

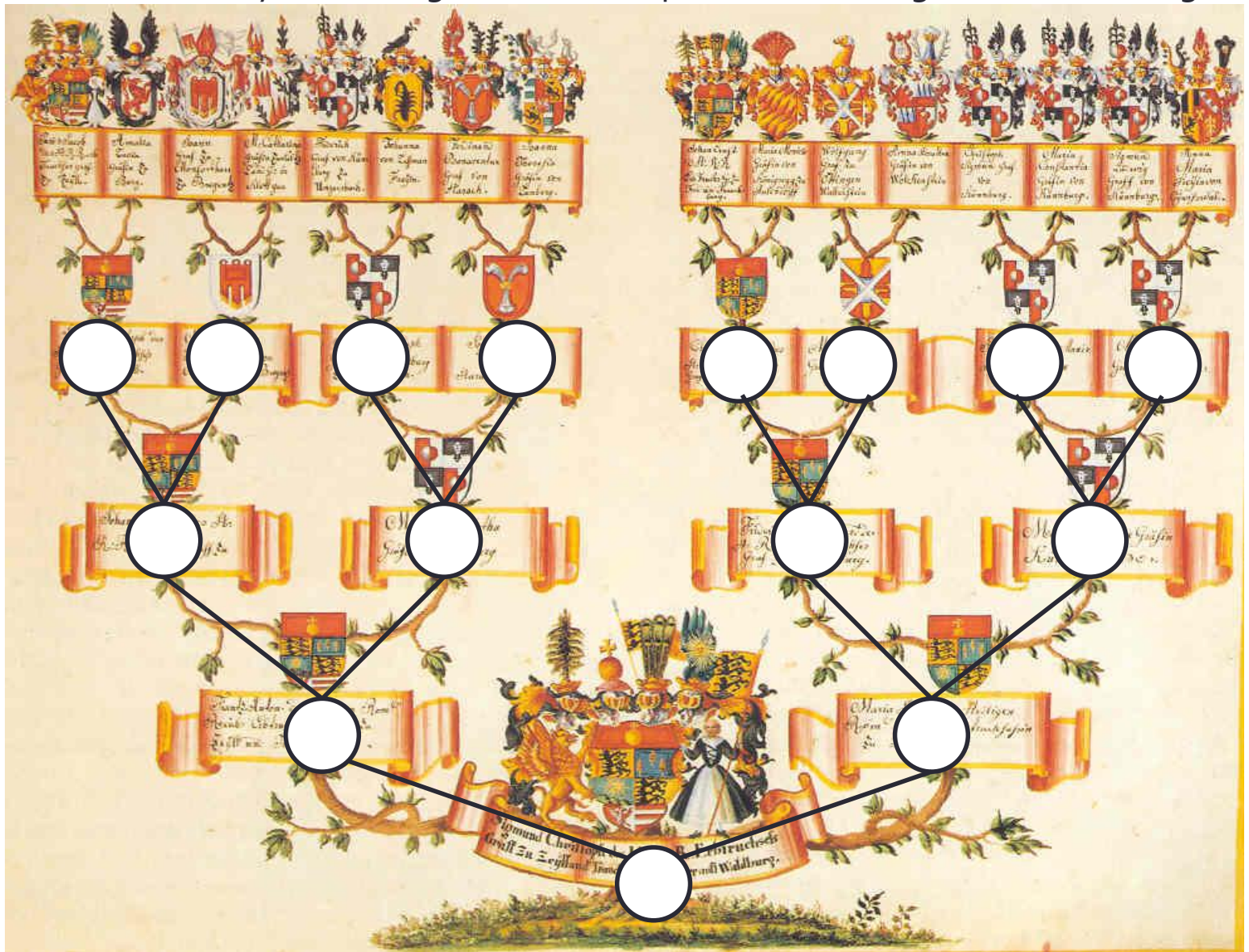
樹

Michael Tsai

2012/3/20

樹

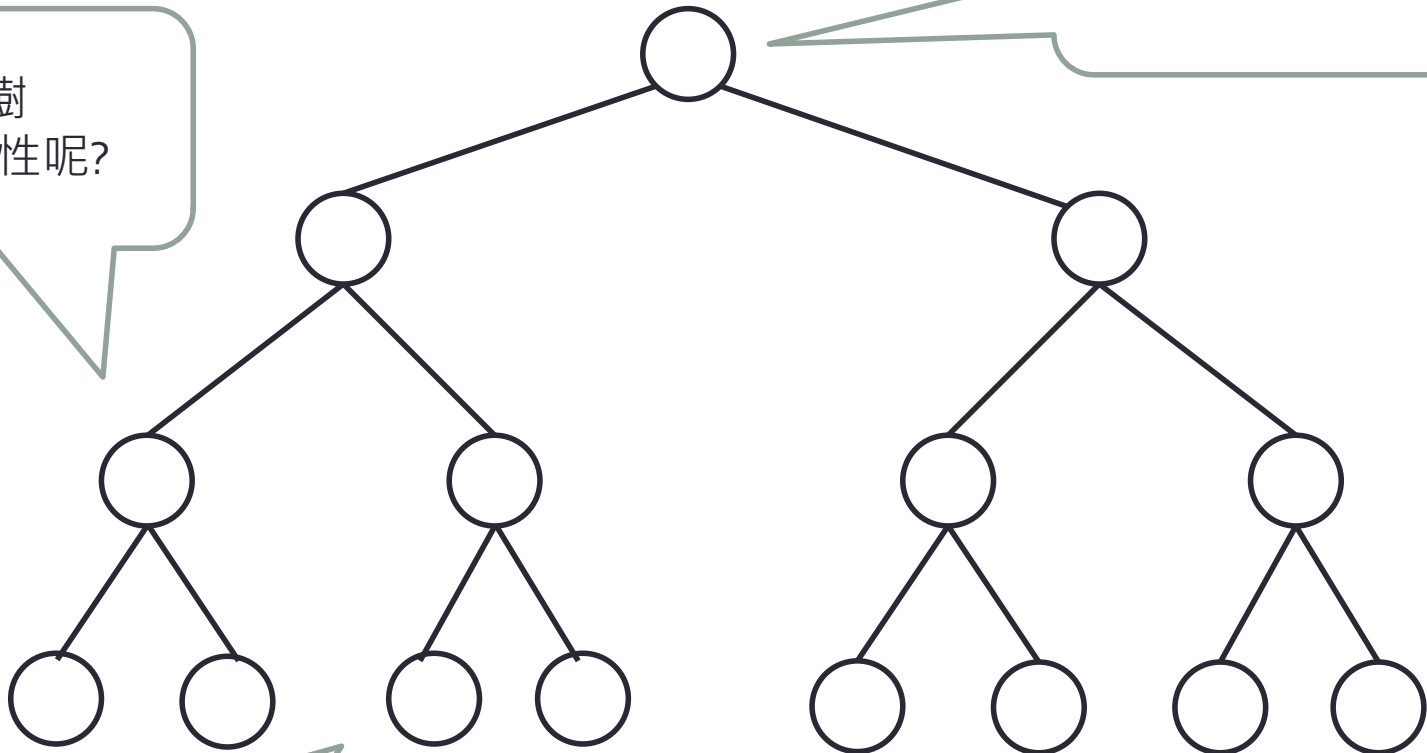
The family tree of Sigmund Christoph von Waldburg-Zeil-Trauchburg



在CS的世界裡，通常我們都把樹畫顛倒

樹根在上面

一棵樹
有什麼特性呢?



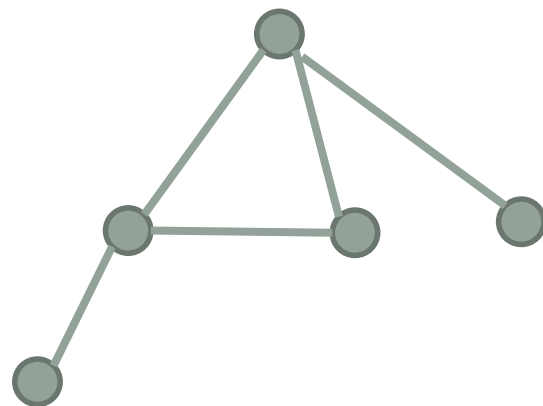
非線性(nonlinear), 具階層式的
(hierarchical)方式表示資料

葉子們在下面

樹的定義

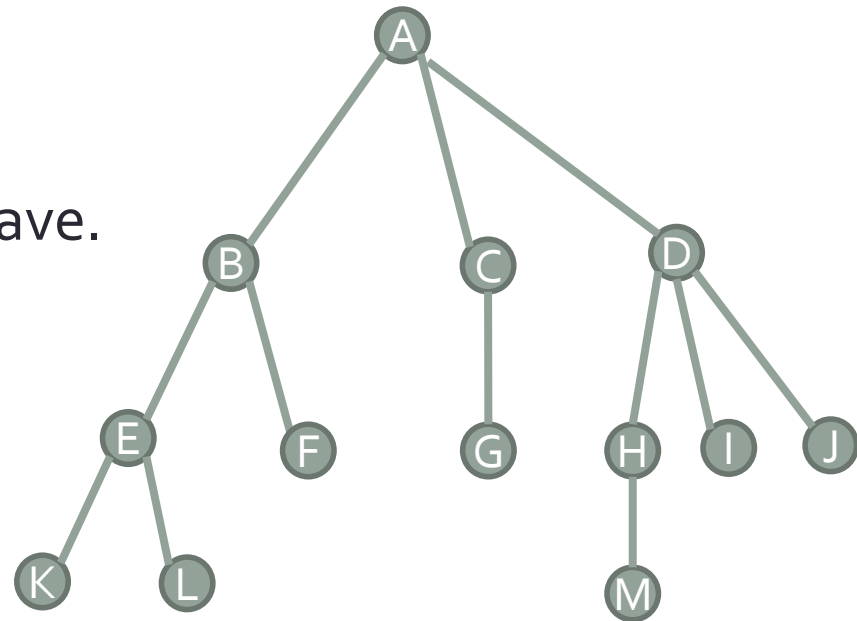
- Definition: A **tree** is a finite set of one or more nodes such that
- (1) There is a specially designated node called the root.
- (2) The remaining nodes are partitioned into $n \geq 0$ disjoint sets T_1, T_2, \dots, T_n , where each of these sets is a tree.
- (3) T_1, T_2, \dots, T_n are called the subtrees of the root.

- 注意以上為遞迴定義
- 一個node沒有子樹的話, 是不是樹?
- 沒有node是不是樹?
- 右邊的是不是樹?
- 違反了什麼規則?



樹的字典

- **Root**
- **Node/Edge (branch)**
- **Degree (of a node):**
how many subtree does it have.
- **Leaf/Terminal node:**
its degree=0
- **Parent/Children**
- **Siblings**
they have the same parent.
- **Ancestors/Descendants**



樹的字典

- **Level/depth (of a node):**

從root走到那邊需要經過的branch數目. (root在level 0)

- **Height (of a tree):**

tree共有幾個level.

(注意Karumanchi裡面的定義略有不同)

- **Size (of a tree):**

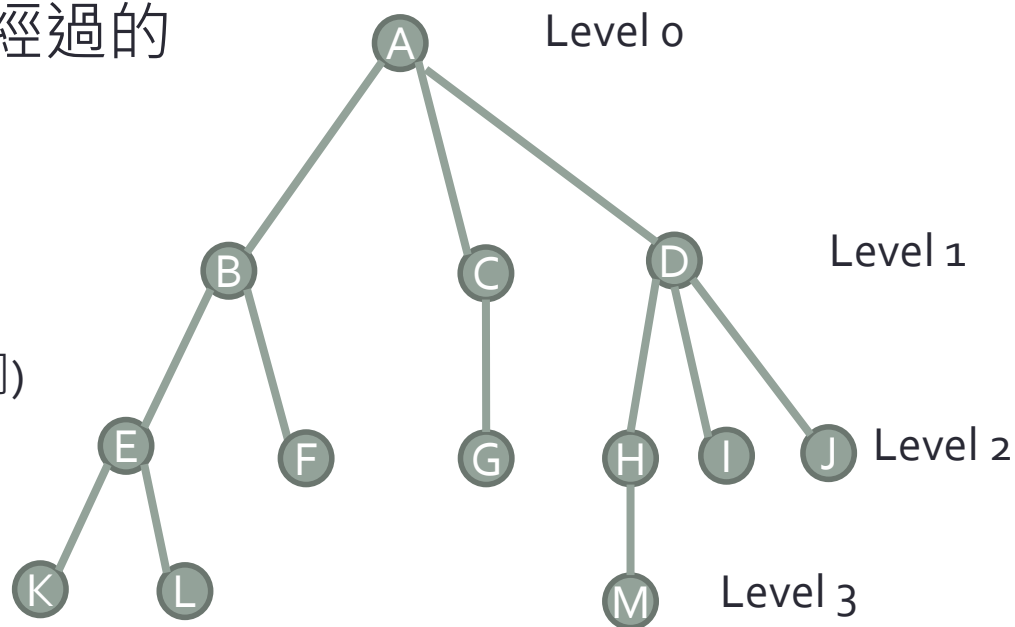
tree裡面總共有幾個node

- **Weight (of a tree):**

tree總共有幾個leaf

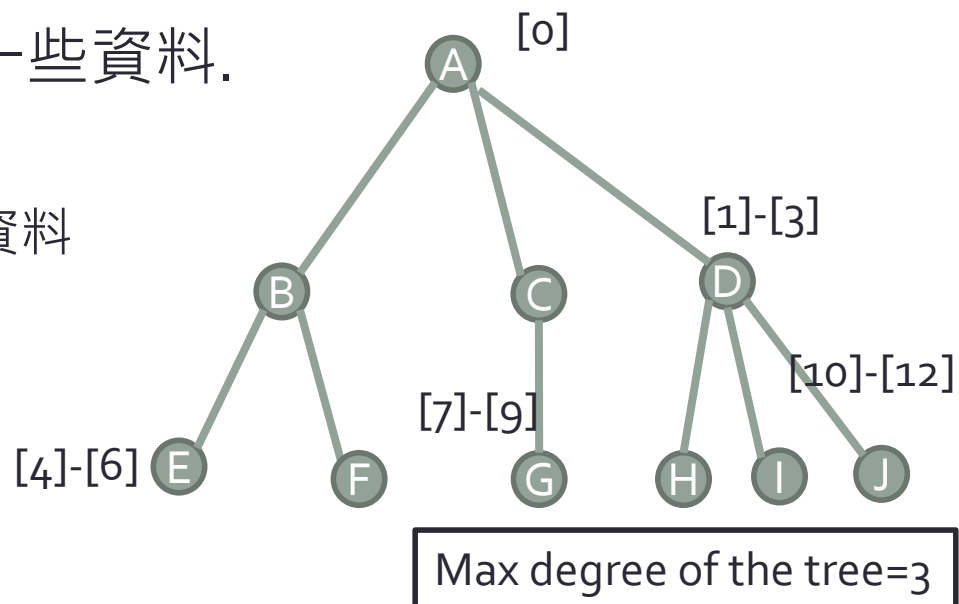
- **Degree (of a tree):**

tree裡面degree最大的node的degree



怎麼在記憶體裡面記一棵樹呢? Array法

- 存什麼? 通常每個node會有一些資料.
- Array法
 - 依照level依序在array裡面儲存資料
- 在array裡面:
- 怎麼找某個node的parent?
- 怎麼找某個node的children?



| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| A | B | C | D | E | F | | G | | | H | I | J |

怎麼在記憶體裡面記一棵樹呢? Array法

- 假設degree=3
(每個node最多有三個children)

- 怎麼找某個node的parent?

- 觀察:

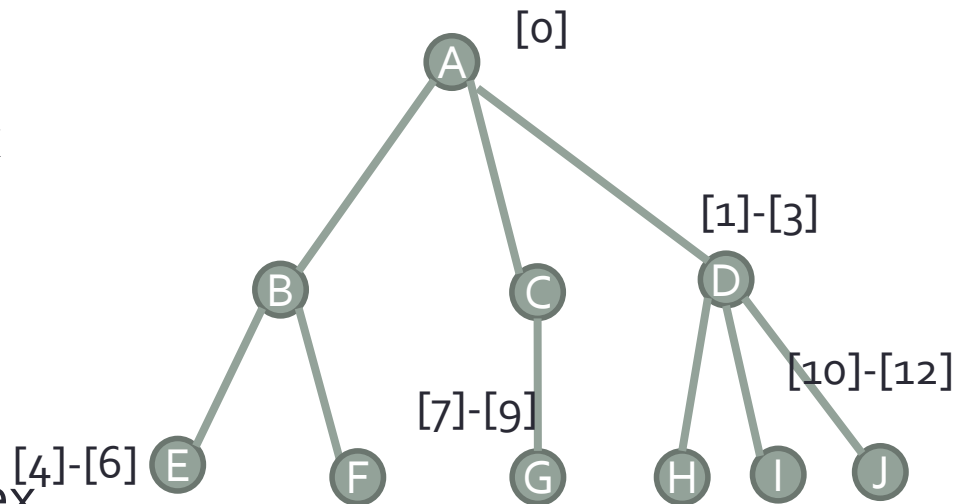
Index為*i*的node, 其parent之index
為 $\lfloor (i - 1) / d \rfloor$

- 怎麼找某個node的children

- 觀察:

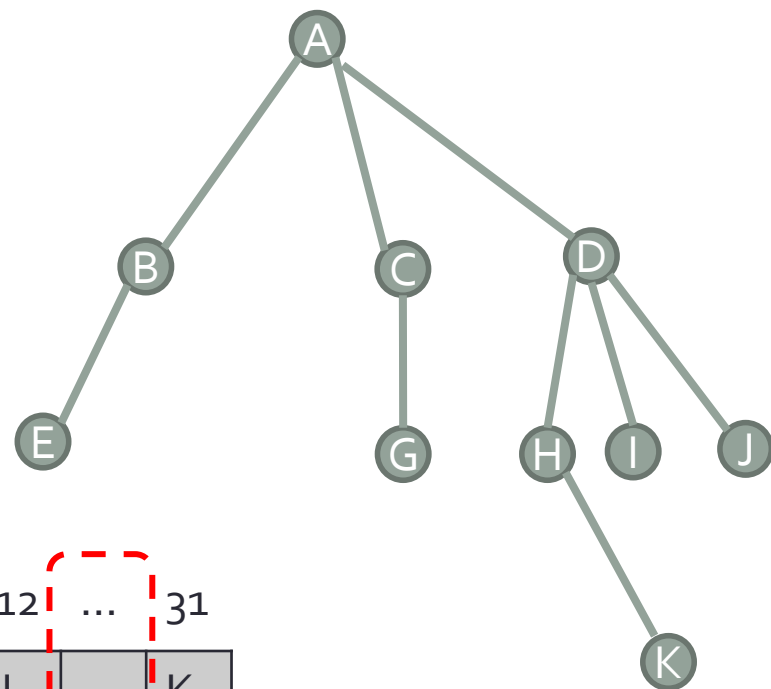
Index為*i*的node, 其children之index
為 $di + 1 \sim di + d$

- 想想看為什麼是這樣?



怎麼在記憶體裡面記一棵樹呢? Array法

- 壞處是什麼?
- 中間如果有許多沒有連接的地方,
- 許多node有少於d個children
- 那array中間會有很多空白



浪費很大!

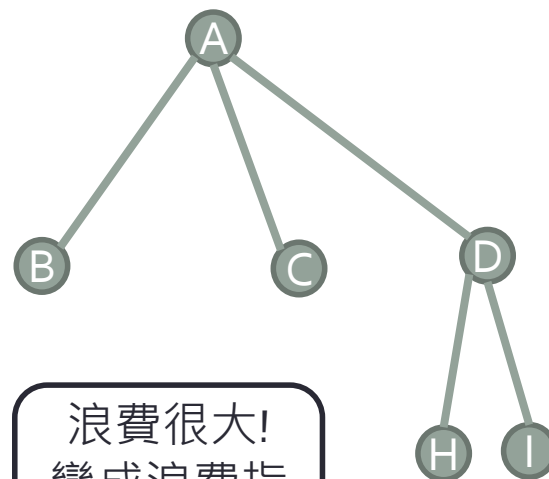
Max degree of the tree=3

怎麼在記憶體裡面記一棵樹呢?

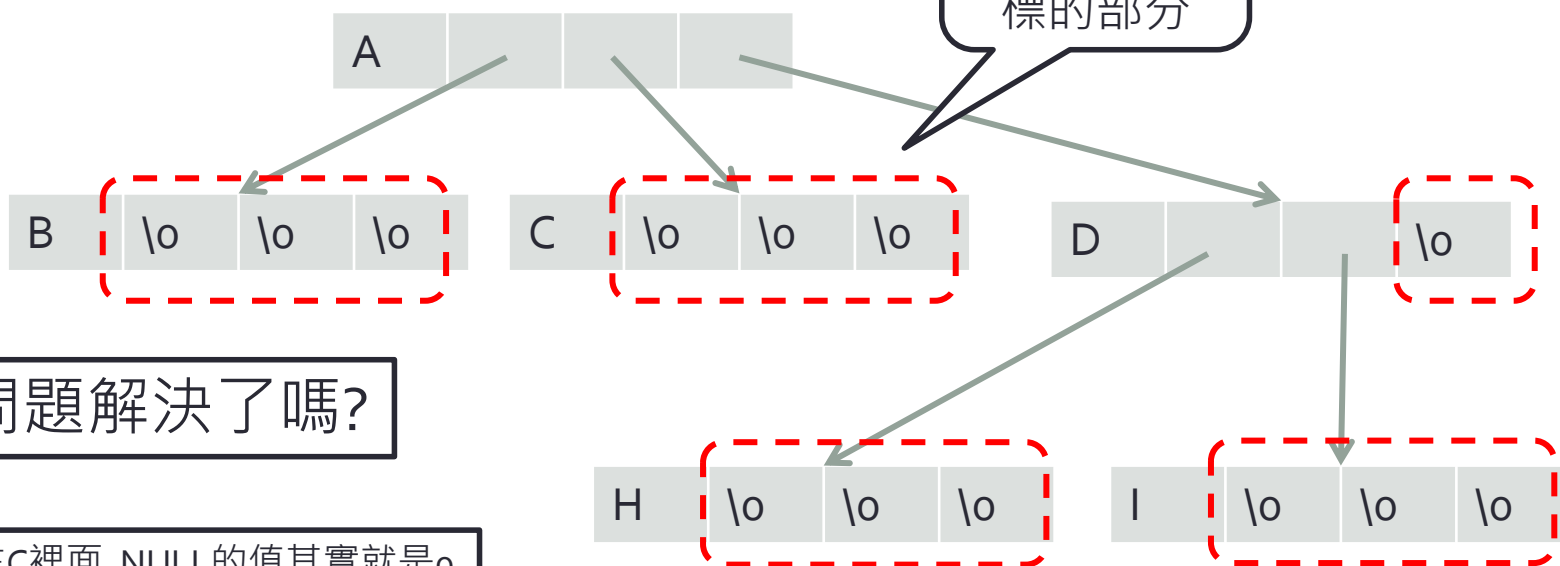
Linked Structure法

- 假設degree=3

```
struct TreeNode{
    char data;
    struct TreeNode* child1;
    struct TreeNode* child2;
    struct TreeNode* child3;
};
```



浪費很大!
變成浪費指
標的部分



這樣問題解決了嗎?

\0代表NULL. 在C裡面, NULL的值其實就是0

怎麼在記憶體裡面記一棵樹呢?

Linked Structure法

Data

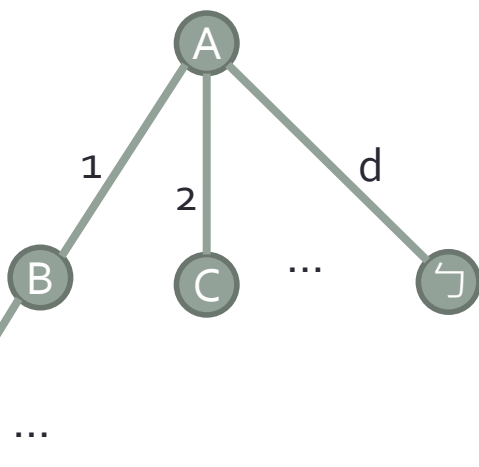
child 1

child 2

child 3

...

child k



- 假設degree of tree = d , 總共有 n 個nodes
- 有多少個 child欄位是NULL?
- 總共有 nd 個欄位
- 但是整棵樹有幾個branch?
- $n - 1$ 個
- $nd - (n - 1) = n(d - 1) + 1$

越小越好!

浪費很大!
變成浪費指
標的部分

左小孩-右兄弟姊妹 表示法

- Left child-right sibling representation

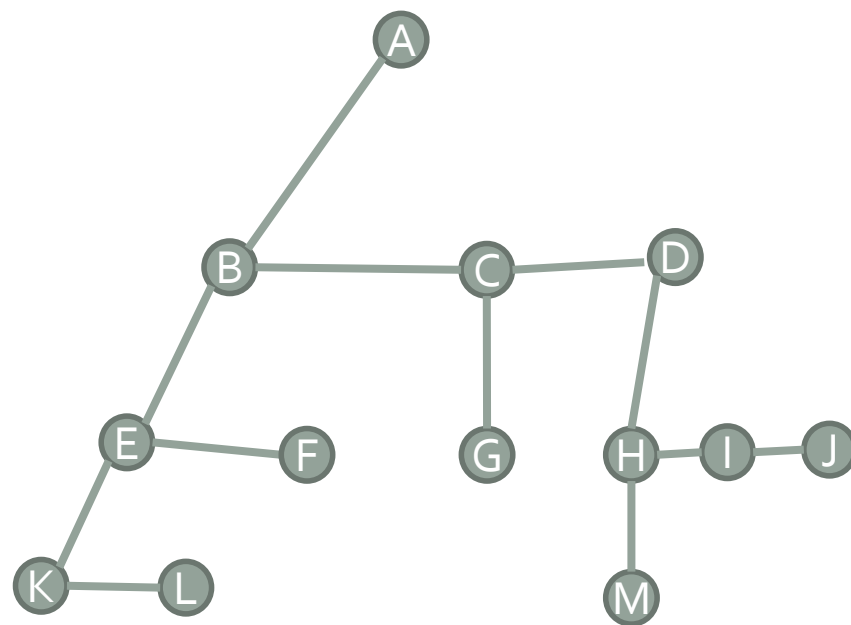
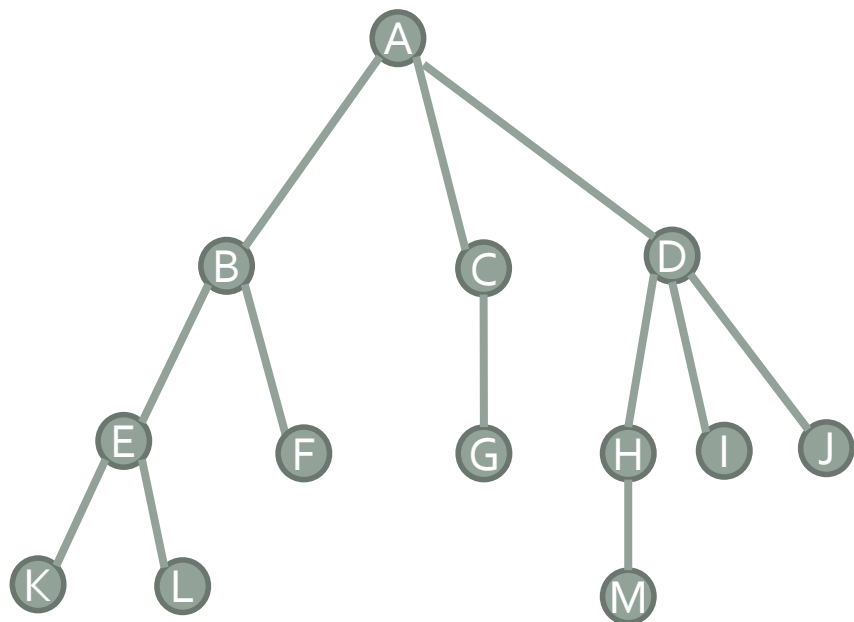
Data

left child

Right sibling

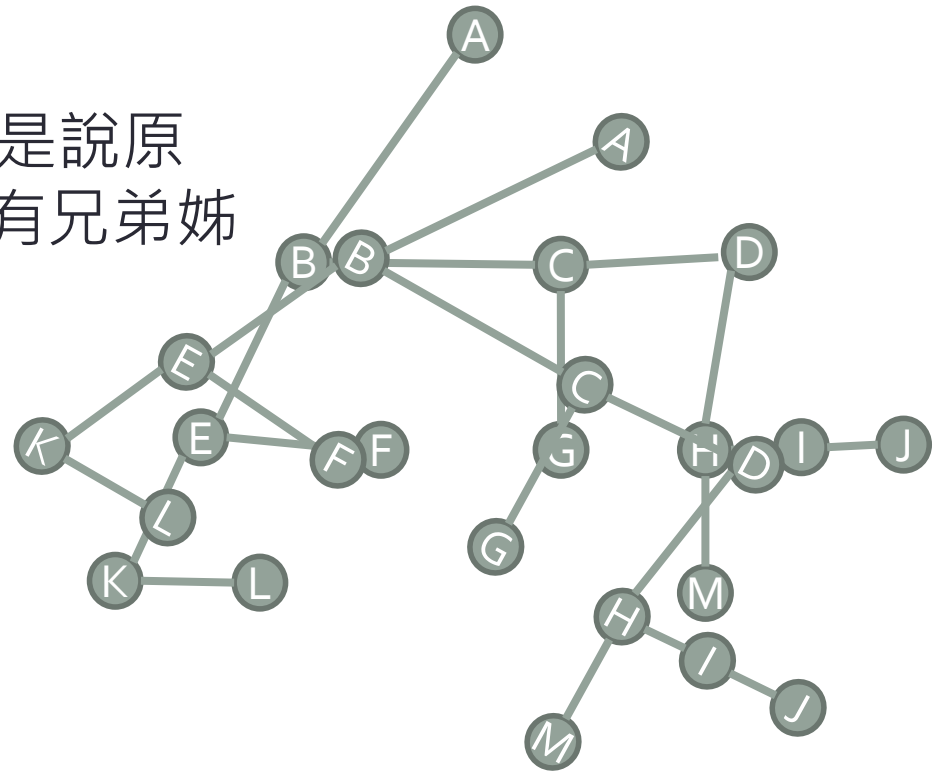
- 觀察:
 1. 每個node最多只有一個最左邊的child (是廢話)
 2. 每個node也最多只有一個最靠近他的右邊的sibling (也是廢話)

來畫一下 怎麼用LC-RS表示這棵樹?



左小孩-右兄弟姊妹 樹

- 可以變成 degree-two tree
- 也就是說, 是一種把普通的樹轉成 degree-two 樹的方法
- Root 沒有右邊的 child (也就是說原本的 LC-RS 樹裡面 root 不會有兄弟姊妹-廢話)



Binary Tree

- Definition: A **binary tree** is a finite set of nodes that is either **empty** or **consists of a root and two disjoint binary trees** called the left subtree and the right subtree.
- 注意: 可以是沒有node
- 比較: 一般tree不可以沒有node
- 注意: children在左邊或右邊是不一樣的 (有順序)
- 比較: 一般tree的children順序沒有差

一些證明

- 1. 在level i 的node數目最多為 $2^i, i \geq 1$ (root在level 0)
- 證明: 用歸納法
- $i=0$ 時, 為root那一層, 所以只有一個node, 也就是最多有 $2^0 = 1$ 個node. (成立)
- 假設 $i=k-1$ 的時候成立 \rightarrow level $k-1$ 最多有 2^{k-1} 個node
- 那麼 $i=k$ 的時候, 最多有幾個node?
- 因為是binary tree, 所以每個node最多有兩個children
- 因此最多有 $2^{k-1+1} = 2^k$ node (得證)

兩些證明(誤)

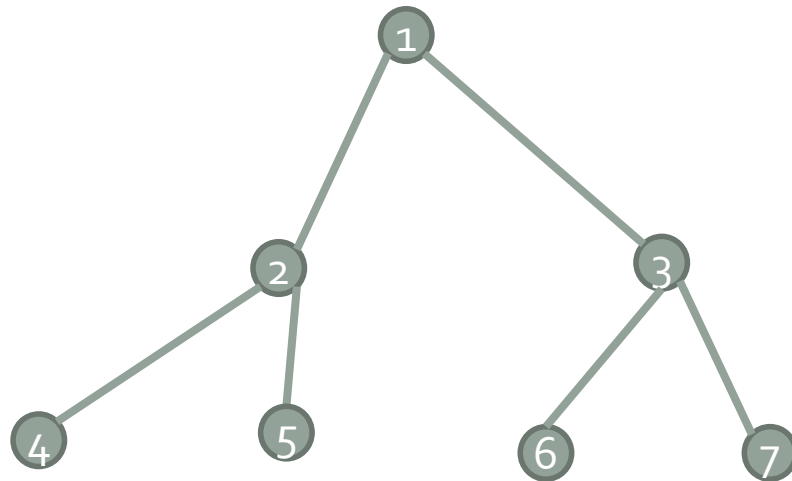
- 2. 一棵height為k的binary tree, 最多有 $2^k - 1$ 個node, $k \geq 1$.
- 證明:
- 利用1的結果
- 則總共node數目最多為
- $\sum_0^{k-1} 2^i = \frac{2^k - 1}{2 - 1} = 2^k - 1$. 喔耶.

三些證明(誤)

- 3. 對於任何不是空的binary tree, 假設 n_0 為leaf node數目, n_2 為degree 2的node數目, 則 $n_0 = n_2 + 1$.
- 證明:
- 假設 n 為所有node數目, n_1 為degree 1的node數目,
- 則 $n = n_0 + n_1 + n_2$. (1)
- 假設 B 為branch的數目, 則 $B = n_1 + 2n_2$. (2)
- 而且 $n = B + 1$ (3). (只有root沒有往上連到parent的branch, 其他的node正好每個人一個)
- (2)代入(3)得 $n = n_1 + 2n_2 + 1$ (4)
- (4)減(1)得 $n_0 = n_2 + 1$. 喔耶.

Full binary tree

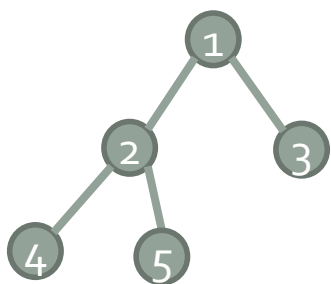
- Definition: a **full binary tree** of depth k is a binary tree of depth k having $2^k - 1$ nodes, $k \geq 1$.
- 也就是說depth k 的樹裡面最多node數目的(滿了)
- 除了leaf每個node都有兩個children



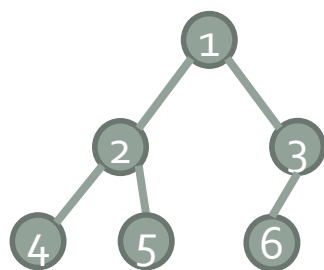
depth=3的full binary tree

Complete binary tree

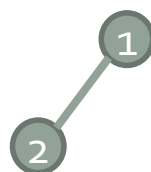
- Definition: A binary tree with n nodes and depth k is complete iff its nodes correspond to the nodes numbered from 1 to n in the full binary tree of depth k .
- 也就是說, 所有在 level $k-1$ 和 $k-2$ (最底和倒數第二層) 的 leaf 都沒有“缺號”。



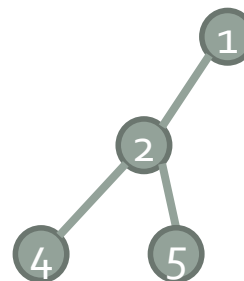
Yes



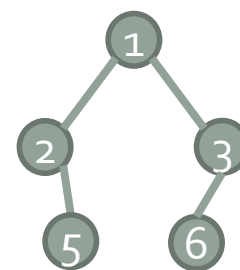
Yes



Yes



No



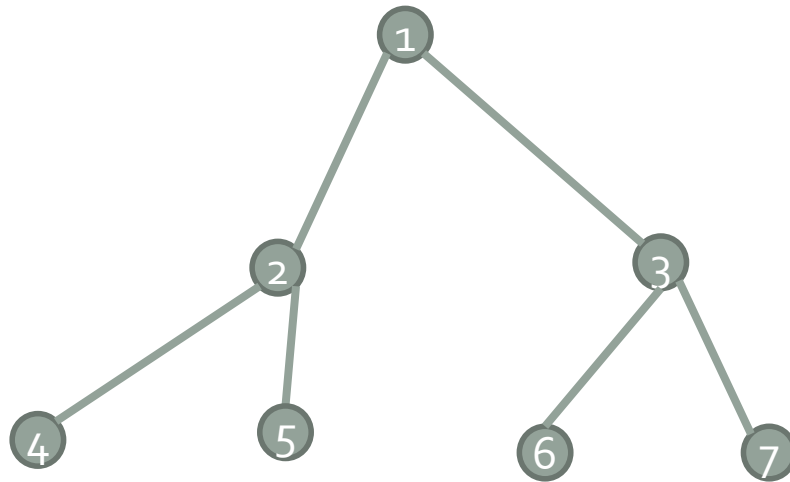
No

Complete binary tree的高度

- 如果一個complete binary tree有 n 個node, 那麼樹的高度為?
- Hint:高度為 k 的full binary tree有 $2^k - 1$ 個node
- Hint: 在最底層總共有多少個node?
- 答:
- 最少有幾個node?
- $2^k - 1 - 2^{k-1} + 1 = 2^{k-1}$
- 所以node個數可能為 $2^{k-1} \sim 2^k - 1$
- $\lceil \log_2(n + 1) \rceil$ 在各種情況都可以得到 k

Binary Tree Traversal

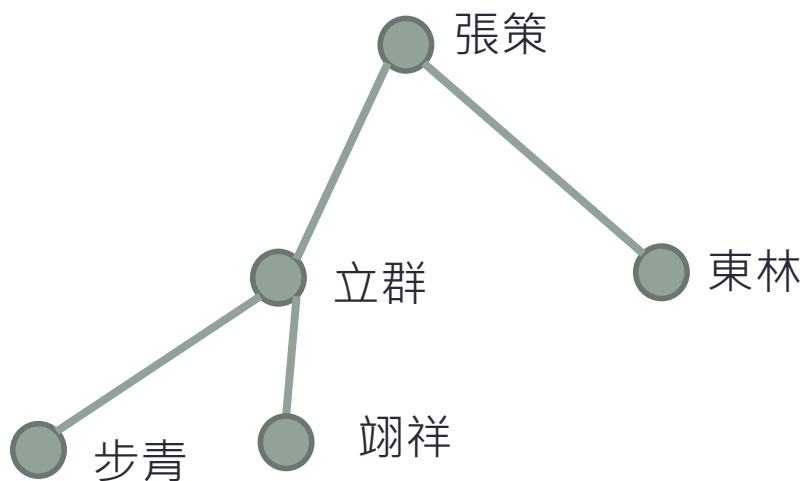
- 有一棵binary tree後, 我們要怎麼把樹的每一個node都走一遍呢?



- 到某一個node的時候, 有三件事情可以做:
 1. 走左邊的child那邊的node們 (用L表示Left branch)
 2. 走右邊的child那邊的node們 (用R表示Right branch)
 3. 處理自己這個node的資料(用V表示Visit)

Binary Tree Traversal

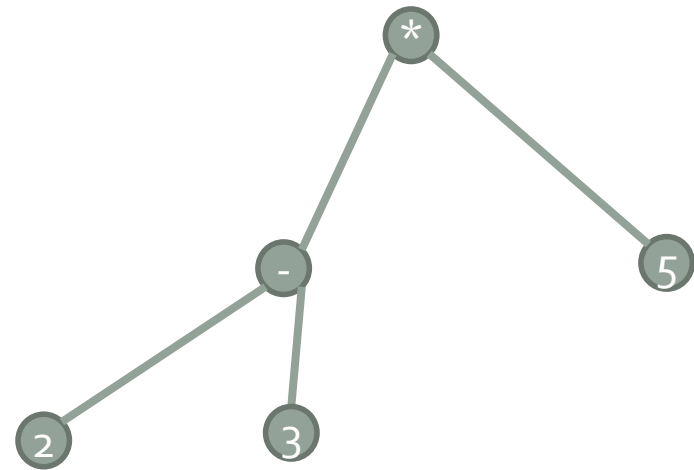
- 如果L一定要在R之前, 那麼有三種
- VLR: preorder
- LVR: inorder
- LRV: postorder



- 請同學說明preorder, inorder, postorder traversal分別順序是如何😊

Binary tree with arithmetic expression

- 每個arithmetic expression都可以建立一個expression tree
- Preorder \rightarrow prefix
- Inorder \rightarrow infix
- Postorder \rightarrow postfix
- 請同學試試看☺
- (亂寫一個expression看看)
- Reading Assignment: Karumanchi 6.9 – How to build a expression tree? (和之前的功課很像!)



Binary Tree Traversal

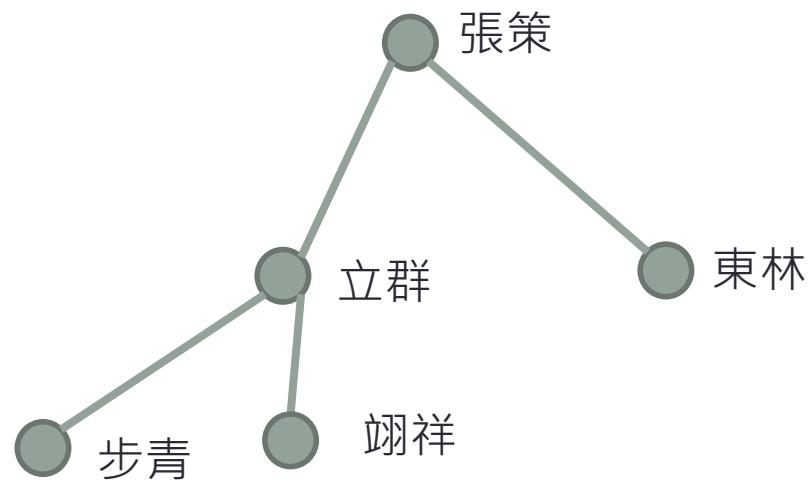
- 可以用recursive寫法來做traversal (會很簡潔):

```
void inorder(treePointer ptr) {  
    inorder(ptr->leftChild);  
    visit(ptr);  
    inorder(ptr->rightChild);  
}
```

那如果不要用recursive寫法呢?

- 用Stack

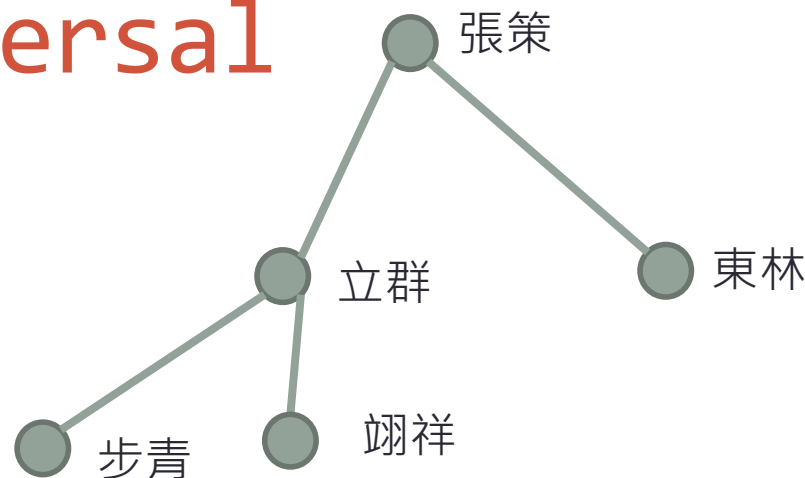
```
for(;;) {  
    for(;node;node=node->leftChild)  
        push(node);  
    node=pop();  
    if (!node) break;  
    printf("%d", node->data);  
    node=node->rightChild;  
}
```



Level-order traversal

- 如果改成用queue呢?

```
add(ptr);  
for(;;) {  
    ptr=delete();  
    if (ptr) {  
        printf("%d", ptr->data);  
        if (ptr->leftChild)  
            add(ptr->leftChild);  
        if (ptr->rightChild)  
            add(ptr->rightChild);  
    } else break;  
}
```

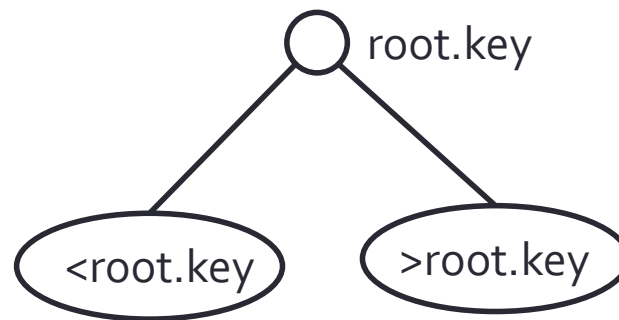


Binary search tree

- 問題: 找系上某個同學的作業成績
- 條件:
- 知道學號 (key)
- 找到學號就可以對應到儲存資料的地方(search)
- 常常有人進來(add node)
- 常常有人出去(delete node)

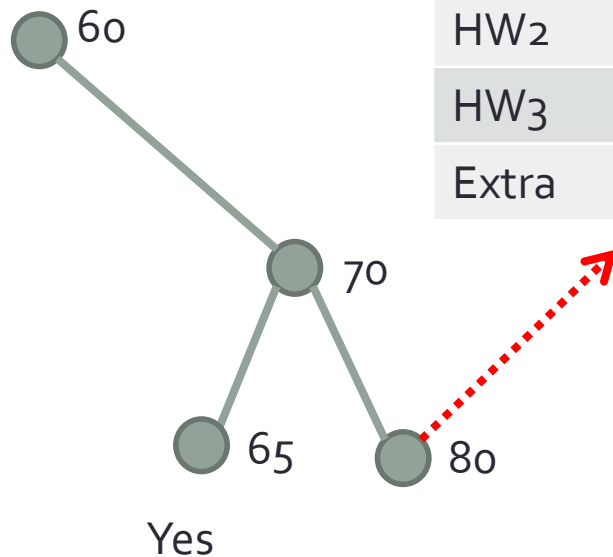
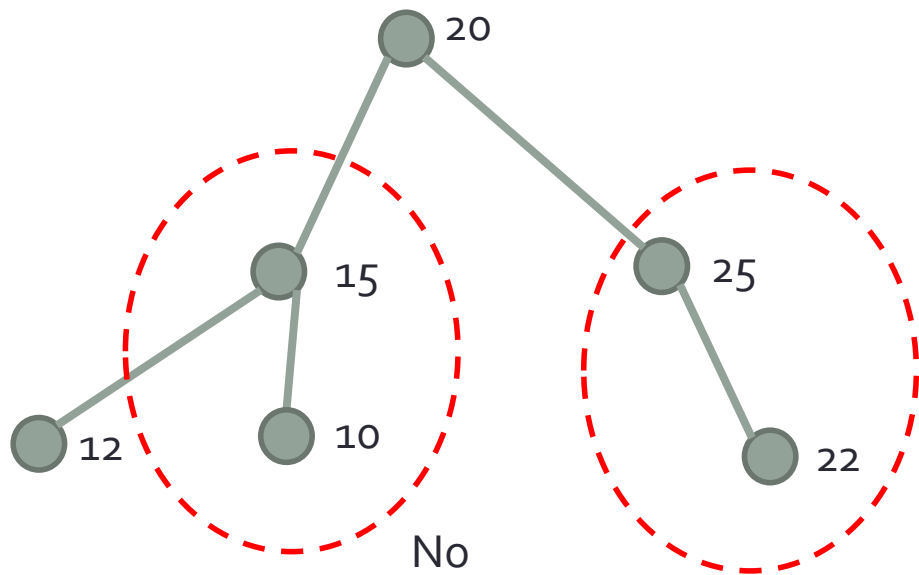
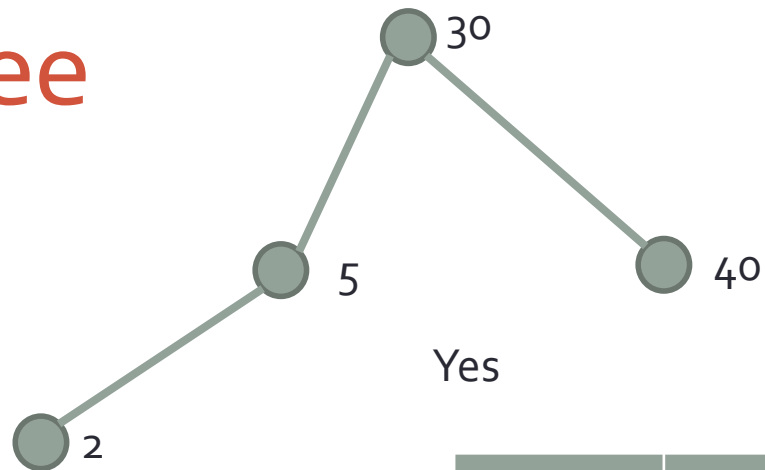
Binary search tree

- Definition: A binary search tree is a binary tree. It may be empty. If it is not empty then it satisfies the following properties:
 - 1. The root has a key.
 - 2. The keys (if any) in the left subtree are smaller than the key in the root
 - 3. The keys (if any) in the right subtree are larger than the key in the root
 - 4. The left and right subtrees are also binary search trees
- (隱藏) All keys are distinct.



Binary search tree

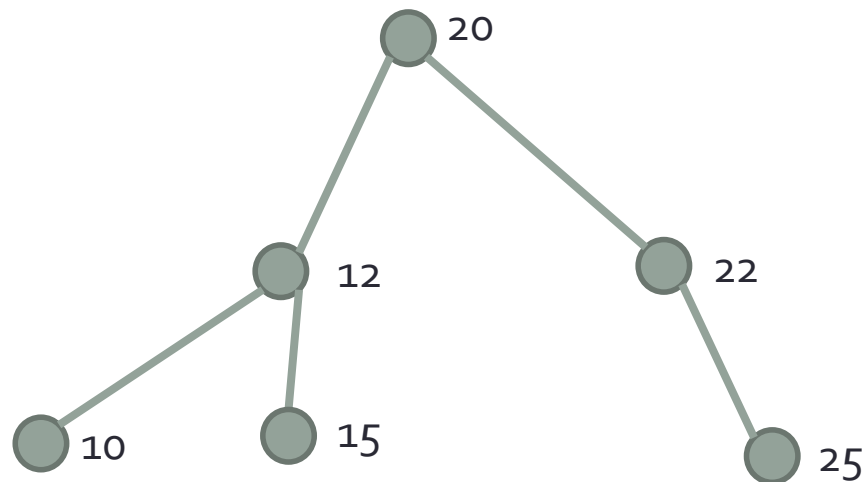
- 這些是不是binary search tree?
- 找到以後要做什麼?



| 80號 | |
|-------|--------|
| HW1 | 65 |
| HW2 | 65 |
| HW3 | 空 |
| Extra | -20000 |

如何尋找?

- 假設要找10

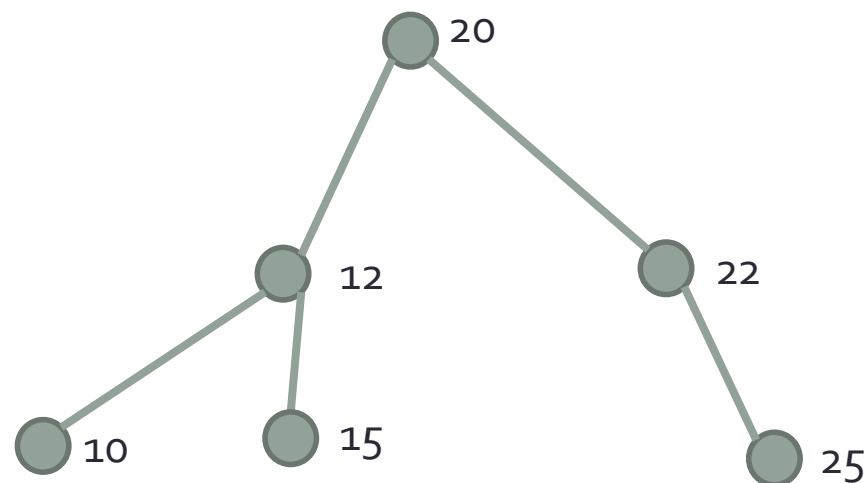


```
struct TreeNode* find(struct TreeNode* root, int data)
{
    if (root==NULL) return NULL;
    if (data==root->data) return root;
    if (data<root->data) return find(root->left, data);
    return find(root->right, k);
}
```

- 簡單. 那time complexity = $O(??)$
- 答: $O(h)$, h : height of the tree.
- Worst case: $O(n)$ Average case: $O(\log_2 n)$

Binary Search Tree的Algorithm常常是以上的型態:
(1)如果key跟現在的node一樣, 那麼就做一些處理後return結果.
(2)如果key比較大或比較小, 分別call自己的分身去處理右邊或左邊的subtree

其他使用方法

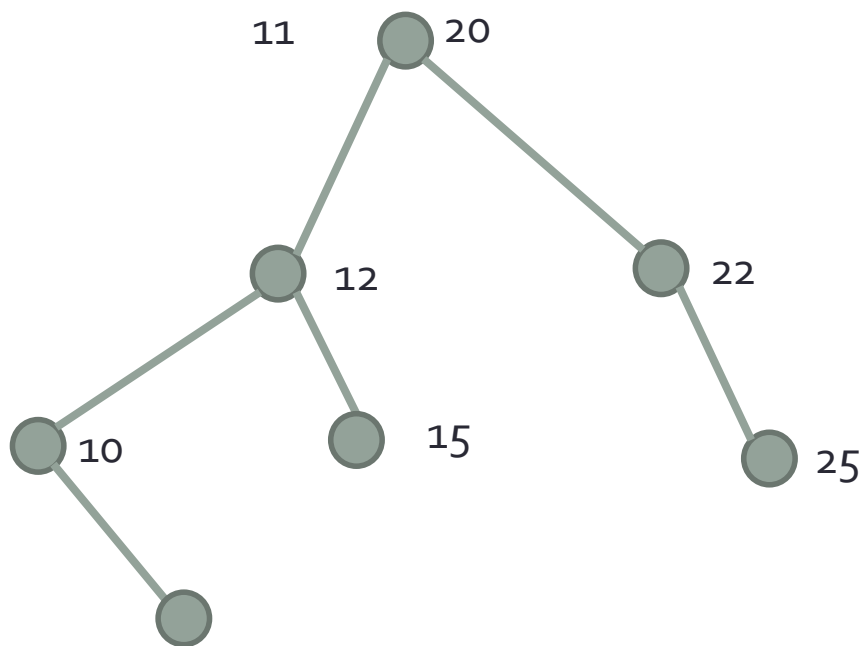


- Q: 怎麼找到Binary Search Tree裡面最大(小)的元素?
- A: 一直往右(左)邊的child走去, 直到碰到NULL為止.
(Karumanchi p.149)

- Q: 怎麼把Binary Search Tree裡面所有的元素依序列出?
- A: 做Binary Search Tree的Inorder Traversal!

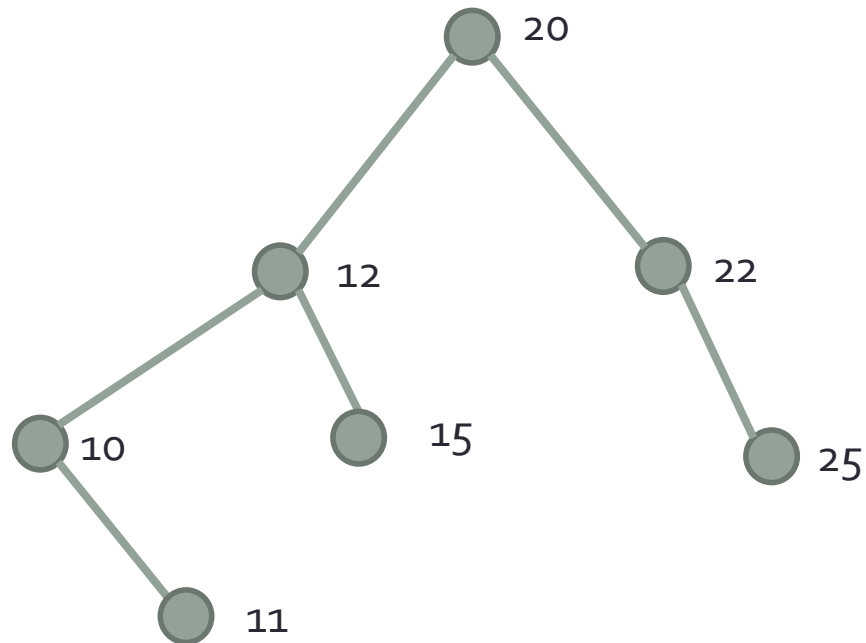
如何插入新的node?

- 先找有沒有一樣的 (隱藏版rule: binary search tree裡面不可以有一樣的key)
- 找不到的話, 插在最後“找不到”的位置
- 插入: 11



如何刪掉一個node?

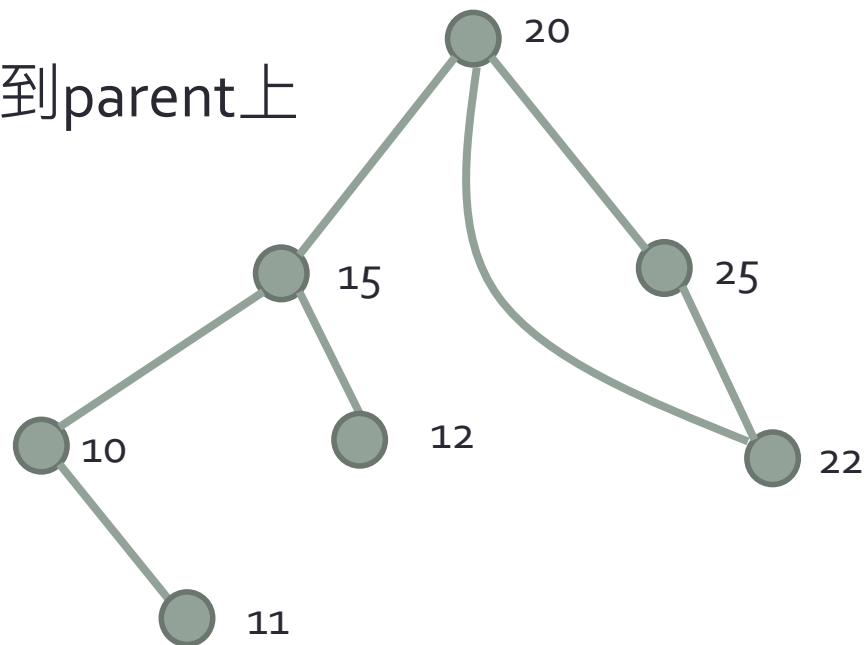
- 首先要先找到那個node
- 接著, 有各種不同情形:
- 如果沒有手($\text{degree}=0$)
- 直接拿掉



如何刪掉一個node?

- 如果只有一隻手 (degree=1)
- 則把那個唯一的child拿上來接到parent上

- 例如: 拿掉25

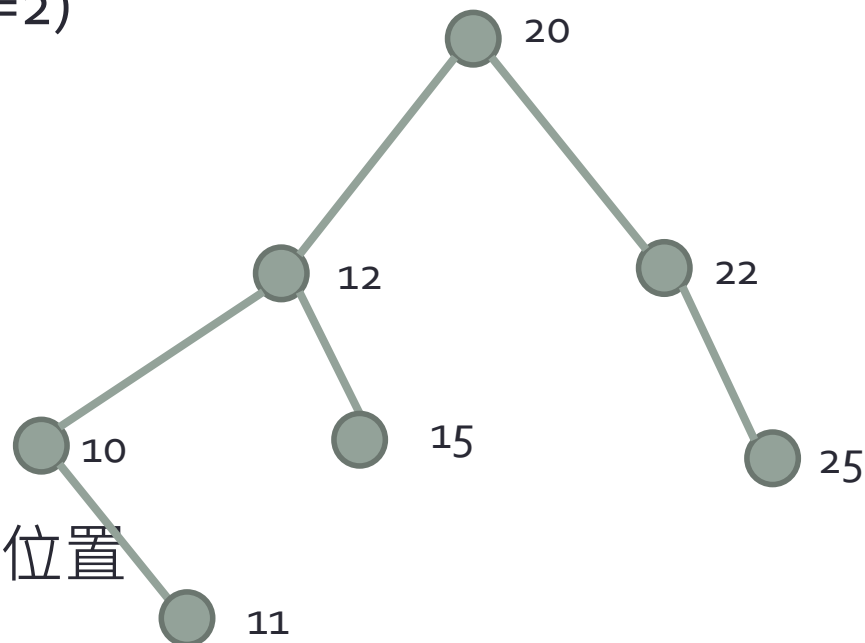


- 問題: 要怎麼接?
- 怎麼記得parent是誰?
- (回傳給return給上一層去設定, slide #37)
- (Karumanchi p.152)

如何刪掉一個node?

- 如果兩手都有東西呢? (degree=2)

- 例如刪掉12
- 找左邊child底下最大的
- (或者是右邊child底下最小的)



- 刪掉它並把它移到原本刪掉的位置
- 問題: 那如果那個最大的底下還有child呢?

答案: 最多只會有一個child, 所以很好處理

```
struct TreeNode *delete(struct TreeNode *root, int data) {
    TreeNode * temp;
    if (root==NULL) {
        printf("error\n");
        return NULL;
    } else if (data < root->data)
        root->left=delete(root->left, data);
    else if (data > root->data)
        root->right=delete(root->right, data);
    else { // data == root->data
        if (root->left && root->right) { //two children
            temp=findmax(root->left);
            root->data=temp->data;
            root->left=delete(root->left, root->data);
        } else { // one child or no child
            temp=root;
            if (root->left==NULL)
                root=root->right;
            if (root->right==NULL)
                root=root->left;
            free(temp);
        }
    }
    return root;
}
```

如果比較大或比較小，交給自己的分身去處理

回傳給上一層，如果把現在這個node殺掉的話，則可以回傳給上一層的pointer接上

如果一樣的話，在這邊處理。

Today's Reading Assignments

本次的reading assignment重要性依序為:

1. Karumanchi 6.1-6.7, 6.11
2. Problem [6-7], [10-11], 14, 30, 31
3. Karumanchi 6.9
4. Problem [49-52, 59]