**COURSE CODE:** DJS22ITL502 **DATE: 7-10-24**

**COURSE NAME:** Advanced Data Structures Laboratory **CLASS:** TY B. TECH

**NAME:** Anish Sharma **DIV:** IT 1 **ROLL:** I011

# EXPERIME NT NO. 5

**CO/LO:** Choose appropriate data structure and use it to design algorithm for solving a specific problem

**AIM / OBJECTIVE:** To implement various operations on Leftist Heap.

**Properties of Leftist Heap:**

1. Skewed Structure: A leftist heap is biased to the left, with a path-heavy left child to minimize merging costs.
2. Heap Property: The heap maintains the min-heap property.
3. Merging: Merging two heaps takes O(logn), as the right paths are as short as possible.
4. Shortest Path Property: The rank (null path length) of the right child is always less than or equal to the left child.
5. Efficient Insert/Delete: Insert and delete-min both take O(logn) time through merging.

**TECHNOLOGY STACK USED: C, C++, JAVA SOURCE CODE:**

import java.util.\*;

class LeftistNode {

    int element, dist;

    LeftistNode left, right;

    public LeftistNode(int element) {

        this(element, null, null);

    }

    public LeftistNode(int element, LeftistNode left, LeftistNode right) {

        this.element = element;

        this.left = left;

        this.right = right;

        this.dist = 0;

    }

}

class LeftistHeap {

    private LeftistNode root;

    public LeftistHeap() {

        root = null;

    }

    public boolean isEmpty() {

        return root == null;

    }

    public void makeEmpty() {

        root = null;

    }

    public void insert(int x) {

        root = merge(new LeftistNode(x), root);

    }

    public int deleteMin() {

        if (isEmpty())

            throw new NoSuchElementException();

        int minItem = root.element;

        root = merge(root.left, root.right);

        return minItem;

    }

    private LeftistNode merge(LeftistNode x, LeftistNode y) {

        if (x == null)

            return y;

        if (y == null)

            return x;

        if (x.element > y.element) {

            LeftistNode temp = x;

            x = y;

            y = temp;

        }

        x.right = merge(x.right, y);

        if (x.left == null) {

            x.left = x.right;

            x.right = null;

        } else {

            if (x.left.dist < x.right.dist) {

                LeftistNode temp = x.left;

                x.left = x.right;

                x.right = temp;

            }

            x.dist = x.right.dist + 1;

        }

        return x;

    }

    public void merge(LeftistHeap rhs) {

        if (this == rhs)

            return;

        root = merge(root, rhs.root);

        rhs.root = null;

    }

}

public class Main {

    public static void main(String[] args) {

        int[] arr = {1, 5, 7, 10, 15};

        int[] arr1 = {22, 75};

        LeftistHeap h = new LeftistHeap();

        LeftistHeap h1 = new LeftistHeap();

        LeftistHeap h2;

        for (int i : arr)

            h.insert(i);

        for (int i : arr1)

            h1.insert(i);

        System.out.println(h.deleteMin());

        System.out.println(h1.deleteMin());

        h.merge(h1);

        h2 = h;

        System.out.println(h2.deleteMin());

    }

}

**OUTPUT:**

****

# CONCLUSION: In this experiment we understood and implemented Leftist heaps

# REFERENCES:

1. Peter Brass, “Advanced Data Structures”, Cambridge University Press, 2008
2. Robert Sedgewick & Kevin Wayne, “Algorithms”, 4th Edition, Addison-Wesley Professional, 2011.