实验报告

算法实现

- 随机数生成

```
int randomValue(){
    int lvl = 1;
    double min = 0,max = 1000;
    random_device seed;//硬件生成随机数种子
    ranlux48 engine(seed());//利用种子生成随机数引擎
    uniform_int_distribution<> distrib(min, max);//设置随机数范围，并为均匀分布
    while((distrib(engine))*0.001<p &&lvl<MAX_LEVEL){
        lvl++;
    }
    return lvl;
}
```

- 插入算法实现

```
void Insert(int searchKey,T newValue){
    SkipNode<T> *update[MAX_LEVEL];
    SkipNode<T> *x=header;
    for(int i = _level-1;i>=0;i--){
        while(x->forward[i]!=nullptr&&x->forward[i]->key<searchKey){
            x = x->forward[i];
        }
        update[i] = x;
    }
    x=x->forward[0];
    if(x->key == searchKey){
        x->val=newValue;
        return;
    }
    else{
        int lvl = randomValue();
        if(lvl> _level){
            for(int i = _level;i<lvl;i++)update[i]=header;
            _level = lvl;
        }
        SkipNode<T> *new_node = new SkipNode<T>(lvl,searchKey,newValue);
        for (int i = lvl - 1; i >=0; i--) {
            new_node->forward[i] = update[i]->forward[i];
            update[i]->forward[i] = new_node;
        }
    }
    return;
}
```

- 搜索实现

```
/*查找到了返回查找个数，否则返回-1*/
int SearchNode(int searchKey){
    int len=0;
    if(header == nullptr)return 0;
    SkipNode<T> *x=header;
    for(int i = _level-1;i>=0;i--){
        while(x->forward[i]->key<searchKey){
            x = x->forward[i];
            len++;
        }
    }
    x = x->forward[0];
    if(x->key==searchKey)return len;
    else return -1;
}
```

- 删除实现

```
bool Delete( int searchKey){
    SkipNode<T> *update[MAX_LEVEL];
    SkipNode<T> *x=header;
    if(x== nullptr)return false;
    for(int i = _level-1;i>=0;i--){
        while(x->forward[i]!=nullptr&&x->forward[i]->key<searchKey){
            x = x->forward[i];
        }
        update[i] = x;
    }
    x=x->forward[0];
    if(x->key==searchKey){
        for(int i=0;i<_level;i++){
            if(update[i]->forward[i]!=x)break;
            update[i]->forward[i]=x->forward[i];
        }
        delete x;
        while (_level >=0 && header->forward[_level-1]==nullptr){
            _level --;
        }
        return true;
    }
    return false;
}
```

测试用例

建立长度（跳表元素个数）分别为 50, 100，200，500，1000，概率 p 分别为 1/2，1/e，1/4，1/8 的跳表。对于长度为n的跳表，插入数字1-n。

进行10000次随机搜索，生成一个[1.n]之间的整数并进行搜索，记录搜索的次数和并除以10000。

- 随机测试相关代码

```
double test(int n,double p){
    SkipList<char> testlist(n,p);
    for(int i=1;i<=n;i++){
        testlist.Insert(i,'A');
    }
    double sum=0;
    int min = 1,max = n;
    random_device seed;//硬件生成随机数种子
    ranlux48 engine(seed());//利用种子生成随机数引擎
    uniform_int_distribution<> distrib(min, max);//设置随机数范围，并为均匀分布
    for(int i=0;i<100000;i++){
        sum+=testlist.SearchNode(distrib(engine));
    }
    for(int i=1;i<=n;i++){
        testlist.Delete(i);
    }
    return double(sum/100000);
}
```

- main函数对测试的调用

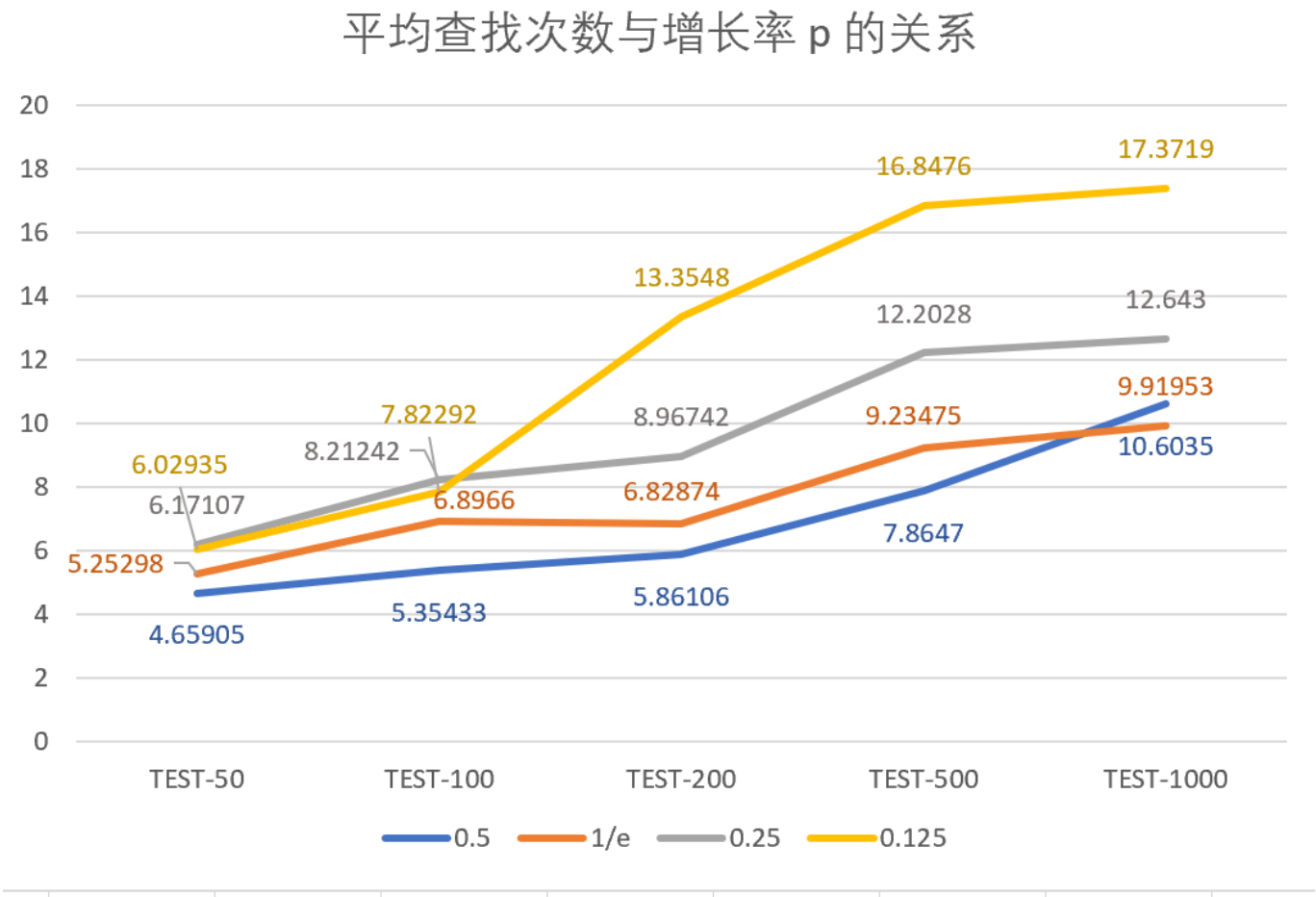
```
int main() {
    cout<<"Testcase_50:"<<
        test(50,0.5)<<" "<<test(50,1/ 2.718281801)<<" "<<
        test(50,0.25)<<" "<<test(50,0.125)<<endl;
    cout<<"Testcase_100:"<<
        test(100,0.5)<<" "<<test(100,1/2.718281801)<<" "<<
        test(100,0.25)<<" "<<test(100,0.125)<<endl;
    cout<<"Testcase_200:"<<
        test(200,0.5)<<" "<<test(200,1/ 2.718281801)<<" "<<
        test(200,0.25)<<" "<<test(200,0.125)<<endl;
    cout<<"Testcase_500:"<<
        test(500,0.5)<<" "<<test(500,1/ 2.718281801)<<" "<<
        test(500,0.25)<<" "<<test
    cout<<"Testcase_1000:"<<
        test(1000,0.5)<<" "<<test(1000,1/ 2.718281801)<<" "<<
        test(1000,0.25)<<" "<<test(1000,0.125);(500,0.125);
    return 0;
}
```

实验结果及分析

- 实验结果

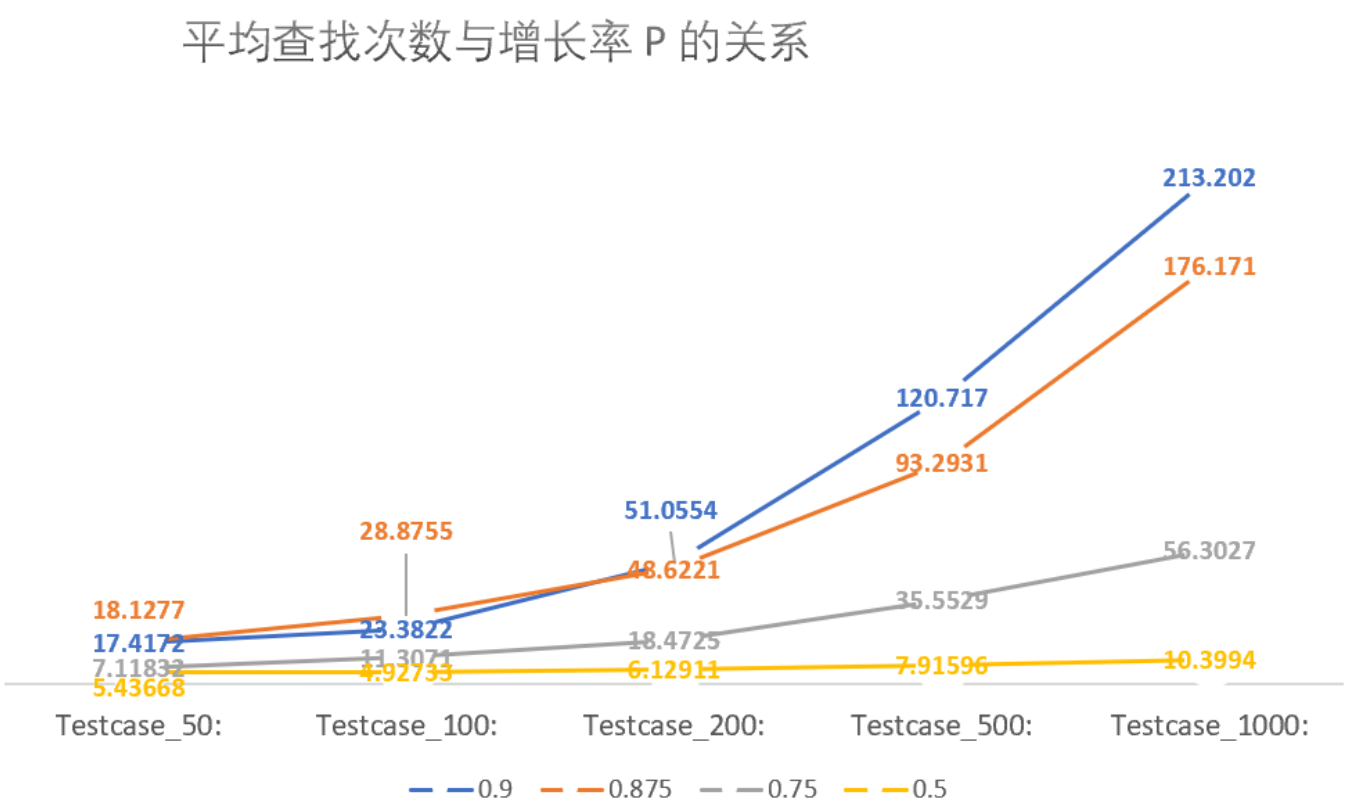
  其中Testcase_n为插入n个数的相关测试，第1-4列的概率分别为概率 p 分别为 1/2，1/e，1/4，1/8 。所生成的数为平均随即搜索次数。

```
C:\Users\BOOK3\Desktop\hw2\cmake-build-debug\hw2.exe
Testcase_50:4.65905 5.25298 6.17107 6.02935
Testcase_100:5.35433 6.8966 8.21242 7.82292
Testcase_200:5.86106 6.82874 8.96742 13.3548
Testcase_500:7.8647 9.23475 12.2028 16.8476
Testcase_1000:10.6035 9.91953 12.643 17.3719
Process finished with exit code 0
```



其中Testcase_n为插入n个数的相关测试，第1-4列的概率分别为概率 p 分别为 9/10，7/8，3/4，1/2 。所生成的数为平均随即搜索次数。

```
Testcase_50:17.4172 18.1277 7.11832 5.43668
Testcase_100:23.3822 28.8755 11.3071 4.92733
Testcase_200:51.0554 48.6221 18.4725 6.12911
Testcase_500:120.717 93.2931 35.5529 7.91596
Testcase_1000:213.202 176.171 56.3027 10.3994
```



- 实验现象

  随着搜索次数增长，平均搜索长度折线呈指数级增加趋势，粗略得知该搜索时间复杂度为`O(logn)`。随着概率P接近于1/2，平均搜索次数呈减少趋势，概率越接近1/2，搜索次数越少。

- 理论分析

  - 该数据结构的时间复杂度 = 索引的高度 * 每层索引遍历元素的个数，即假设每i个结点会抽出一个结点作为上一级索引的结点，原始的链表有n个元素，则一级索引有$\frac{n}{i}$个元素、二级索引有 $\frac{n}{i^2}$ 个元素、k级索引就有 $\frac{n}{i^k}$个元素,最高级索引一般有i个元素。最高级索引 h 满足$i = \frac{n}{i^h}$，即 $h = \log_i n - 1$，最高级索引 h 为索引层的高度加上原始数据一

层，跳表的总高度 $h = \log_i n$。因此，查找函数的时间复杂度为O(logn);插入函数的时间复杂度为O(logn);

根据论文中的数学推断，在查找过程中，最多走h层，每层最多走3次，总查找次数期望上界为$\frac{(1-p)L(n)}{p^2} + \frac{p}{(1-p)^2} + \frac{2p-1}{p^2}, p = \frac{1}{i}$,因此当n相同且足够大时，查找次数期望主要由第一项影响，又只当$p = 0.5$时，$\frac{(1-p)}{p^2}$最小，因此概率越接近1/2，搜索次数越少。