树链剖分与 LCT

NiroBC

Aug, 2021

1 树链剖分

在此之前，我们已经熟悉了使用线段树，对一个序列进行修改与询问的做法。如果把这样的问题放到树上会如何呢？来看下面这道例题：

BZOJ 1036 ZJOI 2008 树的统计 Count

给定一棵 n 个节点的树，节点 i 的权值为 w_i.

有 q 次操作，每次操作为如下类型之一：

- `CHANGE`: 给定 u,t，将节点 u 的权值修改为 t;
- `QSUM`: 给定 u,v，请输出 u 到 v 的路径上所有节点权值的和；
- `QMAX`: 给定 u,v，请输出 u 到 v 的路径上所有节点权值的最大值。

数据范围：$1 \le n \le 3 \times 10^4; 1 \le q \le 2 \times 10^5$

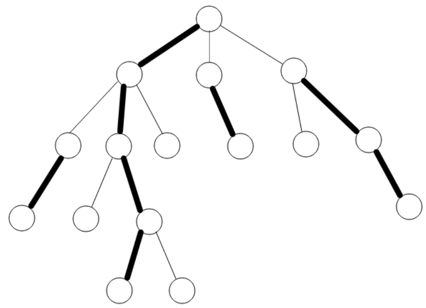
如果这个问题发生在序列上（即，这棵树是一条链），那么，我们早已能够熟练地使用线段树解决它。而当它是一棵树时，就需要通过一些手段，将它变成序列上的问题。

1.1 轻重链剖分

在一棵树上，对于每一个节点 u，我们定义 $size(u)$ 为点 u 的子树内（包括 u 自己）的节点数量。

对于一个非叶节点 u，我们将它孩子中 $size$ 最大的那个点定义为 u 的重孩子（如有并列第一，则任选一个），其余的孩子都是 u 的轻孩子。

连接点 u 和重孩子的边称作重边（下图中用较粗的线表示），连接点 u 和轻孩子的边称作轻边（下图中用较细的线表示）。



我们将所有由重边组成的路径称作重链，每一个节点恰好属于一条重链，整颗树就被划分成了若干条重链的集合。

这样划分能有什么好处？假如 v 是 u 的轻孩子，就必有 $size(u) \geq 2size(v)$. 这样，对于任意一个节点 u，它到根节点的路径上最多只有 $\lfloor \log_2 \left(\frac{n}{size(u)} \right) \rfloor$ 条轻边。

也就是说，任意一条点 u 到点 v 的路径，都可以划分成最多 $O(\log n)$ 段，每段都是重链上的一段区间。

这样，所有对于树上路径的操作或询问，就转化成了对于 $O(\log n)$ 个序列上的区间的操作或询问。对于每一个区间，每次操作或询问的复杂度是 $O(\log n)$，所以，对于一条树上路径的一次操作或询问，时间复杂度为 $O(\log^2 n)$.

2 LCT(Link Cut Tree)

2.1 LCT 基础模板题

维护一个 n 个点的森林，点有权值，进行 q 次以下操作：

1. 给定 u, v，保证 u, v 不联通，在 u, v 间连一条边。
2. 给定 u, v，保证 u, v 间有一条直接的边，将这条边删掉。
3. 给定 u, v，判断 u, v 是否联通，若联通，请输出 u 到 v 的路径上的点权之和。
4. 给定 u, v, w，保证 u, v 相联通，将 u 到 v 的路径上的所有点的权值增加 w。

数据范围：$1 \leq n, q \leq 10^5$

做法

树链剖分仅支持在树的**结构**不发生变化时维护各种修改，但是现在需要支持删边和加边的操作，并同时支持链上的修改与询问。

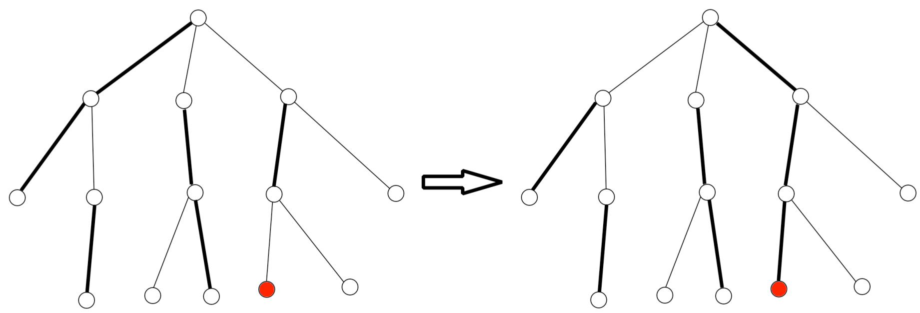
给森林中的每一棵树任意指定一个根。树上的每个点，可以有一个重孩子（不一定是 $size$ 最大的点），也可以没有重孩子（即使它不是叶子）。这样，整棵树被划分成了若干条重链。

由于树的形态会发生变化，链与链之间可能会发生切断、连接，所以，每一条重链我们都用 splay 树来维护，splay 上中序遍历从左往右，就是原树上同一条重链上深度从浅到深。

对于每一条重链，我们再记下该重链的链顶在树上的的父亲，这是为了记录轻边。

接下来，我们来介绍 Link Cut Tree 的一些操作：

Access



Access 操作，就是给定一个点 p，将点 p 到根的所有边都变成重边。如果 p 的某个祖先原来的重孩子不是现在的重孩子，就把原来的重边变成轻边。

对于 p 本身，操作之后 p 将没有重孩子，与孩子间的所有边都变成轻边。

设 x 表示 p 到根路径上原来的轻边数量，就需要进行 $O(x)$ 次 splay 的拆分、合并操作。这样，就能把到根路径上的 $x+1$ 段重链拼起来。

Make root　森林中的每棵树的根都是任意指定的。Make root 操作，就是将某个点 p 变成 p 所在联通块的根。

过程很简单，先执行 `access(p)`，这样 p 到**原来的**根 r 就形成的一条从上到下的重链。然后，在这条重链所在的 splay 树的根部打一个翻转标记，这样的话，原来 p 在下，r 在上，现在 p 在上，r 在下，p 成为了新的根。

Link　在 u, v 间连边。

```
void link(int u, int v)
{
    make_root(u);
    让 u 成为 v 的轻孩子;
}
```

这样就轻松地实现了加边操作。

Cut　将 u, v 间的边删掉。

```
void cut(int u, int v)
{
    make_root(u);
    access(v);
    此时 u 和 v 所在的重链上只有 u 和 v 两个点，断掉即可;
}
```

Find root 找到 p 所在的联通块的根。

```
int find_root(int p)
{
    access(p);
    return p 所在的重链 splay 树上中序遍历最靠左的节点;
}
```

Check connected 询问 u, v 是否在同一个联通块。
这只要看 `find_root(u)` 和 `find_root(v)` 是否相等即可。

Query, Change 询问 u 到 v 的路径上的一些信息，例如路径点权和。
或是对 u 到 v 的整条路径实施修改。

```
int query(int u, int v)
{
    make_root(u);
    access(v);
    return v 所在的 splay 树根部上存储的信息;
}
int change(int u, int v)
{
    make_root(u);
    access(v);
    在 v 所在的 splay 树根部打上整体修改的 tag;
}
```

时间复杂度分析

我们发现，所有的时间复杂度的不确定性都来源于 Access 操作。

设一个点的 size 为其子树中的点数，定义势能函数 Φ 为 $size$ **最大的儿子不是重儿子的非叶节点数量**。

在 `access(p)` 时，设 x 为 p 到根的路上轻边的条数，这样的一次 `access(p)` 就会让势能函数最多增加 $2\log_2(n) - x$.

这样的话，所有 `access(p)` 操作的 x 之和不会超过 $O(q\log n)$.

所以总时间复杂度不超过 $O(q\log^2 n)$.

但是不仅仅如此哦。在 splay 树上，将一个点 p splay 到根的均摊时间复杂度为 $O(\log(size(root)) - \log(size(p)))$. 在不断将 p 和上面的重链相接，不断往上 splay 的过程中，总的复杂度不是 $O(x\log n)$，而是 $O(\log n - \log(size(p)) + x)$.

所以可证明总复杂度不超过 $O(q\log n)$.

2.2 NOI2014 魔法森林

在一张 n 个点，m 条边的无向图上，每条边有两个权值 a_i, b_i，问所有从点 1 到点 n 的路径中，所有边的 $\max\{a_i\} + \max\{b_i\}$ 的最小值是多少。

数据范围：$1 \le n \le 5 \times 10^4, 1 \le m \le 10^5$

做法

考虑枚举 $A=\max\{a_i\}$，在 A 固定的情况下，计算，在只使用 $a_i \le A$ 的边的情况下，从 1 到 N 的路径中 $\max\{b_i\}$ 最小的路径是多少。

所以，这时的答案就是只保留所有 $a_i \le A$ 的边，以 b_i 的最小生成树上 1 到 N 的路径上的 $\max\{b_i\}$。

那么，我们可以把所有的边按 a_i 从小到大排序，将边一条一条加入当前的最小生成树，动态维护最小生成树。

每加入一条边时，需要执行的操作如下：

1. 加入 u 到 v 的第二维权值为 b 的边时，若原先 u 到 v 不联通，就加边。否则如果 u 到 v 的路径上的 b_i 的最大值 $> b$，就把那条 b 最大的边删掉之后再加入 u 到 v 的边，否则就不管 u 到 v 的边。
2. 询问 1 到 N 的路径上的 b_i 的最大值。

2.3 BZOJ 3514 Codechef MARCH14 GERALD07 加强版

有一张 n 个点 m 条边的无向图，第 i 条边连接的点是 u_i 和 v_i.

q 次询问，每次给定 $l, r (1 \le l \le r \le m)$，问在只保留第 $l, l+1, \cdots, r$ 条边的情况下，这张图有多少个联通块。

询问强制在线，必须逐个处理每个询问。

数据范围：$1 \le n, m, q \le 2 \times 10^5$

做法

对于一个询问 $[l, r]$，答案就等于，n 减去：在只保留序号 $\le r$ 的边的情况下，边号的**最大**生成树上，有多少条边号 $\ge l$ 的边。

所以，我们可以按边号从小到大加入每一条边，动态维护最大生成树。在已加入所有边号 $\le r$ 的边的情况下，对于固定的 r，用一棵线段树维护对所有 l 的答案。

由于询问强制在线，你需要把上面这个线段树可持久化。

2.4 LCT 的子树询问

维护一个 n 个点的森林，点有权值，进行 q 次以下操作：

1. 给定 u, v，保证 u, v 不联通，在 u, v 间连一条边。
2. 给定 u, v，保证 u, v 间有一条直接的边，将这条边删掉。
3. 给定 u, v，判断 u, v 是否联通，若联通，请输出 u 到 v 的路径上的点权之和。
4. 给定 u, v, w，保证 u, v 相联通，将 u 到 v 的路径上的所有点的权值增加 w。
5. 给定 p 与 p 所在联通块的根节点，求出 p 的子树内所有点的点权和。

数据范围：$1 \le n, q \le 10^5$

在模板题的基础上增加了子树询问。

做法

对于每一个节点 u，额外记录 $sum_{light}(u)$，表示 u **的所有轻孩子子树内所有节点权值之和**。

这样，对于任意一个节点 u，它的子树内所有节点权值之和，就等于，u 所在的重链上 u 下方（包括 u）所有点的权值之和加上 sum_{light} 之和。

当 u 因 `access` 操作更换重孩子时，只有 $sum_{light}(u)$ 会发生变化，我们只要从 $sum_{light}(u)$ 中减去一棵子树，再加上一棵子树即可。

当 `make_root` 操作翻转一条重链时，没有任何一个点的 sum_{light} 会发生变化。

这样，所有的 sum_{light} 值都可以在 LCT 中维护，我们就实现了子树询问。

不过，这个方法不能用来维护子树内最大/最小值，因为，求最大/最小值操作不存在减法，我们没有办法从一个集合中减去一棵子树。