The Skip List

Group 8: 颜晗、吴俊贤、钱于飞

Abstract: In this project, we have implemented a new data structure, skip list. It is a simple alternative to balanced tree.   Skip lists are balanced by consulting a random number generator. It may have a bad performance in an unlucky case, but in fact no input sequence consistently produces the worst-case performance. To see its real performance in practical use, we designed three test for it and analyze its causes.

As we all know, balanced trees have a good performance in many conditions, but it also has a complex implementation methods. Skip lists can be a probabilistic alternative to balanced trees. Using a random number generator, they can have a not-bad performance at most time. Although they may have a very bad performance, the possibility can be so low that we can afford the cost. We will introduce our implementation and test of skip lists as follows.

1 implementation

The node of skip lists have a key and value the same as single lists, but they have more than one point which point to next node in different levels. Skip lists have a probability which decide how many in one level will climb up to a higher level and a maximum level. We will introduce how search ,insert and delete in skip lists.

Search is the main operation in skip lists as we also need to find the position when we do insertion and deletion. We begin at a high level and the head of list, go right before we meet a node whose key is bigger than what we want to find. When we meet a node having a key bigger, we will go down one level and repeat the process. The pseudo code is as follows.

```
Search(list, Mykey)
    x:=list -> head;
    for i:=list->maxLevel downto 1 do
        while x -> next[i] -> key < Mykey do
            x:=x->next[i]
    x:=x->next[1]
    if x->key == Mykey then return x->value
    else  return invalid value
```

Insertion and deletion are also search processes, but we need to maintain a vector to store the `next` pointer of maximum which is smaller than `Mykey` in different . It can help us do insertion and deletion in different level easily. Of course, insertion may do more things than deletion, as we need to generate random numbers to decide the level. The pseudo code of `randomLevel` and insertion are as follows, deletion is similar to insertion.

```
RandomLevel(p)
    -- p is a number between 1 and 100--
    newLevel:=1
    while( random()%100 < p and newLevel < Maxlevel we set)
        newLevel:= newLevel+1
    return newLevel
insertNode(list, Mykey, value)
    local vector
    x:=list->head
    for i:= list->level downto 1 do
        while x->next[i]->key < Mykey do
            x:= x-> next[i]
        push x to vector
    x:=x->next[1]
    if x->key = Mykey then x->value = value
    else
        newLevel = randomLevel()
        newNode = makeNew(newlevel, Mykey, value)
        if newLevel > list->maxlevel then
            for i:=list->maxlevel upto newLevel do
                head->next[i] = newNode
        for i:=1 upto list->maxlevel do
            newNode->next[i] = vector[i]->next[i]
            vector[i]->next[i] = newNode
        list->maxlevel = newLevel
```

2 Time complexity

Before calculating time complexity, we first define $L(n) = log_{1/p}n$, which is the ideal level to start searching. This is obvious because the number of node at the $L(n)$ level is in single digits and the higher levels have almost no nodes in terms of probability.

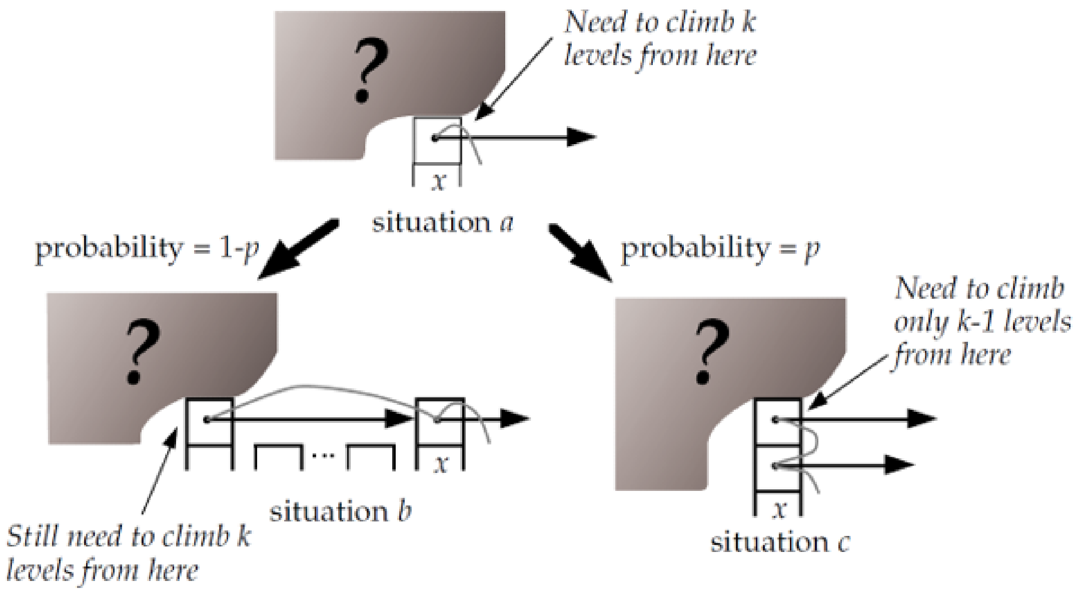

Need to climb k levels from here
situation a
probability = 1-p
probability = p
Need to climb only k-1 levels from here
Still need to climb k levels from here
situation b
situation c
x

Figure 1: Possible situations in backwards traversal of the search path

Then we solve the problem by going backwards on the path of searching, that is, from down to up, from right to left. At any point that has height smaller than $L(n)$, it may go either left or up when backtracking the searching path, with probabilities of 1-p or p. When we move from situation a to situation b, we just move left and still have k level to climb up to $L(n)$. At situation c, we climb up a level and need to climb only $k-1$ levels after. Assume the cost of climbing k nodes in an infinite list is $C(k)$, then:

$$C(0) = 0$$
$$C(k) = (1-p)(1+C(k)) + p(1+C(k-1))$$

By substituting and simplifying we get:

$$C(k) = k/p$$

When climbing from the first level (original list) to level $L(n)$, $k = L(n) - 1$, the cost is $(L(n) - 1)/p$. Since the expectation of amount of nodes on level $L(n)$ is $1/p$, the expectation cost for moving leftward on level $L(n)$ is $1/p$.

The total complexity of searching is the cost of climbing upward and leftward combined:

$$T = (L(n) - 1)/p + 1/p = L(n)/p = log_(1/p)n/p = O(logn)$$

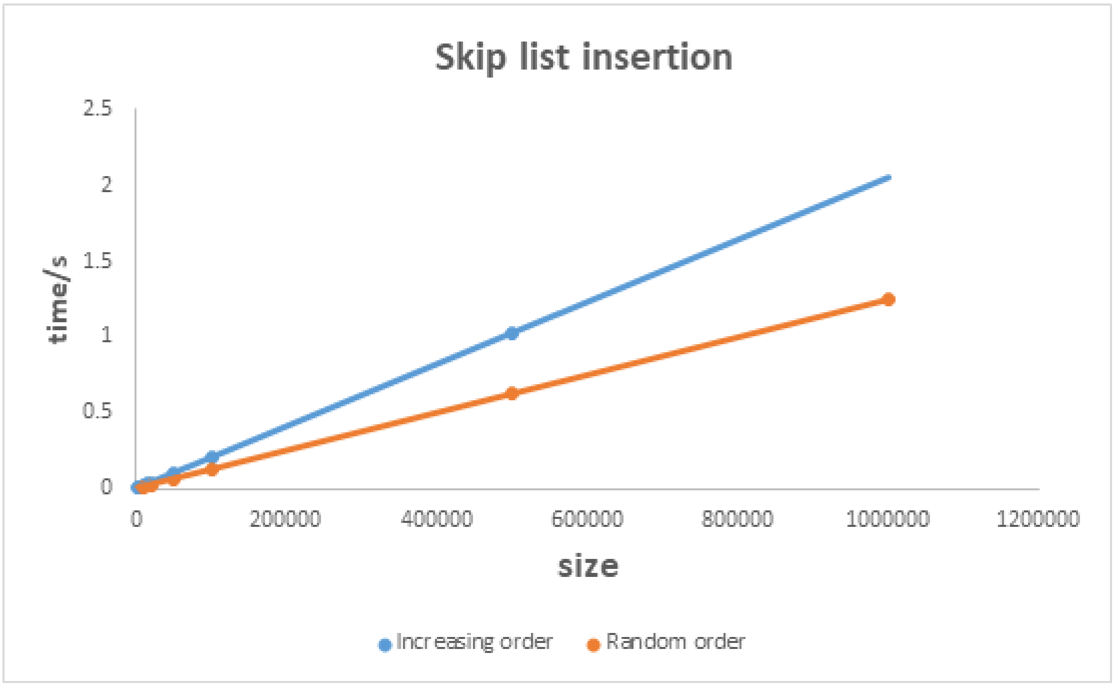
3 Testing

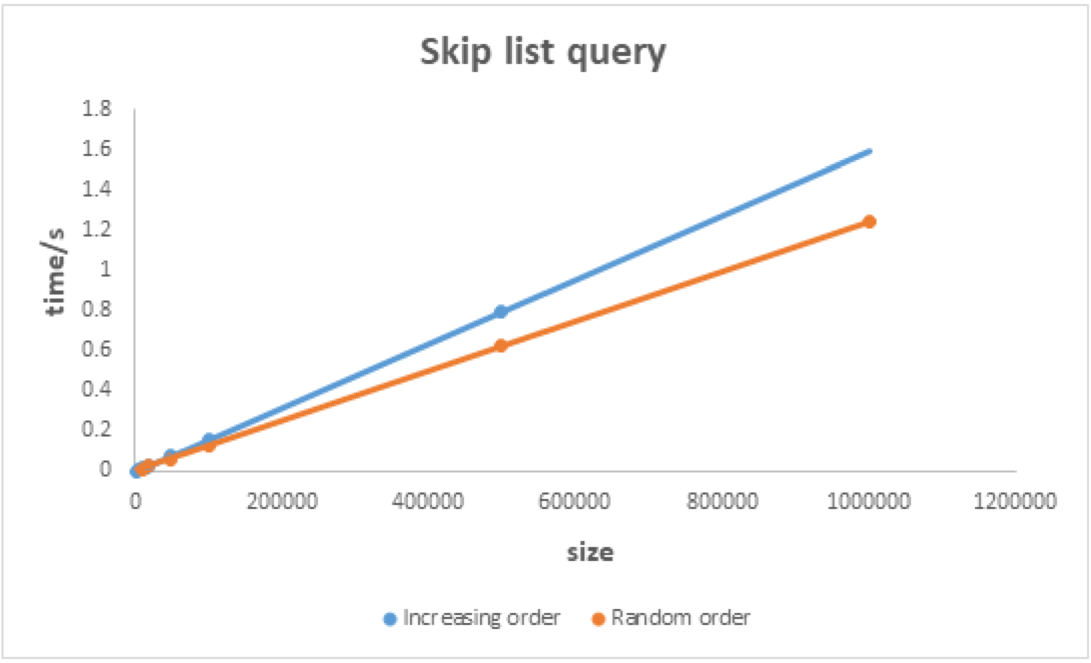
What we show are just parts of the possible result, and the results are influenced by many factors when program runs in PC, so we only analyze these results qualitatively.

We have two kinds of input data source. One kind of numbers is in ascending order, from 1 to the largest number. We average the loop five times to get the final run time. The other kind is not repeated random number. These random numbers through random shuffle function, and generated a total of 10 different random data files, is divided into five groups of each group of two, to insertion, query and deletion. Finally, the final time result is obtained by averaging five groups of different random number tests.

Our tests are roughly divided into three directions: data volume, index probability and maximum number of levels; two data orders: random number and ascending order, and three operations: insert, query and delete.

Skip list in Different data size(max size of levels:16,index probability:50%):





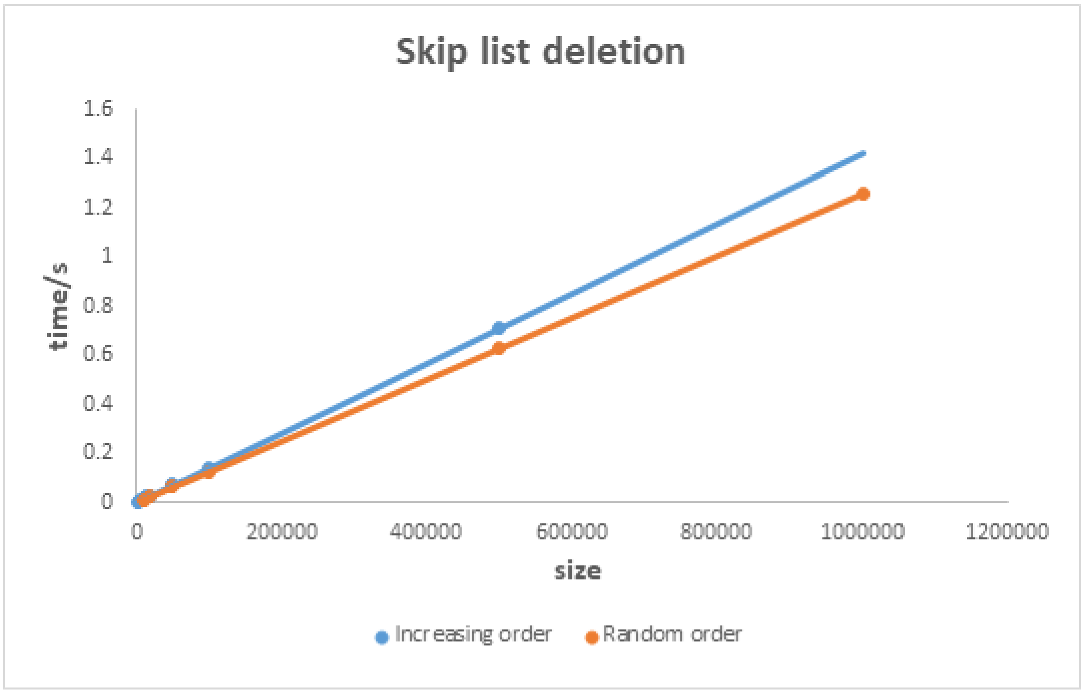
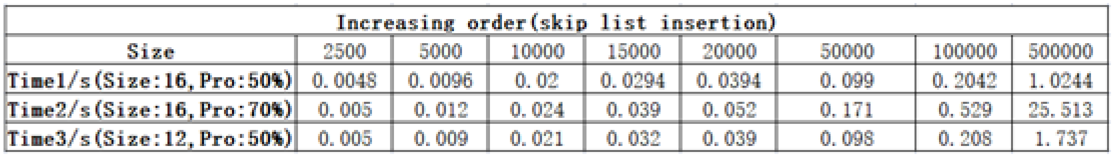


Figure 2:Skip list in Different data size(max size of levels:16,index probability:50%)

In this test,we set max size of levels as 16, index probability as 50% to study different running time of skip list in different size in increasing order and random order. The insertion, query and deletion time of random numbers are all shorter than that of ascending data, and there is a good linear relationship between the consumption time and the size of data.

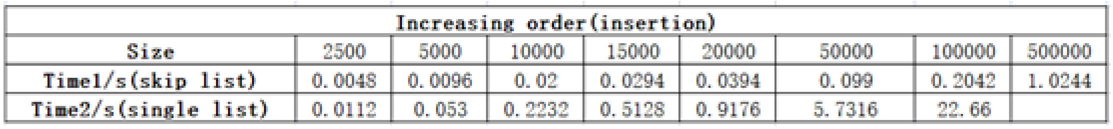
But when the maximum number of levels is reduced to 12 or the index probability is increased to 70%, there will be a jump in the insertion and search of the skip list in testing a large amount of data, while the deletion is basically a stable linear relationship (both ascending and random order show the same trend). The following takes the insertion in ascending order as an example:



Increasing order(skip list insertion)								
Size	2500	5000	10000	15000	20000	50000	100000	500000
Time1/s(Size:16,Pro:50%)	0.0048	0.0096	0.02	0.0294	0.0394	0.099	0.2042	1.0244
Time2/s(Size:16,Pro:70%)	0.005	0.012	0.024	0.039	0.052	0.171	0.529	25.513
Time3/s(Size:12,Pro:50%)	0.005	0.009	0.021	0.032	0.039	0.098	0.208	1.737

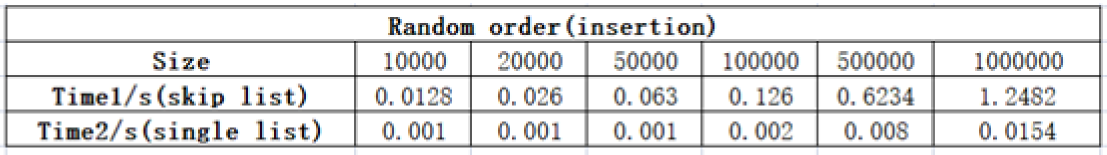
Comparison of skip list and single node list in different size

In this part, we set max size of levels as 16, index probability as 50% to study different running time of skip list and single node list in different size in increasing order and random order. In increasing order, skip list is faster than single node list in data insertion and query. The following takes the insertion in ascending order as an example:

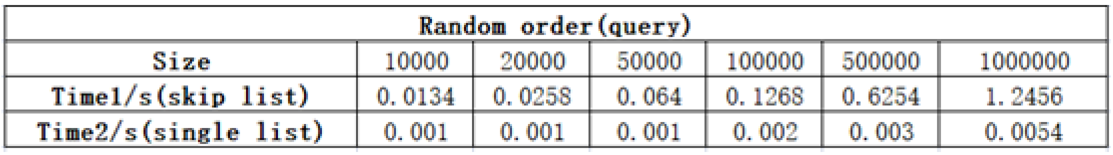


Increasing order(insertion)								
Size	2500	5000	10000	15000	20000	50000	100000	500000
Time1/s(skip list)	0.0048	0.0096	0.02	0.0294	0.0394	0.099	0.2042	1.0244
Time2/s(single list)	0.0112	0.053	0.2232	0.5128	0.9176	5.7316	22.66	

But in random order, we find another different result: single node list is faster than skip list in data insertion and query. The following takes the insertion and query in random order as an example:



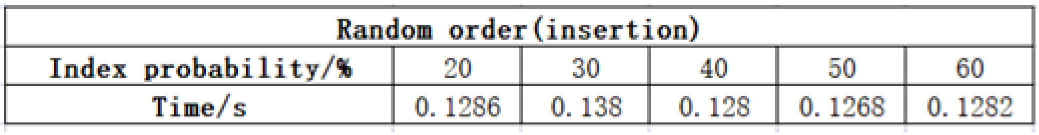
Random order(insertion)						
Size	10000	20000	50000	100000	500000	1000000
Time1/s(skip list)	0.0128	0.026	0.063	0.126	0.6234	1.2482
Time2/s(single list)	0.001	0.001	0.001	0.002	0.008	0.0154



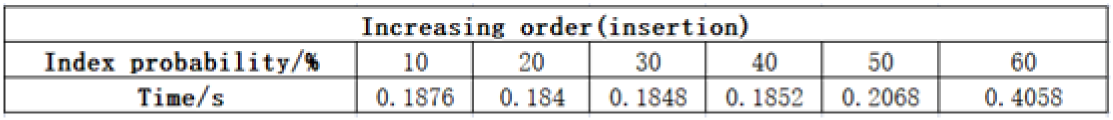
Random order(query)						
Size	10000	20000	50000	100000	500000	1000000
Time1/s(skip list)	0.0134	0.0258	0.064	0.1268	0.6254	1.2456
Time2/s(single list)	0.001	0.001	0.001	0.002	0.003	0.0054

Skip list in different index probability(data size:100000,max size of levels:12)

In this part, we set data size as 100000, max size of levels as 12 to study different running time of skip list in different index probability in increasing order and random order. In random order, running time of skip list is stable in different index probability. But in increasing order,the running time of the skip list is large at both ends and small in the middle, and the part with high probability is much larger. The following takes the insertion in random order and increasing order as an example (query is the same):



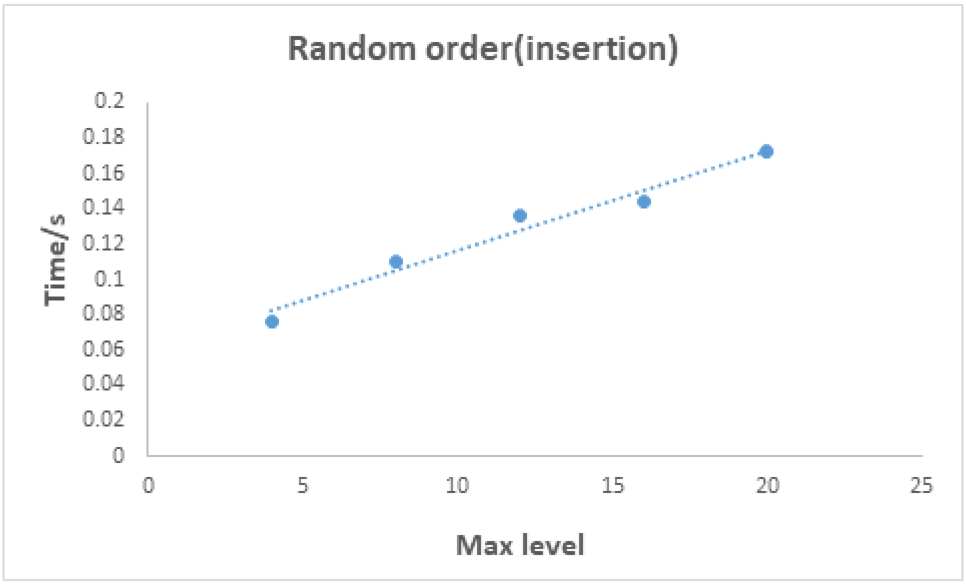
Random order(insertion)					
Index probability/%	20	30	40	50	60
Time/s	0.1286	0.138	0.128	0.1268	0.1282



Increasing order(insertion)						
Index probability/%	10	20	30	40	50	60
Time/s	0.1876	0.184	0.1848	0.1852	0.2068	0.4058

Skip list in different max size of levels(data size:100000,index probability:50%):

In this test, we set data size as 100000, index probability as 50% to study different running time of skip list in different max size of levels in increasing order and random order. In random order, running time of skip list slows down with increasing of the max number of levels and is linear. But in increasing order, the running time of the skip list is large at both ends and small in the middle. In the start, it's a big drop. But in the final, it's a small increase. The following takes the insertion in random order and increasing order as an example (query is the same):



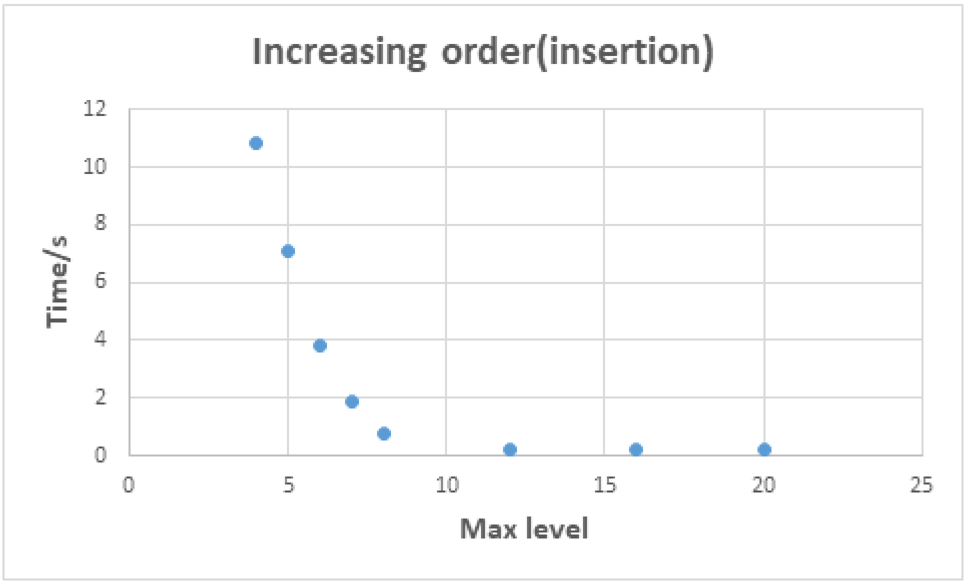


Figure 3:Skip list in different max size of levels(data size:100000,index probability:50%)

4 conclusion

After our testing and analyzing, we can find that skip lists have a very stable performance in suitable index probability and max level. The time complexity of its operation is all $O(log(N))$ and it is simple to implement a skip list so it can be an alternative to balanced tree in practical use.

REFERENCES:

[1] Pugh W . Skip lists: A probabilistic alternative to balanced trees[C]// Workshop on Algorithms and Data Structures. Springer, Berlin, Heidelberg, 1989.