Problem A. 军训 I

可以证明当 $k > 13$ 时无解。

首先，如果把人都集中在一个角落后，那么最多只有 4 种本质不同的方案；同时，对于一个角落来说，最多只有两种方案（如对于左上角而言，上左、左上可能是不一样的）。只操作一次有 4 个方向。一次都不操作有一种。因此最多只有 $4 \times 2 + 4 + 1 = 13$ 种方案。

对于 $k = 1 \sim 13$ 分别考虑。通过打表可以发现只有 $k = 8, 10, 12$ 是一定不行的。

首先可以特判掉 $n = 1$ 或 $m = 1$ 的情况。

对于 $n, m \geq 3$ 的情况（对于 $n = 2$ 或 $m = 2$ 也是类似分类，但需要注意无解的情况），通过打表/人工构造解决：

- $k = 1$：全部填满即可；
- $k = 2$：第一行或第一列不填；
- $k = 3$：某不相邻的两行/两列不填；
- $k = 4$：只填 $(1, 1)$ 即可。
- $k = 5$：

  通过分析可以发现，在 $k = 5$ 时合法必须满足进行了任意一次操作后，再进行另一个方向的任何操作都将没有意义（即所有的行的个数相等、所有的列的个数相等）。

  设 A_i 表示行 i 的个数、B_j 表示列 j 的个数。加入一个点相当于给某 A_i, B_j 同时加 1，最后要使得所有的 A_i 相等、所有的 B_j 相等。那么显然人数为 $\mathrm{lcm}(n, m)$ 的倍数。而当 $\mathrm{lcm}(n, m) = nm$ 时，就必须填满，此时是不合法的。

  因此，在 $k = 5$ 时，当且仅当 $(n, m) \neq 1$ 有解。

  构造方法如下：

  ```
  n=6, m=8   n=6, m=9
  *-*-*-*-   *--*--*--
  -*-*-*-*   -*--*--*-
  *-*-*-*-   --*--*--*
  -*-*-*-*   *--*--*--
  *-*-*-*-   -*--*--*-
  -*-*-*-*   --*--*--*
  ```

- $k = 6$: 只填 $(1, 2)$ 或 $(2, 1)$ 即可。

  ```
  -*-
  ---
  ---
  ```

- $k = 7$:

  ```
  -*****
  *-----
  ------
  ```

- $k = 9$：只填 $(2, 2)$；
- $k = 11$：填 $(1, 2), (2, 1), (1, 4)$。
- $k = 13$：只填 $(1, 3), (3, 1)$ 即可。

Problem B. 军训 II

结论：只要按照大小顺序排，那么**不整齐度**就会最小。

证明：

首先，不妨设 a_i 互不相同（如果相同则考虑给每个数加上一个很小的随机数，或者说随便给一个顺序）。

题目要求 $\sum_{l=1}^{n}\sum_{r=l}^{n}\max(a_l,\cdots,a_r)-\min(a_l,\cdots,a_r)$ 最小，那么可以将上述限制转换为：

1. $\text{Minimize}_a \sum_{l=1}^{n}\sum_{r=l}^{n}\max(a_l,\cdots,a_r)$
2. $\text{Maximize}_a \sum_{l=1}^{n}\sum_{r=l}^{n}\min(a_l,\cdots,a_r)$

这两种情况本质上是一种。考虑取 a_i 的相反数，改写第二种情况：

1. $\text{Minimize}_a \sum_{l=1}^{n}\sum_{r=l}^{n}\max(a_l,\cdots,a_r)$
2. $\text{Minimize}_a \sum_{l=1}^{n}\sum_{r=l}^{n}\max(-a_l,\cdots,-a_r)$

下面证明，如果要取到 $\sum_{l=1}^{n}\sum_{r=l}^{n}\max(a_l,\cdots,a_r)$ 的最小值，那么 a 中的最大值一定在 1 或者 n。

考虑如何计算这个值。考虑对于每个位置 i，找到左边第一个比他大的位置 l_i、右边第一个比他大的位置 r_i，那么，就可以算出一个位置的**管辖范围**，答案即为 $\text{cost}=\sum_{i=1}^{n}a_i\times(i-l_i+1)\cdot(r_i-i+1)$，其中**管辖范围**为 $h_i=(i-l_i+1)\cdot(r_i-i+1)$。

那么如果最大值所在位置不在 1 或者 n，将其移至 1 或 n 后，最大值的**管辖范围**将会变成 n，其余位置的**管辖范围**是不减的。那么 cost 的变化量就是将最大值的**管辖范围**分一些给其他位置，显然是更小的。

因此，要使得 $\sum_{l=1}^{n}\sum_{r=l}^{n}\max(a_l,\cdots,a_r)$ 最小，那么 a 中的最大值一定在 1 或者 n。将这个结论带入至两个限制中，归纳一下就可以得出结论。

因此，将数组顺序或者倒序排序即可，答案可以直接暴力或 $O(n)$ 计算。对于方案数，统计一下每个数的出现次数 cnt_i，那么方案数就是 $2\prod cnt_i!$。特别地，当所有数都一样的时候，答案为 $n!$。

Problem C. 种树

DP 做法

考虑树形 DP，对于一个子树而言，最多可能有 4 种情况：贡献 1 个给父亲、不需要父亲贡献、父亲贡献 1 个给该子树、父亲贡献 2 个给该子树。

那么，设 $f_{u,-1/0/1/2}$ 分别表示子树 u 对应的这些情况。转移时考虑枚举 u 所有儿子 v 的情况，注意一下 u 自己本身是否已经种上树、以及两个 $f_{v_1,1}$、$f_{v_2,1}$ 可以合并（即使用一次操作完成）的情况即可。实现细节可能较多。

贪心做法

对于一次操作，最多完成两个未完成的节点。

对于一个大小为 siz 的子树，且只有根节点完成了，那么一共需要 $\lfloor\frac{siz}{2}\rfloor$ 次操作。

对子树根节点是否完成进行分类讨论：

- 如果根节点已经完成了，那么这个子树对根节点父亲的贡献就有两种情况：用最少次数做完子树时，是否存在多余的、且对子树根节点的父亲有贡献的操作。

- 如果根节点没有完成，则直接将 siz 向上传递。直到遇到一个已经完成的祖先，在那个祖先中对 siz 进行统计即可。

总的时间复杂度为 $O(n)$。

Problem D. 编码器-解码器

如果求的是一个字符串 S 在 T 中出现次数，那么可以使用矩阵乘法去做。具体地，设 $A_{j,k}$ 表示某个串匹配 $S[j,k]$ 的方案数。那么一个字符 c 的贡献就可以用一个矩阵 F_c 表示。最终求得 $F_{S_1} \times F_{S_2} \times \cdots \times F_{S_n}$ 即可求出答案。

那么对于本题，也可以使用类似做法，记 G_i 表示 S_i' 所表示的矩阵，那么有 $G_i = G_{i-1} \times F_{S_i} \times G_{i-1}$。直接递推即可。

时间复杂度为 $O(|T||S|^3)$。

Problem E. 随机过程

对于最大节点数，考虑贪心。考虑深度为 i 的节点最多能有多少个。那么如果 $26^i \le n$，贡献为 26^i；否则贡献为 n。

枚举所有的层，则答案为：

$$\sum_{i=0}^{m} \min\left(n, 26^i\right)$$

对于期望节点数，显然每一层中所有节点出现在 Trie 中的期望是一样的。考虑计算第 i 层某个特定点出现的概率，然后最后乘上总个数并求和即可。对于第 i 层某个节点，其不出现的概率为 $(1-\frac{1}{26^i})^n$。

因此答案为：

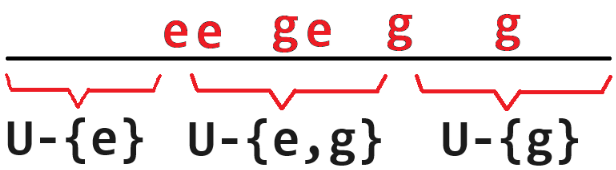
$$\sum_{i=0}^{m}\left[1-\left(1-\frac{1}{26^i}\right)^n\right]26^i$$

Problem F. 包子鸡蛋 III

考虑一个字符串什么时候是好吃的。考虑任意一个只包含 e 和 g 的好吃的串，可以先把前导 g 和后置 e 删除，然后再把倒数两个 g 中间的 e 删掉（只保留有用的 e），那么这个串的长度是 $O(m)$ 级别的。

例如，对于串 gee*g*ege*eg*e*，保留有用的 e 和 g 后，变为 eegegg。

对于 $[l,r]$ 来说，如果是好吃的，那么有唯一一个上述的串 a，满足 a 为 $s[l,r]$ 的子序列。在这个子序列的左侧可以放 g 和其他字符、右侧可以放 e 和其他字符，中间可以放除了 e 和 g 之外的字符，倒数两个 g 中间也可以放 e 字符：



记 f_n 表示一个长度为 n 的字符串好吃的概率，则答案为 $\sum_{i=1}^{n} f_i(n-i+1)$。

假设一共有 cnt 个上述的字符串 $a_1, \cdots, a_{cnt}$，并设每个串的出现概率为 $P_1, \cdots, P_{cnt}$。

枚举是哪个串 i 出现在区间里，该串第一个字符（一定是 e）到倒数第二个字符（一定是 g） 的长度为 m，第一个字符前的长度为 l，有：

$$f_n = \sum_{i=1}^{cnt} P_i \sum_{m=|a_i|-1}^{n} \left(\sum_{l=0}^{n-m-1} (1-p_e)^l (1-p_g)^{n-m-l-1} \times (n-m-l) \times p_g \right) \times \binom{m-2}{|a_i|-3} \times (1-p_g-p_e)^{m-|a_i|}$$

$$f_n = \sum_{m=1}^{n} \left(\sum_{l=0}^{n-m-1} (1-p_e)^l (1-p_g)^{n-m-l-1} \times (n-m-l) \times p_g \right) \times \left(\sum_{i=1}^{cnt} P_i \binom{m-2}{|a_i|-3} \times (1-p_g-p_e)^{m-|a_i|} \right)$$

可以发现，乘法左侧只和 $n-m$ 有关，可以通过递推或卷积算出；右侧只和 m，以及每个串的长度和概率有关。因此若可以算出只保留有用 e 和 g 的情况下，长度为 i 的串出现概率之和，记为 g_i。上式右侧即可通过 g_i 卷积算出，最后再卷积算出 f 即可。

g 可以 dp 解决。设 $dp_{i,j,k}$ 表示考虑长度为 i 的串，目前有 j 个 g 时，一共 k 个 egg 的方案数。转移枚举选 e 或选 g 即可。

总时间复杂度 $O(m^{2.5} + n \log n)$。如果 DP 部分实现的不好，$O(m^3)$ 的 DP 应该也是可以通过的。

实际上（采访一路向北后），DP 部分精细实现可达到 $O(m^2)$；对于卷积部分，可以通过组合数学技巧推式子优化为 $O(n)$。

Problem G. 疯狂星期六

因为 yyq 要尽量多花钱，所以 yyq 的总花费可以首先计算得出，其取决于 yyq 的零花钱以及 yyq 吃的菜品的价钱和。即 V_1 + yyq 吃的菜的总和、a_1 的较小值。算出 yyq 的总花费后，即可根据每个人零花钱的情况和 yyq 的总花费，算出每个人在菜品中最多花多少钱。

计算出以上信息之后，即可建图跑一遍最大流：

- 源点向每一个菜品连容量为菜品费的边；
- 菜品向食用的两个人连容量为菜品费的边；
- 每个人向汇点连容量为每个人最大花费的边。

存在合法解当且仅当最大流等于菜品的总费用。

Problem H. 另一个游戏

首先约定一些表示方式。

opt_i 表示第 i 回合的技能类型，1 表示攻击技能，2 表示辅助技能。

calc1(l, r) 表示区间 $[l, r]$ 中的攻击技能个数，即 $\sum_{i=l}^{r} [opt_i = 1]$

calc2(l, r) 表示区间 $[l, r]$ 中的辅助技能的 a_i 之和，即 $\sum_{i=l}^{r} [opt_i = 2] \times a_i$

最终得分的形式为 $d \times (x \times k1 + y \times k2)$，即 $d \times x \times k1 + d \times (n-x) \times k2$

假如对于 $\forall i \in [1, n]$ 能求出，使用 i 次攻击技能的情况下，造成总伤害 d_i 的最大值，那么相当于有 n 个点对 $(d_i \times i, d_i \times (n-i))$。

而 q 次询问即每次给出一个点对 (k_{1i}, k_{2i})，问其和这 n 个点的点积最大是多少。

考虑点积的几何意义，一个向量在另一个向量上的投影长度 和 另一个向量的长度 的乘积。

那么用一条斜率为 $-\frac{k_1}{k_2}$ 的点由上往下去切那 n 个点，第一个碰到的点即最优解。可能要特判 $k_2 = 0$。

接下来考虑如何求得 d_i。

可以通过观察/打表/大胆猜测等方式得到一个结论。

假设用 i 次攻击的最优解，其使用辅助技能的位置集合为 S_i。即 $S_i = \{p|opt_p = 2\}$

那么一定存在一个 S_i，可以由 S_{i-1} 加上一个元素得到。

也就是说，使初始全部使用攻击技能，然后每次找到替换成辅助技能后，d 的增量最大的位置，将其替换为辅助技能，一定是最优解。

怎么证明呢?

首先，当 i 从攻击技能变成辅助技能后，其对于 d 的增量为 $a_i \times \text{calc1}(i+1,n) - \text{calc2}(1,i-1)$

假设 S_i 是根据上文策略选出的辅助技能位置集合，S_i' 是另一个不符合上文策略，且能获得更大的 d 的辅助技能位置集合

找出满足 $x \in S_i, x \notin S_i'$ 的最小的 x，以及满足 $y \notin S_i, y \in S_i'$ 的最小的 y，即不在对方策略的，最前辅助技能的位置

当 $x < y$ 时，则选 x 比选 y 的收益多出:

$$\begin{aligned}&[a_x \times \text{calc1}(x+1,n) - \text{calc2}(1,x-1)] - [a_y \times \text{calc1}(y+1,n) - \text{calc2}(1,y-1)]\\=\ & a_x \times \text{calc1}(x+1,y) - (a_y - a_x) \times \text{calc1}(y+1,n) + \text{calc2}(x,y-1)\end{aligned}$$

首先，若 $a_x > a_y$，则式子非负，将 y 换成 x 不劣。

否则，记 Δ 为 S_i 情况下多出的收益，Δ' 为 S_i' 情况下多出的收益。

$A = a_x \times \text{calc1}(x+1,y), B = (a_y - a_x) \times \text{calc1}(y+1,n), C = \text{calc2}(x,y-1)$。

则 $\Delta = A - B + C, \Delta' = A' - B' + C'$。

由 S_i 中选择了 x 而不是 y 可知 $\Delta \geq 0$。

由 x, y 的选取方式且 $x < y$ 可知 $A \leq A', B \geq B', C \leq C'$，即 $\Delta' \geq \Delta \geq 0$

故 S_i' 将 y 换成 x 不劣，$x > y$ 时，类似推算可得相同结论。

以此类推，最终 S_i' 被替换为 S_i，即 S_i 更劣的假设不成立，故 S_i 即为最优策略的一种。

现在只需要考虑每次怎么找到最优的位置，将其换成辅助技能

观察一下增量的形式 $\Delta = a_i \times \text{calc1}(i+1,n) - \text{calc2}(1,i-1)$

不妨考虑分块，记第 i 个块的左右端点分别为 l_i, r_i，将式子变形为:

$$-\text{calc1}(r_i+1,n) \times a_i + \Delta = -\text{calc2}(l_i,i-1) + \text{calc1}(i+1,r_i) \times a_i - \text{calc2}(1,l_i-1)$$

因此，在每个块内维护 $(a_i, -\text{calc2}(l_i,i-1) + \text{calc1}(i+1,r_i) \times a_i)$ 构成的凸包。

$\text{calc1}(r_i+1,n)$ 和 $\text{calc2}(1,l_i-1)$ 对于同一个块来说是相同的。

$\text{calc2}(1,l_i-1)$ 仅为整体平移，不影响块内的凸包形状。

$\text{calc1}(r_i+1,n)$ 是单调的，故直接用指针在凸包上移动找最优的修改位置即可。

找到全局最优的修改位置 p，将其由攻击技能变成辅助技能后，还要修改其对于其他位置的影响。

对于 i 所在的块，更新每个点的 y 坐标并重新建凸包，对于其他的块，修改 $\text{calc1}(r_i+1,n)$ 和 $\text{calc2}(1,l_i-1)$ 即可。

假设块长为 B，每次找到新的最优位置，并修改的复杂度为 $O(B + \dfrac{n}{B})$，

取 $B = \sqrt{n}$，总时间复杂度 $O(n\sqrt{n})$。

根据赛时提交，出题人发现自己非常 naive，没有意识到 $O(n \log n)$ 的做法。

大概思路应该是，用平衡树维护只有 $1 \sim a_i$ 时，这 i 个回合，由攻击技能变为辅助技能的最优顺序。

然后二分出 a_{i+1} 的插入位置即可。

Problem I. 找行李

有一个很好的性质，先将人和行李排序，如果第 i 个人选了行李 j，则第 i 个人往后的人选 j 以前的行李，对答案一定是没有贡献的，则有个朴素dp：$f_{i,j,k}$ 表示前 i 个人，最后一个被选的行李是 j，在 j 之前有 k 个行李还没有被选，则每次转移枚举选哪个行李即可（或者不选）。时间复杂度 $O(n^4)$。

但是并不好优化，换一个思路：令 $g(d)$ 表示答案大于等于 d 的方案数，则答案为 $\sum_{d=1}^{\infty} g(d)$，因为一个方案的答案为 x，则会被 $g(1)$ 到 $g(x)$ 分别统计一次。

考虑求一个 $g(d)$，每个人可以选的行李个数是确定的，就是左边距离他大于等于 d 的行李们，且第 i 个人能选的行李集合一定是第 $i+1$ 个人的子集，则可以dp：$f_{i,j}$ 表示前 i 个人选 j 个行李的方案数，转移只需要考虑选和不选。时间复杂度 $O(n^3)$。

Problem J. 找最小

先求出两个序列分别的异或和记为 A, B，则修改位置 i 相当于让两者同时异或 $a_i \oplus b_i$，则可以将所有 $a_i \oplus b_i$ 构造线性基。接下来从 A, B 的高位往低位考虑，对于一位：

1. 如果都是 1，且线性基这一位有值，则同时异或线性基这一位，都变成 0 肯定更优。
2. 如果都是 0，则不管。
3. 如果一个 1 一个 0，后面肯定让这一位是 1 的尽可能小，0 的不用管，则用线性基贪心即可。如果这一位线性基有值，那么需要枚举是否异或这个值，两种情况都往下贪一次即可。

时间复杂度为 $O(n \log w)$，其中 w 为值域。

Problem K. 取沙子游戏

做法：若 $\mathrm{lowbit}(n)$ 小于等于 k ，则先手胜，否则后手胜。

证明：

- 若 $\mathrm{lowbit}(n)$ 小于等于 k ， 则先手取 $\mathrm{lowbit}(n)$ ，之后后手无论取什么数，先手只需重复后手的取数即可。
- 若 $\mathrm{lowbit}(n)$ 大于 k ， 则先手无论取何数字，取之后的 lowbit 一定小于等于 k ，此时后手必胜。

Problem L. 网络预选赛

直接枚举这连续的两行和两列，然后暴力判断即可。

时间复杂度 $O(nm)$。