算法基础（十九）：数学基础 - 数论4 - 快速幂

快速幂

基本思想

如果两个数都是合数，可先将两个数分别分解质因数，再看两个数是否含有相同的质因数。如果没有，这两个数是互质数

快速幂可以快速求出$a^k mod p$的结果，时间复杂度是`O(logk)`

其中$1 \le a, p, k \le 10^9$

如果我们暴力计算的话，循环`k`次，复杂度是`10^9`级别，但是使用快速幂，直接降到了`9log10`，**非常的逆天**

核心思想：

求：$a^k mod p$

我们先预处理一些数的取模结果：

$a^{2^0}, a^{2^1}, a^{2^2} \ldots a^{2^{logk}}$

一共有`logk`个数，也就是时间复杂度

接下来我们将`a^k`拆成以下的形式：

$a^k = a^{2^{x_1}} * a^{2^{x_2}} * a^{2^{x_3}} * \ldots a^{2^{x_t}} = a^{2^{x_1}+2^{x_2}+2^{x_3}+\ldots 2^{x_t}}$

也就是$k = 2^{x_1} + 2^{x_2} + 2^{x_3} + \ldots 2^{x_t}$

这种拆法是很显然成立的，因为`k`写成二进制的模式就是一堆`2`的指数的和

我们在预处理的时候有：
$a^{2^1} = (a^{2^0})^2$

$a^{2^2} = (a^{2^1})^2$

...

$a^{2^{logk}} = (a^{2^{logk-1}})^2$

然后根据`k`的二进制结果加起来即可

另外需要注意的是，由于模运算：`(a*b)modp = (amodp*bmodp)modp`

所以

$a^{2^i} mod p = (a^{2^{i-1}})^2 mod p = (a^{2^{i-1}} mod p * a^{2^{i-1}} mod p) mod p$

所以每一个数模`p`都是上一个数的平方模`p`，都是上一个数模`p`的平方再模`p`，所以在计算的时候直接计算这个数模`p`的结果，在下次计算时直接平方再摸`p`

代码实现

给定 n 组 a_i, b_i, p_i，对于每组数据，求出 $a_i^{b_i} \bmod p_i$ 的值。

输入格式

第一行包含整数 n。

接下来 n 行，每行包含三个整数 a_i, b_i, p_i。

输出格式

对于每组数据，输出一个结果，表示 $a_i^{b_i} \bmod p_i$ 的值。

每个结果占一行。

数据范围

$1 \le n \le 100000$,
$1 \le a_i, b_i, p_i \le 2 \times 10^9$

输入样例：

```
2
3 2 5
4 3 9
```

输出样例：

```
4
1
```

```
#include<iostream>
#include<algorithm>
using namespace std;
//在计算中间结果的时候可能会爆int，需要用LL
typedef long long LL;

//开始计算快速幂
int qmi(int a, int k, int p)
{
    //答案先记成1
    int res = 1;
    //求k的二进制表示
    while(k)
    {

        //如果k的末位为1，则加上结果
        if(k & 1) res = (LL)res * a % p;

        //删掉k的末位，注意此处是k = k >> 1，不能写成k >> 1
        k >>= 1;

        //每次删掉k的末位的时候，然后将a变成下一个
        a = (LL)a * a % p;
    }

    return res;
```

```
}

int main()
{
    int n;
    scanf("%d", &n);

    while(n --)
    {
        int a, k, p;
        scanf("%d%d%d", &a, &k, &p);
        //cout << "dd";
        printf("%d\n", qmi(a, k, p));
    }

    return 0;
}
```

稍微分析一下爆`int`的情况

由于题目中说明了数据范围是小于`2*10^9`的，当`a = 2*10^9`的时候，我们在计算`a = (LL)a * a % p;`的时候，强制类型转换优先级大于乘除运算，先将`a`转换为`LL`类型，然后再计算的时候`c`语言进行自动类型转换，将表达式中的所有的变量的类型都转换为范围最大的那个类型，也就是`LL`这样就不会爆`int`了

所以如果我们不进行类型转换的话，计算`a * a`的时候会导致超出`int`类型范围，出现莫名的错误

注意为什么不会爆`LL`，因为我们每一次的计算结果都是模`p`的，这就会导致本次计算的结果`res`，`a`都不会超过`2 * 10 ^9`，那么在下次计算的过程中当然不会爆`LL`

还需要注意的是，我们每次预处理`a`的时候是符合模乘法运算法则的，这里不再赘述

我们在处理`res`的时候，由于

$$a^k modp = (a^{2^{x_1}} * a^{2^{x_2}} * a^{2^{x_3}} * \ldots a^{2^{x_t}}) modp = (a^{2^{x_1}} modp * a^{2^{x_2}} modp * a^{2^{x_3}} modp * \ldots a^{2^{x_t}} modp) modp$$

我们每次循环其实是算出了从`1 ~ logk`的所有的$a^{2^{x_i}} modp$的，只是在需要的时候(`k`的末位为`1`)，然后乘上结果

也就是代码中`res = (LL)res * a % p;`

符合运算法则

举一个例子:
$3^{101} = 3^5 = 3^{2^2} * 3^0 * 3^{2^0}$

`k = 101`末位为`1`， 所以`res * 1 modp`

也就是其本身

`k`变成`10`，然后计算`0`对应的`a`也就是3^{2^1}

`k = 10`，`k`的末位为`0`，`res`不乘上结果

`k`变成`1`，然后计算`1`对应的`a`也就是3^{2^2}

`k = 1`，`k`的末位为`1`，`res * a(3^4)modp`得到最终结果

`k`变成`0`，计算一下，然后退出循环

快速幂求逆元

基本思想

基本定义：

> 若整数 b，m 互质，并且对于任意的整数 a，如果满足 $b|a$，则存在一个整数 x，使得 $a/b \equiv a \times x \pmod{m}$，则称 x 为 b 的模 m 乘法逆元，记为 $b^{-1} \pmod{m}$。
> b 存在乘法逆元的充要条件是 b 与模数 m 互质。当模数 m 为质数时，b^{m-2} 即为 b 的乘法逆元。

简单来说，如果`a/b`是一个整数，表示`b | a`，我们希望对于一个`b`，可以找到一个`x`，满足:

$a/b \equiv a * x(modm)$

则称`x`为`b`的乘法逆元，记作$x = b^{-1}$

这样我们就可以将所有除以`b`的情况，转化为乘`b`的逆元的情况，注意这里的`a`是任意的

简化版本就是:

正整数`a, n`，如果有 `ax ≡ 1(mod n)`，则称 `x` 的**最小正整数解**为 `a`模 `n`的逆元

基本性质:

对于$a/b \equiv a * x(modm)$

根据同余的法则：`a≡b(mod m)，x≡y(mod m)，则ax≡by(mod m)`

我们左右两边同乘`b`得到:

$a \equiv b * a * b^{-1}(modm)$

因为对任意的一个整数`a`，都存在一个`x`满足:

$a \equiv b * a * b^{-1}(modm)$

那么对于不同的`a`来说，`x`是相同的，也就是b^{-1}是相同的，那么当`x`与`m`互质的时候，根据同余的法则：

`ac ≡ bc(mod m)，且c和m互质，则a ≡ b(mod m)`

于是我们可以得到:

$b * b^{-1} \equiv 1(modm)$ （1）

因为不管`x`是否与`m`互质，`b^-1`这个值是不变的，在互质情况下满足上面等式，那在不互质的情况下b^{-1}没有改变，所以也满足，所以$b * b^{-1} \equiv 1(modm)$与`x`是否与`m`互质无关

当`m`为质数的时候根据费马小定理:

$b^{m-1} \equiv 1(modm)$，其中`m`是质数，`b`与`m`互质（2）

由（1）（2）我们可以得到:

$b * b^{-1} \equiv b^{m-1}(modm)$

即:

$b^{-1} \equiv b^{m-2}(modm)$

注意逆元的严格定义是**满足上面等式的最小正整数解**，我们记

$r = b^{m-2}(modm)$

则：f

$b^{-1} = r + km$

所以逆元`x`为：

$b^{m-2}(modm)$

代码实现

给定 n 组 a_i, p_i，其中 p_i 是质数，求 a_i 模 p_i 的乘法逆元，若逆元不存在则输出 `impossible`。

注意：请返回在 $0 \sim p-1$ 之间的逆元。

乘法逆元的定义

> 若整数 b，m 互质，并且对于任意的整数 a，如果满足 $b|a$，则存在一个整数 x，使得 $a/b \equiv a \times x \pmod m$，则称 x 为 b 的模 m 乘法逆元，记为 $b^{-1} \pmod m$。
> b 存在乘法逆元的充要条件是 b 与模数 m 互质。当模数 m 为质数时，b^{m-2} 即为 b 的乘法逆元。

输入格式

第一行包含整数 n。

接下来 n 行，每行包含一个数组 a_i, p_i，数据保证 p_i 是质数。

输出格式

输出共 n 行，每组数据输出一个结果，每个结果占一行。

若 a_i 模 p_i 的乘法逆元存在，则输出一个整数，表示逆元，否则输出 `impossible`。

数据范围

$1 \le n \le 10^5$,
$1 \le a_i, p_i \le 2 * 10^9$

输入样例：

```
3
4 3
8 5
6 3
```

输出样例：

```
1
2
impossible
```

```
#include<iostream>
using namespace std;
typedef long long LL;

```

```

//求a^(p-2)次幂modp的结果
int qmi(int a, int k, int p)
{
    int res = 1;

    while(k)
    {
        if(k & 1) res = (LL) res * a % p;
        k >>= 1;
        a = (LL)a * a % p;
    }
    return res;
}


int main()
{
    int n;
    scanf("%d", &n);

    while(n --)
    {
        int a, p;
        scanf("%d%d", &a, &p);
        int res = qmi(a, p - 2, p);
        //注意此处逆元必须保证a与p互质
        //由于题目中说了p是质数
        //所以只有a是p的倍数的时候a，p才不互质，这时表示逆元不存在
        //注意此处不能用res = 0来判断逆元不存在，因为当p = 2，a = 4的时候快速幂返回的结果是1，不是0
        if(a % p) cout << res << endl;
        else cout << "impossible" << endl;
    }

    return 0;
}
```

这个算法的时间复杂度是`O(log(p))`