数据结构实验报告

软件学院 软件工程4班 2113850 李鹏

——实验题目：查找树的性能对比（时间性能）

1. **整体思路及算法：**
2. 二叉搜索树相关概念：

左子树上全部结点值均小于根结点

右子树上全部结点值均大于根结点

结点的左右子树本身是一棵二叉搜索树

二叉搜索树中序遍历所得结果是递增排序的结点序列。

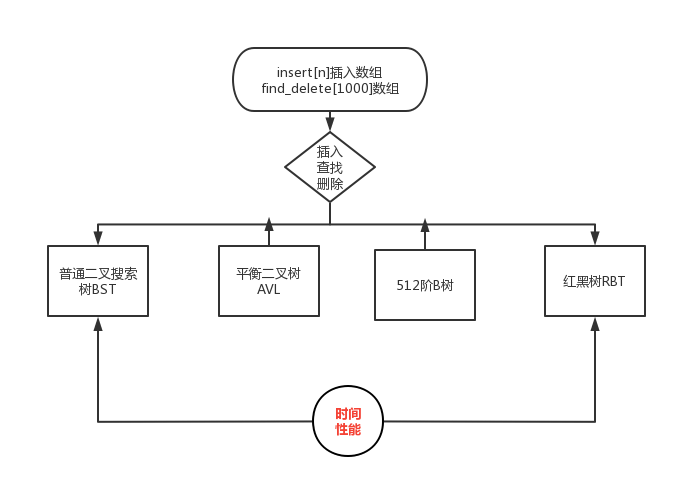
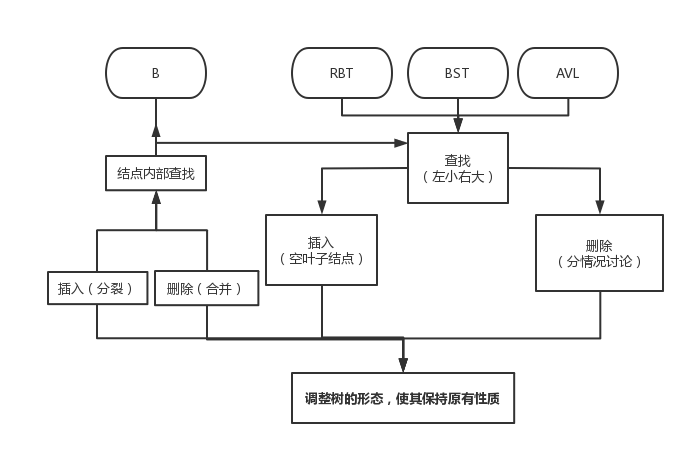
2.分别构建普通二叉搜索树（**BST）**，平衡二叉搜索树（**AVL**），512阶B树，红黑树（**RBT**），实现**插入、查找、删除**的操作

插入、查找、删除操作样例分为以下情况：

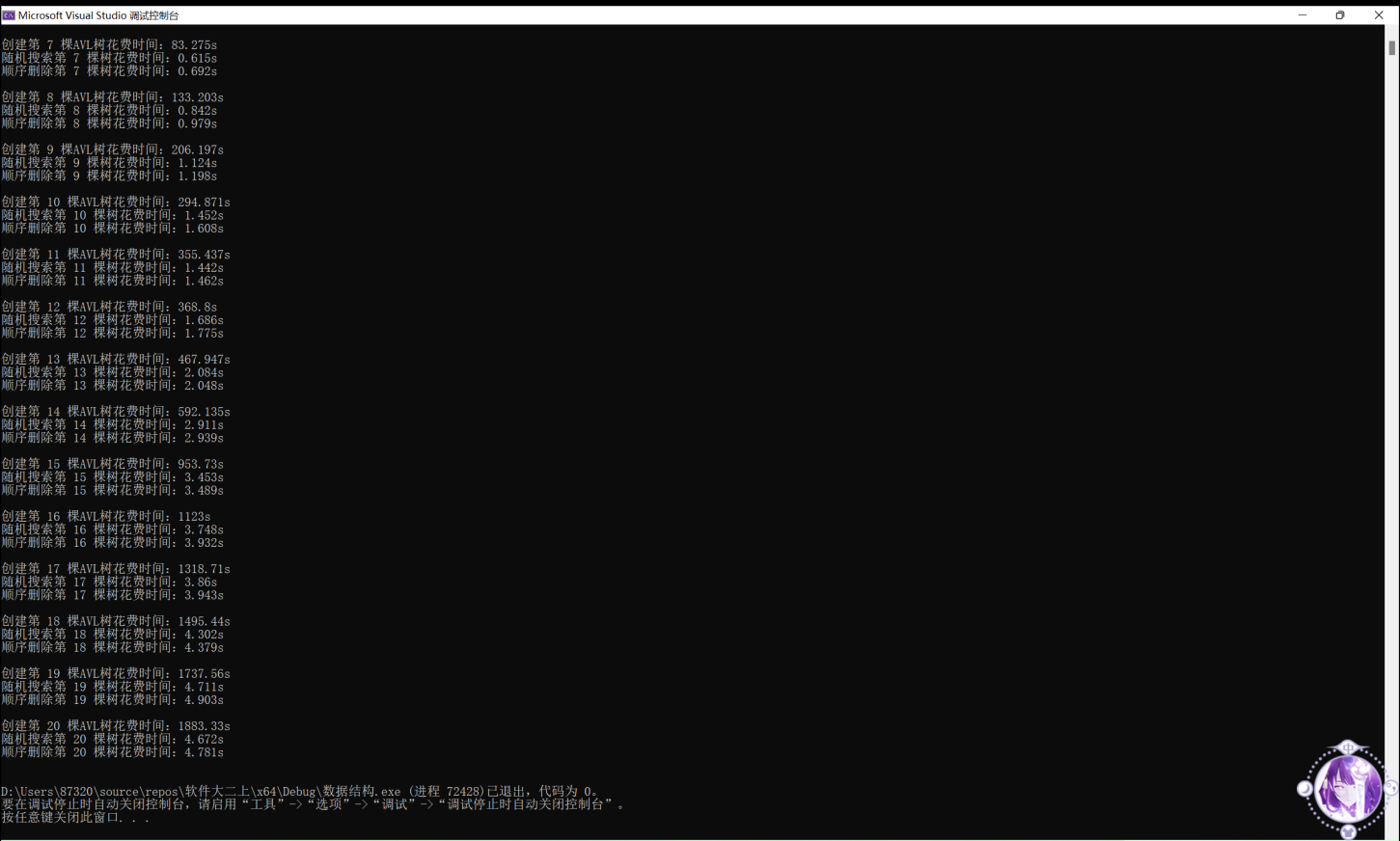
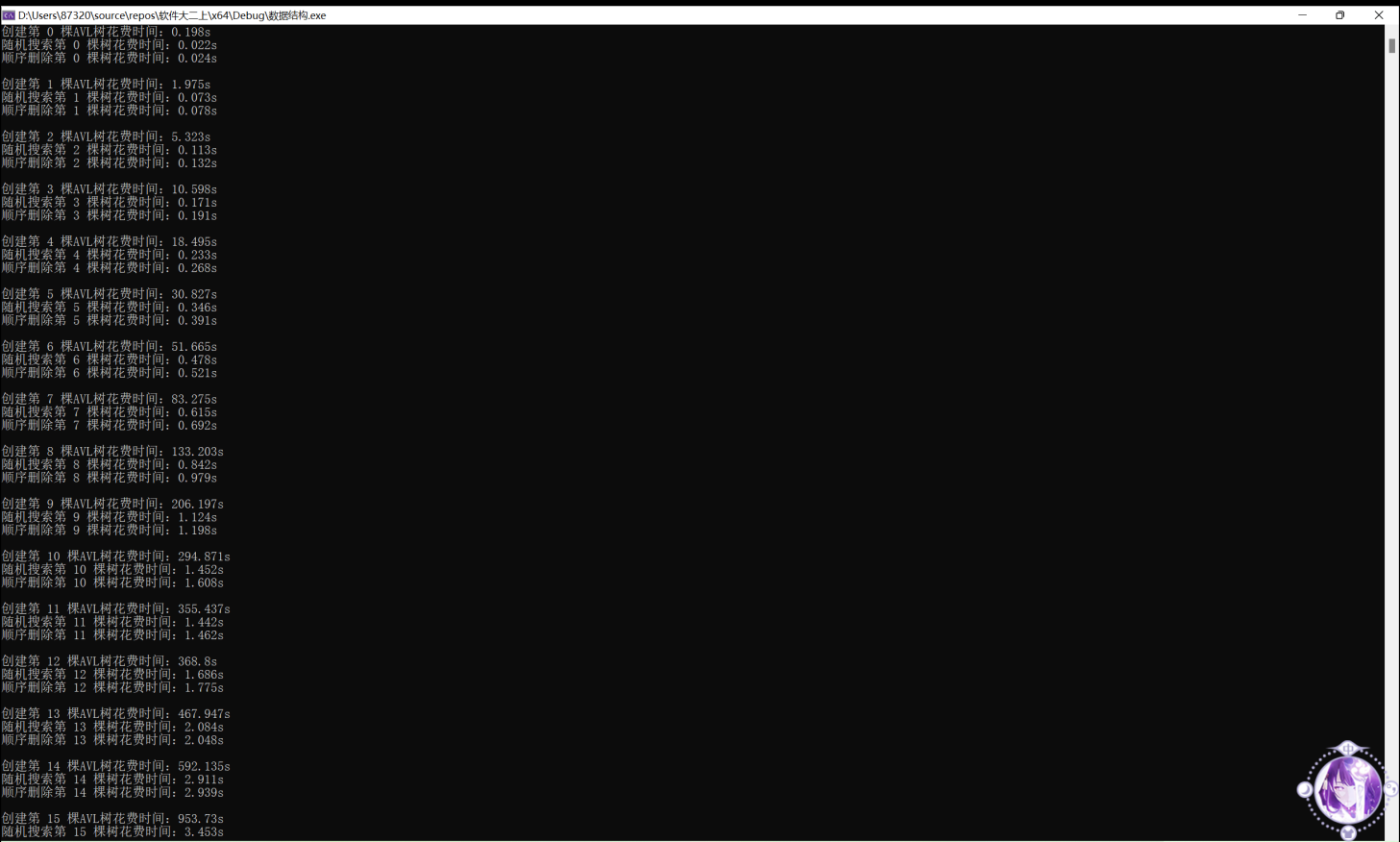
1. 递增插入N个整数，随机查找1000个数，按相同顺序删除
2. 递增插入N个整数，随机查找1000个数，按相反顺序删除
3. 随机插入N个整数，随机查找1000个数，按随机顺序删除

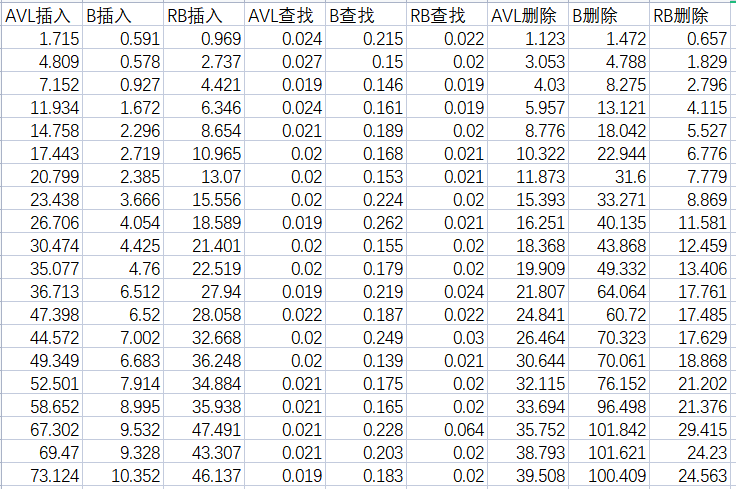
其中，N从10000到400000取值，间隔20000，以数据规模N为横轴，以运行时间为纵轴，分别画出普通二叉搜索树（**BST）**，平衡二叉搜索树（**AVL**），512阶B树，红黑树（**RBT**）的时间性能图。

1. **流程图：**

1. 测试结果：

**测试过程**

**测试数据**

**测试图表**

1. **分析总结：**
2. 理论分析：
3. **BST：**
4. 插入代价：

普通二叉搜索树插入新结点，需要查找到插入该结点的位置，所需代价与查找一个不存在的结点值相同。故BST插入操作的时间复杂度为O(logN)～O(N)。

1. 查找代价：

普通二叉搜索树的查找过程都需要从根结点出发，沿某一个路径朝叶子结点前进，所以查找中数据比较次数与树的形态密切相关。 当树中每一个结点左右子树高度大体相同时，树高为logN。平均查找长度与logN成正比，查找的平均时间复杂度为O(logN)。当前后插入的关键字有序时，BST退化成单支树结构，此时树高n。平均查找长度为(n+1)/2，查找的平均时间复杂度为O(N)。故BST查找操作的时间复杂度为O(logN)～O(N)。

1. 删除代价：

普通二叉搜索树删除一个结点，首先须要定位到该结点，这个过程需要一次查找O(N)。找到该结点的位置后，若是被删除结点的左、右子树只有一个存在，则改变树形态的代价仅为O(1)。若是被删除结点的左、右子树均存在，只须要将当P的左孩子的最右右结点与该结点互换，直接删除叶结点即可。故BST删除操作的时间复杂度为O(logN)～O(N)。

1. **AVL：**
2. 插入代价：

AVL必须要保证严格平衡(|bf|<=1)，那么每一次插入数据使得AVL中某些结点的平衡因子超过1就必须进行旋转操作。事实上，AVL的每一次插入结点操作最多只须要旋转1次(单旋转或双旋转)。故AVL插入操作的时间复杂度为O(logN)～O(N)。

1. 查找代价：

AVL是严格平衡的BST（平衡因子不超过1）。那么查找过程与BST同样，只是AVL不会出现最差状况的BST(单支树)。故AVL查找操作的时间复杂度为O(logN)。

1. 删除代价：

AVL删除结点的算法能够类似BST的删除结点，可是删除以后必须检查从删除结点开始到根结点路径上的全部结点的平衡因子。所以删除的代价稍微要大一些。每一次删除操作最多须要O(logN)次旋转。故AVL删除操作的时间复杂度为O(logN)。

1. **512阶B树**：
2. 插入代价：

B树的插入操作会发生结点的分裂操作。

1. 查找代价：

B树是一种平衡多路查找树(m-叉)，高度很小，故B树的查找时间复杂度很低。

1. 删除代价：

B树的删除操作会发生结点的合并操作，可能会回溯到根结点。

1. **红黑树**：
2. 插入代价：

红黑树插入结点时，需要旋转操作和变色操作。插入结点最多只需要2次旋转，再根据情况染色。故红黑树插入操作的时间复杂度为O(logN)。

1. 查找代价：

因为红黑树的最长路径长度不超过最短路径长度的2倍，所以红黑树虽然不像AVL同样是严格平衡的，但平衡性能仍是要比BST要好。故红黑树查找操作的时间复杂度为O(logN)。

1. 删除代价：

红黑树的删除操作代价要比AVL要好的多，删除一个结点最多只须要3次旋转操作，但仍需进行查找。故红黑树删除操作的时间复杂度为O(logN)。

1. 实验结论：
2. 递增插入N个整数，红黑树、AVL树快，普通二叉树、B树慢
3. 随机插入N个整数，红黑树、AVL树快，普通二叉树、B树慢
4. 随机查找1000个数，B树、AVL树快，红黑树、普通二叉树慢
5. 按相同顺序删除，普通二叉树最快，AVL树、B树、红黑树慢
6. 按相反顺序删除，红黑树最快，普通二叉树、AVL树、B树慢
7. 按随机顺序删除，红黑树、B树快，普通二叉树、AVL树慢