括号匹配（bracket）

考虑这样的贪心策略：从左到右依次遍历字符串，如果遇到"*"就什么都不做，遇到"("就将这个左括号入栈，遇到")"就将栈顶的左括号弹出；如果栈为空，就考虑替换掉之前遇到的"*"来与当前")"进行匹配，每次优先将最左边的"*"替换为"("，如果最左边的"*"在当前")"的右边，则一定无解。遍历完整个字符串后，栈可能还不为空，也就是说还剩一部分没有匹配的左括号，依次将栈顶元素出栈，每次将一个最右边的"*"替换为")"，如果最右边的"*"在当前栈顶左括号的左侧，则一定无解。最后剩余没被替换的"*"，全部替换成空串。显然，时间复杂度为$O(n)$。

下面证明这个贪心算法的正确性。注意到，如果有最短长度的合法解，则不会存在一个被替换为"("的"*"在一个被替换为")"的“*” 右侧。假设存在这样的情况，则交换它们不会使解不合法，但交换后可以发现它们都可以替换为空串，这意味着存在更短长度的解，从而导出矛盾。假设已经得到一个最短长度的合法解，则对于每个被替换为")"的"*"，如果其右侧存在被替换为空串的"*"，则我们可以交换这两个字符的位置，新解的字典序不会更大，对于"("同理。因此最短长度的字典序最小合法解一定是替换最左侧的一部分"*"为"("，最右侧的一部分"*"为")"，中间一部分"*"替换为空串。

本题的数据点较多，如果考场上暂时无法直接想到正解，可以先想出一些近似的贪心解法，也能拿到部分分数。

成绩排名（rank）

对于每个学生来说，分自己是否得到书（也就是A_i是否乘t）两种情况讨论：

当学生i自己没有得到书时，若要自己排位保持不变，需要从得到书后不会对自己的排位产生影响的同学中选取k个，经过简单计算后可以发现：学习能力大于A_i或者学习能力乘t后小于等于A_i的人无法对自己的排位造成影响。假设这样的人共有m个，可以预先对原数组进行排序，使用二分查找计算出m的值，那么符合答案的情况会有$\binom{m}{k}$种。

当自己得到书时，学习能力被强化为tA_i，排名上升（或保持不变），若要i的排名回到原本的位置上，则需要排名被i追赶上的所有人都重新超过他，也就是说这些人必须被分配到书来使学习能力上升，从而完成排名的反超、将i重新“挤回”原本的排名，假设这些人共有d个，如果$d \leq k-1$，则能够满足这些必须的分配，多余的书我们可以分给不会对i的排名产生影响的其他人；如果$d > k-1$则不能满足必须的分配，这种情况的答案为0。经过简单计算后可以发现：学习能力大于A_i并且小于等于tA_i的人，就是排名被i追赶上的人，也就是必须被分配到书的人，通过二分查找计算出这些人的数量d，符合答案的情况共计$\binom{n-d-1}{k-d-1}$种。

综上，将两种情况的答案数相加即为每个学生自己排名不变的情况总数，时间复杂度为$O(nlogn)$。

不会使用逆元和前缀积求组合数的同学，可以使用杨辉三角$O(n^2)$求解组合数，能拿到40%的分数。

字符串距离（distance）

首先对两种操作进行一些简单的分析，可以发现插入操作是无用的，所有的插入操作都可以通过删除操作等效替换。下面给出证明：考虑最终修改完毕的字符串，如果其中某一个位置上的字符对于原字符串A与B来说都是通过插入操作得到的，那么这两步插入其实是多余的，我们不插入这两个字符也可以将两个字符串化为相同的串；如果某一字符对于字符串A来说是原有的字符，而对于字符串B来说是通过插入操作得到的，那么这个操作其实等效于删除掉字符串A上原有的这个字符，而不会对距离的值产生影响。

所以，要求两字符串之间的距离值，也就是求如何通过删除最少的字符使得两字符串变得完全相同，显然，对于两字符串的最长公共子序列部分，可以予以保留，而其他部分必须删除。得到距离值等于$length(A) + length(B) - 2LCS(A, B)$，其中$length$表示字符串长度，$LCS$表示两字符串的最长公共子序列。因此问题转换成了：对于给定的两个字符串A和B，每次询问A上某个字串与B的LCS值

是多少。如果直接使用最简单的LCS动态规划，单次查询的复杂度为$O(nm)$，整体复杂度为$O(qnm)$，可以拿到40%的分数。此时观察数据可知，B串的最大长度仅为20，考虑从这个点入手进行优化。

设$S[l...r]$表示在字符串S上从l到r一段连续的字串。首先求出这样一个数组$f[i][j]$，它的值为在A上从第i个位置后面字符j出现的第一个位置的下标，考虑这样的动态规划状态：假设$dp[i][j]=x$，其中x为使得$B[1...i]$与$A[l...x]$存在长度为j的公共子序列的最小下标。

当$f[dp[i][j]+1][B[i]] \le r$时，状态转移方程如下：

$$dp[i+1][j+1] = min(dp[i][j+1], f[dp[i][j]+1][B[i]])$$

最后的答案，也就是所有$dp[m][i]$中满足值在$[l,r]$内的最大的i值。

预处理$f[i][j]$需要$O(26n)$的时间，每次查询会进行一次$O(m^2)$的dp，总的复杂度为$O(26n+qm^2)$

树上染色（color)

拿到这个问题的时候，可以首先对问题做一步简化，先不考虑颜色不同的情况——也就是先把所有点都看作颜色相同的点。在这种情况下，考虑计算每个点的贡献是多少。假设我们现在要计算x点的贡献值，显然直接计算异或值不太方便，所以我们可以把每个点的权值按二进制拆成20位，那么对于第i位来说，当其值为$1(0)$时，如果有树上另一个结点y值对x产生贡献，那么显然y的值为$0(1)$，并且y不在x的子树上或在x到根节点1的树链上。那么现在的问题就转换成求不在x的子树以及树链上且值与x相反的结点有多少个，将这个数量值乘以2^i就是第i位的贡献值。通过计算出总值，减去不符合条件的结点数量即可，显然对于子树部分我们可以使用dfs序+树状数组等数据结构进行维护，对于树链部分，如果我们使用树链剖分，复杂度为$O(20qlog^2n)$，能够拿到60%的分数。

注意到修改都是单点修改，我们可以构造这样一个数据结构来维护：记树链前缀和$P[x]$为从根节点1到x的树链上的所有结点的权值和，当我们需要对树链做查询时，只需取$P[x]$的值即可；如果对x点的权值做单点修改，那么其子树上的所有前缀和$P[y]$都需要更新，也就是在dfs序上的区间更新。因此，对于树链的处理，我们只需对$P[x]$构建差分树状数组，做区间更新单点查询即可，总体的复杂度降为$O(20qlogn)$。

经过观察发现，不同颜色的点之间是不构成贡献的，也就是每种颜色之间相互独立，互不影响，可以考虑离线。对于每个操作拆分成删除和插入两个新操作，对于每个点的初始状态转化成一次插入操作，共计$2q+n$次新操作，对这些新的操作按颜色排序，依次处理各颜色即可。