calc

考虑按被全部染黑的时间依次写下行的编号，显然 $n!$ 种情况出现概率是一样的，只有 $n, n-1, \cdots, 1$ 是合法的。列同理。

答案为 $\frac{(nm)!}{n!m!}$。

tree

令 $f(i,j)$ 表示 i 节点子树在 j 时刻的剩余蛤蟆数量， 可以得到：

$f(i,0)=siz_i, f(i,j)=\sum_{k\in son_i} f(k,j-1)$。

所以 $f(i)$ 这个序列是其儿子对位相加之后再在最前方加上一个数。 所以考虑暴力合并儿子的 f（只记录 $f(i,j)>0$ 的时刻的值）。 每次将短的合并到长的里面， 就可以得到 i 的 f 序列。 根据长链剖分， 这里的复杂度是 $O(n)$ 的。

但是不能暴力将贡献加到答案里面。 考虑对于一个点的修改， 提前加上其到根节点的链上的答案， 这是一个区间加。 还需要将子节点提前加的答案减去， 这是一个区间减。 用差分数组实现即可。

总时间 $O(n)$。

graph

将不等式变为 $(p-1)t\le pr\le pt$， 维护对于所有合法的 pr 的区间， 发现只会有 $P=\log_{\frac{p}{p-1}} V$ 段（V 是值域）。 具体的， 在计算一个点的信息时用类似于归并的方法将其前继节点的合法区间合并起来，原区间 $[l,r]$ 变成了 $[l+(p-1)w, r+pw]$。 一次合并的复杂度也是 $O(P)$ 的。

查询时可以直接二分。

总时间 $O((n+m)P+q\log P)$。

greedy

考虑重链剖分。

对于每个点， 维护：

- f_i 表示其轻子树内的点到 i 的距离的最小值。
- g_i 表示其轻子树内的点到 i 所在重链链顶的最小值。
- h_i 表示将 i 到所在重链链顶的边的权值取相反数之后 g_i 的值。

可以发现修改 u,v 这条链相当于反转 $1,u$ 以及 $1,v$ 这两条链。

考虑从下往上修改。 对于一条重链， 可以发现要修改的的是一段前缀。对这条重链的影响如下：

- f_i 没有变化。
- 修改的前缀 g,h 交换。
- 未修改的后缀 g,h 分别加上一个值。

这可以使用线段树维护。 为了求得加上的是多少还需要维护链取相反数链和的操作， 这是容易的。

从一条重链跳到另一条重链时， 需要支持单点修改 f， 这也是容易的。

接下来考虑查询操作。

对于 2 操作，分两种情况讨论：

- u 在 x 子树内。
- u 在 x 子树外。 答案是 u 到 x 的距离加上 x 到 x 子树内的最小距离。 这可以转化为前一种情况。

考虑第一种情况如何处理。 假设 x 子树距 u 最近点为 v：

- 若 v 在 u 子树内， 可以发现答案是 u 点所在重链下端的一点的 f 值加上其到 u 距离的最小值。 这容易通过 g_i 来求得。
- 若 v 不在 u 子树内。 考虑从 u 往上跳（枚举 LCA）。类似的， 可以通过维护 h_i 求得答案。

两种情况取 $\min$ 即可。

对于 3 操作 也是两种情况。

- 若 x 子树与 y 子树不相交， 那么答案就是 x 到子树内的最小距离加上 y 到子树内的最小距离加上 x 到 y 的距离。 这可以通过上面的方法求得。
- 否则二者相交。 不妨假设 y 是 x 的祖先。 继续分为两种情况：
  - 若一点在 x 子树内一点不在。 那么答案就是 x 到子树内的最小距离加上 x 到 $sub(y) - sub(x)$ 这个集合内的距离。 上文已经解决过这个问题。
  - 否则两点都在 x 子树内。

    还是考虑枚举 LCA。 若 LCA 在 x 所在重链上， 这有可能是一个点的两个轻儿子到 LCA 的最近距离之和， 可以在修改跳重链时顺便维护。 否则就是重链上选两点 x, y（x 在 y 上方）， 将 $h_x + g_y$ 用来更新答案。 这可以用线段树容易的维护。

    否则 LCA 在某重链上点的轻子树内。 这可以通过实时维护每条链的顶端答案来做到。

可以发现总时间复杂度为 $O((n+q)\log^2 n)$。

实现时有一定细节。