AlgoSeacow 

Tree I

Summer A Class 9

關於這堂課

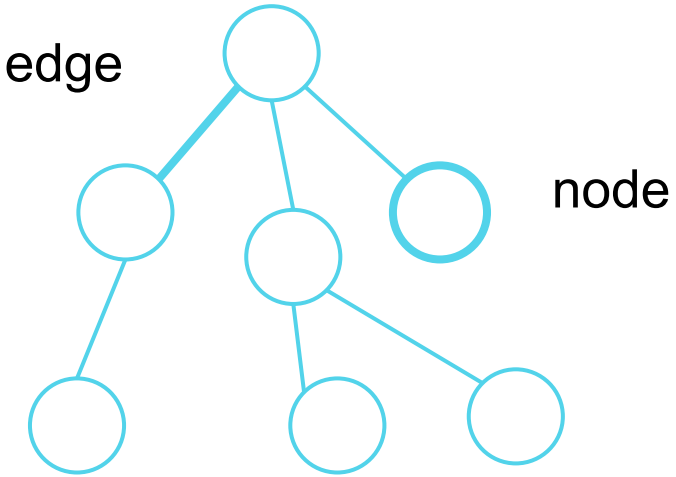
- 先備知識
  - 資料結構 I

- 學習重點
  - Tree 的定義
  - Tree 的走訪
  - Tree 的序列化表示法 (前中後序)
  - Tree DP
  - Tree 的指標格式（以 LeetCode 為例）
  - Appendix - BST (二元搜尋樹)
  - Appendix - Heap (堆)

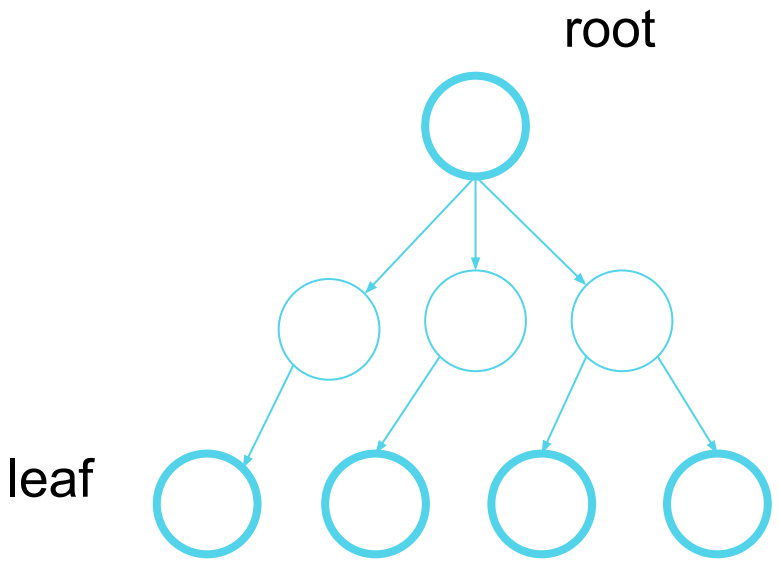
Tree 的定義和儲存方式

Tree 特性

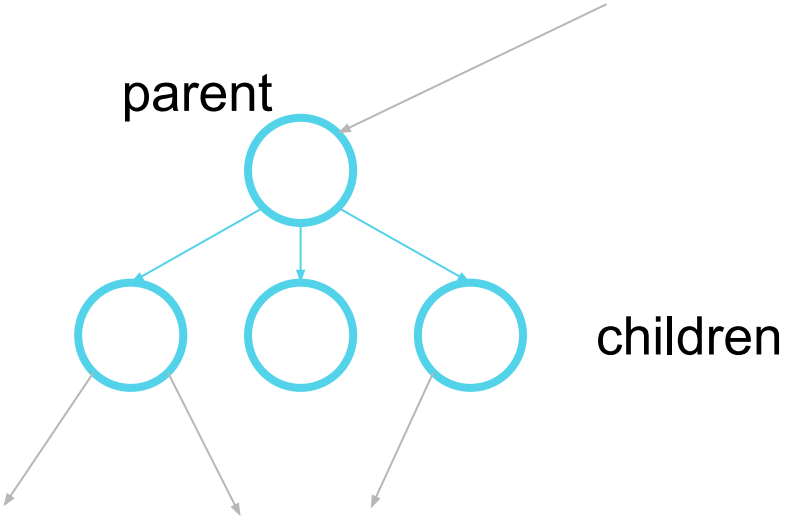
- 由 `n` 個 node 和 `n-1` 條 edge 所組成
- 不存在環
- 任兩點存在唯一路徑



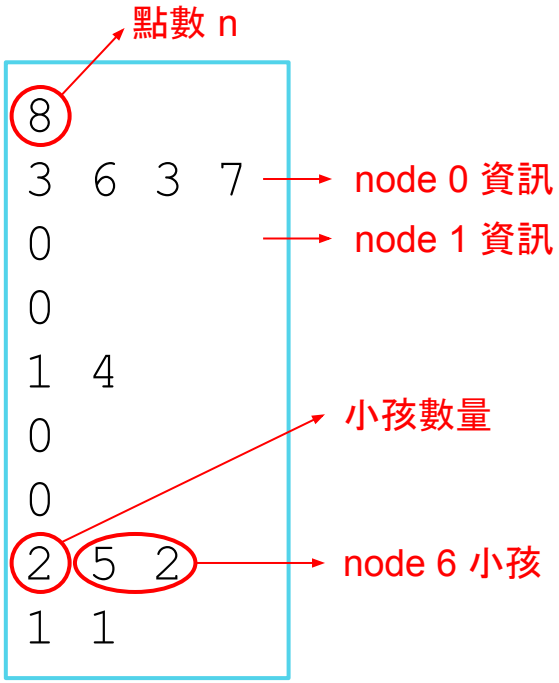
Tree 基本名詞

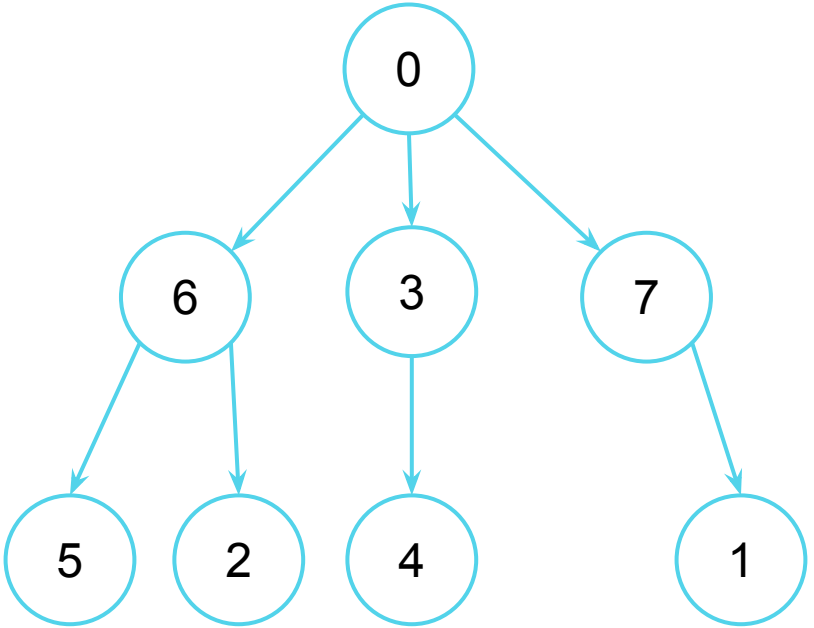


Tree 基本名詞

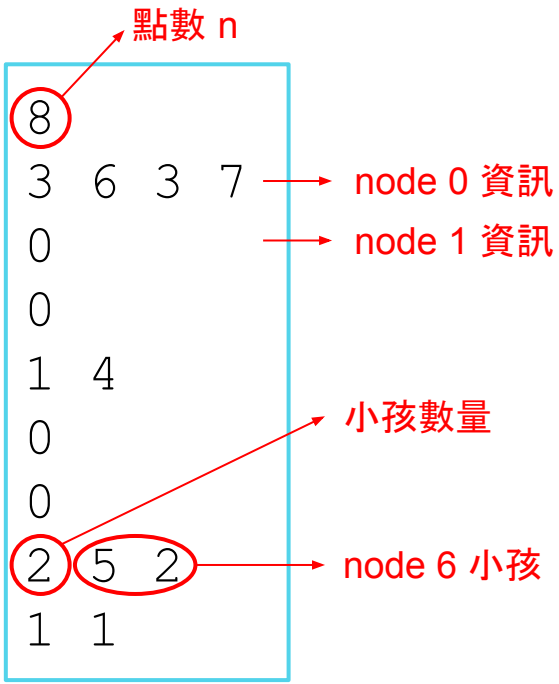


如何儲存 Rooted Tree?



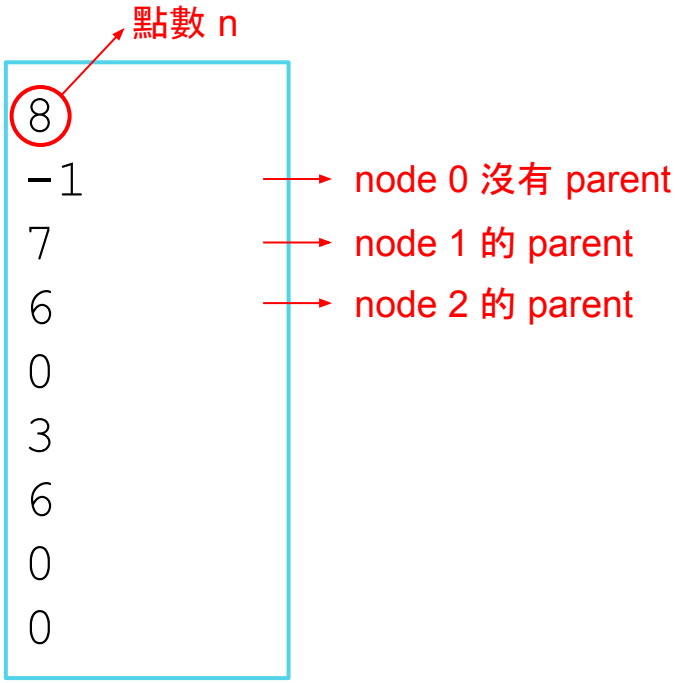


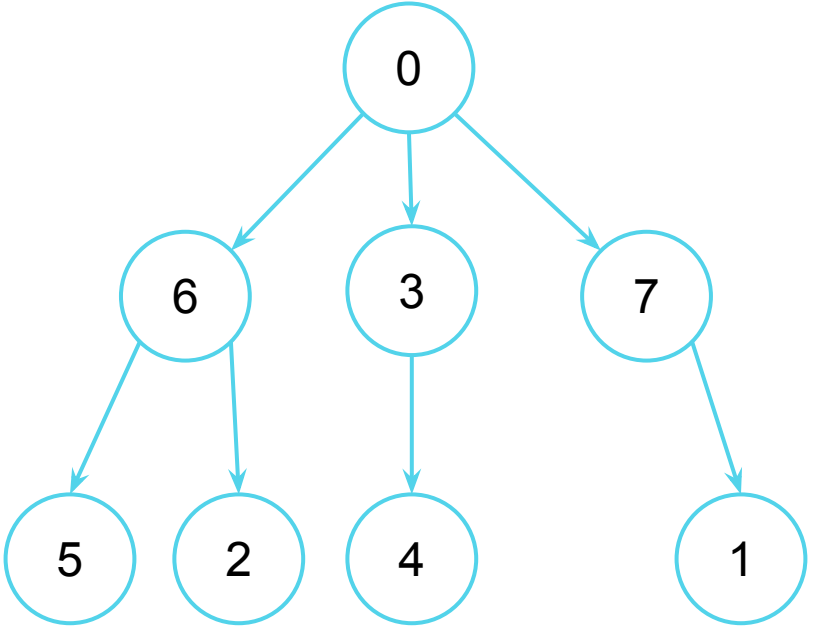
如何儲存 Rooted Tree?



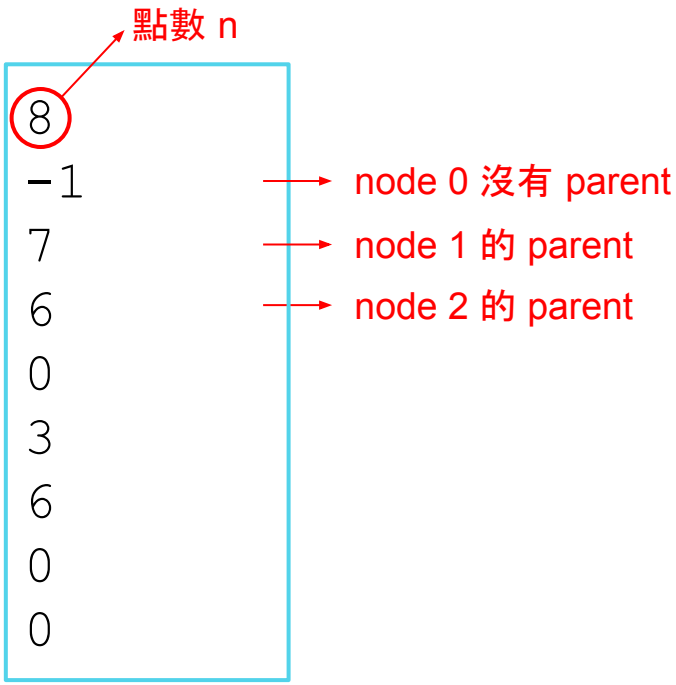

```
const int MAXN = 100005;

int n;
int root = 0;
vector<int> G[MAXN];

void init() {
    cin >> n;
    for (int i = 0; i < n; i++) {
        int k;
        cin >> k;
        G[i].resize(k);
        for (int j = 0; j < k; j++) {
            cin >> G[i][j];
        }
    }
}
```

如何儲存 Rooted Tree?



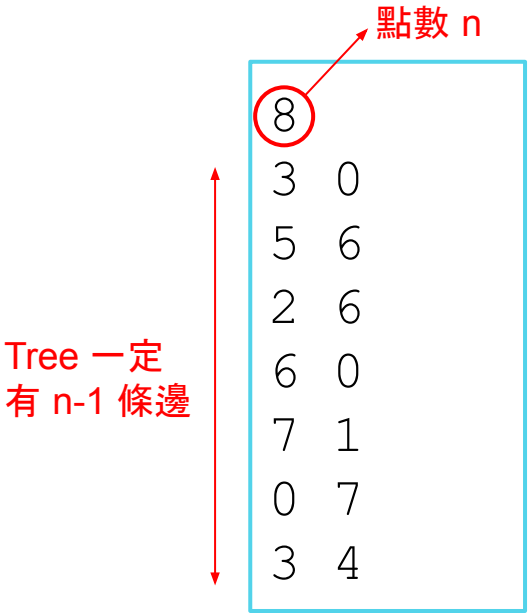


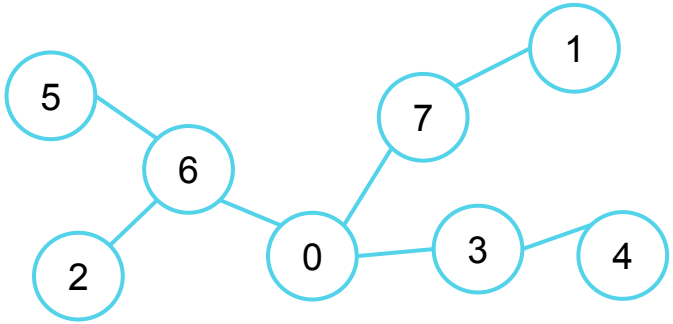
如何儲存 Rooted Tree?

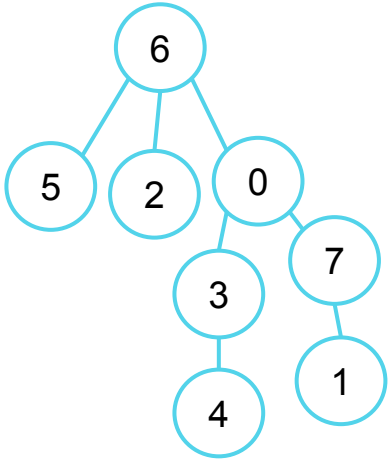


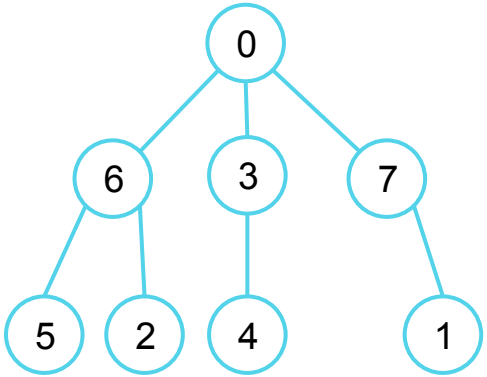

```
const int MAXN = 100005;

int n;
int root = 0;
vector<int> G[MAXN];

void init() {
    cin >> n;
    for (int i = 0; i < n; i++) {
        int par;
        cin >> par;
        if (par == -1) {
            root = par;
        } else {
            G[par].push_back(i);
        }
    }
}
```

如何儲存 Unrooted Tree?

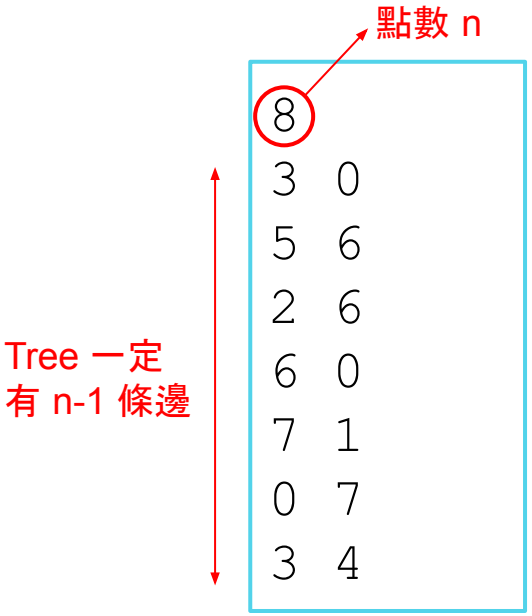








如何儲存 Unrooted Tree?

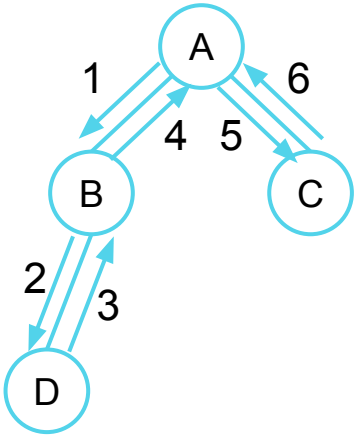



```
const int MAXN = 1e5 + 5;

int n;
int root = 0;
vector<int> G[MAXN];

void init() {
    cin >> n;
    for (int i = 0; i < n - 1; i++) {
        int u, v;
        cin >> u >> v;
        G[u].push_back(v);
        G[v].push_back(u);
    }
}
```

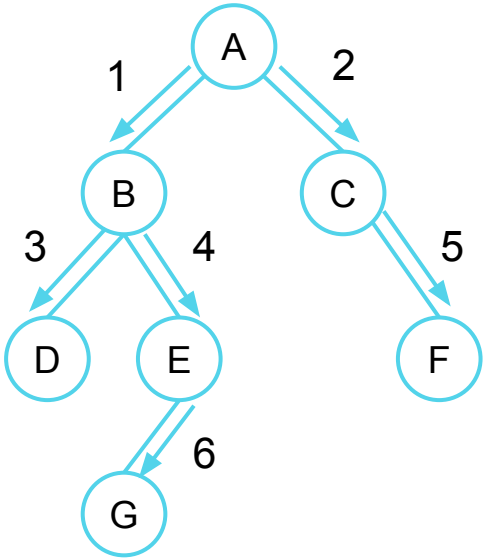
Tree 的走訪

Tree Traversal

- **D**epth **F**irst **S**earch (DFS)
  深度優先搜尋
  - 走迷宮



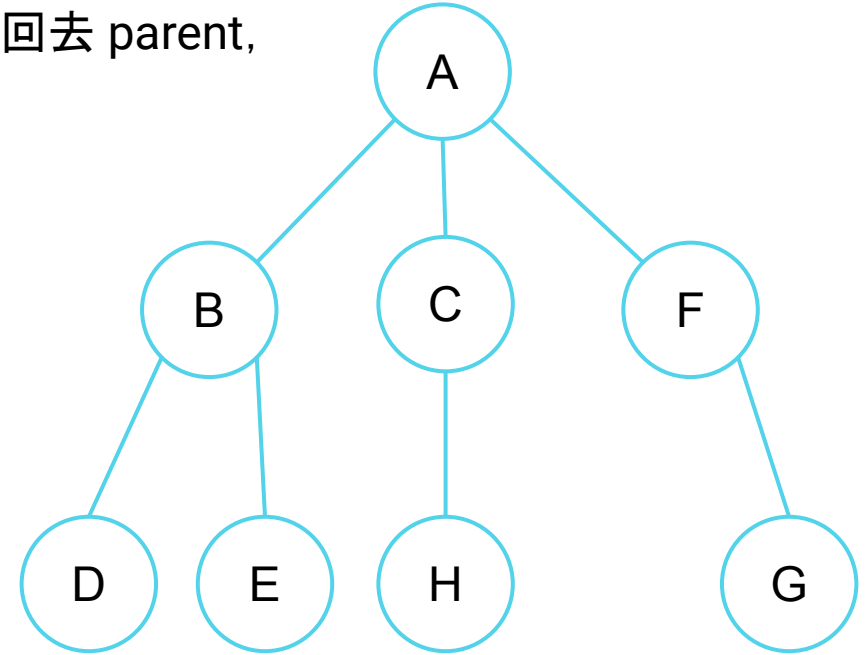
- **B**readth **F**irst **S**earch (BFS)
  廣度優先搜尋
  - 倒水



Tree DFS

- 對於每一個節點, children 都走完才退回去 parent, 否則走訪其中一個 children

- 右圖 DFS 順序
  - _____



Rooted Tree DFS

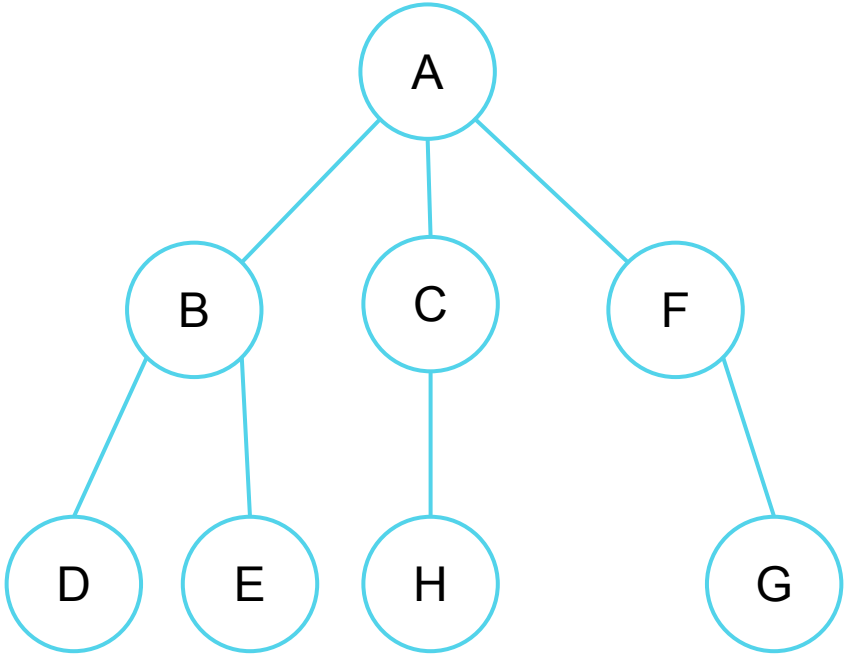
```
void dfs(int u) {
    for (int i = 0; i < (int)G[u].size(); i++) {
        int v = G[u][i];
        dfs(v);
    }
}
```


Unrooted Tree DFS

```
void dfs(int u, int par) {
    for (int i = 0; i < (int)G[u].size(); i++) {
        int v = G[u][i];
        if (v == par)
            continue;
        dfs(v, u);
    }
}
```


Tree BFS

- 一層一層往下走
- 右圖 BFS 順序
  - _____



Rooted Tree BFS

```
void bfs(int s) {
    queue<int> q;
    q.push(s);
    while (q.size()) {
        int u = q.front();
        q.pop();
        for (int i = 0; i < (int)G[u].size(); i++) {
            int v = G[u][i];
            q.push(v);
        }
    }
}
```

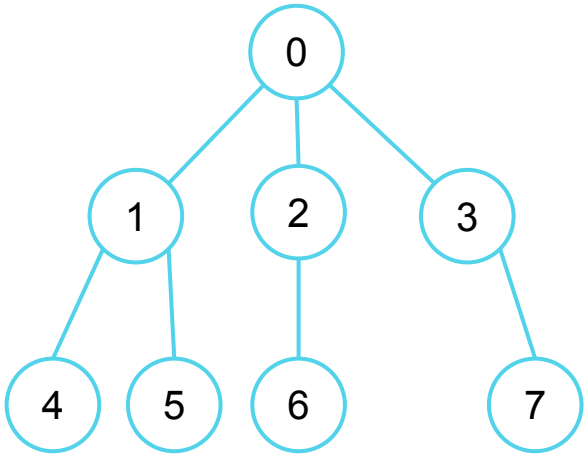

Unrooted Tree BFS

```
void bfs(int s) {
    queue<int> q;
    q.push(s);
    vis[s] = 1;
    while (q.size()) {
        int u = q.front();
        q.pop();
        for (int i = 0; i < (int)G[u].size(); i++) {
            int v = G[u][i];
            if (vis[v])
                continue;
            q.push(v);
            vis[v] = 1;
        }
    }
}
```


CSES - Subordinates

- 題目敘述
  - 給一個公司結構(上司下屬關係)，求每個員工的總下屬有幾位

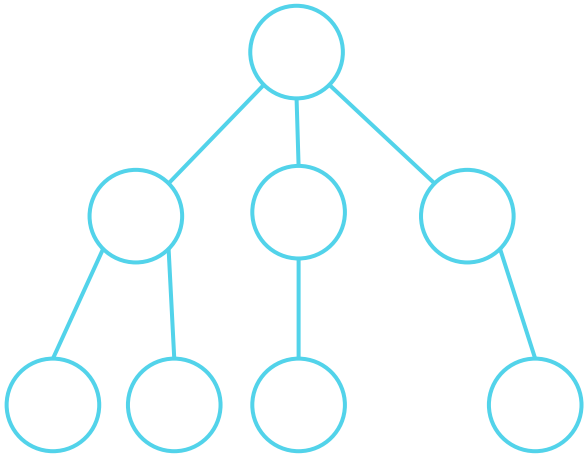
- 測資範圍
  - 員工數 ≤ 200000



CSES - Subordinates

- 按照樹 DFS 的順序對每個節點維護該子樹的子樹大小

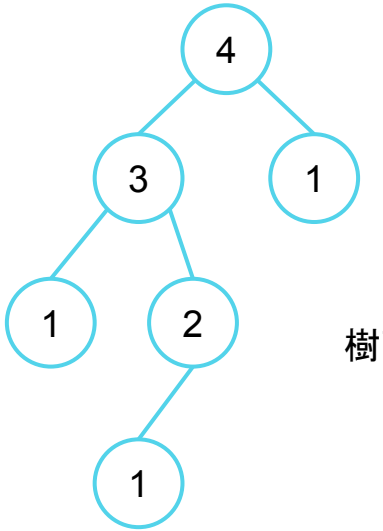
- 當所有 children 都算完子樹大小後，
  parent 的答案就是所有 children 相加
  再加一。



TCIRC d025: 樹的高度與根 (APCS201710)

- 題目敘述
  - 給一棵有根樹, 找出 root 並計算高度總和

- 測資範圍
  - 節點數 n ≤ 100000

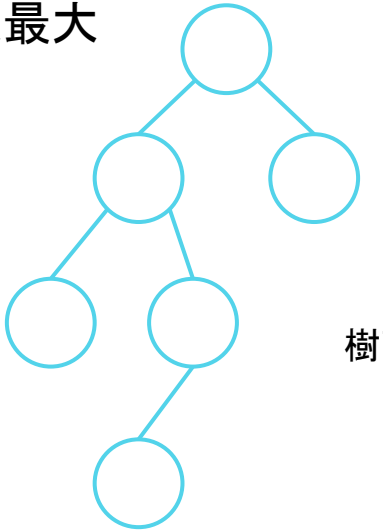


樹高度: 每個點往下最多可以走幾步

TCIRC d025: 樹的高度與根 (APCS201710)

- 按照樹 DFS 的順序對每個節點的子樹高度

- 當所有 children 都算完子樹高度後，
  parent 的答案就是所有 children 取最大
  再加一。

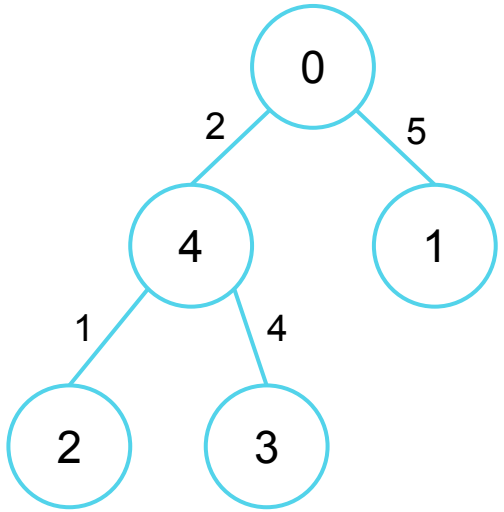


樹高度：每個點往下最多可以走幾步

TCIRC d102: 樹上的推銷員

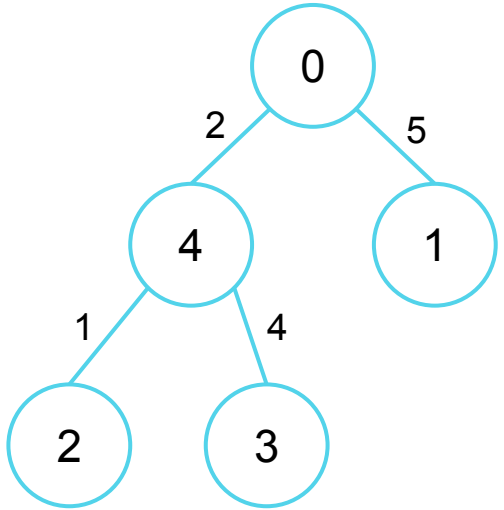
- 題目敘述
  - 有一個推銷員要走訪n 個城市並在結束後回到出發的城市。這些城市以n-1 條道路連接，每條道路連接兩個不同的城市並且雙向可以通行
  - 請你找出一個長度最短的程式走訪順序，因為這樣的順序很多，你必須輸出字典順序最小的那一個

- 測資範圍
  - 城市個數 n ≤ 50000，道路長度不超過100



TCIRC d102: 樹上的推銷員

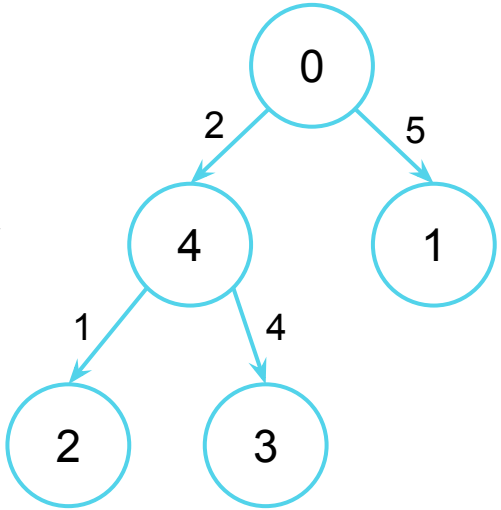
- 觀察其中一種路徑可以發現每條邊都恰走兩次
  - 為一個 DFS 順序
- 最小字典序 DFS
  - 從最小的點開始走
  - 每次盡量走沒走過的最小編號小孩



TCIRC d103: 物流派送系統

- 題目敘述
  - 給一個 rooted tree 包含 n 個節點與 n-1 條有向邊，且每條邊有一個長度
  - 要求 1: 計算從 root 出發長度總和最大的路徑長度
  - 要求 2: 計算從 root 出發經過邊數最多的路徑邊數

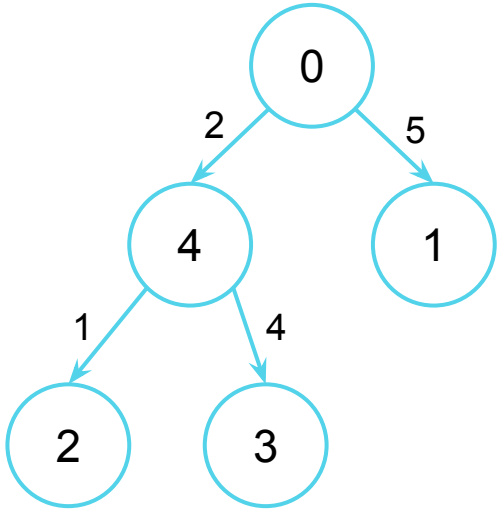
- 測資範圍
  - 節點數 n ≤ 100000，每條邊長度是不超過1000 的整數



TCIRC d103: 物流派送系統

- 要求 1: 計算從 root 出發長度總和最大的路徑長度
  - DFS 的過程中累加經過的路徑長度，比較所有走到葉節點的路徑

- 要求 2: 計算從 root 出發經過邊數最多的路徑邊數
  - DFS 的過程中紀錄走過幾條邊，比較所有走到葉節點的路徑



TCIRC d105: 自動分裝 (APCS202002)

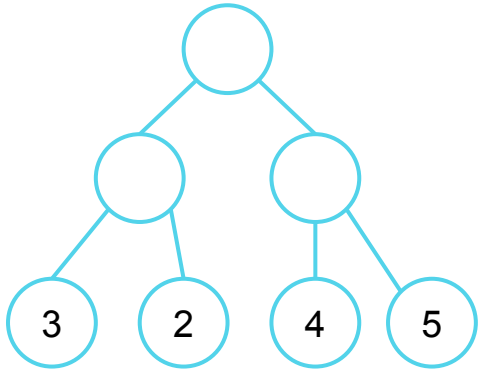
- 題目敘述
  - 給一個有根二元樹，每個葉節點都有一個重量，子樹的重量為該子樹下所有葉節點的總和
  - 放入 m 個物品，物品從根進入樹後不斷往重量比較輕的子樹移動，若兩邊一樣重則往左子樹移動，走到葉節點後該節點重量加w，並輸出葉節點編號

- 測資範圍
  - 葉節點數量 n ≤ 100000，查詢次數 m ≤ 100

TCIRC d105: 自動分裝 (APCS202002)

- 按照給定的規則模擬下去

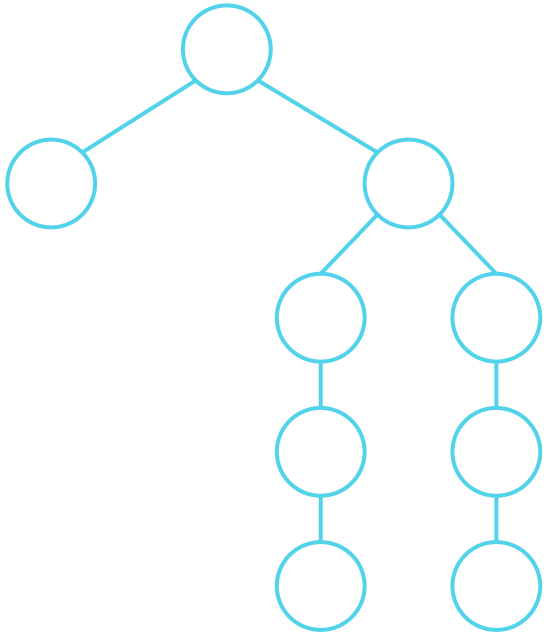
加入 5 個物品重量: 5 4 3 2 1



CSES - Tree Diameter

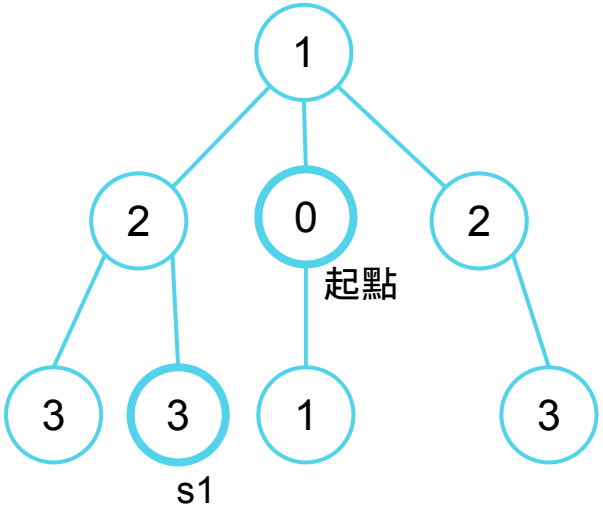
- 題目敘述
  - 給一棵樹，求樹直徑長度
  - 樹直徑為距離最遠的兩點所形成的路徑

- 測資範圍
  - n ≤ 200000

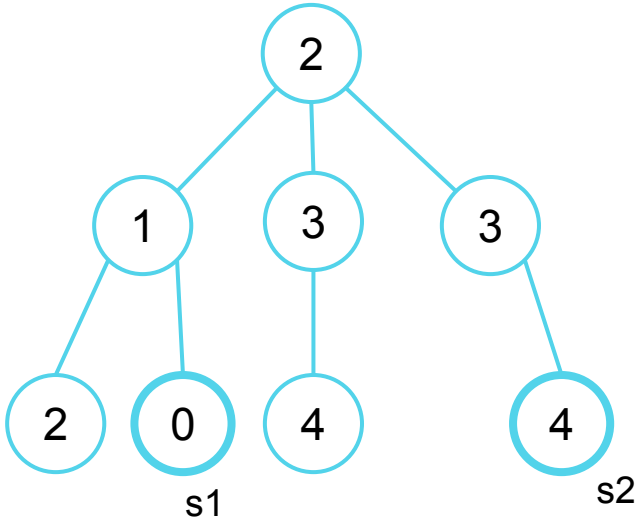


CSES - Tree Diameter

- 做兩次 DFS
  1. 挑選任一 node 當起點做 BFS, 做出每個 node 的 Level



2. 挑選一個最大 Level 的 node s1, 再做一次 BFS, 挑選任意最大 Level 的 node s2, s1 到 s2 即為所求。



題單 - Tree DFS

- CSES - Subordinates (有根樹 DFS, 求子樹大小)
- TCIRC d025: 樹的高度與根 (bottom-up) (APCS201710)
- TCIRC d103: 物流派送系統 (有根樹 DFS / BFS, 帶權重最長路)
- TCIRC d105: 自動分裝 (APCS202002) (DFS / BFS 模擬)
- Zerojudge a249: 00679 - Dropping Balls
- TCIRC d102: 樹上的推銷員 (無根樹 DFS)
- codeforces 115A. Party

二元樹的序列化表示法

二元樹的序列化

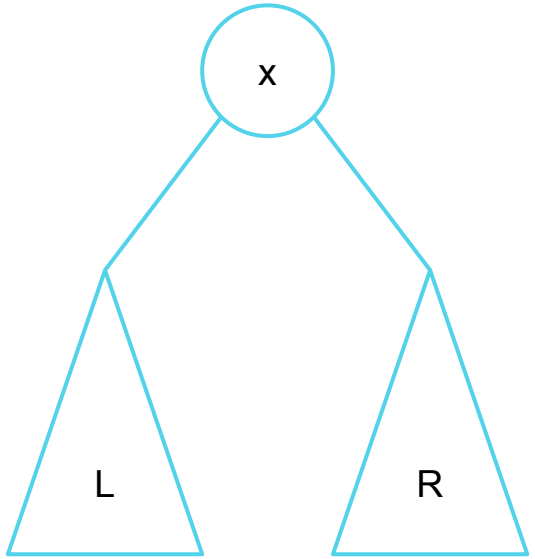
- 將一個二元樹用字串或是序列來表示

- 二元樹序列化種類
  - 前序
  - 中序
  - 後序

樹的序列化 - 前序

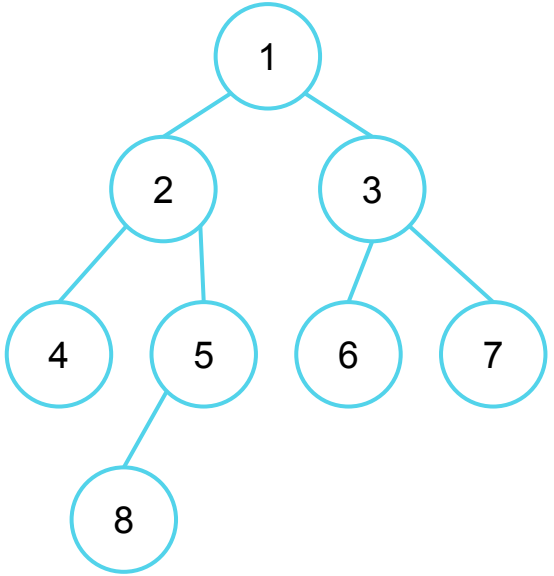
- 給一棵樹如右圖，x 代表根而 L 和 R 代表左右子樹

- 將該樹序列化的方式為
  1. 輸出根節點
  2. 輸出左子樹的序列化序列
  3. 輸出右子樹的序列化序列



樹的序列化 - 前序

- 寫出右圖的前序序列



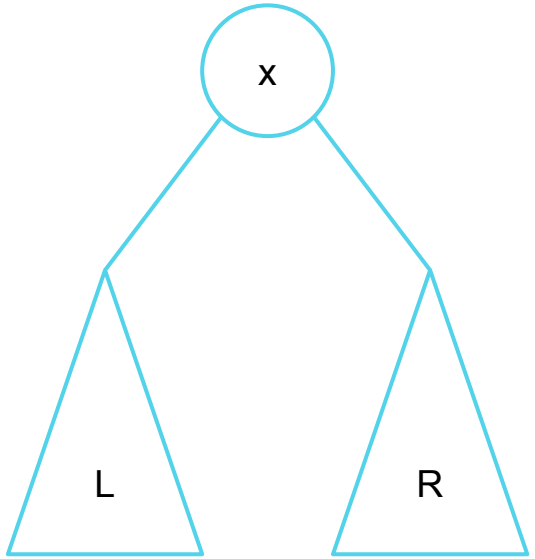
樹的序列化 - 前序

```
void preorder(int u) {
    if (u == 0)
        return ;

    cout << u << ' ';
    preorder(G[u][0]);
    preorder(G[u][1]);
}
```


樹的序列化 - 中序

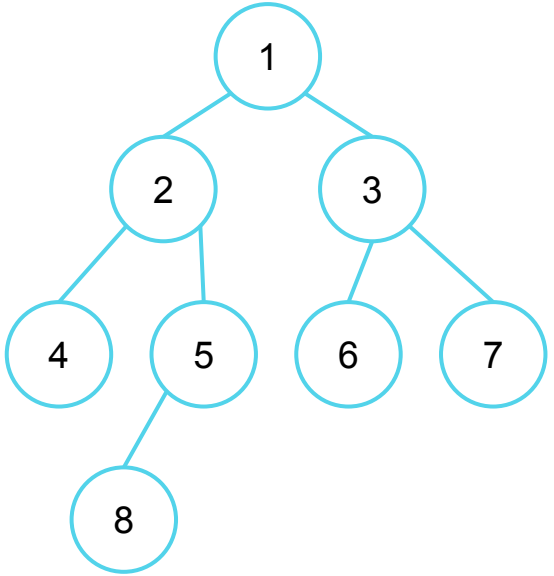
- 給一棵樹如右圖，x 代表根而 L 和 R 代表左右子樹

- 將該樹序列化的方式為
  1. 輸出左子樹的序列化序列
  2. 輸出根節點
  3. 輸出右子樹的序列化序列



樹的序列化 - 中序

- 寫出右圖的中序序列



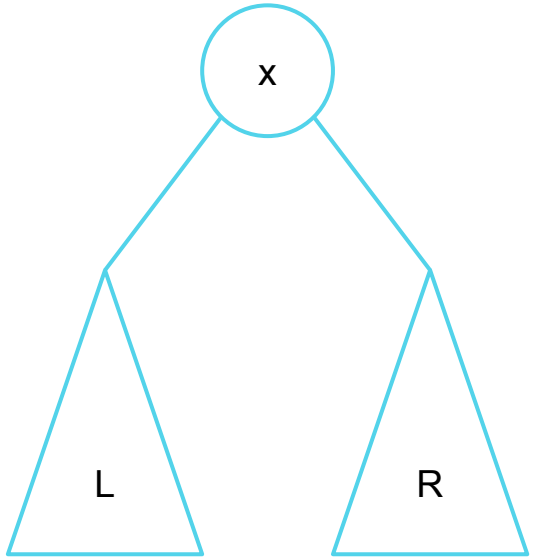
樹的序列化 - 中序

```
void inorder(int u) {
    if (u == 0)
        return ;

    inorder(G[u][0]);
    cout << u << ' ';
    inorder(G[u][1]);
}
```


樹的序列化 - 後序

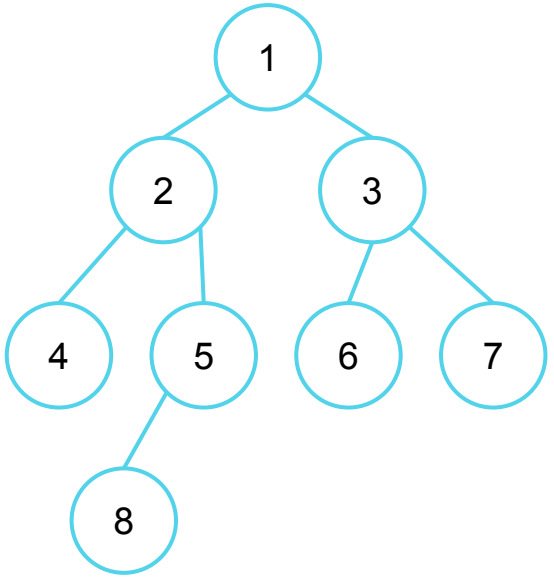
- 給一棵樹如右圖，x 代表根而 L 和 R 代表左右子樹

- 將該樹序列化的方式為
  1. 輸出左子樹的序列化序列
  2. 輸出右子樹的序列化序列
  3. 輸出根節點



樹的序列化 - 後序

- 寫出右圖的後序序列



樹的序列化 - 後序

```
void postorder(int u) {
    if (u == 0)
        return ;

    postorder(G[u][0]);
    postorder(G[u][1]);
    cout << u << ' ';
}
```


APCS 筆試題 - 運算式中序轉後序

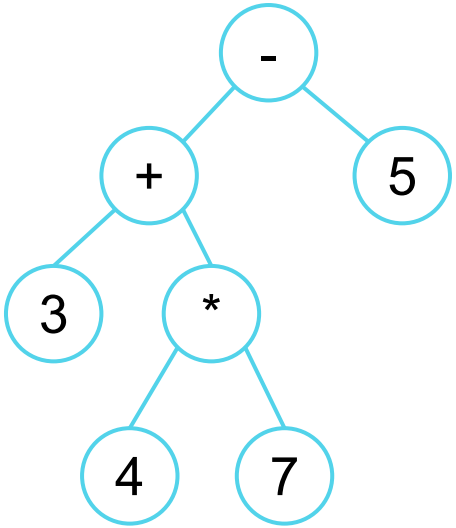
- 給一個中序運算式，轉換成後序運算式
- 運算式的計算順序依序為先乘除後加減，相同優先順序由左至右算

中序：3 + 4 * 7 - 5
後序：3 4 7 * + 5 -

APCS 筆試題 - 運算式中序轉後序

中序 : 3 + 4 * 7 - 5

- 將中序運算式按照運算順序畫成樹狀圖
- 運用後序方式寫出後序運算式



ZeroJudge c126: 00536 - Tree Recovery

- 題目敘述
  - 給樹的前序中序走訪，輸出後序

- 測資範圍
  - 樹編號都是大寫英文字母且不重複，最多26 個

前序： DBACEGF

中序： ABCDEFG

後序： ACBFGED

ZeroJudge c126: 00536 - Tree Recovery

- 可以觀察到，前序的第一個字母一定是根節點，並由中序來判斷如何區分兩子樹的位置，並遞迴下去建立樹
- 最後再用後序走訪輸出答案

前序： DBACEGF

中序： ABCDEFG

後序： ACBFGED

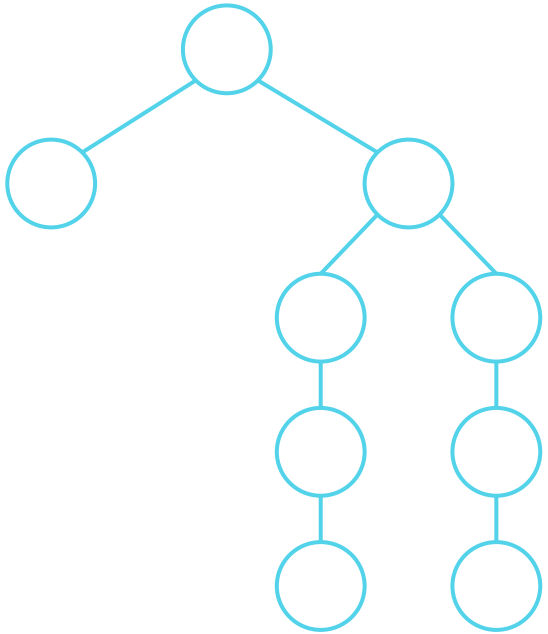
- 延伸題目 : 中序後序求前序

Tree DP

CSES - Tree Diameter

- 題目敘述
  - 給一棵樹，求樹直徑長度
  - 樹直徑為距離最遠的兩點所形成的路徑

- 測資範圍
  - n ≤ 200000



CSES - Tree Diameter

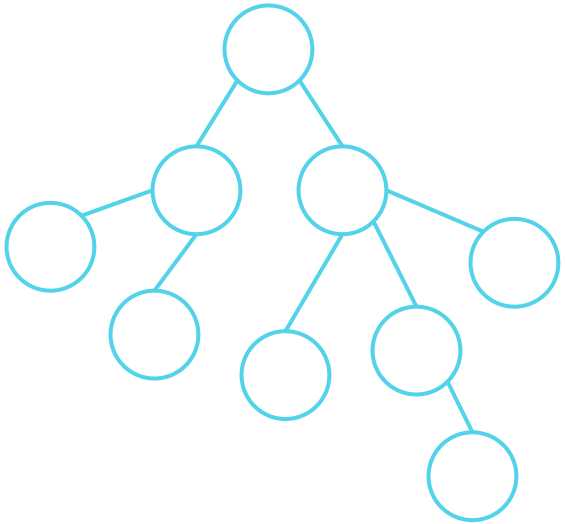
- 狀態
  - dp(u) 代表以 u 為根的子樹中，最長的樹直徑

- 轉移
  - 以 u 為根的子樹中，最長的樹直徑分成兩種類別
    - 有通過 u
      - 找到最深的前兩棵子樹高度
    - 沒有通過 u
      - 每個子樹的 dp 狀態最大值
  - 兩種 case 找最大即為所求

TCIRC d107 樹的最大獨立集

- 題目敘述
  - 輸入一棵有 n 個點的樹，我們要挑選一群彼此不相鄰的點，而且挑選的點越多點越好。請計算最多可以挑多少點

- 測資範圍
  - 節點數量 n ≤ 100000



TCIRC d107 樹的最大獨立集

- 狀態
  - dp(u, 0) 為以 u 為根的子樹並且不選 u 點的最大獨立集
  - dp(u, 1) 為以 u 為根的子樹並且要選 u 點的最大獨立集

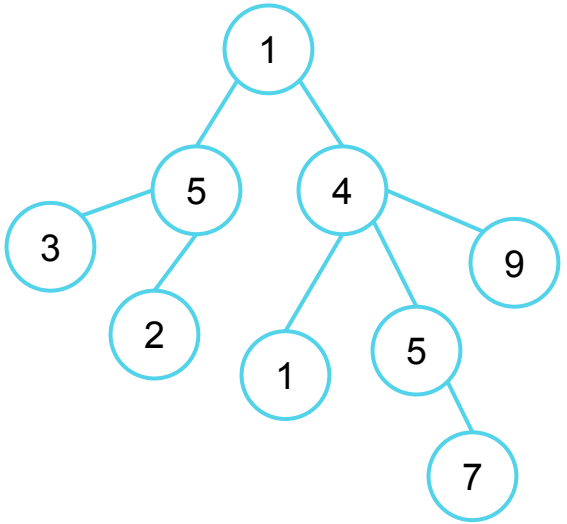
- 轉移
  - dp(u, 0) 為不選 u 點，所以對於每一個小孩 v 取 max(dp(v, 0), dp(v, 1)) 並加起來

  - dp(u, 1) 為要選 u 點，所以對於每一個小孩 v 取 dp(v, 0) 並加起來再加一

TCIRC d109 公司派對 (NCPC)

- 題目敘述
  - 輸入一棵有 n 個點的樹，每個節點有一個權重
  - 挑選一些彼此不相鄰的點，選到的權重總和愈大愈好

- 測資範圍
  - 節點數量 n ≤ 100000



TCIRC d109 公司派對 (NCPC)

- 狀態
  - dp(u, 0) 為以 u 為根的子樹並且不選 u 點的最大權獨立集
  - dp(u, 1) 為以 u 為根的子樹並且要選 u 點的最大權獨立集

- 轉移
  - dp(u, 0) 為不選 u 點，所以對於每一個小孩 v 取 max(dp(v, 0), dp(v, 1)) 並加起來

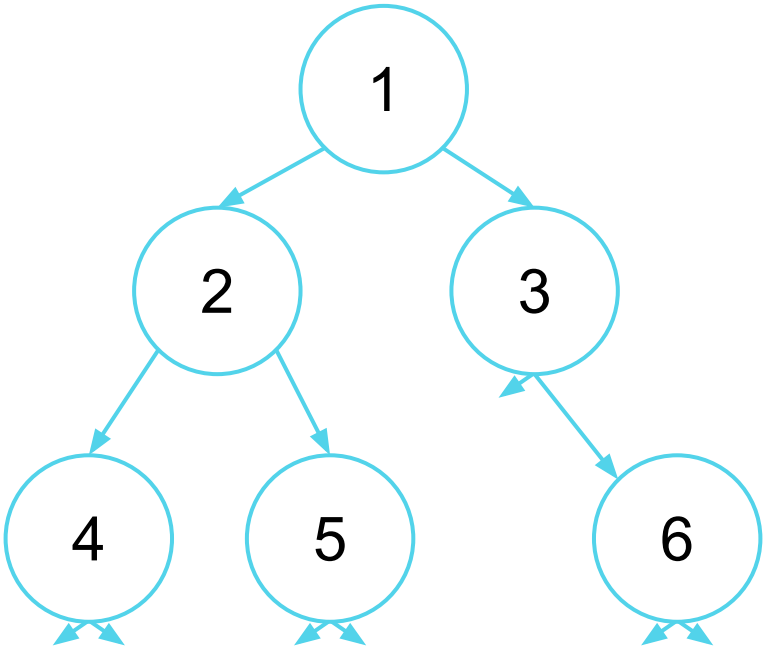
  - dp(u, 1) 為要選 u 點，所以對於每一個小孩 v 取 dp(v, 0) 並加起來再加上 u 的權重

題單 - Tree DP

- CSES - Tree Diameter
- TCIRC d107: 樹的最大獨立集 (樹 DP)
- TCIRC d109: 公司派對 (NCPC) (樹 DP, 樹上最大權重獨立集)
- CSES - Tree Matching (樹 DP, 樹上最大匹配 = 最大獨立集)
- TCIRC d116: 病毒演化 (APCS202007)

Tree 的指標格式
（以 LeetCode 為例）

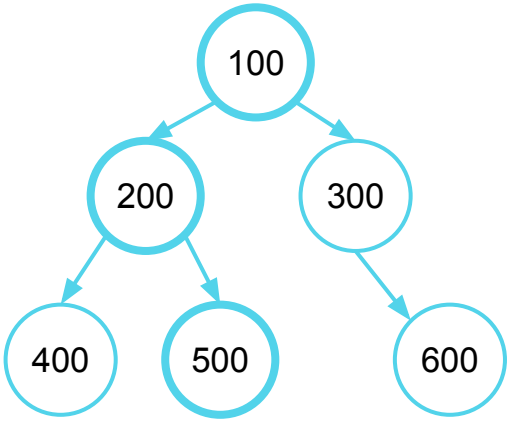
Binary Tree Representation (Pointer)




```
struct TreeNode {
    TreeNode *left;
    TreeNode *right;
    int val;
};
```

Binary Tree Representation (Pointer)

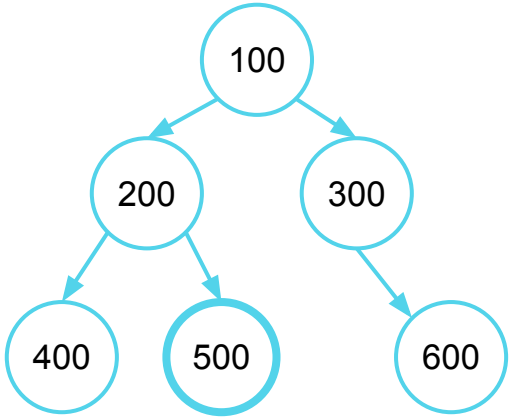
```
struct TreeNode {
    TreeNode *left;
    TreeNode *right;
    int val;
};
```




```
// 建立一個 node 當 root 並將 val 設為 100
TreeNode *root = new TreeNode();
root->val = 100;

// 建立 root 的 left child
root->left = new TreeNode();
root->left->val = 200;

// 建立 root 的 left child 的 right child
root->left->right = new TreeNode();
root->left->right->val = 500;
```


Binary Tree Representation (Pointer)

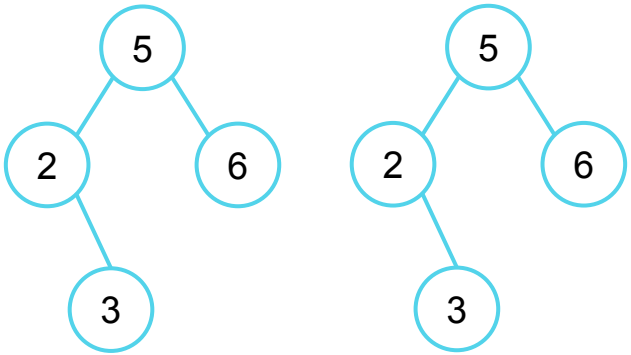
```
struct TreeNode {
  TreeNode *left;
  TreeNode *right;
  int val;
};

// 求 root 的 left child 的 right child 的 val
// pointer 寫法

TreeNode ptr* = root; // 指向 root

ptr = ptr->left; // 先往左走，此時 ptr 指向右圖 val 為 200 的 node

ptr = ptr->right; // 再往左走，此時 ptr 指向右圖 val 為 500 的 node

cout << ptr->val << endl; // 輸出 500
```



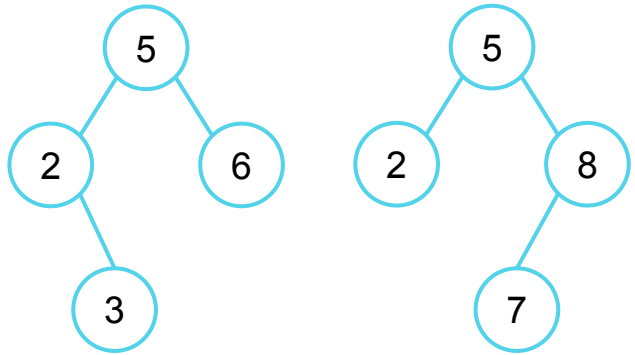
LeetCode 100. Same Tree

- 題目敘述
  - 給兩棵 BST, 判斷是否相同
- 測資範圍
  - node 最多 100 個

true



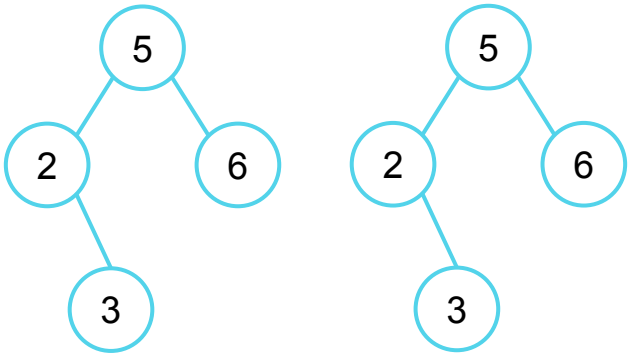
false



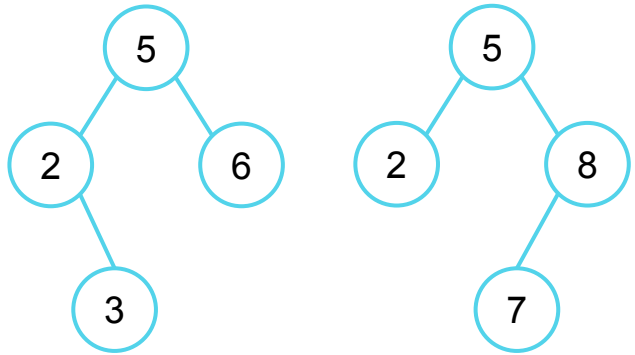
LeetCode 100. Same Tree

- 判斷兩棵樹是否相同的方式為以下條件
  - 根節點數值是否相同
  - 兩棵樹的左子樹是否相同
  - 兩棵樹的右子樹是否相同

true



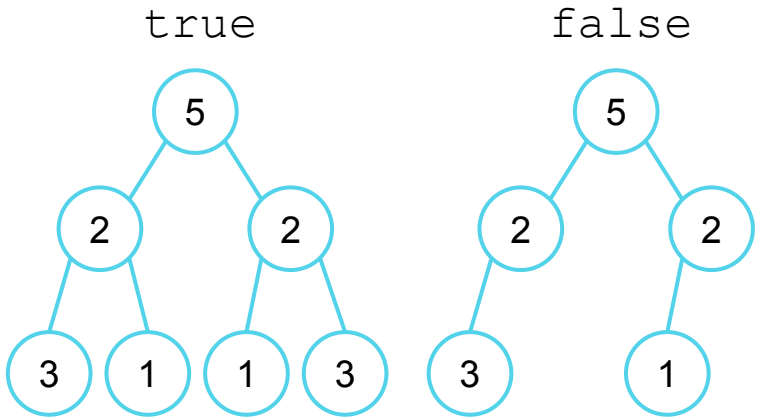
false



LeetCode 101. Symmetric Tree

- 題目敘述
  - 給一棵樹，問是否對稱，需考慮數值

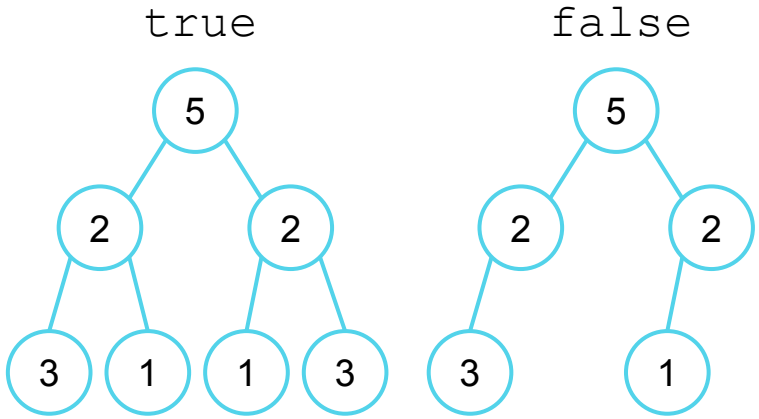
- 測資範圍
  - node 最多 1000 個點



LeetCode 101. Symmetric Tree

- 判斷一棵樹是否對稱為以下條件
  - 判斷左子樹和右子樹是否相反

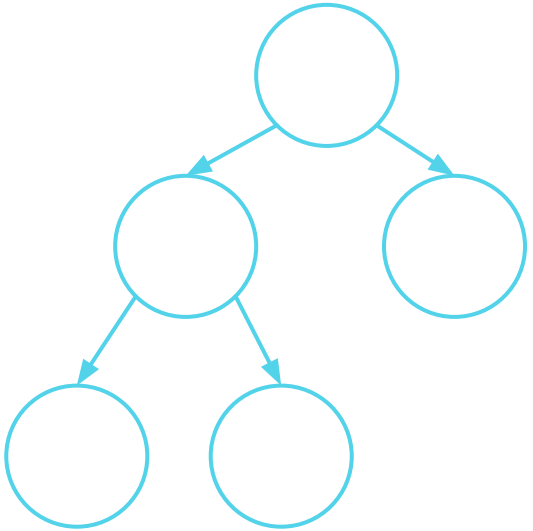
- 判斷兩樹 T1 和 T2 是否相反為以下條件
  - 根節點數值相同
  - T1 的左子樹和 T2 的右子樹相反
  - T1 的右子樹和 T2 的左子樹相反



LeetCode 104. Maximum Depth of Binary Tree

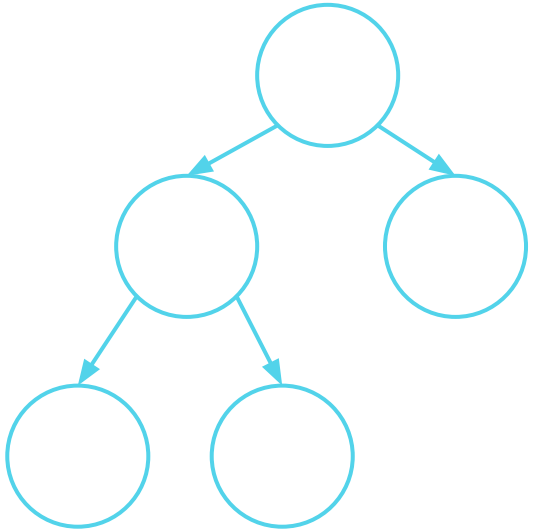
- 題目敘述
  - 給一個指標構成的tree，求最大深度

- 測資範圍
  - node 最多 10000 個



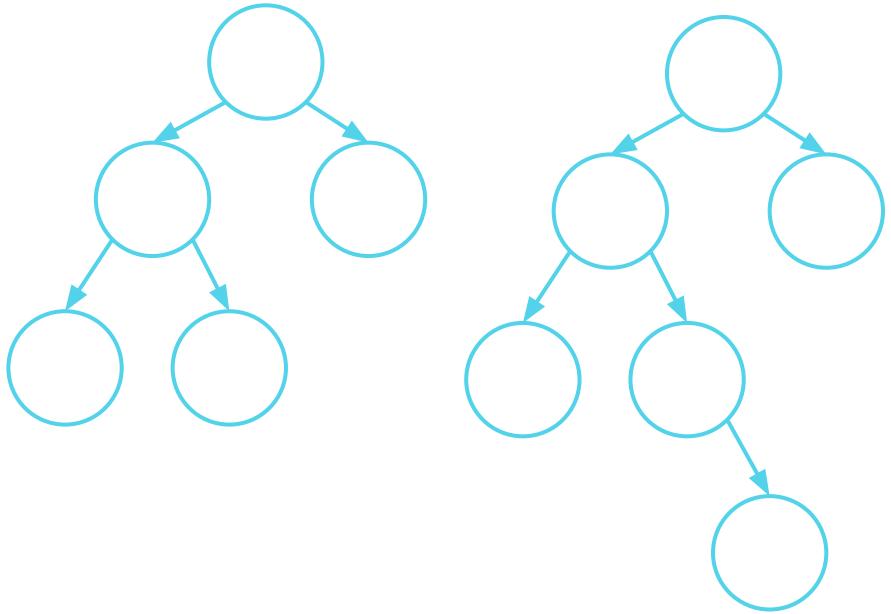
LeetCode 104. Maximum Depth of Binary Tree

- 利用 DFS 計算樹高度
  - 左子樹和右子樹取較大的高度再+1
- 回傳根節點的高度即為所求



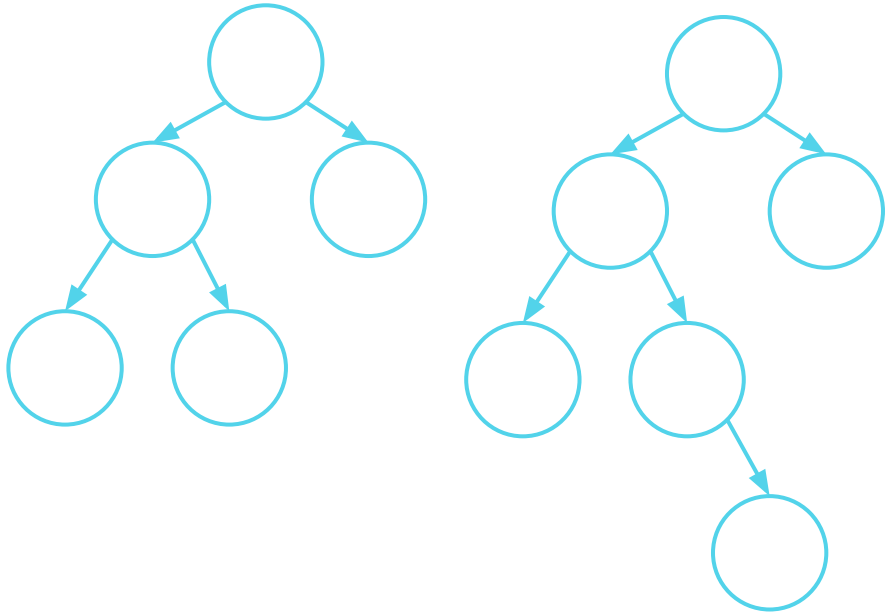
LeetCode 110. Balanced Binary Tree

- 題目敘述
  - 給一個指標構成的tree，問其是否height-balanced
  - heigh-balanced tree 定義：對於每一個 node，它的左子樹和右子樹高度差都不超過1
- 測資範圍
  - node 數量最多 5000 個



LeetCode 110. Balanced Binary Tree

- 一棵樹平衡的條件為
  - 左子樹平衡
  - 右子樹平衡
  - 左子樹深度和右子樹深度差距小於 1



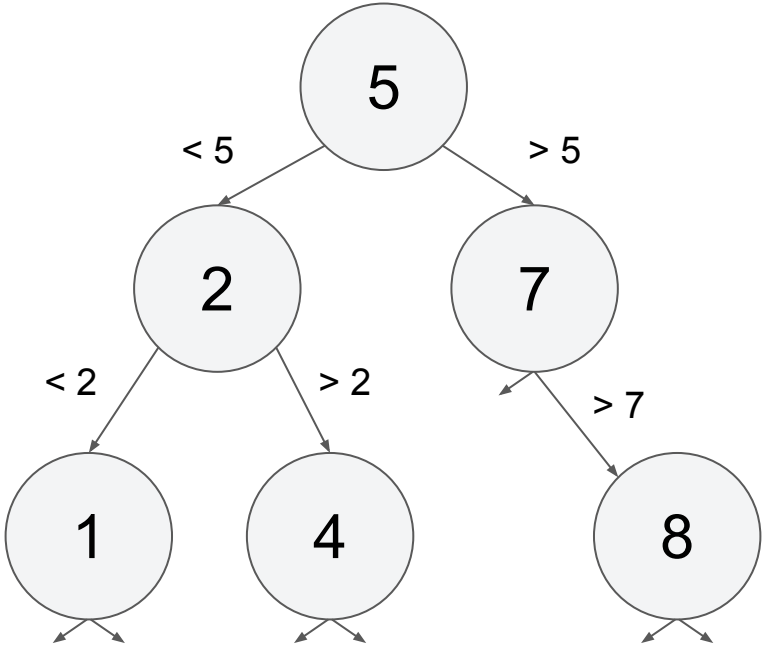
More Practices - Tree DFS

- leetcode 700. Search in a Binary Search Tree
- leetcode 104. Maximum Depth of Binary Tree
- leetcode 110. Balanced Binary Tree
- leetcode 100. Same Tree
- leetcode 101. Symmetric Tree
- leetcode 589. N-ary Tree Preorder Traversal
- leetcode 590. N-ary Tree Postorder Traversal

AlgoSeacow 

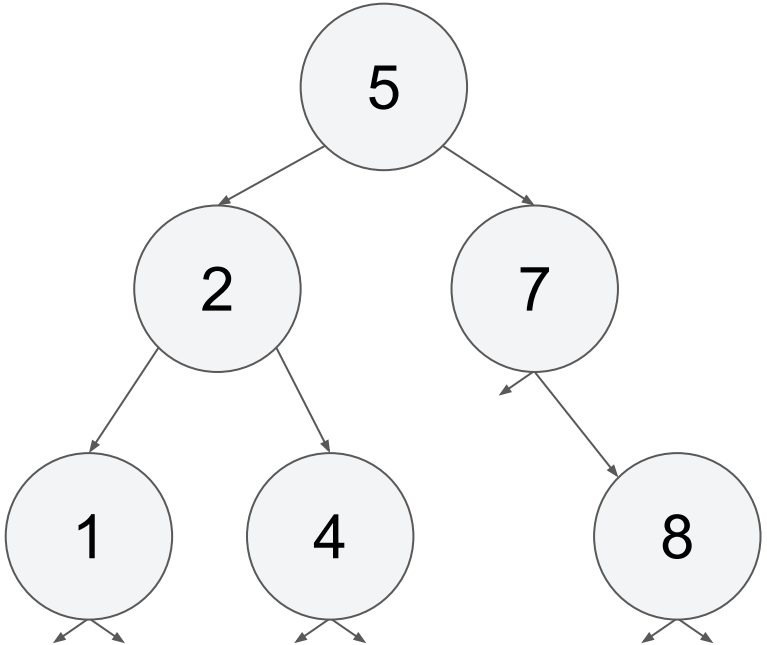
Appendix - BST

Binary Search Tree (BST)



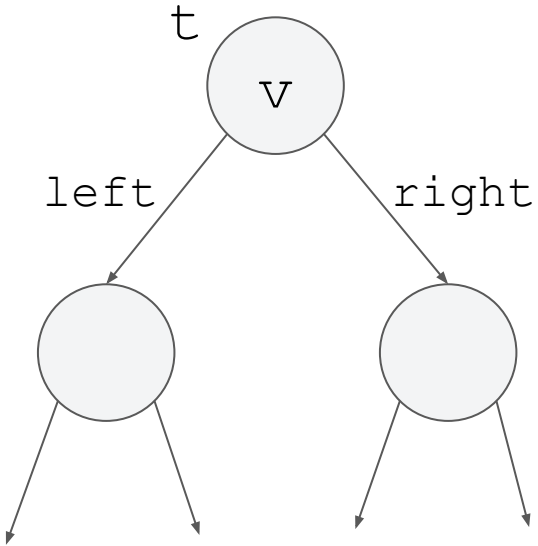
BST Operations

- Find
- Insert



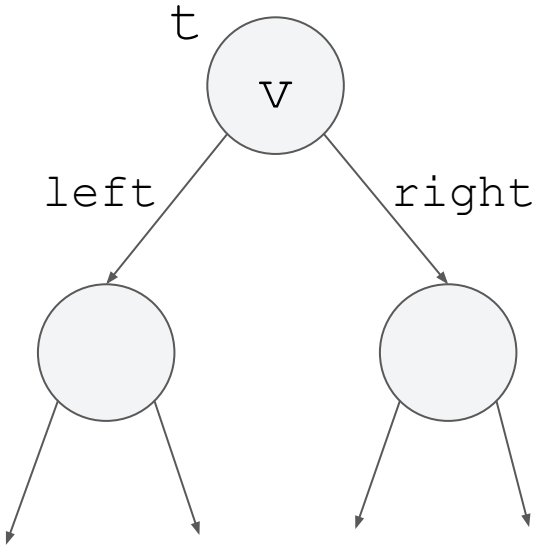
BST Find

- Find value `x` in tree `t`
  - `if t == nullptr`
    - Cannot find value `x`
  - `if x == t->v`
    - Find value `x`
  - `if x < t->v`
    - Find value `x` in tree `t->left`
  - `if x > t->v`
    - Find value `x` in tree `t->right`



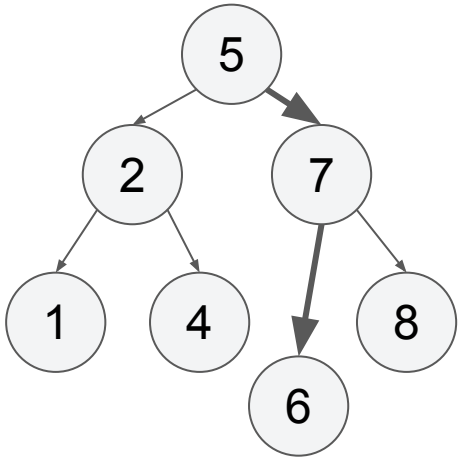
BST Insert

- Insert value `x` in tree `t`
  - `if t == nullptr`
    - Create a node
  - `if x < t->v`
    - Insert value `x` in tree `t->left`
  - `if x > t->v`
    - Insert value `x` in tree `t->right`

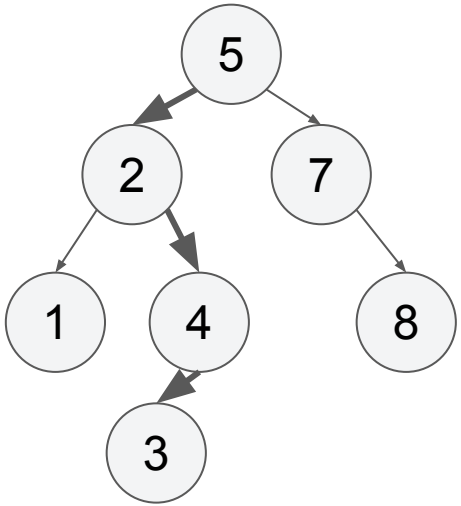


BST Insert - Example

Insert value 6



Insert value 3



Build a BST

給一個序列, 依序 Insert 構造出對應的 BST

Draw: 2 1 4 3 5

Draw: 1 2 3 4 5

BST Operations Complexity

- `Insert` and `Find` is similar
  - `T(n) = _____`
    where `n` is tree size, `h` is tree size.

- When `h ≈ n`
  - `T(n) = _____`

Worst Case

Balanced BST

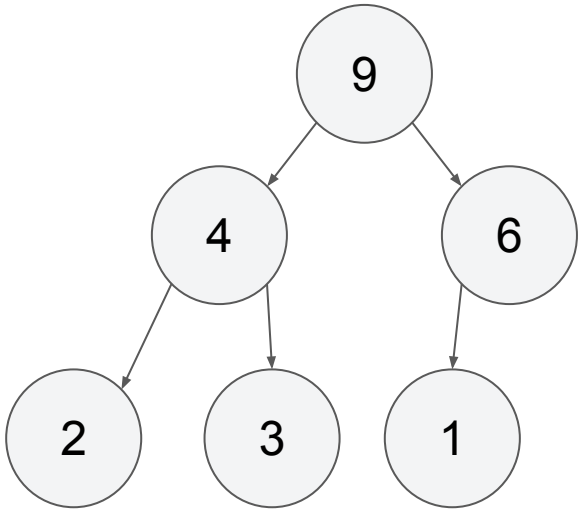
- Worst Case O(lgn)
  - AVL Tree
  - RB Tree
    - std::set, std::map
  - B+ Tree
- Average O(lgn)
  - Splay tree
- Expected O(lgn)
  - Treap

AlgoSeacow 

Appendix - Heap

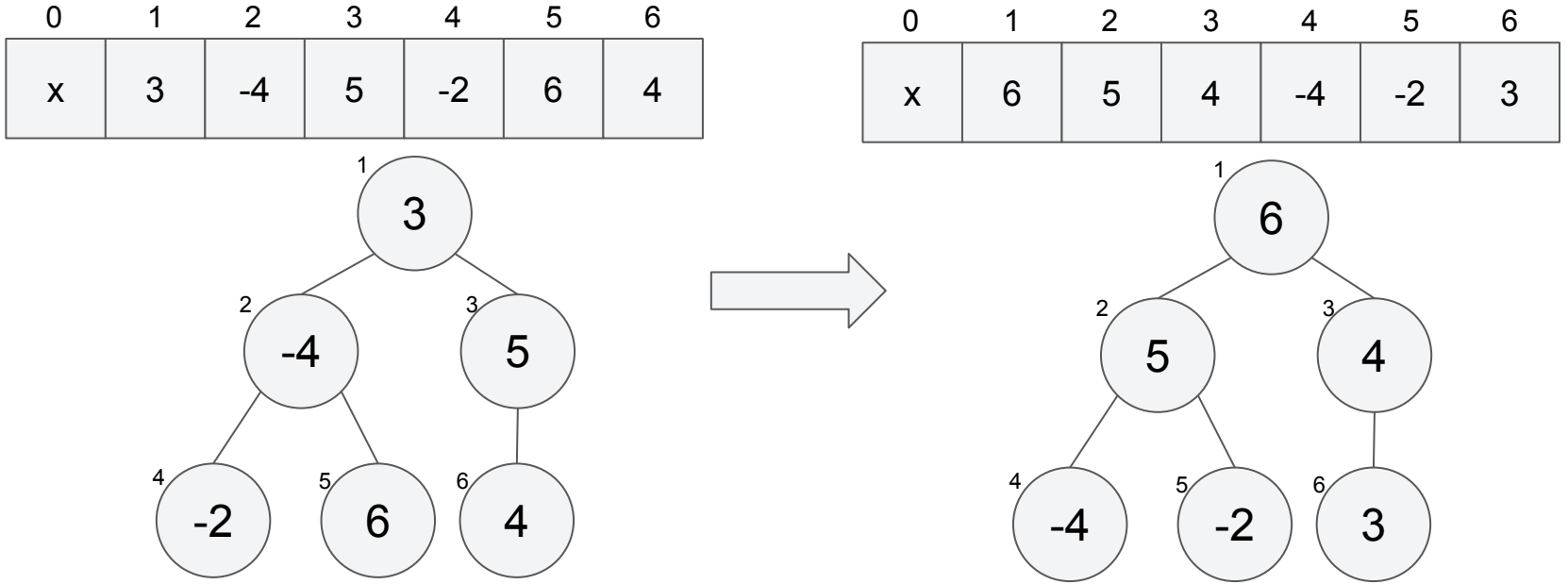
Heap

- 結構為 Complete Binary Tree
- 每個節點的數值都大於小孩數值



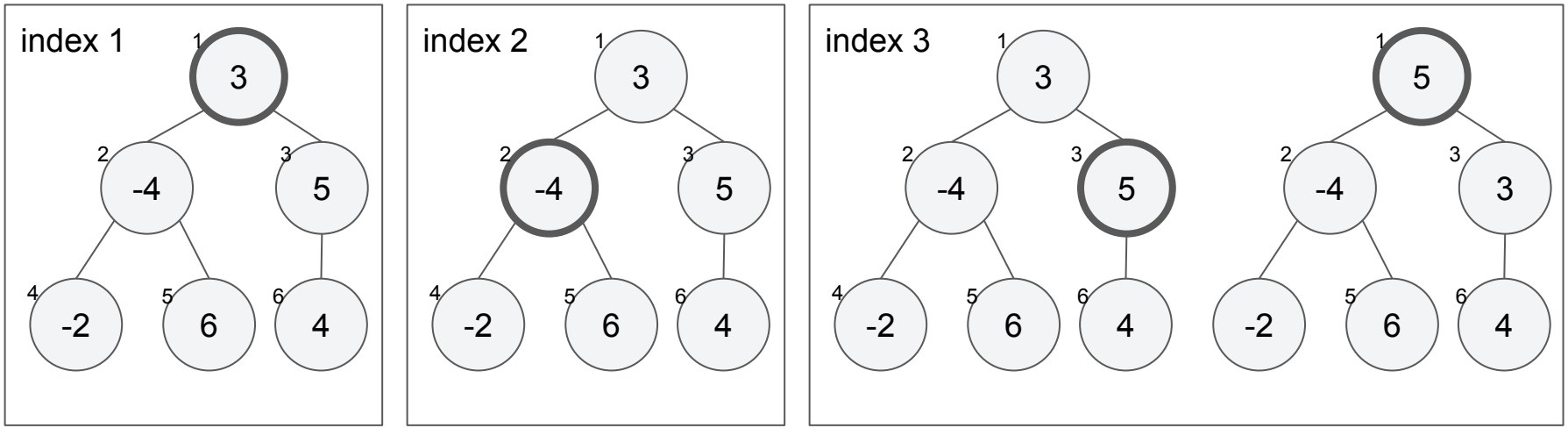
Heapify

- 給一個 Array, 使他變成一個 Heap



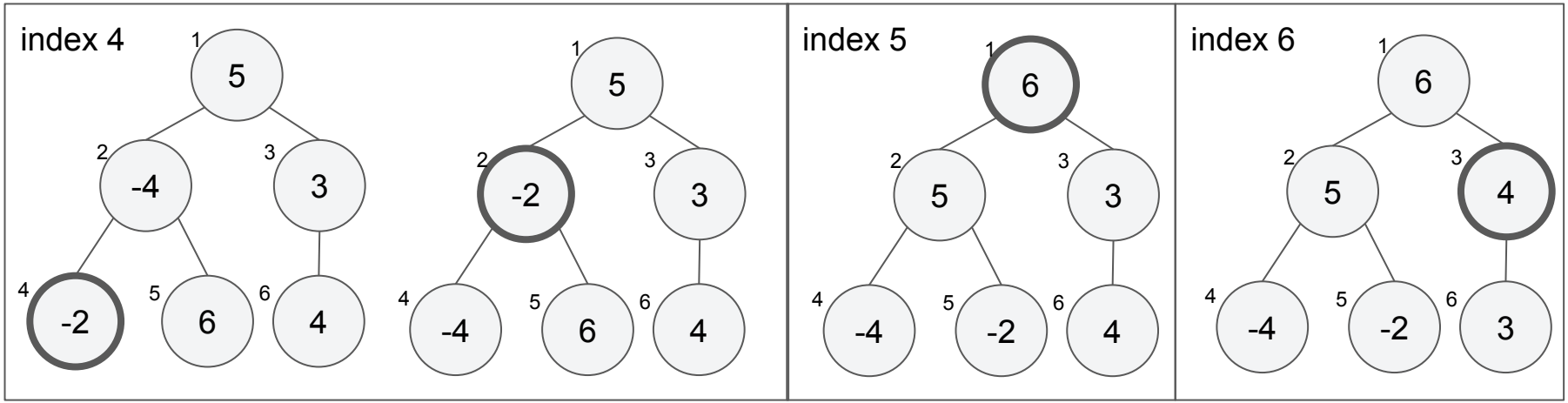
Heapify

- 從 index 小的 node 開始, 如果 parent 的值比自己小就 swap
  直到沒有 parent 或是 parent 值比自己大



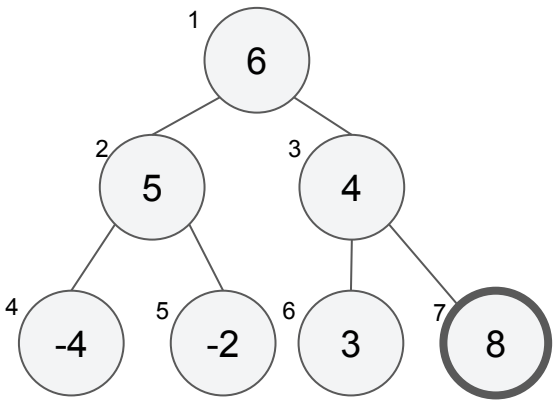
Heapify

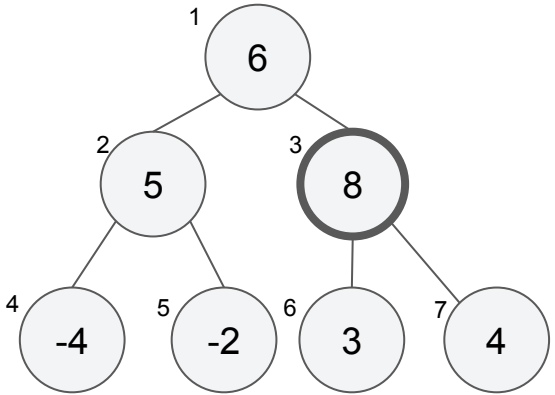
- 從 index 小的 node 開始, 如果 parent 的值比自己小就 swap
  直到沒有 parent 或是 parent 值比自己大

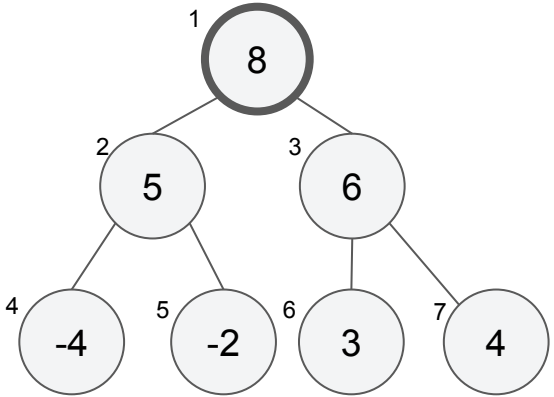


Heap Operation - push

- 按照 Complete Binary Tree 的結構接上新的節點, 並且將他置換到正確位置

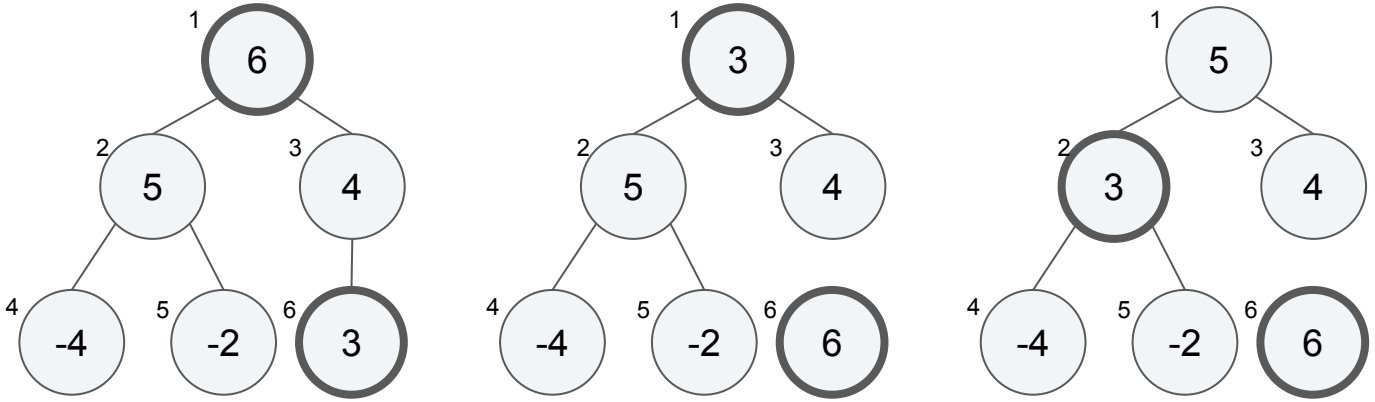






Heap Operation - pop

- 將頂端 node 和 index 最大的 node 交換並調整頂端的 node 位置
  - 和數值較大的小孩比較, 若自己比小孩小則交換, 直到沒有小孩或不能交換



Example - Heap Sort

- 給一個 Array, 使用 Heap Sort 將 Array 排序
  1. heapify
  2. pop, pop, pop, ……

Other Heap

	find max	pop max	insert	merge
Binary Heap	O(1)	O(lgn)	O(lgn)	x
Binomial Heap	O(1)	O(lgn)	O(1)	O(lgn)
Fibonacci Heap	O(1)	O(lgn)	O(1)	O(1)