平衡树 Treap

二叉搜索树 + 随机堆优先级：旋转与无旋 FHQ 全解

HGNU-ACM 实验室算法组

承接第 6 章树上结构：BST 退化痛点 + 旋转 Treap + 无旋 FHQ-Treap + 2 道真题精讲





本章目录

BST 痛点与旋转 Treap 原理

无旋 FHQ-Treap：按值分裂与合并

第 1 节从“有序输入把普通 BST 卡成单链”出发，讲清随机优先级 `pri` 与左右旋转的等价性；第 2 节详解更简洁强大的无旋 FHQ-Treap（`split + merge`）实现全套操作。

本章目录（续）

平衡树两大真题精讲

常见错误、检查清单与练习安排

第 3 节精讲《营业额统计》与《普通平衡树》两道真题；第 4 节总结平衡树高频错误、提交检查清单，并预告下一篇《05_ 动态规划》第 1 章《线性 DP 与背包模型》。

承接第 6 章：当树不再固定，而是要动态插删与查名次

生活问题：实时游戏排行榜（支持 10^5 次动态操作）

假设你是系统后台，要维护所有玩家的实时分数，随时会收到 6 种请求：

- 插入一个新分数 x、删除一个分数 x（若有多个只删一个）；
- 查询分数 x 当前排第几名（排名）、查询当前第 k 名是多少分（第 k 小）；
- 查询比 x 小的最大分数（前驱）、比 x 大的最小分数（后继）。

- 在第 6 章我们处理的树结构是静态不变的；而这里的集合每天每刻都在插入和删除！

本页要点

我们需要一种“既能像有序数组一样二分/查第 k 名，又能像链表一样快速插删”的动态结构！

为什么有序数组、链表、STL set 都不够用？

数据结构	插入 / 删除	查前驱 / 后继	查排名 / 第 k 小	$n = 10^5$ 能否过？
有序数组	$O(n)$ 挪动元素	$O(\log n)$	$O(1)/O(\log n)$	插入太多会 **TLE**
链表	$O(1)$（找到后）	$O(n)$	$O(n)$	查询太多会 **TLE**
STL `set/multiset`	$O(\log n)$	$O(\log n)$	不支持（只能 $O(n)$ 数）	查第 k 小会 **TLE**
平衡树 **Treap**	$O(\log n)$	$O(\log n)$	$O(\log n)$	全部 $O(\log n)$ 通过！

本页要点

只查前驱后继可以用 `set`；一旦题目要求“查排名”或“查第 k 小”，就必须手写平衡树！

复习普通二叉搜索树（BST）：正常长什么样？

BST 的一条核心规则：左子树全 $<$ 根 $<$ 右子树

比如依次插入 3，1，5，2，4：

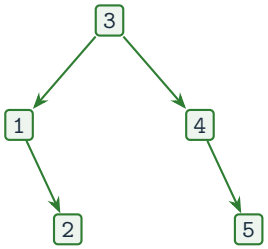
- 3 当根；1 < 3 放左；5 > 3 放右；2 < 3 且 2 > 1 放 1 的右边；4 > 3 且 4 < 5 放 5 的左边。
- 想要查找 4 在不在树里：从根 3 出发，向右到 5，再向左到 4，只走 2 步就找到了！
- 如果整棵树左右均匀，有 n 个节点时树高只有 $\approx \log_2 n$（$n = 10^5$ 时最多走 17 步）。

本页要点

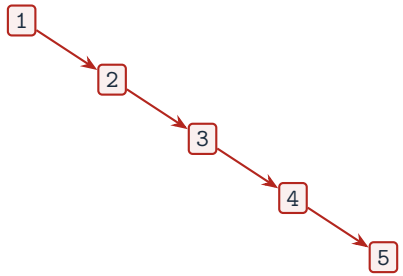
普通 BST 在“数据随机”时很快，但它没有任何自我保护机制，遇到有序数据就会原形毕露！

复习普通二叉搜索树（BST）与它的致命退化

正常随机输入（树高 $O(\log n)$）：



依次插入 1，2，3，4，5（退化成单链！）：



- 暴力 **BST** 痛点：左子树 $<$ 根 $<$ 右子树。一旦题目按 $1, 2, \ldots, 10^5$ 递增插入，新数全挂在右儿子上，树高变成 10^5，查询一次走 10^5 步，总操作 10^{10} 必 **TLE**！

本页要点

普通 BST 的树高完全依赖输入顺序；为了不怕恶意构造的有序数据，我们必须打乱它的形态！

先听懂五个新术语（大白话版）

术语	大白话解释
BST 性质（搜索树序）	对任意节点，左边所有数 $\leq$ 它 $\leq$ 右边所有数（中序遍历是从小到大）。
堆性质（Heap 序）	大根堆里每个爹的优先级 $\geq$ 孩子；小根堆里每个爹的优先级 $\leq$ 孩子。
随机优先级 `pri`	新建节点时用随机数生成器发一个随机整数，和输入的数值大小无关。
Treap（Tree+Heap）	一棵树同时满足两个规则：按数值 `key` 是搜索树，按随机 `pri` 是堆！
子树大小 `sz[u]`	以 u 为根的整棵子树里有多少个节点（靠它才能 $O(\log n)$ 查排名与第 k 小）。

本页要点

Treap 巧妙地把“有序输入”通过随机 `pri` 变成了等价于“随机插入”的形态！

生活类比：为什么 Treap 既要像搜索树，又要像堆？

公司座位表 vs 主管任命制度

把每个节点想象成一位员工：数值 `key` 是他的工号，随机 `pri` 是他抽签得到的权限号。

- 按工号排座位（**BST** 序）：左边的下属工号比自己小，右边的下属工号比自己大，方便按工号二分查找。
- 按抽签号定上下级（堆序）：谁的抽签号大，谁就站到上面当主管；因为抽签是随机的，没人能靠“按工号排队”把汇报链拉成 10^5 层！

本页要点

工号决定左右顺序（好查找），抽签号决定上下高度（防退化），两者互不干扰！

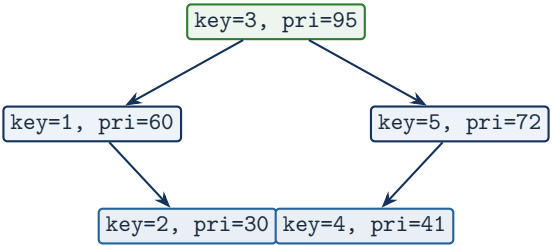
细节追问：为什么要用 mt19937_64，不能用 rand()？

随机数工具	数值范围与质量	$n = 10^5$ 实战表现
老旧 rand()	Windows 下仅 $0 \sim 32767$（15 位）	10^5 个点必有大量重复 pri，局部退化
mt19937 / mt19937_64	32 位 / 64 位梅森旋转高质量随机	几乎零碰撞，树高稳定在 $O(\log n)$

本页要点

在 C++11 及以后写平衡树，随机数一律用 mt19937 或 mt19937_64！

TikZ 图解：一棵同时满足 BST 序与大根堆序的 Treap

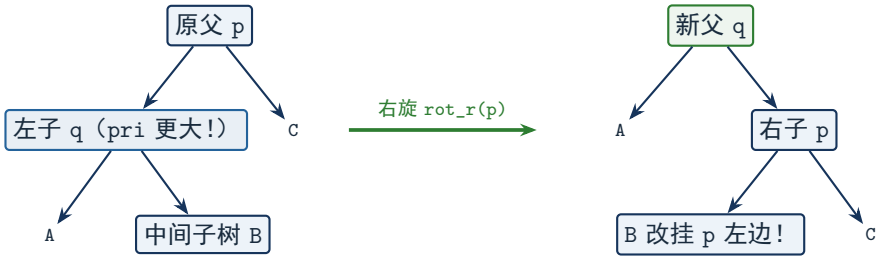


- 横着看（中序遍历）：1 -> 2 -> 3 -> 4 -> 5，严格满足左 $<$ 根 $<$ 右的 BST 序！
- 竖着看（父子关系）：95 > 60 > 30 且 95 > 72 > 41，严格满足大根堆序！
- 为什么高度是 $O(\log n)$？在节点权值和随机优先级都确定后，树形唯一等价于“按 pri 从大到小依次插入普通 BST”——因为 pri 随机，等价于随机打乱插入顺序，期望树高 $\approx 2 \ln n$！

本页要点

即使输入严格按 $1, 2, 3, 4, 5$ 递增，随机 pri 也会自动把最大优先级的点提上去当根！

TikZ 图解：左孩子 pri 更大时，如何用“右旋”拎上来？

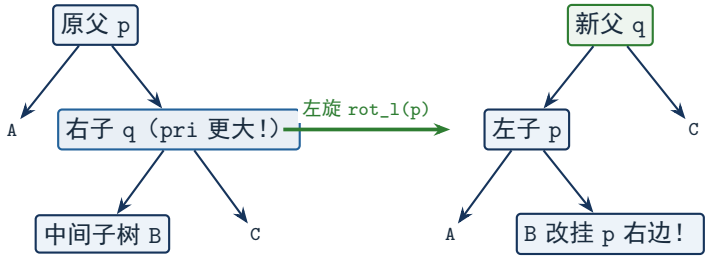


- 旋转前中序：A < q < B < p < C；旋转后中序依然是：A < q < B < p < C!
- 左子 q 升为新父，原父 p 降为 q 的右子，原本 q 的右子树 B 过继给 p 当左子!

本页要点

记住右旋四行代码：`q = tr[p].l; tr[p].l = tr[q].r; tr[q].r = p; return q;`!

TikZ 图解：右孩子 pri 更大时，如何用“左旋”拎上来？



- 左旋是右旋的完全镜像：右子 q 升为新父，原父 p 降为 q 的左子，q 的左子树 B 过继给 p 当右子。
- 旋转前后的中序遍历永远都是 A < p < B < q < C，绝不破坏 BST 的大小关系！

本页要点

口诀：想把哪个孩子拎上来，就往反方向旋转（左子大就右旋，右子大就左旋）！

手算跟踪表：依次插入递增序列 $1, 2, 3, 4$ 看旋转如何拯救单链

设 4 个数分配到的随机优先级分别为：pri[1]=20，pri[2]=50，pri[3]=35，pri[4]=80（大根堆）。

步序	插入元素	刚挂上去时的位置	堆序检查与旋转后结果
1	1（pri=20）	空树，直接作为根	root = 1
2	2（pri=50）	挂在 1 的右子	50 > 20！对 1 左旋 $\Rightarrow$ 2 成新根，1 为左子
3	3（pri=35）	挂在 2 的右子	35 < 50，满足堆序，不旋转！
4	4（pri=80）	挂在 3 的右子	80 > 35 对 3 左旋；回溯 80 > 50 再对 2 左旋！
终止	插入结束	新根变为 4，左子为 2	最终树高从 4 压成平衡形态，全树满足堆序！

本页要点

递归回溯时顺手检查“孩子 pri 是否大于自己”，就能一路把新节点旋到它该在的高度！

为什么旋转 Treap 一定是正确的？（不变量三段论）

论证环节	严格证明要点（维护 BST 序与大根堆序双不变量）
初始化	空树（仅哨兵 `tr[0]`）天然满足 BST 序与堆序。
保持	插入新节点时按 `x < key` 往左、`x > key` 往右，到达空位新建叶子，BST 序保持；回溯时仅在“唯一刚插入的那一侧”可能出现子 `pri` > 父 `pri`，执行一次左旋或右旋：既修复了当前父子的堆序，又因旋转保持中序不变而丝毫不破坏 BST 序。
终止	递归回到 `root` 后，全树所有节点同时满足 BST 序与堆序，期望高度 $O(\log n)$。

本页要点

旋转操作是“既修堆序、又保中序”的桥梁，二者兼得！

旋转 Treap 模板代码（1/2）：结构定义与左右旋

```
struct Node { long long key; uint64_t pri; int l = 0, r = 0; };
vector<Node> tr{{0, 0, 0, 0}};                          // 0 号下标作为统一空节点哨兵，
    防野指针 RE
mt19937_64 rng{114514};                                 // 64 位高质量随机数生成器
int root = 0;

int rot_r(int p) {                                      // 右旋：把左孩子 q 拎上来当新根
    int q = tr[p].l; tr[p].l = tr[q].r; tr[q].r = p; return q;
}
int rot_l(int p) {                                      // 左旋：把右孩子 q 拎上来当新根
    int q = tr[p].r; tr[p].r = tr[q].l; tr[q].l = p; return q;
}
```

本页要点

用 vector 下标替代原生指针，约定 0 号节点为空，代码短且绝不发生内存泄漏！

旋转 Treap 模板代码（2/2）：插入与最近值查找

```
int insert(int p, long long x) {
    if (!p) { tr.push_back({x, rng()}); return (int)tr.size() - 1; }
    if (x < tr[p].key) {
        tr[p].l = insert(tr[p].l, x);
        if (tr[tr[p].l].pri > tr[p].pri) p = rot_r(p); // 左子优先级更高，右旋修正
    } else if (x > tr[p].key) {
        tr[p].r = insert(tr[p].r, x);
        if (tr[tr[p].r].pri > tr[p].pri) p = rot_l(p); // 右子优先级更高，左旋修正
    }
    return p;                                          // 必须返回当前子树的新根！
}
```

本页要点

调用时一定要写成 `root = insert(root, x);`！忘记接住返回值会导致树根丢失！

逐行拆解：为什么右旋 rot_r 的四行代码顺序绝对不能换？

行序	代码	为什么必须放在这一步？
1	`int q = tr[p].l;`	先记住左子 q，否则下一行改写 tr[p].l 后就找不到 q 了
2	`tr[p].l = tr[q].r;`	先把中间子树 B 过继给 p 的左边，再覆盖 tr[q].r
3	`tr[q].r = p;`	把原父 p 挂到新父 q 的右边
4	`return q;`	把新根 q 返回给外层调用者接住

本页要点

先保 q、再过继 B、再改 r[q]、最后返回 q，任何两行交换都会丢子树！

上机验证：3 组测试场景与反事实追问

旋转 Treap 上机必测 3 组场景

1. 严格递增插入：依次插入 $1, 2, \ldots, 100000$ 并查询最近值，耗时 < 0.1 秒不会 TLE；
2. 含负值测试：插入 `-10`，`0`，`10` 后查 `-7` 的最近值，预期差值为 `|-10 - (-7)| = 3`；
3. 重复值插入：先插 `5` 再查 `5` 的最近值，预期返回 `0`（本模板不存重复节点）。

反事实追问

追问：如果只写 `t.insert(t.root, x);`，忘记写 `t.root = ...` 会怎样？
回答：插第一个数时 `t.root` 永远是 `0`；哪怕非空，一旦把根旋下去，`t.root` 仍指着旧根，上面一半节点直接失联，必 **WA**！

本页要点

“旋转改了树根，调用必接返回值”是写递归树形结构的铁律！

为什么沿 BST 查找路径就能包住 x 的前驱和后继？

搜索路径上的天然左右夹逼

查找 x 时，从根出发每一步只有两种走法：

- 如果 `x < tr[p].key`，说明 p 比 x 大（是 x 的候选后继），往左子树走；
- 如果 `x > tr[p].key`，说明 p 比 x 小（是 x 的候选前驱），往右子树走。

- 整个查找路径上“最后一次往左拐的祖先”就是 x 的后继，“最后一次往右拐的祖先”就是 x 的前驱！因此只需一路用 `llabs(tr[p].key - x)` 取最小值即可！

本页要点

只要沿着根走到底，前驱和后继必定都在你经过的这条路上，一个也跑不掉！

扩展：旋转 Treap 如何维护子树大小 sz[p] 与怎么删除？

1. 如果要查第 k 大，旋转末尾怎么更新 sz？

每次 rot_r(p) 或 rot_l(p) 改变了 p 和 q 的父子关系：

- 必须先更新降下去的原父 p：pull(p)；
- 再更新升上来的新父 q：pull(q)；（顺序反了 sz[q] 就会用到旧的 sz[p]！）

2. 旋转 Treap 怎么删除节点？

找到目标节点后，如果它有两个孩子，就不断把 pri 更大的孩子往上旋，把它一步步旋到叶子再删掉——写起来要分好几种情况，不如第 2 节的无旋 FHQ-Treap 简洁！

本页要点

旋转更新 sz 记住“先子 p 后父 q”；嫌旋转删除麻烦，下一节教你无旋写法！

旋转 Treap 高频易错对照（WA / TLE / RE 定位）

易错点	错误写法	正确写法	评测结果
忘记接住新根	只写 `insert(root, x);`	`root = insert(root, x);`	**WA**
旋转孙子接反	`rot_r` 里写 `l[p] = l[q]`	右旋过继 `r[q]` 到 `l[p]`	**WA**
用老旧 `rand()`	`pri = rand()`（15 位易撞车）	用 `mt19937_64 rng`	**TLE/WA**
忘设 0 号空节点	空树直接访问 `tr[0].l` 没预留	`vector<Node> tr{{0,0,0,0}}`	**RE**

本页要点

旋转 Treap 插入虽然短，但如果要支持“删除、区间翻转、查第 k 大”，无旋 FHQ-Treap 更好写！

痛点升级：旋转 Treap 删除太绕，有没有“不用旋转”的平衡树？

旋转 Treap 的短板

用左右旋虽然能实现插入，但如果要支持删除节点、区间反转或可持久化，旋转需要不断讨论左右子树情况，代码又长又容易写错。

- 范浩强（**FHQ**）的绝妙发明：完全抛弃左旋和右旋！
- 只靠两个超级短小的函数——按值分裂（`split`）和按序合并（`merge`），就能像搭积木一样拼出所有平衡树操作（插、删、排名、第 k 大、前驱、后继）！

本页要点

无旋 FHQ-Treap 既保留了“BST 序 + 随机堆序”的高效平衡，又让代码量减少一半！

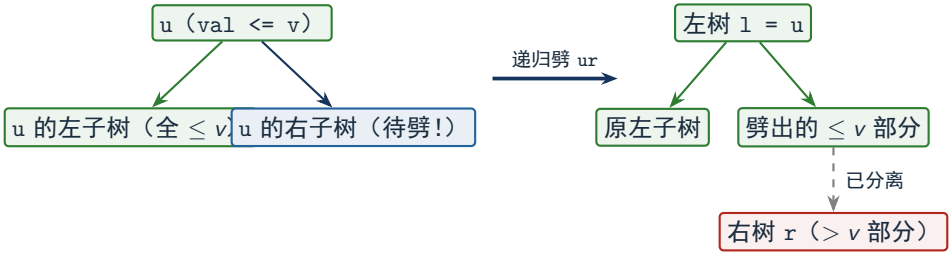
先听懂五个新术语（大白话版）

术语	大白话解释
按值分裂 split(u, v, l, r)	给定分界线 v，把以 u 为根的树劈成两棵：左树 l 全是 $\le v$ 的数，右树 r 全是 $> v$ 的数。
按序合并 merge(l, r)	把两棵满足“l 中所有数 $\le$ r 中所有数”的树重新拼成一棵树。
小根堆优先值	本节采用小根堆：pri 越小的节点越靠近树根（与旋转版大根堆对称，效果完全一样）。
pull(u)（上传更新）	每次分裂或合并改动了孩子指针后，重新计算 sz[u] = 1 + sz[l] + sz[r]。
哨兵空节点 tr[0]	约定 tr[0].sz = 0，避免空树判空漏写造成越界 RE。

本页要点

记住核心前提：merge(l, r) 要求左树 l 的所有值必须 $\le$ 右树 r 的所有值！

TikZ 图解：split(u, v, l, r) 如何一刀把树劈成两半？



- 如果 tr[u].val <= v：说明 u 和它的整个左子树一定全 $\leq v$，统统归入左树 l！只需继续递归劈 u 的右子树：split(tr[u].r, v, tr[u].r, r);
- 如果 tr[u].val > v：同理，u 和整个右子树全归入右树 r，继续递归劈 u 的左子树！

本页要点

每次往下递归只走左或右一侧，深度最多 $O(\log n)$，并在回溯时写 pull(u) 刷新 sz！

手算跟踪表：用 5 个点的树执行 split(root, 5, a, b)

树结构：根 6（左 3，右 8）；点 3 左 1、右 5。分裂阈值 $v = 5$。

层	当前 u	比较	归属决策	下一步递归调用
1	6	$6 > 5$	b = 6（归右），保右 8	split(3, 5, a, l[6])
2	3	$3 \le 5$	a = 3（归左），保左 1	split(5, 5, r[3], l[6])
3	5	$5 \le 5$	r[3] = 5（归左）	split(0, 5, r[5], l[6])
4	0	!u	r[5]=0, l[6]=0	结束！a={1,3,5}, b={6,8}

本页要点

整个分裂过程就像拉链一样从根往下划开，把原来的树干净利落地分成 ≤ 5 和 > 5 两半！

TikZ 图解：merge(l, r) 如何把两半安全拼回一棵树？



- 因为已知 l 里所有值 $\leq$ r 里所有值，所以无论谁当新根，l 一定在 r 的左边！
- 谁的 pri 小谁当爹（维护小根堆）：若 tr[l].pri < tr[r].pri，则 l 当新根，l 的左子树不动，把 l 的右子树和 r 合并起来挂到 tr[l].r！

本页要点

merge 从头到尾不需要比较两个节点的数值 val，只比较随机 pri 就够了！

手算跟踪表：把 a={1,3,5} 和 b={6,8} 重新 merge

设 a 根为 3（pri=20，右子 5 的 pri=45）；b 根为 6（pri=30）。

层	(l, r)	pri 比较	新根决策	下一步递归
1	(3, 6)	20 < 30	3 当根，保左 1	r[3] = merge(5, 6)
2	(5, 6)	45 > 30	6 当根，保右 8	l[6] = merge(5, 0)
3	(5, 0)	r == 0	返回 5	回溯 pull(6)，pull(3)
终止	-	-	合并完成，根 3	满足 BST 序与小根堆序！

本页要点

写完 merge 一定要顺手 pull(l) 或 pull(r)，这样子树大小 sz 永远精确！

为什么无旋 FHQ-Treap 一定是正确的？（不变量三段论）

论证环节	严格证明要点
`split` 不变量	$\le v$ 归 `l`、$> v$ 归 `r`，父子先后不变，两树各自保持 BST 序与小根堆序。
`merge` 不变量	前提“`l` 全值 $\le$ `r` 全值”下，中序保持 `l` 在左、`r` 在右；选 `pri` 更小者做根保堆序。
复杂度	随机 `pri` 保证期望树高 $O(\log n)$，每次 `split`/`merge` 沿一条树高路径走，期望 $O(\log n)$。

本页要点

“分裂保值域、合并保中序、随机保树高”，三句话彻底讲透 FHQ-Treap 的正确性！

搭积木（1/3）：用 split + merge 实现“插入”与“删除单个”

1. 插入一个数 x

- 新建节点 id（val=x，sz=1，pri=rng()）；split(root，x，a，b)；劈成 $\le x$ 与 $> x$；
- 拼回去：root = merge(merge(a，id)，b)；!

2. 删除一个数 x（若有多个相同的 x，只删掉一个！）

- 劈两刀孤立等于 x 的子树 b：split(root，x，a，c)；split(a，x - 1，a，b)；
- 扔掉 b 的树根（只删一个）：b = merge(tr[b].l，tr[b].r)；
- 再把三段拼回去：root = merge(merge(a，b)，c)；!

本页要点

因为 b 中所有节点权值都等于 x，合并 b 的左右儿子就等价于只删掉 1 个 x！

手算模拟：用 split + merge 插入 x = 4 与删除重复数 x = 5

模拟场景	集合 {1, 3, 5, 5, 6, 8} 的分裂与合并状态变化
插入 x = 4	split(root, 4, a, b) 得到 a={1, 3}、b={5, 5, 6, 8}; 新建节点 id={4}，先 merge(a, id) 得 {1, 3, 4}; 再 merge(..., b) 得新树 {1, 3, 4, 5, 5, 6, 8}。
删一个 x = 5	split(root, 5, a, c) 得 a={1,3,4,5,5}、c={6,8}; split(a, 4, a, b) 得 a={1,3,4}、b={5,5}; b = merge(l[b], r[b]) 扔掉根，b 变成 {5}，再合并三段！

本页要点

删节点只扔掉 b 的根而保留其左右子，恰好达到“有多个相同数时只删一个”的要求！

搭积木（2/3）：用 split + merge 实现查排名 rank

怎么查 x 的最小排名（严格小于 x 的个数 $+1$）？

1. 按 `x - 1` 分裂：`split(root, x - 1, a, b);`；
2. 此时左树 `a` 里装的全部是严格小于 x 的数，无论树里有没有 x、有几个 x，它们都在右树 `b` 里；
3. 因此 x 的最小排名就是 `tr[a].sz + 1`！
4. 最后一定要拼回去：`root = merge(a, b);`！

本页要点

按 `x - 1` 分裂就能直接把所有 $< x$ 的元素数出来，哪怕 x 不在树里也成立！

搭积木（3/3）：用 split + merge 实现前驱与后继

操作	分裂阈值与读取答案	收尾恢复
前驱（$< x$ 最大）	`split(root, x - 1, a, b)`；此时 a 全 $< x$，取 a 中最大：`kth(a, tr[a].sz)`	`root = merge(a, b);`
后继（$> x$ 最小）	`split(root, x, a, b)`；此时 b 全 $> x$，取 b 中最小：`kth(b, 1)`	`root = merge(a, b);`

- 为什么前驱劈 `x - 1` 而后继劈 `x`？因为 `split(u, v, l, r)` 是把 $\le v$ 归左边、$> v$ 归右边！所以劈 `x - 1` 让左边全 $< x$，劈 `x` 让右边全 $> x$！

本页要点

查询类操作劈开读完答案后，一定要立刻 `root = merge(a, b);` 把树拼回原样！

第 k 小函数 kth(u，k) 原理与手算跟踪表

以 5 点树（中序：1，3，5，6，8）为例，调用 kth(root，4) 与 kth(root，5)：

轮次	当前 u（val）	期望 k	左子树 lsz	分支决策
例 A-1	点 6（val=6）	k=4	lsz=3（1,3,5）	k == lsz + 1！直接返回 6
例 B-1	点 6（val=6）	k=5	lsz=3	$k > 4$：k -= 4 -> 1，去右子 8
例 B-2	点 8（val=8）	k=1	lsz=0	k == 0 + 1！返回 8（终止）

本页要点

用循环写 kth(u，k) 比递归更省栈空间，只需把 lsz 和 k 分成 $<, =, >$ 三路！

FHQ-Treap 模板代码（1/3）：节点结构与 pull

```
struct Node {
    int l = 0, r = 0, sz = 1, val = 0;          // sz 为当前子树节点总数
    uint32_t pri = 0;                           // 随机优先级（小根堆）
};

vector<Node> tr(1);                             // tr[0] 作空节点哨兵，main 设
    tr[0].sz = 0
mt19937 rng(20261568);

void pull(int u) {
    tr[u].sz = 1 + tr[tr[u].l].sz + tr[tr[u].r].sz;
}
```

本页要点

一定要在 `main()` 开头写 `tr[0].sz = 0;`，否则默认构造的 `sz=1` 会污染所有排名！

FHQ-Treap 模板代码（2/3）：按值分裂 split

```
void split(int u, int v, int &l, int &r) {
    if (!u) { l = r = 0; return; }                  // 空树直接返回两个空根
    if (tr[u].val <= v) {                           // u 及左子树全 <= v，归入左树 l
        l = u; split(tr[u].r, v, tr[u].r, r);       // 继续劈右子树，接回 tr[u].r
    } else {                                        // u 及右子树全 > v，归入右树 r
        r = u; split(tr[u].l, v, l, tr[u].l);       // 继续劈左子树，接回 tr[u].l
    }
    pull(u);                                        // 回溯时刷新子树大小
}
```

本页要点

注意 `int &l, int &r` 是引用传参，才能在递归时把劈出来的子树根接回父节点！

FHQ-Treap 模板代码（3/3）：合并 merge 与第 k 小 kth

```
int merge(int l, int r) {
    if (!l || !r) return l | r;                    // 一侧为空则返回另一侧
    if (tr[l].pri < tr[r].pri) {                   // 小根堆：pri 小的做新根
        tr[l].r = merge(tr[l].r, r); pull(l); return l;
    } else { tr[r].l = merge(l, tr[r].l); pull(r); return r; }
}
int kth(int u, int k) {
    while (true) {
        int lsz = tr[tr[u].l].sz;
        if (k <= lsz) u = tr[u].l;
        else if (k == lsz + 1) return tr[u].val;
        else { k -= lsz + 1; u = tr[u].r; }
    }
}
```

本页要点

与 code/fhq_treap_basic.cpp 完全一致！

思路延伸（一）：按值分裂 vs 按大小分裂（区间平衡树预告）

分裂方式	判断条件	适用场景
按值分裂 split(u, v)	tr[u].val <= v	维护有序集合（排名、前驱、后继）
按大小分裂 split(u, k)	tr[tr[u].l].sz + 1 <= k	维护数组下标区间（区间翻转、区间加）

本页要点

学会了按值分裂，以后只需改一行判断条件就能秒懂“文艺平衡树”区间操作！

思路延伸（二）：重复元素建新节点 vs 加 cnt 计数

处理方案	优点	缺点 / 注意点
每个重复数建新节点	split/merge 逻辑最简单，零特判	空间多占一点（$n = 10^5$ 毫无压力）
节点加 cnt 计数	相同数只占 1 个节点	删除和 split 要拆点，极易写错

本页要点

在竞赛实战中，FHQ-Treap 一律首选“每个数独立建点”，省心、稳定、不易写错！

上机验证：3 组测试场景与反事实追问

FHQ-Treap 上机必测 3 组场景

1. 重复元素删一个：插入两个 5 后删一个 5，再查 5 的排名，树中仍剩一个 5；
2. 查不存在数的前驱/后继：树中只有 10，30，查 20 的前驱和后继，预期 10 和 30；
3. 递增插入 10^5 次：插入 $1..10^5$ 后查第 50000 小，预期输出 50000 且不超时。

反事实追问

追问：删除 x 时，若只写 `split(root, x, a, c)`，不写第二刀 `split(a, x - 1, a, b)` 会怎样？
回答：无法把“等于 x 的点”单独孤立出来，直接删根会把 $< x$ 的别的数一起误删掉！

本页要点

劈两刀 `split(root, x, a, c); split(a, x - 1, a, b);` 是处理重复元素的标准动作！

FHQ-Treap 易错对照与三种工具全面对比

易错点	错误写法	正确写法	评测结果
忘记清零哨兵大小	`tr(1)` 默认 `sz=1`	`main` 开头写 `tr[0].sz = 0;`	**WA**
查完忘记 `merge`	`split` 读完排名没合并回去	`root = merge(a, b);`	**WA**
`merge` 顺序写反	把右树 `b` 放左边 `merge(b, a)`	必须保证左参数 $\leq$ 右参数	**WA**
忘写 `pull(u)`	`split / merge` 末尾漏更新	修改孩子后必须立刻 `pull`	**WA**

- 工具选型：只需插删与前驱后继用 STL `set`；需要查第 k 小、排名、或未来学区间翻转时，无条件选无旋 **FHQ-Treap**！

本页要点

记住 `tr[0].sz = 0` 和“劈开必合并”，就能避开 90% 的 FHQ-Treap 失误！

例 1 | [基础] 营业额统计（一本通 1565）：先读懂要求

大白话题意

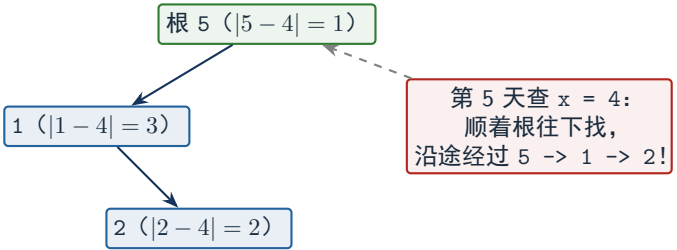
连续 n 天（$1 \le n < 2^{15} = 32768$），第 i 天的营业额为 a_i。定义第 1 天的最小波动值为 a_1；从第 2 天起，第 i 天的最小波动值为：在第 $1 \sim i-1$ 天中找一个与 a_i 最接近的数 a_j，差值 $|a_i - a_j|$。求这 n 天的最小波动值总和。

项目	具体要求
输入	第 1 行 n；接下来 n 行每行一个整数 a_i（绝对值可到 10^6）。
处理	逐天维护历史营业额集合，对第 $i \ge 2$ 天查集合里最接近 a_i 的数并累加差值。
输出	一个整数，表示最小波动值总和（不超过 2^{31}）。
样例	`6 / 5 1 2 5 4 6` ⇒ `5 + |1-5| + |2-1| + |5-5| + |4-5| + |6-5| = 12`。

本页要点

第 1 天面前没有任何历史记录，题目明确规定第 1 天直接计原值 a_1！

例 1 | 怎么想到的与建模 TikZ 图解



- 为什么不能暴力：若每天都从头扫一遍前 $i-1$ 天，总比较次数 $n^2/2 \approx 5 \times 10^8$，容易超时。
- 搜索树性质：在一棵 BST 中顺着根查找 x（小于往左、大于往右），沿途经过的所有节点里必定包含着离 x 最近的左邻居（前驱）和右邻居（后继）！取沿途 `llabs(key - x)` 的最小值即可！

本页要点

甚至不用专门写两个找前驱/后继的函数，沿搜索路径边走边取 `min` 就一举两得！

例 1 | 样例手推跟踪表

输入天数 $n = 6$，营业额序列为 5，1，2，5，4，6。

第 i 天	今日 $x = a_i$	查询前历史集合	沿搜索路径经过的节点	本天最小波动	累计总和
1	5	∅（空）	特判第一天	5	5
2	1	{5}	5	|1-5|=4	9
3	2	{1, 5}	5 -> 1	min(3,1)=1	10
4	5	{1, 2, 5}	撞见 5 == 5	0（直接返回）	10
5	4	{1, 2, 5}	5 -> 1 -> 2	min(1,3,2)=1	11
6	6	{1, 2, 4, 5}	5（假设当前根为 5）	|6-5|=1	12

本页要点

只要在树中撞见 `x == tr[p].key`，说明历史出现过相同营业额，波动立刻为 0！

例 1 ｜ 核心步骤表与相对模板的变化

按顺序计算	纸上得到	核对问题
1. 读入第 i 天 x	当天营业额 x	是否有可能出现负数营业额？
2. 计算当天波动	若 $i=0$ 加 x；否则 `t.closest(x)`	第 1 天有没有误查空树？
3. 把 x 插入 Treap	`t.root = t.insert(t.root, x)`	是否接住了返回的新树根？
4. 全部结束后输出	累加结果 `total`	`total` 是否开 `long long`？

相对旋转 Treap 模板改了哪里？

- 不需要查第 k 大或删除，直接采用轻量级旋转 **Treap**，新增 `closest(x)` 沿路径取 `min` 即可。

本页要点

本题也可直接用第 1 篇学过的 STL `set` 加哨兵对比练习，体会 Treap 与 `set` 的等价性！

例 1 | 核心代码（1/3）：旋转 Treap 结构与左右旋

```
struct Treap {
    struct Node { long long key; uint64_t pri; int l = 0, r = 0; };
    vector<Node> tr{{0, 0, 0, 0}};
    mt19937_64 rng{114514}; int root = 0;
    int rot_r(int p) { int q = tr[p].l; tr[p].l = tr[q].r; tr[q].r = p; return q; }
    int rot_l(int p) { int q = tr[p].r; tr[p].r = tr[q].l; tr[q].l = p; return q; }
```

本页要点

用 tr[0] 作为空节点哨兵，用 mt19937_64 产生 64 位高质量随机优先级。

例 1 | 核心代码（2/3）：closest 最近值查询

```
    long long closest(long long x) const {
        long long best = LLONG_MAX;
        for (int p = root; p;) {
            best = min(best, llabs(tr[p].key - x)); // 沿途更新最小绝对差
            if (x == tr[p].key) return 0;           // 撞见相同值，波动立刻为 0
            p = (x < tr[p].key ? tr[p].l : tr[p].r);
        }
        return best;
    }
};
```

本页要点

与 code/treap_turnover.cpp 一致；用 llabs 对 long long 取绝对值。

例 1 | 核心代码（3/3）：insert 与主函数统计

```
int insert(int p, long long x) {
    if (!p) { tr.push_back({x, rng()}); return (int)tr.size() - 1; }
    if (x < tr[p].key) {
        tr[p].l = insert(tr[p].l, x);
        if (tr[tr[p].l].pri > tr[p].pri) p = rot_r(p);
    } else if (x > tr[p].key) {
        tr[p].r = insert(tr[p].r, x);
        if (tr[tr[p].r].pri > tr[p].pri) p = rot_l(p);
    }
    return p;
}
```

本页要点

主函数：`total += (i == 0 ? x : t.closest(x)); t.root = t.insert(t.root, x);`

例 1 | 易错、自测边界与 STL set 对比

自测场景	测试输入简述	预期输出
最小边界（$n=1$）	`1 / -100`（仅 1 天且为负数）	`-100`
全相同数	`4 / 7 7 7 7`	`7`
严格递增 $n=5$	`5 / 1 2 3 4 5`	`1+1+1+1+1 = 5`

- 常见失分点：第 1 天没特判直接查空树返回 `LLONG_MAX`；或忘记 `t.root = t.insert(...)`。
- 对比 **STL** `set`：本题只查最接近值，也可在 `set<long long>` 里放正负无穷哨兵，用 `lower_bound(x)` 和 `prev(it)` 同样 $O(\log n)$ 搞定！

本页要点

不考名次/第 k 大时，`set` 与旋转 Treap 都能 AC；但手写 Treap 帮你把旋转和树结构彻底练熟！

例 2 | [提高] 普通平衡树（一本通 1568）：　先读懂要求

大白话题意

这是一道平衡树全功能模板题。共有 $n \le 10^5$ 次操作（值域 $-10^7 \le x \le 10^7$）：

- `1 x`：插入 x；　`2 x`：删除 x（若有多个相同数，只删一个）；
- `3 x`：查 x 的最小排名（比 x 小的个数 $+1$）；　`4 x`：查排名第 x 的数；
- `5 x`：查 x 的前驱（$< x$ 的最大数）；　`6 x`：查 x 的后继（$> x$ 的最小数）。

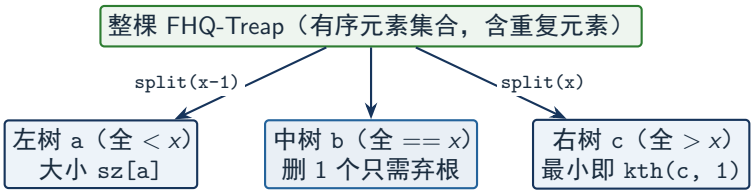
原题样例

10 次操作：`1 106465`，`4 1`，`1 317721`，`1 460929`，`1 644985`，`1 84185`，`1 89851`，`6 81968`，`1 492737`，`5 493598` $\Rightarrow$ 输出 `106465`，`84185`，`492737`。

本页要点

本题有重复元素，删除时只能删掉一个，排名定义是“严格小于 x 的数的个数 $+1$”！

例 2 | 怎么想到的与建模 TikZ 图解



- 遇到需要同时支持“删除单个重复数 + 查排名 + 查第 k 小 + 前驱后继”的全套操作时，无旋 **FHQ-Treap** 是最稳妥、最容易手写 AC 的选择！
- 所有 6 种操作都围绕上面这张“按 $x-1$ 和 x 切分值域”的示意图展开！

本页要点

只要脑子里有“$< x$、$= x$、$> x$ 三段切分图”，6 个操作的代码闭着眼睛也能写出来！

例 2 | 样例手推跟踪表

按原题样例顺序逐步追踪树中元素集合变化与查询结果：

步序	操作	操作后（或查询时）集合内部有序序列	输出
1	1 106465	{106465}	-
2	4 1	查第 1 小 ⇒ kth(root, 1)	106465
3∼7	插入 5 个数	{84185, 89851, 106465, 317721, 460929, 644985}	-
8	6 81968	按 81968 分裂：b 为全部 6 个数，查 kth(b, 1)	84185
9	1 492737	{84185, 89851, 106465, 317721, 460929, 492737, 644985}	-
10	5 493598	按 493597 分裂：a 含前 6 个数，查 kth(a, 6)	492737

本页要点

注意第 8 步和第 10 步查询的 81968 和 493598 都不在树中，依然能精确算出后继与前驱！

例 2 | 核心步骤表与相对模板的变化

按顺序计算	纸上得到	核对问题
1. 初始化哨兵	`tr(1)` 并设 `tr[0].sz = 0`	是否漏写 `tr[0].sz = 0`?
2. 处理 `op = 1, 2`	插入拼节点 / 两刀分裂删一个	删除时是否只合并了 `b` 的左右儿子?
3. 处理 `op = 3, 4`	查排名看 `tr[a].sz+1` / `kth`	查完排名有没有 `merge(a, b)`?
4. 处理 `op = 5, 6`	前驱 `split(x-1)` / 后继 `split(x)`	前驱后继查完有没有 `merge(a, b)`?

相对 FHQ 模板改了哪里?

- 完整实现了 6 种 `op` 分支；重复元素独立建点并用 `b = merge(tr[b].l, tr[b].r)` 精确只删一个。

本页要点

即使要查的数 x 本身不在树中，`split(x-1)` 照样能精准算出它的前驱和排名！

例 2 | 核心代码（1/3）：操作 1、2（插入与删除一个 x）

```
if (op == 1) {                                          // 插入 x
    int id = (int)tr.size(); tr.push_back({0, 0, 1, x, rng()});
    int a, b; split(root, x, a, b);
    root = merge(merge(a, id), b);
} else if (op == 2) {                                   // 删除一个 x
    int a, b, c; split(root, x, a, c); split(a, x - 1, a, b);
    b = merge(tr[b].l, tr[b].r);                        // 扔掉根 b，只删一个
    root = merge(merge(a, b), c);
}
```

本页要点

删除时劈两刀孤立出等于 x 的子树 b，合并其左右儿子即精确只删掉 1 个节点。

例 2 | 核心代码（2/3）：操作 3、4（排名与第 k 小）

```
else if (op == 3) {                                    // 查询 x 的最小排名
    int a, b; split(root, x - 1, a, b);
    cout << tr[a].sz + 1 << '\n';
    root = merge(a, b);
} else if (op == 4) {                                  // 查询第 x 小的数
    cout << kth(root, x) << '\n';
}
```

本页要点

注意 `op == 3` 必须按 `x - 1` 分裂，才能统计出严格小于 x 的元素个数！

例 2 | 核心代码（3/3）：操作 5、6（前驱与后继）

```
else if (op == 5) {                                    // 求 x 的前驱（< x 的最大）
    int a, b; split(root, x - 1, a, b);
    cout << kth(a, tr[a].sz) << '\n';
    root = merge(a, b);
} else {                                               // 求 x 的后继（> x 的最小）
    int a, b; split(root, x, a, b);
    cout << kth(b, 1) << '\n';
    root = merge(a, b);
}
```

本页要点

完整可编译源码位于 `code/fhq_treap_basic.cpp`，所有查询完后都别忘 `merge(a, b)`！

例 2 | 易错与 3 组自测边界

自测场景	测试输入简述	预期输出
负数与极端值	插入 -10000000，0，10000000，查 0 前驱/后继	-10000000、10000000
重复数最小排名	插入两个 5 和一个 3，查 3 5	2
删除单个重复数	再执行 2 5，查 4 2（第 2 小）	5（另一个还在）

- 哨兵大小未清零：忘记在 `main()` 开头写 `tr[0].sz = 0;`，导致全题排名错位（WA）；
- 前驱后继分裂阈值混用：前驱必须按 `x - 1` 劈，后继必须按 `x` 劈。

本页要点

自测必测“重复元素删一个”和“负数前驱后继”，两项全过说明模板稳如泰山！

平衡树 Treap 高频错误对照表

题目 / 场景	常见错误写法	正确做法与原因
旋转指针乱套	左旋右旋把 `tr[q].l` 写反	牢记右旋拎左子、中间孙子 `tr[q].r` 过继给 `tr[p].l`
忘记接住返回值	只调用 `insert(root, x)` 没写 `root =`	旋转后整棵树的根可能换成新节点，必须赋回 `root`
随机种子用 `rand()`	`rand()` 在 Windows 只有 15 位易撞车	使用 `mt19937` / `mt19937_64` 生成高质量随机优先级
哨兵 `tr[0].sz` 忘清零	`tr(1)` 默认构造 `sz=1`	`main` 开头必写 `tr[0].sz = 0;`，否则全树排名全错
FHQ 分裂完忘合并	查排名或前驱后直接离开	只要调用了 `split`，读完答案必写 `root = merge(a, b);`
空树第 1 天处理错	营业额第 1 天也去查前驱后继	题面规定第 1 天直接计原值 a_1（负值也要原样加）

本页要点

“旋转接返回值，FHQ 分裂必合并，哨兵 `tr[0].sz` 必清零”！

评测结果定位指南（WA / TLE / RE 快速排查）

评测结果	最常见诱因	30 秒定位方法
WA	忘记 `tr[0].sz = 0`；忘接 `root =`；分裂阈值 `x` 与 `x-1` 写混；忘写 `pull(u)`	造 3 个数（含重复和负数）手跑一次，打印每步 `sz` 和树根值。
TLE	用了 `rand()` 导致局部退化；未关闭 `cin/cout` 同步或用了 `endl`	换成 `mt19937`，加 `ios::sync_with_stdio(false)` 并用 `'\n'`。
RE	数组下标越界（没设 0 号空节点）；查空树第 k 大死循环	确认 `tr[0]` 存在且在 `kth` 前确保树非空（$1 \le k \le$ `sz`）。

本页要点

排名或第 k 小一错全错时，90% 是忘记 `tr[0].sz = 0;` 或 `split/merge` 漏写 `pull`！

提交前检查清单（六必查）

1. 旋转版所有递归修改树结构的函数（insert）调用处，是否都写了 `root = insert(root, x);`？
2. FHQ-Treap 是否在 `main()` 开头显式写出了 `tr[0].sz = 0;`，并在 split/merge 末尾调用 `pull`？
3. 所有查询类操作（排名、前驱、后继）在 split 读完答案后，是否都写了 `root = merge(a, b);`？
4. 遇到相同值插入与删除需求时，是否用 `split(x)` + `split(x-1)` 精准只删除了一个节点？
5. 随机数生成器是否使用了 `mt19937` / `mt19937_64` 而非老旧的 `rand()`？
6. 累加营业额总和、坐标差绝对值时，结果变量是否用了 `long long`（`llabs`）？

最后一步

故意按 $1, 2, 3, \ldots, 100000$ 严格递增插入再查询，验证不会卡成链超时后再提交！

本章全部练习与完成标准

1. **[基础]** 营业额统计（学校 OJ：1565 ｜营业额统计，配套源码 `code/treap_turnover.cpp`）：旋转 Treap 查找前驱后继最小差值。
2. **[提高]** 普通平衡树（学校 OJ：1568 ｜普通平衡树，配套源码 `code/fhq_treap_basic.cpp`）：FHQ-Treap 完整实现插入、删除、排名、第 k 小、前驱、后继六大操作。

完成标准

- 能用大白话解释随机堆优先级为何能让 BST 期望高度维持在 $O(\log n)$；
- 能独立写出旋转 Treap 与无旋 FHQ-Treap 模板，并在 10^5 递增数据下快速 AC 上述两题。

课后安排与下一篇预告

- 课后复习：在纸上画一次右旋过程，把 A，q，B，p，C 的中序位置从左到右写出来确认不变；再手推一次 split(root，5，a，b)。
- 拓展练习：把 code/treap_turnover.cpp 分别用旋转 Treap、FHQ-Treap 和 STL set 各写一遍，对比代码长度与速度。
- 下一篇预告：恭喜完成全部《04_ 数据结构》篇！下一篇正式进入《05_ 动态规划》第 1 章《线性 DP 与背包模型》——学习如何定义“状态”与“状态转移方程”，用递推解决多阶段最优决策问题！

本页要点

数据结构篇收官：从栈队列、并查集、线段树、树剖到平衡树，你已掌握竞赛全部核心容器！

本章一句话

核心总结

搜索树保左右大小序，随机堆优先级保高度平衡；旋转不变中序，分裂合并搭出万物！

谢谢大家！