МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3 НА ТЕМУ:**

**ОСНОВЫ ТЕОРИИ ЧИСЕЛ И ИХ ИСПОЛЬЗОВАНИЕ В КРИПТОГРАФИИ**

                                                                 Выполнил студент 3 курса 5 группы

Дмитрук Илья Игоревич

Минск 2024

# Задание 1.

Для нахождения НОД двух чисел, использовался алгоритм Евклида. Его суть заключается в том, что мы большее число делим на меньшее и находим остаток. Затем меньшее число делим на найденный остаток, и получаем ещё один остаток. Затем делим больший остаток на меньший, и повторяем операции с делением остатков до тех пор, пока один из остатков не будет 0. Второй не нулевой остаток и будет являться НОД. Для реализации данного алгоритма, была разработана функция GCD. Так же была создана функция, в которой высчитывается НОД для трёх чисел. Она представляет с собой перегруженную функцию GCD, которая принимает три параметра. Она находит НОД для первых двух чисел, вызывая первую функцию GCD, затем ещё раз вызывает эту функцию, и передаёт первым параметром НОД первых двух чисел, а вторым параметром, третье число. Код данных функций представлен в листинге 1.1.

public static int GCD(int a, int b)

{

while (a != 0 && b != 0)

{

if(a > b) a = a % b;

else b = b % a;

}

return a + b;

}

public static int GCD(int a, int b, int c)

{

return GCD(GCD(a, b), c);

}

Листинг 1.1 – Функции для нахождения НОД

НОД высчитывался для чисел 499 и 531, а также чисел 499, 531 и 299 Результат представлен на рисунке 1.1.



Рисунок 1.1 – Результаты нахождения НОД

# Задание 2.

Для поиска простых чисел использовался алгоритм Решето Эратосфена. Его суть заключается в том, что мы берём число 2, умножаем его на само себя до тех пор, пока результат не выйдет за определённый предел, и при каждом умножении, полученный результат помечаем, что он не является простым числом. Затем берём следующее число, которое не помечено, как не простое, и повторяем действие, как с числом 2. И так, пока при первом умножении числа на само себя, результат выйдет за определённый предел. Данный алгоритм реализован в функции SieveEratosthenes. Её код представлен в листинге 2.1.

public static List<int> SieveEratosthenes(int m, int n)

{

bool[] prime = new bool[n + 1];

for (int i = 0; i <= n; i++)

prime[i] = true;

for (int p = 2; p \* p <= n; p++)

{

if (prime[p] == true)

{

for (int i = p \* p; i <= n; i += p)

prime[i] = false;

}

}

List<int> primeNumbers = new List<int>();

for (int i = 0; i <= n; i++)

{

if (prime[i] == true && i >= m)

{

primeNumbers.Add(i);

}

}

return primeNumbers;

}

Листинг 2.1 – Функция для нахождения простых чисел

Простые числа искались с начало в промежутке от 2 до 531. Количество подсчитанных простых чисел в интервале примерно совпадает с *n/*ln*(n)*. Результат представлен на рисунке 2.1.

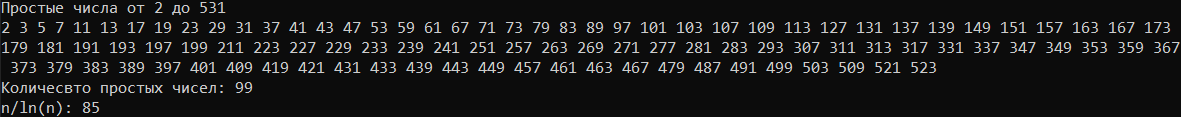


Рисунок 2.1 – Простые числа от 2 до 531

Затем находились простые числа в промежутке от 499 до 531. Результат представлен на рисунке 2.2.



Рисунок 2.2 – Простые числа от 499 до 531

Результат нахождения простых чисел был проверен ручным выполнением алгоритма Решето Эратосфена. Результат представлен на рисунке 2.3. Числа, не выделенные никаким цветом, являются простыми. Как мы видим, результат, найденный вручную, совпадает с программным результатом.

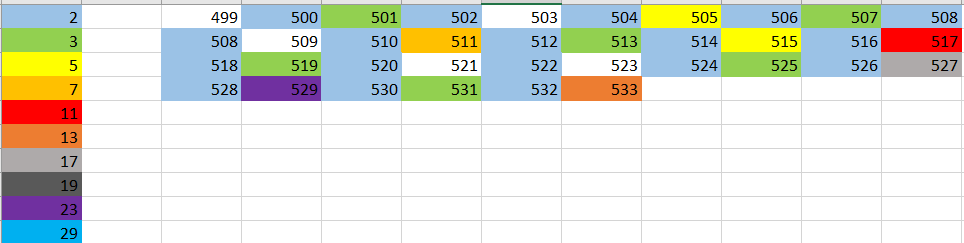


Рисунок 2.3 – Результат ручного поиска простых чисел

# Задание 3.

Для нахождения простых множителей, произведение которых даёт определённое число, и записи этих множителей в канонической форме, была разработана функция GetCanonicalForm. Её алгоритм работает следующим образом: сначала число делим на первое простое число, то есть на 2, затем результат делим так же, пока не получиться 1. Если число не делиться на текущее простое число, то делим на ближайшее простое, на которое делиться. Каждый делитель мы сохраняем в список. Затем функция подсчитывает количество встреч разных множителей. Результат преобразовывает в строку, вставляя символ степени после чисел и в качестве степени вставляет количество встреч числа. Между всеми такими результатами вставляется символ умножения, тем самым получается каноническая форма. Код данной функции представлен в листинге 3.1.

public static string GetCanonicalForm(int number)

{

List<int> primeNumers = new List<int>();

for (int divider = 2; divider \* divider <= number; divider++)

{

while (number % divider == 0)

{

primeNumers.Add(divider);

number /= divider;

}

}

if (number > 1)

{

primeNumers.Add(number);

}

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

foreach (int n in primeNumers) powers[n] = powers.ContainsKey(n) ?

powers[n] + 1 : 1;

string[] canonicalForm = new string[powers.Count];

int i = 0;

foreach (var n in powers)

{

if (n.Value != 1) canonicalForm[i] = $"{n.Key}^{n.Value}";

else canonicalForm[i] = $"{n.Key}";

i++;

}

return string.Join(" \* ", canonicalForm);

}

Листинг 3.1 – Функция для нахождения простых множителей и записи их в канонической форме

Простые множители находились для чисел 499 и 531. Результат в канонической форме представлен на рисунке 3.1.



Рисунок 3.1 – Простые множители в канонической форме

# Задание 4.

Для определения, является ли число, состоящее из конкатенации цифр, простым, была разработана функция IsConcatNumberPrime. Она объедините цифры переданных в неё чисел и, с помощью функции для поиска простых чисел SieveEratosthenes, определяет, находиться ли число в списке простых чисел. Код данной функции представлен в листинге 4.1.

public static bool IsConcatNumberPrime(int n, int m)

{

int concatNumber = Convert.ToInt32(n.ToString() + m.ToString());

List<int> primeNumbers = SieveEratosthenes(0, concatNumber);

return primeNumbers.Contains(concatNumber);

}

Листинг 4.1 – Функция для определения, является ли конкатенация чисел простым числом.

В качестве числа, определяемого как простое или нет, использовалась конкатенация цифр чисел 499 и 531. Результат представлен на рисунке 4.1.



Рисунок 4.1 – Результат определения, является ли конкатенация чисел простым числом

# Вывод

В ходе лабораторной работы были приобретены навыки работы с числами в криптографии. Были освоены алгоритмы для на нахождения НОД между числами, нахождения простых чисел и нахождения простых множителей числа. Так же были разработаны программные реализации данных алгоритмов.