A. Playf and ABC

题目大意

给一个长度为n的，只包含A、B、C三种字符的字符串S。问S中包含多少个三元组(i,j,k)满足i<j<k并且$S_iS_jS_k = ABC$或CBA，且每一个字符最多使用一次。$(1 \leq n \leq 3 \times 10^5)$

解题思路

把所有的A和C交换位置，使得所有A都在C之前，然后只需要计算三元组ABC的个数即可。可以证明这样做是正确的：
记交换后得到的新字符串为S'。
这样做之后，对于任意一个满足题意的三元组(i,j,k)，必然满足$S'_i = A, S'_j = B, S'_k = C$。分几种情况讨论：
① $S_i = A$ 且 $S_k = C$，显然满足解要求。
② $S_i = C$ 且 $S_k = A$，显然满足解要求。
③ $S_i = A$ 且 $S_k = A$，既然$S'_k = C$，说明在k位置发生了交换，一定存在一个被交换的m<k满足$S_m = C$。如果m>j，只需要让k与m建立映射，表示真实的三元组是(i,j,m)；如果m<j，只需要让i与m建立映射，表示真实的三元组是(m,j,k)。此时新三元组满足解要求。
④ $S_i = C$ 且 $S_k = C$，同③。
由于A和C的交换是满足映射关系的，即交换的A和C是两两匹配的，所以③和④中找出的所有真实三元组都是不相交的。
所以答案即为S'中不相交的ABC三元组的个数。时间复杂度O(n)。

B.color

题目中说，可以选择一些有相同颜色的元素，把他们的颜色改成单调不减，使得若干次操作后，所有数组都单调不减。

可以发现每次操作，等价于删除了一种颜色，若干次操作使得剩下的元素是单调不减的。

问题就变成求一个魔改的最长上升子序列，在相同颜色的元素出现位置首次位置保存一下当前的答案，尾的位置更新一下当前的答案即可。用树状数组，线段树，或者上升子序列的二分算法都可实现。

Timetable Queries

- 方法一:
- 使用{map,unordered_map}中的一种容器,
- 嵌套{vector,deque,rope}中的一种容器,
- 以map<int,vector<int>>的形式存储某号车的所有到站时间,
- 预处理：$O(nlogn)$
- 查询：$O(qlogn)$

- Python中使用字典及序列，是相同做法。
- Java中也有TreeMap和HashMap，是相同做法。

Timetable Queries

- 方法二:
- 使用哈希存储，按哈希值对车号进行分类存储，最简单的按模1e6分类也可通过本题。
- 设哈希值为M，a_i最大值A，那么
- 预处理：$O\left(\frac{A}{M}n\right)$
- 查询：$O\left(\frac{A}{M}q\right)$
- 我们取$M=10^6$，则有$\frac{A}{M}=10^3$，又$n,q\le 10^5$，
- $\frac{A}{M}n,\frac{A}{M}q\le 10^8$可以通过本题。

Timetable Queries

- 方法三：离散化+二分，时间复杂度$O(nlogn)$，
- 留给选手自己思考。

Construct A Permutation

- 首先我们考虑求出该绝对值嵌套式的最小值$f(n)$。

- $f(n)$的前四项可以通过遍历所有排列得出:
- 是$\{1,1,0,0\}$，考虑如下构造方法:

- $f(n) = \Big|\Big||\,|f(n-4)-n|-(n-1)\Big|-(n-2)\Big|-(n-3)\Big|$
- $= f(n-4)$
- 所以一定有$f(n) \in \{0,1\}$。

Construct A Permutation

- 当$n \equiv 0 \bmod 4$或$n \equiv 3 \bmod 4$时，根据上面的构造方法，
- 一定有$f(n) = 0$，没有更小的可能。

- 但$n \equiv 1\ mod 4$或$n \equiv 2\ mod 4$时，有没有可能$f(n)$取到0？
- 答案是否定的。因为$f(n)$的奇偶性与$1 \sim n$的和相同。

- $x + y - |x - y| = \max(x, y) + \min(x, y) - \max(x, y) + \min(x, y)$
- $= 2\min(x, y)$是偶数，所以$|x - y|$和$x + y$绝对值相同。

Construct A Permutation

- 当$n \equiv 1\, mod 4$时,
- $f(n) \equiv \frac{(4k+1)(4k+2)}{2} \equiv (4k+1)(2k+1) \equiv 1 \bmod 2$

- 当$n \equiv 2\, mod 4$时,
- $f(n) \equiv \frac{(4k+2)(4k+3)}{2} \equiv (2k+1)(4k+3) \equiv 1 \bmod 2$

- 所以以上两种情况$f(n)$最小值是1。

Construct A Permutation

- 所以，$f(n)=\begin{cases}0,\ n\equiv 0\bmod 4 \text{ or } n\equiv 3\bmod 4\\ 1,\ n\equiv 1\bmod 4 \text{ or } n\equiv 2\bmod 4\end{cases}$。
- 接下来我们考虑让前面$n-1$个元素排成最小，最后一个位置放数字n，可以构造出该式最大值。

- 该式中间的计算过程都在n以内，如果能取到n一定是最大值。
- 如果按照上面的方法取不到n，只能取到$n-1$，那么最后一个元素放置其它数字并不会使结果更优。
- 因为该式最终结果上界是$\max(a[n], n-a[n])$。

Construct A Permutation

- 所以，如下构造方法是正确的。
- $\{n-1, n-2, \ldots, 2, 1, n\}$
- 已知还有很多正确的解法，不一一列出。

Contest Simulation

- 按题意模拟即可。

- 注意同题数罚时选手排名顺序。

D Sakuyalove and Fast FFT

可以证明$a_k = c_k (k = 0, 1, \ldots, n)$.

证明：

$$
\begin{aligned}
c_k &= \sum_{i=0}^{k} C_k^i (-1)^i b_k \\
&= \sum_{i=0}^{k} C_k^i (-1)^i \sum_{j=0}^{i} C_i^j (-1)^j a_j \\
&= \sum_{i=0}^{k} \sum_{j=0}^{i} \frac{k!}{(k-i)!(i-j)!j!} (-1)^{i+j} a_j
\end{aligned}
$$

其中a_j的系数为：

$$
\begin{aligned}
&\sum_{i=j}^{k} \frac{k!}{(k-i)!(i-j)!j!} (-1)^{i+j} \\
=& \sum_{i=j}^{k} \frac{k!(k-j)!}{(k-i)!(i-j)!j!(k-j)!} (-1)^{i+j} \\
=& C_k^j \sum_{i=j}^{k} C_{k-j}^{i-j} (-1)^{i-j}
\end{aligned}
$$

当$j < k$时，根据二项式定理，a_j系数为：

$$C_k^j \sum_{i=j}^{k} C_{k-j}^{i-j}(-1)^{i-j}$$
$$= C_k^j(1-1)^{k-j} = 0$$

当$j = k$时，a_j系数为：

$$C_k^k C_0^0(-1)^0 = 1$$

因此$c_k = a_k$. 直接输出$a_0 \oplus a_1 \oplus \ldots \oplus a_n$即可。复杂度$O(n)$.

J Sakuyalove and Latin Square

假设现在已经填好了前m行，要填第$m+1$行。构造二分图，左侧和右侧各n个点。左侧第i个点表示第i列，右侧第j个点表示第j个数字。如果第i列的前m行没有填数字j，就让左侧的点i与右侧的点j连边。于是填第$m+1$行相当于找到二分图的一个完美匹配。该二分图有$O(n)$个点，$O(n^2)$条边，使用dinic算法可以$O(n^{2.5})$复杂度找到最大匹配。之后每填一行就重新建一次图。总时间复杂度为$O(n^{3.5})$.

E、F stone

题目大意

给定n堆石子，第i堆石子有a_i个，每次操作可以从最左边的石子堆中移除任意个石子(至少1个)，无法操作的人输。两个人玩游戏。每人连续操作k次，问谁能胜利。
一共有T个case。

$$1 \le T \le 1e5, 1 \le \sum n \le 1e5, 1 \le a_i \le 1e9$$

解题思路(easy)

本题中，$k = 1$，即两人轮流操作。
打表或者手动模拟几次即可发现规律:
如果最左边石子堆数量为1，则操作唯一，交换先后手。
如果最左边石子堆数量大于1，则当前操作的人必胜。
如果当前没有石子，则当前操作的人必败。

具体证明会在hard中介绍。

解题思路(hard)

本题中，$1 \le k \le 1e5$。

定义$f(i,j)$表示如果第i堆只剩j个石子，并且其左侧的石子全部拿完的状态下，此时游戏开始(即此时$Alice$即将连续操作k次)的胜负关系。$f(i,j)=1$表示先手必胜，$f(i,j)=0$表示后手必胜。

结论1:

对于状态(i,j)，若操作k次之后，存在状态(i',j')，满足$f(i',j')=0$，则$f(i,j)=1$。

否则，若不存在，则$f(i,j)=0$。

解题思路(hard)

结论2:

若$f(i,j)=1$，则$\forall k>j, f(i,k)=1$

证明:

由于$f(i,j)=1$，则存在一个方案，使得先手操作k次之后，到达一个状态$f(i',j')=0$。

该方案的第一步为从第i堆移除石子至剩余t个，那么对于初始状态$f(i,k)$，第一次操作完一定能使第i堆石子也剩余t个，后续操作均一样，也转移到$f(i',j')$即可。

解题思路(hard)

因此，每个位置i只需要维护一个量$g(i)$，满足$\forall j <= g(i), f(i,j) = 0$且$\forall j > g(i), f(i,j) = 1$。考虑如何计算$g(i)$。

由结论2，显然有，$g(i) <= k$。可以考虑二分$g(i)$的值。

考虑对于某个具体位置$f(i,j)$，可以考虑从该状态开始，连续操作k次，可能转移到哪些状态，这些状态是一段连续的区间。可以利用前缀和或线段树等方式维护某段区间中是否有值为0的状态，需要注意初始化和边界。

该方法的时间复杂度$O(nlogn)$

bonus：本题可以在$O(n)$的时间复杂度内解决。

G.XOR Segments

题目概述

求满足 m 条限制的长度为 n 且元素大小 $a_i \in [0, 2^k)$ 的数组个数。

一组限制条件 l_i, r_i, x_i 表示数组需满足 $a_{l_i} \bigoplus a_{l_i+1} \bigoplus \cdots \bigoplus a_{r_i} = x_i$

正确解法

一组条件相关联的 a_i 太多了，我们可以通过处理前缀异或来简化限制条件。

令 $b_i = a_1 \bigoplus a_2 \bigoplus \cdots \bigoplus a_i$。

那么对于一组限制条件只需满足 $b_{l_i-1} \bigoplus b_{r_i} = x_i$。

而且对于一个的前缀异或数组 $b[1:n]$，也唯一对应了一个原数组 $a[1:n]$，只要求出有多少 b 数组即可。

把限制条件看成一条 $l_i - 1$ 与 r_i 之间的双向连边，边权为 x_i。

如果一个连通块内有一个点的值确定了，那么其他点的值也都可以唯一确定，如果出现矛盾则无解。

如果排除掉无解的情况，每个连通块都可以有 2^k 种答案(除了`0`号点所在的连通块只有 `1` 种答案)。

设除 `0` 所在的连通块以外，连通块个数为 `t`，最终答案就是：

$$2^{kt} \mod 998244353$$

I题解题报告

Lecture by Hao Wu

2023年5月16日

题目大意

计算1/n小数点后面第m位的数字

输入

有多组输入数据.

每组测试数据占一行，两个整数: n，m.

((n≤ 10,000, m ≤ 1,000,000,000)).

输出

对于每组测试数据，输出1/n小数点后面第m位的数字.

输入样例

3 5

233 23

输出样例

3
5

解题思路

快速幂

1/n的第m位，其值为：

$$\left(10^{m-1}\%n\right)*10/n$$

标准答案

```
#include<stdio.h>
int FFP(int n,int m){//快速幂
        int r=1,a=10;
        while(m){
                if(m&1)r=r*a%n;
                a=a*a%n;
                m>>=1;
        }
        return r;
}

int solve(int n,int m){
        return 10*FFP(n,m-1)/n;
}

int main(){
        int n,m;
        while(scanf("%d%d",&n,&m)!=EOF)
          printf("%d\n",solve(n,m));
        return 0;
}
```

解题思路（另）

循环节

数组记录循环小数每一位，并找出循环节

标准答案（另）

```
#include<stdio.h>
#define N 10000
int a[N];
int Recurring_Period(int n){
                int i=0,r=1;
                do{
                   r*=10;
                   a[i++]=      r/n;
                   r%=n;
                }while (r!=1);
                return i;
}

int solve(int n,int m){
                int k=Recurring_Period(n);
                return a[(m-1)%k];
}

int main(){
                int n,m;
                while(scanf("%d%d",&n,&m)!=EOF)
                  printf("%d\n",solve(n,m));
                return 0;
}
```