Manish Sehgal – 40165366
Pulkit Kakkar – 40160971

COMP 6481 – Programming and Problem Solving

Design

For the assignment, two data structures are implemented. A sorted doubly linked list is used if the range is less than the threshold. The definition of insertion operation keeps the list in order in the list. Once the input size exceeds the threshold value, an AVL Tree is used as it takes comparatively less time to perform operations.

Time complexity and space complexity of different methods:

i) *SetSIDCThreshold (Size):*

This method takes O(1) time as it is the threshold value which will be set only once.

ii) *generate():*

This method takes O(n) time to generate any random 8 digit number for list and O(logn) for AVL tree.

iii) *allKeys(CleverSIDC):*

For a doubly linked list, this method takes O(n) time to return all the keys as it needs to traverse and go over each and every node of the linked list and get the key from each one of them. And for the AVL Trees, this method also takes O(n) time to return all the keys of the tree as all the nodes of the tree will be visited once.

iv) *add(CleverSIDC,key,value):*

For the doubly linked list, the add method will take O(n) time as in order to add a new node in a sorted doubly linked list, in the worst case it needs to traverse the whole list and compare with the new node to add at end. For the AVL tree, the add method will take O(log n) time as by comparing the root it can be decided in which direction to traverse and add the element.

v) remove(CleverSIDC,key):

For the doubly linked list, the remove method will take O(n) time as in order to remove a node from a sorted doubly linked list, in the worst case it needs to traverse the whole list and compare with node data to remove from the list. For the AVL tree, the remove method will take O(log n) time as by comparing the root it can be decided in which direction to traverse and remove the element.

vi) *getValues(CleverSIDC,key):*

For the doubly linked list, to get all the values from every node we need to go to each and every element. In other way, we need to traverse the whole list. So, to obtain all the values of the list the time complexity will be O(n). For the AVL tree also we need to visit each and every node to get their value so the time complexity will be o(n).

vii) *nextKey(CleverSIDC,key):*

In doubly linked list, in the worst case to get the next key of a node we need to traverse till end. So, the time complexity will be O(n). And for the AVL tree we can just divide and decide in which direction to traverse until the node is found. So, the time complexity will be O(log n).

viii) *prevKey(CleverSIDC,key):*

In doubly linked list, in the worst case to get the previous key of a node we need to traverse till end. So, the time complexity will be O(n). And for the AVL tree we can just divide and decide in which direction to traverse until the node is found. So, the time complexity will be O(log n).

ix) *rangeKey(key1, key2):*

For doubly linked list, we need to first traverse till the key 1 and then traverse till the key 2 is found. So, in the worst case we can have to traverse the whole linked list. Hence, the time complexity will be O(n). And for the AVL tree, the time complexity will be O(log n).

Psuedo Code

1) METHOD SetSIDCThreshold (size)

```
SET SIZE = size
IF(curentSize < SIZE)
    IF CURRENT_LIST is AVL_TREE
        Convert_To_Doubly_Linked_List()
    END_IF
ELSE
    IF CURRENT_LIST is DOUBLY_LINKED_LIST
        Convert_To_AVL()
    END_IF

END_IF
```

END

2)

```
METHOD PRINT_INORDER(Node n)
  IF(n IS NOT NULL)
      PRINT_INORDER(n.left)
      Print_Data(n);
      PRINT_INORDER(n.right)
  END
END

METHOD PRINT_LIST
  Node temp = Head

  While(temp not null)
      Print(temp);
      Temp = temp.next
  End_While

END
```

```
    METHOD all_keys()
      // Avl Trees
      PRINT_INORDER(this.head);

      //Doubly Linked List

    END

3)  METHOD prevKey(key)
      // Avl Trees
      Int[] arr = getItemsInOrder();
      Int index = findIndexInArr(arr);

      IF(index == 0)
              Print("The Key is First Key")
      Else If(index == -1)
              Print("The Key doesn't exist")
      Else
              Print(arr[index-1])

      // Doubly Linked List
              Node temp = findNode(key);
              If(temp == null)
                      Print("The Key doesn't exist")
              ELSE
                      If(temp.prev is Null)
                              Print("The Key is First Key")
                      Else
                              Print(temp.next)
              End_If
    END

4)  METHOD nextKey(nextKey)
      // Avl Trees
      Int[] arr = getItemsInOrder();
      Int index = findIndexInArr(arr);

      IF(index == arr.length - 1)
              Print("The Key is Last Key")
      Else If(index == -1)
      Print(“The Key doesn’t exist”)
  Else
      Print(arr[index+1])

  // Doubly Linked List
      Node temp = findNode(key);
      If(temp == null)
          Print(“The Key doesn’t exist”)
      ELSE
          If(temp.next is Null)
              Print(“The Key is Last Key”)
          Else
              Print(temp.next)
      End_If
END
```