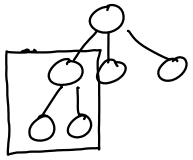
Binary Search Trees

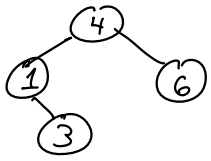
Tree — either empty or a node with zero or more trees

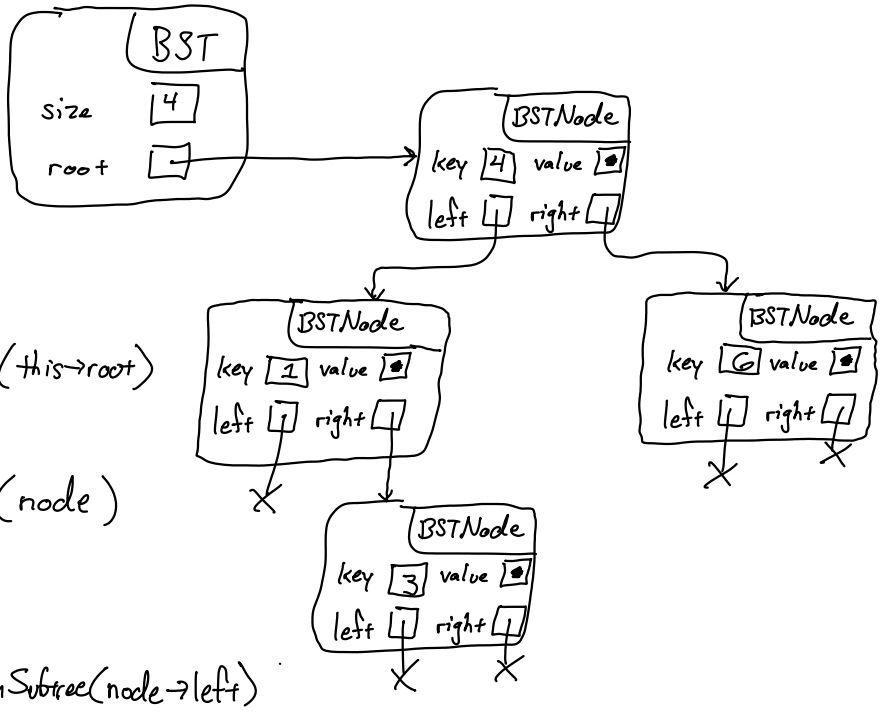


Binary Tree — a tree where each node has
at most one left child and
at most one right child

Binary Search Tree — a binary tree where, for every node, all left descendants have smaller keys than the node and all right descendants have larger keys than the node

---





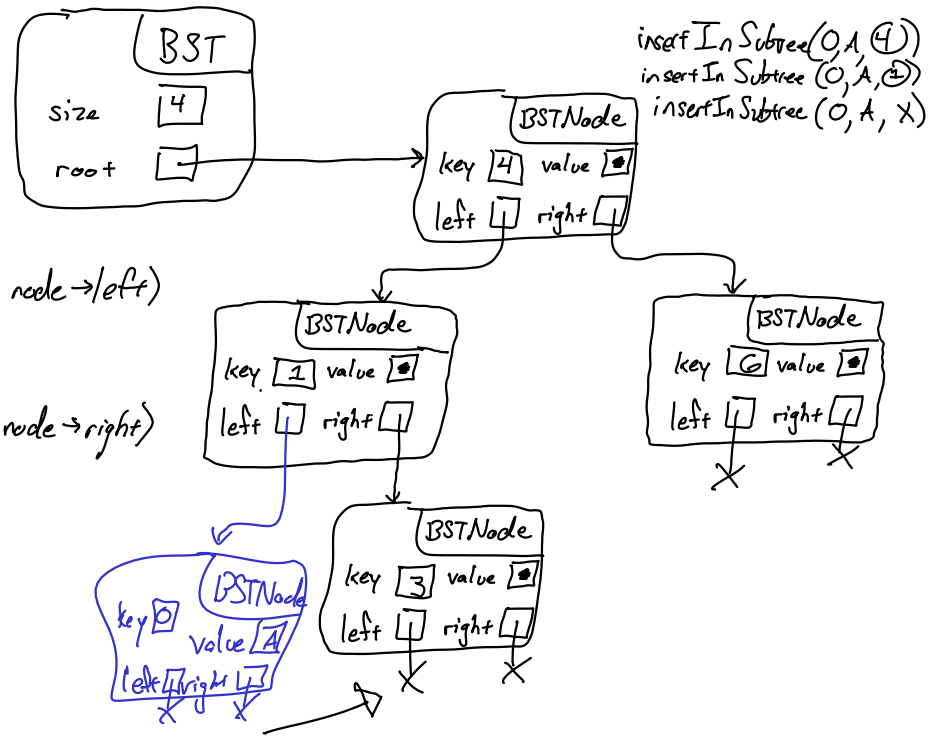

```
Method getMinKey()
    Return getMinKeyInSubtree(this->root)
EndMethod

Method getMinKeyInSubtree(node)
    If node->left == nullptr:
        Return node->key
    Else
        Return this->getMinKeyInSubtree(node->left)
    EndIf
EndMethod
```

```
Method insert(K key, V value)
    this->root <- insertInSubtree(key, value, this->root)
End Method

Method insertInSubtree(K key, V value, BSTNode<K,V>* node)
    If node == nullptr:
        Return new node w/ key, value
    Else If key == node->key:
        (!!)
    Else If key < node->key:
        node->left <- insertInSubtree(key, value, node->left)
        Return node
    Else:
        node->right <- insertInSubtree(key, value, node->right)
        Return node
    EndIf
End Method
```

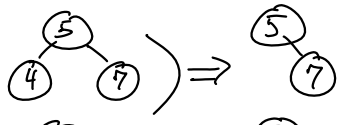
Whoever calls insertInSubtree MUST replace the node that they passed in with the node that insertInSubtree returns

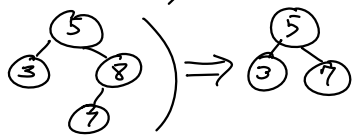


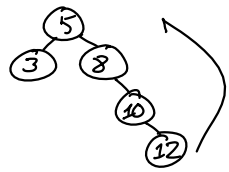
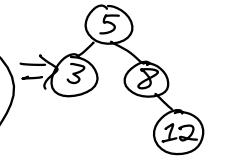
```
Method remove(K key)
    removeInSubtree(key, this->root)
EndMethod
```

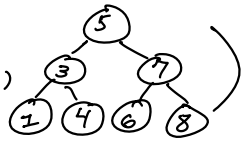
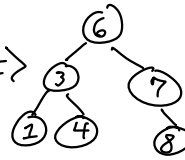
Caller must replace node w/ return value

```
Method removeInSubtree(K key, ... node)
    If node == nullptr:
        (:|)
    Else If key < node->key:
        node->left ← removeInSubtree(key, node->left)
        Return node
    Else If key > node->key:
        node->right ← removeInSubtree(key, node->right)
        Return node
    Else If node->left == nullptr && node->right == nullptr:
        Return nullptr
    Else If node has only left child:
        Return node->left
    Else If node has only right child:
        Return node->right
    Else:
        (we have both children)
        smallkey ← getMinKeyInSubtree(node->right)
        val ← get(key)
        node->key = smallkey
        node->value = val
        node->right ← removeInSubtree(smallkey, node->right)
        Return node
    EndIf
EndMethod
```

no children: remove IS(4, ) ⇒

left child: remove IS(8, ) ⇒

right child: remove IS(10, ) ⇒ 

both children: remove IS(5, ) ⇒ 

go find the smallest node on the right (or the largest on the left) and replace the root's key, value w/ it