Lesson2

欧拉函数：1~n中与n互质的数的个数

计算公式

- N = p1^a1*p2^a2...pk^ak

  则欧拉函数 f(N) = N * (1-1/p1) * (1-1/p2) * .... * (1-1/pk)

  例如 N = 6 = 2 * 3;

  则 f(6) = 6(1-1/2)(1-3) = 2

- 证明：（使用**容斥原理**）
  - 从1~n中去掉p1, p2, ..., pk的所有倍数

    n - n/p1 - n/p2 - ... - n/pk （可能存在多次去除的数）
  - 加上所有pi * pj的倍数（枚举）

    n - n/p1 - n/p2 - ... - n/pk + n/p1p2 + n/p1p3 + ...
  - 减去所有pi * pj * pk的倍数（第一步减去3次，第二部加上3次，但实际需要减去）
  - 加上所有pi * pj * pm * pn的倍数...
  - 减去....
  - 化简后和f(x)相等

作用

- **欧拉定理**：若a与n互质，则有**a^phi(n) 模 n 同余 1**，即**a^(phi(n)) = 1 (mod n)** （=表示同余）

  a = 5, n= 6 则5^phi(6) mod 6 = 25 模 6 同余 1

  证明：

  假设1~n中与**n**互质的数为: a1, a2, ..., a(phi[n])

  且由于a与n互质，则aa1, aa2, ..., aa(phi[n])与n互质，且两两不相同，

  因此两组数是同一组数（在模n情况下），只可能顺序发生变化，所以两组数乘积相同

  所以a^phi(n) * (a1 * a2 * ... *an) 模 n 同余 (a1 * a2 * ... *an)

  所以a^phi(n) 模 n 同余 1

  推论

  - 当n是质数时，a^phi(p) 模 p 同余 1，且phi(p) = p-1，所以a^(p-1)模p同余1（**费马定理**）

    a ^ (p - 1) = 1 (mod p) （费马定理）

具体实现1（定义求法）：对一个数求欧拉函数，时间复杂度为O(sqrt(n))

```
// 返回欧拉函数值
```

```java
static int phi(int x) {
    int res = x;

    for (int i=2; i<=x/i; i++) {
        if (x % i == 0) {
            res = res/i*(i-1);  //整数不支持小数除法，将res*(1-1/i)变换为res/i*(i-1)
            while (x % i == 0) x /= i;
        }
    }

    if (x > 1) res = res/x*(x-1);

    return res;
}


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

    while (n-- > 0) {
        ins.nextToken(); int x = (int)ins.nval;
        out.println(phi(x));
    }

    out.flush();
}
```

具体实现2（筛法求欧拉函数）：借助线性筛

作用：O(n)时间内求出1~n号点中每个点的欧拉函数

```java
static int N = 1000010;

static int n;
static int[] phi = new int[N];  //phi[i]表示i的欧拉函数值
static int[] primes = new int[N];
static int cnt;
static boolean[] st = new boolean[N];

static long getEulers(int n) {
    phi[1] = 1;     //注意边界

    for (int i=2; i<=n; i++) {
        if (!st[i]) {
            primes[cnt++] = i;
            phi[i] = i-1;   //当i时质数时，有定义知1~i-1与i互质，故phi[i] = i-1;
        }

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

```

```java
            if (i % primes[j] == 0) {
                phi[primes[j]*i] = primes[j]*phi[i];
                break;
            }

            phi[primes[j]*i] = phi[i]*(primes[j]-1);
        }
    }

    long res = 0;
    for (int i=1; i<=n; i++) res += phi[i];
    return res;
}

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

    out.println(getEulers(n));

    out.flush();
}
```

快速幂

作用

- 快速求出 a^k mod p 的结果，时间复杂度为O(logk)，其中a, k, p的范围为 1<= a, p, k <= O(1e9)。注意暴力算法时间复杂度为O(k)

思路

- 预处理出a^(2^0) mod p, a^(2^1) mod p, ..., a^(2^logk) mod p，一共logk+1个
  - a^(2^0) = a^1
  - a^(2^1) = (a^(2^0))^2
  - ...
  - a^(2^logk) = (a^(2^(logk-1))^2
- 将a^k 拆分为若干个上述预处理结果的乘积形式

  a^k = a^(2^x1) * a^(2^x2) * ... * a^(2^xt) = a ^ (2^x1+2^x2+...+2^xt)
- 将k化为二进制数，例如(k)10 = (110110)2，则k = 2^1+2^2+2^4+2^5;

例子

- 4^5 mod 10
  - 4^(2^0) = 4 (mod10) （等号表示同余）
  - 4^(2^1) = 6 (mod10)
  - 4^(2^2) = 6 (mod10)
  - 4^5 = 4^(101) = 4^(2^0) + 4(2^2) = 24 = 4 (mod10)

具体实现：时间复杂度时O(logk)

```java
static long qmi(long a, long k, long p) {    //Java实现时需扩大为long类型
    long res = 1 % p;
```

```java
    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;

    while (n-- > 0) {
        ins.nextToken(); int a = (int)ins.nval;
        ins.nextToken(); int b = (int)ins.nval;
        ins.nextToken(); int p = (int)ins.nval;
        out.println(qmi(a, b, p));
    }

    out.flush();
}
```

应用: 快速幂求乘法逆元(把除法变为乘法)

- 若**b**|a时（a表示任意整数），使得a/b = a*x (mod m) (等号表示同余)

  两边同时乘b得a = a* x *b (mod m)

  当m是质数且b与m互质时， **b** * x = 1 (mod p)，由费马定理b^(p-1) = 1 (mod p)

  所以乘法逆元 x = b^(p-2) (mod p)
- 当m时不是质数时，使用扩展欧几里得算法求逆元

```java
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;

    while (n-- > 0) {
        ins.nextToken(); int a = (int)ins.nval;
        ins.nextToken(); int p = (int)ins.nval;

        if (a % p == 0) out.println("impossible");
```

```java
        else out.println(qmi(a, p-2, p));
    }

    out.flush();
}
```

扩展欧几里得算法

裴蜀定理：对于任意一对正整数a,b，一定存在整数x, y，使得ax+by = gcd(a, b)，即a, b的最大公约数是a与b凑出来的最小正整数

- 证明：
  - 正推：ax+by = d，则d一定是gcd(a, b)的倍数，所以ax+by凑出来的最小正整数即是gcd(a, b)
  - 反推：构造x, y（**使用扩展欧几里得算法**）

扩展欧几里得具体实现：时间复杂度O(logn)

```java
static int exgcd(int a, int b, int[] x, int[] y) {  //数组模拟C++中引用
    if (b == 0) {
        x[0] = 1; y[0] = 0;
        return a;
    }

    int d = exgcd(b, a % b, y, x);
    y[0] -= a/b*x[0];
    return d;
}

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

    while (n-- > 0) {
        ins.nextToken(); int a = (int)ins.nval;
        ins.nextToken(); int b = (int)ins.nval;

        int[] x = {0}, y = {0};
        exgcd(a, b, x, y);
        out.println(x[0]+" "+y[0]);
    }

    out.flush();
}
```

- 扩展: (x, y)不唯一，通解如下

  x = x0 - b/gcd(a, b)*k

  y = y0 + a/gcd(a, b)*k

应用：求解线性同余方程

- ax = b (mod m) （等号表示同余符号）

  4x = 3 (mod 5)　　则x = 2，x=7，....

  存在y∈Z，使得 ax = b (mod m) 等价于 ax = my + b，即ax - my = b，另m=-m，所以ax + my = b

  则题意相当于给定a, m, b求x，y。且有解的充分必要条件为gcd(a, m) | b

```
static int exgcd(int a, int b, int[] x, int[] y) {
    if (b == 0) {
        x[0] = 1; y[0] = 0;
        return a;
    }

    int d = exgcd(b, a % b, y, x);
    y[0] -= a/b*x[0];
    return d;
}

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

    while (n-- > 0) {
        ins.nextToken(); int a = (int)ins.nval;
        ins.nextToken(); int b = (int)ins.nval;
        ins.nextToken(); int m = (int)ins.nval;

        int[] x = {0}, y = {0};
        int d = exgcd(a, m, x, y);

        if (b % d == 0) out.println((long)x[0]*(b/d) % m);   //注意数据范围，乘法可能越界
        else out.println("impossible");
    }

    out.flush();
}
```

中国剩余定理

定义

- 给定一组两两互质的序列：m1, m2, .., mk（两两互质），求解线性同余方程组
    - x = a1 (mod m1)　x mod m1 = a1
    - x = a2 (mod m2)　x mod m2 = a2
    - ...
    - x = ak (mod mk)　x mod mk = ak

  令M = m1 * m2 * ... * mk

  令Mi = M/mi，故Mi与mi互质，令Mi^(-1) (mod mi)表示Mi 模 mi的逆

则通解 x = a1 * M1 * M1^(-1) + a2 * M2 * M2^(-1) + ... + ak * Mk * Mk^(-1)

求逆可以使用扩展欧几里得算法解 ax = 1 (mod m)

- 证明：

  对于从1到k的每一项，分别模m1， m2, ...， mk，则可知与原方程组等价

应用：表达整数的奇怪方式

- x mod a1 = m1　　x = k1*a1+m1
- x mod a2 = m2　　x = k2*a2+m2
- ...
- x mod ak = mk　　x = kk*ak+mk

  k1 * a1+m1 = k2 * a2 + m2　　k1 * a1 - k2 * a2 = m2 - m1

  通过扩展欧几里得算法求解，有解则等价于 gcd(a1, a2) | m2 - m1，且 x = k1*a1+m1

  且k1 * a1 - k2 * a2 = m2 - m1 通解为

  k1' = k1 + a2/gcd(a1, a2) *k　(k∈Z)

  k2' = k2 + a1/gcd(a1, a2) *k,

  所以x = (k1 + a2/gcd(a1, a2) *k)a1 + m1 = a1 * k1 + m1 + k * (a1*a2/gcd(a1, a2))

  a1 * k1 + m1 + k*lcm(最小公倍数)(a1, a2)，所以 x = k * a + m (**a = lcm(a1, a2)**, **m = a1 * k1 + m1**)

  合并n-1次 得 x = k*a + m，即 x mod a = m，即**x = m mod a的最小正整数**

具体实现

```java
static long exgcd(long a, long b, long[] x, long[] y) {
    if (b == 0) {
        x[0] = 1; y[0] = 0;
        return a;
    }

    long d = exgcd(b, a % b, y, x);
    y[0] -= a/b*x[0];
    return d;
}

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

    boolean flag = true;
    ins.nextToken(); long a1 = (long)ins.nval;
    ins.nextToken(); long m1 = (long)ins.nval;

    while (n-- > 1) {
        ins.nextToken(); long a2 = (long)ins.nval;
        ins.nextToken(); long m2 = (long)ins.nval;

        long[] x = {0}, y = {0};
        long d = exgcd(a1, a2, x, y);
```

```java
            if ((m2-m1) % d != 0) {
                flag = false; break;
            }

            long k1 = x[0]*(m2-m1)/d;
            long t = a2/d;
            k1 = (k1 % t+t) % t;     //正数范围内最小化k1

            //更新m1, a1
            m1 = a1*k1+m1;
            a1 = Math.abs(a1*t);     //abs保证最终x为正数
        }

        if (flag) out.println((m1%a1+a1)%a1);
        else out.println(-1);

        out.flush();
    }
```