Standard Code Library

123ZDQ

South China University of Technology

August 20, 2022

Contents

开始

定义

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

typedef long long LL;
typedef __int128 LLL;
typedef unsigned u32;
typedef unsigned long long u64;
typedef long double LD;
typedef pair<int,int> pii;
typedef pair<LL,LL> pll;

#define pln putchar('\n')
#define For(i,a,b)  for(int i=(a),(i##i)=(b);i<=(i##i);++i)
#define Fodn(i,a,b) for(int i=(a),(i##i)=(b);i>=(i##i);--i)

const int M=1000000007,INF=0x3f3f3f3f;
const long long INFLL=0x3f3f3f3f3f3f3f3fLL;
const int N=1000009;
```

取消流同步

此时混用 `iostream` 和 `cstdio` 会寄

```cpp
ios::sync_with_stdio(0);cin.tie(0);
```

整数读写

`read()` 读有符号数极限数据可能溢出未定义行为 `write()` 最长能写 `__int128`

```cpp
template <typename T>
bool read(T &x) {
    x = 0; char c = getchar(); int f = 1;
    while (!isdigit(c) && (c != '-') && (c != EOF)) c = getchar();
    if (c == EOF) return 0;
    if (c == '-') f = -1, c = getchar();
    while (isdigit(c)) { x = x * 10 + (c & 15); c = getchar();}
    x *= f; return 1;
}

template <typename T, typename... Args>
bool read(T &x, Args &...args) {
    bool res = 1;
    res &= read(x);
    res &= read(args...);
    return res;
}

template <typename T>
void write(T x) {
    if (x < 0) x = -x, putchar('-');
    static char sta[55];
    int top = 0;
    do sta[top++] = x % 10 + '0', x /= 10; while (x);
    while (top) putchar(sta[--top]);
}
```

数学

快速模乘

- |a|,|b|<m，且 m 在 `longlong` 范围
- 编译环境支持 64 个二进制位精度的 `long doube`（如 x86-64 平台的 GNU G++）
- 替代方案 `__int128`，不行则改用 log 复杂度的龟速乘

```cpp
inline LL mul(LL a, LL b, LL m) {
    u64 r = (u64)a * b - (u64)((LD)a / m * b + 0.5L) * m;
    return r < m ? r : r + m;
}
```

快速幂

- longlong 范围用 fpl

```cpp
inline LL fp(LL a, LL b, LL Mod) {
    LL res = (Mod != 1);
    for (; b; b >>= 1, a = a * a % Mod)
        if (b & 1)
            res = res * a % Mod;
    return res;
}

inline LL fpl(LL a, LL b, LL Mod) {
    LL res = (Mod != 1);
    for (; b; b >>= 1, a = mul(a, a, Mod))
        if (b & 1)
            res = mul(res, a, Mod);
    return res;
}
```

辗转相除 (求最大公约数)

- 实际上 gcd 可以用 <algorithm> 的库函数 __gcd 代替

```cpp
template <typename T>
inline T gcd(T a, T b) {
    while (b){T t = b;b = a % b;a = t;}
    return a;
}

template <typename T>
inline T lcm(T a, T b) { return a / gcd(a, b) * b; }

template<typename T>
inline void exgcd(T a,T b,T&x,T&y,T&d){
    if (b == 0){ x = 1, y = 0, d = a;}
    else{
        exgcd(b, a % b, y, x, d);
        y -= (a / b) * x;
    }
}
```

乘法逆元

用欧拉定理 (费马小定理) 求逆元代码最短

exgcd 求逆元

```cpp
inline int getinv(int a, int p){
    int m = 0, n = 1, x = 1, y = 0, b = p;
    while (b){
        int q = a / b;
        tie(x, m) = make_tuple(m, x - q * m);
        tie(y, n) = make_tuple(n, y - q * n);
        tie(a, b) = make_tuple(b, a - q * b);
    }
    (x < 0) && (x += p);
    return x;
}
```

线性递推求逆元

模 n 意义下，设正整数 i 和 $n\%i$ 均存在逆元，则有递推式：

- $$inv(i) = -(n/i) * inv(n\%i)$$

其中 / 和 % 是 C++ 运算符

一般用于模质数时线性求出 $1..n$ 的逆元

线性同余方程组（中国剩余定理）

- 用于合并方程组 $x \equiv a_i (mod\ m_i)$
- 合法运行的数据:
- – $\{a\}$ 非负
- – $\{m\}$ 为正
- – $2lcm\{m_i\}$ 在 `longlong` 范围
- 用 `pair<LL,LL>` 存 $\{a_i, m_i\}$ 前置算法 快速乘和 辗转相除
- 有解返回 True，并将第二个方程合并到第一个（得到最小非负解）；否则返回 False
- 注意题目求最小非负解还是正数解!!!

```
inline bool crt(pll &a, pll b){
    LL g, u, d;
    exgcd(a.second, b.second, u, d, g);
    d = b.first - a.first;if (d % g) return 0;
    const LL t = b.second / g;
    a.first += mul(u, d / g, t) * a.second;
    a.second *= t;
    a.first %= a.second; (a.first < 0) && (a.first += a.second);
    return 1;
}
```

大指数运算（欧拉降幂）

$$a^x = \begin{cases} a^{x\ mod \phi(m)} & (a,m) = 1 \\ a^x & (a,m)! = 1\ x < c \\ a^{(x-c) mod\ c+c} & (a,m)! = 1\ x \geq c \end{cases}$$

其中 $c = m$ $+1$，是一个 $O(log\ n)$ 的常数，为计算方便常用 $\phi(m)$ 代替

在快速幂中特殊处理，若 $b^c \geq \phi(m)$，返回 $b^{c\ mod\ \phi(m)+\phi(m)}$，否则返回 $b^{c\ mod\ \phi(m)}$

```
inline LL exfp(LL a,LL b,LL Mod){
    /* 注意检查 Mod<=1e9 且 res 初值为 1*/
    bool f=0;LL res=1;
    for(;b;b>>=1){
        if(a>Mod){f=1;a%=Mod;}
        if(b&1){
            res=res*a;
            if(res>Mod){f=1;res%=Mod;}
        }
        a=a*a;
    }return f?res+Mod:res;
}
```

原根

- 只有 $2, 4, p^a, 2p^a$ 有原根，这里的 p 是奇质数
- (王元) 质数 p 的最小正原根是 $O(p^{0.25+\epsilon})$ 的可暴力 check
- 设 g 为 m 的原根，对任意 x 满足 $(x, \phi(m)) = 1$，g^x 也是原根，故原根共有 $\phi(\phi(m))$ 个
- (原根判定定理) m 的某个既约剩余类 g 为原根 $\iff$ 对 $\phi(m)$ 的任意质约数 p 有 $g^{\frac{\phi(m)}{p}} \neq 1 (mod\ m)$
- 用于生成任意 (有上限) 次单位根，做 NTT（number theoretic transforms）

高次同余方程

- 以下 p 均为正整数

离散对数（BSGS/exBSGS）

求最小非负整数 x 满足 $a^x \equiv b(mod\ p)$，$p \leq 10^9$ 且不必是质数

- 前置算法 快速幂 `map/unorded_map` `__gcd()` 求逆元
  - 哈希表 `typedef unordered_map<int,int,neal> HASH;`
- 视 p 范围换快速乘由于复杂度 $O(\sqrt(p))$，实际上难以对一般的 long long 做 BSGS ~~多半是想法歪子~~
- 若多组询问 a,p 为常量且始终满足 (b,p)=1 则应复用 h 降低常数
- 接口传入非负整数 a b 和正整数 p
  - `log()` 要求 $(a,p)=1$ 无解返回-1
  - `operator()` 无特殊要求无解返回-1
- 求最小正整数需稍作修改见注释

```
struct{
    int log(int a,int b,int p){
        if((b-1)%p==0)return 0;
        HASH h;const int t=(int)sqrt(p)+1;
        for(int j=0;j<t;++j,b=(LL)b*a%p)h[b]=j;
        int at=(a=fp(a,t,p));
        for(int i=1;i<=t;++i,a=(LL)a*at%p){
            auto it=h.find(a);
            if(it!=h.end())return i*t-(*it).second;
        }return -1;
    }
    int operator()(int a,int b,int p){//(接口)
        if((b-1)%p==0)return 0;//求正整数解则注释掉
        if((b-a)%p==0)return 1;
        a%=p;b%=p;
        const int _a=a,_b=b,_p=p;
        int d=__gcd(a,p),ax=1,t=0;
        while(d>1){
            if(b%d!=0)break;
            b/=d;p/=d;ax=(LL)ax*(a/d)%p;++t;
            d=__gcd(a,p);
        }
        for(int i=2,at=(LL)_a*_a%_p;i<=t;++i,at=(LL)at*_a%_p)if(at==_b)return i;
        if(d>1)return -1;
        int res=log(a,(LL)b*getinv(ax,p)%p,p);//res 为-1 说明无解 有解则非负
        return res<0?-1:res+t;//求正整数解则要修改
    }
}BSGS;
```

模意义下开根（N 次剩余）

求 x 满足 $x^a \equiv b(mod\ m)$

- 如果：$(b,m)=1$ 且容易计算 m 的原根 g 与 $\phi(m)$
  - 计算离散对数 $y=\log_g b$ （瓶颈所在）
  - 将问题转化为 $x^a=(g^{x'})^a \equiv g^y(mod\ m)$
  - 等价于线性同余方程 $ax' \equiv y(mod\ \phi(m))$

~~对于一般的情形，不会~~

模奇质数的平方根/立方根

二次剩余的判定

- 设 n 是奇质数 p 的一个既约剩余类

- $n^{\frac{p-1}{2}} mod\ p = \begin{cases} 1 & n \\ -1 & n \end{cases}$

平方根（Cipolla 算法）

求最小非负整数 x 满足 $x^2 \equiv n\ (mod\ p)$ p 为不超过 10^9 的质数 n 为非负整数 * 若有两根则另一根为 $p - x$ 若无解返回 -1

- 期望复杂度 $O(log\ p)$ 对 10^{18} 以内的 p 需稍作修改见注释
- 接口 `int ans1 = Square_root(p)(n);`

```
struct Square_root{
    typedef int DAT; //操作数类型
    const DAT P;
    DAT I2;
    Square_root(DAT p = 1) : P(p) {}
    DAT mul(DAT a, DAT b) const{ return (LL)a * b % P;} // DAT=int
    /*DAT mul(DAT a, DAT b) const{
        u64 r = (u64)a * b - (u64)((LD)a / P * b + 0.5L) * P;
        return r < P ? r : r + P;
    }*/ // DAT=long long

    #define X first
    #define Y second
    typedef pair<DAT, DAT> pii;
    pii mul(pii a, pii b) const { return {
        (mul(a.X, b.X) + mul(mul(I2, a.Y), b.Y)) % P,
        (mul(a.X, b.Y) + mul(a.Y, b.X)) % P};}

    template <class T>
    T pow(T a, DAT b, T x){
        for (; b; b /= 2, a = mul(a, a)) if (b & 1)x = mul(x, a);
        return x;
    }

    DAT operator()(DAT n){
        if ((n %= P) <= 1) return n;
        if (pow(n, (P - 1) / 2, (DAT)1) == P - 1) return -1;
        DAT a;
        do a = rand(); // 随机方法可能寄
        while (pow(I2 = (mul(a, a) - n + P) % P, (P - 1) / 2, (DAT)1) == 1);
        DAT x = pow(pii{a, 1}, (P + 1) / 2, {1, 0}).X;
        return min(x, P - x);
    }
    #undef X
    #undef Y
};
```

立方根

求一个非负整数 x 满足 $x^3 \equiv n(mod\ p)$ 无解返回-1

- 复杂度期望 $O(log\ p)$ (暂时不会求最小解/全部解)
- 接口 `int ans2 = Cube_root(n, p)();`
- 封装的 `int` 版本换 `long long` 见注释

```
struct Cube_root{
    typedef int DAT;//操作数类型
    const DAT n,p;Cube_root(DAT _n=1,DAT _p=1):n(_n%_p),p(_p){};
    struct Z3{DAT x,y,z;};
    DAT mul(DAT a,DAT b)const{return (LL)a*b%p;}  //DAT=int
    //DAT mul(DAT a,DAT b)const{u64 r=(u64)a*b-(u64)((LD)a/p*b+0.5L)*p;return r<p?r:r+p;}  //DAT=long long
    Z3 mul(Z3 a,Z3 b)const{return(Z3){
        (mul(a.x,b.x)+mul((mul(a.y,b.z)+mul(a.z,b.y))%p,n))%p,
        ((mul(a.x,b.y)+mul(a.y,b.x))%p+mul(mul(a.z,b.z),n))%p,
        ((mul(a.x,b.z)+mul(a.y,b.y))%p+mul(a.z,b.x))%p
    };}
    template<class T>T pow(T a,DAT b,T x)
        {for(;b;b/=2,a=mul(a,a))if(b&1)x=mul(x,a);return x;}
```

```
    DAT operator()(){
        if(n==0||p<=3)return n;
        if(p%3==2)return pow(n,(2*p-1)/3,(DAT)1);
        if(pow(n,(p-1)/3,(DAT)1)!=1)return -1;
        Z3 r;
        do r=pow((Z3){rand(),rand(),rand()},(p-1)/3,(Z3){1,0,0});
        while(r.x!=0||r.y==0||r.z!=0); // 随机方法可能寄
        return pow(r.y,p-2,(DAT)1);
    }
};
```

线性筛

```
int ntp[N],pri[N],tot;
inline void linear_sieve(int n){
    ntp[1]=1;for(int i=2;i<=n;++i)ntp[i]=0;tot=0;//初始化 一般可省
    for(int i=2;i<=n;++i){
        if(!ntp[i])pri[++tot]=i;
        for(int j=1;j<=tot;++j){
            const LL nex=(LL)i*pri[j];if(nex>n)break;//n<=1e6 则不必开 LL
            ntp[nex]=1;
            if(i%pri[j]==0)break;
        }
    }
}
```

欧拉函数 $\phi(x)$

- 求 $\phi(x)$ 的值可以归约为找 x 的质因子
- $O(\sqrt{x})$ 求点值

```
template <typename T>
inline T phi(T x) {
    T res = x;
    for (T i = 2; i * i <= x; ++i)
        if ((x % i) == 0) {
            res = res / i * (i - 1);
            while ((x % i) == 0) x /= i;
        }
    if (x > 1) res = res / x * (x - 1);
    return res;
}
```

- 或者线性筛出 $\phi(1..n)$

数论分块

对于给定的正整数 n 和 i，使得

$$\lfloor\frac{n}{i}\rfloor=\lfloor\frac{n}{j}\rfloor$$

成立的最大正整数（满足 $i<=j<=n$）的 j 的值为 $\lfloor\frac{n}{\lfloor\frac{n}{i}\rfloor}\rfloor$

即 $\lfloor\frac{n}{i}\rfloor$ 所在块的右端点为 $\lfloor\frac{n}{\lfloor\frac{n}{i}\rfloor}\rfloor$

莫比乌斯反演

- 狄利克雷卷积 $*$ 的代数性质
  - 交换律、结合律
  - 对加法分配律
  - 单位元 $\epsilon=[n==1]$
  - 对 $f(1)\neq 0$ 的 f 存在逆元 g
  - 两个积性函数的狄利克雷卷积仍是积性函数
- $\mathbb{1}*\mu=\epsilon$
- $\mathbb{1}*\phi=id$

- 设 f 和 g 是数论函数 (N->R)，且当 $n > M$ 时有 $f(n) = g(n) = 0$，则
  - $f(n) = \sum_{n|m} g(m) \iff g(n) = \sum_{n|m} \mu(\frac{m}{n}) f(m)$

杜教筛

在学子

给定函数 f 和一个较大的 n，要计算其前缀和 $S(n) = \sum_{i=1}^{n} f(i)$，考虑构造一个函数 g 与 f 作卷积，有

$\sum_{i=1}^{n}(f*g)(i) = \sum_{i=1}^{n}\sum_{xy=i} f(x)g(y) = \sum_{y=1}^{n} g(y)S(\lfloor\frac{n}{y}\rfloor)$

即 $g(1)S(n) = \sum_{i=1}^{n}(f*g)(i) - \sum_{y=2}^{n} g(y)S(\lfloor\frac{n}{y}\rfloor)$

复杂度:

- $(f*g)$ 的前缀和与 g 的区间和都可以快速计算（一般是推出可 O(1) 计算的表达式，或者与 S 同时用杜教筛计算）
- $S(n)$ 可以线性复杂度预处理出前 $N^{\frac{2}{3}}$ 项，后面的项用记忆化搜索

总复杂度 $O(N^{\frac{2}{3}})$

洛谷某 dalao 某模板题代码 Orz

```
typedef long long LL;
const int maxn = 2147483647;
const int maxm = 2000000;

LL S1[maxm], S2[1300];
bool vis[1300];
int N;

LL S(int n) {
  if (n < maxm) return S1[n];
  int x = N / n; // 如果存在某个 x 使得 n = floor(N / x),
                 // 选 x = floor(N / n) 一定可以。
  if (vis[x]) return S2[x];
  vis[x] = true;
  LL &ans = S2[x];
  ans = (LL)n * (n + 1) / 2; // 对 (f*g)(n) = n 求和
  for (int i = 2, j; i <= n; i = j + 1) {
    j = n / (n / i);
    ans -= (j - i + 1) * S(n / i);
  }
  return ans;
}
```

Miller-Rabin 素性测试

loj143 luoguP4718

- $n <= 10^{18}$
- 需要 快速乘 64 位快速幂
- http://miller-rabin.appspot.com/
- 此实现仍有被 hack 可能如果 WA 就换 base:
-
  - 2^64 以内保证正确的 7 个 base: 2, 325, 9375, 28178, 450775, 9780504, 1795265022

```
inline bool MR(LL a,const LL&d,const int&t,const LL&n){//注意 a%n==0 时应判定通过测试
    a%=n;if(a==0)return 1;
    a=fpl(a,d,n);
    if(a==1||a==(n-1))return 1;
    for(int i=1;i<t;++i){
        a=mul(a,a,n);
        if(a==(n-1))return 1;
        if(a==1)return 0;
    }
    return 0;
}
```

```
constexpr LL a[]={2, 123635709730000, 9233062284813009, 43835965440333360, 761179012939631437, 1263739024124850375};
inline bool isPrime(LL n){
    if(n==2||n==3)return 1;if(n<2||(n%2==0)||(n%3==0))return 0;//特判最好别省
    if(n==585226005592931977LL)return 0;//这 6 个基的最小反例 暂不清楚 1e18 内有无其它反例
    LL d=n-1;int t=0;while((d&1)==0)d>>=1,++t;
    for(int i=0;i<6;++i)if(!MR(a[i],d,t,n))return 0;
    return 1;
}
```

Pollard-Rho 分解质因数

- 需要 快速乘 gcd
- 返回 n 的一个大于 1 的真因子不存在就死循环寄!
- 务必在调用前确认 n 是合数 (>1)

```
mt19937 mt(time(0)); //随机化
inline LL PR(LL n) {
    LL x = uniform_int_distribution<LL>(0, n - 1)(mt), s, t, c = uniform_int_distribution<LL>(1, n - 1)(mt); //随机化
    for (int gol = 1; 1; gol <<= 1, s = t, x = 1) {
        for (int stp = 1; stp <= gol; ++stp) {
            t = (mul(t, t, n) + c) % n;
            x = mul(x, abs(s - t), n);
            if ((stp & 127) == 0) {
                LL d = gcd(x, n);
                if (d > 1) return d;
            }
        }
        LL d = gcd(x, n);
        if (d > 1) return d;
    }
}
```

组合数

- 数较小模数为较大质数求逆元
- – 如果模数固定可 O(n) 预处理阶乘逆元
- 数较大模数为较小质数用 $Lucas$ 定理
- –

$$C_n^m \equiv C_{\lfloor \frac{n}{p} \rfloor}^{\lfloor \frac{m}{p} \rfloor} * C_{n \bmod p}^{m \bmod p} (mod\ p)$$

- 数较大模数较小但不保证为质数用 $exLucas$ 算法求 $C_n^m mod\ P$

exLucas

luoguP4720

- 前置算法 快速乘 快速幂 `crt` 合并同余方程
- 单次询问 O(P log P)
- 不要求 P 为质数

```
namespace EXLUCAS {
    inline LL g(LL n, LL p) {
        LL nn = n;
        while (n > 0) nn -= (n % p), n /= p;
        return nn / (p - 1);
    }

    LL f(LL n, LL p, LL pk) {
        if (n == 0) return 1;
        LL res = 1;
        if (n >= pk) {
```

```
            LL t = n / pk, k = 1, els = n - t * pk;
            for (LL i = 1; i <= els; ++i) if (i % p) k = k * i % pk;
            res = k;
            for (LL i = els + 1; i < pk; ++i) if (i % p) k = k * i % pk;
            res = res * fp(k, n / pk, pk) % pk;
        }
        else for (LL i = 1; i <= n; ++i) if (i % p) res = res * i % pk;
        return res * f(n / p, p, pk) % pk;
    }

    inline LL exlucas(LL n, LL m, LL p, LL pk, LL k) {
        LL a = f(n, p, pk) * fp(f(n - m, p, pk) * f(m, p, pk) % pk, pk / p * (p - 1) - 1, pk) % pk;
        LL b = g(n, p) - g(m, p) - g(n - m, p);
        if (b >= k) return 0;
        while (b--) a *= p;
        return a % pk;
    }

    /* 接口 */ inline LL exlucas(LL n, LL m, LL p){
        pll r = {0, 1};
        for (LL i = 2; i * i <= p; ++i)if (p % i == 0){
            LL t = 0, pk = 1;
            while (p % i == 0)++t, p /= i, pk *= i;
            crt(r, {exlucas(n, m, i, pk, t), pk});
        }
        if (p > 1)crt(r, {exlucas(n, m, p, p, 1), p});
        return r.first;
    }
}
```

类欧几里得算法

洛谷模板题

- 计算直线下整点数
- f=∑[(ai+b)/c] g=∑i[(ai+b)/c] h=∑[(ai+b)/c]^2 i=0..n a,b,n∈N c∈N*
- 复杂度 log(MAX{a,c})

```
struct dat{LL f, g, h;};
const LL i2 = 499122177, i3 = 332748118, M = 998244353; //预处理出模 M 意义下 2 和 3 的逆元
dat f(LL a, LL b, LL c, LL n){
    LL ac = a / c, bc = b / c;
    LL n2 = (n * (n + 1) % M) * i2 % M, n3 = n2 * (2ll * n + 1) % M * i3 % M;
    dat res = {
        (n2 * ac % M + (n + 1) * bc % M) % M,
        (ac * n3 % M + bc * n2 % M) % M,
        (ac * ac % M * n3 % M +
            bc * bc % M * (n + 1) % M + ac * bc % M * n2 % M * 2ll) % M};
    a %= c; b %= c; if (a == 0)return res;
    LL m = (a * n + b) / c;
    dat p = f(c, c - b - 1, a, m - 1);
    LL fc = (n * m % M - p.f + M) % M, gc = (n2 * m % M - i2 * (p.f + p.h) % M + M) % M;
    return{(res.f + fc) % M,    (res.g + gc) % M,
    (res.h + 2ll * (bc * fc % M + ac * gc % M) % M +
        n * m % M * m % M - 2ll * p.g - p.f + 3ll * M) % M};}
```

LOJ 模板题

~~在学子 QAQ~~

数值积分

- 自适应辛普森算法
- 复杂度 O(玄学)
- luogu p4542

```cpp
struct IG{
    typedef double Func(double);
    const Func*f;IG(const Func&g):f(&g){}
    double simpson(double l,double r)const{
        double mid=(l+r)/2;
        return (r-l)*(f(l)+4*f(mid)+f(r))/6;
    }
    double find(double l,double r,const double&EPS,double res)const{
        double mid=(l+r)/2;
        double fl = simpson(l, mid), fr = simpson(mid, r);
        if (abs(fl + fr - res) <= 15 * EPS)return fl + fr + (fl + fr - res) / 15;
        return find(l, mid, EPS / 2, fl) +find(mid, r, EPS / 2, fr);
    }
    double operator()(double l,double r,double EPS=1e-8)const{return find(l,r,EPS,simpson(l,r));}
};

struct Integration{
    typedef double Func(double);
    const Func*f;Integration(const Func&g):f(&g){}
    typedef pair<double,double> pdd;
    pdd add(pdd a,pdd b)const{
        double mid=(a.first+b.first)/2;
        return (pdd){mid,f(mid)};
    }
    #define simpson(p1,p2,p3) (((p3).first-(p1).first)*((p1).second+4*(p2).second+(p3).second)/6)
    double find(pdd p1,pdd p3,pdd p5,double EPS,double res,int dep)const{
        pdd p2=add(p1,p3),p4=add(p3,p5);
        double fl=simpson(p1,p2,p3),fr=simpson(p3,p4,p5),d=(fl+fr-res)/15;
        if(abs(d)<=EPS&&dep<0)return fl+fr+d;
        return find(p1,p2,p3,EPS/2,fl,dep-1)+find(p3,p4,p5,EPS/2,fr,dep-1);
    }
    double operator()(double l,double r,double EPS=1e-6/* 精度 */,int dep=12/* 最小递归深度 */)const{
        pdd p1(l,f(l)),p3(r,f(r)),p2=add(p1,p3);
        return find(p1,p2,p3,EPS,simpson(p1,p2,p3),dep);
    }
    #undef simpson
};
```

日期操作

用于跳转的常量

```cpp
const LL year_1[2]={365, 366};
const LL year_400=1460097;
const LL m_day[13]={(LL)0x3f3f3f, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

辅助函数

```cpp
inline bool isLeap(LL t){return (t % 400 == 0) ||((t % 4 == 0) && (t % 100));}
inline bool pick(LL a, LL b){return ((isLeap(a) && b <= 2) ||(isLeap(a + 1) && b > 2));}
inline LL dayThisMonth(LL y, LL m){return m_day[m] + isLeap(y) * (m == 2);}
```

日期和整数的一一对应

- LL 可以改成 int

```cpp
struct MY_DATE{
    LL year, month, day;
    MY_DATE(LL y = 2021, LL m = 1, LL d = 1) : year(y), month(m), day(d){};
    LL p(MY_DATE op = {0, 0, 0}){//日期转换为整数
        LL y = year - op.year, m = month - op.month, d = day - op.day;
        if (m <= 2){ y--; m += 12;}
        return 365 * y + y / 4 - y / 100 + y / 400 + (153 * (m - 3) + 2) / 5 + d - 307;
    }
    MY_DATE run(LL k){//当前日期过 k 天
        k += p();
        LL x = k + 1789995, n = 4 * x / 146097, i, j, d;
        x -= (146097 * n + 3) / 4;
        i = (4000 * (x + 1)) / 1461001;
        x -= 1461 * i / 4 - 31;
```

```
        j = 80 * x / 2447;
        d = x - 2447 * j / 80;
        x = j / 11;
        return MY_DATE(100 * (n - 49) + i + x, j + 2 - 12 * x, d);
    }
};
```

环染色问题

- poj2154 Burnside 引理

```
#include<cstdio>
using namespace std;
typedef long long LL;
#define pln putchar('\n')
const int N=100009;

int n,p,ans;

LL fp(LL a,LL b,LL Mod){
    LL res=(Mod!=1);
    for(;b;b>>=1,a=a*a%Mod)if(b&1)res=res*a%Mod;
    return res;
}

int ntp[N],pri[N],tot;

int f[33],t[33],d;

void dfs(int x,int phi,int u){
    if(x>d){
        ans=(ans+phi*fp(n,n/u-1,p))%p;
        return;
    }
    dfs(x+1,phi,u);
    u*=f[x];
    phi*=(f[x]-1);
    dfs(x+1,phi,u);
    for(int i=2;i<=t[x];++i){
        u*=f[x];
        phi*=f[x];
        dfs(x+1,phi,u);
    }
}

void solve(int T){
    scanf("%d%d",&n,&p);d=0;
    int x=n;
    for(int i=1;i<=tot;++i){
        if(pri[i]*pri[i]>x)break;
        if(x%pri[i])continue;
        int k=0;while(x%pri[i]==0)x/=pri[i],++k;
        f[++d]=pri[i];
        t[d]=k;
    }
    if(x>1){
        f[++d]=x;
        t[d]=1;
    }
    ans=0;
    dfs(1,1,1);
    printf("%d\n",ans);
}

signed main(){
    for(int i=2;i<=40000;++i){
        if(!ntp[i])pri[++tot]=i;
        for(int j=1;j<=tot;++j){
            int nex=pri[j]*i;
            if(nex>40000)break;
            ntp[nex]=1;
```

```
            if(i%pri[j]==0)break;
        }
    }
    int t;scanf("%d",&t);
    for(int i=1;i<=t;++i)solve(i);
    return 0;
}
```

拉格朗日插值

构造一个过给定 n 个点 $(x_1,y_1),(x_2,y_2),...,(x_n,y_n)$ 的多项式

$f(x)=\sum_{i=1}^{n}(y_i*\prod_{j\neq i}\frac{x-x_j}{x_i-x_j})$

曼哈顿距离与切比雪夫距离

设 $A(x_1,y_1)$，$B(x_2,y_2)$

曼哈顿距离 $d(A,B)=|x_1-x_2|+|y_1-y_2|=\max(|(x_1+y_1)-(x_2+y_2)|,|(x_1-y_1)-(x_2-y_2)|)$

这正是 (x_1+y_1,x_1-y_1) 与 (x_2+y_2,x_2-y_2) 的切比雪夫距离

因此将每个点 (x,y) 转化为 $(x+y,x-y)$，原坐标系的曼哈顿距离就是新坐标系下的切比雪夫距离

同理，将每个点 (x,y) 转化为 $(\frac{x+y}{2},\frac{x-y}{2})$，原坐标系的切比雪夫距离就是新坐标系下的曼哈顿距离

因子数 $d(n)$ 的数量级

- A066150 Maximal number of divisors of any n-digit number

4, 12, 32, 64, 128, 240, 448, 768, 1344, 2304, 4032, 6720, 10752, 17280, 26880, 41472, 64512, 103680, 161280, 245760, 368640, 552960, 860160, 1290240, 1966080, 2764800, 4128768, 6193152, 8957952, 13271040, 19660800, 28311552, 41287680, 59719680, 88473600, 127401984, 181665792, 264241152, 382205952, 530841600

置换群

- 可旋转涂色问题

给一个 n 元环涂色，m 种颜色，旋转后得到的方案算同一种，求不同的方案数

$\frac{1}{n}\sum_{i=0}^{n-1}m^{gcd(n,i)}=\frac{1}{n}\sum_{d|n}\phi(d)m^{n/d}$

轮换指标

设 x_i 表示置换中的 i 元环

- 正 n 边形的旋转群

$\frac{1}{n}\sum_{d|n}\phi(d)x_d^{n/d}$

- 正 n 边形的二面体群

$\frac{1}{2n}\sum_{d|n}\phi(d)x_d^{n/d}+\begin{cases}\frac{1}{2}x_1x_2^{\frac{n-1}{2}} & n\\ \frac{1}{4}(x_2^{\frac{n}{2}}+x_1x_2^{\frac{n-2}{2}}) & n\end{cases}$

- 正方体的顶点置换群

$\frac{1}{24}(x_1^8+8x_1^2x_3^2+9x_2^4+6x_4^2)$

- 正方体的边置换群

$\frac{1}{24}(x_1^{12}+8x_3^4+6x_1^2x_2^5+3x_2^6+6x_4^3)$

- 正方体的面置换群

$\frac{1}{24}(x_1^6+8x_3^2+6x_2^3+3x_1^2x_2^2+6x_1^2x_4)$

多项式快速幂

给定 $F(x)(mod\,x^n)$，求 $F^k(x)(mod\,x^n)$

k,n 不大，系数集不成域，$O(n\log n\log k)$ 分治 NTT

k 很大，且系数模奇质数 p ($p>n$):

$F[0]=1$, 根据 $f(x^p)\equiv f^p(x)(mod\,p)$，则 $F^p(x)\equiv F[0]=1(mod\,x^p)=1(modx^n)$, 直接令 $k\%=p$ 后计算 $\exp(k\ln F(x))$

$F[0]!=1$，提出最低次非 0 系数 ux^t 后化为上一种情形 $F^k(x)=u^kx^{tk}G^k(x)$，其中 $G[0]=1$，注意 u 的幂模 $p-1$，G 的幂模 p

广义二项式系数

$C(n,m)=\frac{n^{\underline{m}}}{m!}$，$n$ 是实数

$(1-x)^{-(d+1)}=\sum_{n\geq 0}C(d+n,n)x^n$，$d$ 是实数

Fibonacci 数列

$F_1=1,F_2=1,F_{n+2}=F_{n+1}+F_n$

$F_n=\frac{\sqrt{5}}{5}((\frac{1+\sqrt{5}}{2})^n-(\frac{1-\sqrt{5}}{2})^n)$

错排数

容斥原理：$n!\sum_{k=0}^{n}\frac{(-1)^k}{k!}$

盒子放球方案数

luoguP5824 记得检查 n<=m 之类的合法性条件

给定正整数 n,m，问 n 个球全部放入 m 个盒子的方案数，不同的限制条件如下

划分数 (与第二类 Stirling 数相关)

- 球之间互不相同，盒子之间互不相同。
- – m^n
- 球之间互不相同，盒子之间互不相同，每个盒子至多装一个球。
- – $m^{\underline{n}}$
- 球之间互不相同，盒子之间互不相同，每个盒子至少装一个球。
- – $m!S_2(n,m)$
- 球之间互不相同，盒子全部相同。
- – $\sum_{i=1}^{m}S_2(n,i)$
- 球之间互不相同，盒子全部相同，每个盒子至多装一个球。
- – $[n\leq m]$
- 球之间互不相同，盒子全部相同，每个盒子至少装一个球。
- – $S_2(n,m)$

一类不定方程解数

- 球全部相同，盒子之间互不相同。
- – $C(n+m-1,m-1)$
- 球全部相同，盒子之间互不相同，每个盒子至多装一个球。

- – $C(m,n)$
- 球全部相同，盒子之间互不相同，每个盒子至少装一个球。
- – $C(n-1,m-1)$

盒子有限的整数分拆

- 球全部相同，盒子全部相同。
- – $[x^n](\prod_{i=1}^{m}\frac{1}{1-x^i})$
- 球全部相同，盒子全部相同，每个盒子至多装一个球。
- – $[n\le m]$
- （整数的 m 分拆）球全部相同，盒子全部相同，每个盒子至少装一个球。
- – OGF：$n<m\,?\,0:[x^{n-m}](\prod_{i=1}^{m}\frac{1}{1-x^i})$
  – 记答案为 $p(n,m)$，递推式 $p(n,m)=p(n-1,m-1)+p(n-m,m)$

第二类 Stirling 数

n 个带标号球全部放入 m 个无标号盒子，且所有盒子非空的方案数

- $S_2(n,0)=[n=0]$
- $S_2(n,m)=S_2(n-1,m-1)+mS_2(n-1,m)$
- $S_2(n,m)=\sum_{i=0}^{m}\frac{(-1)^i\,(m-i)^n}{i!\,(m-i)!}$ 可以卷积算一行
- 考虑 n 个带标号球全部放入 m 个带标号盒子共有 m^n 种方案，有和式 $m^n=\sum_{k=0}^{m}S_2(n,k)m^{\underline{k}}$
- 列 EGF $\sum_{n\ge0}S_2(n,m)\frac{x^n}{n!}=\frac{1}{m!}(e^x-1)^m$

整数分拆数

n 个无标号球全部放入一些 (个数无限制) 无标号盒子，要求每个盒子非空

记方案数为 $p(n)$

- $p(n)=\sum_{m=1}^{n}p(n,m)$
- OGF：$\sum_{n\ge0}p(n)x^n=\prod_{n\ge1}\frac{1}{1-x^n}$

第一类 Stirling 数

求有 k 个轮换的 n 元置换的方案数，记为 $S_1(n,k)$

- $S_1(n,0)=[n=0]$
- $S_1(n,k)=S_1(n-1,k-1)+(n-1)S_1(n-1,k)$
- 行 OGF $x^{\overline{n}}=\sum_{k=0}^{n}S_1(n,k)x^k$
- 列 EGF $\sum_{n\ge0}S_1(n,k)\frac{x^n}{n!}=\frac{1}{k!}ln^k(1+x)$

Stirling 反演 (未验证)

$\sum_{k\ge0}S_2(n,k)S_1(k,m)=[n=m]$

$f_n=\sum_{i=0}^{n}S_2(n,i)g_i\iff g_n=\sum_{i=0}^{n}S_1(n,i)f^i$

二维计算几何

- `Point` 直接支持整型和浮点型
- 部分函数可以对整型改写
- 多边形 (凸包) 按逆时针存在下标 `1..n`

点向量基本运算

```cpp
template <typename T>
struct Point {
    T x, y;
    Point() {}
    Point(T u, T v) : x(u), y(v) {}
    Point operator+(const Point &a) const { return Point(x + a.x, y + a.y); }
    Point operator-(const Point &a) const { return Point(x - a.x, y - a.y); }
    Point operator*(const T &a) const { return Point(x * a, y * a); }
    T operator*(const Point &a) const { return x * a.x + y * a.y; }
    T operator%(const Point &a) const { return x * a.y - y * a.x; }
    double len() const { return hypot(x, y); }
    double operator^(const Point &a) const { return (a - (*this)).len(); }
    double angle() const { return atan2(y, x); }
    bool id() const { return y < 0 || (y == 0 && x < 0); }
    bool operator<(const Point &a) const { return id() == a.id() ? (*this) % a > 0 : id() < a.id(); }
};
typedef Point<double> point;

#define sqr(x) ((x) * (x))
const point O(0, 0);
const double PI(acos(-1.0)), EPS(1e-8);
inline bool dcmp(const double &x, const double &y) { return fabs(x - y) < EPS; }
inline int sgn(const double &x) { return fabs(x) < EPS ? 0 : ((x < 0) ? -1 : 1); }
inline double mul(point p1, point p2, point p0) { return (p1 - p0) % (p2 - p0); }
```

位置关系

```cpp
inline bool in_same_seg(point p, point a, point b) {
    if (fabs(mul(p, a, b)) < EPS) {
        if (a.x > b.x) swap(a, b);
        return (a.x <= p.x && p.x <= b.x && ((a.y <= p.y && p.y <= b.y) || (a.y >= p.y && p.y >= b.y)));
    } else return 0;
}

inline bool is_right(point st, point ed, point a) {
    return ((ed - st) % (a - st)) < 0;
}

inline point intersection(point s1, point t1, point s2, point t2) {
    return s1 + (t1 - s1) * (((s1 - s2) % (t2 - s2)) / ((t2 - s2) % (t1 - s1)));
}

inline bool parallel(point a, point b, point c, point d) {
    return dcmp((b - a) % (d - c), 0);
}

inline double point2line(point p, point s, point t) {
    return fabs(mul(p, s, t) / (t - s).len());
}

inline double point2seg(point p, point s, point t) {
    return sgn((t - s) * (p - s)) * sgn((s - t) * (p - t)) > 0 ? point2line(p, s, t) : min((p ^ s), (p ^ t));
}
```

多边形

求多边形面积

```cpp
inline double area(int n, point s[]) {
    double res = 0;
    s[n + 1] = s[1];
    for (int i = 1; i <= n; ++i)
        res += s[i] % s[i + 1];
    return fabs(res / 2);
}
```

判断点是否在多边形内

- 特判边上的点
- 使用了 a[1]...a[n+1] 的数组

```
inline bool in_the_area(point p, int n, point area[]) {
    bool ans = 0; double x;
    area[n + 1] = area[1];
    for (int i = 1; i <= n; ++i) {
        point p1 = area[i], p2 = area[i + 1];
        if (in_same_seg(p, p1, p2)) return 1; //特判边上的点
        if (p1.y == p2.y) continue;
        if (p.y < min(p1.y, p2.y)) continue;
        if (p.y >= max(p1.y, p2.y)) continue;
        ans ^= (((p.y - p1.y) * (p2.x - p1.x) / (p2.y - p1.y) + p1.x) > p.x);
    }
    return ans;
}
```

凸包

- *Andrew* 算法
- O(n log n)
- 可以应对凸包退化成直线/单点的情况但后续旋转卡壳时应注意特判
- 注意是否应该统计凸包边上的点

```
inline bool pcmp1(const point &a, const point &b) { return a.x == b.x ? a.y < b.y : a.x < b.x; }

inline int Andrew(int n, point p[], point ans[]) { //ans[] 逆时针存凸包
    sort(p + 1, p + 1 + n, pcmp1);
    int m = 0;
    for (int i = 1; i <= n; ++i) {
        while (m > 1 && mul(ans[m - 1], ans[m], p[i]) < 0) --m; //特判凸包边上的点
        ans[++m] = p[i];
    }
    int k = m;
    for (int i = n - 1; i >= 1; --i) {
        while (m > k && mul(ans[m - 1], ans[m], p[i]) < 0) --m; //特判凸包边上的点
        ans[++m] = p[i];
    }
    return m - (n > 1); //返回凸包有多少个点
}
```

凸包直径·平面最远点对

- 旋转卡壳算法
- O(n)
- 凸包的边上只能有端点，否则不满足严格单峰
- 凸包不能退化成直线，调用前务必检查 n>=3
- 使用了 a[1]...a[n+1] 的数组

```
inline double Rotating_Caliper(int n, point a[]) {
    a[n + 1] = a[1];
    double ans = 0;
    int j = 2;
    for (int i = 1; i <= n; ++i) {
        while (fabs(mul(a[i], a[i + 1], a[j])) < fabs(mul(a[i], a[i + 1], a[j + 1]))) j = (j % n + 1);
        ans = max(ans, max((a[j] ^ a[i]), (a[j] ^ a[i + 1])));
    }
    return ans;
}
```

平面最近点对

- 分治 + 归并

- O(n log n)

```cpp
namespace find_the_closest_pair_of_points {
    const int N = 200010; //maxn
    inline bool cmp1(const point &a, const point &b) { return a.x < b.x || (a.x == b.x && a.y < b.y); }
    inline bool operator>(const point &a, const point &b) { return a.y > b.y || (a.y == b.y && a.x > b.x); }

    point a[N], b[N];
    double ans;
    inline void upd(const point &i, const point &j) { ans = min(ans, i ^ j); }

    void find(int l, int r) {
        if (l == r) return;
        if (l + 1 == r) {
            if (a[l] > a[r]) swap(a[l], a[r]);
            upd(a[l], a[r]); return;
        }
        int mid = (l + r) >> 1;
        double mx = (a[mid + 1].x + a[mid].x) / 2;
        find(l, mid); find(mid + 1, r);
        int i = l, j = mid + 1;
        for (int k = l; k <= r; ++k) b[k] = a[((j > r) || (i <= mid && a[j] > a[i])) ? (i++) : (j++)];
        for (int k = l; k <= r; ++k) a[k] = b[k];
        int tot = 0;
        for (int k = l; k <= r; ++k) if (fabs(a[k].x - mx) <= ans) {
            for (int j = tot; j >= 1 && (a[k].y - b[j].y <= ans); --j) upd(a[k], b[j]);
            b[++tot] = a[k];
        }
    }

    //接口
    inline double solve(int n, point ipt[]){
        ans = 0x3f3f3f3f3f3f3f3fll; //max distance
        for (int i = 1; i <= n; ++i) a[i] = ipt[i];
        sort(a + 1, a + 1 + n, cmp1);
        find(1, n);
        return ans;
    }
}
```

圆

三点垂心

```cpp
inline point geto(point p1, point p2, point p3) {
    double a = p2.x - p1.x;
    double b = p2.y - p1.y;
    double c = p3.x - p2.x;
    double d = p3.y - p2.y;
    double e = sqr(p2.x) + sqr(p2.y) - sqr(p1.x) - sqr(p1.y);
    double f = sqr(p3.x) + sqr(p3.y) - sqr(p2.x) - sqr(p2.y);
    return {(f * b - e * d) / (c * b - a * d) / 2, (a * f - e * c) / (a * d - b * c) / 2};
}
```

最小覆盖圆

- 随机增量 O(n)

```cpp
inline void min_circlefill(point &o, double &r, int n, point a[]) {
    mt19937 myrand(20011224); shuffle(a + 1, a + 1 + n, myrand); //越随机越难 hack
    o = a[1];
    r = 0;
    for (int i = 1; i <= n; ++i) if ((a[i] ^ o) > r + EPS) {
        o = a[i];
        r = 0;
        for (int j = 1; j < i; ++j) if ((o ^ a[j]) > r + EPS) {
            o = (a[i] + a[j]) * 0.5;
            r = (a[i] ^ a[j]) * 0.5;
            for (int k = 1; k < j; ++k) if ((o ^ a[k]) > r + EPS) {
                o = geto(a[i], a[j], a[k]);
                r = (o ^ a[i]);
```

```
            }
        }
    }
}
```

图论

存图

- 前向星
- 注意边数开够

```
int Head[N], Ver[N*2], Next[N*2], Ew[N*2], Gtot=1;
inline void graphinit(int n) {Gtot=1; for(int i=1; i<=n; ++i) Head[i]=0;}
inline void edge(int u, int v, int w=1) {Ver[++Gtot]=v; Next[Gtot]=Head[u]; Ew[Gtot]=w; Head[u]=Gtot;}
#define go(i,st,to) for (int i=Head[st], to=Ver[i]; i; i=Next[i], to=Ver[i])
```

最短路

Dijkstra

- 非负权图

```
namespace DIJK{//适用非负权图 满足当前 dist 最小的点一定不会再被松弛
    typedef pair<long long,int> pii;
    long long dist[N];//存最短路长度
    bool vis[N];//记录每个点是否被从队列中取出  每个点只需第一次取出时扩展
    priority_queue<pii,vector<pii>,greater<pii> >pq;//维护当前 dist[] 最小值及对应下标 小根堆

    inline void dijk(int s,int n){//s 是源点 n 是点数
        while(pq.size())pq.pop();for(int i=1;i<=n;++i)dist[i]=INFLL,vis[i]=0;//所有变量初始化
        dist[s]=0;pq.push(make_pair(0,s));
        while(pq.size()){
            int now=pq.top().second;pq.pop();
            if(vis[now])continue;vis[now]=1;
            go(i,now,to){
                const long long relx(dist[now]+Ew[i]);
                if(dist[to]>relx){dist[to]=relx;pq.push(make_pair(dist[to],to));}//松弛
            }
        }
    }
}
```

LCA

- 倍增求 lca
- 数组开够

```
namespace LCA_Log{
    int fa[N][22],dep[N];
    int t,now;
    void dfs(int x){
        dep[x]=dep[fa[x][0]]+1;
        go(i,x,to){
            if(dep[to])continue;
            fa[to][0]=x;for(int j=1;j<=t;++j)fa[to][j]=fa[fa[to][j-1]][j-1];
            dfs(to);
        }
    }

    //初始化接口
    inline void lcainit(int n,int rt){//记得初始化全部变量
        now=1;t=0;while(now<n)++t,now<<=1;
        for(int i=1;i<=n;++i)dep[i]=0,fa[i][0]=0;
        for(int i=1;i<=t;++i)fa[rt][i]=0;
        dfs(rt);
    }

```

```
    //求 lca 接口
    inline int lca(int u,int v){
        if(dep[u]>dep[v])swap(u,v);
        for(int i=t;~i;--i)if(dep[fa[v][i]]>=dep[u])v=fa[v][i];
        if(u==v)return u;
        for(int i=t;~i;--i)if(fa[u][i]!=fa[v][i])u=fa[u][i],v=fa[v][i];
        return fa[u][0];
    }
}
```

连通性

有向图强联通分量

- tarjan $O(n)$

```
namespace SCC{
    int dfn[N],clk,low[N];
    bool ins[N];int sta[N],tot; //栈  存正在构建的强连通块
    vector<int>scc[N];int c[N],cnt;//cnt 为强联通块数 scc[i] 存放每个块内点 c[i] 为原图每个结点属于的块
    void dfs(int x){
        dfn[x]=low[x]=(++clk);//low[] 在这里初始化
        ins[x]=1;sta[++tot]=x;
        go(i,x,to){
            if(!dfn[to]){dfs(to);low[x]=min(low[x],low[to]);}//走树边
            else if(ins[to])low[x]=min(low[x],dfn[to]);//走返祖边
        }
        if(dfn[x]==low[x]){//该结点为块的代表元
            ++cnt;int u;
            do{u=sta[tot--];ins[u]=0;c[u]=cnt;scc[cnt].push_back(u);}while(x!=u);
        }
    }
    inline void tarjan(int n){//n 是点数
        for(int i=1;i<=cnt;++i)scc[i].clear();//清除上次的 scc 防止被卡 MLE
        for(int i=1;i<=n;++i)dfn[i]=ins[i]=0;tot=clk=cnt=0;//全部变量初始化
        for(int i=1;i<=n;++i)if(!dfn[i])dfs(i);
        for(int i=1;i<=n;++i)c[i]+=n;//此行 (可以省略) 便于原图上加点建新图 加新点前要初始化 Head[]=0
    }
}
```

二分图匹配

匈牙利算法求二分图无权最大匹配

- 复杂度 O(nm)

```
#include<bits/stdc++.h>
using namespace std;
namespace Hungary{
    const int N=1000009;
    //图中应储存左部到右部的边  左点编号 1..n  右点编号 1..m
    int Head[N],Ver[N*2],Next[N*2],Gtot=1;  //注意边数开够
    inline void graphinit(int n){Gtot=1;for(int i=1;i<=n;++i)Head[i]=0;}
    inline void edge(int u,int v){Ver[++Gtot]=v,Next[Gtot]=Head[u],Head[u]=Gtot;}
    #define go(i,st,to) for(int i=Head[st],to=Ver[i];i;i=Next[i],to=Ver[i]) //st 是左点, to 是 st 能到达的右点

    int match[N],vis[N];//右点的的匹配点和访问标记

    bool dfs(int x){//x 是左点
        go(i,x,to)if(!vis[to]){
            vis[to]=1;
            if(!match[to]||dfs(match[to])){
                match[to]=x;return 1;
            }
        }
        return 0;
    }

    inline int ask(int n,int m){//左点数和右点数 返回最大匹配数
        for(int i=1;i<=m;++i)match[i]=0;int res=0;
```

```
        for(int i=1;i<=n;++i){
            for(int j=1;j<=m;++j)vis[j]=0;
            res+=dfs(i);
        }
        return res;
    }
}
```

KM 算法求二分图带权最大匹配

- 要求该图存在完美匹配该算法将最大化完美匹配的权值和
- 复杂度 O(n^3)
- naive 的写法复杂度为 O((n^2)m) 在完全图上会退化至 O(n^4) luogu P6577

```
#include<bits/stdc++.h>
using namespace std;
namespace KM{
    const int N=509;
    const long long INFLL=0x3f3f3f3f3f3f3f3fll;//注意 INF 应足够大 至少远大于 n*n*Max{W}

    int n;//左右点数均为 n 标号均为 1..n
    int w[N][N];//邻接矩阵存边权
    bool r[N][N];//记录 i j 之间是否有边连接 完全图/非 0 权图则不必要

    inline void init(){//记得重写初始化
        int m;scanf("%d%d",&n,&m);
        for(int i=1;i<=m;++i){
            int u,v,wt;scanf("%d%d%d",&u,&v,&wt);
            w[u][v]=wt;r[u][v]=1;
        }
    }

    long long la[N],lb[N],upd[N];//左右顶标 每个右点对应的最小 delta 要开 longlong
    int last[N];//每个右点对应的回溯右点
    bool va[N],vb[N];
    int match[N];//每个右点对应的左匹配点
    bool dfs(int x,int fa){
        va[x]=1;
        for(int y=1;y<=n;++y)if(r[x][y]&&!vb[y]){
            const long long dt=la[x]+lb[y]-w[x][y];
            if(dt==0){//相等子图
                vb[y]=1;last[y]=fa;
                if(!match[y]||dfs(match[y],y)){
                    match[y]=x;
                    return 1;
                }
            }else if(upd[y]>dt){//下次 dfs 直接从最小 delta 处开始
                upd[y]=dt;
                last[y]=fa;//用 last 回溯该右点的上一个右点以更新增广路
            }
        }
        return 0;
    }

    inline void KM(){
        for(int i=1;i<=n;++i){//初始化顶标
            la[i]=-INFLL;lb[i]=0;
            for(int j=1;j<=n;++j)if(r[i][j])la[i]=max(la[i],(long long)w[i][j]);
        }
        for(int i=1;i<=n;++i){//尝试给每一个左点匹配上右点 匹配失败则扩展相等子图重试至成功
            memset(va,0,sizeof va);//注意复杂度
            memset(vb,0,sizeof vb);
            memset(last,0,sizeof last);
            memset(upd,0x3f,sizeof upd);
            int st=0;match[0]=i;//给起点 i 连一个虚右点 标号为 0
            //不断尝试将右点 st 从已有的匹配中解放 以获得增广路
            while(match[st]){//当 st 到达非匹配右点直接退出
                long long delta=INFLL;
                if(dfs(match[st],st))break;//st 的左点匹配到了新的右点则退出
```

```
                for(int j=1;j<=n;++j)
                    if(!vb[j]&&upd[j]<delta){//下次从最小的 delta 处开始 DFS
                        delta=upd[j];
                        st=j;
                    }
                for(int j=1;j<=n;++j){//将交错树上的左顶标加 delta 右顶标减 delta 使更多的边转为相等边
                    if(va[j])la[j]-=delta;
                    if(vb[j])lb[j]+=delta;
                        else upd[j]-=delta;//小问题: 这里在干啥 每次修改顶标后重新计算 upd 是否可行
                }
                vb[st]=1;
            }
            while(st){//更新增广路
                match[st]=match[last[st]];
                st=last[st];
            }
        }

        long long ans=0;
        for(int i=1;i<=n;++i)ans+=w[match[i]][i];
        printf("%lld\n",ans);
        for(int i=1;i<=n;++i)printf("%d%c",match[i]," \n"[i==n]);
    }
    //signed main(){init();KM();return 0;}
}
```

数据结构

STL

小跟堆

```
priority_queue<vector<int>, int, greater<int> > q;
```

整数哈希

```
#include<bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/hash_policy.hpp>
using namespace std;
typedef unsigned long long u64;

struct neal {
    static u64 A(u64 x) {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }
    u64 operator()(u64 x) const{
        static const u64 C = chrono::steady_clock::now().time_since_epoch().count();
        return A(x + C);
    }
};

unordered_map<int, int, neal> HASH1;

__gnu_pbds::gp_hash_table<int, int, neal> HASH2;
```

二维树状数组

- LOJ135 区间修改 + 区间查询单次操作复杂度 log^2 空间复杂度 N^2
- 对原数组差分 S(n,m) = ∑∑A_ij 用树状数组维护差分数组 A_ij 转化为单点修改
- 区间查询 SS(n,m) = ∑∑S(i,j) 推式子转化为单点查询问题
- SS(n,m) = ∑∑∑∑A_ij = (n+1)*(m+1)*∑∑A_ij - (n+1)*∑∑jA_ij - (m+1)*∑∑iA_ij + ∑∑ijA_ij

```cpp
//省略了文件头
const int N=2051;
int n,m;
struct dat{
    LL c,cx,cy,cxy;
    void operator+=(const dat&a){c+=a.c;cx+=a.cx;cy+=a.cy;cxy+=a.cxy;}
}c[N][N];

inline void add(int a,int b,int k){
    const dat d={k,k*a,k*b,k*a*b};
    for(int i=a;i<=n;i+=(i&(-i)))
    for(int j=b;j<=m;j+=(j&(-j)))c[i][j]+=d;
}

inline LL f(int a,int b){
    dat r={0,0,0,0};
    for(int i=a;i>0;i-=(i&(-i)))
    for(int j=b;j>0;j-=(j&(-j)))r+=c[i][j];
    return (a+1)*(b+1)*r.c-(b+1)*r.cx-(a+1)*r.cy+r.cxy;
}

signed main(){
    read(n,m);
    int op;while(read(op))if(op==1){
        int a,b,c,d,k;read(a,b,c,d,k);
        add(a,b,k);add(c+1,b,-k);add(a,d+1,-k);add(c+1,d+1,k);
    }else{
        int a,b,c,d;read(a,b,c,d);
        printf("%lld\n",f(c,d)-f(a-1,d)-f(c,b-1)+f(a-1,b-1));
    }
    return 0;
}
```

堆式线段树

- 区间求和区间修改
- 空间估算
- 所有数组务必初始化

```cpp
struct SegmentTree_Heap{
    #define TreeLen (N<<2)          //N
    #define lc(x)   ((x)<<1)
    #define rc(x)   ((x)<<1|1)
    #define sum(x)  (tr[x].sum)     //
    #define t(x)    (t[x])          //

    struct dat{
        LL sum;
        /* 满足结合律的基本运算 用于合并区间信息 */
        dat operator+(const dat&brother){
            dat result;
            result.sum=sum+brother.sum;
            return result;
        }
    }tr[TreeLen];
    LL t[TreeLen];  //lazy tag

    /* 单区间修改 */
    inline void change(const int&x,const int&l,const int&r,const LL&d){
        tr[x].sum=tr[x].sum+d*(r-l+1);
        t[x]=t[x]+d;
    }

    inline void pushup(int x){tr[x]=tr[lc(x)]+tr[rc(x)];}

    inline void pushdown(int x,const int&l,const int&r,const int&mid){
        if(t(x)){ // 区间修改注意细节
            change(lc(x),l,mid,t(x));
            change(rc(x),mid+1,r,t(x));
            t(x)=0;
```

```
        }
    }

    void build(int x,int l,int r){
        t(x)=0;   // 记得初始化
        if(l==r){
            sum(x)=0;
            return;
        }
        int mid=(l+r)>>1;
        build(lc(x),l,mid);
        build(rc(x),mid+1,r);
        pushup(x);
    }

    void add(int x,int l,int r,const int&L,const int&R,const LL&d){
        if(L<=l&&r<=R){
            change(x,l,r,d);
            return;
        }
        int mid=(l+r)>>1;pushdown(x,l,r,mid);
        if(L<=mid)add(lc(x),l,mid,L,R,d);
        if(R>mid)add(rc(x),mid+1,r,L,R,d);
        pushup(x);
    }

    LL ask(int x,int l,int r,const int&L,const int&R){
        if(L<=l&&r<=R)return sum(x);
        int mid=(l+r)>>1;pushdown(x,l,r,mid);
        LL res=0;
        if(L<=mid)res=(res+ask(lc(x),l,mid,L,R));
        if(mid<R)res=(res+ask(rc(x),mid+1,r,L,R));
        return res;
    }
};
```

小根堆

```
namespace MyPQ{
    typedef int pqdat;    ////
    pqdat q[N];
    int tot;
    void up(int x){
        while(x>1)
        if(q[x]<q[x/2]){
            swap(q[x],q[x/2]);
            x/=2;
        }else return;
    }
    void down(int x){
        int ls=x*2;
        while(ls<=tot){
            if(ls<tot&&q[ls+1]<q[ls])++ls;
            if(q[ls]<q[x]){
                swap(q[x],q[ls]);x=ls;ls=x*2;
            }else return;
        }
    }
    void push(pqdat x){q[++tot]=x;up(tot);}
    pqdat top(){return q[1];}
    void pop(){if(!tot)return;q[1]=q[tot--];down(1);}
    void pop(int k){if(!tot)return;q[k]=q[tot--];up(k);down(k);}
}
```

Treap

```
#include<bits/stdc++.h>
using namespace std;
const int N=2000007,INF=(1ll<<30)+7;

```

```cpp
//Treap 维护升序多重集
//支持操作: 数 <-> 排名 查询某数前驱后继
//操作数 x 可以不在集合中
//x 的排名: 集合中 <x 的数的个数 +1
//排名 x 的数: 集合中排名 <=x 的数中的最大数
//x 的前驱 比 x 小的最大数

struct treap{//所有点值不同 用副本数实现多重集
    int l,r;
    int v,w;//v 是数据 w 是维护堆的随机值
    int num,sz;//num 是该点副本数 sz 是该子树副本总数
}tr[N];int tot,rt;//tr[0] 始终全 0 使用范围 tr[1..n]
#define lc(x) tr[x].l
#define rc(x) tr[x].r
#define sz(x) tr[x].sz
#define num(x) tr[x].num
#define val(x) tr[x].v
#define wt(x) tr[x].w

inline int New(int x){
    val(++tot)=x; wt(tot)=rand();
    num(tot)=sz(tot)=1; return tot;
}

inline void upd(int p){sz(p)=sz(lc(p))+sz(rc(p))+num(p);}

inline void build(){//初始化 INF 和-INF 两个点
    srand(time(0));
    rt=1;tot=2;
    rc(1)=2;val(1)=-INF;wt(1)=rand();num(1)=1;sz(1)=2;
    val(2)=INF;wt(2)=rand();num(2)=1;sz(2)=1;
}

//调用时记得减一 askrk(rt,x)-1
int askrk(int p,int x){//当前子树中查询 x 的排名
    if(p==0)return 1;//说明某子树所有数均比 x 大
    if(x==val(p))return sz(lc(p))+1;
    return x<val(p)?askrk(lc(p),x):askrk(rc(p),x)+sz(lc(p))+num(p);
}

//调用时记得加一 kth(rt,++rank)
int kth(int p,int rk){//当前子树中查询排名 rk 的数
    if(p==0)return INF;//说明集合大小 <rk
    if(sz(lc(p))>=rk)return kth(lc(p),rk);
    rk-=sz(lc(p))+num(p);
    return (rk>0)?kth(rc(p),rk):val(p);
}

inline void zig(int &p){//与左子节点交换位置
    int q=lc(p);lc(p)=rc(q);rc(q)=p;
    upd(p);p=q;upd(p);
}

inline void zag(int &p){//与右子节点交换位置
    int q=rc(p);rc(p)=lc(q);lc(q)=p;
    upd(p);p=q;upd(p);
}

//insert(rt,x)
void insert(int &p,int x){//当前子树中插入 x
    if(p==0){p=New(x);return;}//x 首次插入
    if(x==val(p)){++num(p);++sz(p);return;}
    if(x<val(p)){
        insert(lc(p),x);
        if(wt(p)<wt(lc(p)))zig(p);//维护大根堆
    }else{
        insert(rc(p),x);
        if(wt(p)<wt(rc(p)))zag(p);//维护大根堆
    }
    upd(p);
}
```

```cpp

//erase(rt,x)
void erase(int &p,int x){//当前子树中删除一个 x
    if(p==0)return;//已经无需删除
    if(val(p)==x){//如果找到了 x 的位置
        if(num(p)>1){//无需删点
            --num(p);--sz(p);return;//如果有多个 x 维护副本数即可
        }
        if(lc(p)||rc(p)){//该点不是叶子节点 则不断向下调整至叶子节点
            if(rc(p)==0||wt(lc(p))>wt(rc(p)))zig(p),erase(rc(p),x);//由于 rand() 的值域 & 大根堆的实现 故省略左子树为空的判断
            else zag(p),erase(lc(p),x);
            upd(p);
        }else p=0;//是叶子节点则直接删除
        return;
    }
    x<val(p)?erase(lc(p),x):erase(rc(p),x);upd(p);
}

int askpre(int x){
    int id=1;//-INF 若没有前驱则返回-INF
    //尝试自顶向下寻找 x 则 x 的前驱有两种情况
    //1) 未找到 x 或 x 没有左子树 则前驱在搜索路径上
    //2) 前驱是 x 的左子树中最大值 即 x 的左子树一直向右走
    int p=rt;
    while(p){
        if(x==val(p)){//找到 x
            if(lc(p)){p=lc(p);while(rc(p))p=rc(p);id=p;}
            break;
        }
        if(val(p)<x&&val(p)>val(id))id=p;//每经过一个点尝试更新前驱
        p=(val(p)>x?lc(p):rc(p));//找 x
    }
    return val(id);
}

int asknxt(int x){
    int id=2;//INF
    int p=rt;
    while(p){
        if(x==val(p)){
            if(rc(p)){p=rc(p);while(lc(p))p=lc(p);id=p;}
            break;
        }
        if(val(p)>x&&val(p)<val(id))id=p;
        p=(val(p)>x?lc(p):rc(p));
    }
    return val(id);
}
```

字符串

KMP

```cpp
//luogu P3375
#include <bits/stdc++.h>
using namespace std;
const int N = 1000009;

char s1[N], s2[N];
int fail[N], n, m;

signed main() {
    scanf("%s%s", s1 + 1, s2 + 1);
    n = strlen(s1 + 1);
    m = strlen(s2 + 1);

    //fail[1] = 0;
    for (int i = 2, j = 0; i <= m; ++i) {
        while (j != 0 && s2[j + 1] != s2[i]) j = fail[j];
        if (s2[j + 1] == s2[i]) ++j;
```

```
        fail[i] = j;
    }

    int p = 0;
    for (int i = 1; i <= n; ++i) {
        while (p != 0 && s2[p + 1] != s1[i]) p = fail[p];
        if (s2[p + 1] == s1[i]) ++p;

        if (p == m) {
            printf("%d\n", i - p + 1);
            p = fail[p];
        }
    }

    for (int i = 1; i <= m; ++i) printf("%d ", fail[i]);
    return 0;
}
```

扩展 KMP(Z 函数)

```
//luogu 5410
#include <bits/stdc++.h>
using namespace std;
const int N = 20000009;

char t[N], p[N];
int n, m, z[N], mc[N];

signed main() {
    scanf("%s%s", t + 1, p + 1);
    n = strlen(t + 1);
    m = strlen(p + 1);

    z[1] = m;
    for (int i = 2, l = 0, r = 0; i <= m; ++i) {
        z[i] = ((r < i) ? 0 : min(r - i + 1, z[i - l + 1]));
        while (z[i] <= m - i && p[i + z[i]] == p[z[i] + 1]) ++z[i];
        if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
    }

    for (int i = 1, l = 0, r = 0; i <= n; ++i) {
        mc[i] = ((r < i) ? 0 : min(r - i + 1, z[i - l + 1]));
        while (mc[i] <= n - i && mc[i] < m && t[i + mc[i]] == p[mc[i] + 1])
            ++mc[i];
        if (i + mc[i] - 1 > r) l = i, r = i + mc[i] - 1;
    }

    unsigned long long u = 0, v = 0;
    for (int i = 1; i <= m; ++i) u ^= (i * (z[i] + 1ull));
    for (int i = 1; i <= n; ++i) v ^= (i * (mc[i] + 1ull));
    printf("%llu\n%llu\n", u, v);
    return 0;
}
```

AC 自动机

```
#include<bits/stdc++.h>
using namespace std;
const int N=1000009;

//洛谷 P3808 AC 自动机 (简单版)
int n;
char s[N];
struct AC{
    struct dat{
        int tp[26],fail,ed;
        void init(){memset(tp,0,sizeof tp);fail=ed=0;}
    }tr[N];int tot;
    void init(){for(int i=0;i<=tot;++i)tr[i].init();tot=0;}

```

```
    void insert(char*s){//C 风格字符串 下标从 1 开始
        int p=0;
        for(int i=1;s[i]!=0;++i){//C 风格字符串 下标从 1 开始
            int&tp=tr[p].tp[s[i]-'a'];
            if(tp==0)tp=(++tot);
            p=tp;
        }
        ++tr[p].ed;
    }

    void build(){//先 insert 所有模式串 然后 build
        queue<int>q;
        for(int i=0;i<26;++i)if(tr[0].tp[i])q.push(tr[0].tp[i]);
        while(q.size()){
            dat&now=tr[q.front()];q.pop();
            for(int i=0;i<26;++i){
                int&tp=now.tp[i];
                if(tp)tr[tp].fail=tr[now.fail].tp[i],q.push(tp);
                else tp=tr[now.fail].tp[i];//路径压缩
            }
        }
    }

    int ask(char*s){//C 风格字符串 下标从 1 开始
        int p=0,res=0;
        for(int i=1;s[i]!=0;++i){//C 风格字符串 下标从 1 开始
            p=tr[p].tp[s[i]-'a'];
            for(int j=p;tr[j].ed!=-1;j=tr[j].fail)res+=tr[j].ed,tr[j].ed=-1;
        }
        return res;
    }
}ac;

signed main(){
    scanf("%d",&n);
    for(int i=1;i<=n;++i){
        scanf("%s",s+1);
        ac.insert(s);
    }
    ac.build();
    scanf("%s",s+1);
    printf("%d\n",ac.ask(s));
    return 0;
}
```

杂项

整数哈希

```
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/hash_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

//https://codeforces.com/blog/entry/62393

struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM = chrono::steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
```

```
};

unordered_map<long long, int, custom_hash> safe_map;
gp_hash_table<long long, int, custom_hash> safe_hash_table;
```

GCC 位运算

需要 gcc 编译器

- `int __builtin_ffs (unsigned int x)` 返回 x 的最后一位 1 的是从后向前第几位，比如 7368（1110011001000）返回 4。
- `int __builtin_clz (unsigned int x)` 返回前导的 0 的个数。
- `int __builtin_ctz (unsigned int x)` 返回后面的 0 个个数，和 `__builtin_clz` 相对。
- `int __builtin_popcount (unsigned int x)` 返回二进制表示中 1 的个数。
- `int __builtin_parity (unsigned int x)` 返回 x 的奇偶校验位，也就是 x 的 1 的个数模 2 的结果。

此外，这些函数都有相应的 `usigned long` 和 `usigned long long` 版本，只需要在函数名后面加上 l 或 ll 就可以了，比如 `int __builtin_clzll`。

来源 https://blog.csdn.net/yuer158462008/article/details/46383635

对拍

for windows

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

//输入文件   input.in
//正确的程序名    ac.cpp   ac.exe   ac.out
//待验证的程序名   wa.cpp   wa.exe   wa.out

void gen(){//生成 INPUT 内的数据
    FILE *op=fopen("input.in","w");
    //mt19937 RAND(time(nullptr));
    //uniform_int_distribution<int>(1,100)(RAND);//范围内均匀随机整数
    //fprintf(op,"%d\n",233);
    fclose(op);
}

bool check(){//比较 ac.out 和 wa.out 的内容   一致则返回 0   否则返回非 0
    return system("fc wa.out ac.out");
}

int main(){
    system("g++ wa.cpp -o wa.exe");
    system("g++ ac.cpp -o ac.exe");
    while(1){
        gen(); int st,ed;

        st=clock();
        system("ac.exe < input.in > ac.out");
        ed=clock();
        printf("ac cost %lld ms\n",(ed-st)*1000ll/CLOCKS_PER_SEC);

        st=clock();
        system("wa.exe < input.in > wa.out");
        ed=clock();
        printf("wa cost %lld ms\n",(ed-st)*1000ll/CLOCKS_PER_SEC);

        if(check()){
            puts("Wrong Answer");
            system("pause");
            return 0;
        }
    }
```

```
    return 0;
}
```

fread/fwrite 实现的读写类

没用

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

struct MY_IO {
    #define MAXSIZE (1 << 20) //缓冲区大小 不小于 1 << 14
    inline bool isdigit(const char &x) {return x >= '0' && x <= '9';} //字符集 看情况改
    inline bool blank(const char &c) { return c == ' ' || c == '\n' || c == '\r' || c == '\t'; }

    char buf[MAXSIZE + 1], *p1, *p2, pbuf[MAXSIZE + 1], *pp;
    MY_IO() : p1(buf), p2(buf), pp(pbuf) {}
    ~MY_IO() { fwrite(pbuf, 1, pp - pbuf, stdout); }

    char gc() {
        if (p1 == p2) p2 = (p1 = buf) + fread(buf, 1, MAXSIZE, stdin);
        return p1 == p2 ? EOF : *p1++;
    }

    void pc(const char &c) {
        if (pp - pbuf == MAXSIZE) fwrite(pbuf, 1, MAXSIZE, stdout), pp = pbuf;
        *pp++ = c;
    }

    template <typename T>
    bool read(T &x) {
        x = 0; char c = gc(); int f = 1;
        while (!isdigit(c) && (c != '-') && (c != EOF)) c = gc();
        if (c == EOF) return 0;
        if (c == '-') f = -1, c = gc();
        while (isdigit(c)) { x = x * 10 + (c & 15); c = gc();}
        x *= f; return 1;
    }

    template <typename T, typename... Args>
    bool read(T &x, Args &...args) {
        bool res = 1;
        res &= read(x);
        res &= read(args...);
        return res;
    }

    int gets(char *s) {
        char c = gc();
        while (blank(c) && c != EOF) c = gc();
        if (c == EOF) return 0;
        int len = 0;
        while (!blank(c) && c != EOF) *s++ = c, c = gc(), ++len;
        *s = 0; return len;
    }

    void getc(char &c) { for (c = gc(); blank(c) && c != EOF; c = gc());}

    template <typename T>
    void write(T x) {
        if (x < 0) x = -x, putchar('-');
        static char sta[55];
        int top = 0;
        do sta[top++] = x % 10 + '0', x /= 10; while (x);
        while (top) putchar(sta[--top]);
    }

    template <typename T>
    void write(T x, const char &Lastchar) {write(x); pc(Lastchar);}

    void puts(char *s) {while ((*s) != 0)pc(*s++);}
```

```
65
66      int getline(char *s){
67          char c = gc(); int len = 0;
68          while (c != '\n' && c != EOF)*s++ = c, c = gc(), ++len;
69          *s = 0; return len;
70      }
71
72      void putline(char *s){ while ((*s) != 0) pc(*s++); pc('\n');}
73  } IO;
74
75  #define read  IO.read    //读一个/多个整数
76  #define write IO.write   //写一个整数
77  #define gc    IO.gc      //getchar()
78  #define pc    IO.pc      //putchar()
79  #define gets  IO.gets    //读一个指定字符集的 C 风格字符串 到 s[0..len-1] 返回 len
80  #define getc  IO.getc    //读一个指定字符集的字符
81  #define puts  IO.puts    //写一个 C 风格字符串
82  #define getl  IO.getline //读一行
83  #define putl  IO.putline //写一个 C 风格字符串并换行
```