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. :- **4**

Practical – 4

AIM:- Skip list structures are used to retrieve the data faster. Implement the structure up to the third level. Show the effect of insert and delete operation

Code:

Node.java

```java
public class Node {
    public Node above;
    public Node below;
    public Node next;
    public Node prev;
    public int key;

    public Node(int key) {
        this.key = key;
    }
}
```

Skiplist.java

```java
import java.util.Random;

public class Skiplist {
    private Node head;
    private Node tail;

    private final int NEG_INF = Integer.MIN_VALUE;
    private final int POS_INF = Integer.MAX_VALUE;

    private int heightOfSkipList = 0;

    public Random rnd = new Random();

    public Skiplist() {
        head = new Node(NEG_INF);
        tail = new Node(POS_INF);
        head.next = tail;
        tail.prev = head;
    }

    public Node skipSearch(int key) {
        Node n = head;

        while (n.below != null) {
            n = n.below;

            while (key >= n.next.key) {
```

```
                n = n.next;
            }
        }

        return n;
    }

    public Node skipInsert(int key) {
        Node pos = skipSearch(key);
        Node q;

        int lvl = -1;
        int numOfHead = -1;

        if (pos.key == key) {
            return pos;
        }

        do {
            lvl++;
            numOfHead++;

            canIncreaseLvl(lvl);
            q = pos;

            while (pos.above == null) {
                pos = pos.prev;
            }

            pos = pos.above;

            q = insertAfterAbove(pos, q, key);

        } while (rnd.nextBoolean() == true);

        return q;
    }

    public Node remove(int key) {
        Node nodeToRemove = skipSearch(key);

        if ((nodeToRemove != null) && (nodeToRemove.key != key)) {
            return null;
        }

        removeRefToNode(nodeToRemove);

        while (nodeToRemove != null) {
            removeRefToNode(nodeToRemove);

            if (nodeToRemove.above != null) {
                nodeToRemove = nodeToRemove.above;
            } else {
                break;
            }
```

```
        }

        return nodeToRemove;
    }

    private void removeRefToNode(Node nodeToRemove) {
        Node afterNodeToRemove = nodeToRemove.next;
        Node beforeNodeToRemove = nodeToRemove.prev;

        beforeNodeToRemove.next = afterNodeToRemove;
        afterNodeToRemove.prev = beforeNodeToRemove;
    }

    private Node insertAfterAbove(Node pos, Node q, int key) {
        Node newNode = new Node(key);
        Node nodeBeforeNewNode = pos.below.below;

        setBeforeAndAfterRef(q, newNode);
        setAboveAndBelowRef(pos, key, newNode, nodeBeforeNewNode);

        return newNode;
    }

    private void setAboveAndBelowRef(Node pos, int key, Node newNode, Node
nodeBeforeNewNode) {
        if (nodeBeforeNewNode != null) {
            while (true) {
                if (nodeBeforeNewNode.next.key != key) {
                    nodeBeforeNewNode = nodeBeforeNewNode.next;
                } else {
                    break;
                }
            }
            newNode.below = nodeBeforeNewNode.next;
            nodeBeforeNewNode.next.above = newNode;
        }
        if (pos != null) {
            if (pos.next.key == key) {
                newNode.above = pos.next;
            }
        }
    }

    private void setBeforeAndAfterRef(Node q, Node newNode) {
        newNode.next = q.next;
        newNode.prev = q;
        q.next.prev = newNode;
        q.next = newNode;
    }

    private void canIncreaseLvl(int level) {
        if (level >= heightOfSkipList) {
            heightOfSkipList++;
            addEmptyLvl();
        }
```

```
    }

    private void addEmptyLvl() {
        Node newHeadNode = new Node(NEG_INF);
        Node newTailNode = new Node(POS_INF);

        newHeadNode.next = newTailNode;
        newHeadNode.below = head;
        newTailNode.prev = newHeadNode;
        newTailNode.below = tail;

        head.above = newHeadNode;
        tail.above = newTailNode;

        head = newHeadNode;
        tail = newTailNode;
    }

    public void printSkipList() {
        StringBuilder sb = new StringBuilder();
        sb.append("\nSkipList starting with top-left most node");

        Node start = head;

        Node highlvl = start;
        int level = heightOfSkipList;

        while (highlvl != null) {
            sb.append("\nLevel : " + level + "\n");

            while (start != null) {
                sb.append(start.key);

                if (start.next != null) {
                    sb.append(" : ");
                }
                start = start.next;
            }

            highlvl = highlvl.below;
            start = highlvl;
            level--;
        }
        System.out.println(sb.toString());
    }

}
```

Main.java

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Skiplist skiplist = new Skiplist();
        Scanner sc = new Scanner(System.in);

        while (true) {
            System.out.println("1. For insert node in skiplist");
            System.out.println("2. For delete node in skiplist");
            System.out.println("3. For search node in skiplist");
            System.out.println("4. For display skiplist");
            System.out.println("5. For Exit");
            int ch = sc.nextInt();

            switch(ch) {
                case 1:
                    System.out.println("Enter Element : ");
                    int key = sc.nextInt();
                    skiplist.skipInsert(key);
                    break;
                case 2:
                    System.out.println("Enter Element to Delete : ");
                    int key2 = sc.nextInt();
                    skiplist.remove(key2);
                    break;
                case 3:
                    System.out.println("Enter Element to Search : ");
                    int key3 = sc.nextInt();
                    Node temp = skiplist.skipSearch(key3);
                    if (temp != null) {
                        System.out.println(key3 + " is Present in skipList");
                    } else {
                        System.out.println(key3 + " is not present in
skipList");
                    }
                    break;
                case 4:
                    skiplist.printSkipList();
                    break;
                case 5:
                    System.exit(0);
            }

        }
    }

}
```

OUTPUT

```
1. For insert node in skiplist
2. For delete node in skiplist
3. For search node in skiplist
4. For display skiplist
5. For Exit
1
Enter Element :
3
1. For insert node in skiplist
2. For delete node in skiplist
3. For search node in skiplist
4. For display skiplist
5. For Exit
1
Enter Element :
6
1. For insert node in skiplist
2. For delete node in skiplist
3. For search node in skiplist
4. For display skiplist
5. For Exit
1
Enter Element :
7
```

```
1. For insert node in skiplist
2. For delete node in skiplist
3. For search node in skiplist
4. For display skiplist
5. For Exit
1
Enter Element :
9
1. For insert node in skiplist
2. For delete node in skiplist
3. For search node in skiplist
4. For display skiplist
5. For Exit
1
Enter Element :
12
1. For insert node in skiplist
2. For delete node in skiplist
3. For search node in skiplist
4. For display skiplist
5. For Exit
1
Enter Element :
19
1. For insert node in skiplist
2. For delete node in skiplist
3. For search node in skiplist
4. For display skiplist
5. For Exit
1
Enter Element :
17
```

```
1. For insert node in skiplist
2. For delete node in skiplist
3. For search node in skiplist
4. For display skiplist
5. For Exit
1
Enter Element :
26
1. For insert node in skiplist
2. For delete node in skiplist
3. For search node in skiplist
4. For display skiplist
5. For Exit
1
Enter Element :
21
1. For insert node in skiplist
2. For delete node in skiplist
3. For search node in skiplist
4. For display skiplist
5. For Exit
1
Enter Element :
25
```

```
1. For insert node in skiplist
2. For delete node in skiplist
3. For search node in skiplist
4. For display skiplist
5. For Exit
4

SkipList starting with top-left most node
Level : 5
-2147483648 : 2147483647
Level : 4
-2147483648 : 9 : 2147483647
Level : 3
-2147483648 : 9 : 2147483647
Level : 2
-2147483648 : 9 : 25 : 2147483647
Level : 1
-2147483648 : 9 : 12 : 17 : 25 : 2147483647
Level : 0
-2147483648 : 3 : 6 : 7 : 9 : 12 : 17 : 19 : 21 : 25 : 26 : 2147483647
```

```
1. For insert node in skiplist
2. For delete node in skiplist
3. For search node in skiplist
4. For display skiplist
5. For Exit
2
Enter Element to Delete :
26
1. For insert node in skiplist
2. For delete node in skiplist
3. For search node in skiplist
4. For display skiplist
5. For Exit
4

SkipList starting with top-left most node
Level : 5
-2147483648 : 2147483647
Level : 4
-2147483648 : 9 : 2147483647
Level : 3
-2147483648 : 9 : 2147483647
Level : 2
-2147483648 : 9 : 25 : 2147483647
Level : 1
-2147483648 : 9 : 12 : 17 : 25 : 2147483647
Level : 0
-2147483648 : 3 : 6 : 7 : 9 : 12 : 17 : 19 : 21 : 25 : 2147483647
```

```
1. For insert node in skiplist
2. For delete node in skiplist
3. For search node in skiplist
4. For display skiplist
5. For Exit
4

SkipList starting with top-left most node
Level : 5
-2147483648 : 2147483647
Level : 4
-2147483648 : 9 : 2147483647
Level : 3
-2147483648 : 9 : 2147483647
Level : 2
-2147483648 : 9 : 2147483647
Level : 1
-2147483648 : 9 : 12 : 17 : 2147483647
Level : 0
-2147483648 : 3 : 6 : 7 : 9 : 12 : 17 : 19 : 21 : 2147483647
1. For insert node in skiplist
2. For delete node in skiplist
3. For search node in skiplist
4. For display skiplist
5. For Exit
3
Enter Element to Search :
19
19 is Present in skipList
```