数据维护

——Peanut Tang

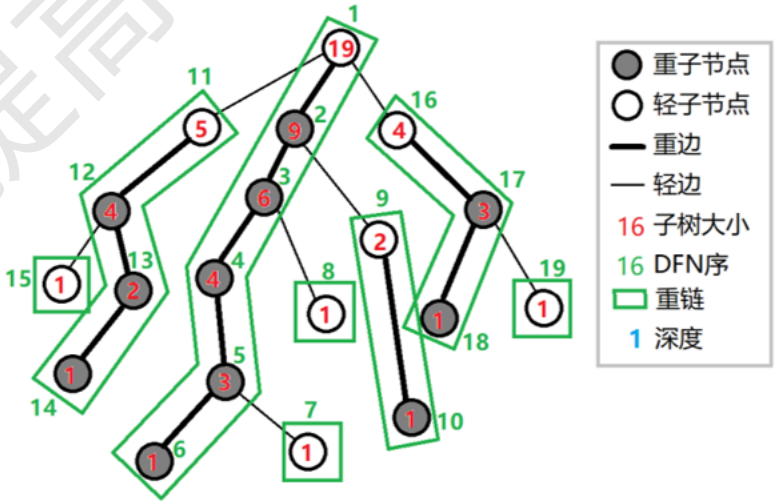
黄盛唐

福建师范大学附属中学⇒中国科学技术大学

重链剖分

概念

- 重儿子：对于一个点 u 的孩子节点，将其中子树大小最大的称为 u 的重儿子，若有多个子树大小最大的，任选一个。
- 轻儿子：不是重儿子的其他孩子节点。
- 重边：一个点与其重儿子的连边。
- 轻边：剩下的边。
- 重链：若干条首尾衔接的重边构成重链（能连的都要连）。
- 我们把落单的点也看成重链，那么整棵树就被剖分成若干条重链，如右图。



轻边个数的级别

- 说了这么多概念有啥用呢？

我们来证明一个惊人的事实，任意节点往上的轻边数量是 $O(\log n)$ 级别的。

因为每往上走过一条轻边，子树大小翻一倍（若不是的话，因为占了超过一半，则该节点必是重儿子）。

那 $O(\log n)$ 这个答案就非常显然了。

加上其他数据结构

这个结论确实很惊人，但是有什么用呢？

我们可以在 dfs 优先遍历重儿子，这样一条重链的 dfs 序就是连续的。

那如果现在有链加、链查这样的操作，如果两端点都在同一条重链中，因为 dfs 序连续，所以可以转化为序列上的操作，用线段树即可。

那如果不在同一条重链中呢？实际上一个简单路径可以被分割为若干条重链（最多三段不完整），把每一条都查出来加起来即可。

因为轻边个数级别，所以分割的重链数量是 $O(\log n)$ 级别的，那么总复杂就是 $O(\log^2 n)$ 的。

推广一下，只要查的东西可加（单向可加也可以），那就可以这样实现链查。

求最近公共祖先

- 其实和上一页的链加操作几乎一致。
  让所在重链链顶深度更深的点跳到所在重链的链顶，当两个点在同一重链时，LCA 就呼之欲出了。
  常数比倍增小得多，空间复杂度更是优秀的线性。

后话

- 重剖虽然套线段树虽然是 $O(\log^2 n)$ 的，但挺多时候都比 $O(\log n)$ 的 LCT 表现得好。
- 与重链剖分类似，还有长链剖分，可以尝试证明任意节点往上的短边个数是 $O(\sqrt{n})$ 级别的。
- 还有实链剖分，LCT 就是基于这个的。
- ~~有一种常见的骗分方法——随机剖分。~~

启发式合并

算法介绍

这是一个十分经典的符合人类直觉的、看似暴力的、实则复杂度正确的算法，又或者思想。
总归就是一句话，大的保留，小的往大的合并。
这样就可以保证每次大小都会至少乘二，复杂度就降为了 $O(\log n)$ 级别。
最常见的应用就是在并查集的按秩合并。

树上启发式合并

例题引入

给一棵 n 个节点以编号为 1 的节点为根的有根树，每个节点有一个颜色 c_i。
现在对于每个结点询问其子树里一共出现了多少种不同的颜色。
$1 \leq n \leq 10^6$。

大致要求小常数 $O(n\log n)$ 的复杂度。

算法介绍

- 直接对每个点暴力 dfs 其子树内的所有点是 $O(n^2)$ 的复杂度的。
但其实仔细思考一下你发现其实有若干浪费，有一个子树的桶存储相关信息可以直接 $O(1)$ 继承到上面。
那肯定是把重儿子的信息继承上来最优秀，这也符合之前启发式合并的套路。
但好像还不是很对？直观上还是 $O(n^2)$ 的。
但实际上这是正确的，只要精细实现，便可以做到 $O(n\log n)$ 的复杂度。

复杂度证明

- 方法基本和证明重链剖分复杂度一致。

我们考虑有哪些情况一个点 x 会在计算结点 u 的答案时被遍历到。
可以发现当且仅当 x 不在 u 的重儿子的子树中，或者是 $u = x$。
而对于每个 x，这样的 u 的个数等于其往上的轻边数量加一。
前面重链剖分我们已经分析出轻边数量是 $O(\log n)$ 级别的。
所以时间复杂度就是 $O(n\log n)$。

后话

- 树上启发式合并又被称作 DSU on tree。

$O(1)$ 直接继承重儿子的所有信息，剩下轻儿子的信息暴力加入。

C++11 以上交换容器是 $O(1)$ 的，实在不放心可以额外再开一个数组实现下标交换。

是离线算法，强制在线就无法使用。

不要乱用 memset，一个一个删除。

暴力插入的复杂度要和插入元素个数有关，不然可能复杂度有问题。

线段树合并

算法介绍

- 对于两颗普通的堆式存储线段树，它们都是完全二叉树，合并两颗线段树的信息可以直接暴力遍历每个节点然后合并。
但如果是动态开点线段树，节点通常是不满的，这个时候再用上面的算法就显得十分拙劣。
这时候就是线段树合并这个算法派上用处的时候。
直接上代码更加直观：
可以较为直观的看出这个算法的复杂度是 O(两个线段树重叠部分) 的。

```
int merge(int u, int v, int l, int r)
{
    if (!u || !v)
        return u | v;
    int mid = l + r >> 1;
    if (l == r)
    {
        // merge u && v
        return u;
    }
    pushdown(u);
    pushdown(v);
    ls[u] = merge(ls[u], ls[v], l, mid);
    rs[u] = merge(rs[u], rs[v], mid + 1, r);
    pushup(u);
    return u;
}
```

例题

和树上启发式合并一样的例题。

给一棵 n 个节点以编号为 1 的节点为根的有根树，每个节点有一个颜色 c_i。

现在对于每个结点询问其子树里一共出现了多少种不同的颜色。

$1 \leq n \leq 10^6$。

解题思路

- 桶换成用线段树存，然后直接用线段树合并即可。
但是复杂度不是那么直观。

复杂度证明

我们证明一个很强的结论，将 n 个只有 1 个元素的线段树合并成 1 个有 n 个元素的线段树的时间复杂度是 $O(n\log n)$ 的。

考虑算法流程，每次 dfs 一个节点，显然我们可以 dfs 到这个节点就证明这两个位置都有节点。

发现合并之后属于线段树 2 的节点就不会被 dfs 到了（因为线段树1的节点替换了它的位置），可能被 dfs 到的属于线段树 2 的节点是被接到线段树 1 的那一部分。

但是那部分节点根本就没被 dfs 到，所以每个节点至多被 dfs 一次。

而一次 dfs 的复杂度是线段树深度即 $O(\log n)$ 级别的，所以总的复杂度就是 $O(n\log n)$。

树上启发式合并能解决的问题，基本都可以用线段树合并在同时间复杂度下解决，但是常数更大，空间开销也更大。

另一种实现

线段树合并还有另一种实现方式，代码如下图：这样不会破坏原来两颗线段树的信息，但是空间常数更大。

```
int merge(int u, int v, int l, int r)
{
    if (!u || !v)
        return u | v;
    int p = ++tot, mid = l + r >> 1;
    if (l == r)
    {
        // p <- merge u && v
        return u;
    }
    pushdown(u);
    pushdown(v);
    ls[p] = merge(ls[u], ls[v], l, mid);
    rs[p] = merge(rs[u], rs[v], mid + 1, r);
    pushup(p);
    return p;
}
```

可持久化数据结构

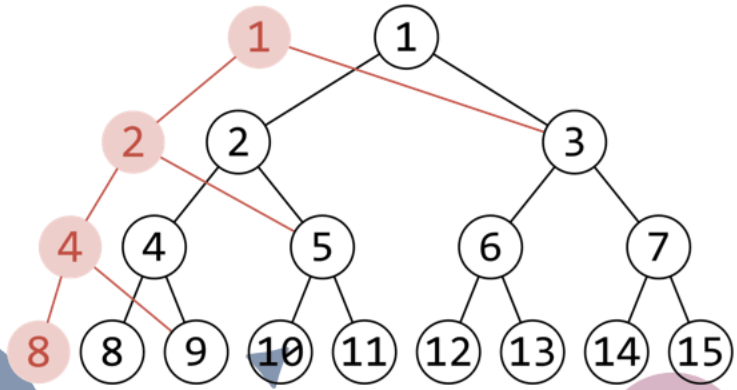
可持久化线段树

最常用的可持久化数据结构是可持久化线段树，基本上其他的可持久化数据结构都基于它，故这里着重也只介绍可持久化线段树。

当我们从原线段树 T 上执行了一个修改操作时，我们可以先让 $T' = T$，再对 T' 修改。这样实现了可持久化，但空间是平方量级的。

实际上我们发现这有很多浪费，因为线段树上的修改操作，不管是单点还是区间，都只会改变 $O(\log n)$ 个节点的信息，剩下的都不变。

像右图那样，并运用动态开点，即可做到 $O(m\log n)$ 的空间复杂度，其中 m 是修改次数。



关于标记下传

如果只有单点修改，没有标记，那显然一次修改改变 $O(\log n)$ 个点。
但如果有打标记，需要下传（pushdown），显然下传后还要开新节点，那这个结论还成立吗？
还是成立的，但这里不打算证明，你只需要知道常数会更大。
但如果只是像区间加这样的操作，我们可以考虑标记永久化，这样就不需要下传了。

函数式线段树

对于一个序列 $\{a_n\}$，我们想要查询区间信息。

如果这个信息和下标在同一个维度中，我们或许可以使用线段树等来维护。

但如果不在同一维度中，可能就要用一些二维数据结构来维护了。

但如果没有修改操作，我们可以考虑使用可持久化线段树来维护。

具体的，我们让第 i 棵线段树存储 $a_1, a_2, \cdots, a_i$ 这个前缀的信息。

按照 $1 \sim n$ 的顺序，第 i 棵先继承第 $i-1$ 棵的信息，再把 a_i 的信息插入，这样空间复杂度就是 $O(n\log n)$。

如果这个信息满足可减性，我们可以在第 $l-1$ 和第 r 棵线段树上分别查，然后相减即可得到 $[l,r]$ 的信息。

题目选讲

luogu P4755 Beautiful Pair

给一个长度为 n 的数列 $\{a_n\}$，求下集合的大小：

$$\{(i,j)|a_i \cdot a_j \le \max_{i \le k \le j} a_k, 1 \le i \le j \le n\}$$

$1 \le n \le 10^5$，$1 \le a_i \le 10^9$。

解析

分治是套路的。和区间最大值相关，于是考虑建出笛卡尔树，在笛卡尔树上分治。

当前在考虑区间 $[l,r]$，考虑用 ST 表找出这个区间最大值的位置 I。于是在笛卡尔树上走，相当于再去考虑区间 $[l,I-1]$ 与 $[I+1,r]$。

枚举 $[l,I]$ 的每一项 a_x，则这一项的贡献就是 $[I,r]$ 中大于等于 $\lfloor a_I/a_x \rfloor$ 的项的个数。

这可以用可持久化线段树维护，使用函数式线段树即可。

根据启发式的套路，选更短的区间去枚举计算贡献。

时间复杂度为 $O(n\log^2 n)$，空间复杂度为 $O(n\log n)$。

十二省联考 2019 春节十二响

给一棵 n 个点的树，每个点有一个权值 w_i。
你每次可以选取树上的一个点集，要求点集中的每个点不能是另一个点的祖先，而选出点集的代价为点集中权值最大点的权值。问将所有点都选一遍的最小代价为多少，每次选的点集不能包含之前已经被选过的点。
$1 \leq n \leq 200000$，$1 \leq w_i \leq 10^9$。

提示：若树是一条链，怎么做。

解析

若 1 号点只有一个儿子，则答案显然为所有 w_i 的和；若有两个儿子，那肯定贪心地选左右儿子子树中最大的两个配对，次大的两个配对，……，剩下没配对的直接加到答案。

推广到一般的树上，每个点开一个堆。对于每个节点，它的堆就是它的所有孩子像上面那样两两合并，最后再加入自己的权值。

直接合并是 $O(n^2)$ 的，但是用启发式合并的套路，每次直接继承大的，剩下的小的加进来，复杂度就正确了。

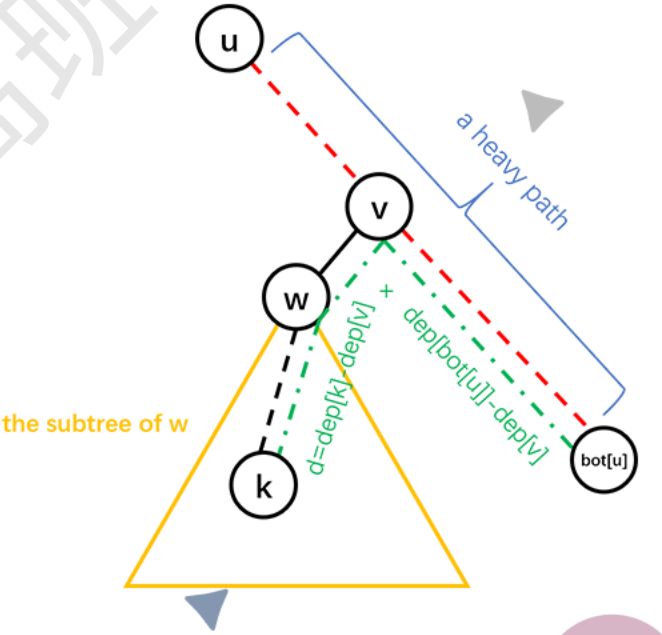
时间复杂度为 $O(n\log^2 n)$，空间复杂度为 $O(n)$。

loj P6669 Nauuo and Binary Tree

有一棵 n 个点的以编号为 1 的点为根的有根二叉树，但是你不知道它长什么样。
你可以进行若干次询问，每次询问任意两点之间的距离。
你需要还原出这颗树。
$1 \leq n \leq 3000$ ，你最多可以进行 30000 次询问。

解析

可以先用 $n-1$ 次询问查一个点和 1 的距离，即它的深度。
按深度从小到大确定每个节点的父亲，这样的话确定一个节点的父亲时其所有祖先一定都是已知的。
确定一个节点 k 的父亲之前，先对树已知的部分进行重链剖分。
从 1 开始，查询 1 所在重链的链底与 k 的距离，即可确定出 k 一直往上走与 1 所在的重链的交点 v，如右图：
如果 k 的深度等于 v 的深度加一，则 k 的父亲就是 v，否则往 k 方向走到对应儿子 w 处（由于是二叉树，这很简单就可以找出），再重复以上过程。



解析

根据轻边数量级别的结论，$O(\log n)$ 次就可以确定父亲。这样询问复杂度就是 $O(n\log n)$。

具体地，设 $T(n)$ 为最坏情况下在一棵大小为 n 的树中找到一个新节点的位置所需的询问次数，可以得到：

$$T(n) \le \begin{cases} 0 & n = 1 \\ T\left(\left\lfloor \frac{n-1}{2} \right\rfloor\right) + 1 & n \ge 2 \end{cases}$$

则最坏查询次数为 $2999 + \sum_{i=1}^{2999} T(i) \le 29940$，随机扰动后基本超不过 21000，可以通过。

时间复杂度为 $O(n^2)$。

NOI 2021 轻重边

给一棵 n 个结点的树，树上的每一条边可能是轻边或者重边。
接下来你需要对树进行 m 次操作，在所有操作开始前，树上所有边都是轻边。操作有以下两种：
1. 给定两个点 a 和 b，首先对于 a 到 b 路径上的所有点 x（包含 a 和 b），你要将与 x 相连的所有边变为轻边。然后再将 a 到 b 路径上包含的所有边变为重边。
2. 给定两个点 a 和 b，你需要计算当前 a 到 b 的路径上一共包含多少条重边。

多组数据，组数为 T。
$1 \leq T \leq 3$，$1 \leq n, m \leq 10^5$，$1 \leq a, b \leq n$。

提示：若第二类操作给出的 a 和 b 之间有边直接相连，怎么做。

解析

把一次修改操作当成染色操作，每一次修改都把从 x 到 y 的路径上的每个点染上一种之前没有被染过的颜色，初始时每个点都是无色的。

我们发现，两个点之间的边是重边，当且仅当两个点的颜色相同，且都不是无色。

那么就有从 x 到 y 的路径上的重边数量等于路径上点的数量减去颜色段数（无色不算颜色段）。

那么问题就变成了将两点间路径上每个点染上一种颜色，查询两点间颜色段数。

序列上做就是：用线段树维护，每个区间维护答案、左端点颜色和右端点颜色，合并的时候看一下左边的右端点和右边的左端点，再打一个区间赋值标记即可。树上做则是套一个树剖即可。

时间复杂度为 $O(T\log^2 n)$，空间复杂度为 $O(n)$，也可以用 LCT 来做。

UNR #1 火车管理

给一个 n 个栈组成的序列 $\{a_n\}$，初始时 n 个栈均空，你要支持如下操作：

1. 在 $a_{l\sim r}$ 的顶部放入元素 x。
2. 取出 a_l 的栈顶元素（a_l 为空则啥也不做）。
3. 查询 $a_{l\sim r}$ 的栈顶元素之和（一个栈为空视为栈顶元素为 0）。

共 m 个操作，强制在线。

$1 \le n, m \le 500000$，$1 \le l \le r \le n$，$1 \le x \le 3000$。

解析

建立一颗可持久化线段树，维护每个栈每个时间的栈顶和栈顶的入栈时间。

再维护一颗线段树用来统计答案。

1. 在可持久化线段树上进行区间覆盖，在答案线段树上修改一下。
2. 由于记录了入栈时间，所以我们可以轻松回退到之前的状态，然后再在答案线段树和可持久化线段树上修改一下即可。
3. 直接在答案线段树里查询。

时空复杂度均为 $O(m\log n)$。

CF741D Arpa's letter-marked tree and Mehrdad's Dokhtar-kosh paths

给一棵有 n 个点的以 1 为根的树，每条边上有一个字符（$a \sim v$ 共 22 种）。

一条简单路径被称为是好的当且仅当路径上的字符经过重新排序后可以变成一个回文串。

求每个子树中最长的好的路径的长度。

$1 \leq n \leq 5 \times 10^5$。

解析

可以重排成回文串当且仅当不超过一种字符的出现次数为奇数。

考虑记 a_u 为从根到 u 路径字符出现次数奇偶性的状态压缩，是一个 22 位二进制数。那么从 u 到 v 路径的状态就是 $a_u \operatorname{xor} a_v$。

再维护一个数组 $f_{u,S}$ 表示 u 的子树中满足 $a_v = S$ 的 v 中的最大深度。

我们直接继承重儿子的 f，然后再把轻儿子的暴力插入。

由于合法的状态只有 23 种，我们可以暴力插入一个点 v 的同时，枚举每种合法状态 T，再查询 $f_{u,a_v \operatorname{xor} T}$ 来更新答案。

时间复杂度为 $O(23n\log n)$，空间复杂度为 $O(n + 2^{22})$。

NOI 2020 命运

给定一棵 n 个点的树和 m 条限制，你可以给树上的每一条边赋一个 0 或 1 的权值。所有限制均可用两个参数 u, v（保证 v 为 u 的祖先）描述，表示你需要保证 u 到 v 上至少有一条边的权值为 1。

求给边赋值的方案数，对 998244353 取模。

$1 \leq n, m \leq 10^5$。

解析

对于下端点在 u 的所有限制，上端点深的能满足的话那么上端点浅的一定也能满足。

据此可以写出 DP 状态：$f_{u,i}$ 表示以 u 为根的子树内，下端点在子树内并且没有被满足的限制中上端点的最深深度为 i，对 u 的子树内赋值的方案数。若没有限制不满足，则 $i=0$。

可以列出转移：

$$f_{u,i} \leftarrow f_{u,i} \sum_{j=0}^{dep_u} f_{v,j} + f_{u,i} \sum_{j=0}^{i} f_{v,j} + f_{v,i} \sum_{j=0}^{i-1} f_{u,j}$$

第一个和式表示边 (u,v) 权值为 1，后面两个表示权值为 0。

令 $g_{u,i} = \sum_{j=0}^{i} f_{u,j}$ 进行前缀和优化即可做到 $O(n^2)$ 的复杂度。

解析

考虑线段树合并，g_{v,dep_u} 可以先从线段树上查一下，剩下的都是和下标相关的。

我们先合并左子树然后合并右子树，走右子树的时候就可以把左边的和加上，整体乘一个数可以打个标记即可。

谢谢聆听！