Easy

F 最甜的小情侣

考虑没有修改的操作。

设计一个简单的 DP，$f(i,0/1/2/3,0/1/2/3)$ 表示当前考虑位置 $[1,i]$，首尾分别选了几个数。

枚举开头三个数的选择情况作为 DP 的起点，DP 中途注意不能连续选超过 3 个。

得到 $f(n,x,y)$ 后用 $x+y\le 3$ 的更新答案即可。

加入修改操作只需要将 DP 状态放到线段树上，因为首尾的选择个数是非常好合并的。和最大子段和的动态做法没有本质区别。

时间复杂度 $O(n\log n)$，自带 16 倍常数，不过时限已经开的很足了。

H 最完全的替换

简单贪心题。

考虑最高位的 1 如果没有被替换过，那么必定需要用 b 的最高位的 1 对齐它去操作一次。否则要么会产生更高位的 1，要么这个 1 永远不会变成 0。

换句话说，不考虑重复操作的情况下，操作的情况是唯一确定的，从高位到低位模拟即可。

复杂度 $O(nm)$。

I 最努力的活着

显然是让活的尽量久的人获得更大的价值更优。

而假设当前轮还剩 n 个人，那么实际会有 $\lfloor n/w\rfloor$ 个人被淘汰。这些人具体的位置我们不关心。

剩下的就是推式子的问题了，需要用到 $\sum i$ 和 $\sum i^2$ 的求和公式。

值得注意的是答案是不会超过 V^3 即 10^{36} 的，所以 `__int128` 足够通过这道题，无需高精度。

对于复杂度，考虑分 $w\le\sqrt{n}$ 和 $w>\sqrt{n}$ 讨论，前者 n 的衰减速度很快，后者 $[1,n]/w$ 下取整只有至多 $O(\sqrt{n})$ 种。故总复杂度为 $O(\sqrt{n})$。

对这题一开始数据过弱再次表示抱歉喵 >_<

K 最多变的序列

考虑什么样的结果是可能被得到的。

发现对于序列中某个值 c 出现的位置，一定是连续的一段区间 $[l,r]$。且这个区间原本的 $\min$ 恰好为 c。

针对这样合法的序列统计，只需要从左到右考虑当前 a_i 能覆盖的一段区间即可。

`DP` 状态可以随意设计，只要按照上述说法去设计 `DP` 都能通过。

时间复杂度 $O(n^2)$。

Medium

A 最遥远的路

数据范围中点数足够少，也就是说我们可以设计 `DP` 时基于点数较为暴力的维护。

具体的，假设没有边权限制，单次询问 $[1,\inf]$。可以考虑设 $f(u,v)$ 表示 $u\to v$ 的最长路，将边按照权值从小到大排序，依次转移。单次转移复杂度 $O(n)$。

同时，将边的权值从大到小排序，仍然可以做到单次转移 $O(n)$，只是枚举起点还是枚举终点的问题。

这样我们就可以考虑分治了。

将问题离线，针对边的权值分治，每次考虑边权限制 $[l,r]$ 跨过 mid 的询问。发现在这个 `range` 中合法的边一定是，mid 往左的一段后缀，和往右的一段前缀。

根据上面讲的 `DP` 暴力维护即可。

较精细的实现可以做到 $O(n(m\log m+q))$。

D 最糖的题目

结论题，而且其实 `shift` 操作无论针对子串还是针对子序列结果都是一样的。

首先要讨论一些 `corner case`：

- 当 $k=1$：a,b 必须初始就完全相等才合法。
- 当 $k=n$：a,b 必须循环相等，利用最小表示法判断循环串是否相等即可。
- 当构成 a,b 序列元素的可重集本来就不一样，那一定是 `NO`。

考虑 a,b 都是 n 的排列的情况。

`shift` 操作等价于，将序列中某个元素，移动到 $k-1$ 位之前。

我们考虑统一将 a,b 序列修改到，各自能达到的，字典序最小的序列。

对于序列 a，假设 1 并不在 a_1：

- 如果 1 在位置 a_k 就直接换到 a_1。

- 如果 1 在位置 $a_i, i > k$，那么将 1 往前换就会让 i 变小，重新开始讨论。
- 如果 1 在位置 $a_i, i < k$，那么进行一次后面元素的提前，可以让 1 移动到 a_{i+1}。

之后 $a_1 = 1$，继续考虑 $a[2..n]$ 的子问题。

这个贪心做法什么时候会失效呢，不难发现是只剩最后 2 个元素的时候。

下面我们证明到最后两个元素无法改变相对位置时，a, b 的表出不一样，那么两者就不能互相表达：

- 注意此时逆序对上有奇偶性的区别，一个是 0 一个是 1。
- 注意当 k 为奇数时，每次 `shift` 操作都无法改变序列逆序对数的奇偶性。
- 注意当 k 为偶数时，可以构造一些列变换使得实现相邻元素 `swap`。

所以对于一般情况：

- 若 a, b 的元素集合中，有元素出现 > 1 次。那么无论 k 的奇偶性，都可以重标号使得 a, b 逆序对数奇偶性一致，所以必定 `YES`。
- 若 a, b 的元素集合元素都只出现一次，那么 a, b 离散化后等价于 n 的排列。
  - 若 k 为奇数：当且仅当 a, b 序列的逆序对数奇偶性相同时，输出 `YES`。
  - 若 k 为偶数：必定为 `YES`。

时间复杂度 $O(n \log n)$，瓶颈是求逆序对数。

E 最自律的松鼠

考察树的直径的必经边。

如果题目并非动态，而是单次询问。

这是经典的结论（参考 `[SDOI2013]直径`），可以随便找一条直径，必经边显然必定在这条直径上。再考虑非直径边，考虑直径上每个点出发，走非直径边的最远距离。如果 直径上某个点的最远距离 与 它到直径左端点的距离 相等，那么显然这个点往左都不会是必经边。往右同理。

在排除掉这种情况后，实际上中间没被排除的一个区间就都是直径的必经边了。

而这里的动态操作本质上对做法没有什么影响：

- 增长直径会导致我们维护的区间的右端点立刻移动到直径最右端。
- 而针对添加叶子的操作，在节点处统计它到直径上点的距离，时刻维护直径上到左右端点距离，比较是否相等，若相等对应的移动 $[l, r]$ 端点即可。

时间复杂度 $O(n)$。

J 最好的位置

一个经典结论是直径的中点是重合的（可能在边上），被称为树的中心。

这里我们选择的点显然就是树的中心，如果在边上就随便选择这条边的某个端点。

所以唯一问题是动态维护直径长度。

又一个经典结论是，如果当前直径为 (a,b)，新加入的点为 c，那么新的直径只会是 $(a,b),(a,c),(b,c)$ 之一（考虑反证法，利用树上任意两点间路径唯一的性质，不难证明），直接比大小判断即可。

时间复杂度 $O(n\log n)$，瓶颈是求树上两点间距离要求 `lca`。

Hard

B 不最近的路

首先考虑最短路怎么求解。

发现路径和定义为前 k 大边还是非常棘手的，但是 n,m 范围都不是很大。

考虑枚举第 k 大边的边权到底是多少，花费 $O(m)$ 的时间。

假设当前枚举的值为 v，将所有边权改为 $w' \leftarrow \max(0,w-v)$，并做最短路，将得到的结果加上 $k\times v$。下面证明这样做能够得到正确答案：

- 当枚举的值恰好为第 k 大边权时，算法确实得到了正确的结果。
- 当枚举的值比第 k 大边权小：会导致路径多计算更小的边 $-v$ 的边权和，结果只可能会偏大。
- 当枚举的值比第 k 大边权大：会导致统一加上的 $k\times v$ 偏多（本质是将 $<v$ 的前 k 的边视为 v 了），结果只可能会偏大。
- 若路径边数 $<k$，枚举 $v=0$ 的时候讨论到了这种情况。

总而言之，最优解一定会被统计到，且没有错误的“更优解”出现。

而对于次短路，只需要记录一下最短路的 `path`，用一样的方法再次求解，但是避免与最短路完全重叠的 `path` 即可。（具体实现上只需要多记录 `pre` 数组并在转移时加上判断）

复杂度 $O(m^2\log m)$。其中 $O(m)$ 是外层枚举，$O(m\log m)$ 是单次最短路。

C 最中间的数

倒着考虑问题，并且每两个选手为一对。

考虑最后一对选手的能力值 (l,r)，假设 $l\le r$。并且坚持到最后的第三个人的能力值为 x。

那么最终胜者为：

- 若 $x<l$，那么能力值为 l 的选手获胜。
- 若 $x>r$，那么能力值为 r 的选手获胜。

- 若 $x \in (l, r)$，那么能力值为 x 的选手获胜。

我们发现，对于这一对 (l, r)，本质上就是将来的值 x 进行一个 `round`，如果 x 在这个区间内它就能胜出，否则 (l, r) 中的一个会胜出。

再考虑倒数第三和第四个人，$(l', r'), l' \le r'$。

我们发现坚持到只剩 5 个选手时，最左边的 x 必须在 $(l', r') \cap (l, r)$ 的范围内才会胜出。也就是说这个区间 `round` 操作的区间是可以求交的。如果此时 $(l', r') \cap (l, r) = \varnothing$，那么胜者一定在这四个人中产生，和左边的情况没有关系，这个胜者可以简单的通过讨论得到。

所以从后往前将每两个选手配对后，变成了区间求交的问题，具有可合并性。利用权值线段树动态的维护配对情况即可。

线段树 `merge` 时讨论略微有些繁琐，不过也是这题实现中唯一的难点了。

时间复杂度 $O(n \log n)$。

G 最绝望的 hidesuwa

考虑一种暴力做法。

对每个位置维护并查集，对于一个输入 $[l, r]$，$\forall i \in [l, r]$，将 a_i 与 $a_{r-(i-l+1)+1}$ 的集合 `merge`，表示它们必须一样。

最后输出 26^{cnt}，其中 cnt 表示不同集合个数。

输出方案只需要简单的 `dfs` 即可，并且由于只有 20 个其实只有字典序最大的集合才会从 $a \sim t$ 变换。

对于优化，考虑实际有效的 `merge` 次数只有至多 $n-1$ 次。所有有效 `merge` 就是此次操作真的将本来不在同一个集合的两个位置合并了。所以我们有理由相信 `merge` 的次数是可以被优化的。

这是一个叫倍增并查集的科技。

对于 $[l, r]$，只考虑 $l < r$ 的情况，设 $x = \lfloor (r-l+1)/2 \rfloor$，实际上就是 $a[l, l+x-1]$ 与 $a[r, r-x+1](reverse)$ 这两段子串的对应位置要 `merge`。

于是考虑建立每个长度为 2^i 的子串的并查集，将 `merge` 操作拆分为若干个长度为 2 的整次幂的区间。

在所有区间都拆分完成之后，再从大到小去将所有 `merge` 操作推到长度为 2^0 的区间，即：

- 假设当前有两个长度为 2^i 的区间 $[l, r], [l', r']$，它们被 `merge` 到一起了，要求对应位置相等。
- 那么它就可以推到 2^{i-1} 的区间，将这两个区间继续细分为两个子区间，这两个子区间对应相等。

由于字符串有正反顺序，需要建立表示 正 和 反 的两种并查集，最终将长度为 2^0 的正反区间合并即可。

总体复杂度 $O(n\alpha(n)\log n)$。这个 $\alpha(n)$ 是并查集的复杂度，反阿克曼函数，在当前数据范围中 $\alpha(n)\leq 5$，可以认为是常数。

K 最有节目效果的一集

大模拟。难度主要是代码实现而非思维。

每个小组内部用平衡树维护分数的变化，再对每两个队伍维护一个链表表示当前关于他们的消息。

这里平衡树推荐用 `c++` 自带的 `pbds`，不过手写肯定也是可以的。

复杂度 $O(n\log n)$。