Principles of Computer Science II

Abstract Data Types

Ioannis Chatzigiannakis

Sapienza University of Rome

Lecture 10

Binary Search Trees

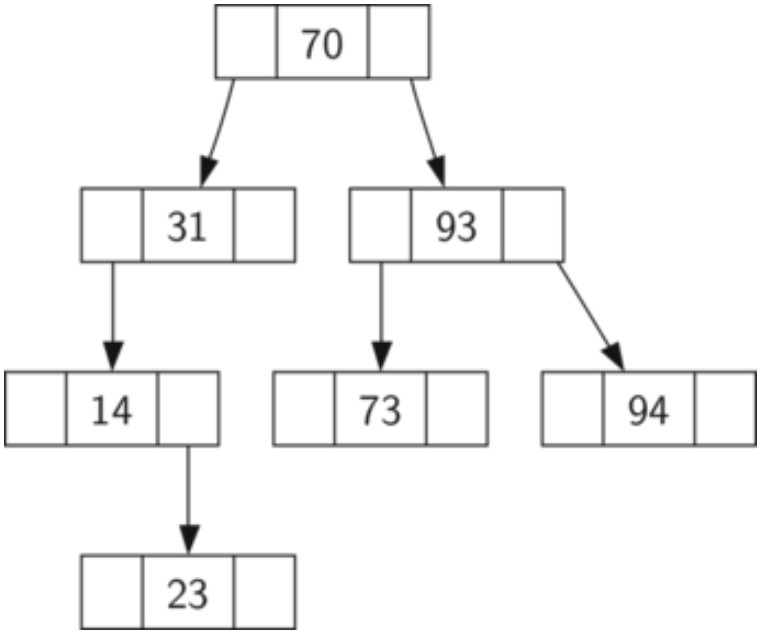
Binary trees organize data depending on the values of the elements,

- keys that are less than the parent are found in the left subtree,
- keys that are greater than the parent are found in the right subtree.

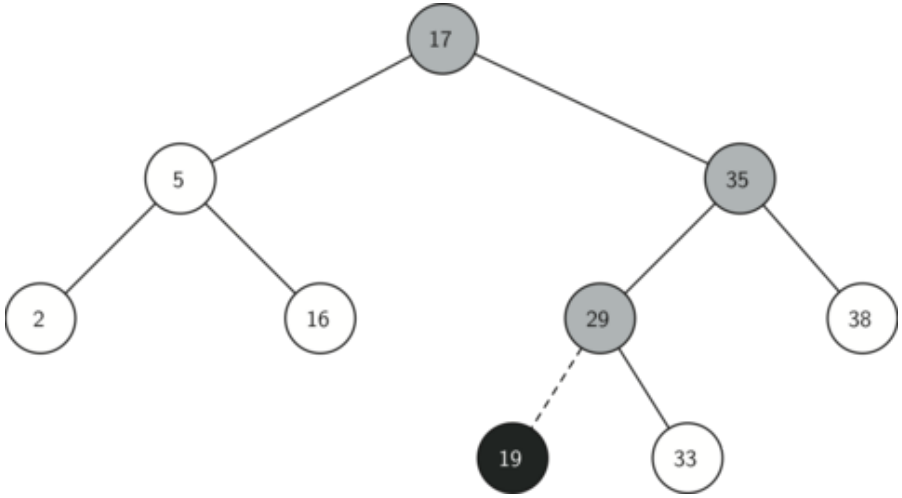
We will call this the bst property.

- Binary Trees are implemented using 2 pointers on each node.

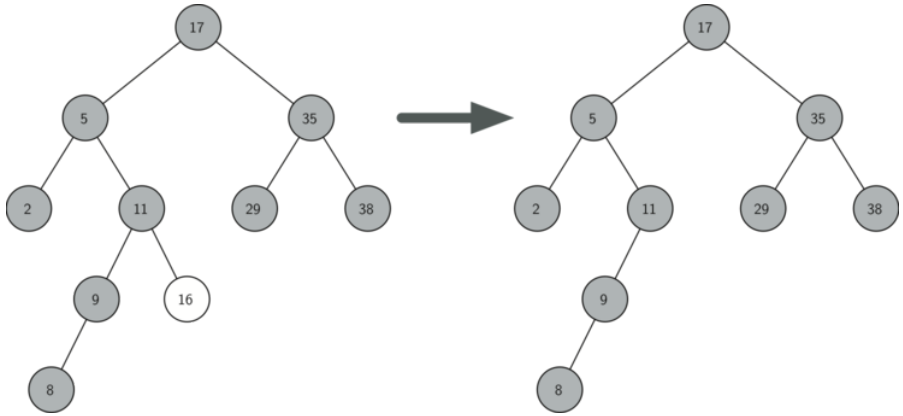
Binary Search Tree



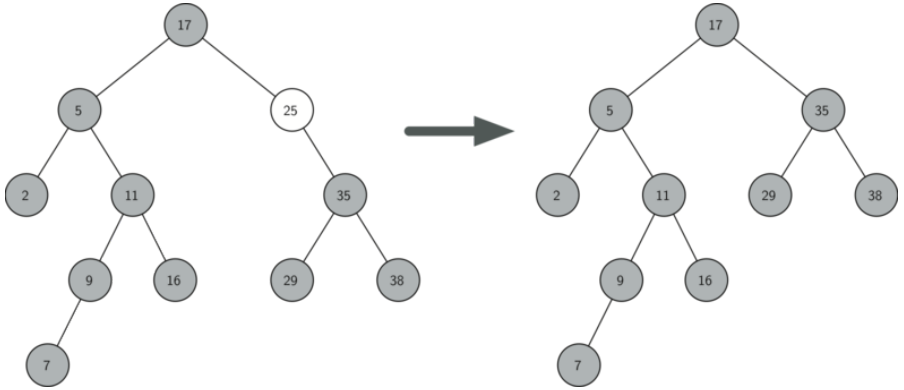
Binary Search Tree: Node Insertion



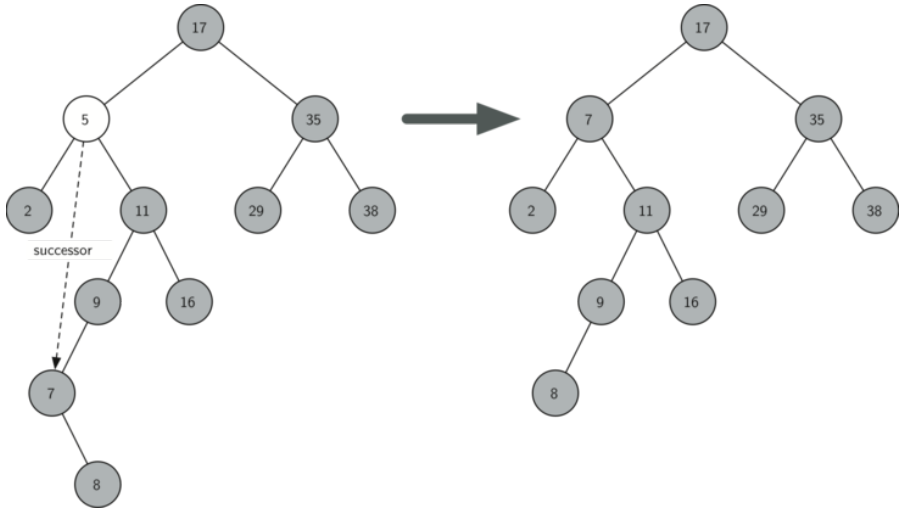
Binary Search Tree: Node Deletion



Binary Search Tree: Node Deletion



Binary Search Tree: Node Deletion



Tree Node

```
class TreeNode:
    def __init__(self,key,val,left=None,right=None,parent=
        None):
        self.key = key
        self.payload = val
        self.leftChild = left
        self.rightChild = right
        self.parent = parent

    def hasLeftChild(self):
        return self.leftChild

    def hasRightChild(self):
        return self.rightChild

    def hasAnyChildren(self):
        return self.rightChild or self.leftChild
```

Tree Node

```
    def hasBothChildren(self):
        return self.rightChild and self.leftChild

    def isLeftChild(self):
        return self.parent and self.parent.leftChild ==
            self

    def isRightChild(self):
        return self.parent and self.parent.rightChild ==
            self

    def isRoot(self):
        return not self.parent

    def isLeaf(self):
        return not (self.rightChild or self.leftChild)
```

Tree Node

```
    def replaceNodeData(self,key,value,lc,rc):
        self.key = key
        self.payload = value
        self.leftChild = lc
        self.rightChild = rc
        if self.hasLeftChild():
            self.leftChild.parent = self
        if self.hasRightChild():
            self.rightChild.parent = self
```

Binary Search Tree

```
class BinarySearchTree:

    def __init__(self):
        self.root = None
        self.size = 0

    def length(self):
        return self.size

    def __len__(self):
        return self.size
```

Binary Search Tree – put

```
    def put(self,key,val):
        if self.root:
            self._put(key,val,self.root)
        else:
            self.root = TreeNode(key,val)
        self.size = self.size + 1
```

Binary Search Tree – put

```
def _put(self,key,val,currentNode):
    if key < currentNode.key:
        if currentNode.hasLeftChild():
            self._put(key,val,currentNode.leftChild)
        else:
            currentNode.leftChild = TreeNode(key,val
                ,parent=currentNode)
    else:
        if currentNode.hasRightChild():
            self._put(key,val,currentNode.rightChild
                )
        else:
            currentNode.rightChild = TreeNode(key,
                val,parent=currentNode)
```

Binary Search Tree – get

```
def get(self,key):
    if self.root:
        res = self._get(key,self.root)
        if res:
            return res.payload
        else:
            return None
    else:
        return None

def _get(self,key,currentNode):
    if not currentNode:
        return None
    elif currentNode.key == key:
        return currentNode
    elif key < currentNode.key:
        return self._get(key,currentNode.leftChild)
    else:
        return self._get(key,currentNode.rightChild)
```

Binary Search Tree – remove

```
def remove(self,currentNode):
     if currentNode.isLeaf(): #leaf
       if currentNode == currentNode.parent.leftChild:
           currentNode.parent.leftChild = None
       else:
           currentNode.parent.rightChild = None

     elif currentNode.hasBothChildren(): #interior
       succ = currentNode.findSuccessor()
       succ.spliceOut()
       currentNode.key = succ.key
       currentNode.payload = succ.payload
```

Binary Search Tree – remove

```
else: # this node has one child
  if currentNode.hasLeftChild():
    if currentNode.isLeftChild():
    currentNode.leftChild.parent = currentNode.parent
    currentNode.parent.leftChild = currentNode.
        leftChild

    elif currentNode.isRightChild():
    currentNode.leftChild.parent = currentNode.parent
    currentNode.parent.rightChild = currentNode.
        leftChild
    else:
    currentNode.replaceNodeData(currentNode.leftChild.
        key,
            currentNode.leftChild.payload,
            currentNode.leftChild.leftChild,
            currentNode.leftChild.rightChild)
```

Binary Search Tree – remove

```
1       else:
2         if currentNode.isLeftChild():
3         currentNode.rightChild.parent = currentNode.parent
4         currentNode.parent.leftChild = currentNode.
              rightChild
5         elif currentNode.isRightChild():
6         currentNode.rightChild.parent = currentNode.parent
7         currentNode.parent.rightChild = currentNode.
              rightChild
8         else:
9         currentNode.replaceNodeData(currentNode.rightChild.
              key,
10                  currentNode.rightChild.payload,
11                  currentNode.rightChild.leftChild,
12                  currentNode.rightChild.rightChild)
```

Binary Search Tree – remove / find successor

```
1     def findSuccessor(self):
2       succ = None
3       if self.hasRightChild():
4           succ = self.rightChild.findMin()
5       else:
6           if self.parent:
7                   if self.isLeftChild():
8                       succ = self.parent
9                   else:
10                      self.parent.rightChild = None
11                      succ = self.parent.findSuccessor()
12                      self.parent.rightChild = self
13      return succ
```

Binary Search Tree – remove / find min

```
1     def findMin(self):
2       current = self
3       while current.hasLeftChild():
4           current = current.leftChild
5       return current
```

Binary Search Tree – remove / spliceOut

```
1   def spliceOut(self):
2       if self.isLeaf():
3       if self.isLeftChild():
4         self.parent.leftChild = None
5       else:
6         self.parent.rightChild = None
7       elif self.hasAnyChildren():
8       if self.hasLeftChild():
9         if self.isLeftChild():
10            self.parent.leftChild = self.leftChild
11        else:
12            self.parent.rightChild = self.leftChild
13        self.leftChild.parent = self.parent
14      else:
15        if self.isLeftChild():
16            self.parent.leftChild = self.rightChild
17        else:
18            self.parent.rightChild = self.rightChild
19        self.rightChild.parent = self.parent
```