OOP 大作业文档——大整数 GCD

2019010175 孔瑞阳 土木 92
13063995383 kry19@mails.tsinghua.edu.cn

一、摘要

本项目实现了使用大整数进行最大公约数的计算，介绍了两种计算最大公约数的算法（辗转相除法和更相减损法），给出了代码的验证思路和验证过程、结论，并简要介绍了大整数 gcd 的应用和相应的拓展。

附录(1)(2)是平时作业 10 的文档内容，记录了大整数计算的模型和验证。

二、项目信息

1、学习/实现内容

求解两个大整数的最大公约数。

用更相减损法、辗转相除法两种算法实现了最大公约数的计算，并比较了两者的特点。

2、软件构件介绍

文件	功能介绍
random.h/cpp	随机数类
C_Integer.h/cpp	实现的大整数类
C_IntegerTest.h/cpp	大整数类的测试
CP_GCD.h/cpp	实现了大整数类的 gcd 计算
CP_GCD_Test.h/cpp	大整数类 gcd 计算的测试
CP_GCD_Main.cpp	主程序

3、结论

gcd 在数论、密码学中有着广泛的应用，并且这些应用需要用到大整数的计算。

计算 gcd 的两种主要算法为更相减损法和辗转相除法。

对于用大整数实现的 gcd 算法，更相减损法的时间、编程、思维复杂度均优于辗转相除法。对于时间复杂度，更相减损法为 O(n^2)，辗转相除法为 O(n^2 log n)，其中 n 为整数位数。并且，辗转相除法的时间常数大概是更相减损法的 10-40 倍。

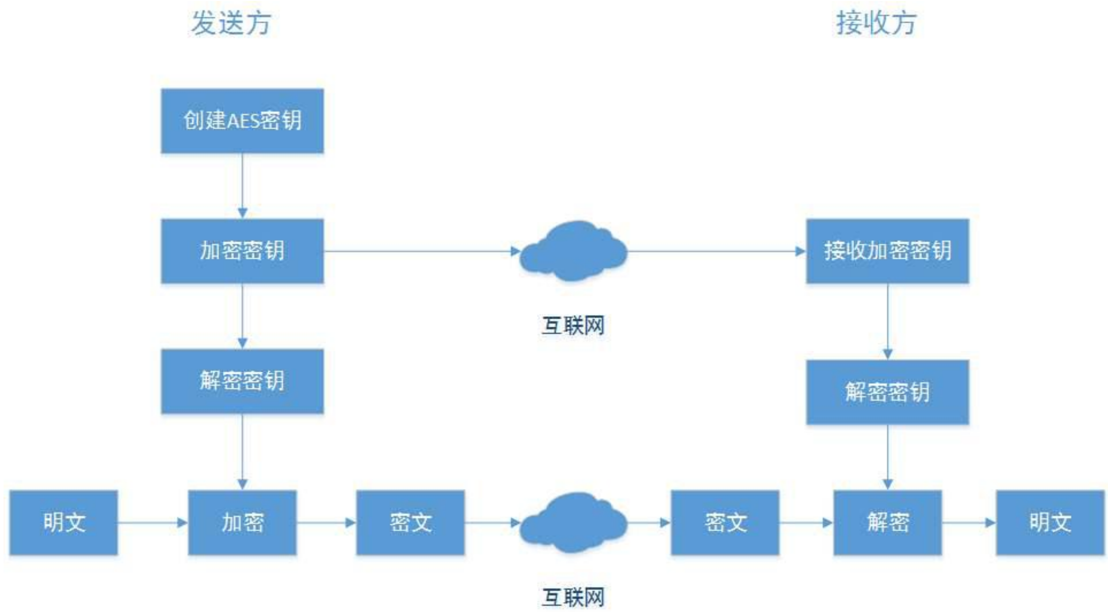
但是辗转相除法有更高的拓展性，拓展为拓展欧几里得算法可以解决不定方程问题。

三、可能的应用

最大公约数的求解属于初等数论内容，初等数论体系在现代密码学中有着广泛的应用。下面，以 RSA 公匙算法的一部分过程为例，简要介绍最大公约数在密码学中的应用。

假设 Tanaka 想通过一个通信不安全的传递过程接受 Krydom 的一条消息，
他可以用下面的方式产生公钥和私钥。

1. 选择两个不相等的大质数 p 和 q，计算 N = pq.
2. 由欧拉函数，求得 r=φ(N)=φ(p)φ(q)=(p-1)(q-1)。
3. 选择一个小于 r 的整数 e,使 e 与 r 互质。**并求得 e 关于模 r 的逆元，命名为 d。**
4. 将 p 和 q 的记录销毁。



可以发现，在第 3 步中，要求 e 关于模 r 的逆元，并且观察到，由于 r=(p-1)(q-1)，一定不是质数，所以不能直接用费马小定理+快速幂求解逆元。由裴蜀定理，只要 gcd(e,r)=1，逆元就一定存在，需要使用拓展欧几里得算法解决这一问题。

裴蜀定理：若 a,b 是整数,且 gcd(a,b)=d，一定存在整数 x,y，使 ax+by=d 成立。

那么 gcd(e,r)=1 时，考虑求解方程 ex+ry=1，求得的 x 就是 e 的逆元。所谓拓展欧几里得算法，就是在求解 gcd 的过程中同时迭代求解这一个方程，在之后的内容中会介绍到。

为了保证安全性,所以 N 的大小至少要 1024 位。而一般的整数,int 是 32 位,long long 是 64 位，__int128 是 128 位，当我们要进行更大位数的整数的计算时，就需要使用到大整数进行 gcd 的计算，也就是本项目的内容。

四、GCD 计算的模型

1、欧几里得算法（辗转相除法）

记(a,b)为 a,b 的最大公约数，大整数的位数为 n，大整数为 N。

那么可以显然地得到若干个性质。

(a,a) = (a,0) = a

(a,b) = (a,a+b) = (a,ka+b)

(a,b) = (a, b mod a) （结论 (2) 的推论）

多次运用上述性质，即可求得两个数的最大公约数。

引理：若 a <= b，则 b mod a < b/2。

引理的证明：

若 a <= b / 2，则 b mod a < a <= b / 2，结论成立。

若 b / 2 < a <= b，则 b mod a = b - a < b / 2，结论成立。

也就是说，我们每进行一次递归，a 和 b 中的某一个数就会减少为原来的 1/2，也就是说，总共只会进行 log 次运算。

那么总的时间复杂度就是 O(n^2 log n)。

（O(log N) = O(n)，进行一次大整数除以大整数的时间复杂度为 O(n log n)）。

2、更相减损法

记(a,b)为 a,b 的最大公约数，大整数的位数为 n，大整数为 N。

那么可以显然地得到若干个性质。

(a,b) = 2 * (a/2,n/2)	(a,b 为偶数)
(a,b) = (a,b/2)	(a 为奇数，b 为偶数)
(a,b) = (a,b-a)	(a,b 为奇数，b>=a)

多次运用上述性质，即可求得两个数的最大公约数。

可以发现，每使用一次性质(1)(2)，就会至少有一个数/2。而对于性质(3)，由于 a 和 b 都是奇数，所以 b-a 是偶数，下一次运用的也一定是性质(2)。所以在经过 2 次计算后至少有一个数会减半，总共也只会进行 log 次运算。

那么总的时间复杂度是 O(n^2)。

（O(log N) = O(n)，进行一次大整数除以小整数的时间复杂度为 O(n)）。

五、GCD 计算的验证

1、手动测试(gcdMunualTest)

1、基本思路

输入两个大整数，验证计算的结果是否正确。

2、核心代码

(1) 验证过程

```
X1.mb_input("请输入第一个大整数 X1 的值");
X2.mb_input("请输入第二个大整数 X2 的值");
X3 = gcd_half(X1, X2);                          // 使用更相减损法计算
X3.mb_show("更相减损法的结果: gcd(X1,X2)=");
X4 = gcd_division(X1, X2);                      // 使用辗转相除法计算
X4.mb_show("辗转相除法的结果: gcd(X1,X2)=");
```

(2) 更相减损法

```
while (x.parity() == 0 && y.parity() == 0) x = x / 2, y = y / 2, ans = ans * 2;
// 性质 2(1)
while (x.parity() == 0) x = x / 2;     // 性质 2(2)
while (y.parity() == 0) y = y / 2;     // 性质 2(2)
if (x < y) y = y - x; else x = x - y; // 性质 2(3)
```

(3) 辗转相除法

```
if (x.mb_checkZero()) return y; // 边界条件,性质 1(1)
while (!y.mb_checkZero())       // 辗转相除法的迭代过程
{
    C_Integer tmp = x % y;          // 性质 1(3)
    x = y;
    y = tmp;
}
return x;
```

注：如果采用递归法实现上述算法，当大整数位数非常大的时候，会导致超出内存。所以只能用**迭代法**实现。

3、等价类划分

更加完备的测试在自动测试中进行，这里只列举简单、典型的案例进行验证。

数的等价类 ： 1、正数 2、负数 3、0

结果的等价类： 1、不为 1 2、为 1 3、GCD(0,0)

4、验证

结果为 1 的情况：

GCD	-91	0	89
-45	1	/	1
0	/	/	/
101	1	/	1

GCD	0	1
0	/	1
-1	1	1

结果不为 1 的情况：

GCD	-45	0	96
-18	9	18	6
0	45	/	96
72	9	72	24

GCD(0,0)：

关于 0 和 0 的最大公约数，主要要 3 种说法：

1、0 和任何数的最大公约数都是那个数本身，所以 0 和 0 的最大公约数是 0。

2、任何数都整除 0，所以 0 和 0 的最大公约数是无穷大。

3、0 和 0 的最大公约数不存在。

在本项目中，规定 GCD(0,0)=0。

验证：gcd(0,0)=0 成立。

2、自动测试(gcdAutoTest)

一开始的验证方案为：

1、生成随机整数 x1，x2。

2、将大整数 X1 赋值为 x1，大整数 X2 赋值为 x2。

3、用更相减损法计算 X1 和 X2 的最大公约数 X3。

4、用辗转相除法计算 X1 和 X2 的最大公约数 X4。

5、用普通整数的 gcd 计算 x1 和 x2 的最大公约数，赋值给 X5。

6、计算 X3、X4、X5 是否相等。

但是在实践过程中，发现这样随机出的最大公约数非常小，可能测试会不完备。

所以将第 1 步拓展为两个步骤：

1(1)、生成随机整数 x1，x2，x3。

1(2)、将 x1、x2 分别乘 x3。

这样，相当于将原来的最大公约数增加了 x3 倍，使得其具有一定的数据规模。

在一集番剧 23 分钟的时间内，自动测试没有发现错误。据计算，在 23 分钟内，大概可以进行 10 亿次量级的计算，基本可以验证的正确性。

六、分析和拓展

1、两种算法的比较分析

在一开始的分析中，已经得出：
辗转相除法的时间复杂度是 O(n^2 log n)，
更相减损法的时间复杂度是 O(n^2)。

并且，更相减损法只涉及到普通的大整数与小整数的乘除运算，实现较为简单。辗转相除法达到最优时间复杂度需要使用快速傅里叶变换进行优化，思维和编程复杂度较高，并且常数也较高。可以使用快速数论变换来代替快速傅里叶变换，由于只涉及整数的运算，所以常数会比较低。

下表是多次计算时间取平均数的结果：

大整数的位数 n	更相减损法所用时间	辗转相除法所用时间
100	44ms	1113ms
200	148ms	4775ms
500	0.6s	24.2s
1000	2.2s	96s

可以发现更相减损法所用的时间比辗转相除法少很多。
同时两者经观察基本是处于同一多项式复杂度的。
两种算法所用时间差基本不随 n 的大小改变，保持在 30-50 倍左右，是常数倍的差距。

2、GCD 的拓展

在 RSA 算法中要求出 e 对于模 r 的逆元，其中(e,r)=1。

拓展欧几里得算法：
考虑如何求得 **ax + by = d** 的一个解。这里 d = (a, b)

考虑使用欧几里德算法的思想，令 a = bq + r，其中 r = a mod b;
递归求出 bx + ry = d 的一个解。

设求出 bx + ry = d 的一个解为 x = x0, y = y0,
考虑如何把它变形成 ax + by = d 的解。
将 a = bq + r 代入 ax + by = d，化简得 b(xq + y) + rx = d。
我们令 xq + y = x0, x = y0，则上式成立。
故 x = y0, y = x0-y0q 为 ax + by = d 的解。
边界情况：b = 0 时，令 x = 1, y = 0。

七、附录(1)大整数的模型

1、大整数的表示

```
protected:
    IntegerStatus m_status;          表示数的类型
    vector<unsigned char> m_data;    表示数的绝对值
```

其中，IntegerStatus 定义如下：

```
    INTEGER_INVALID = -3,     //非数
    INTEGER_NEG_INF = -2,     //负无穷大
    INTEGER_NEG_VALUE = -1,   //常规负数
    INTEGER_ZERO = 0,         //0
    INTEGER_POS_VALUE = 1,    //常规正数
    INTEGER_POS_INF = 2       //正无穷大
```

2、大整数的加减法

如果两个数不同号，则加法变成减法，减法变成加法，所以只考虑同号的情况。

对于加法：按位相加，如果某一位大于 9 则进位。

对于减法：先保证绝对值大的数减绝对值小的数，再按位相减，如果某一位小于 0 则退位。计算完成后去掉最前面的若干个 0，如果变成了 0 则更改数的类型。

3、大整数的乘法

设第一个大整数的位数为 n，第二个大整数的位数为 m。

如果朴素地枚举两个大整数的每一位进行乘法运算再相加，则时间复杂度为 O(nm)，当 n.m 达到 10^5 的级别时，乘法就会变得非常慢。

所以采用快速傅里叶变换(FFT)，用复数单位元进行乘法优化，则时间复杂度为 O((n+m)log(n+m))，时间效率更高。

某个介绍 FFT 进行乘法优化的博客：

https://www.cnblogs.com/zwfymqz/p/8244902.html

4、大整数的除法

如果可以整除的话用快速数论变换(NTT)进行多项式逆元，也可以在 O(nlogn)的时间复杂度内完成(假设 n、m 同阶)。但是不能整除的时候处理麻烦。

所以使用二分法，先确定一个答案可能的范围[l,r]。

每次二分一个可能的结果，用上述的大整数的乘法的操作进行判断，乘之后的结果是否大于被除数。因为可以按照位数估算 l 和 r 的位数，所以可以保证 l<=r<=99l，二分的次数为常数。所以时间复杂度依然是 O(nlogn)。

八、附录(2)大整数的计算验证

1、手动测试(integerTest)

加法

等价类	选取案例	结果
正数+正数	1919810+114514	2034324
正数+0	12345678910111213+0	12345678910111213
负数+负数	(-575687684)+(-687685451)	-1263373135
负数+0	-3141592653589+0	-3141592653589
正数+负数(正绝对值大)	8101919+(-514)	8101405
正数+负数(负绝对值大)	114+(-8101919)	-8101805

减法

等价类	选取案例	结果
正数-负数	114-(-8101919)	8102033
正数-0	12345678910111213-0	12345678910111213
0-正数	0-12345678910111213	-12345678910111213
正数-正数（左绝对值大）	1919810-114514	1805296
正数-正数（右绝对值大）	114514-1919810	-1805296
负数-正数	-8101919-114	-8102033
负数-0	-3141592653589-0	-3141592653589
0-负数	0-(-3141592653589)	3141592653589
负数-负数（左绝对值大）	(-687685451)-(-575687684)	-111997767
负数-负数（右绝对值大）	(-575687684)-(-687685451)	111997767

乘法

等价类	选取案例	结果
正数*正数	1919810*114514	219845122340
正数*0	12345678910111213*0	0
负数*负数	(-575687684)*(-687685451)	395892044606685484
负数*0	-3141592653589*0	0
正数*负数	114*(-8101919)	-923618766

除法

等价类	选取案例	结果
正数/正数	1919810/114514	16
0/正数	0/12345678910111213	0
负数/负数	(-687685451)/(-57568768)	11
0/负数	0/(-3141592653589)	0
正数/负数	8101919/(-514)	-15762
负数/正数	-8101919/114	-71069

2、自动测试(integerAutoTest)

实现了赋值构造函数 C_Integer(long long x);

采用**先将整数赋值到大整数类、再大整数类进行运算**与**整数先运算、再赋值到大整数类**进行对拍。

每次随机生成两个整数 x1,x2（正/负/0）进行对拍。

对于它们之间的所有 10 种运算(+-*//小整数各两种)全部测试一遍。如果出现错误则输出错误的数据，否则一直进行循环。（当先 x1 或 x2 随机出 0 时，不进行/x1 或/x2 的测试。）

经过 10 分钟的对拍，没有出现错误。

根据估算，10 分钟大致可以进行千亿(10^10)次计算，基本可以验证程序的正确性。