Tree: Data Structure

Is made up of nodes

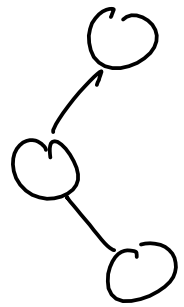
Root node (or it's empty)

Nodes: Data

Children: other nodes

Binary Tree: Tree w at most 2 children per node

Each child: left or right

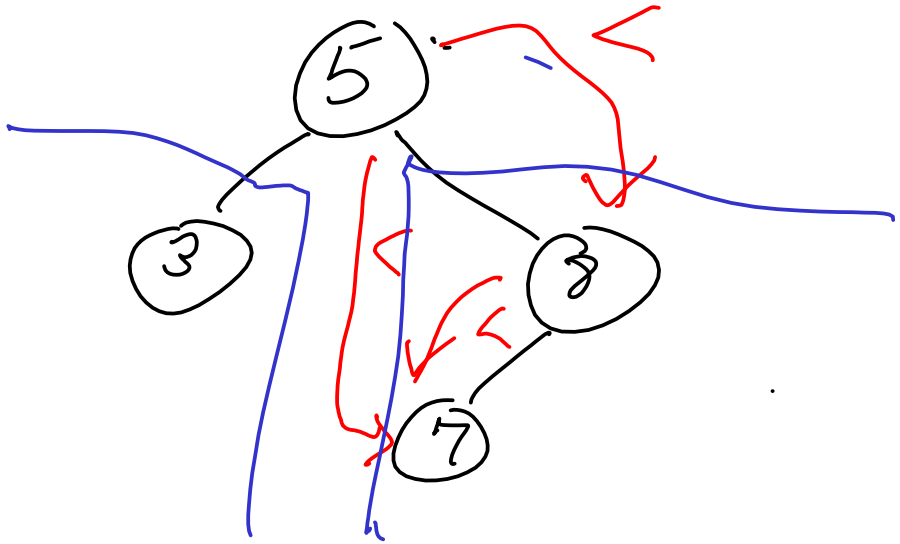


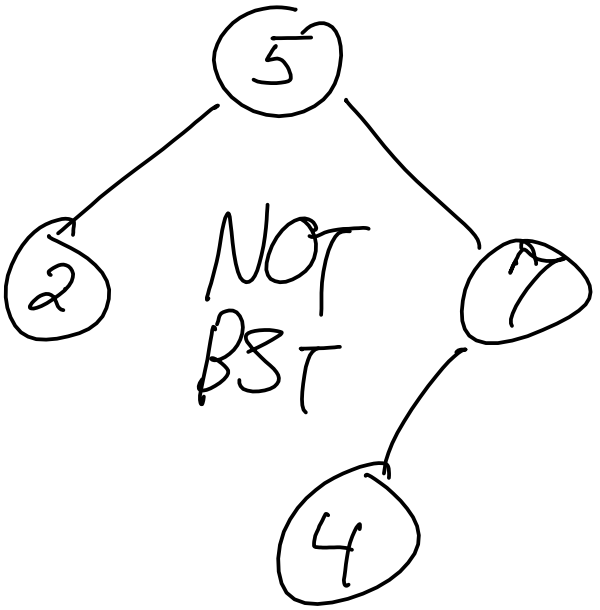
Binary Search Tree:

Binary Tree where data in left subtree < data parent

right " " > " " " "

Forevery node





Dictionary (map, finite mapping, ...):

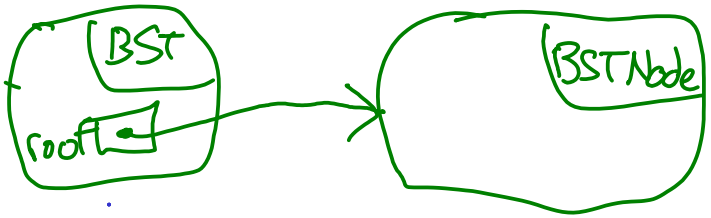
Abstract Data Type

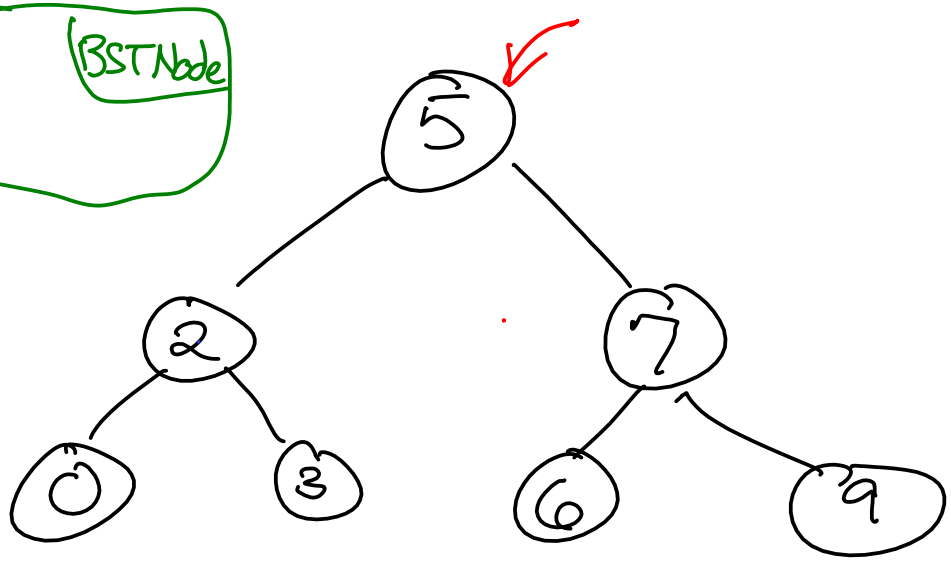
K(ey) V(alue)

- get(K)
- put(K, V)
- ~~remove~~(K)
- get_size()

Use BST to implement Dictionary

BST store at each node:
Key (sort) Value (doesn't sort)





Recursive

OO

Iterative

→ Imperative

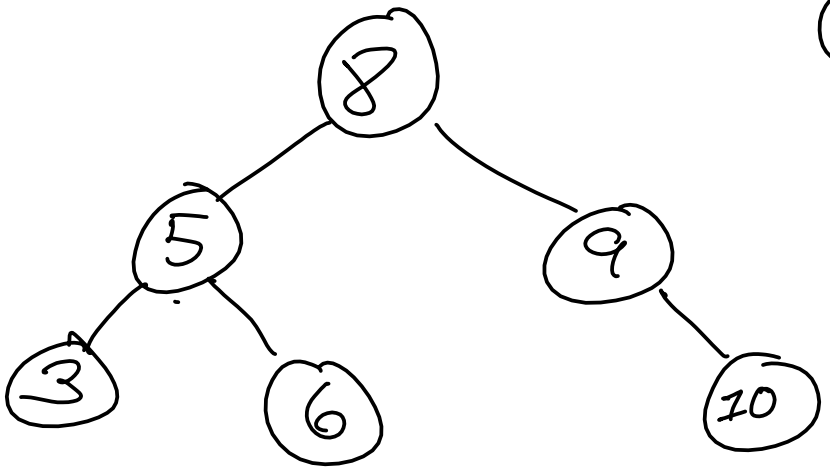
↳ Procedural

```
Method get(key):
    If key = this->key
        Return this->value
    Else If key < this->key
        If this->left ≠ NULL
            Return this->left->get(key)
    Else
        If this->right ≠ NULL
            Return this->right->get(key)
    End If
    Throw exception
```

```
Function get(root, key):
    current ← root
    While current ≠ NULL
        If key = current->key
            Return current->value
        Else If key < current->key
            current ← current->left
        Else
            current ← current->right
    End If
```

```
Method get_min()
    If this->left = NULL
        Return this->key
    Else
        Return this->left ->get_min()
    End If
```

Case : no children



Remove 6

Method Remove ¬L¬R (key)

Parent ← this → find_parent_of(key)

If parent→right ≠ NULL and parent→right→key == key :

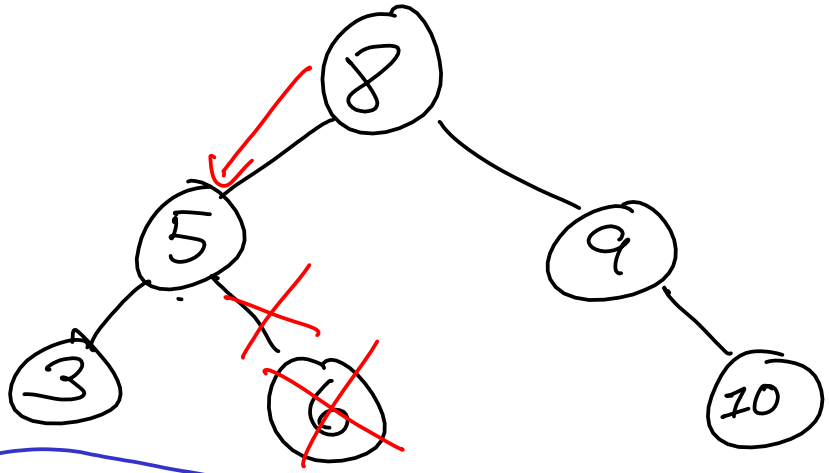
delete ....

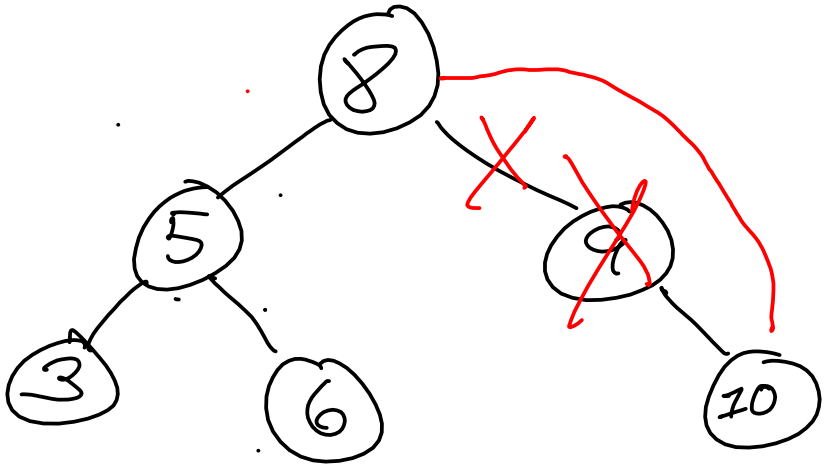
parent→right = NULL

Else

. . . .

End If





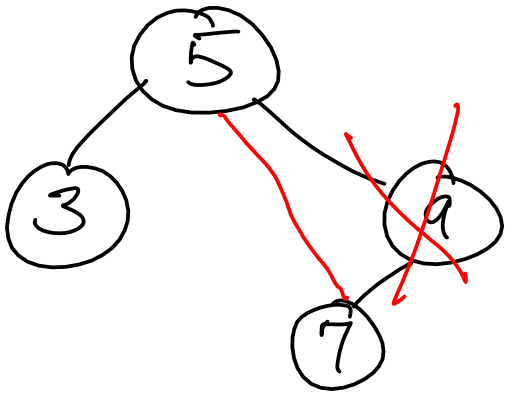
Delete 9

Method Remove$^{R \to L}$ (key)

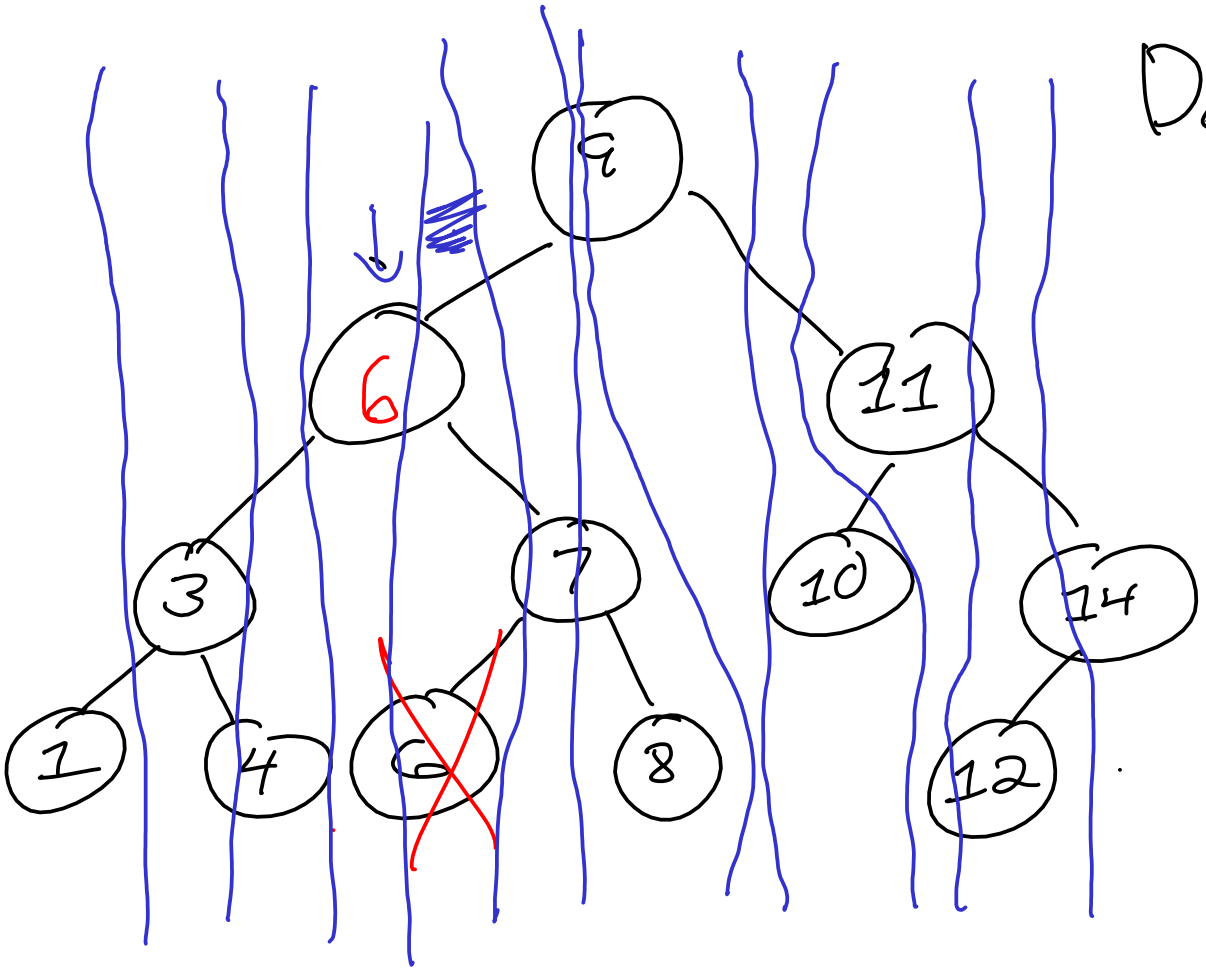
parent ← find_parent(key)

since child of that parent has 1 child:

replace our parent's pointer with the grandchild



Delete 5



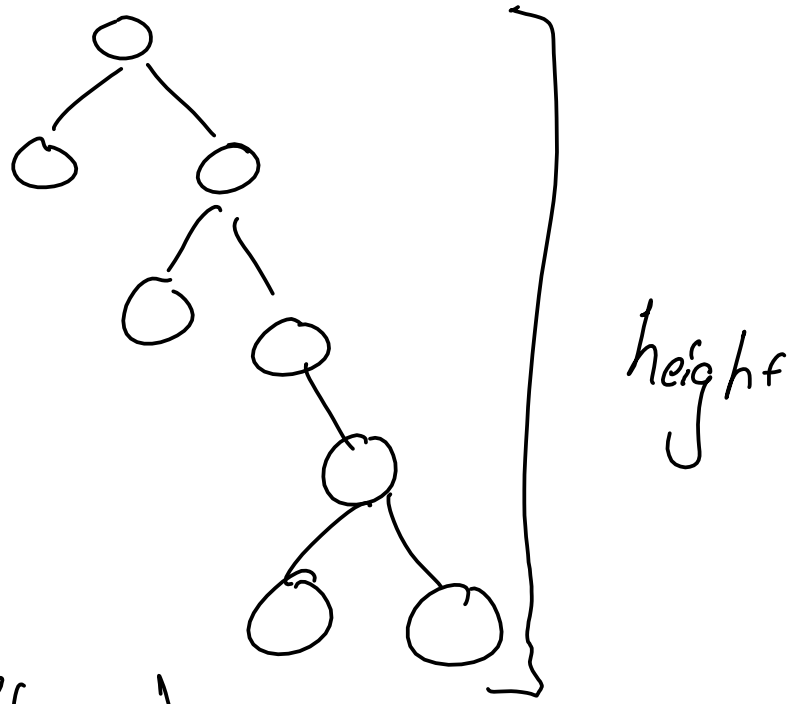
Find node to delete

Find smallest of its right subtree

Steal values from node we found

Remove found node

get : O(height)



We would like if

height is O(log n)