Principles of Computer Science II

Abstract Data Types

Ioannis Chatzigiannakis

Sapienza University of Rome

Lecture 13

Binary Search Trees

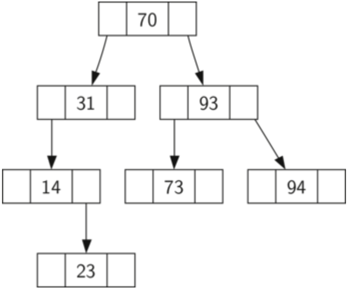
Binary trees organize data depending on the values of the elements,

- 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.

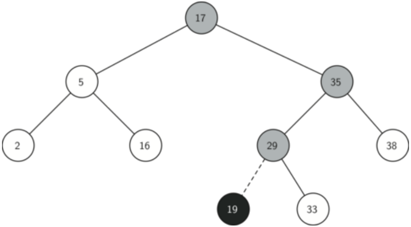
We will call this the bst property.

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

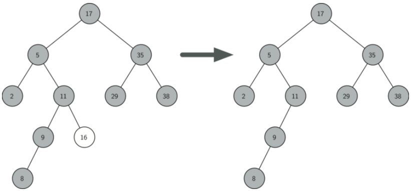
Binary Search Tree



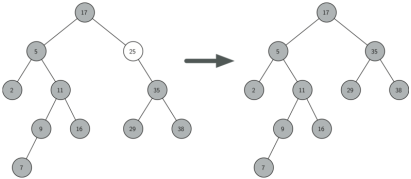
Binary Search Tree: Node Insertion



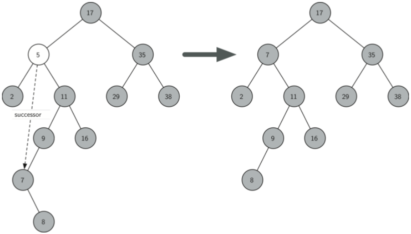
Binary Search Tree: Node Deletion



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

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

Binary Search Tree – remove / find successor

```
def findSuccessor(self):
  succ = None
  if self.hasRightChild():
      succ = self.rightChild.findMin()
  else:
      if self.parent:
             if self.isLeftChild():
                 succ = self.parent
             else:
                 self.parent.rightChild = None
                 succ = self.parent.findSuccessor()
                 self.parent.rightChild = self
  return succ
```

Binary Search Tree – remove / find min

```
def findMin(self):
  current = self
  while current.hasLeftChild():
      current = current.leftChild
  return current
```

Binary Search Tree – remove / spliceOut

```
def spliceOut(self):
    if self.isLeaf():
        if self.isLeftChild():
              self.parent.leftChild = None
        else:
              self.parent.rightChild = None
    elif self.hasAnyChildren():
        if self.hasLeftChild():
              if self.isLeftChild():
                  self.parent.leftChild = self.leftChild
              else:
                  self.parent.rightChild = self.leftChild
              self.leftChild.parent = self.parent
        else:
              if self.isLeftChild():
                  self.parent.leftChild = self.rightChild
              else:
                  self.parent.rightChild = self.rightChild
              self.rightChild.parent = self.parent
```