ACM 模板

dnvtmf

2015

目录

1 数据结构

2 动态规划

3 图论

4 数学专题

4.1 逆元

```
///逆元inverse
//定义：如果a * b = 1 (% MOD), 则b 是a的逆元（模逆元， 乘法逆元）
//a的逆元存在条件：gcd(a, MOD) == 1
//性质：逆元是积性函数，如果c = a * b, 则 inv[c] = inv[a] * inv[b] % MOD
//方法一：  循环找解法（暴力）
//O(n)  预处理inv[1-n]：O（n^2)
LL getInv(LL x, LL MOD)
{
    for(LL i = 1; i < MOD; i++)
        if(x * i % MOD == 1)
            return i;
    return -1;
}

//方法二：  费马小定理和欧拉定理
//费马小定理:a^(p-1) ≡ 1(%p), 其中p是质数，所以a的逆元是a^(p-2)%p
//欧拉定理:x^φ(m) ≡ 1(%m) x与m互素，m是任意整数
//O(log n)(配合快速幂)，预处理inv[1-n]：O（nlog n)
LL qpow(LL x, LL k, LL MOD) {....}
LL getInv(LL x, LL MOD)
{
    //return qpow(x, euler_phi(MOD) - 1, MOD);
    return qpow(x, MOD - 2, MOD);//MOD是质数
}

//方法三：扩展欧几里得算法
//扩展欧几里得算法可解决  a*x + b*y = gcd(a,b)
//所以a * x % MOD = gcd(a, b) % MOD (b = MOD)
//O(log n), 预处理inv[1-n]：O(nlog n)
inline void exgcd(LL a, LL b, LL &g, LL &x, LL &y)
{
    if(!b) g = x, x = 1, y = 0;
    else exgcd(b, a % b, g, y, x), y -= (a / b) * x;
}

LL getInv(LL x, LL mod)
{
    LL g, inv, tmp;
    exgcd(x, mod, g, inv, tmp);
    return g != 1 ? -1 : (inv + mod) % mod;
}

//方法四：积性函数
//已处理inv[1] -- inv[n - 1], 求inv[n],  (MOD > n) (MOD为质数,不存在逆元的i干扰结果)
//MOD = x*n - y(0 <= y < n), ==> x * n = y (% MOD), ==> x * n * inv[y] = y * inv[y] = 1 (%MOD)
//所以inv[n] = x * inv[y] (x = MOD - MOD / n, y  MOD % n)
//O(log n) 预处理inv[1-n]：O(n)
LL inv[NUM];
void inv_pre(LL mod)
{
    inv[0] = inv[1] = 1LL;
    for(int i = 2; i < NUM; i++)
        inv[i] = (mod - mod / i) * inv[mod % i] % mod;
}
LL getInv(LL x, LL mod)
{
    LL res = 1LL;
    while(x > 1)
    {
```

```
        res = res * (mod - mod / x) % mod;
        x  = mod % x;
    }
    return res;
}
//方法五：积性函数+因式分解
//预处理出所有质数的逆元，采用exgcd来实现素数O(log n)求逆
//采用质因数分解， 可在O(log n)求出任意一个数的逆元
//预处理O(n log n)，单个O(log n)
```

4.2 模

```
/*
 模(Module)
 1. 基本运算
    Add: (a + b) % p = (a % p + b % p) % p
    Subtract:(a - b) % p = ((a%p - b%p)%p + p)%p
    Multiply:(a * b) % p = ((a % p) * (b % p)) % p
    Dvidive: (a / b) % p = (a * b^{-1}) % p b^{-1}是b关于p的逆元
    Power: (a^b)%p = ((a%p)^b)%p
 2. 推论
    若a ≡ b(%p), c ≡ d(%p),则(a + c) ≡ (b + d)(%p), (a - c) ≡ (b - d)(%p), (a * c) ≡ (b * d)(%p), (a/c) ≡ (b/d)(%p)

 3. 费马小定理
    若p是素数，对任意正整数x,有 x^p ≡ x(%p).
 4. 欧拉定理
    若p与x互素， 则有x^{φ(p)} ≡ 1(%p).
 5. n! = ap^e, gcd(a, p) == 1, p是素数
    e = (n/p + n/p^2 + n/p^3 + ···)(a不能被p整除)
    威尔逊定理：(p - 1)! ≡ -1(%p)(当且仅当p是素数)
    n!中不能被p整除的数的积:n! = (p - 1)!^{(n/p)} × (n mod p)!
    n!中能被p整除的项为:p, 2p, 3p, ... , (n/p)p, 除以p得到1,2,3,...,n/p(问题从缩减到n/p)
    在O(p)时间内预处理除0 <= n < p范围内中的n! mod p的表
    可在O(log_p n)时间内算出答案
    若不预处理,复杂度为O(p log_p n)
 */
int fact[MAX_P];//预处理n! mod p的表.O(p)
//分解n! = a p^e.返回a % p. O(log_p n)
int mod_fact(int n, int p, int &e)
{
    e = 0;
    if(n == 0) return 1;
    //计算p的倍数的部分
    int res = mod_fact(n / p, p, e);
    e += n / p;
    //由于(p - 1)! ≡ -1,因此只需知n/p的奇偶性
    if(n / p % 2) return res * (p - fact[n % p]) % p;
    return res * fact[n % p] % p;
}

/*
 6. n! = t(p^c)^u, gcd(t, p^c) == 1, p是素数
    1 ~ n中不能被p整除的项模p^c,以p^c为循环节， 预处理出n!%p^c的表
    1 ~ n中能被p整除的项， 提取 n/p 个p出来， 剩下阶乘(n/p)!， 递归处理
    最后， t还要乘上p^u
 */
LL fact[NUM];
LL qpow(LL x, LL k, LL mod);
inline void pre_fact(LL p, LL pc)//预处理n!%p^c O(p^c)
{
    fact[0] = fact[1] = 1;
    for(int i = 2; i < pc; i++)
```

```
    {
        if(i % p == 0)
            fact[i] = fact[i - 1];
        else
            fact[i] = fact[i - 1] * i % pc;
    }
}
//分解n! = t(p^c)^u, n!%pc = t * qpow(p, u, pc)
inline void mod_factorial(LL n, LL p, LL pc, LL &t, LL &u)
{
    t = 1, u = 0;
    while(n)
    {
        t = t * qpow(fact[pc - 1], n / pc, pc) % pc;
        t = t * fact[n % pc] % pc;
        u += n / p;
        n /= p;
    }
}
/*
 7. 大组合数求模, mod不是质数
    求C_n^m%mod
    1) 因式分解:mod = p_1^{c1} p_2^{c2} ··· p_k^{ck}
    2) 对每个因子p^c, 求C_n^m%p^c = n!%pc / (m!%pc(n-m)!%pc)
    3) 根据中国剩余定理求解答案(注: 逆元采用扩展欧几里得求法)
 */
LL fact[NUM];
LL prim[NUM], prim_num;
LL pre_prim();
LL pre_fact(LL p, LL pc);
LL mod_factorial(LL n, LL p, LL pc, LL &t, LL &u);
LL qpow(LL x, LL k, LL mod);
LL getInv(LL x, LL mod);

LL C(LL n, LL m, LL mod)
{
    LL p, pc, tmpmod = mod;
    LL Mi, tmpans, t, u, tot;
    LL ans = 0;
    int i, j;
    //将mod因式分解,mod = p_1^{c1} p_2^{c2} ··· p_k^{ck}
    for(i = 0; prim[i] <= tmpmod; i++)
        if(tmpmod % prim[i] == 0)
        {
            p = prim[i];
            pc = 1;
            while(tmpmod % p == 0)
            {
                pc *= p;
                tmpmod /= p;
            }
            //求C_n^k%pc
            pre_fact(p, pc);
            mod_factorial(n, p, pc, t, u);//n!
            tmpans = t;
            tot = u;
            mod_factorial(m, p, pc, t, u);//m!
            tmpans = tmpans * getInv(t, pc) % pc;//求逆元: 采用扩展欧几里得定律
            tot -= u;
            mod_factorial(n - m, p, pc, t, u); (n - m)!
            tmpans = tmpans * getInv(t, pc) % pc;
            tot -= u;
            tmpans = tmpans * qpow(p, tot, pc);
            //中国剩余定理
```

```
            Mi = mod / pc;
            ans = (ans + tmpans * Mi % mod * getInv(Mi, pc) % mod) % mod;
        }
    return ans;
}

/*
 8. 大组合数求模，mod是素数，Lucas定理
    Lucas定理： C_n^m%mod = C_{n/mod}^{m/mod} · C_{n%mod}^{m%mod}%mod
    采用O(n)方法预处理0~n-1的n!%mod和每个数的逆元，则可在O(log n)时间求出C_n^k%mod
*/
LL fact[NUM], inv[NUM];
void Lucas_init(LL mod);//预处理
LL Lucas(LL n, LL m, LL mod) //mod是质数
{
    LL a, b, res = 1LL;
    while(n && m)
    {
        a = n % mod, b = m % mod;
        if(a < b) return 0LL;
        res = res * fact[a] % mod * inv[fact[b] * fact[a - b] % mod, mod] % mod;
        n /= mod, m /= mod;
    }
    return res;
}
```

4.3 中国剩余定理和线性同余方程组

```
/*线性同余方程
 a_i × x ≡ b_i(mod m_i) (1 ≤ i ≤ n)
 如果方程组有解，那么一定有无穷有无穷多解，解的全集可写为x ≡ b(mod m)的形式.
 对方程逐一求解.令b = 0, m = 1;
 1.x ≡ b(mod m)可写为x = b + m * t;
 2. 带入第i个式子：a_i(b + m * t) ≡ b_i(mod m_i),即a_i × m × t ≡ b_i − a_i × b(mod m_i)
 3.当gcd(m_i, a_i × m)无法整除b_i − a_i × b时原方程组无解,否则用exgcd，求出满足条件的最小非负整数t,

中国剩余定理:
    对x ≡ a_i(mod m_i)(1 ≤ i ≤ n),其中m_1, m_2, ···, m_n两两互素，a_1, a_2, ···, a_n是任意整数，则有解:
    M = ∏ m_i, b = ∑_i^n a_i M_i^{-1} M_i(M_i = M/m_i)
*/
int gcd(int a, int b);
int getInv(int x, int mod);
pair<int, int> linear_congruence(const vector<int> &A, const vector<int> &B, const vector<int> &M)
{
    //初始解设为表示所有整数的x ≡ 0(mod 1)
    int x = 0, m = 1;
    for(int i = 0; i < A.size(); i++)
    {
        int a = A[i]*m, b = B[i] - A[i] * x, d = gcd(M[i], a);
        if(b % d == 0) return make_pair(0, -1);//无解
        int t = b/d * getInv(a / d, M[i] / d) % (M[i] / d);
        x = x + m * t;
        m *= M[i] / d;
    }
    return make_pair<x % m, m>;
}
```

4.4 组合与组合恒等式

/*1. 组合：从n个不同的元素中取r个的方案数C_n^r：

$$C_n^r = \begin{cases} \frac{n!}{r!(n-r)!}, & n \geq r \\ 1, & n \geq r = 0 \\ 0, & n < r \end{cases}$$

推论1：$C_n^r = C_n^{n-r}$

推论2(Pascal公式)：$C_n^r = C_{n-1}^r + C_{n-1}^{r-1}$

推论3：$\sum_{k=r-1}^{n-1} C_k^{r-1} = C_{n-1}^{r-1} + C_{n-2}^{r-2} + \ldots + C_{r-1}^{r-1} = C_n^r$

2. 从重集$B = \{\infty \cdot b_1, \infty \cdot b_2, \ldots, \infty \cdot b_n\}$的r—组合数F(n,r)为

$F(n,r) = C_{n+r-1}^r$

3. 二项式定义

当n是一个正整数时，对任何x和y有：

$$(x+y)^n = \sum_{k=0}^{n} C_n^k x^k y^{n-k}$$

令y=1，有：

$(1+x)^n = \sum_{k=0}^n C_n^k x^k = \sum_{k=0}^n C_n^{n-k} x^k$

广义二项式定理：

广义二项式系数：对于任何实数α和整数k,有

$$C_\alpha^K = \begin{cases} \frac{\alpha(\alpha-1)\ldots(\alpha-k+1)}{k!} & k > 0 \\ 1 & k = 0 \\ 0 & k < 0 \end{cases}$$

设α是一个任意实数，则对满足$|\frac{x}{y}| < 1$的所有x和y，有

$$(x+y)^\alpha = \sum_{k=0}^{\infty} C_\alpha^k x^k y^{\alpha-k}$$

推论：令$z = \frac{x}{y}$,则有

$$(1+z)^\alpha = \sum_{k=0}^{\infty} C_\alpha^k z^k, |z| < 1$$

令$\alpha = -n$(n是正整数),有

$$(1+z)^{-n} = \frac{1}{(1+z)^n} = \sum_{k=0}^{\infty} (-1)^k C_{n+k-1}^k z^k$$

又令z = -rz，(r为非零常数),有

又令n=1,有

$$\frac{1}{1+z} = \sum_{k=0}^{\infty} (-1)^k z^k$$

令z = -z，有

$$\frac{1}{1-z} = \sum_{k=0}^{\infty} z^k$$

令$\alpha = \frac{1}{2}$，有

$$\sqrt{1+z} = 1 + \sum_{k=1}^{\infty} \frac{(-1)^{k-1}}{k \cdot 2^{2k-1}} C_{2k-2}^{k-1} z^k$$

4. 组合恒等式

1. $\sum_{k=0}^n C_n^k = 2^n$
2. $\sum_{k=0}^n (-1)^k C_n^k = 0$
3. 对于正整数 n 和 k,

$$C_n^k = \frac{n}{k} C_{n-1}^{k-1}$$

4. 对于正整数 n,

$$\sum_{k=0}^{n} k C_n^k = \sum_{k=1}^{n} k C_n^k = n \cdot 2^{n-1}$$

5. 对于正整数 n,

$$\sum_{k=0}^{n}(-1)^k kC_n^k = 0$$

6. 对于正整数 n,

$$\sum_{k=0}^{n}k^2 C_n^k = n(n+1)2^{n-2}$$

7. 对于正整数 n,

$$\sum_{k=0}^{n}\frac{1}{k+1}C_n^k = \frac{2^{n+1}-1}{n+1}$$

8. (Vandermonde 恒等式) 对于正整数 n,m 和 p, 有 $p \leq minm, n$,

$$\sum_{k=0}^{p}C_n^k C_m^{p-k} = C_{m+n}^p$$

9. (令 p=m) 对于任何正整数 n,m,

$$\sum_{k=0}^{m}C_m^k C_n^k = C_{m+n}^m$$

10. (又令 m=n) 对于任何正整数 n,

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

11. 对于非负整数 p,q 和 n,

$$\sum_{k=0}^{p}C_p^k C_q^k C_{n+k}^{p+q} = C_n^p C_n^q$$

12. 对于非负整数 p,q 和 n,

$$\sum_{k=0}^{p}C_p^k C_q^k C_{n+p+q-k}^{p+q} = C_{n+p}^p C_{n+q}^q$$

13. 对于非负整数 n,k,

$$\sum_{i=0}^{n}C_i^k = C_{n+1}^{k+1}$$

14. 对于所有实数 α 和非负整数 k,

$$\sum_{j=0}^{k}C_{\alpha+j}^j = C_{\alpha+k+1}^k$$

15.

$$\sum_{k=0}^{n}\frac{2^{k+1}}{k+1}C_n^K = \frac{3^{n+1}-1}{n+1}$$

16.

$$\sum_{k=0}^{m}C_{n-k}^{m-k} = C_{n+1}^m$$

17.

$$\sum_{k=m}^{n}C_k^m C_n^k = C_n^m 2^{n-m}$$

18.

$$\sum_{k=0}^{m}(-1)^k C_n^k = (-1)^m C_{n-1}^m$$

```
 */
```

4.5 博弈论和 SG 函数

```
/*博弈论
 * 组合游戏和SG函数
 *
    组合游戏定义: 两人轮流决策;游戏状态集合有限;参与者操作时可将一状态转移到令一状态, 对任一状态都有可以到达的状态集
 *
    参与者不能操作是, 游戏结束, 按规则定胜负;游戏在有限步内结束(没有平局);参与者有游戏的所有信息.
 *必胜态和必败态: 必胜态(N—position): 当前玩家有策略使得对手无论做什么操作, 都能保证自己胜利
```

```
*               必败态(P—position)：对手的必胜态
*               组合游戏中某一状态不是必胜态就是必败态
*               对任意的必胜态，总存在一种方式转移到必败态
*               对任意的必败态，只能转移到必胜态
*找出必败态和必胜态：   1、按照规则，终止状态设为必败(胜)态
*                       2、将所有能到达必败态的状态标为必胜态
*                       3、将只能到达必胜态的状态标为必败态
*                       4、重复2—3，直到不再产生必败(胜)态
*SG函数(the Sprague—Grundy function)
*定义：游戏状态为x，sg(x)表示状态x的sg函数值，sg(x) = min{n|n ∈ N, n ∉ F(x)}，
    F(x)表示x能够达到的所有状态.
*       一个状态为必败态则sg(x)=0
*SG定理：如果游戏G由n个子游戏组成，G =
    G1 + G2 + G3 + ··· + Gn，并且第i个游戏sg函数值为sgi，则游戏G的sg函数值为g = sg1^sg2^···^sgn
*/
```

5　字符串

6 Java

```
import java.io.*;
import java.util.*;
import java.math.*;
import java.BigInteger;

public class Main{
    public static void main(String arg[]) throws Exception{
        Scanner cin = new Scnner(System.in);

        BigInteger a, b;
        a = new BigInteger(”123”);
        a = cin.nextBigInteger();
        a.add(b);//a + b
        a.subtract(b);// a — b
        a.multiply(b);// a * b
        a.divide(b);// a / b
        a.negate();// —a
        a.remainder(b);//a%b
        a.abs();//|a|
        a.pow(b);//a^b
        //.... and other math fuction, like log();
        a.toString();
        a.compareTo(b);//

    }
}
```