1 balance() 函数

1.1 功能

考察以 t 为 root 的子树，如果它不平衡，则判断它是不平衡中的哪种情况，通过旋转操作让它平衡。

注意！不管它是否平衡，都更新 t 的 height，所以 balance() 是包含更新 t 的 height 这一操作的，下不赘述。

1.2 参数

指向一个子树 root 的指针，并且为引用，因此函数内部修改 t 时，直接修改实参。

1.3 返回值

无。

1.4 功能实现

考察它的左右孩子的 height，若差值大于设定值 ALLOWED_IMBALANCE，再考察 height 大的那个孩子的左右孩子的 height 哪个大，据此分为四种情况，每种情况有对应的旋转方法，相对应做旋转就能使 t 为 root 的子树以及它的所有子树都平衡。

2 我的 remove() 是如何实现的

在 remove() 和 detachMin() 函数的最后一行加一行 balance()，因为它们都存在删除操作，动态操作做完需要 balance。

把 detachMin() 改为递归形式（上次作业是循环形式），这样才能形成调用堆栈，也就是从 remove() 中调用 detachMin(t->right) 开始，向下向左搜索最小值，找到后每向上回溯一次就检查 balance 一次，因为对下面节点进行 balance 操作也是动态操作，会影响上面节点的 height。

remove() 也是同理的，最下面 remove 完之后向上回溯检查 balance。

具体来说，这两个函数的内容都是：

```
{
...
递归（即向下搜索）
...
balance();
}
```

也就是 balance() 在递归之后，所以是做完所有递归（搜索）再开始做 balance()，也就是回溯的时候做 balance()，要是 balance 在递归前，那就是反过来，搜索的时候做 balance 了，那就不对了。