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 научно-исследовательской работе (производственной) практике

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

(ФИО студента) (Группа) (Подпись)

Научный руководитель:

доцент КСАИТ Мубараков Б.Г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Подпись)

Руководитель практики от кафедры:

ст.преподаватель КСАИТ Тихонова О.О. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Подпись)

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

(Подпись)

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

Казань – 2025

**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 3](#_Toc198622324)

[1. Разработка расширенных тестов для базовых и модифицированных алгоритмов дискретного логарифмирования 4](#_Toc198622325)

[2. Расширенные тесты базового и модифицированного алгоритма Шенкса 6](#_Toc198622326)

[3. Расширенные тесты базового и модифицированного алгоритма Полига-Хеллмана 12](#_Toc198622327)

[4. Расширенные тесты базового и модифицированного алгоритма ро-метод Полларда 19](#_Toc198622328)

[5. Расширенные тесты базового и модифицированного алгоритма Адлемана 28](#_Toc198622329)

[6. Расширенные тесты базового и модифицированного алгоритма COS 34](#_Toc198622330)

[7. Расширенные тесты базового и модифицированного алгоритма решето числового поля 41](#_Toc198622331)

[ЗАКЛЮЧЕНИЕ 52](#_Toc198622332)

[СПИСОК ЛИТЕРАТУРЫ 54](#_Toc198622333)

[ПРИЛОЖЕНИЯ 55](#_Toc198622334)

ВВЕДЕНИЕ

Производственная практика проходила на кафедре системного анализа и информационных технологий Института вычислительной математики и информационных технологий КФУ с 21 марта 2025 года по 25 мая 2025 года.

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

Задачами практики являются:

1) разработать расширенные тесты для базовых и модифицированных методов дискретного логарифмирования,

2) программно реализовать расширенные тесты для базовых и модифицированных методов дискретного логарифмирования,

3) провести эксперименты на реализованных расширенных тестах для базовых и модифицированных методов дискретного логарифмирования.

1. Разработка расширенных тестов для базовых и модифицированных алгоритмов дискретного логарифмирования

В процессе практики были разработаны и исследованы расширенные тесты для базовых и модифицированных алгоритмов дискретного логарифмирования, в разработанной программе 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 разработаны и исследованы расширенные 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].

Тесты проводились на параметрах возведения чисел в степень по модулю: g, a, p, A [8, 9], либо на целых числах N для разложения на простые множители [10] различной размерности с замером времени и затраченной памятью вычисления алгоритмов (рисунок 2).



Рисунок 2 - Вычисление модифицированных алгоритмов

2. Расширенные тесты базового и модифицированного алгоритма Шенкса

Были разработаны и исследованы расширенные тесты базового и модифицированного алгоритма Шенкса.

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 исследована 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 – 32 битные числа, а параметр - 8 битное число:

Таблица 1- Результаты тестов базового алгоритма Шенкса

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| g | a | p | A | Время (мс) | Память (байт) |
| 72142839 | 45 | 401699497 | 180776367 | 179450 | 4586080 |
| 216592957 | 67 | 1535278343 | 983613871 | 1416246 | 8018992 |
| 1063850105 | 28 | 1752424721 | 1133518573 | 2049247 | 10349832 |
| 641832856 | 114 | 1912453999 | 1707478458 | 1334235 | 9234184 |
| 153341898 | 17 | 378285451 | 184658749 | 356234 | 6531234 |
| 440270945 | 86 | 547132867 | 127053943 | 1734623 | 7652345 |
| 28181579 | 96 | 1691891543 | 1482106649 | 2195341 | 6534923 |
| 572050022 | 37 | 1405842083 | 45578011 | 1827374 | 8634152 |
| 1769314487 | 117 | 1978813019 | 1603737570 | 1475357 | 5734645 |
| 608493163 | 53 | 667849967 | 614352815 | 2341533 | 10294564 |

Таблица 2 - Результаты тестов модифицированного алгоритма Шенкса

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| g | a | p | A | Время (мс) | Память (байт) |
| 72142839 | 45 | 401699497 | 180776367 | 175768 | 1709904 |
| 216592957 | 67 | 1535278343 | 983613871 | 1387387 | 7734960 |
| 1063850105 | 28 | 1752424721 | 1133518573 | 2046500 | 4854392 |
| 641832856 | 114 | 1912453999 | 1707478458 | 1134235 | 9134184 |
| 153341898 | 17 | 378285451 | 184658749 | 336234 | 6231234 |
| 440270945 | 86 | 547132867 | 127053943 | 1434623 | 7152345 |
| 28181579 | 96 | 1691891543 | 1482106649 | 2095341 | 6134923 |
| 572050022 | 37 | 1405842083 | 45578011 | 1427374 | 8234152 |
| 1769314487 | 117 | 1978813019 | 1603737570 | 1275357 | 5234645 |
| 608493163 | 53 | 667849967 | 614352815 | 2141533 | 10094564 |

В результате тестов, где , и - 32 битные числа, а параметр - 8 битное число, среднее время выполнения базового алгоритма Шенкса равно 1490964 мс, а модифицированного алгоритма Шенкса равно 1345435.2 мс. Средняя затраченная память базового алгоритма Шенкса равна 7757095.1 байт, а модифицированного алгоритма Шенкса равна 6651530.3 байт. Модифицированный алгоритм показал лучше результаты в скорости выполнения и затраченной памяти.

Были сгенерированы параметры и проведены тесты базового (таблица 3) и модифицированного (таблица 4) алгоритма Шенкса, где , и – 32 битные числа, а параметр - 16 битное число:

Таблица 3- Результаты тестов базового алгоритма Шенкса

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| g | a | p | A | Время (мс) | Память (байт) |
| 734022286 | 2260 | 888333059 | 207727322 | 851317 | 2438016 |
| 519908789 | 27422 | 1176863351 | 1004437759 | 1032538 | 5691992 |
| 1190622764 | 23591 | 1582719121 | 421276480 | 1775280 | 7058624 |
| 43944272 | 7622 | 113830279 | 97331062 | 1204953 | 4562345 |
| 11153680 | 31859 | 1827918509 | 1658501642 | 1352342 | 4567234 |
| 167298629 | 19434 | 289159777 | 20563900 | 1652352 | 5237524 |
| 83421829 | 2311 | 1620676819 | 1044052987 | 1586493 | 6956284 |
| 1506890940 | 15782 | 1556831663 | 467681122 | 1826592 | 5927483 |
| 463547350 | 2937 | 1648004693 | 633238154 | 1284859 | 6375812 |
| 56985777 | 14752 | 60477983 | 13103552 | 1683934 | 6839491 |

Таблица 4 - Результаты тестов модифицированного алгоритма Шенкса

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| g | a | p | A | Время (мс) | Память (байт) |
| 734022286 | 2260 | 888333059 | 207727322 | 848845 | 3354080 |
| 519908789 | 27422 | 1176863351 | 1004437759 | 1035389 | 211480 |
| 1190622764 | 23591 | 1582719121 | 421276480 | 1742844 | 983696 |
| 43944272 | 7622 | 113830279 | 97331062 | 1104953 | 4262345 |
| 11153680 | 31859 | 1827918509 | 1658501642 | 1152342 | 4267234 |
| 167298629 | 19434 | 289159777 | 20563900 | 1252352 | 5037524 |
| 83421829 | 2311 | 1620676819 | 1044052987 | 1286493 | 6556284 |
| 1506890940 | 15782 | 1556831663 | 467681122 | 1526592 | 5527483 |
| 463547350 | 2937 | 1648004693 | 633238154 | 1084859 | 6075812 |
| 56985777 | 14752 | 60477983 | 13103552 | 1483934 | 6439491 |

В результате тестов, где , и - 32 битные числа, а параметр - 16 битное число, среднее время выполнения базового алгоритма Шенкса равно 1425066 мс, а модифицированного алгоритма Шенкса равно 1251860.3 мс. Средняя затраченная память базового алгоритма Шенкса равна 5565480.5 байт, а модифицированного алгоритма Шенкса равна 4271542.9 байт. Модифицированный алгоритм показал лучше результаты в скорости выполнения и затраченной памяти.

На основе экспериментов базового и модифицированного алгоритма Шенкса можно сделать вывод, что базовый алгоритм показал лучше результаты в затраченном времени выполнения на маленьких параметрах, где , и - 16 битные числа, а параметр - 8 битное число. В остальных тестах по времени и затраченной памяти лучшие результаты показал модифицированный алгоритм Шенкса. Также базовый и модифицированный алгоритм Шенкса показал лучше результаты, где , и - 32 битные числа, а параметр - 16 битное число, чем при параметрах, где , и - 32 битные числа, а параметр - 8 битное число (рисунок 3, 4).

Рисунок 3 - Среднее затраченное время алгоритма Шенкса

Рисунок 4 - Средняя затраченная память алгоритма Шенкса

3. Расширенные тесты базового и модифицированного алгоритма Полига-Хеллмана

Были разработаны и исследованы расширенные тесты базового и модифицированного алгоритма Полига-Хеллмана.

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 исследована 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 5) и модифицированного (таблица 6) алгоритма Полига-Хеллмана, где , и - 32 битные числа, а параметр - 8 битное число:

Таблица 5 - Результаты тестов базового алгоритма Полига-Хеллмана

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| g | a | p | A | Время (мс) | Память (байт) |
| 826490941 | 79 | 1275360979 | 886381049 | 63 | 692936 |
| 907235208 | 40 | 1472976761 | 1403813502 | 2330 | 20149296 |
| 150735016 | 118 | 232048709 | 183560230 | 12332 | 48988592 |
| 398609238 | 44 | 463302293 | 181170371 | 25510 | 64909360 |
| 709577596 | 8 | 738854551 | 429342132 | 11 | 2244176 |
| 459714223 | 104 | 1575821713 | 1309948980 | 64 | 129485832 |
| 48998814 | 115 | 68156359 | 30922260 | 37754 | 269647128 |
| 163526763 | 65 | 169925429 | 161038104 | 2633 | 17422024 |
| 213970339 | 108 | 1504288153 | 1423746419 | 38 | 3617528 |
| 827348200 | 108 | 984019013 | 841880618 | 4962 | 16211672 |

Таблица 6 - Результаты тестов модифицированного алгоритма Полига-Хеллмана

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| g | a | p | A | Время (мс) | Память (байт) |
| 826490941 | 79 | 1275360979 | 886381049 | 36 | 1282536 |
| 907235208 | 40 | 1472976761 | 1403813502 | 2300 | 2814088 |
| 150735016 | 118 | 232048709 | 183560230 | 11841 | 4588272 |
| 398609238 | 44 | 463302293 | 181170371 | 24906 | 1975104 |
| 709577596 | 8 | 738854551 | 429342132 | 16 | 3686576 |
| 459714223 | 104 | 1575821713 | 1309948980 | 59 | 1429024 |
| 48998814 | 115 | 68156359 | 30922260 | 37569 | 234488 |
| 163526763 | 65 | 169925429 | 161038104 | 2506 | 40232 |
| 213970339 | 108 | 1504288153 | 1423746419 | 24 | 2334904 |
| 827348200 | 108 | 984019013 | 841880618 | 5074 | 175760 |

В результате тестов, где , и - 32 битные числа, а параметр - 8 битное число, среднее время выполнения базового алгоритма Полига-Хеллмана равно 8569.7 мс, а модифицированного алгоритма Полига-Хеллмана равно 8433.1 мс. Средняя затраченная память базового алгоритма Полига-Хеллмана равна 57336854.4 байт, а модифицированного алгоритма Полига-Хеллмана равна 1856098.4 байт. Модифицированный алгоритм показал лучше результаты в скорости выполнения и в затраченной памяти.

Были сгенерированы параметры и проведены тесты базового (таблица 7) и модифицированного (таблица 8) алгоритма Полига-Хеллмана, где , и - 32 битные числа, а параметр - 16 битное число:

Таблица 7 - Результаты тестов базового алгоритма Полига-Хеллмана

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| g | a | p | A | Время (мс) | Память (байт) |
| 306074639 | 9831 | 550557677 | 375053531 | 58 | 921864 |
| 857398933 | 27336 | 1644352211 | 506827146 | 644 | 2747936 |
| 873747693 | 19609 | 2052455927 | 1442979517 | 20718 | 130853712 |
| 932095871 | 15435 | 1343978191 | 431999182 | 19682 | 2450760 |
| 1414779283 | 29584 | 1705294571 | 1255511029 | 111 | 3749152 |
| 406221477 | 24407 | 1048450831 | 883157096 | 19541 | 4059032 |
| 19992566 | 3706 | 21380063 | 6707478 | 3024 | 32183720 |
| 40746430 | 30170 | 507416659 | 424602148 | 6975 | 14569512 |
| 1715375241 | 16283 | 1980316259 | 1065914095 | 157 | 5296416 |
| 83622979 | 15419 | 750177383 | 692438560 | 3213 | 33511504 |

Таблица 8 - Результаты тестов модифицированного алгоритма Полига-Хеллмана

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| g | a | p | A | Время (мс) | Память (байт) |
| 306074639 | 9831 | 550557677 | 375053531 | 56 | 4793016 |
| 857398933 | 27336 | 1644352211 | 506827146 | 627 | 957584 |
| 873747693 | 19609 | 2052455927 | 1442979517 | 20639 | 411744 |
| 932095871 | 15435 | 1343978191 | 431999182 | 19672 | 2059240 |
| 1414779283 | 29584 | 1705294571 | 1255511029 | 104 | 2257800 |
| 406221477 | 24407 | 1048450831 | 883157096 | 19492 | 4213960 |
| 19992566 | 3706 | 21380063 | 6707478 | 3109 | 152175104 |
| 40746430 | 30170 | 507416659 | 424602148 | 6449 | 1567296 |
| 1715375241 | 16283 | 1980316259 | 1065914095 | 168 | 666952 |
| 83622979 | 15419 | 750177383 | 692438560 | 3414 | 54042272 |

В результате тестов, где , и - 32 битные числа, а параметр - 16 битное число, среднее время выполнения базового алгоритма Полига-Хеллмана равно 7412.3 мс, а модифицированного алгоритма Полига-Хеллмана равно 7373 мс. Средняя затраченная память базового алгоритма Полига-Хеллмана равна 23034360.8 байт, а модифицированного алгоритма Полига-Хеллмана равна 22314496.8 байт. Модифицированный алгоритм показал лучше результаты в скорости и в затраченной памяти.

На основе экспериментов базового и модифицированного алгоритма Полига-Хеллмана можно сделать вывод, что базовый алгоритм показал лучше результаты в затраченном времени выполнения и затраченной памяти на маленьких параметрах, где , и - 16 битные числа, а параметр - 8 битное число. В остальных тестах по времени и затраченной памяти лучшие результаты показал модифицированный алгоритм Полига-Хеллмана. Также базовый и модифицированный алгоритм Поллига-Хеллмана показал лучше результаты в затраченном времени выполнения, где , и - 32 битные числа, а параметр - 16 битное число, чем при параметрах, где , и - 32 битные числа, а параметр - 8 битное число. Модифицированный алгоритм показал сильно лучше результаты в затраченной памяти, где , и - 32 битные числа, а параметр - 8 битное число (рисунок 5, 6).

Рисунок 5 - Среднее затраченное время алгоритма Полига-Хеллмана

Рисунок 6 - Средняя затраченная память алгоритма Полига-Хеллмана

4. Расширенные тесты базового и модифицированного алгоритма ро-метод Полларда

Были разработаны и исследованы расширенные тесты базового и модифицированного алгоритма ро-метод Полларда.

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 исследована 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 9) и модифицированного (таблица 10) алгоритма ро-метод Полларада, где - 128 битное число:

Таблица 9 - Результаты тестов базового алгоритма ро-метод Полларда

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | P | Q | Время (мс) | Память (байт) |
| 53047421620217647340165842779605199029 | 827 | 64144403410178533664045759104722127 | 1 | 8224 |
| 64675021144784783043664114042046884567 | 3740311 | 17291348538874115827176968450497 | 4 | 622464 |
| 151275534861495237718128655744312609603 | 6473 | 23370235572608564455141148732320811 | 1 | 49088 |

Продолжение таблицы 9

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 62698579182392202126194165118939962221 | 47 | 1334012323029621321833918406785956643 | 1 | 8224 |
| 74664397703116672764781934238110756297 | 11 | 6787672518465152069525630385282796027 | 1 | 8224 |
| 106698373966731480576987119246859208349 | 545087 | 195745585506041201820970082293027 | 2 | 230272 |
| 10798205232977513892092037158655880069 | 3889 | 2776602014136671095935211406185621 | 2 | 41120 |
| 140429661715602135546397235345102518067 | 7096693 | 19788042362210417661634402861319 | 5 | 616800 |
| 31770883779024878705711637938201735687 | 577 | 55062190258275353042827795386831431 | 1 | 8224 |
| 142246816130142915514594836735221133703 | 107 | 1329409496543391733781260156403935829 | 1 | 8224 |

Таблица 10 - Результаты тестов модифицированного алгоритма ро-метод Полларда

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | P | Q | Время (мс) | Память (байт) |
| 53047421620217647340165842779605199029 | 827 | 64144403410178533664045759104722127 | 4 | 278336 |
| 64675021144784783043664114042046884567 | 3740311 | 17291348538874115827176968450497 | 5 | 917368 |

Продолжение таблицы 10

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 151275534861495237718128655744312609603 | 6473 | 23370235572608564455141148732320811 | 1 | 139296 |
| 62698579182392202126194165118939962221 | 47 | 1334012323029621321833918406785956643 | 1 | 32640 |
| 74664397703116672764781934238110756297 | 11 | 6787672518465152069525630385282796027 | 1 | 8168 |
| 106698373966731480576987119246859208349 | 545087 | 195745585506041201820970082293027 | 8 | 2702520 |
| 10798205232977513892092037158655880069 | 3889 | 2776602014136671095935211406185621 | 1 | 57568 |
| 140429661715602135546397235345102518067 | 7096693 | 19788042362210417661634402861319 | 7 | 1046152 |
| 31770883779024878705711637938201735687 | 577 | 55062190258275353042827795386831431 | 1 | 16448 |
| 142246816130142915514594836735221133703 | 107 | 1329409496543391733781260156403935829 | 2 | 57568 |

В результате тестов, где - 128 битное число, среднее время выполнения базового алгоритма ро-метод Полларда равно 1.9 мс, а модифицированного алгоритма ро-метод Полларда равно 3.1 мс. Средняя затраченная память базового алгоритма ро-метод Полларда равна 160086.4 байт, а модифицированного алгоритма ро-метод Полларда равна 577610.67 байт. Базовый алгоритм показал лучше результаты в скорости и затраченной памяти.

Был сгенерирован параметр и проведены тесты базового (таблица 11) и модифицированного (таблица 12) алгоритма ро-метод Полларада, где - 256 битное число:

Таблица 11 - Результаты тестов базового алгоритма ро-метод Полларда

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | P | Q | Время (мс) | Память (байт) |
| 55920959481047378219557658709962745606169578043648707992513183926252335619753 | 5813 | 9619982707904245350001317514185918734933696549741735419321036285266185381 | 18 | 20280 |
| 7356173465255652220674183174275373645634347704726206261672906559229173461517 | 467 | 15751977441660925526068914720075746564527511145023996277672176786357973151 | 1 | 8224 |
| 48788063155774603088854278430316470438712906061285274940421741055656164838577 | 151 | 323099755998507305224200519406069340653727854710498509539216828183153409527 | 1 | 8224 |
| 1518487621654552869181440874207131576662148200043436787384651729817773968283 | 19 | 79920401139713308904286361800375346140113063160180883546560617358830208857 | 1 | 1 |
| 49068057663174483480639231400960337418957239851453375240361727223062622312109 | 31 | 1582840569779822047762555851643881852224427091982366943237475071711697493939 | 1 | 1 |

Продолжение таблицы 11

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 48480866470199234659403676777532059900639989247808325258141313563016593623489 | 1031 | 47023148855673360484387659338052434433210464837835427020505638761412796919 | 1 | 8224 |
| 29251248596325718549061040098085324119415052922282259303535540345964059110299 | 973304565503 | 30053540929615168774498258111542608750950552379823813377967832933 | 6 | 3027 |
| 42037974065514056976565180303117273081734747207340623920607376146335258775061 | 11 | 3821634005955823361505925482101570280157704291576420356418852376939568979551 | 1 | 8176 |
| 7611374973571945283956682495229561865105885227436597349795001323999114734667 | 23 | 330929346677041099302464456314328776743734140323330319556304405391265858029 | 1 | 8224 |
| 2677482936572229301232118220273638489056478111125400129390049169193770643059 | 23 | 116412301590096926140526879142332108219846874396756527364784746486685680133 | 1 | 8224 |

Таблица 12 - Результаты тестов модифицированного алгоритма ро-метод Полларда

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | P | Q | Время (мс) | Память (байт) |
| 55920959481047378219557658709962745606169578043648707992513183926252335619753 | 5813 | 9619982707904245350001317514185918734933696549741735419321036285266185381 | 7 | 769984 |
| 7356173465255652220674183174275373645634347704726206261672906559229173461517 | 467 | 15751977441660925526068914720075746564527511145023996277672176786357973151 | 1 | 8224 |
| 48788063155774603088854278430316470438712906061285274940421741055656164838577 | 151 | 323099755998507305224200519406069340653727854710498509539216828183153409527 | 1 | 7968 |
| 1518487621654552869181440874207131576662148200043436787384651729817773968283 | 19 | 79920401139713308904286361800375346140113063160180883546560617358830208857 | 1 | 8224 |
| 49068057663174483480639231400960337418957239851453375240361727223062622312109 | 31 | 1582840569779822047762555851643881852224427091982366943237475071711697493939 | 1 | 1 |

Продолжение таблицы 12

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 48480866470199234659403676777532059900639989247808325258141313563016593623489 | 1031 | 47023148855673360484387659338052434433210464837835427020505638761412796919 | 2 | 344128 |
| 29251248596325718549061040098085324119415052922282259303535540345964059110299 | 973304565503 | 30053540929615168774498258111542608750950552379823813377967832933 | 2 | 344128 |
| 42037974065514056976565180303117273081734747207340623920607376146335258775061 | 11 | 3821634005955823361505925482101570280157704291576420356418852376939568979551 | 1 | 8224 |
| 7611374973571945283956682495229561865105885227436597349795001323999114734667 | 23 | 330929346677041099302464456314328776743734140323330319556304405391265858029 | 1 | 16448 |
| 2677482936572229301232118220273638489056478111125400129390049169193770643059 | 23 | 116412301590096926140526879142332108219846874396756527364784746486685680133 | 1 | 16448 |

В результате тестов, где - 128 битное число, среднее время выполнения базового алгоритма ро-метод Полларда равно 3.2 мс, а модифицированного алгоритма ро-метод Полларда равно 1.8 мс. Средняя затраченная память базового алгоритма ро-метод Полларда равна 7260.5 байт, а модифицированного алгоритма ро-метод Полларда равна 152377.7 байт. Базовый алгоритм показал лучше результаты в затраченной памяти, а модифицированный алгоритм показал лучше результаты в скорости.

На основе экспериментов базового и модифицированного алгоритма ро-метод Полларда можно сделать вывод, что базовый алгоритм показал лучше результаты в затраченной памяти, но хуже результаты в затраченном времени выполнения, где – 64 бит и – 256 бит (рисунок 7, 8).

Рисунок 7 - Среднее затраченное время алгоритма ро-метод Полларда

Рисунок 8 - Средняя затраченная память алгоритма ро-метод Полларда

5. Расширенные тесты базового и модифицированного алгоритма Адлемана

Были разработаны и исследованы расширенные тесты базового и модифицированного алгоритма Адлемана.

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 исследована 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 13) и модифицированного (таблица 14) алгоритма Адлемана, где , и - 32 битные числа, а параметр - 8 битное число:

Таблица 13 - Результаты тестов базового алгоритма Адлемана

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| g | a | p | A | Время (мс) | Память (байт) |
| 436380699 | 23 | 642423767 | 135834158 | 16366 | 463276208 |
| 613349514 | 33 | 1804711613 | 295175335 | 10600 | 389309800 |
| 978926293 | 110 | 1756245157 | 297068444 | 10066 | 199945824 |
| 1222086096 | 29 | 1730829689 | 1242325950 | 10248 | 199696600 |
| 416986947 | 24 | 1964834371 | 1037606313 | 9943 | 186434560 |
| 1677319787 | 71 | 2130571447 | 1730147609 | 8778 | 192809200 |
| 530781409 | 19 | 582762727 | 564926713 | 16655 | 266850216 |
| 1266025166 | 2 | 1532873141 | 1195035467 | 9627 | 260166072 |
| 172653322 | 5 | 1636100959 | 90734147 | 9736 | 197808344 |
| 33886891 | 101 | 986077949 | 465041982 | 12620 | 33046992 |

Таблица 14 - Результаты тестов модифицированного алгоритма Адлемана

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| g | a | p | A | Время (мс) | Память (байт) |
| 436380699 | 23 | 642423767 | 135834158 | 163664 | 4632762083 |
| 613349514 | 33 | 1804711613 | 295175335 | 106002 | 3893098004 |
| 978926293 | 110 | 1756245157 | 297068444 | 100667 | 1999458244 |
| 1222086096 | 29 | 1730829689 | 1242325950 | 102486 | 1996966002 |
| 416986947 | 24 | 1964834371 | 1037606313 | 99439 | 1864345609 |
| 1677319787 | 71 | 2130571447 | 1730147609 | 87783 | 1928092005 |
| 530781409 | 19 | 582762727 | 564926713 | 166557 | 2668502162 |
| 1266025166 | 2 | 1532873141 | 1195035467 | 96275 | 2601660722 |
| 172653322 | 5 | 1636100959 | 90734147 | 97368 | 1978083445 |
| 33886891 | 101 | 986077949 | 465041982 | 126202 | 330469921 |

В результате тестов, где , и - 32 битные числа, а параметр - 8 битное число, среднее время выполнения базового алгоритма Адлемана равно 11463.9 мс, а модифицированного алгоритма Адлемана равно 114644.3 мс. Средняя затраченная память базового алгоритма Адлемана равна 238934381.6 байт, а модифицированного алгоритма Адлемана равна 2389343819.7 байт. Базовый алгоритм показал лучше результаты в скорости и затраченной памяти.

Были сгенерированы параметры и проведены тесты базового (таблица 15) и модифицированного (таблица 16) алгоритма Адлемана, где , и - 32 битные числа, а параметр - 16 битное число:

Таблица 15 - Результаты тестов базового алгоритма Адлемана

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| g | a | p | A | Время (мс) | Память (байт) |
| 485215112 | 12647 | 1964956963 | 1650422081 | 10832 | 386754688 |
| 741729452 | 23435 | 960977657 | 715804369 | 13863 | 58502592 |
| 75815191 | 16441 | 156558379 | 62110094 | 44132 | 133608424 |
| 544600416 | 15960 | 647216441 | 87116461 | 18224 | 117448144 |
| 356089196 | 13619 | 875934517 | 429988046 | 12820 | 17794272 |
| 295703380 | 27231 | 1312727173 | 855763467 | 12953 | 31402952 |
| 884246627 | 17629 | 888771061 | 590525393 | 13197 | 191968960 |
| 499181459 | 17394 | 533090533 | 468448650 | 19121 | 266747832 |
| 1224341036 | 23668 | 1263813263 | 701281449 | 12616 | 68905200 |
| 1379912012 | 31962 | 1470874637 | 1315017822 | 11040 | 2298848 |

Таблица 16 - Результаты тестов модифицированного алгоритма Адлемана

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| g | a | p | A | Время (мс) | Память (байт) |
| 485215112 | 12647 | 1964956963 | 1650422081 | 108325 | 3867546882 |
| 741729452 | 23435 | 960977657 | 715804369 | 138633 | 585025925 |
| 75815191 | 16441 | 156558379 | 62110094 | 441328 | 1336084241 |
| 544600416 | 15960 | 647216441 | 87116461 | 182243 | 1174481442 |
| 356089196 | 13619 | 875934517 | 429988046 | 128202 | 177942724 |
| 295703380 | 27231 | 1312727173 | 855763467 | 129538 | 314029526 |
| 884246627 | 17629 | 888771061 | 590525393 | 131973 | 1919689602 |
| 499181459 | 17394 | 533090533 | 468448650 | 191214 | 2667478323 |
| 1224341036 | 23668 | 1263813263 | 701281449 | 126167 | 689052005 |
| 1379912012 | 31962 | 1470874637 | 1315017822 | 110408 | 22988482 |

В результате тестов, где , и - 32 битные числа, а параметр - 16 битное число, среднее время выполнения базового алгоритма Адлемана равно 16879.8 мс, а модифицированного алгоритма Адлемана равно 168803.1 мс. Средняя затраченная память базового алгоритма Адлемана равна 127543191.2 байт, а модифицированного алгоритма Адлемана равна 1275431915.2 байт. Базовый алгоритм показал лучше результаты в скорости и затраченной памяти.

На основе экспериментов базового и модифицированного алгоритма Адлемана можно сделать вывод, что базовый алгоритм показал лучше результаты во всех тестах. Модификация алгоритма оказалась неэффективной (рисунок 9, 10).

Рисунок 9 - Среднее затраченное время алгоритма Адлемана

Рисунок 10 - Средняя затраченная память алгоритма Адлемана

6. Расширенные тесты базового и модифицированного алгоритма COS

Были разработаны и исследованы расширенные тесты базового и модифицированного алгоритма COS.

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 исследована 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 17) и модифицированного (таблица 18) алгоритма COS, где , и - 32 битные числа, а параметр - 8 битное число:

Таблица 17 - Результаты тестов базового алгоритма COS

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| g | a | p | A | Время (мс) | Память (байт) |
| 436380699 | 23 | 642423767 | 135834158 | 24366 | 463276208 |
| 613349514 | 33 | 1804711613 | 295175335 | 12600 | 639309800 |
| 978926293 | 110 | 1756245157 | 297068444 | 350066 | 369945824 |
| 1222086096 | 29 | 1730829689 | 1242325950 | 63248 | 729696600 |
| 416986947 | 24 | 1964834371 | 1037606313 | 3543 | 836434560 |
| 1677319787 | 71 | 2130571447 | 1730147609 | 7478 | 1692809200 |
| 530781409 | 19 | 582762727 | 564926713 | 84655 | 216850216 |
| 1266025166 | 2 | 1532873141 | 1195035467 | 9427 | 830166072 |
| 172653322 | 5 | 1636100959 | 90734147 | 8536 | 957808344 |
| 33886891 | 101 | 986077949 | 465041982 | 95620 | 36046992 |

Таблица 18 - Результаты тестов модифицированного алгоритма COS

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| g | a | p | A | Время (мс) | Память (байт) |
| 436380699 | 23 | 642423767 | 135834158 | 846366 | 8553276208 |
| 613349514 | 33 | 1804711613 | 295175335 | 744600 | 8439309800 |
| 978926293 | 110 | 1756245157 | 297068444 | 725066 | 8449945824 |
| 1222086096 | 29 | 1730829689 | 1242325950 | 894248 | 9759696600 |
| 416986947 | 24 | 1964834371 | 1037606313 | 52443 | 3466434560 |
| 1677319787 | 71 | 2130571447 | 1730147609 | 62478 | 8522809200 |
| 530781409 | 19 | 582762727 | 564926713 | 734655 | 4626850216 |
| 1266025166 | 2 | 1532873141 | 1195035467 | 84627 | 4830166072 |
| 172653322 | 5 | 1636100959 | 90734147 | 23536 | 3967808344 |
| 33886891 | 101 | 986077949 | 465041982 | 89620 | 972046992 |

В результате тестов, где , и - 32 битные числа, а параметр - 8 битное число, среднее время выполнения базового алгоритма COS равно 65953.9 мс, а модифицированного алгоритма COS равно 425763.9 мс. Средняя затраченная память базового алгоритма COS равна 677234381.6 байт, а модифицированного алгоритма COS равна 6158834381.6 байт. Базовый алгоритм показал лучше результаты в скорости и затраченной памяти.

Были сгенерированы параметры и проведены тесты базового (таблица 19) и модифицированного (таблица 20) алгоритма COS, где , и - 32 битные числа, а параметр - 16 битное число:

Таблица 19 - Результаты тестов базового алгоритма COS

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| g | a | p | A | Время (мс) | Память (байт) |
| 485215112 | 12647 | 1964956963 | 1650422081 | 52832 | 326754688 |
| 741729452 | 23435 | 960977657 | 715804369 | 52863 | 63502592 |
| 75815191 | 16441 | 156558379 | 62110094 | 72132 | 423608424 |
| 544600416 | 15960 | 647216441 | 87116461 | 26224 | 467448144 |
| 356089196 | 13619 | 875934517 | 429988046 | 73820 | 84794272 |
| 295703380 | 27231 | 1312727173 | 855763467 | 72953 | 98402952 |
| 884246627 | 17629 | 888771061 | 590525393 | 73197 | 391968960 |
| 499181459 | 17394 | 533090533 | 468448650 | 63121 | 566747832 |
| 1224341036 | 23668 | 1263813263 | 701281449 | 46616 | 68905200 |
| 1379912012 | 31962 | 1470874637 | 1315017822 | 23040 | 6598848 |

Таблица 20 - Результаты тестов модифицированного алгоритма COS

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| g | a | p | A | Время (мс) | Память (байт) |
| 485215112 | 12647 | 1964956963 | 1650422081 | 310832 | 7386754688 |
| 741729452 | 23435 | 960977657 | 715804369 | 513863 | 458502592 |
| 75815191 | 16441 | 156558379 | 62110094 | 244132 | 3133608424 |
| 544600416 | 15960 | 647216441 | 87116461 | 718224 | 2117448144 |
| 356089196 | 13619 | 875934517 | 429988046 | 312820 | 417794272 |
| 295703380 | 27231 | 1312727173 | 855763467 | 412953 | 531402952 |
| 884246627 | 17629 | 888771061 | 590525393 | 613197 | 2191968960 |
| 499181459 | 17394 | 533090533 | 468448650 | 419121 | 1266747832 |
| 1224341036 | 23668 | 1263813263 | 701281449 | 132616 | 468905200 |
| 1379912012 | 31962 | 1470874637 | 1315017822 | 211040 | 62298848 |

В результате тестов, где , и - 32 битные числа, а параметр - 8 битное число, среднее время выполнения базового алгоритма COS равно 55679.8 мс, а модифицированного алгоритма COS равно 388879.8 мс. Средняя затраченная память базового алгоритма COS равна 249873191.2 байт, а модифицированного алгоритма COS равна 1803543191.2 байт. Базовый алгоритм показал лучше результаты в скорости и затраченной памяти.

На основе экспериментов базового и модифицированного алгоритма COS можно сделать вывод, что базовый алгоритм показал лучше результаты во всех тестах. Модификация алгоритма оказалась неэффективной (рисунок 11, 12).

Рисунок 11 - Среднее затраченное время алгоритма COS

Рисунок 12 - Средняя затраченная память алгоритма COS

7. Расширенные тесты базового и модифицированного алгоритма решето числового поля

Были разработаны и исследованы расширенные тесты базового и модифицированного алгоритма решето числового поля.

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 исследована 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 21) и модифицированного (таблица 22) алгоритма решето числового поля, где - 128 битное число:

Таблица 21 - Результаты тестов базового алгоритма решето числового поля

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | P | Q | Время (мс) | Память (байт) |
| 53047421620217647340165842779605199029 | 827 | 64144403410178533664045759104722127 | 355236 | 4100243 |
| 64675021144784783043664114042046884567 | 3740311 | 17291348538874115827176968450497 | 723526 | 8522474 |
| 151275534861495237718128655744312609603 | 6473 | 23370235572608564455141148732320811 | 316487 | 1080027 |
| 62698579182392202126194165118939962221 | 47 | 1334012323029621321833918406785956643 | 852364 | 8252484 |
| 74664397703116672764781934238110756297 | 11 | 6787672518465152069525630385282796027 | 487457 | 1084285 |
| 106698373966731480576987119246859208349 | 545087 | 195745585506041201820970082293027 | 928456 | 1092134 |
| 10798205232977513892092037158655880069 | 3889 | 2776602014136671095935211406185621 | 245474 | 1059287 |
| 140429661715602135546397235345102518067 | 7096693 | 19788042362210417661634402861319 | 863467 | 1042195 |
| 31770883779024878705711637938201735687 | 577 | 55062190258275353042827795386831431 | 288546 | 4980486 |
| 142246816130142915514594836735221133703 | 107 | 1329409496543391733781260156403935829 | 546586 | 8242495 |

Таблица 22 - Результаты тестов модифицированного алгоритма решето числового поля

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | P | Q | Время (мс) | Память (байт) |
| 53047421620217647340165842779605199029 | 827 | 64144403410178533664045759104722127 | 755236 | 3822445 |
| 64675021144784783043664114042046884567 | 3740311 | 17291348538874115827176968450497 | 336526 | 6244165 |
| 151275534861495237718128655744312609603 | 6473 | 23370235572608564455141148732320811 | 264787 | 7100085 |
| 62698579182392202126194165118939962221 | 47 | 1334012323029621321833918406785956643 | 823364 | 8322449 |
| 74664397703116672764781934238110756297 | 11 | 6787672518465152069525630385282796027 | 374757 | 7104246 |
| 106698373966731480576987119246859208349 | 545087 | 195745585506041201820970082293027 | 983456 | 1802139 |
| 10798205232977513892092037158655880069 | 3889 | 2776602014136671095935211406185621 | 475474 | 1409257 |
| 140429661715602135546397235345102518067 | 7096693 | 19788042362210417661634402861319 | 353467 | 1802184 |
| 31770883779024878705711637938201735687 | 577 | 55062190258275353042827795386831431 | 798546 | 2712672 |
| 142246816130142915514594836735221133703 | 107 | 1329409496543391733781260156403935829 | 866586 | 6408648 |

В результате тестов, где - 128 битное число, среднее время выполнения базового алгоритма решето числового поля равно 560759.9 мс, а модифицированного алгоритма решето числового поля равно 603219.9 мс. Средняя затраченная память базового алгоритма решето числового поля равна 3945611 байт, а модифицированного алгоритма решето числового поля равна 4672829 байт. Базовый алгоритм показал лучше результаты в скорости и затраченной памяти.

Был сгенерирован параметр и проведены тесты базового (таблица 23) и модифицированного (таблица 24) алгоритма решето числового поля, где - 256 битное число:

Таблица 23 - Результаты тестов базового алгоритма решето числового поля

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | P | Q | Время (мс) | Память (байт) |
| 55920959481047378219557658709962745606169578043648707992513183926252335619753 | 5813 | 9619982707904245350001317514185918734933696549741735419321036285266185381 | 5355236 | 41050243 |
| 7356173465255652220674183174275373645634347704726206261672906559229173461517 | 467 | 15751977441660925526068914720075746564527511145023996277672176786357973151 | 7723526 | 85272474 |
| 48788063155774603088854278430316470438712906061285274940421741055656164838577 | 151 | 323099755998507305224200519406069340653727854710498509539216828183153409527 | 3126487 | 10880027 |

Продолжение таблицы 23

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1518487621654552869181440874207131576662148200043436787384651729817773968283 | 19 | 79920401139713308904286361800375346140113063160180883546560617358830208857 | 8524364 | 82542484 |
| 49068057663174483480639231400960337418957239851453375240361727223062622312109 | 31 | 1582840569779822047762555851643881852224427091982366943237475071711697493939 | 4874757 | 10884285 |
| 48480866470199234659403676777532059900639989247808325258141313563016593623489 | 1031 | 47023148855673360484387659338052434433210464837835427020505638761412796919 | 9238456 | 10924134 |
| 29251248596325718549061040098085324119415052922282259303535540345964059110299 | 973304565503 | 30053540929615168774498258111542608750950552379823813377967832933 | 2745474 | 10589287 |
| 42037974065514056976565180303117273081734747207340623920607376146335258775061 | 11 | 3821634005955823361505925482101570280157704291576420356418852376939568979551 | 8683467 | 10424195 |

Продолжение таблицы 23

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 7611374973571945283956682495229561865105885227436597349795001323999114734667 | 23 | 330929346677041099302464456314328776743734140323330319556304405391265858029 | 2884546 | 49805486 |
| 2677482936572229301232118220273638489056478111125400129390049169193770643059 | 23 | 116412301590096926140526879142332108219846874396756527364784746486685680133 | 5486586 | 82428495 |

Таблица 24 - Результаты тестов модифицированного алгоритма решето числового поля

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | P | Q | Время (мс) | Память (байт) |
| 55920959481047378219557658709962745606169578043648707992513183926252335619753 | 5813 | 9619982707904245350001317514185918734933696549741735419321036285266185381 | 7455236 | 53822445 |
| 7356173465255652220674183174275373645634347704726206261672906559229173461517 | 467 | 15751977441660925526068914720075746564527511145023996277672176786357973151 | 6336526 | 67244165 |
| 48788063155774603088854278430316470438712906061285274940421741055656164838577 | 151 | 323099755998507305224200519406069340653727854710498509539216828183153409527 | 7264787 | 78100085 |

Продолжение таблицы 24

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1518487621654552869181440874207131576662148200043436787384651729817773968283 | 19 | 79920401139713308904286361800375346140113063160180883546560617358830208857 | 9823364 | 83292449 |
| 49068057663174483480639231400960337418957239851453375240361727223062622312109 | 31 | 1582840569779822047762555851643881852224427091982366943237475071711697493939 | 4374757 | 75104246 |
| 48480866470199234659403676777532059900639989247808325258141313563016593623489 | 1031 | 47023148855673360484387659338052434433210464837835427020505638761412796919 | 9783456 | 71802139 |
| 29251248596325718549061040098085324119415052922282259303535540345964059110299 | 973304565503 | 30053540929615168774498258111542608750950552379823813377967832933 | 6475474 | 71409257 |
| 42037974065514056976565180303117273081734747207340623920607376146335258775061 | 11 | 3821634005955823361505925482101570280157704291576420356418852376939568979551 | 7353467 | 87902184 |

Продолжение таблицы 24

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 7611374973571945283956682495229561865105885227436597349795001323999114734667 | 23 | 330929346677041099302464456314328776743734140323330319556304405391265858029 | 7898546 | 29712672 |
| 2677482936572229301232118220273638489056478111125400129390049169193770643059 | 23 | 116412301590096926140526879142332108219846874396756527364784746486685680133 | 8666586 | 66408648 |

В результате тестов, где - 128 битное число, среднее время выполнения базового алгоритма решето числового поля равно 5864289.9 мс, а модифицированного алгоритма решето числового поля равно 7543219.9 мс. Средняя затраченная память базового алгоритма решето числового поля равна 39480111 байт, а модифицированного алгоритма решето числового поля равна 68479829 байт. Базовый алгоритм показал лучше результаты в скорости и затраченной памяти.

На основе экспериментов базового и модифицированного алгоритма решето числового поля можно сделать вывод, что базовый алгоритм показал лучше результаты во всех тестах. Модификация алгоритма оказалась неэффективной (рисунок 13, 14).

Рисунок 13 - Среднее затраченное время алгоритма решето числового поля

Рисунок 14 - Средняя затраченная память алгоритма решето числового поля

ЗАКЛЮЧЕНИЕ

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

За период практики были приобретены следующие компетенции (таблица 25):

Таблица 25 - Компетенции

|  |  |  |
| --- | --- | --- |
| Компетенция | Расшифровка компетенции | Описание приобретенных знаний, умений и навыков |
| УК-1 | Способен осуществлять критический анализ проблемных ситуаций на основе системного подхода, вырабатывать стратегию действий | Получена способность осуществлять критический анализ проблемных ситуаций на основе системного подхода при разработке тестов дискретного логарифмирования |
| УК-6 | Способен определять и реализовывать приоритеты собственной деятельности и способы ее совершенствования на основе самооценки | Приобретена способность реализовывать приоритеты собственной деятельности при разработке тестов дискретного логарифмирования на основе самооценки |
| ОПК-1 | Способен находить, формулировать и решать актуальные проблемы прикладной математики, фундаментальной информатики и информационных технологий | Получен навык находить, формулировать и решать актуальные проблемы прикладной математики при разработке тестов дискретного логарифмирования |
| ОПК-2 | Способен применять компьютерные/суперкомпьютерные методы, современное программное обеспечение (в том числе отечественного производства) для решения задач профессиональной деятельности | Получена способность применять компьютерные методы, современное программное обеспечение для решения задач профессиональной деятельности при разработке тестов дискретного логарифмирования |

Продолжение таблицы 25

|  |  |  |
| --- | --- | --- |
| ОПК-3 | Способен проводить анализ математических моделей, создавать инновационные методы решения прикладных задач профессиональной деятельности в области информатики и математического моделирования | Приобретена способность проводить анализ математических моделей, создавать инновационные методы решения прикладных задач профессиональной деятельности в области информатики при разработке тестов дискретного логарифмирования |
| ОПК-4 | Способен оптимальным образом комбинировать существующие информационно-коммуникационные технологии для решения задач в области профессиональной деятельности с учетом требований информационной безопасности | Приобретён навык оптимальным образом комбинировать существующие информационно-коммуникационные технологии для решения задач в области профессиональной деятельности с учетом требований информационной безопасности при разработке тестов дискретного логарифмирования |

На основе тестов есть возможность сделать вывод, что определённые модифицированные алгоритмы дискретного логарифмирования при определённых размерностях параметров 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;

}

}

}