Summary

teafrogsf_cyh0709/colldisSavior

X402

August 18, 2018

这是一个很长的总结。

Introduction

更多地，差分是一种技巧。

2018-7-16 NOIpSimulation T2 Magic

你有 $10^{10^{10}}$ 个元件，初始时除了其中 n 个朝上，它们都朝下。朝上的元件编号为 $P_{1\ldots n}$。

2018-7-16 NOIpSimulation T2 Magic

你有 $10^{10^{10}}$ 个元件，初始时除了其中 n 个朝上，它们都朝下。朝上的元件编号为 $P_{1\ldots n}$。

你每次可以让长度为奇质数的连续区间的元件反向，求让所有元件朝上的最小操作次数。

$n \leq 100, P_i \leq 10^7$

2018-7-16 NOIpSimulation T2 Magic

考虑设 $C_i = [P_i \neq P_{i-1}]$，特别地，$C_1 = 0$。显然序列里 $C_i = 1$ 的数量为偶数。

2018-7-16 NOIpSimulation T2 Magic

考虑设 $C_i = [P_i \neq P_{i-1}]$，特别地，$C_1 = 0$。显然序列里 $C_i = 1$ 的数量为偶数。

则每次翻转只修改了 C_l 和 C_{r+1}。我们的目的就是通过各种操作使得 $\forall i, C_i = 0$。

2018-7-16 NOIpSimulation T2 Magic

考虑设 $C_i = [P_i \neq P_{i-1}]$，特别地，$C_1 = 0$。显然序列里 $C_i = 1$ 的数量为偶数。

则每次翻转只修改了 C_l 和 C_{r+1}。我们的目的就是通过各种操作使得 $\forall i, C_i = 0$。

实质上我们是要将当前局面的 $C_i = 1$ 的位置两两匹配来达到用尽量少的操作改变它们的目的。

2018-7-16 NOIpSimulation T2 Magic

考虑设 $C_i = [P_i \neq P_{i-1}]$，特别地，$C_1 = 0$。显然序列里 $C_i = 1$ 的数量为偶数。

则每次翻转只修改了 C_l 和 C_{r+1}。我们的目的就是通过各种操作使得 $\forall i, C_i = 0$。

实质上我们是要将当前局面的 $C_i = 1$ 的位置两两匹配来达到用尽量少的操作改变它们的目的。

考虑两数之差若为奇质数，只需要一次操作。

2018-7-16 NOIpSimulation T2 Magic

考虑设 $C_i = [P_i \neq P_{i-1}]$，特别地，$C_1 = 0$。显然序列里 $C_i = 1$ 的数量为偶数。

则每次翻转只修改了 C_l 和 C_{r+1}。我们的目的就是通过各种操作使得 $\forall i, C_i = 0$。

实质上我们是要将当前局面的 $C_i = 1$ 的位置两两匹配来达到用尽量少的操作改变它们的目的。

考虑两数之差若为奇质数，只需要一次操作。

若为偶数，则需要两次操作。这与哥德巴赫猜想类似，当然 2 可以用 5 和 3，4 可以用 7 和 3。

2018-7-16 NOIpSimulation T2 Magic

考虑设 $C_i = [P_i \neq P_{i-1}]$，特别地，$C_1 = 0$。显然序列里 $C_i = 1$ 的数量为偶数。

则每次翻转只修改了 C_l 和 C_{r+1}。我们的目的就是通过各种操作使得 $\forall i, C_i = 0$。

实质上我们是要将当前局面的 $C_i = 1$ 的位置两两匹配来达到用尽量少的操作改变它们的目的。

考虑两数之差若为奇质数，只需要一次操作。

若为偶数，则需要两次操作。这与哥德巴赫猜想类似，当然 2 可以用 5 和 3，4 可以用 7 和 3。

若为奇合数，则需要分解成偶数与奇质数之差。

2018-7-16 NOIpSimulation T2 Magic

考虑设 $C_i = [P_i \neq P_{i-1}]$，特别地，$C_1 = 0$。显然序列里 $C_i = 1$ 的数量为偶数。

则每次翻转只修改了 C_l 和 C_{r+1}。我们的目的就是通过各种操作使得 $\forall i, C_i = 0$。

实质上我们是要将当前局面的 $C_i = 1$ 的位置两两匹配来达到用尽量少的操作改变它们的目的。

考虑两数之差若为奇质数，只需要一次操作。

若为偶数，则需要两次操作。这与哥德巴赫猜想类似，当然 2 可以用 5 和 3，4 可以用 7 和 3。

若为奇合数，则需要分解成偶数与奇质数之差。

然后我们可以按照奇偶建二分图，若为奇质数就连边，直接跑最大匹配。每剩下同奇偶的一对就让 $ans+ = 2$，若最后还剩下一对奇偶不同的就让 $ans+ = 3$。

teafrogsf's NOIpSimulation D1T2 Leap

Introduction

虚树是一种可以称为数据结构的解题方法。它适用于树上如下情况：

- 1 两点之间的直接信息能够通过 $O(1)$ 或 $O(\log)$ 级别等较低级的复杂度得出，或能离线得出；
- 2 保持原树的结构对解题有利；
- 3 缩链不会影响答案的询问；
- 4 询问总点数 $\sum k$ 不多。

若满足 1/2，我们就可以对每一个询问建立虚树，在虚树上用暴力的解题方法解决问题。因为一般时间复杂度为 $O(queryinfo \times \sum k)$，所以需要满足 4 我们才能解决问题。

Construction

具体来说，虚树的构建方式如下：
首先预处理 dfs 序（包括出栈序 $out[u]$），ST 表。

- 1 对所有询问点按 dfs 序排序，然后两两之间求 LCA。可以证明这样可以求出所有的 LCA，并与 k 同阶。~~具体的证明可以看 zjB 博客上那个不清不楚的感性理解法~~
- 2 重新按 dfs 序排序并去重。
- 3 用一个栈维护虚树序列，如果 $out[stk[tp]] < dfn[vir[i]]$ 就一直弹掉（当 $out[stk[tp]] >= dfn[vir[i]]$ 时 $stk[tp]$ 是 $vir[i]$ 的祖先 实际上不可能出现 $=$ 的情况 ）。
- 4 此时的 $stk[tp]$ 是 $vir[i]$ 的祖先，直接连边并把 $vir[i]$ 入栈。
- 5 重复这个过程，直到虚树序列被完全遍历。

[SDOI2011] 消耗战

在一场战争中，战场由 n 个岛屿和 n-1 个桥梁组成，保证每两个岛屿间有且仅有一条路径可达。现在，我军已经侦查到敌军的总部在编号为 1 的岛屿，而且他们已经没有足够多的能源维系战斗，我军胜利在望。已知在其他 k 个岛屿上有丰富能源，为了防止敌军获取能源，我军的任务是炸毁一些桥梁，使得敌军不能到达任何能源丰富的岛屿。由于不同桥梁的材质和结构不同，所以炸毁不同的桥梁有不同的代价，我军希望在满足目标的同时使得总代价最小。

侦查部门还发现，敌军有一台神秘机器。即使我军切断所有能源之后，他们也可以用那台机器。机器产生的效果不仅仅会修复所有我军炸毁的桥梁，而且会重新随机资源分布（但可以保证的是，资源不会分布到 1 号岛屿上）。不过侦查部门还发现了这台机器只能够使用 m 次，所以我们只需要把每次任务完成即可。

[SDOI2011] 消耗战

设 $dp[i]$ 表示切断 i 的所有子树所需要的最小代价。显然可以发现对于一个 $v \in son[u]$，要么断开 $1 \to v$ 的最小边权，要么把所有子树对应的 dp 值都断掉。

这个过程可以在虚树上处理，预处理记一下最小边权直接 DP 就行了。

[HNOI2014] 世界树

给定一棵树，每个点会被它最近的关键点管辖。
多次询问给定关键点，求每个关键点管辖的点数量。
$n, q, \sum k \leq 3 \times 10^5$

[HNOI2014] 世界树

我们需要在虚树上进行两次 dfs 求出每个虚树上的点被哪些点管辖。用两遍 dfs 的原因是必须把所有的儿子遍历完再与父亲作比较，这样得到的答案才是正确的。

然后对一条虚树上的边，我们分类讨论。

[HNOI2014] 世界树

我们需要在虚树上进行两次 dfs 求出每个虚树上的点被哪些点管辖。用两遍 dfs 的原因是必须把所有的儿子遍历完再与父亲作比较，这样得到的答案才是正确的。

然后对一条虚树上的边，我们分类讨论。

如果两个点同属一个关键点 x，那么 $ans[x]+=siz[son[fa]]-siz[u]$。其中，$son[fa]$ 表示 fa 在 (u,fa) 这条链上的第一个儿子。

否则，我们可以用倍增找到 $bln[x],bln[fa]$ 管辖的分界点 mid，然后两边的答案分别相加。

[HNOI2014] 世界树

我们需要在虚树上进行两次 dfs 求出每个虚树上的点被哪些点管辖。用两遍 dfs 的原因是必须把所有的儿子遍历完再与父亲作比较，这样得到的答案才是正确的。

然后对一条虚树上的边，我们分类讨论。

如果两个点同属一个关键点 x，那么 $ans[x]+=siz[son[fa]]-siz[u]$。其中，$son[fa]$ 表示 fa 在 (u,fa) 这条链上的第一个儿子。

否则，我们可以用倍增找到 $bln[x],bln[fa]$ 管辖的分界点 mid，然后两边的答案分别相加。

注意这样并不一定可以讨论完原树的所有边，每次讨论都要把原树的这部分 siz 减掉，最后直接让 $ans[bln[u]]+=remainsiz[u]$ 就可以了，因为这部分儿子肯定是属于 $bln[u]$ 的。另外，可以把自己这个点直接放到这个部分来加到答案里，这样更加巧妙。

实现上有一些细节需要注意。

Introduction

~~为什么不写点分治呢，因为我基本上不会点分治。~~

Introduction

~~为什么不写点分治呢，因为我基本上不会点分治。~~

静态点分树是指一般点分树的题型，这类题型通常不会要求你更改树的形态，因此点分树结构只需要预处理就可得出。

Introduction

~~为什么不写点分治呢，因为我基本上不会点分治。~~

静态点分树是指一般点分树的题型，这类题型通常不会要求你更改树的形态，因此点分树结构只需要预处理就可得出。

点分树是一个巧妙的暴力数据结构，为了不用 ST 表求两点距离减少常数，一般地，我们会将一个点的所有点分树上的祖先信息预先保存，这样只消耗了 $O(n \log n)$ 的时空复杂度且减少了常数。

Introduction

~~为什么不写点分治呢，因为我基本上不会点分治。~~

静态点分树是指一般点分树的题型，这类题型通常不会要求你更改树的形态，因此点分树结构只需要预处理就可得出。

点分树是一个巧妙的暴力数据结构，为了不用 ST 表求两点距离减少常数，一般地，我们会将一个点的所有点分树上的祖先信息预先保存，这样只消耗了 $O(n \log n)$ 的时空复杂度且减少了常数。

点分树的性质主要有两条：

- 1 树高最多为 $O(log)$。
- 2 点分树上两点的 LCA 在原树路径上。

另外 zzq 说点分树是树上的线段树。

Construction

其实我不是很会。

Construction

其实我不是很会。

与线段树不同的是，点分树的题构建非常灵活。存祖先的信息一般在 dfsinfo 里进行，每遍历到当前子树里的某个点 x 就让 $anc[x][dep] = root$。按深度存能够保证不重复。其他信息在这里也一并处理。

[HNOI2015] 开店

一棵每个点度数 ≤ 3 的树，每个点有点权，每条边有边权。

每次询问给定一个点 u 和两个值 L, R，求权值在 $[L, R]$ 之间的所有点到 u 的带权距离和。

[HNOI2015] 开店

一棵每个点度数 ≤ 3 的树，每个点有点权，每条边有边权。

每次询问给定一个点 u 和两个值 L, R，求权值在 $[L, R]$ 之间的所有点到 u 的带权距离和。

强制在线。

$n \leq 1.5 \times 10^5, q \leq 2 \times 10^5, A \leq 10^9, c \leq 10^3$，其中 A 是点权上限，c 是边权上限。

[HNOI2015] 开店

首先考虑带权距离和在点分树上是怎么计算的。

[HNOI2015] 开店

首先考虑带权距离和在点分树上是怎么计算的。

对于任何一个点 x，所有点到它的带权距离和包含两个部分：它到点分树上祖先的距离 dis 乘上它祖先的 siz；以及 anc 管辖的点到 anc 的带权距离和 sum，当然这个距离和不能包括 x 这块子树的部分。

[HNOI2015] 开店

首先考虑带权距离和在点分树上是怎么计算的。

对于任何一个点 x，所有点到它的带权距离和包含两个部分：它到点分树上祖先的距离 dis 乘上它祖先的 siz；以及 anc 管辖的点到 anc 的带权距离和 sum，当然这个距离和不能包括 x 这块子树的部分。

考虑构建点分树，每进入一个分治子树，子树内的所有点存储以下信息：自己点分树上的祖先 anc，到这个祖先的距离 dis，和当前点对应的祖先的子树编号 num。存第三个信息是为了方便减去这个点对应子树的信息。

[HNOI2015] 开店

首先考虑带权距离和在点分树上是怎么计算的。

对于任何一个点 x，所有点到它的带权距离和包含两个部分：它到点分树上祖先的距离 dis 乘上它祖先的 siz；以及 anc 管辖的点到 anc 的带权距离和 sum，当然这个距离和不能包括 x 这块子树的部分。

考虑构建点分树，每进入一个分治子树，子树内的所有点存储以下信息：自己点分树上的祖先 anc，到这个祖先的距离 dis，和当前点对应的祖先的子树编号 num。存第三个信息是为了方便减去这个点对应子树的信息。

每个点还要存储自己子树里每个点的权值、子树大小和深度。为了能够简便减掉自己这个子树的贡献，我们可以分子树存储每一块的答案，只需要记录一下 dfsinfo 到的每个点属于当前分治重心第几个儿子所管辖就好了。

[HNOI2015] 开店

首先考虑带权距离和在点分树上是怎么计算的。

对于任何一个点 x，所有点到它的带权距离和包含两个部分：它到点分树上祖先的距离 dis 乘上它祖先的 siz；以及 anc 管辖的点到 anc 的带权距离和 sum，当然这个距离和不能包括 x 这块子树的部分。

考虑构建点分树，每进入一个分治子树，子树内的所有点存储以下信息：自己点分树上的祖先 anc，到这个祖先的距离 dis，和当前点对应的祖先的子树编号 num。存第三个信息是为了方便减去这个点对应子树的信息。

每个点还要存储自己子树里每个点的权值、子树大小和深度。为了能够简便减掉自己这个子树的贡献，我们可以分子树存储每一块的答案，只需要记录一下 dfsinfo 到的每个点属于当前分治重心第几个儿子所管辖就好了。

这里需要注意因为点分树没有原树的父子关系，所以必须要每个点单独存储好自己管辖的所有点的信息，并且各个重心之间的信息差也不能简单相减。于是空间复杂度是 $O(n \log n)$ 的。

[HNOI2015] 开店

那么明白了这个，我们当然可以暴跳点分树解决问题啦。但怎么统计在 $[L, R]$ 范围内的呢？

[HNOI2015] 开店

那么明白了这个，我们当然可以暴跳点分树解决问题啦。但怎么统计在 $[L, R]$ 范围内的呢？

因为答案的计算，我们可以按权值排序累上后缀和（所以为了方便我们多存一个 val，并把这三个数据存到一个 vector 里），这样我们就已经求出了按点权排序的一堆 siz 和与 dep 也就是带权距离和。每次询问遍历祖先的时候我们就可以直接用 $lower\_bound$，$upper\_bound$ 二分出一个左闭右开的区间，直接相减就能算出第二部分的 siz 与 dep 和，计好对应答案就能计算完了。对于第一部分我们单独判断一下 anc 的权值范围是否满足。

[HNOI2015] 开店

那么明白了这个，我们当然可以暴跳点分树解决问题啦。但怎么统计在 $[L, R]$ 范围内的呢？

因为答案的计算，我们可以按权值排序累上后缀和（所以为了方便我们多存一个 val，并把这三个数据存到一个 vector 里），这样我们就已经求出了按点权排序的一堆 siz 和与 dep 也就是带权距离和。每次询问遍历祖先的时候我们就可以直接用 $lower\_bound$，$upper\_bound$ 二分出一个左闭右开的区间，直接相减就能算出第二部分的 siz 与 dep 和，计好对应答案就能计算完了。对于第一部分我们单独判断一下 anc 的权值范围是否满足。

因为点度数最多为 3，所以暴力遍历儿子的复杂度是没有问题的。点分治预处理的复杂度是 $O(n \log^2 n)$，查询的复杂度是 $O(q \log n)$。

2018-8-7 NOIpSimulation T3 Revive

一棵树，每条边有边权 w_i，定义 $liveliness(u,v)=\sum_{u\to v} w_i$。整棵树的生机值为：

$$\sum_{u=1}^{n}\sum_{v=u+1}^{n} liveliness^2(u,v)$$

2018-8-7 NOIpSimulation T3 Revive

一棵树，每条边有边权 w_i，定义 $liveliness(u,v)=\sum_{u\to v} w_i$。整棵树的生机值为：

$$\sum_{u=1}^{n}\sum_{v=u+1}^{n} liveliness^2(u,v)$$

有 q 个操作，每次更新（增加）一条边权。求所有操作前及每次操作后的生机值。

2018-8-7 NOIpSimulation T3 Revive

一棵树，每条边有边权 w_i，定义 $liveliness(u,v)=\sum_{u\to v} w_i$。整棵树的生机值为：

$$\sum_{u=1}^{n}\sum_{v=u+1}^{n} liveliness^2(u,v)$$

有 q 个操作，每次更新（增加）一条边权。求所有操作前及每次操作后的生机值。

$n\le 10^5, q\le 5\times 10^5$

2018-8-7 NOIpSimulation T3 Revive

这题首先要把平方拆开:

2018-8-7 NOIpSimulation T3 Revive

这题首先要把平方拆开：

$$ans = \sum (\sum_{u<v} \sum_{u \to v} w_i)^2$$

2018-8-7 NOIpSimulation T3 Revive

这题首先要把平方拆开：

$$ans = \sum_{u<v}(\sum_{u\to v} w_i)^2$$

$$= \sum_{u<v}(\sum_{u\to v} w_i^2 + \sum_{a,b\in(u\to v)} 2w_a \times w_b)$$

2018-8-7 NOIpSimulation T3 Revive

这题首先要把平方拆开:

$$ans = \sum_{u<v}(\sum_{u\to v} w_i)^2$$

$$= \sum_{u<v}(\sum_{u\to v} w_i^2 + \sum_{a,b\in(u\to v)} 2w_a \times w_b)$$

$$= \sum_i w_i^2 \times (passby\ e) + \sum_{a<b} 2w_a \times w_b \times (passby\ a, b)$$

2018-8-7 NOIpSimulation T3 Revive

前面部分的贡献很简单，就是一条边两端的 siz 直接乘一下。主要考虑后面部分。

2018-8-7 NOIpSimulation T3 Revive

前面部分的贡献很简单，就是一条边两端的 siz 直接乘一下。主要考虑后面部分。

此时你可以用其他的一些方法譬如树剖或 DFS 序 +BIT 解决，但我太菜了都不会。

2018-8-7 NOIpSimulation T3 Revive

前面部分的贡献很简单，就是一条边两端的 siz 直接乘一下。主要考虑后面部分。

此时你可以用其他的一些方法譬如树剖或 DFS 序 +BIT 解决，但我太菜了都不会。

考虑~~比较毒瘤~~的点分治，每次计算经过分治重心点的那几条边的贡献。

2018-8-7 NOIpSimulation T3 Revive

前面部分的贡献很简单，就是一条边两端的 siz 直接乘一下。主要考虑后面部分。

此时你可以用其他的一些方法譬如树剖或 DFS 序 +BIT 解决，但我太菜了都不会。

考虑~~比较毒瘤~~的点分治，每次计算经过分治重心点的那几条边的贡献。

首先预处理前面部分的答案，然后对于走到的每个分治重心，对每个子树两两互相乘相应的答案，这个套路和 [APIO2018] 铁人两项比较像。

2018-8-7 NOIpSimulation T3 Revive

前面部分的贡献很简单，就是一条边两端的 siz 直接乘一下。主要考虑后面部分。

此时你可以用其他的一些方法譬如树剖或 DFS 序 +BIT 解决，但我太菜了都不会。

考虑~~比较毒瘤~~的点分治，每次计算经过分治重心点的那几条边的贡献。

首先预处理前面部分的答案，然后对于走到的每个分治重心，对每个子树两两互相乘相应的答案，这个套路和 [APIO2018] 铁人两项比较像。

考虑如何计算子树内的答案和，dfsinfo 的时候，记好每条边在当前分治结构的方向下管辖的 siz 也就是 sub，乘上边权就得到了这条边的答案。sub 可以通过预处理得到。这个和上题类似，只不过是存的边。

2018-8-7 NOIpSimulation T3 Revive

再考虑如何如何让它资瓷修改，我们要把分治结构存下来，存好当前分治结构下每条边的对应 sub，并记好每条边在当前分治结构下被哪条边管辖，当然还有每条边对应子树的贡献系数 sum 以及这个分治重心点的所有子树贡献系数之和 ans。

2018-8-7 NOIpSimulation T3 Revive

再考虑如何如何让它资瓷修改，我们要把分治结构存下来，存好当前分治结构下每条边的对应 sub，并记好每条边在当前分治结构下被哪条边管辖，当然还有每条边对应子树的贡献系数 sum 以及这个分治重心点的所有子树贡献系数之和 ans。

在修改的时候，我们暴跳点分树，对每层深度，我们算出这条边的新增贡献 $delta$，除了这条边对应的子树之外当前重心的其他所有子树的系数都要乘上 $delta$，这个比较显然。

2018-8-7 NOIpSimulation T3 Revive

再考虑如何如何让它资瓷修改，我们要把分治结构存下来，存好当前分治结构下每条边的对应 sub，并记好每条边在当前分治结构下被哪条边管辖，当然还有每条边对应子树的贡献系数 sum 以及这个分治重心点的所有子树贡献系数之和 ans。

在修改的时候，我们暴跳点分树，对每层深度，我们算出这条边的新增贡献 $delta$，除了这条边对应的子树之外当前重心的其他所有子树的系数都要乘上 $delta$，这个比较显然。

然后我们再把 sum, ans 加上对应更新的值，最后更新好前面部分的答案，这个式子非常简单就不写了。最后的最后更新当前边的权值。

2018-8-7 NOIpSimulation T3 Revive

再考虑如何如何让它资瓷修改，我们要把分治结构存下来，存好当前分治结构下每条边的对应 sub，并记好每条边在当前分治结构下被哪条边管辖，当然还有每条边对应子树的贡献系数 sum 以及这个分治重心点的所有子树贡献系数之和 ans。

在修改的时候，我们暴跳点分树，对每层深度，我们算出这条边的新增贡献 $delta$，除了这条边对应的子树之外当前重心的其他所有子树的系数都要乘上 $delta$，这个比较显然。

然后我们再把 sum, ans 加上对应更新的值，最后更新好前面部分的答案，这个式子非常简单就不写了。最后的最后更新当前边的权值。

具体实现的话我们可以让邻接表/链式前向星的初始编号设为 3，这样每次遍历的时候让当前编号 $i/2$ 就是对应边的编号，可以方便地访问或存储信息。

2018-8-12 NOIpSimulation T3 C

这题太神了。

Introduction

长链剖分在 OI 界的应用不广，因为很少有出题人将范围卡到只有 $O(n)$ 能够通过的时候。

Introduction

长链剖分在 OI 界的应用不广，因为很少有出题人将范围卡到只有 $O(n)$ 能够通过的时候。

但它毕竟也算是一个 Trick，可以学习一下套路。

Introduction

长链剖分在 OI 界的应用不广，因为很少有出题人将范围卡到只有 $O(n)$ 能够通过的时候。

但它毕竟也算是一个 Trick，可以学习一下套路。

长链剖分的主要应用是优化以深度为下标的 DP，有 $O(n)$ 的精美复杂度。

Introduction

长链剖分在 OI 界的应用不广，因为很少有出题人将范围卡到只有 $O(n)$ 能够通过的时候。

但它毕竟也算是一个 Trick，可以学习一下套路。

长链剖分的主要应用是优化以深度为下标的 DP，有 $O(n)$ 的精美复杂度。~~问题是我重剖/线段树合并/dsu on tree $O(n\log n)$ 能过又好写的话谁写这个鬼畜的长链剖分啊喂~~

Introduction

长链剖分在 OI 界的应用不广，因为很少有出题人将范围卡到只有 $O(n)$ 能够通过的时候。

但它毕竟也算是一个 Trick，可以学习一下套路。

长链剖分的主要应用是优化以深度为下标的 DP，有 $O(n)$ 的精美复杂度。~~问题是我重剖/线段树合并/dsu on tree $O(n\log n)$ 能过又好写的话谁写这个鬼畜的长链剖分啊喂~~

除此之外有一个不知道有什么用的 $O(n\log n)$ 预处理 $O(1)$ 查询 K 级祖先套路和最坏 $O(\sqrt{n})$ 的跳链，就暂且不提了。

Construction

其实我也不是很会。

Construction

其实我也不是很会。

当然剖分长链是很 SB 的，跟重剖并没有什么太大区别，存一下自己儿子的最大深度 mdp 就可以了。

Construction

其实我也不是很会。

当然剖分长链是很 SB 的，跟重剖并没有什么太大区别，存一下自己儿子的最大深度 mdp 就可以了。~~轻松简单。~~

Construction

其实我也不是很会。

当然剖分长链是很 SB 的，跟重剖并没有什么太大区别，存一下自己儿子的最大深度 mdp 就可以了。~~轻松简单。~~

重点在于剖分完之后的 DP 部分。

Construction

其实我也不是很会。

当然剖分长链是很 SB 的，跟重剖并没有什么太大区别，存一下自己儿子的最大深度 mdp 就可以了。~~轻松简单。~~

重点在于剖分完之后的 DP 部分。

注意到你 DP 的转移式子当只有一个儿子的时候，是可以直接赋值的。

Construction

其实我也不是很会。

当然剖分长链是很 SB 的，跟重剖并没有什么太大区别，存一下自己儿子的最大深度 mdp 就可以了。~~轻松简单。~~

重点在于剖分完之后的 DP 部分。

注意到你 DP 的转移式子当只有一个儿子的时候，是可以直接赋值的。

所以我们沿着长链走，在长链的最后一个点给它分配空间，并赋好初值。

Construction

其实我也不是很会。

当然剖分长链是很 SB 的，跟重剖并没有什么太大区别，存一下自己儿子的最大深度 mdp 就可以了。~~轻松简单。~~

重点在于剖分完之后的 DP 部分。

注意到你 DP 的转移式子当只有一个儿子的时候，是可以直接赋值的。

所以我们沿着长链走，在长链的最后一个点给它分配空间，并赋好初值。

之后我们根据转移式子移动儿子 DP 数组的地址来转移到自己本身。

Construction

其实我也不是很会。

当然剖分长链是很 SB 的，跟重剖并没有什么太大区别，存一下自己儿子的最大深度 mdp 就可以了。~~轻松简单。~~

重点在于剖分完之后的 DP 部分。

注意到你 DP 的转移式子当只有一个儿子的时候，是可以直接赋值的。

所以我们沿着长链走，在长链的最后一个点给它分配空间，并赋好初值。

之后我们根据转移式子移动儿子 DP 数组的地址来转移到自己本身。

Construction

大部分的实现是使用指针，但我太菜了不会。

Construction

大部分的实现是使用指针，但我太菜了不会。

STL 的 deque 有个优秀的性质是可以调用下标，所以我们可以利用它来进行动态地分配内存 $(resize)$，并利用 $push/pop\_front/back$ 来模拟指针的移动。

Construction

大部分的实现是使用指针，但我太菜了不会。

STL 的 deque 有个优秀的性质是可以调用下标，所以我们可以利用它来进行动态地分配内存 $(resize)$，并利用 $push/pop\_front/back$ 来模拟指针的移动。缺点是如果转移隔的很远的话复杂度可能会出锅，~~但我也没做过什么这样的题就算了吧 qwq~~

Construction

大部分的实现是使用指针，但我太菜了不会。

STL 的 deque 有个优秀的性质是可以调用下标，所以我们可以利用它来进行动态地分配内存 $(resize)$，并利用 $push/pop\_front/back$ 来模拟指针的移动。缺点是如果转移隔的很远的话复杂度可能会出锅，~~但我也没做过什么这样的题就算了吧 qwq~~

一定要注意转移完之后要内存回收。

Construction

大部分的实现是使用指针，但我太菜了不会。

STL 的 deque 有个优秀的性质是可以调用下标，所以我们可以利用它来进行动态地分配内存 $(resize)$，并利用 $push/pop\_front/back$ 来模拟指针的移动。缺点是如果转移隔的很远的话复杂度可能会出锅，~~但我也没做过什么这样的题就算了吧 qwq~~

一定要注意转移完之后要内存回收。

然后？然后就直接 DP 啊。

Construction

大部分的实现是使用指针，但我太菜了不会。

STL 的 deque 有个优秀的性质是可以调用下标，所以我们可以利用它来进行动态地分配内存 $(resize)$，并利用 $push/pop\_front/back$ 来模拟指针的移动。缺点是如果转移隔的很远的话复杂度可能会出锅，~~但我也没做过什么这样的题就算了吧 qwq~~

一定要注意转移完之后要内存回收。

然后？然后就直接 DP 啊。可以证明，这样的复杂度是 $O(n)$。~~我不会证明我不会~~

这题卡空间卡了我好久，为啥 BZOJ 只开 128MB ~~虽然这教会了我 pop 是可以回收内存的~~

[POI2014]Hotel ~~Deluxe Edition~~

Introduction

You

Thank