Data Structures and Algorithms

Mohammad GANJTABESH

Department of Computer Science,
School of Mathematics, Statistics and Computer Science,
University of Tehran,
Tehran, Iran.

mgtabesh@ut.ac.ir

Dept. of Computer Science
University of Tehran

AVL Tree

(strictly balanced tree)

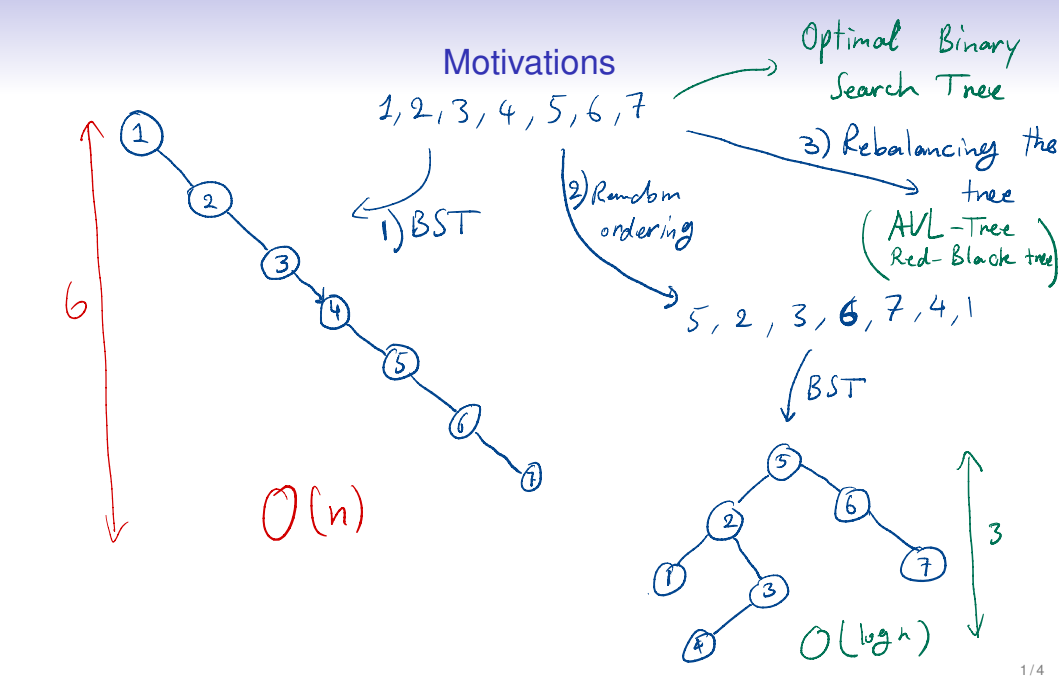
Data Structures and Algorithms

Undergraduate course

Mohammad GANJTABESH
mgtabesh@ut.ac.ir

Dept. of Computer Science
University of Tehran

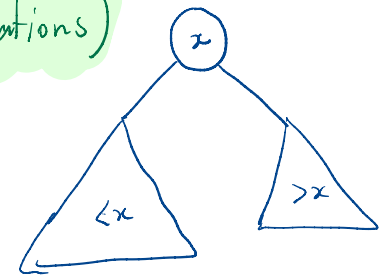
Motivations

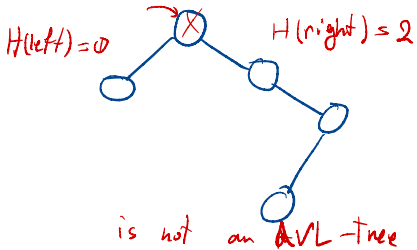


Basic Definitions

AVL-Tree : is a BST in which the difference in height between left and right must always be in $\{-1, 0, +1\}$

(Rotations)

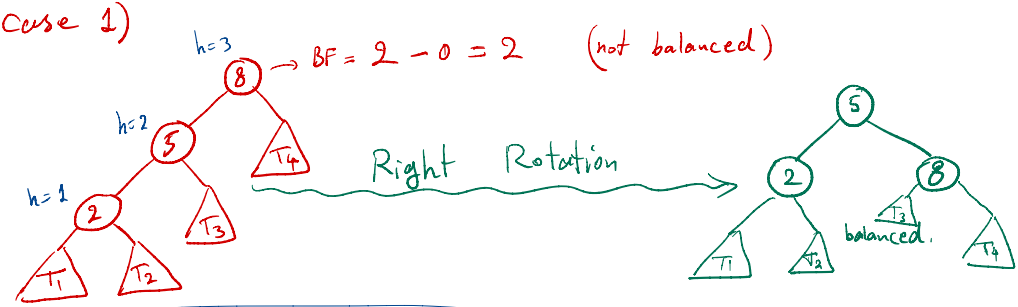




Balance Factor ← BF(node) = Height(node.left) − Height(node.right)

Different Types of Rotations

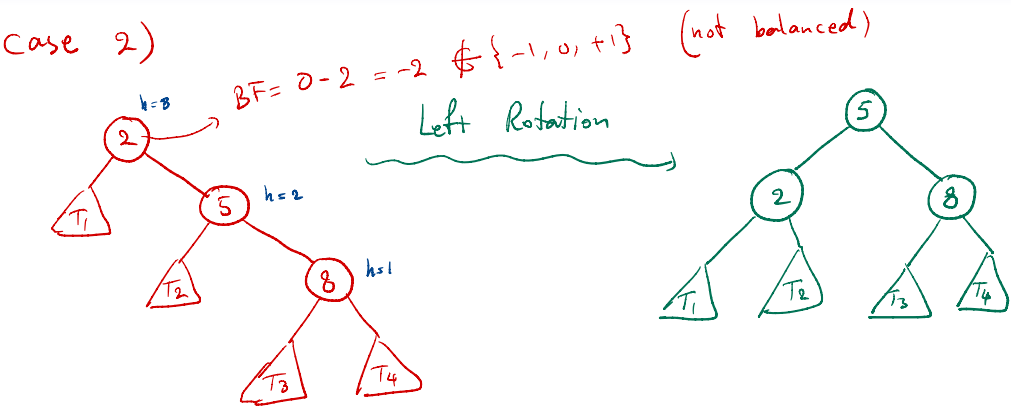
Rotations : try to keep the height of the tree as small as possible (balanced).



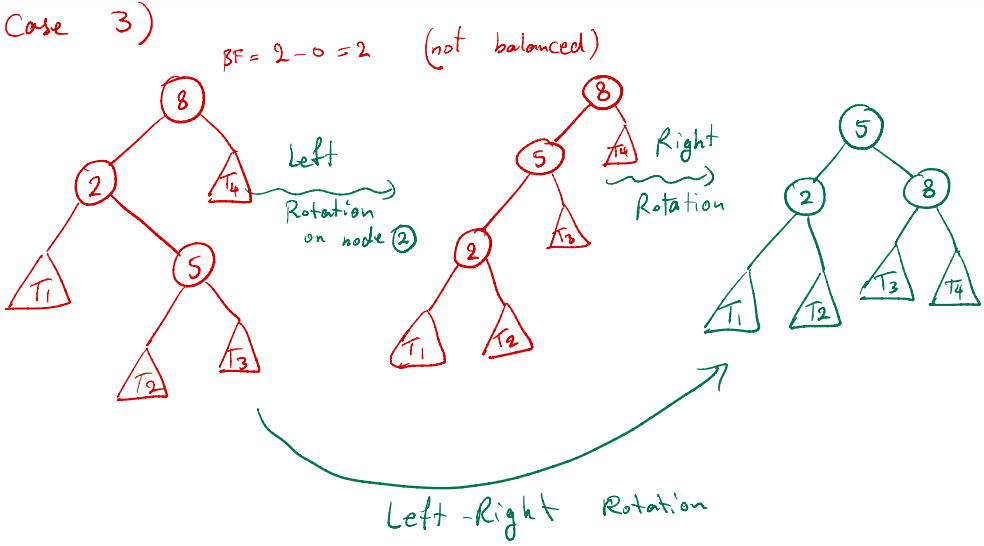
$$H(node) = \begin{cases} 0 & \text{if } node = null \\ \max\{H(node.left), H(node.right)\} + 1 & \text{o.w.} \end{cases}$$

Different Types of Rotations

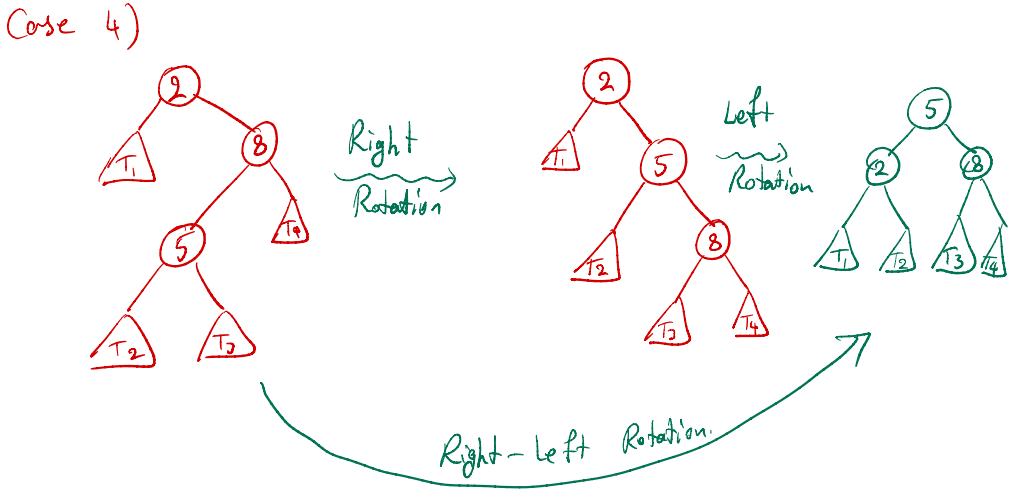
Case 2)



Different Types of Rotations



Different Types of Rotations



AVL Tree Implementation

```
struct Node {
    int key
    Node *left;
    Node *right;
    int height;
}
```


```
Node (int k){
    key ← k;
    left ← right ← null;
    height ← 0
}
```

```
class AVLTree {
    Node *root;

    AVLTree (){
        root ← null;
    }
    // methods
}
```

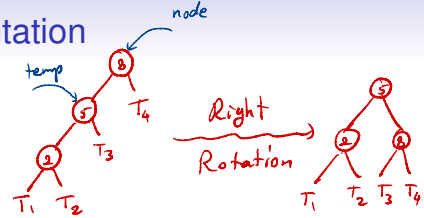
```
int getHeight (Node *node){
    if (node = null)
        return (0);

    return (node.height);
}
```

```
void updateHeight (Node *node){
    if (node = null) return;
    node.height ← max (getHeight(node.left),
                       getHeight(node.right)) + 1;
}
```

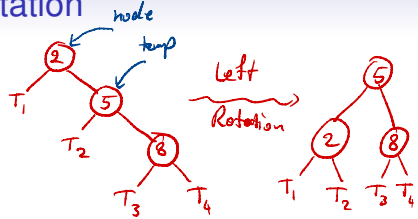
AVL Tree Implementation

```
Node * rotateRight (Node *node) {
    if (node = null) return (node);
    Node *temp ⟵ node.left;
    node.left ⟵ temp.right;
    temp.right ⟵ node;
    updateHeight (node);
    updateHeight (temp);
    return (temp);
}
```



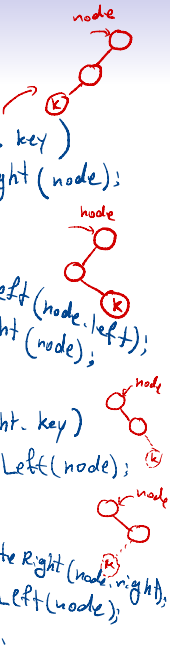
AVL Tree Implementation

```
Node * rotateLeft(Node *node){
    if(node = null) return(node);
    Node *temp ← node.right;
    node.right ← temp.left;
    temp.left ← node;
    updateHeight(node);
    updateHeight(temp);
    return(temp);
}
```



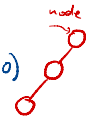
AVL Tree Implementation

```
public insert(int k){
    root ← insert(root, k);
}
private Node* insert(Node *node, int k){
    if(node = null) return(new Node(k));
    if(k < node.key)
        node.left ← insert(node.left, k);
    else if(k > node.key)
        node.right ← insert(node.right, k);
    else
        return(node);
    updateHeight(node);
    b ← getBalance(node);
    if(b > 1){
        if(k < node.left.key)
            node ← rotateRight(node);
        else{
            node.left ← rotateLeft(node.left);
            node ← rotateRight(node);
        }
    }else if(b < -1){
        if(k > node.right.key)
            node ← rotateLeft(node);
        else{
            node.right ← rotateRight(node.right);
            node ← rotateLeft(node);
        }
    }
    return(node);
}
```

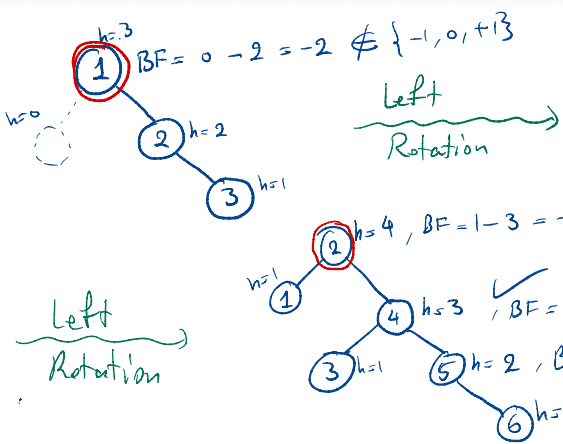


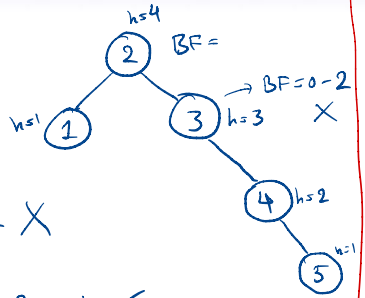
AVL Tree Implementation

```
Public delete (int k){
    root ← delete (root, k);
}
Private Node * delete (Node *node, int k){
    if (node = null) return (null);
    if (k < node.key)
        node.left ← delete (node.left, k);
    else if (k > node.key)
        node.right ← delete (node.right, k);
    else {
        if (node.left = null)
            node ← node.right;
        else if (node.right = null)
            node ← node.left;
        else {
            succ ← FindMin (node.right);
            node.key ← succ;
            node.right ← delete (node.right, succ);
        }
    }
    if (node = null)
        return (null);
    updateHeight (node);
    b ← getBalance (node);
    if (b > 1) {
        if (getBalance (node.left) >= 0)
            node ← rotateRight (node);
        else {
            node.left ← rotateLeft (node.left);
            node ← rotateRight (node);
        }
    }
    else if (b < -1) {
        if (getBalance (node.right) <= 0)
            node ← rotateLeft (node);
        else {
            node.right ← rotateRight (node.right);
            node ← rotateLeft (node);
        }
    }
    return (node);
```

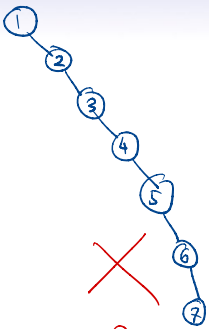


Example : 1, 2, 3, 4, 5, 6, 7, ...

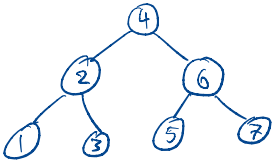


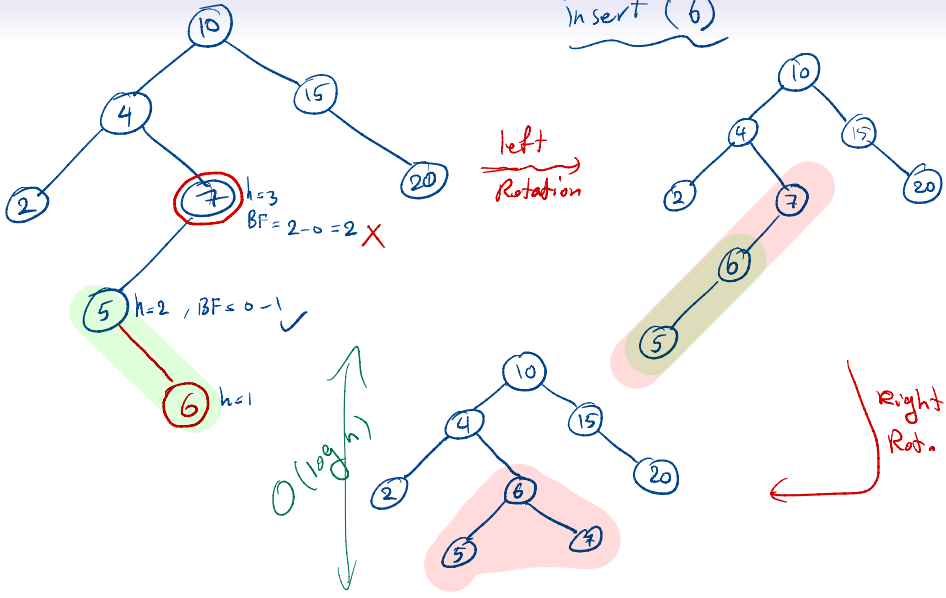


BST



O(h)




insert (6)
10
4
15
2
7
20
h=3
BF = 2-0 = 2
5
h=2 , BF = 0-1
6
h=1
left
Rotation
10
4
15
2
7
20
6
5
O(log n)
10
4
15
2
6
20
5
7
Right
Rot.