第八周实验报告

沈家成

2017 年 11 月 9 日

1　1039 顺序储存二叉树

1.1　需求分析

最终需要输出每个元素对应的位置，和后序遍历的结果。后序遍历可以通过递归实现，而每个元素对应的位置正好是在插入新元素的时候也需要用到的，因此可以专门开辟一个数组用于储存位置。

1.2　具体实现

设立了一个名叫 loc 的数组，储存第 i 个元素在二叉树数组中的位置。

每次插入新元素时，从 loc 数组中取出根结点的位置，再把乘 2、以及乘 2 加 1 的结果赋值给左右结点对应的 loc 数组，最后从 loc 数组中取出左右结点的位置，按顺序存放在二叉树数组中。

1.3　小结

通过设置一个额外的数组，以空间换时间，并且简化了代码的操作，这是值得的。

2 1048 二叉树遍历

2.1 注意点

题目中限制了是完美二叉树，要充分利用这个条件。

第一行的输入是结点数，不是操作数。由于第一次操作增加了三个结点，之后都是增加两个结点，因此操作数总共是结点数减一再除以二。利用完美二叉树结点数为 $2^n - 1$ 的特性，直接除以二所得到的整数就是操作数。

没有父结点的是根结点。利用这个特性，没有出现在每次操作后两个数字的结点为根结点。可以设置一个布尔数组，表示每个结点是否为根结点。每次操作中，把子结点对应的位置设成 false，最后剩下的 true 就是根结点了。

3 1221 bst

3.1 代码复用技巧

书上的代码只有 AVL 树的插入和删除一个结点的操作，而题目中有删除大量结点的操作，将其分解为一个一个删除元素就尤其重要。

在删除大量结点的时候，原本将程序设计成先尽可能多地删除结点，最后再一起调整平衡。造成的问题就是不平衡的情况非常复杂，很难保持平衡。而且一旦无法保持平衡，之后的基于平衡假设的操作也很可能出现运行错误。因此，复用已经设计好的删除操作是更好的解决方案。

每次发现需要删除的元素，只删除该元素并保持平衡，再删除下一个元素。看似有很多冗余操作，但是每次删除时都保持了平衡，免除了最后调整

平衡的麻烦，其实并没有多出多少运算量，还保证了程序的稳定性。

3.2 改造技巧

AVL 树是忽略重复元素的，但是题目中要求记录重复元素，这就需要改造 AVL 树来适应需求。有两种思路，一是把重复元素也插入到树中，而是用一个结点的成员变量来记录重复次数。

第一种思路的好处是对插入操作只需要做少许修改，但是二叉查找树左小右大的特性就被削弱了，可能造成查找、删除操作的错误。尤其是面对题目中要求的删除区间元素，等于情况的处理就会特别麻烦，因此没有选用。

第二种思路的优点是保持了二叉查找树的特性，并且与原本的 AVL 树操作相容性好。

插入操作只需要加一个插入元素与结点元素是否相等的判断，相等就将计数变量加一，直接结束即可，不相等则进入原本的流程。由于没有在物理上插入新结点，并不需要开辟新空间和调整平衡。

删除操作也只需要判断要删除的元素与结点元素是否相等，相等就变量减一，如果还有剩余的就直接退出，没有以及不相等则继续原本流程。

3.3 测试技巧

使用 OJ 评测的时候，无法重现 bug，只能自己测试。但是 OJ 评测时往往有大量操作，自己手动不太现实。因此，需要编写测试程序。

为了测试程序的稳定性，可以用成千上万次的随机操作来考验它。随机数的种子应该设为固定值，这样出现问题之后，可以重现。随机数可以用来生产操作和数据，利用取余限定在一个范围内。gdb 是很好的调试工具，在 gdb 中运行程序，出现错误时会停在错误的地方，方便在“犯罪现场”“收集证据”。

刚开始测试时，规模应该由小逐渐增大，每次增加半个数量级为佳。出现运行错误后，首先利用 gdb 查看出错的原因，常见的有访问空地址、内存泄露等，然后在原始程序中添加输出，便于定位。

再次运行后，就可以知道出现错误的循环次数。然后在错误前十次左右设置断点，查看有没有超出预期的数据。

我就是在错误前查看了数次的运行，才发现在删除大量数据的函数中，将当前结点而不是根节点传给了删除函数，导致 AVL 树的失衡，出现了错误。这样的 bug 单单看代码是很难发现，通过测试找到问题，回溯过程，才能发现这些隐藏极深的 bug。