

电子科技大学信息与软件工程学院

实 验 报 告

学 号 2018091618008

姓 名 袁昊男

(实验) 课程名称 信息安全数学基础

理论教师 熊虎

实验教师 熊虎

电子科技大学

实验报告

学生姓名：袁昊男 学号：2018091618008 指导教师：熊虎

实验地点：信软楼 303 实验时间：2019.12.14

一、实验名称：模指数运算的实现

二、实验学时：2 学时

三、实验目的：

- 1、熟悉一种模指数运算算法；
- 2、运用高级程序设计语言完成一种模指数运算算法的程序，加深对模指数运算的理解；
- 3、提供该算法的源代码及测试用例，给出运行结果分析。

四、实验原理：

1、算法 2.3.1 经典模乘法。

输入：正整数 x 和 y ，模 m ，全都是 b 进制表示。

输出： $x \cdot y \pmod{m}$ 。

- (1) 计算 $x \cdot y$ （使用算法 1.5.3）。
 - (2) 计算 $x \cdot y$ 除以 m 的余数 r （使用算法 1.5.4）。
 - (3) 返回 (r) 。
-

- 2、模的指数运算 $a^k \pmod{m}$ 可以运用重复平方乘算法有效实现，这一运算在很多密码学协议中都有重要用处。算法 2.3.2 是基于以下结果，设 k 的二进制表示为 $k = \sum_{i=0}^t k_i 2^i$ ，其中 $k_i \in \{0,1\}$ ，则

$$a^k = \prod_{i=0}^t a^{k_i 2^i} = (a^{2^0})^{k_0} (a^{2^1})^{k_1} \cdots (a^{2^t})^{k_t}$$

算法 2.3.2 \mathbb{Z}_m 中重复平方乘算法。

输入： $a \in \mathbb{Z}_m$ ，整数 $0 \leq k < m$ ，其二进制表示为 $k = \sum_{i=0}^t k_i 2^i$ 。

输出： $a^k \pmod{m}$ 。

- (1) 令 $b \leftarrow 1$ ，如果 $k = 0$ ，则返回 (b) 。
 - (2) 令 $A \leftarrow a$ 。
-

-
- (3) 如果 $k_0 = 1$ ，则令 $b \leftarrow a$ 。
 - (4) 对 i 从 i 到 t ，作
 - (4.1) 令 $A \leftarrow A^2 \pmod{m}$ ；
 - (4.2) 如果 $k_i = 1$ ，则令 $b \leftarrow A \cdot b \pmod{m}$ 。
 - (5) 返回 (b) 。
-

五、 实验内容：

- 1、掌握常用的模指数算法的实现方法，即：求 $a^e \pmod{m}$ 的值；
- 2、算法采用重复平方乘的方法实现；
- 3、要求 m 的大小为 512 比特~1024 比特， a 和 e 可以随机产生；
- 4、提供算法的源代码及测试用例，给出运行结果分析；
- 5、可将运行结果与标准大数库中的运算进行效率比较；
- 6、按照标准实验报告整理实验内容。

六、 实验器材（设备、元器件）：

电脑一台。

七、 实验步骤：

- 1、学习 \mathbb{Z}_m 中重复平方乘算法，理解算法实现的过程；
- 2、分析题目需求，设计数据结构；
- 3、编码实现，并按测试用例进行输入输出测试；
- 4、分析实验结果，与标准大数库的运算效率作对比。

八、 实验结果与分析（含重要数据结果分析或核心代码流程分析）

1、代码分析

```
1. void ModPow(int a, char k[], char m[])
2. {
3.     int i, j, q;
4.     int len = ToBinary(k);
5.     if (binary[0] == 0)
6.         res[0] = '1';
7.
8.     if (binary[0] == 1)
9.         res[0] = a;
10.    for (i = 1; i < len; i++)
11.    {
12.        int len3=Multiplication(A, A, temp);
13.        char temp4[MAX];
14.        for (j = 0, q = len3 - 1; q >= 0; j++, q--)
15.            temp4[j] = (char)('0' + temp[q]);
16.        memset(A, 0, sizeof(A));
17.    }
```

```

18.         strncpy(A, temp4, len3);
19.
20.         Division(A, m, temp_q, A1);
21.         int len2 = strlen(tempnew);
22.         strcpy(A, tempnew);
23.         if (binary[i] == 1)
24.         {
25.             int len3 = Multiplication(A, res, temp);
26.             char temp4[MAX];
27.             for (j = 0, q = len3 - 1; q >= 0; j++, q--)
28.                 temp4[j] = (char)('0' + temp[q]);
29.             memset(res, 0, sizeof(res));
30.             strncpy(res, temp4, len3);
31.             Division(res, m, temp_q, A1);
32.             int len2 = strlen(tempnew);
33.             strcpy(res, tempnew);
34.         }
35.     }
36. }

```

说明：本函数求 $a^k \bmod m$ ，带有 3 个参数：int 型底数 a、保存在 char 数组中的幂次 k 以及模数 m。由于要求 m 的大小为 512 比特~1024 比特，即二进制数 m 至多可以表示 155~309 位十进制数 ($\lg 2^{512} = 512 \cdot \lg 2 \approx 155$, $\lg 2^{1024} = 1024 \cdot \lg 2 \approx 309$)，因此 m 的数组大小设置为 500。该函数首先调用 ToBinary 函数，将幂次 k 转换为二进制数保存在全局数组变量 binary 中；之后再按算法 2.3.2 的步骤依次执行，用到了实验一中的 Multiplication 大数乘法函数以及 Division 大数除法函数。最终将结果保存在全局数组变量 res 中。（完整代码见文末）

2、测试用例

注：为便于说明算法原理及测试编码本身的有效性和正确性，本测试用例中未使用 309 位十进制数（即 1024 位二进制）

序号	输入(a, k, m)	预期输出	实际输出
1	(5, 596, 1234)	1013	1013
2	(6, 2352, 6729475)	1349586	1349586
3	(12332, 0, 13682169)	1	1

3、运行截图

(1) 测试用例 1

a) 算法过程

$$(596)_{10} = (1001010100)_2$$

i	0	1	2	3	4	5	6	7	8	9
k_i	0	0	1	0	1	0	1	0	0	1

A	5	25	625	681	1011	369	421	779	947	925
b	1	1	625	625	67	67	1059	1059	1059	1013

b) 运行结果

```

C:\袁昊男\学习\大二上\信息安全数学基础\...
5^596 (mod 1234) = 1013
duration: 0.001000 seconds
请按任意键继续. . .

```

注：结果与手工计算验证正确。

(2) 测试用例 2

a) 算法过程

$$(2352)_{10} = (100100110000)_2$$

i	0	1	2	3	4
k_i	0	0	0	0	1
A	6	36	1296	1679616	6315856
b	1	1	1	1	6315856
i	5	6	7	8	9
k_i	1	0	0	1	0
A	3963711	2435396	3025541	2113906	5104161
b	2166766	2166766	2166766	3242946	3242946
i	10	11	/	/	/
k_i	0	1	/	/	/
A	3646296	3655791	/	/	/
b	3242946	1349586	/	/	/

b) 运行结果

```

C:\袁昊男\学习\大二上\信息安全数学基础\...
6^2352 (mod 6729475) = 1349586
duration: 0.001000 seconds
请按任意键继续. . .

```

注：结果与手工计算验证正确。

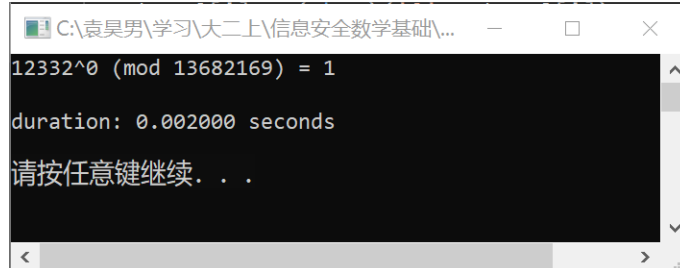
(3) 测试用例 3

a) 算法过程

$$(0)_{10} = (0)_2;$$

$b \leftarrow 1, k = 0$, 返回(b)。

b) 运行结果



注：结果与手工计算验证正确。

4、效率对比

测试次数	本算法运行时间（秒）	标准库运行时间（秒）
1	0.001	0.001
2	0.002	0.000
3	0.001	0.001
4	0.003	0.001
5	0.001	0.001
平均时间（秒）	0.0016	0.0008
效率比	50.00%	

九、 总结及心得体会：

在密码学中使用的整数在大多数情况下是在 \mathbb{Z}_m 中的情况，所有整数都是模 m 的，其中 m 是多精度整数。模乘法比多精度数乘法要复杂得多，不仅要考虑多精度乘法，还要用到模约减的方法。计算模约减的方法是借用多精度数除法算法，如实验一中的大数除法算法。通过本实验，选用 \mathbb{Z}_m 中重复平方乘算法，实现了模数 m 在 512~1024 比特间的一般效率的大数指数运算，与标准大数库的运行效率相比仍有差距。还可以选用 Montgomery 约减、乘法、指数运算来提高效率。

通过本次实验，我掌握了重复平方乘算法，编程实现了大数指数运算，对密码学的编码有了初步的认识，本实验所写的函数还可方便其他函数调用，如扩展的欧几里得算法、RSA 算法等，有较大的实用性。

十、 对本实验过程及方法、手段的改进建议：

本实验可以放在对应章节的学习中，统一安排在期末进行学生完成的时间压力较大。

报告评分：

指导教师签字：

附：完整代码

```
1. #include<stdio.h>
2. #include<string.h>
3. #include<time.h>
4. #define MAX 1000
5.
6. char c[2000]; //全局变量，存储大数运算的结果
7. int temp[MAX];
8. char a[2000];
9. char result[MAX * 2];
10. int binary[MAX];
11. char res[MAX];
12. int A1[MAX];
13. char tempnew[MAX];
14. int temp_h[MAX];
15. char temp_q[MAX];
16.
17. int Judge(char ch[]) //判断字符串 ch 是否全为 0，若全为 1，返回，否则返回 0
18. {
19.     int i, k;
20.     k = strlen(ch);
21.     for (i = 0; i < k; i++)
22.         if (ch[i] != '0')
23.             return 0;
24.     return 1;
25. }
26.
27. int isstr0(char *num)
28. {
29.     int i, len;
30.     len = strlen(num);
31.     for (i = 0; i < len; i++)
32.     {
33.         if (num[i] != '0')
34.             return 1;
35.     }
36.     return 0;
37. }
38.
39. int ToBinary(char from[])
40. {
41.     char big_num[MAX];
42.     char quotient[MAX];
43.
44.     char temp, n;
45.     char remider;
46.     int i, j, k;
47.     long long flag = 1;
48.     memset(big_num, 0, MAX);
49.     strcpy(quotient, from);
50.     i = j = 0;
51.     if (strlen(quotient) >= MAX)
52.     {
53.         printf("Big num string too long. %d bits limited\n", MAX -
1);
54.         exit(0);
```



```

55.     }
56.     k = 0;
57.     do
58.     {
59.         n = 0;
60.         j = 0;
61.         strcpy(big_num, quotient);
62.         for (i = 0; i < strlen(big_num); i++)
63.         {
64.             if (big_num[i] < '0' || big_num[i] > '9')
65.             {
66.                 printf("invalid character in big num string.\n");
67.                 exit(0);
68.             }
69.             if (big_num[i] == '1' && i != strlen(big_num) - 1 && n
70.                 != 10)
71.             {
72.                 n = 10;
73.                 quotient[j] = '0';
74.                 j++;
75.                 continue;
76.             }
77.             else
78.             {
79.                 temp = big_num[i] - '0';
80.                 temp = n + temp;
81.                 quotient[j] = (temp / 2) + '0';
82.                 j++;
83.                 if ((temp & 1) == 1)
84.                     n = 10;
85.                 else n = 0;
86.             }
87.             quotient[j] = '\0';
88.             result[k] = (temp & 1) + '0';
89.             k++;
90.         } while (isstr0(quotient));
91.         result[k] = '\0';
92.         //reverse(result);
93.         //printf("binary: %s\n", result);
94.         int len = strlen(result);
95.
96.         for (i = 0; i < len; i++)
97.             binary[i] = result[i] - '0';
98.         return len;
99.     }
100.
101. int Subtraction(char num1[], char num2[], int sum[])
102. {
103.     int i, j, len, flag;
104.     char *temp;
105.     int n2[MAX] = { 0 };
106.     int len1 = strlen(num1); // 计算数组 num1 的长度, 即大数的位数
107.     int len2 = strlen(num2); // 计算数组 num2 的长度, 即大数的位数
108.
109.     // 在进行减法之前要进行一些预处理
110.     flag = 0; // 为 0 表示结果是正整数, 为 1 表示结果是负整数

```

```

111.     if (len1 < len2) // 如果被减数位数小于减数
112.     {
113.         flag = 1; // 标记结果为负数
114.         // 交换两个数，便于计算
115.         temp = num1;
116.         num1 = num2;
117.         num2 = temp;
118.         len = len1;
119.         len1 = len2;
120.         len2 = len;
121.     }
122.     else if (len1 == len2) // 如果被减数的位数等于减数的位数
123.     {
124.         // 判断哪个数大
125.         for (i = 0; i < len1; i++)
126.         {
127.             if (num1[i] == num2[i])
128.                 continue;
129.             if (num1[i] > num2[i])
130.             {
131.                 flag = 0; // 标记结果为正数
132.                 break;
133.             }
134.             else
135.             {
136.                 flag = 1; // 标记结果为负数
137.                 // 交换两个数，便于计算
138.                 temp = num1;
139.                 num1 = num2;
140.                 num2 = temp;
141.                 break;
142.             }
143.         }
144.     }
145.     len = len1 > len2 ? len1 : len2; // 获取较大的位数
146.     //将 num1 字符数组的数字转换为整型数且逆向保存在整型数组 sum 中，即低
    位在前，高位在后
147.     for (i = len1 - 1, j = 0; i >= 0; i--, j++)
148.         sum[j] = num1[i] - '0';
149.     // 转换第二个数
150.     for (i = len2 - 1, j = 0; i >= 0; i--, j++)
151.         n2[j] = num2[i] - '0';
152.     // 将两个大数相减
153.     for (i = 0; i <= len; i++)
154.     {
155.         sum[i] = sum[i] - n2[i]; // 两个数从低位开始相减
156.         if (sum[i] < 0) // 判断是否有借位
157.         { // 借位
158.             sum[i] += 10;
159.             sum[i + 1]--;
160.         }
161.     }
162.     // 计算结果长度
163.     for (i = len1 - 1; i >= 0 && sum[i] == 0; i--)
164.         ;
165.     len = i + 1;
166.     if (flag == 1)
167.     {

```

```

168.         sum[len] = -1; // 在高位添加一个-1表示负数
169.         len++;
170.     }
171.     return len; // 返回结果的位数
172. }
173.
174. int Multiplication(char num1[], char num2[], int sum[])
175. {
176.     int i, j, len, len1, len2;
177.     int a[MAX + 10] = { 0 };
178.     int b[MAX + 10] = { 0 };
179.     int c[MAX * 2 + 10] = { 0 };
180.
181.     len1 = strlen(num1);
182.     for (j = 0, i = len1 - 1; i >= 0; i--) //把数字字符转换为整型
数
183.         a[j++] = num1[i] - '0';
184.     len2 = strlen(num2);
185.     for (j = 0, i = len2 - 1; i >= 0; i--)
186.         b[j++] = num2[i] - '0';
187.
188.     for (i = 0; i < len2; i++)//用第二个数乘以第一个数,每次一位
189.     {
190.         for (j = 0; j < len1; j++)
191.         {
192.             c[i + j] += b[i] * a[j]; //先乘起来,后面统一进位
193.         }
194.     }
195.
196.     for (i = 0; i < MAX * 2; i++) //循环统一处理进位问题
197.     {
198.         if (c[i] >= 10)
199.         {
200.             c[i + 1] += c[i] / 10;
201.             c[i] %= 10;
202.         }
203.     }
204.
205.     for (i = MAX * 2; c[i] == 0 && i >= 0; i--); //跳过高位的 0
206.     len = i + 1; // 记录结果的长度
207.     for (; i >= 0; i--)
208.         sum[i] = c[i];
209.     return len;
210. }
211.
212. int SubStract(int *p1, int len1, int *p2, int len2)
213. {
214.     int i;
215.     if (len1 < len2)
216.         return -1;
217.     if (len1 == len2)
218.     {
219.         // 判断 p1 > p2
220.         for (i = len1 - 1; i >= 0; i--)
221.         {
222.             if (p1[i] > p2[i]) // 若大,则满足条件,可做减法
223.                 break;
224.             else if (p1[i] < p2[i]) // 否则返回-1
225.                 return -1;

```

```

225.     }
226. }
227. for (i = 0; i <= len1 - 1; i++) // 从低位开始做减法
228. {
229.     p1[i] -= p2[i];           // 相减
230.     if (p1[i] < 0)           // 若是否需要借位
231.     { // 借位
232.         p1[i] += 10;
233.         p1[i + 1]--;
234.     }
235. }
236. for (i = len1 - 1; i >= 0; i--) // 查找结果的最高位
237. {
238.     if (p1[i])               // 最高位第一个不为 0
239.         return (i + 1);      // 得到位数并返回
240. }
241. return 0;                   // 两数相等的时候返回 0
242. }
243.
244. int Division(char num1[], char num2[], char sum[], int r[])
245. {
246.     int i, j;
247.     int len1, len2, len = 0; // 大数位数
248.     int dValue;              // 两大数相差位数
249.     int nTemp;               // Subtract 函数返回值
250.     int num_a[MAX] = { 0 }; // 被除数
251.     int num_b[MAX] = { 0 }; // 除数
252.     int num_c[MAX] = { 0 }; // 商
253.     int temp[MAX * 2 + 10] = { 0 };
254.     int temp1[MAX] = { 0 };
255.     char temp2[MAX];
256.     int temp3[MAX];
257.     int temp4[MAX];
258.
259.     len1 = strlen(num1); // 获得大数的位数
260.     len2 = strlen(num2);
261.
262.     // 将数字字符转换成整型数，且翻转保存在整型数组中
263.     for (j = 0, i = len1 - 1; i >= 0; j++, i--)
264.         num_a[j] = num1[i] - '0';
265.     for (j = 0, i = len2 - 1; i >= 0; j++, i--)
266.         num_b[j] = num2[i] - '0';
267.
268.     if (len1 < len2) // 如果被除数小于除数，直接返回 -1，表示
                       // 结果为 0
269.     {
270.         strcpy(tempnew, num1);
271.         return -1;
272.     }
273.     dValue = len1 - len2; // 相差位数
274.     for (i = len1 - 1; i >= 0; i--) // 将除数扩大，使得除数和被除
                                     // 数位数相等
275.     {
276.         if (i >= dValue)
277.             num_b[i] = num_b[i - dValue];
278.         else // 低位置 0
279.             num_b[i] = 0;
280.     }

```

```

281.     len2 = len1;
282.     for (j = 0; j <= dValue; j++)    //重复调用，同时记录减成功的次
        数，即为商
283.     {
284.         while ((nTemp = SubStract(num_a, len1, num_b + j, len2 -
            j)) >= 0)
285.         {
286.             len1 = nTemp;                //结果长度
287.             num_c[dValue - j]++;        //每成功减一次，将商的相应位
            加 1
288.         }
289.     }
290.     // 计算商的位数，并将商放在 sum 字符数组中
291.     for (i = MAX - 1; num_c[i] == 0 && i >= 0; i--); //跳过高位
        0，获取商的位数
292.     if (i >= 0)
293.         len = i + 1; // 保存位数
294.     for (j = 0; i >= 0; i--, j++)    // 将结果复制到 sum 数组中
295.         sum[j] = num_c[i] + '0';
296.     sum[j] = '\0';    // sum 字符数组结尾置 0
297.
298.     int len_new = Multiplication(num2, sum, temp);
299.     for (i = 0; i < len_new; i++)
300.     {
301.         temp2[i] = (char)('0' + temp[i]);
302.     }
303.     temp2[i] = '\0';
304.
305.     for (i = len_new - 1, j = 0; i >= 0; i--, j++)
306.         temp3[j] = temp2[i] - '0';
307.
308.     for (i = 0; i < len_new; i++)
309.     {
310.         temp2[i] = (char)('0' + temp3[i]);
311.     }
312.     int len3 = Subtraction(num1, temp2, temp4);
313.
314.     i = len3 - 1;
315.
316.     for (j = 0; i >= 0; i--, j++)    // 将结果复制到 sum 数组中
317.         temp_h[j] = temp4[i];
318.
319.     memset(tempnew, 0, sizeof(tempnew));
320.     for (i = 0; i < j; i++)
321.     {
322.         tempnew[i] = (char)('0' + temp_h[i]);
323.     }
324.
325.     return len;    // 返回商的位数
326. }
327.
328. void ModPow(int a, char k[], char m[])
329. {
330.     int i, j, q;
331.     int len = ToBinary(k);
332.     if (binary[0] == 0)
333.         res[0] = '1';
334.     char A[MAX] = "5";

```

```

335.
336.
337.     if (binary[0] == 1)
338.         res[0] = a;
339.     for (i = 1; i < len; i++)
340.     {
341.         int len3=Multiplication(A, A, temp);
342.         char temp4[MAX];
343.         for (j = 0, q = len3 - 1; q >= 0; j++, q--)
344.             temp4[j] = (char)('0' + temp[q]);
345.         memset(A, 0, sizeof(A));
346.
347.         strncpy(A, temp4, len3);
348.
349.         Division(A, m, temp_q, A1);
350.         int len2 = strlen(tempnew);
351.         strcpy(A, tempnew);
352.         if (binary[i] == 1)
353.         {
354.             int len3 = Multiplication(A, res, temp);
355.             char temp4[MAX];
356.             for (j = 0, q = len3 - 1; q >= 0; j++, q--)
357.                 temp4[j] = (char)('0' + temp[q]);
358.             memset(res, 0, sizeof(res));
359.             strncpy(res, temp4, len3);
360.             Division(res, m, temp_q, A1);
361.             int len2 = strlen(tempnew);
362.             strcpy(res, tempnew);
363.         }
364.     }
365. }
366.
367. int main()
368. {
369.     clock_t start, finish;
370.     double duration;
371.     start = clock();
372.
373.     int a = 5;
374.     char k[] = "596";
375.     char m[] = "6729475";
376.     ModPow(a, k, m);
377.     printf("%s\n\n", res);
378.     finish = clock();
379.     duration = (double)(finish - start) / CLOCKS_PER_SEC;
380.     printf("duration: %f seconds\n\n", duration);
381.     system("pause");
382.     return 0;
383. }

```