树上搜索

树上搜索与搜索序及其应用

符水波

宁波市镇海中学

简单树上搜索

树上主要应用深搜。

Problem (DFS 遍历树)

给你一棵有根树，输出每个点的深度，子树大小，以及父亲。

Problem (DFS 遍历树)

给你一棵有根树，输出每个点的深度，子树大小，以及父亲。
要求时间复杂度 $O(n)$。

树的直径

- 树的直径：树上最长的路径。（通常，我们只考虑边权非负情况下的直径。）

Problem (树的直径)

给出一棵树，求出树上最远的点对的距离。

Problem (树的直径)

给出一棵树，求出树上最远的点对的距离。
要求时间复杂度 $O(n)$。

Problem (树的直径)

给出一棵树，求出树上最远的点对的距离。
要求时间复杂度 $O(n)$。

Solution (树的直径)

先随便钦定一个点，dfs 出最远的点。

Problem (树的直径)

给出一棵树，求出树上最远的点对的距离。
要求时间复杂度 $O(n)$。

Solution (树的直径)

先随便钦定一个点，dfs 出最远的点。
在找出那个点上再找一次最远的点。

我们可以看一个直径性质证明的示例。

我们可以看一个直径性质证明的示例。

Problem (示例)

试证明如果树有多条直径，则必定交于一点。

我们可以看一个直径性质证明的示例。

Problem (示例)

试证明如果树有多条直径，则必定交于一点。

Proof (示例)

由树的连通性，设有两条不交于一点的长为 d 的直径 $s_1, \dots, u, \dots, t_1$ 与 $s_2, \dots, v, \dots, t_2$ 通过长为 l 的 $u, \dots, v$ 的路径相连。

设 s_1 到 u 路径长为 $l_{1,1}$，t_1 到 u 路径长为 $l_{1,2}$；s_2 到 v 路径长为 $l_{2,1}$，t_2 到 v 路径长为 $l_{2,2}$。

我们可以看一个直径性质证明的示例。

Problem (示例)

试证明如果树有多条直径，则必定交于一点。

Proof (示例)

由树的连通性，设有两条不交于一点的长为 d 的直径 $s_1, \ldots, u, \ldots, t_1$ 与 $s_2, \ldots, v, \ldots, t_2$ 通过长为 l 的 $u, \ldots, v$ 的路径相连。

设 s_1 到 u 路径长为 $l_{1,1}$，t_1 到 u 路径长为 $l_{1,2}$；s_2 到 v 路径长为 $l_{2,1}$，t_2 到 v 路径长为 $l_{2,2}$。

则存在一条长为 $\max(l_{1,1}, l_{1,2}) + l + \max(l_{2,1}, l_{2,2}) \geqslant \frac{d}{2} + l + \frac{d}{2} > d$ 的路径。

我们可以看一个直径性质证明的示例。

Problem (示例)

试证明如果树有多条直径，则必定交于一点。

Proof (示例)

由树的连通性，设有两条不交于一点的长为 d 的直径 $s_1, \dots, u, \dots, t_1$ 与 $s_2, \dots, v, \dots, t_2$ 通过长为 l 的 $u, \dots, v$ 的路径相连。

设 s_1 到 u 路径长为 $l_{1,1}$，t_1 到 u 路径长为 $l_{1,2}$；s_2 到 v 路径长为 $l_{2,1}$，t_2 到 v 路径长为 $l_{2,2}$。

则存在一条长为 $\max(l_{1,1}, l_{1,2}) + l + \max(l_{2,1}, l_{2,2}) \geqslant \frac{d}{2} + l + \frac{d}{2} > d$ 的路径。

显然假设不成立。

由此可知，树的多条直径必交于其中点。

Proof (树的直径)

同理可证第一次 dfs 后得到的点一定为其中一条直径的一个端点。

Proof (树的直径)

同理可证第一次 dfs 后得到的点一定为其中一条直径的一个端点。
显然第二次 dfs 便可得到直径。
证毕。

Problem (CF734E Anton and Tree)

给一棵 $n(n \leqslant 200000)$ 个结点的树，每个点为黑色或白色，一次操作可以使相同颜色的连通块变成另一种颜色，求使整棵树变成一种颜色的最少操作数。

Problem (CF734E Anton and Tree)

给一棵 $n(n \leqslant 200000)$ 个结点的树，每个点为黑色或白色，一次操作可以使相同颜色的连通块变成另一种颜色，求使整棵树变成一种颜色的最少操作数。

Solution (CF734E Anton and Tree)

并查集缩同色点

Problem (CF734E Anton and Tree)

给一棵 $n(n \leqslant 200000)$ 个结点的树，每个点为黑色或白色，一次操作可以使相同颜色的连通块变成另一种颜色，求使整棵树变成一种颜色的最少操作数。

Solution (CF734E Anton and Tree)

并查集缩同色点
跑一遍直径就好了

Solution (CF734E Anton and Tree)

具体而言，初始时的同色连通块，后续操作时肯定是一起改颜色的，因此可以缩成一个点处理。

Solution (CF734E Anton and Tree)

具体而言，初始时的同色连通块，后续操作时肯定是一起改颜色的，因此可以缩成一个点处理。

缩完点了以后，我们观察到，相邻结点颜色黑白交错。

Solution (CF734E Anton and Tree)

具体而言，初始时的同色连通块，后续操作时肯定是一起改颜色的，因此可以缩成一个点处理。

缩完点了以后，我们观察到，相邻结点颜色黑白交错。

设直径长度为 l。显然至少 $\left\lceil \dfrac{l-1}{2} \right\rceil$ 次操作，否则无法使直径达到同一种颜色。

Solution (CF734E Anton and Tree)

具体而言，初始时的同色连通块，后续操作时肯定是一起改颜色的，因此可以缩成一个点处理。

缩完点了以后，我们观察到，相邻结点颜色黑白交错。

设直径长度为 l。显然至少 $\left\lceil\frac{l-1}{2}\right\rceil$ 次操作，否则无法使直径达到同一种颜色。

我们考虑构造一种方案，使得能在 $\left\lceil\frac{l-1}{2}\right\rceil$ 的次数内完成。

Solution (CF734E Anton and Tree)

具体而言，初始时的同色连通块，后续操作时肯定是一起改颜色的，因此可以缩成一个点处理。

缩完点了以后，我们观察到，相邻结点颜色黑白交错。

设直径长度为 l。显然至少 $\left\lceil \frac{l-1}{2} \right\rceil$ 次操作，否则无法使直径达到同一种颜色。

我们考虑构造一种方案，使得能在 $\left\lceil \frac{l-1}{2} \right\rceil$ 的次数内完成。

我们每次找到直径的中点并改变颜色。（如果 l 为偶数则任取一个。）

那么只要直径长度大于 2，即选的中点不是直径的一个端点，重新缩点了以后直径长度减少了 2。

数学归纳法即可。

Problem (CF911F Tree Destruction)

给你一棵树，每次挑选这棵树的两个叶子，把答案加上他们之间的距离，然后将其中一个点去掉，问你距离之和最大可以是多少。要求输出方案。

Problem (CF911F Tree Destruction)

给你一棵树，每次挑选这棵树的两个叶子，把答案加上他们之间的距离，然后将其中一个点去掉，问你距离之和最大可以是多少。要求输出方案。

Solution (CF911F Tree Destruction)

我们可以发现，每个点被删掉的那次操作最大贡献为那个点到离那个点最远的点。

Problem (CF911F Tree Destruction)

给你一棵树，每次挑选这棵树的两个叶子，把答案加上他们之间的距离，然后将其中一个点去掉，问你距离之和最大可以是多少。要求输出方案。

Solution (CF911F Tree Destruction)

我们可以发现，每个点被删掉的那次操作最大贡献为那个点到离那个点最远的点。

显然对于除了一条直径上的点以外都可以做到。

Problem (CF911F Tree Destruction)

给你一棵树，每次挑选这棵树的两个叶子，把答案加上他们之间的距离，然后将其中一个点去掉，问你距离之和最大可以是多少。要求输出方案。

Solution (CF911F Tree Destruction)

我们可以发现，每个点被删掉的那次操作最大贡献为那个点到离那个点最远的点。

显然对于除了一条直径上的点以外都可以做到。

删掉那些点后最后删一条直径即可。

Problem (CF1000E We Need More Bosses)

给一张无向图，要求找一对 S 和 T，使得其路径上的割边是最多的，输出其数量。

Problem (CF1000E We Need More Bosses)

给一张无向图，要求找一对 S 和 T，使得其路径上的割边是最多的，输出其数量。

Solution (CF1000E We Need More Bosses)

将边双缩点后求树的直径端点即可。

Problem (CF1073F Choosing Two Paths)

有一棵树，从中选取 2 条链，其中任何一条链的端点不能被另一条链包含，求这两条链，使这两条链的公共的点的部分最长，若有相同解，使得总长度最长。

Problem (CF1073F Choosing Two Paths)

有一棵树，从中选取 2 条链，其中任何一条链的端点不能被另一条链包含，求这两条链，使这两条链的公共的点的部分最长，若有相同解，使得总长度最长。

Solution (CF1073F Choosing Two Paths)

我们找到树中有两个或更多儿子的结点。显然只有这些结点才能为公共部分点的端点。以这些点为新的叶子结点，树的直径即可。

Problem (CF1073F Choosing Two Paths)

有一棵树，从中选取 2 条链，其中任何一条链的端点不能被另一条链包含，求这两条链，使这两条链的公共的点的部分最长，若有相同解，使得总长度最长。

Solution (CF1073F Choosing Two Paths)

我们找到树中有两个或更多儿子的结点。显然只有这些结点才能为公共部分点的端点。以这些点为新的叶子结点，树的直径即可。

关于总长度最长，可以在最长路的基础上优先选择从端点向外两条最长链之和最长的一个即可。

利用各类性质将树上问题转化为链上问题，是解决一些棘手问题的一种方法。

利用各类性质将树上问题转化为链上问题，是解决一些棘手问题的一种方法。

其中一类便为通过直径的性质将树的问题转化至直径上。

我们先讲一道链上面的题。

我们先讲一道链上面的题。

Problem ([IOI2016] shorcut)

在一个 n 个点的链上，每个点下挂了一条长度为 d_i 的边。
现在可以添加一条边连接两个链上的点，这条边长度为 c。
现在要使得图中的直径最小。

我们先讲一道链上面的题。

Problem ([IOI2016] shorcut)

在一个 n 个点的链上，每个点下挂了一条长度为 d_i 的边。
现在可以添加一条边连接两个链上的点，这条边长度为 c。
现在要使得图中的直径最小。
$n \leqslant 1000000$。

Solution ([IOI2016] shorcut)

首先求出链上点到第一个点的距离 L_i。

Solution ([IOI2016] shorcut)

首先求出链上点到第一个点的距离 L_i。
对于两个点 $i, j(i < j)$ 初始的最短路是 $L_j - L_i + d_i + d_j$。

Solution ([IOI2016] shorcut)

首先求出链上点到第一个点的距离 L_i。

对于两个点 $i, j(i < j)$ 初始的最短路是 $L_j - L_i + d_i + d_j$。

如果加入了 $a, b(a < b)$ 这条边，那么经过这条边的最短路是 $|L_i - L_a| + |L_j - L_b| + d_i + d_j + c$。

Solution ([IOI2016] shorcut)

首先求出链上点到第一个点的距离 L_i。

对于两个点 $i, j(i < j)$ 初始的最短路是 $L_j - L_i + d_i + d_j$。

如果加入了 $a, b(a < b)$ 这条边，那么经过这条边的最短路是 $|L_i - L_a| + |L_j - L_b| + d_i + d_j + c$。

考虑二分答案为 M，那么对于满足 $L_j - L_i + d_i + d_j > M$ 时都应有 $|L_i - L_a| + |L_j - L_b| + d_i + d_j + c \leqslant M$。

Solution ([IOI2016] shorcut)

首先求出链上点到第一个点的距离 L_i。

对于两个点 $i, j(i < j)$ 初始的最短路是 $L_j - L_i + d_i + d_j$。

如果加入了 $a, b(a < b)$ 这条边，那么经过这条边的最短路是 $|L_i - L_a| + |L_j - L_b| + d_i + d_j + c$。

考虑二分答案为 M，那么对于满足 $L_j - L_i + d_i + d_j > M$ 时都应有 $|L_i - L_a| + |L_j - L_b| + d_i + d_j + c \leqslant M$。

由于是绝对值不超过一个数，所以可以将所有的却对子拆开的情况分别求最小值再求交。

Solution ([IOI2016] shorcut)

首先求出链上点到第一个点的距离 L_i。

对于两个点 $i, j(i < j)$ 初始的最短路是 $L_j - L_i + d_i + d_j$。

如果加入了 $a, b(a < b)$ 这条边，那么经过这条边的最短路是 $|L_i - L_a| + |L_j - L_b| + d_i + d_j + c$。

考虑二分答案为 M，那么对于满足 $L_j - L_i + d_i + d_j > M$ 时都应有 $|L_i - L_a| + |L_j - L_b| + d_i + d_j + c \leqslant M$。

由于是绝对值不超过一个数，所以可以将所有的却对子拆开的情况分别求最小值再求交。

考虑枚举 j 那么 i 的范围一定是将 $L_i + d_i$ 排序后连续的一段中已经出现的，所以显然可以线段树优化。但由于 $L_i + d_i$ 和 $L_i - d_i$ 都不是有序的所以似乎不能双指针。

Solution ([IOI2016] shorcut)

考虑枚举 j 那么 i 的范围一定是将 $L_i + d_i$ 排序后连续的一段中已经出现的，所以显然可以线段树优化。但由于 $L_i + d_i$ 和 $L_i - d_i$ 都不是有序的所以似乎不能双指针。

实际上，如果两个点 $i < j$ 满足 $L_i - d_i > L_j - d_j$，那么由 $L_i < L_j$ 得出 $-L_i - d_i > L_j - d_j$，从而对于 j 之后的，j 作为左边更优。

Solution ([IOI2016] shorcut)

考虑枚举 j 那么 i 的范围一定是将 $L_i + d_i$ 排序后连续的一段中已经出现的，所以显然可以线段树优化。但由于 $L_i + d_i$ 和 $L_i - d_i$ 都不是有序的所以似乎不能双指针。

实际上，如果两个点 $i < j$ 满足 $L_i - d_i > L_j - d_j$，那么由 $L_i < L_j$ 得出 $-L_i - d_i > L_j - d_j$，从而对于 j 之后的，j 作为左边更优。

于是都只需要左边的区间最值和右边的前缀最值的位置进行转移，就可以双指针了。

而后面给定 $L_a - L_b$ 和 $L_a + L_b$ 范围也是容易双指针的。

Solution ([IOI2016] shorcut)

考虑枚举 j 那么 i 的范围一定是将 $L_i + d_i$ 排序后连续的一段中已经出现的，所以显然可以线段树优化。但由于 $L_i + d_i$ 和 $L_i - d_i$ 都不是有序的所以似乎不能双指针。

实际上，如果两个点 $i < j$ 满足 $L_i - d_i > L_j - d_j$，那么由 $L_i < L_j$ 得出 $-L_i - d_i > L_j - d_j$，从而对于 j 之后的，j 作为左边更优。

于是都只需要左边的区间最值和右边的前缀最值的位置进行转移，就可以双指针了。

而后面给定 $L_a - L_b$ 和 $L_a + L_b$ 范围也是容易双指针的。

这样就做到了 $O(n)$ 判断，总时间复杂度为 $O(n \log ans)$。

现在，我们把上一题改到树上。

Problem ([CTSC2017] 网络)

有一个 n 个点的树（每条边长 l_i），现在可以添加一条长度为 L 边连接两个点，要使得图的直径最小。

现在，我们把上一题改到树上。

Problem ([CTSC2017] 网络)

有一个 n 个点的树（每条边长 l_i），现在可以添加一条长度为 L 边连接两个点，要使得图的直径最小。

数据范围：$1 \leqslant n \leqslant 100000$。

现在，我们把上一题改到树上。

Problem ([CTSC2017] 网络)

有一个 n 个点的树（每条边长 l_i），现在可以添加一条长度为 L 边连接两个点，要使得图的直径最小。

数据范围：$1 \leqslant n \leqslant 100000$。

Solution ([CTSC2017] 网络)

可以证明一定是在直径上增加边。

于是对直径上每个点求出向下的深度，就变为了上一题。

时间复杂度 $O(n \log ans)$。

Proof ([CTSC2017] 网络)

我们设增加的边为 (u, v, L)，直径端点为 S 和 T，结点 X 在直径上最近的一个点为 f_X，$\text{dist}(u, v)$ 为 u 和 v 在原树上的距离。

不妨设 $\text{dist}(f_u, S) \leqslant \text{dist}(f_v, S)$。

Proof ([CTSC2017] 网络)

我们设增加的边为 (u, v, L)，直径端点为 S 和 T，结点 X 在直径上最近的一个点为 f_X，$\text{dist}(u, v)$ 为 u 和 v 在原树上的距离。

不妨设 $\text{dist}(f_u, S) \leqslant \text{dist}(f_v, S)$。

要使这种添法有贡献，则

$\text{dist}(u, f_u) + \text{dist}(v, f_v) + L < \text{dist}(f_u, f_v)$。

Proof ([CTSC2017] 网络)

我们设增加的边为 (u, v, L)，直径端点为 S 和 T，结点 X 在直径上最近的一个点为 f_X，$\text{dist}(u, v)$ 为 u 和 v 在原树上的距离。

不妨设 $\text{dist}(f_u, S) \leqslant \text{dist}(f_v, S)$。

要使这种添法有贡献，则
$\text{dist}(u, f_u) + \text{dist}(v, f_v) + L < \text{dist}(f_u, f_v)$。

我们考虑另一种方案（下称“方案二”）: 在 f_u 和 f_v 上增加长度为 L 的边。

Proof ([CTSC2017] 网络)

我们考虑另一种方案（下称“方案二”）：在 f_u 和 f_v 上增加长度为 L 的边。

在方案二中，

- 若直径经过 f_u 和 f_v 两者，即为 S 到 T，则方案二直径为 $\operatorname{dist}(s, f_u) + L + \operatorname{dist}(f_v, t)$，而当前方案直径不低于 $\operatorname{dist}(s, f_u) + \operatorname{dist}(f_u, u) + L + \operatorname{dist}(f_v, v) + \operatorname{dist}(f_v, t)$，因此方案二在此情况下不劣于当前方案。

Proof ([CTSC2017] 网络)

我们考虑另一种方案（下称“方案二”）：在 f_u 和 f_v 上增加长度为 L 的边。

在方案二中，

- 若直径经过 f_u 和 f_v 两者，即为 S 到 T，则方案二直径为 $\text{dist}(s, f_u) + L + \text{dist}(f_v, t)$，而当前方案直径不低于 $\text{dist}(s, f_u) + \text{dist}(f_u, u) + L + \text{dist}(f_v, v) + \text{dist}(f_v, t)$，因此方案二在此情况下不劣于当前方案。
- 若直径既不经过 f_u，也不经过 f_v。由于原先直径为 S 到 T，可以发现直径端点的一端如果改成 S 或 T 可以得到一条更长的路径，因此该种情况不存在。

Proof ([CTSC2017] 网络)

在方案二中，

- 若直径经过 f_u 和 f_v 两者……
- 若直径既不经过 f_u，也不经过 f_v……
- 若直径经过 f_u 和 f_v 其中一者，不妨设经过 f_u 但不经过 f_v，设为 S 到 X。与上一种情况不存在的证明同理可知 f_X 在 S 到 T 的路径中一定在 f_u 和 f_v 之间。则方案二直径为 $\min(\text{dist}(S, f_u) + \text{dist}(f_u, X), \text{dist}(S, f_u) + L + \text{dist}(f_v, X))$。因为 $\text{dist}(f_u, f_v) > \text{dist}(u, f_u) + L + \text{dist}(v, f_v) > \text{dist}(u, f_u)$，所以 $\text{dist}(S, f_u) + \text{dist}(f_u, f_v) + \text{dist}(f_v, v) + L + \text{dist}(u, X) > \text{dist}(S, f_u) + \text{dist}(f_u, u) + \text{dist}(u, X) \geqslant \text{dist}(S, X)$。从而当前方案直径不低于 $\min(\text{dist}(S, f_u) + \text{dist}(f_u, X), \text{dist}(S, f_u) + \text{dist}(f_u, u) + L + \text{dist}(f_v, v) + \text{dist}(v, f_v) + \text{dist}(f_v, X))$。方案二在此情况下不劣于当前方案。

Proof ([CTSC2017] 网络)

在方案二中，

- 若直径经过 f_u 和 f_v 两者……
- 若直径既不经过 f_u，也不经过 f_v……
- 若直径经过 f_u 和 f_v 其中一者，不妨设经过 f_u 但不经过 f_v，设为 S 到 X。与上一种情况不存在的证明同理可知 f_X 在 S 到 T 的路径中一定在 f_u 和 f_v 之间。则方案二直径为 $\min(\text{dist}(S, f_u) + \text{dist}(f_u, X), \text{dist}(S, f_u) + L + \text{dist}(f_v, X))$。因为 $\text{dist}(f_u, f_v) > \text{dist}(u, f_u) + L + \text{dist}(v, f_v) > \text{dist}(u, f_u)$，所以 $\text{dist}(S, f_u) + \text{dist}(f_u, f_v) + \text{dist}(f_v, v) + L + \text{dist}(u, X) > \text{dist}(S, f_u) + \text{dist}(f_u, u) + \text{dist}(u, X) \geqslant \text{dist}(S, X)$。从而当前方案直径不低于 $\min(\text{dist}(S, f_u) + \text{dist}(f_u, X), \text{dist}(S, f_u) + \text{dist}(f_u, u) + L + \text{dist}(f_v, v) + \text{dist}(v, f_v) + \text{dist}(f_v, X))$。方案二在此情况下不劣于当前方案。

综上所述，在所有方案中，只有在直径上增加边才能产生贡献。

树形动规

在一棵树上设计状态的动规，由于通常是沿着树的边转移状态（因此被称为“树形动规”），基本上用树上搜索实现（有时为了减少递归常数，使用 dfs 序或拓扑序实现）。

Problem (没有上司的舞会)

某大学有 N 个职员，编号为 $1 \sim N$。他们之间有从属关系，也就是说他们的关系就像一棵以校长为根的树，父结点就是子结点的直接上司。现在有个周年庆宴会，宴会每邀请来一个职员都会增加一定的快乐指数 R_i，但是呢，如果某个职员的上司来参加舞会了，那么这个职员就无论如何也不肯来参加舞会了。所以，请你编程计算，邀请哪些职员可以使快乐指数最大，求最大的快乐指数。

Problem (没有上司的舞会)

某大学有 N 个职员，编号为 $1 \sim N$。他们之间有从属关系，也就是说他们的关系就像一棵以校长为根的树，父结点就是子结点的直接上司。现在有个周年庆宴会，宴会每邀请来一个职员都会增加一定的快乐指数 R_i，但是呢，如果某个职员的上司来参加舞会了，那么这个职员就无论如何也不肯来参加舞会了。所以，请你编程计算，邀请哪些职员可以使快乐指数最大，求最大的快乐指数。

$(n \leqslant 100000)$，原题 6000 太小了。

最大独立集

- 独立集：图 $G=(V,E)$ 的独立集 S 满足 $(S \subseteq V) \wedge (\forall u,v \in S,(u,v) \notin E)$。

最大独立集

- 独立集：图 $G = (V, E)$ 的独立集 S 满足 $(S \subseteq V) \land (\forall u, v \in S, (u, v) \notin E)$。
- 最大独立集：一个图中顶点权值和最大的独立集。

Solution (没有上司的舞会)

本题即求一棵树的最大独立集。
定义 $f[u][0/1]$ 表示结点 u 选不选

Solution (没有上司的舞会)

本题即求一棵树的最大独立集。

定义 $f[u][0/1]$ 表示结点 u 选不选

$f[u][0]=\sum_{v\in\text{son}(u)}\max(f[v][0],f[v][1])$

$f[u][1]=w[u]+\sum_{v\in\text{son}(u)}f[v][0]$

Solution (没有上司的舞会)

本题即求一棵树的最大独立集。

定义 $f[u][0/1]$ 表示结点 u 选不选

$f[u][0] = \sum_{v \in \text{son}(u)} \max(f[v][0], f[v][1])$

$f[u][1] = w[u] + \sum_{v \in \text{son}(u)} f[v][0]$

然后在 DFS 的同时 dp 下去

Problem (金明的预算方案·加强版)

在 01 背包（n 件物品，容量为 m）的基础上，再加上一些限制条件：选其中一些物品时，必须要选另外一个物品。（保证不会形成间接的自身依赖。）

Problem (金明的预算方案·加强版)

在 01 背包（n 件物品，容量为 m）的基础上，再加上一些限制条件：选其中一些物品时，必须要选另外一个物品。（保证不会形成间接的自身依赖。）

要求时间复杂度 $O(nm^2)$。

Solution (金明的预算方案·加强版)

树形 DP 下去

做 01 背包

```
void dfs(int u) {
    f[w[u]][u] = v[u];
    for (int i = head[u]; i; i = nxt[i])
        if (to[i] != rs[u]) {
            dfs(to[i]);
            for (int j = m; j; --j)
                for (int k = m; k; --k)
                    f[j + k][u] = max(f[j + k][u], f[j][u] + f[k][to[i]]);
        }
}
```

Solution (金明的预算方案·加强版)

如果形成了间接的自身依赖（[HAOI2010] 软件安装），其实可以把每一些相互依赖的物品用一个等价的物品替换（缩点），然后就转化成了这个问题。

Solution (金明的预算方案·加强版)

如果形成了间接的自身依赖（[HAOI2010] 软件安装），其实可以把每一些相互依赖的物品用一个等价的物品替换（缩点），然后就转化成了这个问题。

还有一种时间复杂度更优的方法，我们改每个结点的 dp 状态为使用在它的回溯时完成回溯的（含本身）在体积 i 的情况下能达到的最大价值。时间复杂度 $O(nm)$，在此不详细展开。

树的重心

- 树的重心：

  树的重心，是使得删去这个点后形成的连通块大小的最大值最小的点。

树的重心

- 树的重心：

  树的重心，是使得删去这个点后形成的连通块大小的最大值最小的点。

  对于一个大小为 n 的树与任意一个树中结点 c，称 c 是该树的重心当且仅当在树中删去 c 及与它关联的边后，分裂出的所有子树的大小均不超过 $\left\lfloor \frac{n}{2} \right\rfloor$。

树的重心

- 树的重心：

性质：

1. 一棵树最多有两个重心，如果有两个重心，它们必定有一条边相连。

树的重心

- 树的重心：

性质：

1. 一棵树最多有两个重心，如果有两个重心，它们必定有一条边相连。
2. 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，他们的距离和一样。

树的重心

- 树的重心：

性质：

1. 一棵树最多有两个重心，如果有两个重心，它们必定有一条边相连。
2. 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，他们的距离和一样。
3. 把两棵树通过一条边相连，新的树的重心在原来两棵树重心的连线上。

树的重心

- 树的重心：

性质：

1. 一棵树最多有两个重心，如果有两个重心，它们必定有一条边相连。
2. 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，他们的距离和一样。
3. 把两棵树通过一条边相连，新的树的重心在原来两棵树重心的连线上。
4. 一棵树添加或者删除一个结点，树的重心最多只移动一条边的位置。

Problem (树的重心)

给定一个点，求树的重心。

Problem (树的重心)

给定一个点，求树的重心。

Solution (树的重心)

按照子树大小均不超过 $\left\lfloor \frac{n}{2} \right\rfloor$，我们可以通过子树大小判断是否为重心。

Solution (树的重心)

```
void dfs(int u, int fa) {
    int maxn = 0;
    sz[u] = 1;
    for (int i = head[u]; i; i = nxt[i])
        if (!vis[to[i]] && to[i] != fa) {
            dfs(to[i], u);
            sz[u] += sz[to[i]];
            maxn = max(maxn, sz[to[i]]);
        }
    maxn = max(maxn, n - sz[u]);
    if (maxn < rt) rt = maxn, rt = u;
}
```

Problem ([USACO10MAR] Great Cow Gathering)

给你一棵树，每个点有点权，边有边权，求一个点，使得其他所有点到这个点的距离和最短，输出这个距离

Problem ([USACO10MAR] Great Cow Gathering)

给你一棵树，每个点有点权，边有边权，求一个点，使得其他所有点到这个点的距离和最短，输出这个距离

Solution ([USACO10MAR] Great Cow Gathering)

求带权树的重心，类似 dp 即可。

Problem (来源不明的题目 I)

给出一棵 n 个点的树，从根结点出发，不断往周围的点走，每次走随机选一个与之连通走。给定一个点集，走到点集中的任意一个点就停止。求期望步数。

答案保留 2 位小数

$(n \leqslant 6)$

Solution (来源不明的题目 I)

我们可以按题意模拟，注意到如果走的步数多了，概率就会变小，在精度范围内对答案基本没贡献。

Problem (来源不明的题目 II)

给出一棵 n 个点的树，从根结点出发，不断往周围的点走，每次走随机选一个与之连通走。给定一个点集，走到点集中的任意一个点就停止。求期望步数。

答案对 1004535809 取模

$(n \leqslant 300)$

Solution (来源不明的题目 II)

对于点集外的点，有 $E(u) = \sum_{v \in v(u)} \dfrac{E(v)+1}{\deg(u)}$（其中 $v(u)$ 表示 u 的出边。

Solution (来源不明的题目 II)

对于点集外的点，有 $E(u) = \sum_{v \in v(u)} \frac{E(v)+1}{\deg(u)}$（其中 $v(u)$ 表示 u 的出边。

对于点集内的点，$E(u) = 0$。

Solution (来源不明的题目 II)

对于点集外的点，有 $E(u)=\sum_{v\in v(u)}\dfrac{E(v)+1}{\deg(u)}$（其中 $v(u)$ 表示 u 的出边。

对于点集内的点，$E(u)=0$。

高斯消元即可。

Problem (来源不明的题目 III)

给出一棵 n 个点的树，从根结点出发，不断往周围的点走，每次走随机选一个与之连通走。给定一个点集，走到点集中的任意一个点就停止。求期望步数。

答案对 1004535809 取模

$(n \leqslant 1000000)$

Solution (来源不明的题目 III)

设 $E(u) = A_u \times E(\mathrm{par}(u)) + B_u$。

Solution (来源不明的题目 III)

$$\because E(u)=\frac{E(\mathrm{par}(u))}{\deg(u)}+\frac{\sum_{v\in\mathrm{son}(u)}E(v)}{\deg(u)}+1$$

$$\therefore E(u)=\frac{E(\mathrm{par}(u))}{\deg(u)}+\frac{\sum_{v\in\mathrm{son}(u)}(A_v\times E(u)+B_v)}{\deg(u)}+1$$

$$\therefore E(u)=\frac{E(\mathrm{par}(u))}{\deg(u)}+\frac{\sum_{v\in\mathrm{son}(u)}A_v}{\deg u}E(u)+\frac{\sum_{v\in\mathrm{son}(u)}B_v}{\deg(u)}+1$$

$$\therefore \deg(u)E(u)=E(\mathrm{par}(u))+\left(\sum_{v\in\mathrm{son}(u)}A_v\right)E(u)+\sum_{v\in\mathrm{son}(u)}B_v+\deg(u)$$

$$\therefore \left(\deg(u)-\sum_{v\in\mathrm{son}(u)}A_v\right)E(u)=E(\mathrm{par}(u))+\sum_{v\in\mathrm{son}(u)}B_v+\deg(u)$$

$$\therefore \begin{cases}A_u\left(\deg(u)-\sum_{v\in\mathrm{son}(u)}A_v\right)=1\\ B_u\left(\deg(u)-\sum_{v\in\mathrm{son}(u)}A_v\right)=\deg(u)+\sum_{v\in\mathrm{son}(u)}B_v\end{cases}$$

Solution (来源不明的题目 III)

$$\begin{cases} A_u\left(\deg(u)-\sum_{v\in \mathrm{son}(u)} A_v\right)=1 \\ B_u\left(\deg(u)-\sum_{v\in \mathrm{son}(u)} A_v\right)=\deg(u)+\sum_{v\in \mathrm{son}(u)} B_v \end{cases}$$

Solution (来源不明的题目 III)

$$\begin{cases} A_u\left(\deg(u)-\sum_{v\in \mathrm{son}(u)} A_v\right)=1 \\ B_u\left(\deg(u)-\sum_{v\in \mathrm{son}(u)} A_v\right)=\deg(u)+\sum_{v\in \mathrm{son}(u)} B_v \end{cases}$$

进行树形 dp：

- 叶子结点 $\begin{cases} A_u=1 \\ B_u=1 \end{cases}$ 。

Solution (来源不明的题目 III)

$$\begin{cases} A_u\left(\deg(u)-\sum_{v\in\mathrm{son}(u)}A_v\right)=1 \\ B_u\left(\deg(u)-\sum_{v\in\mathrm{son}(u)}A_v\right)=\deg(u)+\sum_{v\in\mathrm{son}(u)}B_v \end{cases}$$

进行树形 dp：

- 叶子结点 $\begin{cases} A_u=1 \\ B_u=1 \end{cases}$ 。
- 点集中的结点 $\begin{cases} A_u=0 \\ B_u=0 \end{cases}$ 。

Solution (来源不明的题目 III)

$$\begin{cases} A_u\left(\deg(u)-\sum_{v\in \mathrm{son}(u)} A_v\right)=1 \\ B_u\left(\deg(u)-\sum_{v\in \mathrm{son}(u)} A_v\right)=\deg(u)+\sum_{v\in \mathrm{son}(u)} B_v \end{cases}$$

进行树形 dp：

- 叶子结点 $\begin{cases} A_u=1 \\ B_u=1 \end{cases}$。
- 点集中的结点 $\begin{cases} A_u=0 \\ B_u=0 \end{cases}$。
- 其他结点的 A_u 和 B_u 可以通过解以上方程得到。

Solution (来源不明的题目 III)

$$\begin{cases} A_u\left(\deg(u)-\sum_{v\in \text{son}(u)} A_v\right)=1 \\ B_u\left(\deg(u)-\sum_{v\in \text{son}(u)} A_v\right)=\deg(u)+\sum_{v\in \text{son}(u)} B_v \end{cases}$$

进行树形 dp：

- 叶子结点 $\begin{cases} A_u=1 \\ B_u=1 \end{cases}$。
- 点集中的结点 $\begin{cases} A_u=0 \\ B_u=0 \end{cases}$。
- 其他结点的 A_u 和 B_u 可以通过解以上方程得到。然后我们就可以通过 A_u 和 B_u 进行树形 dp 了。

Problem ([SHOI2014] 概率充电器)

有一个 n 个结点的树 T，每条边 $e = (u, v, p)$ 有 p 的概率出现在子图 T' 中。

在 T' 中，每个点有 $q_i\%$ 的概率被标记。定义一个连通块被标记当且仅当它其中存在被标记的结点。

问期望有多少个点在被标记的连通块中。

Problem ([SHOI2014] 概率充电器)

有一个 n 个结点的树 T，每条边 $e=(u,v,p)$ 有 p 的概率出现在子图 T' 中。

在 T' 中，每个点有 $q_i\%$ 的概率被标记。定义一个连通块被标记当且仅当它其中存在被标记的结点。

问期望有多少个点在被标记的连通块中。

数据范围：$1 \leqslant n \leqslant 500000$。

Solution ([SHOI2014] 概率充电器)

由于期望是线性的，我们可以把每个结点的概率分开处理。设 $f(u)$ 为 u 结点是否在被标记的连通块中，$p(u)$ 表示 u 结点在被标记的连通块中的概率。p_u 表示 u 到 $\mathrm{par}(u)$ 的边的 p。

Solution ([SHOI2014] 概率充电器)

由于期望是线性的，我们可以把每个结点的概率分开处理。设 $f(u)$ 为 u 结点是否在被标记的连通块中，$p(u)$ 表示 u 结点在被标记的连通块中的概率。p_u 表示 u 到 $\mathrm{par}(u)$ 的边的 p。

$$E\left(\sum_{i=1}^{n} f(u)\right) = \sum_{i=1}^{n} E(f(u)) = \sum_{i=1}^{n} p(u)$$

Solution ([SHOI2014] 概率充电器)

由于期望是线性的，我们可以把每个结点的概率分开处理。设 $f(u)$ 为 u 结点是否在被标记的连通块中，$p(u)$ 表示 u 结点在被标记的连通块中的概率。p_u 表示 u 到 $\operatorname{par}(u)$ 的边的 p。

$$E\left(\sum_{i=1}^{n} f(u)\right) = \sum_{i=1}^{n} E(f(u)) = \sum_{i=1}^{n} p(u)$$

因此我们只需要求出每个结点的 $p(u)$ 即可。

Solution ([SHOI2014] 概率充电器)

由于期望是线性的，我们可以把每个结点的概率分开处理。设 $f(u)$ 为 u 结点是否在被标记的连通块中，$p(u)$ 表示 u 结点在被标记的连通块中的概率。p_u 表示 u 到 $\operatorname{par}(u)$ 的边的 p。

$$E\left(\sum_{i=1}^{n} f(u)\right) = \sum_{i=1}^{n} E(f(u)) = \sum_{i=1}^{n} p(u)$$

因此我们只需要求出每个结点的 $p(u)$ 即可。
考虑反选，设 $g(u)$ 为 $1 - p(u)$，但是因为事件不独立难以设计状态。

Solution ([SHOI2014] 概率充电器)

由于期望是线性的，我们可以把每个结点的概率分开处理。设 $f(u)$ 为 u 结点是否在被标记的连通块中，$p(u)$ 表示 u 结点在被标记的连通块中的概率。p_u 表示 u 到 $\text{par}(u)$ 的边的 p。

$$E\left(\sum_{i=1}^{n} f(u)\right) = \sum_{i=1}^{n} E(f(u)) = \sum_{i=1}^{n} p(u)$$

因此我们只需要求出每个结点的 $p(u)$ 即可。

考虑反选，设 $g(u)$ 为 $1-p(u)$，但是因为事件不独立难以设计状态。

重新设计状态，令 $h(u)$ 表示 u 因它自己及它的子树中同一连通块的结点未打上标记的概率，即忽略所有 u 子树外的点后 u 结点处在被标记的连通块的概率。这样每个结点的儿子的子树的事件都相互独立，可以直接通过乘积来得到两者同时发生的概率。

Solution ([SHOI2014] 概率充电器)

令 $h(u)$ 表示 u 因它自己及它的子树中同一连通块的结点未打上标记的概率，即忽略所有 u 子树外的点后 u 结点处在被标记的连通块的概率。

注意到，如果它的儿子因自己或自己其他儿子的子树中结点打上标记而在一个被标记的连通块中，那么 u 即使忽略该儿子的子树，肯定也会在一个被标记的连通块中。

Solution ([SHOI2014] 概率充电器)

令 $h(u)$ 表示 u 因它自己及它的子树中同一连通块的结点未打上标记的概率，即忽略所有 u 子树外的点后 u 结点处在被标记的连通块的概率。

注意到，如果它的儿子因自己或自己其他儿子的子树中结点打上标记而在一个被标记的连通块中，那么 u 即使忽略该儿子的子树，肯定也会在一个被标记的连通块中。

因此，我们可以写出 $h(u)$ 的递推式。

$$h(u) = (1 - q_u\%) \prod_{v \in \mathrm{son}(u)} (p_v h(v) - p_v + 1)$$

Solution ([SHOI2014] 概率充电器)

令 $h(u)$ 表示 u 因它自己及它的子树中同一连通块的结点未打上标记的概率，即忽略所有 u 子树外的点后 u 结点处在被标记的连通块的概率。

注意到，如果它的儿子因自己或自己其他儿子的子树中结点打上标记而在一个被标记的连通块中，那么 u 即使忽略该儿子的子树，肯定也会在一个被标记的连通块中。

因此，我们可以写出 $h(u)$ 的递推式。

$$h(u) = (1 - q_u\%) \prod_{v \in \mathrm{son}(u)} (p_v h(v) - p_v + 1)$$

我们发现根的 g 值就是 h 值。

Solution ([SHOI2014] 概率充电器)

令 $h(u)$ 表示 u 因它自己及它的子树中同一连通块的结点未打上标记的概率，即忽略所有 u 子树外的点后 u 结点处在被标记的连通块的概率。

注意到，如果它的儿子因自己或自己其他儿子的子树中结点打上标记而在一个被标记的连通块中，那么 u 即使忽略该儿子的子树，肯定也会在一个被标记的连通块中。

因此，我们可以写出 $h(u)$ 的递推式。

$$h(u) = (1 - q_u\%) \prod_{v \in \mathrm{son}(u)} (p_v h(v) - p_v + 1)$$

我们发现根的 g 值就是 h 值。

考虑父亲结点连通块被除自己子树以外结点打上标记的概率 $\dfrac{g(\mathrm{par}(u))}{1 - p_u + p_u h(u)}$。

那么 $g(u) = h(u)\left(1 - p_u + p_u \dfrac{g(\mathrm{par}(u))}{1 - p_u + p_u h(u)}\right)$。

Solution ([SHOI2014] 概率充电器)

令 $h(u)$ 表示 u 因它自己及它的子树中同一连通块的结点未打上标记的概率，即忽略所有 u 子树外的点后 u 结点处在被标记的连通块的概率。

注意到，如果它的儿子因自己或自己其他儿子的子树中结点打上标记而在一个被标记的连通块中，那么 u 即使忽略该儿子的子树，肯定也会在一个被标记的连通块中。

因此，我们可以写出 $h(u)$ 的递推式。

$$h(u) = (1 - q_u\%) \prod_{v \in \mathrm{son}(u)} (p_v h(v) - p_v + 1)$$

我们发现根的 g 值就是 h 值。

考虑父亲结点连通块被除自己子树以外结点打上标记的概率 $\dfrac{g(\mathrm{par}(u))}{1 - p_u + p_u h(u)}$。

那么 $g(u) = h(u)\left(1 - p_u + p_u \dfrac{g(\mathrm{par}(u))}{1 - p_u + p_u h(u)}\right)$。

那么我们可以通过两遍 dfs 解决。

Solution ([SHOI2014] 概率充电器)

令 $h(u)$ 表示 u 因它自己及它的子树中同一连通块的结点未打上标记的概率，即忽略所有 u 子树外的点后 u 结点处在被标记的连通块的概率。

注意到，如果它的儿子因自己或自己其他儿子的子树中结点打上标记而在一个被标记的连通块中，那么 u 即使忽略该儿子的子树，肯定也会在一个被标记的连通块中。

因此，我们可以写出 $h(u)$ 的递推式。

$$h(u) = (1 - q_u\%) \prod_{v \in \mathrm{son}(u)} (p_v h(v) - p_v + 1)$$

我们发现根的 g 值就是 h 值。

考虑父亲结点连通块被除自己子树以外结点打上标记的概率 $\dfrac{g(\mathrm{par}(u))}{1 - p_u + p_u h(u)}$。

那么 $g(u) = h(u) \left(1 - p_u + p_u \dfrac{g(\mathrm{par}(u))}{1 - p_u + p_u h(u)}\right)$。

那么我们可以通过两遍 dfs 解决。

其实本题用到了换根 dp 的思想。

Problem ([PKUWC2018] 随机游走)

给出一棵 n 个点的树和每次询问一个点集 S，给定一个点进行随机游走，求走过点集中每一个点需要期望多少次。q 组询问。

Problem ([PKUWC2018] 随机游走)

给出一棵 n 个点的树和每次询问一个点集 S，给定一个点进行随机游走，求走过点集中每一个点需要期望多少次。q 组询问。

数据范围：$1 \leqslant n \leqslant 18, 1 \leqslant q \leqslant 5000$。

Problem ([PKUWC2018] 随机游走)

给出一棵 n 个点的树和每次询问一个点集 S，给定一个点进行随机游走，求走过点集中每一个点需要期望多少次。q 组询问。

数据范围：$1 \leqslant n \leqslant 18, 1 \leqslant q \leqslant 5000$。

Solution ([PKUWC2018] 随机游走)

我们发现全部走过并不一定很好处理，考虑到全部走过的期望次数即走到点集中点的次数最大值的期望，最小值即走到一个点我们已经会了，于是可以min-max 容斥。

Problem ([PKUWC2018] 随机游走)

给出一棵 n 个点的树和每次询问一个点集 S，给定一个点进行随机游走，求走过点集中每一个点需要期望多少次。q 组询问。

数据范围：$1 \leqslant n \leqslant 18, 1 \leqslant q \leqslant 5000$。

Solution ([PKUWC2018] 随机游走)

我们发现全部走过并不一定很好处理，考虑到全部走过的期望次数即走到点集中点的次数最大值的期望，最小值即走到一个点我们已经会了，于是可以min-max 容斥。

然后我们就可以用前面的方法 $\Theta(n2^n \log n)$ 求出所有 T 的 $E(\min(T))$，然后用高维前缀和等即可完成所有的 $E(\max(S))$ 的预处理。

Problem ([PKUWC2018] 随机游走)

给出一棵 n 个点的树和每次询问一个点集 S，给定一个点进行随机游走，求走过点集中每一个点需要期望多少次。q 组询问。

数据范围：$1 \leqslant n \leqslant 18, 1 \leqslant q \leqslant 5000$。

Solution ([PKUWC2018] 随机游走)

我们发现全部走过并不一定很好处理，考虑到全部走过的期望次数即走到点集中点的次数最大值的期望，最小值即走到一个点我们已经会了，于是可以min-max 容斥。

然后我们就可以用前面的方法 $\Theta(n2^n \log n)$ 求出所有 T 的 $E(\min(T))$，然后用高维前缀和等即可完成所有的 $E(\max(S))$ 的预处理。

查询直接输出答案即可。

Problem (子树直径问题)

给定一棵大小为 n 的以 1 为根的有根树，求以每个结点为根的子树的直径。

Problem (子树直径问题)

给定一棵大小为 n 的以 1 为根的有根树，求以每个结点为根的子树的直径。$n \leqslant 1000000$。

Problem (子树直径问题)

给定一棵大小为 n 的以 1 为根的有根树，求以每个结点为根的子树的直径。$n \leqslant 1000000$。

Solution (子树直径问题)

直接做树形 dp，可以求出以每个顶点为 LCA 的最长链的长度。

Problem (子树直径问题)

给定一棵大小为 n 的以 1 为根的有根树，求以每个结点为根的子树的直径。
$n \leqslant 1000000$。

Solution (子树直径问题)

直接做树形 dp，可以求出以每个顶点为 LCA 的最长链的长度。
再对每个点求子树最大值即可。

Problem (子树直径问题)

给定一棵大小为 n 的以 1 为根的有根树，求以每个结点为根的子树的直径。
$n \leqslant 1000000$。

Solution (子树直径问题)

直接做树形 dp，可以求出以每个顶点为 LCA 的最长链的长度。
再对每个点求子树最大值即可。
时间复杂度是 $O(n)$ 的。

Problem ([NOIP2018] 赛道修建)

给定一棵 n 个点的树，现在找 m 条边不交的链使得最短链最长。

Problem ([NOIP2018] 赛道修建)

给定一棵 n 个点的树，现在找 m 条边不交的链使得最短链最长。

$n \leqslant 50000$。

Solution ([NOIP2018] 赛道修建)

首先二分答案，考虑如何判断。

Solution ([NOIP2018] 赛道修建)

首先二分答案，考虑如何判断。

对于每个点，每个儿子可能会给出一些向上的链。

首先是最大化这个结点处新增的链的数量，然后是最大化向上连出的链长。

Solution ([NOIP2018] 赛道修建)

首先二分答案，考虑如何判断。

对于每个点，每个儿子可能会给出一些向上的链。

首先是最大化这个结点处新增的链的数量，然后是最大化向上连出的链长。

首先先对这些链排序，然后再二分向上连出的链。这时可以新增的链的数量容易贪心得到。

Solution ([NOIP2018] 赛道修建)

首先二分答案，考虑如何判断。

对于每个点，每个儿子可能会给出一些向上的链。

首先是最大化这个结点处新增的链的数量，然后是最大化向上连出的链长。

首先先对这些链排序，然后再二分向上连出的链。这时可以新增的链的数量容易贪心得到。

时间复杂度 $O(n \log n \log w_i)$。

Solution ([NOIP2018] 赛道修建)

有一种时间复杂度更为优秀的做法。

Solution ([NOIP2018] 赛道修建)

有一种时间复杂度更为优秀的做法。

依然二分答案 x 并树形 dp，在每个结点 u 依旧先最大化这个结点处新增的链的数量，然后最大化向上的链长 f_u。

Solution ([NOIP2018] 赛道修建)

有一种时间复杂度更为优秀的做法。

依然二分答案 x 并树形 dp，在每个结点 u 依旧先最大化这个结点处新增的链的数量，然后最大化向上的链长 f_u。

我们考虑贪心。

Solution ([NOIP2018] 赛道修建)

有一种时间复杂度更为优秀的做法。

依然二分答案 x 并树形 dp，在每个结点 u 依旧先最大化这个结点处新增的链的数量，然后最大化向上的链长 f_u。

我们考虑贪心。

我们先 $\Theta(f(n))$ 对每个儿子 v 的 $f_v + w(u, v)$（$w(u, v)$ 表示 u 到 v 的边长）进行排序。（显然 $f(n) = \Omega(n)$，$\Theta(n)$ 如基排等非比较排序。）

Solution ([NOIP2018] 赛道修建)

有一种时间复杂度更为优秀的做法。

依然二分答案 x 并树形 dp，在每个结点 u 依旧先最大化这个结点处新增的链的数量，然后最大化向上的链长 f_u。

我们考虑贪心。

我们先 $\Theta(f(n))$ 对每个儿子 v 的 $f_v + w(u, v)$（$w(u, v)$ 表示 u 到 v 的边长）进行排序。（显然 $f(n) = \Omega(n)$，$\Theta(n)$ 如基排等非比较排序。）

显然不低于 x 的链可以直接成链。

Solution ([NOIP2018] 赛道修建)

有一种时间复杂度更为优秀的做法。

依然二分答案 x 并树形 dp，在每个结点 u 依旧先最大化这个结点处新增的链的数量，然后最大化向上的链长 f_u。

我们考虑贪心。

我们先 $\Theta(f(n))$ 对每个儿子 v 的 $f_v + w(u,v)$（$w(u,v)$ 表示 u 到 v 的边长）进行排序。（显然 $f(n) = \Omega(n)$，$\Theta(n)$ 如基排等非比较排序。）

显然不低于 x 的链可以直接成链。

对于剩下的数 $\{a_i\}$，我们分成 $< \dfrac{x}{2}$ 和 $\geqslant \dfrac{x}{2}$ 两类。

Solution ([NOIP2018] 赛道修建)

有一种时间复杂度更为优秀的做法。

依然二分答案 x 并树形 dp，在每个结点 u 依旧先最大化这个结点处新增的链的数量，然后最大化向上的链长 f_u。

我们考虑贪心。

我们先 $\Theta(f(n))$ 对每个儿子 v 的 $f_v + w(u,v)$（$w(u,v)$ 表示 u 到 v 的边长）进行排序。（显然 $f(n) = \Omega(n)$，$\Theta(n)$ 如基排等非比较排序。）

显然不低于 x 的链可以直接成链。

对于剩下的数 $\{a_i\}$，我们分成 $< \frac{x}{2}$ 和 $\geqslant \frac{x}{2}$ 两类。

我们先考虑最大化这个结点处新增的链的数量。

Solution ([NOIP2018] 赛道修建)

有一种时间复杂度更为优秀的做法。

依然二分答案 x 并树形 dp，在每个结点 u 依旧先最大化这个结点处新增的链的数量，然后最大化向上的链长 f_u。

我们考虑贪心。

我们先 $\Theta(f(n))$ 对每个儿子 v 的 $f_v + w(u,v)$（$w(u,v)$ 表示 u 到 v 的边长）进行排序。（显然 $f(n) = \Omega(n)$，$\Theta(n)$ 如基排等非比较排序。）

显然不低于 x 的链可以直接成链。

对于剩下的数 $\{a_i\}$，我们分成 $< \dfrac{x}{2}$ 和 $\geqslant \dfrac{x}{2}$ 两类。

我们先考虑最大化这个结点处新增的链的数量。

我们设 a_r 为当前 $< \dfrac{x}{2}$ 的数中最大的一个，a_l 为 $\geqslant \dfrac{x}{2}$ 的数中最小的一个。

Solution ([NOIP2018] 赛道修建)

有一种时间复杂度更为优秀的做法。

依然二分答案 x 并树形 dp，在每个结点 u 依旧先最大化这个结点处新增的链的数量，然后最大化向上的链长 f_u。

我们考虑贪心。

我们先 $\Theta(f(n))$ 对每个儿子 v 的 $f_v + w(u,v)$（$w(u,v)$ 表示 u 到 v 的边长）进行排序。（显然 $f(n) = \Omega(n)$，$\Theta(n)$ 如基排等非比较排序。）

显然不低于 x 的链可以直接成链。

对于剩下的数 $\{a_i\}$，我们分成 $< \dfrac{x}{2}$ 和 $\geqslant \dfrac{x}{2}$ 两类。

我们先考虑最大化这个结点处新增的链的数量。

我们设 a_r 为当前 $< \dfrac{x}{2}$ 的数中最大的一个，a_l 为 $\geqslant \dfrac{x}{2}$ 的数中最小的一个。

$\forall k < r$，如果我们最终选了 a_k 而没有选 a_r，我们把 a_k 换成 a_r 不会对答案产生任何影响，因此我们可以优先选 a_r。

Solution ([NOIP2018] 赛道修建)

我们设 a_r 为当前 $< \frac{x}{2}$ 的数中最大的一个，a_l 为 $\geqslant \frac{x}{2}$ 的数中最小的一个。已证我们可以优先选 a_r。

若存在 (x_1, y_1, x_2, y_2) 满足 $x_1 < x_2 < y_1 < y_2$ 且 a_{x_1} 与 a_{y_1} 配对，a_{x_2} 与 a_{y_2} 配对，则我们改成 a_{x_1} 与 a_{y_2} 配对，a_{x_2} 与 a_{y_1} 配对不会对答案产生任何影响，因此我们对于 a_r，优先取小的 a_l。由于 a_r 单调递减，因此 l 单调递增。

Solution ([NOIP2018] 赛道修建)

我们设 a_r 为当前 $< \frac{x}{2}$ 的数中最大的一个，a_l 为 $\geqslant \frac{x}{2}$ 的数中最小的一个。已证我们可以优先选 a_r。

若存在 (x_1, y_1, x_2, y_2) 满足 $x_1 < x_2 < y_1 < y_2$ 且 a_{x_1} 与 a_{y_1} 配对，a_{x_2} 与 a_{y_2} 配对，则我们改成 a_{x_1} 与 a_{y_2} 配对，a_{x_2} 与 a_{y_1} 配对不会对答案产生任何影响，因此我们对于 a_r，优先取小的 a_l。由于 a_r 单调递减，因此 l 单调递增。

剩余的 $\geqslant \frac{x}{2}$ 的数，可以两两配对构成一条链。

Solution ([NOIP2018] 赛道修建)

我们设 a_r 为当前 $< \frac{x}{2}$ 的数中最大的一个，a_l 为 $\geqslant \frac{x}{2}$ 的数中最小的一个。已证我们可以优先选 a_r。

若存在 (x_1, y_1, x_2, y_2) 满足 $x_1 < x_2 < y_1 < y_2$ 且 a_{x_1} 与 a_{y_1} 配对，a_{x_2} 与 a_{y_2} 配对，则我们改成 a_{x_1} 与 a_{y_2} 配对，a_{x_2} 与 a_{y_1} 配对不会对答案产生任何影响，因此我们对于 a_r，优先取小的 a_l。由于 a_r 单调递减，因此 l 单调递增。

剩余的 $\geqslant \frac{x}{2}$ 的数，可以两两配对构成一条链。

如果有剩余的 $\geqslant \frac{x}{2}$ 的，那么 f_u 选两两配对前剩余最大的即可。

Solution ([NOIP2018] 赛道修建)

我们设 a_r 为当前 $< \frac{x}{2}$ 的数中最大的一个，a_l 为 $\geqslant \frac{x}{2}$ 的数中最小的一个。已证我们可以优先选 a_r。

若存在 (x_1, y_1, x_2, y_2) 满足 $x_1 < x_2 < y_1 < y_2$ 且 a_{x_1} 与 a_{y_1} 配对，a_{x_2} 与 a_{y_2} 配对，则我们改成 a_{x_1} 与 a_{y_2} 配对，a_{x_2} 与 a_{y_1} 配对不会对答案产生任何影响，因此我们对于 a_r，优先取小的 a_l。由于 a_r 单调递减，因此 l 单调递增。

剩余的 $\geqslant \frac{x}{2}$ 的数，可以两两配对构成一条链。

如果有剩余的 $\geqslant \frac{x}{2}$ 的，那么 f_u 选两两配对前剩余最大的即可。

如果没有剩余，我们考虑再次进行贪心。

Solution ([NOIP2018] 赛道修建)

我们设 a_r 为当前 $< \frac{x}{2}$ 的数中最大的一个，a_l 为 $\geqslant \frac{x}{2}$ 的数中最小的一个。已证我们可以优先选 a_r。

若存在 (x_1, y_1, x_2, y_2) 满足 $x_1 < x_2 < y_1 < y_2$ 且 a_{x_1} 与 a_{y_1} 配对，a_{x_2} 与 a_{y_2} 配对，则我们改成 a_{x_1} 与 a_{y_2} 配对，a_{x_2} 与 a_{y_1} 配对不会对答案产生任何影响，因此我们对于 a_r，优先取小的 a_l。由于 a_r 单调递减，因此 l 单调递增。

剩余的 $\geqslant \frac{x}{2}$ 的数，可以两两配对构成一条链。

如果有剩余的 $\geqslant \frac{x}{2}$ 的，那么 f_u 选两两配对前剩余最大的即可。

如果没有剩余，我们考虑再次进行贪心。

我们对于每一个 a_l，都选择一个最小的 a_p 与之匹配，若我们枚举 l 单调递减，p 则单调递增。

Solution ([NOIP2018] 赛道修建)

我们设 a_r 为当前 $< \frac{x}{2}$ 的数中最大的一个，a_l 为 $\geqslant \frac{x}{2}$ 的数中最小的一个。已证我们可以优先选 a_r。

若存在 (x_1, y_1, x_2, y_2) 满足 $x_1 < x_2 < y_1 < y_2$ 且 a_{x_1} 与 a_{y_1} 配对，a_{x_2} 与 a_{y_2} 配对，则我们改成 a_{x_1} 与 a_{y_2} 配对，a_{x_2} 与 a_{y_1} 配对不会对答案产生任何影响，因此我们对于 a_r，优先取小的 a_l。由于 a_r 单调递减，因此 l 单调递增。

剩余的 $\geqslant \frac{x}{2}$ 的数，可以两两配对构成一条链。

如果有剩余的 $\geqslant \frac{x}{2}$ 的，那么 f_u 选两两配对前剩余最大的即可。

如果没有剩余，我们考虑再次进行贪心。

我们对于每一个 a_l，都选择一个最小的 a_p 与之匹配，若我们枚举 l 单调递减，p 则单调递增。

选一个剩余最大的即可，如果没有，则 $f_u = 0$。

Solution ([NOIP2018] 赛道修建)

我们设 a_r 为当前 $< \frac{x}{2}$ 的数中最大的一个，a_l 为 $\geqslant \frac{x}{2}$ 的数中最小的一个。已证我们可以优先选 a_r。

若存在 (x_1, y_1, x_2, y_2) 满足 $x_1 < x_2 < y_1 < y_2$ 且 a_{x_1} 与 a_{y_1} 配对，a_{x_2} 与 a_{y_2} 配对，则我们改成 a_{x_1} 与 a_{y_2} 配对，a_{x_2} 与 a_{y_1} 配对不会对答案产生任何影响，因此我们对于 a_r，优先取小的 a_l。由于 a_r 单调递减，因此 l 单调递增。

剩余的 $\geqslant \frac{x}{2}$ 的数，可以两两配对构成一条链。

如果有剩余的 $\geqslant \frac{x}{2}$ 的，那么 f_u 选两两配对前剩余最大的即可。

如果没有剩余，我们考虑再次进行贪心。

我们对于每一个 a_l，都选择一个最小的 a_p 与之匹配，若我们枚举 l 单调递减，p 则单调递增。

选一个剩余最大的即可，如果没有，则 $f_u = 0$。

总时间复杂度 $\Theta(f(n) \log w_i)$。

Problem (CF1039D You Are Given a Tree)

给你一棵 $n(n \leqslant 100000)$ 个点的树，一个简单路径的集合 S_k 被称为 k 合法当且仅当：树的每个结点至多属于其中一条路径，且每条路径恰好包含 k 个点。对于 $k \in [1, n]$，$|S_k|$ 的最大值

Problem (CF1039D You Are Given a Tree)

给你一棵 $n(n \leqslant 100000)$ 个点的树，一个简单路径的集合 S_k 被称为 k 合法当且仅当：树的每个结点至多属于其中一条路径，且每条路径恰好包含 k 个点。对于 $k \in [1, n]$，$|S_k|$ 的最大值

Solution (CF1039D You Are Given a Tree)

上一题方法的 $\Theta\left(n^2\right)$ 的 dp 并不足以通过此题。

Solution (CF1039D You Are Given a Tree)

我们考虑分段，

- 对于 $1 \leqslant k \leqslant s$，暴力 dp，总时间复杂度 $\Theta(sn)$。

Solution (CF1039D You Are Given a Tree)

我们考虑分段，

- 对于 $1 \leqslant k \leqslant s$，暴力 dp，总时间复杂度 $\Theta(sn)$。
- 对于 $s < k \leqslant n$，注意到 dp 值 $< \frac{n}{s}$，我们可以对于每个 dp 值通过二分处理出左端点和右端点，总时间复杂度 $\Theta\left(\frac{n \log n}{s}\right)$。

Solution (CF1039D You Are Given a Tree)

我们考虑分段，

- 对于 $1 \leqslant k \leqslant s$，暴力 dp，总时间复杂度 $\Theta(sn)$。
- 对于 $s < k \leqslant n$，注意到 dp 值 $< \dfrac{n}{s}$，我们可以对于每个 dp 值通过二分处理出左端点和右端点，总时间复杂度 $\Theta\left(\dfrac{n\log n}{s}\right)$。

当我们取 $s = \Theta\left(\sqrt{n\log n}\right)$ 时，取得最优时间复杂度 $\Theta\left(n\sqrt{n\log n}\right)$。

Solution (CF1039D You Are Given a Tree)

我们考虑分段，

- 对于 $1 \leqslant k \leqslant s$，暴力 dp，总时间复杂度 $\Theta(sn)$。
- 对于 $s < k \leqslant n$，注意到 dp 值 $< \dfrac{n}{s}$，我们可以对于每个 dp 值通过二分处理出左端点和右端点，总时间复杂度 $\Theta\left(\dfrac{n\log n}{s}\right)$。

当我们取 $s = \Theta\left(\sqrt{n\log n}\right)$ 时，取得最优时间复杂度 $\Theta\left(n\sqrt{n\log n}\right)$。

事实上，由于第二段常数较小，s 中 $\sqrt{n\log n}$ 的常数同样较小，需根据实际代码调整。

倍增是一种处理一类静态问题的方法。

倍增是一种处理一类静态问题的方法。

对于树上的很多 dp，我们也可以类似序列上的倍增处理。

倍增求 LCA

我们可以先预处理出每个结点的深度和 2^k 级祖先。

倍增求 LCA

我们可以先预处理出每个结点的深度和 2^k 级祖先。

然后对于每次查询，进行如下操作：

倍增求 LCA

我们可以先预处理出每个结点的深度和 2^k 级祖先。

然后对于每次查询，进行如下操作：

1. 将深度深的一个结点通过倍增找到和另一个结点深度相同的祖先，可以发现 LCA 不变。

倍增求 LCA

我们可以先预处理出每个结点的深度和 2^k 级祖先。

然后对于每次查询，进行如下操作：

1. 将深度深的一个结点通过倍增找到和另一个结点深度相同的祖先，可以发现 LCA 不变。
2. 通过倍增找到最远非公共祖先（如果 2^k 级祖先不同则改求它们的 LCA，正确性显然）。

倍增求 LCA

我们可以先预处理出每个结点的深度和 2^k 级祖先。

然后对于每次查询，进行如下操作：

1. 将深度深的一个结点通过倍增找到和另一个结点深度相同的祖先，可以发现 LCA 不变。
2. 通过倍增找到最远非公共祖先（如果 2^k 级祖先不同则改求它们的 LCA，正确性显然）。
3. LCA 即最后的兄弟结点（就是初始两个结点的最远非公共祖先）的父亲。

倍增求 LCA

我们可以先预处理出每个结点的深度和 2^k 级祖先。

然后对于每次查询，进行如下操作：

1. 将深度深的一个结点通过倍增找到和另一个结点深度相同的祖先，可以发现 LCA 不变。
2. 通过倍增找到最远非公共祖先（如果 2^k 级祖先不同则改求它们的 LCA，正确性显然）。
3. LCA 即最后的兄弟结点（就是初始两个结点的最远非公共祖先）的父亲。

时间复杂度 $\Theta(n) - \Theta(\log n)$。

树上倍增 dp

```
void dfs(int u) {
    for (int i = 1; i < M; i++) fa[u][i] = fa[fa[u][i - 1]][i - 1];
    for (int i = head[u]; i; i = e[i].nxt) {
        int v = e[i].to;
        if (v != *fa[u]) {
            dep[v] = dep[u] + 1;
            *fa[v] = u;
            dfs(v);
        }
    }
}
inline int LCA(int x, int y) {
    if (dep[x] < dep[y]) swap(x, y);
    for (int i = dep[x] - dep[y]; i; i &= i - 1) x = fa[x][__builtin_ctz(i)];
    if (x == y) return x;
    for (int i = M - 1; ~i; i--) if (fa[x][i] != fa[y][i]) x = fa[x][i], y = fa[y][i];
    return fa(x);
}
```

Problem ([ZJOI2012] 灾难)

给出一个有向无环图，对于结点 s,t，t 依赖于 s 当且仅当对于任意路径 $u_1,u_2,\dots,u_k,t$（其中 u_1 无入边），满足 $\exists 1 \leqslant i \leqslant k, u_i = s$。对于每一个点求出多少个点依赖于它。

（注：上段所提图为原题中食物网的反向图。）

Problem ([ZJOI2012] 灾难)

给出一个有向无环图，对于结点 s, t，t 依赖于 s 当且仅当对于任意路径 $u_1, u_2, \dots, u_k, t$（其中 u_1 无入边），满足 $\exists 1 \leqslant i \leqslant k, u_i = s$。对于每一个点求出多少个点依赖于它。

（注：上段所提图为原题中食物网的反向图。）

$1 \leqslant n \leqslant 65534$，图中边数不超过 10^6。

Problem ([ZJOI2012] 灾难)

给出一个有向无环图，对于结点 s, t，t 依赖于 s 当且仅当对于任意路径 $u_1, u_2, \dots, u_k, t$（其中 u_1 无入边），满足 $\exists 1 \leqslant i \leqslant k, u_i = s$。对于每一个点求出多少个点依赖于它。

（注：上段所提图为原题中食物网的反向图。）

$1 \leqslant n \leqslant 65534$，图中边数不超过 10^6。

Solution ([ZJOI2012] 灾难)

为了避免有的点没有依赖的细节处理，我们可以建一个超级源点，将它往所有的点都连一条边。容易发现依赖于每个原有点的点集都没有变，但每个点都依赖于这个超级源点。

Problem ([ZJOI2012] 灾难)

给出一个有向无环图，对于结点 s, t，t 依赖于 s 当且仅当对于任意路径 $u_1, u_2, \ldots, u_k, t$（其中 u_1 无入边），满足 $\exists 1 \leqslant i \leqslant k, u_i = s$。对于每一个点求出多少个点依赖于它。

（注：上段所提图为原题中食物网的反向图。）

$1 \leqslant n \leqslant 65534$，图中边数不超过 10^6。

Solution ([ZJOI2012] 灾难)

为了避免有的点没有依赖的细节处理，我们可以建一个超级源点，将它往所有的点都连一条边。容易发现依赖于每个原有点的点集都没有变，但每个点都依赖于这个超级源点。

对于每一个 t，若它依赖于 $s_1, s_2, \ldots, s_k$，则对于所有的路径，都可表示为 $\ldots, s_1, \ldots, s_2, \ldots, \ldots, \ldots, s_k, \ldots, t$。

Problem ([ZJOI2012] 灾难)

给出一个有向无环图，对于结点 s, t，t 依赖于 s 当且仅当对于任意路径 $u_1, u_2, \dots, u_k, t$（其中 u_1 无入边），满足 $\exists 1 \leqslant i \leqslant k, u_i = s$。对于每一个点求出多少个点依赖于它。

（注：上段所提图为原题中食物网的反向图。）

$1 \leqslant n \leqslant 65534$，图中边数不超过 10^6。

Solution ([ZJOI2012] 灾难)

为了避免有的点没有依赖的细节处理，我们可以建一个超级源点，将它往所有的点都连一条边。容易发现依赖于每个原有点的点集都没有变，但每个点都依赖于这个超级源点。

对于每一个 t，若它依赖于 $s_1, s_2, \dots, s_k$，则对于所有的路径，都可表示为 $\dots, s_1, \dots, s_2, \dots, \dots, \dots, s_k, \dots, t$。

可以发现，s_k 依赖且仅依赖于 $s_1, s_2, \dots, s_{k-1}$。

Solution ([ZJOI2012] 灾难)

我们只用对每个点找到最后一个依赖的点即可，我们可以建成一棵树（超级源点为根），每个点所依赖的点为它的所有祖先结点（不包含它本身）。

Solution ([ZJOI2012] 灾难)

我们只用对每个点找到最后一个依赖的点即可，我们可以建成一棵树（超级源点为根），每个点所依赖的点为它的所有祖先结点（不包含它本身）。

按照拓扑序遍历每一个点，并求出它的最后一个依赖的点：

Solution ([ZJOI2012] 灾难)

我们只用对每个点找到最后一个依赖的点即可，我们可以建成一棵树（超级源点为根），每个点所依赖的点为它的所有祖先结点（不包含它本身）。

按照拓扑序遍历每一个点，并求出它的最后一个依赖的点：

- 只有一条入边：它最后一个依赖的点为那条入边的起始点。

Solution ([ZJOI2012] 灾难)

我们只用对每个点找到最后一个依赖的点即可，我们可以建成一棵树（超级源点为根），每个点所依赖的点为它的所有祖先结点（不包含它本身）。

按照拓扑序遍历每一个点，并求出它的最后一个依赖的点：

- 只有一条入边：它最后一个依赖的点为那条入边的起始点。
- 不止一条入边：它最后一个依赖的点为所有入边的起始点所共同依赖的点中的最后一个，即那些入边的起始点的最后一个依赖的点的树上最近公共祖先。

Solution ([ZJOI2012] 灾难)

我们只用对每个点找到最后一个依赖的点即可，我们可以建成一棵树（超级源点为根），每个点所依赖的点为它的所有祖先结点（不包含它本身）。

按照拓扑序遍历每一个点，并求出它的最后一个依赖的点：

- 只有一条入边：它最后一个依赖的点为那条入边的起始点。
- 不止一条入边：它最后一个依赖的点为所有入边的起始点所共同依赖的点中的最后一个，即那些入边的起始点的最后一个依赖的点的树上最近公共祖先。

所求的就是子树大小减 1（自身结点）。

Solution ([ZJOI2012] 灾难)

我们只用对每个点找到最后一个依赖的点即可，我们可以建成一棵树（超级源点为根），每个点所依赖的点为它的所有祖先结点（不包含它本身）。

按照拓扑序遍历每一个点，并求出它的最后一个依赖的点：

- 只有一条入边：它最后一个依赖的点为那条入边的起始点。
- 不止一条入边：它最后一个依赖的点为所有入边的起始点所共同依赖的点中的最后一个，即那些入边的起始点的最后一个依赖的点的树上最近公共祖先。

所求的就是子树大小减 1（自身结点）。

时间复杂度 $\Theta(n \log n)$。

Solution ([ZJOI2012] 灾难)

我们只用对每个点找到最后一个依赖的点即可，我们可以建成一棵树（超级源点为根），每个点所依赖的点为它的所有祖先结点（不包含它本身）。

按照拓扑序遍历每一个点，并求出它的最后一个依赖的点：

- 只有一条入边：它最后一个依赖的点为那条入边的起始点。
- 不止一条入边：它最后一个依赖的点为所有入边的起始点所共同依赖的点中的最后一个，即那些入边的起始点的最后一个依赖的点的树上最近公共祖先。

所求的就是子树大小减 1（自身结点）。

时间复杂度 $\Theta(n \log n)$。

其实，上面描述的这棵树即这个加上超级源点后 DAG 的支配树（最后一个依赖的点即最近支配点），解法为一种利用 DAG 的性质的支配树求法。

Problem ([SCOI2016] 幸运数字)

给定一棵 n 个点的树，q 次询问，每次查询一条链上点权的最大异或和。

Problem ([SCOI2016] 幸运数字)

给定一棵 n 个点的树，q 次询问，每次查询一条链上点权的最大异或和。$n \leqslant 20000, Q \leqslant 200000$，点权 $< 2^{60}$。

Problem ([SCOI2016] 幸运数字)

给定一棵 n 个点的树，q 次询问，每次查询一条链上点权的最大异或和。
$n \leqslant 20000, Q \leqslant 200000$，点权 $< 2^{60}$。

Solution ([SCOI2016] 幸运数字)

我们可以用线性基维护一个集合的最大异或和。

Problem ([SCOI2016] 幸运数字)

给定一棵 n 个点的树，q 次询问，每次查询一条链上点权的最大异或和。
$n \leqslant 20000, Q \leqslant 200000$，点权 $< 2^{60}$。

Solution ([SCOI2016] 幸运数字)

我们可以用线性基维护一个集合的最大异或和。
由于点权 $< 2^{60}$，所以线性基不超过 60 个。

Problem ([SCOI2016] 幸运数字)

给定一棵 n 个点的树，q 次询问，每次查询一条链上点权的最大异或和。
$n \leqslant 20000, Q \leqslant 200000$，点权 $< 2^{60}$。

Solution ([SCOI2016] 幸运数字)

我们可以用线性基维护一个集合的最大异或和。
由于点权 $< 2^{60}$，所以线性基不超过 60 个。
对每个点倍增维护即可。

Problem ([SCOI2016] 幸运数字)

给定一棵 n 个点的树，q 次询问，每次查询一条链上点权的最大异或和。
$n \leqslant 20000, Q \leqslant 200000$，点权 $< 2^{60}$。

Solution ([SCOI2016] 幸运数字)

我们可以用线性基维护一个集合的最大异或和。
由于点权 $< 2^{60}$，所以线性基不超过 60 个。
对每个点倍增维护即可。
查询时对每条预处理出的链暴力合并。

Problem ([SCOI2016] 幸运数字)

给定一棵 n 个点的树，q 次询问，每次查询一条链上点权的最大异或和。
$n \leqslant 20000, Q \leqslant 200000$，点权 $< 2^{60}$。

Solution ([SCOI2016] 幸运数字)

我们可以用线性基维护一个集合的最大异或和。
由于点权 $< 2^{60}$，所以线性基不超过 60 个。
对每个点倍增维护即可。
查询时对每条预处理出的链暴力合并。
时间复杂度 $\Theta((n\log n + q)\log^2 w)$，其中 w 为最大点权。

■ 换根

换根 dp

我们先讲一种比较常规的换根 dp 方式。我们先随便钦定一个点为根，然后以那个结点为根进行 dp，然后我们考虑对于每一个儿子，通过他的父亲的最终答案和他的儿子的子树答案合并出最终答案，一般来说 dp 式子具有可去性，即我们可以通过一些方式将该子树对父亲结点的贡献除去，然后通过父亲除去该结点子树剩余部分和儿子子树合并得到以该点为根的答案。

换根 dp

对于不满足可去性的一些 dp（如乘积形式时逆元不存在等），我们可以考虑对有向边进行记忆化搜索，我们对一条有向边记录对应子树（删除该边及其反向边后的指向结点所在连通块）的答案，然后每次有向边分为以下三种情况：

- 已经得到该边的答案：直接返回。

换根 dp

对于不满足可去性的一些 dp（如乘积形式时逆元不存在等），我们可以考虑对有向边进行记忆化搜索，我们对一条有向边记录对应子树（删除该边及其反向边后的指向结点所在连通块）的答案，然后每次有向边分为以下三种情况：

- 已经得到该边的答案：直接返回。
- 没有处理过指向结点的信息：一般树形 dp 即可。

换根 dp

对于不满足可去性的一些 dp（如乘积形式时逆元不存在等），我们可以考虑对有向边进行记忆化搜索，我们对一条有向边记录对应子树（删除该边及其反向边后的指向结点所在连通块）的答案，然后每次有向边分为以下三种情况：

- 已经得到该边的答案：直接返回。
- 没有处理过指向结点的信息：一般树形 dp 即可。
- 处理过指向结点的信息，但未得到该有向边的答案：对没有处理的那一条出边进行处理，然后利用前后缀和求出所有指向那个结点的有向边的答案。

Problem ([POI2008] STA-Station)

给定一棵 n 个点的树，求一个结点，使得所有结点深度之和最小。

Problem ([POI2008] STA-Station)

给定一棵 n 个点的树，求一个结点，使得所有结点深度之和最小。

数据范围：$1 \leqslant n \leqslant 10^6$。

Problem ([POI2008] STA-Station)

给定一棵 n 个点的树，求一个结点，使得所有结点深度之和最小。
数据范围：$1 \leqslant n \leqslant 10^6$。

Solution ([POI2008] STA-Station)

我们可以简单地求出每个结点子树内所有点的深度之和与点数。

Problem ([POI2008] STA-Station)

给定一棵 n 个点的树，求一个结点，使得所有结点深度之和最小。

数据范围：$1 \leqslant n \leqslant 10^6$。

Solution ([POI2008] STA-Station)

我们可以简单地求出每个结点子树内所有点的深度之和与点数。

通过上面讲的换根 dp，我们就可以对每一个结点求出以那个结点为根时的深度之和，取一个最小值即可。

Problem ([POI2008] STA-Station)

给定一棵 n 个点的树，求一个结点，使得所有结点深度之和最小。
数据范围：$1 \leqslant n \leqslant 10^6$。

Solution ([POI2008] STA-Station)

我们可以简单地求出每个结点子树内所有点的深度之和与点数。

通过上面讲的换根 dp，我们就可以对每一个结点求出以那个结点为根时的深度之和，取一个最小值即可。

时间复杂度 $\Theta(n)$。

Problem ([CQOI2009] 叶子的染色)

对于每一个结点，我们可以通过 dp 求出子树内该结点的 c 为 0 和 1 时的子树答案。

Problem ([CQOI2009] 叶子的染色)

对于每一个结点，我们可以通过 dp 求出子树内该结点的 c 为 0 和 1 时的子树答案。

结点个数 $m \leqslant 10^4$，我们考虑 $\Theta(m)$ 的做法。

Problem ([CQOI2009] 叶子的染色)

对于每一个结点，我们可以通过 dp 求出子树内该结点的 c 为 0 和 1 时的子树答案。

结点个数 $m \leqslant 10^4$，我们考虑 $\Theta(m)$ 的做法。

Solution ([CQOI2009] 叶子的染色)

同样换根 dp 即可。

Problem ([CQOI2009] 叶子的染色)

对于每一个结点，我们可以通过 dp 求出子树内该结点的 c 为 0 和 1 时的子树答案。

结点个数 $m \leqslant 10^4$，我们考虑 $\Theta(m)$ 的做法。

Solution ([CQOI2009] 叶子的染色)

同样换根 dp 即可。

时间复杂度 $\Theta(m)$。

■ 例题

回到树上搜索。

回到树上搜索。

根据顶点在搜索过程中的顺序，可以得到 DFS 序。

回到树上搜索。

根据顶点在搜索过程中的顺序，可以得到 DFS 序。

根据约定不同，有多种 DFS 序，主要有：

回到树上搜索。

根据顶点在搜索过程中的顺序，可以得到 DFS 序。

根据约定不同，有多种 DFS 序，主要有：

- 普通 DFS 序：长度为 n，即直接按照 DFS 的顺序。

回到树上搜索。

根据顶点在搜索过程中的顺序，可以得到 DFS 序。

根据约定不同，有多种 DFS 序，主要有：

- 普通 DFS 序：长度为 n，即直接按照 DFS 的顺序。
- 括号序：长度为 $2n$，每次访问和结束访问结点时分别加入左括号和右括号。

回到树上搜索。

根据顶点在搜索过程中的顺序，可以得到 DFS 序。

根据约定不同，有多种 DFS 序，主要有：

- 普通 DFS 序：长度为 n，即直接按照 DFS 的顺序。
- 括号序：长度为 $2n$，每次访问和结束访问结点时分别加入左括号和右括号。
- 欧拉序：长度为 $2n-1$，每次访问和回溯时记录结点。实质上是将有向边拆成两条无向边后的欧拉回路。

回到树上搜索。

根据顶点在搜索过程中的顺序，可以得到 DFS 序。

根据约定不同，有多种 DFS 序，主要有：

- 普通 DFS 序：长度为 n，即直接按照 DFS 的顺序。
- 括号序：长度为 $2n$，每次访问和结束访问结点时分别加入左括号和右括号。
- 欧拉序：长度为 $2n-1$，每次访问和回溯时记录结点。实质上是将有向边拆成两条无向边后的欧拉回路。

DFS 序的基本性质：一个子树在 DFS 序中是一个连续的区间。

Problem ([NOI2013] 树的计数)

给定有根树的 DFS 序和 BFS 序，求树的平均深度。
保证两种序中每个点的儿子顺序一样。

Problem ([NOI2013] 树的计数)

给定有根树的 DFS 序和 BFS 序，求树的平均深度。
保证两种序中每个点的儿子顺序一样。
$n \leqslant 200000$。

Problem ([NOI2013] 树的计数)

给定有根树的 DFS 序和 BFS 序，求树的平均深度。
保证两种序中每个点的儿子顺序一样。
$n \leqslant 200000$。

Solution ([NOI2013] 树的计数)

我们可以先把每个结点按照 DFS 序重新编号。
不妨设 DFS 序为 $1, 2, \ldots, n$。

Solution ([NOI2013] 树的计数)

BFS 序是以深度为第一关键字，编号为第二关键字的排序。树的深度就是最后一个点的深度。

Solution ([NOI2013] 树的计数)

BFS 序是以深度为第一关键字，编号为第二关键字的排序。树的深度就是最后一个点的深度。

考虑以 l 为根的子树 $[l, r]$，不妨设 $l \neq r$。

Solution ([NOI2013] 树的计数)

BFS 序是以深度为第一关键字，编号为第二关键字的排序。树的深度就是最后一个点的深度。

考虑以 l 为根的子树 $[l, r]$，不妨设 $l \neq r$。

如果 BFS 序中 $l+1$ 后有子树外的点，设第一个为 i，则 fa_i 一定在这个子树外。

Solution ([NOI2013] 树的计数)

BFS 序是以深度为第一关键字，编号为第二关键字的排序。树的深度就是最后一个点的深度。

考虑以 l 为根的子树 $[l, r]$，不妨设 $l \neq r$。

如果 BFS 序中 $l + 1$ 后有子树外的点，设第一个为 i，则 fa_i 一定在这个子树外。

于是 $l + 1$ 到 i 前一个点都在子树中，且深度在 $\{d_i - 1, d_i\}$ 中。

Solution ([NOI2013] 树的计数)

BFS 序是以深度为第一关键字，编号为第二关键字的排序。树的深度就是最后一个点的深度。

考虑以 l 为根的子树 $[l, r]$，不妨设 $l \neq r$。

如果 BFS 序中 $l+1$ 后有子树外的点，设第一个为 i，则 fa_i 一定在这个子树外。

于是 $l+1$ 到 i 前一个点都在子树中，且深度在 $\{d_i - 1, d_i\}$ 中。

又因为 i 和 fa_i 在 DFS 序上与 $[l, r]$ 关系相同，于是这些点不可能同时取到 $d_i - 1$ 和 d_i，于是这些点都是 l 的儿子。

Solution ([NOI2013] 树的计数)

BFS 序是以深度为第一关键字，编号为第二关键字的排序。树的深度就是最后一个点的深度。

考虑以 l 为根的子树 $[l, r]$，不妨设 $l \neq r$。

如果 BFS 序中 $l+1$ 后有子树外的点，设第一个为 i，则 fa_i 一定在这个子树外。

于是 $l+1$ 到 i 前一个点都在子树中，且深度在 $\{d_i - 1, d_i\}$ 中。

又因为 i 和 fa_i 在 DFS 序上与 $[l, r]$ 关系相同，于是这些点不可能同时取到 $d_i - 1$ 和 d_i，于是这些点都是 l 的儿子。

进而可以得到 l 下的一层子树，容易找到 BFS 最后一个点所在递归。

Solution ([NOI2013] 树的计数)

考虑以 l 为根的子树 $[l, r]$，不妨设 $l \neq r$。

如果 BFS 序中后面的点都是子树中的点，$l+1$ 开始第一个 BFS 序不连续的点 j。其中 j 和 $j+1$ BFS 序不连续。

Solution ([NOI2013] 树的计数)

考虑以 l 为根的子树 $[l,r]$，不妨设 $l \neq r$。

如果 BFS 序中后面的点都是子树中的点，$l+1$ 开始第一个 BFS 序不连续的点 j。其中 j 和 $j+1$ BFS 序不连续。

那么 $l+1 \sim j$ 的点可能排成深度在 $[1, j-l]$ 中的情况且平均是 $\dfrac{j-l+1}{2}$。

Solution ([NOI2013] 树的计数)

考虑以 l 为根的子树 $[l,r]$，不妨设 $l \neq r$。

如果 BFS 序中后面的点都是子树中的点，$l+1$ 开始第一个 BFS 序不连续的点 j。其中 j 和 $j+1$ BFS 序不连续。

那么 $l+1 \sim j$ 的点可能排成深度在 $[1, j-l]$ 中的情况且平均是 $\frac{j-l+1}{2}$。

由于 $j+1$ 在 BFS 序在 j 之后且不是紧接着 j，所以必定深度比 j 大，或者说是 j 的儿子。

Solution ([NOI2013] 树的计数)

考虑以 l 为根的子树 $[l, r]$，不妨设 $l \neq r$。

如果 BFS 序中后面的点都是子树中的点，$l+1$ 开始第一个 BFS 序不连续的点 j。其中 j 和 $j+1$ BFS 序不连续。

那么 $l+1 \sim j$ 的点可能排成深度在 $[1, j-l]$ 中的情况且平均是 $\dfrac{j-l+1}{2}$。

由于 $j+1$ 在 BFS 序在 j 之后且不是紧接着 j，所以必定深度比 j 大，或者说是 j 的儿子。

而这个子树中深度比 j 大 1 的点中最靠前的显然也是 $j+1$。

Solution ([NOI2013] 树的计数)

考虑以 l 为根的子树 $[l, r]$，不妨设 $l \neq r$。

如果 BFS 序中后面的点都是子树中的点，$l+1$ 开始第一个 BFS 序不连续的点 j。其中 j 和 $j+1$ BFS 序不连续。

那么 $l+1 \sim j$ 的点可能排成深度在 $[1, j-l]$ 中的情况且平均是 $\dfrac{j-l+1}{2}$。

由于 $j+1$ 在 BFS 序在 j 之后且不是紧接着 j，所以必定深度比 j 大，或者说是 j 的儿子。

而这个子树中深度比 j 大 1 的点中最靠前的显然也是 $j+1$。

于是又可以确定这一层的点，类似上面处理即可。

Solution ([NOI2013] 树的计数)

考虑以 l 为根的子树 $[l,r]$，不妨设 $l \neq r$。

如果 BFS 序中后面的点都是子树中的点，$l+1$ 开始第一个 BFS 序不连续的点 j。其中 j 和 $j+1$ BFS 序不连续。

那么 $l+1 \sim j$ 的点可能排成深度在 $[1, j-l]$ 中的情况且平均是 $\frac{j-l+1}{2}$。

由于 $j+1$ 在 BFS 序在 j 之后且不是紧接着 j，所以必定深度比 j 大，或者说是 j 的儿子。

而这个子树中深度比 j 大 1 的点中最靠前的显然也是 $j+1$。

于是又可以确定这一层的点，类似上面处理即可。

时间复杂度 $O(n)$。

树上差分

我们一方面可以作为子结点对父结点的差分，与之相对，我们可以通过树上前缀和还原数据。例如在处理链操作时，我们运用差分，将两个端点加并将它们的 LCA 减，最后做一遍树上前缀和即可。

树上差分

我们一方面可以作为子结点对父结点的差分，与之相对，我们可以通过树上前缀和还原数据。例如在处理链操作时，我们运用差分，将两个端点加并将它们的 LCA 减，最后做一遍树上前缀和即可。

此外，树上前缀和也可以单独使用，例如对所有结点到根路径上信息的处理时，我们每 dfs 到一个结点，加上那个点的贡献，并在回溯时减去。通常，一些问题在此基础上可以通过数组平移得到较大的优化。

树上差分

另一方面，可以作为 DFS 序上的差分，是树上搜索（动规）的一种技巧，可用于处理子树满足可减性的信息。

相当于按照括号序不断维护信息，每个结点的答案等于右括号对应信息减左括号对应信息。

树上差分

另一方面，可以作为 DFS 序上的差分，是树上搜索（动规）的一种技巧，可用于处理子树满足可减性的信息。

相当于按照括号序不断维护信息，每个结点的答案等于右括号对应信息减左括号对应信息。

具体实现时维护一个与答案相关的信息，进入时（左括号）在答案处减去，返回时（右括号）在答案处加回。

在运用差分思想的同时，我们常常会使用数组平移这一技巧。

在运用差分思想的同时，我们常常会使用数组平移这一技巧。

我们通过一道例题来介绍。

Problem ([CSP2019D1T2] 括号树)

一棵树，每个结点有一个左括号或右括号，对于每一个结点，求根到它的路径形成的括号串中有多少个合法的括号子串。

Problem ([CSP2019D1T2] 括号树)

一棵树，每个结点有一个左括号或右括号，对于每一个结点，求根到它的路径形成的括号串中有多少个合法的括号子串。

要求时间复杂度 $\Theta(n)$。

Solution ([CSP2019D1T2] 括号树)

我们用 $f(u,i)$ 表示 u 结点对应的括号序列末尾如果加 i 个右括号，末尾形成的括号序列数。（特别地，$f(u,0)$ 表示末尾形成的括号序列数）

Solution ([CSP2019D1T2] 括号树)

我们用 $f(u,i)$ 表示 u 结点对应的括号序列末尾如果加 i 个右括号，末尾形成的括号序列数。（特别地，$f(u,0)$ 表示末尾形成的括号序列数）

若 u 结点为右括号，

$f(u,0)=f(\text{par}_u,1), f(u,1)=f(\text{par}_u,2),\ldots,f(u,i)=f(\text{par}_u,i+1),\ldots$

Solution ([CSP2019D1T2] 括号树)

我们用 $f(u,i)$ 表示 u 结点对应的括号序列末尾如果加 i 个右括号，末尾形成的括号序列数。（特别地，$f(u,0)$ 表示末尾形成的括号序列数）

若 u 结点为右括号，

$f(u,0)=f(\text{par}_u,1), f(u,1)=f(\text{par}_u,2), \dots, f(u,i)=f(\text{par}_u,i+1), \dots$

若 u 结点为左括号，

$f(u,0)=0, f(u,1)=f(\text{par}_u,0)+1, f(u,2)=f(\text{par}_u,1), \dots, f(u,i)=f(\text{par}_u,i-1)$

Solution ([CSP2019D1T2] 括号树)

我们用 $f(u,i)$ 表示 u 结点对应的括号序列末尾如果加 i 个右括号，末尾形成的括号序列数。（特别地，$f(u,0)$ 表示末尾形成的括号序列数）

若 u 结点为右括号，

$f(u,0)=f(\text{par}_u,1), f(u,1)=f(\text{par}_u,2),\dots,f(u,i)=f(\text{par}_u,i+1),\dots$

若 u 结点为左括号，

$f(u,0)=0, f(u,1)=f(\text{par}_u,0)+1, f(u,2)=f(\text{par}_u,1),\dots,f(u,i)=f(\text{par}_u,i-1)$

我们可以通过数组平移来实现状态转移，用 dp 数组维护 $f(u)$，转移相当于 dp 地址的加一或减一，具体实现见代码。

```
long long*dp;
long long ans,sum;
void dfs(int u){
    dp+=val[u];
    sum+=dp[0];
    long long tmp=dp[1];
    ++dp[0],dp[1]=0,ans^=sum*u;
    for(list<int>::iterator it=son[u].begin(); it!=son[u].end(); ++it)
        dfs(*it);
    --dp[0],dp[1]=tmp;
    sum-=dp[0];
    dp-=val[u];
}
```

其中 val[u] 当节点 u 为左括号时为 1，为右括号时为 -1。

在主程序中执行：

```
memset(a,0,sizeof(a));
dp=a+n,ans=0,sum=0;
dp[0]=1;
dfs(1);
```

Problem ([NOIP2015D2T3] 运输计划)

给你一棵 n 个结点的树，有边权，有多个任务，每个要求从 u_i 号结点到 v_i 号结点去。m 个计划, 这 m 个计划会同时开始。当这 m 个任务都完成时，工作完成。

现在可以把任意一个边的边权变为 0，试求出完成工作所需要的最短时间是多少？

Problem ([NOIP2015D2T3] 运输计划)

给你一棵 n 个结点的树，有边权，有多个任务，每个要求从 u_i 号结点到 v_i 号结点去。m 个计划, 这 m 个计划会同时开始。当这 m 个任务都完成时，工作完成。

现在可以把任意一个边的边权变为 0，试求出完成工作所需要的最短时间是多少？

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

Solution ([NOIP2015D2T3] 运输计划)

我们考虑如何判定最短时间能否不超过 x，显然我们不必处理时间不超过 x 的计划，但是必须处理时间超过 x 的计划。我们将时间超过 x 的计划取出，找到他们公共道路（我们可以对经过次数进行树上差分）中最长的一个，然后判断最长的一个减去这条公共道路长度后能否不超过 x。

Solution ([NOIP2015D2T3] 运输计划)

我们考虑如何判定最短时间能否不超过 x，显然我们不必处理时间不超过 x 的计划，但是必须处理时间超过 x 的计划。我们将时间超过 x 的计划取出，找到他们公共道路（我们可以对经过次数进行树上差分）中最长的一个，然后判断最长的一个减去这条公共道路长度后能否不超过 x。

可以观察到我们可以通过不超过 x 的最大的一个计划确定即可推算出最短时间为多少或判定无法完成。由单调性，我们可以二分。

Solution ([NOIP2015D2T3] 运输计划)

我们考虑如何判定最短时间能否不超过 x，显然我们不必处理时间不超过 x 的计划，但是必须处理时间超过 x 的计划。我们将时间超过 x 的计划取出，找到他们公共道路（我们可以对经过次数进行树上差分）中最长的一个，然后判断最长的一个减去这条公共道路长度后能否不超过 x。

可以观察到我们可以通过不超过 x 的最大的一个计划确定即可推算出最短时间为多少或判定无法完成。由单调性，我们可以二分。

时间复杂度 $\Theta(n \log n)$。

Problem ([NOIP2016D1T2] 天天爱跑步)

给定一棵 n 个点的树和 m 条有向的链。
每个结点有一个参数 W_i，对每个结点求出它是多少条链的第 W_i 个点。

Problem ([NOIP2016D1T2] 天天爱跑步)

给定一棵 n 个点的树和 m 条有向的链。
每个结点有一个参数 W_i，对每个结点求出它是多少条链的第 W_i 个点。
$n, m \leqslant 300000$。

Solution ([NOIP2016D1T2] 天天爱跑步)

这样一条有向的链可以拆成以某个时刻开始的由根出发/结束的权值为 ± 1 的链。

Solution ([NOIP2016D1T2] 天天爱跑步)

这样一条有向的链可以拆成以某个时刻开始的由根出发/结束的权值为 ± 1 的链。

对于这些链，对某个结点的贡献都可以看做是子树中某种值的权值和。

Solution ([NOIP2016D1T2] 天天爱跑步)

这样一条有向的链可以拆成以某个时刻开始的由根出发/结束的权值为 ± 1 的链。

对于这些链，对某个结点的贡献都可以看做是子树中某种值的权值和。

这可以一次 DFS 解决。

Solution ([NOIP2016D1T2] 天天爱跑步)

这样一条有向的链可以拆成以某个时刻开始的由根出发/结束的权值为 ± 1 的链。

对于这些链，对某个结点的贡献都可以看做是子树中某种值的权值和。

这可以一次 DFS 解决。

时间复杂度 $O(n+m)$。

Problem (CF1076E Vasya and a Tree)

给定一棵以 1 为根的树，m 次操作，第 i 次为对以 v_i 为根的深度小于等于 d_i 的子树的所有结点权值加 x_i。最后输出每个结点的值

Problem (CF1076E Vasya and a Tree)

给定一棵以 1 为根的树，m 次操作，第 i 次为对以 v_i 为根的深度小于等于 d_i 的子树的所有结点权值加 x_i。最后输出每个结点的值

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

Problem (CF1076E Vasya and a Tree)

给定一棵以 1 为根的树，m 次操作，第 i 次为对以 v_i 为根的深度小于等于 d_i 的子树的所有结点权值加 x_i。最后输出每个结点的值

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

Solution (CF1076E Vasya and a Tree)

我们在每个结点上记录操作，最后进行树上前缀和，运用差分，记录当前结点中子树内与当前结点的距离为 i 的结点比距离 $i-1$ 权值应当少加上的值，访问、回溯时数组平移加上修改当前点信息即可。

Problem (例题 1)

给出一棵 n 个点的有根树，每个结点 u 有一个权值 v_u，初始值为 0。有 m 次操作。要求你支持如下操作：

1. 给子树 u 中的所有结点 v 的 val 加上一个整数 t_i

Problem (例题 1)

给出一棵 n 个点的有根树，每个结点 u 有一个权值 v_u，初始值为 0。有 m 次操作。要求你支持如下操作：

1. 给子树 u 中的所有结点 v 的 val 加上一个整数 t_i
2. 询问 val_u

Problem (例题 1)

给出一棵 n 个点的有根树，每个结点 u 有一个权值 v_u，初始值为 0。有 m 次操作。要求你支持如下操作：

1. 给子树 u 中的所有结点 v 的 val 加上一个整数 t_i
2. 询问 val_u

数据范围

$1 \leqslant n, m \leqslant 10^6$，保证每时每刻的 $|\mathrm{val}_i| \leqslant 10^9$

Problem (例题 1)

给出一棵 n 个点的有根树，每个结点 u 有一个权值 v_u，初始值为 0。有 m 次操作。要求你支持如下操作：

1. 给子树 u 中的所有结点 v 的 val 加上一个整数 t_i
2. 询问 val_u

数据范围

$1 \leqslant n, m \leqslant 10^6$，保证每时每刻的 $|\mathrm{val}_i| \leqslant 10^9$

Solution (例题 1)

我们可以使用树状数组按照 DFS 序维护 val，由于子树在 DFS 序中是一个连续的区间，所以我们只用支持区间加，单点查询即可。

Problem (例题 2)

给出一棵 n 个点的有根树，每个结点 u 有一个权值 v_u，初始值为 0。有 m 个操作。要求你支持如下操作：

1. 给 u 到 v 的路径上的所有结点 k 的 val 加上一个整数 t_i

操作完后询问所有 val_u

Problem (例题 2)

给出一棵 n 个点的有根树，每个结点 u 有一个权值 v_u，初始值为 0。有 m 个操作。要求你支持如下操作：

1. 给 u 到 v 的路径上的所有结点 k 的 val 加上一个整数 t_i

操作完后询问所有 val_u

数据范围

$1 \leqslant n \leqslant 10^6, 1 \leqslant m \leqslant 5 \times 10^6$，保证每时每刻的 $|\mathrm{val}_i| \leqslant 10^9$

Problem (例题 2)

给出一棵 n 个点的有根树，每个结点 u 有一个权值 v_u，初始值为 0。有 m 个操作。要求你支持如下操作：

1. 给 u 到 v 的路径上的所有结点 k 的 val 加上一个整数 t_i

操作完后询问所有 val_u

数据范围

$1 \leqslant n \leqslant 10^6, 1 \leqslant m \leqslant 5 \times 10^6$，保证每时每刻的 $|\mathrm{val}_i| \leqslant 10^9$

Solution (例题 2)

作子结点对于父结点的树上差分即可。

Problem (例题 3)

给出一棵 n 个点的有根树，每个结点 u 有一个权值 v_u，初始值为 0。有 m 个操作。要求你支持如下操作：

1. 给 u 到 v 的路径上所有结点 k 的 val 加上一个整数 t_i

Problem (例题 3)

给出一棵 n 个点的有根树，每个结点 u 有一个权值 v_u，初始值为 0。有 m 个操作。要求你支持如下操作：

1. 给 u 到 v 的路径上所有结点 k 的 val 加上一个整数 t_i
2. 给子树 u 中的所有结点 v 的 val 加上一个整数 t_i。

Problem (例题 3)

给出一棵 n 个点的有根树，每个结点 u 有一个权值 v_u，初始值为 0。有 m 个操作。要求你支持如下操作：

1. 给 u 到 v 的路径上所有结点 k 的 val 加上一个整数 t_i
2. 给子树 u 中的所有结点 v 的 val 加上一个整数 t_i。
3. 询问子树 u 的 val 之和

Problem (例题 3)

给出一棵 n 个点的有根树，每个结点 u 有一个权值 v_u，初始值为 0。有 m 个操作。要求你支持如下操作：

1. 给 u 到 v 的路径上所有结点 k 的 val 加上一个整数 t_i
2. 给子树 u 中的所有结点 v 的 val 加上一个整数 t_i。
3. 询问子树 u 的 val 之和
4. 询问 u 到 v 的路径上所有结点的 val 之和

Problem (例题 3)

给出一棵 n 个点的有根树，每个结点 u 有一个权值 v_u，初始值为 0。有 m 个操作。要求你支持如下操作：

1. 给 u 到 v 的路径上所有结点 k 的 val 加上一个整数 t_i
2. 给子树 u 中的所有结点 v 的 val 加上一个整数 t_i。
3. 询问子树 u 的 val 之和
4. 询问 u 到 v 的路径上所有结点的 val 之和

$1 \leqslant n, m \leqslant 10^6$，保证每时每刻的 $|\mathrm{val}_i| \leqslant 10^9$。

Problem (例题 3)

给出一棵 n 个点的有根树，每个结点 u 有一个权值 v_u，初始值为 0。有 m 个操作。要求你支持如下操作：

1. 给 u 到 v 的路径上所有结点 k 的 val 加上一个整数 t_i
2. 给子树 u 中的所有结点 v 的 val 加上一个整数 t_i。
3. 询问子树 u 的 val 之和
4. 询问 u 到 v 的路径上所有结点的 val 之和

$1 \leqslant n, m \leqslant 10^6$，保证每时每刻的 $|\mathrm{val}_i| \leqslant 10^9$。

好，我们直接讲树链剖分。

1 简单树上搜索

2 树的直径

3 树形动规

4 DFS 序

5 树上差分

6 树链剖分
- 树链剖分的引入
- 树链剖分的算法及基本定义
- 重链剖分
- 一种随机剖分方法
- 长链剖分

7 LCA

8 其他算法注解

树链剖分的算法

我们可以发现，每个非叶结点都和它的一个儿子的 DFS 序连续。

树链剖分的算法

我们可以发现，每个非叶结点都和它的一个儿子的 DFS 序连续。

于是我们可以通过人为选定这一个儿子，使得一条链可以分为较少的区间，从而可以类似于子树，通过多个 DFS 序的区间的操作或询问完成树上链操作或查询。

树链剖分的基本定义

- 重儿子：u 儿子中，$\mathtt{dfn}_v = \mathtt{dfn}_u + 1$ 的结点 v。

树链剖分的基本定义

- 重儿子：u 儿子中，$\mathtt{dfn}_v = \mathtt{dfn}_u + 1$ 的结点 v。
- 轻儿子：不是重儿子的儿子。

树链剖分的基本定义

- 重儿子：u 儿子中，$\texttt{dfn}_v = \texttt{dfn}_u + 1$ 的结点 v。
- 轻儿子：不是重儿子的儿子。
- 重边：连向重儿子的边。

树链剖分的基本定义

- 重儿子：u 儿子中，$\texttt{dfn}_v = \texttt{dfn}_u + 1$ 的结点 v。
- 轻儿子：不是重儿子的儿子。
- 重边：连向重儿子的边。
- 轻边：连向轻儿子的边。

树链剖分的基本定义

- 重儿子：u 儿子中，$\mathtt{dfn}_v = \mathtt{dfn}_u + 1$ 的结点 v。
- 轻儿子：不是重儿子的儿子。
- 重边：连向重儿子的边。
- 轻边：连向轻儿子的边。
- 重链：由重边构成的极长链。

树链剖分的基本定义

- 重儿子：u 儿子中，$\mathtt{dfn}_v = \mathtt{dfn}_u + 1$ 的结点 v。
- 轻儿子：不是重儿子的儿子。
- 重边：连向重儿子的边。
- 轻边：连向轻儿子的边。
- 重链：由重边构成的极长链。
- 顶（top）：一个点所在的重链上深度最低的点。

重链剖分

我们选取儿子中对应子树大小最大的一个作为重儿子。

重链剖分

我们选取儿子中对应子树大小最大的一个作为重儿子。

可以证明，每一条链可以分解成 $O(\log n)$ 条重链和轻边。

重链剖分

我们选取儿子中对应子树大小最大的一个作为重儿子。

可以证明，每一条链可以分解成 $O(\log n)$ 条重链和轻边。

Proof (重链剖分)

我们先考虑轻边的个数。

重链剖分

我们选取儿子中对应子树大小最大的一个作为重儿子。

可以证明，每一条链可以分解成 $O(\log n)$ 条重链和轻边。

Proof (重链剖分)

我们先考虑轻边的个数。

当从点 u 向上经过轻边时，父亲点 p 一定存在一个儿子点 u' 使得 $\mathrm{siz}_{u'} \geqslant \mathrm{siz}_u$，因此 $\mathrm{siz}_p \geqslant 2\,\mathrm{siz}_u + 1$。

重链剖分

我们选取儿子中对应子树大小最大的一个作为重儿子。

可以证明，每一条链可以分解成 $O(\log n)$ 条重链和轻边。

Proof (重链剖分)

我们先考虑轻边的个数。

当从点 u 向上经过轻边时，父亲点 p 一定存在一个儿子点 u' 使得 $\mathrm{siz}_{u'} \geqslant \mathrm{siz}_u$，因此 $\mathrm{siz}_p \geqslant 2\,\mathrm{siz}_u + 1$。

由于根结点的 siz 为 n，所以轻边的个数不超过 $\lfloor \log_2(n+1) \rfloor - 1$。

重链剖分

我们选取儿子中对应子树大小最大的一个作为重儿子。

可以证明，每一条链可以分解成 $O(\log n)$ 条重链和轻边。

Proof (重链剖分)

我们先考虑轻边的个数。

当从点 u 向上经过轻边时，父亲点 p 一定存在一个儿子点 u' 使得 $\mathrm{siz}_{u'} \geqslant \mathrm{siz}_u$，因此 $\mathrm{siz}_p \geqslant 2\,\mathrm{siz}_u + 1$。

由于根结点的 siz 为 n，所以轻边的个数不超过 $\lfloor \log_2(n+1) \rfloor - 1$。

由于两条重链之间一定隔着一条轻边，因此重链和轻边的个数和不超过 $2\lfloor \log_2(n+1) \rfloor - 1$，为 $O(\log n)$。

证毕。

我们可以先进行一次 dfs，将所有结点的重儿子找出来。

```
int sz[MAXN], son[MAXN];
int dep[MAXN], fa[MAXN];
void dfs1(int u) {
    sz[u] = 1;
    for (int i = head[u]; i; i = nxt[i])
        if (to[i] != fa[u]) {
            fa[to[i]] = u;
            dep[to[i]] = dep[u] + 1;
            dfs1(to[i]);
            sz[u] += sz[to[i]];
            if (sz[son[u]] < sz[to[i]])
                son[u] = to[i];
        }
}
```

然后再进行一次 dfs，处理出 dfs 序和 top。

```
int top[MAXN], dfn[MAXN], idx;
void dfs2(int u) {
    dfn[u] = ++idx;
    if (son[u]) {
        top[son[u]] = top[u];
        dfs2(son[u]);
    }
    for (int i = head[u]; i; i = nxt[i])
        if (to[i] != fa[u] && to[i] != son[u])
            top[to[i]] = to[i], dfs2(to[i]);
}

```

然后再进行一次 dfs，处理出 dfs 序和 top。

```
int top[MAXN], dfn[MAXN], idx;
void dfs2(int u) {
    dfn[u] = ++idx;
    if (son[u]) {
        top[son[u]] = top[u];
        dfs2(son[u]);
    }
    for (int i = head[u]; i; i = nxt[i])
        if (to[i] != fa[u] && to[i] != son[u])
            top[to[i]] = to[i], dfs2(to[i]);
}

```

至此，我们完成了重链剖分的预处理。

只需要在主程序里调用两句话即可。

```
    dep[root] = 1; top[root] = root;
    dfs1(root); dfs2(root);
```

树剖求 LCA

由于树剖中常常用到 LCA，所以我们提前讲一下如何求。

树剖求 LCA

由于树剖中常常用到 LCA，所以我们提前讲一下如何求。

我们记 $\mathrm{lca}(u, v) = w$，显然 w 所在的链的 top 深度一定比 u 和 v 的 top 深度低。

树剖求 LCA

由于树剖中常常用到 LCA，所以我们提前讲一下如何求。

我们记 $\mathrm{lca}(u,v)=w$，显然 w 所在的链的 top 深度一定比 u 和 v 的 top 深度低。

因此我们重复下面的规则：

树剖求 LCA

由于树剖中常常用到 LCA，所以我们提前讲一下如何求。

我们记 $\mathrm{lca}(u, v) = w$，显然 w 所在的链的 top 深度一定比 u 和 v 的 top 深度低。

因此我们重复下面的规则：

- `while` u 和 v 不在一条链上

树剖求 LCA

由于树剖中常常用到 LCA，所以我们提前讲一下如何求。

我们记 $\mathrm{lca}(u,v)=w$，显然 w 所在的链的 top 深度一定比 u 和 v 的 top 深度低。

因此我们重复下面的规则：

- `while` u 和 v 不在一条链上
  - 选择 u，v 所在链 top 深度大的那个，记为 x，如果相同随意选一个。

树剖求 LCA

由于树剖中常常用到 LCA，所以我们提前讲一下如何求。

我们记 $\mathrm{lca}(u,v)=w$，显然 w 所在的链的 top 深度一定比 u 和 v 的 top 深度低。

因此我们重复下面的规则：

- `while` u 和 v 不在一条链上
  - 选择 u，v 所在链 top 深度大的那个，记为 x，如果相同随意选一个。
  - `X=fa[top[x]]`

树剖求 LCA

由于树剖中常常用到 LCA，所以我们提前讲一下如何求。

我们记 $\text{lca}(u, v) = w$，显然 w 所在的链的 top 深度一定比 u 和 v 的 top 深度低。

因此我们重复下面的规则：

- `while` u 和 v 不在一条链上
  - 选择 u，v 所在链 top 深度大的那个，记为 x，如果相同随意选一个。
  - `X=fa[top[x]]`
- 返回 u 和 v 中深度低的那个

树剖求 LCA

```cpp
int LCA(int x, int y) {
    while (top[x] != top[y]) {
        if (dep[top[x]] > dep[top[y]]) x = fa[top[x]];
        else y = fa[top[y]];
    }
    if (dep[x] < dep[y]) return x;
    return y;
}
```

常规重链剖分数据结构技巧

可以发现，因为每次询问有 $O(\log n)$ 条链，因此可以使用数据结构维护。

时间复杂度为 $O(\log n f(n))$，其中 $f(n)$ 为数据结构维护的复杂度。

回顾一下之前的 lca 代码，发现其实这个直接遍历了 u 到 v 的链！

所以只要把那个修改不多就能做完了

```
void modify(int x, int y) {
    while (top[x] != top[y]) {
        if (dep[top[x]] < dep[top[y]])
            std::swap(x, y);
        modify(dfn[top[x]], dfn[x]);
        x = fa[top[x]];
    }
    if (dep[x] < dep[y]) std::swap(x, y);
    modify(dfn[y], dfn[x]);
    return y;
}
```

常规重链剖分数据结构技巧

诶诶诶不是还有子树吗

子树不是讲过了吗？

dfs 序是连续的！

Problem (洛谷 3950 部落冲突)

给你一棵树，有 3 个操作：
`Q p q`：询问 p, q 是否连通
`C p q`：把 $p \to q$ 这条边割断
`U x`：恢复第 x 次操作二
数据范围：$1 \leqslant n, m \leqslant 3 \times 10^5$。

Problem (洛谷 3950 部落冲突)

给你一棵树，有 3 个操作：
`Q p q`：询问 p, q 是否连通
`C p q`：把 $p \to q$ 这条边割断
`U x`：恢复第 x 次操作二
数据范围：$1 \leqslant n, m \leqslant 3 \times 10^5$。

Solution (洛谷 3950 部落冲突)

我们将每条边是否被割断记为那条边的边权（0 或 1），然后割断和恢复操作即单点修改，询问是否连通即查询路径和是否为 0。

Problem ([HEOI2016/TJOI2016] 树)

给出一棵树，其中根结点有标记，有两种操作，第一种是在一个结点上打标记，一种是询问一个点最近的打了标记的祖先。

$1 \leqslant N, Q \leqslant 10^5$

Solution ([HEOI2016/TJOI2016] 树)

一种简单的想法是不断的向上跳链，不停地查询最小值，但是这样做要两个 $\log$

换一种想法，计算每个点的贡献，发现一个点放 tag 相当于子树取 $\min$，所以只需要区间修改和单点查询就好了。

实际上第一种方法也可以使用动态开点线段树优化到 $O(n\log n)$

实际上，我们可以对于每一条链维护最高的打标记的点，这样可以 $\Theta(1)$ 判定一个点向上跳到这一条链有没有标记点，我们只需要一次查询一次最近的一个即可。

Problem ([TJOI2018] 异或)

给出一个有根树，点有权值。有两种询问，第一种是询问 x 子树中结点和 y 异或的最大值，第二种是询问 x 到 y 路径上的点与 z 异或的最大值。

数据范围：$1 < n, q \leqslant 10^5, 1 \leqslant \mathrm{val}_i, z < 2^{30}$。

Solution ([TJOI2018] 异或)

询问两个值异或最大值可以使用 trie

因为询问利用了 dfs 序的连续性，同事我们要取出一个区间内的 trie 结点

所以我们在 dfs 序上跑可持久化 trie，然后再普通地树剖一下就过来

复杂度两个 log。

但是这样太暴力了，对于子树，我们直接 dfs 序上可持久化 trie，对于链，维护每个点到根的路径的可持久化 trie，询问时差分一下就好，复杂度一个 log。

Problem ([SDOI2014] 旅行)

一棵树，每个点有一个颜色和权值，询问给定一条有向的链 S 到 T，保证 S 和 T 颜色相同，询问这条链上颜色和 S 相同的所有点的权值和和最大值。同时会有修改单点的颜色或权值的操作。

$1 \leqslant N, Q,$ 颜色 $\leqslant 10^5$

Problem ([SDOI2014] 旅行)

一棵树，每个点有一个颜色和权值，询问给定一条有向的链 S 到 T，保证 S 和 T 颜色相同，询问这条链上颜色和 S 相同的所有点的权值和和最大值。同时会有修改单点的颜色或权值的操作。

$1 \leqslant N, Q,$ 颜色 $\leqslant 10^5$

Solution ([SDOI2014] 旅行)

因为是单点修改，因此可以方便地使用动态开点线段树。
对每一个颜色维护一个线段树。
复杂度两个 $\log$。

Problem ([LNOI2014] LCA)

有根树，每次询问给出 `l r z`，求 $\sum_{l\leqslant i\leqslant r} \mathrm{dep}_{\mathrm{lca}(i,z)}$。

$1 \leqslant n, Q \leqslant 5 \times 10^4$

Solution ([LNOI2014] LCA)

发现直接算不太好算，此时考虑一个点的贡献。

发现 x 对 y 的贡献是将 x 到根的路径全部加 1，y 到根的路径的权值和。

同时这个贡献是可以合并的，这个启发我们前缀和。

既然前缀和了就把询问离线，同时拆成两个区间就做完了。

复杂度两个 $\log$。

也可以使用可持久化线段树维护 $[1, i]$ 的点的上述权值贡献。每加入一个点相当于链修改。在线询问差分即可。

Problem ([HNOI2016] 网络)

给定一棵树。有三种操作：

0 u v t：在 u 到 v 的链上进行重要度为 t 的数据传输。

1 x：结束第 x 个数据传输。

2 x：询问不经过点 x 的数据传输中重要度最大的是多少（无解输出 -1）。

Solution ([HNOI2016] 网络)

首先，查询不经过点 x，我们可以转换为在这条链以外的结点进行修改，然后查询点 x，我们可以利用树剖对链外的其他结点转化成 $O(\log n)$ 个区间。

然后我们考虑结束传输，即删除。

我们可以标记永久化，把每个数插入到树剖中线段树的 $O(\log^2 n)$ 个结点上，并不下传标记，每次查询单点时查询所有线段树上的祖先结点（含本身），删除便从对应线段树结点删除。为了维护点集，对线段树的结点建立两个大根堆，分别表示插入的和删除的，如果堆顶相等就一起弹出，否则对答案无影响不必即时从插入的数中删除。

很多题目如果需要对一条链的结点有 dp 最大值等操作，需要用到平衡树等数据结构，我们可以将轻重儿子分开讨论。

Problem (不知名的题)

有根树，每次询问给出一个点 u 和它子树里的一个点 v，求 u 到 v 路径上除 u 外离 u 最近的结点。

$1 \leqslant n, Q \leqslant 5 \times 10^6$

Solution (不知名的题)

可以发现，倍增已经不太优秀了，空间大。

根据 dfs 序，可以给他的儿子序列开一个平衡树或 vector，可以在里面二分下面的点的 dfs 序，找到包含它的那个儿子。这样空间是小了，但是常数又大了。

考虑到跳上去最后一条边，如果是重边，即是 u 的重儿子，否则就是最后一个从轻链跳上去的结点。复杂度一个 log。

如果离线可以魔改 tarjan 做到 $O(n+Q)$

实际上如果你用 RMQ 维护区间最小值出现的位置可以做到 $O(n\log n+Q)$

如果再上黑科技可以做到在线的 $O(n+q)$

一种随机剖分方法

我们对每一个点 u 随机一个权值 v_u，将一个点的儿子中子树最大权值最大的一个儿子设为重儿子。

设 u 向上每个祖先子树大小依次为 s_i（特别地，$s_0 = \mathrm{siz}(u), s_k = n$），上面的轻边期望个数为 $f(n)$，则有：

一种随机剖分方法

$$
\begin{aligned}
f(u) &= k - \frac{s_0}{s_1 - 1} - \frac{s_1}{s_2 - 1} - \cdots - \frac{s_{k-1}}{s_k - 1} \\
&< k - \frac{s_0}{s_1 - 1} - \frac{s_1 - 1}{s_2 - 1} - \cdots - \frac{s_{k-1} - 1}{s_k - 1} \\
&\leqslant k - k\left(\frac{s_0}{s_k - 1}\right)^{\frac{1}{k}} \\
&\leqslant (n-1) - (n-1)\left(\frac{1}{n-1}\right)^{\frac{1}{n-1}} \\
&= \log n - \frac{\log^2 n + 2}{2n} + O(n^{-2})
\end{aligned}
$$

一种随机剖分方法

那么 $\forall u \in V, f(u) = O(\log n)$。

由于随机值较方便维护，所以在重链剖分的动态化中能有所应用。

长链剖分

长链剖分也就是往长的儿子去剖，而不是大的儿子。

具体应用：

- $O(n\log n) - O(1)$ K 级祖先
- 类 dsu on tree 的技巧（实际上重链剖分也有这个）

k 级祖先

- k 级祖先：对于一个结点满足以下递归定义的那个祖先。
  - 一个结点的 1 级祖先为其父亲。

k 级祖先

- k 级祖先：对于一个结点满足以下递归定义的那个祖先。
  - 一个结点的 1 级祖先为其父亲。
  - 一个结点的 $k+1$ 级祖先为其 k 级祖先的父亲。

k 级祖先

- k 级祖先：对于一个结点满足以下递归定义的那个祖先。
  - 一个结点的 1 级祖先为其父亲。
  - 一个结点的 $k+1$ 级祖先为其 k 级祖先的父亲。

k 级祖先问题，通常给定一棵静态的树，多组询问，每次询问给定一个结点 u 和一个正整数 k，询问 u 的 k 级祖先。

k 级祖先

我们进行长链剖分。

首先，先证明一个结点的 k 级祖先所在的链链长一定大于等于 k。

Proof (k 级祖先)

如果向上跳的时候没有经过轻边的话，显然成立。如果经过了轻边，那么因为长链剖分，跳上轻边后的链长一定大于等于当前的，因为跳了 k，所以对于这种情况也成立。

k 级祖先

我们可以预处理每个 DFS 序的对应结点、每个链顶向上链长的结点编号和每个结点向上跳 2^k 级祖先后的结点。时间复杂度 $\Theta(n\log n)$。

对于每一次查询，我们先向上跳最大的 $r=2^a\leqslant k$ 层，然后由于到达的链长 $\geqslant r$，且 $k-r<r$，因此剩余 $k-r$ 级一定能够 $\Theta(1)$ 到达。

因此，总时间复杂度 $\Theta(n\log n)-\Theta(1)$。

长链剖分同样有一些其他性质，可供贪心等算法使用。

Problem (bzoj3252 攻略)

给定一棵树，选取 k 个点，求根到这些点路径上的点的最大点权和。

Problem (bzoj3252 攻略)

给定一棵树，选取 k 个点，求根到这些点路径上的点的最大点权和。
数据范围：$1 \leqslant n \leqslant 2 \times 10^5$。

Problem (bzoj3252 攻略)

给定一棵树，选取 k 个点，求根到这些点路径上的点的最大点权和。
数据范围：$1 \leqslant n \leqslant 2 \times 10^5$。

Solution (bzoj3252 攻略)

首先因为少选取点不会使价值变大，且选取儿子节点肯定不比父亲结点劣。我们可以贪心地将问题转化为选取不超过 k 个叶子结点，求根到这些点路径上的点的最大点权和。

我们将树进行长链剖分，可以证明选取其中前 k 长的链最优。时间复杂度 $\Theta(n)$。

Proof (bzoj3252 攻略)

对于一种给定的选择方法，我们也对被路径覆盖的点进行带权长链剖分。

如果有一条原树的长链没有被选择而存在剖出的链更短，那么我们将一条剖出的链换成长链显然更优（如果有与之重复的选择那一条，否则任选一条）。

因此可能没有更优的方案只有一种：选择前 k 长的链。

LCA

我们回过头来讲讲 LCA。(我们谈到 LCA 问题时，一般默认为两个结点。)

同 RMQ 问题，在 LCA 问题中时间复杂度的 — 前后也分别连接预处理时间复杂度和单次查询时间复杂度。

倍增求 LCA

上面讲过了。

预处理出每个结点的深度和 2^k 级祖先。

每次查询时：

1. 将深度深的一个结点通过倍增找到和另一个结点深度相同的祖先。
2. 通过倍增找到最远非公共祖先（如果 2^k 级祖先不同则改求它们的 LCA）。
3. LCA 即最后的兄弟结点的父亲。

时间复杂度 $\Theta(n) - \Theta(\log n)$。

树剖求 LCA

上面讲过了。

我们每次把深度（或链顶深度）较大的一个往上跳，直到两个结点处在同一条链上，然后返回深度较小的一个即可。

时间复杂度 $\Theta(n) - O(\log n)$。

树剖求 LCA

使用树剖求 LCA 的好处：

如果你写的好，就是又短又快的。

一般的出题人数据造出来很难把树剖 lca 卡满，因此在很多时候树剖 lca 跑得飞快！

由于空间 $O(n)$，很多时候比倍增不知高明到哪里去了。

~~如果使用倍增优化树剖，可以做到~~$\cancel{O(Q\log\log n)}$~~，并且比~~$\cancel{O(Q\log n)}$~~慢多了。~~

~~详见http://10.49.18.71/submission/36594~~

如何卡满树剖 LCA 呢？

构造一棵满二叉树即可，但是树剖常数依然很小。

这是一个离线算法，可以快速并相对方便地求出 lca。

我们将每个 LCA 询问挂在对应两个结点上，然后进行 DFS，维护每个访问过的结点与当前结点的最近公共祖先（当前访问栈中深度最深的一个），每次回溯时将整个子树挂到父亲结点上即可。

时间复杂度 $\Theta((n+q)\alpha(n))$。（由于 $\alpha(n)$ 变化率极小，有时会在计算时间复杂度时被忽略。）

```
void dfs(int rt) {
    vis[rt] = true;
    int v;
    for (int i = head[rt]; i; i = e[i].nxt) {
        v = e[i].to;
        if (!vis[v]) {
            dfs(v);
            f[v] = find(rt);
        }
    }
    for (int i = headq[rt]; i; i = Q[i].nxt) ans[Q[i].tg] = find(Q[i].oth);
}
```

使用 RMQ 的 LCA 方法

这是一个 DFS 序的应用。

这里使用的是欧拉序，可以证明，这两个点的第一次出现的位置之间的深度最浅的点就是他们的 LCA。

于是问题就变成了 RMQ 问题，可以在 $O(n+q)$ 或 $O(n\log n+q)$ 的时间内完成。

使用 RMQ 的 LCA 方法

```
void dfs(int u, int fa = 0) {
    ST[0][++idx] = u, pos[u] = idx;
    for (int i = head[u]; i; i = e[i].nxt) {
        int v = e[i].to;
        if (v != fa) {
            dep[v] = dep[u] + 1;
            dfs(v, u);
            ST[0][++idx] = u;
        }
    }
}
inline int getmin(int a, int b) {return dep[a] < dep[b] ? a : b;}
inline void init() {
    for (int i = 1; i < 20; i++) {
        for (int j = 1; j + (1 << i) - 1 <= idx; j++) {
            ST[i][j] = getmin(ST[i - 1][j], ST[i - 1][j + (1 << i - 1)]);
        }
    }
}
inline int LCA(int x, int y) {
    int l = pos[x], r = pos[y];
    if (l > r) std::swap(l, r);
    int tmp = LG[r - l + 1];
    return getmin(ST[tmp][l], ST[tmp][r - (1 << tmp) + 1]);
}
```

使用 RMQ 的 LCA 方法

有一种 $\Theta(n)-\Theta(1)$ 的在线 LCA 只用接着上面继续优化即可，我们发现欧拉序相邻点的深度为 1，可以利用这个性质，转化成±1 RMQ 问题，可以在 $\Theta(n)-\Theta(1)$ 的时间复杂度解决。

lca 因为其优良的性质，在题目中有许多应用。

我们通过一些题目来熟悉这些性质。

Problem (森林的直径)

给你一个森林。要求支持动态加边，维护直径。可以离线。
要求时间复杂度 $O((n+q)\log n)$。

Problem (森林的直径)

给你一个森林。要求支持动态加边，维护直径。可以离线。
要求时间复杂度 $O((n+q)\log n)$。

Solution (森林的直径)

我们考虑两棵树合并后的直径。

- 不经过新加的边：其中一棵树的直径。
- 经过新加的边：每棵树中距离这两个点最远的点到这两个点的路径加上新加的边。

注意到一棵树中距离一个点最远的点一定是任意直径的一个端点。

关于查询两点距离，$\mathrm{dis}(u,v)=\mathrm{dis}(u,\mathrm{lca}(u,v))+\mathrm{dis}(v,\mathrm{lca}(u,v))$。我们可以离线后建出最后的森林，直接在最后的森林上查询距离即可。

如果需要在线，请考虑 LCT。

Problem (CF1045C Hyperspace Highways)

给一张 n 个点 m 条边的图，保证若有一个环，一定是完全子图，多次询问两个点之间的最短路径长度

Solution (CF1045C Hyperspace Highways)

由于直接缩点细节可能较多，我们使用圆方树，然后赋权求树上路径长度即可：即 u 到 $\mathrm{lca}(u,v)$ 的路径长度加上 v 到 $\mathrm{lca}(u,v)$ 的路径长度。

Problem ([NOIP2013D1T3] 货车运输)

A 国有 n 座城市，编号从 1 到 n，城市之间有 m 条双向道路。每一条道路对车辆都有重量限制，简称限重。现在有 q 辆货车在运输货物，司机们想知道每辆车在不超过车辆限重的情况下，最多能运多重的货物。

Solution ([NOIP2013D1T3] 货车运输)

显然 u 到 v 的链可以拆分成 u 到 $\mathrm{lca}(u,v)$ 的链和 $\mathrm{lca}(u,v)$ 到 v 的链。

因此我们可以将链拆成两部分后通过任意一种方法处理。

（注：树链剖分并没有显式地求 LCA，但是实际上也拆成了很多条链，其中一些为 u 到 $\mathrm{lca}(u,v)$ 的链拆分成，另一些由 $\mathrm{lca}(u,v)$ 到 v 的链拆分成。）

Problem ([AHOI2008] Meet 紧急集合)

给你一棵树，和 3 个结点，要你找到树上的一个点，使得三个点到这个点的距离和最小，并输出个距离

Solution ([AHOI2008] Meet 紧急集合)

显然选择的点在这三个点之间的路径上。如果只在 a 到 b 和 a 到 c 的路径上，那么我们可以往 b 到 c 的路径移动一条边，显然答案会减少 1。因此，答案为三条两两间路径的交点。

通过分类讨论，我们可以发现，肯定会有两个点对的 lca 相同，那么第三个 lca 就是所求点。

Problem (CF1051F The Shortest Statement)

给出一张 n 个点 $m(m-n\leqslant 20)$ 条边的无向图，q 询问两点之间的最短路。

$n, m, q \leqslant 10^5$

Problem (CF1051F The Shortest Statement)

给出一张 n 个点 $m(m-n \leqslant 20)$ 条边的无向图，q 询问两点之间的最短路。

$n, m, q \leqslant 10^5$

Solution (CF1051F The Shortest Statement)

这个无向图为一棵树加上至多 21 条非树边，共最多连接 42 个点，我们求出它们到所有点的最短路后，任意点的图上最短路一定为树上路径或经过一个特殊点，枚举即可，如使用一般堆优化的 `dijkstra` 与 $O(n \log n) - \Theta(\log n)$ 的 lca 则时间复杂度为 $\Theta(n(m-n)\log m + q(m-n+\log n))$。

$$E(\max(S)) = E\left(\sum_{T\subseteq S}(-1)^{|T|+1}\min(T)\right) = \sum_{T\subseteq S}(-1)^{|T|+1}E(\min(T))$$

（我们规定 $\min(\emptyset) = 0$，即上式忽略 T 为 $\emptyset$ 的情况。）

$$E(\max(S)) = E\left(\sum_{T\subseteq S}(-1)^{|T|+1}\min(T)\right) = \sum_{T\subseteq S}(-1)^{|T|+1}E(\min(T))$$

（我们规定 $\min(\emptyset)=0$，即上式忽略 T 为 $\emptyset$ 的情况。）

Proof (min-max 容斥)

设 $\min(T_0)$ 在 S 中有 k 个数比它大。

- $k=0$：显然只有 $T=\{\min(T_0)\}$ 时产生 1 的贡献。
- $k>0$：那么只有 2^k 个 T 使得 $\min(T_0)$ 产生贡献，其中 2^{k-1} 个集合产生 -1 的贡献，有 2^{k-1} 个集合产生 1 的贡献。两者相互抵消。

即 $\sum_{T\subseteq S}(-1)^{|T|+1}\min(T)=\max(S)$，证毕。

返回

在维护可合并的静态数据时，我们通常使用倍增。

即对于存在（快速的）merge 和 advance 满足 $f(i, a+b, \texttt{args}) = \texttt{merge}(f(i, a, \texttt{args}), f(\texttt{advance}(i, a), b, \texttt{args}))$ 时（args 表示其他状态信息），我们可以预处理出对于所有的 $i, k, \texttt{args}$ 的对应 $f(i, 2^k, \texttt{args})$，然后对每次查询的 $O(\log n)$ 个状态进行合并。

ST 表

在线静态区间最小值（在线 RMQ）问题：我们需要在线回答若干次区间最小值的查询，期间不进行修改操作。（在此类问题中，— 前后分别连接预处理时间复杂度和单次查询时间复杂度，如一般的线段树为 $\Theta(n) - \Theta(\log n)$。）

ST 表常用于要求时间复杂度为 $\Theta(n \log n) - \Theta(1)$ 的情境中。

我们考虑 $f(i,j)$ 表示区间 $[i, i+j)$ 的最小值。

显然我们可以发现 $f(i, a+b) = \min(f(i,a), f(i+a, b))$，那么我们只需预处理出 $f(i, 2^k)$ 就可以完成 $\Theta(n \log n) - \Theta(\log n)$。

ST 表

我们考虑 $f(i,j)$ 表示区间 $[i,i+j)$ 的最小值。

但是最小值有一个比较特殊的性质，$f(a,b)=f(a,c)+f(a+b-d,d)\quad(c+d\geqslant b)$，即我们只需要合并的每个区间的并集为所查询的区间即可，并不需要保证每个数被覆盖且仅被覆盖一次。

若 $2^k\leqslant b\leqslant 2^{k+1}$，则 $f(a,b)=\min(f(a,2^k),f(a+b-2^k,2^k))$。我们可以使用 `bit_width(b)-1`(C++20) 或 `31-__builtin_clz(b)` 来得到 k。

由于这两者是 $\Theta(1)$ 的（内置的），我们可以做到 $\Theta(n\log n)-\Theta(1)$ 的在线 RMQ 做法。

返回

Problem (± 1 RMQ 问题)

对于一个满足 $|a_{i+1} - a_i| = 1$ 的数列 $\{a_i\}$：

- 用 $O(n)$ 的时间复杂度进行预处理：`init(a,n)`。
- 用 $O(1)$ 的时间复杂度进行查询区间最小值及其位置：`minimum(l,r)`。

差分数列：$\{u_i\}$ 的差分数列 $\{v_i\}$ 满足 $\forall i \in [1, n) \cap \mathbb{N}, v_i = u_{i+1} - u_i$。

我们将其平均分块，每一块的块大小为 $l(l \in [1, n] \cap \mathbb{N})$。

我们预处理时显然可以 $\Theta(n)$ 求出每一个块的最小值及其位置。（单块 $\Theta(l)$ 处理即可）

然后对块间最小值及其位置用 ST 表进行预处理，时间复杂度 $\Theta\left(\frac{n}{l}\log\frac{n}{l}\right) = \Theta\left(\frac{n}{l}(\log n - \log l)\right) = \Theta\left(\frac{n\log n}{l}\right)$。

由于一个序列 $\{u_i\}$ 的最小值的位置可以由差分数列 $\{v_i\}$ 唯一确定，而因为 $\{a_i\}$ 的差分数列任意下标对应值为 ± 1，因此所有块最多有 $\min(2^{l-1}, n)$ 种情况，我们可以对每一种情况的每一个区间进行预处理答案，时间复杂度 $\Theta(l^2 \min(2^{l-1}, n))$。

综上所述，预处理时间复杂度为 $\Theta\left(\frac{n\log n}{l} + l^2 \min(2^l, n)\right)$。

- 对于同块的区间，直接查询预处理时处理出来的答案，时间复杂度 $\Theta(1)$。
- 对于跨块的区间，我们可以先对中间整块的部分在 ST 表中查询，然后在对两边一块内的部分按同块区间查询，时间复杂度 $\Theta(1)$。

综上所述，查询时间复杂度为 $\Theta(1)$。

由上可知，查询时间复杂度恒为 $\Theta(1)$，因此只需使预处理时间复杂度满足为 $O(n)$ 即可。

取块大小 $l = \Theta(\log n)$ 且满足 $2^l l^2 = O(n)$ 即可满足 $O(n)$ 的时间复杂度，常取 $l = \left\lfloor \dfrac{\log n}{2} \right\rfloor$。

返回

对于图中所有点双进行缩点，将缩出来的树上的点称为方点，原先图上的点称为圆点，然后我们把缩点后的树中每一个方点与对应圆点相连，最后形成的树就是圆方树（方点之间按缩点后树连边，圆点之间不连边），通常用来解决一些涉及图上涉及点双（边双也可使用类圆方树方式）类问题。

返回