МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ  
Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ   
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность 1—98 01 03 «Программное обеспечение информационной

безопасности мобильных систем»

**Отчет**

по дисциплине «Основы информационной безопасности»

**Исполнитель**

студент(ка) 2 курса 7 группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Банкузов М.О.

подпись, дата

**Руководитель**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ржеутская Н. В.

должность, ученая степень, ученое звание подпись, дата

**Практическое занятие №6**

**Тема « Теория чисел»**

Цель**:**  получение основных сведений из курса теории чисел

**Вариант 2**

|  |  |
| --- | --- |
| 2. | 1-3. *а* = 5999801, *b* = 48685811  4. Найти остаток от деления C:\Users\mysha\AppData\Local\Temp\ksohtml58508\wps3.png на 17. |

using System;

using System.Collections.Generic;

namespace laboib6

{

public static class PrimeFactorization

{

public static void PrintPrimeFactorization(int num)

{

if (num <= 0)

{

Console.WriteLine("Число должно быть положительным.");

return;

}

Dictionary<int, int> primeFactors = new Dictionary<int, int>();

for (int factor = 2; factor <= num; factor++)

{

while (num % factor == 0)

{

if (primeFactors.ContainsKey(factor))

primeFactors[factor]++;

else

primeFactors[factor] = 1;

num /= factor;

}

}

foreach (var kvp in primeFactors)

{

Console.WriteLine($"{kvp.Key}^{kvp.Value}");

}

}

}

public class EvclidForNOD

{

public static int FindNOD(int a, int b)

{

while (b != 0)

{

int temp = b;

b = a % b;

a = temp;

}

return a;

}

}

public class PrimeFactorizationUtility

{

// Функция для разложения числа на простые множители и возврата словаря простых множителей и их степеней

public static Dictionary<int, int> PrimeFactorization(int num)

{

Dictionary<int, int> primeFactors = new Dictionary<int, int>();

for (int factor = 2; factor <= num; factor++)

{

while (num % factor == 0)

{

if (primeFactors.ContainsKey(factor))

primeFactors[factor]++;

else

primeFactors[factor] = 1;

num /= factor;

}

}

return primeFactors;

}

// Функция для нахождения НОД двух чисел на основе разложения на простые множители

public static int FindGCDUsingPrimeFactorization(int a, int b)

{

Dictionary<int, int> primeFactorsA = PrimeFactorization(a);

Dictionary<int, int> primeFactorsB = PrimeFactorization(b);

int gcd = 1;

foreach (var kvpA in primeFactorsA)

{

if (primeFactorsB.ContainsKey(kvpA.Key))

{

int commonPower = Math.Min(kvpA.Value, primeFactorsB[kvpA.Key]);

gcd \*= (int)Math.Pow(kvpA.Key, commonPower);

}

}

return gcd;

}

}

public class ExtendedEuclideanAlgorithm

{

public static void ExtendedGCD(int a, int b, out int nodplus, out int u, out int v)

{

if (b == 0)

{

nodplus = a;

u = 1;

v = 0;

}

else

{

ExtendedGCD(b, a % b, out nodplus, out u, out v);

int temp = u;

u = v;

v = temp - (a / b) \* v;

}

}

}

public class ModularPowerCalculator

{

public static long CalculateModularPower(long baseNumber, long exponent, long modulus)

{

if (modulus == 1)

{

return 0;

}

long result = 1;

baseNumber = baseNumber % modulus;

while (exponent > 0)

{

if (exponent % 2 == 1)

{

result = (result \* baseNumber) % modulus;

}

exponent = exponent >> 1;

baseNumber = (baseNumber \* baseNumber) % modulus;

}

return result;

}

}

class Program

{

static void Main(string[] args)

{

int a = 5999801;

int b = 48685811;

Console.WriteLine($"Число a: {a} ");

Console.WriteLine($"Число b: {b}");

Console.WriteLine("\nЗадание 1");

Console.WriteLine($"Каноническое разложение числа a ({a}):");

PrimeFactorization.PrintPrimeFactorization(a);

Console.WriteLine($"Каноническое разложение числа b ({b}):");

PrimeFactorization.PrintPrimeFactorization(b);

Console.WriteLine("\nЗадание 2 - а. Алгоритм Евклида");

int nod = EvclidForNOD.FindNOD(a, b);

Console.WriteLine($"НОД({a}, {b}) = {nod}");

Console.WriteLine("\nЗадание 2 - b. Разложение чисел на простые множители");

int gcd = PrimeFactorizationUtility.FindGCDUsingPrimeFactorization(a, b);

Console.WriteLine($"НОД({a}, {b}) = {gcd}");

Console.WriteLine("\nЗадание 3. С помощью расширенного алгоритма Евклида найти целые u, v, удовлетворяющие соотношению Безу");

int nodplus, u, v;

ExtendedEuclideanAlgorithm.ExtendedGCD(a, b, out nodplus, out u, out v);

Console.WriteLine($"НОД({a}, {b}) = {nodplus}");

Console.WriteLine($"u = {u}, v = {v}");

Console.WriteLine("\nЗадание 4. Найти остаток от деления 1995^2004 на 16");

long baseNumber = 2005;

long exponent = 2003;

long modulus = 17;

long result = ModularPowerCalculator.CalculateModularPower(baseNumber, exponent, modulus);

Console.WriteLine($"{baseNumber}^{exponent} (mod {modulus}) = {result}");

}

}

}

