Algorithms
Chapter 12 Binary Search Trees

Associate Professor: Ching-Chi Lin
林清池 副教授
chingchi.lin@gmail.com

Department of Computer Science and Engineering
National Taiwan Ocean University

Outline

- **What is a binary search tree?**
- Querying a binary search tree
- Insertion and deletion

Overview

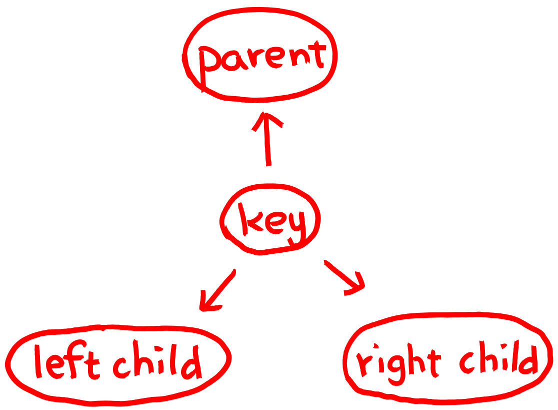
search trees: a. search
b. min, max
c. predecessor, successor
d. insert, delet

- **Search trees** are data structures that support many dynamic-set operations.
  - Dynamic-set operations includes SEARCH, MINIMUM, MAXIMUM, PREDECESSOR, SUCCESSOR, INSERT, and DELETE.
- Can be used as both a dictionary (a, d) and as a priority (b, d) queue.
- Basic operations take time proportional to the height of the tree, i.e., $\Theta(h)$. 基本動作時間: $\theta(h)$ ⇒ 樹高
  - For complete binary tree with n nodes: worst case $\Theta(\lg n)$.
  - For linear chain of n nodes: worst case $\Theta(n)$. 從 $\theta(\log n) \sim \theta(n)$
- Different types of search trees include **binary search trees**, **red-black trees** (Chapter 13), and **B-trees** (Chapter 18).

可控制高度

Binary search trees

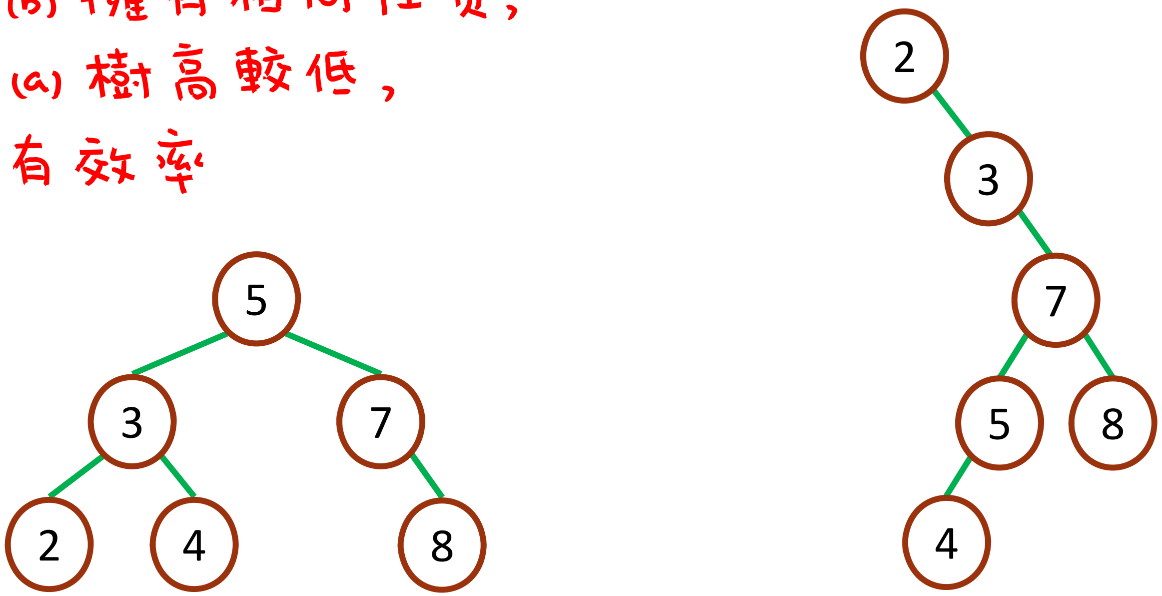
- We represent a binary tree by a linked data structure in which each node is an object.
- Each node contains the fields
  - *key* and possibly other satellite data.
  - *left*: points to left child.
  - *right*: points to right child.
  - *p*: points to parent. $p[root[T]]$ = NIL.



- Stored keys must satisfy the **binary-search-tree property**.
  - If y is in left subtree of x, then $key[y] < key[x]$. 左子樹 < 我
  - If y is in right subtree of x, then $key[y] \geq key[x]$. 右子樹 ≥ 我

Figure 12.1 Binary search trees

(a)(b)擁有相同性質，
但(a)樹高較低，
較有效率



(a) (b)

- A binary search tree on 6 nodes with height 2.
- A less efficient binary search tree with height 4 that contains the same keys.

Inorder tree walk

有了binary search tree的性質，
我們可將key值由小到大印出

- The binary-search-tree property allows us to print keys in a binary search tree in order, recursively.

- Elements are printed in monotonically increasing order.

INORDER-TREE-WALK(x)
1. **if** $x \neq$ NIL
2. INORDER-TREE-WALK($left[x]$) 印左子樹
3. print $key[x]$ 印自己
4. INORDER-TREE-WALK($right[x]$) 印右子樹

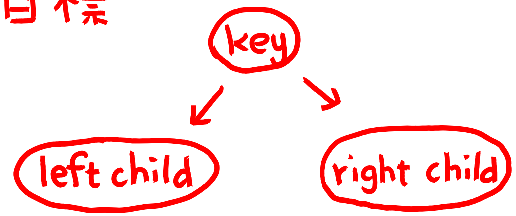
- The inorder tree walk prints the keys in each of the two binary search trees from Figure 12.1 in the order 2, 3, 4, 5, 7, 8.

Properties of binary search trees

- **Theorem** If x is the root of an n-node subtree, then the call INORDER-TREE-WALK(x) takes $\Theta(n)$.

Proof: inorder_tree_walk(x) 執行時間為 $\theta(n)$

- $T(0) = c$, as It takes constant time on an empty subtree.
- Left subtree has k nodes and right subtree has $n-k-1$ nodes.
- d : the time to execute INORDER-TREE-WALK(x), exclusive of the time spent in recursive calls. 假設左樹有k個点，右樹有n-k-1個点
- Prove by substitution method: $T(n)=(c+d)n+c$. ← 目標
- For $n = 0$, we have $(c+d)\cdot 0+c=c=T(0)$.
- For $n > 0$, $T(n) = T(k) + T(n-k-1)+d$



$$= ((c+d)k+c)+((c+d)(n-k-1)+c)+d$$
$$= (c+d)n+c-(c+d)+c+d$$
$$= (c+d)n+c.$$

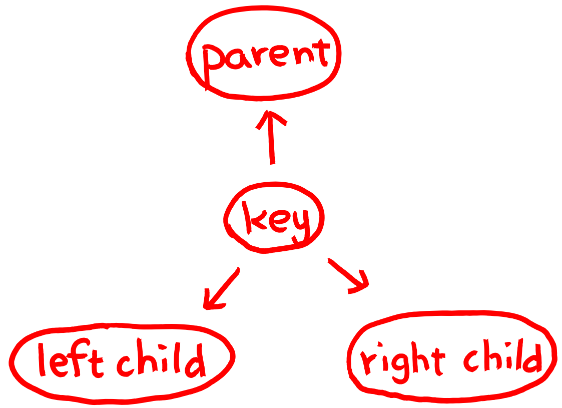
c：當樹為空所需時間

d：只執行1, 3行時間

$T(n)$ = 印左子樹所需時間 + 印右子樹所需時間 + 執行1, 3行時間

Outline

- What is a binary search tree?
- **Querying a binary search tree**
- Insertion and deletion



c：當樹為空所需時間

d：只執行 1, 3 行時間

$T(n) = dn + c(n+1) = (c+d)n + c$

節點 ⇒ 有 n 個

子節點指標為空 ⇒ 共有 n+1 個

I. 每個點都有父節點，root 沒有 ⇒ 有 n-1 個子節點

II. 每個點配置 2 個 ⇒ $n*$(左子 + 右子) $= 2n$

III. 為空 = 全 - 用到 $= 2n-(n-1) = n+1$

Operations on binary search trees

- We shall examine SEARCH, MINIMUM, MAXIMUM, SUCCESSOR, and PREDECESSOR operations. 時間≤樹高的常數倍
- The running times of these operations are all $O(h)$.

TREE-SEARCH(x, k) 遞迴版

if x = NIL or k = key[x]
    then return x
if k < key[x]
    then return TREE-SEARCH(*left*[x], k)
    else return TREE-SEARCH(*right*[x], k)

非遞迴 → 迴圈

ITERATIVE-TREE-SEARCH(x, k)

while $x \neq$ NIL and $k \neq$ key[x]
    if k < key[x]
        then $x \leftarrow$ *left*[x]
        else $x \leftarrow$ *right*[x]
return x

雖然都是≤樹高常數倍，但非遞迴較快

- On most computers, the iterative version is more efficient.
  遞迴時要將目前的資料存到stack，返迴時要pop (compiler做的)
- **Time:** The algorithm visiting nodes on a downward path from the root. Thus, running time is $O(h)$.

Minimum and maximum

- The binary-search-tree property guarantees that
  - the minimum key of a binary search tree is located at the leftmost node, and 最小：最左的点
  - the maximum key of a binary search tree is located at the rightmost node. 最大：最右的点
- Traverse the appropriate pointers (*left* or *right*) until NIL is reached.

TREE-MINIMUM(x)

```
while left[x] ≠ NIL
    do x ← left[x]
return x
```

TREE-MAXIMUM(x)

```
while right[x] ≠ NIL
    do x ← right[x]
return x
```

- **Time:** Both procedures visit nodes that form a downward path from the root to a leaf. Both procedures run in $O(h)$ time.

Successor and predecessor$_{1/2}$

- Assuming that all keys are distinct, the successor of a node *x* is the node *y* such that *key*[*y*] is the smallest key > *key*[*x*].
- The structure of a binary search tree allows us to determine the successor of a node without ever comparing keys.
- If *x* has the largest key in the binary search tree, then we say that *x*'s successor is NIL.
- There are two cases:
  - If node *x* has a non-empty right subtree, then *x*'s successor is the minimum in *x*'s right subtree.
  - If node *x* has an empty right subtree and *x* has a successor *y*, then *y* is the lowest ancestor of *x* whose left child is also an ancestor of *x*.

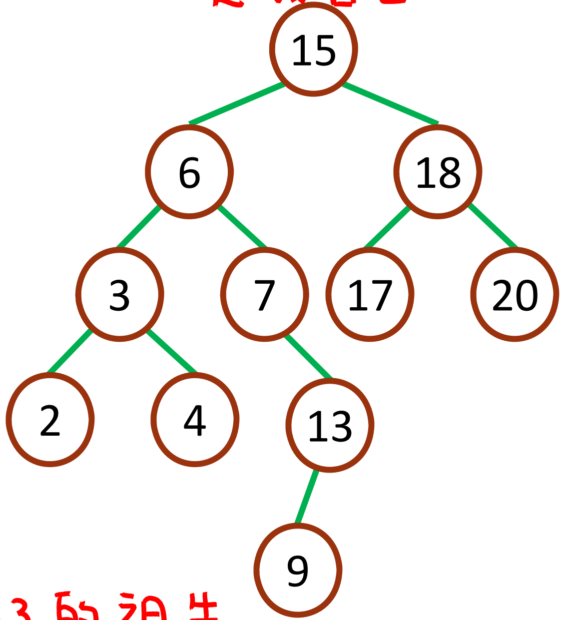
右子樹 < 不空：右子樹是最小的那個
空：最小祖先滿足其左子
I. 也是祖先 or
II. 是我自己

Successor and predecessor$_{2/2}$

```
TREE-SUCCESSOR(x)
    if right[x] ≠ NIL
        return TREE-MINIMUM(right[x])
    y ← p[x]
    while y ≠ NIL and x = right[y]
        x ← y
        y ← p[y]
    return y
```

不空

空



13的左子是9自己　15的左子是6，6也是13的祖先

- The successor of the node with key 13 is the node with key 15.
- **Time:** Since we either follow a path up the tree or follow a path down the tree. The running time is $O(h)$.
- TREE-PREDECESSOR is symmetric to TREE-SUCCESSOR.

Outline

- What is a binary search tree?
- Querying a binary search tree
- **Insertion and deletion**

Insertion and deletion insert 和 delete 会改变樹

- The operations of insertion and deletion cause the dynamic set represented by a binary search tree to change.
- The binary-search-tree property must hold after the change.
- Insertion is more straightforward than deletion.

  insert 比較容易

  在做完 insert 和 delete 後, binary-search-tree 的性质要維持下去

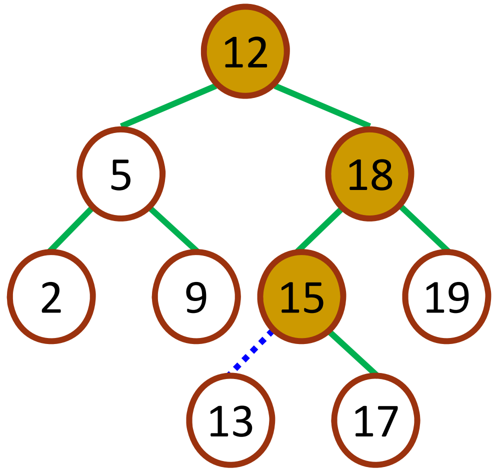
Insertion

```
TREE-INSERT(T, z)
    y ← NIL; x ← root[T ]
    while x ≠ NIL
        y ← x
        if key[z] < key[x]
            x ← left[x]
        else x ← right[x]
    p[z] ← y
    if y = NIL
        root[T ] ← z        /* Tree T was empty */
    else if key[z] < key[y]
        left[y] ← z
    else right[y] ← z
```

找父親y

比父親大放右

比父親小放左



Inserting an item with key 13

- **Time:** Since we follow a path down the tree. The running time is $O(h)$.

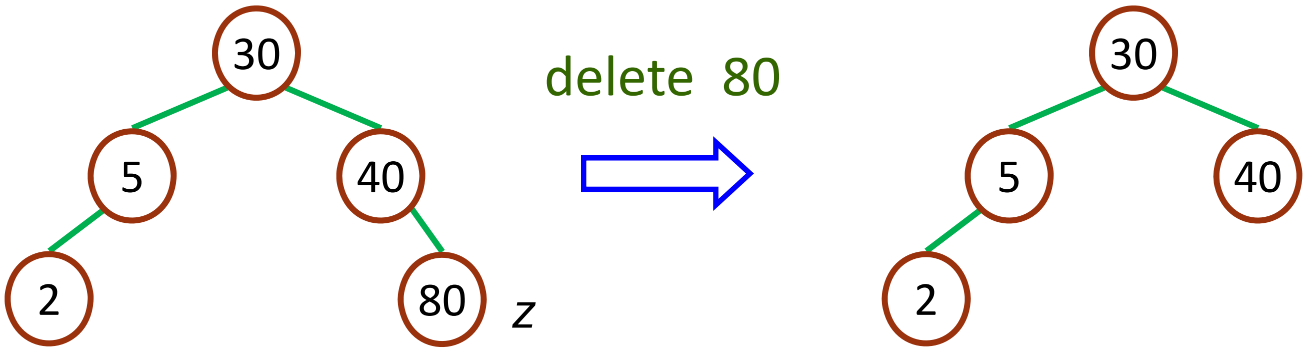
Deletion

- TREE-DELETE is broken into three cases.
- **Case 1:** *z* has no children. 沒有兒子，直接刪
  - Delete *z* by making the parent of *z* point to NIL, instead of to *z*.
- **Case 2:** *z* has one child. 有一個兒子，將兒子連到父親
  - Delete *z* by making the parent of *z* point to *z*'s child, instead of to *z*.
- **Case 3:** *z* has two children.
  - *z*'s successor *y* has either no children or one child.
    (*y* is the minimum node with no left child in *z*'s right subtree.)
  - Delete *y* from the tree (via Case 1 or 2).
  - Replace *z*'s key and satellite data with *y*'s.

①找下一個
②將下一個 copy 過來
③將下一個刪除

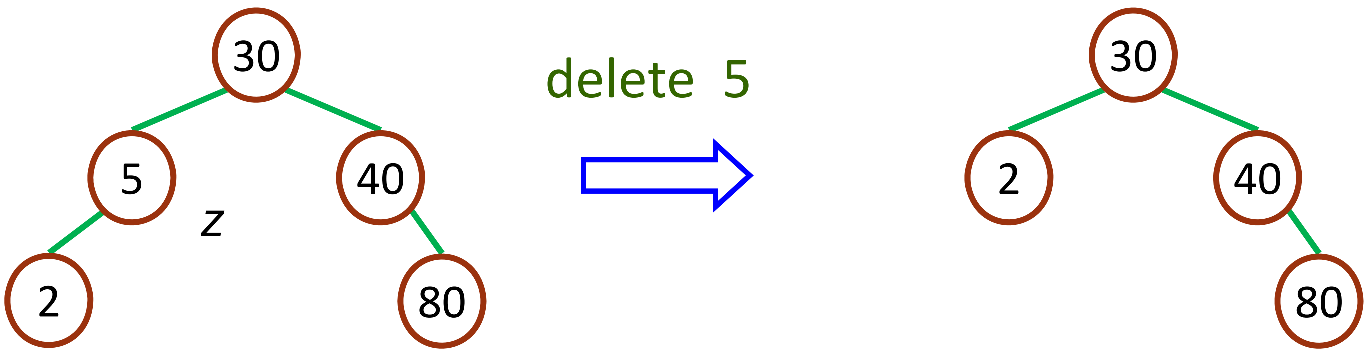
可以證明下一個最多只有一個兒子
⇒最多 delete 2 次

Case 1:



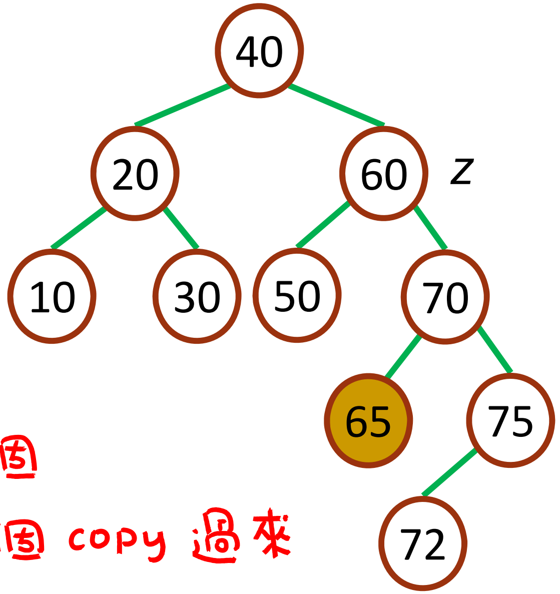
沒有兒子，直接刪

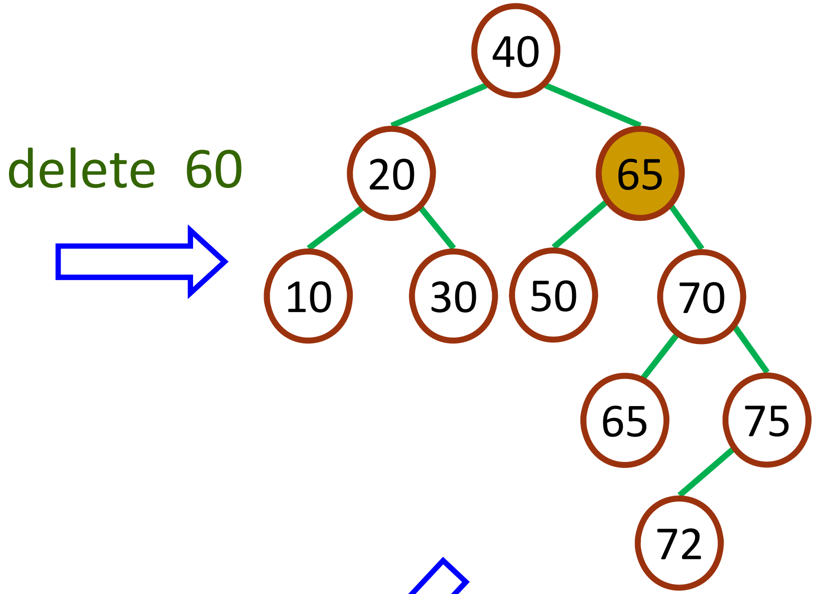
Case 2:



有一個兒子，將兒子連到父親

Case 3:

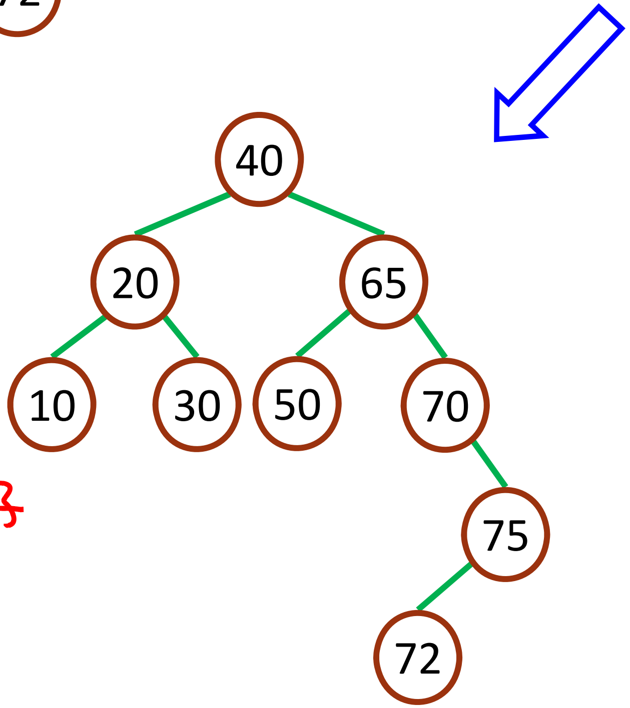




① 找下一個

② 將下一個 copy 過來

③ 將下一個刪除



可以證明下一個最多只有一個兒子

⇒沒有左子

⇒最多 delete 2 次

Deletion

```
TREE-DELETE(T, z)
    if left[z] = NIL or right[z] = NIL      ] y: 真正刪除的点
       y ← z
    else  y ← TREE-SUCCESSOR(z)
    if left[y] ≠ NIL                         ] x: y唯一有可能的兒子
       x ← left[y]
    else  x ← right[y]
    if x ≠ NIL                               ] 設定x的新父親
       p[x] ← p[y]
    if p[y] = NIL                            ] 設定x為新父親的左子或右子
       root[T] ← x
    else if y = left[p[y]]
               left[p[y]] ← x
    else right[p[y]] ← x
    if y ≠ z                                 ] y和z不同点: case 3
        key[z] ← key[y]
        copy y's satellite data into z         ⇒複製y的資料到z
    return y
```

Time: $O(h)$.