Advanced Competitive Programming

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

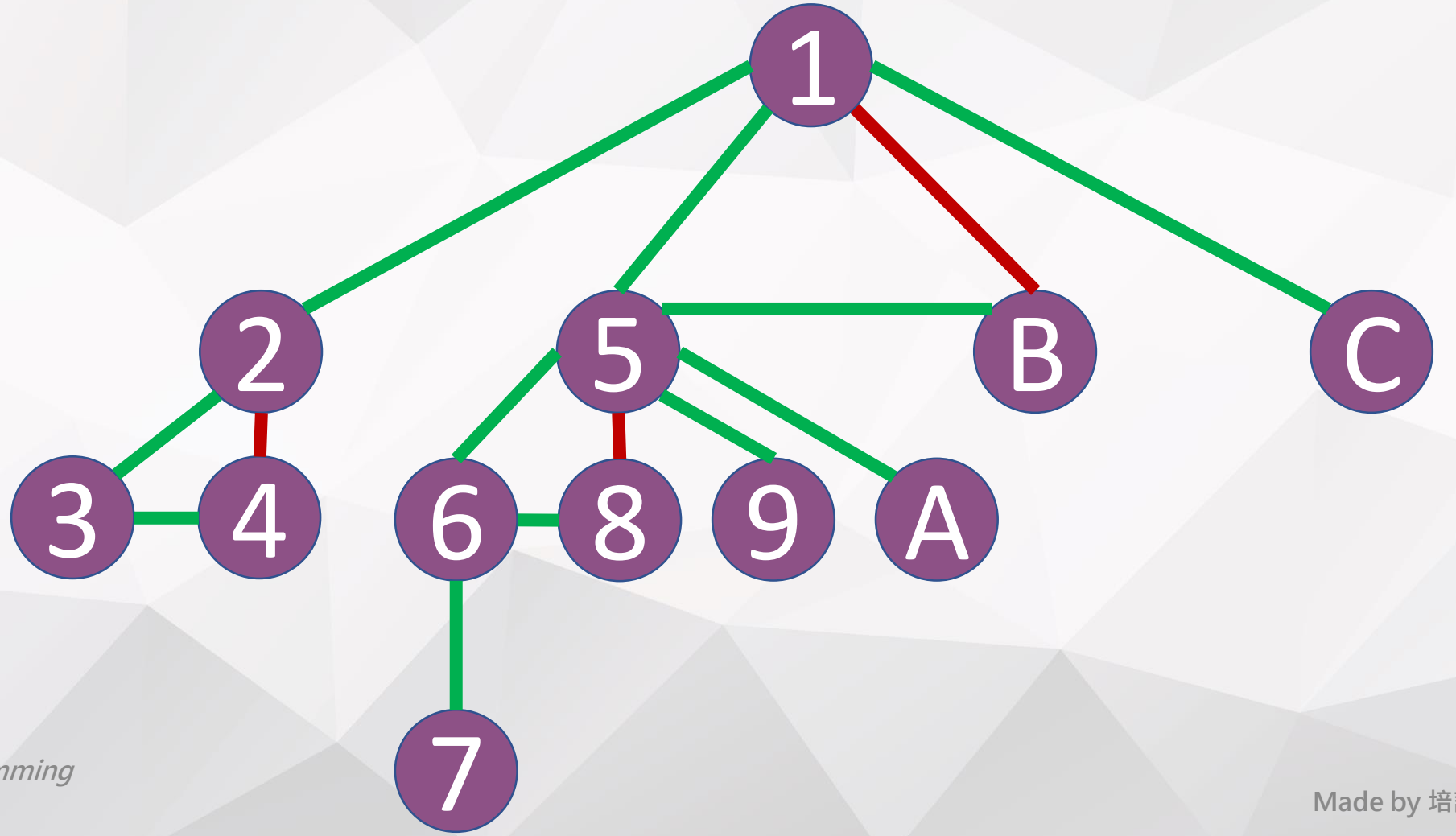
Department of Computer Science and Information Engineering
National Cheng Kung University
Tainan, Taiwan

Outline

- 用 stack 實作 DFS
- 二分搜尋 (Binary Search)

用 stack 實作 DFS

DFS 的**點**遍歷順序



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);
  }
}
```

DFS 實作

第四週給出了 DFS 的兩種實作版本

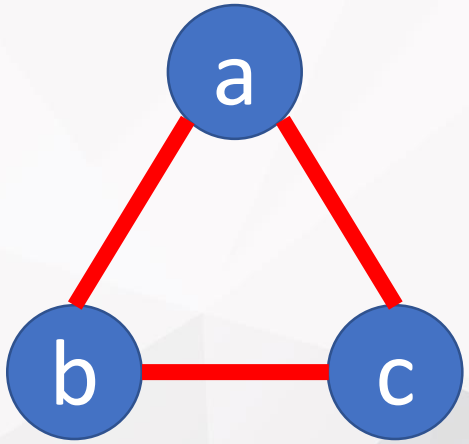
有跑過程式碼或是模擬過一遍的同學，
應該會發現 stack 的實作版本怪怪的

姑且叫它 HRS（我取的）

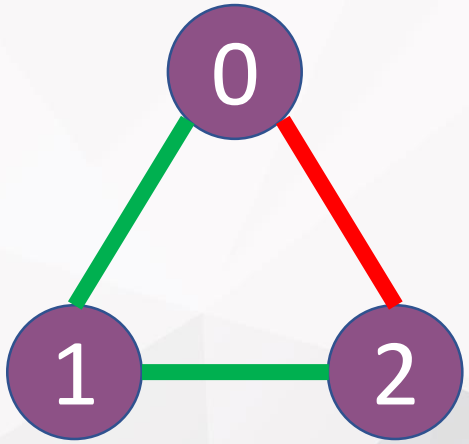
舉個例子

考慮 3 條無向邊

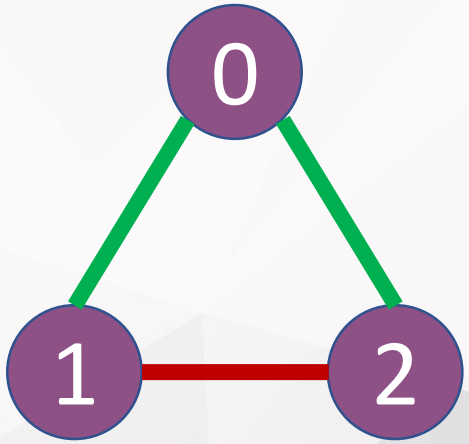
(a, b), (a, c), (b, c)



DFS



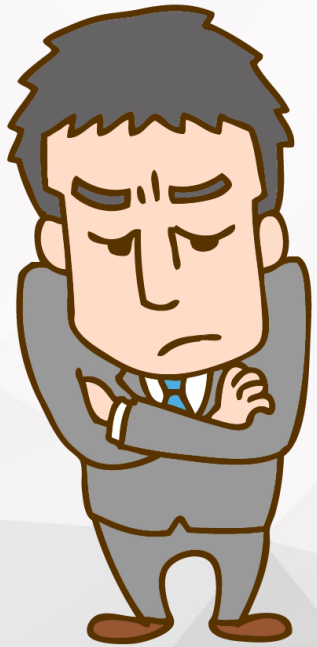
HRS



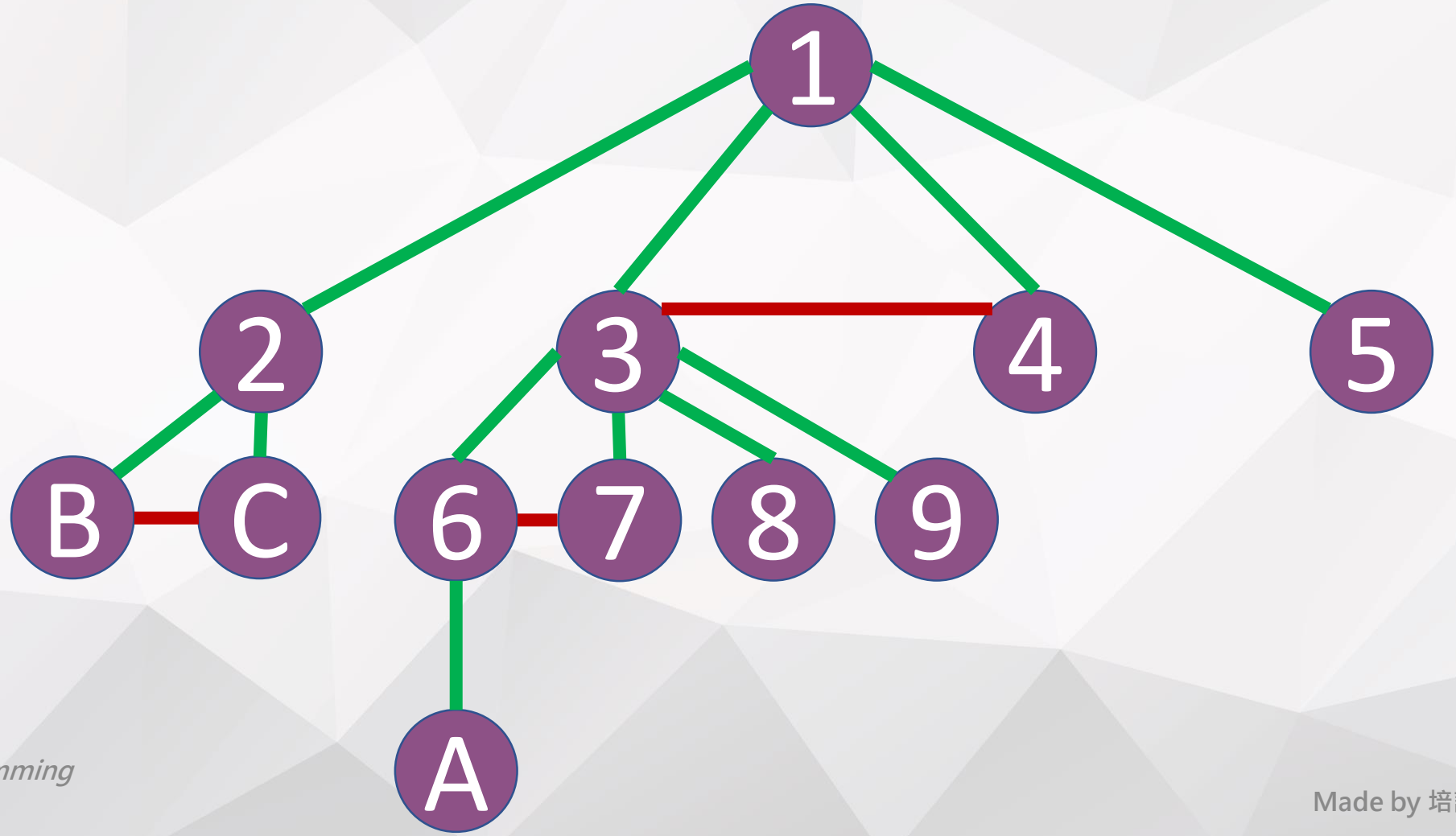
HRS

結論是

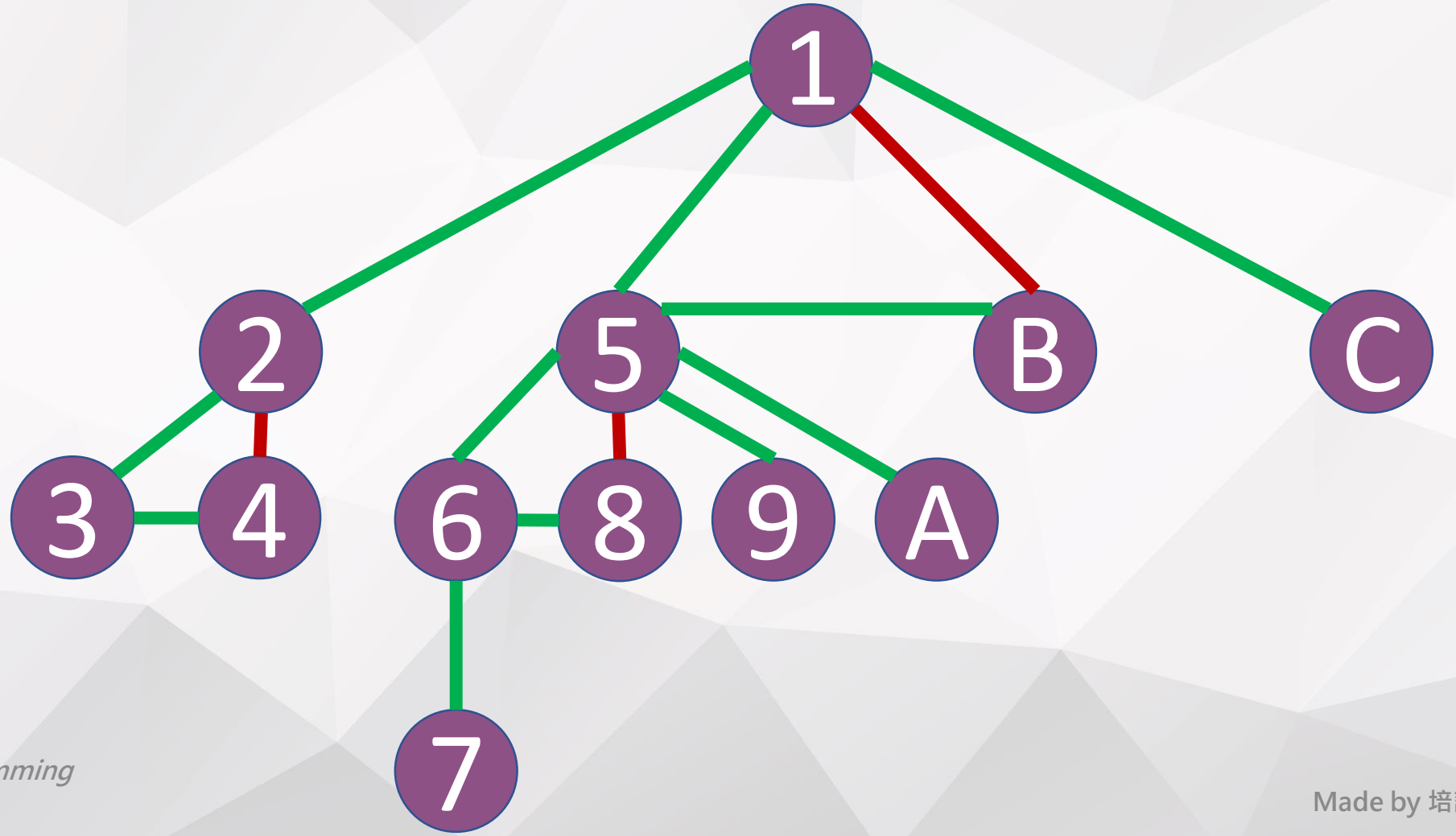
雖然 HRS 可以做到點遍歷
但卻沒有**深度優先**



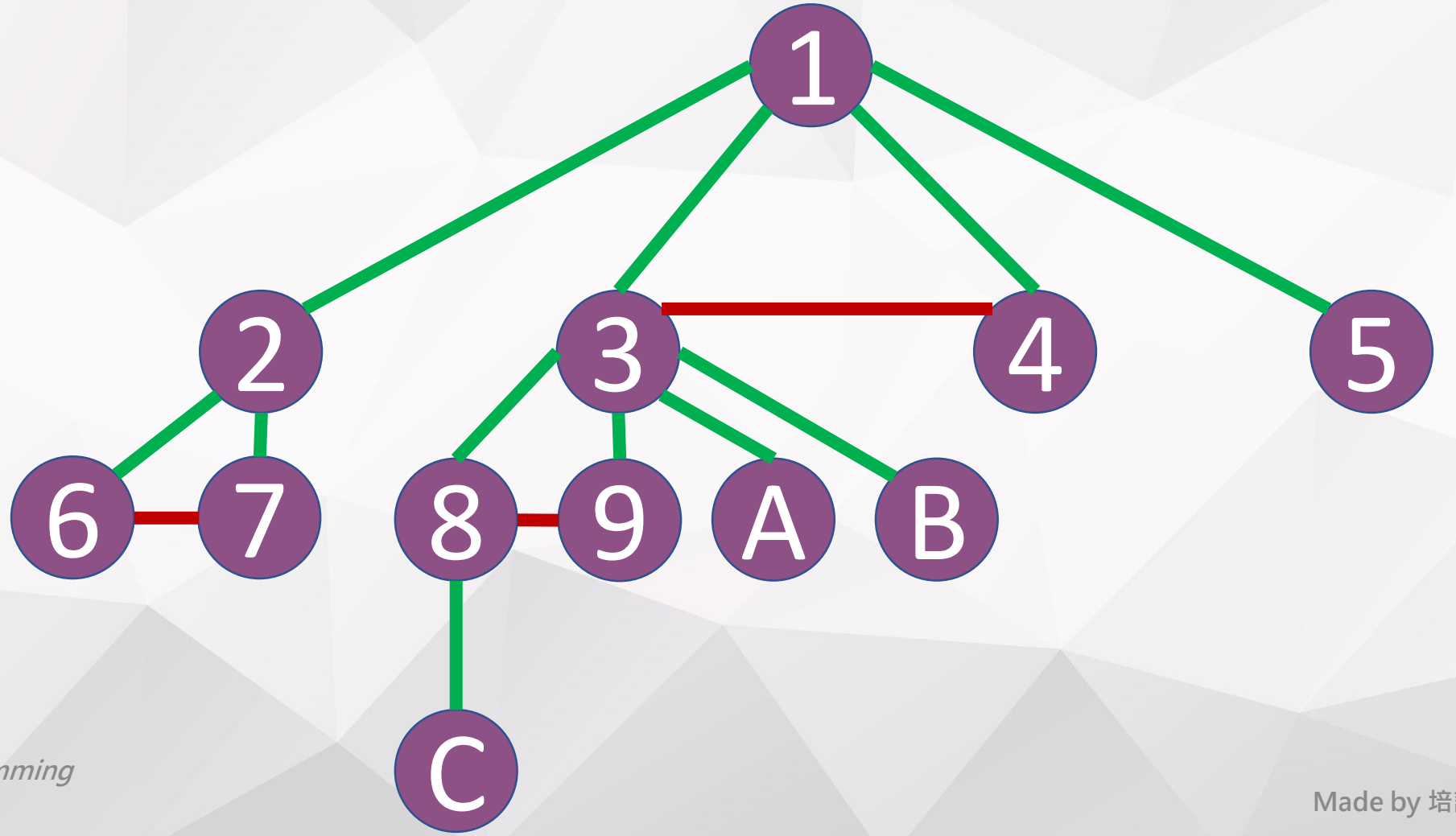
HRS 的點遍歷順序



DFS 的**點**遍歷順序



BFS 的點遍歷順序



HRS

```
stack<int> S;
S.push(root);
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);

  }
}
```

DFS

```
stack<int> S;
S.push(root);
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(u);
    S.push(v);
    break;
  }
}
```

DFS

詳細的內容就請看第四週教材

二分搜尋 (Binary Search)

終極密碼

各位應該都聽過終極密碼

不管有沒有聽過，總之規則如下:

終極密碼

- 兩人以上的遊戲

- 其中一人 P，1 ~ N (N>=1) 中選一個數字(**目標**)，別告訴其他人

- 其他人要想辦法**猜出**這個數字
- P 會告訴猜測者，目前猜的數字**大於**還是**小於**目標
- **等於**時遊戲結束

二分搜尋

- 一開始區間設定為 [0, N]

- 每次猜區間 [L, R] 內的中間值 M

- 如果目標 小於 M
- 區間改為 [L, M]，反之則改為 [M, R]

終極密碼: 二分搜尋

- 例如 [0, 99]，目標值為 42
- 猜 50，區間改 [0, 50]
- 猜 25，區間改 [25, 50]
- 猜 37，區間改 [37, 50]
- 猜 43，區間改 [37, 43]
- 猜 40，區間改 [40, 43]
- 猜 41，區間改 [41, 43]
- 猜 42，區間改 [42, 42]

終極密碼: 二分搜尋

- 共 $\log_2 100 = 6.6438... \leqq 7$ 次猜測 (100 = 99+1)

- 這樣的猜法 複雜度為 O(log N)

推廣一下

如果現在有個**遞增數列**，目標出現**一個以上**

至少要有**兩個位置**(index)，以表達區間內都是目標

這兩個位置分別叫做:

- Lower bound
- Upper bound

區間

例如長度為 8， **起始位置**(index) 為 0
數列 1, 2, 2, 2, 2, 2, 3, 9
目標值為 2

輸出區間 [1, 5] 或是 [1, 6)
又或是 (0, 5]、(0, 6)

都能表達這個區間內的目標值 2

左閉右開

普遍的實作，
會採用 [1, 6) 這樣的**左閉右開**區間

左閉右開的好處，參考第五週的教材

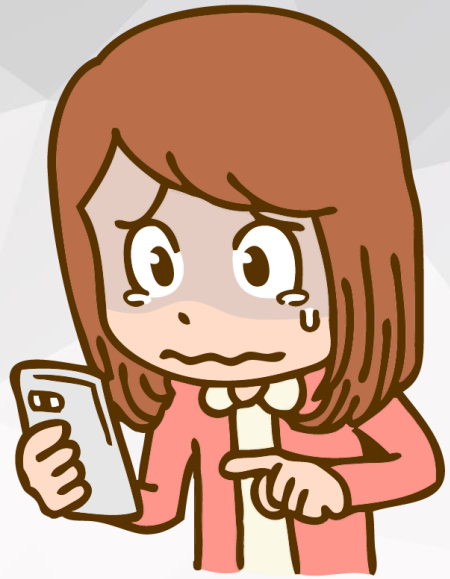
bound

數列 1, 2, 2, 2, 2, 2, 3, 9，目標值為 2

普遍實作中，

- lower bound 為 1
- upper bound 為 6

有個問題



目標值不在數列中怎麼辦?

例如數列 1, 2, 2, 2, 2, 2, 3, 9
當目標值為 42, -42, 5

當目標值不在數列中

數列 1, 2, 2, 2, 2, 2, 3, 9
目標值為 42

- lower bound 為 8
- upper bound 為 8

因為此時 42 若位於 index 8 的位置
1, 2, 2, 2, 2, 2, 3, 9, 42 這樣的數列依然保持遞增(**最適合**)

當目標值不在數列中

數列 1, 2, 2, 2, 2, 2, 3, 9
目標值為 -42

- lower bound 為 0
- upper bound 為 0

因為此時 -42 若位於 index 0 的位置
-42, 1, 2, 2, 2, 2, 2, 3, 9 這樣的數列依然保持遞增(**最適合**)

當目標值不在數列中

數列 1, 2, 2, 2, 2, 2, 3, 9
目標值為 5

- lower bound 為 7
- upper bound 為 7

因為此時 5 若位於 index 7 的位置
1, 2, 2, 2, 2, 2, 3, 5, 9 這樣的數列依然保持遞增(**最適合**)

lower bound : 二分搜尋

回到二等分の搜尋

跟終極密碼一樣，每次只找一半的區間

```
int m = (l+r) / 2; // m := middle
```

lower bound : 二分搜尋

數列 A: 1, 2, 2, 2, 2, 2, 3, 9

目標為 4

lower bound: 7

```
若 A[m] >= 4 則 r = m;          // m 保留
若 A[m] <  4 則 l = m + 1; // m 捨去
```

lower bound : 二分搜尋

有個細節:

```
int m = (l+r) / 2;
```

這個 m 每次都會落在區間內
不會在 r 上，因為它是**開**的

原因是 int 除法會**無條件捨去小數位**

lower bound : 二分搜尋

```
while (l != r) {
   int m = (l+r) / 2;
   if (A[m] >= target) r = m;
   else l = m + 1;
}

return l;
```