对于无旋 Treap 的研究

苏泓燃

2022 年 2 月 13 日

目录

1 无旋 Treap 的结构 **1**

1.1 Treap 的结构 1

1.2 无旋 Treap 的维护方式 1

2 无旋 Treap 的基本操作 **2**

2.1 split 操作 2

2.2 merge 操作 2

3 无旋 Treap 的其他操作 **4**

3.1 insert 操作 4

3.2 delete 操作（单点删除） 4

3.3 delete 操作（键值删除） 4

3.4 rank 操作 5

3.5 arcrank 操作 5

3.6 prefix 操作 6

3.7 suffix 操作 6

4 总结 **7**

1 无旋 Treap 的结构

1.1 Treap 的结构

Treap 是一种平衡树[1]，其遵守二叉搜索树[2]的基本结构。

在此结构基础上，Treap 为每一个节点赋予随机权值 rnd，并保证每一个节点的 rnd 值小于等于其子树内每一个节点的 rnd 值。

定理 1.1.1. *Treap 的结构稳定。*

定理 1.1.1 的证明，可以类比快速排序[3]的证明方法，在此不做证明。

定理 1.1.2. *Treap 的深度级别为 $O(\log n)$。*

定理 1.1.2 可以理解为定理 1.1.1 的推论，在此不做证明。

定理 1.1.3. *对于每一种固定的 rnd 赋值方案，若保证 rnd 值不重复且节点权值不重复，Treap 都有唯一的形态。*

定理 1.1.3 的证明亦可类比快速排序的过程，简略过程如下：

对于每一个节点集合，都能确定唯一的根节点，且对于每一个剩余节点，都能判断其属于根节点的左子树或右子树，而对于根节点的左右子树，都能确定唯一的根节点，递归求证可知 Treap 形态唯一。

1.2 无旋 Treap 的维护方式

无旋 Treap，又名 FHQ-Treap[4]，是一种 Treap 的维护方式，其核心操作有如下：

1. split 操作，该操作可以在 $O(\log n)$ 时间复杂度内将一个 Treap 按键值分裂为两个 Treap，满足其中一个 Treap 中所有节点权值均小于等于键值，另一个 Treap 中所有节点权值均大于键值。
2. merge 操作，该操作可以在 $O(\log n)$ 时间复杂度内将两个 Treap 合并为一个 Treap，满足其中一个 Treap 中所有节点权值均小于另一个 Treap 中任意节点权值。

以上操作的具体过程会在本篇第二章叙述。

其中 split 操作的分裂标准也可以更换为目标 Treap 大小，前提是维护了每个节点的子树大小。

---

[1] 平衡树，一类数据结构的统称。

[2] 二叉搜索树，一种数据结构，平衡树的雏形。

[3] 快速排序，一种在 $O(n \log n)$ 时间复杂度内排序一个长度为 n 的数列的算法。

[4] FHQ-Treap，由范浩强在 2016 年总结并发表的 Treap 维护方法。

2 无旋 Treap 的基本操作

2.1 split 操作

操作步骤如下：

1. 若当前 Treap 为空，则返回空。
2. 若当前节点权值小于等于键值，则以相同键值递归分裂其右子树，将当前节点的右子树设为右子树分裂结果的左半 Treap，更新当前节点信息，将当前节点分裂结果设为当前 Treap 与右子树分裂结果的右半 Treap。
3. 若当前节点权值大于键值，则以相同键值递归分裂其左子树，将当前节点的左子树设为左子树分裂结果的右半 Treap，更新当前节点信息，将当前节点分裂结果设为左子树分裂结果的左半 Treap 与当前 Treap。

定理 2.1.1. *split 操作不会破坏 Treap 的性质。*

定理 2.1.1 的简略证明如下：

由于 split 操作没有改变人两个节点的辈分关系，故没有破坏 rnd 的性质，由于 split 操作没有将向任意节点的任意子树中插入新节点，故没有破坏权值关系。

定理 2.1.2. *split 操作结果中的左半 Treap 中所有节点权值均小于等于键值，右半 Treap 中所有节点权值均大于键值。*

定理 2.1.2 的简略证明如下：

步骤 2. 中若右子树分裂满足定理 2.1.2，由于当前节点权值小于等于键值，当前节点左子树内所有节点权值均小于当前节点权值，故当前分裂结果的左半 Treap 中权值均小于等于键值，右半 Treap 中权值均大于键值，步骤 3. 同理。

定理 2.1.3. *split 操作时间复杂度为 $O(\log n)$。*

定理 2.1.3 的简略证明如下：

每一次递归都会使当前 Treap 的最大深度至少减少 1，根据定理 1.1.2，Treap 深度为 $O(\log n)$ 级别，故递归次数为 $O(\log n)$，级别，由于每次递归的时间复杂度为 $O(1)$，故总复杂度为 $O(\log n)$。

2.2 merge 操作

操作步骤如下：

1. 若当前两个 Treap 中有一个为空，则将另一个 Treap 作为合并结果返回。
2. 若左半 Treap 根节点 rnd 值小于右半 Treap 根节点 rnd 值，递归合并左半 Treap 的右子树与右半 Treap，将左半 Treap 的右子树设为左半 Treap 右子树的合并结果，并将当前合并结果设为左半 Treap。

3. 若左半 Treap 根节点 rnd 值大于等于右半 Treap 根节点 rnd 值，递归合并左半 Treap 与右半 Treap 的左子树，将右半 Treap 的左子树设为右半 Treap 左子树的合并结果，并将当前合并结果设为右半 Treap。

定理 2.2.1. *merge 操作不会破坏 Treap 的性质。*

定理 2.2.1 的简略证明如下：

步骤 2. 中由于右半 Treap 根节点 rnd 值大于等于左半 Treap 根节点 rnd 值，而右半 Treap 中所有节点的 rnd 值均大于等于右半 Treap 根节点 rnd 值，故合并后左半 Treap 中所有节点 rnd 值均小于左半 Treap 根节点 rnd 值，步骤 3. 同理。由于 merge 的左右 Treap 满足左 Treap 中所有节点权值均小于右半 Treap 中任意节点权值，故步骤 2. 中合并后左半 Treap 的右子树内节点权值均大于左半 Treap 根节点权值，步骤 3. 同理。

定理 2.2.2. *merge 操作时间复杂度为* $O(\log n)$。

定理 2.2.2 的简略证明如下：

每进行一次递归，左右 Treap 的深度之和至少下降 1，当深度值和为 0 时不可能继续递归，根据定理 1.1.2，左右 Treap 深度之和为 $O(\log n)$ 级别，故递归次数为 $O(\log n)$ 级别，由于每次递归时间复杂度为 $O(1)$，故总复杂度为 $O(\log n)$。

3 无旋 Treap 的其他操作

3.1 insert 操作

insert 操作支持在 $O(\log n)$ 时间复杂度内将一个节点插入一个 Treap 中。
具体步骤如下：

1. 将 Treap 按新节点的权值分裂为左半 Treap 与右半 Treap。
2. 将左半 Treap 与新节点合并。
3. 将步骤 2. 中合并结果与右半 Treap 合并，并将结果作为答案。

定理 3.1.1. *insert 操作时间复杂度为* $O(\log n)$。

定理 3.1.1 证明如下：
步骤 1. 时间复杂度为 $O(\log n)$，步骤 2. 时间复杂度为 $O(\log n)$，步骤 3. 时间复杂度为 $O(\log n)$，故总时间复杂度为 $O(\log n)$。

3.2 delete 操作（单点删除）

该 delete 操作支持在 $O(\log n)$ 的时间复杂度内删除一个权值为键值的节点。
具体步骤如下：

1. 将 Treap 按键值-1 分为左半 Treap 与右半 Treap。
2. 将步骤 1. 中的右半 Treap 按键值分为左半 Treap 与右半 Treap。
3. 将步骤 2. 中左半 Treap 中删去一个点，方法任意，一般方法有二：
   - 将目标 Treap 按子树大小分为两个 Treap，取右半 Treap 作为结果。
   - 将目标 Treap 左子树与右子树合并作为结果。
4. 合并步骤 1. 中左半 Treap 与步骤 3. 中结果 Treap。
5. 合并步骤 4. 中结果 Treap 与步骤 2. 中右半 Treap，并将结果 Treap 设为答案。

定理 3.2.1. *该 delete 操作时间复杂度为* $O(\log n)$。

定理 3.2.1 证明如下：
所有步骤时间复杂度均为 $O(\log n)$，故总时间复杂度为 $O(\log n)$。

3.3 delete 操作（键值删除）

该 delete 操作支持在 $O(\log n)$ 的时间复杂度内删除所有权值为键值的节点。
具体步骤如下：

1. 将 Treap 按键值-1 分为左半 Treap 与右半 Treap。
2. 将步骤 1. 中的右半 Treap 按键值分为左半 Treap 与右半 Treap。

3. 将步骤 1. 中的左半 Treap 与步骤 2. 中的右半 Treap 合并，并将结果作为答案。

定理 3.3.1. *该 delete 操作时间复杂度为* $O(\log n)$。

定理 3.3.1 证明如下：
所有步骤复杂度均为 $O(\log n)$，故总时间复杂度为 $O(\log n)$。

3.4 rank 操作

该操作支持在 $O(\log n)$ 时间复杂度内查询某一键值在 Treap 中的排名[5]。
具体步骤如下：

1. 将 Treap 按键值-1 分为左半 Treap 与右半 Treap。
2. 将左半 Treap 的大小记为答案。
3. 将步骤 1. 中的左半 Treap 与步骤 1. 中的右半 Treap 合并。

定理 3.4.1. *rank 操作的时间复杂度为* $O(\log n)$。

定理 3.4.1 证明如下：
所有步骤的最大时间复杂度为 $O(\log n)$，故总时间复杂度为 $O(\log n)$。

3.5 arcrank 操作

arcrank 操作支持在 $O(\log n)$ 的时间复杂度内查询目标排名的节点的权值。
具体步骤如下：

1. 将 Treap 按子树大小为目标排名-1 分为左半 Treap 与右半 Treap。
2. 将步骤 1. 中右半 Treap 按子树大小为 1 分为左半 Treap 与右半 Treap。
3. 将步骤 2. 中左半 Treap（实际上是一个节点）的根节点权值设为答案。
4. 将步骤 2. 中左半 Treap 与步骤 2. 中右半 Treap 合并。
5. 将步骤 1. 中左半 Treap 与步骤 4. 中结果 Treap 合并。

定理 3.5.1. *arcrank 操作复杂度为* $O(\log n)$。

定理 3.5.1 证明如下：
所有步骤的最大时间复杂度为 $O(\log n)$，故总复杂度为 $O(\log n)$。

[5]排名，即比键值小的节点个数

3.6 prefix 操作

prefix 操作支持在 $O(\log n)$ 时间复杂度内查询键值在 Treap 中的前驱[6]。
具体步骤如下：

1. 将 Treap 按键值-1 分为左半 Treap 与右半 Treap。
2. 将步骤 1. 中左半 Treap 按子数大小为左半 Treap 大小-1 分为左半 Treap 与右半 Treap。
3. 将步骤 2. 中右半 Treap（实际上是一个节点）根节点的权值设为答案。
4. 合并步骤 2. 中左半 Treap 与步骤 2. 中右半 Treap。
5. 合并步骤 4. 中结果 Treap 与步骤 1. 中右半 Treap。

定理 3.6.1. *prefix 操作的时间复杂度为* $O(\log n)$。

定理 3.6.1 的证明如下：
所有步骤的最大时间复杂度为 $O(\log n)$，故总复杂度为 $O(\log n)$。

3.7 suffix 操作

suffix 操作支持在 $O(\log n)$ 时间复杂度内查询键值在 Treap 中的后继[7]。
具体步骤如下：

1. 将 Treap 按键值分为左半 Treap 与右半 Treap。
2. 将步骤 1. 中右半 Treap 按子树大小为 1 分为左半 Treap 与右半 Treap。
3. 将步骤 2. 中左半 Treap（实际上是一个节点）根节点的权值记为答案。
4. 合并步骤 2. 中的左半 Treap 与步骤 2. 中的右半 Treap。
5. 合并步骤 1. 中的左半 Treap 与步骤 4. 中的结果 Treap。

定理 3.7.1. *suffix 操作的时间复杂度为* $O(\log n)$。

定理 3.7.1 的证明如下：
所有步骤的最大时间复杂度为 $O(\log n)$，故总复杂度为 $O(\log n)$。

[6]前驱，即比键值小的值中的最大值
[7]后继，即比键值大的值中的最小值

4 总结

无旋 Treap 作为平衡树来说功能强大，但是相对于其他平衡树，无旋 Treap 的常数较大，所以使用需要仔细。

无旋 Treap 的另一大优点是可以很简便地支持可持久化，相对于其他平衡树来说。

而且无旋 Treap 在平衡树中算思维难度较为简单的一种，而且其代码长度也要比其他平衡树短，所以如果对常数要求不高的话，无旋 Treap 不失为一种好选择。

对于其他操作中的 rank 操作、arcrank 操作、prefix 操作与 suffix 操作，其实可以套用带旋 Treap 的实现方法，即自顶向下查询答案，而且这种方法常数要小很多，本文中使用 split 操作与 merge 操作的实现方法仅仅是为了展现无旋 Treap 的思想