

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ**

Кафедра системного програмування і спеціалізованих комп'ютерних систем

До захисту допущено

Завідувач кафедри

_____ Віталій РОМАНКЕВИЧ

(підпис)

(ініціали, прізвище)

“ ____ ” _____ 202__ р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою

«Системне програмування та спеціалізовані комп'ютерні системи»

спеціальності

123 «Комп'ютерна інженерія»

на тему: системна утиліта для компресії даних

Виконав:

студент IV курсу, групи _____ КВ-92

(шифр групи)

Оридорога Михайло Васильович

(прізвище, ім'я, по батькові)

(підпис)

Керівник, асистент каф. СПСКС. Молчанов О. А.

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Консультант з нормоконтролю, доц.каф.СПСКС, к.т.н. Клятченко Я.М.

(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали)

(підпис)

Рецензент асистент каф. ОТ Молчанова А. А.

(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали)

(підпис)

Засвідчую, що у цьому дипломному проєкті
немає запозичень з праць інших авторів без
відповідних посилань.

Студент _____

(підпис)

Київ – 2023 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 123 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ

Завідувач кафедри

(підпис)

(ініціали, прізвище)

«__» червня 2023 р.

ЗАВДАННЯ

на дипломний проєкт студента

Оридороги Михайла Васильовча

1. Тема проєкту «Системна утиліта для компресії даних»,
керівник проєкту асистент кафедри СП і СКС Молчанов О. А.,
затверджені наказом по університету від «31» травня 2023 р. № 2107-С
2. Термін подання студентом проєкту _____
3. Вихідні дані до проєкту: див. Технічне завдання
4. Зміст пояснювальної записки:
 - Перелік скорочень, умовних позначень, термінів;
 - Вступ;

- Аналіз існуючих рішень та обґрунтування теми;
- Розроблена утиліта для компресії даних;
- Аналіз отриманих результатів;
- Висновки;
- Список використаної літератури.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо):

- Системна утиліта компресії даних. Схема алгоритму;
- Метод Компресії/Декомпресії Гаффмана. Схема алгоритму;
- Метод Компресії/Декомпресії Лемпеля–Зіва–Велча. Схема алгоритму;
- Системна утиліта компресії даних. Схема структурна.

6. Консультанти розділів проекту*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Розроблення структури додатку	17.01.2023	
2.	Розробка алгоритму та способу взаємодії	04.02.2023	
3.	Програмна реалізація додатку	15.03.2023	
4.	Тестування додатку	04.04.2023	

Студент _____
(підпис)

М. В. Оридорога
(ініціали, прізвище)

Керівник проєкту _____
(підпис)

О. А. Молчанов
(ініціали, прізвище)

* Консультантом не може бути зазначено керівника дипломного проєкту.

АНОТАЦІЯ

Бакалаврський дипломний проєкт включає пояснювальну записку (55 стор., 19 рис., список використаної літератури з 9 найменувань, 4 додатків).

Об'єктом розробки даного проєкту є системна утиліта для компресії даних. Її призначення полягає в забезпеченні стиснення вхідного файлу з метою зменшення розміру.

В процесі роботи над дипломним проєктом виконано наступне:

- проаналізовано існуючі методи стиснення даних та реалізації цих методів;
- визначені вимоги до системної утиліти для компресії даних;
- розроблено утиліту для компресії даних, в якій реалізовано програмні алгоритми стиснення даних;
- виконано тестування розробленої утиліти і проведено аналіз результатів тестування;
- за результатами тестування побудовано стратегію вдосконалення розробленої утиліти.

Результатом розробки є системна утиліта для компресії даних, яка забезпечує швидке та ємне стиснення даних з метою економії дискового простору. Областю використання можуть бути компанії та організації, що працюють з великим обсягом даних та потребують їх ефективного зберігання.

Ключові слова: компресія даних, алгоритми, стиснення даних без втрат, C++.

ABSTRACT

The bachelor diploma project includes an explanatory note (55 p., 19 fig., list of used literature with 9 titles, 4 appendices).

The object of development of this project is a system utility for data compression. Its purpose is to provide compression of the input file in order to reduce its size.

In the process of work on the diploma project, the following was done:

- existing methods of data compression and implementation of these methods were analyzed;
- defined requirements for the system utility for data compression;
- a utility for data compression was developed, in which software algorithms for data compression are implemented;
- the developed utility was tested and the test results were analyzed;
- based on the results of testing, a strategy for improving the developed utility was built.

The result of the development is a system utility for data compression, which provides fast and high-capacity data compression in order to save disk space. The field of use can be companies and organizations that work with a large amount of data and need its efficient storage.

Keywords: data compression, algorithms, lossless data compression, C++.

[illegible]

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ.....	2
2. ПІДСТАВА ДЛЯ РОЗРОБКИ.....	2
3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ.....	2
4. ДЖЕРЕЛА РОЗРОБКИ	2
5. ТЕХНІЧНІ ВИМОГИ.....	3
1.1 Вимоги до програмного продукту, що розробляється.....	3
1.2 Вимоги до апаратного забезпечення.....	3
1.3 Вимоги до програмного та апаратного забезпечення користувача.....	3
6. ЕТАПИ РОЗРОБКИ	4

					ІАЛЦ.045200.002 ТЗ			
Змін	Арк.	№ докум.	Підпис	Дата				
Розробив		Оридорога М.В.			Системна утиліта для компресії даних Технічне завдання			
Перевірив		Молчанов О.А.						
					Літ. Аркуш Аркушів КПІ ім. Ігоря Сікорського, ФПМ КВ-92			
Н. контроль		Клятченко Я.М.						
Затвердив		Романкевич В.О.						

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ

Назва розробки: «Системна утиліта для компресії даних».

Галузь застосування: персональне застосування, стиснення мережевого трафіку.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи першого (бакалаврського) рівня вищої освіти, затверджене кафедрою системного програмування і спеціалізованих комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ

Метою цього проекту є розробка системної утиліти для компресії даних, яка б виконувала задачу стиснення даних для економії ресурсів, необхідних для зберігання або передачі файлів.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом інформації є технічна та науково-технічна література, технічна документація, публікації в періодичних виданнях та електронні статті у мережі Інтернет.

					ІАЛЦ.045200.002 ТЗ	Арк.
						2
Змін.	Арк.	№ докум.	Підпис	Дата		

5. ТЕХНІЧНІ ВИМОГИ

1.1 Вимоги до програмного продукту, що розробляється

- Ефективність компресії: утиліта повинна забезпечувати якісну компресію даних без втрати їх якості.
- Швидкість роботи: утиліта повинна бути достатньо швидкою, щоб компресувати великі обсяги даних за прийнятний час.
- Масштабованість: утиліта повинна бути здатною працювати з різними типами файлів та обсягами даних, від невеликих файлів до великих обсягів даних.
- Сумісність: утиліта повинна бути сумісною з різними операційними системами та пристроями, що забезпечує її доступність для багатьох користувачів.

1.2 Вимоги до апаратного забезпечення

- Процесор: повинен підтримувати паралельність обчислень.
- Оперативна пам'ять: залежить від об'єму вхідних стискаємих даних.
- Жорсткий диск: повинен мати достатній обсяг пам'яті для збереження результуючого файлу.
- Операційна система: Windows/Linux/macOS.

1.3 Вимоги до програмного та апаратного забезпечення користувача

- Операційна система: користувач повинен мати операційну систему, на якій може працювати програмний продукт.
- Достатній обсяг пам'яті: користувач повинен мати достатній обсяг ОЗУ та жорсткого диска для роботи з програмним продуктом.

					ІАЛЦ.045200.002 ТЗ	Арк.
						3
Змін.	Арк.	№ докум.	Підпис	Дата		

6. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів
1.	Видача завдання на дипломне проектування	04.11.2022
2.	Вивчення літератури за тематикою роботи	16.11.2022
3.	Розроблення та узгодження технічного завдання	25.11.2022
4.	Розроблення структури додатку	17.01.2023
5.	Розробка алгоритму та способу взаємодії	04.02.2023
6.	Програмна реалізація додатку	15.03.2023
7.	Тестування додатку	04.04.2023
8.	Підготовка матеріалів текстової частини проекту	24.04.2023
9.	Підготовка матеріалів графічної частини проекту	17.05.2023
10.	Оформлення технічної документації проекту	28.05.2023

					ІАЛЦ.045200.002 ТЗ	Арк.
Змін.	Арк.	№ докум.	Підпис	Дата		4

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/П	Формат	Позначення	Найменування	Кількість листів	Примітка
1	A4	ІАЛЦ.045200.000	Завдання на дипломний проєкт	2	
2	A4	ІАЛЦ. 045200.001 ОА	Опис альбому	2	
3	A1	ІАЛЦ. 045200.002 ТЗ	Технічне завдання	4	
4	A1	ІАЛЦ. 045200.003 ТП	Відомість технічного проєкту	1	
5	A1	ІАЛЦ. 045200.004 ПЗ	Пояснювальна записка	55	
6	A1	ІАЛЦ. 045200.005 Д1	Системна утиліта для компресії даних. Схема алгоритму	1	
7	A1	ІАЛЦ. 045200.006 Д2	Метод Компресії/Декомпресії Гаффмана. Схема алгоритму	1	
8	A1	ІАЛЦ. 045200.007 Д3	Метод Компресії/Декомпресії Лемпеля-Зіва-Велча. Схема алгоритму	1	
9	A1	ІАЛЦ. 045200.008 Д4	Системна утиліта для компресії даних. Схема структурна	1	

					ІАЛЦ.045200.003 ТП			
Змін	Арк.	№ докум.	Підпис	Дата	Системна утиліта для компресії даних Відомість технічного проєкту	Літ.	Аркуш	Аркушів
Розробив	Оридорога М.В							
Перевірив	Молчанов О.А.						1	1
Н. контроль	Клятченко Я.М.					КПІ ім. Ігоря Сікорського, ФПМ КВ-92		
Затвердив	Романкевич В.О.							

Пояснювальна записка до дипломного проєкту

На тему: системна утиліта для компресії даних

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ	2
ВСТУП.....	4
1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБҐРУНТУВАННЯ ТЕМИ	6
1.1 Роль алгоритмів стиснення даних у сьогоденні.....	6
1.2 Використання алгоритмів стиснення даних.....	7
1.3 Класифікація алгоритмів стиснення даних	11
1.4 Методи компресії без втрат	17
1.5 Висновки до розділу 1	22
2 РОЗРОБЛЕНА УТИЛІТА ДЛЯ КОМПРЕСІЇ ДАНИХ	24
2.1 Обґрунтування використаних інструментів	26
2.2 Структура та взаємодія модулів програми.....	28
2.3 Висновки до розділу 2	42
3 АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ	44
3.1 Тестування на матеріалах Canterbury Corpus	45
3.2 Аналіз отриманих результатів	50
3.3 Висновки до розділу 3	51
ВИСНОВКИ.....	53
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	55

					ІАЛЦ.045200.004 ПЗ		
Зм	Лист	№ докум.	Підп.	Дата			
Розроб.		Оридорога М.В.			Системна утиліта для компресії даних <i>Пояснювальна записка</i>	Літ.	Лист
Перев.		Молчанов О.А.					1
						Листів	55
Н. контр.		Клятченко Я.М.				КПІ ім. Ігоря Сікорського, ФПМ КВ-92	
Затв.		Романкевич В.О.					

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

AES (Advanced Encryption Standard) — це алгоритм симетричного блокового шифрування, який використовується для шифрування.

ANS (Asymmetric Numeral Systems) — метод стиснення даних, заснований на ентропійному кодуванні.

ASCII (Американський стандартний код для обміну інформацією) — це система кодування символів, яка представляє літери, символи та цифри в діапазоні 0-127.

Blowfish — це симетричний ключовий блоковий шифр.

DES (Стандарт шифрування даних) — це ненадійний метод шифрування, який використовує 64-бітний розмір блоку та 56-бітний ключ.

FGK (Faller, Gallager, Knuth) — адаптивний алгоритм кодування Гаффмана, названий на честь вчених, які його впровадили.

FullHD — роздільна здатність медіа контенту 1920x1080 пікселів.

GUI (графічний інтерфейс користувача) — це форма інтерфейсу, яка дозволяє безпосередньо маніпулювати графічними об'єктами.

Header — це частина файлу, яка містить додаткову інформацію для декодування даних.

HTTP (Hypertext Transfer Protocol) — це набір протоколів для Інтернету, які використовуються для передачі гіпертекстових документів.

JPEG (Joint Photographic Experts Group) — це формат стиснення даних, який дозволяє вибрати рівень стиснення, який поєднує якість зображення та використання пам'яті.

KWE (Key Word Encoding) — це алгоритм шифрування, у якому ключ є кодовим словом, що використовується відповідно до вхідного тексту.

LZ — це сімейство алгоритмів стиснення з високою швидкістю декомпресії.

MLP (Multilayer Perceptron) — це нейронна мережа, яку також можна використовувати для стиснення.

Node — це структура даних, яка зберігає значення елемента та вказівник на інший вузол.

PNG (Portable Network Graphics) — це формат зберігання даних, заснований на стисненні без втрат.

RAR — архіватор для операційної системи Windows.

RLE (Run-length encoding) — це алгоритм стиснення без втрат, який допомагає кодувати повторювані символи у вхідному тексті.

UNICODE — це стандарт кодування символів для представлення й обробки текстів у комп'ютерах, що має розширену таблицю символів порівняно з ASCII. Він використовує дві форми кодування: 8-бітну та 16-бітну, залежно від типу даних, які кодуються. Форма кодування за замовчуванням — 16-бітна, де кожен символ має ширину 16 біт (2 байти).

ZIP — це формат стиснення файлів, який використовується для стиснення даних без втрат.

Бінарне дерево — це структура дерева, в якій кожен вузол має не більше двох дочірніх вузлів.

Декодер — це інструмент, що перетворює закодовану інформацію в оригінальне представлення.

Енкодер — це інструмент, який інтерпретує та перетворює інформацію в закодований вигляд.

ООП (об'єктно-орієнтоване програмування) — це парадигма програмування, заснована на концепції класів.

Токен — це послідовність символів у лексичному аналізі (лексема).

ВСТУП

У сучасному світі широко використовуються цифрові технології: великі обсяги даних передаються мережевими каналами зв'язку і обробляються мережевими пристроями. Ці процеси потребують значного використання обчислювальних ресурсів. Задля зменшення використання кількості цих ресурсів, застосовуються різні методи стиснення, які дозволяють зменшити обсяг даних, призначених для передачі або обробки, незалежно від їх типу, наприклад відео, аудіо, зображення, текст та інше. Згідно з законом Мура, кількість транзисторів у мікросхемі подвоюється кожні два роки, що сприяє зросту обсягу інформації, що зберігається на електронних носіях. Проте існує фізичне обмеження розмірів транзисторів, яке обмежує їх розміщення на кристалі. Закон Мура вже кілька років як сповільнив свою дію, і не лише через фізичні обмеження, а ще й через вартість створення технологічних процесів виготовлення схем з меншим розміром транзисторів. Однак, обсяг даних, що передаються мережами, зберігається у сховищах, — лише зростає. Для того, щоб зменшити обсяг пам'яті, необхідної для збереження даних, потрібно застосовувати різні методи ущільнення даних. Саме тому розробка нових методів ущільнення і покращення вже існуючих є актуальною задачею.

Навіть до появи комп'ютерів існували шифри, які дозволяли ефективно передавати інформацію між людьми на відстані, оскільки побутові способи комунікації містять багато надлишкової інформації. До найпоширеніших шифрів належать азбука Морзе (комбінація крапок і тире для кодування символів, де найчастіше використовувані символи мають найкоротші послідовності), шрифт Брайля (рельєфний шрифт для осіб з вадами зору), телефонний зв'язок (виділення високих нот) та інші. Однак, саме наприкінці 80-х років, під час виникнення Інтернету, алгоритми компресії отримали широке застосування.

Однією з характеристик даних є надлишковість, яка залежить від типу даних. Наприклад, зображення мають вищий рівень надлишковості порівняно з

					ІАЛЦ.045200.004 ПЗ	Лист 4
Зм	Лист	№ докум.	Підп.	Дата		

текстом. Тому при стисненні відео чи зображень можна використовувати алгоритми з втратами, які призводять до незначних змін в оригінальному файлі, прискорюють їх передачу та обробку. Основним показником ефективності алгоритмів є ступінь стиснення, який залежить від конкретного алгоритму та вмісту даних. Також важливо враховувати контекст для досягнення більшої ефективності, такі як символи в тексті або попередні пікселі в зображенні.

Камери високої роздільної здатності на супутниках є хорошим прикладом сценарію використання алгоритмів стиснення. Таке рішення дозволяє зменшити вимоги до модулів зберігання на борту супутника. Наприклад, для програм дистанційного зондування, де зберігання даних є дорогим, проте висока якість зображень важлива для наукових досліджень.

Зараз алгоритми компресії або архіватори застосовуються на кожному пристрої під час створення резервних копій, копіювання на інші пристрої та передачі даних через мережу. Найвідоміші з них: ZIP, LZH, LZEXE, RAR, JPEG, PNG, MP3, MPEG та інші. Ці методи використовуються для архівації текстових файлів без втрат, тобто інформація відновлюється у своєму початковому вигляді, а також для відео, аудіо та зображень з втратами даних.

					ІАЛЦ.045200.004 ПЗ	Лист
						5
Зм	Лист	№ докум.	Підп.	Дата		

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБҐРУНТУВАННЯ ТЕМИ

1.1 Роль алгоритмів стиснення даних у сьогоденні

Сьогодні стиснення цифрових даних набуло широкої популярності і концепція ущільнення даних потребує постійного розвитку і вдосконалення. Це спричинено тим, що кількість інформації, яка зберігається на пристроях середньостатистичної людини стрімко зростає. Головна причина застарівання інформації не полягає в плинні часу, а в появі нової інформації, яка нівелює попередню інформацію так, що вона більше об'єктивно не відображає явища і закономірності матеріального світу, людського спілкування та мислення[1]. Поняття інформації має кілька визначень, але в контексті оцінки якості інформації важливо, наскільки вона є новою або неочікуваною для споживача. Додатково, стиснення допомагає прискорити передачу даних, оскільки потрібно менше мережевих пакетів, пам'яті та пропускну здатності мережі. Основоположником науки про стиснення інформації вважають Клода Шеннона[1].

Завдання стиснення складається з двох компонентів (рисунок 1.1):

- компресія (приймає потік даних, на виході генерує стиснуте представлення даних, з меншою кількістю бітів);
- декомпресія (зі стиснутого представлення даних реконструює оригінальне повідомлення або, у випадку компресії з втратами, його наближення).

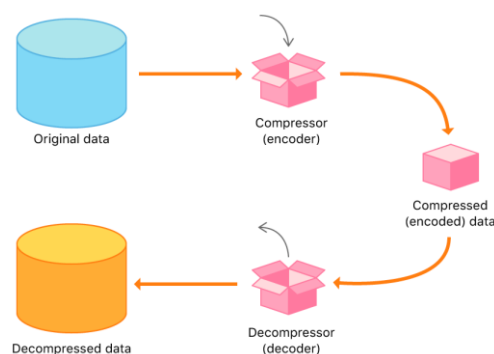


Рисунок 1.1 — Схема компресії/декомпресії даних

У сучасному світі зростає нерозмірність інформації, що перевищує можливості систем та мереж для її передачі. Тому важливо розробляти і впроваджувати нові алгоритми стиснення та ефективно використовувати наявні, щоб зменшити витрати на апаратне забезпечення обчислювальних систем.

Сучасні алгоритми наближаються до Шеннонівської оцінки[1], яка становить близько 1,3 біта на символ. Це означає, що алгоритми намагаються зменшити ентропію для кодування інформації.

Важливо розуміти, що жоден алгоритм не може стиснути всі вхідні повідомлення, оскільки коефіцієнт стиснення або ефективність алгоритму залежать від типу даних. Наприклад, при повторному стисненні однакового рядка він може зменшуватись, але неможливо досягти стиснення до нульового розміру. Інженери також спробували використовувати алгоритми гешування, що зберігають лише геш (хеш) файлу та кількість ітерацій для його відтворення. Однак такі методи мають свої обмеження і проблеми, такі як колізії (дублікати) гешів та збільшений розмір збереженого гешу порівняно з вихідними даними. Тому від цих ідей відмовилися.

Висновок полягає в тому, що розробка і використання алгоритмів стиснення є важливим завданням для оптимізації передачі та зберігання даних, проте існує фізичні обмеження та особливості, які обмежують ступінь стиснення та ефективність цих алгоритмів.

1.2 Використання алгоритмів стиснення даних

Використання алгоритмів стиснення даних є невід'ємною складовою області обробки і передачі інформації. Головною метою цих алгоритмів є ефективне зменшення розміру даних з мінімальною втратою інформації. Широке застосування алгоритмів стиснення даних спостерігається у різних галузях, включаючи зберігання, передачу та обробку великих обсягів даних.

Архівування є важливою галуззю застосування алгоритмів стиснення даних, який включає процес збереження та упакування надлишкових інформаційних ресурсів з метою зниження обсягу зайнятого простору, поліпшення ефективності зберігання даних, а також забезпечення зручного доступу до цих даних у майбутньому. Застосування алгоритмів стиснення даних в процесі архівування дозволяє знизити витрати на зберігання та забезпечує оптимальне використання ресурсів, що є особливо корисним у ситуаціях, де критичним фактором є обмежений доступ до простору зберігання даних.

У сучасному контексті архівування даних, спостерігається використання різноманітних програмних засобів, які пропонують різні підходи до компресії та зберігання інформації. Серед найбільш поширених програм для архівування можна відзначити "WinRAR", "7-Zip" та "WinZip", які базуються на алгоритмах стиснення, таких як DEFLATE, LZMA, або ZIP[3]. Додатково, існують спеціалізовані програми, які використовують формати архівування, такі як "Tar", які поєднуються з різними компресійними алгоритмами, зокрема GZIP[4] та BZIP2. Наведені програми надають користувачам широкий спектр можливостей для ефективного архівування та упакування даних з урахуванням різних вимог та пріоритетів.

Особливістю "tar" є можливість створення архівів без стиснення для збереження структури директорії. Зазвичай tar архіви (tarball) комбінують з різними стискаючими алгоритмами.

"Gzip"[4] використовує безвтратний алгоритм стиснення DEFLATE, який базується на комбінації методів Гаффмана та LZ77. Завдяки своїй швидкодії та ефективності, "gzip" є одним з найпоширеніших програмних засобів стиснення даних.

"Bzip" використовує алгоритм BZIP2, який базується на блочному стисненні даних і забезпечує високу ступінь стискання. Його особливості включають використання адаптивного кодування довжини блоків, а також

відсутність обмежень на розмір блоків, що дозволяє досягти високої ефективності стиснення навіть для великих файлів.

"WinRAR" є однією з найпопулярніших програм для архівування, яка надає широкі можливості для компресії та розпакування файлів. Вона підтримує різні алгоритми стиснення, включаючи DEFLATE, що використовується в форматі ZIP. Крім того, "WinRAR" забезпечує підтримку шифрування даних з використанням алгоритму AES (Advanced Encryption Standard), який є одним з найбільш сучасних і надійних алгоритмів шифрування.

"7-Zip" також є потужним інструментом для архівування, який підтримує різні формати, включаючи 7z, ZIP, TAR та інші. Він використовує алгоритми стиснення, такі як LZMA (Lempel-Ziv-Markov chain Algorithm) і LZMA2, що забезпечують високу ступінь стиснення даних. "7-Zip" також підтримує шифрування за допомогою алгоритмів AES і DES (Data Encryption Standard), які забезпечують безпеку та конфіденційність даних.

"WinZip" є іншою популярною програмою для архівування, яка дозволяє стискувати та розпаковувати файли в різних форматах, зокрема ZIP, ZIPX, RAR та інші. Вона підтримує алгоритми стиснення, такі як DEFLATE і BZIP2, які забезпечують ефективну компресію даних. "WinZip" також забезпечує можливість шифрування файлів з використанням алгоритмів AES, який гарантує захист від несанкціонованого доступу.

Останні три з перелічених програмних засобів, "WinRAR", "7-Zip" та "WinZip", демонструють підтримку шифрування за допомогою алгоритму AES, що забезпечує високий рівень безпеки. Крім того, "WinRAR" та "7-Zip" також підтримують алгоритм шифрування DES, а "WinZip" підтримує шифрування алгоритмом Blowfish. Ці функції шифрування дозволяють захистити конфіденційні дані під час архівування та передачі файлів.

Стиснення даних в мережі є процесом зменшення обсягу даних з використанням різних методів та алгоритмів, що базуються на принципах

компресії. Цей процес спрямований на ефективне використання пропускну́ї здатності мережі та зменшення часу передачі даних.

Мережевому стисненні даних можуть використовуватись технології компресії на рівні протоколів, такі як алгоритми компресії заголовків IP (Internet Protocol) або алгоритми компресії HTTP для передачі веб-сторінок. Технології компресії на рівні протоколів використовуються для стиснення даних безпосередньо на рівні мережевих протоколів, що дозволяє зменшити обсяг передаваних даних і підвищити ефективність мережевої комунікації. Нижче розглянемо декілька таких технологій.

Компресія заголовків IP (IP Header Compression): ця технологія використовується для стиснення IP-заголовків, що передаються в мережі. Вона дозволяє зменшити розмір заголовків IP, що впливає на загальний обсяг передаваних даних. Компресія заголовків IP забезпечується використанням спеціальних алгоритмів, які вилучають непотрібну інформацію та використовують сжатие без втрат даних.

Компресія HTTP (HTTP Compression) (рисунок 1.2): ця технологія застосовується для стиснення даних, що передаються через протокол HTTP. Вона дозволяє зменшити розмір веб-сторінок, що передаються між клієнтом і сервером, забезпечуючи економію пропускну́ї здатності мережі і знижуючи час завантаження сторінок. Компресія HTTP використовує алгоритми стиснення, такі як gzip або deflate, для стискання текстового контенту перед передачею.

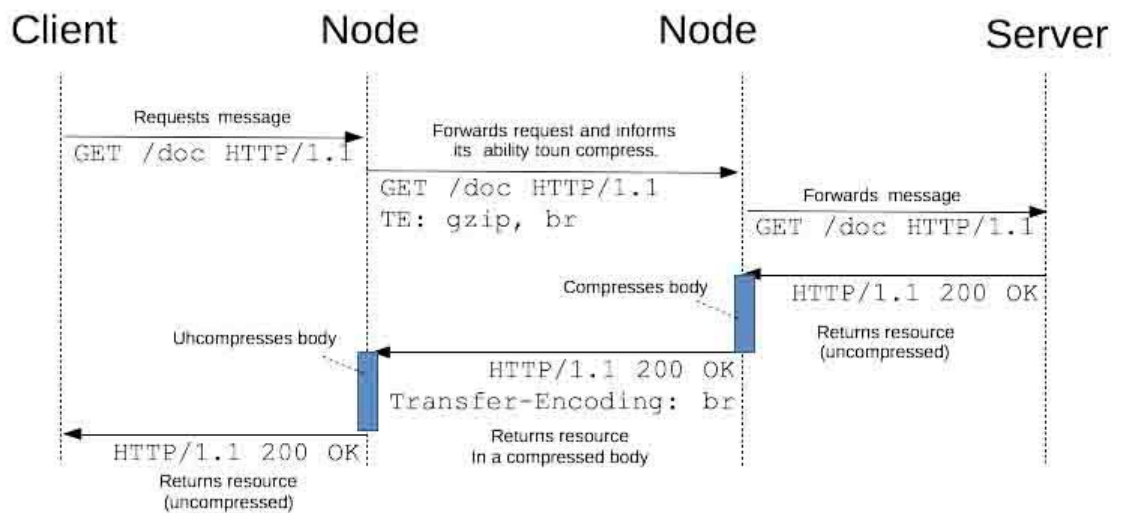


Рисунок 1.2 — Схема компресії/декомпресії HTTP

Компресія на рівні TLS (Transport Layer Security): TLS є протоколом шифрування та забезпечення безпеки в мережевому зв'язку. В деяких випадках TLS може включати можливість компресії даних. За допомогою алгоритмів, таких як DEFLATE, дані можуть бути стиснуті перед шифруванням і передачею по мережі. Проте слід зазначити, що використання компресії на рівні TLS може бути обмежене з міркувань безпеки.

Використання відповідних алгоритмів компресії і налаштування правильних параметрів дозволяє досягти оптимальних результатів в зменшенні обсягу даних і покращенні продуктивності мережі.

1.3 Класифікація алгоритмів стиснення даних

Стиснення даних включає в себе процес перетворення вхідного потоку необроблених даних (біти, коди ASCII, байти або значення пікселів) в інший потік з меншим розміром. Цей потік може бути представлений у вигляді файлу, буфера у пам'яті або іншого типу інформації.

Методи стиснення даних можна класифікувати наступним чином:

— методи без втрат забезпечують точне відновлення оригінального документа зі стиснутої інформації, без втрат жодного біта при

декодуванні (наприклад, методи Run-length, Huffman, LZ77, LZ78 і LZW). Такі методи зазвичай не застосовуються для стиснення медіа контенту;

- методи з втратами зменшують розмір файлу шляхом видалення зайвої інформації. Вони застосовуються до аудіо, зображень і відео, де втрата деякої незначної інформації має незначний вплив на оригінальне повідомлення (наприклад, JPEG, PNG і MPEG). В таких методах відбуваються зміни бітів відносно закодованої та декодованої інформації. Хоча ці зміни можуть бути непомітними для людського сприйняття, такі як контрастність зображення, втрата одного кольору або злиття різних відтінків кольорів. Коли такі дані передаються по мережі або зберігаються на сервері, часто вже не потрібно додатково стискувати, оскільки вони вже мають стислий формат.

Алгоритми стиснення без втрат можуть бути застосовані до будь-якого типу даних, в той час як алгоритми з втратами базуються на певних передумовах. Проте, для деяких даних спотворення (різниця між оригіналом та реконструкцією) зовсім неприпустимі.

Це може стосуватися:

- медичні зображення: у медицині точність зображень, наприклад, рентгенівських знімків, магнітно-резонансної томографії (МРТ) чи комп'ютерної томографії (КТ), є критичною для діагностики та лікування. Застосування алгоритмів стиснення з втратами до медичних зображень може спотворити деталі та важливі ознаки, що може призвести до неправильного розпізнавання або пропуску важливих патологічних ознак;
- аудіо- та відеодані: у деяких випадках точність звукових або відеозаписів може бути критичною, наприклад, у правоохоронних або судових системах, де аудіо- або відеодокази можуть впливати на результати розслідування або судового процесу. Застосування

алгоритмів стиснення з втратами до таких даних може призвести до втрати важливої інформації або зміни звуку та зображення, що може спотворити або унеможливити їх аналіз та використання;

— наукові дані: в наукових дослідженнях та експериментах точність та цілісність даних є вирішальними. Застосування алгоритмів стиснення з втратами до наукових даних, таких як результати експериментів, симуляцій або оброблені дані, може призвести до втрати деталей, значущих для аналізу, та викривлення результатів досліджень.

В інших випадках можна використовувати методи стиснення з втратами разом з методом відновлення інформації. Наприклад, при наявності великого обсягу даних можна використовувати метод відновлення compressive sensing[5], що дозволяє отримати та відновити сигнал, використовуючи знання про його попередні значення, які розрізнені або стиснуті (рисунок 1.3).

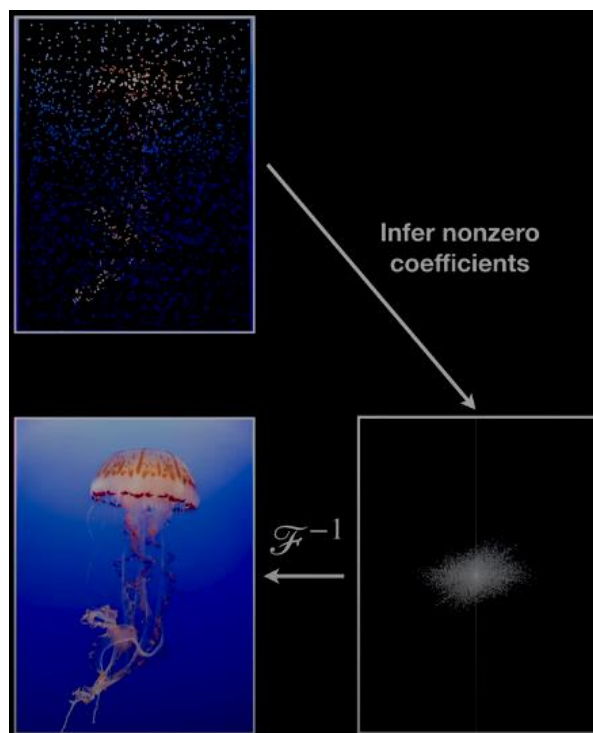


Рисунок 1.3 — Метод compressive sensing

Для відновлення аналогових сигналів існує теорема дискретизації, яка стверджує, що можливо однозначно відновити втрачені дані за умови:

$$f_{\text{дискр}} = 2 * F_{\text{max}}$$

де $f_{\text{дискр}}$ є частотою дискретизації, а F_{max} - максимальною частотою сигналу.

Частота дискретизації визначає кількість сигналів, які отримуються за одну секунду і потрібні для перетворення неперервного аналогового сигналу в дискретний формат. Процес дискретизації полягає в тому, що початковий аналоговий сигнал розбивається на окремі моменти часу (відліки), які представляють його значення у вигляді чисел.

Методи без втрат стиснення можна класифікувати (рисунок 1.4) наступним чином:

- статичне стиснення: кодові слова визначаються заздалегідь перед передачею. Таким чином, одна й та ж інформація буде представлена однаковим кодовим словом у повідомленні;
- адаптивне (або динамічне) стиснення: процес кодування змінюється з часом, адаптуючись до змінних умов. Це дозволяє зменшити середню частоту дискретизації і базується на апроксимації вимірювальних значень функцією часу;
- гібридне стиснення: цей метод поєднує елементи як статичного, так і адаптивного стиснення.

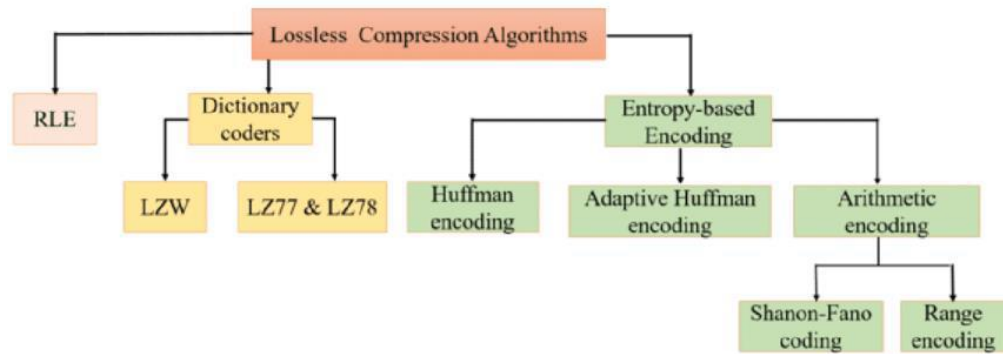


Рисунок 1.4 — Класифікація методів стиснення інформації без втрат

Зображення є масивом даних, де кожна комірка містить інформацію про відповідний колір. Тому, якщо ми кодуємо кольори і яскравість кожного пікселя одним байтом, зображення з роздільною здатністю FullHD займатиме майже два мегабайти. Без стиснення даних ми б не могли завантажувати фільми, оскільки час завантаження був би надто довгим.

Для стиснення з втратами, яке використовується у зображеннях, відео та анімаціях, розглянемо наступні методи[6]:

- фрактальне стиснення: частини зображення, які мають схожі характеристики, перетворюються на фрактальні коди, які використовуються для відтворення початкового зображення;
- трансформоване кодування: використовуються оборотні лінійні коефіцієнти, що відображають перетворення зображення;
- дискретне конусне перетворення: це варіація перетворення Фур'є, яке відображає сигнал у частотній області шляхом представлення його у вигляді відповідних коефіцієнтів. Це перетворення застосовується до кожного кольорового каналу окремо;
- дискретне-вейвлет (хвилькове) перетворення: це розкладання часового ряду на декілька наборів, кожен з яких представляє сигнал відповідно до певного діапазону частот.

При значному стисненні зображення може стати нерозбірливим, тому потрібно збалансувати ступінь стиснення з якістю зображення. Експериментальним шляхом встановлено, що найбільш допустиме стиснення

становить 10:1, оскільки при більшому стисненні стає складно розрізнити деталі на зображенні (рисунок 1.5).

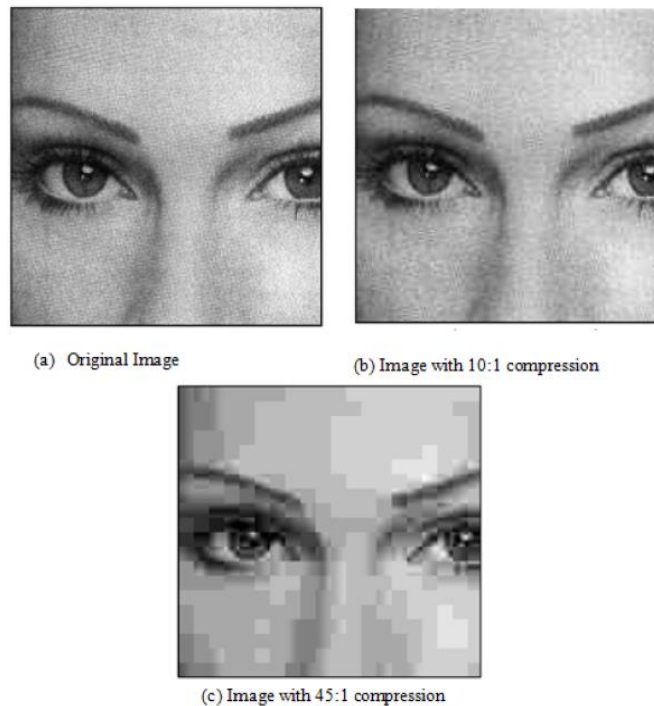


Рисунок 1.5 — Приклад стиснення зображення з втратами

Класифікація алгоритмів стиснення даних може бути здійснена на основі способу їх реалізації. Одним із способів класифікації алгоритмів є методи, які вони використовують, наприклад перетворення потоку, статичні методи та блокове перетворення.

Розглянемо кожен із цих методів докладніше:

- перетворення потоку: цей метод стиснення передбачає аналіз і обробку безперервного потоку даних без розбиття його на окремі блоки. Дані обробляються послідовно за допомогою таких методів, як алгоритми Лемпеля-Зіва-Велча (LZW) або кодування довжини серії. Цей підхід зазвичай використовується для стиснення без втрат;
- статичні методи: такі методи зазвичай використовують декілька проходів і передбачають використання фіксованого алгоритму стиснення для всього набору даних. Він спирається на статистичні

властивості даних і часто застосовується до текстових даних. Наприклад, для статичного стиснення можна використовувати алгоритм Гаффмана, Шеннона-Фано або арифметичне кодування[7];

— перетворення блоків: цей метод ділить вхідні дані на блоки фіксованого розміру та застосовує стиснення до кожного блоку незалежно. Типові алгоритми, що використовуються для блокового перетворення, включають алгоритми стиснення з втратами, такі як JPEG, PNG для стиснення зображення або MP3 для стиснення аудіо. Ефективність стиснення в цьому методі може залежати від розміру блоку та характеристик даних.

1.4 Методи компресії без втрат

В цьому дипломному проєкті використовуються лише алгоритми стиснення без втрат, тому є доцільним розглянути це поняття і виокремити основні з них.

Методи компресії без втрат можна використовувати окремо або комбінувати для досягнення оптимального стиснення на основі типу даних і бажаного рівня точності. Кожен підхід має свої переваги та обмеження, і вибір конкретного методу залежить від конкретної програми та типу даних, які потрібно стиснути.

Розглянемо найбільш поширені алгоритми[8].

Кодування повторів (RLE) є простим методом стиснення даних, де послідовності повторюваних символів або блоків замінюються одним символом та числом повторень. Наприклад, рядок "AAABBBCCC" може бути закодований як "A3B3C3".

Переваги: RLE простий у реалізації та має низьку обчислювальну складність. Він ефективний для даних з великою кількістю повторюваних символів або блоків, таких як зображення зі значними площинами однакового кольору.

Недоліки: RLE має невелику ступінь стиснення для даних з низькою повторюваністю або випадковими символами. Він не підходить для стиснення даних, де повторюваність не переважає над самими символами, наприклад, для випадкових текстів.

Алгоритм Гаффмана використовує статистичний підхід до стиснення даних, в якому будується оптимальний префіксний код, де часті токени отримують короткі біти, а рідкі - довші. Цей ефект досягається шляхом використання бінарного дерева Гаффмана, де частоти токенів визначають їх положення у дереві.

Переваги: алгоритм Гаффмана забезпечує ефективне стиснення для даних з нерівномірною частотою символів. Він здатний досягти оптимальної ступені стиснення, де кожен символ представлений набором бітів, які відповідають його частоті появи у дані;

Недоліки: побудова бінарного дерева Гаффмана вимагає попереднього перегляду всього набору даних для обчислення частот символів. Це може бути недоцільним для потокових даних або даних невідомого розміру. Додатково, для ефективного використання, кодування Гаффмана потребує зберігання таблиці кодів, що може займати додаткове місце.

Також треба зазначити, що існують адаптації алгоритму Гаффмана.

Алгоритм FGK — це модифікований варіант класичного алгоритму Гаффмана для стиснення даних. Цей алгоритм був розроблений Фаллером, Геледжером та Кнудом і відрізняється від звичайного алгоритму Гаффмана тим, що побудова оптимального префіксного коду відбувається не наперед заданою множиною символів, а поступово, адаптивно, з кожним новим символом, що з'являється.

Робота: алгоритм розпочинається зі створення початкового дерева, що містить символ з найбільшою частотою появи. З кожним новим символом, що з'являється, алгоритм змінює дерево, розширюючи, змінюючи або зливаючи гілки дерева в залежності від значень частот символів. Частоти символів

оновлюються з кожним новим символом. Це дозволяє алгоритму адаптуватися до змін у вхідних даних і підтримувати оптимальні префіксні коди.

Переваги: алгоритм може працювати зі змінними частотами символів і адаптуватися до змін у вхідних даних, що дозволяє досягти кращої компресії. За рахунок побудови оптимальних префіксних кодів на основі частот символів, алгоритм FGK забезпечує ефективну компресію даних.

Недоліки: оскільки алгоритм FGK адаптивний, він може вимагати більше обчислювальних ресурсів порівняно з неадаптивними алгоритмами стиснення. Для побудови і підтримки дерева символів з їх частотами, алгоритм FGK вимагає додаткової пам'яті для збереження цієї інформації.

Залежність від порядку символів: порядок, в якому символи з'являються у вхідних даних, може вплинути на ефективність стиснення алгоритмом FGK. При несприятливому порядку може виникнути менш ефективне стиснення.

Алгоритм Шеннона-Фано також використовує статистичний підхід до стиснення даних, проте він відрізняється від алгоритму Гаффмана у способі побудови коду. В алгоритмі Шеннона-Фано, множина токенів рекурсивно розбивається на підмножини зі схожою ймовірністю виникнення, і символам присвоюються коди з різною довжиною, таким чином забезпечуючи рівномірність префіксних кодів.

Переваги: алгоритм Шеннона-Фано ефективний для даних зі збалансованою ймовірністю символів. Він має просту рекурсивну структуру та дозволяє швидше кодування та декодування, порівняно з алгоритмом Гаффмана.

Недоліки: алгоритм Шеннона-Фано не досягає оптимальної ступені стиснення, як алгоритм Гаффмана, особливо для даних з нерівномірною частотою символів. Він може мати деякий надмір кодування символів з меншою ймовірністю, що зменшує ступінь стиснення.

Алгоритм KWE (Кодування з використанням експоненційних функцій) базується на математичній концепції експоненційних функцій. Він

використовує функції для заміни повторюваних фрагментів даних з деякими експоненційними шаблонами, що зменшують загальний обсяг даних.

Переваги: алгоритм KWE може забезпечити високу ступінь стиснення для певних видів даних, особливо для даних з великою кількістю повторюваних шаблонів. Він може бути ефективним при стисненні текстових даних або даних з певними локальними шаблонами.

Недоліки: алгоритм KWE вимагає складних обчислень для знаходження оптимальних експоненційних шаблонів та їх заміни. Він не підходить для всіх типів даних і може бути менш ефективним для даних з низькою повторюваністю або неправильними шаблонами.

Алгоритм LZW є словарним методом стиснення даних, який побудований на основі попереднього використання існуючих токенів для кодування нових токенів. Вхідні дані поділяються на послідовні фрагменти, які додаються до словника, а кодові слова надсилаються як вихідні дані.

Переваги: алгоритм LZW ефективний для стиснення текстових даних, особливо для документів з великим словником часто використовуваних слів або токенів. Він здатний досягти хорошої ступені стиснення без втрати даних.

Недоліки: алгоритм LZW потребує попереднього побудови та підтримки словника, що може призводити до великого обсягу пам'яті для зберігання даних. Він також має певну втрату стиснення при роботі з даними, які не мають повторюваних токенів або великого словника.

Алгоритм MLP (Многошаровий персептрон) використовує штучні нейронні мережі з багатьма шаровими вузлами для стиснення та відновлення даних. Вхідні дані подаються на вхідний шар мережі, проходять крізь приховані шари з функціями активації та на виході отримуються стиснуті дані.

Переваги: алгоритм MLP може забезпечити ефективне стиснення для широкого спектру даних, включаючи текст, зображення та звук. Він може враховувати складні залежності між даними та здатний досягти високої ступені стиснення при використанні великої нейронної мережі.

Недоліки: алгоритм MLP вимагає значних обчислювальних ресурсів та тривалого часу для тренування нейронної мережі. Він може бути складним у реалізації та вимагати досвідченого розробника для досягнення оптимальних результатів. Крім того, декодування стиснених даних також може бути витратним з точки зору обчислень.

Алгоритм DEFLATE є комбінацією алгоритмів стиснення LZ77 і Гаффмана. Він використовує поєднання методів заміни повторюваних фрагментів даних і побудови оптимальних префіксних кодів для досягнення стиснення.

Переваги: алгоритм DEFLATE є одним з найефективніших і поширених алгоритмів стиснення. Він може забезпечити високу ступінь стиснення для різноманітних типів даних і використовує комбінацію різних методів для досягнення оптимальних результатів.

Недоліки: алгоритм DEFLATE може вимагати великого обсягу обчислювальних ресурсів для кодування та декодування даних. Він також може бути трохи повільним у порівнянні з іншими алгоритмами стиснення, особливо для великих обсягів даних. Крім того, його ефективність може залежати від специфіки вхідних даних і їх повторюваності.

Кожен з наведених алгоритмів має свої переваги та недоліки, і вибір певного алгоритму залежить від типу даних, розміру файлу, вимог до швидкості обробки та розміру стисненого файлу. Наприклад, алгоритми RLE та Шеннона-Фано можуть бути ефективними для простих типів даних, таких як повторювані символи або текстові рядки. Алгоритми Гаффмана, LZW та DEFLATE зазвичай забезпечують високу ступінь стиснення для різноманітних типів даних, включаючи тексти, зображення та відео. Алгоритми KWE та MLP можуть бути використані для специфічних випадків, де є повторювані шаблони або складні залежності між даними.

1.5 Висновки до розділу 1

У даному розділі досліджено та проаналізовано актуальність теми. Зазначено, що стиснення цифрових даних стало широко поширеним явищем завдяки збільшенню обсягу інформації, яка зберігається на пристроях користувачів. Внаслідок цього, галузь стиснення даних постійно розвивається та вдосконалюється. Ключовою постаттю в науці про стиснення інформації є Клод Шеннон, який зробив значний внесок у цю галузь. Розглянуто, що стиснення даних складається з двох основних компонентів: компресії та декомпресії, як показано на рисунку 1.1.

Звернена увага на тому, що розробка та використання алгоритмів стиснення є важливим завданням для оптимізації передачі та зберігання даних. Однак, існують фізичні обмеження та особливості, які впливають на ступінь стиснення та ефективність цих алгоритмів.

Розкрита тема використання алгоритмів стиснення даних у багатьох сферах, включаючи архівування і компресію даних у мережах.

Також детально розглянуті наступні програмні засоби, такі як "tar", "Gzip", "Bzip", "WinRAR", "7-Zip" та "WinZip", які є популярними інструментами для архівації та стиснення даних. Деякі з цих програмних засобів також підтримують шифрування для забезпечення безпеки даних під час їх зберігання та передачі.

Виокремлені інструменти стиснення даних в мережевому середовищі стиснення даних, які є важливим засобом зменшення обсягу даних, що передаються через обмежені пропускні здатності каналу передачі даних. Поширеними алгоритмами стиснення в Інтернеті є "gzip" та "deflate". Вони використовуються для стиснення веб-сторінок перед їх передачею на клієнтський браузер, що дозволяє скоротити час завантаження сторінок та зменшити споживання пропускної здатності.

Розглянуто не менш поширеніші методи стиснення даних, що використовуються на рівні протоколів: компресія заголовків IP, компресія HTTP, компресія на рівні TLS.

Проведений систематичний огляд різних методів стиснення даних. Він вказує, що стиснення даних передбачає перетворення вихідного потоку необроблених даних в інший потік з меншим обсягом. Зазначено, що методи стиснення даних можна класифікувати як методи без втрат та методи з втратами. Також вказано на класифікацію підходів застосовуваних в алгоритмах компресії: статичне стиснення, адаптивне, гібридне стиснення.

Розглянуто методи компресії, які застосовують для стиснення з втратами: Фрактальне стиснення, Трансформоване кодування, Дискретне конусне перетворення, Дискретне-вейвлет (хвилькове) перетворення. Вказано, що для деяких типів даних, які не допускають жодного спотворення, рекомендується використовувати методи без втрат, в той час як методи з втратами можуть бути застосовані з обережністю та залежно від вимог до якості.

Детально описано різні методи стиснення даних без втрат. Проаналізовано та порівняно найпоширеніші алгоритми компресії без втрат, зокрема такі як: кодування повторів (RLE), алгоритм Гаффмана, алгоритм FGK, алгоритм Шеннона-Фано, алгоритм KWE, алгоритм LZW, алгоритм MLP та алгоритм DEFLATE.

В цьому розділі проведено чіткий огляд класифікації та безвтратних методів стиснення даних, визначаючи основні поняття, що характеризують процес стиснення даних.

2 РОЗРОБЛЕНА УТИЛІТА ДЛЯ КОМПРЕСІЇ ДАНИХ

У дипломному проекті розроблена програма, яка пропонує ефективний засіб об'єднання алгоритмів компресії даних, що мають спільний програмний інтерфейс, і виконує їх паралельно. Ця програма запускає кожен наданий алгоритм компресії в окремому потоці, що дозволяє досягти максимальної продуктивності.

Завдяки використанню паралельного виконання алгоритмів компресії, програма забезпечує стиснення даних з найменшим можливим розміром вихідного файлу. Цей підхід дозволяє зберігати значні обсяги даних і ефективно використовувати ресурси для передачі або зберігання інформації.

Використовуючи наведену утиліту, користувачі зможуть зменшити обсяг даних, з якими працюють, і відчутно знизити витрати на зберігання та передачу інформації. Компресія даних стає надзвичайно важливою в сучасному світі, де об'єми даних постійно зростають, і використання такої програми може стати вирішальним фактором у забезпеченні ефективного управління інформацією.

Цей дипломний проект відкриває нові перспективи в галузі компресії даних і демонструє успішність результатів програми, що об'єднує алгоритми компресії даних. Впровадження такої програми може принести значні переваги в областях, де ефективне управління даними відіграє важливу роль, таких як мережеві технології, зберігання даних та обробка інформації.

Цей дипломний проект пропонує програму, яка має значні технічні переваги в галузі компресії даних.

Ось деякі з цих переваг:

- паралельне виконання: програма запускає кожен алгоритм компресії в окремому потоці, що дозволяє використовувати всю потужність багатопроцесорних систем і забезпечує швидкість обробки даних. Це дозволяє досягти значного збільшення продуктивності та скорочення часу компресії;

- мінімальний розмір файлу: завдяки паралельному виконанню алгоритмів компресії, програма генерує файл з найменшим можливим розміром. Це дозволяє зберігати значні обсяги даних на носіях з обмеженим простором або зменшити витрати на зберігання;
- універсальний інтерфейс: програма об'єднує алгоритми компресії даних з однаковим інтерфейсом, що дозволяє легко додавати нові алгоритми або модифікувати існуючі. Це забезпечує гнучкість та розширюваність системи, а також спрощує її використання для різних типів даних і сценаріїв застосування;
- ефективне використання ресурсів: використання паралельного виконання алгоритмів компресії дозволяє ефективно використовувати ресурси системи. Це означає, що програма працює швидко і споживає менше процесорного часу та пам'яті, що є важливими факторами в області обробки великих обсягів даних;
- висока якість стиснення: алгоритми компресії, які використовуються в програмі, мають високу якість стиснення даних. Вони здатні ефективно виявляти та усувати зайву інформацію, що дозволяє досягти значного зменшення розміру файлів без значних втрат в якості даних;
- покращена передача та зберігання даних: використання наведеної програми дозволяє значно зменшити обсяг даних, що передаються по мережі або зберігаються на носіях. Це забезпечує економію пропускну здатності мережі та простору для зберігання даних, що є важливими перевагами в умовах обмежених ресурсів.

Загалом, наведена утиліта з паралельним використанням декількох алгоритмів компресії даних демонструє значне зменшення об'єму файлу.

2.1 Обґрунтування використаних інструментів

Для розробки була використана мова програмування C++, оскільки вона успадковує багато корисних інструментів від низькорівневої мови C і дозволяє працювати з даними на найнижчому рівні абстракції, що є необхідним для реалізації ефективного алгоритму компресії. У порівнянні з мовою C, C++ відрізняється тим, що вона є об'єктно-орієнтованою мовою і має підтримку об'єктів та класів. C++ є крос-платформовою мовою середнього рівня, розробленою Б'ярном Страуструпом[9]. Додатки, написані на цій мові, відзначаються високою продуктивністю, оскільки розробник має контроль над ресурсами та пам'яттю.

C++ також піддається компіляції безпосередньо до машинного коду цільової системи перед виконанням, що сприяє високій продуктивності програм, написаних на цій мові. Вона широко використовується для написання алгоритмів, які вимагають високої швидкості, оскільки має прямий доступ до пам'яті. Крім того, C++ застосовується у написанні компіляторів, рушіїв для ігор, графічних програм, які потребують швидкого відображення, а також в банківській сфері для обробки мільйонів транзакцій та в інших високонавантажених системах.

Розглянемо обґрунтування використання мови C++ для реалізації програми, яка забезпечує багатопотокову компресію даних, на основі науково-технічних факторів. Вибір мови C++ обумовлений низкою причин, що відображають її переваги та придатність для даного контексту.

По-перше, важливо зазначити, що мова C++ володіє високою ефективністю та продуктивністю, що є критичними факторами при роботі з обчислювальними ресурсами. Вона надає можливість оптимізувати алгоритми, забезпечуючи швидке виконання завдань і низьку накладну витрату. Це особливо актуально в багатопотоковому середовищі, де кожен потік вимагає швидкого та ефективного оброблення даних.

По-друге, мова C++ дозволяє ефективно працювати з пам'яттю та маніпулювати даними, що є необхідними у процесі компресії даних. Завдяки розширеним можливостям, які надає мова C++, можна реалізувати оптимальні алгоритми компресії, що вимагають роботи з великими обсягами даних та швидкого доступу до них. Це дозволяє досягти високої продуктивності та ефективності при компресії даних.

По-третє, мова C++ підтримує об'єктно-орієнтоване програмування, що сприяє структуруванню програмного коду на модулі та класи. Це сприяє підвищенню читабельності, модульності та повторному використанню коду. Використання об'єктно-орієнтованого підходу полегшує реалізацію алгоритмів компресії даних та їх керування.

Також, однією з причин вибору мови C++ стала її модель пам'яті[9], представлена в стандарті мови C++11 (рисунок 2.1). Її переваги для програми, яка реалізує багатопотокову компресію даних, є можливість ефективної роботи з низькорівневими операціями над пам'яттю. Мова C++ надає доступ до вказівників та дозволяє контролювати розміщення та використання даних у пам'яті. Це сприяє ефективній управлінню пам'яттю та дозволяє використовувати оптимізовані структури даних для збереження та обробки інформації.

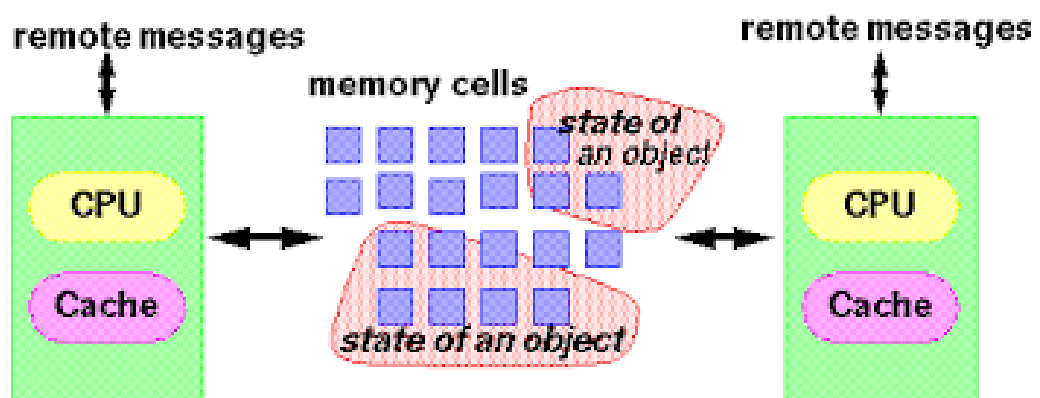


Рисунок 2.1 — Модель пам'яті C++

Крім того, модель пам'яті C++ підтримує статичне та динамічне виділення пам'яті, що дозволяє ефективно керувати розміром та розподілом пам'яті в процесі компресії даних. Це особливо корисно при обробці великих обсягів даних, оскільки програма може ефективно використовувати ресурси пам'яті та уникати переповнення або надмірного використання пам'яті.

Крім того, модель пам'яті C++ підтримує роботу з покажчиками, що дозволяє ефективно передавати та обробляти дані між різними потоками в процесі багатопотокової компресії даних. Це забезпечує можливість швидкого та безперервного обміну даними між потоками та допомагає уникнути конфліктів та перекриття даних.

Таким чином, модель пам'яті C++ забезпечує багато переваг для програми, яка виконує багатопоточну компресію даних, включаючи ефективну роботу з низькорівневими операціями над пам'яттю, можливість ефективного керування пам'яттю та оптимізованого використання ресурсів, а також підтримку передачі та обробки даних між потоками.

Отже, вибір мови C++ для реалізації програми, яка виконує багатопотокову компресію даних, обумовлений її високою ефективністю, розширеними можливостями роботи з пам'яттю та обробки даних, а також підтримкою об'єктно-орієнтованого програмування. Це дозволяє досягти оптимальної продуктивності та ефективності в процесі багатопотокової компресії даних.

2.2 Структура та взаємодія модулів програми

В даному розділі роботи розглядається структура та взаємодія модулів програми, призначеної для стиснення даних. Забезпечення ефективного та надійного стиснення даних вимагає ретельного планування архітектури програмного забезпечення. Відповідно, цей розділ присвячений докладному

опису структури програми, яка включає різні модулі та компоненти, що взаємодіють між собою з метою виконання процесу стиснення даних.

Спочатку будуть розглянуті окремі модулі програми, їх реалізацію та призначення. Кожен модуль буде проаналізований з точки зору його ролі та вкладу у процес стиснення даних. Подальше дослідження буде сфокусоване на взаємодії між модулями та способах обміну даними між ними. Аналіз взаємодії модулів дозволить отримати більш повне уявлення про внутрішню структуру програми та способи взаємодії її компонентів.

Метою цього розділу є розкриття архітектури програми для стиснення даних та з'ясування взаємозв'язків між модулями, що допоможе зрозуміти принципи її роботи. Результати дослідження структури та взаємодії модулів програми створять основу для подальшого вдосконалення алгоритмів стиснення даних та оптимізації процесу компресії.

У рамках представленого дослідження розроблено два власних алгоритми стиснення даних, що базуються на відомих алгоритмах Гаффмана та Лемпеля-Зіва Велча. Ці алгоритми були розроблені з метою наглядної демонстрації принципів роботи утиліти, яка дозволяє розпаралелити декілька алгоритмів стиснення даних задля знаходження оптимального алгоритму, який задовольнятиме вимоги користувача. Застосовуючи ці алгоритми, утиліта може виконувати одночасну роботу декількох алгоритмів стиснення, які мають спільний інтерфейс. Кожен алгоритм стиснення запускається в окремому потоці, що забезпечує ефективне використання обчислювальних ресурсів та прискорення процесу стиснення даних. Завдяки такому підходу, утиліта може проводити порівняльний аналіз різних алгоритмів стиснення та вибирати оптимальний алгоритм на основі критеріїв, які задаються користувачем. Це дозволяє забезпечити максимальну ефективність стиснення даних з урахуванням вимог до швидкодії та розміру вихідного файлу.

Перший алгоритм є модифікацією алгоритму Гаффмана, який використовує бінарне дерево Гаффмана для створення оптимального кодового

словника на основі частоти вхідних символів. Застосовуючи цей алгоритм, дана реалізація стискає дані шляхом заміни повторюваних символів бітовими кодами, що використовуються в створеному кодовому словнику. На початку скомпресованого файлу зберігається Header, який містить частоту кожного символу. Це потрібно для побудови бінарного дерева при декодуванні. Частота кожного символу зберігається в наступному форматі: 8 біт займає символ, 2 біти сигналізують розмір змінної, яка потрібна для зберігання частоти, наступною зберігається частота присутності символу в файлі, яка займає стільки байт, скільки вказано в минулих двох бітах. Така структура Header`а потрібна для економії місця. З використанням зазначеного вище алгоритму, досягається ефективна компресія даних зі збереженням їх структури.

Другий алгоритм, заснований на алгоритмі Лемпеля-Зіва-Велча, використовує безвтратну компресію даних шляхом створення словника з безперервних токенів вхідного тексту. Застосовуючи цей алгоритм, повторювані токени замінюються відповідними індексами у створеному словнику, що дозволяє зберігати дані з меншим обсягом і забезпечує високу ступінь стиснення.

Обидва розроблені алгоритми стиснення базуються на відомих методах, але вони були модифіковані та оптимізовані для досягнення кращих результатів у відповідності з конкретними вимогами та характеристиками даних. Ці алгоритми є цінним внеском у галузь компресії даних та можуть бути використані для ефективного стиснення різноманітних типів даних, зокрема текстових.

В даному науковому контексті розроблена реалізація алгоритмів, що базується на кодуванні/декодуванні Гаффмана (рисунк 2.2). Алгоритм використовується для стиснення та декодування даних з метою зменшення обсягу зберігання і передачі інформації.

```

11 namespace NHuffman
12 {
13     using namespace std;
14
15     enum class SIZE_IN_BYTES;
16     enum class BITS_LEFT;
17
18     class HuffmanBase
19     {
20     protected:
21         struct Node{ ... };
22
23         struct Comparator{ ... };
24
25         using node_pointer_t = shared_ptr<Node>;
26         void BuildQueue();
27         void BuildTree();
28         void AssignCodes(node_pointer_t root, size_t depth = 0, string path = "");
29
30         node_pointer_t m_TreeRoot = nullptr;
31         vector<uint8_t> m_output;
32         map<uint8_t, size_t> m_nodes;
33         map<uint8_t, string> m_codes;
34         priority_queue<node_pointer_t, vector<node_pointer_t>, Comparator> m_queue;
35     };
36
37     class Encoder : public HuffmanBase
38     {
39     public:
40         Encoder(const vector<uint8_t>& input);
41         const vector<uint8_t>& GetOutput() const { return m_output; }
42
43     private:
44         void CountFrequencies();
45         string EncodeInput();
46         string WriteHeaderData();
47
48         const vector<uint8_t>& m_input;
49     };
50
51     class Decoder : public HuffmanBase
52     {
53     public:
54         Decoder(const vector<uint8_t>& input);
55         const vector<uint8_t>& GetOutput() const { return m_output; }
56
57     private:
58         void ReadHeaderData();
59         size_t ReadFromInput(SIZE_IN_BYTES);
60         void DecodeData();
61         uint8_t SearchForCode(node_pointer_t root, size_t& steps);
62
63         stringstream m_input;
64     };
65
66     void compress(const vector<uint8_t>& input, vector<uint8_t>& out);
67     void decompress(const vector<uint8_t>& input, vector<uint8_t>& out);
68 }

```

Рисунок 2.2 — Реалізація алгоритмів кодування/декодування Гаффмана

Реалізація включає низку компонентів, кожен з яких виконує конкретну функцію в процесі кодування та декодування. Використовуються різні класи, перерахування та функції для ефективної обробки та представлення даних.

Основні компоненти алгоритму включають:

Перерахування `SIZE_IN_BYTES` та `BITS_LEFT`, які використовуються для визначення розміру та кількості бітів, що залишилися для обробки.

Константні словники `size_map` та `size_to_value_map`, які містять відповідні значення розміру в байтах для кодування та декодування.

Функції `convertToSizeInBytes` та `write_value_to_string`, які використовуються для конвертації та запису значень в рядок відповідного розміру.

Клас `Encoder`, що забезпечує кодування вхідних даних за допомогою алгоритму Гаффмана. Він включає методи для підрахунку частоти символів, побудови черги пріоритетів, побудови бінарного дерева Гаффмана, присвоєння кодів символам, запису заголовкових даних та кодування вхідних даних.

Клас `Decoder`, що забезпечує декодування закодованих даних за допомогою алгоритму Гаффмана. Він включає методи для читання `Header`у`, побудови черги пріоритетів, побудови бінарного дерева Гаффмана, декодування даних та отримання вихідних даних.

Функції `compress` та `decompress`, які використовуються для стиснення та декодування даних відповідно.

Клас `HuffmanBase`, використований у наданому коді (рисунок 2.3), має важливе призначення в контексті реалізації алгоритму стиснення Гаффмана. Цей клас є основою для побудови структур даних та алгоритмів, необхідних для виконання компресії та декомпресії даних згідно з цим методом стиснення.

Головна мета класу `HuffmanBase` полягає в забезпеченні основних елементів та методів для виконання кодування та декодування з використанням алгоритму Гаффмана. Цей клас визначає та керує такими важливими аспектами, як побудова бінарного дерева Гаффмана, генерація кодів символів, зберігання статистики частоти символів, а також методи для стиснення та декомпресії даних.

```

18 class HuffmanBase
19 {
20 protected:
21 struct Node
22 {
23     Node(const size_t& frqnc, const vector<uint8_t>& v1,
24         const shared_ptr<Node>& lft = nullptr, const shared_ptr<Node>& right = nullptr)
25         : frequency(frqnc), value(v1), left(lft), right(right) {}
26
27     size_t frequency = 0;
28     vector<uint8_t> value;
29
30     shared_ptr<Node> left = nullptr;
31     shared_ptr<Node> right = nullptr;
32
33     bool IsLeaf() const
34     {
35         return !left && !right;
36     }
37 };
38
39 struct Comparator
40 {
41     bool operator()(const shared_ptr<Node>& a, const shared_ptr<Node>& b) const
42     {
43         return a->frequency > b->frequency;
44     }
45 };
46
47 using node_pointer_t = shared_ptr<Node>;
48 void BuildQueue();
49 void BuildTree();
50 void AssignCodes(node_pointer_t root, size_t depth = 0, string path = "");
51
52 node_pointer_t m_TreeRoot = nullptr;
53 vector<uint8_t> m_output;
54 map<uint8_t, size_t> m_nodes;
55 map<uint8_t, string> m_codes;
56 priority_queue<node_pointer_t, vector<node_pointer_t>, Comparator> m_queue;
57 };

```

Рисунок 2.3 — Реалізація базового класу для кодування/декодування Гаффмана

Зокрема, клас HuffmanBase включає реалізацію алгоритмів для побудови бінарного дерева Гаффмана на основі вхідних даних, розрахунку частоти появи кожного символу, визначення кодів символів та їх збереження. Ці функції виконуються з урахуванням принципів, що лежать в основі алгоритму Гаффмана, забезпечуючи оптимальну розподіленість бітових кодів в залежності від частоти входження символів.

Клас також містить методи, що використовуються для компресії та декомпресії даних, базуючись на використанні побудованого бінарного дерева

Гаффмана, з якого виводяться коди символів для виконання стиснення та відновлення оригінальних даних відповідно. Ці методи дозволяють зменшити розмір даних шляхом заміни символів їхніми стислими кодами та повернути початкові дані шляхом декодування цих кодів з використанням бінарного дерева Гаффмана.

Отже, клас HuffmanBase відіграє ключову роль у реалізації алгоритму стиснення Гаффмана, надаючи необхідні інструменти та функції для побудови, управління та використання дерева Гаффмана, а також виконання компресії та декомпресії даних. Його використання є важливим для забезпечення ефективного стиснення та відновлення інформації за допомогою алгоритму Гаффмана в різних областях, де зменшення обсягу даних та оптимізація передачі є важливими факторами.

Клас Encoder, представлений у наданому коді (рисунок 2.4), має важливе призначення у контексті реалізації алгоритму стиснення Гаффмана. Цей клас виконує конкретні функції, необхідні для здійснення процесу кодування даних з використанням алгоритму Гаффмана, який забезпечує ефективне стиснення та оптимізовану передачу інформації.

Головна мета класу Encoder полягає в перетворенні вхідних даних в коди Гаффмана, що є скомпресованим представленням цих даних. Цей процес включає такі критичні етапи, як аналіз частоти входження символів у вхідних даних, побудова бінарного дерева Гаффмана на основі цих статистичних даних, визначення унікальних кодів для кожного символу та створення вихідного закодованого потоку бітів.

```

59 class Encoder : public HuffmanBase
60 {
61 public:
62     Encoder(const vector<uint8_t>& input);
63     const vector<uint8_t>& GetOutput() const { return m_output; }
64
65 private:
66     void CountFrequencies();
67     string EncodeInput();
68     string WriteHeaderData();
69
70     const vector<uint8_t>& m_input;
71 };
72

```

Рисунок 2.4 — Реалізація класу для кодування Гаффмана

Клас Encoder володіє методами, які забезпечують виконання цих завдань з високою ефективністю та точністю. Він включає реалізацію алгоритмів для акумулювання статистики частоти входження символів та побудови оптимального бінарного дерева Гаффмана з використанням цих статистичних даних. Крім того, клас надає можливість визначення кодів для кожного символу, відповідно до його місця у дереві Гаффмана, та генерацію бітового потоку, що представляє стислі дані.

Використання класу Encoder у контексті алгоритму Гаффмана дозволяє досягти важливих результатів у стисненні даних. Він використовує сильні математичні основи, що базуються на статистичних даних та оптимальному розподілі бітових кодів, для забезпечення високого ступеня стиснення та ефективності при збереженні та передачі інформації.

Клас Decoder, який можна знайти у наданому коді (рисунок 2.5), відіграє важливу роль у контексті реалізації алгоритму декомпресії Гаффмана. Його призначення полягає в відновленні початкових даних зі стислого представлення, яке отримано під час кодування Гаффмана.

```

73 class Decoder : public HuffmanBase
74 {
75 public:
76     Decoder(const vector<uint8_t>& input);
77     const vector<uint8_t>& GetOutput() const { return m_output; }
78
79 private:
80     void ReadHeaderData();
81     size_t ReadFromInput(SIZE_IN_BYTES);
82     void DecodeData();
83     uint8_t SearchForCode(node_pointer_t root, size_t& steps);
84
85     stringstream m_input;
86 };
87

```

Рисунок 2.5 — Реалізація класу для декодування Гаффмана

Основна мета класу Decoder полягає у розкодуванні стислого бітового потоку, що містить закодовані символи з використанням алгоритму Гаффмана. Цей процес включає критичні етапи, такі як розбиття бітового потоку на послідовності символів, використання бінарного дерева Гаффмана для відновлення оригінальних символів та відновлення початкових даних.

Клас Decoder має методи, які забезпечують виконання цих завдань з високою точністю та ефективністю. Він включає реалізацію алгоритмів для розбиття бітового потоку на послідовності символів, декодування кожного символу з використанням бінарного дерева Гаффмана та відновлення початкових даних.

Застосування класу Decoder у контексті алгоритму Гаффмана дозволяє досягти важливих результатів у декомпресії даних. Він використовує принципи, що базуються на структурі та правилах дерева Гаффмана, для ефективного відновлення оригінальних символів та даних. Це дозволяє повернути дані у їх початковий формат з мінімальним використанням ресурсів та забезпечити коректність процесу декомпресії.

Використання класу Decoder у контексті алгоритму Гаффмана є важливим кроком у повному циклі стиснення та декомпресії даних. Він гарантує відновлення оригінальної інформації та її відтворення з високою точністю, забезпечуючи оптимальну використання ресурсів та швидкість відновлення. Клас Decoder є важливим елементом системи стиснення даних і розширює можливості алгоритму Гаффмана у сфері декомпресії.

Алгоритм Гаффмана є ефективним способом стиснення даних, який базується на статистичній аналізі частоти входять символів. В процесі кодування, символам присвоюються унікальні бітові коди, з якими вони зберігаються або передаються. При декодуванні, ці коди розкодовуються, що дозволяє відновити початкові дані. Реалізація алгоритму Гаффмана, яку представлено в даному коді, може бути використана для стиснення та декодування даних у різних областях, таких як ефективне зберігання файлів, передача даних по мережі та інші ситуації, де критично важливим фактором є ефективне використання ресурсів.

Розглядається реалізація другого алгоритму (рисунок 2.6) з використанням класів LZWBase, Encoder та Decoder. Клас LZWBase є базовим класом для Encoder та Decoder і містить спільні властивості та методи. У конструкторі LZWBase приймається вхідний вектор байтів, ініціалізуються поля `m_input`, та генерується початковий вміст `m_dict`.

```

9  class LZWBase
10 {
11 public:
12     LZWBase(const std::vector<uint8_t>& input)
13         : m_input(input), m_dict(MAX_DEF) {}
14     const std::vector<uint8_t>& GetOutput() const { return m_output; }
15
16 protected:
17     const std::vector<uint8_t>& m_input;
18     std::vector<uint8_t> m_output;
19     std::unordered_map<std::string, size_t> m_dict;
20 };
21
22 class Encoder : public LZWBase
23 {
24 public:
25     Encoder(const std::vector<uint8_t>& input);
26 };
27
28 class Decoder : public LZWBase
29 {
30 public:
31     Decoder(const std::vector<uint8_t>& input);
32 };

```

Рисунок 2.6 — Реалізація алгоритмів кодування/декодування Лемпеля-Зіва-Велча

Клас Encoder успадковується від LZWBase та має конструктор, який передає параметри в конструктор LZWBase. У методі Encoder::Encoder виконується створення вектору байтів bytes, а також рядків augmented та working. В циклі ітерується через значення байтів у вхідному векторі. Значення байта додається до рядка augmented. Якщо augmented знаходиться в словнику (m_dict), то working приймає значення augmented. У протилежному випадку, augmented додається до словника з новим індексом, а потім виконується процес кодування індексу working та додавання отриманих байтів до вихідного вектору m_output. Рядок working оновлюється з поточним байтом.

У кінці методу перевіряється, чи містить рядок working якесь значення. Якщо так, то виконується кодування індексу working та додавання байтів до вихідного вектору m_output.

Клас Decoder також успадковується від LZWBase та має конструктор, який успадковує конструктор LZWBase. Він виконує процес декодування стислого вектора байтів у вихідний вектор m_output.

Цей код реалізує алгоритм стиснення LZW (Lempel-Ziv-Welch) для компресії даних. Він використовує словник, що розширюється з кожним новим символом, що зустрічається, для ефективного кодування та декодування даних. Клас Encoder відповідає за кодування даних, тоді як клас Decoder здійснює процес декодування.

Розглянуто реалізацію інтерфейсу (рисунок 2.7), що представлена у вигляді класу AlgorithmWrapper і успадкованих від нього HuffmanAlgorithmWrapper та LZWAlgorithmWrapper. Клас AlgorithmWrapper є базовим класом, який має конструктор, приймаючий назву алгоритму. Він також містить два віртуальних методи: Compress для стиснення даних та Decompress для розпакування стислих даних. Ці методи є чисто віртуальними та мають бути перевизначені в похідних класах.

```
5 class AlgorithmWrapper
6 {
7 public:
8     AlgorithmWrapper(const std::string name)
9         : m_alg_name(name) {};
10    virtual std::vector<uint8_t> Compress(const std::vector<uint8_t>& input) const = 0;
11    virtual std::vector<uint8_t> Decompress(const std::vector<uint8_t>& input) const = 0;
12 protected:
13     const std::string m_alg_name;
14 };
15
16 class HuffmanAlgorithmWrapper : public AlgorithmWrapper
17 {
18 public:
19     HuffmanAlgorithmWrapper()
20         : AlgorithmWrapper("Huffman algorithm") {};
21    virtual std::vector<uint8_t> Compress(const std::vector<uint8_t>& input) const;
22    virtual std::vector<uint8_t> Decompress(const std::vector<uint8_t>& input) const;
23 };
24
25 class LZWAlgorithmWrapper : public AlgorithmWrapper
26 {
27 public:
28     LZWAlgorithmWrapper()
29         : AlgorithmWrapper("LZW algorithm") {};
30    virtual std::vector<uint8_t> Compress(const std::vector<uint8_t>& input) const;
31    virtual std::vector<uint8_t> Decompress(const std::vector<uint8_t>& input) const;
32 };
```

Рисунок 2.7— Реалізація інтерфейсу для алгоритмів кодування/декодування

Клас HuffmanAlgorithmWrapper успадковується від AlgorithmWrapper та має конструктор, який передає параметри в конструктор базового класу та встановлює назву алгоритму як "Huffman algorithm". Наведений клас також перевизначає методи Compress та Decompress, які реалізовані для стиснення та розпакування даних за допомогою алгоритму Гаффмана.

Клас LZWAlgorithmWrapper також успадковується від AlgorithmWrapper та має конструктор, який передає параметри в конструктор базового класу та встановлює назву алгоритму як "LZW algorithm". Він також перевизначає методи Compress та Decompress, які реалізовані для стиснення та розпакування даних за допомогою алгоритму LZW.

Цей код представляє структуру, що дозволяє згортати (стискати) та розгортати (розпаковувати) дані за допомогою різних алгоритмів стиснення. Клас AlgorithmWrapper є абстрактним базовим класом, що визначає загальну структуру алгоритмів та надає інтерфейс для стиснення та розпакування даних. Класи HuffmanAlgorithmWrapper та LZWAlgorithmWrapper успадковуються від AlgorithmWrapper і реалізують конкретні алгоритми стиснення - алгоритм Гаффмана та алгоритм LZW відповідно. Кожен з цих класів перевизначає методи Compress та Decompress для виконання відповідних дій з даними.

Розглянуто клас AlgorithmsManager (рисунок 2.8), який виконує керування алгоритмами стиснення та розпакування даних. Конструктор класу отримує набір алгоритмів, команду та вхідні дані.

```

8 class AlgorithmsManager
9 {
10     using algorithms_t = std::vector<std::shared_ptr<AlgorithmWrapper>>;
11 public:
12     AlgorithmsManager(algorithms_t& algorithms, const SCommand& command, const std::vector<uint8_t>& input)
13         : m_algorithms(algorithms), m_command(command), m_input(input) {};
14
15     std::vector<uint8_t> ExecuteCommand();
16
17 private:
18     auto Measure(const AlgorithmWrapper& alg, std::vector<uint8_t>& out) { ... }
19
20     const algorithms_t m_algorithms;
21     const SCommand& m_command;
22     const std::vector<uint8_t> m_input;
23 };

```

Рисунок 2.8 — Реалізація класу керування алгоритмами стиснення та розпакування даних

Метод `ExecuteCommand` виконує виконання команди, залежно від типу операції. У випадку операції стиснення (`EOperation::Compress`), метод створює окремий потік для кожного алгоритму з набору. Кожен потік викликає функцію `Measure` для вимірювання часу виконання стиснення за допомогою відповідного алгоритму. Функція `Measure` вимірює час початку та закінчення виконання алгоритму та повертає цей час. Після завершення виконання всіх потоків, отримані часи виконання зберігаються та використовуються для подальшого аналізу. Крім того, під час виконання кожного алгоритму, отримані стиснуті дані зберігаються у відповідному векторі.

Після завершення виконання всіх алгоритмів, виводиться результат - час виконання та розмір стислих даних для кожного алгоритму. Для цього використовується збережений час виконання та вектор стислих даних. Потім вибирається алгоритм з найменшим розміром стислих даних, який повертається як результат методу.

У випадку операції розпакування (`EOperation::Decompress`), алгоритм працює подібно до операції стиснення, виконуються вимірювання часу виконання для кожного алгоритму та розпакування даних. Отримані результати виводяться, і зберігаються з метою використання для подальшого аналізу.

Цей код забезпечує потужну і гнучку систему для керування алгоритмами стиснення та розпакування даних. Він дозволяє легко порівнювати різні алгоритми за швидкістю та ефективністю, вимірювати час виконання та отримувати розмір стислих або розпакованих даних. Код є гнучким і розширюваним, що дозволяє додавати нові алгоритми стиснення та розпакування без зміни основної логіки програми.

2.3 Висновки до розділу 2

У даному розділі описано призначення і взаємодію модулів розробленої програми, що пропонує метод об'єднання алгоритмів компресії даних з спільним інтерфейсом та виконання їх паралельно. Використання паралельного виконання алгоритмів компресії дозволяє досягти стиснення даних з мінімальним розміром вихідного файлу. Ця програма має значні технічні переваги, такі як паралельне виконання, мінімальний розмір вихідного файлу та універсальний інтерфейс. Результати дипломного проекту демонструють перспективи програми, яка об'єднує алгоритми компресії даних. Впровадження такої програми може мати значні переваги в областях, де ефективно управління даними відіграє важливу роль, зокрема у мережевих технологіях, зберіганні даних та обробці інформації.

В підрозділі 2.1 Обґрунтовано використання мови програмування C++ для реалізації програми, яка забезпечує багатопотокову компресію даних. Причини вибору мови C++ були засновані на науково-технічних факторах і перевагах, які ця мова надає у даному контексті. Однією з головних переваг мови C++ є її висока ефективність та продуктивність. Це особливо важливо для оптимізації алгоритмів компресії та забезпечення швидкого виконання завдань у багатопотоковому середовищі. Мова C++ дозволяє ефективно працювати з обчислювальними ресурсами та має низьку накладну витрату. Крім того, мова C++ надає розширені можливості для роботи з пам'яттю та маніпулювання

даними, що є необхідними для компресії даних. Завдяки цим можливостям, можна реалізувати оптимальні алгоритми компресії, які вимагають роботи з великими обсягами даних та швидкого доступу до них. Об'єктно-орієнтований підхід, який підтримує мова C++, сприяє структуруванню програмного коду та полегшує реалізацію алгоритмів компресії даних. Він покращує читабельність, модульність та повторне використання коду, що є важливими факторами при розробці складних програм. Модель пам'яті мови C++ дозволяє ефективно управляти розміром та розподілом пам'яті в процесі використання багатопоточних алгоритмів компресії даних. Це особливо корисно при обробці великих обсягів даних в багатопроцесорних системах.

В розділі 2.2 розглянуто структуру взаємодії модулів програми, а також описано роботу кожного з використаних модулів в роботі програми. Призначення кожного модуля детально аргументовано. Також надано пояснення щодо взаємодії модулів між собою. Зокрема була розглянута реалізація інтерфейсу алгоритмів компресії даних на базі якої проектуються і впроваджуються реалізації необхідних алгоритмів компресії, що застосовуватимуться в процесі роботи програми. Як приклад, розглянуто два класи, які реалізують алгоритми Гафмана та Лемпеля-Зіва-Велча відповідно, успадковуючі запропонований вище інтерфейс. Описано модуль, який дозволяє одночасно запускати декілька алгоритмів компресії даних з метою досягнення кращого результату компресії

3 АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

На основі проведених експериментів з використанням утиліти для компресії даних, яка імплементує алгоритми, базовані на алгоритмах Гаффмана, Лемпеля-Зіва-Велча і Роберта Райса, на матеріалах Canterbury Corpus, який є еталоном для оцінювання методів стиснення без втрат. Набір файлів Canterbury Corpus розроблено спеціально для тестування нових алгоритмів стиснення. Файли були відібрані на основі їх здатності надавати репрезентативні результати продуктивності. Цей набір файлів призначений для заміни Calgary Corpus, якому вже більше десяти років. Calgary Corpus вперше був представлений у книзі «Стиснення тексту» Белла, Клірі та Віттена, опублікованій у 1990 році. Відтоді багато дослідників використовували ці матеріали для порівняння результатів алгоритмів компресії. Canterbury Corpus складається з чотирьох основних підкорпусів: Artificial Corpus, Large Corpus, Miscellaneous Corpus та Canterbury. Нижче детальніше розглянуто кожен з них.

Artificial Corpus: Цей підкорпус містить штучно створені тексти, які були згенеровані згідно з певними правилами та структурою. Такі тексти дозволяють вивчати та оцінювати ефективність алгоритмів стиснення на штучно створених даних.

Large Corpus: Цей підкорпус складається з великої кількості різноманітних текстових документів, які представляють собою реальні тексти, зібрані з різних джерел. Він включає текстові файли різного розміру, що дозволяє використовувати його для оцінювання та порівняння різних методів стиснення даних.

Miscellaneous Corpus: Цей підкорпус містить тексти різних типів та жанрів, таких як новини, наукові статті, літературні твори тощо. Його метою є представлення різноманітності даних та перевірка, наскільки добре алгоритми стиснення працюють на різних типах текстових документів.

Canterbury: Canterbury є основним підкорпусом Canterbury Corpus і включає тексти, які походять з різних джерел та представляють собою різні

тексти різного розміру. Він є основною частиною набору даних, яка використовується для оцінювання та порівняння методів стиснення даних без втрат.

Calgary: Calgary є ще одним набором даних, який використовується для тестування алгоритмів стиснення даних без втрат. Цей корпус складається з різноманітних файлів різних типів даних, таких як текстові документи, програми, зображення та інші. Використання корпусу Calgary дозволяє оцінити ефективність методів стиснення на реальних даних та порівняти їх з іншими алгоритмами стиснення.

3.1 Тестування на матеріалах Canterbury Corpus

Надалі розглянемо як кожен алгоритм впорався з окремим підкорпусом.

Розглянемо результати тестування алгоритмів на матеріалах підкорпусу Artificial. З отриманих даних (рисунок 3.1) бачимо, що найкраще задачу компресії виконують алгоритм Гаффмана та Лемпеля-Зіва-Велча. Ця тенденція буде простежуватися впродовж усіх тестів. Також бачимо, що в деяких випадках, особливо коли розмір файлу дуже малий (в тестовому варіанті це 1 байт), алгоритми стиснення можуть збільшувати розмір файлу. Це аргументується тим, що в файл додаються необхідні позначки з інструкціями або інформацією, необхідні для коректної та/або ефективнішої декомпресії обробки стиснутого файлу.

Файл	Розмір вхідного файлу	Час Huffman (мс)	Розмір Huffman (байт)	Час LZW (мс)	Розмір LZW (байт)	Час RICE (мс)	Розмір RICE (байт)	Розмір на виході утиліти (байт)
artificial/a.txt	1	0.007	6	0.027	11	0.056	11	1
artificial/aaa.txt	100000	3.704	1252	12.691	536	77.194	100010	536
artificial/alpha.txt	100000	10.720	59704	14.523	3059	185.579	100010	3059
artificial/random.txt	100000	22.140	75214	12.995	104456	447.694	97657	75214

Рисунок 3.1 — Результати тестів підкорпусу Artificial

Розглянемо результати тестування алгоритмів на матеріалах підкорпусу Calgary. З отриманих даних (рисунок 3.2) бачимо, схожу картину що і в минулому тестуванні. Алгоритм Гаффмана закономірно показує кращі результати для тих файлів, в яких була використана менша кількість унікальних символів. А алгоритм Лемпеля-Зіва-Велча демонструє кращі показники у випадку, якщо файл містить багато повторювальних комбінацій символів.

Файл	Розмір вхідного файлу	Час Huffman (мс)	Розмір Huffman (байт)	Час LZW (мс)	Розмір LZW (байт)	Час RICE (мс)	Розмір RICE (байт)	Розмір на виході утиліти (байт)
calgary/bib	111261	14.680	73007	20.933	60288	264.867	106627	60288
calgary/book1	768771	104.233	438625	112.342	418900	1689.241	746509	418900
calgary/book2	610856	81.391	368598	94.918	325179	1460.897	592704	325179
calgary/geo	102400	15.311	73224	21.365	79259	270.566	103207	73224
calgary/news	377109	44.503	246704	64.916	231435	937.669	363168	231435
calgary/obj1	21504	2.712	16644	4.920	13685	40.117	21598	13685
calgary/obj2	246814	33.139	194833	46.002	134552	639.124	247942	134552
calgary/paper1	53161	5.830	33586	8.950	28962	138.503	51413	28962
calgary/paper2	82199	7.925	47853	11.564	42679	177.315	80077	42679
calgary/paper3	46526	5.769	27494	8.993	25356	101.706	45360	25356
calgary/paper4	13286	1.853	8060	2.280	7144	17.676	12918	7144
calgary/paper5	11954	1.357	7654	2.346	6807	21.778	11542	6807
calgary/paper6	38105	4.764	24265	6.737	20539	98.459	36657	20539
calgary/pic	513216	44.951	106961	59.257	65819	699.311	392400	65819
calgary/progc	39611	5.379	26156	6.190	21137	75.107	37577	21137
calgary/progl	71646	8.309	43213	9.652	30468	192.467	67692	30468
calgary/progp	49379	6.277	30456	7.660	21533	109.200	46698	21533
calgary/trans	93695	12.622	65502	16.008	46902	249.224	88410	46902

Рисунок 3.2 — Результати тестів підкорпусу Calgary

Ілюстративним прикладом переваги алгоритму Лемпеля-Зіва-Велча є файл "bib", який становить собою текст Біблії (див. рисунок 3.3). Очевидно, що вміст цього файлу містить значну кількість повторюваних лексичних конструкцій. Наприклад, поєднання "And the" зустрічається у тексті 3383 рази, а поєднання "and the" - 5964 рази. Ці часті повтори можуть бути відображені у вигляді кодів, наданих їм у словнику, що дозволяє здійснити кодування певних лексичних комбінацій за допомогою відповідних кодів.

In the beginning God created the heaven and the earth. And the earth was without form, and void; and darkness was upon the face of the deep. And the Spirit of God moved upon the face of the waters. And God said, Let there be light: and there was light. And God saw the light, that it was good: and God divided the light from the darkness. And God called the light Day, and the darkness he called Night. And the evening and the morning were the first day. And God said, Let there be a firmament in the midst of the waters, and let it divide the waters from the waters. And God made the firmament, and divided the waters which were under the firmament from the waters which were above the firmament: and it was so. And God called the firmament Heaven. And the evening and the morning were the second day. And God said, Let the waters under the heaven be gathered together unto one place, and let the dry land appear: and it was so. And God called the dry land Earth; and the gathering together of the waters called he Seas: and God saw that it was good. And God said, Let the earth bring forth grass, the herb yielding seed, and the fruit tree yielding fruit after his kind, whose seed is in itself, upon the earth: and it was so. And the earth brought forth grass, and herb yielding seed after his kind, and the tree yielding fruit, whose seed was in itself, after his kind: and God saw that it was good. And the evening and the morning were the third day. And God said, Let there be lights in the firmament of the heaven to divide the day from the night; and let them be for signs, and for seasons, and for days, and years: And let them be for lights in the firmament of the heaven to give light upon the earth: and it was so. And God made two great lights; the greater light to rule the day, and the lesser light to rule the night: he made the stars also. And God set them in the firmament of the heaven to give light upon the earth, And to rule over the day and over the night, and to divide the light from the darkness: and God saw that it was good. And the evening and the morning were the fourth day. And God said, Let the waters bring forth abundantly the moving creature that hath life, and fowl that may fly above the earth in the open firmament of heaven. And God created great whales, and every living creature that moveth, which the waters brought forth abundantly, after their kind, and every winged fowl after his kind: and God saw that it was good. And God blessed them, saying, Be fruitful, and multiply, and fill the waters in the seas, and let fowl multiply in the earth. And the evening and the morning were the fifth day. And God said, Let the earth bring forth the living creature after his kind, cattle, and creeping thing, and beast of the earth after his kind: and it was so. And God made the beast of the earth after his kind, and cattle after their kind, and every thing that creepeth upon the earth after his kind: and God saw that it was good. And God said, Let us make man in our image, after our likeness: and let them have dominion over the fish of the sea, and over the fowl of the air, and over the cattle, and over all the earth, and over every creeping thing that creepeth upon the earth. So God created man in his own image, in the image of God created he him; male and female created he them. And God blessed them, and God said unto them, Be fruitful, and multiply, and replenish the earth, and subdue it: and have dominion over the fish of the sea, and over the fowl of the air, and over every living creature that creepeth upon the earth, wherein there is life, I have given every green herb for meat: and you it shall be for meat. And God said, Behold, I have given you every herb bearing seed, which is upon the face of all the earth, and every tree, in the which is the fruit of a tree yielding seed; to you it shall be for meat. And to every beast of the earth, and to every fowl of the air, and to every thing that creepeth upon the earth, wherein there is life, I have given every green herb for meat: and it was so. And God saw every thing that he had made, and, behold, it was very good. And the evening and the morning were the sixth day. Thus the heavens and the earth were finished, and all the host of them. And on the seventh day God ended his work which he had made; and he rested on the seventh day from all his work which he had made. And God blessed the seventh day, and sanctified it: because that in it he had rested from all his work which God created and made. These are the generations of the heavens and of the earth when they were created, in the day that the LORD God made the earth and the heavens, And every plant of the field before it was in the earth, and every herb of the field before it grew: for the LORD God had not caused it to rain upon the earth, and there was not a man to till the ground. But there went up a mist from the earth, and watered the whole face of the ground. And the LORD God formed man of the dust of the ground, and breathed into his nostrils the breath of life; and man became a living soul. And the LORD God planted a garden eastward in Eden; and there he put the man whom he had formed. And out of the ground made the LORD God to grow every tree that is pleasant to the sight, and good for food; the tree of life also in the midst of the garden, and the tree of knowledge of good and evil. And a river went out of Eden to water the garden; and from thence it was parted, and became into four heads. The name of the first is Pison: that is it which compasseth the whole land of Havilah, where there is gold; And the gold of that land is good: there is bdellium and the onyx stone.

Рисунок 3.3 — Вміст файлу "bib"

Розглянемо результати тестування алгоритмів на матеріалах підкорпусу Canterbury. З отриманих даних (рисунок 3.4) бачимо, схожу картину що і в минулому тестуванні. В цьому підкорпусі алгоритми показують приблизно однакові результати.

Файл	Розмір вхідного файлу	Час Huffman (мс)	Розмір Huffman (байт)	Час LZW (мс)	Розмір LZW (байт)	Час RICE (мс)	Розмір RICE (байт)	Розмір на виході утиліті (байт)
canterbury/alice29.txt	152089	21.225	87896	26.185	76910	310.104	146686	76910
canterbury/asyoulik.txt	125179	18.639	76009	21.349	67318	323.438	121298	67318
canterbury/cp.html	24603	3.961	16423	4.409	12766	59.075	23580	12766
canterbury/fields.c	11150	0.895	7247	2.300	4970	29.232	10518	4970
canterbury/grammar.lsp	3721	0.404	2347	0.925	1819	3.552	3519	1819
canterbury/kennedy.xls	1029744	123.278	463352	131.563	269875	2287.038	773249	269875
canterbury/lcet10.txt	426754	61.649	250820	63.273	218335	901.344	413883	218335
canterbury/plrabcn12.txt	481861	63.581	275827	69.574	253751	1003.362	466888	253751
canterbury/ptt5	513216	42.664	106961	60.678	65819	719.880	392400	65819
canterbury/sum	38240	5.299	26257	7.411	20032	99.101	35536	20032
canterbury/xargs.1	4227	0.519	2774	1.179	2345	5.895	4108	2345

Рисунок 3.4 — Результати тестів підкорпусу Canterbury

Розглянемо результати тестування алгоритмів на матеріалах підкорпусів Large&Misc. З отриманих даних (рисунок 3.5) бачимо, схожу картину що і в минулому тестуванні.

Файл	Розмір вхідного файлу	Час Huffman (мс)	Розмір Huffman (байт)	Час LZW (мс)	Розмір LZW (байт)	Час RICE (мс)	Розмір RICE (байт)	Розмір на виході утиліти (байт)
large/bible.txt	4047392	540.915	2218687	545.102	1832483	8546.534	3932576	1832483
large/E.coli	4638690	375.115	1159700	573.355	1350030	7099.368	4638700	1159700
large/world192.txt	2473400	338.800	1558910	387.502	1414925	6247.919	2367806	1414925
misc/pi.txt	1000000	140.432	424942	147.576	504100	2505.126	875010	424942

[illegible]

				ІАЛЦ.045200.004 ПЗ	Ли
Лист	№ докум.	Підп.	Дата		4

У тестах використовувалися різні вхідні файли з різними розмірами. надано наступні висновки.

Алгоритм Гаффмана, що застосовує утиліта, продемонстрував доцільність використання методів статистичного кодування у зменшенні розміру даних використовуючи оптимальні префіксні коди. Виявлено, що алгоритм Гаффмана відносно добре справляється з компресією файлів з Canterbury Corpus, забезпечуючи значне скорочення обсягу даних.

Алгоритм Лемпеля-Зіва-Велча, використаний в утиліті, демонструє доцільність використання методів компресії з виявленням повторюваних послідовностей та створенням словника, що допомагає у зменшенні розміру даних. Цей алгоритм забезпечує високу ступінь стиснення даних, зокрема на матеріалах Canterbury Corpus, де часто зустрічаються повтори та шаблони.

Крім того, реалізація алгоритму Роберта Райса, яка базується на алгоритмі Голомба в утиліті показала ефективність на дуже специфічних даних, кодуванні цілочисельних значень, які зазвичай застосовуються для монохромних зображень та створенні відносно компактного представлення даних. Застосування алгоритму Голомба дозволяє знизити розмір файлів з Canterbury Corpus шляхом кодування цілочисельних даних та використання нерівномірних кодів. Однак воно є дуже ресурсозатратним, бо підбір необхідних параметрів потребує багаторазового проходження по вхідним даним.

Таким чином, аналізуючи проведені тести, можемо підтвердити високу ефективність одночасного використання реалізацій алгоритмів Гаффмана, Лемпеля-Зіва-Велча та Голомба в утиліті для компресії даних на матеріалах Canterbury Corpus. Впровадження цих алгоритмів забезпечує значне скорочення розміру файлів та збереження даних з оптимальним ступенем компресії.

3.2 Аналіз отриманих результатів

Проаналізувавши отримані результати роботи утиліти і досвід від взаємодії з нею був створений план з подальшого вдосконалення розробленої утиліти для компресії даних. Запропоновано наступні пропозиції:

1. здійснити дослідження і розробку нових алгоритмів компресії, використовуючи передові методи, такі як адаптивне кодування або контекстне моделювання. Це може сприяти подальшому зменшенню розміру файлів та покращенню ефективності компресії;
2. розширити функціональність програми шляхом додавання підтримки для інших алгоритмів компресії, таких як алгоритми на основі вейвлет-перетворення або алгоритми, що використовують машинне навчання. Це може дозволити використовувати програму для різноманітних типів даних і отримати універсальніший інструмент для компресії;
3. провести оптимізацію алгоритмів та реалізації програми з метою зменшення часу компресії та декомпресії. Це може включати оптимізацію алгоритмічних операцій, використання гешування або оптимізацію паралельної обробки;
4. покращити управління пам'яттю та ресурсами програми, зокрема шляхом використання ефективних алгоритмів стиснення та зменшення використання оперативної пам'яті;
5. розробити зручний та інтуїтивний GUI, що спрощує використання програми та надає можливості налаштування параметрів компресії, а також моніторингу процесу компресії та візуалізації результатів;
6. провести детальний аналіз та оцінку якості компресії для різних типів даних та наборів даних, щоб встановити оптимальні параметри компресії та забезпечити максимальну ефективність;
7. використовувати автоматичне визначення типу даних та вибір оптимального алгоритму компресії на основі аналізу властивостей вхідних даних;

8. провести порівняльні тести та аналіз результатів компресії з іншими популярними програмами для компресії даних, щоб визначити конкурентні переваги та можливості подальшого вдосконалення програми;
9. врахувати принципи забезпечення надійності та стійкості програми, зокрема шляхом виявлення та виправлення помилок, обробки виняткових ситуацій та забезпечення захисту від вторгнень;
10. провести додаткові експерименти та дослідження з метою встановлення можливостей для подальшого розширення функцій програми та оптимізації процесу компресії для специфічних вимог та потреб користувачів.

3.3 Висновки до розділу 3

На основі проведених експериментів з використанням утиліти для компресії даних, яка імплементує алгоритми, базовані на алгоритмах Гаффмана, Лемпеля-Зіва-Велча і Роберта Райса, на матеріалах Canterbury Corpus, який є еталоном для оцінювання методів стиснення без втрат, були отримані наступні висновки.

Алгоритм Гаффмана, що застосовує утиліта, продемонстрував роботу зі зменшення розміру даних за рахунок статистичного кодування, використовуючи оптимальні префіксні коди. Виявлено, що алгоритм Гаффмана впорався з компресією більшості файлів з Canterbury Corpus, забезпечуючи значне скорочення обсягу даних.

Алгоритм Лемпеля-Зіва-Велча, використаний в утиліті, демонструє роботу з виявлення повторюваних послідовностей та створенні словника, що допомагає у зменшенні розміру даних. Цей алгоритм найкраще продемонстрував себе, зокрема на матеріалах Canterbury Corpus, де часто зустрічаються повтори лексем.

Крім того, реалізація алгоритму Роберта Райса, яка базується на алгоритмі Голомба в утиліті, показала ефективність на дуже специфічних даних, кодуванні цілочисельних значень, які зазвичай застосовуються для монохромних зображень та створенні відносно компактного представлення даних. Застосування алгоритму Голомба дозволяє знизити розмір файлів з Canterbury Corpus шляхом ефективного кодування цілочисельних даних та використання нерівномірних кодів. Однак воно є дуже ресурсозатратним, бо підбір необхідних параметрів потребує багаторазового проходження по вхідним даним.

Результати виконання алгоритмів були аргументовані, а процес їх виконання пояснений на прикладних матеріалах. Це дало змогу наглядно на прикладі зрозуміти на чому ґрунтується робота наведених алгоритмів і вміти в подальшому приблизно оцінювати ефективність неведених алгоритмів для різних вхідних даних.

З метою подальшого вдосконалення програми для компресії даних, проаналізовані отримані результати роботи алгоритмів компресії і досвід взаємодії з утилітою. В результаті цього розроблена стратегія для поліпшення роботи і зручної взаємодії з утилітою.

ВИСНОВКИ

У бакалаврському дипломному проєкті досліджені принципи роботи алгоритмів стиснення даних, оцінено їх ефективність, розглянуто основні поняття теорії інформації та порівняно основні методи компресії. Також розглянуті технології, які використовувалися при розробці програмного забезпечення.

Проаналізовано актуальність проблеми та вказано на необхідність створення або вдосконалення алгоритмів стиснення в зв'язку зі зростанням обсягу інформації. Проведено дослідження наявних базових алгоритмів, які можуть виконувати поставлену задачу, але мають обмеження щодо використання пам'яті або часу відтворення даних.

Розроблено системну утиліту для компресії даних, яка об'єднує в собі декілька алгоритмів компресії даних з метою зменшення розміру результуючого файлу, збору даних та аналізу параметрів роботи обраних алгоритмів.

У результаті розробки системної утиліти для компресії даних проведені дослідження щодо ефективності різних алгоритмів стиснення та їх часу виконання та розміру стиснених даних. Обґрунтовано отримані дані і проведений їх аналіз з метою вдосконалення існуючого рішення.

Утиліта надає зручний і гнучкий програмний інтерфейс для масштабування, вибору та виконання різних алгоритмів стиснення та розпакування даних. Вона спрощує процес порівняння різних алгоритмів та надає користувачам можливість вибрати найкращий алгоритм для їх конкретних потреб.

Показано, що утиліта дозволяє отримати найкращий результат для заданого набору вхідних умов, використовуючи зазначені вище алгоритми компресії.

Розроблена утиліта може бути застосована в різних сферах життя, де важливо забезпечити стиснення та розпакування даних. Вона може бути використана в комп'ютерних системах для зменшення обсягу даних та

покращення швидкодії передачі і зберігання інформації. Додатково, утиліта може бути корисною для обробки та передачі даних у веб-серверах, мережевих протоколах, базах даних та багатьох інших ситуаціях, де стиснення даних є важливою задачею.

Здійснені дослідження дозволили порівняти різні алгоритми стиснення, зокрема алгоритм Гаффмана та алгоритм LZW, та оцінити їх ефективність. Виміряно час виконання кожного алгоритму та отримано розмір стислих даних. Аналіз отриманих результатів показав, що алгоритм Гаффмана демонструє значне зменшення розміру файлу та високу швидкодію, але його ефективність залежить від характеристик вхідних даних. Алгоритм LZW виявився не менш ефективним в термінах стиснення, і його швидкодія також була прийнятною.

Методи покращення утиліти можуть включати розширення набору алгоритмів стиснення, вдосконалення алгоритмів, оптимізацію процесу вимірювання часу виконання та розміру даних, а також впровадження додаткових функцій для аналізу результатів та визначення найкращого алгоритму для конкретних завдань.

Отже, поставлену задачу успішно вирішено. Розроблена утиліта дозволяє досягти вищого ступеня збереження пам'яті та компресії даних, а використаний інструментарій, за рахунок крос-платформеності і розширюваності програмного інтерфейсу, дозволяє використовувати утиліту в різних операційних системах, програмних засобах та сферах діяльності людини. Крім того, є потенціал для подальшого вдосконалення.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

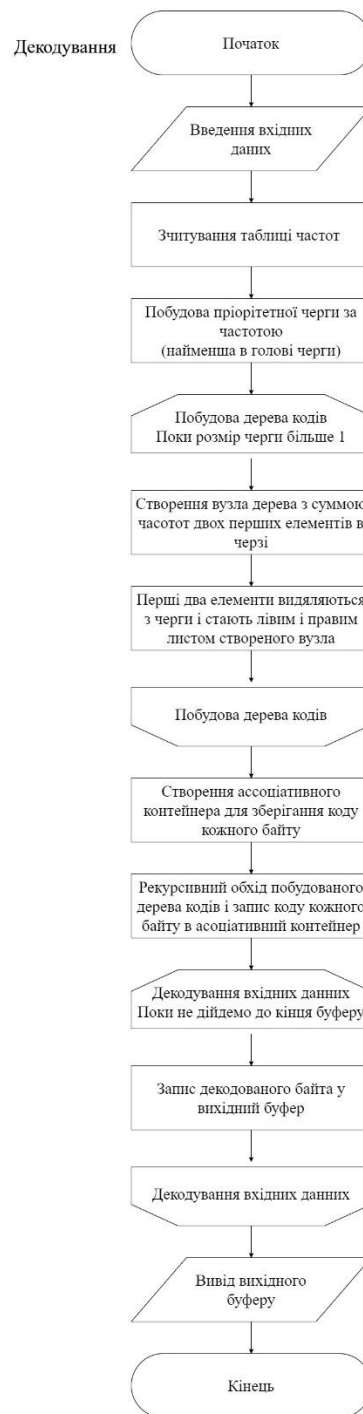
1. Information, Entropy, and Coding. Chapter 8. URL: https://www.princeton.edu/~cuff/ele201/kulkarni_text/information.pdf (дата звернення: 16.02.2023)
2. Khalid Sayood. Introduction to Data Compression. THIRD EDITION, 2006.
3. Guy E. Blelloch. Introduction to Data Compression, 2013.
4. GNU Gzip. URL: <https://www.gnu.org/software/gzip/> (дата звернення: 27.03.2023)
5. David Salomon. Data Compression: The complete Reference, 2006.
6. Peter Wayner. Data Compression for Real Programmers.
7. Khalid Sayood. Mathematical Preliminaries for Lossless Compression. URL: <https://www.sciencedirect.com/topics/computer-science/text-compression> (дата звернення: 18.04.2023)
8. Colt McAnlis & Aleks Haecky. Understanding Compression. DATA COMPRESSION FOR MODERN DEVELOPERS.
9. Bjarne Stroustrup. The C++ Programming Language. Fourth Edition.

Додатки до дипломного проєкту

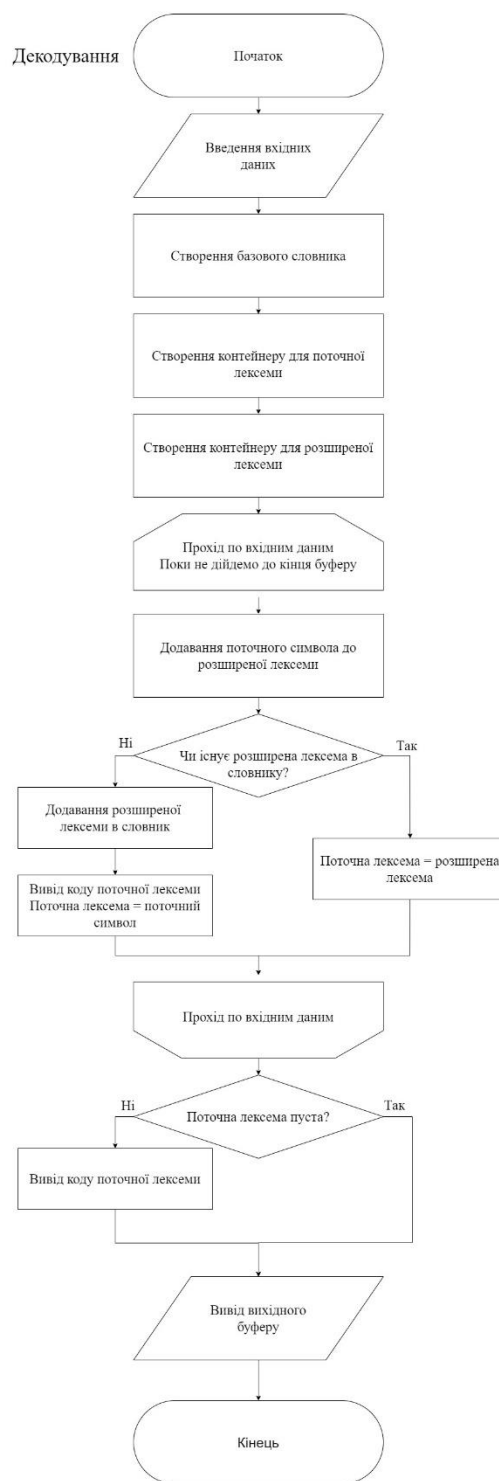
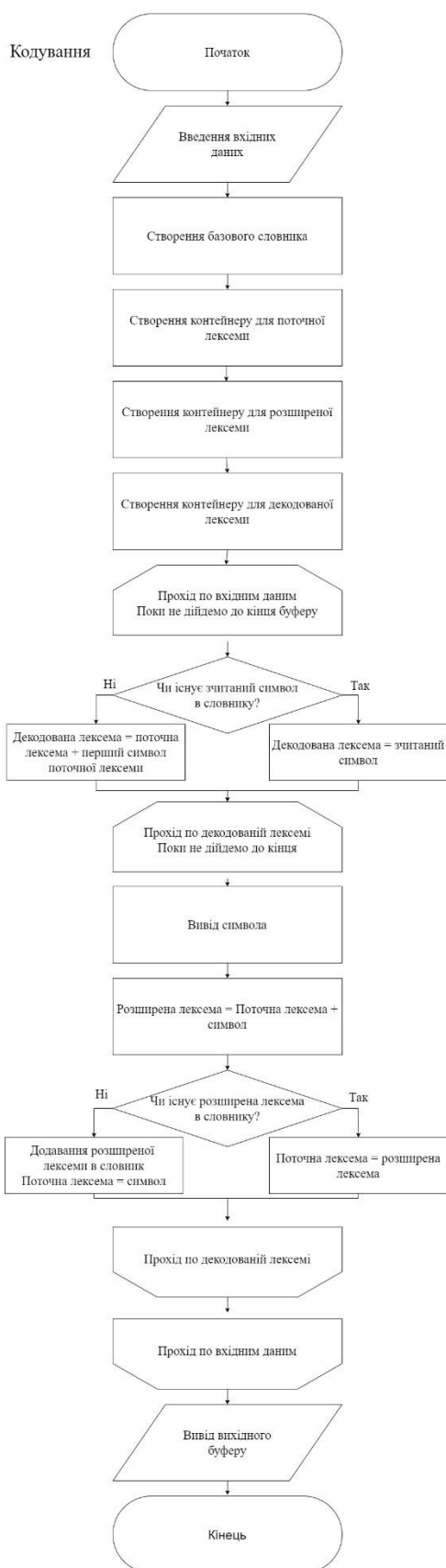
На тему: системна утиліта для компресії даних



					ІАЛЦ.045200.005 Д1										
Зм.	Лист	№ докум.	Підп.	Дата	Системна утиліта компресії даних. Схема алгоритму					Літ.		Аркуш		Аркушів	
Розробив	Оридорога М.В.												1		1
Перев.	Молчанов О.А.														
Н. контр.	Клятченко Я.М.														
Затвер.	Романкевич О.М.									КПІ ім. Ігоря Сікорського, ФПМ КВ-92					



ІАЛЦ.045200.006 Д2					Літ.			Аркуш	Аркушів
Зм.	Лист	№ докум.	Підп.	Дата	Метод Компресії/Декомпресії Гаффмана Схема алгоритму			1	1
Розробив	Оридорога М.В.								
Перев.	Молчанов О.А.								
Н. контр.	Клятченко Я.М.							КПІ ім. Ігоря Сікорського, ФПМ КВ-92	
Затвер.	Романкевич О.М.								

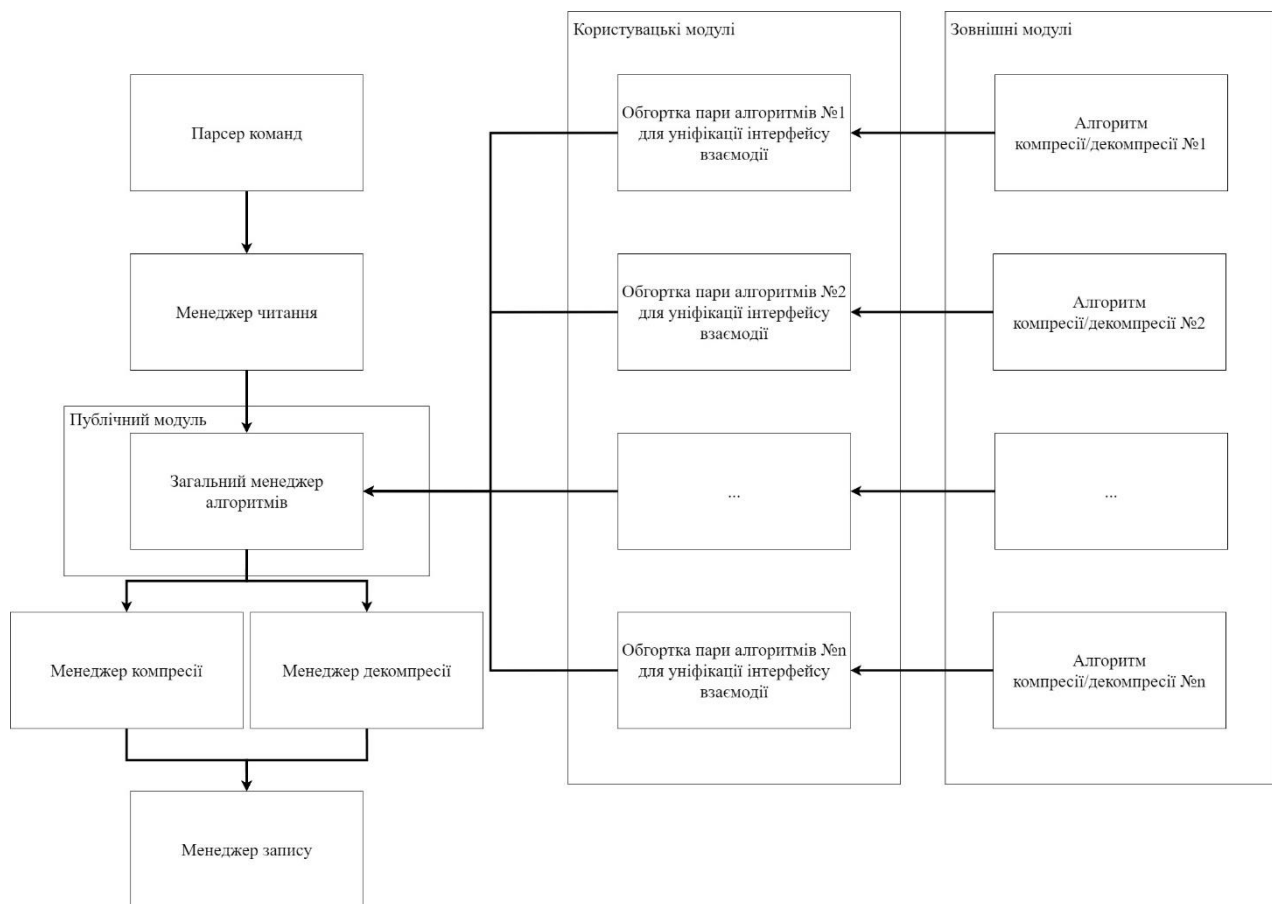


Зм.	Лист	№ докум.	Підп.	Дата
Розробив	Оридорога М.В.			
Перев.	Молчанов О.А.			
Н. контр.	Клятенко Я.М.			
Затвер.	Романкевич О.М.			

ІАЛЦ.045200.007 ДЗ

Метод Компресії/Декомпресії
Лемпеля–Зіва–Велча.
Схема алгоритму

Літ.	Аркуш	Аркушів
	1	1
КПІ ім. Ігоря Сікорського, ФПМ КВ-92		



					ІАЛЦ.045200.008 Д4						
Зм.	Лист	№ докум.	Підп.	Дата	Системна утиліта для компресії даних. Схема структурна				Літ.	Аркуш	Аркушів
Розробив	Оридорога М.В.										
Перев.	Молчанов О.А.									1	1
									КПІ ім. Ігоря Сікорського, ФПМ КВ-92		
Н. контр.	Клятченко Я.М.										
Затвер.	Романкевич О.М.										