# 数学

~~玄之又玄~~

## 快速幂

ll fpow(ll x, ll y, ll p) {  
 ll ret = 1;  
 while (y > 0) {  
 if (y % 2 == 1) {  
 ret = ret \* x % p;  
 }  
 x = x \* x % p;  
 y /= 2;  
 }  
 return ret;  
}

上面代码中，ans用于存储答案，b表示m的2^x次方

## 分解质因数

先提供一种朴素做法：

vector<int> breakdown(int N) {  
 vector<int> result;  
 for (int i = 2; i \* i <= N; i++) {  
 if (N % i == 0) { // 如果 i 能够整除 N，说明 i 为 N 的一个质因子。  
 while (N % i == 0) N /= i;  
 result.push\_back(i);  
 }  
 }  
 if (N != 1) { // 说明再经过操作之后 N 留下了一个素数  
 result.push\_back(N);  
 }  
 return result;  
}

## 欧拉筛（线性筛）

基本的思想是筛到 i % prime[j] == 0 就不筛了，保证每个数只被筛一次。

struct Euler\_sieve {  
 vector<int> pri;  
 bool not\_prime[N];  
  
 void init(int k) {  
 for (int i = 2; i <= n; ++i) {  
 if (!not\_prime[i]) {  
 pri.push\_back(i);  
 }  
 for (int pri\_j : pri) {  
 if (i \* pri\_j > n) break;  
 not\_prime[i \* pri\_j] = true;  
 if (i % pri\_j == 0) break;  
 // i % pri\_j == 0  
 // 换言之，i 之前被 pri\_j 筛过了  
 // 由于 pri 里面质数是从小到大的，所以 i 乘上其他的质数的结果一定会被 pri\_j 的倍数筛掉，就不需要在这里先筛一次，所以这里直接 break 掉就好了  
 }  
 }  
 }  
   
 void is\_prime(int x) {  
 return !not\_prime[x];  
 }  
};

### 欧拉函数 $ $

$ (n) $ 表示小于等于 $ n $ 和 $ n $ 互质的数的个数，譬如 $ (1) = 1 $。

下面列举一些性质

* 欧拉函数是积性函数，但不是完全的积性函数。具体而言，若 $ (a, b) = 1 $，那么 $ (a b) = (a) (b) $。特别地，当 $ n $ 是奇数时，有 $ (2n) = 2 (n) $。
* $ n = \_{d n} (d) $。
* 由定义，若 $ n = p^k $，其中 $ p $ 是质数，那么 $ (n) = p^k - p^{k - 1} $。
* 设 $ n $ 的质因数分解为 $ n = *{i = 1}^{s} p*{i}^{k\_i} $，其中 $ p\_i $ 是质数，则有 $ (n) = n \_{i = 1}^s $。

由于第四条性质，显然如果只求一个数的欧拉函数，那可以在分解质因数的同时求解。

int euler\_phi(int n) {  
 int ans = n;  
 for (int i = 2; i \* i <= n; i++)  
 if (n % i == 0) {  
 ans = ans / i \* (i - 1);  
 while (n % i == 0) n /= i;  
 }  
 if (n > 1) ans = ans / n \* (n - 1);  
 return ans;  
}

* **线性筛求欧拉函数**

### 线性筛之约数个数

## 数论

### GCD & LCM

记得开long long

long long gcd(long long a, long long b) {  
 if (b == 0) return a;  
 else return gcd(b, a % b);  
}

### 同余理论和乘法逆元

以上不仅适用于加法，亦适用于减法、乘法。

* 模意义下的除法（**乘法逆元**）

若存在，使得，则称是在对取模意义下的乘法逆元。也就是说相当于在取模意义下，除以就相当于乘以。

拓展欧几里得求法：

long long ExGCD(long long a, long long b, long long &x, long long &y) {  
 if(b == 0) {  
 x = 1, y = 0;  
 return a;  
 }  
 long long d = ExGCD(b, a % b, x, y), t = x;  
 x = y, y = t - a / b \* x;  
 return d;   
}  
  
int ExGcdInv(int a, int b) { // a在模b意义下的逆元  
 int x, y;  
 ExGCD(a, b, x, y);  
 return x;  
}

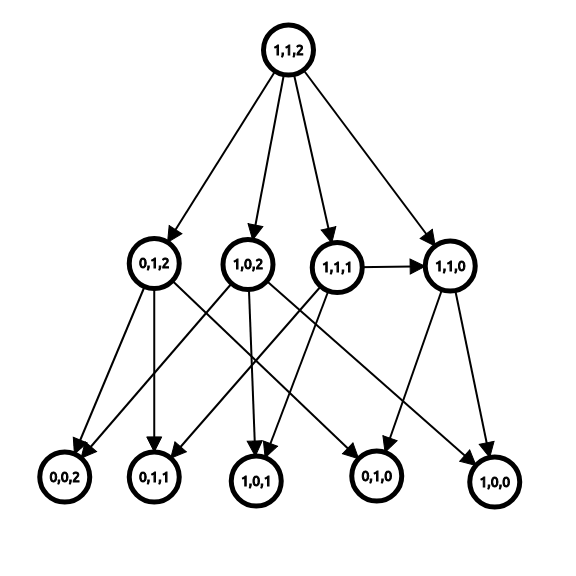
费马小定理求法（使用模数是大质数的情况）：

int FermatInv(int a, int b) {  
 return PowMod(a, b - 2, b); // 快速幂  
}

## 博弈论

一般的公平组合游戏以玩家无法行动为结束；每个状态能做出的操作与玩家无关；同一状态无法多次到达。

若把状态视作点，边视作转移，那公平组合游戏可以看成一个 DAG：



DAG

先定义必胜状态和必败状态：

1. 没有后继的状态是必败状态；
2. 至少有一个后继的状态是必胜状态；
3. 后继全是必败的状态是必败状态。

* Nim 游戏

堆物品，每一堆物品有个，两个玩家轮流取走任意一堆的任意个物品，但不能不取。取走最后一个物品的人获胜。

**Nim 和**，当且仅当 Nim 和为0时该状态为必败状态。

#include <cstdio>  
#include <iostream>  
using namespace std;  
  
int main() {  
 int t; scanf("%d", &t);  
 while (t--) {  
 int n, x, s = 0; scanf("%d", &n);  
 while (n--) {  
 scanf("%d", &x); s ^= x;  
 }  
 if (s) printf("Yes\n"); else printf("No\n");  
 }  
 return 0;  
}

### SG 定理

定义 函数表示集合中没有出现的最小的元素：

给定DAG 中节点 和它的所有**直接相连**后继 ，可以定义 函数如下：

表示与 节点**直接相连**的所有节点后继的 值中没有出现的最小的，规定**没有后继的必败状态节点** 。

对于有多个有向图组成的组合游戏，它们的起点分别为 ，则有定力：**当且仅当** **时，这个游戏是先手必败的。**

这一定理被称作 Sprague-Grundy 定理，简称 SG 定理。

SG可以递归地求。——林ye