Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

Практическое задание 6

По дисциплине «Основы информационной безопасности»

На тему «Теория чисел»

Выполнила:

Студентка 2 курса 2 группы

Глухова Д.В.

Вариант 6

Преподаватель: ст. пр. Ржеутская Н. В.

2023, Минск

|  |
| --- |
| **Практическое задание № 6** |
| **Тема «Теория чисел»** |
| Цель**:**  получение основных сведений из курса теории чисел |
| **Теоретические сведения** |

Ниже рассматриваются: *N* – множество натуральных чисел, *Z* – множество рациональных чисел. Множество целых чисел *Z* – счетное, состоит из элементов 0; ±1; ±2; …; ± *n*,…. На нем определены две алгебраические операции – сложение и умножение. Эти операции обладают следующими свойствами (для любых ):

1. ассоциативность: ; ;

2. коммутативность: ; ;

3. существует нейтральный элемент – 0 и 1 соответственно:



4.  – закон дистрибутивности;

5. для каждого целого  существует единственное противоположное, то есть такое целое *b*, что *a* + *b* = *b* + *a* = 0.

В этом равенстве  называют остатком, а  – частным (неполным частным – при ) от деления *a*  на  При *r* = 0 величины *b* и *q* называют делителями или множителями числа *а*. Читатель со школьной скамьи умеет находить частное и остаток методом деления уголком.

*Следствие.* Пусть  – натуральное число,  Для всякого целого числа *a*  и максимального целого  с условием  существуют единственные целые  такие, что 

Такое равенство записывают сокращённо  или  (если *b* известно по контексту) и называют записью числа *a* в *b* – ичной позиционной системе счисления или системе счисления по основанию *b*. Нам кажется естественной привычная десятичная позиционная система записи целых чисел . В различных ситуациях более удобными оказываются другие основания. К примеру, во всех компьютерах на микроуровне вычисления проводятся в двоичной системе счисления. Для перехода к ней с десятичной применяют промежуточную – 16 - ричную систему счисления.

*Лемма 2.1.* Если в равенстве  все слагаемые – целые числа и все, кроме может быть одного, делятся на целое , то и это исключенное слагаемое делится на .

**Определение 2.1*.***Если целые числа  делятся на целое , то *d*  называют их *общим делителем*.

В дальнейшем речь идет только о положительных целых делителях.

**Определение 2.2.** Максимальный из общих делителей целых чисел  называется их *наибольшим общим делителем* и обозначается через НОД ().

*Теорема 2.2.* Если *,* то НОД *(a, b)*=НОД *(b, c).*

*Теорема 2.3.* Наибольший общий делитель целых чисел  *a* и *b*   равен последнему отличному от нуля остатку цепочки равенств:

*;*

*;*

*…………………*

**

**

то есть  *=* НОД *.*

Теорема 2.3 формулирует алгоритм Евклида нахождения наибольшего общего делителя целых чисел. Его вариантом является следующий – второй способ вычисления наибольшего общего делителя по алгоритму Евклида – вычисляем последовательно разности  до получения последней ненулевой разности, которая и совпадает с НОД *(a, b).*

*Теорема 2.4.* Если *d* = НОД *(a, b)*, то существуют такие целые *u*  и *v*, что выполняется следующее соотношение (Безу): *d = au+ bv.*

Такой способ получения соотношения Безу для конкретных целых чисел называется расширенным алгоритмом Евклида. Он состоит из двух этапов: собственно алгоритма Евклида - прогонки вниз и прогонки вверх – последовательного выражения остатков в каждом из шагов предыдущего этапа (с соответствующим приведением подобных на каждом шаге).

**Определение 2.3.** Натуральное число ** называется *простым*, если оно делится только на1 и на себя.

*Теорема 2.5.* Всякое натуральное число ** либо является простым числом, либо имеет простой делитель.

Заметим, что из соотношения  натуральных чисел, больших единицы, следует, что, либо *p,* либо *q* принадлежит отрезку . Легко видеть, что наименьший натуральный делитель ** натурального числа ** является простым числом. Исторически первый метод проверки натурального числа ** на простоту заключается в делении его на простые числа, не превосходящие , носит название “решета Эратосфена”. К настоящему времени разработан достаточно большой цикл алгоритмов проверки числа на простоту.

*Теорема 2.6 (Евклид).* Простых чисел бесконечно много.

Значение простых чисел в том, что они по теореме 2.5 являются составными кирпичиками всех натуральных чисел.

**Определение 2.4.** Целые числа *a*  и  *b* называются *взаимно простыми,* еслиНОД .

*Теорема 2.7* (*Критерий взаимной простоты целых чисел*). Целые числа  *a* и *b* взаимно просты тогда и только тогда, когда существуют такие целые u и v, что выполняется равенство .

**Следствие.** НОД** тогда и только тогда, когдаНОД иНОД .

Важным в теории чисел и ее приложениях является следующее свойство взаимно простых целых чисел.

*Лемма 2.2.* Пусть произведение целых чисел *ab* делится на целое число *с* и НОД . Тогда *b*  делится на  *с*.

*Теорема 2.8**(Основная теорема арифметики)*. Всякое целое число ** однозначно раскладывается в произведение простых множителей

*.*

Если в этом равенстве собрать одинаковые множители, то получим каноническое разложение целого числа: .

**Определение 2.5.**Целые числа *а* и *b* называются сравнимыми по модулю *m*, если они удовлетворяют одному из условий теоремы 2.9.Этот факт обозначают формулой ** илии называют данную формулу сравнением.

Основные свойства сравнений:

**1.** Пусть *.* Тогда  для всякого целого *c*, то есть к обеим частям сравнения можно добавить (или вычесть из обеих частей) одно и то же число.

**2.** Сравнения можно почленно складывать и вычитать: если **, *,* то  

**3.** Сравнения можно почленно перемножать: если ** *,* то **.

**4.** Сравнения можно почленно возводить в любую натуральную степень: если *,* то **.

**5.** Если в сравнении ** числа *a*, *b*, *m* имеют общий множитель *d*, то на него сравнение можно сократить: **.

**6.** Сравнение можно сократить на общий множитель, взаимно простой с модулем: если **, НОД (*d*, *m*) = 1, то из сравнения  следует сравнимость  и  по модулю .

**7.** Сравнение можно умножить на любой целый множитель: если **, то  для всякого целого *t*.

**8.** Рефлексивность: ** для любого целого *а* и всякого натурального *m* >1.

**9.** Симметричность: если **, то **.

**10.** Транзитивность: если **, **, то .

*Теорема 2.10*(*Малая теорема Ферма*). Пусть *p –* простое число и целое число *a* не делится на . Тогд*а .*

Теория сравнений и малая теорема Ферма позволяют быстро находить остаток от деления большого числа на простое число.

**Ответы на вопросы**

**Сформулируйте алгоритм Евклида нахождения наибольшего общего делителя целых чисел.**

Алгоритм Евклида для нахождения наибольшего общего делителя (НОД) двух целых чисел a и b основан на последовательных делениях с остатком. Вот его формулировка:

a) Пусть a и b - два целых числа (a >= b).  
b) Если b равно 0, то НОД(a, b) равен a.  
c) Если b не равно 0, выполнить следующие шаги:

* + Разделить a на b и получить остаток r.
  + Присвоить a значение b и b значение r.
  + Вернуться к шагу c.

Повторяя шаги c до тех пор, пока остаток r не станет равным 0, получим НОД(a, b) в переменной a.

**Что значит расширенным алгоритмом Евклида?**

Расширенный алгоритм Евклида - это модификация алгоритма Евклида, которая помимо нахождения НОД(a, b) также позволяет выразить его через целочисленные коэффициенты x и y. Расширенный алгоритм Евклида может быть использован для нахождения обратного элемента по модулю и решения линейных диофантовых уравнений. Он основан на обратной подстановке в обратном порядке, начиная с последнего деления с остатком в обычном алгоритме Евклида.

**Какие числа называются взаимно простыми?**

Два целых числа называются взаимно простыми, если их наибольший общий делитель (НОД) равен 1. Другими словами, взаимно простые числа не имеют общих делителей, кроме 1.

**Объясните малую теорему Ферма?**

Малая теорема Ферма утверждает, что если p - простое число, а a - целое число, не делящееся на p, то a в степени p минус 1 при делении на p дает остаток 1. Формально, малая теорема Ферма может быть сформулирована следующим образом: Если p - простое число, а a - целое число, не делящееся на p, то a^(p-1) ≡ 1 (mod p). Здесь a^(p-1) обозначает a в степени p минус 1, а "≡" обозначает сравнение по модулю.

**Задание для выполнения**

1. Найти канонические разложения чисел а и b.
2. Найти НОД  пользуясь:

a) алгоритмом Евклида,

б) разложением чисел на простые множители.

1. С помощью расширенного алгоритма Евклида найти целые u, v, удовлетворяющие соотношению Безу: au + bv = НОД .
2. Найти остаток от деления данного числа на простое.

|  |  |
| --- | --- |
|  | 1-3. *а* = 5336161097, *b* = 196210799. .  4. Найти остаток от деления  на 19. |

**Решение:**

|  |
| --- |
|  |
| using System;  class Program  {  static void Main(string[] args)  {  long a = 5336161097;  long b = 196210799;  long primeNumber = 19;  // Находим каноническое разложение чисел а и b  Console.WriteLine($"Каноническое разложение числа а:");  CanonicalFactorization(a);  Console.WriteLine();  Console.WriteLine($"Каноническое разложение числа b = {b}:");  CanonicalFactorization(b);  Console.WriteLine();  // Находим НОД с использованием алгоритма Евклида  long gcdByEuclidean = GCDByEuclidean(a, b);  Console.WriteLine("НОД с использованием алгоритма Евклида: " + gcdByEuclidean);  Console.WriteLine();  // Находим НОД, используя разложение чисел на простые множители  long gcdByPrimeFactorization = GCDByPrimeFactorization(a, b);  Console.WriteLine("НОД с использованием разложения чисел на простые множители: " + gcdByPrimeFactorization);  Console.WriteLine();  // Находим целые u и v с помощью расширенного алгоритма Евклида  long u, v;  ExtendedEuclideanAlgorithm(a, b, out u, out v);  Console.WriteLine("Целые u и v, удовлетворяющие соотношению Безу: au + bv = НОД");  Console.WriteLine("u = " + u);  Console.WriteLine("v = " + v);  Console.WriteLine();  // Находим остаток от деления числа 1998 в степени 2001 на 19  long powerMod = CalculatePowerMod(1998, 2001, primeNumber);  Console.WriteLine("Остаток от деления 1998^2001 на 19: " + powerMod);  }  static void CanonicalFactorization(long number)  {  long factor = 2;  while (number > 1)  {  if (number % factor == 0)  {  Console.WriteLine($"{number}\t|" + factor + " ");  number /= factor;  }  else  {  factor++;  }  }  }  static long GCDByEuclidean(long a, long b)  {  while (b != 0)  {  long temp = b;  b = a % b;  a = temp;  }  return a;  }  static long GCDByPrimeFactorization(long a, long b)  {  long gcd = 1;  long factor = 2;  while (a > 1 && b > 1)  {  if (a % factor == 0 && b % factor == 0)  {  gcd \*= factor;  a /= factor;  b /= factor;  }  else if (a % factor == 0)  {  a /= factor;  }  else if (b % factor == 0)  {  b /= factor;  }  else  {  factor++;  }  }  return gcd;  }  static void ExtendedEuclideanAlgorithm(long a, long b, out long u, out long v)  {  long u1 = 1, u2 = 0, u3 = a;  long v1 = 0, v2 = 1, v3 = b;  while (v3 != 0)  {  long q = u3 / v3;  long t1 = u1 - q \* v1;  long t2 = u2 - q \* v2;  long t3 = u3 - q \* v3;  u1 = v1;  u2 = v2;  u3 = v3;  v1 = t1;  v2 = t2;  v3 = t3;  }  u = u1;  v = u2;  }  static long CalculatePowerMod(long baseNumber, long exponent, long modulus)  {  long result = 1;  while (exponent > 0)  {  if (exponent % 2 == 1)  {  result = (result \* baseNumber) % modulus;  }  baseNumber = (baseNumber \* baseNumber) % modulus;  exponent /= 2;  }  return result;  }  } |