A. Yasaka with Postfix Expression

算法：表达式值的计算（栈的基本应用方法）

（1）前缀表达式：前缀表达式又称波兰式，**前缀表达式的运算符位于操作数之前；**

比如: $- \times + 3\ 4\ 5\ 6$

前缀表达式的计算机求值过程如下：

从右至左扫描表达式，遇到数字时，将数字压入堆栈，遇到运算符时，弹出栈顶的两个数，用运算符对它们做相应的计算（栈顶元素 op 次顶元素），并将结果入栈；重复上述过程直到表达式最左端，最后运算得出的值即为表达式的结果

- 例如: $- \times + 3\ 4\ 5\ 6$
- 从右至左扫描，将 6 、5 、4 、3 压入堆栈
- 遇到 $+$ 运算符，因此弹出 3 和 4 (3 为栈顶元素，4 为次顶元素，注意与后缀表达式做比较），计算出 $3+4$ 的值，得 7 ，再将 7 入栈
- 接下来是 $\times$ 运算符，因此弹出 7 和 5 ，计算出 $7 \times 5 = 35$ ，将 35 入栈
- 最后是 $-$ 运算符，计算出 $35 - 6$ 的值，即 29 ，由此得出最终结果

（2）中缀表达式：中缀表达式就是常见的运算表达式，如 $(3+4) \times 5 - 6$

此处应有py！！！

```
str=input()
print(eval(str))
```

有兴趣的同学可以去查找一下中缀表达式的计算过程，不同于其它两种表达式，中缀表达式需要2个stack进行维护，一个存计算结果，一个存计算符号！

（3）后缀表达式：后缀表达式又称逆波兰表达式,与前缀表达式相似，**只是运算符位于操作数之后；**扫描顺序是从左至右！**过程省略！**

此题采用的是后缀表达式计算过程！

B. Yasaka with Physical Examination

算法：组合数+快速幂

利用排列组合思想来解题。直接求相邻两个位置状态相同的方案数不好解决，因为连续 2 个位置、连续 3 个位置，...，连续 k 个位置状态相同的都有连续两个位置状态相同，比较复杂。这里可以先求 k 个位置 n 中状态的所有情况，很容易推出：

$sum_1 = n^k$

k 个位置相邻两个位置不相同的有:

$sum_2 = n * (n-1)^{k-1}$

答案 ans 就等于 $sum_1 - sum_2$

因为有指数运算，这里就可以使用快速幂。基本原理如下：

假设我们要求 a^b ,按照朴素算法就是把 a 连乘 b 次，这样一来时间复杂度是 $O(b)$，即是 $O(n)$ 级别，快速幂能做到 $O(logn)$。它的原理如下：

假设我们要求 a^b，那么其实 b 是可以拆成二进制的，该二进制数第 i 位的权为 2^{i-1}，例如当 $b=11$ 时，$a^{11}=a^{2^0+2^1+2^3}$

11 的二进制是 1011，$11=2^3\times1+2^2\times0+2^1\times1+2^0\times1$，因此，我们将 a^{11} 转化为算 $a^{(2^0)}*a^{(2^1)}*a^{(2^3)}$ 。由于是二进制，使用位运算实现更加简洁： $\&$ 和 $>>$。

$\&$ 运算用于二进制取位操作：例如 $b\&1$ ，表示取 b 二进制的最低位,判断和 1 是否相同，相同返回 1 ，否则返回0；也可用于判断奇偶：$b\&1==0$ 为偶数，$b\&1==1$ 为奇数。

$>>$ 运算把 b 的二进制右移一位，即去掉其二进制位的最低位。

C. Yasaka with Prime

算法：素数打表+lower_bound（二分查找）

首先筛选出符合条件的所有素数，存在一个 $vector$ 里面（假设为 in ）。对于 n 来说，可以定义一个迭代器 $it=lower(in.begin(),in.end(),n)$，这样就找到了第一个大于等于 n 的素数，如果是 n 本身，则根据题目要求输出 0；如果不是 n，则说明 n 不是素数，$ans=*it-*(it-1)$。

知识点延伸：

$lower\_bound()$ 和 $upper\_bound()$ 都是利用二分查找的方法在一个排好序的数组中进行查找的。

1）在从小到大的排序数组中：

$lower\_bound(begin,end,num)$ ：从数组的 $begin$ 位置到 $end-1$ 位置二分查找第一个大于或等于 num 的数字，找到返回该数字的地址，不存在则返回 end 。通过返回的地址减去起始地址 $begin$ ，得到找到数字在数组中的下标。

$upper\_bound(begin,end,num)$：从数组的 $begin$ 位置到 $end-1$ 位置二分查找第一个大于 num 的数字，找到返回该数字的地址，不存在则返回 end 。通过返回的地址减去起始地址 $begin$，得到找到数字在数组中的下标。

2）在从大到小的排序数组中，重载 $lower\_bound()$ 和 $upper\_bound()$

$lower\_bound(begin,end,num,greater<type>())$：从数组的 $begin$ 位置到 $end-1$ 位置二分查找第一个小于或等于 num 的数字，找到返回该数字的地址，不存在则返回 end。通过返回的地址减去起始地址 $begin$，得到找到数字在数组中的下标。

$upper\_bound(begin,end,num,greater<type>())$：从数组的 $begin$ 位置到 $end-1$ 位置二分查找第一个小于 num 的数字，找到返回该数字的地址，不存在则返回 end。通过返回的地址减去起始地址 $begin$ ，得到找到数字在数组中的下标。

D. Yasaka with WF

http://www.usaco.org/current/data/sol_coupons.html

带反悔的贪心。

先把优惠券给前 k 个 c_i 最小的用，因为使用优惠券情况下他们是必买的，其中第i个物品省下的是 $p_i - c_i$ 元。 其中有的物品可能优惠前后都很便宜，使用优惠券省的很少，因此可以花费 $p_i - c_i$ 剥夺它的优惠券给有需要的物品。

之后的考虑顺序为添加一个物品所要付出的代价，两种决策：原价购买剩余的 p_i 最小的物品 or 先付出一定代价剥夺一张优惠券，再用它购买剩余的 c_i 最小的物品，最小化每一次的花费，重复这个过程直到没钱为止。使用 3 个优先队列分别维护 p 最小，c 最小，已购买的 $p - c$ 最小。

E. Yasaka with Kanako

给两个序列 X 与 Y ，将 Y 整体加一个非负整数，使得两个序列没有重复数字。

nm 处理两序列中任意两数的差 $x_i - y_j$ ，显然 $ans = mex(\text{所有的非负}\, x_i - y_j)$

其中 mex 为序列中最小的没有出现的非负整数。

F. Yasaka with fAKe Tree

要求字典序最小的解，容易想到只需要每步取可访问的编号最小的点即可保证。

依然使用优先队列维护可选的点的集合，当前可选的点的深度为 $min(\text{未访问的点的}\, a_i)$ ，且随着未访问的点渐渐减少，可选深度是单调增加的，每次将新出现的深度的点加入备选即可。

G. Yasaka with 星間飛行

2019 ICPC 西安邀请赛 M (<>ω·)☆KIRA!

边权是假的，因为它只能限制飞船能否经过，路径上的边权的和在这题没有意义。

我们发现，如果升级 k 次是可行的 ，那么所有大于 k 的升级次数也都是可行的，也就是说答案具有单调性，所以我们可以二分答案。

如何判定升级次数 k 可不可行？

所有大于 $d * k$ 的边都不能走，反之则可以走，我们要判断 $e * k$ 次内是否能走到 n。

问题就变成了边权只有 0 和 1 的最短路，直接 BFS 即可。

存不存在无解的情况？

图是联通的，d 和 e 也都大于 0，所以我们一定能听到兰卡酱的演唱会。

Macross真的好看，星間飛行真的好听

H. Yasaka with Aimo

前置芝士：$Trie$

板子题

https://www.luogu.org/problemnew/show/P2580

这题几乎就是板子题了，答案是经过 $trie$ 的某个节点的字符串的个数乘以它的深度的最大值。

I. Sayaka with Infinity No.7

前置芝士：最小(大)生成树，倍增求 LCA

LCA 对于萌新比较不友好，建议补完并且弄懂其他题再来看此题。

模板题

https://www.luogu.org/problemnew/show/P3379

手玩一下就会发现，其实图中很多边无论如何都是不会去选择走它的。事实上我们只需要求出原图当中每个联通块的最大生成树，走的边一定是最大生成树上的。

证明：当最大生成树确定下来后，如果存在一条不在最大生成树上的边，能使答案更大，那我们一定能够把这条边替换掉路径上的某条边，使生成树更大，矛盾。$Q.E.D$ 。

求完生成树后我们该如何处理这 q 个询问？

其实原问题变成了求树上某两个点之间的链的最小值。

怎么统计链上信息？

可以树剖大力搞，但由于问题是静态的，我们直接上倍增 LCA 就行了。

在 dfs 时，我们倍增求 LCA 的 fa 数组的同时，可以一起维护 val 数组，表示当前节点跳到它某个祖先的路径上的最小值。

这样我们就得到了一个 $O(m*logm+(n+q)*logn)$ 的做法。

J. Kaguya with QAQ

对于每个 'A' 计算它前面的且与它不相邻的每个字符 'Q' 有多少个，表示它作为 'QAQ' 的第二个字符，能接在几个 'Q' 后面，记为 pre_j 。

再对于每个 'Q' 计算它前面的且与它不相邻的每个字符 'A'，累加它的 pre_j 到答案里。其意义就是这个 'Q' 作为 'QAQ' 的第三个字符时，能接在多少个 'QA' 的组合后面。

K. Kaguya with arithmetic progression

给定一个值，求首项最小的、公差为 1 的等差数列的和为这个值。

尽管首项确定时，我们能 $O(1)$ 算出它能不能成为答案。但 n 太大了，复杂度很可能退化成 $O(n)$ 的。

考虑枚举等差数列的长度，注意到长度为 n 的等差数列和是 $O(n^2)$ 的，我们只需要最多枚举 $O(\sqrt{n})$ 次就行了。

设长度为 n 时，等差数列的首项为 k，和为 sum 。

那么 $2*sum=(2*k+n-1)*n$ ，$k=\lfloor(2*sum+n-n^2)/(2*n)\rfloor$ 。

那么我们就 $O(1)$ 判断它能否更新答案了。

L. Kaguya with poetry

注意到,如果我们枚举到的位置能够组成一句诗,那么将他们计入答案,再往后枚举显然是最好的,因为后面的句子就有机会组成新的诗句，当有两个韵脚 A , B 分别出现两次后,会出现的情况是:

$AABB, ABAB, ABBA$

当 A, B 相同时: $AAAA$.

都能组成诗

所以,只需要记录下每个韵脚出现的次数,

当有两个两次后,就统计答案就行了.