**Linux实验报告**

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 庞竹 | 班级 | 硬件二班 |
| 学号 | 161403223 | 日期 | 2019.6.3 |

1. **实验目的**  
   1、了解Linux编程环境；掌握在Linux环境下编程的常用工具，例如：shell, vim, make, gedit, git及各种语言的集成开发环境。
2. 了解Linux系统的内存、进程、线程、同步、通信的基本原理和其在实际程序实际中的应用。
3. 掌握Linux环境下的应用程序设计、开发和项目管理。
4. 通过16学时的实验，在Linux环境下设计并实现一个代码量不小于2000的项目。
5. **实验要求**

在Linux环境下使用Java语言结合数据结构相关知识完成以下项目的具体实现：

1. 堆
2. 栈
3. 红黑二叉树
4. B+树
5. **实验内容**
6. **堆**

**（1）大顶堆与小顶堆的具体实现（以小顶堆为例，实验代码均可实现）（HeapSort.java）**

\*实现堆排序需解决两个问题：

（1）如何将n个待排序的数建成堆；

（2）输出堆顶元素后，怎样调整剩余n-1个元素，使其成为一个新堆。

\*建堆方法（小顶堆）：

对初始序列建堆的过程，就是一个反复进行筛选的过程。

（1）n个结点的完全二叉树，则最后一个结点是第n/2个结点的子树。

（2）筛选从第n/2个结点为根的子树开始（n/2是最后一个有子树的结点），使该子树成为堆。

（3）之后向前依次对各结点为根的子树进行筛选，使之成为堆，直到根结点。

\*调整堆的方法（小顶堆）：

（1）设有m 个元素的堆，输出堆顶元素后，剩下m-1 个元素。将堆底元素送入堆顶，堆被破坏，其原因仅是**根结点不满足堆**的性质。

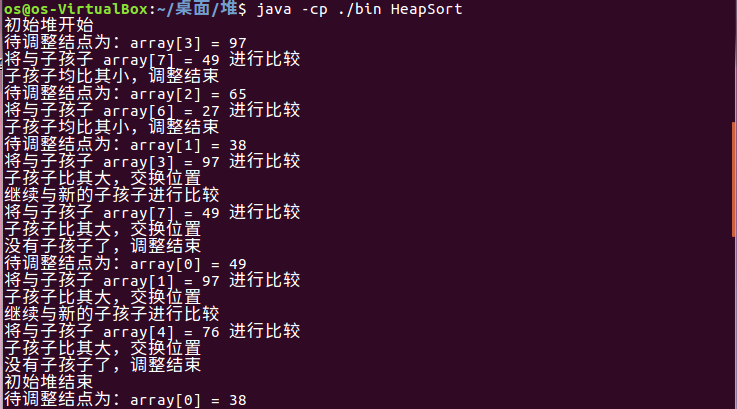
（2）将根结点与左、右子树中较小元素的进行交换。

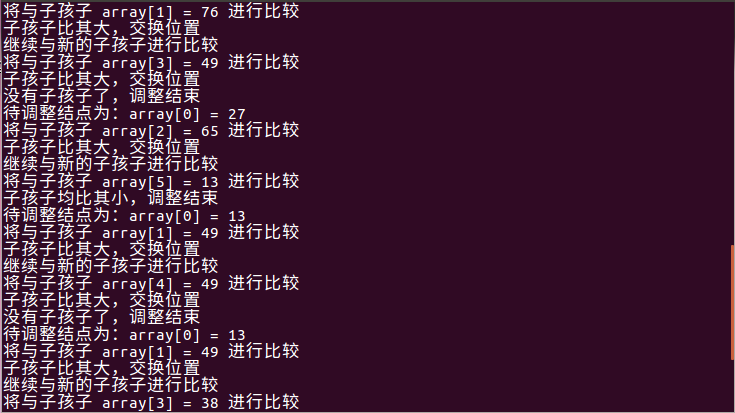
（3）若与左子树交换：如果左子树堆被破坏，则重复方法(2).

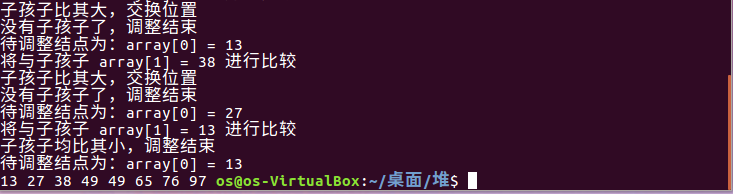
（4）若与右子树交换，如果右子树堆被破坏，则重复方法(2).

（5）继续对不满足堆性质的子树进行上述交换操作，直到叶子结点，堆被建成。

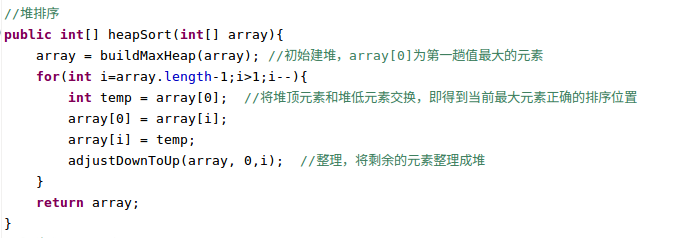
**小顶堆运行结果：**

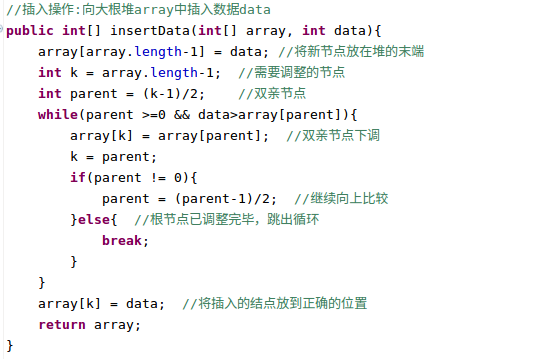




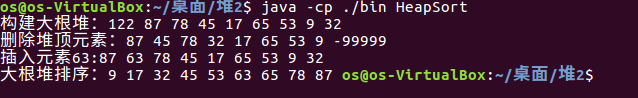


1. **堆插入及排序(HeapSort\_delete.java)：**





**运行结果：**



1. **栈（EvaluateExpression.java）**

输入运算式并以#结尾，得到运算结果。

具体过程：

\* 使用两个栈，分别是操作数栈（存储数字）和操作符栈（存储运算符）；

\* 读入表达式时，如果是操作数，则入操作数栈；如果是运算符，则入操作符栈；

\* 当运算符入栈时，和操作符栈栈顶元素比较优先级；

\* 如果优先级比栈顶元素高，则入栈,并接收下一个字符；

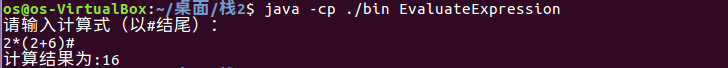
\* 如果和栈顶元素相同，则脱括号，并接收下一个字符（因为相同只有( )括号）；

\* 如果小于栈顶元素优先级，则操作数出栈，操作符出栈，并计算运算结果再入栈。

\*\*循环的退出条件 直到运算全部结束，即当前栈顶元素和读入的操作符均为#。



**运行结果：**



1. **红黑二叉树（DRBTree.java)**

实现红黑二叉树，并进行删除操作。

红黑树二叉树的四大性质：

\*红黑树节点的颜色非红即黑；

\*红色节点的两个子节点必须是黑色；

\*叶子都为黑色（这里的叶子节点是指NULL节点）；

\*每个节点到叶子节点的所有路径包含的黑色节点个数要相同。

根据以上性质完成红黑二叉树的实现及删除等操作。

\*插入时需要重新进行红黑树平衡的三种情况，假设父节点是左孩子，叔父节点是右孩子，反之为对称情况好理解：

（1）父亲节点是红色，叔父节点也是红色

需要转化为：父节点涂黑，叔父节点也涂黑，祖父节点涂红，把祖父节点赋值为当前节点，然后变为下面2,3情况一种

（2）当前节点为父节点的右孩子，叔父节点为黑色

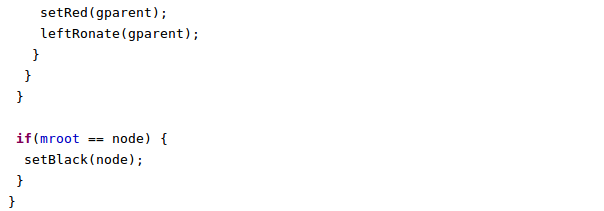
需要转化为：把父节点左旋，当前节点赋值为父节点，父节点赋值为当前节点，此时变为第3种情况

（3)当前节点是父节点的左孩子，叔父节点为黑色

需要转化为：父节点涂黑，祖父节点涂红并右旋，此时红黑树变为平衡状态







\*删除需要进行的四种情况(删除的节点可以由后继节点来填补，后继节点一般是大于被删除节点的最小节点，所以真正需要平衡转化的地方是后继节点那里）

假设当前节点（）是左孩子，兄弟节点是右孩子，反之为对称情况好理解

（1）当前节点为黑色包含空的情况（大多数为空），兄弟节点为红色

需要转化为：兄弟涂黑，父亲涂红并左旋，如果变成2情况，则不会转化3,4情况，重新循环，如果变化为3情况，则会转化为第4种情况，如果变化为4情况，红黑树重新平衡并跳出循环

（2）当前节点为黑色包含空的情况（大多数为空）,兄弟节点为黑，并且兄弟节点的左孩子和右孩子都为黑

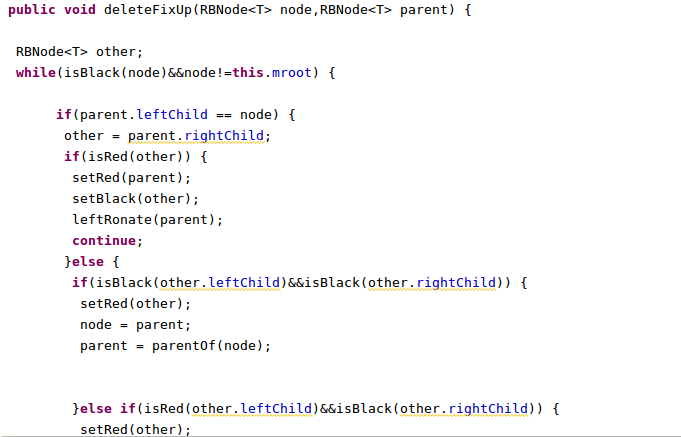
需要转化为：兄弟节点涂红，父节点赋值为当前节点，重新循环

（3）当前节点为黑色包含空的情况（大多数为空），兄弟节点为黑，并且兄弟左孩子为红色，右孩子为黑色

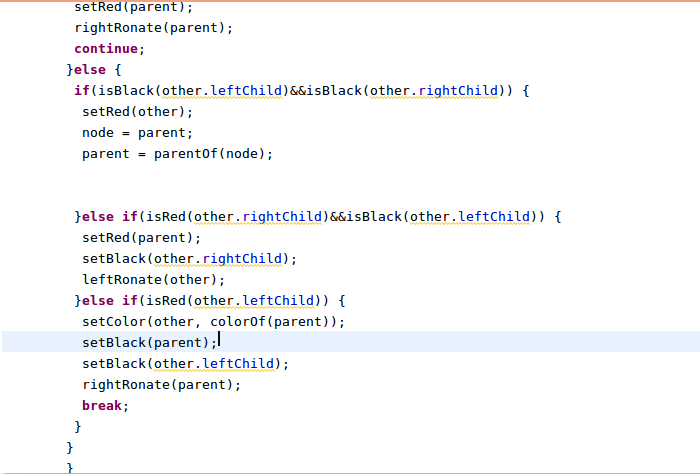
需要转化为：兄弟节点涂红，左孩子涂黑，并把兄弟节点右旋  则转化为第4种情况

（4）当前节点为黑色包含空的情况（大多数为空），兄弟节点为黑色，并且兄弟右孩子为红色，左孩子为任一颜色

需要转化为：兄弟节点涂成父节点的颜色，父节点涂黑，兄弟右孩子涂黑，并把父节点左旋，红黑树达到平衡状态

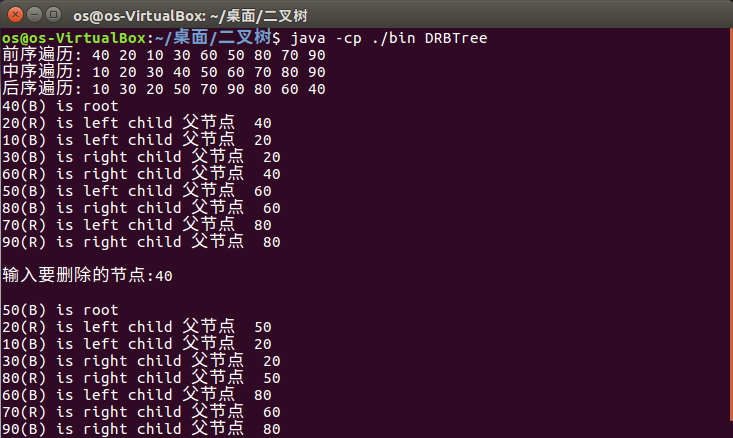








**运行结果：**



1. **B+树（BplusTree.java、Test.java、Product.java）**

实现B+数，对过程进行详细分析。

\*首先考虑的问题是数据类型，用来作为B+树索引的键的肯定是某个拥有很多个属性的对象，那么数据类型应该使用泛型；

\*类型确定用泛型之后，接下来应该思考B+树实现应该有哪些类，按照写二叉树的经验，首先考虑到的就是节点类。每个节点应该有一系列的键，键的数量取决于多种因素，那么最好采用数组。其实还应该有指向父节点和子节点的指针，其中父节点只需要有一个，而子节点有多个，同样需要采用数组。最好还有一个变量来方便得记录子节点和键的数量，这样获取节点数量时比较方便。

\*在确定了节点的属性之后，要考虑节点类会有哪些方法，首先构造方法肯定需要有的，在构造方法中完成相关属性的初始化。

节点分为叶节点和非叶节点，叶节点需要额外存储数据，所以数据结构不太一样，非叶节点也有自己的查询和插入逻辑，所以应该把节点类作为抽象类，叶节点和非叶节点都继承这个类。

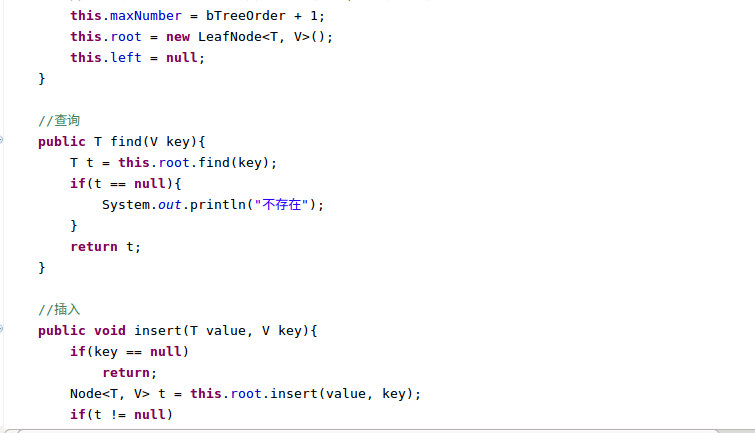
非叶节点的查询是遍历整个数组，找到对应的子节点然后进行递归查询。非叶节点的插入同样也是找到应该插入的子树进递归插入。

叶节点需要专门定义一个数组用来存储键对应的值，还需要实现具体的查询和插入方法。查询可以顺序查询，这里采用二分搜索来节约点时间。插入就比较复杂了，因为插入叶节点时需要保证整个B+树的平衡。

叶节点进行插入时，首先找到数组里面应该插入的数据的正确位置，然后把数组挪一挪把数据放进来。挪完数组后，就需要判断数据数量是否超过上限了，如果超过上限，则需要对当前节点进行分裂，奇数时左边比右边少一个数据，偶数时都一样。分裂成两个节点并且完成数据复制后，还没有结束，需要把新产生的节点插入父节点，所以非叶节点需要有一个方法来处理这种情况。因为插入父节点的同时还需要更新子节点指针，所以干脆把两个节点作为参数传过去。同时，父节点键也有很多，为了精准地进行插入，我们还需要传入旧键来弄清楚插入的两个节点应该放到哪里。

父节点的插入方法拿到了旧键和需要插入的两个节点，那么首先根据旧键找到了应该插入的位置，然后新的两个键取代旧的一个键，新的两个指针取代旧的指针，又是疯狂地挪位置。完成插入之后，和叶节点的插入类似，也需要判断是否超出上限了，如果超出了上限那么同样需要进行拆分，拆分也和子节点拆分类似，拆完了递归调用自身，这样就能够保证B+树始终是平衡的。

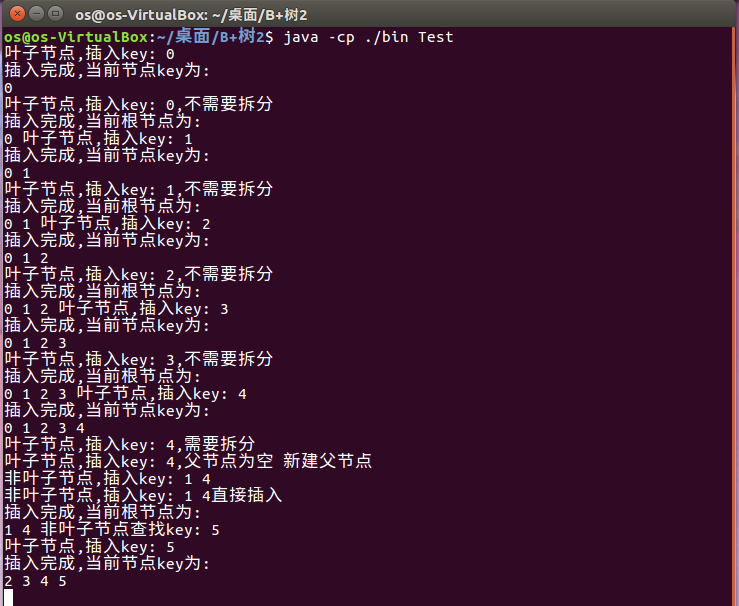






之后是关于节点父类、叶子节点类、非叶子节点类的具体实现，详情见代码。

**运行结果：**



1. **实验总结**

本次实验使用Java实现了数据结构四种经典构造及其应用，加深了我对数据结构的理解及对Java语言的使用，也让我对Linux环境更为熟悉，提高了我的学习能力及动手能力。