# hw2 实验报告

学号：521021910952

姓名：杜心敏

## 1. 结构说明

- 整体结构是一个单链表`List < node* >`
- 链表中每个单元`node`存放一个值`key`和一系列指针（用`vector`存储），为了方便，记录了每个`node`的高度`level`
- 单链表中记录了表头`head`，与表尾`tail`。表头数值为`INT_MIN`，尾部为`INT_MAX`，尾部的指针为空指针。

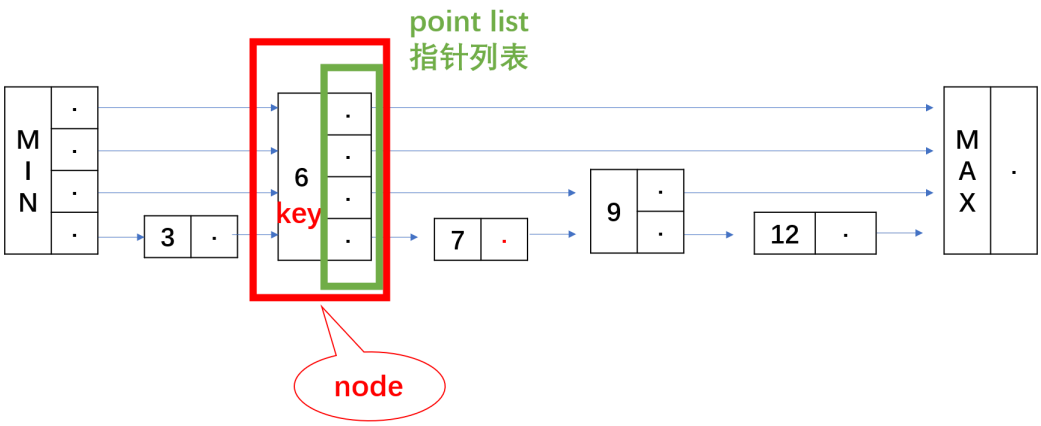



图1 数据结构示意

```
struct node{
    int key;
    int level;
    vector<node*> point_list;
};

class SkipList{
    int Max_level; //整个结构最高层数
    node* head;
    node* tail;
    ...
};
```

## 2. search、insert代码说明

### 2.1 insert

```
insert(value)
    local update_list[Max_level + 1]
    curr_level := Max_level
    next := head

```

```
    while curr_level >= 0 do
        x := next->point_list[curr_level]
        if value < x->key  then
            update_list[curr_level] := next
            curr_level--
        if value > x->key then
            next := x
    NewNode = new node(val, next->point_list[0]);
    while rand() % 1000 < P do
        NewNode.level ++
        if  NewNode.level > Max_level  then
            Max_level ++
            head.push_back( NewNode )
            NewNode.push_back( tail )
        else
            NewNode.push_back( update_list[NewNode.level] -> next )
            update_list[NewNode.level] -> next := NewNode
```

- 需要一个额外数组`update_list`去记录哪些`node`中的指针可能要更新。数组下标是需要更新的指针位于`node`的第几层，数组存放指向该`node`的指针。
- 第一个while循环在查找插入位置。层次由高到低，横向查找。
  - 如果当前值大于下一个，则往左。
  - 如果小于下一个，则往下，并且将所在的`node`记录到`update_list`中。
  - 如下图，插入10的时候，红色的是路径，黄色的格子都需要记录，他们的指针都可能改变（具体要看10的层高）。
- 第二个while循环确定10的层高，根据概率增加10的层高
  - 当层高小于`Max_level`时，改变相应`update_list`中的指针指向
  - 当层高超过`Max_level`时，更新层高，增加`head`的层高

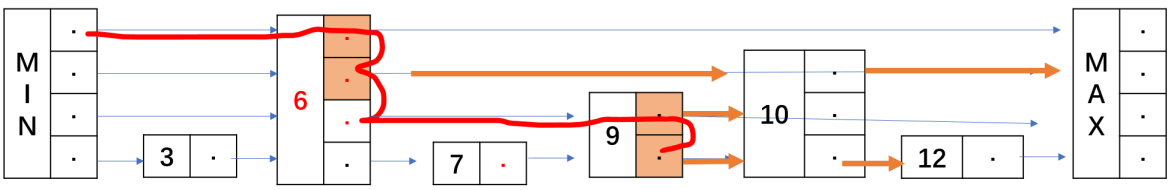



图2 插入10的示例

### 2.1 search

`search`其实就是`insert`前一个循环，不再做解释。搜索路径示例如图2红色曲线。

```
search( value)
    curr_level := Max_level
    curr_node := head

    while curr_level >= 0  do
        if value == curr_node->next.key
            return step
        if value < curr_node->next.key
            curr_level--
        else
            curr_node = curr_node->next

```

> 注：为了方便叙述，伪代码中命名与实际略有不同
>
> 实际操作时，增加了step变量记录搜素长度，每次比较增加1

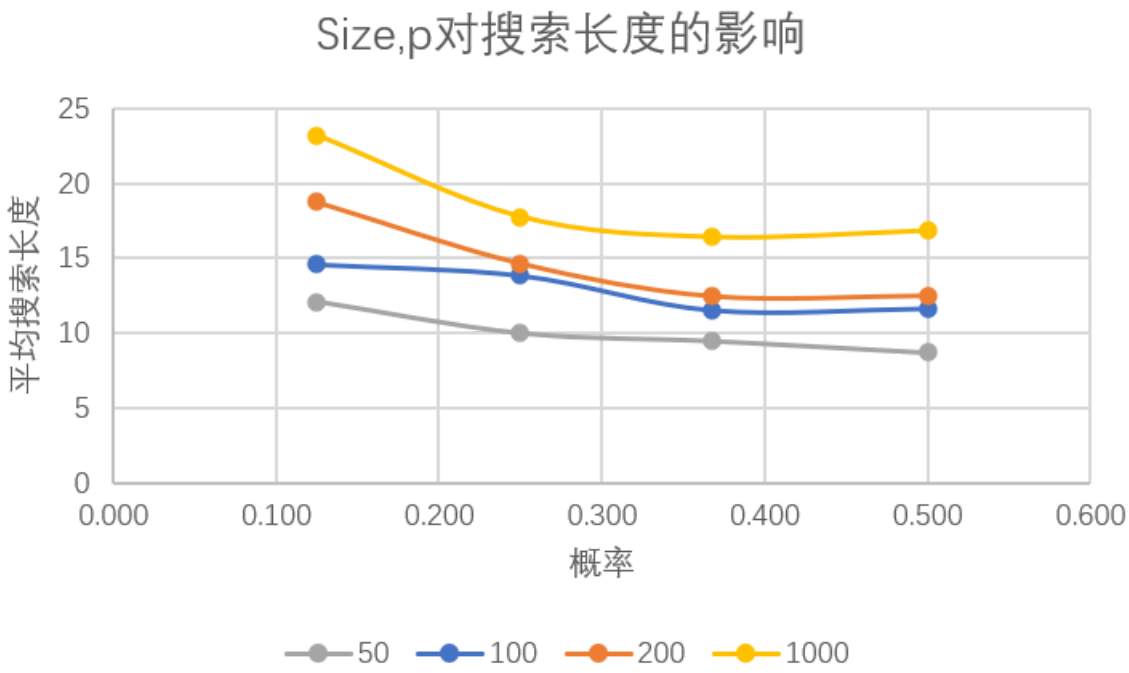
### 3. 测试设计说明

- `Size`：元素个数
- 搜索次数：10000次
- 将含有1—Size的`vector`，用`random_shuffle`打乱插入
- 搜索时，随机生成1—Size范围内的整数进行搜索
- 实验时分别固定Size = 50, 100，200，500，1000 。然后改变p值，观察搜索长度变化。

表1 数据记录

| Size | p | 平均搜索长度 |
| --- | --- | --- |
| 50 | 1/2 | 8.7047 |
| 50 | 1/e = 0.368 | 9.4682 |
| 50 | 1/4 | 9.9965 |
| 50 | 1/8 | 12.0668 |
| 100 | 1/2 | 11.6413 |
| 100 | 1/e | 11.522 |
| 100 | 1/4 | 13.8559 |
| 100 | 1/8 | 14.5972 |
| 200 | 1/2 | 12.489 |
| 200 | 1/e | 12.4522 |
| 200 | 1/4 | 14.6428 |
| 200 | 1/8 | 18.7394 |
| 500 | 1/2 | 14.774 |
| 500 | 1/e | 14.6397 |
| 500 | 1/4 | 15.0703 |
| 500 | 1/8 | 28.0146 |
| 1000 | 1/2 | 16.8195 |
| 1000 | 1/e | 16.4022 |
| 1000 | 1/4 | 17.7567 |
| 1000 | 1/8 | 23.1891 |

## 4.数据分析




- 跳表的平均查找长度在$O(\log n)$。对比同一p下，Size越大，越接近理想值（二分查找），可能是Size较大时，生成随机链表高度较为稳定且合适，Size较小时，随机概率不稳定，容易出现退化单链表的形式。
- 总体来看，p值越大，层高越高，搜索长度越短。但是1/e和1/2的搜索长度接近，有时1/e更好。在实验中发现这两个概率的结果不太稳定，会上下浮动1-2的大小，尤其是Size较小的时候。p值较大时，层数过高，也增加了比较次数。
- Size越大，p对搜索长度的影响越大。就像AVL树一样，规模越大，越需要平衡。
- 总体来看p选择[0.25,0.5]之间较为合适。论文中选择0.25和0.5，因为是2的幂次，产生随机数更快更方便。