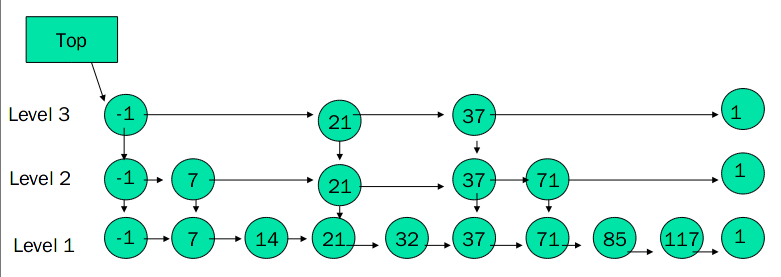
**Chapter 3: Testing Results**

## test the correctness

First, we use small-size data to test our program’s correctness. When we input data [7, 14, 21, 32, 37, 71, 85, 117]. And the result is shown below:



## test the time complexity

For time complxity test, in this project, we use random method to create a series of test cases. While testing data of small input sizes, we should use different test cases in only one test instance, so we use the random method for several times in one test instance.

There is some example test cases, including input sizes of 300, 1000, 3000, 10000, 30000, 100000, which is shown in our appendix. However, actually, we use the power of 2, from 256 to 65536 to test the time complexity. In these test cases, the ‘key’ is distinct and one ‘value’ may be the same with another ‘value’.

Here is the run time table, we test the performance of searching, inserting and deleting, and here is the result:

Searching Performance

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | 256 | 512 | 1024 | 2048 | 4096 | 8192 | 16384 | 32768 | 65536 |
| time(ms)/every 1000 search | 3 | 6 | 17 | 45 | 83 | 113 | 130 | 170 | 175 |

Inserting Performance

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | 256 | 512 | 1024 | 2048 | 4096 | 8192 | 16384 | 32768 | 65536 |
| time(ms)/every 1000 inserts | 19 | 16 | 20 | 43 | 80 | 120 | 140 | 180 | 185 |

Deleting Performance

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | 256 | 512 | 1024 | 2048 | 4096 | 8192 | 16384 | 32768 | 65536 |
| time(ms)/every 1000 deletion | 3 | 2 | 10 | 16 | 46 | 86 | 135 | 163 | 172 |

And we plot the run times vs. input sizes for illustration, the result is shown below:

As we can see, the time complexity of all there operations is approximately **O(log n).**

**Chapter 4: Analysis and comments**

## time complexity

The time required to execute the Search, Delete and Insert operations is dominated by the time required to search for appropriate element. For the Insert and Delete operations, there is an additional cost proportional to the level of the node being inserted or deleted. The time required to find an element is proportional to the length of the search path, which is determined by the pattern in which elements with different levels appear as we traverse the list.

We analyze the search path backwards, traveling up and to the left. Although the levels of nodes in the list are known and fixed when the search is performed, we act as if the level of a node is being determined only when it is observed while backtracking the search path.

At any particular point in the climb, we are at a situation similar to situation a in Figure 6below : When we are at the i-th forward pointer of a node x and we have no knowledge about the levels of node to the left of x or about the level of x, other than that the level of x must be at least i. If the level of x is equal to i, then we are in situation b. If the level of x is greater than i, then we are in situation c. The probability that we are in situation c is p. The probability that we are in situation c is p. Each time we are in situation c, we climb up a level. Let C(k) = the expected cost (i.e, length) of a search path that climbs up k levels in an infinite list:

C(0) = 0

C(k) = (1–p) (cost in situation b) + p (cost in situation c)

By substituting and simplifying, we get:

C(k) = (1–p) (1 + C(k)) + p (1 + C(k–1))

C(k) = 1/p + C(k–1)

C(k) = k/p

As we can see, when we bump form level 1 to level L(n) (L(n) is the expected higher level in the skip list, log1/pn), the upper bound time is L(n)/p.

We use a different analysis technique for the rest of the journey. The number of leftward movements remaining is bounded by the number of elements of level L(n) or higher in the entire list, which has an expected level of 1.

So total expected cost to climb out of a list of n elements <= L(n)/p+1

which is O(log n)

## space complexity

From the point of expectation the i-th is expected to have n/(2^i-1) nodes, so the nodes in the skip list should be n+n/2+n/4+……=2n, so the space complexity is O(n).

## Comments

The skip list’s advantage is that it is easy to implement. The time of insertion, deleting and searching is not only an average bound, but also a “with high probability” bound. That is to say, as n grows, the time will more and more distribute on c lg n, which is a significative and strong proposition.