盒饭盲盒

签到题

a表示素菜数量，$b = n - a$表示荤菜数量，答案为$\frac{b^3}{n^3-a^3}$

万历十五年

显然是一个背包问题

对于背包问题，可以采取二进制优化提高“总权值为C”的01背包问题复杂度

为$O(c\sqrt{c}/64$） 证明可以网上找个博客贴上去

可是，对于本问题而言，总的复杂度是$O(c\sqrt{c}/64\log c$）无疑在c =1e6的时候会超时，所以需要减小log的复杂度。具体做法可以分类讨论，当节点c比较大的时候，采用以上的背包方法，c比较小的时候，采用简单dp做法，这样log的值就会非常小，总计的运算次数大约在3e8,5s内完全能够跑的完。

此外，也可以fft做，背包问题是一个多项式，转移相当于 $F(x)* = (x^k + 1)$ 其中k是当前值。fft本身带log，归并一个log，树一个log，总复杂度 $O(n\log^3 n)$ 也可以通过

攻城

签到题

重点在于需要最后一次性完成，所以有几个要求：

- 城堡总生命值s能够整除$n+6$
- 生命最少的城堡也必须大于$s/(n+6)$，即攻击次数

但是有个特殊情况，只有一个城堡的时候，特判即可。

两串糖果

首先预处理出$v[i][j]$表示区间$[i,j]$内的满意度，$revv[i][j]$表示翻转区间$[i,j]$的满意度，这里可以用$O(n^2)$的方法预处理，转移方程为：

$v[i][j] = v[i+1][j-1] + a[i]*b[i] + a[j]*b[j]$

$revv[i][j] = revv[i+1][j-1] + a[i]*b[j] + a[j]*b[i]$

接着考虑$dp[i]$表示处理到第i个位置为止的最优情况，转移方程为：

$dp[i] = max(dp[i], dp[j] + max(v[j+1][i], revv[j+1][i])), j \in [0, i)$

复杂度$O(n^2)$。

只想要保底

二分+状态压缩

先二分答案，然后对于每个答案，遍历每行，对于每行中所有符合条件的列的集合，以状态压缩的二进制形式加入到map中，然后判断是否已经存在一个map中的值，使得两行的并为全1。

复杂度$O(\log A_{max} \cdot n \cdot 2^m)$

捣乱的神

这里有一个性质，如果有三个连续的值的二进制表示的最高位相同，那么把后两个数进行异或，得到的结果一定小于第一个数，即$1xx \oplus 1xx = 0xx < 1xx$。

因此只要数字的个数超过$3 * 30 = 90$，一定只需要一次操作，数字个数较小时，可以任意选取方法，暴力遍历所有可能性即可。

归零

可以发现，对于每一组(l, r, k)，只需要分两步计算答案即可，对于$A_i \leq \frac{k}{2}$的值，显然是将其减到0划算，对答案的贡献为A_i；对于$A_i > \frac{k}{2}$的值，显然是将其加到k划算，对答案的贡献为$k - A_i$。

因此最简单的方案的复杂度为$O(n^2)$。

现在考虑优化，本质上我们只需要找出区间$[l, r]$内，大于$\frac{k}{2}$的数的数量，并且计算数据和，由于有多组(l, r, k)，因此考虑主席树。按照A_i的大小依次放入主席树，这样对于询问(l, r, k)，只需要根据k的大小确定根节点的编号，就可以找出区间$[l, r]$内小于$\frac{k}{2}$值的数的数量以及总和。

整体复杂度$O(n \log n)$。

打工人

使用$dp[i]$表示到第i个物品为止的最少次数，显然有：

$$dp[i] = min(dp[j] + sump[j][i] + 2), j \in [0, i] \&\& sumw[j][i] \leq W$$

$sump[j][i]$表示运送区间$[j, i]$货物需要的单位时间，$sumw[j][i]$表示区间$[j, i]$内的货物总重量，这两个值可以预处理得到。

对于$dp[i]$，可以使用单调队列进行优化，复杂度$O(n)$。

另类排序

看起来是一道强制在线的题目，但是可以注意到每次询问的答案会$\%7$，也就是说，$lastans \in [0, 6]$，$l \in [l' - 6, l' + 6], r \in [r' - 6, r' + 6]$，因此只需要将所有的值加入到答案，离线使用莫队算法即可。

具体细节中，使用树状数组维护数量，每次新增一个值v，求出大于该值的数的数量a、总和$suma$，以及小于该值的数的数量b、总和$sumb$，$ans' = ans + sumb + v \cdot a$。删除一个值类似。

复杂度$O(13 \cdot n\sqrt{n} \log n)$。

收集者

$dp[i]$表示以i开头的答案

- sum表示目前的答案
- $dp[i]$的更新为目前答案+1，即$sum + 1$
- sum的更新为目前答案加上新的答案，减去原先以该数字开头的答案，即 $sum + dp[i] - pre[dp[i]]$

- 其中$pre[dp[i]]$可以通过预处理得到。

复杂度$O(n)$。

乐观的R家族

签到题

对于每道题，计算每个答案的数量，取最大值，乘以a_i加入答案即可。