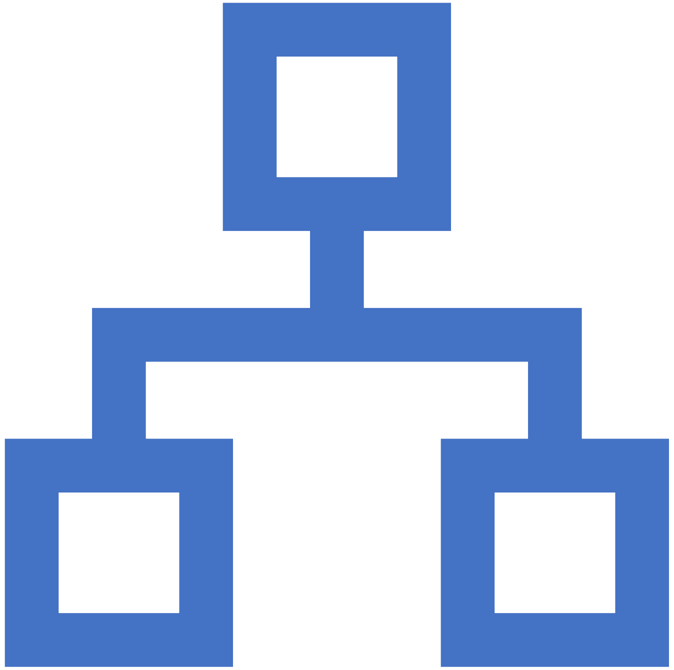


Tree

日月卦長

什麼是樹?

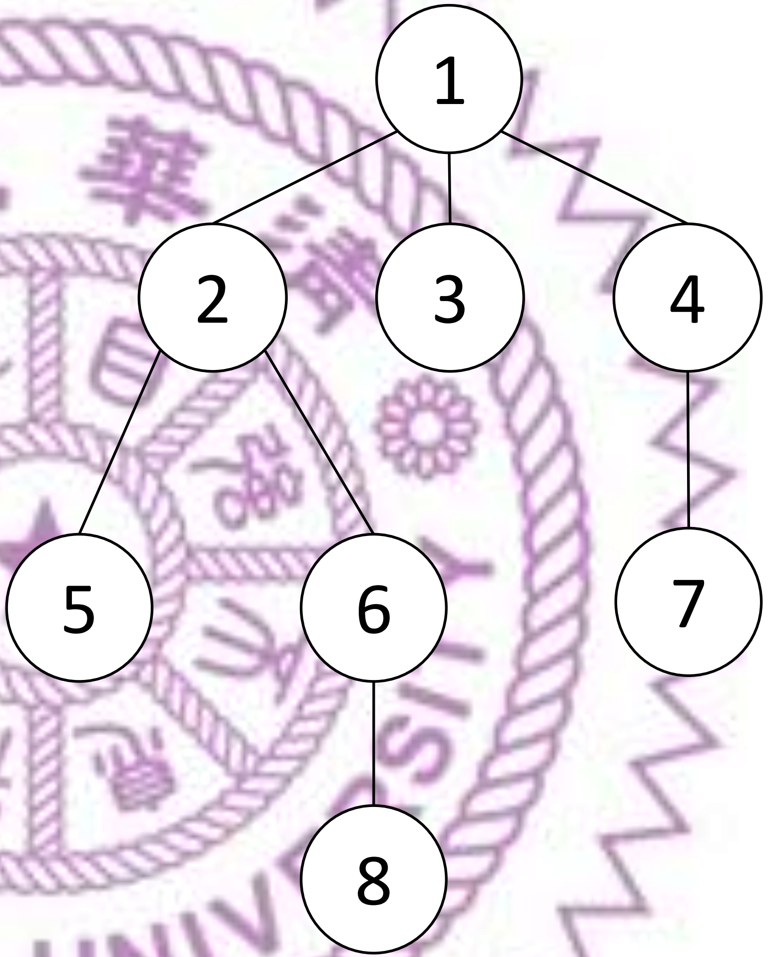
- 這是一棵樹

什麼是樹?

- 這是一棵樹
- 資訊領域中的樹

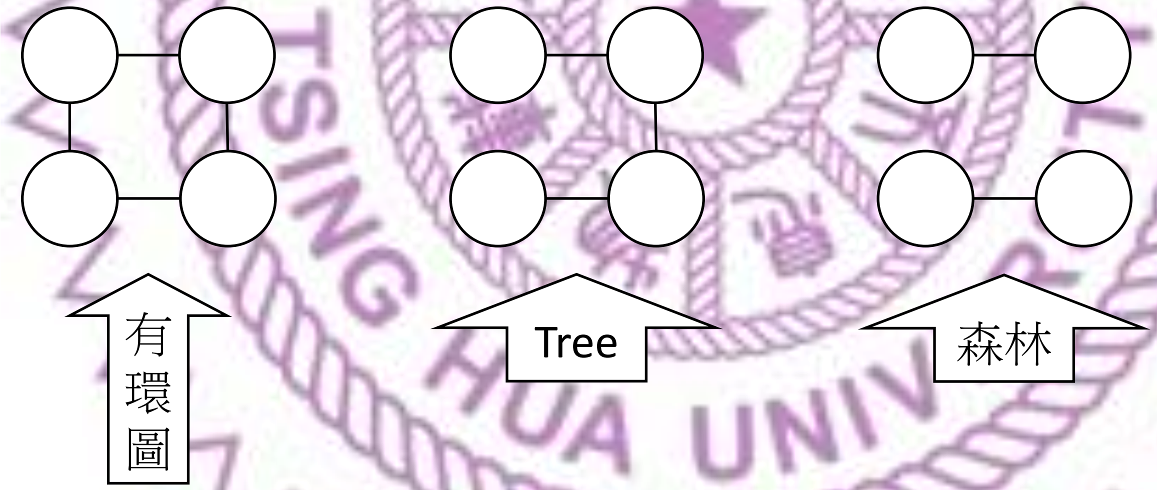
什麼是樹?

- 這是一棵樹
- 資訊領域中的樹



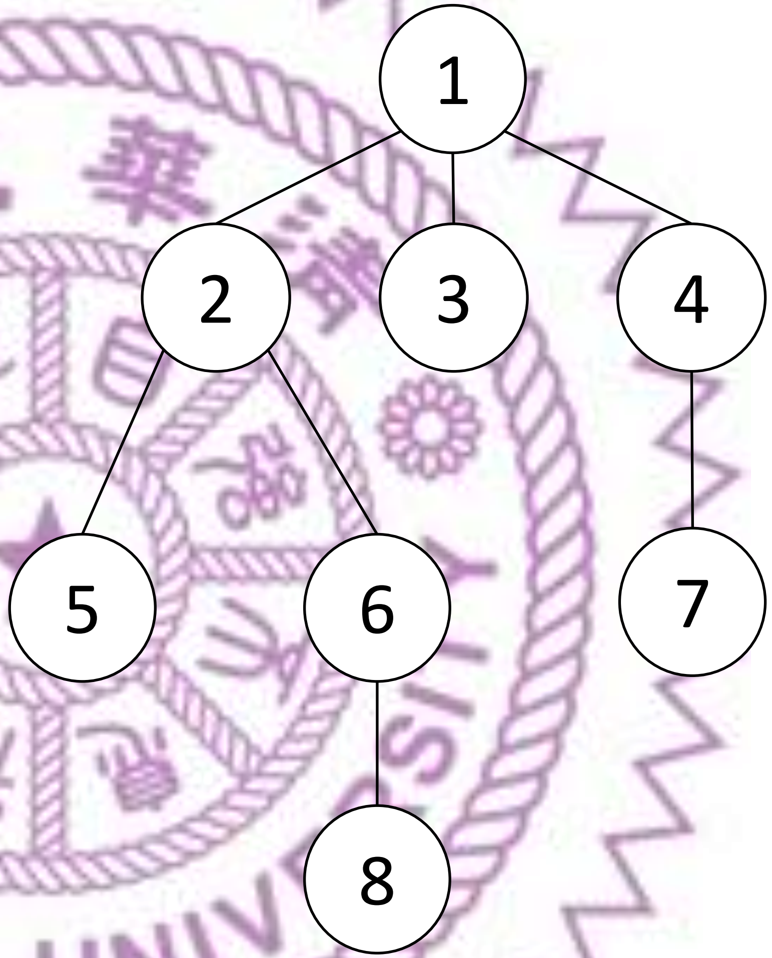
Tree 定義

- 定義：沒有環的連通圖
- 環是什麼？連通圖是什麼？



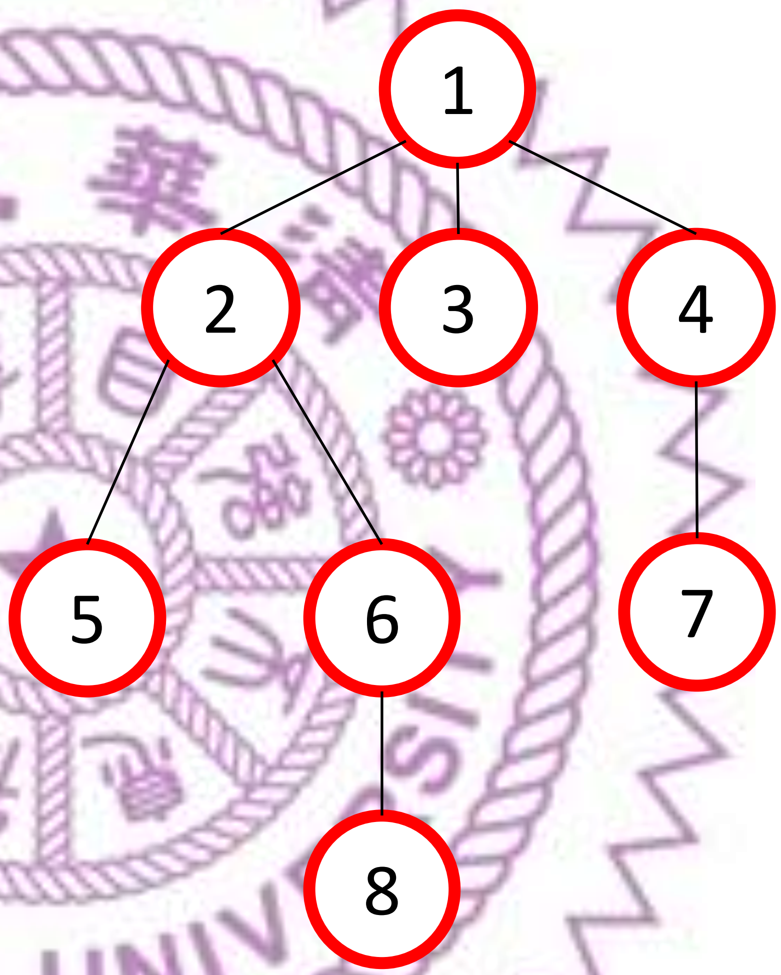
名詞介紹

- 節點(node or vertex)
- 邊(edge)
- 根節點(root)
- 葉節點(leaf)
- 父母節點(parent)
- 子節點(child)
- 祖先(ancestor)
- 子代(descendant)
- 子樹(subtree)
- 層(level)
- 深度(depth)



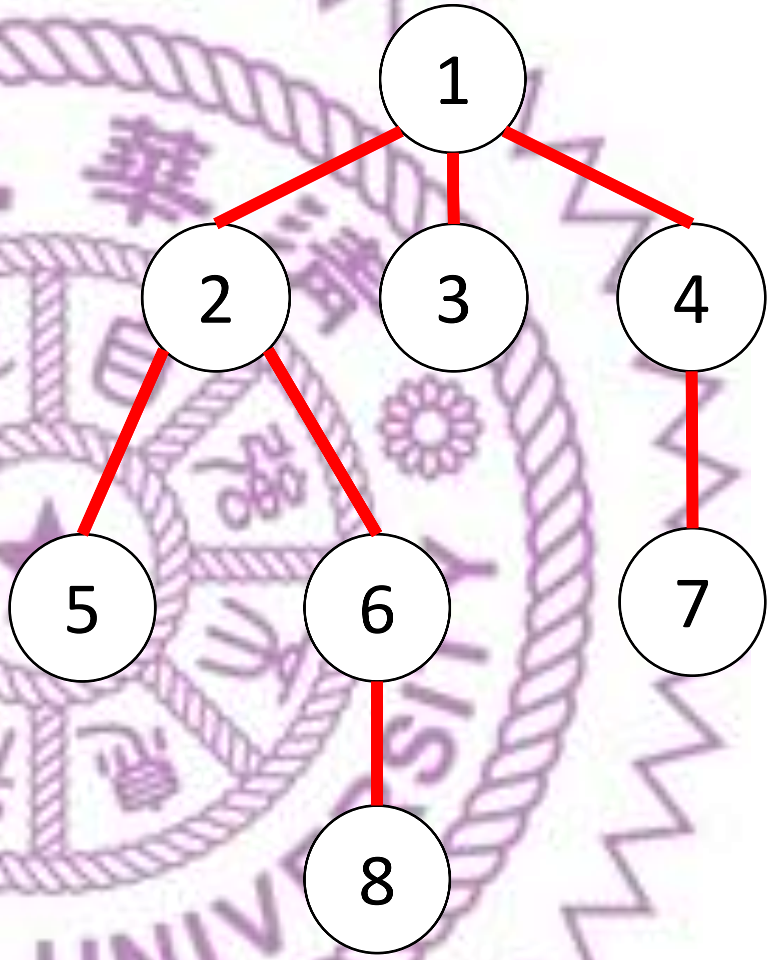
名詞介紹

- **節點(node or vertex)**
- 邊(edge)
- 根節點(root)
- 葉節點(leaf)
- 父母節點(parent)
- 子節點(child)
- 祖先(ancestor)
- 子代(descendant)
- 子樹(subtree)
- 層(level)
- 深度(depth)



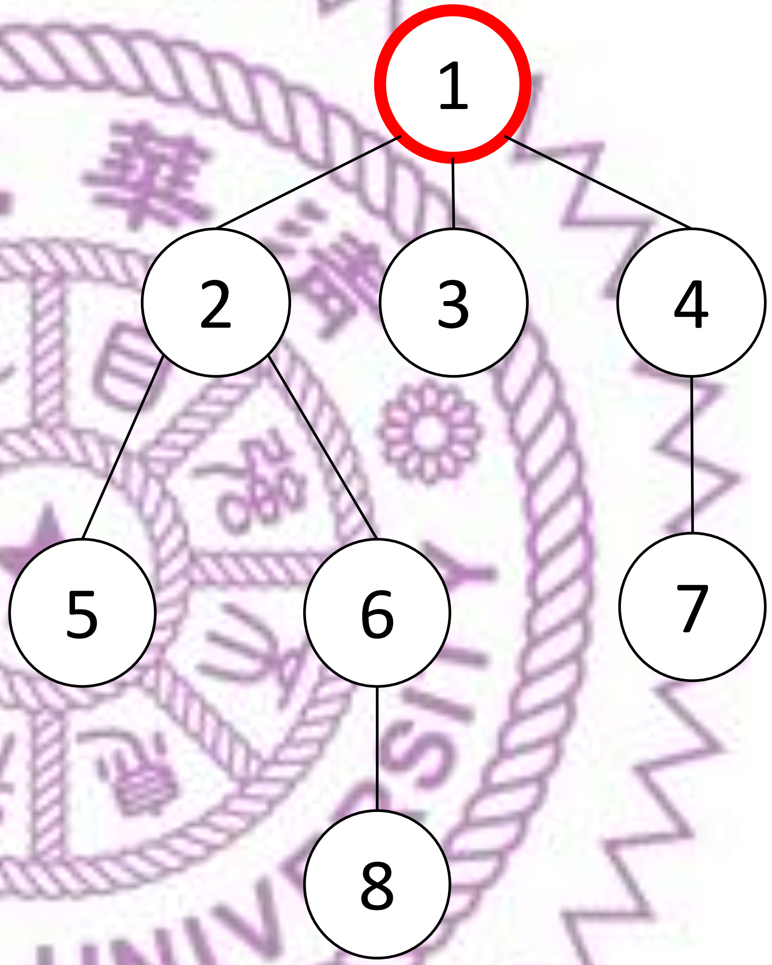
名詞介紹

- 節點(node or vertex)
- **邊(edge)**
- 根節點(root)
- 葉節點(leaf)
- 父母節點(parent)
- 子節點(child)
- 祖先(ancestor)
- 子代(descendant)
- 子樹(subtree)
- 層(level)
- 深度(depth)



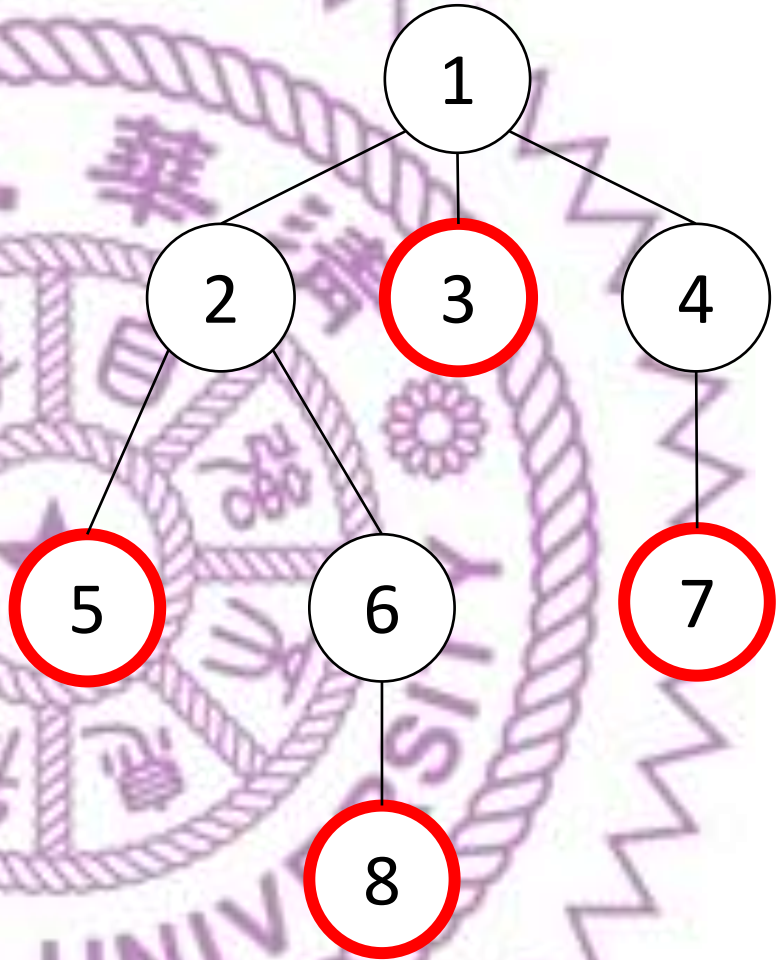
名詞介紹

- 節點(node or vertex)
- 邊(edge)
- **根節點(root)**
- 葉節點(leaf)
- 父母節點(parent)
- 子節點(child)
- 祖先(ancestor)
- 子代(descendant)
- 子樹(subtree)
- 層(level)
- 深度(depth)



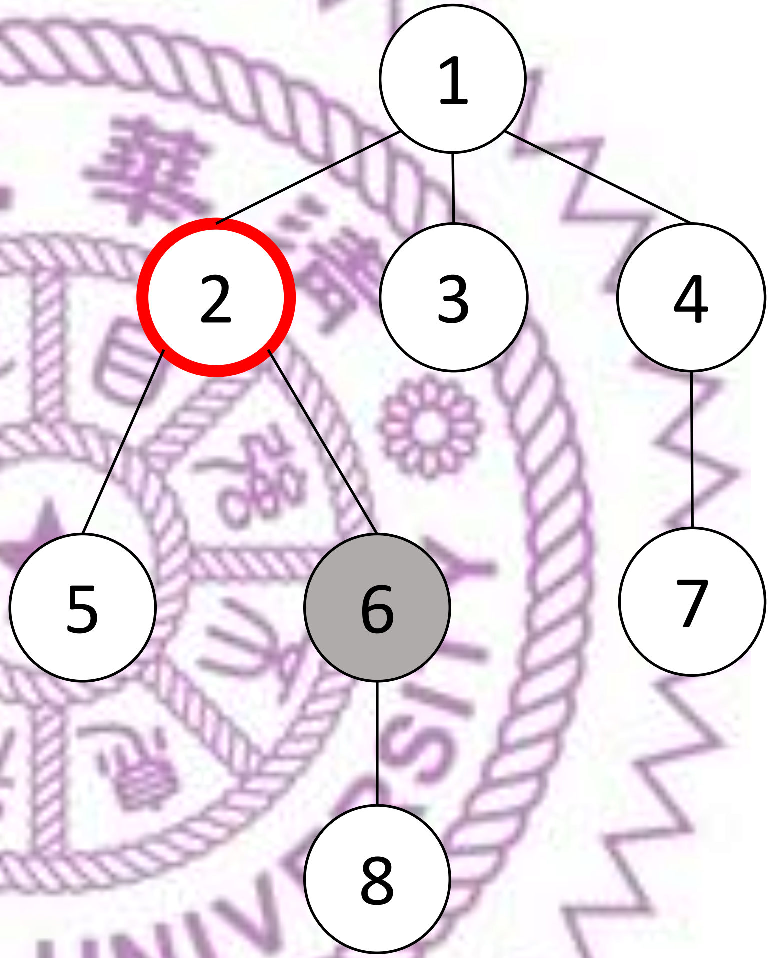
名詞介紹

- 節點(node or vertex)
- 邊(edge)
- 根節點(root)
- **葉節點(leaf)**
- 父母節點(parent)
- 子節點(child)
- 祖先(ancestor)
- 子代(descendant)
- 子樹(subtree)
- 層(level)
- 深度(depth)



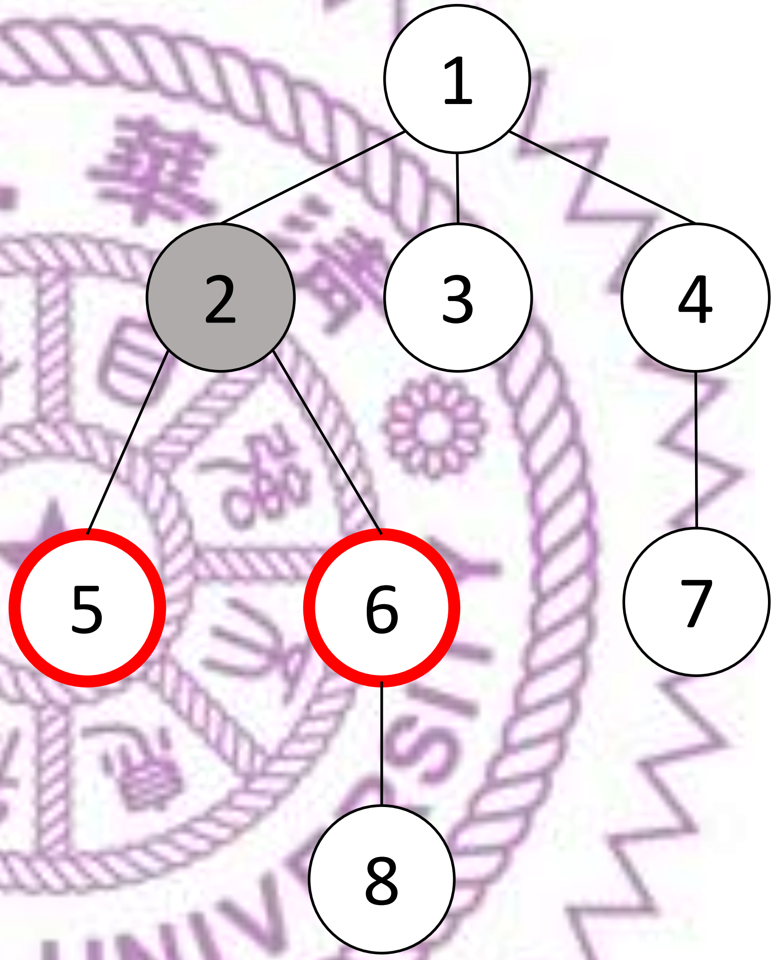
名詞介紹

- 節點(node or vertex)
- 邊(edge)
- 根節點(root)
- 葉節點(leaf)
- **父母節點(parent)**
- 子節點(child)
- 祖先(ancestor)
- 子代(descendant)
- 子樹(subtree)
- 層(level)
- 深度(depth)



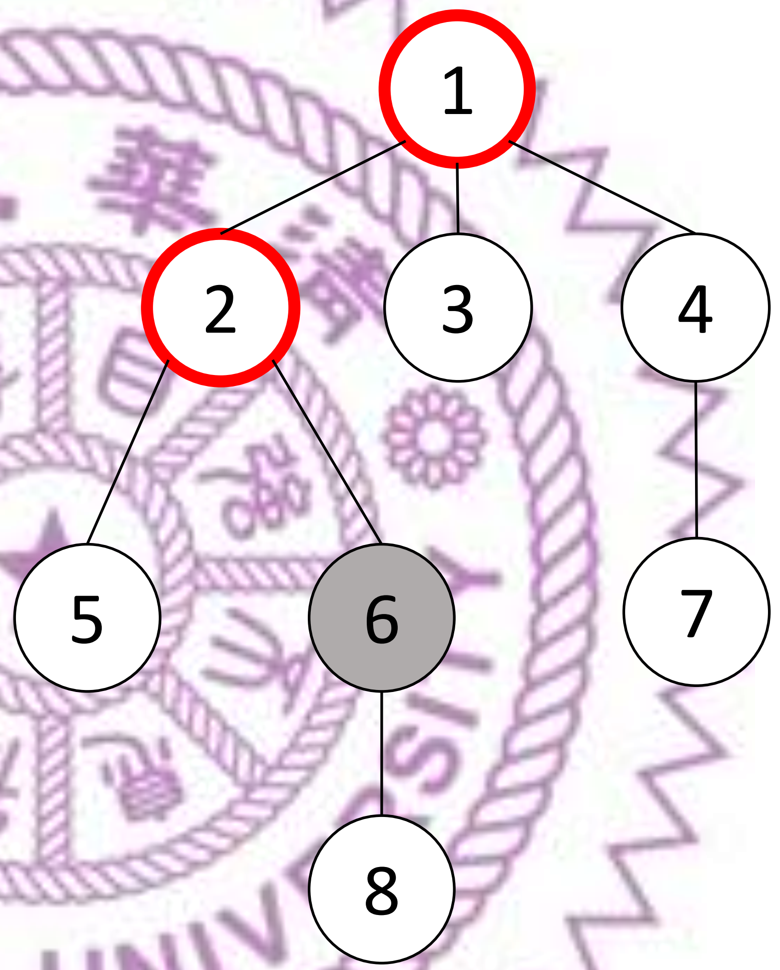
名詞介紹

- 節點(node or vertex)
- 邊(edge)
- 根節點(root)
- 葉節點(leaf)
- 父母節點(parent)
- **子節點(child)**
- 祖先(ancestor)
- 子代(descendant)
- 子樹(subtree)
- 層(level)
- 深度(depth)



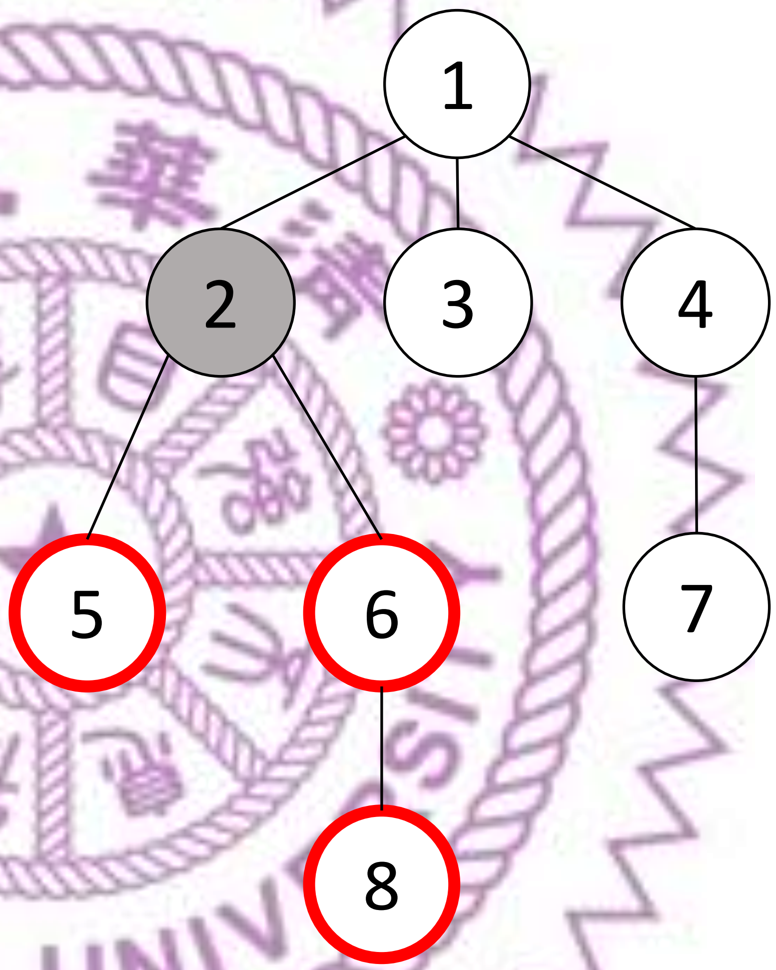
名詞介紹

- 節點(node or vertex)
- 邊(edge)
- 根節點(root)
- 葉節點(leaf)
- 父母節點(parent)
- 子節點(child)
- **祖先(ancestor)**
- 子代(descendant)
- 子樹(subtree)
- 層(level)
- 深度(depth)



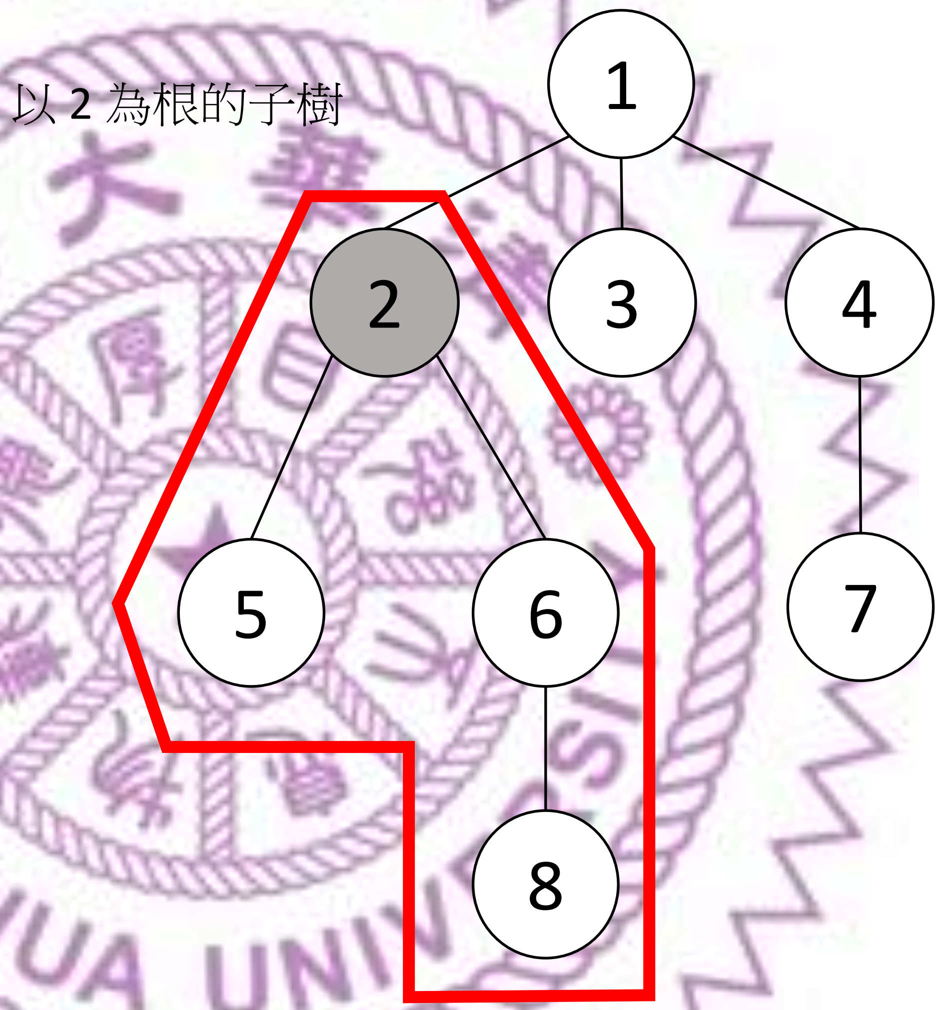
名詞介紹

- 節點(node or vertex)
- 邊(edge)
- 根節點(root)
- 葉節點(leaf)
- 父母節點(parent)
- 子節點(child)
- 祖先(ancestor)
- **子代(descendant)**
- 子樹(subtree)
- 層(level)
- 深度(depth)



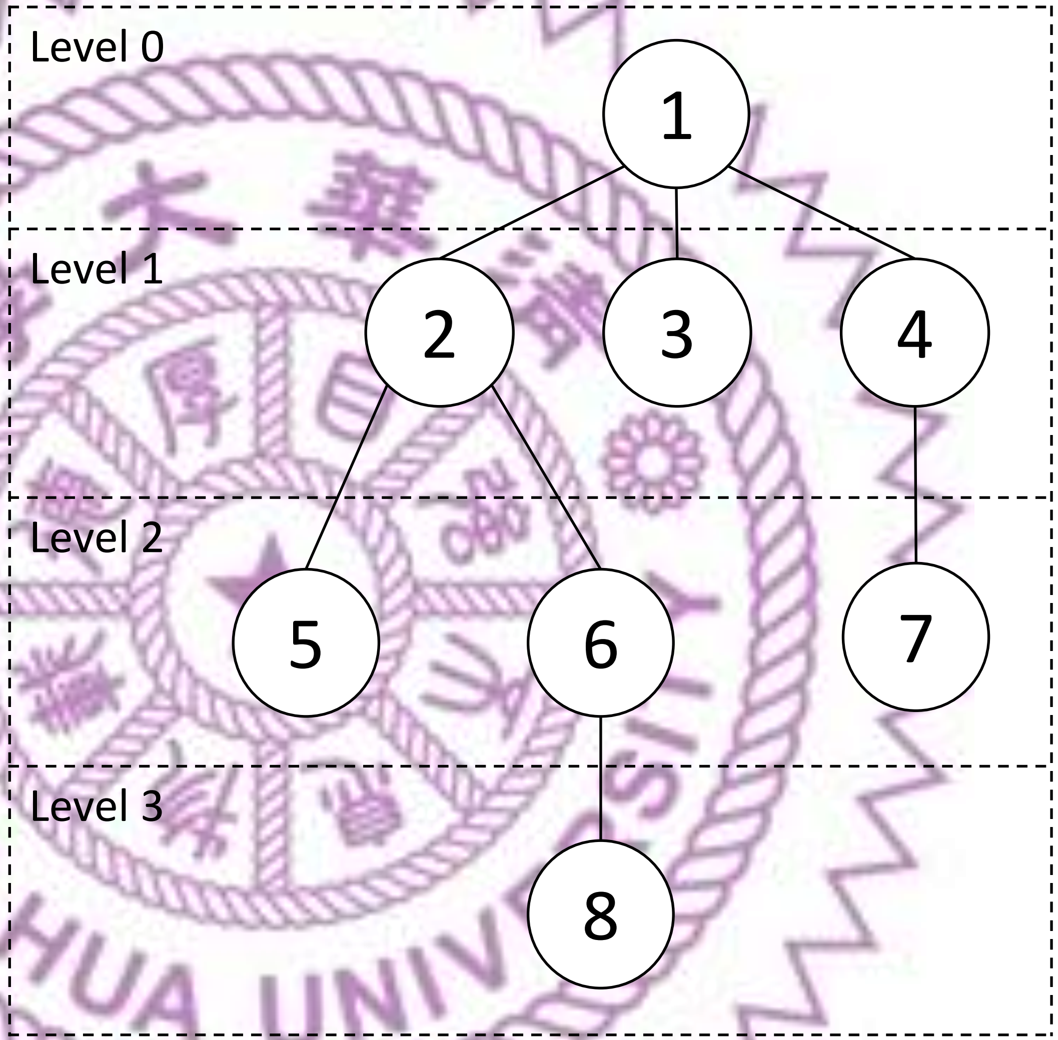
名詞介紹

- 節點(node or vertex)
- 邊(edge)
- 根節點(root)
- 葉節點(leaf)
- 父母節點(parent)
- 子節點(child)
- 祖先(ancestor)
- 子代(descendant)
- **子樹(subtree)**
- 層(level)
- 深度(depth)



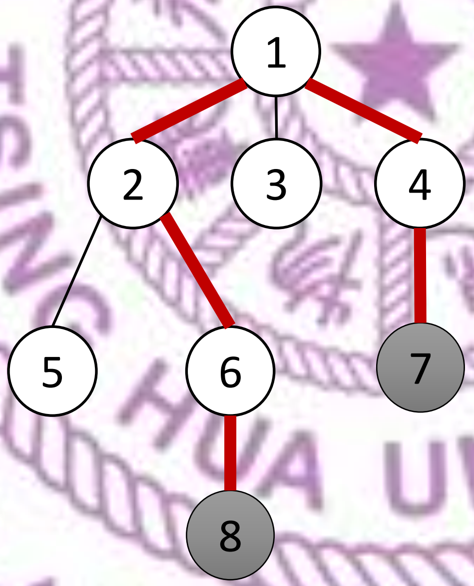
名詞介紹

- 節點(node or vertex)
- 邊(edge)
- 根節點(root)
- 葉節點(leaf)
- 父母節點(parent)
- 子節點(child)
- 祖先(ancestor)
- 子代(descendant)
- 子樹(subtree)
- **層(level)**
- **深度(depth)**



性質

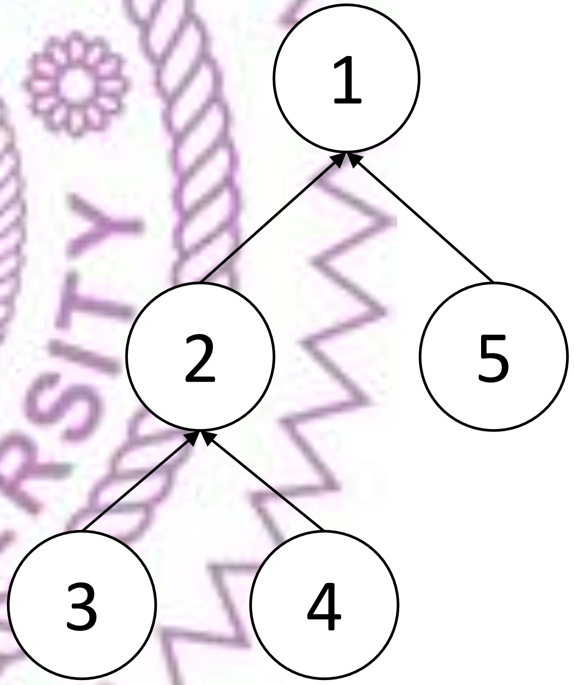
- 任兩點間洽只有一條簡單路徑 (simple path)
- 一顆有 n 個點的樹恰好有 $n-1$ 條邊



有根樹儲存方式

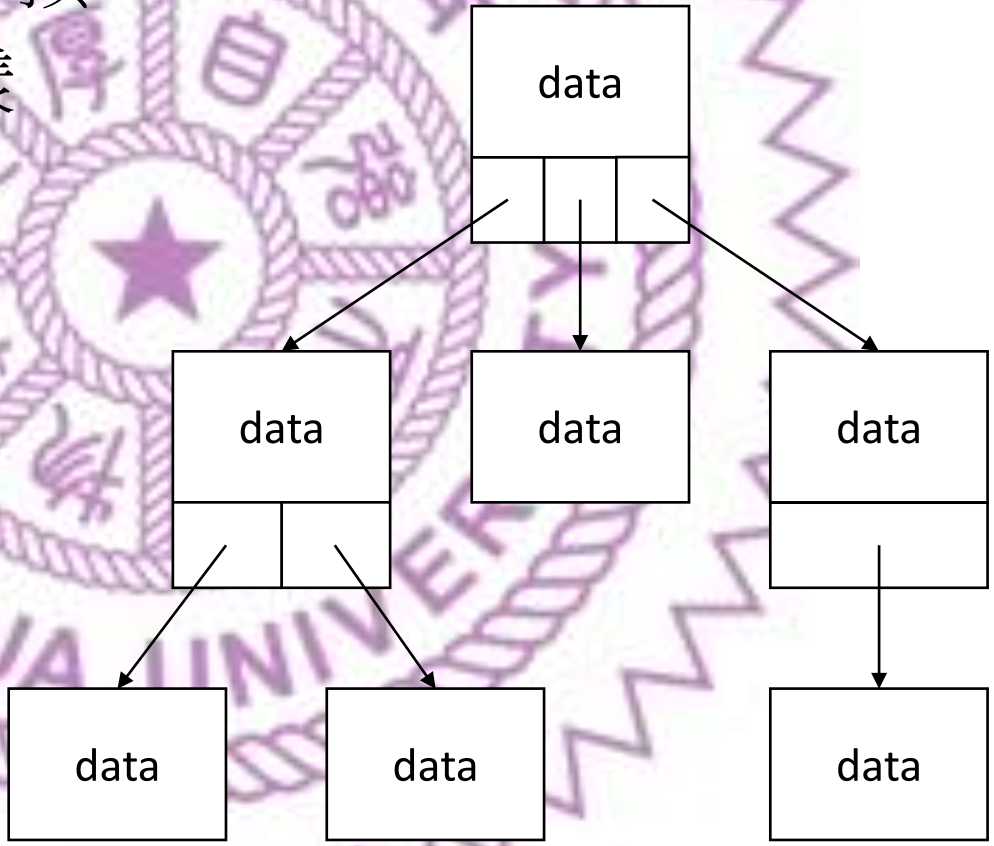
- 每個點紀錄 parent

1	2	3	4	5
-1	1	2	2	1



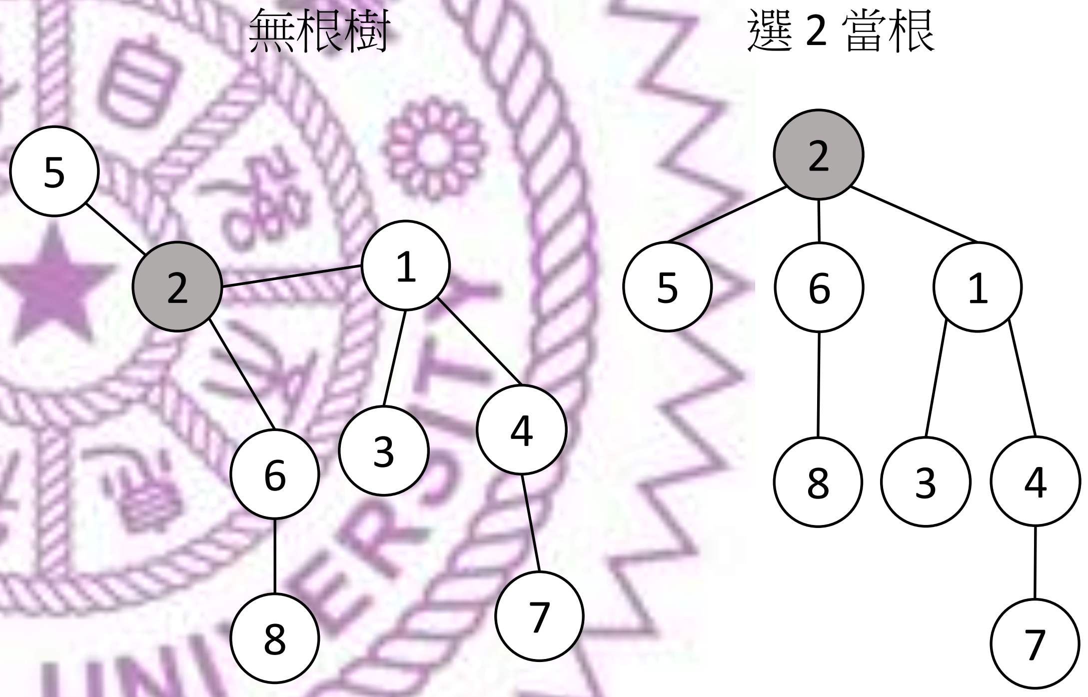
有根樹儲存方式

- 每個點只紀錄自己有哪些小孩
- 但題目輸入通常不會長這樣



無根樹

- 大多數題目的輸入形式
- 只知道兩點之間是否有邊
- 不知道誰是根
- 通常會隨便選一個點當根



無根樹的輸入
(與圖的輸入相同)

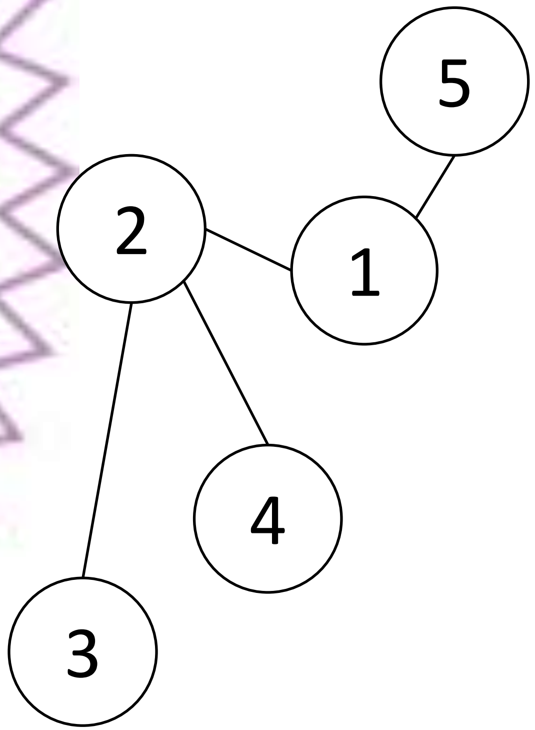
1	2	5	
2	1	3	4
3	2		
4	2		
5	1		

n 個點

```
5
1 2
2 3
2 4
1 5
```

$n-1$ 條邊

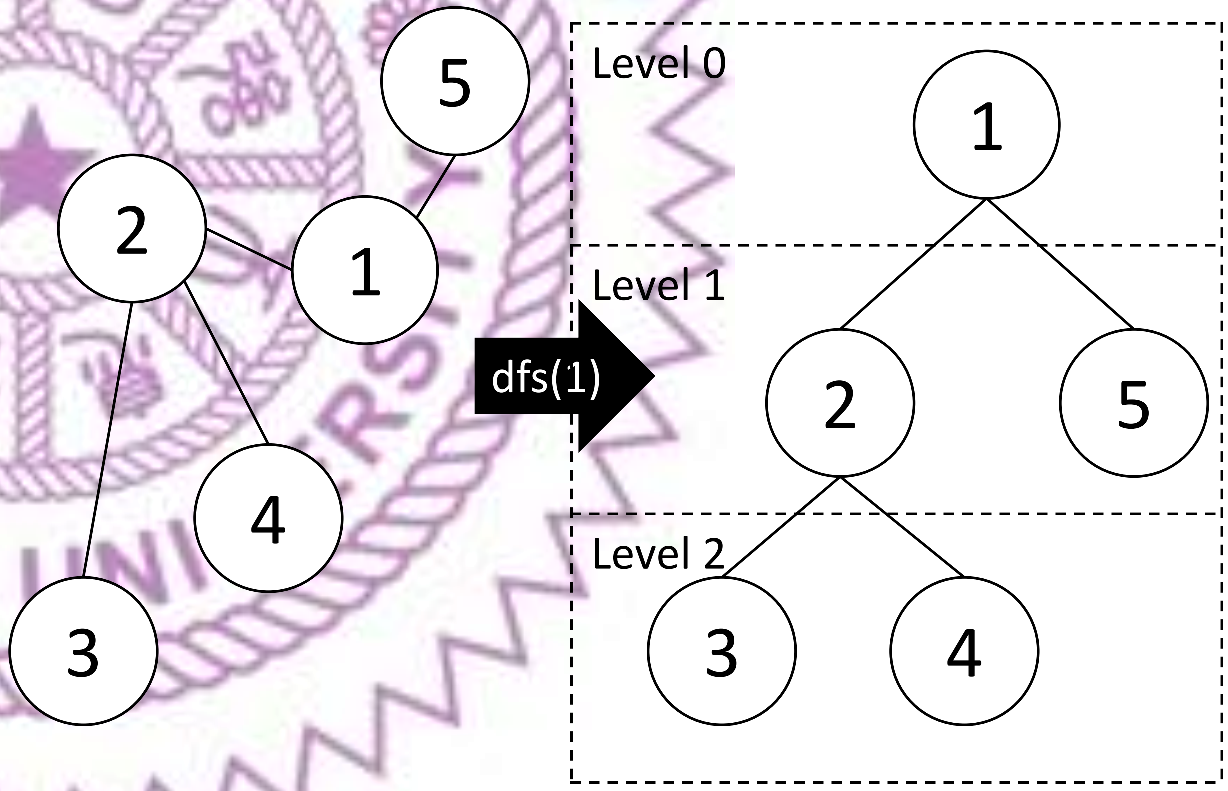
```cpp
vector<vector<int>> Tree;
int n;
cin >> n;
Tree.assign(n + 1, {});
for (int i = 0; i < n - 1; ++i) {
  int u, v;
  cin >> u >> v;
  Tree[u].emplace_back(v);
  Tree[v].emplace_back(u);
}
```



無根樹選一個點當根 ➔ 透過 dfs 走訪

- 範例：計算每個點的深度

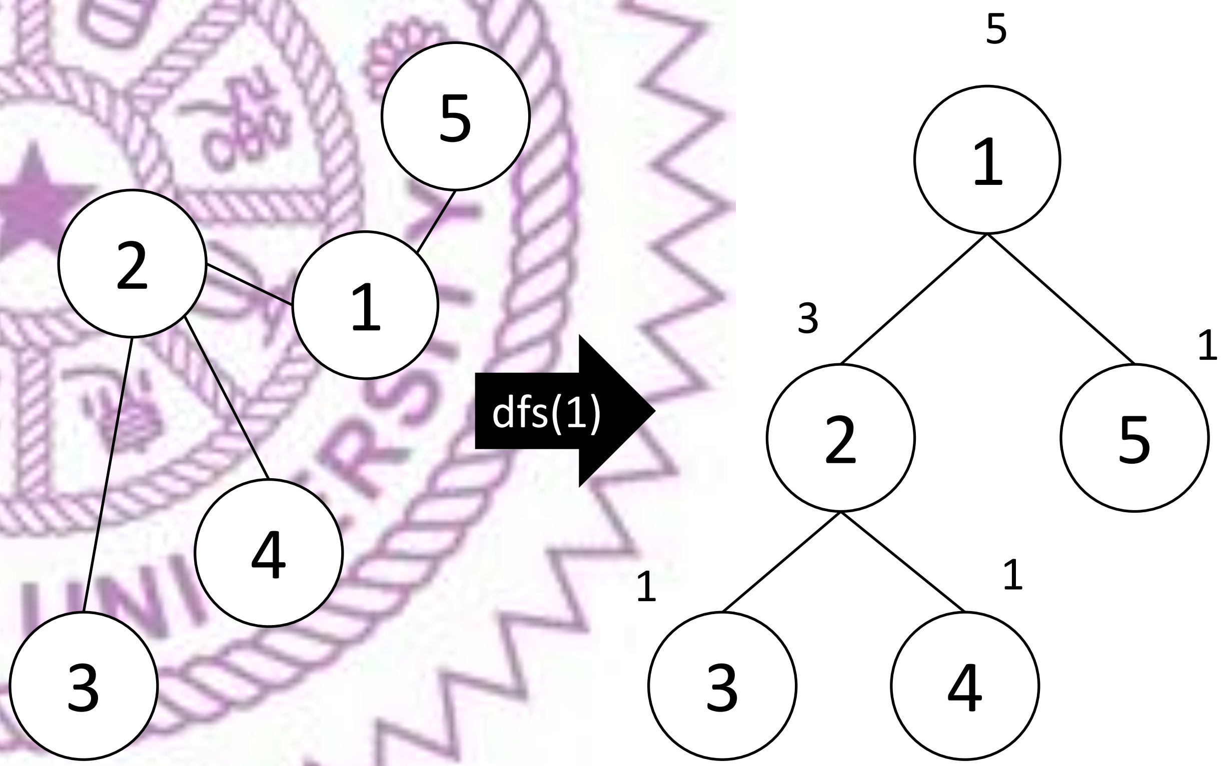
```
vector<int> level;

void dfs(int u, int parent = -1) {
  if(parent == -1) level[u] = 0;
  else level[u] = level[parent] + 1;
  for (int v : Tree[u]) {
    if (v == parent) continue;
    dfs(v, u);
  }
}
```



無根樹選一個點當根 ➔ 透過 dfs 走訪

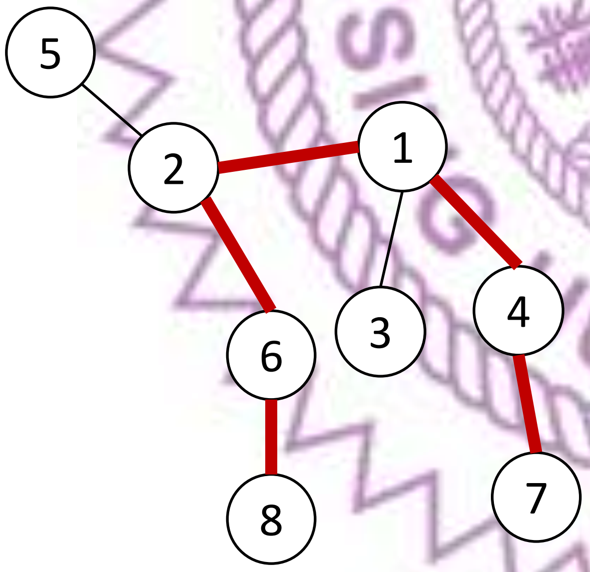
- 範例：計算每個點的子樹大小

```
vector<int> size;

int dfs(int u, int parent = -1) {
  size[u] = 1;
  for (int v : Tree[u]) {
    if (v == parent) continue;
    size[u] += dfs(v, u);
  }
  return size[u];
}
```



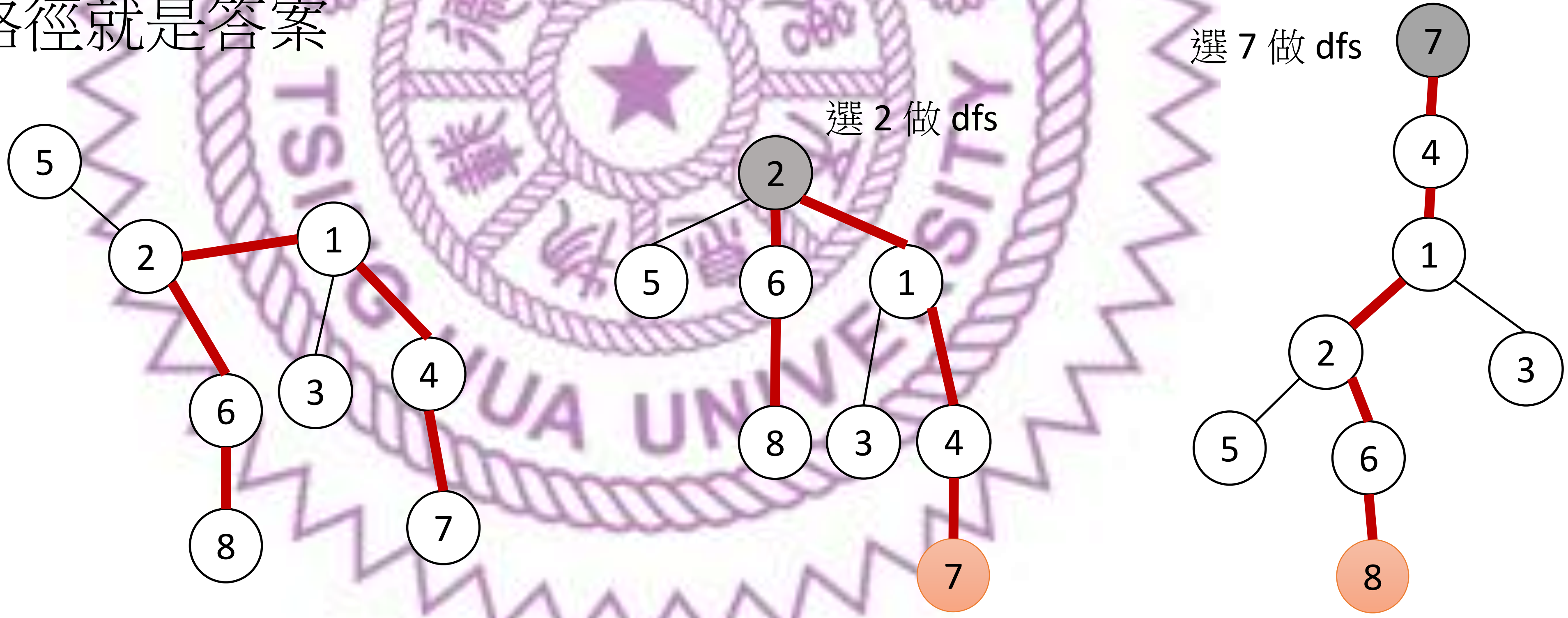
樹直徑

- 樹上距離最遠的兩點之間的路徑



樹直徑

- 隨便選個點 x 做 dfs，找距離 x 最遠的 a
- 對 a 做 dfs，找距離 a 最遠的 b
- $a \rightarrow b$ 路徑就是答案



樹直徑

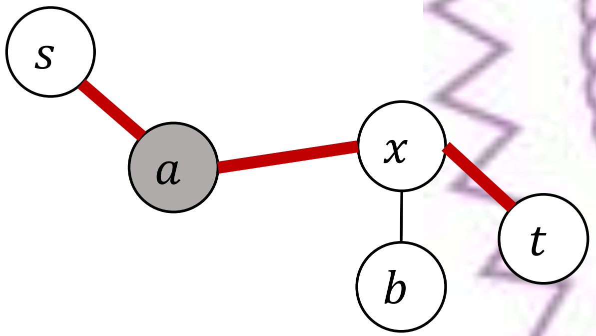
```cpp
vector<int> level;

void dfs(int u, int parent = -1) {
  if(parent == -1) level[u] = 0;
  else level[u] = level[parent] + 1;
  for (int v : Tree[u]) {
    if (v == parent) continue;
    dfs(v, u);
  }
}
```

```cpp
dfs(1); // 隨便選一個點
int a = max_element(level.begin(), level.end()) - level.begin();
dfs(a); // a 必然是直徑的其中一個端點
int b = max_element(level.begin(), level.end()) - level.begin();
cout << level[b] << endl;
```

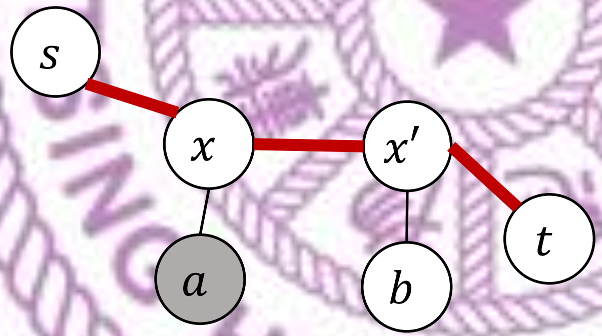
證明：反證法

對於任意點 a，呼叫 $dfs(a)$ ，能走到的最遠點必然是直徑的某個端點

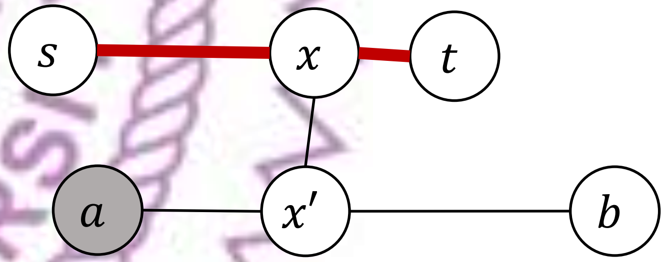
假設 $s \to t$ 路徑是直徑，距離 a 最遠的點是 b



$$|a \to b| > |a \to t|$$
$$|x \to b| > |x \to t|$$
$$|s \to b| > |s \to t|$$



$$|a \to b| > |a \to t|$$
$$|x \to b| > |x \to t|$$
$$|s \to b| > |s \to t|$$

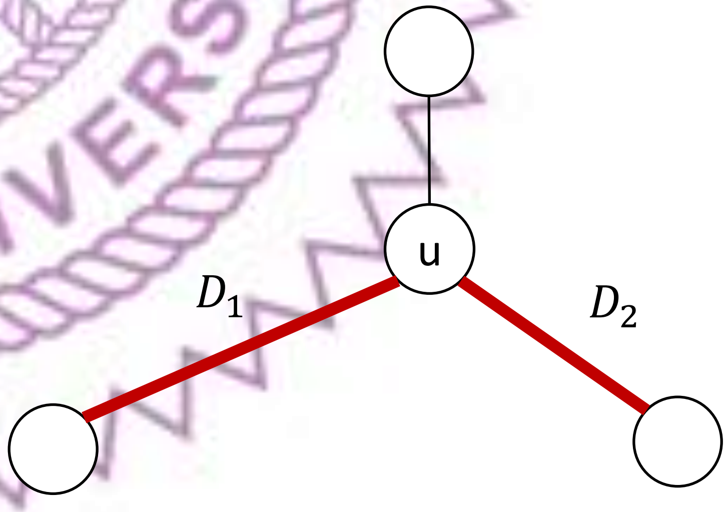


$$|a \to b| > |a \to t|$$
$$|x' \to b| > |x' \to t|$$
$$|x \to b| > |x \to t|$$
$$|s \to b| > |s \to t|$$

樹直徑：另一種方法

```
vector<int> D1, D2; // 最遠、次遠距離
int ans = 0; // 直徑長度

void dfs(int u, int parent = -1) {
  D1[u] = D2[u] = 0;
  for (int v : Tree[u]) {
    if (v == parent) continue;
    dfs(v, u);
    int dis = D1[v] + 1;
    if (dis > D1[u]) {
      D2[u] = D1[u];
      D1[u] = dis;
    } else
      D2[u] = max(D2[u], dis);
  }
  ans = max(ans, D1[u] + D2[u]);
}
```

- 隨便選一個點當 root
- 每個點紀錄與後代的
  - 最遠距離 D_1
  - 次遠距離 D_2
- 直徑就會是所有 $D_1 + D_2$ 最長的那個



樹重心

- 若以某個點為根，會使得最大的子樹節點數量最小 $\left(\le \frac{n}{2}\right)$ 稱之為樹種新
- 樹重心最多只有兩個

找出其中一個樹重心

```
vector<int> size;

int ans = -1;
void dfs(int u, int parent = -1) {
  size[u] = 1;
  int max_son_size = 0;
  for (auto v : Tree[u]) {
    if (v == parent) continue;
    dfs(v, u);
    size[u] += size[v];
    max_son_size = max(max_son_size, size[v]);
  }
  max_son_size = max(max_son_size, n - size[u]);
  if (max_son_size <= n / 2) ans = u;
}
```

注意計算連向 parent 的子樹

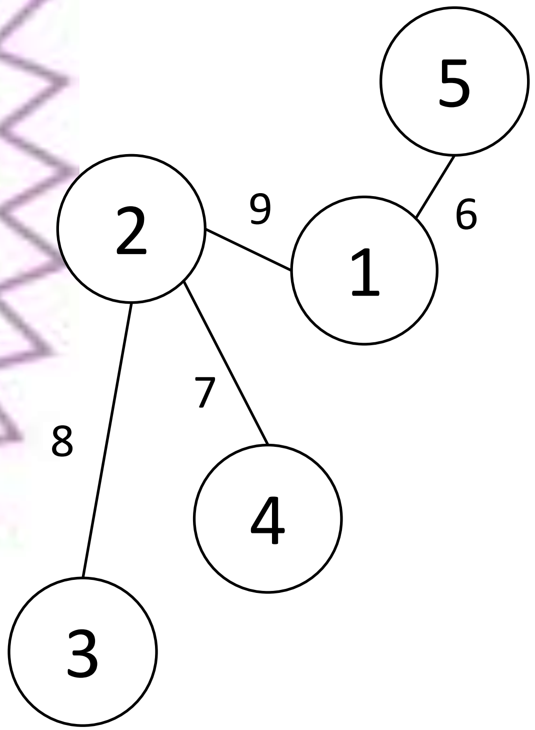
如果邊有權重

n 個點

$n-1$ 條邊

```
5
1 2 9
2 3 8
2 4 7
1 5 6
```

```cpp
vector<vector<pair<int, int>>> Tree;
int n;
cin >> n;
Tree.assign(n + 1, {});
for (int i = 0; i < n - 1; ++i) {
  int u, v, cost;
  cin >> u >> v >> cost;
  Tree[u].emplace_back(v, cost);
  Tree[v].emplace_back(u, cost);
}
```



二元樹

Binary Tree

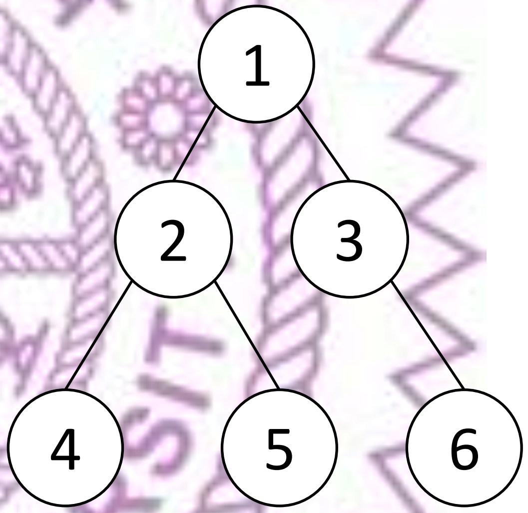
Binary Tree

- 中文：二元樹
- 每個節點最多只會有兩個子節點
- 第 k 層最多有 2^k 個節點
- 深度為 k 的二元樹最多有 $2^{k+1}-1$ 個節點

```
struct node {
  int data;
  node *lc, *rc;

  node(int data = 0) : data(data), lc(nullptr), rc(nullptr) {}
};
```

前序遍歷 preorder

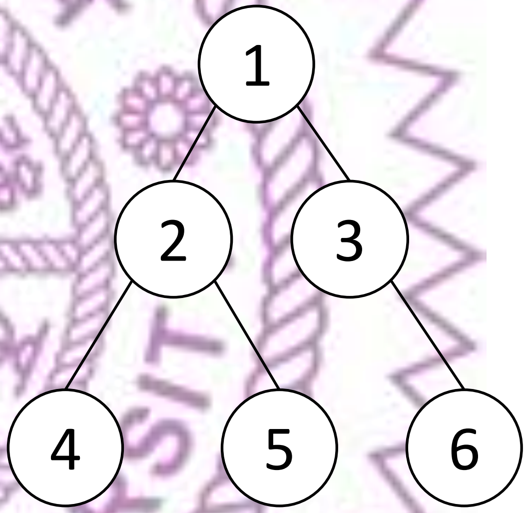
```
void dfs(node *nd) {
  if (nd == nullptr) return;
  cout << nd->data << ' ';
  dfs(nd->lc);
  dfs(nd->rc);
}
```



1 2 4 5 3 6

中序遍歷 inorder

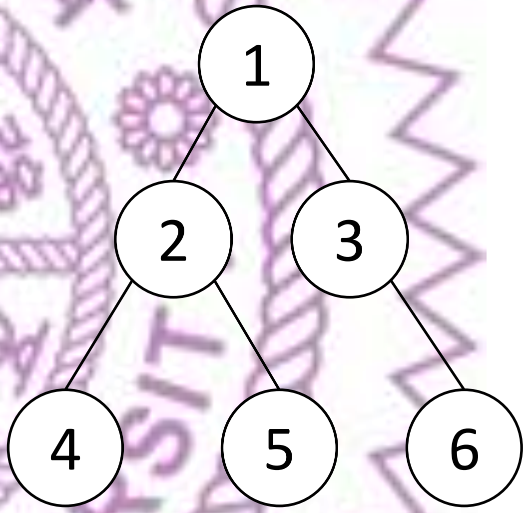
```
void dfs(node *nd) {
  if (nd == nullptr) return;
  dfs(nd->lc);
  cout << nd->data << ' ';
  dfs(nd->rc);
}
```



4 2 5 1 3 6

後序遍歷 postorder

```
void dfs(node *nd) {
  if (nd == nullptr) return;
  dfs(nd->lc);
  dfs(nd->rc);
  cout << nd->data << ' ';
}
```



4 5 2 6 3 1

構造二元樹定理

前序： A $\underbrace{B\ D\ E\ H}_{L}$ $\underbrace{C\ F\ G\ I}_{R}$

中序： $\underbrace{D\ B\ H\ E}_{L}$ A $\underbrace{F\ C\ G\ I}_{R}$

給定
1. 前序 or 後序
2. 中序

保證能構造出唯一的二元樹

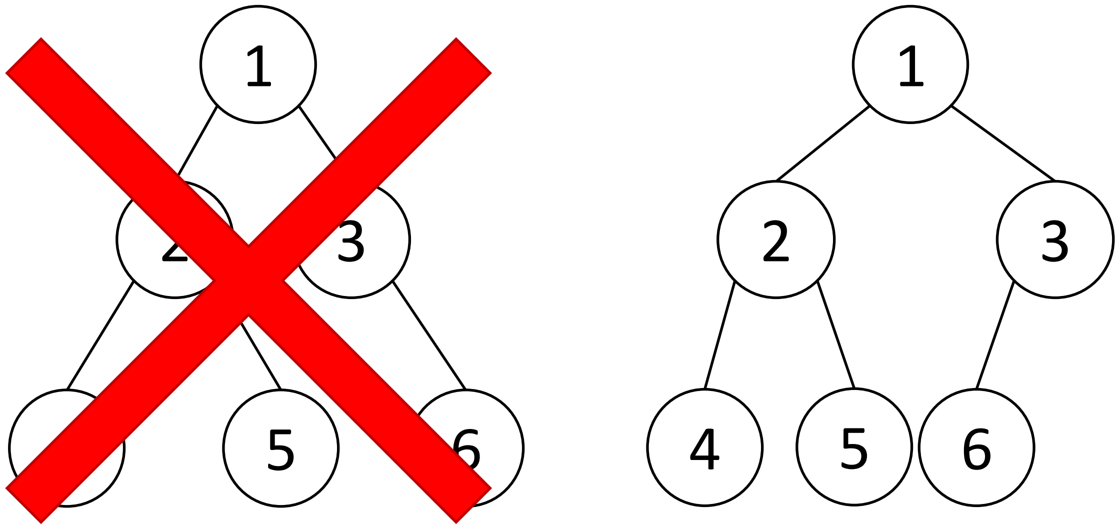
真正線性時間的做法

用來特判 root ，要是沒出現過的數字

```
vector<int> inorder, preorder; // Input

int i = 0, j = 0;
node *dfs(int rightBoundary = INT_MAX) {
  if (j == preorder.size() || inorder[i] == rightBoundary)
    return nullptr;
  node *nd = new node(preorder[j++]);
  nd->lc = dfs(root->data);
  ++i;
  nd->rc = dfs(rightBoundary);
  return nd;
}
```

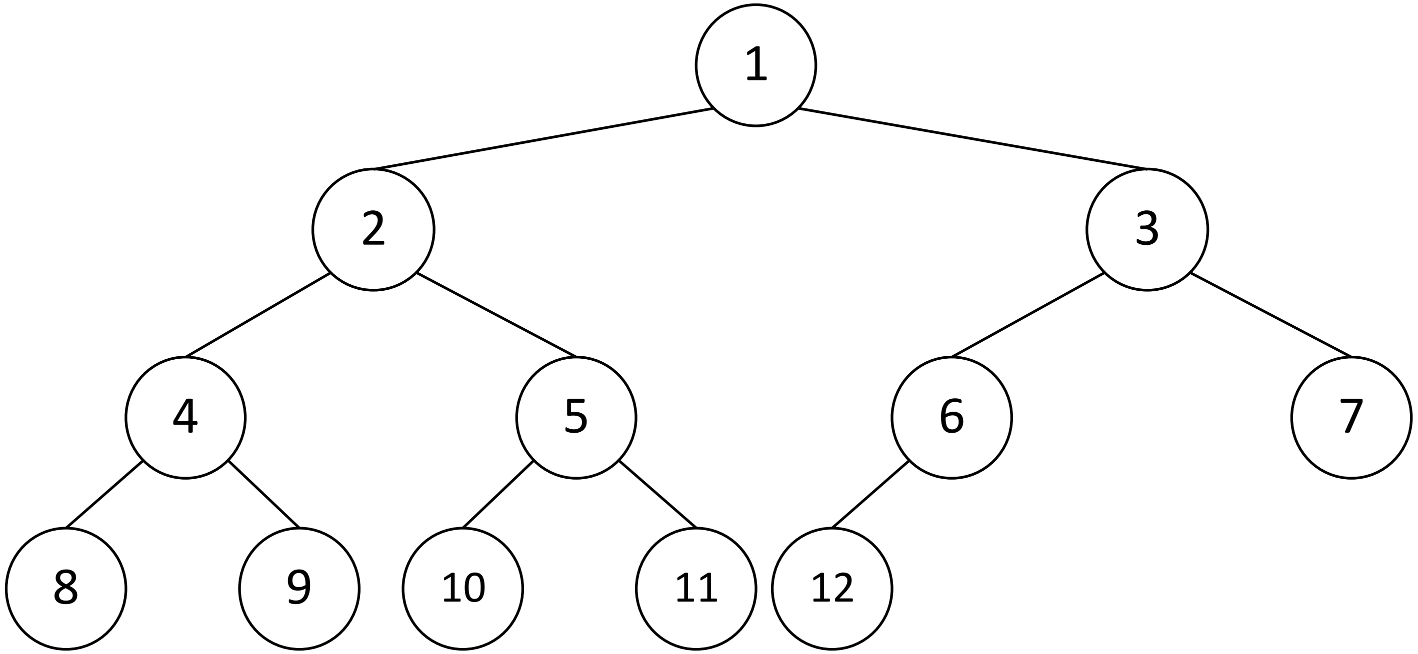
Complete binary tree

- 除了最後一層，每一層都是填滿的
- 最後一層的元素盡量往左靠



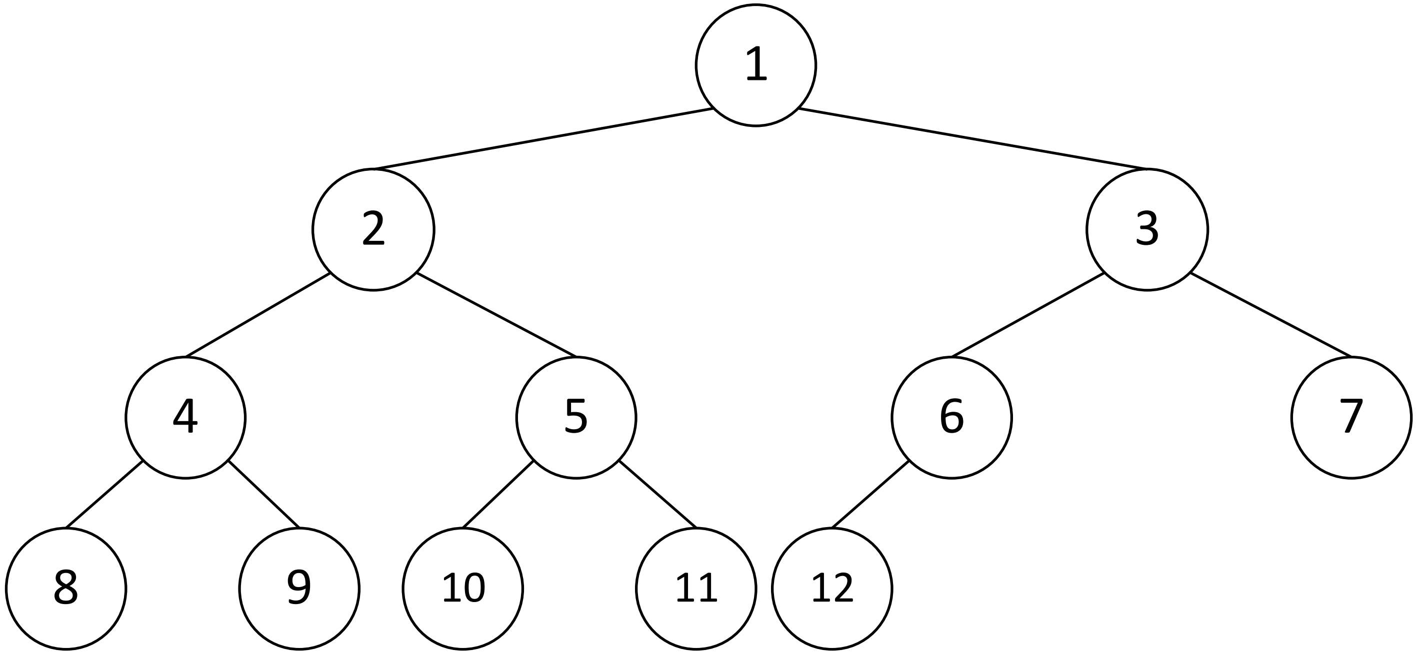
Complete binary tree

- 儲存方式
- 編號為 K 的節點
  其左右子節點的編號分別為
  左：$2K$
  右：$2K+1$
- 編號為 K 的節點
  其 parent 編號為$\lfloor K/2 \rfloor$



Complete binary tree

- 深度保證小於等於 $\lceil \log_2 n \rceil$
- 可以用陣列存

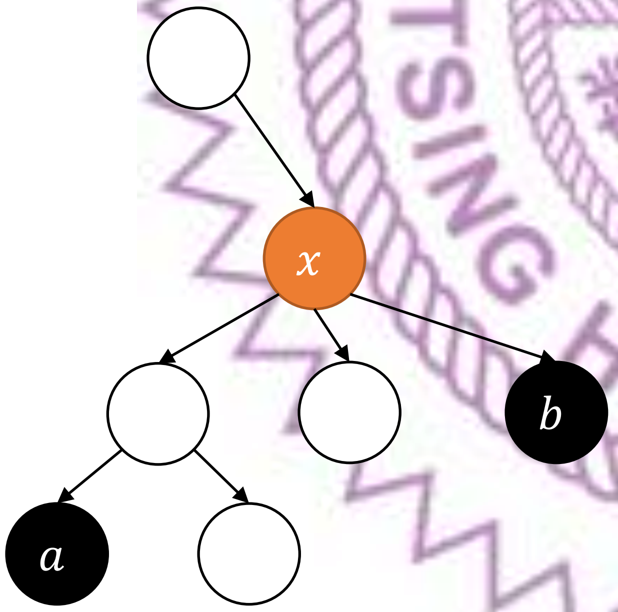


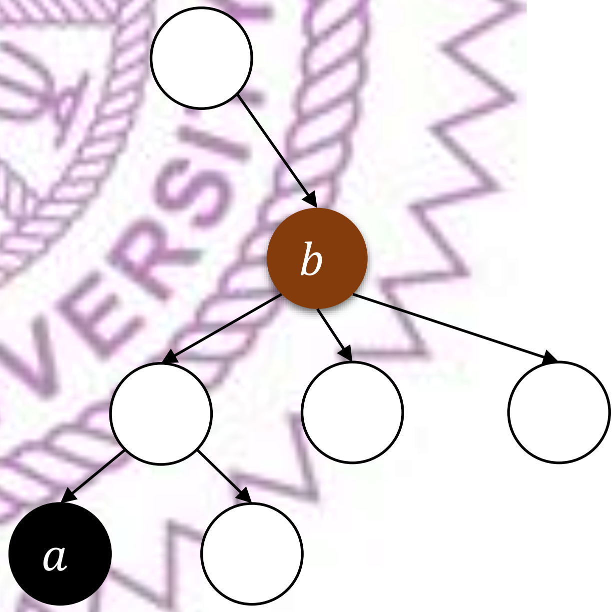
注意index為0的位置不會用到喔

X	1	2	3	4	5	6	7	8	9	10	11	12

最近共同祖先 (LCA)

- 給你一棵有根樹 T，對於樹上任意兩點 a, b 可以找到一個點 x 滿足：x 是 a, b 的祖先且深度最深
- 我們稱 x 是 a, b 的最近共同祖先 (LCA)

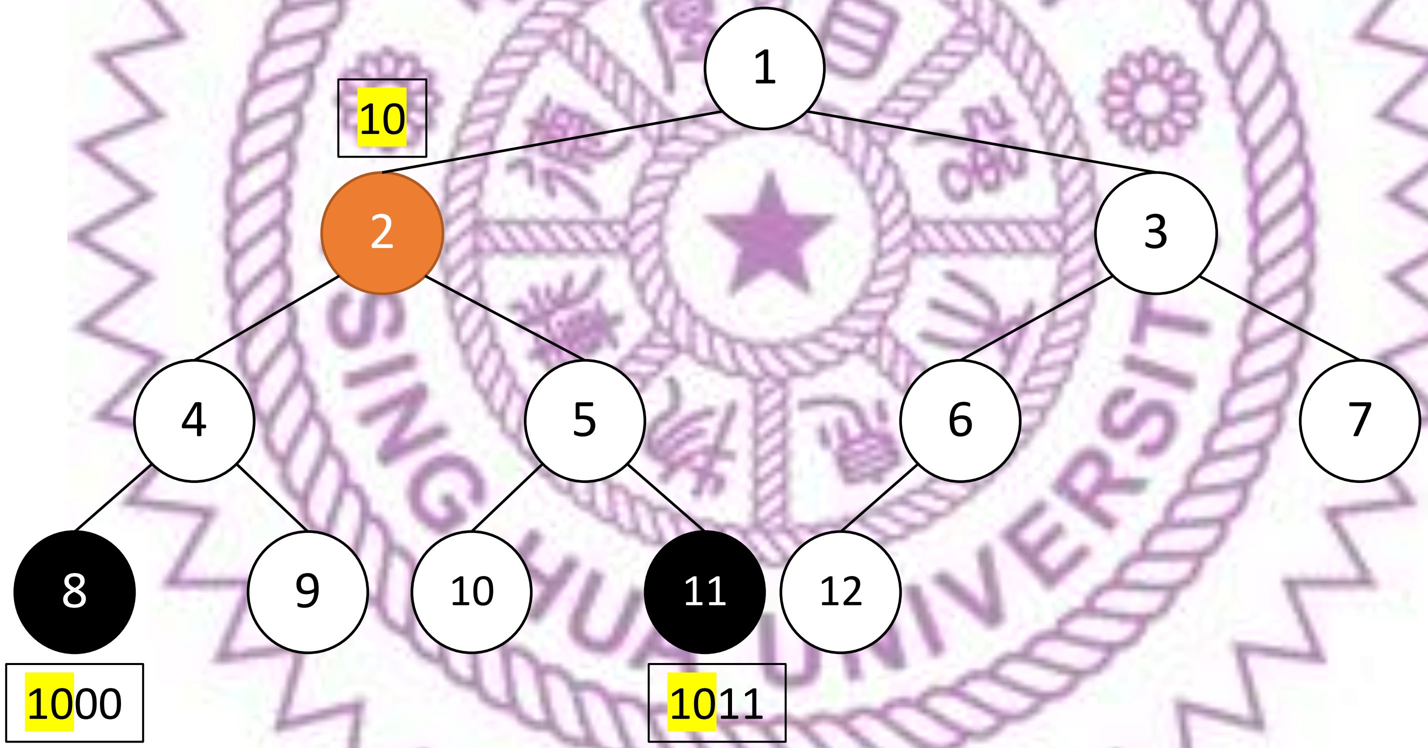




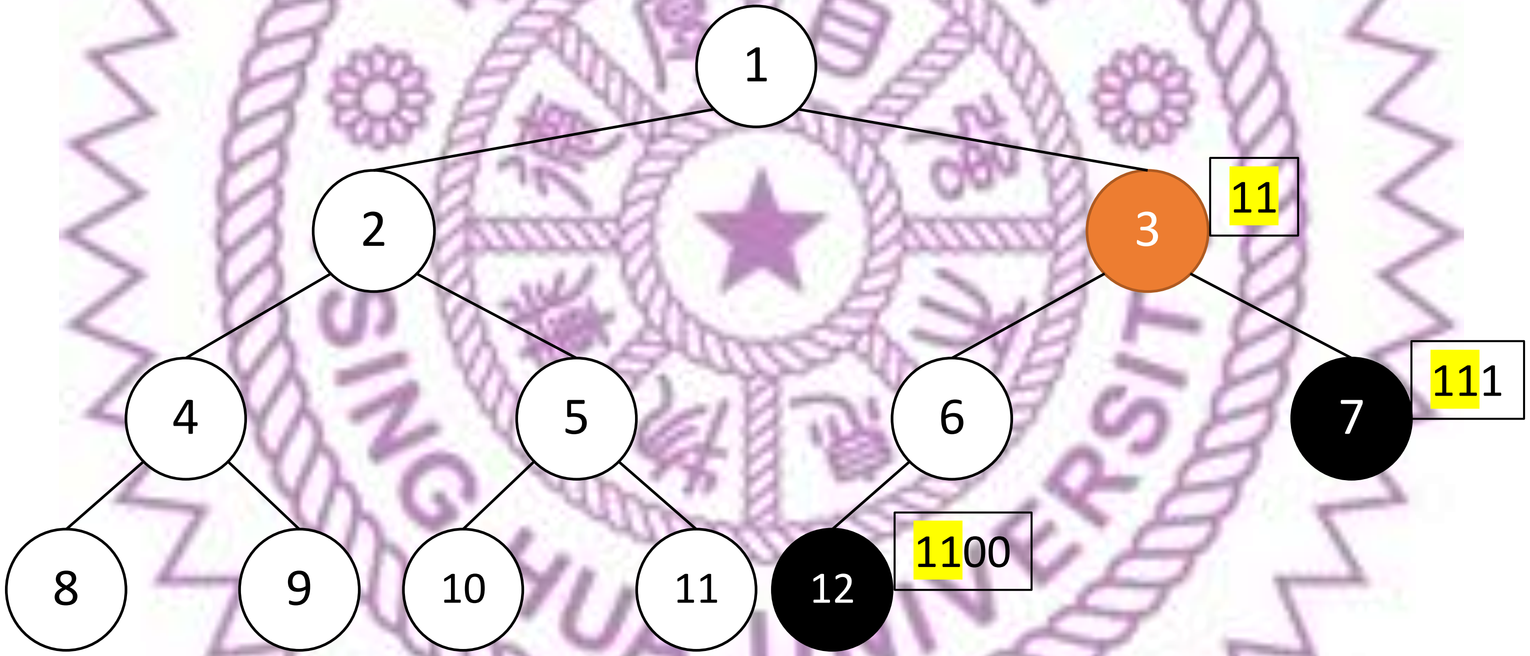
觀察二進位

$$1000_2 \oplus 1011_2 = 0011_2$$

$$1000_2 \gg (\lfloor \log_2 0011_2 \rfloor + 1) = 0010_2$$
$$1011_2 \gg (\lfloor \log_2 0011_2 \rfloor + 1) = 0010_2$$

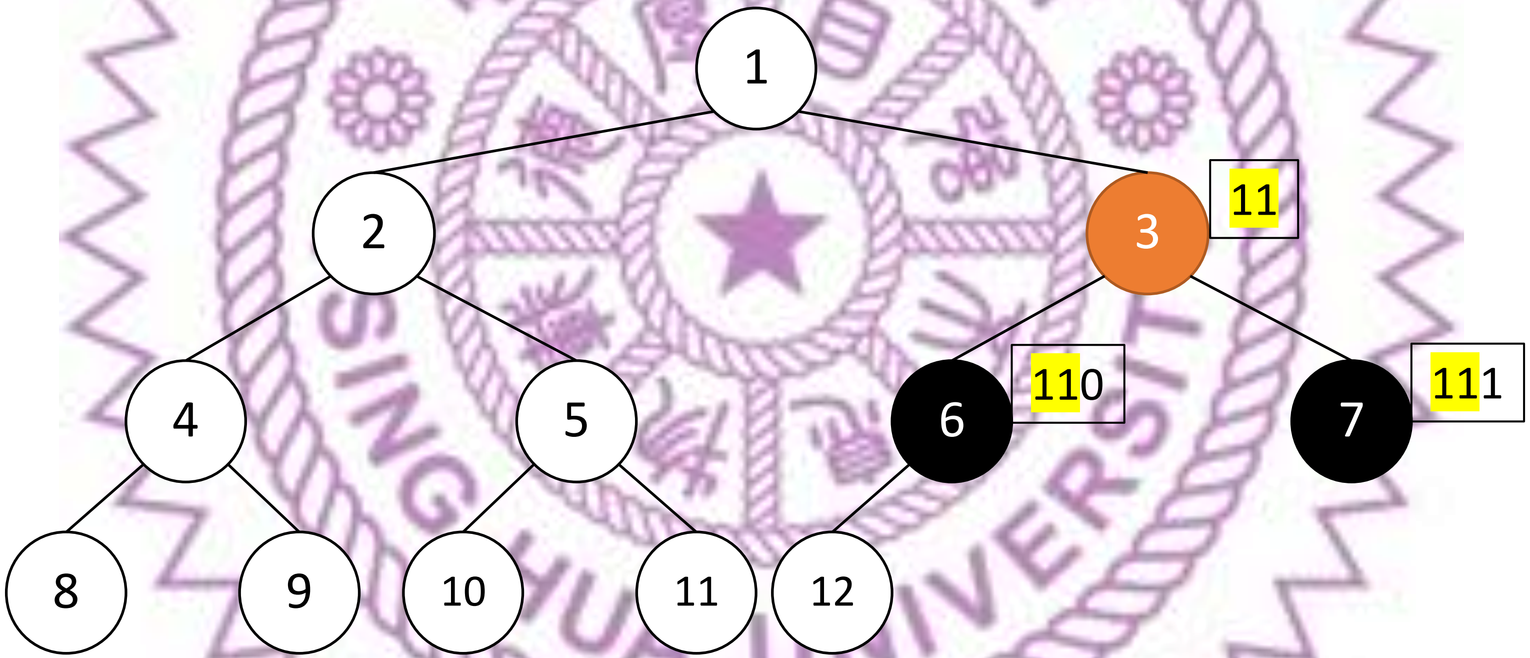


觀察二進位



觀察二進位

$$1100_2 \gg (\lfloor \log_2 1100_2 \rfloor - \lfloor \log_2 0111_2 \rfloor) = 0110_2$$



$O(1)$ 計算 $\lfloor \log_2 x \rfloor$ 的黑魔法 – std::__lg(x)

```
#include <algorithm>
#include <iostream>

int main() {
  // 0 0 1 1 2 2 2 2 3 3
  for (int i = 0; i < 10; ++i)
    std::cout << std::__lg(i) << ' ';
  return 0;
}
```

注意 clang 上沒有這個函數，要自己寫

```
inline int __lg(int __n) {
  return sizeof(int) * __CHAR_BIT__ - 1 - __builtin_clz(__n);
}
```

Complete binary tree - $O(1)$ LCA

```
unsigned getLCA(unsigned a, unsigned b) {
  if (a < b)
    b >>= __lg(b) - __lg(a);
  else
    a >>= __lg(a) - __lg(b);
  return a >> (__lg(a ^ b) + (a != b));
}
```