

Binary Search Trees: Basic Operations

Daniel Kane

Department of Computer Science and Engineering
University of California, San Diego

Data Structures Fundamentals
Algorithms and Data Structures

Learning Objectives

- Implement basic operations on Binary Search Trees.
- Understand some of the difficulties with making updates.

Outline

- 1 Find
- 2 Next Element
- 3 Search
- 4 Insert
- 5 Delete

Find

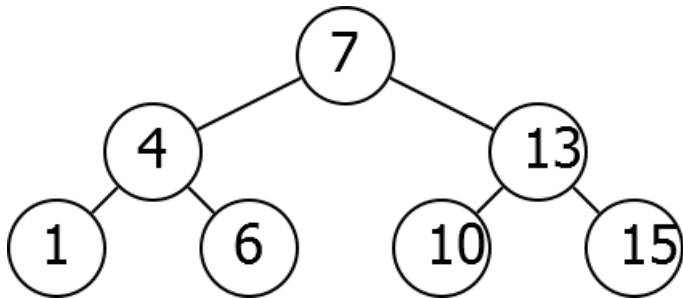
Find

Input: Key k , Root R

Output: The node in the tree of R with key k

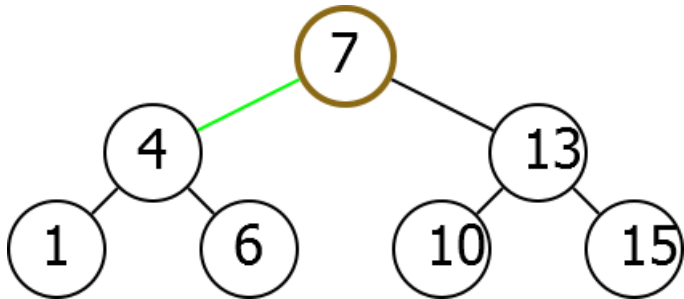
Idea

Find(6)



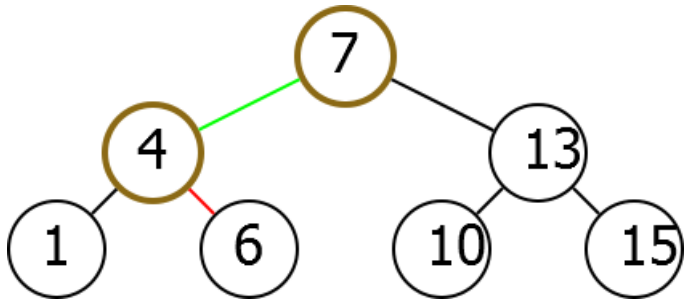
Idea

Find(6)



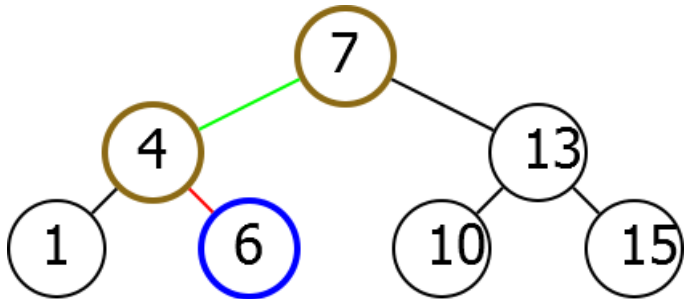
Idea

Find(6)



Idea

Find(6)



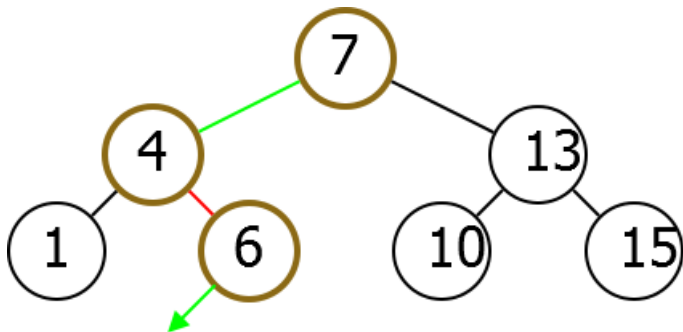
Algorithm

Find(k, R)

```
if  $R.Key = k$ :  
    return  $R$   
else if  $R.Key > k$ :  
    return Find( $k, R.Left$ )  
else if  $R.Key < k$ :  
    return Find( $k, R.Right$ )
```

Missing Key

Run Find(5).



Key not in tree. Did find point where it should be.

Missing Key

If you stop before reaching a null pointer, you find the place in the tree where k would fit.

Modification

Find (modified)

```
else if  $R.\text{Key} > k$ :  
    if  $R.\text{Left} \neq \text{null}$ :  
        return Find( $k, R.\text{Left}$ )  
    return  $R$ 
```

Outline

- 1 Find
- 2 **Next Element**
- 3 Search
- 4 Insert
- 5 Delete

Adjacent Elements

Given a node N in a Binary Search Tree, would like to find adjacent elements.

Next

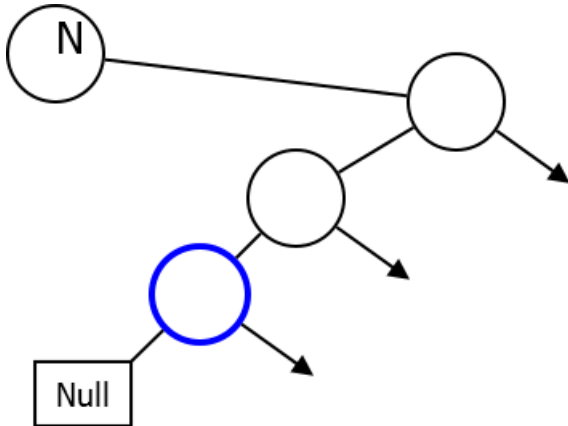
Next

Input: Node N

Output: The node in the tree with the next largest key.

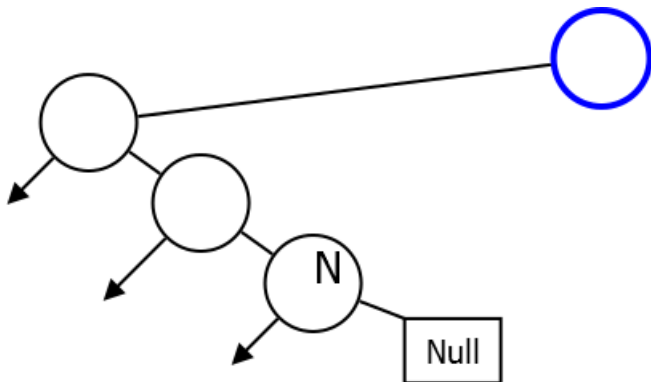
Case I

If you have right child.



Case II

No right child.



Next

Next(*N*)

```
if N.Right  $\neq$  null:  
    return LeftDescendant(N.Right)  
else:  
    return RightAncestor(N)
```

Left Descendant

LeftDescendant(N)

```
if  $N.\text{Left} = \text{null}$   
    return  $N$   
else:  
    return LeftDescendant( $N.\text{Left}$ )
```

Right Ancestor

RightAncestor(N)

```
if  $N.Key < N.Parent.Key$   
    return  $N.Parent$   
else:  
    return RightAncestor( $N.Parent$ )
```

Outline

- 1 Find
- 2 Next Element
- 3 Search**
- 4 Insert
- 5 Delete

Range Search

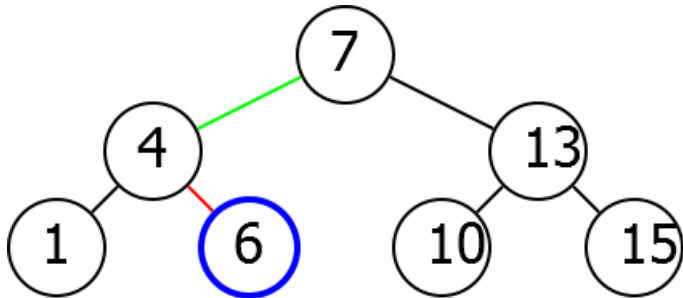
Range Search

Input: Numbers x , y , root R

Output: A list of nodes with key between x and y

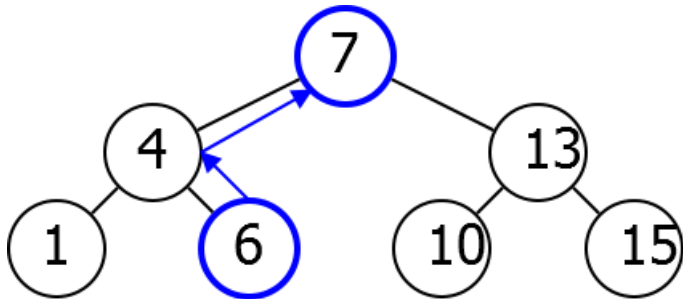
Idea

RangeSearch(5, 12).



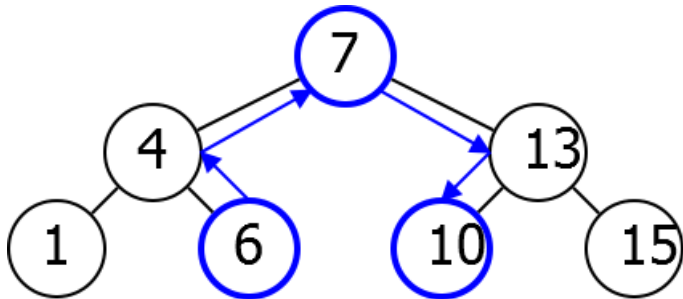
Idea

RangeSearch(5, 12).



Idea

RangeSearch(5, 12).



Implementation

RangeSearch(x, y, R)

$L \leftarrow \emptyset$

$N \leftarrow \text{Find}(x, R)$

while $N.\text{Key} \leq y$

 if $N.\text{Key} \geq x$:

$L \leftarrow L.\text{Append}(N)$

$N \leftarrow \text{Next}(N)$

return L

Outline

- 1 Find
- 2 Next Element
- 3 Search
- 4 Insert**
- 5 Delete

Insert

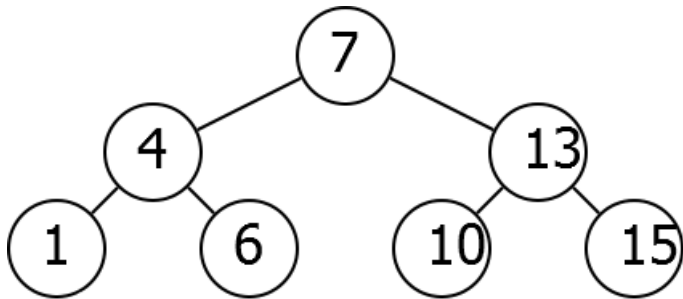
Insert

Input: Key k and root R

Output: Adds node with key k to the tree

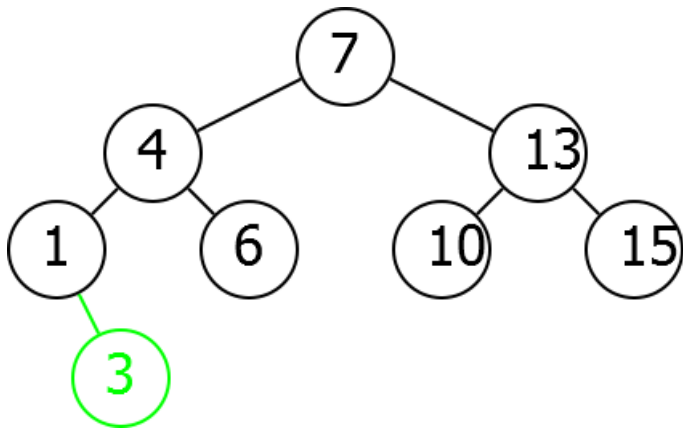
Insert Idea

Insert(3)



Insert Idea

Insert(3)



Implementation

$\text{Insert}(k, R)$

$P \leftarrow \text{Find}(k, R)$

Add new node with key k as child
of P

Outline

- 1 Find
- 2 Next Element
- 3 Search
- 4 Insert
- 5 Delete

Delete

Delete

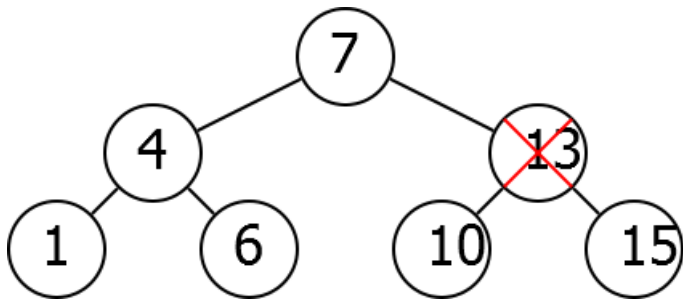
Input: Node N

Output: Removes node N from the tree

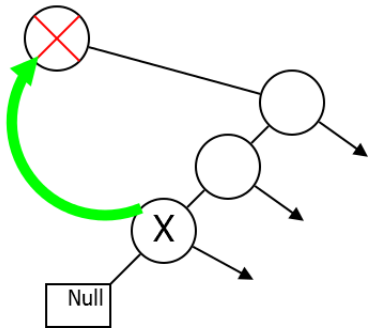
Difficulty

Cannot simply remove.

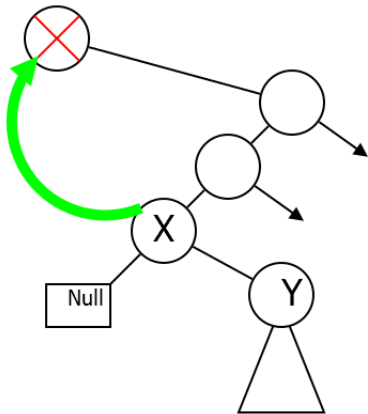
Delete(13)



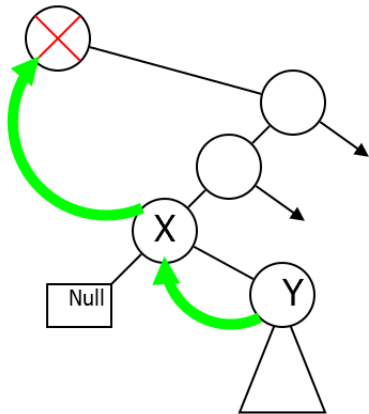
Idea



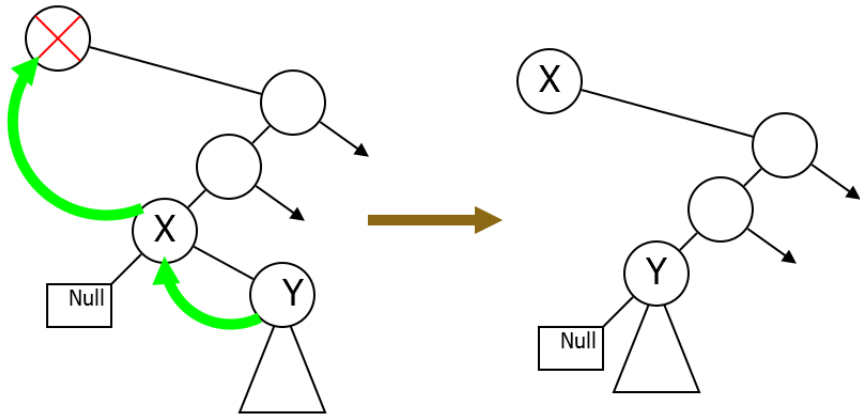
Idea



Idea



Idea



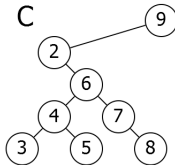
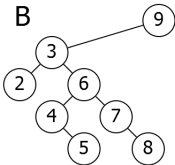
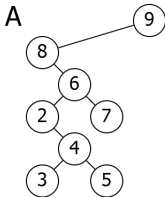
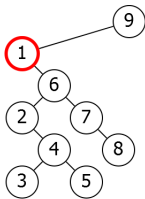
Implementation

Delete(N)

```
if  $N.Right = \text{null}$ :  
    Remove  $N$ , promote  $N.Left$   
else:  
     $X \leftarrow \text{Next}(N)$   
     $X.Left = \text{null}$   
    Replace  $N$  by  $X$ , promote  $X.Right$ 
```

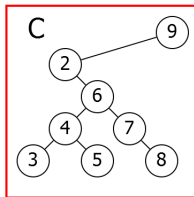
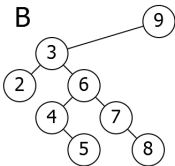
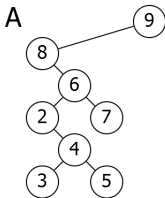
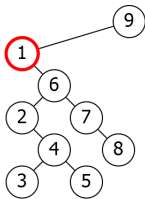
Problem

Which of the following trees is obtained when the selected node is deleted?



Problem

Which of the following trees is obtained when the selected node is deleted?



Next Time

Runtime and balance.