**二叉查找树**

**二叉查找树可以用链接结构实现。**

**一条路径的长度是指在该路径上的边的个数。**

**一个结点的深度是指从根结点到该结点的路径长度。**

**习惯上，将空树的高度定义为-1.**

**二叉查找树（没有重复元素）的特征是：对于树中的每一个结点，它的左子树中结点的值都小于该结点的值，而它的右子树中结点的值都大于该结点的值。**

**查找一个元素**

**让current指向根结点，重复下面的步骤直到current为null或者元素匹配current.element。**

**1.如果element小于current.element，就将current.left赋给current。**

**2.如果element大于current.element，就将current.right赋给current。**

**3.如果element等于current.element，就返回true。**

**如果current为null，那么子树为空且该元素不在这棵树中。**

**public boolean search(E element){**

**TreeNode<E> current =root;**

**while(current!=null)**

**if(element<current.element){**

**current=current.left;**

**}else if(element>currnet.element){**

**current=current.right;**

**}else**

**return true;**

**return false;**

**}**

**在BST中插入一个元素**

**如果这棵树是空的，就使用新元素创建一个根结点；否则，寻找新元素结点的父结点的位置。为该元素创建一个新结点，然后将该结点链接到它的父结点上。如果新元素的值小于父元素的值，则将新元素的结点设置为父结点的左子结点；如果新元素的值大于父元素的值，则将新元素的结点设置为父结点的右子结点。**

**树的遍历**

**分为中序、前序、后序、深度优先和广度优先。**

**中序遍历法：中序遍历法以递增顺序显示BST中的所有结点。**

**后序遍历法：可以找出一个文件系统中目录的个数。**

**前序遍历法：可以打印书的目录。**

**删除BST中的一个元素**

**为了从一棵二叉查找树中删除一个元素，首先需要定位该元位置，然后在删除该元素以及重新连接树前，考虑两种情况——该结点有或者没有左子结点。**

**为了从一棵二叉查找树中删除一个元素，首先需要定位包含该元素的结点，以及它的父结点。假设current指向二叉查找树中包含该元素的结点，而parent指向current结点的父结点。current结点可能是parent结点的左子结点，也可能是右子结点。这里需要考虑两种情况：**

**情况1：当前结点没有左子结点，这时只需要将该结点的父结点和该结点的右子结点相连。**

**（注意：如果当前结点是叶子结点，这就是属于情况1。）**

**情况2：current结点有左子结点。假设rightMost指向包含current结点的左子树中的最大元素的结点，而parentOfRightMost指向rightMost结点的父结点。注意，rightMost结点不能有右子结点，但是可能会有左子结点。使用rightMost结点中的元素值替换current结点中的元素值，将parentOfRightMost结点和rightMost结点的左子结点相连，然后删除rightMost结点。**

**（注意：如果current的左子结点没有右子结点，那么current.left指向current左子树的大元素。在这种情况下，rightMost是current.left，而parentOfRightMost是current。必须考虑这种特殊情况，重新连接rightMost的右子结点和parentOfRightMost。）**

**迭代器**

**迭代器是一个重要的软件设计模式。它提供遍历容器内元素的统一方法，同时隐藏该容器的构造细节。通过实现相同的接口java.util.Iterator，可以编写一个程序，以相同的方式遍历所有容器的元素。**

**java.util.Iterator定义一个前向迭代器，它以前向的方向遍历迭代器中的元素，每个元素只能遍历一次。java API还提供java.util.ListIterator，它支持前向遍历和后向遍历。如果你的数据结构要保证遍历的灵活性，可以将迭代器类定义为java.util.ListIterator的一个子类。**

**迭代器的实现不是很高效。每次通过迭代器删除一个元素时，整个线性表都要重新构造。客户程序应该总是采用BST类中的delete方法来删除一个元素。为了防止用户使用迭代器中的remove方法，如下实现迭代器：**

**public void remove(){**

**throw new UnsupportedOperationException("**

**Removing an element from the iterator is not supported");**

**}**

**在使得remove方法不被迭代器类支持后，无须为树中的元素维护一个线性表使得迭代器更加高效。可以使用栈来存储结点，栈顶端的结点包含从next（）方法返回的元素。如果树是平衡的，最大的栈的大小将为O(logn)。**

**数据压缩**

**霍夫曼编码通过使用更少的比特对经常出现的字符编码来压缩数据。字符的编码是基于字符在文本中出现的次数使用二叉树来构建的，该树称为霍夫曼编码树。**

**构建算法：**

**1.从由树构成的森林开始。每棵树都包含一个字符结点。每个结点的权重就是文本中字符的出现次数。**

**2.重复一下步骤来合并树，直到只有一棵树为止：选择两棵有最小权重的树，创建一个新结点作为它们的父结点。这棵树的权重是子树的权重和。**

**3.对于每个内部结点，给它的左边赋值0，而给它的右边赋值1.所有叶子结点都表示文本中的字符。**

**这里使用的是贪婪算法（greedy algorithm）的一个示例。贪婪算法经常用于解决优化问题。算法做出局部最优的选择，并希望这样的选择会导致全局最优。然而，贪婪算法不是总能找到最有结果。**

**小结：**

**1.二叉查找树是一种分层的数据结构。**

**2.迭代器是一个提供了遍历像集合、线性表或二叉树这样的容器中的元素的统一方法的对象。**

**3.霍夫曼编码是一种压缩数据的方案，它使用较少的比特来编码经常出现的字符。字符的编码是使用二叉树基于它在文本中出现的次数来构建的，该二叉树称为霍夫曼编码树。**