Introduction to Algorithms

Lecture 6

Recap

- Direct-accessible table
- Hash tables
- Hash functions
  - Universal hashing
- Perfect Hashing
- Open addressing

Today's topics

- Binary-Search Trees
- Operations
  - Walk
  - Search, Minimum, Maximum
  - Insert, Delete

Binary Search Trees

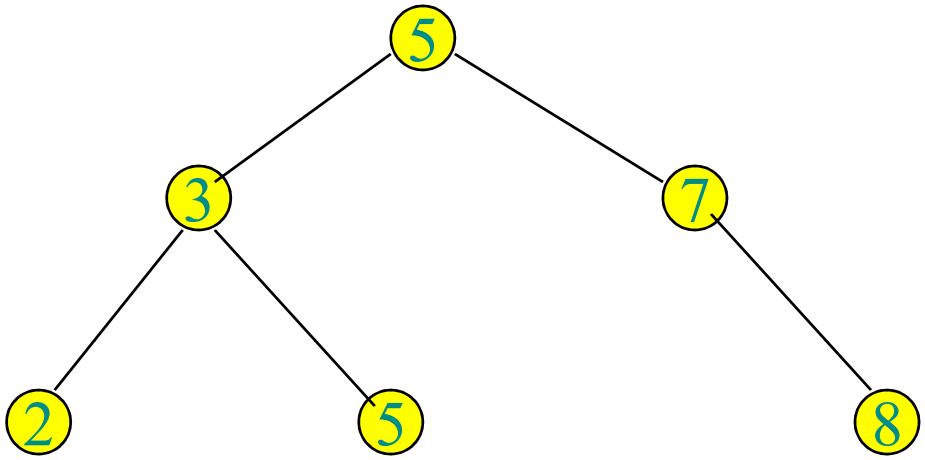
- Implement Dictionary + Find-Min.
- Assume that the input has a key that is an integer, or a string, or another data type on which a comparison < is well-defined. For simplicity we identify an item *a* and its key *a.key*.
- Items are placed on the nodes of a binary tree.

Basic Idea

The following property is maintained:

- If *a* sits at a node, all the items on the left subtree have a key smaller than *a*.
- If *a* sits at a node, all the items on the right subtree have a key bigger than *a*.

Binary Search Trees



Implementation in memory

- A node is a structure with three points to father, left child, right child.
- The root has a NIL pointer for father.
- Leaves has NIL pointers for children. A node may have only one child; then the other is a NIL pointer.

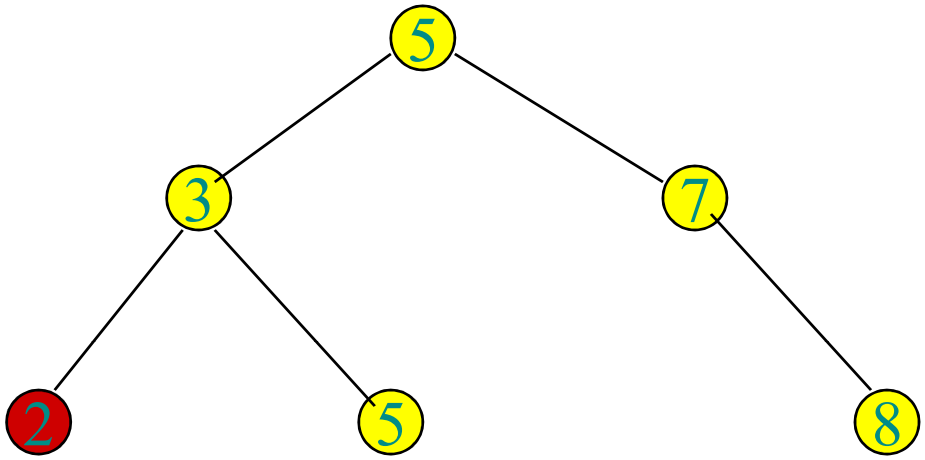
Walk

We can output the set of keys in sorted order in linear time.

- inorder tree walk
- preorder tree walk
- postorder tree walk

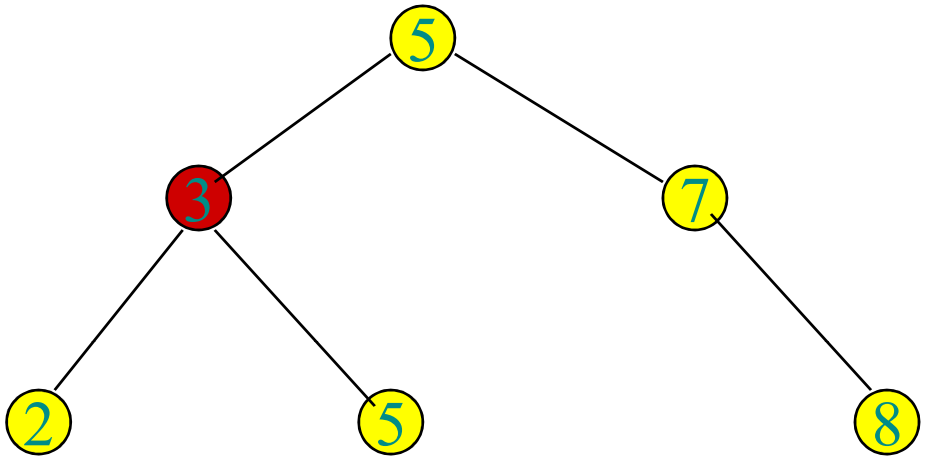
```
Inorder-Tree-Walk(x)                                  T(n) =
    if x ≠ NIL
    then Inorder-Tree-Walk(left[x])                   T(k) +
         print key[x]
         Inorder-Tree-Walk(right[x])                  T(n – k) + d
```

Example of Inorder



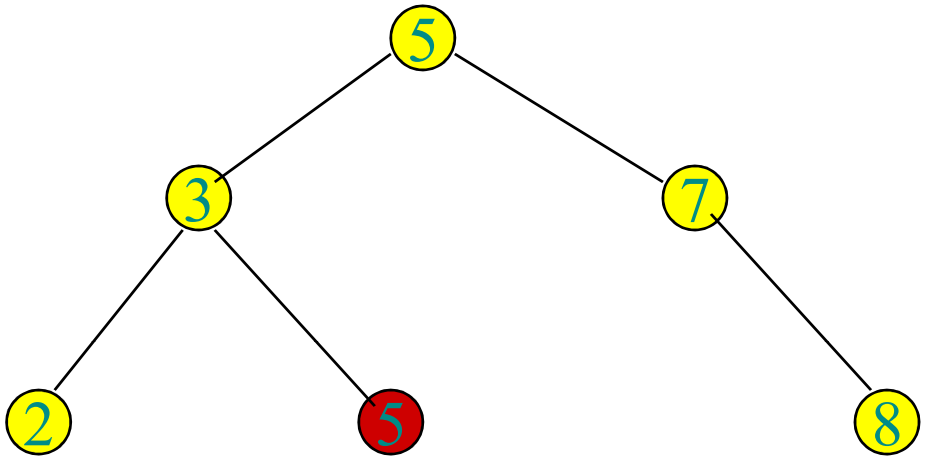
2

Example of Inorder



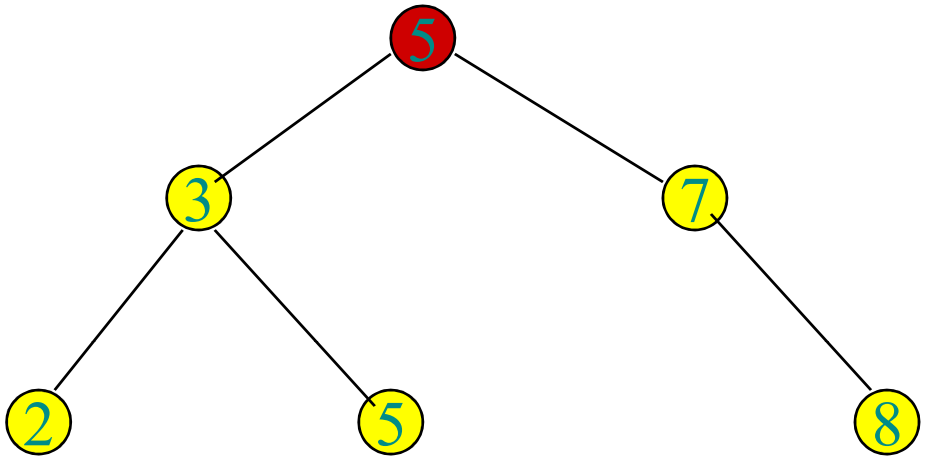
2 3

Example of Inorder



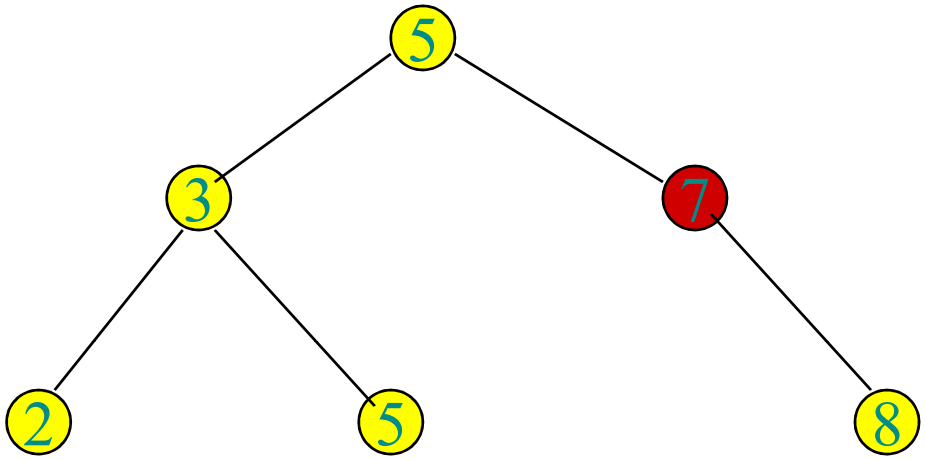
2 3 5

Example of Inorder



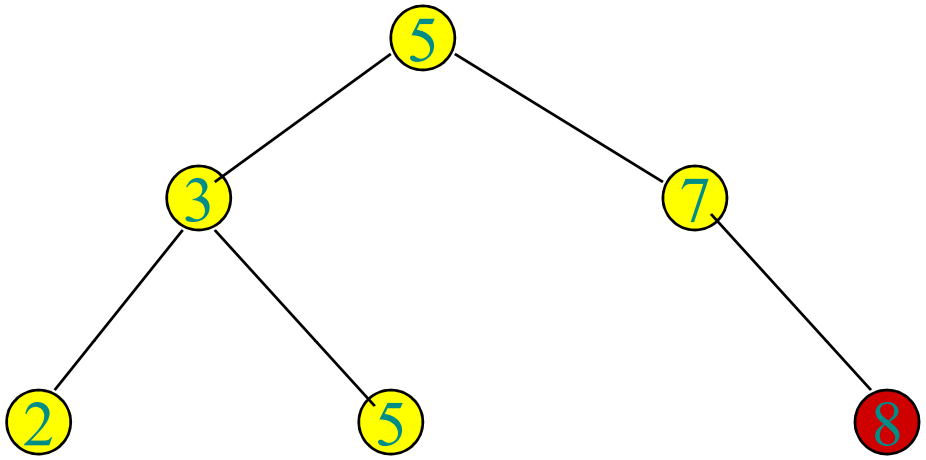
2 3 5 5

Example of Inorder



2 3 5 5 7

Example of Inorder



2 3 5 5 7 8

Implementing Operations

Search(a).

Start a pointer p at the root. If the key is equal to a, done.

If the key is smaller than a proceed on the right child;

otherwise proceed on the left child.

Fail if the child you want to reach does not exist.

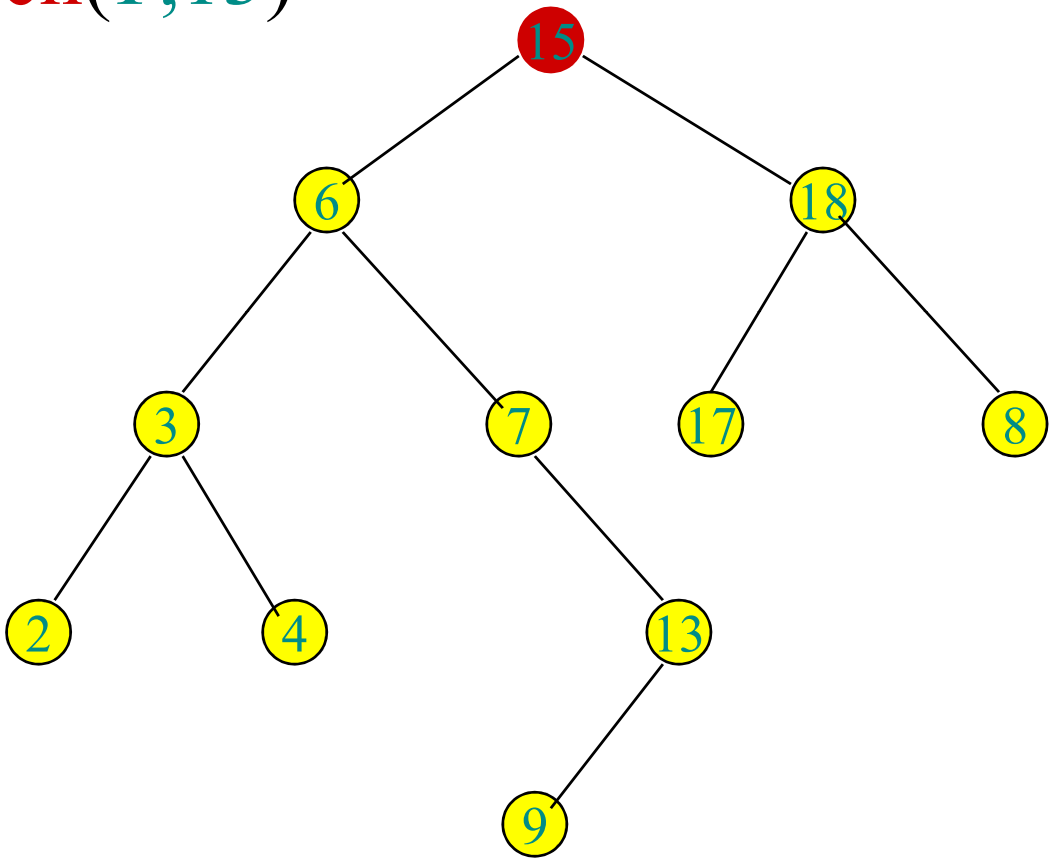
Search

```
Search(T, k)
    x ← root(T)
    while x ≠ NIL and key[x] ≠ k
        do if k < key[x]
                then x ← left[x]
                else x ← right[x]
    return x
```

- running time: $O(h)$.

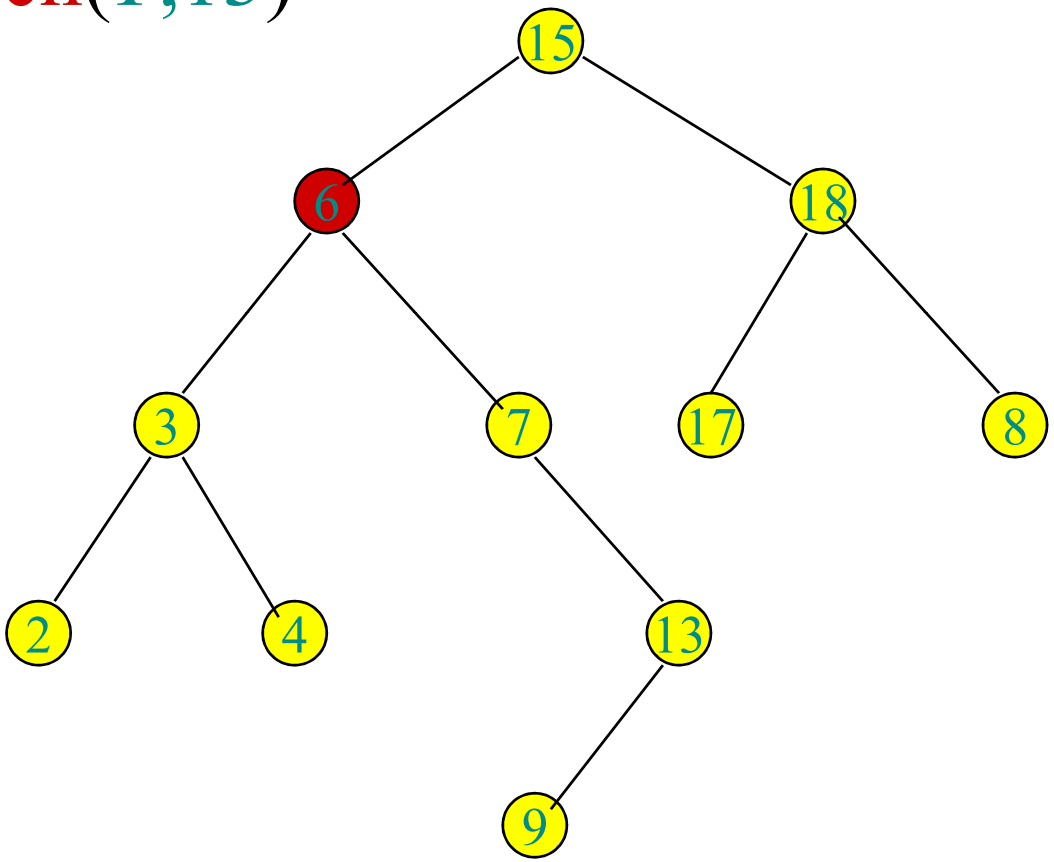
Example of Search

Search(T,13)



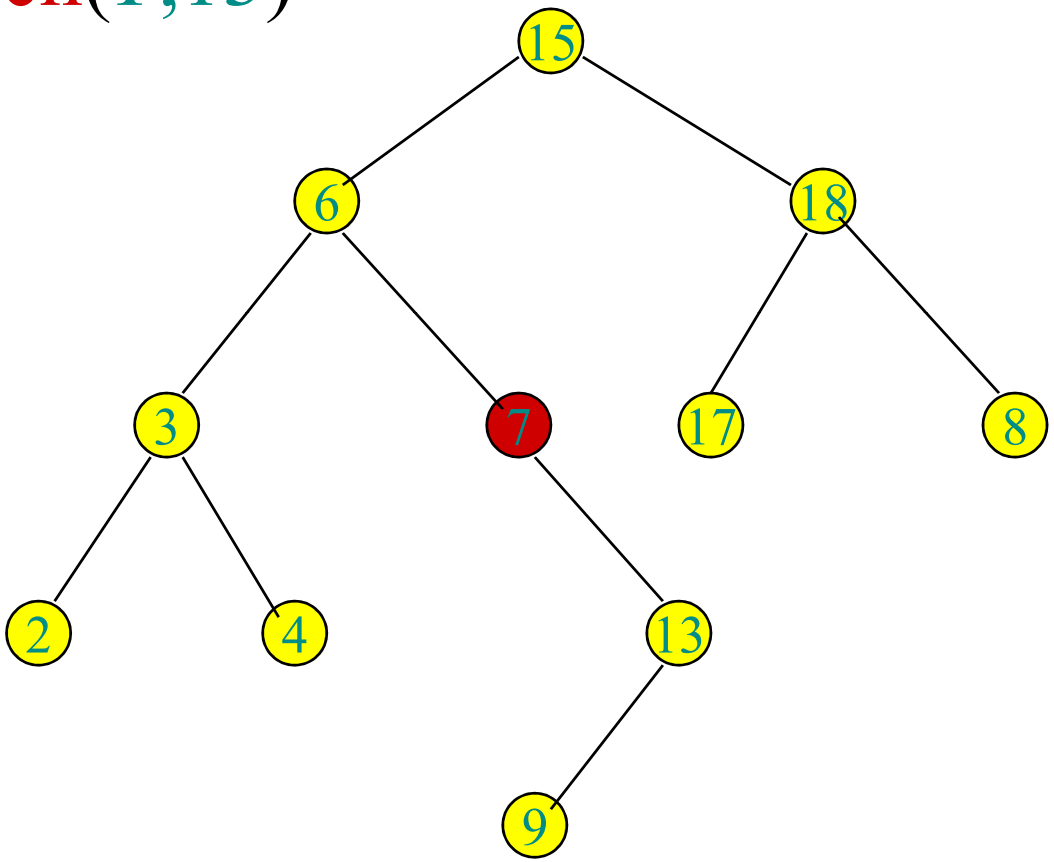
Example of Search

Search(T,13)



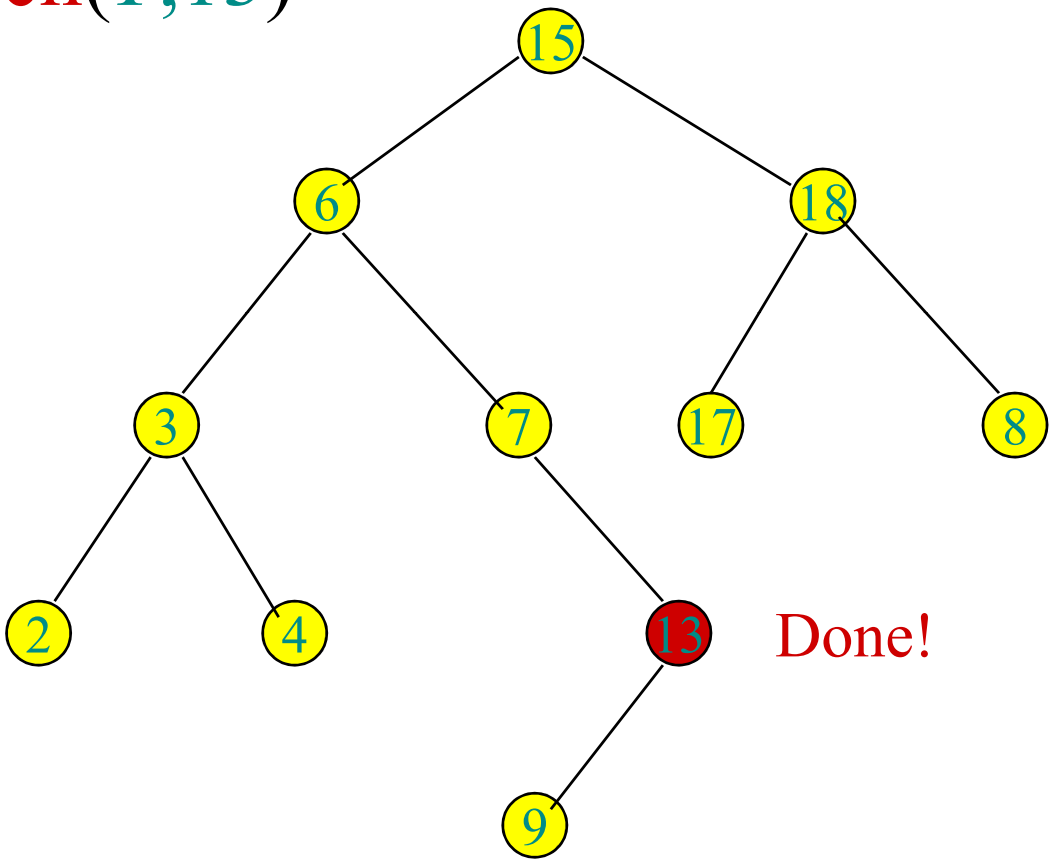
Example of Search

Search(T,13)



Example of Search

Search(T,13)



Implementing Operations

Find-min.

Start at the root, and go to the left as much as possible.

Insert(a).

Do as in Search(a); create the child where you want to go but does no exist.

Delete(a). ??

Minimum

Tree-Minimum(x)

while $left[x] \neq$ NIL

do $x \leftarrow left[x]$

return x

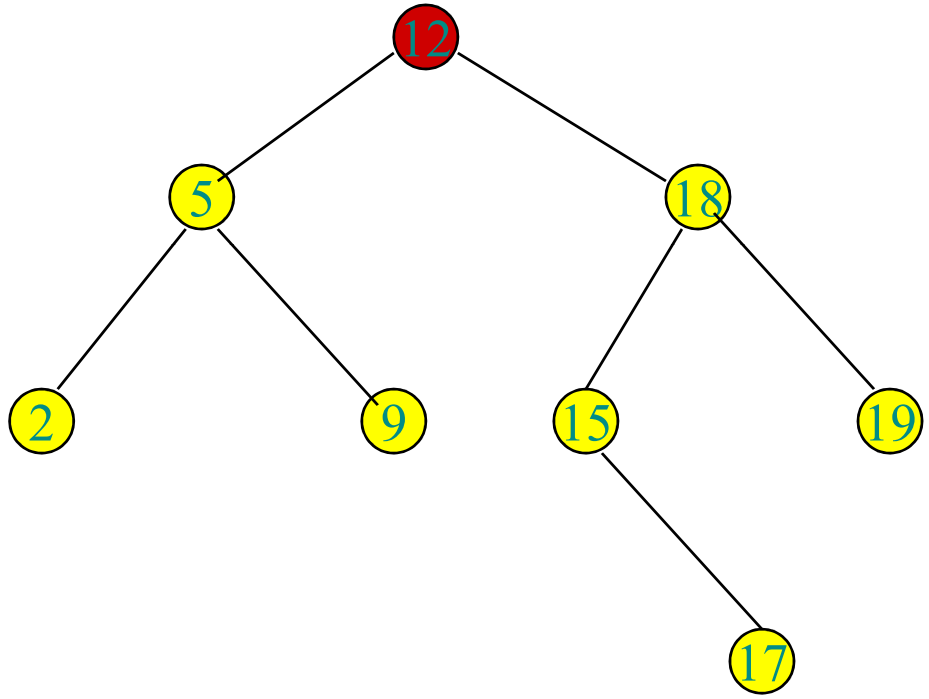
Dynamic BST

Insert(T, k)

- run search until reach NIL (presume $left[x] = \text{NIL}$)
- $left[x] \leftarrow$ new node with $key = k$
  $left = \text{NIL}$
  $right = \text{NIL}$
  $parent = x$

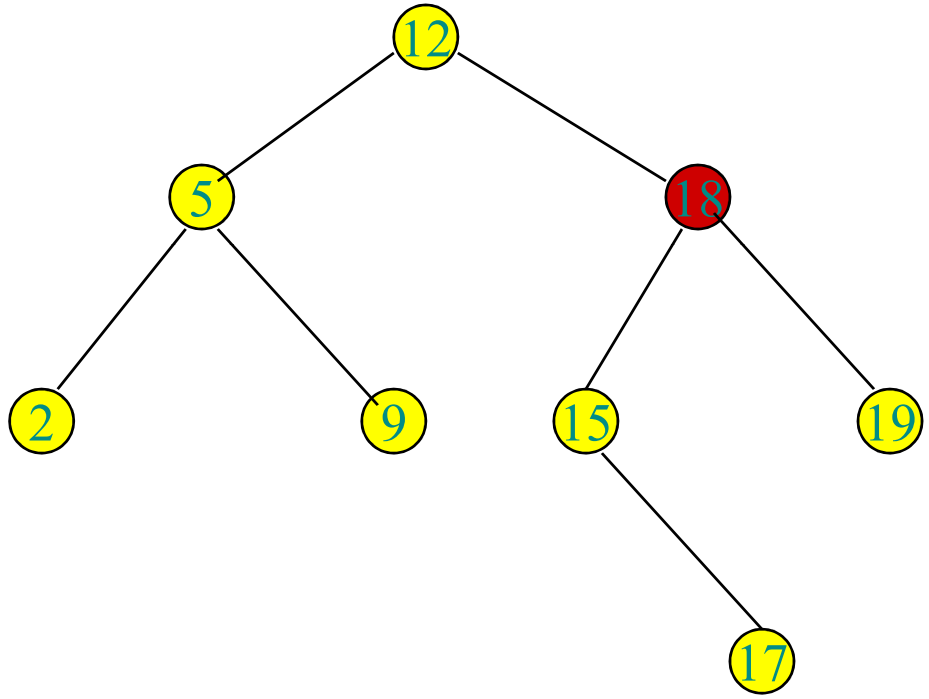
Example of Insert

Insert(T, 13)



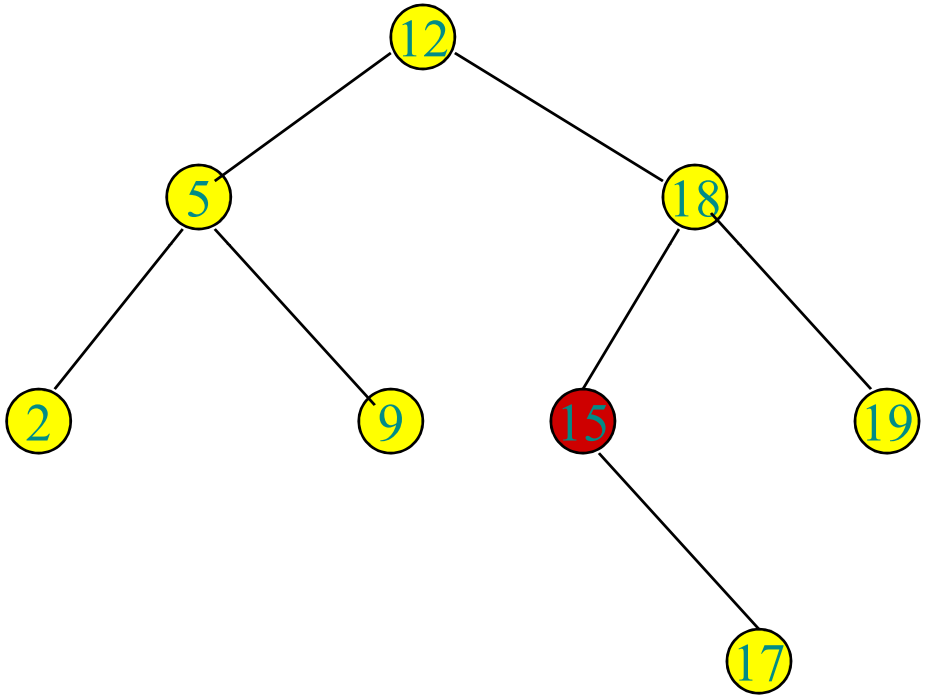
Example of Insert

Insert(T, 13)



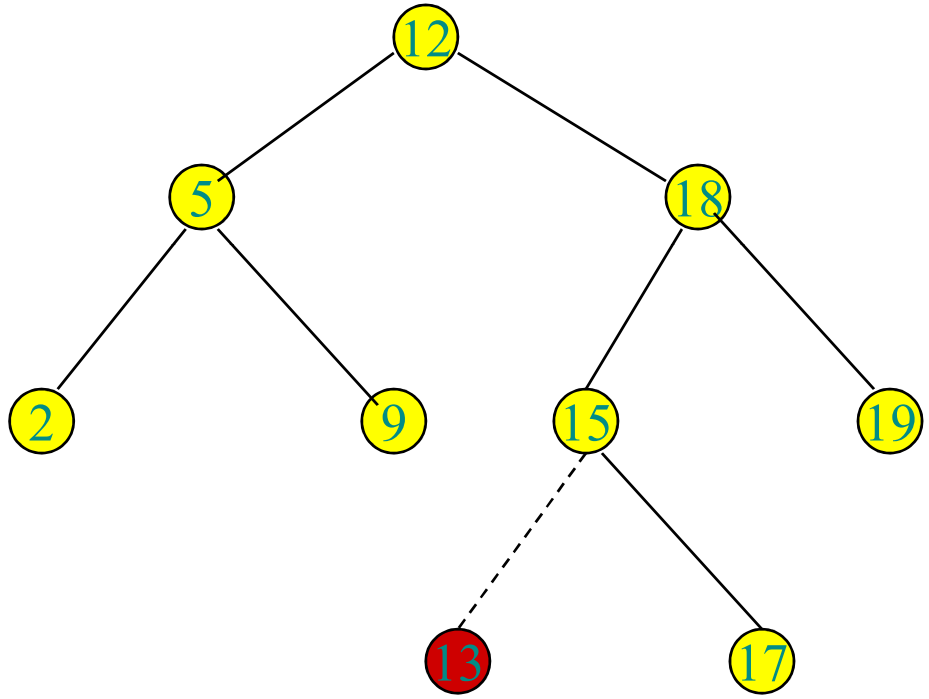
Example of Insert

Insert(T, 13)



Example of Insert

Insert(T, 13)



BST Sort

```
BST-Sort(A)
    for i ← 1 to n
        do Insert(A[i])
    Inorder(root)
```

Analysis: same comparisons as quicksort, but in different order.

Other Operation

Given a pointer to a node p with key a, we can find the successor of a.

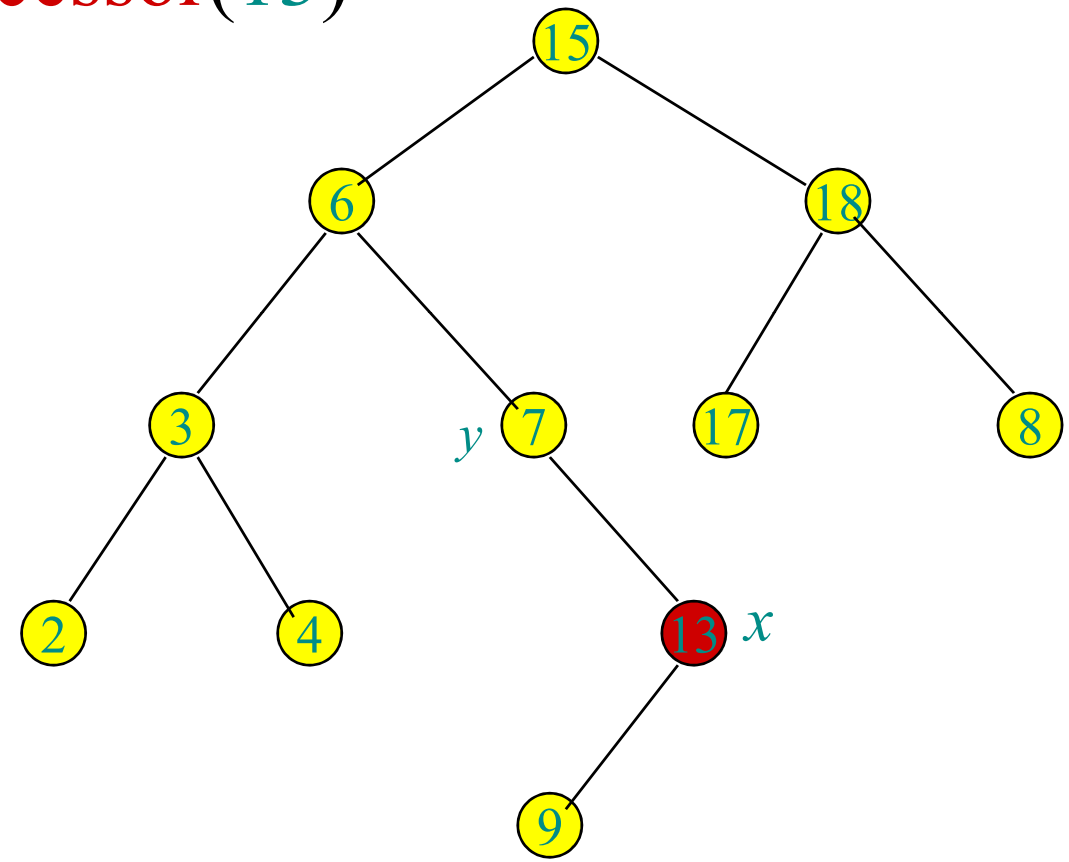
– next higher element in inorder sequence.

Successor

```
Tree-Successor(x)
    if right[x] ≠ NIL
       then return Tree-Minimum(right[x])
    y ← p[x]
    while y ≠ NIL and x = right[y]
         do x ← y
              y ←  p[y]
    return y
```

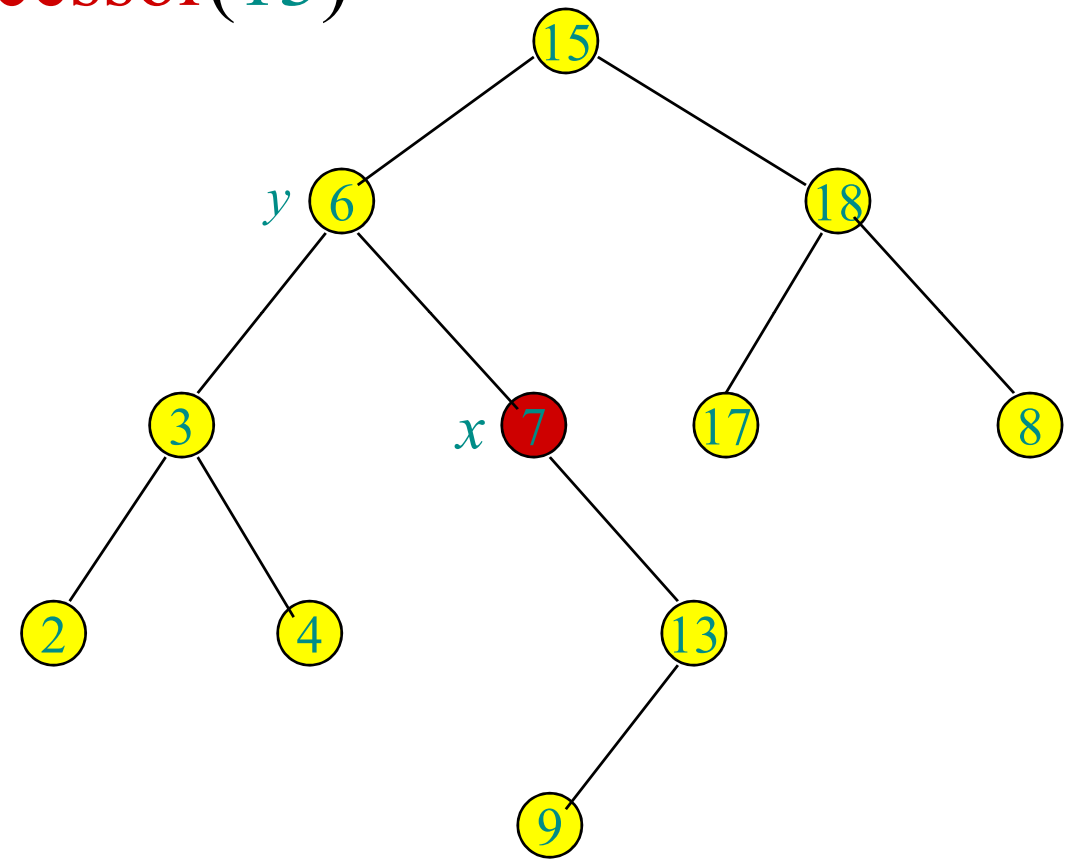
Example of Successor

Successor(13)



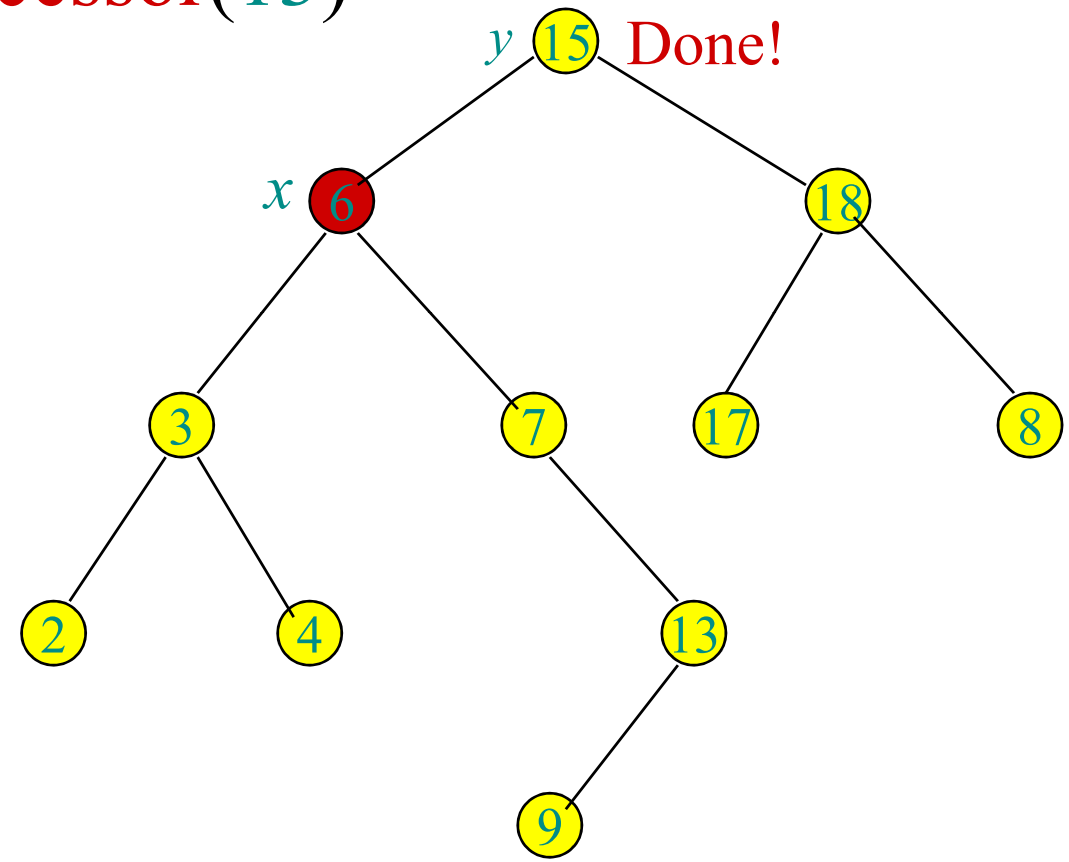
Example of Successor

Successor(13)



Example of Successor

Successor(13)



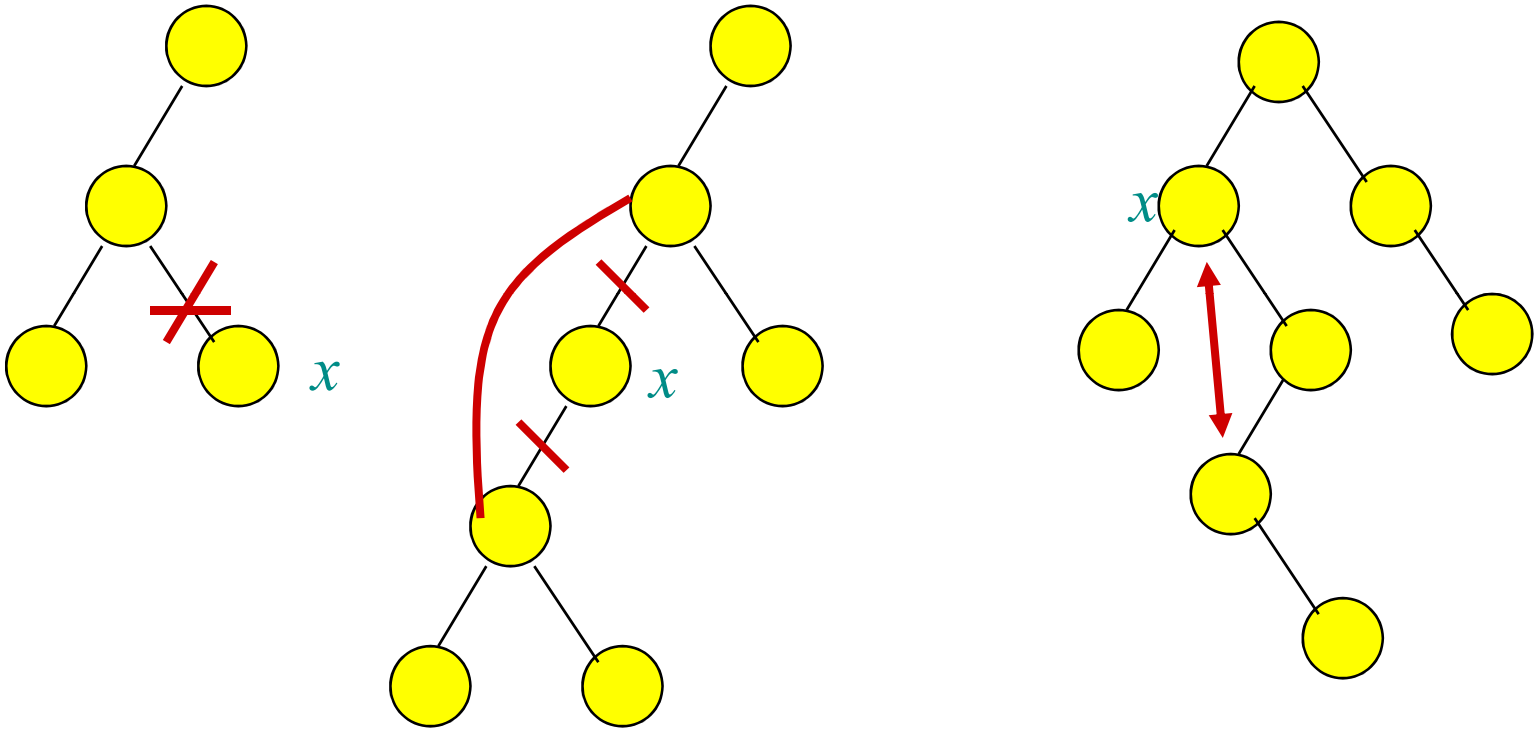
Delete

Delete node pointed by p, storing key a.

- If there is one or no child: remove the node, attach the child, if any, to the father of p in place of p.

- If there are two children,
  1. find the successor of a in the tree, let it be in the node pointed by q with value b.
  2. Delete q, as in previous case; copy b in place of a on node pointed by p.

Example of Delete



Running Time

- The height of the tree bounds in the worst case the running time of each operation.
- The height can be n. The height is always at least $\Omega(\lg n)$.
- For n random inputs the height is $O(\lg n)$.
- Analysis for random inputs becomes messy when deletions are allowed.
- Assumption of random inputs in unrealistic.

Randomized methods

- Modify the tree randomly (but with care) at each insert/delete, so that the height will be logarithmic on the average with respect to the random choices but in worst case with respect to the inputs.

- Difficult to do. Not much more efficient than worst-case efficient methods.

Excerise

- 12.1-2, 12.1-4, 12.2-8, 12.3-2, 12.4-2