Научная статья
УДК 519.68
doi: 10.34822/1999-7604-2022-3-114-125

МОДЕЛЬНЫЙ КОМПЛЕКС ПОИСКА ОСНОВАНИЙ МОДУЛЯРНОЙ АРИФМЕТИКИ ДЛЯ ПРЕДОТВРАЩЕНИЯ БИВАЛЕНТНОГО ЭФФЕКТА

Сергей Арнольдович Инютин [1], ***Виктор Олегович Иванов*** [2]✉
[1, 2] *Московский авиационный институт (национальный исследовательский университет), Москва, Россия*
[1] *inyutin_int@mail.ru, https://orcid.org/0000-0001-6879-8643*
[2] *viktorivanov95@gmail.com ✉, https://orcid.org/0000-0001-8597-1284*

Аннотация. Описана работа программного комплекса для решения задачи поиска пар простых чисел, равноудаленных от некоторого произвольного числа. Получены результаты работы с выделенными числами 2^{31} и 2^{32} на пределе возможностей серийных персональных компьютеров. Разработаны и описаны два разных алгоритма решения данной задачи с использованием языка программирования Python. Совпадение результатов, полученных двумя независимыми алгоритмами, можно трактовать как верификацию результатов решения задачи, визуальная идентификация которых затруднена из-за больших значений числовых величин. Полученные пары простых чисел можно использовать в качестве оснований модулярной арифметики, предназначенной для работы с большими числами, в которой устранен бивалентный эффект.

Ключевые слова: модулярная арифметика, бивалентный эффект цифрового регистра, модельный программный комплекс

Для цитирования: *Инютин С. А., Иванов В. О. Модельный комплекс поиска оснований модулярной арифметики для предотвращения бивалентного эффекта // Вестник кибернетики. 2022. № 3 (47). С. 114–125. DOI 10.34822/1999-7604-2022-3-114-125.*

Original article

MODEL COMPLEX FOR SEARCHING FOR MODULAR ARITHMETIC BASES TO PREVENT THE BIVALENT EFFECT

Sergey A. Inyutin [1], ***Viktor O. Ivanov*** [2]✉
[1, 2] *Moscow Aviation Institute (National Research University), Moscow, Russia*
[1] *inyutin_int@mail.ru, https://orcid.org/0000-0001-6879-8643*
[2] *viktorivanov95@gmail.com ✉, https://orcid.org/0000-0001-8597-1284*

Abstract. The article describes a software package aimed at solving the problem of searching for pairs of prime numbers equidistant from a certain arbitrary number. The results of work with the selected numbers 2^{31} and 2^{32} are obtained at the limit of the capabilities of mass-produced personal computers. Two different algorithms for solving the problem have been developed using the Python programming language and are described herein. The results' agreement obtained via two independent algorithms can be interpreted as verification of the problem solution results, which are difficult to identify visually due to large numerical values. The obtained pairs of prime numbers can be used as bases in modular arithmetic designed to work with large numbers, with bivalent effect eliminated.

Keywords: modular arithmetic, bivalent effect of digital register, model software package

For citation: *Inyutin S. A., Ivanov V. O. Model Complex for Searching for Modular Arithmetic Bases to Prevent the Bivalent Effect // Proceedings in Cybernetics. 2022. No. 3 (47). P. 114–125. DOI 10.34822/1999-7604-2022-3-114-125.*

ВВЕДЕНИЕ

Для записи машинных кодов в компьютерных системах в настоящее время широко используется позиционная двоичная система счисления. Позиционная система счисления опирается на способ векторной записи числовой величины, в котором компоненты вектора (цифры) имеют различные веса, зависящие от ее положения в векторной записи. Зависимость веса цифр от их положения требует учета переносов из одних разрядов в соседние. Это замедляет выполнение компьютерных операций и усложняет арифметико-логическое устройство вычислителя [1–2].

На современном этапе развития вычислительных систем особый интерес стали представлять компьютерные форматы для чисел, представленных в непозиционных системах счисления с параллельной структурой, которые позволяют сводить вычисления в основном к параллельным кольцевым операциям. Такой параллелизм позволяет делать модулярная система счисления (система остаточных классов).

Модулярная система счисления является непозиционной системой счисления, элементы которой описал китайский математик Сунь Цзы предположительно в III веке н. э. следующей формулировке:

- существует группа натуральных чисел $m_1, m_2, \dots, m_n$, называемых модулями (основаниями), и наибольший общий делитель каждой наугад выбранной пары чисел равен 1;

- существует группа натуральных чисел $r_1, r_2, \dots, r_n$, называемых остатками (вычетами), для которых соблюдается следующее неравенство $0 \le r_i < m_i$, где $i \in \{1, 2, 3 \dots n\}$;

- натуральное число N можно представить в виде группы чисел r, в которой $r_i \equiv \equiv N(\mathrm{mod}\, m_i)$;

- если найдутся два натуральных числа N_1 и N_2, у которых для каждого r будет выполняться равенство $r_i(N_1) = r_i(N_2)$, то $|N_1|_M = |N_2|_M$, где $M_n = m_1 \cdot m_2 \cdot \dots \cdot m_n$.

Числовое значение натурального числа N в модулярной системе счисления представлено в виде вектора, компонентами которого являются остатки (вычеты) от деления натурального числа N на выбранные модули (основания) системы. Оно имеет следующий вид [3–5]:

$$N = (r_1, r_2, \dots, r_n).$$

Разряды числовых величин при вычислениях в модулярной системе счисления меняются независимо друг от друга, что позволяет распараллелить и ускорить процесс вычисления, а также устранить эффект размножения ошибок. Оставаясь в рамках позиционной системы счисления, значительного ускорения выполнения операций добиться практически невозможно [6–7].

При отображении в цифровых регистрах компонента вектора модулярного представления числовой величины возникает бивалентный эффект, заключающийся в том, что не все возможные двоичные комбинации в цифровом двоичном регистре используются для отображения этих компонент. Доказано, что для уменьшения бивалентного эффекта при использовании форматов модулярной арифметики в качестве оснований должны быть взяты пары простых чисел, равноудаленные от некоторого серединного основания (например, 2^{31} или 2^{32}). Выбор этих значений для серединных оснований удобен при эмуляции модулярной арифметики на 32-разрядных серийных компьютерах [8].

Для поиска таких чисел было разработано два разных алгоритма на языке программирования Python, выбранном по причине наличия библиотек, сильно упрощающих написание кода для научных вычислений (SciPy, NumPy), а также из-за возможности легкого создания графической визуализации результатов расчета (с помощью библиотеки Matplotlib).

Поиск простых чисел такой величины является сложной задачей почти на пределе возможностей серийных персональных компьютеров. Итерационный процесс в алгоритмах содержит много шагов. Для повышения скорости работы алгоритмов используются ресурсы библиотеки Numba со встроенным в нее компилятором LLVM, которые ускоряют выполнение кода на алгоритмическом языке Python [9].

Алгоритмы дают одинаковый результат, что доказывает их взаимную правильность при решении поставленной задачи.

МАТЕРИАЛЫ И МЕТОДЫ

Для поиска пар простых чисел, равноудаленных от 2^{31} и 2^{32} было разработано два различных алгоритма, запрограммированных на алгоритмическом языке Python. Каждый блок программного кода в обоих алгоритмах сопровожден комментарием, и в отладочном режиме для него выполняется трассировка алгоритма, а также выводятся результаты выполнения.

Скрипт первого алгоритма размещен на проекте GitHub [10].

Алгоритм содержит следующие этапы:

1. Подключаются две библиотеки (NumPy и Time):

- библиотека NumPy включает в себя функции для обработки массивов, что позволило сильно упростить процесс разработки алгоритма;

- библиотека Time позволила подсчитать время выполнения алгоритма.

2. Предлагается введение серединного числа (используется функция input), для которого необходимо найти пары равноудаленных простых чисел.

3. В алгоритме для проверки на простоту используется решето Эратосфена с делителями до целой части арифметического квадратного корня из удвоенного серединного числа.

Для иллюстрации работы алгоритма введем серединное число 25 и рассмотрим его выполнение ($[\sqrt{50}] = 7$). Проверке подлежат пары чисел: 24, 26; 23, 27 и т. д.

Если одно число из пары делится на число из множества [2, 3, 4, 5, 6, 7] без остатка, то цикл переходит к проверке следующей пары равноудаленных чисел.

Результат выполнения кода для значения 25 показан на рис. 1.

В начале трассировки были отброшены пары чисел 24 и 26, 23 и 27, 22 и 28, 21 и 29, 20 и 30. Далее началась проверка пар чисел 19 и 31, последовательным нахождением остатка от деления каждого из чисел пары на 2, 3, 4, 5, 6 и 7. Каждый раз при делении остаток был больше 0, из-за чего программа в итоге выдала сообщение «Все возможные делители закончились, следовательно, 19 и 31 – простые числа!». Далее, таким же образом были найдены пары простых чисел 13 и 27.

Имеется особенность в случае проверки на простоту пар чисел, где одно число меньше или равно самому большому элементу во множестве делителей (в примере число равно $[\sqrt{50}] = 7$), например, пара чисел 7 и 43. По алгоритму число 43 проверяется на простоту делением на 2, 3, 4, 5, 6, 7. Первое число этой пары (7) таким образом проверить не можем, стандартная проверка показала бы, что число не является простым и пара простых чисел 7 и 43 не была бы зарегистрирована (т. к. 7 % 7 = 0). Поэтому был настроен фильтр, что числа, меньшие или равные самому большому числу во множестве проверочных делителей, проверяются на простоту делением только до числа меньшего проверяемого на единицу.

Фильтр также проверяет, равно ли меньшее число из пары делителю из массива. Если число не равно делителю, и если остаток при делении на него не равен нулю, то проверяется большее число из пары делением на это же число.

В примере проверяем, что 7 не равно 2, и что остаток от деления 7 на 2 не равен 0, а затем проверяем, не равен ли нулю остаток от деления 43 на 2 (рис. 2).

В конце трассировки появляются строки:

«7 <= 7 и 43 % 7 ! = 0 поэтому проводим проверку этой пары дальше все возможные делители закончились, следовательно, 7 и 43 – простые числа! 3 найденная пара».

7 <= 7, поэтому деление 7 на 7 для проверки не выполняется, выполняется только проверка 43 % 7.

Аналогичная продедура выполняется для пары чисел 3 и 47, также являющихся простыми и равноудаленными от 25.

Дополненный вариант первого алгоритма для работы с большими числами представлен в [11].

В работе этого алгоритма подключается библиотека Numba, которая позволяет ускорить выполнение алгоритма благодаря встроенному компилятору LLVM. Его использование способствует сокращению времени вычислений примерно в 40–50 раз.

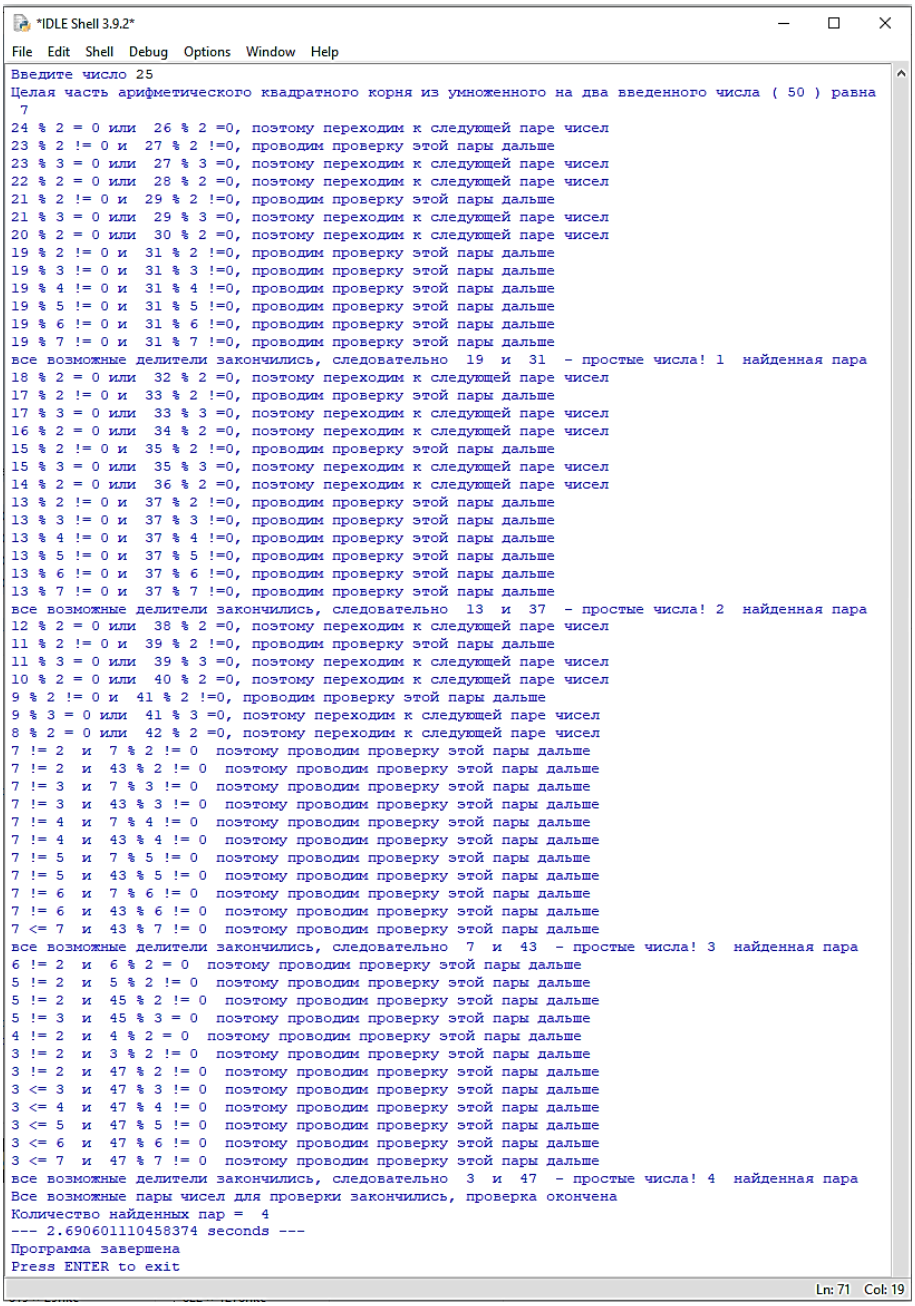

Рис. 1. Результат выполнения первого алгоритма для введенного числа 25
Примечание: составлено авторами.

```
7 != 2  и  7 % 2 != 0  поэтому проводим проверку этой пары дальше
7 != 2  и  43 % 2 != 0  поэтому проводим проверку этой пары дальше
7 != 3  и  7 % 3 != 0  поэтому проводим проверку этой пары дальше
7 != 3  и  43 % 3 != 0  поэтому проводим проверку этой пары дальше
7 != 4  и  7 % 4 != 0  поэтому проводим проверку этой пары дальше
7 != 4  и  43 % 4 != 0  поэтому проводим проверку этой пары дальше
7 != 5  и  7 % 5 != 0  поэтому проводим проверку этой пары дальше
7 != 5  и  43 % 5 != 0  поэтому проводим проверку этой пары дальше
7 != 6  и  7 % 6 != 0  поэтому проводим проверку этой пары дальше
7 != 6  и  43 % 6 != 0  поэтому проводим проверку этой пары дальше
7 <= 7  и  43 % 7 != 0  поэтому проводим проверку этой пары дальше
все возможные делители закончились, следовательно  7  и  43  - простые числа! 3  найденная пара
```

Рис. 2. Результат выполнения первого алгоритма для проверки чисел 7 и 43
Примечание: составлено авторами.

Также в этом алгоритме добавлено значение «шага подсчета», благодаря которому можно узнать сколько пар простых чисел, равноудаленных от введенного числа,

находится на заданном от него расстоянии. Для нашего примера с числом 25, введем в качестве шага подсчета значение «5». Увидим результат, представленный на рис. 3.

Графическое избражение результатов выполения алгоритма поиска пар простых чисел представлено на рис. 4.

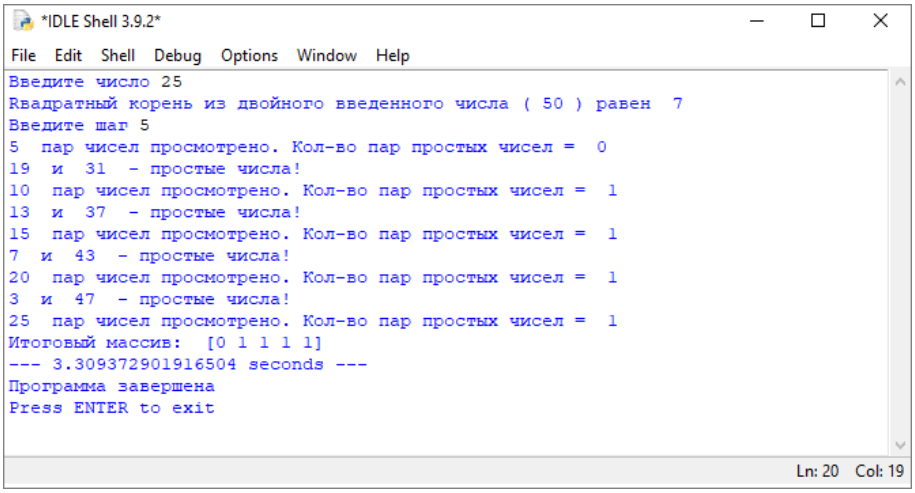

Рис. 3. Результат выполнения первого алгоритма для работы с большими числами
Примечание: составлено авторами.

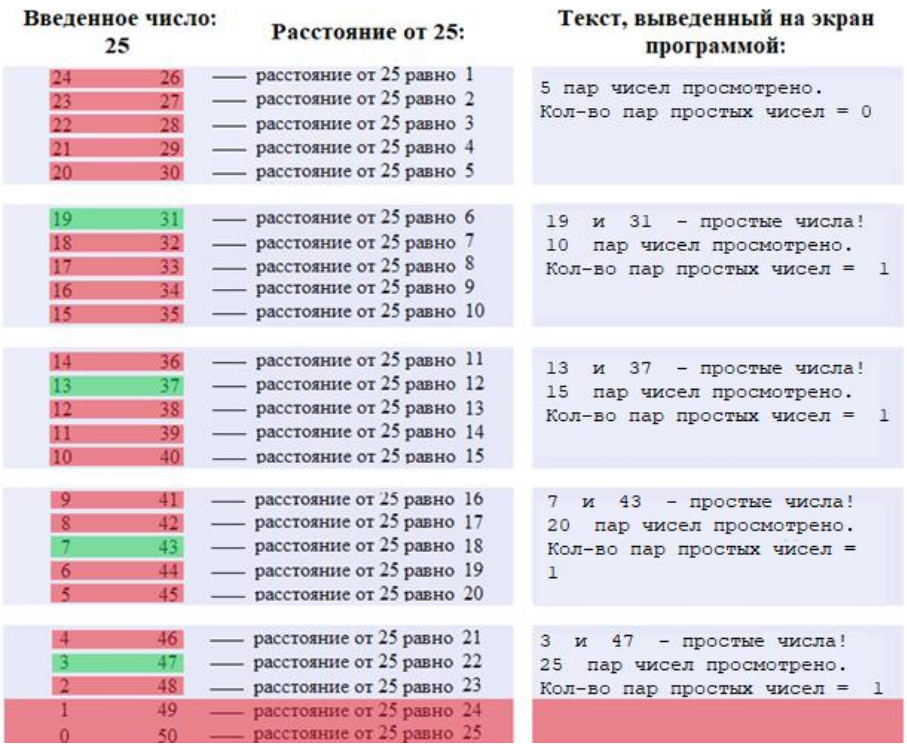

Рис. 4. Графическое изображение результатов выполнения алгоритма поиска пар простых чисел, равноудаленных от 25, с шагом подсчета 5
Примечание: составлено авторами.

На рис. 4 в первом столбце показаны пары чисел, равноудаленные от 25. Красным выделены те пары чисел, которые являются составными, зеленым – пары чисел, являющиеся простыми. Весь ряд чисел разделен на 5 блоков (по параметру «шаг подсчета», в каждом из блоков находится 5 пар чисел (т. к. 25/5 = 5). В последнем блоке

последние две пары (1, 49) и (0, 50) полностью выделены красным, т. к. проверка этих пар чисел на простоту не осуществляется.

РЕЗУЛЬТАТЫ И ИХ ОБСУЖДЕНИЕ

Протестируем этот алгоритм для числа 2^{31} = 2147483648. Выберем шаг подсчета 10^8 = 100000000. Для вычислений использовался компьютер, имеющий двухъядерный процессор и оперативную память 4 ГБ. Запустим алгоритм без вывода на экран самих простых чисел, а только итоговый результат (рис. 5).

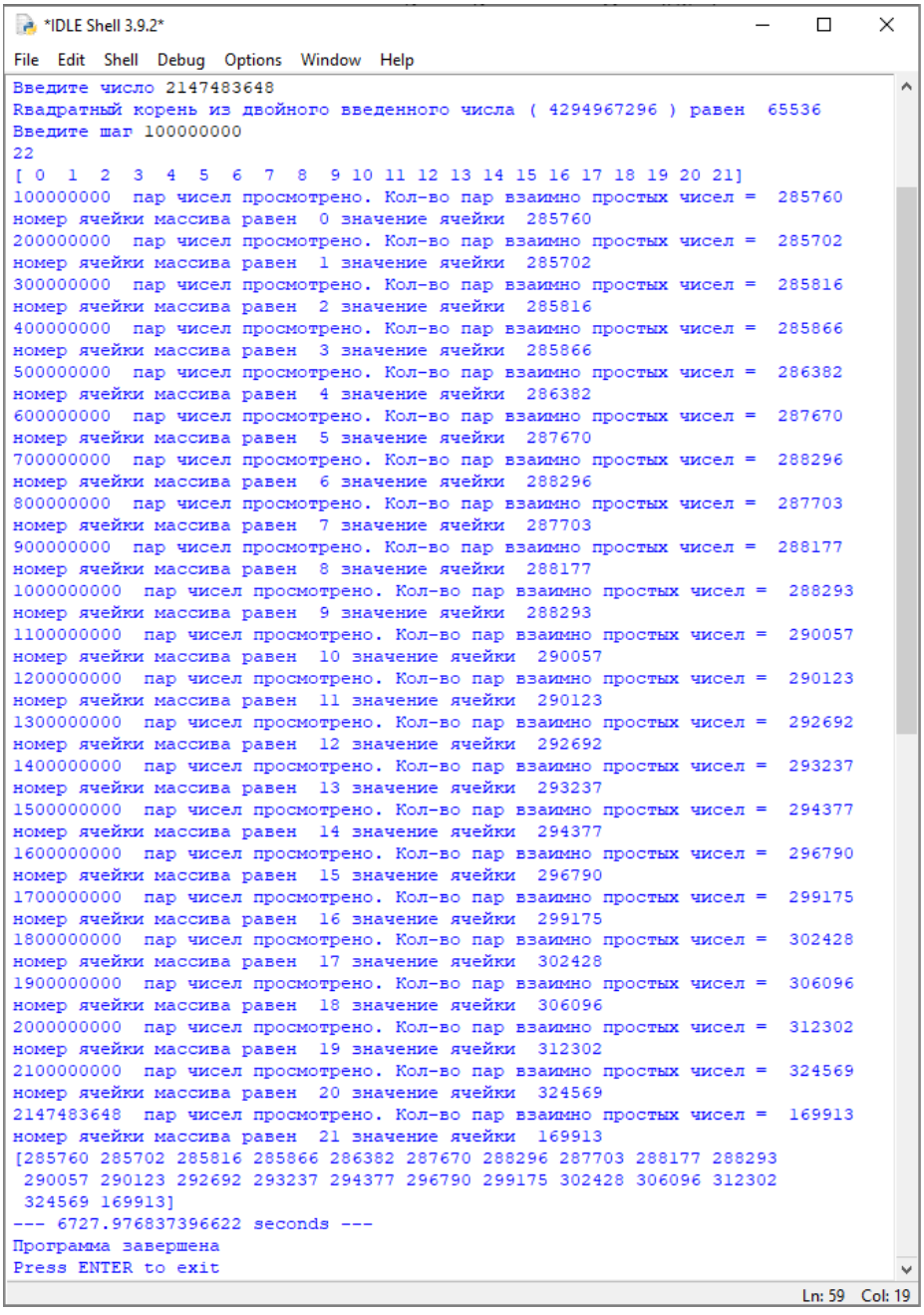

Рис. 5. Графическое изображение результата выполнения первого алгоритма для числа 2147483648 (2^{31}) с шагом подсчета 10^8

Примечание: составлено авторами.

В результате получено количество пар простых чисел, равноудаленных от 2^{31}. Вычисление удалось выполнить за 6196 секунд (около 103 минут).

Для проверки правильности результатов разработан второй алгоритм выполнения задачи, использующий принципиально другой способ вычислений. Попробуем оптимизировать его таким образом, чтобы вычисление происходило значительно быстрее.

Скрипт второго алгоритма представлен в [12].

Программа второго алгоритма содержит 13 блоков. Выполним его для числа 25 с шагом подсчета 5. Получим результат, представленный на рис. 6.

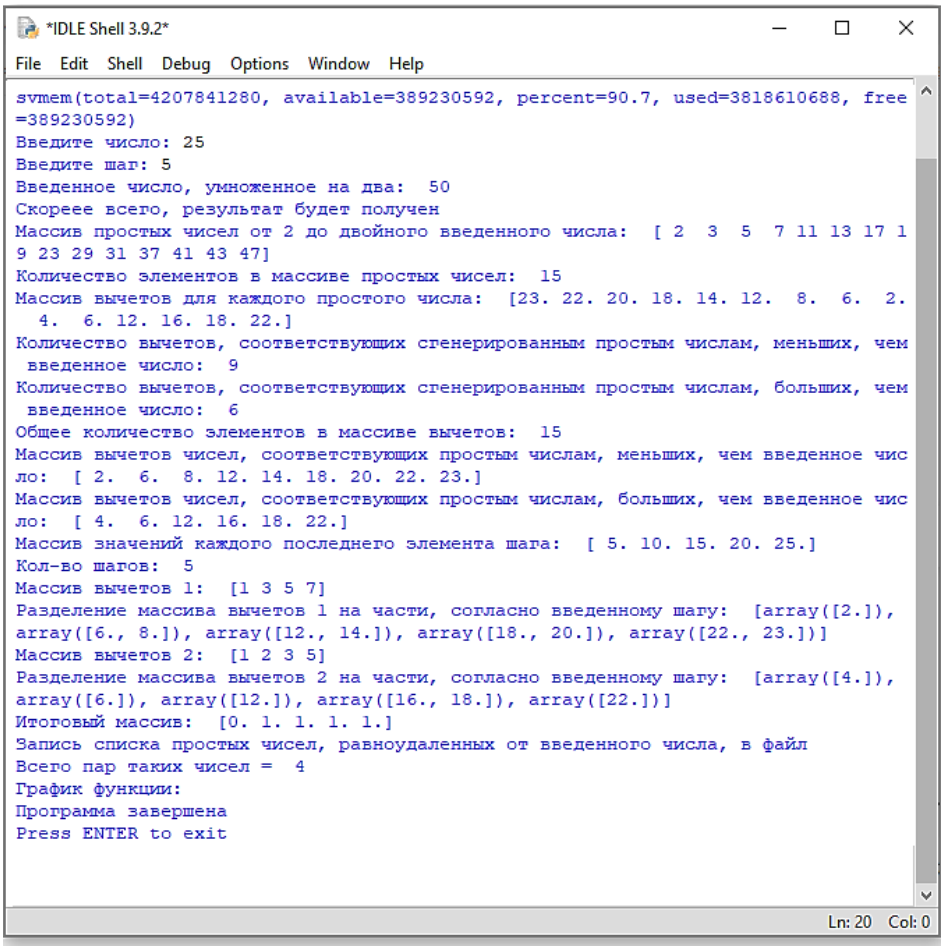

Рис. 6. Графическое изображение результата выполнения второго алгоритма для числа 25 с шагом подсчета 5
Примечание: составлено авторами.

В начале его работы подключаются библиотеки Numba, Numpy, Time, Math, Psutil и модуль pyplot из библиотеки Matplotlib.

Описание трех библиотек (Numba, Numpy и Time) дано ранее, дадим описание других:

- Math содержит много математических функций, которые позволили упростить процесс разработки алгоритма [13];
- Psutil позволяет оперативно получать информацию о системных параметрах компьютера [14];
- Matplotlib позволяет визуализировать полученные данные с помощью графиков, а ее модуль pyplot содержит ряд функций в стиле команд MATLAB [15].

Далее, в переменную mem вносится информация о системных параметрах компьютера (с помощью функции virtual_memory библиотеки psutil), которая выводится на экран:

- первый элемент «total» показывает общую доступную физическую память в байтах (в примере доступно 4207841280 байт (4,21 ГБ));
- второй элемент «available» показывает общее количество физической памяти, которое может быть отдано процессу без свопин-

га (в нашем случае доступно 497000448 байт, 0,5 ГБ);

- третий элемент «percent» вычисляет общее количество незанятой оперативной памяти в процентах (в примере 88,2 %);

- четвертый параметр «used» равен разности объемов памяти:

4207841280 – 497000448 = 3710840832.

Первый блок программы называется «Основные переменные». В нем с помощью функции input вводится серединное число, относительно которого нужно найти пары равноудаленных от него простых чисел, а также «шаг подсчета».

Как и в первом алгоритме, в качестве примера введем число 25 и «шаг подсчета» 5.

Как показала практика, результат с большей вероятностью будет вычислен, если введенное число примерно вдвое меньше доступной физической памяти в байтах, об этом говорит текст «Результат будет получен». В нашем случае общий объем доступной памяти равен 4207841280 байт, следовательно, максимальное число, для которого с большей вероятностью можно получить результат вычисления, равно 2103920640, что чуть меньше 2^{31}. Также практическим путем было обнаружено, что если число немного превышает деленный на два объем физической памяти, то результат все равно будет вычислен. В случае, если введенное число значительно превышает объем физической памяти, деленный на два, то с большей вероятностью возникнет ошибка «Out of memory».

В блоке «Генератор простых чисел» используется генератор простых чисел, опубликованный автором Robert William Hanks на портале Stackoverflow [16]. Автор утверждает, что данный генератор «Faster & more memory-wise pure Python code» («Наиболее быстрый и ресурсоемкий генератор простых чисел, разработанный на чистом Python»). Алгоритм генерирует простые числа, начиная с числа 3.

Третий этап алгоритма – преобразование полученного массива простых чисел в массив формата numpy, это необходимо для дальнейшей работы с ним с применением библиотеки numpy.

На четвертом этапе к массиву формата numpy добавляется число 2. Массив numpy с простыми числами выводится на экран.

Пятый этап алгоритма содержит вычисление вычетов для каждого простого числа (из каждого сгенерированного простого числа находится величина модуля разности с серединным числом 25). То есть, вычисляются значения |2 – 25|, |3 – 25|, |5 – 25| … |43 – 25|, |47 – 25|.

На шестом этапе алгоритма вычисляется каждый последний элемент шага и общее количество шагов.

На следующих четырех этапах (7–10) массивы вычетов (соответствующих простым числам меньшим, чем введенное число и большим, чем введенное число) делятся в соответствии с шагами.

На 11-м этапе алгоритма записывается итоговый массив, который равен [0. 1. 1. 1. 1. 1.].

Результат совпадает с тем результатом, который получен с помощью первого алгоритма.

На 12-м этапе полученный результат записывается в файл формата txt, затем результат отображается на графике (13-й этап).

Теперь для второго алгоритма введем серединное число 2^{31} = 2147483648, с шагом 10^8 = 100000000. Получим результат, представленный на рис. 7. Программа была запущена без вывода на экран пошаговых результатов.

Полученный результат совпадает с результатом работы первого алгоритма.

Графическое изображение, которое выводится на экран после выполнения алгоритма, представлено на рис. 8.

Изображение окна текстового файла с парами чисел, равноудаленных от введенного числа (в данном случае 2^{31}), которое создается в той же папке, где находится сам запускаемый алгоритм, представлено на рис. 9.

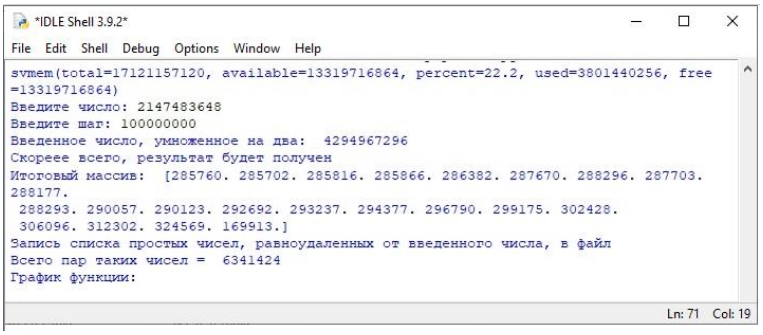

Рис. 7. Графическое изображение результата выполнения второго алгоритма для числа 2^{31} с шагом подсчета 10^8
Примечание: составлено авторами.

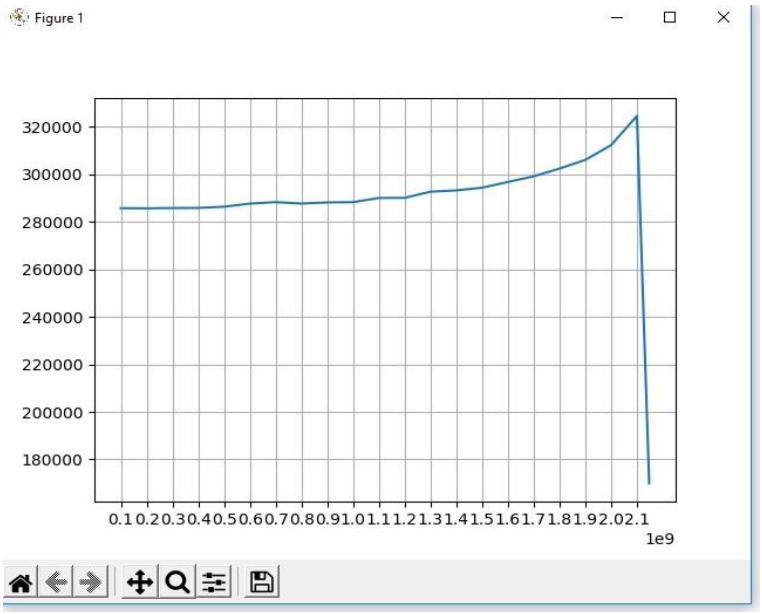

Рис. 8. График количества пар простых чисел, равноудаленных от 2^{31}.
По оси Y – количество пар чисел, по оси X – значение шага подсчета
Примечание: составлено авторами на основании данных, полученных в исследовании.

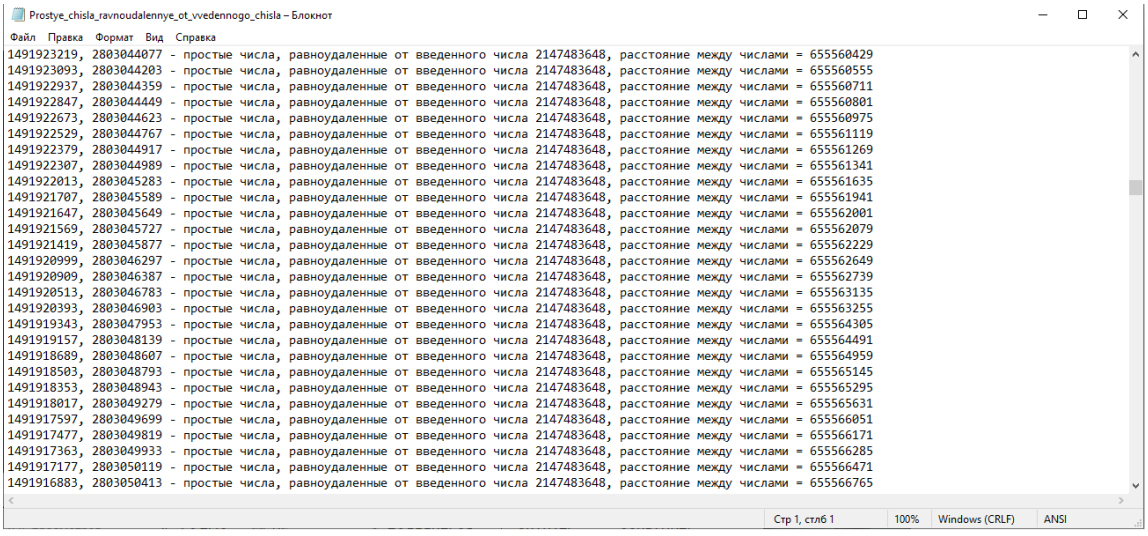

Рис. 9. Графическое изображение результатов
выполнения алгоритма поиска пар простых чисел, равноудаленных от 2^{31}
Примечание: составлено авторами на основании данных, полученных в исследовании.

Выполним второй алгоритм для серединного числа 2^{32} = 4294967296 с шагом 10^8 = = 100000000 на компьютере с оперативной памятью 16 ГБ (рис. 10).

Графическое изображение, которое выводится на экран после выполнения алгоритма, представлено на рис. 11.

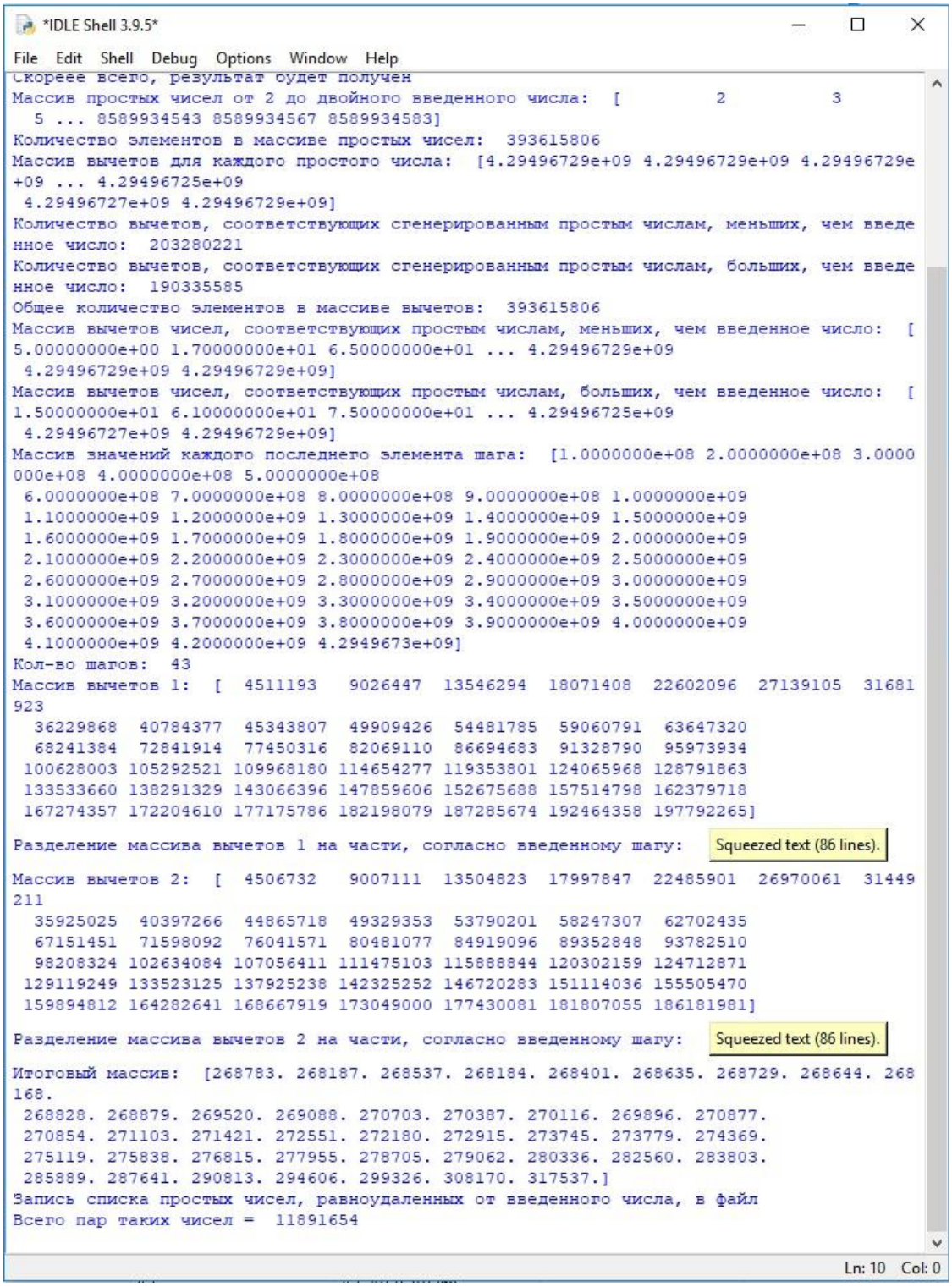

Рис. 10. Графическое изображение результата выполнения второго алгоритма для числа 2^{32} с шагом подсчета 10^8
Примечание: составлено авторами.

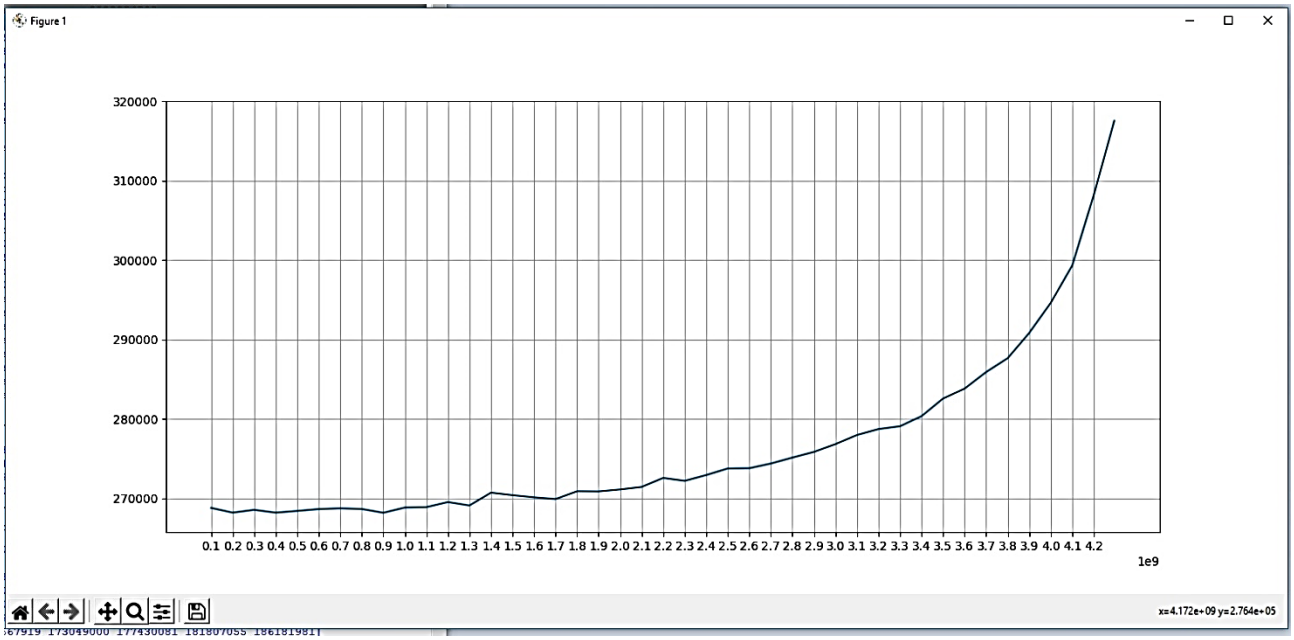

Рис. 11. График количества пар простых чисел, равноудаленных от 2^{32}.
По оси Y – количество пар чисел, по оси X – значение шага подсчета
Примечание: составлено авторами на основании данных, полученных в исследовании.

ЗАКЛЮЧЕНИЕ

При выборе серединного числа 2^{31} получено 6341424 пар простых чисел. Файл, содержащий их, имеет объем 755,7 МБ. При выборе серединного числа 2^{32} получено 11891654 пар простых чисел. Файл, содержащий их, имеет объем 1,39 ГБ. Полученные пары простых чисел можно использовать в качестве оснований модулярной арифметики, предназначенной для работы с большими числами и в которой устранен бивалентный эффект. Результаты работы двух различных алгоритмов полностью совпадают. Совпадение результатов работы свидетельствует о верификации программных реализаций алгоритмов. Сравнительный анализ алгоритмов свидетельствует, что первый алгоритм работает медленнее второго, с его использованием поиск пар простых чисел для заданного числа 2^{31} выполнен за 6196 секунд, а с использованием второго алгоритма аналогичный поиск выполнен за 51 секунду. Поиск пар простых чисел, равноудаленный от серединного числа 2^{32}, вторым алгоритмом был выполнен за 112 секунд.

Список источников

1. Инютин С. А. Модулярная алгоритмика многоразрядных вычислений. М. : Изд-во МАИ, 2020. 160 с.
2. Амербаев В. М., Балака Е. С., Тельпухов Д. В., Соловьев Р. А. Применение информационной избыточности для повышения надежности арифметического узла вычислительного элемента бимодульной арифметики // Параллельная компьютерная алгебра и ее приложения в новых инфокоммуникационных системах – 2014 : сб. науч. тр. I Междунар. конф. Ставрополь, 2014. С. 347–358.
3. Червяков Н. И., Бабенко М. Г., Кучеров Н. Н. Применение корректирующих кодов СОК для диагностики работы модулярных процессоров //

References

1. Inyutin S. A. Moduliarnaia algoritmika mnogorazriadnykh vychislenii. Moscow : Publishing House Moscow Aviation Institute, 2020. 160 p. (In Russian).
2. Amerbaev V. M., Balaka E. S., Telpukhov D. V., Solovyev R. A. Application Information Redundancy to Improve Reliability the Arithmetic Unit Computing Elements Bimodule Arithmetic // Parallelnaia kompiuternaia algebra i ee prilozheniia v novykh infokommunikatsionnykh sistemakh – 2014 : Proceedings of the 1st International Conference. Stavropol, 2014. P. 347–358. (In Russian).
3. Chervyakov N. I., Babenko M. G., Kucherov N. N. The Use of Error-Correcting Codes for the Diagnosis of RNS Modular Processors // Science. Innovations. Technologies. 2014. No. 3. P. 24–39. (In Russian).

Наука. Инновации. Технологии. 2014. № 3. С. 24–39.
4. Лавриненко А. В. Метод преобразования кода системы остаточных классов в позиционный с коррекцией ошибок на основе искусственных нейронных сетей // Наука. Инновации. Технологии. 2015. № 3. С. 7–36.
5. Эрдниева Н. С. Использование специальных модулей системы остаточных классов для избыточного представления // Вестн. АГТУ. Сер. Упр., вычислительная техника и информатика. 2013. № 2. С. 75–84.
6. Золотарева Н. С., Инютин С. А. Методы выбора оснований, понижающих бивалентный дефект в системе остаточных классов // Вестник кибернетики. 2020. С. 6–7.
7. Магомедов Ш. Г. Преобразование представлений чисел в модулярной арифметике в системах остаточных классов с разными основаниями // Вестн. АГТУ. Сер. Управление, вычислительная техника и информатика. 2014. № 4. С. 32–39.
8. Инютин С. А. Модулярные процессоры – оценки, история борьбы и победы над бивалентным дефектом // Развитие вычислительной техники в России и странах бывшего СССР: история и перспективы. SoRuCom-2017 : сб. тр. IV Междунар. конф., Зеленоград, 3–5 октября 2017 г. М., 2017. С. 72–77.
9. A ~5 Minute Guide to Numba. URL: https://numba.readthedocs.io/en/stable/user/5minguide.html (дата обращения: 01.09.2022).
10. Primesequidistants1.1. URL: https://github.com/viktorivanov95/math/blob/main/primesequidistants1.1 (дата обращения: 19.09.2022).
11. Primesequidistants1.2. URL: https://github.com/viktorivanov95/math/blob/main/primesequidistants1.2 (дата обращения: 19.09.2022).
12. Primesequidistants1.2. URL: https://github.com/viktorivanov95/math/blob/main/primesequidistants2.1 (дата обращения: 19.09.2022).
13. Math – Mathematical Functions. URL: https://docs.python.org/3/library/math.html (дата обращения: 01.09.2022).
14. Psutil 5.9.2. URL: https://pypi.org/project/psutil/ (дата обращения: 01.09.2022).
15. Matplotlib: Visualization with Python. URL: https://matplotlib.org (дата обращения: 01.09.2022).
16. 16. Fastest Way to List All Primes below N. URL: https://stackoverflow.com/a/3035188/14513464 (дата обращения: 19.09.2022).

4. Lavrinenko A. V. Method of Conversion from Residue Number System to Positional with Error Correctness Based on Artificial Neural Networks // Science. Innovations. Technologies. 2015. No. 3. P. 7–36. (In Russian).
5. Erdnieva N. S. Use of Special Modules of the Residue Number System for Redundant Representation // Vestnik of Astrakhan State Technical University. Series: Management, Computer Science and Informatics. 2013. No. 2. P. 75–84. (In Russian).
6. Zolotareva N. S., Inyutin S. A. Methods for Selection of Bases Reducing Bivalent Defect in Residue Number System // Proceedings in Cybernetics. 2020. P. 6–7. (In Russian).
7. Magomedov Sh. G. Conversion of Numeration in Modular Arithmetic in the Systems of Residual Classes with Different Bases // Vestnik of Astrakhan State Technical University. Series: Management, Computer Science and Informatics. 2014. No. 4. P. 32–39. (In Russian).
8. Inyutin S. A. Moduliarnye protsessory – otsenki, istoriia borby i pobedy nad bivalentnym defektom // Computer Technology in Russia and in the Former Soviet Union. SoRuCom-2017 : Proceedings of the 4th International Conference, Zelenograd, October 3–5, 2017. Moscow, 2017. P. 72–77. (In Russian).
9. A ~5 Minute Guide to Numba. URL: https://numba.readthedocs.io/en/stable/user/5minguide.html (accessed: 01.09.2022).
10. Primesequidistants1.1. URL: https://github.com/viktorivanov95/math/blob/main/primesequidistants1.1 (accessed: 19.09.2022).
11. Primesequidistants1.2. URL: https://github.com/viktorivanov95/math/blob/main/primesequidistants1.2 (accessed: 19.09.2022).
12. Primesequidistants1.2. URL: https://github.com/viktorivanov95/math/blob/main/primesequidistants2.1 (accessed: 19.09.2022).
13. Math – Mathematical Functions. URL: https://docs.python.org/3/library/math.html (accessed: 01.09.2022).
14. Psutil 5.9.2. URL: https://pypi.org/project/psutil/ (accessed: 01.09.2022).
15. Matplotlib: Visualization with Python. URL: https://matplotlib.org (accessed: 01.09.2022).
16. Fastest Way to List All Primes below N. URL: https://stackoverflow.com/a/3035188/14513464 (accessed: 19.09.2022).

Информация об авторах

С. А. Инютин – доктор технических наук, профессор.

В. О. Иванов – аспирант.

Information about the authors

S. A. Inyutin – Doctor of Sciences (Engineering), Professor.

V. O. Ivanov – Postgraduate.