Харківський національний університет імені В.Н. Каразіна
Факультет комп'ютерних наук
Безпека інформаційних систем і технологій

«Допущено до захисту»
Зав. кафедрою БІСТ
І.І. Сватовський _____
« » червня 2023 р.

Пояснювальна записка

до кваліфікаційної роботи бакалавра
спеціальність: 125 Кібербезпека

на тему «Моделювання та програмна реалізація тестування низького ступеня у криптографічних протоколах доказу з нульовим розголошенням»

Оцінка « _____»

Керівник проф. Кузнецов О.О. _____

Голова ЕК
Лемешко О.В._____

Рецензент проф. Краснобаєв В.А._____

Виконавець студент групи КБ-42
Аверков Олег Юрійович _____

Харків - 2023

РЕФЕРАТ

Пояснювальна записка містить 88 сторінок, 5 рисунків, 8 таблиць, 10 додатків, 26 джерел.

Метою кваліфікаційної роботи є вивчення моделювання та тестування низького ступеня у криптографічних протоколах доказу із нульовим розголошенням шляхом програмної реалізації першого етапу Arethmetization роботи протоколу ZK-STARK.

Об'єктом дослідження кваліфікаційної роботи є процес забезпечення обчислювальної цілісності транзакції мережі Блокчейн.

Предметом розробки є ZK-STARK – криптографічний протокол, який використовує публічні докази з нульовим розголошенням.

Методи дослідження: математичні відомості з Теорії інтерполювання, Теорії груп, Теорії чисел; відомості про числа Фібоначчі; відомості про функцію Ейлера; породжуючий елемент групи; циклічні групи; інтерполяційний поліном Лагранжа та послідовність обчислень.

Для програмної реалізації була обрана мова програмування C^{++}, середа розробки додатків Visual Studio 2022 та бібліотека «NTL» для розробки додатків, які використовують Теорію чисел.

Результатами проведеної роботи. У роботі проведено моделювання та тестування низького ступеня у криптографічних протоколах доказу із нульовим розголошенням шляхом програмної реалізації першого етапу «Арифметизація» роботи протоколу ZK-STARK.

Створено програмну реалізацію першого етапу роботи протоколу STARK «Арифметизація» шляхом написання двох програм КПСІВРП-STARK-01 та КПСІВРП-STARK-02 на мові програмування C^{++} на «домашньому комп'ютері». Програма КПСІВРП-STARK-01 використовує метод Гауса для пошуку коренів інтерполяційного полінома. Програма КПСИВРП-STARK-02, яка використовує метод обернених матриць для

пошуку коренів інтерполяційного полінома.

Тестування першого етапу «Арифметизація» протоколу ZK-STARK для оцінки тривалості роботи програми від розміру матриці пошуку коренів інтерполяційного поліному виявило, що Програма КПСІВРП-STARK-01 потребує меншого об'єму пам'яті, ніж програма КПСІВРП-STARK-02. Апроксимація за вхідними даними обох програм в Mathcad виявила що: залежність об'єму пам'яті, яку потребують обидві програми при роботі, від розміру матриці пошуку коренів інтерполяційного полінома має нелінійний характер.

Ефективність створених Програм з урахуванням можливостей «домашнього комп'ютеру» та міжнародного досвіду вирішення СЛАР з великою кількістю невідомих та рівнянь, показало, що залежність часу роботи Програми КПСІВРП-STARK-01 та КПСІВРП-STARK-02 від розміру матриці пошуку коренів інтерполяційного полінома має поліноміальний характер, який можна виразити якісною залежністю $f(x) = x^3$. Залежність об'єму пам'яті, які потребують дві програми при роботі, від розміру матриці пошуку коренів інтерполяційного полінома має також нелінійний характер. Залежність об'єму пам'яті, яку потребує матриця пошуку коренів інтерполяційного полінома від її розміру насить також нелінійний характер. Більшу частину часу роботи першого етапу «Арифметизація» протоколу ZK-STARK займає саме розрахунок коренів інтерполяційного поліному.

Вирішення СЛАР з великою кількістю невідомих та рівнянь – це дуже складна обчислювальна задача. Тому, якщо діяльність протоколу STARK зумовлює рішення СЛАР, такого розміру, які наведені у таблицях чи більшого розміру, то, наразі, має сенс розподілити обчислення між комп'ютерами обчислювальної мережі задля прискорення роботи першого етапу роботи протоколу або збільшити потужність окремих вузлів мережі.

Для більш швидкої реалізації протоколу слід використовувати метод Гауса для пошуку коренів інтерполяційного полінома, ніж метод обернених матриць.

Для оброблення дуже великих обсягів даних, протокол STARK має бути реалізований на обладнанні, яке значно потужніше, ніж те, на якому було проведене тестування протоколу, наприклад, на квантовому комп’ютері чи мережі звичайних комп’ютерів, бо залежність часу роботи протоколу та обсягу пам’яті, яку він займає при роботі від розміру матриці пошуку коренів інтерполяційного поліному носить нелінійний характер.

Ключові слова: ПРОТОКОЛ, ZK-STARK, ARETHMETIZATION/АРИФМЕТИЗАЦІЯ, МОДЕЛЮВАННЯ, ПРОГРАМА, РЕАЛІЗАЦІЯ, МОВА ПРОГРАМУВАННЯ C^{++}, ТЕСТУВАННЯ, БЛОКЧЕЙН, ЕФЕКТИВНІСТЬ.

ABSTRACT

The explanatory note contains 88 pages, 5 figures, 8 tables, 10 appendices, 26 sources.

The purpose of the qualification work is to study the modeling and testing of low-degree in cryptographic protocols of proof with zero disclosure by software implementation of the first stage of Arethmetization of the ZK-STARK protocol.

The object of study of the qualification work is the process of ensuring the computational integrity of the Blockchain network transaction.

The subject of development is ZK-STARK - a cryptographic protocol that uses public proofs with zero disclosure.

Research methods: mathematical information from the Theory of Interpolation, Group Theory, Number Theory; information about Fibonacci numbers; information about the Euler function; generating element of a group; cyclic groups; Lagrange interpolation polynomial and the sequence of calculations.

For software implementation, the C^{++} programming language, Visual Studio 2022 application development environment, and the NTL library for developing applications that use Number Theory were chosen.

Results of the work. In this work, we have modeled and tested a low degree of proof in cryptographic protocols with zero disclosure by software implementation of the first stage «Arithmetization» of the ZK-STARK protocol.

A software implementation of the first stage of the STARK protocol «Arithmetization» was created by writing two programs KPSIVRP-STARK-01 and KPSIVRP-STARK-02 in the C^{++} programming language on a «home computer». The program KPSIVRP-STARK-01 uses the Gauss method to find the roots of an interpolation polynomial. Program KPSIVRP-STARK-02, which uses the method of inverse matrices to find the roots of an interpolation polynomial.

Testing of the first stage «Arithmetization» of the ZK-STARK protocol to estimate the duration of the program operation depending on the size of the matrix

for finding the roots of the interpolation polynomial revealed that the program KPSIWRP-STARK-01 requires less memory than the program KPSIWRP-STARK-02. The approximation by the input data of both programs in Mathcad revealed that: the dependence of the amount of memory required by both programs on the size of the matrix for finding the roots of the interpolation polynomial is nonlinear.

The effectiveness of the created programs, taking into account the capabilities of a «home computer» and international experience in solving SLARs with a large number of unknowns and equations, showed that the dependence of the operating time of the KPSIVRP-STARK-01 and KPSIVRP-STARK-02 programs on the size of the interpolation polynomial root search matrix is polynomial in nature, which can be expressed by the quality dependence f(x) = x3. The dependence of the amount of memory required by two programs during operation on the size of the root search matrix of an interpolation polynomial is also nonlinear. The dependence of the amount of memory required by the matrix for finding the roots of an interpolation polynomial on its size is also nonlinear. The calculation of the roots of the interpolation polynomial takes up most of the time of the first stage of the ZK-STARK protocol, Arithmetization.

Solving a PDE with a large number of unknowns and equations is a very complex computational task. Therefore, if the activity of the STARK protocol leads to the solution of SLARs of the size shown in the tables or larger, then it makes sense to distribute the computation between computers in the computer network to speed up the first stage of the protocol or to increase the power of individual network nodes.

For faster protocol implementation, the Gaussian method should be used to find the roots of the interpolation polynomial rather than the inverse matrix method.

To process very large amounts of data, the STARK protocol should be implemented on hardware that is much more powerful than the one on which the protocol was tested, for example, on a quantum computer or a network of ordinary

computers, since the dependence of the protocol's running time and the amount of memory it occupies during operation on the size of the interpolation polynomial root search matrix is nonlinear.

Keywords: PROTOCOL, ZK-STARK, ARETHMETIZATION/ARITHMETIZATION, MODELING, PROGRAM, IMPLEMENTATION, C^{++} PROGRAMMING LANGUAGE, TESTING, BLOCKCHAIN, EFFICIENCY.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА СИМВОЛІВ 10
ВСТУП 11
1 ОГЛЯД ЛІТЕРАТУРИ 17
2 ТЕОРЕТИЧНА РЕАЛІЗАЦІЯ ЕТАПУ ПРОТОКОЛУ STARK «АРИФМЕТИЗАЦІЯ» 26
3 ПРОГРАМНА РЕАЛІЗАЦІЯ ЕТАПУ ПРОТОКОЛУ STARK «АРИФМЕТИЗАЦІЯ» 31
3.1 Опис та умови функціонування першої Програми реалізації протоколу STARK 31
3.1.1 Загальні відомості. Позначення та найменування програми 31
3.1.2 Програмне забезпечення необхідне для роботи програми 31
3.1.3 Мова програмування програми 31
3.1.4 Функціональне призначення програми 31
3.1.5 Опис логічної структури програми 32
3.1.6 Структура програми з описом функцій складових частин і зв'язку між ними 32
3.1.7 Зв'язки програми з іншими програмами Програма не пов'язана з іншими програмами Програми працює в режимі Offline 33
3.1.8 Технічні засоби, що використовуються 34
3.1.9 Виклик і завантаження 34
3.1.10 Опис вхідних і вихідних даних. 34
3.2 Опис та умови функціонування другої програми реалізації протоколу STARK 35

3.2.1 Загальні відомості. Позначення та найменування програми 35
3.2.2 Програмне забезпечення необхідне для роботи програми 35
3.2.3 Мова програмування програми 35
3.2.4 Функціональне призначення програми 35
3.2.5 Опис логічної структури програми 36
3.2.6 Структура програми з описом функцій складових частин і зв'язку між ними 37
3.2.7 Зв'язки програми з іншими програмами 38
3.2.8 Технічні засоби, що використовуються 38
3.2.9 Виклик і завантаження 38
3.2.10 Опис вхідних і вихідних даних 38
4 ОЦІНКИ ЕФЕКТИВНОСТІ РОБОТИ ПРОГРАМИ ПЕРШОГО ЕТАПУ РОБОТИ ПРОТОКОЛУ STARK НА ДОМАШНЬОМУ КОМП’ЮТЕРІ 40
4.1 Порівняльна оцінка часу роботи програм КПСИВРП-STARK-01 КПСИВРП-STARK-02 в залежності від вхідних даних 40
4.2 Порівняльна оцінка залежності об’єму пам’яті, яку потребує програми КПСИВРП-STARK-01 й КПСИВРП-STARK-02 при роботі 42
4.3 Порівняльна оцінка залежності об’єму пам’яті, яку потребує матриця пошуку коренів інтерполяційного полінома від її розміру на «домашньому комп’ютері» 44
ВИСНОВКИ 50
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 53
ДОДАТКИ 57

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА СИМВОЛІВ

КПСІВРП-STARK	-	Консольна Програма для створення і візуалізації роботи протоколу STARK
СЛА	-	Система Лінійних Алгебраїчних Рівнянь
ZK-STARK	-	Zero-Knowledge Scalable Transparent Argument of Knowledge
ZK-SNARK	-	Zero-Knowledge Succinct Non-interactive Argument of Knowledge

ВСТУП

Сучасний стан проблеми, світові тенденції. З появою Інтернету світ почав стрімко змінюватися, до того ж темп змін постійно зростає, тому проблема збереження та обробки даних стає все більш актуальною. Стара модель збереження даних передбачає довіру клієнта до того, хто зберігає дані, що він їх не змінює, не оприлюднює та не блокує до них доступу без дозволу власника даних – це модель «старого світу». Наразі, також виявили, що устрій традиційних централізованих систем зберігання даних не дозволяє користувачам бути впевненими у цілісності та достовірності даних які зберігаються – усе, що залишається клієнту, це довіряти базі даних (централізованого виду). У сучасному світі користувачі все частіше хочуть не просто довіряти, а мати можливість перевірити, особливо, якщо мова йде про володіння активами у банку (які належать банку, а не клієнту банка) [1-2]. До того ж, маніпуляції активами за допомогою транзакцій теж виконуються банком, а не клієнтом, тому цілісність та конфіденційність транзакції залежать від банку, а не від його клієнту. Наприклад, фінансова криза 2008 року змусила людей замислитися, що, можливо, сліпа довіра і віра у фінансові інститути – це не те, що потрібно клієнту банку [3]. Тому саме технологія децентралізованого збереження та обробки даних Блокчейн виявилася вирішенням цієї «головоломки», якої не вистачало та яке сприяє створенню криптографічно захищеної системи, яка не потребує довіри і не потребує участі централізованих систем зберігання та обробки даних. Так через безліч спроб з'явилися криптовалюти, що використовують цю технологію, наприклад, Біткоїн та Ефіріум, які дозволяють зберігати активи клієнтів на технічному обладнанні клієнтів. Тому активи клієнта наразі належать саме клієнту та відповідальність за конфіденційність, цілісність та доступність активів наразі лежить, перш за все, на клієнті та, по-друге, на його технічному обладнанні, а безпека активів та безпека маніпуляцій

активами в мережі гарантується сучасними протоколами захисту цілісності транзакцій [1].

Наразі, ця нова модель, або модель «нового світу» реалізована в Інклюзивних Блокчейнах і виражена в девізі «Не вір, а перевіряй» і вже реалізована в криптовалюті Ефіріум у вигляді протоколу ZK-SNARK [1, 4]. Обчислювальна цілісність транзакції такого Блокчейна гарантується всеосяжною підзвітністю: вузол зі стандартною обчислювальною установкою (наприклад, ноутбук, під'єднаний до Інтернету) може перевірити цілісність усіх транзакцій клієнта в мережі Блокчейн. Для перевірки правильності транзакції в централізованих системах, зазвичай, використовують метод відтворення цієї транзакції в повному вигляді, але це тягне за собою втрату приватності транзакції і масштабованості обчислень. Для оптимізації перевірки правильності транзакції технологія Блокчейн використовує системи доказів з нульовим розголошенням. На сьогоднішній день, вони є визнаним інструментом для вирішення проблеми конфіденційності та масштабованості в мережі Блокчейн. Цей підхід реалізовано в протоколі SNARK, STARK та інших протоколах із «нульовим знанням» [1].

Наразі саме STARK - це новий, перспективний, стійкий до квантових атак, особливий вид системи доказів, яка забезпечує масштабованість, прозорість і приватність транзакції, і заснований на спеціальній криптографії [1, 5]. Властивості масштабованості та прозорості дають змогу STARK, швидко і без довіри третьої сторони, перевіряти властивість обчислювальної цілісності транзакції. Сенсом існування протоколу STARK є лаконічна і прозора перевірка обчислювальної цілісності транзакції [1]. Перший етап роботи протоколу називається «Арифметизацією». Він полягає у створенні сліду виконання транзакції, створенні полінома за слідом виконання та перевірці цього полінома на маленький ступінь. Цей поліном буде маленького ступеня, якщо і тільки якщо, слід виконання правильний, і відповідно, правильні й дані, за якими було створено слід обчислення.

Практично вирішені задачі та застосування. Елі Бен-Сассон, Іддо

Бентов, Інон Хореші та Михайло Рябзєв написали перші статті, що описують протокол STARK, ще у 2018 році [6]. На даний момент існує єдина відома утиліта ZK-STARK, яка розробляється компанією StarkWare, створеною дизайнерами ZK-STARK. Метою є розробка тестового шару, який дозволить використовувати технологію на Блокчейні, децентралізованих біржах та багато де інше. На сьогодні, протокол STARK не був широко протестований, вивчений і досі широко не застосований в жодній реальній виробничій системі, навіть у світі криптовалют [5]. Але компанія StarkWare вже запустила альфа-версію блокчену Layer 2 Rollup Network, де використовуються ZK-STARK для криптовалюти Ефіріум [6].

Взагалі, основне застосування систем перевірки знань, таких як ZK-STARK, зосереджено на створенні високозахищених і приватних систем. Систем, де існує повна децентралізація інформації, і доступ до неї можливий лише за дотримання низки чітко виражених умов. Умови, які також важко досягти нетрадиційними способами, наприклад, зломом [5].

Системи з децентралізацією інформації включають такі системи, як криптовалюти, де використання криптовалют не лише забезпечує безпеку мережі, а й захищає користувачів, забезпечує їм конфіденційність та анонімність залежно від ситуації [5]. І саме в останньому випадку ZK-SNARK особливо вирізняється з-поміж інших протоколів схожого призначення, тому що, він добре підходить для забезпечення приватності та анонімності, не розкриваючи інформацію жодним чином, але водночас залишаючи інструмент для підтвердження транзакції однозначним і детермінованим. Іншими словами, ZK-STARK не розкриває інформацію, яку шифрує, але ви завжди можете перевірити її достовірність, незважаючи ні на що [5].

Можливо застосування ZK-STARK - підвищення маштабованості Блокчейна, дозволяючи криптографічним тестам займати менше місця. У таких криптовалютах, як Біткоїн, де розмір блоку обмежує кількість транзакцій, які можуть бути оброблені за секунду, це життєво важливо. При

меншому розмірі криптографічних тестів транзакції також займають менше місця, і в кожен блок може поміститися більше транзакцій. Ефект стає сильнішим, коли його застосовують до тисяч транзакцій, а разом із ним поліпшується масштабованість [5]. Однак це лише частина вирішення проблеми масштабованості, оскільки найнезначніші криптографічні тести не призведуть до різкого збільшення продуктивності Блокчейна.

Ще одним можливим застосуванням систем такого типу могла б стати, наприклад, повністю зашифрована і захищена система потокового мовлення, захищена авторськими правами. Для цього не знадобляться нинішні системи шифрування, що базуються в основному на симетричній криптографії. Електронні системи голосування також отримують значні переваги від використання систем такого типу. Це відбувається тому, що вони дозволяють виборцю віддати свій голос, він може бути перевірений, але ми жодним чином не дізнаємося, хто його віддав [5]. Потенціал застосування ZK-STARK величезний.

Протокол STARK - це нова технологія, яка ще не має широкого застосування в практиці. Однак, на даний момент, деякі прогалини і проблеми, пов'язані з його застосуванням, вже виявлені: обчислювальна складність, можлива погана сумісність з іншими протоколами, стійкість до квантових атак і інші. Тому у нашій роботі ми вирішили доповнити висвітлення проблеми, пов’язаної з обчислювальною складністю та запропонувати варіанти вирішення цієї проблеми.

Актуальність роботи. У протоколі STARK Перевіряльник і Верифікатор STARK є найшвидшими в класі (перевершуючи всі теоретико-числові конструкції) і досягають значно більшого масштабу, ніж конкуренти, спираючись при цьому на меншу кількість і безпечніші криптографічні припущення. Зокрема, вони пост-квантово безпечні та прозорі, що означає, що вони не використовують небезпечні «криптографічні токсичні відходи» в процесі налаштування і не мають пов'язаних із ними «люків», які можуть поставити під загрозу їхню безпеку [7].

Протокол STARK є новим і складним у реалізації, і він має потенціал розв'язати головну проблему багатьох сучасних протоколів захисту інформації - стійкість до атак квантових комп'ютерів, тому його застосування потребує додаткових досліджень і розробки. А тому поетапне вивчення та моделювання протоколу є актуальним завданням для його впровадження та поширення з метою захисту властивості обчислювальної цілісності транзакції в сучасних мережах Блокчейн.

Новизна роботи. Результати тестування двох програмних реалізацій першого етапу Arethmetization протоколу STARK на мові програмування C^{++} свідчать, що для більш швидкої реалізації протоколу найефективнішим є використовування методу Гауса для пошуку коренів інтерполяційного полінома, ніж методу обернених матриць.

Показано, що залежність часу роботи першого етапу протоколу Arethmetization та обсягу пам’яті, яку він займає при роботі, від розміру матриці пошуку коренів інтерполяційного поліному, носить нелінійний характер.

Встановлено, що більшу частину часу роботи першого етапу Arethmetization протоколу ZK-STARK займає саме розрахунок коренів інтерполяційного поліному.

Протокол STARK є новим і складним у реалізації, і він має потенціал розв'язати головну проблему багатьох сучасних протоколів захисту інформації - стійкість до атак квантових комп'ютерів, тому його застосування потребує додаткових досліджень і розробки. А тому поетапне вивчення та моделювання протоколу є актуальним завданням для його впровадження та поширення з метою захисту властивості обчислювальної цілісності транзакції в сучасних мережах Блокчейн.

Галузь застосування результатів. Ця робота є корисною задля отримання знань щодо сутності протоколу «нульового знання» STARK, його функціонування та ефективності його використання.

Ця робота допомагає впроваджувати технологію доказів із нульовим

знанням у реальні мережі Блокчейн, зокрема впроваджувати технологію ZK-STARK, розкриваючи її сутність та ефективність на конкретних реалізаціях на мові програмування C^{++}.

Особливості використання протоколу ZK-STARK, що показані у роботі дозволяють рекомендувати його до використання у мережі Блокчейн, яка набула сьогодні величезного попиту у фінансовій сфері діяльності.

Мета роботи. Метою роботи є вивчення моделювання та тестування низького ступеня у криптографічних протоколах доказу із нульовим розголошенням шляхом програмної реалізації першого етапу Arethmetization роботи протоколу ZK-STARK.

Об'єкт. Процес забезпечення обчислювальної цілісності транзакції мережі Блокчейн.

Предмет. ZK-STARK – криптографічний протокол, який використовує публічні докази з нульовим розголошенням.

Задля цього потрібно виконати такі *завдання:*

1) Визначити основні риси функціонування та відмінності й спільні риси схожих криптографічних протоколів STARK і SNARK.

2) Визначити особливості функціонування першого етап роботи протоколу STARK.

3) Створити теоретичну та програмну реалізацію Arethmetization першого етапу роботи протоколу STARK.

4) Провести тестування першого етапу Arethmetization протоколу ZK-STARK для оцінки залежності тривалості роботи програми від розміру матриці пошуку коренів інтерполяційного поліному.

5) Зробити аналіз результатів тестування першого етапу Arethmetization протоколу ZK-STARK та надати рекомендації щодо найбільш ефективної роботи реалізації протоколу.

1 ОГЛЯД ЛІТЕРАТУРИ

Історія протоколу ZK-STARK починається 2 липня 2018 року. Тоді Ethereum Foundation виділив компанії StarkWare 2-річний грант для створення STARK-дружньої хеш-функції (SFH), яка повинна була би використовуватися в прозорих і правдоподібних постквантових системах доказу в мережі Блокчейн [8]. З того часу актуальним залишається питання програмної реалізації протоколу Zero-Knowledge STARK (ZK-STARK). Цим питанням і сьогодні займається компанія StarkWare тому, що застосування протоколу STARK дає перевагу перед іншими протоколами, які використовуються для захисту обчислювальної цілісності транзакцій в мережі Блокчейн. Фахівці StarkWare показали, що цей протокол має низку переваг перед іншими протоколами у захисті інформації в мережі Блокчейн [5].

Насьогодні мережа Блокчейн стала першим кроком у впровадженні протоколу ZK-STARK до життя. Відомо, що Блокчейн - це технологія обробки, зберігання інформації та ідентифікації клієнтів. Дослівно з англійської Блокчейн (blockchain) перекладається, як «ланцюжок блоків», а сама технологія була запропонована в 2008 році Сатоши Накамато (псевдонім людини або групи людей) [9].

Технологія Блокчейн визначається головними особливостями:

1) прозорість - в блокчейні зберігаються дані про всі проведені операції за всю історію створення системи (криптовалюти)

2) стабільність - ви не можете видалити або замінити інформацію «заднім числом», а тільки здійснити нову угоду

3) незалежність - інформація зберігається не на одному центральному сервері, а на безлічі комп'ютерів учасників мережі.

Іншими словами Блокчейн - це система (база даних), яка є сукупною реалізацією таких основних ідей [9]:

1) Транзакції (eng. Transactions)

2) Незмінні бухгалтерські реєстри (eng. Immutable ledgers)

3) Децентралізовані вузли мережі (eng. Decentralized peers)

4) Процеси шифрування (eng. Encryption processes)

5) Механізм консенсусу (eng. Consensus mechanisms)

6) Смарт-контракти (eng. Optional Smart Contracts)

Наразі є можливим використовувати ZK-STARK протокол у захисті транзакцій мережі Блокчейн. Відомо, що для перевірки транзакції на автентичність, технологія Блокчейн використовує консенсусні моделі захисту автентичності транзакцій, щоб дозволити групі користувачів, які не мають взаємної довіри один до одного, працювати разом. Фахівці [10] стверджують, що консенсус по суті є процедурою ухвалення рішення. Його мета – забезпечити те, щоб всі учасники мережі погодили свій поточний стан після додавання нової інформації, блоку даних або пакета транзакцій. Інакше кажучи, консенсус-протокол гарантує те, що ланцюг правильний, і дає стимули для того, щоб учасники залишалися чесними. Це важлива структура для запобігання ситуації, коли хтось контролює всю систему, і вона гарантує те, що всі дотримуються правил мережі, тобто це механізм, який дозволяє перевіряти автентичність транзакції «за більшістю голосів вузлів мережі за автентичність конкретної транзакції» та додавати її до Блокчейну. Одним із таких засобів реалізації консенсусу в мережі Блокчейн можуть бути протоколи перевірки автентичності транзакції STARK або, наприклад, SNARK, які може використовувати алгоритм консенсусу «PoWeight» та «PoB» [11, 12]. Наприклад, Zcash – це перше доступне застосування ZK-SNARKs алгоритму досягнення консенсусу в мережі Блокчейн [13].

STARK і SNARK це - схожі конкуруючі технології, які прагнуть до досягнення консенсусу між вузлами мережі Блокчейн, але підходять до досягнення консенсусу в мережі по-різному [6]. Обидва протоколи STARK і SNARK мають низку суттєвих відмінностей, але один спільний для них етап роботи, який називається «Арифметизація», що створює докази і перевіряє їх

на хибність або достовірність для того, щоб не приєднати або приєднати транзакцію до Блокчейну відповідно. Для більшого розуміння суті роботи протоколу STARK необхідно розглянути ще й протокол SNARK. На основі аналізу схожих і відмінних рис протоколів, можливе більш глибоке розуміння роботи протоколу STARK.

З початку слід почати з визначення абревіатури в назві протоколу STARK. Його абревіатура дослівно означає «масштабований прозорий аргумент знання, що масштабується». Абревіатура розшифровується таким чином, що протокол є «масштабованим», коли він не вимагає процедуру «препроцесингу» даних (процедура, що готує вже створені публічні параметри протоколу для того, щоб почав працювати сам протокол), яку, наприклад, вимагає протокол SNARK [14]. Термін «прозорий» означає, що протокол STARK не потребує «довіреного набору» даних (використовується для створення публічних параметрів протоколу), якого, наприклад, потребує протокол SNARK. В обох протоколах STARK і SNARK термін «аргумент знання» відноситься до криптографічного доказу, що використовується для встановлення автентичності або права власності на частину даних або транзакцію в мережі Блокчейн. Це спосіб демонстрації, що одна зі сторін, які взаємодіють у мережі Блокчейн, має знання про конкретний фрагмент інформації, не розкриваючи саму інформацію. Це означає, що протокол використовує технологію «нульового знання» або принцип «нульового розголошення».

Структура всіх протоколів із нульовим розголошенням, до яких відносяться протокол ZK-STARK та ZK-SNARK, зумовлює наявність сутностей «Прувер» та «Верифікатор», які у процесі взаємодії доводять, чи спростовують доказ. У протоколі Zero-Knowledge STARK (ZK-STARK) Прувер та Верифікатор виконують різні ролі:

- «Прувер» є стороною, яка намагається довести правильність певного твердження або ствердження про обчислення без розкриття конфіденційної інформації. Прувер зазвичай має конкретний набір даних або обчислень, які

він бажає довести, і його ціль - переконати Верифікатора в цій правильності. Прувер виконує деякі обчислення, генерує доказ, який надсилається Верифікатору.

- «Верифікатор» є стороною, яка перевіряє доказ, наданий Прувером. Верифікатор не має доступу до конфіденційної інформації, яку Прувер бажає захистити, але він може перевірити, чи є доказ вірним. Його мета - визначити, чи можна довіряти результатам, наданим Прувер. Веріфікатор проводить перевірку доказу, використовуючи певні математичні алгоритми і правила, щоб переконатися, що Прувер не обманює і представляє вірну інформацію.

Самі протоколи з нульовим розголошенням (префікс ZK у назві протоколу ZK-STARK) - це криптографічні протоколи, які дозволяють Пруверу переконати Верифікатора, що він володіє певними знаннями, не розкриваючи ці знання. Більш формально, нехай R - це відношення. Тоді протокол ZK протокол для відношення R дозволяє Пруверу переконати верифікатора у тому, що він знає свідка w для деякого заданого твердження s, так що (s;w) є R, не розкриваючи нічого про саме w. Наприклад, Прувер може довести що хеш s є хешем деяких добре сформованих даних w. (У випадку R складається з усіх пар (s;w), де s = hash(w) і w є добре сформованим) [15].

Протоколи ZK є важливим будівельним блоком у криптографії, оскільки вони можуть допомогти забезпечити чесної поведінки від потенційно зловмисних сторін. Наприклад, доказ може забезпечити гарантію того, що сторона уповноважена виконувати певні дії або мати доступ до певної конфіденційної інформації [15].

Безпека протоколів ZK виражається через властивості повноти, достовірності, нульового знання та доказу знання. Повнота забезпечує коректну роботу протоколу, якщо і той, хто доводить, і той, хто перевіряє, чесно слідують протоколу. Обґрунтованість гарантує, що для «неправильних» твердженнях (тобто, без свідків) Прувер може переконати Верифікатора лише з невеликою ймовірністю. Доведення знань гарантує, що

будь-який Прувер, який успішно переконує Верифікатора, насправді знає свідка. Нульове знання встановлює, що будь-який Прувер, який обманює Верифікатора не може нічого дізнатися про свідка під час виконання протоколу [15].

Взагалі, на думку фахівців [14, 16] різниця між абревіатурами протоколів STARK та SNARK полягає в тому, що:

1) Протокол STARK «маштабований», а протокол SNARK «лаконічний» і вимагає процедуру «препроцесингу».

Властивість «лаконічність» означає, що перевірка доказів проходить експоненціально швидше, ніж виконуються обчислення в самому протоколі.

2) Протокол STARK «прозорий», а протокол SNARK «неінтерактивний» і вимагає процедуру зі створення «довіреного набору». Неінтерактивність означає, що в неінтерактивному доказі достовірності аргументу, Перевіряючому достатньо одного повідомлення (яке називається доказом) для Верифікатора, щоб Верифікатор перевірив достовірність цього повідомлення, з метою прийняття його чи ні для утвердження чи спростування достовірності транзакції відповідно, у мережі Блокчейн (Перевіряючий та Верифікатор - два об'єкти, засобами, за допомогою яких здійснюється доказ правильності транзакції в протоколах STARK та SNARK).

Якщо сумувати наведені дані [6, 17-19], то суттєву різницю між цими протоколами є можливим зобразити у вигляді таблиці 1.

Таблиця 1 - Головні переваги та недоліки алгоритмів STARK та SNARK

Спільні риси протоколів	SNARK	STARK
1	2	3
Складність алгоритму Прувера	$O(n * \log(n))$	$O(n * poly\log(n))$

Продовження таблиці 1

1	2	3
Складність алгоритму Верифікатора	$\sim O(1)$	$O(poly\log(n))$
Комунікаційна складність	$\sim O(1)$	$O(poly\log(n))$
Оцінка розміру одиничної транзакції	транзакція 200 Б, ключ 50 МБ	45 КБ
Оцінка розміру десяти тисяч транзакцій	транзакція 200 Б, ключ 500 ГБ	135 КБ
Вартість газу, який потребує віртуальна машина Ефіріуму	$\sim 600k(Groth16)$	$\sim$2,5 мілліони (не залежить від конкретного алгоритму)
Потрібність у довіреному наборі	так	ні
Пост-квантова стійкість	ні	так
Крипто-допущення	сильний	стійкий до геш-колізій
Масштабованість	менш масштабований ніж STARK	більш масштабований ніж SNARK
Розмір доказу	288 Б, тобто $\sim O(1)$	45 КБ – 200 КБ, тобто $O(poly\log(n))$
Час доказу	2,3 с	1,6 с
Час перевірки доказу	10 мс	16 мс
Тип криптографії	криптографія з білінійними паруваннями	геш-функції

За таблицею є можливим визначити головні переваги та недоліки алгоритмів STARK та SNARK. І SNARK, і STARK мають свої переваги та

недоліки, і вибір між ними залежить від конкретних вимог користувача [20]. Якщо далі порівняти самі протоколи, то можна констатувати, що ZK-STARKS і ZK-SNARK є різновидами неінтерактивних протоколів доказів з нульовим розголошенням. Однак вони відрізняються з різних точок зору [19]:

1) STARK, у порівнянні зі ZK-SNARK, забезпечують більшу масштабованість, прозорість і безпеку Блокчейнів. На відміну від більшості SNARK, STARK покладаються на хеш-функції, які вважаються квантово-стійкими. Використання хеш-функцій для технології STARK дає певні переваги, наприклад, квантову стійкість. STARK може запропонувати підвищену безпеку завдяки відсутності необхідності налаштування «довіреного набору», однак ZK-STARK мають більший розмір доказу і потребують більше часу для перевірки доказу, ніж SNARK [17-23].

2) Як наслідок, ZK-SNARKs мають набагато більшу підтримку, ніж ZK-STARKs, якщо розробник почне використовувати технології нульового знання. Крім того, очікується, що ZK-SNARK використовуватимуть лише 24% газу, а це означає, що використання ZK-SNARK для транзакцій буде значно дешевшим для кінцевого користувача [19]. SNARK були розроблені на шість років раніше, ніж STARK, що допомогло їм отримати перевагу в плані впровадження [20]. Крім того, ZK-STARK все ще перебувають на початковій стадії і мають ще довго доводити свою ефективність розробникам та іншим зацікавленим сторонам в екосистемі Блокчейну [19]. Також ZK-STARK поки-що масово не використовується, але у світі криптовалют ці протоколи мають великий потенціал і можуть стати новаторським шляхом до широкого впровадження [13].

Взагалі, слід зазначити, що можуть існувати системи перевірки автентичності транзакцій, які одночасно є SNARK і STARK. Це означає, що «неінтерактивні» STARK можуть бути SNARK, а «прозорі» SNARK з «лаконічним» налаштуванням можуть бути STARK. Існують також системи доказу, які не є ні SNARK, ні STARK, але корисні завдяки властивості

«нульового знання» (тобто не розголошення суті транзакції перед перевіркою доказу про істинність транзакції). Як приклад можна назвати Bulletproofs [24].

Ще одним прикладом можуть бути SNARK, яким не потрібен «довірений набор», наприклад Plonky2. Plonky2 – це рекурсивний SNARK. Він поєднує в собі властивості STARK – швидкі докази і відсутність довіреного набору, і властивості SNARK – підтримка рекурсії і низька вартість верифікації в Ethereum [25].

По-перше протокол STARK має три головних етапу роботи:

1) Генерація доказу - на цьому етапі створюється доказ, який показує, що певні обчислення були виконані правильно. Доказ будується за допомогою методу інтерактивної перевірки доказів (Interactive Proof Protocol).

2) Верифікація доказу - на цьому етапі отриманий доказ перевіряється, щоб переконатися, що він коректний і доводить необхідне твердження.

3) Побудова довірчого інтервалу - на цьому етапі використовується технологія ZK-STARK, щоб створити довірчий інтервал для доказу. Цей інтервал дає змогу переконатися в коректності доказу з високою ймовірністю, не виконуючи повну верифікацію.

На відміну від протоколу STARK, SNARK не має третього етапу роботи, оскільки він використовує інший підхід до створення доказів, який дає змогу досягти високого ступеня стиснення і швидкості роботи без необхідності взаємодії між Прувером і Верифікатором.

По-друге, виходячи із вище зазначеного матеріалу щодо протоколів STARK та SNARK є можливим визначити їх головну спільну рису:

Це - перший етап роботи протоколів STARK [2] і SNARK – «Арифметизація» [13], який є головною спільною рисою двох протоколів. Етап складається з двох кроків: перший - генерація сліду виконання та поліноміальних обмежень для створеної користувачем транзакції, другий - перетворення цих двох об'єктів на один поліном низького ступеня [2], який

буде поліномом низького ступеня лише тоді, коли слід виконання правильний, і, відповідно, правильними є і дані, за якими було створено слід обчислення [26].

У протоколах STARK і SNARK цей етап роботи протоколу реалізовано по-різному. SNARK реалізує етап «Арифметизація» за допомогою методу r1cs, а STARK - за допомогою методу AIR [14].

У зв’язку з швидким розвитком квантових комп’ютерів та їх розповсюдженням, протокол STARK стане надійною заміною усім протоколам, які не є стійкими до атак зі сторони квантових комп’ютерів (наприклад, протокол SNARK).

Загалом протокол STARK є новим і складним у реалізації, і він має потенціал розв'язати головну проблему багатьох сучасних протоколів захисту інформації - стійкість до атак квантових комп'ютерів, тому його застосування потребує додаткових досліджень і розробки. А тому поетапне вивчення та моделювання протоколу є актуальним завданням для його впровадження та поширення з метою захисту властивості обчислювальної цілісності транзакції в сучасних мережах Блокчейн.

2 ТЕОРЕТИЧНА РЕАЛІЗАЦІЯ ЕТАПУ ПРОТОКОЛУ STARK «АРИФМЕТИЗАЦІЯ»

З урахуванням того, що етап «Арифметизації» має на увазі використання «Кодів корекції помилок» план здійснення цього етапу може виглядати таким чином [24]:

1) переформулювати слід виконання в поліноміальний вигляд.

2) розширити слід виконання на більш велику область.

3) перетворити слід виконання, використовуючи поліноміальні обмеження, на ще один поліном, який гарантовано матиме низький ступінь, якщо і тільки якщо слід виконання правильний.

На основі наявної інформації можна змоделювати роботу протоколу на конкретному прикладі.

Нехай за умовою задачі нам дано:

Поле Z_{13} = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12} і воно складається з $|Z_{13}| = 13$ чисел зі складанням і множенням за модулем 13. Це поле має мультиплікативну підгрупу G, таку, що G=6 і породжувальний елемент групи g = 4. Існування такої підгрупи гарантоване, оскільки 6 ділить розмір цієї групи (який дорівнює 12) без залишку [24].

Припустімо, що розглянуте твердження про необхідність перевірки обчислювальної цілісності транзакції звучить так: «Перевіряючий має послідовність із 6 чисел, усі з яких це числа Фібоначчі».

Цю послідовність чисел Фібоначчі необхідно перевірити, прочитавши істотно менше 6 чисел [24].

Ця задача має такий розв'язок:

Послідовність чисел Фібоначчі формально визначається так:

$a_0 = 1$,

$a_1 = 1$,

$a_{n+2} = (a_{n+1} + a_n) \bmod 13$.

Можливо створити слід виконання, просто записавши поспіль усі 6 чисел Фібоначчі: 1, 1, 2, 3, 5, 8. Тоді, поліноміальні обмеження можуть мати вигляд [24]:

$$\begin{cases} A_0 - 1 = 0 \text{ та } A_1 - 1 = 0, \\ \forall\ 0 \le i < 4\text{: } A_{i+2} - A_{i+1} - A_i = 0, \\ A_5 - 8 = 0. \end{cases}$$

Тепер приведемо поліноміальні обмеження до поліноміального вигляду:

Рекурентне співвідношення Фібоначчі втілює набір обмежень на весь слід виконання, і його можна альтернативно інтерпретувати так [24]:

$$\forall\ 0 \le i < 4\text{: } f\left(g^{i+2}\right) - f\left(g^{i+1}\right) - f\left(g^{i}\right) = 0\,,$$

Тепер Верифікатор може створити поліном композицію за формулою [24]:

$$q(x) = \frac{f(g^{i+2}) - f(g^{i+1}) - f(g^{i})}{\prod_{i=0}^{3}(x - g^{i})}\ ,$$

Обчислення цього виразу для особливого випадку, коли ступені g утворюють підгрупу, може бути виконано так [24]:

$$x^{|G|} - 1 = \prod_{g \in G}(x - g),$$

Ця рівність є правильною, оскільки обидві сторони є многочленами ступеня |G|, корені яких у точності є елементами G [24].

А фактичний знаменник розглянутого композиційного полінома Фібоначчі можна отримати, переписавши його у вигляді [24]:

$$\frac{f(g^{i+2})-f(g^{i+1})-f(g^{i})}{\prod_{i=0}^{3}(x-g^{i})}=\frac{(w-g^{4})*(w-g^{5})*[f(g^{2}*w)-f(g^{1}*w)-f(w)]}{w^{6}-1}, \qquad (2.1)$$

де $w \in \{1, g^1, g^2, g^3, g^4, g^5\}$.

Чисельне розв'язання задачі відбувається таким чином:

1) Знаходимо інтерполяційний поліном Лагранжа для шести чисел Фібоначчі:

Поліном має вигляд:

$$f(x) = 7*x^5 + 10*x^4 + 8*x^3 + 6*x^2 + 10*x + 12.$$

Перевірка показує, що поліном порахований правильно:

При $x = g^0 = 1$:

$$f(x) = 7*1^5 + 10*1^4 + 8*1^3 + 6*1^2 + 10*1 + 12 = 1 \pmod{13}.$$

Відповідає першому числу Фібоначчі.

При $x = g^1 = 4$:

$$f(x) = 7*4^5 + 10*4^4 + 8*4^3 + 6*4^2 + 10*4 + 12 = 1 \pmod{13}.$$

Відповідає другому числу Фібоначчі.

При $x = g^2 = 3$:

$$f(x) = 7*3^5 + 10*3^4 + 8*3^3 + 6*3^2 + 10*3 + 12 = 2 \pmod{13}.$$

Відповідає третьому числу Фібоначчі.

При $x = g^3 = 12$:

$$f(x) = 7*12^5 + 10*12^4 + 8*12^3 + 6*12^2 + 10*12 + 12 = 3 \pmod{13}.$$

Відповідає четвертому числу Фібоначчі.

При $x = g^4 = 9$:

$$f(x) = 7*9^5 + 10*9^4 + 8*9^3 + 6*9^2 + 10*9 + 12 = 5 \pmod{13}.$$

Відповідає пятому числу Фібоначчі.

При $x = g^5 = 10$:

$$f(x) = 7*10^5 + 10*10^4 + 8*10^3 + 6*10^2 + 10*10 + 12 = 8 \pmod{13}.$$

Відповідає шостому числу Фібоначчі.

2) Перевіряємо властивість рекурентного співвідношення, яке має вигляд:

$$\forall x \in \{g^0, g^1, g^2, g^3\}: f(g^2x) - f(gx) - f(x) = 0.$$

Для цього, в інтерполяційний поліном для шести чисел, підставляють замість x - значення x, gx, g^2x таким чином, що утворюються поліноми:

$$f(x) = 7*x^5 + 10*x^4 + 8*x^3 + 6*x^2 + 10*x + 12,$$
$$f(g*x) = 5*x^5 + 12*x^4 + 5*x^3 + 5*x^2 + 1*x + 12,$$
$$f(g^2*x) = 11*x^5 + 4*x^4 + 8*x^3 + 2*x^2 + 4*x + 12.$$

Тоді поліном композиції має такий вигляд:

$$q(x) = \frac{f(g^{i+2}) - f(g^{i+1}) - f(g^i)}{x^6 - 1} =$$
$$= \frac{12 * x^5 + 8 * x^4 + 8 * x^3 + 4 * x^2 + 6 * x + 1}{x^6 - 1}.$$

Виходячи з рівності (2.1), знаходжу сам поліном композиції q(x)[3]:

$$q(x) = \frac{(x-9)*(x-10)*(f(g^{i+2}) - f(g^{i+1}) - f(g^{i}))}{x^6 - 1} =$$

$$= \frac{(x-9)*(x-10)*(12*x^5 + 8*x^4 + 8*x^3 + 4*x^2 + 6*x + 1)}{x^6 - 1}.$$

У тому випадку, якщо Перевіряючий отримує правильні дані та створює правильний поліном у знаменнику полінома композиції, виходить, що поліном композиції q(x) є поліномом ступеня, меншого за два, і виглядає таким чином:

$$q(x) = \frac{(12*x^7 + x^6 + x + 12)}{x^6 - 1} = 12 * x + 1\ .$$

За допомогою такого алгоритму дій Верифікатор перевіряє дані транзакції на достовірність.

На основі наявного алгоритму дій Верифікатора було написано програму мовою програмування Сі$^{++}$ і протестовано на різній кількості чисел Фібоначчі. За замовчуванням, Перевіряючий завжди вводить правильні дані.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ЕТАПУ ПРОТОКОЛУ STARK «АРИФМЕТИЗАЦІЯ»

3.1 Опис та умови функціонування першої Програми реалізації протоколу STARK

3.1.1 Загальні відомості. Найменування програми: «Консольна Програма для створення і візуалізації роботи протоколу STARK».

Позначення програми - КПСІВРП-STARK-01, або, скорочено, progstark.

Позначення: Програма КПСІВРП-STARK-01 версії 1.0.

Розробник: Аверков О.Ю.

Тел. 0989198104

E-mail: olegaverkov072@gmail.com

3.1.2 Програмне забезпечення необхідне для роботи програми. Програма КПСІВРП-STARK-01 функціонує під керуванням операційної системи не нижче Windows 10. Додатково потрібне встановлення середовища розроблення Visual Studio 2022, створення проекту та підключення до проекту бібліотеки NTL.

3.1.3 Мова програмування програми. Програму КПСІВРП-STARK-01 розроблено та реалізовано мовою програмування C^{++} у середовищі розробки Visual Studio 2022.

3.1.4 Функціональне призначення програми. «Консольна Програма для створення та візуалізації роботи протоколу STARK» призначена для створення та візуалізації роботи протоколу STARK за умови, що сутність Prover завжди працює коректно.

Ключові функції програми:

1) Введення числа, що позначає довжину адитивної групи і має сенс модуля, за яким проходять обчислення в програмі.

2) Введення числа, що позначає першообразний корінь групи.

3) Введення числа, що позначає довжину підгрупи.

4) Відображення полінома ступеня менше двох.

5) Вихід із програми.

Відомості про функціональні обмеження програми на застосуванні:

- Обсяг пам'яті, який займає програма під час функціонування, залежить від довжини підгрупи. Що більшою є довжина підгрупи, то більшою є квадратна матриця для пошуку коренів інтерполяційного полінома (залежність обсягу пам'яті, який займає програма, від довжини підгрупи має нелінійний характер). Програма КСІВРП-STARK-01 потребує для функціонування щонайменше 10 Мега байтів вільної оперативної пам'яті, за меншої кількості вільної пам'яті програма може не функціонувати.

- Програма КПСІВРП-STARK-01 вимагає введення тільки натуральних цілих чисел, можливе введення також нуля. Діапазон обмежується можливостями бібліотеки NTL.

3.1.5 Опис логічної структури програми. Програма написана і виконується з текстом програми мовою C^{++}.

Алгоритм програми та використовувані методи. Програма обробляє одержувані дані через меню керування командами в консолі користувача. Параметр, що відповідає за довжину підгрупи, вводиться окремо прямо в тіло програми.

Методи, що використовуються, ґрунтуються на можливостях апаратних модулів технічного засобу, на якому запущено Програму.

Алгоритм і методи:

1) Запуск програми

2) Створення компонентів

3) Ініціалізація компонентів користувачем

4) Обробка введених даних

5) Візуалізація даних.

3.1.6 Структура програми з описом функцій складових частин і зв'язку між ними. Структурний опис програми із її складовими можливо подати у

вигляді рисунка 3.1.

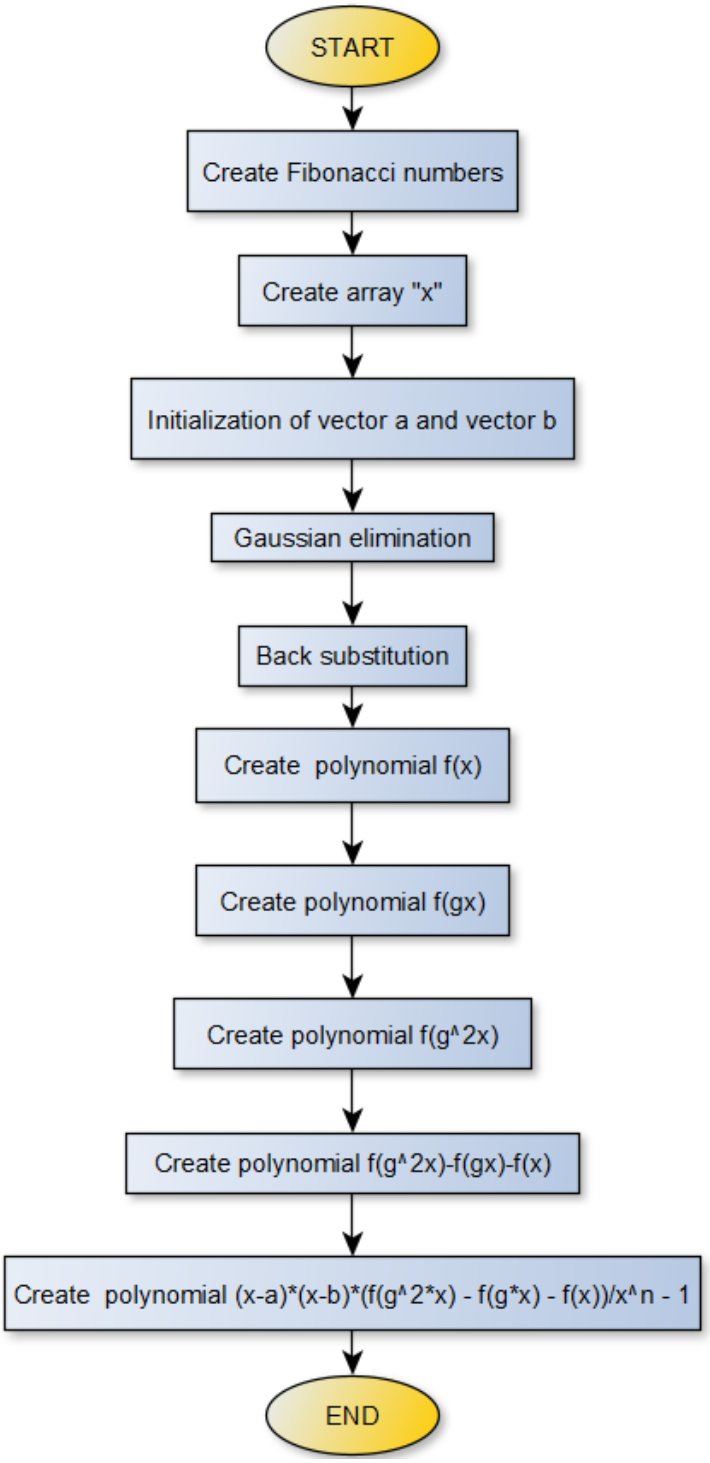

Рисунок 3.1 - Структура програми КПСІВРП-STARK-01

3.1.7 Зв'язки програми з іншими програмами. Програма не пов'язана з іншими програмами. Програми працює в режимі Offline.

3.1.8 Технічні засоби, що використовуються. Програма встановлюється на обладнанні з такими мінімальними характеристиками:

- CPU: Intel(R) Core(TM) i7-9700F CPU @ 3.00GHz 3.00 GHz;
- RAM: – 64 Gb;
- об’єм диска – 270 Gb (залежить від планованої кількості АТТ);
- операційна система Windows 10.

3.1.9 Виклик і завантаження. Після встановлення на комп'ютер, запуск програми КПСІВРП-STARK-01 проводиться вручну після завантаження операційної системи. Спочатку з архіву відкривається проект, потім відкривається файл, прикріплений до проекту.

3.1.10 Опис вхідних і вихідних даних. *Вхідні дані.* Вхідними даними для Програми є:

1) Файл конфігурації Програми

2) Команди, що вводяться користувачем за допомогою інтерфейсу командного рядка.

Організація файлу конфігурації являє собою набір текстових рядків англійською мовою в кодуванні ASCII, що відповідають командам інтерфейсу командного рядка Програми. Текстові рядки розділяються символами кінця рядка. У файлі конфігурації містяться налаштування програми за замовчуванням. Можливі зміни файлу конфігурації. Не рекомендується проводити зміни файлу конфігурації.

Команди інтерфейсу командного рядка представлені текстовими рядками англійською мовою в кодуванні ASCII. Елементи команди (слова) відповідають за змістом діям, що виконуються командою.

Список команд, підтримуваних Програмою, можна представити структурою з одним початковим вузлом. Попередньої підготовки команд інтерфейсу командного рядка не потрібно.

Вихідні дані. Вихідними даними для Програми є:

1) Файл конфігурації програми

2) Повідомлення, що передаються оператору за допомогою інтерфейсу

командного рядка.

Характер, організація, формат, опис і спосіб кодування файлу конфігурації відповідає описаним для вхідних даних у частині файлу конфігурації.

Повідомлення, що передаються оператору за допомогою інтерфейсу командного рядка, являють собою текстові рядки і символи англійською мовою в кодуванні ASCII.

Повний код програми наведено в додатку (Додаток А, Додаток Б).

3.2 Опис та умови функціонування другої Програми реалізації протоколу STARK

3.2.1 Загальні відомості. Позначення та найменування програми. Найменування програми: «Консольна Програма для створення і візуалізації роботи протоколу STARK».

Позначення програми - КПСІВРП-STARK-02, або, скорочено, progstark2.

Позначення: Програма КПСІВРП-STARK-02 версії 1.0.

Розробник: Аверков О.Ю.

Тел. 0989198104

E-mail: olegaverkov072@gmail.com

3.2.2 Програмне забезпечення необхідне для роботи програми. Програма КПСІВРП-STARK-02 функціонує під керуванням операційної системи не нижче Windows 10. Додатково потрібне встановлення середовища розроблення Visual Studio 2022, створення проекту і підключення до проекту бібліотеки NTL.

3.2.3 Мова програмування програми. Програму КПСІВРП-STARK-02 розроблено та реалізовано мовою програмування C^{++} у середовищі розробки Visual Studio 2022.

3.2.4 Функціональне призначення програми. «Консольна Програма для створення та візуалізації роботи протоколу STARK» призначена для створення та візуалізації роботи протоколу STARK за умови, що сутність

Prover завжди працює коректно.

Ключові функції програми:

1) Введення числа, що позначає довжину адитивної групи і має сенс модуля, за яким проходять обчислення в програмі.

2) Введення числа, що позначає першообразний корінь групи.

3) Введення числа, що позначає довжину підгрупи.

4) Відображення полінома ступеня менше двох.

5) Вихід із програми.

Відомості про функціональні обмеження програми на застосуванні:

- Обсяг пам'яті, який займає програма під час функціонування, залежить від довжини підгрупи. Що більшою є довжина підгрупи, то більшою є квадратна матриця для пошуку коренів інтерполяційного полінома (залежність обсягу пам'яті, який займає програма, від довжини підгрупи має нелінійний характер). Програма КСІВРП-STARK-02 потребує для функціонування щонайменше 10 Мега байтів вільної оперативної пам'яті, за меншої кількості вільної пам'яті програма може не функціонувати.

- Програма КПСІВРП-STARK-02 вимагає введення тільки натуральних цілих чисел, можливе введення також нуля. Діапазон обмежується можливостями бібліотеки NTL.

3.2.5 Опис логічної структури програми. Програма написана і виконується з текстом програми мовою C^{++}.

Алгоритм програми та використовувані методи. Програма обробляє одержувані дані через меню керування командами в консолі користувача. Параметр, що відповідає за довжину підгрупи, вводиться окремо прямо в тіло програми.

Методи, що використовуються, ґрунтуються на можливостях апаратних модулів технічного засобу, на якому запущено Програму.

Алгоритм і методи:

1) Запуск програми

2) Створення компонентів

3) Ініціалізація компонентів користувачем

4) Обробка введених даних

5) Візуалізація даних.

3.2.6 Структура програми з описом функцій складових частин і зв'язку між ними. Структурний опис програми можливо подати у вигляді рисунка 3.2.

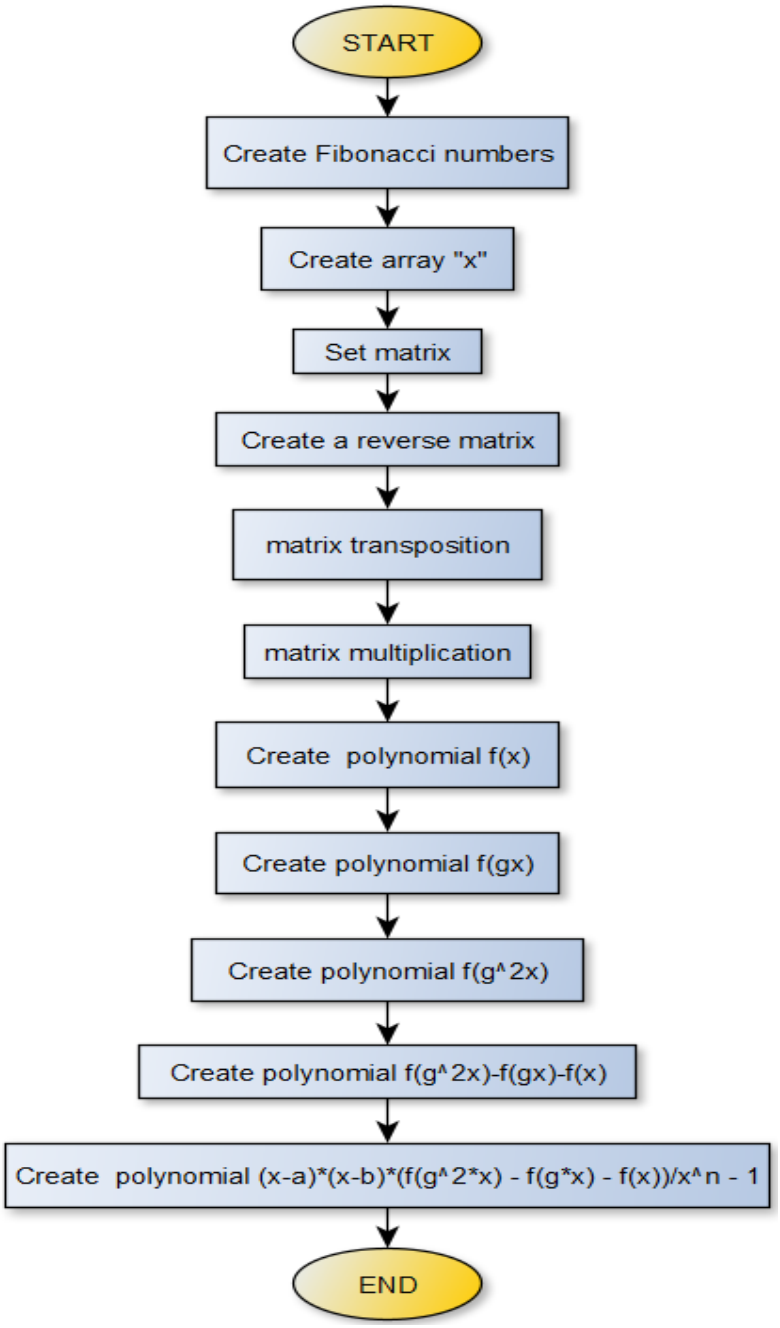

Рисунок 3.2 - Структура програми КПСІВРП-STARK-02

3.2.7 Зв'язки програми з іншими програмами. Програма не пов'язана з іншими програмами. Програми працює в режимі Offline.

3.2.8 Технічні засоби, що використовуються. Програма встановлюється на обладнанні з такими мінімальними характеристиками:

- CPU: Intel(R) Core(TM) i7-9700F CPU @ 3.00GHz 3.00 GHz;
- RAM: - 64 Gb;
- об'єм диска - 270 Gb (залежить від планованої кількості АТТ);
- операційна система Windows 10.

3.2.9 Виклик і завантаження. Після встановлення на комп'ютер, запуск програми КПСІВРП-STARK-01 проводиться вручну після завантаження операційної системи. Спочатку з архіву відкривається проект, потім відкривається файл, прикріплений до проекту.

3.2.10 Опис вхідних і вихідних даних. *Вхідні дані.* Вхідними даними для Програми є:

1) Файл конфігурації Програми;

2) Команди, що вводяться користувачем за допомогою інтерфейсу командного рядка.

Організація файлу конфігурації являє собою набір текстових рядків англійською мовою в кодуванні ASCII, що відповідають командам інтерфейсу командного рядка Програми. Текстові рядки розділяються символами кінця рядка. У файлі конфігурації містяться налаштування програми за замовчуванням. Можливі зміни файлу конфігурації. Не рекомендується проводити зміни файлу конфігурації.

Команди інтерфейсу командного рядка представлені текстовими рядками англійською мовою в кодуванні ASCII. Елементи команди (слова) відповідають за змістом діям, що виконуються командою.

Список команд, підтримуваних Програмою, можна представити структурою з одним початковим вузлом. Попередньої підготовки команд інтерфейсу командного рядка не потрібно.

Вихідні дані. Вихідними даними для Програми є:

1) Файл конфігурації програми

2) Повідомлення, що передаються оператору за допомогою інтерфейсу командного рядка.

Характер, організація, формат, опис і спосіб кодування файлу конфігурації відповідає описаним для вхідних даних у частині файлу конфігурації.

Повідомлення, що передаються оператору за допомогою інтерфейсу командного рядка, являють собою текстові рядки і символи англійською мовою в кодуванні ASCII.

Повний код програми наведено в додатку (Додаток В, Додаток Г).

4 ОЦІНКА ЕФЕКТИВНОСТІ РОБОТИ ПРОГРАМИ ПЕРШОГО ЕТАПУ РОБОТИ ПРОТОКОЛУ STARK НА ДОМАШНЬОМУ КОМП'ЮТЕРІ

На нашу думку доцільним буде провести порівняльний аналіз ефективності роботи першого етапу роботи протоколу STARK за допомогою програм КПСІВРП-STARK-01 КПСІВРП-STARK-02 на «домашньому комп'ютері».

4.1 Порівняльна оцінка часу роботи програм КПСІВРП-STARK-01 КПСІВРП-STARK-02 в залежності від вхідних даних

Оцінимо залежність часу роботи програм КПСІВРП-STARK-01 й КПСІВРП-STARK-02 від розміру матриці пошуку коренів інтерполяційного полінома. У таблиці 4.1 наведено дані, які показують залежність часу роботи програми КПСІВРП-STARK-01 від розміру матриці пошуку коренів інтерполяційного полінома.

Таблиця 4.1 - Залежність часу роботи програми КПСІВРП-STARK-01 від розміру матриці пошуку коренів інтерполяційного полінома

Розмір матриці, nxn	Час роботи програми, секунда
12x12	0,007
112x112	0,103
506x506	6,68
1012x1012	52,352
2112x2112	469,331
3330x3330	1846,28

У таблиці наведені розміри матриці, що являють собою розміри підгрупи, які створені першообразним елементом групи. У таблиці n дорівнює довжині підгрупи. Ці розміри матриці пошуку коренів інтерполяційного поліному використовуються в таблицях та графіках

(Додаток Д).

У таблиці 4.2 наведено дані, що показують залежність часу роботи програми КПСІВРП-STARK-02 від розміру матриці пошуку коренів інтерполяційного полінома.

Таблиця 4.2 - Залежність часу роботи програми КПСІВРП-STARK-02 від розміру матриці пошуку коренів інтерполяційного полінома

Розмір матриці, nxn	Час роботи програми, секунда
12x12	0,019
112x112	0,401
506x506	27,597
1012x1012	236,69
2112x2112	2187,62

Порівняємо залежності часу роботи програми КПСІВРП-STARK-01 та КПСІВРП-STARK-02 від розміру матриці пошуку коренів інтерполяційного полінома. Отримана залежність подана на рисунку 4.1.

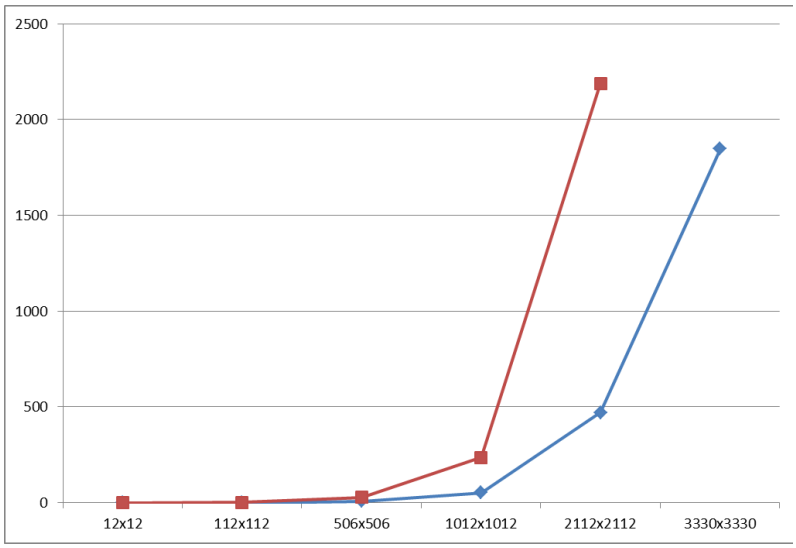

Рисунок 4.1 - Залежність часу роботи програми КПСІВРП-STARK-01 та КПСІВРП-STARK-02 від розміру матриці пошуку коренів інтерполяційного полінома

За графіком можна зробити висновок, що:

1) Програма КПСІВРП-STARK-01, яка використовує метод Гауса для пошуку коренів інтерполяційного полінома, працює швидше, ніж програма КПСИВРП-STARK-02, яка використовує метод обернених матриць для пошуку коренів інтерполяційного полінома.

2) За допомогою методу обробки результатів експерименту методом найменших квадратів системи комп'ютерної алгебри Mathcad була зроблена апроксимація таблично заданої функції за вхідними даними таблиць 4.1 та 4.2. Результати розрахунків подані у Додатку Е та Додатку Ж. За даними розрахунків в Mathcad можна зробити висновок, що: залежність часу роботи програми КПСІВРП-STARK-01 та КПСІВРП-STARK-02 від розміру матриці пошуку коренів інтерполяційного полінома має поліноміальний характер, який можна виразити якісною функціональною залежністю $f(x) = x^3$. Це зумовлено тим, що часова складність алгоритму обох програм, у нотації Великого О, дорівнює $O(Marray^3)$. Де Marray – довжина підгрупи, яка утворена адитивною групою, наприклад, розміром 13 чисел (як в задачі Розділу 2). У нашому випадку фактична часова складність кожної програми може відрізнятися від даних апроксимації через особливості обладнання, на якому відбувається оброблення інформації.

4.2 Порівняльна оцінка залежності об'єму пам'яті, яку потребує програми КПСІВРП-STARK-01 й КПСІВРП-STARK-02 під час своєї роботи

Оцінимо залежність об'єму пам'яті програм, який вони потребують під час своєї роботи, від розміру матриці пошуку коренів інтерполяційного полінома. У таблиці 4.3 подані дані, які показують залежність об'єму пам'яті програми КПСІВРП-STARK-01 від розміру матриці пошуку коренів інтерполяційного полінома.

Таблиця 4.3 - Залежність об’єму пам’яті програми КПСІВРП-STARK-01 від розміру матриці пошуку коренів інтерполяційного полінома

Розмір матриці, nxn	Пам'ять процесу, МБ
12x12	0,94
112x112	0,99
506x506	13
1012x1012	47
2112x2112	200
3330x3330	495

У таблиці 4.4 подані дані, які показують залежність об’єму пам’яті програми КПСІВРП-STARK-02 від розміру матриці пошуку коренів інтерполяційного полінома.

Таблиця 4.4 - Залежність об’єму пам’яті програми КПСІВРП-STARK-02 від розміру матриці пошуку коренів інтерполяційного полінома

Розмір матриці, nxn	Пам'ять процесу, МБ
12x12	0,94
112x112	2
506x506	24
1012x1012	93
2112x2112	300
3330x3330	600

Порівняємо залежності об’ємів пам’яті, яку потребує програми КПСІВРП-STARK-01 та КПСІВРП-STARK-02 від розміру матриці пошуку коренів інтерполяційного полінома. Якщо за даними двох таблиць створити графіки (рис. 4.2), то є можливим отримати такі залежності:

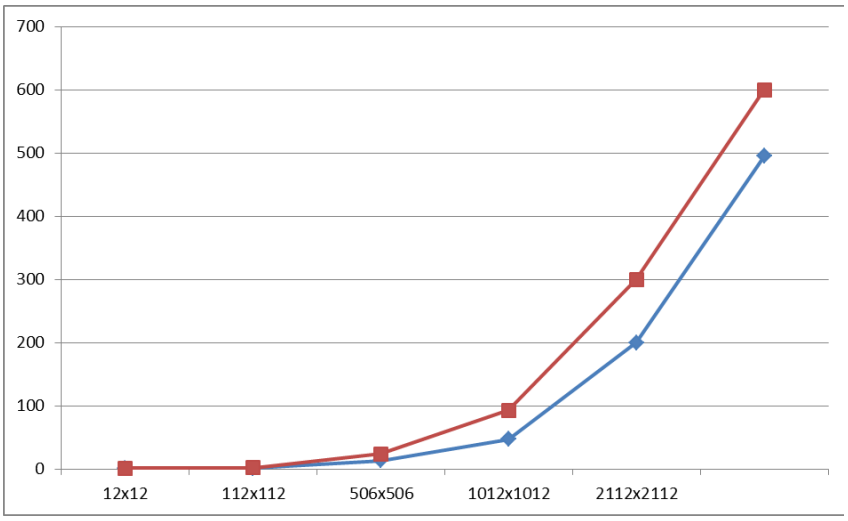

Рисунок 4.2 - Залежність об’єму пам’яті, який потребує програми КПСІВРП-STARK-01 та КПСІВРП-STARK-02 від розміру матриці пошуку коренів інтерполяційного полінома

За графіком можна зробити висновок, що:

1) Програма КПСИВРП-STARK-01, яка використовує метод Гауса для пошуку коренів інтерполяційного полінома, потребує меншого об’єму пам’яті, ніж програма КПСИВРП-STARK-02, яка використовує метод обернених матриць для пошуку коренів інтерполяційного полінома.

2) За допомогою методу обробки результатів експерименту методом найменших квадратів системи комп’ютерної алгебри Mathcad була зроблена апроксимація за вхідними даними таблиці 4.3 та 4.4. Результати розрахунків подані у Додатку И та Додатку К. За даними розрахунків в Mathcad можна зробити висновок, що: залежність об’єму пам’яті, які потребують дві програми при роботі, від розміру матриці пошуку коренів інтерполяційного полінома має нелінійний характер та просторова складність двох програм є, як мінімум, O(Marray^2) у нотації Великого О.

4.3 Порівняльна оцінка залежності об’єму пам’яті, яку потребує матриця пошуку коренів інтерполяційного полінома від її розміру на «домашньому комп’ютері»

«Домашній комп’ютер», що було застосовано для тестування Програм

відповідає наступним характеристикам:

Ім’я вузла: WIN-BGK8K18O2H4

Назва ОС: Майкрософт Windows 10 Pro

Версія ОС: 10.0.19044 Н/Д побудова 19044

Виробник ОС: Microsoft Corporation

Параметри ОС: Ізольована робоча станція

Виробник системи: Gigabyte Technology Co., Ltd.

Модель системи: Z390 D

Тип системи: x64-based PC

Процесор(и): Число процесорів - 1.

[01]: Intel64 Family 6 Model 158 Stepping 13 GenuineIntel ~3000 МГц

Версія BIOS: American Megatrends Inc. F3c, 18.12.2019

Повний обсяг фізичної пам’яті: 65 469 МБ

Доступна фізична пам’ять: 58 882 МБ

Віртуальна пам’ять: Макс. розмір: 75 197 МБ

Віртуальна пам’ять: Доступна: 66 502 МБ

Віртуальна пам’ять: Використовується: 8 695 МБ**,**

Покажемо залежність об’єму пам’яті, яку потребує матриця пошуку коренів інтерполяційного полінома від її розміру, яку може показати «домашній комп’ютер» засобами Microsoft Notepad (таб. 4.5).

Таблиця 4.5 - Залежність об’єму пам’яті, яку потребує матриця пошуку коренів інтерполяційного полінома від її розміру

Розмір матриці, nxn	Пам'ять, яку займає сама матриця, КБ
1	2
12x12	1
112x112	38
506x506	976
1012x1012	3902

Продовження таблиці 4.5

1	2
2112x2112	19417
3330x3330	50477
5004x5004	128518
7506x7506	289540
10056x10056	537690
15012x15012	1157806
20787x20787	2483554
24192x24192	3356745

Цю залежність можна ілюструвати графіком (рис. 4:3).

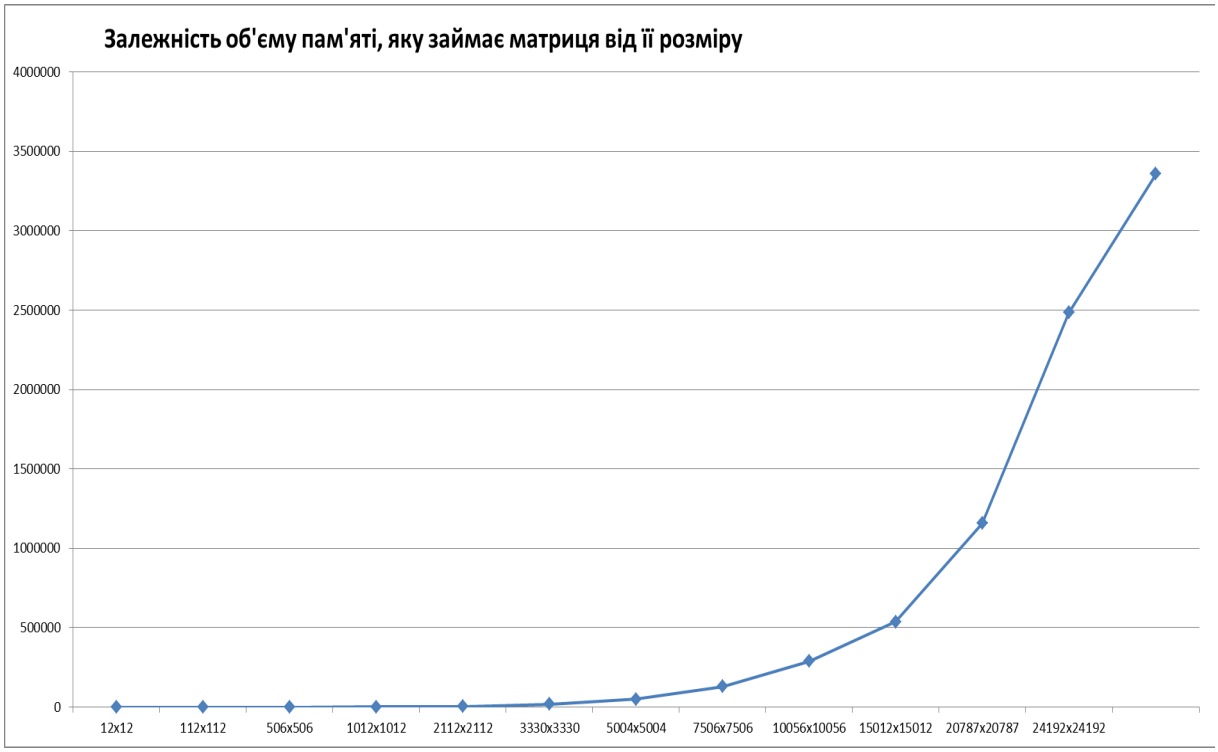

Рисунок 4.3 - Залежність об’єму пам’яті, яку потребує матриця пошуку коренів інтерполяційного полінома від її розміру

За допомогою комп’ютерної алгебри Mathcad була зроблена апроксимація за вхідними даними таблиці 4.5. Результати розрахунків подані у Додатку Л. За даними розрахунків в Mathcad можна зробити висновок, що: ця залежність має нелінійний характер.

Також, максимальний розмір матриці, яку було створено та переміщено в файл формату .txt і яку є можливим переглянути засобами Microsoft Notepad становить 10056x10056 невідомих та рівнянь.

Враховуючи результати проведеного тестування можливо зробити висновки щодо ефективності побудованих програм з урахуванням можливостей комп’ютеру, на якому було створено програми та міжнародного досвіду вирішення системи лінійних алгебраїчних рівнянь (СЛАР) з великою кількістю невідомих та рівнянь.

1) Залежність часу роботи програми КПСИВРП-STARK-01 та КПСИВРП-STARK-02 від розміру матриці пошуку коренів інтерполяційного полінома має поліноміальний характер, який можна виразити якісною функціональною залежністю $f(x) = x^3$. Це зумовлено тим, що часова складність алгоритму обох програм, у нотації Великого О, дорівнює $O(Marray^3)$.

2) Залежність об’єму пам’яті, які потребують дві програми при роботі, від розміру матриці пошуку коренів інтерполяційного полінома має нелінійний характер та просторова складність двох програм є, як мінімум, $O(Marray^2)$ у нотації Великого О.

3) Залежність об’єму пам’яті, яку потребує матриця пошуку коренів інтерполяційного полінома від її розміру насить нелінійний характер.

4) Більшу частину часу роботи протоколу займає саме розрахунок коренів інтерполяційного поліному.

Підтвердженням цього є дані, що наведені у таблицях 4.6 та 4.7.

Таблиця 4.6 – Зведені дані щодо ефективності програми КПСИВРП-STARK-01

Розмір матриці, nxn	Час пошуку коренів інтерполяційного поліному, секунда	Загальний час роботи програми, секунда	Час пошуку коренів інтерполяційного поліному, %
12x12	0,004	0,015	26,66666667
112x112	0,292	0,309	94,49838188
506x506	28,193	28,69	98,26768909
1012x1012	232,564	233,149	99,74908749

Дані таблиці показують, що більшу частину часу роботи протоколу займає саме розрахунок коренів інтерполяційного поліному.

Таблиця 4.7 – Зведені дані щодо ефективності програми КПСИВРП-STARK-02

Розмір матриці, nxn	Час пошуку коренів інтерполяційного поліному, секунда	Загальний час роботи програми, секунда	Час пошуку коренів інтерполяційного поліному, %
12x12	0,00000015	0,005	0,003
112x112	0,08	0,093	86,02150538
506x506	6,723	6,87	97,86026201
1012x1012	51,618	52,203	98,87937475

Дані таблиці показують, що більшу частину часу першого етапу роботи протоколу займає саме розрахунок коренів інтерполяційного поліному.

Дані таблиць 4.6 та 4.7 також означають, що відсоток часу, який займає розрахунок коренів інтерполяційного поліному у протоколі STARK від загального часу роботи першого етапу протоколу, не залежить від методу

розрахунку коренів інтерполяційного поліному, а залежить від розміру самої матриці.

Той факт, що відсоток часу, який займає розрахунок коренів інтерполяційного поліному у протоколі STARK від загального часу роботи першого етапу протоколу, не залежить від методу розрахунку коренів інтерполяційного поліному, може свідчити про те, що існує якась фізична межа для здійснення розрахунків на першому етапі роботи протоколу STARK, яка зв’язана із фізичними можливостями, що зв’язані із швидкодією існуючих комп'ютерів.

З цього випливає, що вирішення СЛАР з великою кількістю невідомих та рівнянь – це дуже складна обчислювальна задача. Тому, якщо діяльність протоколу STARK зумовлює рішення СЛАР, такого розміру, які наведені у таблицях чи більшого розміру, то має сенс розподілити обчислення між комп’ютерами обчислювальної мережі задля прискорення роботи протоколу або тільки збільшити потужність окремих вузлів мережі. У майбутньому, коли квантові комп’ютери будуть масово використовуватися, складність розрахунків у протоколі STARK буде менше впливати на час його роботи, тому обчислення будуть проходити швидше і менш затратно з точки зору електроспоживання. А в купі із стійкістю до атак квантових комп’ютерів, протокол STARK буде дуже корисний задля захисту інформації в комп’ютерних мережах, наприклад, у криптовалютах. Це свідчить про те, що протокол STARK має великий потенціал та зможе його розкрити зі збільшення обчислювальної потужності комп’ютерів.

ВИСНОВКИ

У роботі проведено моделювання та тестування низького ступеня у криптографічних протоколах доказу із нульовим розголошенням шляхом програмної реалізації першого етапу «Арифметизація» роботи протоколу ZK-STARK.

1) Визначено основні спільні риси функціонування схожих криптографічних протоколів STARK і SNARK: обидва протоколи STARK і SNARK мають спільний для них етап роботи «Арифметизація», що створює докази і перевіряє їх на хибність або достовірність для того, щоб не приєднати або приєднати транзакцію до Блокчейну відповідно.

2) Визначено основні відмінності схожих криптографічних протоколів STARK і SNARK: протокол STARK «маштабований», а протокол SNARK «лаконічний» і вимагає процедуру «препроцесингу», протокол STARK «прозорий», а протокол SNARK «неінтерактивний» і вимагає процедуру зі створення «довіреного набору».

3) Створено програмну реалізацію першого етапу роботи протоколу STARK «Арифметизація» шляхом написання двох програм КПСІВРП-STARK-01 та КПСІВРП-STARK-02 на «домашньому комп’ютері». Програма КПСІВРП-STARK-01 використовує метод Гауса для пошуку коренів інтерполяційного полінома. Програма КПСИВРП-STARK-02, яка використовує метод обернених матриць для пошуку коренів інтерполяційного полінома.

4) Тестування першого етапу «Арифметизація» протоколу ZK-STARK для оцінки тривалості роботи програми від розміру матриці пошуку коренів інтерполяційного поліному виявило, що Програма КПСІВРП-STARK-01 потребує меншого об’єму пам’яті, ніж програма КПСІВРП-STARK-02. Апроксимація за вхідними даними обох програм в Mathcad виявила що:

залежність об’єму пам’яті, яку потребують обидві програми при роботі, від розміру матриці пошуку коренів інтерполяційного полінома має нелінійний характер.

5) Ефективність створених Програм з урахуванням можливостей «домашнього комп’ютеру» та міжнародного досвіду вирішення СЛАР з великою кількістю невідомих та рівнянь, показало, що залежність часу роботи Програми КПСІВРП-STARK-01 та КПСІВРП-STARK-02 від розміру матриці пошуку коренів інтерполяційного полінома має поліноміальний характер, який можна виразити якісною залежністю $f(x) = x^3$. Залежність об’єму пам’яті, які потребують дві програми при роботі, від розміру матриці пошуку коренів інтерполяційного полінома має також нелінійний характер. Залежність об’єму пам’яті, яку потребує матриця пошуку коренів інтерполяційного полінома від її розміру насить також нелінійний характер. Більшу частину часу роботи першого етапу «Арифметизація» протоколу ZK-STARK займає саме розрахунок коренів інтерполяційного поліному.

6) Вирішення СЛАР з великою кількістю невідомих та рівнянь – це дуже складна обчислювальна задача. Тому, якщо діяльність протоколу STARK зумовлює рішення СЛАР, такого розміру, які наведені у таблицях чи більшого розміру, то, наразі, має сенс розподілити обчислення між комп’ютерами обчислювальної мережі задля прискорення роботи першого етапу роботи протоколу або збільшити потужність окремих вузлів мережі.

7) Для більш швидкої реалізації протоколу слід використовувати метод Гауса для пошуку коренів інтерполяційного полінома, ніж метод обернених матриць.

8) Для оброблення дуже великі обсягів даних, протокол STARK має бути реалізований на обладнанні, яке значно потужніше, ніж те, на якому було проведене тестування протоколу, наприклад, на квантовому комп’ютері чи мережі звичайних комп’ютерів, бо залежність часу роботи протоколу та обсягу пам’яті, яку він займає при роботі носить

поліноміальний характер, який можна виразити якісною залежністю $f(x) = x^3$.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кравченко П., Скрбін Б., Дубініна О. Блокчейн і децентралізовані системи: навчальний посібник [для студентів вищих навчальних закладів, які навчаються за напрямами підготовки «Безпека інформаційних і комунікаційних систем»]. Х. : ХНУ імені В. Н. Каразіна, 2013. – 632 с.
2. Arithmetization I / StarkWare // Medium. URL: https://medium.com/starkware/arithmetization-i-15c046390862 (дата звернення: 30.04.2023).
3. Lorne Lantz and Daniel Cawrey. Mastering Blockchain // Errata. URL: http://oreilly.com/catalog/errata.csp?isbn=9781492054702 (дата звернення: 25.09.2022).
4. Introduction to zk-SNARKs / Blog // CONSENSYS. URL: https://consensys.net/blog/developers/introduction-to-zk-snarks/ (дата звернення: 30.04.2023).
5. What are zk-STARKs? // bit2me ACADEMY. URL: https://academy.bit2me.com/en/what-are-zk-stark/ (дата звернення: 30.04.2023).
6. Zero-Knowledge Proofs: STARKs vs SNARKs / BLOCKCHAIN EXPLAINED // CONSENSYS. URL: https://consensys.net/blog/blockchain-explained/zero-knowledge-proofs-starks-vs-snarks/ (дата звернення: 30.04.2023).
7. STARK Math – A Very Short Primer // STARKWARE. URL: https://starkware.co/stark-math-a-very-short-primer/ (дата звернення: 30.04.2023).
8. Кузнецов О., Полуяненко М. Основні поняття та визначення в моделюванні консенсусу децентралізованих систем. Протоколи

консенсусу відповідно до моделі «Доказ виконаної роботи»: методичні рекомендації з дисципліни «Технології блокчейн»[для студентів вищих навчальних закладів, які навчаються за напрямами підготовки «Безпека інформаційних і комунікаційних систем»]. Х. : ХНУ імені В.Н. Каразіна, 2019. – 45 с.

9. Consensus Algorithms: The Root Of Blockchain Technology / Guides // 101Blockchains. URL: https://101blockchains.com/consensus-algorithms-blockchain/ (дата звернення: 30.04.2023).
10. Comparison Among The Consensus Algorithms / Rutu Manhandi // Linkedin. URL: https://www.linkedin.com/pulse/comparison-among-consensus-algorithms-rutu-mandhani (дата звернення: 30.04.2023).
11. zk-SNARKs and zk-STARKs Explained // BINANCE-Academy. URL: https://academy.binance.com/en/articles/zk-snarks-and-zk-starks-explained (дата звернення: 30.04.2023).
12. SNARKs vs. STARKs / ZK Whiteboard Sessions // ZK Hack. URL: https://zkhack.dev/whiteboard/module-four/ (дата звернення: 30.04.2023).
13. zk-SNARKs: A Gentle Introduction // ENS. URL: https://www.di.ens.fr/~nitulesc/files/Survey-SNARKs.pdf (дата звернення: 30.04.2023).
14. zk-STARKs vs zk-SNARKs: Differences in zero-knowledge technologies // Panther Academy. URL: https://blog.pantherprotocol.io/zk-snarks-vs-zk-starks-differences-in-zero-knowledge-technologies/ (дата звернення: 30.04.2023).
15. The Zero Knowledge Frontier: On SNARKs, STARKs, and Future Applications // TheTie. URL: https://research.thetie.io/zero-knowledge-starks-snarks/ (дата звернення: 30.04.2023).
16. zk-STARKs vs. zk-SNARKs explained // COINTELEGRAPH The future of themoney. URL: https://cointelegraph.com/explained/zk-starks-vs-zk-snarks-explained (дата звернення: 30.04.2023).

17. Understanding the Difference Between zk-SNARKs and zk-STARKS / EDUCATION // Chainlink|Blog. URL: https://blog.chain.link/zk-snarks-vs-zk-starks/ (дата звернення: 30.04.2023).
18. Zero-Knowledge Proofs: STARKs vs SNARKs / BLOCKCHAIN EXPLAINED // CONSENSYS. URL: https://consensys.net/blog/blockchain-explained/zero-knowledge-proofs-starks-vs-snarks/ (дата звернення: 30.04.2023).
19. Scalable, transparent, and post-quantum secure computational integrity // iacr. URL: https://eprint.iacr.org/2018/046.pdf (дата звернення: 30.04.2023).
20. Bikramaditya Singhal, Gautam Dhameja, Priyansu Sekhar Panda. Beginning Blockchain. A Beginner's Guide to Building Blockchain Solutions // SpringerLink. URL: https://doi.org/10.1007/978-1-4842-3444-0 (дата звернення: 25.09.2022).
21. Theoretical and practical introduction to ZK-SNARKs and ZK-STARKs // muni. URL: https://is.muni.cz/th/ovl3c/SNARKs_STARKs_introduction_Archive.pdf (дата звернення: 30.04.2023).
22. STARK Math: The Journey Begins / StarkWare // Medium. URL: https://medium.com/starkware/stark-math-the-journey-begins-51bd2b063c71 (дата звернення: 30.04.2023).
23. Firsov Denis. Zero-Knowledge in EasyCrypt // Guardtime Tallinn University of Technology Tallinn, Estonia / Dominique Unruh Tartu University Tartu, Estonia. URL: https://m.guardtime.com/files/Zero-Knowledge%20in%20EasyCrypt.pdf (дата звернення: 30.04.2023).
24. StarkWare Team*. ethSTARK Documentation Version 1.1 // iacr. URL: https://eprint.iacr.org/2021/582.pdf (дата звернення: 30.04.2023).
25. Pethuru Raj, Kavita Saini, Chellammal Surianarayanan. Blockchain Technology and Applications // amazon. URL: https://www.amazon.com/Blockchain-Technology-Applications-Pethuru-

Raj-ebook-dp-B08DWBSPGW/dp/B08DWBSPGW/ref=mt_other?_encoding=UTF8&me=&qid (дата звернення: 02.05.2023).

26. Introducing Plonky2 / Polygon Labs // Polygon. URL: https://polygon.technology/blog/introducing-plonky2 (дата звернення: 02.05.2023).

ДОДАТОК А

Лістинг програми КСІВРП-STARK-01

```
#include <iostream>
#include <fstream>
#include <ctime>
#include <cmath>
#include <vector>
#include <iostream>
#include <vector>
#include <chrono>

#include "NTL\ZZ.h"

using namespace std;
using namespace NTL;
using namespace std;
using namespace std;

template <typename T> void FreeMem(T** matr, int n);
void gaussian_elimination(int n);
vector<ZZ> back_substitution(int n);
/*

СЛАУ-Г-04.txt

*/

//const int Marray = 24192;
//ZZ g = conv < ZZ>("2");
//ZZ Mod = conv<ZZ>("96769");
```

```
ZZ Mod = conv<ZZ>("13");


ZZ g = conv < ZZ>("4");


const int Marray = 6;


ZZ* argX = new ZZ[Marray];


ZZ number00 = conv < ZZ>("0");
ZZ number01 = conv < ZZ>("1");
ZZ number02 = conv < ZZ>("2");
vector<vector<ZZ>> a;
vector<ZZ> b;
void main()
{
    cout << "Enter parametr Mod: " << endl;
    cin >> Mod;
    cout << "Enter parametr g: " << endl;
    cin >> g;
    double time_spent = 0.0;
    clock_t begin = clock();


    setlocale(0, "");
    int n = 1;


    const int size01 = 1;
    cout << "Enter the size of the matrix: ";
    n = Marray;
    cout << n << endl;
    ZZ* Fibonacci = new ZZ[n];
    //=====================================================================
        // Формирование чисел Фибоначчи
    Fibonacci[0] = conv < ZZ>("1");
    Fibonacci[1] = conv < ZZ>("1");
```

```
for (int j = 2; j < Marray; j++)
{
   Fibonacci[j] = Fibonacci[j - 1] + Fibonacci[j - 2];
   // cout << Fibonacci[j] << endl;
}
//=============================================================================

cout << "Fibonacci numbers - done" << endl;
//=============================================================================
   // Поиск значий циклической группы по заданному первообразному корню g
for (int j = 0; j < Marray; j++)
{
   argX[j] = PowerMod(g, j, Mod);//PowerMod(g, j, Mod)

}
ZZ tmp01[3]{};
tmp01[0] = 1;//при x^2
tmp01[1] = (-(argX[Marray - 1] + argX[Marray - 2]));//при x^1
tmp01[2] = ((argX[Marray - 1] * argX[Marray - 2]));//при x^0
cout << "Args X - done" << endl;
//=============================================================================




a.assign(Marray, vector<ZZ>(Marray));
b.assign(Marray, number00);


for (int i = 0; i < a.size(); i++) {
   for (int j = 0; j < a.size(); j++)
      a[i][j] = PowerMod(argX[i], j, Mod);
}

for (int i = 0; i < Marray; i++)
```

```
    {
        b[i] = Fibonacci[i];
    }

    delete[] argX;

    cout << "The matrix was created " << endl;

//*****************************************************************************************************************
************

    //**************************************************Решение СЛАУ методом
Гаусса**************************************************

    auto start = chrono::high_resolution_clock::now();

    gaussian_elimination(Marray);
    auto x = back_substitution(Marray);

    auto end2 = chrono::high_resolution_clock::now();
    double time_taken =
        chrono::duration_cast<chrono::milliseconds>(end2 - start).count();

    cout << "Solution:\n";
    /* for (int i = 0; i < Marray; ++i)
         cout << "x[" << i << "] = " << x[i] << "\n";*/

    cout << "Time taken: " << time_taken / 1000 << " seconds\n";

//*****************************************************************************************************************
************

//*****************************************************************************************************************
************
    for (int i = 0; i < n / 2; i++) {
        ZZ temp = x[n - i - 1];
        x[n - i - 1] = x[i];
        x[i] = temp;
    }
```

```
cout << "Полином f(x): " << endl;
for (int i = 0; i < Marray; i++)
{

    cout << x[i] << " ";
}

ZZ* f_gx_ = new ZZ[Marray];
ZZ* f_g2x_ = new ZZ[Marray];
ZZ* subtractionArr = new ZZ[Marray];

cout << endl;
cout << "Polynom f(g*x):" << endl;
int counter02 = Marray - 1;
for (int i1 = 0;i1 < Marray;i1++) {
    f_gx_[i1] = x[i1] * PowerMod(g, counter02, Mod);
    f_gx_[i1] = f_gx_[i1] % Mod;
    /*cout << "f_gx_ [" << i1 << "] = " << f_gx_[i1] << endl;*/
    cout << f_gx_[i1] << " ";
    counter02--;
}
cout << endl;



cout << "Polynom f(g^2*x):" << endl;
ZZ number03 = PowerMod(g, number02, Mod);
int counter03 = Marray - 1;
for (int i = 0;i < Marray;i++) {
    f_g2x_[i] = x[i] * PowerMod(number03, counter03, Mod);
    f_g2x_[i] = f_g2x_[i] % Mod;
    /*cout << "f_g2x_ [" << i << "] = " << f_g2x_[i] << endl;*/
    cout << f_g2x_[i] << " ";
    counter03--;
}
cout << endl;
```

```
cout << "Polynom f(g^2*x) - f(g*x) - f(x):" << endl;
for (int i = 0; i < Marray; i++) {
    subtractionArr[i] = f_g2x_[i] - f_gx_[i] - x[i];
    subtractionArr[i] = subtractionArr[i] % Mod;
   /*cout << "subtractionArr [" << i << "] = " << subtractionArr[i] << endl;*/
    cout << subtractionArr[i] << " ";
}
cout << endl;
delete[]f_gx_;
delete[]f_g2x_;
//delete[]x;

cout << "Polynom (x-a)*(x-b)*(f(g^2*x) - f(g*x) - f(x))" << endl;
const int NewSize = Marray + 2;
ZZ* mul = new ZZ[NewSize];



for (int i = Marray - 1; i >= 0; i--) {
    for (int j = 2; j >= 0; j--) {
        mul[i + j] += subtractionArr[i] * tmp01[j];//tmp02+
        mul[i + j] = mul[i + j] % Mod;

    }
}
delete[]subtractionArr;
int i10 = 0;
for (i10 = 0; i10 < NewSize; i10++) {
   /* cout << "mul [" << i10 << "]  =" << mul[i10] << endl;*/
    cout << mul[i10] << " ";
}
cout << endl;
cout << "Polynom (x-a)*(x-b)*(f(g^2*x) - f(g*x) - f(x))/x^n - 1" << endl;
//=============================================================================
//=============================================================================
//=============================================================================
```

```
//
   //Polynom temp = p1;
const int degreeOfNumerator = NewSize;//degree = 7
const int degreeOfDivider = Marray;//degree = 6
int rdeg = degreeOfNumerator - degreeOfDivider;

ZZ* res = new ZZ[NewSize - Marray + 1]{};
ZZ* PolynomialDivider = new ZZ[degreeOfNumerator]{};

for (int i10 = 0; i10 < degreeOfNumerator; i10++) {
   PolynomialDivider[i10] = 0;
}

PolynomialDivider[1] = 1;
PolynomialDivider[degreeOfNumerator - 1] = 12;

//for (i10 = 0; i10 < degreeOfNumerator; i10++) {
//   /* cout << "PolynomialDivider [" << i10 << "]  =" << PolynomialDivider[i10] << endl;*/
//    cout << PolynomialDivider[i10] << " ";
//}

//**************************************************Рабочее**************************************************
//cout << "degreeOfNumerator = " << degreeOfNumerator << endl;
//cout << "rdeg = " << rdeg << endl;
int counter04 = 1;

for (int i11 = 0; i11 < rdeg; i11++) {
   res[rdeg - i11 - 1] = mul[0 + i11] / PolynomialDivider[1];
   //cout << "res [" << i11 << "]  =" << (res[rdeg - i11 - 1]) << endl;
   cout << (res[rdeg - i11 - 1]) << " ";

   for (int j4 = 0; j4 <= degreeOfNumerator; j4++) {
      mul[0 + j4 + i11] -= PolynomialDivider[counter04] * res[rdeg - i11 - 1];
      //cout << "mul [" << 0 + j4 + i11 << "]  =" << mul[0 + j4 + i11] % Mod << endl;
      counter04++;
```

```
            if (counter04 == NewSize) {
                break;
            }

        }

        counter04 = 1;

    }
        delete[]res;
        delete[]mul;
        delete[]PolynomialDivider;
        //*****************************************************************************************************************
    //*************<Time>*************************************************************
    //*****************************************************************************************************************
        clock_t end = clock();
        // рассчитать прошедшее время, найдя разницу (end - begin) и
            // деление разницы на CLOCKS_PER_SEC для перевода в секунды
        time_spent += (double)(end - begin) / CLOCKS_PER_SEC;
        cout << endl;
        cout << "The elapsed time is %f" << time_spent << " seconds" << endl;

} // end of main



//*****************************************************************************************************
//                      Дополнительные функции
//*****************************************************************************************************
//Функция освобождения памяти
template <typename T> void FreeMem(T** matr, int n)
{
    for (int i = 0; i < n; i++)
        delete[] matr[i];
    delete[] matr;
}
```

```
// Gaussian elimination to transform matrix to triangular form
void gaussian_elimination(int n) {
    for (int row = 0; row < n; ++row) {
        // Find row with maximum value in the column
        int max_row = row;
        for (int i = row + 1; i < n; ++i) {
            if (abs(a[i][row]) > abs(a[max_row][row]))
                max_row = i;
        }
        // Swap rows
        swap(a[row], a[max_row]);
        swap(b[row], b[max_row]);
        // Eliminate column
        //if (abs(a[row][row]) < EPS)
        //    continue;
        ZZ invElofGroup = conv<ZZ>("1");
        ZZ temp = a[row][row];
        for (int ii = 1; ii < Mod; ii++)
        {
            ZZ antiDivider = temp * ii;
            ZZ antiDivider1 = antiDivider % Mod;
            if (antiDivider1 == 1)
            {
                invElofGroup = ii;
            }
        }
        ZZ inv_diag = (number01 * invElofGroup) % Mod;
        for (int i = row + 1; i < n; ++i) {
            ZZ k = (a[i][row] * inv_diag) % Mod;
            for (int j = row + 1; j < n; ++j)
                a[i][j] -= (k * a[row][j]) % Mod;
            b[i] -= (k * b[row]) % Mod;
            a[i][row] = number00;
        }
```

```
    }
}

// Back substitution to solve triangular system
vector<ZZ> back_substitution(int n) {
    vector<ZZ> x(n);
    for (int i = n - 1; i >= 0; --i) {
        ZZ sum = number00;
        for (int j = i + 1; j < n; ++j)
            sum += (a[i][j] * x[j]) % Mod;

        ZZ invElofGroup = conv<ZZ>("1");
        ZZ temp = a[i][i];
        for (int ii = 1; ii < Mod; ii++)
        {
            ZZ antiDivider = temp * ii;
            ZZ antiDivider1 = antiDivider % Mod;
            if (antiDivider1 == 1)
            {
                invElofGroup = ii;
            }
        }

        x[i] = ((b[i] - sum) * invElofGroup) % Mod;

    }
    return x;
}
```

ДОДАТОК Б

Результат роботи програми КСІВРП-STARK-01

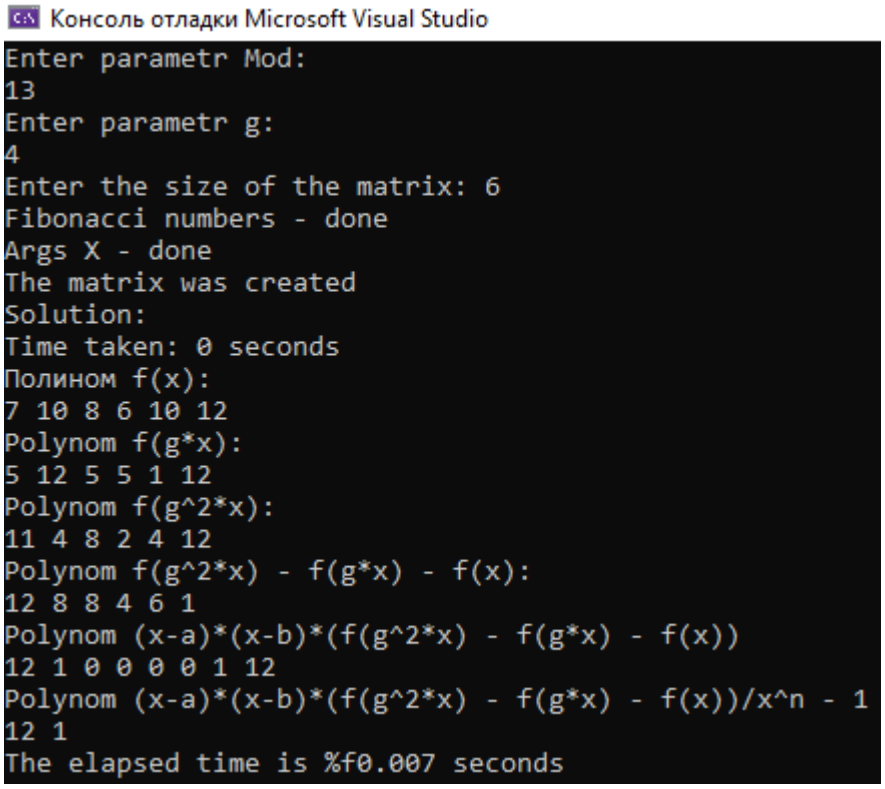

Рисунок Б - Результат роботи програми КСІВРП-STARK-01

ДОДАТОК В

Лістинг програми КСІВРП-STARK-02

```
#include <iostream>
#include <fstream>
#include <ctime>
#include <cmath>
#include "NTL\ZZ.h"

using namespace std;
using namespace NTL;
using namespace std;
using namespace std;

template <typename T> void FreeMem(T** matr, int n);
template <typename T> void SetMtx(T** matr, int n);
template <typename T> void TransponMtx(T** matr, T** tMatr, int n);
void inversion(ZZ** A, int N);
const int Marray = 6;
ZZ g = conv < ZZ>("4");
ZZ Mod = conv<ZZ>("13");
ZZ* argX = new ZZ[Marray];
ZZ* f_gx_ = new ZZ[Marray];
ZZ* f_g2x_ = new ZZ[Marray];
ZZ* subtractionArr = new ZZ[Marray];
ZZ* Coef = new ZZ[Marray];

void main()
{
   cout << "Enter parametr Mod: " << endl;
   cin >> Mod;
```

```
cout << "Enter parametr g: " << endl;
cin >> g;
double time_spent = 0.0;
clock_t begin = clock();

setlocale(0, "");
int n = 1;
ZZ number00 = conv < ZZ>("0");
ZZ number01 = conv < ZZ>("1");
ZZ number02 = conv < ZZ>("2");
const int size01 = 1;
cout << "Enter the size of the matrix: ";
n = Marray;
cout << n << endl;
ZZ** matr = new ZZ * [n];

ZZ Fibonacci[Marray][size01];

//=========================================================================
   // Формирование чисел Фибоначчи
Fibonacci[0][0] = conv < ZZ>("1");
Fibonacci[1][0] = conv < ZZ>("1");
for (int j = 2; j < Marray; j++)
{
   Fibonacci[j][0] = Fibonacci[j - 1][0] + Fibonacci[j - 2][0];
}
//=========================================================================

cout << "Fibonacci numbers - done" << endl;
//=========================================================================
   // Поиск значий циклической группы по заданному первообразному корню g
for (int j = 0; j < Marray; j++)
{
   argX[j] = PowerMod(g, j, Mod);//PowerMod(g, j, Mod)
```

```
}
ZZ tmp01[3]{};
tmp01[0] = 1;//при x^2
tmp01[1] = (-(argX[Marray - 1] + argX[Marray - 2]));//при x^1
tmp01[2] = ((argX[Marray - 1] * argX[Marray - 2]));//при x^0
cout << "Args X - done" << endl;
//=========================================================================



for (int i = 0; i < n; i++) {
    matr[i] = new ZZ[n];

}

cout << "The process of creating the matrix is underway. Please wait for..." << endl;
SetMtx(matr, n);

delete[] argX;
cout << "The matrix was created " << endl;
// FreeMem(matr, n);
cout << "The inverse matrix is being calculated. Please wait for..." << endl;
double time_spent1 = 0.0;
clock_t begin1 = clock();



ZZ temp;
cout << "I create an inverse matrix..." << endl;
ZZ** E = new ZZ * [n];

for (int i = 0; i < n; i++) {
    E[i] = new ZZ[n];
}

for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
```

```
    {
        E[i][j] = 0.0;

        if (i == j)
            E[i][j] = 1.0;
    }
cout << "An array of E..." << endl;

for (int k = 0; k < n; k++)
{
    temp = matr[k][k] % Mod;
    ZZ antiDet = conv<ZZ>("0");
    for (int ii = 1; ii < Mod; ii++)
    {
        ZZ antiDivider = temp * ii;
        ZZ antiDivider1 = antiDivider % Mod;
        if (antiDivider1 == 1)
        {
            antiDet = ii;
        }
    }
    for (int j = 0; j < n; j++)
    {
        matr[k][j] *= antiDet % Mod;
        E[k][j] *= antiDet % Mod;
    }

    for (int i = k + 1; i < n; i++)
    {
        temp = matr[i][k] % Mod;

        for (int j = 0; j < n; j++)
        {
            matr[i][j] -= (matr[k][j] * temp) % Mod;
            E[i][j] -= (E[k][j] * temp) % Mod;
```

```
        }
      }
    }
    cout << "The first cycle has passed..." << endl;
    for (int k = n - 1; k > 0; k--)
    {
      for (int i = k - 1; i >= 0; i--)
      {
        temp = matr[i][k] % Mod;

        for (int j = 0; j < n; j++)
        {
          matr[i][j] -= (matr[k][j] * temp) % Mod;
          E[i][j] -= (E[k][j] * temp) % Mod;
        }
      }
    }
    cout << "The second cycle has passed..." << endl;
    for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
        matr[i][j] = E[i][j] % Mod;

    for (int i = 0; i < n; i++)
      delete[] E[i];

    delete[] E;

    cout << "Array E is deleted ..." << endl;
    cout << "Transpose matrices..." << endl;

    for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
        matr[j][i] = matr[i][j];
  //
////=============================================================================================
===
```

```
    // ////                    Произведение матриц

    //
////=============================================================================================================
===
```

```
    cout << "Multiplying matrices..." << endl;

    //Умножение матриц

    ZZ** c = new ZZ * [Marray];

    for (int i = 0; i < n; i++)

    {

        c[i] = new ZZ[1];

        for (int j = 0; j < 1; j++)

        {

            c[i][j] = 0;

            for (int k = 0; k < Marray; k++)

                c[i][j] += (matr[i][k] * Fibonacci[k][j]) % Mod;

            c[i][j] = c[i][j] % Mod;

        }

    }

    //delete Fibonacci;

    //delete tobr_matr;



    FreeMem(matr, n);

    //FreeMem(tobr_matr, n);

    // Вывод матрицы произведения

    cout << "Work matrix" << endl;


    cout << "A one-dimensional array of coefficients: " << endl;

    int counter01 = 0;

    for (int i = Marray - 1; i >= 0; i--)

    {

        Coef[counter01] = c[i][0];

        // cout << Coef[i] << " ";
```

```
    counter01++;
}

clock_t end1 = clock();
// рассчитать прошедшее время, найдя разницу (end - begin) и
    // деление разницы на CLOCKS_PER_SEC для перевода в секунды
time_spent1 += (double)(end1 - begin1) / CLOCKS_PER_SEC;
cout << endl;
cout << "The elapsed time is %f" << time_spent1 << " seconds" << endl;

FreeMem(c, n);
cout << "Polynom f(x): " << endl;
for (int i = 0; i < Marray; i++)
{
    // Coef[i] = c[i][0];
    cout << Coef[i] << " ";
}



cout << endl;
cout << "Polynom f(g*x):" << endl;
int counter02 = Marray - 1;
for (int i1 = 0;i1 < Marray;i1++) {
    f_gx_[i1] = Coef[i1] * PowerMod(g, counter02, Mod);
    f_gx_[i1] = f_gx_[i1] % Mod;
    //cout << "f_gx_ [" << i1 << "] = " << f_gx_[i1] << endl;
    cout << f_gx_[i1] << " ";
    counter02--;
}
//delete[] Coef;
cout << endl;

cout << "Polynom f(g^2*x):" << endl;
ZZ number03 = PowerMod(g, number02, Mod);
int counter03 = Marray - 1;
```

```
for (int i = 0;i < Marray;i++) {
    f_g2x_[i] = Coef[i] * PowerMod(number03, counter03, Mod);
    f_g2x_[i] = f_g2x_[i] % Mod;
    //cout << "f_g2x_ [" << i << "] = " << f_g2x_[i] << endl;
    cout << f_g2x_[i] << " ";
    counter03--;
}
cout << endl;
cout << "Polynom f(g^2*x) - f(g*x) - f(x):" << endl;
for (int i = 0; i < Marray; i++) {
    subtractionArr[i] = f_g2x_[i] - f_gx_[i] - Coef[i];
    subtractionArr[i] = subtractionArr[i] % Mod;
    // cout << "subtractionArr [" << i << "] = " << subtractionArr[i] << endl;
    cout << subtractionArr[i] << " ";
}
cout << endl;
delete[]f_gx_;
delete[]f_g2x_;
delete[]Coef;

cout << "Polynom (x-a)*(x-b)*(f(g^2*x) - f(g*x) - f(x))" << endl;
const int NewSize = Marray + 2;
ZZ* mul = new ZZ[NewSize];



for (int i = Marray - 1; i >= 0; i--) {
    for (int j = 2; j >= 0; j--) {
        mul[i + j] += subtractionArr[i] * tmp01[j];//tmp02+
        mul[i + j] = mul[i + j] % Mod;

    }
}
delete[]subtractionArr;
int i10 = 0;
for (i10 = 0; i10 < NewSize; i10++) {
```

```
    // cout << "mul [" << i10 << "]  =" << mul[i10] << endl;
    cout << mul[i10] << " ";
}
cout << endl;
cout << "Polynom (x-a)*(x-b)*(f(g^2*x) - f(g*x) - f(x))/x^n - 1" << endl;
//=============================================================================================
//=============================================================================================
//=============================================================================================
//
    //Polynom temp = p1;
const int degreeOfNumerator = NewSize;//degree = 7
const int degreeOfDivider = Marray;//degree = 6
//cout << "max2 = " << NewSize << endl;
//cout << "Marray = " << Marray << endl;
int rdeg = degreeOfNumerator - degreeOfDivider;
//int rdeg2 = rdeg;

ZZ* res = new ZZ[NewSize - Marray + 1]{};
ZZ* PolynomialDivider = new ZZ[degreeOfNumerator]{};

for (int i10 = 0; i10 < degreeOfNumerator; i10++) {
    PolynomialDivider[i10] = 0.0;
}

PolynomialDivider[1] = 1;
PolynomialDivider[degreeOfNumerator - 1] = 12;

for (i10 = 0; i10 < degreeOfNumerator; i10++) {
    //cout << "PolynomialDivider [" << i10 << "]  =" << PolynomialDivider[i10] << endl;
}



//**************************************************Рабочее**************************************************
//cout << "degreeOfNumerator = " << degreeOfNumerator << endl;
```

```
    //cout << "rdeg = " << rdeg << endl;
    int counter04 = 1;

    for (int i11 = 0; i11 < rdeg; i11++) {
        res[rdeg - i11 - 1] = mul[0 + i11] / PolynomialDivider[1];
        /*cout << "res [" << i11<< "]  =" << (res[rdeg - i11 - 1]) << endl;*/
        cout << (res[rdeg - i11 - 1]) << " ";

        for (int j4 = 0; j4 <= degreeOfNumerator; j4++) {
            mul[0 + j4 + i11] -= PolynomialDivider[counter04] * res[rdeg - i11 - 1];

            counter04++;
            if (counter04 == NewSize) {
                break;
            }

        }

        counter04 = 1;

    }
    delete[]res;
    delete[]mul;
    delete[]PolynomialDivider;

    //*********************************************************************************************************
//*************<Time>*************************************************************
//*********************************************************************************************************
    clock_t end = clock();
    // рассчитать прошедшее время, найдя разницу (end - begin) и
        // деление разницы на CLOCKS_PER_SEC для перевода в секунды
    time_spent += (double)(end - begin) / CLOCKS_PER_SEC;
    cout << endl;
    cout << "The elapsed time is %f" << time_spent << " seconds" << endl;
} // end of main
```

```
//*****************************************************************************************************
//                    Дополнительные функции
//*****************************************************************************************************
//Функция транспонирования матрицы
template <typename T> void TransponMtx(T** matr, T** tMatr, int n) {//
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            tMatr[j][i] = matr[i][j];
}
//Функция освобождения памяти
template <typename T> void FreeMem(T** matr, int n)
{
    for (int i = 0; i < n; i++)
        delete[] matr[i];
    delete[] matr;
}

//функция заполнения матрицы
template <typename T> void SetMtx(T** matr, int n)
{
    cout << "Enter a matrix:\n";
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            matr[i][j] = PowerMod(argX[i], j, Mod);
        }
    }

}

//функция печати матрицы
template <typename T> void PrintMtx(T** matr, int n)
{
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
```

```
            cout << matr[i][j] << " ";
        cout << endl;

    }
}

void inversion(ZZ** A, int N)
{
    ZZ temp;

    ZZ** E = new ZZ * [N];

    for (int i = 0; i < N; i++)
        E[i] = new ZZ[N];

    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
        {
            E[i][j] = 0.0;

            if (i == j)
                E[i][j] = 1.0;
        }

    for (int k = 0; k < N; k++)
    {
        temp = A[k][k];
        ZZ antiDet = conv<ZZ>("0");
        for (int ii = 1; ii < Mod; ii++) {
            ZZ antiDivider = temp * ii;
            ZZ antiDivider1 = antiDivider % Mod;
            if (antiDivider1 == 1)
            {
                antiDet = ii;
            }
```

```
    }
    for (int j = 0; j < N; j++)
    {
        A[k][j] *= antiDet;
        E[k][j] *= antiDet;
    }

    for (int i = k + 1; i < N; i++)
    {
        temp = A[i][k];

        for (int j = 0; j < N; j++)
        {
            A[i][j] -= A[k][j] * temp;
            E[i][j] -= E[k][j] * temp;
        }
    }
}

for (int k = N - 1; k > 0; k--)
{
    for (int i = k - 1; i >= 0; i--)
    {
        temp = A[i][k];

        for (int j = 0; j < N; j++)
        {
            A[i][j] -= A[k][j] * temp;
            E[i][j] -= E[k][j] * temp;
        }
    }
}

for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
```

```
            A[i][j] = E[i][j] % Mod;


    for (int i = 0; i < N; i++)
        delete[] E[i];


    delete[] E;
}
```

ДОДАТОК Г

Результат роботи програми КСІВРП-STARK-02

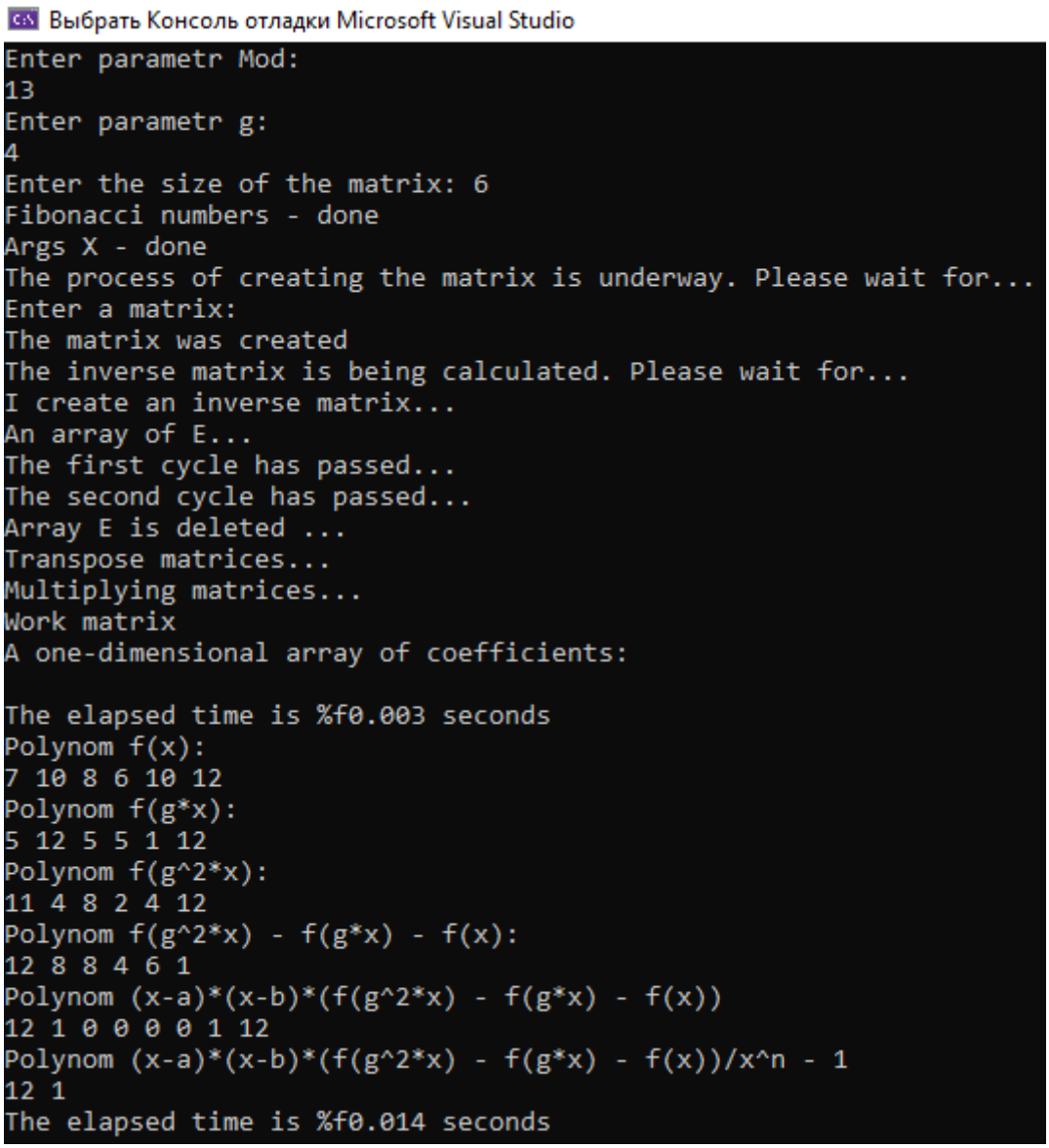

Рисунок Б - Результат роботи програми КСІВРП-STARK-01

ДОДАТОК Д

Розміри матриці, що являють собою розміри підгрупи, які створені першообразним елементом групи

Таблиця Д - Розміри матриці, що являють собою розміри підгрупи, які створені першообразним елементом групи

Розмір матриці, nxn	Група	Першообразний елемент групи
12x12	13	2
112x112	113	3
506x506	1013	9
1012x1012	1013	3
2112x2112	2113	5
3330x3330	3331	3
5004x5004	15013	7
7506x7506	15013	4
10056x10056	20113	3
15012x15012	15013	2
20787x20787	97013	11
24192x24192	96769	2

ДОДАТОК Е

Поліноміальна апроксимація за вхідними даними таблиці 4.1

Обробка результатів експерименту методом найменших квадратів

Вихідний набір експеременталь них точок:

Найменування: КПСІВРП-STARK-01

Метод: Гаус

$n := 6$

$x := (12 \quad 112 \quad 506 \quad 1012 \quad 2112 \quad 3330)^T$

$f := (0.007 \quad 0.103 \quad 6.68 \quad 52.352 \quad 469.331 \quad 1846.28)^T$

$m := 3$ - ступінь поліному, приймемо як базис апроксимації в вигляді ступенивих поліномів

Розв'язок:

$i := 0..m \qquad j := 0..m \qquad k := 0..n-1$

$A_{i,j} := \sum_k (x_k)^{i+j} \qquad B_j := \sum_k \left[(x_k)^j \cdot f_k \right] \qquad C := \left(A \cdot A^{-1}\right)^T A^{-1} \cdot B$

$$C = \begin{pmatrix} -0.188 \\ 2.831 \times 10^{-3} \\ -3.087 \times 10^{-6} \\ 5.068 \times 10^{-8} \end{pmatrix}$$

Создамо функцію Poly

$Poly(x) := \sum_i \left(C_i \cdot x^i \right) \qquad z := 0, 0.1..3000$ - діапазон точок для графіка функції

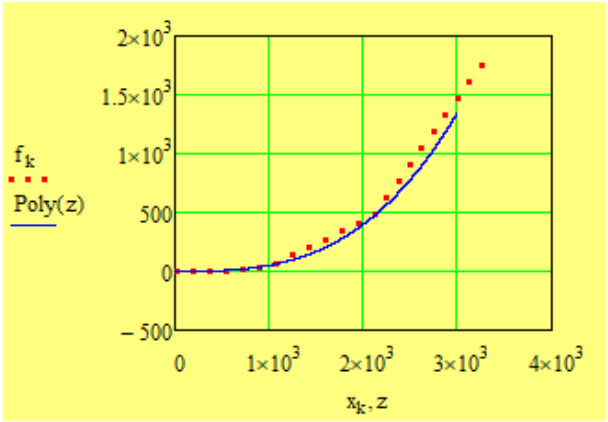

Рисунок Е - Поліноміальна апроксимація, що зроблена за допомогою методу обробки результатів експерименту методом найменших квадратів системи комп'ютерної алгебри Mathcad

ДОДАТОК Ж

Поліноміальна апроксимація за вхідними даними таблиці 4.1

Обробка результатів експерименту методом найменших квадратів

Вихідний набір експерементальних точок:

Найменування: КПСІВРП-STARK-02

$n := 5$

Метод: метод обернених матриць

$x := (12 \quad 112 \quad 506 \quad 1012 \quad 2112)^T \qquad f := (0.019 \quad 0.401 \quad 27.597 \quad 236.69 \quad 2187.62)^T$

$m := 3$ - ступінь поліному, приймемо як базис апроксимації в вигляді ступенивих поліномів

Розв'язок:

$i := 0..m \qquad j := 0..m \qquad k := 0..n-1$

$A_{i,j} := \sum_k (x_k)^{i+j} \qquad B_j := \sum_k \left[(x_k)^j \cdot f_k \right] \qquad C := \left(A \cdot A^{-1}\right)^T A^{-1} \cdot B$

$$C = \begin{pmatrix} 0.334 \\ -5.571 \times 10^{-3} \\ 1.739 \times 10^{-8} \\ 2.334 \times 10^{-7} \end{pmatrix}$$

Создамо функцію Poly

$Poly(x) := \sum_i \left(C_i \cdot x^i\right) \qquad z := 0, 0.1..2000$ - діапазон точок для графіка функції

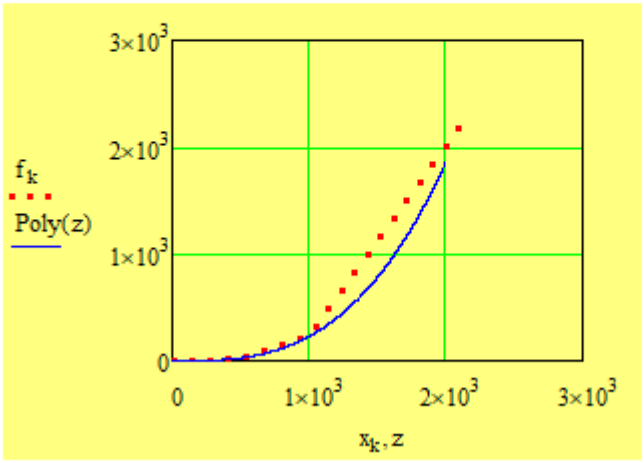

Рисунок Ж - Поліноміальна апроксимація, що зроблена за допомогою методу обробки результатів експерименту методом найменших квадратів системи комп’ютерної алгебри Mathcad

ДОДАТОК И

Поліноміальна апроксимація за вхідними даними таблиці 4.3

Обробка результатів експерименту методом найменших квадратів

Вихідний набір експерементальних точок:

Найменування: КПСІВРП-STARK-01

Метод: Гаус

$n1 := 6$

$x1 := (12 \quad 112 \quad 506 \quad 1012 \quad 2112 \quad 3330)^T$

$f1 := (0.94 \quad 0.99 \quad 13 \quad 47 \quad 200 \quad 495)^T$

$m1 := 1$ - ступінь поліному, приймемо як базис апроксимації в вигляді ступенивих поліномів

Розв'язок:

$i := 0..m1 \quad j := 0..m1 \quad k := 0..n1 - 1$

$A_{i,j} := \sum_k (x1_k)^{i+j} \quad B_j := \sum_k \left[(x1_k)^j \cdot f1_k \right] \quad C := \left(A \cdot A^{-1} \right)^T A^{-1} \cdot B$

$$C = \begin{pmatrix} -371.705 \\ -0.129 \\ 9.788 \times 10^{-4} \\ -1.637 \times 10^{-7} \end{pmatrix}$$

Создамо функцію Poly

$Poly(x1) := \sum_i \left(C_i \cdot x1^i \right)$ $\quad z1 := 0, 0.1..3000$ - діапазон точок для графіка функції

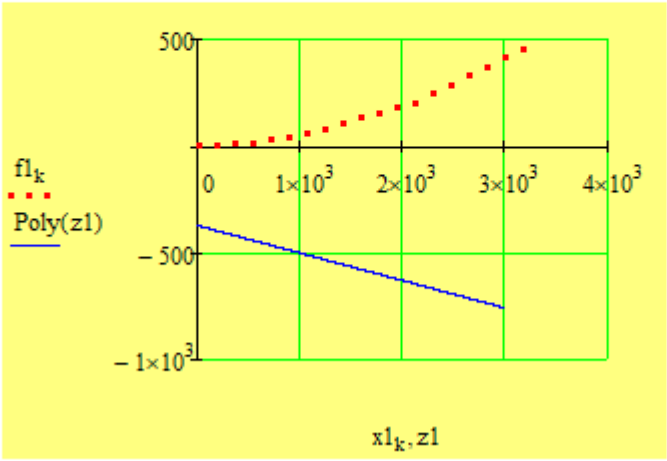

Рисунок И - Поліноміальна апроксимація, що зроблена за допомогою методу обробки результатів експерименту методом найменших квадратів системи комп'ютерної алгебри Mathcad

ДОДАТОК К

Поліноміальна апроксимація за вхідними даними таблиці 4.4

Обробка результатів експерименту методом найменших квадратів

Вихідний набір експерементальних точок:

Найменування: КПСІВРП-STARK-02

Метод: метод обернених матриць

$n := 6$

$x := (12 \quad 112 \quad 506 \quad 1012 \quad 2112 \quad 3330)^T \qquad f := (0.94 \quad 2 \quad 24 \quad 93 \quad 300 \quad 600)^T$

$m := 1$ - ступінь поліному, приймемо як базис апроксимації в вигляді ступенивих поліномів

Розв'язок:

$i := 0..m \qquad j := 0..m \qquad k := 0..n-1$

$A_{i,j} := \sum_k (x_k)^{i+j} \qquad B_j := \sum_k \left[(x_k)^j \cdot f_k \right] \qquad C := \left(A \cdot A^{-1} \right)^T A^{-1} \cdot B$

$$C = \begin{pmatrix} -341.001 \\ -0.083 \\ 8.424 \times 10^{-4} \\ -1.124 \times 10^{-7} \end{pmatrix}$$

Создамо функцію Poly

$Poly(x) := \sum_i \left(C_i \cdot x^i \right) \qquad z := 0, 0.1 .. 3000$ - діапазон точок для графіка функції

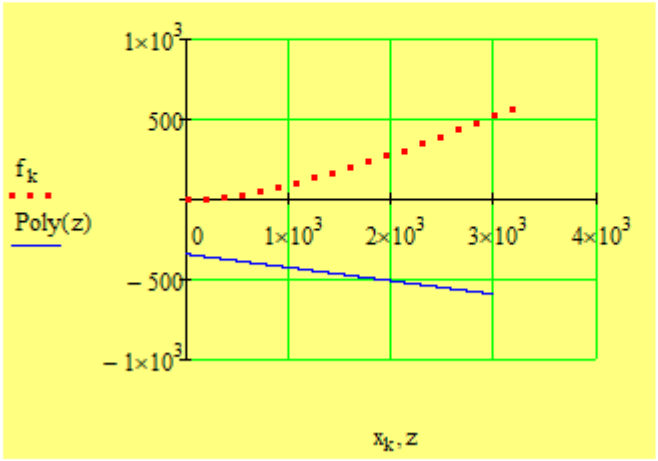

Рисунок К - Поліноміальна апроксимація, що зроблена за допомогою методу обробки результатів експерименту методом найменших квадратів системи комп'ютерної алгебри Mathcad

ДОДАТОК Л

Поліноміальна апроксимація за вхідними даними таблиці 4.5

Обробка результатів експерименту методом найменших квадратів

Вихідний набір експеременталь них точок:

Найменування: КПСІВРП-STARK-02

Метод: метод обернених матриць

$n := 12$

$x := (12 \quad 112 \quad 506 \quad 1012 \quad 2112 \quad 3330 \quad 5004 \quad 7506 \quad 10056 \quad 15012 \quad 20787 \quad 24192)^T$

$f := (1 \quad 38 \quad 976 \quad 3902 \quad 19417 \quad 50477 \quad 128518 \quad 289540 \quad 537690 \quad 1157806 \quad 2483554 \quad 3356745)^T$

$m := 1$ - ступінь поліному, приймемо як базис апроксимації в вигляді ступенивих поліномів

Розв'язок:

$i := 0..m \qquad j := 0..m \qquad k := 0..n-1$

$A_{i,j} := \sum_k (x_k)^{i+j} \qquad B_j := \sum_k \left[(x_k)^j \cdot f_k \right] \qquad C := \left(A \cdot A^{-1} \right)^T A^{-1} \cdot B$

$$C = \begin{pmatrix} -3.554 \times 10^5 \\ 131.757 \\ 0.541 \\ -2.481 \times 10^{-4} \end{pmatrix}$$

Создамо функцію Poly

$Poly(x) := \sum_i \left(C_i \cdot x^i \right)$ $\qquad z := 0, 0.1..30000$ - діапазон точок для графіка функції

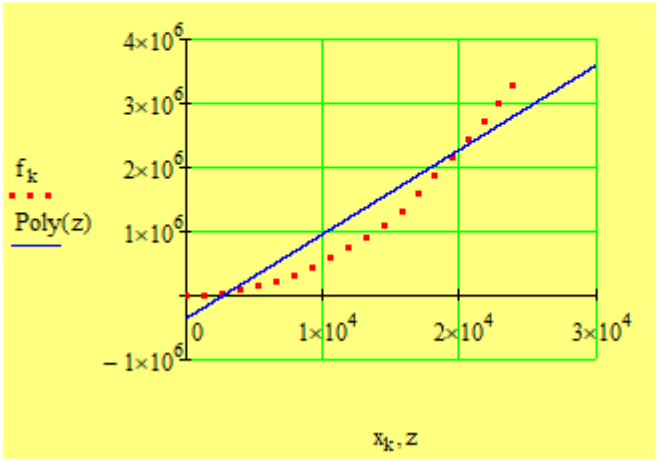

Рисунок Л - Поліноміальна апроксимація, що зроблена за допомогою методу обробки результатів експерименту методом найменших квадратів системи комп'ютерної алгебри Mathcad