COL106: Data Structures and Algorithms

Ragesh Jaiswal, IIT Delhi

Data Structures: Balanced Binary Search Trees

- Consider the following implementation:

Code

```
class Node{
   public int key;
   public String value;
   public Node leftChild;
   public Node rightChild;
   public Node parent;
}
public class BST{
   public int size;
   public Node root;
   public BST(){
      size = 0;root = null;
   }
   public boolean isLeaf(Node N){//To be written}
   public String get(int k){//To be written}
   public void put(int k, String v){//To be written}
   public void remove(int k){//To be written}
}
```

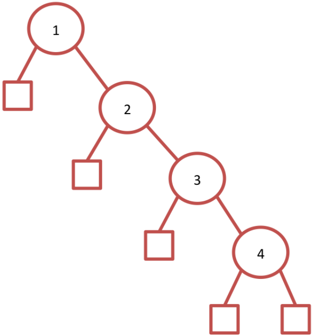
- What is the worst case running time of each of the following operations?
  - `get(k)`:
  - `put(k, v)`:
  - `remove(k)`:

- What is the worst case running time of each of the following operations?
  - `get(k)`: $O(n)$
  - `put(k, v)`: $O(n)$
  - `remove(k)`: $O(n)$

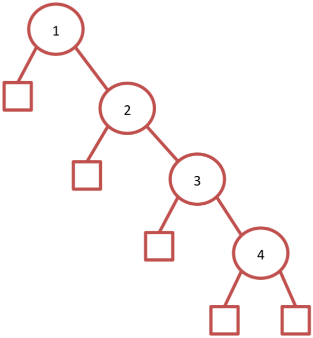
- What is the worst case running time of each of the following operations when the BST is balanced?
  - `get(k)`:
  - `put(k, v)`:
  - `remove(k)`:
- A BST is perfectly balanced if for every internal node, there are equal number of nodes in its left and right sub-trees.

- What is the worst case running time of each of the following operations when the BST is balanced?
  - `get(k)`: $O(\log n)$
  - `put(k, v)`: $O(\log n)$
  - `remove(k)`: $O(\log n)$
- So, our next goal shall be to build balanced binary search trees.

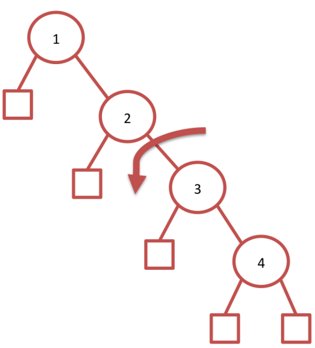
- Suppose we start with an empty BST and insert the keys $1, 2, 3, 4$, then the BST obtained is shown below.



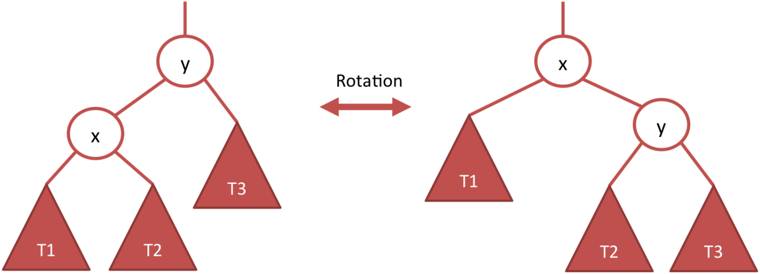
- Suppose we start with an empty BST and insert the keys $1, 2, 3, 4$, then the BST obtained is shown below.
- This tree is not balanced. Can you think of a way to balance this tree?



- Suppose we start with an empty BST and insert the keys $1, 2, 3, 4$, then the BST obtained is shown below.
- This tree is not balanced. Can you think of a way to balance this tree?



- Rotation for tree balancing.



- Tri-node restructuring for a node x, its parent y, and its grandparent z.

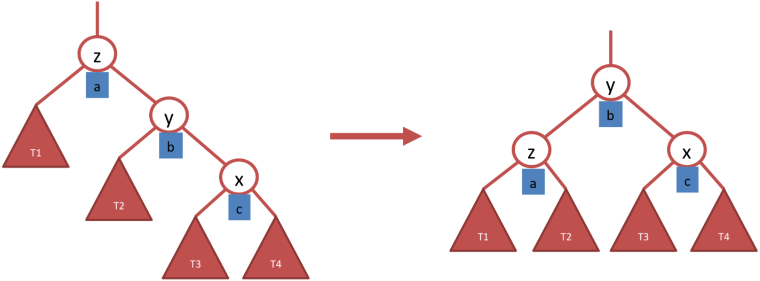


Figure : Case #1

- Tri-node restructuring for a node x, its parent y, and its grandparent z.

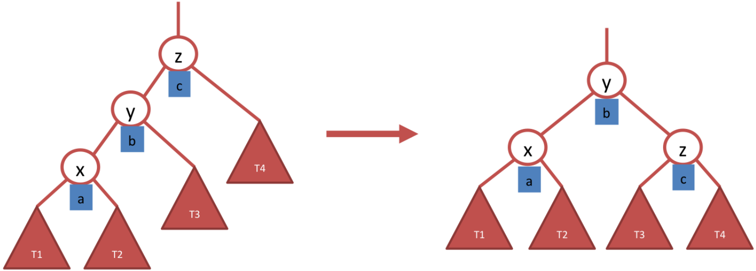


Figure : Case #2

- Tri-node restructuring for a node x, its parent y, and its grandparent z.

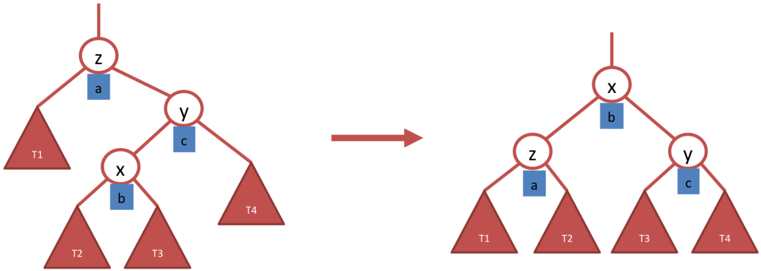


Figure : Case #3

- Tri-node restructuring for a node x, its parent y, and its grandparent z.

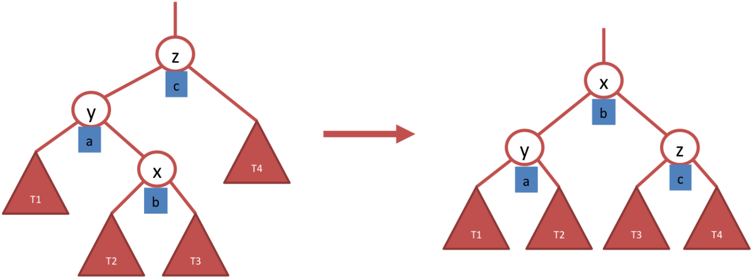
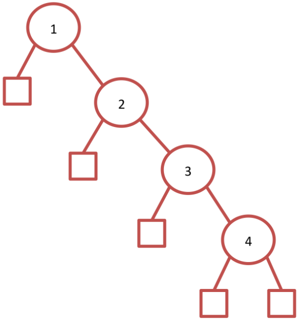


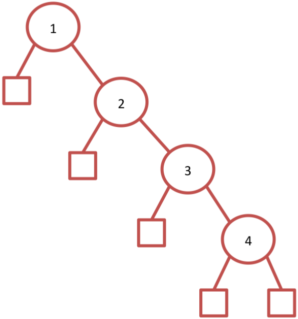
Figure : Case #4

- AVL Tree: An AVL tree is a binary search tree that satisfies the following property:
  Height balance property: For every internal node of the tree, the heights of its children differ by at most 1.

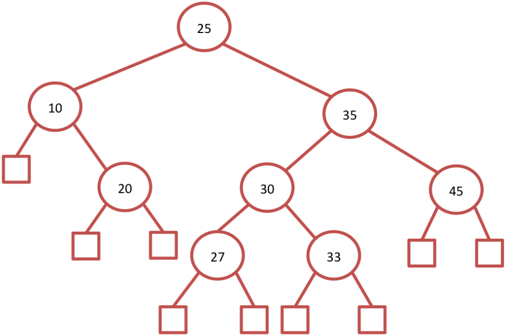
- AVL Tree: An AVL tree is a binary search tree that satisfies the following property:
  Height balance property: For every internal node of the tree, the heights of its children differ by at most 1.
- Is the binary search tree below an AVL tree?



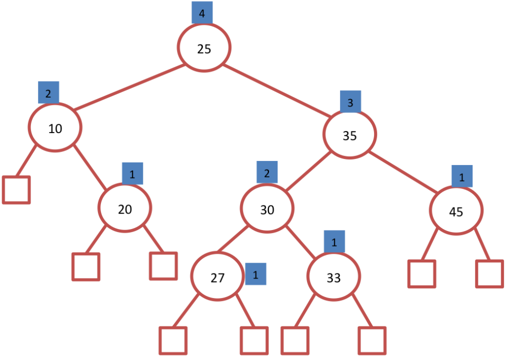
- AVL Tree: An AVL tree is a binary search tree that satisfies the following property:
  Height balance property: For every internal node of the tree, the heights of its children differ by at most 1.
- Is the binary search tree below an AVL tree? No



- AVL Tree: An AVL tree is a binary search tree that satisfies the following property:
  Height balance property: For every internal node of the tree, the heights of its children differ by at most 1.
- Is the binary search tree below an AVL tree?



- AVL Tree: An AVL tree is a binary search tree that satisfies the following property:
  Height balance property: For every internal node of the tree, the heights of its children differ by at most 1.
- Is the binary search tree below an AVL tree? Yes



- AVL Tree: An AVL tree is a binary search tree that satisfies the following property:
  Height balance property: For every internal node of the tree, the heights of its children differ by at most 1.
- Claim: The height of any AVL tree storing n nodes is $O(\log n)$.

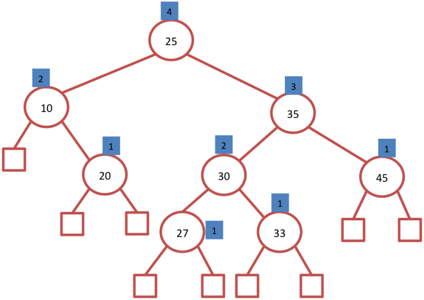
- AVL Tree: An AVL tree is a binary search tree that satisfies the following property:
  Height balance property: For every internal node of the tree, the heights of its children differ by at most 1.
- Claim: The height of any AVL tree storing n nodes is $O(\log n)$.
  - Let $n(h)$ denote the minimum number of internal nodes in an AVL tree with height h.
  - Try writing a recurrence relation for $n(h)$ and solving it to get a lower bound.

- AVL Tree: An AVL tree is a binary search tree that satisfies the following property:
  Height balance property: For every internal node of the tree, the heights of its children differ by at most 1.
- Question: How do we perform `get(k)` operation on an AVL tree?

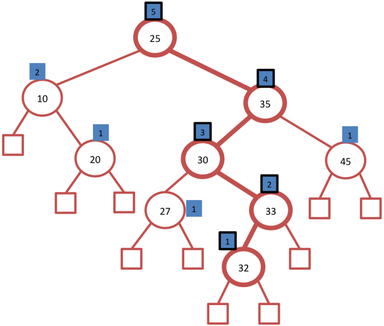
- AVL Tree: An AVL tree is a binary search tree that satisfies the following property:
  Height balance property: For every internal node of the tree, the heights of its children differ by at most 1.
- Question: How do we perform `get(k)` operation on an AVL tree?
  The same as BST

- AVL Tree: An AVL tree is a binary search tree that satisfies the following property:
  Height balance property: For every internal node of the tree, the heights of its children differ by at most 1.
- Question: How do we perform `put(k, v)` operation on an AVL tree?

- AVL Tree: An AVL tree is a binary search tree that satisfies the following property:
  Height balance property: For every internal node of the tree, the heights of its children differ by at most 1.
- Question: How do we perform `put(k, v)` operation on an AVL tree?
  - Same as in BST. However, you also have to make sure that after insertion, the height balance property is maintained.
  - Consider inserting an entry with key 32 in the Tree below.



- AVL Tree: An AVL tree is a binary search tree that satisfies the following property:
  Height balance property: For every internal node of the tree, the heights of its children differ by at most 1.
- Question: How do we perform `put(k, v)` operation on an AVL tree?
  - Same as in BST. However, you also have to make sure that after insertion, the height balance property is maintained.
  - Consider inserting an entry with key 32 in the Tree below.



- Question: How do we perform `put(k, v)` operation on an AVL tree?

Algorithm

//p denotes the node that is inserted.

`BalanceAfterPut(Node` p`)`

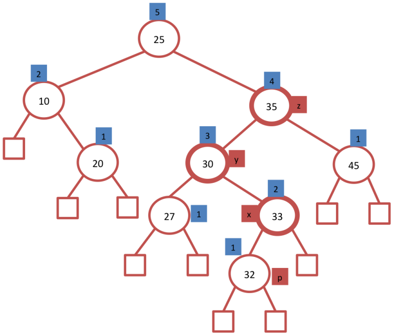
- While going up from p, let z denote the first node for which the height balance property is not satisfied.
- Let y be the child of z with greater height.
- Let x be the child of y with greater height.
- Perform a tri-node restructuring w.r.t. nodes x, y, z.

Algorithm

//p denotes the node that is inserted.

`BalanceAfterPut(Node` p`)`

- While going up from p, let z denote the first node for which the height balance property is not satisfied.
- Let y be the child of z with greater height.
- Let x be the child of y with greater height.
- Perform a tri-node restructuring w.r.t. nodes x, y, z.

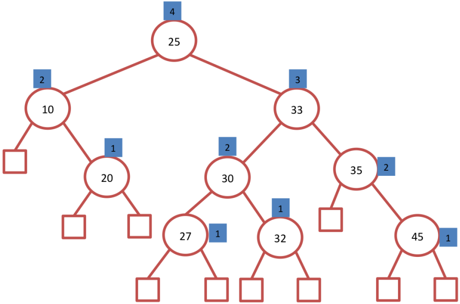


Algorithm

//p denotes the node that is inserted.

`BalanceAfterPut(Node` p`)`

- While going up from p, let z denote the first node for which the height balance property is not satisfied.
- Let y be the child of z with greater height.
- Let x be the child of y with greater height.
- Perform a tri-node restructuring w.r.t. nodes x, y, z.



End