1. 有趣的数

1.1. 题意回顾

统计有多少个不超过N的正整数n整除它们的数字之和$f(n)$。

对于 10%的数据，$N \le 10^6$；

对于30%的数据，$N \le 10^9$；

对于60%的数据，$N \le 10^{12}$；

对于100%的数据，$1 \le N \le 10^{18}$。

1.2. 算法一

当$N \le 10^6$时，可以直接枚举正整数n，算出数字之和$f(n)$，验证是否满足题意。用这个算法可以得到 10 分。

1.3. 算法二

当$N \le 10^9$时，枚举n显然会超时，但是别忘了比赛有 5 个小时，代码长度限制也是比较宽松的。所以可以用算法一写一个打表程序，每隔10^6个数就把中间结果输出出来，很快就能打出这个表。用这个算法可以得到 30 分。

1.4. 算法三

当$N \le 10^{12}$时，如果继续使用打表的方法，一方面程序运行时间很长，另一方面打出的表很大，可能会超出 cena 的代码长度限制。你可以尽量优化你的打表方法，也可以考虑其他的解法，都能够通过这一部分数据，总之，这一部分的数据就是用来过渡的。

1.5. 算法四

当$N \le 10^{18}$时，纵使你有天大的能耐，打表也不能通过了，我们需要找出更优秀的做法。上述的算法都基于枚举n，而n的范围很大不容易枚举。整个问题与$f(n)$紧密相关，显然$f(n)$并不会太大，至多只有$9 \times 18 = 162$。所以考虑枚举$f(n)$的值。

当$f(n)$确定时，显然应使用数位 dp 来解决这个问题。枚举$f(n) = sum$，设计这样的状态进行 dp：

$f[i][s][m][t](0 \le i \le \lceil \log_{10} N \rceil, 0 \le s \le sum, 0 \le m \le sum - 1, t \in \{0,1\})$表示由高到低已经 dp 了$i$位，前$i$位的和为$s$，前$i$位组成的数$mod\ sum$的值为$m$，$t$表示这个数已经一定严格小于$N$。设定好了状态，转移时枚举最低的一位$x$，我们很容易写出转移方程：

$$f[i+1][s+x][(m \times 10 + x)\%sum]\big[t||x < N[i+1]\big] += f[i][s][m][t]$$

边界不详述。

显然，一次 dp 的时间复杂度是$O(sum^2 \times \log_{10} N \times 10)$，枚举$sum$之后时间复杂度为$O(\frac{1}{6} \times sum^3 \times \log_{10} N \times 10) \approx O(sum^4)$，你要说它不靠谱也罢，反正当$N \le 10^{18}$时还是可以在规定时间内算出答案的。

2. 可见的点

2.1. 题意回顾

在一个N维空间中，所有整点都是不透明的。原点处有一个摄像头，再第j维上只能看到坐标不超过$Fib[m_j]$的点。求一共能够看到多少个坐标是Fib数列中的项的点。如果一个坐标在Fib数列中多次出现，那要按多个点进行计算，例如坐标为$(1,1,\dots,1)$的点要计算2^N次。

对于100%的数据，$1 \le T \le 50, 1 \le N \le 50, 1 \le m_j \le 10^6$。

<table>
<tr><th>测试点编号</th><th>$T \le$</th><th>$N \le$</th><th>$m_j \le$</th><th>其他约定</th></tr>
<tr><td>1</td><td>5</td><td rowspan="2">8</td><td>10</td><td rowspan="5">无</td></tr>
<tr><td>2</td><td>30</td><td>20</td></tr>
<tr><td>3</td><td>5</td><td rowspan="3">2</td><td rowspan="2">10^3</td></tr>
<tr><td>4</td><td rowspan="2">50</td></tr>
<tr><td>5</td><td>10^6</td></tr>
<tr><td>6</td><td>10</td><td>10</td><td>10^5</td><td rowspan="2">所有的m_j都相等</td></tr>
<tr><td>7</td><td>50</td><td>50</td><td>10^6</td></tr>
<tr><td>8</td><td>10</td><td>10</td><td>10^5</td><td rowspan="3">无</td></tr>
<tr><td>9</td><td>30</td><td>30</td><td rowspan="2">10^6</td></tr>
<tr><td>10</td><td>50</td><td>50</td></tr>
</table>

2.2. 算法一

先预处理出Fib数列的前若干项，暴力枚举每一维的坐标$x_1, x_2, \dots, x_N$，判断它是否能够被看到。如果一个点被挡住，当且仅当$\gcd(x_1, x_2, \dots, x_N) \neq 1$，且可以在枚举的同时维护出最大公约数，所以复杂度是$O(T \times m^n)$。这个算法可以得到测试点 1 的那 10 分。

2.3. 算法二

在算法一的基础上容易发现，绝大多数的Fib点都是可以看到的。这提醒我们，如果枚举看不到的点的坐标，可能会在很大程度上获得优化。事实上确实如此。经过尝试可以发现，当$n = 8, m_1 = \cdots = m_n = 20$时也只有大约$2 \times 10^6$个看不到的$Fib$点。考虑到提问次数稍微偏多，可以先枚举的方式预处理出这些看不到的点的坐标，每次提问的时候只需扫一遍，以提高效率。这个算法可以得到测试点 1 和 2 的共 20 分。

2.4. 算法三

当m比较大的时候，Fib点的坐标也很大了，显然不能强行高精度。这提醒我们Fib数列一定有什么性质。于是打表发现，

$$\gcd(Fib[i], Fib[j]) = Fib[\gcd(i, j)]$$

显然是可以推广到N维的，也就是

$$\gcd(Fib[i_1], Fib[i_2], \dots, Fib[i_N]) = Fib[\gcd(i_1, i_2, \dots, i_N)]$$

其实这个性质很早以前就被人挖出来出题了，只是最近见得少。由此，如果要满足$\gcd(Fib[i_1], Fib[i_2], \dots, Fib[i_N]) = 1$，就等价于$Fib[\gcd(i_1, i_2, \dots, i_N)] = 1$。因为只有$Fib[1]$和$Fib[2]$是等于 1 的，所以也等价于$\gcd(i_1, i_2, \dots, i_N) = 1$ 或 2。

有了这样的转化，整个问题变得与Fib数列无关了。于是，测试点 3 可以直接枚举i_1, i_2；测试点 4 可以仿照算法二枚举出看不到的点，然后每次提问时扫一遍。这个算法可以的到 30 或 40 分。

2.5. 算法四

后面的测试点的m的范围都相对较大，能不能再直接枚举坐标，我们需要更深入的推导。

$$Ans = \sum_{d=1}^{2} \sum_{i_1=1}^{m_1} \sum_{i_2=1}^{m_2} \dots \sum_{i_3=1}^{m_3} \langle \gcd(i_1, i_2, \dots, i_N) = d \rangle$$

$$= \sum_{d=1}^{2} \sum_{d|e}^{\min(m_j)} \mu\left(\frac{e}{d}\right) \sum_{i_1=1}^{m_1} \sum_{i_2=1}^{m_2} \dots \sum_{i_3=1}^{m_3} \langle e| \gcd(i_1, i_2, \dots, i_N) \rangle$$

$$= \sum_{d=1}^{2} \sum_{d|e}^{\min(m_j)} \mu\left(\frac{e}{d}\right) \left\lfloor \frac{m_1}{e} \right\rfloor \left\lfloor \frac{m_2}{e} \right\rfloor \dots \left\lfloor \frac{m_N}{e} \right\rfloor$$

$$= \sum_{d|e}^{\min(m_j)} \left\lfloor \frac{m_1}{e} \right\rfloor \left\lfloor \frac{m_2}{e} \right\rfloor \dots \left\lfloor \frac{m_N}{e} \right\rfloor \sum_{d|e}^{2} \mu\left(\frac{e}{d}\right)$$

预处理$a[e] = \sum_{d|e}^{2} \mu\left(\frac{e}{d}\right)$

$$= \sum_{d|e}^{\min(m_j)} \left\lfloor \frac{m_1}{e} \right\rfloor \left\lfloor \frac{m_2}{e} \right\rfloor \dots \left\lfloor \frac{m_N}{e} \right\rfloor a[e] \cdots\cdots\cdots (1)$$

用(1)式进行计算可以做到每次询问$O(nm)$的复杂度，总复杂度即为$O(Tnm)$，这个算法可以比算法三多得到测试点 6 和测试点 8 的 20 分，即可以得到 60 分。

2.6. 算法五

我们知道$\left\lfloor \frac{m}{e} \right\rfloor$的$e$从 1 枚举到$m$也只会有不超过$2 \times \sqrt{m}$个取值，大量连续的$e$取得的$\left\lfloor \frac{m}{e} \right\rfloor$都是相等的。所以，$\left\lfloor \frac{m_1}{e} \right\rfloor \left\lfloor \frac{m_2}{e} \right\rfloor \dots \left\lfloor \frac{m_N}{e} \right\rfloor$的值可以分成$O(n\sqrt{m})$段，每段内都是相等的，所以只需要预处理出$a[e]$的前缀和，就能够在$O(n\sqrt{m})$解决每次询问。预处理需要$O(m)$的时间，所以总的时间复杂度是$O(m + Tn\sqrt{m})$，这个算法可以通过所有测试点，得到 100 分。

3. 精明的壕

3.1. 题意回顾

给一个长度为n的序列$a_1, a_2, \ldots, a_n$，每次时询问给出L, R，找出一对l, r，满足$L \le l \le r \le R$, (l,r)的len值为$r-l+1$, (l,r)的mex值为$mex\{a_l, a_{l+1}, \ldots, a_r\}$。求$\frac{len+1-mex}{len+1+mex}$的最小值。

对于 10%的数据，$n, m \le 200$；

对于 40%的数据，$n, m \le 2000$；

对于 60%的数据，$n, m \le 20000$；

对于另外 20%的数据，$a_i \le 20$；

对于 100%的数据，$1 \le n, m \le 100000, 0 \le a_i \le n$。

为了方便描述，定义(l,r)的val值为$\frac{len+1-mex}{len+1+mex}$。

3.2. 算法一

每次提问都枚举l，在从左到右枚举r，开一个$bool$数组维护每个数是否出现过，每加一个新的数就调整mex，更新答案。每次询问的时间复杂度是$O(n^2)$，总的复杂度是$O(mn^2)$。这个算法可以得到 10 分，如果常数写得特别好，可以得到 40 分。

3.3. 算法二

先不考虑询问，枚举l，从左到右枚举r，同样是开一个$bool$数组来维护每个数是否出现过，就能够预处理得到任意的一组(l,r)的mex值，也就得到了(l,r)的val值。然后用下面这个简单的递推可以得到任意一个提问的答案

$$ans[L][R] = \min\{ans[L][R-1], ans[L+1][R], val(L,R)\}$$

这样预处理之后，每次查询就可以做到$O(1)$了。总的复杂度是$O(n^2+m)$，可以得到 40 分。

3.4. 算法三

首先，有一个很经典的题目：给一个序列，每次提问一个区间的mex值。今年的 Pkusc 上就有这道题的一个变形。这个问题的做法：离线所有询问，枚举左端点l，用数据结构维护一个关于右端点r的序列$mex(l,r)$，显然这个序列是单调不降的；当删掉a_l时，右边第一个与a_l相等的是a_k，则只需要将$l<r<k$的所有大于a_l的mex值改成a_l即可，也就是区间赋值，用线段树来实现。

本题是要找出$val = \frac{len+1-mex}{len+1+mex}$的最小值。容易发现，如果$mex$相等，$len$越小则$val$越小；如果$len$相等，$mex$越大则$val$越小。很自然的想法就是找出$len$和$mex$有一个固定时，另一个的最值，然后再来找$val$的最值。经过比较可以发现，一种可以实现的方法是找出所有的(l,r)，这些(l,r)如果再向中间缩短就会使mex减小，即$mex(l,r) > mex(l+1,r)$且$mex(l,r) > mex(l,r-1)$。

由经典问题的做法我们知道，当l确定时，关于r的序列$mex(l,r)$是单调不降的。所以我们可以把它分成很多小段，每一小段内的$mex(l,r)$都是相等的。那么只有每一小段的最左边一个位置做r才能满足$mex(l,r) > mex(l,r-1)$。另外，

在经典问题的做法中我们发现，如果左端点由l变成$l+1$时，$mex(l,r)$发生了变化，即被重新赋值了，那么这些r才能满足$mex(l,r) > mex(l+1,r)$。综上，要想提取出可能对答案做贡献的(l,r)，只需在经典问题的做法中的赋值操作的同时用赋值前的mex值的每一个相等的小段的最左边一个位置做r。显然，这样找出的(l,r)数量是$O(n)$级别的。

所以问题转化为，有了$O(n)$个三元组(l,r,val)，每次查询$L \le l \le r \le R$的三元组中val的最小值。如果离线，很容易用扫描线法在$O((n+m)\log n)$的时间里解决；如果硬要在线，也可以用其他的方式轻松解决。用这个算法，本题可以轻松地得到 100 分。