Part 1

通过阅读对fhq treap的简单介绍，你将初步认识这种有趣的数据结构。你需要理解该数据结构的一些基础操作，并根据文字解释和源代码，补全这些操作的jml规格。

背景概述

首先我们来复习一下二叉搜索树，一棵二叉搜索树需要满足：

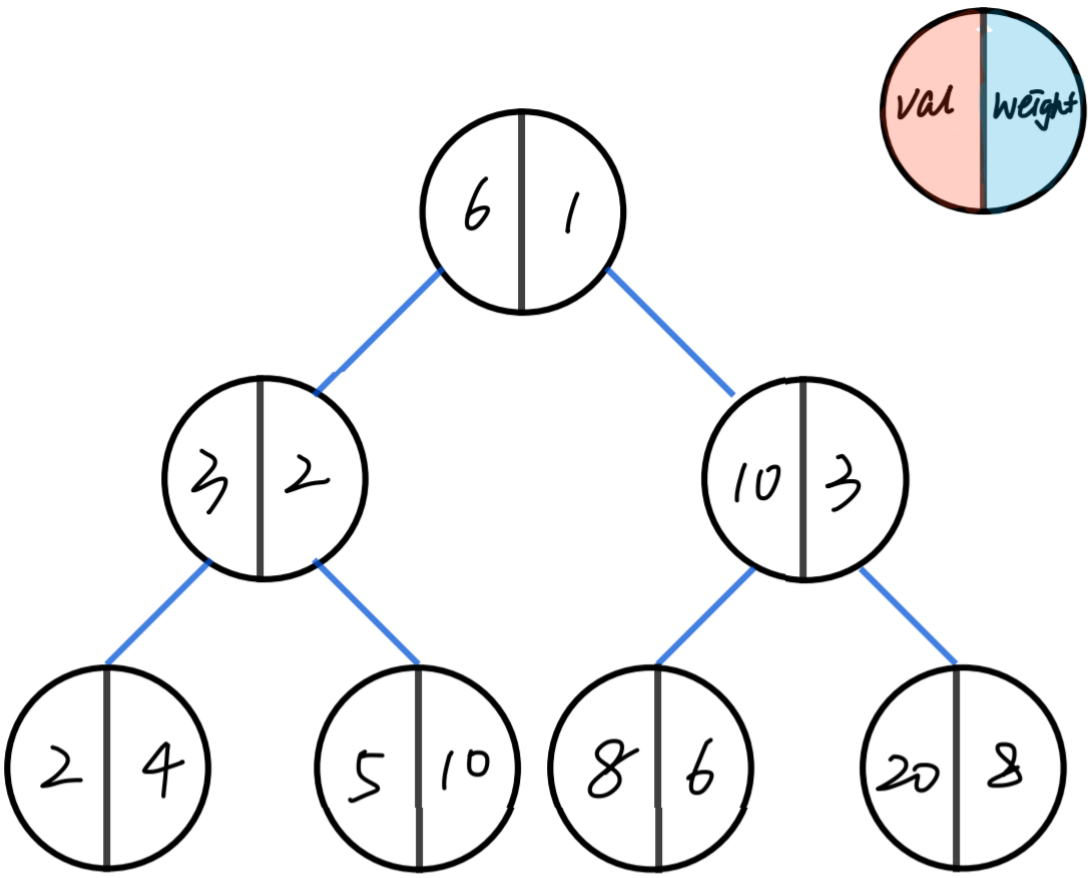
- 非叶子节点最多拥有两个子节点；
- 非叶子节值大于左边子节点、小于右边子节点

但是普通二叉搜索树的形状完全取决于插入顺序，在极端情况下甚至会退化成链表，使各种操作的时间开销大大增加。因此我们对二叉搜索树进行优化，就有了各种各样的平衡树，使得其深度总是维持在 $O(\log N)$ 左右。之前的数据结构课上，同学们接触过的AVL就是一种平衡树，它严格保证两子树的高度差小于等于1。而今天要介绍的fhq treap是一类不那么严格的平衡树。

treap定义

treap，即为tree + heap，具体来说，它的每个节点有两个属性——val和weight。一棵合法的 Treap 仍然是一棵二叉树，它既保证了 val 在树中具有二叉搜索树的性质，又保证了 weight 在树中具有堆的性质，这里约定堆为小顶堆，即子节点的weight值总是大于等于父节点的weight值。在本次训练中，我们保证treap中每个节点的val值不会相同。

举例说明，下面是一棵合法的 treap：



任务内容

定义FhqTreap类为fhq treap的节点，下面我们介绍fhq treap的一些基本操作：

- public FhqTreap split_l(int key)：
  - 功能：返回treap中所有val值比key小的节点组成的树（简称“小树”）
- public FhqTreap split_r(int key)：
  - 功能：返回treap中所有val值比key大的节点组成的树（简称“大树”）
- public static FhqTreap merge(FhqTreap treapA, FhqTreap treapB)：
  - 功能：将两棵treap合并，返回合并后得到的树
  - 前置条件：treapA中val的最大值必须小于treapB中val的最小值（此前置条件无需在规格中体现）
- public FhqTreap insert(FhqTreap node)：
  - 功能：向treap中插入一个新节点，返回插入后得到的树

你需要根据指导书中的文字描述和我们提供的源码，补全上述操作所对应的方法的jml规格。

Part 1官方代码链接：https://gitlab.buaaoo.top/oo_2021_public/training/UNIT-3/TRAINING-3/tree/master/part-1

操作的具体实现

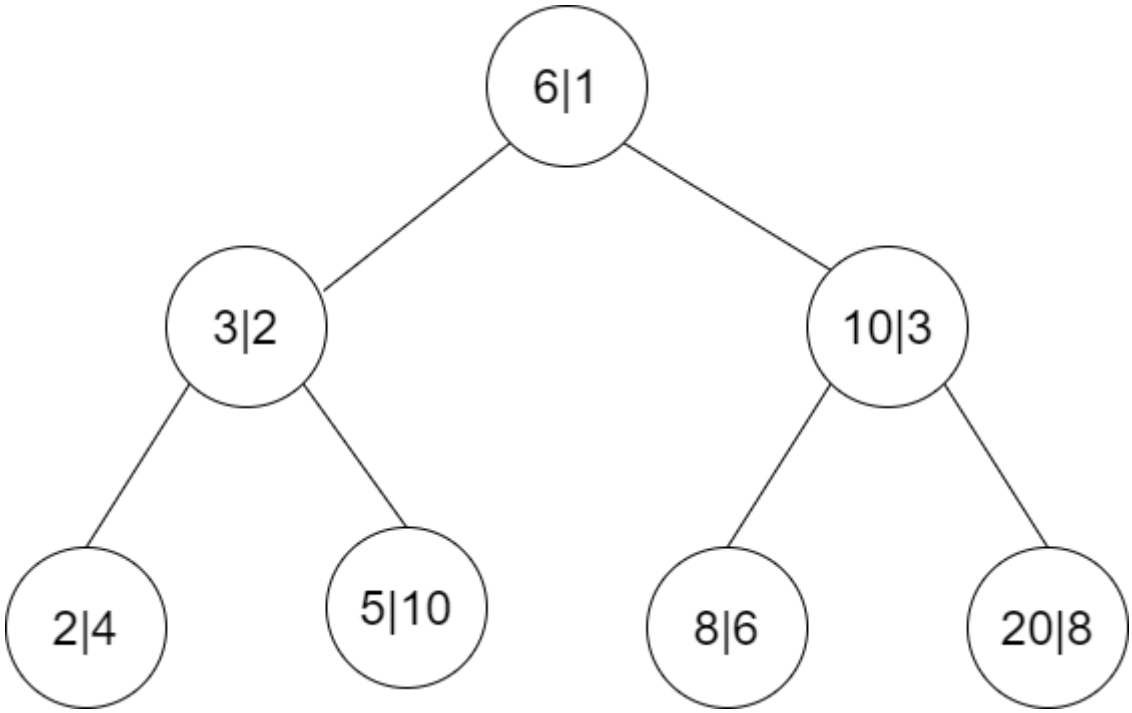
注意：以下方法大多采用递归实现。理解递归过程的描述，需要你仔细区分描述中的***treap 节点***和***以该节点为根的树***

split_l:

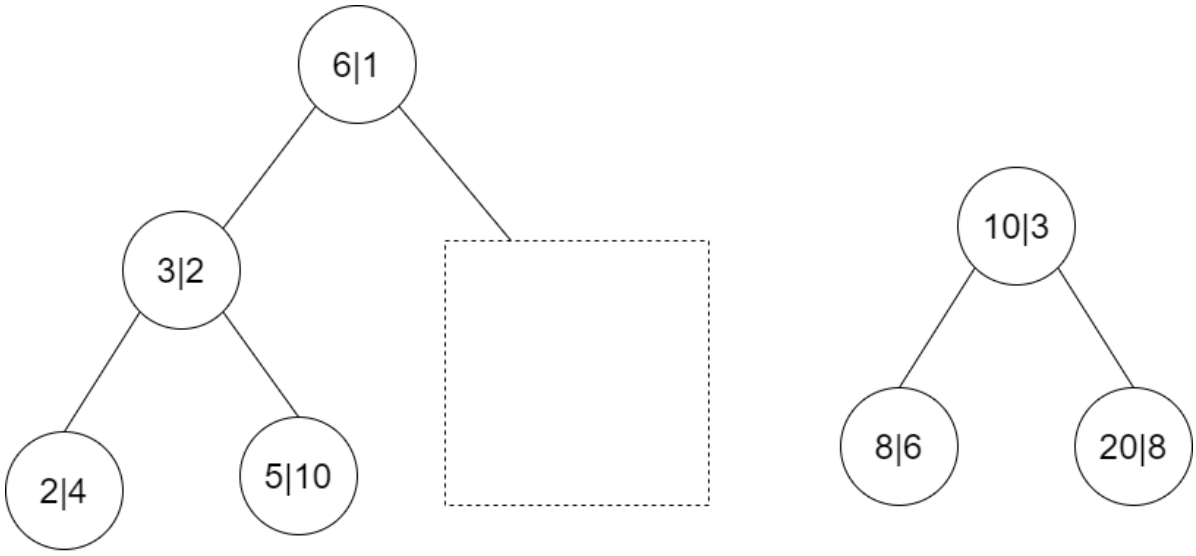
比较该节点的val值和key的大小：

- 如果val>key，则该节点不在结果中，自然地，该节点的右子树都满足val>key，因此只需要对其左孩子递归调用split_l，在其左子树中继续搜索符合条件的节点；
- 如果val<key，则将该节点加入结果，自然地，该节点的左子树也需要加入结果，然后只需要在该节点右子树中继续搜索符合条件的节点。

  下面我们以key = 9为例，对下图的treap进行分裂，获得root.split_l(9)的结果。

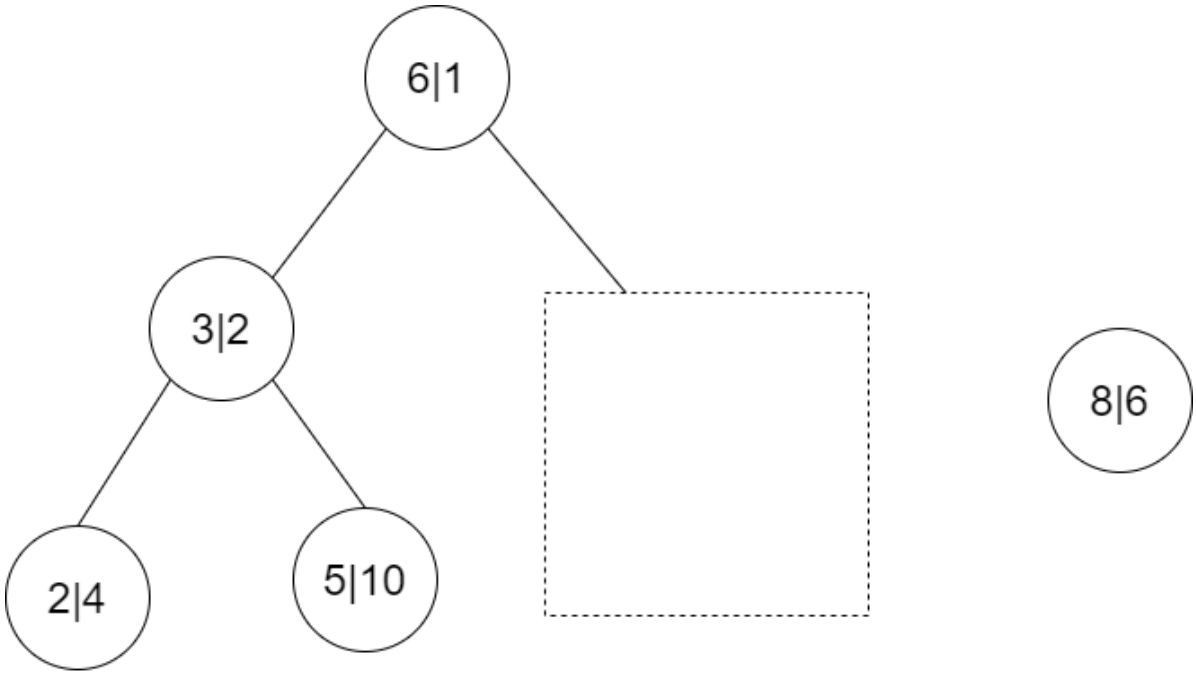


- 根节点的val值为6，小于key值，因此将根节点和其左子树移入结果中，其右子树成为待考察部分。



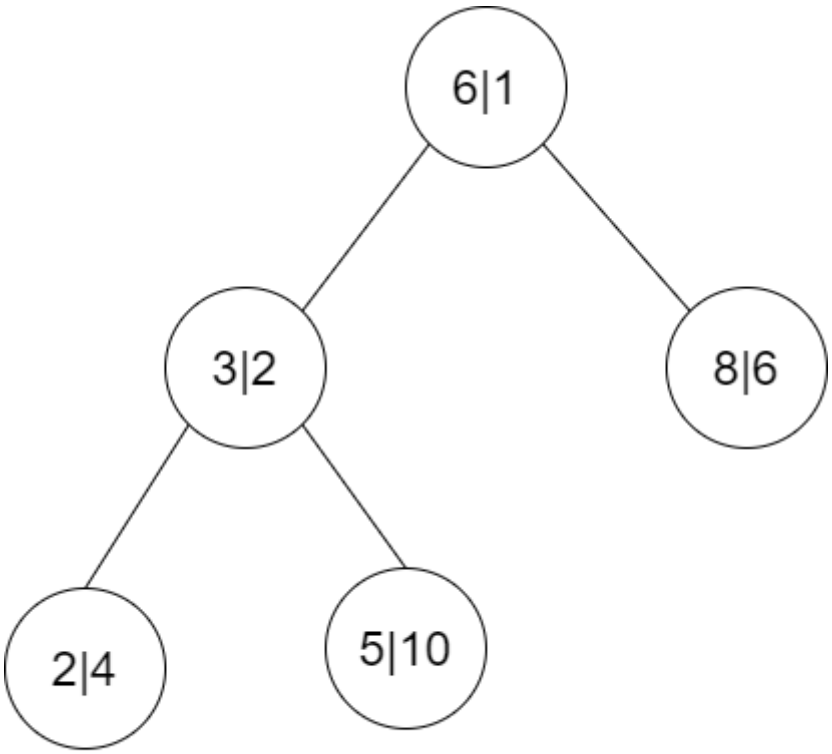
小树　　待考察部分

- 待考察部分根节点val值为10，大于key值，因此将根节点和其右子树舍去，留下其左子树作为待考察部分。



小树　　待考察部分

- 待考察部分根节点val值为8，小于key值，因此将根节点及其左子树(null)移入结果中，本应该将其右子树留在待考察部分，但因为其右子树为null，因此结束递归，得到结果。



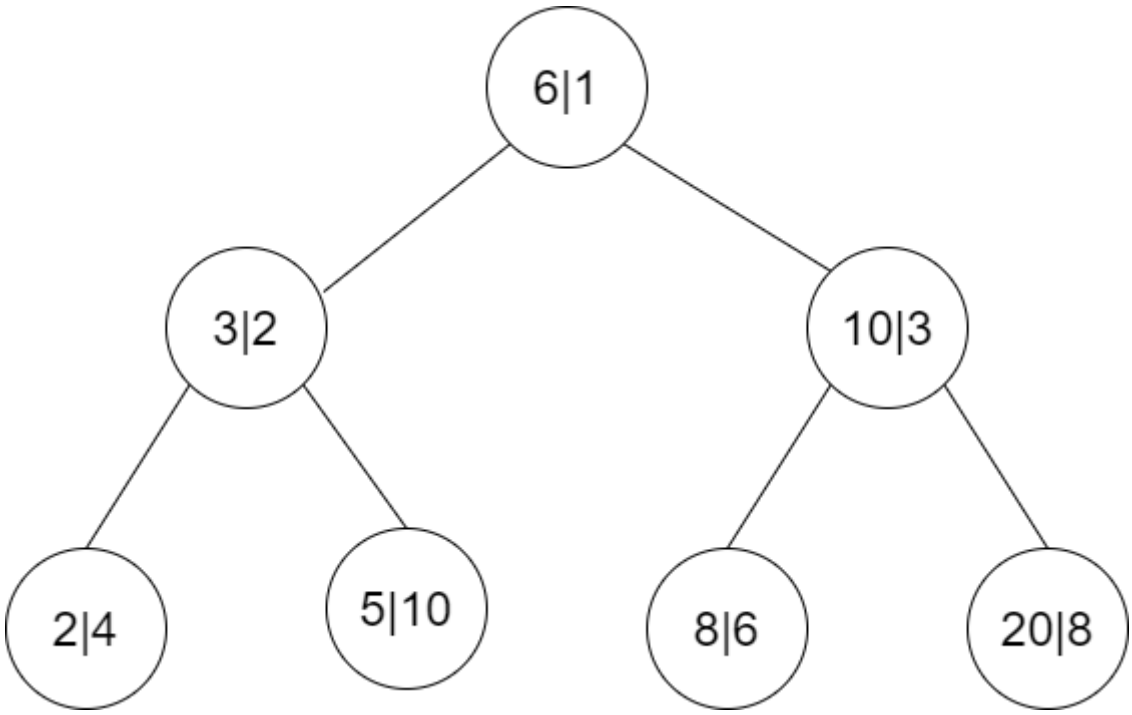
小树

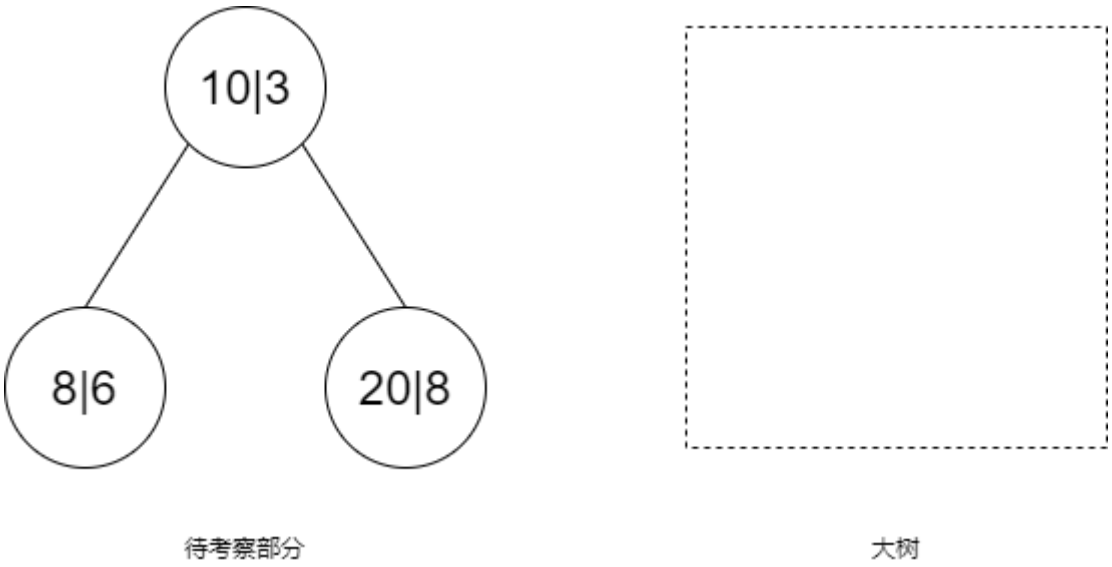
split_r:

与split_l类似，比较该节点的val值和key的大小：

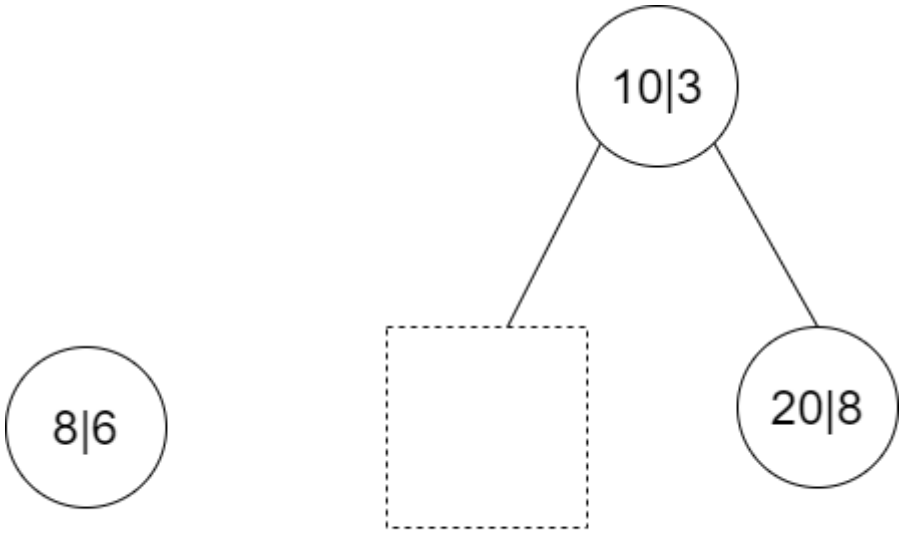
- 如果val<key，则该节点不在结果中，自然地，该节点的左子树都满足val<key，因此只需要对其右孩子递归调用split_l，在其右子树中继续搜索符合条件的节点；
- 如果val>key，则将该节点加入结果，自然地，该节点的右子树也需要加入结果，然后只需要在该节点左子树中继续搜索符合条件的节点。

  同样以key = 9为例，对下图的treap进行分裂，获得root.split_r(9)的结果：

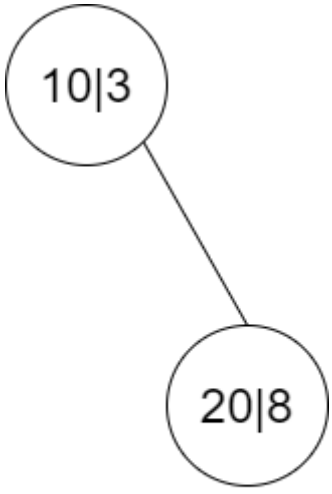




待考察部分　　大树

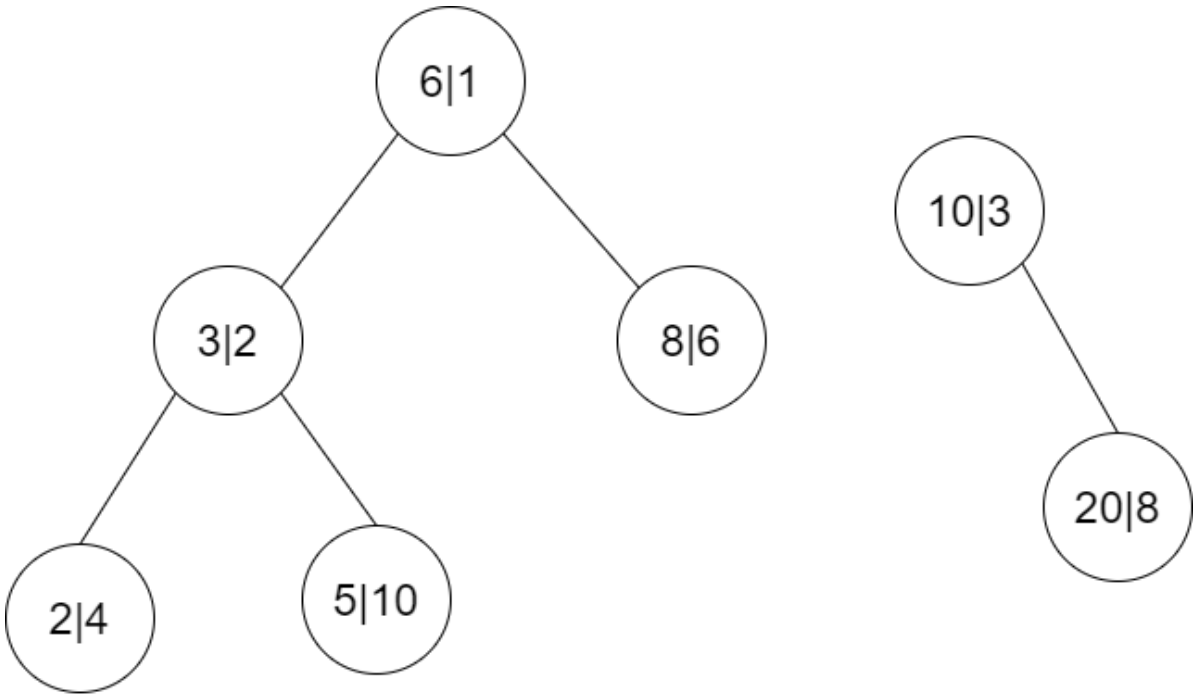


待考察部分　　大树



大树

最后我们验证一下最终的分裂结果，发现通过split_l和split_r，确实能够根据key值将原treap分裂为两个treap。



小树　　大树

merge:

将两棵treap（treapA中val的最大值小于treapB中val的最小值）按堆的性质合并为一棵treap：

因为treapA节点的val值小于treapB节点的val值，为了保持二叉搜索树的性质，因此这两个节点的合并结果只有两种可能：

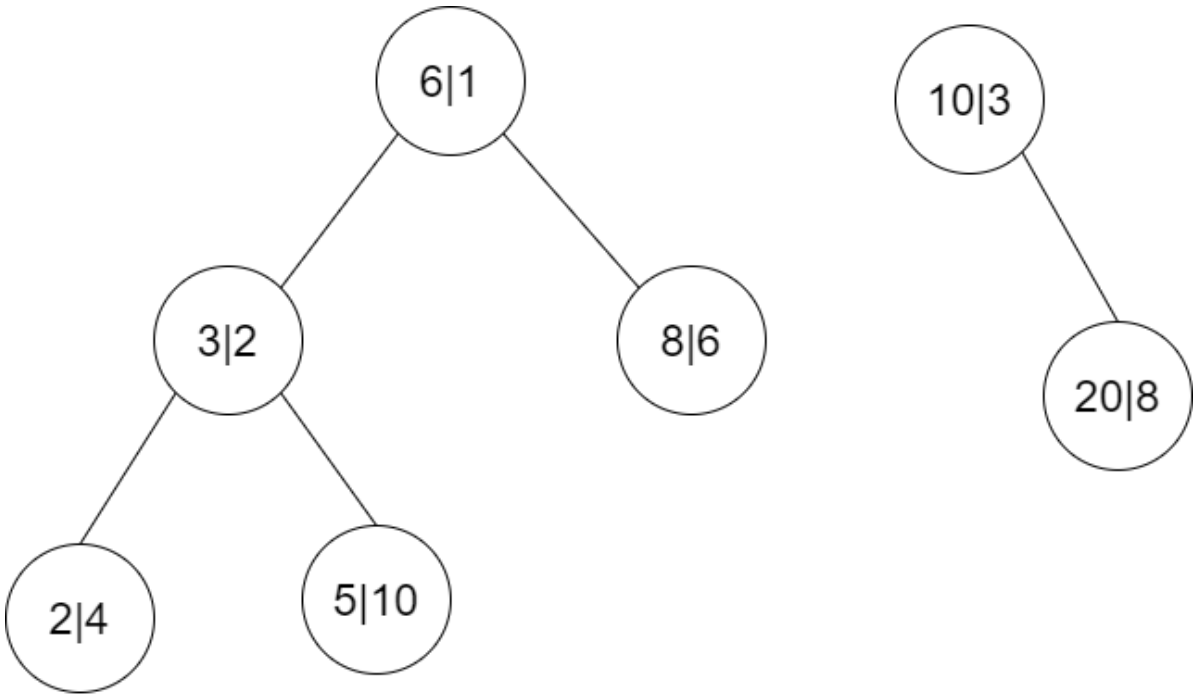
- treapA节点在treapB节点的左子树中
- treapB节点在treapA节点的右子树中

此时需要比较treapA.weight和treapB.weight。

若treapA节点的weight值较小，则按照堆的性质，treapA节点应该是treapB节点的祖先，即treapB节点在treapA节点的右子树中。又因为treapA节点的val值小于treapB中的val最小值，为了保持二叉搜索树的性质，treapB的所有节点都在treapA节点的右子树中，需要对treapB与treapA节点的右子树进行merge。

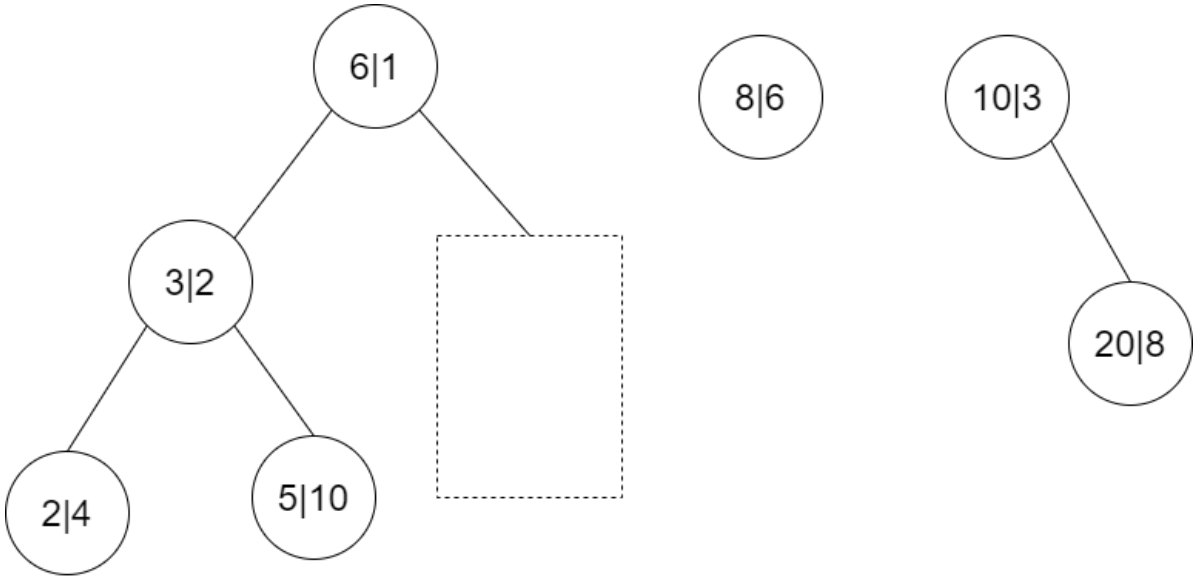
下面我们演示一下如何将上图分裂得到的两棵treap合并回去：

- 因为是通过分裂得到的treapA，treapB，必然满足合并的前置条件



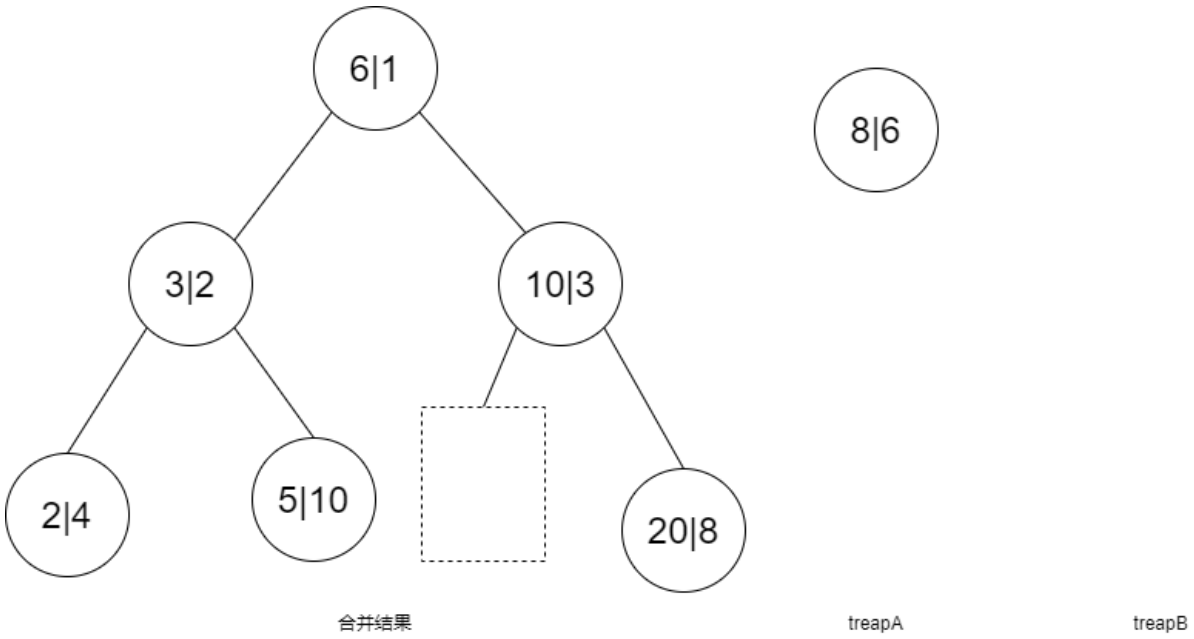
treapA　　treapB

- A的根节点的weight值为1，小于B的对应值3，因此将A节点和其左子树移入结果中，将B与A节点的右子树继续合并。

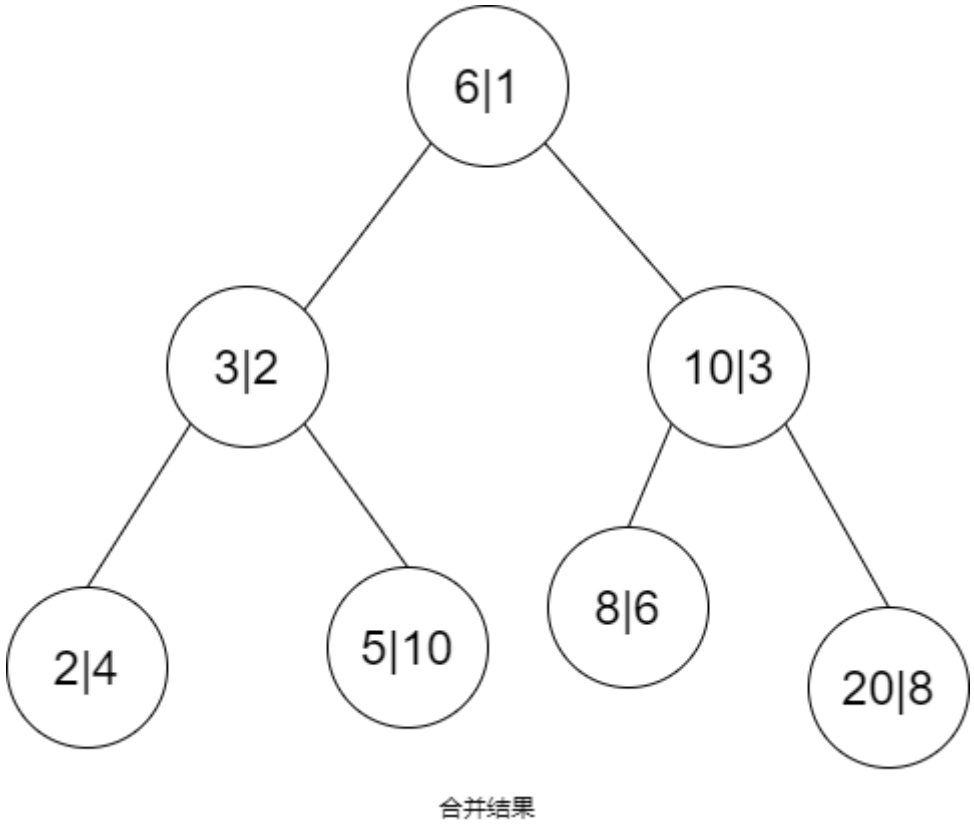


合并结果　　treapA　　treapB

- B的根节点的weight值为3，小于A的对应值6，因此将B节点和其右子树移入结果中，将A与B节点的左子树继续合并。



- B树为空，结束递归。



insert：

有了上面的铺垫，insert操作就很简单了，总的来说就是先分裂，后合并：

首先根据新节点node的val值，对treap进行分裂，得到两棵treap。显然任意一棵treap和node节点满足merge的条件，合并后的结果与另一棵treap也满足merge的条件。因此再经过两次merge，就能够完成将node插入treap。

Part 2

`IntSet` 对象可以用来存储和管理规模未知的一组无重复整数，其中：

- 方法 `elementSwap()` 完成两个 `IntSet` 对象所管理的数据集合的交换。
- 方法 `symmetricDifference()` 完成求两个 `IntSet` 对象所管理的数据集合的对称差。

任务内容

任务一：

补全 `elementSwap()` 及 `symmetricDifference()` 两个方法的规格，无需填写代码。

任务二：

根据不变式在MySet.java文件中补全 `repOk()` 这一方法。

任务三：

对已实现的函数 `insert()` 及 `delete()` 开展 JUnit 单元测试，说明所找出 bug 的现象，同时直接在 MySet.java 文件中修改原码以消除 bug 。

要求：在测试前输出 "Test Start!"，在测试后输出 "Test End!"。

所需提交的commit中应包含如下文件：

- IntSet.java：补全任务一中所要求规格的原 IntSet.java 文件。
- MySet.java：补全任务二中所要求方法，并修改完已实现函数中 bug 的原 MySet.java 文件。
- MySetTest.java：对 MySet 类开展 JUnit 单元测试的文件。
- README.md：对通过 JUnit 单元测试所找到的 bug 进行说明的文件。

Part 2官方代码链接：https://gitlab.buaaoo.top/oo_2021_public/training/UNIT-3/TRAINING-3/tree/master/part-2