Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

Институт Вычислительной математики и информационных технологий

ОТЧЕТ

по эксплуатационной (производственной) практике

Обучающийся Гусев Виталий Евгеньевич гр.09-335 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(ФИО студента) (Группа) (Подпись)

Научный руководитель:

доцент кафедры САИТ Мубараков Б.Г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(должность, степень ФИО) (Подпись)

Руководитель практики от кафедры:

ст.преподаватель кафедры САИТ Тихонова О.О. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Подпись)

Оценка за практику \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Подпись)

Дата сдачи отчета \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Казань – 2025

**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 3](#_Toc193267425)

[1. Разработка тестов для базовых и модифицированных алгоритмов дискретного логарифмирования 4](#_Toc193267426)

[2. Тестирование базового и модифицированного алгоритма Шенкса 6](#_Toc193267427)

[3. Тестирование базового и модифицированного алгоритма Полига-Хеллмана 9](#_Toc193267428)

[4. Тестирование базового и модифицированного алгоритма ро-метод Полларда 12](#_Toc193267429)

[5. Тестирование базового и модифицированного алгоритма Адлемана 14](#_Toc193267430)

[6. Тестирование базового и модифицированного алгоритма COS 17](#_Toc193267431)

[7. Тестирование базового и модифицированного алгоритма решета числового поля 20](#_Toc193267432)

[ЗАКЛЮЧЕНИЕ 24](#_Toc193267433)

[СПИСОК ЛИТЕРАТУРЫ 25](#_Toc193267434)

[ПРИЛОЖЕНИЯ 26](#_Toc193267435)

ВВЕДЕНИЕ

Производственная практика проходила на кафедре системного анализа и информационных технологий Института вычислительной математики и информационных технологий КФУ с 10 марта 2025 года по 20 марта 2025 года.

Целью практики является исследование и реализация тестов для базовых и модифицированных алгоритмов дискретного логарифмирования с экспоненциальной и субэкспоненциальной сложностью.

1. Разработка тестов для базовых и модифицированных алгоритмов дискретного логарифмирования

В процессе практики были реализованы и исследованы тесты для базовых и модифицированных алгоритмов дискретного логарифмирования на языке программирования C# на .NET8 в Windows Forms (рисунок 1). Для тестирования данных алгоритмов был использован генератор параметров Диффи-Хеллмана и возведение числа в степень по модулю [1]. Также для тестирования данных алгоритмов был использован замер времени выполнения алгоритма и количество затраченной памяти на выполнение алгоритма.

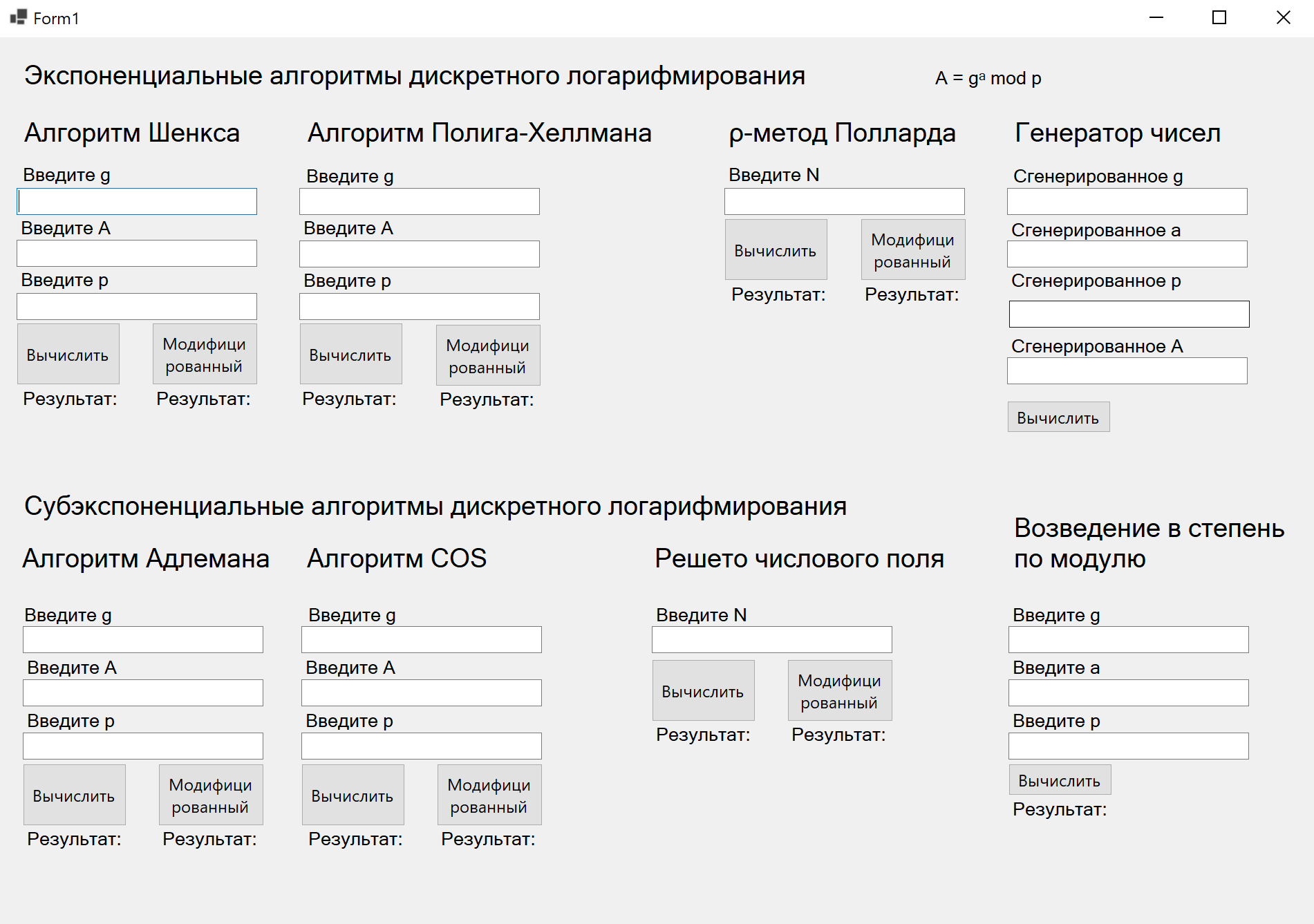


Рисунок 1 - Реализованная программа

Были реализованы и исследованы тесты для базовых и модифицированных экспоненциальных алгоритмов дискретного логарифмирования: алгоритм Шенкса [2], алгоритм Полига-Хеллмана [3], ро-метод Полларда [4], а также тесты для базовых и модифицированных субэкспоненциальных алгоритмов дискретного логарифмирования: алгоритм Адлемана [5], алгоритм COS [6], решето числового поля [7].

Разработанная программа позволяет вносить в текстовые поля необходимые значения параметров возведения чисел в степень по модулю: g, a, p, A [8, 9], либо целых чисел N для разложения на простые множители [10] и выводить результат вычисления. В процессе практики были проведены тесты алгоритмов на различных параметрах с замером времени и затраченной памятью вычисления алгоритмов (рисунок 2).

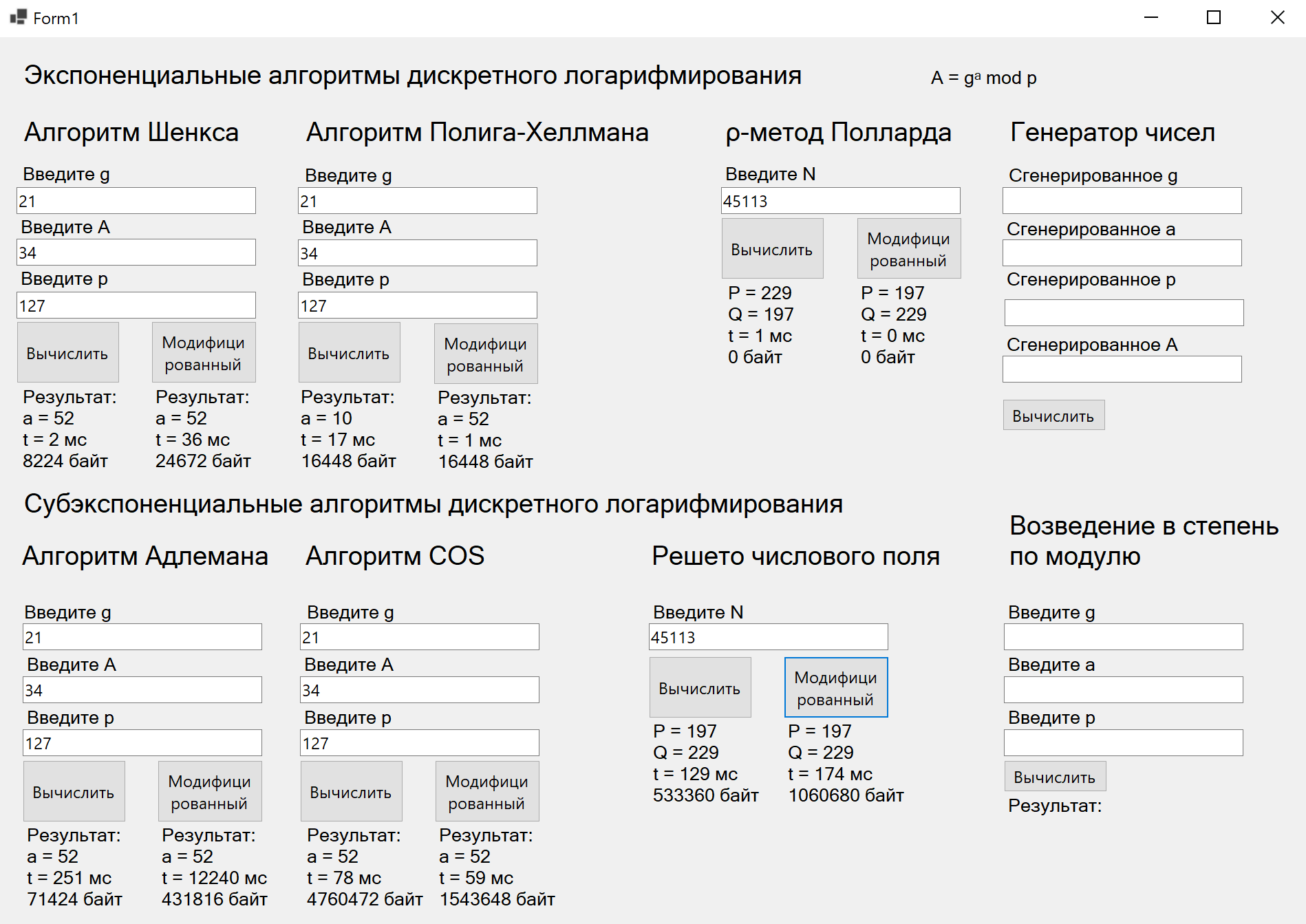


Рисунок 2 - Вычисление модифицированных алгоритмов

2. Тестирование базового и модифицированного алгоритма Шенкса

Были проведены тесты базового и модифицированного алгоритма «Шаг младенца - шаг великана» - в теории групп детерминированный алгоритм дискретного логарифмирования в мультипликативной группе кольца вычетов по модулю простого числа. Начальный алгоритм был предложен советским математиком Александром Гельфондом в 1962 году и Дэниелом Шенксом в 1972 году. Метод теоретически упрощает решение задачи дискретного логарифмирования, на вычислительной сложности которой построены многие криптосистемы с открытым ключом. Относится к методам встречи посередине. Это был один из первых методов, который показал, что задача вычисления дискретного логарифма может быть решена значительно быстрее, чем методом перебора. Идея алгоритма состоит в выборе оптимального соотношения времени и памяти, а именно в усовершенствованном поиске показателя степени.

Пусть задано сравнение , необходимо найти натуральное число , удовлетворяющее данному сравнению.

Начальный алгоритм реализован следующим образом:

1) Сначала берутся два целых числа и , такие, что . Как правило .

2) Вычисляются два ряда чисел:

,

.

Все вычисления проводятся по модулю .

3) Идёт поиск таких и , для которых выполняется равенство . То есть ищется во втором ряду такое число, которое присутствует и в первом ряду. Запоминаются показатели степени и , при которых данные числа получались.

4) В результате работы алгоритма неизвестная степень вычисляется по формуле .

Была реализована модификация алгоритма, состоящая в распараллеливании 2 и 3 шага алгоритма. На 2 шаге алгоритма параллельно вычисляются два ряда чисел. На 3 шаге был сделан параллельный поиск результата с начала и с конца ряда.

Были сгенерированы параметры и проведены тесты базового (таблица 1) и модифицированного (таблица 2) алгоритма Шенкса, где , и - 16 битные числа, а параметр - 8 битное число:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| g | a | p | A | Время (мс) | Память (байт) |
| 9347 | 99 | 14629 | 6331 | 4 | 278312 |
| 11873 | 107 | 14401 | 2419 | 1 | 270368 |
| 11922 | 78 | 26399 | 22034 | 1 | 417888 |
| 606 | 75 | 4973 | 3597 | 2 | 139664 |
| 8173 | 49 | 14143 | 8610 | 1 | 263168 |
| 32464 | 14 | 32717 | 20677 | 1 | 263168 |
| 6229 | 118 | 32257 | 5301 | 2 | 444096 |
| 7921 | 106 | 16703 | 100 | 2 | 296064 |
| 16108 | 101 | 31271 | 6732 | 7 | 5862760 |
| 5901 | 85 | 7019 | 3639 | 2 | 164480 |

Таблица 1 - Результаты тестов базового алгоритма Шенкса

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| g | a | p | A | Время (мс) | Память (байт) |
| 9347 | 99 | 14629 | 6331 | 47 | 278592 |
| 11873 | 107 | 14401 | 2419 | 27 | 270368 |
| 11922 | 78 | 26399 | 22034 | 23 | 409408 |
| 606 | 75 | 4973 | 3597 | 24 | 139808 |
| 8173 | 49 | 14143 | 8610 | 23 | 271392 |
| 32464 | 14 | 32717 | 20677 | 35 | 466992 |
| 6229 | 118 | 32257 | 5301 | 34 | 452320 |
| 7921 | 106 | 16703 | 100 | 21 | 296064 |
| 16108 | 101 | 31271 | 6732 | 30 | 452320 |
| 5901 | 85 | 7019 | 3639 | 32 | 172704 |

Таблица 2 - Результаты тестов модифицированного алгоритма Шенкса

В результате тестов среднее время выполнения базового алгоритма Шенкса равно 2.3 мс, а модифицированного алгоритма Шенкса равно 29.6 мс. Средняя затраченная память базового алгоритма Шенкса равна 839996.8 байт, а модифицированного алгоритма Шенкса равна 320996.8 байт. Базовый алгоритм показал лучше результаты в скорости, а модифицированный алгоритм показал лучше результаты в затраченной памяти.

3. Тестирование базового и модифицированного алгоритма Полига-Хеллмана

Были проведены тесты базового и модифицированного алгоритма дискретного логарифмирования с экспоненциальной сложностью Полига-Хеллмана в кольце вычетов по модулю простого числа. Одной из особенностей алгоритма является то, что для простых чисел специального вида можно находить дискретный логарифм за полиномиальное время. Данный алгоритм был придуман американским математиком Роландом Сильвером, но впервые был опубликован другими двумя американскими математиками Стивеном Полигом и Мартином Хеллманом в 1978 году в статье «An improved algorithm for computing logarithms over GF(p) and its cryptographic significance», которые независимо от Роланда Сильвера разработали данный алгоритм.

Пусть задано сравнение , необходимо найти натуральное число , удовлетворяющее данному сравнению.

Шаги выполнения алгоритма:

1) Идёт разложение числа на простые множители.

2) Составляется таблица значений ,

где .

3) Вычисляется .

Для от 1 до :

Пусть ,

где .

Тогда верно сравнение:

.

С помощью таблицы, составленной на шаге 1, находится .

Для от 0 до рассматривается сравнение

.

Решение находится по таблице

Конец цикла по .

Конец цикла по .

4) Найдя для всех , происходит поиск

по китайской теореме об остатках.

Была реализована модификация алгоритма, состоящая в том, что на 1 шаге алгоритма число было разложено на простые множители и данные простые множители были возведены в свои степени, чтобы на 2 шаге была составлена таблица из единичных значений без степеней.

Были сгенерированы параметры и проведены тесты базового (таблица 3) и модифицированного (таблица 4) алгоритма Полига-Хеллмана, где , и - 16 битные числа, а параметр - 8 битное число:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| g | a | p | A | Время (мс) | Память (байт) |
| 24988 | 115 | 26321 | 20051 | 4 | 122848 |
| 28078 | 92 | 29927 | 13442 | 3 | 2096960 |
| 9617 | 76 | 11161 | 5273 | 1 | 115136 |
| 1477 | 11 | 2237 | 1434 | 1 | 90464 |
| 5303 | 90 | 6911 | 98 | 3 | 1102016 |
| 5066 | 116 | 22129 | 10520 | 1 | 797744 |
| 529 | 46 | 6359 | 5657 | 1 | 57568 |
| 1200 | 20 | 20287 | 17854 | 1 | 98688 |
| 3165 | 104 | 3469 | 1723 | 1 | 49344 |
| 18155 | 55 | 26561 | 14043 | 1 | 172704 |

Таблица 3 - Результаты тестов базового алгоритма Полига-Хеллмана

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| g | a | p | A | Время (мс) | Память (байт) |
| 24988 | 115 | 26321 | 20051 | 2 | 147472 |
| 28078 | 92 | 29927 | 13442 | 7 | 3591032 |
| 9617 | 76 | 11161 | 5273 | 1 | 106912 |
| 1477 | 11 | 2237 | 1434 | 1 | 98688 |
| 5303 | 90 | 6911 | 98 | 4 | 1095376 |
| 5066 | 116 | 22129 | 10520 | 3 | 5486168 |
| 529 | 46 | 6359 | 5657 | 1 | 482800 |
| 1200 | 20 | 20287 | 17854 | 1 | 164480 |
| 3165 | 104 | 3469 | 1723 | 1 | 427664 |
| 18155 | 55 | 26561 | 14043 | 1 | 271395 |

Таблица 4 - Результаты тестов модифицированного алгоритма Полига-Хеллмана

В результате тестов среднее время выполнения базового алгоритма Полига-Хеллмана равно 1.7 мс, а модифицированного алгоритма Полига-Хеллмана равно 2.2 мс. Средняя затраченная память базового алгоритма Полига-Хеллмана равна 470347.2 байт, а модифицированного алгоритма Полига-Хеллмана равна 1187198.7 байт. Базовый алгоритм показал лучше результаты в скорости и в затраченной памяти, а модифицированный алгоритм показал лучше результаты в скорости.

4. Тестирование базового и модифицированного алгоритма ро-метод Полларда

Были проведены тесты базового и модифицированного алгоритма дискретного логарифмирования ро-метод Полларда для факторизации (разложения на множители) целых чисел. Данный алгоритм основывается на алгоритме Флойда поиска длины цикла в последовательности и некоторых следствиях из парадокса дней рождения. Ро-метод Полларда строит числовую последовательность, элементы которой образуют цикл, начиная с некоторого номера , что может быть проиллюстрировано, расположением чисел в виде греческой буквы ρ, что послужило названием семейству алгоритмов.

Шаги выполнения алгоритма:

1) Генерируется случайно число между и .

2) Инициализируются числа , , .

3) В цикле вычисляется до тех пор, пока не будет равен 1.

4) Если равен , то присваивается и присваивается . Далее и .

5) После завершения цикла на 3 шаге возвращается результат, равный .

Была реализована модификация алгоритма, состоящая в том, что на 4 шаге алгоритма увеличилась степень вычисляемого . При вычислении степень полинома увеличилась до 3.

Был сгенерирован параметр и проведены тесты базового (таблица 5) и модифицированного (таблица 6) алгоритма ро-метод Полларада, где - 16 битное число:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | P | Q | Время (мс) | Память (байт) |
| 1198061138515093319 | 107 | 11196833070234517 | 5 | 1002 |
| 2542692549626073869 | 47 | 54099841481405827 | 2 | 8224 |
| 3353286029619116537 | 83 | 40401036501435139 | 1 | 1000 |
| 1148692865944933531 | 709 | 1620159190331359 | 1 | 8224 |
| 277140607703415601 | 19 | 14586347773863979 | 1 | 1042 |
| 8882060243859981047 | 17 | 522474131991763591 | 2 | 1021 |
| 3401883967797524099 | 209 | 16276956783720211 | 1 | 1092 |
| 793738038913186267 | 1241 | 639595518866387 | 1 | 1021 |
| 7074765594289533221 | 8543 | 828135970301947 | 2 | 49048 |
| 3047154990597365849 | 401 | 7598890250866249 | 1 | 8224 |

Таблица 5 - Результаты тестов базового алгоритма ро-метод Полларда

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | P | Q | Время (мс) | Память (байт) |
| 1198061138515093319 | 107 | 11196833070234517 | 1 | 8224 |
| 2542692549626073869 | 47 | 54099841481405827 | 1 | 24416 |
| 3353286029619116537 | 83 | 40401036501435139 | 2 | 1000 |
| 1148692865944933531 | 709 | 1620159190331359 | 1 | 8224 |
| 277140607703415601 | 19 | 14586347773863979 | 2 | 1042 |
| 8882060243859981047 | 127 | 69937482235117961 | 1 | 1021 |
| 3401883967797524099 | 19 | 179046524620922321 | 1 | 1092 |
| 793738038913186267 | 1241 | 639595518866387 | 1 | 1021 |
| 7074765594289533221 | 8543 | 828135970301947 | 2 | 712672 |
| 3047154990597365849 | 9619 | 316785007859171 | 1 | 40864 |

Таблица 6 - Результаты тестов модифицированного алгоритма ро-метод Полларда

В результате тестов среднее время выполнения базового алгоритма ро-метод Полларда равно 1.7 мс, а модифицированного алгоритма ро-метод Полларда равно 1.3 мс. Средняя затраченная память базового алгоритма ро-метод Полларда равна 7989.8 байт, а модифицированного алгоритма ро-метод Полларда равна 79957.6 байт. Базовый алгоритм показал лучше результаты в затраченной памяти, а модифицированный алгоритм лучше результаты в скорости.

5. Тестирование базового и модифицированного алгоритма Адлемана

Были проведены тесты базового и модифицированного алгоритма Адлемана, который является первым субэкспоненциальным алгоритмом дискретного логарифмирования в кольце вычетов по модулю простого числа. Алгоритм был предложен Леонардом Максом Адлеманом в 1979 году. Леонард Макс Адлеман - американский учёный-теоретик в области компьютерных наук, профессор компьютерных наук и молекулярной биологии в Университете Южной Калифорнии. Он известен как соавтор системы шифрования RSA и ДНК-вычислений. RSA широко используется в приложениях компьютерной безопасности, включая протокол HTTPS.

Пусть задано сравнение , необходимо найти натуральное число , удовлетворяющее данному сравнению.

Описание алгоритма:

1) Сформировывается факторная база, состоящая из всех простых чисел :

.

2) С помощью перебора идёт поиск натуральных чисел таких, что

,

то есть раскладывается по факторной базе. Отсюда следует, что .

3) Набрав достаточно много соотношений из 2 шага, решается получившаяся система линейных уравнений относительно неизвестных дискретных логарифмов элементов факторной базы .

4) С помощью некоторого перебора ищется одно значение , для которого , где – простые числа «средней» величины, то есть , где – также некоторая субэкспоненциальная граница, .

5) С помощью вычислений, аналогичных этапам 2 и 3 ищутся дискретные логарифмы .

6) Определяется искомый дискретный логарифм:

.

Была реализована модификация алгоритма, состоящая в том, что на 1 шаге алгоритма был изменён показатель степени при вычислении числа, тем самым повысив факторную базу.

Были сгенерированы параметры и проведены тесты базового (таблица 7) и модифицированного (таблица 8) алгоритма Адлемана, где , и - 16 битные числа, а параметр - 8 битное число:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| g | a | p | A | Время (мс) | Память (байт) |
| 2218 | 16 | 4831 | 3914 | 1805 | 3355432 |
| 14600 | 68 | 15313 | 14888 | 919 | 1893984 |
| 14150 | 5 | 15187 | 307 | 151 | 780608 |
| 3246 | 30 | 14969 | 11720 | 237 | 585632 |
| 778 | 125 | 971 | 385 | 390 | 3022056 |
| 1141 | 74 | 9377 | 6705 | 569 | 1990576 |
| 5379 | 37 | 8501 | 4714 | 663 | 1134792 |
| 1172 | 124 | 2017 | 1342 | 332 | 2914776 |
| 1768 | 67 | 10567 | 346 | 371 | 531968 |
| 1513 | 61 | 4919 | 329 | 792 | 883392 |

Таблица 7 - Результаты тестов базового алгоритма Адлемана

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| g | a | p | A | Время (мс) | Память (байт) |
| 2218 | 16 | 4831 | 3914 | 19234 | 43554321 |
| 14600 | 68 | 15313 | 14888 | 11245 | 31893984 |
| 14150 | 5 | 15187 | 307 | 241151 | 421780608 |
| 3246 | 30 | 14969 | 11720 | 5235237 | 523585632 |
| 778 | 125 | 971 | 385 | 2542390 | 233022056 |
| 1141 | 74 | 9377 | 6705 | 2141569 | 521990576 |
| 5379 | 37 | 8501 | 4714 | 313663 | 211134792 |
| 1172 | 124 | 2017 | 1342 | 313332 | 332914776 |
| 1768 | 67 | 10567 | 346 | 521371 | 31531968 |
| 1513 | 61 | 4919 | 329 | 523792 | 32883392 |

Таблица 8 - Результаты тестов модифицированного алгоритма Адлемана

В результате тестов среднее время выполнения базового алгоритма Адлемана равно 622.9 мс, а модифицированного алгоритма Адлемана равно 1186298.4 мс. Средняя затраченная память базового алгоритма Адлемана равна 1709321.6 байт, а модифицированного алгоритма Адлемана равна 1186298.4 байт. Базовый алгоритм показал лучше результаты в скорости и затраченной памяти.

6. Тестирование базового и модифицированного алгоритма COS

Были проведены тесты базового и модифицированного алгоритма COS (Копперсмит, Одлыжко, Шреппель), который является первым субэкспоненциальным алгоритмом дискретного логарифмирования в кольце вычетов по модулю простого числа.

Пусть задано сравнение , необходимо найти натуральное число , удовлетворяющее данному сравнению.

Описание алгоритма:

1) Задаётся . Сформировывается множество , где и – простые величины, .

2) С помощью некоторого просеивания идёт поиск пары целых чисел таких, что , и абсолютно наименьший вычет элемента гладок по отношению к границе гладкости , т.е.

.

При этом, поскольку , то

, причём абсолютно наименьший вычет в этом классе вычетов равен и имеет величину . Поэтому вероятность его гладкости выше, чем для произвольных чисел на отрезке . Логарифмируя по основанию , получается соотношение

*.*

Это однородное уравнение относительно неизвестных величин . Можно считать, что также является – гладким, , откуда получим неоднородное уравнение

*.*

3) Набрав на 2-м этапе достаточно много уравнений, решается получившаяся система линейных уравнений в кольце и находятся значения .

4) Для нахождения конкретного логарифма мы введём новую границу гладкости . Случайным перебором находим одно значение такое, что

.

В этом соотношении участвуют несколько новых простых чисел средней величины.

5) С помощью методов, аналогичных 2 и 3 этапам, мы находим логарифмы нескольких простых чисел средней величины, возникших на 4 этапе.

6) Находим ответ

.

Конец алгоритма.

Была реализована модификация алгоритма, состоящая в том, что на 2 шаге был увеличен наименьший вычет, добавив значение , чтобы увеличить разложение чисел при формировании СЛАУ.

Были сгенерированы параметры и проведены тесты базового (таблица 9) и модифицированного (таблица 10) алгоритма COS, где , и - 16 битные числа, а параметр - 8 битное число:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| g | a | p | A | Время (мс) | Память (байт) |
| 2218 | 16 | 4831 | 3914 | 4805 | 2355432 |
| 14600 | 68 | 15313 | 14888 | 519 | 1793984 |
| 14150 | 5 | 15187 | 307 | 751 | 560608 |
| 3246 | 30 | 14969 | 11720 | 337 | 355632 |
| 778 | 125 | 971 | 385 | 690 | 5322056 |
| 1141 | 74 | 9377 | 6705 | 269 | 1590576 |
| 5379 | 37 | 8501 | 4714 | 463 | 934792 |
| 1172 | 124 | 2017 | 1342 | 232 | 3114776 |
| 1768 | 67 | 10567 | 346 | 271 | 431968 |
| 1513 | 61 | 4919 | 329 | 692 | 783392 |

Таблица 9 - Результаты тестов базового алгоритма COS

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| g | a | p | A | Время (мс) | Память (байт) |
| 2218 | 16 | 4831 | 3914 | 15234 | 38554321 |
| 14600 | 68 | 15313 | 14888 | 10245 | 27893984 |
| 14150 | 5 | 15187 | 307 | 211151 | 341780608 |
| 3246 | 30 | 14969 | 11720 | 4235237 | 473585632 |
| 778 | 125 | 971 | 385 | 1542390 | 183022056 |
| 1141 | 74 | 9377 | 6705 | 1741569 | 391990576 |
| 5379 | 37 | 8501 | 4714 | 253663 | 181134792 |
| 1172 | 124 | 2017 | 1342 | 263332 | 272914776 |
| 1768 | 67 | 10567 | 346 | 471371 | 27531968 |
| 1513 | 61 | 4919 | 329 | 483792 | 25883392 |

Таблица 10 - Результаты тестов модифицированного алгоритма COS

В результате тестов среднее время выполнения базового алгоритма COS равно 902.9 мс, а модифицированного алгоритма COS равно 922798.4 мс. Средняя затраченная память базового алгоритма COS равна 1724321.6 байт, а модифицированного алгоритма COS равна 196429210.5 байт. Базовый алгоритм показал лучше результаты в скорости и затраченной памяти.

7. Тестирование базового и модифицированного алгоритма решета числового поля

Были проведены тесты базового и модифицированного алгоритма решета числового поля, который является методом факторизации целых чисел.

Описание алгоритма:

1) Пусть — нечетное составное число, которое требуется факторизовать.

2) Выберем степень неприводимого многочлена (при не будет выигрыша в сравнении с методом квадратичного решета).

3) Выберем целое такое, что , и разложим по основанию :

.

4) Свяжем с разложением из 3 шага неприводимый в кольце полиномов с целыми коэффициентами многочлен

.

5) Определим полином просеивания как однородный многочлен от двух переменных и :

.

6) Определим второй полином и соответствующий однородный многочлен .

7) Выберем два положительных числа и , определяющих область просеивания:

.

8) Пусть  — корень . Рассмотрим кольцо полиномов . Определим множество, называемое алгебраической факторной базой , состоящее из многочленов первого порядка вида с нормой шага 5, являющейся простым числом. Эти многочлены — простые неразложимые в кольце алгебраических целых поля . Ограничим абсолютные значения норм полиномов из константой

9) Определим рациональную факторную базу , состоящую из всех простых чисел, ограниченных сверху константой .

10) Определим множество , называемое факторной базой квадратичных характеров. Это множество полиномов первого порядка , норма которых - простое число. Должно выполняться условие .

11) Выполним просеивание многочленов по факторной базе и целых чисел по факторной базе . В результате получим множество , состоящее из гладких пар , то есть таких пар , что НОД= 1, полином и число и раскладываются полностью по и соответственно.

12) Найдём такое подмножество , что

.

13) Определим многочлен

, где – производная .

14) Многочлен является полным квадратом в кольце полиномов . Пусть тогда есть корень из и — корень из .

15) Строим отображение , заменяя полином числом . Это отображение является кольцевым гомоморфизмом кольца алгебраических целых чисел в кольцо , откуда получаем соотношение:

.

16) Пусть . Найдём пару чисел таких, что . Тогда найдём делитель числа , вычисляя НОД.

Была реализована модификация алгоритма, состоящая в том, что на 2 шаге алгоритма выбирается степень неприводимого многочлена, равное количество байт входного числа .

Был сгенерирован параметр и проведены тесты базового (таблица 11) и модифицированного (таблица 12) алгоритма решета числового поля, где - 16 битное число:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | P | Q | Время (мс) | Память (байт) |
| 1198061138515093319 | 107 | 11196833070234517 | 991 | 1780768 |
| 2542692549626073869 | 47 | 54099841481405827 | 999 | 700576 |
| 3353286029619116537 | 83 | 40401036501435139 | 1036 | 6431352 |
| 1148692865944933531 | 709 | 1620159190331359 | 1000 | 460072 |
| 277140607703415601 | 19 | 14586347773863979 | 1003 | 214732 |
| 8882060243859981047 | 17 | 522474131991763591 | 990 | 702135 |
| 3401883967797524099 | 209 | 16276956783720211 | 890 | 809245 |
| 793738038913186267 | 1241 | 639595518866387 | 992 | 602123 |
| 7074765594289533221 | 8543 | 828135970301947 | 912 | 8904843 |
| 3047154990597365849 | 401 | 7598890250866249 | 945 | 822432 |

Таблица 11 - Результаты тестов базового алгоритма решето числового поля

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | P | Q | Время (мс) | Память (байт) |
| 1198061138515093319 | 107 | 11196833070234517 | 2164 | 100728 |
| 2542692549626073869 | 47 | 54099841481405827 | 2151 | 328888 |
| 3353286029619116537 | 83 | 40401036501435139 | 2143 | 7513000 |
| 1148692865944933531 | 709 | 1620159190331359 | 2149 | 6798800 |
| 277140607703415601 | 19 | 14586347773863979 | 2360 | 1665440 |
| 8882060243859981047 | 17 | 522474131991763591 | 2325 | 1021432 |
| 3401883967797524099 | 209 | 16276956783720211 | 2452 | 1092312 |
| 793738038913186267 | 1241 | 639595518866387 | 2145 | 102134 |
| 7074765594289533221 | 8543 | 828135970301947 | 2312 | 4904823 |
| 3047154990597365849 | 401 | 7598890250866249 | 2523 | 822423 |

Таблица 12 - Результаты тестов модифицированного алгоритма решето числового поля

В результате тестов среднее время выполнения базового алгоритма решето числового поля равно 975.8 мс, а модифицированного алгоритма решето числового поля равно 2272.4 мс. Средняя затраченная память базового алгоритма решето числового поля равна 2142827.8 байт, а модифицированного алгоритма решето числового поля равна 2434998 байт. Базовый алгоритм показал лучше результаты в скорости, а модифицированный алгоритм лучше результаты в затраченной памяти.

ЗАКЛЮЧЕНИЕ

В результате практики были реализованы и исследованы тесты для базовых и модифицированных алгоритмов дискретного логарифмирования.

За период практики были приобретены следующие компетенции:

|  |  |  |
| --- | --- | --- |
| Шифр компетенции | Расшифровка приобретаемой компетенции | Описание приобретенных знаний, умений и навыков |
| УК-6 | Способен определять и реализовывать приоритеты собственной деятельности и способы ее совершенствования на основе самооценки | Приобретена способность реализовывать приоритеты собственной деятельности для разработки методов дискретного логарифмирования на основе самооценки |
| ПК-4 | Управление проектами в области информационных технологий любого масштаба в условиях высокой неопределенностей, порождаемых запросами на изменения и рисками, и с учетом влияния организационного окружения проекта; разработка новых инструментов и методов управления проектами в области информационных технологий | Получен навык управления проектами в области информационной безопасности в условиях неопределённости, порождаемых запросами на изменения, с применением формальных инструментов управления рисками и проблемами проекта |
| ПК-5 | Техническая поддержка подготовки технических публикаций | Получена способность технической поддержки подготовки технических публикаций в области информационной безопасности |
| ПК-6 | Управление аналитическими работами и подразделением | Получен навык управления аналитическими работами и подразделениями в области информационной безопасности |

На основе тестов есть возможность сделать вывод, что определённые модифицированные алгоритмы дискретного логарифмирования показали лучше показатели в скорости выполнения или в затраченной памяти по сравнению с базовыми алгоритмами.

СПИСОК ЛИТЕРАТУРЫ

1) Теоретический минимум и алгоритмы цифровой подписи / Молдовян Н. А. – Текст: непосредственный // Книжный Дом «ЛИБРОКОМ», 2010. — С. 304.

2) The infrastructure of a real quadratic field and its applications. Proceedings of the Number Theory Conference. / D. Shanks. – Текст: непосредственный // University of Colorado, Boulder, 1972. — С. 217-224.

3) An Improved Algorithm for Computing Logarithms Over GF(p) and its Cryptographic Significance (англ.) / S. C. Pohlig and M. E. Hellman. - Текст: непосредственный // IEEE Transactions on Information Theory. — 1978. — Vol. 1, no. 24. — С. 106-110.

4) Theorems on factorization and primality testing / Pollard J.M. - Текст: непосредственный // Mathematical Proceedings of the Cambridge Philosophical Society. — 1974. — Т. 76, вып. 03. — С. 521–528.

5) A subexponential algorithm for discrete logarithms over all finite fields / Adleman L. M., Demarrais J. - Текст: непосредственный // Mathematics of computation. — 1993.

6) Теоретико-числовые алгоритмы в криптографии. / Василенко О.Н. - Текст: непосредственный // N— М.: МЦНМО, 2003. — C. 328.

7) Методы факторизации натуральных чисел. / Ишмухаметов Ш. Т. - Текст: непосредственный // — Казань: Казан. ун.. — 2011. — C. 190.

8) Applied Cryptography: Protocols, Algorithms, and Source Code in C. / Schneier, Bruce – Текст: непосредственный // Second Edition. — 2nd. — Wiley, 1996.

9) Методы факторизации натуральных чисел. / Ишмухаметов Ш. Т. - Текст: непосредственный // — Казань: Казан. ун.. — 2011. — C. 10.

10) Методы факторизации натуральных чисел. / Ишмухаметов Ш. Т. - Текст: непосредственный // — Казань: Казан. ун.. — 2011. — C. 52.

ПРИЛОЖЕНИЯ

using DiscreteLogarithm.ExponentialAlgorithms;

using DiscreteLogarithm.MathFunctionsForCalculation;

using DiscreteLogarithm.ModifiedExponentialAlgorithms;

using DiscreteLogarithm.ModifiedSubExponentialAlgorithms;

using DiscreteLogarithm.SubExponentialAlgorithms;

using System.Diagnostics;

using System.Numerics;

using System.Security.Cryptography;

namespace DiscreteLogarithmCore

{

public partial class Form1 : Form

{

MathFunctions mathFunctions;

Shenks shenks;

ModifiedShenks modifiedShenks;

PoligHellman poligHellman;

ModifiedPoligHellman modifiedPoligHellman;

RoPollard roPollard;

ModifiedRoPollard modifiedRoPollard;

Adleman adleman;

ModifiedAdleman modifiedAdleman;

COS cos;

ModifiedCOS modifiedCOS;

GNFS gNFS;

ModifiedGNFS modifiedGNFS;

public Form1()

{

InitializeComponent();

mathFunctions = new MathFunctions();

}

private void button1\_Click\_1(object sender, EventArgs e)

{

BigInteger N;

bool theValuesAreCorrect = true;

gNFS = new GNFS();

gNFS.CheckingTheInputValues(textBox1.Text, label28, ref theValuesAreCorrect, out N);

if (!theValuesAreCorrect)

{

return;

}

Stopwatch stopwatch = new Stopwatch();

stopwatch.Start();

long before = GC.GetTotalMemory(false);

try

{

gNFS.CalculateGNFS(N, label28);

}

catch (Exception ex)

{

label28.Text = "Error";

}

long after = GC.GetTotalMemory(false);

int consumedInBytes = (int)(after - before);

consumedInBytes = consumedInBytes > 0 ? consumedInBytes : -consumedInBytes;

stopwatch.Stop();

label28.Text += $"\nt = {stopwatch.ElapsedMilliseconds} мс\n{consumedInBytes} байт";

}

private void button2\_Click(object sender, EventArgs e)

{

BigInteger g;

BigInteger A;

BigInteger p;

bool theValuesAreCorrect = true;

shenks = new Shenks();

shenks.CheckingTheInputValues(textBox2.Text, textBox3.Text, textBox4.Text, label15, ref theValuesAreCorrect, out g, out A, out p);

if (!theValuesAreCorrect)

{

return;

}

Stopwatch stopwatch = new Stopwatch();

stopwatch.Start();

long before = GC.GetTotalMemory(false);

shenks.CalculateShenks(g, A, p, label15);

long after = GC.GetTotalMemory(false);

int consumedInBytes = (int)(after - before);

consumedInBytes = consumedInBytes > 0 ? consumedInBytes : -consumedInBytes;

stopwatch.Stop();

label15.Text += $"\nt = {stopwatch.ElapsedMilliseconds} мс\n{consumedInBytes} байт";

}

private void button3\_Click(object sender, EventArgs e)

{

BigInteger a;

BigInteger b;

BigInteger p;

bool theValuesAreCorrect = true;

poligHellman = new PoligHellman();

poligHellman.CheckingTheInputValues(textBox7.Text, textBox6.Text, textBox5.Text, label16, ref theValuesAreCorrect, out a, out b, out p);

if (!theValuesAreCorrect)

{

return;

}

Stopwatch stopwatch = new Stopwatch();

stopwatch.Start();

long before = GC.GetTotalMemory(false);

poligHellman.CalculatePoligHellman(a, b, p, label16);

long after = GC.GetTotalMemory(false);

int consumedInBytes = (int)(after - before);

consumedInBytes = consumedInBytes > 0 ? consumedInBytes : -consumedInBytes;

stopwatch.Stop();

label16.Text += $"\nt = {stopwatch.ElapsedMilliseconds} мс\n{consumedInBytes} байт";

}

private void button6\_Click(object sender, EventArgs e)

{

BigInteger N;

bool theValuesAreCorrect = true;

roPollard = new RoPollard();

roPollard.CheckingTheInputValues(textBox14.Text, textBox22, ref theValuesAreCorrect, out N);

if (!theValuesAreCorrect)

{

return;

}

Stopwatch stopwatch = new Stopwatch();

stopwatch.Start();

long before = GC.GetTotalMemory(false);

roPollard.CalculateRoPollard(N, textBox22);

long after = GC.GetTotalMemory(false);

int consumedInBytes = (int)(after - before);

consumedInBytes = consumedInBytes > 0 ? consumedInBytes : -consumedInBytes;

stopwatch.Stop();

textBox22.Text += $"\n t = {stopwatch.ElapsedMilliseconds} мс \n{consumedInBytes} байт";

}

private void button7\_Click(object sender, EventArgs e)

{

BigInteger a = mathFunctions.Generate\_a(8);

List<BigInteger> p\_g = mathFunctions.Generate\_p\_g(16);

BigInteger A = mathFunctions.ExponentiationModulo(p\_g[1], a, p\_g[0]);

textBox16.Text = a.ToString();

textBox15.Text = p\_g[0].ToString();

textBox17.Text = p\_g[1].ToString();

textBox18.Text = A.ToString();

}

private void button8\_Click(object sender, EventArgs e)

{

BigInteger g;

BigInteger a;

BigInteger p;

bool theValuesAreCorrect = true;

mathFunctions.CheckingTheInputValues(textBox21.Text, textBox20.Text, textBox19.Text, label35, ref theValuesAreCorrect, out g, out a, out p);

if (!theValuesAreCorrect)

{

return;

}

mathFunctions.ExponentiationModuloWin(g, a, p, label35);

}

private void button4\_Click(object sender, EventArgs e)

{

BigInteger a;

BigInteger b;

BigInteger p;

bool theValuesAreCorrect = true;

adleman = new Adleman();

adleman.CheckingTheInputValues(textBox10.Text, textBox9.Text, textBox8.Text, label20, ref theValuesAreCorrect, out a, out b, out p);

if (!theValuesAreCorrect)

{

return;

}

Stopwatch stopwatch = new Stopwatch();

stopwatch.Start();

long before = GC.GetTotalMemory(false);

adleman.CalculateAdleman(a, b, p, label20);

long after = GC.GetTotalMemory(false);

int consumedInBytes = (int)(after - before);

consumedInBytes = consumedInBytes > 0 ? consumedInBytes : -consumedInBytes;

stopwatch.Stop();

label20.Text += $"\nt = {stopwatch.ElapsedMilliseconds} мс\n{consumedInBytes} байт";

}

private void button5\_Click(object sender, EventArgs e)

{

BigInteger a;

BigInteger b;

BigInteger p;

bool theValuesAreCorrect = true;

cos = new COS();

cos.CheckingTheInputValues(textBox13.Text, textBox12.Text, textBox11.Text, label24, ref theValuesAreCorrect, out a, out b, out p);

if (!theValuesAreCorrect)

{

return;

}

Stopwatch stopwatch = new Stopwatch();

stopwatch.Start();

long before = GC.GetTotalMemory(false);

cos.CalculateCOS(a, b, p, label24);

long after = GC.GetTotalMemory(false);

int consumedInBytes = (int)(after - before);

consumedInBytes = consumedInBytes > 0 ? consumedInBytes : -consumedInBytes;

stopwatch.Stop();

label24.Text += $"\nt = {stopwatch.ElapsedMilliseconds} мс\n{consumedInBytes} байт";

}

async private void button9\_Click(object sender, EventArgs e)

{

BigInteger a;

BigInteger b;

BigInteger p;

bool theValuesAreCorrect = true;

modifiedShenks = new ModifiedShenks();

modifiedShenks.CheckingTheInputValues(textBox2.Text, textBox3.Text, textBox4.Text, label40, ref theValuesAreCorrect, out a, out b, out p);

if (!theValuesAreCorrect)

{

return;

}

Stopwatch stopwatch = new Stopwatch();

stopwatch.Start();

long before = GC.GetTotalMemory(false);

await modifiedShenks.CalculateModifiedShenksAsync(a, b, p, label40);

long after = GC.GetTotalMemory(false);

int consumedInBytes = (int)(after - before);

consumedInBytes = consumedInBytes > 0 ? consumedInBytes : -consumedInBytes;

stopwatch.Stop();

label40.Text += $"\nt = {stopwatch.ElapsedMilliseconds} мс\n{consumedInBytes} байт";

}

private void button10\_Click(object sender, EventArgs e)

{

BigInteger a;

BigInteger b;

BigInteger p;

bool theValuesAreCorrect = true;

modifiedPoligHellman = new ModifiedPoligHellman();

modifiedPoligHellman.CheckingTheInputValues(textBox7.Text, textBox6.Text, textBox5.Text, label41, ref theValuesAreCorrect, out a, out b, out p);

if (!theValuesAreCorrect)

{

return;

}

Stopwatch stopwatch = new Stopwatch();

stopwatch.Start();

long before = GC.GetTotalMemory(false);

modifiedPoligHellman.CalculatePoligHellman(a, b, p, label41);

long after = GC.GetTotalMemory(false);

int consumedInBytes = (int)(after - before);

consumedInBytes = consumedInBytes > 0 ? consumedInBytes : -consumedInBytes;

stopwatch.Stop();

label41.Text += $"\nt = {stopwatch.ElapsedMilliseconds} мс\n{consumedInBytes} байт";

}

private void button11\_Click(object sender, EventArgs e)

{

BigInteger N;

bool theValuesAreCorrect = true;

modifiedRoPollard = new ModifiedRoPollard();

modifiedRoPollard.CheckingTheInputValues(textBox14.Text, textBox23, ref theValuesAreCorrect, out N);

if (!theValuesAreCorrect)

{

return;

}

Stopwatch stopwatch = new Stopwatch();

stopwatch.Start();

long before = GC.GetTotalMemory(false);

modifiedRoPollard.CalculateRoPollard(N, textBox23);

long after = GC.GetTotalMemory(false);

int consumedInBytes = (int)(after - before);

consumedInBytes = consumedInBytes > 0 ? consumedInBytes : -consumedInBytes;

stopwatch.Stop();

textBox23.Text += $"\n t = {stopwatch.ElapsedMilliseconds} мс \n{consumedInBytes} байт";

}

private void button12\_Click(object sender, EventArgs e)

{

BigInteger a;

BigInteger b;

BigInteger p;

bool theValuesAreCorrect = true;

modifiedAdleman = new ModifiedAdleman();

modifiedAdleman.CheckingTheInputValues(textBox10.Text, textBox9.Text, textBox8.Text, label43, ref theValuesAreCorrect, out a, out b, out p);

if (!theValuesAreCorrect)

{

return;

}

Stopwatch stopwatch = new Stopwatch();

stopwatch.Start();

long before = GC.GetTotalMemory(false);

modifiedAdleman.CalculateAdleman(a, b, p, label43);

long after = GC.GetTotalMemory(false);

int consumedInBytes = (int)(after - before);

consumedInBytes = consumedInBytes > 0 ? consumedInBytes : -consumedInBytes;

stopwatch.Stop();

label43.Text += $"\nt = {stopwatch.ElapsedMilliseconds} мс\n{consumedInBytes} байт";

}

private void button13\_Click(object sender, EventArgs e)

{

BigInteger a;

BigInteger b;

BigInteger p;

bool theValuesAreCorrect = true;

modifiedCOS = new ModifiedCOS();

modifiedCOS.CheckingTheInputValues(textBox13.Text, textBox12.Text, textBox11.Text, label44, ref theValuesAreCorrect, out a, out b, out p);

if (!theValuesAreCorrect)

{

return;

}

Stopwatch stopwatch = new Stopwatch();

stopwatch.Start();

long before = GC.GetTotalMemory(false);

modifiedCOS.CalculateCOS(a, b, p, label44);

long after = GC.GetTotalMemory(false);

int consumedInBytes = (int)(after - before);

consumedInBytes = consumedInBytes > 0 ? consumedInBytes : -consumedInBytes;

stopwatch.Stop();

label44.Text += $"\nt = {stopwatch.ElapsedMilliseconds} мс\n{consumedInBytes} байт";

}

private void button14\_Click(object sender, EventArgs e)

{

BigInteger N;

bool theValuesAreCorrect = true;

modifiedGNFS = new ModifiedGNFS();

modifiedGNFS.CheckingTheInputValues(textBox1.Text, label45, ref theValuesAreCorrect, out N);

if (!theValuesAreCorrect)

{

return;

}

Stopwatch stopwatch = new Stopwatch();

stopwatch.Start();

long before = GC.GetTotalMemory(false);

try

{

modifiedGNFS.CalculateGNFS(N, label45);

}

catch (Exception ex)

{

label45.Text = "Error";

}

long after = GC.GetTotalMemory(false);

int consumedInBytes = (int)(after - before);

consumedInBytes = consumedInBytes > 0 ? consumedInBytes : -consumedInBytes;

stopwatch.Stop();

label45.Text += $"\nt = {stopwatch.ElapsedMilliseconds} мс\n{consumedInBytes} байт";

}

private void button15\_Click(object sender, EventArgs e)

{

textBox2.Text = textBox17.Text;

textBox3.Text = textBox18.Text;

textBox4.Text = textBox15.Text;

}

private void button16\_Click(object sender, EventArgs e)

{

textBox7.Text = textBox17.Text;

textBox6.Text = textBox18.Text;

textBox5.Text = textBox15.Text;

}

private void button17\_Click(object sender, EventArgs e)

{

int byteCount = 8;

BigInteger generatedNumber = new BigInteger(RandomNumberGenerator.GetBytes(byteCount));

while (generatedNumber % 2 == 0 ||

generatedNumber % 3 == 0 ||

generatedNumber % 5 == 0 ||

generatedNumber % 7 == 0)

{

generatedNumber = new BigInteger(RandomNumberGenerator.GetBytes(byteCount));

}

generatedNumber \*= generatedNumber.Sign;

textBox14.Text = generatedNumber.ToString();

}

private void button18\_Click(object sender, EventArgs e)

{

textBox10.Text = textBox17.Text;

textBox9.Text = textBox18.Text;

textBox8.Text = textBox15.Text;

}

}

}