Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“___” _____ 20__ р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти

на тему

“Дослідження та програмна реалізація системи пошуку даних у децентралізованих одноранговых мережах”

Виконав здобувач вищої освіти
ІІ курсу, групи КІ-22М-2
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Копаніцький С.М.
«____» _____ 20 р.

Керівник проекту
доктор технічних наук, професор
_____ Єлизавета МЕЛЕШКО
«____» _____ 20 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань *12 "Інформаційні технології"*

Спеціальність *123 "Комп'ютерна інженерія"*

Освітньо-професійна (освітньо-наукова) програма *"Комп'ютерна інженерія"*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____Олексій СМІРНОВ
« __ » _____ 20__ року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ДРУГИМ (МАГІСТЕРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Копаніцькому Сергію Михайловичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація системи пошуку даних у децентралізованих одноранговых мережах*

2. Керівник роботи *Мелешко Єлизавета Владиславівна, доктор техн. наук, професор*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу №__-__ від __.__.20__ року

3. Строк подання роботи до захисту __.__.*2023 р.*

4. Мета та завдання випускної кваліфікаційної роботи: *Дослідження та програмна реалізація системи пошуку даних у децентралізованих одноранговых мережах*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Наукова новизна
7. Економічна ефективність розробленої програми.
8. Заходи з охорони праці та техніки безпеки.
9. Висновки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Наукова новизна	*1 аркуш*
Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*
Діаграма процесів	*1 аркуш*
Показники економічної ефективності	*1 аркуш*

6. Консультанти по роботі, із зазначенням розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Савеленко Г.В., к.т.н., доцент	25.10.2023	10.11.2023
Охорона праці	Оришака О.В., к.т.н., доцент	03.11.2023	21.11.2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.10.2023 р.	
2.	Постановка задачі, оформлення ТЗ	16.10.2023 р.	
3.	Розробка моделі компонента	20.10.2023 р.	
4.	Розробка структур даних	25.10.2023 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.10.2023 р.	
6.	Програмування алгоритмів	10.11.2023 р.	
7.	Розрахунок економічної ефективності	13.11.2023 р.	
8.	Розрахунки з охорони праці та техніки безпеки	15.11.2023 р.	
9.	Оформлення ПЗ	17.11.2023 р.	
10.	Попередній захист роботи	10.12.2023 р.	

Дата видачі завдання
«__»_____20 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
«__»_____20 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Копаніцький С.М. Дослідження та програмна реалізація системи пошуку даних у децентралізованих однорангових мережах. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2023.

В даній магістерській роботі розроблено програмне забезпечення, яке призначено для реалізації системи пошуку даних у децентралізованих однорангових мережах.

Метою розробки є дослідження та програмна реалізація системи пошуку даних у децентралізованих однорангових мережах.

Об'єктом дослідження є процес обробки даних у децентралізованих однорангових комп'ютерних мережах.

Предметом дослідження є методи пошуку даних у децентралізованих однорангових комп'ютерних мережах.

Методи дослідження базуються на теорії комп'ютерних мереж, теорії алгоритмів і структур даних, теорії графів, теорії статистики, теорії імітаційного моделювання та методах розробки програмного забезпечення.

Результат роботи – програмна реалізація системи пошуку даних у децентралізованих однорангових мережах.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ архітектури IBM PC з ОС Windows 10/11.

Програму розроблено на мові програмування високого рівня Python.

Ключові слова: комп'ютерна інженерія, комп'ютерні мережі, однорангові мережі, децентралізовані мережі, пошук даних.

ABSTRACT

Kopanitskyi S.M. Research and software implementation of a data retrieval system in decentralised peer-to-peer networks. 123 Computer engineering. Central Ukrainian National Technical University. Kropyvnytskyi 2023.

In this master's thesis, software designed for a simulation system of computer network and data transfer processes.

The purpose of the development is the research and program implementation of a simulation system for computer network and data transfer processes.

The object of the research is the process of data processing in decentralized peer-to-peer computer networks.

The subject of the research is the methods of searching for data in decentralized peer-to-peer computer networks.

The research methods are based on the theory of computer networks, the theory of algorithms and data structures, the theory of graphs, the theory of statistics, the theory of simulation modeling, and methods of software development.

The result of the work is the software implementation of a simulation system for computer network and data transfer processes.

In the process of working on a software model an analysis of existing hardware and software was performed. All components of the software developed are fully described.

User-friendly user interface is developed. These are instructions for working with software.

The program can be used on PCs of IBM PC architecture with Windows 10/11 OS.

The program was developed in the high-level programming language – Python.

Keywords: computer engineering, simulation modeling, computer modeling, computer networks, data retrieval.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

BitTorrent – це протокол і система обміну файлами, які дозволяють користувачам спільно завантажувати та роздавати файли у P2P-мережах; забезпечує швидкий та ефективний обмін файлами між багатьма користувачами.

DHT (Distributed Hash Table) – це розподілена хеш-таблиця, яка використовується для зберігання інформації та пошуку даних в децентралізованих системах. DHT розподіляє дані між вузлами мережі, використовуючи ключі хешування для швидкого доступу до даних.

ECDSA (Elliptic Curve Digital Signature Algorithm) – це алгоритм цифрового підпису, який використовується для перевірки автентичності та цілісності даних. Він базується на криптографії на еліптичних кривих і часто використовується для забезпечення безпеки в мережах, таких як блокчейн і криптовалютні системи.

Gnutella – це протокол та система обміну файлами у P2P-мережах, які дозволяють користувачам спільно завантажувати та роздавати файли. Gnutella має відкрите програмне забезпечення та багато різних реалізацій клієнтів.

ID (identifier) – це ідентифікатор або унікальний номер, який відокремлює один об’єкт від інших у контексті конкретної системи чи мережі.

Kademlia – це алгоритм для структуризації децентралізованих обчислювальних систем, таких як P2P-мережі. Він використовує спеціальну систему хеш-функцій для розподілення даних і вузлів у мережі, що дозволяє швидко знаходити інформацію в такій мережі.

P2P (peer-to-peer) – це технологія обміну ресурсами або інформацією у комп’ютерних мережах без посередництва центрального сервера. У P2P мережах кожен вузол (або пір) є однаковим рівноправним учасником, який може спільно з іншими вузлами обмінюватися даними, ресурсами або послугами.

TCP (Transmission Control Protocol) – це протокол передачі даних у комп'ютерних мережах, який забезпечує надійний та послідовний обмін інформацією між вузлами. Він відповідає за розбиття та складання повідомлень, контроль цілісності даних та перевірку доставки.

UDP (User Datagram Protocol) – це протокол передачі даних у мережах, який працює на транспортному рівні. Він надає швидку передачу даних, але не гарантує надійності або послідовності доставки, що робить його підходящим для передачі відео, аудіо даних тощо.

Децентралізована мережа – це тип мережі, в якій управління та контроль розподілені між різними вузлами чи членами мережі, зазвичай без наявності центрального адміністратора або сервера. Це дозволяє досягти вищого рівня резистентності до відмов та більшої стійкості. Є частковим випадком однорангових мереж, у якому не використовується трекер для координації зв'язку між клієнтами.

Одноранговa мережа (також відома як P2P мережа) – це мережа, в якій всі вузли мають однакові права та можуть безпосередньо взаємодіяти один з одним без центрального сервера. Кожен вузол може виконувати як клієнтські, так і серверні функції.

ВСТУП

Актуальність теми. Пошук даних у децентралізованих однорангових мережах на сьогоднішній день є важливою і актуальною проблемою в інформаційному суспільстві. Децентралізовані мережі, такі як блокчейн, файлообмінники, децентралізоване телебачення та стримінг й інші подібні технології, набувають все більшого поширення і важливості в різних сферах, включаючи фінанси, розподілені системи, обробку даних тощо.

Тож, на сьогоднішній день відбувається зростання кількості децентралізованих систем та даних у них. Вони дозволяють користувачам здійснювати безпосередні обміни даними та ресурсами, обходячи посередників і централізовані структури. Це може бути особливо корисним у сферах, де забезпечення довіри та прозорості грає важливу роль, наприклад, у фінансових операціях та управлінні ланцюгами постачання.

Забезпечення ефективного та безпечного пошуку даних у децентралізованих мережах є важливим завданням для розробників та дослідників. Така система повинна бути здатна враховувати особливості децентралізованих мереж, такі як відсутність централізованого управління та глобальної ідентифікації, а також забезпечувати конфіденційність та безпеку даних під час пошуку.

До того ж, розробка системи пошуку даних у децентралізованих мережах вимагає інноваційних підходів до обробки даних, розподіленого обчислення та забезпечення надійності. Вона також може мати важливий вплив на розвиток децентралізованих технологій та їхнє впровадження у реальному світі.

Отже, програмна реалізація системи пошуку даних у децентралізованих однорангових мережах є актуальною і важливою задачею, що відкриває нові можливості для забезпечення надійності та ефективності обміну даними в децентралізованих середовищах.

Мета й завдання дослідження. Метою розробки є дослідження та програмна реалізація системи пошуку даних у децентралізованих однорангових мережах.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Дослідження існуючих систем пошуку даних у децентралізованих однорангових мережах.

– Розробка методів та алгоритмів роботи системи пошуку даних у децентралізованих однорангових мережах.

– Програмна реалізація системи пошуку даних у децентралізованих однорангових мережах.

Об'єктом дослідження є процес обробки даних у децентралізованих однорангових комп'ютерних мережах.

Предметом дослідження є методи пошуку даних у децентралізованих однорангових комп'ютерних мережах.

Методи дослідження базуються на теорії комп'ютерних мереж, теорії алгоритмів і структур даних, теорії графів, теорії статистики, теорії імітаційного моделювання та методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

1. Запропоновано метод імітаційного моделювання децентралізованих однорангових комп'ютерних мереж та процесів передачі даних у них.

2. Розроблено вітчизняний продукт для системи пошуку даних у децентралізованих однорангових комп'ютерних мережах, який має більш широкі можливості, на відміну від існуючих аналогів та може бути використаний у розподілених системах зберігання даних.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі пошуку даних у децентралізованих однорангових мережах, а також програмного імітаційного

моделювання цього процесу.

Достовірність наукових результатів підтверджена теоретичними викладками, результатами комп'ютерного імітаційного моделювання, результатами тестування розробленого програмного забезпечення, а також відповідністю отриманих результатів окремим результатам, наведеним у науковій літературі.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація пошуку даних у децентралізованих однорангових мережах, є актуальною задачею, яка потребує вирішення у даній кваліфікаційній магістерській роботі.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Система пошуку даних у децентралізованих однорангових мережах призначена для знаходження і доступу до ресурсів, інформації або служб, розподілених між різними вузлами мережі без централізованого сервера чи координатора. Основні призначення системи пошуку даних у децентралізованих P2P мережах включають такі аспекти:

Децентралізовані однорангові мережі мають широкий спектр застосувань і можуть бути використані для різних видів додатків і сервісів. Ось деякі з них:

1. **Файлообмінники**. Децентралізовані P2P-мережі широко використовуються для файлообміну, дозволяють користувачам спільно обмінюватися файлами без централізованого сервера.

2. **Криптовалютні біржі**. Криптовалютні біржі базуються на технології блокчейн і використовують P2P-мережі для обробки транзакцій та створення нових блоків.

3. **Децентралізовані соціальні мережі**. Деякі соціальні мережі і форуми використовують P2P-технології для забезпечення анонімності та безпеки користувачів і дозволяючи їм створювати децентралізовані спільноти.

4. **Хмарні сховища даних**. Деякі додатки для зберігання файлів використовують P2P-мережі для розподіленого зберігання файлів в децентралізованому середовищі.

5. **Інтернет речей (IoT)**. Деякі системи в IoT використовують P2P-мережі для підключення і керування пристроями, що розташовані в різних місцях.

6. **Онлайн-ігри**. Деякі онлайн-ігри використовують P2P технології для створення серверів та обміну даними між гравцями.

7. **Додатки для обчислень**. Додатки для розподілених обчислень використовують P2P-мережі для розподілу завдань та ресурсів між вузлами.

8. **Інтернет-телебачення та стрімінг**. Деякі стрімінгові платформи використовують P2P-технології для розподілу відео- та аудіоконтенту, що дозволяє зменшити навантаження на сервери.

Загальна ідея системи пошуку даних у децентралізованих P2P мережах полягає в тому, щоб забезпечити користувачам можливість ефективно знаходити та отримувати доступ до ресурсів, розподілених у мережі, навіть без централізованого управління.

1.2 Область застосування

Системи пошуку даних у децентралізованих однорангових мережах мають широкий спектр застосувань у різних галузях та сценаріях. Тож, користувачами однорангових децентралізованих мереж можуть бути:

– будь-які індивідуальні користувачі, які потребують обмінювати файлами або ресурсами, такими як музика, відео, документи тощо;

– користувачі, які шукають конфіденційні та безпечні спільноти та засоби спілкування у децентралізованих соціальних мережах;

– інвестори, майнери, користувачі криптовалют і блокчейн-платформ;

– організації, які потребують безпечного і розподіленого зберігання даних;

– організації та розробники IoT-проектів;

– гравці онлайн-ігор для створення серверів і обміну даними;

– науковці та організації, що потребують великої обчислювальної потужності;

– тощо.

Застосування систем пошуку даних у децентралізованих однорангових мережах стають все більш важливими в сучасному світі, оскільки вони дозволяють забезпечити високу доступність, надійність та збереження даних без залежності від централізованих інфраструктур та посередників.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми магістерської роботи

P2P-мережі пропонують децентралізований та стійкий підхід до спільного використання ресурсів та зв'язку, що робить їх цінними для різних додатків, де важливі масштабованість і надійність. Центрального серверу немає, і однорангові вузли безпосередньо обмінюються ресурсами та спілкуються один з одним, використовуючи певні протоколи. Мережа є надійною, масштабованою та ефективною, що робить її корисною для різних програм, таких як обмін файлами, розподілені обчислення та зв'язок.

Приклади P2P-мереж:

– BitTorrent – популярний протокол обміну файлами, який використовується для поширення великих файлів серед багатьох користувачів.

– **eMule –** популярний клієнт для файлообміну в децентралізованій мережі. Він дозволяє користувачам обмінюватися різними видами файлів.

– Bitcoin – криптовалютна мережа, яка базується на технології блокчейн і використовує P2P-протокол для обробки транзакцій та створення нових блоків.

– **Ethereum –** криптовалютна мережа, яка використовує P2P-модель. Вона включає смарт-контракти та додатки, які виконуються на блокчейні.

– **Ripple –** криптовалютна мережа і платіжна система, яка використовує P2P-мережу для обміну валюти і виконання платежів.

– Skype – у попередніх версіях Skype спілкування між користувачами здійснювалося через мережу P2P.

– **Tox –** безпечний і конфіденційний месенджер, який використовує P2P-протокол для обміну повідомленнями.

– **ZeroNet –** децентралізована платформа для створення та розгортання веб-сайтів, яка використовує P2P-мережу для розподіленого зберігання та доступу до вмісту.

– **Ace Stream –** популярний стрімінговий сервіс, який використовує P2P-технології для надання можливості глядачам обмінюватися відеоконтентом під час онлайн-трансляцій.

– **WebTorrent –** технологія та платформа для стрімінгу відео та аудіо за допомогою P2P-мережі прямо в браузері.

– **SopCast –** P2P-стрімінговий сервіс, який дозволяє користувачам транслювати і дивитися відео та телевізійні канали в режимі реального часу.

Алгоритми та застосування мереж P2P

Вперше термін "peer-to-peer" була використана в 1984 році компанією IBM у розробці мережевої архітектури для побудови динамічної маршрутизації через комп'ютерні мережі з довільною топологією – Advanced Peer to Peer Networking.

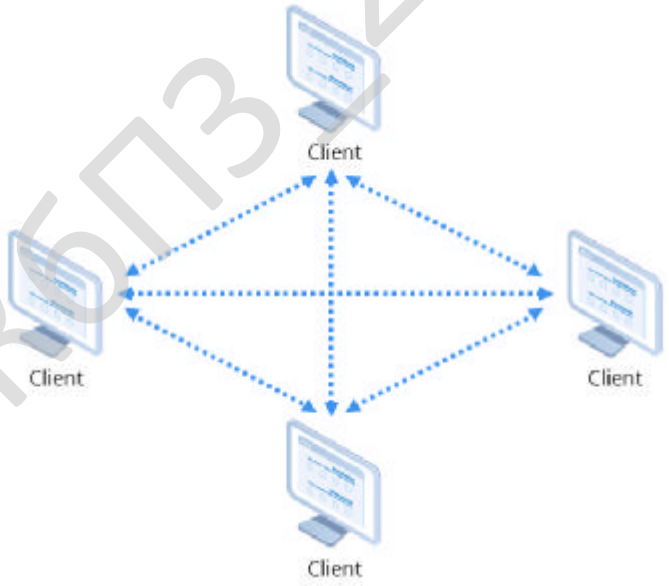

Рисунок 2.1 – P2P-мережа

P2P-мережа складається із рівноправних вузлів. Кожен вузол може взаємодіяти з кожним.

В основі технології лежить принцип децентралізації, тобто всі вузли мережі P2P – рівноправні (рис. 2.1). Цей принцип забезпечив такі переваги технології P2P перед клієнт-серверним підходом, як відмовостійкість до втрати зв'язку з вузлами мережі, збільшення швидкості копіювання за рахунок копіювання відразу з декількох джерел, можливість поділу ресурсів без прив'язки до конкретних IP-адрес, величезна потужність мережі загалом та інших.

Таблиця 2.1 – Характеристики P2P-мереж

Характеристика	Опис
Децентралізація	Немає центрального сервера; усі вузли мають однакове повноваження, що робить мережу стійкою до збоїв
Спільне використання ресурсів	Вузли спільно використовують файли, пропускну здатність і обчислювальну потужність, підвищуючи доступність ресурсів
Масштабованість	Розмір мережі може змінюватися динамічно без впливу на продуктивність, нові функції можна легко додати
Анонімність	Високий рівень анонімності, але його можна використати для незаконних дій
Безпека	Стійкі до деяких атак, але вразливий до слабких місць протоколу або вузла
Топологія мережі	Можуть бути повністю децентралізовані або гібридні залежно від цілей та складності
Автономність	Користувачі можуть контролювати ресурси, але це може вплинути на координацію та продуктивність мережі

Створювані програми на початку ставили вузьку мету – прискорення отримання копії файлу, іноді за рахунок скорочення смуги та мережевих

можливостей конкуруючих процесів та користувачів. У цих додатках використовувалася особливість протоколу TCP, що розподіляє смугу пропускання порівну між конкуруючими процесами.

Відмовостійкість, властива алгоритмам P2P, може стати причиною впровадження цієї технології для управління технологічно небезпечними виробництвами, наприклад атомними електростанціями. Тут важливо, щоб вузли P2P не розташовувалися в тому самому приміщенні і не були підключені до загальних каналів живлення. Вихід із ладу частини керуючих машин може зменшити функціональність, але не керованість системи.

P2P може запропонувати досить високий рівень катастрофостійкості, коли потрібно гарантувати збереження певних даних у разі пожежі або якихось інших природних або рукотворних інцидентів. Для таких програм географічна віддаленість вузлів є важливою перевагою.

Для підтримки надійності обміну можуть використовуватися різні технології, зокрема, протокол FEC (Forward Error Correction) або MDC (Multi Description Coding) у разі транспортування мультимедійних даних, коли повторна передача пакетів неможлива або вкрай небажана.

Незамінними можуть виявитися системи P2P у банківському бізнесі, де важливо забезпечити збереження даних та високу надійність розрахунків. Методики дублювання у банківських додатках використовуються вже досить давно.

Широко використовувані зараз розподілені обчислювальні системи типу GRID є окремим випадком реалізації технології P2P.

Так, за деякими даними, в даний час в мережі Internet більше половини всього трафіку припадає на трафік файлообмінних P2P-мереж, а розміри найбільших з них перевалили позначку мільйон одночасно працюючих вузлів, що розділяють петабайти. Загальна кількість зареєстрованих учасників файлообмінних мереж P2P у всьому світі становить близько ста мільйонів.

Хоча в першу чергу P2P-мережі використовуються зараз для поділу файлів, існує ще багато інших областей, де ця технологія також успішно застосовується. Це телебачення та аудіо трансляції, паралельне програмування, розподілене кешування ресурсів для розвантаження серверів, розсилка повідомлень та статей, підтримка системи доменних імен, індексування розподілених ресурсів та їх пошук, резервне копіювання та створення стійких розподілених сховищ даних, обмін повідомленнями, створення систем-серверів стійких до атак типу «відмова у обслуговуванні», поширення програмних модулів. Є безліч клієнтських програм для роботи з P2P-мережами, як комерційних, так і з відкритим кодом. Постійно триває робота з удосконалення протоколів та збільшення функціональності систем, і, зважаючи на все, недалекий той момент, коли клієнтське програмне забезпечення для P2P буде інтегроване з операційними системами.

У існуючих сьогодні реалізаціях P2P файлообмінних мереж, спочатку був використаний змішаний підхід, з присутністю виділених вузлів (і/або серверів), проте останні кілька років усі найбільші мережі включили підтримку протоколів, що забезпечують повністю автономне функціонування мережі без серверів – чистий P2P-підхід.

P2P-протокол – мережевий протокол, що забезпечує можливість створення та функціонування мережі рівноправних вузлів, їх взаємодії.

Протоколом, або набором протоколів, визначається логічна топологія мережі, механізм підключення та відключення вузлів від мережі, а також алгоритм взаємодії вузлів. Вирішення таких завдань, як корекція помилок, формати повідомлень та службових запитів та відгуків, протоколи маршрутизації в умовах постійного підключення та відключення вузлів – також визначається протоколом P2P.

У моделі стека мережевих протоколів TCP/IP протоколи P2P відносяться до прикладного рівня, таким чином, P2P мережа є накладеною мережею (overlay),

що функціонує поверх мережі Інтернет і використовує існуючі транспортні протоколи TCP або UDP.

Клієнтська програма P2P – програма, що реалізує функціональність вузла, сама є реалізацією закладеного основою мережі P2P протоколу. Клієнт може запитувати сервер або виділені вузли, отримувати відповідь з інформацією про запитані файли, вузли на яких вони знаходяться, і далі працювати безпосередньо з зазначеними вузлами. В останніх реалізаціях клієнтів закладено також можливість обміну службовою інформацією, побудови запитів та пошуку ресурсів клієнтом у всій мережі без участі серверів.

ID вузла – унікальний ідентифікатор вузла, що обчислюється за допомогою хеш-функції з IP – адреси та додаткової інформації (імені комп'ютера, MAC-адреси мережевої карти тощо). Надається при реєстрації в мережі P2P і використовується для ідентифікації вузла.

ID (або ключ) ресурсу – унікальний ідентифікатор файлу або будь-якого іншого ресурсу, обчислюється за допомогою хеш-функції з імені файлу та його вмісту. Використовується для визначення ресурсу.

Протоколами забезпечується рівномірний розподіл ключів ресурсів разом з ідентифікаторами вузлів, що опублікували даний ресурс, по всіх вузлах (або деяких виділених вузлів та/або серверів), зареєстрованих в мережі. Завдання пошуку ресурсу зводиться до знаходження ID вузла, у якому зберігається ключ ресурсу (рис. 2.2).

На рис. 2.2 наведено приклад чистої мережі P2P, створеної за протоколом Kademlia з використанням розподілених хеш-таблиць (DHT). На рисунку до мережі, здатної підтримувати максимум 16 вузлів та 16 ресурсів, приєдналися 7 вузлів (чорні кружки), що розділяють 12 ресурсів (номери у білих прямокутниках). Вузлам присвоєно відповідні ID, ресурсам присвоєно ключі. Ключі разом з адресами вузлів, що їх опублікували (адреси на рисунку не показані) рівномірно розподілені між вузлами мережі. На рисунку зображено, які ключі зберігаються і яких вузлах.

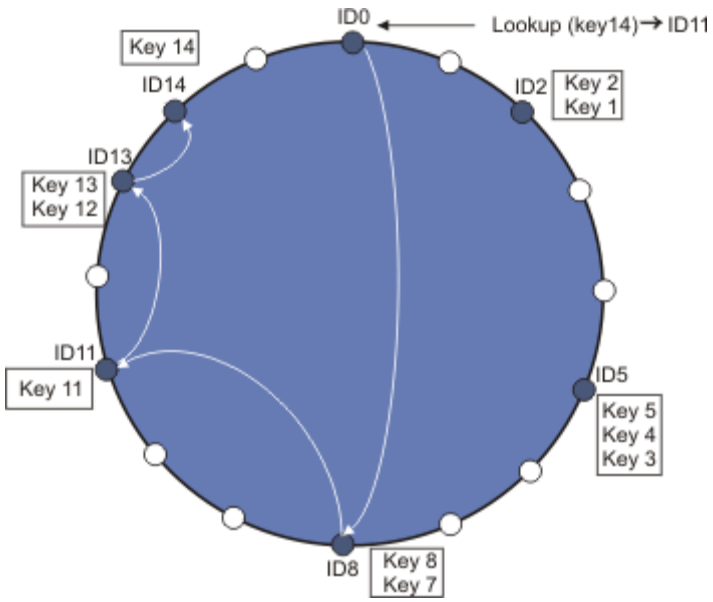

Рисунок 2.2 – Розподіл ключів ресурсів на вузлах P2P-мережі та пошук ресурсу

Вузол з ID 0 хоче знайти ресурс, який відповідає ключу 14, для цього посилає запит на пошук. Запит проходить певний маршрут і досягає вузла, на якому знаходиться ключ 14. Далі вузол ID 14 пересилає вузлу ID 0 адреси всіх вузлів, що володіють ресурсом, що відповідає ключу 14.

Причинами великого зростання популярності мереж P2P є привабливі ідеї даної технології: децентралізація, розподіленість та самоорганізованість мережі. Ці принципи забезпечують такі переваги мережі, як простота та дешевизна впровадження та підтримки, стійкість до відмов, масштабованість, збільшення швидкості копіювання, колосальна потужність мережі в цілому.

Однією з перших пірінгових мереж була Gnutella, створена в 2000. Мережа функціонує досі, хоча через серйозні недоліки алгоритму користувачі зараз віддають перевагу мережі Gnutella2.

Клієнт, що підключився, отримує від вузла, з яким йому вдалося з’єднатися, список з 5 активних вузлів, відсилає їм запит на пошук ресурсу за ключовим словом. Вузли шукають відповідні запиту ресурси і, якщо не знаходять, пересилають запит своїм активним вузлам вгору по дереву, поки не знайдеться

ресурс або не буде перевищено максимальну кількість кроків. Такий пошук називається розмноженням запитів. За допомогою протоколу також відстежується, щоб топологія мережі вузлів, що створюється, мала деревоподібну структуру графа.

Зрозуміло, що такий алгоритм веде до експоненційного зростання кількості запитів і, відповідно, на верхніх рівнях може призвести до відмови в обслуговуванні, що й спостерігалося практично неодноразово. Розробниками було проведено роботи з поліпшенню алгоритму, запроваджено правила, відповідно до якими запити можуть пересилати вгору деревом лише певні вузли – звані виділені вузли, інші вузли можуть лише посилати їм запити.

Недоліки протоколу Gnutella ініціювали розробки принципово нових алгоритмів пошуку маршрутів та ресурсів і призвели до створення групи протоколів DHT (Distributed Hash Tables), зокрема протоколу Kademlia, який зараз широко використовується в найбільших мережах.

Запити в мережі Gnutella пересилаються TCP або UDP протоколом, копіювання файлів відбувається за допомогою протоколу HTTP.

Останнім часом з’явилися розширення для клієнтських програм, що дозволяють копіювати файли UDP, робити XML-запити метаінформації про файли.

У 2003 році з’явився новий протокол Gnutella2. Відповідно до цього протоколу, деякі вузли стають концентраторами, інші є звичайними вузлами. Кожен звичайний вузол має з’єднання з одним-двома концентраторами. У концентратора є зв’язок із сотнями звичайних вузлів та десятки з’єднань з іншими концентраторами. Кожен вузол періодично пересилає концентратору список ідентифікаторів ключових слів, за якими можуть бути знайдені ресурси, що публікуються даним вузлом. Ідентифікатори зберігаються у загальній таблиці на концентраторі. Коли вузол хоче знайти ресурс, він посилає запит за ключовим словом свого концентратора, який або знаходить ресурс у своїй таблиці і повертає ID-вузла, що володіє ресурсом, або повертає список

інших концентраторів, які вузол знову запитує по черзі випадковим чином. Такий пошук називається пошуком за допомогою методу блукань.

Для покращення якості пошуку використовуються метадані файлів – інформація про зміст, рейтинг. Примітною особливістю даної мережі є можливість розмноження інформації про файл у мережі без копіювання файлу, що дуже корисно для відстеження вірусів.

Для пакетів, що передаються в Gnutella2 розроблений власний формат, схожий на XML, що гнучко реалізує можливість нарощування функціональності мережі, шляхом додавання додаткової службової інформації. Запити та списки ID ключових слів надсилаються за протоколом UDP.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Вибір мови програмування для розробки системи пошуку даних у децентралізованих однорангових мережах є важливою задачею, від якої буде залежати продуктивність системи та вартість її розробки. Python є однією з популярних мов програмування, і він може бути гарним вибором для розробки такої системи з наступних причин:

1. **Простота і читабельність коду**. Python славиться своєю простотою та легкістю читання коду. Це дозволяє розробникам створювати зрозумілий та підтримуваний код, що особливо корисно у децентралізованих мережах, де спільна робота розробників може бути необхідною.

2. **Велика кількість бібліотек**. Python має велику кількість сторонніх бібліотек та фреймворків, які спрощують розробку. Для P2P-мереж, наприклад, існують бібліотеки, такі як Twisted і asyncio, які допомагають створювати асинхронні додатки для обробки мережевої взаємодії.

3. **Переносимість**. Python є переносимою мовою програмування, що означає, що код, написаний на Python, може працювати на різних операційних

системах, включаючи Windows, macOS і Linux. Це важливо для забезпечення сумісності з різними платформами.

4. **Спільнота та документація**. Python має велику та активну спільноту розробників, а також обширну документацію. Це робить Python добрим вибором для розробників, які шукають підтримку та ресурси для роботи над складними проектами.

5. **Широкі можливості мережевого програмування**. Python надає потужні бібліотеки для мережевого програмування, що дозволяє легко створювати, керувати та взаємодіяти з мережевими з'єднаннями. Це важливо для розробки P2P-систем.

6. **Швидкість розробки**. Python є високорівневою мовою програмування, що сприяє швидкій розробці прототипів та експериментів. Це корисно при створенні та тестуванні нових ідей у P2P-системах.

7. **Підтримка асинхронного програмування**. Завдяки асинхронним фреймворкам, таким як asyncio, Python дозволяє створювати ефективні та швидкі додатки для роботи у децентралізованих мережах, де асинхронна обробка запитів є важливою.

Загалом, Python є мовою програмування з великою кількістю переваг, які роблять її відмінним вибором для розробки систем пошуку даних у децентралізованих однорангових мережах. Вона дозволяє розробникам швидко створювати функціональні та ефективні рішення для різних застосувань у цьому контексті.

2.3 Розгорнута постановка завдання

У сучасному світі існує зростаюча потреба в розробці децентралізованих систем пошуку даних, які б забезпечували високу доступність, надійність і приватність при пошуку і обміні інформацією. Метою цієї роботи є створення системи пошуку даних у децентралізованій одноранговій мережі, яка здатна

забезпечити ефективний та швидкий пошук даних серед вузлів цієї мережі.

Вимоги до розроблюваного програмного забезпечення:

1. Створення імітаційної моделі децентралізованої P2P-мережі. Кожен вузол мережі у моделі повинен мати можливість підключатися до інших вузлів та обмінюватися даними.

2. Пошук даних за запитами користувачів. Система повинна підтримувати можливість пошуку даних, які задаються користувачами у вигляді запитів.

3. Розподілена індексація даних. Для підвищення ефективності пошуку, система повинна підтримувати розподілену індексацію даних. Кожен вузол повинен мати можливість індексувати свої власні дані та надавати іншим вузлам доступ до індексу.

4. Захист приватності і безпеки. Система повинна забезпечувати захист приватності користувачів та безпеку даних під час їх передачі та зберігання в мережі. Механізми шифрування та автентифікації повинні бути впроваджені.

5. Масштабованість та надійність. Система повинна бути масштабованою і надійною, здатною працювати у великих мережах.

6. Графічний інтерфейс користувача. Система повинна мати графічний інтерфейс для користувачів, який дозволяє їм вводити пошукові запити та переглядати результати пошуку.

Технічні вимоги:

- Мова програмування Python.
- Використання бібліотек та фреймворків для мережевого програмування та розподіленої індексації даних.
- Використання криптографічних бібліотек для забезпечення безпеки даних.
- Використання графічних бібліотек для створення інтерфейса користувача та візуалізації імітаційної моделі однорангової мережі.

Завдання та етапи реалізації:

1. Розробка мережевої архітектури для децентралізованої P2P-мережі.

2. Реалізація механізму індексації та пошуку даних в мережі.

3. Забезпечення приватності та безпеки даних у мережі.

4. Розробка графічного інтерфейсу користувача для взаємодії з системою.

5. Тестування та оптимізація системи для забезпечення ефективності та надійності.

Розробка системи пошуку даних у децентралізованих однорангових мережах вимагає використання сучасних мережевих та криптографічних технологій, а також глибокого розуміння принципів децентралізованих мереж та пошуку даних.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Для реалізації роботи децентралізованої однорангової мережі та пошуку даних у ній було вирішено використати розподілені хеш-таблиці і протокол Kademlia.

Розподілена хеш-таблиця (DHT) – це свого роду структура даних, яка зберігається на кількох комп’ютерах і спрямована на вирішення проблеми маршрутизації та доступу до даних в однорангових децентралізованих мережах.

Структура DHT може бути розбита кілька основних компонентів. Вона ґрунтується на абстрактному просторі ключів (keyspace), такому як набір 160-бітних рядків (кількість біт може змінюватись). Схема розбиття простору ключів розподіляє належність ключів серед вузлів, що беруть участь. Потім оверлейна мережа з’єднує вузли, допомагаючи знайти власника будь-якого ключа у просторі ключів. Комп’ютери-учасники можна розглядати як такі, що утворюють кільцеву структуру в порядку зростання за годинниковою стрілкою. Ключі, призначені певному комп’ютеру, будуть ключами, розташованими між ідентифікатором комп’ютера та ідентифікатором комп’ютера після нього. Ця геометрія також називається узгодженим хешуванням (рис. 3.1).

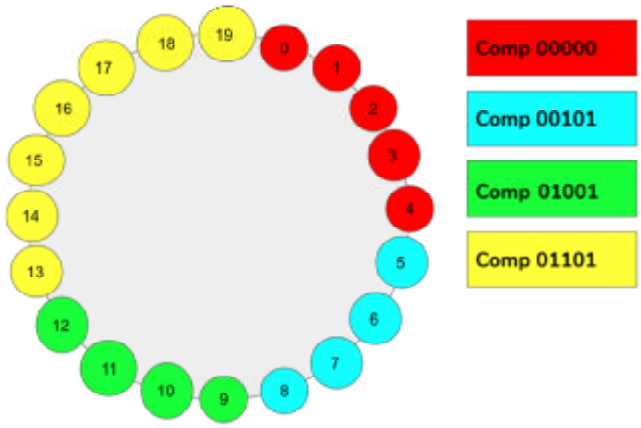

Рисунок 3.1 – Геометрія кільцевого узгодженого хешування

Показано чотири комп’ютери. А саме, у нас є комп’ютер 00000, комп’ютер 00101, комп’ютер 01001 і комп’ютер 01101. Тобто комп’ютери 0, 5, 9 і 13 відповідно. Ці числа призначаються випадковим чином. За кольоровим кодом ми бачимо, що комп’ютер 5 містить усі ідентифікатори від 5 до значення наступного комп’ютера (9). Цей малюнок продовжується навколо кільця.

Це забезпечує переваги, згадані вище, завдяки тому, що кільце стає все щільнішим і щільнішим, оскільки в ньому бере участь все більше комп’ютерів. Подібним чином, ключові проміжки між ідентифікаторами комп’ютерів, швидше за все, зменшаться та стануть відносно однорідними, оскільки більше комп’ютерів долучатимуться до мережі з випадковими ідентифікаторами. Комп’ютер з id може зберігати адреси комп’ютерів у формі successor((id+2^i) для різних ступенів двох до n, де n – кількість комп’ютерів у кільці (рис. 3.2).

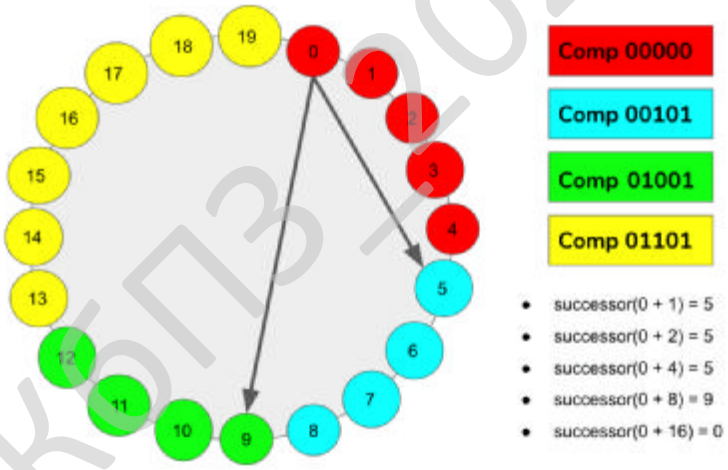

Рисунок 3.1 – Комп’ютери, видимі з ідентифікатора 0

Тобто ми починаємо з 0 і додаємо послідовні ступені 2 до кількості вузлів в рингу. У нашому випадку конкретно 20 комп’ютерів. Це означає, що ми додамо числа 1, 2, 4, 8 і 16 до нашого початкового вузла (0) і викличемо послідовник (k) для кожного з цих вузлів. Наш результат – список комп’ютерів, видимих з ідентифікатора вузла 0.

Пошуки можуть складатися з вузлів, які рекурсивно звертаються до відомого їм вузла, який є найближчим до цільового ключа. Тривалість виконання пошуку вузла O(log(*n*)).

Протокол Kademlia

Припустимо, що всі ідентифікатори та ключі знаходяться в просторі ключів [0,1,…23−1][0,1,…23−1] і представлені у двійковому вигляді. Ми можемо представити цей простір як повне бінарне дерево, де кожен листовий вузол є ключем. Обведені кружками листочки – це ідентифікатори, які відповідають комп'ютеру, що бере участь у мережі. У наведеному вище прикладі (рис. 3.2) три комп'ютери беруть участь у протоколі з ідентифікаторами 000, 110 і 111 відповідно.

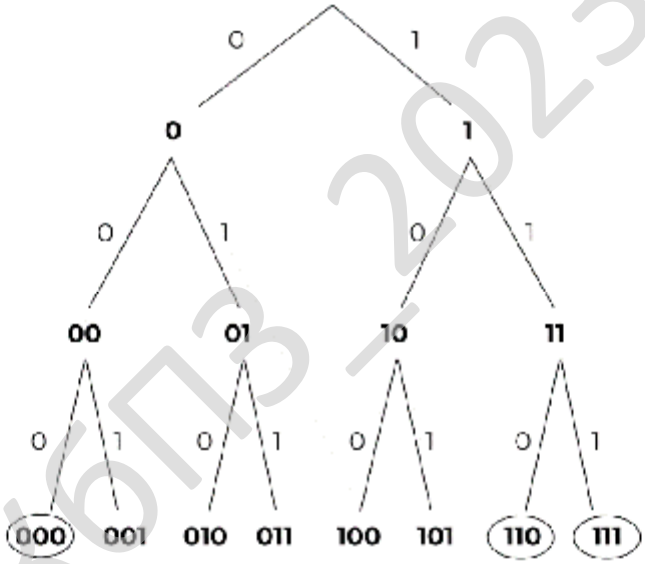

Рисунок 3.3 – Бінарне дерево для пошуку, у якому обведені листочки представляють ідентифікатори комп'ютерів

Природною сегментацією може бути призначення ключа вузлу з найменшим спільним предком. Потім можна закодувати діаграму наступним кольором, де листя того самого кольору, що й лист комп'ютера, означають, що ключі мають зберігатися на цьому комп'ютері (рис. 3.4).

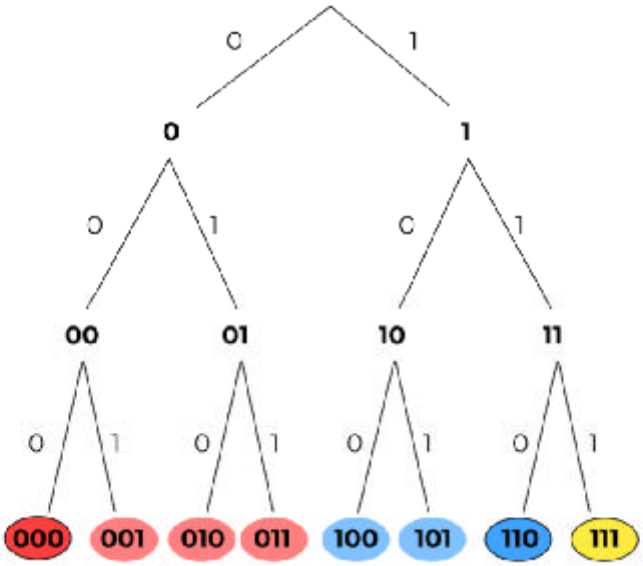

Рисунок 3.4 – Бінарне дерево для пошуку – листки позначені кольором, щоб показати, що ліві 4 біти мають найменшого спільного предка з 000 і, таким чином, призначені цьому комп'ютеру

Подібним чином 100 і 101 призначаються комп'ютеру з ідентифікатором 110, а 111 нічого не призначено. Насправді це помилково, але згодом спонукатиме до «поліпшення» близькості.

На жаль, серед комп'ютерних вузлів існувала рівність щодо найнижчого спільного предка: 110 і 111 обидва мають спільного предка, що представляє спільний префікс «1__». Таким чином, було дещо довільним вибором призначити ці вузли комп'ютеру 110.

Якщо два комп'ютери мають найдовший загальний префікс, замість цього можна переглянути найстарший бітовий індекс *b* де вони відрізняються. Тоді ми можемо перевірити біт *b* ключа та призначити ключ комп'ютеру, біт якого *b* у його id рівний біту *b* ключа.

Умовно нехай 100_i буде *i*-м бітом 100, з індексом 0 зліва направо. Наприклад, $100_0 = 1$. Далі розглянемо ключ 100 та ідентифікатори комп'ютерів 110 і 111. І 100, і 110, і 111 мають спільний префікс 1. Однак біт 1 110 і 111 однакові: 110_1. Отже, розглянемо перший біт, за яким вони

відрізняються: біт 2. Оскільки 110_2 але $111_2 \neq 100_2$, 100 слід призначити комп'ютеру з ідентифікатором 110.

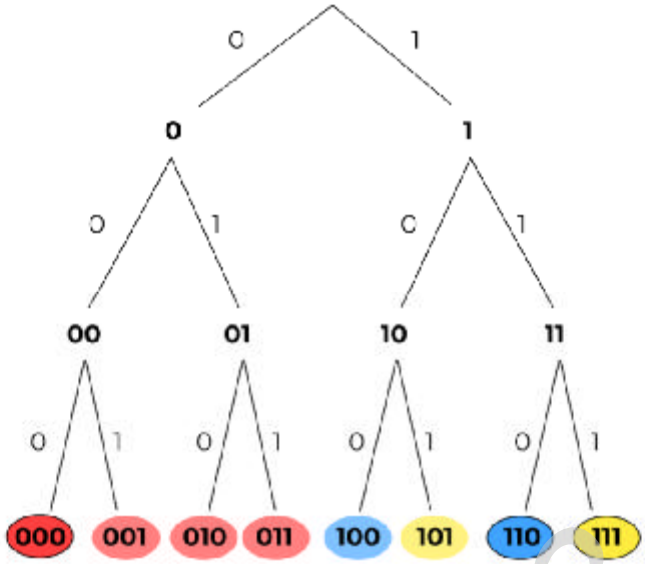

Рисунок 3.5 – Бінарне дерево для пошуку з правильними призначеннями від 100 до 110 і від 101 до 111

Якщо в наборі кандидатів залишилося більше одного ідентифікатора комп'ютера, треба переходити за бітовим індексом від старшого біта до молодшого. Якщо принаймні один ідентифікатор кандидата має той самий біт, що й ключ у нашому поточному індексі біта, видаліть усі ідентифікатори кандидатів, які відрізняються від ключа з цим індексом біта. Якщо у нас залишився один ідентифікатор, цей ідентифікатор комп'ютера є найближчим ідентифікатором.

Для механічного визначення відстані між ключем та ідентифікатором комп'ютера треба ініціалізувати значення відстані на 0 та порівняти ключ і ідентифікатор комп'ютера порозрядно. Якщо ключ і ідентифікатор комп'ютера відрізняються в *i*-му молодшому значущому біті (причому молодший значущий біт має індекс i=0), додати штраф у розмірі 2^i до значення відстані.

Оперативне визначення відстані між ключами та ідентифікаторами комп'ютера – відстань між ключем та ідентифікатором комп'ютера є XOR їхніх значень, інтерпретованих як ціле число.

Таким чином, ключ має зберігатися на комп’ютері, який є найближчим до ключа або має найнижче значення XOR для ключа відносно всіх інших ідентифікаторів комп’ютера.

З цього моменту відстань визначатиметься як XOR між двома ідентифікаторами/ключами, інтерпретованими як невід’ємне ціле число, а «найближчим» комп’ютером до ключа є комп’ютер із найменшою відстанню XOR між його ідентифікатором і ключем.

Розділи двійкового дерева як таблиці маршрутизації

Потрібно визначити, як комп’ютери можуть знаходити ключі з інших комп’ютерів. Інтуїтивно зрозуміло, що ми хотіли б зберігати лише адреси невеликої кількості комп’ютерів, але мати можливість відносно швидко знаходити ключі. Однією з можливих цілей є те, що кожен запит до комп’ютера має наблизити нас принаймні «на один біт» до комп’ютера, який насправді містить цікавий ключ. Це може приблизно дати нам час роботи $O(\log(n))$. Припустімо, що всі наші комп’ютери-учасники існують на початку і ніколи не виходять, тому їх можна представити деревом, показаним вище.

Тому ми могли б зберігати логарифмічне число ідентифікаторів комп’ютерів і відповідних їм IP-адрес в організованій книзі контактів.

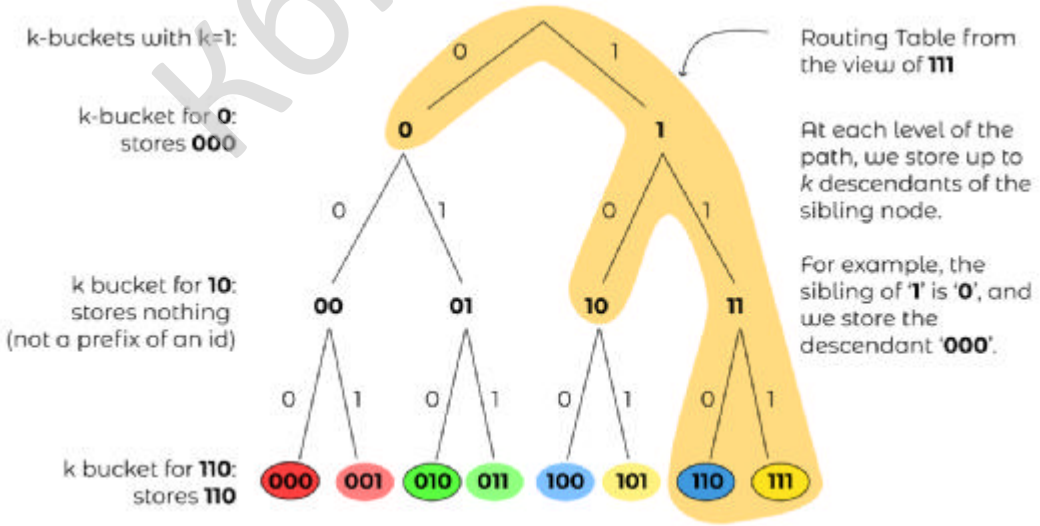

Рисунок 3.6 – Таблиця маршрутизації комп’ютера з ідентифікатором 111 із відповідним *k*-бакетом для *k*=1.

У цих однорідних вузлах ми можемо просто зберігати будь-який комп’ютер, ідентифікатор якого є нащадком цього однорідного вузла. Наприклад, для вузла, який представляє префікс «0», ми можемо зберегти «000». Немає комп’ютерів, які починаються з префікса «10», тому внутрішній вузол буде порожнім. Для вузла, який представляє префікс "110", ми можемо зберегти адресу комп’ютера "110". Кожен із цих однорідних вузлів називається «k-бакетом».

Пошук ключів/комп’ютера

Розглянемо як буде працювати запит GET(011) для комп’ютера 111. Цей ключ зберігатиметься на комп’ютері 010. Однак «010» не вказано в адресній книзі нашого вузла 111. Найближчим комп’ютером до 010, про який ми знаємо, є комп’ютер 000. Тоді ми можемо попросити комп’ютер 000 повідомити нам, чи має він ключ 011, чи знає комп’ютер 000 будь-які ближчі до 011 комп’ютери. Комп’ютер 000, у свою чергу, повинен мати 010 у своїй адресній книзі , тому 000 може повернути IP-адресу 010 до 111, а потім 111 може запитати у 010 ключ 011, який він повинен мати.

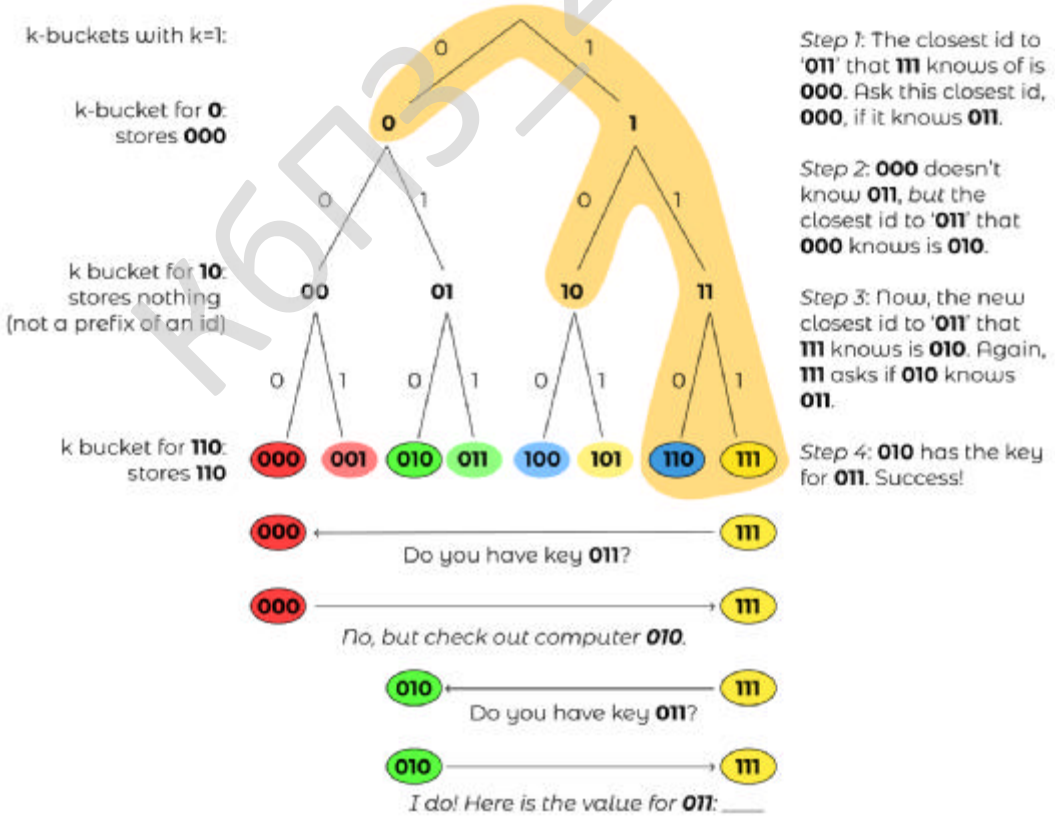

Рисунок 3.7 – Комп’ютер з ID 111 хоче виконати GET(011)

Формалізувати цей процес можна так:

1.Знайдіть найближчий комп’ютер у таблиці маршрутизації (через XOR).

2.Хоча найближчий комп’ютер, який ви знаєте, не має ключа та ще не відповів, зробіть наступні кроки.

3.Попросіть найближчий комп’ютер, який ви знаєте, щоб отримати ключ або ближчий комп’ютер.

4.Якщо найближчий комп’ютер відповідає, оновіть нашу змінну найближчого комп’ютера.

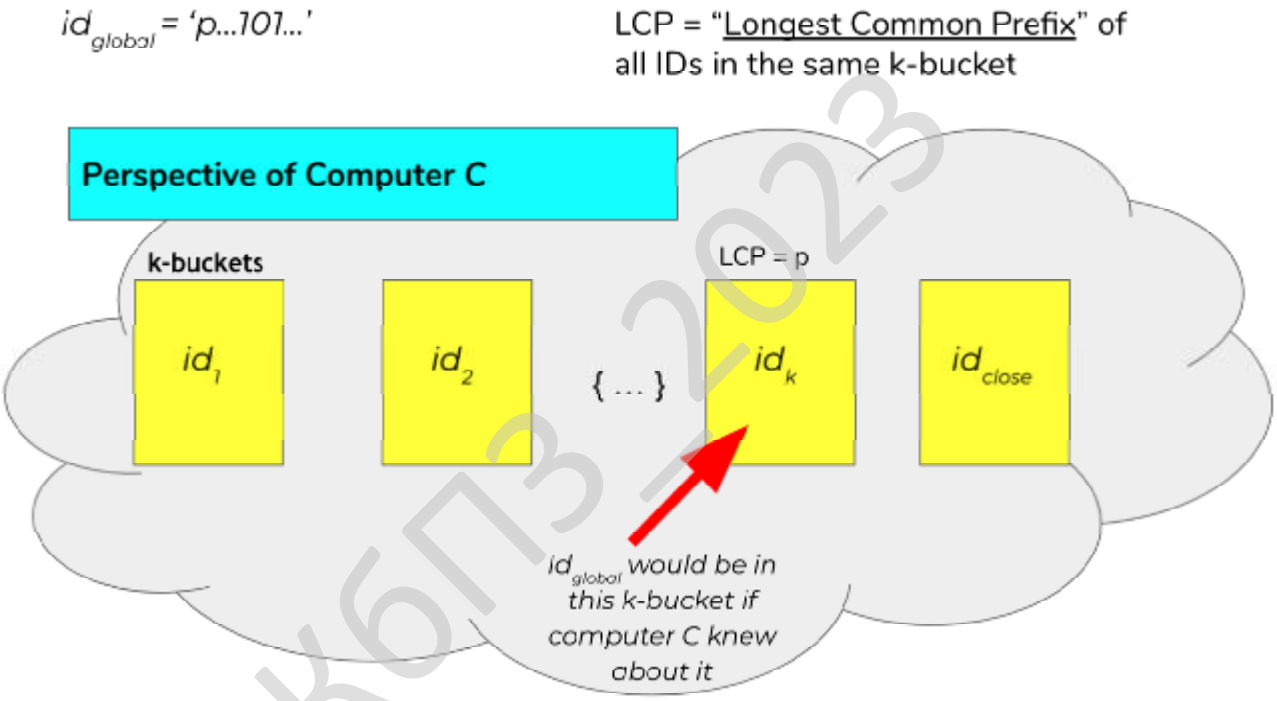

Рисунок 3.8 – Таблиця маршрутизації комп’ютера *c*

Кожен з *k*-бакетів містить ідентифікатор із певним найдовшим загальним префіксом. Наприклад, *k*-бакет зліва містить id 1 і представляє всі ідентифікатори, які відрізняються від ідентифікатора *c* першим бітом.

Операція GET(key) – комп’ютерний пошук ключа з наступним викликом FIND_VALUE() для *k* найближчих комп’ютерів до ключа за ідентифікатором. Операція PUT(key, value) – комп’ютерний пошук у key з наступною операцією STORE(key) на *k* найближчих комп’ютерах.

3.2 Розробка структурної схеми

Структурна схема розробленого програмного забезпечення зображена на рисунку 3.9.

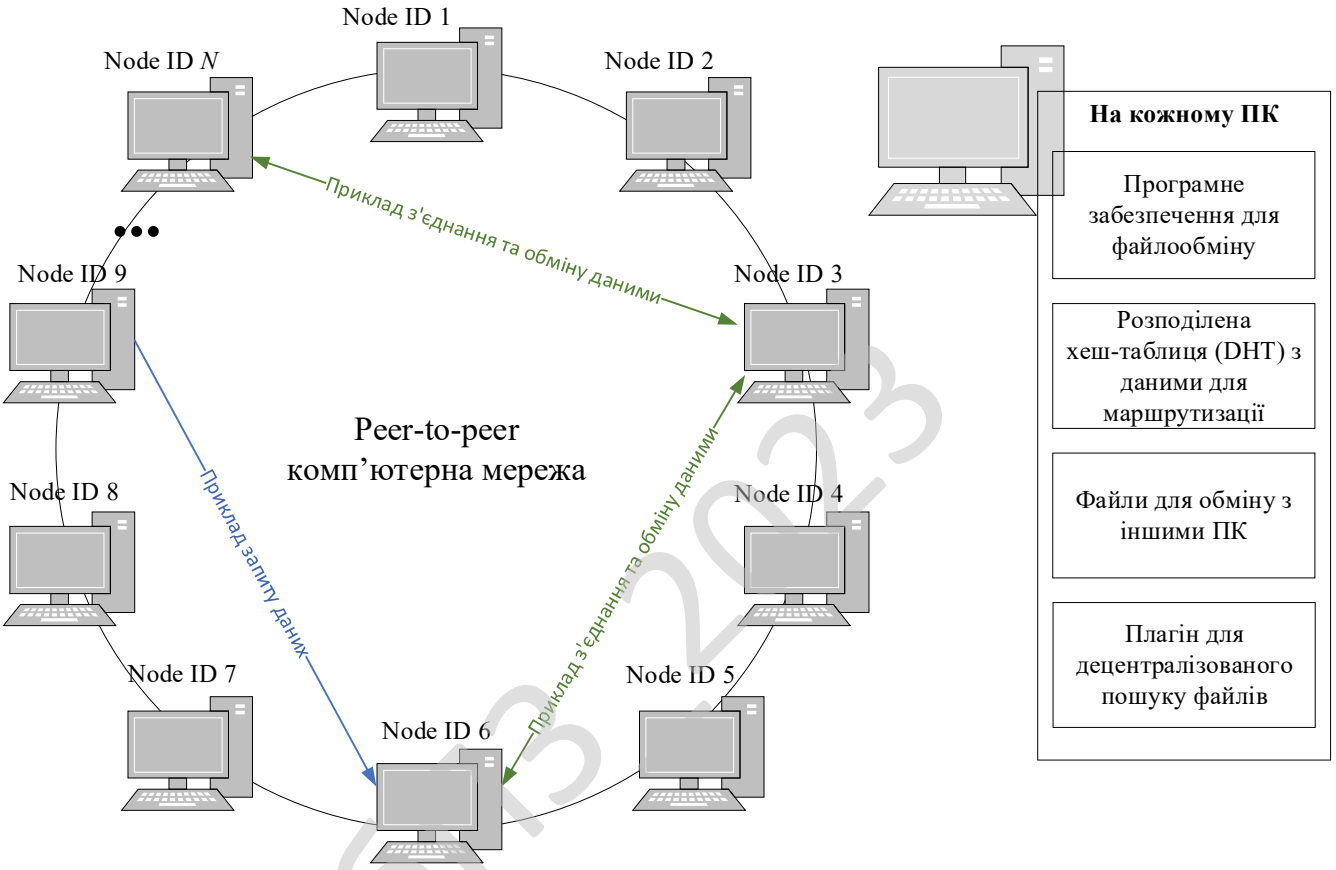

Рисунок 3.9 – Структурна схема системи

З рисунку видно, що система складається з рівноправних вузлів, кожен з яких представлений комп'ютером та містить наступні програмні елементи:

– Програмне забезпечення для файлообміну.

– Розподілену хеш-таблию (DHT) з даними для маршрутизації.

– Файли для обміну з іншими ПК.

– Плагін для децентралізованого пошуку файлів.

3.3 Розробка функціональної схеми

Функціональна схема системи наведена на рис. 3.10.

Рисунок 3.10 – Функціональна схема системи

Розроблена система складається з наступних модулів:

– Програмна модель однорангової децентралізованої комп'ютерної мережі.

– Метод пошуку даних в одноранговій комп'ютерній мережі.

Програмна модель однорангової децентралізованої комп'ютерної мережі містить наступні елементи:

– Моделювання структури комп'ютерної мережі у вигляді стохастичного графа.

– Моделювання контенту однорангової комп’ютерної мережі.

– Моделювання пошукових запитів користувачів комп’ютерної мережі.

– Відстеження результатів моделювання, візуалізація та текстові звіти.

Метод пошуку даних в одноранговій комп’ютерній мережі містить наступні елементи:

– Маршрутизацію трафіку однорангової децентралізованої комп’ютерної мережі.

– Таблиці маршрутизації у вигляді розподілена хеш-таблиці (DHT).

– Пошук та фільтрація даних.

– Базу даних з посиланнями на знайдені дані.

3.4 Розробка діаграми процесів

На рисунку 3.11 зображена діаграма процесів, що описує наявні у системі процеси та їх взаємодію.

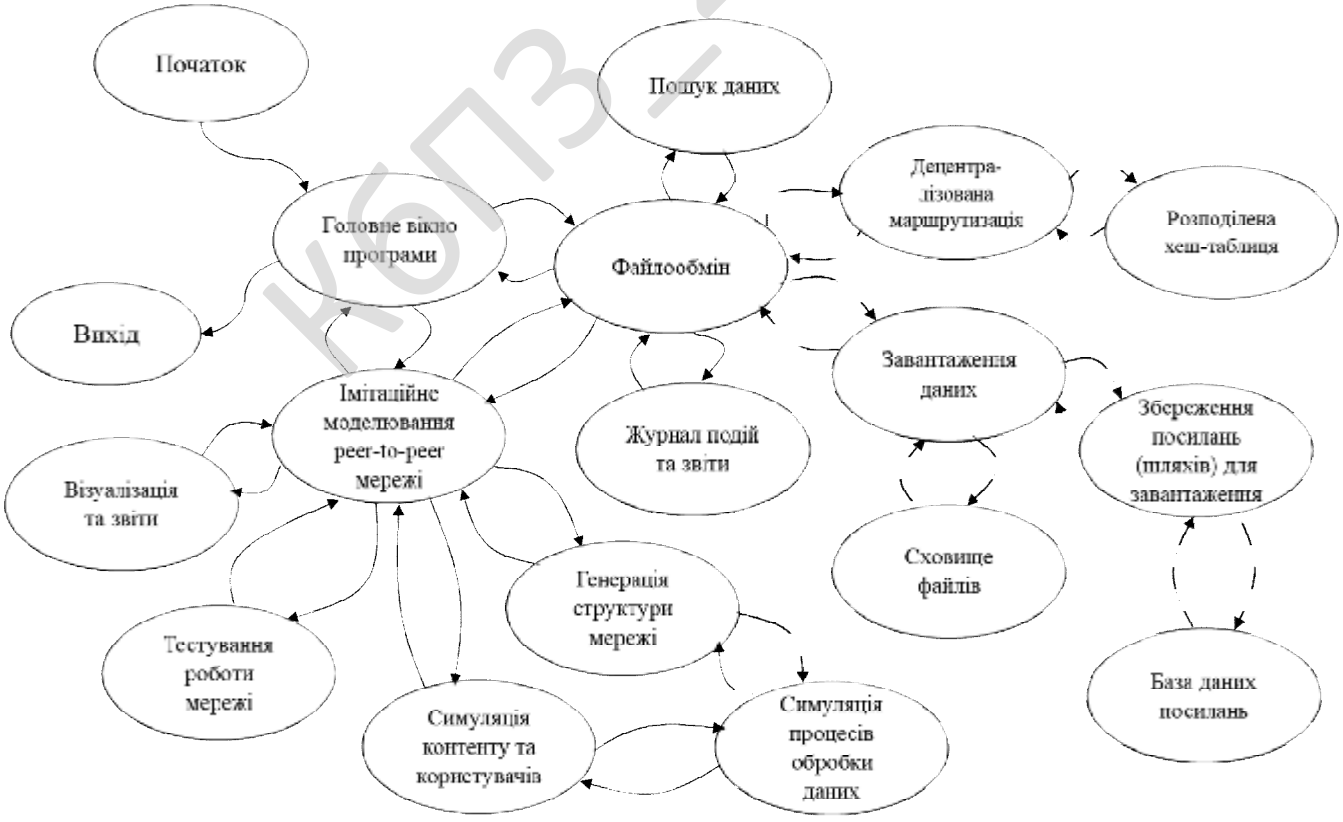

Рисунок 3.5 – Діаграма процесів системи

У розробленій системі присутні наступні процеси:

- Головне вікно програми.
- Файлообмін.
- Пошук даних.
- Журнал подій та звіти.
- Децентралізована маршрутизація.
- Розподілена хеш-таблиця.
- Завантаження даних.
- Сховище файлів.
- Збереження посилань (шляхів) для завантаження.
- База даних посилань.
- Імітаційне моделювання peer-to-peer мережі.
- Генерація структури мережі.
- Симуляція процесів обробки даних.
- Симуляція контенту та користувачів.
- Тестування роботи мережі.
- Візуалізація та звіти.

Розроблена діаграма відображає послідовність роботи основних процесів, що реалізують роботу усієї системи.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Блок-схема основної програми зображена на рис. 4.1. Вона відображає логіку роботи розробленого програмного забезпечення та послідовність виконання дій для реалізації програмного імітаційного моделювання однорангової децентралізованої мережі та процесу пошуку даних у ній.

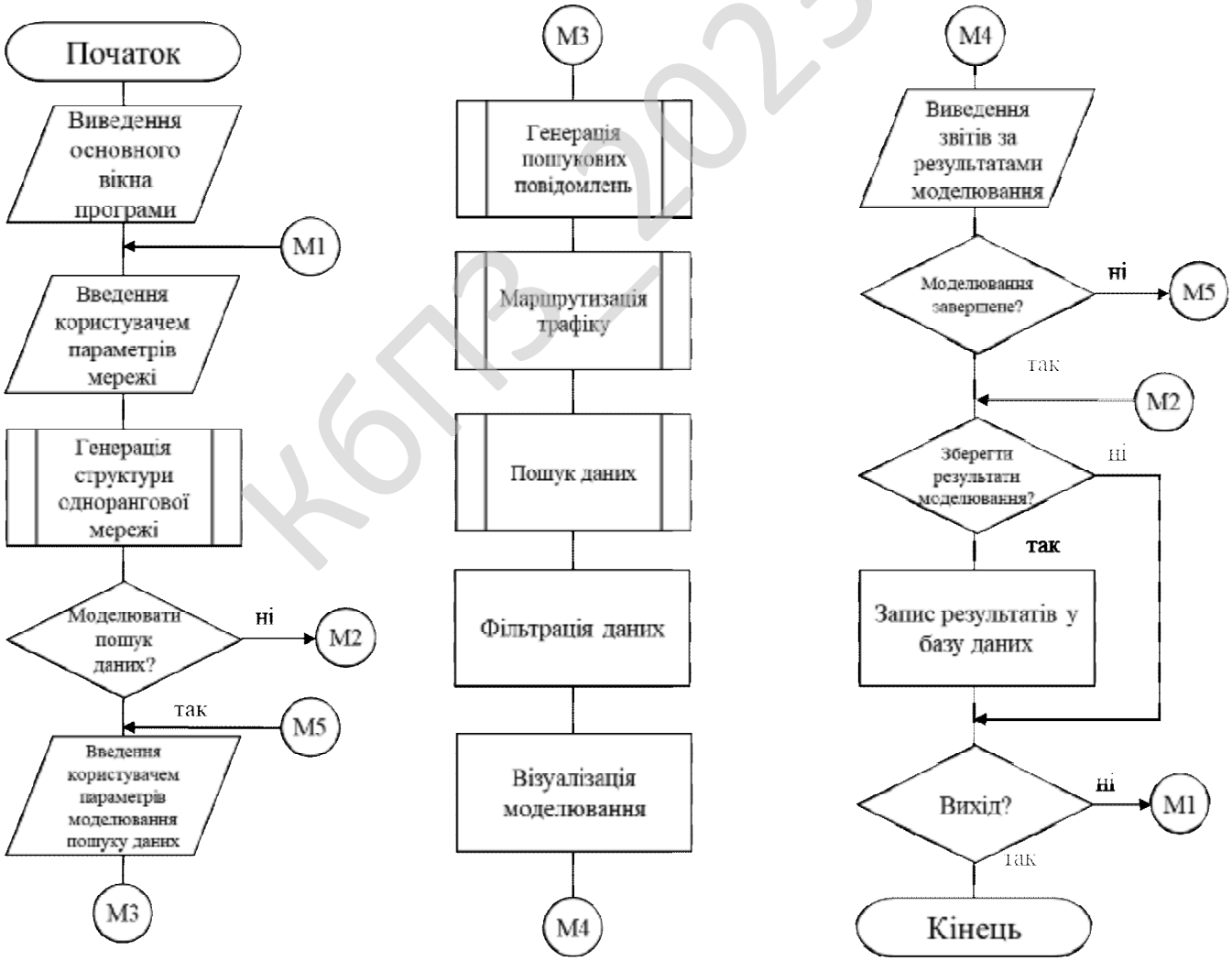

Рисунок 4.1 – Блок-схема основної програми

Загальний опис процесу пошуку даних у децентралізованих однорангових мережах:

1. Реєстрація ресурсу. Спершу ресурс (наприклад, файл, веб-сторінка або будь-який інший контент) повинен бути доданий до мережі. Це може включати в себе реєстрацію ресурсу та створення хеш-значення (унікального ідентифікатора) цього ресурсу.

2. Розподілення даних. Коли ресурс зареєстрований, він може бути розподілений по різних вузлах мережі. Це означає, що кожен вузол може мати копію частини або всього ресурсу.

3. Пошук запитів. Користувач, який хоче знайти певний ресурс, створює запит і надсилає його до мережі. Запит може бути побудований на основі хеш-значення ресурсу, ключових слів, метаданих тощо.

4. Передача запиту. Запит пересилається вузлами мережі. Кожен вузол перевіряє свої дані та пересилає запит далі на інші вузли.

5. Пошук ресурсу. Коли запит доходить до вузла, який має відповідний ресурс, він відправляє цей ресурс назад до вузла-ініціатора запиту.

6. Передача ресурсу. Ресурс передається до вузла, який ініціював запит. Вузол-ініціатор може також отримувати ресурс від кількох джерел одночасно, що покращує швидкодію і надійність.

7. Кешування даних. В децентралізованих мережах часто використовується кешування, коли популярні ресурси зберігаються тимчасово на вузлах для підвищення швидкості доступу та зменшення навантаження на мережу.

8. Реплікація та актуалізація. Дані можуть бути репліковані на кількох вузлах для забезпечення надійності та доступності. Крім того, ресурси можуть актуалізуватися або змінюватися з часом, і вони повинні оновлюватися у всіх копіях.

9. Завершення запиту. Після успішного пошуку та отримання ресурсу вузол-ініціатор запиту завершує запит та може розпочати використовувати

отриманий ресурс.

10. Аутентифікація та безпека. Децентралізовані мережі також можуть включати механізми безпеки та аутентифікації, щоб переконатися, що доступ до ресурсів обмежений лише для вповноважених користувачів.

Для маршрутизації даних в одноранговій мережі було використано алгоритм Kademlia). Це алгоритм для пошуку та зберігання даних в децентралізованих мережах, таких як мережі DHT (розподілені хеш-таблиці).

На рис. 4.2 наведено блок-схему алгоритму децентралізованого пошуку даних.

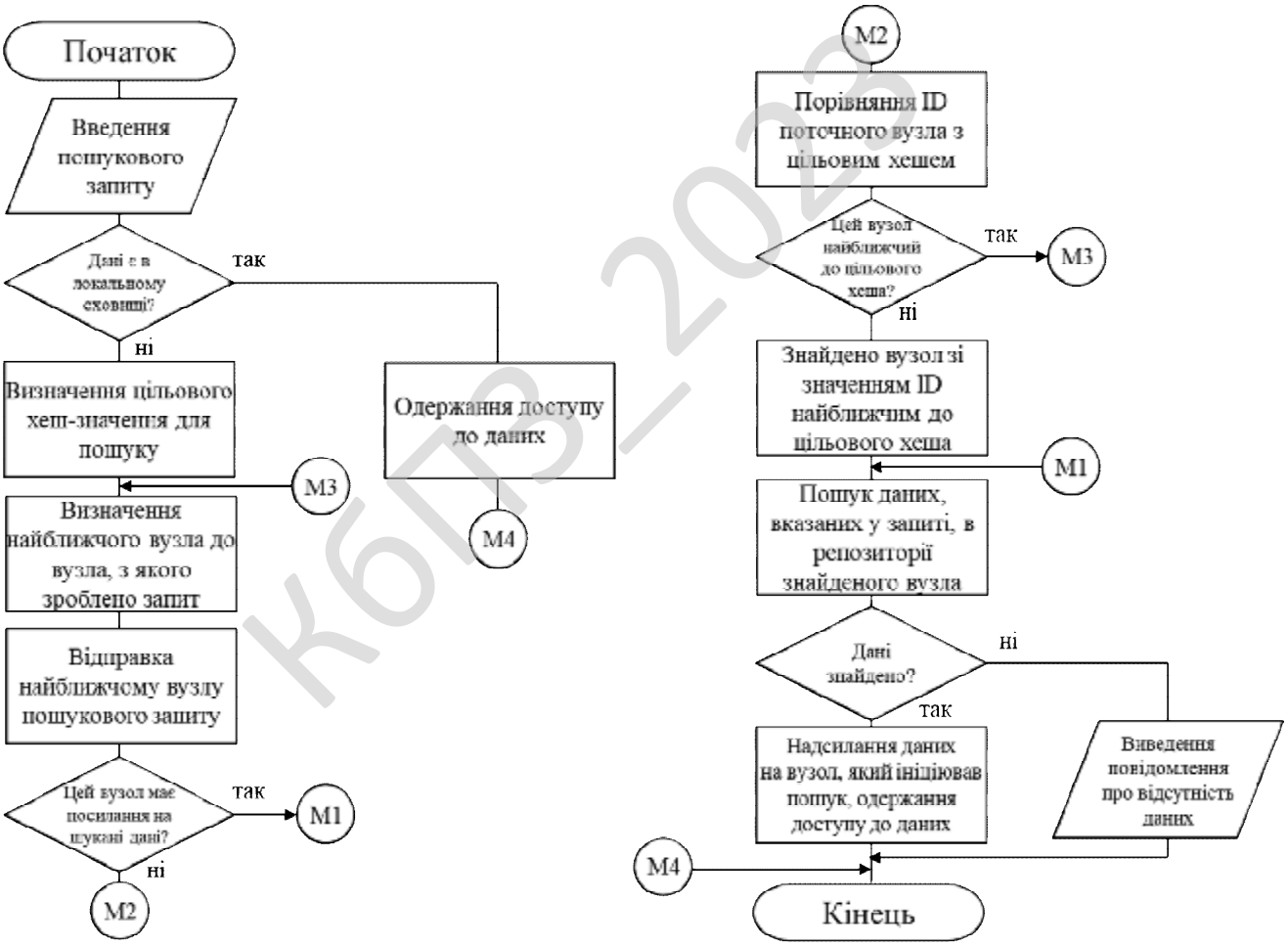

Рисунок 4.2 – Блок-схема алгоритму децентралізованого пошуку даних

Імітаційне моделювання пошуку файлу у децентралізованій одноранговій мережі реалізоване наступним чином:

```
import tkinter as tk
from tkinter import ttk
from tkinter import Canvas, Text
import random
import math

class Node:
    def __init__(self, id, x, y, hash_value):
        self.id = id
        self.x = x
        self.y = y
        self.hash = hash_value
        self.neighbors = []
        self.dht_table = {}  # DHT таблиця для кожного вузла

    def add_to_dht(self, key, value):
        self.dht_table[key] = value

    def get_dht_table(self):
        return self.dht_table

class Network:
    def __init__(self):
        self.nodes = []

    def add_node(self, node):
        self.nodes.append(node)

    def add_edge(self, node1, node2):
        node1.neighbors.append(node2)
        node2.neighbors.append(node1)

    def find_node_by_id(self, id):
        for node in self.nodes:
            if node.id == id:
                return node
        return None

class DHTTable:
    def __init__(self, root):
        self.root = root
        self.root.title("Table")
        self.text = Text(self.root, wrap=tk.WORD)
        self.text.pack()

    def show_table(self, node):
        self.text.delete("1.0", tk.END)
        dht_table = node.get_dht_table()
        for key, value in dht_table.items():
            self.text.insert(tk.END, f"Key: {key}, Value: {value}\n")

def kademia_algorithm(network, start_node, target_hash, dht_table):
    visited = set()
    visited.add(start_node)  # Почнемо з першого вузла

    stack = [(start_node, [])]
```

```
    while stack:
        current_node, path = stack.pop()

        if current_node.hash == target_hash:
            return path + [current_node]

        # Визначте список сусідів в порядку збільшення їх хеш-значень
        sorted_neighbors = sorted(current_node.neighbors, key=lambda node:
node.hash)

        for neighbor in sorted_neighbors:
            if neighbor not in visited:
                visited.add(neighbor)
                stack.append((neighbor, path + [current_node]))
                # Оновлення таблиці DHT з відвіданими вузлами
                dht_table.add_entry(neighbor, f"Visited by Node
{current_node.id}")
                neighbor.add_to_dht(target_hash, f"Visited by Node
{current_node.id}")

    return None

class NetworkSimulationApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Network Simulation")
        self.canvas = Canvas(root, width=800, height=400)
        self.canvas.pack()
        self.network = Network()
        self.dht_table_window = None

        # Додавання 20 вузлів в мережу, розташованих рівномірно на колі з
випадковими номерами
        num_nodes = 20
        circle_center_x = 400
        circle_center_y = 200
        circle_radius = 150
        angles = [2 * math.pi * i / num_nodes for i in range(num_nodes)]
        random.shuffle(angles)

        def create_kbucket_links(self, node, k=3):
            # Визначаємо список сусідів в порядку збільшення їх хеш-значень
            sorted_neighbors = sorted(self.network.nodes, key=lambda neighbor:
neighbor.hash)

            # Знайдемо позицію поточного вузла у відсортованому списку
            index = sorted_neighbors.index(node)

            # Додамо зв'язки з k ближніми сусідами з обох сторін (крім себе)
            for i in range(1, k + 1):
                if index - i >= 0:
                    self.network.add_edge(node, sorted_neighbors[index - i])
                if index + i < len(sorted_neighbors):
                    self.network.add_edge(node, sorted_neighbors[index + i])

        # Додавання зв'язків між вузлами з врахуванням k-бакетів
        for i, angle in enumerate(angles):
            x = circle_center_x + circle_radius * math.cos(angle)
            y = circle_center_y + circle_radius * math.sin(angle)
            hash_value = f"hash{i + 1}"
            node = Node(i + 1, x, y, hash_value)
            self.network.add_node(node)
```

```
            create_kbucket_links(self, node)



        self.network.add_edge(self.network.nodes[-1], self.network.nodes[0])  #
Замкнути коло

        self.start_node_id_entry = ttk.Entry(root, width=5)
        self.start_node_id_entry.insert(0, "1")
        self.start_node_id_entry.pack()

        self.target_hash_entry = ttk.Entry(root, width=10)
        self.target_hash_entry.insert(0, "hash20")
        self.target_hash_entry.pack()

        self.start_button = ttk.Button(root, text="Start Simulation",
command=self.start_simulation)
        self.start_button.pack()

        self.clear_button = ttk.Button(root, text="Clear",
command=self.clear_lines)
        self.clear_button.pack()

        self.dht_label = ttk.Label(root, text="Table:")
        self.dht_label.pack()

        self.dht_listbox = tk.Listbox(root, width=40, height=10)
        self.dht_listbox.pack()

    def start_simulation(self):
        start_node_id = int(self.start_node_id_entry.get())
        target_hash = self.target_hash_entry.get()

        start_node = self.network.find_node_by_id(start_node_id)

        if start_node:
            result = kademia_algorithm(self.network, start_node, target_hash,
self)
            if result:
                self.visualize_path(result)
            else:
                print("Шлях не знайдено.")
        else:
            print("Початковий вузол не знайдено.")

    def visualize_path(self, path):
        self.canvas.delete("all")  # Очистити всі малюнки на канвасі

        # Малювання вузлів мережі
        for node in self.network.nodes:
            fill_color = "green" if node in path else "blue"
            if node.id == path[0].id:
                fill_color = "red"  # Перший вузол червоний
            elif node.id == path[-1].id:
                fill_color = "green"  # Останній вузол зелений
            self.canvas.create_oval(node.x - 10, node.y - 10, node.x + 10, node.y
+ 10, fill=fill_color)
            self.canvas.create_text(node.x, node.y - 20, text=str(node.id))

        # # Малювання шляху
        for i in range(len(path) - 1):
            node1 = path[i]
            node2 = path[i + 1]
```

```
            self.canvas.create_line(node1.x, node1.y, node2.x, node2.y,
fill="green", width=2, arrow=tk.LAST, arrowshape=(20, 25, 5))

      # Малювання вузлів мережі
        for node in self.network.nodes:
            fill_color = "yellow" if node in path else "blue"
            if node.id == path[0].id:
                fill_color = "red"  # Перший вузол червоний
            elif node.id == path[-1].id:
                fill_color = "green"  # Останній вузол зелений
            self.canvas.create_oval(node.x - 10, node.y - 10, node.x + 10, node.y
+ 10, fill=fill_color)
            self.canvas.create_text(node.x, node.y - 20, text=str(node.id))


    def clear_lines(self):
        if self.dht_table_window:
            self.dht_table_window.clear_table()

    def add_entry(self, node, entry):
        self.dht_listbox.insert(tk.END, f"Node {node.id}: {entry}")

    def show_dht_table(self, node):
        if self.dht_table_window:
            self.dht_table_window.destroy()
        self.dht_table_window = DHTTable(tk.Toplevel())
        dht_table = node.get_dht_table()
        self.dht_table_window.show_table(node)

if __name__ == "__main__":
    root = tk.Tk()
    app = NetworkSimulationApp(root)
    root.mainloop()
```

Для реалізації вузла мережі використано наступний код:

```
from operator import itemgetter
import heapq


class NetworkNode:
    """
    Простий об'єкт, що інкапсулює поняття Вузла (мінімально ID, але можливо також
IP і порт, якщо це представляє собою вузол в мережі).
    Зазвичай цей клас не слід створювати безпосередньо, оскільки він є
низькорівневою конструкцією, в основному використовуваною маршрутизатором.
    """
    def __init__(self, node_id, ip=None, port=None):
        """
        Створити екземпляр Вузла.

        Args:
            node_id (int): Значення від 0 до 2^160
            ip (string): Необов'язкова IP-адреса, де знаходиться цей Вузол
            port (int): Необов'язковий порт для цього Вузла (встановлюється, коли
встановлено IP)
        """
        self.id = node_id  # pylint: disable=invalid-name
        self.ip = ip  # pylint: disable=invalid-name
```

```
        self.port = port
        self.long_id = int(node_id.hex(), 16)

    def same_home_as(self, node):
        return self.ip == node.ip and self.port == node.port

    def distance_to(self, node):
        """
        Отримати відстань між цим вузлом та іншим.
        """
        return self.long_id ^ node.long_id

    def __iter__(self):
        """
        Дозволяє використовувати Вузол як кортеж - наприклад, tuple(node) працює.
        """
        return iter([self.id, self.ip, self.port])

    def __repr__(self):
        return repr([self.long_id, self.ip, self.port])

    def __str__(self):
        return "%s:%s" % (self.ip, str(self.port))


class NetworkNodeHeap:
    """
    Купа вузлів, впорядкованих за відстанню до заданого вузла.
    """
    def __init__(self, node, maxsize):
        """
        Конструктор.

        @param вузол: Вузол для вимірювання всіх відстаней від нього.
        @param максимальний_розмір: Максимальний розмір, до якого може зрости ця
купа.
        """
        self.node = node
        self.heap = []
        self.contacted = set()
        self.maxsize = maxsize

    def remove(self, peers):
        """
        Видалити список ідентифікаторів однолітків з цієї купи.
        Хоча ця купа має постійний видимий розмір (на основі ітератора),
        її фактичний розмір може бути значно більшим, ніж те, що відображено.
        Тому видалення вузлів може не змінити видимий розмір, оскільки раніше
        додані вузли раптово стають видимими.
        """
        peers = set(peers)
        if not peers:
            return
        nheap = []
        for distance, вузол in self.heap:
            if в node.id not in peers:
                heapq.heappush(nheap, (distance, node))
        self.heap = nheap

    def get_node(self, node_id):
        for _, node in self.heap:
if node.id == node_id:
                return node
```

```
        return None

    def have_contacted_all(self):
        return len(self.get_uncontacted()) == 0

    def get_ids(self):
        return [n.id for n in self]

    def mark_contacted(self, вузол):
        self.contacted.add(вузол.id)

    def popleft(self):
        return heapq.heappop(self.heap)[1] if self else None

    def push(self, nodes):
        """
        Додати вузли в купу.

        @param вузли: Це може бути один елемент або список.
        """
        if not isinstance(вузли, list):
               nodes = [nodes]

        for вузол in вузли:
            if вузол not in self:
                відстань = self.node.distance_to(вузол)
                heapq.heappush(self.heap, (distance, node))

    def __len__(self):
        return min(len(self.heap), self.maxsize)

    def __iter__(self):
        вузли = heapq.nsmallest(self.maxsize, self.heap)
        return iter(map(itemgetter(1), nodes))

    def __contains__(self, node):
        for _, other in self.heap:
            if node.id == other.id:
                return True
        return False

    def get_uncontacted(self):
        return [n for n in self if n.id not in self.contacted]
```


Для реалізації методу маршрутизації у мережі використано наступний код:

```
import heapq
import time
import operator
import asyncio

from itertools import chain
from collections import OrderedDict
from kademlia.utils import shared_prefix, bytes_to_bit_string

class KBucket:
    def __init__(self, rangeLower, rangeUpper, ksize,
replacementNetworkNodeFactor=5):
        self.range = (rangeLower, rangeUpper)
```

```
        self.nodes = OrderedDict()
        self.replacement_nodes = OrderedDict()
        self.touch_last_updated()
        self.ksize = ksize
        self.max_replacement_nodes = self.ksize * replacementNetworkNodeFactor

    def touch_last_updated(self):
        self.last_updated = time.monotonic()

    def get_nodes(self):
        return list(self.nodes.values())

    def split(self):
        midpoint = (self.range[0] + self.range[1]) // 2
        one = KBucket(self.range[0], midpoint, self.ksize)
        two = KBucket(midpoint + 1, self.range[1], self.ksize)
        nodes = chain(self.nodes.values(), self.replacement_nodes.values())
        for node in nodes:
            bucket = one if node.long_id <= midpoint else two
            bucket.add_node(node)

        return (one, two)

    def remove_node(self, node):
        if node.id in self.replacement_nodes:
            del self.replacement_nodes[node.id]

        if node.id in self.nodes:
            del self.nodes[node.id]

            if self.replacement_nodes:
                newnode_id, newnode = self.replacement_nodes.popitem()
                self.nodes[newnode_id] = newnode

    def has_in_range(self, node):
        return self.range[0] <= node.long_id <= self.range[1]

    def is_new_node(self, node):
        return node.id not in self.nodes

    def add_node(self, node):
        """
        Додайте вузол C{NetworkNode} до C{KBucket}. Повертає True, якщо успішно,
        False, якщо бакет повний.

        Якщо бакет повний, відстежуйте вузол у списку заміщення,
        """
        if node.id in self.nodes:
            del self.nodes[node.id]
            self.nodes[node.id] = node
        elif len(self) < self.ksize:
            self.nodes[node.id] = node
        else:
            if node.id in self.replacement_nodes:
                del self.replacement_nodes[node.id]
            self.replacement_nodes[node.id] = node
            while len(self.replacement_nodes) > self.max_replacement_nodes:
                self.replacement_nodes.popitem(last=False)
            return False
        return True

    def depth(self):
        vals = self.nodes.values()
```

```
        sprefix = shared_prefix([bytes_to_bit_string(n.id) for n in vals])
        return len(sprefix)

    def head(self):
        return list(self.nodes.values())[0]

    def __getitem__(self, node_id):
        return self.nodes.get(node_id, None)

    def __len__(self):
        return len(self.nodes)

class TableTraverser:
    def __init__(self, table, startNetworkNode):
        index = table.get_bucket_for(startNetworkNode)
        table.buckets[index].touch_last_updated()
        self.current_nodes = table.buckets[index].get_nodes()
        self.left_buckets = table.buckets[:index]
        self.right_buckets = table.buckets[(index + 1):]
        self.left = True

    def __iter__(self):
        return self

    def __next__(self):
        """
        Вибірка елемента з лівого піддерева, потім правого, потім лівого, і т. д.
        """
        if self.current_nodes:
            return self.current_nodes.pop()

        if self.left and self.left_buckets:
            self.current_nodes = self.left_buckets.pop().get_nodes()
            self.left = False
            return next(self)

        if self.right_buckets:
            self.current_nodes = self.right_buckets.pop(0).get_nodes()
            self.left = True
            return next(self)

        raise StopIteration

class RoutingTable:
    def __init__(self, protocol, ksize, node):
        """
        @param node: Вузол, який представляє цей сервер. Він не буде
        доданий до таблиці маршрутизації, але пізніше буде потрібний
        для визначення, які бакети розщеплювати або ні.
        """
        self.node = node
        self.protocol = protocol
        self.ksize = ksize
        self.flush()

    def flush(self):
        self.buckets = [KBucket(0, 2 ** 160, self.ksize)]

    def split_bucket(self, index):
        one, two = self.buckets[index].split()
        self.buckets[index] = one
        self.buckets.insert(index + 1, two)
```

```
    def lonely_buckets(self):
        """
        Отримайте всі бакети, які не оновлювалися протягом понад години.
        """
        hrago = time.monotonic() - 3600
        return [b for b in self.buckets if b.last_updated < hrago]

    def remove_contact(self, node):
        index = self.get_bucket_for(node)
        self.buckets[index].remove_node(node)

    def is_new_node(self, node):
        index = self.get_bucket_for(node)
        return self.buckets[index].is_new_node(node)

    def add_contact(self, node):
        index = self.get_bucket_for(node)
        bucket = self.buckets[index]


        # це буде успішним, якщо бакет повний
        if bucket.add_node(node):
            return


        # розділити бакет,
        # якщо в бакеті знаходиться вузол
        # в його діапазоні або якщо глибина не конгруентна 0 mod 5
        if bucket.has_in_range(self.node) or bucket.depth() % 5 != 0:
            self.split_bucket(index)
            self.add_contact(node)
        else:
            asyncio.ensure_future(self.protocol.call_ping(bucket.head()))


    def get_bucket_for(self, node):
        """
        Отримайте індекс бакету, в який попаде даний вузол.
        """
        for index, bucket in enumerate(self.buckets):
            if node.long_id < bucket.range[1]:
                return index
        return None


    def find_neighbors(self, node, k=None, exclude=None):
        k = k or self.ksize
        nodes = []
        for neighbor in TableTraverser(self, node):
            notexcluded = exclude is None or not neighbor.same_home_as(exclude)
            if neighbor.id != node.id and notexcluded:
                heapq.heappush(nodes, (node.distance_to(neighbor), neighbor))
            if len(nodes) == k:
                break

        return list(map(operator.itemgetter(1), heapq.nsmallest(k, nodes)))
```

Для пошуку даних використано наступний код:

```
from collections import Counter
import logging

from kademlia.node import NetworkNode, NetworkNodeHeap
from kademlia.utils import gather_dict

log = logging.getLogger(__name__)  # pylint: disable=invalid-name

# pylint: disable=too-few-public-methods
class SpiderCrawl:
    """
    Пошук в мережі і пошук даних за 160-бітними ключами.
    """
    def __init__(self, protocol, node, peers, ksize, alpha):
        """
        Створіть нового C{SpiderCrawl}er.

        Args:
            protocol: Екземпляр :class:`~kademlia.protocol.KademliaProtocol`.
            node: :class:`~kademlia.node.NetworkNode`, що представляє ключ, який
ми
                  шукаємо.
            peers: Список екземплярів :class:`~kademlia.node.NetworkNode`, які
                   надають вхідну точку до мережі.
            ksize: Значення k згідно з паперу.
            alpha: Значення alpha згідно з паперу.
        """
        self.protocol = protocol
        self.ksize = ksize
        self.alpha = alpha
        self.node = node
        self.nearest = NetworkNodeHeap(self.node, self.ksize)
        self.last_ids_crawled = []
        log.info("створення пошуку з вузлами-першоджерелами: %s", peers)
        self.nearest.push(peers)

    async def _find(self, rpcmethod):
        """
        Отримати або значення або список вузлів.

        Args:
            rpcmethod: Протокол callfindValue або call_find_node.

        Процес:
          1. викликає find_* для поточних ALPHA найближчих вузлів,
             які ще не були запитані, додаючи результати до поточного списку
             найближчих списків k вузлів.
          2. поточний список найближчих потрібно відстежувати, хто був
             запитаний вже, сортувати за найближчими, зберігати KSIZE.
          3. якщо список такий самий, як і минулого разу, наступний виклик
             повинен бути до всіх, хто ще не був запитаний.
          4. повторюйте, якщо поточний список найближчих був запитаний всім,
             тоді ви завершили.
        """
        log.info("пошук в мережі з найближчими: %s", str(tuple(self.nearest)))
        count = self.alpha
        if self.nearest.get_ids() == self.last_ids_crawled:
            count = len(self.nearest)
```

```
        self.last_ids_crawled = self.nearest.get_ids()

        dicts = {}
        for peer in self.nearest.get_uncontacted()[:count]:
            dicts[peer.id] = rpcmethod(peer, self.node)
            self.nearest.mark_contacted(peer)
        found = await gather_dict(dicts)
        return await self._nodes_found(found)

    async def _nodes_found(self, responses):
        raise NotImplementedError


class ValueSpiderCrawl(SpiderCrawl):
    def __init__(self, protocol, node, peers, ksize, alpha):
        SpiderCrawl.__init__(self, protocol, node, peers, ksize, alpha)
        # Відстежувати єдиний найближчий вузол без значення - щоб встановити ключ
там, якщо знайдено
        self.nearest_without_value = NetworkNodeHeap(self.node, 1)

    async def find(self):
        """
        Знайти або найближчі вузли, або запитане значення.
        """
        return await self._find(self.protocol.call_find_value)

    async def _nodes_found(self, responses):
        """
        Обробити результат ітерації в _find.
        """
        toremove = []
        found_values = []
        for peerid, response in responses.items():
            response = RPCFindResponse(response)
            if not response.happened():
                toremove.append(peerid)
            elif response.has_value():
                found_values.append(response.get_value())
            else:
                peer = self.nearest.get_node(peerid)
                self.nearest_without_value.push(peer)
                self.nearest.push(response.get_node_list())
        self.nearest.remove(toremove)

        if found_values:
            return await self._handle_found_values(found_values)
        if self.nearest.have_contacted_all():
            # не знайдено!
            return None
        return await self.find()

    async def _handle_found_values(self, values):
        """
        Ми отримали деякі значення. Але переконаймося,
        що вони всі однакові або трохи різні.
        """
        value_counts = Counter(values)
        if len(value_counts) != 1:
            log.warning("Отримано кілька значень для ключа %i: %s",
                        self.node.long_id, str(values))
        value = value_counts.most_common(1)[0][0]

        peer = self.nearest_without_value.popleft()
```

```
        if peer:
            await self.protocol.call_store(peer, self.node.id, value)
        return value


class NetworkNodeSpiderCrawl(SpiderCrawl):
    async def find(self):
        """
        Знайдіть найближчі вузли.
        """
        return await self._find(self.protocol.call_find_node)

    async def _nodes_found(self, responses):
        """
        Обробити результат ітерації в _find.
        """
        toremove = []
        for peerid, response in responses.items():
            response = RPCFindResponse(response)
            if not response.happened():
                toremove.append(peerid)
            else:
                self.nearest.push(response.get_node_list())
        self.nearest.remove(toremove)

        if self.nearest.have_contacted_all():
            return list(self.nearest)
        return await self.find()


class RPCFindResponse:
    def __init__(self, response):
        """
        Обгортка для результату пошуку RPC.

        Args:
            response: Це буде кортеж (<відповідь, отримана>, <значення>)
                      де <значення> буде списком кортежів, якщо не знайдено
                      або словником {'value': v}, де v - бажане значення
        """
        self.response = response

    def happened(self):
        """
        Чи відповів інший хост насправді?
        """
        return self.response[0]

    def has_value(self):
        return isinstance(self.response[1], dict)

    def get_value(self):
        return self.response[1]['value']

    def get_node_list(self):
        """
        Отримати список вузлів у відповіді. Якщо значення відсутнє, його
        треба встановити.
        """
        nodelist = self.response[1] or []
        return [NetworkNode(*nodeple) for nodeple in nodelist]
```

4.2 Захист розробленого програмного забезпечення

Для захисту програмного забезпечення від несанкціонованого використання та поширення пропонується використовувати реєстрацію користувача та введення ключів активації. Захист ключів активації буде відбуватися за допомогою криптографічного алгоритму ECDSA.

Для реалізації авторизації користувача та введення ключа активації пропонується використовувати наступний код:

```
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox

class RegistrationApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Реєстрація та активація")

        self.username_label = ttk.Label(root, text="Ім’я користувача:")
        self.username_label.pack()
        self.username_entry = ttk.Entry(root, width=30)
        self.username_entry.pack()

        self.email_label = ttk.Label(root, text="Електронна пошта:")
        self.email_label.pack()
        self.email_entry = ttk.Entry(root, width=30)
        self.email_entry.pack()

        self.email_label = ttk.Label(root, text="Ключ активації:")
        self.email_label.pack()
        self.email_entry = ttk.Entry(root, width=30)
        self.email_entry.pack()

        self.activate_button = ttk.Button(root, text="Активувати",
command=self.activate_account)
        self.activate_button.pack()

    def activate_account(self):
        username = self.username_entry.get()
        email = self.email_entry.get()

        # Перевірка чи введені дані не пусті
        if not username or not email:
            messagebox.showerror("Помилка", "Будь ласка, заповніть всі
поля.")
            return


        ... # збереження даних в базі даних та надсилання повідомлення
користувачу на вказану пошту

        # Виведення повідомлення про успішну активацію
        messagebox.showinfo("Успішно", "Обліковий запис успішно створено і
```

```
активовано.")
                self.root.destroy()

        if __name__ == "__main__":
            root = tk.Tk()
            app = RegistrationApp(root)
            root.mainloop()
```

Для реалізації захисту ключів активації пропонується робити наступне:

1. Генерація ключів ECDSA:

```
import ecdsa

# Генерація приватного ключа
private_key = ecdsa.SigningKey.generate(curve=ecdsa.SECP256k1)

# Отримання відповідного публічного ключа
public_key = private_key.get_verifying_key()

# Збереження ключів
private_key_hex = private_key.to_string().hex()
public_key_hex = public_key.to_string().hex()
```

2. Створення ключа активації та його підпис:

```
import hashlib

# Інформація про користувача або пристрій (якщо це пристрій, то унікальний
ідентифікатор пристрою)
user_info = "user123"

# Створення ключа активації на основі інформації
activation_key = hashlib.sha256(user_info.encode()).hexdigest()

# Підпис ключа активації приватним ключем
signature = private_key.sign(activation_key.encode())
```

3. Перевірка ключа активації:

```
# Ключ активації, який ввів користувач
user_activation_key =
"c1e1ff8e19b69f193c5b1737caaed798164631c76d272ab9e4d799f21c42ac0f"

# Перевірка підпису
try:
    public_key.verify(signature, user_activation_key.encode())
    print("Ключ активації вірний. Програма активована.")
except ecdsa.BadSignatureError:
    print("Помилка: ключ активації не вірний.")
```

4. Захист і збереження ключів та ліцензій. Важливо зберігати приватний ключ ECDSA та інші ліцензійні дані у безпечному місці, наприклад, у захищеному файлі або базі даних.

5. Періодична перевірка ліцензій та відкликання ключів. Ця функціональність може бути реалізована на серверній стороні програмного забезпечення та вимагає регулярної перевірки ліцензій та можливості відкликання ключів активації.

6. Захист інфраструктури та даних користувачів. Забезпечення безпеки серверів та інших складових інфраструктури, а також захисту особистих даних користувачів, включає в себе широкий спектр заходів і не обмежується одним конкретним кодом.

Реалізація повноцінної системи ліцензій та активації програмного забезпечення вимагає комплексного підходу та є важливою задачею.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Для використання розробленого программного забезпечення необхідно встановити наступні бібліотеки для мови Python:

– **tkinter** – бібліотека для створення графічних інтерфейсів користувача (GUI). Вона використовується для створення інтерактивних ігор, додатків з графічним інтерфейсом і багато іншого.

– **asyncio –** бібліотека для асинхронного програмування в Python. Вона використовується для створення асинхронних програм, які можуть виконувати паралельно різні задачі без блокування головного потоку виконання.

– **logging –** дозволяє логувати повідомлення в Python-додатках. Використовується для записування історії подій та відладки програм.

– **itemgetter –** використовується для отримання елементів зі структур даних, таких як списки та кортежі, за допомогою їх індексів або ключів.

– **heapq –** модуль для роботи з чергами з пріоритетами (heap). Він використовується для сортування та обробки даних за допомогою черги з пріоритетами.

– **pickle –** використовується для серіалізації та десеріалізації об’єктів Python. Дозволяє зберігати об’єкти у файлі або передавати їх по мережі. Використовується для зберігання стану програми, обміну даними між процесами або для передачі даних через мережу.

– **time –** надає функції для роботи з часом. Використовується для затримок, обчислення часу виконання та інших операцій, пов’язаних з часом.

– **abc** – містить інструменти для створення абстрактних класів та методів в Python. Використовується для задання структури об’єктів і забезпечення, щоб певні методи були реалізовані в підкласах.

– **hashlib –** використовується для обчислення хеш-суми об’єктів. Часто використовується для перевірки цілісності даних.

– **kademlia** – реалізує алгоритм Kademlia, який використовується для пошуку та зберігання інформації в децентралізованих мереж.

– **rpcudp** – надає можливість здійснення віддалених викликів процедур (RPC) за допомогою протоколу UDP. RPC використовується для виклику функцій або методів на віддаленому сервері через мережу.

– **itertools** – містить інструменти для роботи з ітераторами та ітераційними операціями в Python. Вона надає функції для створення та обробки ітераторів, такі як об'єднання послідовностей, фільтрація, карта, згортання тощо.

– **collections** – містить додаткові структури даних, яких немає в стандарті наприклад, включає структури, такі як OrderedDict (словник з фіксованим порядком ключів), Counter (лічильник елементів в послідовності) та інші.

Приклади роботи розробленого програмного забезпечення у режимі програмного імітаційного моделювання децентралізованої однорангової мережі та пошуку даних у ній наведені на рис. 5.1-5.3.

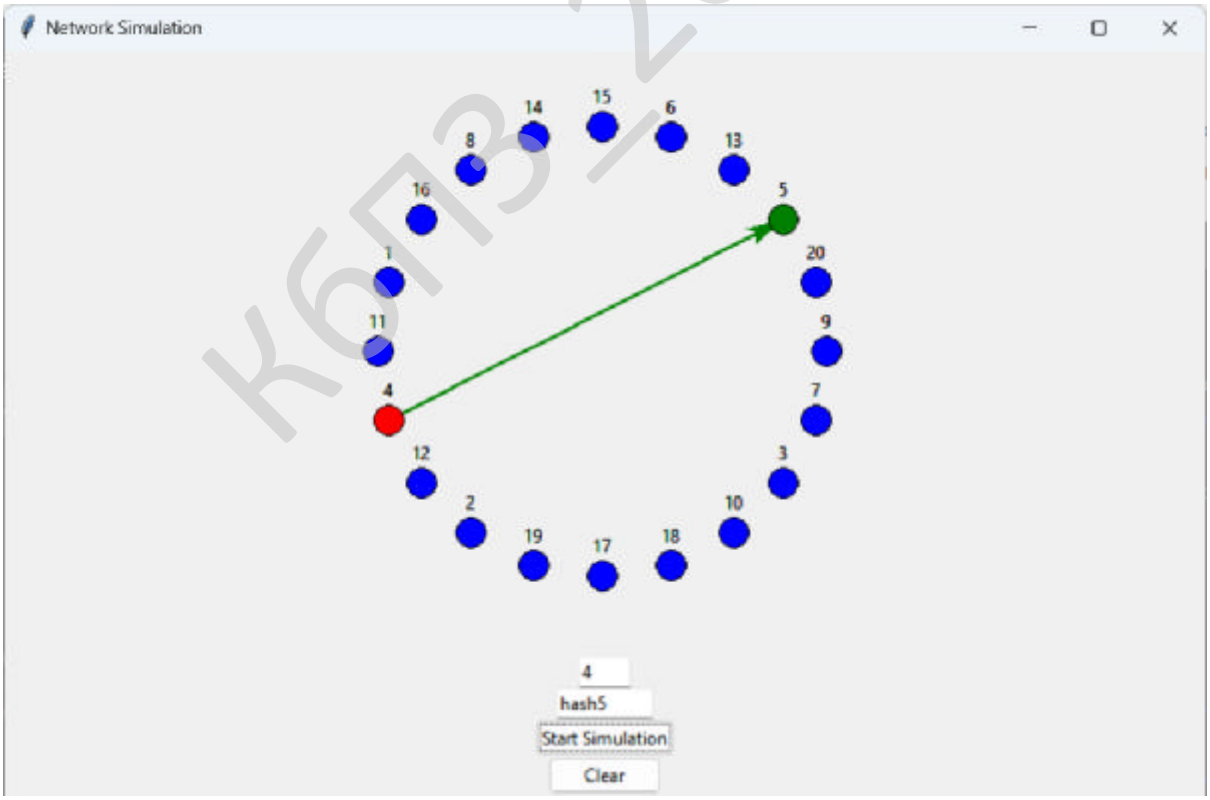

Рисунок 5.1 – Приклад роботи розробленого програмного забезпечення – вузол 4 здійснив запит даних з хеш-значенням 5

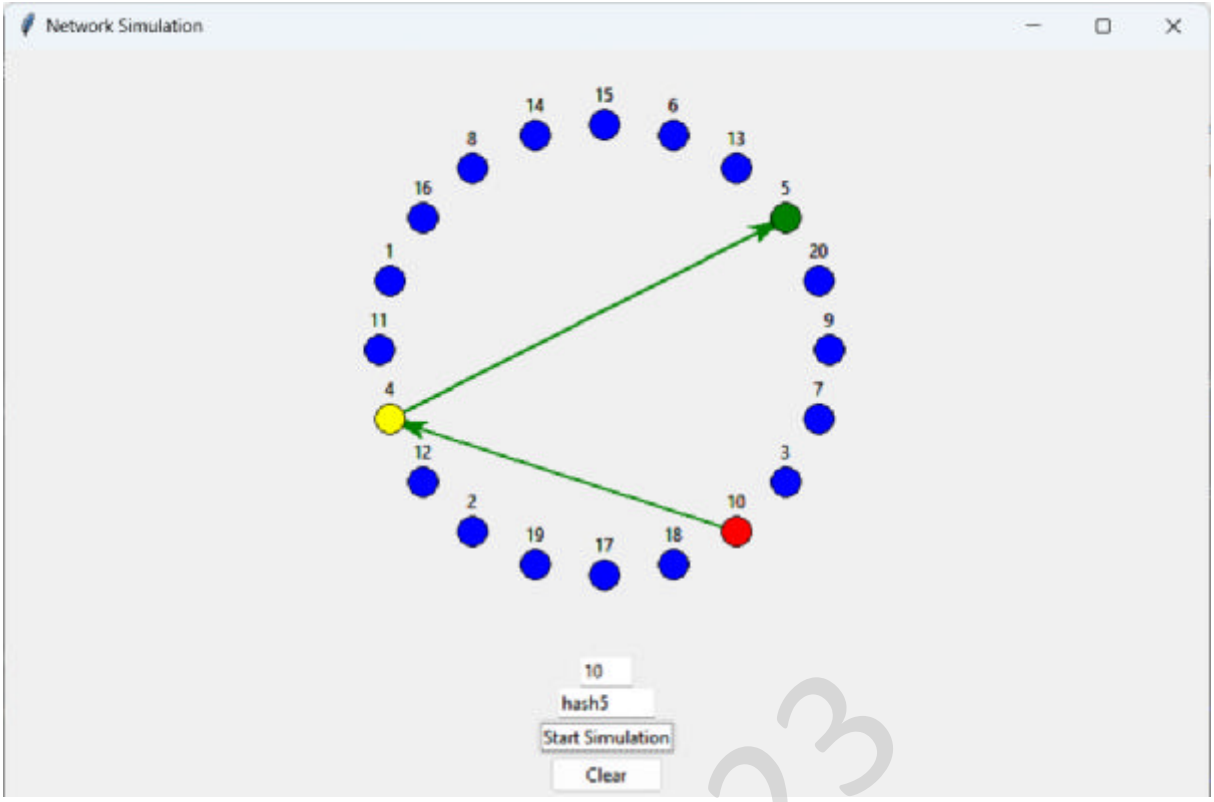

Рисунок 5.2 – Приклад роботи розробленого програмного забезпечення – вузол 10 здійснив запит даних з хеш-значенням 5

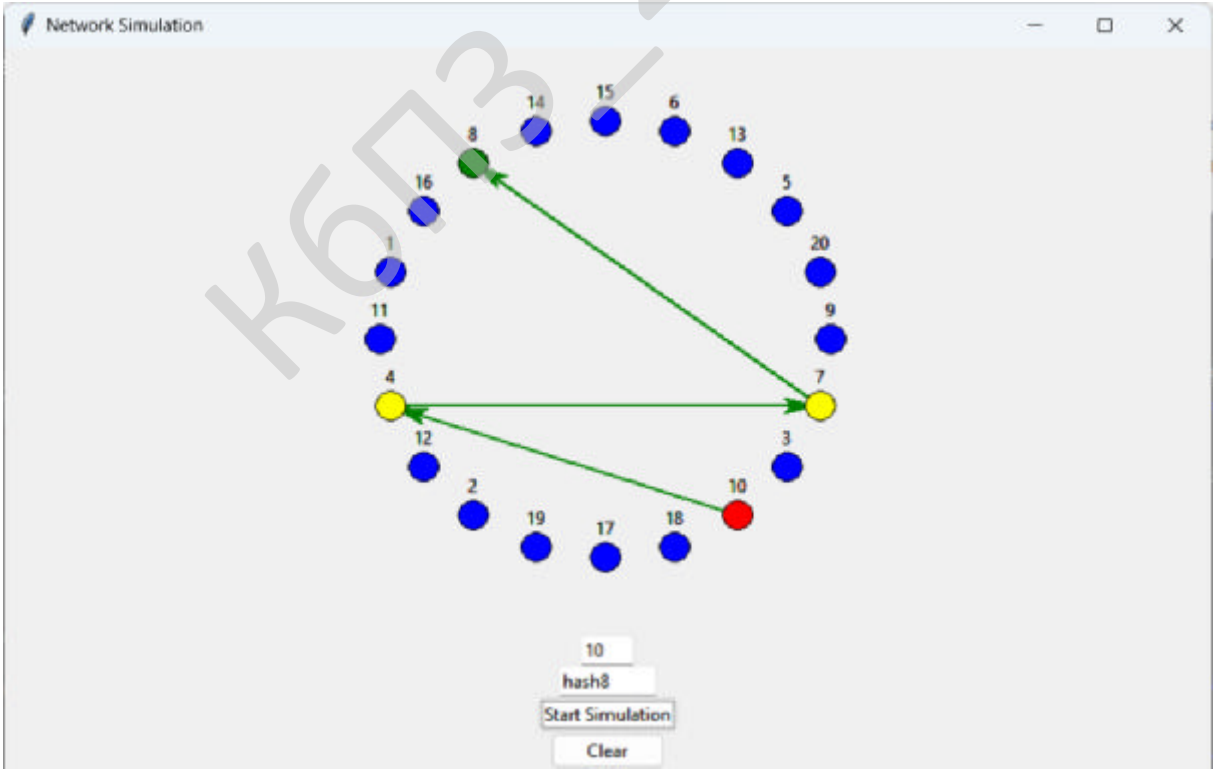

Рисунок 5.3 – Приклад роботи розробленого програмного забезпечення – вузол 10 здійснив запит даних з хеш-значенням 8

6 НАУКОВА НОВИЗНА

У магістерській роботі розроблено програмне забезпечення, яке призначено для реалізації системи пошуку даних у децентралізованих однорангових мережах.

Об'єктом дослідження є процес обробки даних у децентралізованих однорангових комп'ютерних мережах.

Предметом дослідження є методи пошуку даних у децентралізованих однорангових комп'ютерних мережах.

Методи дослідження базуються на теорії комп'ютерних мереж, теорії алгоритмів і структур даних, теорії графів, теорії статистики, теорії імітаційного моделювання та методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

1. Запропоновано метод імітаційного моделювання децентралізованих однорангових комп'ютерних мереж та процесів передачі даних у них.

2. Розроблено вітчизняний продукт для системи пошуку даних у децентралізованих однорангових комп'ютерних мережах, який має більш широкі можливості, на відміну від існуючих аналогів та може бути використаний у розподілених системах зберігання даних.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі пошуку даних у децентралізованих однорангових мережах, а також програмного імітаційного моделювання цього процесу.

7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

7.1 Техніко-економічне обґрунтування теми магістерської роботи

Після ознайомлення з підприємством та засобами розробки програмної продукції був розроблений план розробки програми. Був підрахований необхідний час для розробки та впровадження програми. Цей час склав 60 днів (три місяці).

В магістерській роботі було проведене дослідження та виконана програмна реалізація системи пошуку даних у децентралізованих однорангових мережах.

Розроблене програмне забезпечення має достатню надійність і задовольняє усім поставленим умовам, а саме:

а) невеликий розмір;

б) невеликі системні потреби;

в) незалежність від встановлених на комп'ютері баз даних;

г) зручність у користуванні та надійність.

Таблиця 7.1 – Початкові дані

Показники	Позначення	Характеристика або величина
1	2	3
1. Кількість розроблених програм період, шт.	N	1
2. Кількість екземплярів програм, шт.	Ne	280
3. Запланований термін розробки, днів	Fpq	60 (3 місяці)
4. Група задачі підсистеми управління (1-6)	–	1
5. Ступінь новизни задачі (А, Б, В, Г)	–	Б
6. Складність алгоритму (1, 2, 3)	–	2

Продовження таблиці 7.1

1	2	3
7. Кількість макетів вхідної інформації	–	3
8. Кількість форм вихідної інформації.	–	4
9. Мова програмування (1-6)	–	3
10. Попередній досвід (1-6)	–	3
11. Гнучкість проекту ПП (1-6)	–	3
12. Детальність проекту ПП (1-6)	–	2
13. Рівень спрацьованості колективу (1-6)	–	2
14. Ступінь вимірності процесів (1-6)	–	3
15. Необхідна надійність програмного забезпечення (1-6)	–	2
16. Розмір бази даних (порівняно з розміром програми) (1-6)	–	2
17. Складність кінцевого програмного продукту (1-6)	–	2
18. Необхідний рівень забезпечення повторного використання (1-6)	–	2
19. Документованість відповідно до планованого життєвого циклу (1-6)	–	2
20. Вимоги до швидкодії ПП (1-6)	–	2
21. Обмеження на розміри основного сховища даних (1-6)	–	2
22. Різноманітність використовуваних обчислювальних платформ (1-6)	–	2
23. Професійний рівень аналітиків (1-6)	–	2
24. Професійний рівень програмістів (1-6)	–	2
25. Постійність складу команди розробників (1-6)	–	2
26. Досвід розробки додатків (1-6)	–	2
27. Досвід роботи з обчислювальною платформою (1-6)	–	2

Продовження таблиці 7.1

1	2	3
28. Досвід роботи з мовою і інструментами середовища розробки (1-6)	–	2
29. Досвід роботи з програмними інструментами розробки (1-6)	–	3
30. Розробка ПЗ для декількох серверів одночасно (1-6)	–	2
31. Вимоги до дотримання встановленого графіка робіт (1-6)	–	2
32. Вартість ПЗ у розробника (НМА), грн.	–	28000
33. Норматив додаткової зарплати, % :	$Н_д$	10
34. Норматив відрахувань у соціальні фонди, %	Нс	22
35. Норматив загальногосподарських витрат, %	Нг	15
36. Норматив витрат на освоєння нових мов програмування, %	Нп	15
37. Рівень рентабельності програмної продукції, %	Ре	50
38. Ставка податку на додану вартість, %	Ндв	20

7.2 Розрахунок трудомісткості розробки програмної продукції

Значення трудомісткості розробки програмного забезпечення для стадій ТЗ, ЕК, ТП та ВП визначаємо по типовим нормам часу приведеним в додатках МВ. Стадія РП є найбільш тривалою і трудомісткою, що робить значний вплив на інші стадії проекту.

Визначимо трудомісткість розробки ПЗ для стадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

$$T_{ном} = A \, \text{Size}^B, \quad (7.1)$$

де: A – коефіцієнт Боема, A = 2,45;

Size – загальний об’єм відлагодженого програмного коду, тис. рядків;

B – показник ступеня, що визначається співвідношенням:

$$B = 1{,}01 + 0{,}001\sum W_i, \quad (7.2)$$

де: *Wi* – сумарне значення п’яти показників (МВ, додаток 2), що відображають особливості розробки проекту програмного продукту (ПП) і колективу розробників.

$$B = 1{,}01+0{,}001(2{,}43+3{,}64+3{,}38+3{,}95+2{,}73) = 1{,}027.$$

$$T_{ном} = 2{,}45 \cdot 2{,}7^{1{,}026} = 6{,}78 \text{ люд-міс.}$$

Визначаємо уточнені (з урахуванням приведених в МВ додатку 3 сімнадцяти додаткових коефіцієнтів) трудовитрати, люд-міс.:

$$T_{уточн} = T_{ном}\, ПВ_j, \quad (7.3)$$

де: $ПВ_j$ – добуток сімнадцяти додаткових коефіцієнтів, приведених в МВ додатку 3.

$$T_{уточн} = 6{,}78 \cdot (0{,}88 \cdot 0{,}93 \cdot 0{,}88 \cdot 0{,}91 \cdot 0{,}95 \cdot 1 \cdot 1 \cdot 0{,}87 \cdot 1{,}22 \cdot 1{,}16 \cdot 1{,}1 \cdot 1{,}1 \cdot$$
$$\cdot 1{,}12 \cdot 1{,}1 \cdot 1 \cdot 1{,}1 \cdot 1{,}1) = 9{,}37 \text{ люд-міс.}$$

Ці коефіцієнти дозволяють диференційовано оцінювати результати роботи програмістів, беручи до уваги швидкодію програми, використання різноманітних обчислювальних платформ і інструментів розробки, взаємодію декількох серверів, вимоги до об’ємів баз даних і ін.

Визначаємо підсумкові трудовитрати по стадії робочий проект, люд-дні:

$$T_{РП} = 0{,}3 C T_{уточн}^{0{,}33+0{,}2(B-1{,}01)} S, \quad (7.4)$$

де: *C* – визначений емпірично коефіцієнт, запропонований авторами методики, (МВ, додаток 4);

S – коефіцієнт стиснення (або подовження) графіка робіт %, що дозволяє коректувати терміни розробки ПЗ згідно встановленим вимогам. Вибираємо в межах (25…350)%.

$$T_{РП} = 0{,}3 \cdot 2{,}85 \cdot 9{,}37^{0{,}33+0{,}2(1{,}026-1{,}01)} \cdot 65 = 118 \text{ люд/день.}$$

Для зручності визначення загальної трудомісткості на розробку програмного забезпечення результати розрахунків по стадіям зводимо до таблиці 7.2.

Таблиця 7.2 – Визначення трудомісткості розробки програмного забезпечення

Стадії розробки	Трудомісткість за типовими нормами та розрахунками	
	Величина, люд/дні	Підстава
Технічне завдання	9	Д5
Ескізний проект	10	Д6
Технічний проект	9	Д7
Робочий проект	118	Ф 7.1-7.4
Впровадження	13	Д13
Всього	159	–

7.3 Визначення чисельності виконавців і планового фонду зарплати

Чисельність ставок інженерів-програмістів для розробки програмного забезпечення визначається за формулою:

$$Ч = \frac{T_{пз} N}{F_{рq} - H_{ев}}, \quad (7.5)$$

де: $F_{рq}$ – плановий фонд робочого часу одного спеціаліста, днів;

$T_{пз}$ – трудомісткість розробки програмного забезпечення люд-дні.

$$Ч = \frac{159 \cdot 1}{60 - 5} = 2{,}9 \text{ ставки.}$$

Чисельність інженерів-електронщиків для проведення технічного обслуговування та ремонту комп'ютерних мереж визначається в залежності від наявності технічних засобів і норм витрат часу на виконання профілактичних робіт на протязі року.

Визначаємо затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за період розробки. Результати розрахунку зводимо до таблиці 7.3.

Таблиця 7.3 – Затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за розрахунковий період

Найменування обладнання	Профілактичне обслуговування			
	Кількість хв. на один. обл.	Кількість облад-нання	Затрати часу в хв.	Затрати часу в год.
Системний блок ПК	90	7	630	10,5
Монітор	60	7	420	7
Клавіатура	30	7	210	3,5
Маніпулятор «мишка»	30	7	210	3,5
Принтер матричний	60	0	0	0,0
Принтер лазерний	120	2	240	4
Принтер струминний	60	1	60	1
Сканер	20	1	20	0,33
Концентратор–маршрутизатор	30	2	60	1
Кабельні господарства ЛВС на 1 м. п.	2,5	300	750	12,5
Копіювальний апарат	140	1	140	2,33
Усього за рік:			$З_ч$	45,66

Час на профілактику обладнання в загальному балансі робочого часу інженерів-електронщиків неповинен складати більше 10%.

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

$$Ф_{др}^{с} = \frac{З_ч \cdot n_{міс}}{1{,}2}, \qquad (7.6)$$

$$\Phi_{др}^{с} = \frac{45,66 \cdot 3}{1,2} = 114,15 \text{ год.}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

$$Ч_{ел} = \frac{\Phi_{др}^{с}}{F_{др} \cdot T_{зм}}, \quad (7.7)$$

$$Ч_{ел} = 114,15/(60 \cdot 8) = 0,2 \text{ ставки.}$$

Для забезпечення нормального технічного обслуговування засобів ТО та мереж, необхідно прийняти найбільше ціле значення розрахункової чисельності інженерів-електронщиків.

Таблиця 7.4 – Розрахунок чисельності штатного персоналу сектору системного та адміністративного обслуговування засобів ОТ та комп'ютерних мереж

Посада	Вид роботи	Час	К-ть штатних одиниць
Адміністратор загальної мережі, аналітик	Адміністрування локальної мережі, поштового та серверу DNS (ОС FreeBSD), маршрутизатора Cisco, доменного контролеру Windows Server 2022, серверу доступу ADSL (ОС Linux), налаштування ADSL, VPN, PPPoE, Frame Relay, Wi-Fi	2	0,5
	Налаштування і конфігурування базової станції безпровідного зв'язку (CMTS)	0,5	
	Розробка та впровадження проектів з організації зв'язку між віддаленими об'єктами, ЛОМ	0,5	
	Забезпечення цілодобової роботи зв'язку клієнтів до мережі Інтернет	1	
Всього		4	

Продовження таблиці 7.4

Посада	Вид роботи	Час	К-ть штатних одиниць
Продакт-менеджер	Презентації нової продукції, пошук каналів збуту	1	0,25
	Підтримка постійних клієнтів	0,5	
	Оформлення договорів, ведення тендерів	0,25	
	Контроль взаєморозрахунків з постачальниками	0,25	
Всього		2	
Дизайнер WEB	Розробка концепції оформлення та інтерфейсу сайту, оптимізація дизайну існуючих, проектує їх структуру та навігацію	1	0,25
	Створення графічних і стилістичних елементів сайту	0,5	
	Розміщення графіки і контенту на Інтернет сторінках	0,5	
Всього		2	
Інженер верстальник	Розробка та верстка макетів рекламної продукції та технічної документації	1	0,25
	Верстка друкованих видань	0,5	
	Додрукова підготовка макетів	0,25	
	Розміщення графіки і контенту на Інтернет сторінках	0,25	
Всього		2	

Чисельність інженерів-системотехніків, адміністраторів мережі, дизайнерів WEB вузлів, системних програмістів (аналітиків), бухгалтерів-економістів визначається за потребою в залежності від функціональних

обов’язків. Після визначення чисельності персоналу складається штатний розклад. Складемо штатний розклад виконавців (табл. 7.5).

Таблиця 7.5 – Штатний розклад виконавців

Посада	Кількість ставок	Середньомісячний оклад, грн.	Всього за період розробки, грн.
Керівник (ІТ-менеджер)	1	11979	35937
Продакт-менеджер	0,25	10000	7500
Інженер-програміст	2,9	10500	91350
Інженер-електронщик	0,2	10000	6000
Інженер-системотехнік	0,25	10000	7500
Адміністратор мережі	0,5	10000	15000
Системний програміст	0,25	10000	7500
Дизайнер WEB	0,25	10000	7500
Інженер-верстальник	0,25	10000	7500
Бухгалтер-економіст	0,5	10000	15000
Всього за період розробки	$R_{сп}$ = 6,35	-	$Ф_{роб}$ = 200787

Розрахуємо середньоденну зарплату одного виконавця:

$$З_{сд} = \frac{Ф_{роб}}{R_{сп} F_{рд}}, \quad (7.8)$$

де: $Ф_{роб}$ – загальна сума зарплати за плановий період, грн.

$$З_{сд} = \frac{200787}{6,35 \cdot 60} = 527 \ грн.$$

7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

Балансова вартість будівель визначається з урахуванням кількості робочих місць виконавців, питомої площі на одне робоче місце, та вартості одного

квадратного метра виробничої площі:

$$Б_{уд} = R^1_{сп} S_у Ц_{пл}, \quad (7.9)$$

де: $R^1_{сп}$ – кількість робочих місць виконавців, шт. Приймаємо 8 робочих місць;

$S_у$ – питома площа на одне робоче місце, $м^2$;

$Ц_{пл}$ – вартість одного квадратного метра площі, грн.

Згідно даних ТОВ науково-дослідницького консалтингового підприємства «Пекторaль» (м. Кіровоград) ціна одного квадратного метра площі новобудови, вік якої не перевищує 25 років, по місту складає 400…1600 $у.о./м^2$.

Враховуючи, що курс складає 1 у.о. = 37 грн. приймаємо для розрахунку вартість одного метра квадратного рівною 20000 грн./м^2.

На кожне робоче місце у середньому потрібно 8 $м^2$. З урахуванням цього:

$$Б_{уд} = 8 \cdot 8 \cdot 20000 = 1280000 \text{ грн.}$$

Вартість передавальних пристроїв складає 10% від вартості будівель, і у даному випадку вона складе: 128000 грн.

Балансова вартість інвентарю розраховується за нормою 3500 грн. на одне робоче місце. Тобто:

$$І_{нв} = R^1_{сп} \cdot Ц_м, \quad (7.10)$$

де: $Ц_м$ – ціна меблів для одного робочого місця, грн.

$$І_{нв} = 8 \cdot 3500 = 28000 \text{ грн.}$$

Балансова вартість обчислювальної техніки визначається по оптовим цінам постачальника з врахуванням витрат на транспортування.

Специфікація на обчислювальну техніку наведена в таблиці 7.7.

Дані по оптовій ціні на обладнання та комплектуючі вибирались по прайсу Інтернет-магазину Supercomp https://supercomp.kiev.ua/ за 06.11.23.

Таблиця 7.6 – Специфікація

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Персональний комп’ютер		10947
Системний блок		7347
Процесор	INTEL Pentium G6405 (BX80701G6405) 1200, 2 ядра, 4 потоки, 4.1 GHz, TDP – 58 Вт, 14nm	
Системна плата	ASRock H470M-HDV 1200, DDR4, 64 ГБ, 2933 MHz, LAN - 1 Гбіт/с, D-Sub (VGA), DVI, HDMI, 4 x Sata 6.0 Gb/s, Micro-ATX	
Відеокарта	Intel UHD Graphics 610,	
Жорсткий диск	SSD 2.5" 240GB AS340X Apacer (AP240GAS340XC-1) Серія - AS340X, 240 GB, 3D NAND, 2.5", SATA III (6Gb/s)	
Оперативна пам’ять	DDR4 8GB 2666 MHz Kingston (KVR26N19S8/8) DDR4, 8 ГБ, В наборі - 1	
Блок живлення	Gamemax 500W (GM-500B) ATX 12V v2.3, 500 Вт, 20+4 pin, CPU - 4+4pin, GPU - 1x6 pin, SATA - 3, Peripheral - 2, +12V1 - 20A, 1x120 мм, 150 x 140 x 86 мм	
Корпус	Vinga CS210B Класичний, Miditower, ATX, Micro - ATX, Mini - ITX, Слотів ррозширення - 7, Додотково - Front 2x12 см, Back 1x12 см, Side 2x12 см, GPU - до 370 мм, CPU - 166 мм	

Продовження таблиці 7.6

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Кулер	–	–
Кардрідер внутрішній	USB 2.0 Card reader STORM CR-35U1A4-B, int. 3.5", 1*USB2.0+AUDIO+1394, multi: All Type Cards, black	220
інше	Клавіатура, мишка	Подару-нок
Монітор	22" TFT, ASUS VW223D (5ms, 300/3000: 1, 170/160, D-SUB, Wide)	3600
Принтер лазерний	Canon i-SENSYS LBP6030W	2700
Принтер струминний	Epson Stylus Photo P50 (C11CA45341) + USB cable	5500
Копіювальний апарат	Canon i-SENSYS MF217W with Wi-Fi	5965

Таблиця 7.7 – Балансова вартість обчислювальної техніки

Найменування обчислювальної техніки	Кількість, шт.	Ціна за одиницю, грн.	Витрати на транспортування, монтаж та випробовування.	Загальна вартість, грн.
Персональні комп'ютери	15	10947	16420,5	180625,5
Принтер лаз.	2	2700	540	5940
Принтер струм.	1	5500	550	6050
Сканери	-	-	-	0
Копіюв. апарат	1	5965	596,5	6561,5
Всього	–	–	–	199177

Витрати на транспорт, монтаж та випробування можуть бути прийняті в межах до 10% від оптової ціни.

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.8.

Таблиця 7.8 – Вартість основних фондів та амортизаційні відрахування розробника

Групи та види основних фондів	Балансова вартість, грн.	Амортизація	
		Норма, %	Відрахування, грн.
1	2	3	4
Група 3			
1. Будівлі	1280000	-	-
2. Передавальні пристрої	128000	-	-
Всього по групі	1408000	5	70400
Група 4			
3. Обчислювальна техніка	199177	-	-
Всього по групі	199177	50	99588,5
Група 5, 6			
4. Вимірювальні пристрої	5190	25	1297,5
5. Транспортні засоби	0	20	0,0
6. Господарський інвентар	28000	25	7000
Всього по групі	33190	-	8297,5
7. Нематеріальні активи	120000	10	12000
Разом	$К_р$ = 1760367		$А_р$ = 190286

7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців:

$$З_о = \frac{З_{сд} \cdot Т_{пз}}{N_е}, \quad (7.11)$$

де: $N_е$ – кількість екземплярів програм, шт.

$$З_о = 527 \cdot 159/280 = 300 \text{ грн.}$$

Визначимо додаткову зарплату (оплата відпусток, виконання державних та суспільних обов'язків) на рівні 10%:

$$З_д = З_о \cdot Н_q \cdot 0{,}01, \quad (7.12)$$

де: $Н_q$ – норматив додаткової зарплати, %.

$$З_д = 300 \cdot 10 \cdot 0{,}01 = 30 \text{ грн.}$$

Відрахування на соціальні потреби за нормативом $Н_с = 22\%$ від суми основної та додаткової зарплати:

$$С_{оц} = 0{,}01 \cdot Н_с (З_о + З_д), \quad (7.13)$$

де: $Н_с$ – відрахування на соціальні потреби, %.

$$С_{оц} = 0{,}01 \cdot 22(300+30) = 73 \text{ грн.}$$

Визначимо загальногосподарські витрати (електроенергію, ремонт і утримання приміщень і т.д) за нормативом $Н_г = 15\%$ від основної зарплати:

$$Г_{осп} = З_о \cdot Н_г \cdot 0{,}01, \quad (7.14)$$

де: $Н_г$ – загальногосподарські витрати, %.

$$Г_{осп} = 300 \cdot 15 \cdot 0{,}01 = 45 \text{ грн.}$$

Визначимо витрати на матеріали для розробки програмної продукції за нормами споживання та діючими цінами за одиницю виміру:

$$З_М = (З_{М1} + З_{М2} + З_{М3})/N_е, \quad (7.15)$$

де: $З_{М1}$ – вартість паперу, грн.;

$З_{М2}$ – вартість запам'ятовуючих пристроїв, грн.;

$З_{М3}$ – вартість фарби, картриджів, тонеру, грн.;

N_e – кількість екземплярів програм, шт.

Згідно виданих норм приймаємо 1/6 пачки паперу на місяць розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п = 210$ грн., визначаємо вартість паперу за період розробки $N_м = 3$ міс:

$$З_{М1} = Ц_п \cdot n_р \cdot N_м. \quad (7.16)$$

$$З_{М1} = 210 \cdot 1/6 \cdot 3 = 105 \text{ грн.}$$

Згідно виданих норм до вартості запам'ятовуючих пристроїв входить вартість CD/DVD дисків в кількості 100 примірників:

$$З_{М2} = \sum Ц_д , \quad (7.17)$$

де: $Ц_д$ – вартість дисків CD/DVD: CDR TDK 700Mb, 80Min, 52x Cake box – 33,6 грн./шт., DVD-R LG 4,7Gb, 16x speed Cake box – 33,6 грн./шт.

$$З_{М2} = 33,6 \cdot 100 = 3360 \text{ грн.}$$

Згідно виданих норм одноразовій заправці підлягають усі друкуючі пристрої і становить:

$$З_{М3} = \sum Ц_з., \quad (7.18)$$

де: $Ц_з$ – вартість розхідних матеріалів друкуючих пристроїв: відновлення та заправка картриджу для Canon i-SENSYS LBP6030W – 574 грн.; картридж для Epson Stylus Photo P50 – 558 грн.; відновлення картриджу для MF217W – 570 грн.

$$З_{М3} = 574+558+570 = 1702 \text{ грн.}$$

$$З_М = (105 + 3360 + 1702)/280 = 18 \text{ грн.}$$

Визначимо витрати на освоєння нових мов програмування або операційних систем за нормативом ($Н_п = 15\%$) від основної зарплати виконавців:

$$О_п = З_о \cdot Н_п \cdot 0,01, \quad (7.19)$$

де: $Н_п$ – норматив витрат на освоєння нових мов програмування, %.

$$О_п = 300 \cdot 15 \cdot 0,01 = 45 \text{ грн.}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_e = 280$ прим.):

$$A_м = \frac{A_р \cdot N_{міс}}{N_е \cdot 12}, \quad (7.20)$$

де: $A_р$ – загальна річна сума амортизаційних відрахувань, грн.

$$A_м = 190286 \cdot 3/(280 \cdot 12) = 170 \text{ грн.}$$

Повна собівартість ПЗ визначається як сума витрат за попередніми статтями калькуляції:

$$C_п = З_о + З_д + C_{оц} + Г_{осп} + З_м + О_п + A_м. \quad (7.21)$$

$$C_п = 300+30+73+45+18+45+170 = 681 \text{ грн.}$$

Величини ціна підприємства, податок на додану вартість, відпускна ціна програмної продукції визначаються за формулами, приведеними в таблиці 7.9

Таблиця 7.9 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

Найменування статей витрат	Позначення	Величина, грн.
1. Основна зарплата виконавців	$З_о$	300
2. Додаткова зарплата виконавців	$З_д$	30
3. Відрахування на соціальні потреби	$C_{оц}$	73
4. Загальногосподарські витрати	$Г_{осп}$	45
5. Витрати на матеріали	$З_М$	18
6. Освоєння нових операційних систем, мов програмування	$О_п$	45
7. Амортизація основних фондів	$A_м$	170
8. Повна собівартість програмного забезпечення	$C_п$	681
9. Плановий прибуток	$П_р$	341
10.Ціна підприємства $Ц_п = C_п + П_р$	$Ц_п$	1022
11.Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$	$ПДВ$	204,4
12.Відпускна ціна програмної продукції $Ц = Ц_п + ПДВ$	$Ц$	1226,4

Визначимо плановий прибуток за рівнем рентабельності (P_n) програмної продукції, яка залежить від складності програми та ступеня новизни задачі.

Для даного програмного забезпечення рівень рентабельності складає 50%.

$$П_р = 0{,}01 \cdot P_n \cdot C_n, \quad (7.22)$$

де: P_n – рівень рентабельності, %.

$$П_р = 0{,}01 \cdot 50 \cdot 681 = 341 \text{ грн.}$$

7.6 Визначення об'єму капітальних вкладень у споживача програмної продукції

Об'єм капітальних вкладень у споживача програмної продукції визначаємо на основі балансової вартості основних фондів, яка враховує ціну, транспортно-заготівельні витрати, вартість будівель, монтажних та пусконалагоджувальних робіт, а також витрати на випробування у виробничих умовах.

Результати розрахунків зводимо у таблицю 7.9.

Таблиця 7.10 – Розрахунок об'єму капітальних вкладень у споживача програмної продукції

Найменування капітальних вкладень	Сума за варіантами, грн.	
	Базовий	Новий
Вартість програмної продукції	–	1226
Всього капітальних витрат	–	1226

7.7 Визначення експлуатаційних витрат

Експлуатаційні витрати у споживача програмної продукції визначаємо при умові роботи підсистеми на протязі року. Результати зводимо до таблиці 7.11.

Таблиця 7.11 – Розрахунок експлуатаційних витрат у споживача програмної продукції

Найменування статей витрат	Позначення	Сума витрат за варіантами, грн.	
		Базовий	Новий
1. Витрати на обслуговування системи	$З_р$	15190	3032
2. Витрати на електроенергію	$З_{ел}$	2692	1864
3. Витрати на амортизацію	$З_{ам}$	0	307
Всього витрат за рік	$І$	17882	5203

Витрати на профілактичні роботи:

$$З_р = Т_р \cdot З_г \cdot (1 + 0{,}01 \cdot Н_q) \cdot (1 + 0{,}01 \cdot Н_с), \quad (7.23)$$

де: $Т_р$ – кількість годин обслуговування кожного комп'ютера за рік, *год.*;

$З_г$ – заробітна плата обслуговуючого персоналу, *грн / год*.

Після купівлі нового програмного забезпечення витрати на обслуговування системи зменшились з 15190 грн до 3032 грн на рік.

Витрати по амортизації визначаються на основі норм амортизаційних відрахувань, вартості програмної продукції і основних фондів. Для розрахунку складаємо таблицю 7.12.

Таблиця 7.12 – Розрахунок амортизаційних відрахувань

Групи основних фондів	Норма амортизації %	Балансова вартість, грн., за варіантами		Сума відрахувань, грн., за варіантами	
		Базовий	Новий	Базовий	Новий
Програмна продукція	25	–	1226	–	306,5
Всього відрахувань	-	–	1226	–	306,5

Витрати на електроенергію визначаються з урахуванням споживаємої потужності ($П_{ел}$) в кіловатах, часу експлуатації технічних засобів ($T_р$) в годинах та ціни однієї кіловат-години ($Ц_{ел}$):

$$З_{ел} = П_{ел} \cdot T_р \cdot Ц_{ел}. \quad (7.24)$$

$$З_{ел\ баз} = 0{,}545 \cdot 1300 \cdot 3{,}8 = 2692 \text{ грн.}$$

$$З_{ел\ нов} = 0{,}545 \cdot 900 \cdot 3{,}8 = 1864 \text{ грн.}$$

7.8 Визначення економічної ефективності програмної продукції

Економічна ефективність програмного забезпечення визначається для виготовлювача і споживача за такими показниками.

Величина економічного ефекту при виготовленні програмної продукції, розраховуємо за формулою:

$$E_в = (Ц_п - C_п) \cdot N_е - \sum_{i=1}^{m} E_{p_m} \cdot K_{p_m}, \quad (7.25)$$

де: $K_р$ – балансова вартість основних фондів розробника, грн.; $E_р$ – розрахунковий коефіцієнт капіталовкладень.

$$Ев = (1022 - 681) \cdot 280 - (0{,}05 \cdot 1408000 + 0{,}5 \cdot 199177 +$$
$$+0{,}25 \cdot 33190 + 0{,}1 \cdot 28000) \cdot 3/12 = 50208 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

$$T_в = \frac{K_р}{(Ц_п - C_п) \cdot N_е}, \quad (7.26)$$

де: $K_р$ – балансова вартість основних фондів розробника.

$$T_в = \frac{1760367}{(1022 - 681) \cdot 280 \cdot 12 / 3} = 4{,}6 \text{ роки}$$

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

$$E_{сп} = (I_б - I_н) - E_н (K_н - K_б), \quad (7.27)$$

де: $I_б$, $I_н$ – величина експлуатаційних витрат за базовим и новим варіантом відповідно;

$K_б$, $K_н$ – об'єм капітальних вкладень за варіантами, що порівнюються.

$$E_{сп} = (17882\text{-}5203)\text{-}0{,}25 \cdot 1226 = 12373 \text{ грн.}$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.13.

Таблиця 7.13 – Показники економічної ефективності програмної продукції

Найменування показників	Одиниця виміру	Величина
1. Кількість екземплярів програми	Прим.	280
2. Повна собівартість розробленої програми	Грн.	681
3. Ціна розробленої програми	Грн.	1022
4. Плановий прибуток від реалізації розробленої програми	Грн.	341
5. Рентабельність програмної продукції	%	50
6. Об'єм додаткових капітальних вкладень у виробника програмної продукції	Грн.	1760367
7. Загальний прибуток від реалізації програмної продукції	Грн.	95480
8. Величина економічного ефекту при виготовлені програмної продукції	Грн.	50208
9. Період окупності додаткових капітальних вкладень у виробника програмної продукції	Роки	4,6
10.Об'єм додаткових капітальних вкладень у споживача програмної продукції	Грн.	1226
11.Величина економічного ефекту у користувача програмної продукції	Грн.	12373
12.Період окупності додаткових капітальних вкладень у користувача програмної продукції	Років	0,1

Визначимо період окупності додаткових капітальних вкладень у споживача програмної продукції за рахунок зниження експлуатаційних витрат:

$$T_{сп} = \frac{K_н - K_б}{I_б - I_н}, \tag{7.28}$$

$$T_{сп} = \frac{1226}{17882 - 5203} = 0{,}1 \; року \, .$$

7.9 Висновки

Розроблена програма економічно вигідна. За рахунок впровадження програмного забезпечення досягається скорочення часу обробки інформації, підвищується культура праці, підвищення якості приймаючих управлінських рішень.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

Законом України “Про охорону праці” регламентуються загальні положення державної політики в галузі охорони праці, а конкретизуються ці положення нормативно-правовими актами про охорону праці, зокрема Наказом Міністерства соціальної політики України 14.02.2018 № 207, який зареєстровано в Міністерстві юстиції України 25 квітня 2018 р. за №508/31960 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями», яким затверджено нормативно-правовий акт з охорони праці НПАОП 0.00-7.15-18, «Правила охорони праці під час експлуатації електронно-обчислювальних машин», та «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98.

Програмісти у процесі роботи мають негативний вплив на органи зору, а також мають значну розумову напругою і нервово-емоційне навантаження. Руки (суглоби польців та м’язи рук) при роботі з клавіатурою мають теж істотне навантаженням. До шкідливих факторів, які впливають на робитників галузі інформаційних технологій (ІТ) спеціалісти відносять високочастотні електромагнітні коливання (випромінювання) роботи апаратної частини ЕОМ та виділення шкідливих газів.

Ці шкідливі фактори можуть привести до професійних захворювань.

Розглянемо шкідливі чинники роботи програмістів керуючись наступними нормативно-правовими актами: «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98, та «Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» НПАОП 0.00-7.15-18.

Умови праці програміста включають наступні фактори:

- параметри повітряного середовища в приміщенні;
- вентиляція приміщення;
- освітлення приміщення;
- параметри повітряного середовища в приміщенні, тощо.

Щоб запропонувати заходи щодо зменшення негативного впливу комп'ютера на організм людини визначемо фактори, які можуть викликати професійне захворювання і впливають на працездатність програміста.

8.2 Шкідливі і небезпечні фактори при роботі з комп'ютером

Електронно-обчислювальна машин (ЕОМ) та інше обладнання є джерелами небезпеки ураження електричним струмом. Так як робота програміста характеризується істотним зоровим навантаженням, то вимагає належного освітлення. У приміщенні, в якому працюють люди (у т.ч. програмісти) необхідно створити належний мікроклімат, праметри якого регламентуються, Державними санітарними правилами і нормами, зокрема ДСанПіН 3.3.2.007-98.

При роботі з використанням ЕОМ відзначають наступні небезпечні та шкідливі фактори:

- ризик виникнення надзвичайних ситуацій природного або штучного характеру на об'єкті або території.
- ризик виникнення пожежі;
- негативний вплив на органи зору людини;
- ризики ураження електричним струмом;
- недостатня, або надмірна освітленість робочого місця;
- електромагнітні (у т.ч. високочастотні) електромагнітні випромінювання (коливання);
- несприятливі мікрокліматичні умови;
- нервово-емоційна напруженість праці;

- інтелектуальні навантаження;

- монотонність праці;

- невідповідність ергономічних показників робочого місця діючим вимогам;

- шум;

- статичні навантаження на кістково-м’язовий апарат;

8.3 Аналіз санітарно-гігієнічних умов праці на робочому місці програміста

Розглянемо умови праці у приміщенні, в якому працюють програмісти. Геометричні розміри приміщення наведено у таблиці 8.1.

У зазначеному приміщенні працюють двоє людей. За даними, які наведено у табл. 8.1, та табл. 8.2, можна зробити висновок, що площа та об’єм приміщення у розрахунку на одно робоче місце програміста не відповідають нормативним вимогам ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин», але відповідають нормативним вимогам Наказу Міністерства соціальної політики України № 207, від 14.02.2018 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» та НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин»). Таним чином можна зробити висновок, що санітарно-гігієнічні умови праці на робочому місці програміста відповідають вимогам.

Таблиця 8.1 – Розміри приміщення

Найменування	Значення, м
Ширина	3
Довжина	4,6
Висота	3

Температура повітря в приміщенні визначається впливом температури зовнішнього повітря і тепловою енергією, яка виділяється всередині приміщення. Джерелами виділення теплоти в даному приміщенні є електроустаткування,

освітлювальні прилади, а також люди. У світлий час доби джерелом надлишкового тепла є сонячна радіація. Згідно Постанови № 42 від 01.12.1999 Головного державного санітарного лікаря України, робота, виконувана в даному приміщенні, відноситься до категорії Іа. В цьому випадку людина витрачає енергії до 120 ккал у годину. Вологість повітря в приміщенні визначається впливом багатьох факторів, серед яких: вологість атмосферного повітря, виділення вологи людьми (при диханні та випарами з поверхні шкіри).

Таблиця 8.2 – Площа та обсяг приміщення, на одного працюючого*

Геометрична характеристика	**Одиниця виміру**	**Нормативне значення***	**Фактичне значення**
Площа, S	$м^2$	не менше 6.0	6,9
Об'єм, V	$м^3$	не менше 20.0	20,7

* Згідно ДСанПіН 3.3.2.007-98 (Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин).

Мікроклімат повітряного середовища в приміщенні характеризується запиленістю та загазованістю повітря. Мікроклімат приміщення визначається діючим на організм людини поєднанням, вологості, температури, швидкості руху повітря та інтенсивності теплового випромінювання. Аналіз мікроклімату складається з визначення зазначених вище факторів і порівняння результатів із встановленими нормами.

У таблиці 8.3 наведено оптимальні та фактичні значення параметрів мікроклімату як для категорії ваги робіт Іа, так і розглянутого приміщення. У приміщеннях, де встановлено ЕОМ, рекомендується застосування тільки оптимальних значень показників мікроклімату.

Таблиця 8.3 – Оптимальні і фактичні значення параметрів мікроклімату

Пора року	**Оптимальні для Іа**			**Фактичні**		
	Температура, °С	**Вологість,%**	**Швидкість повітря, м/с**	**Температура,°С**	**Вологість%**	**Швидкість повітря, м/с**
Холодна	22-24	40-60	0,1	22-24	40-55	0,12
Тепла	23-25	50-70	0,1	24-25	50-65	0,9

Проведений аналіз показує, що показники мікроклімату в приміщенні відповідають установленим нормам. Штучне опалення застосовується у холодний період року.

В літню пору застосовується кондиціонер.

Для боротьби з пилом робляться регулярні провітрювання та вологі прибирання приміщенні.

У приміщенні знаходяться наступні джерела шуму: принтер Xerox WorkCentre 3025BI (3025VBI), електродвигуни вентиляторів ЕОМ.

Одним з найважливіших факторів, які впливають на ефективність трудової діяльності людини, та попереджають травматизм і професійні захворювання програмістів є освітлення на робочому місці.

З 2019 року діють Державні будівельні норми України “Природне і штучне освітлення” – ДБН В.2.5-28:2018, у яких прописані вимоги до використання всіх освітлювальних приладів, у т.ч. світлодіодних.

Працю працівника, який постійно працює за комп’ютером, згідно ДБН В.2.5-28:2018, можна віднести до роботи з малою точністю (найменший розмір об’єкта розрізнення від 1 до 5 мм) V-го розряду зорової роботи, з великою контрастністю об’єкта розрізнення (символів на екрані дисплея), з темним тлом (під розряд зорової роботи В). Приміщення можна віднести до 1-ої групи приміщень, у яких проводиться розрізнення об’єктів зорової роботи при фіксованому напрямку лінії зору того, що працює на робочу поверхню. Для такого типу приміщень і розряду зорової роботи нормоване значення коефіцієнта природної освітленості (КПО) робочої поверхні (при поєднаному, спільному освітленні), повинен становити не більше 1,5%, освітленість при штучному висвітленні повинна становити 300 Лк., Крім того все поле зору повинне бути освітлено достатньо рівномірно – ця основна гігієнічна вимога. Так як яскраве світло на ділянці периферійного зору значно збільшує напруженість очей і, як наслідок, призводить до їх швидкої стомлюваності, ступінь освітлення приміщення і яскравість екрану комп’ютера повинні бути приблизно однаковими.

8.4 Розробка заходів з умов поліпшення охорони праці

Згідно аналізу умов праці в розглянутому приміщенні, ми одержали наступні результати:

- розмірі приміщення, у розрахунку на одному працюючого, відповідають нормативам;

- мікроклімат відповідає нормативному значенню;

- акустичні умови роботи не перевищують нормативних значеннь;

Таким чином можна припустити, що основною причиною можнивого зниження працездатності програміста є психофізіологічний фактор, тому основна пропозиція буде така: дотримання позитивної психологічної атмосфери в колективі та регламентованого режиму праці та відпочинку, організація робочого місця з урахуванням ергономічних вимог.

Рекомендовані заходи: регулярні періодичні наочні огляди персоналом шляхів для евакуації людей із приміщення, відповідно до плану евакуації (який повинен розташовуватись на видному місці у приміщенні), включення до колективного договору мінімально можливого вмісту аптечок з обов'язковою наявністю масок-клапанів, або іншого спорядження для штучного дихання. Регулярна періодична перевірка параметрів заземлення та занулення (вимірювання опору ланцюга).

Регулярна наочне знайомство персоналу із шляхами для евакуації людей із приміщення відповідно до плану евакуації, забезпечення розподільних щитів спеціальними розетками з заземлюючими контактами; організація заземлення всіх приладів і пристроїв, які працюють при напрузі вище 36 В.

Так як при ураженні електричним струмом у людини може статися фібриляція шлуночків серця, в організації бажано мати дефібрилятор і підготовлений персонал для роботи з ним.

8.5 Розрахункова частина

Початкові дані для розрахунку захисного штучного заземлення: опір заземлювача, який нормується: $R_{ЗН} = 4$ *Ом*.

Для захисного штучного заземлення застосовуються ветикальні електроди з метелевого прутка діаметром 30 мм. (*D=30 мм.=0,03 м.)* довжиною *L*=2,5 *м.* та горизонтальний електрод — метелева полоса з перетином 40*4 мм. тип грунта — глина (питомий опір 40 Ом*м). Відстань між вертикальними заземлювачами (електродами) *А*=3 *м*.

Глибина закладення горизонтального контура заземлення *t= 0,8 м*.

Умовна товщина верхнього шару грунта: *H=0,4 м*. Напруга — 220/380 В. Розрахункова схема розташування заземлюючих електродів — у ряд.

Розрахунок проводиться за допустимим опором розтіканню струму заземлювача.

Необхідно визначити необхідну кількість вертикальних заземлювачів та длину полоси (горизонтального заземлювача).

Розрахунок захисного заземлення можна автоматизувати за допомогою програми, сирцевий код якої опублікован на стр. 13-16 [52], або аналогічної.

Розрахунок.

Відстань від центра вертикального заземлювача до поверхні землі:

$$T=t+L/2=0,8+2,5/2=2,05 \text{ м.}$$

Розрахунковий питомий опір грунта (з врахуванням того, що фактично вся конструкція заземлювача розташовується у нижньому шарі грунта):

$$\rho = \psi \rho_2 = 1,36*40=54,5 \text{ Ом*м.}$$

де $\psi = 1,36$ – табличне значення коефіцієнта сезонності для відповідної кліматичної зони у багатошаровому грунті; $\rho_1 = 50$ *Ом*м.* – табличне значення питомого опору верхнього шару грунта; $\rho_2 = 40$ *Ом*м*. – табличне значення питомого опору нижнього шару грунта.

Опір розтіканню електричного струму одного електрода вертикального

заземлювача:

$$R_o = 0{,}366\frac{\rho}{L}\left(lg\frac{2L}{D} + \frac{1}{2}lg\frac{4T+L}{4T-L}\right) = 0{,}366\frac{54{,}5}{2{,}5}\left(lg\frac{2\cdot 2{,}5}{0{,}03} + \frac{1}{2}lg\frac{4\cdot 2{,}05+2{,}5}{4\cdot 2{,}05-2{,}5}\right) = 20{,}1 \text{ Ом.}$$

Відношення *A/L=3/2,5=1,2.*

Визначаємо коефіцієнт екранування вертикальних електродів *Кев=0,8* при попередній (орієнтовній) кількості вертикальних електродів, яке дорівнює 4.

Визначаємо необхідну кількість вертикальних заземлювачів (без врахування горизонтального заземлювача), при $R_{ЗН} = 4$ *Ом* :

$$N=R_O /(Кев\ R_{ЗН})= 20{,}1 / (0{,}8*4)=5{,}87 \approx 6 \text{ шт.}$$

Визначаємо довжину з'єднуючої полоси:

$$L_П =1{,}05*A*N= 1{,}05*3*6=18{,}8\approx 19 \text{ м.}$$

Опір розтіканню електричного струму з'єднуючої полоси:

$$R_П =0{,}366(\rho_2*K_П/L_П)lg(2(L_П *L_П)/(К*t))=$$

$$=0{,}366(40*5/16)*[lg(2*16*16)/(0{,}04*0{,}8)]=16{,}5 \text{ Ом.}$$

де $K_П=5$ - табличне значення коефіцієнта сезонності для відповідної кліматичної зони з'єднуючої полоси.

Загальний опір розтіканню електричного струму заземлювача:

$$R =(R_O*R_П)/(R_O*\eta_П+ N*R_П*Кев)=$$

$$=(20{,}1* 16{,}5)/(20{,}1* 0{,}75+ 6* 16{,}5* 0{,}8)=3{,}32 \text{ Ом.}$$

де $\eta_П = 0{,}75$ - табличне значення коефіцієнта екранування з'єднуючої полоси.

Умова $R \le R_{ЗН}$ виконується ($3{,}32 \le 4$).

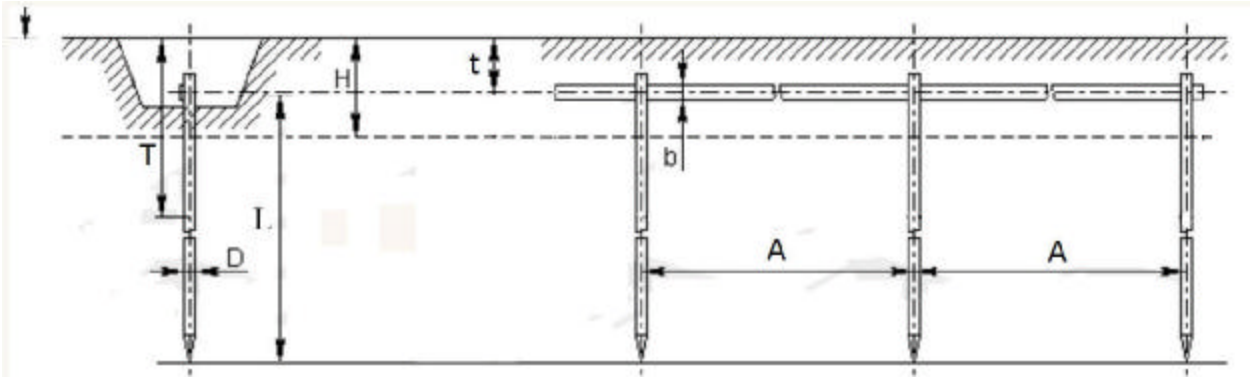

Рисунок 8.1 – Схема штучного заземлення

8.6 Висновки

Дотримання всіх необхідних умов праці не лише сприяє збереженню здоров’я працівників, а також підвищує ефективність виробництва вцілому.

З цих міркувань було здійснено аналіз приміщення, призначеного для праці програмістів, проведено розгляд небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи. Виконано розрахунок штучного освітлення, як одного з ключових факторів впливу на працездатність та здоров’я програміста. Розроблено заходи з охорони праці.

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання магістерської роботи, призначено для реалізації системи пошуку даних у децентралізованих однорангових мережах.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

У магістерській роботі наведені теоретичне узагальнення й рішення наукового завдання дослідження методів пошуку даних у децентралізованих однорангових мережах.

Рішення даного завдання полягало у вирішенні наступних задач:

– Дослідження існуючих систем пошуку даних у децентралізованих однорангових мережах.

– Розробка методів та алгоритмів роботи системи пошуку даних у децентралізованих однорангових мережах.

– Програмна реалізація системи пошуку даних у децентралізованих однорангових мережах.

Розроблені під час виконання магістерської роботи алгоритми дозволяють успішно вирішувати завдання пошуку даних у децентралізованих однорангових мережах.

Проведено аналіз предметної галузі в ході якого були виявлені об’єкти, взаємодія яких носить істотний характер для функціональної діяльності предметної галузі, і їхні основні характеристики; побудована алгоритм і вибраний середовище розробки.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об’єктно-

орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня Python у середовищі розробки Anaconda. Дана мова програмування дозволяє найбільш ефективно здійснювати поставлені задачі. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для захисту програмного забезпечення від несанкціонованого використання та поширення був обраний криптографічний алгоритм ECDSA.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

Розроблена програма має реальний економічний ефект від її впровадження у виробництво у сумі 12373 грн. З урахуванням вартості розробки програми та обладнання, строк окупності становить 0,1 роки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Aho A.V., Hopcroft J.E., Ullman J.D. Data Structures and Algorithms. – Pearson, 2001. – 620 p.
2. Aweya J. IP Routing Protocols: Link-State and Path-Vector Routing Protocols. 1st ed. CRC Press, 2021. 438 p.
3. Barabási A.-L. Network Science. Cambridge University Press, 2018. 475 p. URL: http://networksciencebook.com/.
4. Barabási A.-L., Albert R. Emergence of scaling in random networks. Science. 1999. Vol. 286, no. 5439 P. 509–512. DOI: 10.1126/science.286.5439.509.
5. Bellet A., Guerraoui R., Taziki M., Tommasi M. Personalized and Private Peer-to-Peer Machine Learning. Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics.2018. Vol. 84. P. 473-481. URL: http://proceedings.mlr.press/v84/bellet18a/bellet18a.pdf
6. Cisco. Dynamic routing protocols. Cisco Press. 2001. URL: https://www.ciscopress.com/articles/article.asp?p=24090&seqNum=4.
7. Cisco. IP Routed protocols. Technology Support. 2022. URL: https://www.cisco.com/c/en/us/tech/ip/ip-routed-protocols/index.html.
8. Distributed Hash Tables with Kademlia. URL: https://codethechange.stanford.edu/guides/guide_kademlia.html
9. Gnutella Protocol Development. 2003. URL: https://rfc-gnutella.sourceforge.net
10. Gusfield D. Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology 1st Edition. – Cambridge University Press, 2008. – 556 p.
11. Kademlia: A Design Specification. 2010. URL: https://xlattice.sourceforge.net/components/protocol/kademlia/specs.html
12. Klots Y., Muliar I., Cheshun V., Burdyug O. Use of distributed hash tables

to provide access to cloud services. Collection of scientific works of the Military Institute of Kyiv National Taras Shevchenko University, 2020. Vol. 11(67). P. 85–95.

13. Knuth D. Art of Computer Programming, The: Combinatorial Algorithms, Volume 4A, Part 1.Addison-Wesley Professional; 1st edition. 2011. 912p.

14. Knuth, Donald E. The Art of Computer Programming: Fundamental Algorithms. 3rd Ed. Addison-Wesley, 1997.

15. Knuth, Donald E. The Art of Computer Programming: Seminumerical Algorithms. 3rd Ed. Addison-Wesley, 1998.

16. Knuth, Donald E. The Art of Computer Programming: Sorting and Searching. 2nd Ed. Addison-Wesley, 1998.

17. Koo S.G.M. Multimedia Content Distribution Using Peer-to-Peer Overlay Networks: The Design and Analysis of the Next Generation Peer-to-Peer Networks. VDM Verlag Dr. Müller. 2008. 88 p.

18. Lambert K. A. Fundamentals of Python: First Programs, 2nd Edition. – Cengage, 2019.

19. Li Q.-L., Lui J. C. S. Block-structured supermarket models. Discrete Event Dynamic Systems. 2014. Vol. 26, no. 2. P. 147–182. DOI: 10.1007/s10626-014-0199-1.

20. Lutz M. Learning Python, 5th Edition Fifth Edition. - O'Reilly Media, 2016. - 1643 p.

21. Lutz M. Python: Pocket Reference Fourth Edition. - O'Reilly Media, 2016. - 210 p.

22. Milojicic D.S., Kalogeraki V., Lukose R., Nagaraja K., Pruyne J., Richard B., Rollins S., Xu Z. Peer-to-peer computing. Technical Report HPL-2002-57, HP Labs. 2002. 51 p. URL: https://www.cs.kau.se/cs/education/courses/dvad02/p2/seminar4/Papers/HPL-2002-57R1.pdf

23. Moy J. T. OSPF: Anatomy of an Internet Routing Protocol. Addison-Wesley Professional, 1998.

24. Raaijmakers Y., Albrecher H., Boxma O. The single server queue with mixing dependencies. Methodology and Computing in Applied Probability. 2019. Vol.

21. P. 1023–1044. URL: http://www.hec.unil.ch/halbrech_files/QueueMixing.pdf.

25. Ratnasamy S., Stoica I., Shenker S. Routing Algorithms for DHTs: Some Open Questions. Peer-to-Peer Systems. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg. 2002. Vol. 2429. P. 45-52. DOI: 10.1007/3-540-45748-8_4

26. RicardoD. Introduction to Bitcoin: Understanding Peer-to-Peer Networks, Digital Signatures, the Blockchain, Proof-of-Work, Mining, Network Attacks, Bitcoin Core ... Safety. Expiscor Books; 1st edition. 2021. 60 p.

27. Riposo Ju. Diffusion on the Peer-to-Peer Network.LAP LAMBERT Academic Publishing. 2022. 100 p.

28. Shen Yb., Gadekallu T.R. Resource Search Method of Mobile Intelligent Education System Based on Distributed Hash Table. Mobile Netw Appl. 2022. Vol. 27. P. 1199-1208. DOI: 10.1007/s11036-022-01940-8

29. Shukla N., Datta D., Pandey M. Towards software defined low maintenance structured peer-to-peer overlays. Peer-to-Peer NetwAppl.2021. Vol. 14. P. 1242–1260. DOI: 10.1007/s12083-021-01112-7

30. Sobh T., Elleithy K., Mahmood A. Novel Algorithms and Techniques in Telecommunications and Networking. Springer, 2010. P. 41–46.

31. Stoica I., Morris R., Karger D.R., Kaashoek M.F., Balakrishnan H. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. ACM SIGCOMM Computer Communication Review.2001. Vol. 31(4). DOI: 10.1145/964723.383071

32. Tadimety P. R. OSPF messages. OSPF: A Network Routing Protocol. Berkeley, CA: Apress, 2015. DOI: 10.1007/978-1-4842-1410-7_18.

33. The BitTorrent Protocol Specification. 2017. URL: http://www.bittorrent.org/beps/bep_0003.html

34. The Python Tutorial. –https://docs.python.org/3/tutorial/index.html

35. Traag V. A. Algorithms and dynamical models for communities and reputation in social networks. Springer International Publishing. 2014. P. 229. URL: https://doi.org/10.1007/978-3-319-06391-1.

36. URL: https://link.springer.com/article/10.1007/BF02368250#citeas.

37. Watts D. J., Strogatz S. H. Collective dynamics of “small-world” networks. Nature. 1998. Vol. 393, no. 6684. P. 440–442. URL: https://www.nature.com/articles/30918.

38. Zeinalipour-Yazti D., Kalogeraki V., Gunopulos D. Information retrieval techniques for peer-to-peer networks. Computing in Science & Engineering, 2004. Vol. 6, No. 4.P. 20-26. DOI: 10.1109/MCSE.2004.12

39. Алгоритми і структура даних: Навчальний посібник / В.М.Ткачук. - ІваноФранківськ : Видавництво Прикарпатського національного університету імені Василя Стефаника, 2016. 286 с.

40. Алгоритми та структури даних. Навчальний посібник / Т. О. Коротєєва. Львів : Видавництво Львівської політехніки, 2014. - 280 с.

41. Державні будівельні норми України: ДБН В.2.5-28:2018. - Режим доступу до ресурсу: https://e-construction.gov.ua/laws_detail/3074958732556240833?doc_type=2

42. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин: ДСанПІН 3.3.2-007-98. - Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/v0007282-98

43. Дрєєва Г. М., Мелешко Є. В., Міхав В. В. Програмна імітаційна модель комп’ютерної мережі для тестування алгоритмів маршрутизації трафіку. Автоматика, комп’ютерно-інтегровані технології та проблеми енергоефективності в промисловості і сільському господарстві: матеріали Міжнар. наук.-техн. конф. (Кропивницький, 10–11 листоп. 2022 р.) / М-во освіти і науки України, Центральноукр. нац. техн. ун-т. Кропивницький: Ексклюзив-Систем, 2022. С. 44–45.

44. Закон України «Про охорону праці» від 14.10.1992 р. № 2694-XII. - Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2694-12

45. Зеркалов Д. В. Охорона праці в Галузі: Загальні вимоги: навч. посіб. Київ: Основа. 2011. 551 с.

46. Креневич А.П. Алгоритми і структури даних. Підручник. Київ, ВПЦ "КиївськийУніверситет".2021. 200 с.

47. Мелешко Є. В., Якименко М.С., Поліщук Л.І. Алгоритми та структури даних // Навчальний посібник для студенів технічних спеціальностей денної та заочної форми навчання. – Кропивницький: Видавець – Лисенко В.Ф., 2019. – 156 с. – URL: http://dspace.kntu.kr.ua/jspui/handle/123456789/8944

48. Міхав В. В., Мелешко Є. В., Лавданський А. О. Дослідження принципів роботи однорангових децентралізованих структурованих комп'ютерних мереж. *Інформаційна безпека та комп'ютерні технології*: тези доп. VI Міжнар. наук.-практ. конф. (Кропивницький, 20–21 квіт. 2023 р.) / М-во освіти і науки України, Центральноукр. нац. техн. ун-т. Кропивницький: ЦНТУ, 2023. С. 51–52.

49. Міхав В. В., Мелешко Є. В. Метод роботи рекомендаційної системи у комп'ютерній мережі типу peer to peer. *Системи управління, навігації та зв'язку*: ПНТУ, Полтава. 2023. Т. 1(71). С. 112-117. DOI: 10.26906/SUNZ.2023.1.112. URL: http://journals.nupp.edu.ua/sunz/article/
view/2839

50. Наказ Міністерства соціальної політики України 14.02.2018 № 207 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями». - Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0508

51. Оришака, О. В. Основи охорони праці: навч. посіб. / О. В. Оришака, Г. П. Горбачова, К. М. Марченко; М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т. - Кропивницький : ЦНТУ, 2022. - 175 с. – Режим доступу до ресурсу: http://dspace.kntu.kr.ua/jspui/handle/123456789/12161 (дата звернення 19.09.22).

52. Охорона праці. Ч. 1. Захисне заземлення: метод. вказ. до викон. розрахунків з викор. персон. ЕОМ ІВМ сумісного типу / Кіровоград. ін-т с.-г. машинобуд.; [укл. О. В. Оришака, Є. К. Солових, В. О. Оришака]. - Кіровоград:

КІСМ, 1997. - 20 с. Режим доступу до ресурсу: http://dspace.kntu.kr.ua/jspui/handle/123456789/4358

53. Пасічник В. В., Іванущак Н. М. Дослідження та моделювання складних мереж. Східно-Європейський журнал передових технологій. 2010. Вип. 2, № 3 (44). С. 43–48.

54. Постанова № 42 від 01.12.1999 Головного державного санітарного лікаря України «Санітарні норми мікроклімату виробничих приміщень ДСН 3.3.6.042-99. - Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/va042282-99

Додаток А
(обов'язковий)

Технічне завдання

Зміст

					ВКРМ-123.23.0039.00.00.ТЗ		
Вим.	Арк.	№ документа	Підпис	Дата			
Розробив		Копаніцький С.М.			Дослідження та програмна реалізація системи пошуку даних у децентралізованих однорангових мережах	Літ. М	Аркуш 1 / Аркушів 6
Перевірив		Мелешко Є.В.					
Н. Контр.		Коваленко А.С.				ЦНТУ КІ-22М-2	
Затв.		Смірнов О.А.					

1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію системи пошуку даних у децентралізованих однорангових мережах.

2 Підстава для розробки

Підставою для розробки служить завдання на магістерську роботу, видане на кафедрі кібербезпеки та програмного забезпечення (нак. №__-__ від __.__.20__ року).

3 Мета та призначення розробки

Метою магістерської роботи є дослідження та програмна реалізація системи пошуку даних у децентралізованих однорангових мережах.

4 Джерела розробки

Джерелом цієї магістерської роботи є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– техніко-економічне обґрунтування доцільності прийнятого до розробки

програмного забезпечення;

– аналіз умов праці розробників програмного забезпечення;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– пошук даних у децентралізованих однорангових мережах;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Застосунок, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

5.8.2 Мова програмування

Мова програмування високого рівня Python.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД.

7 Економічні вимоги

7.1 Для ПЗ необхідно виробити функціонально-вартісний аналіз варіантів розробки.

7.2 Виконати розрахунок витрат показників економічного ефекту з урахуванням цін на 1 вересня 2023 року.

8 Вимоги щодо охорони праці

В частині охорони праці випускної кваліфікаційної роботи повинні бути розглянуті умови праці програмістів під час розробки програмного забезпечення.

9 Перелік документів, що розробляються

– Наукова новизна – 1 аркуш.
– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуші.
– Показники економічної ефективності – 1 аркуш.
– Пояснювальна записка – 93 аркушів.

10 Етапи розробки

10.1 Збір і обробка інформації по темі магістерської роботи. Постановка задачі на виконання магістерської роботи (складання ТЗ).

10.2 Проведення досліджень або експериментальних робіт для уточнення основних положень магістерської роботи.

10.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Розрахунок з техніко-економічного обґрунтування.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання магістерської роботи на попередній захист __.__.2023 р.

11.2 Подання магістерської роботи на захист __.__.12.2023 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ
Керівник випускної кваліфікаційної роботи
за другим (магістерським) рівнем вищої освіти
_____ Є.В. Мелешко

Дослідження та програмна реалізація системи пошуку даних у децентралізованих однорангових мережах

Лістинг програми

Код документу 12

Носій: DVD-диск

Загальна кількість аркушів: 23

Літера: РП

Кропивницький – 2023 року

Файл simulation.py – імітаційне моделювання пошуку файлу у децентралізованій одноранговій мережі

```
import tkinter as tk
from tkinter import ttk
from tkinter import Canvas, Text
import random
import math

class NetworkNode:
    def __init__(self, id, x, y, hash_value):
        self.id = id
        self.x = x
        self.y = y
        self.hash = hash_value
        self.neighbors = []
        self.dht_table = {}  # DHT таблиця для кожного вузла

    def add_to_dht(self, key, value):
        self.dht_table[key] = value

    def get_dht_table(self):
        return self.dht_table

class Network:
    def __init__(self):
        self.nodes = []

    def add_node(self, node):
        self.nodes.append(node)

    def add_edge(self, node1, node2):
        node1.neighbors.append(node2)
        node2.neighbors.append(node1)

    def find_node_by_id(self, id):
        for node in self.nodes:
            if node.id == id:
                return node
        return None

class DHTTable:
    def __init__(self, root):
        self.root = root
        self.root.title("Table")
        self.text = Text(self.root, wrap=tk.WORD)
        self.text.pack()

    def show_table(self, node):
        self.text.delete("1.0", tk.END)
        dht_table = node.get_dht_table()
        for key, value in dht_table.items():
            self.text.insert(tk.END, f"Key: {key}, Value: {value}\n")

def kademia_algorithm(network, start_node, target_hash, dht_table):
    visited = set()
    visited.add(start_node)  # Почнемо з першого вузла

    stack = [(start_node, [])]

    while stack:
        current_node, path = stack.pop()
```

```
        if current_node.hash == target_hash:
            return path + [current_node]

        # Визначте список сусідів в порядку збільшення їх хеш-значень
        sorted_neighbors = sorted(current_node.neighbors, key=lambda node:
node.hash)

        for neighbor in sorted_neighbors:
            if neighbor not in visited:
                visited.add(neighbor)
                stack.append((neighbor, path + [current_node]))
                # Оновлення таблиці DHT з відвіданими вузлами
                dht_table.add_entry(neighbor, f"Visited by NetworkNode
{current_node.id}")
                neighbor.add_to_dht(target_hash, f"Visited by NetworkNode
{current_node.id}")

    return None

class NetworkSimulationApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Network Simulation")
        self.canvas = Canvas(root, width=800, height=400)
        self.canvas.pack()
        self.network = Network()
        self.dht_table_window = None

        # Додавання 20 вузлів в мережу, розташованих рівномірно на колі з
випадковими номерами
        num_nodes = 20
        circle_center_x = 400
        circle_center_y = 200
        circle_radius = 150
        angles = [2 * math.pi * i / num_nodes for i in range(num_nodes)]
        random.shuffle(angles)

        def create_kbucket_links(self, node, k=3):
            # Визначаємо список сусідів в порядку збільшення їх хеш-значень
            sorted_neighbors = sorted(self.network.nodes, key=lambda neighbor:
neighbor.hash)

            # Знайдемо позицію поточного вузла у відсортованому списку
            index = sorted_neighbors.index(node)

            # Додамо зв'язки з k ближніми сусідами з обох сторін (крім себе)
            for i in range(1, k + 1):
                if index - i >= 0:
                    self.network.add_edge(node, sorted_neighbors[index - i])
                if index + i < len(sorted_neighbors):
                    self.network.add_edge(node, sorted_neighbors[index + i])

        # Додавання зв'язків між вузлами з врахуванням k-бакетів
        for i, angle in enumerate(angles):
            x = circle_center_x + circle_radius * math.cos(angle)
            y = circle_center_y + circle_radius * math.sin(angle)
            hash_value = f"hash{i + 1}"
            node = NetworkNode(i + 1, x, y, hash_value)
            self.network.add_node(node)
            create_kbucket_links(self, node)



        self.network.add_edge(self.network.nodes[-1], self.network.nodes[0])  #
Замкнути коло

        self.start_node_id_entry = ttk.Entry(root, width=5)
        self.start_node_id_entry.insert(0, "1")
        self.start_node_id_entry.pack()
```

```
        self.target_hash_entry = ttk.Entry(root, width=10)
        self.target_hash_entry.insert(0, "hash20")
        self.target_hash_entry.pack()

        self.start_button = ttk.Button(root, text="Start Simulation",
command=self.start_simulation)
        self.start_button.pack()

        self.clear_button = ttk.Button(root, text="Clear",
command=self.clear_lines)
        self.clear_button.pack()

        self.dht_label = ttk.Label(root, text="Table:")
        self.dht_label.pack()

        self.dht_listbox = tk.Listbox(root, width=40, height=10)
        self.dht_listbox.pack()

    def start_simulation(self):
        start_node_id = int(self.start_node_id_entry.get())
        target_hash = self.target_hash_entry.get()

        start_node = self.network.find_node_by_id(start_node_id)

        if start_node:
            result = kademia_algorithm(self.network, start_node, target_hash,
self)
            if result:
                self.visualize_path(result)
            else:
                print("Шлях не знайдено.")
        else:
            print("Початковий вузол не знайдено.")

    def visualize_path(self, path):
        self.canvas.delete("all")  # Очистити всі малюнки на канвасі

        # Малювання вузлів мережі
        for node in self.network.nodes:
            fill_color = "green" if node in path else "blue"
            if node.id == path[0].id:
                fill_color = "red"  # Перший вузол червоний
            elif node.id == path[-1].id:
                fill_color = "green"  # Останній вузол зелений
            self.canvas.create_oval(node.x - 10, node.y - 10, node.x + 10,
node.y + 10, fill=fill_color)
            self.canvas.create_text(node.x, node.y - 20, text=str(node.id))

        # # Малювання шляху
        for i in range(len(path) - 1):
            node1 = path[i]
            node2 = path[i + 1]
            self.canvas.create_line(node1.x, node1.y, node2.x, node2.y,
fill="green", width=2, arrow=tk.LAST, arrowshape=(20, 25, 5))

      # Малювання вузлів мережі
        for node in self.network.nodes:
            fill_color = "yellow" if node in path else "blue"
            if node.id == path[0].id:
                fill_color = "red"  # Перший вузол червоний
            elif node.id == path[-1].id:
                fill_color = "green"  # Останній вузол зелений
            self.canvas.create_oval(node.x - 10, node.y - 10, node.x + 10,
node.y + 10, fill=fill_color)
            self.canvas.create_text(node.x, node.y - 20, text=str(node.id))


    def clear_lines(self):
```

```
            if self.dht_table_window:
                self.dht_table_window.clear_table()

        def add_entry(self, node, entry):
            self.dht_listbox.insert(tk.END, f"NetworkNode {node.id}: {entry}")

        def show_dht_table(self, node):
            if self.dht_table_window:
                self.dht_table_window.destroy()
            self.dht_table_window = DHTTable(tk.Toplevel())
            dht_table = node.get_dht_table()
            self.dht_table_window.show_table(node)

    if __name__ == "__main__":
        root = tk.Tk()
        app = NetworkSimulationApp(root)
        root.mainloop()
```

Файл protocol.py – протокол роботи мережі

```
import random
import asyncio
import logging

from rpcudp.protocol import RPCProtocol

from kademlia.node import NetworkNode
from kademlia.routing import RoutingTable
from kademlia.utils import digest

log = logging.getLogger(__name__)  # pylint: disable=invalid-name

class KademliaProtocol(RPCProtocol):
    def __init__(self, source_node, storage, ksize):
        RPCProtocol.__init__(self)
        self.router = RoutingTable(self, ksize, source_node)
        self.storage = storage
        self.source_node = source_node

    def get_refresh_ids(self):
        """
        Отримати ідентифікатори для пошуку для оновлення старих бакетів.
        """
        ids = []
        for bucket in self.router.lonely_buckets():
            rid = random.randint(*bucket.range).to_bytes(20, byteorder='big')
            ids.append(rid)
        return ids

    def rpc_stun(self, sender):  # pylint: disable=no-self-use
        return sender

    def rpc_ping(self, sender, nodeid):
        source = NetworkNode(nodeid, sender[0], sender[1])
        self.welcome_if_new(source)
        return self.source_node.id

    def rpc_store(self, sender, nodeid, key, value):
        source = NetworkNode(nodeid, sender[0], sender[1])
        self.welcome_if_new(source)
        log.debug("отримано запит на зберігання від %s, зберігаємо '%s'='%s'",
                  sender, key.hex(), value)
        self.storage[key] = value
        return True

    def rpc_find_node(self, sender, nodeid, key):
        log.info("пошук сусідів для %i в локальній таблиці",
                 int(nodeid.hex(), 16))
        source = NetworkNode(nodeid, sender[0], sender[1])
        self.welcome_if_new(source)
        node = NetworkNode(key)
        neighbors = self.router.find_neighbors(node, exclude=source)
        return list(map(tuple, neighbors))

    def rpc_find_value(self, sender, nodeid, key):
        source = NetworkNode(nodeid, sender[0], sender[1])
        self.welcome_if_new(source)
        value = self.storage.get(key, None)
        if value is None:
            return self.rpc_find_node(sender, nodeid, key)
        return {'value': value}

    async def call_find_node(self, node_to_ask, node_to_find):
```

```
        address = (node_to_ask.ip, node_to_ask.port)
        result = await self.find_node(address, self.source_node.id,
                                      node_to_find.id)
        return self.handle_call_response(result, node_to_ask)

    async def call_find_value(self, node_to_ask, node_to_find):
        address = (node_to_ask.ip, node_to_ask.port)
        result = await self.find_value(address, self.source_node.id,
                                       node_to_find.id)
        return self.handle_call_response(result, node_to_ask)

    async def call_ping(self, node_to_ask):
        address = (node_to_ask.ip, node_to_ask.port)
        result = await self.ping(address, self.source_node.id)
        return self.handle_call_response(result, node_to_ask)

    async def call_store(self, node_to_ask, key, value):
        address = (node_to_ask.ip, node_to_ask.port)
        result = await self.store(address, self.source_node.id, key, value)
        return self.handle_call_response(result, node_to_ask)

    def welcome_if_new(self, node):
        """
        Для нового вузла надсилайте йому всі ключі/значення, які він повинен
зберігати,
        а потім додайте його до таблиці маршрутизації.

        @param node: Новий вузол, який тільки що приєднався (або про якого ми
тільки що дізналися).
        """
        if not self.router.is_new_node(node):
            return

        log.info("раніше не бачили %s, додаємо в таблицю маршрутизації", node)
        for key, value in self.storage.items():
            keynode = NetworkNode(digest(key))
            neighbors = self.router.find_neighbors(keynode)
            if neighbors:
                last = neighbors[-1].distance_to(keynode)
                new_node_close = node.distance_to(keynode) < last
                first = neighbors[0].distance_to(keynode)
                this_closest = self.source_node.distance_to(keynode) < first
            if not neighbors or (new_node_close and this_closest):
                asyncio.ensure_future(self.call_store(node, key, value))
        self.router.add_contact(node)

    def handle_call_response(self, result, node):
        """
        Якщо ми отримали відповідь, додайте вузол до таблиці маршрутизації. Якщо
        ми не отримали відповідь, переконайтеся, що він видаляється з таблиці
маршрутизації.
        """
        if not result[0]:
            log.warning("немає відповіді від %s, видаляємо з таблиці
маршрутизації", node)
            self.router.remove_contact(node)
            return result

        log.info("отримали успішну відповідь від %s", node)
        self.welcome_if_new(node)
        return result
```

Файл NetworkNode.py – представлення вузла мережі

```
from operator import itemgetter
import heapq


class NetworkNode:
    """
    Простий об'єкт, що інкапсулює поняття Вузла (мінімально ID, але можливо
також IP і порт, якщо це представляє собою вузол в мережі).
    Зазвичай цей клас не слід створювати безпосередньо, оскільки він є
низькорівневою конструкцією, в основному використовуваною маршрутизатором.
    """
    def __init__(self, node_id, ip=None, port=None):
        """
        Створити екземпляр Вузла.

        Args:
            node_id (int): Значення від 0 до 2^160
            ip (string): Необов'язкова IP-адреса, де знаходиться цей Вузол
            port (int): Необов'язковий порт для цього Вузла (встановлюється,
коли встановлено IP)
        """
        self.id = node_id  # pylint: disable=invalid-name
        self.ip = ip  # pylint: disable=invalid-name
        self.port = port
        self.long_id = int(node_id.hex(), 16)

    def same_home_as(self, node):
        return self.ip == node.ip and self.port == node.port

    def distance_to(self, node):
        """
        Отримати відстань між цим вузлом та іншим.
        """
        return self.long_id ^ node.long_id

    def __iter__(self):
        """
        Дозволяє використовувати Вузол як кортеж - наприклад, tuple(node)
працює.
        """
        return iter([self.id, self.ip, self.port])

    def __repr__(self):
        return repr([self.long_id, self.ip, self.port])

    def __str__(self):
        return "%s:%s" % (self.ip, str(self.port))


class NetworkNodeHeap:
    """
    Купа вузлів, впорядкованих за відстанню до заданого вузла.
    """
    def __init__(self, node, maxsize):
        """
        Конструктор.

        @param вузол: Вузол для вимірювання всіх відстаней від нього.
        @param максимальний_розмір: Максимальний розмір, до якого може зрости ця
купа.
        """
        self.node = node
        self.heap = []
```

```
        self.contacted = set()
        self.maxsize = maxsize

    def remove(self, peers):
        """
        Видалити список ідентифікаторів однолітків з цієї купи.
        Хоча ця купа має постійний видимий розмір (на основі ітератора),
        її фактичний розмір може бути значно більшим, ніж те, що відображено.
        Тому видалення вузлів може не змінити видимий розмір, оскільки раніше
        додані вузли раптово стають видимими.
        """
        peers = set(peers)
        if not peers:
            return
        nheap = []
        for distance, вузол in self.heap:
            if в node.id not in peers:
                heapq.heappush(nheap, (distance, node))
        self.heap = nheap

    def get_node(self, node_id):
        for _, node in self.heap:
            if node.id == node_id:
                return node
        return None

    def have_contacted_all(self):
        return len(self.get_uncontacted()) == 0

    def get_ids(self):
        return [n.id for n in self]

    def mark_contacted(self, вузол):
        self.contacted.add(вузол.id)

    def popleft(self):
        return heapq.heappop(self.heap)[1] if self else None

    def push(self, nodes):
        """
        Додати вузли в купу.

        @param вузли: Це може бути один елемент або список.
        """
        if not isinstance(вузли, list):
            nodes = [nodes]

        for вузол in вузли:
            if вузол not in self:
                відстань = self.node.distance_to(вузол)
                heapq.heappush(self.heap, (distance, node))

    def __len__(self):
        return min(len(self.heap), self.maxsize)

    def __iter__(self):
        вузли = heapq.nsmallest(self.maxsize, self.heap)
        return iter(map(itemgetter(1), nodes))

    def __contains__(self, node):
        for _, other in self.heap:
            if node.id == other.id:
                return True
        return False

    def get_uncontacted(self):
        return [n for n in self if n.id not in self.contacted]
```

Файл networkP2P.py – представлення одноранговоі мережі

```
"""
Пакет для взаємодії на високому рівні в мережі.
"""
import random
import pickle
import asyncio
import logging

from kademlia.protocol import KademliaProtocol
from kademlia.utils import digest
from kademlia.storage import ForgetfulStorage
from kademlia.node import NetworkNode
from kademlia.searchX import ValueSpiderCrawl
from kademlia.searchX import NetworkNodeSpiderCrawl

log = logging.getLogger(__name__)  # pylint: disable=invalid-name


# pylint: disable=too-many-instance-attributes
class Server:
    """
    Високорівневе представлення екземпляра вузла. Це об'єкт, який слід створити,
    щоб почати прослуховування як активний вузол в мережі.
    """

    protocol_class = KademliaProtocol

    def __init__(self, ksize=20, alpha=3, node_id=None, storage=None):
        """
        Створіть екземпляр сервера. Це почне прослуховування на вказаному порту.

        Args:
            ksize (int): Параметр k
            alpha (int): Параметр alpha
            node_id: Ідентифікатор цього вузла в мережі.
            storage: Екземпляр, що реалізує інтерфейс
                 :class:`~kademlia.storage.IStorage`
        """
        self.ksize = ksize
        self.alpha = alpha
        self.storage = storage or ForgetfulStorage()
        self.node = NetworkNode(node_id or digest(random.getrandbits(255)))
        self.transport = None
        self.protocol = None
        self.refresh_loop = None
        self.save_state_loop = None

    def stop(self):
        if self.transport is not None:
            self.transport.close()

        if self.refresh_loop:
            self.refresh_loop.cancel()

        if self.save_state_loop:
            self.save_state_loop.cancel()

    def _create_protocol(self):
        return self.protocol_class(self.node, self.storage, self.ksize)

    async def listen(self, port, interface='0.0.0.0'):
        """
        Почніть слухати на вказаному порту.
```

```
        Надайте інтерфейс = "::", щоб приймати адреси ipv6
        """
        loop = asyncio.get_event_loop()
        listen = loop.create_datagram_endpoint(self._create_protocol,
                                               local_addr=(interface, port))
        log.info("Вузол %i слухає на %s:%i",
                 self.node.long_id, interface, port)
        self.transport, self.protocol = await listen
        # нарешті, заплануйте оновлення таблиці
        self.refresh_table()

    def refresh_table(self, interval=3600):
        log.debug("Оновлення маршрутної таблиці")
        asyncio.ensure_future(self._refresh_table())
        loop = asyncio.get_event_loop()
        self.refresh_loop = loop.call_later(interval, self.refresh_table)

    async def _refresh_table(self):
        """
        Оновити бакети, які не мали жодного запиту на останню годину
        """
        results = []
        for node_id in self.protocol.get_refresh_ids():
            node = NetworkNode(node_id)
            nearest = self.protocol.router.find_neighbors(node, self.alpha)
            spider = NetworkNodeSpiderCrawl(self.protocol, node, nearest,
                                            self.ksize, self.alpha)
            results.append(spider.find())

        # виконати наш пошукХ
        await asyncio.gather(*results)

        # тепер публікуємо ключі старше однієї години
        for dkey, value in self.storage.iter_older_than(3600):
            await self.set_digest(dkey, value)

    def bootstrappable_neighbors(self):
        """
        Отримайте :class:`list` пар (ip, port) :class:`tuple`, які підходять для
        використання як аргумент методу bootstrap.

        Сервер вже повинен був отримати завантаження,
        це просто утиліта для отримання деяких сусідів і потім
        зберігання їх, якщо цей сервер буде вимкнутий на деякий час.
        Коли він повернеться, список вузлів можна використовувати для
завантаження.
        """
        neighbors = self.protocol.router.find_neighbors(self.node)
        return [tuple(n)[-2:] for n in neighbors]

    async def bootstrap(self, addrs):
        """
        Завантажте сервер, підключившись до інших відомих вузлів в мережі.

        Args:
            addrs: Список пар (ip, port) у вигляді `tuple`. Зверніть увагу,
                   що приймаються лише IP-адреси - імена хостів спричинять
помилку.
        """
        log.debug("Спроба завантажити вузол з %i початкових контактів",
                  len(addrs))
        cos = list(map(self.bootstrap_node, addrs))
        gathered = await asyncio.gather(*cos)
        nodes = [node for node in gathered if node is not None]
        spider = NetworkNodeSpiderCrawl(self.protocol, self.node, nodes,
                                        self.ksize, self.alpha)
        return await spider.find()
```

```
    async def bootstrap_node(self, addr):
        result = await self.protocol.ping(addr, self.node.id)
        return NetworkNode(result[1], addr[0], addr[1]) if result[0] else None

    async def get(self, key):
        """
        Отримати ключ, якщо він є в мережі.

        Returns:
            :class:`None`, якщо не знайдено, значення інакше.
        """
        log.info("Пошук ключа %s", key)
        dkey = digest(key)
        # якщо цей вузол має його, поверніть його
        if self.storage.get(dkey) is not None:
            return self.storage.get(dkey)
        node = NetworkNode(dkey)
        nearest = self.protocol.router.find_neighbors(node)
        if not nearest:
            log.warning("Немає відомих сусідів для отримання ключа %s", key)
            return None
        spider = ValueSpiderCrawl(self.protocol, node, nearest,
                                  self.ksize, self.alpha)
        return await spider.find()

    async def set(self, key, value):
        """
        Встановіть вказаний ключ рядка на вказане значення в мережі.
        """
        if not check_dht_value_type(value):
            raise TypeError(
                "Значення повинно бути типу int, float, bool, str або bytes"
            )
        log.info("встановлення '%s' = '%s' в мережі", key, value)
        dkey = digest(key)
        return await self.set_digest(dkey, value)

    async def set_digest(self, dkey, value):
        """
        Встановіть вказаний ключ SHA1 digest (байти) на вказане значення в
мережі.
        """
        node = NetworkNode(dkey)

        nearest = self.protocol.router.find_neighbors(node)
        if not nearest:
            log.warning("Немає відомих сусідів для встановлення ключа %s",
                        dkey.hex())
            return False

        spider = NetworkNodeSpiderCrawl(self.protocol, node, nearest,
                                        self.ksize, self.alpha)
        nodes = await spider.find()
        log.info("встановлення '%s' на %s", dkey.hex(), list(map(str, nodes)))

        # якщо цей вузол також близький, тоді зберігайте його також тут
        biggest = max([n.distance_to(node) for n in nodes])
        if self.node.distance_to(node) < biggest:
            self.storage[dkey] = value
        results = [self.protocol.call_store(n, dkey, value) for n in nodes]
        # повертайте true, лише якщо принаймні один виклик збереження був
успішним
        return any(await asyncio.gather(*results))

    def save_state(self, fname):
        """
        Збережіть стан цього вузла (alpha/ksize/id/негайні сусіди)
        у файл кеша з вказаним ім'ям fname.
        """
```

```
        log.info("Збереження стану в %s", fname)
        data = {
            'ksize': self.ksize,
            'alpha': self.alpha,
            'id': self.node.id,
            'neighbors': self.bootstrappable_neighbors()
        }
        if not data['neighbors']:
            log.warning("Немає відомих сусідів, тому не записуємо в кеш.")
            return
        with open(fname, 'wb') as file:
            pickle.dump(data, file)

    @classmethod
    async def load_state(cls, fname, port, interface='0.0.0.0'):
        """
        Завантажте стан цього вузла (alpha/ksize/id/негайні сусіди)
        із файлу кеша з вказаним ім'ям fname, а потім завантажте вузол
        (використовуючи вказаний порт/інтерфейс для
прослуховування/завантаження).
        """
        log.info("Завантаження стану з %s", fname)
        with open(fname, 'rb') as file:
            data = pickle.load(file)
        svr = cls(data['ksize'], data['alpha'], data['id'])
        await svr.listen(port, interface)
        if data['neighbors']:
            await svr.bootstrap(data['neighbors'])
        return svr

    def save_state_regularly(self, fname, frequency=600):
        """
        Зберігайте стан вузла з заданою регулярністю у вказаному файлі.

        Args:
            fname: Ім'я файлу для регулярного зберігання
            frequency: Частота в секундах, з якою стан повинен бути збережений.
                       За замовчуванням, 10 хвилин.
        """
        self.save_state(fname)
        loop = asyncio.get_event_loop()
        self.save_state_loop = loop.call_later(frequency,
                                               self.save_state_regularly,
                                               fname,
                                               frequency)


def check_dht_value_type(value):
    """
    Перевірте, чи тип значення є дійсним типом для
    розміщення в DHT.
    """
    typeset = [
        int,
        float,
        bool,
        str,
        bytes
    ]
    return type(value) in typeset  # pylint: disable=unidiomatic-typecheck
```

Файл routing.py – маршрутизація в одноранговій мережі

```
import heapq
import time
import operator
import asyncio

from itertools import chain
from collections import OrderedDict
from kademlia.utils import shared_prefix, bytes_to_bit_string

class KBucket:
    def __init__(self, rangeLower, rangeUpper, ksize,
replacementNetworkNodeFactor=5):
        self.range = (rangeLower, rangeUpper)
        self.nodes = OrderedDict()
        self.replacement_nodes = OrderedDict()
        self.touch_last_updated()
        self.ksize = ksize
        self.max_replacement_nodes = self.ksize * replacementNetworkNodeFactor

    def touch_last_updated(self):
        self.last_updated = time.monotonic()

    def get_nodes(self):
        return list(self.nodes.values())

    def split(self):
        midpoint = (self.range[0] + self.range[1]) // 2
        one = KBucket(self.range[0], midpoint, self.ksize)
        two = KBucket(midpoint + 1, self.range[1], self.ksize)
        nodes = chain(self.nodes.values(), self.replacement_nodes.values())
        for node in nodes:
            bucket = one if node.long_id <= midpoint else two
            bucket.add_node(node)

        return (one, two)

    def remove_node(self, node):
        if node.id in self.replacement_nodes:
            del self.replacement_nodes[node.id]

        if node.id in self.nodes:
            del self.nodes[node.id]

            if self.replacement_nodes:
                newnode_id, newnode = self.replacement_nodes.popitem()
                self.nodes[newnode_id] = newnode

    def has_in_range(self, node):
        return self.range[0] <= node.long_id <= self.range[1]

    def is_new_node(self, node):
        return node.id not in self.nodes

    def add_node(self, node):
        """
        Додайте вузол C{NetworkNode} до C{KBucket}. Повертає True, якщо успішно,
        False, якщо бакет повний.

        Якщо бакет повний, відстежуйте вузол у списку заміщення,
        """
        if node.id in self.nodes:
            del self.nodes[node.id]
            self.nodes[node.id] = node
```

```
            elif len(self) < self.ksize:
                self.nodes[node.id] = node
            else:
                if node.id in self.replacement_nodes:
                    del self.replacement_nodes[node.id]
                self.replacement_nodes[node.id] = node
                while len(self.replacement_nodes) > self.max_replacement_nodes:
                    self.replacement_nodes.popitem(last=False)
                return False
            return True

        def depth(self):
            vals = self.nodes.values()
            sprefix = shared_prefix([bytes_to_bit_string(n.id) for n in vals])
            return len(sprefix)

        def head(self):
            return list(self.nodes.values())[0]

        def __getitem__(self, node_id):
            return self.nodes.get(node_id, None)

        def __len__(self):
            return len(self.nodes)

    class TableTraverser:
        def __init__(self, table, startNetworkNode):
            index = table.get_bucket_for(startNetworkNode)
            table.buckets[index].touch_last_updated()
            self.current_nodes = table.buckets[index].get_nodes()
            self.left_buckets = table.buckets[:index]
            self.right_buckets = table.buckets[(index + 1):]
            self.left = True

        def __iter__(self):
            return self

        def __next__(self):
            """
            Вибірка елемента з лівого піддерева, потім правого, потім лівого, і т.
д.
            """
            if self.current_nodes:
                return self.current_nodes.pop()

            if self.left and self.left_buckets:
                self.current_nodes = self.left_buckets.pop().get_nodes()
                self.left = False
                return next(self)

            if self.right_buckets:
                self.current_nodes = self.right_buckets.pop(0).get_nodes()
                self.left = True
                return next(self)

            raise StopIteration

    class RoutingTable:
        def __init__(self, protocol, ksize, node):
            """
            @param node: Вузол, який представляє цей сервер. Він не буде
            доданий до таблиці маршрутизації, але пізніше буде потрібний
            для визначення, які бакети розщеплювати або ні.
            """
            self.node = node
            self.protocol = protocol
            self.ksize = ksize
            self.flush()
```

```
def flush(self):
    self.buckets = [KBucket(0, 2 ** 160, self.ksize)]

def split_bucket(self, index):
    one, two = self.buckets[index].split()
    self.buckets[index] = one
    self.buckets.insert(index + 1, two)

def lonely_buckets(self):
    """
    Отримайте всі бакети, які не оновлювалися протягом понад години.
    """
    hrago = time.monotonic() - 3600
    return [b for b in self.buckets if b.last_updated < hrago]

def remove_contact(self, node):
    index = self.get_bucket_for(node)
    self.buckets[index].remove_node(node)

def is_new_node(self, node):
    index = self.get_bucket_for(node)
    return self.buckets[index].is_new_node(node)

def add_contact(self, node):
    index = self.get_bucket_for(node)
    bucket = self.buckets[index]

    # це буде успішним, якщо бакет повний
    if bucket.add_node(node):
        return

    # розділити бакет,
    # якщо в бакеті знаходиться вузол
    # в його діапазоні або якщо глибина не конгруентна 0 mod 5
    if bucket.has_in_range(self.node) or bucket.depth() % 5 != 0:
        self.split_bucket(index)
        self.add_contact(node)
    else:
        asyncio.ensure_future(self.protocol.call_ping(bucket.head()))

def get_bucket_for(self, node):
    """
    Отримайте індекс бакету, в який попаде даний вузол.
    """
    for index, bucket in enumerate(self.buckets):
        if node.long_id < bucket.range[1]:
            return index
    return None

def find_neighbors(self, node, k=None, exclude=None):
    k = k or self.ksize
    nodes = []
    for neighbor in TableTraverser(self, node):
        notexcluded = exclude is None or not neighbor.same_home_as(exclude)
        if neighbor.id != node.id and notexcluded:
            heapq.heappush(nodes, (node.distance_to(neighbor), neighbor))
        if len(nodes) == k:
            break

    return list(map(operator.itemgetter(1), heapq.nsmallest(k, nodes)))
```

Файл searchX.py – пошук даних в одноранговій мережі

```
from collections import Counter
import logging

from kademlia.node import NetworkNode, NetworkNodeHeap
from kademlia.utils import gather_dict

log = logging.getLogger(__name__)  # pylint: disable=invalid-name

# pylint: disable=too-few-public-methods
class SpiderCrawl:
    """
    Пошук в мережі і пошук даних за 160-бітними ключами.
    """
    def __init__(self, protocol, node, peers, ksize, alpha):
        """
        Створіть нового C{SpiderCrawl}er.

        Args:
            protocol: Екземпляр :class:`~kademlia.protocol.KademliaProtocol`.
            node: :class:`~kademlia.node.NetworkNode`, що представляє ключ, який
ми
                  шукаємо.
            peers: Список екземплярів :class:`~kademlia.node.NetworkNode`, які
                   надають вхідну точку до мережі.
            ksize: Значення k згідно з папером.
            alpha: Значення alpha згідно з папером.
        """
        self.protocol = protocol
        self.ksize = ksize
        self.alpha = alpha
        self.node = node
        self.nearest = NetworkNodeHeap(self.node, self.ksize)
        self.last_ids_crawled = []
        log.info("створення пошуку з вузлами-першоджерелами: %s", peers)
        self.nearest.push(peers)

    async def _find(self, rpcmethod):
        """
        Отримати або значення або список вузлів.

        Args:
            rpcmethod: Протокол callfindValue або call_find_node.

        Процес:
          1. викликає find_* для поточних ALPHA найближчих вузлів,
             які ще не були запитані, додаючи результати до поточного списку
             найближчих списків k вузлів.
          2. поточний список найближчих потрібно відстежувати, хто був
             запитаний вже, сортувати за найближчими, зберігати KSIZE.
          3. якщо список такий самий, як і минулого разу, наступний виклик
             повинен бути до всіх, хто ще не був запитаний.
          4. повторюйте, якщо поточний список найближчих був запитаний всім,
             тоді ви завершили.
        """
        log.info("пошук в мережі з найближчими: %s", str(tuple(self.nearest)))
        count = self.alpha
        if self.nearest.get_ids() == self.last_ids_crawled:
            count = len(self.nearest)
        self.last_ids_crawled = self.nearest.get_ids()

        dicts = {}
        for peer in self.nearest.get_uncontacted()[:count]:
            dicts[peer.id] = rpcmethod(peer, self.node)
```

```
            self.nearest.mark_contacted(peer)
        found = await gather_dict(dicts)
        return await self._nodes_found(found)

    async def _nodes_found(self, responses):
        raise NotImplementedError


class ValueSpiderCrawl(SpiderCrawl):
    def __init__(self, protocol, node, peers, ksize, alpha):
        SpiderCrawl.__init__(self, protocol, node, peers, ksize, alpha)
        # Відстежувати єдиний найближчий вузол без значення - щоб встановити
ключ там, якщо знайдено
        self.nearest_without_value = NetworkNodeHeap(self.node, 1)

    async def find(self):
        """
        Знайти або найближчі вузли, або запитане значення.
        """
        return await self._find(self.protocol.call_find_value)

    async def _nodes_found(self, responses):
        """
        Обробити результат ітерації в _find.
        """
        toremove = []
        found_values = []
        for peerid, response in responses.items():
            response = RPCFindResponse(response)
            if not response.happened():
                toremove.append(peerid)
            elif response.has_value():
                found_values.append(response.get_value())
            else:
                peer = self.nearest.get_node(peerid)
                self.nearest_without_value.push(peer)
                self.nearest.push(response.get_node_list())
        self.nearest.remove(toremove)

        if found_values:
            return await self._handle_found_values(found_values)
        if self.nearest.have_contacted_all():
            # не знайдено!
            return None
        return await self.find()

    async def _handle_found_values(self, values):
        """
        Ми отримали деякі значення. Але переконаймося,
        що вони всі однакові або трохи різні.
        """
        value_counts = Counter(values)
        if len(value_counts) != 1:
            log.warning("Отримано кілька значень для ключа %i: %s",
                        self.node.long_id, str(values))
        value = value_counts.most_common(1)[0][0]

        peer = self.nearest_without_value.popleft()
        if peer:
            await self.protocol.call_store(peer, self.node.id, value)
        return value


class NetworkNodeSpiderCrawl(SpiderCrawl):
    async def find(self):
        """
        Знайдіть найближчі вузли.
        """
        return await self._find(self.protocol.call_find_node)
```

```
    async def _nodes_found(self, responses):
        """
        Обробити результат ітерації в _find.
        """
        toremove = []
        for peerid, response in responses.items():
            response = RPCFindResponse(response)
            if not response.happened():
                toremove.append(peerid)
            else:
                self.nearest.push(response.get_node_list())
        self.nearest.remove(toremove)

        if self.nearest.have_contacted_all():
            return list(self.nearest)
        return await self.find()


class RPCFindResponse:
    def __init__(self, response):
        """
        Обгортка для результату пошуку RPC.

        Args:
            response: Це буде кортеж (<відповідь, отримана>, <значення>)
                      де <значення> буде списком кортежів, якщо не знайдено
                      або словником {'value': v}, де v - бажане значення
        """
        self.response = response

    def happened(self):
        """
        Чи відповів інший хост насправді?
        """
        return self.response[0]

    def has_value(self):
        return isinstance(self.response[1], dict)

    def get_value(self):
        return self.response[1]['value']

    def get_node_list(self):
        """
        Отримати список вузлів у відповіді. Якщо значення відсутнє, його
        треба встановити.
        """
        nodelist = self.response[1] or []
        return [NetworkNode(*nodeple) for nodeple in nodelist]
```

Файл storage.py – сховище даних в одноранговій мережі

```
import time
from itertools import takewhile
import operator
from collections import OrderedDict
from abc import abstractmethod, ABC


class IStorage(ABC):
    """
    Локальне сховище для цього вузла.
    Реалізації IStorage методу get повинні повертати той самий тип, що і
встановлюється за допомогою set.
    """

    @abstractmethod
    def __setitem__(self, key, value):
        """
        Встановити ключ на задане значення.
        """

    @abstractmethod
    def __getitem__(self, key):
        """
        Отримати заданий ключ. Якщо елемент не існує, видає C{KeyError}
        """

    @abstractmethod
    def get(self, key, default=None):
        """
        Отримати заданий ключ. Якщо не знайдено, повертає значення за
замовчуванням.
        """

    @abstractmethod
    def iter_older_than(self, seconds_old):
        """
        Повернути ітератор по кортежам (ключ, значення) для елементів старше
        заданого значення secondsOld.
        """

    @abstractmethod
    def __iter__(self):
        """
        Отримати ітератор для цього сховища, повинен повертати кортежі (ключ,
значення).
        """


class ForgetfulStorage(IStorage):
    def __init__(self, ttl=604800):
        """
        За замовчуванням, максимальний вік - тиждень.
        """
        self.data = OrderedDict()
        self.ttl = ttl

    def __setitem__(self, key, value):
        if key in self.data:
            del self.data[key]
        self.data[key] = (time.monotonic(), value)
        self.cull()

    def cull(self):
```

```
        for _, _ in self.iter_older_than(self.ttl):
            self.data.popitem(last=False)

    def get(self, key, default=None):
        self.cull()
        if key in self.data:
            return self[key]
        return default

    def __getitem__(self, key):
        self.cull()
        return self.data[key][1]

    def __repr__(self):
        self.cull()
        return repr(self.data)

    def iter_older_than(self, seconds_old):
        min_birthday = time.monotonic() - seconds_old
        zipped = self._triple_iter()
        matches = takewhile(lambda r: min_birthday >= r[1], zipped)
        return list(map(operator.itemgetter(0, 2), matches))

    def _triple_iter(self):
        ikeys = self.data.keys()
        ibirthday = map(operator.itemgetter(0), self.data.values())
        ivalues = map(operator.itemgetter(1), self.data.values())
        return zip(ikeys, ibirthday, ivalues)

    def __iter__(self):
        self.cull()
        ikeys = self.data.keys()
        ivalues = map(operator.itemgetter(1), self.data.values())
        return zip(ikeys, ivalues)
```

Файл utils.py – допоміжні функції для роботи одноранговоі мережі

```
import hashlib
import operator
import asyncio


async def gather_dict(dic):
"""
    Загальний обробник для функцій, які не мають сенсу як методи.
    Збирає результати виконання асинхронних функцій з словника та повертає
словник з результатами.
    """

    cors = list(dic.values())
    results = await asyncio.gather(*cors)
    return dict(zip(dic.keys(), results))



def digest(string):
"""
    Обчислює хеш від рядка або байтів.

    Args:
        рядок: Рядок або байти, для яких потрібно обчислити хеш.

    Returns:
        Хеш-значення об'єкту, обчислене за допомогою SHA-1.

    """
    if not isinstance(string, bytes):
        string = str(string).encode('utf8')
    return hashlib.sha1(string).digest()



def shared_prefix(args):
  """
    Знаходить спільний префікс серед рядків.

    Args:
        args: Список рядків, для яких потрібно знайти спільний префікс.

    Returns:
        Спільний префікс серед рядків.

    Наприклад:

        спільний_префікс(['blahblah', 'blahwhat'])

    повертає 'blah'.
    """
    i = 0
    while i < min(map(len, args)):
        if len(set(map(operator.itemgetter(i), args))) != 1:
            break
        i += 1
    return args[0][:i]



def bytes_to_bit_string(bites):
"""
    Конвертує байти у рядок бітів.
```

```
    Args:
        bites: Список байтів, які потрібно конвертувати.

    Returns:
        Рядок бітів, представлення байтів у бінарному форматі.
"""

    bits = [bin(bite)[2:].rjust(8, '0') for bite in bites]
    return "".join(bits)
```