项目作业: 系统架构师

钟琦
混合 2103　3210103612

2024 年 1 月 5 日

1　问题描述

给出 BinarySearchTree.h,AvlTree.h,RedBlackTree.h 和 SplayTree.h 四个头文件，根据它们各自的用途, 整理它们的逻辑关系, 重构全部代码, 并用继承关系予以表达。要求至少增加一个基础虚类:BinaryTree, 作为最顶级的基类, 用以规范全部二叉树的基本操作。

2　设计思路

2.1　基类功能的选取

通过对四个头文件进行比较整理，基类 BinaryTree 需要包含以下二叉树基本功能：

- findMin/findMax：用于寻找二叉树结构中的最小/最大值
- contains：用于判断二叉树结构中是否包含值为 x 的节点
- isEmpty：用于判断二叉树是否为空
- printTree：用于打印二叉树
- makeEmpty：用于清空二叉树
- insert：用于插入某数
- remove：用于移除某数

为实现这些功能，还需要二叉树最基本的结构体 BinaryNode，包含节点值 element，左节点 BinaryNode *left 和右节点 BinaryNode *right。同时，为了用户使用通用性，将 findMin/findMax、contains、makeEmpty、printTree 这类在二叉树结构内部需要辅之以根节点 root 实现的函数提供公有版本接口函数和私有版本接口函数。

还需要注意的是，要将基类中的 private 改为 protected，便于派生类的访问。

2.2　对基类功能实现程度的考量

若将 BinaryTree 作为一个不可实例化的抽象基类，则可以使用纯虚函数的形式提供上述功能函数的接口，再在派生类中逐一加以实现。

然而，观察比较 BinarySearchTree 和 AvlTree，发现除了 AvlTree 会在 insert 和 remove 后加上 blance(t) 操作外其他实现基本无变化，因此为了防止复制粘贴代码造成冗余和提高代码犯错概

率，我决定将上述基本功能都在基类 BinaryTree 中加以实现。为了功能的完整性，BinaryTree 中添加了构造函数和析构函数，因此也要写上 clone 函数。

2.3 虚函数和多态的选取

上一部分主要是基于对 BinarySearchTree 和 AvlTree 两者的考量，当然，我们也要同步考虑到 SplayTree 和 RedBlackTree 的影响。在后者的两份头文件中，函数代码与前两者相比有较大的不同。一是添加了 nullNode，header 等特殊节点，会影响条件的判断；二是在实现思想上有所不同，例如 SplayTree 中的 EmptyTree 是通过反复寻找最大值并进行删除根节点的操作实现的。因此，基类中需要引入虚函数来实现多态功能。从实现思路上，**2.1** 中列举的除 printTree 外的基本功能都需要加上 virtual 成为虚函数。另外，由于 SplayTree 和 RedBlackTree 中增加了 nullNode 导致判断条件不同，所以需要覆写 isEmpty 函数，而 printTree 中调用了 isEmpty 函数，为了保证 isEmpty 调用的为自己类内的，因此也需要覆写 printTree 函数。

综上，Class BinaryTree 中基本功能函数（构造、析构、赋值重载除外）的 public 版本均需要以 virtual 形式出现，protected 版本中除 findMin/findMax 和 contains 之外也需要以 virtual 形式出现，而这三个函数不需要加 virtual 是因为 BinarySearchTree 和 SplayTree 这两个类中这三个函数与基类相同，且另外两个类的 findMin/findMax 和 contains 直接在 public 版本中实现，不需要调用 private 版本，也就不需要覆写。

2.4 各种 Node 之间的关系

BinarySearchTree 和 SplayTree 中的 Node 均由节点值 element，节点左指针 *left 和右指针 *right 组成，而 AvlTree 中为判断节点的平衡度增加了 height 这一元素，RedBlackTree 增加了 color 这一元素。所以基类节点 BinaryNode 可以写成由节点值 element，节点左指针 *left 和右指针 *right 组成的结构体，派生类直接继承或继承后再添加 height/color。

在此份头文件中，BinarySearchTree、AvlTree 和 SplayTree 的 Node 确实都通过继承实现了其节点功能，而 RedBlackTree 中由于调用节点函数的限制，若继承基类中的 Node 会造成函数调用时分不清 BinaryNode 和 RedBlackNode 而产生错误，需要对 RedBlackTree 中的函数了加以大量修改。因此直接重写了 RedBlackNode 来进行清晰的调用。

2.5 补充说明

事实上，通过上述思考在写完基类后发现其所有基础功能都已包含了 BinarySearchTree 中的功能，所以 BinarySearchTree 中可以不用写任何函数而直接加以继承。从另一方面讲，其实也可以灵活地把 BinarySearchTree 直接作为基类。

3 测试说明

3.1 测试函数正确性

在 TestTree.cpp 文件中，我参照了给出的 Test 文件，以 BinarySearchTree 为例，首先创建了 BinarySearchTree bst，用以检查 **2.1** 中列出的基本功能，利用报错输出和 printTree 输出的树来检查函数是否正确。接着：

- 测试复制赋值运算符:

BinarySearchTree<int> bst1;

bst1=bst;

- 测试拷贝构造函数:

  BinarySearchTree<int> bst2(bst1);

- 测试移动构造函数:

  BinarySearchTree<int> bst3(move(bst2));

- 测试移动赋值运算符:

  BinarySearchTree<int> bst4; bst4=move(bst3);

其余几类派生类的测试思路与此相同，输出结构在 output 文件中，经检验全部正确。

3.2 测试各类树的效率

由于 remove 的缺失，首先不测试 RedBlackTree。在 NUMS=200000,GAP=37 的条件下，BinarySearchTree、AvlTree、SplayTree 实现每一类相同操作的总时间分别为 9.37s、9.39s、0.12s，可见 SplayTree 在插入数据中有很大一部分是有序的情况下效率最佳。

在 NUMS=200000,GAP=3711 的条件下，BinarySearchTree、AvlTree、SplayTree 实现每一类相同操作的总时间分别为 0.21s、0.19s、0.21s，可见 BinarySearchTree、AvlTree 在插入数据基本乱序的情况下效率有明显改善，因此数据的输入顺序对树的操作效率有很大的影响。

然后不考虑 remove 操作以及赋值、构造等操作，在 NUMS=200000,GAP=37 的条件下，BinarySearchTree、AvlTree、SplayTree、RedBlackTree 实现每一类相同操作的总时间分别为 7.50s、7.20s、0.079s、0.079s；在 NUMS=200000,GAP=3711 的条件下，BinarySearchTree、AvlTree、SplayTree、RedBlackTree 实现每一类相同操作的总时间分别为 0.16s、0.15s、0.13s、0.09s 说明 SplayTree 和 RedBlackTree 的期望操作效率高一些。