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 skiplist has the following characteristics:

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

Chapter 2: Data Structure / Algorithm Specification

1. The data structure:

```
01. //every node consists of its value and pointers to next nodes in each level
02. struct Node {
03.     int value;//value should be positive integer
04.     //set value to be 0 when the node is empty
05.     struct Node* next[1];//pointers to next nodes in each level
06. };
07.
08. struct SkipList {
09.     int level;
10.     struct Node* top;//pointer to the head of skip list
11. };
```

2. 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:

```
initialize the empty skip list

insert 1 successfully
1
1

insert 2 successfully
1
1 2

insert 1 failed
1 is already in skip list
1
1 2

insert 3 successfully
1
1 2 3

insert 10 successfully
1 10
1 2 3 10

3 is in skip list

8 is not in skip list

delete failed
8 is not in skip list

delete 3 successfully
1 10
1 2 10
```

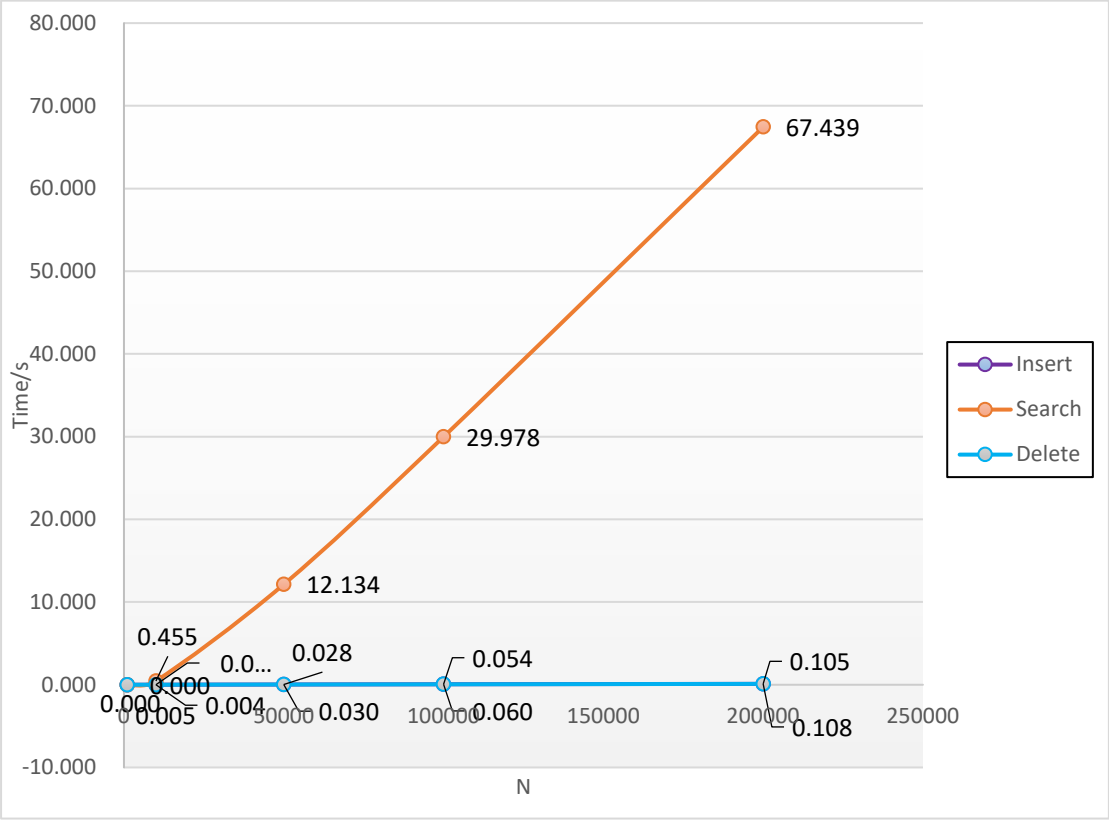
3.2 Time complexity of the program

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：

```
Please input the number of items inserted into the skiplist:200000
initialize the empty skip list
Time for inserting by second is 0.116000
Time for searching by second is 66.164000
Time for deleting by second is 0.100000
```

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(logn)
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)

```
#include<stdio.h>
#include<stdlib.h>
#include<vector>
#include<time.h>
using namespace std;

#define MAXLEVEL 10 //max level of skip list
clock_t start, stop;
double duration;

//every node consists of its value and pointers to next nodes in each level
struct Node {
    int value;//value should be positive integer
```

```
        //set value to be 0 when the node is empty
        struct Node* next[1];//pointers to next nodes in each level
};

struct SkipList {
        int level;
        struct Node* top;//pointer to the head of skip list
};
struct SkipList* skiplist;
struct Node* route[MAXLEVEL+1];//record the search route
struct Node* tmp1 = NULL;
struct Node* tmp2 = NULL;

//create a new node
struct Node* AddNode( int value, int level ) {
        struct Node* newnode
                = (struct Node*)malloc(sizeof(struct Node) + sizeof(struct Node*) * level);
        newnode->value = value;
        return newnode;
}

//initialize the empty skip list
void InitSkiplist() {
        printf("initialize the empty skip list\n");
        skiplist = (struct SkipList*)malloc(sizeof(struct SkipList));
        skiplist->level = 1;
        skiplist->top = AddNode(0, MAXLEVEL);
        int i;
        for (i = 1; i <= MAXLEVEL; i++) {
                skiplist->top->next[i] = NULL;
        }
}

//printf the whole skip list
void Print() {
        tmp1 = NULL;
```

```
        tmp2 = skiplist->top;
        int i;

        //print from the top level
        for (i = skiplist->level; i>=1; i--) {
                tmp2 = skiplist->top;
                while (1) {
                        tmp1 = tmp2->next[i];
                        if (tmp1 != NULL) {
                                //printf("%d ", tmp1->value);
                                tmp2 = tmp1;
                        } else {
                                break;
                        }
                }
        //      printf("\n");
        }
        //printf("\n");
}

//randomly get the level to insert
//prelevel is the present total level of skiplist
int Random(int prelevel) {
        int level = 1;

        //toss a coin
        while (rand()%2 && level<prelevel+1 && level<MAXLEVEL) {
                level++;
        }
        //update the total level
        if (level > prelevel) {
                int i;
                for (i = prelevel+1; i<= level; i++) {
                        route[i] = skiplist->top;
                }
                skiplist->level = level;
```

```
        }
        return level;
}

void GetRoute(int value) {
        tmp1 = NULL;
        tmp2 = skiplist->top;
        int i;

        //clear route[]
        for (i = skiplist->level; i>=1; i--) {
                route[i] = NULL;
        }
        //record the search route and find the place to insert
        for (i = skiplist->level; i>=1; i--) {
                while (1) {
                        tmp1 = tmp2->next[i];
                        if (tmp1 != NULL) {
                                if (tmp1->value < value) {
                                        tmp2 = tmp1;
                                } else {
                                        break;
                                }
                        } else {
                                break;
                        }
                }
                route[i] = tmp2;
        }
}

//insert a new num into skip list
void Insert(int value){
        int i;
        struct Node* tmp = NULL;
```

```
        //record the search route and find the place to insert
        GetRoute(value);
        //judge whether value is repeated
        if (tmp1 != NULL) {
                if (tmp1->value == value) {
                        //printf("%d is already in skip list\n", value);
                        //Print();
                        return;
                }
        }
        //randomly get the level to insert
        int insertlevel = Random(skiplist->level);
        //initial the new node
        tmp = AddNode(value, insertlevel);
        //insert and update every level
        for (i = 1; i <= insertlevel; i++) {
                tmp->next[i] = route[i]->next[i];
                route[i]->next[i] = tmp;
        }
        //Print();
}

//judge whether the value is in skip list or not
void Search (int value) {
        tmp1 = NULL;
        tmp2 = skiplist->top;
        int i;

        //search from the top level
        for (i = skiplist->level; i>=1; i--) {
                tmp2 = skiplist->top;
                while (1) {
                        tmp1 = tmp2->next[i];
                        if (tmp1 != NULL) {
                                if (tmp1->value <= value) {
                                        if (tmp1->value == value) {
```

```
                        //printf("%d is in skip list\n\n",value);
                        return;
                    }
                    tmp2 = tmp1;
                } else {
                    break;
                }
            } else {
                break;
            }
        }
        //printf("%d is not in skip list\n\n",value);
}

void Delete(int value) {
        //record the search route and find the place to insert
        GetRoute(value);
        int j;
        if (tmp1 &&tmp1->value == value) {
            //delete and update every level
            for (j = 1; j <= skiplist->level; j++) {
                if (route[j]->next[j] == tmp1) {
                    route[j]->next[j] = tmp1->next[j];
                }
            }
            free(tmp1);
            //update the total level
            for (j = skiplist->level; j>=1; j--) {
                if (skiplist->top->next[j] == NULL) {
                    skiplist->level--;
                }
            }
            //printf("delete %d successfully\n", value);
            //Print();
            return;
```

```
        }
        //printf("delete failed\n%d is not in skip list\n\n", value);
        return;
}

int main()
{
        long Num;
        int* Array;
        printf("Please input the number of items inserted into the skiplist:");
        scanf("%ld", &Num);
        Array = (int*)malloc(sizeof(int) * Num);

        InitSkiplist();
        srand((unsigned)time(NULL));

        //test for insert
       start = clock();
       for(int i = 0; i < Num; ++i){
                Array[i] = rand();
                Insert(Array[i]);
       }
        stop = clock();
        duration = ((double)(stop - start))/CLK_TCK;
        printf("Time for inserting by second is %lf\n", duration);

        //test for search
       start = clock();
       for(int i = 0; i < Num; ++i){
                Search(Array[rand() % Num]);
       }
        stop = clock();
        duration = ((double)(stop - start))/CLK_TCK;
        printf("Time for searching by second is %lf\n", duration);

        //test for delete
```

```
    start = clock();
    for(int i = 0; i < Num; ++i){
            Delete(Array[rand() % Num]);
    }
     stop = clock();
     duration = ((double)(stop - start))/CLK_TCK;
     printf("Time for deleting by second is %lf\n", duration);

    return 0;
}
```

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 " Skip Lists " is of our independent effort as a group.

Signatures

张雯琪

陈咏琛

吴沁珒