快速幂取模算法



**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 *ab* mod *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：

**ab 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*) mod *c*  [(*a* mod *c*)(*b*mod *c*)]mod *c*

证明：

*a* mod *c*  *d*  *a*  *tc*  *d*

*b*mod *c*  *e*  *b*  *kc* *e*

*ab*mod *c*  (*tc*  *d*)(*kc* *e*) mod *c*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | (*tkc*  (*te*  *dk*)*c*  2 |  | *de*) mod *c* |

*de*mod *c*  [(*a* mod *c*)(*b*mod *c*)]mod *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* | | | | | | |  | mod *c*  (*a* mod | | | | | | | | | |  | *c*)  *b* |  | mod *c* | |
|  | | | 证明：[(*a* | | | |  | mod *c*) ] mod *c*  *b* | | | | | | | | | |
|  | | | | [((*a* mod *c*) mod | | | | | | |  | *c*) ] mod *c*(由上面公式的迭代)  *b* | | | | | | | | | | | | | |
|  | [(*a* mod | | | |  | *c*) ] mod *c*  *b* | | | | | | |  |  |  | *a*  *b* |  | | mod *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：

第 1 页 共 6 页

快速幂取模算法



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* |  | mod *c* |  | ((*a* ) ) mod  2 *b*/ 2 |  | *c*,*b*是偶数 | | |
|  | *a*  *b* |  | mod *c* |  | ((*a* )  2 *b*/ 2 *a*) mod | | |  | *c*,*b*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 = a2 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 = a2 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;**

第 2 页 共 6 页

快速幂取模算法



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 ab 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**（**log2b**），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。

第 3 页 共 6 页

快速幂取模算法



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。

*ab* mod *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*( *a a* ...*a a*

10) 1 1 0

*n n* (2)

那么，实际上， .

*b a* 2 *a* 1 2*n* ...*a* 2 *a*

*n*

*n* 1 1

*n* 1 0

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *a*  *b* | | |  |  |  | *a* 2 *a* 2 ...*a* 2 *a*  *n*  *n*1 1  *a*  *n n*1 1 0 | | | | |
|  |  |  | *a* 2  *n*  *a*  *n* | | | | |  | *a*  *a*  2  *n*1  *n*1 |  | ... *a a*  *a a*  2    1 0 | |

所以

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *a*  *b* | | |  | mod *c* | | |  | *n*  (*a*  *a* 2  *n* | | |  | *n*1  *a*  *a* 2  *n*1 | |  | *a* 2  ...*a*  1 | | |  | *a*  *a* ) mod  0 |  | *c* |
|  |  |  | *n*  [(*a* mod  *a* 2  *n* | | | | | | |  | *n*1  *c*) (*a*  *a*  2  *n*1 | | |  | mod | |  | *c*) ...(*a* mod  *a*  0 | | | | |  | *c*)] mod |  | *c* |
| 注意此处的 | | | | | |  | *a* 要么为 0，要么为 1，如果某一项 *a*  0 ，那么这一项就是 1，这个对应了上  *k k* | | | | | | | | | | | | | | | | | | | | |

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( mod )  *aa*0 *c* |  | *a*  *n*  依次乘到( mod )  *a* 2 *c*  *n* |  | 。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*  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

*k*

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。

第 4 页 共 6 页

快速幂取模算法



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 abmod c 。

**-** 输入数据

只有一行，三个正整数 a，b，c。其中 1≤a,b＜c≤32768

**-** 输出数据

只有一个数，即 abmod 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

第 5 页 共 6 页