

“数据结构”

课程设计报告

**设计题目**  Skip List的实现

**姓 名**  袁焕发

**学 号**  2019217769

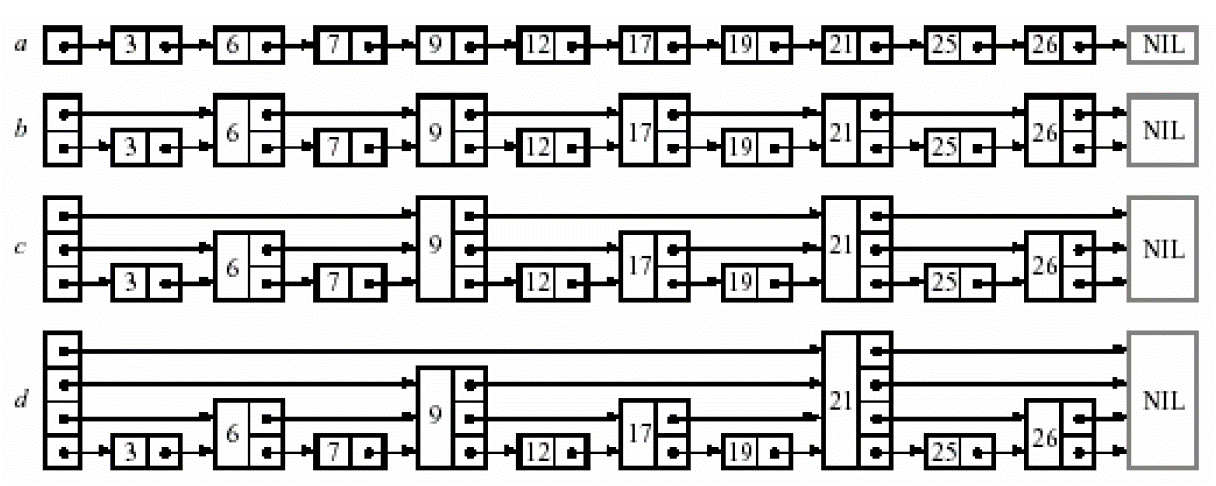
**专 业**  物联网工程

**班 级**  19-2班

**完成日期**  2020年06月23日

### （一）、需求和规格说明

### （1）SkipList作为有序链表结构的一种扩展，如下图所示，其中a是普通的单链表；而b是在次基础上加上第二层（level 2）的额外指针，这些额外的指针指向间隔为2的下一个结点，skip list 因此得名；类似的c是加上level 3后的 skip list；d 是加上level 4 后的 skip list。



Skip List 上查找的基本思想是先从最高的 Level 层上查找，找到 key 所在

的范围后，再从较低的层次继续重复查找操作，直到 Level 1。

Skip List 上的插入操作如下图所示。Skip List 上的删除操作只需直接删除元素即可（包括局部范围内的指针调

整）。

本设计题目的基本内容是构造并实现 Skip List 的 ADT，并能对其维护动态

数据集合的效率进行一定的实验验证。

**（2）课程设计目的**

认识并应用 Skip List 数据结构，体会线性表结构的变形形式。

**（3）基本要求**

①ADT 中应包括初始化、查找、插入、删除等基本操作。

②分析各基本操作的时间复杂性。

③针对实现 Skip List 上基本操作的动态演示（图形演示）。

④能对 Skip List 维护动态数据集合的效率进行实验验证，获得一定量的实验数据，如给定随机产生 1000 个数据并将其初始化为严格 Skip List，在此基础上进行一些列插入、删除、查找操作（操作序列也可以随机生成），获得各种操作的平均时间（或统计其基本操作个数）；获得各操作执行时间的变化情况，应该是越来越大，当大到一定程度后应该进行适当的整理，需设计相应的整理算法，并从数量上确定何时较为合适；能和其他简单线性数据结构，如排序数组的折半查找进行各类操作效率上的数量对比。

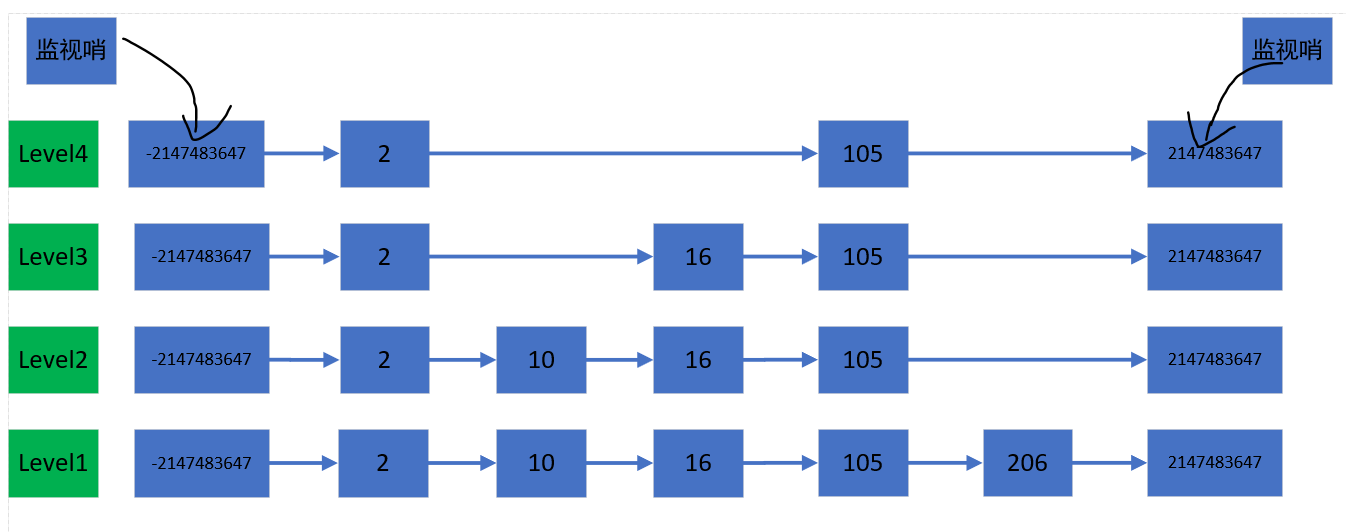
**（4）实现提示**

需仔细设计整理算法。

### (二)、设计

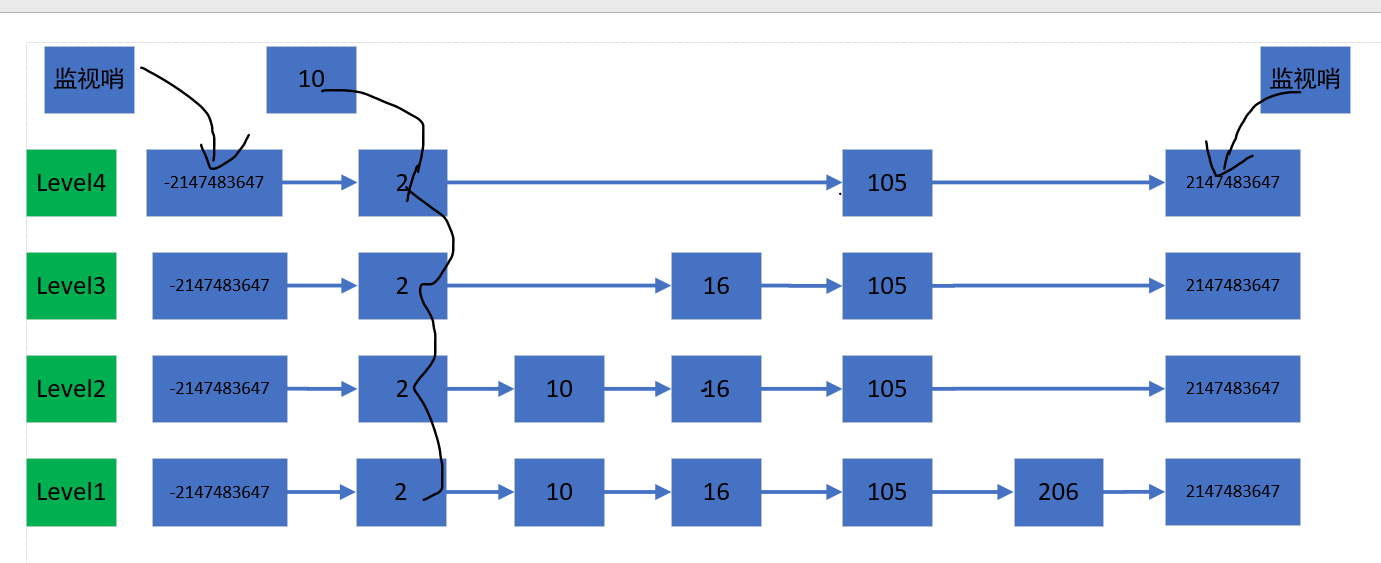
### 设计思想

根据跳跃链表的构成，可以看做是几层单链表的组合，每层之间又有联系，在原有单链表的基础上，在结构体中增加指针数组，使得原本的指针可以指向后继以及跳跃链表中的下一层链表。利用随机数，随机出某个元素占有的层数level，利用类似二分查找的方法，不断比较关键字key来确定元素插入，删除以及查找的位置。假设跳跃链表最大level为4，构建一个跳跃链表存入数据2,10,16,105,206。



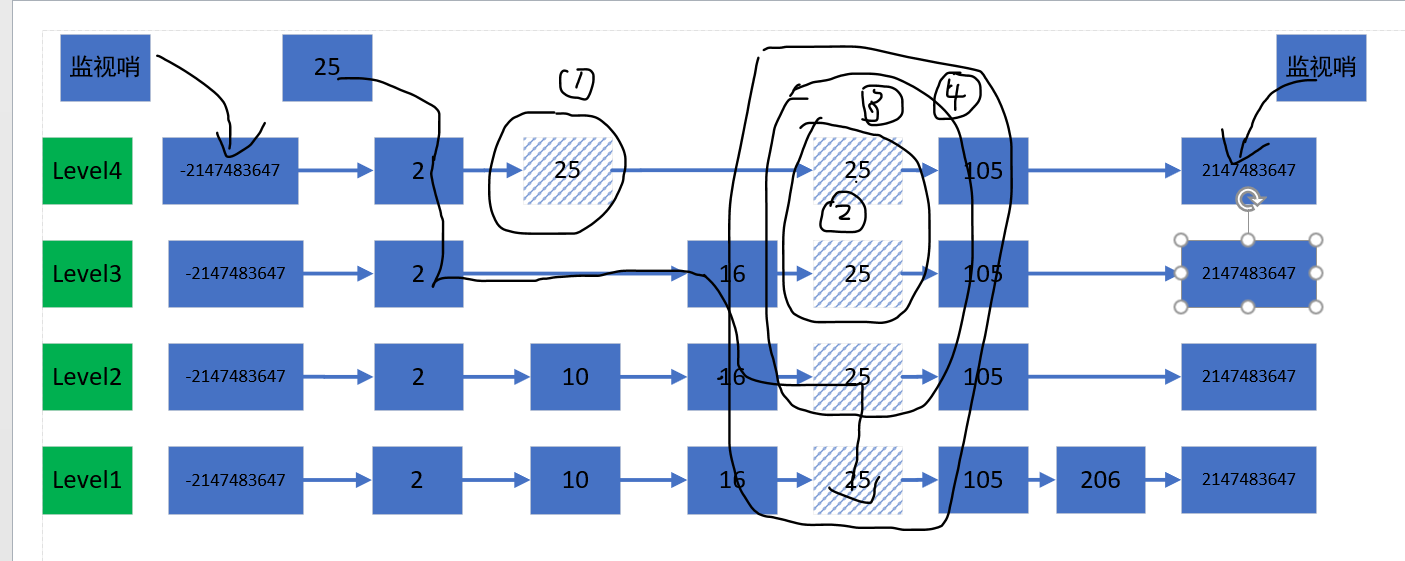
1. **查找数据**

假设查找数据10，在level4层比较10与2的大小，10大于2，head指针后移到数据2处，继续比较10与指针指向的下一个数105的大小，10小于105，下移指针到level3的数据2处, 10与16比较大小，10小于16，下移指针到level2的数据2处,10与10较大小，10不大于10，继续下移指针到level1的数据2处，判断10是否等于指针指向的下一个元素，若等于则查找成功，如果不等于，则没有此元素。后面的插入，删除都是基于此操作。



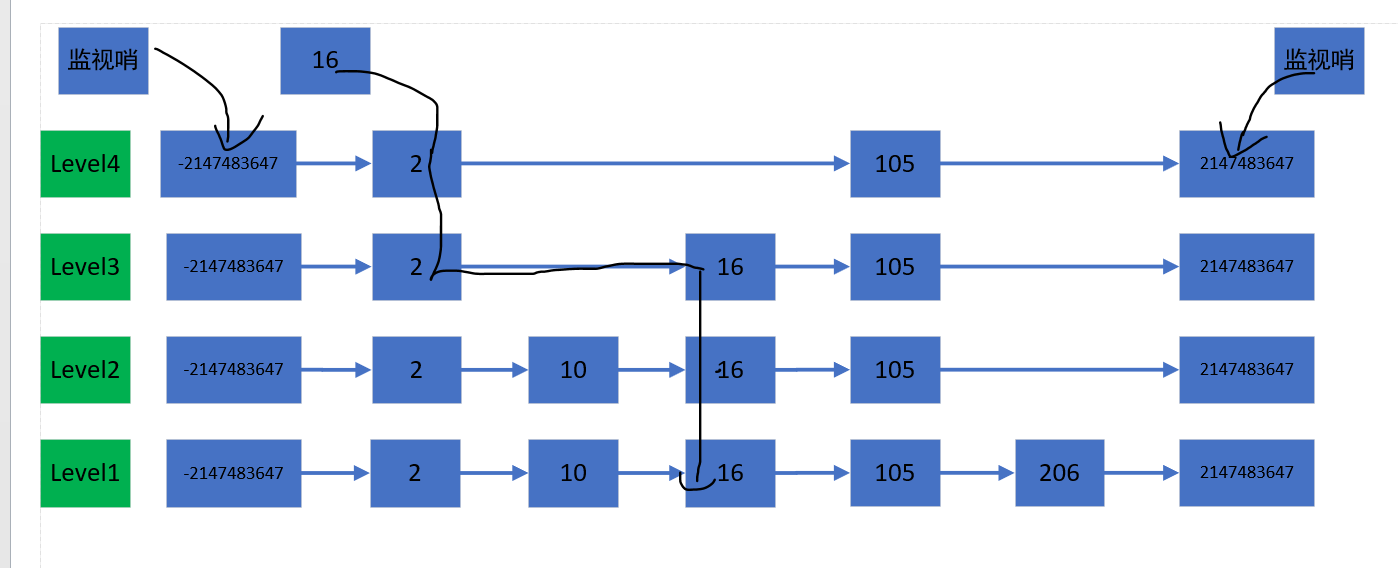
1. **插入数据**

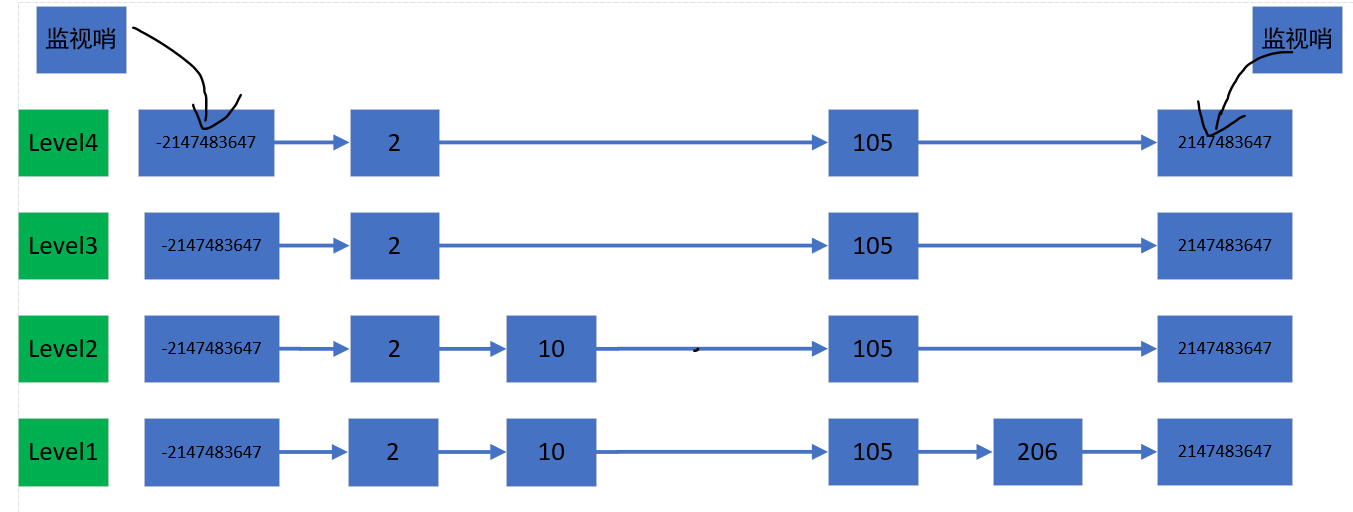
假设插入数据25，层数为4，level4中25与2比较，25比2大，head指向数据2，继续比较25与105大小，比105小，在该层指针之后插入25，下移一层到level3；继续比较指针指向的下一个元素16，25比16大，指针后移指向数据16，继续比较25与指针指向下一个数据105的大小，25比105小，该层指针后插入元素25，下移一层到level2；继续比较25与指针指向的下一个元素105的大小，25比105小，该层指针后插入元素25，下移一层到level1；25与105比较大小，25小于105，该层指针后插入元素25，这时找到了插入点在105前面。由于指针的移动，使得插入的整列数据25，都放置在了合适的位置。



1. **删除数据**

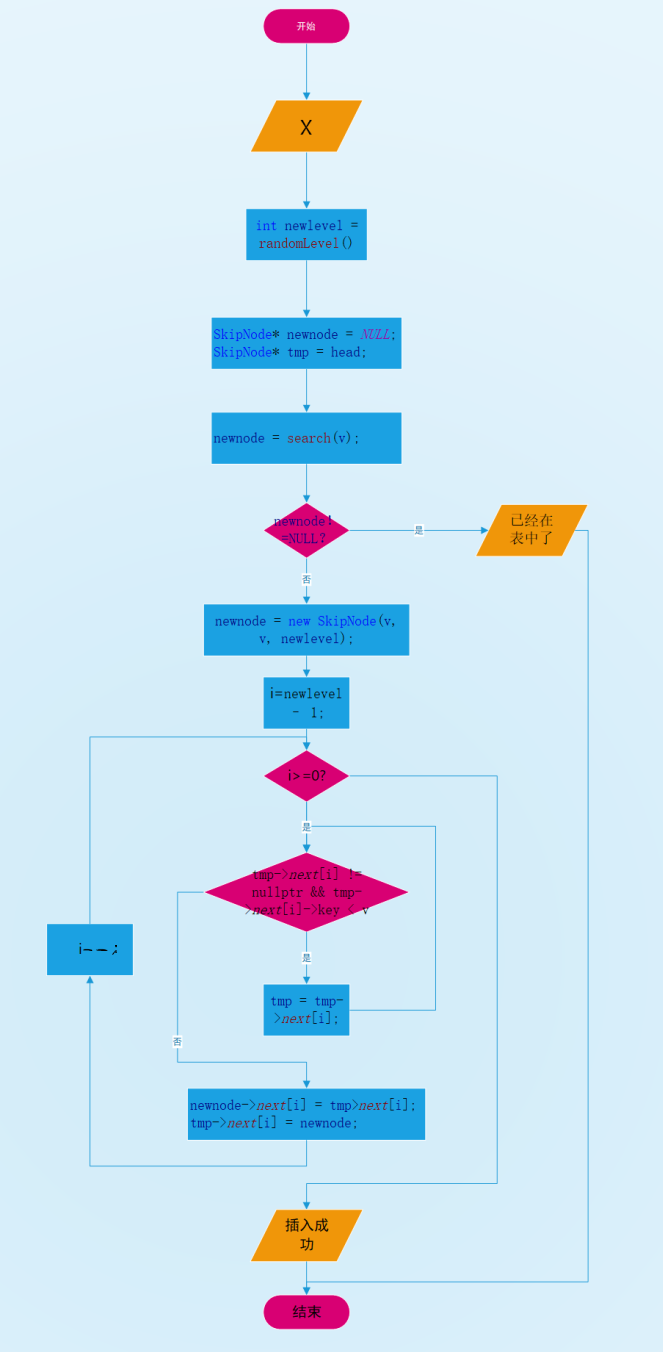
假设删除数据16，按照查找的方法查找到最高层16出现的位置，然后保留其上一个数据的指针数组，从上到下删除整列，然后将上一列元素的指针指向被删除元素的后面一个元素



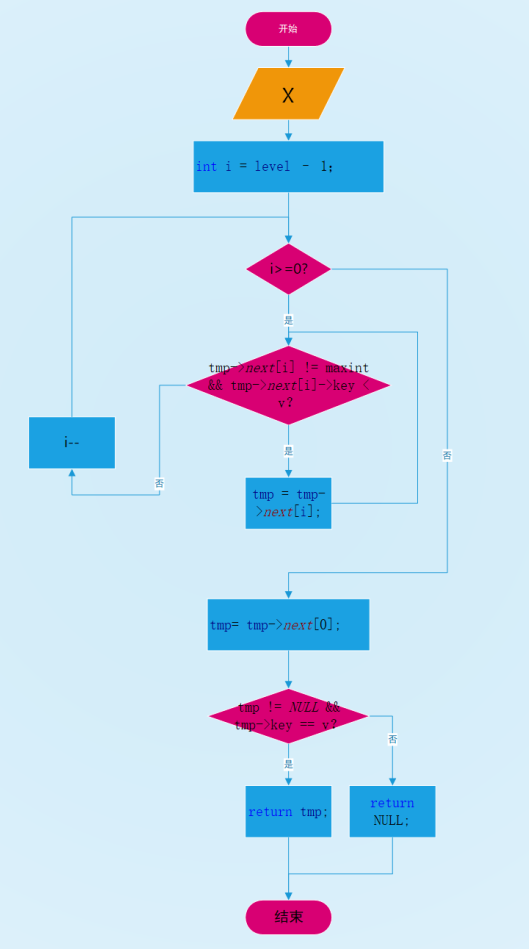


### （2）程序框图

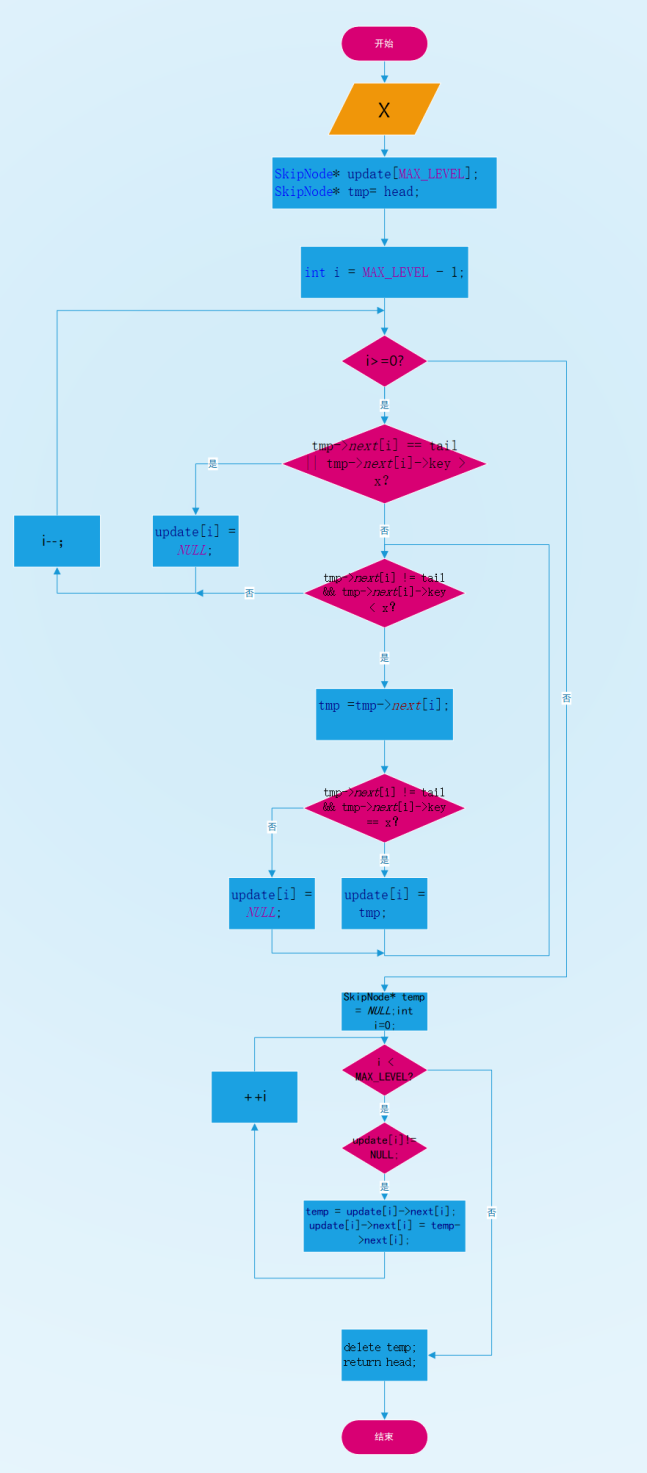
1）插入流程



2）查找流程



3）删除流程



### （三）、使用手册

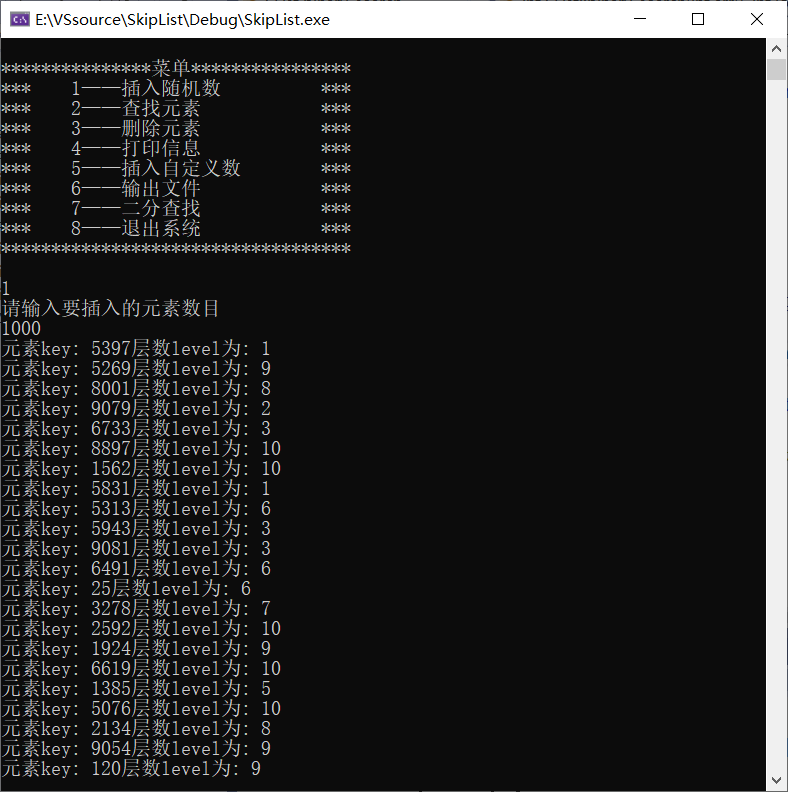
根据给出的提示输入相应的序号进行操作演示。

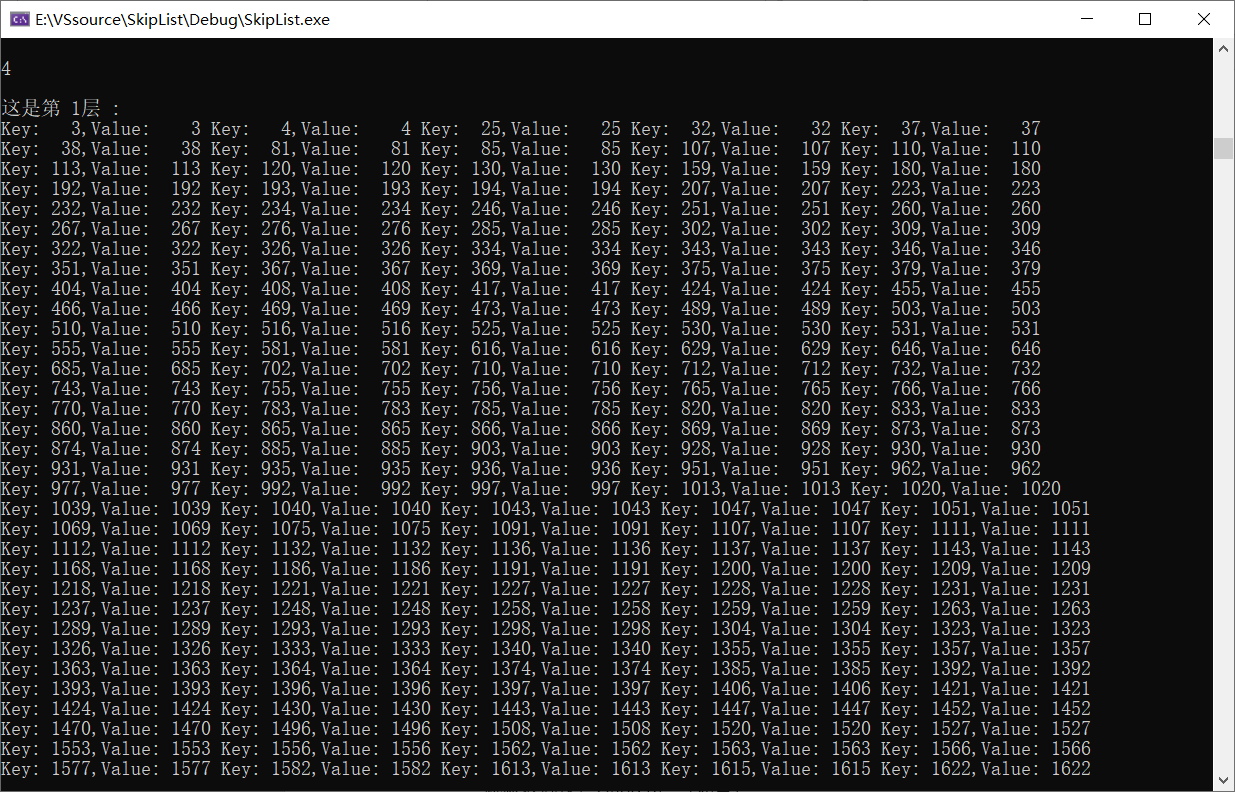
### （四）、调试与测试

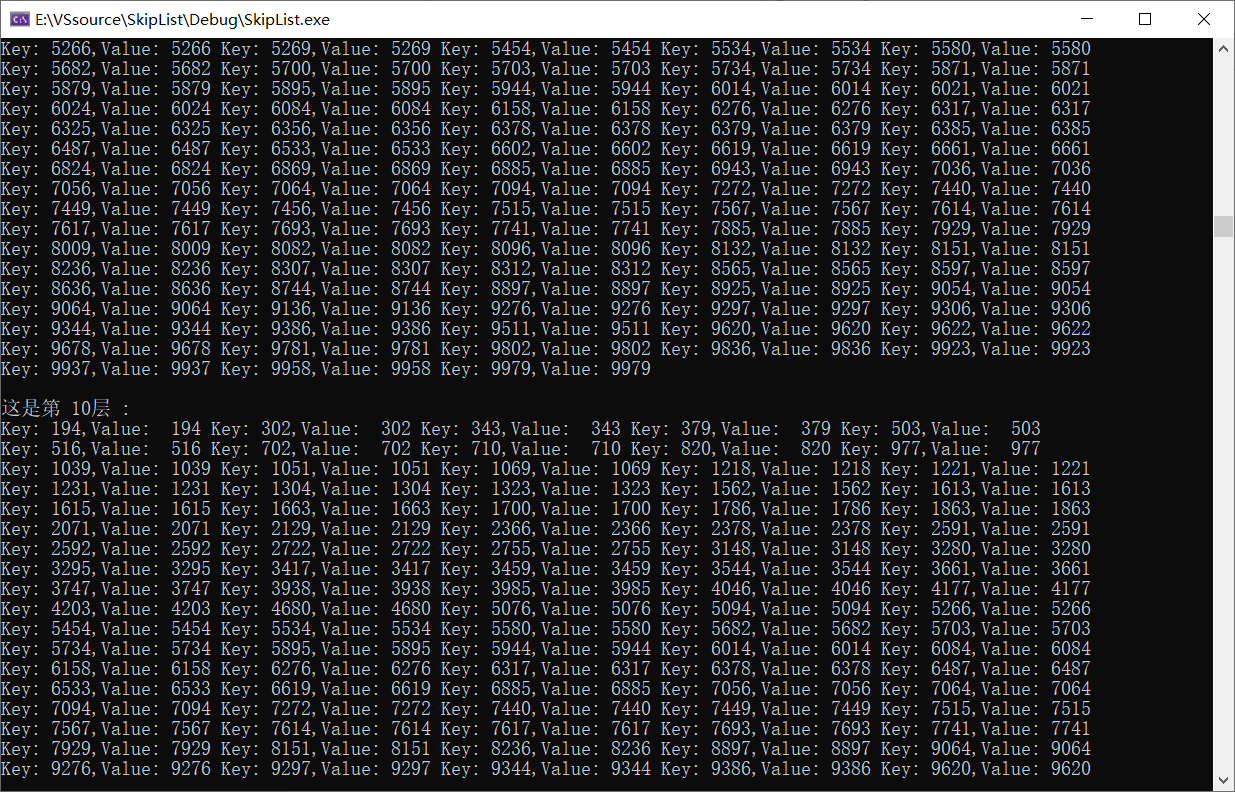
由于程序运行时间短暂，运用时间种子获得的随机数的值会相同，所以对于level的生成没有采用抛硬币方式，而是一次随机生成level，随机数函数参考来C++11random库随机数函数。

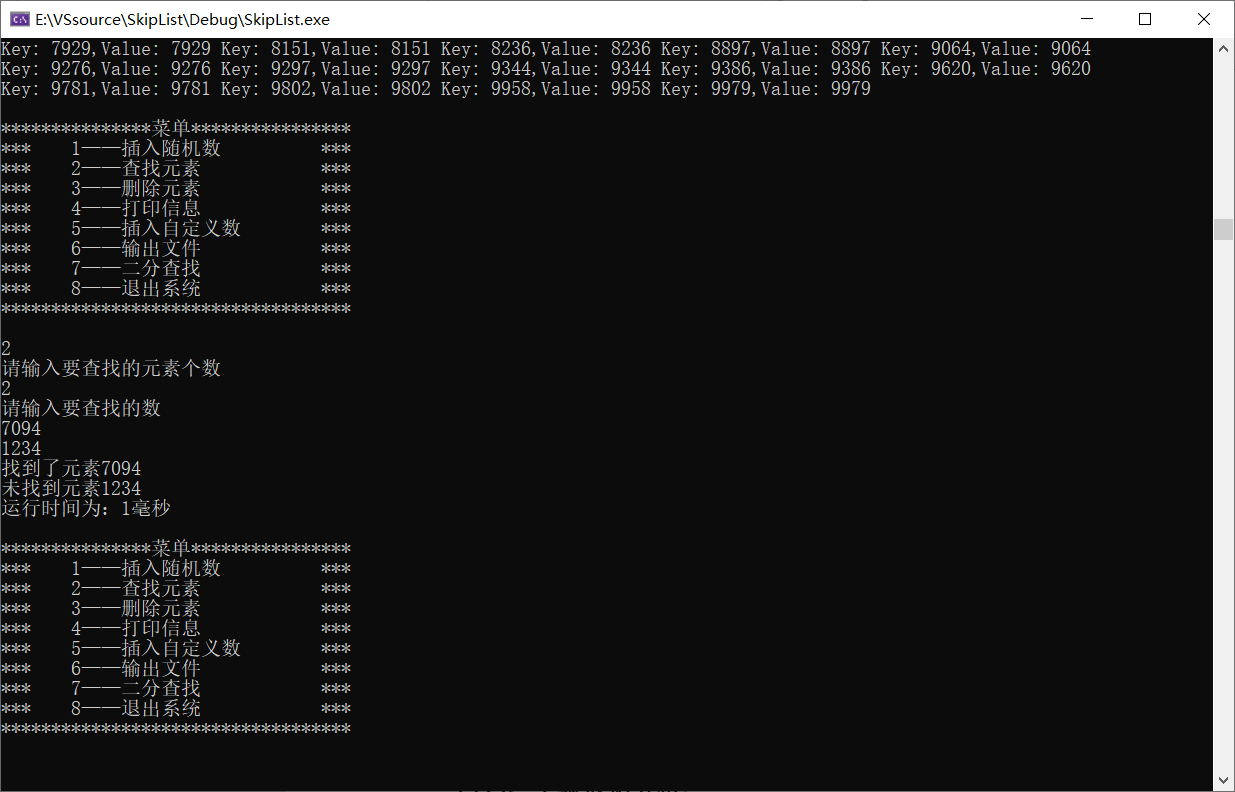
### （五）、运行实例

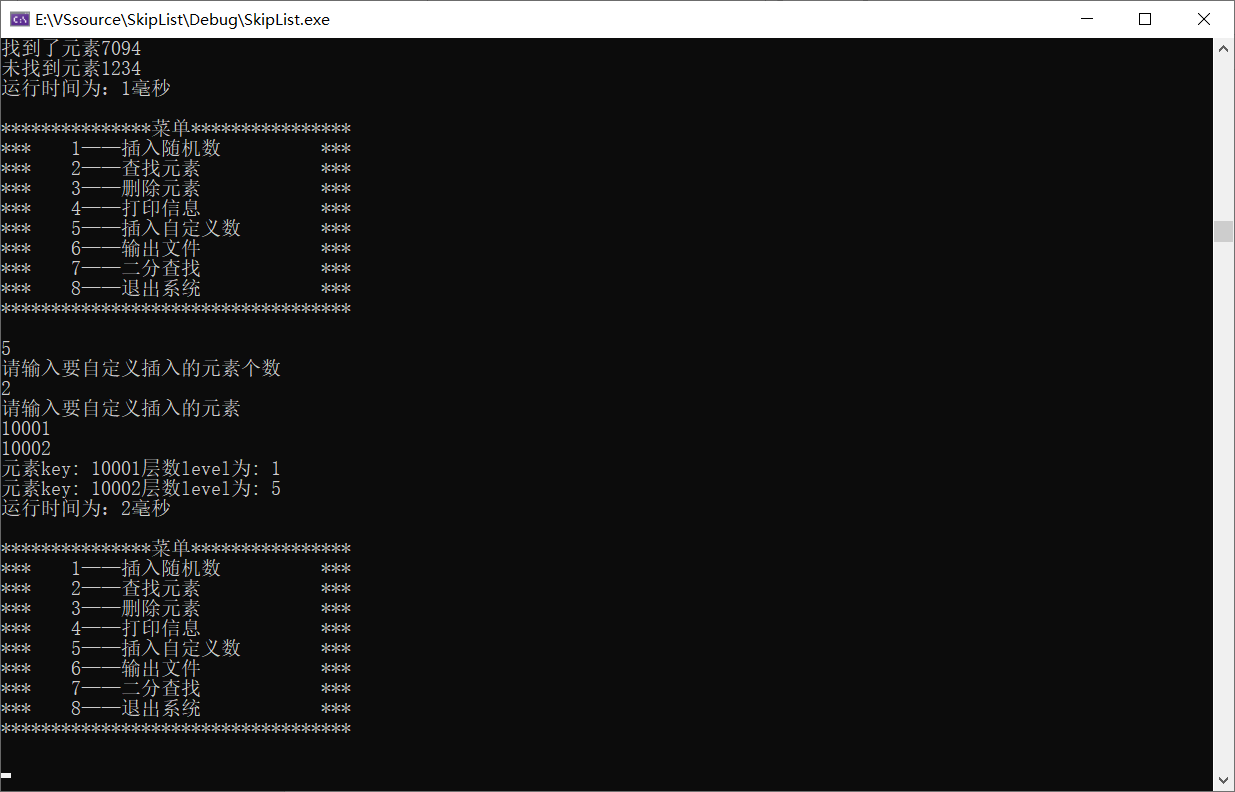
由于数据太大，部分数据略

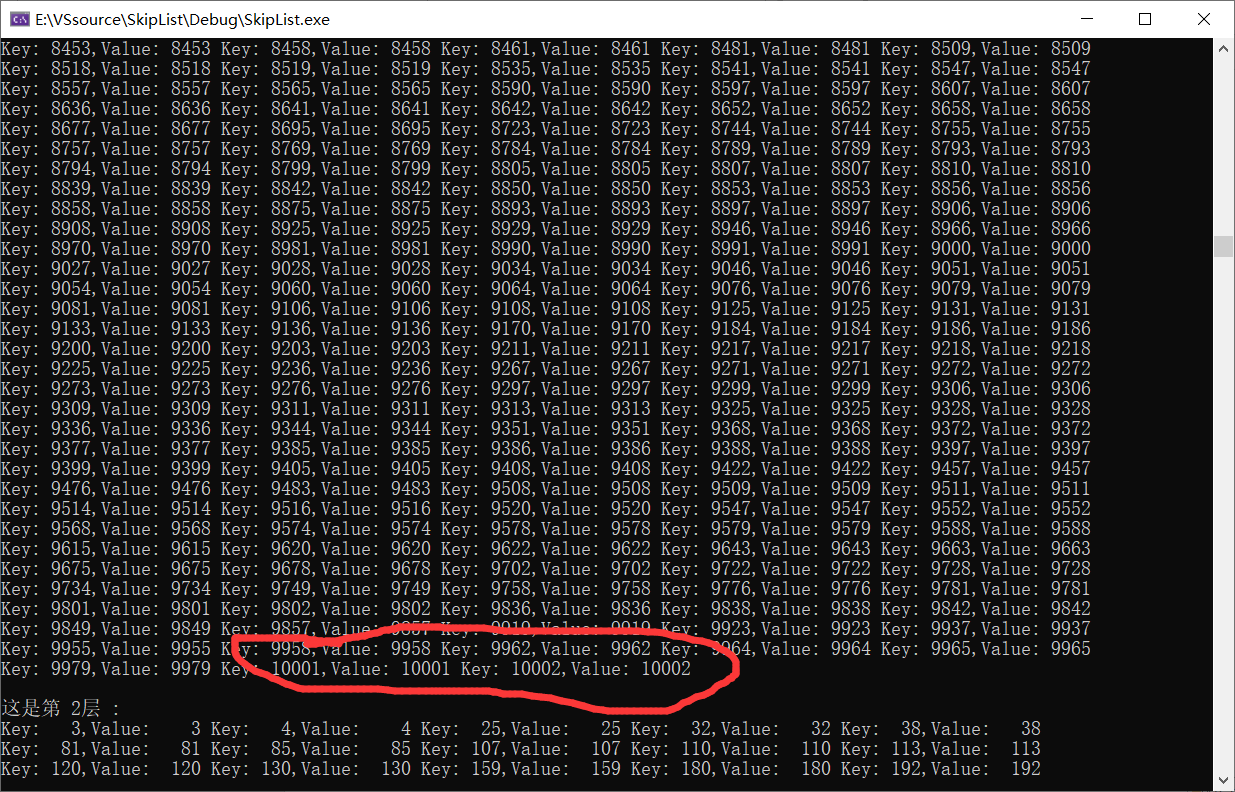


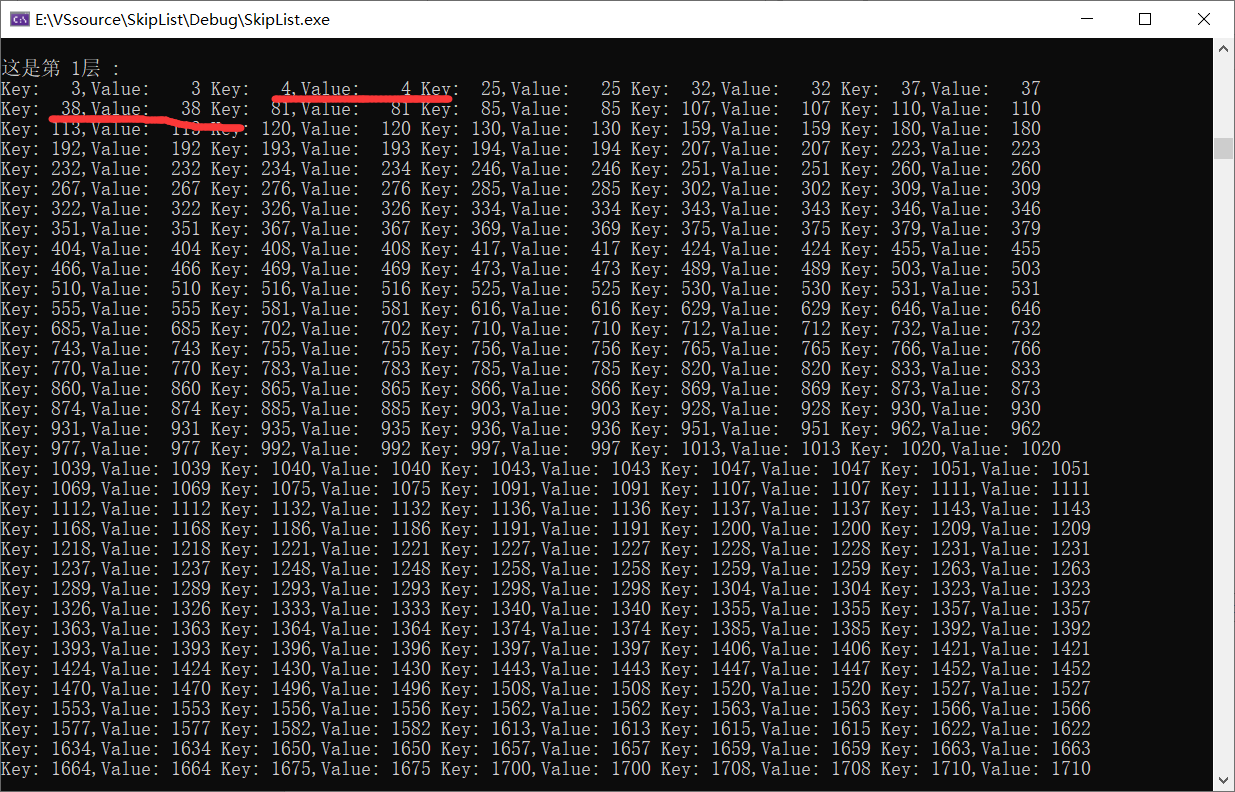


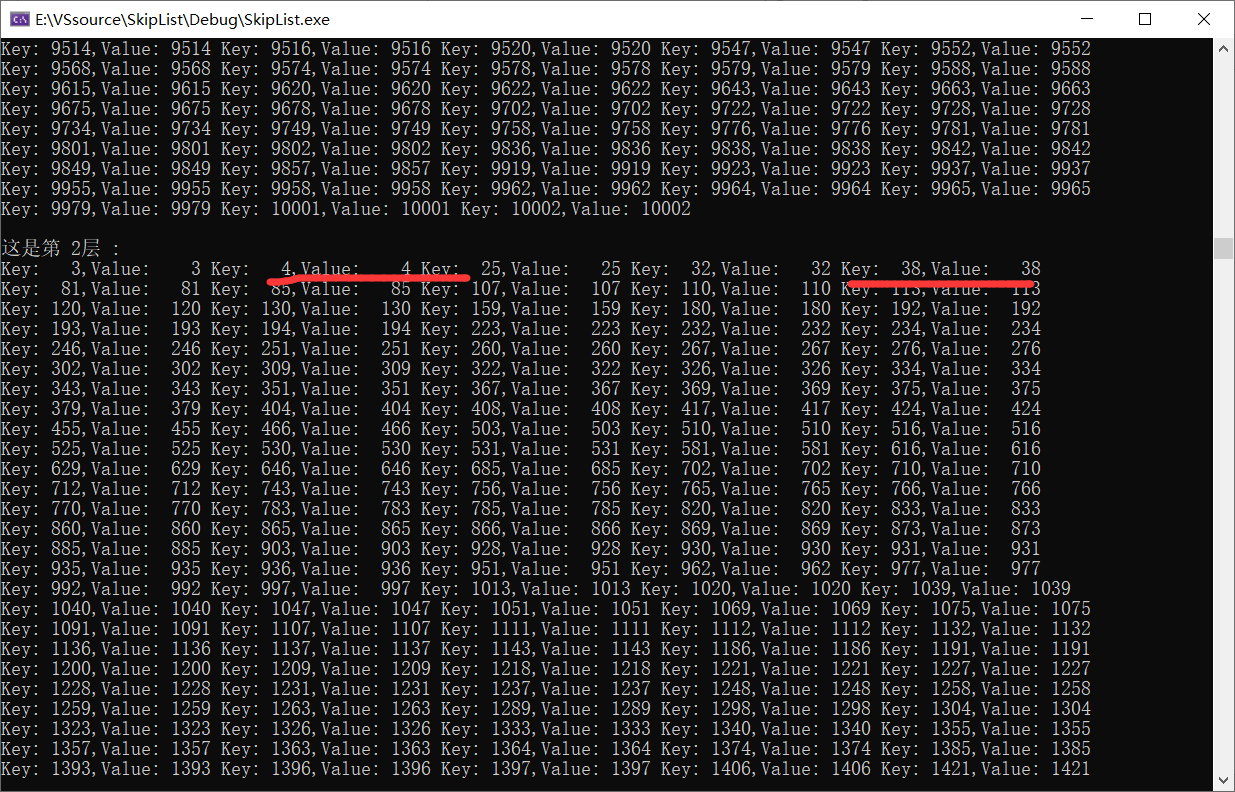


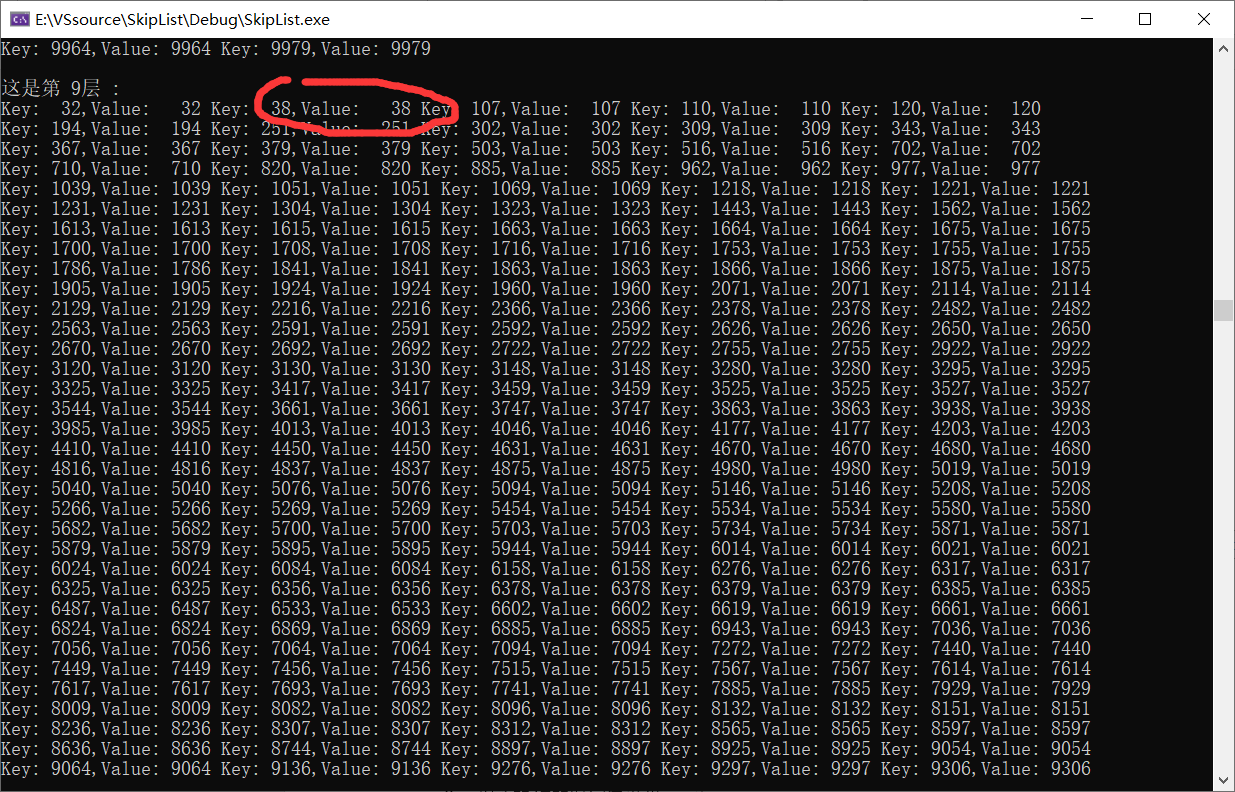


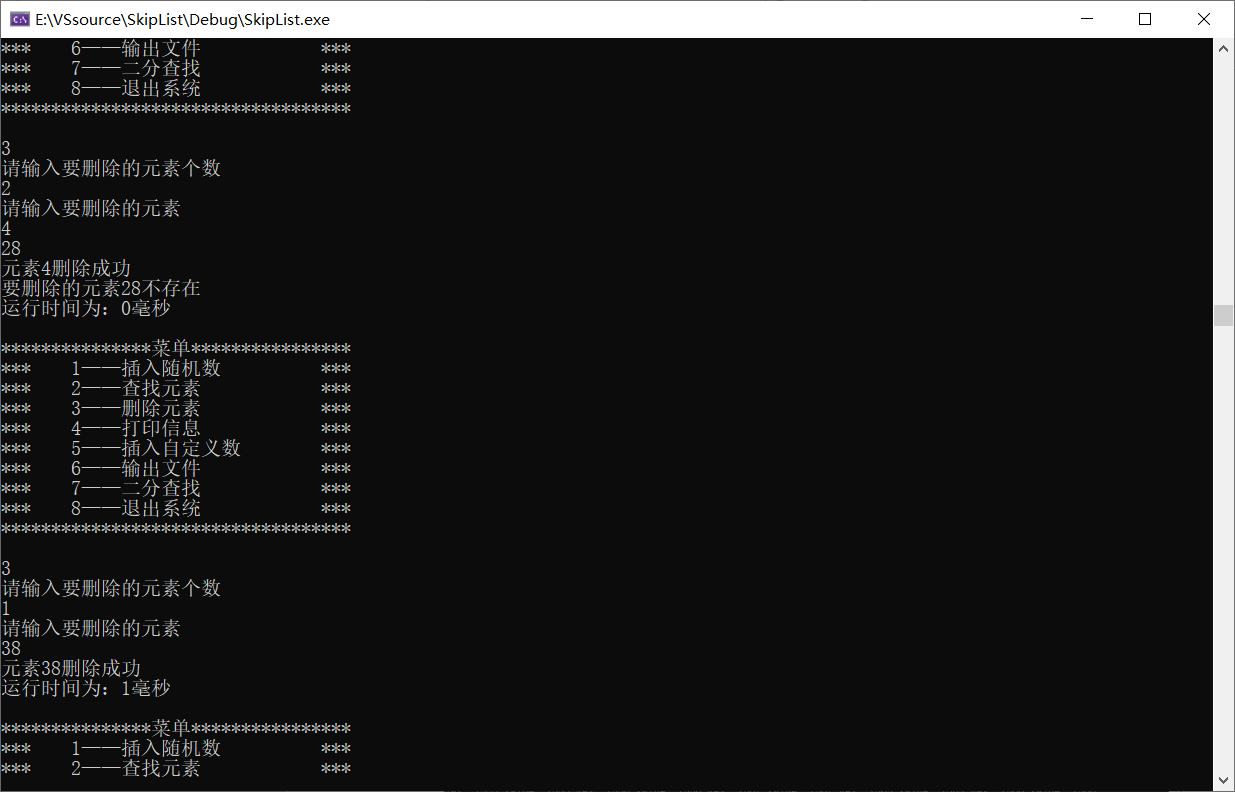


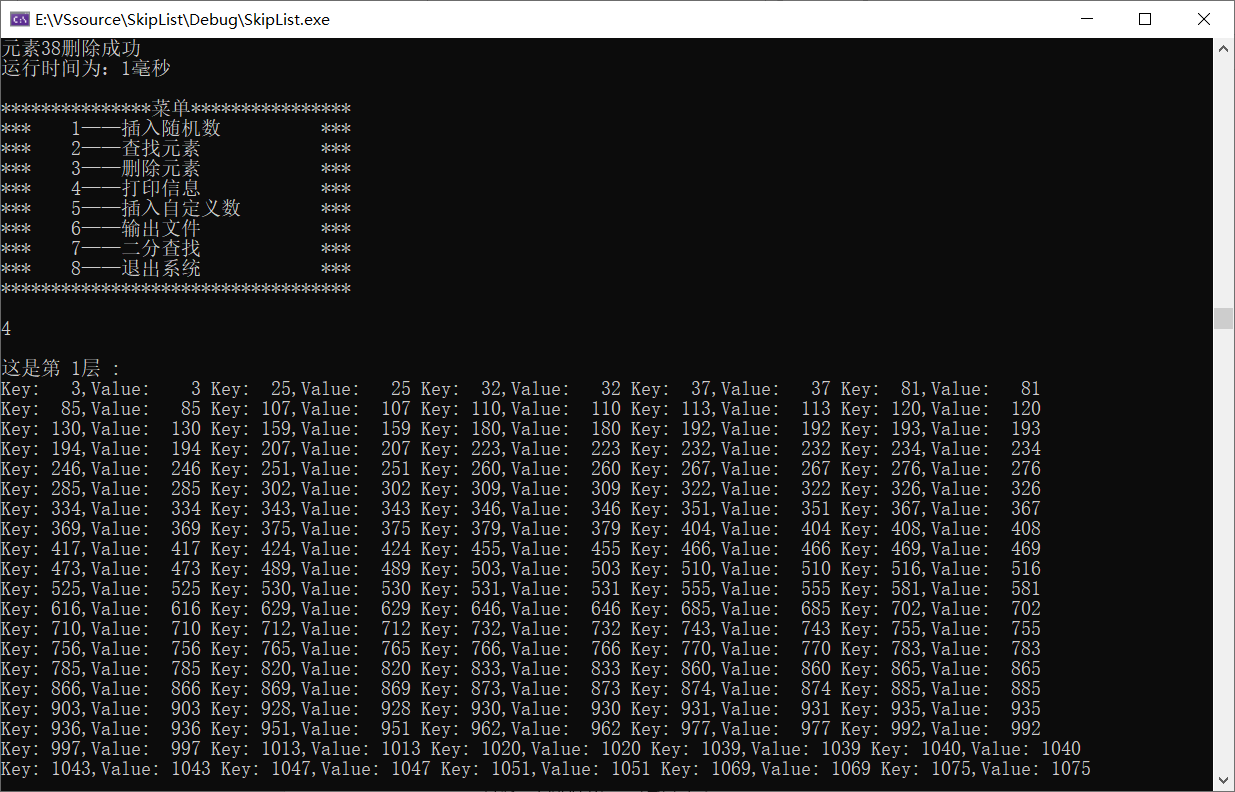


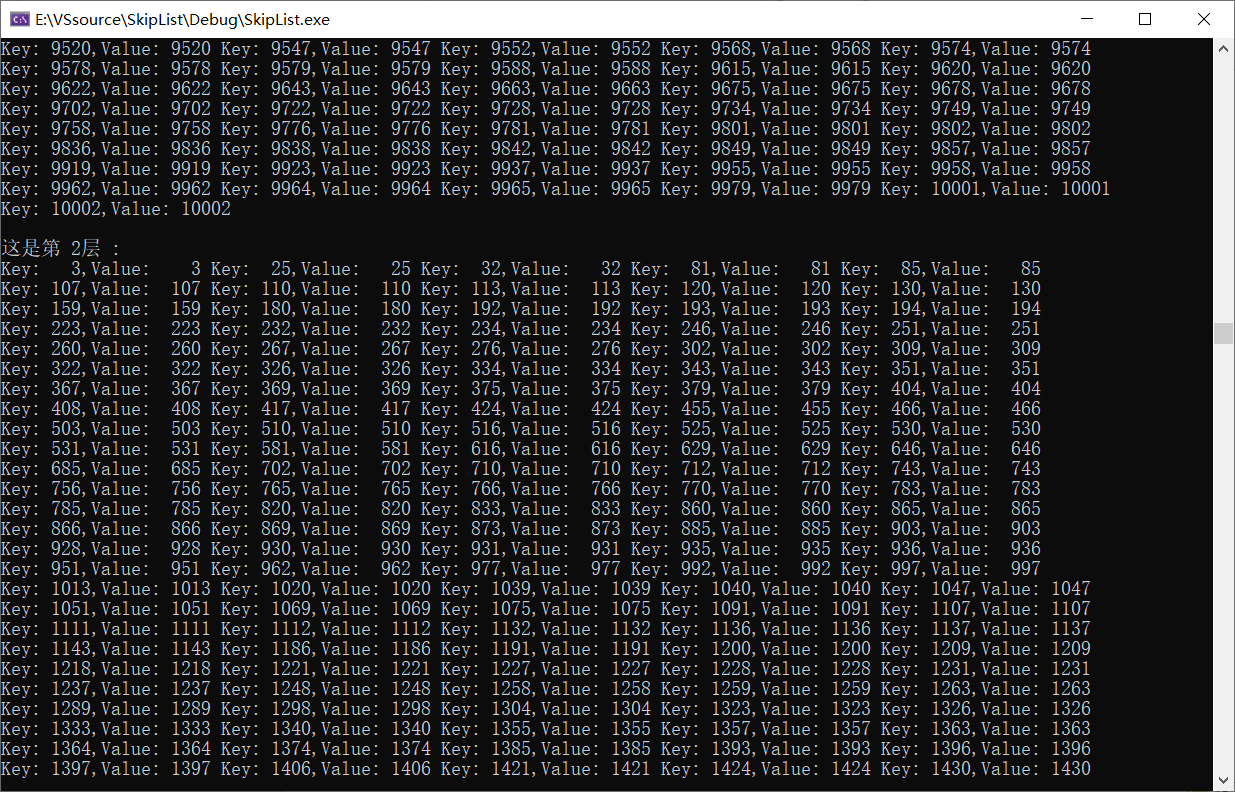


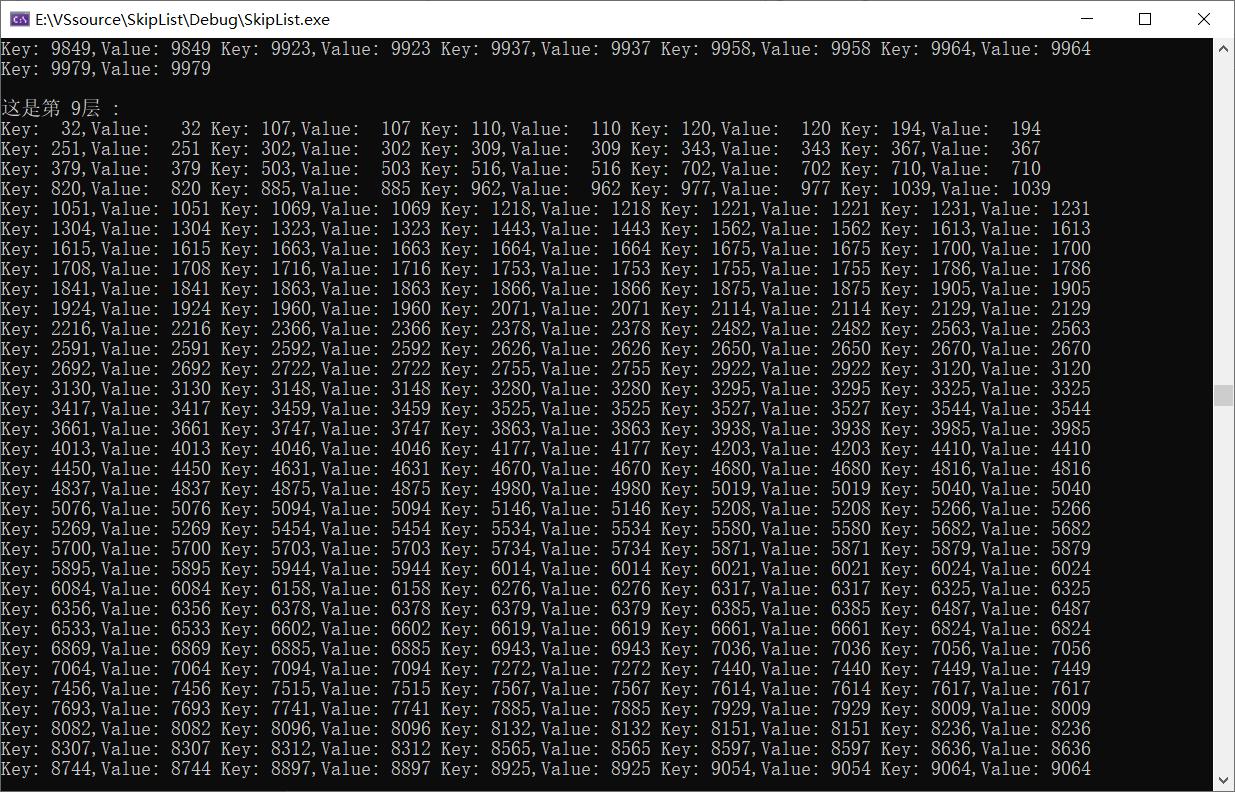


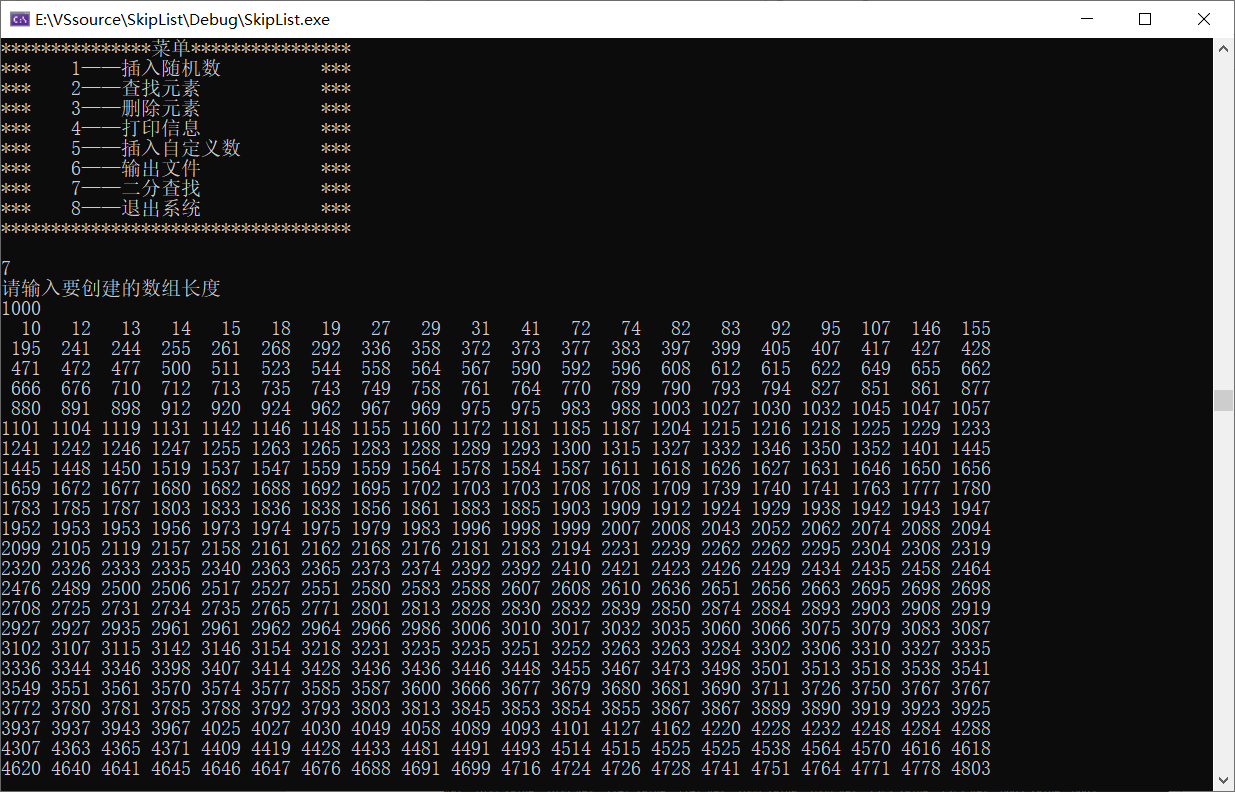


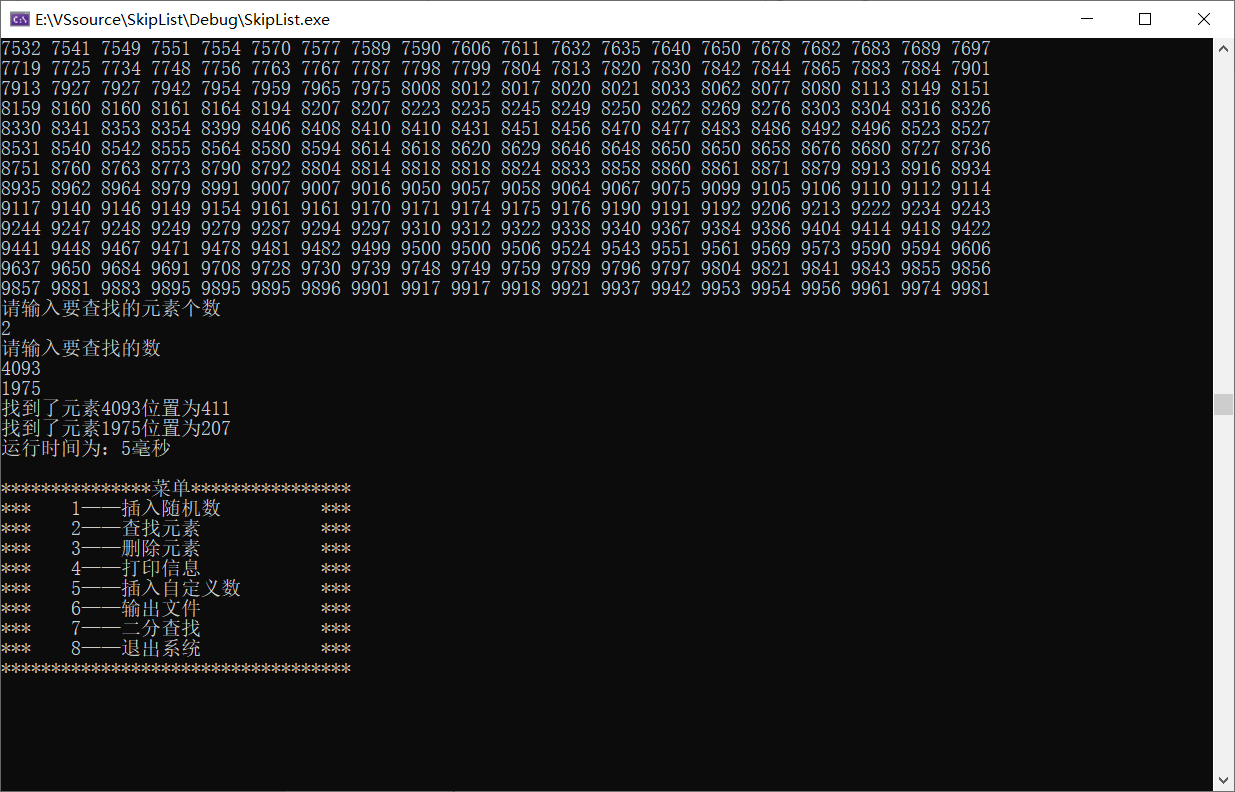












### （六）、实验数据分析

1. 跳跃表的相关复杂度：

       空间复杂度：O(n)

       跳跃表高度： h = log2n

1. 相关操作的时间复杂度：

查找：O(log n)    根据跳表查找时的规律，每层最多遍历3三个数据，查找次数为

3\* log2n，所以时间复杂度为O(log n)

      插入： O(log n)    类似查找操作

      删除： O(log n)   类似查找操作

1. 操作时间统计表

表1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数据量    时间  操作 毫秒 | 10 | 100 | 300 | 600 | 1000 |
| 查询一个数 | 2 | 2 | 1 | 1 | 2 |
| 查询一个数 | 1 | 2 | 1 | 2 | 1 |
| 查询一个数 | 2 | 2 | 3 | 1 | 1 |
| 平均时间 | 1.67 | 2 | 1.67 | 1.33 | 1.33 |
| 查询两个数 | 5 | 4 | 2 | 2 | 2 |
| 查询两个数 | 2 | 2 | 4 | 1 | 3 |
| 查询两个数 | 4 | 3 | 3 | 3 | 2 |
| 平均时间 | 3.67 | 3 | 3 | 2 | 2.33 |
| 插入一个数 | 2 | 1 | 2 | 1 | 2 |
| 插入一个数 | 2 | 2 | 1 | 2 | 1 |
| 插入一个数 | 1 | 2 | 2 | 3 | 1 |
| 平均时间 | 1.67 | 1.67 | 1.67 | 2 | 1.33 |
| 插入两个数 | 2 | 2 | 2 | 2 | 1 |
| 插入两个数 | 5 | 4 | 5 | 4 | 1 |
| 插入两个数 | 3 | 2 | 5 | 2 | 3 |
| 平均时间 | 3.33 | 2.67 | 4 | 2.67 | 1.67 |
| 删除一个数 | 1 | 4 | 3 | 2 | 2 |
| 删除一个数 | 4 | 2 | 1 | 2 | 1 |
| 删除一个数 | 2 | 2 | 2 | 2 | 1 |
| 平均时间 | 2.33 | 2.67 | 2 | 2 | 1.33 |
| 删除两个数 | 5 | 6 | 4 | 1 | 3 |
| 删除两个数 | 5 | 6 | 2 | 1 | 2 |
| 删除两个数 | 7 | 5 | 2 | 5 | 3 |
| 平均时间 | 5.67 | 5.67 | 2.67 | 2.33 | 2.67 |

表2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数据量  时间  操作 毫秒 | 10 | 100 | 300 | 600 | 1000 |
| 跳跃链表查找 | 2 | 2 | 1 | 1 | 2 |
| 跳跃链表查找 | 1 | 2 | 1 | 2 | 2 |
| 跳跃链表查找 | 2 | 2 | 3 | 1 | 2 |
| 平均时间 | 1.67 | 2 | 1.67 | 1.33 | 2 |
| 二分查找 | 1 | 1 | 1 | 1 | 1 |
| 二分查找 | 1 | 2 | 2 | 1 | 3 |
| 二分查找 | 1 | 2 | 2 | 2 | 2 |
| 平均时间 | 1 | 1.67 | 1.67 | 1.33 | 2 |

二分查找假设长度为n, n=1, 查找次数x=，时间复杂度O(log n)

可以看此数据量增加，跳表的索引增加，跳表各种操作的运行时间减少，并且查找操作和二分查找的时间相差无几。

### （七）、心得体会

这次的课设题目虽然一开始让人摸不到头脑，但是看了题目中附带的解释之后，大概明白了这是单链表的一种衍生结构，本来打算行列两个方向都采用指针方式连接，但是在编写时遇到了问题，感觉难继续进行，所以竖列改为指针数组，将指针存放在数组里，在对元素的level层数进行随机确定时，一开始打算采用抛硬币决定，后来发现随机出来的数相同，因为时间太短，改为直接一次随机决定层数，查找函数花了点时间进行编写，然后发现插入和删除函数都是基于查找方法，这节省了一些时间，然后就是随机大量的数据进行测试，比较运行时间，来得出数据越大，索引越多，查询速度越快的结论。

### 附录 源代码：

#pragma once

#include <vector>

#include <iostream>

using namespace *std*;

#define MAX\_LEVEL 10

struct SkipNode

{

int key;

int value;

int level;

SkipNode\* next[MAX\_LEVEL];

SkipNode(int key0 = 0, int v = 0, int level0 = 0)//构造函数

{

value = v;

key = key0;

level = level0;

for (int i = 0; i < level; i++)

{

next[i] = *NULL*;

}

}

};

#include "SkipNode.h"

const int minint = *numeric\_limits*<int>::*min*();//定义监视哨

const int maxint = *numeric\_limits*<int>::*max*();

#include <iostream>

#include <fstream>

#include <ctime>

#include <string.h>

#include <windows.h>

#include <random>

#include <iomanip>

using namespace *std*;

class SList

{

public:

SList(int);

~SList();

int randomLevel();//随机层数level

SkipNode\* insert(int v);//插入

SkipNode\* search(int k);//查找

SkipNode\* deleteNode(int k);//删除

void print();//打印信息

void outfile();//输出文件

int binary\_search(int arr[], int low, int high, int key);

void Mergesort(int\*a,int\*b,int low,int high);

void Merge(int\* a, int\*b, int low, int mid, int high);

private:

SkipNode\* tail;

SkipNode\* head;

int level;

};

#include "SList.h"

SList::SList(int v)

{

head = new SkipNode(minint, v, MAX\_LEVEL);

tail = new SkipNode(maxint, v, MAX\_LEVEL);

//所有层数上的头结点指向尾节点

for (int i = 0; i < MAX\_LEVEL; i++)

{

head->*next*[i] = tail;

}

}

SList::~SList()

{

if (head->*next*[0] != tail)//两个监视哨之间有元素

{

SkipNode\* ptr = head->*next*[0];

while (ptr->*next*[0] != tail)

{

SkipNode\* tmp = ptr;//后移指针并删除是上个元素

ptr = ptr->*next*[0];

delete tmp;

}

for (int i = 0; i < MAX\_LEVEL; i++) {

head->*next*[i] = tail;

}

level = 0;

delete head;

delete tail;

}

}

int SList::randomLevel()

{

//原本打算利用抛硬币方法决定层数，后来发现运行时间太短随机数种子无法正常使用

*random\_device* rd; //来源于c++11random库

*uniform\_int\_distribution*<unsigned> u(1, 10);

/\*

level = 1;

int p = 0;

//srand((unsigned)time(NULL));

p = 1 + rand ()% MAX\_LEVEL;

level = p;\*/

level = u(rd);

return level;

}

SkipNode\* SList::insert(int v)

{

int newlevel = randomLevel();

SkipNode\* newnode = *NULL*;

SkipNode\* tmp = head;

newnode = search(v);

//结点元素已经存在了

if (newnode) {

newnode->value = v;

// new\_node->key = v;

*cout* << "元素" << v << "已经在表中了 " << *endl*;

return head;

}

*cout* << "元素key: " << v << "层数level为: " << newlevel << *endl*;

// cout << "value: " << v << endl;;

newnode = new SkipNode(v, v, newlevel);//存入数据

for (int i = (newlevel - 1); i >=0; i--)

{

while (tmp->*next*[i] != nullptr && tmp->*next*[i]->key < v)//如果原有表的最后元素后继不为空且比待插入元素小

{

tmp = tmp->*next*[i];//指针后移

}

/随着i减小插满竖列

}

return head;

}

SkipNode\* SList::search(int v)

{

SkipNode\* tmp = *NULL*;

tmp = head;

for (int i = level - 1; i >= 0; i--)

{

while (tmp->*next*[i]->key != maxint && v > tmp->*next*[i]->key)//搜索值大于下一个数据的值

{

tmp= tmp->*next*[i];

}

//否则i--，转到下一层

}

tmp= tmp->*next*[0];//找到最底层

if (tmp != *NULL* && tmp->key == v)

{

//cout << "找到了元素 " <<tmp->value<< endl;

}

return *NULL*;

}

SkipNode\* SList::deleteNode(int x)

{

int flag = 0;

SkipNode\* update[MAX\_LEVEL];

SkipNode\* tmp= head;

// 寻找每一层上待删除节点之前的节点

for (int i = MAX\_LEVEL - 1; i >= 0; --i)

{

if (tmp->*next*[i] == tail || tmp->*next*[i]->key > x)//如果头尾之间没元素，或者下一个元素大于要删除的元素

update[i] = *NULL*;

else {

while (tmp->*next*[i] != tail && tmp->*next*[i]->key < x)//头尾间有元素，并且下一个元素小于待删除元素

tmp =tmp->*next*[i]; //不断继续查找

if (tmp->*next*[i] != tail && tmp->*next*[i]->key == x)//判断下一个元素是否是要找的

{

update[i] = tmp; //是，存入结点

flag = 1;

}

else

{

//否，说明元素不存在

}

}

}

SkipNode\* temp = *NULL*;

// 在每层上的链表中删除节点,并将链表连接好

for (int i = 0; i < MAX\_LEVEL; ++i) {

if (update[i])

{

temp = update[i]->*next*[i];

update[i]->*next*[i] = temp->*next*[i];

}

}

//

if (flag==1)

{

*cout* << "元素" << x << "删除成功" << *endl*;

}

else

*cout* << "要删除的元素" << x << "不存在" << *endl*;

delete temp;

return head;

}

int SList::binary\_search(int arr[], int low, int high, int target)

{

int middle = (low + high) / 2;

if (low > high)

return -1;

if (arr[middle] == target)

return middle;

if (arr[middle] > target)

return binary\_search(arr, low, middle - 1, target);

if (arr[middle] < target)

return binary\_search(arr, middle + 1, high, target);

}

void SList::Merge(int\* a, int\* b, int low, int mid, int high)

{

int i = low, j = mid + 1, p = 0; //对应a数组的下标

int n = 0; n = high - low + 1;

while (i <= mid && j <= high)

{

b[p++] = (a[i] <= a[j]) ? a[i++] : a[j++];

}

while (i <= mid)

b[p++] = a[i++];

while (j <= high)

b[p++] = a[j++];

for (p = 0, i = low; i <= high; p++, i++)

a[i] = b[p];

}

void SList::Mergesort(int\*a,int\*b,int low,int high)

{

if (low < high)

{

int mid = (low + high) / 2;

Mergesort(a, b, low, mid);//归并划分

Mergesort(a, b, mid + 1, high);

Merge(a, b, low, mid, high);//进行排序

}

}

void SList::print()

{

for (int i = 0; i < MAX\_LEVEL; i++)

{

SkipNode\* tmp = head;

int lineLen = 1;

if (tmp->*next*[i]->key != maxint)

{

*cout* << *endl* << "这是第 " << i + 1 << "层 :" << *endl*;

while (tmp->*next*[i] != *NULL* && tmp->*next*[i]->key != maxint)

{

*cout* << *setw*(2) << "Key: " << *setw*(3) << tmp->*next*[i]->key << "," << *setw*(5) << "Value: " << *setw*(4) << tmp->*next*[i]->value << " ";

tmp = tmp->*next*[i];

if (lineLen != 0 && lineLen++ % 5 == 0)

{

*cout* << *endl*;

}

}

*cout* << *endl*;

}

}

}

void SList::outfile()

{

*ofstream* outFile;

outFile.*open*("skiplist.txt", *ofstream*::*app*);

for (int i = 0; i < MAX\_LEVEL; i++)

{

SkipNode\* tmp = head;

int lineLen = 1;

if (tmp->*next*[i]->key != maxint)

{

outFile << *endl* << "这是第 " << i + 1 << "层 :" << *endl*;

while (tmp->*next*[i] != nullptr && tmp->*next*[i]->key != maxint)

{

outFile << *setw*(6) << "Key: " << *setw*(4) << tmp->*next*[i]->key << "," << *setw*(6) << "Value: " << *setw*(4) << tmp->*next*[i]->value << " ";

tmp = tmp->*next*[i];

if (lineLen != 0 && lineLen++ % 5 == 0)

{

outFile << *endl*;

}

}

outFile << *endl*;

}

}

}

#include <iostream>

#include "SList.h"

double starttime, endtime;

void randomnum(int number[], int num)//随机数函数

{

*random\_device* rd;

*uniform\_int\_distribution*<unsigned> u(1, 10000);

for (int i = 0; i < num; i++)

{

number[i] = u(rd);

}

}

void showmenu()

{

*cout* << *endl*;

*cout* << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*菜单\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << *endl*;

*cout* << "\*\*\* 1——插入随机数 \*\*\*" << *endl*;

*cout* << "\*\*\* 2——查找元素 \*\*\*" << *endl*;

*cout* << "\*\*\* 3——删除元素 \*\*\*" << *endl*;

*cout* << "\*\*\* 4——打印信息 \*\*\*" << *endl*;

*cout* << "\*\*\* 5——插入自定义数 \*\*\*" << *endl*;

*cout* << "\*\*\* 6——输出文件 \*\*\*" << *endl*;

*cout* << "\*\*\* 7——二分查找 \*\*\*" << *endl*;

*cout* << "\*\*\* 8——退出系统 \*\*\*" << *endl*;

*cout* << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << *endl*;

*cout* << *endl*;

}

void choosemenu()

{

int v = 0;

SList S(v);

int choice = 1;

while (choice)

{

showmenu();

*cin* >> choice;

if (choice > 8 || choice < 0)

{

*cout* << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << *endl*;

*cout* << "\*\*\* 选择操作有误，请重新输入 \*\*\*" << *endl*;

*cout* << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << *endl*;

continue;

}

switch (choice)

{

case 1:

{

//srand((int)time(NULL));

int num;

*cout* << "请输入要插入的元素数目" << *endl*;

*cin* >> num;

int\* a = new int[num];

randomnum(a, num);

starttime = *clock*();

for (int i = 0; i <num; i++)

{

S.insert(a[i]);

}

endtime = *clock*();

*cout* << "运行时间为：" << endtime - starttime << "毫秒" << *endl*;

break;

}

case 2:

{

int n;

*cout* << "请输入要查找的元素个数" << *endl*;

*cin* >> n;

int\* anum = new int[n];

*cout* << "请输入要查找的数" << *endl*;

for (int i = 0; i < n; i++)

{

*cin* >> anum[i];

}

starttime = *clock*();

for (int i = 0; i < n; i++)

{

if (S.*search*(anum[i]))

{

*cout* << "找到了元素" << anum[i] << *endl*;

}

else

{

*cout* << "未找到元素" << anum[i] << *endl*;

}

}

endtime = *clock*();

*cout* << "运行时间为：" << endtime - starttime << "毫秒" << *endl*;

break;

}

case 3:

{

int n;

*cout* << "请输入要删除的元素个数" << *endl*;

*cin* >> n;

*cout* << "请输入要删除的元素" << *endl*;

int\* dlnum = new int[n];

for (int i = 0; i < n; i++)

{

*cin* >> dlnum[i];

}

starttime = *clock*();

for (int i = 0; i < n; i++)

{

S.deleteNode(dlnum[i]);

}

endtime = *clock*();

*cout* << "运行时间为：" << endtime - starttime << "毫秒" << *endl*;

break;

}

case 4:

S.print();

break;

case 5:

{

int n;

*cout* << "请输入要自定义插入的元素个数" << *endl*;

*cin* >> n;

*cout* << "请输入要自定义插入的元素" << *endl*;

int\* innum = new int[n];

for (int i = 0; i < n; i++)

{

*cin* >> innum[i];

}

starttime = *clock*();

for (int i = 0; i < n; i++)

{

S.insert(innum[i]);

}

endtime = *clock*();

*cout* << "运行时间为：" << endtime - starttime << "毫秒" << *endl*;

break;

}

case 6:

S.outfile();

break;

case 7:

{

int BSnum=0;

*cout* << "请输入要创建的数组长度" << *endl*;

*cin* >> BSnum;int\* BSarr = new int[BSnum];

int\* newBSarr = new int[BSnum];

randomnum(BSarr, BSnum);

S.Mergesort(BSarr, newBSarr, 0, BSnum - 1);

for (int i = 0; i < BSnum; i++)

{

if (i!=0&&i % 20== 0)

{

*cout* << *endl*;

}

*cout* << *setw*(4) <<newBSarr[i] << " ";

}

*cout* << *endl*;

int n;

*cout* << "请输入要查找的元素个数" << *endl*;

*cin* >> n;

int\* sonum = new int[n];

*cout* << "请输入要查找的数" << *endl*;

for (int i = 0; i < n; i++)

{

*cin* >> sonum[i];

}

starttime = *clock*();

for (int i = 0; i < n; i++)

{

//cout << S.binary\_search(newBSarr, 0, BSnum - 1, sonum[i])<< endl;

if (S.*binary\_search*(BSarr, 0, BSnum - 1, sonum[i]) == -1)

{

*cout* << "未找到元素" << sonum[i] << *endl*;

}

else

{

*cout* << "找到了元素" << sonum[i] <<"位置为"<<S.*binary\_search*(newBSarr, 0, BSnum - 1, sonum[i])+1<< *endl*;

}

}

endtime = *clock*();

*cout* << "运行时间为：" << endtime - starttime << "毫秒" << *endl*;

break;

break;

}

}

}

}

int main()

{

//showmenu();

choosemenu();

return 0;

}