ASSIGNMENT 3

Programming and Problem Solving

Submitted by

Zalakben Rajendrakumar Patel (40164315), Jhanvi Arora (40162512).

CONTENTS :

1. Pseudocodes

2. Complexity Analysis

3. Design Structure and Observations

4. Demonstrations

PSEUDOCODES:

1. **Range Key (Recursive)**

Function rangeKey:

```
Input -> Student ID key1, Student ID key2
Output -> Number of keys present in range given
IF data structure is linked list:
    Node current = head
    flag = 0
    count = 0
    WHILE current is not null and current.data != k2
        IF current.data == k1
            flag = 1
        current = current.next
        IF flag == 1
            Count = Count + 1

    IF next node of Current == Null
        Print “Key not Found!”
        return -1
    return count-1

ELSE IF data structure is AVL tree:
    IF Root Node is NULL:
        Print “There are no records present in the tree.”
    IF Key1 node is not present:
        Print “Key 1 is not present”
    ELSE IF Key2 node is not present:
        Print “Key 2 is not present”
    ELSE
        rangeRecursive(rootNode, key1, key2)
```

Function rangeRecursive:

```
Input -> RootNode, Student ID key1, Student ID key2
Output -> Number of keys present in range given
IF RootNode is NULL:
    Return 0
IF RoodNode.studentID > key1 AND RootNode.studentID < key2:
    Return 1 + rangeRecursive(RootNode.LeftNode, key1, key2) + rangeRecursive(RootNode.rightNode, key1, key2)
ELSE IF RootNode.studentID < key1:
    rangeRecursive(RootNode.rightNode, key1, key2)
ELSE
    rangeRecursive(RootNode.leftNode, key1, key2)
```

2. **Get values for given key function (Recursive Function)**

Function getValues:

```
Input -> Student ID to search
Output -> Student Information of given student ID

IF data structure is linked list:
    Node temp = find(student ID)
    IF temp is not NULL:
        Return temp.getVal
    ELSE
```

```
        Print “Node not found”
        Return NULL
ELSE IF data structure is AVL Tree:
    IF RootNode is NULL:
        Return NULL
    foundNode = getStudentInfo(RootNode, Student ID)
    IF foundNode is NULL:
        Return “Student ID doesn’t exists”
    Return “Student Info of Given Node” + foundNode.studentInfo
```

Function getStudentInfo:

```
Input -> RootNode, Student ID
Output -> Matched node
IF RootNode is NULL:
    Return NULL
IF Student ID < Student ID of RootNode:
    Return getStudentInfo(RootNode.LeftNode, Student ID)
ELSE IF Student ID > Student ID of RootNode:
    Return getStudentInfo(RootNode.RightNode, Student ID)
ELSE IF Student ID == Student ID of RootNode:
    Return RootNode
Return NULL
```

Function find:

```
Input -> Student ID
Output -> Matched node
IF Head is NULL:
    Print “No node exists in the list!”
```

```
IF Head.data == Student ID:
    Return Head
ELSE:
    Node Current = Head
    WHILE Current is not NULL:
        IF Current.data == Student ID:
            Return Current
        Current = Current.NEXT
    Return NULL
```

3. ADD entry with Given Key and Value

Function addKey:

Input -> Student ID, Student Information

Output -> Created Node

```
If size==Threshold:
    Convert Linked List to AVL Tree Function
IF dataStructure is LinkedList:
    IF head==null:
        head = new Node(Student ID, Student Information)
    ELSE:
        IF head.getData() > ID:
            Node node = new Node(Student ID, head, Student
            Information)
            head=node
            size = size +1

        ELSE:
            Node tempNode = head
                WHILE tempNode.next is not NULL and tempNode.next.data  < Student ID:
                    tempNode = tempNode.next

            Node temp1 = tempNode.next;
            Node temp = new Node(Student ID, NULL, Student Information)
            tempNode.next = temp
            temp.next=temp1
            size = size + 1

IF data structure is AVL Tree:
    IF Node with Student ID Doesn’t Exists:
        IF root == null :
            return new AVL Node(Student ID, Student Info)
        IF student ID < root.student ID:
            root.leftNode = insertNode(root.leftNode, Student ID, Student Info)
        ELSE IF student ID > root.student ID:
            root.rightNode = insertNode(root.rightNode, Student ID, Student Info)
        Update height and balance for root
        Return BalanceAVLTree(root)
```

4. All Keys Present (Recursive function)

Function allKeys:

Input -> Student ID, Student Information

Output -> Created Node

```
IF dataStructure is LinkedList:
        Node current = head
        WHILE current is not NULL
                Print current.getData()+”\t”
                current = current.next

IF dataStructure is AVL Tree:
        IF node is NULL:
                return
        allKeys(node.leftNode)
        Print node.studentID + " \t"
        allKeys(node.rightNode)
```

COMPLEXITY ANALYSIS:

Time Complexity Change : **O(n) -> O(log n)**

- Insertion Operation
- Remove Operation
- getValues Operation
- Next Key
- Prev Key
- Range Key
- Generate

Justification:
All operations on linked list are performed by traversing until node satisfying required condition is met, hence the worst-case complexity comes out to be the order of O(n).
Operations performed on AVL are split across left and right subtrees depending on the value – hence the operation cost comes to be the order of O (Log n)

Time Complexity: **O(n)**

- allKeys()

Justification:
Traversing and returning all n keys either in LinkedList or AVL Tree shall result in same time complexity of order O(n).

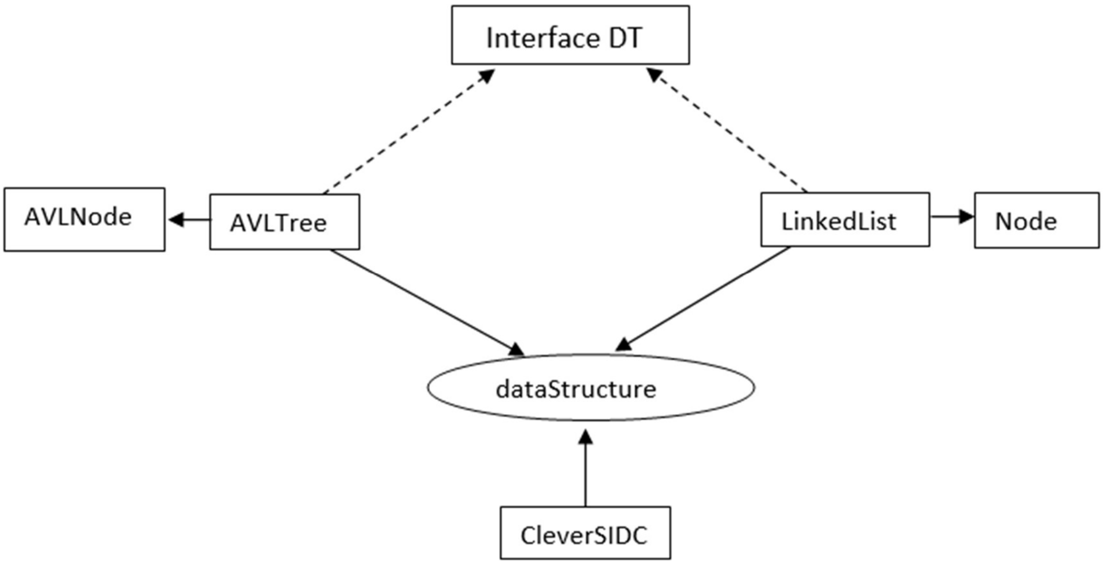
Space Complexity:

- Insert and Remove Operation: O(n+n) = O(2n) =O(n)

  Justification:
  Requires an extra temp pointer to store the object data structure while conversion.
- All other operations are completed occupying a space of order O(n), with additional few pointers of order O(1) depending on operation
- AVLNodes occupy an extra pointer than the LinkedList, thereby requiring n extra pointers to maintain the list.

DESIGN STRUCTURE AND OBSERVATIONS:

Design



Interface DT:

DT Interface implements methods that are common to both LinkedList and AVLTree classes- insert, remove, get all keys, find element, get next/previous key, find number of keys occurring in a specific range.

Classes:

1. Linked List:

- Implements Interface DT and defines implementation of the operations
- Is Composed of external Object derived from Class Node which consist of int, String and Node type respectively.
- Insertion of element (add Method): The elements in the linked list are inserted in sorted order to obtain keys in sorted order while calling the allkeys function.

2. AVL Tree:

- Implements Interface DT and defines implementation of operations recursively.
- Is Composed of external Object derived from the Class AVLNode which consists of int, String and 2 AVLNodes.

3. CleverSIDC:

- Contains an object dataStructure of Interface DT which based on the size of input switches between a LinkedList and AVL Tree
- When size is less than a given threshold, element is added in a linked list, for the threshold+1 element the dataStructure object converts its type and shifts to AVLTree type.
- Operations are called on dataStructure object which is implemented by either classes hence saves the overhead of maintaining two different Objects in the class.
- While converting the object from one to another, an intermediate pointer is created to retrieve data which is emptied while transferring elements.

Overall Structure:

Deriving implementation of both classes, the interface DT – serves as an adapter to store elements in the CleverSIDC. Fitting to each other's classes in the Driver Class, CleverSIDC – the proposed implementation follows Adapter Pattern Architecture as the design pattern.

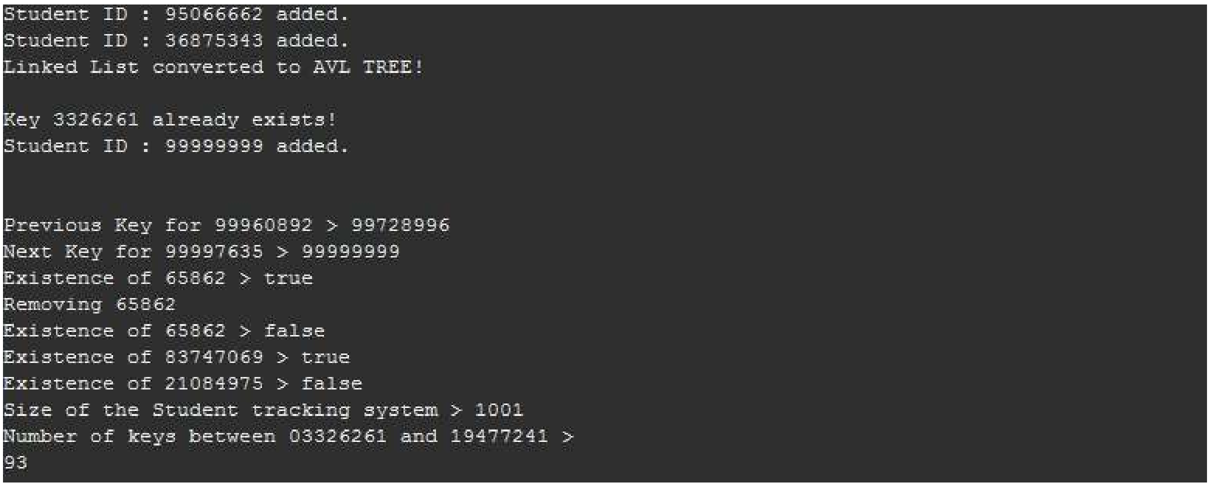
Saving space and availing the ease to switch between two data structures as required, the implemented structure upholds the pace of operations even with increasing size.

Observations:

- LinkedList allows faster insertion and access of elements ~70,000 elements but for entries over that the performance degrades and fails to yield prompt results.
- In contrast, AVLTree allows faster insertion of elements upto ~ 1,000,000 elements (maximum tested limit)
- Hence, the transition and adapting to AVL data structure from LinkedList is apt and fitting to problem statement all the while compromising the space for n extra pointers over saving the time by more than half.

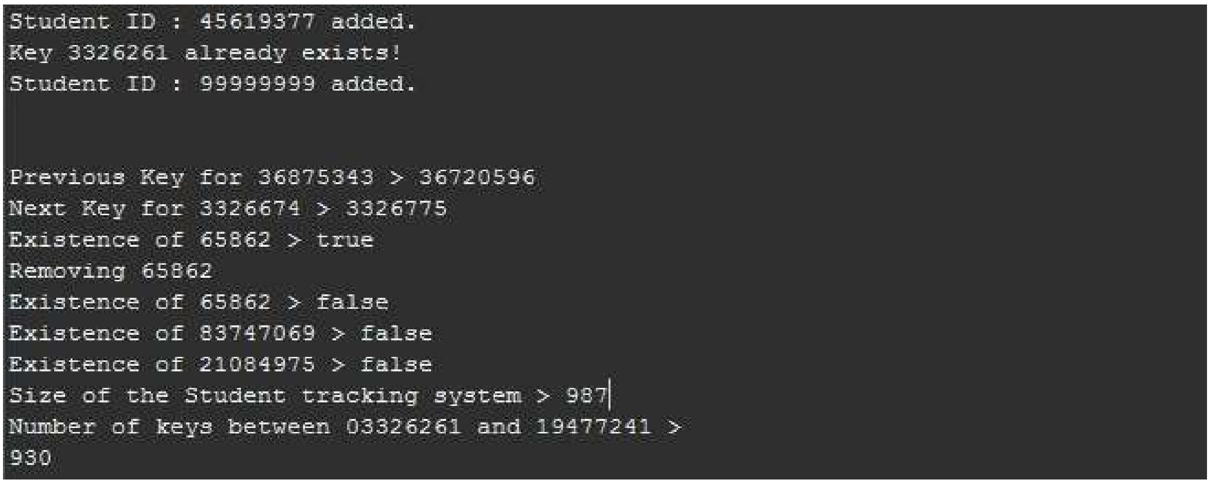
DEMONSTRATIONS:

Case 1: 1002 Entries, invoked change from LL to AVL



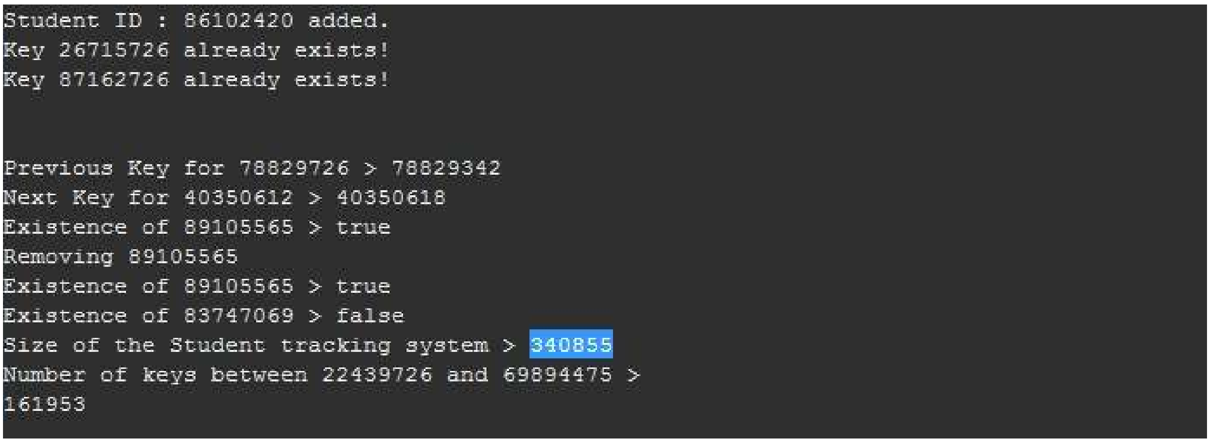
```
Student ID : 95066662 added.
Student ID : 36875343 added.
Linked List converted to AVL TREE!

Key 3326261 already exists!
Student ID : 99999999 added.


Previous Key for 99960892 > 99728996
Next Key for 99997635 > 99999999
Existence of 65862 > true
Removing 65862
Existence of 65862 > false
Existence of 83747069 > true
Existence of 21084975 > false
Size of the Student tracking system > 1001
Number of keys between 03326261 and 19477241 >
93
```

Case 2: Verifying functions of LinkedList, cannot add existing keys.



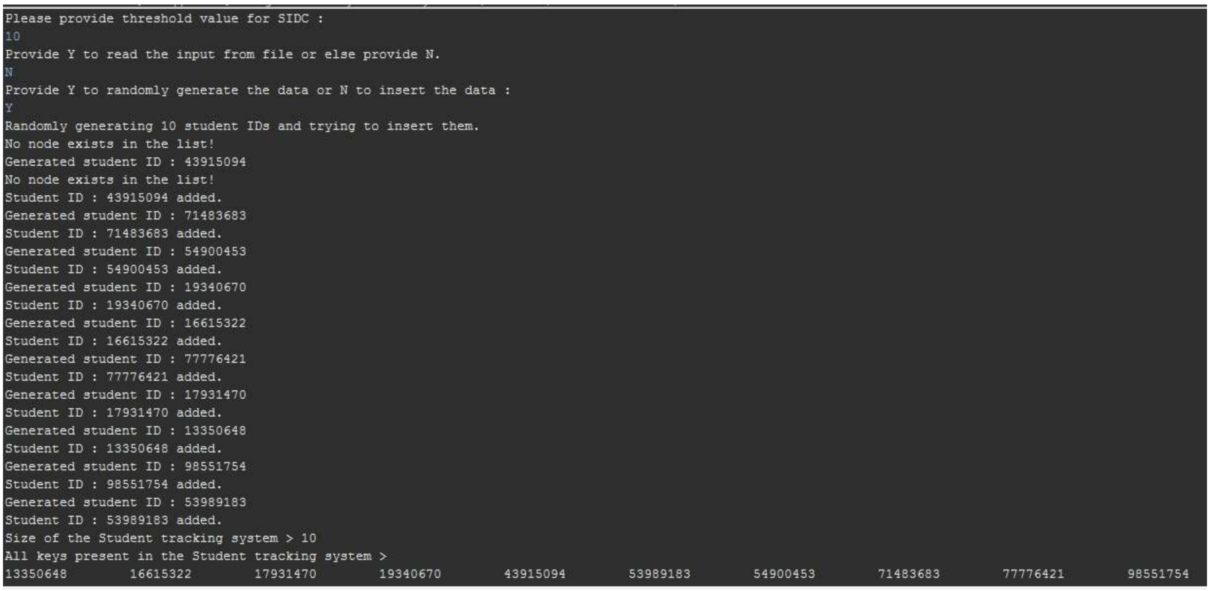
```
Student ID : 45619377 added.
Key 3326261 already exists!
Student ID : 99999999 added.


Previous Key for 36875343 > 36720596
Next Key for 3326674 > 3326775
Existence of 65862 > true
Removing 65862
Existence of 65862 > false
Existence of 83747069 > false
Existence of 21084975 > false
Size of the Student tracking system > 987
Number of keys between 03326261 and 19477241 >
930
```

Case 3: Verifying functions of AVL Tree



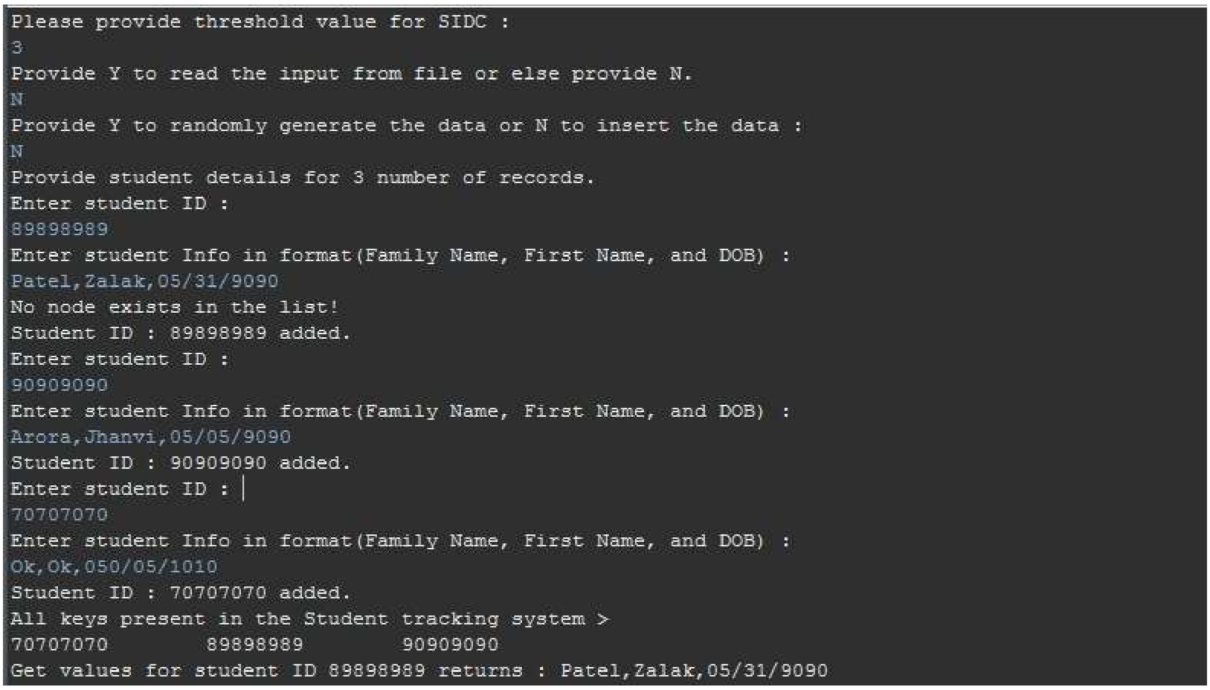
```
Student ID : 86102420 added.
Key 26715726 already exists!
Key 87162726 already exists!


Previous Key for 78829726 > 78829342
Next Key for 40350612 > 40350618
Existence of 89105565 > true
Removing 89105565
Existence of 89105565 > true
Existence of 83747069 > false
Size of the Student tracking system > 340855
Number of keys between 22439726 and 69894475 >
161953
```

Case 4: Adding keys via Generate Function and Adding them based on number of entries generated, verifying all keys function as well.



```
Please provide threshold value for SIDC :
10
Provide Y to read the input from file or else provide N.
N
Provide Y to randomly generate the data or N to insert the data :
Y
Randomly generating 10 student IDs and trying to insert them.
No node exists in the list!
Generated student ID : 43915094
No node exists in the list!
Student ID : 43915094 added.
Generated student ID : 71483683
Student ID : 71483683 added.
Generated student ID : 54900453
Student ID : 54900453 added.
Generated student ID : 19340670
Student ID : 19340670 added.
Generated student ID : 16615322
Student ID : 16615322 added.
Generated student ID : 77776421
Student ID : 77776421 added.
Generated student ID : 17931470
Student ID : 17931470 added.
Generated student ID : 13350648
Student ID : 13350648 added.
Generated student ID : 98551754
Student ID : 98551754 added.
Generated student ID : 53989183
Student ID : 53989183 added.
Size of the Student tracking system > 10
All keys present in the Student tracking system >
13350648        16615322        17931470        19340670        43915094        53989183        54900453        71483683        77776421        98551754
```

Case 5: Verifying and Displaying entries via getValues function



```
Please provide threshold value for SIDC :
3
Provide Y to read the input from file or else provide N.
N
Provide Y to randomly generate the data or N to insert the data :
N
Provide student details for 3 number of records.
Enter student ID :
89898989
Enter student Info in format(Family Name, First Name, and DOB) :
Patel,Zalak,05/31/9090
No node exists in the list!
Student ID : 89898989 added.
Enter student ID :
90909090
Enter student Info in format(Family Name, First Name, and DOB) :
Arora,Jhanvi,05/05/9090
Student ID : 90909090 added.
Enter student ID :
70707070
Enter student Info in format(Family Name, First Name, and DOB) :
Ok,Ok,050/05/1010
Student ID : 70707070 added.
All keys present in the Student tracking system >
70707070        89898989        90909090
Get values for student ID 89898989 returns : Patel,Zalak,05/31/9090
```