

C++資料結構與程式設計

樹狀結構(*Tree*)

NTU CSIE

大綱

樹 (Tree)

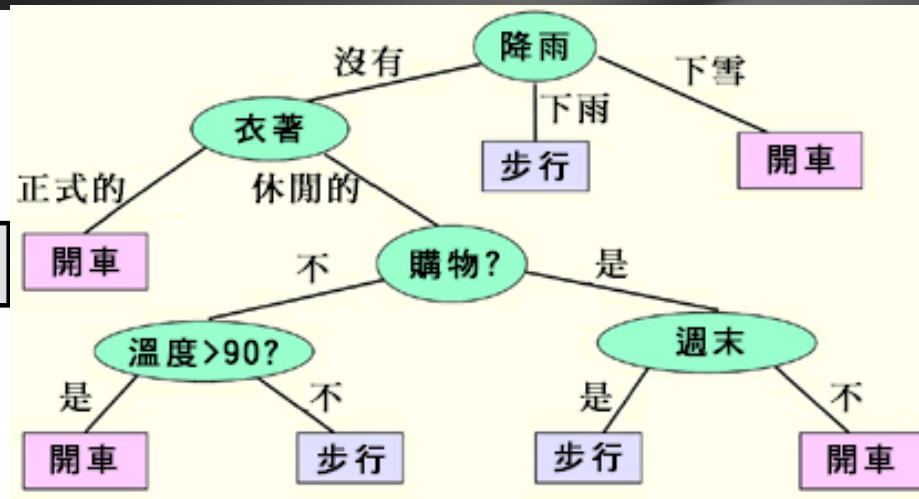
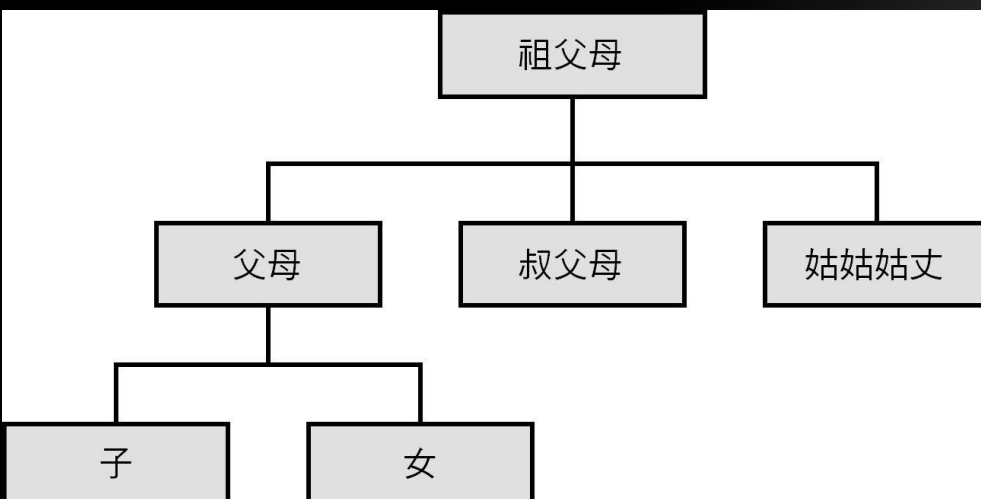
二元樹 (Binary Tree)

二元搜尋樹 (Binary Search Tree)

樹

「樹」(Tree)

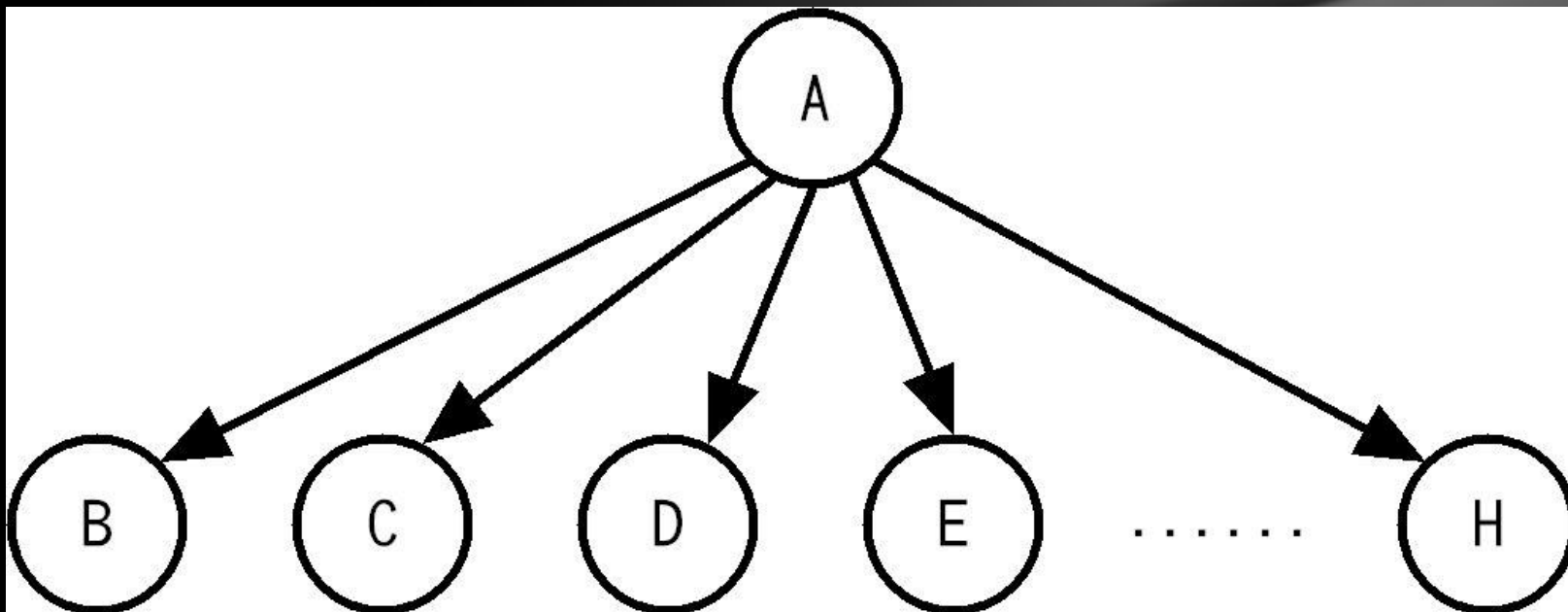
- 是一種模擬現實生活中樹幹和樹枝的資料結構，屬於一種階層架構的非線性資料結構，例如：家族族譜, 決策模型



樹的基本術語

樹的樹根稱為「**根節點**」(Root)，在根節點之下是樹的樹枝，擁有0到n個「**子節點**」(Children)，即樹的「**分支**」(Branch)

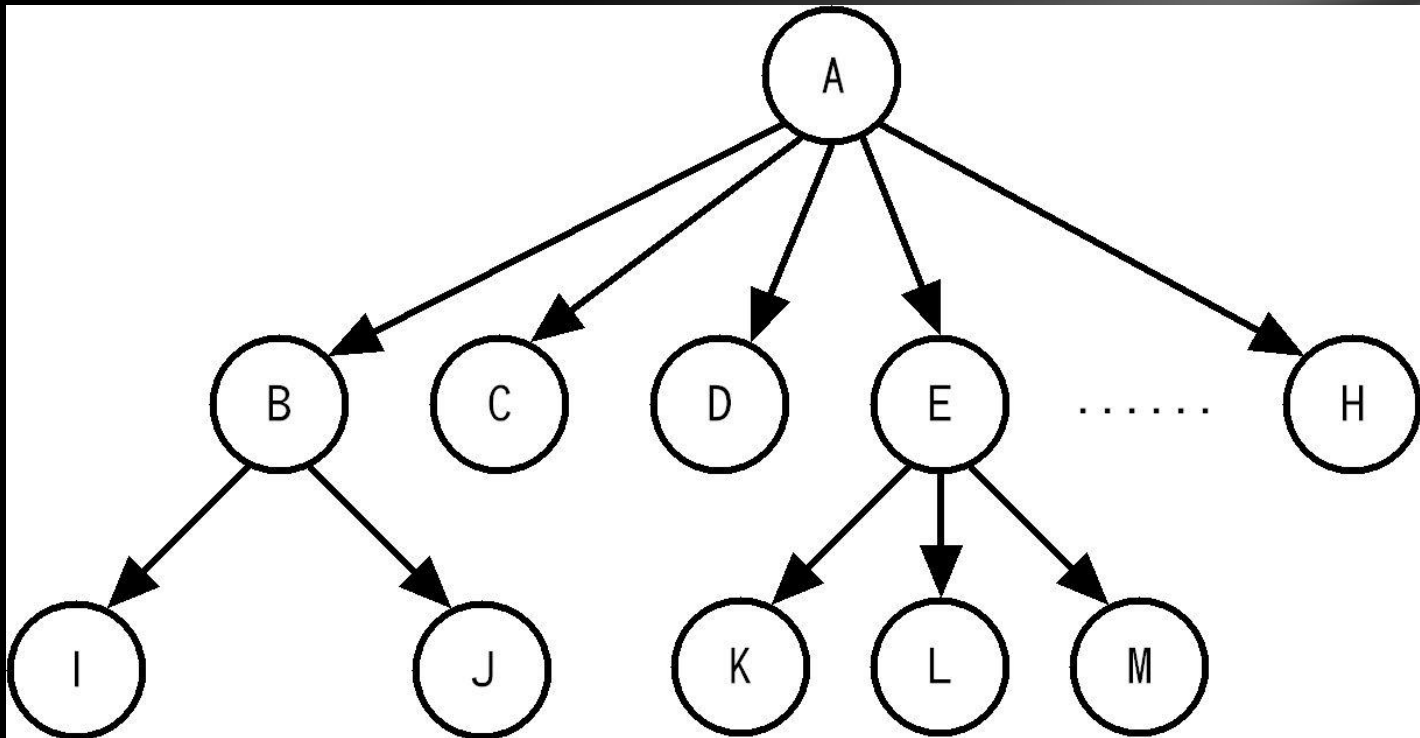
- **節點A**是樹的**根節點**
- **B、C、D...H**是節點**A**的**子節點**



樹的基本術語

在樹枝下還可以擁有下一層樹枝，**I**和**J**是**B**的子節點，**K**、**L**和**M**是**E**的子節點，節點**B**是**I**和**J**的「父節點」(Parent)，節點**E**是**K**、**L**和**M**的父節點

節點**I**和**J**擁有共同父節點，稱為「兄弟節點」(Siblings)，**K**、**L**和**M**是兄弟節點，**B**、**C**...和**H**節點也是兄弟節點



樹的相關術語

n元樹：樹的一個節點最多擁有n個子節點。

二元樹（ Binary Trees ）：樹的節點最多只有兩個子節點。

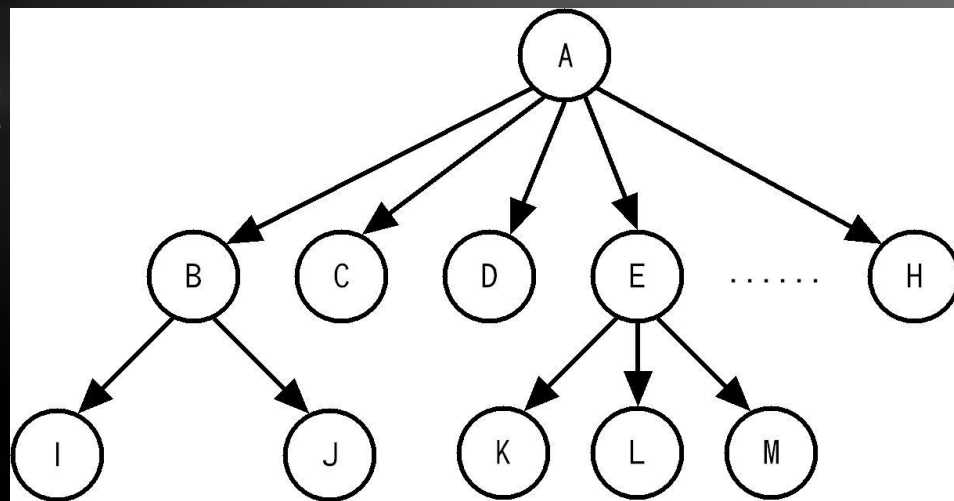
根節點（ Root ）：沒有父節點的節點是根節點。

- 例如：節點A。

葉節點（ Leaf ）：節點沒有子節點的節點稱為葉節點。

- 例如：節點I、J、C、D、K、L、M、F、G和H。

祖先節點（ Ancenstors ）：
指某節點到根節點之間所經過的所有節點，都是此節點的祖先節點。



樹的相關術語

非終端節點（ Non-terminal Nodes ）：除了葉節點之外的其它節點稱為非終端節點。

- 例如：節點A、B和E是非終端節點。

分支度（ Degree ）：指每個節點擁有的子節點數

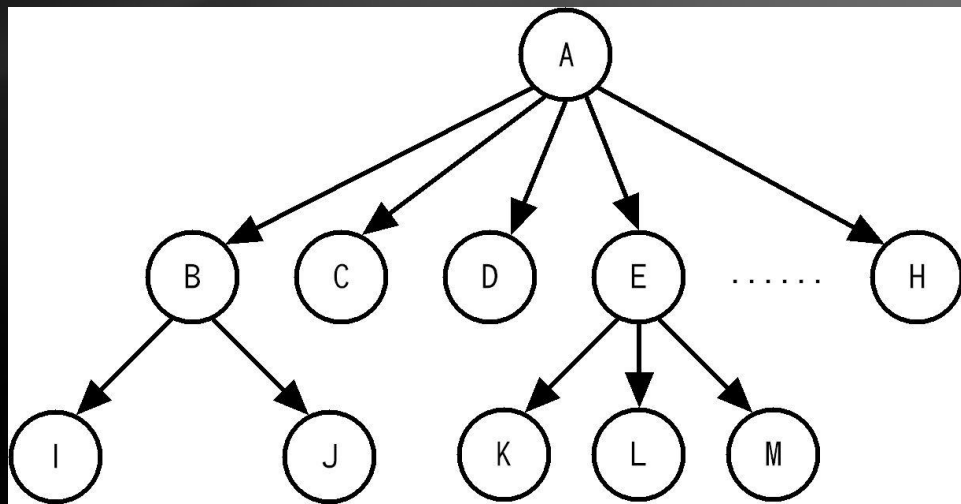
- 例如：節點B的分支度是2，節點E的分支度是3。

階層（ Level ）：如果樹根是1，其子節點是2，依序可以計算出樹的階層數。

- 例如：上述圖例的節點A階層是1，B、C到H是階層2，I、J到M是階層3。

樹高（ Height ）：
樹高又稱為樹深（ Depth ）
，指樹的最大階層數。

- 例如：圖例的樹高是3。



大綱

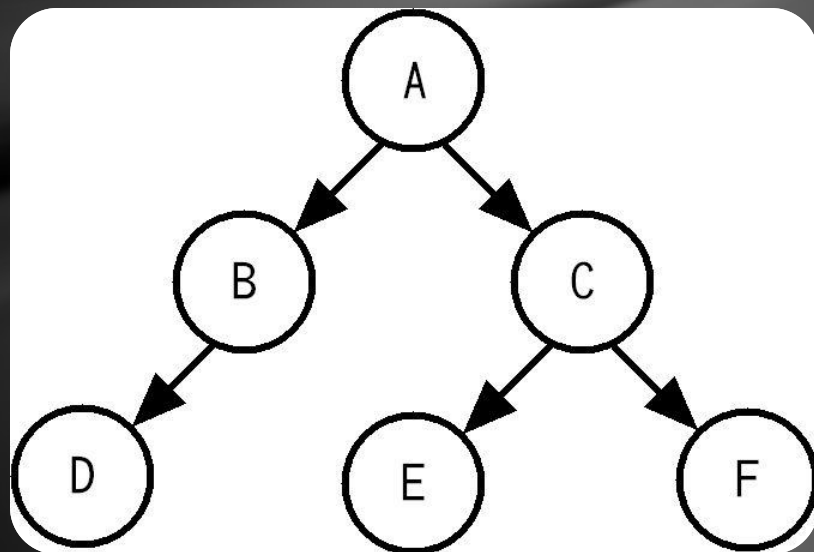
樹 (Tree)

二元樹 (Binary Tree)

二元搜尋樹 (Binary Search Tree)

二元樹

- 樹依不同分支度可以區分成很多種，在資料結構中最廣泛使用的樹狀結構是「二元樹」(Binary Trees)
- 二元樹是指樹中的每一個「節點」(Node) 最多只能擁有2個子節點，即分支度小於或等於2
- 二元樹的定義如下所示：
- 二元樹的節點個數是
 - 一個有限集合
 - 沒有節點的空集合
- 二元樹的節點可以分成兩個沒有交集的子樹，稱為
 - 「左子樹」
(Left Sub-tree)
 - 「右子樹」
(Right Sub-tree)



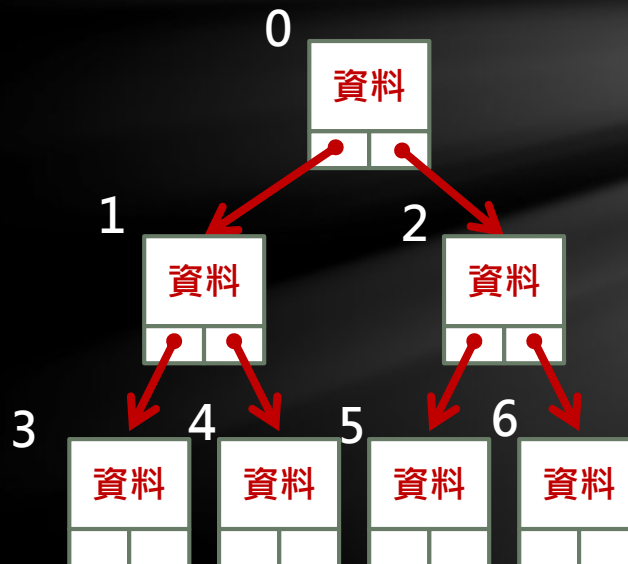
二元樹: 歪斜樹

- ▶ 左邊這棵樹沒有右子樹，右邊這棵樹沒有左子樹，雖然擁有相同節點，但是這是兩棵不同的二元樹，因為所有節點都是向左子樹或右子樹歪斜，稱為「**歪斜樹**」 (Skewed Tree)



二元樹: 完滿二元樹

- 若二元樹的樹高是 h 且二元樹的節點數是 $2^h - 1$ ，滿足此條件的樹稱為「完滿二元樹」(Full Binary Tree)

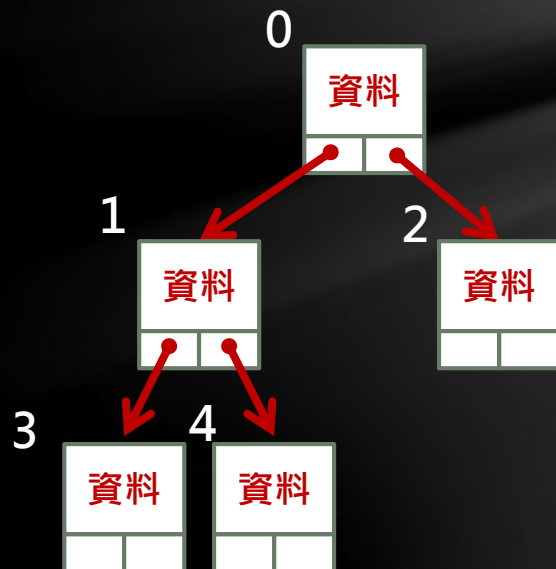


二元樹: 完滿二元樹

- ▶ 因為二元樹的每一個節點有2個子節點，二元樹樹高是3，也就是有3個階層（Level），各階層的節點數，如下所示：
 - ▶ 第1階： $1 = 2^{(1-1)} = 2^0 = 1$
 - ▶ 第2階：第1階節點數的2倍， $1*2 = 2^{(2-1)} = 2$
 - ▶ 第3階：第2階節點數的2倍， $2*2 = 2^{(3-1)} = 4$
- ▶ 以此類推，得到每一階層的最大節點數是： $2^{(l-1)}$ ， l 是階層數，整棵二元樹的節點數一共是： $2^0 + 2^1 + 2^2 = 7$ 個，即 $2^3 - 1$ ，可以得到：
 - ▶ $2^0 + 2^1 + 2^2 + \dots + 2^{(h-1)} = 2^h - 1$ ， h 是樹高

二元樹: 完整二元樹

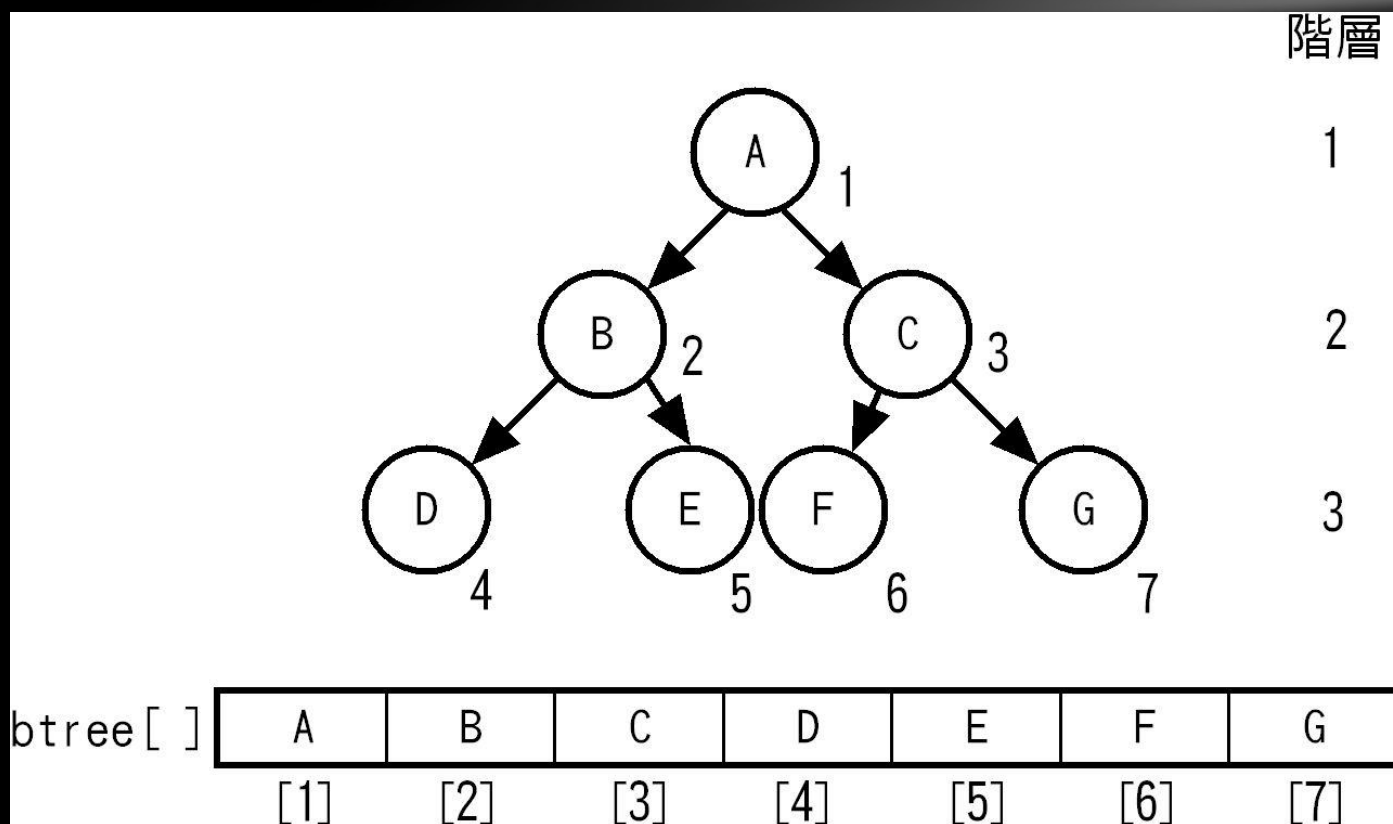
- 若二元樹的節點總數不足 2^h-1 ，其中 h 是樹高，除最後一層外全填滿，最後一層節點全靠左，而且其節點編號是對應相同高度完滿二元樹的節點編號，因為編號完整，中間沒有空號，因此稱為完整二元樹 (Complete Binary Tree)



二元樹陣列表示法

一棵樹高 h 擁有 2^h-1 個節點的二元樹，這是二元樹在樹高 h 所能擁有的最大節點數

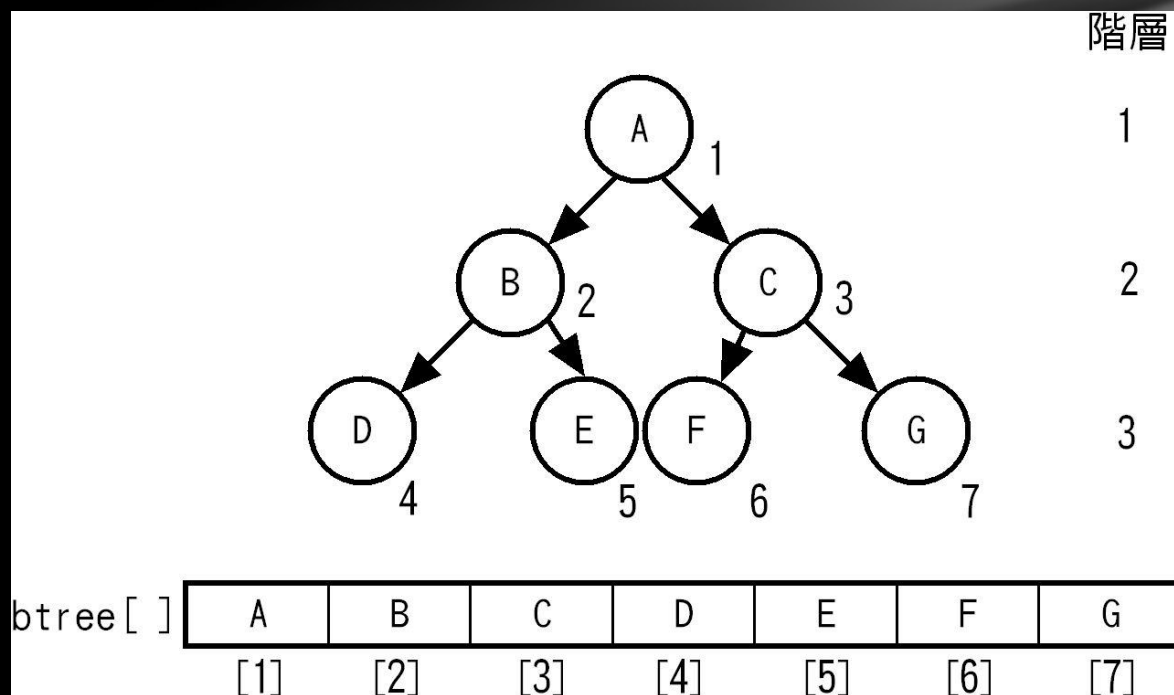
換句話說，只需配置 2^h-1 個元素，我們就可以儲存樹高 h 的二元樹，如下圖所示



二元樹陣列表示法

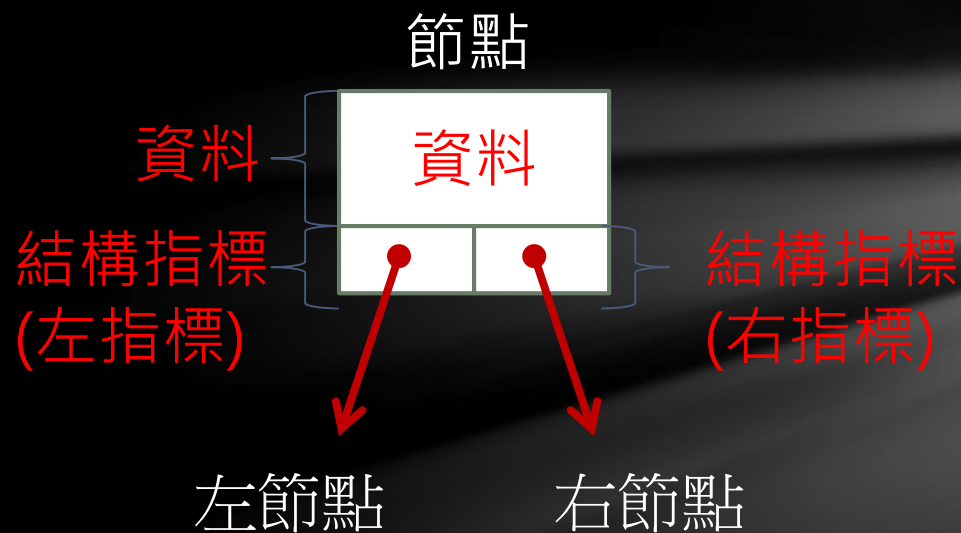
二元樹的節點編號擁有循序性，根節點1的子節點是節點2和節點3，節點2是4和5，依此類推可以得到節點編號的規則，如下所示：

- 左子樹是父節點編號乘以2
- 右子樹是父節點編號乘以2加1



二元樹鏈結表示法

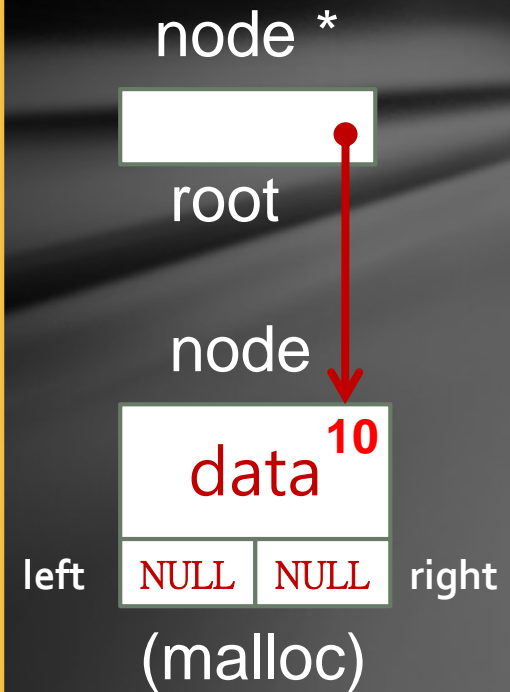
▶ 二元樹節點鏈結表示法



二元樹節點 = 資料 + 結構指標(左) + 結構指標(右)

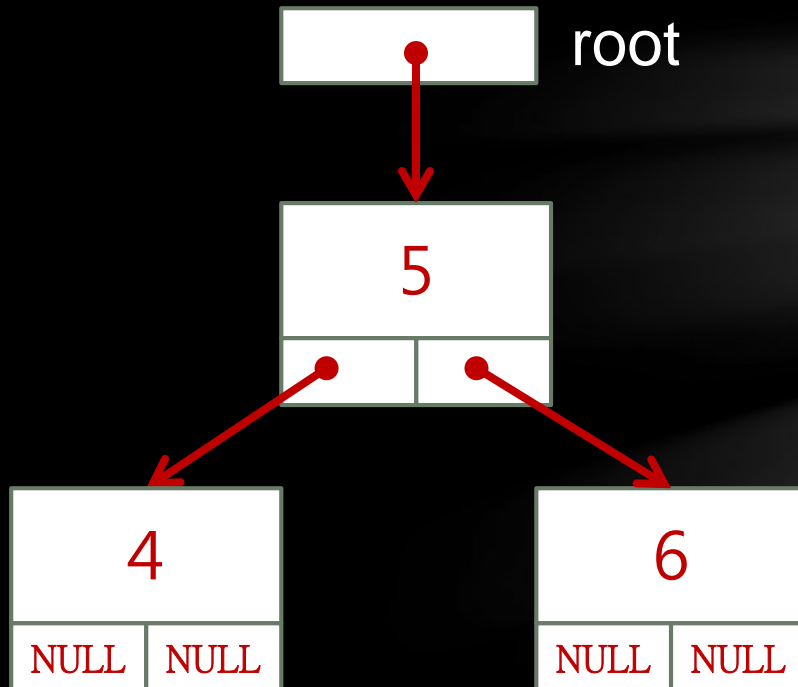
二元樹鏈結表示法

```
struct binary_tree_node
{
    資料型態 變數名稱;
    int data;
    struct binary_tree_node *left;
    struct binary_tree_node *right;
};
typedef struct binary_tree_node node;
...
node *root;
root = (node *)malloc(sizeof(node));
root ->data= 10;
root ->left= NULL;
root ->right= NULL;
```



小練習 (Binary_Tree.c)

- ▶ 試著建立一個二元樹如下圖所示



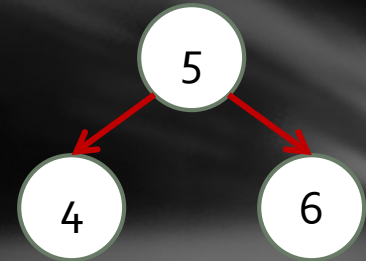
```
node *root;
```

```
// Level 1
```

```
root = (node *)malloc(sizeof(node));  
root->data = 5;  
root->left = NULL;  
root->right = NULL;
```

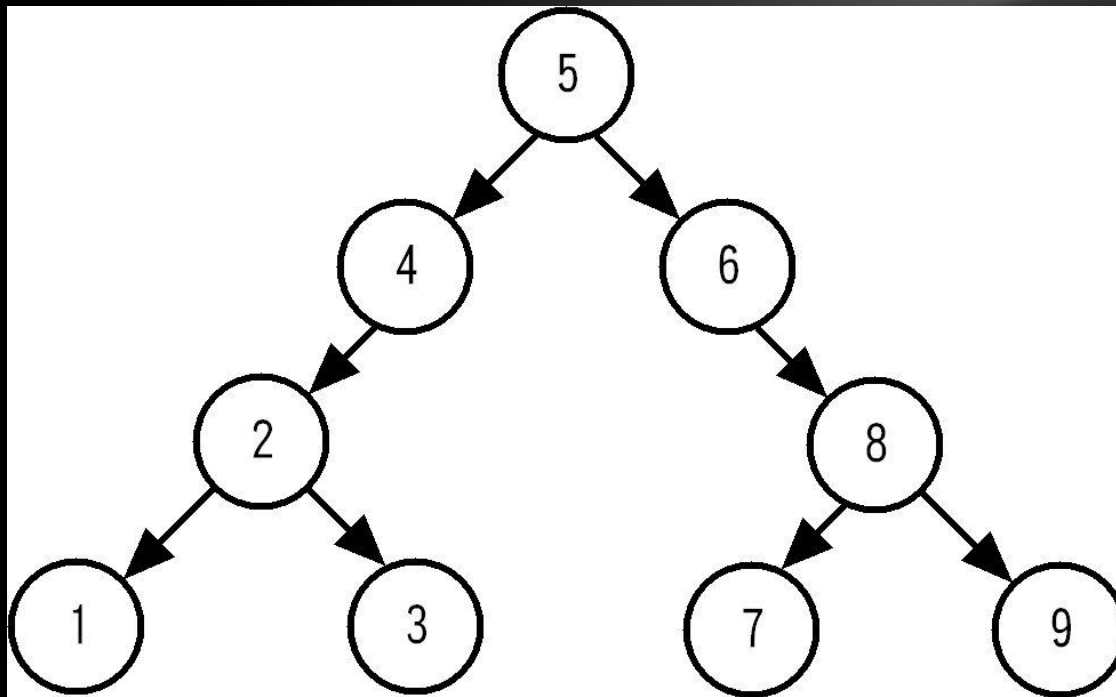
```
// Level 2
```

```
root->left = (node *)malloc(sizeof(node));  
root->left->data = 4;  
root->left->right = NULL;  
root->left->left = NULL;  
root->right = (node *)malloc(sizeof(node));  
root->right->data = 6;  
root->right->left = NULL;  
root->right->right = NULL;
```



二元樹的走訪

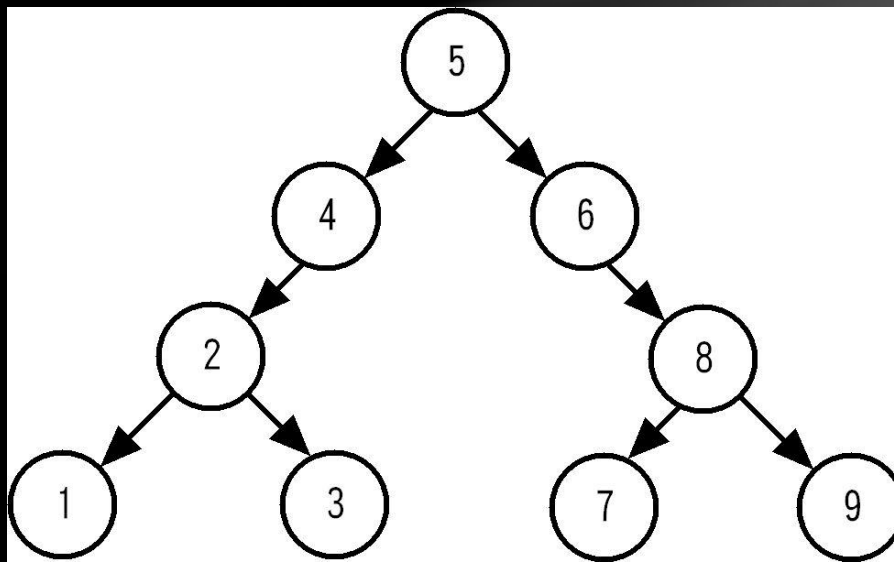
陣列和單向鏈結串列都只能從頭至尾或從尾至頭執行單向「**走訪**」(Traverse)，不過二元樹的每一個節點都擁有指向**左**和**右**2個子節點的指標，所以走訪可以有**兩條路徑**。



二元樹的走訪種類

二元樹的走訪過程是持續決定向左或向右走，直到沒路可走。二元樹的走訪是一種**遞迴走訪**，依照遞迴函數中呼叫的**排列順序不同**，可以分成**三種**走訪方式，如下所示：

- **中序**走訪方式（ Inorder Traversal ）。
- **前序**走訪方式（ Preorder Traversal ）。
- **後序**走訪方式（ Postorder Traversal ）。



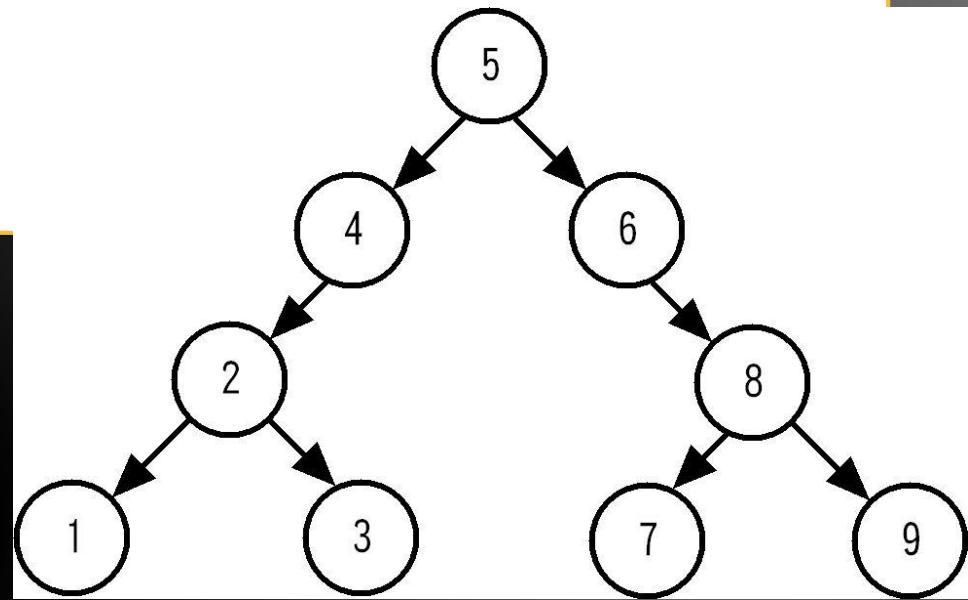
二元樹的走訪種類

中序走訪 (Inorder Traversal)

```
void print_inorder(node *ptr)
{
    if(ptr != NULL)
    {
        print_inorder(ptr->left);
        printf("[%2d]\n", ptr->data);
        print_inorder(ptr->right);
    }
}
```

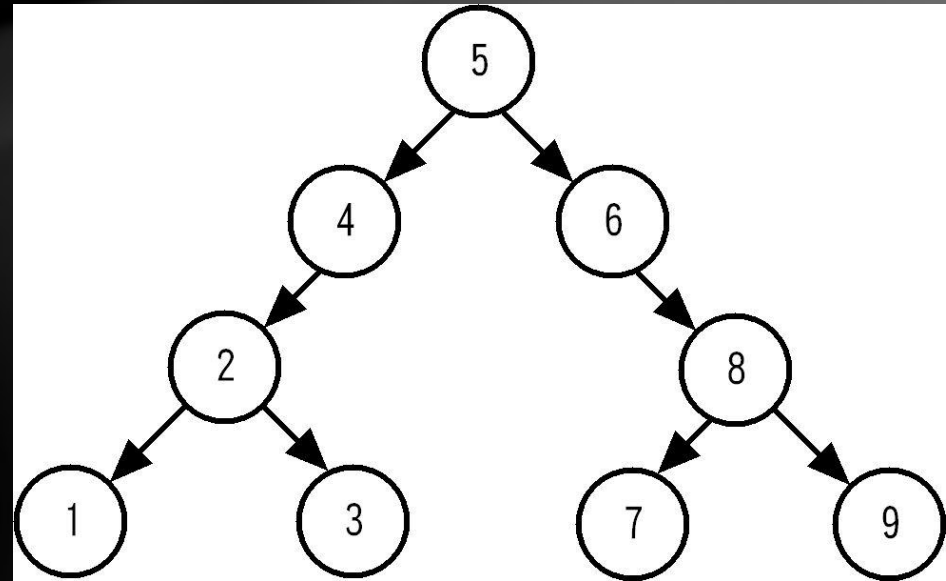
- 先印左子節點
- 再印自己(中)
- 右子節點最後

以從小到大方式列印
中序: 1,2,3,4,5,6,7,8,9



二元樹的走訪種類

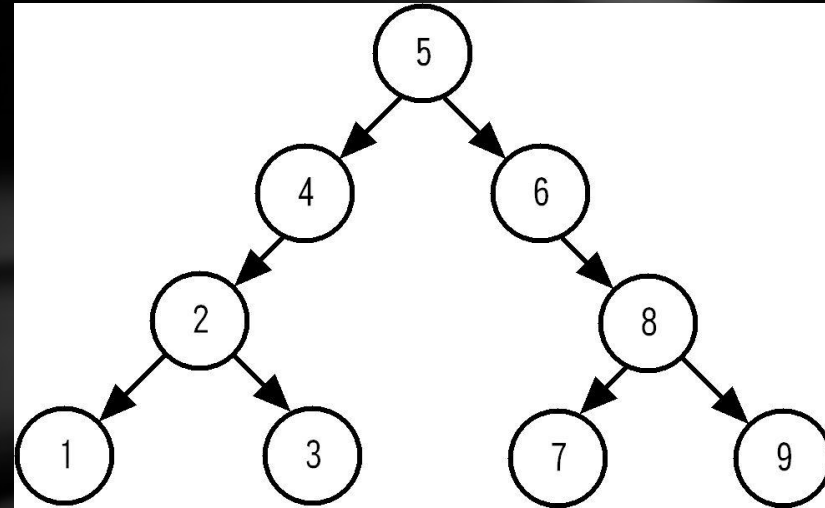
- 前序走訪 (Preorder Traversal)
 - 每節點皆會比它的左子節點及右子節點先印
 - 左子節點又比右子節點先印 (自己->左->右)
 - **5,4,2,1,3,6,8,7,9**
- 後序走訪 (Postorder Traversal)
 - 印自己前先印左節點
 - 再印右節點 自己最後
 - (左->右->自己)
 - **1,3,2,4,7,9,8,6,5**



小練習 (Binary_Tree.c)

前序走訪 (Preorder Traversal)

```
void print_preorder(node *ptr)
{
    if(ptr != NULL)
    {
        // 前序: 5,4,2,1,3,6,8,7,9
    }
}
```



後序走訪 (Postorder Traversal)

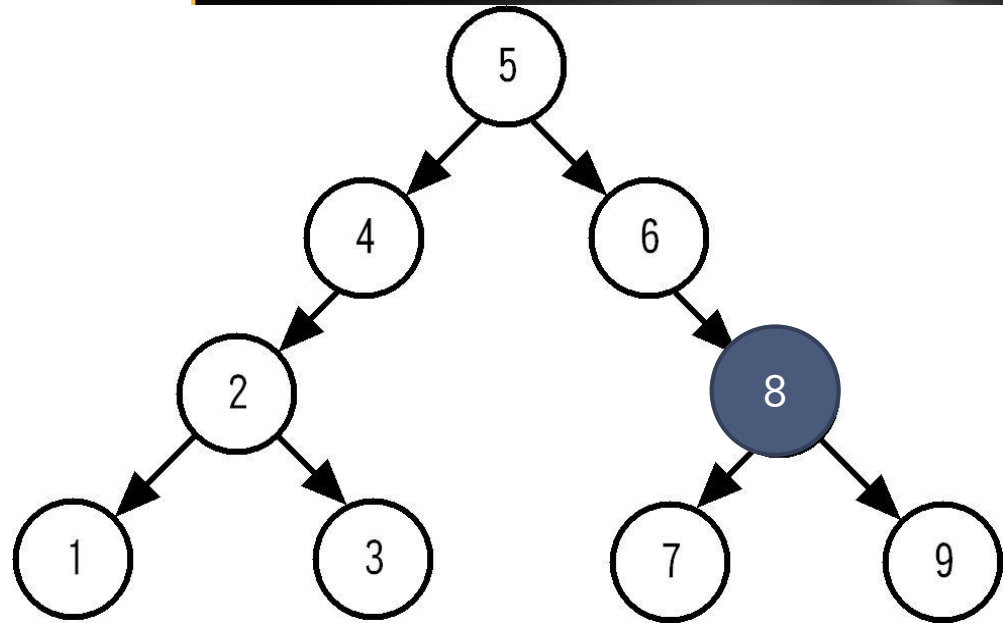
```
void print_postorder(node *ptr)
{
    if(ptr != NULL)
    {
        // 後序: 1,3,2,4,7,9,8,6,5
    }
}
```

二元樹的搜尋 (Tree_LinearSearch.c)

利用二元樹走訪實作二元樹的搜尋

```
node *search_node(node *ptr, int value)
{
    node *temp;

    if(ptr != NULL)
    {
        if(ptr->data == value)
            return ptr;
        else
        {
            temp = search_node(ptr->left, value);
            if(temp != NULL)
                return temp;
            temp = search_node(ptr->right, value);
            if(temp != NULL)
                return temp;
        }
    }
    return NULL;
}
```



大綱

樹 (Tree)

二元樹 (Binary Tree)

二元搜尋樹 (Binary Search Tree)

二元搜尋樹

「二元搜尋樹」(Binary Search Trees) 是一種二元樹，其節點資料的排列擁有一些特性，如下所示：

- 二元樹的每一個節點值都不相同，在整棵二元樹中的每一個節點都擁有不同值
- 每一個節點的資料
 - 大於左子節點的資料
 - 且小於右子節點的資料
- 節點的左、右子樹也是一棵二元搜尋樹

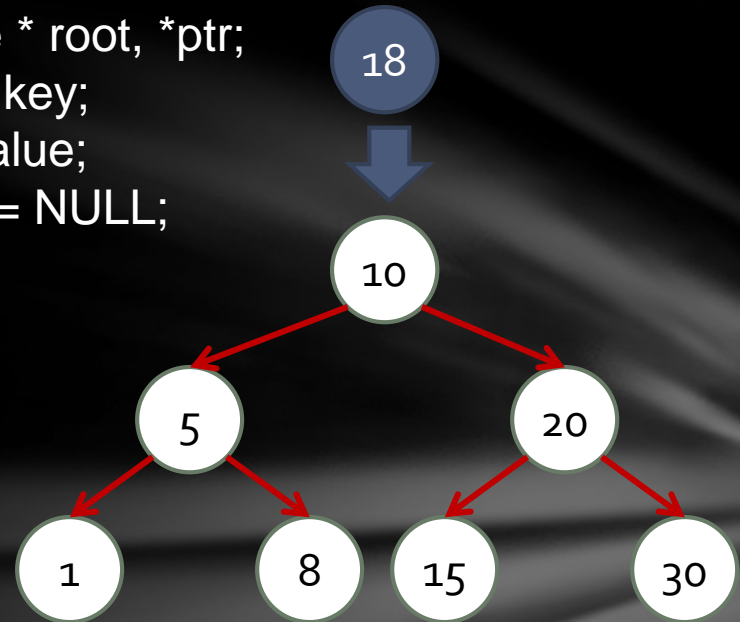
二元搜尋樹: 插入

```
node *insert_node(node *root, int value)
{
    node *new_node;
    node *current;
    node *parent;

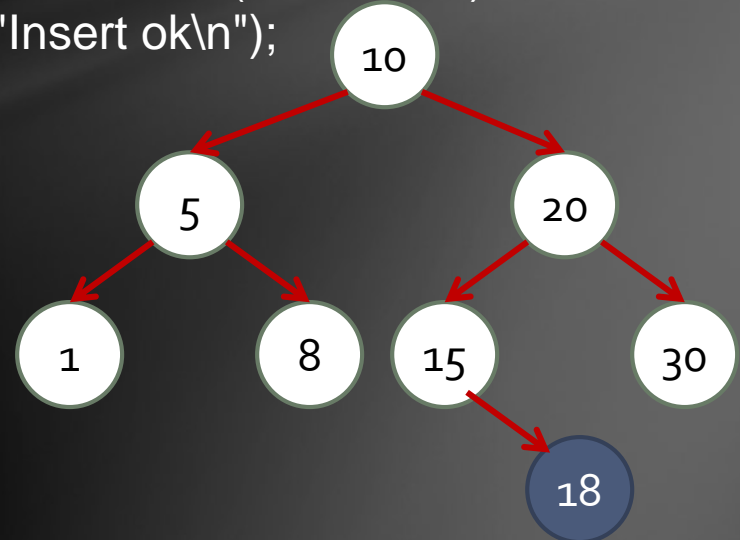
    new_node = (node *)malloc(sizeof(node));
    new_node->data = value;
    new_node->left = NULL;
    new_node->right = NULL;
    if(root == NULL)
    {
        return new_node;
    }
    else
    {
        current = root;
        while(current != NULL)
        {
            parent = current;
            if(current->data > value)
                current = current->left;
            else
                current = current->right;
        }
        if(parent->data > value)
            parent->left = new_node;
        else
            parent->right = new_node;
    }
    return root;
}
```



```
node * root, *ptr;
char key;
int value;
root = NULL;
```



```
scanf("%d",&value);
ptr = root;
root=insert_node(root, value);
printf("Insert ok\n");
```



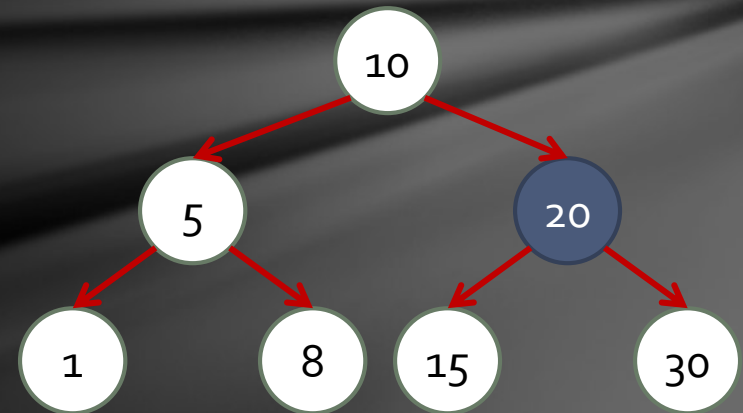
小練習 (BinarySearchTree.c)

- 請實作上述之二元搜尋樹的插入演算法
- 並製做一選單讓使用者輸入 'i' 新增節點
- 可輸入數字並依據數值大小順序插入節點
(假設輸入之數字不會重覆)
- 並試著建構出如上頁圖中的二元樹
 - Hint:輸入的次序很重要，為什麼？
- 輸入 'l' 可選擇列印順序印出所有節點內容
各以前/中/後序列印出來

二元搜尋樹: 搜尋

比較：一般二元樹的搜尋 (search_node())

```
node *find_node(node *ptr, int value)
{
    while(ptr != NULL)
    {
        if(ptr->data == value)
            return ptr;
        else
        {
            if(ptr->data > value)
                ptr = ptr->left;
            else
                ptr = ptr->right;
        }
    }
    return NULL;
}
```



```
scanf("%d",&value);
ptr = find_node(root, value);
if(ptr != NULL)
    printf("found: %d\n", ptr->data)
else
    printf("Not found\n");
```

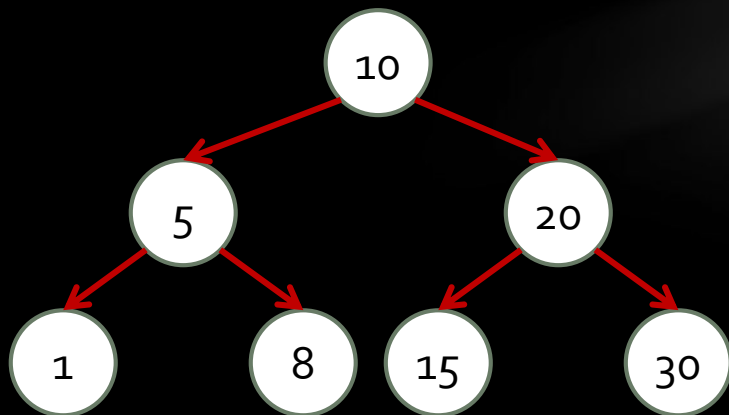
二元搜尋樹：刪除

必須先取得欲刪節點之父節點

判斷此節點為父節點之左節點或右節點

考量三種情況：

- 情況1: 節點沒有左子樹
- 情況2: 節點沒有右子樹
- 情況3: 節點有左右子樹
- (程式碼詳見範例程式)

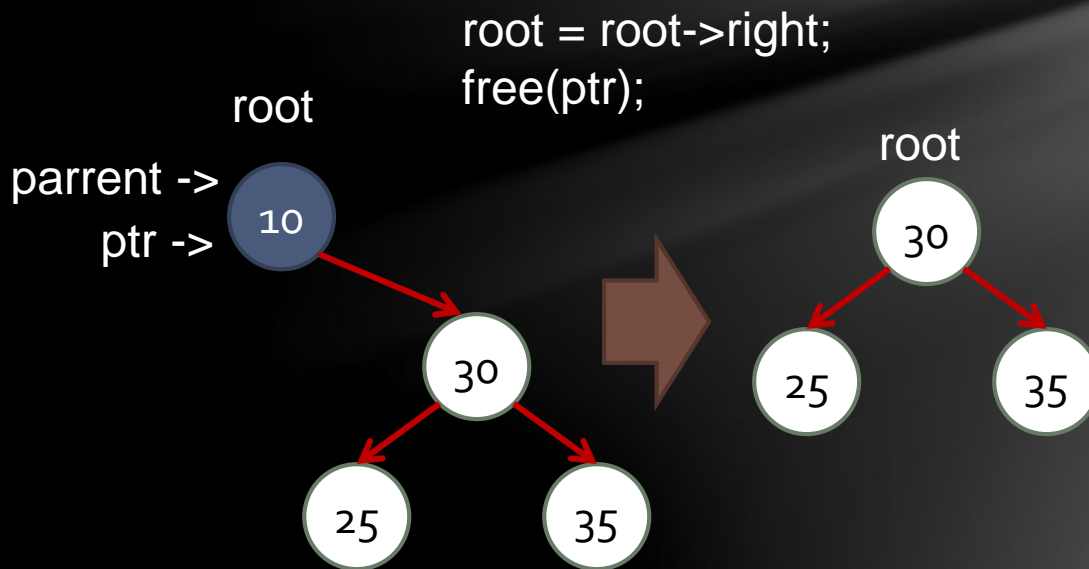


```
scanf("%d",&value);  
ptr = find_node(root, value);  
if(ptr != NULL)  
    root= delete_node(root, value);  
    printf("Delete ok\n");  
}  
else  
    printf("Can not delete\n");
```

二元搜尋樹: 刪除

情況1: 節點沒有左子樹

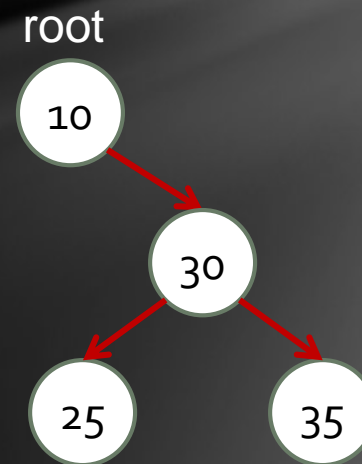
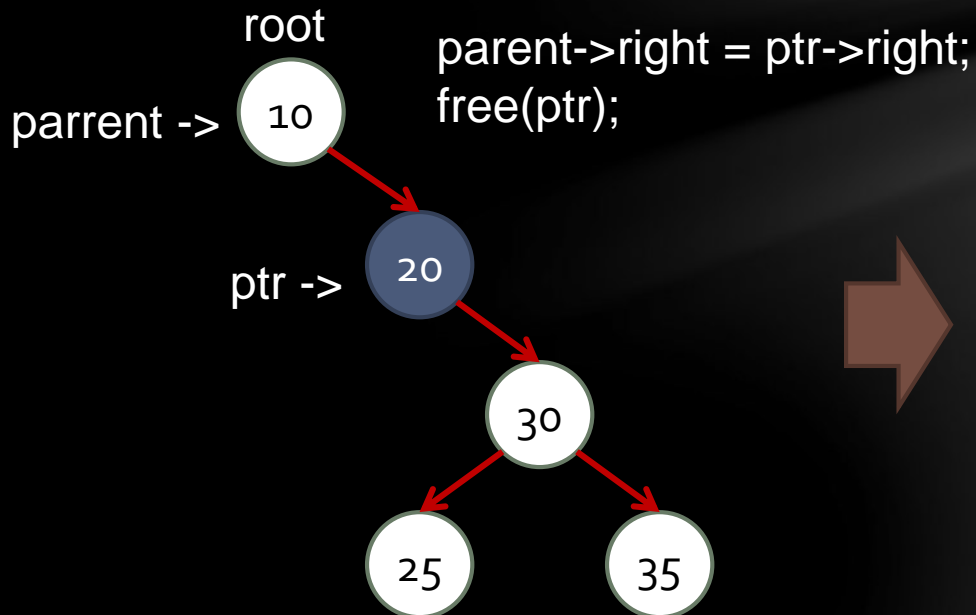
- 如果要刪的是根節點
- 要刪除的節點在父節點右方
- 要刪除的節點在父節點左方



二元搜尋樹: 刪除

情況1: 節點沒有左子樹

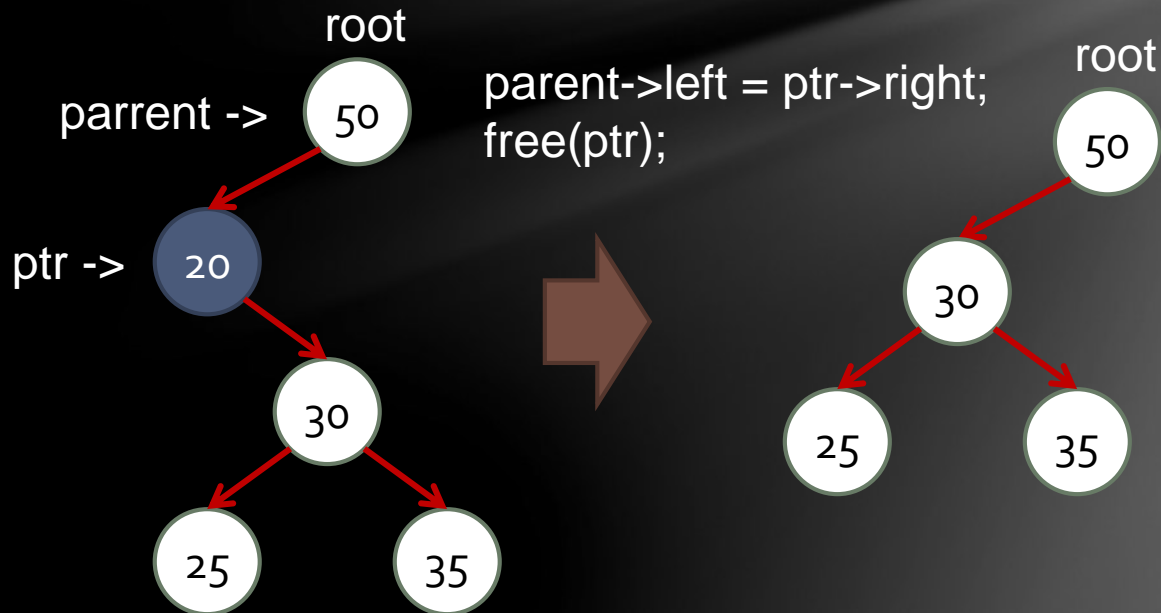
- 如果要刪的是根節點
- 要刪除的節點在父節點右方
- 要刪除的節點在父節點左方



二元搜尋樹: 刪除

情況1: 節點沒有左子樹

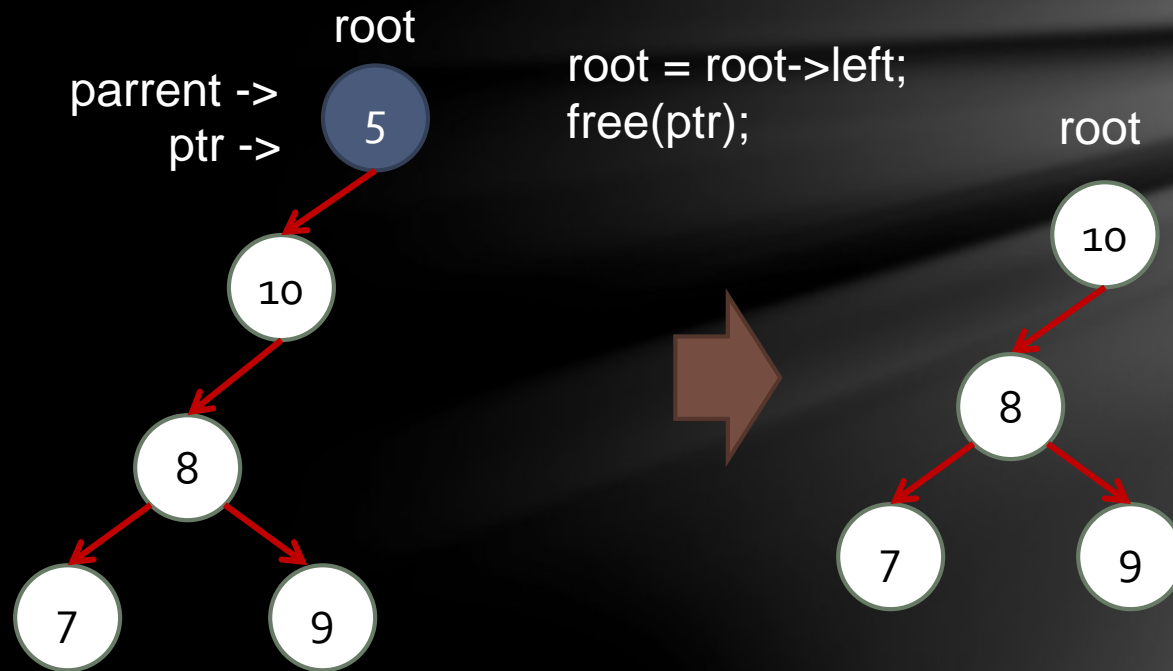
- 如果要刪的是根節點
- 其他
 - 要刪除的節點在父節點右方
 - 要刪除的節點在父節點左方



二元搜尋樹: 刪除

情況2: 節點沒有右子樹

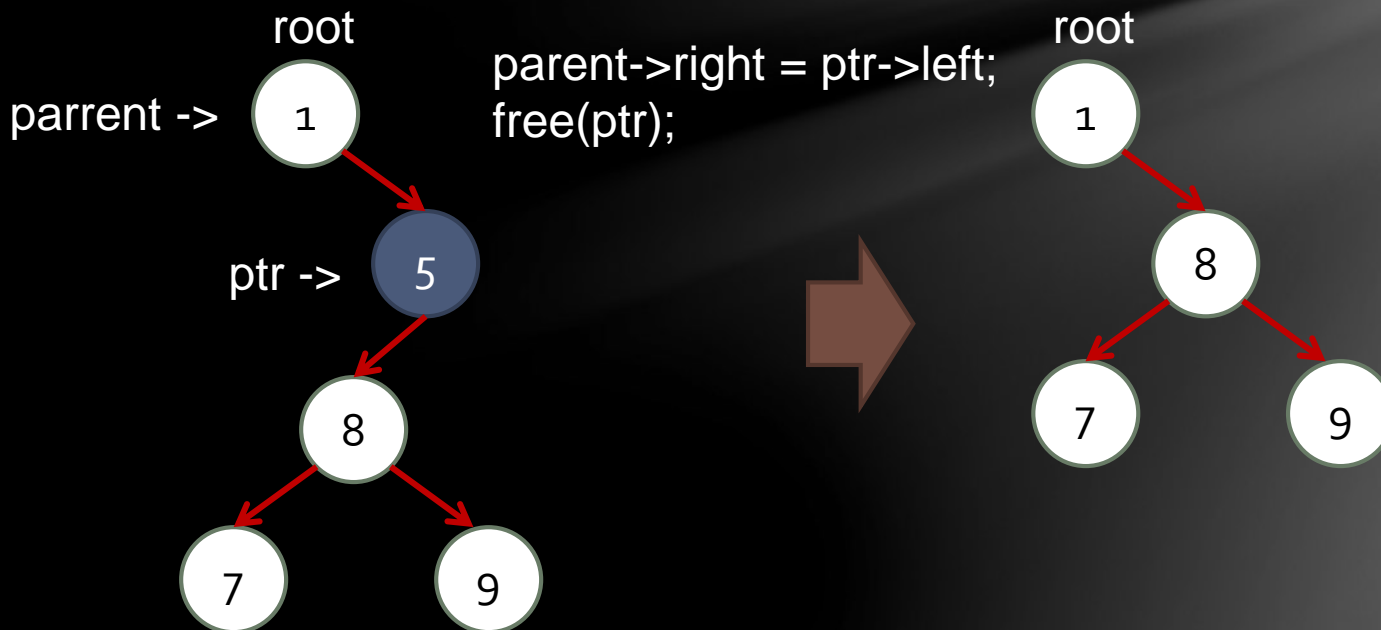
- 如果要刪的是根節點
- 其他



二元搜尋樹: 刪除

情況2: 節點沒有右子樹

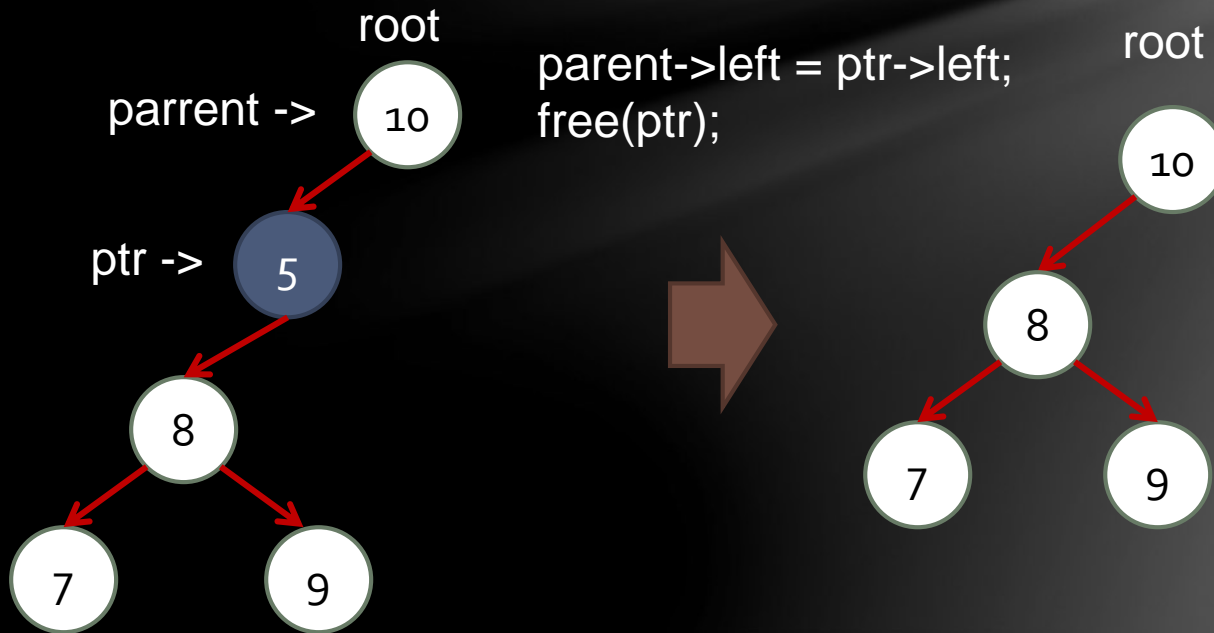
- 如果要刪的是根節點
- 其他
 - 要刪除的節點在父節點右方
 - 要刪除的節點在父節點左方



二元搜尋樹: 刪除

情況2: 節點沒有右子樹

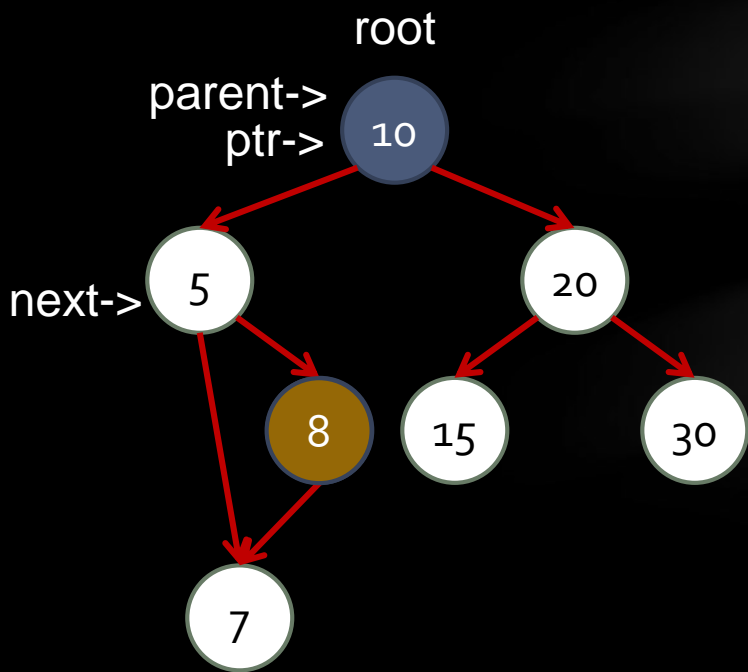
- 如果要刪的是根節點
- 其他
 - 要刪除的節點在父節點右方
 - 要刪除的節點在父節點左方



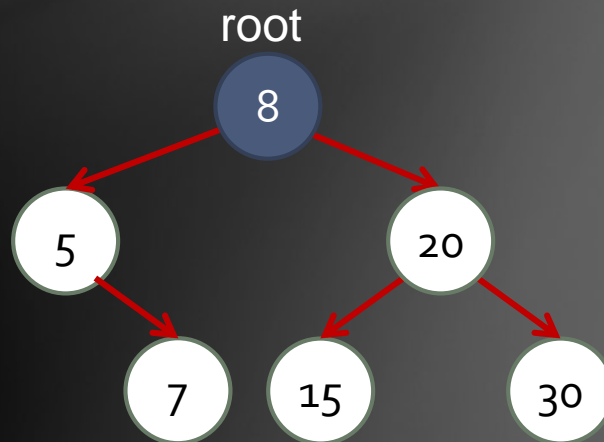
二元搜尋樹: 刪除

情況3: 節點有左右子樹

- 往左子節點之右子樹
- 找最大值當取代點



```
parent = ptr;  
next = ptr->left; // 找取代點next, 從左邊開始找  
if(next->right != NULL) {  
    while(next->right != NULL)  
    { // 往左子節點之右子樹找最大值當取代點  
        parent = next; // parent為next父節點  
        next = next->right;  
    } // 繞過next節點  
    parent->right = next->left;  
}  
else {  
    ptr->left = NULL;  
}  
ptr->data = next->data; // 取代!!  
free(next); // 刪除next (注意: 不是刪除ptr)
```

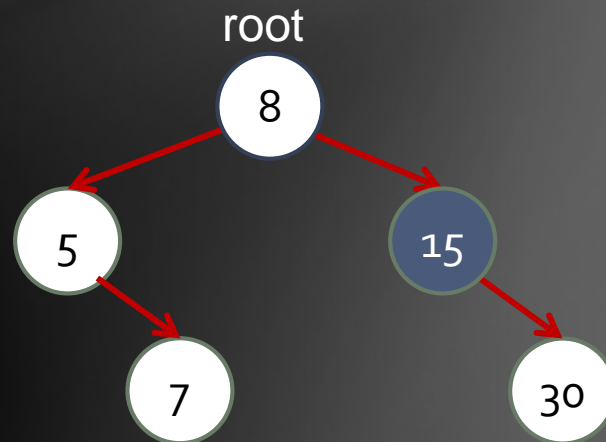
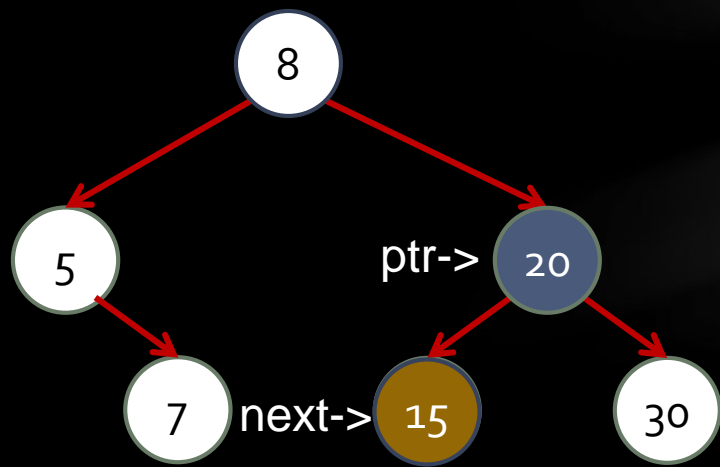


二元搜尋樹: 刪除

情況3: 節點有左右子樹

- 往左子節點之右子樹
- 找最大值當取代點

```
parent = ptr;  
next = ptr->left; //找取代點next,從左邊開始找  
if(next->right != NULL) {  
    while(next->right != NULL)  
    { // 往左子節點之右子樹找最大值當取代點  
        parent = next; //parent為next父節點  
        next = next->right;  
    } // 繞過next節點  
    parent->right = next->left;  
}  
else {  
    ptr->left = NULL;  
}  
ptr->data = next->data; // 取代!!  
free(next); // 刪除next (注意: 不是刪除ptr)
```



小練習 (BinarySearchTree.c)

完成實做以上介紹之二元搜尋樹

功能：

- 輸入'**i**' 新增節點,可輸入數字, 依據數值大小順序插入節點 (假設輸入之數字不會重覆)
- 輸入'**d**'接著輸入數字，可將一筆資料節點中之數字相同者刪除(假設輸入之數字不會重覆)
- 輸入'**f**'接著輸入一個數字，可將一筆資料節點中之數字相同者印出資料
- 輸入'**l**'依據數字順序印出所有節點內容
- 輸入'**q**' 讀取離開程式

回家作業(Addressbook_Tree.c)

使用二元搜尋樹製作一個電話簿

功能：

- 輸入' **i** ' 新增節點,可輸入 **姓名**, **電話**, 依據 **姓名字母順序** 插入節點(假設輸入之姓名不會重覆)
- 輸入' **d** ' 接著輸入 **姓名**, 可將一筆資料節點中之 **姓名相同者** 刪除(假設輸入之姓名不會重覆)
- 輸入' **f** ' 接著輸入一個 **姓名**, 可將一筆資料節點中之 **姓名相同者** 印出資料
- 輸入' **l** ' 依據 **姓名字母順序** 印出所有節點內容
- 輸入' **q** ' 讀取離開程式