第六周实验报告

沈家成

2017 年 10 月 25 日

1　1077 加分二叉树

1.1　问题分析

只知道二叉树的中序遍历，这就意味着有很多种可能的情况，因此就需要在这些情况中找到分数最高的二叉树。原本根据示例的情况，想当然的认为最大的情况应该是以最小的元素作为根，这样可以充分利用大数乘大数是一个更大的数的性质。然而，后来发现这并不能找到分数最高的二叉树，原因在于叶结点的地方并不能总是达到最大。因此，考虑使用枚举法。

1.2　枚举 + 递归

利用枚举，把每个元素当作根结点，计算最高分数。利用递归的思想，以每个元素为根结点的最高分数，应该是对应的左子树最高分数与右子树最高分数的乘积加上根结点的值。因此，计算最高分数，就是递归调用计算左右子树的最高分数。

1.3 具体实现

1.3.1 求最高分数

函数功能为取一段子树的最高分数，参数为该子树在中序遍历中的起始位置和终止位置。递归终止条件为终止位置小于等于起始位置：等于则代表只有一个元素，返回该元素值；小于代表空结点，返回 1。其他情况则向下传递左子树和右子树的参数，形成递归。

1.3.2 前序遍历

由于使用了枚举，前序遍历很难在求最高分数的过程中获得，因此，只能根据最高分数对应的根结点位置来求得。为此，设置了一个数组专门用于储存每段子树中最高分数对应的根结点位置，每次求最高分数时顺便储存该位置。寻找到最高分数，即确定最合适的根结点后，再利用递归导出前序遍历。

1.4 性能优化

因为使用了枚举和递归，对系统的资源开销就特别大。分析其中，在枚举的过程中，每一次左子树和右子树都要重新计算，因此有许多重复的操作。为了充分利用这些先前的操作，设置了一个数组，专门记录每段子树的最高分数。在递归时，如果该段子树已经计算过，就直接返回值。用空间换时间，大大提高了效率。

2 1258 圣诞树 2

2.1 难点分析

实现功能并不难，但是多达 100000 的数据，意味着时间复杂度上的高要求。最容易想到的算法是遍历每个元素，与其后的元素相比较，如果有元素

比它小就计数器加一，这样的时间复杂度是 $O(N^2)$，在测试 100000 次数据的时候会有严重的延时。

2.2 可以提升的地方在哪儿

最容易想到的算法中，第一次遍历的时候其实已经可以获得整个数列的信息，但是并没有完全利用，所以之后每个元素遍历的时候，都重复了之前的操作，这就造成了效率的底下。因此，可以换种角度，设置一个数据结构，储存查看过的数组。每次查看一个元素，就在该数据结构中寻找，比它大的有多少，这就是一部分的不和谐度，再把它有序地插入之前的元素。这样，只需要寻找一个插入和查找操作都是 $O(logN)$ 的数据结构，就可以将整个数据结构降低到 $O(NlogN)$ 了。

2.3 尝试过的方法

考虑到我们现在正在学习二叉树，设想每次往二叉树里添加元素，并通过一定的方式保持顺序，便于二分查找。但是自己实现的过程中，细节部分经常出错，产生不明所以的 bug。最后虽然解决了所有 bug，但是二叉树的深度往往达不到最小，导致查找的时间复杂度依然为 $O(N)$，并没有达到想要的速度。

后来感觉这样太过复杂，于是选择改装线性表来适应需求。考虑到查找的操作最为频繁，选择用数组实现的线性表。这样查找的时间复杂度降到了 $O(logN)$，但是插入元素需要移动许多其他元素，平均下来插入操作的时间复杂度为 $O(N)$，总的时间复杂度还是 $O(N^2)$，也没有达到效果。

后来又想到数组和链表相结合，用一个主数组进行粗略的估计，再用一个次数组精确定位。尝试实现之后，又由于数组初始规模不好确定，经常出错，也不能达到最终的效果。

2.4 小结

涉及插入和查找操作的场景很多，因此需要一个插入和查找操作都是 $O(logN)$ 的数据结构。我现在还不能自己设计出一个满足这种要求的数据结构，希望在以后的学习中能学习到这样的数据结构。

3 1212 levelOrder

3.1 层次遍历

读取树的结构不难，有难度的地方在于层次遍历。层次遍历可以通过队列实现，因此自己实现一个队列即可。

3.2 遇到的问题

刚开始自己用链表实现了一个队列，有一个地方没有设计好，导致队列是否为空的判断，在读取最后一项队列任务，再加入一项队列任务后，依然返回空的错误状态。后来重新设计了队列，才解决了这个问题。因此，在设计模块后要进行充分的测试再投入使用，否则发生了问题，往往寻找问题的根源会耗费更多的时间。

4 1530 字符二叉树

4.1 复杂的地方

涉及到了前序遍历、中序遍历、后序遍历的编码解码，比较复杂。不过只要耐心地把每个功能模块调试好，最后再通过指令选择调用哪个函数即可。

4.2 递归实现

编码的功能中，输入层序遍历，用数组储存是最合适的。前序遍历、中序遍历和后序遍历都可以通过递归实现。

解码的功能中，输入中序遍历，想要直接解码是很困难的。因此，可以换一种角度，正难则反。利用完全二叉树的性质，每一种树的尺寸大小就对应一种中序遍历。因此，可以先对一个假想的层序遍历数组进行中序遍历，再把最后的结果与输入的中序遍历对应起来，就可以获得二叉树的结构了。