数学模板的整理

1. 数论基础知识和方法：
2. 素数判定相关
3. 整数分解
4. 欧拉函数
5. 欧几里得及其扩展
6. 乘法逆元
7. 同余方程
8. 中国剩余定理及其扩展
9. 数论高级方法：
10. Lucas定理及其扩展
11. 线性筛
12. 莫比乌斯反演
13. FFT及NTT
14. 容斥及二项式定理及排列组合
15. 矩阵相关：
16. 矩阵优化线性递推
17. 高斯消元及线性基
18. 计算几何（见[计算几何大模板](计算几何模板大集合.cpp)）
19. 基础知识
20. 凸包
21. 半平面交
22. 旋转卡壳
23. 辛普森积分
24. 最小圆覆盖
25. 杂项
26. 概率与期望
27. 博弈论

素数判定及其相关：

int w=sqrt(n+0.5);

for(int i=1;i<=w;i++)

{

if(n%i==0)return 0;

}

return 1;

普通筛法

prime[1]=2;

for(int i=2;i<=n;i++)

{

if(inq[i])continue;

prime[++tot]=i;

for(int ii=i;(long long)i\*ii<=n;ii++)

{

inq[i\*ii]=1;

}

}

线性筛：见后面

Miller\_Rabin算法：

基本思想基于费马小定理的逆及二次探测定理，详情见代码注释：

//大整数相乘并对大模数(longlong)取模

long long mul(long long a,long long b,long long k)

{

//主要是将乘法变为加法，依次枚举b的二进制位来进行。

long long ret=0,tt=a%k;

while(b)

{

if(b&1)ret=(ret+tt)%k;

tt=(tt<<1)%k;b>>=1;

}

return ret;

}

long long qkpow(long long a,long long r,long long mod)

{

long long ans=1;

while(r)

{

if(r&1)ans=mul(ans,a,mod);

a=mul(a,a,mod);r>>=1;

}

return ans;

}

//先对n与2的关系进行判定，再对a求r次方，然后不断进行二次探测即可。

bool miller\_rabin(long long n)

{

if(n<2)return false;

if(n==2)return true;

if(n%2==0)return false;

//n-1转换为r\*(2的d次方)

long long r=n-1,d=0,a,x,xx;

while(r%2==0)d++,r=r/2;

for(int i=0;i<=10;i++)

{

a=rand()%(n-2)+2;

x=qkpow(a,r,n);

for(int ii=1;ii<=d;ii++)

{

xx=mul(x,x,n);

//若x\*x mod p =1,x=1或x=n-1;

if(xx==1&&x!=1&&x!=n-1)return false;

x=xx;

}

if(xx!=1)return false;

}

return true;

}

void work()

{

srand(20001217);

//调用miller\_rabin即可

}

欧拉函数：

𝜑(𝑛)表示 1..n 中与 n 互素的正整数个数

当 n=1 时𝜑(1) = 1。

下面是利用容斥原理推导的：

设𝑝0, 𝑝1, ⋯ , 𝑝𝑘是 n 的不同质因子，则有

欧拉函数𝝋(𝒏)的性质:

性质 1、当𝑛为质数时，𝜑(𝑛) = n − 1。

性质 2、当(𝑛, 𝑚) = 1时， 𝜑(𝑛 × 𝑚) = 𝜑(𝑛) × 𝜑(𝑚)。（理解：结合公式理解）

性质 3、假设素数𝑝是𝑛的质因数，那么

如果𝑝还能整除𝑛/𝑝, 𝜑(𝑛) = 𝜑(𝑛/p) ∗ 𝑝 (理解：根据公式，质因数并未增多)

如果𝑝不能整除𝑛/𝑝, 𝜑(𝑛) = 𝜑(𝑛/p) ∗ (𝑝 − 1) (理解：结合性质2积性函数的性质)

性质 4、当𝑛 > 1时，1. . 𝑛 − 1中与𝑛互质的数的和为，证明考虑对称性，即gcd(x,n)==1,那么gcd(n-x,n)==1,根据加减的辗转相除法。

性质 5、

令g[n]=,那么gcd(i,j)==1时，g[ij]=g[i]\*g[j](结合乘法原理思考)，那么考虑n=p^a时

故得证

欧拉函数：n,a为正整数，gcd(n,a)==1,，

一个小扩展:当p>=𝝋(n)时，，不需要gcd(n,a)==1

int phi[maxn];

void get\_phi()

{

phi[1]=1;

int mn=100000;

for(int i=2;i<=mn;i++)

{

if(!inq[i])

{

prime[++tot]=i;

phi[i]=i-1;

}

for(int ii=1;ii<=tot&&prime[ii]\*i<=mn;ii++)

{

inq[prime[ii]\*i]=1;

if(i%prime[ii]==0)

{

phi[i\*prime[ii]]=prime[ii]\*phi[i];

break;

}

phi[prime[ii]\*i]=(prime[ii]-1)\*phi[i];

}

}

}

用途：求一些与gcd有关的，推导的方法和莫比乌斯大致相同，即推出特征式（即定义式或前面的各种性质中的式子），和莫比乌斯反演没必要分得那么开，毕竟都是推式子，推出特征式后该用谁用谁。

整数分解：

方法一O(sqrt(n))枚举质因数或直接枚举质数，注意要特判除出来后该数是否为1，否则该数仍然为该数的质因数

方法二 rho大整数分解

#include<iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

#include<cstdlib>

const int maxn=1000005;

using namespace std;

long long sta[maxn];int top=0;

long long mul(long long a,long long b,long long k)

{

long long ans=0;a=a%k;

while(b)

{

if(b&1)ans=(ans+a)%k;

a=(a<<1)%k;b>>=1;

}

return ans;

}

long long qkpow(long long a,long long t,long long mod)

{

long long ans=1;

while(t)

{

if(t&1)ans=mul(ans,a,mod);

a=mul(a,a,mod);t=t>>1;

}

return ans;

}

int miller\_rabin(long long n)

{

if(n<2)return 0;

if(n==2)return 1;

if(n%2==0)return 0;

long long r=n-1,d=0,x,xx;

while(r%2==0)r/=2,d++;

for(int i=1;i<=10;i++)

{

x=qkpow(rand()%(n-2)+2,r,n);

for(int ii=1;ii<=d;ii++)

{

xx=mul(x,x,n);

if(xx==1&&x!=1&&x!=n-1)return 0;

x=xx;

}

if(x!=1)return 0;

}

return 1;

}

long long gcd(long long x,long long y)

{

if(y==0)return x;

else return gcd(y,x%y);

}

//这里判圈是不断扩大圈的范围，同时保证起点，当圈绕回来时就可以退了。

long long rho(long long n,long long c)

{

long long x=rand()%n,y=x,t=n,k=2;

for(long long i=2;;i++)

{

x=(mul(x,x,n)+c)%n;

t=gcd(abs(x-y),n);

if(t>1||y==x)break;//判有没有找到一个质因数和圈是否已经开始循环了。

if(i==k)y=x,k<<=1;//这里在扩大圈的长度，同时让起点==x;

}

return t;

}

//流程：判当前是否是质因数，否则找当前的一个因数，然后递归即可。

void zys(long long n)//求出来的质数不一定有序，要重新排过，且若该数有p^a这个因数，那么求出来的为a个p,而不是一个

{

if(n==1)return;

if(miller\_rabin(n)){sta[++top]=n;return;}//判是否为素数

long long t=n;

while(t>=n)t=rho(n,rand()%(n-1)+1);//找出一个因数

zys(t);

zys(n/t);

}

int main()

{

srand(20001217);

long long x=0;

while(cin>>x)

{

top=0;

zys(x);

sort(sta+1,sta+top+1);

top=unique(sta+1,sta+top+1)-sta;top--;

cout<<x<<" = ";

for(int i=1;i<=top;i++)

{

int tmp=0;

cout<<sta[i];

while(x%sta[i]==0)

{

tmp++;

x=x/sta[i];

}

if(tmp!=1)cout<<'^'<<tmp;

if(i!=top)cout<<" \* ";

}

cout<<'\n';

}

return 0;

}

欧几里德及其扩展，乘法逆元的三种求法（递推（要求质数），exgcd(要求互质)，qkpow(mod-2)（要求质数））均见lucas。

中国剩余定理：代码仍然见lucas。

扩展中国剩余定理：

ll exgcd(ll a,ll b,ll &x,ll &y)

{

if(b==0){x=1,y=0;return a;}

ll g=exgcd(b,a%b,y,x);y-=(a/b)\*x;

return g;

}

ll lcm(ll a,ll b)

{

ll x,y;

ll g=exgcd(a,b,x,y);

return a/g\*b;

}

ll calc(ll a,ll b,ll c)

{

ll x,y;

ll g=exgcd(a,b,x,y);

if(c%g!=0)return -1;

ll t=b/g;

x=c/g\*x;

return (x%t+t)%t;

}

ll a[maxn],b[maxn];

ll crt(ll m)

{

for(int i=2;i<=m;i++)

{

ll tmp=calc(a[i-1],a[i],b[i]-b[i-1]);

if(tmp<0)return 0;

a[i]=lcm(a[i],a[i-1]);

b[i]=(tmp\*a[i-1]+b[i-1])%a[i];

}

return b[m];

}

同余方程：

LL calc(LL a,LL b,LL c) //得到 ax+by=c 中 x 最小的非负整数解

{

LL x,y,d;

d=exgcd(a,b,x,y);

if(c%d) return -1;

//目前x为ax+by=gcd(a,b)的解。

x=x\*(c/d);

LL t=abs(b/d); //通解：x'=x+t\*k(t=b/d)以及x'=x(mod t)

return (x%t+t)%t; //最小的非负整数解

}

普通lucas

//以下是核心

long long inv[mod+5],f[mod+5];

void ready()

{

f[0]=1;

for(int i=1;i<mod;i++)

f[i]=f[i-1]\*i%mod;

inv[1]=1;

for(int i=2;i<mod;i++)

inv[i]=(mod-mod/i)\*inv[mod%i]%mod;

inv[0]=1;

for(int i=1;i<mod;i++)

inv[i]=inv[i-1]\*inv[i]%mod;

}

long long lucas(int n,int m)//直接带n,m即可求出其中n>m;

{

if(n<m)return 0;//判特殊情况

if(n<mod&&m<mod)return f[n]\*inv[m]%mod\*inv[n-m]%mod;//判可直接求的地方

return lucas(n/mod,m/mod)\*lucas(n%mod,m%mod)%mod;

}

Lucas定理及其扩展

//以下为核心，扩展Lucas定理是用来求 c(n,m) mod p^k，p为素数的值,

//一般后接中国剩余定理把modnum拆成这个样子。

int top=0;

long long p,n,m,w[9],pri[maxn],pt[maxn];//p为模数

void getfac(long long p)

{

for(long long i=2;i\*i<=p;i++)

{

if(p%i==0)pri[++top]=1,pt[top]=i;

while(p%i==0)pri[top]\*=i,p/=i;

}

if(p!=1)pri[++top]=p,pt[top]=p;

}

long long qkpow(long long a,long long t,long long mod)

{

long long ans=1;

while(t)

{

if(t&1)ans=ans\*a%mod;

a=(a\*a)%mod;t>>=1;

}

return ans;

}

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 g=exgcd(b,a%b,y,x); y-=(a/b)\*x;

return g;

}

long long getinv(long long a,long long mod)

{

long long x,y;

long long g=exgcd(a,mod,x,y);

return g==1?(x+mod)%mod:-1;

}

//扩展lucas求阶乘

ll getfac(ll n,ll pr,ll p)//pr为模数，p为要模的质数

{

if(n==0)return 1;

ll re=1;

if(n>=pr)

{

for(int i=2;i<pr;i++)//首先大于p的会不断循环，顾算一个

if(i%p)re=re\*i%pr;

re=qkpow(re,n/pr,pr);//qkpow n/p次

}

ll r=n%pr;//暴力算最后一截

for(int i=2;i<=r;i++)

if(i%p)re=re\*i%pr;

return re\*getfac(n/p,pr,p)%pr;//多的递归处理

}

ll getmyc(ll n,ll m,ll pr,ll p)

{

if(m>n)return 0;

//统计需要乘上的p的个数

ll c=0;

for(ll i=n;i;i/=p)c+=i/p;

for(ll i=m;i;i/=p)c-=i/p;

for(ll i=n-m;i;i/=p)c-=i/p;

ll tmp=qkpow(p,c,pr);

if(tmp==0)return 0;

//计算阶乘

ll x=getfac(n,pr,p),y=getfac(m,pr,p),z=getfac(n-m,pr,p);

ll ans=x\*getinv(y,pr)%pr\*getinv(z,pr)%pr\*tmp%pr;//把y,z变成逆元，再乘上qkpow;

return ans;

}

//

long long crt(long long n,long long m)//中国剩余定理

{

long long re=0;

for(int i=1;i<=top;i++)

{

long long wmod=p/pri[i];//inv是在算y,\*wmod是在算x,然后\*getc得答案。

long long a=getmyc(n,m,pri[i],pt[i])\*wmod%p\*getinv(wmod,pri[i])%p;

//cout<<"haha:"<<getc(n,m,pri[i],pt[i])<<'\n';

re=(re+a)%p;

}

return re;

}

核心应用：优化求大组合数对较小数取模，其中lucas定理复杂度为O（），其中p为模数，q为组合数中较大者，扩展Lucas 时间复杂度约为O（），带较大常数，它的时间复杂度主要花在求阶乘的过程中暴力计算和递归部分，而快速幂部分也要耗时，但非主要。

Lucas的难点不在于lucas定理，往往在于推出组合数以及对组合数式子的优化处理，对于推式子型，往往要想其实际意义或发现其规律，从而加以运用，而优化型考虑其共同部分（或循环部分）加以提出，或看它的内在规律，或该式的实际意义来重新优化，（具体分析见每道题目（笔记本上））。

Lucas定理与卡常数：考虑hnoi2017的lucas题目，共一下几种方法：

预处理出要用到的逆元，组合数数组，快速幂。对一些常量事先求出，考场上直接写到代码里，而不是在程序中求。在扩展lucas中若模数已知，事先分解该模数，预处理p的循环数组（fac里的那个循环预处理出），同时在getmyc函数中先处理快速幂，为0了直接退出不在算阶乘。

线性筛：

用途：O(n)筛积性函数和筛质数。

基本思想：线性筛的线性体现在使每个数只被“筛”一次

关于复杂度：

线性筛的关键就在于这句话if(i%prime[ii]==0)break;

传统的筛法之所以慢，是因为每个数会被筛多次，而线性筛的每一个数只会被它的最小质因子筛一次，因为如果一个数x被非最小质因数筛掉了，设该数x=prime\*w,那么w一定为x的最小质因数的倍数，那么它一定不会有机会\*prime,因为它会在乘上最小质因子后被筛掉，顾线性筛是线性的。例如：12只会被质因数2筛掉，而不会被3筛掉。

筛积性函数的套路：

讨论三种情况：设该函数为f[i]

1.当i为质数时，f[i]的值

2.当i为prime\*ii,其中gcd(prime,ii)==1时,f[i]的值。

3.当i为prime\*ii,其中ii为prime的倍数时f[i]的值。

莫比乌斯反演：

该题为bzoj2301

对于给出的n个询问，每次求有多少个数对(x,y)，满足a≤x≤b，c≤y≤d，且gcd(x,y) = k，gcd(x,y)函数为x和y的最大公约数。

#include<iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

#include<cmath>

const int maxn=50005;

using namespace std;

int prime[maxn],inq[maxn],mu[maxn],n,tot=0;

void get\_mu()//筛莫比乌斯函数

{

mu[1]=1;

for(int i=2;i<=50000;i++)

{

if(!inq[i])

{

prime[++tot]=i;

mu[i]=-1;//此处为为质数的值

}

for(int ii=1;(long long)prime[ii]\*i<=50000;ii++)

{

inq[prime[ii]\*i]=1;

if(i%prime[ii]==0)

{

mu[i\*prime[ii]]=0;//此处为倍数的值

break;

}

mu[prime[ii]\*i]=-mu[i];

//prime[ii]\*i只会被筛到一次，而prime[ii]为-1；此处为互质的值

}

}

for(int i=1;i<=50000;i++)//求莫比乌斯前缀和

{

mu[i]+=mu[i-1];

}

}

int work(int a,int b,int k)//此为常用操作，是为了求，但直接枚举i显然太慢，我们可以找出n/I,m/I的值变化的地方，然后这里用了前缀和。

{

int tmp=0;

int last=0;

for(int i=1;(long long)i\*k<=a&&(long long)i\*k<=b;i=last+1)

{

last=min(a/(a/i),b/(b/i));

tmp+=(mu[last]-mu[i-1])\*(a/(last\*k))\*(b/(last\*k));

}

return tmp;

}

μ(i)(以下用u代替μ):

1.u[1]=1.

2.u[w]=(-1)^k,w为k个不同质数的乘积.

3.其他情况均为0.

那么我们如何筛？

考虑上一页的3种情况。

首先当i为质数，u[i]显然为-1。然后当i为prime\*ii,其中gcd(prime,ii)==1时,u[i]=-u[ii],考虑u第二条性质可得。 最后当i为prime\*ii,其中ii为prime的倍数时,u[i]为0。

证明：

这里要用到

该式的证明：

当n==1时显然，

当n!=1时，首先分解n为,首先u不为0的仅有ai=1或ai=0时的情况，那么原式=

考虑二项式定理：

我们令x=-1,y=1即可证明。

当我们通常通过

得：

这样就可以处理前缀和了，

ll getans(int n,int m)

{

int last=0;

ll ans=0;

for(int i=1;i<=n;i=last+1)

{

last=min(n/(n/i),m/(m/i));

ans=(ans+(mu[last]-mu[i-1])\*(n/last)\*(m/last));

}

return ans;

}

莫比乌斯解题策略：

对每道题的具体分析见笔记本

核心思想：勇于不断尝试，当一种表示化不出来的时候，就考虑交换一下sigma之类的顺序，过程中不要害怕式子变复杂，要想化简单，一定需要将一些项提出，过程自然会变复杂，当结果会变简单。

目的：将式子中共有的部分提出，加以预处理，对相同的项要分别乘的求和后在乘，总之，省略一些重复的步骤，从而加以优化。

套路：想将办法往gcd上靠，因为一旦出现gcd，我们就可以往莫比乌斯上套，例如（更多见笔记本）

而含u的式子往往可以通过交换使其脱离和n,m的关系从而预处理，例如sdoi2017把u提出来使其脱离n,m来做。

FFT与NTT

struct fftnum//fftnum结构体的定义

{

double x;

double y;

fftnum(){}

fftnum(double \_x,double \_y){x=\_x;y=\_y;}

friend fftnum operator +(const fftnum &x,const fftnum &y)

{

return fftnum(x.x+y.x,x.y+y.y);

}

friend fftnum operator -(const fftnum &x,const fftnum &y)

{

return fftnum(x.x-y.x,x.y-y.y);

}

friend fftnum operator \*(const fftnum &x,const fftnum &y)

{

return fftnum(x.x\*y.x-x.y\*y.y,x.y\*y.x+x.x\*y.y);

}

};

void fft(fftnum \*c,int f)

{

for(int i=0;i<n;i++)if(i<r[i])swap(c[i],c[r[i]]);

for(int i=1;i<n;i=(i<<1))//第一维控制将被操作的长度

{

fftnum wn(cos(pi/i),sin(f\*pi/i)),x,y;//先cos后sin

for(int ii=0;ii<n;ii+=(i<<1))//第二维控制起点

{

fftnum w(1,0);

for(int i3=0;i3<i;i3++,w=w\*wn)//合并最低的两块

{

x=c[ii+i3];y=w\*c[i+ii+i3];//都是前一个无变化，后一个要乘

c[ii+i3]=x+y;

c[i+ii+i3]=x-y;

}

}

}

}

//a,b为目标数组,下面不变即可。

\_scanf(a,n);

\_scanf(b,n);

m=n\*2;

for(n=1;n<m;n=(n<<1))l++;

for(int i=0;i<n;i++)r[i]=((r[i>>1]>>1)|((i&1)<<(l-1)));

fft(a,1);

fft(b,1);

for(int i=0;i<n;i++)a[i]=a[i]\*b[i];

fft(a,-1);//fft逆过程，-1即可

for(int i=0;i<n;i++)a[i].x/=n,a[i].y/=n;//注意除以n

//此时a便为新函数的值。

<bz3992NTT.cpp>

用途：优化卷积，优化多项式乘法，优化卷积时可以结合多项式乘法理解即可,卷积：

难点仍然不在其本身，关键在于推导出w[i]的表达式，优化dp时dp的转移方程。FFT也可以与其它知识混用，例如分治（就是类似于快速幂）:

那么我们对于g提出来后进行快速幂，即：

多项式可以直接理解为一个数，而卷积即是多项式和多项式之间的乘法，我们对g直接进行卷积，多余的部分舍去，因为我们进行的是多项式乘法，而对于某些题目，多的部分是需要模下来的，这个时候就要结合题意和多项式本身来理解。

二项式定理：

证明：(𝑎 + 𝑏) ^n = (𝑎 + 𝑏) × (𝑎 + 𝑏) × ⋯ × (𝑎 + 𝑏),考虑每一项选a,还是选b。

容斥：

首先分析全集，然后考虑不同的性质，往往答案就为

做题时要弄清楚定义，搞清楚到底什么是性质，什么是全集，然后往上面套。

for(int i=0;i<(1<<(n-1));i++)

{

r[i]=r[i>>1]+(i&1);

for(int ii=1;ii<n;ii++)

{

if((i>>(ii-1))&1)

work();

}

//注意这里要分清性质到底是什么，这里应该加还是减

if((n-1-r[i])&1)ans-=getans();

else ans+=getans();

}

举一个例子：cqoi2015选数，全集为gcd为k的倍数的方案数，性质为gcd为k的i倍的倍数的方案数。这里可以dp把方案改为恰好为k的i倍的方案数，这样就能省掉容斥。

容斥一般具体实现也就两种，上面文本框中的一种，还有一种dp实现。dp实现的方法即特征：例如我们设出来的f[i]和之前的状态是有重叠的，那么我们就要该设f[i]为不含前面xxx的状态的方案数，从而进行容斥，这样就只需减一遍，不用减多次，还有就是以dp为主，只是转移时要容斥，例如有标号连通图的方案数个数（全部-不连通的）。

卡特兰数：

卡特兰数一般应用：主要是往它的递推式上套，然后就可以转为公式求解了。

矩阵优化线性递推：

<矩阵优化线性递推.cpp>

struct matrix

{

long long v[25][25];

int n,m;

matrix()

{

n=0;

m=0;

memset(v,0,sizeof(v));

}

friend matrix operator \*(matrix one,matrix two)

{

matrix c;

c.m=one.m;c.n=two.n;

for(int i=1;i<=c.m;i++)

for(int ii=1;ii<=c.n;ii++)

for(int i3=1;i3<=one.n;i3++)

{

c.v[i][ii]=(c.v[i][ii]+one.v[i][i3]\*two.v[i3][ii]%modnum)%modnum;

}

return c;

}

};

matrix mypow(matrix a,int t)

{

if(t<=0)

{

matrix c;//单位矩阵

c.n=c.m=a.m;

for(int i=1;i<=a.m;i++)

{

c.v[i][i]=1;

}

return c;

}

matrix c=mypow(a,t/2);

if(t%2==0)return c\*c;

return c\*c\*a;

}

void work()

{

matrix c,c2,c3;

c.n=c.m=k;

//先建立转移矩阵，即下一次该位置的f函数有那些函数构成，例如斐波那契数列就只需要两项。

for(int i=1;i<=k;i++)

{

c.v[1][i]=a[i];

}

for(int i=2;i<=c.m;i++)

{

c.v[i][i-1]=1;

}

c2.m=k,c2.n=1;//c2存初始值

for(int i=1;i<=k;i++)

{

c2.v[i][1]=f[k-i+1];

}

if(n>=2)c3=mypow(c,n-k)\*c2;//最后乘上来即可。

else

{

cout<<f[n]<<'\n';

return;

}

cout<<c3.v[1][1]<<'\n';

}

高斯消元和线性基

内附讲解

<bz3168矩阵求逆.cpp>

//矩阵求逆操作方法就是消a为单位矩阵，同时对单位矩阵进行同样的操作以达到构造出a矩阵的逆的效果。

设*A*是数域上的一个n阶[方阵](https://baike.baidu.com/item/%E6%96%B9%E9%98%B5" \t "_blank)，若在相同数域上存在另一个n阶矩*B*，使得： *AB*=*BA*=*E*。 则我们称*B*是*A*的逆矩阵，而A则被称为[可逆矩阵](https://baike.baidu.com/item/%E5%8F%AF%E9%80%86%E7%9F%A9%E9%98%B5/11035614" \t "_blank)。其中，E为单位矩阵。

int gauss(int n)

{

int r=0;

for(int i=1;i<=n;i++)

inv[i][i]=1;

for(int i=1;i<=n;i++)

{

r=i;

for(int ii=i+1;ii<=n;ii++)

if(myabs(a[ii][i])>myabs(a[r][i]))r=ii;

//对于矩阵求逆一定要从1开始走，因为逆矩阵并不一定被消成上三角矩阵，故都要操作

for(int ii=1;ii<=n;ii++)

{

swap(a[r][ii],a[i][ii]);

swap(inv[r][ii],inv[i][ii]);

}

if(a[i][i]==0)return -1;

long long t=qkpow(a[i][i],mod-2);

for(int ii=1;ii<=n;ii++)

{

a[i][ii]=a[i][ii]\*t%mod;

inv[i][ii]=inv[i][ii]\*t%mod;

}

for(int ii=1;ii<=n;ii++)

{

if(ii!=i&&a[ii][i]!=0)

{

t=a[ii][i];

for(int i3=1;i3<=n;i3++)//同理，不然这里同样要错。

{

a[ii][i3]=(a[ii][i3]-a[i][i3]\*t%mod+mod)%mod;

inv[ii][i3]=(inv[ii][i3]-inv[i][i3]\*t%mod+mod)%mod;

}

}

}

}

return 1;

}

[1944高斯消元带判无解和多解的模板.cpp](file:///D:\竞赛旧\moban\1944高斯消元带判无解和多解的模板.cpp)

int gauss(int n)

{

int r=0;

int t=0;

for(int i=1;i<=n;i++)//a数组n\*n的为方程前的系数，n+1项为该式子的答案,i为当前在解的是那个未知数

{

r=i-t;

for(int ii=i-t+1;ii<=n;ii++)

if(a[ii][i]>a[r][i])r=ii;

for(int ii=i;ii<=n+1;ii++)

swap(a[i-t][ii],a[r][ii]);

if(a[i-t][i]==0){t++;continue;}

di=a[i-t][i];

for(int ii=i;ii<=n+1;ii++)

a[i-t][ii]/=di;

for(int ii=i-t+1;ii<=n;ii++)

{

if(a[ii][i]!=0)

{

di=a[ii][i];

for(int i3=i;i3<=n+1;i3++)

{

a[ii][i3]-=a[i-t][i3]\*di;

}

}

}

}

if(t==0)

{//带回求值

ans[n]=a[n][n+1];

for(int i=n-1;i>=1;i--)

{

ans[i]=a[i][n+1];

for(int j=i+1;j<=n;j++)

ans[i]-=(a[i][j]\*ans[j]);

}

}

for(int i=n-t+1;i<=n;i++)

if(abs(a[i][n+1])>eps)return -1;//此时这些方程前面未知数前的系数均为0，而=右边不为0,故无解。

return t;//t即为自由元数量

}

[bz4031矩阵树定理&&辗转相除高斯消元.cpp](file:///D:\竞赛旧\moban\bz4031矩阵树定理&&辗转相除高斯消元.cpp)

矩阵树定理：

基尔霍夫Kirchhoff矩阵 K =度数矩阵 D - 邻接矩阵 A

度数矩阵：d[i][j]=0,d[i][i]等于id的度数

临接矩阵：d[i][j]=d[j][i]=i到j的边数

对于一个无向图 G ，它的生成树个数等于其基尔霍夫Kirchhoff矩阵任何一个N-1阶主子式的行列式的绝对值。

所谓的N-1阶主子式就是对于一个任意的一个 r ，将矩阵的第 r 行和第 r 列同时删去得到的新矩阵。

//该消元过程适合模数鬼畜而要求行列式的值不变的情况下

//倘若模数不鬼畜，用逆元即可。倘若不要求行列式的值不变，则可以取lcm来除，这样就可以避免除法。不用变原式

void gauss(int n)

{

int num=0;

for(int i=1;i<=n;i++)

{

num=i;

for(int ii=i+1;ii<=n;ii++)

if(Abs(c[ii][i])>Abs(c[num][i]))num=ii;

if(num!=i)

{

mark^=1;//在mod意义下均为正数，故要标记正负，否则并不知道最后是正和负

for(int ii=i;ii<=n;ii++)

swap(c[num][ii],c[i][ii]);

}

for(int ii=i+1;ii<=n;ii++)

{

while(c[ii][i]!=0)//利用辗转相除法来消去该列，这里不用mod的方法，而用乘的方法，否则有些项可能为0导致问题。

{

long long tmp=c[ii][i]/c[i][i];

for(int k=i;k<=n;k++)

c[ii][k]=(c[ii][k]+mod-tmp\*c[i][k]%mod)%mod;

if(c[ii][i]==0)break;//该式注意在此break

mark^=1;

for(int k=i;k<=n;k++) swap(c[ii][k],c[i][k]);

}

}

}

}

[1953线性基标准版求一堆数中任选一些数可以异或出的第k小的元素.cpp](1953线性基标准版求第k小的元素.cpp)：

线性基是一个**数的集合**，并且每个序列都拥有**至少一个**线性基，取线性基中若干个数异或起来可以得到原序列中的任何一个数。原序列里面的任意一个数都可以由线性基里面的一些数异或得到

线性基里面的任意一些数异或起来都不能得到0

线性基里面的数的个数唯一，并且在保持性质一的前提下，数的个数是最少的

int gauss(long long x)//O(nm+m^2);动态插入

{

for(int i=63;i>=0;i--)

{

if((x>>i)&1)

{

if(a[i]==0)

{

a[i]=x;

for(int ii=i-1;ii>=0;ii--)if((a[i]>>ii)&1)a[i]^=a[ii];//注意这里要先把该数多余的东西异或掉

for(int ii=63;ii>i;ii--)if((a[ii]>>i)&1)a[ii]^=a[i];//去异或掉高位的1

break;

}

else

{

x^=a[i];

if(x==0)return 1;//特判能否异或出0；

}

}

}

return 0;

}

int isok=0;

for(int ii=1;ii<=n;ii++)

{

\_scanf(ai);

isok=gauss(ai)?1:isok;

}

sort(a,a+65);

n=unique(a,a+65)-a;n--;

\_scanf(ai);//ai是询问的第k小

ai-=isok;

long long ans=0;

if((ai>>n)!=0)

{

printf("-1\n");

continue;

}

for(int i3=1;i3<=n;i3++)

if((ai>>(i3-1))&1)ans=ans^a[i3];

\_printf(ans);

高斯消元与期望部分：

高斯消元解图上dp序混乱的期望dp:

DP序

DP序是一道DP的所有状态的一个排列，使状态x所需的所有前置状态都位于状态x前；

此类DP中，存在DP序混乱的情况，一般是x的p/q可以转移到y，y的a/b可以转移到x。

此类方法的暴力做法：确定迭代次数，不断迭代，直至符合精度为止，但此法极易被卡。

那么高效做法？

做法1：这是概率dp的高效做法，即建立方程来进行求解。

见<https://blog.csdn.net/popoqqq/article/details/43481601>

十分详细，就是通过解方程来得到的高斯消元。

一般做法：

高斯消元即可，最后ans\*答案向量（例如只有p/d停止）即可。

其中ans为答案向量，S为初始的行向量，T为转移矩阵，I为单位矩阵。这个还可以直接矩阵求逆来算答案。

做法2：当上述方法不好想时或者转移不是一步一步转移时，可以用令一种方法：转期望dp,即算期望经过次数，因为到达终态后即意味游戏结束，故终态节点的概率即为经过该点的期望次数。那么设f[i]为经过该点的期望次数即可以转移并理解了。

同时对于高斯消元的期望版本，设从当前点到终点还需要经过的期望次数，就可以算了，同时对于一些情况可以移过来递推（即通过E[终态]为0来递推k和c）。

计算几何

//acos返回值在[0,pi]之间

//asin返回值在[-pi/2,pi/2]

//atan [-pi/2,pi/2]

//atan2(y,x),返回值[-pi,pi]

#include<iostream>

#include<cstring>

#include<algorithm>

#include<cstdio>

#include<cmath>

const double eps=1e-10;

const int maxn=310;

using namespace std;

int dcmp(double x)

{

if(fabs(x)<=eps)return 0;

else return x<0?-1:1;

}

struct point

{

double x,y;

point(double x0=0,double y0=0){x=x0;y=y0;}

};

typedef point vec;

vec operator +(const vec &x,const vec &y){return vec(x.x+y.x,x.y+y.y);}

vec operator -(const vec &x,const vec &y){return vec(x.x-y.x,x.y-y.y);}

vec operator \*(const vec &x,const double &y){return vec(x.x\*y,x.y\*y);}

vec operator /(const vec &x,const double &y){return (point){x.x/y,x.y/y};}

bool operator ==(const vec &x,const vec &y){return x.x==y.x&&x.y==y.y;}

bool operator <(const vec &x,const vec &y){return x.x!=y.x?x.x<y.x:x.y<y.y;}

double dot(const vec &x,const vec &y){return x.x\*y.x+x.y\*y.y;}

double cross(const vec &x,const vec &y){return x.x\*y.y-x.y\*y.x;}

double length(const vec &x){return sqrt(dot(x,x));}

//计算几何很多都是从dot和length引申出来的。

//规范相交，原理在于判两条线段的两个端点是否在异侧

//那么不需要规范相交呢?一定要单独判端点是否在另一条直线上，易判，否则两在同一直线上的不同线段会影响。

//先判四次点是否在直线上，再判四次点是否重叠(可不单独判，若相交不是规范相交的话)

bool spi(const point &x,const point &y,const point &z,const point &w)//规范相交

{

double t1=cross(y-w,z-w),t2=cross(x-w,z-w),

t3=cross(w-y,x-y),t4=cross(z-y,x-y);

if(dcmp(t1)\*dcmp(t2)<0&&dcmp(t3)\*dcmp(t4)<0)return true;

return false;

}

double dts(const point &p,const point &x,const point &y)//点到线段的距离

{

if(x==y)return length(p-x);

vec v1=p-x,v2=p-y,v3=y-x,v4=x-y;

if(dcmp(dot(v1,v3))<0)return length(v1);

else if(dcmp(dot(v2,v4))<0)return length(v2);

else return fabs(cross(v1,v3))/length(v3);

}

point getlinei(const point &p,const vec &v,const point &q,const vec &w)//两直线求交点

{

vec u=p-q;

double t=cross(w,u)/cross(v,w);

return p+v\*t;

}

//点在直线上：基本思路是x,p,y,也是规范在直线上，对于每个点要单独判。

bool ons(const point &p,const point &x,const point &y)//点在线段上，不含端点

{

//若要含端点,第二处改为<=0

return dcmp(cross(x-p,y-p))==0&&dcmp(dot(x-p,y-p))<0;

}

vec rotate(const vec &x,double ang){

return vec(x.x\*cos(ang)-x.y\*sin(ang),x.x\*sin(ang)+x.y\*cos(ang));}//旋转

vec normal(const vec &x){double t=length(x);return vec(-x.y/t,x.x/t);}//法向量

int getchull(point \*p,int n,point \*ch)//求凸包

{

sort(p+1,p+n+1);//首先凸包要先排序。

int m=0;

for(int i=1;i<=n;i++)

{

//这里取不取等于取决于凸包边上的点保不保留，方向到时候可再推

while(m>=2&&cross(ch[m]-ch[m-1],p[i]-ch[m-1])<=0)m--;

ch[++m]=p[i];

}

//两个细节：此处首先要新建底，其次是m>k

int k=m;

for(int i=n-1;i>=1;i--)//其次第n个点就不在算一遍了，因为已经放进去了，但1号点仍然要算，否则不对

{

while(m>k&&cross(ch[m]-ch[m-1],p[i]-ch[m-1])<=0)m--;

ch[++m]=p[i];

}

if(n>1)m--;//这里退掉是因为1号点被重复入队了两次，但当n==1时，1是仅被入队一次的。

return m;

}

double parea(point \*ch,int m)//三角形法求凸多边形面积

{

double tmp=0;

for(int i=2;i<m;i++)

{

//注意此处是逆时针有向面积，且是平行四边行，要除以2的。

tmp+=cross(ch[i]-ch[1],ch[i+1]-ch[1]);

}

return tmp/2;

}

//绕数法

int inp(const point &p,point \*poly,int n)//点与凸多边形的位置关系

{

//假想有一个水平向右的射线，求通过顺逆多少次。（对于该点的左边，我们可以把它旋转过来理解）

//I~I,~表示点的话，绕数为2。

poly[n+1]=poly[1];//注意要将第1号点复制到第n+1号点去。

int wn=0;

for(int i=1;i<=n;i++)

{

if(ons(p,poly[i],poly[i+1]))return -1;//点在多边行边上

int k=dcmp(cross(poly[i+1]-poly[i],p-poly[i]));

int d1=dcmp(poly[i].y-p.y);

int d2=dcmp(poly[i+1].y-p.y);

//注意一个细节，只能一边取等(至于是哪边，是无所谓的，因为不会一条边两个点都与它重，否则会算重。

if(k>0&&d1<=0&&d2>0)wn++;//多边形的边逆时针穿过。

if(k<0&&d2<=0&&d1>0)wn--;//多边形的边顺时针穿过。

}

if(wn!=0)return 1;

return 0;

}

double getmax(point \*p,int n)//旋转卡壳

{

int j=2;

double tmp=0;

p[0]=p[n];

p[n+1]=p[1];

//基本原理是利用了点移动的单调性

for(int i=1;i<=n;i++)//旋转卡壳基本思路是对于每一条边，都找出对应的最远的边。

{

//及一直旋转到不能再旋转距离一定减少为止，此时可以保证对于点i,求得的一定是最远点

while(cross(p[i+1]-p[i],p[(j+1)%n]-p[j])>0)j=(j+1)%n;

tmp=max(tmp,length(p[i+1]-p[j]));

tmp=max(tmp,length(p[i+1]-p[(j+1)%n]));

tmp=max(tmp,length(p[i]-p[j]));

tmp=max(tmp,length(p[i]-p[(j+1)%n]));

}

return tmp;

}

struct line

{

point p;

vec v;

double ang;

line(){}

line(point p,vec v):p(p),v(v){ang=atan2(v.y,v.x);}

bool operator <(const line &l)const{

return ang<l.ang;

}

};

point getlinei(line x,line y)//两线相交

{

vec u=x.p-y.p;

double t=cross(y.v,u)/cross(x.v,y.v);

return x.p+x.v\*t;

}

bool onleft(point p,line l)//点在线左

{

//此处等于即以为边上可行，否则边上不可行。

return dcmp(cross(l.v,p-l.p))>=0;

}

line q[maxn];

int hpi(line \*l,int n,point \*p)//半平面交

{

//按极角排序

sort(l+1,l+n+1);

int first=1,last=0;

for(int i=1;i<=n;i++)

{

//删除前端无用平面，last在前，first在后

//删除末端无用平面，通过判断点是否在线左来确定

while(first<last && !onleft(p[last-1],l[i])) last--;

while(first<last && !onleft(p[first],l[i])) first++;

if(dcmp(l[i].ang-q[last].ang)==0)//当出现平行时选一个在左边的，这里用若不用ang用叉积也行

q[last]=onleft(l[i].p,q[last])?l[i]:q[last];

else q[++last]=l[i];//否则加入该边

if(last>first)p[last-1]=getlinei(q[last],q[last-1]);//更新新的点

}

while(first<last&&!onleft(p[last-1],q[first]))last--;//最后删除前端不正确的点

if(last-first<=1)return 0;//但处理无边界问题时，我们要外加边界的，故若算出来够不成一个区域就要返回无解，故至少应该有3条线段

p[last]=getlinei(q[last],q[first]);

return last-first+1;

}

double f(double x)//求函数值

{

return 0;

}

double cut(double l,double r,double fl,double fm,double fr)//辛普森公式

{

return (r-l)/6\*(4\*fm+fl+fr);

}

//考虑有些f运算非常慢，所以尽可能的重复使用一些变量

double simpson(double l,double mid,double r,double fl,double fm,double fr,double s)//辛普森主程序

{

double mid1=(l+mid)/2,mid2=(mid+r)/2;

double fm1=f(mid1),fm2=f(mid2);

double s1=cut(l,mid,fl,fm1,fm);

double s2=cut(mid,r,fm,fm2,fr);

if(dcmp(s-s1-s2)==0)return s;

return simpson(l,mid1,mid,fl,fm1,fm,s1)+simpson(mid,mid2,r,fm,fm2,fr,s2);

}

void mcc(point \*p,int n,point &cc,double &r)//最小圆

{

random\_shuffle(p+1,p+n+1);//随机化p个点

cc=p[1];

r=0;

for(int i=2;i<=n;i++)

{

if(dcmp(length(cc-p[i])-r)>0)//当出现一个点不在当前圆中。

{

cc=p[i];

r=0;

for(int ii=1;ii<i;ii++)

{

if(dcmp(length(cc-p[ii])-r)>0)//当出现第二点不在圆中

{

cc.x=(p[i].x+p[ii].x)/2;

cc.y=(p[i].y+p[ii].y)/2;

r=length(cc-p[i]);

for(int i3=1;i3<ii;i3++)

{

if(dcmp(length(cc-p[i3])-r)>0)//当出现第三个点不在圆中

{

cc=getcc(p[i],p[ii],p[i3]);//两两中垂线算焦点

r=length(cc-p[i]);

}

}

}

}

}

}

}

int main()

{

return 0;

}

概率与期望

//推期望倒推，概率正推

博弈论

Nim游戏是经典的公平组合游戏(ICG)，对于ICG游戏我们有如下定义：

* 两名选手
* 两名选手轮流行动，每一次行动可以在有限合法操作集合中选择一个
* 游戏的任何一种可能的局面(position)，合法操作集合只取决于这个局面本身，不取决于轮到哪名选手操作、以前的任何操作、骰子的点数或者其它因素；局面的改变称为“移动”(move)
* 如果轮到某名选手移动，且这个局面的合法的移动集合为空（也就是说此时无法进行移动），则这名选手负

对于第三条，我们有更进一步的定义Position，我们将Position分为两类：

1. P-position：在当前的局面下，先手必败
2. N-position：在当前的局面下，先手必胜

它们有如下性质：

1. 合法操作集合为空的局面是P-position
2. 可以移动到P-position的局面是N-position
3. 所有移动都只能到N-position的局面是P-position

SG函数的建立

首先定义mex(minimal excludant)运算，这是施加于一个集合的运算，表示最小的不属于这个集合的非负整数。例如mex{0,1,2,4}=3、mex{2,3,5}=0、mex{}=0。

对于一个给定的有向无环图，定义关于图的每个顶点的SG函数sg如下：sg(x)=mex{ sg(y) | y是x的后继 }。也就是说，一个点的SG函数为在它所有后继中都未出现的最小的值。

SG函数的性质

来看一下SG函数的性质。首先，所有的没有出边的顶点，其SG值为0，因为它的后继集合是空集。然后对于一个sg(x)=0的顶点x，它的所有后继y都满足 sg(y)≠ 0。对于一个sg(x)≠ 0的顶点，必定存在一个后继y满足sg(y)=0。

这个时候你就应该有所发现了！SG函数的性质和N,P局面的性质非常相似！ 以上表明，顶点x所代表的postion是P-position当且仅当sg(x)=0（跟P-positioin/N-position的定义是完全对应的）。

后手必胜当且仅当sg的异或和为0

结合定义求即可。

模型即一些方法

<https://www.cnblogs.com/Randolph87/p/5804798.html>

SG函数的计算方法：

一个局面的SG为mex{后继局面的SG}

mex运算为集合中没出现的最小的自然数

几个局面的和的SG为单个的SG亦或

SG不为0时先手必胜，SG为0时后手必胜

1.Nim Game

最为经典

n堆石子，每次可以从一堆里面取任意个石子

对于一堆石子，SG函数就是石子数

整个游戏的SG函数是每一堆石子的SG函数的亦或和

必胜：SG不为0

必败：SG为0

2.Bash Game

每次最多取m个石子，其他同Nim

一堆石子的SG函数为石子数mod(m+1)

整个游戏的SG函数是每一堆石子的SG函数的亦或和

必胜：SG不为0

必败：SG为0

3.Nimk Game

每次最多可以同时从k堆石子进行操作，这k堆可以取不同数量的石子

一堆石子的SG函数为石子数

对每一个二进制位单独算，求SG函数每一个二进制位1的个数mod(k+1)，如果都为0，则必败，否则必胜

证明：

对于必败态不管怎么走都只能走到必胜态

对于变化的SG的最高位，你至少变化为1，最多变化为k，所以这一位1的个数不可能mod(k+1)还是为0

对于必胜态我们肯定可以找到一种方法走到必败态

我们从高位往低位做，记s为这一位可以随意填值的数字个数（如果把某一位从1变成0，那么更低位就能随便取值了）

假设我们现在做到第k位，记n为除了能随便取值的s位以外这一位1的个数mod(k+1)

如果n+s<=k，那么很简单，我们取出n个第k位为1的让这些数字的第k位变成0，那s个数字这一位也变成0，然后s+=n

如果n+s>k，即n+s>=k+1，那么s>=k+1-n，我们在s中间取k+1-n个变为1，其他变为0就可以满足条件了

4.Anti-Nim Game

不能取石子的一方获胜

必胜：SG不为0且至少有一堆石子数大于0，SG为0且每一堆石子数都为1

必败：其余为必败

5.Staircase Nim

阶梯博弈

每次可以从一个阶梯上拿掉任意数量石子放到下一层阶梯，不能操作的为输

SG函数为奇数阶梯上的石子的亦或和

如果移动偶数层的石子到奇数层，对手一定可以继续移动这些石子到偶数层，使得其SG不变

6.Wythoff Game

有两堆石子，每次可以从一堆或者两堆里拿走一样数目的石子，不能取的为输

必败态为(1,2)(3,5)(4,7)(6,10)...

差为1，2，3，4.....每一对数的第一个数为前面没出现的最小的正整数

7.Take & Break

每次可以把一堆石子分成两堆甚至多堆不为0的石子，不能操作的为输

暴力计算SG

8.树上删边游戏

给定根节点，每次可以删掉一条边，不与根节点相连的部分删除

叶子节点SG为0，其他节点的SG函数为子树SG+1的亦或和

证明：

将子树SG+1看做石子数（我们可以定义没有节点的图的SG为-1），然后就变成了取石子游戏

9.无向图删边

规则同树上删边游戏

结论：把奇环缩成一个点加一条新边，把偶环缩成一个点，不影响SG，然后套用树上删边游戏

10.翻硬币游戏

n枚硬币排成一排，有的正面朝上，有的反面朝上。   
游戏者根据某些约束翻硬币（如：每次只能翻一或两枚，或者每次只能翻连续的几枚），但他所翻动的硬币中，最右边的必须是从正面翻到反面。   
谁不能翻谁输。

需要先开动脑筋把游戏转化为其他的取石子游戏之类的,然后用如下定理解决:   
局面的 SG 值等于局面中每个正面朝上的棋子单一存在时的 SG 值的异或和。

证明的基本套路：

必胜局面存在一个操作到达必败局面，必败局面无论怎么操作都会到必胜局面