1001

先考虑 $B=1$ 的时候怎么做，考虑二分答案。

$$
\begin{aligned}
mid &\le \frac{sum_r - sum_{l-1}}{r-l+1+bias} \\
(r-l+1+bias) \times mid &\le sum_r - sum_{l-1} \\
sum_{l-1} - (l-1) \times mid &\le sum_r - r \times mid - bias \times mid
\end{aligned}
$$

其中 $sum_n = \sum\limits_{i=1}^{n} a_i$。

若不等式成立，则增大 mid，否则减小 mid。

这个式子把 r 和 $l-1$ 分离了，对于每次 check，我们可以 $1 \sim n$ 遍历一遍，实时记录最小的 $sum_{i-1} - (i-1) \times mid$，并与当前的 $sum_r - r \times mid - bias \times mid$ 比较，判断是否满足条件，若满足则 check 成功。

现在考虑如何处理上取整。

对于一个左端点 l，他的最佳右端点 r 必定有：

$$
r \in \{l + k \times B - 1 | k \in [1, \lfloor \frac{n-l+1}{B} \rfloor]\} \cap \{n\}
$$

因为 $a_i > 0$，所以在不改变分子的情况下右端点要尽量靠右，因此在除了 $r=n$ 的情况下，区间长度肯定是 B 的倍数。

我们可以枚举最佳区间左端点 l 模 B 的余数，不妨令 $l \bmod B = m$。

在二分 check 时只需要遍历同样模 B 余 m 的点和 n 就可以了。

时间复杂度：$O(B \times \lceil \frac{n}{B} \rceil \times \log V) = O(n \log V)$。其中 V 为值域。

Trick：小数二分建议直接设置二分次数，而不是使用 eps。

1002

$f_{i,j}$ 表示考虑前 i 个顾客的所有子集中，价值为 j 的集合个数。发现这就是经典的 01 背包但是价值计算方式是乘模。

发现转移过程有结合律，于是可以用快速幂或者倍增优化，先算出前 f_n 再考虑合并，$f_{n+n,(j \times k) \bmod P} = \sum f_{n,j} \times f_{n,k} \bmod 998244353$，时间复杂度 $O(nP + P^2 \log Q)$。

注意到保证 P 是质数，所以可以用原根将乘法转换为加法。假设 g 是 P 的原根，把 $[0,p)$ 变成 g 的幂指数，即 $g^x \to x$, 令 $mp_{g^x \bmod P} = x \bmod (P-1)$, 则原本 $(j \times k) \bmod P \to (mp_j + mp_k) \bmod (P-1)$。

对于每个顾客有母函数 $x^{mp_1}+x^{mp_{a_i}}$, 可以用 NTT 做循环卷积，最后每一项的系数就是 `res[i]`，复杂度为 $O(nP\log P+P\log P\log Q)$

但其实前 n 个多项式非常稀疏，并且之前也提到这是一个 01 背包，所以应该先 dp 预处理前 n 个顾客，然后用循环卷积每次合并 n 个顾客，时间复杂度 $O(nP+P\log P\log Q)$。

1003

这道题最初是作为一道纯贪心题设计的。如果你熟悉一些高级的字符串算法，比如 SA 或 Lyndon Word 等，可能会更容易解决，但这里我们不讨论这些内容。

首先，我们先处理一些特殊情况：

- 当 $k=1$ 时，答案一定是 **YES**，因为只需要划分成一段即可。
- 当 $k>r-l+1$ 时，答案一定是 **NO**，因为划分的段数超过了字符串的长度范围。

接下来，我们假设存在一个可行的划分方案，那么将字符串划分成 k 段一定是更优的。我们可以通过枚举第 k 段的开头字符 c 来进一步讨论。

情况 1：第 k 段长度为 1

如果第 k 段的长度为 1，那么前面的 $k-1$ 段字符串的字典序都必须小于 c。换句话说，我们需要在字符串的区间 $[l,r-1]$ 中找到 $k-1$ 个字符，这些字符的字典序都小于 c。如果能找到，那么这种情况是可行的；否则，不可行。这个判断过程相对简单。

情况 2：第 k 段长度大于 1

如果第 k 段的长度大于 1，我们希望在前面的字符串中划分出 $k-1$ 段，每段的字典序都小于等于 c。这样可以尽可能多地选择候选字符 c 作为最后一段的开头。前面的 $k-1$ 段可以分为以下几种情况：

1. 字典序小于 c 的段，可以直接单独划分。
2. 字典序等于 c 的段，如果它的后继字符的字典序不大于 c，也可以单独划分。
3. 字典序大于 c 的段，需要与前面的字符合并成一个字符串，直到找到字典序小于 c 的字符。

通过上述分类，我们可以利用二分查找来确定前面 $k-1$ 段至少需要使用多少字符。接下来，在字符串的后半部分查找是否存在字符 c，使得第 k 段的长度大于 1。

现在，我们需要将前面 $k-1$ 段的长度扩展到第 k 段。我们发现，只要前面 $k-1$ 段中存在字典序小于 c 的字符，就可以任意扩展这些段的长度。因为在从第 $k-1$ 段到第 k 段开头的区间内，不存在字符 c，否则这部分就会被划分到第 k 段。

还有一种特殊情况：前面 $k-1$ 段中没有任何字符的字典序小于 c。例如，当 $k=5$，字符串 $S=$ `ccccaabcbacabb` 时，前 $k-1$ 段都是字符 c。这种问题可以转化为：判断字符串是否能划分成两段，使得第二段的字典序更大。

我们可以通过以下步骤解决这个问题：

1. 将字符串按照字符 c 划分成若干段。例如，对于字符串 $S = \texttt{c|c|c|caab|cba|cabb}$，每一段都是一个新的元素。
2. 使用单调栈找到每个元素最早大于自身的位置。这里可以直接枚举每一位进行比较。由于每次相同的子串都需要倍长，因此对于一个字符，求解的复杂度是 $O(n \log n)$。请注意这里的大小比较为，判断能否把前面的当做第一段，后面的当做第二段，使得第二段字典序比第一段大。
3. 或者，可以维护一个 lcp 数组，表示单调栈内相邻字符串的最长公共前缀长度。当比较新字符串时，前面相同的部分就不需要重新比较了。

这样依次递归，预处理每个字符开头的情况。

预处理的复杂度为 $O(|\Sigma| n \log n)$，求解的复杂度为 $O(|\Sigma| q \log n)$，因此，总复杂度为 $O(|\Sigma|(n+q) \log n)$，其中 $|\Sigma|$ 表示字符集的大小。

1004

我们发现，只有在一段连续的首字符才会被修改。那我们用线段树在首字符的位置维护这段的长度。然后用线段树二分的方法找到最早出现大于等于 k 的位置。注意每次翻转可能会和前后拼接起来的细节。

时间复杂度 $O(q \log n)$。

1005

如果不存在边双联通分量，你可以直接在树上找一条直径。

首先对于一个点数大于 1 的边双连通分量，我们可以从随意一个点开始进入这个边双连通分量收集这个边双中的所有点的宝物并返回。

如果存在若干个边双连通分量，那么这些联通分量中的点还有这些它们路径上的点，都是可以从任意一个点出发收集所有的这些点上的宝物并返回任意一个点。你可以通过并查集合并边双连通分量或者你可以用 bfs 的方式将外部的孤立点一个个剥离。

接下来的问题，你可以从边双作为根然后在树上找一条路径 $u \to lca \to root \to lca \to v$，其中 lca 表示 u, v 的最近公共祖先。这可以在 dfs 的时候合并当前节点作为 lca 的答案。

时间复杂度 $O(n) \sim O(n \log n)$。

1006

考虑最小割。

我们用点 $node_{i,j}$ 表示第 i 户村民，选择第 j 栋楼房。

在不考虑 k 个限制条件的情况下，我们这么建图：

- $node_{i,j-1}$ 往 $node_{i,j}$ 连一条流量为 $s_{i,j}$ 的边（割掉这条边代表让 i 住进 j 号楼）

- 源点 S 往 $node_{i,1}$ 连一条流量为 $s_{i,1}$ 的边（割掉这条边代表让 i 住进 1 号楼）
- $node_{i,m}$ 往汇点 T 连一条流量为 ∞ 的边（不能割这条边）

现在考虑如何添加这 k 个限制条件。

对于一组限制 (x,y,d)，我们添加如下边：

对于任意满足 $dis(u,v-1)\le d$ 且 $dis(u,v)>d$ 的点对 (u,v)，不妨令 $u<v$，其中 $dis(u,v)$ 表示 u 号楼和 v 号楼之间的距离。

- $node_{x,v}$ 往 $node_{y,u}$ 连一条流量为 ∞ 的边
- $node_{y,v}$ 往 $node_{x,u}$ 连一条流量为 ∞ 的边

如果我们选择了不满足限制条件的两条边比如我们让 x 住进了 u，y 住进了 $v+1$，那么就存在一条 $s\to node_{y,v}\to node_{x,u}\to T$ 的增广路，这就不是一个割。

如此就能够完成建图。

总点数：$nm+2$，总边数：$n(m+1)+2km$

1007

令 dp_u 表示从 u 出发只考虑 u 这个子树内的节点，最多获得多少分。转移为

$$dp_u=\sum_{v\in \text{son}(u)}\max(0,dp_v+p+q)$$

从 s 开始 dfs，对于在 s 到 t 路径上的点 u，对于 u 的儿子 v，如果 v 的子树内不包含 t，那么 ans_u 直接加入 $\max(dp_v+p+q,0)$ 的贡献，否则加 ans_v+p 的贡献。

时间复杂度 $O(n)$。

1008

把吉他与键盘看作同一种职业计算乐队个数，再减去两个键盘组成的乐队个数即可

$$ans=a\times b\times c\times (C_{d+e}^2-C_e^2)$$

时间复杂度 $O(1)$。

1009

将项目看作是一条连接 a_i,b_i 的边，对于一个连通块最多可以选择小于等于点数的项目。将所有项目按照价值从大到小排序，用并查集维护一下连通块已经有的点数和边数，如果点数已经等于边数就不加入。

时间复杂度 $O(m\log m+n\log n)$。

1010

令 $a'_i = a_i - i$ 且 $b'_i = b_i - i$

分析变换后的约束条件：

1. $0 \le a'_1 \le a'_2 \le \cdots \le a'_N \le M - N$.
2. $0 \le b'_1 \le b'_2 \le \cdots \le b'_N \le M - N$.
3. $a_i < b_i$.

这个问题等价于计算形状为 (N, N)（即两行，每行 N 个格子）的半标准杨表的数量，其中杨表的格子填充的数字来自集合 $\{0, 1, \ldots, M - N\}$

一个形状为 λ 的 SSYT 是一个在 λ 的格子上填数字的方式，使得：

- 每行内的数字非递减。
- 每列内的数字严格递增。
- 数字来自指定的集合。

在我们这个问题中，令杨表的第一行为 $a'_1, a'_2, \ldots, a'_N$，第二行为 $b'_1, b'_2, \ldots, b'_N$.

$$T = \begin{pmatrix} a'_1 & a'_2 & \ldots & a'_N \\ b'_1 & b'_2 & \ldots & b'_N \end{pmatrix}$$

条件 $0 \le a'_1 \le \cdots \le a'_N \le M - N$ 和 $0 \le b'_1 \le \cdots \le b'_N \le M - N$ 满足行非递减的要求。

条件 $a'_i < b'_i$ 满足列严格递增的要求。 数字来自集合 $\{0, 1, \ldots, M - N\}$，该集合的大小为 $M' = M - N + 1$.

对于形状为 $\lambda = (N, N)$，使用 M' 个可用数字 $\{0, 1, \ldots, M' - 1\}$ 填充的 SSYT 的数量可以通过 hook-content formula 计算。 公式为：

$$\prod_{c \in \lambda} \frac{M' + \mathrm{content}(c)}{\mathrm{hook}(c)}$$

其中 c 代表杨表中的一个格子，$\mathrm{content}(c)$ 是格子的内容（第 j 列第 i 行的格子内容是 $j - i$），$\mathrm{hook}(c)$ 是格子的钩子长度（格子本身 + 右边的格子数 + 下边的格子数）。

对于形状 $\lambda = (N, N)$:

- 格子 $(1, j)$ 的内容是 $j - 1$, 钩子长度是 $(N - j) + 1 + 1 = N - j + 2$. ($j = 1 \ldots N$)
- 格子 $(2, j)$ 的内容是 $j - 2$, 钩子长度是 $(N - j) + 0 + 1 = N - j + 1$. ($j = 1 \ldots N$)

$$\prod \mathrm{hook}(c) = \prod_{j=1}^{N} (N - j + 2) \times \prod_{j=1}^{N} (N - j + 1) = (N + 1)! \times N!$$

$$\begin{aligned}\prod(M'+\text{content}(c)) =& \prod_{j=1}^{N}(M'+j-1)\times\prod_{j=1}^{N}(M'+j-2)\\ =& \frac{(M'+N-1)!}{(M'-1)!}\times\frac{(M'+N-2)!}{(M'-2)!}\end{aligned}$$

所以 SSYT 数量，即满足条件的方案总数 $W(N,M)$ 为

$$\begin{aligned}W(N,M) =& \frac{(M'+N-1)!(M'+N-2)!}{(M'-1)!(M'-2)!}\times\frac{1}{(N+1)!N!}\\ =& \frac{M!(M-1)!}{(M-N)!(M-N-1)!}\times\frac{1}{(N+1)!N!}\\ =& \frac{M!}{N!(M-N)!}\times\frac{(M-1)!}{N!(M-N-1)!}\times\frac{N!}{(N+1)!}\\ =& C(M,N)\times C(M-1,N)\times\frac{1}{N+1}\end{aligned}$$

预处理阶乘，然后组合数直接计算即可。

时间复杂度 $O(n+m)$。
