# 图论

吴清月

2020 年 10 月 3 日

# Trails and Glades

# Trails and Glades

给一张图，添加最少的边使得整张图存在一条欧拉回路。
$1 \leq n \leq 10^6, 0 \leq m \leq 10^6$

# Trails and Glades

# Trails and Glades

欧拉回路存在的充要条件是所有点的度数均为偶数，并且至多只有一个存在边的连通块。

# Trails and Glades

欧拉回路存在的充要条件是所有点的度数均为偶数，并且至多只有一个存在边的连通块。

所以如果整张图已经连通那么答案就是奇点数量 /2。

若原图不连通，则设原图分成了 $k$ 个存在边的连通块，其中有 $x$ 个存在奇点，另外 $k-x$ 个不存在奇点，奇点总数为 $s$。

# Trails and Glades

欧拉回路存在的充要条件是所有点的度数均为偶数，并且至多只有一个存在边的连通块。

所以如果整张图已经连通那么答案就是奇点数量 /2。

若原图不连通，则设原图分成了 $k$ 个存在边的连通块，其中有 $x$ 个存在奇点，另外 $k-x$ 个不存在奇点，奇点总数为 $s$。

首先可以选取奇点将 $x$ 个连通块连在一起，奇点总数 $-2(x-1)$，添加的边数 $+(x-1)$。此时图中只有一个连通块存在奇点，其余 $k-x$ 个不存在奇点。

接下来每次选取一个奇点和另一个连通块相连。一次连边后奇点总数不变，添加的边数 $+1$。

最后再把剩下的奇点两两相连。

# Trails and Glades

欧拉回路存在的充要条件是所有点的度数均为偶数，并且至多只有一个存在边的连通块。

所以如果整张图已经连通那么答案就是奇点数量 /2。

若原图不连通，则设原图分成了 $k$ 个存在边的连通块，其中有 $x$ 个存在奇点，另外 $k-x$ 个不存在奇点，奇点总数为 $s$。

首先可以选取奇点将 $x$ 个连通块连在一起，奇点总数 $-2(x-1)$，添加的边数 $+(x-1)$。此时图中只有一个连通块存在奇点，其余 $k-x$ 个不存在奇点。

接下来每次选取一个奇点和另一个连通块相连。一次连边后奇点总数不变，添加的边数 $+1$。

最后再把剩下的奇点两两相连。

总边数为 $(x-1)+(k-x)+\frac{s-2(x-1)}{2}=\frac{s}{2}+k-x$，注意特判不存在奇点的情况。

时间复杂度 $O(n)$。

# Graph

# Graph

给定一张 $n$ 个点 $m$ 条边的图，每一条边权为 1 或 2。

你需要给每一个点分配一个实数点权，使得边权等于相连的点权之和。在此基础上使得所有点权的绝对值相加最小。

$1 \leq n \leq 100000, 0 \leq m \leq 200000$

# Graph

# Graph

注意到，一旦 $x$ 的点权确定，$x$ 所在的连通块的点权就确定了。

# Graph

注意到，一旦 $x$ 的点权确定，$x$ 所在的连通块的点权就确定了。

设 $x$ 的点权为 $v$，则其余所有的点权都可以表示成 $kv+b$ 的形式，其中 $k \in \{1,-1\}$，并且这个表达式可以通过 dfs 求出来。

# Graph

注意到，一旦 $x$ 的点权确定，$x$ 所在的连通块的点权就确定了。

设 $x$ 的点权为 $v$，则其余所有的点权都可以表示成 $kv+b$ 的形式，其中 $k \in \{1,-1\}$，并且这个表达式可以通过 dfs 求出来。

由于原图可能存在环，因此在 dfs 的过程中可能出现某个点的点权同时被表示成 $k_1v+b_1$ 和 $k_2v+b_2$ 的情况。此时分三种情况：

# Graph

注意到，一旦 $x$ 的点权确定，$x$ 所在的连通块的点权就确定了。

设 $x$ 的点权为 $v$，则其余所有的点权都可以表示成 $kv+b$ 的形式，其中 $k \in \{1,-1\}$，并且这个表达式可以通过 dfs 求出来。

由于原图可能存在环，因此在 dfs 的过程中可能出现某个点的点权同时被表示成 $k_1v+b_1$ 和 $k_2v+b_2$ 的情况。此时分三种情况：

- $k_1 \neq k_2$：$v$ 可以直接解出来；
- $k_1 = k_2$ 且 $b_1 = b_2$：无影响；
- $k_1 = k_2$ 且 $b_1 \neq b_2$：无解。

# Graph

# Graph

一旦 $v$ 可以解出来，整个连通块内所有的点就都确定了。这时候再进行一遍 dfs 求出所有的点权，如果矛盾直接无解。

现在唯一的问题就是当 $v$ 无法解出来的时候怎样安排点权使得绝对值之和最小。

# Graph

一旦 $v$ 可以解出来，整个连通块内所有的点就都确定了。这时候再进行一遍 dfs 求出所有的点权，如果矛盾直接无解。

现在唯一的问题就是当 $v$ 无法解出来的时候怎样安排点权使得绝对值之和最小。

如果一个点的点权是 $kx+b$，那么它的贡献就是 $|kx+b|$，当 $x=-\frac{b}{k}$ 取到最小值 0，两边斜率分别为正负 1。

我们的问题相当于要求 $n$ 个这样的函数加起来之后最低点是多少，也就是斜率由正变为负的点是多少。

# Graph

一旦 $v$ 可以解出来，整个连通块内所有的点就都确定了。这时候再进行一遍 dfs 求出所有的点权，如果矛盾直接无解。

现在唯一的问题就是当 $v$ 无法解出来的时候怎样安排点权使得绝对值之和最小。

如果一个点的点权是 $kx+b$，那么它的贡献就是 $|kx+b|$，当 $x=-\frac{b}{k}$ 取到最小值 0，两边斜率分别为正负 1。

我们的问题相当于要求 $n$ 个这样的函数加起来之后最低点是多少，也就是斜率由正变为负的点是多少。

可以发现答案就是所有函数零点的中位数。

这样这道题就完成了，时间复杂度 $O(n)$。

# Village (Minimum)

# Village (Minimum)

一棵 $n$ 个点的树，每一个点上有一个人。

你需要把所有人的位置做一个置换，要求每个人的位置都必须发生改变，**最小化**所有人的移动距离之和。

$1 \leq n \leq 10^5$

# Village (Minimum)

# Village (Minimum)

显然一条边的贡献只有可能是 0 或者 2，当贡献大于 2 的时候我们可以通过调整使得总移动距离减小。

# Village (Minimum)

显然一条边的贡献只有可能是 0 或者 2，当贡献大于 2 的时候我们可以通过调整使得总移动距离减小。

接下来的问题就是选取最少的贡献为 2 的边覆盖所有的点，也就是最小边覆盖问题。

# Village (Minimum)

显然一条边的贡献只有可能是 0 或者 2，当贡献大于 2 的时候我们可以通过调整使得总移动距离减小。

接下来的问题就是选取最少的贡献为 2 的边覆盖所有的点，也就是最小边覆盖问题。

树上的最小边覆盖问题可以直接 dfs 贪心解决。时间复杂度 $O(n)$。

# Village (Maximum)

# Village (Maximum)

一棵 $n$ 个点的树，每一个点上有一个人。

你需要把所有人的位置做一个置换，要求每个人的位置都必须发生改变，**最大化**所有人的移动距离之和。

$1 \leq n \leq 10^5$

# Village (Maximum)

# Village (Maximum)

类比上一道题，我们考虑每一条边的贡献。若这条边连接的两棵子树大小分别为 $a, b$，则这条边贡献最多可以是 $2\min(a, b)$。

# Village (Maximum)

类比上一道题，我们考虑每一条边的贡献。若这条边连接的两棵子树大小分别为 $a, b$，则这条边贡献最多可以是 $2\min(a, b)$。

选取重心将整棵树分成若干棵连通子树，设每一部分的大小分别为 $s_1, s_2, \ldots, s_k$（注意重心自己也算作一个大小为 1 的连通子树），则 $\max s_i \leq \frac{n}{2}$。

每次选取最大值和次大值进行匹配，则匹配之后的子树大小仍然满足条件。注意特判一下三个 1 的情况。

# Village (Maximum)

类比上一道题，我们考虑每一条边的贡献。若这条边连接的两棵子树大小分别为 $a, b$，则这条边贡献最多可以是 $2\min(a, b)$。

选取重心将整棵树分成若干棵连通子树，设每一部分的大小分别为 $s_1, s_2, \ldots, s_k$（注意重心自己也算作一个大小为 1 的连通子树），则 $\max s_i \leq \frac{n}{2}$。

每次选取最大值和次大值进行匹配，则匹配之后的子树大小仍然满足条件。注意特判一下三个 1 的情况。

这样每一条边都达到了理论最大值，所以移动距离一定也达到了最大值。

时间复杂度 $O(n)$。

# Removing Coins

# Removing Coins

一棵 $n$ 个点的树，初始每一个点上有一枚棋子。

先手后手轮流进行操作，每次操作选定树上的一个有棋子的点，移除这个点上所有的棋子并且将其它棋子向这个点移动一条边。

求是否先手必胜。

$1 \leq n \leq 2 \times 10^5$

# Removing Coins

# Removing Coins

每次操作要么使直径长度 $-1$ 要么使直径长度 $-2$，并且这两种方案一定都存在。

# Removing Coins

每次操作要么使直径长度 $-1$ 要么使直径长度 $-2$，并且这两种方案一定都存在。

所以就变成了一个基础博弈论问题，当直径长度为 3 的倍数时先手必胜，否则后手必胜。

时间复杂度 $O(n)$。

# Royal Questions

# Royal Questions

有 $x$ 个王子 $y$ 个公主，每一个公主喜欢其中两个王子，还有一个美丽度。

你需要将王子和公主配对，使得每一个公主都和自己喜欢的王子配对，并且配对的公主美丽度之和最大。

$x, y \leq 200000$

# Royal Questions

# Royal Questions

王子抽象为点，公主抽象为边，可以建出图来。

# Royal Questions

王子抽象为点，公主抽象为边，可以建出图来。

将所有配对的公主拿出来，可以发现答案形成了若干个连通块，每一个连通块点数等于边数（基环树）或点数等于边数 +1（树）。

实际上就是要求整张图的最大生成环套树森林。

# Royal Questions

# Royal Questions

类比最大生成树算法，先将所有边排序。接下来对于每一条边 $(u, v)$：

# Royal Questions

类比最大生成树算法，先将所有边排序。接下来对于每一条边 $(u, v)$：

- 若 $u, v$ 不连通且分别位于两棵树内：加入这条边，得到一棵新的树。
- 若 $u, v$ 不连通且分别位于一棵树和一棵基环树内：加入这条边，得到一个基环树。
- 若 $u, v$ 不连通且分别位于两棵基环树内：无法加入。
- 若 $u, v$ 连通且位于一棵树内：加入这条边，树变成基环树。
- 若 $u, v$ 连通且位于一棵基环树内：无法加入。

# Royal Questions

类比最大生成树算法，先将所有边排序。接下来对于每一条边 $(u, v)$：

- 若 $u, v$ 不连通且分别位于两棵树内：加入这条边，得到一棵新的树。
- 若 $u, v$ 不连通且分别位于一棵树和一棵基环树内：加入这条边，得到一个基环树。
- 若 $u, v$ 不连通且分别位于两棵基环树内：无法加入。
- 若 $u, v$ 连通且位于一棵树内：加入这条边，树变成基环树。
- 若 $u, v$ 连通且位于一棵基环树内：无法加入。

只需要在并查集的基础上多记录一个 flag 数组表示该连通块是树还是基环树即可。

时间复杂度 $O(m \log m)$。

# 树树树

# 树树树

给你平面上的 $n$ 个点，求最大曼哈顿距离生成树。
$n \leq 100000$

# 树树树

# 树树树

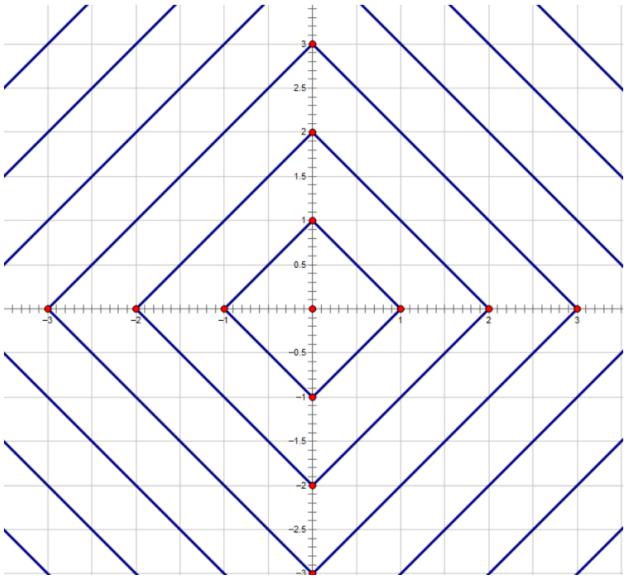
首先上一个套路：曼哈顿转切比雪夫。

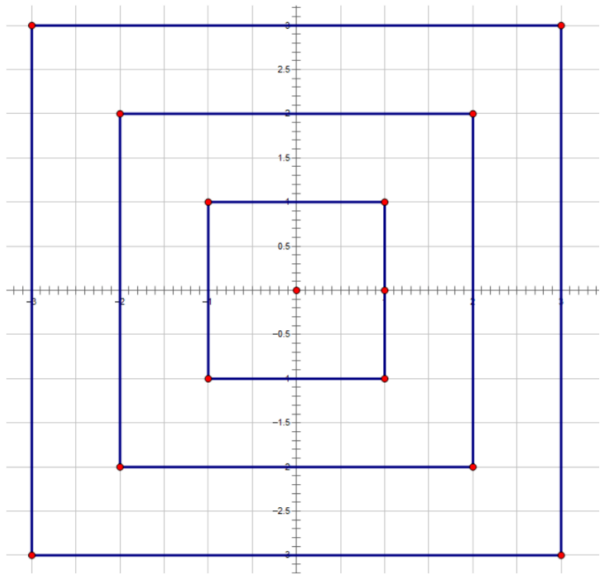
# 树树树

首先上一个套路：曼哈顿转切比雪夫。

两个点的切比雪夫距离定义为 $\max(|x_1 - x_2|, |y_1 - y_2|)$。

如果我们以一个点 $(x, y)$ 为中心，将到它的距离相等的点连成一条“等距线”，那么曼哈顿距离和切比雪夫距离分别长成下面两个样子：





# 树树树

# 树树树

直观地讲，我们将坐标系旋转 $45°$，就可以实现两个距离之间的互相转化。

对于一个点 $(x, y)$，我们把它变成 $(x + y, x - y)$ 即可（如果切比雪夫转曼哈顿的话最后还得除以 2）。

# 树树树

直观地讲，我们将坐标系旋转 $45°$，就可以实现两个距离之间的互相转化。

对于一个点 $(x, y)$，我们把它变成 $(x + y, x - y)$ 即可（如果切比雪夫转曼哈顿的话最后还得除以 2）。

这样做有什么好处呢？

# 树树树

直观地讲，我们将坐标系旋转 45°，就可以实现两个距离之间的互相转化。

对于一个点 $(x, y)$，我们把它变成 $(x + y, x - y)$ 即可（如果切比雪夫转曼哈顿的话最后还得除以 2）。

这样做有什么好处呢？

我们考虑之前提到的 Borůvka 算法。对于每个连通块，我们希望求出离它最远的点。

容易发现，最远的点只有可能是横/纵坐标最小/最大的点，

直接记录一下四个方向的最大值即可。

# 树树树

直观地讲，我们将坐标系旋转 45°，就可以实现两个距离之间的互相转化。

对于一个点 $(x, y)$，我们把它变成 $(x + y, x - y)$ 即可（如果切比雪夫转曼哈顿的话最后还得除以 2）。

这样做有什么好处呢？

我们考虑之前提到的 Borůvka 算法。对于每个连通块，我们希望求出离它最远的点。

容易发现，最远的点只有可能是横/纵坐标最小/最大的点，

直接记录一下四个方向的最大值即可。

时间复杂度其实是 $O(n)$ 的，因为第一次合并完之后就只剩下最多两个集合了。

# Three Circuits

# Three Circuits

$n$ 个点 $m$ 条边的图，不含自环和重边，问是否能找出三个环，刚好覆盖所有的边。

环可以包含重复点，但是不能包含重复边。

$1 \leq n, m \leq 10^5$

（原题保证了连通图，但是不连通也可以做）

# Three Circuits

# Three Circuits

看似十分简单的一道题，然而……

# Three Circuits

看似十分简单的一道题，然而……

如果存在度数为奇数的点那么一定不可能。

否则可以发现每一个连通块都存在欧拉回路，可以被一个环覆盖。

我们的思路是求出每一个连通块最多能够被多少环覆盖（大于 3 个记为 3 个），然后看整张图能否被 3 个环覆盖。

# Three Circuits

# Three Circuits

接下来开始分情况讨论：

# Three Circuits

接下来开始分情况讨论：

- 所有点的度数均为 2：最多只能被一个环覆盖。
- 最大度数大于 4：可以被三个环覆盖。

# Three Circuits

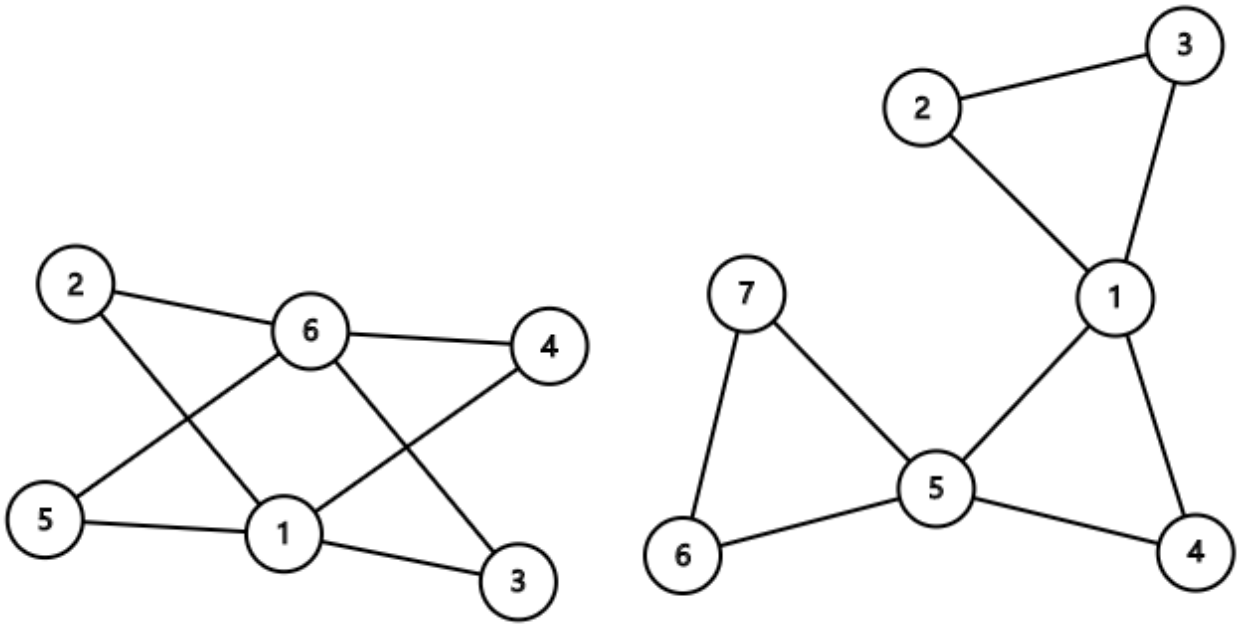
接下来开始分情况讨论：

- 所有点的度数均为 2：最多只能被一个环覆盖。
- 最大度数大于 4：可以被三个环覆盖。
- 最大度数等于 4：
  - 只存在一个度数等于 4 的点：最多可以被两个环覆盖。
  - 存在至少三个度数等于 4 的点：可以被三个环覆盖。

# Three Circuits

# Three Circuits

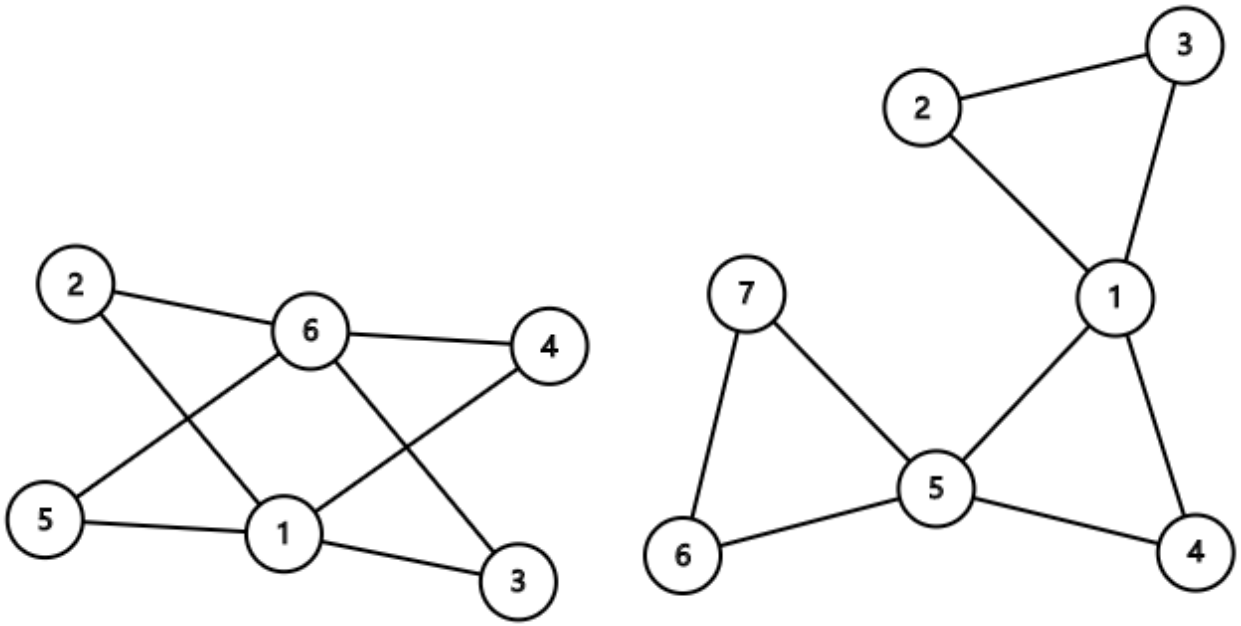
对于恰好存在两个度数为 4 的点的情况，需要分两种情况讨论：




判就完了。

# Three Circuits

对于恰好存在两个度数为 4 的点的情况，需要分两种情况讨论：




判就完了。
时间复杂度 $O(n)$。

# Graph and Queries

# Graph and Queries

$n$ 个点 $m$ 条边的点带权的图，点权形成了 1 到 $n$ 的一个排列，支持如下两个操作：

- 删除一条边
- 查询 $x$ 所在的连通块内的最大点权，并将这个点的点权重置为 0。

允许离线。

$1 \leq n \leq 2 \times 10^5, 1 \leq m \leq 3 \times 10^5, 1 \leq q \leq 5 \times 10^5$

# Graph and Queries

# Graph and Queries

前置技能：Kruskal 重构树。

# Graph and Queries

前置技能：Kruskal 重构树。

按照删除时间建树，父节点的删除时间早于子节点。

则对于任意一次询问，我们可以通过树上倍增定位到此时包含这个点的连通块是哪棵子树，问题变成了支持查询子树内的最大点权并且重置为 0。

# Graph and Queries

前置技能：Kruskal 重构树。

按照删除时间建树，父节点的删除时间早于子节点。

则对于任意一次询问，我们可以通过树上倍增定位到此时包含这个点的连通块是哪棵子树，问题变成了支持查询子树内的最大点权并且重置为 0。

可以转 dfs 树变成一维问题，用线段树即可实现。

时间复杂度 $O(n \log n)$。

# 运输计划

# 运输计划

一棵 $n$ 个点的树，给你 $m$ 条路径。
你需要选出一条边把边权改成 0，让这些路径长度的最大值最小。
$n, m \leq 3 \times 10^5$

# 运输计划

# 运输计划

最大值最小，先二分。

# 运输计划

最大值最小，先二分。
二分之后，有一些路径的长度已经满足要求了，直接忽略。
剩下的路径由于太长，所以每一条路径都必须经过被删除的边。

# 运输计划

最大值最小，先二分。

二分之后，有一些路径的长度已经满足要求了，直接忽略。

剩下的路径由于太长，所以每一条路径都必须经过被删除的边。

假设有 $k$ 条超过限制的路径，对于每一条路径，我们把这条路径上所有边的 cnt+1 ，然后看被所有 cnt=k 的边，删掉其中最长的边，看是否能满足条件。

# 运输计划

最大值最小，先二分。

二分之后，有一些路径的长度已经满足要求了，直接忽略。

剩下的路径由于太长，所以每一条路径都必须经过被删除的边。

假设有 $k$ 条超过限制的路径，对于每一条路径，我们把这条路径上所有边的 cnt+1 ，然后看被所有 cnt=k 的边，删掉其中最长的边，看是否能满足条件。

对于 cnt+1 的操作，我们可以用树上差分实现。每次让cnt[u]++,cnt[v]++,cnt[lca]-=2。

时间复杂度 $O((n+m)\log n)$，有点卡常。

# Complete Compress

# Complete Compress

一棵 $n$ 个点的树，其中有若干个点上有一枚棋子。

每次你可以选择两个距离至少为 2 的有棋子的点，并且从这两个点中各自拿出一枚棋子，向中间移动一条边。

求是否能将所有棋子移动到一个点上。如果能，求最小移动次数。

$2 \leq n \leq 2000$

# Complete Compress

# Complete Compress

先枚举一下最后汇聚到哪个点上。若棋子到这个点的距离之和为奇数直接不可能，否则尝试构造一组解。

# Complete Compress

先枚举一下最后汇聚到哪个点上。若棋子到这个点的距离之和为奇数直接不可能，否则尝试构造一组解。

以这个点为根进行 dfs，设 $g_i$ 表示 $i$ 这棵子树内的点到 $i$ 的距离之和，$f_i$ 表示 $i$ 这棵子树内的点经过内部消耗后，到 $i$ 的距离之和最小可以变成多少。则若根节点的 $f = 0$ 那就合法，代价即为距离之和。

$g_i$ 可以直接求出，问题是求 $f_i$。

# Complete Compress

先枚举一下最后汇聚到哪个点上。若棋子到这个点的距离之和为奇数直接不可能，否则尝试构造一组解。

以这个点为根进行 dfs，设 $g_i$ 表示 $i$ 这棵子树内的点到 $i$ 的距离之和，$f_i$ 表示 $i$ 这棵子树内的点经过内部消耗后，到 $i$ 的距离之和最小可以变成多少。则若根节点的 $f = 0$ 那就合法，代价即为距离之和。

$g_i$ 可以直接求出，问题是求 $f_i$。

设 $i$ 的子树分别为 $j_1, j_2, \ldots, j_k$，则如果其中 $f_j + size_j$ 最大的子树不超过整体的一半，就可以消耗到 0 或 1（取决于奇偶性），否则就只能消耗到 $f_j + size_j - \sum(g_{j'} + size_{j'})$。

枚举 $f + size$ 最大的 $j$，即可求出 $f_i$。

# Complete Compress

先枚举一下最后汇聚到哪个点上。若棋子到这个点的距离之和为奇数直接不可能，否则尝试构造一组解。

以这个点为根进行 dfs，设 $g_i$ 表示 $i$ 这棵子树内的点到 $i$ 的距离之和，$f_i$ 表示 $i$ 这棵子树内的点经过内部消耗后，到 $i$ 的距离之和最小可以变成多少。则若根节点的 $f = 0$ 那就合法，代价即为距离之和。

$g_i$ 可以直接求出，问题是求 $f_i$。

设 $i$ 的子树分别为 $j_1, j_2, \ldots, j_k$，则如果其中 $f_j + size_j$ 最大的子树不超过整体的一半，就可以消耗到 0 或 1（取决于奇偶性），否则就只能消耗到 $f_j + size_j - \sum(g_{j'} + size_{j'})$。

枚举 $f + size$ 最大的 $j$，即可求出 $f_i$。

时间复杂度 $O(n^2)$。

# Bichrome Spanning Tree

# Bichrome Spanning Tree

有一个 $n$ 个点 $m$ 条边的带权无向图。另外，你还被给定了一个整数 $x$。

请求出将每条边染成黑色或白色，且满足下面的条件的方案数模 $10^9+7$ 的值：

图中存在一棵同时包含染成白色的边和染成黑色的边的生成树。另外，在所有这种生成树中，权重最小的一棵的权重恰为 $x$。

$1 \leq n \leq 1000, 1 \leq m \leq 2000, 1 \leq x \leq 10^{12}, 1 \leq w \leq 10^9$

# Bichrome Spanning Tree

# Bichrome Spanning Tree

首先求出最小生成树，设它的权值为 $v$。

# Bichrome Spanning Tree

首先求出最小生成树，设它的权值为 $v$。

若 $v = x$，则相当于要求存在一棵同时包含黑边和白边的最小生成树。容斥一下，变成所有的最小生成树都有相同的颜色。

# Bichrome Spanning Tree

首先求出最小生成树，设它的权值为 $v$。

若 $v = x$，则相当于要求存在一棵同时包含黑边和白边的最小生成树。容斥一下，变成所有的最小生成树都有相同的颜色。

先随便求一棵最小生成树，接下来考虑不在最小生成树上的边 $(u, v)$。若 $w(u, v)$ 恰好等于最小生成树上 $u \to v$ 这条链上的最大权值，那么我们就得到了一棵新的最小生成树。

通过这种操作我们可以得到所有在最小生成树上的边。设边数共有 $k$ 条，则答案即为 $2^m - 2 \times 2^{m-k}$。

# Bichrome Spanning Tree

# Bichrome Spanning Tree

若 $v < x$，则所有在最小生成树上的边的颜色必定相同。

# Bichrome Spanning Tree

若 $v < x$，则所有在最小生成树上的边的颜色必定相同。

先丢一个结论：在满足上述要求的情况下，包含不同颜色的边的最小生成树 $T'$ 一定能够通过更改至多一条边变为最小生成树。

# Bichrome Spanning Tree

若 $v < x$，则所有在最小生成树上的边的颜色必定相同。

先丢一个结论：在满足上述要求的情况下，包含不同颜色的边的最小生成树 $T'$ 一定能够通过更改至多一条边变为最小生成树。

证明.

不妨设最小生成树上的边均为白色。
假设 $T'$ 不能通过更改任何一条边变为最小生成树。则任取一条 $T'$ 中的黑边，设它为 $(u, v)$。

# Bichrome Spanning Tree

若 $v < x$，则所有在最小生成树上的边的颜色必定相同。

先丢一个结论：在满足上述要求的情况下，包含不同颜色的边的最小生成树 $T'$ 一定能够通过更改至多一条边变为最小生成树。

证明.

不妨设最小生成树上的边均为白色。
假设 $T'$ 不能通过更改任何一条边变为最小生成树。则任取一条 $T'$ 中的黑边，设它为 $(u, v)$。
任取一个最小生成树 $T$，断掉 $u \to v$ 这条链上最大的边并加入 $(u, v)$。这样得到的生成树是包含 $(u, v)$ 的生成树中最小的，因此一定不劣于 $T'$。而同时它又能通过更改一条边变为最小生成树，矛盾。 □

# Bichrome Spanning Tree

# Bichrome Spanning Tree

所以我们枚举所有不在最小生成树上的边 $(u, v)$，按照上述方式找到包含它的最小生成树，设为 $S$。按照 $S$ 的权值将所有边分为三种，小于 $x$，等于 $x$ 和大于 $x$。

# Bichrome Spanning Tree

所以我们枚举所有不在最小生成树上的边 $(u, v)$，按照上述方式找到包含它的最小生成树，设为 $S$。按照 $S$ 的权值将所有边分为三种，小于 $x$，等于 $x$ 和大于 $x$。

- 对于第一类边，必须和最小生成树颜色相同。
- 对于第二类边，必须存在至少一条边和最小生成树颜色不同。
- 对于第三类边，可以任意染色。

# Bichrome Spanning Tree

所以我们枚举所有不在最小生成树上的边 $(u, v)$，按照上述方式找到包含它的最小生成树，设为 $S$。按照 $S$ 的权值将所有边分为三种，小于 $x$，等于 $x$ 和大于 $x$。

- 对于第一类边，必须和最小生成树颜色相同。
- 对于第二类边，必须存在至少一条边和最小生成树颜色不同。
- 对于第三类边，可以任意染色。

方案数可以直接计算。时间复杂度 $O(n \log n)$，不想写查询链上最大的边的话可以直接写 $O(n^2)$ 的暴力。

# Flights for Regular Customers

# Flights for Regular Customers

有 $n$ 个城市，还有 $m$ 个航班，第 $i$ 个航班从 $u_i$ 飞往 $v_i$，需要你之前坐过至少 $d_i$ 次航班。

求从 1 到 $n$ 至少需要坐多少次航班。无解输出 `Impossible`。

$2 \le n \le 150, 1 \le m \le 150, 0 \le d \le 10^9$

# Flights for Regular Customers

# Flights for Regular Customers

先讲所有边按照 $d_i$ 排序。

# Flights for Regular Customers

先讲所有边按照 $d_i$ 排序。

考虑设 $f_{i,j,k}$ 表示从 1 开始，只考虑限制小于等于 $d_i$ 的边，走 $j$ 步是否能到达 $k$。转移方程如下：

$$f_{i,j,k} = \begin{cases} f_{i-1,j,k} & d_i > j \\ f_{i,j-1,v} & \exists w(v,k) \le d_i \end{cases}$$

# Flights for Regular Customers

# Flights for Regular Customers

注意到，只要 $i$ 是固定的，第二种转移的情况就是固定的。而 $i$ 又只有最多 150 段。

# Flights for Regular Customers

注意到，只要 $i$ 是固定的，第二种转移的情况就是固定的。而 $i$ 又只有最多 150 段。

所以可以分段矩乘。处理出每一段矩阵，然后每次让当前数组乘上这个矩阵的 $d_{i+1} - d_i$ 次方即可。

# Flights for Regular Customers

注意到，只要 $i$ 是固定的，第二种转移的情况就是固定的。而 $i$ 又只有最多 150 段。

所以可以分段矩乘。处理出每一段矩阵，然后每次让当前数组乘上这个矩阵的 $d_{i+1} - d_i$ 次方即可。

矩阵中的数只有 01，因此可以用 bitset 优化。

时间复杂度 $O\left(\frac{n^3}{w} m \log d\right)$

# Nastya and Time Machine

# Nastya and Time Machine

有一棵 $n$ 个点的树，你从 1 号点开始遍历，最后回到 1 号点。通过一条边需要的时间是 1，初始时间是 0。

你有一台时间机器，可以让你在不改变所在位置的情况下穿越到任意一个小于当前时间且大于等于 0 的时间点。（必须是整数）

由于不能违背宇宙法则，你不能在两次在同一个时间点位于同一个节点。

你需要设置一个遍历次序，满足所有时间点的最大值最小。

$1 \leq n \leq 10^5$

# Nastya and Time Machine

首先考虑一些特殊情况。

# Nastya and Time Machine

首先考虑一些特殊情况。
一条链？

# Nastya and Time Machine

首先考虑一些特殊情况。
一条链？
除了只有两个点的答案为 1，其余答案都是 2。

# Nastya and Time Machine

首先考虑一些特殊情况。
一条链？
除了只有两个点的答案为 1，其余答案都是 2。
一个菊花图？

# Nastya and Time Machine

首先考虑一些特殊情况。
一条链？
除了只有两个点的答案为 1，其余答案都是 2。
一个菊花图？
答案即为 $n-1$。

# Nastya and Time Machine

首先考虑一些特殊情况。
一条链？
除了只有两个点的答案为 1，其余答案都是 2。
一个菊花图？
答案即为 $n-1$。
似乎节点的度数会对答案造成很大的影响。

# Nastya and Time Machine

# Nastya and Time Machine

设最大时间为 $T$。

对于一棵子树，如果你在时间点 $t+1$ 到达这棵子树的根，那么一定可以在时间点 $t$ 遍历完整棵子树并且回到根。（在根上用一次穿越）

# Nastya and Time Machine

设最大时间为 $T$。

对于一棵子树，如果你在时间点 $t+1$ 到达这棵子树的根，那么一定可以在时间点 $t$ 遍历完整棵子树并且回到根。（在根上用一次穿越）

也就是说对于一个点，一个分支只会消耗 1 单位时间。（$t$ 出去，$t+1$ 回来）

# Nastya and Time Machine

设最大时间为 $T$。

对于一棵子树，如果你在时间点 $t+1$ 到达这棵子树的根，那么一定可以在时间点 $t$ 遍历完整棵子树并且回到根。（在根上用一次穿越）

也就是说对于一个点，一个分支只会消耗 1 单位时间。（$t$ 出去，$t+1$ 回来）

唯一的特殊情况是 $t=T$ 的情况，这时候我们没法直接进入子树，只能先穿越到 0 时间，然后在 1 时间回来。

# Nastya and Time Machine

设最大时间为 $T$。

对于一棵子树，如果你在时间点 $t+1$ 到达这棵子树的根，那么一定可以在时间点 $t$ 遍历完整棵子树并且回到根。（在根上用一次穿越）

也就是说对于一个点，一个分支只会消耗 1 单位时间。（$t$ 出去，$t+1$ 回来）

唯一的特殊情况是 $t=T$ 的情况，这时候我们没法直接进入子树，只能先穿越到 0 时间，然后在 1 时间回来。

进一步发现其实 $T$ 和 0 没区别，因为无论如何下一步我们一定要穿越到 0。（除非已经到终点了，这时候我们可以穿越到 0 使得整个路径成为一个环）

# Nastya and Time Machine

# Nastya and Time Machine

我们发现，对于一个度数为 $d$ 的点，我们只需要 $d$ 的时刻就够了！

# Nastya and Time Machine

我们发现，对于一个度数为 $d$ 的点，我们只需要 $d$ 的时刻就够了！

1. $t$ 时刻进入并且往第一个分支走，$t+1$ 回来；
2. $t+1$ 时刻往第二个分支走，$t+2$ 回来；
3. ……
4. $d-1$ 时刻往第 $d-t$ 个分支走，$d$ 回来；
5. 传送到 0 时刻，往第 $d-t+1$ 个分支走，1 回来；
6. ……
7. $t-2$ 时刻往第 $d-1$ 个分支走，$t-1$ 回来。
8. $t-1$ 时刻遍历完整棵子树并且回到根。

# Nastya and Time Machine

# Nastya and Time Machine

答案即为 $\max d_i$。顺着这个思路构造就行了。每一个点设一个 cur 表示当前到了第几个时刻。

时间复杂度 $O(n)$。