Homework 5 : Red Black Tree

获取实验数据

```
static inline uint64_t rdtsc() {
    uint32_t low, high;
    asm volatile ("rdtsc" : "=a" (low), "=d" (high));
    return ((uint64_t) high << 32) | low;
}
```

Insert

要求即实现

要求

- 五组输入集
- 采用顺序、乱序两种输入方式

实现

- 五组输入集大小 : 100， 1000， 10000， 20000， 30000
- 说明
  - 前三个为指数关系增长，后三个成线性增长
  - 后三组数据更大，增加实验可靠性
- 顺序插入时使用`set`容器

顺序插入

红黑树、AVL树、跳表 耗时

实验图片

```
RBT Case 1 : 1819086
RBT Case 2 : 968844
RBT Case 3 : 9936716
RBT Case 4 : 20845020
RBT Case 5 : 34360708

AVL Tree Case 1 : 236662
AVL Tree Case 2 : 1380580
AVL Tree Case 3 : 15908120
AVL Tree Case 4 : 32532818
AVL Tree Case 5 : 49371216

Skip List Case 1 : 755660
Skip List Case 2 : 7900142
Skip List Case 3 : 607724428
Skip List Case 4 : 2408436980
Skip List Case 5 : 5253255644
```

	100	1000	10000	20000	30000
RedBlack Tree	1819086	968844	9936716	20845020	34360708
AVL Tree	236662	1380580	15908120	32532818	49371216
Skip List	755660	7900142	607724428	2408436980	5253255644

红黑树、AVL树 旋转次数

实验图片

```
RBT Case 1 : 89
RBT Case 2 : 983
RBT Case 3 : 9976
RBT Case 4 : 19974
RBT Case 5 : 29973

AVL Tree Case 1 : 93
AVL Tree Case 2 : 990
AVL Tree Case 3 : 9986
AVL Tree Case 4 : 19985
AVL Tree Case 5 : 29985
```

	100	1000	10000	20000	30000

RedBlack Tree	89	983	9976	19974	29973
AVL Tree	93	990	9986	19985	29985

乱序插入

红黑树、AVL树、SkipList耗时

实验图片

```
RBT Case 1 : 2030620
RBT Case 2 : 1128228
RBT Case 3 : 13981796
RBT Case 4 : 28032260
RBT Case 5 : 44021068

AVL Tree Case 1 : 173476
AVL Tree Case 2 : 1470464
AVL Tree Case 3 : 17962140
AVL Tree Case 4 : 39781918
AVL Tree Case 5 : 60911564

Skip List Case 1 : 330692
Skip List Case 2 : 7048718
Skip List Case 3 : 546717404
Skip List Case 4 : 2808112498
Skip List Case 5 : 7687804696
```

	100	1000	10000	20000	30000
RedBlack Tree	2030620	1128228	13981796	28032260	44021068
AVL Tree	173476	1470464	17962140	39781918	60911564
Skip List	330692	7048718	546717404	2808112498	7687804696

红黑树、AVL树旋转次数

实验图片

```
RBT Case 1 : 58
RBT Case 2 : 572
RBT Case 3 : 5812
RBT Case 4 : 11922
RBT Case 5 : 17559

AVL Tree Case 1 : 74
AVL Tree Case 2 : 719
AVL Tree Case 3 : 7018
AVL Tree Case 4 : 14108
AVL Tree Case 5 : 21232
```

	100	1000	10000	20000	30000
RedBlack Tree	58	572	5812	11922	17559
AVL Tree	74	719	7018	14108	21232

Search

要求即实现

要求

- 五组查找集
- 乱序和顺序两种查找方式
- 所有的key都是已经存在的

实现

- 五组查找集合见上述插入集，为插入集的10倍
- 插入集数据：1000， 10000， 100000， 200000， 300000
- 插入值另存在数组中，取值思路为

```
ofstream fout("search.txt") // 声明
for(int i = 1; i <= n; i++) // 插入n个值
    fout << "s " << dataBase[rand % size + 1] << std::endl;// 读入文件
/*
 * 空间开销巨大，(约10Mb)，但是可以保证这些时间开销不会算入运算时间
 * 并且存入外存，可以保证栈空间足够使用
 */
```

顺序查找耗时 : 红黑树、 AVL树、 跳表

实验图片

```
RBT Case 1 : 1212432
RBT Case 2 : 1143826
RBT Case 3 : 11943128
RBT Case 4 : 24156590
RBT Case 5 : 38587452

AVL Tree Case 1 : 209508
AVL Tree Case 2 : 1496810
AVL Tree Case 3 : 16419942
AVL Tree Case 4 : 35624548
AVL Tree Case 5 : 52120334

Skip List Case 1 : 530070
Skip List Case 2 : 8185354
Skip List Case 3 : 580568614
Skip List Case 4 : 2344820676
Skip List Case 5 : 5198827908
```

	1000	10000	100000	200000	300000
RedBlack Tree	1212432	1143826	11943128	24156590	38587452
AVL Tree	209508	1496810	16419942	35624548	52120334
Skip List	530070	8185354	580568614	2344820676	5198827908

乱序查找耗时 : 红黑树、 AVL树、 跳表

实验图片

```
RBT Case 1 : 2018476
RBT Case 2 : 1086382
RBT Case 3 : 15625980
RBT Case 4 : 28828804
RBT Case 5 : 45099022

AVL Tree Case 1 : 200362
AVL Tree Case 2 : 1487008
AVL Tree Case 3 : 19033600
AVL Tree Case 4 : 43670616
AVL Tree Case 5 : 64584306

Skip List Case 1 : 369750
Skip List Case 2 : 5978854
Skip List Case 3 : 554060466
Skip List Case 4 : 3078596760
Skip List Case 5 : 7171322430
```

	1000	10000	100000	200000	300000
RedBlack Tree	2018476	1086382	15625980	28828804	45099022
AVL Tree	200362	1487008	19033600	43670616	64584306
Skip List	369750	5978854	554060466	3078596760	7171322430

分析实验结果

要求

分析所测红黑树、SkipList、 AVL 树的插入和查询操作开销是否符合理论（至少要对比在顺序和乱序输
入场景下三种结构的插入操作的耗时和两种树的旋转次数的对比以及查找操作的耗时对比），并说明自
己对数据结果差异的理解：

结论

红黑树，skiplist， AVL插入和查询操作时间开销上，跳表时间开销或许过大。

但是根据数据点可以看出，在这些操作中插入和查找操作基本符合`O(logN)`的时间复杂度。

旋转操作上也符合几乎为`O(logN)`的旋转次数,而且红黑树旋转次数也明显少于AVL树旋转次数，因为AVL树目的是几乎严格维护BST且平衡（平衡因子在1以内），红黑树的平衡是相对而言的平衡

- 比较顺序与乱序：
  - 旋转操作上，顺序比乱序有明显更多的旋转操作。
  - 时间开销上，顺序比乱序所花费时间开销更大，因为顺序与乱序相比是更接近最差情况插入或查找顺序。
    - 但是跳表在这个时候会表现出更快插入的优势。