DATA STRUCTURES AND ALGORITHMS LABORATORY

GROUP A

ASSIGNMENT 2

Name :- Ojus Pravin Jaiswal

Roll No. :- SACO19108

Division :- A

DATA STRUCTURE LABORATORY

Group A

Assignment 2

Title: For given set of elements create skip list. Find the element in the set that is closest to some given value. (note: Decide the level of element in the list Randomly with some upper limit)

Objectives:

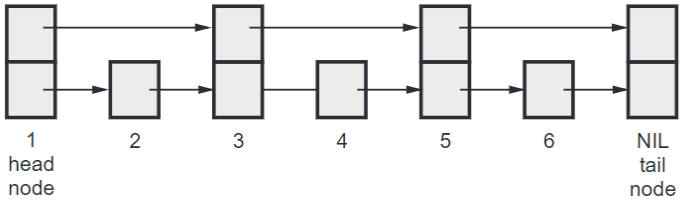
1. To understand SkipList
2. To understand concept of inserting into skiplist
3. To understand concept of searching a skiplist.

Outcome:

- Define class for Skiplist
- Analyze the insert and search functionalities of Skiplist data structure

Theory:

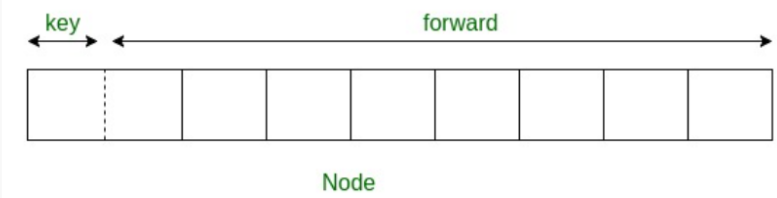
- A skip list is a probabilistic data structure. Made up of series of nodes connected one after the another
- Each node contains a key – value pair as well as one or more references, or pointers to nodes further along the lists
- The number of references is determined randomly. The number of references a node contains is called the node level
- There are two special nodes :-Head node & Tail Node
- Efficient implementation of dictionary using sorted chain. The skip list is used to store a sorted list of elements or data with a linked list.
- It allows the process of the elements or data to view efficiently. In one single step, it skips several elements of the entire list, which is why it is known as a skip list.



Each element in the list is represented by a node, the level of the node is chosen randomly while insertion in the list. Level does not depend on the number of elements in the node. The level for node is decided by the following algorithm –

randomLevel()
lvl := 1
//random() that returns a random value in [0...1)
while random() < p and lvl < MaxLevel do
lvl := lvl + 1
return lvl

MaxLevel is the upper bound on number of levels in the skip list. It can be determined as – $L(N) = \log_{\{p/2\}}\{N\}$. Above algorithm assure that random level will never be greater than MaxLevel. Here p is the fraction of the nodes with level i pointers also having level i+1 pointers and N is the number of nodes in the list.
Each node carries a key and a forward array carrying pointers to nodes of a different level. A level i node carries i forward pointers indexed through 0 to i.



Insertion in Skip List

We will start from highest level in the list and compare key of next node of the current node with the key to be inserted. Basic idea is If –

Key of next node is less than key to be inserted then we keep on moving forward on the same level
Key of next node is greater than the key to be inserted then we store the pointer to current node i at update[i] and move one level down and continue our search.
At the level 0, we will definitely find a position to insert given key. Following is the pseudo code for the insertion algorithm –

Insert(list, searchKey)
local update[0...MaxLevel+1]
x := list -> header
for i := list -> level downto 0 do
while x -> forward[i] -> key forward[i]
update[i] := x
x := x -> forward[0]
lvl := randomLevel()

```
if lvl > list -> level then
for i := list -> level + 1 to lvl do
   update[i] := list -> header
   list -> level := lvl
x := makeNode(lvl, searchKey, value)
for i := 0 to level do
   x -> forward[i] := update[i] -> forward[i]
   update[i] -> forward[i] := x
```

Searching an element in Skip list

Searching an element is very similar to approach for searching a spot for inserting an element in Skip list. The basic idea is if –

Key of next node is less than search key then we keep on moving forward on the same level.
Key of next node is greater than the key to be inserted then we store the pointer to current node i at update[i] and move one level down and continue our search.
At the lowest level (0), if the element next to the rightmost element (update[0]) has key equal to the search key, then we have found key otherwise failure.
Following is the pseudo code for searching element –

Search(list, searchKey)

```
x := list -> header
-- loop invariant: x -> key  level downto 0 do
   while x -> forward[i] -> key  forward[i]
x := x -> forward[0]
if x -> key = searchKey then return x -> value
else return failure
```

Software Required: Python

Input: Elements to be inserted and searched

Program:

```
import random

class Node(object):
```

```
        '''
        Class to implement node
        '''
        def __init__(self, key, level):
                self.key = key

                # list to hold references to node of different level
                #Allocate memory to forward & Fill forward with NULL
                self.forward = [None]*(level+1)

class SkipList(object):
        '''
        Class for Skip list
        '''
        def __init__(self, max_lvl, P):
                # Maximum level for this skip list
                self.MAXLVL = max_lvl

                # P is the fraction of the nodes with level
                # i references also having level i+1 references
                self.P = P

                # create header node and initialize key to -1
                self.header = self.createNode(self.MAXLVL, -1)

                # current level of skip list
                self.level = 0

        # create new node
        def createNode(self, lvl, key):
                n = Node(key, lvl)
                return n

        # create random level for node
        def randomLevel(self):
                lvl = 0
                while random.random()<self.P and \
```

```
                lvl<self.MAXLVL:lvl += 1
        return lvl

# insert given key in skip list
def insertElement(self, key):
        # create update array and initialize it
        update = [None]*(self.MAXLVL+1)
        current = self.header

        '''
        start from highest level of skip list
        move the current reference forward while key
        is greater than key of node next to current
        Otherwise inserted current in update and
        move one level down and continue search
        '''
        for i in range(self.level, -1, -1):
                while current.forward[i] and \
                        current.forward[i].key < key:
                        current = current.forward[i]
                update[i] = current

        '''
        reached level 0 and forward reference to
        right, which is desired position to
        insert key.
        '''
        current = current.forward[0]

        '''
        if current is NULL that means we have reached
        to end of the level or current's key is not equal
        to key to insert that means we have to insert
        node between update[0] and current node
'''
        if current == None or current.key != key:
                # Generate a random level for node
```

```
                rlevel = self.randomLevel()

                '''
                If random level is greater than list's current
                level (node with highest level inserted in
                list so far), initialize update value with reference
                to header for further use
                '''
                if rlevel > self.level:
                        for i in range(self.level+1, rlevel+1):
                                update[i] = self.header
                        self.level = rlevel

                # create new node with random level generated
                n = self.createNode(rlevel, key)

                # insert node by rearranging references
                for i in range(rlevel+1):
                        n.forward[i] = update[i].forward[i]
                        update[i].forward[i] = n

                print("Successfully inserted key {}".format(key))


def searchElement(self, key):
        current = self.header

        '''
        start from highest level of skip list
        move the current reference forward while key
        is greater than key of node next to current
        Otherwise inserted current in update and
        move one level down and continue search
        '''
        for i in range(self.level, -1, -1):
                while(current.forward[i] and\
                        current.forward[i].key < key):
```

```
                    current = current.forward[i]

            # reached level 0 and advance reference to
            # right, which is prssibly our desired node
            current = current.forward[0]

            # If current node have key equal to
            # search key, we have found our target node
            if current and current.key == key:
                print("Found value ", key)
            else:
               print ("Closest value",current.key)



    # Display skip list level wise
    def displayList(self):
        print("\n*****Skip List******")
        head = self.header
        for lvl in range(self.level+1):
            print("Level {}: ".format(lvl), end=" ")
            node = head.forward[lvl]
            while(node != None):
                print(node.key, end=" ")
                node = node.forward[lvl]
            print("")

# Driver to test above code
def main():
    print("----------SKIP LIST----------\n")
    l=int(input("Enter number of maximum level : "))
    f=float(input("Enter the fraction of the nodes : "))
    lst = SkipList(l,f)
    n=int(input("Enter number of elements : "))
    for i in range(1, n+1):
        e=int(input("\nEnter element no. %d : " %i))
        lst.insertElement((e))
    while True:
```

```
        print("\n-----Menu-----\n1) Insert an element in skip list\n2) Display the skip list\n3) Search an element in skip list\n4) Exit the program")
        c=int(input("Enter your choice : "))
        if c==1:
                e=int(input("\nEnter new element : "))
                lst.insertElement(e)
        elif c==2:
                lst.displayList()
        elif c==3:
                s=int(input("Enter element to be searched : "))
                lst.searchElement(s)
        elif c==4:
                print("Program Exited!!!")
                exit(0)
        else:
                print("Wrong choice entered!!!")

main()
```

DATA STRUCTURE LABORATORY

Output:

```
----------SKIP LIST----------

Enter number of maximum level : 3
Enter the fraction of the nodes : 0.5
Enter number of elements : 10

Enter element no. 1 : 0
Successfully inserted key 0

Enter element no. 2 : 10
Successfully inserted key 10

Enter element no. 3 : 20
Successfully inserted key 20

Enter element no. 4 : 30
Successfully inserted key 30

Enter element no. 5 : 40
Successfully inserted key 40

Enter element no. 6 : 60
Successfully inserted key 60

Enter element no. 7 : 70
Successfully inserted key 70

Enter element no. 8 : 80
Successfully inserted key 80

Enter element no. 9 : 90
Successfully inserted key 90

Enter element no. 10 : 100
Successfully inserted key 100
```

DATA STRUCTURE LABORATORY

```
-----Menu-----
1) Insert an element in skip list
2) Display the skip list
3) Search an element in skip list
4) Exit the program
Enter your choice : 1

Enter new element : 50
Successfully inserted key 50

-----Menu-----
1) Insert an element in skip list
2) Display the skip list
3) Search an element in skip list
4) Exit the program
Enter your choice : 2

*****Skip List******
Level 0:  0 10 20 30 40 50 60 70 80 90 100
Level 1:  0 10 40 50 80 90 100
Level 2:  40 90 100
Level 3:  40 90

-----Menu-----
1) Insert an element in skip list
2) Display the skip list
3) Search an element in skip list
4) Exit the program
Enter your choice : 3
Enter element to be searched : 39
Closest value 40

-----Menu-----
1) Insert an element in skip list
2) Display the skip list
3) Search an element in skip list
4) Exit the program
Enter your choice : 4
Program Exited!!!

[Program finished]
```

Conclusion: This program statement implements Skiplist .