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
“____” _____ 2025 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти

на тему

“Дослідження та програмна реалізація системи ієрархічного зберігання даних з використанням технології Tiered Storage”

Виконав здобувач вищої освіти
ІІ курсу, групи КН-24М
ОПП «Комп’ютерні науки»
спеціальності 122 «Комп’ютерні науки»
_____ Лобода П.А.
«____» _____ 2025 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Коваленко А.С.
«____» _____ 2025 р.
Рецензент _____
_____

м. Кропивницький

АНОТАЦІЯ

Лобода П.А. Дослідження та програмна реалізація системи ієрархічного зберігання даних з використанням технології Tiered Storage. 122 Комп'ютерні науки. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи ієрархічного зберігання даних з використанням технології Tiered Storage.

Метою розробки є дослідження та програмна реалізація системи ієрархічного зберігання даних з використанням технології Tiered Storage.

Об'єктом дослідження є процес ієрархічного зберігання даних з використанням технології Tiered Storage.

Предметом дослідження є методи ієрархічного зберігання даних з використанням технології Tiered Storage.

Методи дослідження базуються на методах великих даних, методах математичної статистики, методах розробки програмного забезпечення.

Результат роботи – програмна реалізація системи ієрархічного зберігання даних з використанням технології Tiered Storage.

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 з ОС Windows 10/11.

Програму розроблено в середовищі Python.

Ключові слова: комп'ютерні науки, зберігання даних, Tiered Storage

ABSTRACT

Loboda P.A. Research and software implementation of a hierarchical data storage system using Tiered Storage technology. 122 Computer Science. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.

In this final qualification work for the second (master's) level of higher education, software has been developed, which is intended for a hierarchical data storage system using Tiered Storage technology.

The purpose of the development is the research and software implementation of a hierarchical data storage system using Tiered Storage technology.

The object of the research is the process of hierarchical data storage using Tiered Storage technology.

The subject of the research is methods of hierarchical data storage using Tiered Storage technology.

The research methods are based on big data methods, methods of mathematical statistics, and methods of software development.

The result of the work is a software implementation of a hierarchical data storage system using Tiered Storage technology.

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

A user-friendly user interface has been developed. Instructions for working with the software are provided.

The program can be used on a PC with Windows 10/11.

The program is developed in the Python environment.

Keywords: computer science, data storage, Tiered Storage

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

LZW – Алгоритм Зіва-Лемпела

ВСТУП

Актуальність теми. Програмне забезпечення, також відоме як багаторівневе сховище, було розроблено для розподілених серверних середовищ, щоб автоматизувати процес ідентифікації холодних наборів даних та автоматичної міграції їх з основного диска на менш дорогі оптичні та стрічкові пристрої зберігання даних. Повертаючись до ери мейнфреймів, HSM також мав автоматично обробляти запити на відкликання файлів щоразу, коли користувач натискав на файл-заглушку.

На жаль, ці ранні продукти HSM мали низку недоліків. Ці недоліки були настільки серйозними, що HSM стало «поганим словом» серед IT-фахівців. Багато з цих IT-фахівців вважали, що єдиний життєздатний спосіб управління сховищем – це просто продовжувати додавати більше потужностей до основного рівня.

Оскільки ландшафт центрів обробки даних змінився, організації отримали широкий спектр варіантів зберігання даних, флеш-пам'ять замінила високопродуктивні фізичні дискові накопичувачі як сховища першого рівня. Високопродуктивні та звичайні фізичні жорсткі диски тепер функціонують як вторинні та третинні рівні зберігання. Доступні хмарні сховища файлів та об'єктів для обробки великих обсягів довгострокових вимог до зберігання. Всі ці опції необхідні для боротьби з неструктурованим напливом даних (а також розростанням даних та високими витратами на зберігання даних), з якими стикається більшість організацій. Однак основна проблема залишається: як автоматично виявляти «теплі» та «холодні» набори даних, а потім безперервно переносити їх на найефективніший рівень зберігання, одночасно керуючи всім життєвим циклом файлу.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи ієрархічного зберігання даних з використанням технології Tiered Storage.

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

– Огляд існуючих систем ієрархічного зберігання даних з використанням технології Tiered Storage.

– Дослідження системи ієрархічного зберігання даних з використанням технології Tiered Storage.

– Програмна реалізація системи ієрархічного зберігання даних з використанням технології Tiered Storage.

Об'єктом дослідження є процес ієрархічного зберігання даних з використанням технології Tiered Storage.

Предметом дослідження є методи ієрархічного зберігання даних з використанням технології Tiered Storage.

Методи дослідження базуються на методах великих даних, методах математичної статистики, методах розробки програмного забезпечення.

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

– Удосконалено метод ієрархічного зберігання даних з використанням технології Tiered Storage.

– Розроблено вітчизняний продукт ієрархічного зберігання даних з використанням технології Tiered Storage, який має більш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі ієрархічного зберігання даних з використанням технології Tiered Storage.

Достовірність наукових результатів підтверджена теоретичними викладеннями, даними комп'ютерного моделювання, коректними дослідженнями

параметрів на функціонуючій обчислювальній мережі, а також відповідністю отриманих результатів окремим результатам, наведеним у науковій літературі.

Робота апробована на LVII Науково-технічній конференції здобувачів вищої освіти LV науково-технічної конференції «Наука в ЦНТУ: основні досягнення та перспективи розвитку» (2025 р.), основні положення випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти надруковані у статті збірника праць молодих науковців ЦНТУ, випуск №15.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи ієрархічного зберігання даних з використанням технології Tiered Storage, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Управління неструктурованими даними – це категорія програмного забезпечення, яка з'явилася для вирішення проблеми стрімкого зростання обсягу неструктурованих даних у підприємствах та сучасної реальності гібридного хмарного сховища. У звіті Komprise 2025 про стан управління неструктурованими даними 32% організацій повідомляють, що вони керують 10 ПБ даних або більше. Це еквівалентно 110 000 фільмів надвисокої чіткості (UHD) або половині даних, що зберігаються Бібліотекою Конгресу США. Більшість (73%) організацій витрачають понад 30% свого IT-бюджету на зберігання даних.

Постачальники технологій зберігання та резервного копіювання даних зараз усвідомлюють важливість управління неструктурованими даними, оскільки дані переживають інфраструктуру, а для використання хмарного сховища даних необхідна мобільність даних.

Крім того, неструктуровані дані визнаються паливом для корпоративного штучного інтелекту. Ефективне управління ними робить це паливо зручним, надійним та економічно ефективним.

Управління неструктурованими даними має бути незалежним та агностичним від платформ зберігання даних, резервного копіювання та хмарної інфраструктури.

Існує 5 вимог до рішень для управління неструктурованими даними:

1. Повинно виходити за рамки ефективності зберігання та допомагати створювати більшу цінність даних.

2. Має бути багато направленим.

3. Більшість не порушує роботу користувачів та робочих процесів.

4. Слід створювати нові способи використання ваших даних (наприклад,

все більше посилювати ініціативи у сфері штучного інтелекту).

5. Ставте свої дані на перше місце та уникайте прив’язки до постачальника.

Рішення для керування неструктурованими даними на основі аналітики приносить цінність, аналізуючи всі дані в сховищах у локальних, хмарних та периферійних середовищах, щоб отримати глибоке розуміння. Ці знання допомагають IT-менеджерам приймати кращі рішення з урахуванням користувачів, оптимізувати витрати та зменшувати ризики безпеки та дотримання нормативних вимог. Ці дані виходять за рамки традиційних показників сховища, таких як затримка, IOPS та пропускна здатність мережі. Крім того, правильне рішення для керування неструктурованими даними повинно надавати правильні набори даних у рідному форматі для аналітичних та штучних інтелектів служб.

1.2 Область застосування

Ось деякі з нових показників, які стали можливими завдяки програмному забезпеченню для управління неструктурованими даними:

– Найважливіші власники/користувачі даних: переглядайте тенденції використання та можливі проблеми з дотриманням вимог, такі як зберігання окремими користувачами надмірної кількості відеофайлів або зберігання файлів PII у незахищеному місці.

– Поширені типи файлів: Можливість перегляду даних за розширенням файлу спрощує процес пошуку всіх файлів, пов’язаних із проектом, і може бути корисною для майбутніх дослідницьких ініціатив. Це може бути так просто, як знайти всі файли журналів, файли трасування або витяги з певної програми чи інструменту та перенести їх до озера даних для аналізу.

– Витрати на зберігання для повернення платежу або відшкодування: Незалежно від того, чи потрібні вимоги щодо повернення платежу, зацікавлені сторони повинні розуміти витрати у своєму відділі та мати можливість

переглядати показники. Це допоможе визначити сфери, де недороге зберігання або багаторівневе розміщення даних в архівному сховищі є життєздатною можливістю для зниження витрат.

– Темпи зростання даних: Високорівневі показники зростання даних дозволяють керівникам ІТ-відділів та бізнес-відділів бути в курсі справ, щоб вони могли співпрацювати над рішеннями щодо управління даними. Зрозумійте, які групи та проекти найшвидше зростають обсягом даних, і переконайтеся, що створення/зберігання даних відповідає загальному бізнес-пріоритету.

– Вік даних та моделі доступу. У більшості підприємств 60-80% даних є «холодними» та не використовувалися протягом року або довше. Метрики, що показують відсоток холодних, теплих та гарячих даних, є критично важливими для забезпечення того, щоб дані знаходилися в потрібному місці у потрібний час відповідно до їхньої бізнес-цінності, та для оптимізації витрат.

Окрім оптимізації витрат, інструменти та методи управління неструктурованими даними можуть допомогти отримати нову цінність від даних.

Неструктуровані дані – це паливо, необхідне для ШІ, проте їх важко використовувати, оскільки неструктуровані дані важко знайти, шукати та переміщувати через їхній розмір та розподіл у гібридних хмарних середовищах. Тегування та автоматизація можуть допомогти підготувати неструктуровані дані для ШІ та програм аналітики великих даних. Тактики включають:

– Попередньо обробляйте дані на периферії, щоб їх можна було проаналізувати та позначити новими метаданими перед переміщенням у хмарне озеро даних. Це може суттєво зменшити витрати та зусилля, пов'язані з переміщенням та зберіганням непотрібних даних, а також мінімізувати виникнення «болот» даних.

– Застосування автоматизації для полегшення сегментації, очищення, пошуку та збагачення даних. Ви можете зробити це за допомогою позначення даних тегами, видалення або розподілу холодних даних за політиками та переміщення даних в оптимальне сховище, звідки вони можуть бути використані

інструментами великих даних та машинного навчання. Провідним новим підходом є можливість ініціювати та виконувати робочі процеси з даними.

– Використовуйте рішення, яке зберігає теги метаданих під час переміщення даних з одного місця в інше. Наприклад, файли, позначені як такі, що містять ключові слова проекту стороннім сервісом штучного інтелекту, повинні зберігати ці теги необмежений час, щоб новій дослідницькій команді не довелося повторювати той самий аналіз – за високих витрат.

– Належним чином плануйте масштабні міграції даних, ретельно перевіряючи їх та перевіряючи. Це може запобігти поширеним проблемам із мережею та безпекою, які затримують міграцію даних та призводять до помилок або втрати даних.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи ієрархічного зберігання даних з використанням технології Tiered Storage, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

Багато компаній стикаються з проблемою адаптації своєї інфраструктури зберігання даних до необмеженого зростання обсягу даних. Питання витрат зазвичай стоїть на першому місці. Водночас критерії дуже вимогливі:

– Гнучкість: Можливо, є переваги у закупівлі всіх компонентів у одного постачальника. Однак, в принципі, доцільно уникати виключних зобов'язань перед одним виробником – так званої прив'язки до постачальника. У нашій попередній статті блогу «Незалежність постачальника та захист інвестицій» ми пояснили, як ви можете зберегти максимально можливу гнучкість у проектуванні інфраструктури зберігання даних, використовуючи стандартизовані компоненти.

– Масштабованість: Основою сталості стратегії зберігання даних є її масштабованість: вона повинна мати можливість гнучко та легко зростати разом із обсягами даних.

– Плавні робочі процеси: Користувачі в бізнес-відділах насамперед стурбовані власними робочими процесами: вони очікують швидкого та безперебійного доступу до даних. Тому незалежно від фізичного місця зберігання в інфраструктурі, інтерфейс користувача повинен забезпечувати цю безперервність.

– Архівування, що відповідає вимогам законодавства: Компаніям необхідно архівувати значну частину своїх даних у спосіб, що відповідає вимогам законодавства. Це означає, наприклад, що файли повинні зберігатися протягом певного періоду часу таким чином, щоб їх не можна було змінити або видалити. У нашій статті «Рішення для зберігання даних для архівування, що відповідає

вимогам законодавства», ми вже розглядали питання про те, як ви можете вирішити цю проблему за допомогою розумного планування архітектури сховища.

Управління життєвим циклом інформації: гарячі, теплі та холодні дані

Концепція управління життєвим циклом інформації передбачає, що дані проходять життєвий цикл з різних фаз – залежно від частоти, з якою вони використовуються.

– Створення та гаряча фаза: Відразу після створення дані є актуальними та використовуються в щоденних операціях. На цій фазі дані повинні бути швидко доступні.

– Тепла та холодна фаза: З часом своєчасність та частота доступу зменшуються, і дані спочатку стають теплими, а потім холодними.

«Гаряча фаза» більшості даних, що накопичуються в компаніях, досить коротка. Значна частина доступного простору для зберігання зазвичай займає холодні дані, навіть якщо вони майже не використовуються або не використовуються взагалі.

Спрощено кажучи, дані можна описати як активні та неактивні дані. Вважається, що в більшості компаній до 80% даних є неактивними. Аналіз інвентаризації даних та місць зберігання даних часто призводить до висновку, що швидке та дороге первинне сховище перевантажене неактивними даними – іншими словами, холодними даними, для яких не потрібне особливо високопродуктивне сховище.

Отже, настав час оптимізувати інфраструктуру зберігання даних та управління ними. Тут є великий потенціал для підвищення ефективності та економії коштів.

Ієрархічне управління сховищами: оптимізація інфраструктури

Для відображення концепції управління життєвим циклом інформації можна використовувати багаторівневу архітектуру сховища. Ця архітектура

сховища складається з кількох рівнів, на яких можна використовувати технології зберігання з різними властивостями.

– Для активних даних використовується високопродуктивне, але дороге первинне сховище.

– Ці системи зберігання поєднуються з дешевшим, але повільнішим вторинним сховищем, яке призначене для холодних даних.

Програмне забезпечення для керування даними та сховищами автоматично переміщує дані на рівень сховища, який відповідає частоті доступу та використання в кожному випадку. Таке переміщення між різними рівнями сховища називається багаторівневим розподілом:

– Дані, які більше не використовуються протягом певного періоду часу, можна перемістити на нижчий рівень сховища.

– Первинне сховище розвантажується та використовується лише для даних, до яких потрібен швидкий доступ.

Щодо вибору компонентів та послуг зберігання, гнучкість має вирішальне значення, тобто: ви повинні мати можливість вільно вибирати системи зберігання для відповідного застосування – без прив'язки до постачальника та можливих пов'язаних з цим обмежень. Саме тут вступає в гру стандартизація ключових слів:

– Використовуючи стандартизовані інтерфейси, протоколи, формати тощо, ви можете комбінувати системи зберігання даних відповідно до ваших індивідуальних потреб.

– Існуючі системи та наявні інвестиції можуть продовжувати використовуватися.

– Зі збільшенням обсягів даних, ємність можна гнучко адаптувати, додаючи додаткові компоненти або розширення послуг.

Сумісне архівування

В рамках такого ієрархічного управління сховищами (HSM) можна легко реалізувати юридично сумісне архівування важливих даних компанії. Для цієї мети використовується рівень архівного сховища. В принципі, архівування

відбувається наступним чином:

– Програмне забезпечення автоматично переміщує дані, що підлягають архівуванню, з основного сховища на архівний рівень сховища на основі визначених користувачем правил. Там програмне забезпечення забезпечує дотримання правових вимог щодо архівування.

– Принцип «запис одного разу, читання багато разів», захист від WORM, запобігає змінам в архівних файлах. Якщо до архівного файлу звертаються та змінюють його, створюється нова версія файлу. Оригінальний файл залишається незмінним.

– Програмне забезпечення контролює дотримання терміну зберігання за допомогою так званого управління зберіганням. Наприклад, можна запобігти видаленню файлу до закінчення заданого терміну.

Через періоди зберігання, які іноді становлять кілька десятиліть, система HSM також повинна враховувати міграцію систем зберігання даних. В ідеалі, відповідне програмне забезпечення підтримує фонову міграцію, щоб не було потреби в перериванні бізнесу.

Зберігайте якомога більше даних за найнижчою можливою ціною – та одночасно архівуйте їх у спосіб, що відповідає законодавству: Ви можете вирішити проблеми управління даними та сховищами за допомогою ієрархічної архітектури сховища, в рамках якої дані переміщуються відповідно до принципу управління життєвим циклом інформації. Якщо ви покладаєтеся на стандартизовані компоненти для цього, ви залишаєтеся гнучкими у виборі та розширеннях.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Python – високорівнева мова програмування, яку називають другою за популярністю в світі. Її використовують для розробки вебзастосунків,

програмного забезпечення, машинного навчання. Python застосовують для вирішення робочих завдань у компаніях Google, Instagram, Facebook, IBM, NASA, Dropbox, Netflix та інших. Розробники цінують цю мову програмування за простоту у вивченні, ефективність та мультиплатформність.

Python – скриптова мова програмування з досить простим синтаксисом. Для розуміння достатньо порівняти принципи написання найпростішої програми, яка виводить на екран текстове повідомлення. Саме тому мова програмування Python більш доступна для новачків, а професіонали встигли адаптувати її для вирішення великої кількості завдань. Це мультиплатформне рішення, тому знання Python дає можливість працювати у різних сферах: від розробки мобільних застосунків до ігрової індустрії та штучного інтелекту.

У мови програмування динамічна типізація: є можливість передавати до функцій будь-який тип даних без попереднього вказання. Інтерпретованість дозволяє знаходити помилки у коді ще до повної збірки у робочий застосунок. При цьому Python дуже чітко дає зрозуміти, де та через що виникла помилка.

Це мова об'єктноорієнтованого програмування (ООП). Програмне забезпечення на Python оформлене у вигляді моделей, які можуть бути зібраними у пакети. Тип та структуру кожного об’єкта можна запитати під час виконання програми. Для кожного з об’єктів можна отримати всю інформацію щодо його внутрішньої структури. Окрім того:

– у мови логічний синтаксис, завдяки чому вихідний код легко читати та розуміти;

– гнучкість та масштабованість Python дозволяє адаптувати високорівневу логіку та розширяти складні застосунки, як тільки виникне така необхідність;

– розробка на Python у більшості випадків проходить швидше, ніж на інших мовах програмування;

– Python – інтерпретована мова програмування. Це значить, що код можна написати у будь-якому текстовому файлі на будь-якій платформі, і потім успішно запустити;

– у Python – колосальна спільнота однодумців. Тож будь-які складнощі конкретних розробників вирішуються колективно.

Проте є декілька особливостей, які можна віднести до недоліків. Це повільність (ця мова програмування хоч і універсальна, проте повільніша за інші), велика кількість ресурсів, необхідних для роботи та «прив'язаність» до системних бібліотек.

Мова програмування Python використовується у наступних сферах:

1. Розробка програмних застосунків будь-якого напрямку.

2. Розробка серверної частини мобільних застосунків (найпопулярніший напрямок).

3. Ігри. Багато сучасних ігор для комп'ютерів (наприклад, World of Tanks) частково чи повністю написані на Python.

4. Вбудовані системи для різних пристроїв. Дуже часто Python використовують для написання внутрішніх платформ управління банкоматами.

5. Скрипти та плагіни до уже реалізованих програм для автоматизації процесів чи створення інших рішень.

6. Тестування (автоматизація цього процесу).

7. Машинне навчання. – основна мова для написання алгоритмів і аналітичних застосунків у сфері Machine Learning.

Бібліотеки Python

Різні бібліотеки Python використовують для виконання конкретних завдань. Наприклад, Matplotlib підходить для відображення даних у двовимірній та тривимірній графіці. Pandas підходить для зручної роботи з даними. NumPy дозволяє створювати масиви та керувати ними. Requests використовується для веброзробки. OpenCV-Python відкриває можливості для обробки зображень з метою оптимізації систем «машинного зору».

Найвідоміші фреймворки для мови програмування Python

Фреймворки Python допомагають створити зручне та функціональне середовище для розробки. У них міститься набір інструментів, модулів та

бібліотек, корисних для виконання конкретних завдань. Це значно полегшує роботу: наприклад, дає змогу не витрачати час на розписування дій, які повторюються, а використати релевантний інструмент. Тож є можливість позбутися рутинних процесів та сконцентруватися на логіці проєкту.

Серед найпопулярніших фреймворків для Python:

– Django – найстаріший та найвідоміший. Створений для реалізації великих інтерактивних проєктів;

– Pyramid – зручний у налаштуваннях, і дає можливість реалізувати складні нестандартні ідеї;

– Web2py – підходить в першу чергу для вебзастосунків і може використовуватись на будь-яких архітектурах.

Популярні Python IDE

IDE або інтегровані середовища розробки – це програмне забезпечення, яке надає розробникам необхідні інструменти для написання, редагування, тестування та налаштування коду. Для розробки на Python найчастіше використовують IDE PyCharm, IDLE, Spyder та Atom.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи ієрархічного зберігання даних з використанням технології Tiered Storage.

В процесі розробки випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі.

Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Існують проблеми, пов’язані з обробкою, переміщенням, зберіганням або обробкою великих обсягів чого завгодно, і в абсолютно не пов’язаних між собою сферах ці проблеми можуть виглядати дуже схожими. Розглянемо поштову службу: для доставки листів і посилок туди, куди їм місце, потрібна складна мережа літаків, кораблів, напівпричепів, машин, транспортних засобів доставки та людей. І на кожному етапі існують величезні ризики, пов’язані з ефективністю, вартістю, затримками та помилками.

Наприклад, уявіть собі використання напівпричепа для доставки пошти від дверей до дверей і скільки місця буде витрачено у вантажівці. Або уявіть собі парк невеликих електричних поштових транспортних засобів, які перевозять пошту з Аляски до Флориди, замість використання літака.

Це спрощені приклади, але вони допомагають пояснити, чому існує ієрархічне управління сховищами (HSM). Організації з великими обсягами даних постійно стикаються з проблемами ефективності, і багато зусиль витрачається на планування того, як зберігати, переміщувати та обробляти всю цю інформацію. Ієрархічне управління сховищами (HSM) – це історичний метод, який гарантує, що організації по суті не використовуватимуть напівпричепи як засоби доставки своїх цифрових даних.

У даному розділі ми розглянемо HSM, проблеми, які він прагне вирішити, та деякі сучасні альтернативи йому.

Що таке ієрархічне управління сховищами?

Ієрархічне управління сховищем, або HSM, – це процес управління цифровими даними, метою якого є максимально економічне використання носіїв інформації, мінімізуючи при цьому неефективність використання даних.

Два ключові факти лежать в основі HSM: по-перше, різні методи зберігання цифрових носіїв мають різні характеристики. По-друге, не всі дані обробляються однаково. Щодо першого пункту, найбільш очевидною відмінністю між різними цифровими носіями інформації є вартість. Найшвидші, найдоступніші та найуніверсальніші носії інформації, як правило, є найдорожчими. А що стосується другого пункту, деякі дані використовуються щодня, тоді як деякі дані використовуються набагато рідше.

Прибуток багатьох компаній залежить від швидкого доступу до найважливіших даних. Але було б надзвичайно неефективно платити стільки ж за зберігання та доступ до даних, які вони використовують лише частку часу, з тим самим рівнем швидкості та доступності.

Різні організації впроваджують HSM по-різному – єдиного зводу правил для використання HSM не існує. Але щоразу, коли організація розділяє своє сховище даних щонайменше на два рівні, HSM – це процес встановлення правил щодо того, що де зберігається та як це переміщується.

Переваги ієрархічного управління сховищами

Організації можуть побачити низку переваг від впровадження ієрархічного управління сховищами. Економія коштів є найбільш очевидною перевагою HSM: переміщуючи менш термінові дані на дешевші носії, компанії можуть пожертвувати доступністю заради вартості. Продуктивність також загалом покращується завдяки принципам HSM. Коли програми, яким потрібен доступ до даних, не витрачають час на перегляд старих, застарілих або нерелевантних даних, вони можуть забезпечити кращі результати швидше.

І хоча HSM може здатися складним, чіткі правила визначення місця розташування різних категорій даних та їх автоматичне застосування призводять до спрощеного управління даними. HSM також оптимізує використання сховища, оскільки автоматично переносить дані на відповідний рівень сховища на основі правил, встановлених IT-фахівцями.

Як працює ієрархічне управління сховищами?

HSM може складатися з багатьох шарів носія інформації, але його суть полягає в тому, що на одному кінці знаходиться високопродуктивний рівень, а на іншому – повільніший і дешевший рівень. Високопродуктивний рівень традиційно складався з пам'яті класу сховища, твердотільних накопичувачів (SSD) корпоративного класу та високопродуктивних жорстких дисків. На нижчому рівні знаходяться такі пристрої, як оптичні диски та навіть стрічкові накопичувачі.

Фактичне впровадження політик HSM є досить складним, але HSM по суті працює, визначаючи частоту доступу до файлу, і з плином часу система автоматично переміщує рідко використовувані файли до повільнішого та дешевшого сховища. ІТ-команди пишуть правила для параметрів, які визначають, коли дані переміщуються, які дані виключені з цих правил та інші уточнення. Але HSM, як правило, є автоматизованим процесом, який оптимізує доступ до даних та витрати на їх зберігання.

Які рівні HSM?

Ключ до розуміння HSM та використання носіїв інформації, таких як стрічкові накопичувачі, полягає в тому, що колись різниця між вартістю, продуктивністю та швидкістю була надзвичайною. Хоча ці розриви зменшилися і постійно зменшуються, був час, коли різниця у вартості між оптичними дисками та твердотільними накопичувачами була достатньо суттєвою, щоб виправдати складні методи сортування даних, щоб ви ніколи не витрачали гроші даремно.

Рівень HSM з найменшим обсягом та найвищою продуктивністю зазвичай називається Рівнем 0. Це критично важливі дані, які не можуть дозволити собі затримок або перебоїв у роботі. Рівень 1 часто називають «гарячими даними», даними, які постійно використовуються для щоденних бізнес-операцій, а терміновість яких може бути збалансована витратами на зберігання. Рівень 2 – це «теплі дані», де вартісним міркуванням надається значний пріоритет, і куди поміщаються дані, до яких не часто звертаються. Нарешті, Рівень 3 зазвичай

стосується «холодних даних», або даних, до яких рідко звертаються або оновлюють дані, якщо взагалі звертаються.

Альтернативи рівням HSM, що базуються на повністю флеш-пам'яті

В ідеальному світі компанія могла б мати швидке, високодоступне сховище на всіх рівнях, з тонкими розмежуваннями між ними. Протягом десятиліть мрією було повністю флеш-сховище даних на рівні підприємства. Але донедавна це було просто неможливо. Однак за останні кілька десятиліть вартість повністю флеш-сховища на рівні підприємства не просто можлива, це робиться регулярно.

Pure Storage є помітним лідером у сфері повністю флеш-сховищ для підприємств і ще у 2012 році розробила рішення для флеш-сховищ, які могли задовольнити потреби корпоративної мережі рівня Tier 0. Коли FlashArray//C ™ був випущений, Pure Storage міг гарантувати стабільну затримку в одну мілісекунду для критично важливих бізнес-навантажень і даних корпоративних мереж, з доступністю 99,9999% та оновленнями без переривання роботи.

Це саме по собі було новаторським, а потім FlashArray//X™ та FlashArray//XL™ дозволили запускати на флеш-пам'яті все: від величезних баз даних до хмарних програм. Навіть з цими досягненнями все ще вважалося, що рівні 2 та 3 ніколи не потраплять до флеш-пам'яті через вартість.

Але у 2023 році Pure Storage випустила FlashArray//E ™ та FlashBlade//E ™, які кидають виклик низькоякісним обертовим дискам та стрічкам. Розроблений для довготривалого зберігання, FlashArray//E забезпечує ємність зберігання необроблених, уніфікованих файлових та блочних даних від 1 ПБ до 4 ПБ. FlashBlade//E може заощадити компаніям купу грошей під час зберігання неструктурованих та об'єктних робочих навантажень.

Pure Storage показує приклад повністю флеш-альтернатив рівням HSM для організацій будь-якого розміру. Хоча ми не скасовуємо HSM як такий, ми радикально змінюємо межі, що розділяють рівні HSM.

Ієрархічне управління сховищами виникло як необхідна відповідь на

величезну різницю у вартості, що існувала між найшвидшими, найдорожчими та найповільнішими, але найдоступнішими формами зберігання даних. Технологічні обмеження призвели до появи цілої сфери кар'єри, присвяченої автоматичній категоризації, переміщенню та оптимізації рішень для зберігання даних.

HSM все ще є важливим процесом економії коштів, але повністю флеш-рішення корпоративного масштабу, такі як ті, що пропонуються Pure Storage, швидко позбавляються чітких розмежувань між ними, водночас забезпечуючи величезні обсяги швидких даних за значно меншу ціну, ніж раніше.

3.2 Розробка структурної схеми

Організація ієрархічних даних є унікальною проблемою в галузі управління базами даних СУБД. Ієрархічні структури поширені в багатьох галузях, від організації в діаграмах до систем зберігання даних та категорій продуктів.

Для ефективного зберігання та запитування ієрархічних даних у реляційних базах даних RDBMS необхідний ретельний розгляд схеми бази даних та обраної моделі зберігання.

У цій статті ми розглянемо доступні варіанти зберігання ієрархічних даних у реляційній базі даних, досліджуючи їхні переваги, недоліки та варіанти використання. Основні варіанти:

1. Модель списку суміжності.
2. Перерахування шляхів.
3. Модель вкладеного набору.
4. Модель матеріалізованого шляху.

Перш ніж заглибитися, давайте розберемося з цими концепціями:

Що таке реляційна база даних?

Реляційна база даних – це тип бази даних, яка впорядковує дані в рядки та стовпці, що разом утворюють таблицю, де точки даних пов'язані одна з одною

(РСБД).

SQL-запити агрегують дані, допомагаючи фірмам аналізувати ефективність бізнесу, оптимізувати процеси та генерувати аналітичні висновки. Вони впорядковують дані, пов'язуючи таблиці за допомогою первинних та зовнішніх ключів, виявляючи взаємозв'язки.

Зберігання ієрархічних даних у базі даних

Керування ієрархічними даними в реляційних базах даних є складним завданням через невідповідність між ієрархічними структурами та табличною природою реляційних баз даних. Стратегії пояснюються нижче:

1. Модель списку суміжності

Модель списку суміжності – це простий та інтуїтивно зрозумілий спосіб представлення ієрархічних даних у реляційних базах даних. У цій моделі кожен запис містить посилання на батьківський запис, утворюючи деревоподібну структуру. Наприклад, таблиця співробітників може містити поле, що посилається на ідентифікатор менеджера для кожного співробітника.

Переваги:

– Легко зрозуміти та застосувати на практиці.

– Гнучкість є відображенням асиметричних ієрархій.

Недоліки:

– Можуть виникати неефективні запити та перетин ієрархій, особливо для глибоко вкладених структур.

– Часто потрібні рекурсивні запити, які можуть бути складними та ресурсомісткими.

2. Перерахування шляхів

У реляційній базі даних перерахування шляхів – це метод зберігання ієрархічних даних, в якому записується повний шлях кожного вузла від кореневого вузла. За допомогою заданого шляху цей метод спрощує отримання батьківських або дочірніх вузлів, але це може призвести до уповільнення запитів, особливо для великих наборів даних.

Перерахування шляхів спрощує отримання ієрархічних даних, але може збільшити тривалість пошуку, особливо для структур з кількома рівнями вкладеності. Крім того, може знадобитися оновити кілька рядків для оновлення ієрархії, що може вплинути на продуктивність. Через це важливо ретельно зважити всі компроміси під час вибору формату зберігання реляційної бази даних для ієрархічних даних.

Переваги:

– Простий у впровадженні.

– Просте отримання батьківських або дочірніх вузлів за певним шляхом.

Недоліки:

– Отримання та перетин ієрархій може бути повільним та ресурсомістким, особливо для великих наборів даних.

– Обмежена підтримка таких операцій, як запити до піддерев або переупорядкування вузлів.

3. Модель вкладених множин

У моделі вкладених множин два числа – ліве значення та праве значення – представляють кожен вузол у дереві для зберігання ієрархічних даних у реляційній базі даних. Спосіб розподілу цих значень дозволяє ефективно запитувати ієрархічну структуру, отримувати піддерева та виконувати такі операції, як підрахунок нащадків.

Переваги:

– Ефективний для пошуку піддерев та операцій, таких як підрахунок нащадків.

– Добре підходить для ієрархій з частими операціями читання.

Недоліки:

– Складно обслуговувати, особливо під час вставки або видалення вузлів.

– Запити, що включають оновлення ієрархії, можуть бути складними та обчислювально дорогими.

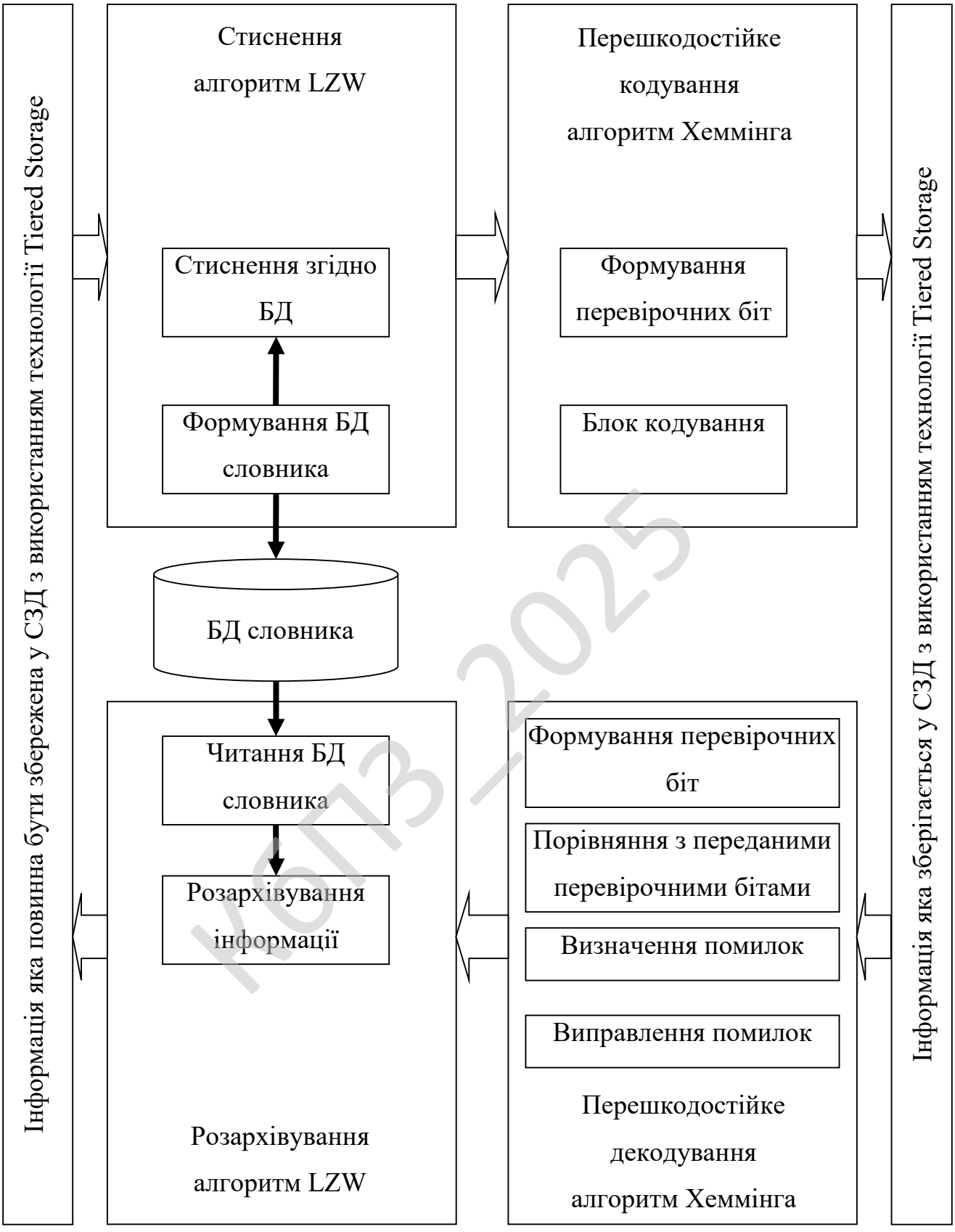

Рисунок 3.1 – Структурна схема системи

Модель матеріалізованого шляху

Подібно до перерахування шляхів, матеріалізована модель шляху зберігає повний шлях кожного вузла, а також додаткові оптимізації, такі як зберігання глибини кожного вузла.

Переваги:

– Спрощує запити та перехід через ієрархії.

– Ефективно підтримує такі операції, як пошук піддерева та запити на основі шляхів.

Недоліки:

– Може знадобитися додаткове місце для зберігання.

– Оновлення ієрархії можуть бути складними, особливо коли вузли переміщуються або реорганізуються.

На завершення, вибір відповідної моделі зберігання ієрархічних даних у реляційній базі даних залежить від різних факторів, включаючи розмір і складність ієрархії, частоту оновлень і типи запитів, які будуть виконуватися. Хоча кожна модель зберігання має свої переваги та недоліки, розуміння нюансів кожного підходу є вирішальним для розробки ефективних і масштабованих схем баз даних, які ефективно керують ієрархічними даними.

3.3 Розробка функціональної схеми

Для реалізації програмного забезпечення зберігання даних з використанням технології Tiered Storage з підвищеною надійністю пропонується використовувати алгоритм LZW-стиску та перешкодостійкий алгоритм Хеммінга.

Алгоритм LZW-стиску

Алгоритм LZW-стиску заміняє рядки символів деякими кодами. Це робиться без якого-небудь аналізу вхідного тексту. Замість цього при додаванні кожного нового рядка символів проглядається таблиця рядків. Стиск відбувається, коли код заміняє рядок символів. Коди, генеруємі LZW-

алгоритмом, можуть бути будь-якої довжини, але вони повинні містити більше біт, ніж одиничний символ. Перші 256 кодів (коли використовуються 8-бітні символи) за замовчуванням відповідають стандартному набору символів. Інші коди відповідають оброблюваним алгоритмом рядкам.

Стиск

Алгоритм LZW-стиску в найпростішій формі наведений нижче. Щораз, коли генерується новий код, новий рядок додається в таблицю рядків. LZW постійно перевіряє, чи є рядок уже відомим, і, якщо так, виводить існуючий код без генерації нового.

Процедура LZW-стиску:

```
РЯДОК = черговий символ із вхідного потоку
WHILE вхідний потік не порожній DO
    СИМВОЛ = черговий символ із вхідного потоку
    IF РЯДОК+СИМВОЛ у таблиці рядків THEN
        РЯДОК = РЯДОК+СИМВОЛ
    ELSE
        вивести у вихідний потік код для РЯДОК
        додати в таблицю рядків РЯДОК+СИМВОЛ
        РЯДОК = СИМВОЛ
    END of IF
END of WHILE
вивести у вихідний потік код для РЯДОК
```

Простий рядок, використаний для демонстрації алгоритму, наведений в таблиці 3.1. Вхідний рядок є коротким списком англійських слів, розділених символом "/".

Вхідний рядок: /WED/WE/WEE/WEB/WET.

Таблиця 3.1 – Алгоритм стиску

Вхід(символи)	Вихід(коди)	Нові коди й відповідні рядки
/W	/	256 = /W
E	W	257 = WE
D	E	258 = ED
/	D	259 = D/
WE	256	260 = /WE
/	E	261 = E/
WEE	260	262 = /WEE
/W	261	263 = E/W
EB	257	264 = WEB
/	B	265 = B/
WET	260	266 = /WET
<EOF>	T	

Як ви можете помітити, аналізуючи алгоритм, його робота починається з того, що на першому кроці циклу він виконує перевірку на наявність рядка "/W" у таблиці. Коли він не знаходить цей рядок, то генерує код для "/" і додає в таблицю рядок "/W". Т.к. 256 символів уже визначені для кодів 0-255, то першому певному рядку може бути поставлений у відповідність код 256. Після цього система читає наступну букву ("E"), додає другий підрядок ("WE") у таблицю й виводить код для букви "W".

Цей процес повторюється доти, поки другий підрядок, що складається із прочитаних символів "/" і "W", не зіставиться зі строковим номером 256. У цьому випадку система виводить код 256 і додає трьохсимвольний підрядок в таблицю. Цей процес триває доти, поки не вичерпається вхідний потік і все коди не будуть виведені.

Вихідний потік для заданого рядка показаний у таблиці 3.1, також як і отримана в результаті таблиця рядків. Як ви можете помітити, ця таблиця швидко

заповнюється, тому що новий рядок додається в таблицю щораз, коли генерується код. У цьому явно виродженому прикладі було виведено п'ять закодованих підрядків і сім символів. Якщо використовувати 9-бітні коди для виводу, то 19-символьний вхідний рядок буде перетворений в 13.5-символьний вихідний рядок. Звичайно, цей приклад був обраний тільки для демонстрації. У дійсності стиск звичайно не починається доти, поки не буде побудована досить велика таблиця, звичайно після прочитання порядку 100 вхідних байт.

Розпакування

Алгоритму стиску відповідає свій алгоритм розпакування. Він одержує вихідний потік кодів від алгоритму стиску й використовує його для точного відновлення вхідного потоку. Однією із причин ефективності LZW-алгоритму є те, що він не має потреби в зберіганні таблиці рядків, отриманої при стиску. Таблиця може бути точно відновлена при розпакуванні на основі вихідного потоку алгоритму стиску. Це можливо тому, що алгоритм стиску виводить СТРОКОВІ й СИМВОЛЬНІ компоненти коду перш ніж він помістить цей код у вихідний потік. Це означає, що стислі дані не обтяжені необхідністю тягти за собою більшу таблицю перекладу.

Алгоритм розпакування представлений на нижче. Відповідно до алгоритму стиску, він додає новий рядок у таблицю рядків щораз, коли читає із вхідного потоку новий код. Усе, що йому необхідно зробити в добавок – це перевести кожний вхідний код у рядок і переслати її у вихідний потік.

```
Процедура LZW-розпакування:
читати СТАРИЙ_КОД
вивести СТАРИЙ_КОД
WHILE вхідний потік не порожній DO
    читати НОВИЙ_КОД
    РЯДОК = перевести НОВИЙ_КОД
    вивести РЯДОК
    СИМВОЛ = перший символ РЯДКА
```

додати в таблицю перекладу СТАРИЙ_КОД+СИМВОЛ

СТАРИЙ_КОД = НОВИЙ_КОД

END of WHILE

Таблиця 3.2 – Алгоритм розпакування

Вхід НОВИЙ КОД	СТАРИЙ КОД	РЯДОК Вихід	СИМВОЛ	Новий вхід таблиці
/	/	/		
W	/	W	W	256 = /W
E	W	E	E	257 = WE
D	E	D	D	258 = ED
256	D	/W	/	259 = D/
E	256	E	E	260 = /WE
260	E	/WE	/	261 = E/
261	260	E/	E	262 = /WEE
257	261	WE	W	263 = E/W
B	257	B	B	264 = WEB
260	B	/WE	/	265 = B/
T	260	T	T	266 = /WET

У таблиці 3.2 наведена схема роботи алгоритму на основі стислих даних, отриманих у вище наведеному прикладі. Важливо відзначити, що побудова таблиці рядків алгоритмом розпакування закінчується саме тоді, коли побудована таблиця рядків алгоритму стиску.

Вихідний потік ідентичний вхідному потоку алгоритму стиску. Відзначимо, що перші 256 кодів уже визначені для перекладу одиночних символів, також як і в алгоритмі стиску.

Проблеми

До нещастя алгоритм розпакування, наведений у таблиці 3.2, є все таким занадто простим. В алгоритмі стиску існують деякі виняткові ситуації, які створюють проблеми при розпакуванні. Якщо існує рядок, що представляє пари (РЯДОК СИМВОЛ) і вже певну в таблиці, а вхідний потік, що переглядається, містить послідовність РЯДОК СИМВОЛ РЯДОК СИМВОЛ РЯДОК, алгоритм стиску виведе код перш, ніж розпаковник одержить можливість визначити його.

Простий приклад ілюструє це. Припустимо, рядок "JOEYN" визначений у таблиці з кодом 300. Коли послідовність "JOEYNJOEYNJOEY" з'являється в таблиці, вихідний потік алгоритму стиску виглядає подібно тому, як показано в таблиці 3.3.

Вхідний рядок:...JOEYNJOEYNJOEY...

Таблиця 3.3 – Кодування

Вхід(символи)	Вихід(коди)	Нові коди й відповідні рядки.
JOEYN	288 = JOEY	300 = JOEYN
A	N	301 = NA
.	.	.
.	.	.
.	.	.
JOEYNJ	300 = JOEYN	400 = JOEYNJ
JOEYNJO	400	401 = JOEYNJO

Коли розпаковник переглядає вхідний потік, він спочатку декодує код 300, потім виводить рядок "JOEYN" і додає визначення для, скажемо, коду 399 у таблицю, хоча він уже міг там бути. Потім читає наступний вхідний код, 400, і виявляє, що його немає в таблиці. Це вже проблема. На щастя, це відбудеться тільки в тому випадку, якщо розпаковник зустріне невідомий код. Тому що це фактично єдина колізія, те можна без праці вдосконалити алгоритм.

Модифікований алгоритм передбачає спеціальні дії для ще невизначених кодів. У прикладі нижче розпаковник виявляє код 400, що ще не визначений. Тому що цей код не відомий, те декодується значення СТАРОГО_КОДУ, рівне 300. Потім розпаковник додає значення СИМВОЛУ, рівне "J", до рядка. Результатом є правильний переклад коду 400 у рядок "JOEYNJ".

Процедура LZW-розпакування:

```
читати СТАРИЙ_КОД
вивести СТАРИЙ_КОД
СИМВОЛ = СТАРИЙ_КОД
WHILE вхідний потік не порожній DO
    читати НОВИЙ_КОД
    IF NOT у таблиці перекладу НОВИЙ_КОД THEN
        РЯДОК = перевести СТАРИЙ_КОД
        РЯДОК = РЯДОК+СИМВОЛ
    ELSE
        РЯДОК = перевести НОВИЙ_КОД
    END of IF
    вивести РЯДОК
    СИМВОЛ = перший символ РЯДКА
    додати в таблицю перекладу СТАРИЙ_КОД+СИМВОЛ
    СТАРИЙ_КОД = НОВИЙ_КОД
END of WHILE
```

Реалізація

У програмі використовувалися коди довжиною 12, 13 і 14 біт. При довжині коду 12 біт потенційно можливо зберігати до 4096 рядків у таблиці. Щораз, коли читається новий символ, таблиця рядків повинна проглядатися для зіставлення. Якщо зіставлення не знайдене, новий рядок повинна бути додана в таблицю. Тут виникають дві проблеми. По-перше, таблиця рядків може досить

швидко стати дуже великою. Навіть якщо довжина рядків у середньому обмежується 3 або 4 символами кожна, верхня межа довжин рядків може легко перевищити 7 або 8 байт на код. До того ж кількість пам'яті, необхідної для зберігання рядків, заздалегідь не відомо, тому що воно залежить від загальної довжини рядків.

Друга проблема полягає в організації пошуку рядків. Щораз, коли читається новий символ, необхідно організувати пошук для нового рядка виду РЯДОК+СИМВОЛ. Це означає підтримку відсортованого списку рядків. У цьому випадку пошук для кожного рядка включає число порівнянь порядку log2 від загального числа рядків. Використання 12-бітних слів потенційно дозволяє виконувати не більше 12 порівнянь для кожного коду.

Перша проблема може бути вирішена зберіганням рядків як комбінацій код/символ. Тому що кожний рядок у дійсності є поданням комбінації вже існуючого коду й додаткового символу, можна зберігати кожний рядок як окремий код плюс символ. Наприклад у розібраному вище прикладі рядок "/WEE" зберігається як код 260 і символ "E". Це дозволяє використовувати для зберігання тільки 3 байти замість 5 (включающих додатковий байт для кінця рядка). Ідучи назад, можна визначити, що код 260 зберігається як код 256 плюс додатковий символ "E". Нарешті, код 256 зберігається як "/" плюс "W".

Виконання порівняння рядків є небагато більше важким. Новий метод зберігання збільшує час, необхідне для порівняння рядків, але він не впливає на число порівнянь. Ця проблема вирішується використанням алгоритму хешування для зберігання рядків. Це означає, що код 256 не зберігається в якому-небудь масиві за адресою 256, а зберігається в масиві за адресою, сформованому на основі самого рядка. При визначенні місця зберігання даного рядка можна використовувати тестовий рядок для генерації хеш-адреси й потім знайти цільовий рядок однократним порівнянням. Тому що код для будь-якого даного рядка не можна довідатися надалі інакше як по його позиції в масиві, необхідно зберігати код для даного рядка разом з даними рядка. У демонстраційній

програмі для цього використовуються елементи трьох масивів: code_value[i], prefix_code[i] і append_character[i].

Коли необхідно додати новий код у таблицю, використовується хеш-функція в процедурі find_match для генерації коректного і. Процедура find_match генерує адресу й потім перевіряє, чи не використовувався він уже. Якщо це так, то find_match виконує другу пробу й так доти, поки не найдеться вільне місце.

Хеш-функція, використана в цій програмі – проста "xor"-типу хеш-функція. Префікс коду й додатковий символ комбінуються для формування адреси масиву. Якщо вміст префікса коду й символ у масиві зіставляються їм, то вертається коректна адреса. Якщо елемент масиву по цій адресі вже використаний, виконується фіксований зсув для пошуку нового місця. Це виконується доти, поки не буде знайдені вільне місце або не відбудеться зіставлення. Середнє число пошуків у такій таблиці – менше 3, якщо використовується таблиця на 25% більшого розміру, ніж необхідно. Воно може бути поліпшене шляхом збільшення розміру таблиці. Необхідно відзначити, що для того, щоб порядок вторинних проб працював, розмір таблиці повинен бути простим числом. Це пояснюється тим, що проба може бути будь-яким цілим між 1 і розміром таблиці. Якщо проба й розмір таблиці не є взаємно простими, пошук вільних місць може закінчитися невдачею, навіть якщо вони є.

Реалізація алгоритму розпакування має свій набір проблем. Одна із проблем алгоритму стиску тут зникає. Коли виконується стиск, необхідно організувати пошук у таблиці для даного рядка. При розпакуванні необхідно організувати перегляд для окремого коду. Це означає, що можна зберігати префікси кодів і додаткові символи, індексуючись по їхньому строковому коді. Це усуває необхідність у хеш-функції й звільняє масив, що використовувався для зберігання значень кодів.

На жаль метод, використаний для зберігання строкових величин, приводить до того, що декодування рядків повинне виконуватися в інверсному порядку. Це значить, що всі символи для даного рядка при декодуванні повинні

міститися в стековий буфер, а потім виводитися у зворотному порядку. У наведеній програмі це виконується функцією decode_string.

Проблема з'являється, коли читання вхідного потоку переривається при досягненні кінця потоку. Для цієї частки випадку в програмі зарезервований останній обумовлений код MAX_VALUE як ознака кінця даних. Це не є необхідним при читанні файлу, але може допомогти при читанні буфера стислих даних з пам'яті. Витрати на втрату одного обумовленого коду досить малі порівняно з усім процесом.

Результати

Досить важко охарактеризувати результативність якої-небудь техніки стиску даних. Ступінь стиску визначається різними факторами. LZW-стиск виділяється серед інших, коли зустрічається з потоком даних, що містять повторювані рядки будь-якої структури. Із цієї причини він працює досить ефективно, коли зустрічає англійський текст. Рівень стиску може досягати 50% і вище. Відповідно, стиск відеоформ і копій екранів показує ще кращі результати.

Труднощі при стиску файлів даних трохи більші. Залежно від даних, результат стиску може бути як гарним, так і не дуже задовільним. У деяких випадках "стислий" файл може перевершувати по своїх розмірах вихідний текст. Невеликий експеримент дасть Вам подання про те, добре або погано впаковуються Ваші дані.

Однією із проблем є те, що наведена програма не адаптується до різної довжини файлів. Використання 14– або 15-бітних кодів дає кращий ступінь стиску на великих файлах (це пояснюється тим, що для них будуються більші таблиці рядків), але гірше працює з маленькими файлами. Такі програми, як "ARC", вирішують цю проблему використанням кодів змінної довжини. Наприклад, коли величина next_code перебуває між 256 і 511, "ARC" читає й виводить 9-бітні коди. Коли величина next_code стає настільки великий, що необхідно 10-бітні коди, процедури стиску й розпакування збільшують розмір

коду. Це значить, що 12– і 15-бітні варіанти програми працюють добре й на маленьких файлах.

Іншою проблемою великих файлів є те, що зі збільшенням числа прочитаних байтів ступінь стиску може почати погіршуватися. Причина проста: тому що розмір таблиці рядків фіксований, після занесення певного числа рядків їх уже просто нікуди додати. Але побудована таблиця потрібна тільки для тої частини файлу, по якій вона була побудована. Частини, що залишилися, файлу можуть мати інші характеристики й у дійсності потрібна вже трохи відмінна таблиця.

Звичайним способом рішення цієї проблеми є контроль ступеня стиску. Після того, як таблиця рядків заповнена, пакувальник стежить за поводженням коефіцієнта стиску. Після певного ступеня його погіршення таблиця рядків очищається й починає будуватися заново.

Процедура розпакування визначає цей момент тим, що пакувальник записує у свій вихідний потік спеціальний код. Альтернативним способом є визначення найбільш часте рядків, що зустрічаються, і чищення інших. Адаптивна техніка, подібна цієї, може, однак, зустріти труднощі реалізації в програмах розумного розміру.

І, нарешті, можна брати вироблювані LZW-методом коди й пропускати їх через кодуючий фильтр Хаффмана, що адаптується,. Це дає трохи більший ступінь стиску, але вартість такої роботи більше високий, також як і час обробки.

Для підвищення надійності зберігання даних, при ушкодженні архівного файлу пропонується використовувати алгоритм Хеммінга.

Алгоритм Хеммінга

Коди Хеммінга є кодами, що самоконтролюються, тобто кодами, що дозволяють автоматично виявляти найбільш імовірні помилки при передачі даних. Для їхньої побудови досить приписати до кожного слова один додатковий (контрольний) двійковий розряд і вибрати цифру цього розряду так, щоб загальна кількість одиниць у зображенні будь-якого числа була, наприклад, парнею.

Одиночна помилка в якому-небудь розряді переданого слова (у тому числі, може бути, і в контрольному розряді) змінить парність загальної кількості одиниць. Лічильники за модулем 2, що підраховують кількість одиниць, які втримуються серед двійкових цифр числа, можуть давати сигнал про наявність помилок.

При цьому, зрозуміло, ми не одержуємо ніяких вказівок про те, у якому саме розряді відбулася помилка, і, отже, не маємо можливості виправити неї. Залишаються непоміченими також помилки, що виникають одночасно у двох, у чотирьох або взагалі в парній кількості розрядів. Втім, подвійні, а тим більше чотириразові помилки покладаються малоймовірними.

Коди, що самокоректуються

Коди, у яких можливо автоматичне виправлення помилок, називаються що самокоректуються. Для побудови коду, що самокоректується, розрахованого на виправлення одиночних помилок, одного контрольного розряду недостатньо. Як видно з подальшого, кількість контрольних розрядів k повинне бути обране так, щоб задовольнялося нерівності:

$$2^k \geq k + m + 1,$$

або:

$$k \geq \log_2(k + m + 1),$$

де m – кількість основних двійкових розрядів кодового слова. Мінімальні значення k при заданих значеннях m, знайдені відповідно до цієї нерівності, наведені в таблиці 3.4.

Таблиця 3.4 – Мінімальні значення k при заданих значеннях m

Діапазон m	k_{min}
1	2
2-4	3
5-11	4
12-26	5
27-57	6

Маючи m+k розрядів, що самокоректується код можна побудувати наступним чином.

Привласнимо кожному з розрядів свій номер – від 1 до m+k; запишемо ці номери у двійковій системі числення. Оскільки $2^k > m + k$,кожний номер можна представити, мабуть, k-розрядним двійковим числом.

Припустимо далі, що всі m+k розрядів коду розбиті на контрольні групи, які частково перекриваються, причому так, що одиниці у двійковому поданні номера розряду вказують на його приналежність до певних контрольних груп. Наприклад: розряд № 5 належить до 1-й і 3-й контрольним групам, тому що у двійковому поданні його номера $5_{10} = \ldots 000101_2$ – 1-й і 3-й розряди містять одиниці.

Серед m+k розрядів коду при цьому є k розрядів, кожний з яких належить тільки до однієї контрольної групи:

Розряд № 1: $1_{10} = \ldots 000001_2$ належить тільки до 1-й контрольної групи.

Розряд № 2: $2_{10} = \ldots 000010_2$ належить тільки до 2-й контрольній групі.

Розряд № 4: $4_{10} = \ldots 000100_2$ належить тільки до 3-й контрольній групі.

…

Розряд № 2^{k-1} належить тільки до k-*й* контрольної групи.

Ці k розрядів ми й будемо вважати контрольними. Інші m розрядів, кожний з яких належить, щонайменше, до двох контрольних груп, будуть інформаційними розрядами.

У кожній з k контрольних груп будемо мати по одному контрольному розряді. У кожний з контрольних розрядів помістимо таку цифру (0 або 1), щоб загальна кількість одиниць у його контрольній групі було парним.

Наприклад, досить розповсюджений код Хеминга з m=7 і k=4.

Нехай вихідне слово (кодове слово без контрольних розрядів) – 0110101_2.

Позначимо P_i – контрольний розряд №i; а D_i – інформаційний розряд №i, де i = 1,2,3,4…

Таблиця 3.5 – Параметри й значення коду

№ розряду	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011
Розподіл контрольних і інформаційних розрядів	p_1	p_2	d_1	p_3	d_2	d_3	d_4	p_4	d_5	d_6	d_7
Інформаційне кодове слово:			0		1	1	0		1	0	1
p_1	1		0		1		0		1		1
p_2		0	0			1	0			0	1
p_3				0	1	1	0				
p_4								0	1	0	1
Кодове слово з контрольними розрядами:	1	0	0	0	1	1	0	0	1	0	1

Цікаво подивитися, як перекриваються контрольні групи в цьому випадку. Перша група контролює розряди № 3,5,7,9,11 вихідні коди, друга – 3,6,7,10,11 (№ групи = № контрольного розряду). Очевидно, що вони частково перекриваються.

Припустимо тепер, що даного кодового слова 10001100101 відбулася помилка в 11 -м розряді, так, що було прийнято нове кодове слово 10001100100. Зробивши в прийнятому коді перевірку парнесті усередині контрольних груп, ми виявили б, що кількість одиниць непарне в 1-й, 2-й і 4-й контрольних групах, і парне в 3-й контрольній групі. Це вказує, по-перше, на наявність помилки, по-друге, означає, що номер помилково прийнятого розряду у двійковому поданні містить одиниці на першому, другому й четвертому місцях і нуль – на третім місці, т.я. помилка тільки одна, і 3-я контрольна сума виявилася вірною.

Таблица 3.6 – Параметри коду при помилці

№ розряду	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011		
Розподіл контрольних і інформаційних розрядів	p_1	p_2	d_1	p_3	d_2	d_3	d_4	p_4	d_5	d_6	d_7	Контроль по парнесті в групі	Контрольний біт
Передане кодове слово:	1	0	0	0	1	1	0	0	1	0	1		
Прийняте кодове слово:	1	0	0	0	1	1	0	0	1	0	0		
p_1	1		0		1		0		1		0	Fail	1
p_2		0	0			1	0			0	0	Fail	1
p_3				0	1	1	0					Pass	0
p_4								0	1	0	0	Fail	1

Таблиця 3.7 – Виявлення помилки

	p_4	p_3	p_2	p_1	
У двійковому поданні	1	0	1	1	
У десятковому поданні	8		2	1	$\Sigma = 11$

З таблиці видно, що помилка відбулася в 11-м розряді і її можна виправити. Побудований код, зрозуміло, не розрахований на можливість одночасної помилки у двох розрядах.

Наприклад, коли помилки одночасно пройшли в 3-м і 7-м розрядах вихідного коду, перші й другий контрольні біти навіть не помітять підміни.

Коли в прийнятому коді виробляється перевірка парнесті усередині контрольних груп, випадок подвійної помилки нічим зовні не відрізняється від випадку одиночної помилки.

Код Хеммінга

Можна побудувати й такий код, що виявляв би подвійні помилки й виправляв одиночні. Для цього до коду, що самокоректується, розрахованому на виправлення одиночних помилок, потрібно приписати ще один контрольний розряд (розряд подвійного контролю). Повна кількість розрядів коду при цьому буде m+k+1. Цифра в розряді подвійного контролю встановлюється такий, щоб загальна кількість одиниць у всіх $m + k + 1$ розрядах коду було парним. Цей розряд не включається в загальну нумерацію й не входить у жодну контрольну групу.

Наприклад, код Хеминга з m=7 і k=4 Нехай інформаційне кодове слово – 0110101

Таблиця 3.8 – Параметри коду

№ розряду	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	Second Parity
Розподіл контрольних і інформаційних розрядів	p_1	p_2	d_1	p_3	d_2	d_3	d_4	p_4	d_5	d_6	d_7	
Інформаційне кодове слово:			0		1	1	0		1	0	1	
p_1	1		0		1		0		1		1	
p_2		0	0			1	0			0	1	
p_3				0	1	1	0					
p_4								0	1	0	1	
Кодове слово з контрольними розрядами:	1	0	0	0	1	1	0	0	1	0	1	1

При цьому можуть бути наступні випадки.

1. У прийнятому коді в цілому й по всіх контрольних групах кількість одиниць парне. Якщо потрійні помилки й помилки в більшій кількості розрядів виключаються, то перший випадок відповідає безпомилковому прийому коду.

Таблиця 3.9 – Параметри коду при помилці

№ розряду	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	Контроль по парнесті в групі	Контрольний біт	Контроль по парнесті в цілому	Контрольний біт у цілому
Розподіл контрольних і інформаційних розрядів	p_1	p_2	d_1	p_3	d_2	d_3	d_4	p_4	d_5	d_6	d_7				
Передане кодове слово:	1	0	0	0	1	1	0	0	1	0	1				
Прийняте кодове слово:	1	0	0	0	1	1	0	0	1	0	1				
p_1	1		0		1		0		1		1	Pass	0		
p_2		0	0			1	0			0	1	Pass	0		
p_3				0	1	1	0					Pass	0		
p_4								0	1	0	1	Pass	0	1	Pass

Таблиця 3.10 – Виявлення помилки

	p_4	p_3	p_2	p_1	
У двійковому поданні	0	0	0	0	
У десятковому поданні					$\Sigma = 0$

2. У прийнятому коді в цілому кількість одиниць непарне, але у всіх контрольних групах кількість одиниць парне. Другий випадок – помилки тільки в розряді подвійного контролю.

Таблиця 3.11 – Параметри коду при помилці

№ розряду	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	Контроль по парнесті в групі	Контрольний біт	Контроль по парнесті в цілому	Контрольний біт у цілому
Розподіл контрольних і інформаційних розрядів	p_1	p_2	d_1	p_3	d_2	d_3	d_4	p_4	d_5	d_6	d_7				
Передане кодове слово:	1	0	0	0	1	1	0	0	1	0	1				
Прийняте кодове слово:	1	0	0	0	1	1	0	0	1	0	1				
p_1	1		0		1		0		1		1	Pass	0		
p_2		0	0			1	0			0	1	Pass	0		
p_3				0	1	1	0					Pass	0		
p_4								0	1	0	1	Pass	0	0	Fail

Таблиця 3.12 – Виявлення помилки

	p_4	p_3	p_2	p_1	
У двійковому поданні	0	0	0	0	
У десятковому поданні					$\Sigma = 0$

3. У прийнятому коді в цілому й у деяких з контрольних груп кількість одиниць непарне. Третій випадок – одиночної помилки в якому-небудь із інших розрядів (можна виправити відповідно до наведеного вище правилами).

Таблиця 3.13 – Параметри коду при помилці

№ розряду	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	Контроль по парності в групі	Контрольний біт	Контроль по парності в цілому	Контрольний біт у цілому
Розподіл контрольних і інформаційних розрядів	p_1	p_2	d_1	p_3	d_2	d_3	d_4	p_4	d_5	d_6	d_7				
Передане кодове слово:	1	0	0	0	1	1	0	0	1	0	1				
Прийняте кодове слово:	1	0	0	0	1	1	0	0	1	0	0				
p_1	1		0		1		0		1		0	Epic Fail	1		
p_2		0	0			1	0			0	0	Fail	1		
p_3				0	1	1	0					Pass	0		
p_4								0	1	0	0	Fail	1	1	Fail

Таблиця 3.14 – Виявлення помилки

	p_4	p_3	p_2	p_1	
У двійковому поданні	1	0	1	1	
У десятковому поданні	8		2	1	$\Sigma = 11$

З таблиці видно, що помилка відбулася в 11-м розряді й що її можна виправити.

4. У прийнятому коді в цілому кількість одиниць парне, але в деяких контрольних групах є непарна кількість одиниць – подвійна помилка.

Таблиця 3.15 – Параметри коду при помилці

№ розряду	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	Контроль по парності в групі	Контрольний біт	Контроль по парності в цілому	Контрольний біт у цілому
Розподіл контрольних і інформаційних розрядів	p_1	p_2	d_1	p_3	d_2	d_3	d_4	p_4	d_5	d_6	d_7				
Передане кодове слово:	1	0	0	0	1	1	0	0	1	0	1				
Прийняте кодове слово:	1	0	1	0	1	0	0	0	1	0	1				
p_1	1		1		1		0		1		1	Fail	1		
p_2		0	1			0	0			0	1	Pass	0		
p_3				0	1	0	0					Fail	1		
p_4								0	1	0	1	Pass	0	1	Pass

Таблиця 3.16 – Виявлення помилки

	p_4	p_3	p_2	p_1	
У двійковому поданні	0	1	0	1	
У десятковому поданні		4		1	$\Sigma = 5$

Раз сума, що вийшла, не дорівнює нулю, а контрольний біт указує на помилку передачі, те виявляємо подвійну помилку. Виправлення подвійних помилок тут, звичайно, неможливо.

Збільшуючи далі кількість контрольних розрядів, можна було б побудувати коди, розраховані на виправлення подвійних помилок і виявлення потрійних і т.д. Однак методи побудови цих кодів не цілком розроблені.

На рисунку 3.1 зображена структурна схема розробленого програмного забезпечення.

На ній відображено процес архівування та розархівування інформації за алгоритмом LZW, та підданя заархівованої інформації перешкодостійкому кодуванню за алгоритмом Хеммінга.

Якщо коротко, то алгоритм LZW діє наступним чином.

Даний алгоритм при стиску (кодуванні) динамічно створює таблицю перетворення рядків: певним послідовностям символів (словам) ставляться у відповідність групи біт фіксованої довжини (звичайно 12-бітні). Таблиця ініціалізується всіма 1-символьними рядками (у випадку 8-бітних символів – це 256 записів). У міру кодування, алгоритм переглядає текст символ за символом, і зберігає кожну нову, унікальну 2-символьний рядок у таблицю у вигляді пари код/символ, де код посилається на відповідний перший символ. Після того як нова 2-символьний рядок збережений у таблиці, на вихід передається код першого символу. Коли на вході читається черговий символ, для нього по таблиці перебуває вже, що зустрічався рядок, максимальної довжини, після чого в таблиці зберігається код цього рядка з наступним символом на вході; на вихід видається код цього рядка, а наступний символ використовується в якості початку наступного рядка.

Алгоритму декодування на вході потрібно тільки закодований текст, оскільки він може відтворити відповідну таблицю перетворення безпосередньо по закодованому тексту.

Алгоритм LZW

– Ініціалізація словника всіма можливими односимвольними фразами. Ініціалізація вхідної фрази w першим символом повідомлення.

– Зчитати черговий символ K з кодуємого повідомлення.

– Якщо КІНЕЦЬ_ПОВІДОМЛЕННЯ, то видати код для w, інакше

– Якщо фраза w уже є в словнику, привласнити вхідній фразі значення w і перейти до Кроку 2, інакше видати код w, додати w у словник, привласнити вхідній фразі значення K і перейти до Кроку 2.

Кінець.

Алгоритм Хеммінга

Алгоритм Хеммінга діє наступними чином.

Код Хеммінга являє собою блоковий код, що дозволяє виявити й виправити помилково переданий біт у межах переданого блоку. Звичайно код Хеммінга характеризується двома цілими числами, наприклад, (11,7) використовуваний при передачі 7-бітних ASCII-кодів. Такий запис говорить, що при передачі 7-бітного коду використовується 4 контрольних біта (7+4=11). При цьому передбачається, що мала місце помилка в одному біті й що помилка у двох або більше бітах істотно менш імовірна. З обліком цього виправлення помилки здійснюється з певною ймовірністю. Наприклад, нехай можливі наступні правильні коди (всі вони, крім перш і останнього, відстоять друг від друга на відстань 4):

00000000

11110000

00001111

11111111

При одержанні коду 00000111 не важко припустити, що правильне значення отриманого коду дорівнює 00001111. Інші коди відстоять від отриманого на більшу відстань Хеммінга.

Розглянемо приклад передачі коду букви s = 0x073 = 1110011 з використанням коду Хеммінга (11,7).

Таблиця 3.17 – Структура повідомлення по коду Хеммінга

Позиція біта:	11	10	9	8	7	6	5	4	3	2	1
Значення біта:	1	1	1	*	0	0	1	*	1	*	*

Символами * позначені чотири позиції, де повинні розміщатися контрольні біти. Ці позиції визначаються цілим ступенем 2 (1, 2, 4, 8 і т.д.). Контрольна сума формується шляхом виконання операції XOR (виключаєче АБО) над кодами позицій ненульових біт. У цьому випадку це 11, 10, 9, 5 і 3. Обчислимо контрольну суму.

Таблиця 3.18 – Контрольна сума

11 =	1011
10 =	1010
09 =	1001
05 =	0101
03 =	0011
S =	1110

Таким чином, приймач одержить наступний код.

Таблиця 3.19 – Отриманий код

Позиція біта:	11	10	9	8	7	6	5	4	3	2	1
Значення біта:	1	1	1	1	0	0	1	1	1	1	0

Просумуємо знову коди позицій ненульових біт і одержимо нуль.

Таблиця 3.20 – Сума кодів позицій ненульових біт

11 =	1011
10 =	1010
09 =	1001
08 =	1000
05 =	0101
04 =	0100
03 =	0011
02 =	0010
S =	0000

Ну а тепер розглянемо два випадки помилок в одному з біт посилки, наприклад, у біті 7 (1 замість 0) і в біті 5 (0 замість 1). Просумуємо коди позицій ненульових біт ще раз.

Таблиця 3.21 – Сума кодів позицій ненульових біт

11 =	1011
10 =	1010
09 =	1001
08 =	1000
07 =	0111
05 =	0101
04 =	0100
03 =	0011
02 =	0010
S =	0111

11 =	1011
10 =	1010
09 =	1001
08 =	1000
04 =	0100
03 =	0011
02 =	0010
S =	0001

В обох випадках контрольна сума дорівнює позиції біта, переданого з помилкою. Тепер для виправлення помилки досить інвертувати біт, номер якого зазначений у контрольній сумі. Зрозуміло, що якщо помилка відбудеться при передачі більш ніж одного біта, код Хеммінга при даній надмірності виявиться марний.

У загальному випадку код має N=M+C біт і передбачається, що не більш ніж один біт у коді може мати помилку. Тоді можливо N+1 стан коду (правильний стан і n помилкових). Нехай M=4, а N=7, тоді слово-повідомлення буде мати вигляд: M4, M3, M2, C3, M1, C2, C1. Тепер спробуємо обчислити значення C1, C2, C3. Для цього використовуються рівняння, де всі операції являють собою додавання за модулем 2:

$$C1 = M1 + M2 + M4$$

$$C2 = M1 + M3 + M4$$

$$C3 = M2 + M3 + M4$$

Для визначення того, чи доставлене повідомлення без помилок, обчислюємо наступні вираження (додавання за модулем 2):

$$C11 = C1 + M4 + M2 + M1$$

$$C12 = C2 + M4 + M3 + M1$$

$$C13 = C3 + M4 + M3 + M2$$

Результат обчислення інтерпретується в такий спосіб.

Таблиця 3.21 – Результат обчислення

C11	C12	C13	Значення
1	2	4	Позиція біт
0	0	0	Помилок немає
0	0	1	Біт С3 не вірний
0	1	0	Біт С2 не вірний
0	1	1	Біт М3 не вірний
1	0	0	Біт С1 не вірний
1	0	1	Біт М2 не вірний
1	1	0	Біт М1 не вірний
1	1	1	Біт М4 не вірний

Описана схема легко переноситься на будь-яке число n і M.

Число можливих кодових комбінацій M завадостійкого коду ділиться на n класів, де N – число дозволених кодів. Поділ на класи здійснюється так, щоб у кожний клас увійшов один дозволений код і найближчі до нього (по відстані Хеммінга) заборонені коди. У процесі прийому даних визначається, до якого класу належить код, що прийшов. Якщо код прийнятий з помилкою, він заміняється найближчим дозволеним кодом. При цьому передбачається, що кратність помилки не більше q_m.

Можна довести, що для виправлення помилок із кратністю не більше q_mm (як правило, воно вибирається рівним $D = 2q_m + 1$). У теорії кодування існують наступні оцінки максимального числа N n-розрядних кодів з відстанню D.

Таблиця 3.22 – Визначення кількості помилок в залежності від кодової відстані

d=1	$n=2^n$
d=2	$n=2\ n^{-1}$
d=3	$n\ 2^n /(1+n)$
d=2q+1	$N \le 2^n (1+\sum_{i=1}^{d} C_n^i)^{-1}$ (для коду Хеммінга ця нерівність перетворюється в рівність)

У випадку коду Хеммінга перші k розрядів використовуються в якості інформаційних, причому k= n – log(n+1), звідки випливає (логарифм по підставі 2), що k може приймати значення 0, 1, 4, 11, 26, 57 і т.д., це й визначає відповідні коди Хеммінга (3,1); (7,4); (15,11); (31,26); (63,57) і т.д.

На рисунку 3.2 зображена функціональна схема системи. Нижче розглянемо її більш докладно.

Основними функціями розробленого програмного забезпечення є наступні:

- Вибір ступеню стиску файлів.
- Блок формування багатотомних архівів.
- Блок добування інформації з багатотомних архівів.
- Блок формування архівів, що саморозпаковуються.
- Вибір режиму solid.
- Блок виводу моніторингу роботи архіватора.
- Блок вибору способу керування програмою.
- Блок вибору мови інтерфейсу.
- Блок допомоги.

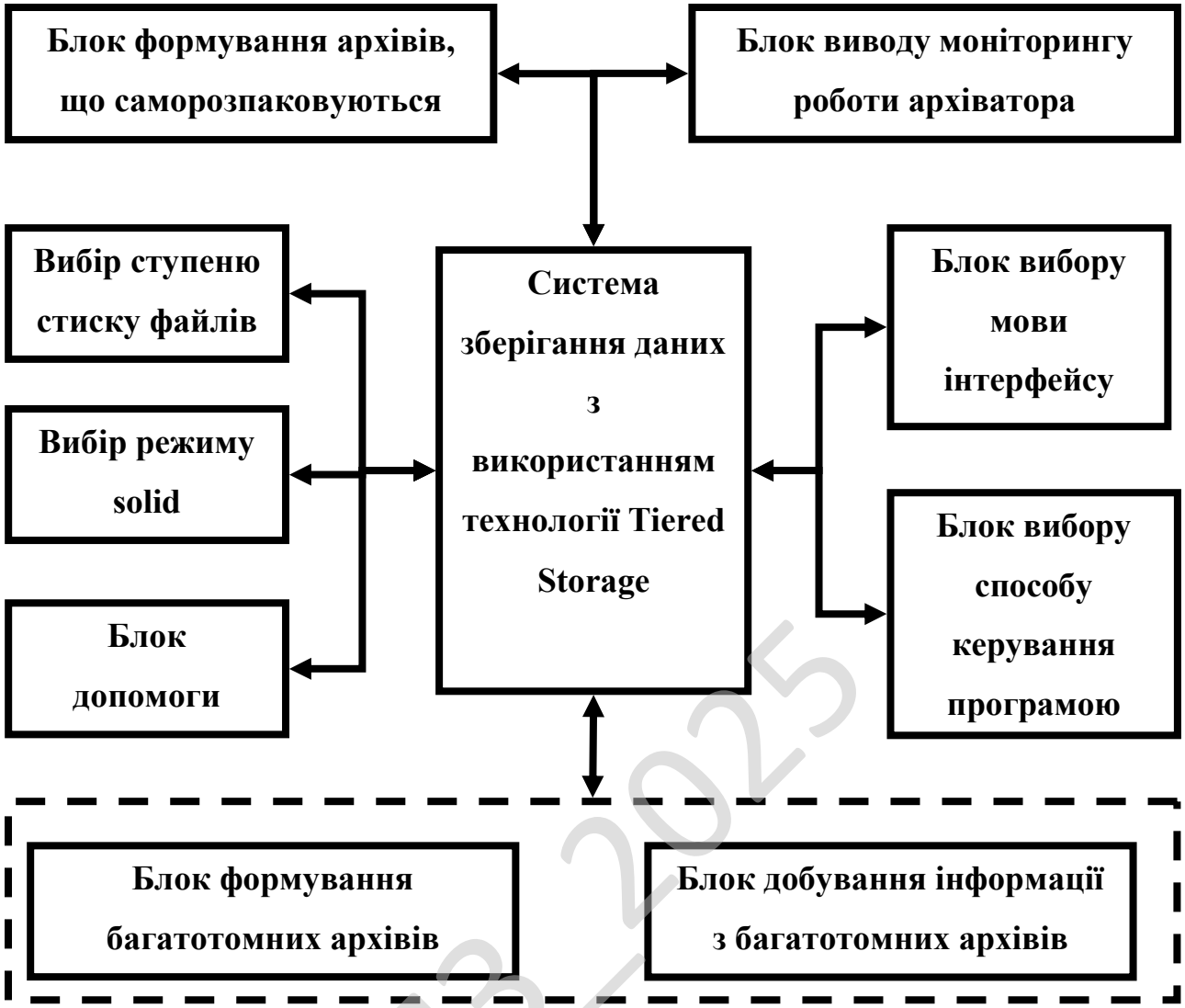

Рисунок 3.2 – Функціональна схема розробленої системи

Розглянемо ці блоки більш детальніше.

Великі за обсягом архівні файли можуть бути розміщені на декількох дисках (томах). Такі архіви називаються багатотомними. Том – це складова частина багатотомного архіву.

Програми-архіватори дозволяють створювати й такі архіви, для добування з яких файлів, що втримуються в них, не потрібні які-небудь програми, тому що самі архівні файли можуть містити програму розпакування. Такі архівні файли називаються такими, що саморозпаковуються.

Архівний файл, що саморозпаковується, – це завантажувальний модуль, що виконується, який здатний до самостійного розархівації файлів, що перебувають у ньому, без використання програми-архіватора. Архів, що саморозпаковується, одержав назву SFX-архів (Self-eXtracting). Архіви такого типу в MS DOS звичайно створюються у формі.exe-файлу. Багато програм-архіваторів роблять розпакування файлів, вивантажуючи їх на диск, але є й такі, які призначені для створення впакованого модуля, що виконується, (програми). У результаті такого впакування створюється програмний файл із тими ж ім'ям і розширенням, що при завантаженні в оперативну пам'ять саморозпаковується й відразу запускається. Разом з тим можливо й зворотне перетворення програмного файлу в розпакований формат. Програми-архіватори крім звичайного режиму стиску, мають режим solid, у якому створюються архіви з підвищеним ступенем стиску й особою структурою організації. У таких архівах всі файли стискуються як один потік даних, тобто областю пошуку повторюваних послідовностей символів є вся сукупність файлів, завантажених в архів, і тому розпакування кожного файлу, якщо він не перший, пов'язана з обробкою інших. Архіви такого типу переважніше використовувати для архівування великої кількості однотипних файлів.

Способи керування програмою-архіватором

Керування програмою-архіватором здійснюється одним із двох способів:

– за допомогою командного рядка MS DOS, у якій формується команда запуску, що містить ім'я програми-архіватора, команду керування й ключі її настроювання, а також імена архівного й вихідного файлів;

– за допомогою убудованої оболонки й діалогових панелей, що з'являються після запуску програми й дозволяють вести керування з використанням меню й функціональних клавіш, що створює для користувача більше комфортні умови роботи.

Виконуючи запропоновані їй дії, програма-архіватор, як правило, виводить на екран протокол своєї роботи. Всі сучасні програми-архіватори оснащені

екранами допомоги, які викликаються при уведенні в командному рядку тільки одного ім'я програми або ім'я із ключем /?. Допомога може бути короткої – на одному екрані або розгорнутої – на декількох. Багато хто архіватори мають екрани допомоги із прикладами складання команд для виконання різних операцій. Інформація допомоги звичайно виводиться на англійській або іншій міжнародній мові.

Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі.

3.4 Розробка діаграми процесів

Діаграма взаємодії процесів системи, розробленої у результаті виконання магістерської роботи, наведена на рисунку 3.3. При детальному її розгляді можна побачити як саме проходе взаємодія у розробленій системі. Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі.

Діаграма взаємодії процесів використовується для візуалізації процесів обробки даних (структурне проектування). Для розробника вважається звичним спочатку креслити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи. Ця діаграма в подальшому підлягає уточненню шляхом деталізації процесів та потоків даних з метою показати систему що розробляється. Діаграми потоків даних містять чотири типи елементів:

– Процеси які являють собою трансформацію даних в рамках описуваної системи.

– Сховища даних (репозиторії).

– Зовнішні по відношенню до системи сутності.

– Потоки даних між елементами трьох попередніх типів.

Рисунок 3.3 – Діаграма взаємодії процесів

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

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

4.1 Блок-схеми та опис алгоритмів функціонування системи

Первинною стадією без якої не відбувається розробка програмного забезпечення це звичайно розробка блок-схем.

На рисунку 4.1 зображена основна блок-схема програми, на рисунку 4.2 зображено роботу підпрограми.

З яких видно що робота основної програми складається з початкових етапів ініціалізації ПЗ, перевірки наявності ресурсів системи, блоку початку основного циклу з чеканням запиту від користувача в якому відбувається виклик підпрограми та останньої стадії – перевірка поточного стану з завершенням роботи розробленого ПЗ. При роботі підпрограми виконується основний функціонал системи з циклічними послідовностями, перевіркою поточного стану та поверненням в основну програму прапорів стану виконання.

Опис алгоритмів функціонування системи

Блок-схеми є першоджерелами стратегії розвитку ПЗ. Тому від точності і детальної блок-схеми залежить результат всієї програми.

При виборі початкової точки відліку при побудові схем було враховано, що виходячи з вибору мови програмування і інших технічних засобів, програма буде об’єктно-орієнтована що вимагає оптимізації програми високого рівня, також те, що при розробці програми слід надати особливу увагу модулю зберігання даних з використанням технології Tiered Storage.

При складанні блок-схем програмного забезпечення і напрацювання алгоритмів я зіткнувся з масою проблем, які вимагали напрацювання процедур і

функцій над основною проблематикою. Для чого були створені додаткові класи, типи даних і константи, що забезпечило вирішення проблем.

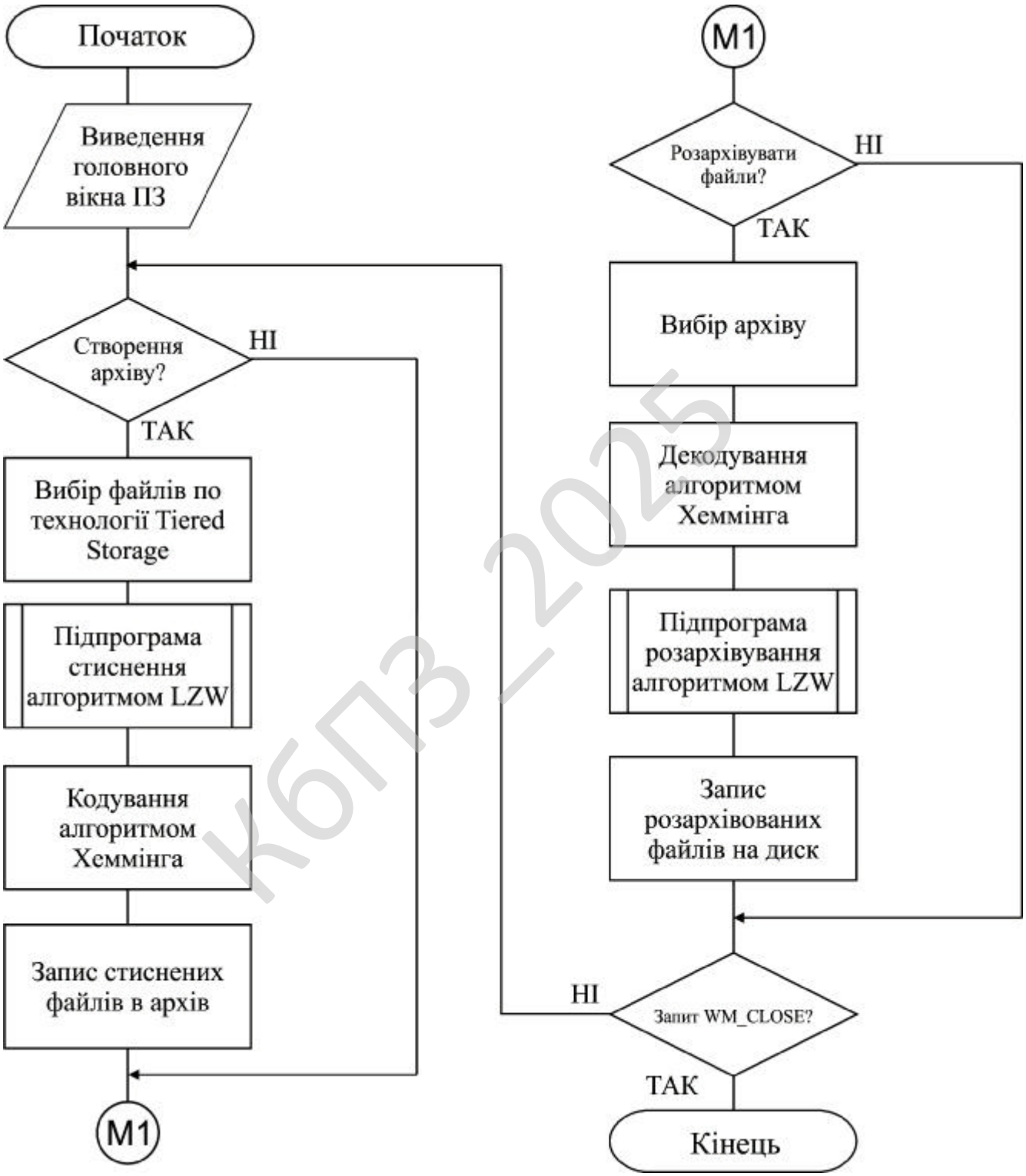

Рисунок 4.1 – Блок-схема основної програми

Організація запису й читання файлу

Перш ніж приступитися до реалізації алгоритмів зберігання даних з використанням технології Tiered Storage варто подумати про спосіб збереження даних на диску (і читання з диска). Стандартними засобами будь-якої мови програмування можно здійснювати читання-запис файлу тільки по байтах (я маю на увазі мінімально можливий тип даних). Для написання ПЗ нам необхідно мати можливість читати й записувати дані по бітах.

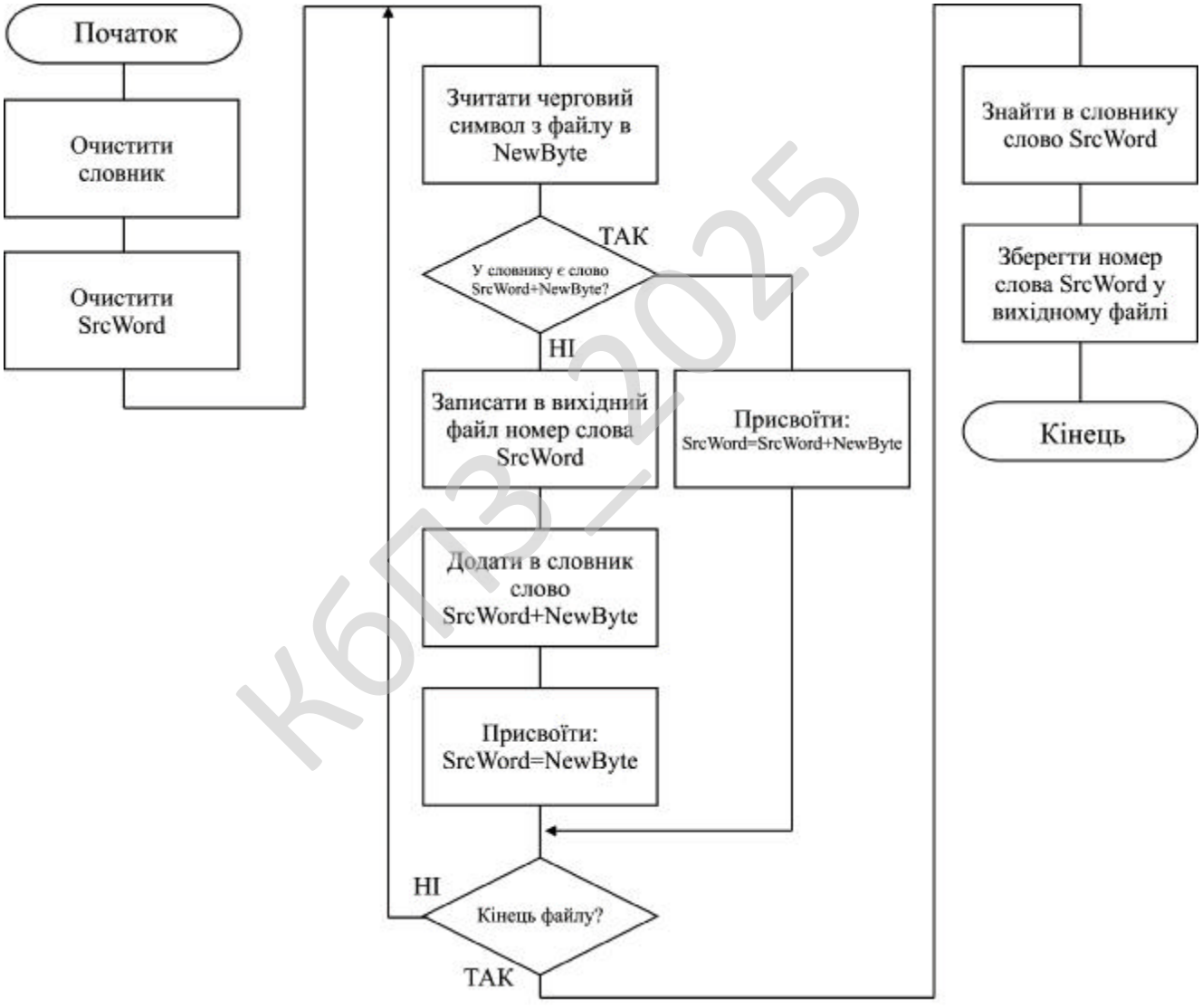

Рисунок 4.2 – Блок-схема роботи підпрограми

Опис системи

Система ієрархічного зберігання даних з використанням концепції Tiered Storage моделює роботу сховища даних датацентру з декількома рівнями продуктивності. Програма на мові пайтон досліджує вплив політики розміщення об’єктів на рівні зберігання, оцінює середню затримку доступу, заповнення об’єму та кількість міграцій між рівнями. Система працює у вигляді консольного застосунку і надає можливість налаштувати параметри рівнів зберігання та профіль навантаження.

Архітектура програмної системи

Програмна реалізація логічно поділяється на декілька блоків. Перший блок описує моделі даних. Другий блок реалізує політику розміщення та міграції об’єктів між рівнями. Третій блок реалізує ядро системи зберігання. Четвертий блок формує навантаження та сценарії досліджень. Пятий блок збирає та виводить показники ефективності.

У моделі даних система використовує сутність об’єкт зберігання. Кожен об’єкт має ідентифікатор, розмір у гігабайтах, рівень важливості, час створення, час останнього доступу та лічильник звернень. Ці параметри дозволяють реалізувати дослідження політики розміщення, яка враховує частоту доступу, вік даних та пріоритет важливості.

Сутність рівень зберігання описує логічний диск або масив певного класу. Для кожного рівня задається назва, максимальний обєм у гігабайтах, середня затримка читання, середня затримка запису, орієнтовна вартість одного гігабайта та коефіцієнт надійності. Рівень підтримує реєстрацію об’єктів, перевірку вільного місця, обчислення заповнення та видалення об’єктів під час міграції або очищення.

Політика Tiered Storage

Політика Tiered Storage формує правила розподілу об’єктів між гарячим, теплим та холодним рівнями. Політика використовує пороги для кількості доступів до об’єкта та поріг віку з моменту останнього звернення. Для нових

об’єктів політика враховує важливість даних та розмір. Об’єкти з високою важливістю за можливості розміщуються у гарячому рівні. Об’єкти великого розміру частіше потрапляють у теплий рівень. Якщо гарячий рівень не має вільного об’єму, політика переходить до наступного рівня.

Для вже розміщених об’єктів політика використовує механізм оцінки міграції. Якщо об’єкт має високу кількість доступів, він повертається до гарячого рівня. Якщо об’єкт використовується рідко та має великий вік з моменту останнього звернення, політика намагається перемістити його у холодний рівень, якщо там достатньо вільного місця. Таким чином система реалізує класичну ідею ієрархічного зберігання. Гарячий рівень зберігає найактивніші об’єкти, холодний рівень зберігає рідко використовувані дані з низьким пріоритетом.

Ядро системи ієрархічного зберігання

Ядро системи реалізує клас, який інкапсулює набір рівнів зберігання, політику Tiered Storage та індекс відповідності об’єктів певному рівню. Ядро підтримує операції запису, читання та видалення.

Операція запису створює новий об’єкт зберігання та звертається до політики розміщення. Політика обирає цільовий рівень. Ядро перевіряє наявність місця та реєструє об’єкт у відповідній структурі. Одночасно ядро оновлює показники метрик. Реєструється кількість операцій запису, сумарна затримка та сумарний боєм записаних даних.

Операція читання знаходить рівень, в якому зберігається об’єкт. Якщо об’єкт відсутній, система реєструє промах та повертає порожній результат. Якщо об’єкт існує, система оновлює час останнього доступу та лічильник звернень. Далі ядро оновлює статистику читання, додає затримку доступу згідно параметрів рівня та об’єму об’єкта.

Після цього ядро передає об’єкт політиці, яка приймає рішення про міграцію. Якщо політика пропонує інший рівень, ядро переносить об’єкт, оновлює індекс та реєструє факт міграції у метриках.

Операція видалення видаляє об’єкт з відповідного рівня, звільняє обєм та видаляє запис з індексу. Ця операція потрібна для сценаріїв дослідження, які передбачають очищення архіву або видалення застарілих даних.

Механізм збору метрик

Система зберігання супроводжується модулем збору метрик. Модуль містить лічильники для кількості операцій читання, кількості операцій запису, промахів, сумарної затримки читання, сумарної затримки запису, сумарного об’єму прочитаних та записаних даних, кількості міграцій та об’єму мігрованих даних.

Модуль надає методи для обчислення середньої затримки читання та середньої затримки запису.

Метод формування словникового подання метрик готує дані для подальшої візуалізації у вигляді таблиць чи графіків. Таким чином пояснювальна записка може містити числові результати експериментів, що підтверджують доцільність обраної політики Tiered Storage.

Генерація навантаження та сценарії дослідження

Для дослідження системи необхідно сформувати кероване навантаження. Програма включає конфігурацію навантаження, яка задає кількість різних об’єктів, діапазон розмірів у гігабайтах, ймовірність виконання операції запису, ймовірність звернення до популярних об’єктів та загальну кількість операцій.

Генератор навантаження створює послідовність операцій. На початку генератор частіше створює нові об’єкти, поки не досягне заданої кількості. Після цього генератор перемикається на режим інтенсивного читання. Частина звернень виконується до підмножини популярних об’єктів, що моделює гарячі дані.

Інша частина звернень звертається до випадкових об’єктів, що моделює фон доступу до менш популярних даних. Для кожної операції запису генератор повертає ідентифікатор, розмір та випадковий рівень важливості.

Клас запуску експерименту приймає екземпляр системи зберігання та генератор навантаження.

Під час виконання сценарію метод послідовно обробляє кроки навантаження. Операції запису передаються до ядра через метод запису. Операції читання використовують метод читання. Після завершення експерименту клас повертає знімок стану системи, який включає поточні параметри рівнів зберігання та значення метрик.

Налаштування системи зберігання

Для зручності дослідження програма містить окрему функцію побудови типової конфігурації. Функція створює три рівні. Гарячий рівень моделює високопродуктивне сховище на базі твердотільних накопичувачів.

Йому призначається невеликий обєм, дуже мала затримка та висока вартість одиниці об’єму. Теплий рівень моделює дисковий масив середнього класу. Він має більший обєм, середню затримку та середню вартість. Холодний рівень моделює архівне сховище з великим обємом, високою затримкою та низькою вартістю.

Функція також створює політику Tiered Storage з обраними порогами. Поріг для гарячого рівня відповідає кількості звернень, після якої дані вважаються дуже популярними. Поріг для теплого рівня відповідає нижньому рівню активності.

Поріг віку визначає час неактивності, після якого дані можна переміщати до холодного рівня. Користувач може змінювати параметри конфігурації та проводити серію експериментів, порівнюючи середні затримки, кількість міграцій та заповнення рівнів.

Консольний інтерфейс

Основна функція програми ініціалізує генератор випадкових чисел, створює конфігурацію рівнів зберігання, налаштовує профіль навантаження та запускає експеримент.

Після завершення виконання на екран виводиться короткий звіт. Для кожного рівня зберігання виводиться назва, використаний обєм, повний обєм, відносне заповнення та кількість об’єктів. Далі виводиться зведена статистика

читання і запису, середні затримки та кількість промахів, а також кількість міграцій.

Отримані числові показники використовуються у пояснювальній записці для обгрунтування ефективності вибраної політики Hierarchical Storage Management. У тексті можна порівняти різні конфігурації порогів, розміри рівнів та профілі навантаження, використовуючи дані, що формує реалізована система.

```
import time
import random
from dataclasses import dataclass, field
from typing import Dict, List, Optional, Tuple

#Клас для зберігання статистики роботи системи
@dataclass
class Metrics:
    total_reads: int = 0
    total_writes: int = 0
    total_misses: int = 0
    total_read_latency_ms: float = 0.0
    total_write_latency_ms: float = 0.0
    bytes_read_gb: float = 0.0
    bytes_written_gb: float = 0.0
    migration_events: int = 0
    migrated_gb: float = 0.0

    def register_read(self, latency_ms: float, size_gb: float) -> None:
        #Реєстрація операції читання
        self.total_reads += 1
        self.total_read_latency_ms += latency_ms
        self.bytes_read_gb += size_gb

    def register_write(self, latency_ms: float, size_gb: float) -> None:
        #Реєстрація операції запису
        self.total_writes += 1
        self.total_write_latency_ms += latency_ms
        self.bytes_written_gb += size_gb

    def register_miss(self) -> None:
        #Реєстрація промаху при спробі читання
        self.total_misses += 1

    def register_migration(self, size_gb: float) -> None:
        #Реєстрація події міграції обєкта між рівнями
        self.migration_events += 1
        self.migrated_gb += size_gb

    def average_read_latency(self) -> float:
        #Обчислення середньої затримки читання
        if self.total_reads == 0:
            return 0.0
        return self.total_read_latency_ms / float(self.total_reads)

    def average_write_latency(self) -> float:
        #Обчислення середньої затримки запису
        if self.total_writes == 0:
            return 0.0
        return self.total_write_latency_ms / float(self.total_writes)
```

```
    def to_dict(self) -> Dict[str, float]:
        #Формування словникового представлення метрик
        return {
            "total_reads": self.total_reads,
            "total_writes": self.total_writes,
            "total_misses": self.total_misses,
            "total_read_latency_ms": self.total_read_latency_ms,
            "total_write_latency_ms": self.total_write_latency_ms,
            "bytes_read_gb": self.bytes_read_gb,
            "bytes_written_gb": self.bytes_written_gb,
            "migration_events": self.migration_events,
            "migrated_gb": self.migrated_gb,
            "average_read_latency_ms": self.average_read_latency(),
            "average_write_latency_ms": self.average_write_latency(),
        }

#Клас обєкта зберігання даних
@dataclass
class DataObject:
    object_id: str
    size_gb: float
    importance: int = 1
    created_at: float = field(default_factory=time.time)
    last_access_at: float = field(default_factory=time.time)
    access_count: int = 0

    def register_access(self) -> None:
        #Оновлення статистики при доступі до обєкта
        self.access_count += 1
        self.last_access_at = time.time()

#Клас рівня зберігання даних
@dataclass
class StorageTier:
    name: str
    max_capacity_gb: float
    read_latency_ms: float
    write_latency_ms: float
    cost_per_gb: float
    reliability: float
    objects: Dict[str, DataObject] = field(default_factory=dict)
    used_capacity_gb: float = 0.0

    def can_store(self, size_gb: float) -> bool:
        #Перевірка наявності вільного місця у рівні зберігання
        return self.used_capacity_gb + size_gb <= self.max_capacity_gb

    def add_object(self, obj: DataObject) -> None:
        #Додавання обєкта до рівня зберігання
        if not self.can_store(obj.size_gb):
            raise ValueError("Недостатньо місця у рівні зберігання")
        self.objects[obj.object_id] = obj
        self.used_capacity_gb += obj.size_gb

    def remove_object(self, object_id: str) -> Optional[DataObject]:
        #Видалення обєкта з рівня зберігання
        obj = self.objects.pop(object_id, None)
        if obj is not None:
            self.used_capacity_gb -= obj.size_gb
        return obj

    def get_object(self, object_id: str) -> Optional[DataObject]:
```

```
        #Пошук обєкта у рівні зберігання
        return self.objects.get(object_id)

    def utilization(self) -> float:
        #Обчислення відносного заповнення рівня зберігання
        if self.max_capacity_gb == 0.0:
            return 0.0
        return self.used_capacity_gb / float(self.max_capacity_gb)

#Клас політики розміщення та міграції обєктів між рівнями
class TieringPolicy:
    def __init__(
        self,
        hot_tier_name: str,
        warm_tier_name: str,
        cold_tier_name: str,
        hot_access_threshold: int = 40,
        warm_access_threshold: int = 5,
        cold_age_threshold_seconds: float = 600.0,
    ) -> None:
        #Ініціалізація політики Tiered Storage
        self.hot_tier_name = hot_tier_name
        self.warm_tier_name = warm_tier_name
        self.cold_tier_name = cold_tier_name
        self.hot_access_threshold = hot_access_threshold
        self.warm_access_threshold = warm_access_threshold
        self.cold_age_threshold_seconds = cold_age_threshold_seconds

    def choose_tier_for_new(
        self,
        obj: DataObject,
        tiers: Dict[str, StorageTier],
    ) -> StorageTier:
        #Вибір рівня зберігання для нового обєкта
        hot = tiers[self.hot_tier_name]
        warm = tiers[self.warm_tier_name]
        cold = tiers[self.cold_tier_name]

        if obj.importance >= 3 and hot.can_store(obj.size_gb):
            return hot

        if obj.size_gb > hot.max_capacity_gb * 0.05 and
warm.can_store(obj.size_gb):
            return warm

        if hot.can_store(obj.size_gb):
            return hot

        if warm.can_store(obj.size_gb):
            return warm

        return cold

    def decide_migration(
        self,
        obj: DataObject,
        current_tier: StorageTier,
        tiers: Dict[str, StorageTier],
    ) -> StorageTier:
        #Прийняття рішення про можливу міграцію обєкта
        now = time.time()
        idle_time = now - obj.last_access_at
```

```
        hot = tiers[self.hot_tier_name]
        warm = tiers[self.warm_tier_name]
        cold = tiers[self.cold_tier_name]

        if obj.access_count >= self.hot_access_threshold and hot is not
current_tier:
            if hot.can_store(obj.size_gb):
                return hot

        if obj.access_count >= self.warm_access_threshold and warm is not
current_tier:
            if warm.can_store(obj.size_gb):
                return warm

        if idle_time >= self.cold_age_threshold_seconds and cold is not
current_tier:
            if cold.can_store(obj.size_gb):
                return cold

        return current_tier

#Клас системи ієрархічного зберігання даних
class TieredStorageSystem:
    def __init__(self, tiers: List[StorageTier], policy: TieringPolicy) -> None:
        #Ініціалізація системи ієрархічного зберігання
        self.tiers: Dict[str, StorageTier] = {tier.name: tier for tier in tiers}
        self.policy = policy
        self.object_index: Dict[str, str] = {}
        self.metrics = Metrics()

    def put(self, object_id: str, size_gb: float, importance: int = 1) -> None:
        #Операція запису нового обєкта
        if object_id in self.object_index:
            raise ValueError("Обєкт з таким ідентифікатором уже існує")

        obj = DataObject(object_id=object_id, size_gb=size_gb,
importance=importance)
        target_tier = self.policy.choose_tier_for_new(obj, self.tiers)
        target_tier.add_object(obj)
        self.object_index[object_id] = target_tier.name
        self.metrics.register_write(target_tier.write_latency_ms, size_gb)
```

Також при розробці магістерської роботи було використано наступні підходи UML: діаграма діяльності (діаграми поведінки типу); діаграма прецедентів (діаграми поведінки типу); Діаграма класів.

Діаграма діяльності. Це візуальне представлення графу діяльностей. Граф діяльностей є різновидом графу станів скінченного автомату, вершинами якого є певні дії, а переходи відбуваються по завершенню дій.Дія є фундаментальною одиницею визначення поведінки в специфікації. Дія отримує множину вхідних сигналів, та перетворює їх на множину вихідних сигналів.

Одна із цих множин, або обидві водночас, можуть бути порожніми. Виконання дії відповідає виконанню окремої дії. Подібно до цього, виконання

діяльності є виконанням окремої діяльності, буквально, включно із виконанням тих дій, що містяться в діяльності. Кожна дія в діяльності може виконуватись один, два, або більше разів під час одного виконання діяльності. Щонайменше, дії мають отримувати дані, перетворювати їх та тестувати, деякі дії можуть вимагати певної послідовності.

Специфікація діяльності (на вищих рівнях сумісності) може дозволяти виконання декількох (логічних) потоків, та існування механізмів синхронізації для гарантування виконання дій у правильному порядку.

Діаграма прецедентів це діаграма, на якій зображено відношення між акторами та прецедентами в системі. Також, перекладається як діаграма варіантів використання.

Діаграма прецедентів є графом, що складається з множини акторів, прецедентів (варіантів використання) обмежених границею системи (прямокутник), асоціацій між акторами та прецедентами, відношень серед прецедентів, та відношень узагальнення між акторами. Діаграми прецедентів відображають елементи моделі варіантів використання.

Суть даної діаграми полягає в наступному: проектована система представляється у вигляді безлічі сутностей чи акторів, що взаємодіють із системою за допомогою так званих варіантів використання. Варіант використання (use case) використовують для описання послуг, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, який виконує система при діалозі з актором.

При цьому нічого не говориться про те, яким чином буде реалізована взаємодія акторів із системою.

У мові UML є кілька стандартних видів відношень між акторами і варіантами використання:

– асоціації (association relationship);

– включення (include relationship);

– розширення (extend relationship);

– узагальнення (generalization relationship).

При цьому загальні властивості варіантів використання можуть бути представлені трьома різними способами, а саме – за допомогою відношень включення, розширення і узагальнення.

Відношення асоціації – одне з фундаментальних понять у мові UML і в тій чи іншій мірі використовується при побудові всіх графічних моделей систем у формі канонічних діаграм.

Включення (include) у мові UML – це різновид відношення залежності між базовим варіантом використання і його спеціальним випадком. При цьому відношенням залежності (dependency) є таке відношення між двома елементами моделі, при якому зміна одного елемента (незалежного) приводить до зміни іншого елемента (залежного).

Відношення розширення (extend) визначає взаємозв'язок базового варіанта використання з іншим варіантом використання, функціональна поведінка якого задіюється базовим не завжди, а тільки при виконанні додаткових умов.

Діаграма класів це статичне представлення структури моделі. Відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення.

Діаграма класів, також, може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка розкривається в інших типах діаграм.

Діаграма класів (class diagram) служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. На цій діаграмі показують класи, інтерфейси, об'єкти й кооперації, а також їхні відносини.

В UML існують наступні типи зв'язків які використовуються у діаграмі класів: Асоціації; Агрегація; Композиція.

Асоціації це якщо між двома класами визначена асоціація, то можна переміщатися від об'єктів одного класу до об'єктів іншого. Цілком припустимі випадки, коли обидва кінці асоціації відносяться до одного і того ж класу. Це означає, що з об'єктом деякого класу дозволено зв'язати інші об'єкти з того ж класу. Асоціація, що зв'язує два класи, називається бінарної. Можна, хоча це рідко буває необхідним, створювати асоціації, що зв'язують відразу кілька класів. Графічно асоціація зображується у вигляді лінії, що з'єднує клас сам з собою або з іншими класами.

Асоціації може бути присвоєно ім'я, яке описує природу відносини. Зазвичай ім'я асоціації не вказується, якщо тільки ви не хочете явно задати для неї рольові імена або у вашій моделі настільки багато асоціацій, що виникає необхідність посилатися на них і відрізняти один від одного. Ім'я буде особливо корисним, якщо між одними і тими ж класами існує кілька різних асоціацій.

Клас, що бере участь в асоціації, грає в ній деяку роль. По суті, це "обличчя", яким клас, що знаходиться на одній стороні асоціації, звернений до класу з іншого її боку. Можна явно позначити роль, яку клас грає в асоціації.

Часто при моделюванні буває важливо вказати, скільки об'єктів може бути пов'язано допомогою одного примірника асоціації. Це число називається кратністю (Multiplicity) ролі асоціації та записується або як вираз, значенням якого є діапазон значень, або в явному вигляді.

Вказуючи кратність на одному кінці асоціації, ви тим самим говорите, що на цьому кінці саме стільки об'єктів повинно відповідати кожному об'єкту на протилежному кінці. Кратність можна задати рівною одиниці (1), можна вказати діапазон: "нуль або одиниця" (0..1), "багато" (0.. *), "одиниця або більше" (1.. *). Дозволяється також вказувати певне число (наприклад, 3). За допомогою списку можна задати і більш складні кратності, наприклад 0.. 1, 3..4, 6.. *, що означає "будь-яке число об'єктів, крім 2 і 5".

Агрегація це проста асоціація між двома класами відображає структурний відношення між рівноправними сутностями, коли обидва класу знаходяться на

одному концептуальному рівні і ні один не є більш важливим, ніж інший. Але іноді доводиться моделювати відношення типу «частина/ціле», в якому один з класів має більш високий ранг (ціле) і складається з декількох менших за рангом (частин).

Ставлення такого типу називають агрегацією; воно зараховане до відносин типу «має» (з урахуванням того, що об'єкт-ціле має кілька об'єктів-частин). Агрегація є окремим випадком асоціації і зображується у вигляді простої асоціації з незафарбованим ромбом з боку «цілого». Графічно агрегація представляється порожнім ромбом на блоці класу, і лінією, яка від цього ромба до міститься класу.

Композиція це більш суворий варіант агрегації. Відома також як агрегація за значенням.

Композиція має жорстку залежність часу існування екземплярів класу контейнера та примірників містяться класів. Якщо контейнер буде знищений, то весь його вміст буде також знищено. Графічно представляється як і агрегація, але з зафарбовані ромбиком.

4.2 Захист розробленого програмного забезпечення

Дані які використовуються у даній роботі захищаються алгоритмом ДСТУ 7624:2014 («Калина»). Одним із основних алгоритмів симетричного блокового шифрування, що використовуються в Україні, є ДСТУ 7624:2014 («Калина»), який визначає сучасний алгоритм симетричного блокового перетворення для забезпечення конфіденційності й цілісності інформації при її обробці та встановлює режими його роботи.

В алгоритмі шифрування даних «Калина» використовуються криптографічні перетворення, які відповідають сучасним вимогам до рівня криптостійкості та швидкодії.

Даний стандарт розроблено з урахуванням існуючих та потенційних загроз, подальшого інтенсивного розвитку інформаційних технологій та необхідності активного використання протягом кількох наступних десятиліть.

Стандарт блокового симетричного шифрування ДСТУ 7624:2014 визначає десять різних режимів роботи, що широко поширені відповідно до міжнародних стандартів ISO/IEC 10116:2006.

Це спрямовано на забезпечення широкого застосування ДСТУ 7624:2014, у тому числі для великих даних, що передається комп'ютерними мережами, прозорого шифрування жорстких дисків і змінних носіїв, електронних документів, ключових даних.

Ефективність реалізації систем, засобів та протоколів криптографічного великих даних в інформаційно-телекомунікаційних системах різного призначення може бути забезпечена саме наявністю такої кількості режимів роботи алгоритму.

До блокового шифру «Калина» ставляться такі вимоги: високий рівень криптографічної стійкості з достатнім запасом у разі появи нових атак протягом тривалого часу; висока швидкодія програмної реалізації на сучасних та перспективних платформах; компактність програмної та програмно-апаратної реалізації; можливість ефективної інтеграції декількох алгоритмів в одному засобі криптографічного захисту; прозорість проектування, консервативний підхід до забезпечення стійкості; вища (або однакова) ефективність порівняно з найкращими світовими рішеннями.

Криптографічні алгоритми, які визначаються стандартами ДСТУ 7624:2014 і ДСТУ 7564:2014, є гнучкими, підтримують розмір блоку і довжину ключа від 128 до 512 біт.

Стандарт симетричного блокового шифрування «Калина» є результатом багаторічної плідної співпраці Державної служби спеціального зв'язку та великих даних України та провідних українських вчених.

Даний алгоритм шифрування враховує досвід і результати проведення міжнародних та відкритих національних конкурсів криптографічних алгоритмів.

Алгоритм ДСТУ 7624:2014 забезпечує досить високий рівень криптостікості порівняно з міжнародним стандартом AES (ISO/IEC 18033-3:2010), оскільки дає можливість застосовувати блок даних і ключ шифрування розміром аж до 512 біт.

Крім того, він має аналогічну або навіть більш високу швидкодію на сучасних і перспективних програмних та програмно-апаратних платформах.

На даний момент продовжуються роботи зі стандартизації вітчизняних криптографічних алгоритмів та протоколів.

При цьому не обмежується застосування гармонізованих стандартів у сфері захисту конфіденційної інформації.

Також зусилля зосереджено на використанні кращих практик застосування стандартів шифрування даних для великих даних в інформаційно-комунікаційних системах [10, 11].

Оскільки в стандартах симетричного блокового шифрування «Калина» та AES використовуються аналогічні криптографічні перетворення, на наш погляд, буде доцільним порівняти ці два алгоритми.

Основними відмінностями «Калина» від «Rijndael» (AES) є: збільшена кількість циклів шифрування (запас стійкості); використання додавання за модулем 264 і за модулем 2 для введення ключової інформації (захист від алгебричних атак, лінійного та диференціального криптоаналізів, інтерполяційної атаки тощо); використання чотирьох блоків нелінійного перетворення (S-блоків) замість одного (додатковий захист від алгебричних атак, поліпшення властивостей розсіювання алгоритму – покращені статистичні властивості, відповідно, більш високий рівень стійкості до диференціального та лінійного криптоаналізів тощо); використання випадково сформованих чотирьох блоків, відібраних критеріями стійкості до диференціального, лінійного криптоаналізів, ступені нелінійності булевих функцій (на відміну від S-блоку Rijndael/Camellia та

інших шифрів, що використовують звернення в полі та, відповідно, квадратичні залежності між входом і виходом, – захист від алгебричних атак); принципово нова схема створення підключів (захист від усіх відомих атак на схеми створення підключів); досить висока продуктивність; можливість відновлення сеансового ключа за окремим підключем (додатковий захист від атак, що виконують відновлення підключів).

Усі поліпшення спрямовані на збільшення стійкості та запобігання потенційним вразливостям відносно Rijndael, виявленим в останні роки [12].

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 зображено інтерфейс програмного забезпечення, розробленого у результаті виконання магістерської роботи.

Розроблене програмне забезпечення зберігання даних з використанням технології Tiered Storage складається з наступних функціональних блоків:

– Навігаційне меню: Файл; Параметри; Довідка.

– Функціональних кнопок ПЗ: Оновлення; Архівувати; Розархівувати; Довідка.

– Обрання диску перегляду поточних даних.

– Розділу відображення файлів обраного файлового каталогу на диску.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші.

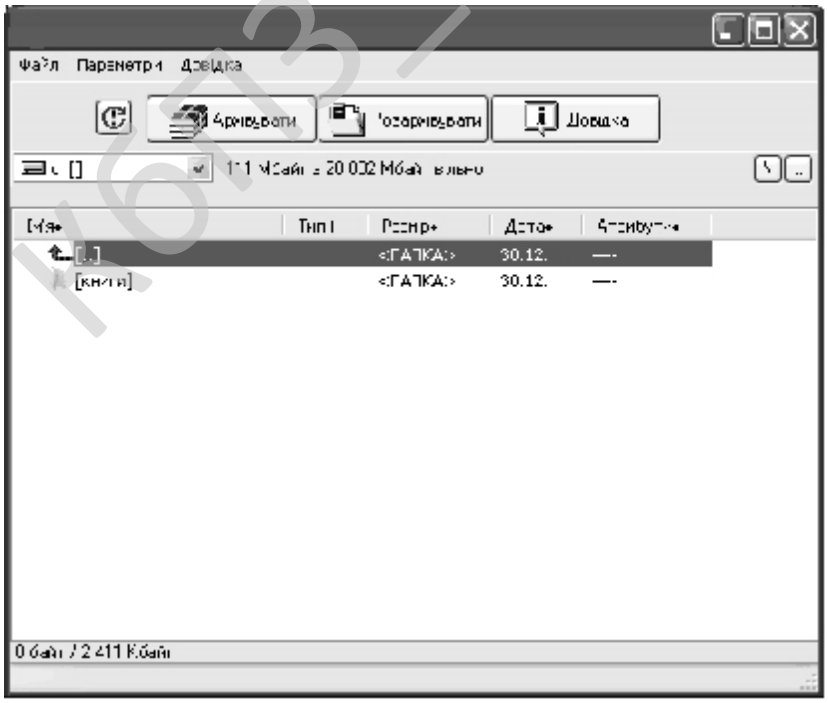

Рисунок 5.1 – Головне вікно розробленого ПЗ

Для перегляду короткої довідки про програму слід натиснути на основному вікні кнопку авторського права, після чого на екрані з’явиться вікно показане на рисунку 5.2.

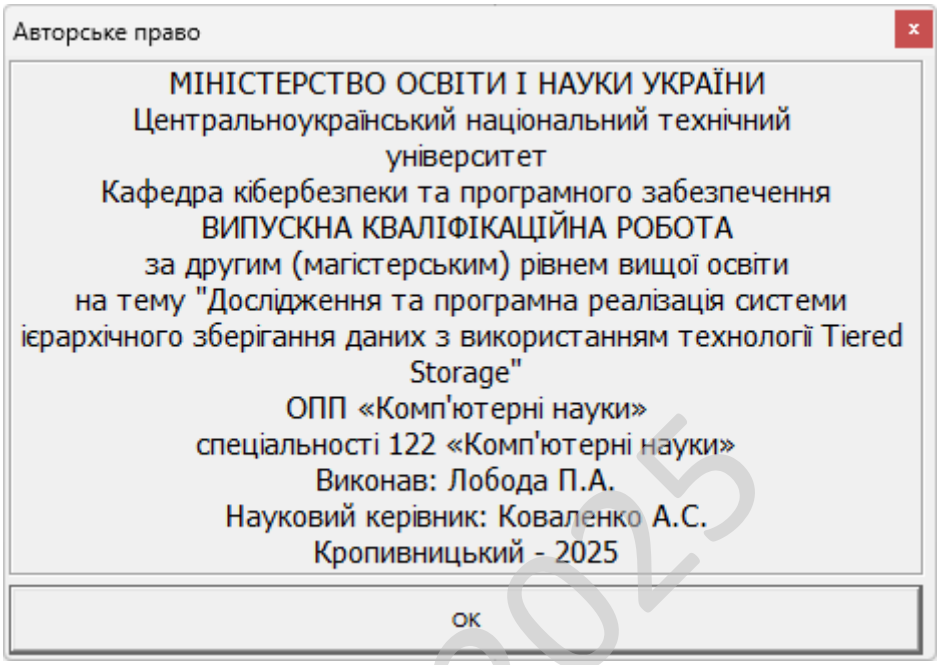

Рисунок 5.2 – Вікно розробника ПЗ

Під час роботи над програмою було проведено тестування програмного забезпечення, тобто технічне дослідження, призначене для виявлення інформації про якість продукту відносно контексту, в якому воно має використовуватись.

Тестування включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою їх оцінки.

Проводилась оцінка:

– відповідності поставленим вимогам;

– правильна відповідь для усіх можливих вхідних даних;

– виконання функцій за прийнятний час;

– практичність;

– сумісність з ОС та стороннім ПЗ.

Оскільки число можливих тестів для програмних компонент практично нескінченне, тому стратегія тестування полягала в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів.

Як результат ПЗ тестувалось стандартним виконанням програми з метою виявлення помилок або інших дефектів.

Проводилось тестування форматом білої скриньки та чорної скриньки.

Тестування форматом білої скриньки засноване на аналізі керуючої структури програми. Програма вважається повністю перевіреною, якщо проведено вичерпне тестування маршрутів (шляхів) її графа управління.

У цьому випадку формуються тестові варіанти, в яких:

– Гарантується перевірка всіх незалежних маршрутів програми.

– Знаходяться гілки True, False для всіх логічних рішень.

– Виконуються всі цикли (у межах їхніх кордонів та діапазонів).

– Аналізується правильність внутрішніх структур даних.

Недоліки тестування "білої скриньки":

– Кількість незалежних маршрутів може бути дуже велика.

– Повне тестування маршрутів не гарантує відповідності програми вихідним вимогам до неї.

– У програмі можуть бути пропущені деякі маршрути.

– Не можна виявити помилки, поява яких залежить від даних.

Переваги тестування "білої скриньки" пов'язані з тим, що принцип «білої скриньки» дозволяє врахувати особливості програмних помилок:

– Кількість помилок мінімально в «центрі» і максимально на «периферії» програми.

– Попередні припущення про ймовірність потоку керування або даних у програмі часто бувають некоректними. У результаті типовим може стати маршрут, модель обчислень за яким опрацьована слабо.

– При записі алгоритму програмного забезпечення у вигляді тексту на мові програмування можливе внесення типових помилок трансляції (синтаксичних та семантичних).

– Деякі результати в програмі залежать не від вихідних даних, а від внутрішніх станів програми.

Проводилось тестування чорної скриньки.

Основне місце програми тестів «чорної скриньки» – інтерфейс ПЗ. Відомі: функції програми. Досліджується: робота кожної функції на всій області визначення.

Ці тести демонструють:

– Як виконуються функції програми.

– Як приймаються вихідні дані.

– Як виробляються результати.

– Як зберігається цілісність зовнішньої інформації.

При тестуванні «чорної скриньки» розглядаються системні характеристики програм, ігнорується їхня внутрішня логічна структура. Вичерпне тестування, як правило, неможливе.

Наприклад, якщо в програмі 10 вхідних величин і кожна приймає по 10 значень, то кількість тестових варіантів становитиме 10^10. Тестування «чорної скриньки» не реагує на багато особливостей програмних помилок.

Тестування «чорної скриньки» (функціональне тестування) дозволяє отримати комбінації вхідних даних, які забезпечують повну перевірку всіх функціональних вимог до програми.

Програмний виріб тут розглядається як «чорна скринька», чию поведінку можна визначити тільки дослідженням його входів та відповідних виходів. При такому підході бажано мати:

– Набір, утворений такими вхідними даними, які призводять до аномалій у поведінці програми (назвемо його ITc).

– Набір, утворений такими вхідними даними, які демонструють дефекти програми (назвемо його ОТ).

Будь-який спосіб тестування «чорної скриньки» повинен:

– Виявити такі вхідні дані, які з високою ймовірністю належать набору ITс;

– Сформулювати такі очікувані результати, які з високою імовірністю є елементами набору ОТ.

Принцип «чорної скриньки» не альтернативний принципу «білої скриньки». Скоріше це доповнює підхід, який виявляє інший клас помилок.

Тестування «чорної скриньки» забезпечує пошук наступних категорій помилок:

– Некоректних чи відсутніх функцій.

– Помилок інтерфейсу.

– Помилок у зовнішніх структурах даних або в доступі до зовнішньої бази даних.

– Помилок характеристик (необхідна ємність пам'яті і т.д.).

– Помилок ініціалізації та завершення.

Обрано умови розповсюдження – Shareware. Під умовно-безплатним програмним забезпеченням можна розуміти спосіб або метод розповсюдження комерційного ПЗ на ринку (тобто на шляху до кінцевого користувача), при якому випробувачеві пропонується обмежена за можливостями (не повнофункціональна або демонстраційна версія), терміном дії (тріал версія) або версія з вбудованим набридливим нагадуванням про необхідність оплати використання програми.

В угоді про використання (ліцензії для кінцевого користувача, EULA) також може бути обумовлена заборона на комерційне або професійне (не тестове) її використання. Основний принцип умовно-безплатного ПЗ – «спробуй, перш ніж купити» (try before you buy). ПЗ що поширюється як умовно-безплатний, надається користувачам безоплатно. Звичайно користувач платить тільки за час завантаження файлів через Інтернет або за носій (CD диск, флешку, ключ). Протягом певного терміну, що становить зазвичай тридцять днів, він може

користуватися програмою, тестувати її, освоювати її можливості. Якщо після закінчення цього терміну користувач вирішить продовжити використання ПЗ, він зобов'язаний купити його (зареєструватися), заплативши авторові певну суму. В іншому випадку користувач повинен припинити використання ПЗ та видалити його зі свого комп'ютера.

6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи ієрархічного зберігання даних з використанням технології Tiered Storage.

Метою розробки є дослідження та програмна реалізація системи ієрархічного зберігання даних з використанням технології Tiered Storage.

Об'єктом дослідження є процес ієрархічного зберігання даних з використанням технології Tiered Storage.

Предметом дослідження є методи ієрархічного зберігання даних з використанням технології Tiered Storage.

Методи дослідження базуються на методах великих даних, методах математичної статистики, методах розробки програмного забезпечення.

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

– Удосконалено метод ієрархічного зберігання даних з використанням технології Tiered Storage.

– Розроблено вітчизняний продукт ієрархічного зберігання даних з використанням технології Tiered Storage, який має більш широкі можливості, на відміну від існуючих аналогів.

7 МАРКЕТИНГОВЕ ТА ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ ІТ-ПРОЄКТУ

7.1 Визначення цільової аудиторії кінцевого готового продукту

Результати цього дослідження насамперед можуть бути корисними для великих підприємств, які працюють із великими обсягами даних і прагнуть оптимізувати витрати на зберігання інформації. Це стосується банків, телекомунікаційних компаній, наукових центрів, державних структур і будь-яких організацій, що мають розгалужену ІТ-інфраструктуру. Для таких установ питання ефективного управління даними має стратегічне значення, адже вартість зберігання інформації без оптимізації постійно зростає.

Також результати проєкту можуть зацікавити хмарних провайдерів і компанії, що надають послуги дата-центрів. Технологія Tiered Storage дозволяє їм забезпечити більш гнучке керування сховищами, підвищити рентабельність сервісів і запропонувати клієнтам персоналізовані тарифи на основі інтенсивності використання даних.

Окрім цього, дослідження має цінність для ІТ-відділів організацій, які планують цифрову трансформацію і прагнуть зменшити навантаження на основні системи. Автоматичний розподіл даних між рівнями сховищ дозволяє вивільнити ресурси для пріоритетних процесів і зменшити ризики перевантаження інфраструктури.

7.2 Оцінка привабливості шляхом застосування методів експертних оцінок

Оцінити привабливість впровадження системи можна за допомогою методу експертних оцінок, який ґрунтується на думці фахівців у сфері ІТ-

інфраструктури та управління даними. До участі залучаються експерти з великих компаній, дата-центрів і хмарних провайдерів, які оцінюють проєкт за п’ятьма ключовими критеріями: технологічна новизна, масштабованість, потенціал економії, складність інтеграції та рівень ринкової потреби.

Наприклад, за результатами опитування 10 експертів середня оцінка за критерієм технологічної новизни може становити 9 балів, економічного потенціалу – 8, масштабованості – 8, складності інтеграції – 6, а ринкової потреби – 9. Після розрахунку середньозваженого показника загальна привабливість становить 8,1 бала з 10, що свідчить про високий рівень перспективності реалізації.

Такий підхід дозволяє не лише підтвердити доцільність розробки, а й виявити слабкі місця – наприклад, складність інтеграції або потребу в додатковому навчанні персоналу. Результати експертного аналізу можуть стати підґрунтям для подальшого залучення інвестицій чи грантового фінансування.

7.3 Вибір методу оцінки вартості ПЗ

Оптимальним для оцінки вартості цього проєкту є витратний метод у поєднанні з аналізом життєвого циклу (Life Cycle Costing). Витратний метод дозволяє точно визначити всі фінансові вкладення – закупівлю серверного обладнання, SSD, HDD і архівних носіїв, придбання ліцензійного ПЗ, а також витрати на інтеграцію й навчання персоналу.

Додатково доцільно застосувати метод оцінки за життєвим циклом, який враховує не лише первинні інвестиції, але й подальші витрати на обслуговування, електроенергію, модернізацію системи та масштабування. Таким чином, можна оцінити не лише стартову ціну проєкту, а й реальну економію протягом кількох років.

Також важливо включити прогнозовану вигоду у вигляді зниження витрат на енергоспоживання, скорочення часу простоїв і вивільнення IT-ресурсів.

Поєднання цих методів створює повну картину економічної доцільності, яка є зрозумілою для інвесторів і керівництва підприємства.

7.4 Розрахунок економічної ефективності від впровадження реалізованого ПЗ як фактору його привабливості

Підприємство зберігає великий обсяг даних на єдиному типі накопичувачів (SSD або HDD), що призводить до перевитрат. Частина даних використовується постійно (оперативні файли, бази даних), тоді як інші – рідко (архіви, звітність, резервні копії). Через це значна кількість високошвидкісних ресурсів зайнята малозатребуваними файлами, які могли б зберігатися на дешевших носіях.

Впровадження системи ієрархічного зберігання даних (Tiered Storage) дозволяє автоматично переміщувати дані між рівнями зберігання – від дорогих SSD до HDD або хмарних архівів. Це забезпечує оптимальне використання ресурсів, зниження витрат на зберігання та підвищення ефективності. Вхідні дані зафіксовано в таблиці 7.1.

Розрахунок економічного ефекту демонструє наступне: поточна структура витрат без Tiered Storage – 6 000 000 грн, нова структура з Tiered Storage – 2 670 000 грн, економія на апаратному забезпеченні – 3 330 000 грн, економія на енергоспоживанні та охолодженні – 400 000 грн/рік, чистий економічний ефект – 1 230 000 грн у перший рік, термін окупності (Payback Period) ≈ 0,67 року (8 місяців), рентабельність інвестицій – 149 %.

Додаткові нефінансові переваги: підвищення ефективності IT-інфраструктури – дані зберігаються на оптимальних носіях, що скорочує час доступу до важливих файлів, автоматизація процесів – алгоритми самостійно визначають, які файли потрібно перенести між рівнями, зменшуючи навантаження на IT-відділ, зниження ризику втрати даних – архівні файли дублюються у хмарі, що забезпечує додаткову надійність, масштабованість –

система легко адаптується до зростання обсягів інформації без великих додаткових інвестицій, екологічність – зниження енергоспоживання скорочує викиди CO_2, що відповідає сучасним "зеленим" стандартам IT.

Таблиця 7.1 – Вихідні дані для розрахунку

Показник	**До впровадження**	**Після впровадження**	**Економічний ефект**
Загальний обсяг даних, ТБ	500	500	—
Частка "гарячих" даних (SSD)	100%	30%	–70%
Вартість 1 ТБ SSD, грн	12 000	12 000	—
Вартість 1 ТБ HDD, грн	3 000	3 000	—
Вартість 1 ТБ архівного зберігання (Cloud Cold Storage), грн	1 200	1 200	—
Середні щорічні витрати на енергоспоживання та охолодження	800 000	400 000	–400 000
Початкові інвестиції у впровадження системи Tiered Storage (обладнання, ліцензії, інтеграція)	—	—	2 500 000 грн

У підсумку підприємство отримує економію понад 3,7 млн грн на рік, стабільну роботу ІТ-систем і можливість масштабування без втрати ефективності.

7.5 Пропозиція алгоритму просування проєкту розробки ПЗ

Першим кроком у просуванні проєкту має бути створення демонстраційного рішення або пілотного проєкту, який би показував реальні результати оптимізації витрат. Наприклад, можна провести тестування в межах одного підрозділу компанії й продемонструвати економію на зберіганні даних у порівнянні з традиційними методами. Такий підхід створює довіру до технології та дозволяє показати її ефективність на практиці.

Далі необхідно зосередитися на партнерстві з ІТ-інтеграторами та постачальниками обладнання, які вже мають клієнтів серед великих підприємств. Вони можуть включити рішення Tiered Storage у свої пакети послуг або пропонувати його як додатковий модуль до існуючих систем зберігання даних.

Після цього доцільно вийти на публічну комунікацію – участь у конференціях, публікація кейсів, демонстрація економічних результатів. Головне у просуванні – зробити акцент не на технічних термінах, а на практичній користі: зниження витрат, підвищення швидкості доступу до даних і покращення управління ресурсами.

7.6 Оптимізація каналів збуту та шляхів реалізації ПЗ

Для підвищення ефективності реалізації варто застосовувати комбіновану стратегію: прямі продажі великим корпоративним клієнтам і партнерські програми з ІТ-компаніями. Прямий канал дозволяє встановити довгострокові відносини з великими споживачами, такими як банки, промислові концерни чи телеком-оператори, які зберігають значні обсяги інформації.

Паралельно можна створити партнерську екосистему, у межах якої інтегратори та системні адміністратори пропонуватимуть рішення своїм клієнтам із прибутковою комісією. Це допоможе масштабувати продажі без значного розширення власного відділу маркетингу.

Додатково можна впровадити SaaS-модель, де клієнти платитимуть за використання системи на основі підписки. Такий підхід зробить технологію доступнішою для середнього бізнесу, який не завжди готовий до великих початкових інвестицій, але потребує ефективного управління даними.

7.7 Визначення ключових факторів успіху конкретного проєкту

Основним фактором успіху є стабільна й ефективна робота системи, яка повинна гарантувати безперебійний доступ до даних незалежно від рівня їх зберігання. Якщо користувач не відчуває різниці у швидкості, коли звертається до “гарячих” або “холодних” даних, – це показник ідеальної оптимізації.

Не менш важливим чинником є простота впровадження та масштабованість. Система має бути гнучкою й сумісною з різними типами сховищ – від локальних серверів до хмарних рішень. Успіх залежить також від рівня автоматизації: чим менше ручних дій потребує система, тим ефективніше вона працює.

Також велике значення має інформаційна підтримка та комунікація з клієнтами. Якщо компанія надає зрозумілі звіти про економію ресурсів, енергоспоживання та покращення продуктивності, це зміцнює довіру й стимулює нових замовників до впровадження. У результаті успіх проєкту визначатиметься не лише технологічною перевагою, а й здатністю команди довести її реальну користь у цифрах і результатах.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

Незважаючи на стрімке розповсюдження комп'ютерних мереж, з’ємні носії інформації не втрачають актуальності у ряді сфер професійної діяльності.

Стрімкий розвиток галузі інформаційних технологій (ІТ) не тільки у сфері управління виробництвом, в банківській систему, бізнесі, системі освіти, але також на транспорті, сфері обслуговування призвів до того, що десятки мільйонів людей у всьому світі виявились втягнутими у взаємодію людини з комп'ютером. Природно виникає запитання: настільки безпечною є ця взаємодія для людини? Адже відома аксіома про те, що будь-яка взаємодія людини та засобів праці двостороння.

Впровадження комп'ютерних технологій принципово змінило характер праці різних категорій фахівців. Працівники, використовують комп'ютерну техніку, на своєму досвіді оцінили її величезні можливості. Одночасно виникла певна безтурботність при її експлуатації.

Недотримання вимог безпеки призводить до того, що й через кілька днів роботи за комп'ютером співробітник починає відчувати певний дискомфорт: в нього виникає головний біль і різь у власних очах, з'являються почуття виснаження й дратівливість. В окремих людей порушується сон, погіршується зір, занедужують руки, шия, поперек тощо.

До недоліків умов праці користувачів комп’ютерної техніки можна віднести:

– недостатню площу і обсяг виробничого приміщення;

– недотримання вимог, мікроклімату на робочих місцях;

– низький рівень освітленості у приміщеннях і на робочих поверхнях апаратури;

– підвищений рівень низькочастотних магнітних полів від моніторів;

– порушення вимог організації робочих місць;

– недотримання вимог до режимам праці та відпочинку;

– надмірне виробничу навантаження працівників;

– відсутність навичок зниження впливу психоемоційного напруги.

Відповідно до ст.14 Закону «Про охорони праці» [1] на роботодавця покладено обов'язок забезпечити: безпеку працівників при експлуатації устаткування; застосування коштів індивідуальної захисту працівників; відповідні вимоги охорони праці, умови праці в кожному робоче місце; дотримання режиму праці та відпочинку працівників; навчання безпечним методам і прийомам виконання; інструктаж з охорони праці; організацію контролю над станом умов праці в робочих місць; проведення атестації робочих місць в умовах праці.

Максимально зменшити кількість шкідливих впливів на людину при високій продуктивності праці, створити комфортні умови для роботи людей – ось одна з головних задач охорони праці.

8.2 Аналіз умов праці

Приміщення розташовано на третьому поверсі п'ятиповерхового будинку. У приміщенні розташовано 3 робочих місць з комп'ютерами (далі ПК). Відповідно до норм «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98 [2] площа, що відводиться для робочого місця з комп'ютером повинна бути не менше 6 м^2, об'єм не менше 20 м^3. Розміри даного приміщень складають: довжина – 6 м, ширина – 4,5 м, висота – 3,5 м, тобто загальна фактична площа складає 27 м^2. Необхідна площа на 3 робочих місця із установленими ПК складає 18 м^2, що не перевищує фактичну. Обсяг кабінету на одного працюючого складає 31,5м^3, отже відповідає нормі ДСанПіН 3.3.2-007-98 – не менше 20 м^3 [2].

При роботі з ПК людина може піддатися впливу шкідливих та небезпечних факторів. Під шкідливими виробничими факторами розуміють фактори, тривалий вплив яких викликає розвиток професійних захворювань. Небезпечні виробничі фактори – вплив яких на працюючого викликає травму, тобто пошкодження організму. Шкідливі і небезпечні чинники, з якими стикається бібліограф при роботі з ПК, приведені в таблиці 8.1.

Таблиця 8.1 – Перелік шкідливих та небезпечних виробничих факторів

Найменування факторів	Можливі джерела їх виникнення	Характер дії
Небезпека ураження електричним струмом	Мережа живлення	Небезпечний
Пожежонебезпечність приміщень	Наявність матеріалів, що згорають і джерел запалення (електроапаратура)	Небезпечний та шкідливий
Іонізація повітря	Статична електрика випромінювання	Шкідливий
Підвищений рівень шуму	Шум створюється перетворювачем напруги ЕОМ, її технічною периферією, а також людьми, що працюють в приміщенні	Шкідливий
Несприятлива освітленість	Недостатнє штучне і природне освітлення	Шкідливий
Незадовільні параметри мікроклімату	Незадовільний стан системи опалення і вентиляції	Шкідливий
Психофізіологічні напруження	Монотонність праці, перенапруженість зорових аналізаторів, розумова напруженість, незручність і статичність пози	Шкідливий

По категорії вибухо- і пожежонебезпеки, згідно дане приміщення відноситься до категорії В – пожежонебезпечне, тому що присутні тверді матеріали, що горять, такі як дерев'яні столи, папір і інше. Виходячи з категорії пожежонебезпеки і поверховості будинку, ступінь вогнестійкості будівлі II. Згідно з ДБН В 1.1-7-2016 «Пожежна безпека об’єктів будівництва» [13] ЕОМ повинні розташовуватись в будівлі не менше ніж II ступню вогнестійкості.

По ступені небезпеки поразки людей електричним струмом відділ, згідно, класифікується як приміщення з підвищеною небезпекою, тому що не виключена можливість одночасного дотику людини до маючих з'єднання з землею конструкціям будинку, з одного боку, і до металевих корпусів електроустаткування, що можуть виявити під напругою – з іншого.

Для забезпечення вищевказаних оптимальних метеорологічних умов у помешканні передбачена система опалення (загальне парове) в холодному періоді, та вентиляція і кондиціювання в теплий період року, згідно ДБН2.5–67–2013 «Опалення, вентиляція та кондиціювання» [4]. При виконанні замірів параметрів мікроклімату, значення їх відповідали оптимальним та допустимим параметрам відповідно до ДСанПіН3.3.2.007–98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно – обчислювальних машин» [2].

Припустимий рівень іонізації повітря помешкання відповідно до СН 21.52-80 повинен складати 1500 – 3000 один./м^3.

Нормування освітлення здійснюється відповідно до ДБН В.2.5 – 28 – 2006 «Природне та штучне освітлення». [5]

Відділ забезпечений комбінованим освітленням. В темний час доби передбачається загальне і/або місцеве рівномірне штучне, а в світлий – бокове одностороннє природне освітлення два віконних прорізи.

8.3 Техніка безпеки та протипожежна профілактика

Відповідно ДБН В 1.1-7-2016 «Пожежна безпека об’єктів будівництва» [13] будинок можна віднести до II групи по ступені вогнестійкості й до категорії Д по ступені пожежонебезпеки.

Від розподільного щита по праву й ліву сторони встановлені кондиціонери, зовнішня електропроводка, поміщена в ізольований кабель. Висота проводки становить 2,2 м від рівня підлоги, її кріплення здійснюється за допомогою металевих власників. Біля кожного стола організований розподільний щит, розташований на текстолітовій пластинці, закріпленої на стіні на рівні 1м від підлоги. Усього до складу входять п'ять розеток і дві клеми заземлення. Всі обчислювальні машини з'єднані із клемами заземлення. Чотири з п'яти розеток забезпечують подачу напруги 220 V, а одна, забезпечує подачу напруги в 36 В. Про це є відповідні написи на кожному розподільному щиті.

Вимоги до пожежної безпеки на підприємстві неухильно повинен дотримуватися кожен співробітник, а організаційна складова при цьому покладається на посадових осіб за відповідним рішенням керівництва і прописується в посадових інструкціях і положеннях по структурним підрозділам.

Зокрема, вказуються конкретні території, ділянки, зони, об’єкти, цілі будівлі і їх частини, поверхи, на яких відповідального співробітника повинне проводити такі організаційні роботи.

Відповідальні особи зобов’язуються розробити, впровадити та підтримувати в певному інструкцією і положенням на ввірених їм об’єктах протипожежний режим і інструкції відповідно до вимог, викладених в нормативних актах.

Передбачено також створення підрозділу добровільної пожежної охорони та пожежно-рятувальної команди в його складі.

Встановлений режим включає порядки з описом місць спеціального призначення та правила їх користування та утримання, наприклад:

– евакуаційних шляхів;

– так званих «курилок»;

– місць складування продукції та сировини;

– стоянки транспорту.

Також встановлюється порядок роботи та технічного обслуговування:

– вентиляційного устаткування;

– засобів пожежогасіння і захисту від загорянь;

– нагрівальних приладів;

– електрообладнання.

Розробляються і впроваджуються правила роботи з відкритим вогнем і горючими матеріалами. Створюються графіки проходження інструктажів з пожежної безпеки співробітників, а також порядок і терміни перевірок знань пожежно-технічного мінімуму, в тому числі, тих працівників, які відповідальні за цю ділянку роботи на підприємстві. При цьому можуть передбачатися внутрішні лекції, семінари, тренінги та практичні заняття на підприємстві, а також зовнішні – на базі спеціалізованих навчальних центрів з професійними викладачами.

Важливою складовою протипожежного режиму на будь-якому об’єкті є розробка і впровадження порядку дій при виникненні пожежі. Неодмінно має бути план евакуації, описано, як повинні відключатися електроустановки, що і в якій послідовності необхідно робити співробітникам.

Відповідно, для кожного об’єкта, кожного приміщення (крім коридорів, санвузлів, басейнів і подібних приміщень), окремих видів робіт складаються інструкції, за якими повинен працювати персонал, залучений на певних ділянках і в виконанні окремих видів робіт. За інструкціями проводиться навчання (інструктаж) персоналу з подальшим контролем знань.

Відповідно НПАОП 40.1-1.21-98 “Правил безпечної експлуатації електроустановок споживачів” [6], приміщення можна віднести до приміщень без підвищеної небезпеки, оскільки це приміщення, сухе, з нормальною

температурою й ізолюючими підлогами, що не має заземлених металоконструкцій.

Персональні ЕОМ можна віднести до першого класу електротехнічних виробів по способі захисту людини від поразки електричним струмом, оскільки їхні корпуси зроблені з ізолюючої пластмаси й кожен пристрій має заземлення. Відповідно правилам пристрою електроустановок ЕОМ можна віднести до електроустановок з робочою напругою до 1000 В.

Однієї з достовірних причин пожежі в приміщенні з обчислювальною технікою може бути коротке замикання, що спричиняє спалах електропроводки. Для його попередження вся обчислювальна техніка, а також інші електричні пристрої повинні бути обладнані плавкими запобіжниками, а на вході електромережі повинен бути передбачений автомат захисту. Не слід користуватися електричними подовжувачами й трійниками, що не мають сертифікатів відповідності вимогам безпеки.

Необхідно передбачити наявність у межах досяжності первинних засобів гасіння пожежі (вогнегасників) для локалізації вогню власними засобами до приїзду команди пожежної охорони. Повинен бути розроблений план екстреної евакуації персоналу при виникненні загоряння. Кількість евакуаційних виходів повинне бути не менш двох. Допускається використання одного евакуаційного виходу, якщо відстань найбільш віддаленого робочого місця до цього виходу не перевищує 25 м.

8.4 Розрахункова частина

Для захисного штучного заземлення застосовуються вертикальні електроди: металевий куток 63х63х6 мм, (згідно з ДСТУ 2251-93 «Кутики сталеві гарячекатані рівнополичні. Сортамент») довжиною L=1,6 м, та горизонтальний електрод – металева полоса з перетином 60х5 мм. Напруга – 220/380 В.

Розрахункова схема розташування заземлюючих електродів – по контуру прямокутником (рис. 8.1).

Розрахунок проведемо за допустимим опором розтіканню струму заземлювача.

Початкові дані для розрахунку захисного заземлення: тип верхнього шару ґрунту – чорнозем, нижнього шару ґрунту – глина (питомий опір ρ_2 = 40 Ом·м). Умовна товщина верхнього шару ґрунту: H=0,55 м. Відстань між вертикальними заземлювачами (електродами) A=3 м. Глибина закладення горизонтального контура заземлення t=0,6 м. Опір заземлювача, який нормується: $R_{ЗН}$ = 4 Ом. Необхідно визначити необхідну кількість вертикальних заземлювачів та довжину полоси (горизонтального заземлювача).

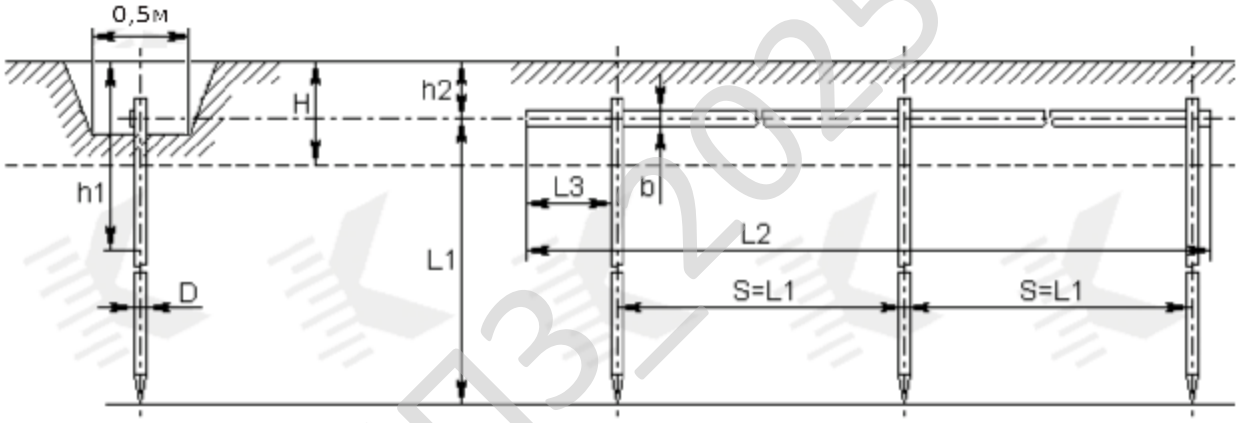

Рисунок 8.1 – Схема штучного заземлення

Виконаємо розрахунок.

Відстань від центра вертикального заземлювача до поверхні землі:

$$T = t + L/2 = 0,6 + 1,6/2 = 1,4 \text{ м.}$$

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

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

де ψ = 1,36 – табличне значення коефіцієнта сезонності для відповідної кліматичної зони у багатошаровому ґрунті [12];

ρ_2 = 40 Ом·м – табличне значення питомого опору нижнього шару ґрунту (глина) [12].

Еквівалентний діаметр вертикального електрода (кутка) [12]:

$$D_в = 0,95 \cdot К = 0,95 \cdot 63 = 59,85 \text{ мм} = 0,0598 \text{ м},$$

де К= 63 мм – розмір металевого кутка (заданий).

Відношення A/L=3/1,6 =1,87.

Опір розтіканню електричного струму одного електроду вертикального заземлювача з урахуванням заглиблення заземлювача [12]:

$$R_О=0,366 \cdot (\rho/L) \cdot [\lg(2L/D_в)+(1/2) \cdot \lg((4T+L)/(4T-L))]=$$

$$=0,366 \cdot (54,5/1,6) \cdot [(\lg(2 \cdot 1,6/0,0598)+(1/2) \cdot \lg((4 \cdot 1,4+1,6)/(4 \cdot 1,4-1,6))]=23 \text{ Ом}.$$

Визначаємо коефіцієнт екранування вертикальних електродів $К_{ев}$=0,62 при орієнтовній кількості вертикальних електродів, яке дорівнює 5 [12].

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

$$N=R_О /(К_{ев} R_{ЗН}) = 23 / (0,62 \cdot 4) = 9,3 \approx 9 \text{ шт}.$$

Визначаємо довжину з'єднуючої полоси:

$$L_П =1,05 \cdot A \cdot N = 1,05 \cdot 3 \cdot 9=28,35 \approx 28 \text{ м}.$$

Опір розтіканню електричного струму з'єднуючої полоси з урахуванням кліматичного коефіцієнта питомого опору ґрунту $К_П$ [12]:

$$R_П = 0,366 \cdot (\rho \cdot К_П/L_П) \cdot \lg(2 \cdot L_П^2/(B \cdot t))=$$

$$=0,366 \cdot (40 \cdot 5/28) \cdot \lg((2 \cdot 28^2)/(0,06 \cdot 0,6)) =12,2 \text{ Ом}.$$

де $К_П$=5 – табличне значення кліматичного коефіцієнта питомого опору ґрунту для відповідної кліматичної зони для з'єднуючої полоси [12]:

B = 60 мм = 0,06 м – ширина з'єднуючої полоси (задана).

Загальний опір розтіканню електричного струму заземлювача [12]:

$$R =(R_О \cdot R_П)/(R_О \cdot \eta_П + N \cdot R_П \cdot К_{ев})=$$

$$= (23 \cdot 12,2)/(23 \cdot 0,6 + 9 \cdot 12,2 \cdot 0,62) =3,48 \text{ Ом}.$$

де $\eta_П$ = 0,6 – табличне значення коефіцієнта екранування з'єднуючої полоси [12].

Умова $R \le R_{ЗН}$ виконується ($3,48 \le 4$).

Оскільки при 9 вертикальних електродах R суттєво менше $R_{ЗН}$, зменшимо кількість вертикальних електродів N до 8 і виконаємо перерахунок. У результаті остаточно отримали: R=3,91 Ом при кількості вертикальних електродів N=8.

8.5 Висновки до розділу

Дотримання всіх необхідних умов праці не лише сприяє збереженню здоров'я працівників, а також підвищує ефективність виробництва в цілому.

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.

Тільки повна усвідомленість працівника про можливі небезпеки, що можуть підстерігати його на робочому місці та дотримання вимог нормативних актів з питань охорони праці та відповідних рекомендацій фахівців, дозволять значною мірою знизити негативний вплив шкідливих та небезпечних факторів при роботі з комп'ютером на організм людини.

Виконано розрахунок захисного штучного заземлення, як одного з ключових факторів безпеки програміста.

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для системи ієрархічного зберігання даних з використанням технології Tiered Storage.

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

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти наведені теоретичне узагальнення й рішення наукового завдання дослідження методів ієрархічного зберігання даних з використанням технології Tiered Storage.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем ієрархічного зберігання даних з використанням технології Tiered Storage.

– Досліджена система ієрархічного зберігання даних з використанням технології Tiered Storage.

– На основі отриманих результатів досліджень створена програмна реалізація системи ієрархічного зберігання даних з використанням технології Tiered Storage.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання ієрархічного зберігання даних з використанням технології Tiered Storage.

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. Дана мова програмування дозволяє найбільш ефективно обробляти дані. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

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

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм ДСТУ 7624:2014.

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.

Проведено маркетингове та економічне обґрунтування ІТ-проєкту, що дозволило визначити ключові фактори успіху даного проекту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Лобода П.А. Дослідження та програмна реалізація системи ієрархічного зберігання даних з використанням технології Tiered Storage // Збірник праць молодих науковців ЦНТУ. – Вип. 15. – Кропивницький: ЦНТУ, 2025.

2. Wendell Odom. «CCNA 200-301 Official Cert Guide, Volume 1». Cisco Press. 2020. – 848 p.

3. Wendell Odom. «CCNA 200-301 Official Cert Guide, Volume 2 Premium Edition eBook and Practice Test». Cisco Press. 2020. – 624 p.

4. Scott Jernigan «CompTIA Network+ Certification All-in-One Exam Guide, Eighth Edition». 2022. – 976 p.

5. Doug Lowe «Networking For Dummies 12th Edition». 2020. – 480 p.

6. Ramon Nastase «Computer Networking: The Beginner's guide for Mastering Computer Networking, the Internet and the OSI Model». 2018. – 186 p.

7. Russ White & Ethan Banks «Computer Networking Problems and Solutions: An Innovative Approach to Building Resilient, Modern Networks». 2017. – 832 p.

8. Вінтенко Б., Смірнов О., Миронець І., Смірнова Т., Смірнов С. «Імітаційна модель шляхів вхідних даних комп'ютерної інтелектуальної системи підтримки оператора енергоблоку АЕС». *Комбінаторні конфігурації та їхні застосування: Матеріали XXVII Міжнародного науково-практичного семінару, присвяченого 125-річчю Національного університету «Запорізька політехніка»* (Запоріжжя-Кропивницький-Київ, 4-6 червня 2025 р.). Запоріжжя: НУ «Запорізька політехніка», 2025. С.82-91.

9. Al-Azzeh, J., Ayyoub, B., Mesleh, A., Smirnova, T., Gnatyuk, S., Drieiev, O., Smirnov, O., Dorenskyi, O. «Cloud-Based Information System for Evaluating Caverns in the Process of Blasting Metal Surfaces of Details». *International Review on Modelling and Simulations* 18 (1), 2025. pp. 32-42.

10. Смірнова Т.В., Конопліцька-Слободенюк О.К., Буравченко К.О., Смірнов С.А., Кравчук О.В., Козірова Н.Л., Смірнов О.А. «Дослідження технологій забезпечення кібербезпеки хмарних сервісів IaaS, PaaS та SaaS». *Кібербезпека: освіта, наука, техніка.* 2024. №4(24), С. 6-27.

11. Батрак О., Смірнова Т., Гнатюк В., Одарченко Р., Смірнов О. «Дослідження показників ефективності функціонування та перспектив розвитку систем IP-телефонії». *Підводні технології,* 2024, № 13, с. 28-35.

12. Kuznetsov, O., Kryvinska, N., Ilchenko, O., Smirnova, T., Ulianovska, Y. «Comparative Analysis of Cryptocurrency Trading Platforms Using the Analytic Hierarchy Process». *CEUR Workshop Proceedings*, 2023, 3628, pp. 106-115.

13. Al-Mudhafar Aqeel, A.M., Smirnova, T., Buravchenko, K., Smirnov, O. «The method of assessing and improving the user experience of subscribers in software-configured networks based on the use of machine learning». *Advanced Information Systems*, 2023, 7(2), pp. 49-56.

14. Smirnov, O., Sydorenko, V., Aleksander, M., Zhyharevych, O., Yenchev, S. «Simulation of the cloud IoT-based monitoring system for critical infrastructures». *CEUR Workshop Proceedings,* Volume 3530, 2023, pp. 256-265.

15. Smirnov, O., Odarchenko, R., Smirnova, T., Bondar, S., Volosheniuk, D. «Optimal Structure Construction of Private 5G Network for the Needs of Enterprises». *Lecture Notes on Data Engineering and Communications Technologies*, 2023, 178, pp. 208–223.

16. Аль-Мудхафар Акіл Абдулхуссейн М., Смірнова Т.В., Буравченко К.О., Смірнов О.А. «Метод оцінки та підвищення користувальницького досвіду абонентів в програмно-конфігурованих мережах на основі використання машинного навчання». *Сучасні інформаційні системи,* 2023, том 7, № 2, С. 49-56.

17. Smirnova, T., Gnatyuk, S., Yudin, O., Sydorenko, V., Polozhentsev, A., «The Model for Calculating the Quantitative Criteria for Assessing the Security Level of Information and Telecommunication Systems». CEUR Workshop Proceedings Volume 3156, 2022, Pages 390-399.

18. Смірнова Т.В., Гнатюк С.О., Сидоренко В.М., Юдін О.Ю., Сидоренко С.Ю., «Модель визначення критичності галузевих інформаційно-телекомунікаційних систем». *Проблеми інформатизації та управління,* № 2(70). 2022. С. 28-37.

19. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Смірнов С.А., Поліщук Л.І., «Дослідження стійкості до диференціального криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Системи управління, навігації та зв'язку,* 2022, № 3(69). С. 93-98.

20. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Поліщук Л.І., Смірнов С.А. «Дослідження статистичної стійкості та швидкісних характеристик запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Вісник Хмельницького національного університету. Серія: «Технічні науки»,* № 2 (307). С. 46-52. 2022.

21. Смірнов О.А., Смірнова Т.В., Константинова Л.В., Смірнов С.А., Якименко Н.М., «Дослідження стійкості до лінійного криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Системи управління, навігації та зв'язку*, 2022, № 1(67). С. 84-89.

22. Smirnova T., Gnatyuk S., Berdibayev R., Avkurova Zh., Iavich M. «Cloud-Based Cyber Incidents Response System and Software Tools». *Communications in Computer and Information Science*, 2021, vol 1486. Springer, Cham. pp 169-184.

23. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova T. «Non-binary constant weight coding technique». *CEUR Workshop Proceedings*. Volume 2740, 2020, Pages 102-114.

24. Smirnov O., Alimseitova Zh., Adranova A., Akhmetov B., Lakhno V., Zhilkishbayeva G. «Models and algorithms for ensuring functional stability and

cybersecurity of virtual cloud resources». *Journal of theoretical and applied information technology* Vol.98. No 21, 2020, P. 3334-3346.

25. Smirnov O., Kuznetsov A., Kiian A., Cherep A., Kanabekova M., Chepurko I. «Testing of code-based pseudorandom number generators for post-quantum application». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT),* Ukraine, Kyiv, May 14-18. 2020. P. 172-177.

26. Smirnov O., Kuznetsov A., Pushkar'ov A., Serhiienko R., Babenko V., Kuznetsova T., «Representation of Cascade Codes in the Frequency Domain». In: Radivilova T., Ageyev D., Kryvinska N. (eds) Data-Centric Business and Applications. Lecture Notes on Data Engineering and Communications Technologies, vol 48. Springer, Cham. 2021. pp 557-587.

27. Smirnov, O., Markovets, O. Vovk, N., Turchyn, Y., «Model of informational support for social network administrators' content creation». *CEUR Workshop Proceedings* Volume 2616, 2020, Pages 125-136.

28. Smirnov, O., Drieieva, H., Drieiev, O., Polishchuk, Y., Brzhanov, R., Aleksander, M. «Method of fractal traffic generation by a model of generator on the graph». *CEUR Workshop Proceedings* Volume 2616, 2020, Pages 366-379.

29. Smirnov, O., Drieieva, H., Drieiev, O., Simakhin, V., Bondar, S., Odarchenko, R. «Managing multifractal properties of the binary sequence generated with the Markov chains», *CEUR Workshop Proceedings* Volume 2608, 2020, Pages 633-645.

30. Smirnov O. Kuznetsov A., Zaicenko Yu., Pastukhov M., Oleshko O., Kuznetsova K., «Formation of Discrete Signals with Special Correlation Properties». *International Conference on Information and Telecommunication Technologies and Radio Electronics, UkrMiCo 2019*; Odessa; Ukraine; 9-13 September 2019. P.22-28.

31. Smirnov, O., Kuznetsov, A., Kolovanova, I., Kuznetsova, T., «Noise immunity of the algebraic geometric codes». International Journal of Computing; 2019,

Volume 18, Issue 4 – Research Institute for Intelligent Computer Systems – 2019. – P. 393-407.

32. Smirnov, O., Kuznetsov, A., Reshetniak, O., Ivko, N., Katkova, T., Kuznetsova, T., «Generators of Pseudorandom Sequence with Multilevel Function of Correlation». *2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T),* Kyiv, Ukraine, 8 – 11 October 2019. P.517-522.

33. Smirnov, O., Odarchenko, R., Abakumova, A., Usik, P., Kundyz, M., «QoE optimization technique for media delivery in 5G networks». *2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T),* Kyiv, Ukraine, 8 – 11 October 2019. P.597-601.

34. Smirnov, O., Krasnobayev, V., Yanko, A., Kuznetsova, T. «Methods of nulling numbers in the system of residual classes». *CEUR Workshop Proceedings*, Vol 2588, P. 90-106, 2019.

35. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Averchev, A., Pastukhov, M., Kuznetsova, K., «Formation of Pseudorandom Sequences with Special Correlation Properties», *2019 3rd International Conference on Advanced Information and Communications Technologies, AICT -2019*/ Lviv, Ukraine, 2-6 July, 2019, P. 395-399.

36. Smirnov, O., Kuznetsov, A., Kiian, A., Zamula, A., Rudenko, S., Hryhorenko, V., «Variance Analysis of Networks Traffic for Intrusion Detection in Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 353-358.

37. Smirnov, O., Kuznetsov, A., Kavun, S., Babenko, B., Nakisko, O., Kuznetsova, K., «Malware Correlation Monitoring in Computer Networks of Promising Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 347-352.

38. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Pastukhov, M., Kuznetsova, K., Prokopovych-Tkachenko, D., «Discrete Signals with Special

Correlation Properties», *CEUR Workshop Proceedings* Volume 2353, *CEUR Workshop Proceedings* 2019, Pages 618-629.

39. Smirnov A.A., Kuzne tsov A.A., Danilenko D.A., Berezovsky A., «The statistical analysis of a network traffic for the intrusion detection and prevention systems», *Telecommunications and Radio Engineering*. – Volume 74, Issue 1. – Begel House Inc. – 2015. – P. 61-78.

40. Смірнов О.А., Смірнова Т.В., Буравченко К.О., Кравченко С.С., Горбов В.О., «Хмарна система підтримки прийняття рішень технологічного процесу відновлення поверхонь конструкцій і деталей машин». *Сучасні інформаційні системи*. 2021. Т. 5, № 4. С. 79-95

41. Смірнов О.А., Усік П.С., Миронець І.В., Буравченко К.О., Якименко Н.М. «Метод підвищення ефективності розподіленої обробки даних у комп'ютерних системах операторів стільникового зв'язку» *Вісник Черкаського державного технологічного університету. Технічні науки.* №4. С. 103-110. 2020.

42. О.А.Смірнов, Т.В.Смірнова, Л.І. Поліщук, К.О. Буравченко, А.О.Макевнін, «Дослідження хмарних технологій як сервісів», *Кібербезпека: освіта, наука, техніка*. № 3(7). С. 43-62. 2020.

43. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В., Поліщук Л.І. Інформаційна безпека в комп'ютерних мережах. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2020. – 294 с.

44. О.А. Смірнов, П.С. Усік, «Дослідження перспектив використання технологічних рішень в мережах 5G» у *Кібербезпека та інформаційні технології: монографія.* – Х.: ТОВ «ДІСА ПЛЮС», 2020.С. 122-135.

45. Смірнов О.А., Дрєєва Г.М., Дрєєв О.М., Смірнова Т.В. «Фрактальний аналіз генератора самоподібного трафіку на основі ланцюга Маркова». *Центральноукраїнський науковий вісник. Технічні науки.* № 2(33). с. 161-172, 2019.

46. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В. Поліщук Л.І. Проектування комп'ютерних систем та мереж. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2019. – 264 с.

47. Smirnov, O., Kuznetsov, A., Kuznetsova., K. Synthesis of Discrete Signals with Improved Correlation Properties. Монографія: In.: ISCI'2019: Information Security in Critical Infrastructures. Collective monograph. Edited by Ivan D. Gorbenko and Alexandr A. Kuznetsov, ASC Academic Publishing, USA, 2019, pp. 281-299. – ISBN: 978-0-9989826-8-7 (Hardback), ISBN: 978-0-9989826-9-4 (Ebook).

48. Смірнов О.А., Дрєєва Г.М. Метод генерування фрактального трафіку за допомогою моделі генератора на графі. Монографія: Інформаційна безпека та інформаційні технології: монографія / за заг. ред. В. С. Пономаренка. – Х.: Вид. Рожко С.Г. 2019. С. 123-139

49. Дрєєва Г.М., Смірнов О.А., Дрєєв О.М. Метод генерування фрактальноподібної числової послідовності на основі скінченного автомату для моделювання трафіку у мережі. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 173-183, 2019.

50. Смірнова Т.В., Солових Є.К., Смірнов О.А., Дрєєв О.М. Побудова хмарних інформаційних технологій оптимізації технологічного процесу відновлення та зміцнення поверхонь деталей. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 184-194, 2019.

51. Смірнов О.А., Смірнов С.А., Поліщук Л.І., Смірнова Т.В., Конопліцька-Слободенюк О.К. Метод формування антивірусного захисту даних з використанням безпечної маршрутизації метаданих. Кібербезпека: освіта, наука, техніка. – Том 3 № 3. – Київ: КУ ім. Бориса Грінченка. – 2019. – С. 63-87.