四种搜索树性能比较

2313652 郝硕洋 软件学院

1. **实验目的**

对一般的二叉搜索树，AVL树, B（512阶）树，红黑树分别实现指定操作，并分析比较这些数据结构对应一系列插入和删除操作的效率。要求测试对N个不同整数进行下列操作的效率。

1. **实验方案**

采用分别测试的方法，单独测试每一种情况下程序用时。最后以操作数规模N为横轴，运行时间为纵轴，画出每种树对应的运行曲线作为对比。

1. **二叉搜索树分析**
2. BST的操作代价分析：
3. 插入

任何一个数据插入的过程都需要从根结点出发，沿某一个路径朝叶子结点前进。因此查找中数据比较次数与树的形态密切相关。 当树中每个结点左右子树高度大致相同时，树高为log（N）。则平均查找长度与log（N）成正比，查找的平均时间复杂度在O(logN)数量级上。 当先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在O(N)数量级上。

1. 查找

查找操作与插入操作基本一致。

1. 删除

若待删除结点p只有一个子树或者本身是叶子结点，删除的代价为O（1），但是若左右子树均存在，需要将p与p的左孩子的右孩子的右孩子···右孩子交换，然后删除交换到树叶的p结点，整体删除不会超过O(log N)

1. AVL的操作代价分析：
2. 插入

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需要旋转1次(单旋转或双旋转)。因此，总体上插入操作的代价仍然在O(logN)级别上(插入结点需要首先查找插入的位置)。

(二) 查找

AVL是严格平衡的BST（平衡因子不超过1）。那么查找过程与BST一样，只是AVL不会出现最差情况的BST(单支树)。因此查找效率最好，最坏情况都是O(logN)数量级的。

1. 删除

AVL删除结点的算法可以参见BST的删除结点，但是删除之后必须检查从删除结点开始到根结点路径上的所有结点的平衡因子。因此删除的代价稍微要大一些。每一次删除操作最多需要O(logN)次旋转。因此，删除操作的时间复杂度为O(logN)+O(logN)=O(2logN)

1. RBT的操作代价分析
2. 查找

由于红黑树的性质(最长路径长度不超过最短路径长度的2倍)，可以说明红黑树虽然不像AVL一样是严格平衡的，但平衡性能还是要比BST要好。其查找代价基本维持在O(logN)左右，但在最差情况下(最长路径是最短路径的2倍少1)，比AVL要略逊色一点。

1. 插入

RBT插入结点时，需要旋转操作和变色操作。但由于只需要保证RBT基本平衡就可以了。因此插入结点最多只需要2次旋转，这一点和AVL的插入操作一样。虽然变色操作需要O(logN)，但是变色操作十分简单，代价很小。

1. 删除

RBT的删除操作代价要比AVL要好的多，删除一个结点最多只需要3次旋转操作

1. B树(512)阶操作代价分析
2. 查找

B树的查找分成两种：一种是从一个结点查找另一结点的地址的时候，需要定位磁盘地址(查找地址)，查找代价极高。另一种是将结点中的有序关键字序列放入内存，进行优化查找(可以用折半)，相比查找代价极低。而B树的高度很小，因此在这一背景下，B树比任何二叉结构查找树的效率都要高很多。而且B+树作为B树的变种，其查找效率更高。

1. 插入

B树的插入需要涉及到树节点的分裂和生长。涉及到的磁盘I/O操作次数较多，插入的代价比较大。

1. 删除

B-Tree的删除会发生结点合并操作。最坏情况下磁盘访问次数是3h＝（找到包含被删除元素需要h次读访问）+（获取第2至h层的最相邻兄弟需要h-1次读访问）+（在第3至h层的合并需要h-2次写访问）+（对修改过的根节点和第2层的两个节点进行3次写访问）。

总结:

BST树在最坏情况下会退化成链表，树的高度巨大，运行效率低下， AVL 和RBT 都是二叉查找树的优化。其性能要远远好于二叉查找树。他们之间都有自己的优势，其应用上也有不同。   
**结构对比**： AVL的结构高度平衡，RBT的结构基本平衡。平衡度AVL > RBT.   
**查找对比**： AVL 查找时间复杂度最好，最坏情况都是O(logN)。RBT 查找时间复杂度最好为O(logN)，最坏情况下比AVL略差。   
**插入删除对比**：   
1. AVL的插入和删除结点很容易造成树结构的不平衡，而RBT的平衡度要求较低。因此在大量数据插入的情况下，RBT需要通过旋转变色操作来重新达到平衡的频度要小于AVL。   
2. 如果需要平衡处理时，RBT比AVL多一种变色操作，而且变色的时间复杂度在O(logN)数量级上。但是由于操作简单，所以在实践中这种变色仍然是非常快速的。   
3. 当插入一个结点都引起了树的不平衡，AVL和RBT都最多需要2次旋转操作。但删除一个结点引起不平衡后，AVL最多需要logN 次旋转操作，而RBT最多只需要3次。因此两者插入一个结点的代价差不多，但删除一个结点的代价RBT要低一些。   
4. AVL和RBT的插入删除代价主要还是消耗在查找待操作的结点上。因此时间复杂度基本上都是与O(logN) 成正比的。

**总体评价：大量数据实践证明，RBT的总体统计性能要好于平衡二叉树**。

1. **运行效率比较折折线图**

**由于BST运行时间较长，将BST三种操作单独对比，在递增情况下各种操作,BST运行时间可视为N/A。**

以上三个折线图均为随机情况下测试，随机数均由mt19937引擎生成，随机种子为0，数据范围为int类型的取值范围。

**随机情况分析**：

BST树由于高度不可控，整体运行效率最低，其余树对于高度作为了限制，整体效率差别不大。

**递增情况分析**:

BST树效率非常低，超过30s视为N/A，在图中为0，除BST递增情况下的插入用时，其余单位均为us,B树由于分裂生长操作的存在，在N = 80000数据左右会出现波浪状。

1. **反思总结**

通过实现各种树，让我更深刻的理解了数据结构的优化作用，可以将30s运行时间压缩到ms级别， 在测试时红黑树删除出现了越界问题，经过逐行修改逻辑，正确处理了所有可能的删除情况。遇到bug的时候，坚持修改一定可以完成。

代码: