2021 秋季学期高等理工学院“数据结构”课程大作业实验报告

学号：20376203　　姓名：陈俊一　　2021-12-23

一、实验选题、实验内容及功能说明

实验选题：查找树的实现与性能分析

实验内容：

1. 实现 AVL 树、B-树和红黑树；
2. 测试不同数量级节点下各查找树性能；
3. 分析并给出理由并建立动态的量化关系。

功能说明：

1. 包括 AVL 树、B-树、红黑树的插入，查找，删除，遍历等基本功能；
2. 自动化数据生成器以及自动化运行效率测试工具；
3. 利用 Python 绘制折线图、三维散点图和曲面

二、设计方案与设计思路

首先实现三棵平衡树最基本的增删查功能，三棵平衡树的源代码见工作目录下`/AVLTree.c`，`/BTree.c` 和`/RBTree.c`

然后对于每一棵平衡树用 C++语言编写对应的测试程序（这是为了方便用 STL 库函数实现生成不重复的随机数），源代码见工作目录下`/AVLTree_tb.cpp`，`/BTree_tb.cpp` 和`/RBTree_tb.cpp`

每个测试程序均实现 `void SimpleFunctionTest()`和 `void CombinationFunctionTest()`两个函数，分别对应程序运行效果中的 1 和 2 中所

需的测试功能，同时利用 `vector<int> generate_random_array(int n)`实现生成 `n` 个不重复的随机数的功能（实质是把$[1, n]$的元素随机打乱）。

在 `void SimpleFunctionTest()`函数中，直接利用生成的元素依次插入搜索删除并利用 `clock()`函数计时，重复 5 次取平均值作图。

在 `void CombinationFunctionTest()`函数中，由于一次性开大规模数组会导致内存占用过大，因此采用折中方法，先调用生成随机数的函数将生成的随机数写入文件`/tmp/tmp_ins.txt`，`/tmp/tmp_del.txt` 和`/tmp/tmp_sea.txt` 中，然后再分别从文件中读取数据，进行各项操作，结果表明 I/O 的时间影响并不大。

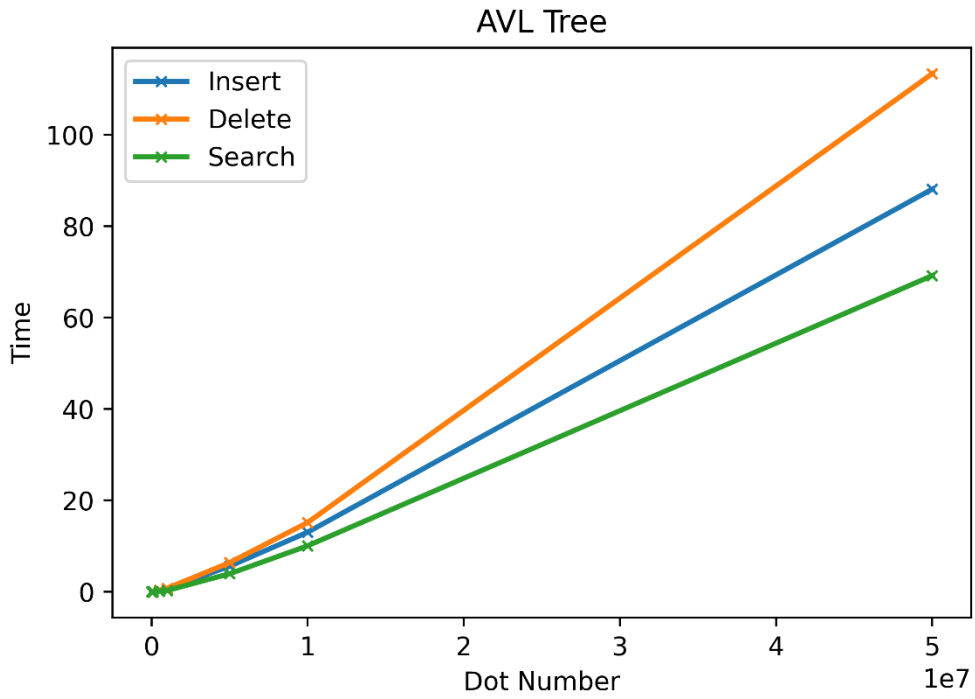
完成测试以后，采用 Python 程序画出本文中的所有统计图，代码和数据文件均在`/ProcessResult` 目录下，主要是运用了 `numpy` 和 `matplotlib.pyplot` 两个工具，`numpy` 负责储存点的坐标信息，使用 `matplotlib` 画图，实现的自动化读取数据画图的 Python 程序是`/ProcessResult/draw_simple.py` 和 `/ProcessResult/draw_comb.py`

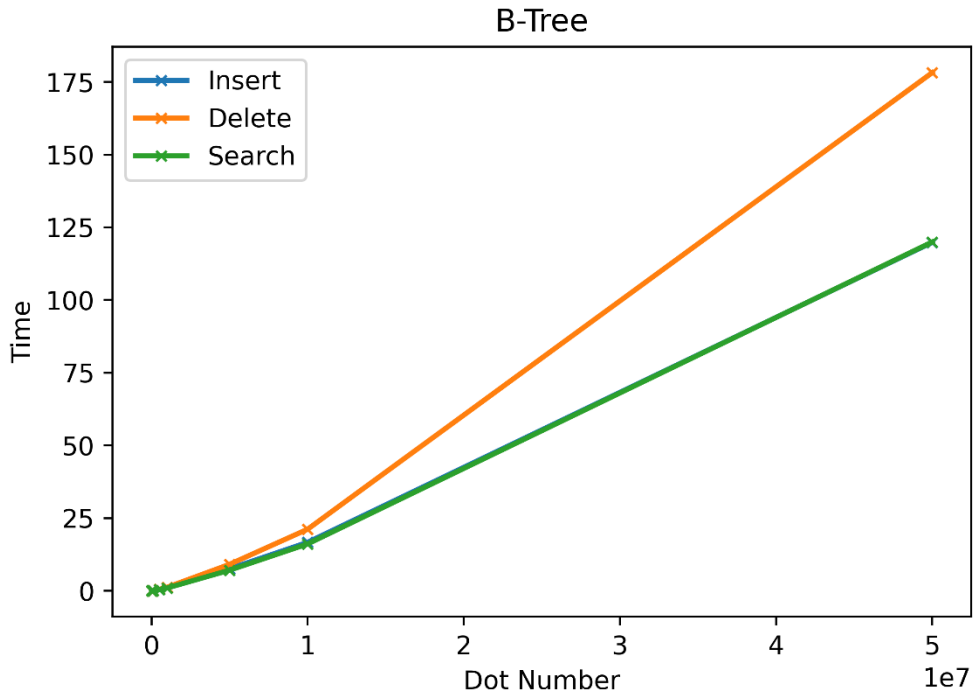
三、程序运行效果

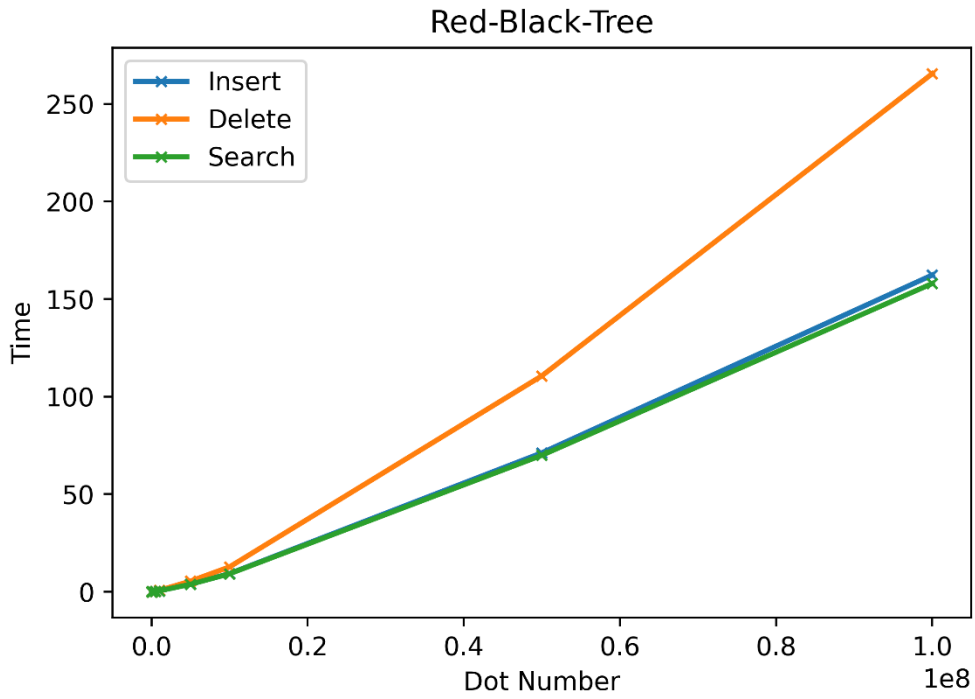
1. 测试不同数据规模下各平衡树性能

2021 秋季学期高等理工学院“数据结构”课程大作业实验报告

学号：20376203 姓名：陈俊一 2021-12-23



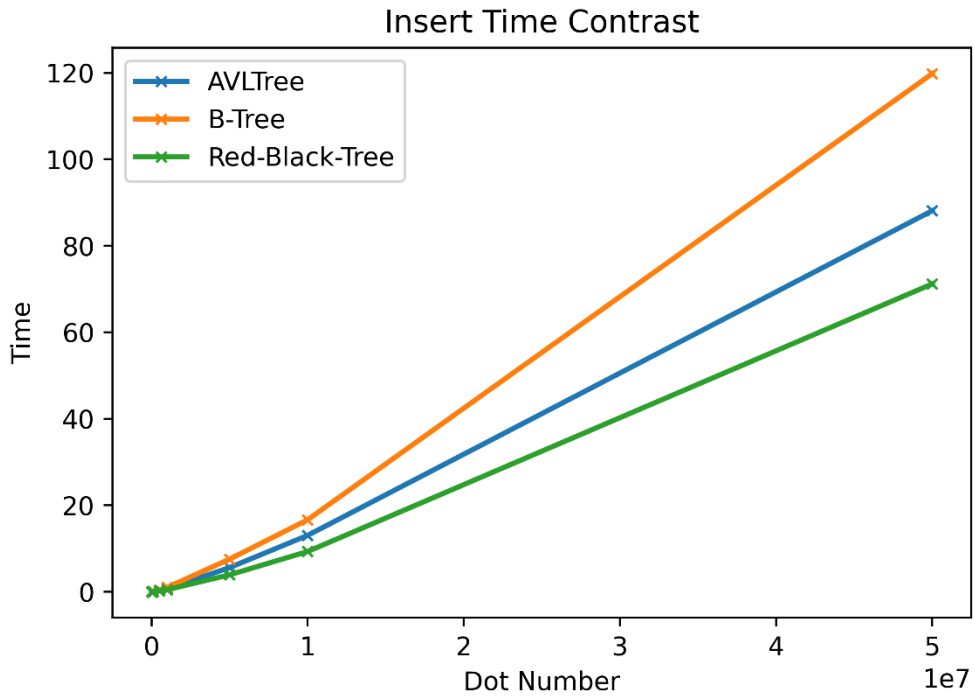


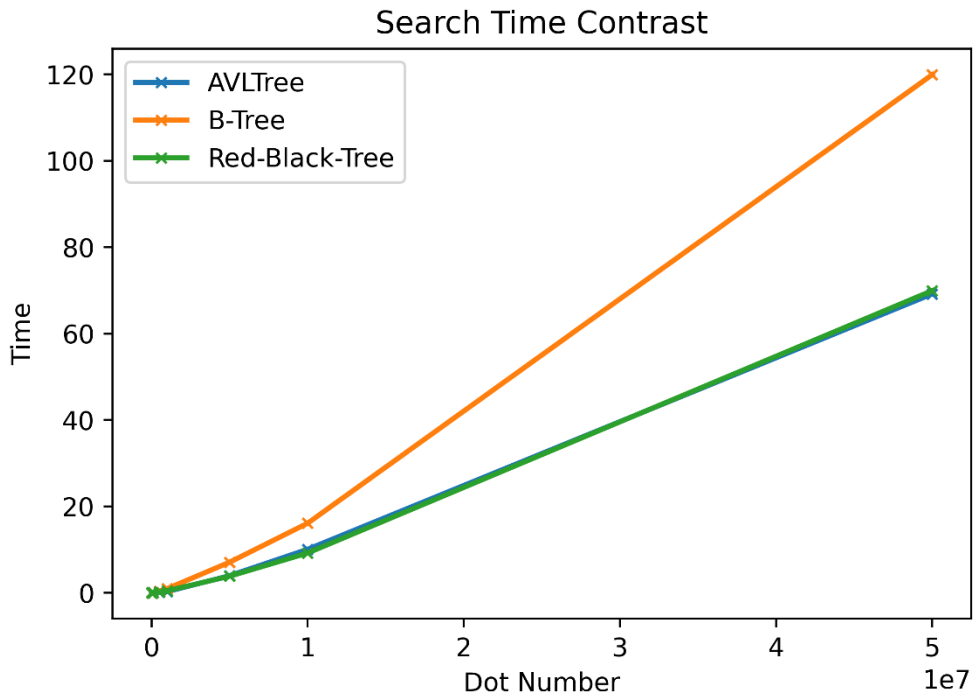


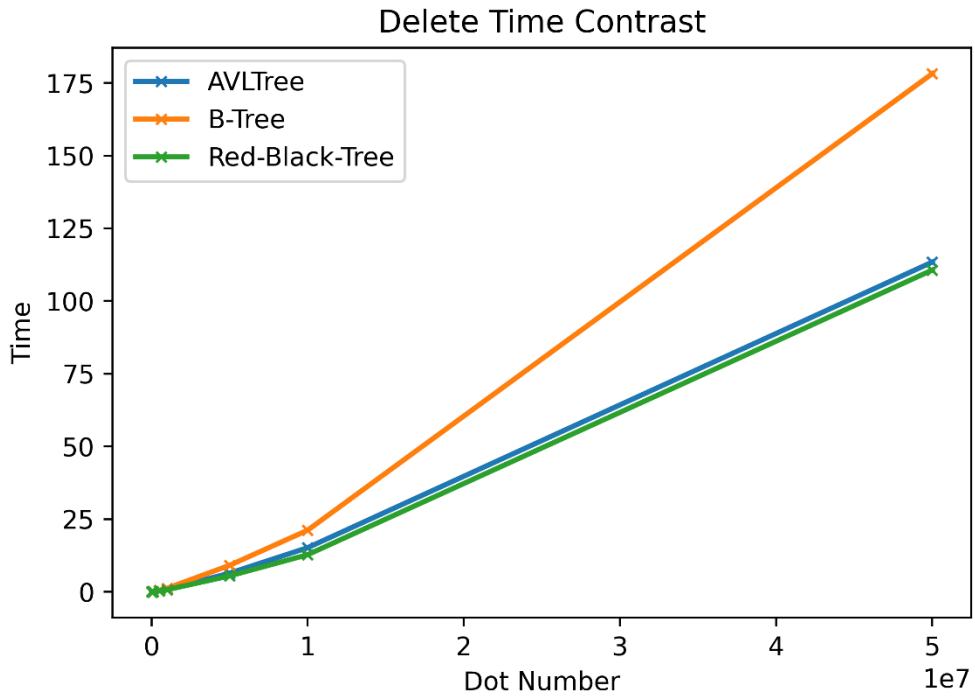
以上三张折线图是由 Python 绘制的不同规模数据下插入、查找与删除所用的时间，对于 AVL 树、B-树，测试数据规模最大是插入/搜索/删除各 5000 0000 次，对于红黑树，测试用例规模最大是插入/搜索/删除各 1 0000 0000 次，测试用例规模按照从 1000, 5000, 1 0000…递增，相同规模测试用例随机生成 5 次取平均值，测试平台为 Windows 11 64 位，Intel Core i7-10750H@2.60GHz，16 GB 内存。

原始数据文件见工作目录下`/res/avl_simple.txt`，`/res/btree_simple.txt` 和 `/res/rbtree_simple.txt`，分别对应 AVL 树，B-树和红黑树的测试结果。

同时对于不同的操作我也进行了横向对比，如下面三张图所示。


Insert Time Contrast
AVLTree
B-Tree
Red-Black-Tree
Time
Dot Number
1e7
0
20
40
60
80
100
120
0
1
2
3
4
5

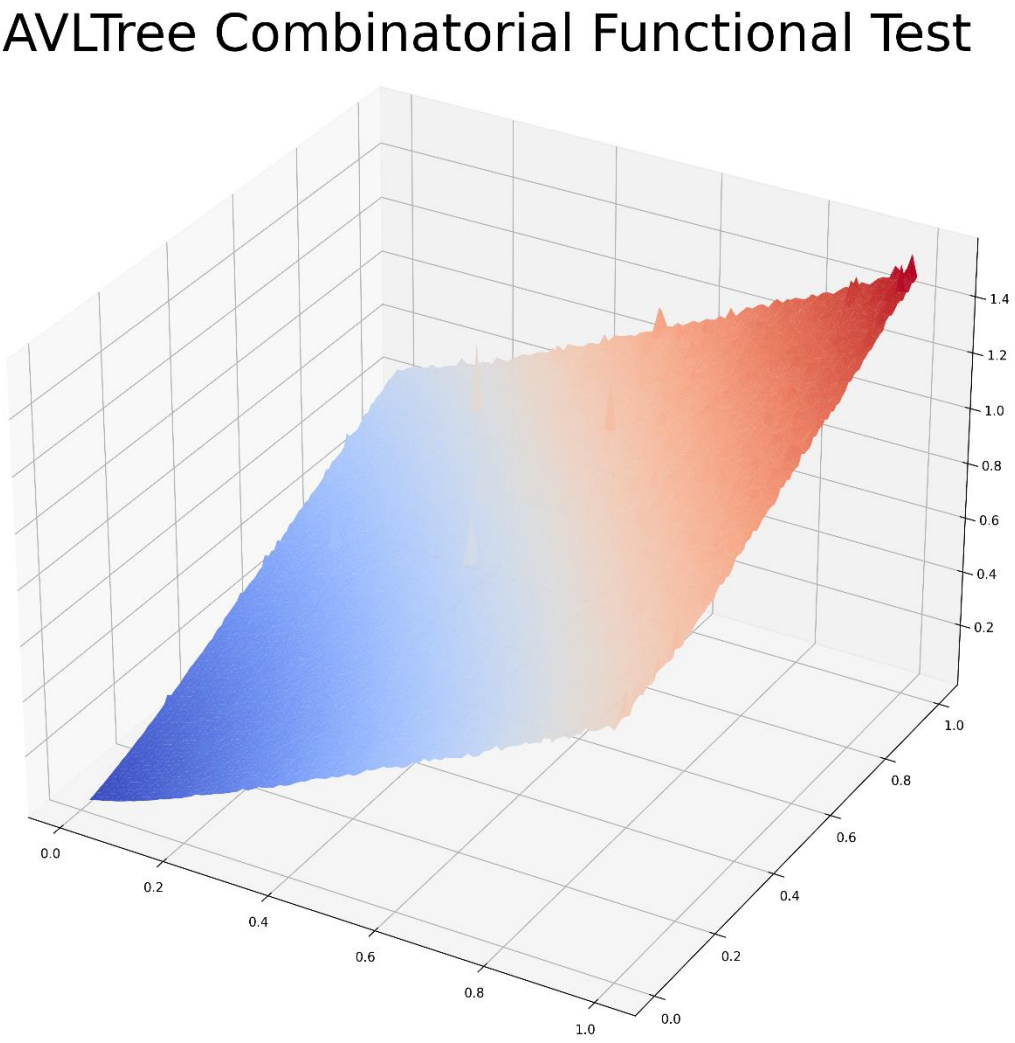

Search Time Contrast
AVLTree
B-Tree
Red-Black-Tree
Time
Dot Number
1e7
0
20
40
60
80
100
120
0
1
2
3
4
5

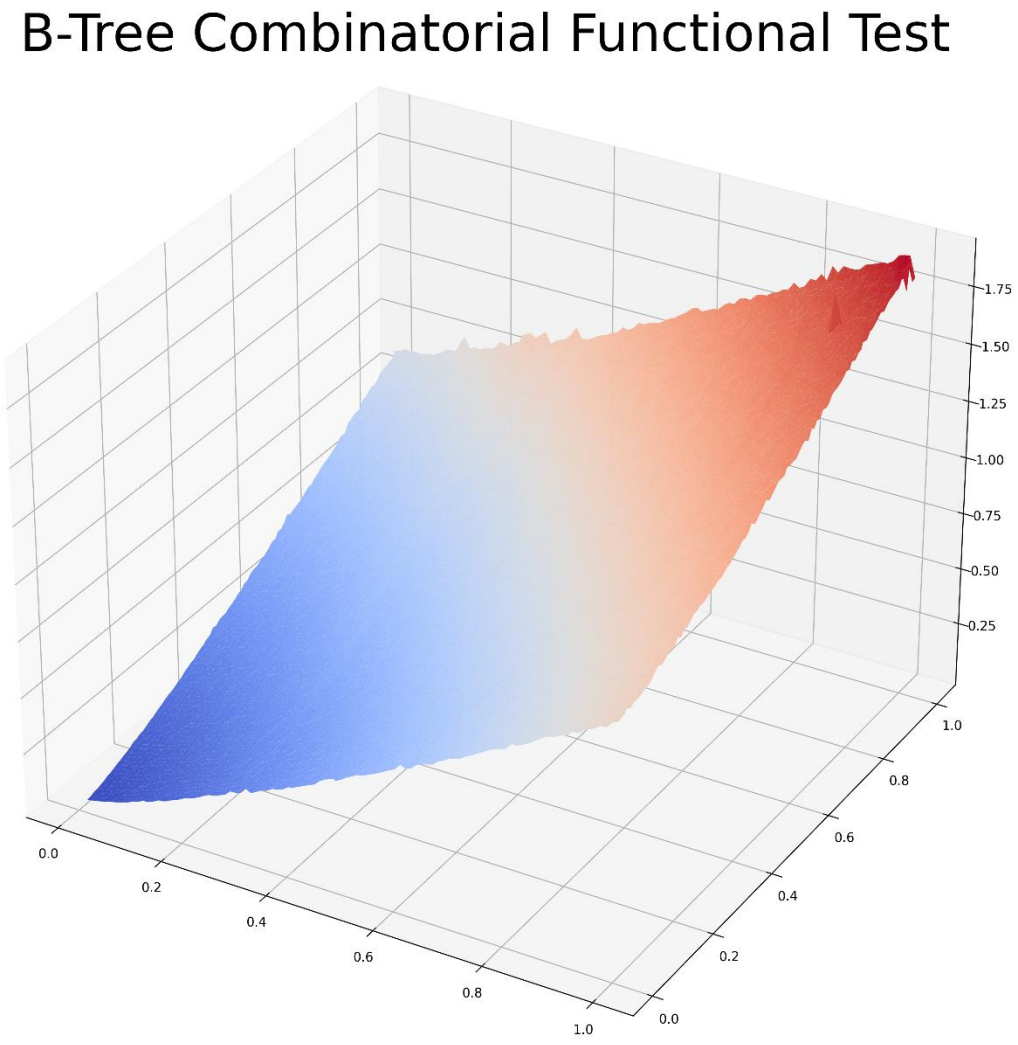


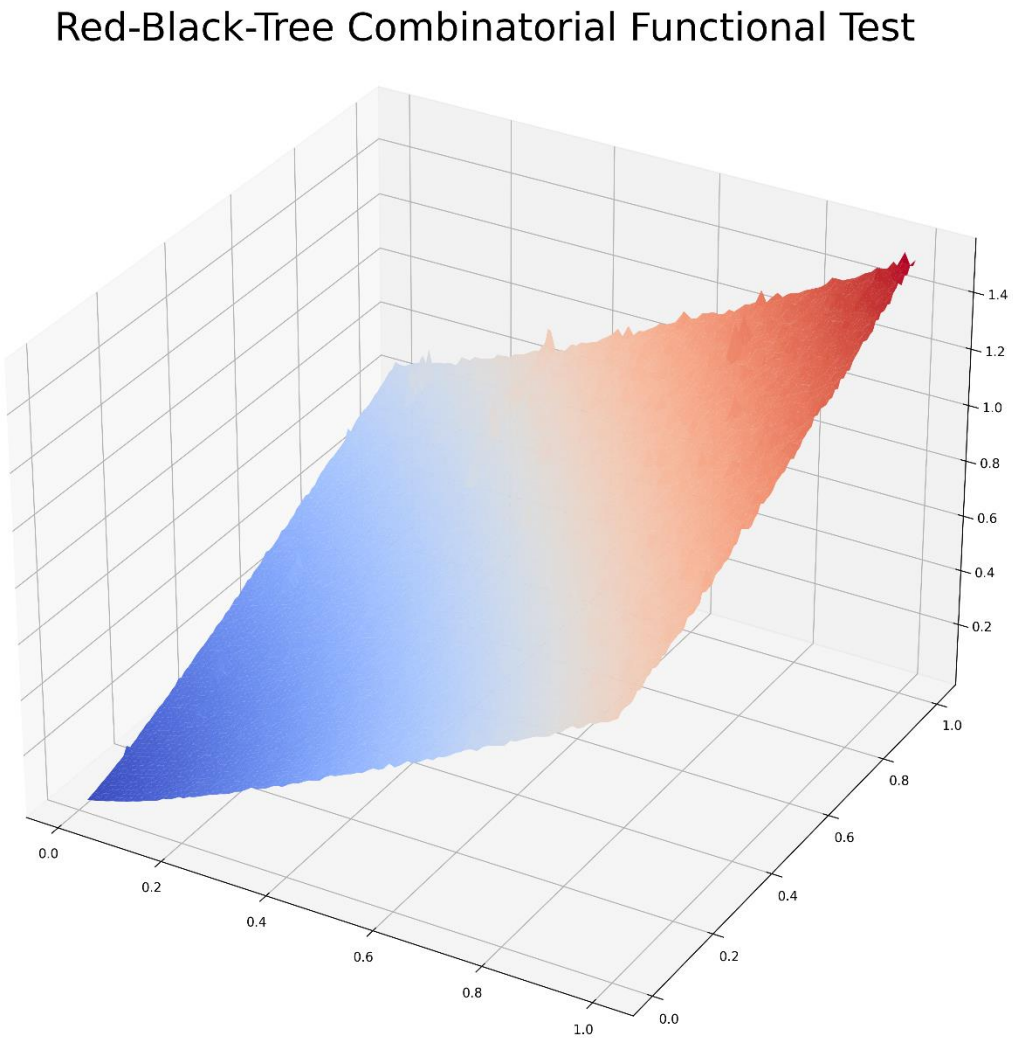
其中 B-树的耗时明显高于另外两棵平衡树，这可能是由于实现过程中 B-树采用顺序查找而非二分查找所致。值得关注的是，红黑树树的插入速度明显快于 AVL 树和 B 树，但是搜索和删除的速率却很相似，因此可以说红黑树的综合性能更好。

2. 建立动态量化关系

下面三张图是大量数据测试后得到的散点拟合出的曲面图：


AVLTree Combinatorial Functional Test
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


B-Tree Combinatorial Functional Test
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0



图中纵坐标（即 z 轴）代表操作耗费的总时间，x 轴代表$\frac{\text{删除操作次数}}{\text{插入操作次数}}$，$y$ 轴代表$\frac{\text{搜索操作次数}}{\text{插入操作次数}}$，在本项测试中，我固定了插入操作次数为 100 0000 次，调整 x 轴，y 轴所代表的比例值，试图寻找到最适合 AVL 树、B-树和红黑树的删除、搜索占比。通过调整步长为 0.01，获得足够多（1 0000 组）数据后绘制得到上述图像，每棵平衡树仅测试一次，测试平台为腾讯云 2 核 4G 轻量级应用服务器，系统为 Ubuntu 20.04-LTS。

原始数据文件见工作目录下`/res/avl_comb.txt`，`/res/btree_comb.txt` 和

/res/rbtree_comb.txt，分别对应 AVL 树，B-树和红黑树的测试结果。

这三张图都有着相同的结构，即一个平行四边形平面，这与我们预期一致，因为必然搜索和删除的次数越多越耗费时间，但是从这三张图中，我们并不能发现任何搜索、删除的占比影响时间的因素，换言之，我们的结论是**从实验的结果看来，AVL 树和红黑树在删除和搜索的耗时几乎没有任何差距(5%以内)，甚至操作系统任何一点轻微的扰动带来的差距都有可能比这大。**

但是从之前的单项测试中我们也可以看出**无论是插入还是删除，红黑树相对于 AVL 树都要略胜一筹，但优势仅限于这么可见的一点点。**作为旋转型平衡树的代表，两者的效率并没有本质上的差异。

测试中调用了 Python 的 matplotlib、numpy 库以画出上面的各类统计图。

同时为了不占用本机资源，本次测试的大部分程序都跑在远程服务器上。

四、设计亮点（选填）

1. 自动测试工具，随机生成数据进行自动化测试；同时生成了各类统计图形，可以直观的观察不同操作比例下三种平衡树的性能，便于分析；

2. AVL 树和红黑树的设计额外满足了支持重复元素，以及实现名次树接口的功能，简而言之就是实现了下列四个函数

```
int searchNext(int key, Node *o, int ans);
int searchPrev(int key, Node *o, int ans);
int kth(int k, Node *o, int rk);
int rank(int key, Node *o);
```

即支持搜索当前元素的前驱和后继，支持查询当前元素排名，支持查询排名第 k 位的元素

五、实验总结

（1）在实验中遇到了哪些问题？是如何解决的？

1. 首先最困难的就是三棵平衡树尤其是 B-树和红黑树的编写，因为之前很少会去手写 B-树和红黑树，接触更多的平衡树是伸展树 Splay 和树堆 Treap，因此学习 B-树和红黑树是一大难题，解决方法是上网查找相关博客、文章进行学习；

2. 然后就是如何测试自己的平衡树的正确性，这里我找到了一个非常好的网站 Data Structure Visualization (usfca.edu)可以可视化三种平衡树的建立、查找、删除的过程，可以根据这个网页的行为来判定自己的平衡树的行为是否正确；

3. 测试程序的编写就是非常自然的，调用相关的库函数以生成不重复的随机数；测试时最大的问题是**内存泄漏**，在进行小规模数据测试时，忘记回收的内存占用不会非常严重，但是当测试数据达到百万甚至千万的规模时，内存泄漏非常严重就会导致程序崩溃，在本机测试时表现为电脑卡顿、白屏，在服务器端测试时表现为进程直接被操作系统 Kill 掉，解决方案就是删除节点时一定记得及时 `free` 掉 `malloc` 或者 `calloc` 分配的内存空间

（2）请简要地总结一下自己在数据结构课程中各方面的收获。

1. 最主要的收获是熟悉了 Python 语言

2. 基础数据结构：栈、队列、树、图……；基础算法：KMP、Dijkstra、Kruscal……；

3. 自己动手写了一个非常原始的爬虫，掌握了基本技术；

3. 通过大作业对平衡树有了更新的认识，学习了新的平衡树写法

（3）你对课程的教学、实验等环节有没有自己的建议？（比如课程知识点安排、实验题目难易等等）

1. 改进理论课教学，多用**动图**和**真实可以运行的代码**，而非使用伪代码纸上谈兵，上面我所提到的网站完全可以在课堂上动态模拟数据结构的建立和操作过程；**更新课件和幻灯片**，替换掉代码中过时的内容和一些明显的错误；

2. 改进实验课上机，个人感觉通过上机题目仅仅掌握了数据结构的建立和操作，而对于数据结构在解决算法题目等的应用没有得到更深入的认识和理解，这方面题目可以作为课后作业布置，以增强代码能力