左偏树(Leftist Tree)是一种可并堆的实现。左偏树是一棵二叉树，它的节点

除了和二叉树的节点一样具有左右子树指针( left, right ) 外，还有两个属性：键值

和距离(dist)。键值上面已经说过，是用于比较节点的大小。距离则是如下定义

的：

节点i称为外节点(external node)，当且仅当节点i的左子树或右子树为空

( left(i) = NULL或right(i) = NULL )；节点i的距离( dist( i ) ) 是节点i到它的后代中，

最近的外节点所经过的边数。特别的，①如果节点i本身是外节点，则它的距离

为 0；而②空节点的距离规定为-1 (dist(NULL) = -1)。在本文中，有时也提到③

一棵左偏树的距离，这指的是该树根节点的距离。

左偏树满足下面两条基本性质：

[性质 1] 节点的键值小于或等于它的左右子节点的键值。

即 key(i) ≤key(parent(i)) 这条性质又叫堆性质。符合该性质的树是堆有序

的(Heap-Ordered)。有了性质 1，我们可以知道左偏树的根节点是整棵树的最小

节点，于是我们可以在 O(1) 的时间内完成取最小节点操作。

[性质 2] 节点的左子节点的距离不小于右子节点的距离。

即 dist(left(i)) ≥ dist(right(i)) 这条性质称为左偏性质。性质 2 是为了使我们

可以以更小的代价在优先队列的其它两个基本操作（插入节点、删除最小节点）

进行后维持堆性质。在后面我们就会看到它的作用。

这两条性质是对每一个节点而言的，因此可以简单地从中得出，左偏树的左

右子树都是左偏树。

由这两条性质，我们可以得出左偏树的定义：左偏树是具有左偏性质的堆有

序二叉树。

十三、左偏树（Leftist Tree）

树这个数据结构内容真的很多，上一节所讲的二叉堆，其实就是一颗二叉树，这次讲的左偏树（又叫“左翼堆”），也是树。

二叉堆是个很不错的数据结构，因为它非常便于理解，而且仅仅用了一个数组，不会造成额外空间的浪费，但它有个缺点，那就是很难合并两个二叉堆，对于“合并”，“拆分”这种操作，我觉得最方面的还是依靠指针，改变一下指针的值就可以实现，要是涉及到元素的移动，那就复杂一些了。

左偏树跟二叉堆比起来，就是一棵真正意义上的树了，具有左右指针，所以空间开销上稍微大一点，但却带来了便于合并的便利。BTW：写了很多很多的程序之后，我发觉“空间换时间”始终是个应该考虑的编程方法。:)

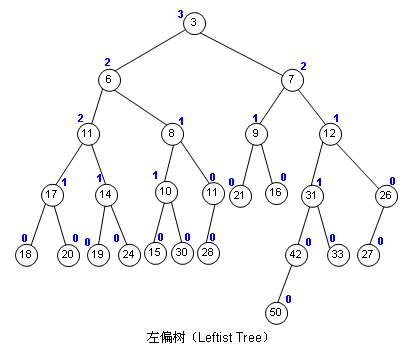
左偏左偏，给人感觉就是左子树的比重比较大了，事实上也差不多，可以这么理解：左边分量重，那一直往右，就一定能最快地找到可以插入元素的节点了。所以可以这样下个定义：左偏树就是对其任意子树而言，往右到插入点的距离（下面简称为“距离”）始终小于等于往左到插入点的距离，当然了，和二叉堆一样，父节点的值要小于左右子节点的值。

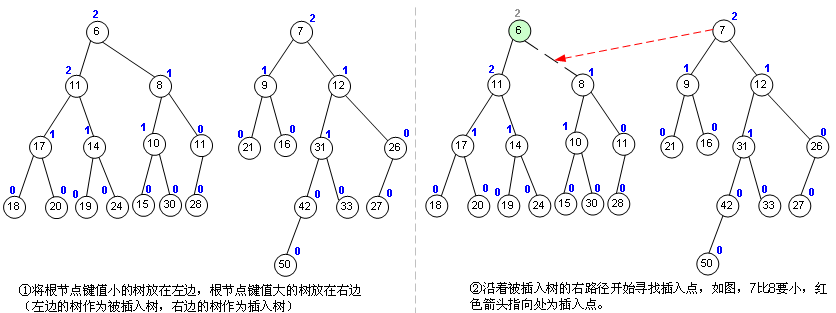
如果节点本身不满，可插入，那距离就为0，再把空节点的距离记为-1，这样我们就得出：父节点的距离 = 右子节点距离 + 1，因为右子节点的距离始终是小于等于左子节点距离的。我把距离的值用蓝色字体标在上图中了。

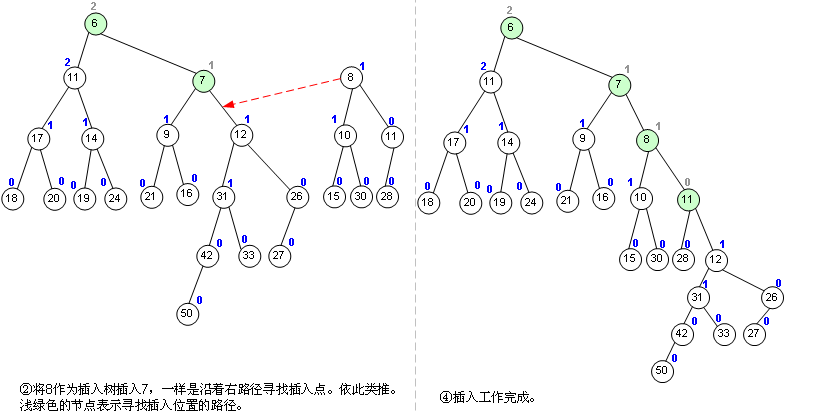
左偏树并一定平衡，甚至它可以很不平衡，因为它其实也不需要平衡，它只需要像二叉堆那样的功能，再加上合并方便，现在来看左偏树的合并算法，如图：

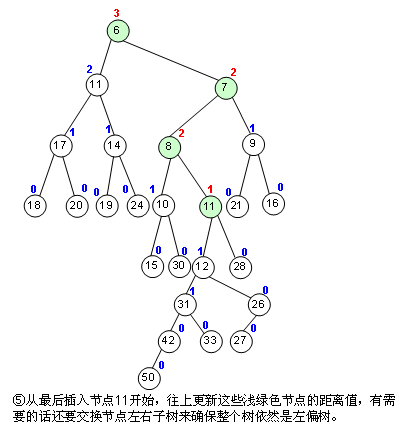
这种算法其实很适合用递归来做，但我还是用了一个循环，其实也差不多。对于左偏树来说，这个合并操作是最重要最基本的了。为什么？你看哦：Enqueue，我能不能看作是这个左偏树的root和一个单节点树的合并？而Dequeue，我能不能看作是把root节点取出来，然后合并root的左右子树？事实上就是这样的，我提供的代码就是这样干的。

十三、左偏树（Leftist Tree）  
树这个数据结构内容真的很多，上一节所讲的二叉堆，其实就是一颗二叉树，这次讲的左偏树（又叫“左翼堆”），也是树。  
二叉堆是个很不错的数据结构，因为它非常便于理解，而且仅仅用了一个数组，不会造成额外空间的浪费，但它有个缺点，那就是很难合并两个二叉堆，对于“合并”，“拆分”这种操作，我觉得最方面的还是依靠指针，改变一下指针的值就可以实现，要是涉及到元素的移动，那就复杂一些了。  
左偏树跟二叉堆比起来，就是一棵真正意义上的树了，具有左右指针，所以空间开销上稍微大一点，但却带来了便于合并的便利。BTW：写了很多很多的程序之后，我发觉“空间换时间”始终是个应该考虑的编程方法。:)  
左偏左偏，给人感觉就是左子树的比重比较大了，事实上也差不多，可以这么理解：左边分量重，那一直往右，就一定能最快地找到可以插入元素的节点了。所以可以这样下个定义：左偏树就是对其任意子树而言，往右到插入点的距离（下面简称为“距离”）始终小于等于往左到插入点的距离，当然了，和二叉堆一样，父节点的值要小于左右子节点的值。

  
如果节点本身不满，可插入，那距离就为0，再把空节点的距离记为-1，这样我们就得出：父节点的距离 = 右子节点距离 + 1，因为右子节点的距离始终是小于等于左子节点距离的。我把距离的值用蓝色字体标在上图中了。  
左偏树并一定平衡，甚至它可以很不平衡，因为它其实也不需要平衡，它只需要像二叉堆那样的功能，再加上合并方便，现在来看左偏树的合并算法，如图：







这种算法其实很适合用递归来做，但我还是用了一个循环，其实也差不多。对于左偏树来说，这个合并操作是最重要最基本的了。为什么？你看哦：Enqueue，我能不能看作是这个左偏树的root和一个单节点树的合并？而Dequeue，我能不能看作是把root节点取出来，然后合并root的左右子树？事实上就是这样的，我提供的代码就是这样干的。