ИНЖИНИРИНГ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ, ПРОГРАММИРОВАНИЕ СЕТЕВЫХ ПРИЛОЖЕНИЙ И АСПЕКТЫ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

УДК 543.552.054.1

ВИЗУАЛИЗАЦИЯ АЛГОРИТМА ДИКСОНА: ИНСТРУМЕНТ ДЛЯ ФАКТОРИЗАЦИИ И ОБУЧЕНИЯ

Арванова Саният Мухамедовна, старший преподаватель,
Кашежев Алим Заурбекович, студент,
Шогенов Эльдар Заурович, студент,
ФГБОУ ВО «КБГУ», г. Нальчик

VISUALIZATION OF THE DIXON ALGORITHM: A TOOL FOR FACTORIZATION AND LEARNING

Arvanova Saniyat Mukhamedovna, Senior Lecturer, sani_07@mail.ru,
Kashezhev Alim Zaurbekovich, Student, alim.kashezhev@562@gmail.com,
Shogenov Eldar Zaurovich, student, eldarshogenov03@gmail.com,
Kabardino-Balkarian State University, Nalchik

Статья посвящена разработке и анализу программы, реализующей алгоритм Диксона для факторизации целых чисел с использованием PyQt5. Данная работа подчеркивает образовательный и практический потенциал инструмента в области криптографии и теории чисел.

Ключевые слова: *алгоритм Диксона, факторизация, визуализация, криптография, линейная алгебра.*

The article is devoted to the development and analysis of a program implementing the Dixon algorithm for factoring integers using PyQt5. This work highlights the educational and practical potential of the tool in the field of cryptography and number theory.

Keywords: *Dixon's algorithm, factorization, visualization, cryptography, linear algebra.*

На сегодняшний день факторизация чисел представляет собой одну из фундаментальных задач в теории чисел, которая обладает глубокими связями с криптографией, особенно в контексте систем с открытым ключом, таких как RSA [1]. В свою очередь, сложность разложения больших составных чисел на простые множители лежит в основе безопасности многих криптографических протоколов, что делает разработку и изучение эффективных алгоритмов факторизации актуальной задачей как для исследователей, так и для практиков. Одним из таких алгоритмов является метод Диксона, представляющий собой вероятностный подход, который основан на поиске B-гладких чисел и решении систем линейных уравнений по модулю 2. Однако, несмотря на свою значимость, алгоритм Диксона может быть сложен для понимания, особенно это может проявляться среди студентов и начинающих специалистов, в связи с тем, что отсутствуют наглядные инструменты, которые смогли бы в полной мере продемонстрировать его работу.

В данной статье представлена программа, разработанная на базе библиотеки PyQt5, которая решает эту проблему, предоставляя графический интерфейс для пошаговой визуализации процесса факторизации чисел с помощью алгоритма Диксона, представленный на рисунке 1.

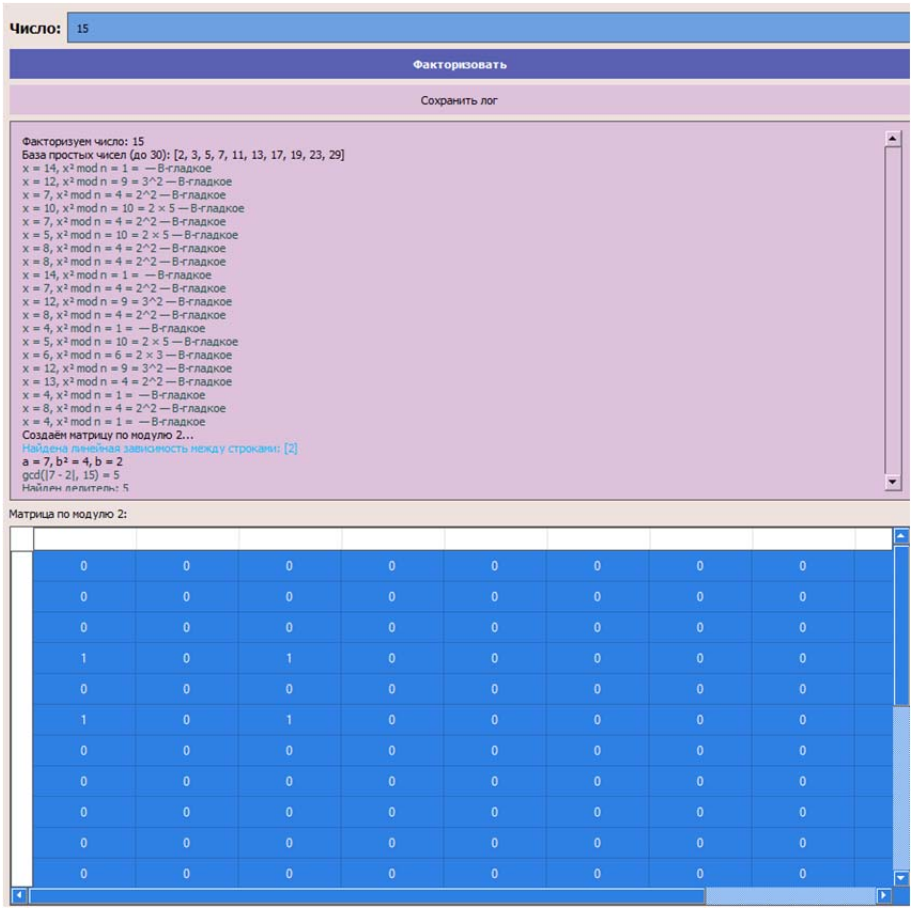

Рисунок 1 – Пользовательский интерфейс

В свою очередь, программа способна не только выполнять разложение чисел на множители, но и позволяет пользователям наблюдать за каждым этапом алгоритма через цветную подсветку логов и отображение матрицы зависимостей, что делает процесс обучения более интуитивным и доступным. Целью статьи является описание разработки программы, её функциональных возможностей и демонстрация того, как визуализация может способствовать лучшему пониманию сложных алгоритмических концепций.

Итак, рассмотрим, программную реализацию алгоритма Диксона. Как было отмечено ранее, для реализации этого алгоритма и его наглядной демонстрации была разработана программа с использованием библиотеки PyQt5, которая обеспечивает кроссплатформенный графический интерфейс. В разработке также применялись библиотеки SymPy и NumPy. Стоит отметить, что SymPy отвечает за B-гладкости чисел и их разложения на простые множители, а NumPy используется для эффективной работы с матрицами при поиске линейных зависимостей. Основная функция программы, *dixon_factor_verbose,* выполняет факторизацию числа *n*, принимая на вход параметры *bound* (значение B) и *max,elations* (максимальное число векторов для построения матрицы). Она сначала проверяет, не является ли *n* четным, и, если это так, сразу возвращает делитель 2 [2]. С кодом реализуемой функции можно ознакомиться в листинге 1.

Листинг 1 – Функция *dixon_factor_verbose* по факторизации числа *n*.

```
def dixon_factor_verbose(n, bound=30, max_relations=20):
logs = []
matrix_data = []
if n % 2 == 0:
logs.append(("Число чётное, делится на 2.", 'success'))
return 2, logs, matrix_data
logs.append((f"Факторизуем число: {n}", 'info'))
factor_base = list(primerange(2, bound))
```

```
logs.append((f"База простых чисел (до {bound}): {factor_base}", 'info'))
relations = []
x_list = []
x2_list = []
while len(relations) < max_relations:
x = random.randint(int(sqrt(n)) + 1, n - 1)
x2 = x * x % n
exponents = is_smooth(x2, factor_base)
if exponents is not None:
relations.append(exponents)
x_list.append(x)
x2_list.append(x2)
factorization_str = []
for p, e in zip(factor_base, exponents):
if e > 0:
factorization_str.append(f"{p}^{e}" if e > 1 else str(p))
logs.append((f"x = {x}, x² mod n = {x2} = {' × '.join(factorization_str)} — B-гладкое", 'success'))
else:
logs.append((f"x = {x}, x² mod n = {x2} — не B-гладкое", 'fail'))
logs.append(("Создаём матрицу по модулю 2...", 'info'))
matrix = mod2_matrix(relations)
matrix_data = matrix.tolist()
dependency = find_dependency(matrix)
if not dependency:
logs.append(("Не удалось найти линейную зависимость.", 'fail'))
return None, logs, matrix_data
a = 1
b_squared = 1
for idx in dependency:
a = (a * x_list[idx]) % n
b_squared = (b_squared * x2_list[idx]) % n
b = math.isqrt(b_squared)
g = gcd(a - b, n)
logs.append((f"Найдена линейная зависимость между строками: {dependency}", 'highlight'))
logs.append((f"a = {a}, b² = {b_squared}, b = {b}", 'info'))
logs.append((f"gcd(|{a} - {b}|, {n}) = {g}", 'success' if 1 < g < n else 'fail'))
if 1 < g < n:
logs.append((f"Найден делитель: {g}", 'success'))
return g, logs, matrix_data
else:
logs.append(("Не удалось найти множители. Попробуйте снова.", 'fail'))
return None, logs, matrix_data
```

Затем формирует факторный базис с помощью функции *primerange* из SymPy, которая генерирует простые числа до *bound*. Далее в цикле генерируются случайные x, вычисляется $y = x^2 \bmod n$, и с помощью функции *is_smooth* (которая представлена для ознакомления в листинге 2) проверяется, является ли y B-гладким [4]. В свою очередь, данная функция использует *factorint* из SymPy для разложения y на простые множители и проверяет, все ли они принадлежат базису, возвращая вектор показателей или None, если y не B-гладкое.

Листинг 2 – Функция *is_smooth* для проверки y на B-гладкость.

```
def is_smooth(n, factor_base):
factors = factorint(n)
exponents = [factors.get(p, 0) for p in factor_base]
for p in factors:
if p not in factor_base:
```

return None
return exponents

В случае, если *y* B-гладкое, его вектор добавляется в коллекцию, а x и y сохраняются для последующих вычислений. Стоит также отметить, что все шаги логируются. Так, успешное нахождение B-гладкого числа отмечается как успех, а неудача — как провал. После накопления достаточного количества векторов вызывается функция *mod2_matrix*, которая преобразует собранные векторы в бинарную матрицу, приводя показатели по модулю 2 [5]. С функцией *mod2_matrix* также можно ознакомиться, но уже в листинге 3.

Листинг 3 – Функция *mod2_matrix*, для преобразования векторов в бинарную матрицу

def mod2_matrix(relations):
return np.array([[e % 2 for e in rel] for rel in relations])

Затем функция *find_dependency* выполняет поиск линейной зависимости в этой матрице с использованием метода Гаусса по модулю 2. Она добавляет к матрице единичную матрицу справа, чтобы отслеживать, какие строки участвуют в зависимости, и проводит преобразования, пока не найдёт строку, соответствующую нулевому вектору слева и ненулевому справа, что указывает на зависимые строки. Причем эти строки используются для вычисления *a* как произведения соответствующих *x* и *b* как корня из произведения *y*, после чего gcd(a – b, n) даёт делитель [3].

В свою очередь, интерфейс программы был спроектирован так, чтобы пользователь мог легко взаимодействовать с алгоритмом и наблюдать за его работой. Основное окно включает поле ввода для числа n, кнопку "Факторизовать", которая вызывает функцию *run*, запускающую *dixon_factor_verbose* и отображающую результаты, и текстовую область для логов. Логи выводятся с помощью метода *append_log*, который применяет цветовую подсветку, где зелёный для успешных шагов, красный для неудач и синий для ключевых событий, таких как нахождение зависимости. Таблица, реализованная через *QTableWidget*, отображает матрицу по модулю 2, обновляясь в реальном времени с помощью метода *show_matrix*, который заполняет её строками, соответствующими B-гладким числам. Кнопка "Сохранить лог" вызывает метод *save_log*, использующий *QFileDialog* для сохранения логов в файл, что удобно для анализа или подготовки отчётов. Таким образом, были описаны все компоненты пользовательского интерфейса представленного на рисунке 1.

Подводя итоги, разработанное программное решение делает алгоритм Диксона доступным для понимания, позволяя пользователю наблюдать за каждым этапом, начиная с генерации *x* и проверки B-гладкости до построения матрицы и нахождения делителей. Её функционал, включающий визуализацию, сохранение логов и поддержку больших чисел, делает её ценным инструментом для образовательных целей и практического применения в криптографии и теории чисел.

Список использованных источников

1. Кирин Д.А., Сааков В.В. Методы и программно-аппаратные средства защиты информации от несанкционированного доступа в сети интернет // Инновационные научные исследования. 2021. № 9-1(11). С. 31-37. URL: https://ip-journal.ru/

2. Рыбалов А.Н. О генерической сложности проблемы факторизации целых чисел // Прикладная дискретная математика. – 2023. – №61. – С. 121–126.

3. Макаренко А.В., Пыхтеев А.В., Ефимов С.С. Параллельная реализация и сравнительный анализ алгоритмов факторизации в системах с распределённой памятью // Вестник Воронежского государственного университета. Серия: Системный анализ и информационные технологии. – 2021. – №3. – С. 45–52.

4. Мажара В.В., Подколзина Е.Ю. Алгоритмы факторизации целых чисел с экспоненциальной сложностью // Сборник статей Международной научно-практической конференции «Инновационные подходы в науке и образовании». – 2023. – С. 56–60.

5. Георгиева М. А., Ксенофонтов А. С., Блиева О. З., Джамихова Ф. Х., Езаова Б. З., Тлепшева Д. А. Разработка калькулятора для решения систем линейных алгебраических уравнений // Современная наука: актуальные проблемы теории и практики. Серия: Естественные и технические науки. 2023. № 1. С. 71–75.