## Lesson3

> **高斯消元**
>
> 作用：
>
> - 可以在O(n^3)时间复杂度内求解一个包含n个方程n个未知数的多元线性方程组
>
>   解的三种情况
>
>   - 唯一解
>   - 无穷多组解
>   - 无解
>
> 矩阵的3种等价变换操作（初等行列变换，不会影响方程组的解）
>
> - 某一行乘一个非零的数
> - 交换某两行
> - 把某行的若干倍加到另外一行之上
>
>   目的：**将增广矩阵变为上三角矩阵**
>
>   解的情况
>
>   - 唯一解：完美的阶梯型
>   - 无解：化简之后，存在方程左边无未知数，右边不为0（0 = 非零）
>   - 无穷多组解：化简之后，存在0 = 0情况的方程
>
> 算法步骤
>
> ```
> 枚举每一列c：
>     找到该列绝对值最大的一行（出于精度考虑）
>     将该行换至当前最上面（没有被换到过）
>     将该行的第一个非零数变为1。（初等行变换）
>     将下面的所有行的第c列消为0
>
> 判断解的情况，若有解则从最后一行的方程开始求解
> ```
>
> 具体实现：时间复杂度O(n^3)
>
> ```
> static int N = 110; static double eps = 1e-6;
>
> static int n;
> static double[][] a = new double[N][N];
>
> static int gauss() {
>     int r, c;
>
>     for (c=0,r=0; c<n; c++) {
>         //找出第c列绝对值最大的行t
>         int t = r;
>         for (int i=r; i<n; i++)
>             if (Math.abs(a[i][c]) > Math.abs(a[t][c]))  t = i;
>
>         //判断是否为0
>         if (Math.abs(a[t][c]) < eps) continue;
>
>         //将第t行与当前最上方行互换
>         double[] tmp = a[r]; a[r] = a[t]; a[t] = tmp;
>
>         //将a[r][c]置1
>         for (int i=n; i>=c; i--) a[r][i] /= a[r][c];
>
>         //将r行下所有行c列位置清0
>         for (int i=r+1; i<n; i++)
>             if (Math.abs(a[i][c]) > eps)
>                 for (int j=n; j>=c; j--)
>                     a[i][j] -= a[i][c]*a[r][j];
>
>         r++;     //该行处理完毕
>     }
>
>     //不是唯一解
>     if (r < n) {
>         for (int i=r; i<n; i++)
>             if (Math.abs(a[i][n]) > eps) return 2;
>
>         return 1;
>     }
>
> ```

```
    //唯一解，从最后一行开始求解
    for (int i=n-1; i>=0; i--)
        for (int j=i+1; j<n; j++)
            a[i][n] -= a[i][j]*a[j][n];

    return 0;
}

public static void main(String[] args) throws Exception {
    ins.nextToken(); n = (int)ins.nval;

    //注意数据类型
    for (int i=0; i<n; i++)
        for (int j=0; j<=n; j++) { ins.nextToken(); a[i][j] = (double)ins.nval; }

    int t = gauss();

    //浮点精度处理
    for (int i=0; i<n; i++)
        if (Math.abs(a[i][n]) < eps) a[i][n] = 0;

    if (t == 0)
        for (int i=0; i<n; i++) out.printf("%.2f\n", a[i][n]);
    else if (t == 1) out.println("Infinite group solutions");
    else out.println("No solution");

    out.flush();
}
```

- e2: 解异或线性方程组

  异或：不进位加法

```
static int N = 110;

static int n;
static int[][] a = new int[N][N];

static int gauss() {
    int c, r;

    for (c=0,r=0; c<n; c++) {
        int t = r;
        for (int i=r; i<n; i++)
            if (a[i][c] != 0) {
                t = i; break;
            }

        if (a[t][c] == 0) continue;

        int[] tmp = a[r]; a[r] = a[t]; a[t] = tmp;

        for (int i=r+1; i<n; i++)
            if (a[i][c] != 0)
                for (int j=n; j>=c; j--)
                    a[i][j] ^= a[r][j];

        r++;
    }

    if (r < n) {
        for (int i=r; i<n; i++)
            if (a[i][n] != 0) return 2;

        return 1;
    }

    for (int i=n-1; i>=0; i--)
        for (int j=i+1; j<n; j++)
            a[i][n] ^= a[i][j]*a[j][n];

    return 0;
}

public static void main(String[] args) throws Exception {
    ins.nextToken(); n = (int)ins.nval;

    for (int i=0; i<n; i++)
        for (int j=0; j<=n; j++) { ins.nextToken(); a[i][j] = (int)ins.nval; }

```

```java
    int t = gauss();

    if (t == 0)
        for (int i=0; i<n; i++) out.println(a[i][n]);
    else if (t == 1) out.println("Multiple sets of solutions");
    else out.println("No solution");

    out.flush();
}
```

**组合数**

根据**数据范围**选择使用的算法

**求组合数1**:

- 范围:

  1 ≤ n ≤ 10000
  1 ≤ b ≤ a ≤ 2000

- 递推式: **C(a, b) = C(a-1, b) + C(a-1, b-1)**;

  证明:

  C(a, b)表示从a个苹果中选出b个的方案数

  把所有选法分成两种情况: 包含某一个苹果的选法C(a-1, b-1), 不包含某一个苹果的选法C(a-1, b)

具体实现: 将所有组合数预处理出来 (递推) 时间复杂度O(n^2)

```java
static int N = 2010, mod = (int)1e9+7;

static int[][] C = new int[N][N];

static void init() {
    for (int i=0; i<N; i++)      //注意从0开始
        for (int j=0; j<=i; j++)     //注意从0开始
            if (j == 0) C[i][j] = 1;
            else C[i][j] = (C[i-1][j]+C[i-1][j-1]) % mod;
}

public static void main(String[] args) throws Exception {
    ins.nextToken(); int n = (int)ins.nval;

    init();

    while (n-- > 0) {
        ins.nextToken(); int a = (int)ins.nval;
        ins.nextToken(); int b = (int)ins.nval;
        out.println(C[a][b]);
    }

    out.flush();
}
```

**求组合数2**:

- 范围

  1 ≤ n ≤ 10000,
  1 ≤ b ≤ a ≤ 10^5

- 思路: 预处理阶乘
  - fact[i] = i! mod 1e9+7
  - infact[i] = ( i! )^(-1) mod 1e9+7
  - C(a, b) = fact[a] * infact[a-b] * infact[b]

具体实现: 时间复杂度O(nlogn)

```java
static int N = 100010, mod = (int)1e9+7;

static long[] fact = new long[N], infact = new long[N];

static long qmi(long a, long k, long p) {
    long res = 1 % p;
    while (k != 0) {
        if ((k&1) == 1) res = res*a % p;
        k >>= 1;
```

```java
        a = a*a % p;
    }

    return res;
}

public static void main(String[] args) throws Exception {
    //预处理阶乘
    fact[0] = infact[0] = 1;
    for (int i=1; i<N; i++) {
        fact[i] = fact[i-1]*i % mod;
        infact[i] = infact[i-1]*qmi(i, mod-2, mod) % mod;
    }

    ins.nextToken(); int n = (int)ins.nval;
    while (n-- > 0) {
        ins.nextToken(); int a = (int)ins.nval;
        ins.nextToken(); int b = (int)ins.nval;
        out.println(fact[a]*infact[a-b] % mod*infact[b] % mod);
    }

    out.flush();
}
```

#### 求组合数3

- 范围

  $1 \le n \le 20$,
  $1 \le b \le a \le 10^{18}$,
  $1 \le p \le 10^5$

- 思路

  **卢卡斯(lucas)定理**：**C(a, b) = C(a mod p, b mod p) * C(a / p, b / p) (mod p)** （等号表示同余）

  时间复杂度O(log(p)N * p * logp) ≈ O(p * logN * logp)，当N为10^18级别时，logN≈64

  但p增大时log(p)N急速减少，所以计算次数为10^5 * 20 * 20 ≈ 4*10^7

  **证明：**

  a = ak * p^k + a(k-1) * P^(k-1) + ... + a0 * p^0（将a化为类p进制）

  b = bk * p^k + b(k-1) * P^(k-1) + ... + b0 * p^0（将b化为类p进制）

  (1+x)^p = C(p, 0) * x^0 + C(p, 1) * x^1 + ... + C(p, p) * x^p

  且p为质数，故C(p, 1), C(p, 2), ..., C(p, p-1) 模 p 为 0，故 (1+x)^p = 1+x^p (mod p) （等号同余）

  将a分解

  (1+x)^a = ((1+x)^a0) * (((1+x)^p)^a1) * ... * (((1+x)^(p^k))^ak)

  = ((1+x)^a0) * ((1+x^p)^a1) * ... * ((1+x^(p^k))^ak)

  对于x^b，等式左边其系数为C(a, b)，等式右边其系数为C(ak, bk) * C(a(k-1), b(k-1)) * ... * C(a0, b0)

  所以左=右即： C(a, b) = C(ak, bk) * C(a(k-1), b(k-1)) * ... * C(a0, b0) (mod p)

  若存在bi > ai，则C(a, b) = 0

  P.S. 最后一步对应没太懂QAQ

**Lucas 定理** 设 $p$ 是一个素数，将 $m$ ，$n$ 写成 $p$ 进制数:
$m = a_kp^k + a_{k-1}p^{k-1} + \cdots + a_1p + a_0$ ，
$n = b_kp^k + b_{k-1}p^{k-1} + \cdots + b_1p + b_0$ ,其中 $0 \le a_i, b_i \le p(i = 1, 2, \cdots, k)$ .则
$C_m^n \equiv \prod_{i=0}^{k} C_{a_i}^{b_i} (mod\ \ p)$ .

**证明** 不难看出，只需证：当 $m = m_1p + a_0$ ，$n = n_1p + b_0$ 时，$C_m^n \equiv C_{m_1}^{n_1} C_{a_0}^{b_0} (mod\ \ p)$ .

注意到，$(1+x)^m = \sum_{n=0}^{m} C_m^n x^n = \sum_{n_1=0}^{m_1} \sum_{b_0=0}^{a_0} C_m^n x^{n_1p+b_0}$ ，

$(1+x)^{m_1p+a_0} = ((1+x)^p)^{m_1}(1+x)^{a_0}$

$= (\sum_{i=0}^{p} C_p^i x^i)^{m_1}(1+x)^{a_0}$

$\equiv (1+x^p)^{m_1}(1+x)^{a_0}$

$= (\sum_{n_1=0}^{m_1} C_{m_1}^{n_1} x^{n_1p})(\sum_{b_0=0}^{a_0} C_{a_0}^{b_0} x^{b_0})$

$\equiv \sum_{n_1=0}^{m_1} \sum_{b_0=0}^{a_0} C_{m_1}^{n_1} C_{a_0}^{b_0} x^{n_1p+b_0} (mod\ \ p)$ .

而 $(1+x)^m = (1+x)^{m_1p+a_0}$ ,故 $C_m^n \equiv C_{m_1}^{n_1} C_{a_0}^{b_0} (mod\ \ p)$ .

具体实现：O(log(n)p×p×log(2)p) = O(plog(2)n)

```java
static BufferedReader inb = new BufferedReader(new InputStreamReader(System.in));
static PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));

static long qmi(long a, long k, long p) {
    long res = 1 % p;
    while (k > 0) {
        if ((k&1) == 1) res = res*a % p;
        k >>= 1;
        a = a*a % p;
    }

    return res;
}

static long C(long a, long b, long p) {
    if (b > a) return 0;

    long res = 1;
    for (long i=1, j=a; i<=b; i++, j--) {
        res = res*j % p;
        res = res*qmi(i, p-2, p) % p;
    }

    return res;
}

static long lucas(long a, long b, long p) {
    if (a<p && b<p) return C(a, b, p);
    else return C(a % p, b % p, p)*lucas(a/p, b/p, p) % p;
}

public static void main(String[] args) throws Exception {
    int n = Integer.parseInt(inb.readLine());

    while (n-- > 0) {
        //备注: StreamTokenizer读取大于10^16的整数时会发生精度丢失
```

```
        String[] tmp = inb.readLine().split("\\s+");
        // \\s表示空格,回车,换行等空白符, +号表示一个或多个的意思
        long a = Long.parseLong(tmp[0]);
        long b = Long.parseLong(tmp[1]);
        long p = Long.parseLong(tmp[2]);
        out.println(lucas(a, b, p));
    }

    out.flush();
}
```

**求组合数4**

范围

- 输入 a, b,求 C(a, b) 的值
- 1 ≤ b ≤ a ≤ 5000

思路

- 将C(a, b)分解质因数

  C(a, b) = a! / (b! (a- b)!)

  优化：对于一个质因数P，用分子的指数减去分母的指数即可得最终指数

- **a! 中包含p（质因子）的个数 = a/p + a/p^2 + ... + a/p^n (当p^n>a截止）（/表示下取整）**

  **证明:**

  a/p表示1~a中p的倍数个数（从每个p的倍数中拿出一个质因子，可能存在漏网之鱼，例如p^2中有两个p，a/p表示只从其中取了一个p），a /p^2表示1~a中p ^ 2的倍数个数（从每个p^2的倍数中继续拿出一个质因子，仍可能存在漏网之鱼，继续重复该过程）..., a /p^n表示1~a中p ^ n的倍数个数 。

  从单独每个数角度考虑，若1~a!中某数ai中含有k个质因子p，则在a/p会被计算1次，a/p^2会被计算1次，... a/p^k会被计算1次，总共会被计算k次，不重不漏

```
//求出n!中某个质因子p的个数
int get(int n, int p) {
    int res = 0;
     //计算公式 a!中包含p的个数 = a/p + a/p^2 + … + a/p^n (当a/p^n==0截止)
    //第一次循环 res += n/p
     //第二次循环 res += n/p^2
    //...
     while (n) {
        res += n/p;
         n /= p;
    }

    return res;
}
```

- 实现高精度乘法

具体实现

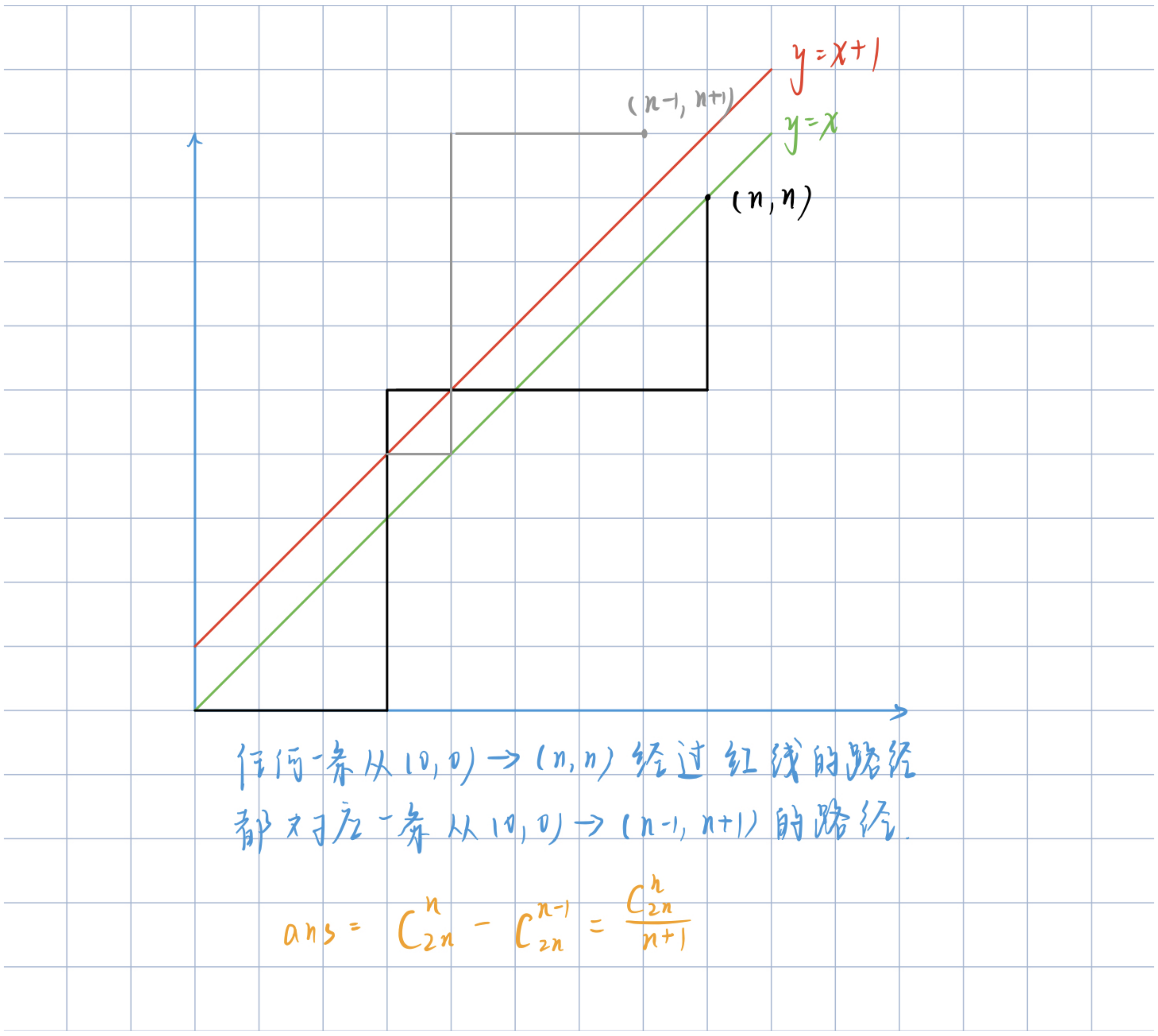
```
static int N = 5010;

static int[] primes = new int[N];
static int cnt;
static boolean[] st = new boolean[N];
static int[] sum = new int[N];

//线性筛
static void get_primes(int n) {
    for (int i=2; i<=n; i++) {
        if (!st[i]) primes[cnt++] = i;

        for (int j=0; primes[j]<=n/i; j++) {
            st[primes[j]*i] = true;

            if (i % primes[j] == 0) break;
        }
    }
}

//计算n!中质因子p的幂次
static int get(int n, int p) {
    int res = 0;
    while (n > 0) {
```

```java
        res += n/p;
        n /= p;
    }

    return res;
}

//高精度乘法
static List<Integer> mul(List<Integer> A, int b) {
    List<Integer> res = new ArrayList<>();

    for (int i=0, t=0; i<A.size() || t != 0; i++) {
        if (i < A.size()) t += A.get(i)*b;
        res.add(t % 10);
        t /= 10;
    }

    while (res.size()>1 && res.get(res.size()-1) == 0) res.remove(res.size()-1);

    return res;
}

public static void main(String[] args) throws Exception {
    ins.nextToken(); int a = (int)ins.nval;
    ins.nextToken(); int b = (int)ins.nval;

    get_primes(a);

    for (int i=0; i<cnt; i++) {
        int p = primes[i];
        sum[i] = get(a, p)-get(a-b, p)-get(b, p);
    }

    List<Integer> ans = new ArrayList<>(); ans.add(1);

    for (int i=0; i<cnt; i++)
        for (int j=0; j<sum[i]; j++)
            ans = mul(ans, primes[i]);

    for (int i=ans.size()-1; i>=0; i--) out.print(ans.get(i));

    out.flush();
}
```

**卡特兰数**

- e1：满足条件的01序列
    - **C(2n, n) - C(2n, n-1) = 1/(n+1) C(2n, n)（卡特兰数）**

      证明：等式左半部分画图进行理解证明，参见《ACWing数学知识4》 2:00




```java
static int mod = (int)1e9+7;

static long qmi(long a, long k, long p) {
    long res = 1 % p;
    while (k > 0) {
        if ((k&1) == 1) res = res*a % p;
        k >>= 1;
        a = a*a % p;
    }

    return res;
}

public static void main(String[] args) throws Exception {
    ins.nextToken(); int n = (int)ins.nval;

    int a = 2*n, b = n;
    long res = 1;
    for (int i=1, j=a; i<=b; i++, j--) {
        res = res*j % mod;
        res = res*qmi(i, mod-2, mod) % mod;
    }

    res = res*qmi(n+1, mod-2, mod) % mod;
    out.println(res);

    out.flush();
}
```