苏州大学ACM模板

– how to override zxjnb

Huchi

2018年 10月 12日

How to Override zxjnb:

1 Number Theory

1.1 最大公因数

找寻a,b的最大公因数, 利用辗转相除法：

```
int gcd(int a, int b) { return b?gcd(b,a%b):a; } // __gcd
```

常见公式：

1. $gcd(a^n-1, a^m-1) = a^{gcd(n,m)}-1$
2. 扩展 $a>b, gcd(a,b)=1$ 时 $gcd(a^m-b^m, a^n-b^n) = a^{gcd(m,n)}-b^{gcd(m,n)}$
3. $G = gcd(C_n^1, C_n^2, ..., C_n^{n-1})$

* n为质数时G=n
* n有多个质因子时，G=1
* n有一个质因子p，G=p

4. F_n 为斐波那契额数列, $gcd(F_n, F_m) = F_{gcd(n,m)}$
5. 给定两个互素的正整数A,B，那么他们组合的数$C = p*A+q*B > 0$, C最大为AB-A-B,不能组合的个数为$\frac{(A-1)*(B-1)}{2}$
6. $(n+1)lcm(C_n^0, C_n^1, .., C_n^n) = lcm(1,2,3,..,n+1)$
7. 对于质数p,$(x+y+...+w)^p\%p == (x^p+y^p+...+w^p)\%p$

1.2 素数测定

素数测定常用Miller-Rabin素数测试，是基于费马小定理 $a^p \equiv a(mod\ p)$ p为质数
反过来，如果满足该式子，p大多数为质数

```
#define ll long long
using namespace std;
const int Times = 10;
ll multi(ll a, ll b ,ll m) { ll ans = 0; a%=m;
    while(b) { if(b&1)ans = (ans+a)%m; a = (a+a)%m; b >>= 1; } return ans;
}
ll quick_mod(ll a, ll b, ll m) { ll ans = 1; a %= m;
    while(b) {
        if(b&1)ans = multi(ans, a, m); a = multi(a,a,m); b>>=1;
    } return ans;
}
bool Miller_Rabin(ll n) {
    if(n==2)return true;
    if((n<2) || !(n&1))return false;
    ll m = n-1; int k = 0;
    while((m&1)==0) { k++; m>>=1; }
    for(int i = 0; i < Times; i++) {
        ll a = rand()%(n-1)+1; ll x = quick_mod(a,m,n); ll y = 0;
        for(int j = 0; j < k; j++) {
            y = multi(x,x,n);
            if(y==1 && x!=1 && x!= n-1) return false;
            x = y;
        } if(y!=1) return false;
    } return true;
}
```

51Nod 质数检测：

```
import java.util.Scanner;
import java.math.*;
public class Main {
    public static void main(String[] args) {
        Scanner cin = new Scanner(System.in);
```

```
        BigInteger x; x=cin.nextBigInteger();
        if(x.isProbablePrime(1)) System.out.println("Yes"); // java has Miller_Rabin
        else System.out.println("No");
    }
}
```

1.3 扩展欧几里得

对于二元一次方程 ax+by=gcd(a, b) 利用扩展欧几里得可以求得一组可行解。

```
void e_gcd(int a, int b, int &d, int &x, int &y) {
    if(!b) { d=a,x=1,y=0; return; }
    else { e_gcd(b,a%b,d,y,x); y -= (a/b)*x; }
}
```

$a > b$时x为正数。可以令y=(y%a + a) % a 把y变成正数，再利用原始计算x

1.4 毕达哥拉斯三元组本原解

对于$x^2 + y^2 = z^2$ 我们只考虑本原解，即x,y,z互质。

这样的话，我们不妨假设x为偶数，y，z为奇数，那么就存在互质的数n,m

而且n,m一奇一偶且$m > n$, 有$x = 2 * m * n, y = m^2 - n^2, z = m^2 + n^2$

POJ-1305

题意：给一个正整数n求解，求解n的范围内gcd值为1的勾股数和不是勾股数的个数。

```
#include <bits/stdc++.h>
const int N = 1e6+9;
using namespace std;
int biao[N], n; bool flag[N];
int gcd(int a, int b) { return b?gcd(b,a%b):a; }
void init() {
    for(int i = 2; i*i < N; i++) {
        for(int j = 1; j < i; j++) {
            if(gcd(i,j)!=1) continue; if((i&1)&&(j&1)) continue;
            if(i*i+j*j<N) biao[i*i+j*j]++; // i^2+j^2 = c^2
        }
    } for(int i = 1; i < N; i++) biao[i]+=biao[i-1];
}

int main(){
    init(); while(~scanf("%d",&n)) {
        memset(flag,0,sizeof flag); int ans = 0;
        for(int i = 2; i*i < N; i++) {
            for(int j = 1; j < i; j++) {
                if(gcd(i,j)!=1) continue; if((i&1)&&(j&1)) continue;
                for(int k = 1; k*(i*i+j*j)<=n; k++) // c^2
                    flag[2*i*j*k]=flag[k*(i*i-j*j)]=flag[k*(i*i+j*j)]=1; // used
            }
        } for(int i = 1; i <= n; i++) if(flag[i])ans++;
        printf("%d %d\n",biao[n],n-ans);
    } return 0;
}
```

1.5 欧拉函数与欧拉定理

我们定义欧拉函数$\phi(n)$= 小于n且与n互质的数， $\phi(1) = 1, \phi(2) = 1$

1. m与n互素， $\phi(mn)=\phi(n)*\phi(m)$
2. $n=p_1^{a_1}p_2^{a_2}p_3^{a_3}...p_n^{a_n}, \phi(n)=n*(1-\frac{1}{p_1})*(1-\frac{1}{p_2})*(1-\frac{1}{p_3})*...*(1-\frac{1}{p_n})$
3. $\sum_{d|n}\phi(d)=n$ 即n的因子的欧拉函数之和为n

容斥计算欧拉函数，复杂度$O(\sqrt{n})$：(利用质数可以加快时间)

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

1.6 欧拉筛

欧拉筛是可以做到O(n) 预处理1-n之间所有的质因数，通过对其修改，我们也能预处理出欧拉函数和莫比乌斯函数以及因数个数。一般欧拉函数不需要修改，可以作为模板.

```
const int N = 5e4+5; // d:number of yinzi
int mu[N],prime[N],d[N],ti[N],phi[N], cnt; bool vis[N];
void init() {
    mu[1] = d[1] = phi[1] = 1;
    for(int i = 2 ; i < N; i ++) {
        if(!vis[i]) { prime[cnt++] = i; mu[i] = -1; d[i] = 2; ti[i] = 1; phi[i] = i-1; }
        for(int j = 0; j < cnt; j++) if(i*prime[j] < N) {
            vis[i*prime[j]] = 1;
            if(i%prime[j]==0) {
                ti[i*prime[j]] = ti[i] + 1;
                d[i*prime[j]] = d[i]/(ti[i]+1)*(ti[i]+2);
                phi[i*prime[j]]=phi[i]*prime[j]; break;
            } mu[i*prime[j]] = -mu[i];
            d[i*prime[j]] = d[i]*2; ti[i*prime[j]] = 1;
            phi[i*prime[j]] = phi[i]*(prime[j]-1);
        }
    }
}
```

欧拉筛例题

题意: 给出一个区间[L,U]，求给定区间内的质数距离最小的一对和质数距离最大的一对。$U \le 1e9, U-L \le 1e6$

方法: 预处理$\sqrt{1e9}$ 的质数, 然后对[L, U] 区间内去晒(即标记)

```
#define uint unsigned int
const int N = 5e4;
int prime[N], cnt; bool judge[21*N], vis[N];
using namespace std;

void init(){
    for(int i = 2; i<N; i++){ if(!vis[i]) prime[cnt++] = i;
        for (int j = 0; j < cnt; j++) if(i*prime[j] < N) { vis[i*prime[j]] = 1; if(i%prime[j]==0)
            break; }
    }
}

int main(){ init(); uint u,v;
    while(~scanf("%d%d",&u,&v)) { if(u==1) u=2; // 1不是质数
        memset(judge,0,sizeof judge);
        for(int i = 0; i < cnt && prime[i] <= v; i++){
            uint j=((u-1)/prime[i]+1)*prime[i]; // 第一个>=u的prime[i]的倍数
            if(j == prime[i]) j += prime[i]; // j为质数跳过
```

```
            for(;j <= v; j+=prime[i]) judge[j-u]=1; // 晒去prime[i]的倍数
        } int st,stlen=N,ed,edlen=0,cot=0,len=0;
        for(int i = 0; i <= v-u; i++) {
            if(!judge[i]) { // 即u+i为质数
                if(len && len<stlen) {
                    st = i-len; stlen = len; // 更新最近的质数对
                } if(len && len>=edlen) {
                    ed = i-len; edlen = len; // 更新最远的质数对
                } cot++; len = 0; // 统计区间质数个数
            } len ++ ;
        } if(cot>1)printf("%d,%d are closest, %d,%d are most distant.\n",u+st,u+st+stlen,u+ed,u+ed+
            edlen);
        else puts("There are no adjacent primes.");
    } return 0;
}
```

直角三角形个数

题意: 求$a,b,c <= L$的三角形个数且满足a¡b¡c.abc互质.$L < 1e12$

利用三元组本原解枚举n,m实现sqrt(1e12), 容斥去重

```
#include <bits/stdc++.h>
#define ll long long
const int N = 1e6+9;
using namespace std;
int prime[N/10], phi[N], check[40], cnt, num; ll ans, L; bool vis[N];

void init(){
    for (int i = 2; i < N; i++) {
        if(!vis[i]) { prime[cnt++]=i;phi[i]=i-1; }
        for (int j = 0; j < cnt; j++) {
            if(i*prime[j] >= N) break; vis[i*prime[j]] = 1;
            if(i%prime[j]==0) { phi[i*prime[j]]=phi[i]*prime[j];break;}
            phi[i*prime[j]]=phi[i]*(prime[j]-1);
        }
    }
}

void prime_check(int n){ // 质因数分解
    num = 0;
    if(!vis[n]){ check[num++]=n; return ; } //n 为质数直接返回
    for(int i = 0; i < cnt && prime[i]*prime[i] <= n; i++) {
        if(n%prime[i]==0){ check[num++]=prime[i]; while(n%prime[i]==0) n/=prime[i]; }
    } if(n>1) check[num++]=n;
}

//容斥原理计算无关的数fun(j)=j-(j/c1+j/c2+j/c3)+(j/(c1*c2)+j/(c1*c3)+j/(c2*c3))-(j/(c1*c2*c3))。
void dfs(int k, int r, int s, int n){
    if(k==num){
        if(r&1) ans -= n/s;
        else ans+= n/s;
        return;
    } dfs(k+1, r, s, n);
    dfs(k+1, r+1, s*check[k], n);
}

int main(){
```

```
    init(); int t;scanf("%d",&t);
    while(t--){ ans = 0; scanf("%lld",&L);
        int m = (int)sqrt(1.0*L + 0.5);
        for(int i = m; i > 0; i--) { // 枚举m ,  n<m 只要计算有多少符合n就行
            int p = (int)sqrt(L - (ll)i*i + 0.5); // n的上界
            if ( i<=p ) {
                if(i&1){
                    prime_check(i); dfs(0,0,1,i>>1);
                } else ans += phi[i]; // 对于偶数，与其互质的数都为奇数可以直接
            } else {
                // i>p时，j所取的范围在[1,p]
                prime_check(i);
                if(i&1)dfs(0,0,1,p>>1);
                else dfs(0,0,1,p);
            }
        } printf("%lld\n",ans);
    }
}
```

1.7 逆元

对于$am \equiv 1(mod\ m)$, 这个同余方程中最小的x正整数解叫做a模m的逆元; 也就是说$\frac{1}{a} \equiv x(mod\ m)$
由费马小定理得a对于模数mod为质数的逆元a^{mod-2}可以用快速幂log求得

```
ll qp(ll a, int k) { ll ans = 1; a %= mod;
    while(k) {
        if(k & 1) ans = ans * a % mod;
        a = a*a % mod; k >> = 1;
    } return ans;
}
ll inv(ll a) { return qp(a%mod, mod-2); }
```

预处理逆元:
我们可预处理1-n的逆元是O(n)的复杂度

```
inv[1]=1;
for (int i=2;i<=n;++i) inv[i]=(mod-mod/i)*inv[mod%i]%mod;
```

预处理n! 的逆元inv

```
ll fac[N], inv[N];
void init() {
    fac[0] = inv[0] = 1;
    for (int i = 1; i < N; i++) fac[i]=fac[i-1]*i%mod;
    inv[N-1] = qp(fac[N-1], mod-2);
    for (int i = N-2; i; i--) inv[i]=(i+1)*inv[i+1]%mod;
}
```

1.8 等比数列求和

$S_n = (a + a^2 + .. + a^n)modM$
如果使用高中的公式那么就是a = 1 时$s_n = n \mod M$, 如果a!=1, $S_n = \frac{a-a^{n+1}}{1-a}$
这里还有一种方法: 二分
对于n%2 == 0 $S_n = (1 + a^{\frac{n}{2}})S_{\frac{n}{2}}$,否则 $S_n = (1 + a^{\frac{n+1}{2}})S_{\frac{n+1}{2}} + a^{\frac{n+1}{2}}$
POJ - 3233 等比矩阵求和
如果利用上面的递推式子就可以解决矩阵的i次求和问题.

```cpp
using namespace std;
int n,k;
int mod = 1e9+7;
struct Matrix {
    int mp[35][35];
    Matrix() { memset(mp,0,sizeof mp); }
    Matrix(int v){ memset(mp,0,sizeof mp); for(int i = 1; i <= n; i++){ mp[i][i] = v; } }
}a;
inline Matrix multi(Matrix a, Matrix b){
    Matrix c;
    for(int i = 1; i <= n; i++){
        for(int k = 1; k <= n; k++) if(a.mp[i][k])
            for(int j = 1; j <= n; j++) if(b.mp[k][j])
                c.mp[i][j] = (c.mp[i][j] + a.mp[i][k]*b.mp[k][j])%mod;
        }
    } return c;
}
inline Matrix pls(Matrix a, Matrix b){
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= n; j++) a.mp[i][j] = (a.mp[i][j]+b.mp[i][j])%mod;
    } return a;
}
inline Matrix powmul(Matrix a, int k){
    Matrix c = Matrix(1);
    while(k){
        if(k&1)c = multi(c,a);
        a = multi(a,a); k>>=1;
    } return c;
}
inline Matrix S(Matrix a, int k){
    if(k==1) return a;
    Matrix temp = Matrix(1);
    if(k&1){
        Matrix temp2 = powmul(a,(k+1)/2);
        return pls(multi(pls(temp, temp2), S(a,(k-1)/2)), temp2);
    } else {
        Matrix temp2 = powmul(a,k/2);
        return multi(pls(temp, temp2), S(a,k/2));
    }
}

int main(){
    while(~scanf("%d%d%d",&n,&k,&mod)) {
        for(int i = 1; i <= n; i++) {
            for(int j = 1; j <= n; j++)
                scanf("%d",&a.mp[i][j]);
        } a = S(a,k);
        for(int i = 1 ; i <= n; i++) {
            for(int j = 1; j < n; j++) printf("%d ",a.mp[i][j]);
            printf("%d\n", a.mp[i][n]);
        }
    } return 0;
}
```

这个复杂度时有点高的,还可以构造矩阵利用矩阵快速幂一步到位,但是模数不为质数就得用这个了

我们要求的矩阵和为S_n

那么

$$\begin{bmatrix} S_n \\ A^n \end{bmatrix} = \begin{bmatrix} 1 & A \\ 0 & A \end{bmatrix} \begin{bmatrix} S_{n-1} \\ A^{n-1} \end{bmatrix}$$

1.9 组合数计算

从n个物品中选取k个物品有C_n^k 种方法。

组合数表预处理 mod任意:

```
void init(){
    c[0][0] = 1;
    for (int i = 1; i < N; i++) {
        c[i][0] = c[i][i] = 1;
        for (int j = 1; j < i; j++) c[i][j] = (c[i-1][j] + c[i-1][j-1]) % mod;
    }
}
```

预处理阶乘， mod只能为质数

```
void init() {
    fac[0] = inv[0] = 1;
    for (int i = 1; i < N; i++) fac[i]=fac[i-1]*i%mod;
    inv[N-1] = qp(fac[N-1], mod-2);
    for (int i = N-2; i; i--) inv[i]=(i+1)*inv[i+1]%mod;
}
ll C(int n, int m) { return fac[n]*inv[m]%mod*inv[n-m]%mod; }
```

Lucas 模数必须为小质数

```
int Lucas(int n, int m){
    if(m==0)return 1;
    return C(n%p,m%p)*Lucas(n/p,m/p)%p;
}
```

1.10 中国剩余定理CRT

有n组同余方程

$$x \equiv a_1(mod\ m_1)x \equiv a_2(mod\ m_2)x \equiv a_3(mod\ m_3)...x \equiv a_n(mod\ m_n)$$

利用CRT既可以解模数互质的CRT

```
void e_gcd(int a, int b, int &d, int &x, int &y) {
    if(!b){ d = a; x = 1; y = 0; return ; }
    e_gcd(b, a%b, d, y, x); y -= x*(a/b);
}
//中国剩余定理
int CRT(int a[], int m[], int n){
    int M = 1; int ans = 0;
    for(int i = 0; i < n; i++) M *= m[i];
    for(int i = 0; i < n; i++) {
        int x,y,d; int Mi = M/m[i];
        e_gcd(Mi, m[i], d, x, y);
        ans = (ans + Mi*x*a[i])%M;
    } if(ans<0)ans+= M;
    return ans;
}
```

对于普通中国剩余定理要求的$m_1, m_2, \&, m_k$互素.但如果发生不互素时,需要采用两两合并.

1.11 反素数

f(n)为n的因数个数，对于$0 < i < n, f(i) < f(n)$, 则称n为反素数。

反素数一般满足质因数的幂次随着质因数的大小增大而减小 ,即若$n = 2^{t_1}3^{t_2}$, 则必有$t_1 \geq t_2$

求最小的数使其因数为n,CF27E

$n <= 1000$，我们贪心的认为其质因数不会超过前50个质数，而且幂次不会超过63。

```
void dfs(int depth, ll tmp, int num) { //depth 为第几个质数, tmp为当前的值, um为因数个数
    if(num > n || depth>=cnt) return ;
    if(num == n && ans > tmp) { ans = tmp; return; // 满足的情况下更新 }
    for(int i= 1; i <= 63; i++){ // 枚举幂次, 事实上可以记录上一次的幂次来优化剪枝
        if(ans/prime[depth]<tmp || num*(i+1)>n) return; // 用除法防止溢出
        dfs(depth+1, tmp*=prime[depth], num*(i+1));
    }
}
```

求1-n因数个数最大的数 URAL 1748

```
void dfs(int depth, ll ansa, int ansb, int limit){ //ansa为值, ansb为因数个数
    if(depth >= tot || ansa > n)return;
    if(ansb > b|| ((ansb == b) && a>ansa)){
        a = ansa; b = ansb;
    } for(ll i = 1; depth < tot && i <= limit; i++){
        if(ansa > n/prime[depth])break;
        dfs(depth+1, ansa*=prime[depth], ansb*(i+1), i);
    } return ;
}
```

HDU4542

题意： X mod ai = bi 对于ai在区间 [1, X] 范围内每个值取一次时，有K个ai使bi等于0 。告诉你有K个ai使得(不使得)bi等于0, 找最小的X

解析：K个相等的时候我们去dfs暴力枚举，K个不相等的话这个数一定不大，先预处理

```
void init() {
    for(int i = 1; i < N; i++) d[i] = i;
    for(int i = 1; i < N; i++) {
        for(int j = i; j <N; j+=i) d[j]--; // 容斥减去因子个数
        if(!d[d[i]]) d[d[i]] = i; // 更新因子个数为d[i]位置的数为i, 节约了空间,这样也保是最小的
        d[i] = 0
    }
}
void dfs(int depth, ll ansa, int ansb, int limit){
    if(ansb > k || depth > 16) return ;
    if(ansb == k && ansa < ans) { ans = ansa; return ; }
    for(int i = 1; i <= limit; i++) {
        if(ansa > pos/prime[depth] || ansb*(i+1)>k) break;
        ansa*=prime[depth];
        if(k%(ansb*(i+1))==0) dfs(depth+1, ansa, ansb*(i+1), i);
    }
}
int main(){
    init(); int T,t=0; scanf("%d",&T);
    while(t++<T) {
        scanf("%d%d",&type,&k); ans = flag = 0;
        if(type) {
            if(d[k]) printf("Case %d: %d\n",t,d[k]);
            else printf("Case %d: Illegal\n",t);
        } else {
```

```
            ans = ~0ull; dfs(0,1,1,63);
            if(ans > inf) printf("Case %d: INF\n",t);
            else printf("Case %d: %llu\n",t,ans);
        }
    } return 0;
}
```

1.12 指数循环节

$$a^b \equiv a^{b\%\phi(c)+\phi(c)}(mod\ c), b > \phi(c)$$

HDU4335

这里求$n^{n!} \equiv b(mod\ p)$有多少个成立的n，对于指数%phi(p)为0后的数，是存在循环节的。只要记录一下一次循环的个数p，最后多余的部分去跑一下记忆化的就知道有多少组了。

```
#include <bits/stdc++.h>
#define ll unsigned long long
const int maxn = 1e5+5;
using namespace std;

ll phi(ll n) { ll res = n;
    for(ll i = 2; i <= sqrt(n); i++) {
        if(n%i==0) { res = res - res/i; while(n%i==0) n/=i; }
    } if(n > 1) res = res - res/n; return res;
}

ll quickmod(ll a, ll b, ll c){ ll ans = 1;
    while(b) {
        if(b&1) ans = ans*a%c;
        a = a*a%c; b >>= 1;
    } return ans;
}

ll b, p, m, fa[maxn];
int main(){
    int t=0,T;scanf("%d",&T);
    while(t++<T){
        scanf("%I64u%I64u%I64u", &b, &p, &m); printf("Case #%d: ",t);
        if(p==1){
            if(m==18446744073709551615ull) printf("18446744073709551616\n"); // 防止溢出
            else printf("%I64u\n",m+1); continue;
        }else{
            ll i = 0, ans = 0, fac = 1; ll phic = phi(p);
            for(i = 0; i <= m && fac<=phic; i++) {
                if(quickmod(i, fac, p) == b) ans ++; // 计算n!  小于 phi(p)的个数
                fac *= i+1;
            } fac = fac%phic; //指数循环节公式
            for(;i <= m && fac; i++){
                if(quickmod(i, fac + phic, p) == b) ans++;
                fac = (fac*(i+1))%phic;
            } // 至多跑一个循环或者不超过M
            if(i <= m){
                ll cnt = 0;
                for(int j = 0; j < p; j++){
                    fa[j] = quickmod(i+j, phic, p); // 记忆化p长度的循环
```

```
                if(fa[j] == b) cnt ++;
            }
            ll idx = (m-i+1)/p; // 循环个数
            ans += cnt * idx; // 循环的种数
            ll remain = (m-i+1)%p; // 不足一个循环的数
            for(int j = 0; j < remain; j++) if(fa[j] == b) ans ++ ;
        } printf("%I64u\n",ans);
    }
} return 0;
}
```

牛客练习赛 22E

题意：给一个长为n的序列，m次操作，每次操作：

1.区间[l, r]加x

2.对于区间[l,r]，查询$a[l]^{a[l+1]^{a[l+2]\cdots}} mod p$，一直到$a_r$, 请注意每次的模数不同。

区间加值我们可以考虑使用线段树lazy标记，查询就直接使用指数循环节公式倒过来计算就行了。

这道题因为数据太多，用文件读入才能过

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const int N = 5e5+9;
const int M = 2e7+9;
ll val[N]; bool judge[M];
int prime[M],p[M],mod[N],tot;
using namespace fastIO;

inline void init(){ p[1] = 1;
    for(int i = 2; i < M; i++) {
        if(!judge[i]) { prime[tot++]=i; p[i] = i-1; }
        for(int j = 0; j < tot; j++) {
            if(i*prime[j]>=M)break; judge[i*prime[j]] = 1;
            if(i%prime[j] == 0) { p[i*prime[j]] = p[i]*prime[j]; break;}
            p[i*prime[j]]=p[i]*(prime[j]-1);
        }
    }
}
inline int phi(int n) { return p[n]; }
inline int quickmod(ll a, int b, int c) {
    ll ans = 1; a %= c;
    while(b) {
        if(b&1) ans = ans*a%c;
        a = a*a%c; b >>= 1;
    } return ans;
}

inline int slove(ll a, int b, int c) { // 指数循环节公式
    if(b*log(a) >= log(c)) return quickmod(a, b, c)+c;
    else return quickmod(a, b, c);
}
int n,m,op;

// 线段树模板
namespace sgt {
    long long chg[1000010];
    int son[1000010][2],cnt,root;
```

```
    void build(int &x,int l,int r) {
        x=++cnt;
        if(l==r) return ;
        int mid=(l+r)>>1;
        build(son[x][0],l,mid);
        build(son[x][1],mid+1,r);
    } void update(int a,int b,int k,int l,int r,long long v) {
        if(a>b || l>r)return;
        if(a<=l && b>=r)chg[k]+=v;
        else {
            int mid=(l+r)>>1;
            if(b<=mid)update(a,b,son[k][0],l,mid,v);
            else if(a>mid)update(a,b,son[k][1],mid+1,r,v);
            else update(a,mid,son[k][0],l,mid,v),update(mid+1,b,son[k][1],mid+1,r,v);
        }
    } long long query(int a,int k,int l,int r,long long v) {
        if(l==r)return v+chg[k]+val[a];
        int mid=(l+r)>>1;
        if(a<=mid)return query(a,son[k][0],l,mid,v+chg[k]);
        else return query(a,son[k][1],mid+1,r,v+chg[k]);
    }
}

int main(){
    init(); read(n); read(m);
    for (int i = 1; i <= n; i++) read(val[i]);
    sgt::build(sgt::root,1,n);
    for (int t = 0; t < m; t++) {
        read(op); ll x; int l, r;
        if(op==1){
            read(l); read(r); read(x);
            sgt::update(l, r, sgt::root, 1, n, x); // 区间加值
        } else {
            read(l); read(r); read(x); int ans = 1; mod[l] = x;
            for(int i = l+1; i <= r; i++){
                mod[i] = phi(mod[i-1]); // 预处理幂次的幂次的模数
                if(mod[i] <= 2) r = i; // 某幂次<2那么基本就是0了可以停止了
            } for(int i = r; i >= l; i--){
                ans = slove(sgt::query(i, sgt::root, 1, n, 0), ans, mod[i]); //单点查询，倒着计算
            } printf("%d\n",ans%x);
        }
    } return 0;
}
```

1.13 积性函数

积性函数的定义

1. 若f(n)的定义域为正整数域，值域为复数，即$f:Z^+-Cf:Z^+-C$，则称f(n)为**数论函数**。
2. 若f(n)为数论函数，且f(1)=1，对于互质的正整数p,q有f(pq)=f(p)f(q)，则称其为**积性函数**。
3. 若f(n)为积性函数，且对于任意正整数p,q都有f(pq)=f(p)f(q)，则称其为**完全积性函数**。

常见的积性函数

1. 除数函数$\sigma_k(n)=\sum_{d|n}d^k$, 表示n的约数的k次幂和
2. 约数个数函数$d(n)=\sigma_0(n)=\sum_{d|n}1$, 表示约数个数
3. 约数和函数$\sum_{d|n}d$

4. 欧拉函数$\phi(n) = \sum_{i=1}^{n}[(n,i) == 1]$, 对于n¿=2 有 $\phi(n) = \sum_{i=1}^{n}[(n,i) == 1]\Delta i = n\phi(n)/2$
5. 莫比乌斯函数$u(n)$, 与恒等函数互为逆元
6. 元函数 $e(n) = [n == 1]$
7. 恒等函数$I(n) = 1$
8. 单位函数$id(n) = n$
9. 幂函数$id^k(n) = n^k$

两个常用的公式

1. $e(n) = \sum_{d|n} u(d)$, 可以将u(d)看作是容斥的系数
2. $n = \sum_{d|n} \phi(d)$

积性函数性质

若f(n)为积性函数，对于正整数$n = p_1^{k_1}p_2^{k_2}...p_n^{k_n}$
有$f(n) = \prod_{i=1}^{t} f(p_i^{k_i})$, 对于完全积性函数有$f(n) = \prod_{i=1}^{t} f(p_i)^{k_i}$

1.13.1 分块技巧

题意：有一份包含个 bug 的n($1 \le n \le 10^6$)行代码，运一次到崩溃需要的时间为 r($\le 1r \le 10^9$)。你可以任意行添加 printf 语句来输出调试,即你知道是否在执行 printf 语句前就崩溃了。每设置一个 printf语句需要花费 p($1 \le p \le 10^9$) 问最坏的情况下，最少需要多少时间可以定位bug的行数

解释：我们设置f(n)为n行代码debug需要的最少时间。那么对于f(n)我们时可以由前面的状态转移过来的，我们对于n行代码分成2块，那么我们就需要f ((n+1)/2) + r + p的时间，我们分成k块，那么最大的块的大小就为 (n+k-1)/k 向下取整个，所以总体时间就是 f((n+k-1)/k) + r + (k-1)*p;

这样我们从1-n递推过来就是O(n^2) 的算法，我们要考虑优化，随着k增大，我们会发现有很多(n+k-1)/k 是相同，但是我们应该取相同种k最小的，所以我们反过来枚举(n+k-1)/k的大小，也就是每个块最大为l，那么最多有(n+l-1)/l 块，所以这样总体时间就是f(l) + r +((n+l-1)/l)* p;

对于n $\sqrt{n}$的算法还是很吃力的，我们考虑使用记忆化搜索来降低复杂度。

```
#include <bits/stdc++.h>
using ll = long long;
using namespace std;
const int N = 1e6+9;
int n, r, p, vis[N], t;
ll f[N];
ll dfs(int n){
    if(vis[n] == t || n == 1) return f[n]; // 如果已经计算过直接返回
    ll ans = 1e18+7; vis[n] = t; // 表示t组样例的时候f[n]被跟新了
    for(int j = 1; j <= sqrt(n+0.5); j++) {
        ans = min(ans, dfs((n+j)/(j+1)) + r + 1LL*j*p); // 枚举j+1块
        ans = min(ans, dfs(j)+r+((n+j-1)/j-1LL)*p); // 枚举每块最大为j个
    } return f[n] = ans; // 记忆化
}

int main(){
    while(~scanf("%d%d%d", &n, &r, &p)) {
        t++; printf("%lld\n", dfs(n)); // 每组，记忆话搜索一下
    } return 0;
}
```

1.14 迪利克雷卷积

设f,g为两个数论函数，则满足$h(n) = \sum_{d|n} f(d)g(\frac{n}{d})$ 称为f与g的迪利克雷卷积。

积性函数例题

题意：f(n)=为[0, n)种选2个数a,b且ab不为n的倍数的方案数。求$g(n) = \sum_{d|n} f(d)\%2^{64}$,$1 \le T \le 2e4, 1 \le n \le 1e9$

解析：我们先考虑f(n) ， f(n) = n*n - ab为n的倍数的方案数,即$f(n) = n^2 - \sum_{d|n} \phi(\frac{n}{d})d$

令$h(n) = \sum_{d|n} \phi(\frac{n}{d})d$

那么原式=$\sum_{d|n} d^2 - \sum_{d|n} h(d)$, 很显然这两个函数都是积性函数，我们如果直接对其质因数分解是可以做的。不过处理后面的还是很麻烦。

$\sum_{d|n}\sum_{w|d}\phi(w)*\frac{d}{w}=\sum_{w|n}w\Delta\sum_{\frac{d}{w}|\frac{n}{w}}\phi(\frac{d}{w})=\sum_{w|n}n=n\Delta d(n)$
这样处理就方便了很多

```cpp
#include <bits/stdc++.h>
const int N = 1e5+8;
using namespace std;
using ll = unsigned long long;
using lll = unsigned __int128;
ll a[N], num[N], cnt, prime[N], tot, n;
bool vis[N];
void init(){
    for(int i = 2; i < N; i++) {
        if(!vis[i]) prime[tot++] = i;
        for (int j = 0; j < tot; j++) {
            if(i*prime[j]>=N) break;
            vis[i*prime[j]] = 1;
            if(i%prime[j]==0) break;
        }
    }
}
void d(int n){
    #define j prime[i]
    for(int i = 0; 1LL*j*j <= n && i < tot; i++){
        if(n%j==0) {
            num[cnt] = 0; a[cnt] = j;
            while(n%j==0) { num[cnt]++; n /= j; } cnt++;
        }
    } if(n>1) a[cnt]=n,num[cnt++]=1;
    #undef j
}
lll qp(lll a, int k){
    lll ans = 1;
    while(k) {
        if(k&1) ans *= a;
        a *= a; k >>= 1;
    } return ans;
}
ll p(ll n){
    if(n == 1) return 1; ll ans = 1;
    for (int i = 0; i < cnt; i++) ans *= (qp(a[i], num[i]*2+2)-1)/(a[i]*a[i]-1);
    return ans;
}
ll q(ll n) {
    ll ans = n;
    for (int i = 0; i < cnt; i++) ans *= num[i] + 1;
    return ans;
}
int main(){
    int t;scanf("%d", &t); init();
    while(t--) {
        scanf("%lld", &n); cnt = 0; d(n);
        printf("%llu\n", p(n) - q(n));
    } return 0;
}
```

积性函数习题 **757E**

题意: $f_0(n)$为 (p, q) == 1, 且p*q = n的个数, $f_{r+1}(n) = \sum_{u*v=n} \frac{f_r(u)+f_r(v)}{2}$,求$f_r(n)\%1e9+7$

解释：显然$f_0(n) = 2^{d(n)}$，d(n)为质因数个数，这是积性函数，我们可以对其化简$f_0(n) = f_0(p_1^{k_1})f_0(p_2^{k_2})...f_0(p_t^{k_t}) = \prod_{i=1}^{t}(1+k_i)$

$f(p^k)$只与k有关，而k又是logn的，由$f_r(n)$定义可知这是一个积性函数，所以做r次前缀和就可以了。

```cpp
#include <bits/stdc++.h>

using namespace std;
using ll = long long;
const int N = 1e6+9;
ll f[N][21];
const int mod = 1e9+7;
int prime[N], cnt; bool vis[N];
int q, r, n;

void init(){
    for (int i = 2; i < N; i++) {
        if(!vis[i]) prime[cnt++] = i;
        for (int j = 0; j < cnt; j++) {
            if(i*prime[j] >= N) break; vis[i*prime[j]] = 1;
            if(i%prime[j]==0) break;
        }
    } f[0][0] = 1;
    for(int i = 1; i < 21; i++) f[0][i] = 2;
    for(int i = 1; i < N; i++) {
        f[i][0] = 1;
        for (int j = 1; j < 21; j++) f[i][j] = (f[i][j-1]+f[i-1][j])%mod; //前缀和
    }
}

ll d(int n){
    #define j prime[i]
    ll ans = 1;
    for (int i = 0; i < cnt && j*j <= n; i++) {
        if(n%j==0) {
            int t = 0;
            while(n%j==0)n/=j,t++; // 算$p^k$
            ans = (ans * f[r][t])%mod; // 计算每个质因数的贡献
        }
    } if(n>1) ans = (ans * f[r][1]) % mod;
    return ans;
    #undef j
}

int main() {
    init(); scanf("%d", &q);
    while(q--) {
        scanf("%d%d", &r, &n);
        printf("%lld\n", d(n));
    } return 0;
}
```

1.15 莫比乌斯反演

$$f(n) = \sum_{d|n} g(d) <=> g(n) = \sum_{d|n} u(d) f(\frac{n}{d})$$

这就是莫比乌斯函数，对于已知的f(n)，我们都可以利用该等式来反过来求g(n);

求互质的数对数

题意：求$1 \le i \le b, 1 \le j \le d$ 由多少对(i,j)满足gcd(i, j) = k

解析：我们先解决$1 \le i \le n, 1 \le j \le m.gcd(i, j) == 1$的问题

令F(d) 为有多少对i，j满足gcd(i, j) == d的倍数

f(d) 为有多少对gcd(i, j) == d，那么我们就有$F(d) = \sum f(id)$, $f(1) = \mu(1)F(1) + \mu(2)F(2) + ...$

显然$f(1) = \sum_{i=1}^{min(n,m)} \mu(i)F(i) = \sum_{i=1}^{min(n,m)} \mu(i)\frac{n}{i}\frac{m}{i}$,

利用分块就可以做到$\sqrt{n} + \sqrt{m}$ 查询

所以对于原问题，b,d/=k就转成了gcd(i,j)==1, 再去减去不符合的就可以了

```
#include <bits/stdc++.h>
#define ll long long
const int N = 1e5+10;
using namespace std;
bool vis[N];
int mu[N], prime[N], cnt;
inline void init(){
    mu[1] = 1;
    for(int i = 2; i < N; i++){
        if(!vis[i]){ prime[cnt++] = i; mu[i] = -1; }
        for(int j = 0; j < cnt; j++){
            if(i*prime[j]>N)break;
            vis[i*prime[j]]= 1;
            if(i%prime[j] == 0){ mu[i*prime[j]] = 0; break; }
            mu[i*prime[j]] = -mu[i];
        }
    } for (int i = 2; i < N; i++) mu[i] += mu[i-1];
}
inline ll slove(int l, int r){
    ll ans = 0;
    for(int i = 1, tmp; i <= l; i=tmp+1) {
        tmp = min(l/(l/i), r/(r/i));
        ans += (ll)(mu[tmp]-mu[i-1])*(l/i)*(r/i);
    } return ans;
}
int main(){
    int b,d,k,T=0, t; scanf("%d",&t); init();
    while(t--) {
        scanf("%*d%d%*d%d%d",&b,&d,&k);
        if(!k) {
            printf("Case %d: 0\n",++T); continue;
        } b/=k;d/=k; if(b > d) swap(b, d);
        printf("Case %d: %lld\n",++T,slove(b,d) - slove(b, b)/2);
    } return 0;
}
```

GCD表中的质数

题意：有一个M * N的表格，行与列分别是1 - M和1 - N，格子中间写着行与列的最大公约数Gcd(i, j)($1 \le i \le M, 1 \le j \le N$)。求质数个数。

解析：我们枚举p,利用上面的结论就有$ans = \sum_p \sum_{i=1}^{\frac{n}{p}} \lfloor \frac{n}{ip} \rfloor \lfloor \frac{m}{ip} \rfloor$，根据这个公式的复杂度$O(T(\frac{n}{log\ n}\Delta(\sqrt{n} + \sqrt{m})))$，这样复杂度有点高，我们对原始做一些优化，交换两个求和符号就可以得到$\sum_{i=1}^{n} \lfloor \frac{n}{i} \rfloor \lfloor \frac{m}{i} \rfloor \sum_{j|i} \mu(\frac{i}{j})$&& j为质数

如果我们预处后面的一个求和，那么就可以做到分块查询

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 5e6+9;
using ll = long long;
int prime[N/5], mu[N], cnt, n, m;
ll f[N]; // 为后一个求和
bool vis[N];

inline void init(){
    mu[1] = 1;
    for (int i = 2; i < N; i++) {
        if(!vis[i]) {prime[cnt++] = i; mu[i]=-1; f[i]=1;} // 对于质数f[i] = mu[1] = 1;
        for (int j = 0; j < cnt; j++) {
            if(i*prime[j]>=N) break;
            vis[i*prime[j]] = 1;
            if(i%prime[j]==0) { f[i*prime[j]] = mu[i]; break; } //prime[j]次数为2时，之前的u都变成了0，只剩下p=prime[j],所以f[i*prime[j]]=mu[i];
            f[i*prime[j]] = -f[i] + mu[i]; mu[i*prime[j]] = -mu[i]; //次数为1时，u变成原来的相反数，再多上新增的。
        }
    } for (int i = 2; i < N; i++) f[i] += f[i-1]; // 前缀和
}

inline ll solve(int n, int m) { // 分块查询
    ll ans = 0;
    for (int i = 1, j; i <= n; i=j+1) {
        j = min(n/(n/i), m/(m/i));
        ans += (f[j]-f[i-1])*(n/i)*(m/i);
    } return ans;
}

int main(){
    int T; scanf("%d", &T); init();
    while(T--){
        scanf("%d%d", &n, &m); if(n > m) swap(n, m);
        printf("%lld\n", solve(n, m));
    }
}
```

诸如此类的还有

1. $\sum_{a=1}^{N}\sum_{b=1}^{M} gcd(a,b) = \sum_{d=1}^{min(N,M)} \phi(d)\lfloor\frac{N}{d}\rfloor\lfloor\frac{M}{d}\rfloor$
2. $\sum_{a=1}^{N}\sum_{b=1}^{M} lcm(a,b) = \frac{1}{4}\sum_{d=1}^{min(N,M)}\lfloor\frac{N}{d}\rfloor\lfloor\frac{M}{d}\rfloor(\lfloor\frac{N}{d}\rfloor+1)(\lfloor\frac{M}{d}\rfloor+1)d\sum_{d'|d} d'u(d')$
3. $\sum_{a=1}^{N}\sum_{b=1}^{N} lcm(a,b) = \sum_{i=1}^{N}(-i+2\sum_{j=1}^{i} lcm(i,j))$, 预处理后可以O(1)输出
4. $\sum_{i=1}^{n} gcd(i,n) = \sum_{d|n}\phi(d)\frac{n}{d}$
5. $\sum_{i=1}^{n} e(gcd(i,n)) = \sum_{d|n}\mu(d)\frac{n}{d}$
6. 约束和之和$\sum_{i=1}^{n} = \sum_{i=1}^{n} i\lfloor\frac{n}{i}\rfloor$
7. $g(n) = \frac{n}{2}(\phi(n)+e(n))$
8. $\sum_{i=1}^{n} lcm(i,n) = n\sum_{d|n} g(\frac{n}{d})$
9. $\sum_{i=1}^{n}\sum_{j=1}^{m} ij = \frac{n(n+1)m(m+1)}{4}$
10. $\sum_{i=1}^{n}\sum_{j=1}^{m} ij * e(gcd(i,j)) = \sum_{d=1}^{n}\mu(d)\sum_{i=1}^{\lfloor\frac{n}{d}\rfloor}\sum_{j=1}^{\lfloor\frac{m}{d}\rfloor} ij$

1.16 杜教筛

设$S(n) = \sum_{i=1}^{n} f(i)$, 若存在数论函数g,设$h = f * g$, 则有$\sum_{i=1}^{n} h(i) = \sum_{i=1}^{n} g(i)S(\lfloor\frac{n}{i}\rfloor)$, 化简得
$g(1)S(n) = \sum_{i=1}^{n} h(i) - \sum_{i=2}^{n} g(i)S(\lfloor\frac{n}{i}\rfloor)$,

如果我们可以$O(\sqrt{n})$计算$\sum_{i=1}^{n} h(i)$ff, O(1)计算g(n)前缀和，那么我们就可以分块查询S(n), 复杂度为$O(n^{\frac{3}{4}})$ff, 如果预处理前$n^{\frac{2}{3}}$项就可以优化到$O(n^{\frac{2}{3}})$.

欧拉函数前缀和 51nod 1239

题意：求$\sum_{i=1}^{n} \phi(n)$, $n < 1e11$;

解析：$S(n) = \sum_{i=1}^{n} i - \sum_{i=2}^{n} 1 * S(\lfloor \frac{n}{i} \rfloor)$

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const int N = 1.5e7+9;
const int mod = 1e9+7;
int prime[N], phi[N], cnt;
bool vis[N];
unordered_map<ll, ll> mp; // 记忆化
void init(){
    phi[1] = 1;
    for (int i = 2; i < N; i++) {
        if(!vis[i]) {
            prime[cnt++] = i; phi[i] = i - 1;
        } for (int j = 0; j < cnt; j++) {
            if(i*prime[j] >= N) break;
            vis[i*prime[j]] = 1;
            if(i%prime[j] == 0) { phi[i*prime[j]] = phi[i]*prime[j]; break; }
            phi[i*prime[j]] = phi[i]*(prime[j]-1);
        }
    } for (int i = 2; i < N; i++) phi[i] = (phi[i-1] + phi[i])%mod;
}
ll qp(ll a, int k){
    ll ans = 1;
    while(k) {
        if(k & 1) ans = ans * a % mod;
        a = a* a %mod; k >>= 1;
    } return ans;
}ll inv2 = qp(2, mod-2);

ll dfs(ll n) {
    if(n < N) return phi[n]; // 预处理
    if(mp.find(n) != mp.end()) return mp[n];
    ll ans = n%mod*(n%mod+1)%mod*inv2%mod; // n*(n+1)/2
    for (ll i = 2,j; i <= n; i=j+1) { // 分块
        j = n/(n/i);
        ans = (ans - (j-i+1)%mod*dfs(n/i)%mod) %mod; // 1 从i加到j = (j-i+1)
    } return mp[n]=(ans%mod + mod)%mod;
}

ll n;
int main() {
    init(); scanf("%lld", &n);
    return printf("%lld\n", dfs(n)), 0;
}
```

莫比乌斯函数前缀和

题意：求$\sum_{i=1}^{n} \mu(i)$, $i <= 1e10$

解析：u*I=e, 所以$S(n) = 1 - \sum_{i=2}^{n} S(\lfloor \frac{n}{i} \rfloor)$

```cpp
#include <bits/stdc++.h>
```

```cpp
using namespace std;
using ll = long long;
ll a, b;
const int N = 1e7+9;
int prime[N], cnt;ll mu[N];
bool vis[N];
unordered_map<ll, ll> mp;
void init(){ // 欧拉筛
    mu[1] = 1;
    for (int i = 2; i < N; i++) {
        if(!vis[i]) { prime[cnt++] = i; mu[i] = -1; }
        for (int j = 0; j < cnt; j++) {
            if(i*prime[j] >= N) break; vis[i*prime[j]] = 1;
            if(i%prime[j]==0) break; mu[i*prime[j]] = - mu[i];
        }
    } for (int i = 2; i < N; i++) mu[i] += mu[i-1];
}
ll dfs(ll n){
    if(n < N) return mu[n];
    if(mp.count(n)) return mp[n];
    ll ans = 1; // e
    for (ll i = 2,j; i <= n; i=j+1){
        j = n/(n/i); ans -= dfs(n/i)*(j-i+1); // 和上一题一样
    } return mp[n]=ans;
}

int main() {
    scanf("%lld%lld", &a, &b); init();
    return printf("%lld\n", dfs(b)-dfs(a-1)), 0;
}
```

一道习题：

$N^2-3N+2=\sum_{d|N} f(d)$, 求$\sum_{i=1}^{N} f(i)\ mod\ 1e9+7, N \le 1e9$

这道题很容易用杜教筛做，不过他的前1e7前缀预处理可以用容斥来写

```cpp
inline void init(){
    for(int i = 1; i < N; i++){
        f[i] = (i-1LL)*(i-2LL)%mod;
    } for (int i = 1; i < N; i++) {
        for (int j = i+i; j < N; j+=i)
            f[j] = ((f[j]-f[i])+mod)%mod; // 容斥
    } for (int i = 2; i < N; i++) f[i]=(f[i]+f[i-1])%mod; // 前缀和
}
```

1.17 Extended Eratosthenes Sieve**洲阁筛**

洲阁筛时一种在$O(\frac{n^{\frac{3}{4}}}{log\ n})$ 时间内计算大多数积性函数的前缀和的方法。

F(x)是一个积性函数，要求在低于线性的时间内求出。当p为质数时，$F(p^c)$是关于p的低阶多项式。

对于1-n中的所有数，我们按照是否含有大于$\sqrt{n}$ 的质因子分为两类，则显然有

$$\sum_{i=1}^{n} F(i) = \sum_{i=1\&\&i\ has't\ bigger\ than \sqrt{n} prime\ d}^{n} F(i)(1+\sum_{\sqrt{n}\le j\le \lfloor \frac{n}{i} \rfloor \&\&j\ is\ prime} F(j))$$

事实上，我们可以预处理 $1 \le i < \sqrt{n}$ 的F, 那么现在的问题就是要解决

1. $\sum_{\sqrt{n}<j<\le \lfloor \frac{n}{i} \rfloor \&\&jisprime} F(j)$

我们令g(i, j) 表示[1, j]中与前i各质数互质的数的k次幂和。显然有

$$g(i,j) = g(i-1,j) - p_i^k g(i-1, \lfloor \frac{j}{p_i} \rfloor)$$

$$p_i^2 > j, g(i,j) = g(i-1,j) - p_i^k$$

2.

$$\sum_{\sqrt{n} \le i \le n \&\& i\ has't\ bigger\ than \sqrt{n}} F(i)$$

设f(i, j) 表示[1, j]中仅由i个质数个组成的数的F(x)之和

$$f(i,j) = f(i-1,j) + \sum_{c \ge 1} F(p_i^c) f(i-1, \lfloor \frac{j}{p_i^c} \rfloor)$$

$$when\ p_i^2 > j \text{fi} f(i,j) = f(i-1,j) + F(p_i)$$

素数个数

题意：求1-n中素数个数，n¡1e11;

解析：对于这个虽然不是积性函数，但是我们依然可以分为两块1-$\sqrt{n}$ 和$(\sqrt{n}+1)-n$

前者可以O(n)预处理，后者就可以利用上面第一块的$g(\sqrt{n}, n)$递推

```
#include <bits/stdc++.h>
const int N = 4e6+9;
using ll = long long;
using namespace std;
int prime[N], cnt, res[N]; bool vis[N];
...
ll n, v[N], k, last[N], g[N];
int solve(){
    if(n < N) return printf("%d\n", res[n]), 0;
    int tot = 0, sn = sqrt(n+0.5); // sqrt(n)
    int pos = upper_bound(prime, prime+cnt, sn) - prime; // 第一个大于sqrt(n)的质数
    for (ll i = n; i >= 1; i = n/(n/i+1) ) v[++tot] = n/i; // 分块
    for (int i = 1; i <= tot; i++) g[i]=v[i], last[i] = 0; // 初始化
    for (int i = 0; i < pos; i++) {
        for (int j = tot; j; j--) {
            k = v[j]/prime[i]; if(k < prime[i]) break; // 忽略j小于p*p
            k = k < sn? k: tot - n/k + 1; // 找到在v中的下标
            g[j] -= g[k] - (i - last[k]); // 减去g[k]中已经减过的
            last[j] = i + 1; // 上一次处理的是第i+1个质数(从1开始)
        }
    } printf("%lld\n", res[sn]*1LL + g[tot] - 1);
}

int main(){
    init(); scanf("%lld", &n); return solve(), 0;
}
```

洲阁筛实现起来较为复杂，现在已经被Min_25筛替代。

1.18 其他杂题

题意：定义w(n) = n的质因子个数，$g(n) = 2^{w(n)}$, 求$S(n) = \sum_{i=1}^{n} g(i)\ mod\ 1e9+7$， CCPC 杭州2016J

$$
\begin{aligned}
ans &= \sum_{i=1}^{n} g(i) = \sum_{i=1}^{n} \sum_{d|i} \mu^2(d) \\
&= \sum_{i=1}^{n} \sum_{d|i} \sum_{k^2|d} \mu(k) \\
&= \sum_{k=1}^{n} \mu(k) \sum_{k^2|d} \lfloor \frac{n}{d} \rfloor \\
&= \sum_{k=1}^{n} \mu(k) \sum_{i=1}^{\lfloor \frac{n}{k^2} \rfloor} \lfloor \frac{n}{k^2 i} \rfloor \\
&= \sum_{k=1}^{\sqrt{n}} \mu(k) S(\lfloor \frac{n}{k^2} \rfloor)
\end{aligned}
$$

```
#include <bits/stdc++.h>
const int N = 1e6+9;
const int mod = 1e9+7;
using namespace std;
using ll = long long;
int prime[N], cnt, mu[N];
bool vis[N];
...
ll n;int f[N];
inline int S(ll n) {
    if(n < N && f[n]) return f[n];
    ll ans = 0;
    for (ll i = 1, j; i <= n; i = j+1) {
        j = n/(n/i); ans = (ans + (n/i)*(j-i+1))%mod;
    } if(n < N) f[n] = ans;
    return ans;
}

int main() {
    int T, ks=1; scanf("%d", &T); init();
    while(T--) {
        scanf("%lld", &n);
        ll ans = 0;
        for(ll k = 1; k * k <= n; k++) {
            if(mu[k]) ans = (ans + mu[k]*S(n/k/k)) % mod;
        } printf("Case #%d: %lld\n",ks++, (ans+mod)%mod);
    }
}
```

2018 四川省赛

题意：求$\sum_{i=1}^{n}\sum_{j=1}^{i} n\%(ij), n \le 1e11$

原式=$\sum_{i=1}^{n}\sum_{j=1}^{i} n - \lfloor \frac{n}{ij} \rfloor = n*n*(n+1)/2 - \sum_{i=1}^{n}\sum_{j=1}^{i}\lfloor \frac{n}{ij} \rfloor$

$= n*n*(n+1)/2 - 1/2\sum_{i=1}^{n}\sum_{j=1}^{n}\lfloor \frac{n}{ij} \rfloor - 1/2\sum_{i=1}^{n}\lfloor \frac{n}{i^2} \rfloor$

欧拉筛预处理前$n^{2/3}$项后其他的暴力算，总复杂度是$O(n^{2/3})$

由于最后答案较大需要使用大数或者__int128

预处理的话，可以考虑$f[n] = \sum_{i=1}^{i}\lfloor \frac{n}{i} \rfloor i$, 那么$f[n] - f[n-1] = \sum_{i=1}^{n} i*(\lfloor \frac{n}{i} \rfloor - \lfloor \frac{n-1}{i} \rfloor) = \sum_{i|n} i$, 这个在欧拉筛中可以预处理,然后求一遍前缀和

```
#include <bits/stdc++.h>
#define ll long long
typedef __int128 dll;
const int N = 3e7;
```

```cpp
using namespace std;
ll n;
int prime[N], cnt, t[N];
dll f[N];
bool vis[N];
void init(){
    f[1] = t[1] = 1;
    for (int i = 2; i < N; i++) {
        if(!vis[i]) {prime[cnt++]=i; f[i]=i+1; t[i]=i;}
        for (int j = 0; j < cnt; j++) {
            if(i*prime[j]>=N) break; vis[i*prime[j]] = 1;
            if(i%prime[j] == 0) {
                f[i*prime[j]] = f[i/t[i]] + f[i]*prime[j];
                t[i*prime[j]] = t[i] * prime[j]; break;
            } f[i*prime[j]] = f[i]*(prime[j]+1); t[i*prime[j]] = prime[j];
        }
    } for (int i = 2; i < N; i++) f[i] += f[i-1];
}

inline dll F(ll n) { // 分块求∑n/i * i
    if(n < N) return f[n];
    dll ans = 0;
    for (ll i = 1,j; i <= n; i=j+1) {
        j = n/(n/i);
        ans += ((dll)(j-i+1))*(i+j)/2*(n/i);
    } return ans;
}

void print(dll x){ // int128位数输出
    if(x > 9) print(x/10);
    putchar(x%10 + '0');
}

int main() {
    int t; scanf("%d", &t); init();
    while(t--) {
        scanf("%lld", &n);
        dll ans = n; ans *= n; ans += ans * n; // ans = n*n*(n+1);
        for (int i = 1; 1LL * i * i <= n; i++) ans -= n/i/i*i*i;
        for (ll i = 1,j; i <= n; i=j+1) {
            j=n/(n/i);
            ans -= ((dll)(j-i+1))*(i+j)/2*F(n/i);
        } print(ans/2); puts("");
    } return 0;
}
```

1.19 rng_58-clj equation

$$\sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{k=1}^{c} d(ijk) = \sum_{gcd(i,j)=gcd(j,k)=gcd(i,k)=1} \lfloor\frac{a}{i}\rfloor\lfloor\frac{b}{j}\rfloor\lfloor\frac{c}{k}\rfloor$$

事实上这个等式可以扩展至任意维。

题意：求$\sum_{i=1}^{a}\sum_{j=1}^{b}$d(ij), T, a, b ¡ 5e4;

原式化简为$\sum_{gcd(i,j)=1}\lfloor\frac{a}{i}\rfloor\lfloor\frac{b}{j}\rfloor = \sum_{g=1}^{min(a,b)}\mu(g)\sum_{i=1}^{a/g}\sum_{j=1}^{b/g}\lfloor\frac{a}{i}\rfloor\lfloor\frac{b}{j}\rfloor$

分块查询即可,代码略

题意：$\sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{k=1}^{c} d(ijk)$, a,b,c ¡ 2000

化简：$\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{t} \lfloor\frac{n}{i}\rfloor\lfloor\frac{m}{j}\rfloor\lfloor\frac{t}{k}\rfloor[(i,j)=1][(j,k)=1][(i,k)=1]$

$=¿\sum_{k=1}^{t}\lfloor\frac{t}{k}\rfloor\sum_{d=1}^{n}\mu(d)\sum_{i=1}^{\lfloor\frac{n}{i}\rfloor}\lfloor\frac{n}{i*d}\rfloor[(di,k)=1]\sum_{j=1}^{\lfloor\frac{m}{j}\rfloor}\lfloor\frac{m}{i*d}\rfloor[(dj,k)=1]$

暴力算一下就行了(gcd可以记忆化一下)

```
#include <bits/stdc++.h>
#define gcd __gcd
const int N = 2e3+5;
const int mod = (1<<30) - 1;
using namespace std;

int mu[N], prime[N], a, b, c, gd[N][N], cnt;
bool vis[N];
...
inline int cal(int d, int x){
    int ans = d;
    for(int i = 2; i <= d; i++) if(gd[i][x] == 1) ans += d/i;
    return ans;
}

int main(){
    scanf("%d%d%d",&a,&b,&c); init();
    int ans = 0; if(b > c) swap(b, c);
    for(int i = 1; i <= a; i++) { for(int j = 1; j <= b; j++)
        if(gcd(i, j) == 1 && mu[j]) ans += (a/i)*mu[j]*cal(b/j, i)*cal(c/j, i);
    } return printf("%u\n",ans&mod), 0;
}
```

BZOJ4176

题意：$\sum_{i=1}^{n}\sum_{j=1}^{n} d(ij), n \le 1e9$

化简得$=\sum_{g=1}^{n}\mu(g)\sum_{i=1}^{n/g}\sum_{j=1}^{n/g}\lfloor\frac{n}{i}\rfloor\lfloor\frac{n}{j}\rfloor$

分块查询的难调在于u(g)的前缀和，这个使用杜教筛就可以解决。

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const int N = 1e7+9;
const int mod = 1e9+7;
int prime[N], cnt;ll mu[N];
bool vis[N];
unordered_map<ll, ll> mp;
ll n;
...
ll dfs(ll n){ // 莫比乌斯前缀和
    if(n < N) return mu[n];
    if(mp.count(n)) return mp[n];
    ll ans = 1; // e
    for (ll i = 2,j; i <= n; i=j+1){
        j = n/(n/i); ans -= dfs(n/i)*(j-i+1);
    } return mp[n]=ans;
}

ll F(ll n) { // 分块求和
```

```
    ll ans = 0;
    for (ll i = 1,j; i <= n; i=j+1) {
        j = n/(n/i);
        ans = (ans + (j-i+1)%mod*(n/i)) % mod;
    } return ans * ans % mod;
}

int main() {
    init(); scanf("%lld", &n); ll ans = 0;
    for (ll i=1,j; i <= n; i=j+1) {
        j = n/(n/i);
        ans = (ans + (dfs(j) - dfs(i-1) + mod)%mod*F(n/i)%mod) % mod;
    } printf("%lld\n", ans);
}
```

1.20 Min_25筛

Min_25筛是一个替代洲阁晒的新产物，也是用于求解积性函数求和的问题，时间空间复杂度都比洲阁晒要优秀。

同样考虑筛质数，对1-n内所有质数求和。

设函数$S(x,j)=\sum_{i=2}^{x} i*[i$为质数或i的最小质因子大于$p_j]$, 质数从p1开始

S(x, 0)=x*(x+1)/2 - 1; 我们要求的就是S(n, p0)

考虑转移：

$p_j^2 > x, S(x,j)=S(x,j-1)$

否则$S(x,j)=S(x,j-1)-f(p_j)*(S(\lfloor\frac{x}{p_j}\rfloor,j-1)-\sum_{i=1}^{j-1} f(p_i))$

求具体积性函数和时

$G(x,j)=G(x,j+1)+\sum_{k=1}(G(\lfloor\frac{x}{p_j^k}\rfloor,j+1)-\sum_{i=1}^{x} f(p_i))*f(p_j^k)+\sum_{k=2} f(p_j^k)$

能力有限，以后再学。

section原根

每一个质数可以去求解其原根g，将原本的乘法运算转化为指数上的加法运算，可以利用FFT优化。

```
int tp[50];
int find_root(int x) {
    int f, phi = x-1;
    for(int i = 0; phi && i < cnt; i++) {
        if(phi % prime[i] == 0) {
            tp[++tp[0]] = prime[i];
            while(phi%prime[i]==0) phi/=prime[i];
        }
    } for(int g = 2; g <= x-1; g++) {
        f = 1;
        for(int i=1;i<=tp[0];i++) {
            if(qp(g, (x-1)/tp[i], x)==1) {
                f=0; break;
            }
        } if(f) return g;
    } return 0;
}
```

牛客camp day 2

一个数列a ,2e5个数，输出n个数，表示存在多少数对(i,j) st. $a_i*a_j\%p=a_k$

```
#include <bits/stdc++.h>
using ll = long long;
using namespace std;
const int mod = 1e9+7;
const int N = 5e5+9;
```

```
int prime[N], p[N], cnt;
ll a[N], x[N], ans[N], e[N];
...
namespace FFT {
    #define rep(i,a,b) for(int i=(a);i<=(b);i++)
    const double pi=acos(-1);
    const int maxn=1<<19;
    struct cp {
        double a,b;
        cp(){}
        cp(double _x,double _y){a=_x,b=_y;}
        cp operator +(const cp &o)const{return (cp){a+o.a,b+o.b};}
        cp operator -(const cp &o)const{return (cp){a-o.a,b-o.b};}
        cp operator *(const cp &o)const{return (cp){a*o.a-b*o.b,b*o.a+a*o.b};}
        cp operator *(const double &o)const{return (cp){a*o,b*o};}
        cp operator !()const{return (cp){a,-b};}
    } x[maxn],y[maxn],z[maxn],w[maxn];
    void fft(cp x[],int k,int v) {
        int i,j,l;
        for(i=0,j=0;i<k;i++) {
            if(i>j)swap(x[i],x[j]);
            for(l=k>>1;(j^=l)<l;l>>=1);
        } w[0]=(cp){1,0};
        for(i=2;i<=k;i<<=1) {
            cp g=(cp){cos(2*pi/i),(v?-1:1)*sin(2*pi/i)};
            for(j=(i>>1);j>=0;j-=2)w[j]=w[j>>1];
            for(j=1;j<i>>1;j+=2)w[j]=w[j-1]*g;
            for(j=0;j<k;j+=i) {
                cp *a=x+j,*b=a+(i>>1);
                for(l=0;l<i>>1;l++) {
                    cp o=b[l]*w[l];
                    b[l]=a[l]-o;
                    a[l]=a[l]+o;
                }
            }
        } if(v)for(i=0;i<k;i++)x[i]=(cp){x[i].a/k,x[i].b/k};
    }
    void mul(ll *a,ll *b,ll *c,int l1,int l2) {
        if(l1<128&&l2<128) {
            rep(i,0,l1+l2)a[i]=0;
            rep(i,0,l1)rep(j,0,l2)a[i+j]+=b[i]*c[j];
            return;
        } int K;
        for(K=1;K<=l1+l2;K<<=1);
        rep(i,0,l1)x[i]=cp(b[i],0);
        rep(i,0,l2)y[i]=cp(c[i],0);
        rep(i,l1+1,K)x[i]=cp(0,0);
        rep(i,l2+1,K)y[i]=cp(0,0);
        fft(x,K,0);fft(y,K,0);
        rep(i,0,K)z[i]=x[i]*y[i];
        fft(z,K,1);
        rep(i,0,l1+l2)a[i]=(ll)(z[i].a+0.5);
    }
};

```

```
int main() {
    init(); int n, p, now, i, g;
    while(~scanf("%d%d",&n,&p)) {
        g = find_root(p); now = 1;
        for (int i = 1; i < p; i++) {
            now = now * g % p;
            e[now]=i%(p-1);
        } memset(a, 0, sizeof a);
        ll res = 0;
        for (int i = 1; i <= n; i++) {
            scanf("%d", &x[i]);
            if(x[i] % p) a[e[x[i]%p]]++;
            else res ++;
        } memset(ans, 0, sizeof ans);
        FFT::mul(ans, a, a, p-1, p-1);
        for (int i = p-1; i < 2*p; i++) {
            ans[i-(p-1)] += ans[i];
        } for (int i = 1; i <= n; i++) {
            if(x[i] >= p) puts("0");
            else if(x[i] == 0) printf("%lld\n", res*res + res*2*(n-res));
            else printf("%lld\n", ans[e[x[i]]]);
        }
    } return 0;
}
```

1.21 二次剩余

$x^2 \equiv n \pmod p$ 我们只讨论p为奇素数的情况。

$$\left(\frac{a}{p}\right) = \begin{cases} 1, & a\text{在模}p\text{意义下是二次剩余} \\ -1, & a\text{在模}p\text{意义下是非二次剩余} \\ 0, & a \equiv 0 \pmod p \end{cases}$$

定理2：$\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \pmod p$

找到t使得$\left(\frac{t^2-n}{n}\right) = -1$，设 $\omega = \sqrt{t^2-n}$

解为$a \equiv (t+\omega)^{\frac{P+1}{2}} \pmod P$

未检验的模板：

```
#define fo(i,a,b) for(int i=a;i<=b;++i)
#define fod(i,a,b) for(int i=a;i>=b;--i)
#define min(q,w) ((q)>(w)?(w):(q))
#define max(q,w) ((q)<(w)?(w):(q))
using namespace std;
typedef int LL;
const int N=1500;
int mo;
int read(int &n) {
    char ch=' ';int q=0,w=1;
    for(;(ch!='-')&&((ch<'0')||(ch>'9'));ch=getchar());
    if(ch=='-')w=-1,ch=getchar();
    for(;ch>='0' && ch<='9';ch=getchar())q=q*10+ch-48;n=q*w;return n;
}
int m,n,ans;
LL W;
struct qqww {
```

```
    LL x,y;
    qqww(LL _x=0,LL _y=0){x=_x,y=_y;}
    friend qqww operator *(qqww q,qqww w){return qqww((q.x*w.x+q.y*w.y%mo*W)%mo,(q.x*w.y+q.y*w.x)%mo)
        ;}
};
LL ksm(LL q,int w,int Mo) {
    LL ans=1;q=q%Mo;(q<0?q=q+Mo:0);
    for(;w;w>>=1,q=q*q%Mo)if(w&1)ans=ans*q%Mo;
    return ans;
}
qqww ksm(qqww q,int w,int Mo) {
    ::mo=Mo;
    qqww ans(1,0);
    for(;w;w>>=1,q=q*q)if(w&1)ans=ans*q;
    return ans;
}
int RD(int mo){return rand()%mo;}
LL Cipolla(int n,int mo) {
    if(w==2)return 1;
    LL q=ksm(n,(mo-1)>>1,mo);
    if(q==0||q==mo-1)return -1;
    for(q=RD(mo);ksm(q*q-n+mo,((mo-1)>>1),mo)!=mo-1;q=RD(mo));
    qqww t(q,1);W=(q*q-n+mo)%mo;
    t=ksm(t,((mo+1)>>1),mo);
    return (t.x+mo)%mo;
}
int main() {
    int q,w,_;
    srand(19890604);
    for(read(_);_;_--) {
        read(q),read(w);
        ans=Cipolla(q,w);
        if(ans==-1)printf("No root\n");
        else if(ans==w-ans)printf("%d\n",ans);
        else if(ans<w-ans)printf("%d %d\n",ans,w-ans);
        else printf("%d %d\n",w-ans,ans);
    } return 0;
}
```

1.22 pell方程与BSGS

pell 方程： $x^2 - dy^2 = 1$ ， d为正整数

c++:

```
bool pell(int D, int& x, int& y) {
    int sqrtD = sqrt(D + 0.4);
    if( sqrtD * sqrtD == D ) return false;
    int c = sqrtD, q = D - c * c, a = (c + sqrtD) / q;
    int step = 0;
    int X[] = { 1, sqrtD };
    int Y[] = { 0, 1 };
    while( true ) {
        X[step] = a * X[step^1] + X[step];
        Y[step] = a * Y[step^1] + Y[step];
        c = a * q - c;
```

```
        q = (D - c * c) / q;
        a = (c + sqrtD) / q;
        step ^= 1;
        if( c == sqrtD && q == 1 && step ) {
            x = X[0], y = Y[0];
            return true;
        }
    }
}
```

java:

```
static class Pell {
    int D;
    BigInteger x, y ;
    boolean status = true;
    Pell() {};
    Pell(int d) {D = d;}
    void slove() {
        int sqrtD = (int)Math.sqrt((double)D);
        if(sqrtD * sqrtD == D){ status = false; return ; }
        BigInteger N = BigInteger.valueOf(D);
        BigInteger SqrtD = BigInteger.valueOf(sqrtD);
        BigInteger c = SqrtD;
        BigInteger q = N.subtract(c.multiply(c));
        BigInteger a = c.add(SqrtD).divide(q);

        int step = 0;
        BigInteger[] X = {BigInteger.ONE, SqrtD};
        BigInteger[] Y = {BigInteger.ZERO, BigInteger.ONE};
        while(true) {
            X[step] = a.multiply(X[step ^ 1]).add(X[step]);
            Y[step] = a.multiply(Y[step ^ 1]).add(Y[step]);

            c = a.multiply(q).subtract(c);
            q = (N.subtract(c.multiply(c))).divide(q);
            a = (c.add(SqrtD)).divide(q);
            step ^= 1;
            if (c.equals(SqrtD) && q.equals(BigInteger.ONE) && step == 1) {
                x = X[0]; y = Y[0]; return;
            }
        }
    }
}
```

java：连分数法(答案很大)

```
/*
 * 逐项求解sqrt(d)的连分数
 */
BigInteger x = BigInteger.ONE;
BigInteger y = BigInteger.ONE;
BigInteger a,ak,N,P1,P2,p1,p2,Q1,Q2,q1,q2;
q1 = p2 = P1 = BigInteger.ZERO;
p1 = q2 = Q1 = BigInteger.ONE;
N = BigInteger.valueOf(n);
a = BigInteger.valueOf(k);
```

```
ak = a;
while(!x.multiply(x).subtract(N.multiply(y).multiply(y)).equals(BigInteger.ONE)){
    x = ak.multiply(p1).add(p2);
    y = ak.multiply(q1).add(q2);
    P2 = ak.multiply(Q1).subtract(P1);
    Q2 = N.subtract(P2.multiply(P2)).divide(Q1);
    ak = P2.add(a).divide(Q2);

    P1 = P2; Q1 = Q2;

    p2 = p1; p1 = x;
    q2 = q1; q1 = y;
}
```

1.22.1 BSGS

bsgs大步小步算法解决： $a^x \equiv b(mod\ p)$,算x

```
const int mod = 1e5 + 7;
ll has[mod + 100], id[mod + 100];
ll find(ll x){
    ll t = x % mod;
    while(has[t] != x && has[t] != -1){ t = (t+1)%mod; }
    return t;
}
void insert(ll x, ll i){
    ll pos = find(x);
    if(has[pos] == -1){ has[pos] = x; id[pos] = i; }
}
ll get(ll x){
    ll pos = find(x);
    return has[pos] == x? id[pos] : -1;
}
void ex_gcd(ll a, ll b, ll &d, ll &x, ll &y){
    if(b == 0){ x = 1; y = 0; d = a; return ; }
    ex_gcd(b, a%b, d, y, x); y -= a/b*x;
}
ll inv(ll a, ll p){
    ll x, y ,d; ex_gcd(a, p, d, x, y);
    return d==1?(x%p+p)%p : -1;
}

ll BSGS(ll a, ll b, ll p){
    memset(has, -1, sizeof has);
    memset(id, -1, sizeof id);
    ll m = (ll) ceil(sqrt(p+0.5)); ll tmp = 1;
    for(ll i = 0; i < m; i++){
        insert(tmp , i);
        tmp = tmp * a % p;
    } ll base = inv(tmp, p);
    if(base == -1) return -1;
    ll res = b, z;
    for(ll i = 0; i < m; i++){
        if((z = get(res)) != -1) return i * m + z;
        res = res * base % p;
```

```
    } return -1;
}
```

扩展ex_bsgs, 模数任意

```
ll slove(ll a, ll b, ll p){
    ll tmp = 1;
    for(int i = 0; i < 50; i++) {
        if(tmp == b) return i;
        tmp = tmp * a % p;
    } ll cnt = 0, d = 1 % p;
    while((tmp = gcd(a, p)) != 1){
        if(b % tmp)return -1;
        b /= tmp; p /= tmp;
        d = a / tmp * d % p;
        cnt ++ ;
    } b = b * inv(d, p) % p;
    ll ans = BSGS(a, b, p);
    if(ans == -1)return -1;
    else return ans + cnt;
}
```

1.23 扩展Lucas

解决$C(n,m)\%p$, p 不为质数的情况

```
ll C(ll n, ll m, ll p) {
    if(m>n) return 0;
    ll res = 1, i, a, b;
    for(i = 1; i <= m; i++) {
        a = (n+1-i) % p;
        b = inv(i%p, p);
        res = res*a%p*b%p;
    } return res;
}

ll Lucas(ll n, ll m, ll p) {
    if(m == 0) return 1;
    return Lucas(n/p, m/p, p)*C(n%p, m%p, p) % p;
}

ll cal(ll n, ll a, ll b, ll p) {
    if(!n) return 1;
    ll i, y = n/p, tmp = 1;
    for(i = 1; i <= p; i++) if(i%a) tmp = tmp*i%p;
    ll ans = pow(tmp, y, p);
    for(i = y*p+1; i <= n; i++) if(i%a) ans = ans*i%p;
    return ans * cal(n/a, a, b, p)%p;
}

ll multiLucas(ll n, ll m, ll a, ll b, ll p) {
    ll i, t1, t2, t3, s = 0, tmp;
    for(i = n; i; i/=a) s += i/a;
    for(i = m; i; i/=a) s -= i/a;
    for(i = n-m; i; i/=a) s -= i/a;
    tmp = pow(a, s, p);
```

```
    t1 = cal(n, a, b, p);
    t2 = cal(m, a, b, p);
    t3 = cal(n-m, a, b, p);
    return tmp*t1%p*inv(t2, p)%p*inv(t3, p)%p;
}

ll exLucas(ll n, ll m, ll p) {
    ll i, d, c, t, x, y, q[100], a[100], e = 0;
    for(i = 2; i*i <= p; i++) {
        if(p % i == 0) {
            q[++e] = 1;
            t = 0;
            while(p%i==0) {
                p /= i;
                q[e] *= i;
                t++;
            }
            if(q[e] == i) a[e] = Lucas(n, m, q[e]);
            else a[e] = multiLucas(n, m, i, t, q[e]);
        }
    }
    if(p > 1) {
        q[++e] = p;
        a[e] = Lucas(n, m, p);
    }
    for(i = 2; i <= e; i++) {
        d = exgcd(q[1], q[i], x, y);
        c = a[i]-a[1];
        if(c%d) exit(-1);
        t = q[i]/d;
        x = (c/d*x%t+t)%t;
        a[1] = q[1]*x+a[1];
        q[1] = q[1]*q[i]/d;
    }
    return a[1];
}
```

1.24 关于模数非素数的问题

关于模数为合数的情况，我们通常采用几种情况：

1. 使用java种的BigIntegeter

2. 使用自带的c++大数模板或者__int128

3. 如果除数为n, 那么就讲模数改为n*mod, 得到的最后答案处以n，这个情况就要求你最后的数是一定能够被n 整除的。与此同时，还要注意一个事项模数大了之后，乘法运算会溢出。

采用快速乘法，或者使用:

```
ll qmul(ll x, ll y) {
    return (x*y - (ll)((long double)x/mod*y)*mod+mod)%mod
}
```

2 Combinatorial Math

2.1 常见数列

2.1.1 错排

0,0,1,2,9,44,265
当n个编号元素放在n个编号位置，元素编号与位置编号各不对应的方法数

$$D(n) = (n-1) * (D(n-1) + D(n-2))$$

2.1.2 卡特兰数

h0=1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670
计算公式:

$$h_n = \frac{1}{n+1}C_{2n}^n = \frac{(2n)!}{(n+1)!n!}, h_n = C_{2n}^n - C_{2n}^{n-1}, h_0 = 1 \ \&\& \ h_{n+1} = \frac{2(2n+1)}{n+2}h_n, h_{n+1} = \sum_{i=0}^{n} h_i h_{n-i}$$

好看的性质：奇卡特兰数都满足$2^k - 1$
1. 括号问题
$P = \prod_{i=1}^{n} a_i$ 不改变顺序，只用括号表示成对的乘积。问方案数：h(n)
2. 栈问题
给出一个n，问存在多少2n长度的01序列，使得序列任意前缀1的个数不少于0的个数。
本质就是有n个数进栈，求有多少不同的出栈序列方案。
3. 在n × n格点中不越过对角线的单调路径的方案个数：h(n)
4. n+1个叶子的二叉树的个数：h(n)
5. n+2的凸多边形分为n个三角形的方法个数 h(n)
6. 所有不同构的含n个分枝结点的满二叉树的个数
7. 用n个长方形填充一个高度为n的阶梯状图形的方法个数

2.1.3 第二类斯特灵数

第二类斯特林数：s(p, k)把p元素集合划分到k个不可区分的盒子里且没有空盒子的划分个数。

n=0	1
n=1	0 1
n=2	0 1 1
n=3	0 1 3 1
n=4	0 1 7 6 1
n=5	0 1 15 25 10 1
n=6	0 1 31 90 65 15 1

通项公式:

$$S(p,p) = 1, S(p,0) = 0 (p \geq 1), S(p,k) = k * S(p-1,k) + S(p-1,k-1) (1 \leq k \leq p-1)$$

$$S(n,m) = \frac{1}{m!}\sum_{k=0}^{m}(-1)^k C(m,k)(m-k)^n$$

$$n^k = \sum_{i=0}^{k} S(k,i) \times i! \times C(n,i)$$

n个不同的球，放入m个无区别的盒子，不允许盒子为空：S(n,m)
n个不同的球，放入m个无区别的盒子，允许盒子为空。 $\sum_{k=0}^{m} S(n,k)$

2.1.4 第一类斯特林数

第一类Stirling数s(p,k)计数的是把p个对象排成k个非空循环排列的方法数。

n=1	0 1	0 1
n=2	0 1 1	0 -1 1
n=3	0 2 3 1	0 2 -3 1
n=4	0 6 11 6 1	0 -6 11 -6 1
n=5	0 24 50 35 10 1	0 24 -50 35 -10 1

$$S(p,p)=1(p\geq 0), S(p,0)=0(p\geq 1), S(p,k)=(p-1)S(p-1,k)+S(p-1,k-1)$$

2.1.5 bell数

1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975

Bell数B(p)是将p元素集合分到非空且不可区分盒子的划分个数（没有说分到几个盒子里面）。

$$B_n=\sum_{k=1}^{n} S2(n,k)$$

$$B_{n+1}=\sum_{k=0}^{n} C(n,k)B_k$$

对于任何质数p，有 $B_{p+n}\equiv B_n+B_{n+1}(modp)$

2.2 奇怪结论

求(0, 0) − > (n+1, m+1) 的两条不相交(可重合)的路径。

$$M=\begin{pmatrix} e(a_1,b_1) & e(a_1,b_2) & \cdots & e(a_1,b_n) \\ e(a_2,b_1) & e(a_2,b_2) & \cdots & e(a_2,b_n) \\ \vdots & \vdots & \ddots & \vdots \\ e(a_n,b_1) & e(a_n,b_2) & \cdots & e(a_n,b_n) \end{pmatrix}$$

图 1:

e(a， b) 表示路径数

将期中一条向右下角平移(1,-1), 变为(0, 0)-¿(n+1,m+1), (1, -1)-¿(n+2, m);

所以该题答案为$(C_{n+m}^{n})^2-C_{n+m}^{m-1}\Delta C_{n+m}^{n-1}$

2.3 二项式反演

$$f_n=\sum_{i=0}^{n}(-1)^i\binom{n}{i}g_i\Leftrightarrow g_n=\sum_{i=0}^{n}(-1)^i\binom{n}{i}f_i \quad f_n=\sum_{i=0}^{n}\binom{n}{i}g_i\Leftrightarrow g_n=\sum_{i=0}^{n}(-1)^{n-i}\binom{n}{i}f_i$$

n个家庭，每个家庭有a_i女生，b_i个男生,相同家庭的人不能结婚，问有多少种配对方法满足题意；

考虑每个家庭不满足的方案数，至少冲突k个即$C(a_i,k)*C(b_i,k)*k!$ff 种。

这样就有多项式f(x) = x... ； x的系数表示至少的方案数。

由二项式定理可知，第二个式子可以求出恰好冲突0项。

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const ll mod = 998244353;
const ll g = 3; // 原根
const int N = 1e5+9;
vector<ll> v[N];
ll x1[N<<1], x2[N<<1], fac[N], inv[N], ans, flag;
int n, all, cnt, sz;
```

```
10 inline ll qp(ll a, ll k) { ll ans(1); a %= mod; while(k) { if(k&1) ans = ans * a % mod; k >>= 1; a =
       a * a % mod; } return ans; }
11 inline ll C(ll n, ll m) { return m>n?0:fac[n]*inv[m]%mod*inv[n-m]%mod; }
12 struct cmp{ bool operator ()(int a, int b) { return v[a].size() > v[b].size(); } };
13 priority_queue<ll, vector<ll>, cmp> q;
14 void init() {
15     fac[0] = inv[0] = 1;
16     for (int i = 1; i < N; i++) fac[i] = fac[i-1]*i%mod;
17     inv[N-1] = qp(fac[N-1], mod-2);
18     for (int i = N-2; i; i--) inv[i] = (i+1)*inv[i+1]%mod;
19 }
20 inline void change(ll y[], int len) {
21     for (int i = 1, j = len/2; i < len-1; i++) {
22         if(i < j) swap(y[i], y[j]);
23         int k = len/2;
24         while(j >= k) { j -= k; k/=2; }
25         if(j < k) j += k;
26     }
27 }
28 inline void ntt(ll y[], int len, int on) {
29     change(y, len);
30     for (int h = 2; h <= len; h <<= 1) {
31         ll wn = qp(g, (mod-1)/h);
32         if(on == -1) wn = qp(wn, mod-2);
33         for (int j = 0; j < len; j += h) {
34             ll w = 1;
35             for (int k = j; k < j+h/2; k++) {
36                 ll u = y[k];
37                 ll t = w * y[k+h/2]%mod;
38                 y[k] = (u+t)%mod;
39                 y[k+h/2]=(u-t+mod)%mod;
40                 w = w*wn % mod;
41             }
42         }
43     } if(on == -1) {
44         ll t = qp(len, mod-2);
45         for(int i = 0; i < len; i++) y[i] = y[i]*t%mod;
46     }
47 }
48 void mul(vector<ll> &a, vector<ll> &b) {
49     int len = 1;
50     int sz1 = a.size(), sz2 = b.size();
51     while(len <= sz1 + sz2 - 1) len <<= 1;
52     for (int i = 0; i < sz1; i++) x1[i] = a[i];
53     for (int i = sz1; i <= len; i++) x1[i] = 0;
54     for (int i = 0; i < sz2; i++) x2[i] = b[i];
55     for (int i = sz2; i <= len; i++) x2[i] = 0;
56     ntt(x1, len, 1); ntt(x2, len, 1);
57     for (int i = 0; i < len; i++) x1[i]*=x2[i];
58     ntt(x1, len, -1); b.resize(sz1+sz2-1);
59     for (int i = 0; i <= sz1+sz2-2; i++) b[i]=x1[i];
60 }
61 int main() {
62     init(); int T; scanf("%d", &T);
63     while(T--) {
```

```
        scanf("%d", &n); all = 0;
        while(!q.empty()) q.pop();
        for (int i = 1; i <= n; i++) {
            int x, y; scanf("%d%d", &x, &y);
            v[i].resize(min(x,y)+1);
            for (int k = 0; k <= min(x, y); k++) {
                v[i][k] = C(x, k)*C(y, k)%mod*fac[k]%mod;
            } q.push(i); all += x;
        } int x(1), y(1);
        for (int i = 1; i < n; i++) {
            x = q.top(); q.pop();
            y = q.top(); q.pop();
            mul(v[x], v[y]); q.push(y);
        } ans = 0; flag = 1; sz = (int)v[y].size();
        for (int i = 0; i < sz; i++) {
            ans = (ans + flag*fac[all-i]*v[y][i]%mod+mod)%mod;
            flag = - flag;
        } printf("%lld\n", ans);
    } return 0;
}
```

2.4 Burnside 引理与polya原理

设G=a1,a2,…ag是目标集[1,n]上的置换群。每个置换都写成不相交循环的乘积。 c1(ak)是在置换ak的作用下不动点的个数，也就是长度为1的循环的个数(其实就是被置换ak置换过后位置不变的元素个数)。通过上述置换的变换操作后可以相等的元素属于同一个等价类。若G将[1,n]划分成L个等价类，则：

等价类个数为： $L = \frac{1}{|G|} * \sum_{i=1}^{g} c_1(a_i)$

在一个置换群G=a1,a2,a3……ak中，设C(ak)是在置换ak的循环节个数，那么用m种颜色染图这n个对象，则不同的染色方案数为：

$$L = \frac{1}{|G|} \sum_{ai \Phi G} m^{\lambda(ai)}$$

记$_L(ai)$是置换ai中长度为L的循环的数量则

$$L = \frac{1}{|G|} \sum_{ai \in G} m^{\lambda_1(ai)+\lambda_2(ai)+...+\lambda_n(ai)}$$

常见的polya原理题：

n元环手镯，考虑旋转有n种，如果没有限制就是颜色种类数$m^{gcd(i,n)}$，存在限制的话，比如说相邻位不能同色，那么dp[i][0]表示与第一位怎么怎么样，矩阵快速幂加速一下就行。

如果需要考虑反转的话，要分奇偶。

n*n方格有8种置换，正方体的12条棱涂色也有24种置换(考虑点线面)

关于爆搜问题，给出一个1*3*3的魔方，有n种颜色

```
#include <bits/stdc++.h>
using namespace std;
#define pb push_back
#define pi acos(-1)
typedef long long ll;
typedef unsigned long long ull;
const long long mod = 1000000007;
#define maxn 111111
#define maxm 11111
//三个基础置换
//右边一条棱拧一下
int a1[31] = {0, 1, 2, 18, 4, 5, 15, 7, 8, 12, 10,
              11, 9, 13, 14, 6, 16, 17, 3, 19, 20,
              22, 21, 23, 24, 25, 26, 27, 30, 29, 28};
```

```
//绕着上面中心旋转90顺时针转动
int a2[31] = {0, 7, 4, 1, 8, 5, 2, 9, 6, 3, 16,
              13, 10, 17, 14, 11, 18, 15, 12, 28, 29,
              30, 25, 26, 27, 19, 20, 21, 22, 23, 24};
//前后翻转
int a3[31] = {0, 16, 17, 18, 13, 14, 15, 10, 11, 12, 7,
               8, 9, 4, 5, 6, 1, 2, 3, 24, 23,
              22, 21, 20, 19, 27, 26, 25, 30, 29, 28};
vector <int> qun[4]; //将三种置换群放到vector中，方便后面的运算。
set<vector<int> > zh;
set<vector<int> > ::iterator it;
vector<int> mul(vector<int> a, vector<int> b) {
    vector<int> ans;
    for(int i = 0; i <= 30; i++) ans.pb(a[i]);
    for(int i = 0; i <= 30; i++) ans[i] = b[ans[i]];
    return ans;
}
//dfs出所有的置换
void dfs(vector<int> x) {
    for(int i = 1; i <= 3; i++) {
        vector<int> temp = mul(x, qun[i]);
        if(zh.count(temp) == 0) {
            zh.insert(temp);
            dfs(temp);
        }
    }
}
map<int, int> mp;
map<int, int> ::iterator it2;
int vis[31];
//找出所有置换的循环节
void findcycle() {
    for(it = zh.begin(); it != zh.end(); it++) {
        vector<int> temp = *it;
        memset(vis, 0, sizeof vis);
        int cnt = 0;
        for(int i = 1; i <= 30; i++) {
            if(vis[i] == 0) {
                cnt++;
                int tt = i;
                vis[i] = 1;
                while(!vis[temp[tt]]) {
                    tt = temp[tt];
                    vis[tt] = 1;
                }
            }
        } mp[cnt]++;
    }
}
void init() {
    for(int i = 1; i <= 3; i++) qun[i].clear();
    for(int i = 0; i <= 30; i++) qun[1].pb(a1[i]);
    for(int i = 0; i <= 30; i++) qun[2].pb(a2[i]);
    for(int i = 0; i <= 30; i++) qun[3].pb(a3[i]);
    vector<int> temp;
```

```
    for(int i = 0; i <= 30; i++) temp.pb(i);
    zh.insert(temp);
    dfs(temp);
    findcycle();
}

//快速乘法
long long qmut(long long a, long long b, long long p) {
    long long ans = 0; long long temp = a;
    while(b) {
        if(b & 1LL) ans = (ans + temp) % p;
        temp = (temp + temp) % p;
        b /= 2LL;
    } return ans;
}
//快速幂
long long qpow(long long x, long long n, long long p) {
    long long ans = 1; long long temp = x;
    while(n) {
        if(n & 1LL) ans = qmut(ans, temp, p);
        temp = qmut(temp, temp, p);
        n>>=1;
    } return ans;
}


ll qmul(ll x, ll y) {
    return (x*y - (ll)((long double)x/mod*y)*mod+mod)%mod
}

long long n, p;
int main() {
    init(); int t; scanf("%d", &t);
    long long sz = zh.size();
    while(t--) {
        scanf("%lld%lld", &n, &p);
        long long ans = 0;
        for(it2 = mp.begin(); it2 != mp.end(); it2++)
            ans = (ans + (*it2).second * qpow(n, (*it2).first, p * sz) % (p * sz)) % (p * sz);
        ans /= sz;
        printf("%lld\n", ans);
    }
    return 0;
}
```

2.5 常用组合公式

$$\sum_{k=1}^{n}(2k-1)^2 = \frac{n(4n^2-1)}{3}$$

$$\sum_{k=1}^{n}k^3 = (\frac{n(n+1)}{2})^2$$

$$\sum_{k=1}^{n}(2k-1)^3 = n^2(2n^2-1)$$

$$\sum_{k=1}^{n} k^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

$$\sum_{k=1}^{n} k^5 = \frac{n^2(n+1)^2(2n^2+2n-1)}{12}$$

$$\sum_{i=0}^{n} iC_n^i = n2^{n-1}$$

$$\sum_{k=0}^{n} (C_n^k)^2 = C_{2n}^n$$

stirling逼近$n! = \sqrt{2\pi n}(\frac{n}{e})^n$

3 Normal Math

3.1 数值计算

3.1.1 Simpson积分

```
double f(double x) { return x * x + x;//写要求辛普森积分的函数 }

double simpson(double L, double R) {//三点辛普森积分法，要求f(x)是全局函数
    double mid = (L + R) / 2.0;
    return (f(L) + 4.0 * f(mid) + f(R)) * (R - L) / 6.0;
}

double integral(double L, double R, double Eps){//自适应辛普森积分递归过程
    double mid = (L + R) / 2.0;
    double ST = simpson(L, R), SL = simpson(L, mid), SR = simpson(mid, R);
    if(fabs(SL + SR - ST) <= 15.0 * Eps) return SL + SR + (SL + SR - ST) / 15.0;//直接返回结果
    return integral(L, mid, Eps/2.0) + integral(mid, R, Eps/2.0);//对半划分区间
}
```

红书Simpson 积分，红书还拥有Romberg积分

```
template<class T>
double simpson(const T&f, double a, double int) {
    const double h = (b-a)/n;
    double ans = f(a) + f(b);
    for (int i = 1; i < n; i+=2) ans += 4*f(a+i*h);
    for (int i = 2; i < n; i+=2) ans += 2*f(a+i*h);
    return ans * h / 3;
}
```

3.1.2 拉格朗日插值

```
namespace polysum {
    #define rep(i,a,n) for (int i=a;i<n;i++)
    #define per(i,a,n) for (int i=n-1;i>=a;i--)
    const int D=2010;
    ll a[D],f[D],g[D],p[D],p1[D],p2[D],b[D],h[D][2],C[D];
    ll powmod(ll a,ll b){ll res=1;a%=mod;assert(b>=0);for(;b;b>>=1){if(b&1)res=res*a%mod;a=a*a%mod;}
        return res;}
    ll calcn(int d,ll *a,ll n) { // a[0].. a[d] a[n]
        if (n<=d) return a[n];
        p1[0]=p2[0]=1;
        rep(i,0,d+1) {
```

```
            ll t=(n-i+mod)%mod;
            p1[i+1]=p1[i]*t%mod;
        }
        rep(i,0,d+1) {
            ll t=(n-d+i+mod)%mod;
            p2[i+1]=p2[i]*t%mod;
        }
        ll ans=0;
        rep(i,0,d+1) {
            ll t=g[i]*g[d-i]%mod*p1[i]%mod*p2[d-i]%mod*a[i]%mod;
            if ((d-i)&1) ans=(ans-t+mod)%mod;
            else ans=(ans+t)%mod;
        }
        return ans;
    }
    void init(int M) {
        f[0]=f[1]=g[0]=g[1]=1;
        rep(i,2,M+5) f[i]=f[i-1]*i%mod;
        g[M+4]=powmod(f[M+4],mod-2);
        per(i,1,M+4) g[i]=g[i+1]*(i+1)%mod;
    }
    ll polysum(ll m,ll *a,ll n) { // a[0].. a[m] ∑_{i=0}^{n-1} a[i]
        ll b[D];
        for(int i=0;i<=m;i++) b[i]=a[i];
        b[m+1]=calcn(m,b,m+1);
        rep(i,1,m+2) b[i]=(b[i-1]+b[i])%mod;
        return calcn(m+1,b,n-1);
    }
    ll qpolysum(ll R,ll n,ll *a,ll m) { // a[0].. a[n] ∑_{i=0}^{m-1} a[i] * R^i
        if (R==1) return polysum(n,a,m);
        a[m+1]=calcn(m,a,m+1);
        ll r=powmod(R,mod-2),p3=0,p4=0,c,ans;
        h[0][0]=0;h[0][1]=1;
        rep(i,1,m+2) {
            h[i][0]=(h[i-1][0]+a[i-1])*r%mod;
            h[i][1]=h[i-1][1]*r%mod;
        }
        rep(i,0,m+2) {
            ll t=g[i]*g[m+1-i]%mod;
            if (i&1) p3=((p3-h[i][0]*t)%mod+mod)%mod,p4=((p4-h[i][1]*t)%mod+mod)%mod;
            else p3=(p3+h[i][0]*t)%mod,p4=(p4+h[i][1]*t)%mod;
        }
        c=powmod(p4,mod-2)*(mod-p3)%mod;
        rep(i,0,m+2) h[i][0]=(h[i][0]+h[i][1]*c)%mod;
        rep(i,0,m+2) C[i]=h[i][0];
        ans=(calcn(m,C,n)*powmod(R,n)-c)%mod;
        if (ans<0) ans+=mod;
        return ans;
    }
} // polysum::init();
```

3.2 牛顿迭代开方

使用牛顿迭代开方比二分快许多

```
double NewtonMethod(double fToBeSqrted) {
    double x = 1.0;
    while(abs(x*x-fToBeSqrted) > 1e-5) x = (x+fToBeSqrted/x)/2;
    return x;
}
```

快速求解开方的倒数(真的很快)：

```
float Q_rsqrt( float number ) {
   long i;
   float x2, y;
   const float threehalfs = 1.5F;

   x2 = number * 0.5F;
   y = number;
   i = * ( long * ) &y; // evil floating point bit level hacking
   i = 0x5f3759df - ( i >> 1 ); // what the fuck?
   y = * ( float * ) &i;
   y = y * ( threehalfs - ( x2 * y * y ) ); // 1st iteration
   // y = y * ( threehalfs - ( x2 * y * y ) ); // 2nd iteration, this can be removed

   #ifndef Q3_VM
   #ifdef __linux__
     assert( !isnan(y) ); // bk010122 - FPE?
   #endif
   #endif
   return y;
}
```

3.3 矩阵

```
struct Matrix{
    ll m[15][15];
    Matrix() { memset(m,0,sizeof m); }
};
Matrix mul(Matrix a, Matrix b){
    Matrix c;
    for(int i = 1; i <= n+2; i++){
        for(int k = 1; k <= n+2; k++)
        if(a.m[i][k]){
            for(int j = 1; j <= n+2; j++)
            if(b.m[k][j]) c.m[i][j] = (c.m[i][j]+a.m[i][k]*b.m[k][j])%mod;
        }
    } return c;
}

Matrix powmul(Matrix a, int k){
    Matrix b;
    for(int i = 1; i <= n+2; i++) b.m[i][i] = 1;
    while(k){
        if(k&1) b = mul(b,a);
        a = mul(a,a); k>>=1;
    } return b;
}
```

3.3.1 带模行列式求解

```cpp
// 高斯消元法行列式求模,复杂度O(n^3logn)。
// n为行列式大小，计算|mat| % m
const int MAXN = 200;
typedef long long ll;
int detMod(int n, int m, int mat[][MAXN]) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++) mat[i][j] %= m;
    ll ret = 1;
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++)
            while (mat[j][i] != 0) {
                ll t = mat[i][i] / mat[j][i];
                for (int k = i; k < n; k++) {
                    mat[i][k] = (mat[i][k] - mat[j][k] * t) % m;
                    int s = mat[i][k];
                    mat[i][k] = mat[j][k]; mat[j][k] = s;
                } ret = -ret;
            }
        if (mat[i][i] == 0) return 0;
        ret = ret * mat[i][i] % m;
    } if (ret < 0) ret += m;
    return (int)ret;
}

// 高斯消元法行列式求模。复杂度O(n^3 + n^2logn)
// n为行列式大小，计算|mat| % m
// 速度只比O(n^3logn)的快一些，推荐用另外那个。
const int MAXN = 200;
typedef long long ll;
int detMod(int n, int m, int mat[][MAXN]) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)mat[i][j] %= m;
    ll ret = 1;
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            ll x1 = 1, y1 = 0, x2 = 0, y2 = 1, p = i, q = j;
            if (mat[i][i] < 0) {
                x1 = -1; mat[i][i] = -mat[i][i]; ret = -ret;
            } if (mat[j][i] < 0) {
                y2 = -1; mat[j][i] = -mat[j][i]; ret = -ret;
            } while (mat[i][i] != 0 && mat[j][i] != 0) {
                if (mat[i][i] <= mat[j][i]) {
                    int t = mat[j][i] / mat[i][i];
                    mat[j][i] -= mat[i][i] * t;
                    x2 -= x1 * t;
                    y2 -= y1 * t;
                } else {
                    int t = mat[i][i] / mat[j][i];
                    mat[i][i] -= mat[j][i] * t;
                    x1 -= x2 * t;
                    y1 -= y2 * t;
                }
            }
```

```
            x1 %= m; y1 %= m;
            x2 %= m; y2 %= m;
            if (mat[i][i] == 0 && mat[j][i] != 0) {
                ret = -ret;
                p = j; q = i;
                mat[i][i] = mat[j][i];
                mat[j][i] = 0;
            }
            for (int k = i + 1; k < n; k++) {
                int s = mat[i][k], t = mat[j][k];
                mat[p][k] = (s * x1 + t * y1) % m;
                mat[q][k] = (s * x2 + t * y2) % m;
            }
        } if (mat[i][i] == 0) return 0;
        ret = ret * mat[i][i] % m;
    }
    if (ret < 0) ret += m;
    return (int)ret;
}
```

3.3.2 高斯消元

消元部分

```
void Gauss(double a[][N], int n ,int m, int &r, int &c){
    r=c=0;
    for(;r<n&&c<m;r++,c++) {
        int maxi = r;
        for(int i = r+1; i < n; i++)
            if(fabs(a[i][c])>fabs(a[maxi][c])) maxi = i;
        if(maxi!=r)
            for(int i = r; i < m+1; i++) swap(a[maxi][i],a[r][i]);
        if(!a[r][c]){ r--; continue; }
        for(int i = r+1; i < n; i++){
            if(a[i][c]){
                double t = -a[i][r]/a[r][r];
                for(int j = r; j < m+1; j++) a[i][j] += t*a[r][j];
            }
        }
    }
}
// 模方程
void Gauss(int a[][N], int n, int m, int &r, int &c){
    memset(fre, 0, sizeof fre);
    r = c = cnt = 0;
    for(; r < n && c < m ; r++,c++){
        int maxi = r;
        for(int i = r+1; i < n; i++)
            if(abs(a[maxi][c]) < abs(a[i][c])) maxi = i;
        if(maxi!=r)
            for(int i = r; i < m+1; i++) swap(a[r][i],a[maxi][i]);
        if(!a[r][c]) { fre[cnt++]=c; r --;continue; }
        for(int i = r +1; i < n; i++){
            if(a[i][c]){
                int x = abs(a[i][c]), y = abs(a[r][c]);
```

```
                int LCM = lcm(x,y);
                int tx = LCM/x, ty = LCM/y;
                if(a[r][c]*a[i][c] < 0) ty = -ty;
                for(int j =c ; j < m+1; j++)
                    a[i][j] = (a[i][j]%mod*tx%mod - a[r][j]%mod*ty%mod+mod)%mod;
            }
        }
    }
}
```

回代部分

```
void Rewind(double a[][N], double x[], int n, int m, int r, int c){
    for(int i = r-1; i >= 0; i--){
        //再不改变原方程的情况下计算解
        double t = a[i][c];
        for(int j = i+1; j < c; j++) t-=a[i][j]*x[j];
        if(a[i][i]) x[i] = t/a[i][i];
    }
}
//模方程
int Rewind(int a[][N], int x[N], int n, int m, int r, int c){
    for(int i = r ; i < n; i++) if(a[i][c]) return -1; // 无解
    if(cnt || r<m) return 1; //多解
    //求解
    memset(x, 0, sizeof x);
    for(int i = r-1; i >= 0; i--) {
        int temp = a[i][c] % mod;
        for(int j = i+1; j < c; j++) temp -= a[i][j]*x[j];
        //求逆元的欧几里得
        int d,xx,yy;
        e_gcd(a[i][i], mod, d, xx, yy);
        xx = (xx%mod+mod)%mod;
        x[i] = ((temp%mod*xx%mod)%mod+mod)%mod;
        if(x[i]<3) x[i] += mod;
    } return 0;
}
```

3.3.3 异或方程组

```
int xor_guass(int m, int n) {// A是异或方程组系数矩阵返回秩
    int i = 0, j = 0, k, r, u;
    while(i < m && j < n){//当前正在处理第i个方程,第j个变量
        r = i;
        for(int k = i; k < m; k++) if(A[k][j]){r = k; break;}
        if(A[r][j]){
            if(r != i) for(k = 0; k <= n; k++) swap(A[r][k], A[i][k]);
            // 消元完成之后第i行的第一个非0列是第j列,且第u>i行的第j列全是0
            for(u = i + 1; u < m; u++) if(A[u][j])
                for(k = i; k <= n; k++) A[u][k] ^= A[i][k];
            i++;
        } j++;
    } return i; // 返回矩阵的秩即, 非自由变元的个数
}
```

3.3.4 线性基

```cpp
struct L_B{
    long long d[61],p[61];
    int cnt;
    L_B() {
        memset(d,0,sizeof(d));
        memset(p,0,sizeof(p));
        cnt=0;
    }
    //插入
    bool insert(long long val) {
        for (int i=60;i>=0;i--)
        if (val&(1LL<<i)) {
            if (!d[i]) {
                d[i]=val;
                break;
            } val^=d[i];
        } return val>0;
    }
    //查询最大值
    long long query_max() {
        long long ret=0;
        for (int i=60;i>=0;i--) if ((ret^d[i])>ret) ret^=d[i];
        return ret;
    }
    //查询最小值
    long long query_min() {
        for (int i=0;i<=60;i++) if (d[i]) return d[i];
        return 0;
    }
    //查找第k小时的重建(将线性基改造成每一位都相互独立)
    void rebuild() {
        for (int i=60;i>=0;i--) for (int j=i-1;j>=0;j--)
            if (d[i]&(1LL<<j)) d[i]^=d[j];
        for (int i=0;i<=60;i++) if (d[i]) p[cnt++]=d[i];
    }
    long long kthquery(long long k) {
        int ret=0;
        if (k>=(1LL<<cnt)) return -1;
        for (int i=60;i>=0;i--) if (k&(1LL<<i)) ret^=p[i];
        return ret;
    }
}
//合并
L_B merge(const L_B &n1,const L_B &n2) {
    L_B ret=n1;
    for (int i=60;i>=0;i--) if (n2.d[i]) ret.insert(n1.d[i]);
    return ret;
}
```

3.4 大整数

```cpp
struct BigInteger {
```

```
    int len, s[SIZE + 5];

    BigInteger () {
        memset(s, 0, sizeof(s));
        len = 1;
    }
    BigInteger operator = (const char *num) { //字符串赋值
        memset(s, 0, sizeof(s));
        len = strlen(num);
        for(int i = 0; i < len; i++) s[i] = num[len - i - 1] - '0';
        return *this;
    }
    BigInteger operator = (const int num) { //int 赋值
        memset(s, 0, sizeof(s));
        char ss[SIZE + 5];
        sprintf(ss, "%d", num);
        *this = ss;
        return *this;
    }
    BigInteger (int num) {
        *this = num;
    }
    BigInteger (char* num) {
        *this = num;
    }
    string str() const { //转化成string
        string res = "";
        for(int i = 0; i < len; i++) res = (char)(s[i] + '0') + res;
        if(res == "") res = "0";
        return res;
    }
    BigInteger clean() {
        while(len > 1 && !s[len - 1]) len--;
        return *this;
    }

    BigInteger operator + (const BigInteger& b) const {
        BigInteger c;
        c.len = 0;
        for(int i = 0, g = 0; g || i < max(len, b.len); i++) {
            int x = g;
            if(i < len) x += s[i];
            if(i < b.len) x += b.s[i];
            c.s[c.len++] = x % 10;
            g = x / 10;
        }
        return c.clean();
    }

    BigInteger operator - (const BigInteger& b) {
        BigInteger c;
        c.len = 0;
        for(int i = 0, g = 0; i < len; i++) {
            int x = s[i] - g;
            if(i < b.len) x -= b.s[i];
```

```
57          if(x >= 0) g = 0;
58          else {
59              g = 1;
60              x += 10;
61          }
62          c.s[c.len++] = x;
63      }
64      return c.clean();
65  }
66
67  BigInteger operator * (const int num) const {
68      int c = 0, t;
69      BigInteger pro;
70      for(int i = 0; i < len; ++i) {
71          t = s[i] * num + c;
72          pro.s[i] = t % 10;
73          c = t / 10;
74      }
75      pro.len = len;
76      while(c != 0) {
77          pro.s[pro.len++] = c % 10;
78          c /= 10;
79      }
80      return pro.clean();
81  }
82
83  BigInteger operator * (const BigInteger& b) const {
84      BigInteger c;
85      for(int i = 0; i < len; i++) {
86          for(int j = 0; j < b.len; j++) {
87              c.s[i + j] += s[i] * b.s[j];
88              c.s[i + j + 1] += c.s[i + j] / 10;
89              c.s[i + j] %= 10;
90          }
91      }
92      c.len = len + b.len + 1;
93      return c.clean();
94  }
95
96  BigInteger operator / (const BigInteger &b) const {
97      BigInteger c, f;
98      for(int i = len - 1; i >= 0; --i) {
99          f = f * 10;
100         f.s[0] = s[i];
101         while(f >= b) {
102             f = f - b;
103             ++c.s[i];
104         }
105     }
106     c.len = len;
107     return c.clean();
108 }
109 //高精度取模
110 BigInteger operator % (const BigInteger &b) const{
111     BigInteger r;
```

```
        for(int i = len - 1; i >= 0; --i) {
            r = r * 10;
            r.s[0] = s[i];
            while(r >= b) r = r - b;
        }
        r.len = len;
        return r.clean();
    }

    bool operator < (const BigInteger& b) const {
        if(len != b.len) return len < b.len;
        for(int i = len - 1; i >= 0; i--)
            if(s[i] != b.s[i]) return s[i] < b.s[i];
        return false;
    }
    bool operator > (const BigInteger& b) const {
        return b < *this;
    }
    bool operator <= (const BigInteger& b) const {
        return !(b < *this);
    }
    bool operator == (const BigInteger& b) const {
        return !(b < *this) && !(*this < b);
    }
    bool operator != (const BigInteger &b) const {
        return !(*this == b);
    }
    bool operator >= (const BigInteger &b) const {
        return *this > b || *this == b;
    }
    friend istream & operator >> (istream &in, BigInteger& x) {
        string s;
        in >> s;
        x = s.c_str();
        return in;
    }
    friend ostream & operator << (ostream &out, const BigInteger& x) {
        out << x.str();
        return out;
    }
};
```

分数类见红书

3.5 卷积

3.5.1 FFT

```
namespace FFT {
    #define rep(i,a,b) for(int i=(a);i<=(b);i++)
    const double pi=acos(-1);
    const int maxn=1<<19;
    struct cp {
        double a,b;
        cp(){}
        cp(double _x,double _y){a=_x,b=_y;}
```

```
        cp operator +(const cp &o)const{return (cp){a+o.a,b+o.b};}
        cp operator -(const cp &o)const{return (cp){a-o.a,b-o.b};}
        cp operator *(const cp &o)const{return (cp){a*o.a-b*o.b,b*o.a+a*o.b};}
        cp operator *(const double &o)const{return (cp){a*o,b*o};}
        cp operator !()const{return (cp){a,-b};}
    } x[maxn],y[maxn],z[maxn],w[maxn];
    void fft(cp x[],int k,int v) {
        int i,j,l;
        for(i=0,j=0;i<k;i++) {
            if(i>j)swap(x[i],x[j]);
            for(l=k>>1;(j^=l)<l;l>>=1);
        } w[0]=(cp){1,0};
        for(i=2;i<=k;i<<=1) {
            cp g=(cp){cos(2*pi/i),(v?-1:1)*sin(2*pi/i)};
            for(j=(i>>1);j>=0;j-=2)w[j]=w[j>>1];
            for(j=1;j<i>>1;j+=2)w[j]=w[j-1]*g;
            for(j=0;j<k;j+=i) {
                cp *a=x+j,*b=a+(i>>1);
                for(l=0;l<i>>1;l++) {
                    cp o=b[l]*w[l];
                    b[l]=a[l]-o;
                    a[l]=a[l]+o;
                }
            }
        } if(v)for(i=0;i<k;i++)x[i]=(cp){x[i].a/k,x[i].b/k};
    }
    void mul(ll *a,ll *b,ll *c,int l1,int l2) {
        if(l1<128&&l2<128) {
            rep(i,0,l1+l2)a[i]=0;
            rep(i,0,l1)rep(j,0,l2)a[i+j]+=b[i]*c[j];
            return;
        } int K;
        for(K=1;K<=l1+l2;K<<=1);
        rep(i,0,l1)x[i]=cp(b[i],0);
        rep(i,0,l2)y[i]=cp(c[i],0);
        rep(i,l1+1,K)x[i]=cp(0,0);
        rep(i,l2+1,K)y[i]=cp(0,0);
        fft(x,K,0);fft(y,K,0);
        rep(i,0,K)z[i]=x[i]*y[i];
        fft(z,K,1);
        rep(i,0,l1+l2)a[i]=(ll)(z[i].a+0.5);
    }
};
```

典型例题：CodeForces - 528D

题目大意：给出一个母串和一个模板串，求模板串在母串中的匹配次数。匹配时，如果用s[i]匹配t[j]，那么只要s[i-k]-s[i+k]中有字母与t[j]相同即可算作匹配成功。其中s[i]表示母串的第i位，t[j]表示模板串的第j位。

我们分别考虑每个字符，那么s[i+p] = t[i] 那么我们反转母串就是卷积。a[j+i] - b[i] 反转 -¿ a[n-1-j-i] - b[i] 那么下标和就是 n - 1 - j

我们取卷积后前n个元素就是匹配的个数，所有字符计算一下求和判断是不是—T—即可。

如果字符匹配不是一一配对，也就是说可以 a[j+i+p] - b[i] (p ¡= k) 那么我们只要用树状数组取标记是否有贡献即可。

```
#include <bits/stdc++.h>

using namespace std;
const int N = 1e6+9;
```

```
5
6  namespace fft{
7  #define MAX 500000
8      struct complx{
9          double real, img;
10         inline complx(){ real = img = 0.0; }
11         inline complx(double x){ real = x, img = 0.0; }
12         inline complx(double x, double y){ real = x, img = y; }
13         inline void operator += (complx &other){ real += other.real, img += other.img; }
14         inline void operator -= (complx &other){ real -= other.real, img -= other.img; }
15         inline complx operator + (complx &other){ return complx(real + other.real, img + other.img); }
16         inline complx operator - (complx &other){ return complx(real - other.real, img - other.img); }
17         inline complx operator * (complx& other){ return complx((real * other.real) - (img * other.img
               ), (real * other.img) + (img * other.real)); }
18     };
19     int rev[MAX];
20     complx P[MAX >> 1], P1[MAX], P2[MAX];
21     void FFT(complx *ar, int n, int inv){
22         int i, j, k, l, len, len2;
23         const double p = 4.0 * inv * acos(0.0);
24         for (i = 0, l = __builtin_ctz(n) - 1; i < n; i++){
25             rev[i] = rev[i >> 1] >> 1 |((i & 1) << l);
26             if(i < rev[i]) swap(ar[i], ar[rev[i]]);
27         }
28         for (len = 2; len <= n; len <<= 1){
29             len2 = len >> 1;
30             double theta = p / len;
31             complx mul(cos(theta), sin(theta));
32             for (i = 1, P[0] = complx(1, 0); i < len2; i++) P[i] = (P[i - 1] * mul);
33             for (i = 0; i < n; i += len){
34                 complx t, *x = ar + i, *y = ar + i + len2, *l = ar + i + len2, *z = P;
35                 for (; x != l; x++, y++, z++){
36                     t = (*y) * (*z), *y = *x - t;
37                     *x += t;
38                 }
39             }
40         }
41         if (inv == -1){
42             double tmp = 1.0 / n;
43             for (i = 0; i < n; i++) ar[i].real *= tmp;
44         }
45     }
46     void cmplx_conv(int n, complx* A, complx* B){
47         int m = 1 << (32 - __builtin_clz(n) - (__builtin_popcount(n) == 1));
48         for (int i = n; i < m; i++) A[i] = B[i] = complx(0);
49         FFT(A, m, 1), FFT(B, m, 1);
50         for (int i = 0; i < m; i++) A[i] = A[i] * B[i];
51         FFT(A, m, -1);
52     }
53     void conv(int n, int* A, int* B){
54         for (int i = 0; i < n; i++) P1[i] = complx(A[i], 0.0), P2[i] = complx(B[i], 0.0);
55         cmplx_conv(n, P1, P2);
56         for (int i = 0; i < n; i++) A[i] = floor(P1[i].real + 0.5);
57     }
58 }
```

```

char S[N], T[N];
const char DNA[] = "ACGT";
int n, m, k, a[N], b[N], tree[N], cont[4][N];
inline int lb(int i) {return i&(-i);}
void update(int p, int v) { for (int i = p; i <= n; i += lb(i)) { tree[i] += v; } }

void update(int l, int r, int v) {
    if(l > r) return ;
    update(l, v); update(r + 1, -v);
}

int query(int p) {
    int ans = 0;
    for (int i = p; i; i -= lb(i)) { ans += tree[i]; }
    return ans;
}

int main() {
    scanf("%d%d%d%s%s", &n, &m, &k, S, T);
    for (int i = 0; i < 4; i++) {
        memset(a, 0, sizeof a); memset(b, 0, sizeof b); memset(tree, 0, sizeof tree);
        for (int j = 0; j < m; j++) a[j] = (T[j] == DNA[i]);
        for (int j = 0; j < n; j++)
            if (S[j] == DNA[i])
                update(max(0, j - k) + 1, min(j+k+1, n), 1);
        for (int j = 0; j < n; j++) if(query(j+1)) b[j] = 1;
        reverse(b, b+n); fft::conv(n, a, b);
        for (int j = 0; j < n; j++) cont[i][j] = a[n - j - 1];
    } int ans = 0;
    for (int i = 0; (i+m) <= n; i++)
        if(cont[0][i] + cont[1][i] + cont[2][i] + cont[3][i] == m) ans ++;
    return printf("%d\n", ans), 0;
}
```

3.5.2 FWT

```
void FWT(ll *a,int opt) {
    for(int i=1;i<n;i<<=1)
        for(int p=i<<1,j=0;j<n;j+=p)
            for(int k=0;k<i;++k)
            {
                ll X=a[j+k],Y=a[i+j+k];
                a[j+k]=(X+Y)%mod;a[i+j+k]=(X+mod-Y)%mod;
                if(opt==-1)a[j+k]=1ll*a[j+k]*inv2%mod,a[i+j+k]=1ll*a[i+j+k]*inv2%mod;
            }
}
```

典型例题:51nod 多校第九场A A[i][j] = a[i xor j]

For example, when n = 4, the equations look like

A[0][0]*x[0] + A[0][1]*x[1] + A[0][2]*x[2] + A[0][3]*x[3] = b[0] (mod p)

A[1][0]*x[0] + A[1][1]*x[1] + A[1][2]*x[2] + A[1][3]*x[3] = b[1] (mod p)

A[2][0]*x[0] + A[2][1]*x[1] + A[2][2]*x[2] + A[2][3]*x[3] = b[2] (mod p)

A[3][0]*x[0] + A[3][1]*x[1] + A[3][2]*x[2] + A[3][3]*x[3] = b[3] (mod p)

```
int main() {
    scanf("%d", &n); inv2 = qp(2);
    for (int i = 0; i < n; i++) {
        scanf("%lld", &a[i]);
    } for (int i = 0; i < n; i++) {
        scanf("%lld", &b[i]);
    } FWT(a, 1); FWT(b, 1);
    for (int i = 0; i < n; i++) {
        b[i] = (b[i]*qp(a[i])) % mod;
    } FWT(b, -1);
    for (int i = 0; i < n; i++) printf("%lld\n", gm(b[i]));
}
```

3.6 01分数规划

给出 n对 a_i, b_i, 求最大的$\sum \frac{a_i}{b_i}$,最多可以删除k对ai,bi, 除了二分以外，可以使用01分数规划迭代,可以解决最优比例生成树

```
const int maxn = 1e3+10;
using namespace std;
int n,k;
struct node{ int a,b; } val[maxn];
double p;
int cmp(node x, node y){ return x.a - (p * x.b) > y.a - (p * y.b); }

bool check(double mid){
    p = mid;
    sort(val, val + n, cmp);
    double total_a = 0,total_b = 0;
    for(int i = 0; i < n - k; i++){
        total_a += val[i].a;
        total_b += val[i].b;
    } return (total_a/total_b) > mid;
}

int main(){
    while(~scanf("%d%d",&n,&k) && n){
        for(int i = 0; i < n; i++) scanf("%d",&val[i].a);
        for(int i = 0; i < n; i++) scanf("%d",&val[i].b);
        double l = 0, r = 1;
        double mid;
        for(int i = 0; i < 200; i++){
            mid = (l+r)/2;
            if(check(mid)) l = mid;
            else r = mid;
        } printf("%d\n",(int)round(mid*100));
    } return 0;
}
```

3.7 三分

```
for(int i = 0; i < 200; i++){
    mid1 = l + (r-l)/3;
    mid2 = r - (r-l)/3;
```

```
    if(cal(mid1) < cal(mid2)) r = mid2;
    else l = mid1;
}
```

3.7.1 黄金比例三分

```
const double phi=(sqrt(5.0)-1)/2;
double l = 0, r = 2*PI, mid1, mid2,f1=0,f2=100;
int left = 0,right = 0;
for(int i = 0; i < 35 && fabs(l-r)> 1e-5; i++){
  if(!left)mid1 = r - (r-l)*phi, f1 = cal(mid1);
  if(!right)mid2 = l + (r-l)*phi, f2 = cal(mid2);
  if(f1 < f2) r = mid2; mid2 = mid1;left = 0; right = 1; f2 = f1;
  else l = mid1; mid1 = mid2; right = 0; left = 1; f1 = f2;
}
```

3.8 模拟退火

3.8.1 费马点

给n个点，找出一个点，使这个点到其他所有点的距离之和最小，也就是求费马点。

```
#include <bits/stdc++.h>
#define N 1005
#define eps 1e-8 //搜索停止条件阀值
#define INF 1e99
#define delta 0.98 //温度下降速度
#define T 100 //初始温度

using namespace std;

int dx[4] = {0, 0, -1, 1};
int dy[4] = {-1, 1, 0, 0}; //上下左右四个方向

struct Point { double x, y; }p[N];

double dist(Point A, Point B) {
    return sqrt((A.x - B.x) * (A.x - B.x) + (A.y - B.y) * (A.y - B.y));
}

double GetSum(Point p[], int n, Point t) {
    double ans = 0;
    while(n--) ans += dist(p[n], t);
    return ans;
}

double Search(Point p[], int n) {
    Point s = p[0]; //随机初始化一个点开始搜索
    double t = T; //初始化温度
    double ans = INF; //初始答案值
    while(t > eps) {
        bool flag = 1;
        while(flag) {
            flag = 0;
            for(int i = 0; i < 4; i++) { //上下左右四个方向
```

```
                Point z;
                z.x = s.x + dx[i] * t;
                z.y = s.y + dy[i] * t;
                double tp = GetSum(p, n, z);
                if(ans > tp) { ans = tp; s = z; flag = 1; }
            }
        } t *= delta;
    } return ans;
}

int main() {
    int n;
    while(scanf("%d", &n) != EOF) {
        for(int i = 0; i < n; i++) scanf("%f %f", &p[i].x, &p[i].y);
        printf("%.0lf\n", Search(p, n));
    } return 0;
}
```

3.8.2 距离直线簇最近点

平面上给定n条线段，找出一个点，使这个点到这n条线段的距离和最小。

```
struct Point { double x, y; }s[N], t[N];
double cross(Point A, Point B, Point C) { return (B.x - A.x) * (C.y - A.y) - (B.y - A.y) * (C.x - A.x); }
double multi(Point A, Point B, Point C) { return (B.x - A.x) * (C.x - A.x) + (B.y - A.y) * (C.y - A.y); }
double dist(Point A, Point B) { return sqrt((A.x - B.x) * (A.x - B.x) + (A.y - B.y) * (A.y - B.y)); }

double GetDist(Point A, Point B, Point C) {
    if(dist(A, B) < eps) return dist(B, C);
    if(multi(A, B, C) < -eps) return dist(A, C);
    if(multi(B, A, C) < -eps) return dist(B, C);
    return fabs(cross(A, B, C) / dist(A, B));
}

double GetSum(Point s[], Point t[], int n, Point o) {
    double ans = 0;
    while(n--) ans += GetDist(s[n], t[n], o);
    return ans;
}
```

3.8.3 包住所有点的球

```
double Search(Point p[], int n) {
    Point s = p[0];
    double t = T;
    double ans = INF;
    while(t > eps) {
        int k = 0;
        for(int i = 0; i < n; i++)
            if(dist(s, p[i]) > dist(s, p[k])) k = i;
        double d = dist(s, p[k]);
        ans = min(ans, d);
        s.x += (p[k].x - s.x) / d * t;
```

```
        s.y += (p[k].y - s.y) / d * t;
        s.z += (p[k].z - s.z) / d * t;
        t *= delta;
    } return ans;
}
```

3.8.4 函数最值问题

```
int Judge(double x, double y) {
    if(fabs(x - y) < eps) return 0;
    return x < y ? -1 : 1;
}

double Random() {
    double x = rand() * 1.0 / RAND_MAX;
    if(rand() & 1) x *= -1;
    return x;
}

void Init() {
    for(int i = 0; i < ITERS; i++) x[i] = fabs(Random()) * 100;
}

double SA(double y) {
    double ans = INF;
    double t = T;
    while(t > eps) {
        for(int i = 0; i < ITERS; i++) {
            double tmp = F(x[i], y);
            for(int j = 0; j < ITERS; j++) {
                double _x = x[i] + Random() * t;
                if(Judge(_x, 0) >= 0 && Judge(_x, 100) <= 0) {
                    double f = F(_x, y);
                    if(tmp > f) x[i] = _x;
                }
            }
        } t *= delta;
    }
    for(int i = 0; i < ITERS; i++) ans = min(ans, F(x[i], y));
    return ans;
}

srand(time(NULL));
Init();
```

4 Graph

4.1 存图

二维矩阵

```
int mp[N][N];
void read(){
    for (int i = 1; i <= n; i++)
```

```
        for (int j = 1; j <= n; j++) scanf("%d", mp[i][j]);
}
```

邻接矩阵

```
vector<int> mp[N];
void read(){
    for (int i = 0; i < m; i++) {
        scanf("%d%d", &u, &v);
        mp[u].push_back(v); mp[v].push_back(u);
    }
}
```

链式前向星：

```
int head[N], cnt;
struct Edge{ int v, nxt; }e[N<<1];

void ad(int u, int v) { e[cnt]={v, head[u]}; head[u] = cnt++; }
void add(int u, int v) { ad(u, v); ad(v, u); }
void init(){ cnt = 0; memset(head, -1, size of head); }

void read(){
    for (int i = 0; i < m; i++) { scanf("%d%d", &u, &v); add(u, v); }
}
```

4.2 最短路

4.2.1 Dijkstra

```
void Dijkstra(int s){
    memset(d, 0x3f, sizeof d);
    memset(vis, 0, sizeof vis);
    priority_queue<pair<int, int> > q; d[s] = 0;
    q.push(make_pair(-d[s], s));
    while(!q.empty()) {
        pair<int, int> tp = q.top(); q.pop();
        int u = tp.second;
        if(vis[u]) continue;
        vis[u] = 1;
        for (int i = head[u]; ~i; i = e[i].nxt){
            int v = e[i].v;
            if(d[u] + e[i].w < d[v] && !vis[v]) {
                d[v] = d[u] + e[i].w;
                q.push(make_pair(-(d[v]), v));
            }
        }
    }
}
```

4.2.2 bfs找最短路

```
void bfs(){
    memset(vis, 0, sizeof vis);
    memset(d, -1, sizeof d); d[s] = 0;
    queue<int> q; q.push(s);
```

```
    while(!q.empty()){
        int u = q.front(); q.pop();
        for(int i = head[u]; ~i; i = e[i].nxt){
            int v = e[i].v;
            if(d[v]==-1){ d[v] = d[u] + 1; q.push(v); }
        }
    }
}
```

4.2.3 spfa判负环

```
bool spfa(){
    memset(vis, 0, sizeof vis); memset(d, 0, sizeof d);
    d[s] = 0; vis[s] = 1; deque<int> q; q.push_back(s);
    while(!q.empty()) {
        int u = q.front(); q.pop_front(); vis[u] = 0;
        for (int i = head[u]; ~i; i = e[i].nxt) {
            int v = e[i].v;
            if(d[v] > d[u] + e[i].c) {
                cnt[v] ++;
                if(cnt[v] > n) return 1;
                d[v] = d[u] + e[i].c;
                if(!vis[v]) {
                    vis[v] = 1;
                    if(!q.empty() && d[v] < d[q.front()]) q.push_front(v);
                    else q.push_back(v);
                }
            }
        }
    } return 0;
}
```

4.2.4 flyod

```
for (int k = 1; k <= n; k++) for (int i = 1; i <= n; i++)
for (int j = 1; j <= n; j++)
    mp[i][j] = min(mp[i][j], mp[i][k] + mp[k][j]);

bitset<N> mp[N]; // 声明二维矩阵0表示不连通，1联通
for (int k = 1; k <= n; ++k) for (int i = 1; i <= n; ++i)
    if (e[i][k]) e[i] |= e[k];
```

4.3 最小生成树

prim:

```
inline void prim(){
    int di,v;
    for(int i = 0; i < n; i++) { d[i] = mp[0][i]; vis[i] = 0; }
    for(int i = 1; i <= n; i++) {
        di = inf;
        for(int j = 0; j < n; j++)
            if(!vis[j] && di > d[j]){ v = j; di = d[j]; }
        vis[v] = 1;
```

```
        for(int j = 0; j < n; j++)
            if(!vis[j] && d[j] > mp[v][j]) d[j] = mp[v][j];
    }
}
```

kruskal

```
void init(){ for(int i = 1; i <maxn; i++) pre[i] = i; }
int Find(int x){ return x==fa[x]?x:fa[x]=Find(fa[x]); }

int join(int x, int y){
    int fx = Find(x),fy = Find(y);
    if(fx != fy){ fa[fy] = fx; return 1; }
    return 0;
}

void kruskal(){
    sort(e,e+m,cmp);
    int cont = 0,sum = 0;
    for(int i = 0; i < m; i++){
        if(join(e[i].u,e[i].v)){
            cont++;
            sum+=e[i].w;
        } if(cont == n-1) break;
    } printf("%d\n",sum);
}
```

4.4 匹配

二分图匹配

```
bool Find(int x){
    for(int i = 1; i <= c; i++) {
        if(mp[x][i]&&!used[i]) {
            used[i] = 1;
            if(!f[i]||Find(f[i])) {
                f[i] = x; return 1;
            }
        }
    } return 0;
}
for(int i = 1; i <= r; i++){
    memset(used, 0, sizeof used);
    if(Find(i)) ans++;
}
```

```
bool Find(int s) {
    for(int i = head[s]; ~i; i = e[i].nxt) {
        int v = e[i].v;
        if(!used[v]&&dy[v]==dx[s]+1){
            used[v] = 1;
            if(my[v]!=-1&&dy[v]==dss)continue;
            if(my[v]==-1||Find(my[v])){ my[v] = s; mx[s] = v; return 1; }
        }
    } return 0;
}
```

```
inline void init() { memset(head, -1, sizeof head); memset(f, 0, sizeof f); tot = 0; }
inline void add(int u, int v) { e[tot] = {v, head[u]}; head[u] = tot++; }

inline int dis(int i, int u, int v) {
    return (x[i]-u)*(x[i]-u)+(y[i]-v)*(y[i]-v);
}
const int inf = 0x3f3f3f3f;
bool searchP() {
    queue<int> q; dss = inf;
    memset(dy, -1, sizeof dy); memset(dx, -1, sizeof dx);
    for(int i = 1; i <= n; i++){
        if(mx[i] == -1) { q.push(i); dx[i] = 0; }
    } while(!q.empty()) {
        int u = q.front(); q.pop();
        if(dx[u]>dss) break;
        for (int i = head[u]; ~i; i = e[i].nxt) {
            int v = e[i].v;
            if(dy[v] == -1) {
                dy[v] = dx[u] + 1;
                if(my[v]==-1) dss=dy[v];
                else {
                    dx[my[v]] = dy[v]+1;
                    q.push(my[v]);
                }
            }
        }
    } return dss!=inf;
}

int slove(){
    int ans = 0;
    memset(mx, -1, sizeof mx); memset(my, -1, sizeof my);
    while(searchP()) {
        memset(used, 0, sizeof used);
        for(int i = 1; i <= n; i++)
            if(mx[i] == -1 && Find(i)) ans++;
    } return ans;
}
```

4.5 强连通分量

其实并不怎么会,割点割桥,双联通

```
void tarjan(int u){
    int v;
    low[u] = dfn[u] = ++idx;
    top++[stck] = u;
    inStack[u] = 1;
    for (int i = head[u]; ~i; i = e[i].nxt) {
        v = e[i].v;
        if(!dfn[v]) {
            tarjan(v);
            if(low[u] > low[v])
                low[u] = low[v];
            } else if(inStack[v] && low[u] > dfn[v]) {
```

```
                low[u] = dfn[v];
            }
    }
    if(low[u] == dfn[u]) {
        scc++;
        do {
            v = stck[--top];
            be[v] = scc;
            inStack[v] = 0;
        } while(v!=u);
    }
}
```

4.6 网络流

基本概念

求解可行流: 给定一个网络流图, 初始时每个节点不一定平衡 (每个节点可以有盈余或不足), 每条边的流量可以有上下界, 每条边的当前流量可以不满足上下界约束. 可行流求解中一般没有源和汇的概念, 算法的目的是寻找一个可以使所有节点都能平衡, 所有边都能满足流量约束的方案, 同时可能附加有最小费用的条件 (最小费用可行流).

求解最大流: 给定一个网络流图, 其中有两个特殊的节点称为源和汇. 除源和汇之外, 给定的每个节点一定平衡. 源可以产生无限大的流量, 汇可以吸收无限大的流量. 标准的最大流模型, 初始解一定是可行的 (例如, 所有边流量均为零), 因此边上不能有下界. 算法的目的是寻找一个从源到汇流量最大的方案, 同时不破坏可行约束, 并可能附加有最小费用的条件 (最小费用最大流).

扩展的最大流: 在有上下界或有节点盈余的网络流图中求解最大流. 实际上包括两部分, 先是消除下界, 消除盈余, 可能还需要消除不满足最优条件的流量 (最小费用流), 找到一个可行流, 再进一步得到最大流. 因此这里我们的转化似乎是从最大流转化为可行流再变回最大流, 但其实质是将一个过程 (扩展的最大流) 变为了两个过程 (可行流 + 最大流).

4.6.1 最大流

关于网络流的问题，其重点在于如何建图，而对于算法细节，其实不太重要，一般常用Dinic，建立分层图来增广，实现起来比较简单，也有isap的，其复杂度上线都是O(n^2m), 不过实际上isap的算法时间非常占优势，不过没有Dinic容易实现。

Dinic

```
bool bfs(){ // 建立分层图并判别能否到达汇点t.
    memset(d, 0, sizeof d); d[s] = 1;
    queue<int> q; q.push(s);
    while(!q.empty()){
        int u = q.front(); q.pop();
        for(int i = head[u]; ~i; i = e[i].nxt){
            int v = e[i].v;
            if(!d[v] && e[i].w>0){
                d[v] = d[u] + 1; q.push(v);
                if(v == t) return 1;
            }
        }
    } return 0;
}

int dfs(int u, int w){ // 增广
    if(u == t || w == 0) return w;
    int dlt = w; // 剩余流量
    for(int &i = cur[u]; i != -1; i = e[i].nxt){
        int v = e[i].v;
        if(d[v] == d[u] + 1 && e[i].w > 0){
            int x = dfs(v, min(dlt, e[i].w));
```

```
            e[i].w -= x; e[i^1].w += x;
            dlt -= x; if(!dlt)return w;
        }
    } return w-dlt;
}

int dinic(){
    int sum = 0;
    while(bfs()) { // 最多n次增广
        for(int i = 0; i <= n; i++) cur[i] = head[i];
        sum += dfs(s, inf);
    } return sum;
}
```

问题1:项目发展规划(Develop)

Macrosoft® 公司准备制定一份未来的发展规划。公司各部门提出的发展项目汇总成了一张规划表，该表包含了许多项目。对于每个项目，规划表中都给出了它所需的投资或预计的盈利。由于某些项目的实施必须依赖于其它项目的开发成果，所以如果要实施这个项目的话，它所依赖的项目也是必不可少的。现在请你担任Macrosoft公司的总裁，从这些项目中挑选出一部分，使你的公司获得最大的净利润。

答：建立N顶点代表N个项目，另外增加源s与汇t。若项目i必须依赖于项目j，则从顶点i向顶点j引一条容量为无穷大的弧。对于每个项目i，若它的预算C_i为正（盈利），则从源s向顶点i引一条容量为C_i的边；若它的预算C_i为负（投资），则从顶点i向汇t引一条容量为-C_i的边。求这个网络的最小割（S, T），设其容量C(S,T)＝F。设R为所有盈利项目的预算之和（净利润上界），那么R-F就是最大净利润；S中的顶点就表示最优方案所选择的项目。

根据最大流最小割定理，网络的最小割可以通过最大流的方法求得.

最大权闭合子图

所谓闭合子图就是给定一个有向图，从中选择一些点组成一个点集V。对于V中任意一个点，其后续节点都仍然在V中。

所以求解闭合子图中所有点权值最大的问题就是最大权闭合子图问题。

结论：最大权闭合子图 = 正点和-最小割。

最小割可以利用最大流来解决. 设置源点与值为正的点建边，边权为点权，汇点与值为负的点建边，边权为点权绝对值，其他边的权值均为无穷，求得的最大流即最小割。

CF 498C

题意：给出一组数，然后给出他们之间的边，构成一张由奇数点和偶数点组成的二分图，然后存在边的数可以除以他们的公约数，问最多的操作次数。

我们对于每个数都去计算他所有的因数和个数，我们可以将一个奇数点和偶数点都拆成他和他的因子们，那么每次同除左右的两个数也就是流过了具有相同因数的两个数所组成的边，所以最多的操作个数就可以跑一次最大流。具体建图方式见代码。

```
#include <bits/stdc++.h>
const int N = 1e5+9;
const int inf = 1e9+9;
using namespace std;

int head[N], prime[N], d[N], a[N], idx[111], sum[111];
int n, m, tot, cnt, s, t;
bool vis[N];
/** -------------------------- 最大流 ----------------------------------------- **/
struct Edge{int u,v, w, nxt;}e[N<<1];
void ad(int u, int v, int w){e[cnt]={u,v,w,head[u]};head[u]=cnt++;}
void add(int u, int v, int w){ad(u, v, w); ad(v, u, 0);}

bool bfs(){
    for (int i = 0; i <= t; i++) d[i] = 0;
    d[s] = 1; queue<int> q; q.push(s);
    while(!q.empty()){
        int u = q.front(); q.pop();
```

```
        for (int i = head[u]; ~i; i = e[i].nxt) {
            int v = e[i].v;
            if(!d[v] && e[i].w > 0) {
                d[v] = d[u] + 1;
                if(v == t) return 1;
                q.push(v);
            }
        }
    } return 0;
}

int dfs(int u, int w) {
    if(u == t || w == 0) return w;
    int dlt=w, x;
    for (int i = head[u]; ~i; i = e[i].nxt) {
        int v = e[i].v;
        if(e[i].w > 0 && d[v] == d[u] + 1) {
            x = dfs(v, min(dlt, e[i].w));
            e[i].w -= x; e[i^1].w += x;
            dlt -= x; if(!dlt) return w;
        }
    } return w - dlt;
}

int dinic(){
    int sum = 0;
    while(bfs()) {
        sum += dfs(s, inf);
    } return sum;
}
/** --------------------------- 最大流 ----------------------------------------- **/
void init(){
    cnt = 0; // 先使用欧拉筛预处理1e5以下的质数
    for (int i = 2; i < N; i++) {
        if(!vis[i]) prime[tot++] = i;
        for (int j = 0; j < tot; j++) {
            if(i*prime[j] >= N) break;
            vis[i*prime[j]] = 1;
            if(i%prime[j]==0) break;
        }
    } memset(head, -1, sizeof head); // 链式前向星初始化
}
vector<pair<int, int>> ph[111]; // 用来存储每个数的因数和个数
void bridge(int u, int v){ // 对左右奇偶点相同因数建边
    int i=0, j=0, iup = ph[u].size(), jup = ph[v].size();
    while(i < iup && j < jup) {
        if(ph[u][i].first == ph[v][j].first) { // 如果有一组因数相同就建一条容量为其个数最小值的边
            add(idx[u-1]+1+i+1, idx[v-1]+1+j+1, min(ph[u][i].second, ph[v][j].second));
            i++;j++;
        } else if(ph[u][i] > ph[v][j]) j++;
        else i++;
    }
}

int main(){
```

```
    init(); scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &a[i]);
        int tmp = a[i]; sum[i] = 0; // sum[i]统计因数的总数
        for (int j = 0; j < tot && prime[j] <= tmp; j++) {
            if(tmp%prime[j]==0) {
                int cc = 0; // 计算prime[j]因数的个数
                while(tmp%prime[j]==0) tmp /= prime[j],cc++;
                ph[i].emplace_back(prime[j], cc); // 推入向量中
                if(i&1)add(idx[i-1]+1, idx[i-1]+1+ph[i].size(), cc); //奇数点与其因数建边
                else add(idx[i-1]+1+ph[i].size(), idx[i-1]+1, cc); //偶数因数与其数建边
                sum[i] += cc; // 统计所有因数个数
            }
        } if(tmp>1) { // 存在大于1e5的因数
            sum[i] += 1;
            ph[i].emplace_back(tmp, 1);
            if(i&1)add(idx[i-1]+1, idx[i-1]+1+ph[i].size(), 1);
            else add(idx[i-1]+1+ph[i].size(), idx[i-1]+1, 1);
        } idx[i] = idx[i-1] + 1 + ph[i].size(); // 记录最后一个点编号
    } s = idx[n]+1; t = s+1; // s为源点, t为汇点
    for (int i = 1; i <= n; i++) {
        if(i&1)add(s, idx[i-1]+1, sum[i]); // s与奇数点建边
        else add(idx[i-1]+1, t, sum[i]); // 偶数点与t建边, 容量为因数个数
    } int u, v;
    for (int i = 1; i <= m; i++) {
        scanf("%d%d", &u, &v);
        if(v&1) swap(u, v);
        bridge(u, v); // 对奇偶数建边
    } return printf("%d\n", dinic()), 0; // 输出最大流
}
```

4.6.2 费用流

```
bool spfa(){ // spfa寻找是否
    memset(d, 0x3f, sizeof d); // 初始化最大距离
    d[s] = inq[s] = 1;
    deque<int> q; q.push_back(s); // 双端队列优化
    while(!q.empty()) {
        int u = q.front(); q.pop_front(); inq[u] = 0; // 出队
        for (int i = head[u]; ~i; i = e[i].nxt) {
            int v = e[i].v;
            if(e[i].w > 0 && d[v] > d[u] + e[i].c) {
                d[v] = d[u] + e[i].c; // 更新
                if(!inq[v]) {
                    inq[v] = 1; // 进队标记
                    if(!q.empty() && d[v] < d[q.front()]) q.push_front(v);
                    else q.push_back(v);
                }
            }
        }
    } return d[t] < inf; // 判断残留网络是否还有可行流
}

int dfs(int u, int w){ // 增广
```

```
    vis[u] = 1;
    if(u == t) return w; // 到达汇点停止
    int dlt = w, x; // 剩余流量
    for (int i = head[u]; ~i; i = e[i].nxt) {
        int v = e[i].v;
        if(!vis[v] && e[i].w > 0 && d[u]+e[i].c==d[v]) {
            x = dfs(v, min(dlt, e[i].w)); // 子节点增广
            e[i].w -= x; e[i^1].w += x;
            ans += x * e[i].c; // 计算费用
            dlt -= x; if(!dlt) return w; // 流满了可以直接返回
        }
    } return w - dlt; // 返回这次增广后的流量
}

int MCMF(){
    int sum = 0;
    while(spfa()) {
        do{
            memset(vis, 0, sizeof vis); // 初始化
            sum += dfs(s, inf);
        } while(vis[t]);
    } return sum; // 返回最大流
}
```

有向环最小权值覆盖

题意：给出n个点m条单向边边以及经过每条边的费用，让你求出走过一个哈密顿环（除起点外，每个点只能走一次）的最小费用。题目保证至少存在一个环满足条件。

解法：我们对于每个点都拆成前后两个点，前面的点与源连接，后面的点与汇连接，对于每条有向边u-¿v, u前面的点连接v后面的点，所有流量都为1。这样建图就是说每个节点都会指向后面一个节点，且不同节点指向后面的节点都不相同。

去重边

```
inline bool ad(int u, int v, int w, int c){
    int i;
    for (i = head[i]; ~i; i = e[i].nxt) {
        if(v == e[i].v) break;
    } if(~i) {
        if(c < e[i].c) e[i].c = c, e[i^1].c = -c;
        return 0;
    }
    e[cnt].v = v; e[cnt].w = w; e[cnt].c = c;
    e[cnt].nxt = head[u];
    head[u] = cnt++; return 1;
}

inline void add(int u, int v, int w, int c){
    if(ad(u, v, w, c)) ad(v, u, 0, -c);
}
```

HDU 3435 无向环最小权值覆盖

对于无向图我们只要建两条边

```
#include <bits/stdc++.h>
const int N = 2e3+9;
const int M = 6e4+9;
const int inf = 1e9+7;
using namespace std;
```

```
6
7  int n, m, s, t, T, ans,kas;
8  int d[N], head[N], cnt;
9  bool vis[N], inq[N];
10 struct Edge{ int v, nxt, c, w; }e[M];
11
12 inline bool ad(int u, int v, int w, int c){
13     int i;
14     for (i = head[u]; ~i; i = e[i].nxt) {
15         if(v == e[i].v) break;
16     } if(~i) {
17         if(c < e[i].c) e[i].c = c, e[i^1].c = -c;
18         return 0;
19     }
20     e[cnt].v = v; e[cnt].w = w; e[cnt].c = c;
21     e[cnt].nxt = head[u];
22     head[u] = cnt++; return 1;
23 }
24 inline void add(int u, int v, int w, int c){ if(ad(u, v, w, c)) ad(v, u, 0, -c); }
25
26 inline bool spfa(){
27     for(int i = 0; i <= t; i++) d[i] = inf;
28     d[s] = inq[s] = 1;
29     deque<int> q; q.push_back(s);
30     while(!q.empty()) {
31         int u = q.front(); q.pop_front(); inq[u] = 0;
32         for (int i = head[u]; ~i; i = e[i].nxt) {
33             int v = e[i].v;
34             if(e[i].w > 0 && d[v] > d[u] + e[i].c) {
35                 d[v] = d[u] + e[i].c;
36                 if(!inq[v]) {
37                     inq[v] = 1;
38                     if(!q.empty() && d[v] < d[q.front()]) q.push_front(v);
39                     else q.push_back(v);
40                 }
41             }
42         }
43     } return d[t] < inf;
44 }
45
46 int dfs(int u, int w){
47     vis[u] = 1;
48     if(u == t) return w;
49     int dlt = w, x;
50     for (int i = head[u]; ~i; i = e[i].nxt) {
51         int v = e[i].v;
52         if(!vis[v] && e[i].w > 0 && d[u]+e[i].c==d[v]) {
53             x = dfs(v, min(dlt, e[i].w));
54             e[i].w -= x; e[i^1].w += x;
55             ans += x * e[i].c;
56             dlt -= x; if(!dlt) return w;
57         }
58     } return w - dlt;
59 }
60
```

```cpp
int MCMF(){ //
    int sum = 0;
    while(spfa()) {
        do{
            for(int i = 0; i <= t; i++) vis[i] = 0;
            sum += dfs(s, inf);
        } while(vis[t]);
    } return sum;
}

inline void init(){ //
    ans = cnt = 0; for (int i = 0; i <= 2*n+2; i++) head[i] = -1; kas ++;
}

inline int slove(){
    init(); int u, v, w;
    for (int i = 0; i < m; i++) {
        scanf("%d%d%d", &u, &v, &w);
        add(u, v+n, 1, w);
        add(v, u+n, 1, w); // 无向图
    } s = 2*n+1; t = s+1;
    for (int i = 1; i <= n; i++) {
        add(i+n, t, 1, 0); // 加边
        add(s, i, 1, 0);
    } if(MCMF()==n) printf("Case %d: %d\n", kas, ans); // 最大流为n则代表匹配成功
    else printf("Case %d: NO\n", kas);
}

int main(){
    scanf("%d", &T);
    while(T--) {
        scanf("%d%d",&n,&m); slove();
    } return 0;
}
```

最大费用流 题意：给你一个N*N的矩阵，每个元素代表该处的权值。要求每个点只能走一次，左上角和右下角可以走两次但该处的权值只能获取一次。问你从左上角走到右下角（只能向下或右移动），再从右下角回到左上角（只能向上或左移动）所能得到的最大权值。

解析：从左上角走到右下角，再从右下角回到左上角所能得到的最大权值，我们可以转化为从【左上角起点】到【右下角终点】走两次所获的最大权值。题目要求起点终点可以走多次，其他定只能走一次。按这种思路的话起点和终点的权值我们多算了一次，最后结果要减去。

对于最大费用流，我们只要修改一下spfa就行。

```cpp
#include <bits/stdc++.h>
const int N = 8e5+9;
const int M = 1e7+9;
const int inf = 1e9+7;
using namespace std;

int n, m, s, t, T, ans;
int d[N], head[N], cnt;
bool vis[N], inq[N];
int mp[602][602];
/** MCMF模板 **/
struct Edge{int v, nxt, c, w;}e[M];
inline bool ad(int u, int v, int w, int c){
```

```
    e[cnt].v = v; e[cnt].w = w; e[cnt].c = c;
    e[cnt].nxt = head[u];
    head[u] = cnt++; return 1;
}

inline void init(){ans = cnt = 0; memset(head, -1, sizeof head);}
inline void add(int u, int v, int w, int c){if(ad(u, v, w, c)) ad(v, u, 0, -c);}
inline bool spfa() {
    for(int i = 0; i <= t; i++) d[i] = -inf;
    d[s] = inq[s] = 1;
    deque<int> q; q.push_back(s);
    while(!q.empty()) {
        int u = q.front(); q.pop_front(); inq[u] = 0;
        for (int i = head[u]; ~i; i = e[i].nxt) {
            int v = e[i].v;
            if(e[i].w > 0 && d[v] < d[u] + e[i].c) {
                d[v] = d[u] + e[i].c;
                if(!inq[v]) {
                    inq[v] = 1; q.push_back(v);
                }
            }
        }
    } return d[t] > -inf;
}

int dfs(int u, int w) {
    vis[u] = 1;
    if(u == t) return w;
    int dlt = w, x;
    for (int i = head[u]; ~i; i = e[i].nxt) {
        int v = e[i].v;
        if(!vis[v] && e[i].w > 0 && d[u]+e[i].c==d[v]) {
            x = dfs(v, min(dlt, e[i].w));
            e[i].w -= x; e[i^1].w += x;
            ans += x * e[i].c;
            dlt -= x; if(!dlt) return w;
        }
    } return w - dlt;
}

inline int MCMF(){
    int sum = 0;
    while(spfa()) {
        do{
            for(int i = 0; i <= t; i++) vis[i] = 0;
            sum += dfs(s, inf);
        } while(vis[t]);
    } return sum;
}
/** **/
inline int slove(){
    init(); int u, v, w; s = 1; t = 2*n*n;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if(i+j==2 || i+j==2*n) add(n*(i-1)+j, n*(i-1)+j+n*n, 2, mp[i][j]); //源点汇点流量为2
```

```
            else add(n*(i-1)+j, n*(i-1)+j+n*n, 1, mp[i][j]); // 流量为1，即每个点只经过1次
            if(i < n) add(n*(i-1)+j+n*n, n*(i)+j, 1, 0); // 往下走
            if(j < n) add(n*(i-1)+j+n*n, n*(i-1)+j+1, 1, 0); // 往右走
        }
    } MCMF(); printf("%d\n", ans-mp[1][1]-mp[n][n]); // 除去重复计算的值
}

inline int read(){
    char ch; int x;
    while((ch = getchar())<33);
    for (x = 0; ch >= '0' && ch <= '9'; ch=getchar())x = x*10+ch-'0'; return x;
}

int main(){
    while(~scanf("%d",&n)) {
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) mp[i][j]=read(); // 快读
        } slove();
    } return 0;
}
```

费用为流量平方倍

题意：用 1 到 n 运送k 个货物，m条边,每条边表示运送 x 单元货物的花费为$a*x^2$，c表示最大流量，求最小费用，若不能全部运送输出 -1;(c <= 5)

解析：对于每条边，我们进行拆边。对于一条u-¿v的边，系数为a，最大流为c, 我们可以拆成c条，每条流量都为1，系数分别为a, 3a, 5a, 7a ... 这样我们选取最小的x条，就能得到$a*x^2$

```
#include <bits/stdc++.h>
const int N = 120;
const int M = 5e5+9;
const int inf = 1e9+7;
using namespace std;
// 费用流部分开始
int d[N], head[N];
bool vis[N], inq[N];
int n, m, k, s, t, cnt, ans;
struct Edge{int v, w, c, nxt;} e[M];
inline void ad(int u, int v, int w, int c) {e[cnt] = {v, w, c, head[u]}; head[u] = cnt++;}
inline void add(int u, int v, int w, int c) {ad(u, v, w, c); ad(v, u, 0, -c);}
inline void init() { ans = cnt = 0; memset(head, -1, sizeof head);}

bool spfa(){
    for (int i = 0; i <= t; i++) d[i] = inf;
    deque<int> q; q.push_back(s); d[s] = inq[s] = 1;
    while(!q.empty()) {
        int u = q.front(); q.pop_front(); inq[u] = 0;
        for (int i = head[u]; ~i; i = e[i].nxt) {
            int v = e[i].v;
            if(e[i].w > 0 && d[v] > d[u] + e[i].c) {
                d[v] = d[u] + e[i].c;
                if(!inq[v]) {
                    inq[v] = 1;
                    if(!q.empty() && d[v] < d[q.front()]) q.push_front(v);
                    else q.push_back(v);
                }
            }
```

```
        }
    } return d[t] < inf;
}

int dfs(int u, int w) {
    vis[u] = 1;
    if(u == t) return w;
    int dlt = w, x;
    for (int i = head[u]; ~i; i = e[i].nxt) {
        int v = e[i].v;
        if(!vis[v] && e[i].w > 0 && d[u] + e[i].c == d[v]) {
            x = dfs(v, min(dlt, e[i].w));
            e[i].w -= x; e[i^1].w += x;
            ans += x * e[i].c;
            dlt -= x; if(!dlt) return w;
        }
    } return w - dlt;
}

inline int MCMF(){
    int sum = 0;
    while(spfa()) {
        do {
            for (int i = 0; i <= t; i++) vis[i] = 0;
            sum += dfs(s, k);
        } while(vis[t]);
    } return sum;
}
// 费用流部分结束
int main() {
    std::ios::sync_with_stdio(0);
    while(cin>>n>>m>>k){ init();
        int u, v, w, a; s = 0, t = n; add(s, 1, k, 0); // 为限制节点1流量，另设立源点流向1节点
        for (int i = 0; i < m; i++) {
            cin>>u>>v>>a>>w;
            for (int i = 1; i <= min(w, k); i++) add(u, v, 1, a*(2*i-1)); // 拆边
        } if(MCMF() >= k) cout<<ans<<endl; // 能运送k个
        else cout<<"-1"<<endl;
    } return 0;
}
```

混合图欧拉回路

题意：给你一个有向图，给你一个起点一个终点，需要你删除或者保留边来使得图满足下列要求：

1. 起点的入度 = 出度 - 1
2. 重点的入度 = 出度 + 1
3. 其他点的入度 = 出度

每条边保留或者删除都有一个花费，求使得满足条件最小的花费。

解析：我们另设源点与汇点，对于已给的边的两端节点做一下出度与入读的统计，对于删边花费b¿留边花费a时，我们从b连向a一条容量为1费用为b-a的边，反之我们从a连向b一条a-b的边，表示我们先优先选择花费小的方式，但是这种方式并不能保证每个节点入度与出度的要求，我们对于入读数偏多的点建立从源点向该点的边，容量为入读-出度，反之则建从该点到汇点的边，这样跑一次最小费用流，如果跑满了就能让各个节点平衡了，此时的费用加上之前选择的就是最后的答案。

```
#include <bits/stdc++.h>
const int N = 200;
using namespace std;
```

```
/** ---------------------------------- MCMF-START ---------------------------------- **/
int head[N], d[N], ind[N], oud[N];
int s, t, n, m, cnt, ans;
bool vis[N], inq[N];

struct Edge{int v, w, c, nxt;}e[N*N];
inline init(){cnt=ans=0;memset(head, -1, sizeof head);memset(ind, 0, sizeof ind);memset(oud, 0,
    sizeof oud);}
inline void ad(int u, int v, int w, int c){e[cnt] = {v, w, c, head[u]}; head[u] = cnt++;}
inline void add(int u, int v, int w, int c){ad(u, v, w, c); ad(v, u, 0, -c);}

inline bool spfa(){
    memset(d, 0x3f, sizeof d);
    d[s] = inq[s] = 1; deque<int> q; q.push_front(s);
    while(!q.empty()) {
        int u = q.front(); q.pop_front(); inq[u] = 0;
        for (int i = head[u]; ~i; i = e[i].nxt) {
            int v = e[i].v;
            if(e[i].w > 0 && d[v] > d[u] + e[i].c) {
                d[v] = d[u] + e[i].c;
                if(!inq[v]) {
                    inq[v] = 1;
                    if(!q.empty() && d[q.front()] > d[v]) q.push_front(v);
                    else q.push_back(v);
                }
            }
        }
    } return d[t] < d[N-1];
}

int dfs(int u, int w){
    vis[u] = 1;
    if(u == t) return w;
    int dlt=w,x;
    for (int i = head[u]; ~i; i = e[i].nxt) {
        int v = e[i].v;
        if(!vis[v] && e[i].w > 0 && d[v] == d[u] + e[i].c) {
            x = dfs(v, min(dlt, e[i].w));
            e[i].w -= x; e[i^1].w += x;
            ans += e[i].c * x;
            dlt -= x; if(!dlt) return w;
        }
    } return w - dlt;
}

inline int MCMF(){
    int sum=0;
    while(spfa()) {
        do{
            memset(vis, 0, sizeof vis);
            sum += dfs(s, 1);
        }while(vis[t]);
    } return sum;
}
/** ---------------------------------- MCMF-END ---------------------------------- **/
```

```
int main(){
    int T; scanf("%d",&T);
    for (int ks = 1; ks <= T; ks++){ init();
        scanf("%d%d%d%d", &n, &m, &s, &t); ind[s]++;oud[t]++; // 起始点和终点要满足其条件
        int u, v, a, b, l=0; s = n+1, t = s+1;
        for (int i = 0; i < m; i++) {
            scanf("%d%d%d%d", &u, &v, &a, &b);
            ans += min(a, b); // 选择较小的
            if(a < b) add(v, u, 1, b-a), ind[v]++, oud[u]++; // 如果a<b 则建反向的“后悔”边
            else add(u, v, 1, a-b); // 否则就建 u->v 的边，费用就是删边到留边的差距
        } for (int i = 1; i <= n; i++) {
            l += abs(oud[i] - ind[i]); // 统计流入流出的流量总和
            if(ind[i] > oud[i]) add(s, i, ind[i] - oud[i], 0);// 流入大于流出，与源点建边
            else add(i, t, oud[i] - ind[i], 0); // 否则与汇点建边
        } a = MCMF();
        if(a == l/2) printf("Case %d: %d\n", ks, ans); // 流满代表各个节点平衡了
        else printf("Case %d: impossible\n", ks); // 否则就是不成立
    } return 0;
}
```

4.6.3 上下界网络流

无源汇的上下界网络流 SGU - 194

题目大意：给n个点，及m根管道，每根管道用来流躺液体的，单向的，每时每刻每根管道流入=流出，要使得m条管道组成一个循环流。并满足每根管道流量限制范围为[Li,Ri].

答：进行再构造让每条边的下界为0，对于每根管子上界容量R_i和下界容量L_i，我们让管子的容量下界变为0，上界为$R_i - L_i$，不过这样做就违背了每个节点的流入量=流出量这个原则，所以我们添加额外的源点和汇点与不平衡的节点建边来补全就行了。

```
#include <bits/stdc++.h>
const int N = 1e4+9;
const int M = 1e6+9;
const int inf = 1e9+7;
using namespace std;
int n, m, cnt, s, t;
int head[N], cur[N], d[N], du[N], dn[M];
/** --------------------------------最大流---------------------------------- **/
struct Edge{int v, nxt, w;}e[M];
void ad(int u, int v, int w){e[cnt]={v,head[u],w};head[u] = cnt++;}
void add(int u, int v, int w){ad(u, v, w); ad(v, u, 0);}

bool bfs(){
    for(int i = 0; i <= t; i++) d[i] = 0; d[s] = 1;
    queue<int> q; q.push(s);
    while(!q.empty()) {
        int u = q.front(); q.pop();
        for (int i = head[u]; ~i; i = e[i].nxt) {
            int v = e[i].v;
            if(!d[v] && e[i].w > 0) {
                d[v] = d[u] + 1;
                q.push(v); if(v == t) return 1;
            }
        }
    } return 0;
}
```

```
27
28 int dfs(int u, int w){
29     if(u == t || w == 0) return w;
30     int dlt = w;
31     for (int &i = cur[u]; ~i; i = e[i].nxt) {
32         int v = e[i].v;
33         if(d[v]==d[u]+1 && e[i].w > 0) {
34             int x = dfs(v, min(dlt, e[i].w));
35             e[i].w -= x; e[i^1].w += x;
36             dlt -= x; if(!dlt) return w;
37         }
38     } return w - dlt;
39 }
40
41 int dinic(){
42     int ans = 0;
43     while(bfs()) {
44         for (int i = 0; i <= t; i++) cur[i] = head[i];
45         ans += dfs(s, inf);
46     } return ans;
47 }
48 /** --------------------------------最大流---------------------------------- **/
49
50 int main(){
51     int u, v, b, c;
52     memset(head, -1, sizeof head); // 链式前向星初始化
53     scanf("%d%d", &n, &m); s = n+1, t = s+1;
54     for(int i = 1; i <= m; i++) {
55         scanf("%d%d%d%d", &u, &v, &b, &c);
56         add(u, v, c-b); // 添加u->v的上界为c-b的边
57         du[u] -= b;du[v] += b; // 统计每个点的流入流出
58         dn[i] = b; // 记录每条边的流入下界
59     } for (int i = 1; i <= n; i++) {
60         if(du[i] > 0) add(s, i, du[i]); // 对于流入大于流出的，s与其建边
61         else if(du[i] < 0) add(i, t, -du[i]); // 对于流出大于流入的，其与t建边
62     } dinic();// printf("%d\n", dinic());
63     for (int i = head[s]; ~i; i = e[i].nxt) {
64         if(e[i].w>0) return puts("NO"), 0; // 如果有边没有跑满即不存在可行流
65     } puts("YES");
66     for (int i = 1; i <= m; i++) {
67         printf("%d\n", e[i*2-1].w+dn[i]); // 输出每条边的流量
68     } return 0;
69 }
```

有源汇的上下界网络流ZOJ - 3229

题意：一个男生给m个女神拍照，计划拍照n天，每一天男生最多给C个女神拍照，每天拍照数不能超过D张，而且给每个女神i拍照有数量限制$[L_i$fi$R_i]$，对于每个女神n天的拍照总和至少为G_i，如果有解求男生最多能拍多少张照，并求每天给对应女神拍多少张照；不存在可行解输出-1。

解题思路：对于如果不存在下界L_i, 那么这幅图就是一个简单的最大流，我们对于源点与n天分别建一条边容量为D_i，每天与其对应的c个女生建一条边容量为R_i，每个女生与汇点建一条边容量为inf - G_i, 然后我们就跑一次最大流就可以了。对于有源汇的最大流，我们将其变为无源汇的网络流只需要将原来的汇点与源点建一条边，这样就是一条循环流。如此按照无源汇的方法再添加附加源与附加汇, 对于附加源与附加汇跑一次最大流来看看是否存在满足下界的可行流，如果满足，我们再跑一次初始源与初始汇的最大流来获得最大流

```
1 #include<bits/stdc++.h>
2 const int N = 2e3+9;
```

```cpp
const int M = 1e6+9;
const int inf = 1e9+7;
using namespace std;

int n, m, s, t, cnt, cno;
int head[N], cur[N], d[N], du[N], dn[M], id[M];
/** ---------------------------------- 最大流 -------------------------- **/
struct Edge{int v, nxt, w;}e[M];
void ad(int u, int v, int w){e[cnt]={v,head[u],w};head[u] = cnt++;}
void add(int u, int v, int w){ad(u, v, w); ad(v, u, 0);}

bool bfs(){
    for(int i = 0; i <= t; i++) d[i] = 0; d[s] = 1;
    queue<int> q; q.push(s);
    while(!q.empty()) {
        int u = q.front(); q.pop();
        for(int i = head[u]; ~i; i = e[i].nxt) {
            int v = e[i].v;
            if(!d[v] && e[i].w > 0) {
                d[v] = d[u] + 1;
                q.push(v); if(v == t) return 1;
            }
        }
    } return 0;
}

int dfs(int u, int w) {
    if(u == t || w == 0) return w;
    int dlt = w;
    for (int &i = cur[u]; ~i; i = e[i].nxt) {
        int v = e[i].v;
        if(d[v] == d[u]+1 && e[i].w > 0){
            int x = dfs(v, min(dlt, e[i].w));
            e[i].w -= x; e[i^1].w += x;
            dlt -= x; if(!dlt) return w;
        }
    } return w - dlt;
}

int dinic(){
    int ans = 0;
    while(bfs()){
        for (int i = 0; i <= t; i++) cur[i] = head[i];
        ans += dfs(s, inf);
    } return ans;
}
/** ---------------------------------- 最大流 -------------------------- **/

void init(){ // 初始化
    memset(head, -1, sizeof head);
    memset(du, 0, sizeof du);
    memset(dn, 0, sizeof dn);
    cnt = cno = 0;
}

```

```
int main(){
    int flag = 0;
    while(~scanf("%d%d", &n, &m)) {
        init(); int C, D, T, L, R, x; flag = 0;
        for (int i = 1; i <= m; i++) {
            scanf("%d", &x);
            du[n+i] -= x; du[n+m+2] += x; // 统计每个女生节点与初始汇节点的流入流出
            add(n+i, n+m+2, inf-x); // 建议上界-下界的边 = inf - Gi
        } for (int i = 1; i <= n; i++) {
            scanf("%d%d", &C, &D);
            add(n+m+1, i, D); // 建立初始源与天数i的边，不存在下界无需统计
            for (int j = 0; j < C; j++) {
                scanf("%d%d%d", &T, &L, &R);T++;
                add(i, n+T, R-L); dn[cno] = L; id[cno++] = cnt-2; // 记录下界与边数id
                du[i] -= L; du[n+T] += L; // 统计流量
            }
        } for(int i = 1; i <= n+m+2; i++) {
            if(du[i] > 0) add(n+m+3, i, du[i]); // 补全每个节点缺少的流量
            else if(du[i] < 0) add(i, n+m+4, -du[i]);
        } s = n+m+3; t = n+m+4; add(n+m+2, n+m+1, inf); // 变无源汇
        int ans = dinic(); //printf("%d\n", dinic());
        for (int i = head[s]; ~i; i = e[i].nxt) {
            if(e[i].w > 0) { flag = 1; puts("-1\n"); break; } // 检测是否所有从附加源出发的边都跑满了
        } if(flag) continue;
        s = n+m+1; t=s+1;
        printf("%d\n", dinic()); // 跑初始源与初始汇的最大流
        for (int i = 0; i < cno; i++){
            printf("%d\n", dn[i]+e[id[i]^1].w); // 输出每条边流量
        } puts("");
    } return 0;
}
```

有源汇的上下界最小流 SGU - 176

题目：有一个工业加工生产的机器，源为1，汇为n，中间生产环节有货物加工数量限制，输出u v z c，当c等于1时表示这个加工的环节必须对纽带上的货物全部加工（即上下界都为z），c等于0表示没有下界，求节点1（起点）最少需要投放多少货物才能传送带正常工作。

解析：如果只是普通的最小流，我们只要跑一次从汇到源的最大流，那么此时我们可以根据残留网络中的边统计出最小流。此题，我们先不考虑最小流，那就是一个源为1，汇为n，的一个上下界网络流，统计每个节点的流入流出就可以判断出是否存在可行解。

```
#include <bits/stdc++.h>
const int N = 110;
const int M = 1.09e4+9;
const int inf = 1e9+7;
using namespace std;
int n, m, cnt, s, t;
int head[N], cur[N], du[N], dn[M], d[N];
/** ---------------------------- 最大流 ---------------------------------------- **/
struct Edge{int w, nxt, v;}e[M];
void ad(int u, int v, int w) {e[cnt] = {w, head[u], v};head[u] = cnt++;}
void add(int u, int v, int w){ad(u, v, w); ad(v, u, 0);}

bool bfs(){
    for (int i = 0; i <= n+2; i++) d[i]=0; d[s] = 1;
    queue<int> q; q.push(s);
    while(!q.empty()) {
```

```
        int u = q.front(); q.pop();
        for (int i = head[u]; ~i; i = e[i].nxt) {
            int v = e[i].v;
            if(!d[v] && e[i].w > 0) {
                d[v] = d[u] + 1;
                q.push(v); if(v == t) return 1;
            }
        }
    } return 0;
}

int dfs(int u, int w){
    if(u == t || w == 0) return w;
    int dlt = w;
    for (int &i = cur[u]; ~i; i = e[i].nxt) {
        int v = e[i].v;
        if(d[v] == d[u]+1 && e[i].w > 0) {
            int x = dfs(v, min(e[i].w, dlt));
            e[i].w -= x; e[i^1].w += x;
            dlt -= x; if(!dlt) return w;
        }
    } return w - dlt;
}

int dinic(){
    int ans = 0;
    while(bfs()) {
        for(int i = 0; i <= n+2; i++) cur[i] = head[i];
        ans += dfs(s, inf);
    } return ans;
}
/** --------------------------- 最大流 ------------------------------------- **/

int main(){
    scanf("%d%d", &n, &m);
    memset(head, -1, sizeof head);
    int u, v, z, c; s = n+1; t = s+1;
    for (int i = 0; i < m; i++) {
        scanf("%d%d%d%d", &u, &v, &z, &c);
        if(c) {
            add(u, v, 0); dn[i] = z; // 记录边下界
            du[u] -= z; du[v] += z; // 统计流入流出
        } else add(u, v, z); // 只有上界的边
    } for (int i = 1; i <= n; i++) {
        if(du[i] > 0) add(s, i, du[i]); // 附加源建边
        else if(du[i] < 0) add(i, t, -du[i]); // 附加汇建边
    } int tmp = cnt; add(n, 1, inf); dinic(); // 转无源汇跑最大流
    int ans = e[tmp^1].w; // 记录可行流中的流量（即n->1的边中的流量）
    for (int i = head[s]; ~i; i = e[i].nxt) {
        if(e[i].w > 0) return puts("Impossible"), 0; // 无可行流则结束
    } e[tmp].w = 0; e[tmp^1].w = 0;s = n; t = 1; ans -= dinic(); // 去除1-n之间的边并跑一次最小流
    if(ans < 0) { // 能去除的流量比可行流还要大
        add(n+1, 1, -ans); // 附加源与源建一条边，容量为差的边
        s = n+1; t = n; ans = 0; // 即最小流为0
        dinic(); // 补足可行流
```

```
    } printf("%d\n", ans); // 输出去除可以去除的流量后的最小值
    for(int i = 0; i < 2*m; i+=2) {
        printf("%d ", dn[i/2]+e[i^1].w); // 输出每条边的流量
    } return 0;
}
```

5 String

5.1 Hash

有时候我们要统计很大的数的个数时，但是这个时候我们又开不下这么大的数组，我们常常可以借助map，或者$hash_table$之类的工具。

Hash 统计数字个数

有时候map占用的数据也非常大，我们可以利用链式前向星来代替链表实现一个简易的hash来统计每个数的个数，这个所占用的空间比map要小的多。

5.1.1 统计个数

```
const int mod=1<<15;
struct Edge{int v,nxt,w;}e[mod];
int head[mod], cnt;
void init(){ memset(head, -1, sizeof head); cnt = 0; }

int gethash(int x){ // 将数字hash
    return (x+ mod) & (mod-1);
}

void insert(int x) { // 插入
    int id=gethash(x); // 得到hash值
    for(int i=head[id]; i != -1;i=e[i].nxt) { // 便利冲突的元素
        if(e[i].v==x) { e[i].w++; return; }
    } e[cnt] = {x, head[id], 1}; head[id]=cnt++; // 新元素额外添加
}

void search(int x) { // 查找个数
    int id=gethash(x);
    for(int i=head[id] ; i!=-1;i=e[i].nxt) {
        if(e[i].v==x) return e[i].w;
    } return 0;
}
```

5.2 统计字符串

如果我们就考虑小写字母的话，那么有26种小写字母，那么对于1e6长度的字符串，就有$10^{1414973}$多种情况，而对于我们所要处理的字符串实在是太少，所以我们对于字符串hash很少来处理冲突，如果发现确实有冲突，那么我们可以采用使用两种不同的hash来降低同时冲突的概率。

一般我们对于字符串是如此hash的：

比如有一个字符串abc

我们构造一个函数：$f(x) = a[0] * x^{n-1} + a[1] * x^{n-2} + ... + a[n-1]$

x就是我们自己选取的数，一般常用一个质数，比如131，233之类的也用字母种类数比如26之类的。

那么对于上面这个字符串$f(26) = a * x^2 + b * x^2 + c$

a,b,c的值可以是1,2,3或者是他们本身的ASCII值。

使用这种方式的好处就是一次O(n)的预处理后可以O(1)查询任意字串的值，除此之外还有滚动哈希之类的操作.

5.3 子串种类

题意: 给定一个字符串s, 查询长度为m的子串一共有多少种.

滚动hash

滚动hash其实就是我们先计算第一个长度为m的字符串的hash值,我们去计算第二个长度为m的子串时,我们只需要使用上一个hash值减去其第一个字母的影响然后乘x再加上后面缺少的那个字母的值. 举例如下

$f(x) = a[0] * x^{n-1} + a[1] * x^{n-2} + ... + a[n-1]$, 我们先减去 $a[0] * x^{n-1}$ 然后整体乘x , 最后加上a[n].

```
#include <cstdio>
#include <cstring>
const int N = 1.6e7+9;
using namespace std;
char s[N];
bool has[N], vis[256];
int n, nc, a[256], m, cnt;

int main(){
    scanf("%d%*d%s",&m,s); n = strlen(s);
    for (int i = 0; i < n; i++) {
        if(!vis[s[i]]) {
            vis[s[i]] = 1; a[s[i]]=nc++; // 对于字母出现的顺序来决定大小
        }
    } int ans = 0, x = 0, y = 1;
    for (int i = 0; i < m; i ++) {
        x = x*nc + a[s[i]]; // 计算第一个长度为m的字符串的hash值
        y *= nc;
    } y/= nc; has[x] = 1; ans ++; // y = nc^{m-1}
    for (int i = 1; i <= n-m; i++) {
        x = (x - a[s[i-1]]*y) * nc + a[s[i+m-1]]; // 滚动hash
        if(!has[x]) has[x] = 1, ans++;
    } return printf("%d\n", ans), 0;
}
```

O(n)预处理后整体查询

由于map的速度实在是太慢了,而且占用空间也大,我们使用之前自己写的.

这里我们o(n)预处理的h(i) 代表的是以i为结尾的字串的hash值. bit则存放的是x^i, 那么我们想要得到一个字串(l,r)的值的话我们对于0-r的字串的值减去前0-(l-1) 的字串的影响就行了.

```
#include <cstdio>
#include <map>
#include <cstring>
#define ull unsigned long long
const int base = 1e9+9;
const int N = 2e6+9;
using namespace std;
int n, m;
ull bit[N], h[N];
char s[N];

const int mod=1<<15;
struct Edge{int v,nxt,w;}e[mod];
int head[mod], cnt;
void init(){memset(head, -1, sizeof head); cnt = 0;}
int gethash(int x){return (x+ mod) & (mod-1);}
void insert(int x) {
    int id=gethash(x);
    for(int i=head[id]; i != -1;i=e[i].nxt) {
```

```
        if(e[i].v==x) { e[i].w++; return;}
    } e[cnt].v = x; e[cnt].nxt = head[id]; e[cnt].w = 1; head[id]=cnt++;
}

int main(){
    memset(head, -1, sizeof head); // 初始化
    scanf("%d%*d%s", &m, s+1); bit[0] = 1; n = strlen(s+1);
    for (int i = 1; i < N; i++) bit[i] = bit[i-1]*base; // x^i
    for (int i = 1; i <= n; i++) h[i] = h[i-1]*base + s[i]; // 以i为结尾的字串的hash值
    for (int i = m; i <= n; i++) {
        ull tmp = (h[i] - h[i-m]*bit[m]); // i-m+1, i 字串的hash值
        insert(tmp);
    } printf("%d\n", cnt);
    return 0;
}
```

二维字符矩阵

题意: 给出一个n * m的矩阵。让你从中发现一个最大的正方形。使得这样子的正方形在矩阵中出现了至少两次。输出最大正方形的边长。

$$f(x,y) = a[0][0]*x^{m-1}*y^{n-1}+a[0][1]*x^{m-2}*y^{n-1}+...+a[0][m-1]*y^{n-1} \quad +a[1][0]*x^{m-1}*y^{n-2}+a[1][1]*x^{m-2}*y^{n-2}+...+a[1][$$

这样下来我们如果要计算一个左上角(l1, r1) ,右下角为(l2, r2)的字符矩阵的hash值就方便的多了.

如果x边长的正方形矩阵满足题意,那么1- (x-1)边长的正方形矩阵都满足,所以具有二分性质,我们可以使用二分来枚举边长

```
#include <bits/stdc++.h>

using namespace std;
using ull = unsigned long long;
const int N = 555;
const int mod = 1e5+7;
struct Edge { ull v; int nxt; }e[N*N];
int head[mod];
ull base1 = 1313, base2 =13131;
ull has1[N][N], has2[N][N], bit1[N], bit2[N];
char s[N][N];
int n, m, cnt;

void init(){ // 初始化
    bit1[0] = bit2[0] = 1; cnt = 0;
    for (int i = 1; i < N; i++) {
        bit1[i] = bit1[i-1]*base1;
        bit2[i] = bit2[i-1]*base2;
    } for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            has1[i][j] = has1[i][j-1]*base1+s[i][j]; // 计算的是i行的以j为末尾的字符串的hash值
            has2[i][j] = has2[i-1][j]*base2+has1[i][j]; // 计算1 − i, 1 − j 的矩阵的hash值
        }
    }
}

void inser(int u, ull v){ e[cnt].v = v; e[cnt].nxt = head[u]; head[u] = cnt++;}

ull cal(int i, int j, int x){ // 计算 (i,j) (i+x-1, j+x-1) 矩阵的hash值
    ull ret = has2[i+x-1][j+x-1];
```

```
    ret -= has2[i+x-1][j-1] * bit1[x] + has2[i-1][j+x-1]*bit2[x];
    ret += has2[i-1][j-1] * bit1[x] * bit2[x];
    return ret;
}

bool sech(ull x){
    int t = x % mod;
    for (int i = head[t]; ~i; i = e[i].nxt) {
        if(e[i].v == x) return 1;
    } return 0;
}

bool check(int x){ // 判断是否存在满足题意的 x长度的边长的矩阵
    cnt = 0; memset(head, -1, sizeof head); // 清空hash表种的元素
    for (int i = 1; i+x-1 <= n; i++) {
        for (int j = 1; j+x-1 <= m; j++) {
            ull t = cal(i, j, x); // 每次计算一个矩阵
            if(sech(t)) return 1; // 查询是否有同样的矩阵
            inser(t%mod, t); // 没有就将矩阵的hash值插入
        }
    } return 0;
}

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) {
        scanf("%s", s[i]+1);
    } init(); int l = 1, r = min(m, n);
    while(l <= r){ // 二分边长
        int mid = (l+r)>>1;
        if(check(mid)) l = mid + 1;
        else r = mid - 1;
    } return printf("%d\n", l-1), 0;
}
```

最小表示法

求字符串的循环表示法：

我们有一个字符串s,我们可以不停的把头上的字母放在尾巴上这样就构成一个循环串s[k],s[k+1],s[k+2],..,s[n],s[1],..,s[k-1]; 我们可以得到n个循环串，我们要去寻找字典序最小的串的首字母的原始下标。

我们可以使用双指针法，可以做到O(n)，也可以利用hash和二分算法不过这个复杂度是O(nlogn)的；

```
int getMi(){
    int i = 0, j = 1, l; // i,j 就是需要比较的两个循环串的首字母下标，l是长度
    while(i < n && j < n){
        for(l = 0; l < n; l++){
            if(s[(i+l)%n] != s[(j+l)%n]) break; // 找到第一个不相等的。
        } if(l >= n) break; // 没找到说明两个循环串相等
        if(s[(i+l)%n] < s[(j+l)%n]) j += l+1; // 如果i串更小，那么j移动l+1位,这里改成>就是字典序最大
        else i += l+1; // 否则i串移动l+1位
        if(i==j) j++; // 如果i==j了让j往后移动一格
    } return i<j?i:j; // 返回下标小的(此时要么有一个>n或者i,j串相等)
}
```

二分+ Hash

hash的做法是n个字符串我们从第一个开始比较，始终维护最小的循环串，比较字符串是log(n)比较的，方法是二分到两个字符串第一个不同的位置。所以总体复杂度是O(nlogn)

```cpp
#include <bits/stdc++.h>
using namespace std;
using ull = unsigned long long;
const int N = 1e5+9;
const int base = 131;
int n;
char s[N];
ull bit[N], has[N];

bool check(int x, int y, int l){ // 判断x,y开始l长度的字符串是否相等
    ull u = has[x+l-1] - has[x-1]*bit[l];
    ull v = has[y+l-1] - has[y-1]*bit[l];
    return u == v;
}

bool ok(int x, int y){ // 比较那个下标开始的串更小
    int l = 0, r = n;
    while(l <= r) {
        int mid = (l+r) >> 1;
        if(check(x, y, mid)) l = mid + 1;
        else r = mid - 1;
    } if(l-1==n) return 0;
    return (s[x+l-1]) < (s[y+l-1]);
}

int main() {
    scanf("%s", s+1); bit[0] = 1; n = strlen(s+1);
    for (int i = 1; i <= n; i++) s[i+n] = s[i];
    for (int i = 1; i < N; i++) bit[i] = bit[i-1]*base;
    for (int i = 1; i <= n*2; i++) has[i] = has[i-1]*base + s[i];
    int x = 1;
    for (int i = 2; i <= n; i++) {
        if(ok(i, x)) x = i;
    } return printf("%d\n", x), 0;
}
```

5.4 Trie树

01 字典树

```cpp
namespace trie{
    const int NODE = 2e6 * 2;
    int ch[NODE][2], size[NODE], pool = 2;
    // int vis[NODE], vid = 1;

    inline void init(){
        memset(ch, 0, sizeof ch);
        memset(size, 0, sizeof size);
    }

    inline void ins(bs a, bool flag) {
        int u = 1;
        for (int i = 31; i >= 0; i--) {
            if (!ch[u][a[i]]) ch[u][a[i]] = pool++;
            u = ch[u][a[i]];
```

```
            if (flag) size[u] ++;
            else size[u] --;
        }
    }

    inline ll find(bs a) {
        int u = 1; ll ans = 0;
        for (int i = 31; i >= 0; i--) {
            if (ch[u][1-a[i]] && size[ch[u][1-a[i]]]) {
                ans |= (1LL << i );
                u = ch[u][1-a[i]];
            } else {
                u = ch[u][a[i]];
            }
        } return ans;
    }
};
```

可持久化字典树，区间查询

```
#define bs bitset<32>
const int N = 6e5+9;
namespace trie{
    int ch[N<<5][2], root[N], size[N<<5], pool = 1;
    bs tmp;

    inline void ins(int &nw, int od, int idx=24) {
        nw = pool++;
        for (int i = 0; i < 2; i++) ch[nw][i] = ch[od][i];
        size[nw] = size[od] + 1;
        if(!idx) return ;
        int x = tmp[idx-1];
        ins(ch[nw][x], ch[od][x], idx-1);
    }

    inline int query(int l, int r, int idx=24) {
        if(!idx) return 0;
        int x = tmp[idx-1];
        if(size[ch[r][x]] > size[ch[l][x]])
            return query(ch[l][x], ch[r][x], idx-1) + (1<<(idx-1));
        return query(ch[l][1-x], ch[r][1-x], idx-1);
    }
};
```

带延迟操作的01字典树合并-csl

```
#include <bits/stdc++.h>
using namespace std;

const int maxn = 2e7;
struct Node {
    int sz, tag;
    Node* ch[2];
    Node() : sz(0), tag(0) { memset(ch, 0, sizeof(ch)); }
    void maintain() {
        sz = 0;
        if (ch[0] != nullptr) sz += ch[0]->sz;
```

```
12          if (ch[1] != nullptr) sz += ch[1]->sz;
13      }
14      void pushdown() {
15          if (tag == 0) return;
16          if (tag & 1) {
17              swap(ch[0], ch[1]);
18              if (ch[0] != nullptr) ch[0]->tag++;
19          }
20          if (tag > 1) {
21              if (ch[0] != nullptr) ch[0]->tag += (tag >> 1);
22              if (ch[1] != nullptr) ch[1]->tag += (tag >> 1);
23          } tag = 0;
24      }
25  } mem[maxn];
26
27  Node* newNode() {
28      static int tot = 0;
29      assert(tot + 1 < maxn);
30      return &mem[tot++];
31  };
32
33  struct Trie {
34      void insert(Node*& o, int x, int dep = 30) {
35          if (o == nullptr) o = newNode();
36          if (dep == 0) {
37              o->sz++;
38              return;
39          }
40          o->pushdown();
41          insert(o->ch[x & 1], x >> 1, dep - 1);
42          o->maintain();
43      }
44      Node* merge(Node* o, Node* k, int dep = 30) {
45          if (o == nullptr) return k;
46          if (k == nullptr) return o;
47          if (o == nullptr && k == nullptr) return nullptr;
48          if (dep == 0) {
49              o->sz += k->sz;
50              return o;
51          }
52          o->pushdown(), k->pushdown();
53          o->ch[0] = merge(o->ch[0], k->ch[0], dep - 1);
54          o->ch[1] = merge(o->ch[1], k->ch[1], dep - 1);
55          o->maintain();
56          return o;
57      }
58      int query(Node* o, int x, int dep) {
59          if (o == nullptr) return 0;
60          if (dep == 0) return o->sz;
61          o->pushdown();
62          return query(o->ch[x & 1], x >> 1, dep - 1);
63      }
64  };
65
66  struct DSU {
```

```
    vector<int> fa;
    DSU() {}
    DSU(int n) { fa.resize(n), iota(fa.begin(), fa.end(), 0); }
    int find(int x) { return fa[x] == x ? x : fa[x] = find(fa[x]); }
    bool same(int x, int y) { return find(x) == find(y); }
    void merge(int x, int y) { fa[find(y)] = find(x); }
};

int main() {
    int n, m;
    scanf("%d%d", &n, &m);
    DSU d(n);
    Trie t;
    vector<Node*> rt(n);
    for (int i = 0, x; i < n; i++) {
        scanf("%d", &x);
        t.insert(rt[i], x);
    }
    while (m--) {
        static int op, u, v, k, x;
        scanf("%d", &op);
        if (op == 1) {
            scanf("%d%d", &u, &v);
            --u, --v;
            if (!d.same(u, v)) {
                rt[d.find(u)] = t.merge(rt[d.find(u)], rt[d.find(v)]);
                d.merge(u, v);
            }
        } else if (op == 2) {
            scanf("%d", &u); --u;
            rt[d.find(u)]->tag++;
        } else {
            scanf("%d%d%d", &u, &k, &x); --u;
            printf("%d\n", t.query(rt[d.find(u)], x, k));
        }
    } return 0;
}
```

5.5 KMP

```
void getNxt(){
    int i = 0, j = -1;
    nxt[i] = -1;
    while(i<m){
        if(j==-1||p[i]==p[j]){
            i++; j++; nxt[i] = j;
        }else j = nxt[j];
    }
}
```

5.6 马拉车

```
char s[maxn];
```

```
int p[maxn];
int Manacher(){
    int len = strlen(s),id=0,r=0;
    for(int i = len; i >= 0; i--){
        s[i*2+2] = s[i];
        s[i*2+1] = '#';
    }
    s[0] = '#';
    for(int i = 2; i < 2*len + 1; i++){
        if(p[id] + id>i)p[i] = min(p[2*id - i], p[id]+id-i);
        else p[i] = 1;
        while(s[i-p[i]] == s[i+p[i]]) ++ p[i];
        if(id + p[id] < i + p[i])id = i;
        if(r < p[i]) r = p[i];
    }
    return r - 1;
}
```

5.7 AC自动机

题意：第一行输入测试数据的组数，然后输入一个整数n，接下来的n行每行输入一个单词，最后输入一个字符串，问在这个字符串中有多少个单词出现过。

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e6;
const int maxk = 26;
struct Node{
    int nxt[26];
    int fail;
    int cnt;
    void init(){
        memset(nxt, -1, sizeof nxt);
        fail = -1;
        cnt = 0;
    }
} trie[maxn];
int root, num;
int que[maxn], head, tail;
char w[60], str[maxn];

void init(){ root=num=0; trie[root].init(); }

inline void inser(char *s){
    int idx, p = root;
    while(*s){
        idx = *s - 'a';
        if(trie[p].nxt[idx]==-1){
            trie[++num].init();
            trie[p].nxt[idx] = num;
        }
        p = trie[p].nxt[idx];
        s ++;
    }
    trie[p].cnt++;
```

```
}

void AC(){
    int cur,p,son;
    head = tail = 0;
    que[tail++] = root;
    while(head < tail){
        cur = que[head++];
        for(int i = 0; i < maxk; i++){
            if(trie[cur].nxt[i]!=-1){
                son = trie[cur].nxt[i];
                p = trie[cur].fail;
                if(cur == root) trie[son].fail = root;
                else trie[son].fail = trie[p].nxt[i];
                que[tail++] = son;
            } else {
                p = trie[cur].fail;
                if(cur == root){
                    trie[cur].nxt[i] = root;
                }
                else trie[cur].nxt[i] = trie[p].nxt[i];
            }
        }
    }
}

int Find(char *s){
    int p = root, cnt = 0, idx, tmp;
    while(*s){
        idx = *s - 'a';
        p = trie[p].nxt[idx];
        if(p==-1) p = root;
        tmp = p;
        while(tmp != root && trie[tmp].cnt != -1){
            cnt += trie[tmp].cnt;
            trie[tmp].cnt = -1;
            tmp = trie[tmp].fail;
        }
        s ++;
    }
    return cnt;
}
int n;
int main(){
    int t;scanf("%d",&t);
    while(t--){
        scanf("%d",&n);
        init();
        getchar();
        for(int i = 0; i < n; i++){
            gets(w);
            inser(w);
        }
        AC();
        gets(str);
```

```
        printf("%d\n",Find(str));
    }
}
```

6 Data Structure

6.1 RMQ

```
int mx[maxn][20];
int f[maxn], num[maxn];

void build(){
    for(int i = 1; i <= n; i++)
        mx[i][0] = f[i];
    for(int j = 1; (1<<j)<= n; j++)
        for(int i = 1; i + (1<<j)- 1<= n; i++)
            mx[i][j] = max(mx[i][j-1], mx[i + (1<<(j-1))][j - 1]);
}

int RMQ(int s, int v){
    if(s > v)return 0;
    int k = (int)(log(v - s + 1)/ log(2));
    return max(mx[s][k], mx[v - (1<<k) + 1][k]);
}
```

二维RNQ

```
int mp[maxn][maxn];
int mx[maxn][maxn][10], mi[maxn][maxn][10];

void build(){
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            mx[i][j][0] = mi[i][j][0] = mp[i][j];
    for (int i = 1; i <= n; i++)
        for (int j = 1; (1<<j)<= n; j++)
        for (int k = 1; k <= n; k++){
            mx[i][k][j] = max(mx[i][k][j-1], mx[i][k+(1<<(j-1))][j-1]);
            mi[i][k][j] = min(mi[i][k][j-1], mi[i][k+(1<<(j-1))][j-1]);
        }
}

int query(int x, int y){
    int k = (int)(log(b)/log(2));
    int maxx = -inf, minx = inf;
    for(int i = x; i <= x+b-1; i++){
        maxx = max(max(mx[i][y][k], mx[i][y - (1<<k) + b][k]), maxx);
        minx = min(min(mi[i][y][k], mi[i][y - (1<<k) + b][k]), minx);
    }
    return maxx - minx;
}
```

01 RMQ

```
#include <bits/stdc++.h>
using namespace std;
```

```
#define PB push_back
#define ZERO (1e-10)
#define INF (1<<29)
#define CL(A,I) (memset(A,I,sizeof(A)))
#define DEB printf("DEB!\n");
#define D(X) cout<<" "<<#X": "<<X<<endl;
#define EQ(A,B) (A+ZERO>B&&A-ZERO<B)
typedef long long ll;
typedef long double ld;
typedef pair<ll,ll> pll;
typedef vector<int> vi;
typedef pair<int,int> ii;
typedef vector<ii> vii;
#define IN(n) int n; cin >> n;
#define FOR(i, m, n) for (int i(m); i < n; i++)
#define REP(i, n) FOR(i, 0, n)
#define F(n) REP(i, n)
#define FF(n) REP(j, n)
#define FT(m, n) FOR(k, m, n)
#define aa first
#define bb second
void ga(int N,int *A){F(N)scanf("%d",A+i);}
#define LG (20)
#define MX (1<<LG)
#define BT (12) //K =~= log(N)/2
#define SZ (MX/BT+1)
#define P2(v) (!(v&(v-1)))
//structure for RMQ which differs exactly by 1
struct RMQ{
    /*
     * T[b][i][j]: Smallest element on interval i-j, in gradient of shape "b"
     * G: Precalculate logarithms
     * B: Bitmask of gradients of each interval of size BT
     * I: Lowest element on each little interval
     * E: Value of lowest element on each interval
     * dp: RMQ: Sparse table
     */
    char T[1<<BT][BT][BT],G[MX];
    short B[SZ];
    int X,I[SZ],E[SZ],O=-1,dp[SZ][LG];
    //Precaltucaliton of table T: easy simulation of process for each possible bitmask
    //Complexity is B^2*2^B
    void tb(int B){
        F(1<<B)FF(B+1){
            int d=0,b=0,I=j;
            FT(j,B+1){
                T[i][j][k]=I;
                d+=i&1<<k?1:-1;
                if(b>d)b=d,I=k+1;
            }
        }
    }
    //Makes bitmask of an interval
    //It is just BT-1 (not BT) since we are interested in differences only
    int bt(int*A){
```

```
58          int b=0;
59          //Check difference and shift it to i-th position
60          F(BT-1)b|=(A[i+1]>A[i])<<i;
61          return b;
62      }
63      //Body of function which calls other functions - setting proper values to arrays
64      void ini(int N,int*A){
65          if(!X++){tb(BT-1);FT(1,MX)G[k]=O+=P2(k);}
66          F(BT*2)A[N+i]=A[N+i-1]+1;
67          F(N/BT+1)E[i]=*min_element(A+i*BT,A+i*BT+BT),I[i]=min_element(A+i*BT,A+i*BT+BT)-A,B[i]=bt(A+i*
                BT);
68          ST(N/BT+1);
69      }
70      //Function which calculates SPARSE TIBLE (of sizeN/BT)
71      void ST(int N){
72          F(N)dp[i][0]=i;
73          //Precalcute so that there is index to array "E".
74          //Length 2^i: Minimal value from index i to i+2^i-1
75          //Counting intervals from smaller to bigger
76          FT(1,k-(1<<k)+N+1)F(N+1-(1<<k))
77              if(E[dp[i][k-1]]<E[dp[i+(1<<(k-1))][k-1]])
78                  dp[i][k]=dp[i][k-1];
79              else dp[i][k]=dp[i+(1<<(k-1))][k-1];
80      }
81      //Returns minimal value on L - R interval
82      //j: logarithm of interval-size
83      //Taking interval from L to L+2^j-1 and from R-2^j+1 to R, and takes
84      //minimum of thos (it will surely cover the whole interval)
85      int QQ(int L,int R,int*A){
86          int j=G[R-L+1];return A[dp[L][j]]<=A[dp[R-(1<<j)+1][j]]?dp[L][j]:dp[R-(1<<j)+1][j];
87      }
88      inline int sc(int a){return a-a%BT;}
89      //Query to interval of size BT
90      //Determines borders of interval
91      void qp(int L,int R,int&b,int&J,int*A){
92          int u=T[B[L/BT]][L%BT][R%BT];
93          if(b>A[sc(L)+u])b=A[J=sc(L)+u];
94      }
95      //Function which solves RMQ on interval
96      //It solves it in multiple steps: it solves it on the BIG interval + it solves the
97      //borders of it
98      //All call are O(1)
99      int qy(int L,int R,int*A){
100         int a,b=INF,J=-1;
101         if(L/BT==R/BT)return qp(L,R,b,J,A),J;
102         if(L%BT)a=L,L=sc(L)+BT,qp(a,L-1,b,J,A);
103         if(R%BT+1!=BT)a=R,R=sc(R)-1,qp(R+1,a,b,J,A);
104         if(L<=R&&E[a=QQ(L/BT,R/BT,E)]<b)b=a,J=I[a];
105         return J;
106     }
107 };
108 struct LCA{
109     /*
110      * R: RMQ for +/-1
111      * g: Edges
```

```
     * l: Size of E
     * E: Flattened tree in rder of Euler Tour (with nesting)
     * L: Depth of node
     * H: Index of first position of node int 'E'
     * D: Depth of node. With parallel to 'E'
     */
    RMQ R;
    vi g[MX];
    int l,E[MX],L[MX],H[MX],D[MX];
    //dfs for euler tour: Node, Parent, Depth, Indext to E
    void dfs(int u,int p,int d,int&l){
        H[u]=l,L[u]=d++,E[l]=u,D[l++]=L[u];
        for(auto&h:g[u])if(h^p)dfs(h,u,d,l),E[l]=u,D[l++]=L[u];
    }
    //Init - nothing interesting (just calls)
    void pre(int N,int r=0){dfs(r,r,0,l=0),R.ini(l,D);}
    //Returns LCA: Returns value of E on index given by RMQ
    int lca(int a,int b){a=H[a],b=H[b];if(a>b)swap(a,b);return E[R.qy(a,b,D)];}
    //Adds edge
    void ADD(int A,int B){g[A].PB(B),g[B].PB(A);}
    //Clears graph
    void CLR(){for(auto&h:g)h.clear();}
}L;
```

6.2 线段树

```
namespace sgt {
    long long chg[1000010];
    int son[1000010][2],cnt,root;
    void build(int &x,int l,int r) {
        x=++cnt;
        if(l==r) return ;
        int mid=(l+r)>>1;
        build(son[x][0],l,mid);
        build(son[x][1],mid+1,r);
    } void update(int a,int b,int k,int l,int r,long long v) {
        if(a>b || l>r)return;
        if(a<=l && b>=r)chg[k]+=v;
        else {
            int mid=(l+r)>>1;
            if(b<=mid)update(a,b,son[k][0],l,mid,v);
            else if(a>mid)update(a,b,son[k][1],mid+1,r,v);
            else update(a,mid,son[k][0],l,mid,v),update(mid+1,b,son[k][1],mid+1,r,v);
        }
    } long long query(int a,int k,int l,int r,long long v) {
        if(l==r)return v+chg[k]+val[a];
        int mid=(l+r)>>1;
        if(a<=mid)return query(a,son[k][0],l,mid,v+chg[k]);
        else return query(a,son[k][1],mid+1,r,v+chg[k]);
    }
}
```

6.3 LCA

垃圾lca

```
int lca(int x, int y){
    if(depth[x] > depth[y]) swap(x, y);
    while(depth[x] != depth[y]){
        y = fa[y];
    } while(x!=y){
        x = fa[x]; y = fa[y];
    } return x;
}
```

倍增LCA

```
void dfs(int x){
    depth[x] = depth[fa[x][0]]+1;
    for(int i = 0; fa[x][i]; i++)
        fa[x][i+1] = fa[fa[x][i]][i];
    for(int i = head[x]; i != -1; i=edge[i].next){
        int to = edge[i].to;
        if(!depth[to]){
            fa[to][0] = x;
            dis[to] = dis[x] + edge[i].cost;
            dfs(to);
        }
    }
}

int Lca(int x, int y){
    if(depth[x] > depth[y]) swap(x, y);
    for(int i = 20; i >= 0; i--)
        if(depth[fa[y][i]] >= depth[x])
            y = fa[y][i];
    if(x == y) return y;
    for(int i = 20; i >= 0; i--){
        if(fa[y][i] != fa[x][i]){
            y = fa[y][i];
            x = fa[x][i];
        }
    }
    return fa[x][0];
}
```

6.4 树状数组

逆序对数

```
#include<bits/stdc++.h>
#define ll long long
const int maxn = 5e5+10;
using namespace std;
int bit[maxn], n;
struct Num {
    int val,pos;
    bool operator <(const Num &a)const {
        return val < a.val;
```

```
    }
}num[maxn];

int lowbit(int x){ return x&(-x); }
int sum(int i) {
    int res = 0;
    while(i > 0) {
        res += bit[i];
        i -= lowbit(i);
    } return res;
}

void add(int i, int x) {
    while(i <= n) {
        bit[i] += x;
        i += lowbit(i);
    }
}

int main(){
    while(~scanf("%d",&n) && n){
        memset(bit, 0, sizeof bit);
        ll ans = 0;
        for(int i = 0; i < n; i++){
            scanf("%d",&num[i].val);
            num[i].pos = i+1;
        } sort(num, num + n);
        for(int i = 0; i < n; i++){
            int tmp = sum(num[i].pos);//小于他的个数
            ans += i - tmp;
            add(num[i].pos, 1);
        } printf("%lld\n",ans);
    } return 0;
}
```

二维树状数组

```
int lb(int x){
    return x&(-x);
}

int sum(int x, int y){
    int res = 0;
    for(int i = x; i > 0; i -= lb(i))
        for(int j = y; j > 0; j -= lb(j))
            res += bit[i][j];
    return res;
}

void add(int x, int y, int k){
    for(int i = x; i < maxn; i += lb(i))
        for(int j = y; j < maxn; j += lb(j))
            bit[i][j] += k;
}
```

7 Dynamic Program

7.1 背包问题

7.1.1 01背包

hdu 2955 01背包

要求在不被抓的概率下偷到更多的东西,概率p是double不能作为背包的空间,考虑m因为n*m¡100000.所以只要找最大的可以满足ans[i]¿1-p的i;

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<cmath>
using namespace std;

int main(){
    int t;scanf("%d",&t);
    double p1;int n;
    double ans[111111],p[111];
    int m[111];
    while(t--){
        scanf("%lf%d",&p1,&n);
        memset(ans, 0, sizeof ans);
        ans[0]=1;
        int sum = 0;
        //cout << n << endl;
        for(int i = 0; i < n; i++){
            scanf("%d%lf",&m[i],&p[i]);
            sum += m[i];
            p[i] = 1 - p[i];
        }
        for(int i = 0; i < n; i++){
            for(int j = sum; j >= 0; j--){
                ans[j] = max(ans[j], ans[j-m[i]]*p[i]);
            }
        }
        int maxn = 0;
        for(int i = sum; i >= 0; i--){
            if(ans[i]>1-p1){
                maxn = i;break;
            }
        }
        printf("%d\n",maxn);
    }
    return 0;
}
```

普通dp,https://vjudge.net/problem/HDU-1864,虽然有点像01背包,但是写的时候只要考虑每增加一种物品与前面的最优解的关系dp[i] = max(dp[j]+sum[i], dp[i]),

```
#include<cstdio>
#include<iostream>
#include<cstring>
#include<cmath>
#include<algorithm>
using namespace std;
```

```
7
8  double bag[300];
9
10 double max(double a, double b){
11     if(a>b)return a;
12     return b;
13 }
14
15 int main(){
16     double edge;int n;
17     while(~scanf("%lf%d",&edge,&n) && n){
18         memset(bag, 0 , sizeof bag);
19         int k;
20         double sum[1000],summ=0; int tot=0;
21         for(int i = 0; i < n; i++){
22             bool flag = 0;
23             summ=0;
24             scanf("%d",&k);
25             char a;double money;
26             double cal[30]={0};
27             for(int j = 0; j < k; j++){
28                 scanf(" %c:%lf",&a,&money);
29                 summ+=money;
30                 if(money>600)flag = 1;
31                 cal[a-'A']+=money;
32                 if(!flag){
33                     for(int i = 0; i < 3; i++)
34                         if(cal[i]>600)flag = 1;
35                     if(a!='A' && a!='B' && a!='C')flag=1;
36                     if(summ>1000)flag=1;
37                 }
38             } if(!flag)sum[++tot]=summ;
39         } for(int i = 1; i <= tot; ++i){
40             for(int j = 0; j <= i; j++){
41                 if(bag[j] + sum[i] <= edge){
42                     bag[i] = max(bag[i], bag[j]+sum[i]);
43                 }
44             }
45         } double maxn = 0;
46         for(int i = 0; i <= tot; i++){
47             maxn = max(maxn, bag[i]);
48         } printf("%.2lf\n",maxn);
49     } return 0;
50 }
```

7.1.2 完全背包

这里的怪物是无限的,但是存在最多杀多少个怪的限制.构造dp[k][s],代表杀s个怪物消耗k耐久度.
所以有dp[j][t+1] = max(dp[j][t+1] , dp[j-耐久度][t]+经验值)
在这里,便利杀怪和便利杀第几怪是没有关联的,也就是说他们的顺序是可以交换的

```
1 #include<cstdio>
2 #include<iostream>
3 #include<cstring>
4 using namespace std;
```

```

int n,m,k,s;
int thing[111][2];
int dp[111][111];
int main(){
    while(~scanf("%d%d%d%d",&n,&m,&k,&s)){
        for(int i = 0; i < k; i++){
            scanf("%d%d",&thing[i][0],&thing[i][1]);
        } memset(dp, 0 ,sizeof dp);
        for(int i = 0; i < k; i++) {
            for(int j = thing[i][1]; j <= m; j++) {
                for(int t = 0; t < s; t++)
                    dp[j][t+1] = max(dp[j-thing[i][1]][t]+thing[i][0],dp[j][t+1]);
            }
        }
        int ans = -1;
        bool flag = 0;
        for(int i = 0; i <= m; i++){
            for(int j = 0; j <= s; j++){
                if(dp[i][j]>=n){
                    ans = m-i; flag = 1; break;
                }
            } if(flag)break;
        } printf("%d\n",ans);
    } return 0;
}
```

7.1.3 多重背包

```
procedure MultiplePack(cost,weight,amount)
    if cost*amount>=V
        CompletePack(cost,weight)
        return
    integer k=1
    while k<amount
        ZeroOnePack(k*cost,k*weight)
        amount=amount-k
        k=k*2
    ZeroOnePack(amount*cost,amount*weight)
```

例题

```
#include<cstdio>
#include<algorithm>
#include<iostream>
#include<cstring>
using namespace std;

int a[111],c[111],dp[111111];
int n,m;

void CompletePack(int v){
    for(int i = v; i <= m; i++)
        dp[i]=max(dp[i],dp[i-v]+v);
}
```

```
14
15 void ZeroOnePack(int v){
16     for(int i = m; i>=v; i--){
17         dp[i]=max(dp[i],dp[i-v]+v);
18     }
19 }
20
21 //多重背包
22 void MultiplePack(int v, int n){
23     if(v*n>=m){
24         CompletePack(v);
25         return ;
26     }
27     int k=1;
28     while(k<n){
29         ZeroOnePack(k*v);
30          n -= k;
31          k <<= 1;
32     }
33     ZeroOnePack(n*v);
34 }
35
36 int main(){
37     while(~scanf("%d%d",&n,&m)&& n){
38         for(int i = 0; i < n; i++)
39             scanf("%d",&a[i]);
40         for(int i = 0; i < n; i++)
41             scanf("%d",&c[i]);
42         memset(dp, 0, sizeof dp);
43         for(int i = 0; i < n; i++){
44             MultiplePack(a[i], c[i]);
45         }
46         int cont = 0;
47         for(int i = 1; i <= m; i++)
48             if(dp[i]==i)cont++;
49         printf("%d\n",cont);
50     }
51     return 0;
52 }
```

习题:

1. 有n¡1000堆石头，第i堆有ai和bi属性，每次拿一堆(假设第i堆)后，所有的石头的a值都减去bi.求最后拿到的a的和的最大值。

直接暴力求解

```
1 #include<iostream>
2 #include<cstring>
3 #include<cstdio>
4 #include<algorithm>
5 #include<cmath>
6 using namespace std;
7
8 int a[1111],b[1111];
9 int he[1111];
10 bool cmp(int a, int b){
11     return a>b;
```

```
}
int main(){
    int n;
    while(~scanf("%d",&n) && n){
        for(int i = 0; i < n; i++){
            scanf("%d%d",&a[i],&b[i]);
        } int ans = -999999;
        for(int i = 0; i <= n; i++){
            for(int j = 0; j < n; j++){
                he[j] = a[j]-b[j]*i;
            } int temp = 0;
            sort(he, he+n,cmp);
            for(int k = 0; k < i; k++){
                temp += he[k];
            } ans = max(temp, ans);
        } printf("%d\n",ans);
    } return 0;
}
```

2. 有n堆石头，每拿一堆（假设是第i堆），没有被拿的石头堆的a都要减去bi,求能够拿的a的和的最大值。
a值的顺序对结果没有影响,有影响的是b,从小到大排序.dp[i][j]表示选第i堆时还要选j堆.
dp[i][j] = max(dp[i-1][j],dp[i-1][j+1]+a[i]-b[i]*j)

```
#include<cstdio>
#include<iostream>
#include<cstring>
#include<algorithm>
#define maxn 1100
#define INF 0x3f3f3f3f
using namespace std;

struct Inf{
    int a,b;
}save[maxn];

int dp[maxn][maxn];

bool cmp(struct Inf a, struct Inf b){
    return a.b<b.b;
}

int main(){
    int n;
    while(~scanf("%d",&n) && n){
        memset(save, 0, sizeof save);
        for(int i = 1; i <= n; i++)
            scanf("%d%d",&save[i].a,&save[i].b);
        sort(save+1, save+n+1, cmp);
        memset(dp, -INF, sizeof dp);
        for(int i = 0; i <= n; i++)dp[0][i]=0;
        for(int i = 1; i <= n; i++){
            for(int j = 0; j <= n; j++){
                dp[i][j] = max(dp[i-1][j], dp[i-1][j+1]+save[i].a-save[i].b*j);
            }
        }
        printf("%d\n",dp[n][0]);
```

```
    }
    return 0;
}
```

7.2 分组背包

习题2:13件装备,戒指可以带两个.盾和武器是单手的.

这里可以先把盾和武器合并放在双手武器中,两个戒指也再合并一下.但是出现了只有一个盾或者只有一个剑的情况,由于数量少,可以把单个剑,单把盾也放在双手武器中,一个戒指也算一对戒指.

然后就是一个背包的问题了.分组背包.

```
for 所有的组k
    for v=V..0
        for 所有的i属于组k
            f[v]=max{f[v],f[v-c[i]]+w[i]}
```

这里我们可以把v设定为耐久度,如果超过v也算作是v,这样减小了内存.

```
#include<cstdio>
#include<iostream>
#include<cstring>
#include<vector>
using namespace std;
char name[20][20]={"Head", "Shoulder", "Neck", "Torso", "Hand", "Wrist", "Waist", "Legs", "Feet", "
    Finger", "Shield", "Weapon", "Two-Handed"};
int t,n,m;
//记录装备
struct Good{
    int d, h;
    Good(int d,int h):d(d), h(h){}
};
//从装备类型到编号
int getnum(char *s){
    for(int i = 0; i < 13; i ++){
        if(strcmp(s, name[i])==0)
            return i;
    }
}

vector<Good> v[13];
int dp[15][50001];

int main(){
    char s[20];scanf("%d",&t);
    int n,d,h,edge;
    while(t--){
        for(int i = 0; i < 13; i++)v[i].clear();
        scanf("%d%d",&n,&edge);
        for(int i = 0; i < n; i++){
            scanf("%s%d%d",s,&d,&h);
            int num = getnum(s);
            v[num].push_back(Good(d,h));
            //把武器盾放在双手持器中
            if(num==11 || num== 10){
                v[12].push_back(Good(d,h));
            }
```

```
        }

        for(int i = 0; i < v[11].size(); i++){
            for(int j = 0; j < v[10].size(); j++){
                v[12].push_back(Good(v[11][i].d+v[10][j].d, v[11][i].h+v[10][j].h));
            }
        }
        //清空武器和盾,并存放戒指的结合
        v[10].clear();v[11].clear();
        for(int i = 0; i < v[9].size(); i++){
            v[10].push_back(v[9][i]);
            for(int j = i+1; j < v[9].size(); j++){
                v[10].push_back(Good(v[9][i].d+v[9][j].d, v[9][i].h+v[9][j].h));
            }
        }
        v[9].clear();

        memset(dp, -1, sizeof dp);
        dp[13][0] = 0;
/*
        for(int i = 0; i < v[12].size(); i++){
            Good g = v[12][i];
            int vh = g.h > edge?edge: g.h;
            int vd = g.d;
            dp[11][vh] = max(dp[11][vh], vd);
        }
*/
        for(int k = 12; k >= 0; k--){
            for(int j = 0; j <= edge; j++){
                dp[k][j] = max(dp[k][j], dp[k+1][j]);
                if(dp[k+1][j] == -1)continue;
                for(int i = 0; i < v[k].size(); i++){
                    Good g = v[k][i];
                    int vh = (g.h+j)>edge?edge:(g.h+j);
                    int vd = g.d + dp[k+1][j];
                    dp[k][vh] = max(dp[k][vh], vd);
                }
            }
        } printf("%d\n",dp[0][edge]);
    }
}
```

习题三:分组背包lcm;

一个数n分成$a_1+a_2+...+a_k$, 求最大的划分使得$lcm(a_1,a_2,...,a_k)$最大.首先确认a之间是两两互质的,否则可以马上提出他的gcd()作为一个新的数.

所以最后的答案其实就是$p_1^{k_1}p_2^{k_2}...p_s^{k_s}$,而我们只要找符合条件的最大值就行.考虑dp.也就是完全背包.

首先打一个质数表.然后从第一位开始更新.因为如果是所有的质数lcm,会炸llong long,考虑用对数记录最大值.

第一层考虑第几个质数,第二层考虑weight,由于可以分解为1,也就是说背包不需要填满,从n开始是因为每一个质数的n次方中只能取一个,这就想当是一个分组背包了.

按照伪代码写.并跟新ans

```
#include<cstdio>
#include<iostream>
#define ll long long
#include<bitset>
#include<cmath>
```

```cpp
#define maxn 3600
using namespace std;

ll gcd(ll a, ll b){
    return b?gcd(b,a%b):a;
}

bitset<maxn> judge;
int prime[maxn];
int tot=0;
void init(){
    for(int i = 2; i < maxn; i++){
        if(!judge[i]){
            prime[tot++]=i;
            for(int j = i+i; j < maxn; j+=i){
                judge[j]=1;
            }
        }
    }
}

double dp[maxn];
ll ans[maxn];

int main(){
    init();
    int n,m;
    while(~scanf("%d %d",&n,&m)){
        for(int i = 0; i <= n; i++){
            dp[i]=0;
            ans[i]=1;
        }
        for(int i = 0; i < tot && prime[i]<=n; i++){
            double tmp = log(prime[i]*1.0);
            for(int j = n; j >= prime[i]; j--){
                for(ll k = prime[i], q=1; k <= j; k*=prime[i], q++){
                    if(dp[j-k]+q*tmp>dp[j]){
                        dp[j] = dp[j-k]+q*tmp;
                        ans[j] = ans[j-k]*k%m;
                    }
                }
            }
        }
        printf("%lld\n",ans[n]);
    }
    return 0;
}
```

习题四,完全背包有取值上限的.

n组ai,bi中挑选m组使得和ai-和bi的绝对值最大,如果有相同的,选ai+bi和最大的那种,输出m组ai的和和bi的和,并且输出相应的组数.

```cpp
#include<cstdio>
#include<iostream>
#include<cstring>
#include<cmath>
```

```cpp
#include<algorithm>
#include<vector>
using namespace std;

int dp[22][888];//dp[j][k]:取j个候选人，使其辩控差为k的所有方案中，辩控和最大的方案的辩控和
vector<int> path[22][888];
int n,m;
int p[222],d[222],s[222],v[222];

int main(){
    int t=0;
    while(~scanf("%d%d",&n,&m) && n){
        for(int i = 0; i <= m; i++){
            for(int j = 0; j < 881 ; j++)path[i][j].clear();
        } memset(dp, -1, sizeof dp);
        for(int i = 1; i <= n; i++){
            scanf("%d %d", &p[i], &d[i]);
            s[i] = p[i] + d[i];
            v[i] = p[i] - d[i];
        }
        //修正.
        int fix = m*20;
        //即真正的dp[0][0]
        dp[0][fix]=0;
        int i,j,k;
        for(i = 1; i <= n; i++)
        for(j = m; j > 0; j--)
        for(k = 0; k < 2*fix; k++){
            if(dp[j-1][k] < 0)continue;
            if(dp[j][k+v[i]] <= dp[j-1][k] + s[i]){
                dp[j][k+v[i]] = dp[j-1][k] + s[i];
                path[j][k+v[i]] = path[j-1][k];
                path[j][k+v[i]].push_back(i);
            }
        }
        //output
        int div;
        for(i = 0; dp[m][fix+i]==-1 && dp[m][fix-i]==-1; i++);
        div = (dp[m][fix+i]>dp[m][fix-i])?i:-i;
        printf("Jury #%d\n",++t);
        printf("Best jury has value %d for prosecution and value %d for defence:\n",(dp[m][fix+div] +
            div)/2,(dp[m][div+fix]-div)/2);
        for(i = 0; i < m; i++)
            printf(" %d",path[m][fix+div][i]);
        puts("");puts("");
    } return 0;
}
```

7.2.1 多重集组合数DP

题目: 有n种物品, 第i种物品有a个. 不同种类的物品可以互相区分, 但相同种类的无法区分.
从这些物品中取出m个, 有多少种取法? 求出数模M的余数.
例如: 有n=3种物品, 每种a=1,2,3个, 取出m=3个, 取法result=6(0+0+3, 0+1+2, 0+2+1, 1+0+2, 1+1+1, 1+2+0).
dp[i][j]表示前i种物品一共拿了j个物品的方法数.
所以有$dp[i][j] = \sum_{k=0}^{min(j,x[i])} dp[i-1][j-k]$

这样的复杂度是O(nm*m)
分情况讨论:
1. $j <= x[i]$时
把最后一项单独拿出来,前面的进行合并于是就有$dp[i][j] = dp[i-1][j] + \sum_{k=0}^{j-1} dp[i-1][j-1-k]$
也就是说dp[i][j] = dp[i-1][j] + dp[i-1][j]
1. $j > x[i]$时
这时存在$dp[i][j] = dp[i-1][j] + \sum_{k=0}^{x[i]-1} dp[i-1][j-1-k]$
即dp[i][j] = dp[i-1][j] + dp[i][j-1] - dp[i][j-1-x[i]]
递推式:

```
for(int i = 0; i < n; i++)for(int j = 1; j <= m; j++)
if(j>a[i])dp[i+1][j] = (dp[i][j] + dp[i+1][j-1] - dp[i][j-1-x[i]] + mod)%mod;
else dp[i+1][j] = (dp[i][j] +dp[i+1][j-1])%mod;
```

7.3 状压dp

```
//n为总元素个数,m为选中个数,i为状态。
for (int i = (1<<m)-1; i < (1<<n);){
    for (int j = 0; j < n; j++) ans = max(ans, dp[i][j]);
    int x = i&-i;int y = i+x;
    i = ((i&~y)/x>>1)|y;
}

for(i=s;i;i=(i-1)&s);//枚举s子集
```

7.4 数位dp

```
#include<cstdio>
#include<iostream>
#include<cstring>
#define ll long long
const int maxn = 17;
using namespace std;
ll dp[maxn], a[maxn], ten[maxn] = {1};

void init(){
    for(int i = 1; i < maxn; i++) ten[i] = ten[i-1]*10;
}

int cnt(int n){
    int ans = 0;
    while(n){
        ans += ((n%10)==1); n /= 10;
    } return ans;
}

ll dfs(int pos, ll num, int v, int limit){
    if(pos==-1)return v;
    if(!limit && dp[pos]!=-1)return dp[pos]+v*ten[pos+1];
    ll ans = 0;
    int ed = (limit?a[pos]:9);
    for(int i = 0; i <= ed; i++){
        ans += dfs(pos-1, (num*10+i), v+(i==1), limit&&(i==ed));
    } if(!limit)dp[pos] = ans;
```

```
    return ans;
}

ll slove(int x){
    int pos = 0;
    memset(dp, - 1, sizeof dp);
    while(x){
        a[pos++] = x%10;
        x /= 10;
    } return dfs(pos-1,0,0,1);
}

int main(){
    int n;
    init();
    while(~scanf("%d",&n)){
        printf("%lld\n",slove(n));
    } return 0;
}
```

7.5 常见dp

最大字段和：给定一个数列a, 求最大的连续字段和的最大值

```
ll slove(ll a[]) {
    memset(dp, 0, sizeof dp);
    for(int i = 1; i <= n; i++) {
        dp[i] = max(dp[i], dp[i-1]+a[i]);
    } return *max_element(dp, dp+n+1);
}
```

循环字段和：$a[i], a[i+1], a[i+2], ...a[j\%n]$

```
#include <bits/stdc++.h>
const int N = 5e4+9;
using namespace std;
using ll = long long;
ll dp[N], n, a[N];
ll slove(ll a[]) {
    memset(dp, 0, sizeof dp);
    for(int i = 1; i <= n; i++) {
        dp[i] = max(dp[i], dp[i-1]+a[i]);
    } return *max_element(dp, dp+n+1);
}

int main() {
    scanf("%lld", &n); ll sum = 0;
    for (int i = 1; i <= n; i++) scanf("%lld", &a[i]), sum += a[i];
    ll ans = slove(a);
    for (int i = 1; i <= n; i++) a[i] = -a[i];
    return printf("%lld\n", max(ans, sum+slove(a))), 0;
}
```

最大子矩阵和：一个M*N的矩阵种找一个子矩阵且这个和是最大的，输出和

枚举行的上界和下界，这样就变成了最大子矩阵和

```
#include<bits/stdc++.h>
```

```
const int N = 555;
using namespace std;
using ll = long long;
ll a[N][N];
int n, m, t;
int main() {
    while(~scanf("%d%d",&m,&n)) {
        memset(a,0,sizeof(a));
        for(int i=1;i<=n;i++) {
            for(int j=1;j<=m;j++) {
                scanf("%d",&t); a[i][j]=a[i-1][j]+t;
            }
        } ll ans=0;
        for(int i=1;i<=n;i++) {
            for(int j=i;j<=n;j++) {
                ll sum=0;
                for(int k=1;k<=m;k++) {
                    sum+=(a[j][k]-a[i-1][k]);
                    sum=max(sum, 0LL); ans=max(ans, sum);
                }
            }
        } printf("%lld\n", ans);
    }
}
```

最小正子段和

```
#include <bits/stdc++.h>
const int N = 5e4+9;
using namespace std;
using ll = long long;
ll a[N], n;
int idx[N];
bool cmp(int x, int y) { return a[x] < a[y]; }

int main() {
    scanf("%lld", &n);
    for (int i = 1; i <= n; i++) scanf("%lld", &a[i]), a[i]+=a[i-1], idx[i]=i;
    sort(idx, idx+n+1, cmp);
    ll ans = 1e9+7;
    for (int i = 1; i <= n; i++) {
        if(idx[i]>idx[i-1] && a[idx[i]] > a[idx[i-1]]) ans = min(ans, a[idx[i]]-a[idx[i-1]]);
    } printf("%lld\n", ans);
}
```

最大M字段和 v2

N个整数组成的序列a[1],a[2],a[3],…,a[n]，将这N个数划分为互不相交的M个子段，并且这M个子段的和是最大的。如果M ¿= N个数中正数的个数，那么输出所有正数的和。例如：-2 11 -4 13 -5 6 -2，分为2段，11 -4 13一段，6一段，和为26。

```
#include <bits/stdc++.h>
const int N = 2e5+9;
using namespace std;
using ll = long long;

int n, m, t, l[N], r[N];
ll Heap[N], si;
```

```
ll v[N], x;

bool cmp(int a, int b) { return abs(v[a]) > abs(v[b]); }
inline void push(int x) { Heap[++si] = x; push_heap(Heap+1, Heap+1+si, cmp); }
inline void pop() { pop_heap(Heap+1, Heap+1+si--, cmp); }
inline void Merge(int L, int R, int p) {
    v[p]+=v[L]+v[R]; L = l[L], R=r[R];
    l[p]=L; r[L]=p; l[R]=p; r[p]=R;
}

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) {
        scanf("%lld", &x);
        if(!x) continue;
        if(v[t]*x<=0) v[++t] = x;
        else v[t] += x;
    } ll ans = 0; n = t;
    l[0]=0; r[0]=1; l[t+1]=n; r[t+1]=t+1;
    for (int i = 1; i <= t; i++) {
        l[i] = i-1; r[i] = i+1; push(i);
        if(v[i] > 0) m--, ans += v[i];
    }
    while(m<0) {
        int p = Heap[1], lp=l[p], rp=r[p]; pop();
        if(r[lp] != p || l[rp] != p) continue;
        if(v[p] < 0 && (lp < 1 || n < rp)) continue;
        if(!v[p]) continue;
        ans -= abs(v[p]); m++; Merge(lp, rp, p); push(p);
    } return printf("%lld\n", ans), 0;
}
```

最大子段和 V2

N个整数组成的序列a[1],a[2],a[3],⋯,a[n]，你可以对数组中的一对元素进行交换，并且交换后求a[1]至a[n]的最大子段和，所能得到的结果是所有交换中最大的。当所给的整数均为负数时和为0。例如：-2,11,-4,13,-5,-2, 4将 -4 和 4 交换，-2,11,4,13,-5,-2, -4，最大子段和为$11 + 4 + 13 = 28$。

```

#include <bits/stdc++.h>
const int N = 5e4+9;
using namespace std;
using ll = long long;
int n;
pair<ll, ll> st;
ll inf = 1LL<<55;
ll a[N], mx[N], sum[N], ans;

void slove(){
    a[n+1]=-inf;
    for (int i = n; i; i--) {
        mx[i] = max(mx[i+1], a[i]);
        sum[i] = sum[i+1] + a[i];
    }
    int k = 0;
    ll pre=-inf, aft=-inf;
    for (int i = 1; i <= n; i++) {
```

```
        pre = max(pre, sum[i]);
        aft = max(aft, pre-a[i]);
        ans = max(ans, aft - sum[i+1] + mx[i+1]);
    }
}

int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        scanf("%lld", &a[i]);
    } slove(); reverse(a+1, a+n+1); slove();
    return printf("%lld\n", ans), 0;
}
```

最大字段和 V3

环形最大M子段和，N个整数组成的序列排成一个环，a[1],a[2],a[3],…,a[n]（a[n-1], a[n], a[1]也可以算作1段），将这N个数划分为互不相交的M个子段，并且这M个子段的和是最大的。如果M ¿= N个数中正数的个数，那么输出所有正数的和。例如：-2 11 -4 13 -5 6 -1，分为2段，6 -1 -2 11一段，13一段，和为27。

```
#include <bits/stdc++.h>
const int N = 2e5+9;
using namespace std;
using ll = long long;

int n, m, t, l[N], r[N];
ll Heap[N], si;
ll v[N], x;

bool cmp(int a, int b) { return abs(v[a]) > abs(v[b]); }
inline void push(int x) { Heap[++si] = x; push_heap(Heap+1, Heap+1+si, cmp); }
inline void pop() { pop_heap(Heap+1, Heap+1+si--, cmp); }
inline void Merge(int L, int R, int p) {
    v[p]+=v[L]+v[R]; L = l[L], R=r[R];
    l[p]=L; r[L]=p; l[R]=p; r[p]=R;
}

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) {
        scanf("%lld", &x);
        if(!x) continue;
        if((v[t]<=0 && x>0) || (v[t]>=0 && x <0)) v[++t] = x;
        else v[t] += x;
    } ll ans = 0;
    if(v[1]*v[t]>=0 && t > 1) { v[1] += v[t]; t--; } n = t;
    for (int i = 1; i <= t; i++) {
        l[i] = i-1; r[i] = i+1; push(i);
        if(v[i] > 0) m--, ans += v[i];
    } l[1] = t; r[t] = 1;
    while(m<0) {
        int p = Heap[1], lp=l[p], rp=r[p]; pop();
        if(r[lp] != p || l[rp] != p) continue;
        if(v[p] < 0 && (lp < 1 || n < rp)) continue;
        if(!v[p]) continue;
        ans -= abs(v[p]); m++; Merge(lp, rp, p); push(p);
    } return printf("%lld\n", ans), 0;
```

```
}
```

LCS 最长公共子序列

```
#include <bits/stdc++.h>

using namespace std;
const int N = 1e3+9;
char s1[N], s2[N];
int dp[N][N], n, m;
vector<char> ans;

int main() {
    while(~scanf("%s%s",s1+1,s2+1)) {
        memset(dp, 0, sizeof dp);
        n = strlen(s1+1); m = strlen(s2+1);
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= m; j++) {
                if(s1[i] == s2[j]) {
                    dp[i][j] = dp[i-1][j-1] + 1;
                } else dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
            }
        } ans.clear();
        int len = dp[n][m];
        while(dp[n][m]) {
            if(dp[n][m] == dp[n-1][m]) n--;
            else if(dp[n][m] == dp[n][m-1]) m--;
            else ans.push_back(s1[n]), n--, m--;
        }
        for (int i = len-1; ~i; i--) {
            printf("%c", ans[i]);
        } puts("");
    } return 0;
}
```

8 Geometry

9 Tech

9.1 bitset

bitset可以优化背包，竞赛图之类的东西。

```
bitset<17>BS;
BS[1] = BS[7] = 1;
cout<<BS._Find_first()<<endl; // prints 1

BS[1] = BS[7] = 1;
cout<<BS._Find_next(1)<<','<<BS._Find_next(3)<<endl; // prints 7,7

for(int i=BS._Find_first();i< BS.size();i = BS._Find_next(i))
    cout<<i<<endl;
```

9.2 笛卡尔树

```cpp
int build(){
    int tp=0;
    for (int i = 1; i <= n; i++) l[i]=r[i]=vis[i]=0;
    for (int i = 1; i <= n; i++) {
        int k = tp;
        while(k > 0&&a[stk[k-1]]<a[i]) --k;
        if(k) r[stk[k-1]] = i;
        if(k < tp) l[i] = stk[k];
        stk[k++] = i;
        tp = k;
    } for (int i = 1; i <= n; i++) {
        vis[l[i]] = vis[r[i]] = 1;
    } for (int i = 1; i <= n; i++) if(!vis[i]) return i;
}
```

9.3 昌昌的快读

```cpp
namespace fastIO{
#define BUF_SIZE 100000
#define OUT_SIZE 100000
#define ll long long
  //fread->read
  bool IOerror=0;
  inline char nc(){
    static char buf[BUF_SIZE],*p1=buf+BUF_SIZE,*pend=buf+BUF_SIZE;
    if (p1==pend){
      p1=buf; pend=buf+fread(buf,1,BUF_SIZE,stdin);
      if (pend==p1){IOerror=1;return -1;}
      //{printf("IO error!\n");system("pause");for (;;);exit(0);}
    }
    return *p1++;
  }
  inline bool blank(char ch){return ch==' '||ch=='\n'||ch=='\r'||ch=='\t';}
  inline void read(int &x){
    bool sign=0; char ch=nc(); x=0;
    for (;blank(ch);ch=nc());
    if (IOerror)return;
    if (ch=='-')sign=1,ch=nc();
    for (;ch>='0'&&ch<='9';ch=nc())x=x*10+ch-'0';
    if (sign)x=-x;
  }
  inline void read(ll &x){
    bool sign=0; char ch=nc(); x=0;
    for (;blank(ch);ch=nc());
    if (IOerror)return;
    if (ch=='-')sign=1,ch=nc();
    for (;ch>='0'&&ch<='9';ch=nc())x=x*10+ch-'0';
    if (sign)x=-x;
  }
  inline void read(double &x){
    bool sign=0; char ch=nc(); x=0;
    for (;blank(ch);ch=nc());
    if (IOerror)return;
    if (ch=='-')sign=1,ch=nc();
```

```
38    for (;ch>='0'&&ch<='9';ch=nc())x=x*10+ch-'0';
39    if (ch=='.'){
40      double tmp=1; ch=nc();
41      for (;ch>='0'&&ch<='9';ch=nc())tmp/=10.0,x+=tmp*(ch-'0');
42    }
43    if (sign)x=-x;
44  }
45  inline void read(char *s){
46    char ch=nc();
47    for (;blank(ch);ch=nc());
48    if (IOerror)return;
49    for (;!blank(ch)&&!IOerror;ch=nc())*s++=ch;
50    *s=0;
51  }
52  inline void read(char &c){
53    for (c=nc();blank(c);c=nc());
54    if (IOerror){c=-1;return;}
55  }
56  //fwrite->write
57  struct Ostream_fwrite{
58    char *buf,*p1,*pend;
59    Ostream_fwrite(){buf=new char[BUF_SIZE];p1=buf;pend=buf+BUF_SIZE;}
60    void out(char ch){
61      if (p1==pend){
62        fwrite(buf,1,BUF_SIZE,stdout);p1=buf;
63      }
64      *p1++=ch;
65    }
66    void print(int x){
67      static char s[15],*s1;s1=s;
68      if (!x)*s1++='0';if (x<0)out('-'),x=-x;
69      while(x)*s1++=x%10+'0',x/=10;
70      while(s1--!=s)out(*s1);
71    }
72    void println(int x){
73      static char s[15],*s1;s1=s;
74      if (!x)*s1++='0';if (x<0)out('-'),x=-x;
75      while(x)*s1++=x%10+'0',x/=10;
76      while(s1--!=s)out(*s1); out('\n');
77    }
78    void print(ll x){
79      static char s[25],*s1;s1=s;
80      if (!x)*s1++='0';if (x<0)out('-'),x=-x;
81      while(x)*s1++=x%10+'0',x/=10;
82      while(s1--!=s)out(*s1);
83    }
84    void println(ll x){
85      static char s[25],*s1;s1=s;
86      if (!x)*s1++='0';if (x<0)out('-'),x=-x;
87      while(x)*s1++=x%10+'0',x/=10;
88      while(s1--!=s)out(*s1); out('\n');
89    }
90    void print(double x,int y){
91      static ll mul[]={1,10,100,1000,10000,100000,1000000,10000000,100000000,
92        1000000000,10000000000LL,100000000000LL,1000000000000LL,10000000000000LL,
```

```
        1000000000000000LL,10000000000000000LL,100000000000000000LL,1000000000000000000LL};
      if (x<-1e-12)out('-'),x=-x;x*=mul[y];
      ll x1=(ll)floor(x); if (x-floor(x)>=0.5)++x1;
      ll x2=x1/mul[y],x3=x1-x2*mul[y]; print(x2);
      if (y>0){out('.'); for (size_t i=1;i<y&&x3*mul[i]<mul[y];out('0'),++i); print(x3);}
    }
    void println(double x,int y){print(x,y);out('\n');}
    void print(char *s){while (*s)out(*s++);}
    void println(char *s){while (*s)out(*s++);out('\n');}
    void flush(){if (p1!=buf){fwrite(buf,1,p1-buf,stdout);p1=buf;}}
    ~Ostream_fwrite(){flush();}
  }Ostream;
  inline void print(int x){Ostream.print(x);}
  inline void println(int x){Ostream.println(x);}
  inline void print(char x){Ostream.out(x);}
  inline void println(char x){Ostream.out(x);Ostream.out('\n');}
  inline void print(ll x){Ostream.print(x);}
  inline void println(ll x){Ostream.println(x);}
  inline void print(double x,int y){Ostream.print(x,y);}
  inline void println(double x,int y){Ostream.println(x,y);}
  inline void print(char *s){Ostream.print(s);}
  inline void println(char *s){Ostream.println(s);}
  inline void println(){Ostream.out('\n');}
  inline void flush(){Ostream.flush();}
#undef ll
#undef OUT_SIZE
#undef BUF_SIZE
};
using namespace fastIO;
```

9.4 快速计算1-n质数个数

```
namespace pcf {
#define clr(ar) memset(ar,0,sizeof(ar))
#define chkbit(ar,i) (((ar[(i)>>6])&(1<<(((i)>>1)&31))))
#define setbit(ar,i) (((ar[(i)>>6])|=(1<<(((i)>>1)&31))))
#define isprime(x) (((x)&&((x)&1)&&(!chkbit(ar,(x))))||((x)==2))
const int MAXN=100;
const int MAXM=100010;
const int MAXP=666666;
const int MAX=10000010;
    ll dp[MAXN][MAXM];
    unsigned int ar[(MAX>>6)+5]={0};
    int len=0,primes[MAXP],counter[MAX];
    void Sieve() {
        setbit(ar,0),setbit(ar,1);
        for(int i=3;(i*i)<MAX;i++,i++) {
            if(!chkbit(ar,i)) {
                int k=i<<1;
                for(int j=(i*i);j<MAX;j+=k) setbit(ar,j);
            }
        } for(int i=1;i<MAX;i++) {
            counter[i]=counter[i-1];
            if(isprime(i)) primes[len++]=i,counter[i]++;
```

```
        }
    }
    void init() {
        Sieve();
        for(int n=0;n<MAXN;n++) {
            for(int m=0;m<MAXM;m++) {
                if(!n) dp[n][m]=m;
                else dp[n][m]=dp[n-1][m]-dp[n-1][m/primes[n-1]];
            }
        }
    }
    ll phi(ll m, int n) {
        if(n==0) return m;
        if(primes[n-1]>=m) return 1;
        if(m<MAXM&&n<MAXN) return dp[n][m];
        return phi(m,n-1)-phi(m/primes[n-1],n-1);
    }
    ll Lehmer(ll m) {
        if(m<MAX) return counter[m];
        ll w,res = 0;
        int i,a,s,c,x,y;
        s=sqrt(0.9+m),y=c=cbrt(0.9+m);
        a=counter[y],res=phi(m,a)+a-1;
        for(i=a;primes[i]<=s;i++) res=res-Lehmer(m/primes[i])+Lehmer(primes[i])-1;
        return res;
    }
}
```

9.5 Int_128

```
struct Int_128{
    unsigned long long a,b;
    Int_128(long long x) {
        a=0,b=x;
    }
    friend bool operator < (Int_128 x,Int_128 y) {
        return x.a < y.a || x.a == y.a && x.b < y.b ;
    }
    friend Int_128 operator + (Int_128 x,Int_128 y) {
        Int_128 re(0);
        re.a=x.a+y.a+(x.b+y.b<x.b);
        re.b=x.b+y.b;
        return re;
    }
    friend Int_128 operator - (Int_128 x,Int_128 y) {
        y.a=~y.a;y.b=~y.b;
        return x+y+1;
    }
    void Div2() {
        b>>=1;b|=(a&1ll)<<63;a>>=1;
    }
    friend Int_128 operator * (Int_128 x,Int_128 y) {
        Int_128 re=0;
        while(y.a || y.b) {
```

```
            if(y.b&1) re=re+x;
            x=x+x; y.Div2();
        } return re;
    }
    friend Int_128 operator % (Int_128 x,Int_128 y) {
        Int_128 temp=y;
        int cnt=0;
        while(temp<x)
            temp=temp+temp,++cnt;
        for(;cnt>=0;cnt--) {
            if(temp<x)
                x=x-temp;
            temp.Div2();
        } return x;
    }
};
```

9.6 最大割

```
int mp[N][N], in[N], ou[N], n, ans;
void dfs(int k, int s, int ni, int no) {
    int sum = 0;
    if(s >= ans) return;
    for (int i = 1; i <= ni; i++) sum += mp[in[i]][k];
    if(k == n) { if(s + sum < ans) ans = s + sum; }
    else { in[ni+1] = k; dfs(k+1, s+sum, ni+1, no); }
    sum = 0;
    for (int i = 1; i <= no; i++) sum += mp[ou[i]][k];
    if(k == n) { if(s + sum<ans) ans = s+sum; }
    else { ou[no+1] = k; dfs(k+1, s+sum, ni, no+1); }
}

int main() {
    scanf("%d", &n); ans = 0;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++) {
            scanf("%d", &mp[i][j]);
            ans += mp[i][j];
        }
    int k = ans/2, p=0,q=0;
    for (int i = 1; i < n/2; i++) {
        for (int j = 1; j < n/2; j++) p += mp[i][j];
    } p /= 2;
    for (int i = n/2; i <= n; i++) {
        for (int j = n/2; j <= n; j++) q += mp[i][j];
    } q /= 2;
    ans = p+q; in[1] = 1; dfs(2, 0, 1, 0);
    printf("%d\n", k - ans);
}
```

9.7 等差数列异或和

```
ll get(ll l, ll p, ll num, ll dis) {
```

```
    ll ans = (l/p)*num + (dis/p)*num*(num-1)/2;
    l %= p; dis %= p;
    if(dis*num+l < p) return ans;
    return ans + get((dis*num+l)%p, dis, (dis*num+l)/p, p);
}
l,r公差为dis;
ll getSum(ll l, ll r, ll dis) {
    ll num = (r - l) / dis + 1, ans=0, sum;
    for (ll i = 1; i <= r; i<<=1) {
        sum = get(l, i, num, dis);
        if(sum & 1) ans += i;
    } return ans;
}
```

9.8 折半

礼物折半

```
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>
#include <vector>
#include <map>

using namespace std;

const int N = 31;
int v[N], w[N];
int s[16][10000], cnt[16];
int n, t;

int slove() {
    memset(cnt, 0, sizeof cnt);
    int l = (n+1)/2, r = n/2;
    for (int i = 0; i < (1<<l); i++) {
        int ret = 0, num = 0;
        for (int j = 0; j < l; j++) {
            if(i&(1<<j)) ret += v[j], num++;
            else ret -= w[j];
        }
        s[num][cnt[num]++] = ret;
    }
    for (int i = 0; i <= l; i++) sort(s[i], s[i]+cnt[i]);
    int ans = 0x3f3f3f3f;
    for (int i = 0; i <(1<<r); i++) {
        int ret = 0, num = 0;
        for (int j = 0; j < r; j++) {
            if(i&(1<<j)) ret -= v[j+l], num++;
            else ret += w[j+l];
        }
        int pos = lower_bound(s[l-num], s[l-num]+cnt[l-num], ret) - s[l-num];
        if(pos >= cnt[l-num] && cnt[l-num] > 0) ans = min(ans, abs(s[l-num][cnt[l-num]-1] - ret));
        else {
            ans = min(ans, abs(s[l-num][pos] - ret));
```

```
            if(pos > 0) ans = min(ans, abs(s[l-num][pos-1] - ret));
        }pos = lower_bound(s[r-num], s[r-num]+cnt[r-num], ret) - s[r-num];
        if(pos >= cnt[r-num] && cnt[r-num] > 0) ans = min(ans, abs(s[r-num][cnt[r-num]-1] - ret));
        else {
            ans = min(ans, abs(s[r-num][pos] - ret));
            if(pos > 0) ans = min(ans, abs(s[r-num][pos-1] - ret));
        }
    }
    printf("%d\n", ans);
}

int main() {
    scanf("%d", &t);
    while(t --) {
        scanf("%d", &n);
        for (int i = 0; i < n; i++) scanf("%d", &v[i]);
        for (int i = 0; i < n; i++) scanf("%d", &w[i]);
        slove();
    }
}
```

平衡树

```
const int MAXN = 100010;
const int MAXLOG = 20;
const int MAXNODE = MAXN * MAXLOG;
namespace trie{
    int ch[MAXNODE][2], sz[MAXNODE], id = 2;
    inline void ins(int x, int d) {//插入
        int i, u = 1;
        for(i = MAXLOG - 1; i >= 0; i--){
            bool v = (x >> i) & 1;
            if(!ch[u][v]) ch[u][v] = newnode();
            u = ch[u][v];
            sz[u] += d;
        }
    }
    inline int nlt(int x) {// 找x的下标
        int i, u = 1, ret = 0;
        for(i = MAXLOG - 1; i >= 0; i--) {
            bool v = (x >> i) & 1;
            if(v) ret += sz[ ch[u][0] ];
            u = ch[u][v];
            if(!u) break;
        }
        return ret;
    }
    inline int kth(int k) {// 找第K大
        int i, u = 1, ret = 0;
        for(i = MAXLOG - 1; i >= 0; i--) {
            int lsz = sz[ ch[u][0] ];
            if( k <= lsz ) u = ch[u][0];
            else k -= lsz, ret |= (1 << i), u = ch[u][1];
        }
        return ret;
    }
```

```
34      int newnode() {//配合清零
35          ch[id][0] = ch[id][1] = sz[id] = 0;
36          return id++;
37      }
38      void clear(){//清零
39          ch[1][0] = ch[1][1] = 0;
40          id = 2;
41      }
42  }
```