

Advanced Competitive Programming

國立成功大學ACM-ICPC程式競賽培訓隊
nckuacm@imslab.org

Department of Computer Science and Information Engineering
National Cheng Kung University
Tainan, Taiwan

Outline

- 賽後檢討
- Graph (基本資料結構)
- map, set and priority_queue
- DFS & BFS

賽後檢討

常用手法以及容易忽略的細項

Outline

- 解題建議步驟
- Coding 的注意事項
- 自我修練的方法

Outline

- 解題建議步驟
- Coding 的注意事項
- 自我修練的方法

演算法競賽跟數學競賽很像

以第一次比賽的 z004 [傷害](#) 作為例子

把題目描述看過

簡單的看過 Input & Output 格式說明

對著 Sample 自己手動模擬一次

接著注意看各種條件限制

思考演算法的實現(通常先考慮正確性再想效率)

Outline

- 解題建議步驟
- **Coding** 的注意事項
- 自我修練的方法

測資範圍

陣列開大一點點

假設測資大小為 $1 \sim 2 * 10^5$

```
int const maxn = 2e5 + 10;  
int a[maxn];
```


cin / cout

- 有很好的型態支援度
- 速度比 `scanf` 和 `printf` 慢？
 - 原因是出在要配合 `scanf` 和 `printf`，如果我們把它取消來看看會發生什麼事情。
 - `ios::sync_with_stdio(0)`
 - `cin.tie(0)`

a002. sort speed up	AC (0.9s, 19.4MB)	CPP	2019-02-25 17:37
-----	-----	-----	2019-02-25 17:38
a002. sort speed up	AC (1.7s, 19.4MB)	CPP	2019-02-25 17:18

getline

```
cin.ignore();
```

```
while (getline(cin, str)) {  
    if (str.empty()) break;  
    :  
    .  
}
```

Outline

- 解題建議步驟
- Coding 的注意事項
- 自我修練的方法

讀書

- 算法競賽入門經典（第2版）
- 挑戰程序設計競賽（第2版）
- 資訊之芽
- 建中資訊科培訓講義

看題解

每打完一場比賽就去看別人怎麼寫的

不管該題自己是否已經解出來，別人的想法都值得學習

有時候可以為別人挑毛病，有效率的減少自己出錯率

有時別人比你更勝一籌，用更優雅的方法實作了演算法

看題解

不會的題目，不建議先直接看解答，最好先想過一遍，束手無策才去看解答

因為從無到有**想法的誕生**過程很重要，這些幾乎是別人很難教給你的

討論

對於某個領域，大致可以分成強者以及弱者

而面對這兩類人，

強者：把問題**清晰**地描述出來，請教他

弱者：把解答**清晰**地描述出來，教會他

事物能清晰表達 ⇒ 看出對於事物的掌握度
⇒ 能很快的實現演算法

Questions?

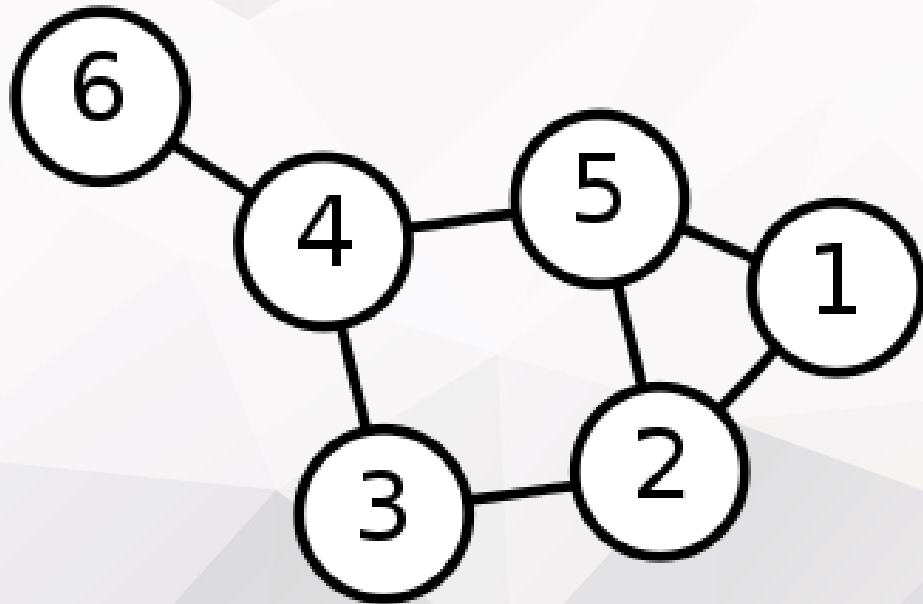
Data Structures

Outline

- Graph
- Tree
- Disjoint sets

Graph

- 圖 (Graph) ，是一個由邊 (Edge) 集合與點 (Vertex) 集合所組成的資料結構。



Graph

- 點 (vertex) : 組成圖的最基本的元素
- 邊 (edge) : 點與點的關係
- 有向圖 (directed graph) : 邊帶有方向性
- 無向圖 (undirected graph) : 每條邊都是雙向的
-> 沒有方向性

Graph

- 道路 (walk) : 點邊相間的序列 ,
e.g. $v_0 e_1 v_1 e_2 v_2 \dots e_n v_n$
- 路徑 (path) : 點不重複的道路
- 環 (cycle) : 路徑的起點與終點連接後形成環
- 走訪/遍歷 (traversal/search) : 走完全部的點或邊

該怎麼表示一張圖呢

Graph 鄰接矩陣

- 用一個二維陣列 $g[i][j]$ 來表示從 i 點到 j 點的距離
- 通常會有一個特殊的值來表示無法到達的情況
- 例如 `INT_MAX` or `-1`

Graph 鄰接表

- 常用 vector 表示一張圖

```
vector<int> e[N];
```

```
int from, to;
```

```
while (cin >> from >> to) {
```

```
    e[from].push_back(to);
```

```
}
```

- $e[i]$ 代表 i 能夠走到的點的陣列

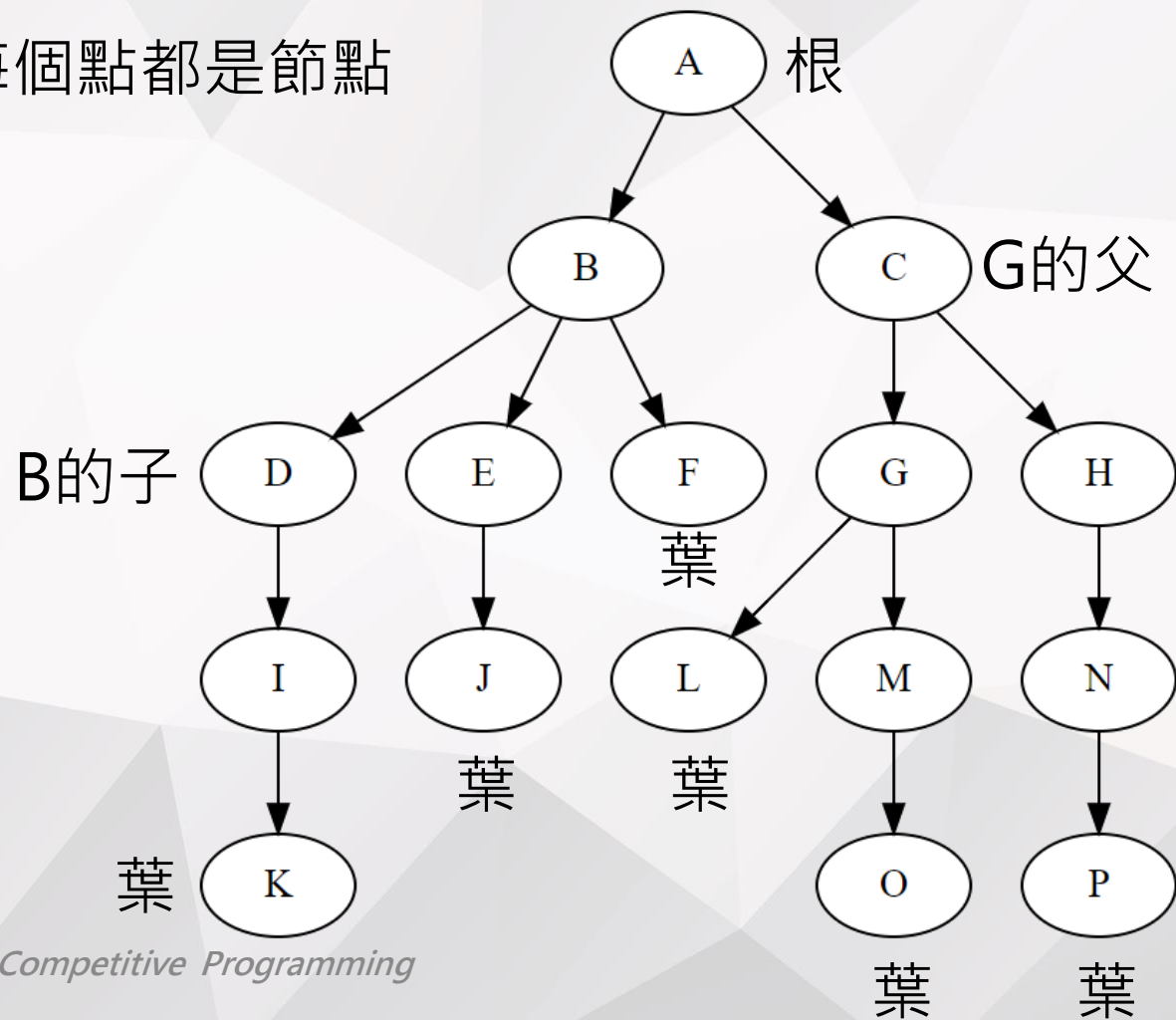
Tree

- Tree 是一個有向無環連通圖
- 節點 (node) : 一般樹上的點
- 父 (parent) : 節點能反向拜訪的第一個節點
- 子 (child) : 節點能正向拜訪的第一個節點
- 根 (root) : 沒有父節點的節點
- 葉 (leaf) : 沒有子節點的節點



Tree

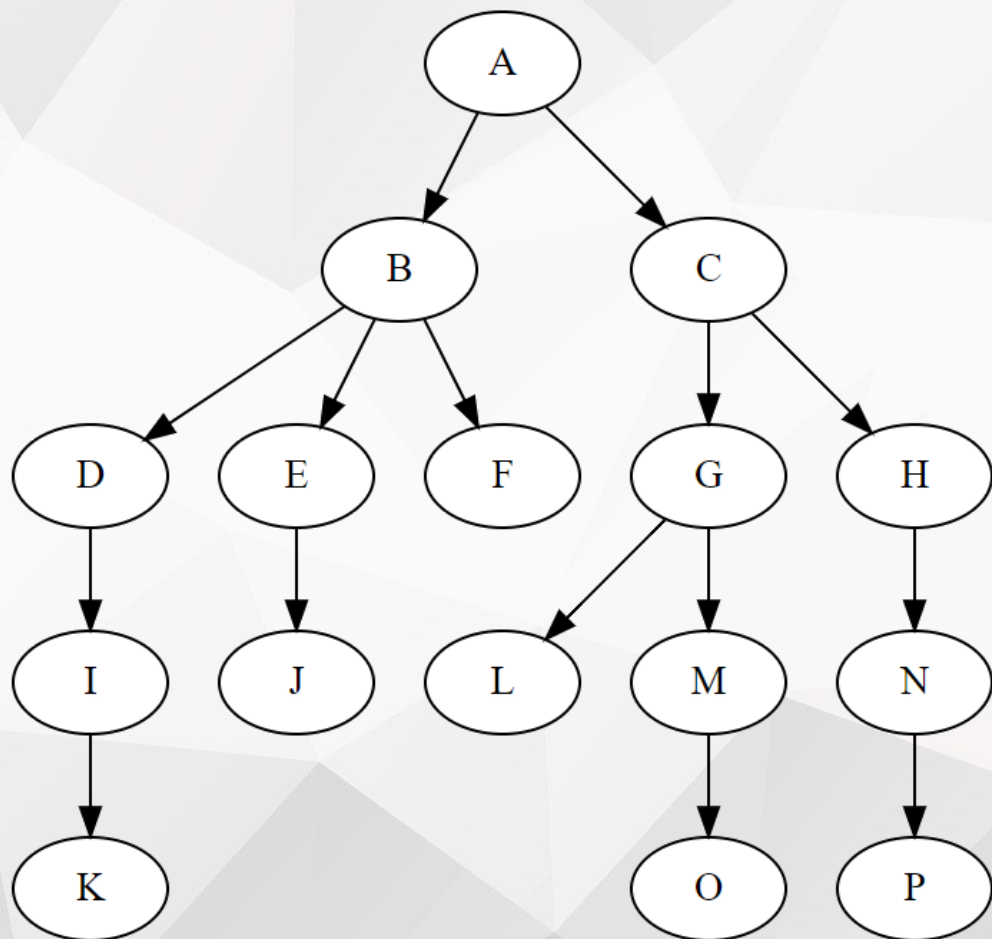
每個點都是節點



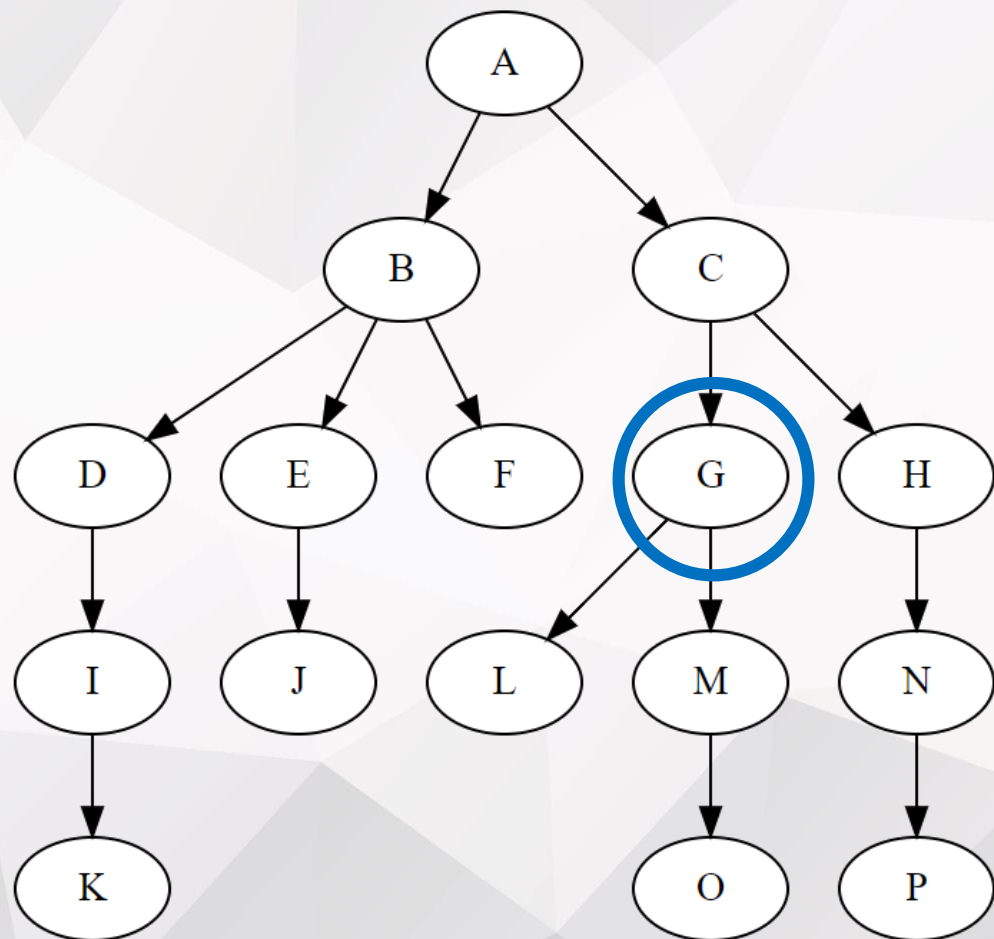
Tree

- 祖先 (ancestor) : 節點能反向拜訪的所有節點
- 孫子 (descendant) : 節點能正向拜訪的所有節點
- 深度 (depth) : 節點的深度為從根到該節點所經過的邊數
- 森林 (forest) : 一個集合包含所有不相交的 Tree
- 每個非根節點只有一個父節點

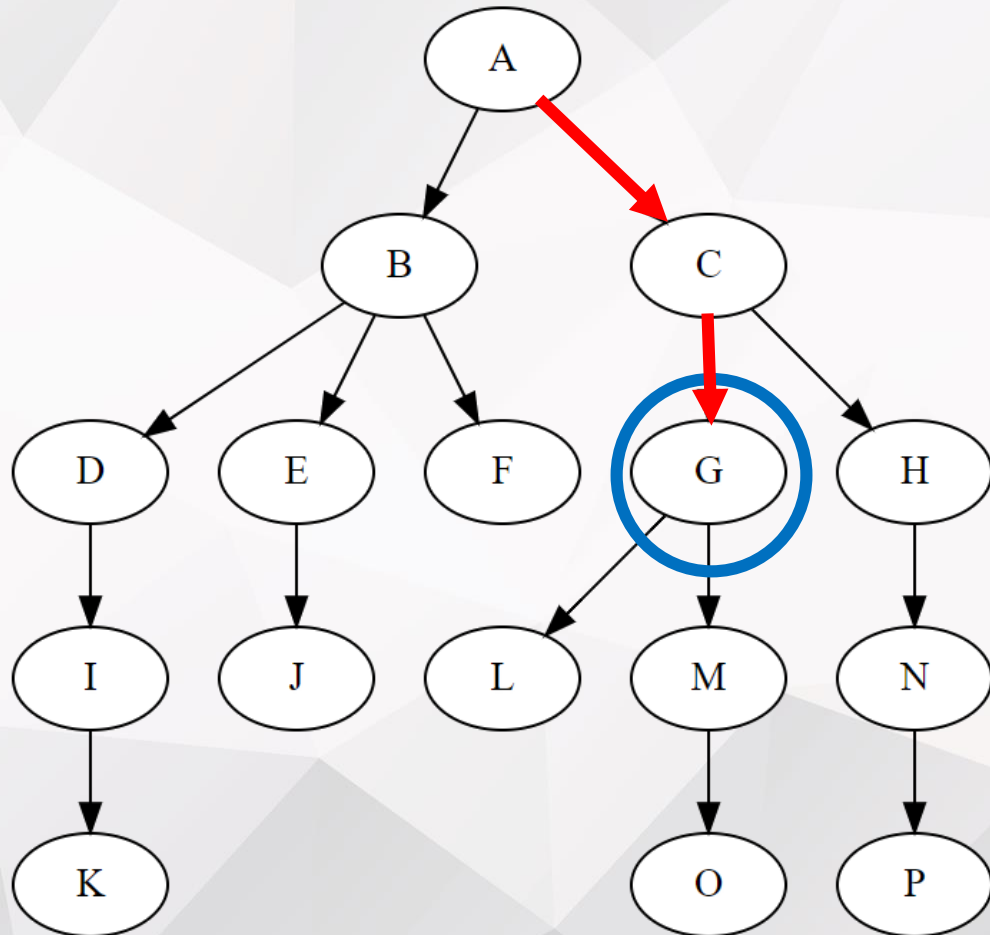
Tree



Tree

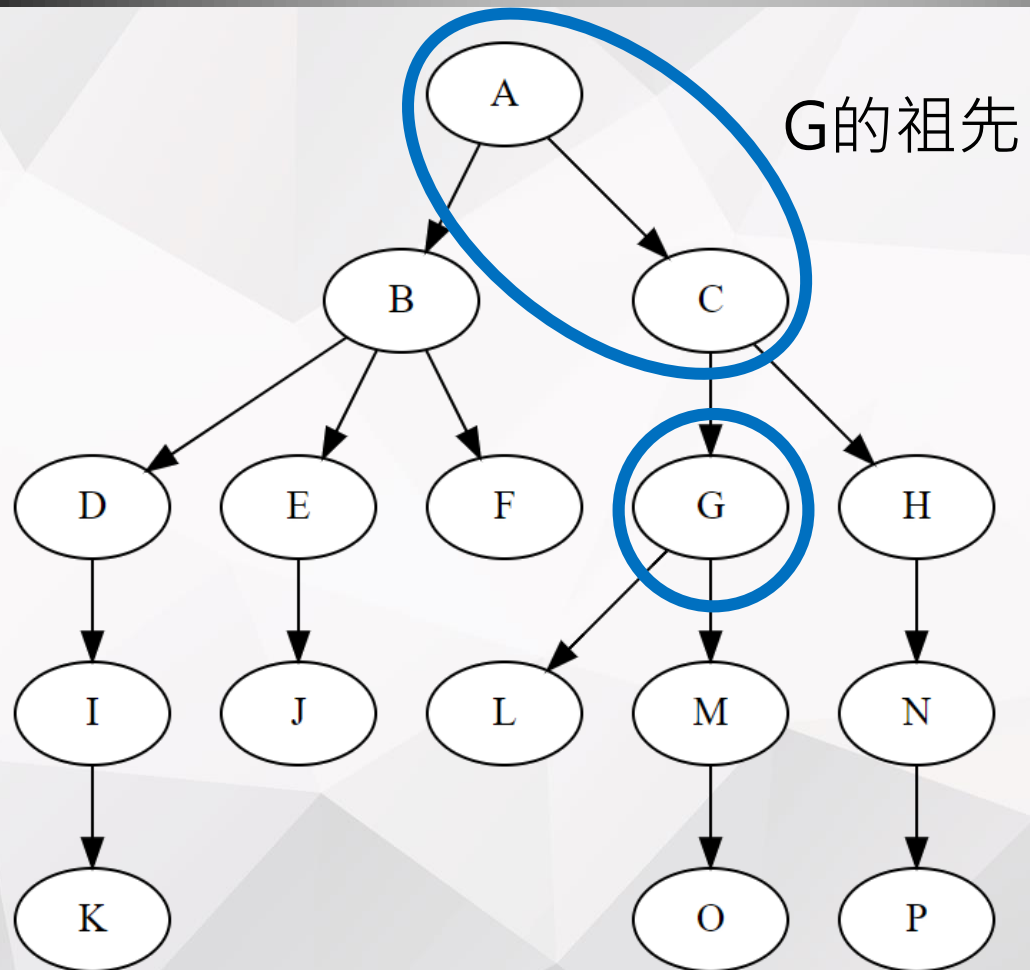


Tree

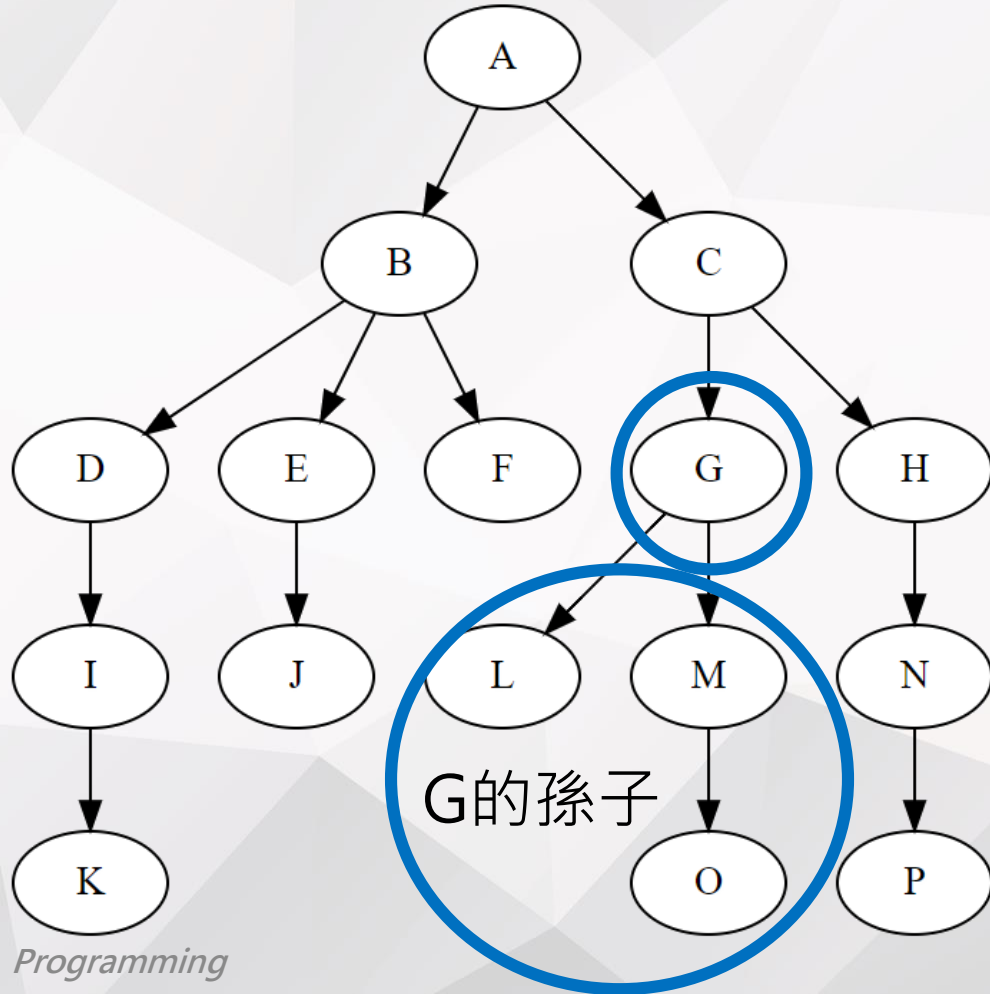


G的Depth為2

Tree



Tree



Disjoint sets

- 有點像是把人分組
- 不同組不會有相同的元素 (不能分身)
- 要對組別或是人做大量的操作

Disjoint sets 操作

- 將新的人加入組別
- 將兩個組合併
- 查詢一個組別的人數
- 查詢某人屬於哪一組
- 從某個組刪除一個人 (較難)

Disjoint sets 實作方式

- 為了能夠快速完成操作，採用以下的方式表達 Disjoint sets
- 以一棵樹來表達一個組

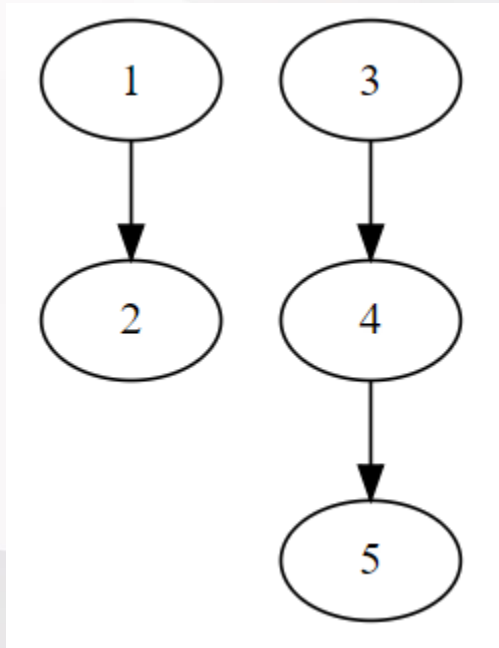
Disjoint sets Initialization

```
for (v = 1; v <= N; v++) group[v] = v
```



Disjoint sets Find

假設有元素 1 ~ 5，其中 1,2 一組，3,4,5 一組



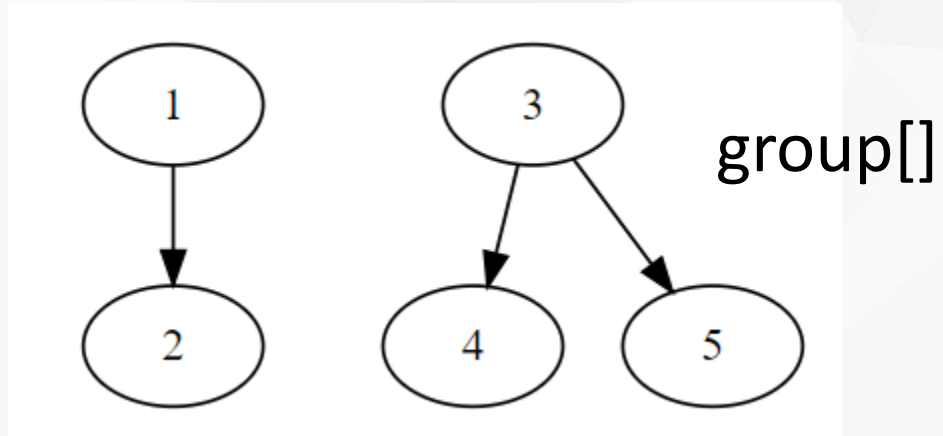
group[]

[1]	[2]	[3]	[4]	[5]
1	1	3	3	4

Disjoint sets Find

```
int Find(int v) {  
    if (v == group[v]) return v;  
    return group[v] = Find(group[v]);  
}
```

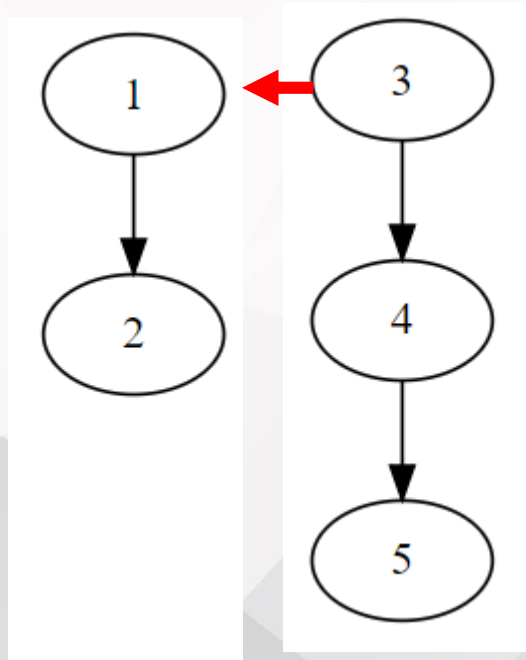
Disjoint sets Find



[1]	[2]	[3]	[4]	[5]
1	1	3	3	3

Disjoint sets Union

```
void Union(int u, int v) {  
    group[Find(u)] = Find(v);  
}
```



Disjoint sets Union

- 範例 [UVa OJ 879 Circuit Net](#)

Questions?

map, set and priority_queue

好用的 **S**tandard **T**emplate **L**ibrary 第三彈

map, set and priority_queue 共同特色

- 依照給定的規則儲存。

map

用整數索引找資料

```
int a[5] = {3, 7, 2, 7, 5};  
cout << a[3]; //7  
cout << a[1]; //7  
cout << a[0]; //3
```

- 考慮下列情況：
- 從名字查詢年齡，需要怎麼規劃儲存與查詢的方法？

map

用整數索引找資料

```
int a[5] = {3, 7, 2, 7, 5};
```

資料型態

索引

- 在 array 中：
資料型態可以自由定義
索引卻只能是整數

map

```
map<char,string> mymap;
```

```
mymap['a']="an element";
```

```
mymap['b']="another element";
```

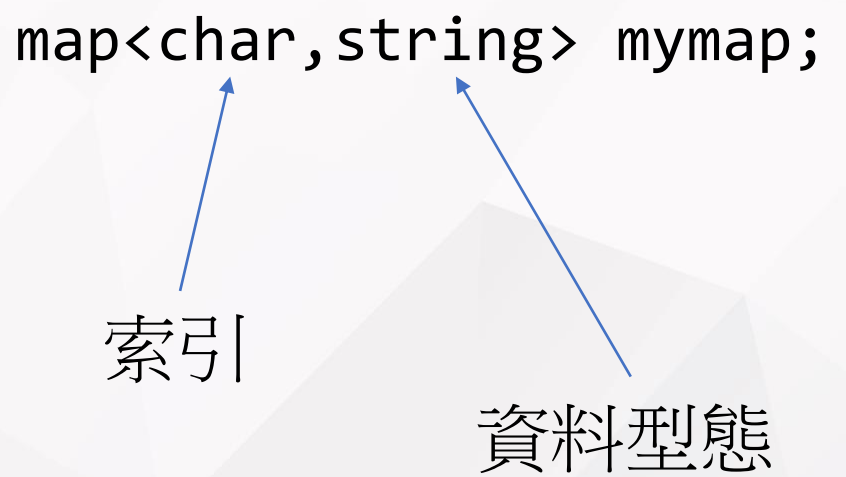
```
mymap['c']=mymap['b'];
```

```
cout << mymap['b']; //another element
```

```
cout << mymap['a']; //an element
```

```
map<char,string> mymap;
```

索引



The diagram illustrates the components of the map declaration. A blue arrow points from the Chinese character '索引' (index) to the 'char' type in the code. Another blue arrow points from the Chinese characters '資料型態' (data type) to the 'string' type in the code.

資料型態

map 好用的代價

```
mymap['b']="apple";  
cout << mymap['b'];
```

新增與取值的操作為 $O(\log N)$

```
a[0]=18;  
cout << a[0];
```

新增與取值的操作為 $O(1)$

map 遍歷

```
map<char,int> mymap;  
mymap['b'] = 100, mymap['a'] = 200, mymap['c'] = 300;  
  
for (auto it = mymap.begin(); it != mymap.end(); it++)  
    cout << it->first << " => " << it->second << endl;
```

Output:

a => 200

b => 100

c => 300

常見的 map member function

- `map::size`
- `map::clear`

題目賞析 - CodeForces 1133D

- 第一行有一個整數 N ，代表之後兩行各有 N 個整數

→ 5
1 2 3 4 5
2 4 7 11 3

題目賞析 - CodeForces 1133D

- 第二、三行有 N 個整數，是 $A_1 \sim A_N$ 以及 $B_1 \sim B_N$

5

→ 1 2 3 4 5

→ 2 4 7 11 3

題目賞析 - CodeForces 1133D

- 請問選出實數 D 做一個操作後形成新的數列 C

$$A_i \times D + B_i = C_i$$

- 請問這個實數 D 最多可以讓數列 C 有幾個 0 。

題目賞析 - CodeForces 1133D

• Input

5

1 2 3 4 5

2 4 7 11 3

• Input

3

13 37 39

1 2 3

• Output

2

• Output

2

D 選擇 -2

D 選擇-1/13



題目賞析 - CodeForces 1133D

- 列出 $A_i \times D_i + B_i = 0$ 的數列 D
- 看哪一個 D_i 重複最多次
- 為了避免浮點數誤差我們使用分數

索引

資料型態

- 這個分數在數列 D 中出現的次數

題目賞析 - CodeForces 1133D

- 這題有兩個難點：
 - 找出 **G**reatest **C**ommon **D**ivisor 使分子分母互質
 - 當有 0 出現的case



set

- 考慮下列情況：
- 在一篇文章中計算使用了哪些不同的字。

set 遍歷

```
int myints[] = {75,23,65,42,13,75,65};  
set<int> myset(myints,myints+7);  
  
for (auto it = myset.begin(); it != myset.end(); it++)  
    std::cout << ' ' << *it;
```

Output:

13 23 42 65 75

set 特性

$\{2, 4, 4, 4, 4, 6\}$ and $\{2, 4, 6\}$
是一樣的集合

常見的 set member function

- 新增元素 `set::insert`
- 元素存在查詢 `set::count`
- 元素刪除 `set::find`, `set::erase` 搭配使用

set 刪除

```
int myints[] = {75,23,65,42,13,75,65};  
set<int> myset(myints,myints+7);  
myset.erase(myset.find(65));  
  
for (auto it = myset.begin(); it != myset.end(); it++)  
    std::cout << ' ' << *it;
```

Output:

13 23 42 75

priority_queue

- 類似 queue 的元素使用方式
- 類似 set 的元素順序性

priority_queue

```
priority_queue<int> mypq;
mypq.push(30);
mypq.push(100);
mypq.push(25);
mypq.push(40);
while (!mypq.empty()) {
    int now = mypq.top(); mypq.pop();
    cout << ' ' << now;
}
```

Output:

100 40 30 25

常見的 `priority_queue` member function

- 跟 `queue` 很像
 - `priority_queue::push`
 - `priority_queue::pop`
 - `priority_queue::top`
 - `priority_queue::empty`

練習題

- a005: Good Cake Defender
- a006: Zero Quantity Maximization

參考解答

• a006:



Questions?

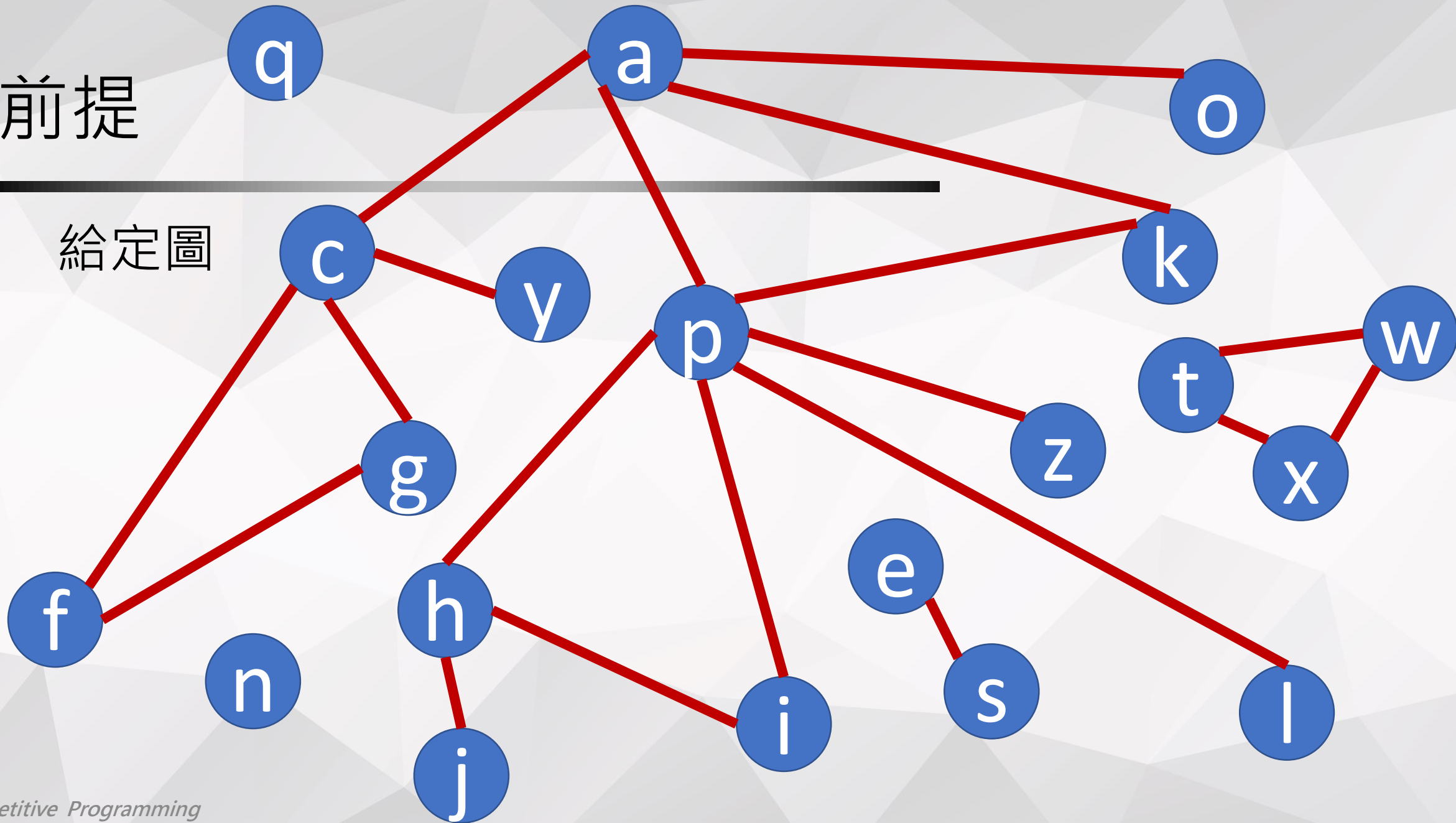
Outline

1. 深度優先搜尋 (Depth-First Search)
2. 廣度優先搜尋 (Breadth-First Search)



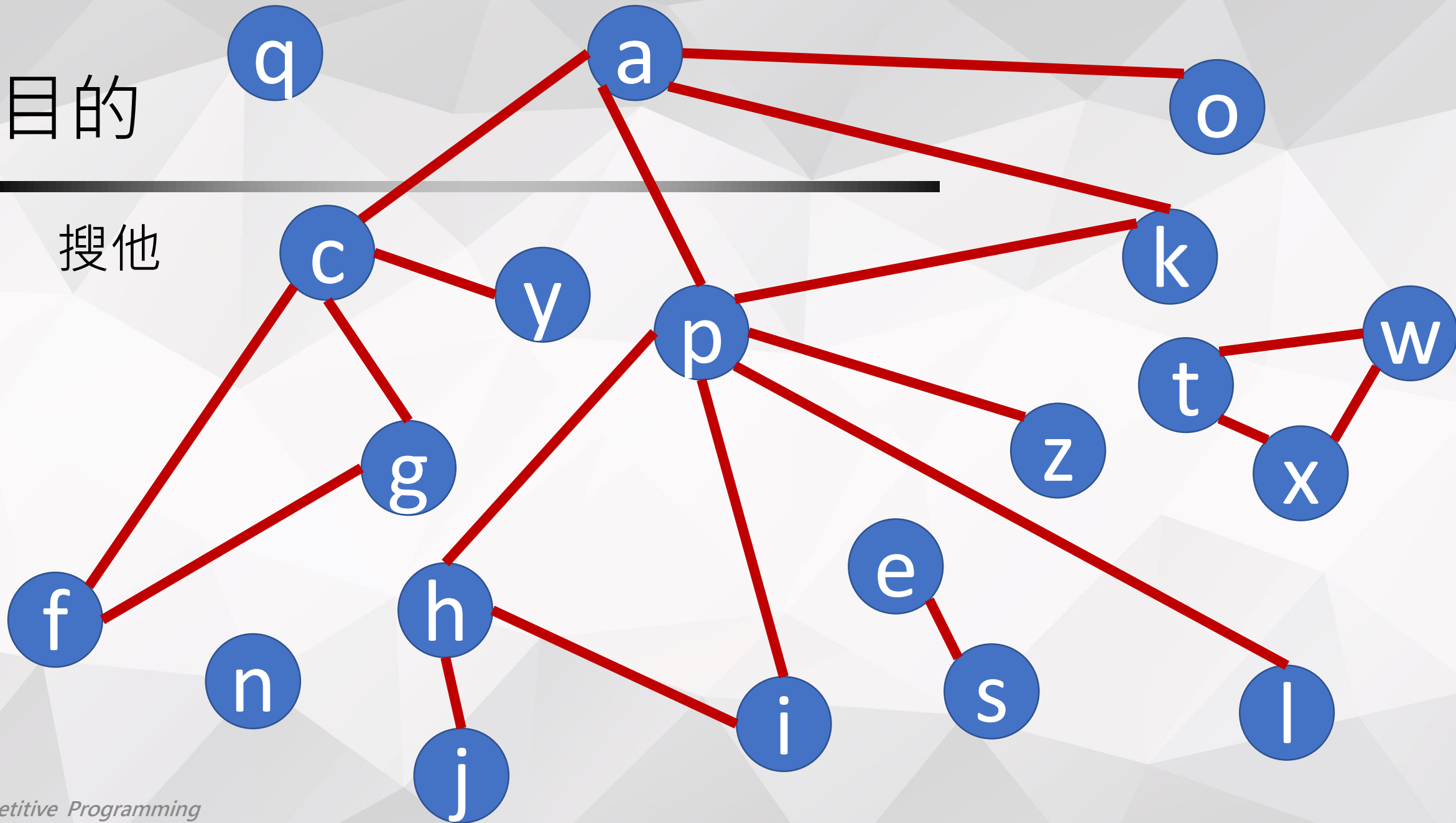
前提

給定圖



目的

搜他



通常

沒有想像中的美好

可能你的起點被限制的很嚴苛
離目標有一大段距離

寸步難行，要考慮很多條件
有些點可能是陷阱，
不能使用轉移水晶



深度優先搜尋

DFS

深度優先搜尋 (Depth-First Search) 簡稱 DFS



DFS 的點遍歷順序

為每拜訪一個**未曾拜訪**節點 (拜訪中)
就往**其一鄰點**拜訪過去

當**拜訪完**此節點，返回到父節點

*節點: DFS 遍歷完會產生一顆樹

*某節點拜訪完: 其子孫節點都拜訪完

DFS 實作

```
void dfs(int u, int dep) { // dep := depth
    for (auto v: E[u]) {
        if (vis[v]) continue;
        vis[v] = true;
        dfs(v, dep+1);
    }
}
```

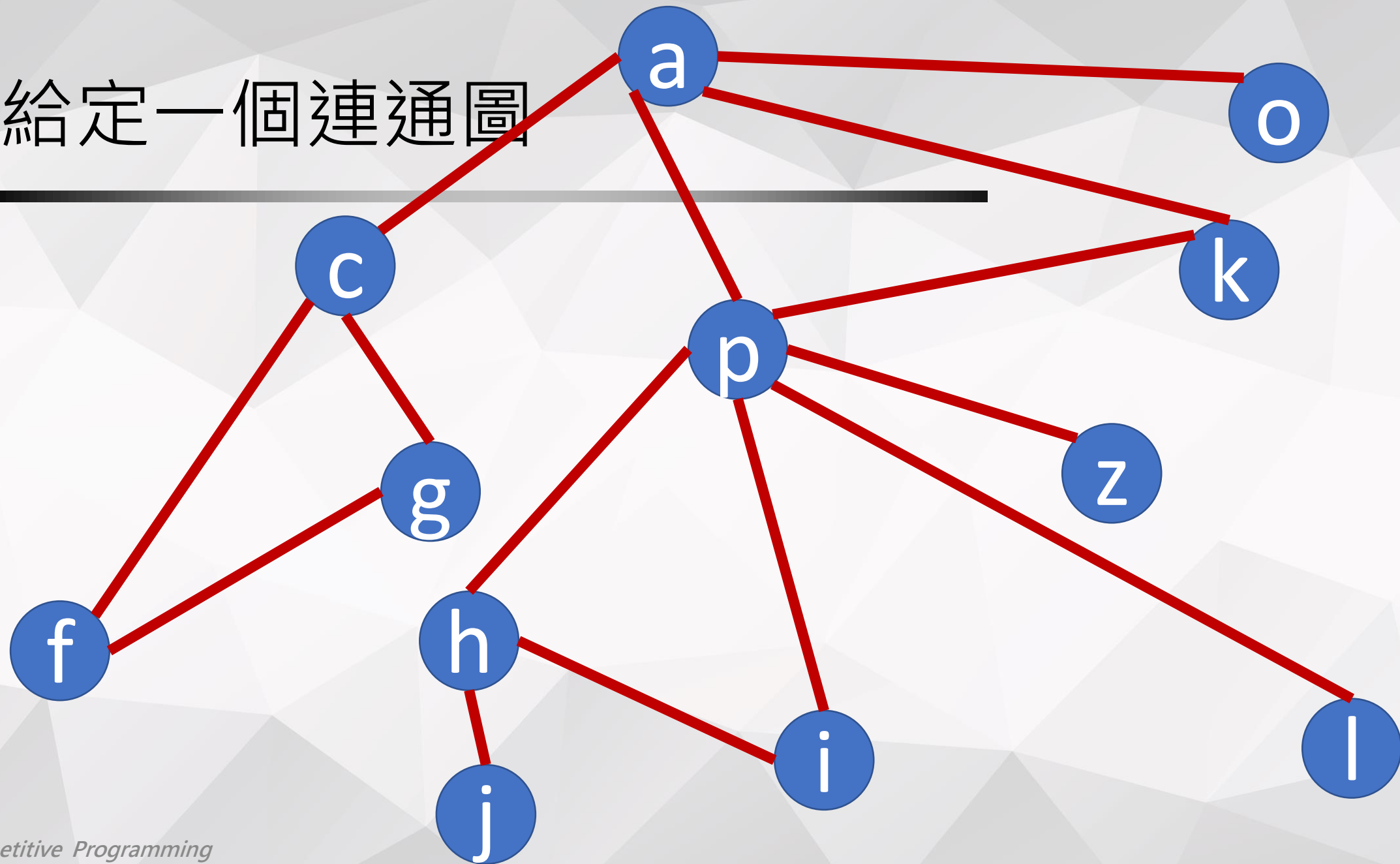


DFS 實作 (非遞迴)

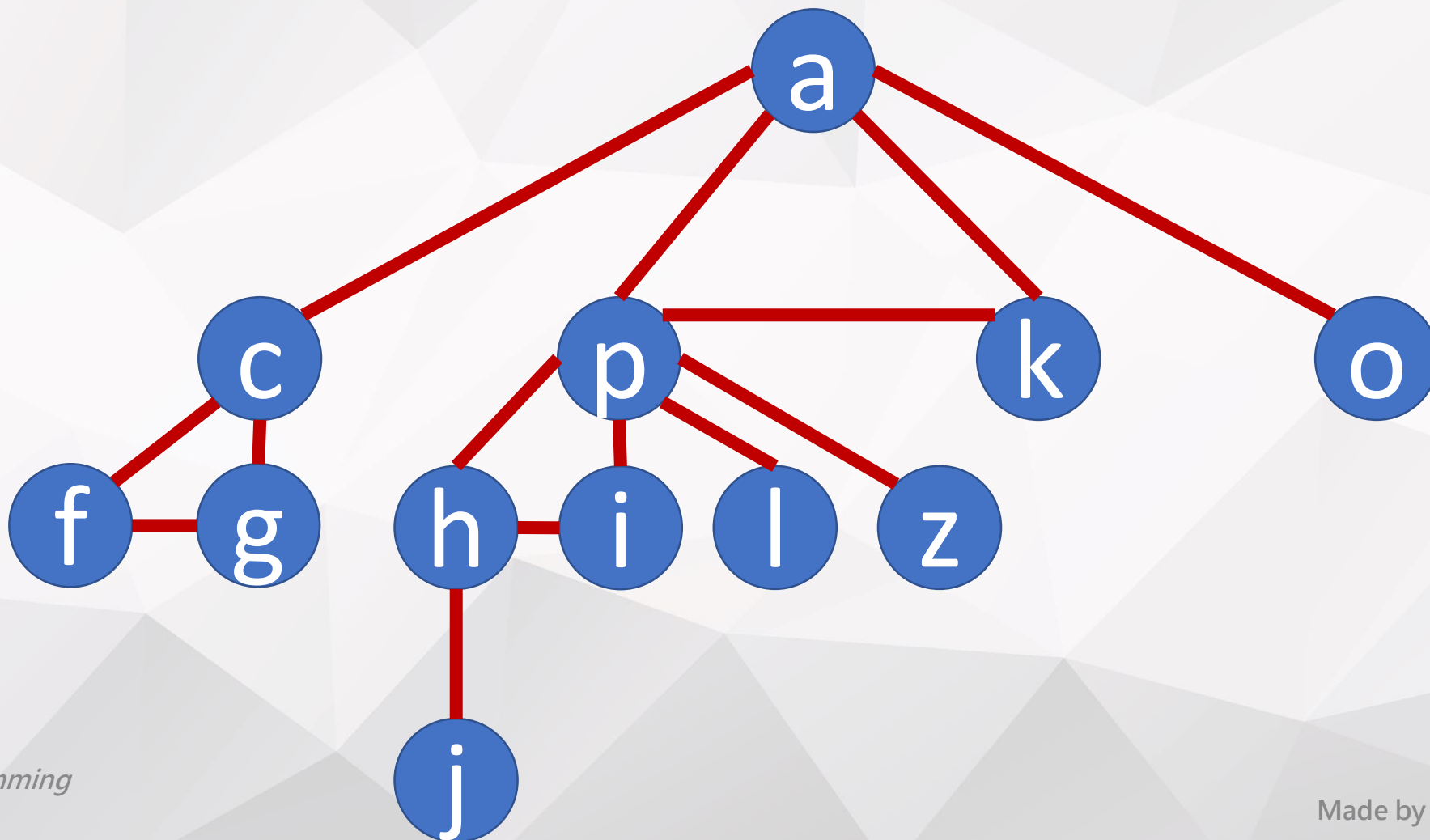
```
stack<int> S; // 此處少記錄一個 dep  
S.push(root); // root 代表走訪此圖的起點  
vis[root] = true;
```

```
while (!S.empty()) {  
    int u = S.top(); S.pop();  
    for (auto v: E[u]) {  
        if (vis[v]) continue;  
        vis[v] = true;  
        S.push(v);  
    }  
}
```

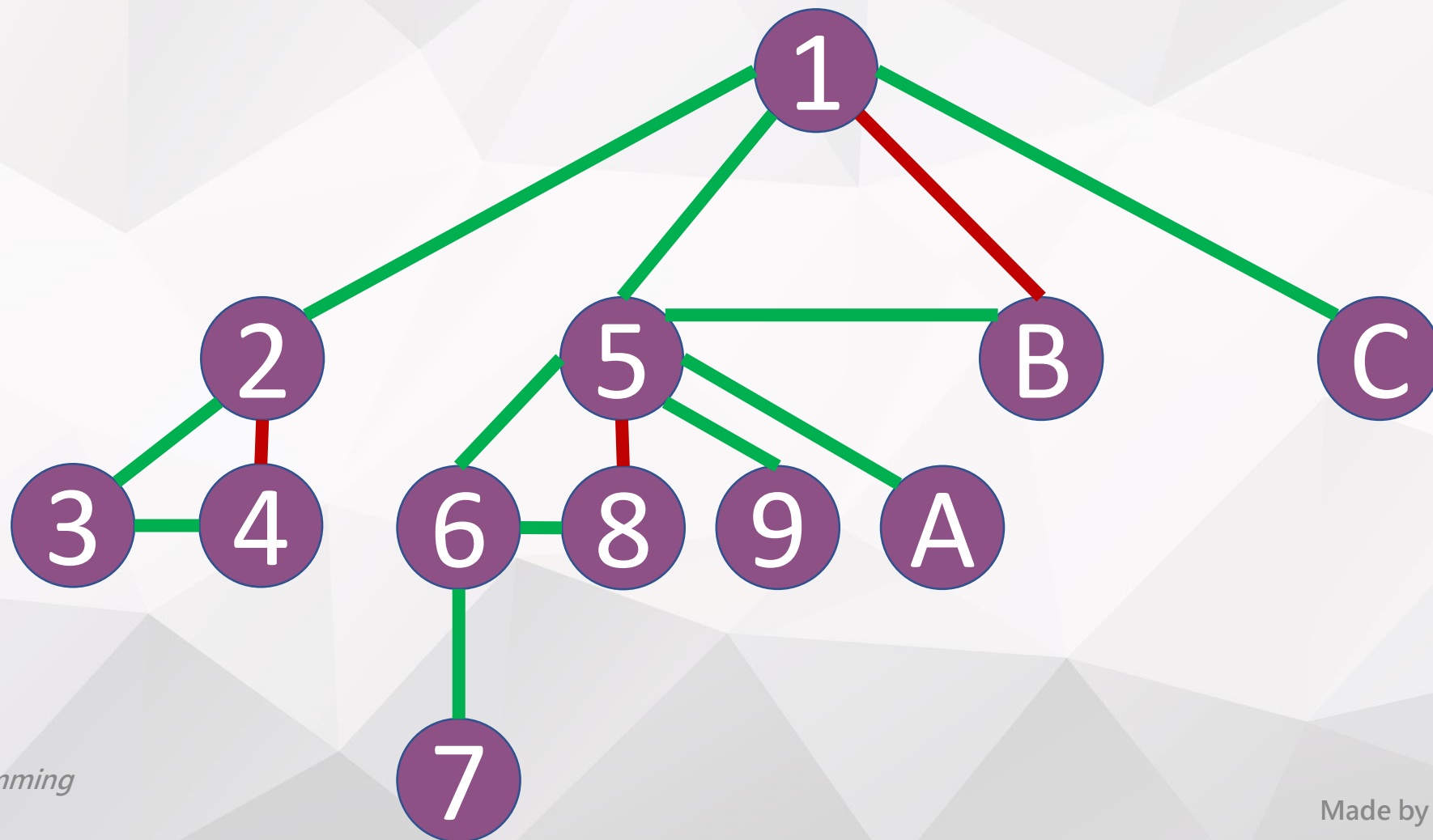
給定一個連通圖



整理一下

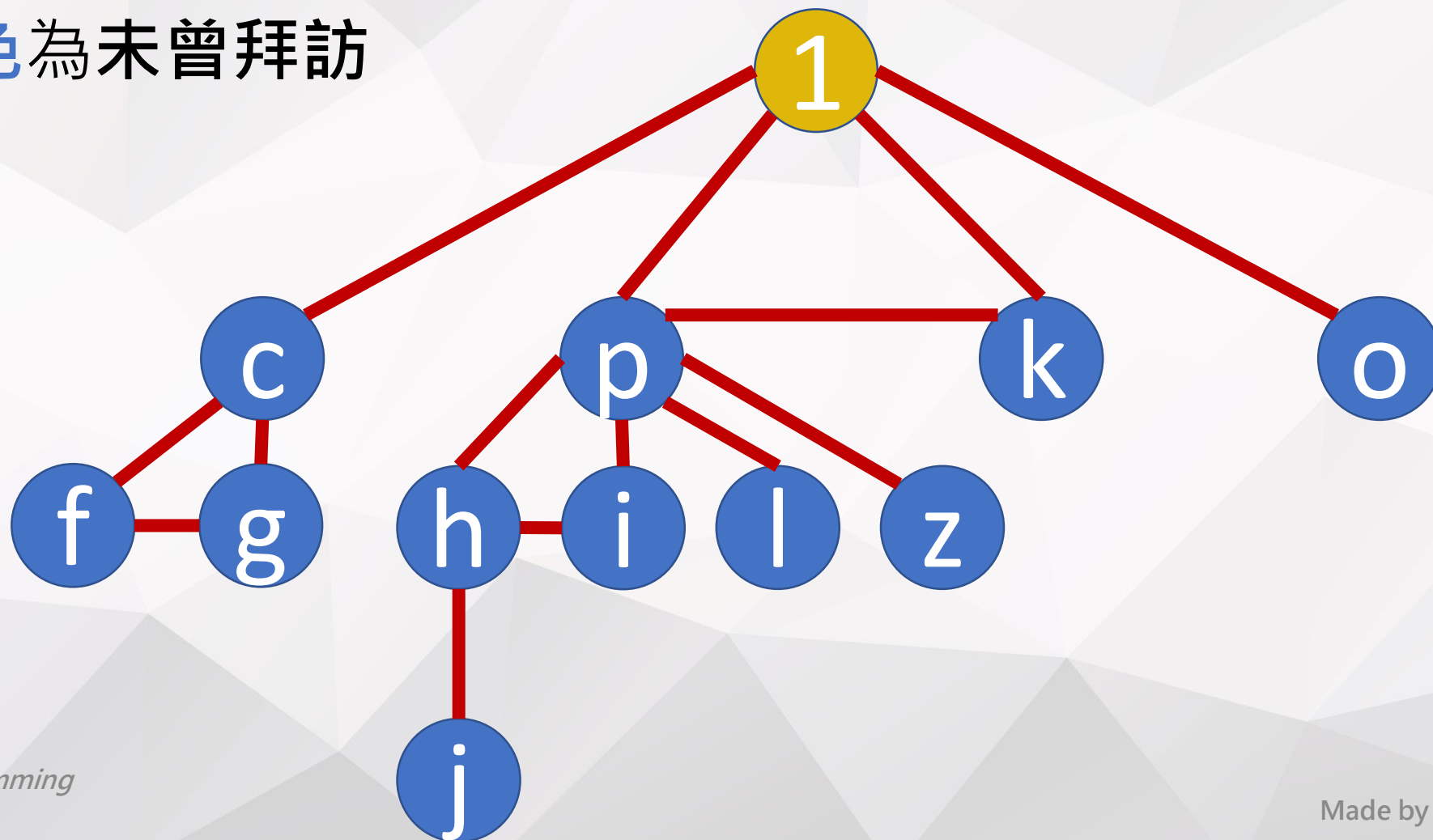


DFS 的點遍歷順序



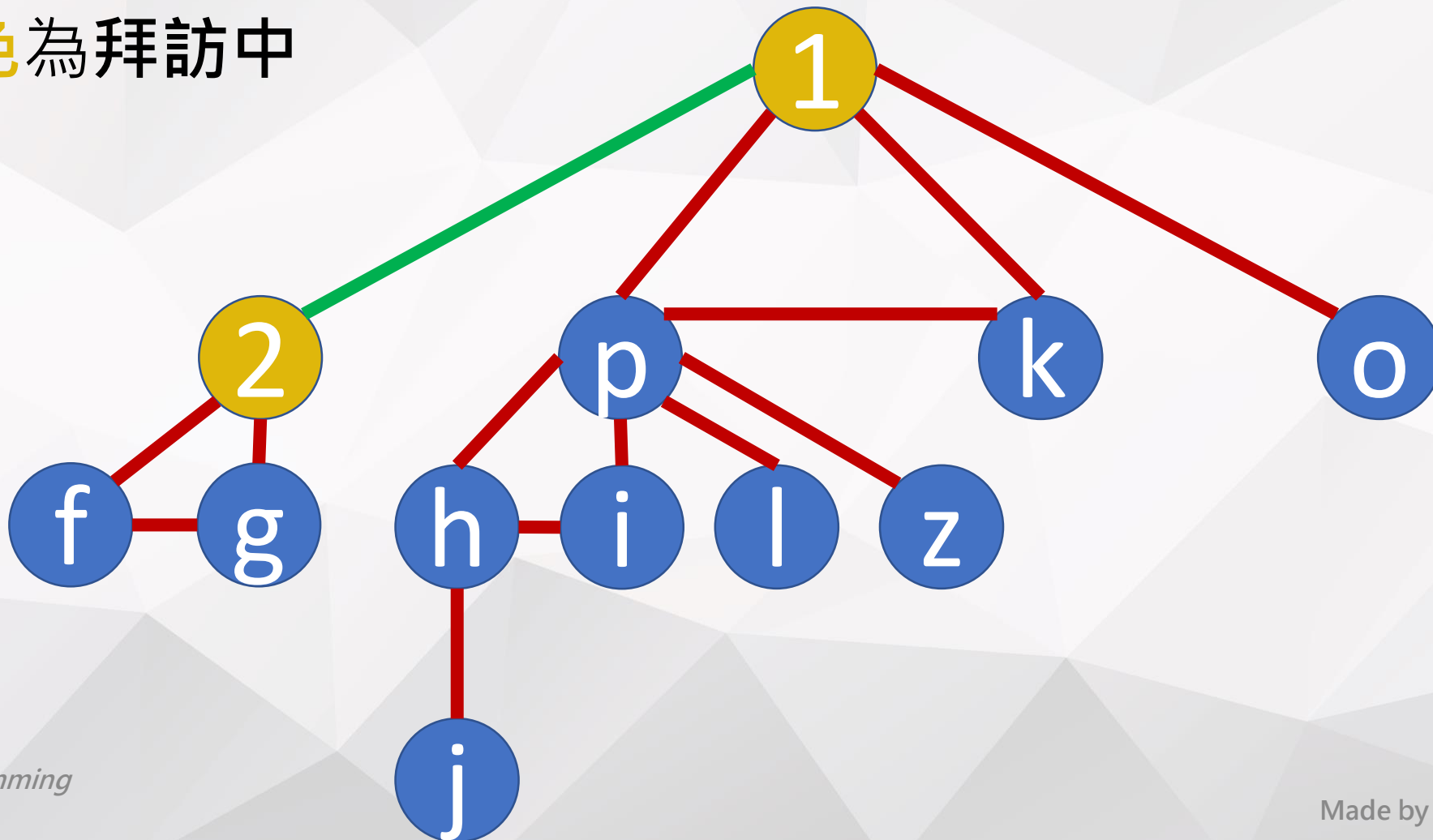
第一個拜訪的為根

藍色為未曾拜訪

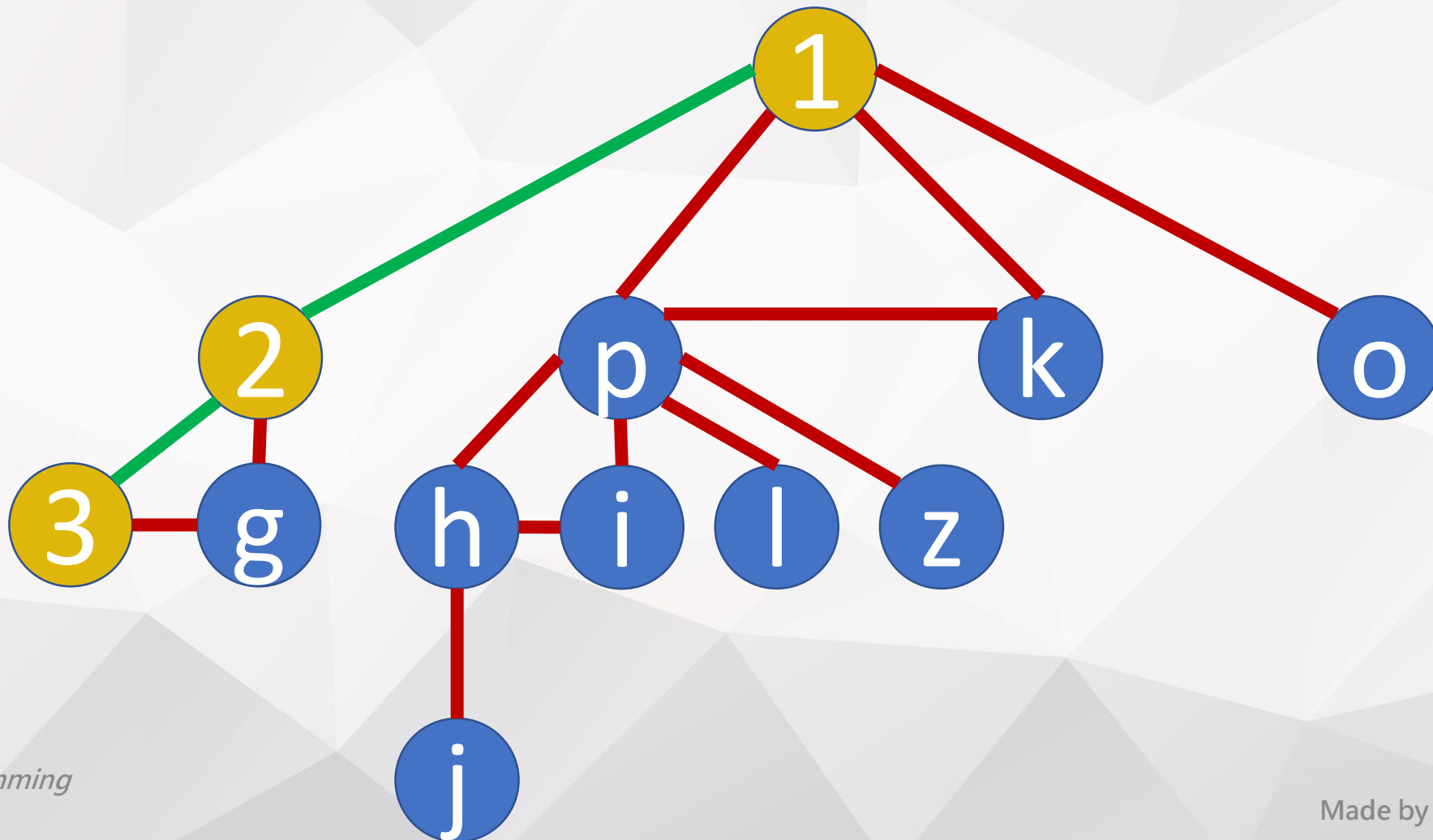


拜訪鄰點

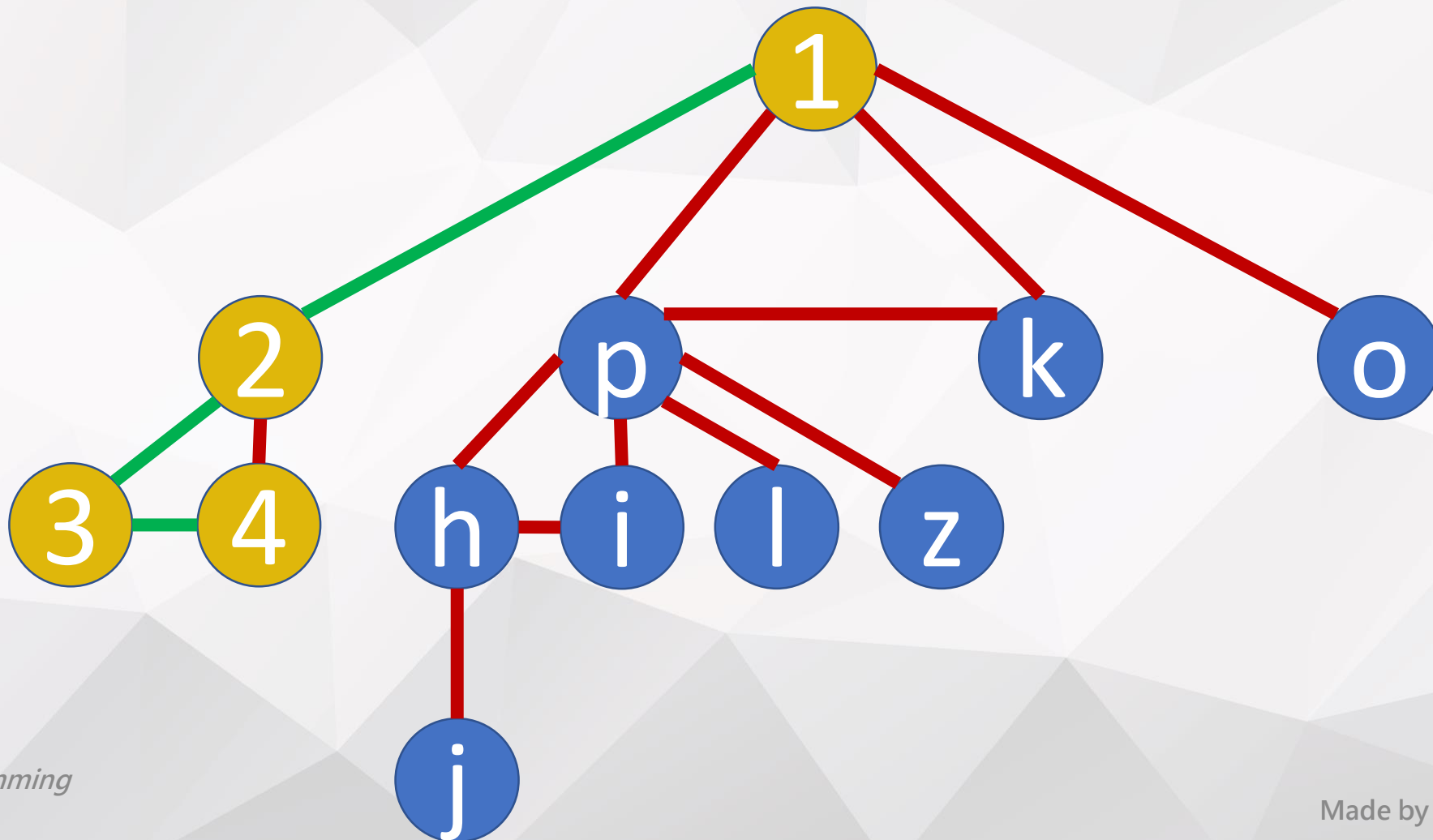
黃色為拜訪中



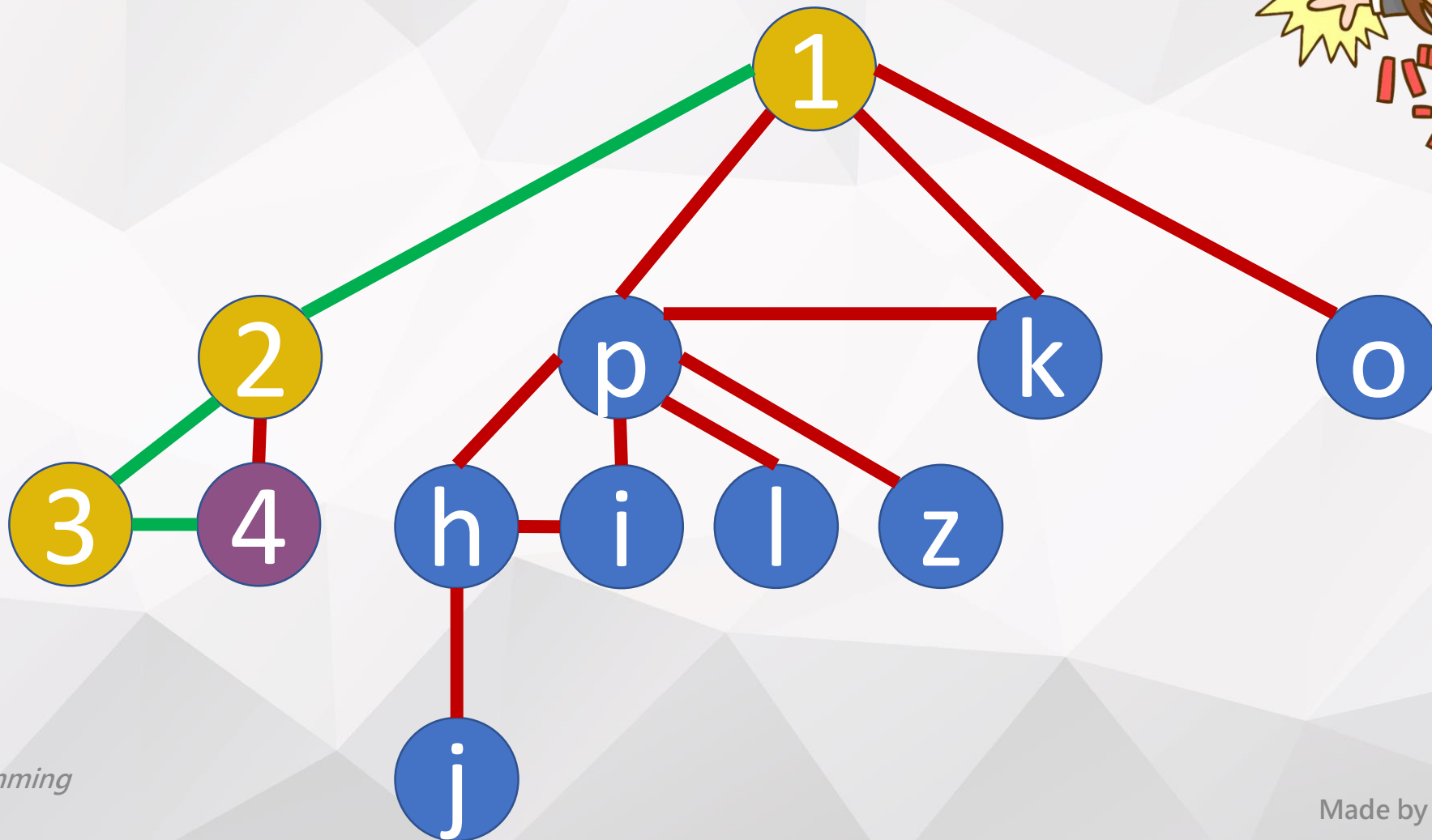
拜訪鄰點



拜訪鄰點

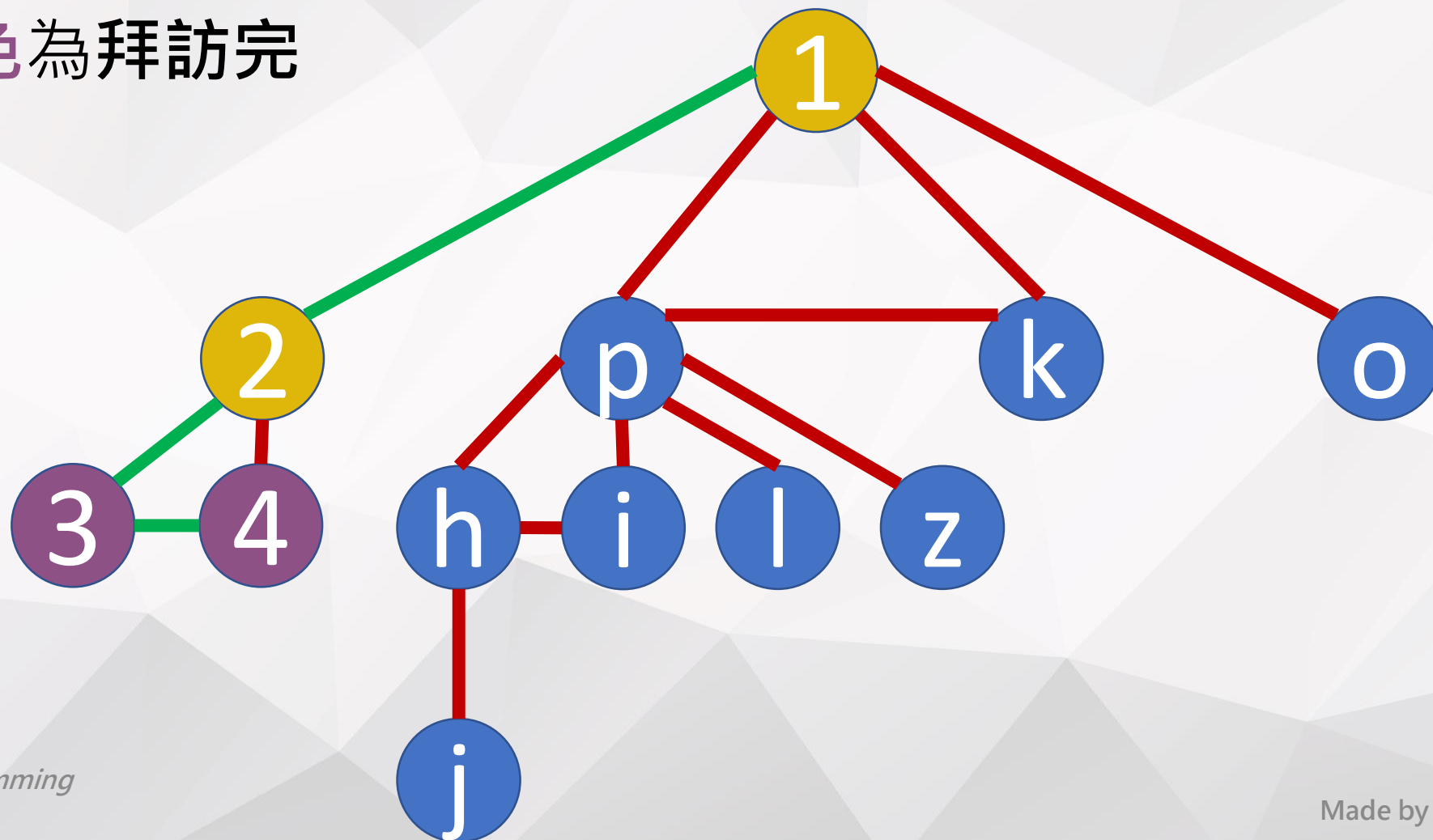


拜訪完

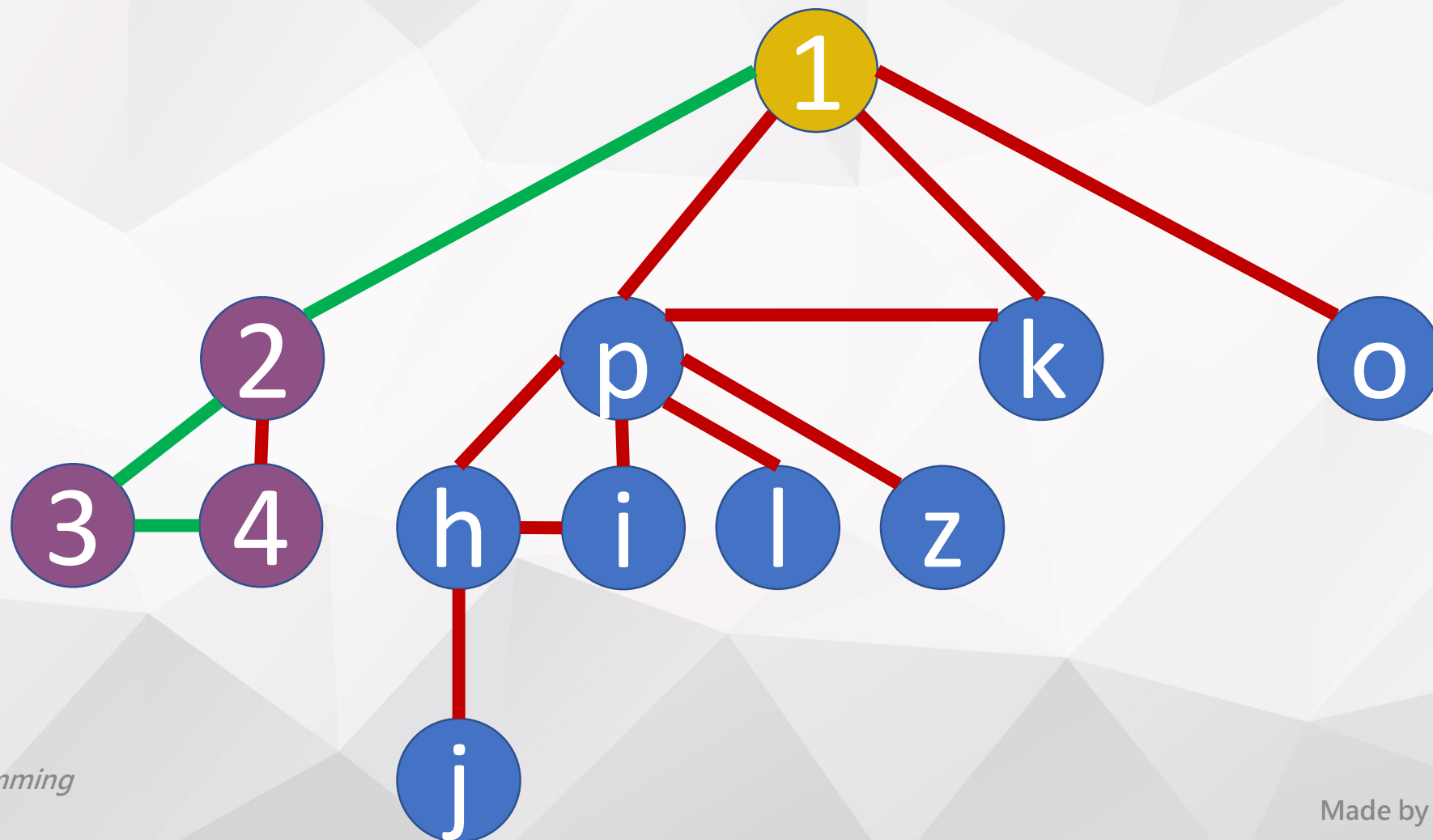


拜訪完

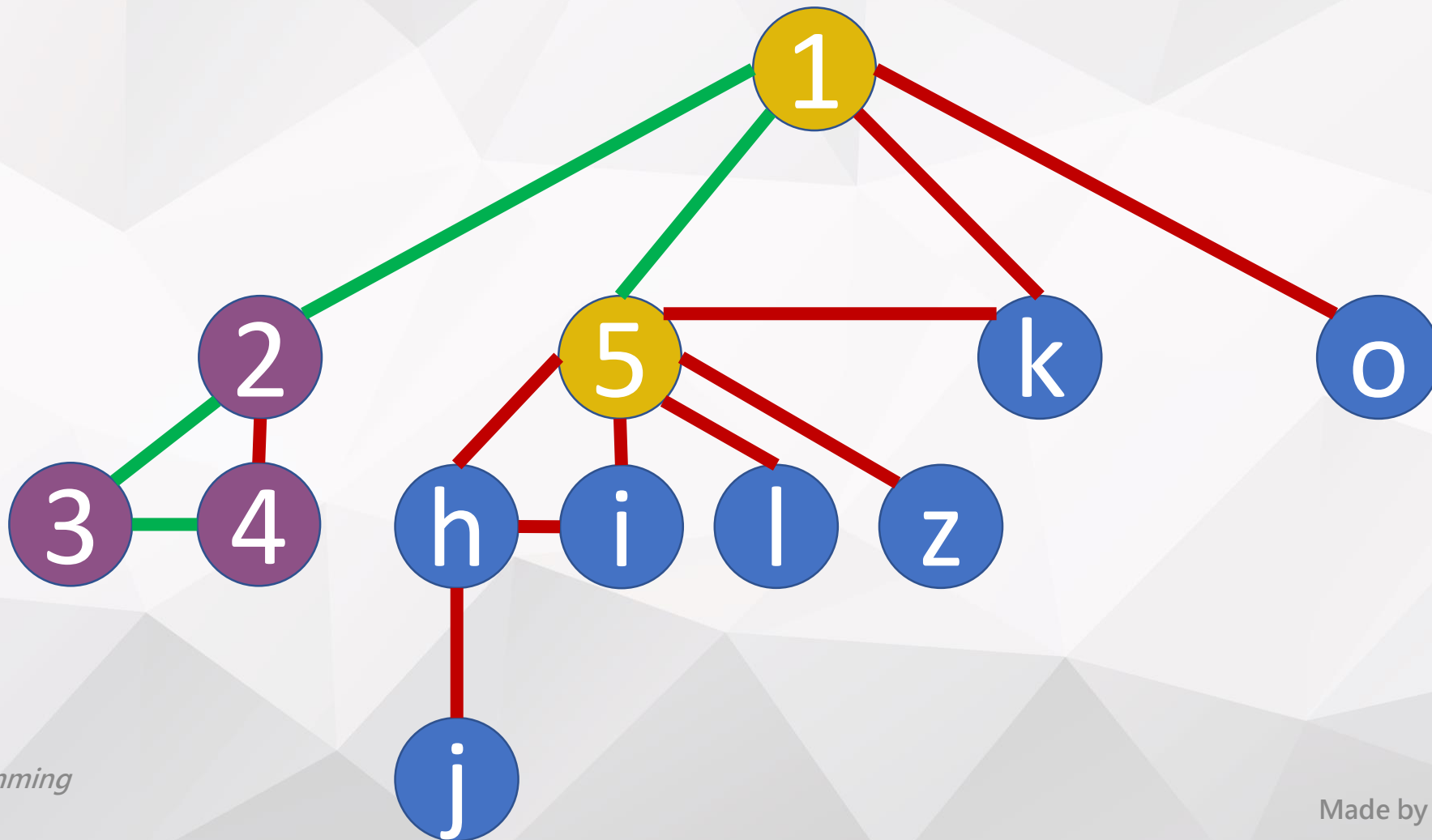
紫色為拜訪完



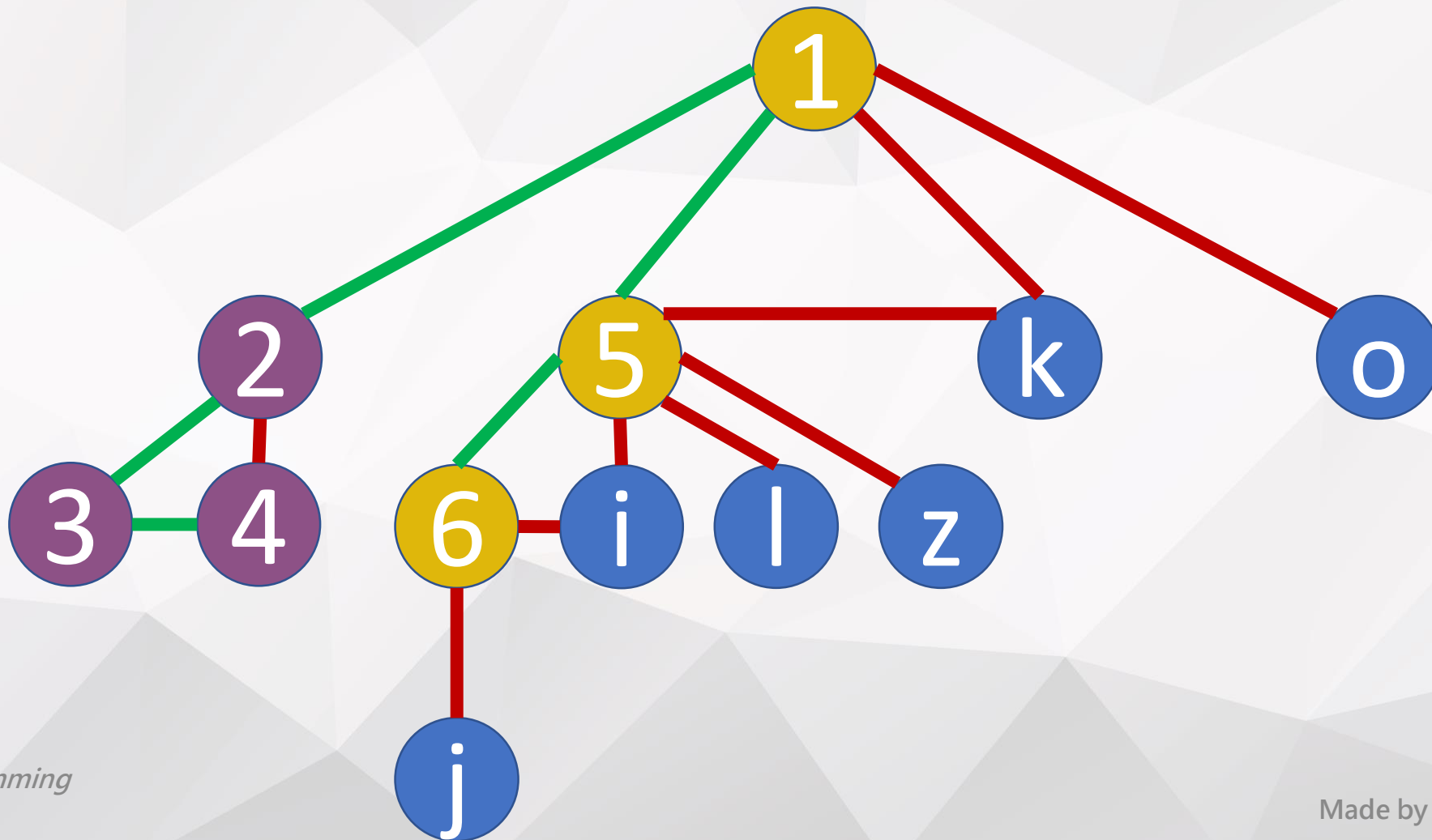
拜訪完



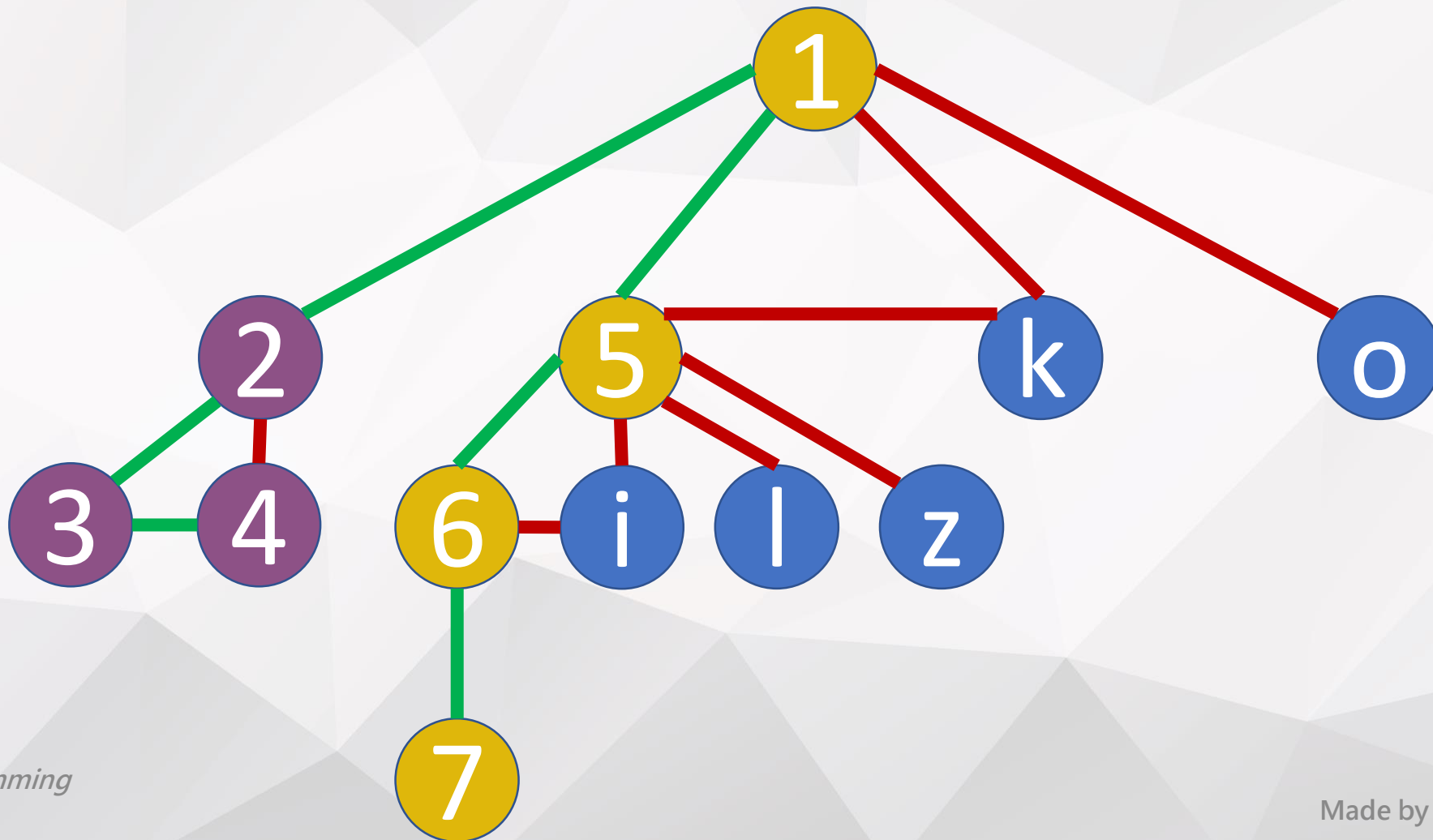
拜訪鄰點



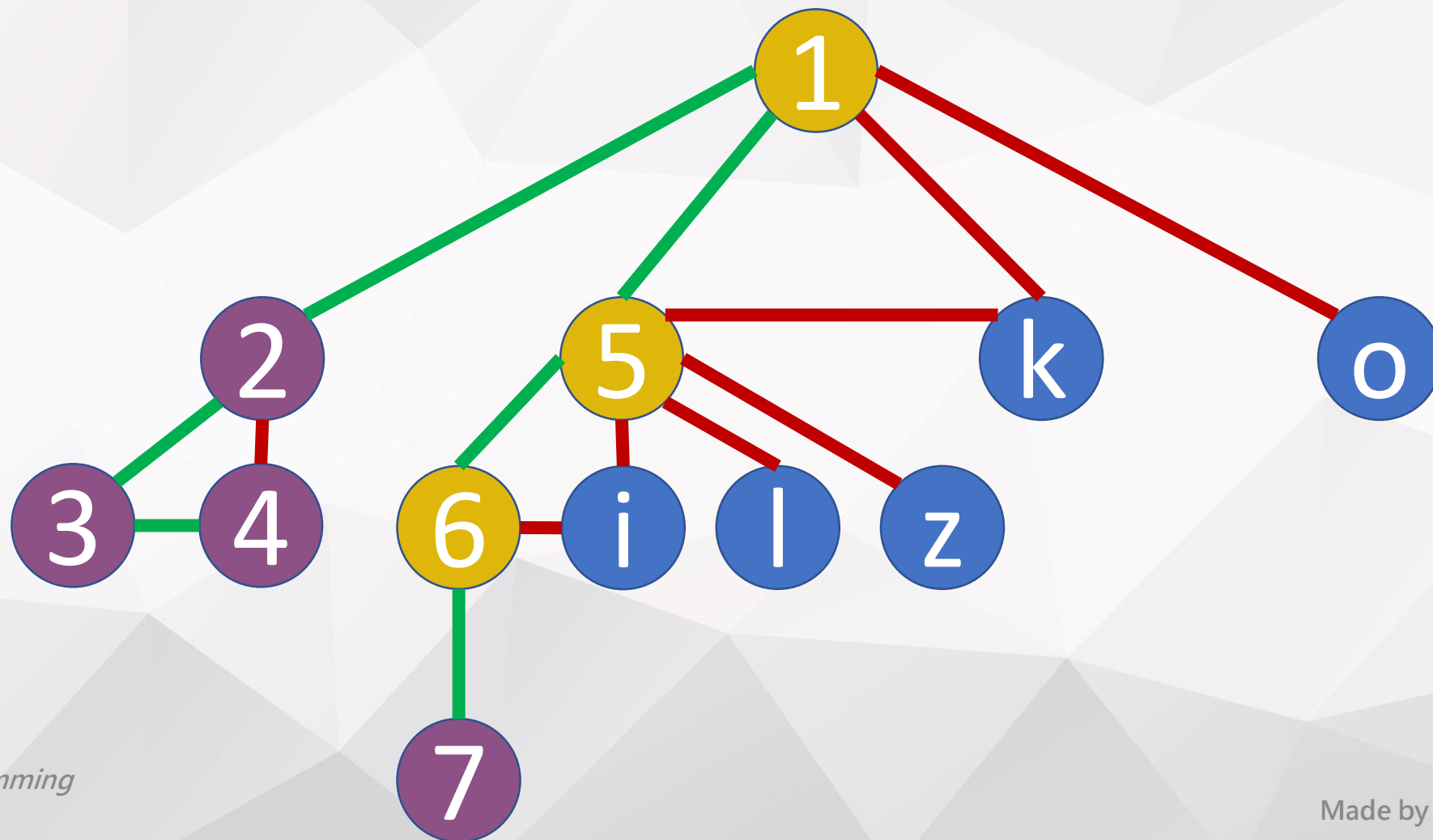
拜訪鄰點



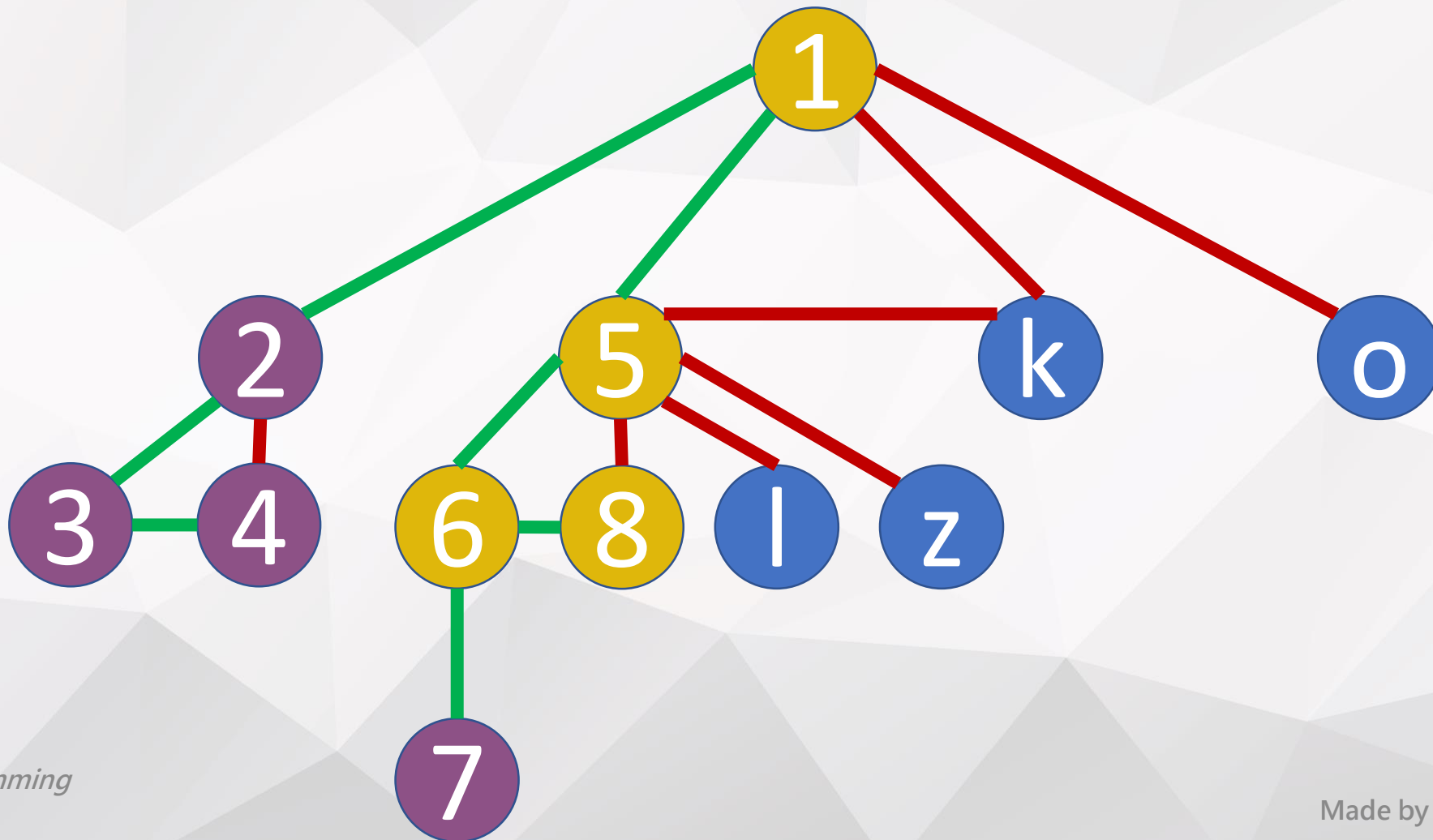
拜訪鄰點



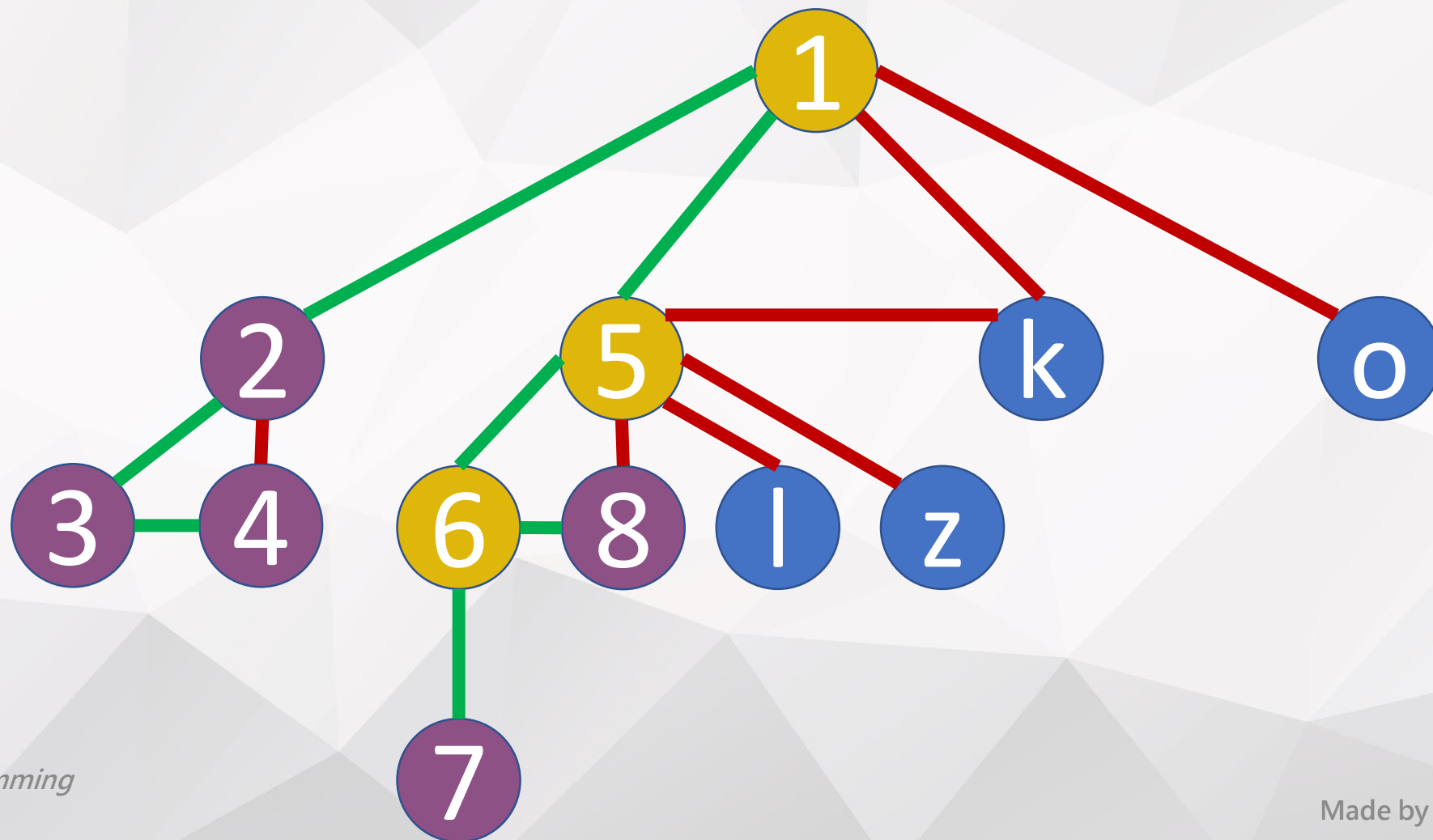
拜訪完



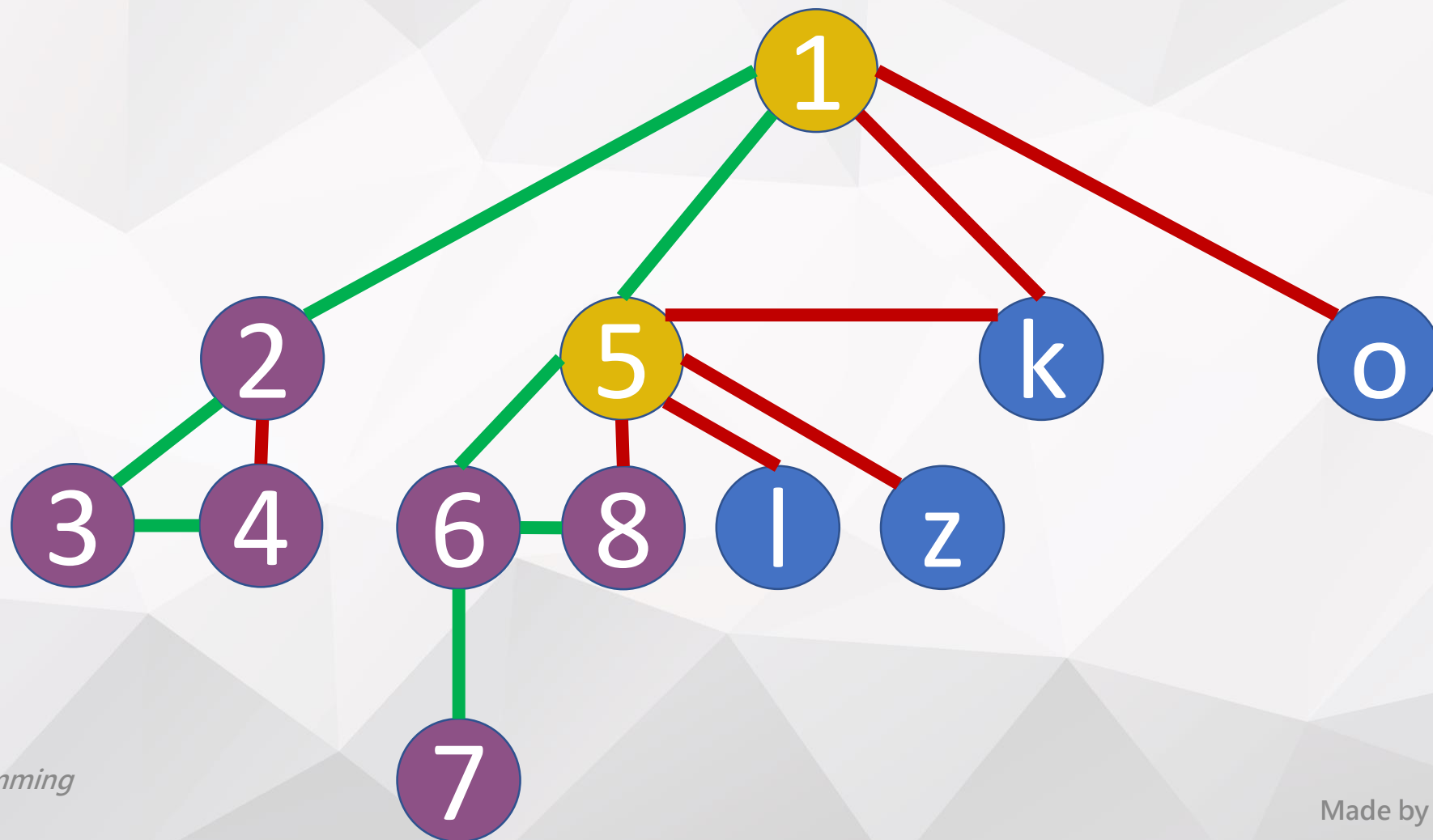
拜訪鄰點



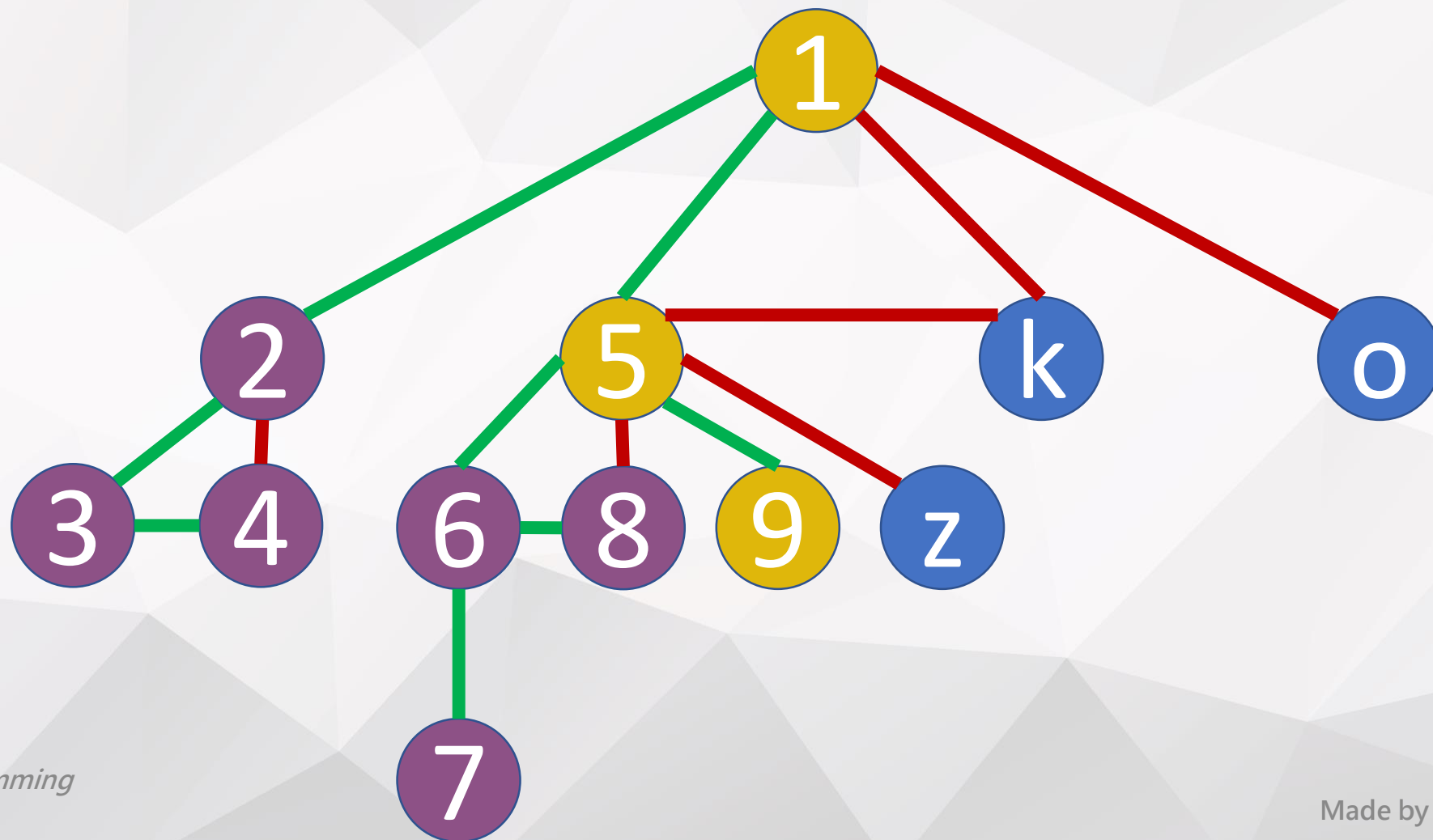
拜訪完



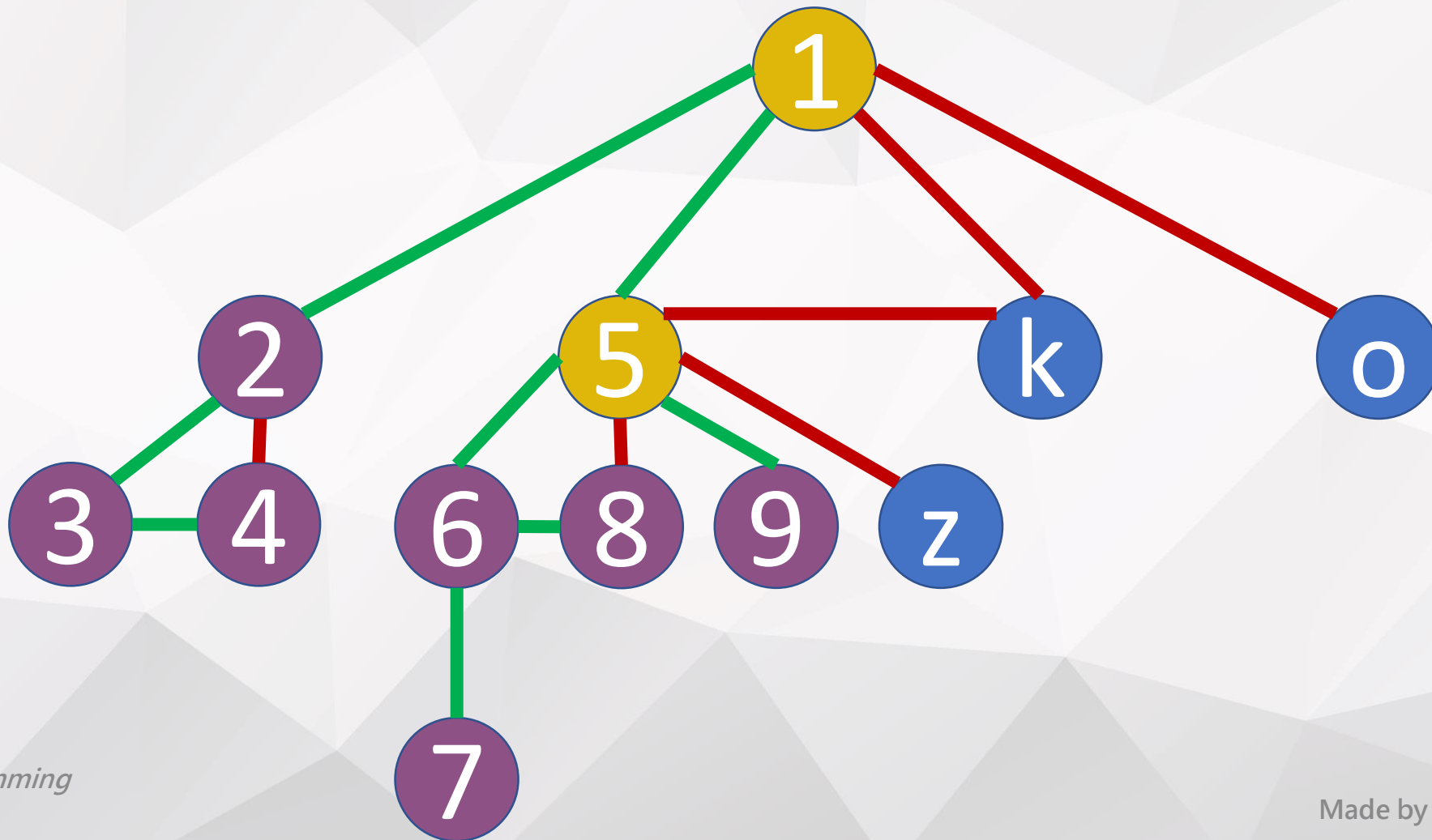
拜訪完



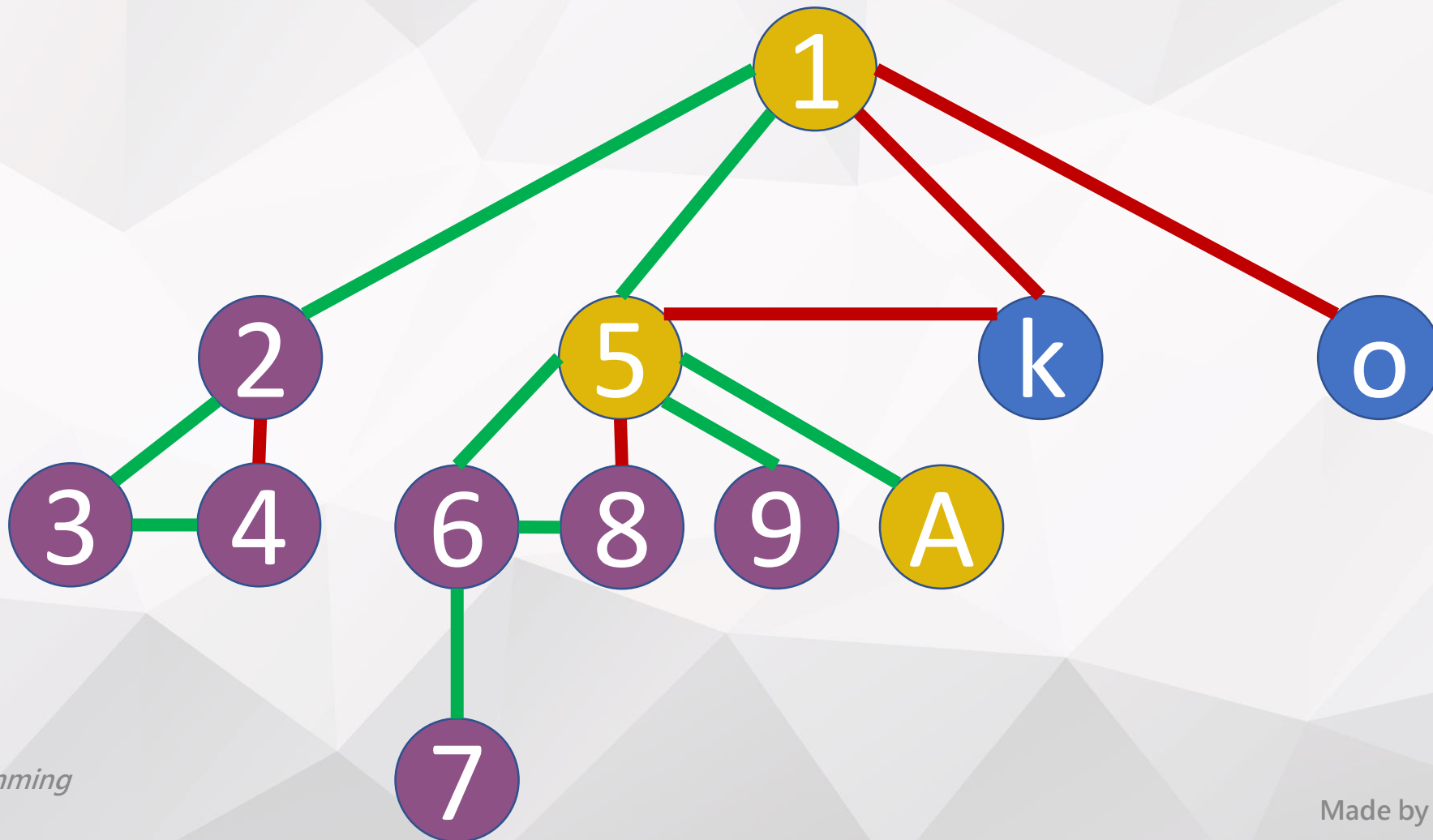
拜訪鄰點



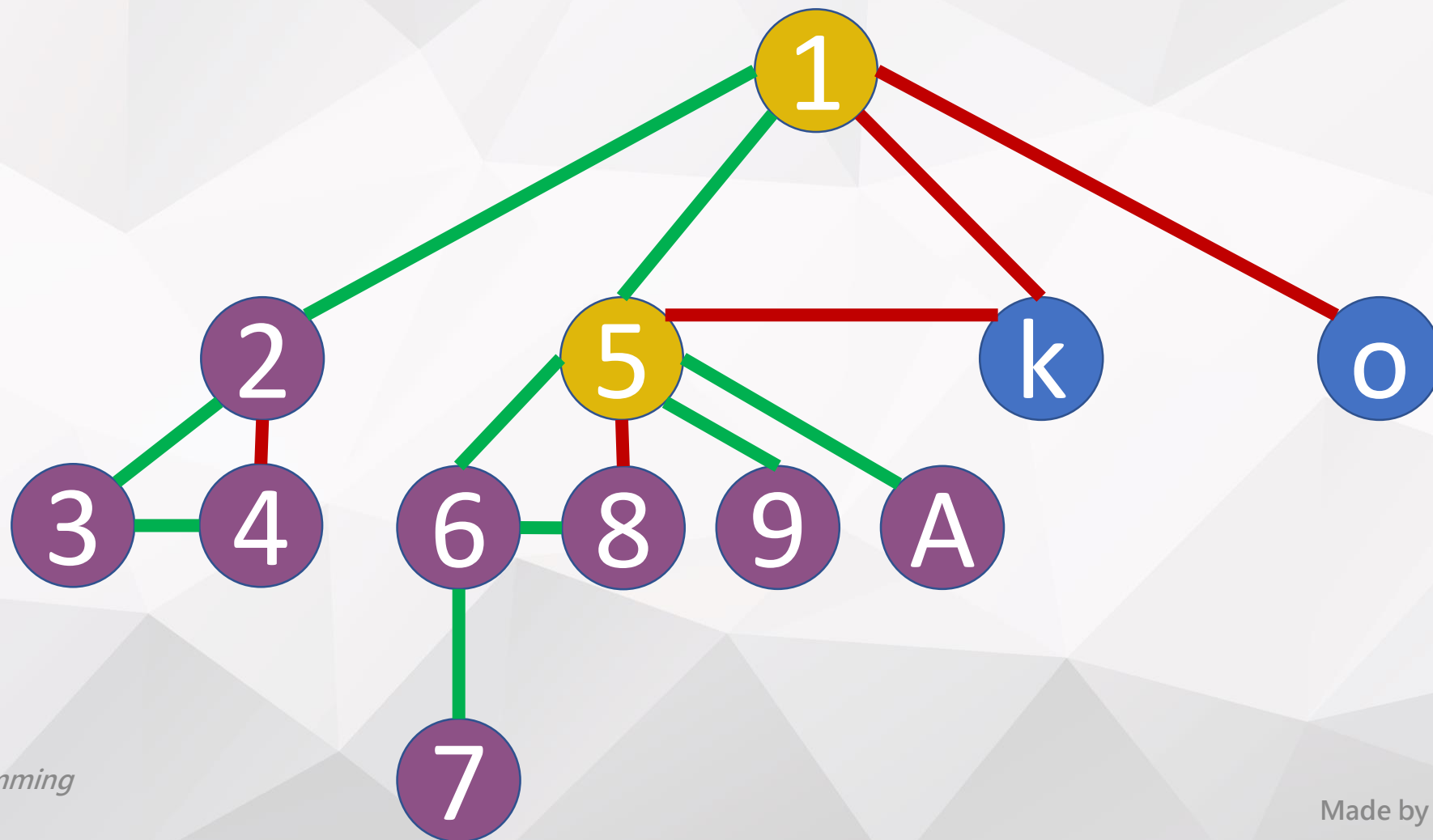
拜訪完



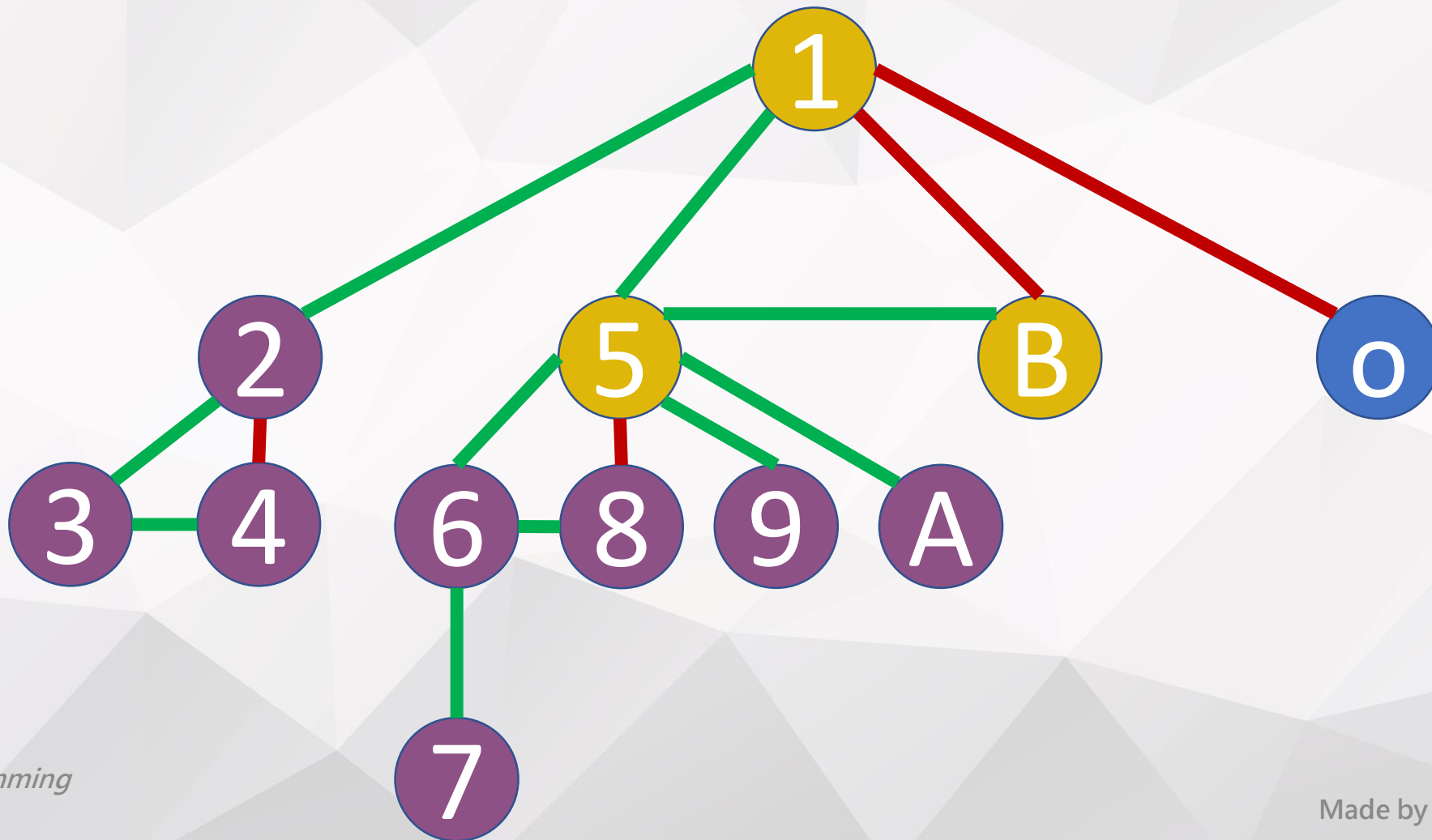
拜訪鄰點



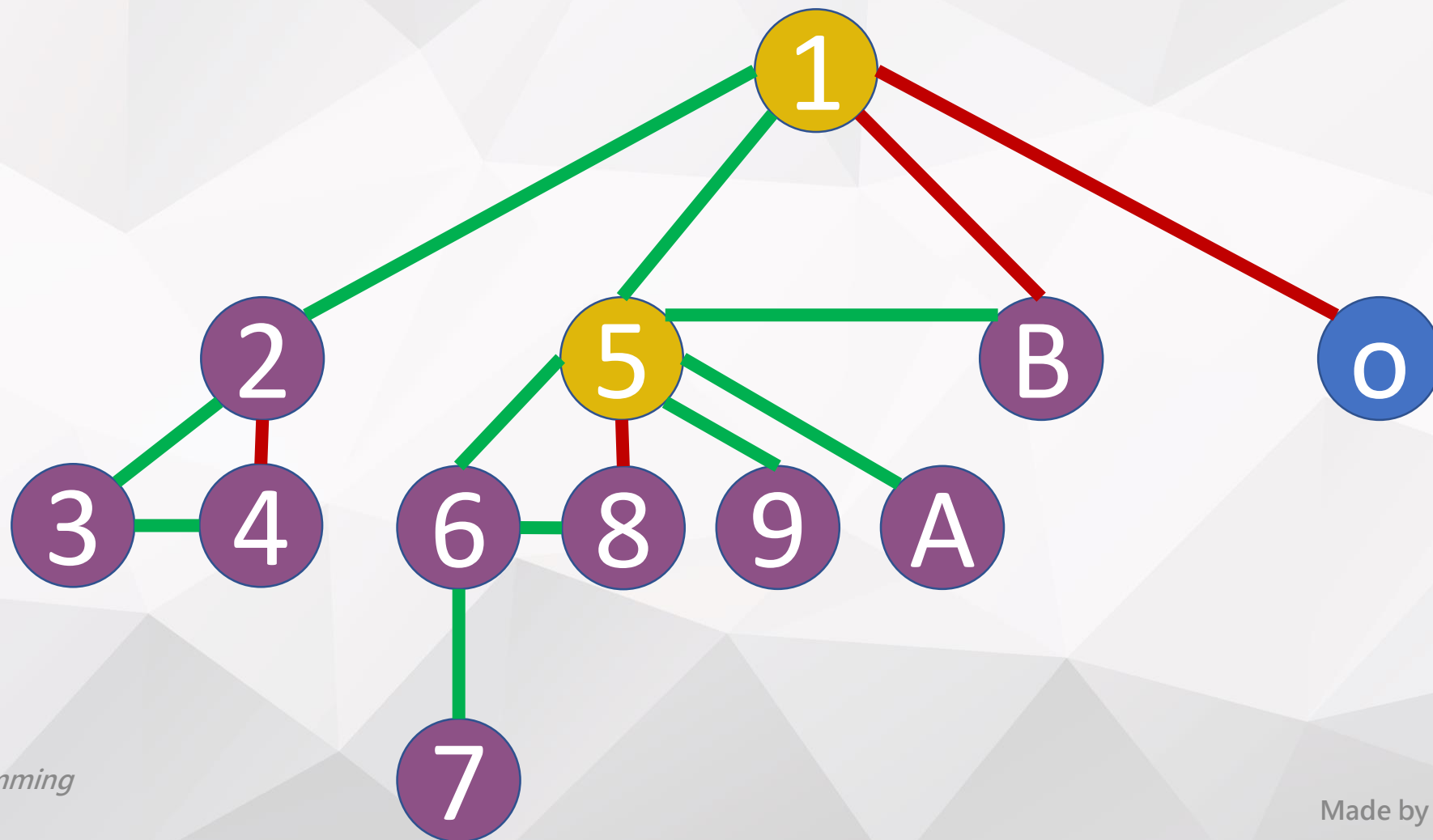
拜訪完



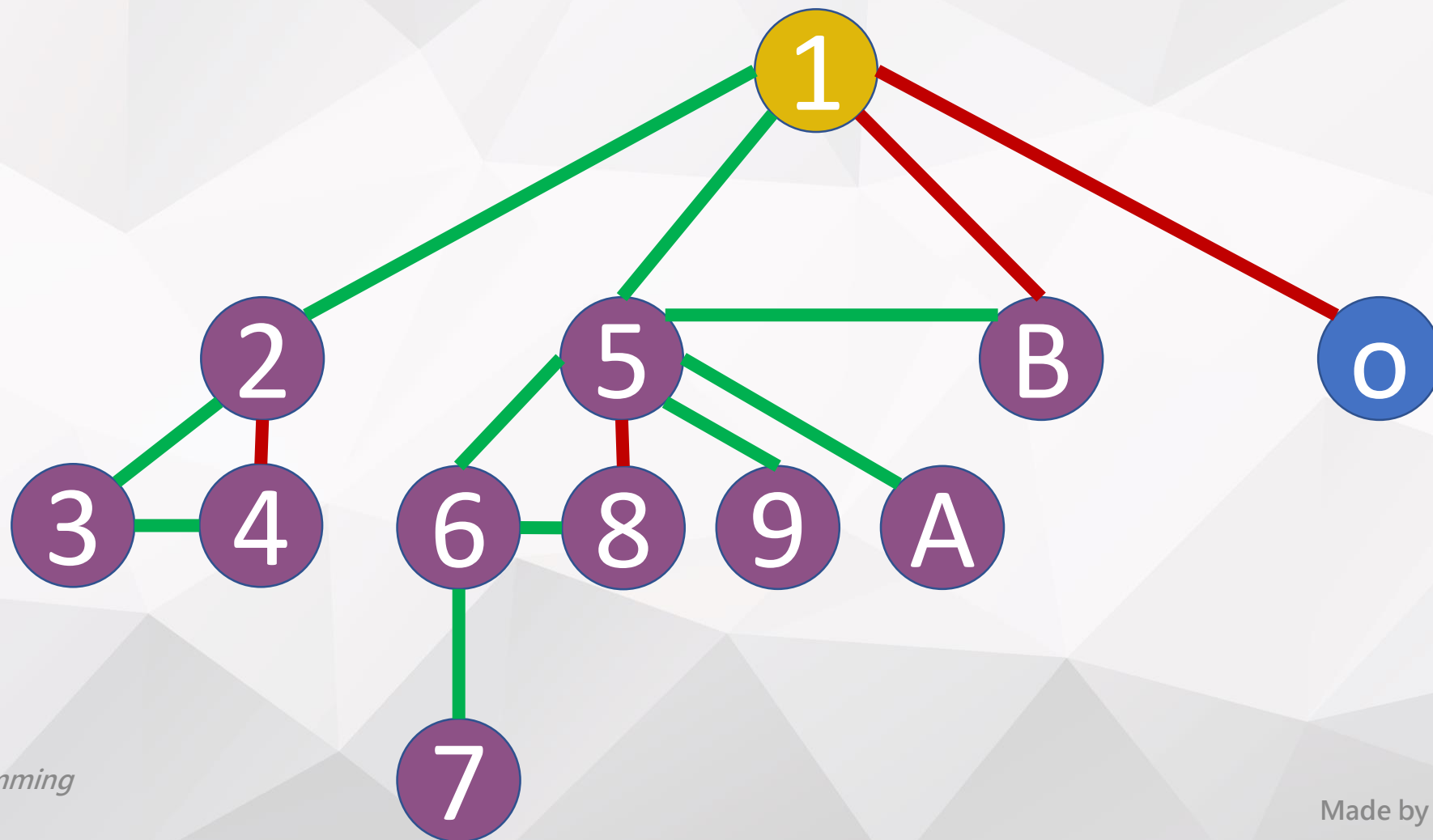
拜訪鄰點



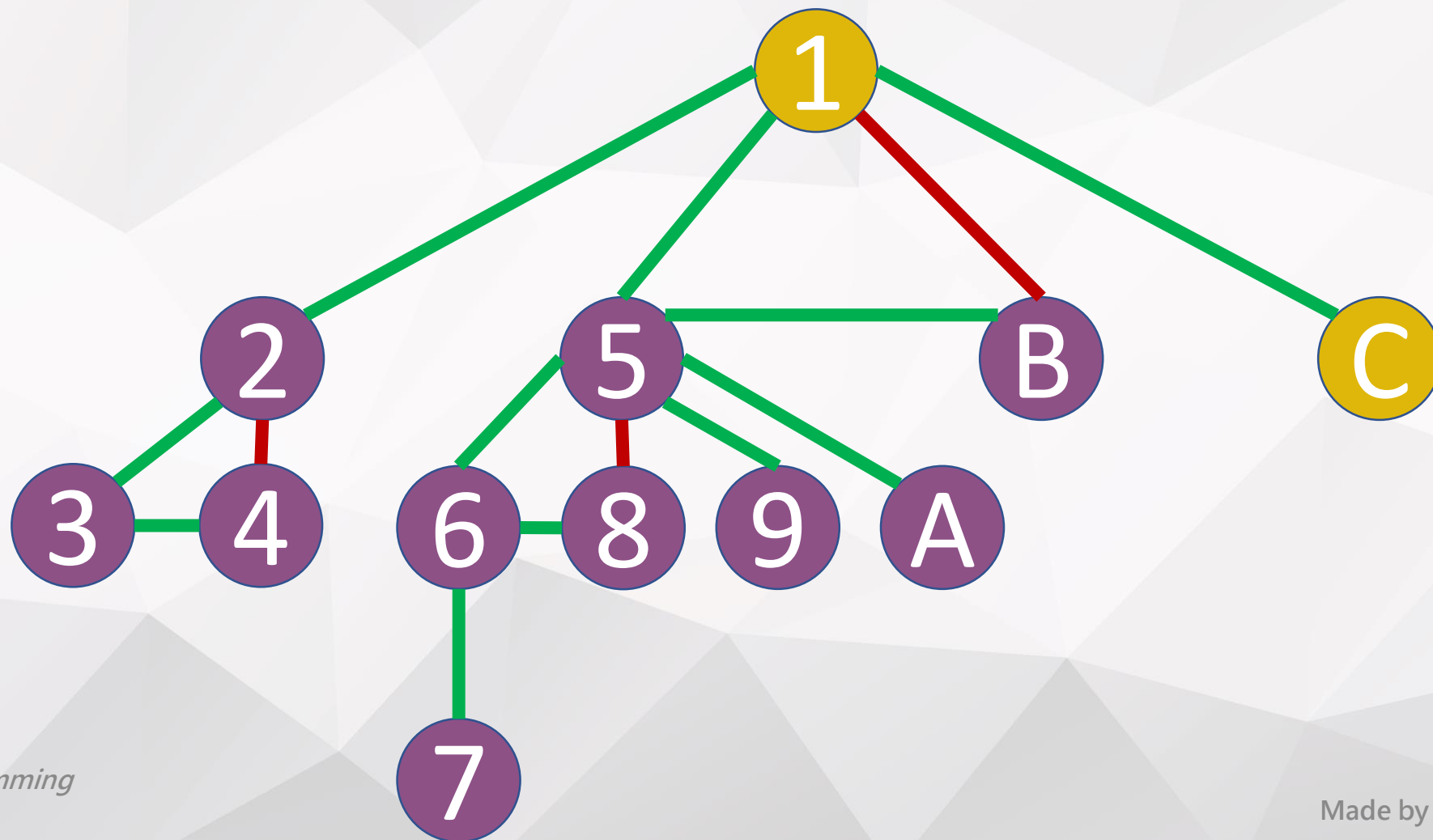
拜訪完



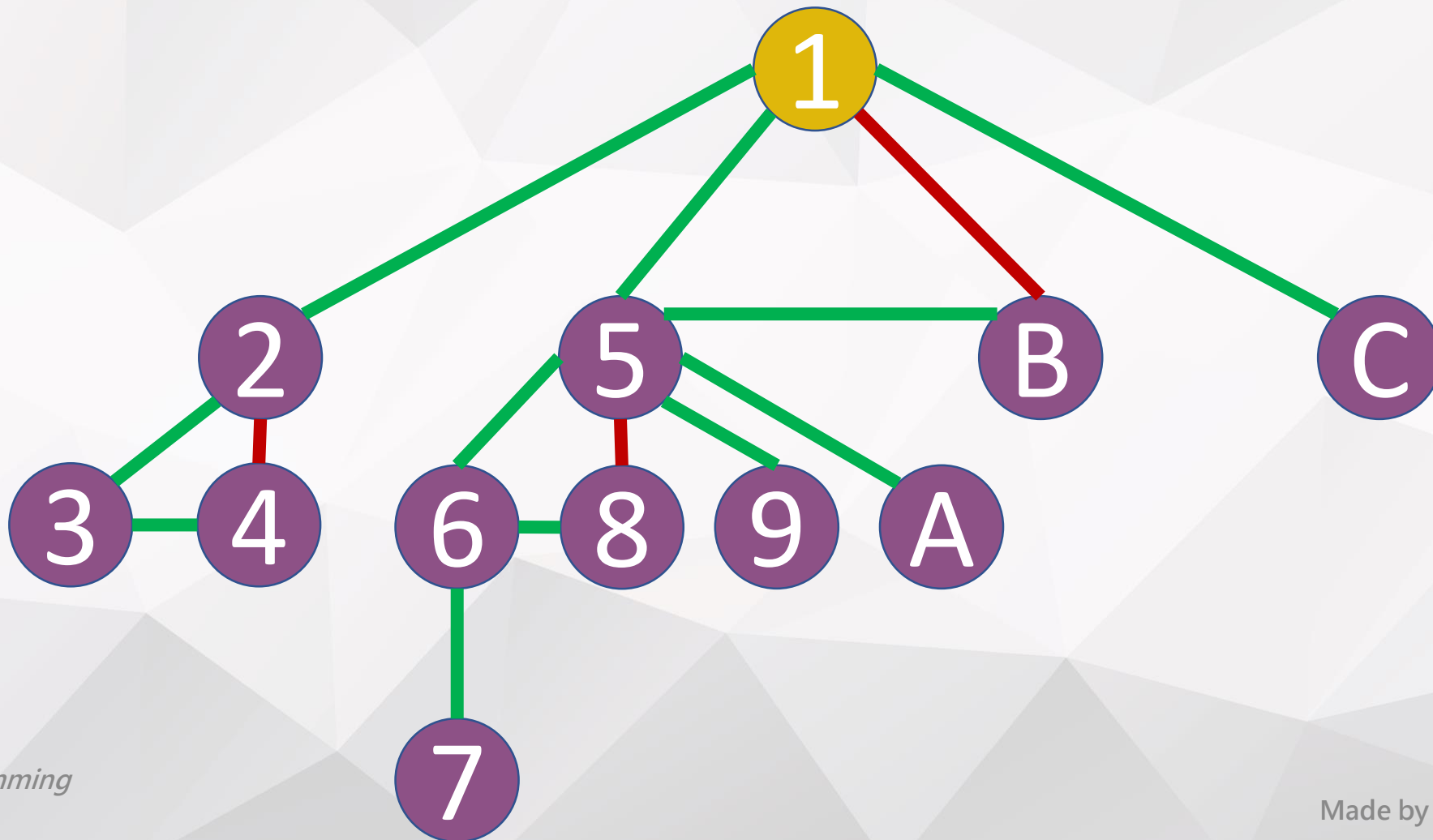
拜訪完



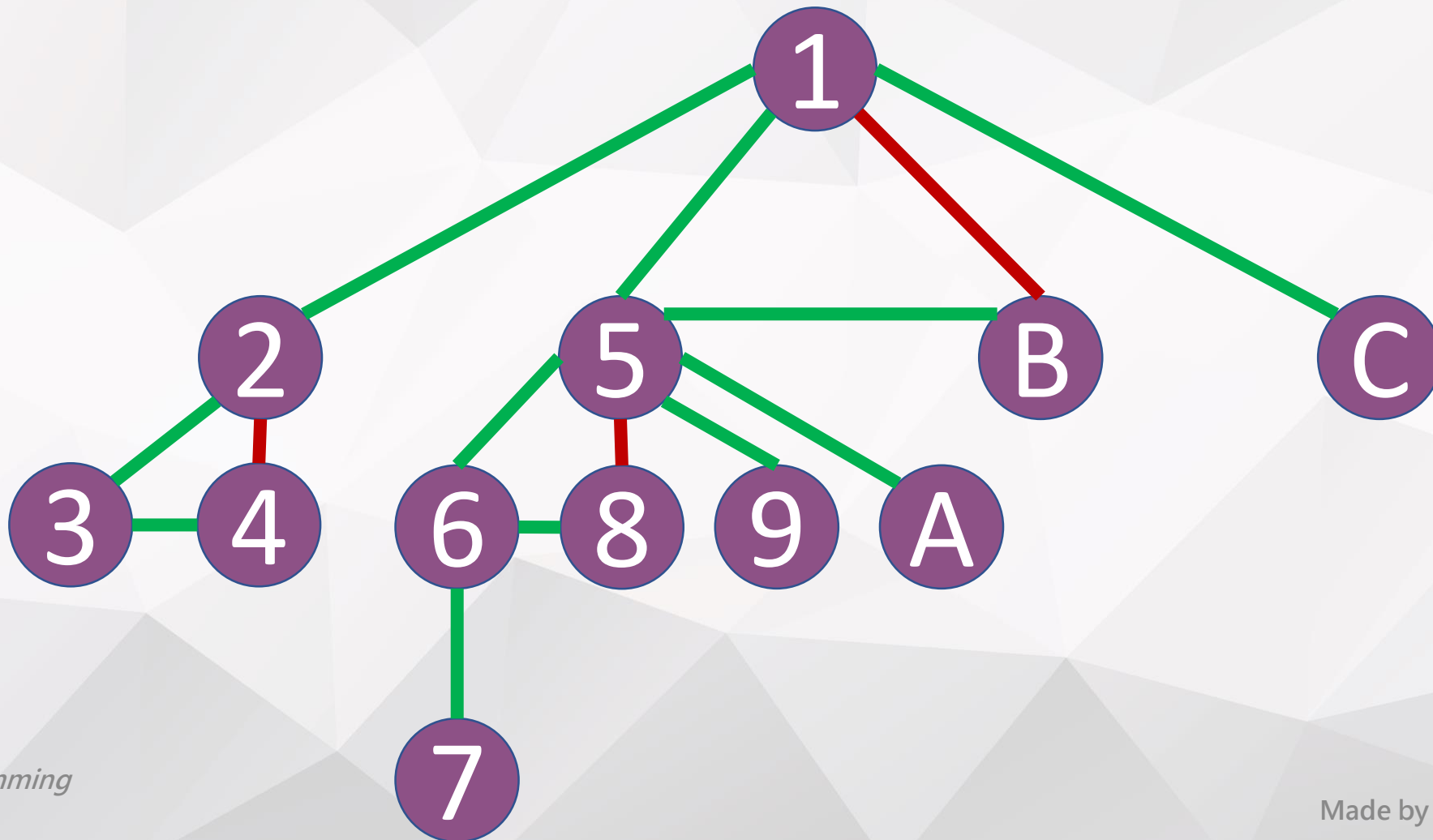
拜訪鄰點



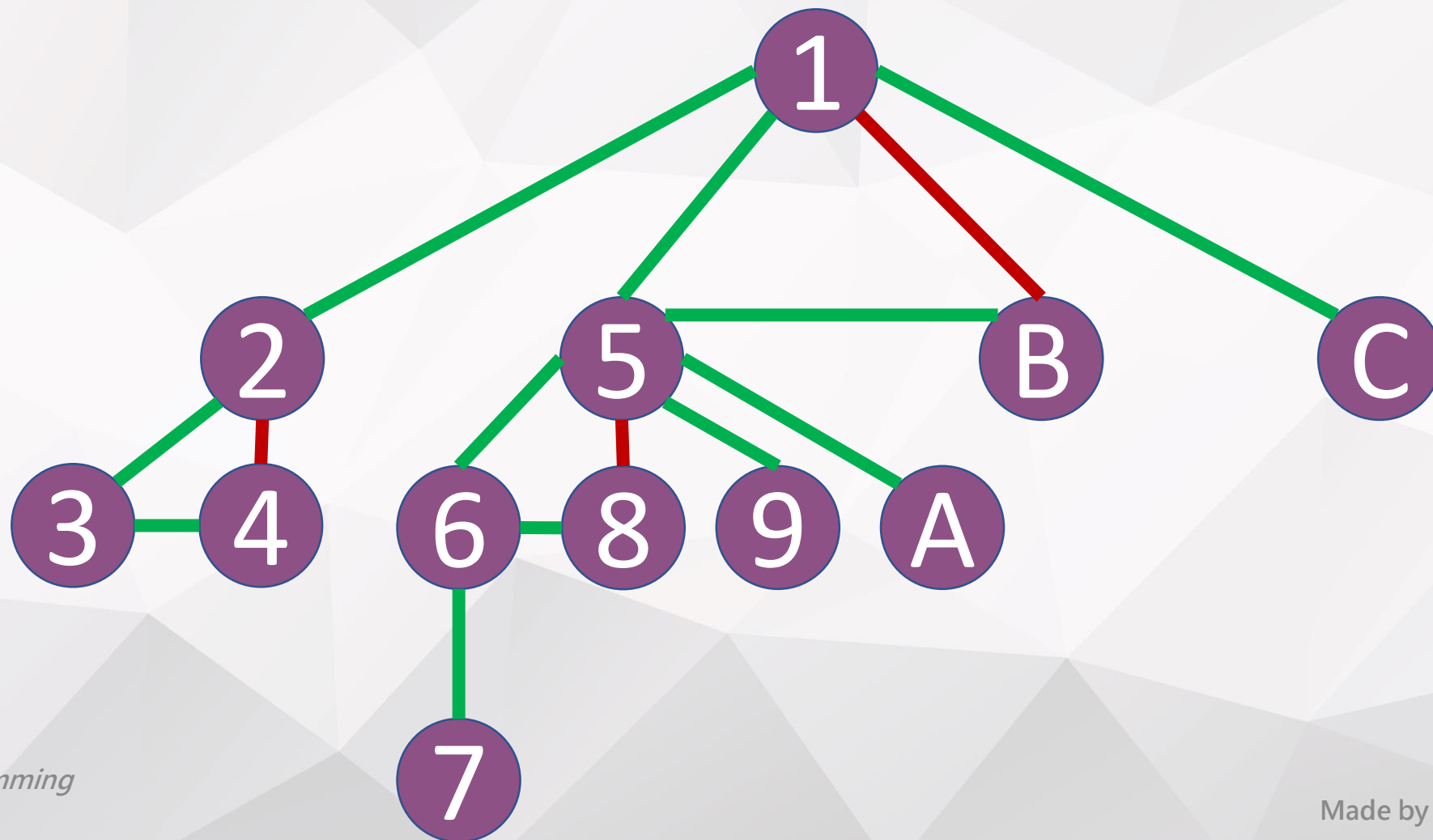
拜訪完



根拜訪完

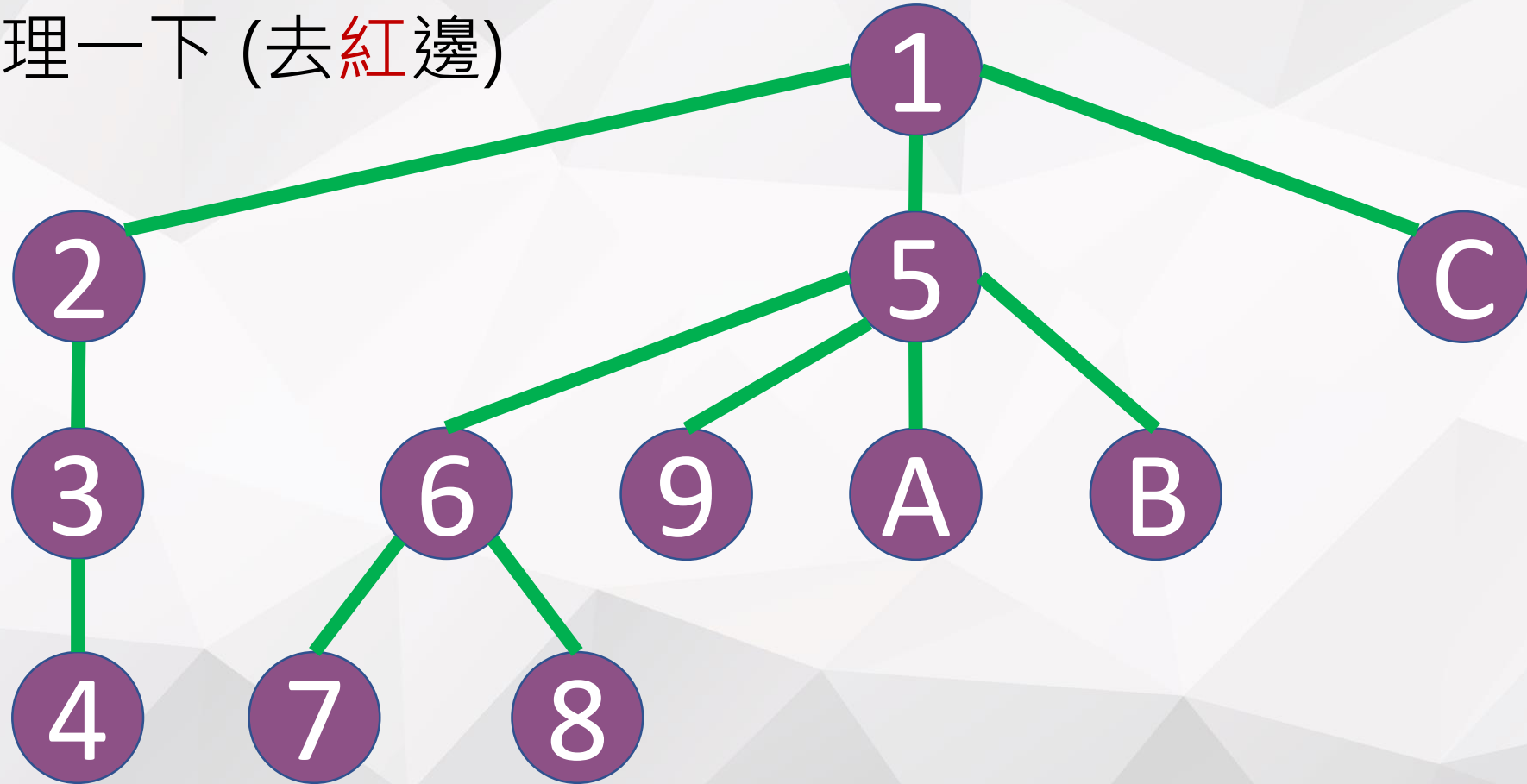


此樹稱為 DFS 樹



此樹稱為 DFS 樹

整理一下 (去紅邊)



Uva 572 Oil Deposits

GeoSurvComp 石油公司負責探勘某塊地底下的石油含量，這塊地是矩形的，並且為了探勘的方便被切割為許多小塊。

他們使用儀器對每個小塊去探勘。含有石油的小塊稱為一個 pocket。假如兩個 pocket 相連，則這兩個 pocket 屬於同一個 oil deposit

你的任務就是要找出這塊地包含幾個不同的 oil deposit

Uva 572 Oil Deposits

輸入:

1 1

*

輸出:

0

Uva 572 Oil Deposits

輸入:

3 5

@@*

@

@@*

輸出:

1

Uva 572 Oil Deposits

輸入:

1 8

@@*****@*

輸出:

2

Uva 572 Oil Deposits

輸入:

5 5

*****@

@@@

*@**@

@@@*@

@@**@

輸出:

2

Uva 572 Oil Deposits

```
int count = 0;
for (int i = 0; i < m; i++)
    for (int j = 0; j < n; j++)
        if (plot[i][j] == '@') {
            dfs(i, j);
            count++;
        }
```

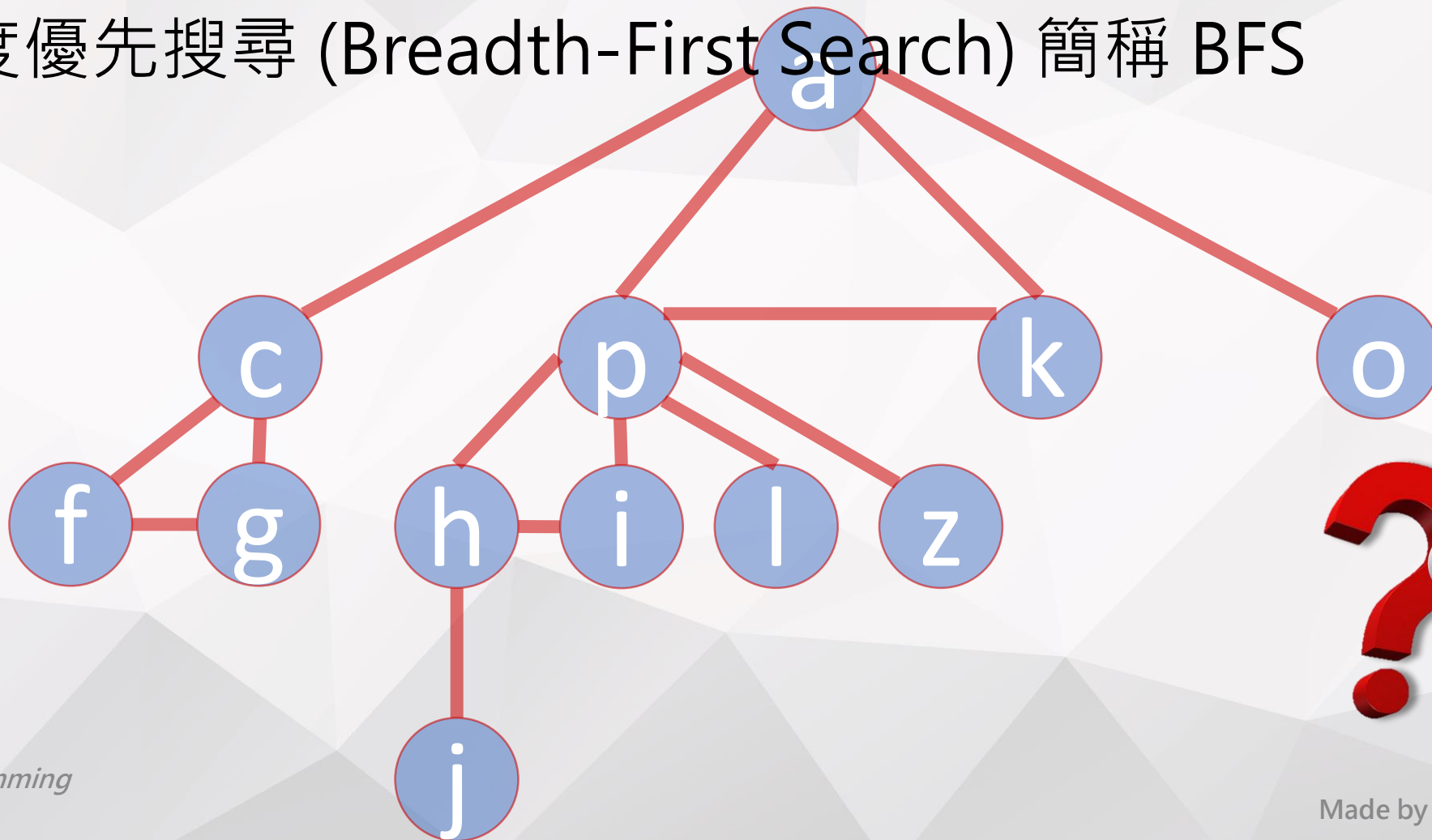
Uva 572 Oil Deposits

```
void dfs(int r, int c) {  
    if (plot[r][c] == '*') return;  
    plot[r][c] = '*';  
  
    for (int dr = -1; dr <= 1; dr++)  
        for (int dc = -1; dc <= 1; dc++)  
            if (r+dr >= 0 && r+dr < m  
                && c+dc >= 0 && c+dc < n) dfs(r+dr, c+dc);  
}
```


廣度優先搜尋

BFS

廣度優先搜尋 (Breadth-First Search) 簡稱 BFS



BFS 的點遍歷順序

為每拜訪一個**未曾拜訪**節點 (拜訪中)

就往**所有鄰點**拜訪過去

當**拜訪完**此節點，回第一個拜訪中鄰點

*節點: BFS 遍歷完會產生一顆樹

*某節點拜訪完: 其鄰點都拜訪中

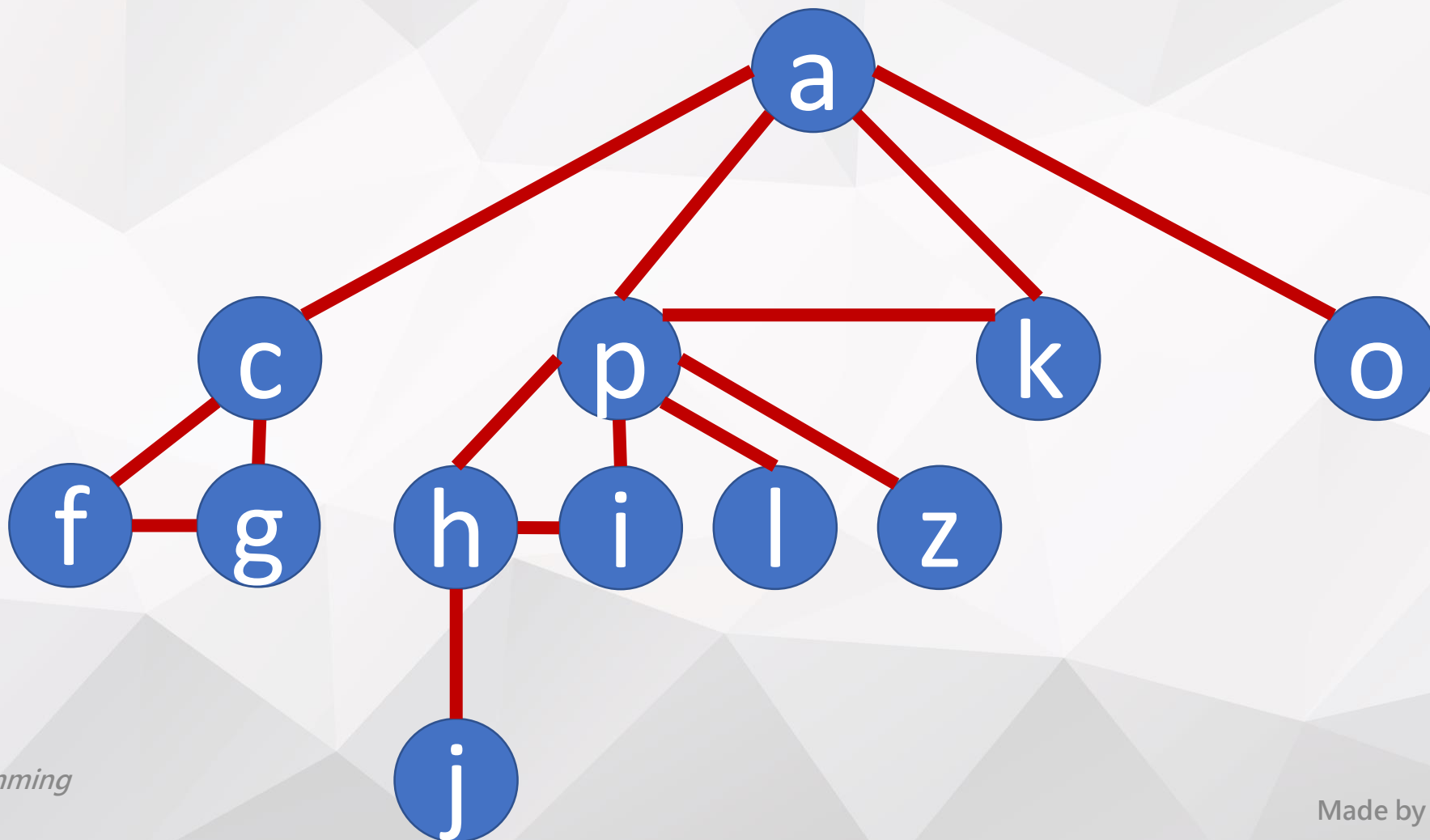
BFS 程式碼

```
queue<int> Q;  
Q.push(root); //root 代表走訪此圖的起點  
vis[root] = true;
```

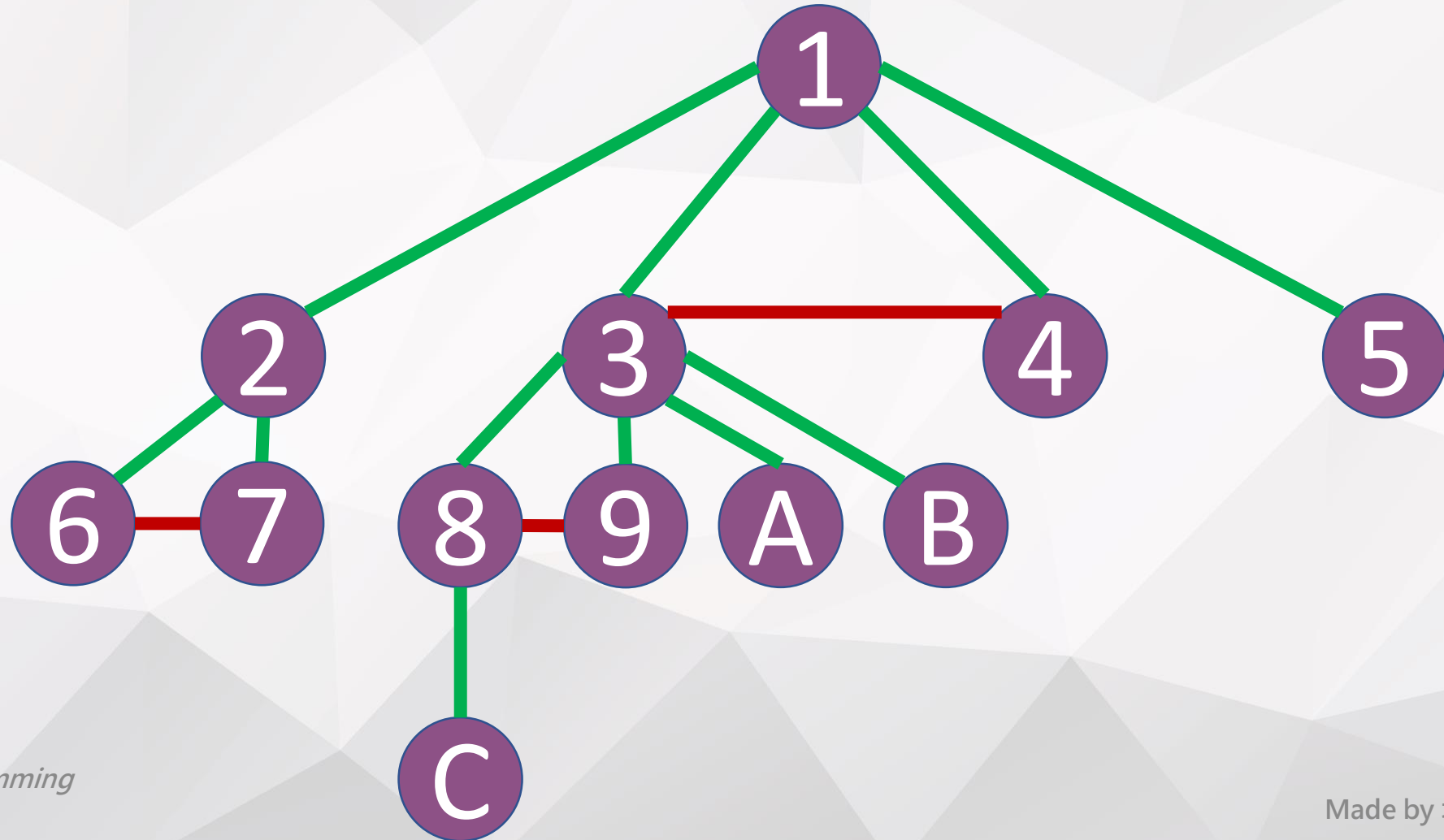
```
while (!Q.empty()) {  
    int u = Q.front(); Q.pop();  
    for (auto v: E[u]) {  
        if (vis[v]) continue;  
        vis[v] = true;  
        Q.push(v);  
    }  
}
```



BFS 的點遍歷順序

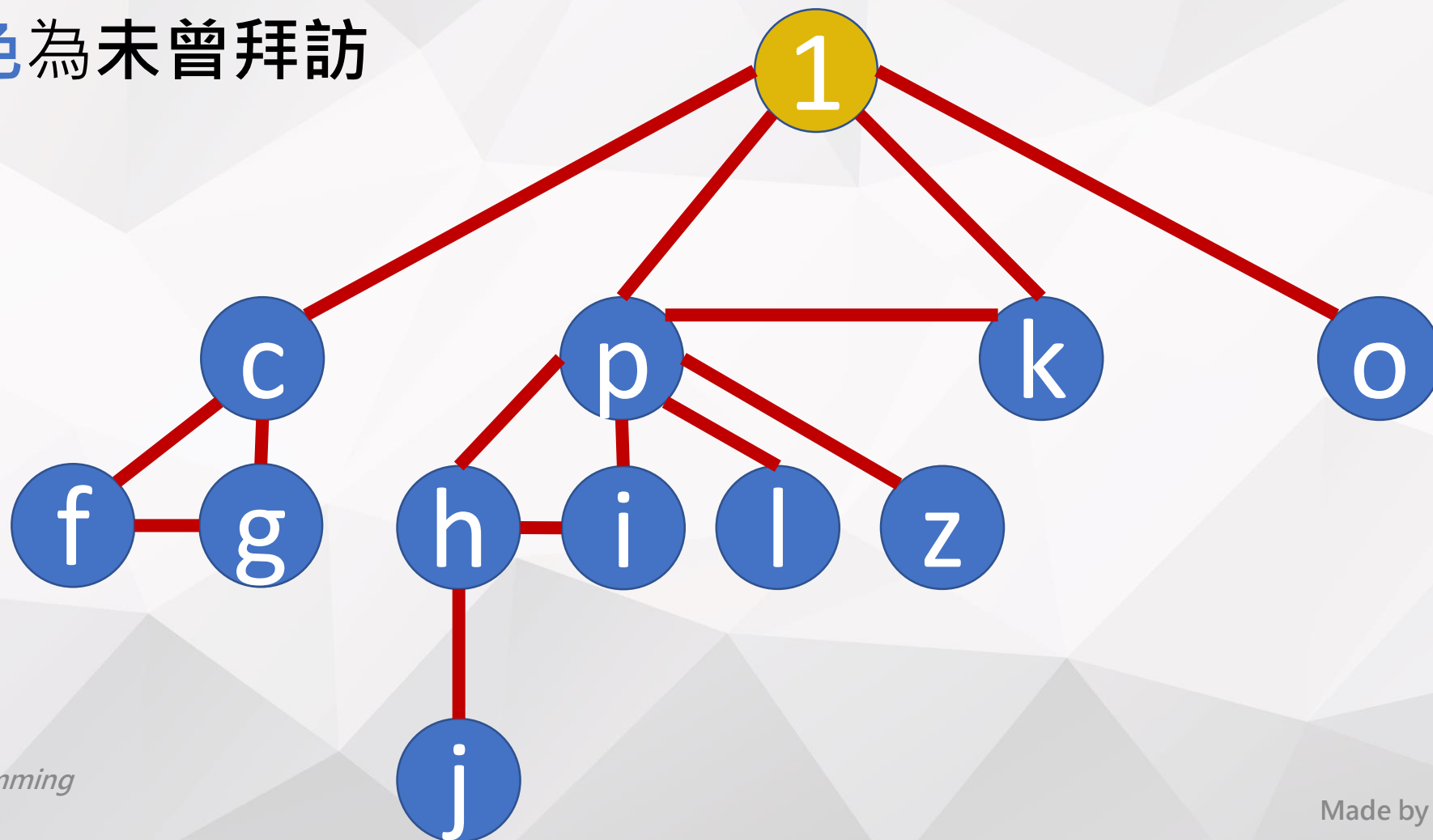


BFS 的點遍歷順序



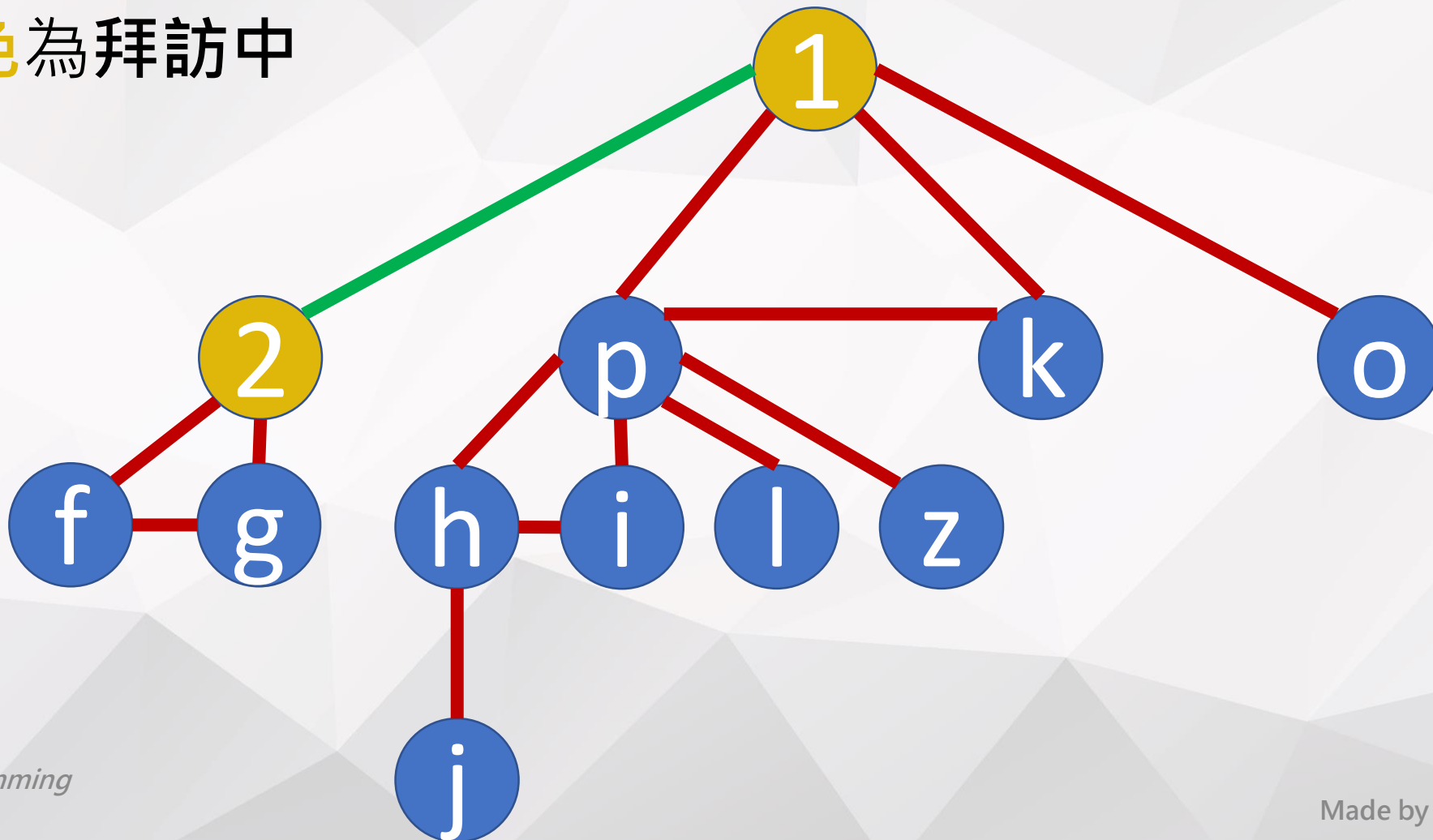
第一個拜訪的為根

藍色為未曾拜訪

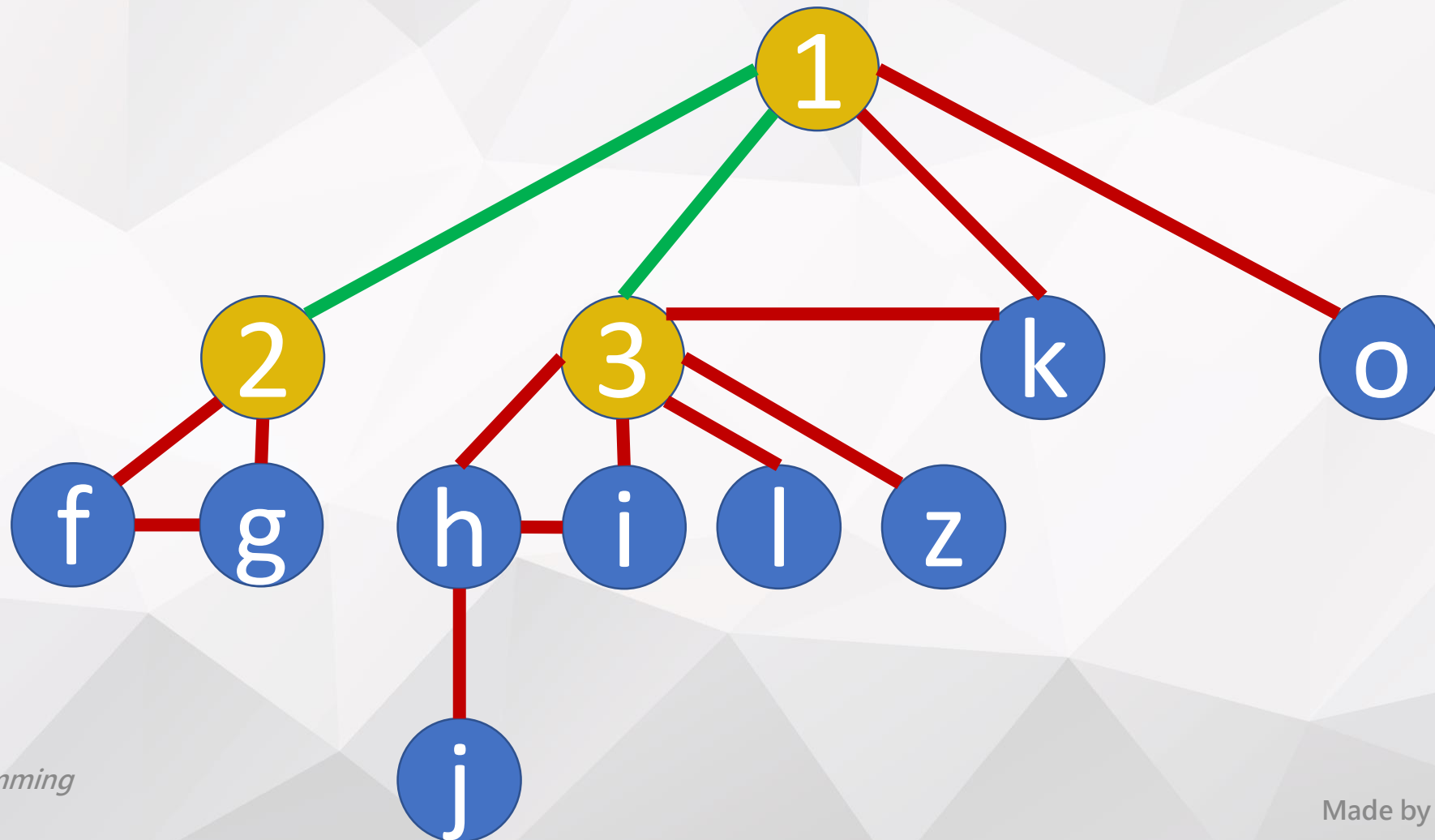


拜訪所有鄰點

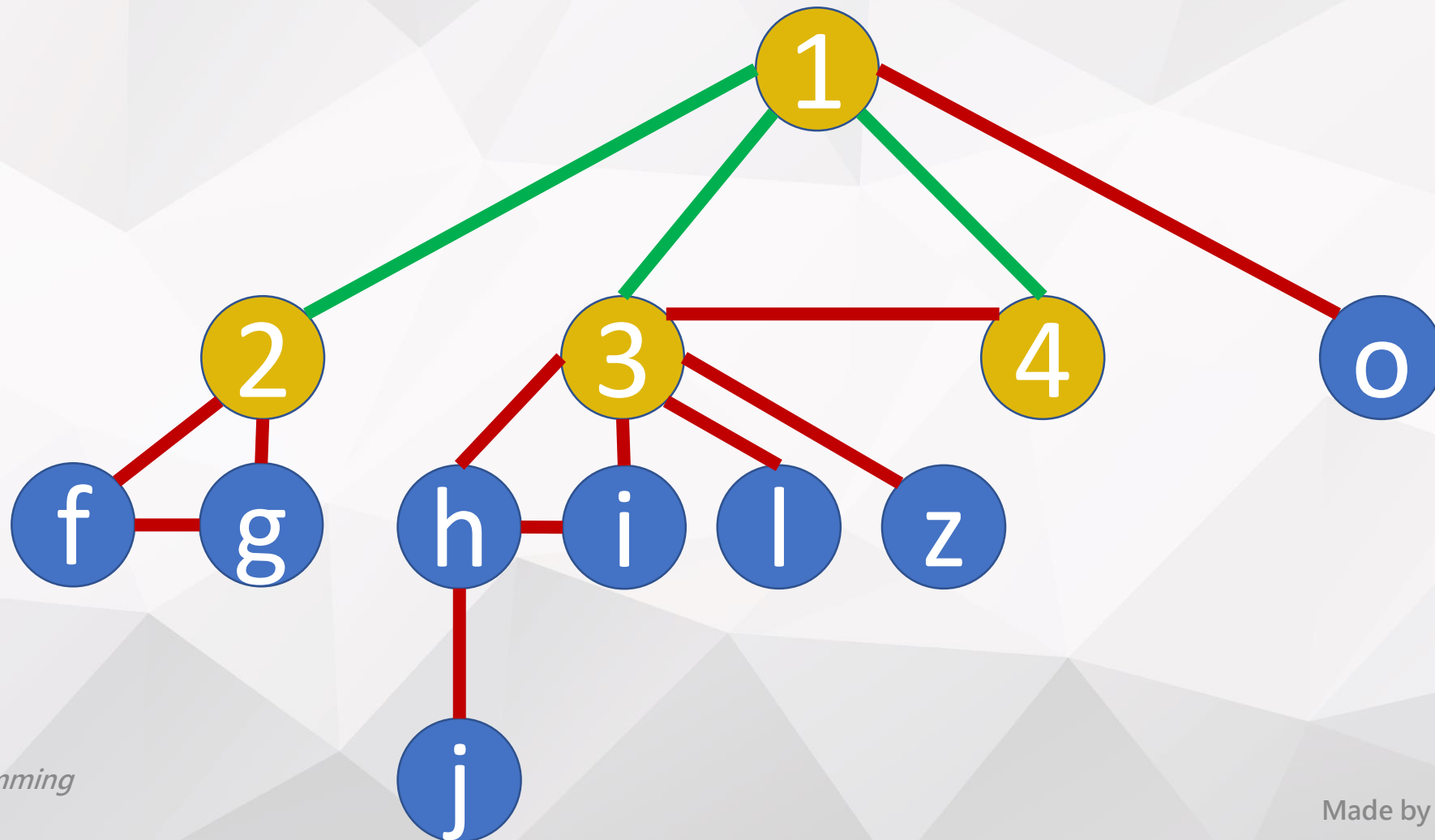
黃色為拜訪中



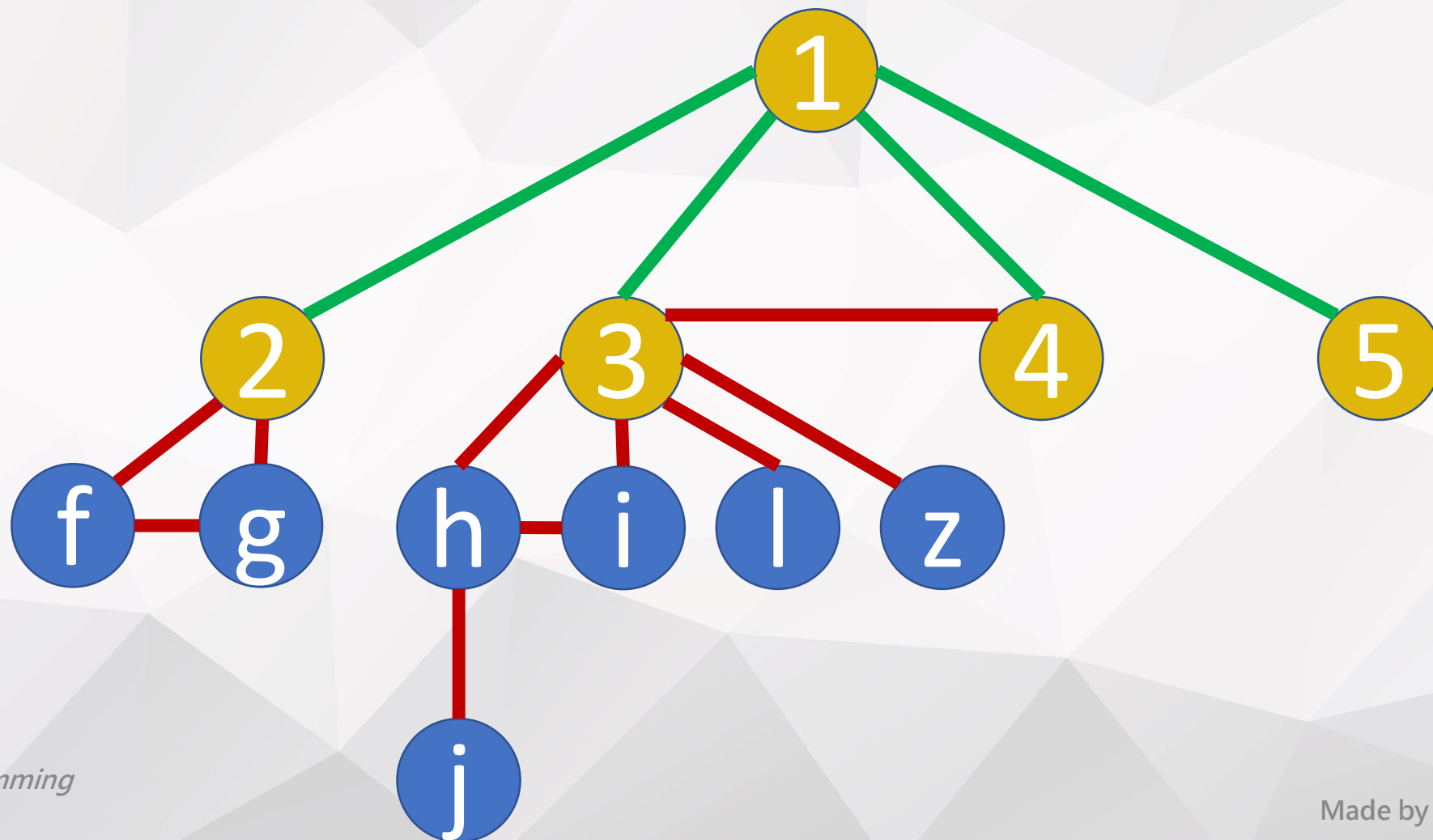
拜訪所有鄰點



拜訪所有鄰點

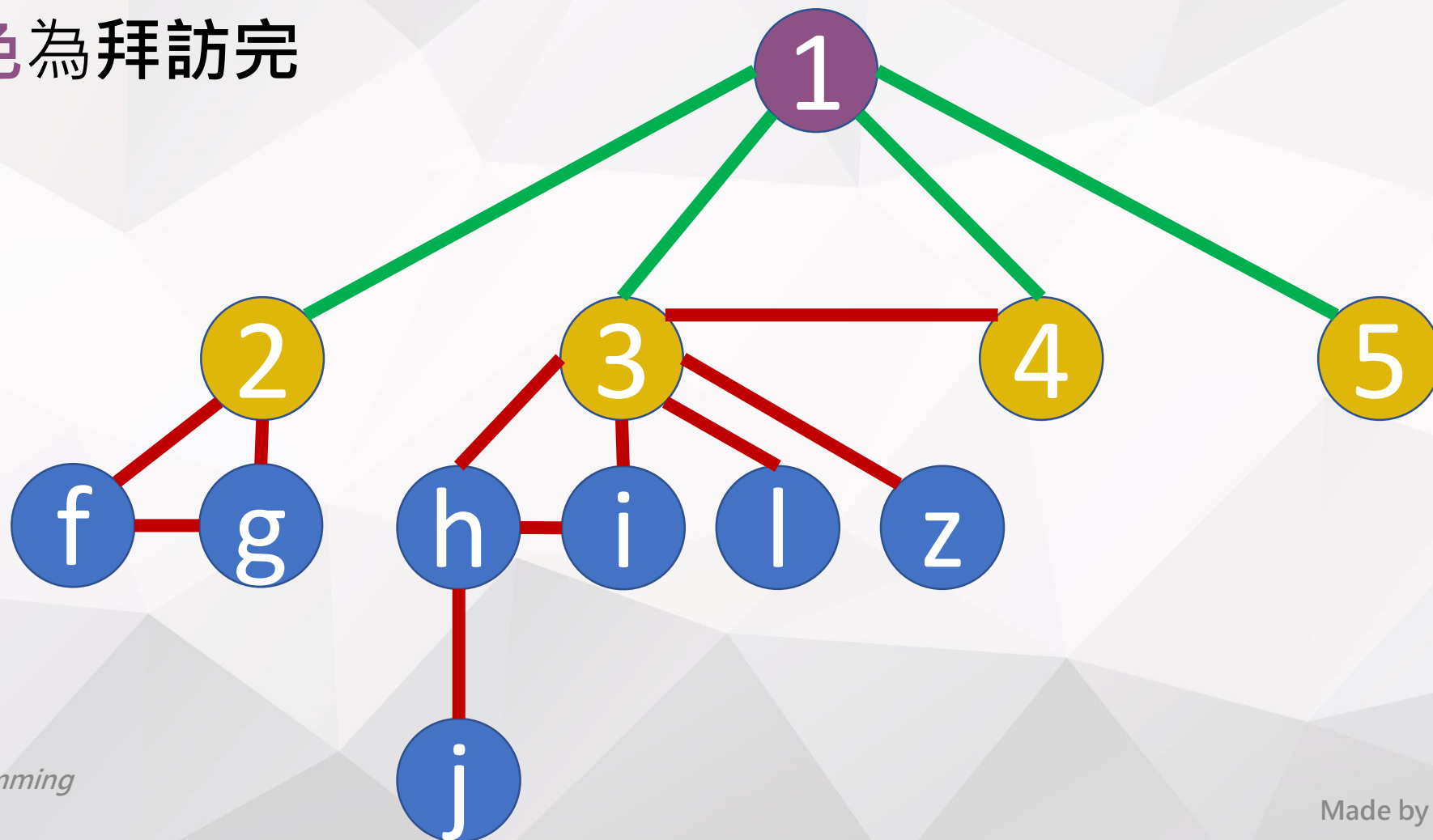


拜訪所有鄰點

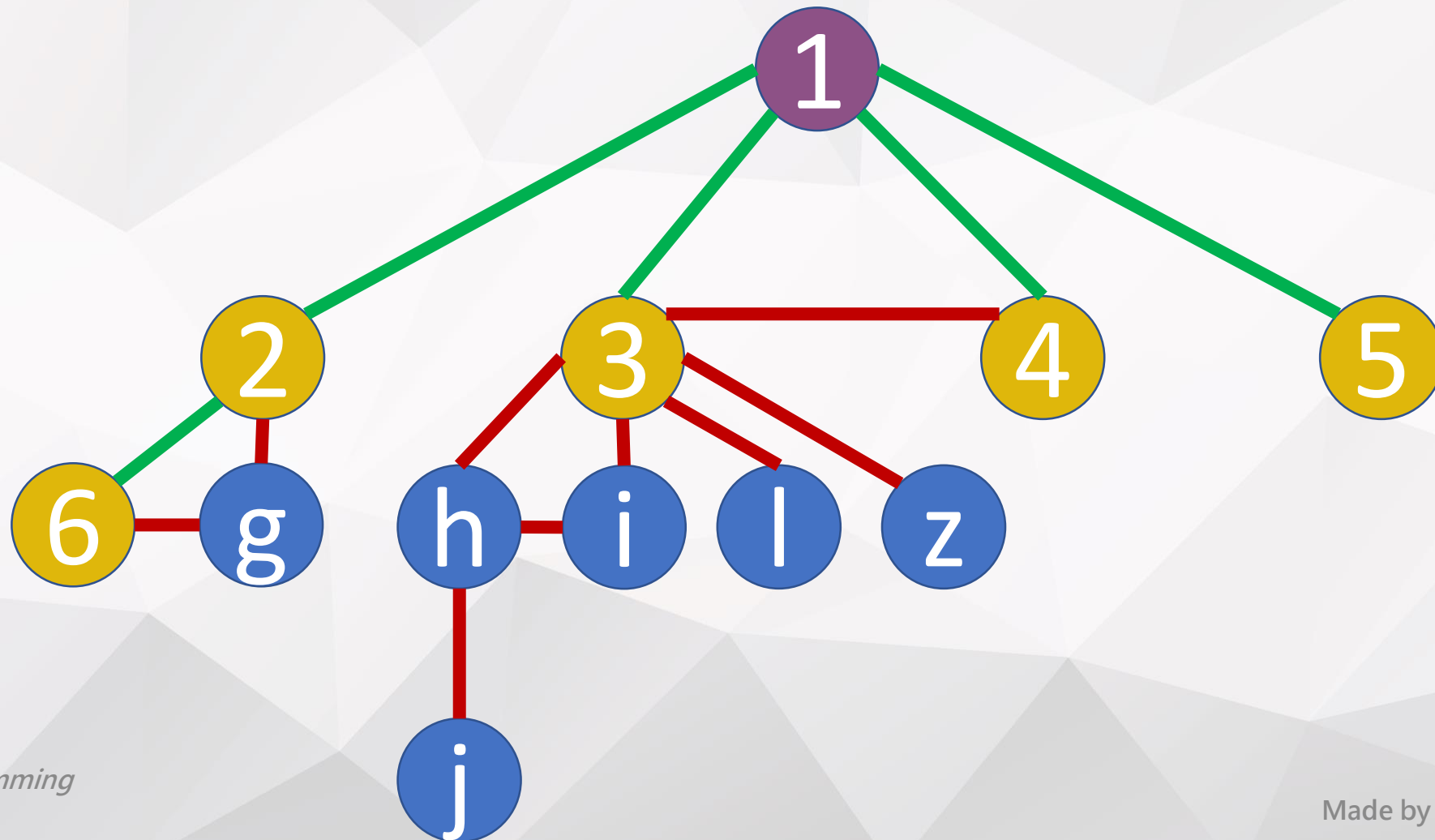


根拜訪完

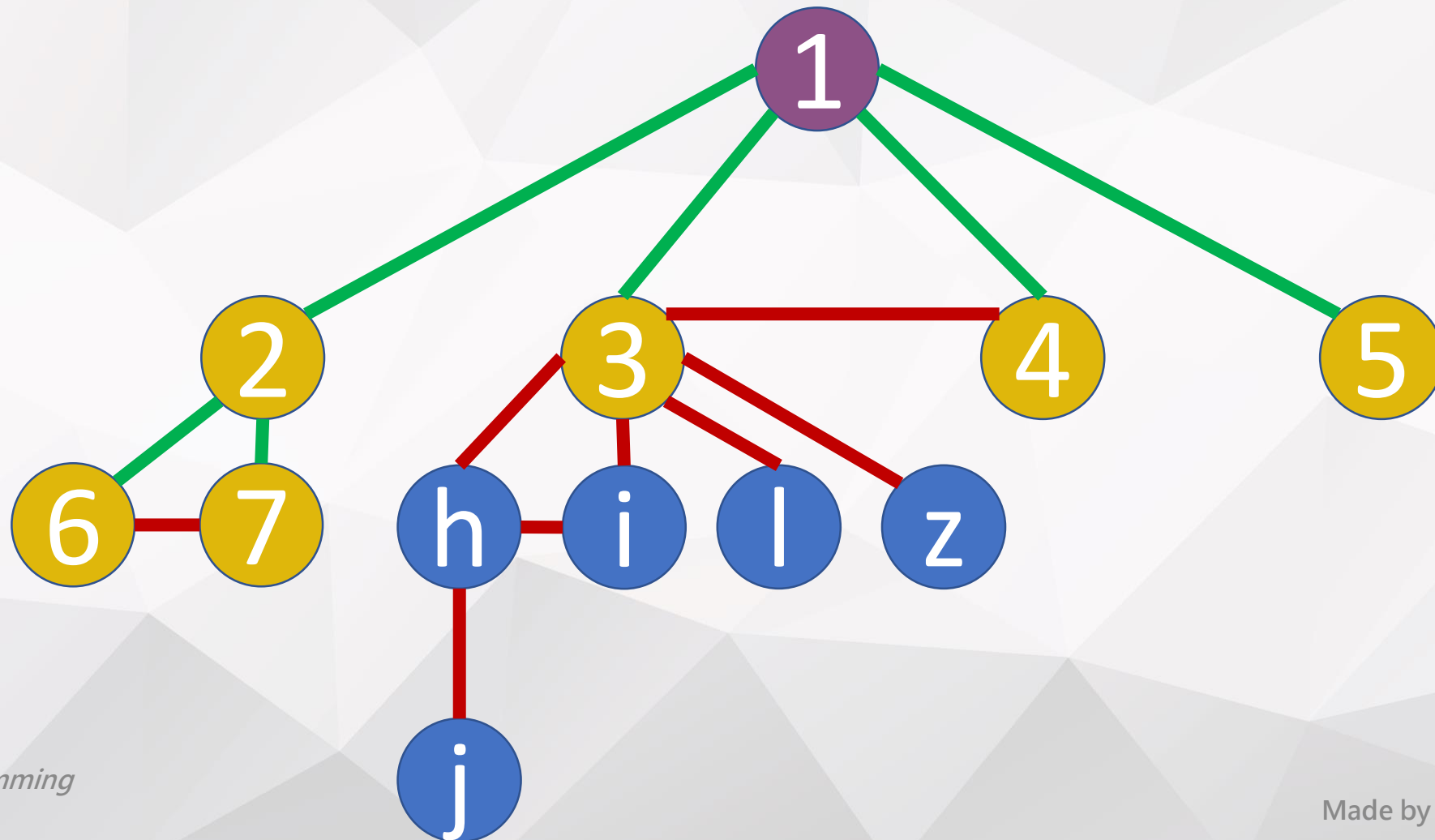
紫色為拜訪完



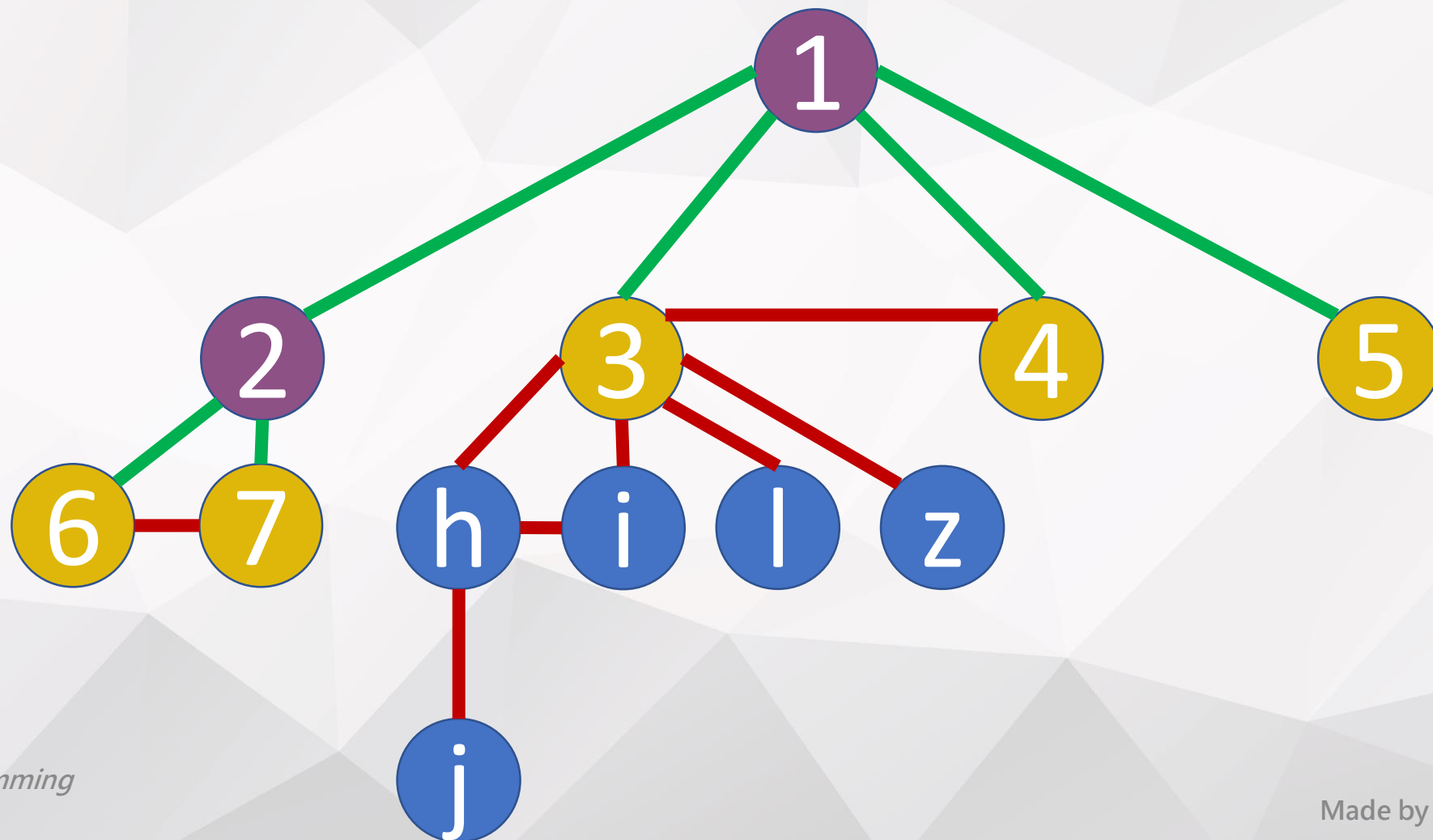
拜訪所有鄰點



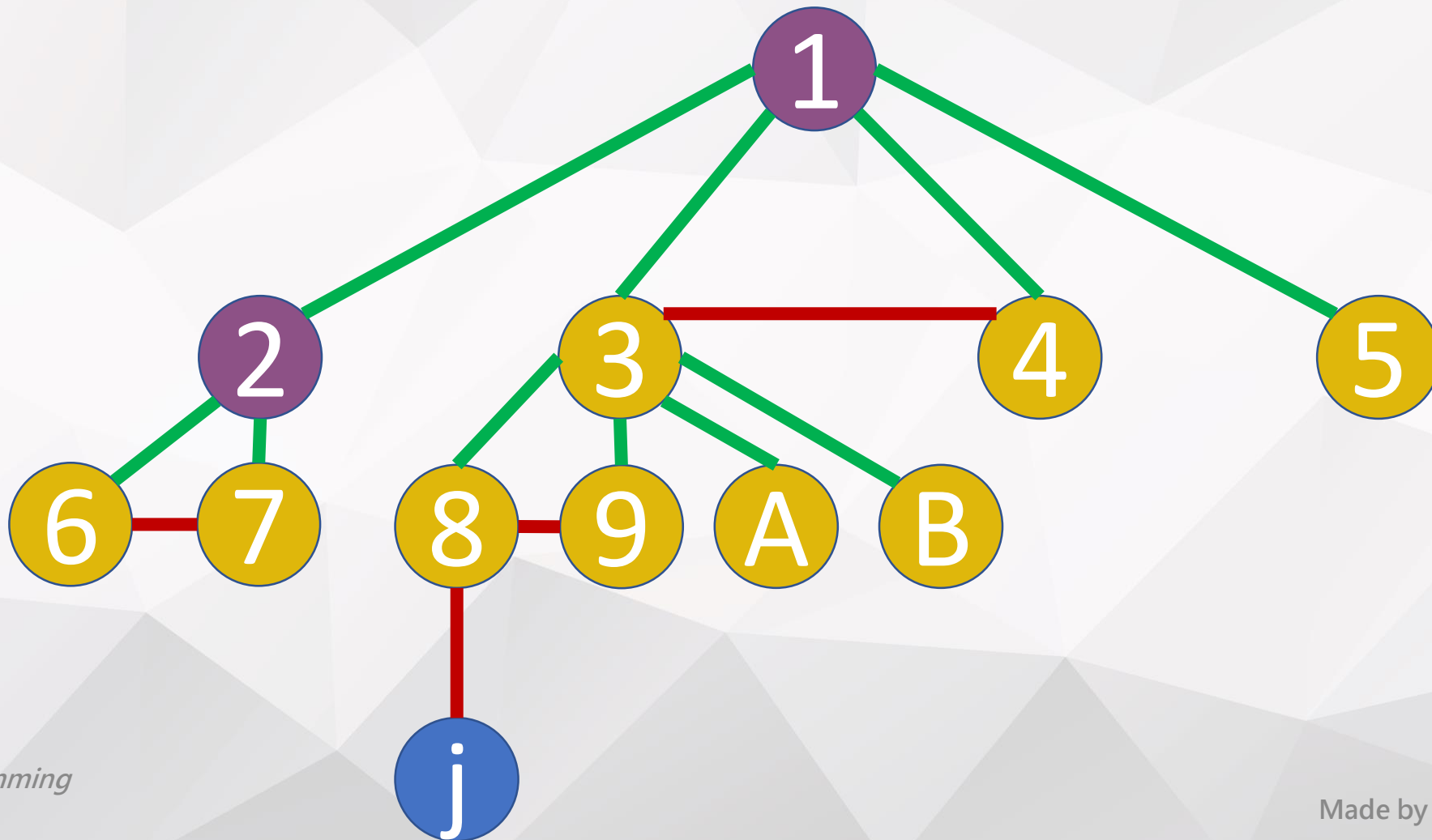
拜訪所有鄰點



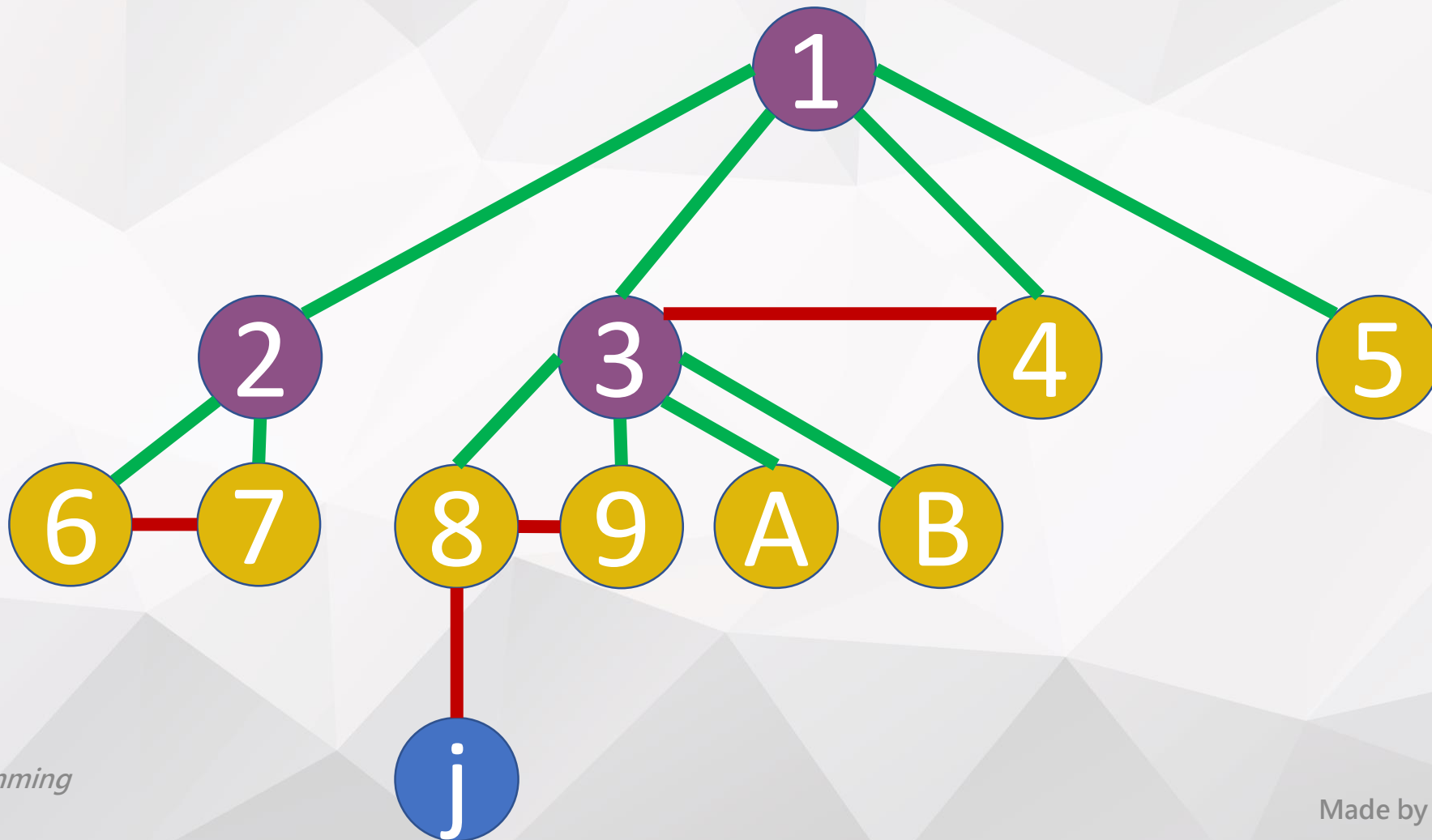
拜訪完



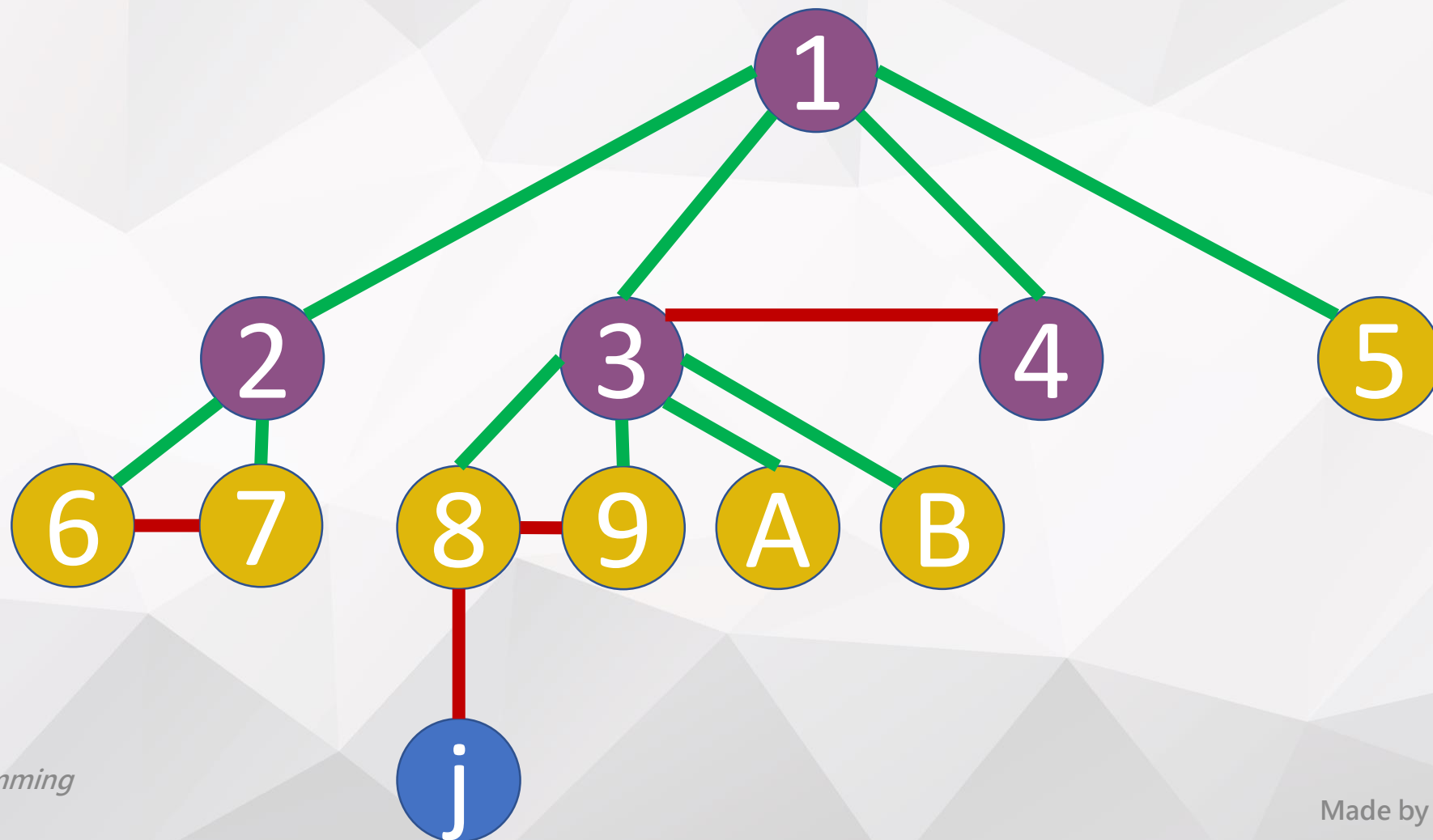
拜訪所有鄰點



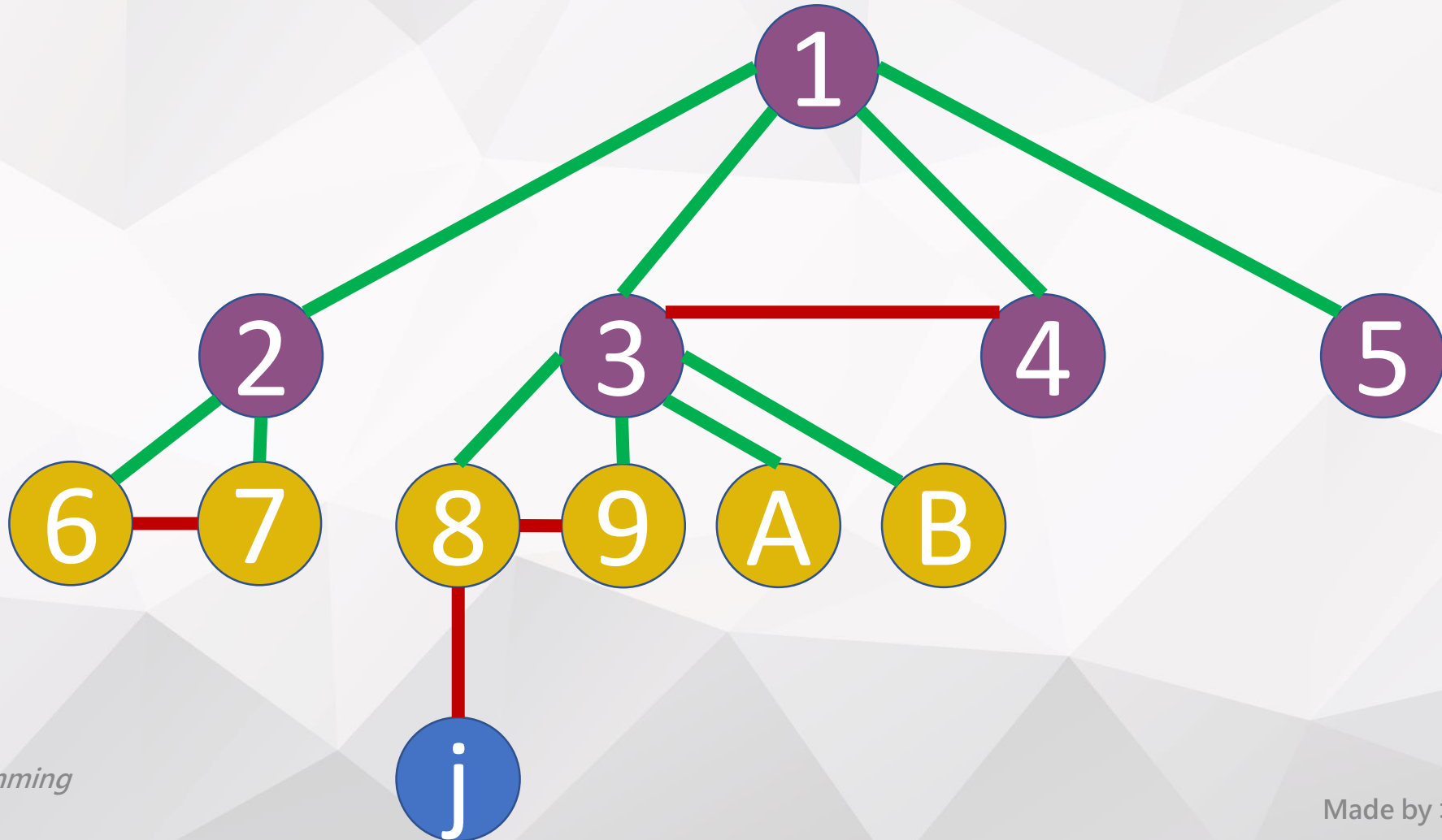
拜訪完



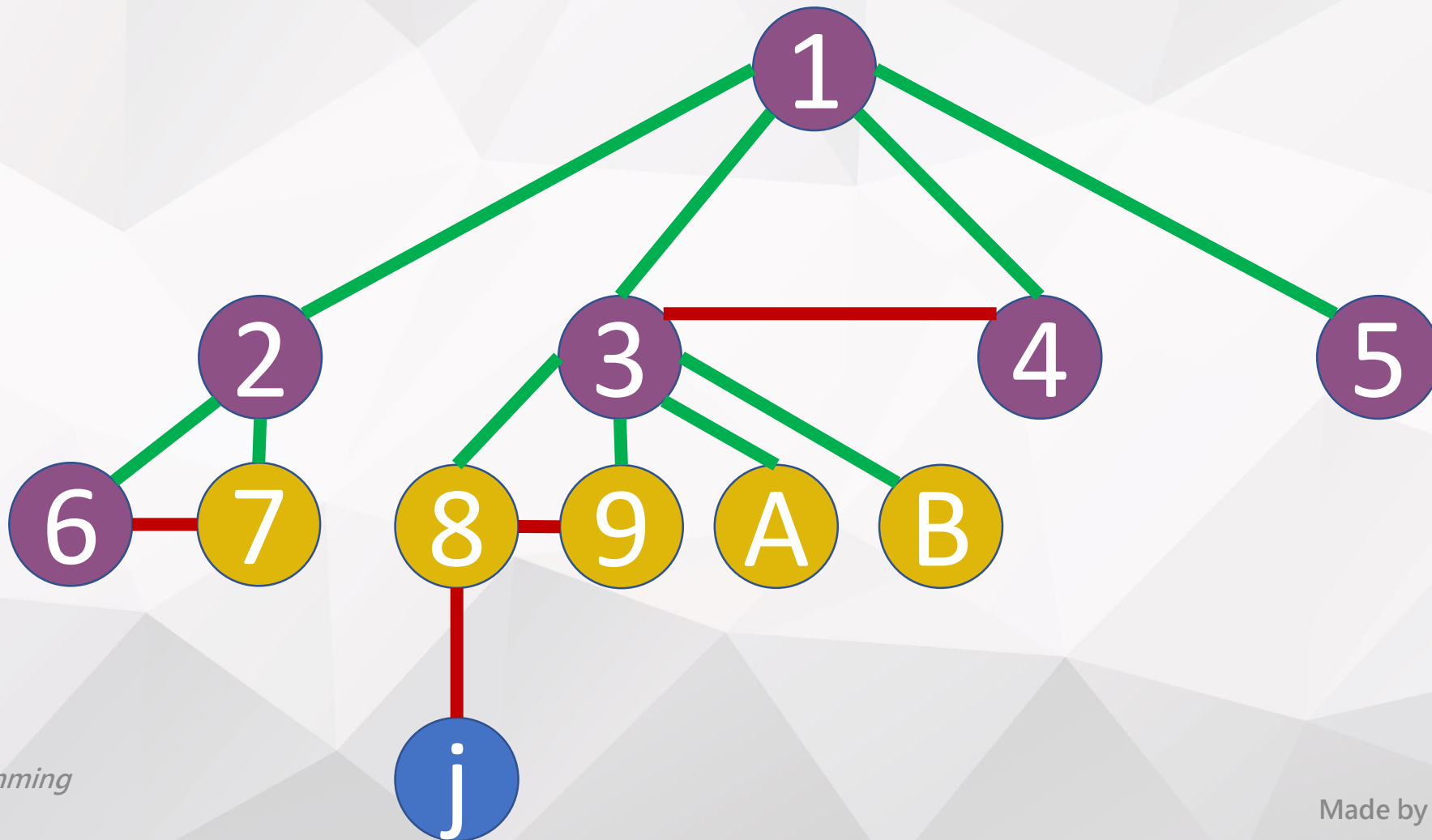
拜訪完



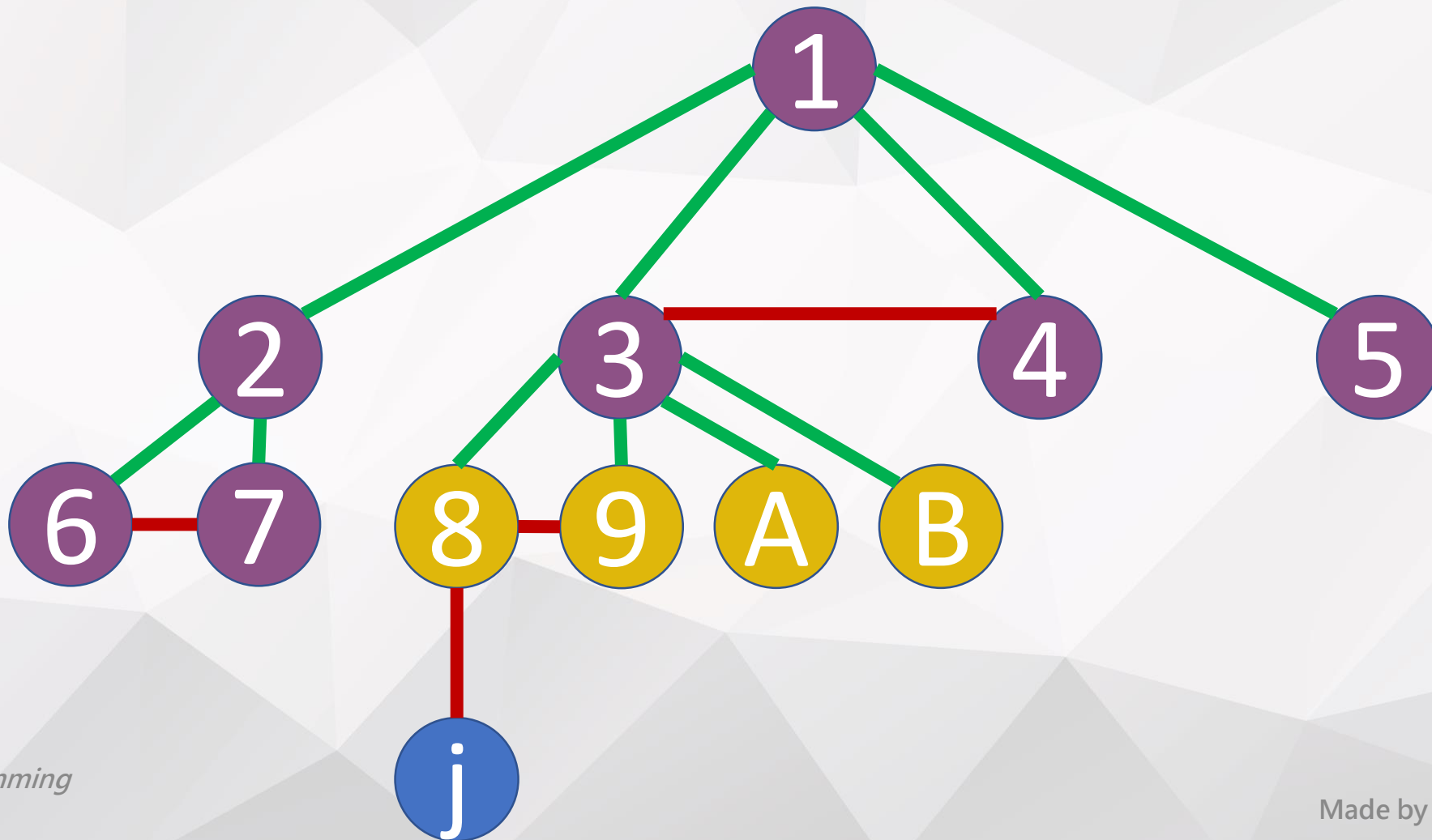
拜訪完



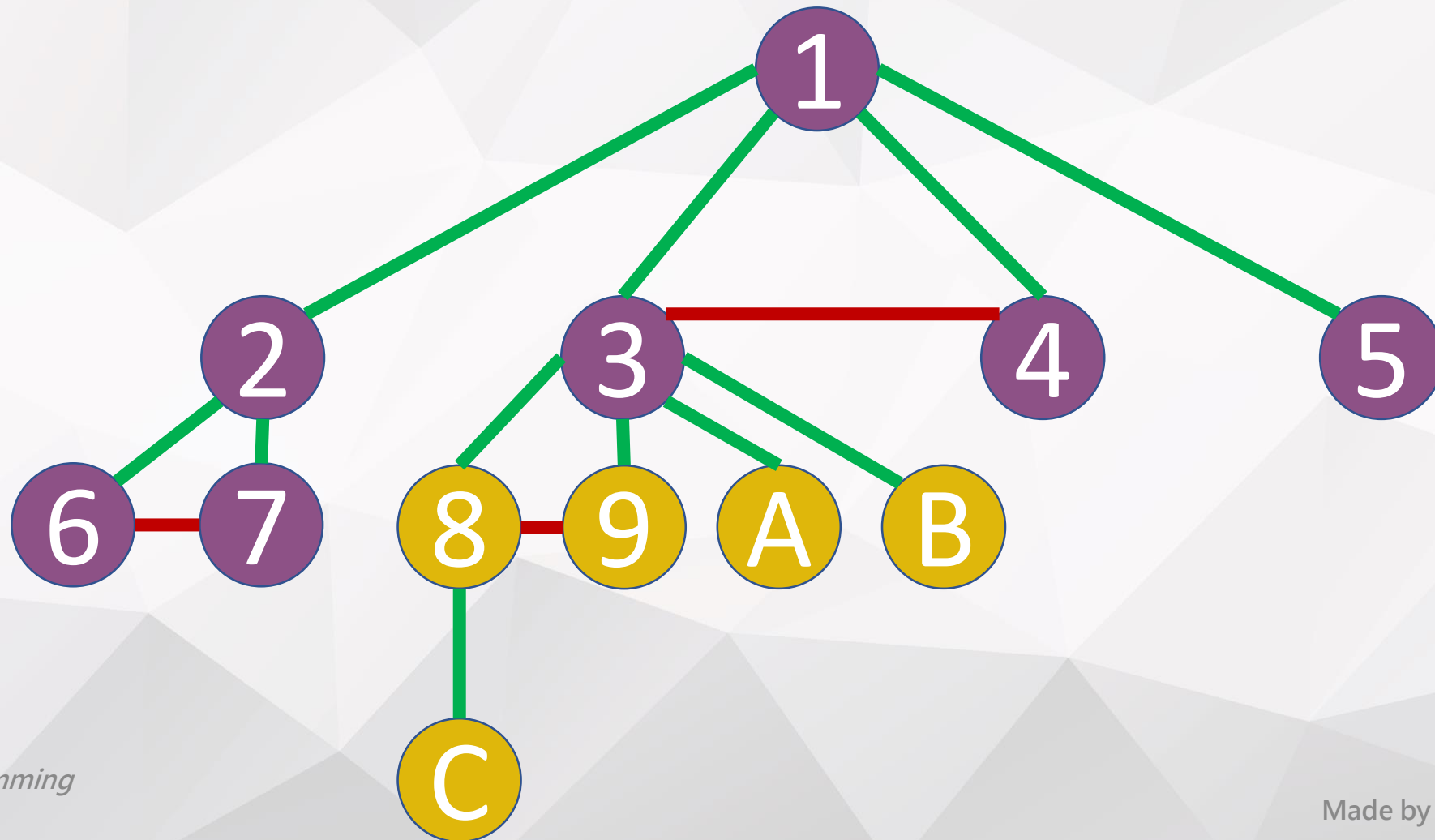
拜訪完



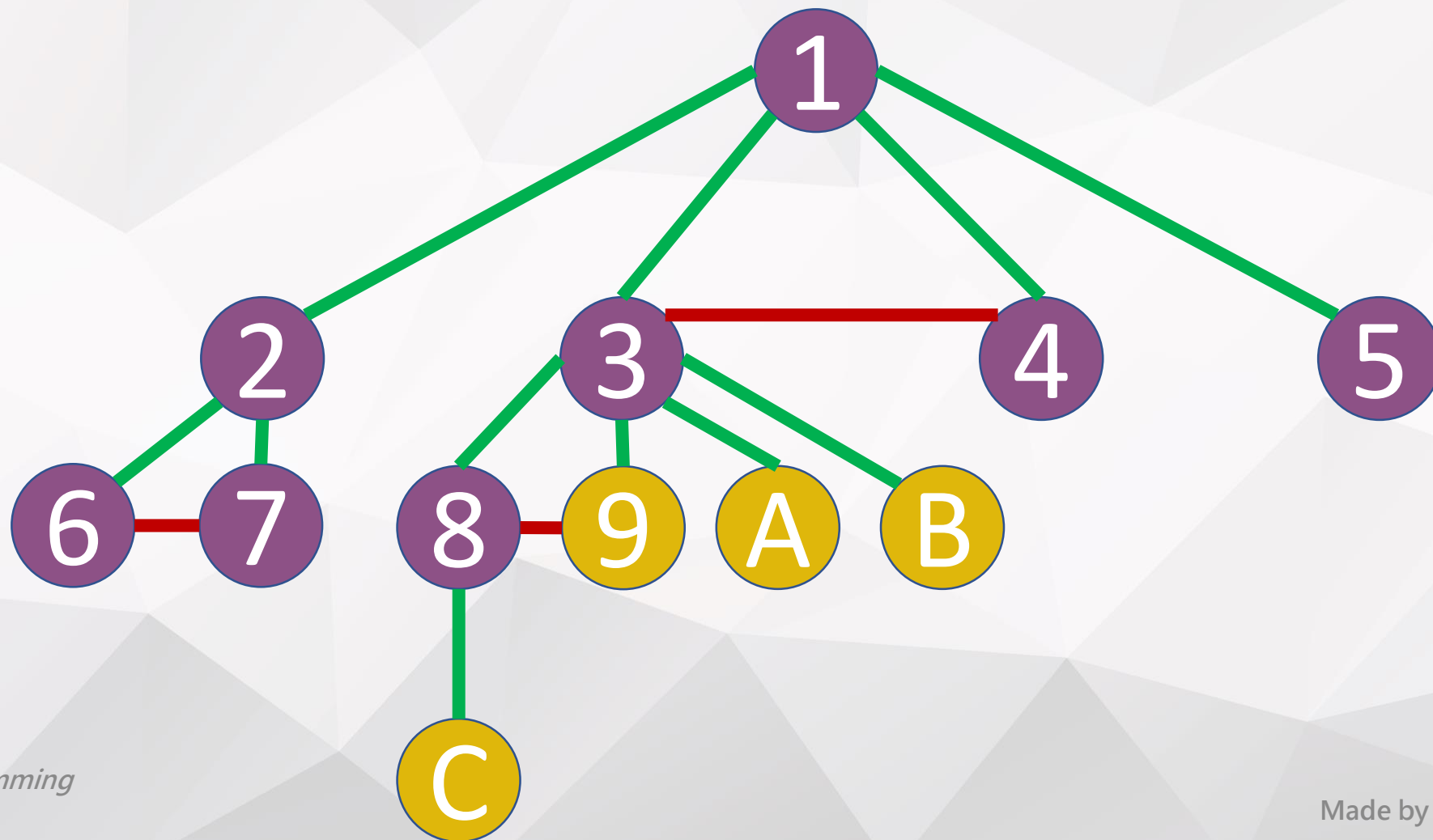
拜訪完



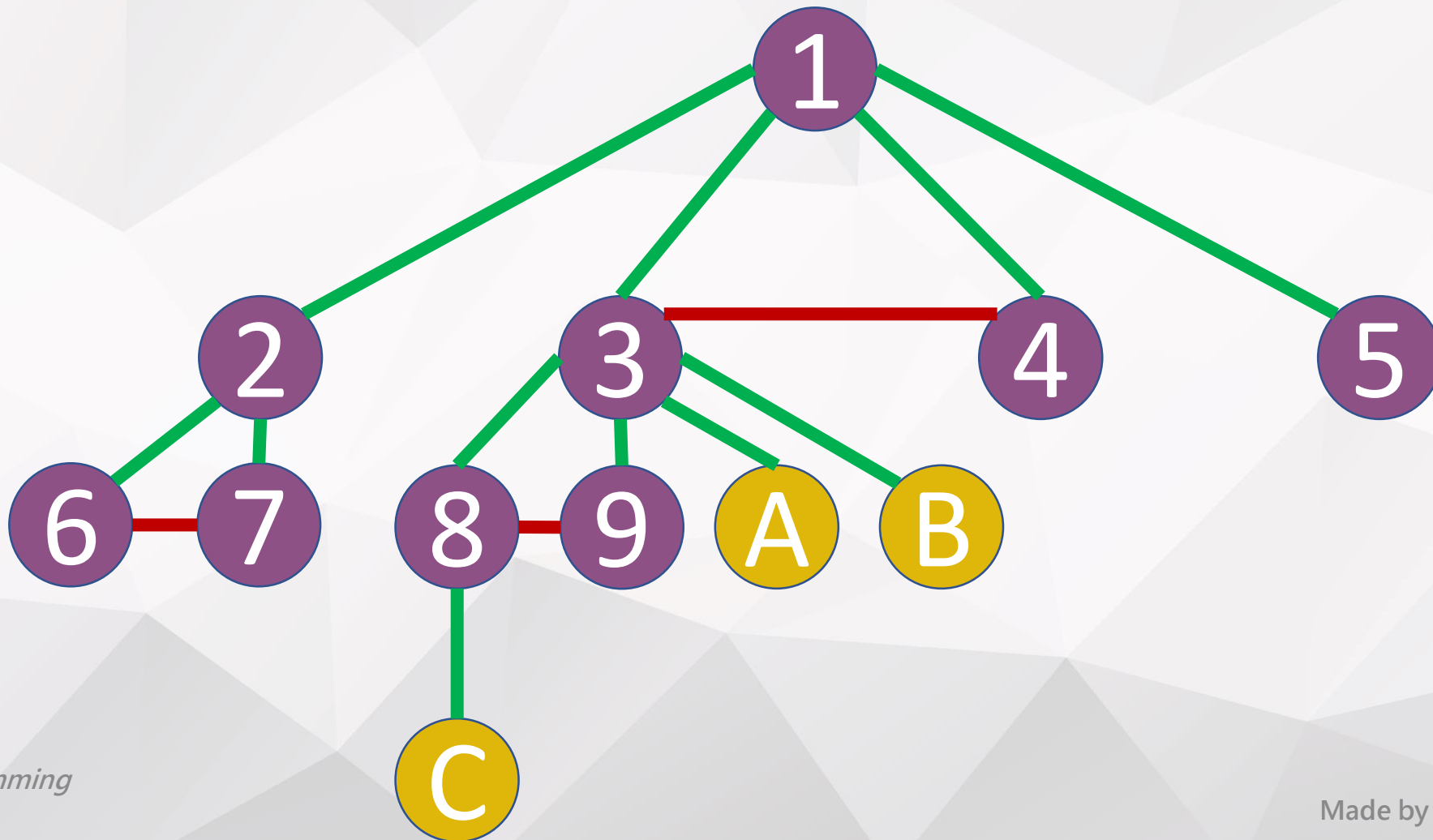
拜訪所有鄰點



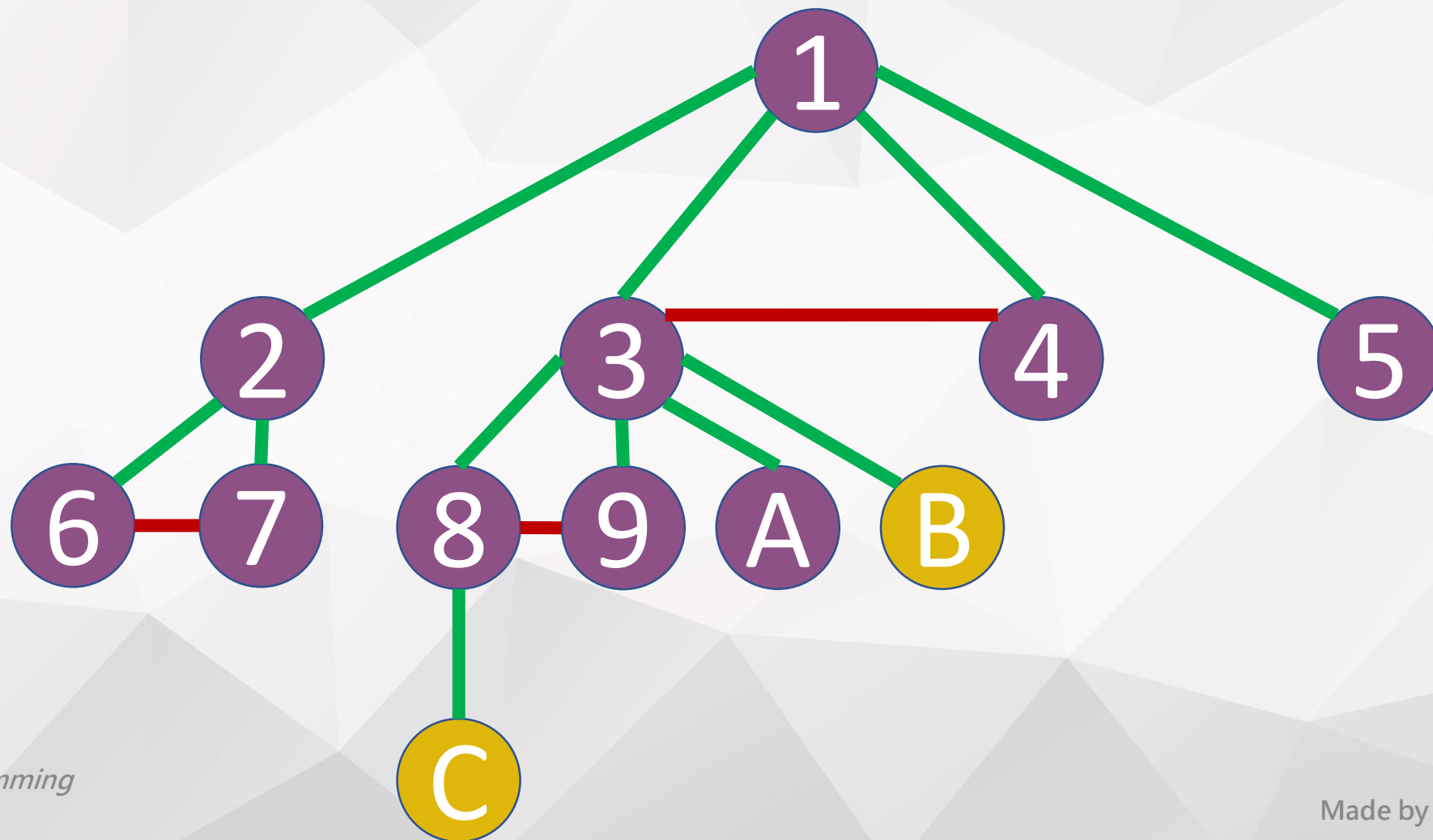
拜訪完



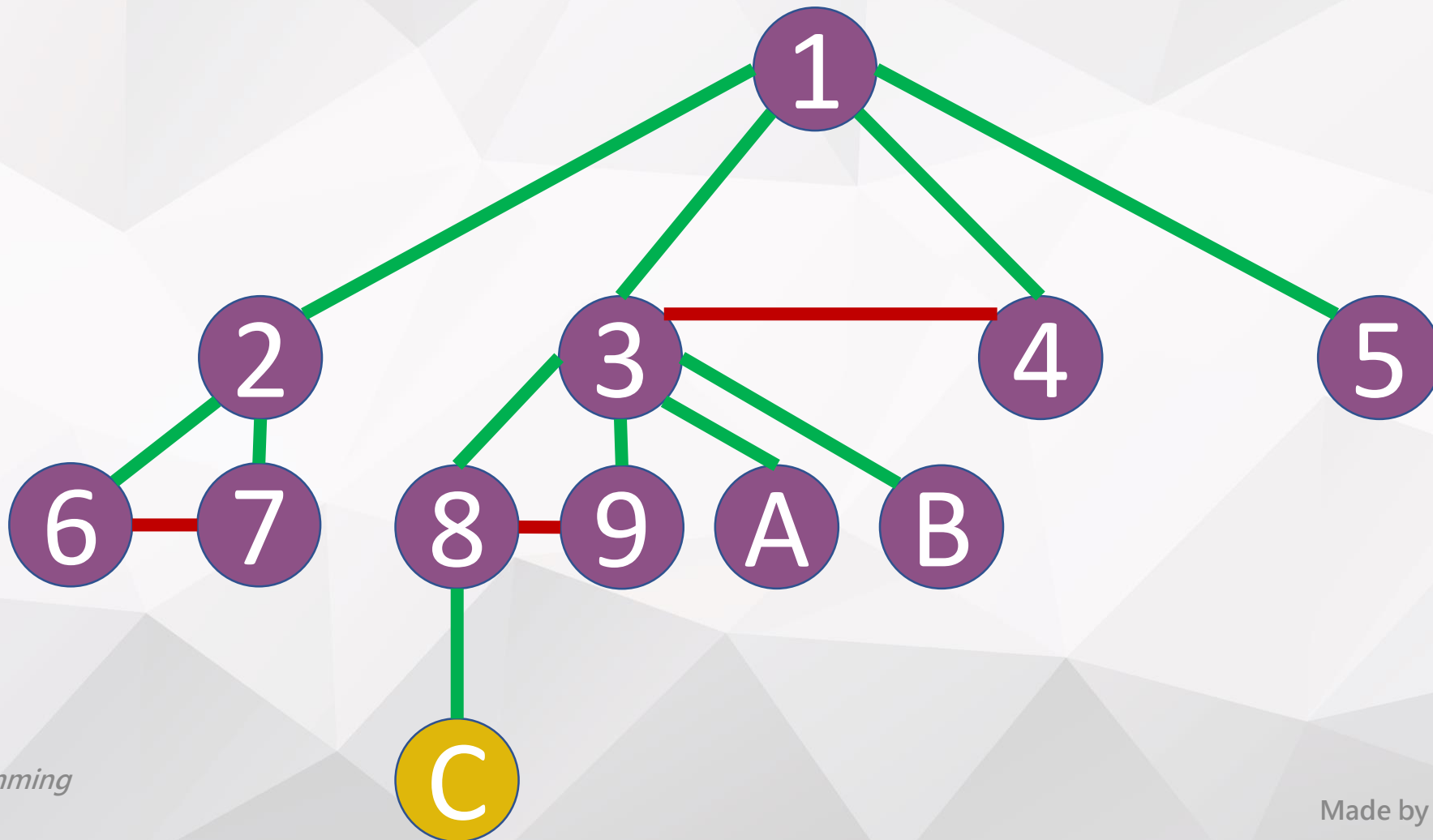
拜訪完



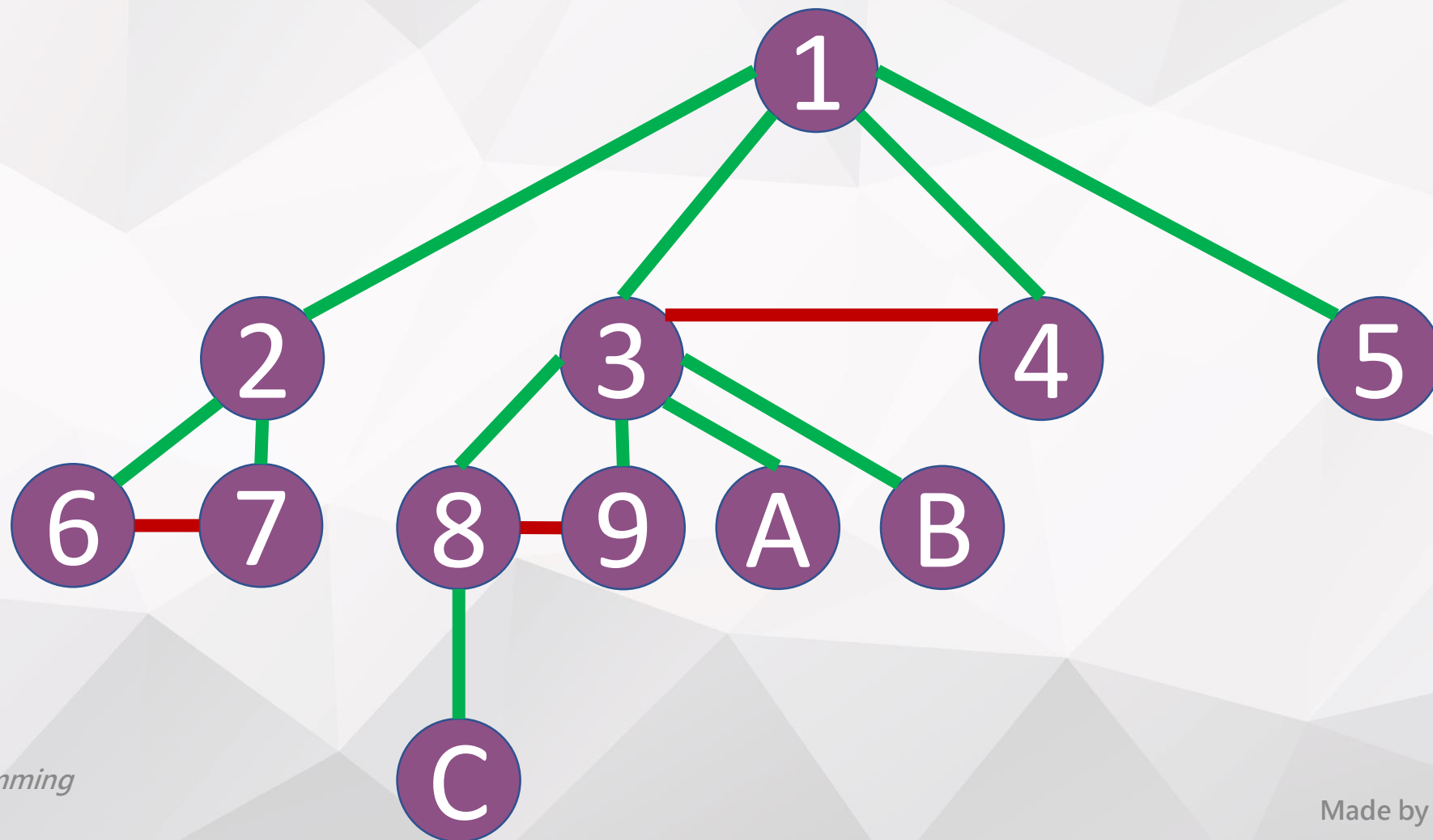
拜訪完



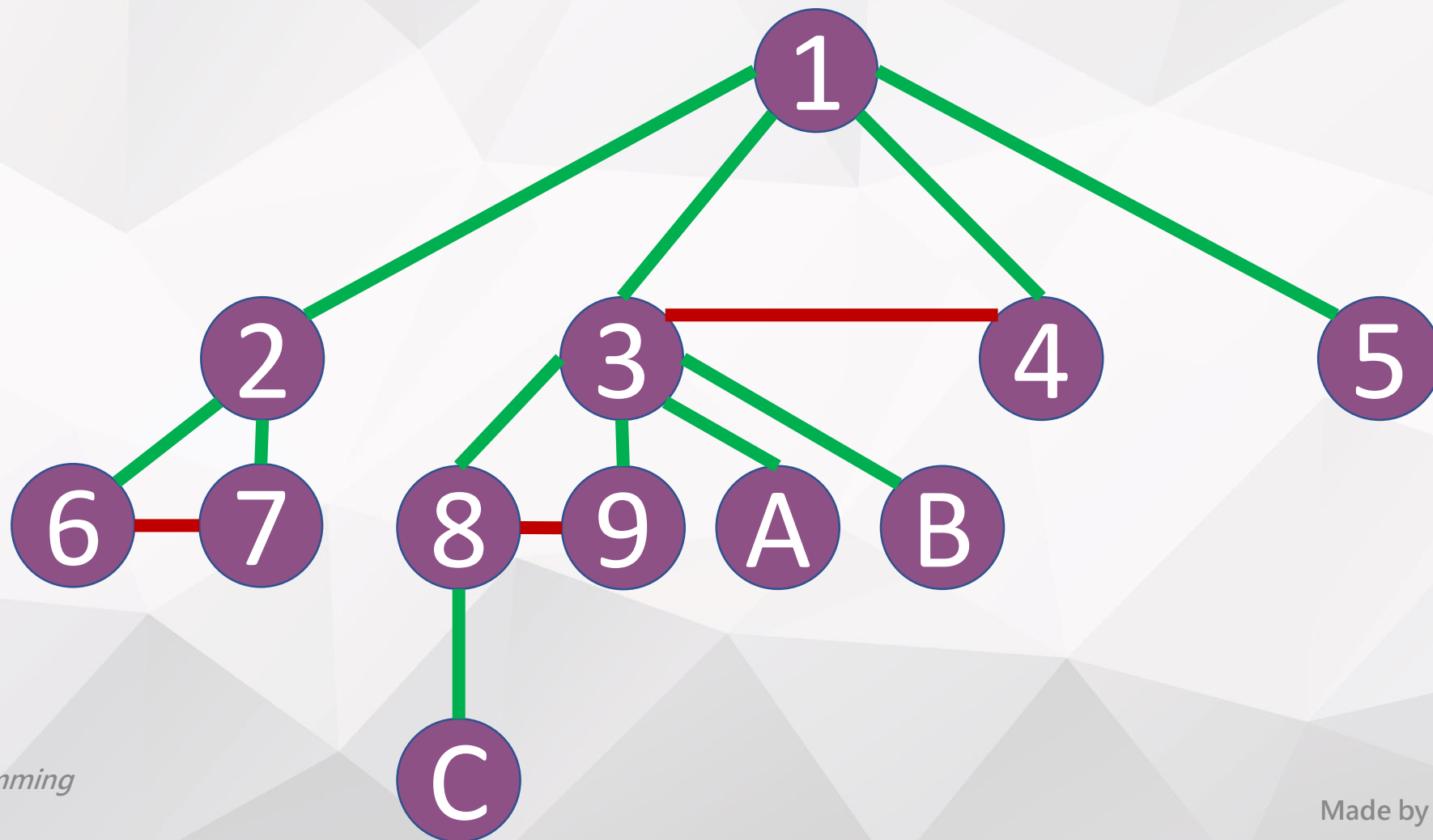
拜訪完



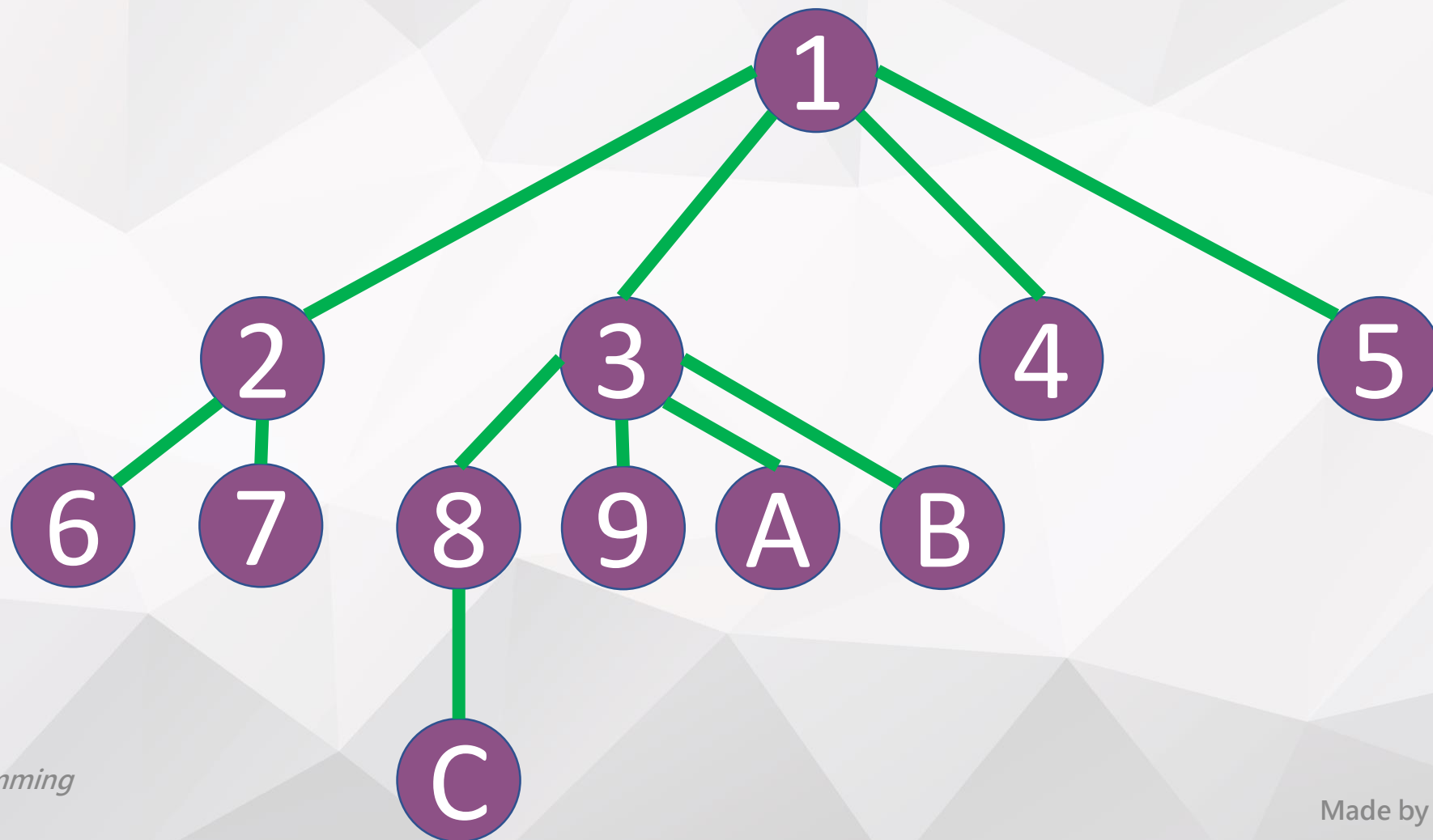
拜訪完



BFS 樹



BFS 樹



Uva 11624 Fire!

Joe 在一個迷宮工作，不幸的是，迷宮中有幾處發生火災，Joe 想要逃離這個迷宮

Joe 和火每分鐘移動一格，可往東西南北四個方向行走，但火每次向四個方向擴散一格

給定一個地圖，給出 Joe 逃離迷宮的最短時間

Uva 11624 Fire!

輸入：

4 4

####

#JF#

#.#

#.#

輸出：

3

Uva 11624 Fire!

輸入：

3 3

###

#J.

#.F

輸出：

IMPOSSIBLE

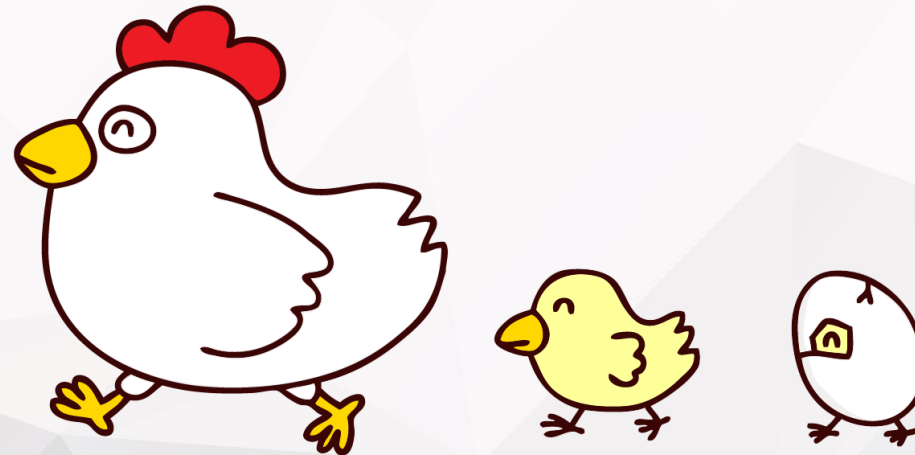
Uva 11624 Fire!

第四週教材裡有題解

BFS 與 DFS 很相似

比較一下

各位可能會發現，兩者程式碼非常相似



DFS 程式碼

```
stack<int> S;  
S.push(root); //root 代表走訪此圖的起點  
vis[root] = true;
```

```
while (!S.empty()) {  
    int u = S.top(); S.pop();  
    for (auto v: E[u]) {  
        if (vis[v]) continue;  
        vis[v] = true;  
        S.push(v);  
    }  
}
```

BFS 程式碼

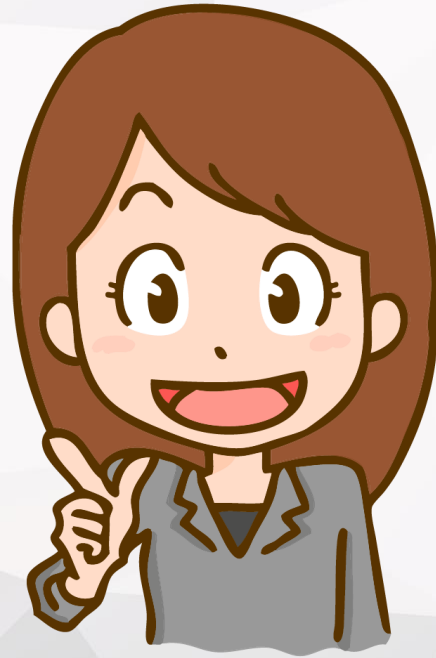
```
queue<int> Q;  
Q.push(root); //root 代表走訪此圖的起點  
vis[root] = true;
```

```
while (!Q.empty()) {  
    int u = Q.front(); Q.pop();  
    for (auto v: E[u]) {  
        if (vis[v]) continue;  
        vis[v] = true;  
        Q.push(v);  
    }  
}
```

複雜度

因為每條邊都會走走看

所以複雜度為 $O(E)$
其中 E 為原圖的總邊數



Questions?
