信息学竞赛
树与数据结构

主讲：李宁远

树的直径的定义

树上任意两节点之间最长的简单路径即为树的直径。——OI-Wiki

其实在实际使用里，有时「直径」一词也指最长路径的长度。

有的时候也可以用来表达在树上的某个点集内，任意两节点之间最长的简单路径。

树的直径的 dfs 求法

首先从任意节点 x 开始进行第一次 dfs，到达距离其最远的节点，记为 y，然后再从 y 开始做第二次 dfs，到达距离 y 最远的节点，记为 z，则 y 和 z 之间的路径即为树的直径。
证明可以看 OI-Wiki。

树的直径的性质

若树上所有边边权均为正，则树的所有直径中点重合。
可以仿照上面的证明，分类讨论中点所在的位置，容易证明。
例题：https://yundouxueyuan.com/p/734

树的重心的定义

如果在树中选择某个节点并删除，这棵树将分为若干棵子树，统计子树节点数并记录最大值。取遍树上所有节点，使此最大值取到最小的节点被称为整个树的重心。
有时点带有点权，此时的带权重心一般是指统计子树节点点权之和，并记录最大值。

树的重心的性质

树的重心如果不唯一，则至多有两个，且这两个重心相邻。
以树的重心为根时，所有子树的大小都不超过整棵树大小的一半。
树中所有点到某个点的距离和中，到重心的距离和是最小的；如果有两个重心，那么到它们的距离和一样。
把两棵树通过一条边相连得到一棵新的树，那么新的树的重心在连接原来两棵树的重心的路径上。
在一棵树上添加或删除一个叶子，那么它的重心最多只移动一条边的距离。

树的重心的求法

在 DFS 中计算每个子树的大小，记录「向下」的子树的最大大小，利用总点数 - 当前子树（这里的子树指有根树的子树）的大小得到「向上」的子树的大小，然后就可以依据定义找到重心了。
参考代码 - OI-Wiki
例题：CF685B

定义

最近公共祖先简称 LCA（Lowest Common Ancestor）。两个节点的最近公共祖先，就是这两个点的公共祖先里面，离根最远的那个。一个点集的 LCA，就是它们所有点的公共祖先里面，离根最远的那个。

性质

u 是 v 的祖先，当且仅当 $\mathrm{LCA}(u,v)=u$；
如果 u 不为 v 的祖先并且 v 不为 u 的祖先，那么 u,v 分别处于 $\mathrm{LCA}(u,v)$ 的两棵不同子树中；
两点集并的最近公共祖先为两点集分别的最近公共祖先的最近公共祖先，即 $\mathrm{LCA}(A\cup B)=\mathrm{LCA}(\mathrm{LCA}(A),\mathrm{LCA}(B))$；
两点的最近公共祖先必定处在树上两点间的最短路上；
$d(u,v)=h(u)+h(v)-2h(\mathrm{LCA}(u,v))$，其中 d 是树上两点间的距离，h 代表某点到树根的距离。

倍增法求 LCA

预处理出 $fa_{x,i}$，$\mathrm{fa}_{x,i}$ 表示点 x 的第 2^i 个祖先。$\mathrm{fa}_{x,i}$ 数组可以通过 dfs 预处理出来。

现在要求 $\mathrm{LCA}(u,v)$，可以不断让它们向上跳，直到两个点一样。先将深的跳到同一高度，再共同跳跃。

现在我们看看如何优化这些跳转：在调整游标的第一阶段中，我们要将 u,v 两点跳转到同一深度。我们可以计算出 u,v 两点的深度之差，设其为 y。通过将 y 进行二进制拆分，我们将 y 次游标跳转优化为「y 的二进制表示所含 1 的个数」次游标跳转。在第二阶段中，我们从最大的 i 开始循环尝试，一直尝试到 0（包括 0），如果 $\mathrm{fa}_{u,i}\neq\mathrm{fa}_{v,i}$，则 $u\leftarrow\mathrm{fa}_{u,i},v\leftarrow\mathrm{fa}_{v,i}$，那么最后的 LCA 为 $\mathrm{fa}_{u,0}$。

模板：P3379

树上差分

树上差分可以理解为对树上的某一段路径进行差分操作，这里的路径可以类比一维数组的区间进行理解。例如在对树上的一些路径进行频繁操作，并且询问某条边或者某个点在经过操作后的值的时候，就可以运用树上差分思想了。树上差分又分为点差分与边差分，在实现上会稍有不同。

点差分

举例：对树上的一些路径进行访问，问每个点被访问的次数。
对于一次 $\delta(s,t)$ 的访问，这里进行差分操作：

$$d_s \leftarrow d_s + 1$$
$$d_{lca} \leftarrow d_{\mathtt{lca}} - 1$$
$$d_t \leftarrow d_t + 1$$
$$d_{f(\mathtt{lca})} \leftarrow d_{f(\mathtt{lca})} - 1$$

其中 $f(x)$ 表示 x 的父亲节点，d_i 为点权 a_i 的差分数组。
例题：P3128

边差分

若是对路径中的边进行访问，就需要采用边差分策略了，使用以下公式：

$$d_s \leftarrow d_s + 1$$
$$d_t \leftarrow d_t + 1$$
$$d_{lca} \leftarrow d_{lca} - 2$$

由于在边上直接进行差分比较困难，所以将本来应当累加到红色边上的值向下移动到附近的点里，那么操作起来也就方便了。对于公式，有了点差分的理解基础后也不难推导，同样是对两段区间进行差分。

树上启发式合并

在树上处理子树问题时常用，如果想得到子树的一些信息，可以尝试将非最大的子树的孩子们全都合并到最大的孩子。这个合并指的是对于一些信息的合并，比如子树颜色集合，诸如此类的。如果合并两个集合所用的时间是较小的集合大小（或者乘上一些 log），那么总的复杂度就是它再乘一个 log。

比如要维护子树的颜色数，这个信息就可以是一个 set。每次将小的 set 的每个元素取出来插入大的 set。

例题：P9233

证明

考虑每个点的贡献。只有当这个点的某个祖先被当成小孩子被合并了，它才会贡献一点复杂度。如果被当成小孩子，说明其父节点至少大于其两倍，所以最多只有 $O(\log n)$ 次被当成小孩子。

倍增思想

对序列、树等，仅考虑关键区间 $[x, x+2^k-1]$ 的信息。在树上，就是向上的 2^k 长的信息。好处有：

- 若可以快速合并两个区间的信息，那么可以快速计算得到所有关键区间的信息。从小到大枚举 k 即可实现。
- 对于可重复信息（如 $\min, \max$ 等），可以用两个区间合并出任意区间的信息。
- 对任意区间，可以用 $O(\log n)$ 段关键区间拼出该区间的信息。
- 可以替代二分查找：从大到小枚举 k，维护一个当前满足条件的位置 p，每次判断 $p+2^k$ 是否满足条件。如果判断时需要用信息，那么可以通过 p 位置信息快速得到 $p+2^k$ 位置信息。

ST 表

要求维护的是可重复的静态信息。
OI-Wiki

线段树

OI-Wiki

树状数组

OI-Wiki
重要的是其结构。

例题

冰火战士

解法

观察到答案为冰火人分别的能量和的最小值，而且这两个函数分别随温度升高而上升/下降，于是可以二分位置，判定冰前缀和与火后缀和之大小关系，直至左侧火大，右侧冰大。
此为 $O(n\log^2 n)$，不得行。
优化：观察树状数组结构，考虑顺应结构做倍增，以此达到快速计算前缀和的办法。

平衡树

OI-Wiki

Treap

Treap 维护环

例：圆圈舞

分块

OI-Wiki

例题

Vines

维护一个长为 n 的序列 $a_{0,1,\ldots,n-1}$，最初都为 0。有 q 次操作，每次操作有三种：

1. 给定 i，令 $a_i = 1$；
2. 给定 k，对所有 $i \bmod c \equiv k$ 的 i，令 $a_i \leftarrow a_i + 1$；其中 c 是一个初始时给定的常数；
3. 给定 i，不断令 $i \leftarrow \min(n, i + a_i)$（认为 $a_n = 0$），求最后会停在哪里。

$1 \le n, q \le 10^5$，$1 \le c \le n$。

解法

如果 $c = n$，那么就是单点加，可以考虑一个分块算法 (LCT 也可以，不过相当复杂而且常数巨大，所有不考虑)：令块长为 B，那么每 B 个元素作为一块，维护它们向后跳到块外的位置（或者就在块内停下），这样每次修改就暴力重构该块，查询可以快速跳跃，复杂度 $O(n + q(B + \frac{n}{B}))$。对于一般的 c，首先可以快速维护 a_i。当某次修改时 $a_i + i$ 已经跳出块时，就不必再重构这个块了，否则暴力重构。这样，对于所有点，重构次数最多是 $O(B(n+q))$ 的（初始时每个点 B 次，操作 1 会使每个点多 B 次），于是总复杂度就是 $O(\frac{nq}{B} + B^2(n+q))$，平衡一下就是 $O((n+q)^{5/3})$ 的。

矩阵维护信息

如果要维护的信息是可以用矩阵乘法或广义矩阵乘法来维护的，那么可以考虑用矩阵维护信息。

资料 1

资料 2

在这里我只讲述较简单的维护方式。更复杂、更数学的维护方式（例如历史和）请参考上面的资料，我更擅长感性理解它的结构。

~~我之前花了一下午也没能理解最普通的历史和怎么做的~~

例题

数据传输

解法

-

复杂度均摊

OI-Wiki

例题

冰山

解法

使用 map 暴力模拟，并且打全局 tag 表示 map 内所有数的加减。

考虑令势能函数为 map 内元素数量，也就是冰山的体积种类数。

- 当体积减小时，如果删除了 c 个冰山，那么只需要 $O(c \log n)$ 的时间，此时势能函数减少 c。
- 当体积增加时，如果分裂的冰山个数为 c，那么只需要 $O(c \log n)$ 的时间，此时势能函数至少减小 $c-2$。
- 当新增冰山时，势能函数增加 1，操作复杂度为 $O(\log n)$。

考虑把一份势能函数对应成 $\log n$，那么均摊下来，每次操作的复杂度为 $O(\log n)$。

Thank You