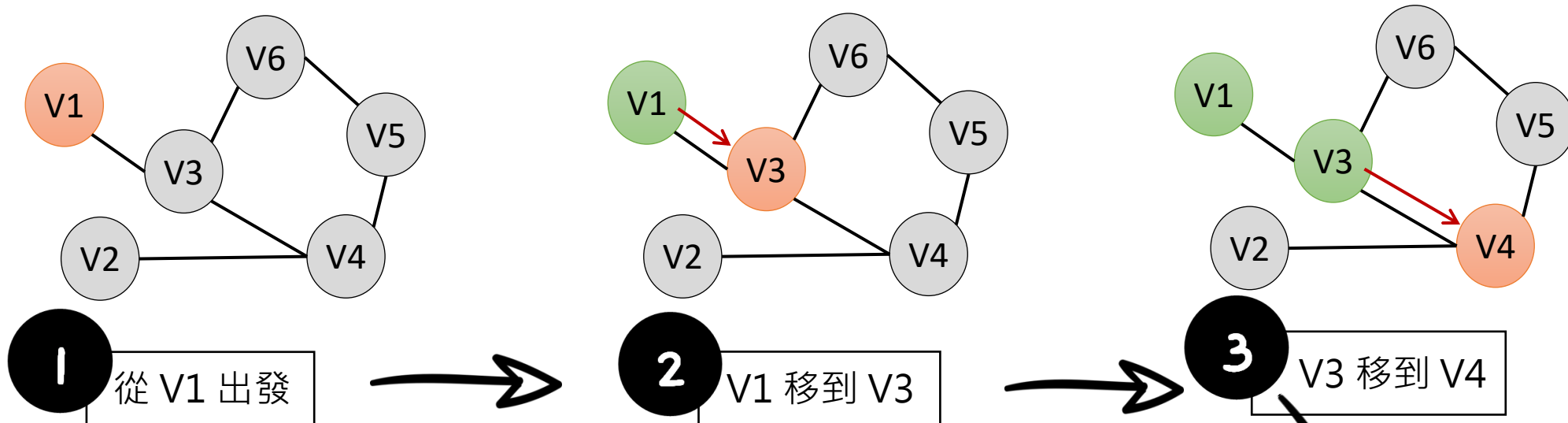
- 概念是一樣的！
- 所有相鄰點都要先拜訪過才能再往外找！



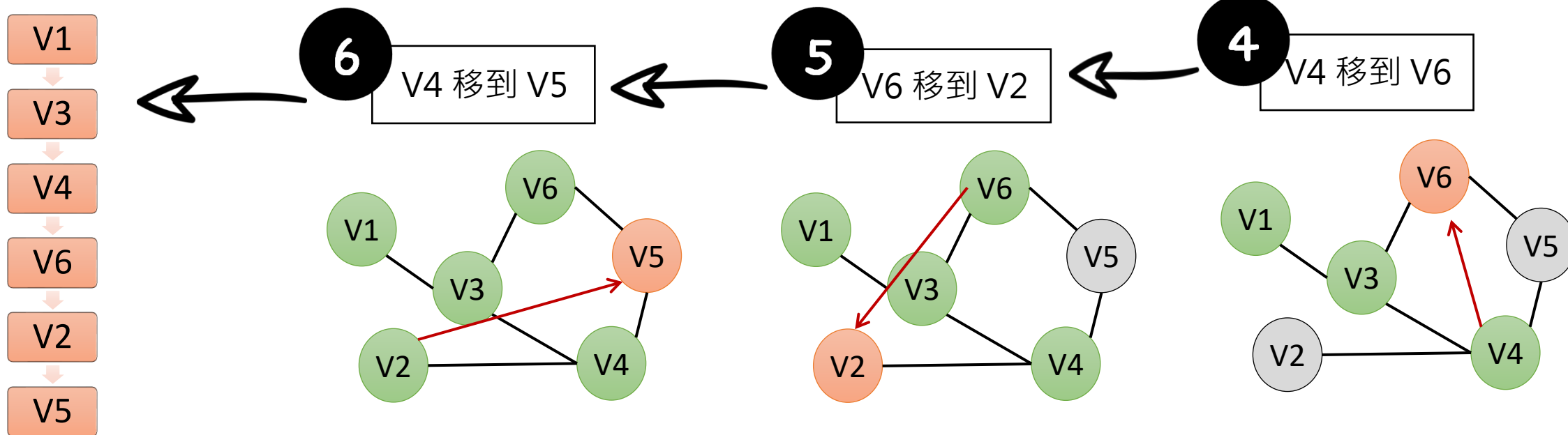
BFS 應用在圖形結構

- 將 V1 視為原點，從 V1 出發
- V1 的相鄰點有 V3，因此移到 V3
- V3 的相鄰點有 V4與 V6，先移到 V4
- 再移到V3的另一個相鄰點 V6
- V4的相鄰點有 V2與 V5，先移到 V2
- 再移到V4的另一個相鄰點 V5
- 到達V5後，6個頂點都已經拜訪過，BFS 也就完成了





完成 BFS



實作廣度優先搜尋 – 搭配佇列

```
void BFS(int vertex, int verticesCount, int graph[][MAX_VERTICES], int visited[MAX_VERTICES])
{
    int j, ancestor[verticesCount+1];
    queueMaster *queue;
    memset(ancestor, -1, sizeof(ancestor));
    queue = createQueue(queue);
    enqueue(queue, vertex);

    while(!isEmpty(queue)) {
        vertex = dequeue(queue);
        if (visited[vertex] == 1)
            continue;
        visited[vertex] = 1;
        printf("Visit v%d\n", vertex);
        for(j = 0; j < verticesCount; j++) {
            if(visited[j] == 0 && graph[vertex][j] == 1) {
                enqueue(queue, j);
                ancestor[j] = vertex;
            }
        }
    }
    printf("\n");
}
```

判斷佇列內是否還有節點未拜訪

從佇列內取出位拜訪的節點

使用佇列記錄曾經看到但尚未拜訪過的節點





延伸的概念

概念 1: 圖的走訪 Graph traversal

- 透過 DFS 或 BFS 使用的圖形搜尋，因為本質是要依照邊的相連而依序列出全部各個點，因此這種搜尋我們稱為**圖的走訪**或是**圖的追蹤**，即『**Graph Traversal**』！