Institute Of Technology, Nirma university

BRANCH :- **Computer Science Engineering**

PRACTICAL SUBMISSION

|*|*STUDENT INFO*|*|

Name :- **Pratik Kansara**

Roll No. :- **20BCE510**

Division :- **E4**

|*|*SUBJECT INFO*|*|

Subject :- **Advanced Data Structures**

Practical No. :- **3**

Practical – 3

AIM :- Rebalancing operation can be delayed until a certain threshold is attained. Scapegoat tree uses partial rebuilding for balancing a search tree. Implement scapegoat tree to demonstrate the partial rebuilding operation.

Code:

SCGNode.java

```java
package Practical3;


public class SCGNode {

    protected SCGNode right, left, parent;

    protected int obj, key;


    public SCGNode(int key, int obj) {

        right = left = parent = null;

        this.key = key;

        this.obj = obj;

    }

}
```

ScapeGoatTree.java

```
package Practical3;


public class ScapeGoatTree {

    private SCGNode root;

    // Size of the tree

    int n;

    // overestimate of n

    int q;


    public ScapeGoatTree() {

        root = null;

        n = 0;

    }


    public boolean IsEmpty() {

        if (root == null) {

            return true;

        }

        return false;

    }


    public int GetSize(SCGNode Rt) {

        if (Rt == null) {

            return 0;

        } else {

            return (GetSize(Rt.left) + 1 + GetSize(Rt.right));
```

```
    }
}

public int SearchKey(int Key) {
    return SearchKey(root, Key);
}

private int SearchKey(SCGNode Rt, int Key) {
    if (Rt == null || Rt.key == Key) {
        return Rt.key;
    } else if (Rt.key > Key) {
        return SearchKey(Rt.left, Key);
    } else {
        return SearchKey(Rt.right, Key);
    }
}

private static final int Log32(int Val) {
    final double lg32 = 2.4663034623764317;
    return (int) Math.ceil(lg32 * Math.log(Val));
}

public int InsertWithDepth(SCGNode N) {
    SCGNode Rt = root;
    if (Rt == null) {
        root = N;
        n++;
        q++;
```

```
            return 0;
        }
        boolean inserted = false;
        int depth = 0;
        do {
            if (N.key < Rt.key) {
                if (Rt.left == null) {
                    Rt.left = N;
                    N.parent = Rt;
                    inserted = true;
                } else {
                    Rt = Rt.left;
                }
            } else if (N.key > Rt.key) {
                if (Rt.right == null) {
                    Rt.right = N;
                    N.parent = Rt;
                    inserted = true;
                } else {
                    Rt = Rt.right;
                }
            } else {
                return -1;
            }
            depth++;
        } while (!inserted);
        n++;
        q++;
```

```
        return depth;
    }


    public void Insert(int Key, int obj) {
        SCGNode n = new SCGNode(Key, obj);
        int depth = InsertWithDepth(n);
        if (depth > Log32(q)) // n can be used as well
        {
            // Depth exceeded, find scapegoat
            SCGNode temp = n.parent;
            while (3 * GetSize(temp) <= 2 * GetSize(temp.parent)) {
                temp = temp.parent;
                Rebuild(temp.parent);
            }
        }
    }


    public void Rebuild(SCGNode N) {
        int NodeSize = GetSize(N);
        SCGNode Parent = N.parent;
        SCGNode[] Arr = new SCGNode[NodeSize];
        PackIntoArr(N, Arr, 0);
        if (Parent == null) {
            root = BuildBalanced(Arr, 0, NodeSize);
            root.parent = null;
        } else if (Parent.right == N) {
            Parent.right = BuildBalanced(Arr, 0, NodeSize);
            // Parent.Right.Parent=Parent;
```

```
            System.out.println(Parent.right.key);
        } else {
            Parent.left = BuildBalanced(Arr, 0, NodeSize);
            // Parent.Left.Parent=Parent;
        }
    }

    public int PackIntoArr(SCGNode N, SCGNode[] Arr, int i) {
        if (N == null) {
            return i;
        }
        i = PackIntoArr(N.left, Arr, i);
        Arr[i++] = N;

        return PackIntoArr(N.right, Arr, i);
    }

    public SCGNode BuildBalanced(SCGNode[] Arr, int i, int NodeSize) {
        if (NodeSize == 0) {
            return null;
        }
        int m = NodeSize / 2;
        Arr[i + m].left = BuildBalanced(Arr, i, m);
        if (Arr[i + m].left != null) {
            Arr[i + m].left.parent = Arr[i + m];
        }
        Arr[i + m].right = BuildBalanced(Arr, i + m + 1, NodeSize - m - 1);
        if (Arr[i + m].right != null) {
```

```
            Arr[i + m].right.parent = Arr[i + m];
        }
        return Arr[i + m];
    }

    public void Delete(int Key) {
        root = Delete(root, Key);
    }

    private SCGNode Delete(SCGNode Rt, int Key) {
        if (Rt == null) {
            return Rt;
        }
        if (Key < Rt.key) {
            Rt.left = Delete(Rt.left, Key);
        } else if (Key > Rt.key) {
            Rt.right = Delete(Rt.right, Key);
        } else {
            if (Rt.left == null) {
                return Rt.right;
            } else if (Rt.right == null) {
                return Rt.left;
            }
            Rt.key = MinVal(Rt.right);
            Rt.right = Delete(Rt.right, Rt.key);
        }
        return Rt;
    }
```

```
public int MinVal(SCGNode Rt) {
    int Minimum = Rt.key;
    while (Rt.left != null) {
        Minimum = Rt.left.key;
        Rt = Rt.left;
    }
    return Minimum;
}

public void InOrder() {
    System.out.println("Inorder Traversal of Scapegoattree is : ");
    InOrder(root);
}

private void InOrder(SCGNode Rt) {
    if (Rt != null) {
        String str = "";

        if (Rt.left == null) {
            str += ".";
        } else {
            str += Rt.left.key;
        }

        str += " <- " + Rt.key + " --> " + Rt.obj + " -> ";

        if (Rt.right == null) {
```

```
                str += ".";
            } else {
                str += Rt.right.key;
            }

            InOrder(Rt.left);
            System.out.println(str);
            InOrder(Rt.right);
        }

    }

    public void PreOrder() {
        System.out.println("Preorder Traversal of Scapegoattree is : ");
        PreOrder(root);
    }

    private void PreOrder(SCGNode Rt) {

        if (Rt != null) {
            System.out.println(Rt.key + " --> " + Rt.obj);
            PreOrder(Rt.left);
            PreOrder(Rt.right);
        }
    }
}
```

ScapeGoatMain.java

```
package Practical3;


import java.util.Scanner;


public class ScapeGoatMain {

    public static void main(String[] args) {

        ScapeGoatTree sgt = new ScapeGoatTree();

        Scanner sc = new Scanner(System.in);

        while (true) {

            System.out.println("1. For Insert");

            System.out.println("2. For Delete");

            System.out.println("3. For Display");

            System.out.println("4. For Exit");

            int ch = sc.nextInt();


            switch (ch) {

                case 1:

                    int key, val;

                    System.out.println("Enter Key : ");

                    key = sc.nextInt();

                    System.out.println("Enter Value : ");

                    val = sc.nextInt();

                    sgt.Insert(key, val);

                    break;

                case 2:

                    int k;
```

```
                    System.out.println("Enter Key to Delete : ");
                    k = sc.nextInt();
                    sgt.Delete(k);
                    System.out.println(k + " is deleted successfully");
                    break;
                case 3:
                    System.out.println("1. For inorder traversal");
                    System.out.println("2. For preorder traversal");
                    int c = sc.nextInt();
                    if (c == 1) {
                        sgt.InOrder();
                    } else if (c == 2){
                        sgt.PreOrder();
                    } else {
                        System.out.println("Enter Valid Choice!!");
                    }
                    break;
                case 4:
                    System.exit(0);
                default:
                    System.out.println("Enter Valid Choice!!");
            }
        }


    }
}
```

OUTPUT

Insert

```
1. For Insert
2. For Delete
3. For Display
4. For Exit
1
Enter Key :
7
Enter Value :
0
1. For Insert
2. For Delete
3. For Display
4. For Exit
1
Enter Key :
6
Enter Value :
0
1. For Insert
2. For Delete
3. For Display
4. For Exit
1
Enter Key :
3
Enter Value :
0
1. For Insert
2. For Delete
3. For Display
4. For Exit
1
Enter Key :
1
Enter Value :
0
```

```
1. For Insert
2. For Delete
3. For Display
4. For Exit
1
Enter Key :
0
Enter Value :
0
1. For Insert
2. For Delete
3. For Display
4. For Exit
1
Enter Key :
8
Enter Value :
0
1. For Insert
2. For Delete
3. For Display
4. For Exit
1
Enter Key :
9
Enter Value :
0
1. For Insert
2. For Delete
3. For Display
4. For Exit
1
Enter Key :
4
Enter Value :
0
```

```
1. For Insert
2. For Delete
3. For Display
4. For Exit
1
Enter Key :
5
Enter Value :
0
1. For Insert
2. For Delete
3. For Display
4. For Exit
1
Enter Key :
2
Enter Value :
0
1. For Insert
2. For Delete
3. For Display
4. For Exit
1
Enter Key :
10
Enter Value :
0
```

Display

```
1. For Insert
2. For Delete
3. For Display
4. For Exit
3
1. For inorder traversal
2. For preorder traversal
1
Inorder Traversal of Scapegoattree is :
. <- 0 --> 0 -> .
0 <- 1 --> 0 -> 2
. <- 2 --> 0 -> .
1 <- 3 --> 0 -> 4
. <- 4 --> 0 -> 5
. <- 5 --> 0 -> .
3 <- 6 --> 0 -> .
6 <- 7 --> 0 -> 8
. <- 8 --> 0 -> 9
. <- 9 --> 0 -> 10
. <- 10 --> 0 -> .
```

```
1. For Insert
2. For Delete
3. For Display
4. For Exit
3
1. For inorder traversal
2. For preorder traversal
2
Preorder Traversal of Scapegoattree is :
7 --> 0
6 --> 0
3 --> 0
1 --> 0
0 --> 0
2 --> 0
4 --> 0
5 --> 0
8 --> 0
9 --> 0
10 --> 0
```

Delete and Display

```
1. For Insert
2. For Delete
3. For Display
4. For Exit
2
Enter Key to Delete :
10
10 is deleted successfully
1. For Insert
2. For Delete
3. For Display
4. For Exit
3
1. For inorder traversal
2. For preorder traversal
1
Inorder Traversal of Scapegoattree is :
. <- 0 --> 0 -> .
0 <- 1 --> 0 -> 2
. <- 2 --> 0 -> .
1 <- 3 --> 0 -> 4
. <- 4 --> 0 -> 5
. <- 5 --> 0 -> .
3 <- 6 --> 0 -> .
6 <- 7 --> 0 -> 8
. <- 8 --> 0 -> 9
. <- 9 --> 0 -> .
```

```
1. For Insert
2. For Delete
3. For Display
4. For Exit
2
Enter Key to Delete :
9
9 is deleted successfully
1. For Insert
2. For Delete
3. For Display
4. For Exit
3
1. For inorder traversal
2. For preorder traversal
2
Preorder Traversal of Scapegoattree is :
7 --> 0
6 --> 0
3 --> 0
1 --> 0
0 --> 0
2 --> 0
4 --> 0
5 --> 0
8 --> 0
```