高级数据结构

紊莫

2025 年 2 月 3 日

目录

可并堆（左偏树）

- 左偏树是一种可并堆，具有堆的性质，并且可以快速合并。

左偏树的定义和性质

- 对于一棵二叉树，我们定义外节点为子节点数小于两个的节点。

左偏树的定义和性质

- 对于一棵二叉树，我们定义外节点为子节点数小于两个的节点。
- 定义一个节点的 dist 为其到子树中最近的外节点所经过的边的数量。

左偏树的定义和性质

- 对于一棵二叉树，我们定义外节点为子节点数小于两个的节点。
- 定义一个节点的 dist 为其到子树中最近的外节点所经过的边的数量。
- 空节点的 dist 为 0。

左偏树的定义和性质

- 对于一棵二叉树，我们定义外节点为子节点数小于两个的节点。
- 定义一个节点的 dist 为其到子树中最近的外节点所经过的边的数量。
- 空节点的 dist 为 0。
- 左偏树是一棵二叉树，它不仅具有堆的性质，并且是**左偏**的：每个节点左儿子的 dist 都大于等于右儿子的 dist。

左偏树的定义和性质

- 对于一棵二叉树，我们定义外节点为子节点数小于两个的节点。
- 定义一个节点的 dist 为其到子树中最近的外节点所经过的边的数量。
- 空节点的 dist 为 0。
- 左偏树是一棵二叉树，它不仅具有堆的性质，并且是**左偏**的：每个节点左儿子的 dist 都大于等于右儿子的 dist。
- 因此，左偏树每个节点的 dist 都等于其右儿子的 dist 加一。

左偏树的定义和性质

- 对于一棵二叉树，我们定义外节点为子节点数小于两个的节点。
- 定义一个节点的 dist 为其到子树中最近的外节点所经过的边的数量。
- 空节点的 dist 为 0。
- 左偏树是一棵二叉树，它不仅具有堆的性质，并且是**左偏**的：每个节点左儿子的 dist 都大于等于右儿子的 dist。
- 因此，左偏树每个节点的 dist 都等于其右儿子的 dist 加一。
- 需要注意的是，dist 不是深度，左偏树的深度没有保证，一条向左的链也符合左偏树的定义。

合并操作

- 假设要合并 x, y 两个堆（以小根堆为例）。

合并操作

- 假设要合并 x, y 两个堆（以小根堆为例）。
- 首先设 $\mathrm{val}(x) < \mathrm{val}(y)$，那么将 x 作为新堆的根，其左儿子不变，递归的合并右儿子和 y 即可。

合并操作

- 假设要合并 x, y 两个堆（以小根堆为例）。
- 首先设 $\mathrm{val}(x) < \mathrm{val}(y)$，那么将 x 作为新堆的根，其左儿子不变，递归的合并右儿子和 y 即可。
- 返回上来的时候维护左偏的性质，交换两个儿子并且更新 dist 即可。

合并操作

- 假设要合并 x, y 两个堆（以小根堆为例）。
- 首先设 $\mathrm{val}(x) < \mathrm{val}(y)$，那么将 x 作为新堆的根，其左儿子不变，递归的合并右儿子和 y 即可。
- 返回上来的时候维护左偏的性质，交换两个儿子并且更新 dist 即可。
- 这样的复杂度是 $\mathcal{O}(\log n)$ 的。

复杂度证明

证明.

根据性质：左偏树每个节点的 dist 都等于其右儿子的 dist 加一。
每次递归都会使得 dist 减一，而根的 dist 是 $O(\log n)$ 的。
一棵根的 dist $= x$ 的二叉树至少有 $x-1$ 层是满二叉树，那么就至少有 $2^x - 1$ 个节点。
所以 n 个点的树 dist 是 $O(\log n)$ 级别的。 □

随机交换法

- 你可以采用每次以 $\frac{1}{2}$ 的概率交换两个儿子的方式，就不用维护 dist 了。

随机交换法

- 你可以采用每次以 $\frac{1}{2}$ 的概率交换两个儿子的方式，就不用维护 dist 了。
- 复杂度为平均 $\mathcal{O}(\log n)$，实测常数差距不大。

随机交换法

- 你可以采用每次以 $\frac{1}{2}$ 的概率交换两个儿子的方式，就不用维护 dist 了。
- 复杂度为平均 $\mathcal{O}(\log n)$，实测常数差距不大。
- 想看证明的点我。

常规操作

- 加入点：将原堆和一个大小为 1 的可并堆合并。

常规操作

- 加入点：将原堆和一个大小为 1 的可并堆合并。
- 删除最小值：合并根的左右儿子。

常规操作

- 加入点：将原堆和一个大小为 1 的可并堆合并。
- 删除最小值：合并根的左右儿子。
- 像一些常见数据结构一样打打 tag，做点整体操作。

P1456 Monkey King

- 有 n 个集合，一开始只有一个元素 s_i。
- 有 m 次操作，每次将两个集合内元素最大值减半，然后合并这两个集合。
- 输出每次操作完后最终集合内的最大值。

代码

```cpp
int n, m, fa[N], s[N], ls[N], rs[N], dist[N];
int find(int x) { return fa[x] == x ? x : fa[x] = find(fa[x]); }
int merge(int x, int y) {
    if (!x || !y) return x + y;
    if (s[x] < s[y]) swap(x, y); rs[x] = merge(rs[x], y);
    if (dist[ls[x]] < dist[rs[x]]) swap(ls[x], rs[x]);
    dist[x] = dist[rs[x]] + 1; return x;
}
int upd(int x) {
    s[x] /= 2; int rt = merge(ls[x], rs[x]);
    ls[x] = rs[x] = 0; return fa[x] = fa[rt] = merge(rt, x);
}
```

P3642 [APIO2016] 烟火表演

被要求加入的一个题。

- 给出一棵大小为 n 的树，边有边权，要求改变每一条边的边权，将边权从 x 改为 y 的代价为 $|x-y|$。
- 使得根节点到每个叶子的距离相同，求最小代价。

要求复杂度 $\mathcal{O}(n\log n)$。

解法

首先设出朴素的 DP，$f_{u,i}$ 表示 u 到 u 子树内的叶子节点路径长度均为 i 的最小代价。

转移有 $f_{u,i} = \sum_{v \in \text{son}_u} \min_{j=0}^{i}(|w-j| + f_{v,i-j})$，不妨考虑逐个儿子合并。

因为绝对值函数是下凸的，运用归纳法容易得到 f_i 也是下凸的。

然后可以用上 Slope Trick 经典的维护方法，维护这个下凸函数的拐点。

可持久化可并堆优化 K 短路问题

- 给出一张 n 个点，m 条边的有向图，带正权边。
- 求 s 到 t 的第 k 短路径。

要求复杂度 $O(n \log n)$（假设 n, k, m 同阶）。

如何刻画这个问题

- 这背后实际上是一类贪心问题，即每次精确的在现有的解空间中找到最优解拓展并不重不漏。

如何刻画这个问题

- 这背后实际上是一类贪心问题，即每次精确的在现有的解空间中找到最优解拓展并不重不漏。
- 放在这个题就是每次找没被找过的路径中最短那个，不重不漏就好了。

解法

- 建立到终点的最短路树，记 dis_i 表示 i 到终点的距离。

解法

- 建立到终点的最短路树，记 dis_i 表示 i 到终点的距离。
- 用一个非树边集合来表示当前的决策。

解法

- 建立到终点的最短路树，记 dis_i 表示 i 到终点的距离。
- 用一个非树边集合来表示当前的决策。
- 对于一组 $e_1, e_2, \ldots, e_m$ 的意思是说，从起点开始，先到 e_1 的起点，走过 e_1 后沿着树边走到 e_2 的起点，依次类推，最后走到终点。

解法

- 建立到终点的最短路树，记 dis_i 表示 i 到终点的距离。
- 用一个非树边集合来表示当前的决策。
- 对于一组 $e_1, e_2, \ldots, e_m$ 的意思是说，从起点开始，先到 e_1 的起点，走过 e_1 后沿着树边走到 e_2 的起点，依次类推，最后走到终点。
- 这个解的后继要么是修改 e_m 的终点，要么是加入一条边 e_{m+1}，按这个意思解法自然是不重不漏的。

解法

- 建立到终点的最短路树，记 dis_i 表示 i 到终点的距离。
- 用一个非树边集合来表示当前的决策。
- 对于一组 $e_1, e_2, \ldots, e_m$ 的意思是说，从起点开始，先到 e_1 的起点，走过 e_1 后沿着树边走到 e_2 的起点，依次类推，最后走到终点。
- 这个解的后继要么是修改 e_m 的终点，要么是加入一条边 e_{m+1}，按这个意思解法自然是不重不漏的。
- 定义 $e = (u, v, w)$ 的权值为 $\Delta e = dis_v + w - dis_u$。

解法

- 建立到终点的最短路树，记 dis_i 表示 i 到终点的距离。
- 用一个非树边集合来表示当前的决策。
- 对于一组 $e_1, e_2, \ldots, e_m$ 的意思是说，从起点开始，先到 e_1 的起点，走过 e_1 后沿着树边走到 e_2 的起点，依次类推，最后走到终点。
- 这个解的后继要么是修改 e_m 的终点，要么是加入一条边 e_{m+1}，按这个意思解法自然是不重不漏的。
- 定义 $e = (u, v, w)$ 的权值为 $\Delta e = dis_v + w - dis_u$。
- 一个非树边集合的路径长度可以表示为 $dis_s + \sum \Delta e$。

解法

- 看来我们只要能快速维护好一个非树边集合的后继就好了，而且非树边集合只需要记录最后一条边。

解法

- 看来我们只要能快速维护好一个非树边集合的后继就好了，而且非树边集合只需要记录最后一条边。
- 对于修改终点，那么起点是固定的，直接维护后继。

解法

- 看来我们只要能快速维护好一个非树边集合的后继就好了，而且非树边集合只需要记录最后一条边。
- 对于修改终点，那么起点是固定的，直接维护后继。
- 对于加入一条边，那么能加的边有什么要求呢？

解法

- 看来我们只要能快速维护好一个非树边集合的后继就好了，而且非树边集合只需要记录最后一条边。
- 对于修改终点，那么起点是固定的，直接维护后继。
- 对于加入一条边，那么能加的边有什么要求呢？
- 只能是起点是 v_m 的祖先的边。

解法

- 看来我们只要能快速维护好一个非树边集合的后继就好了，而且非树边集合只需要记录最后一条边。
- 对于修改终点，那么起点是固定的，直接维护后继。
- 对于加入一条边，那么能加的边有什么要求呢？
- 只能是起点是 v_m 的祖先的边。
- 那么我们要找到以一个点及其祖先为起点的边中的最小值，使用可持久化可并堆即可。

可并堆的可持久化

- 和其他的可持久化数据结构一样，每次访问到的点新开一个拷贝即可。

目录

bitset

我们直接看 OI-wiki。

几个注意点

- 动态开 bitset 或者其他操作可以直接手写，本质上就是压位和位运算。
- bitset 使用 cout 输出是从高位到低位。

目录

K-D Tree

- K-D Tree 是一种可以高效处理 k 维空间信息的数据结构。

K-D Tree

- K-D Tree 是一种可以高效处理 k 维空间信息的数据结构。
- 在算法竞赛的题目中，一般有 $k = 2$，下文默认解决的是二维平面的问题。

K-D Tree

- K-D Tree 是一种可以高效处理 k 维空间信息的数据结构。
- 在算法竞赛的题目中，一般有 $k = 2$，下文默认解决的是二维平面的问题。
- KDT 的每个节点都维护了一个矩形，表示子树中的点都在这个矩形范围内，其中用作分割的点也要记录下来，下文均采用这种 Nodey 的表示方式。

构建

- `build(l, r)` 表示对 $[l, r]$ 内的点构建一个 KDT。

构建

- `build(l, r)` 表示对 $[l, r]$ 内的点构建一个 KDT。
- 每次划分平面相当于是选择一个点，注意几个优化：

构建

- `build(l, r)` 表示对 $[l, r]$ 内的点构建一个 KDT。
- 每次划分平面相当于是选择一个点，注意几个优化：

  1. 轮流选择 k 个维度（或选择方差最大的维度）。

构建

- `build(l, r)` 表示对 $[l, r]$ 内的点构建一个 KDT。
- 每次划分平面相当于是选择一个点，注意几个优化：
  1. 轮流选择 k 个维度（或选择方差最大的维度）。
  2. 每次选择按这个维度排序的中位数。

构建

- `build(l, r)` 表示对 $[l, r]$ 内的点构建一个 KDT。
- 每次划分平面相当于是选择一个点，注意几个优化：
  1. 轮流选择 k 个维度（或选择方差最大的维度）。
  2. 每次选择按这个维度排序的中位数。
- 实现利用 `nth_element` 即可做到复杂度 $\mathcal{O}(n \log n)$，同时也满足了树高为 $\mathcal{O}(\log n)$ 级别。

插入和删除

- 类似二叉查找树地向下搜索，找到点为空即可插入。

插入和删除

- 类似二叉查找树地向下搜索，找到点为空即可插入。
- 删除可以采用懒惰删除。

插入和删除

- 类似二叉查找树地向下搜索，找到点为空即可插入。
- 删除可以采用懒惰删除。
- 此处如果暴力操作会导致复杂度退化，需要对 KDT 进行重构，常见方法有：

插入和删除

- 类似二叉查找树地向下搜索，找到点为空即可插入。
- 删除可以采用懒惰删除。
- 此处如果暴力操作会导致复杂度退化，需要对 KDT 进行重构，常见方法有：

  1. 替罪羊式，即 $\mathrm{size}(u)\alpha < \max(\mathrm{size}(\mathrm{lson}), \mathrm{size}(\mathrm{rson}))$ 的时候暴力重构子树，这个 α 实际通常取 0.75 左右，**但是这种方法只能保证树高是** $O(\log n)$，在后面的复杂度分析需要严格 $\log n + O(1)$。

插入和删除

- 类似二叉查找树地向下搜索，找到点为空即可插入。
- 删除可以采用懒惰删除。
- 此处如果暴力操作会导致复杂度退化，需要对 KDT 进行重构，常见方法有：

  1. 替罪羊式，即 $\mathrm{size}(u)\alpha < \max(\mathrm{size}(\mathrm{lson}), \mathrm{size}(\mathrm{rson}))$ 的时候暴力重构子树，这个 α 实际通常取 0.75 左右，**但是这种方法只能保证树高是** $O(\log n)$，在后面的复杂度分析需要严格 $\log n + O(1)$。

  2. 根号重构式，即每 B 次操作进行一次重构，删除的话就是子树内删除点数大于 B 的时候重构子树。

- 类似二叉查找树地向下搜索，找到点为空即可插入。
- 删除可以采用懒惰删除。
- 此处如果暴力操作会导致复杂度退化，需要对 KDT 进行重构，常见方法有：

1. 替罪羊式，即 $\mathrm{size}(u)\alpha < \max(\mathrm{size}(\mathrm{lson}), \mathrm{size}(\mathrm{rson}))$ 的时候暴力重构子树，这个 α 实际通常取 0.75 左右，**但是这种方法只能保证树高是** $O(\log n)$，在后面的复杂度分析需要严格 $\log n + O(1)$。
2. 根号重构式，即每 B 次操作进行一次重构，删除的话就是子树内删除点数大于 B 的时候重构子树。
3. 二进制分组，维护多个 KDT，每次新建一个大小为 1 的 KDT，不断合并大小相同的树，查询则分别查询。

矩形查询

- 若当前矩形和查询矩形无交或查询矩形包含当前矩形，则统计信息并返回。

矩形查询

- 若当前矩形和查询矩形无交或查询矩形包含当前矩形，则统计信息并返回。
- 否则向左右递归查询。

矩形查询

- 若当前矩形和查询矩形无交或查询矩形包含当前矩形，则统计信息并返回。
- 否则向左右递归查询。
- 如果有矩形修改之类的也是一样，打上标记即可，可参考『2024.11.8 NOIP 模拟赛 T4』。

矩形查询

- 若当前矩形和查询矩形无交或查询矩形包含当前矩形，则统计信息并返回。
- 否则向左右递归查询。
- 如果有矩形修改之类的也是一样，打上标记即可，可参考『2024.11.8 NOIP 模拟赛 T4』。
- 时间复杂度是 $\mathcal{O}(\sqrt{n})$ 的，下面进行分析。

矩形查询复杂度分析

- 设查询矩形为 R，树中的矩形要么和 R 无交，要么被 R 完全包含，要么和 R 有交。
- 显然复杂度只和遍历到的第三类点的个数有关，分为两种情况：完全包含 R、互不包含。
- 根据树高的性质，第一种情况只有 $O(\log n)$ 个点，下面主要分析第二种情况。
- 此处放缩一下，考虑 R 的每条边经过的矩形个数。
- 一个点到其四个孙子，经过了横向和纵向的划分，也就是这条边至多经过两个孙子节点。
- $T(n) = 2T\left(\frac{n}{4}\right) + O(1)$，得到时间复杂度 $O(\sqrt{n})$。
- 拓展到 k 维的时间复杂度是 $O(n^{1-\frac{1}{k}})$。

应用

大概分成矩形查询和邻域查询，以及一些数据结构能干的事情。

P4848 崂山白花蛇草水

- 每次插入点 (x, y, w)，查询矩形内 w 的 k 大值。
- 要求复杂度 $O(n\sqrt{n}\log V)$。

P4848 崂山白花蛇草水

- 采用树套 KDT 的方式，外层套上你喜欢的以值域为下标的数据结构，这里以线段树为例。

P4848 崂山白花蛇草水

- 采用树套 KDT 的方式，外层套上你喜欢的以值域为下标的数据结构，这里以线段树为例。
- 插入点，即在线段树上 $\log n$ 个节点所表示的 KDT 上插入一个点。

P4848 崂山白花蛇草水

- 采用树套 KDT 的方式，外层套上你喜欢的以值域为下标的数据结构，这里以线段树为例。
- 插入点，即在线段树上 $\log n$ 个节点所表示的 KDT 上插入一个点。
- 查询就是线段树上二分，KDT 支持查询矩形内点的个数即可。

P4848 崂山白花蛇草水

- 采用树套 KDT 的方式，外层套上你喜欢的以值域为下标的数据结构，这里以线段树为例。
- 插入点，即在线段树上 $\log n$ 个节点所表示的 KDT 上插入一个点。
- 查询就是线段树上二分，KDT 支持查询矩形内点的个数即可。
- 时间复杂度：$\mathcal{O}(n\sqrt{n}\log V)$。

P4631 [APIO2018] 选圆圈

- 给出 n 个平面上的圆，每次从未删除的圆中选择半径最大的（如有同，选择编号最小的）圆 i，并删除和它相交或相切的圆 j，称圆 j 被圆 i 删除，求每个圆被哪个圆删除。
- 数据范围：$n \leq 3 \times 10^5$。

P4631 [APIO2018] 选圆圈

- 两个圆相交的必要条件是其外接正方形相交。

P4631 [APIO2018] 选圆圈

- 两个圆相交的必要条件是其外接正方形相交。
- 以此建立 KDT，每次查询时剪枝。

P4631 [APIO2018] 选圆圈

- 两个圆相交的必要条件是其外接正方形相交。
- 以此建立 KDT，每次查询时剪枝。
- 注意，KDT 在解决此类问题是复杂度是错误的，可能退化到 n^2，但是实际情况往往比较优秀，不失为一种好方法。

P4631 [APIO2018] 选圆圈

- 两个圆相交的必要条件是其外接正方形相交。
- 以此建立 KDT，每次查询时剪枝。
- 注意，KDT 在解决此类问题是复杂度是错误的，可能退化到 n^2，但是实际情况往往比较优秀，不失为一种好方法。
- 其余邻域查询问题也可类比，设计一个估价函数，来表示出尽可能精确的必要条件即可。

- 每个点有一个坐标 (x_i, y_i)，有若干次连边，每次形如一个点向一个矩形内的所有点连一条边权为 w_i 的边。
- 求从 1 开始的单源最短路。
- 数据范围：$1 \le n \le 70000$,$1 \le m \le 150000$。

P5471 [NOI2019] 弹跳

- 显然的优化建图题，说明 KDT 其实也存在很多常见数据结构的衍生用法。

P5471 [NOI2019] 弹跳

- 显然的优化建图题，说明 KDT 其实也存在很多常见数据结构的衍生用法。
- 一个点会向 $\mathcal{O}(\sqrt{n})$ 个 KDT 上的节点连边，KDT 上每个点连向其左右儿子和该点对应的点。

- 显然的优化建图题，说明 KDT 其实也存在很多常见数据结构的衍生用法。
- 一个点会向 $\mathcal{O}(\sqrt{n})$ 个 KDT 上的节点连边，KDT 上每个点连向其左右儿子和该点对应的点。
- 不过你不能真的把边连出来，而是要用的时候去找一下就好了。

P9598 [JOI Open 2018] 山体滑坡

- n 个点，多次删边和加边，问断掉一端编号小于等于 p_i，另一端编号大于 p_i 的边后形成的联通块数。
- 注：本题做法繁多，如果是单纯做题，不建议思考 KDT 做法，但是在此处请先思考 KDT。
- KDT 做法要求复杂度为 $O(n\sqrt{n}\log n)$。

P9598 [JOI Open 2018] 山体滑坡

- ~~本题做法有操作分块，KDT 分治，LCT + 线段树分治等，不建议写 KDT。~~

P9598 [JOI Open 2018] 山体滑坡

- ~~本题做法有操作分块，KDT 分治，LCT + 线段树分治等，不建议写 KDT。~~
- 下文会提到 LCT 做法，复杂度更优。

P9598 [JOI Open 2018] 山体滑坡

- ~~本题做法有操作分块，KDT 分治，LCT + 线段树分治等，不建议写 KDT。~~
- 下文会提到 LCT 做法，复杂度更优。
- 发现 $[1, p_i]$ 和 $(p_i, n]$ 两个点集的答案是没影响的，分别求一下。

- ~~本题做法有操作分块，KDT 分治，LCT + 线段树分治等，不建议写 KDT。~~
- 下文会提到 LCT 做法，复杂度更优。
- 发现 $[1, p_i]$ 和 $(p_i, n]$ 两个点集的答案是没影响的，分别求一下。
- 那么现在是一个二维的问题，一个边贡献到的时间和 p 都是一个区间，仿照线段树分治，写一个 KDT 分治即可。

P9598 [JOI Open 2018] 山体滑坡

- ~~本题做法有操作分块，KDT 分治，LCT + 线段树分治等，不建议写 KDT。~~
- 下文会提到 LCT 做法，复杂度更优。
- 发现 $[1, p_i]$ 和 $(p_i, n]$ 两个点集的答案是没影响的，分别求一下。
- 那么现在是一个二维的问题，一个边贡献到的时间和 p 都是一个区间，仿照线段树分治，写一个 KDT 分治即可。
- 时间复杂度是 $O(n\sqrt{n}\log n)$ 的，居然跑得还可以。

目录

Link Cut Tree

- Link Cut Tree 是一种数据结构，我们用它来解决动态树问题。

Link Cut Tree

- Link Cut Tree 是一种数据结构，我们用它来解决动态树问题。
- Link Cut Tree 简称 LCT，但它不叫动态树，动态树是指一类问题。

Link Cut Tree

- Link Cut Tree 是一种数据结构，我们用它来解决动态树问题。
- Link Cut Tree 简称 LCT，但它不叫动态树，动态树是指一类问题。
- Splay Tree 是 LCT 的基础，但是 LCT 用的 Splay Tree 和普通的 Splay 在细节处不太一样。

- 常见的解决树链信息的方法是重链剖分，通过重链的性质来得到优秀的复杂度。

Link Cut Tree

- 常见的解决树链信息的方法是重链剖分，通过重链的性质来得到优秀的复杂度。
- 但是遇到树的形态不固定的情况，常见为加边，删边，换根等，这种剖分方法就显得捉襟见肘了。

Link Cut Tree

- 常见的解决树链信息的方法是重链剖分，通过重链的性质来得到优秀的复杂度。
- 但是遇到树的形态不固定的情况，常见为加边，删边，换根等，这种剖分方法就显得捉襟见肘了。
- LCT 采用一种更为**自由**的剖分方式——虚实链剖分，对于每个点，选择至多一个实儿子，其余为虚儿子，相应的可以定义虚边，实边，实链等，并采用 Splay Tree 来维护这些实链。

辅助树

- LCT 在对树的形态以及修改时，不会修改原树，而是对于一棵“辅助树”修改。

辅助树

- LCT 在对树的形态以及修改时，不会修改原树，而是对于一棵“辅助树”修改。
- 你可以认为，一棵 Splay 对应一条实链，它们构成了一个辅助树，而若干辅助树则构成了一个辅助森林。

辅助树

- LCT 在对树的形态以及修改时，不会修改原树，而是对于一棵“辅助树”修改。
- 你可以认为，一棵 Splay 对应一条实链，它们构成了一个辅助树，而若干辅助树则构成了一个辅助森林。
- 这里的 Splay 用原树中的**深度**作为排序关键字。

辅助树性质

1. 辅助树由多棵 Splay 组成，每棵 Splay 维护原树中的一条路径，且中序遍历这棵 Splay 得到的点序列，从前到后对应原树「从上到下」的一条路径。
2. 原树每个节点与辅助树的 Splay 节点一一对应。
3. 每个 Splay 的根节点连向其原树上的父亲，但是不作为父亲的儿子出现，也即父不认子。

由于辅助树的以上性质，我们维护任何操作都不需要维护原树，辅助树可以在任何情况下拿出一个唯一的原树，我们只需要维护辅助树即可，图示可见 OI-wiki。

Splay 部分

1. `Get(x)` 获取 x 是父亲的哪个儿子。
2. `Splay(x)` 通过和 Rotate 操作联动实现把 x 旋转到当前 Splay 的根。
3. `Rotate(x)` 将 x 向上旋转一层的操作。

LCT 部分

1. `access(x)` 打通一条包含且仅包含根到 x 的一条实链，为 LCT 的核心操作。
2. `notroot(x)` 判断 x 是否**不是**所在树的根。
3. `makeroot(x)` 使 x 点成为其所在树的根。
4. `link(x, y)` 在 x, y 两点间连一条边。
5. `cut(x, y)` 把 x, y 两点间边删掉。
6. `findroot(x)` 找到 x 所在树的根节点编号。
7. `split(x, y)` 提取出 x, y 间的路径，方便做区间操作。

access 操作

- 从 x 开始，自下而上的生成所要的实链。

access 操作

- 从 x 开始，自下而上的生成所要的实链。
- 将这个实链的 Splay 根节点记为 y。

access 操作

- 从 x 开始，自下而上的生成所要的实链。
- 将这个实链的 Splay 根节点记为 y。
- 每次操作，先将 x 转到所在 Splay 的根，然后将其右儿子更改为 y。

- 从 x 开始，自下而上的生成所要的实链。
- 将这个实链的 Splay 根节点记为 y。
- 每次操作，先将 x 转到所在 Splay 的根，然后将其右儿子更改为 y。
- 我们按照深度作为关键字，那么深度小于等于 x 的点显然就是 x 不断跳父亲，大于 x 的点我们已经得到了 y 这个树，那么直接改掉就好了，此处并没有修改原来 x 的右儿子，也就是符合了上面提到的父不认子，相当于把那条边变成了虚边。

access 操作

- 从 x 开始，自下而上的生成所要的实链。
- 将这个实链的 Splay 根节点记为 y。
- 每次操作，先将 x 转到所在 Splay 的根，然后将其右儿子更改为 y。
- 我们按照深度作为关键字，那么深度小于等于 x 的点显然就是 x 不断跳父亲，大于 x 的点我们已经得到了 y 这个树，那么直接改掉就好了，此处并没有修改原来 x 的右儿子，也就是符合了上面提到的父不认子，相当于把那条边变成了虚边。
- 然后将 $y \leftarrow x$，$x \leftarrow \text{father}(x)$，如此到根即可。

notroot 操作

要用父亲是否存在这个儿子来判断。

makeroot 操作

此处需要引入 Splay 的 reverse 操作，即像文艺平衡树一样下放标记。

- 首先我们打通一条根到 x 的路径，然后将其 Splay 到根。

makeroot 操作

此处需要引入 Splay 的 reverse 操作，即像文艺平衡树一样下放标记。

- 首先我们打通一条根到 x 的路径，然后将其 Splay 到根。
- 此时发现其一定没有右儿子（深度为关键字），而根一定没有左儿子，那么直接 reverse 这个 Splay 就好了。

findroot 操作

- 类似 makeroot，先 access(x) + splay(x)。
- 根一定是深度最小的，不断跳左儿子即可。

注意：返回之前要 splay(root)，否则会导致复杂度错误。

link 操作

- 假设 x, y 不连通，连接他们的边为虚边。

link 操作

- 假设 x, y 不连通，连接他们的边为虚边。
- 直接 makeroot(x)，然后将 father(x) = y 即可，父不认子。

split 操作

- 用于取出 (x, y) 这个路径，做法是 makeroot(x) + access(y) + splay(y)。

split 操作

- 用于取出 (x, y) 这个路径，做法是 makeroot(x) + access(y) + splay(y)。
- 此时我们得到的一个以 y 为根的，大小为路径上点数的 Splay。

split 操作

- 用于取出 (x, y) 这个路径，做法是 makeroot(x) + access(y) + splay(y)。
- 此时我们得到的一个以 y 为根的，大小为路径上点数的 Splay。
- 在 cut 操作时 y 的左儿子为 x。

cut 操作

- 同样假设存在边 (x, y)。

cut 操作

- 同样假设存在边 (x, y)。
- 那么首先 split(x, y)，取出这个边，然后直接修改父子信息即可，注意要同时修改父亲和自己，否则可能只是实边变成虚边而已。

P3690 【模板】动态树（LCT）

给定 n 个点以及每个点的权值 a_i，要你处理接下来的 m 个操作。
操作有四种：

- 询问从 x 到 y 的路径上的点的权值的 xor 和。保证 x 到 y 是联通的。
- 连接 x 到 y，若 x 到 y 已经联通则无需连接。
- 删除边 (x, y)，不保证边 (x, y) 存在。
- 将点 x 上的权值变成 y。

数据范围：$1 \le n \le 10^5$，$1 \le m \le 3 \times 10^5$，$1 \le a_i \le 10^9$。

P3690 【模板】动态树（LCT）

此处应有一段代码。

CF1109F

- 给一个 $n \times m$ 的网格图，每个格子上有个权值 f_{ij}，保证 f_{ij} 构成一个 $1 \sim nm$ 的排列。
- 问有多少区间 l, r 满足 $1 \leq l \leq r \leq nm$ 且权值在 l, r 内的格子构成的连通块是一棵树。
- 数据范围：$n, m \leq 1000, nm \leq 2 \times 10^5$。

CF1109F

- 首先，可以双指针求出一个 l 对应的极大的 r，使得 $[l,i],(i\in[l,r])$ 不含环。

CF1109F

- 首先，可以双指针求出一个 l 对应的极大的 r，使得 $[l,i],(i\in[l,r])$ 不含环。
- 这一步可以利用 LCT 求解。

CF1109F

- 首先，可以双指针求出一个 l 对应的极大的 r，使得 $[l,i],(i\in[l,r])$ 不含环。
- 这一步可以利用 LCT 求解。
- 但是要判断森林就没有这么好的性质了，因为随着 i 变化，图的形态可能在树和森林间变化。

CF1109F

- 首先，可以双指针求出一个 l 对应的极大的 r，使得 $[l, i], (i \in [l, r])$ 不含环。
- 这一步可以利用 LCT 求解。
- 但是要判断森林就没有这么好的性质了，因为随着 i 变化，图的形态可能在树和森林间变化。
- 于是运用树的点数减去边数等于 1 的性质，同时维护一个线段树来计算即可，查询 $[l, r]$ 内的 1 的个数可以变成查询最小值的个数。

利用 LCT 的均摊

- 有时候题目中的操作可能和 LCT 不谋而合，那么可以在 LCT 的基础上魔改一下。

利用 LCT 的均摊

- 有时候题目中的操作可能和 LCT 不谋而合，那么可以在 LCT 的基础上魔改一下。
- 这样的话可以借助 LCT 的势能分析来做到比较优秀的复杂度。

先来证明一下 LCT 的复杂度

前面 Splay 部分省略了。

- LCT 的核心操作是 access，那么只需要证明 access 操作的复杂度。
- 重虚边：从节点 v 到其父节点的虚边，其中 $size(v) > \frac{1}{2} size(parent(v))$。
- 轻虚边：从节点 v 到其父节点的虚边，其中 $size(v) \leq \frac{1}{2} size(parent(v))$。
- 定义势能是重虚边的条数。
- 一次 access 至多走 $O(\log n)$ 条轻虚边，至多带来 $O(\log n)$ 条重虚边，而每次访问重虚边会以 $O(1)$ 的代价减少 1 的势能，可以忽略。
- 最终把初始势能加上势能变化量，得到复杂度为 $O(m \log n)$，m 为操作次数。

P3703 [SDOI2017] 树点涂色

- 给出一棵 n 个点的有根树，其中 1 号点是根节点，点 i 的初始颜色为 i。
- 定义一条路径的权值是：这条路径上的点（包括起点和终点）共有多少种不同的颜色。
- 你需要维护如下几种操作，对于第 i 次操作：

1. 把点 x 到根节点的路径上所有的点染上颜色 $i+n$。
2. 求 x 到 y 的路径的权值。
3. 在以 x 为根的子树中选择一个点，使得这个点到根节点的路径权值最大，求最大权值。

数据范围：$1 \leq n, m \leq 10^5$。

P3703 [SDOI2017] 树点涂色

- 题目给出的奇怪染色竟然就是 LCT 中的 access 操作！
- 这样的话一个点的答案就是到根的路径上虚边的个数。
- 对于路径查询，直接差分，对于子树查询，额外维护一个线段树即可。

时间复杂度是 $\mathcal{O}(n\log^2 n)$ 的。

CF1344E

- 给定 n 个点的一棵树，以 1 为根，边有边权。
- 有 m 辆从 1 开始到 s_i 的火车，第 i 辆火车在 t_i 时刻时从根出发，向着目标 s_i 前进，当火车在 x 时刻到达一个点 u 时，假设下一个路径上的点是 v，两点间边权是 d，则在 $x+d$ 时刻到达 v，火车在到达目标点后停止。
- 由于每个点可能可以到达多个点，每个点有且仅有一个当前时刻可以到的儿子，每秒钟只能切换某一个点可以到的儿子，切换比火车开动先进行，若一辆火车走向了非目标方向的点，则立刻自爆。每个点初始能到的儿子是确定的。
- 求出能否不发生自爆，如果不能，第一次自爆最晚什么时候发生？在此基础上，至少切换多少次一个点可以到的儿子。

数据范围：$n, m \le 10^5$。

CF1344E

- 按照时间操作，当遇到一个点需要修改的时候，设上一次经过该点的时间为 x，当前时间为 t，那么这个点就需要在 $(x, t]$ 内被操作一次。

CF1344E

- 按照时间操作，当遇到一个点需要修改的时候，设上一次经过该点的时间为 x，当前时间为 t，那么这个点就需要在 $(x,t]$ 内被操作一次。
- 把这样的区间按照左端点排序，每次操作能操作中的右端点最小的区间，就是最优策略。

CF1344E

- 按照时间操作，当遇到一个点需要修改的时候，设上一次经过该点的时间为 x，当前时间为 t，那么这个点就需要在 $(x, t]$ 内被操作一次。
- 把这样的区间按照左端点排序，每次操作能操作中的右端点最小的区间，就是最优策略。
- 那么这样的区间只会在 LCT access 虚实边切换的时候出现，只有 $O(m \log n)$ 个，直接来就好了。

LCT 可以支持动态的维护树（如最小生成树），和图结构。
在维护的时候，往往只能支持加边，不能删边，否则会变得不可名状。

- 给出 n 个点，m 条边，每条边可以表示为一个四元组 (u, v, a, b)，其中 u, v 是边，a, b 是权值，要求一条从 1 到 n 的路径，使得经过的边 $\max a + \max b$ 最小。
- 数据范围：$n \leq 5 \times 10^4$，$m \leq 10^5$。

[NOI2014] 魔法森林

- LCT 维护最小生成树的板子，不过只能加边，不能删边。

[NOI2014] 魔法森林

- LCT 维护最小生成树的板子，不过只能加边，不能删边。
- 有两个权值，不妨按照 a 从小到大排序，那么对 a 扫描线，维护当前可用的边中 1 到 n 的边权最大值最小即可，那么这一定是最小生成树上的路径。

[NOI2014] 魔法森林

- LCT 维护最小生成树的板子，不过只能加边，不能删边。
- 有两个权值，不妨按照 a 从小到大排序，那么对 a 扫描线，维护当前可用的边中 1 到 n 的边权最大值最小即可，那么这一定是最小生成树上的路径。
- 这里注意 LCT 没有像重剖一样边化点的方法，需要给每个边开一个虚点，link 的时候若两者已经在同一个连通块内，那么必然有环形成，cut 掉权值最大的边就可以。

[NOI2014] 魔法森林

- LCT 维护最小生成树的板子，不过只能加边，不能删边。
- 有两个权值，不妨按照 a 从小到大排序，那么对 a 扫描线，维护当前可用的边中 1 到 n 的边权最大值最小即可，那么这一定是最小生成树上的路径。
- 这里注意 LCT 没有像重剖一样边化点的方法，需要给每个边开一个虚点，link 的时候若两者已经在同一个连通块内，那么必然有环形成，cut 掉权值最大的边就可以。
- 这种东西可能会结合线段树分治使用。

P9598 [JOI Open 2018] 山体滑坡

现在我们给出 $O(n\log^2 n)$ 的 LCT 解法。

- 首先线段树分治，然后记边 (u,v) 的边权为 $\max(u,v)$，用 LCT 维护最小生成树。

P9598 [JOI Open 2018] 山体滑坡

现在我们给出 $O(n\log^2 n)$ 的 LCT 解法。

- 首先线段树分治，然后记边 (u,v) 的边权为 $\max(u,v)$，用 LCT 维护最小生成树。
- 对于一个询问 $[1,P_i]$ 查询 $\max(u,v)\le P_i$ 的边的个数，用一个树状数组辅助统计即可。

现在我们给出 $O(n\log^2 n)$ 的 LCT 解法。

- 首先线段树分治，然后记边 (u,v) 的边权为 $\max(u,v)$，用 LCT 维护最小生成树。
- 对于一个询问 $[1,P_i]$ 查询 $\max(u,v)\le P_i$ 的边的个数，用一个树状数组辅助统计即可。
- 对于 $(P_i,n]$ 的询问是对称的。

P5489 EntropyIncreaser 与动态图

- 有一个 n 个点的图，初始没有边。
- 有 q 个操作，分为 3 种，具体如下：
- 1 u v 表示在 u, v 之间连一条无向边。
- 2 u v 表示求 u, v 间的割边数量。
- 3 u v 表示求 u, v 间的割点数量。

强制在线，要求复杂度 $O(n \log n)$。

LCT 维护割边

- 首先维护原图的一个生成树，对于一次 link(u, v) 来说，若 u, v 未连通，则直接相连（建立虚点）。

LCT 维护割边

- 首先维护原图的一个生成树，对于一次 link(u, v) 来说，若 u, v 未连通，则直接相连（建立虚点）。
- 否则将这一条链上的权值赋值为 0 表示其再也不可能作为割边出现了，打上标记即可。

LCT 维护割点

- 静态维护割点的方式是圆方树，不妨用 LCT 动态的维护圆方树。

LCT 维护割点

- 静态维护割点的方式是圆方树，不妨用 LCT 动态的维护圆方树。
- 同样考虑 link(u, v) 时 u, v 已经连通的情况，那么直接暴力将环上的边断开，连向一个新的方点即可，此处的结构可能有点怪，会出现方点连向方点的情况，但是不影响答案和复杂度。

LCT 维护割点

- 静态维护割点的方式是圆方树，不妨用 LCT 动态的维护圆方树。
- 同样考虑 link(u, v) 时 u, v 已经连通的情况，那么直接暴力将环上的边断开，连向一个新的方点即可，此处的结构可能有点怪，会出现方点连向方点的情况，但是不影响答案和复杂度。
- 复杂度证明考虑每次是以 $O(L\log n)$ 的代价将长度为 L 的环删掉，那么就是 $O(n\log n)$ 的。

LCT 维护子树信息

LCT 通过对虚子树和子树信息的维护，可以实现一定程度上的子树信息维护。
下面通过一道例题来介绍。

Query on a tree VI

- 给出一个 n 个点的树，每个点为黑色或白色，初始均为黑色，要求支持 m 次操作。
- 每次询问有多少节点到 u 的路径上点颜色相同，或者翻转一个点的颜色。

数据范围：$1 \leq n, m \leq 10^5$。

解法

- 常见的想法是把同色的连通块合并起来，查询的是连通块大小，但是这样的复杂度是 $O(d \log n)$ 的，d 是度数。
- 这里用到一个技巧，以白色为例，每个白色点向父节点连边，黑色点则不连，那么最后会形成一个森林，真实的连通块是这些森林去掉每个树的根后得到的新森林。
- 然后在每个点维护子树大小和虚子树大小，在 link 和 access 的时候修改即可。

加以扩展即可解决 CF1172E / P5526。

LCT 维护一些序列问题

我做到的这些题基本上是在 P3203 弹飞绵羊的基础上扩展。
所以也只能介绍这一点。

P3203 [HNOI2010] 弹飞绵羊

- 游戏一开始，Lostmonkey 在地上沿着一条直线摆上 n 个装置，每个装置设定初始弹力系数 k_i，当绵羊达到第 i 个装置时，它会往后弹 k_i 步，达到第 $i+k_i$ 个装置，若不存在第 $i+k_i$ 个装置，则绵羊被弹飞。
- 绵羊想知道当它从第 i 个装置起步时，被弹几次后会被弹飞。
- 为了使得游戏更有趣，Lostmonkey 可以修改某个弹力装置的弹力系数，任何时候弹力系数均为正整数。

P3203 [HNOI2010] 弹飞绵羊

- 将 i 和 $i + k_i$ 连边，每次加边删边，查询深度。

- 将 i 和 $i+k_i$ 连边，每次加边删边，查询深度。
- 这种根节点确定的情况，LCT 的 link 函数对应的可以写的简单一点，不必要 makeroot，实测常数变小很多。

P3203 [HNOI2010] 弹飞绵羊

- 将 i 和 $i+k_i$ 连边，每次加边删边，查询深度。
- 这种根节点确定的情况，LCT 的 link 函数对应的可以写的简单一点，不必要 makeroot，实测常数变小很多。
- 相似题可以尝试 CF1039E。

Good Luck and Have Fun.