DHA SUFFA UNIVERSITY
Department of Computer Science
CS-2001L
Data Structures & Algorithms Lab
Fall 2024
LAB 09

Objective

Learning about implementation of following concepts of Binary Search Tree.

- Search
- Insertion
- Deletion

Theory

Binary Search Tree

Binary Search Tree is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree.
  There must be no duplicate nodes.

The above properties of Binary Search Tree provide an ordering among keys so that the operations like search, minimum and maximum can be done fast. If there is no ordering, then we may have to compare every key to search a given key.

Creating a BST:

```
class Node
{
    int key;
    Node left, right;

    public Node(int item)
    {
        key = item;
        left = right = null;
    }
}
class BST
{
    Node root;            // Root of Binary Tree

    BST(int key)  // Constructors
    {
        root = new Node(key);
    }

    BST()
    {
        root = null;
    }

     void printPostorder(Node node)
    {
        if (node == null)
            return;

        // first recur on left subtree
        printPostorder(node.left);

        // then recur on right subtree
        printPostorder(node.right);

        // now deal with the node
         System.out.print(node.key + " ");
    }
```

```java
/* Given a binary tree, print its nodes in inorder*/
void printInorder(Node node)
{
    if (node == null)
        return;

    /* first recur on left child */
    printInorder(node.left);

    /* then print the data of node */
    System.out.print(node.key + " ");

    /* now recur on right child */
    printInorder(node.right);
}
```

```java
/* Given a binary tree, print its nodes in preorder*/
void printPreorder(Node node)
{
    if (node == null)
        return;

    /* first print data of node */
    System.out.print(node.key + " ");

    /* then recur on left sutree */
    printPreorder(node.left);

    /* now recur on right subtree */
    printPreorder(node.right);
}
```

Main Method:

```
public static void main(String[] args)
{
    BST tree = new BST();

    tree.root = new Node(8); //create root

    Node a = new Node(3);
    tree.root.left = a;
    Node b = new Node(10);
    tree.root.right = b;

    a.left = new Node(1);
    a.right = new Node(6);

    b.left=new Node(9);
    b.right=new Node(14);

    System.out.println("Pre-Order");
    tree.printPreorder(tree.root);
    System.out.println("\nPost-Order");
    tree.printPostorder(tree.root);
    System.out.println("\nIn-Order");
    tree.printInorder(tree.root);
}
```

Searching a key

To search a given key in Binary Search Tree, we first compare it with root, if the key is present at root, we return root. If key is greater than root's key, we recur for right subtree of root node. Otherwise we recur for left subtree.

1. Start from root.
2. Compare the inserting element with root, if less than root, then recurse for left, else recurse for right.
3. If element to search is found anywhere, return true, else return false.

```
    // A utility function to search a given key in BST
public Node search(Node root, int key)
{
    // Base Cases: root is null or key is present at root
    if (root==null || root.key==key)
        return root;

    // val is greater than root's key
    if (root.key > key)
        return search(root.left, key);

    // val is less than root's key
    return search(root.right, key);
}
```

Insertion of a key

A new key is always inserted at leaf. We start searching a key from root till we hit a leaf node. Once a leaf node is found, the new node is added as a child of the leaf node.

```
Node insertRec(Node root, int key) {

    /* If the tree is empty, return a new node */
    if (root == null) {
        root = new Node(key);
        return root;
    }
    else{
    /* Otherwise, recur down the tree */
    if (key < root.key)
        root.left = insertRec(root.left, key);
    else if (key > root.key)
        root.right = insertRec(root.right, key);
    }
    /* return the (unchanged) node pointer */
    return root;
}
```

Deletion:

We have discussed BST search and insert operations. Further, delete operation is discussed. When we delete a node, there possibilities arise.

1) ***Node to be deleted is leaf****:* Simply remove from the tree.
2) ***Node to be deleted has only one child****:* Copy the child to the node and delete the child

3) ***Node to be deleted has two children****:* Find inorder successor of the node. Copy contents of the inorder successor to the node and delete the inorder successor. Note that inorder predecessor can also be used.

```
/* A recursive function to insert a new key in BST */
Node deleteRec(Node root, int key)
{
        /* Base Case: If the tree is empty */
    if (root == null)  return root;

        /* Otherwise, recur down the tree */
    if (key < root.key)
        root.left = deleteRec(root.left, key);
    else if (key > root.key)
        root.right = deleteRec(root.right, key);

        // if key is same as root's key, then This is the node
        // to be deleted
    else
    {
        // node with only one child or no child
        if (root.left == null)
            return root.right;
        else if (root.right == null)
            return root.left;

        // node with two children: Get the inorder successor (smallest
        // in the right subtree)
        root.key = minValue(root.right);

        // Delete the inorder successor
        root.right = deleteRec(root.right, root.key);
    }
    return root;
}
```

```
int minValue(Node root)
{
    int minv = root.key;
    while (root.left != null)
    {
        minv = root.left.key;
        root = root.left;
    }
    return minv;
}
```

LAB TASKS

1) Design a binary expression tree representation of the following arithmetic expression.

((5+2)*(2-1))/(2+9)

Create a function for the traversal of the expression tree.

2) Convert a given sorted array into Binary Search Tree.

3) Convert a Binary tree into Binary Search Tree.

Submission Guidelines

- Write your codes in different notepad (.java) files.
- Place all files in a folder named your roll number e.g. cs162001.
- Compress all files into one (.zip) file named same as your roll number.
- Upload it on LMS
- -**100** policies for plagiarism