快速幂取模算法

所谓的快速幂，实际上是快速幂取模的缩写，简单的说，就是快速的求一个幂式的模(余)。在程序设计过程中，经常要去求一些大数对于某个数的余数，为了得到更快、计算范围更大的算法，产生了快速幂取模算法。

我们先从简单的例子入手：求 $a^b \bmod c =$ 几。

算法 1.首先直接地来设计这个算法：

```
int ans = 1;
for(int i = 1;i<=b;i++)
{
ans = ans * a;
}
ans = ans % c;
```

这个算法的时间复杂度体现在 for 循环中，为 ***O（b）***.这个算法存在着明显的问题，如果 a 和 b 过大，很容易就会溢出。

那么，我们先来看看第一个改进方案：在讲这个方案之前，要先有这样一个公式：

a^b mod c=(a mod c)^b mod c 这个公式大家在离散数学或者数论当中应该学过，不过这里为了方便大家的阅读，还是给出证明：

引理 1：

公式: $(ab) \bmod c = [(a \bmod c) \times (b \bmod c)] \bmod c$

证明：

$a \bmod c = d \Rightarrow a = tc + d$

$b \bmod c = e \Rightarrow b = kc + e$

$ab \bmod c = (tc+d)(kc+e) \bmod c$

$= (tkc^2 + (te+dk)c + de) \bmod c$

$= de \bmod c = [(a \bmod c) \times (b \bmod c)] \bmod c$

上面公式为下面公式的引理，即积的取余等于取余的积的取余。

公式：$a^b \bmod c = (a \bmod c)^b \bmod c$

证明:$[(a \bmod c)^b] \bmod c$

$=[((a \bmod c) \bmod c)^b] \bmod c$(由上面公式的迭代)

$[(a \bmod c)^b] \bmod c = a^b \bmod c$

证明了以上的公式以后，我们可以先让 a 关于 c 取余，这样可以大大减少 a 的大小，于是不用思考的进行了改进：

算法 2：

```
int ans = 1;
a = a % c;                    //加上这一句
for(int i = 1;i<=b;i++)
{
   ans = ans * a;
}
ans = ans % c;
```

聪明的读者应该可以想到，既然某个因子取余之后相乘再取余保持余数不变，那么新算得的 ans 也可以进行取余，所以得到比较良好的改进版本。

算法 3：

```
int ans = 1;
a = a % c;                      //加上这一句
for(int i = 1;i<=b;i++)
{
   ans = (ans * a) % c;         //这里再取了一次余

}
ans = ans % c;
```

这个算法在时间复杂度上没有改进，仍为 ***O(b)***，不过已经好很多的，但是在 c 过大的条件下，还是很有可能超时，所以，我们推出以下的**快速幂算法**。

快速幂算法依赖于以下明显的公式，我就不证明了。

$$a^b \bmod c = ((a^2)^{b/2}) \bmod c, b\text{是偶数}$$

$$a^b \bmod c = ((a^2)^{b/2} \times a) \bmod c, b\text{是奇数}$$

有了上述两个公式后，我们可以得出以下的结论：

1、如果 b 是偶数，我们可以记 k = a^2 mod c，那么求**$(k)^{b/2}$ mod c** 就可以了。

2、如果 b 是奇数，我们也可以记 k = a^2 mod c，那么求 $((k)^{b/2}$ mod c × a) mod c =**$((k)^{b/2}$ mod c * a) mod c** 就可以了。

那么我们可以得到以下算法：

算法 4：

```
int ans = 1;
a = a % c;
if(b%2==1)
    ans = (ans * a) mod c;        //如果是奇数，要多求一步，可以提前算到 ans 中
k = (a*a) % c;                     //我们取 a2 而不是 a
for(int i = 1;i<=b/2;i++)
{
   ans = (ans * k) % c;
}
ans = ans % c;
```

我们可以看到，我们把时间复杂度变成了 **O(b/2)**.当然，这样子治标不治本。但我们可以看到，当我们令 k = (a * a) mod c 时，状态已经发生了变化，我们所要求的最终结果即为 $(k)^{b/2}$ mod c 而不是原来的 a^b mod c，**所以我们发现这个过程是可以迭代下去的**。当然，对于奇数的情形会多出一项 a mod c，所以为了完成迭代，当 b 是奇数时，我们通过 ans = (ans * a) % c;来弥补多出来的这一项，此时剩余的部分就可以进行迭代了。

形如上式的迭代下去后，当 b=0 时，所有的因子都已经相乘，算法结束。于是便可以在 ***O（log b）***的时间内完成了。于是，有了最终的算法：快速幂算法。

算法 5：快速幂算法

```
int ans = 1;
a = a % c;
while(b>0)
{
   if(b % 2 == 1)
        ans = (ans * a) % c;
   b = b/2;
   a = (a * a) % c;
}
```

将上述的代码结构化，也就是写成函数：

```
int PowerMod(int a, int b, int c)
{
     int ans = 1;
     a = a % c;
     while(b>0)
          {
               if(b % 2 = = 1)
                  ans = (ans * a) % c;
               b = b/2;
               a = (a * a) % c;
          }
     return ans;
}
```

本算法的时间复杂度为 O（$\log_2 b$），能在几乎所有的程序设计（竞赛）过程中通过，是目前最常用的算法之一。

以下内容仅供参考：

扩展：有关于快速幂的算法的推导，还可以从另一个角度来想。

$a^b \bmod c = ?$ 求解这个问题，我们也可以从进制转换来考虑：

将 10 进制的 b 转化成 2 进制的表达式：$b_{(10)} = \overline{a_n a_{n-1} \ldots a_1 a_0}_{(2)}$

那么，实际上，$b = a_n \cdot 2^n + a_{n-1} \cdot 2^{n-1} + \ldots a_1 \cdot 2^1 + a_0$.

$$a^b = a^{a_n \cdot 2^n + a_{n-1} \cdot 2^{n-1} + \ldots a_1 \cdot 2^1 + a_0}$$
$$= a^{a_n \cdot 2^n} \cdot a^{a_{n-1} \cdot 2^{n-1}} \cdot \ldots a^{a_1 \cdot 2} \cdot a^{a_0}$$

所以

$$a^b \bmod c = (a^{a_n \cdot 2^n} \cdot a^{a_{n-1} \cdot 2^{n-1}} \cdot \ldots a^{a_1 \cdot 2} \cdot a^{a_0}) \bmod c$$
$$= [(a^{a_n \cdot 2^n} \bmod c) \cdot (a^{a_{n-1} \cdot 2^{n-1}} \bmod c) \cdot \ldots (a^{a_0} \bmod c)] \bmod c$$

注意此处的 a_k 要么为 0，要么为 1，如果某一项 $a_k = 0$，那么这一项就是 1，这个对应了上面算法过程中 b 是偶数的情况，为 1 对应了 b 是奇数的情况[不要搞反了，读者自己好好分析，可以联系 10 进制转 2 进制的方法]，我们从 $(a^{a_0} \bmod c)$ 依次乘到 $(a^{a_n \cdot 2^n} \bmod c)$。对于每一项的计算，计算后一项的结果时用前一项的结果的平方取余。对于要求的结果而言，为 $a_k = 0$ 时 ans 不用把它乘起来，[因为这一项值为 1]，为 1 项时要乘以此项再取余。这个算法和上面的算法在本质上是一样的，读者可以自行分析，这里我说不多说了，希望本文有助于读者掌握快速幂算法的知识点，当然，要真正的掌握，不多练习是不行的。

例题 1：高级机密

源程序名 secret. pas

输入文件名 secret.in

输出文件名 secret.out

- 问题描述

在很多情况下，我们需要对信息进行加密。特别是随着 Internet 的飞速发展，加密技术就显得尤为重要。

很早以前，罗马人为了在战争中传递信息，频繁地使用替换法进行信息加密。然而在计算机技术高速发展的今天，这种替换法显得不堪一击。因此密码研究人员正在试图寻找一种易于编码、但不易于解码的编码规则。

目前比较流行的编码规则称为 RSA，是由美国麻省理工学院的三位教授发明的。这种编码规则是基于一种求模算法的：对于给出的三个整数 a，b，c，计算 a 的 b 次方除以 c 的余数。

你的任务是编写一个程序，计算 $a^b \bmod c$ 。

- 输入数据

只有一行，三个正整数 a，b，c。其中 $1 \leqslant a,b < c \leqslant 32768$

- 输出数据

只有一个数，即 $a^b \bmod c$ 的值。

- 样例输入

2 6 11

- 样例输出

9

【提示】

利用结论：a*b mod c=a*(b mod c) mod c

例题 2: A^B mod C

Time Limit:1000MS　Memory Limit:65536K
提交：http://acm.cugb.edu.cn/JudgeOnline/showproblem?problem_id=1222

源程序名　**powerm. pas**
输入文件名　**powerm.in**
输出文件名　**powerm.out**

Description
数论课上，老师给 DreamFox 安排了一项任务，用编程实现 A 的 B 次方模 C。这个当然难不了 ACMer。于是 DreamFox 回去后就开始用代码实现了。

Input
三个整数，a，b，c　$(0 \leqslant a,c < 2^{31}, 0 \leqslant b < 2^{63})$。

Output
一个整数,a^b mod c 的结果。

Sample Input
5 100000000000000 12830603

Sample Output
5418958

算法一：

- **根据性质(a×b) mod c=((a mod c)×(b mod c)) mod c，将a^b变形为b个a相乘的形式，然后进行模运算。**

算法二：

$$a^b \bmod c = (a^{b\,div\,2} \times a^{b\,div\,2} \times a^{b \bmod 2}) \bmod c$$
$$= ((a^{b\,div\,2} \bmod c)^2 \times a^{b \bmod 2}) \bmod c$$

边界条件：$b=0 \quad exit(1)$

$b=1 \quad exit(a)$