C++資料結構與程式設計

樹狀結構(Tree)

NTU CSIE

大綱

樹 (Tree)

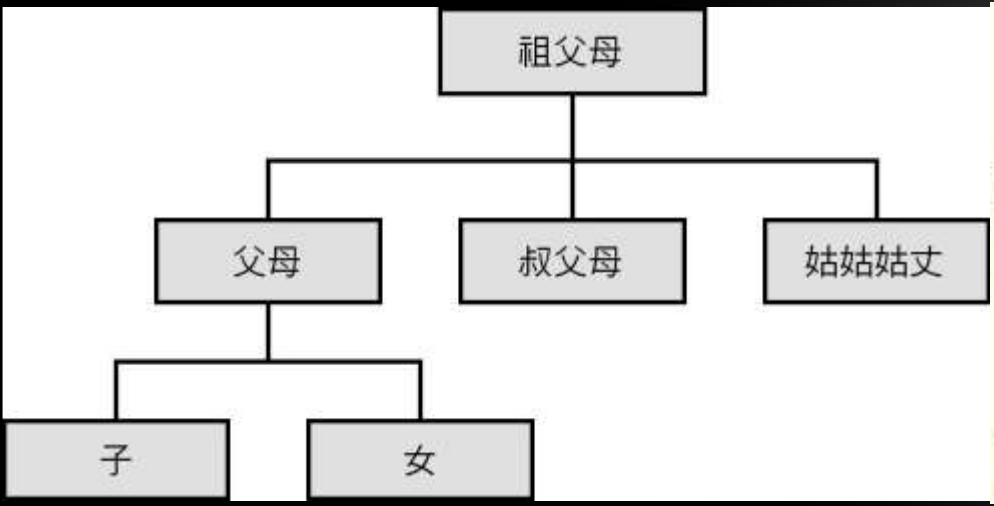
二元樹 (Binary Tree)

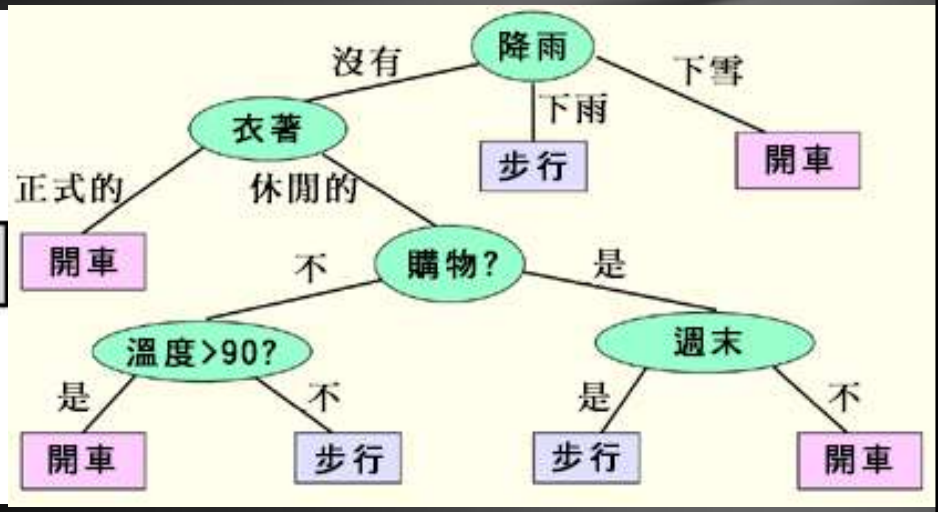
二元搜尋樹 (Binary Search Tree)

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

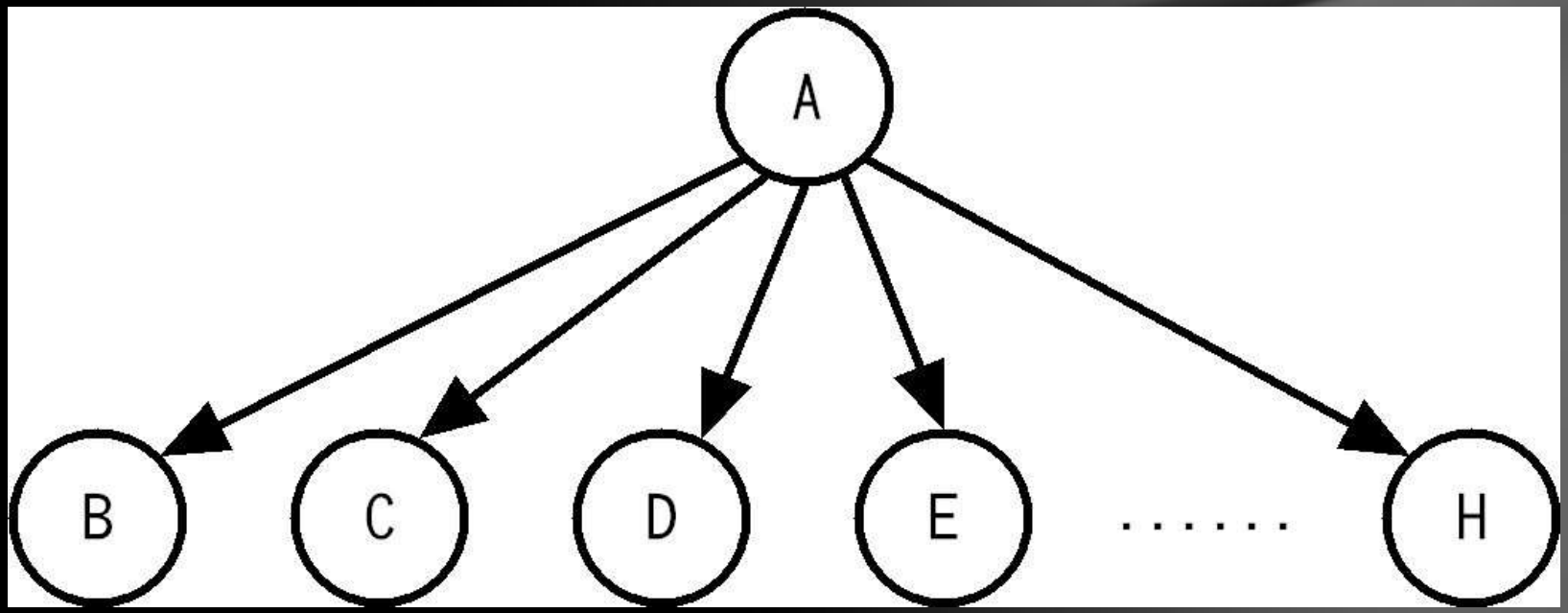




樹的基本術語

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）

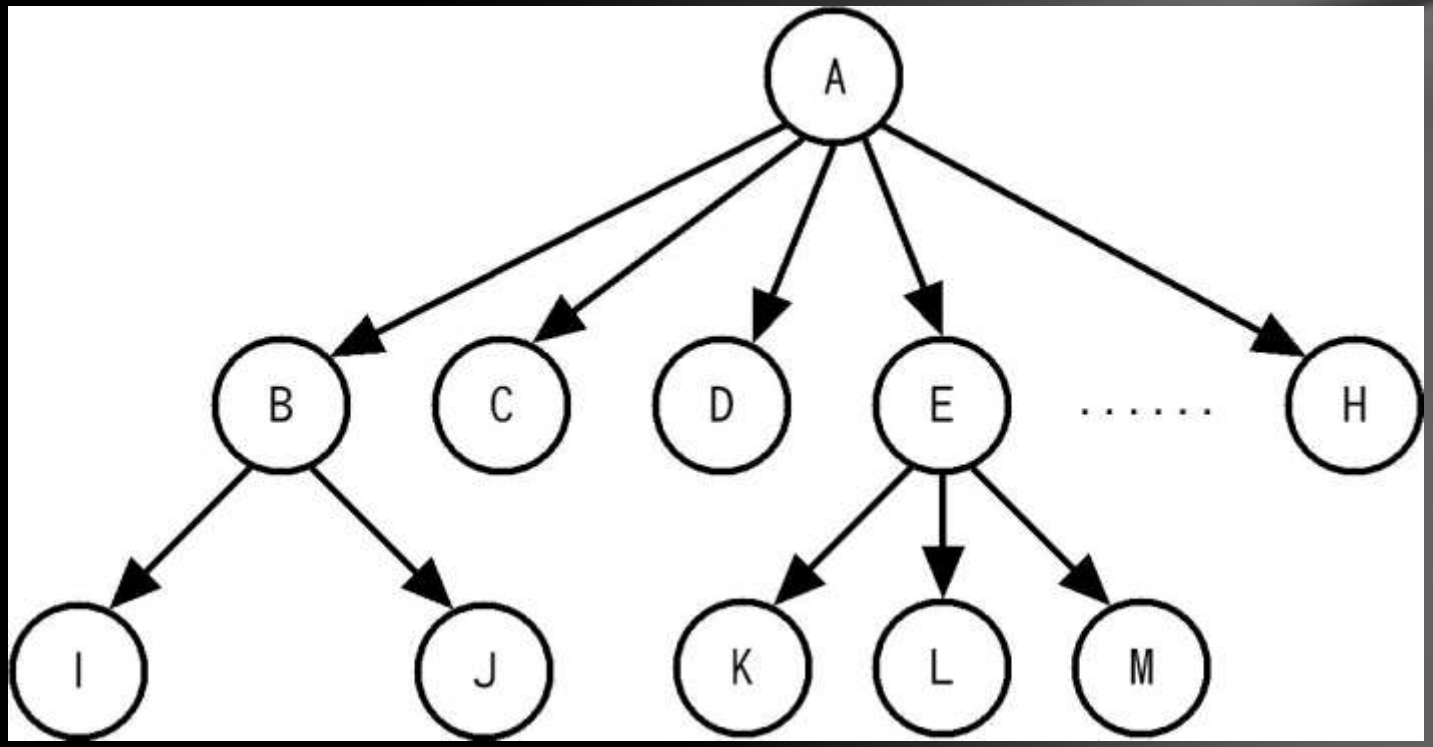
- 節點A是樹的根節點
- B、C、D….H是節點A的子節點



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。

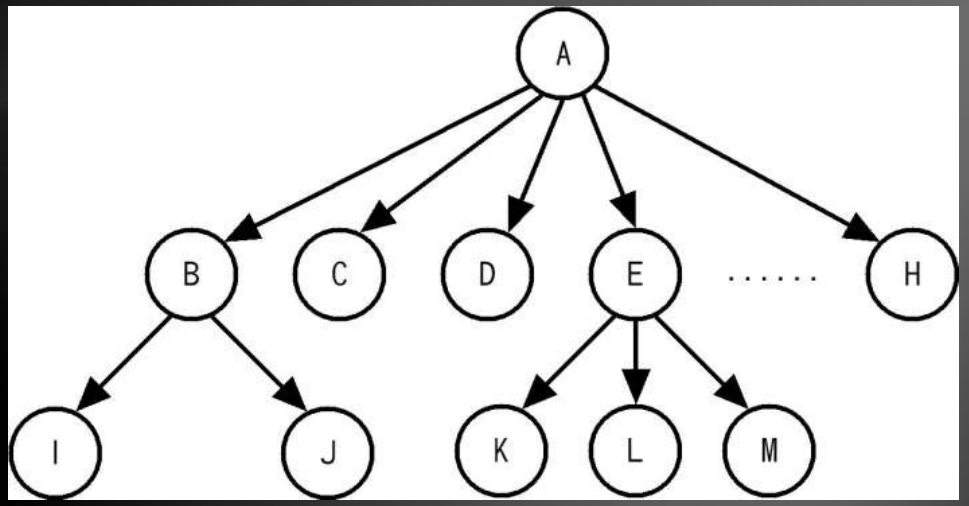
根節點（Root）：沒有父節點的節點是根節點。

- 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（Dregree）：指每個節點擁有的子節點數

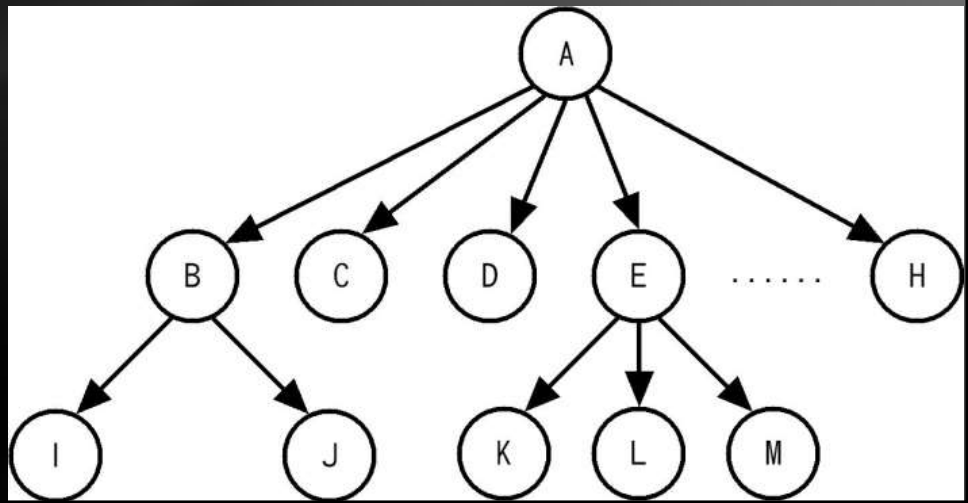
- 例如：節點B的分支度是2，節點E的分支度是3。

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

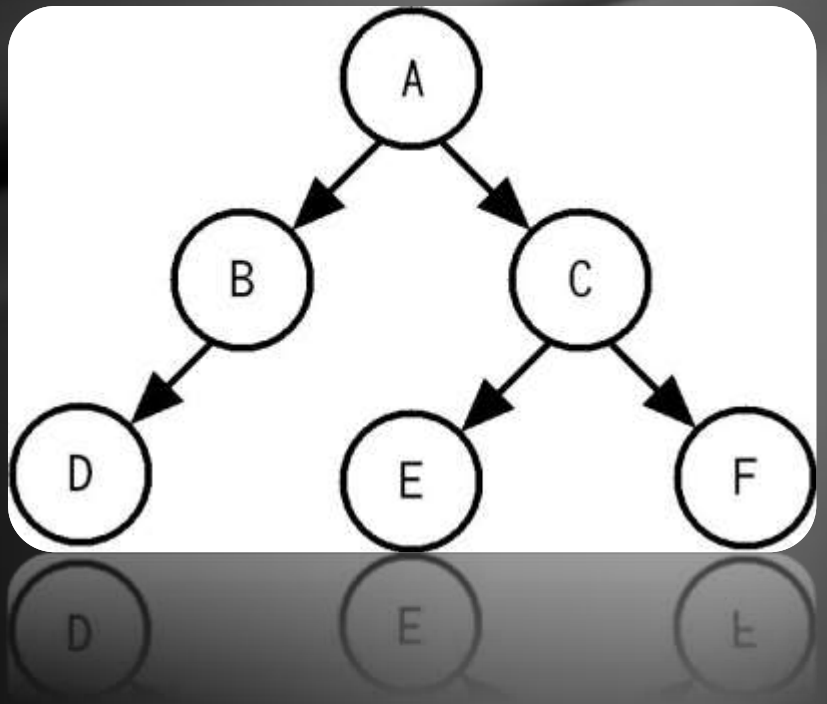
樹 (Tree)

二元樹 (Binary Tree)

二元搜尋樹 (Binary Search Tree)

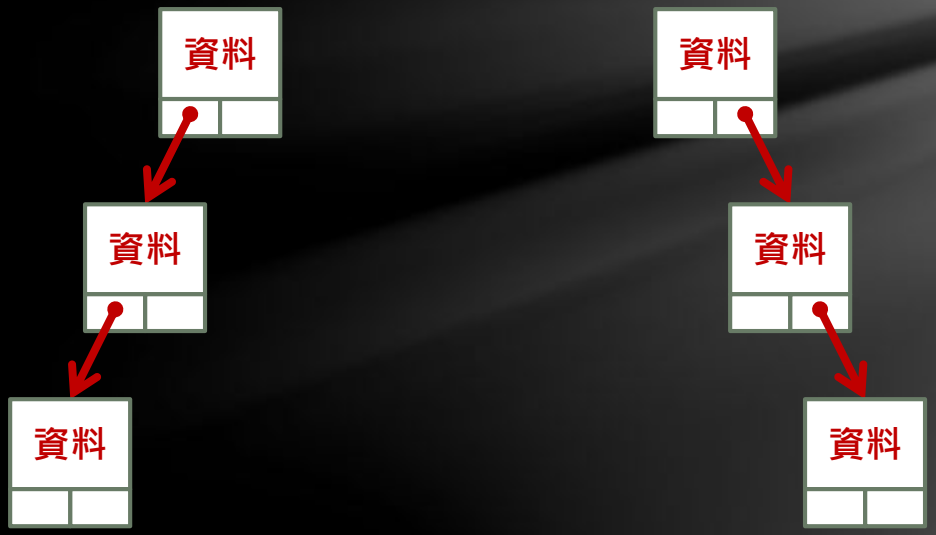
二元樹

- 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）



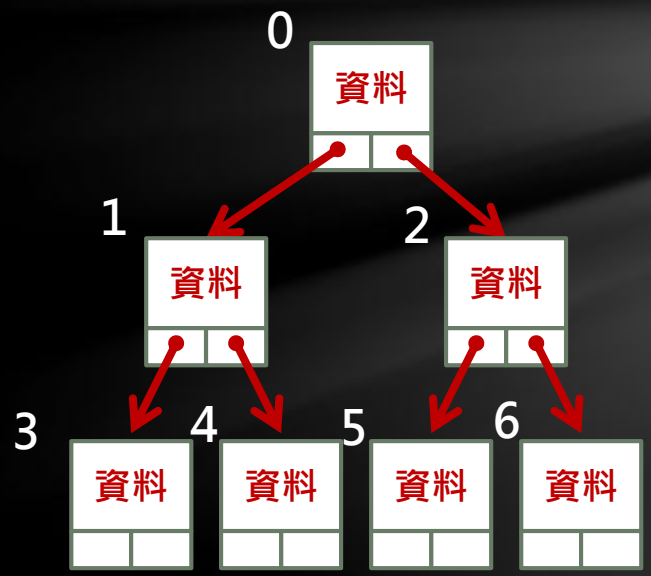
二元樹: 歪斜樹

- 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）

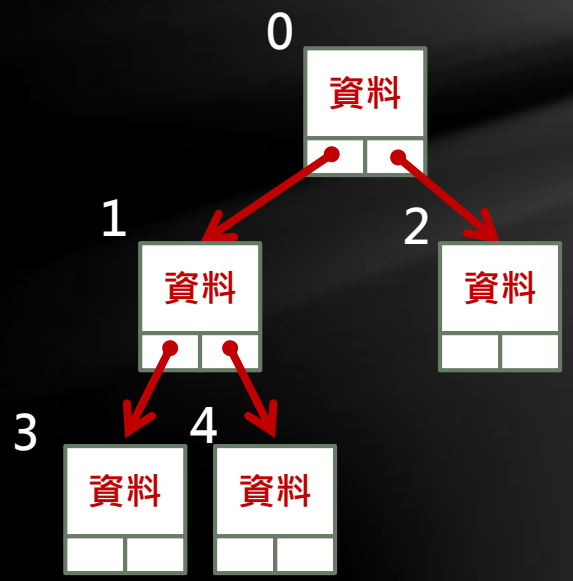


二元樹: 完滿二元樹

- 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：
  - $2^0+2^1+2^2+....+2^{(h-1)} = 2^h-1$，h是樹高

二元樹: 完整二元樹

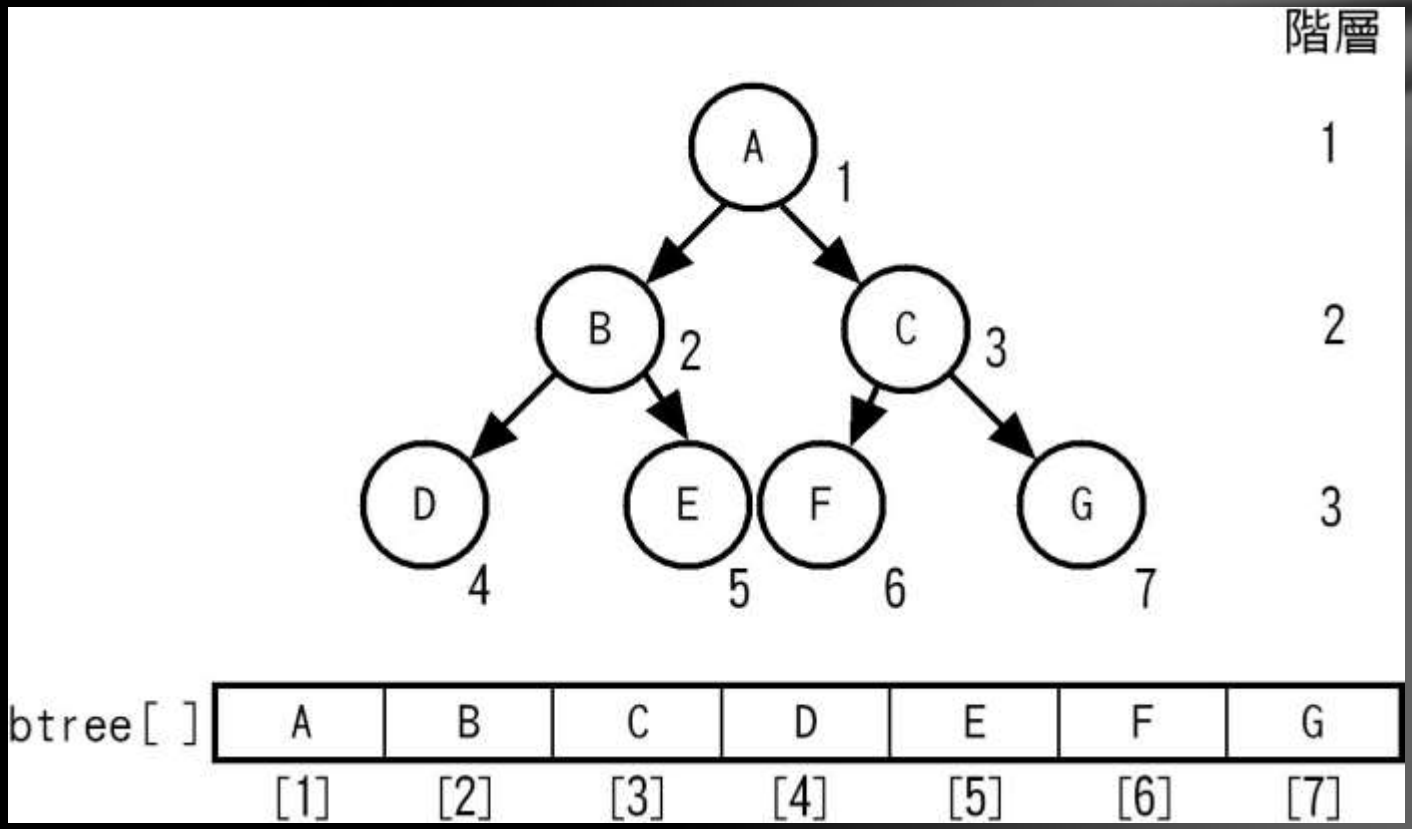
- 若二元樹的節點總數不足2^h-1，其中h是樹高，除最後一層外全填滿，最後一層節點全靠左，而且其節點編號是對應相同高度完滿二元樹的節點編號，因為編號完整，中間沒有空號，因此稱為完整二元樹（Complete Binary Tree）



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

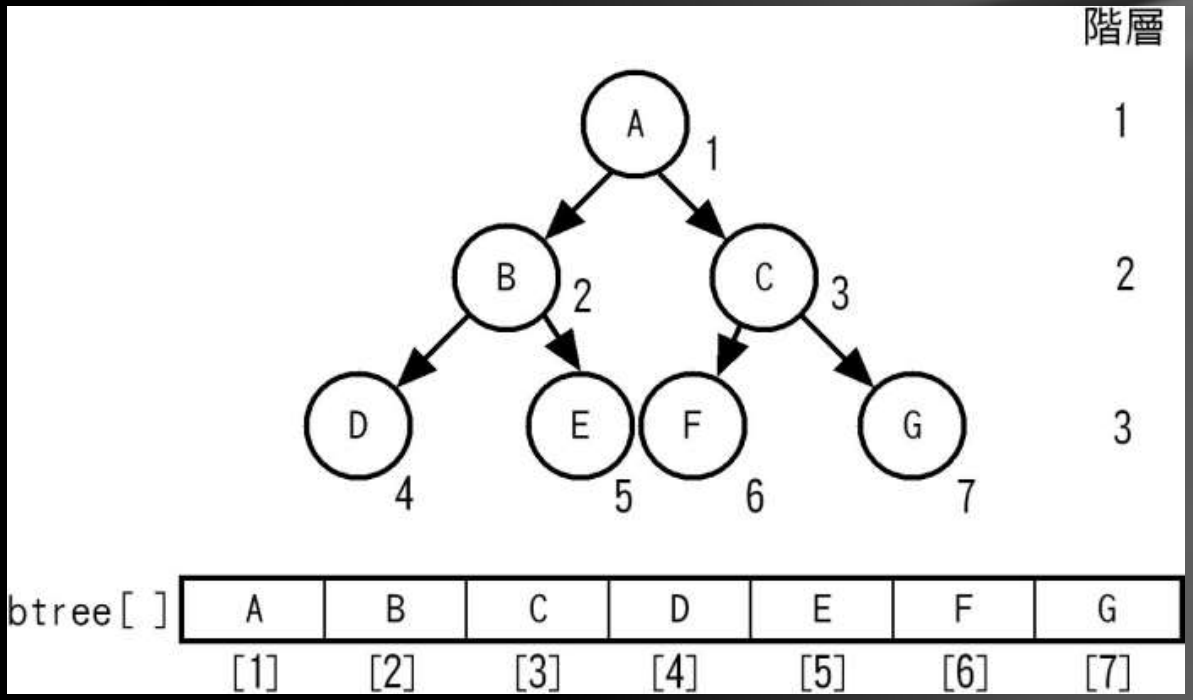
換句話說，只需配置2^h-1個元素，我們就可以儲存樹高h的二元樹，如下圖所示



二元樹陣列表示法

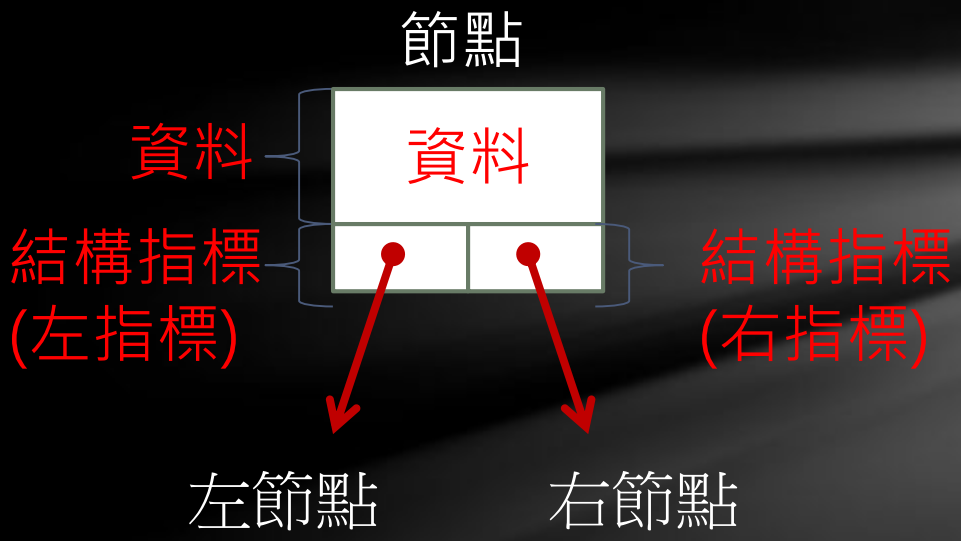
二元樹的節點編號擁有循序性，根節點1的子節點是節點2和節點3，節點2是4和5，依此類推可以得到節點編號的規則，如下所示：

- 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

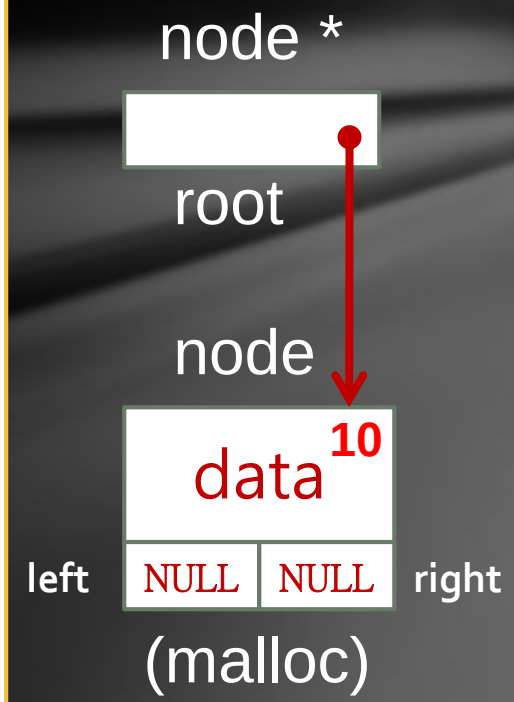
- 二元樹節點鏈結表示法



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

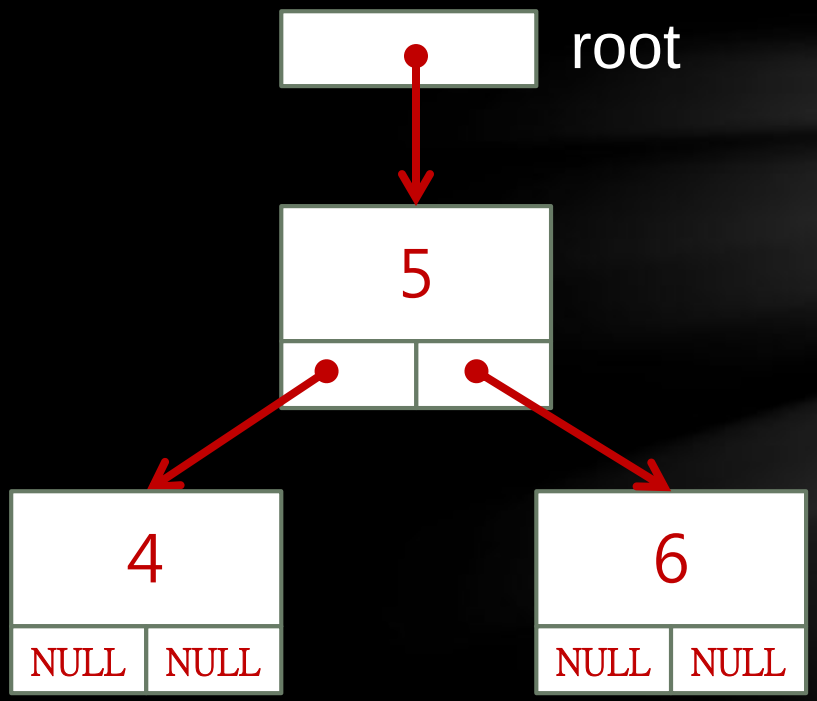
```
struct binary_tree_node
{
        資料型態 變數名稱;
        int data;
        struct binary_tree_node *left;
        struct binary_tree_node *right;
};
typedef struct binary_tree_node node;
...
node *root;
root = (node *)malloc(sizeof(node));
root ->data= 10;
root ->left= NULL;
root ->right= NULL;
```

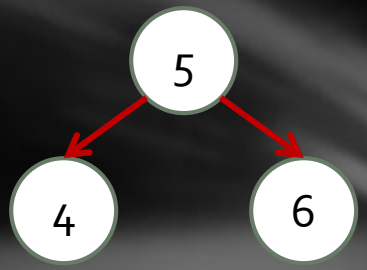


https://goo.gl/GKi3mC

小練習 (Binarry_Tree.c)

- 試著建立一個二元樹如下圖所示



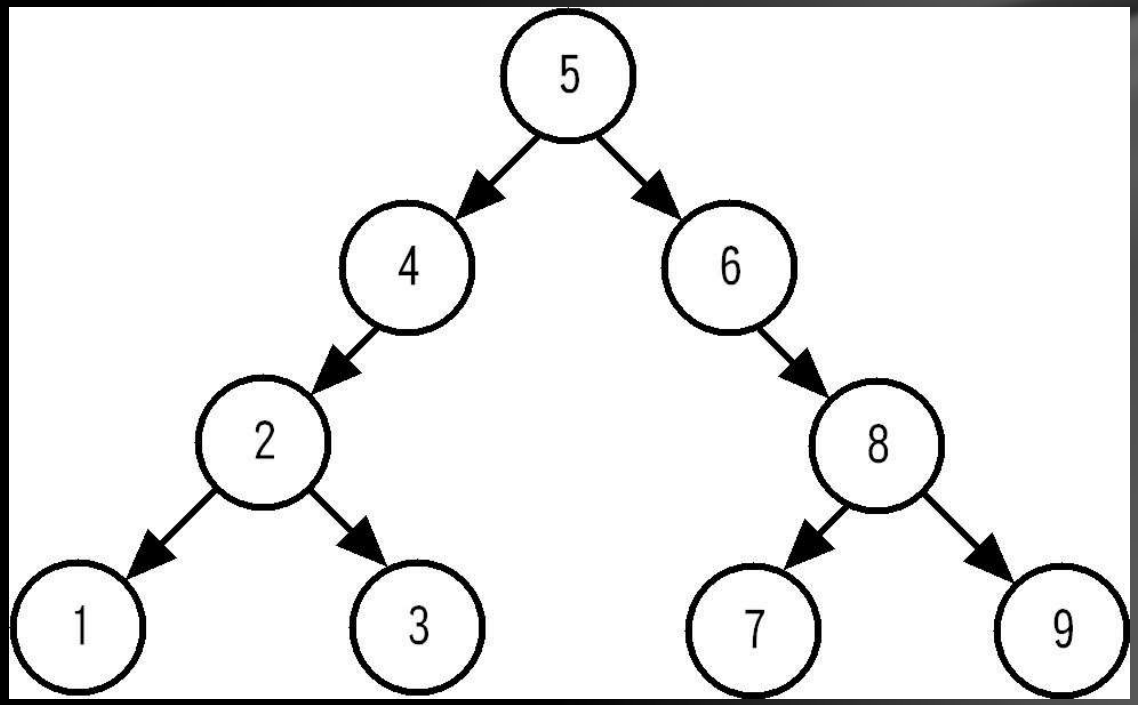



```
node *root;

// 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;
```

https://goo.gl/7sefTS

二元樹的走訪

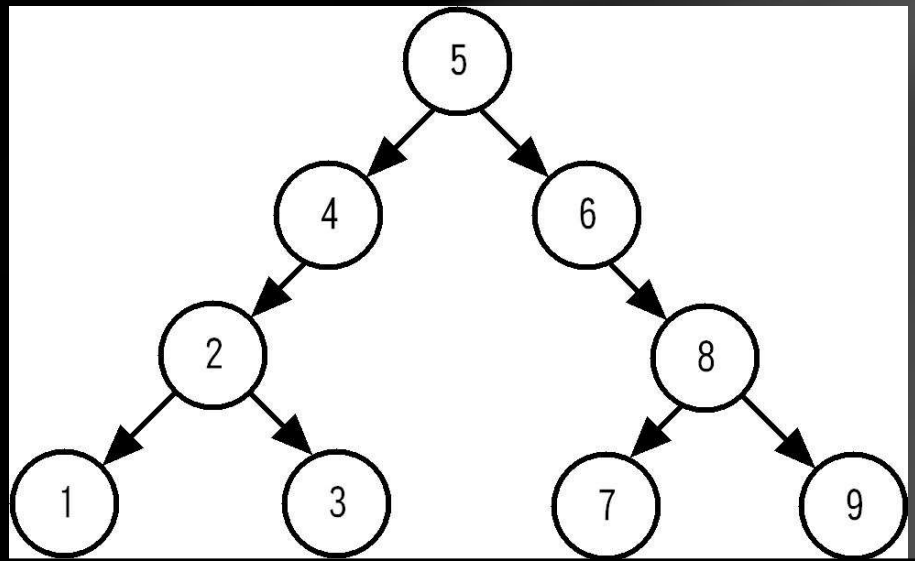
陣列和單向鏈結串列都只能從頭至尾或從尾至頭執行單向「走訪」（Traverse），不過二元樹的每一個節點都擁有指向左和右2個子節點的指標，所以走訪可以有兩條路徑。



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）。

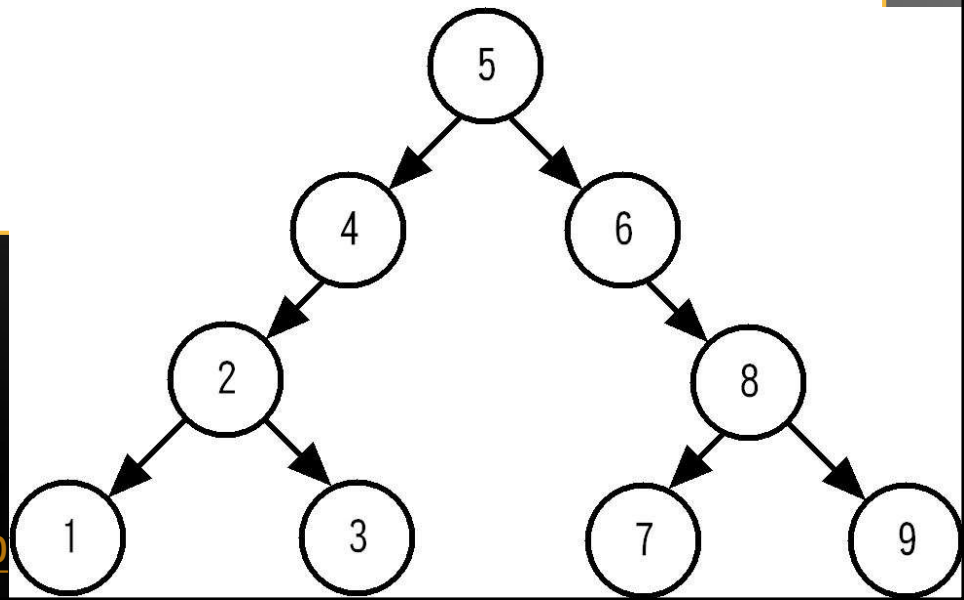


二元樹的走訪種類

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



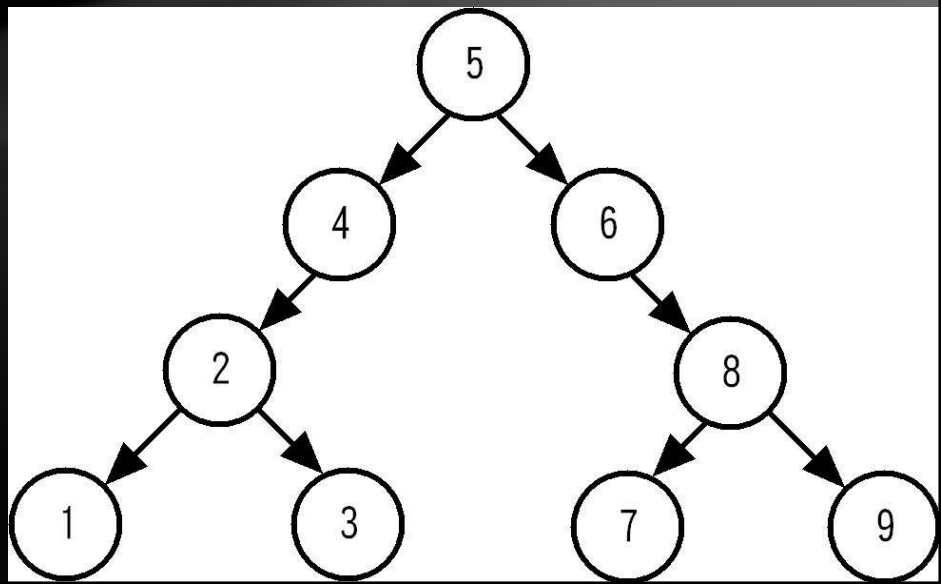
以從小到大方式列印

中序: 1,2,3,4,5,6,7,8,9

https://goo.gl/jhbd8o https://goo.gl/Q8XdCb

二元樹的走訪種類

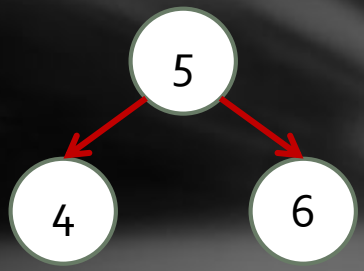
- 前序走訪（Preorder Traversal）
  - 每節點皆會比它的左子節點及右子節點先印
  - 左子節點又比右子節點先印 (自己->左->右)
  - **5,4,2,1,3,6,8,7,9**
- 後序走訪（Postorder Traversal）
  - 印自己前先印左節點
  - 再印右節點 自己最後
  - (左->右->自己)
  - **1,3,2,4,7,9,8,6,5**



小練習 (Binarry_Tree.c)

前序走訪（Preorder Traversal）

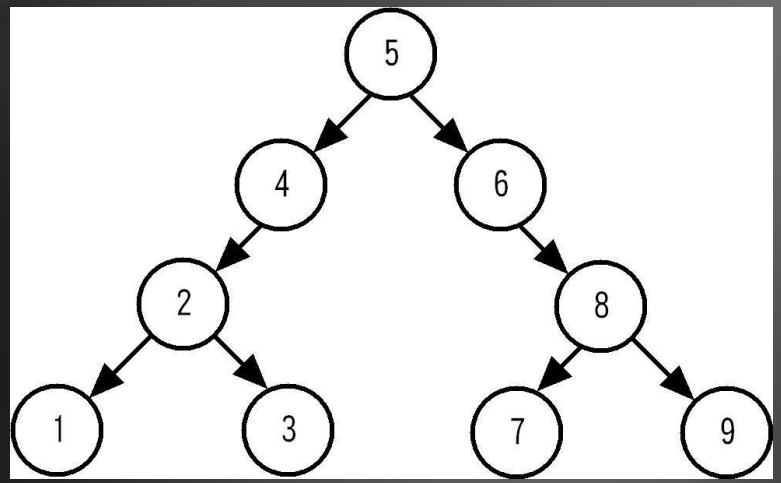
```
void print_preorder(node *ptr)
{
   if(ptr != NULL)
   {
      // 前序: 5,4,2,1,3,6,8,7,9
   }
}
```



https://jgirl.ddns.net/problem/0/2031
https://jgirl.ddns.net/problem/0/2039

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
   }
}
```

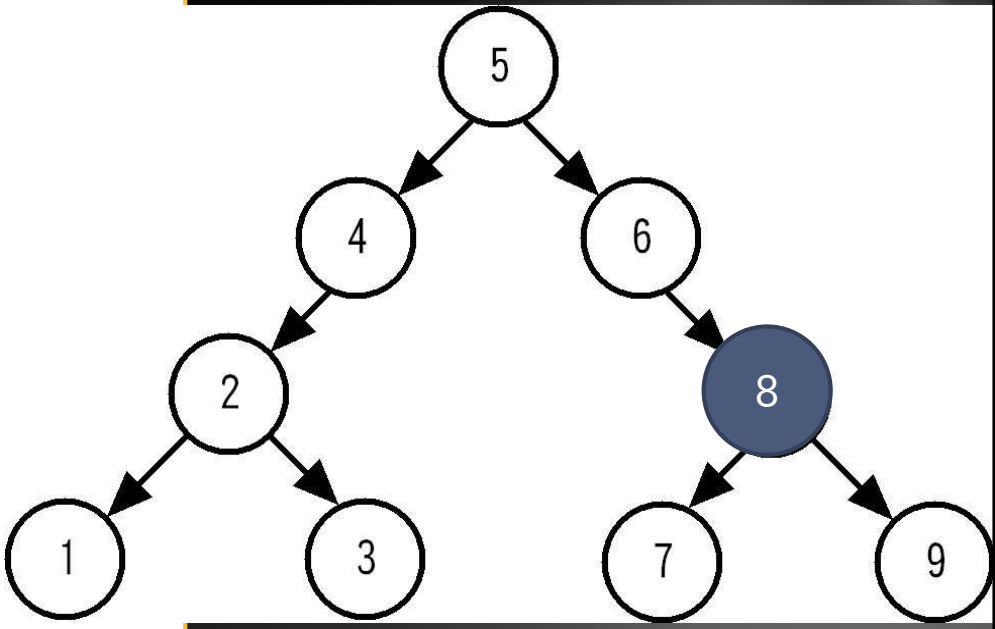


二元樹的搜尋 (Tree_LinearSearch.c)

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;
}
```

```
ptr = search_node(root, 8);
if(ptr!=NULL){
        printf("found %d\n",ptr->data);
}
else{
        puts("not found");
}
```



https://tinyurl.com/yaldgtu6

小練習

- 使用前序走訪實做搜尋
- 如何回收整顆樹的記憶體？

https://jgirl.ddns.net/problem/0/2040

https://goo.gl/CJQDCV

大綱

樹 (Tree)

二元樹 (Binary Tree)

二元搜尋樹 (Binary Search Tree)

二元搜尋樹

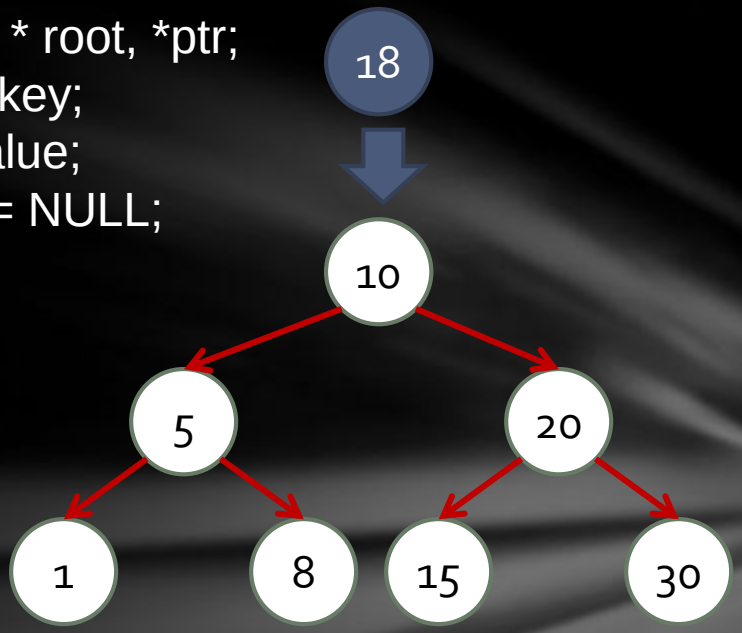
「二元搜尋樹」（Binary Search Trees）是一種二元樹，其節點資料的排列擁有一些特性，如下所示：

- 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

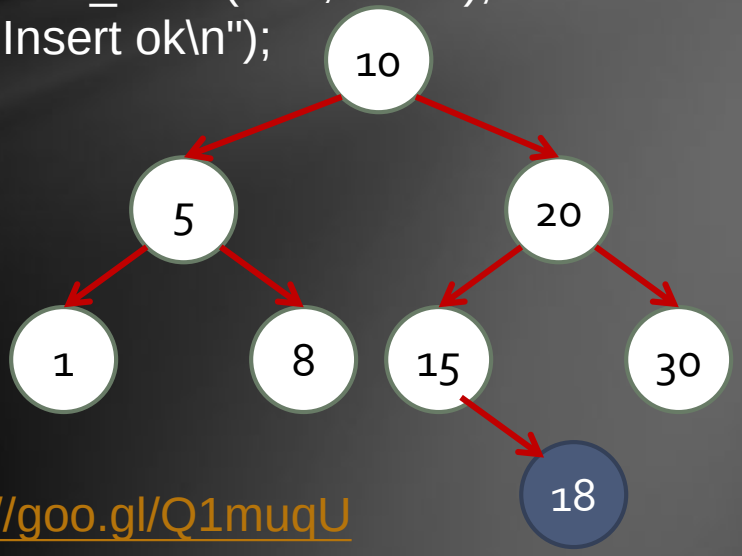
二元搜尋樹: 插入

```
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)
  {
    root = new_node; //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");
```



https://goo.gl/Q1muqU

小練習(BinarySearchTree.c)

```
m
10 5 1 8 20 15 30
請按任意鍵繼續 . . .
```

- 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插入節點
  (假設輸入之數字不會重覆) 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

```
n
1 8 5 15 30 20 10
請按任意鍵繼續 . . .
```

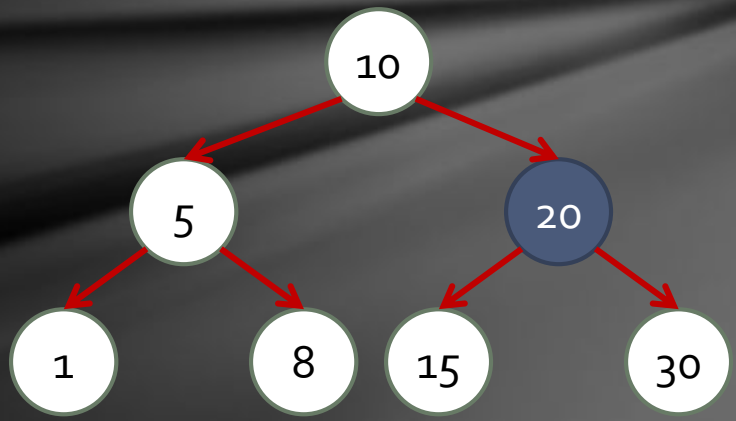
- 輸入m為前序，n為後序列印
- 試將新增節點函式中的while改成遞迴
- 並試著建構出如上頁圖中的二元樹
  - Hint:輸入的次序很重要，為什麼？

```
l
1 5 8 10 15 20 30
請按任意鍵繼續 . . .
```

二元搜尋樹: 搜尋

比較：一般二元樹的搜尋 (search_node())

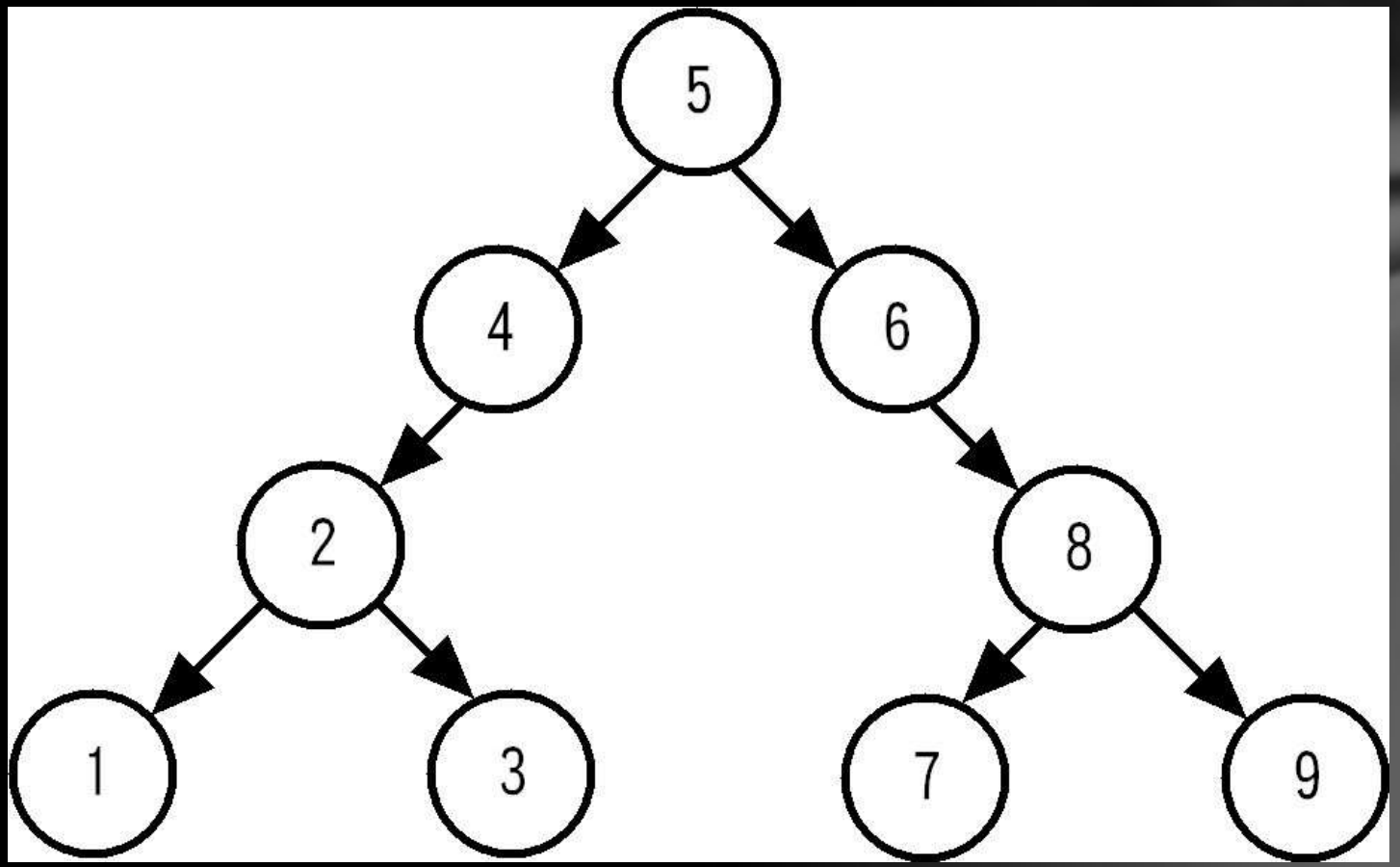
```
node *find_node(node *ptr, int value)
{
   while(ptr != NULL)
   {
      if(ptr->data == value)
         return ptr;
      else
      {
         if(ptr->data > value)
            ptr = ptr->left;
         else
            ptr = ptr->right;
      }
   }
   return NULL;
}
```




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

範例

https://tinyurl.com/ya5epfox



比較二種搜尋法

- 二元樹 vs 二元搜尋樹
- https://ideone.com/sNzF9f

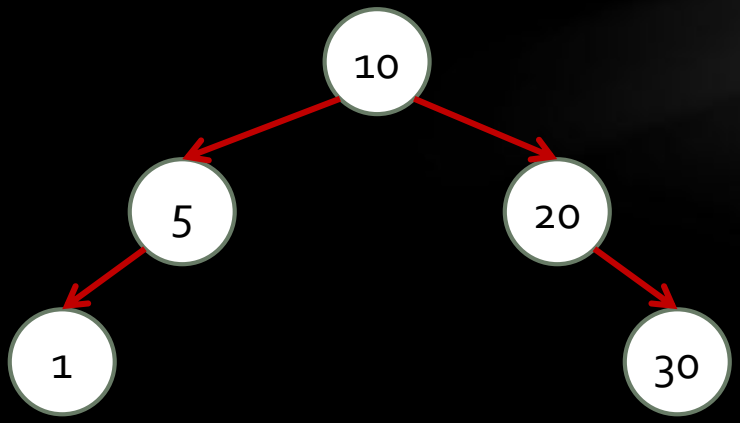
二元搜尋樹: 刪除

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)

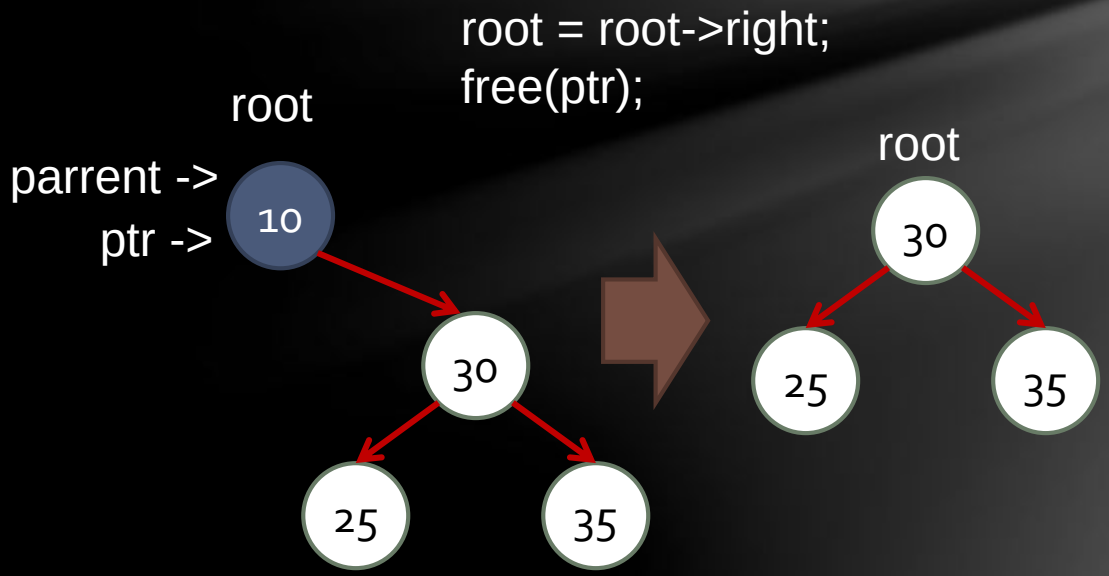



```
scanf("%d",&value);
ptr = find_node(root, value);
if(ptr != NULL){
        root= delete_node(root, value);
        printf("Delete ok\n");
}
else
        printf("Can not delete\n");
```

二元搜尋樹: 刪除

情況1: 節點沒有左子樹

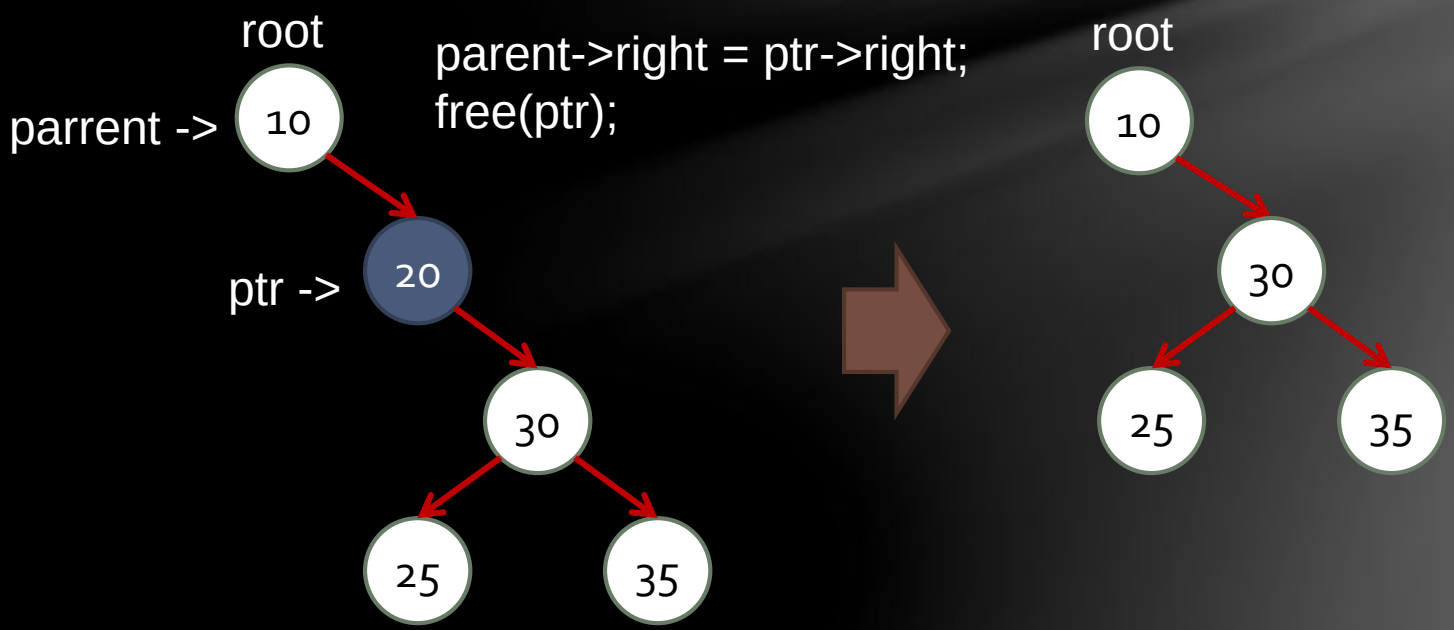
- 如果要刪的是根節點
- 要刪除的節點在父節點右方
- 要刪除的節點在父節點左方



二元搜尋樹: 刪除

情況1: 節點沒有左子樹

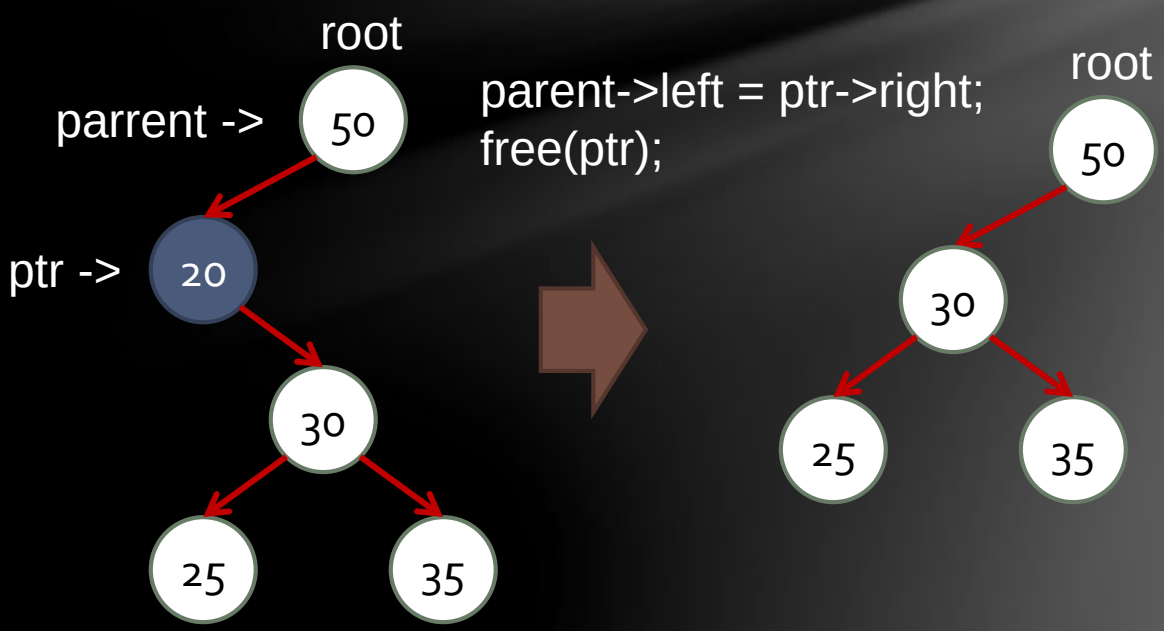
- 如果要刪的是根節點
- 要刪除的節點在父節點右方
- 要刪除的節點在父節點左方



二元搜尋樹: 刪除

情況1: 節點沒有左子樹

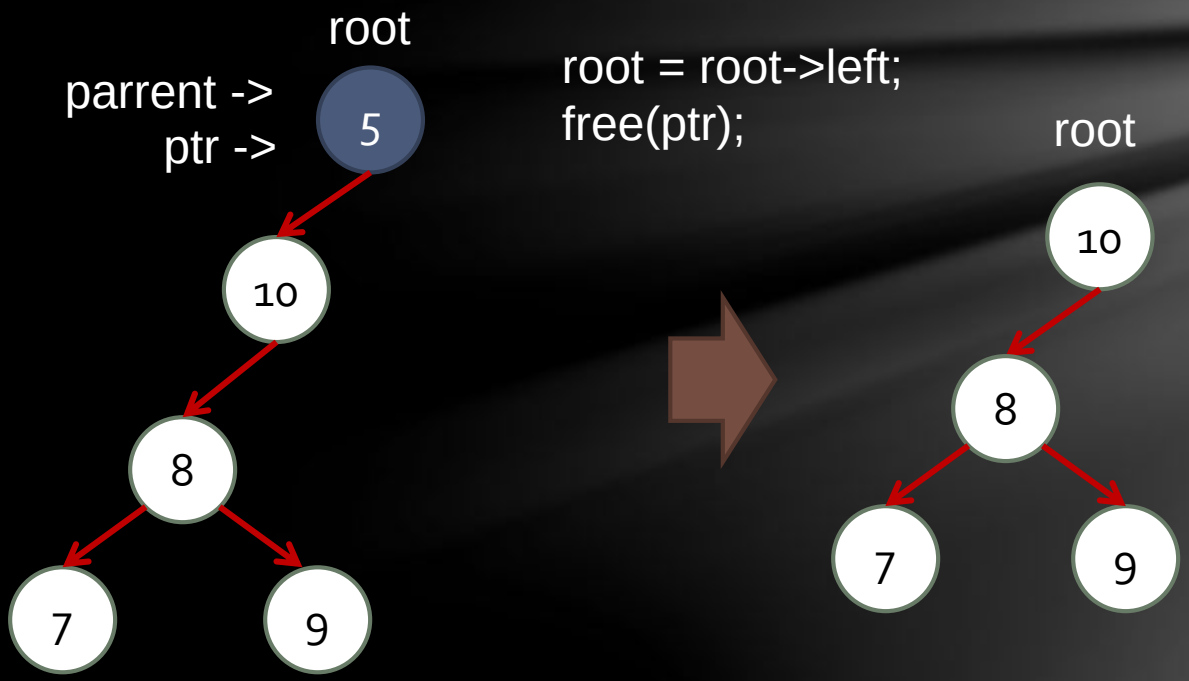
- 如果要刪的是根節點
- 其他
  - 要刪除的節點在父節點右方
  - 要刪除的節點在父節點左方



二元搜尋樹: 刪除

情況2: 節點沒有右子樹

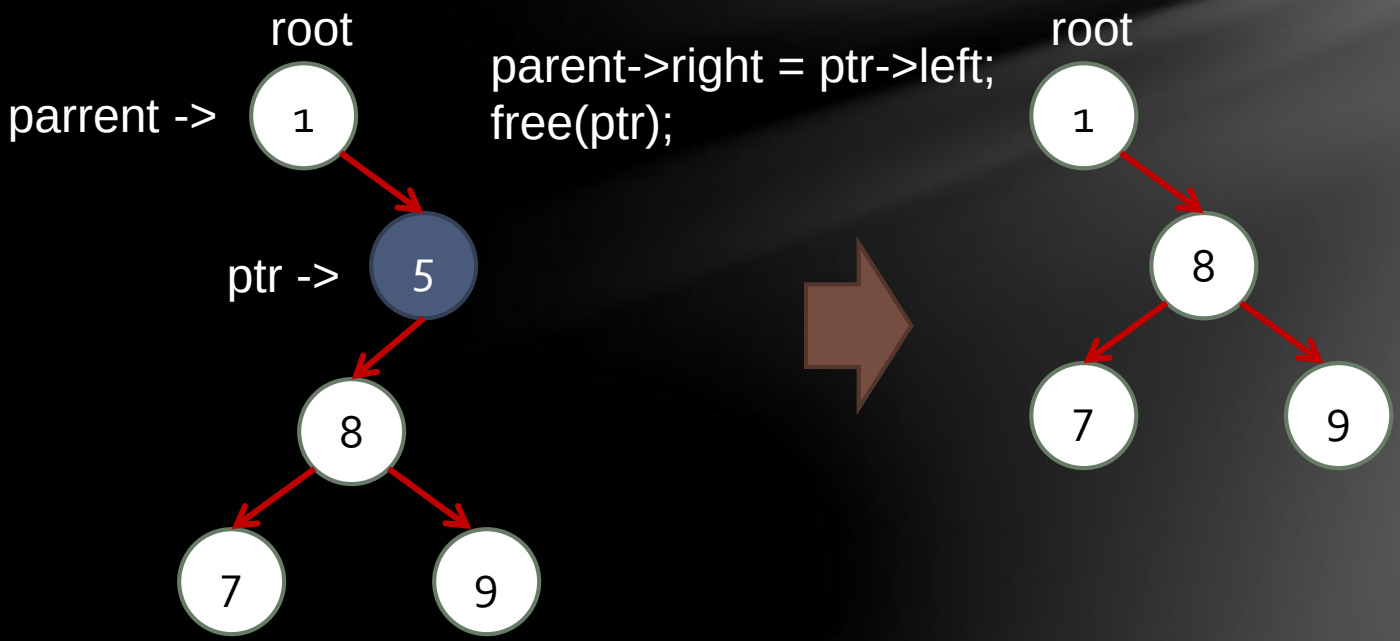
- 如果要刪的是根節點
- 其他



二元搜尋樹: 刪除

情況2: 節點沒有右子樹

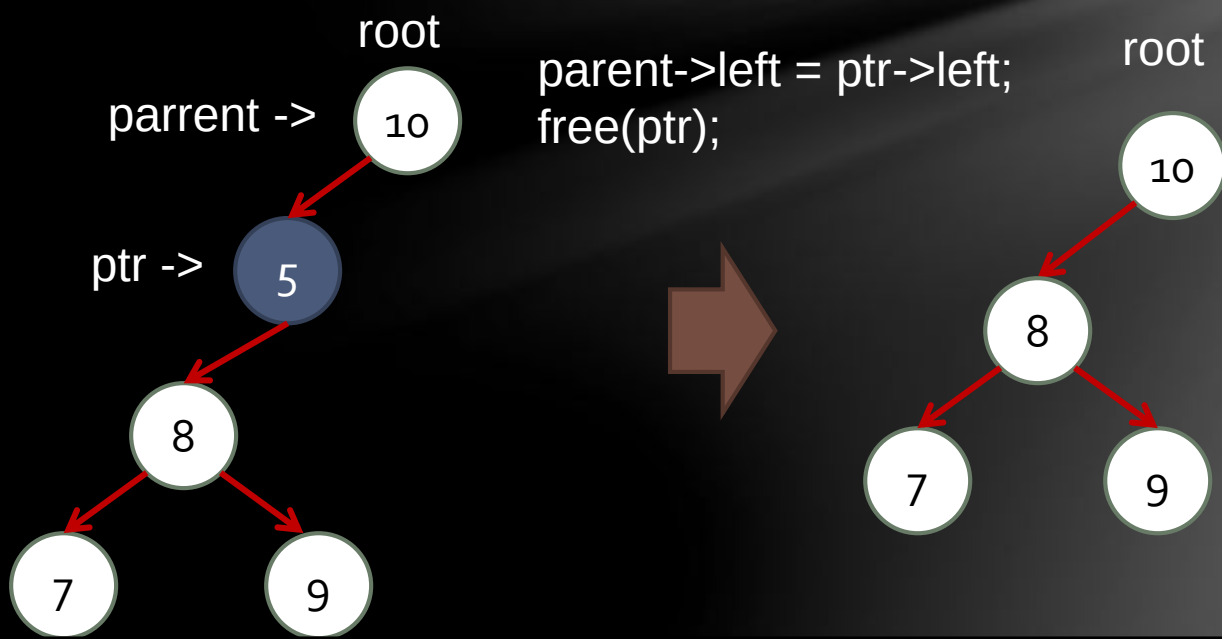
- 如果要刪的是根節點
- 其他
  - 要刪除的節點在父節點右方
  - 要刪除的節點在父節點左方



二元搜尋樹: 刪除

情況2: 節點沒有右子樹

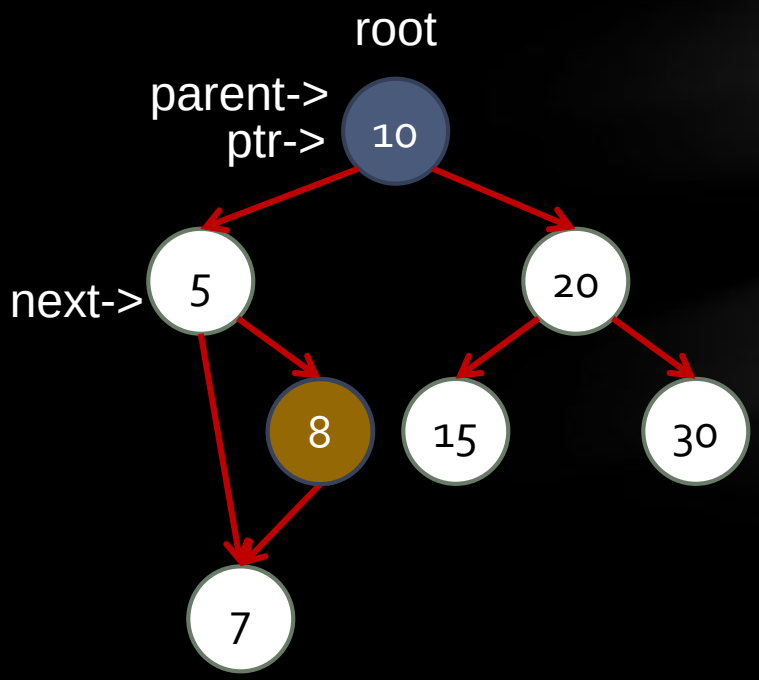
- 如果要刪的是根節點
- 其他
  - 要刪除的節點在父節點右方
  - 要刪除的節點在父節點左方



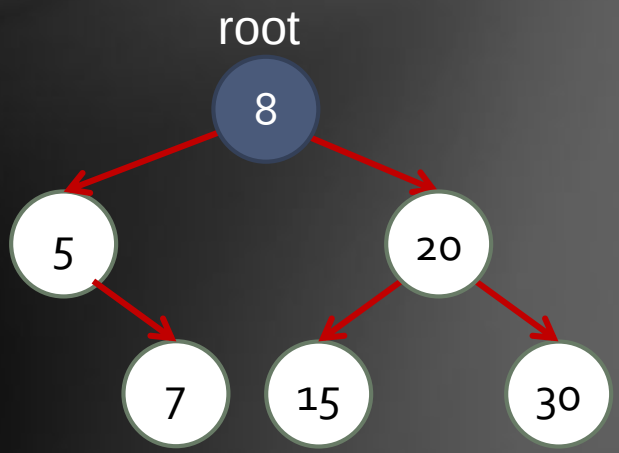
二元搜尋樹: 刪除

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
- 刪除10




```
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 = next->left;
}
ptr->data = next->data;     // 取代!!
free(next);       // 刪除next (注意: 不是刪除ptr)
```

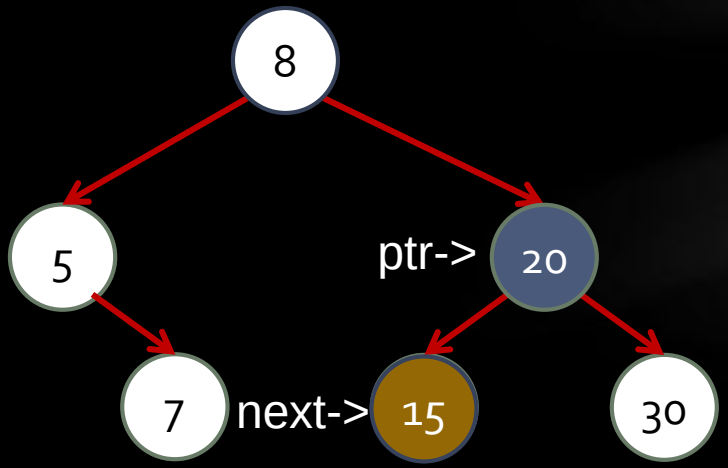


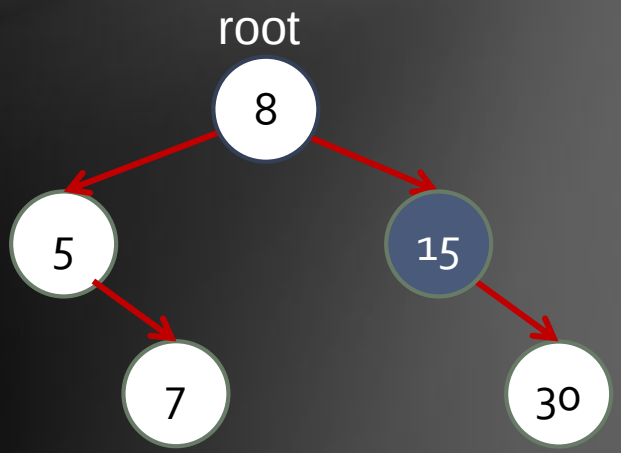
二元搜尋樹: 刪除

情況3: 節點有左右子樹

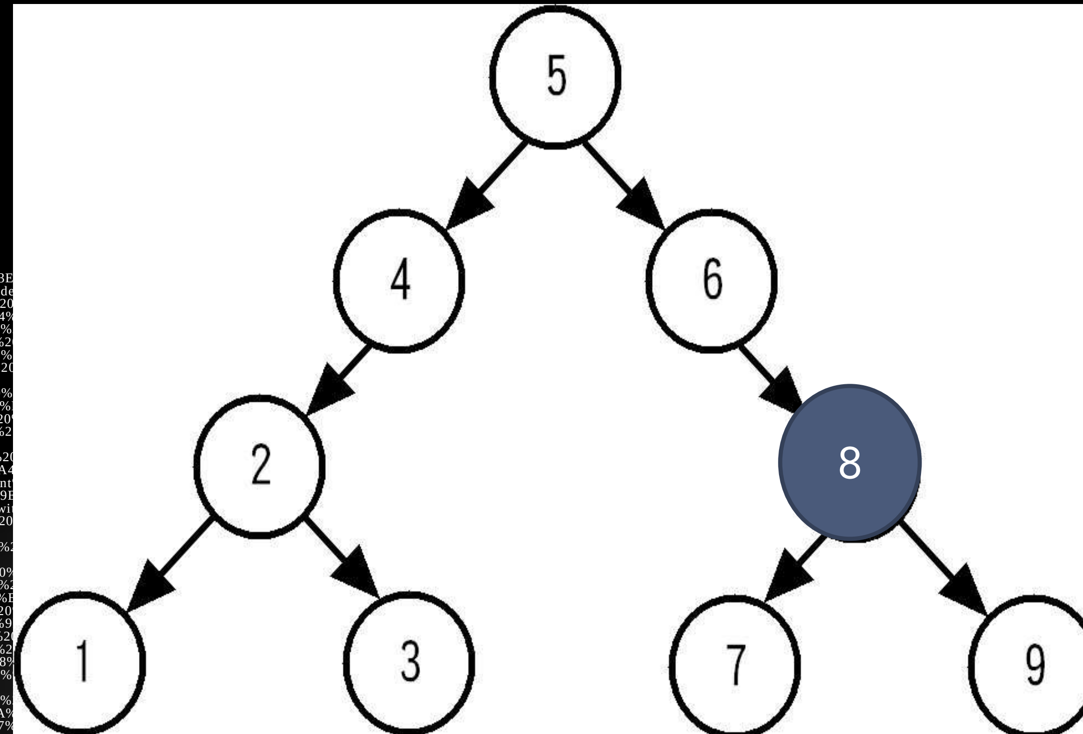
- 往左子節點之右子樹
- 找最大值當取代點
- 刪除20

```
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 = next->left;
}
ptr->data = next->data;     // 取代!!
free(next);      // 刪除next (注意: 不是刪除ptr)
```





修正問題



http://jgirl.ddns.net:88/VisualCode/visualize.html#code=%23include%3Cstdio.h%3E%0A%23include%3Cstdlib.h%3E
0A%20%20%20%20struct%20BTreeNode%20*right%3B%0A%7D%3B%0Atypedef%20struct%20BTreeNode%20node
%20%20%7B%0A%20%20%20%20%20%20%20%20print_inorder%28ptr-%3Eleft%29%3B%0A%20%20%20%20%20
%3Eright%29%3B%0A%20%20%20%20%7D%0A%7D%0A%0A//%20%E6%89%BE%E6%9F%90%E5%80%BC%E4%
%7B%0A%20%20%20%20node%20*parent%3B%0A%0A%20%20%20%20parent%20%3D%20ptr%3B%20%20%20%
ile%28ptr%20!%3D%20NULL%29%20%20%20%20%0A%20%20%20%20%7B%0A%20%20%20%20%20%20%20%2
%3Edata%20%3D%3D%20value%29%20%20%20%20//%20%E6%89%BE%E5%88%B0%E7%9B%AE%E6%A8%99%
2%B3%E6%AD%A4%E7%AF%80%E9%BB%9E%E4%B9%8B%E7%88%B6%E7%AF%80%E9%BB%9E%0A%20%20
%20%3D%20ptr%3B%0A%20%20%20%20%20%20%20%20%20%20%20%20if%28ptr-
%3Edata%20%3E%20value%29%0A%20%20%20%20%20%20%20%20%20%20%20%20%7B%0A%20%20%20%20%
1%3B%20%20%20%20%20%20%20%20%20%20%20%20//%20ptr%E5%9C%A8parent%E5%B7%A6%E5%AD%90%
%3Eleft%3B%20%20%20%20//%20%E5%BE%80%E5%B7%A6%E6%89%BE%0A%20%20%20%20%20%20%20%20%
20%7B%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20*pos%20%3D%201%3B%20%20%2
%20%20%20%20%20%20%20%20%20%20%20ptr%20%3D%20ptr-
%3Eright%3B%20%20%20%20//%20%E5%BE%80%E5%8F%B3%E6%89%BE%0A%20%20%20%20%20%20%20%20%20
%20%20%20%20//%20%E6%89%BE%E4%B8%8D%E5%88%B0%0A%7D%0A%0A//%20%E5%88%AA%E9%99%A4
parent%3B%0A%20%20%20%20node%20*ptr%3B%0A%20%20%20%20node%20*next%3B%0A%20%20%20%20int
5%BE%9Eroot%E9%96%8B%E5%A7%8B,%E6%89%BEvalue%E4%B9%8B%E7%88%B6%E7%AF%80%E9%BB%9E
A%96%E5%82%99%E5%88%AA%E9%99%A4%0A%20%20%20%20%7B%0A%20%20%20%20%20%20%20%20swit
E7%88%B6%E7%AF%80%E9%BB%9E%E5%B7%A6%E9%82%8A%E6%88%96%E5%8F%B3%E9%82%8A%0A%20
1%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20ptr%20%3D%20parent-
%3Eleft%3B%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20break%3B%0A%20%20%20%2
%20parent-
%3Eright%3B%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20break%3B%0A%20%20%20%
D%20parent%3B%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20break%3B%0A%20%20%2
%3Eleft%20%3D%3D%20NULL%29%20%20%20%20%20%20%20%20//%20%E6%83%85%E6%B3%811%3A%20%E
81%E5%88%AA%E7%9A%84%E6%98%AF%E6%A0%B9%E7%AF%80%E9%BB%9E%0A%20%20%20%20%20%20
A6%82%E6%9E%9C%E8%A6%81%E5%88%AA%E7%9A%84%E6%98%AF%E6%A0%B9%E7%AF%80%E9%BB%9
%3Eright%3B%0A%20%20%20%20%20%20%20%20%20%20%20%20else%20%20%20%20%20%20%20%20%2
20%20%20%20%20%20%20%20%20%20%20%20%20%20if%28%20pos%20%3D%3D%201%20%29%0A%20%20%2
%20%20%20%20//%E8%A6%81%E5%88%AA%E9%99%A4%E7%9A%84%E7%AF%80%E9%BB%9E%E5%9C%A8%
%E9%BB%9E%E7%9A%84%E5%8F%B3%E7%AF%80%E9%BB%9E%E6%8C%87%E5%90%91%E5%88%AA%E9%
0%20%20%20%20%20%20%20%20%20%20parent-%3Eright%20%3D%20ptr-
%3Eright%3B%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%7D%0A%20%20%20%20%
%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20//%E8%A6%81%E5%88%AA%
,%E6%89%80%E4%BB%A5%E5%B0%87%E7%88%B6%E7%AF%80%E9%BB%9E%E7%9A%84%E5%B7%A6%E7%
%AF%80%E9%BB%9E%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20parent-%3Eleft%20%3D%20ptr-
%3Eright%3B%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%7D%0A%20%20%20%20%20%20%20%20%20%20%20%20%7D%0A%20%20%20%20%20%20%20%20%20%20%20%20%20free%28ptr%29%3B%0A%20%20%20%20%20
%20%20%20%20%7D%0A%20%20%20%20%20%20%20%20else%20if%28ptr-
%3Eright%20%3D%3D%20NULL%29%20%20%20%20//%20%E6%83%85%E6%B3%812%3A%20%E7%AF%80%E9%BB%9E%E6%B2%92%E6%9C%89%E5%8F%B3%E5%AD%90%E6%A8%B9%0A%20%20%20%20%20%20%20%20%7B%0A%20%20%20%
20%20%20%20%20%20%20%20%20%20if%28parent%20!%3D%20ptr%29%0A%20%20%20%20%20%20%20%20%20%20%20%20%7B%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20if%28%20pos%20%3D%3D%201%20%29%
0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%7B%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20//%E8%A6%81%E5%88%AA%E9%99%A4%E7%9A%84%E7%AF%80%E9%BB%9E%
E5%9C%A8%E7%88%B6%E7%AF%80%E9%BB%9E%E5%8F%B3%E6%96%B9,%E6%89%80%E4%BB%A5%E5%B0%87%E7%88%B6%E7%AF%80%E9%BB%9E%E7%9A%84%E5%8F%B3%E7%AF%80%E9%BB%9E%E6%8C%87%E5%90%91%E5%8
8%AA%E9%99%A4%E7%AF%80%E9%BB%9E%E7%9A%84%E5%B7%A6%E7%AF%80%E9%BB%9E%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20parent-%3Eright%20%3D%20ptr-
%3Eleft%3B%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%7D%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20else%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%7B%0A
%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20//%E8%A6%81%E5%88%AA%E9%99%A4%E7%9A%84%E7%AF%80%E9%BB%9E%E5%9C%A8%E7%88%B6%E7%AF%80%E9%BB%9E%E5%B7%A6%E6%96%B9
,%E6%89%80%E4%BB%A5%E5%B0%87%E7%88%B6%E7%AF%80%E9%BB%9E%E7%9A%84%E5%B7%A6%E7%AF%80%E9%BB%9E%E6%8C%87%E5%90%91%E5%88%AA%E9%99%A4%E7%AF%80%E9%BB%9E%E7%9A%84%E5%B7%A6%E7
%AF%80%E9%BB%9E%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20parent-%3Eleft%20%3D%20ptr-
%3Eleft%3B%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%7D%0A%20%20%20%20%20%20%20%20%20%20%20%20%7D%0A%20%20%20%20%20%20%20%20%20%20%20%20else%0A%20%20%20%20%20%20%20%20%20%20%20%20
%20%20%20%20%20%20%20%20%20root%20%3D%20root-
%3Eleft%3B%0A%20%20%20%20%20%20%20%20%20%20%20%20free%28ptr%29%3B%0A%20%20%20%20%20%20%20%20%7D%0A%20%20%20%20%20%20%20%20else%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20
%20%20%20%20%20%20//%20%E6%83%85%E6%B3%813%3A%20%E7%AF%80%E9%BB%9E%E6%9C%89%E5%B7%A6%E5%8F%B3%E5%AD%90%E6%A8%B9%0A%20%20%20%20%20%20%20%20%7B%0A%20%20%20%20%20%20%20%20%20%20%20%20
%20%20%20parent%20%3D%20ptr%3B%0A%20%20%20%20%20%20%20%20%20%20%20%20next%20%3D%20ptr-
%3Eleft%3B%20%20%20%20//%20%E6%89%BE%E5%8F%96%E4%BB%A3%E9%BB%9Enext,%20%E5%BE%9Eptr%E5%B7%A6%E9%82%8A%E9%96%8B%E5%A7%8B%E6%89%BE%0A%20%20%20%20%20%20%20%20%20%20%20%20while%28n
ext-
%3Eright%20!%3D%20NULL%29%20%20%20%20//%20%E5%BE%80%E5%B7%A6%E5%AD%90%E7%AF%80%E9%BB%9E%E4%B9%8B%E5%8F%B3%E5%AD%90%E6%A8%B9%E6%89%BE%E6%9C%80%E5%A4%A7%E5%80%BC%E7%95%B6%
E5%8F%96%E4%BB%A3%E9%BB%9E%0A%20%20%20%20%20%20%20%20%20%20%20%20%7B%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20parent%20%3D%20next%3B%20%20%20%20%20%20%20%20//%20parent%E
7%82%BAnext%E4%B9%8B%E7%88%B6%E7%AF%80%E9%BB%9E%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20next%20%3D%20next-
%3Eright%3B%0A%20%20%20%20%20%20%20%20%20%20%20%20%7D%0A%20%20%20%20%20%20%20%20%20%20%20%20ptr-%3Edata%20%3D%20next-
%3Edata%3B%20%20%20%20%20%20%20%20//%20%E5%8F%96%E4%BB%A3!!%0A%20%20%20%20%20%20%20%20%20%20%20%20parent-%3Eright%20%3D%20next-
%3Eleft%3B%20//%20%E7%B9%9E%E9%81%8Enext%E7%AF%80%E9%BB%9E%0A%20%20%20%20%20%20%20%20%20%20%20%20free%28next%29%3B%20%20%20%20//%20%E5%88%AA%E9%99%A4next%20%28%E6%B3%A8%E6%84%8F%3
A%20%E4%B8%8D%E6%98%AF%E5%88%AA%E9%99%A4ptr%29%0A%20%20%20%20%20%20%20%20%7D%0A%20%20%20%20%7D%0A%20%20%20%20return%20root%3B%20%20%20%20//%20%E5%9B%9E%E5%82%B3%E6%AD%A4%E6%A
8%B9%0A%7D%0Aint%20main%28%29%0A%7B%0A%20%20%20%20node%20*root%20%3D%20NULL%3B%0A%20%20%20%20node%20*ptr%3B%0A%20%20%20%20%20%20%20%0A%20%20%20%20//Level%201%0A%20%20%20%20root%2
0%3D%20%28node%20*%29malloc%28sizeof%28node%29%29%3B%0A%20%20%20%20root-%3Edata%20%3D%205%3B%0A%20%20%20%20%20%0A%20%20%20%20//Level%202%0A%20%20%20%20root-
%3Eleft%20%3D%20%28node%20*%29malloc%28sizeof%28node%29%29%3B%0A%20%20%20%20root-%3Eleft-%3Edata%20%3D%204%3B%0A%20%20%20%20root-%3Eleft-%3Eleft%20%3D%20NULL%3B%0A%20%20%20%20root-%3Eleft-
%3Eright%20%3D%20NULL%3B%0A%20%20%20%20%20%0A%20%20%20%20root-%3Eright%20%3D%20%28node%20*%29malloc%28sizeof%28node%29%29%3B%0A%20%20%20%20root-%3Eright-
%3Edata%20%3D%206%3B%0A%20%20%20%20root-%3Eright-%3Eleft%20%3D%20NULL%3B%0A%20%20%20%20root-%3Eright-
%3Eright%20%3D%20NULL%3B%0A%20%20%20%20%20%0A%20%20%20%20//Level%203%0A%20%20%20%20root-%3Eleft-%3Eleft%20%3D%20%28node%20*%29malloc%28sizeof%28node%29%29%3B%0A%20%20%20%20root-%3Eleft-%3Eleft-
%3Edata%20%3D%202%3B%0A%20%20%20%20root-%3Eleft-%3Eleft-%3Eleft%20%3D%20NULL%3B%0A%20%20%20%20root-%3Eleft-%3Eleft-%3Eright%20%3D%20NULL%3B%0A%20%20%20%20%20%0A%20%20%20%20root-%3Eright-
%3Eright%20%3D%20%28node%20*%29malloc%28sizeof%28node%29%29%3B%0A%20%20%20%20root-%3Eright-%3Eright-%3Edata%20%3D%208%3B%0A%20%20%20%20root-%3Eright-%3Eright-
%3Eleft%20%3D%20NULL%3B%0A%20%20%20%20root-%3Eright-%3Eright-%3Eright%20%3D%20NULL%3B%0A%20%20%20%20%20%0A%20%20%20%20//Level%204%0A%20%20%20%20root-%3Eleft-%3Eleft-
%3Eleft%3D%20%28node%20*%29malloc%28sizeof%28node%29%29%3B%0A%20%20%20%20root-%3Eleft-%3Eleft-%3Eleft-%3Edata%20%3D%201%3B%0A%20%20%20%20root-%3Eleft-%3Eleft-%3Eleft-
%3Eleft%20%3D%20NULL%3B%0A%20%20%20%20root-%3Eleft-%3Eleft-%3Eleft-%3Eright%20%3D%20NULL%3B%0A%20%20%20%20%20%0A%20%20%20%20root-%3Eleft-%3Eleft-
%3Eright%3D%20%28node%20*%29malloc%28sizeof%28node%29%29%3B%0A%20%20%20%20root-%3Eleft-%3Eleft-%3Eright-%3Edata%20%3D%203%3B%0A%20%20%20%20root-%3Eleft-%3Eleft-%3Eright-
%3Eleft%20%3D%20NULL%3B%0A%20%20%20%20root-%3Eleft-%3Eleft-%3Eright-%3Eright%20%3D%20NULL%3B%0A%20%20%20%20%20%0A%20%20%20%20root-%3Eright-%3Eright-
%3Eleft%20%3D%20%28node%20*%29malloc%28sizeof%28node%29%29%3B%0A%20%20%20%20root-%3Eright-%3Eright-%3Eleft-%3Edata%20%3D%207%3B%0A%20%20%20%20root-%3Eright-%3Eright-%3Eleft-
%3Eleft%20%3D%20NULL%3B%0A%20%20%20%20root-%3Eright-%3Eright-%3Eleft-%3Eright%20%3D%20NULL%3B%0A%20%20%20%20%20%0A%20%20%20%20root-%3Eright-%3Eright-
%3Eright%20%3D%20%28node%20*%29malloc%28sizeof%28node%29%29%3B%0A%20%20%20%20root-%3Eright-%3Eright-%3Eright-%3Edata%20%3D%209%3B%0A%20%20%20%20root-%3Eright-%3Eright-%3Eright-
%3Eleft%20%3D%20NULL%3B%0A%20%20%20%20root-%3Eright-%3Eright-%3Eright-
%3Eright%20%3D%20NULL%3B%0A%20%20%20%20%20%0A%20%20%20%20root%20%3D%20delete_node%28root,%204%29%3B%0A%20%20%20%20root%20%3D%20delete_node%28root,%206%29%3B%0A%20%20%20%20//root%20%3D%20delete_
node%28root,%203%29%3B%0A%20%20%20%20root%20%3D%20delete_node%28root,%205%29%3B%0A%20%20%20%20root%20%3D%20delete_node%28root,%208%29%3B%0A%20%20%20%20%20%0A%20%20%20%20print_inorder%28root%29%3B%
0A%20%20%20%20%0A%20%20%20%20return%200%3B%0A%7D&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=c&rawInputLstJSON=%5B%5D&textReferences=false

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

延申閱讀

- 二元樹的旋轉
- 平衡樹 https://zh.wikipedia.org/wiki/%E5%B9%B3%E8%A1%A1%E6%A0%91
- AVL樹 https://zh.wikipedia.org/wiki/AVL%E6%A0%91
- 紅黑樹 https://zh.wikipedia.org/wiki/%E7%BA%A2%E9%BB%91%E6%A0%91
  https://zhuanlan.zhihu.com/p/31805309