Multiply 2 Divide 2

令$m=100000$为值域上界。

首先要注意到可能对同一个数进行乘或除操作，并且对同一个数来说一定是先除后乘。

很多选手可能猜测操作完的数不会很大，实际上在题目数据范围内我们可以造出达到2^{183}的数据，由此我们出了另一道题。

我们可以先有一个简单的dp，设$f_{i,j,k}$表示前i个数不降，第i个数除了j次，再乘了k次的最小操作次数。第二维是$\log m$级别，第三维要开到200左右。即使转移$O(1)$也不能通过本题。

再注意到一个性质，如果一个数被操作后超过了m，那么后面的数也都会超过m。一个数大于m之后(设为$a_i=x$)，从它开始的后缀都至少乘了一次。那么有一个引理，如果这个数多乘一次($a_i=2\cdot x$)，后面的最优决策一定是在原最优解基础上都多乘一次。因为假如$a_i=2\cdot x$有另一个更优解，那么这个在这个更优解基础上后面的数都少乘一次（一定可以做到，因为大家都大于m，最后一步一定是乘法），就得到了一个比$a_i=x$原最优解更优的解。矛盾。

由此我们可以把f的第三维缩减到$\log m$级别，只计算到刚好超过m的时候。后面的部分我们可以再进行dp，设$g_{i,j}$表示第i个数除了j次并且恰好超过m时，后面最少需要操作多少次。g可以倒着dp转移，转移的时候要么后面整体乘2，要么不动，转移一个状态是$O(\log m)$，计算g的复杂度为$O(n\log^2 m)$。转移f的时候可以把所有状态记录下来，双指针$O(1)$转移，计算f的总复杂度是$O(n\log^2 m)$。总复杂度是$O(n\log^2 m)$。

Hack of Multiply 2 Divide 2

首先，我们分析一下 a_n 最大为什么级别。

规定一些记号，设 m 为值域。

考虑最终的结果序列（操作次数最少前提下 a_n 最小）。设最高位为 2^i 的数的个数为 c_i，有如下引理：

对满足的 $2^p>m$ 的 p，有 $\sum\limits_{i=p+1}^{\infty} c_i < 2\log_2 m\cdot c_p$。

这是因为，若上式不成立，则可以多花费最多 $2\log_2 m\cdot c_p$ 的代价把这 c_p 个数先全部变成2的幂，再把它们变成 2^k。这样，后面最高位为 2^{k+1} 的数都可以少乘一次，也不会比前一位少。后面的数同理。于是，这种方案可以比原来少 $\sum\limits_{i=p+1}^{\infty} c_i$ 的代价。这样即可得到更优解（要么操作次数少了，要么次数不变 a_n 小了），矛盾。

通过对上述引理的进一步分析，可以得到 a_n 最多会乘 $O(\log n\log m)$ 次。于是可以发现题目Multiply 2 Divide 2的暴力算法其实只比标算多一个 $\log$。这里不再赘述。

上述引理的证明，其实已经提供了构造的思路。接下来介绍标答的构造思路。

我们希望构造 $k,k-1..2,1$ 这样的序列，让每个数都比前一个数多乘一次。但是由上述引理这是不可能的。所以要根据需要，来适当增加某些数的个数，即增大 c_p 使 c_p 满足引理的式子。同时，为了让除法的代价较高，要提高所有数字的 $lowbit$。

最后的算法为，倒序枚举 i ，输出 d_i 个 $i\cdot 512$。dp 求 d_i 的值，d_i 为满足 $d_i\cdot(2\log_2 \mathrm{lowbit}(\mathrm{i}\cdot 512)+2) > \sum\limits_{\mathrm{j}=1}^{\mathrm{i}-1}\mathrm{d_j}$ 的最小整数。同时要注意，设 p 为令 $d_p\neq 0$ 的最大整数，则要满足 $d_p>\frac{n}{2}$。否则，可以令这 d_p 个数做一次除法，后面的数都少乘一次。递推到不满足这个条件就结束，之后输出即可。标答做到了 $a_n=2^{183}$。

Find the Number of Paths

将$n+k$个点重编号，原来编号为i的点改为编号为$n+k-i$的点。

问题重述为：

有$n+k$个城市，第$i(0<i\le n+k-1)$个城市到第$i-1$个城市有i条可区分的单向道路。

第$i(0\le i\le n+k-1-x)$个城市到第$i+x$个城市有a_x条可区分的单向道路。

对于$m=0,1,\dots,n-1$，求从城市$n-1$到城市m经过道路条数恰好为i的路径数量。

用$dp_{i,j}$表示从城市$n-1$到城市j经过道路条数恰好为i的路径数量。

有初值$dp_{0,n-1}=1,dp_{0,j}=0(j\neq n-1)$。

转移为

$$dp_{i+1,j}=dp_{i,j+1}\times(j+1)+\sum_{k=0}^{j-1}dp_{i,k}a_{j-k}$$

注意：当$x\ge n$，$a_x=0$。

令多项式

$$f(i,x)=\sum_{j=0}^{n+k-1}dp_{i,j}x^j$$
$$A(x)=\sum_{i=1}^{n-1}a_ix^i$$

根据转移可以得到

$$f(i+1,x)=f'(i,x)+f(i,x)A(x)$$

初值为

$$f(0,x)=x^{n-1}$$

考虑构造

$$g(x)=e^{\int A(x)dx}=e^{\sum_{i=2}^{n}\frac{a_{i-1}}{i}x^i}$$

令

$$p(i,x)=f(i,x)g(x)$$

则有

$$p(i+1,x)=(f'(i,x)+f(i,x)A(x))g(x)=p'(i,x)$$
$$p(k,x)=p^{(k)}(0,x)$$

初值为

$$p(0,x)=x^{n-1}g(x)$$

可以用多项式exp把$p(0,x)$转成不带exp的形式，随后只需对$p(0,x)$求k阶导数即可得到$p(k,x)$。

随后由$p(k,x)=f(k,x)g(x)$得到$f(k,x)=p(k,x)g^{-1}(x)$。

最后答案即为$f(k,x)$的系数。

多项式exp的时间复杂度为$O(nlogn)$，求导的时间复杂度为$O(n)$，多项式求逆的时间复杂度为$O(nlogn)$。

总的时间复杂度为$O(nlogn)$。

Yet Another Easy Permutation Count Problem

首先令 $x_i=\sum_{j=1}^{i-1}[P_i<P_j]$。

显然 x 与 P 双射，于是就可以通过 x 刻画好位置了。

那么就是每次冒泡排序后统计 x 中 >0 的数的连续段个数。

每次冒泡排序后 x 中大于 0 的元素会减一然后整体前移。

可以发现序列 x 的贡献为 $\sum\max(x_i-x_{i-1},0)$。

这个东西可以拆开然后算贡献。

当然，$\sum\max(x_i-x_{i-1},0)$等价与 $\sum_{i=1}^{n}v_i$，v_i 表示第 i 个位置后面比 P_i 小的那些数对应的位置的连续段个数。

这个就比较好算，枚举一个位置 i，一个位置 $j(i<j)$ 当且仅当 $P_i>P_j$ 且 $P_{j+1}\ge P_i$。

算的时候枚举填啥，然后其他数随便填，得到一个 $\mathcal{O}(n^3)$ 的算法。

拆柿子后可以用树状数组/线段树优化，可以做到 $\mathcal{O}(n+m\log n)$。

Yet Another Easy Function Sum Problem

出题人已经麻了！

~~题解 1~~

利用杭电特性，打表套出 5 个数据然后跑暴力。

出题人在想：雅礼的两个队是不是分开各跑一组极限数据。

还有那个套数据 `puts("HAHAHA")` 的/fn/fn，过分了。

正解

这里先说一下 $p(i)=H(i)$。

显然可以得出 $p(ij)=p(i)p(j)\frac{1}{p(\gcd(i,j))}$。

这里先定义几个东西。

$$G(N)=\sum_{d|N}\mu(d)f(N/d)$$

$$f(N)=\frac{1}{p(N)}$$

显然要求这么个东西

$$\sum_{T=1}^{n}\mu(T)\sum_{i=1}^{n/T}p(iT)\sum_{j=1}^{n/T}p(jT)$$

这个怎么做呢?

考虑对 T 进行阈值分治，设一个阈值 $B=n^{\frac{2}{3}}$。

$$pA=\sum_{T=1}^{B}\mu(T)\sum_{\lfloor\frac{n}{B}\rfloor<\max(i,j)\le\lfloor\frac{n}{T}\rfloor}p(iT)p(jT)$$

$$pB=\sum_{T=1}^{n}\mu(T)\sum_{\max(i,j)\le\min(\lfloor\frac{n}{B}\rfloor,\lfloor\frac{n}{T}\rfloor)}p(iT)p(jT)$$

然后对 pB 变形。

$$pB^{'}=\sum_{i=1}^{n/B}\sum_{j=1}^{n/B}\sum_{T=1}^{n/\max(i,j)}\mu(T)p(iT)p(jT)$$

如果我们能快速求出 $pA+pB^{'}$ 就解决问题了。

先定义一些东西。

$$R(N,T)=\sum_{i=1}^{N}p(iT)$$

$$F(N,i,j)=\sum_{T=1}^{N}\mu(T)p(iT)p(jT)$$

$$W(N,T)=\sum_{i=1}^{N}\mu(iT)p(iT)^2$$

显然可以推出

$$pA=\sum_{T=1}^{B}\mu(T)(R(\lfloor\frac{n}{T}\rfloor,T)^2-R(\lfloor\frac{n}{B}\rfloor,T)^2)$$

于是来快速解决 $R(N,T)$。

$$R(N,T)=\sum_{i=1}^{N}p(i)p(T)f(\gcd(i,T))$$

$$R(N,T)=p(T)\sum_{d|T}f(d)\sum_{i=1}^{N/d}p(id)[\gcd(i,T/d)=1]$$

$$R(N,T)=p(T)\sum_{V|T}G(V)\sum_{i=1}^{N/V}p(iV)=p(T)\sum_{V|T}G(V)R(N/V,V)$$

于是可以递归计算，当 $T=1$ 时即为 $p(i)$ 的前缀和。

恶心的地方在于 $pB^{'}$。

$$pB^{'}=\sum_{i=1}^{n/B}\sum_{j=1}^{n/B}F(n/\max(i,j),i,j)$$

来推一推 F。

$$F(N,i,j)=p(i)p(j)\sum_{T=1}^{N}\mu(T)p(T)^2f(\gcd(i,T))f(\gcd(j,T))$$

$$p(i)p(j)\sum_{d_1|i}\sum_{d_2|j}f(d_1)f(d_2)\sum_{T=1}^{N/lcm(d_1,d_2)}\mu(Tlcm(d_1,d_2))p(Tlcm(d_1,d_2))^2[\gcd(i/d_1,Tlcm(d_1,d_2)/d_1)=1][\gcd(j/d_2,Tlcm(d_1,d_2)/d_2)=$$

$$p(i)p(j)\sum_{d_1D_1|i}f(d_1)\mu(D_1)\sum_{d_2D_2|j}f(d_2)\mu(D_2)\sum_{T=1}^{N/lcm(d_1D_1,d_2D_2)}\mu(Tlcm(d_1D_1,d_2D_2))p(Tlcm(d_1D_1,d_2D_2))^2$$

你会发现一个极好的性质，那就是后面的求和函数都是 $?(Tlcm)$ 的形式。

于是就会出现前文 $W(N,T)=\sum_{i=1}^{N}\mu(iT)p(iT)^2$ 的定义。

$$F(N,i,j)=p(i)p(j)\sum_{T_1|i}G(T_1)\sum_{T_2|j}G(T_2)W(N/lcm(T_1,T_2),lcm(T_1,T_2))$$

来推 W。

$$W(N,T)=\sum_{i=1}^{N}\mu(i)\mu(T)[\gcd(i,T)=1]p(i)^2p(T)^2$$

$$W(N,T)=\mu(T)p(T)^2\sum_{i=1}^{N}\mu(i)p(i)^2[\gcd(i,T)=1]$$

$$W(N,T)=\mu(T)p(T)^2\sum_{d|T}\mu(d)\sum_{i=1}^{N/d}\mu(id)p(id)^2$$

$$W(N,T)=\mu(T)p(T)^2\sum_{d|T}\mu(d)W(N/d,d)$$

递归计算即可，当 $T=1$ 时，即为 $\mu(i)p(i)^2$ 的前缀和。

但是，这真的能过吗？看上去就很垃圾。

注意到只需要 $p(i)$ 的前缀和与 $\mu(i)p(i)^2$ 的前缀和的 $\sqrt{n}$ 个点值，进行 min25 块筛即可。

但是写出来发现还是跑的很慢，有下面几个剪枝。

1.枚举 T 的因数时只需要最多枚举到 $\sqrt{n}$，因为无论时 $W(N,T)$ 还是 $R(N,T)$ 都必须满足枚举的因数得小于等于 N。

2.G 函数与 μ 函数有一些为 0 的取值，直接跳过不要算。

3.预处理 $N\times T\le P$ 的时候的 W,R 函数的值，std 取的是 1.4×10^6。

最后差不多开上 O2，在 luogu 极限数据 2.7s（实际上有一半时间都在预处理）。

Maex

用$sz[u]$表示u子树节点个数，$dp[u]$表示u 子树内 $sum(b_i)$ 可以达到的最大值。

那么考虑转移，u本身的$b[u]$一定可以是$sz[u]$ ，而因为a必须两两不同，0只能在u的某个子树内，所以除了某个子树，其他子树节点的b_i一定都为0.

那么转移就是:

$dp[u]=sz[u]+max(dp[v])(v\in substree(u))$

复杂度$O(n)$

Shinobu loves trip

从s走d天之后会来到$(s*a^d)\%P$ ，因此，判断一个点x是否会被一个计划(s,d)经过，只需要判断是否存在一个$0\le k\le d$使得$(s*a^k)\%P=x$ ，由于P是质数，所以已知x和s的值，可以得到a^k的值。

预处理所有$a^0,a^1....a^{200000}$，然后直接查询算出来的a^k是否在预处理出的数里面，如果不在，这个计划一定不经过x，否则可以得到k 的具体值,判断$k\le d$即可。

要注意特判$s=0$的情况。

哈希预处理$a^0,a^1...a^{max(d_i)}$，每个计划需要求s的逆元，$nlog(P)$ 预处理。对于每个查询，遍历所有计划得到答案。

复杂度$O(T*(max(d_i)+nlog(P)+nq))$

题外话：

> 本来还有$n=2e5, max(d_i) = 1000$ 这种数据范围，对应的做法复杂度是$n*max(d_i)log(n)$，但是被批评两种没什么相干的做法揉在一起不三不四就削掉了。
>
> Heltion老师在验题时指出可以直接把环抠出来做差分，这样可以做到$q=n=10^5,P\le10^9, max(d_i) \le 10^9$，非常的有意义，但是因为hdu容易卡常，不太敢冒险加强，于是还是保留了现在这个削弱版本的作为萌萌签到题。

Shinobu Loves Segment Tree

考虑第a天，会对$value[x]$ 产生的影响：

依次看x次高位到最低位:

如果这一位是1，则说明x在右子树，当前区间长度$a = \lfloor \frac{a}{2} \rfloor$

如果这一位是0，则说明x在左子树，当前区间长度$a = \lceil \frac{a}{2} \rceil$

由此可以在$O(log)$时间快速模拟得到某一天$value[x]$的变化。

现在考虑$calc(l,r,p)$表示，从第l天到第r天，上述模拟过程从x的第p位进行到第0位，对$value[x]$产生的贡献。

那么可以发现，$calc(l,r,p)$可以由$calc(l/2,r/2,p-1)(\text{第}p\text{位为}1\text{，下取整的情况})$或者$calc((l+1)/2,(r+1)/2,p-1)(\text{第}p\text{位为}0\text{，上取整的情况})$ 推得 。

具体的，对任意$k>0$, 第$2k$天和第$2k+1$天，下取整之后，他们对应的区间长度相同，后续的操作也相同，因此对答案产生的贡献也相同。上取整同理可得类似关系。

在计算的时候注意考虑边界情况，如l和$l+1$操作后是否相同，$r-1$和r操作后是否相同，$l=1$情况的特判等。

$calc$一共会递归log层，每次的边界情况贡献$O(log)$模拟得到，总复杂度$Tlog^2(n)$。

(看了大家的比赛代码，感觉这题有一万种过法)

Map

本题的题目背景是著名的巴拿赫不动点定理. 由于大地图到小地图的映射关系是一个压缩映射, 因此存在唯一不动点. 求这个不动点的方式有很多, 这里从代数和几何的两个角度给出两种比较简单的实现方法.

方法1(代数方法)

设所求的不动点为点P. 由于$AB \perp AD$, 因此可设:

$$\vec{OP} = \vec{OA} + \lambda \vec{AB} + \mu \vec{AD}$$

其中λ, μ为待定参数. 由于P在两个地图上的对应关系相同, 因此有:

$$\vec{OP} = \vec{Oa} + \lambda \vec{ab} + \mu \vec{ad}$$

联立消去$\vec{OP}$得:

$$\lambda(\vec{AB} - \vec{ab}) + \mu(\vec{AD} - \vec{ad}) = \vec{Oa} - \vec{OA}$$

分别考虑x分量和y分量可以得到一个关于λ, μ的二元一次方程组. 根据巴拿赫不动点定理, 方程有唯一解. 求出λ, μ后即可得到P的坐标.

方法2(几何方法)

设所求的不动点为点P, 小地图m的边长是大地图的k倍. 于是对于大地图上任意一点Q, 设它在小地图上对应点为q, 那么就有$PQ:Pq=1:k$. 于是P在以Q,q为对应点, k为定比的阿波罗尼斯圆上. 在地图上取一些对应点, 先特判这些点是否是不动点, 然后做出这些点对应的阿波罗尼斯圆, 这些圆的交点就是不动点. 由于不动点唯一, 因此当取的圆较多时, 一定只有一个交点. 对于题目的数据, 对应点只需要取$A-a, B-b, C-c, D-d$即可.

以上两种做法复杂度都为$O(1)$. 此外本题还有很多需要三分或者大量浮点开根号的做法. 为了使这些做法能够通过, 本题开了非常充足的时限($10s$).

Planar Graph

本题的题目背景是平面图欧拉定理的一种证明方式. 根据平面图欧拉定理$V-E+F=k+1$(V,E,F,k分别为点, 边, 面, 连通分量的个数), 每去掉一条边(加一个隧道可以理解为删除一条边), 平面图G的面的个数就会增加1. 因此最终边的个数为$V+1-k-1=V-k$, 因此删边个数为$E-V+k$. 于是对于每个连通分量, 保留了$V'-1$条边(V'为该连通分量的点数). 如果该连通分量有圈, 那么这个圈内的区域一定不与外部的无穷平面连通, 不符合题意. 因此每个连通分量都不含圈, 且有$V'-1$条边, 因此每个连通分量都变成树.

现在从另外一个角度考虑. 我们构造一个新图$\bar{G}$. 我们把平面图的每个面当作一个点, 如果两个面被平面图的某条边分隔, 那么这两个面在$\bar{G}$对应的点之间连一条边(于是G和$\bar{G}$的边是一一对应关系). $\bar{G}$的点个数为$F=k+1-V+E$, G的删边个数为$E-V+k=F-1$, 由于建隧道之后任意两个区域可达, 因此如果把G中删掉的边在$\bar{G}$中考虑, 删除$\bar{G}$中在G保留的边, 那么边个数为$F-1$, 且任意两点连通. 因此这些边构成$\bar{G}$的生成树.

于是, 问题转化为求$\bar{G}$的字典序最小的生成树. 由于每条边边权不同, 因此字典序最小的生成树等价于$\bar{G}$最小生成树, 等价于$\bar{G}$的生成树边权和最小(考虑$kruskal$算法的构造过程, 并注意所有边权不同时最小生成树唯一). 由于所有边边权和为定值, 因此$\bar{G}$生成树边权和最小等价于G的每个连通分量对应的树边权和最大. 于是只需要对于G的每个连通分量求出最大生成树, 然后这些最大生成树没有用到的边就是$\bar{G}$的最小生成树的边.

Find different

如果只考虑数组数量，答案应该是m^n。

如果不考虑操作2，只考虑操作1，操作1下的原数组的差分数组模m意义下是不变的，于是我们可以枚举差分数组的数量，保证差分数组元素和模m为0，可以任意枚举前$n-1$个差分数组元素的大小，用最后一个元素恰好使它们的和模m为0，答案应该是m^{n-1}。

同时考虑操作1、2，可以发现，对于原数组的差分数组(模m意义下)，如果它每个位置后d个元素都与前d个元素相同，这样最大的d称该数组循环节大小为 d，d一定是n的因子，该差分数组会被重复计算d次。

于是我们设$F(n,d)$为差分数组长度为n，循环节大小为d的数量。$G(n,d)$为差分数组长度为n，循环节大小小于等于d的数量。$ans(n)$为长度为n的答案。

差分数组的循环节要满足循环节中d个元素的和乘上 $\frac{n}{d}$后模m为0，即d个元素的和模m后为 $\frac{m}{gcd(\frac{n}{d},m)}\times k,k\in\{0,1,\ldots,gcd(\frac{n}{d},m)-1\}$,故$d$个元素的和模$m$后有$gcd(\frac{n}{d},m)$个不同的值。所以

$G(n,d)=m^{d-1}\times gcd(\frac{n}{d},m)$，（前$d-1$个元素任意，最后一个元素补上），

$F(n,d)=G(n,d)-\sum\limits_{t|d,t\neq d}F(n,t)$，（小于等于答案的减去小于的答案），

$ans(n)=\sum\limits_{d|n}F(n,d)\times\frac{1}{d}$(重复算了$d$次，所以除以$d$).

对每个$ans(i)$用$O(log^2 i)$的时间单独计算即可

总复杂度为$O(nlog^2 n)$，该复杂度可过

注意到以上式子容斥部分可以用莫比乌斯反演做，

$F(n,d)=\sum\limits_{t|d}G(n,t)\times\mu(\frac{d}{t})$，

$ans(n)=\sum\limits_{d|n}F(n,d)\times\frac{1}{d}=\sum\limits_{d|n}\sum\limits_{t|d}G(n,t)\times\mu(\frac{d}{t})\times\frac{1}{d}=\sum\limits_{d|n}\sum\limits_{t|d}m^{t-1}\times gcd(\frac{n}{t},m)\times\mu(\frac{d}{t})\times\frac{1}{d}$

$ans(n)=\sum\limits_{t|n}m^{t-1}\times gcd(\frac{n}{t},m)\times\sum\limits_{t|d}\mu(\frac{d}{t})\times\frac{1}{d}$

$ans(n)=\sum\limits_{t|n}m^{t-1}\times gcd(\frac{n}{t},m)\times\sum\limits_{j=1}^{\frac{n}{t}}\frac{\mu(j)}{j}\times\frac{1}{t}$，(三个部分都可以预处理)

总复杂度为$O(nlogn)$.

实际$\sum\limits_{j=1}^{n}\frac{\mu(j)}{j}=\frac{\varphi(n)}{n}$，故最终答案也可为$ans(n)=\sum\limits_{t|n}m^{t-1}\times gcd(\frac{n}{t},m)\times\frac{\varphi(\frac{n}{t})}{n}$

Loop

考虑一次操作，我们会贪心的从头往后找到第一个$a_i>a_{i-1}$ (此时前面$i-2$个元素一定都大于等于a_{i-1}),然后将a_{i-1}弹出，放到第一个小于a_{i-1}的数前面

注意到每次操作本质是将一个元素拿出，然后把后面所有元素往前移，最后再把拿出的元素插入到拿出位置的后面的任意位置

所以k次操作可以先弹出k个元素，最后再插入回去。从头往后做如果当前$a_i>a_{i-1}$就把a_{i-1}弹出放入堆中(这里可以发现a_{i-1}弹出是一个类似维护单调栈的操作)，最后将剩余数组与堆中k个元素做一次归并操作即可。