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

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.

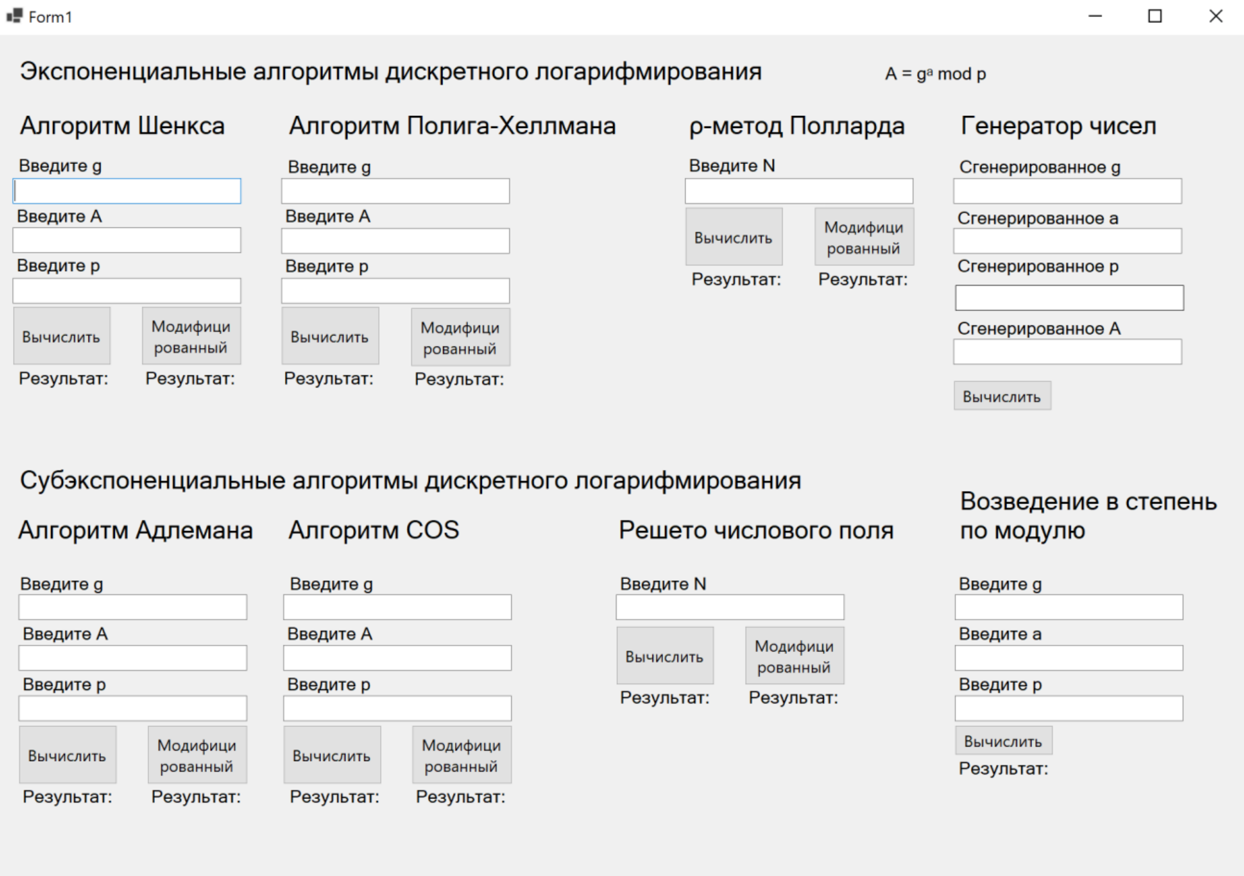

Рисунок 1 - Реализованная программа

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).

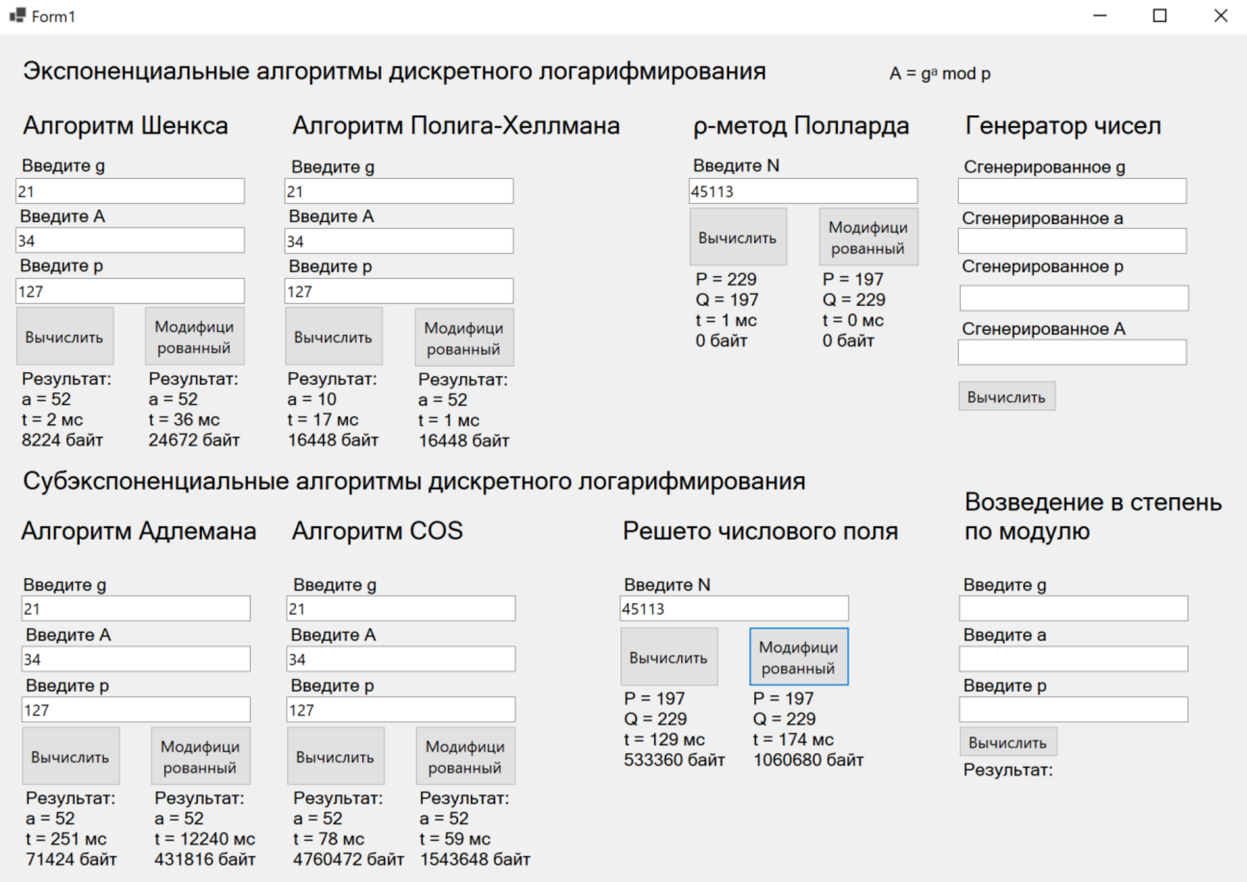

Рисунок 2 - Вычисление модифицированных алгоритмов

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 $a^x \equiv b \ (mod\ p)$, необходимо найти натуральное число x, удовлетворяющее данному сравнению.

Начальный алгоритм реализован следующим образом:

1) Сначала берутся два целых числа m и k, такие, что $mk > p$. Как правило $m = k = \sqrt[2]{p} + 1$.

2) Вычисляются два ряда чисел:

$a^m, a^{2m}, \dots, a^{km}\ mod\ p,$

$b, ba, ba^2,\ \dots, ba^{m-1}\ mod\ p.$

Все вычисления проводятся по модулю p.

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

4) В результате работы алгоритма неизвестная степень вычисляется по формуле $x = im - j$.

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 g, p и A - 16 битные числа, а параметр a - 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 $a^x \equiv b \ (mod \ p)$, необходимо найти натуральное число x, удовлетворяющее данному сравнению.

Шаги выполнения алгоритма:

1) Идёт разложение числа $\varphi(p) = p - 1$ на простые множители.

2) Составляется таблица значений $\{r_{i,j}\}$,

где $r_{i,j} = a^{j*\frac{p-1}{q_i}}, i \in \{1, \dots, k\}, j \in \{0, \dots, q_i - 1\}$.

3) Вычисляется $\log_a b \ \ mod \ q_i^{\alpha_i}$.

Для i от 1 до k:

Пусть $x \equiv \log_a b \equiv x_0 + x_1 q_i + \cdots + x_{\alpha_i - 1} q_i^{\alpha_i - 1} \ (mod \ q_i^{\alpha_i})$,

где $0 \leq x_i \leq q_i - 1$.

Тогда верно сравнение:

$$a^{x_0 * \frac{p-1}{q_i}} \equiv b^{\frac{p-1}{q_i}} \ (mod \ p).$$

С помощью таблицы, составленной на шаге 1, находится x_0.

Для j от 0 до $\alpha_i - 1$ рассматривается сравнение

$$a^{x_j * \frac{p-1}{q_i}} \equiv \left(b a^{-x_0 - x_1 q_i - \cdots - x_{j-1} q_i^{j-1}}\right)^{\frac{p-1}{q_i^{j+1}}} \ (mod \ p).$$

Решение находится по таблице

Конец цикла по j.

Конец цикла по i.

4) Найдя $\log_a b \ mod\ q_i^{\alpha_i}$ для всех i, происходит поиск $\log_a b \ mod\ (p-1)$ по китайской теореме об остатках.

Была реализована модификация алгоритма, состоящая в том, что на 1 шаге алгоритма число $\varphi(p) = p - 1$ 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 g, p и A - 16 битные числа, а параметр a - 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 n, что может быть проиллюстрировано, расположением чисел в виде греческой буквы ρ, что послужило названием семейству алгоритмов.

Шаги выполнения алгоритма:

1) Генерируется случайно число x между 1 и $N-2$.

2) Инициализируются числа $y=0$, $i=0$, $stage=2$.

3) В цикле вычисляется $GCD(N, abs(x-y))$ до тех пор, пока не будет равен 1.

4) Если i равен $stage$, то y присваивается x и $stage$ присваивается $stage * 2$. Далее $x=(x*x+1)(mod\ N)$ и $i=i+1$.

5) После завершения цикла на 3 шаге возвращается результат, равный $GCD(N, abs(x-y))$.

Была реализована модификация алгоритма, состоящая в том, что на 4 шаге алгоритма увеличилась степень вычисляемого $x=(x*x*x-1)\ (mod\ N)$. При вычислении x степень полинома увеличилась до 3.

Был сгенерирован параметр и проведены тесты базового (таблица 5) и модифицированного (таблица 6) алгоритма ро-метод Полларада, где N - 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 $a^x \equiv b \ (mod\ p)$, необходимо найти натуральное число x, удовлетворяющее данному сравнению.

Описание алгоритма:

1) Сформировывается факторная база, состоящая из всех простых чисел q: $q \leq B = e^{const\sqrt{\log p \log\log p}}$.

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

$a^{r_i} \equiv \prod_{q \leq B} q^{\alpha_{iq}} \ (mod\ p)$,

то есть $a^{r_i}\ mod\ p$ раскладывается по факторной базе. Отсюда следует, что $r_i \equiv \sum_{q \leq B} \alpha_{iq} \log_a q \ \ (mod\ p-1)$.

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

4) С помощью некоторого перебора ищется одно значение r, для которого $a^r \equiv \prod_{q \leq B} q^{\beta_q} p_1 * \dots * p_k \ (mod\ p)$, где $p_1, \dots, p_k\ mod\ p$ – простые числа «средней» величины, то есть $B < p_i < B_1$, где B_1 – также некоторая субэкспоненциальная граница, $B_1 = e^{const\sqrt{\log p \log\log p}}$.

5) С помощью вычислений, аналогичных этапам 2 и 3 ищутся дискретные логарифмы $\log_a p_i$.

6) Определяется искомый дискретный логарифм:

$$x \equiv \log_a b \equiv \sum_{q \le B} \beta_q \log_a q + \sum_{i=1}^{k} \log_a p_i \ \ (mod\ p-1).$$

Была реализована модификация алгоритма, состоящая в том, что на 1 шаге алгоритма был изменён показатель степени при вычислении числа $q \le B = e^{const\sqrt{\log p \log p}}$, тем самым повысив факторную базу.

Были сгенерированы параметры и проведены тесты базового (таблица 7) и модифицированного (таблица 8) алгоритма Адлемана, где g, p и A - 16 битные числа, а параметр a - 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 $a^x \equiv b \ (mod\ p)$, необходимо найти натуральное число x, удовлетворяющее данному сравнению.

Описание алгоритма:

1) Задаётся $H = \left[p^{1/2}\right] + 1, J = H^2 - p > 0$. Сформировывается множество $S = \left\{q \mid q - \text{простое}, q < L^{1/2}\right\} \cup \left\{H + c \mid 0 < c < L^{\frac{1}{2}+\varepsilon}\right\}$, где L и ε – простые величины, $L = L_p\left[\frac{1}{2}; 1\right], 0 < \varepsilon < 1$.

2) С помощью некоторого просеивания идёт поиск пары целых чисел c_1, c_2 таких, что $0 < c_i < L^{\frac{1}{2}+\varepsilon}, i = 1{,}2$, и абсолютно наименьший вычет элемента $(H + c_1)(H + c_2)\ (mod\ p)$ гладок по отношению к границе гладкости $L^{1/2}$, т.е.

$$(H + c_1)(H + c_2) \equiv \prod_{\substack{q < L^{\frac{1}{2}}, \\ q - \text{простое}}} q^{\alpha_q(c_1, c_2)}\ (mod\ p).$$

При этом, поскольку $J = O\left(p^{1/2}\right)$, то

$(H + c_1)(H + c_2) \equiv J + (c_1 + c_2)H + c_1c_2\ (mod\ p)$, причём абсолютно наименьший вычет в этом классе вычетов равен $J + (c_1 + c_2)H + c_1c_2$ и имеет величину $O\left(p^{\frac{1}{2}+\varepsilon}\right)$. Поэтому вероятность его гладкости выше, чем для произвольных чисел на отрезке $[1, p - 1]$. Логарифмируя по основанию a, получается соотношение

$$\log_a(H + c_1) + \log_a(H + c_2) \equiv \sum_{\substack{q < L^{\frac{1}{2}}, \\ q - \text{простое}}} \alpha_q(c_1, c_2) \log_a q\ \ (mod\ p - 1).$$

Это однородное уравнение относительно неизвестных величин $\log_a(H + c), \log_a q$. Можно считать, что a также является $L^{1/2}$ – гладким, $a = \prod_{q < L^{1/2}} q^{\beta_q}$, откуда получим неоднородное уравнение

$$1 \equiv \sum_q \beta_q \log_a q\ \ (mod\ p - 1).$$

3) Набрав на 2-м этапе достаточно много уравнений, решается получившаяся система линейных уравнений в кольце $\mathbb{Z}/(p-1)\mathbb{Z}$ и находятся значения $\log_a(H+c)$, $\log_a q$.

4) Для нахождения конкретного логарифма $x = \log_a b$ мы введём новую границу гладкости L^2. Случайным перебором находим одно ω значение такое, что

$$a^{\omega} b \equiv \prod_{\substack{q < L^{\frac{1}{2}}, \\ q - \text{простое}}} q^{g_q} \prod_{\substack{L^{\frac{1}{2}} \le u < L^2, \\ u - \text{простое}}} u^{h_u} \pmod p.$$

В этом соотношении участвуют несколько новых простых чисел u средней величины.

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

6) Находим ответ

$$x = \log_a b \equiv -\omega + \sum_{\substack{q < L^{\frac{1}{2}}, \\ q - \text{простое}}} g_q \log_a q + \sum_u h_u \log_a u \pmod{p-1}.$$

Конец алгоритма.

Была реализована модификация алгоритма, состоящая в том, что на 2 шаге был увеличен наименьший вычет, добавив значение $(H + c_3)$, чтобы увеличить разложение чисел при формировании СЛАУ.

Были сгенерированы параметры и проведены тесты базового (таблица 9) и модифицированного (таблица 10) алгоритма COS, где g, p и A - 16 битные числа, а параметр a - 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 n — нечетное составное число, которое требуется факторизовать.

2) Выберем степень неприводимого многочлена $d \geq 3$ (при $d = 2$ не будет выигрыша в сравнении с методом квадратичного решета).

3) Выберем целое m такое, что $\left[n^{1/(d+1)}\right] < m < \left[n^{1/d}\right]$, и разложим n по основанию m:

$$n = m^d + a_{d-1}m^{d-1} + \cdots + a_0.$$

4) Свяжем с разложением из 3 шага неприводимый в кольце $\mathbb{Z}[x]$ полиномов с целыми коэффициентами многочлен

$$f_1(a,b) = b^d * f_1\left(\frac{a}{b}\right) = a^d + a_{d-1}a^{d-1}b + a_{d-2}a^{d-2}b^2 + \cdots + a_0 b^d.$$

5) Определим полином просеивания $F_1(a,b)$ как однородный многочлен от двух переменных a и b:

$$F_1(a,b) = b^d * f_1\left(\frac{a}{b}\right) = a^d + a_{d-1}a^{d-1}b + a_{d-2}a^{d-2}b^2 + \cdots + a_0 b^d.$$

6) Определим второй полином $f_2 = x - m$ и соответствующий однородный многочлен $F_2(a,b) = a - bm$.

7) Выберем два положительных числа L_1 и L_2, определяющих область просеивания:

$$SR = \{1 \leq b \leq L_1, -L_1 \leq a \leq L_2\}.$$

8) Пусть θ — корень $f_1(x)$. Рассмотрим кольцо полиномов $\mathbb{Z}[\theta]$. Определим множество, называемое алгебраической факторной базой FB_1, состоящее из многочленов первого порядка вида $a - b\theta$ с нормой шага 5, являющейся простым числом. Эти многочлены — простые неразложимые в кольце алгебраических целых поля $K = \mathbb{Q}[\theta]$. Ограничим абсолютные значения норм полиномов из FB_1 константой B_1.

9) Определим рациональную факторную базу FB_2, состоящую из всех простых чисел, ограниченных сверху константой B_2.

10) Определим множество FB_3, называемое факторной базой квадратичных характеров. Это множество полиномов первого порядка $c - d\theta$, норма которых - простое число. Должно выполняться условие $FB_1 \cap FB_3 = \emptyset$.

11) Выполним просеивание многочленов $\{a - b\theta | (a, b) \in SR\}$ по факторной базе FB_1 и целых чисел $\{a - bm | (a, b) \in SR\}$ по факторной базе FB_2. В результате получим множество M, состоящее из гладких пар (a, b), то есть таких пар (a, b), что НОД(a, b)= 1, полином и число $a - b\theta$ и $a - bm$ раскладываются полностью по FB_1 и FB_2 соответственно.

12) Найдём такое подмножество $S \subseteq M$, что

$\prod_{(a,b)\in S} Nr(a - b\theta) = H^2, H \in \mathbb{Z}$; $\prod_{(a,b)\in S}(a - bm) = B^2, B \in \mathbb{Z}$.

13) Определим многочлен

$g(\theta) = {f'}_1^2(\theta) * \prod_{(a,b)\in S}(a - bm)$, где ${f'}_1^2(x)$ – производная $f_1(x)$.

14) Многочлен $g(\theta)$ является полным квадратом в кольце полиномов $\mathbb{Z}(\theta)$. Пусть тогда $\alpha(\theta)$ есть корень из $g(\theta)$ и B — корень из B^2.

15) Строим отображение $\phi: \theta \to m$, заменяя полином $\alpha(\theta)$ числом $\alpha(m)$. Это отображение является кольцевым гомоморфизмом кольца алгебраических целых чисел $\mathbb{Z}_K$ в кольцо $\mathbb{Z}$, откуда получаем соотношение:

$$A^2 = g(m)^2 \equiv \phi\big(g(\alpha)\big)^2 \equiv \phi\big({f'}_1^2(\theta) * \prod_{(a,b)\in S}(a - b\theta)\big) \equiv {f'}_1^2(m) * \prod_{(a,b)\in S}(a - bm) \equiv {f'}_1^2(m) * C^2 \ (mod\ n).$$

16) Пусть $B = f'(m) * C$. Найдём пару чисел (A, B) таких, что $A \equiv B\ (mod\ n)$. Тогда найдём делитель числа n, вычисляя НОД$(n, A \pm B)$.

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

Был сгенерирован параметр и проведены тесты базового (таблица 11) и модифицированного (таблица 12) алгоритма решета числового поля, где N - 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. — С. 328.

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

10) Методы факторизации натуральных чисел. / Ишмухаметов Ш. Т. - Текст: непосредственный // — Казань: Казан. ун.. — 2011. — С. 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;
        }
    }
}
```