**Mathematics**

**欧拉函数**

论如何利用欧拉函数来加快幂的取模运算。a^b mod c。

欧拉函数有个性质，就是对于两个互质的数a和p，a^phi(p) ≡ 1 (mod p)。

如果有a，c互质的话，答案显然就是a^b mod c = a^(b mod phi(c)) % c

但是如果没有a，c互质的话，就需要用这个公式

a^b mod c = a^(b mod phi(c) + phi(c)) % c ，仅在b>=phi(c)时才成立。

**带模除法**

如果a/b是整数然后要求(a/b)%c，可以直接算(a%(bc))/b。证明很简单。

这种用法是用来处理取余的那个数不是一个质数的时候。而且只进行一次除法。

注意如果(bc)即新的取模的数达到了long long的范围，那之前的乘法就可能是long long和long long相乘，最后就直接爆出去了。所以这个时候在进行乘法的时候应该考虑用快速加法，就是logn的模拟加法。

当然还有一种处理方式。就是如果取模的数并不是质数，或者除数与取模的数并不互质（当然也就没有逆元直说），此时可以将取模的数进行分解质因数处理，对每个质因数分别算出答案，然后再用中国剩余定理把它们合并回去。

快速 O(n) 处理 1~p-1 的逆元。。

inv[P mod i]\*(P mod i) mod P = 1

inv[P mod i]\*(P-P/i\*i) mod P = 1

inv[P mod i]\*i\*(-P/i) mod P = 1

i\*(P-P/i)\*inv[P mod i] modP = 1

一个 for 循环就搞定了。

**斐波那契**

1.gcd(fib(n),fib(m))=fib(gcd(n,m))

Fibonacci 素数：和 Fibonacci 序列中比它小的数都互质的数称作 Fibonacci 素数。。注意 Fibonacci 素数并不一定都是素数。。但是除了前几项以外斐波那契的素数项的都是斐波那契素数。。

2.如果fib(k)能被x整除，则fib(k\*i)都可以被x整除。

3.f(0)+f(1)+f(2)+…+f(n)=f(n+2)-1

4.f(1)+f(3)+f(5)+…+f(2n-1)=f(2n)

5.f(2)+f(4)+f(6)+…+f(2n) =f(2n+1)-1

6.[f(0)]^2+[f(1)]^2+…+[f(n)]^2=f(n)·f(n+1)

7.f(0)-f(1)+f(2)-…+(-1)^n·f(n)=(-1)^n·[f(n+1)-f(n)]+1

8.f(m+n)=f(m-1)·f(n-1)+f(m)·f(n)

9.[f(n)]^2=(-1)^(n-1)+f(n-1)·f(n+1)

10.f(2n-1)=[f(n)]^2-[f(n-2)]^2

11.3f(n)=f(n+2)+f(n-2)

12.f(2n-2m-2)[f(2n)+f(2n+2)]=f(2m+2)+f(4n-2m) [ n>m≥-1,且n≥1]

**Lucas定理**

C(n,m)%p ≡ C(n%p, m%p) \* C(n/p,m/p) % p

其中定义C(n,0) = 1, C(i,j) = 0 (i<j)。

注意在另求 C(x,y) 的时候要一个一个去乘，因为预处理阶乘在处理组合数上是有问题的。

Lucas定理的另外一种形式，更直观来说，就是把n和m写成p进制，然后取各自对应的位来得到组合数进行乘法。

**数值分析**

论如何处理精度。。

首先分母必须不能取 0 。。这是要时刻回避的。。

其次要尽可能地避免特别大的数和特别小的数相乘。。虽然乘出来的可能是一个比较中和的值但是这之中就很容易产生误差。。也要避免。。

在数据输入的部分就把一些极限情况给特判掉，这是非常必要的。。

还有时候取对数也可以把 “大数与小数相乘” 给化简掉。。当然前提是并不要求一个完全精确的值。。

**拉格朗日插值法数**

对某个多项式函数，已知有给定的k + 1个取值点。

假设任意两个不同的自变量对应的函数值都互不相同，那么应用拉格朗日插值公式所得到的拉格朗日插值多项式为

L(x) = Σ yi \* li(x)

其中每个为**拉格朗日基本多项式**（或称**插值基函数**），其表达式为：

li(x) = Π (x-xi) / (xj-xi) (i≠j)

拉格朗日基本多项式 L(i) 的特点是在 Xi 上取值为1，在其它的点 Xj ( j≠i ) 上取值为0。

**Burnside定理**

不同等价类的个数 = 每个置换对应的（在当前置换下保持不变的方案数，即不动点）的个数之和 / 总的置换个数

**小技巧**

遇到 Lcm 。。 99% 要转化成 Gcd 来处理。。

组合数取模。。

如果底数特别大，且 mod 是可因式分解成多个质数的一次方相乘的（这些质数们不能特别大。。），可以用 Lucas 定理 + CRT 合并。。复杂度大概是 O(lgN\*P)。。

如果 mod 是任意给定的，则底数必定不会太大。由于组合数一定是正整数，记录每个质数的幂即可，最后快速幂。。复杂度是 O(NlgN) 。。

底数特别大 + mod 任意给定貌似不可做。。

。。下面的题是利用矩阵的列向量来处理问题。。与极大线性无关向量组有关。。和高斯消元关系不是很密切。。

Zoj 3636 - Decode

首先要 C(n,3) 枚举变动的位，然后就变成了判定给定 N 个数，能否通过异或得出 X。。当然一个直观的想法就是每次都建立矩阵（。。大小为N\*K。。），然后 O(N^2\*K) 的高斯消元。。不过明显不行。。这里利用异或的特殊性。。可以把系数矩阵中的 column 相互异或。。之后也是消成一个 反-行阶梯矩阵，这一步预处理是 O(NK) 。。然后就可以每次 O(N) 地判断是否能够异或出 X 了。。（。。其实就和高斯消元关系不是很明显了。。）

扩展。。如果想要判断 异或得X 的方案数的话就不能够这么做了。。只能用原始的高斯消元 + 求自由变元数。。

Hdu 3949 - XOR

同上面那题。。列向量从大到小消成的线性无关的后，线性无关的向量个数（其实就是秩。。）设为 Size 则最多有 2^Size-1 种不同值（不考虑 0 。。0 需要之前特判掉。。）。。对 K 进行二进制拆解。。对应位为 1 的异或进答案即可。。不能直接加= =

**Data structure**

**树状数组**

lowbit(x) = x & (-x)

树状数组只有两种操作：1、单点修改；2、求1~n的区间和

由这两个基本操作衍生出了一些别的操作。

（以下的修改都是加上一个数）

1、单点修改+任意区间查询。[l,r] = [1,r] - [1,l-1]。这个技巧下面每一个都会用到。

2、区间修改+单点查询。做法是把原序列弄成一次差分数列，然后单点查询就是求前缀和了。

3、区间修改+区间查询。首先把区间用1、的技巧拆开（这一步是必要的）做法还是弄成一次查分序列，按照单点查询的方法来查询一个区间（就是多个点的和）。。第 i 个数被加了(n-i+1)次，然后把 i\*d[i] 做为一个新的量来维护。两个树状数组就可以完成操作。

现在来考虑二维的

4、单点修改+区间查询。[x0,y0,x1,y1] = [1,1,x1,y1] - [1,1,x0-1,y1] - [1,1,x1,y0-1] + [1,1,x0-1,y0-1]。

5、区间修改+单点查询。把修改区间 [x0,y0,x1,y1] 变成修改点 + [x0,y0] - [x0,y1+1] - [x1+1,y0] + [x1+1,y1+1]。

6、区间修改+区间查询。把上面两个技巧一起用，把查询区间用4、的技巧拆开，归为四个左上角为[1,1]的区间。查一个区间就相当于查这个区间的所有单点，对每个这样的区间分别考虑区间内每个点被计算了多少次，展开式子用三个树状数组来维护就行。

**分块**

题目大意：给出一堆离散的集合，总长度 <=N。。修改 + 求和。。

以 sqrtN 为分界。。

1、小集合的查询，直接求和原序列的数，以及和每个大集合的相交个数\*对应的 delta 值。。O(N^0.5)

2、大集合的查询，直接调去 ans 值。。O(1)

修改操作：每次修改操作都要修改每个（N^0.5个）大集合的 ans 值。。O(N^0.5)

3、小集合的修改，直接在原序列上修改、O(N^0.5)

4、大集合的修改，直接在 delta 值上改O(1)

题目大意：给一个序列，每个元素<=5\*10^4，Q个询问，问每个区间 [L,R] 上有多少对相同的数。

经典的莫队算法。。考虑离线操作。。按左端点分块，相同块内部按右端点排序。然后维护两个指针暴力扫过每个区间。。总时间复杂度O(N^1.5)

**CDQ分治**

前面的对后面的有影响。。则可以用分治。。考虑对于每个区间。。前半部分对后半部分的影响。。分治的复杂度是nlgn。。排序或者再套用数据结构维护也就是nlgnlgn的级别。

分治是为了维护时间戳的顺序。。因为如果对其中一维排序的话，顺序就打乱了，需要分成两块考虑前半部分对后半部分的影响。。显然在取中点的时候理论复杂度是最低的。。

模板题。。

1、二维LIS

2、Change 改变一个二维矩阵中一个格子的number。。Query 查询任意矩阵的和。。Add 次数10^5，Query 次数10^4。。

树状数组怎么做。。？离散化也开不起空间。。但是分治可以秒杀。。先把 Change 变成 Add，再把查询任意矩阵拆成四个矩阵的 运算和 。。然后这就是考虑一个区间前半部分的 Add 对后半部分 Query 的影响。。

3、二维的动态查询最近点对。。每次只考虑一个点左下角的点，做四次。

**Dynamic Programming**

**概率DP**

概率/期望DP的话。。如何表示状态很关键。。转移比较好想。。然后就是怎么去处理这个转移。。因为一般情况下暴力消元是不可行的。。

题目的一般解法一、列出方程直接高斯消元

题目的一般解法二、因为答案可能会很大高斯消元会精度溢出。。要手动处理方程得到递推式以避免消元。。这之中又分为两种一般的方法。。

①、设根据方程组的特点（如都含有某一项等特性）设计新的能够直接递推出的函数。

②、手动模拟方程的迭代

**斜率优化**

斜率优化的使用效果：可以O(1)找到决策点。

斜率优化的本质是维护一个队列，队列中的任意两项的“斜率”（其实就是手算的一个值）都是单增，而且队列中的决策点也是单增的。这样每次只要取队列中的队首作为决策点就好。

使用斜率优化之前要证明几点：

1、较优的决策点对后续状态影响具有持续性，简单来说就是如果在状态I处，选择了J作为决策点，那在I+1处的决策点一定不可能在J之前。

。。但是光有这点是不能把O(1)找到决策点的。。

2、把DP方程写出来，左边化为J，K的形式，右边化为I的形式。（这里左边的式子一定要是斜率的形式？而且分母部分得是单调的？？）

3、进行队首与队尾的维护。

**积性函数**

n = ∏ pi^ai -> f(n) = ∏ f(pi^ai) 。。为积性函数。。

n = ∏ pi^ai -> f(n) = ∏ f^ai(pi) 。。为完全积性函数。。

一般来说非完全积性函数比较常见。。而且在计算的时候其实复杂度基本是一样的，因为积性函数都有一些其他的性质。。

常见的积性函数。。欧拉函数 φ 。。莫比乌斯函数 μ 。。正约数数目 d。。正约数之和 σ。。因子函数 σk，即所有约数的 k 次幂之和。。这些函数都是可以通过 线性筛法 求出来的。。比如在求 d 和 σ 的时候要记录 最小质因子 的数目（。。求 σ 的时候线性筛目测效率不高。。）。。不过为了求稳不如不在迫不得已时先写 枚举约数 的筛吧。。

常见的完全积性函数。。单位函数 id 。。对于狄利克雷卷积的乘法单位 e (ε) 。。在数论变换有很重要的作用。。

其中非常重要的变换 N = Σ φ(d) d|n ; e(N) = Σ μ(d) d|n 。。前者考虑每个数与 n 的 gcd。。后者二项式定理。。

对于两个数论函数 f、g。。

(f\*g)(N) = f(N) \* g(N)

(f×g)(N) = ∑ f(d) \* g(N/d) (d|n) 。。 （。。显然满足交换律。。）

如果 f、g 均为积性函数。。f\*g、f×g 也均为积性函数。。后者成为狄利克雷卷积。。在数论变换中很常见。。

之前两个变换的狄利克雷积表示。。id = φ × 1 。。e = μ × 1 。。再举个例子。。ΣGcd(n,k) (k<=n) = φ × id。。

id = φ × 1，φ = id × μ。。1 = e × 1，e = 1 × μ。。

莫比乌斯反演。。对于任意积性函数 f、g 有。。f = g × 1 <==> g = 1 × μ。。理论上来说用不到。。太复杂了。。

一些常见的数论变换！

1D。。(k<=n)

Σ e(Gcd(n,k)) 。。φ。。；

Σ Gcd(n,k) 。。 φ × id。。；O(N) - O(sqrtN)。。

Σ e(Gcd(n,k))\*k 。。可容斥。。也可数论变换。。id\*(φ+e)/2。。O(N) - O(1)。。

2D。。(i<=n，j<=m)

ΣΣ e(Gcd(i,j)) 。。 Σ μ(d)\*(n/d)\*(m/d) (d<=n)。。；O(N) - O(sqrtN)。。

ΣΣ Gcd(i,j)。。 Σ φ(d)\*(n/d)\*(m/d) (d<=n)。。；O(N) - O(sqrtN)。。

1D Lcm Sum。。Σ Lcm(n,k) = n(((id\*φ)+1)×1)/2。。由 1D 中互质的和推出。。O(N) - O(sqrtN)。。

2D Lcm Sum。。ΣΣ Lcm(i,j)。。

。。O(n) 做法。。

= Σ d\*S(n/d,m/d) (d<=n) 。。其中 S(p,q) 表示 (i<=p,j<=q) 中互质对数的乘积。。

这里第一次分块。。

S(n,m) = ΣΣ i\*j\*e(gcd(i,j)) = ΣΣ i\*j\*Σμ(d) (d|i && d|j) = Σ μ(d)\*d\*d\*Sum(n/d,m/d)。。

这里第二次分块。。

其中 Sum(n,m) 为 (1~n)到(1~m) 中对数的乘积，即 Σi \* Σj。。

**博弈论**

SG函数 和 SG定理 是博弈中的核心。

因为一个组合游戏的一个状态要么必胜要么必败。必败态后继没有必胜态，必胜态后继必有一个必败态。根据这个可以更加转换成 SG函数。SG定理其实就是多个组合游戏的和，道理是一样的。

思考一。。

（sg函数 -> 当前的SG值如果非零，则为必胜态。必有一个方法达到必败态，即异或一个 子游戏的SG值 和该子游戏中其中一个后继状态的SG值。即为SG定理。。  
一个状态 -> 一个/几个游戏的新状态。。之前的状态消失了。。即用现异或值去异或原状态和新状态。。

SG 定理。。一个大游戏的 SG 值之所以是好多子游戏的 SG 值的异或和是因为只要动了任何一个子游戏新的大游戏SG值一定不会取这个值。。而且比该值小的值肯定有一种方法可以达到。。（用最高位的思想来考虑即可。。）

对于组合游戏的求解。。状态的设计是至关重要的。。

思考二。。

当然也存在用dp来解决的博弈问题。。这种题并不是SG函数无法解决，而是没有一种合理的方式来表示状态。。换一种思路的话，即放弃SG函数，直接用记忆化搜索会让问题变得更简单。。

在进行记忆化搜索的时候要考虑清楚当前状态下是谁行动。。不过这都是一些小细节问题。。综合来说还是学好dp这类中档难度的问题一般都可以迎刃而解。。

一般的dp状态设计是在当前状态下先手的最优方法，或先手+后手的最优方法（有时候只记录先手并不能导出后手的最优方法，比如poj1678。。不过要是当前状态的总值是定值则显然只记录先手即可。。）。。

思考三。。

好多看似新颖的游戏都可以转化成一些经典的游戏。。转化的时候只要每个状态（的值）每个转移都能对应上，即便看似再荒谬也是一个等价转化。。！。。如果实在无法找到转化的方法当然打表找规律也是比较常见的。。

Anti - Nim。。谁走最后一步即是 Loser 。。

网上那道多校题目完全是错的。。Hdu 3590。。09年论文中很详细地有写：

对于任意一个Anti-SG游戏，如果我们规定当局面中所有的单一游戏的SG值为0时，游戏结束，则先手必胜当且仅当：（1）游戏的SG函数不为0且游戏中某个单一游戏的SG函数大于1；（2）游戏的SG函数为0且游戏中没有单一游戏的SG函数大于1。

这个前提条件是非常重要的。。

Every - SG。。每次行动每个子游戏都要走一步。。

考虑让每个必胜的子游戏尽可能长地玩下去，必败的尽可能短。。然后就是简单的 DP 了。。

dp(u) = 0, max(step(v))+1), min(step(v))+1  
结论：先手必胜当且仅当单一游戏中最大的 dp 值为奇数。。关键在于 SG=0 和 SG!=0 的点的 dp 值的奇偶性是不同的。。

树的删边游戏。。SG(u) = (SG(v0)+1) ^ (SG(v1)+1) ^ ...。。非常好证。。

无向图删边游戏（环不共用边 && 与基础树只有一个公共点）。。奇环切断后链长同奇偶。。SG值为1，用长为 1 的链替代。。偶环切断后链长异奇偶。。SG值为0，直接删掉。。变成了树的删边游戏。。

翻硬币游戏。。绝大多数情况下都可以转化成一些经典的 Nim 游戏。。在有条件规定最右必须从 Heads 翻到 Tails 这种条件时是可以归纳成每个 Heads 朝上棋子时单一存在时的 SG 值的异或和。。

不平等博弈

NULL。。QAQ

**省赛临阵磨枪**

**后缀数组：**

一、做后缀数组的时候一定要加上最后一个字符。。而且最后一个字符一定要是 \0 。。它可以很好的控制不等长串之间的字典序大小。

二、在 contencate 两个串的时候，中间要用一个完全没有出现过的符号隔开。。当然不能用 \0 。。一般用 $ 是比较好的 23333。。

三、敲模板的时候眼睛擦亮点。。别整的模板都没抄对。。

四、大概如果一道题真的只有后缀数组的话。。还是很好写的。不过要套上个什么乱七八糟的。。就真的不太好调了。。（比如笛卡尔树www。。好可怕。。

**Splay Tree**

一、在最初建树的过程中不要图方便给建成一个链。。不如去写一个类似线段树那样 build(l, r, mid) 函数。。上来直接给弄成平衡的。。

二、最好手写一个 find(x) 函数。。用来返回排在第 x 的节点值。。find 后立即提到根节点。。

三、在某点后面插入一个节点时，先将该节点提到根，然后裸插就行，分类讨论该点右子是否为 Null

四、删除一个节点时，先将该节点提到根，分类讨论左子是否为 Null；在删除完节点后千万别让整棵树的 sz -- 。。因为整棵树的 sz 作用只在插入新节点的时候给节点编号起了作用。。删除节点后只是让这个节点和整个树都脱离了联系。。并不是真正意义上的删除。

五、在进行操作的过程中，随时注意哪个点时根。。

六、想要把某个点提到根的左子、右子上，只需先把根和那颗子树的边去掉；然后直接 Splay 那个点就行，最后再维护一下根和该点就好。

七、0 节点的父亲和 0 节点的 sz 值要时刻保持为 0 。。否则一定不对。

八、别忘了加上两个虚拟点；不然在询问区间设计到两头的是无法处理的。。进行询问的时候先把闭区间变成开区间，把开区间左端点转到根，右端点转到根的右子，直接返回右端点的左子树的答案就行。

九、标记下传的时候和线段树是类似的，在某个点有标记的时候该点的值已经维护完毕；下传时需修改子节点的标记值和维护值；下传的时候在 update 和 splay 的时候。

**要删除等于一个值的一个元素（multiset）**Set.erase(Set.find(x));

**vector 去重 。。** vec2.erase(unique(vec2.begin(), vec2.end()), vec2.end());

**开栈。。**

int size = 256 << 20; // 256MB

char \*p = (char\*)malloc(size) + size;

\_\_asm\_\_("movl %0, %%esp\n" :: "r"(p));

#pragma comment(linker, "/STACK:102400000,102400000")