Binary Search Tree (Chapter 11: Tamassia, Goldwasser, Goodrich)

Chapter 12: Cormen, Leiserson, Rivest, Stein)
CLRS

- Node : n

$n.key$, $n.value$, $n.left$, $n.right$, $n.parent$

↓
Satellite Data

- Tree : T

$T.root$ = root of the tree

Properties

(I) • keys have a total ordering

1-dim: $1 < 2$ (total order)

d-dim:

$\begin{bmatrix} 2 \\ 1 \end{bmatrix}$ ≶ $\begin{bmatrix} 1 \\ 2 \end{bmatrix}$ (only partial order)

$\begin{bmatrix} 1 \\ 0 \end{bmatrix} < \begin{bmatrix} 2 \\ 1 \end{bmatrix}$

(II) • Pick nodes n_1 and n_2

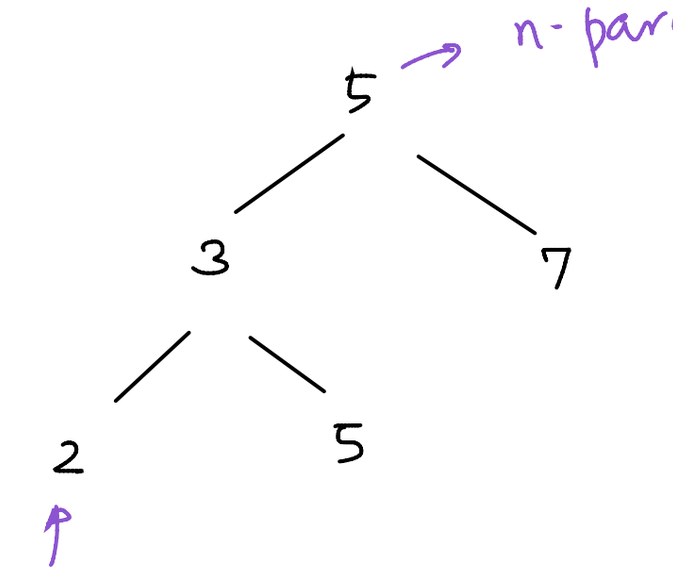
- if n_1 is in the left subtree of n_2

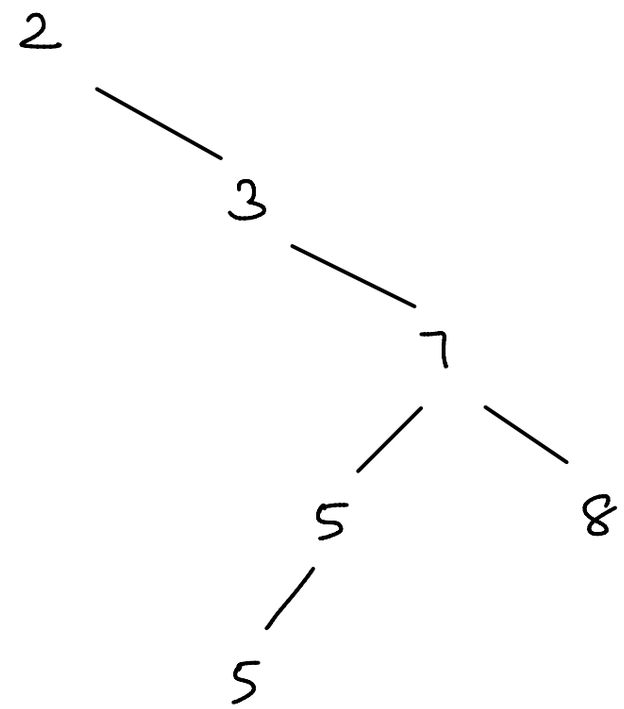
$$\Rightarrow n_1.\text{key} \leq n_2.\text{key}$$

- if n_1 is in the right subtree of n_2

$$\Rightarrow n_2.\text{key} \leq n_1.\text{key}$$

Examples of BST

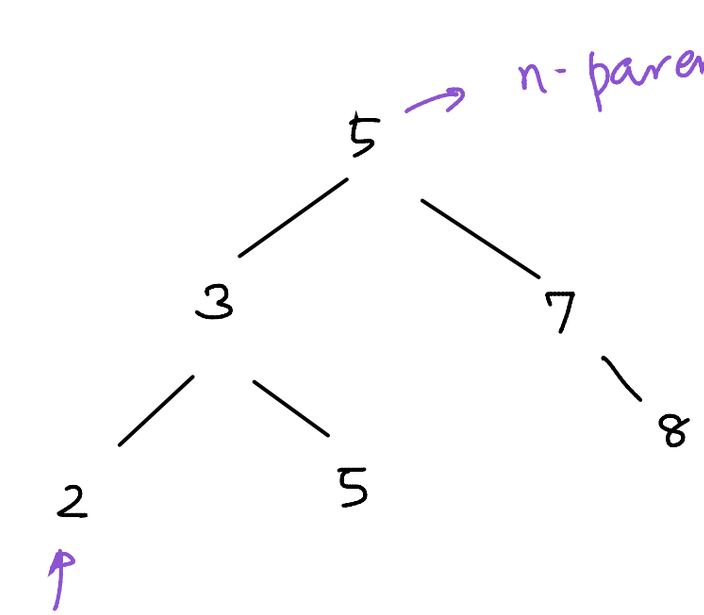




- (I) & (II) allows duplicate keys.
- Can disallow duplicate keys.

Inorder traversal of BST gives a sorted order (ascending) of keys

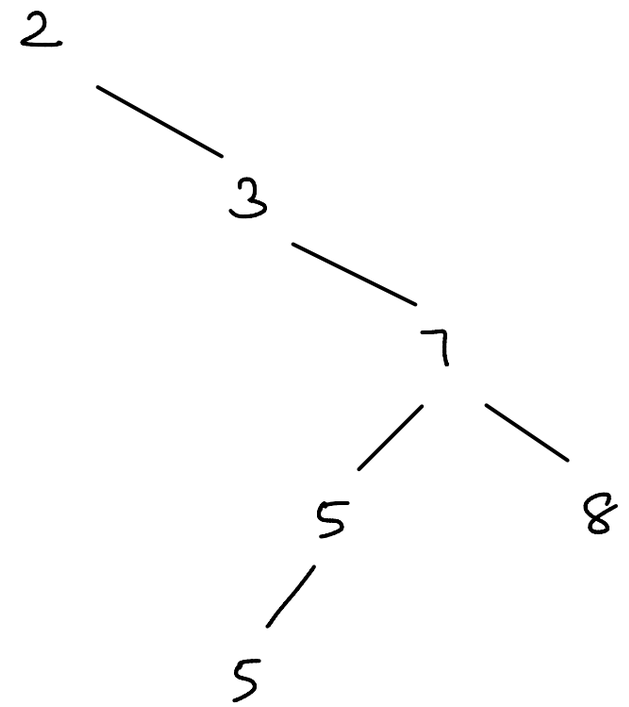
```
inorder-tree-walk (n)
    if n ≠ None
        inorder-tree-walk (n.left)
        print (n.key)
        inorder-tree-walk (n.right)
```



n.left = None

n.right = None

2 3 5 5 7 8



2 3 5 5 7 8

Insertion

n is a new node

Tree- Insert (T, n)

parent node ← None

current node ← T. root

← assignment

while current node ≠ None

while continues until we hit a leaf node

temp variable → parent node ← current node

if n.key < current node.key

then current node ← current node. left

else current node ← current node. right

n. parent ← parent node

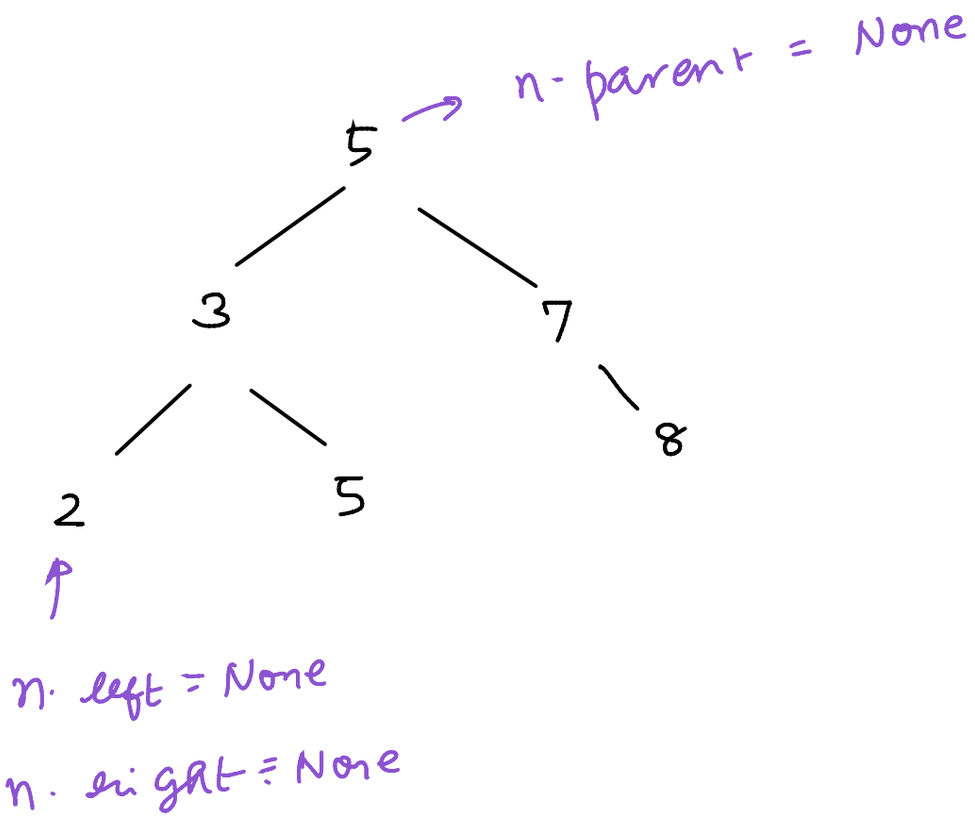
inserting very first element → if parent node = None

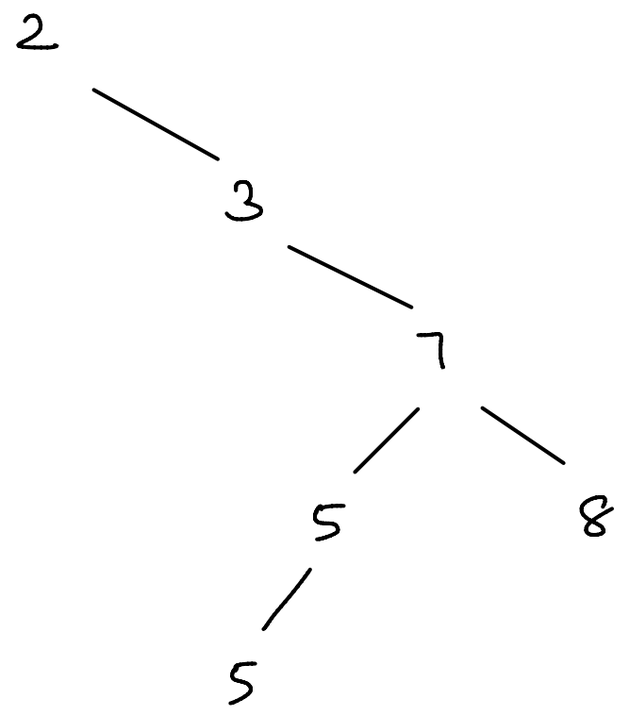
then T. root ← node

else if n.key < parent. key

then parent. left ← n

else parent. right ← n





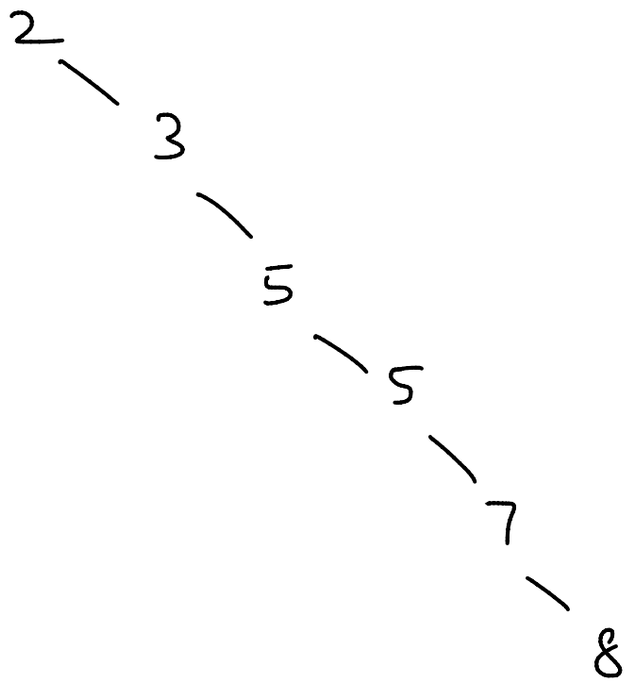
Inorder walk 2 3 5 5 7 8

2 3 5 5 7 8

order of insertion : 5 3 7 5 8 2

2 3 7 5 8 5

Say order of insertion was 2 3 5 5 7 8 itself



Searching: For a key 'k' in a tree 'T' rooted at node 'n'

```
Tree-Search(n, k)
    if  n = None  or  k = n.key
        return  n
    if  k < n.key
        then return( Tree-Search(n.left, k) )
        else return( Tree-search(n.right, k) )
```

} example of recursive call

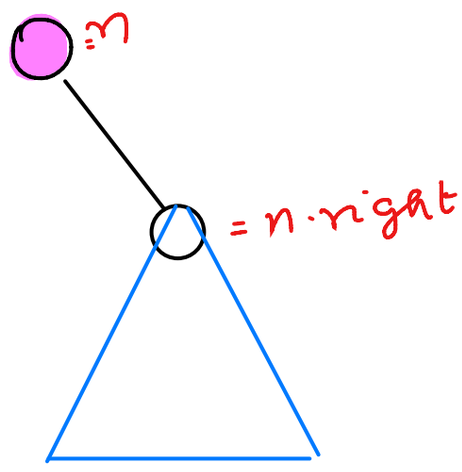
Exercise: write a non-recursive / iterative pseudocode for Tree-Search

Node with Minimum key

```
Tree-Minimum(n)
    while n.left ≠ None
        n ← n.left
    return n
```

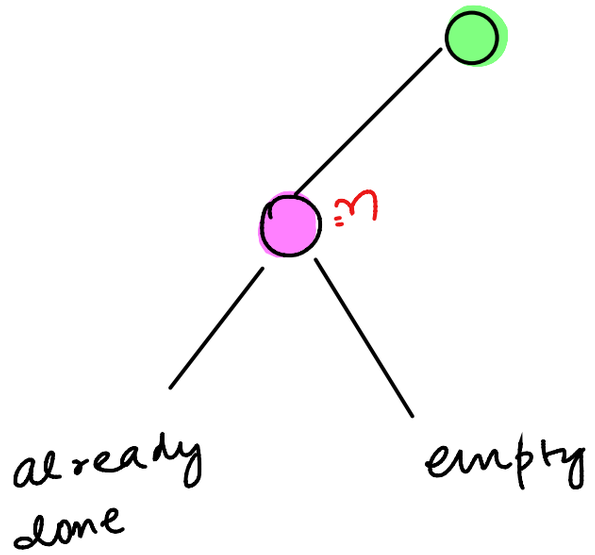
Successor of node 'n' : Assume that all keys are distinct, successor(n) is the node with the smallest key greater than n.key

Case 1 :



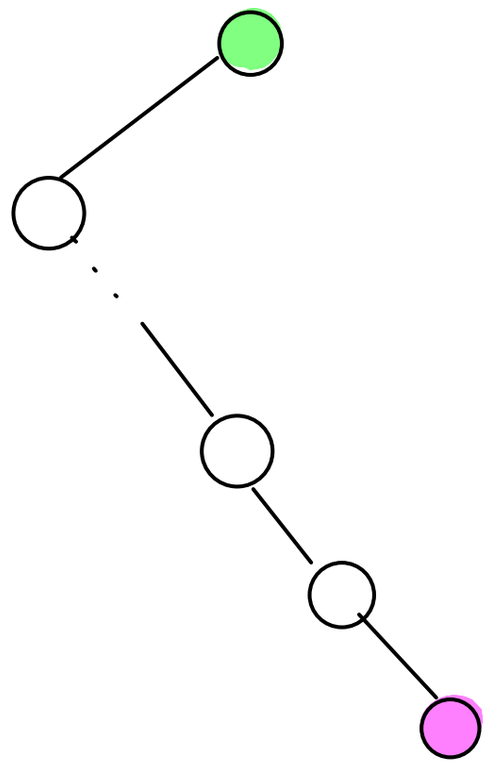
Tree-Minimum(n.right)

Case 2: Right subtree is empty



lucky case 2: if node is left child of its parent then parent itself is successor

- exhausted left, node, right is empty
- if at all anything, it has to be with parent
- lucky case 2 does not happen ⇒



Find that ancestor to whose left subtree node n belongs to