COEN 352 – SUMMER 2023

AVL Tree

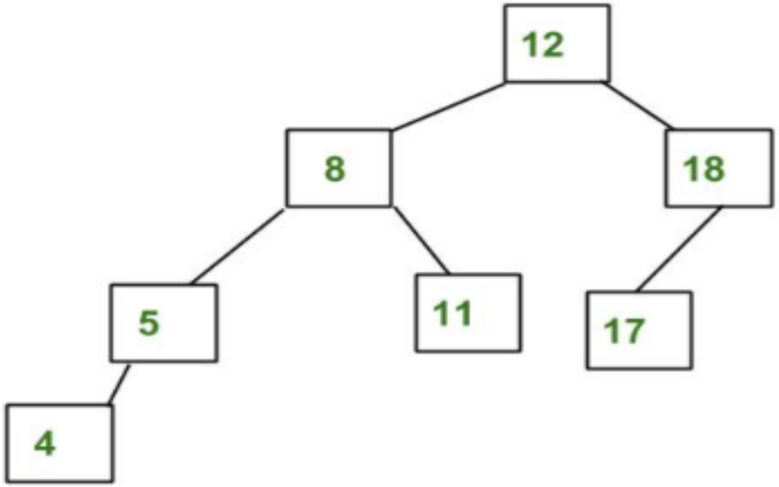
OUTLINE

- AVL tree
  - Properties
  - Rotation
- Exercise

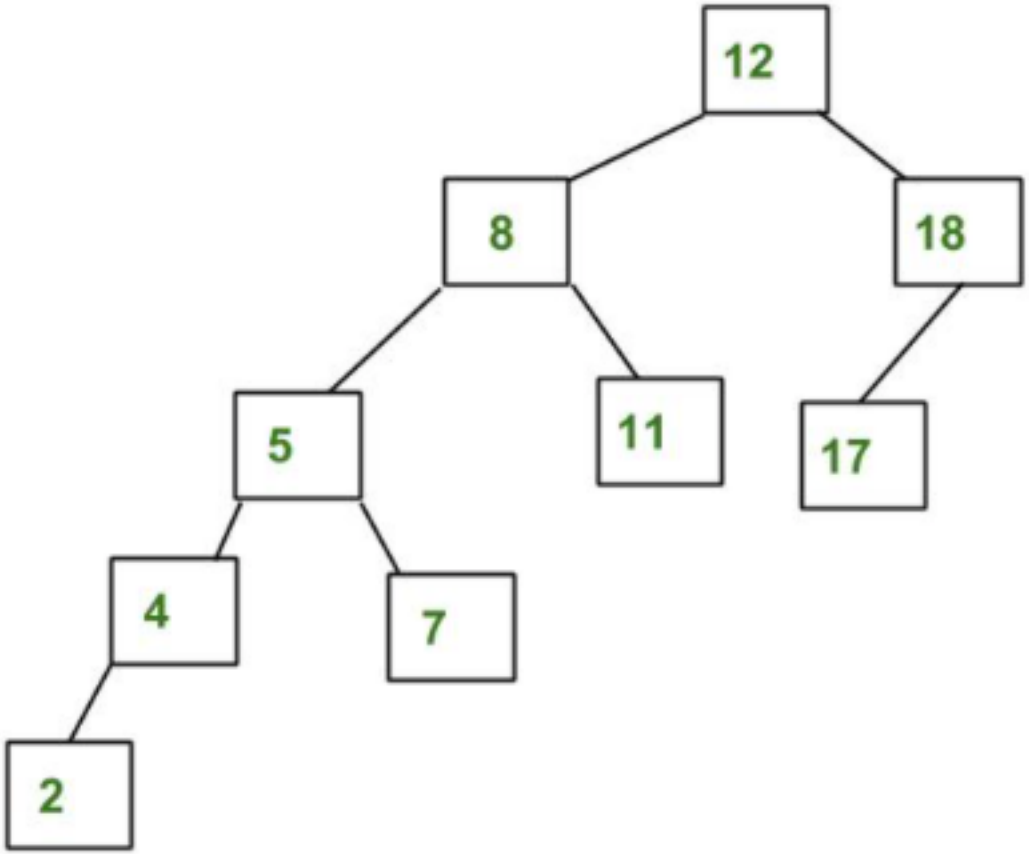
AVL TREE OVERVIEW(1)

- BST tree : have the same properties
- An extra property: Is a self-balanced BST→ the difference between the heights of the left and right subtrees is -1,0 or 1

AVL Tree : example



A BAD AVL TREE



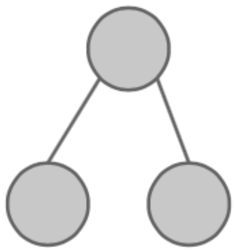
AVL TREE : OVERVIEW

- Balance factor of a node (BF) x= heigth(lchid)-heigth(rchid)
  - BF(x) = 0: **x** is **balanced**
  - BF(x) = 1: **x** is **right-heavy**
  - BF(x) = -1: **x** is **left-heavy**

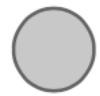
  Above case are good case

  BF(x) > 1 or < -1: **x** is **imbalanced (not good)**

Special case of height



h = 1



h = 0

NIL

h = -1

Note: height is measured by the number of edges.

Operations on AVL tree:

- Insertion(root,k)
- Search(root,k)
- Delete(root,k)

Two of these operation can make the new tree violate the AVL tree property: insertion and deletion

Université Concordia University

Step for insertion

Let the newly inserted node be w:

1. Perform standard BST insert for w.
2. Starting from w, travel up and find the first unbalanced node.
3. Let z be the first unbalanced node, y be the child of z that comes on the path from w to z and x be the grandchild of z that comes on the path from w to z.

INSERTION(CONT'D)

3. Re-balance the tree by performing appropriate rotations on the subtree rooted with z. There can be 4 possible cases that needs to be handled as x, y and z can be arranged in 4 ways. Following are the possible 4 arrangements:
   a. y is left child of z and x is left child of y (Left Left Case)
   b. y is left child of z and x is right child of y (Left Right Case)
   c. y is right child of z and x is right child of y (Right Right Case)
   d. y is right child of z and x is left child of y (Right Left Case)

c) Right Right Case

```
  z                                y
 /  \                            /   \
T1   y     Left Rotate(z)       z      x
    /  \   - - - - - - - ->    / \    / \
   T2   x                     T1  T2 T3  T4
       / \
      T3  T4
```

d) Right Left Case

```
     z                            z                            x
    / \                          / \                          /  \ 
  T1   y   Right Rotate (y)    T1   x      Left Rotate(z)   z      y
      / \  - - - - - - - - ->     /  \   - - - - - - - ->  / \    / \
     x   T4                      T2   y                  T1  T2  T3  T4
    / \                              /  \
  T2   T3                           T3   T4
```

a) Left Left Case

```
T1, T2, T3 and T4 are subtrees.
         z                                      y
        / \                                   /   \
       y   T4      Right Rotate (z)          x      z
      / \          - - - - - - - - ->       / \    / \
     x   T3                                T1  T2 T3  T4
    / \
  T1   T2
```

b) Left Right Case

```
     z                               z                             x
    / \                            /   \                          /  \
   y   T4  Left Rotate (y)        x    T4  Right Rotate(z)      y      z
  / \      - - - - - - - - ->    /  \      - - - - - - - ->    / \    / \
T1   x                          y    T3                      T1  T2 T3  T4
    / \                        / \
  T2   T3                    T1   T2
```

DELETE

Let w be the node to be deleted

1. Perform standard BST delete for w.
2. Starting from w, travel up and find the first unbalanced node. Let z be the first unbalanced node, y be the larger height child of z, and x be the larger height child of y.

The definitions of x and y are different from insertion here.

Delete (cont'd)

3. Re-balance the tree by performing appropriate rotations on the subtree rooted with z. There can be 4 possible cases that needs to be handled as x, y and z can be arranged in 4 ways. Following are the possible 4 arrangements:
   a) y is left child of z and x is left child of y (Left Left Case)
   b) y is left child of z and x is right child of y (Left Right Case)
   c) y is right child of z and x is right child of y (Right Right Case)
   d) y is right child of z and x is left child of y (Right Left Case)

Exercise

- Give the final AVL tree after inserting the following keys: 30,40,24,58,48,26,11,13
- Extend your BST class tree to implement AVL tree (make AVLTree subclass of BST). Add a subclass AVLNode to class Node and add an extra fields to store the height of the subtree rooted at this node. Implement the following methods in AVL:
  - private AVLNode rotateRight(AVLNode t)
  - private AVLNode rotateLeft(AVLNode t)
  - private AVLNode rotateRightLeft(AVLNode t)

Université Concordia University

Exercise

- private AVLNode rotateLeftRight(AVLNode t)
- public int height()
- private int balancefactor(AVLNode n)
- public int find(int data)
- private int find(int data, AVLNode n)
- public void delete (int data)
- private AVLNode delete(int data, AVLNode n)
- private AVLNode findMinvalueNode(AVLNode n)
- public void preOrder()