string：

20%：首先我们发现多个*连在一起与一个*是等价的。我们只要枚举每个*把上一个字母复制了多少遍就行了。

60%：用 f[i][j]表示 s[1…i]能否变为 t[1…j]。如果 s[i]不是*那么 f[i][j]=f[i-1][j-1]，否则 f[i][j]=max{f[i-1][k]}，其中 t[k+1],…,t[j]=s[i-1]。

满足条件的 k 显然是一个区间，对每个 j 预处理出 k，然后用前缀和优化转移即可。

时间复杂度 O(T|s||t|)

100%：将 s 和 t 划分为若干个极长段，满足每段以字母开头且段内只有一种字母。显然这些段是一一对应的，对于 s 中每一段，如果有*，那么它可以变为字母个数>=它的全字母段。

时间复杂度 O(T(|s|+|t|))

or：

中间 30%：对于每个条件，如果 pi=0，那么 x[li]…x[ri] 显然都为 0，否则至少要有一个 1。

由于我们只需要构造出一组可行解，那么我们可以把没被要求为 0 的 x[i]都设为 1，然后判断是否满足每个条件即可。

区间赋值和区间询问可以用线段树维护。

时间复杂度 O(mlogn)

100%：由于或运算每一位是相互独立的，因此可以将上述做法推广到 pi>1 的情况。

初始时 x[i]=2^30-1，对于每个条件，将 x[li]…x[ri]中 pi=0 的位修改为 0，最后判断是否满足条件。同样使用线段树维护。

时间复杂度 O(mlogn)

shop：

20%：按题意暴力模拟即可。

时间复杂度 O(nm)。

中间 40%：首先对物品按价格排序并预处理出前缀和。对于每次询问，我们二分找出能买得起的最贵的物品 i，再二分找出能买得起的连续一段物品 i~j。

由于买下 i~j 后买不起 j+1，并且 j+1 的价格不大于 i 的价格，因此至少花费了一半的钱。那么我们只需要二分 logw 次即可。

时间复杂度 O(nlognlonw)。

100%：不难发现上述做法可以推广到每种物品个数>1 的情况，也就是二分能全部买下的一段，再求一下下一个物品能买多少个。

时间复杂度 O(nlognlonw)。