Data Structures

Binary Search Trees

Andres Mendez-Vazquez

November 12, 2016

Outline

1. Introduction
   - Basic Concepts
2. BST Representation
3. Operations
   - Get
   - Put
   - Minimum and Maximum
   - Remove
   - Tree Delete
   - Examples of Deletion

Why Linked Representation of Binary Trees?

Complexity Of Search and Insert: They are used many operations

Data Structure	Worst		Expected	
	Search	Insert	Search	Insert
Sorted List (Array)	$O(\log n)$	$O(n)$	$O(\log n)$	$O(n)$
Sorted List (Chain)	$O(n)$	$O(n)$	$O(n)$	$O(n)$

Challenge

Efficient implementations of get() and put() and ordered iteration.

Why Linked Representation of Binary Trees?

Complexity Of Search and Insert: They are used many operations

Data Structure	Worst		Expected	
	Search	Insert	Search	Insert
Sorted List (Array)	$O(\log n)$	$O(n)$	$O(\log n)$	$O(n)$
Sorted List (Chain)	$O(n)$	$O(n)$	$O(n)$	$O(n)$

Challenge

Efficient implementations of get() and put() and ordered iteration.

Outline

1 Introduction
- Basic Concepts

2 BST Representation

3 Operations
- Get
- Put
- Minimum and Maximum
- Remove
- Tree Delete
- Examples of Deletion

Basic Concepts

Def

A BINARY SEARCH TREE is a binary tree in symmetric order.

Basically

A binary tree is either:

- Empty
- A key-value pair and two binary trees.

Basic Concepts

Def

A BINARY SEARCH TREE is a binary tree in symmetric order.

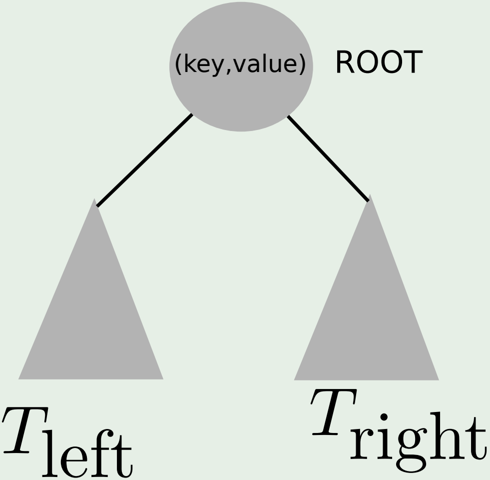
Basically

A binary tree is either:

- Empty
- A key-value pair and two binary trees.

Example

Thus



Symmetric Order

Meaning

- Every node has a key
- Every node's key
  - It is larger than all keys in its left subtree
  - It is smaller than all keys in its right subtree

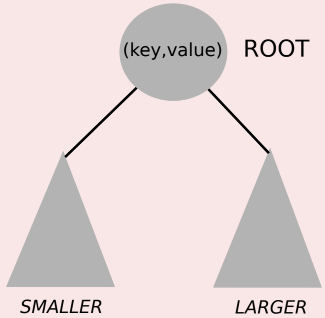
Thus

Symmetric Order

Meaning

- Every node has a key
- Every node's key
  - It is larger than all keys in its left subtree
  - It is smaller than all keys in its right subtree

Thus



BST Representation

A BST is a reference to a Node

A Node is comprised of four fields:

- A key and a value.
- A reference to the left and right subtree.

Code

Properties

- Key and Value are generic types;
- Key is Comparable

BST Representation

A BST is a reference to a Node

A Node is comprised of four fields:

- A key and a value.
- A reference to the left and right subtree.

Code

```
private class Node{
  Key key;
  Value val;
  Node left, right;
}
```

Properties

- Key and Value are generic types;
- Key is Comparable

BST Representation

A BST is a reference to a Node

A Node is comprised of four fields:

- A key and a value.
- A reference to the left and right subtree.

Code

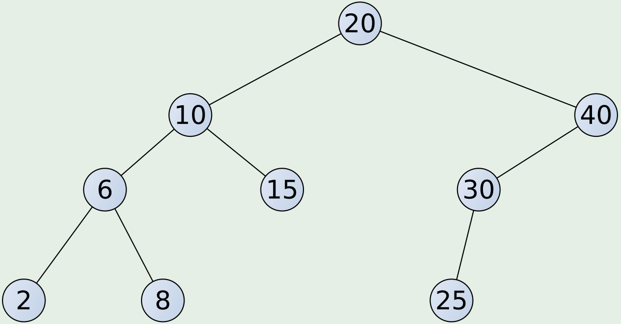
```
private class Node{
  Key key;
  Value val;
  Node left, right;
}
```

Properties

- Key and Value are generic types;
- Key is Comparable

Example

Only keys are shown



Code For the Class

We have this

```
public class BST<Key extends Comparable<Key>, Value> {
    private BinaryTreeNode root;
    private class BinaryTreeNode
    {
        Key key;
        Value val;
        BinaryTreeNode left, right;
        BinaryTreeNode(Key key, Value val)
        {
         this.key = key;
         this.val = val;
        }
    }
    public void put(Key key, Value val)...
    public Val get(Key key)...
}
```

Outline

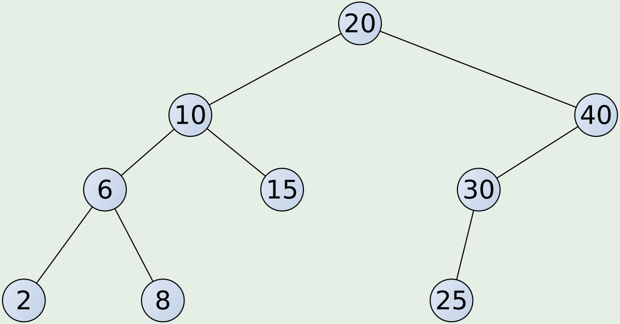
1 Introduction
- Basic Concepts

2 BST Representation

3 Operations
- Get
- Put
- Minimum and Maximum
- Remove
- Tree Delete
- Examples of Deletion

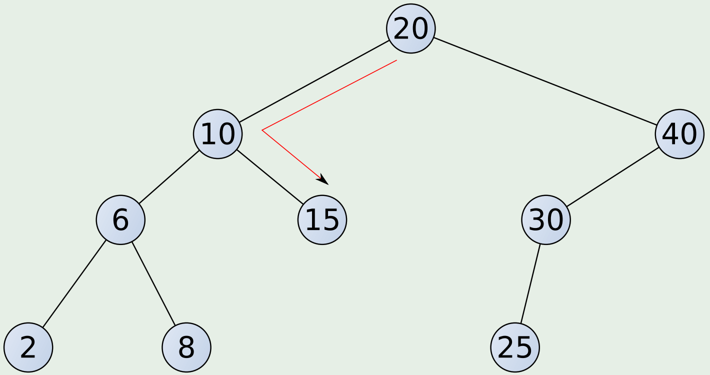
Operations: Get

We have the following



Operations: Get

Binary Search



Operations: Get

We have the following

```java
public Value get(Key key)
{
  BinaryTreeNode x = root;
  while (x != null)
  {
     int cmp = key.compareTo(x.key);
     if (cmp == 0)
          return x.val;
     else if (cmp < 0) x = x.left;
     else if (cmp > 0) x = x.right;
  }
  return null;
}
```

Complexity

We have the following

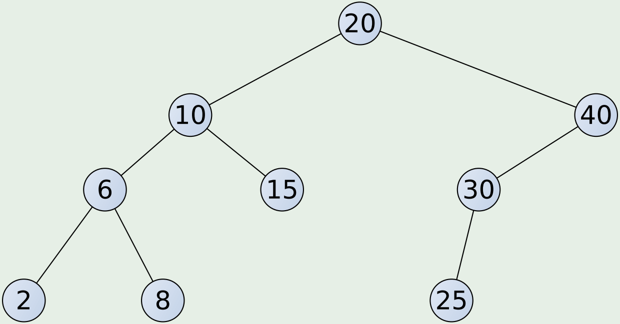
Complexity is $O(h) = O(n)$, where n is number of nodes/elements.

Outline

1. Introduction
   - Basic Concepts
2. BST Representation
3. Operations
   - Get
   - Put
   - Minimum and Maximum
   - Remove
   - Tree Delete
   - Examples of Deletion

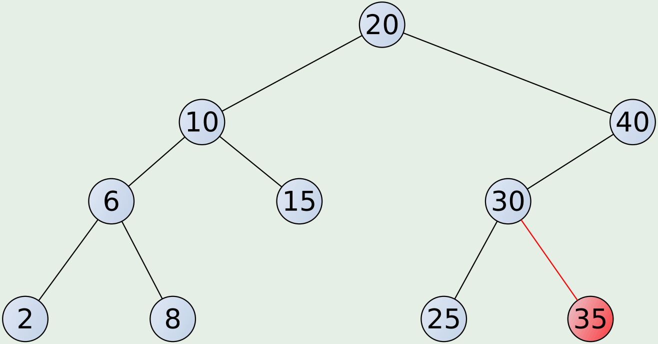
What about the operation put?

Put a pair whose key is 35



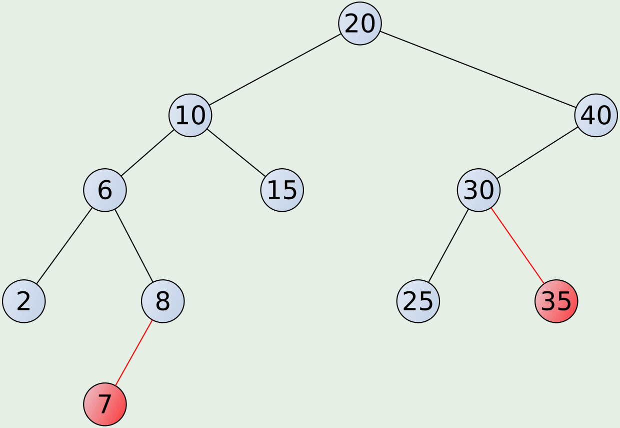
What about the operation put?

Put a pair whose key is 7



What about the operation put?

Thus...



Operations: Put

Code

```
public void put(Key key, Value val)
{
  BinaryTreeNode x = this.root;
  BinaryTreeNode temp;
  int cmp;
  while (x != null)
  {
     temp = x;
     cmp = key.compareTo(x.key);
     if (cmp == 0)
          break;
     else if (cmp < 0) x = x.left;
     else if (cmp > 0) x = x.right;
  }
  if (x == null)
     this.root = new BinaryTreeNode(Key key, Value val);
  else
      if (cmp==0)
        x.val = val;
      else if (temp.key<key)
             temp.right = new BinaryTreeNode(Key key, Value val);
          else
             temp.left = new BinaryTreeNode(Key key, Value val);

}
```

Complexity: Tree Shape

Something Notable

- Many BSTs correspond to same input data.
- Cost of search/insert is proportional to depth of node.

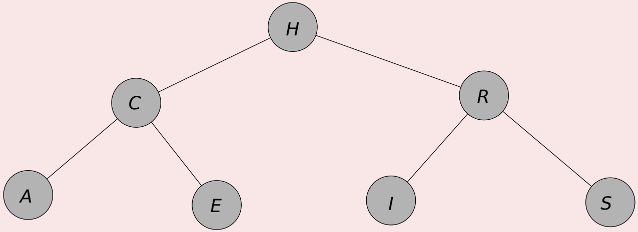
For example: Full Binary Tree

Complexity: Tree Shape

Something Notable

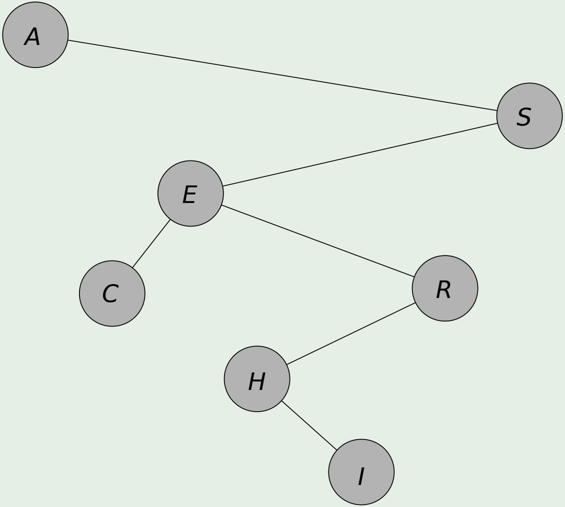
- Many BSTs correspond to same input data.
- Cost of search/insert is proportional to depth of node.

For example: Full Binary Tree



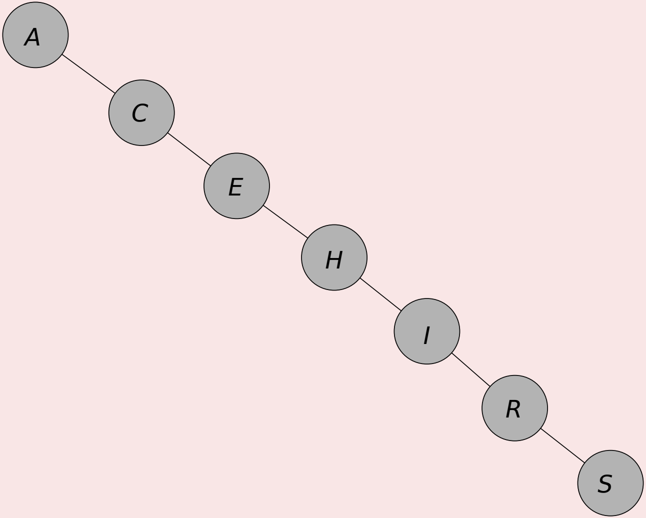
Other Examples

Example: Typical Tree



Other Examples

Example: Worst Case



Then, we want self-balancing trees

We depend on the height of the tree

Important, we want well balanced trees or near to the full tree structure... because going down the tree cost $O(h)$

We will look Next Class

- At a way to keep the binary trees well balanced...
- Examples of these techniques:
  - 2-3 trees
  - AA trees
  - **AVL trees**
  - Red-Black Trees
  - Splay Trees

Then, we want self-balancing trees

We depend on the height of the tree

Important, we want well balanced trees or near to the full tree structure... because going down the tree cost $O(h)$

We will look Next Class

- At a way to keep the binary trees well balanced...
- Examples of these techniques:
  - 2-3 trees
  - AA trees
  - **AVL trees**
  - Red-Black Trees
  - Splay Trees

Outline

1 Introduction
- Basic Concepts

2 BST Representation

3 Operations
- Get
- Put
- Minimum and Maximum
- Remove
- Tree Delete
- Examples of Deletion

Operations: Minimum

Minimum

Minimum(x)

1. while $x.left \neq$NIL
2. $\qquad x = x.left$
3. return x

Complexity

$$O(h) \qquad (1)$$

where h is the height of the tree $\Rightarrow$ **we look for well balanced trees.**

Operations: Minimum

Minimum

Minimum(x)

1. while $x.left \neq$NIL
2. $\quad x = x.left$
3. return x

Complexity

$$O\left(h\right) \tag{1}$$

where h is the height of the tree $\Rightarrow$ **we look for well balanced trees.**

Operations: Maximum

Maximum

Maximum(x)

1. while $x.right \neq$NIL
2. $\qquad x = x.right$
3. return x

Complexity

$$O(h) \tag{2}$$

where h is the height of the tree $\Rightarrow$ **we look for well balanced trees.**

Operations: Maximum

Maximum

Maximum(x)

1. while $x.right \neq$NIL
2. $\quad\quad x = x.right$
3. return x

Complexity

$$O\left(h\right) \tag{2}$$

where h is the height of the tree $\Rightarrow$ **we look for well balanced trees.**

Outline

1 Introduction
- Basic Concepts

2 BST Representation

3 Operations
- Get
- Put
- Minimum and Maximum
- Remove
- Tree Delete
- Examples of Deletion

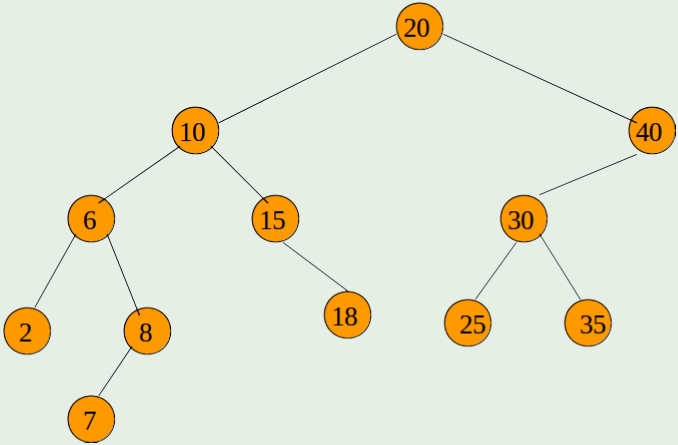
Operation: Remove

We have the following cases

- Element is in a leaf.
- Element is in a degree 1 node.
- Element is in a degree 2 node.

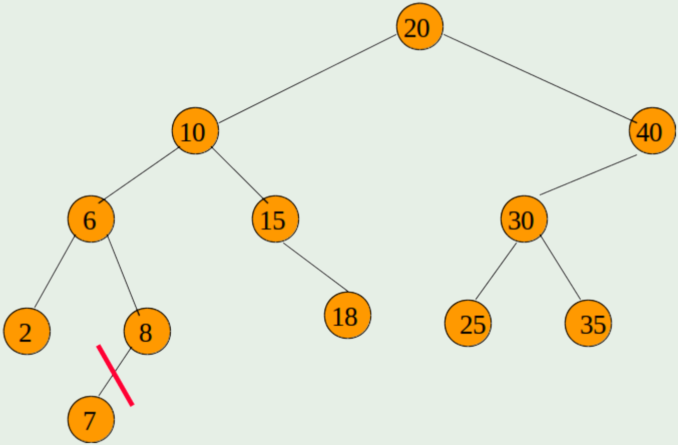
Remove from a Leaf

Remove a leaf element key = 7



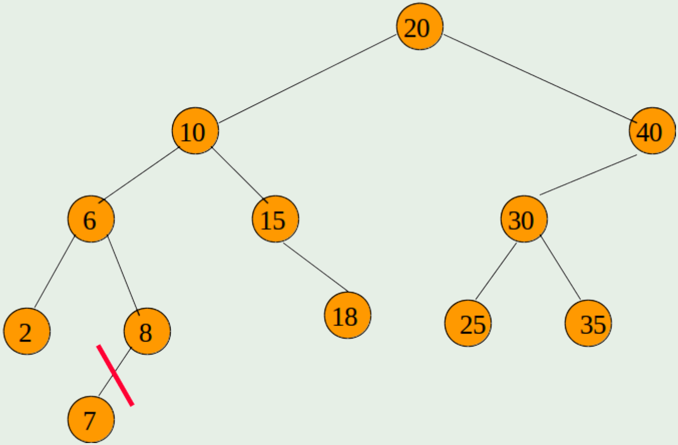
Remove from a Leaf

Remove a leaf element key = 7



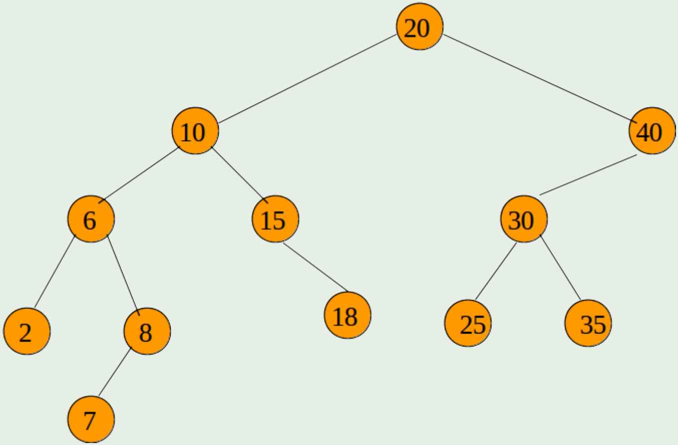
Remove from a Leaf

Remove a leaf element key = 7



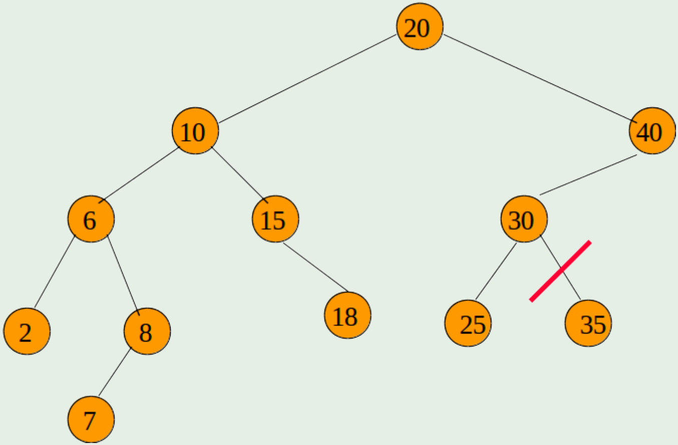
Remove from a Leaf

Remove a leaf element key = 35



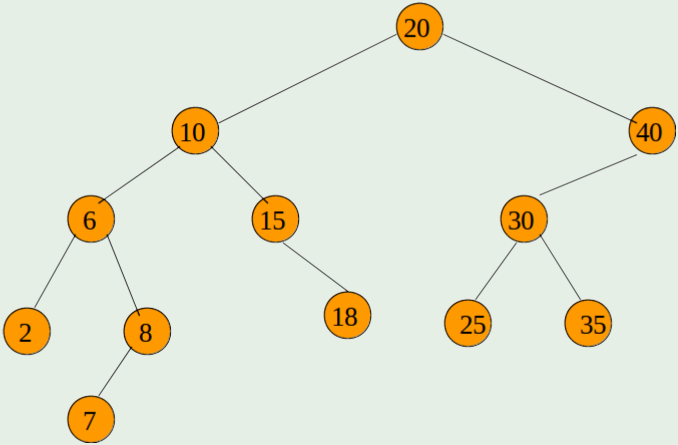
Remove from a Leaf

Remove a leaf element key = 35



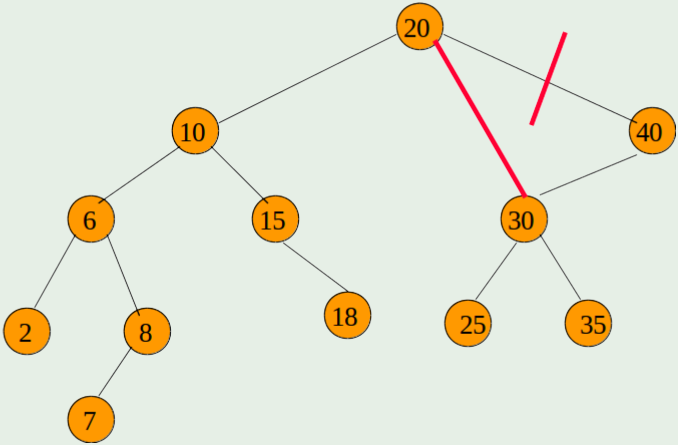
Remove From A Degree 1 Node

Remove from a degree 1 node key = 40



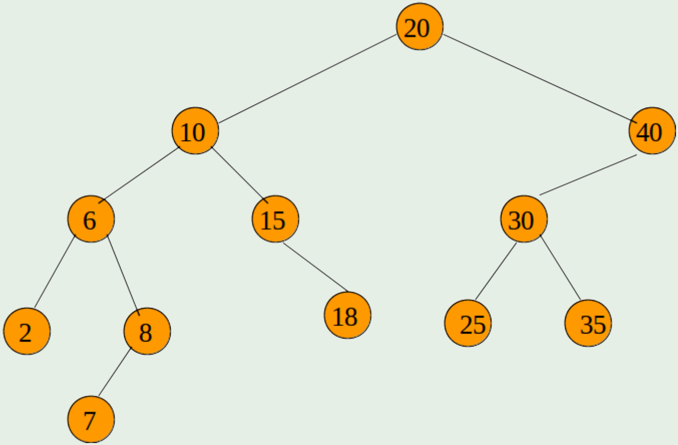
Remove From A Degree 1 Node

Remove from a degree 1 node key = 40



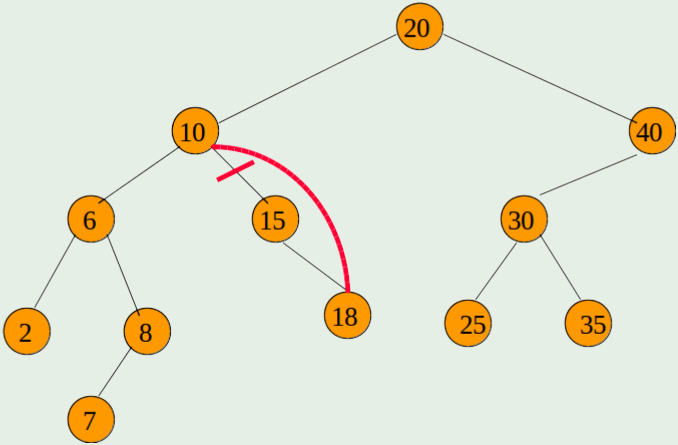
Remove From A Degree 1 Node

Remove from a degree 1 node key = 15



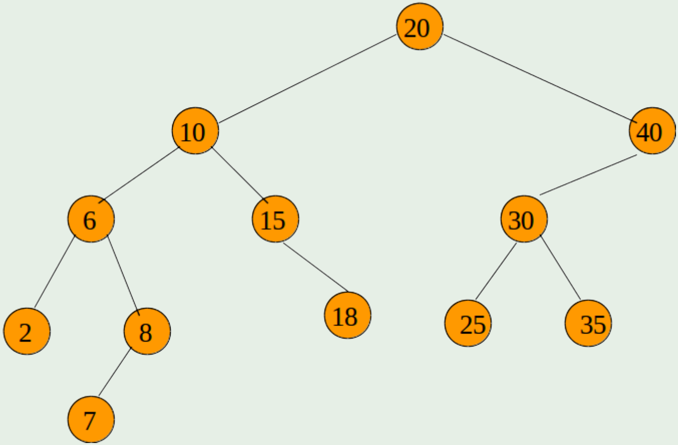
Remove From A Degree 1 Node

Remove from a degree 1 node key = 15



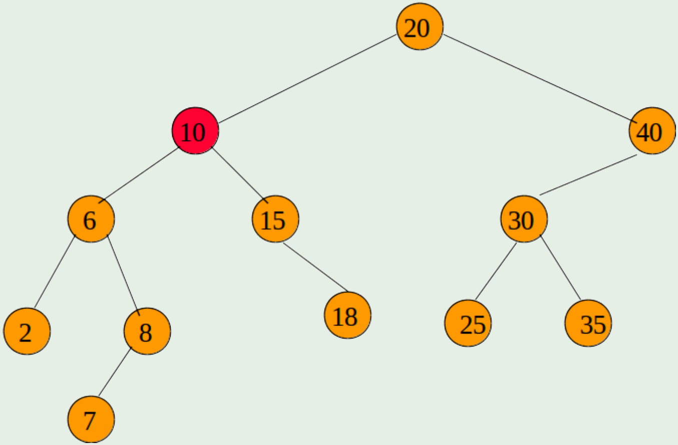
Remove From A Degree 2 Node

Remove from a degree 2 node key = 10



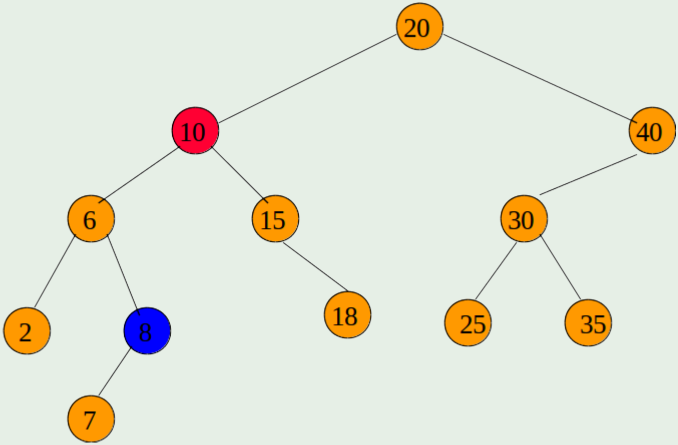
Remove From A Degree 2 Node

Remove from a degree 2 node key = 10



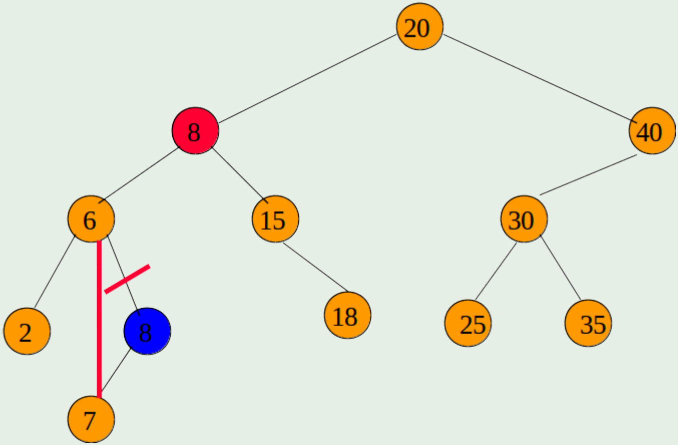
Remove From A Degree 2 Node

Replace with largest key in left subtree (or smallest in right subtree)



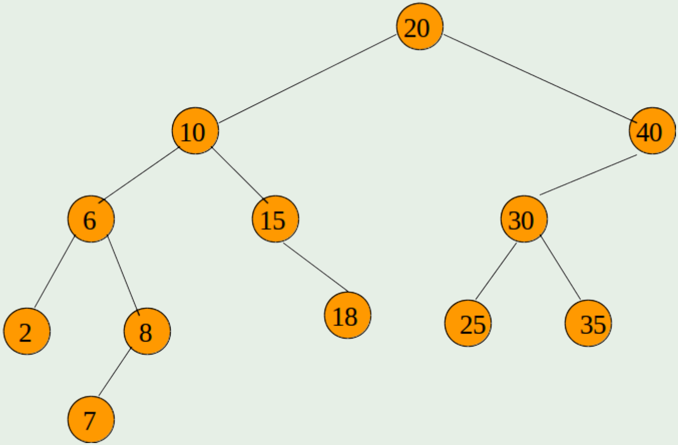
Remove From A Degree 2 Node

Largest key must be in a leaf or degree 1 node



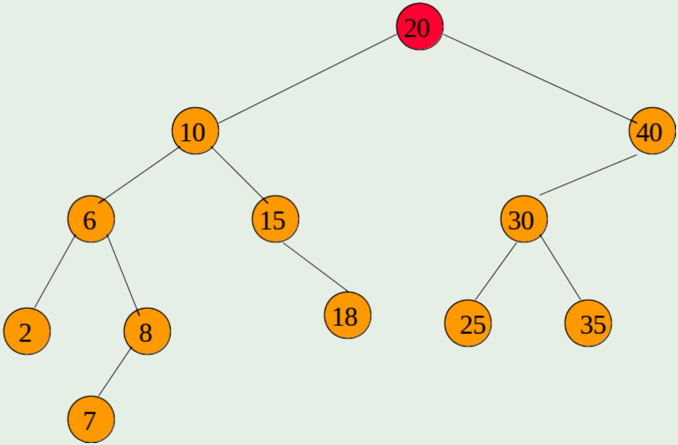
Another Example

Remove from a degree 2 node key = 20



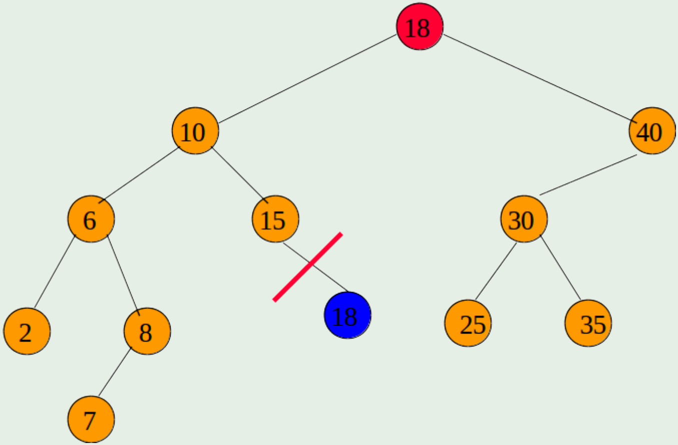
Another Example

Replace with largest in left subtree



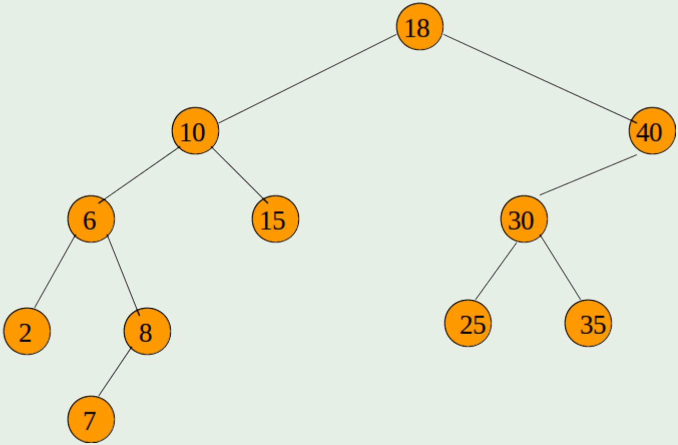
Another Example

Replace with largest in left subtree



Another Example

Replace with largest in left subtree



Outline

1 Introduction
- Basic Concepts

2 BST Representation

3 Operations
- Get
- Put
- Minimum and Maximum
- Remove
- Tree Delete
- Examples of Deletion

TREE-DELETE

TREE-DELETE(z)

```
if z.left == NIL
    Transplant(z, z.right)
elseif z.right == NIL
    Transplant(z, z.left)
else
        y=Tree-minimum(z.right)
        if y.p ≠ z
            Transplant(y, y.right)
            y.right = z.right
            y.right.p = y
        Transplant(z, y)
        y.left = z.left
        y.left.p = y
```

Case 1

- Basically if the element z to be deleted has a NIL left child simply replace z with that child!!!

TREE-DELETE

TREE-DELETE(z)

if $z.left$ == NIL
    Transplant($z, z.right$)
elseif $z.right$ == NIL
    Transplant($z, z.left$)
else
    y=Tree-minimum($z.right$)
    if $y.p \neq z$
        Transplant($y, y.right$)
        $y.right = z.right$
        $y.right.p = y$
    Transplant(z, y)
    $y.left = z.left$
    $y.left.p = y$

Case 2

- Basically if the element z to be deleted has a NIL right child simply replace z with that child!!!

TREE-DELETE

TREE-DELETE(z)

if $z.left$ == NIL
    Transplant($z, z.right$)
elseif $z.right$ == NIL
    Transplant($z, z.left$)
else
    y=Tree-minimum($z.right$)
    if $y.p \neq z$
        Transplant($y, y.right$)
        $y.right = z.right$
        $y.right.p = y$
    Transplant(z, y)
    $y.left = z.left$
    $y.left.p = y$

Case 3

- The z element has not empty children you need to find the successor of it.

TREE-DELETE

TREE-DELETE(z)

```
if z.left == NIL
    Transplant(z, z.right)
elseif z.right == NIL
    Transplant(z, z.left)
else
    y=Tree-minimum(z.right)
    if y.p ≠ z
        Transplant(y, y.right)
        y.right = z.right
        y.right.p = y
    Transplant(z, y)
    y.left = z.left
    y.left.p = y
```

Case 4

- if $y.p \neq z$ then $y.right$ takes the position of y after all $y.left ==$ NIL
  - take $z.right$ and make it the new $right$ of y
  - make the $(y.right == z.right).p$ equal to y

TREE-DELETE

TREE-DELETE(z)

```
if z.left == NIL
    Transplant(z, z.right)
elseif z.right == NIL
    Transplant(z, z.left)
else
    y=Tree-minimum(z.right)
    if y.p ≠ z
        Transplant(y, y.right)
        y.right = z.right
        y.right.p = y
    Transplant(z, y)
    y.left = z.left
    y.left.p = y
```

Case 4

- put y in the position of z
- make $y.left$ equal to $z.left$
- make the $(y.left == z.left).p$ equal to y

Support Operations

Transplant(u, v)

if $u.p == $ NIL

  $root = v$

elseif $u == u.p.left$

  $u.p.left = v$

else $u.p.right = v$

if $v \neq$NIL

  $v.p = u.p$

Case 1

- If u is the root then make the root equal to v

Support Operations

Transplant(u, v)

if $u.p == $ NIL

  $root = v$

elseif $u == u.p.left$

  $u.p.left = v$

else $u.p.right = v$

if $v \neq$NIL

  $v.p = u.p$

Case 2

- if u is the left child make the left child of the parent of u equal to v

Support Operations

Transplant(u, v)

```
if u.p == NIL
    root = v
elseif u == u.p.left
    u.p.left = v
else u.p.right = v
if v ≠ NIL
    v.p = u.p
```

Case 3

- Similar to the second case, but for right child

Support Operations

Transplant(u, v)

1. if $u.p == \text{NIL}$
2. $root = v$
3. elseif $u == u.p.left$
4. $u.p.left = v$
5. else $u.p.right = v$
6. if $v \neq$NIL
7. $v.p = u.p$

Case 4

- If $v \neq$ NIL then make the parent of v the parent of u

Complexity

Height of the BT

$$O\left(height\right)$$

Outline

1 Introduction
- Basic Concepts

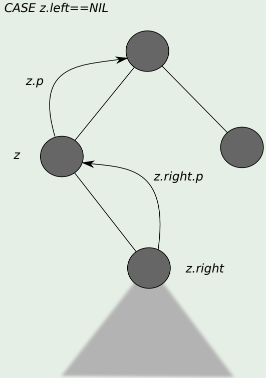
2 BST Representation

3 Operations
- Get
- Put
- Minimum and Maximum
- Remove
- Tree Delete
- Examples of Deletion

Example: Deletion in BST

Case $z.left == NIL$

- if $z.left ==$ NIL
- Transplant$(T, z, z.right)$...

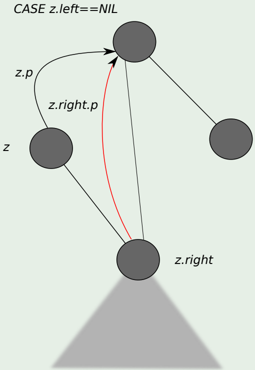


Example: Deletion in BST

Case $z.left == NIL$

Transplant$(T, z, z.right)$

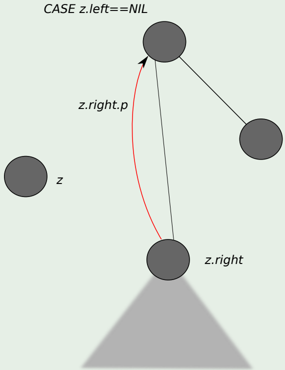
- elseif $z == z.p.left$
- $\quad z.p.left = z.right$
- if $z.right \neq$NIL
- $\quad z.right.p = z.p$



Example: Deletion in BST

Case $z.left == NIL$

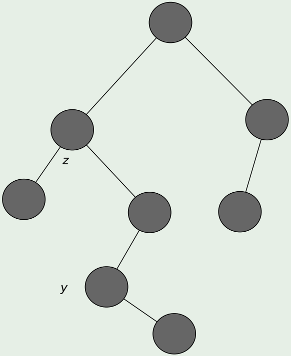
Remove the node z once you get out of the procedure



Another Example: Deletion in BST

Case $z.left \neq NIL$ and $z.right \neq NIL$

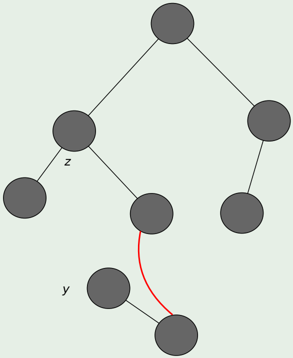
- y=Tree-minimum$(z.right)$



Another Example: Deletion in BST

Case $z.left \neq NIL$ and $z.right \neq NIL$

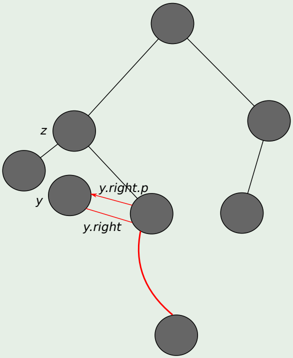
- if $y.p \neq z$
- Transplant$(T, y, y.right)$



Another Example: Deletion in BST

Case $z.left \neq NIL$ and $z.right \neq NIL$

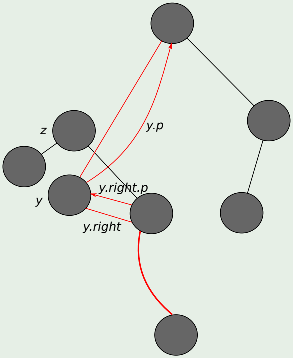
- $y.right = z.right$
- $y.right.p = y$



Another Example: Deletion in BST

Case $z.left \neq NIL$ and $z.right \neq NIL$

- Transplant(T, z, y)
- $y.left = z.left$
- $y.left.p = y$



Another Example: Deletion in BST

Case $z.left \neq NIL$ and $z.right \neq NIL$

- Transplant(T, z, y)
- $y.left = z.left$
- $y.left.p = y$