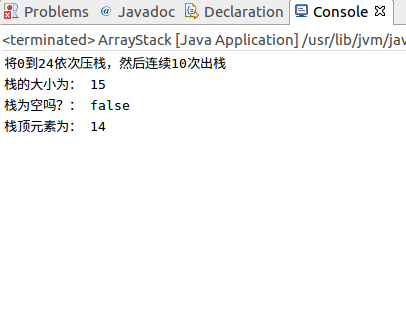
**Linux实验报告**

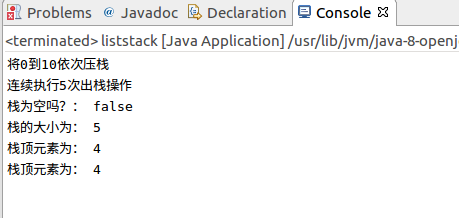
1. **实验目的：**
2. 熟练掌握堆，栈，B+树，红黑二叉树四种数据结构
3. 可以利用编码实现这四种数据结构及其相关应用
4. 能够在Linux中顺利调试运行出代码
5. **实验内容：**
6. **栈**

栈（stack）又名堆栈，它是一种运算受限的线性表。其限制是仅允许在表的一端进行插入和删除运算。这一端被称为栈顶，相对地，把另一端称为栈底。向一个栈插入新元素又称作进栈、入栈或压栈，它是把新元素放到栈顶元素的上面，使之成为新的栈顶元素；从一个栈删除元素又称作出栈或退栈，它是把栈顶元素删除掉，使其相邻的元素成为新的栈顶元素。栈在java中的实现有两种

1. 数组实现：



1. 链表实现：



栈中数据的增删具有特殊性: 只在栈顶入栈和出栈。也就是说数组实现的栈在增加和删除元素时并不需要移动大量的元素, 只是在数组扩容时需要进行复制。而链表实现的栈入栈和出栈时都需要将数据包装成Node或者从Node中取出数据, 还需要维护栈顶指针和前驱指针。所以数组方法的出栈入栈速度要优于链表

1. 栈的应用

3.1平衡符号 括号的匹配 利用堆栈来处理，符合条件就弹出，否则push进去



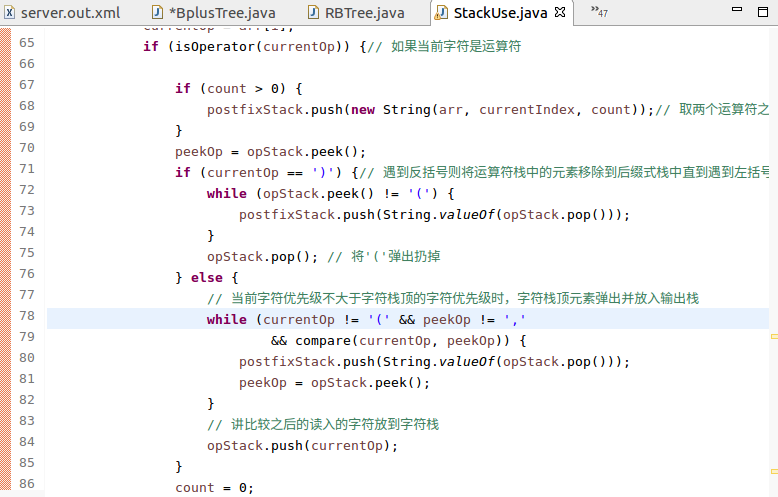
3.2将表达式转换成为后缀式，栈转换的规则是：

1、将符号放入符号栈中，将元素输出

2、若符号栈中顶部的元素的优先级小于下一个读入的符号的优先级，则下一个符号进栈

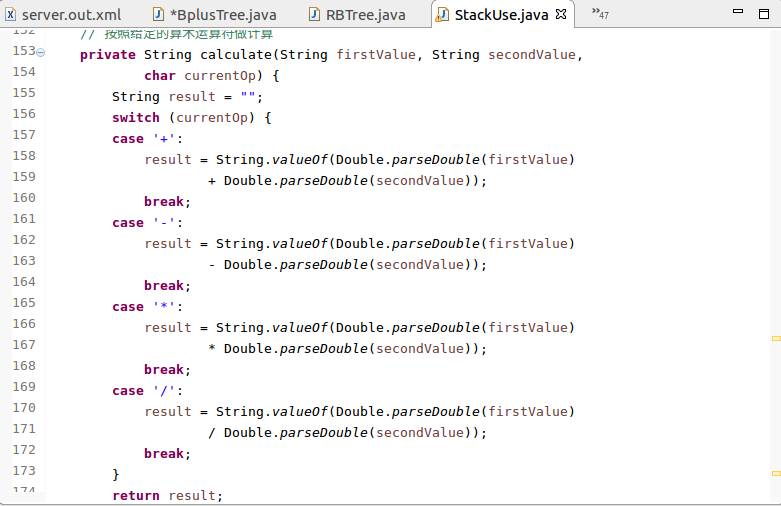
3、若符号栈中顶部的元素的优先级大于下一个读入的符号的优先级，则将符号栈中的符号弹出，直到遇见小于这个读入符号优先级的符号 然后这个读入符号进栈

4、对于右括号")"，它具有最高的优先级，直到遇见"("左括号，将这两个符号同时扔掉，继续向下读取

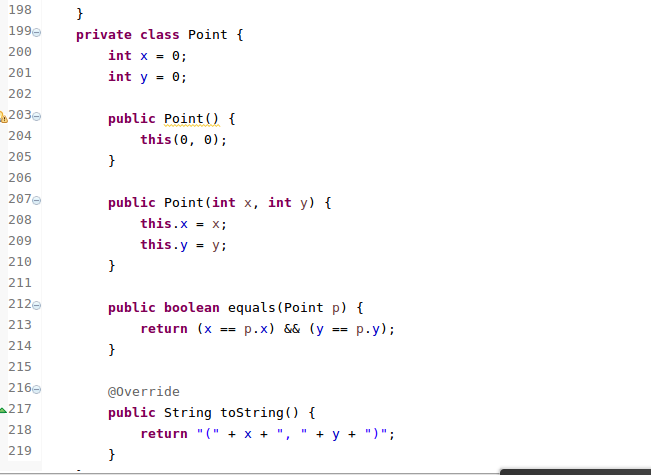




3.3计算后缀表达式

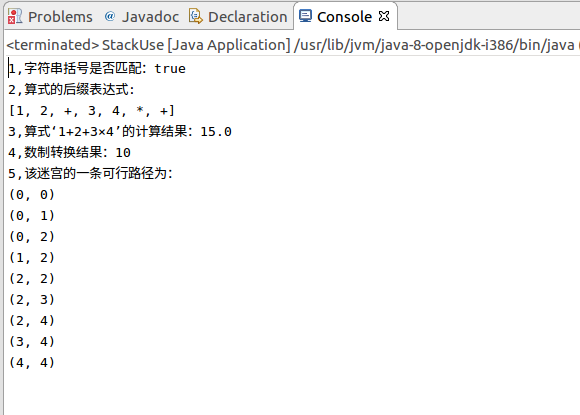


3.4数制转换



3.5求解迷宫路径

运行结果图：



**2、堆：**

堆通常是一个可以被看做一棵树的数组对象。堆总是满足下列性质：

1.堆中某个节点的值总是不大于或不小于其父节点的值；

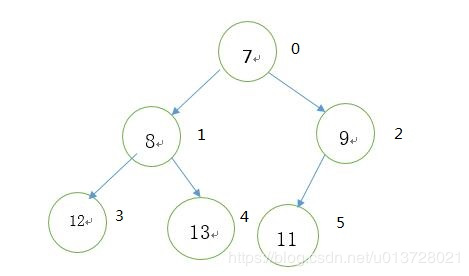
2.堆总是一棵完全二叉树。

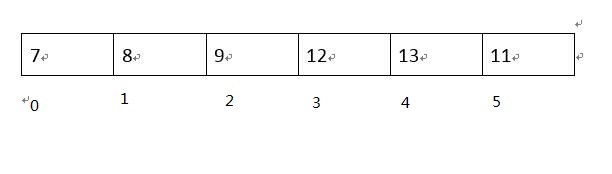
堆的定义：n个元素的序列{k1,k2,ki,…,kn}当且仅当满足下关系时，称之为堆。

(ki <= k2i,ki <= k2i+1)或者(ki >= k2i,ki >= k2i+1), (i = 1,2,3,4…n/2)

小根推：堆是一颗完全二叉树，在这棵树中，所有父节点都满足大于等于其子节点的堆叫大根堆，所有父节点都满足小于等于其子节点的堆叫小根堆。

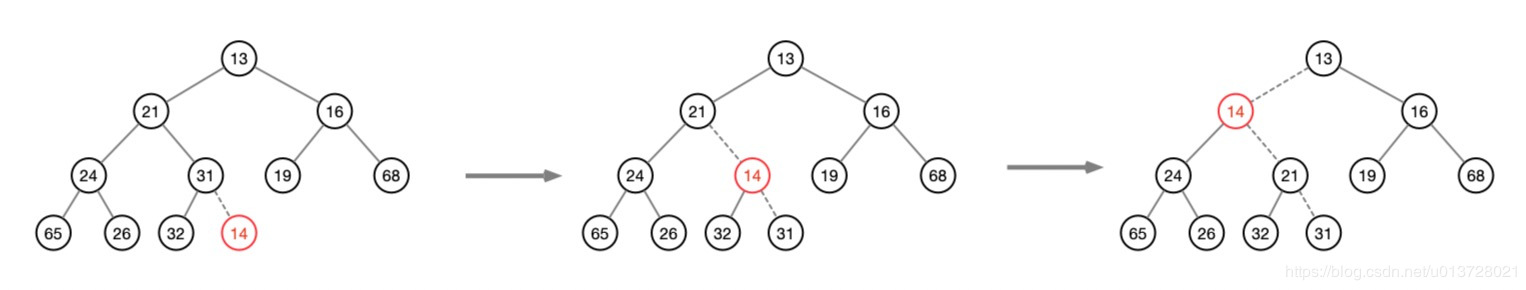
通常存放在一个数组中，父节点和孩子节点的父子关系通过数组下标来确定。



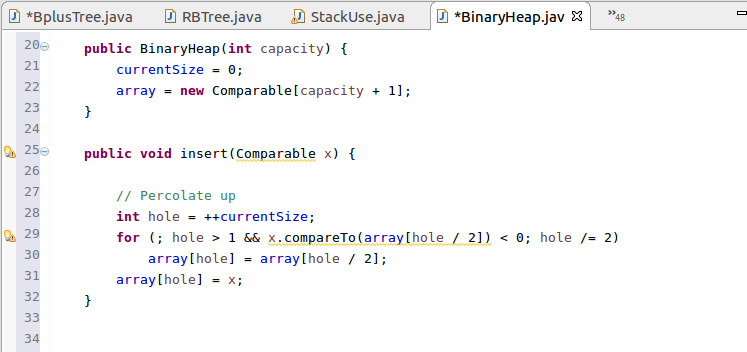


小根堆及其存放数组

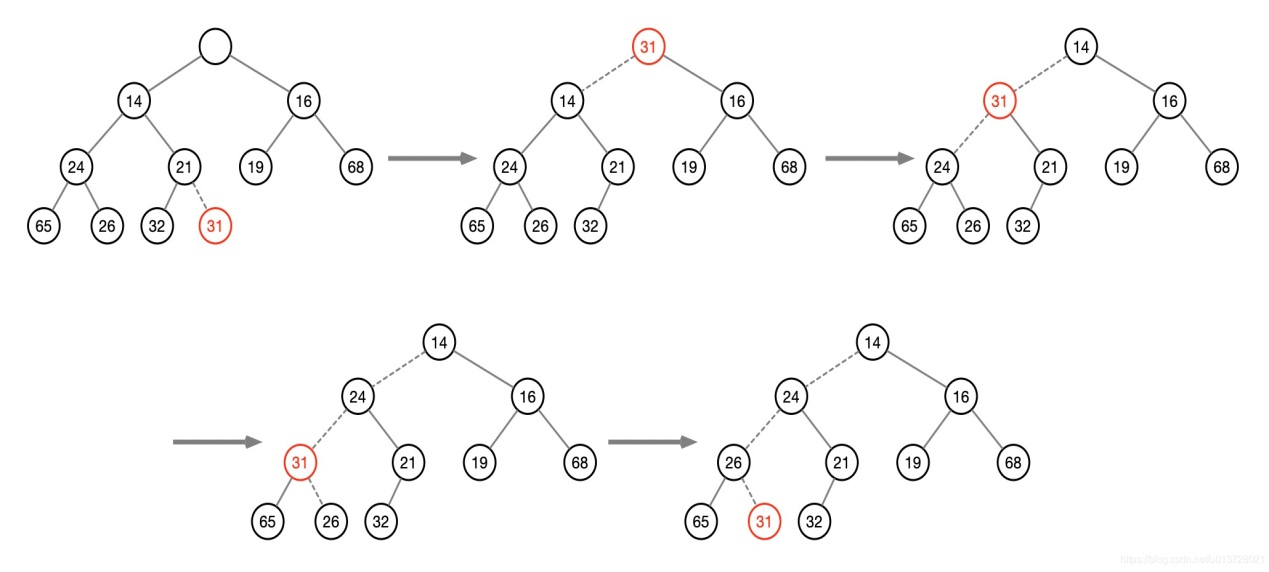
1. 堆的插入结点过程：



代码实现：



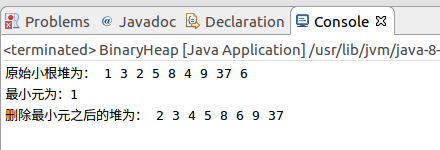
1. 删除最小元过程：



代码实现



运行结果：



删除了最小元1后的结果是：2 3 4 5 8 6 9 37

**3、B+树**：

（1）B+树相关定义：

B+树是一种多路平衡查找树,是对B树(B-Tree)的扩展.（涉及B树的相关知识，报告中就不做一说明了）

B+树基本定义与B树相同,不同之处在于:

1. 非叶节点仅有索引作用，具体信息均存放在叶节点;
2. 树的所有叶子节点构成一个有序链表，可以按照键的排序次序遍历全部记录

B+树主要应用于数据库索引

（2）B+树相较于B树的优点：

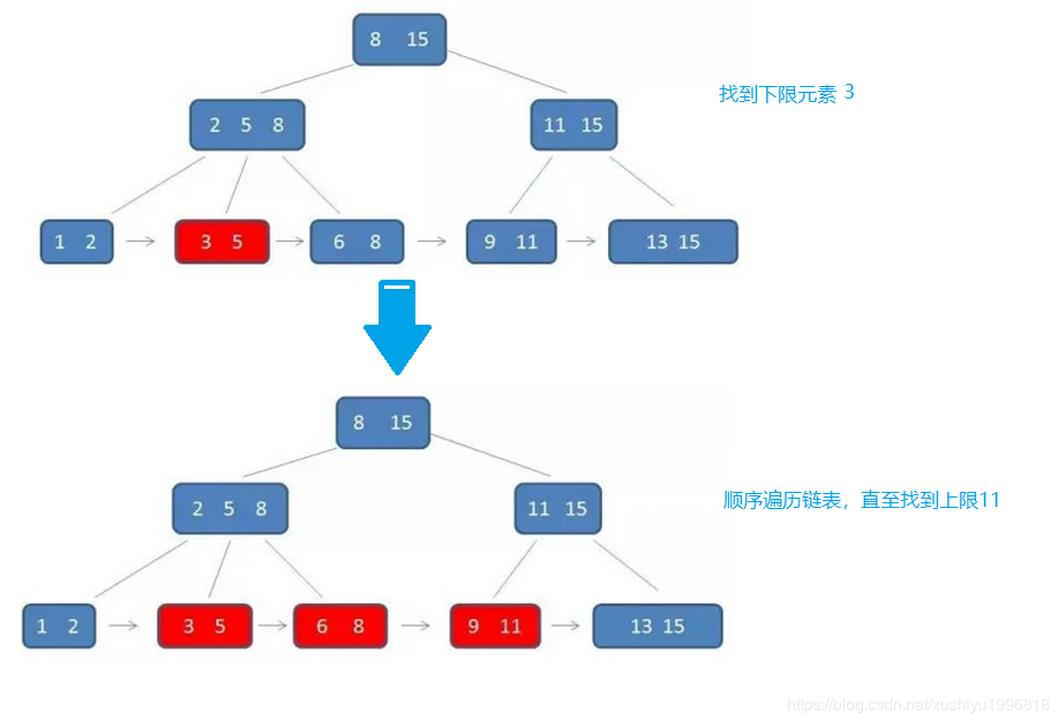
1. B+-tree的磁盘读写代价更低
2. B+-tree的查询效率更加稳定

1.单一节点存储更多的元素，使得查询的IO次数更少；

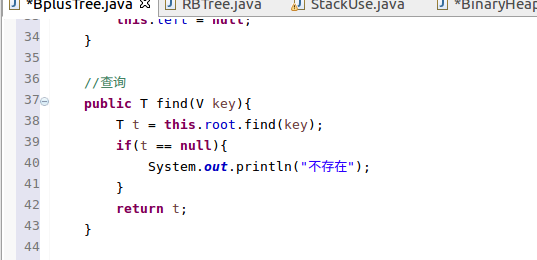
2.所有查询都要查找到叶子节点，查询性能稳定；

3.所有叶子节点形成有序链表，便于范围查询。

（3）B+树的查找过程：



相关代码：





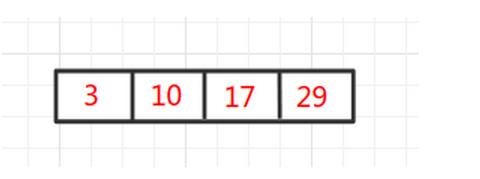
（4）B+树的插入过程：

B+树的插入与B树的插入过程类似。不同的是B+树在叶结点上进行，如果叶结点中的关键码个数超过m，就必须分裂成关键码数目大致相同的两个结点，并保证上层结点中有这两个结点的最大关键码。

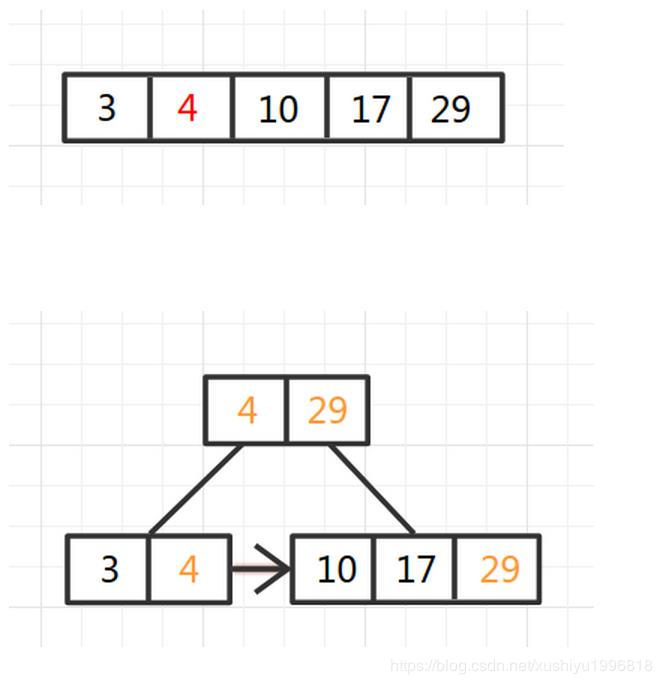
在增加值的时候，如果存在满员的情况，将选择结点中的值作为新的索引。例如：

建立一个阶为4的B+树,随机一些数据进行插入10,17,3,29,4,5,18,6,22

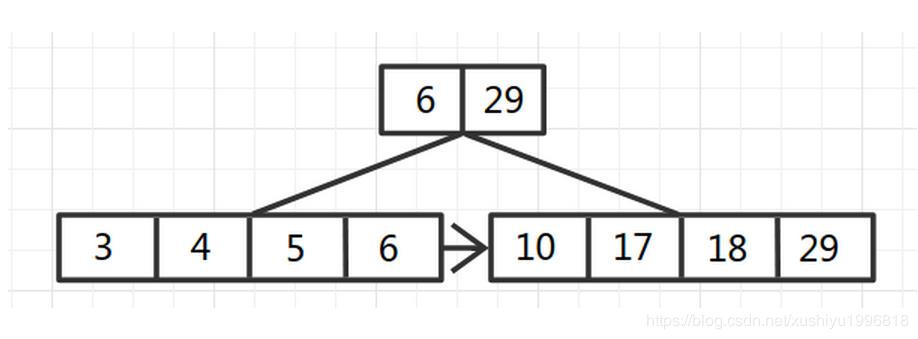
首先,依次插入10,17,3,19,都能存放在初始节点中,在插入时会查找到正确的位置并且进行插入:



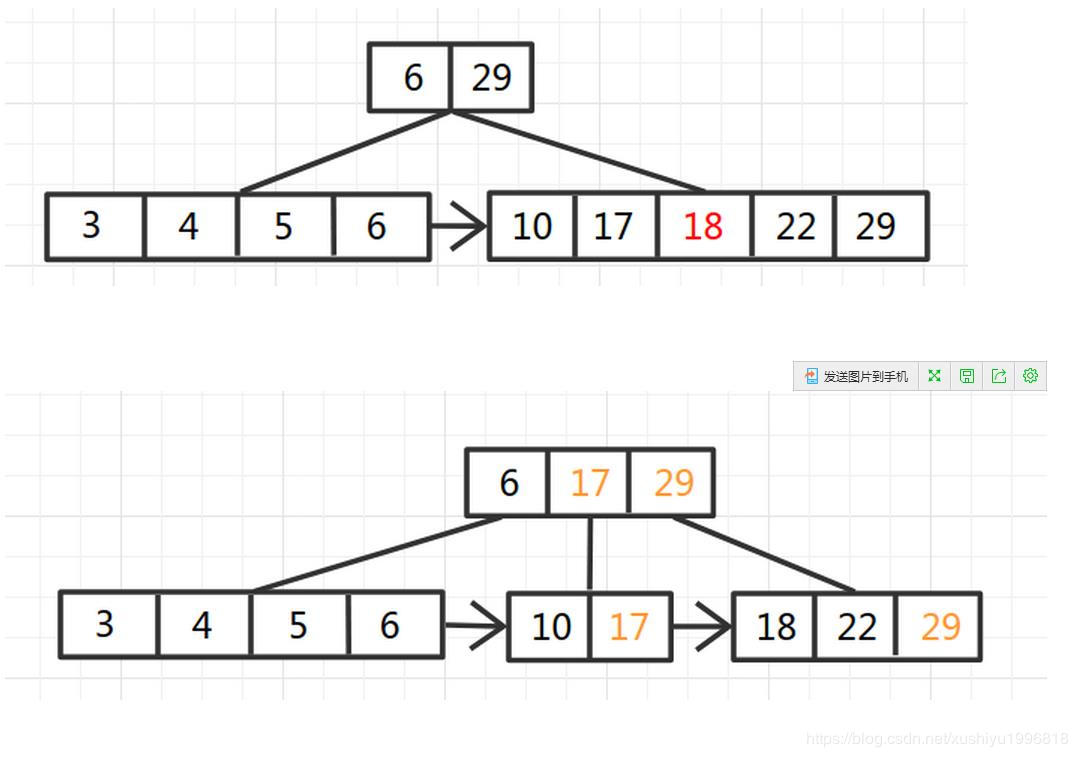
之后插入4,插入成功后发现当前节点的键的数量为5,大于了最大值4,所以需要从中间拆分为两部分,同时把拆分后的两个节点最大的键取出来插入到父节点中(图中橙色节点):



之后继续插入5,18,6都能够成功插入:

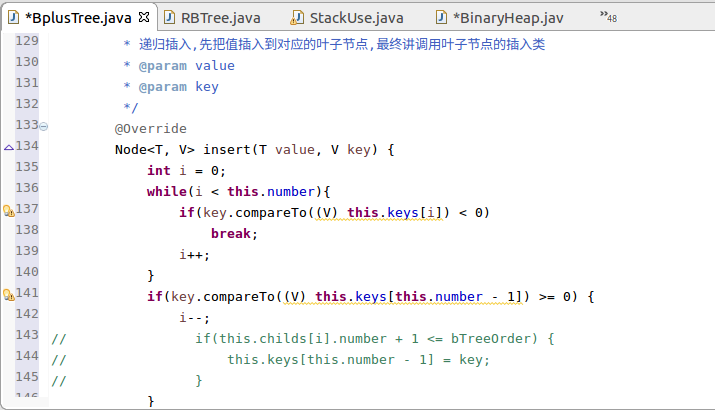


之后插入22,叶子节点超过上限,进行拆分,拆分后仍然将拆分的两部分的最大值插入到父节点:

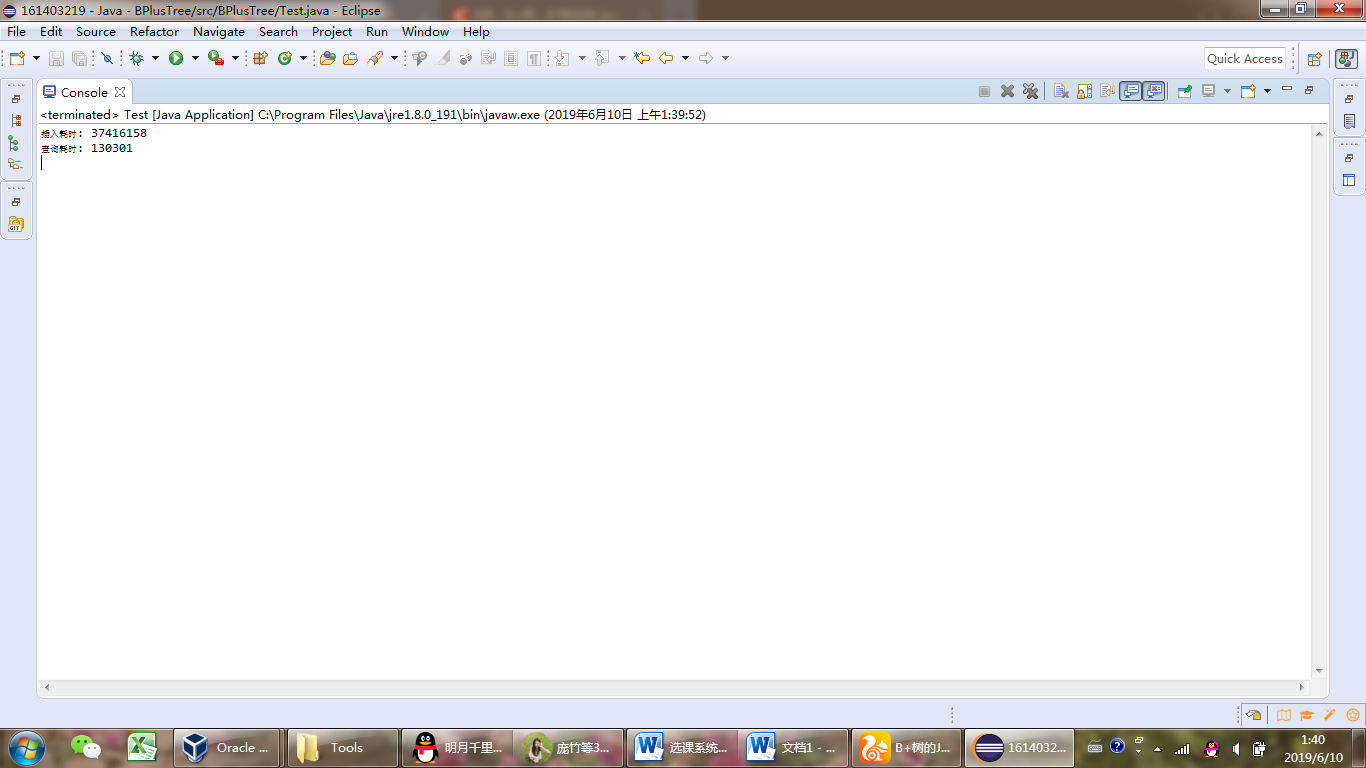


相关代码：





运行结果：



**4、红黑二叉树：**

（1）红黑二叉树的定义：

红黑树是一颗二叉查找树，且具有如下特性：

 1、每个节点或者是黑色，或者是红色。

 2、根节点是黑色。

 3、每个叶子节点是黑色。（注意：这里叶子节点，是指为空的叶子节点！）

 4、如果一个节点是红色的，则它的子节点必须是黑色的。

 5、从一个节点到该节点的子孙节点的所有路径上包含相同数目的黑色节点。



红黑二叉树

（2）红黑二叉树的基础操作

1. 左旋

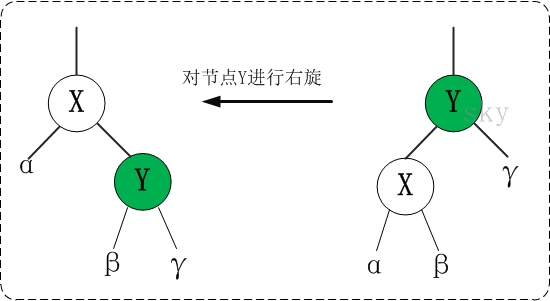


对x进行左旋，意味着"将x变成一个左节点"。

相关代码：



1. 右旋



对y进行右旋，意味着"将y变成一个右节点"。

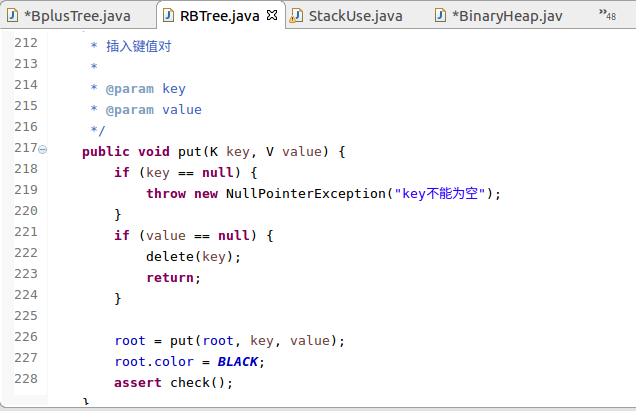
相关代码：

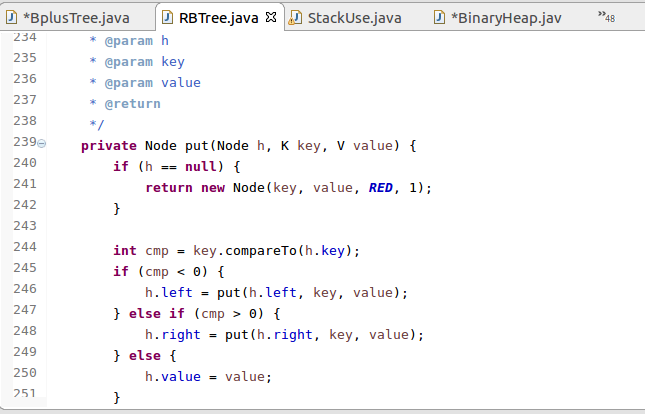


1. 添加节点操作：
2. 将红黑树当作二叉查找树，将节点插入。
3. 将插入的节点着色为"红色"。
4. 通过一系列的旋转或着色等操作，使之重新成为一颗红黑树。

对于"特性(4)"，是有可能违背的！那接下来，想办法使之"满足特性(4)"，就可以将树重新构造成红黑树了。

相关代码：





1. 删除节点操作
2. 将红黑树当作一颗二叉查找树，将节点删除。

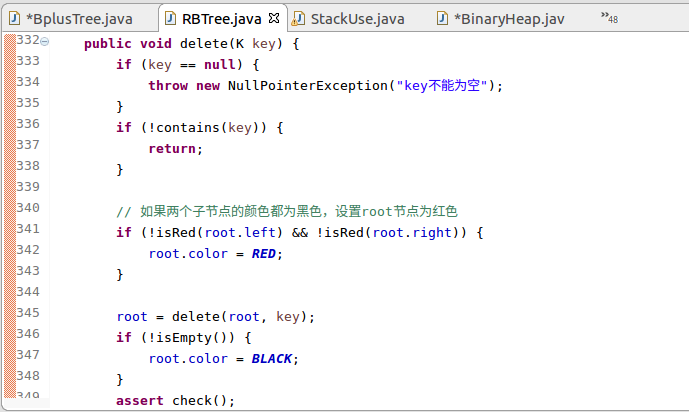
① 被删除节点没有儿子，即为叶节点。那么，直接将该节点删除就OK了。

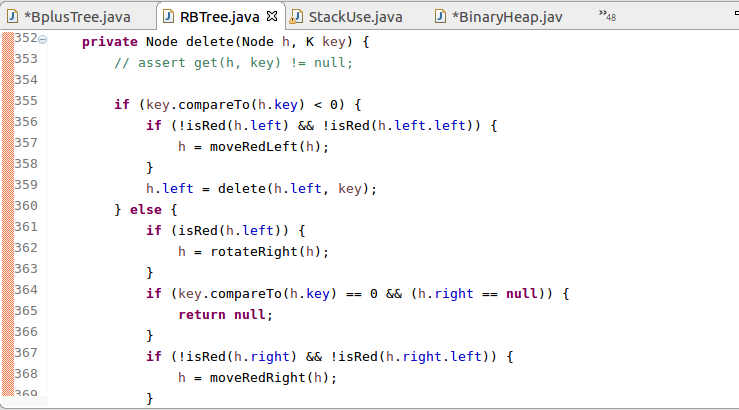
② 被删除节点只有一个儿子。那么，直接删除该节点，并用该节点的唯一子节点顶替它的位置。

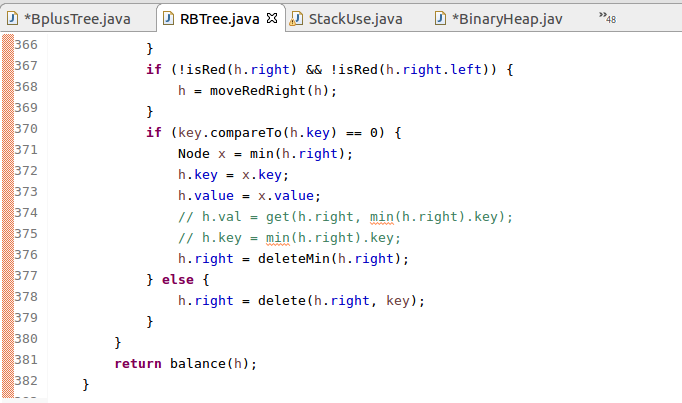
③ 被删除节点有两个儿子。那么，先找出它的后继节点；然后把“它的后继节点的内容”复制给“该节点的内容”；之后，删除“它的后继节点”。

1. 通过"旋转和重新着色"等一系列来修正该树，使之重新成为一棵红黑树。

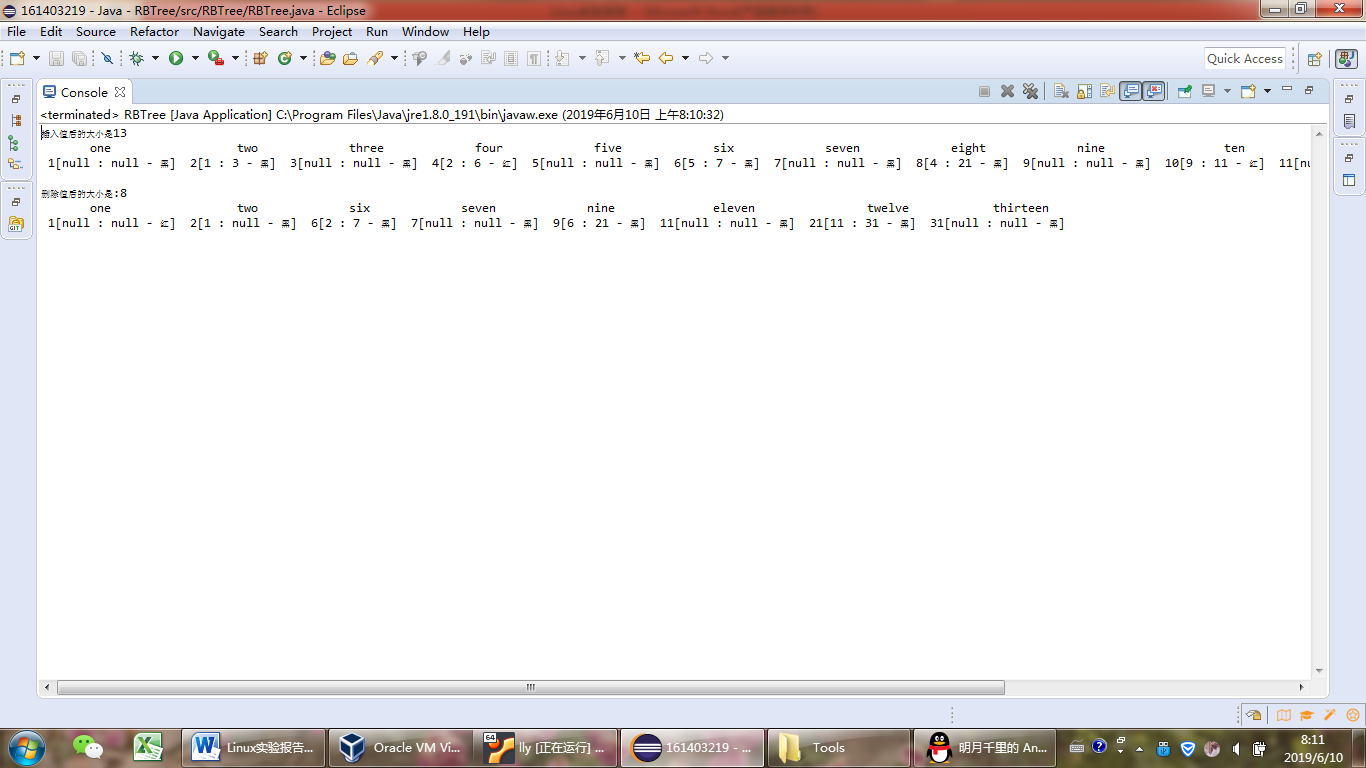
相关代码：







运行结果：



**三．实验小结：**

通过本次试验，我对堆，栈，B+树，红黑二叉树四种数据结构

有了更加深入透彻的理解，对于堆栈排序和相关应用掌握的较好，但是B+树和红黑二叉树需要更深入的研究学习，并且对Linux命令也有了一定了解，能够在Linux环境下运行出程序。