Problem A. 赛博朋克 2077

显然，如果某个时刻我们控制的节点数小于等于 p，我们只能同时瘫痪这些节点。而这种情况下，我们一次瘫痪的节点必然是一整层的所有节点。因此设层数为 d，每次只需要判断 2^{d-1} 和 p 的大小关系即可。

一旦控制的节点数超过 p，我们接下来的每次操作（除了最后一次）就必然能恰好瘫痪 p 个节点，因为每当我们瘫痪一个非叶节点，我们控制的节点总数就会 $+1$，而我们必然会先瘫痪所有非叶节点，才会尝试去瘫痪叶节点。因此，瘫痪剩余节点需要的时间就是 $\lceil \frac{s}{p} \rceil$，其中 s 是剩余节点数。单次复杂度 $O(h)$。

Problem B. 美味星球

考虑 $m = n - 1$ 的部分分，由于图是联通的，它必然是一棵树。在这棵树上，考虑宽度最小的一条边。（思考：为什么会想到要考虑它？）对于这条边，存在一个结论：灰球君必然会完整地吃完它的某一侧（在这之前完全不吃它的另一侧），随后通过这条边，并不再回头。为什么呢？首先，不管用什么策略，只要最后能吃完，必然会存在某个时刻，灰球君恰好吃完了这条边的某一侧，并且仍然能通过这条边。并且，由于这条边是宽度最小的边，只要我们能通过这条边，就一定能在被吃完的那一侧自由运动（因此优先吃完那一侧不会受到任何阻碍），而如果先吃了若干另一侧的食物，就会导致我们通过这条边时受到更多的限制。

因此，对于宽度最小的边，设它的宽度为 w，我们依照上述策略，考虑它的影响。不妨称它的两侧分别为左边和右边，分别用 l 和 r 表示。有两种方案可选：先吃完左边或先吃完右边。对于先吃完左边的情况，我们起始的直径 s 需要满足条件：能吃完左边；能在吃完左边后通过这条边；能在吃完左边后吃完右边。翻译成数学语言就是，$f_x \le f_l$；$f_x + s_l \le w$；$f_x + s_l \le f_r$，其中 f_x, f_l, f_r 分别表示整棵树的答案，递归求解的左边和右边的答案，s_l 和 s_r 则表示左边和右边所有食物让灰球君增长的直径和。实际上，只要灰球君能在吃完左边后通过这条宽度最小的边，它就一定还能在左边自由运动，也就是说 $f_x \le f_l$ 的条件会被自动满足，不需要额外判断（虽然标程还是判了，但是不判也对）。因此，最终有 $f_x \le \min(f_r, w) - s_l$。但是这是先吃左边的情况，对于先吃右边的情况，我们对称地可以得到 $f_x \le \min(f_l, w) - s_r$。这两个条件只需要满足其中一个即可，所以最终得到：
$f_x = \max(\min(f_r, w) - s_l, \min(f_l, w) - s_r)$。

如果对于整棵树，先找最小的边，再将它切断，递归到两边进行操作，复杂度会比较高。考虑递归到两边之后的操作，仍然是每次找到最小的边继续递归，直到将整棵树切成 n 个点为止，单点的 f 值显然是 $+\infty$。那么，我们完全可以反其道而行之：从 n 个点的状态出发，每次找到**最大**的边，连接两个子块，在这个过程中不断地求出新的连通块的答案。这个过程用并查集维护连通块会十分方便。

这个时候，我们突然发现，刚刚的做法和最大生成树的 Kruskal 算法非常接近。考虑一张图的情况：假设图中出现了一个环，那么环上宽度最小的边是没有用的。因为我们想从这一侧到达另一侧的话，完全可以从环上剩余的边走。因此，只需要求出原图的最大生成树，问题就转化为了 $m = n - 1$ 的情况。（实际上求最大生成树的过程和前面的算法完全可以合并）

Problem C. 古墓丽影：崛起

Source: [「JOISC 2020 Day1」建筑装饰 4]

有一个 $O(n^2)$ 的朴素 DP：设 $f_{i,j,0/1}$ 表示当前在第 i 座山峰，已经经过了 j 座来自 A 山脉的山峰，当前落脚的山峰属于 A/B 山脉是否可行。转移非常简单，只是为了输出方案，需要记录每个状态的前驱状态。

一个邪门的优化是，打表发现，对于某个 i，可行的 j 是一段区间。因此对于每个 $i, 0/1$，记录区间端点，可以 $O(1)$ 转移出 $i+1, 0/1$ 的区间。

正经一点的做法（其实和上面的思路很像，只不过带了一些证明），设 $f_{i,0/1}$ 表示当前在第 i 座山峰，当前落脚的山峰属于 A/B 山脉，最多能经过多少座 A 山脉的山峰； $g_{i,0/1}$ 表示当前在第 i 座山峰，当前落脚的山峰属于 A/B 山脉，最多能经过多少座 B 山脉的山峰。如果根本无法在第 i 座山峰的 A/B 山脉落脚，对应的 f, g 值则为 $-\infty$。那么，有解当且仅当 $f_{2n,0} \geq n, g_{2n,0} \geq n$，或 $f_{2n,1} \geq n, g_{2n,1} \geq n$。

必要性很好证明：如果**最多**经过的山峰数都不足 n，那显然是无解的。

充分性则可以通过构造来证明。我们从一个虚构的，第 $2n+1$ 座山峰开始，倒推回之前的每一座山峰。假设我们当前在第 $x+1$ 座山峰，已经经过了 u 座 A 山脉的山峰和 v 座 B 山脉的山峰，那么有 $x+u+v=2n$。我们判断能否倒推回 A 山脉的第 x 座山峰的条件是：山峰高度符合递增的要求；$f_{x,0} \geq n-u$； $g_{x,0} \geq n-v$。B 山脉同理，下标 0 替换为 1 即可。

现在，我们只需要证明，A 山脉和 B 山脉的第 x 座山峰中，必然存在一个满足条件的。

不妨假设我们当前在 A 山脉的第 $x+1$ 座山峰，根据之前选择山脉的条件，我们知道 $f_{x+1,0} \geq n-(u-1)$，且 $g_{x+1,0} \geq n-v$。注意，如果 $f_{x+1,0} = n-(u-1)$，这说明我们已经找到了一种满足条件的方案：直接沿着 $f_{x+1,0}$ 转移来的路线倒推，最后一定刚好经过了 n 座 A 山脉的山峰。$g_{x+1,0}$ 同理。因此，我们后面的讨论认为，$f_{x+1,0} > n-(u-1), g_{x+1,0} > n-v$。

现在对两条山脉的第 x 座山峰与 A 山脉的第 $x+1$ 座山峰的高度关系讨论。

- 由于我们当前就在 A 山脉的第 $x+1$ 座山峰，$f_{x+1,0}$ 和 $g_{x+1,0}$ 必然不是 $-\infty$，也就是说至少有一条山脉的第 x 座山峰高度小于等于 A 山脉的第 $x+1$ 座山峰。
- 如果仅有 A 山脉的第 x 座山峰高度小于等于 A 山脉的第 $x+1$ 座山峰，由 DP 转移的条件可知，$f_{x+1,0} = f_{x,0}+1, g_{x+1,0} = g_{x,0}$。因此，当我们选择倒退回 A 山脉的第 x 座山峰时，有 $f_{x,0} = f_{x+1,0}-1 > n-(u-1)-1 = n-u, g_{x,0} = g_{x+1,0} > n-v$，满足了之前的条件。
- 如果仅有 B 山脉的第 x 座山峰高度小于等于 A 山脉的第 $x+1$ 座山峰，类似的思路可知选择倒推回 B 山脉的第 x 座山峰也满足之前的条件。
- 如果两座山脉的第 x 座山峰高度都小于等于 A 山脉的第 $x+1$ 座山峰，不妨假设 A 山脉的第 x 座山峰高度大于等于 B 山脉的第 x 座山峰高度，那么我们会发现一个事实：但凡第 $x-1$ 座山峰高度小于等于 B 山脉的第 x 座山峰的，都一定小于等于 A 山脉的第 x 座山峰。也就是说，只要 $f_{x-1,0/1}$ 和 $g_{x-1,0/1}$ 能转移到 $f_{x,1}$ 和 $g_{x,1}$，就一定也能转移到 $f_{x,0}$ 和 $g_{x,0}$。这有什么用呢？这说明，$f_{x,0} \geq f_{x,1}+1, g_{x,0}+1 \geq g_{x,1}$。这又有什么用呢？由于 $f_{x+1,0}$ 可以同时从 $f_{x,0/1}$ 转移，我们有 $f_{x+1,0} = \max(f_{x,0}, f_{x,1})+1 = f_{x,0}+1 > n-(u-1)$，得 $f_{x,0} > n-u$；而 $g_{x+1,0} = \max(g_{x,0}, g_{x,1}) \geq g_{x,0}+1 > n-v$，移项可得 $g_{x,0} > n-v-1$，即 $g_{x,0} \geq n-v$。因此，在这样的条件下，倒推回 A 山脉的第 x 座山峰是完全可行的。如果 A 山脉的第 x 座山峰高度小于 B 山脉的第 x 座山峰高度，类似地，也可以推出倒推回 B 山脉的第 x 座山峰也是可行的。

因此，我们证明了，以这样的方式回推，必然能构造出一组解。

Problem D. 艾尔登法环

由于植物初始高度均为 0，将所有植物按生长速度排序后，任意时刻它们的高度也是单调不降的。由于植物的高度是单调不降的，对于每次修剪，我们可以二分出第一棵在此次修剪中被实际剪了（也就是高度达到了被剪的位置）的植物。

有了思路，我们尝试将它具体实现出来。对于所有植物，初始的标记都是 0（也就是从来没被修剪过），这样，我们确定了被修剪的植物范围（必然是某个 x 到 n 的连续段）之后，就可以将这一段都打上本次修剪的标记。由于每次打标记会覆盖之前的标记，而且如果本次的左端点小于之前某次的左端点，之前那次修剪的标记就会被完全抹除。因此，我们只需要用一个栈，维护所有现存标记的左端点。每次加入新的左端点的时候，就将所有大于它的标记左端点全都出栈即可。在出栈的同时，我们可以统计这一连续段的植物被剪下来的长度和（本段的标记相同，因此从上一次修剪以来经过的时间、高度初值等全部相同，可以直接算出对答案的贡献）。另外，我们在二分第一棵被实际剪了的植物时，可以直

接对当前栈顶的左端点判定是否被剪了，如果被剪了则说明第一棵在这里或更靠左的位置，可以依照上面的方式直接将栈顶出栈并统计答案；如果没被剪，说明更靠左的位置也不会被剪，直接在栈顶代表的连续段里二分找到这个位置即可。

出入栈和统计答案，每次修改最多只会贡献一次端点。因此均摊复杂度为 $O(1)$，总复杂度为 $O(m\log m)$，瓶颈在于每次询问的二分。