记开始时'o'的个数为init(o), init(x)同理.
记开始时从头起最长连续'o'的个数为left(o), 从尾起最长连续'x'的个数为right(x).
记当前'o'的个数为cnt(o), cnt(x)同理.
记当前可删除的'o'的个数为inside(o), inside(x)同理.

先确定双方的策略.

注意到无论何时, (cnt(o) - cnt(x))都为(init(o) - init(x))或(init(o) - init(x) + 1), 所以:
  (i) 若init(o) > init(x), 先手胜;
  (ii) 若init(o) = init(x), 先手胜或平局;
  (iii) 若init(o) < init(x), 后手胜或平局.
我们只关心先手胜的情况, 即(i)和(ii).

(i)
这种情况下先手必胜, 于是只要考虑最后的cnt(o).
由于后手每操作一回合, cnt(o)都会减1, 所以对于先手来说, 要尽可能减少后手的操作次数.
后手能够操作的条件是inside(o) > 0, 并且后手每次操作将造成inside(o)减少1.
故先手通过减少inside(o)来减少后手的操作次数.
先手的一次操作可能造成inside(o)
  (*) 减少: 当且仅当删去了最左边的极长连续'x'段(长度必定为1), 使得最左边两段极长连续'o'段合并, 如ooxooxoxx -> ooooxoxx;
  (**) 不变: (*)以外的情况, 如ooxxooxoxx -> ooxooxoxx或ooxxooxoxx -> ooxoooxx.
只有情况(*)才能造成inside(o)减少, 并且先手希望情况(*)尽早发生, 使得更多的'o'被合并到最左边而不是被后手删除.
所以先手的最优策略之一为每次删除最左边的'x'.
而后手希望更多的'o'被自己删除而不是被合并到最左边, 所以后手的最优策略之一为删除最左边的'o'.
记在(*)中被合并到最左边的'o'的个数为merged(o), 则最后cnt(o) = left(o) + merged(o).

(ii)
这种情况下先手要成为最后一个操作的人才能获胜.
当inside(x) <= inside(o)时, 先手可以在最后一步前不删除最左边的'x', 这样inside(o)只会在后手回合减少1, 所以能够一直保持inside(x) <= inside(o), 这样先手就能保证最后一个操作. 这种情况下, 后手的最优策略之一也是尽量不删除最右边的'o', 否则将减少操作回合, 从而使得结束时'o'更多.
当inside(x) > inside(o)时, 通过类似的做法, 后手可以保证平局.

然后考虑计数.
先枚举left(o)和right(x), 那么其余的都是inside了.

(i)
从左到右确定每个'?', 用f[inside(x)][inside(o)][merged(o)]来dp.
加入一个'x'是好处理的.
加入一个'o'需要判断它是否被合并到最左边了, 即在后手删这个'o'之前先手是否已经把其左边的'x'都删去, 即inside(x) <= inside(o) - merged(o) + 1是否成立.

(ii)
直接用(i)中得到的dp数组就可以计算答案.

注意一些边界情况.

时间复杂度: O(n ^ 5).

说下部分分.
暴力根据优劣程度可以获得（也许是）5~30不等的分数.
分析出获胜条件可以获得R = 1的20分.
分析出双方策略可以获得不含'?'的20分以及前30的暴力分（虽然觉得能分析到这一步应该就能计数了吧……）.