高级数据结构Lab1：无旋Treap

无旋Treap简介

Treap与无旋Treap

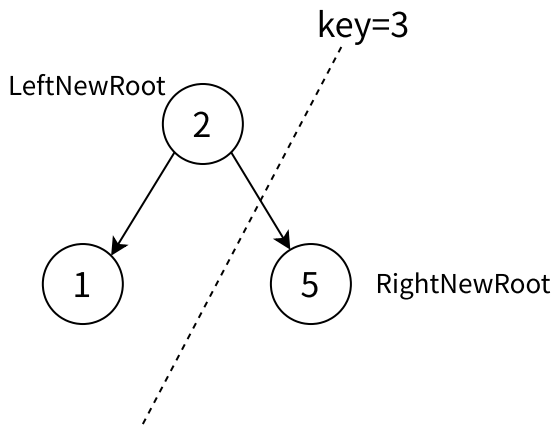
在了解无旋Treap之前首先介绍普通Treap。Treap是一种弱平衡的搜索树，它是由tree和heap两种数据结构组合而成的。Treap的每个节点上要额外储存一个值weight，代表每个节点的权重。因此对于每个节点，不仅要满足二叉搜索树的基本性质，还需额外满足父节点的weight大于两个子节点的weight（这里以最大堆为例）。Treap可以用来实现很多高效的操作，比如插入、删除、查询、求第k大元素等。

- 插入操作：首先按照二叉搜索树的方法找到合适的位置，然后建立新的节点，存储元素和随机权重。接着从新节点开始向上旋转，直到满足父节点的权重大于子节点的权重。
- 删除操作：首先按照二叉搜索树的方法找到要删除的节点。如果该节点是叶子节点，直接删除即可。如果该节点有两个子节点，选择权重较小的子节点进行旋转，然后重复这个过程，直到将要删除的节点转移到叶子位置，再删除即可。

无旋Treap又称FHQ Treap，是一种不进行旋转的Treap。它通过分裂和合并两种操作来实现插入、删除、查询等功能。下面介绍无旋Treap实现插入删除的主要方式。

分裂操作

分裂操作：按照一个key将一棵树分成两颗子树，左子树中的值都小于等于key，右子树的值都大于key。

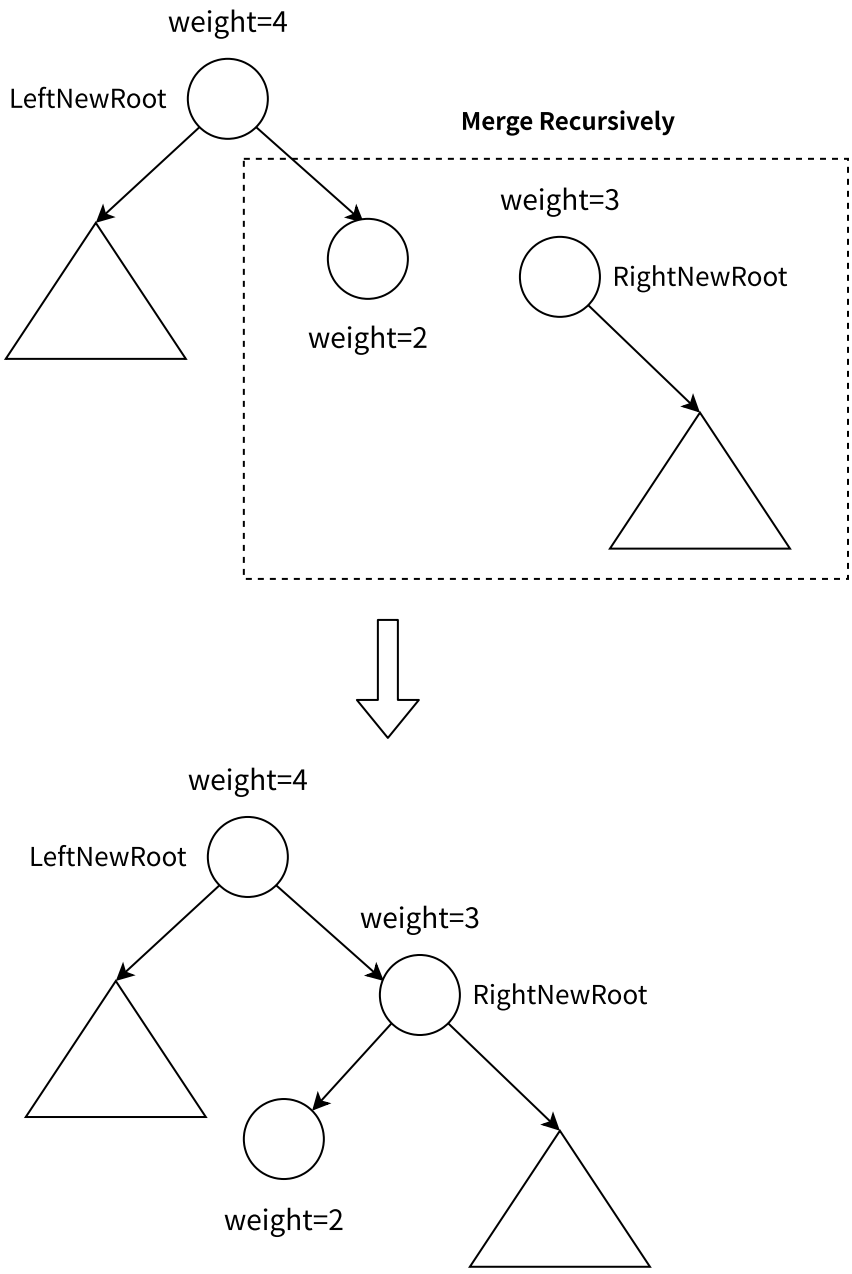


合并操作

合并时需要保持最大堆的性质。假设左子树根节点的权重大于等于右子树的权重，则将右子树递归合并到左子树的右儿子中，反之亦然。

以下图为例：

1. 初始要把weight=4和weight=3的两个节点合并，因为要保证最大堆的性质，所以将weight=3的子树和weight=4的右子树递归合并
2. 接下来合并weight=2和weight=3的两个点。因为2<3，递归合并weight=2的点和weight=3的点的左子树。
3. 因为weight=3的左子树是空的，直接将weight=2的点接到weight=3节点的左边。
4. 合并出来的新子树（以weight=3为根）接到weight=4的右子树。



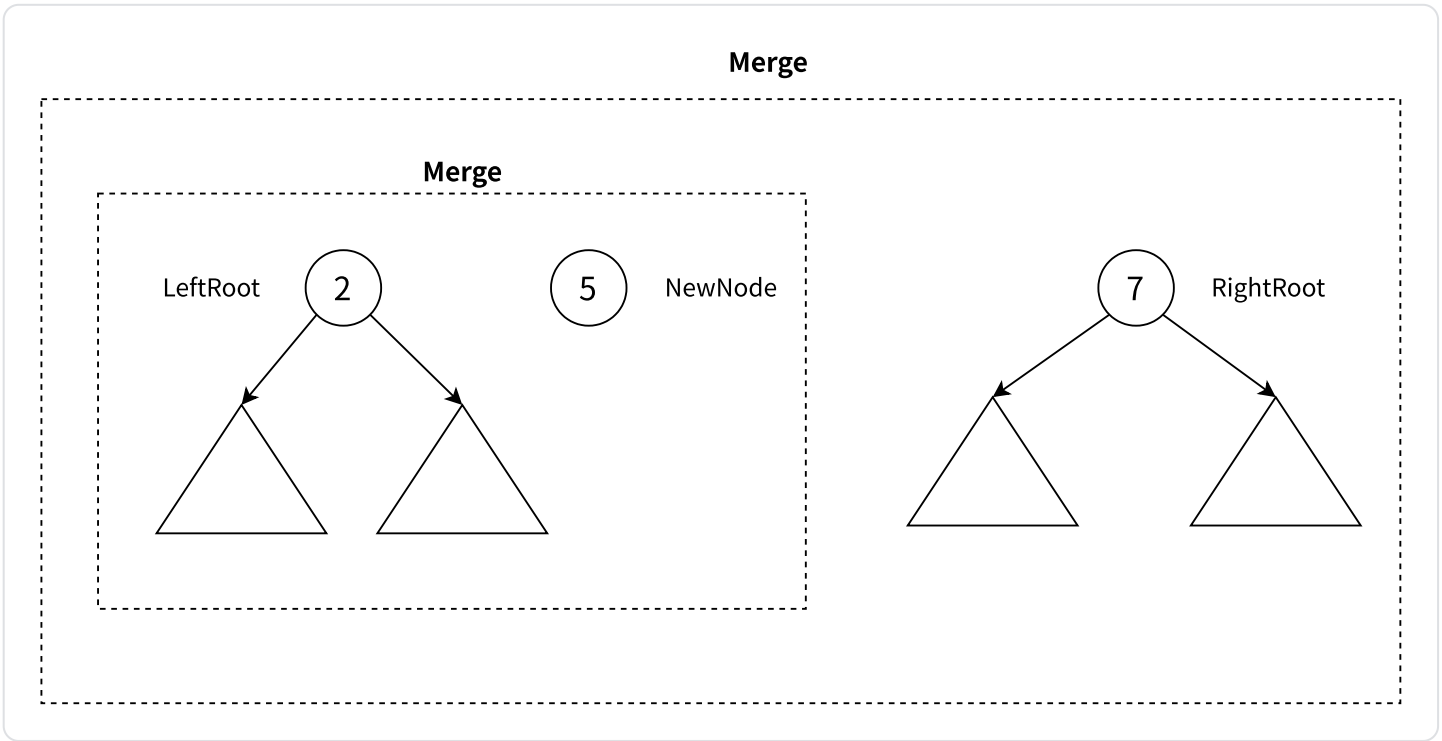
插入操作

假设即将插入的值为val。插入操作首先将整棵树划分成小于等于val的左子树(图中LeftRoot对应子树）和大于val的右子树（图中RightRoot对应子树）。如果左子树中已经存在val则将其count++，否则创建一个新节点，将其和左子树合并一颗新的左子树，然后将新的左子树与右子树合并。

以下图为例要向树中插入val=5的新节点。

1. 按照5将整棵树分成左右两部分。其中左子树的值全部小于等于5，右子树的值全部严格大于5。
2. 在左子树中查找是否已经存在val=5的节点，假设存在则增加其counter，结束插入操作。
3. 假设不存在，则新创建一个val=5的节点。
4. 将val=5的节点和左子树（图中LeftRoot) 合并成一颗新的左子树。
5. 将新的左子树和最初分裂出来的右子树(图中RightRoot)进行合并。

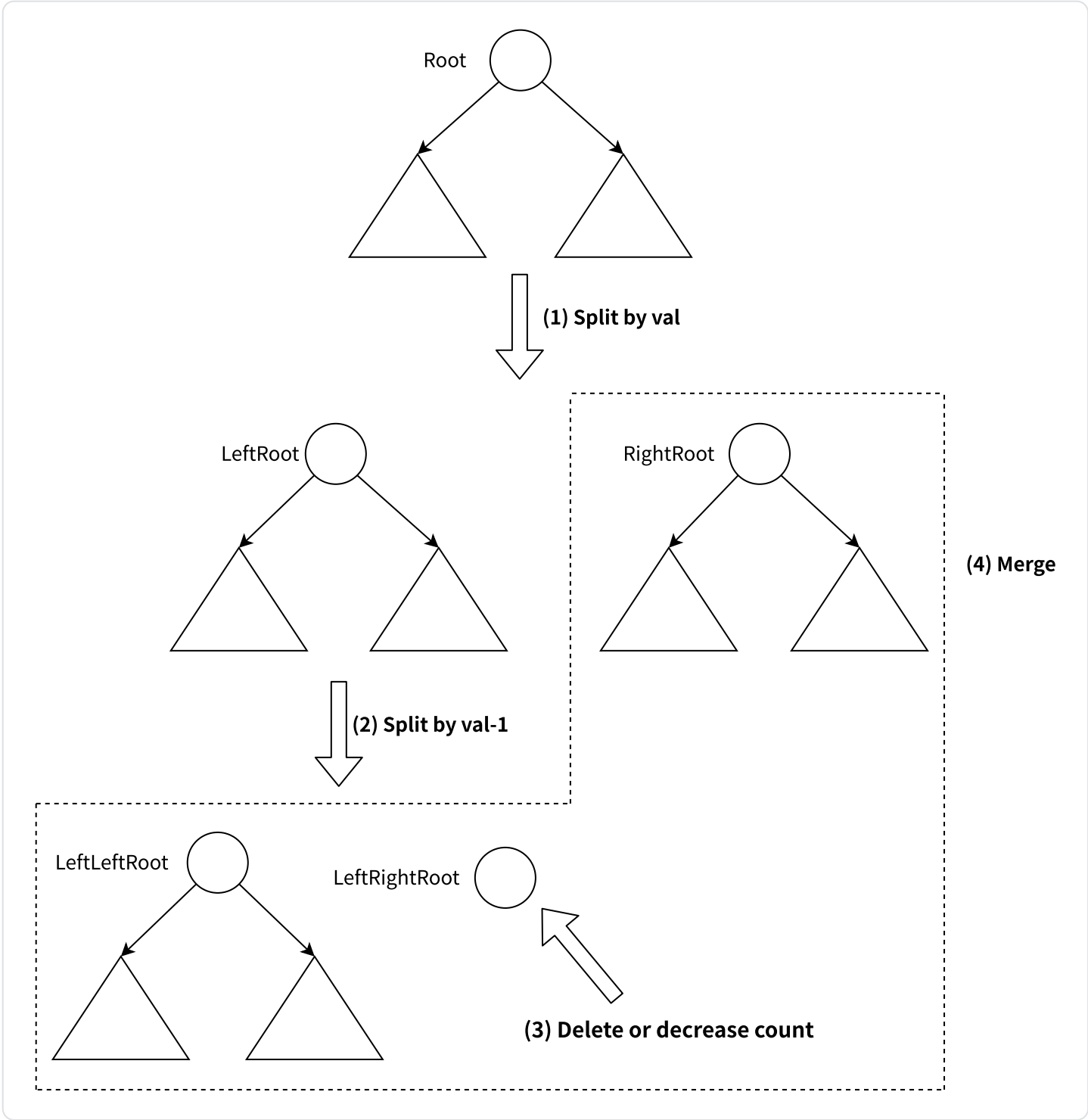
> 注意在合并的时候维护weight最大堆的性质（具体参考合并操作）。



删除操作

删除操作主要分为以下4步:

1. 将整颗树划分为小于等于val的左子树(图中LeftRoot）和大于val的的右子树（图中RightRoot）。
2. 将左子树划分为小于val的sub左子树(图中LeftLeftRoot)和等于val的sub右子树(图中LeftRightRoot)。
3. 如果sub右子树存在则count--，如果减少到0则删除sub右子树。
4. 将sub左子树、sub右子树(如果没有被删除的话)以及右子树进行合并。



任务

功能实现

在Lab-2中，你应该实现一个无旋Treap，它具有以下功能：（1）插入（2）删除（3）查询排名（4）查询数字（5）查询前驱（6）查询后继（7）清空树（8）输出前序遍历结果。

输入和输出的约定如下：

输入的第一行n表示操作的个数，接下来n行，每一行的第一个数字（op）表示操作的类型，第二个数字(num)表示操作对应的参数。

每个操作数对应的含义如下：

1. 插入单个数，插入完成后输出当前树的前序遍历结果。
2. 删除单个数（若有多个相同的数，删除一个即可），删除完成后输出当前树的前序遍历结果。
3. 输出某数的排名，排名从1开始。
4. 输出排名为k的数。
5. 输出数的前驱元素。
6. 输出数的后继元素。

将你的实现填写在 `treap.h` 中，在项目的根目录执行 `make run` 进行评测。

开放性问题

1. 请你针对较大的测试用例(比如data/input-3.txt或者自行生成更大的测试用例)分析程序中耗时较长的操作都有哪些(比如分裂，合并），并给出在程序运行期间这些耗时较长的操作的平均耗时（以时钟周期为单位）。你可以选择手动breakdown或者借助Linux平台的各种性能分析工具。
   a. 如果选择手动breakdown，你可以使用rdtsc指令来获取某段机器指令的耗时(比如下面的示例）。
   b. 如果选择使用性能分析工具，请在文档中说明你的使用过程。

```
static inline uint64_t rdtsc()
{
    uint32_t low, high;
    asm volatile ("rdtsc" : "=a" (low), "=d" (high));
    return ((uint64_t)high << 32) | low;
}

int main() {
    uint64_t start, end;
    start = rdtsc();
    // operation you want to breakdown
    end = rdtsc();
    printf("The program took %lu cycles to run.\n", end - start);
}
```

2. 请你画出平衡树高度与key数量的关系，使用一张横坐标为key数量，纵坐标为树高度的曲线图表示。

上述的两个问题写在一份文档中，格式为PDF，字数不限。将文档放在项目的根目录后，将整个项目压缩为zip文件上传。

> 请注意：更多的字数并不意味着更高的得分。我们更关注你分析问题的思路。

提示与注意事项

1. Treap的实现可以使用最大堆或最小堆。为保证树结构的一致，你应该使用最大堆实现无旋Treap。
2. Treap在存储重复元素的时候可以选择将重复元素存储在多个节点，也可以选择将重复元素存储在一个节点并维护计数器。为了保证树的结构一致，你的实现应该采用后一种方式。
3. 合并时如果左子树根节点的权重与右子树根节点的权重相同，则将右子树合并到左子树中。
4. 请使用Treap.h中提供的 `RandGenerator::treap_rand()` 生成随机数。
5. 在输出树的前序遍历过程中：
   a. 如果树为空，则输出 `empty` 。
   b. 如果树中存在重复元素，仅输出1次即可。
6. 在以下几种情况下，你应该输出-1：
   a. 查询某个值的排名时，如果该值不存在
   b. 树中不存在第k个元素
   c. 不存在前驱元素
   d. 不存在后继元素
7. 注意在 `clear()` 函数中重置随机数生成器。

构建方式

使用Linux/MacOS的同学直接运行以下命令即可：

```
# 构建可执行文件
make build

# 运行测试，当看到 "Tests All Passed :)" 说明3个测试用例全部通过。
make run

# 使用gdb调试
make gdb
```

使用Windows的同学建议使用WSL运行以上命令。