二叉搜索树排序设计思路和测试说明

October 29, 2021

1. 项目设计思路:

   1. BinarySearchTree.h的实现:

      I BinarySearchTree.h的框架摘自课本4.3节的二叉查找树的代码。在能实现二叉搜索树排序的前提下，我略去了头文件中与本项目无关的类成员函数。此外我改写了printTree()函数，使得main.cpp调用该函数后即可将排序完成的数据存储在vector中。因为二叉搜索树满足:对于所有节点X，它左子树的所有项均小于X的项，右子树所有项均大于X中的项。所以只需对二叉搜索树进行中序遍历，并将每个节点用push_back()操作推入vector的尾端即可。

   2. main.cpp的实现:

      I **数据的随机生成**：用户输入进行排序的元素个数N后，程序将自动生成一个大小为N的 vector<TN> DATA。由于本程序使用的具体的模板类型是Int,因此，为了方便验证排序的结果,DATA将恰好含有[0,N-1]中的全部整数(这样排序好的数据恰好是0,1,...N-1 ，**可以方便我们的检验**)。具体的实现方式是，对于[0,N-1]中的全部整数，给其一个随机的下标p。如果vector[p]已经被赋值，则向下寻找至第一个未赋值的单元并给其赋值。

      II **Randomized_BST_sort()的实现**：根据项目要求，我们首先对DATA中的数据进行打乱。实现方式与随机生成几乎一模一样，只需给每个数赋予一个新的随机下标即可。然后，将打乱的数据按顺序插入二叉搜索树中，并调用printTree()把排序后的结果覆盖到DATA中以便之后的输出。

   3. **算法的时间复杂度**:

      二叉搜索树排序主要涉及树的建立和中序遍历两个步骤，具体分析如下:

      I 二叉搜索树的建立：对于一个有N个节点的二叉树，根据课本4.3.6的分析，任意节点的期望深度是O(log N)(以2为底，下

同)。对于每个节点，在期望深度下插入到树中，因此这一步的时间复杂度是O(N log N)。

II 中序遍历：设N个节点遍历所需时间为T(N)，则根据printTree()的代码，可得：T(N)=2*T(N/2)+1。该递推式满足主定理的第一种情况，由此得T(N)=θ(N)。

III 综上，理论上二叉搜索树排序的时间效率是O(N log N)，为了验证这个结论，程序对随机生成的DATA进行多次打乱(**默认200次**)，并调用high_resolution_clock对整个排序操作进行计时，取平均值后输出。

2．测试说明：

1．测试输入：

I 运行 make 命令，自动编译 main.cpp 并生成可执行文件 test；

II 运行 bash run 命令，**在同一行后输入一个正整数N，表示进行排序的元素的个数**。

III **注意事项：建议N的值不要太小也不要太大**。如果N的值太小，程序运行过快。由于排序运行的平均时间是以微秒为单位的，可能会输出0 microseconds的结果；如果N的值过大，程序运行会十分缓慢。**建议的N的值是50-50000。**

2．测试输出：

I 测试输出打乱排序前随机生成的数组(**恰好含有[0,N-1]中的全部整数**)、排序后的数组和二叉搜索树排序所用平均时间T(N)。具体在输出中都有详细的说明。

3．测试结论：

1．经验证，无论N取什么值，输出的排序后的DATA都是0,1,...N-1，这说明**该二叉搜索树的排序算法是正确的**。

2．对于二叉搜索树排序所用的平均时间T(N)，经多次测试得如下平均结果(部分值)：

N	T(N)/ms
100	0.006
1000	0.12
10000	1.3
100000	23

经过计算，可以发现当N较小(<500)或者过大(>10000)时，T(N)的增长速率要大于(N log N)。**对此的解释是**，虽然理论上说二叉搜

索树排序的运行时间是O(N log N)，但是它有一个明显的问题，即在中序遍历的时候用到了线性附加内存。对于printTree()中的push_back()操作，正如在Vector模板类的实现中所分析的，SPARE_CAPACITY不能过大或者过小。因此。当N过大时，不断进行push_back()操作都会明显地减慢排序的速度；当N过小时，进行push_back()操作时间在总排序的时间中占比较大，而且此时的二叉查找树不平衡的概率较大(因为节点数较少)。这些因素从而使得T(N)的增长速率要大于(N log N)。

为了验证该想法的正确性，我将BinarySearchTree.h头文件中的push_back()操作改为直接输出至屏幕(这样能保证对不同N的输出操作时间几乎都是相同的)，经多次测试得到如下平均结果(部分值)：

N	T(N)/ms
100	0.59
1000	6.8
10000	72
100000	780

由此可见，虽然T(N)增大了不少(因为输出的用时较长)，但是T(N)的增长速率要小于(N log N)。由此验证了根据理论得出的结论：平均情况下，二叉搜索树排序的时间复杂度是O(N log N)。