**Advanced Data Structures and**

**Algorithm Analysis**

# Laboratory Projects

**Skip Lists**

**Author Names**

Zhang Wenqi

Wu Qinran

Chen Yongchen

**Date: 2020-05-22**

### Chapter 1: Introduction

Problem description:

skiplist is a kind of randomized data structure, which is a relatively efficient data structure. Skiplist is based on a parallel linked list. The time complexity of simple insertion, deletion, and search are all O (logN). The performance of skiplist is similar to that of the red-black tree and the implementation is relatively simple. Therefore, in many famous projects, skiplist is used to replace red and black trees, such as leveldb and lucence.

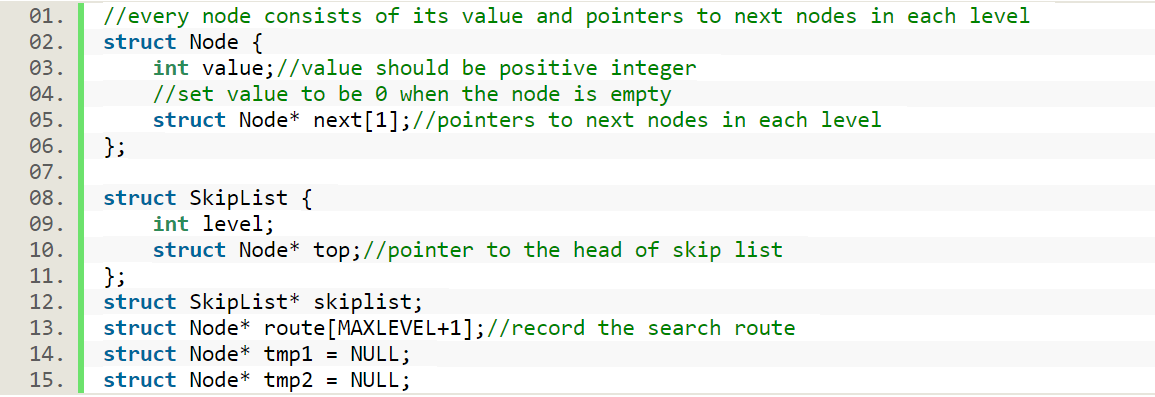
Background of the data structures and the algorithms:

The skip table has the following characteristics:

1. The skiplist is composed of many layers, usually 10-20 layers;
2. Layer 0 of the hop table contains all elements;
3. The node values of each layer are ordered;
4. If element x appears in layer I, all layers smaller than I contain X;
5. The higher the number of layers, the fewer nodules;
6. Each node contains two pointers, one pointing to the next element in the same layer of linked list, and the other pointing to the element with the same value in the next layer;

### Chapter 2: Data Structure / Algorithm Specification

1. The data structure:



1. The key algorithm:

Insert(int value) uses to insert the value in skiplist. There are four steps to insert a skiplist. The first step is to find the place to insert. The second step is to determine whether the value already exists. The third step is randomly get the level to insert. The fourth step is to adjust the pointer.

Function:

void Insert(int value){

//record the search rote and find the place to insert

GetRoute(value);

IF value is repeated

Print : value is already in skip list

return

END IF

insertlevel = randomly get the level to insert

tmp = initial the new node of tmp

For insertlevel time

insert and update every level

END FOR

}

Search(int value) uses to judge whether the value is in skip list or not.

Function:

void Search (int value){

Initialize tmp1 to NULL, tmp2 to skiplist->top

FOR the level of skiplist time

tmp2 to skiplist->top

WHILE 1

tmp1 to tmp2->next[i]

IF tmp1 is not NULL

IF the value of tmp1 <= value

IF the value of tmp1 == value

Print: value is in skip list

END IF

ELSE break

ELSE break

END WHILE

END FOR

}

The delete operation is similar to the insert operation. The first step is to find the node to delete. The second step is to determine whether the value already exists. The third step is delete the node update every level. The fourth step is to update the total level.

Function:

void Delete(int value){

//record the search rote and find the place to insert

GetRoute(value);

IF tmp1 is not NULL and the value of tmp1 == value

FOR the level of skiplist time

delete and update every level

END FOR

FOR the level of skiplist time

update the total level

END FOR

Print: delete value succeddfully

END IF

Print: delete failed

{

### Chapter 3: Testing Results

3.1 Accuracy of the program

In order to test the correctness of the program, we design a test program to print out the list after inserting and deleting to test whether it matches the standard of the skiplist.

Test case includes: insert, search and delete values, insert values that are already existing, search and delete values that are not existing.

Input:

//Input file(test.txt) is like:

// I 1

// I 2

// I 1

// I 3

// I 10

// S 3

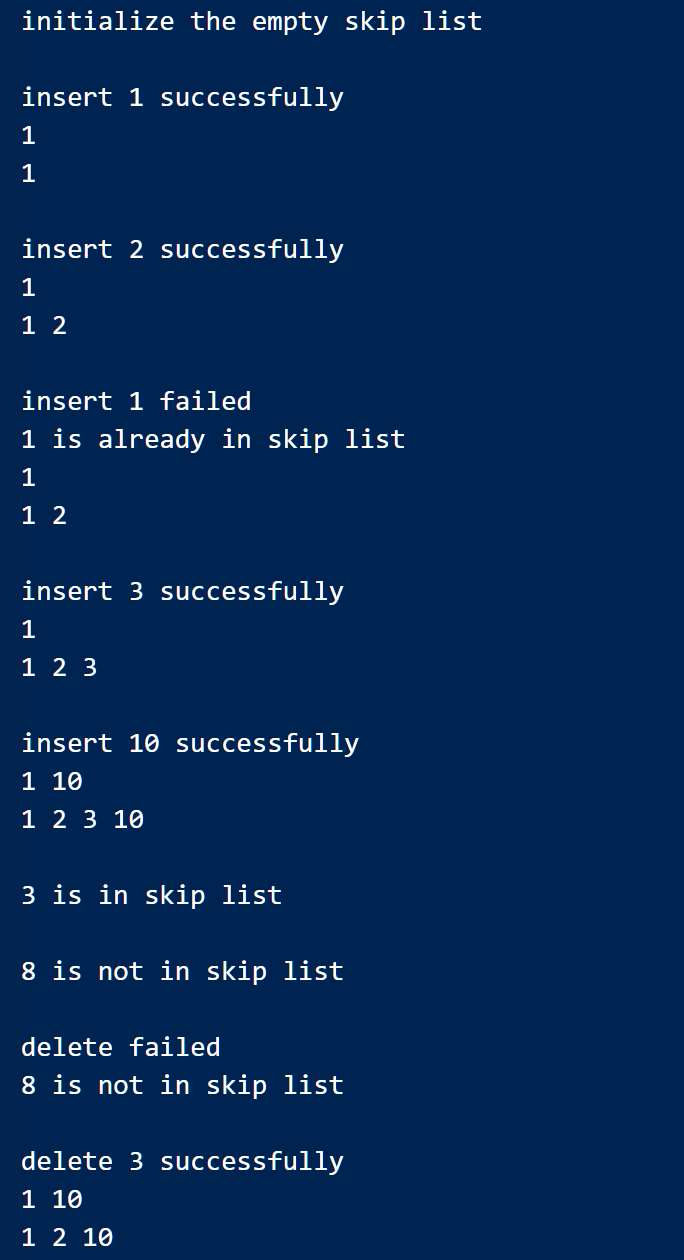
// S 8

// D 8

// D 3

// ......

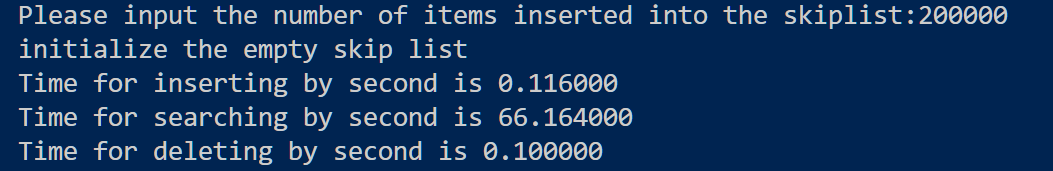
Output:



3.2 Time complexity of the program

We generate several test cases of different sizes to illustrate the time bound of the program by calculating the running time of the algorithm.

Output file is like：



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | Insert | | | Search | | | Delete | | |
| 1000 | 0.000 | 0.000 | 0.000 | 0.005 | 0.004 | 0.005 | 0.000 | 0.001 | 0.000 |
| Avg | 0.000 | | | 0.005 | | | 0.000 | | |
| 10000 | 0.004 | 0.004 | 0.005 | 0.452 | 0.454 | 0.459 | 0.004 | 0.003 | 0.004 |
| Avg | 0.004 | | | 0.455 | | | 0.004 | | |
| 50000 | 0.032 | 0.029 | 0.029 | 11.783 | 12.832 | 11.788 | 0.030 | 0.021 | 0.028 |
| Avg | 0.030 | | | 12.134 | | | 0.028 | | |
| 100000 | 0.061 | 0.056 | 0.063 | 29.807 | 30.332 | 29.794 | 0.050 | 0.055 | 0.057 |
| Avg | 0.060 | | | 29.978 | | | 0.054 | | |
| 200000 | 0.116 | 0.106 | 0.103 | 66.164 | 67.580 | 68.573 | 0.100 | 0.108 | 0.107 |
| Avg | 0.108 | | | 67.439 | | | 0.105 | | |

### Chapter 4: Analysis and Comments

4.1 Time complexity analysis

Time complexity for each operation are as follows:

InitSkiplist() O(1)

GetRoute() O(logn)

Insert () O(1+logn)=O(logn)

Search() O(n)

Delete() O(1+logn)=O(logn)

In “insert” operation, we use function GetRoute() to record the search route and find the place to insert, then randomly get the level to insert and update, thus the time complexity reaches O(logn)

In “search” operation, we need to go through from the top level to the bottom, thus the time complexity reaches O(n).

In “delete” operation, similar to “insert” operation, we record the search route and find the place of the given value and update each level, thus the time complexity reaches O(logn)

4.2 Space complexity analysis

Let’s suppose the length of a single list has N nodes, the first level index has about N / 2 nodes, the second level index has about N / 4 nodes, and the third level index has about N / 4 nodes, etc. Thus an equal ratio sequence is formed: N / 2, N / 4, N / 8,... 8, 4, 2. The indexes of these levels add up to n-2, so the space complexity of skip list reaches O (n)

4.3 Comments

We know in an ordinary sorted linked list, search, insert and delete operations are in O(n) because the list must be scanned node-by-node from the head to find the relevant node. Skip list allows us to scan down the list in bigger steps, thus reducing the cost of scanning.

### Appendix: Source Code (if required)

### References

[1] Mark Allen Weiss, “Data Structures and Algorithm Analysis in C”, *机械工业出版社*,p.345~351, (2004.1)

### Author List

### Declaration

***We hereby declare that all the work done in this project titled "*** ***Shortest Path Algorithm with Heaps " is of our independent effort as a group.***

### Signatures

张雯琪

陈咏琛

吴沁珃