Tree

by music960633

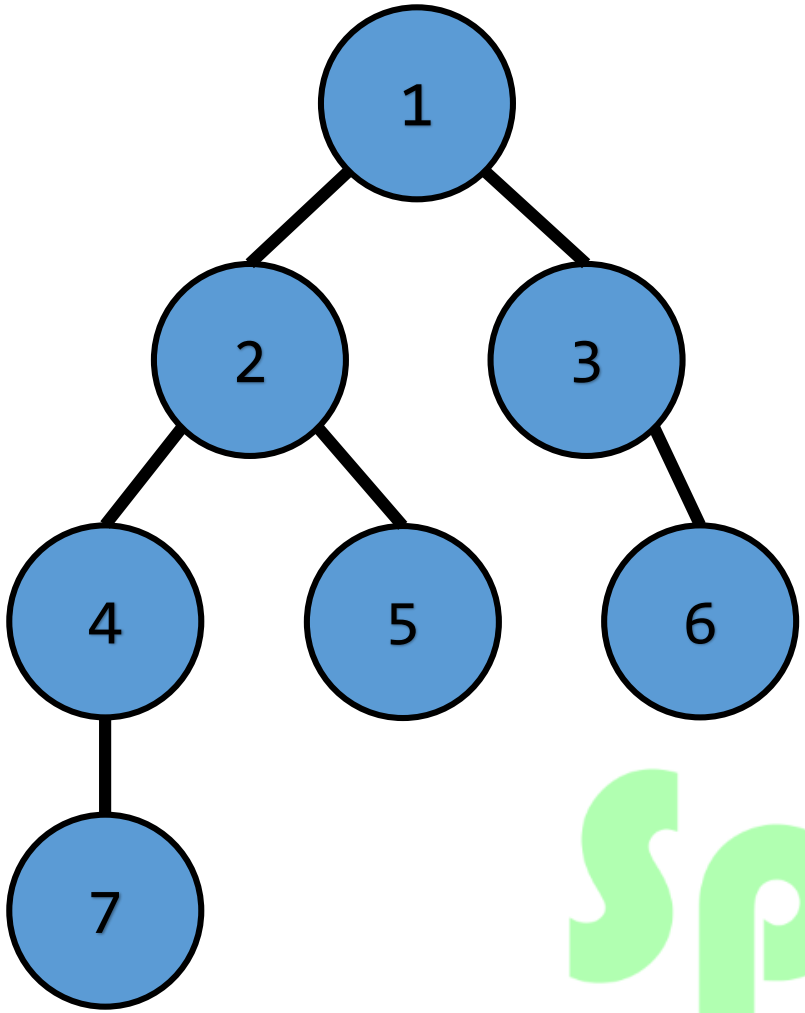
Sprout

課程內容

- Tree！
  - 定義
  - 名詞介紹
  - 性質
  - 紀錄方法
  - 遍歷
  - binary tree
  - complete binary tree

What is a tree?

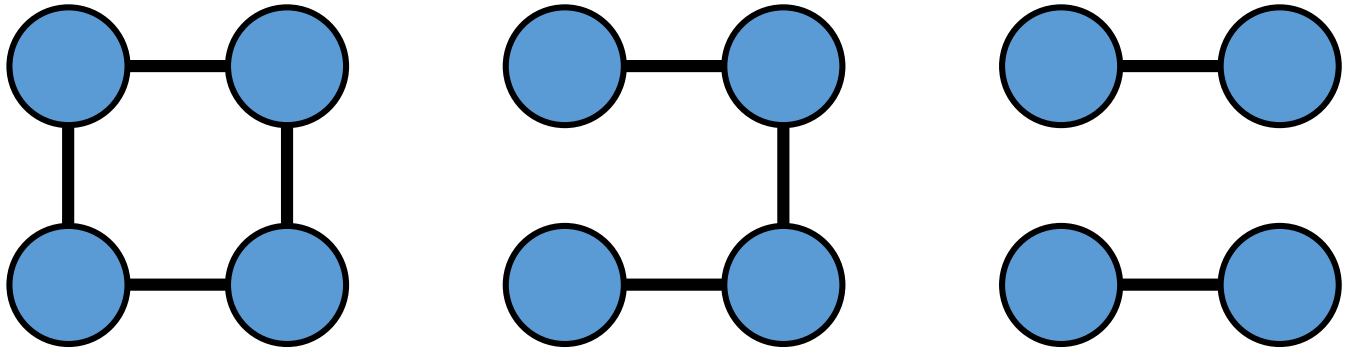
- 這是一顆樹
- 資訊領域中的樹



Sprout

Tree-定義

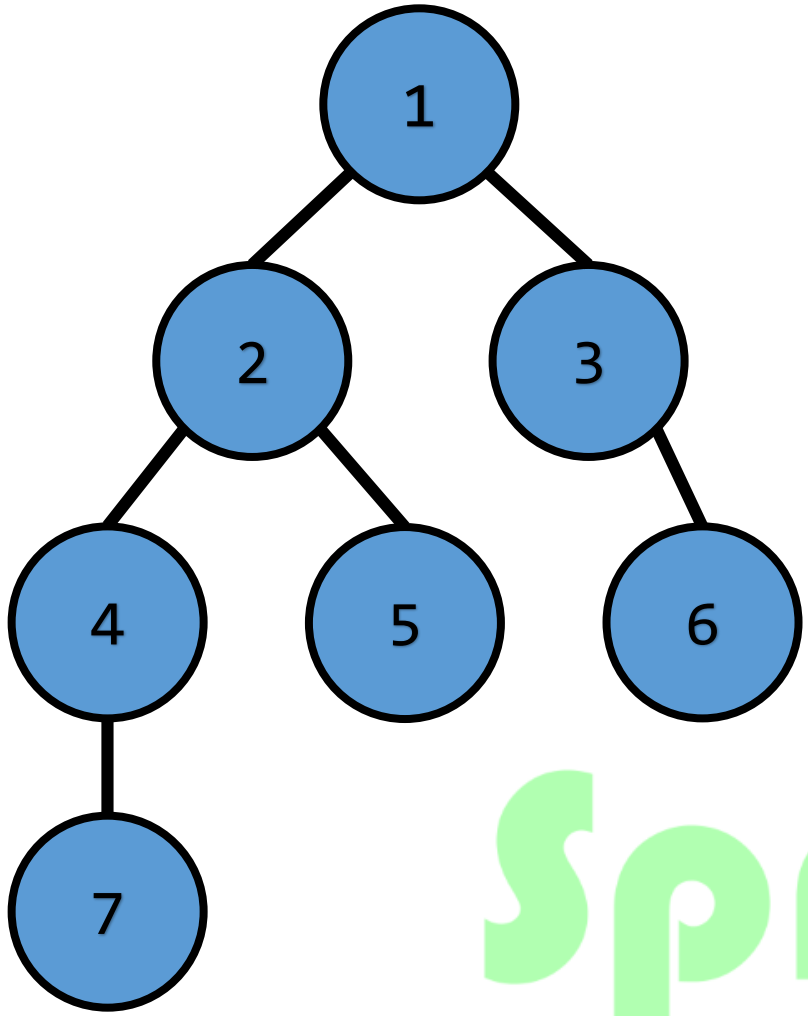
- 定義：沒有環的連通圖
- What? 什麼是環？ 什麼是連通？



Tree!

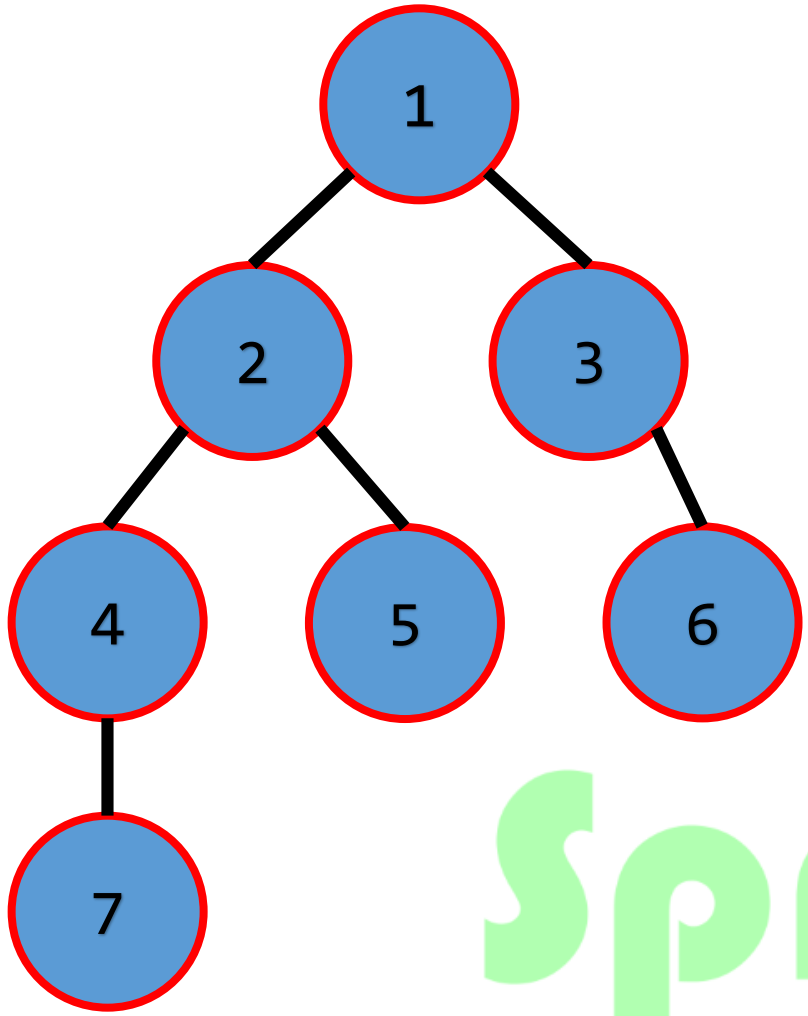
Tree-名詞介紹

- 一棵樹的相關名詞
  - 節點(node)
  - 邊(edge)
  - 根節點(root)
  - 葉節點(leaf)
  - 父節點(parent)
  - 子結點(child)
  - 祖先(ancestor)
  - 子代(descendant)
  - 子樹(subtree)
  - 層(level)
  - 深度(depth)



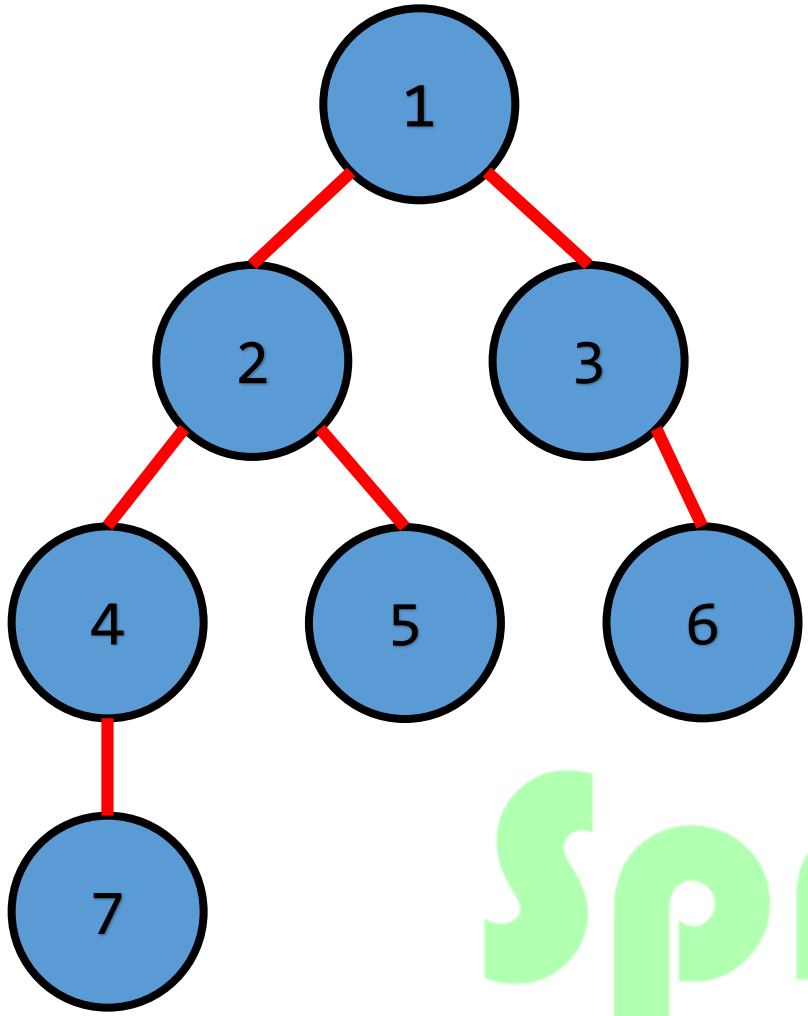
Tree-名詞介紹

- 一棵樹的相關名詞
  - 節點(node)
  - 邊(edge)
  - 根節點(root)
  - 葉節點(leaf)
  - 父節點(parent)
  - 子結點(child)
  - 祖先(ancestor)
  - 子代(descendant)
  - 子樹(subtree)
  - 層(level)
  - 深度(depth)



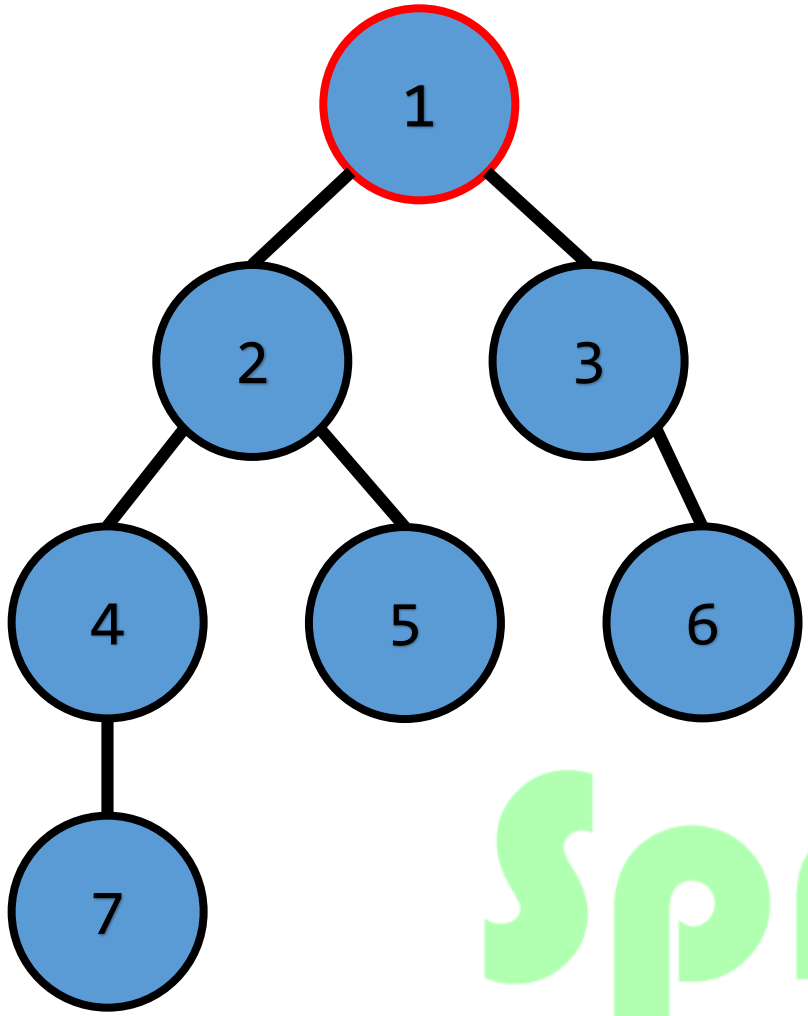
Tree-名詞介紹

- 一棵樹的相關名詞
  - 節點(node)
  - 邊(edge)
  - 根節點(root)
  - 葉節點(leaf)
  - 父節點(parent)
  - 子結點(child)
  - 祖先(ancestor)
  - 子代(descendant)
  - 子樹(subtree)
  - 層(level)
  - 深度(depth)



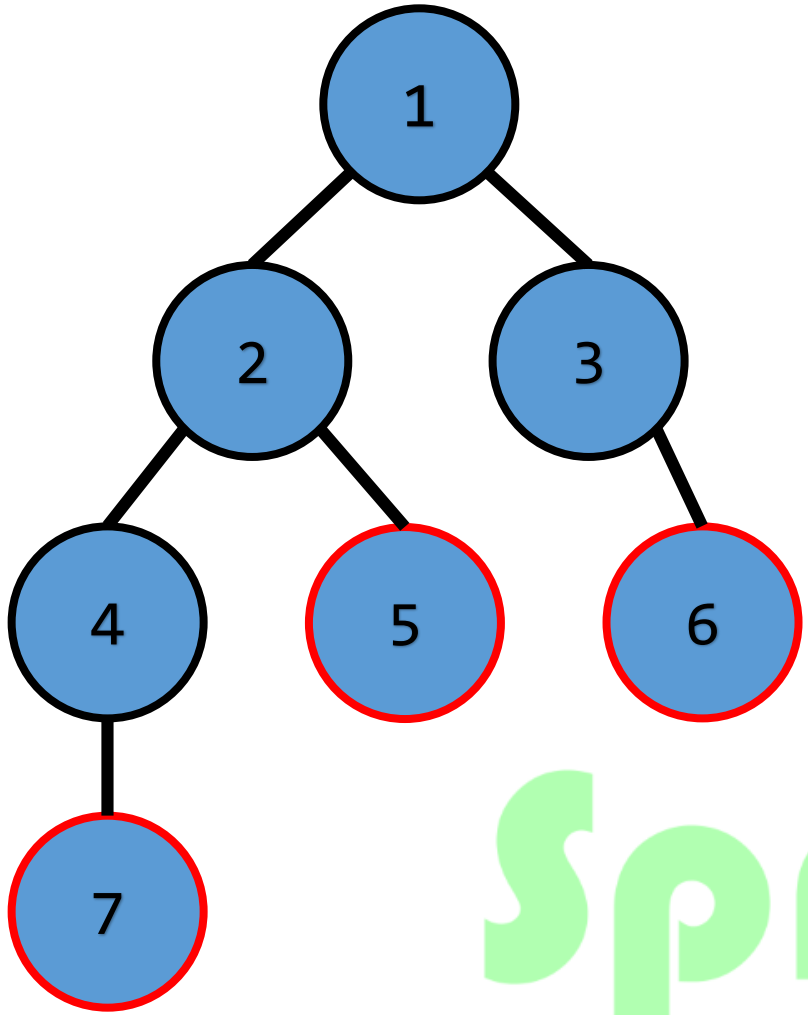
Tree-名詞介紹

- 一棵樹的相關名詞
  - 節點(node)
  - 邊(edge)
  - 根節點(root)
  - 葉節點(leaf)
  - 父節點(parent)
  - 子結點(child)
  - 祖先(ancestor)
  - 子代(descendant)
  - 子樹(subtree)
  - 層(level)
  - 深度(depth)



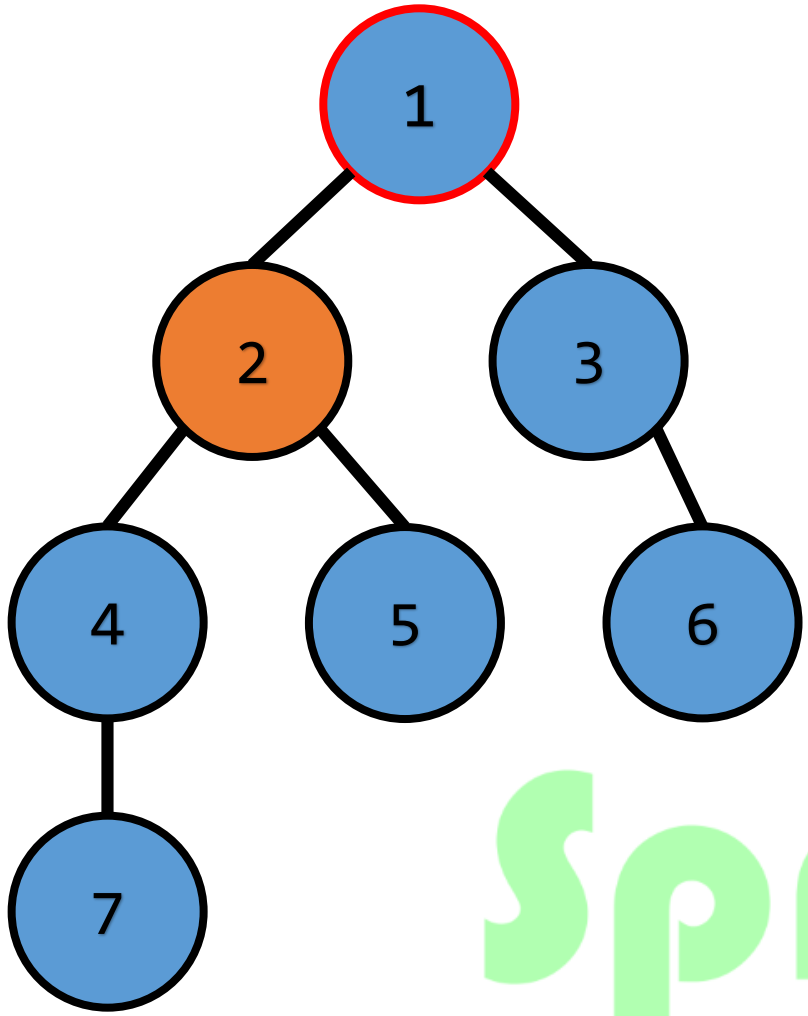
Tree-名詞介紹

- 一棵樹的相關名詞
  - 節點(node)
  - 邊(edge)
  - 根節點(root)
  - 葉節點(leaf)
  - 父節點(parent)
  - 子結點(child)
  - 祖先(ancestor)
  - 子代(descendant)
  - 子樹(subtree)
  - 層(level)
  - 深度(depth)



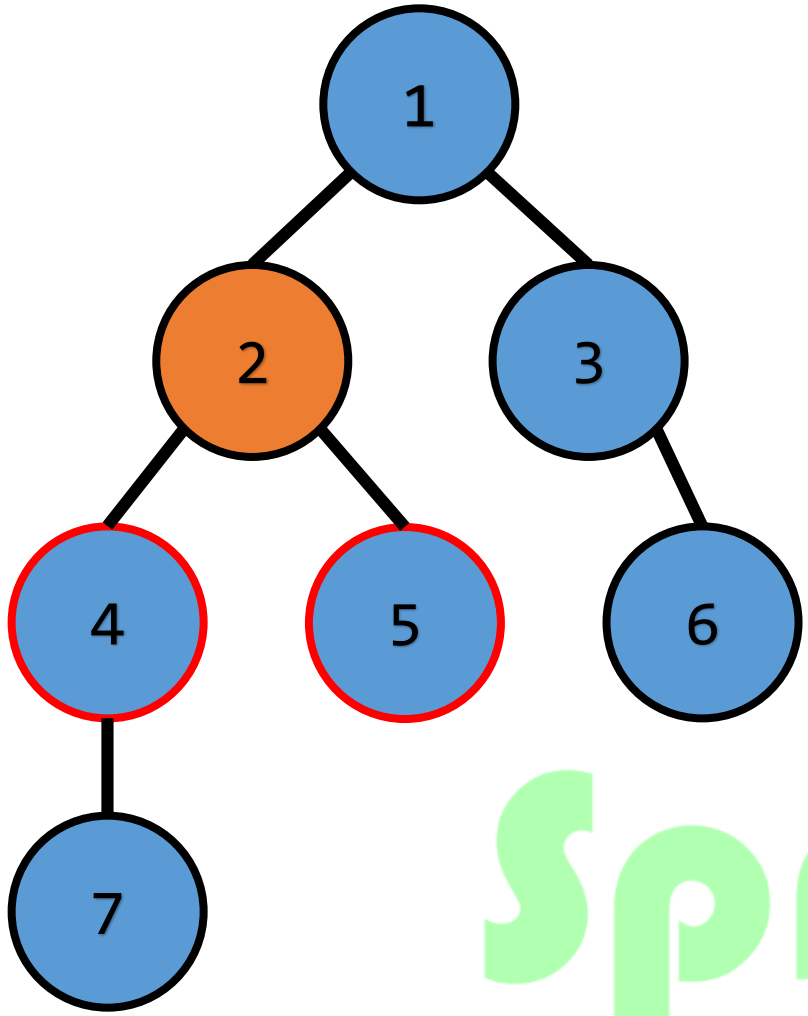
Tree-名詞介紹

- 一棵樹的相關名詞
  - 節點(node)
  - 邊(edge)
  - 根節點(root)
  - 葉節點(leaf)
  - 父節點(parent)
  - 子結點(child)
  - 祖先(ancestor)
  - 子代(descendant)
  - 子樹(subtree)
  - 層(level)
  - 深度(depth)



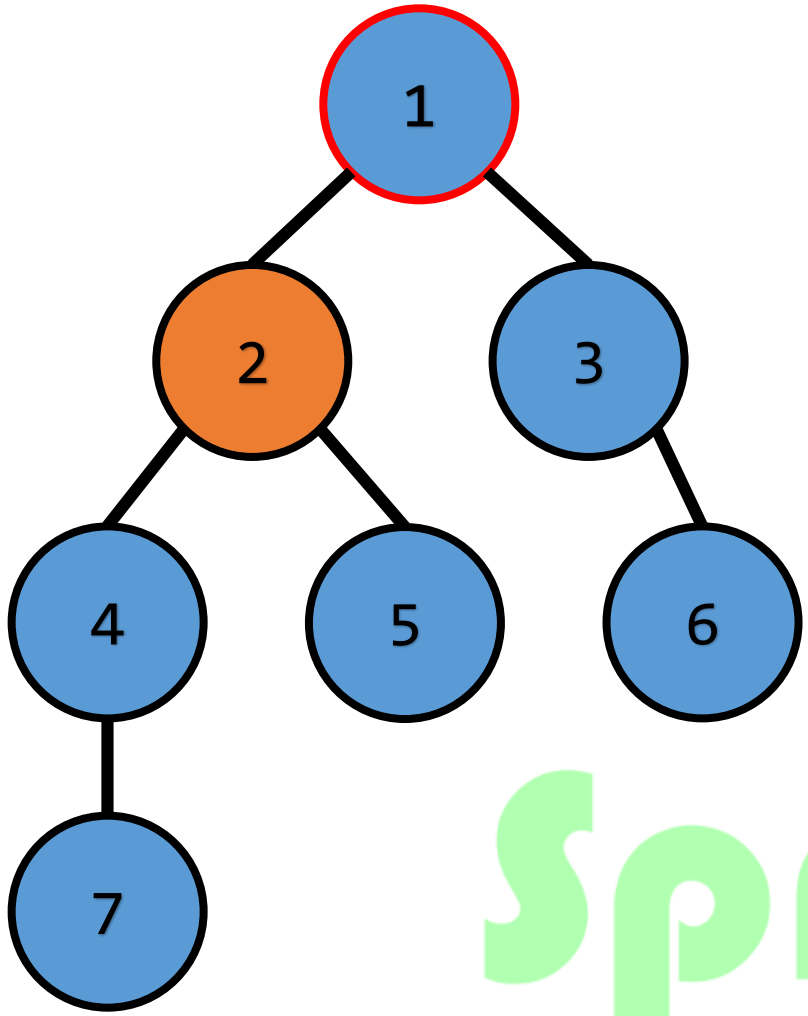
Tree-名詞介紹

- 一棵樹的相關名詞
  - 節點(node)
  - 邊(edge)
  - 根節點(root)
  - 葉節點(leaf)
  - 父節點(parent)
  - 子結點(child)
  - 祖先(ancestor)
  - 子代(descendant)
  - 子樹(subtree)
  - 層(level)
  - 深度(depth)



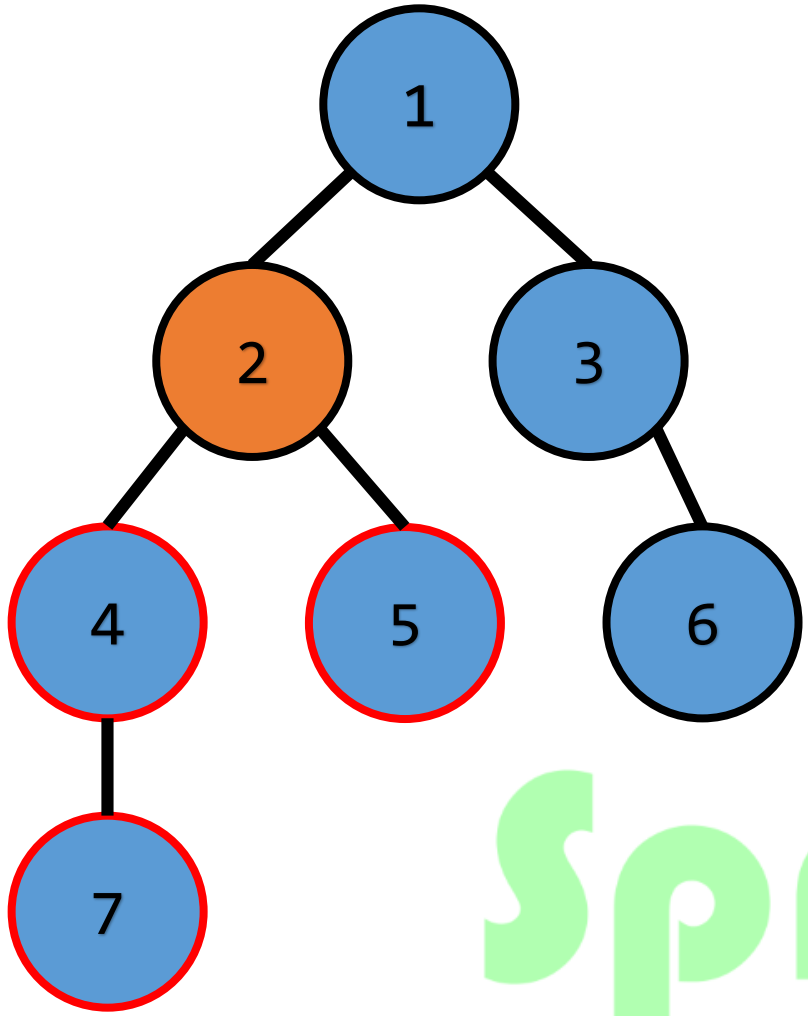
Tree-名詞介紹

- 一棵樹的相關名詞
  - 節點(node)
  - 邊(edge)
  - 根節點(root)
  - 葉節點(leaf)
  - 父節點(parent)
  - 子結點(child)
  - 祖先(ancestor)
  - 子代(descendant)
  - 子樹(subtree)
  - 層(level)
  - 深度(depth)



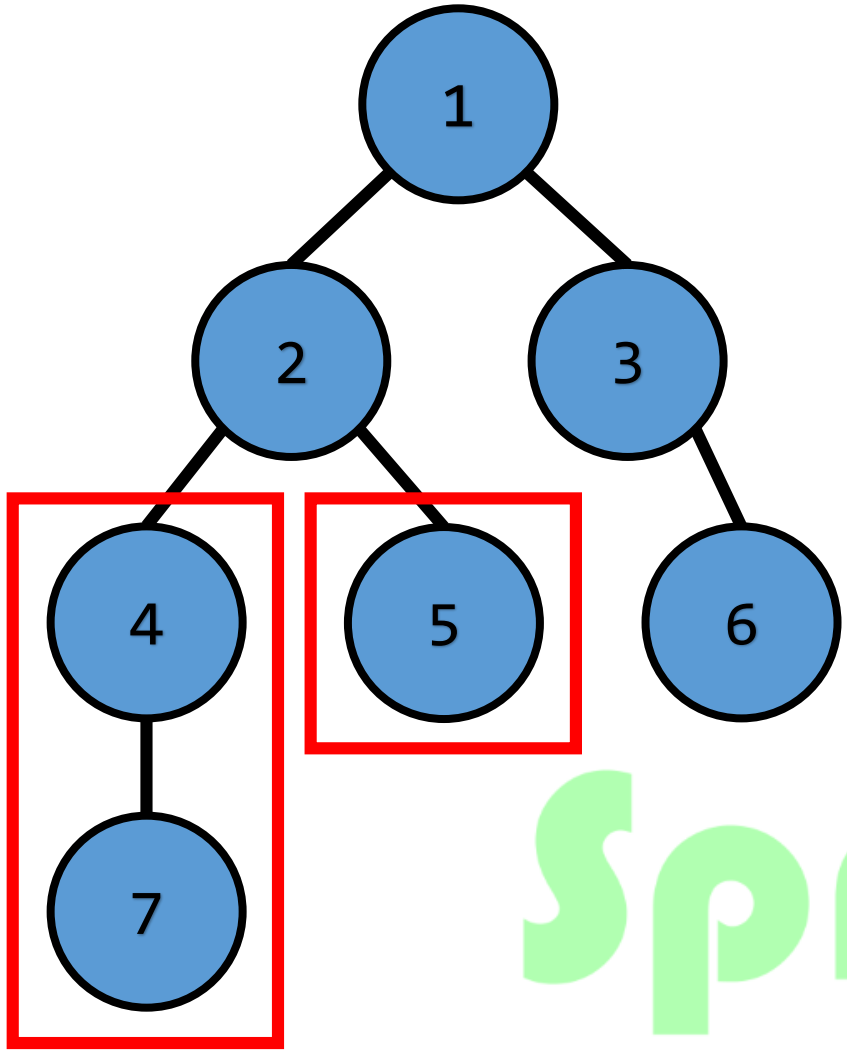
Tree-名詞介紹

- 一棵樹的相關名詞
  - 節點(node)
  - 邊(edge)
  - 根節點(root)
  - 葉節點(leaf)
  - 父節點(parent)
  - 子結點(child)
  - 祖先(ancestor)
  - 子代(descendant)
  - 子樹(subtree)
  - 層(level)
  - 深度(depth)



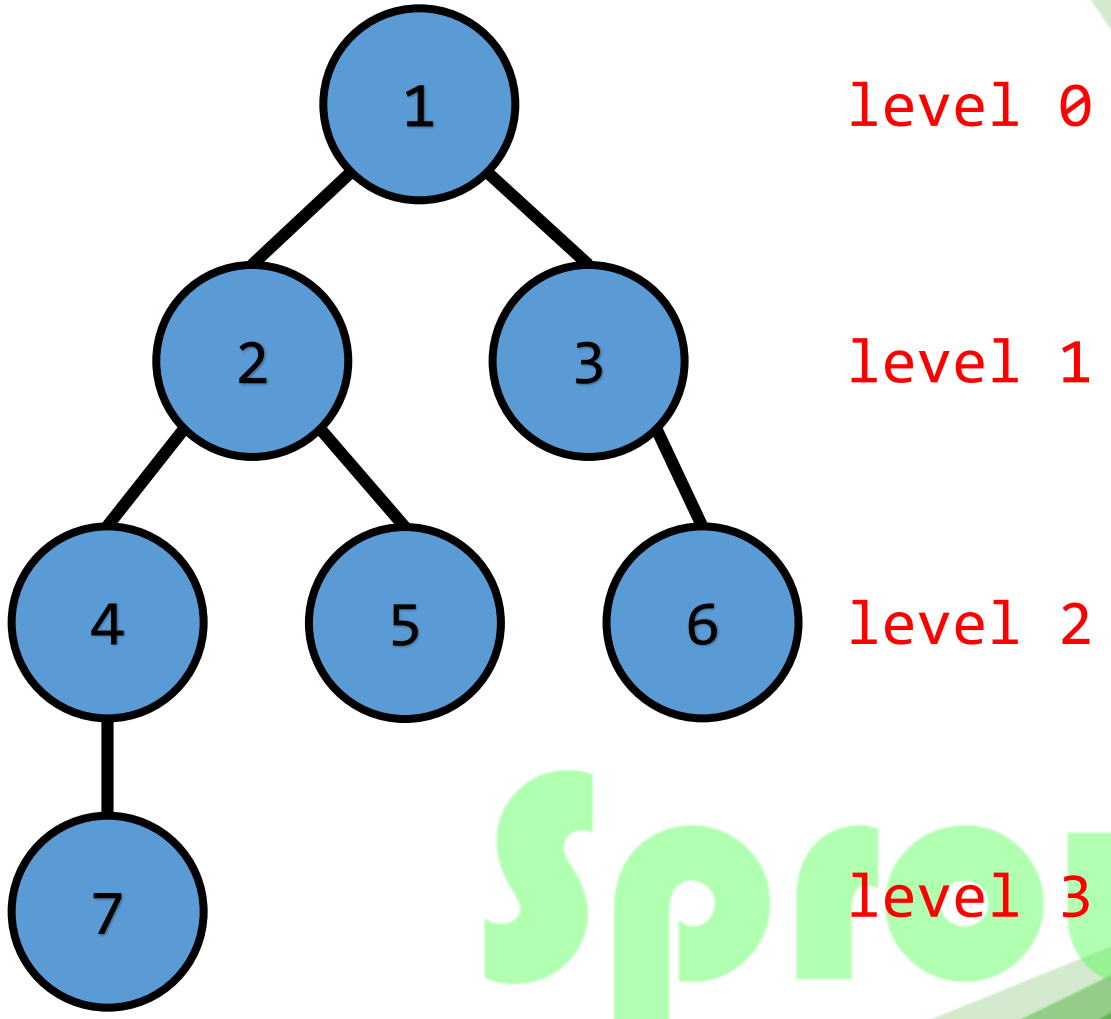
Tree-名詞介紹

- 一棵樹的相關名詞
  - 節點(node)
  - 邊(edge)
  - 根節點(root)
  - 葉節點(leaf)
  - 父節點(parent)
  - 子結點(child)
  - 祖先(ancestor)
  - 子代(descendant)
  - 子樹(subtree)
  - 層(level)
  - 深度(depth)



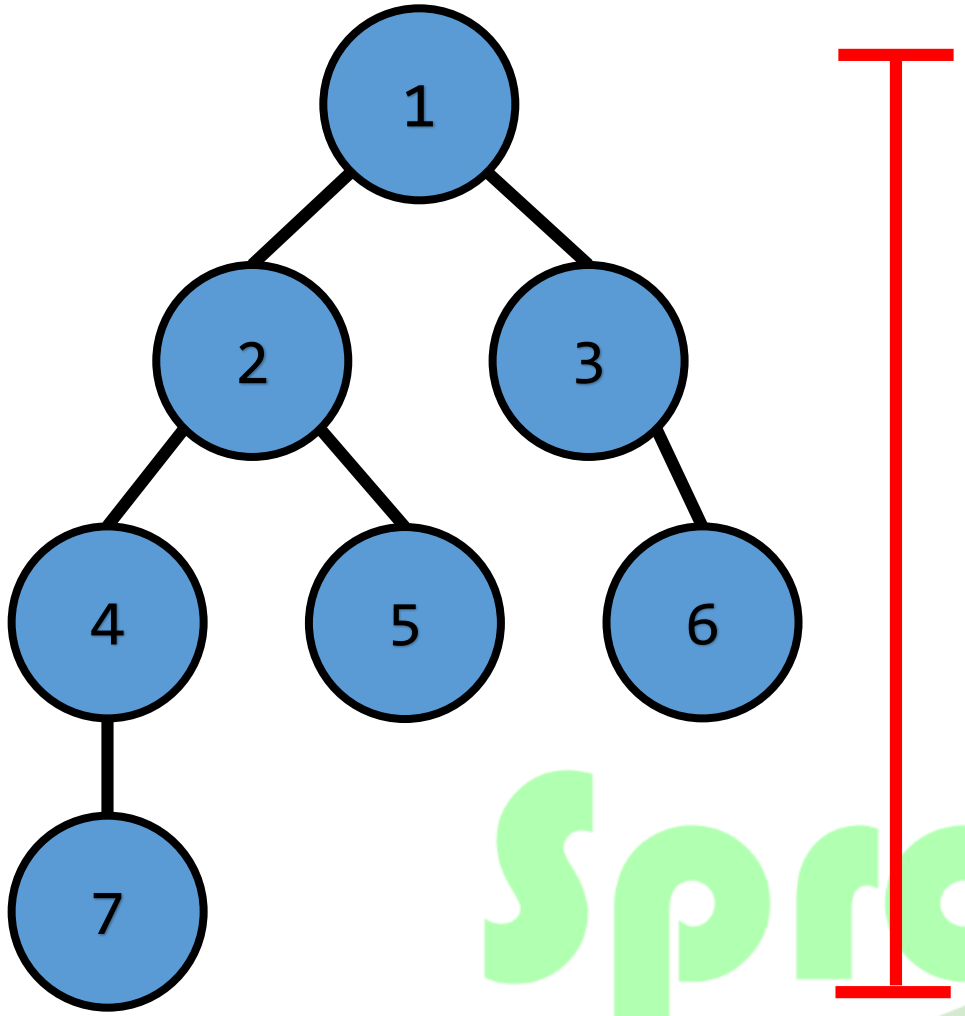
Tree-名詞介紹

- 一棵樹的相關名詞
  - 節點(node)
  - 邊(edge)
  - 根節點(root)
  - 葉節點(leaf)
  - 父節點(parent)
  - 子結點(child)
  - 祖先(ancestor)
  - 子代(descendant)
  - 子樹(subtree)
  - 層(level)
  - 深度(depth)



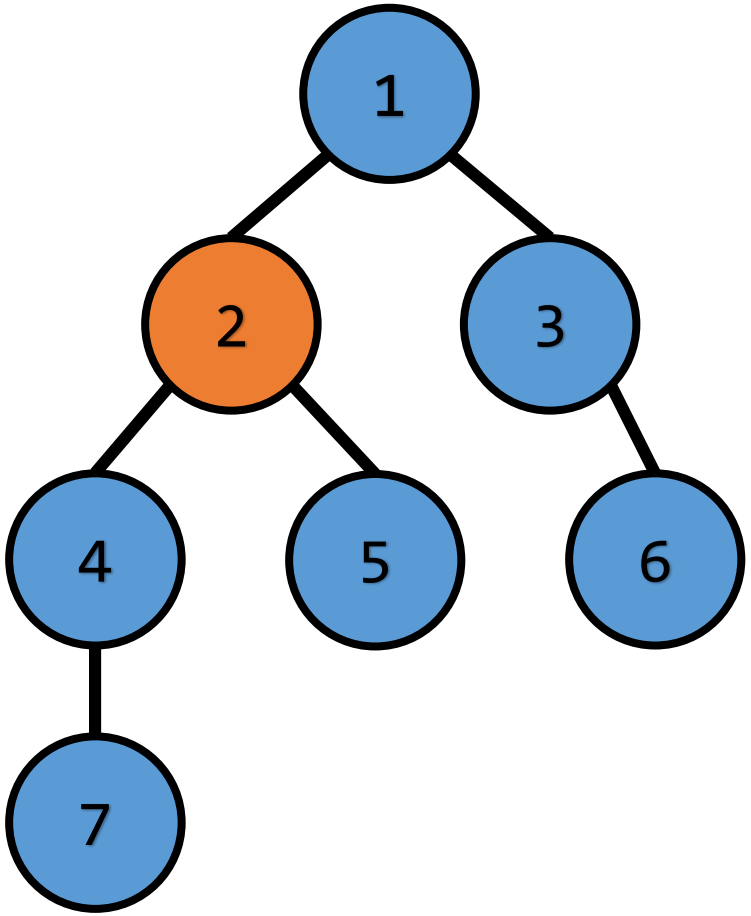
Tree-名詞介紹

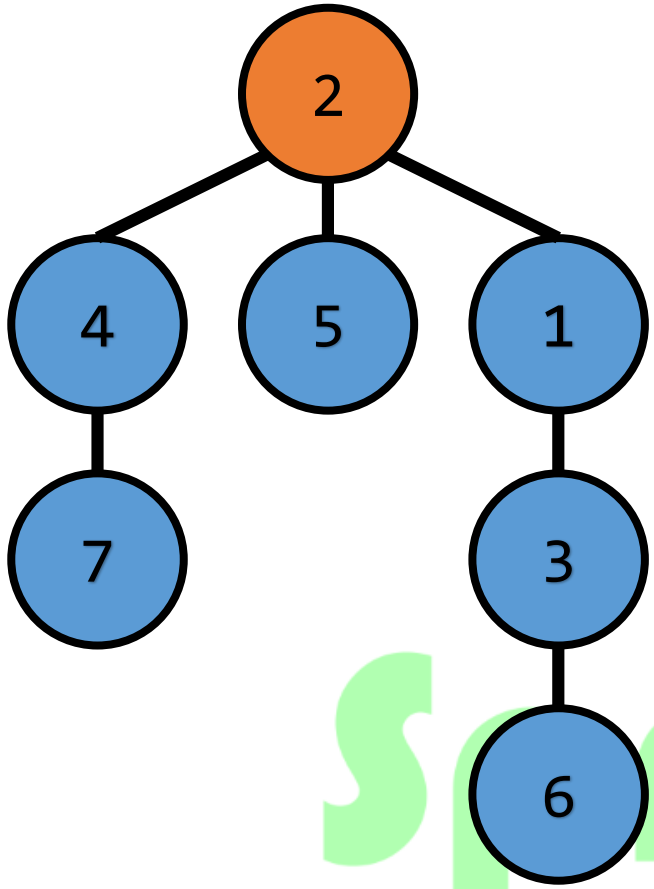
- 一棵樹的相關名詞
  - 節點(node)
  - 邊(edge)
  - 根節點(root)
  - 葉節點(leaf)
  - 父節點(parent)
  - 子結點(child)
  - 祖先(ancestor)
  - 子代(descendant)
  - 子樹(subtree)
  - 層(level)
  - 深度(depth)



Tree-性質

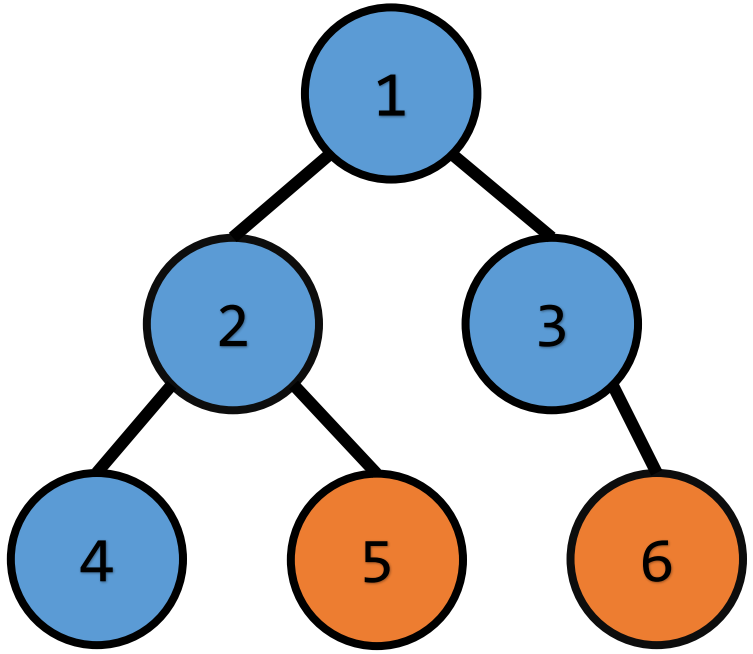
- 1. 任何一點都可以當作root





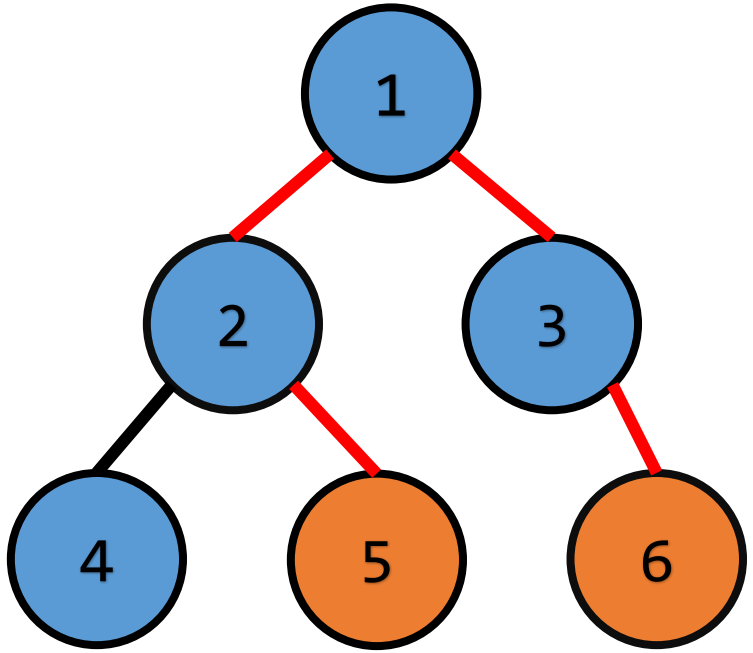
Tree-性質

- 1. 任何一點都可以當作root
- 2. 任兩點間恰有一條不經過重複點的路徑



Tree-性質

- 1. 任何一點都可以當作root
- 2. 任兩點間恰有一條不經過重複點的路徑



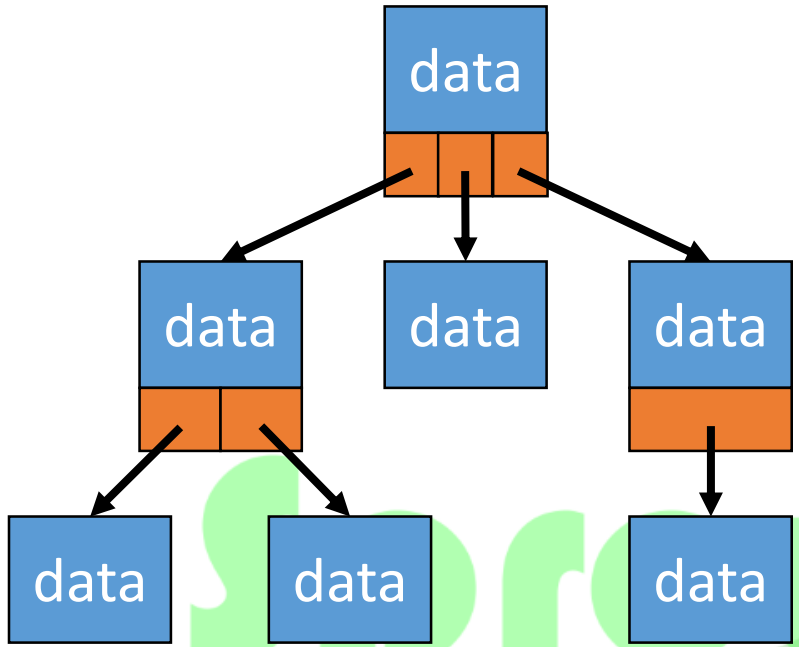
Tree-性質

- 1. 任何一點都可以當作root
- 2. 任兩點間恰有一條不經過重複點的路徑
- 3. 一棵有N個節點的樹恰有(N-1)條邊

Tree-紀錄

- 如何紀錄這樣的一個結構呢？
- 看到Node.....類似linked list的結構！
- 問題：到底要開幾個pointer?

```
1  #include <stdio.h>
2
3  struct Node{
4      int _data;
5      Node *_child1, *_child2, ...???
6  };
7
```



Tree-紀錄

- Solution: 利用linked list或dynamic array

```cpp
#include <stdio.h>
#include <vector>

using namespace std;

struct Node{
    int             _data;
    vector<Node*> _child;
};

```

Tree-記錄

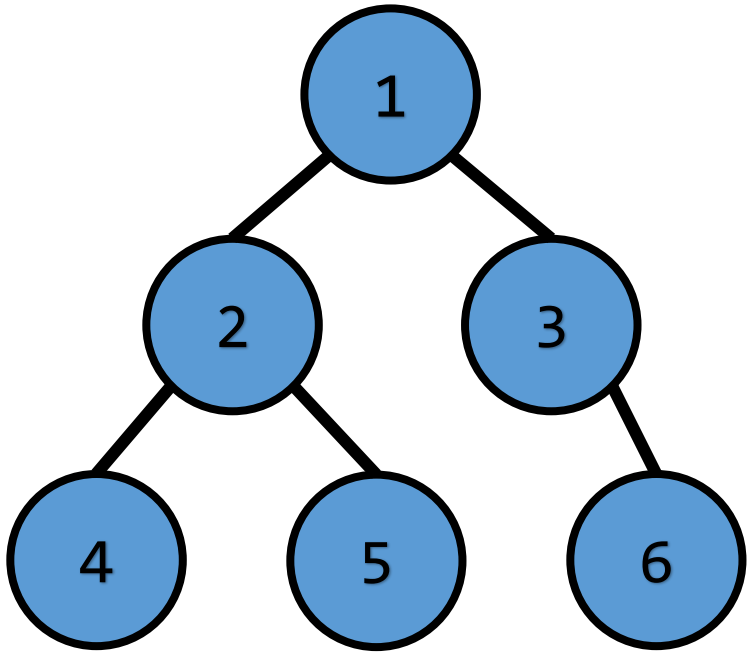
- 另外一個常見的方法：

```
3   using namespace std;
4
5   int data[SIZE];
6   vector<int> child[SIZE];
```

- 使用時機：當題目給定每個點的編號的時候

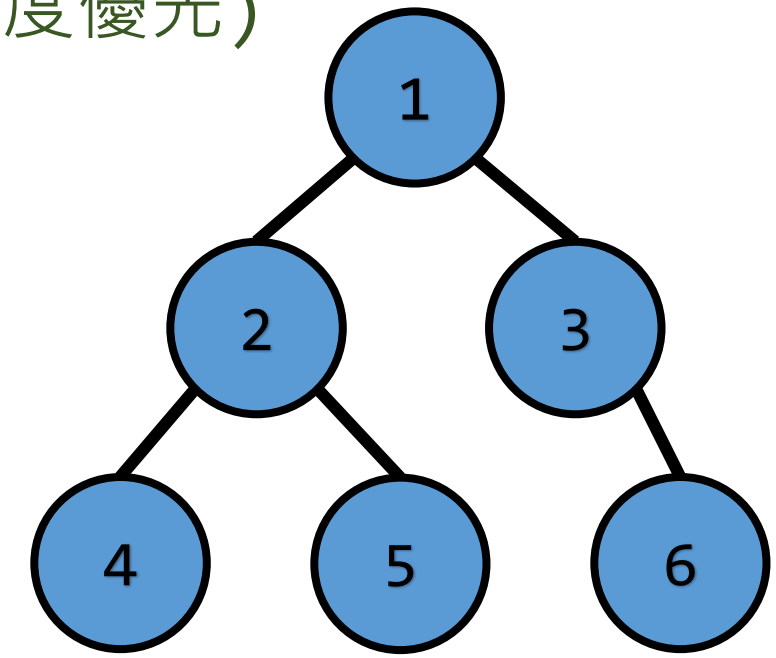
Tree-遍歷

- 如何從root開始，走遍所有的node呢？
  (例如：將每個節點上的data印出來)



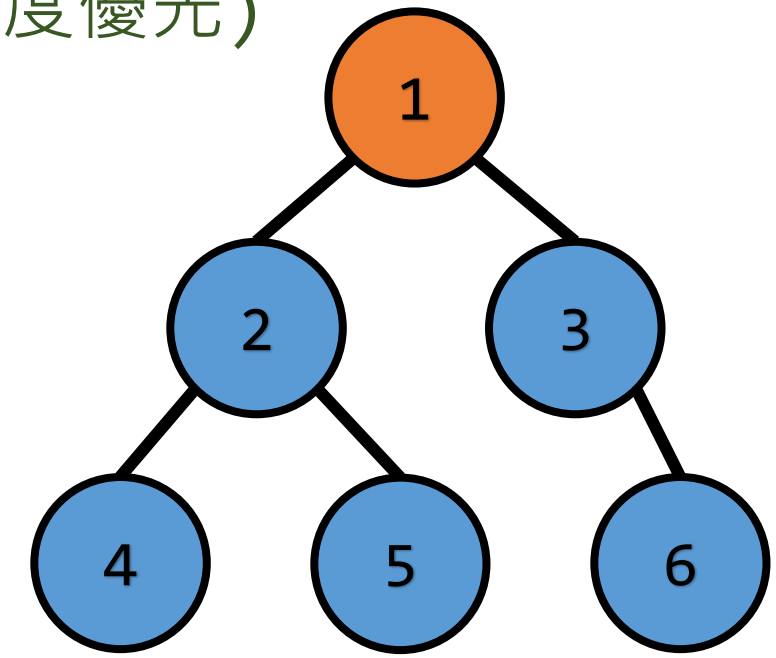
Tree-遍歷

- 如何從root開始，走遍所有的node呢？
  (例如：將每個節點上的data印出來)
- DFS(深度優先)



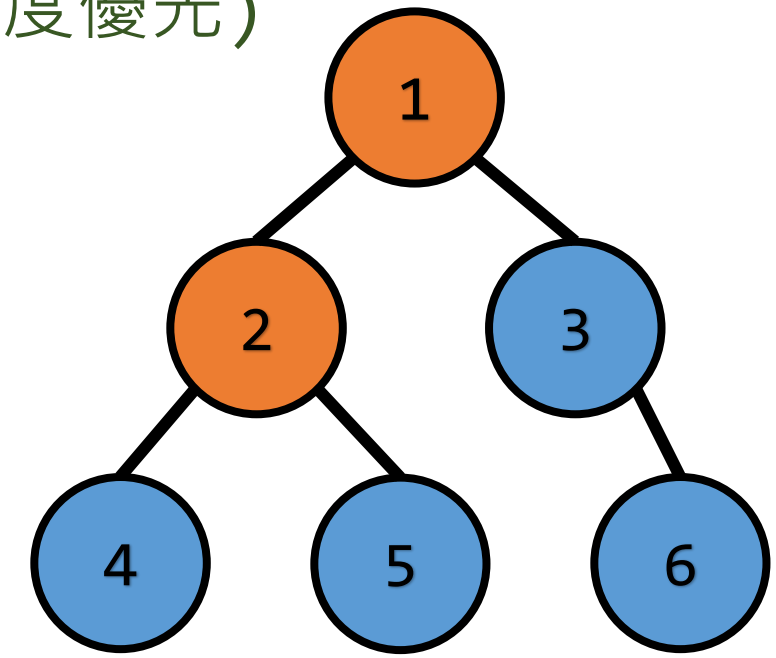
Tree-遍歷

- 如何從root開始，走遍所有的node呢？
  (例如：將每個節點上的data印出來)
- DFS(深度優先)



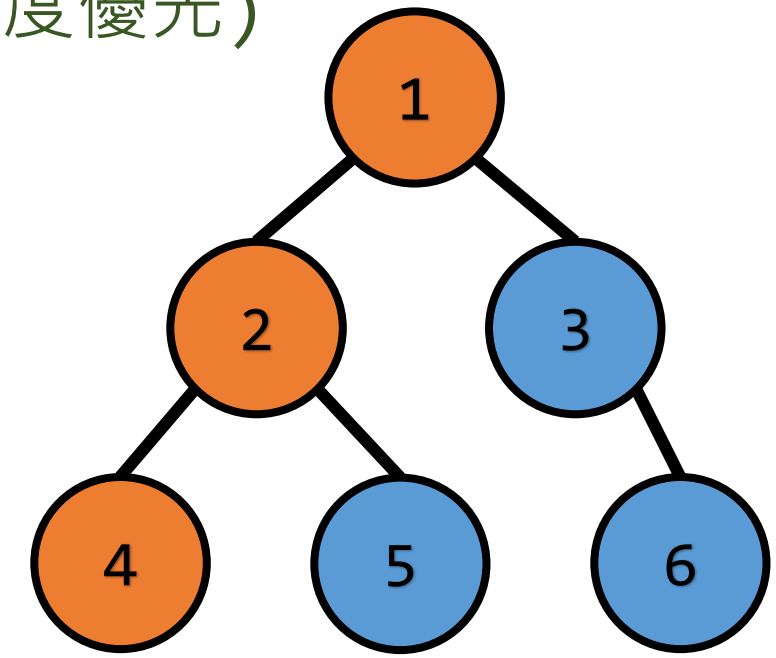
Tree-遍歷

- 如何從root開始，走遍所有的node呢？
  (例如：將每個節點上的data印出來)
- DFS(深度優先)



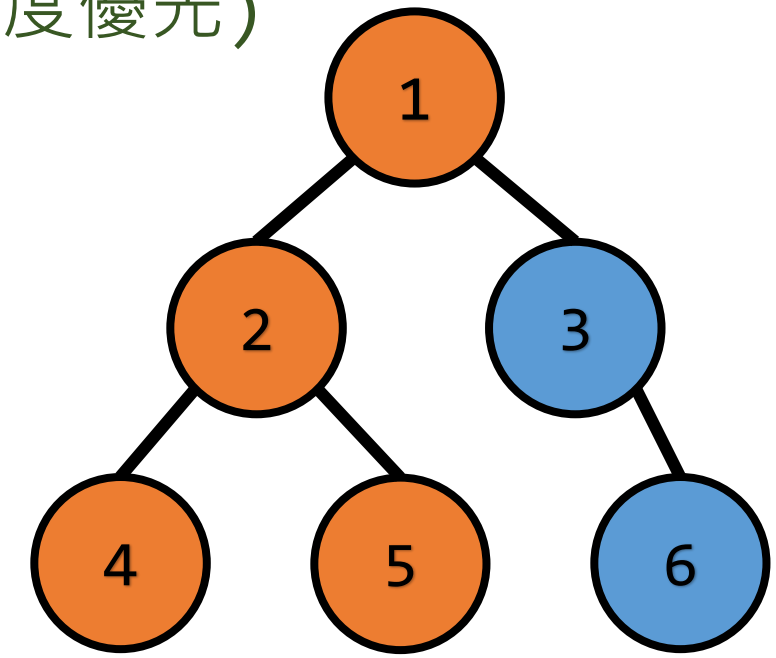
Tree-遍歷

- 如何從root開始，走遍所有的node呢？
  (例如：將每個節點上的data印出來)
- DFS(深度優先)



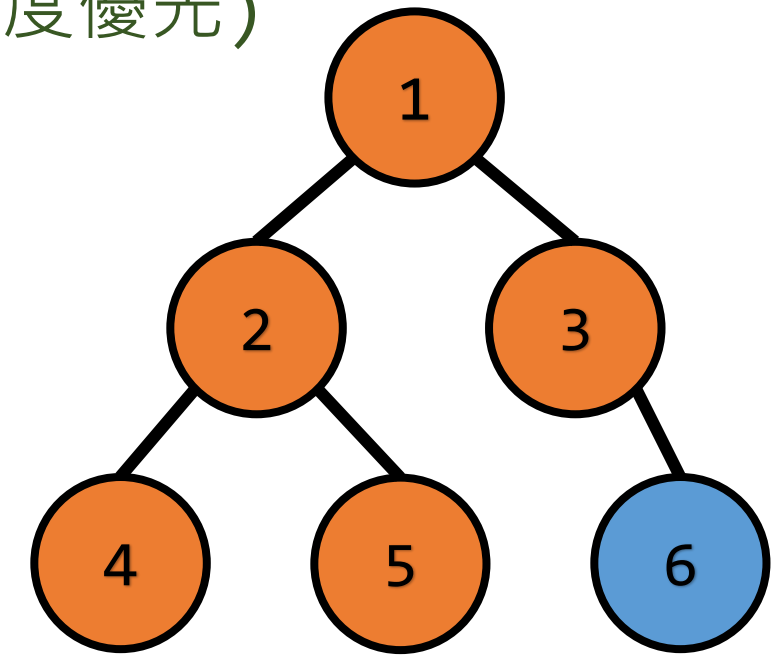
Tree-遍歷

- 如何從root開始，走遍所有的node呢？
  (例如：將每個節點上的data印出來)
- DFS(深度優先)



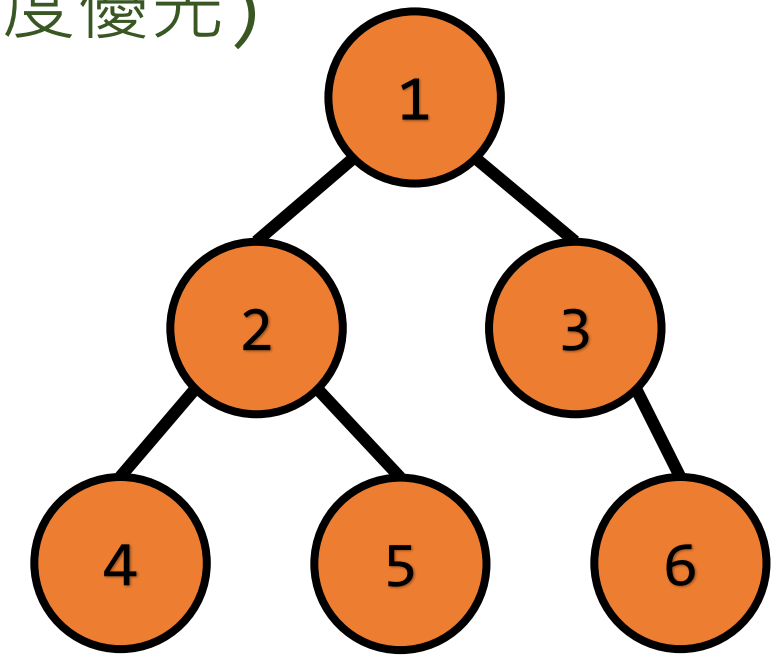
Tree-遍歷

- 如何從root開始，走遍所有的node呢？
  (例如：將每個節點上的data印出來)
- DFS(深度優先)



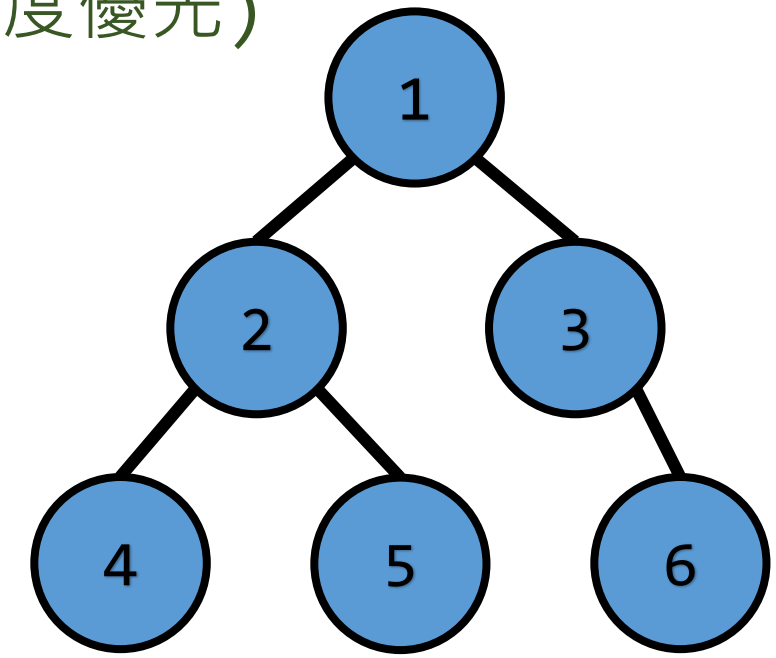
Tree-遍歷

- 如何從root開始，走遍所有的node呢？
  (例如：將每個節點上的data印出來)
- DFS(深度優先)



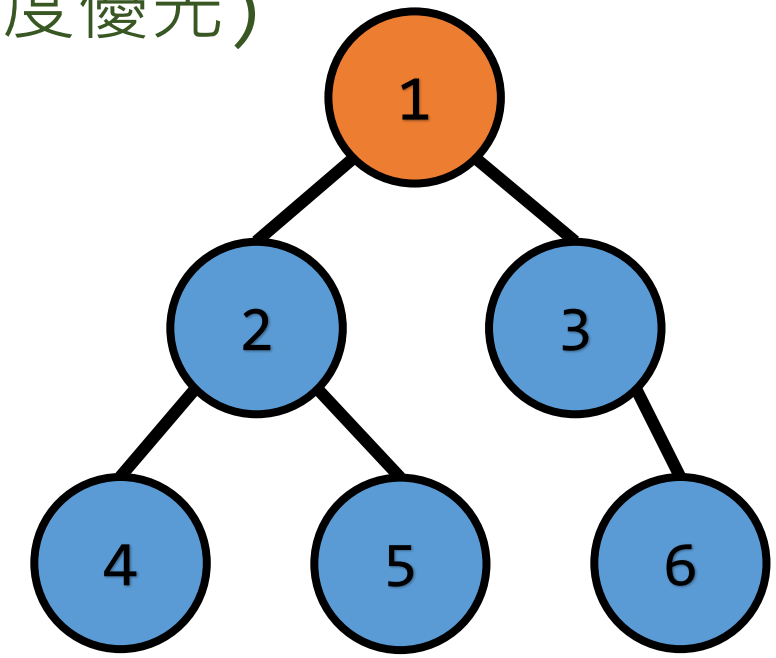
Tree-遍歷

- 如何從root開始，走遍所有的node呢？
  (例如：將每個節點上的data印出來)
- BFS(廣度優先)



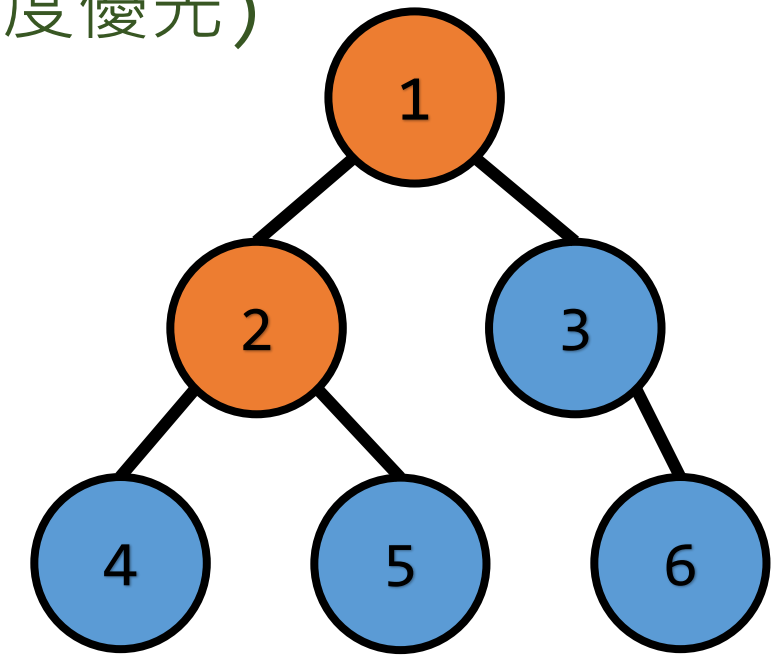
Tree-遍歷

- 如何從root開始，走遍所有的node呢？
  (例如：將每個節點上的data印出來)
- BFS(廣度優先)



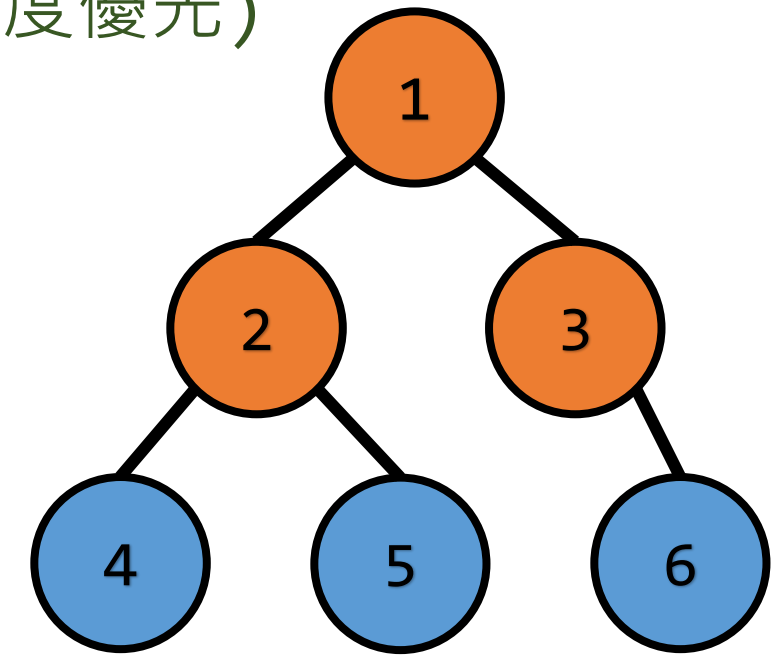
Tree-遍歷

- 如何從root開始，走遍所有的node呢？
  (例如：將每個節點上的data印出來)
- BFS(廣度優先)



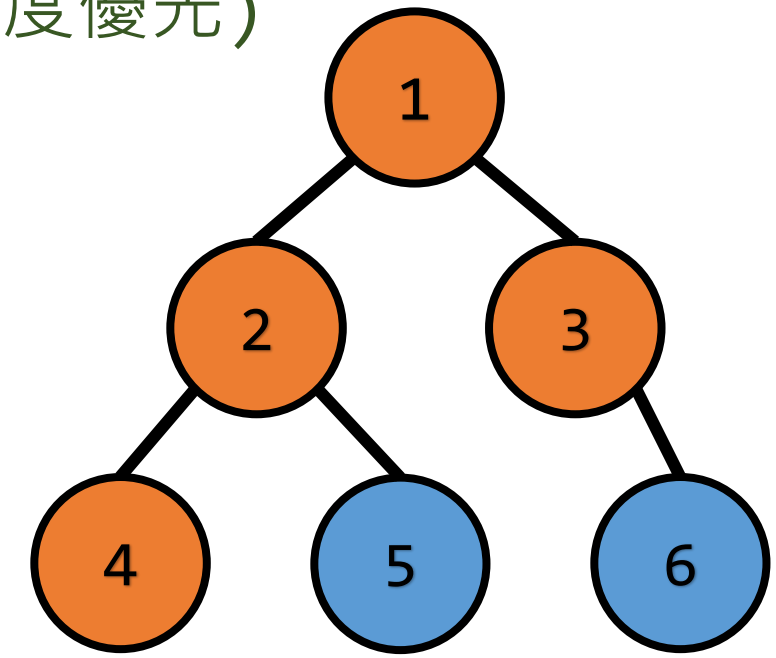
Tree-遍歷

- 如何從root開始，走遍所有的node呢？
  (例如：將每個節點上的data印出來)
- BFS(廣度優先)



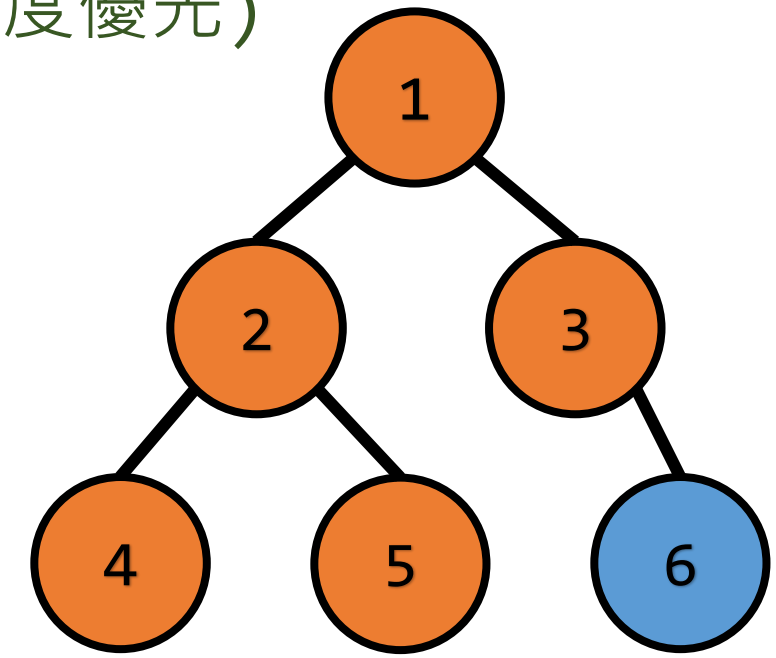
Tree-遍歷

- 如何從root開始，走遍所有的node呢？
  (例如：將每個節點上的data印出來)
- BFS(廣度優先)



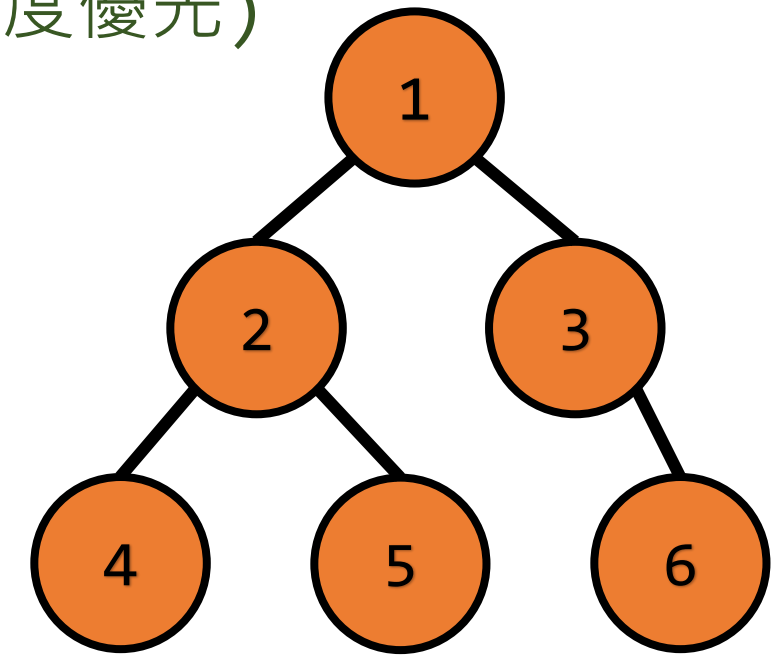
Tree-遍歷

- 如何從root開始，走遍所有的node呢？
  (例如：將每個節點上的data印出來)
- BFS(廣度優先)



Tree-遍歷

- 如何從root開始，走遍所有的node呢？
  (例如：將每個節點上的data印出來)
- BFS(廣度優先)



Binary tree

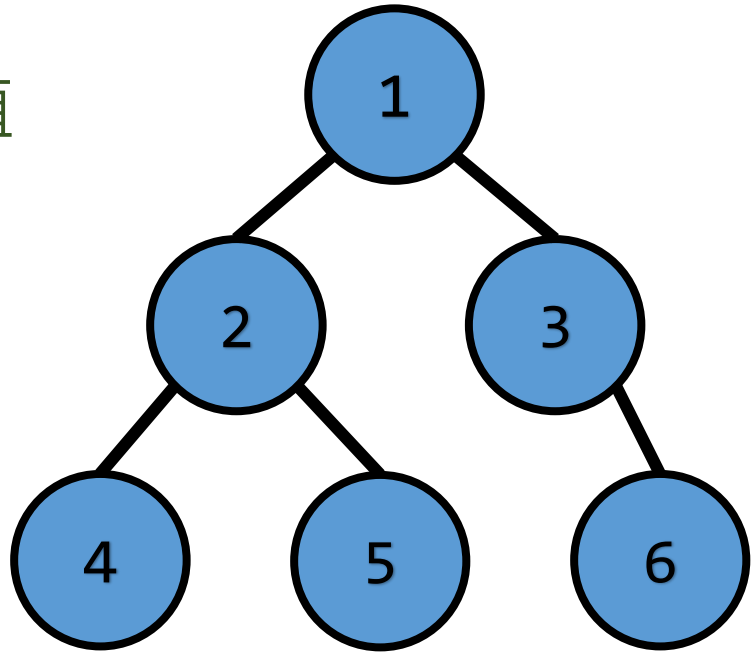
- 中文：二元樹
- 每個節點最多只有兩個子節點
- 第k層最多有2^k個節點
- 一棵深度為k的二元樹最多有$2^{k+1}-1$個節點

```
1
2  struct Node{
3      int _data;
4      Node *_lchild, *_rchild;
5  };
```

Binary tree

- 遍歷：前序、中序、後序
- 差別：什麼時候印出一個node的值
- 前序

```
2  void dfs(Node* node){
3      node->printData();
4      if(node->_lchild != NULL)
5          dfs(node->_lchild);
6      if(node->_rchild != NULL)
7          dfs(node->_rchild);
8  }
```

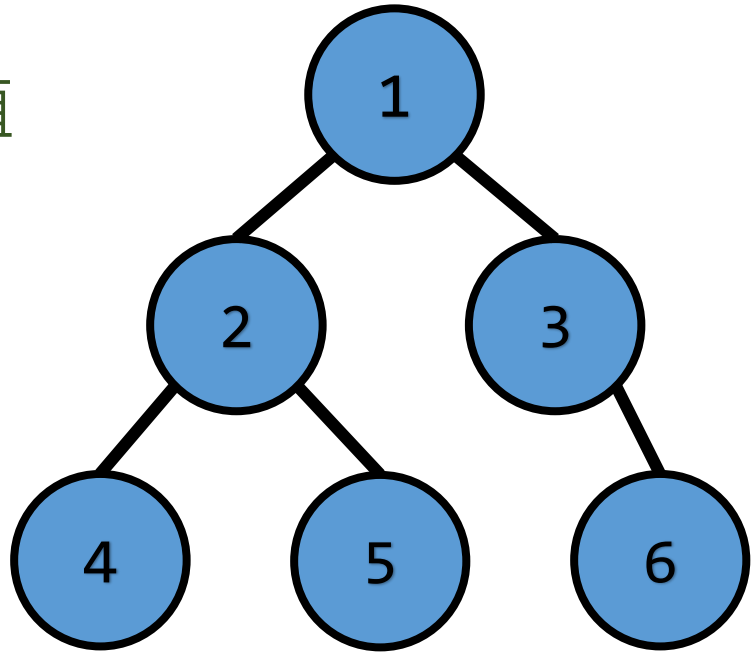


- output: 1 2 4 5 3 6

Binary tree

- 遍歷：前序、中序、後序
- 差別：什麼時候印出一個node的值
- 中序

```
2  void dfs(Node* node){
3      if(node->_lchild != NULL)
4          dfs(node->_lchild);
5      node->printData();
6      if(node->_rchild != NULL)
7          dfs(node->_rchild);
8  }
```

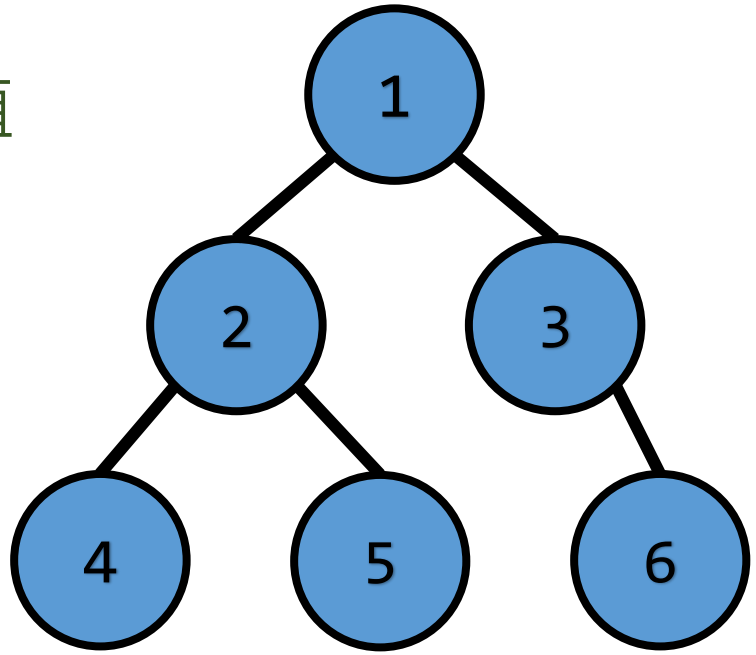


- output: 4 2 5 1 3 6

Binary tree

- 遍歷：前序、中序、後序
- 差別：什麼時候印出一個node的值
- 後序

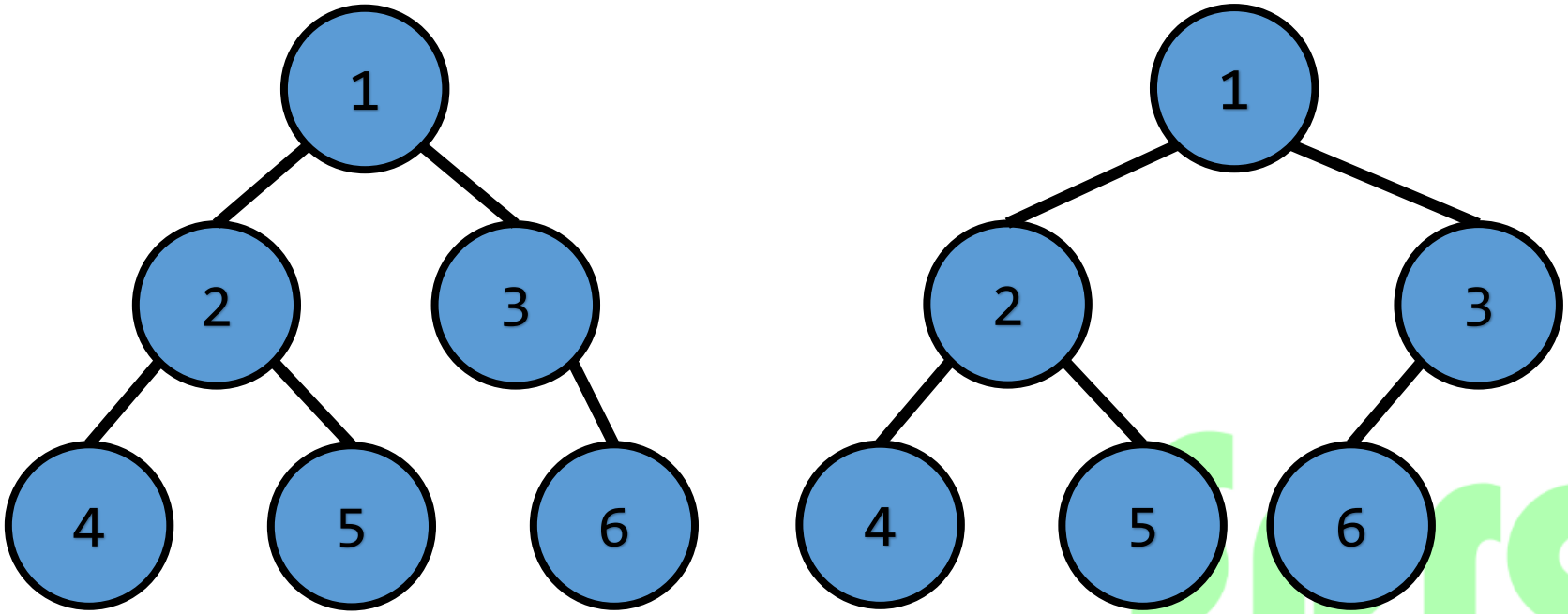
```
2  void dfs(Node* node){
3      if(node->_lchild != NULL)
4          dfs(node->_lchild);
5      if(node->_rchild != NULL)
6          dfs(node->_rchild);
7      node->printData();
8  }
```



- output: 4 5 2 6 3 1

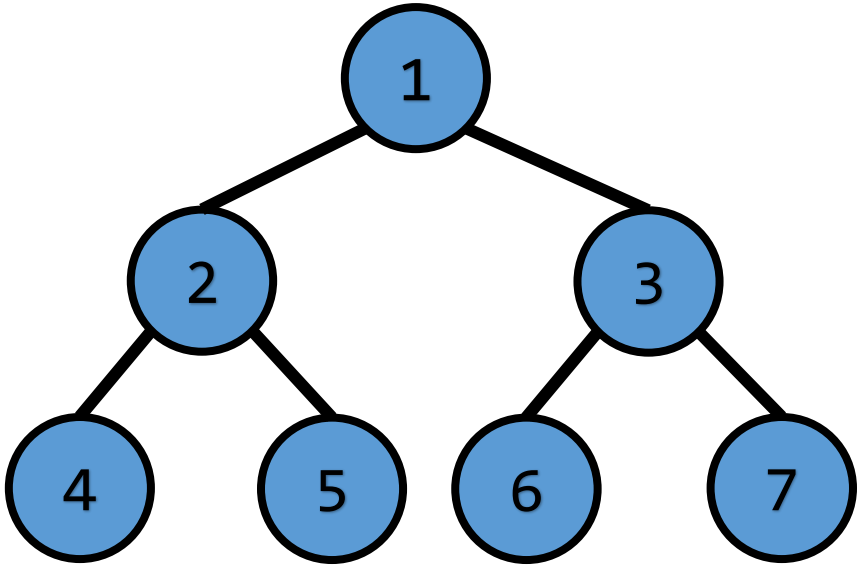
Complete binary tree

- 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



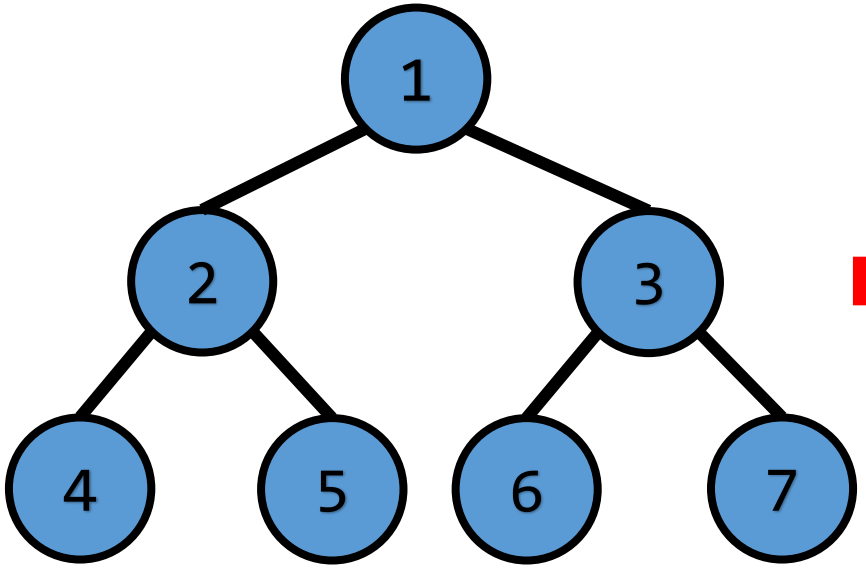
Complete binary tree

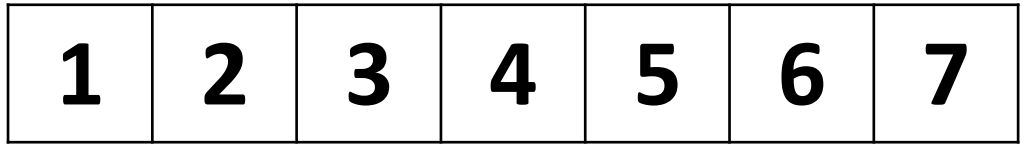
- Complete binary tree 的儲存方式
- 編號為k的兩個child編號分別為2k和2k+1
- 編號為k的parent編號為[k/2]
- 一個有n個元素的complete binary tree的深度約為log(n)



Complete binary tree

- 於是就可以用陣列存了！





- 想一想，為什麼一般的binary tree不適合用陣列存呢？