

Principles of Computer Science II

Abstract Data Types

Ioannis Chatzigiannakis

Sapienza University of Rome

Lecture 17



Binary Search Trees

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

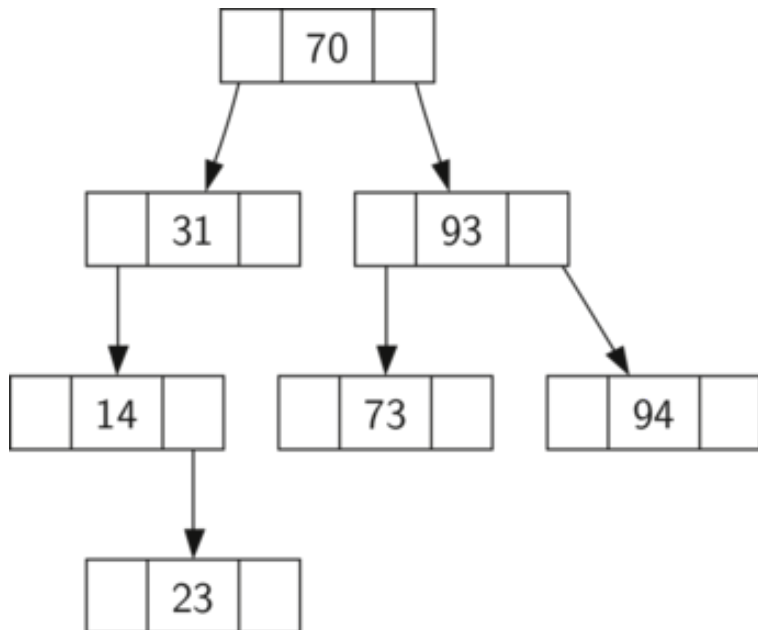
- ▶ keys that are less than the parent are found in the left subtree,
- ▶ keys that are greater than the parent are found in the right subtree.

We will call this the bst property.

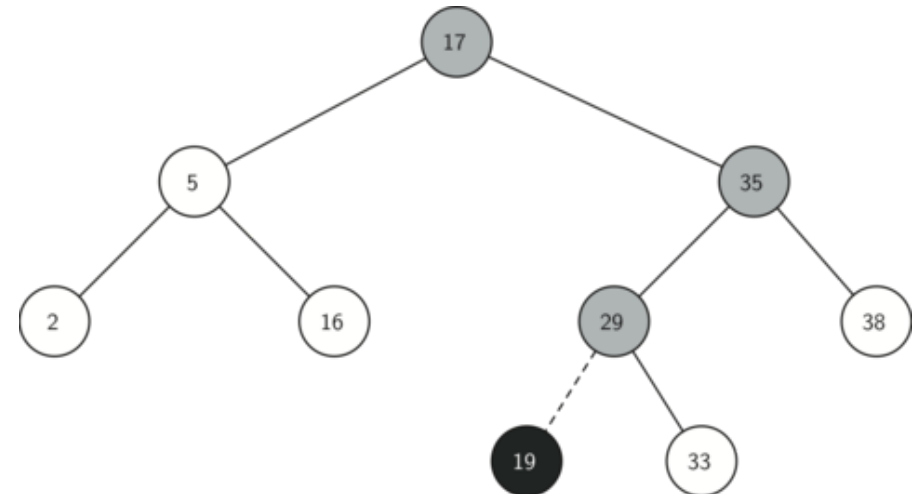
- ▶ Binary Trees are implemented using 2 pointers on each node.



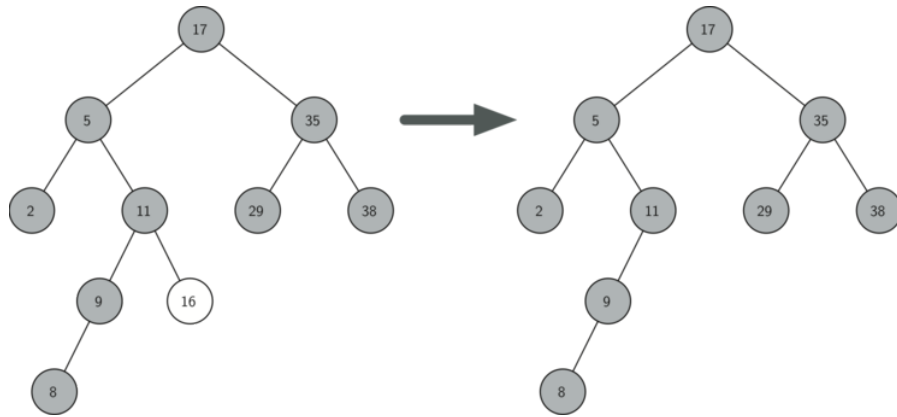
Binary Search Tree



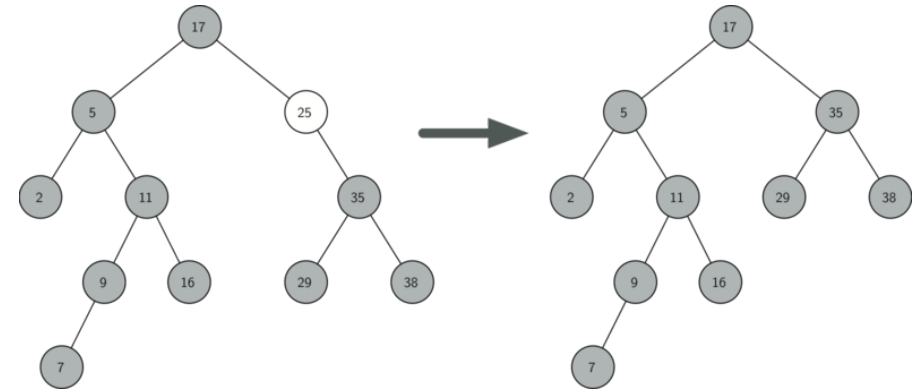
Binary Search Tree: Node Insertion



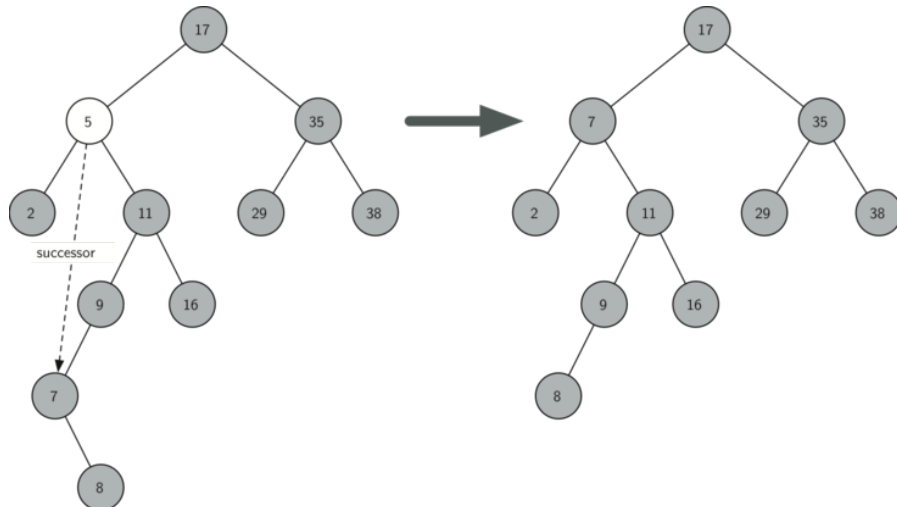
Binary Search Tree: Node Deletion



Binary Search Tree: Node Deletion



Binary Search Tree: Node Deletion



Tree Node

```

1 class TreeNode:
2     def __init__(self, key, val, left=None, right=None, parent=None):
3         self.key = key
4         self.payload = val
5         self.leftChild = left
6         self.rightChild = right
7         self.parent = parent
8
9     def hasLeftChild(self):
10        return self.leftChild
11
12    def hasRightChild(self):
13        return self.rightChild
14
15    def hasAnyChildren(self):
16        return self.rightChild or self.leftChild
    
```



Tree Node

```
1 def hasBothChildren(self):
2     return self.rightChild and self.leftChild
3
4 def isLeftChild(self):
5     return self.parent and self.parent.leftChild ==
6         self
7
8 def isRightChild(self):
9     return self.parent and self.parent.rightChild ==
10        self
11
12 def isRoot(self):
13     return not self.parent
14
15 def isLeaf(self):
16     return not (self.rightChild or self.leftChild)
```



Tree Node

```
1 def replaceNodeData(self, key, value, lc, rc):
2     self.key = key
3     self.payload = value
4     self.leftChild = lc
5     self.rightChild = rc
6     if self.hasLeftChild():
7         self.leftChild.parent = self
8     if self.hasRightChild():
9         self.rightChild.parent = self
```



Binary Search Tree

```
1 class BinarySearchTree:
2
3     def __init__(self):
4         self.root = None
5         self.size = 0
6
7     def length(self):
8         return self.size
9
10    def __len__(self):
11        return self.size
```



Binary Search Tree – put

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



Binary Search Tree – put

```
1 def _put(self, key, val, currentNode):
2     if key < currentNode.key:
3         if currentNode.hasLeftChild():
4             self._put(key, val, currentNode.leftChild)
5         else:
6             currentNode.leftChild = TreeNode(key, val
7                 , parent=currentNode)
8     else:
9         if currentNode.hasRightChild():
10            self._put(key, val, currentNode.rightChild
11                )
12        else:
13            currentNode.rightChild = TreeNode(key,
14                val, parent=currentNode)
```



Binary Search Tree – get

```
1 def get(self, key):
2     if self.root:
3         res = self._get(key, self.root)
4         if res:
5             return res.payload
6         else:
7             return None
8     else:
9         return None
10
11 def _get(self, key, currentNode):
12     if not currentNode:
13         return None
14     elif currentNode.key == key:
15         return currentNode
16     elif key < currentNode.key:
17         return self._get(key, currentNode.leftChild)
18     else:
19         return self._get(key, currentNode.rightChild)
```



Binary Search Tree – remove

```
1 def remove(self, currentNode):
2     if currentNode.isLeaf(): #leaf
3         if currentNode == currentNode.parent.leftChild:
4             currentNode.parent.leftChild = None
5         else:
6             currentNode.parent.rightChild = None
7
8     elif currentNode.hasBothChildren(): #interior
9         succ = currentNode.findSuccessor()
10        succ.spliceOut()
11        currentNode.key = succ.key
12        currentNode.payload = succ.payload
```



Binary Search Tree – remove

```
1 else: # this node has one child
2     if currentNode.hasLeftChild():
3         if currentNode.isLeftChild():
4             currentNode.leftChild.parent = currentNode.parent
5             currentNode.parent.leftChild = currentNode.
6                 leftChild
7
8         elif currentNode.isRightChild():
9             currentNode.leftChild.parent = currentNode.parent
10            currentNode.parent.rightChild = currentNode.
11                leftChild
12        else:
13            currentNode.replaceNodeData(currentNode.leftChild.
14                key,
15                currentNode.leftChild.payload,
16                currentNode.leftChild.leftChild,
17                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
```



5th Assignment

- ▶ <https://www.rosalind.info/>
 - ▶ Complete the following **challenges**:
prot, splc, tran, hamm, tree, pdst, sseq, lcsq, orf, perm, grph, inod
 - ▶ <http://rosalind.info/problems/{challenge}>
- ▶ Create a GitHub repository and upload the code for each exercise.
- ▶ Email ichatz@diag.uniroma1.it
Subject: [PCS2] Homework 5
A .zip or a .tar.gz file with your python solutions, for all challenges.
Also send your account user account link:
<http://rosalind.info/users/{username}>
- ▶ **Deadline: 17/December/2019, 23:59**

