Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

**Арифметические операции над целыми числами и многочленами**

ОТЧЕТ ПО ДИСЦИПЛИНЕ

«ТЕОРЕТИКО-ЧИСЛОВЫЕ МЕТОДЫ В КРИПТОГРАФИИ»

Лабораторная работа №1

студентки 5 курса 531 группы

специальности 10.05.01 «Компьютерная безопасность»

факультета компьютерных наук и информационных технологий

Громовой Наталии Викторовны

|  |  |  |
| --- | --- | --- |
| Преподаватель  профессор | \_\_\_\_\_\_\_\_\_\_ | В.А. Молчанов |
|  | подпись, дата |  |

Саратов 2018

**Цель работы:** используя алгоритм Евклида создать программу, которая для чисел *a* и *b* определяет наибольший общий делитель, изучить быстродействие алгоритмов.

**Теоретический материал**

Натуральное число p, больше единицы называется простым, если оно делится нацело только на единицу и на себя.

Теорема о делении. Для любого целых чисел *a* и *b*, *b* > 0, существуют, и притом единственные, целые числа *q* и *r*, такие, что *a* = *bq* + *r*, 0 ≤ *r* < *b*.

Теорема. Для любого составного числа наименьший отличный от единицы положительный делитель является простым числом.

Определение. Наибольшим общим делителем (НОД) целых чисел *a1, a2, …, ak* называется такой положительный общий делитель этих чисел, который делится на любой другой делитель этих чисел. Если *d* – наибольший общий делитель для чисел *a* и *b*, то для него вводится обозначение (*a, b*) = *d*.

*Алгоритм Евклида*

Для двух целых чисел *a* и *b* существует сравнительно быстрый метод вычисления наибольший общий делитель. Упомянутый метод вычисления наибольший общий делитель называется алгоритмом Евклида. Приведем схему работы этого алгоритма.

1. Делим число *a* на число *b*, получаем *a* = *bq0* + *r*1;

2. Делим число *b* на число *r*1, имеем *b* = *r1q1* + *r2*;

3. Делим число *r1* на число *r2*, запишем *r*1 = *r2q2* + *r3*;

4. Делим число *r2* на число *r3*, получаем

*r2* = *r3q3* + *r4*;

…;

*rt-1*= *rtqt* + *rt+1*.

Если остаток от деления *rt+1*= 0, то в этом случае наибольший общий делитель равен числу *r*t и алгоритм вычисления наибольший общий делитель завершается.

Сложность алгоритма Евклида. *Теорема* (Ламе). Для любого натурального числа *N*>0 число делений в алгоритме Евклида для нахождения наибольшего общего делителя чисел *А* и *В*, 0<*B*<*A***≤***N* не превосходит 1+ logR*N*.

Доказательство. Докажем, что *fi***≤***rk+1-I* при *i*=1,…,*k+2*. При *i*=1 верно. При i+1, в силу предположения индукции, имеем



Поэтому, *А***≥***r-1***≥***fk+2***≥***Rk*, откуда получается искомая оценка числа делений *k*+1**≤** 1+ logR*N*.

Непосредственно из этой теоремы можно получить оценку сложности алгоритма Евклида для двух n-разрядных чисел:

****

*Расширенный алгоритм Евклида*

Пусть алгоритм Евклида на каждом шаге, кроме частного *di* и остатка *ri,* вычисляет еще два значения *ui*, *vi* по правилу

*u-1*=1,*u*0=0;

*v-1*=0,*v0*=1;

*ui*=*ui-2-diui-1,*1<=*i*<=*k;*

*vi*=*vi-2-divi-*1, 1<=*i*<=*k.*

Такой алгоритм будем называть расширенным алгоритмом Евклида. В расширенном алгоритме Евклида для всех *i* ∈ {–1, 0, ..., *k*} выполняется равенство *uix*1 + *vix*2 = *ri.* Значение расширенного алгоритма Евклида состоит в том, что он дает линейное разложение наибольшего общего делителя *ukx1* + *vkx2* = *r*k = (*x1, x2*).

1. Сложность этого алгоритма О(log2n).

*Бинарный алгоритм Евклида*

В эффективности данный алгоритм превосходит метод Евклида, что связано с использованием сдвигов, то есть операций деления на степень двойки:

1. Инициализируем переменную *k* значением 1. Ее задача – подсчитывать «несоразмерность», полученную в результате деления. В то время как *A* и  *B* сокращаются вдвое, она будет увеличиваться вдвое;
2. Пока *A* и *B* одновременно не равны нулю, выполняем
   * если *A* и *B* – четные числа, то делим надвое каждое из них: *A*←*A*/2, *B*←*B*/2, а *k* увеличивать вдвое: *k*←*k*\*2, до тех пор, пока хотя бы одно из чисел *A* или *B* не станет нечетным;
   * если *A* – четное, а *B* – нечетное, то делим *A*, пока возможно деление без остатка;
   * если *B* – четное, а *A* – нечетное, то делим *B*, пока возможно деление без остатка;
   * если *A*≥*B*, то заменим *A* разностью *A* и *B*, иначе *B* заменим разностью *B* и *A*.
3. После выхода из 2-ого пункта, остается вернуть в качестве результата произведение *B* и *k*: НОД(*A*, *B*)=*B*\**k.*
4. Сложность этого алгоритма О(log2n).

*Китайская теорема об остатках*

Пусть *k* натуральное число и *m1, . . ., mk*целые, взаимно простые числа, произведение которых равно. Тогда любого набора целых чисел *a1, . . . , ak* решение системы сравнений

единственно по модулю *M* и удовлетворяет сравнению

где *bi*= и *ci* ≡ ).

Алгоритм решения:

1. Вычисляем {\displaystyle M={\displaystyle \prod \_{i=1}^{n}a\_{i}}}.

2. Для всех {\displaystyle i\in \{1,2,\dots ,n\}}находим  {\displaystyle M\_{i}={\frac {M}{a\_{i}}}}.

3. Находим {\displaystyle M\_{i}^{-1}={\frac {1}{M\_{i}}}{\bmod {a\_{i}}}}

4. Вычисляем искомое значение по формуле .

*Алгоритм Гарнера*

Из китайской теоремы об остатках следует, что можно заменять операции над числами операциями над кортежами. Это может найти широкое применение на практике (помимо непосредственного применения для восстановления числа по его остаткам по различным модулям), поскольку мы таким образом можем заменять операции в длинной арифметике операциями с массивом "коротких" чисел:

1. Определить *i* = 2, *b* = 1, *s* = *a1* (mod *m1*).

2. Пока *i* <= *k* выполнять

2.1. определить *b* *= bmi−1* и *d ≡ b −1* (mod *mi*),

2.2. вычислить *x ≡ d(ai − s)* (mod *mi*),

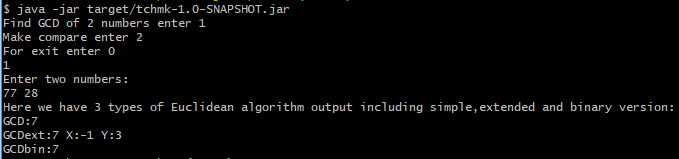
2.3. вычислить *s = s + xb* и положить *i = i* + 1.

3. Вернуть значение s.

**Практическая часть**

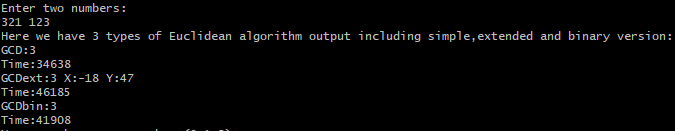
Все вышеописанные алгоритмы были написаны на языке Java. Листинги кода предоставлены в приложении.

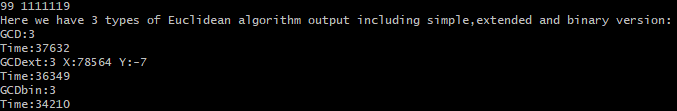
Вывод алгоритма Евклида (простой, расширенный и бинарный):



Тестирование по времени (секунды):

-для двух чисел



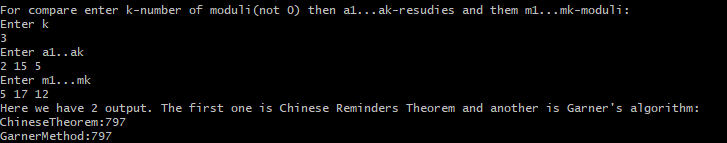


|  |  |  |
| --- | --- | --- |
| **Алгоритм Евклида** | **Небольшие числа**  **344, 989**  **(c)** | **Большие числа**  **16727277, 1726**  **(c)** |
| Обычный | 4.875E-5 | 1.03916E-4 |
| Расширенный | 6.4146E-5 | 1.05198E-4 |
| Бинарный | 4.3619E-5 | 1.37272E-4 |

* для диапазонов (миллион элементов)

|  |  |  |
| --- | --- | --- |
| **Алгоритм Евклида** | **Диапазон**  **1:1000**  **(c)** | **Диапазон**  **20000:21000**  **(c)** |
| Обычный | 1.934527100433767E-7 | 1.8201433366633368E-7 |
| Расширенный | 1.597978004671338E-7 | 2.664804555444555E-7 |
| Бинарный | 1.1590655702369036E-6 | 2.975377952047952E-7 |

Вывод Китайской теоремы об остатках и алгоритма Гарнера



Тестирование по времени (секунды):

|  |  |  |
| --- | --- | --- |
| **Алгоритм** | **Модуль до 100**  **(a1, a3, a3)=78,17,11**  **(m1, m3, m3)=99,23,19**  **(c)** | **Модуль до 1000**  **(a1, a3, a3)=** **767,918,19**  **(m1, m3, m3)=** **892,997,717**  **(c)** |
| Китайская теорема об остатках | 0.003434773 | 0.003764053 |
| Гарнера | 1.4668E-4 | 1.501E-4 |

**Приложение.**

Листинг 1

алгоритм Евклида(простой,бинарный,расширенный)

**package** first;  
  
**class** Euclid {  
 **static** String EuclidGCD(**int** a, **int** b) {  
 **if** (b == 0)  
 **return "GCD:"**+a;  
 **else return** *EuclidGCD*(b, a % b);  
 }  
  
 **static** String EulcidEXT(**int** a, **int** b) {  
 **int** x1 = 1;  
 **int** y1 = 0;  
 **int** x2 = 0;  
 **int** y2 = 1;  
 **return** *EuclidEXT*(a, b, x1, y1, x2, y2);  
 }  
  
 **private static** String EuclidEXT(**int** a, **int** b, **int** x1, **int** y1, **int** x2, **int** y2) {  
 **while** (b != 0) {  
 **int** q = a / b;  
 **int** r = a % b;  
 a = b;  
 b = r;  
 **int** t = x2;  
 x2 = x1 - q \* x2;  
 x1 = t;  
 t = y2;  
 y2 = y1 - q \* y2;  
 y1 = t;  
 }  
 **return "GCDext:"** + a + **" X:"** + x1 + **" Y:"** + y1;  
 }  
  
 **static** String EuclidBIN(**int** a, **int** b) {  
 **int** k = 1;  
 **while** (a != 0 && b != 0) {  
 **while** (a % 2 == 0 && b % 2 == 0) {  
 a /= 2;  
 b /= 2;  
 k \*= 2;  
 }  
 **while** (a % 2 == 0)  
 a /= 2;  
 **while** (b % 2 == 0)  
 b /= 2;  
 **if** (a >= b) a -= b;  
 **else** b -= a;  
 }  
 **return "GCDbin:"**+b \* k;  
 }  
}

Листинг 2

Китайская теорема об остатках, алгоритм Гарнера

**package** first;  
  
  
  
**import** java.math.BigInteger;  
  
**public class** Comparisons {  
 **static** String ChineseTheorem(**int** k, **int**[] a, **int**[] mi) {  
 Long M = 1L;  
 Long[] Mi = **new** Long[mi.**length**];  
 Long[] Mi\_revert = **new** Long[mi.**length**];  
 **for** (**int** i = 0; i < k; i++) {  
 M \*= mi[i];  
 }  
 **for** (**int** i = 0; i < k; i++) {  
 Mi[i] = M / mi[i];  
 Mi\_revert[i] = BigInteger.*valueOf*(Mi[i]).modInverse(BigInteger.*valueOf*(mi[i])).longValue();  
 }  
 Long x = 0L;  
 **for** (**int** i = 0; i < k; i++) {  
 x += BigInteger.*valueOf*(a[i] \* Mi[i] \* Mi\_revert[i]).mod(BigInteger.*valueOf*(M)).longValue();  
 }  
 **return "ChineseTheorem:"**+String.*valueOf*(BigInteger.*valueOf*(x).mod(BigInteger.*valueOf*(M)).intValue());  
 }  
  
 **static** String GarnerMethod(**int** k, **int**[] a, **int**[] mi) {  
 **int**[] mi\_revert = **new int**[mi.**length**];  
 **int**[] mu = **new int**[mi.**length**];  
 **int** M = 0;  
 **for** (**int** i = 1; i < k; i++) {  
 M = mi[0];  
 **for** (**int** j = 1; j < i; j++) {  
 M \*= mi[j];  
 }  
 mu[i] = M;  
 mi\_revert[i] = BigInteger.*valueOf*(mu[i]).modInverse(BigInteger.*valueOf*(mi[i])).intValue();  
 }  
 M = mu[k - 1] \* mi[k - 1];  
 **int** temp;  
 **int** v = a[0];  
 **for** (**int** i = 1; i < k; i++) {  
 temp = BigInteger.*valueOf*((a[i] - v) \* mi\_revert[i]).mod(BigInteger.*valueOf*(mi[i])).intValue();  
 v += temp \* mu[i];  
 }  
 v = BigInteger.*valueOf*(v).mod(BigInteger.*valueOf*(M)).intValue();  
 **return "GarnerMethod:"**+String.*valueOf*(v);  
 }  
  
  
}

Листинг 3

Main класс

|  |
| --- |
| package first; |
|  |  |
|  |  |
|  | import java.util.Scanner; |
|  |  |
|  | public class Main { |
|  | public static void main(String[] args) { |
|  | Scanner scanner =new Scanner(System.in); |
|  | System.out.println("Find GCD of 2 numbers enter 1"); |
|  | System.out.println("Make compare enter 2"); |
|  | System.out.println("For exit enter 0"); |
|  | while (true) { |
|  | int init =scanner.nextInt(); |
|  | if (init==1) { |
|  | System.out.println("Enter two numbers:"); |
|  | int a = scanner.nextInt(); |
|  | int b = scanner.nextInt(); |
|  | System.out.println("Here we have 3 types of Euclidean algorithm output including simple,extended and binary version: "); |
|  | System.out.println(Euclid.EuclidGCD(a, b)); |
|  | System.out.println(Euclid.EulcidEXT(a, b)); |
|  | System.out.println(Euclid.EuclidBIN(a, b)); |
|  | } else if (init ==2) { |
|  | System.out.println("For compare enter k-number of moduli(not 0) then a1...ak-resudies and them m1...mk-moduli:"); |
|  | System.out.println("Enter k"); |
|  | int k = scanner.nextInt(); |
|  | int[] a = new int[k]; |
|  | int[] m = new int[k]; |
|  | System.out.println("Enter a1..ak"); |
|  | for (int i = 0; i < k; i++) { |
|  | a[i] = scanner.nextInt(); |
|  | } |
|  | System.out.println("Enter m1...mk"); |
|  | for (int i = 0; i < k; i++) { |
|  | m[i] = scanner.nextInt(); |
|  | } |
|  | System.out.println("Here we have 2 output. The first one is Chinese Reminders Theorem and another is Garner's algorithm:"); |
|  | System.out.println(Comparisons.ChineseTheorem(k, a, m)); |
|  | System.out.println(Comparisons.GarnerMethod(k, a, m)); |
|  | } else if (init ==0) |
|  | return; |
|  | else System.out.println("Error.Try another command:)"); |
|  | System.out.println("You can choose new number (0,1,2):"); |
|  | } |
|  | } |
|  | } |

 {\displaystyle x=\sum \_{i=1}^{n}r\_{i}M\_{i}M\_{i}^{-1}\mod M}.