

正睿OI NOIP Day1 题解

作者：正睿OI杜瑜皓，南师大附中杨天祺，南京外国语学校陈孙立

旅行

注意到每经过一条新的边，必定访问了一个新的点。除了起点外，恰好访问了 $n - 1$ 个新的点，也就是整个过程只会经过 $n - 1$ 条边。这意味着经过的边形成了一棵生成树。

若原图就是一棵树，我们考虑如何计算最优字典序。首先，我们一定选择 1 号点为起点。可以发现题目中的限制等价于对原图的一次dfs，形成的序列就是dfs序。因此我们只需要确定访问孩子的顺序使得dfs序字典序最小，这显然只需要按照编号从小到大访问孩子即可。算上排序，对树求答案的复杂度就是 $O(n \log n)$ 。

如果 $m = n$ ，根据之前的结论一定有某一条边没有经过，可以枚举这是哪一条边，如果剩下的图是一棵树，就按照树的方法做一遍，然后更新答案。复杂度 $O(n^2 \log n)$ 。

不过我们并不需要每次都对节点排序，只需要开始时对每个节点的相邻节点排序即可，复杂度为 $O(n^2)$ 。

填数游戏

由于 n, m 地位相同，假设 $n \leq m$ 。当 $n = 1$ 时答案显然是 2^m ，因此假设 $n \geq 2$ 。

用 $w_{i,j}$ 表示第 i 行第 j 列的数。根据题目的要求首先可以得到条件1： $w_{i,j} \leq w_{i-1,j+1}$ 。另外，条件2是如果 $w_{i,j} = w_{i-1,j+1}$ ，那么从 $(i, j+1)$ 到 (n, m) 的所有路径对应的字符串都应当相等，这个条件可以通过小样例推算得出。不难发现这两个条件也是充分的：考虑任意两条路径，它们开始会有一段公共部分，接着分开，然后到某个位置（可能是终点）再次汇合。那么对于第一次汇合点 (i, j) 之前的这一段所代表的字符串，由条件1可知它们是满足字典序关系的，如果它们不相等就已经满足条件。否则，由于我们是第一次汇合，一定有 $w_{i-1,j} = w_{i,j-1}$ ，那么根据条件2，无论这两条路径后面是怎么样子的，最终得到的字符串都一定相等。

于是问题转化为求有多少个 01 矩阵同时满足上面两个条件。注意到，对于条件1，相当于我们只关心每条左上->右下的对角线上，第一个1的位置。而对于条件2，如果 $w_{i-1,j} = w_{i,j-1}$ ，只需要满足在左上角是 (i, j) ，右下角是 (n, m) 的矩形中，任意一条对角线上的数都相等。注意到如果 (i, j) 满足这个性质， $w_{i+1,j} = w_{i,j+1}$ ，且 $(i+1, j)$ 和 $(i, j+1)$ 对应的矩形也满足这个性质。不难发现这个转化是等价的。

考虑按照对角线从左往右进行递推， $f(i, j)$ 表示第 i 条对角线，且当前需要满足上述性质的矩形集合是 j 的方案数。这里，根据上面的推理， j 需要记录当前对角线每个位置是否需要满足这个性质。由于当前对角线的填法影响的是下一条对角线，还需要记录下一条对角线的每个位置。因此状态数的上界是 2^{2n} 。转移的方法是直接枚举这一条对角线上的填法，由条件1可知不同的填法只有 $O(n)$ 个，因此这样递推的复杂度就是 $O(2^{2n} mn)$ 。递推的过程中需要较多的边界处理。

然而这样的复杂度过高。不过我们估计的只是一个粗略的上界，可以通过计算发现真正有用的状态数很少。当 $n = 8$ 时有用的状态只有 28 个，因此递推的复杂度降到 $O(28 \cdot mn)$ 。当然，我们还可以继续优化：注意到当 $m > n$ 时，中间一部分的转移是完全一样的，只有开头 n 个和结尾 n 个需要特殊处理。可以用一个 28×28 的矩阵表示这个转移。我们只需要做 $O(n)$ 次递推，中间的过程用矩阵快速幂加速，复杂度 $O(28n^2 + 28^3 \log m)$ 。

也可以通过找规律得到简单做法。若令 $g(n, m)$ 表示答案，可以通过上面的递推得到 n, m 较小的 g 值。可以发现当 $m > n$ 时， $g(n, m+1) = 3g(n, m)$ ，因此只要求出 $g(n, n)$ 和 $g(n, n+1)$ 的答案。这样只需要用原本的 $O(2^{2n} nm)$ 的递推打出表即可。

保卫王国

首先考虑单组询问且没有额外限制，那么问题并不困难，可以用一个树形dp来解决，令 $dp[u][0/1]$ 表示 u 这个子树， u 这个点选或者不选的最小代价。转移的时候考虑 u 这个点选或者不选，以及儿子选或者不选即可。

对于有额外限制的时候，我们只要在做到限制的点的时候处理一下，比如说把不合法的状态的 dp 值设为无限大即可。这样我们得到了一个 $O(nm)$ 的算法，可以获得44分。

考虑一下A类数据，即一条链的情况，这个时候我们可以用线段树来解决，对每个区间 $[l, r]$ ，维护 l 这个点选或者不选， r 这个点选或者不选四种情况的最小代价，合并的时候枚举一下中间这个点的选法即可。每次我们限制了一组点 $x, y (x < y)$ ，那么我们只要分别求出 $[1, x], [x, y], [y, n]$ 的答案，然后合并即可，时间复杂度为 $O(n \log n)$ ，结合前面的算法可以获得68分。

对于一般树的问题，我们考虑一个更加简单的问题，如果我们只是限制了一个点，那么怎么解决？

容易发现这个问题可以用一个 dp 解决，如果这个点必须选，那么它的邻居可以选或不选。如果这个点不能选，那么它的邻居必须选。答案就是删除这个点之后它所有邻居的子树的对应的 dp 值加起来。这里我们用到的 dp 信息都是某一棵子树的信息，所以可以通过修改上面的 dp （也就是要写一个从上往下转移的动态规划），将所有的答案预处理出来。

现在我们考虑原问题，有两个点 a, b 限制，如果我们确定了 a 到 b 这条链上所有点的选法，那么剩下的答案可以直接计算。我们可以这么想，把这条链从树上删去，整棵树变成了一堆子树，链上的每个点可以确定这些子树要不要被选入。比如链上相邻三个点是 u, v, w ， v 在树上的邻居是 $u, w, v_1, v_2, \dots, v_k$ ，如果 v 被不被选入，所有的 v_1, v_2, \dots, v_k 必须被选入。

我们可以把问题理解成这样，对于 a, b 这条链，我们要确定上面每个点要不要选，选取这个点的代价我们额外需要加上的邻居的 dp 值。对于上面的例子，我们记 $dp_2[x][0/1]$ 表示 x 这个子树， x 必须选/可以选或不选的最小代价。那么我们选入 v 的代价为 $p_v + \sum dp_2[v_k][1]$ ，如果不选，那么代价为 $\sum dp_2[v_k][0]$ 。

所以当路径长度不大的时候，我们可以想办法把这条路径上所有点的代价算出来。对于这条链，我们要求相邻的点至少有一个被选中，所以对这些点跑一个动态规划即可。注意到这个树形 dp 记录的信息是可以减的，所以对于一个点的额外代价就相当于把它的所有邻居的信息加起来然后减去链上的两个邻居即可。所以单组询问的时间复杂度是关于路径长度线性。可以额外获得B和2子任务的答案，结合前面的算法可以获得80分。

现在我们考虑如何快速维护上面所得东西。对于一个点 u ，将它的父亲记作 v ， v 的父亲记作 w ，如果一条链是从 u 往上走到 w ，那么 v 这个点的代价为它所有邻居的信息加起来减去 u 和 w ，我们可以把这个代价记到 u 到 v 这条边上。对于 a 到 b 的询问，记 c 是 a 和 b 的最近公共祖先，那么询问可以理解成将点 a ， a 到 c 这条链， c 这个点， c 到 b 这条链和点 b 这五个部分的信息合并起来(这里的信息合并可以理解成对 a 到 b 这条链做动态规划，将 dp 信息合并起来)。问题到这里就变成了树上链加的操作，我们可以用倍增解决，记 $info[u][k]$ 表示将 u 往上走 2^k 步的信息合并起来的结果，递推一下即可。总的时间复杂度是 $O(n \log n + q \log n)$ ，可以获得100分。

其实这个题可以用离线求LCA的Tarjan算法优化到 $O(n\alpha(n))$ ，留给大家自行思考。