**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ   
ІМЕНІ ТАРАСА ШЕВЧЕНКА**

Факультет комп’ютерних наук та кібернетики

Кафедра математичної інформатики

«До захисту допущено»

Завідувач кафедри

Терещенко В.М

(підпис)

«\_\_\_\_» \_\_\_\_\_\_\_\_\_ 20\_\_р.

**Дипломна робота**

**На здобуття ступеня бакалавра**

за спеціальністю 122 Комп’ютерні науки

на тему:

**ПЕРЕТВОРЕННЯ БАРРОУЗА-УІЛЛЕРА ТА ЙОГО ЗАСТОСУВАННЯ**

Виконав студент 4-го курсу  
Волохович Ігор Ігорович

(підпис)

Науковий керівник:

професор, доктор фіз.-мат. наук

Анісімов Анатолій Васильович

(підпис)

Засвідчую, що в цій курсовій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент

(підпис)

Київ – 2021

**Реферат**

Обсяг роботи ХХ сторінки, ХХ ілюстрацій, ХХ таблиць, ХХ джерел посилань.

ПЕРЕТВОРЕННЯ БАРРОУЗА-УІЛЛЕРА, МЕТОДИ СТИСНЕННЯ ДАНИХ, АЛГОРИТМ ЛЕМПЕЛЯ–ЗІВА–ВЕЛЧА, АЛГОРИТМ ХАФФМАНА

Об’єктом роботи є дослідження методів стиснення даних, перетворення Барроуза – Віллера[[9]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1) та доцільність його використання у методах стиснення.

Метою роботи є розробка алгоритмів стиснення даних та перетворення Барроуза–Віллера[[9]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1). Модифікація одного з алгоритму та порівняння якість та час стиснення даних.

Методи розроблення: аналіз алгоритмів стискання, кодування методів стиснення, аналіз якості та часу стиснення алгоритмів та їх модифікацій, розробка загальної бібліотеки алгоритмів, розробка прикладів використання алгоритмів, розробка веб–застосунку для демонстрації можливостей бібліотеки.

Інструменти розроблення. Для розробки реалізації було використано інтерактивне середовище розробки JebBrains Rider[[1]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1). Було використано мову C#[[2]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1). Створено .NET Library[[3]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1) та використовуючи ASP.NET Framework [[4]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1) створено веб – застосунок. Розроблено систему логування часу та обрахування відсотку стиснення.

Результат роботи. Було досліджено методи стиснення даних та вже існуючі продукти стиснення на ринку. Розроблено декілька демонстраційних консольних програм та веб-застосунок. Також було запроваджено стандартну загальну бібліотеку методів, яка було викладена в мережу на правах open-source та доступна всім розробникам. Було показано використання бібліотеки на різних платформах.

**Зміст**

[ВСТУП 5](#_Toc71324408)

[РОЗДІЛ 1. ЗАГАЛЬНІ ВІДОМОСТІ ПРО СТИСНЕННЯ ДАНИХ 7](#_Toc71324409)

[1.1 Що таке стиснення даних? 7](#_Toc71324410)

[1.2 Чому стиснення даних так важливо? 7](#_Toc71324411)

[1.3 Історія створення методів стиснення 8](#_Toc71324412)

[1.4 Зростання дефляції 10](#_Toc71324413)

[1.5 Типи стиснення 12](#_Toc71324414)

[1.6 Переваги та недоліки стискання даних 13](#_Toc71324415)

[1.7 Поточні програми архівації 14](#_Toc71324416)

[1.8 Майбутні розробки 15](#_Toc71324417)

[РОЗДІЛ 2. ОГЛЯД МЕТОДІВ СТИСНЕННЯ 17](#_Toc71324418)

[2.1 Вступ 17](#_Toc71324419)

[2.2 Кодування довжин серій (RLE) 17](#_Toc71324420)

[2.3 Перетворення Барроуза – Віллера (BWT) 18](#_Toc71324421)

[2.4 Ентропійне кодування (Entropy coding) 20](#_Toc71324422)

[2.5 Алгоритм Шеннона – Фано (Shannon – Fano coding) 21](#_Toc71324423)

[2.6 Алгоритм Хаффмана (Huffman coding) 22](#_Toc71324424)

[2.7 Арифметичне кодування (Arithmetic coding) 23](#_Toc71324425)

[РОЗДІЛ 3. ОГЛЯД АЛГОРИТМІВ СТИСНЕННЯ 25](#_Toc71324426)

[3.1 Алгоритми «Ковзаючого вікна» (Sliding window) 25](#_Toc71324427)

[3.1.1 Алгоритм Лемпеля – Зіва та його модифікацій 25](#_Toc71324428)

[3.1.2 DEFLATE та DEFLATE64 25](#_Toc71324429)

[3.2 Алгоритми з використанням словника 25](#_Toc71324430)

[3.3 Алгоритми без використання словників 25](#_Toc71324431)

[3.3.1 Передбачення щодо часткового збігу (PPM) 25](#_Toc71324432)

[3.3.2 Bzip2 25](#_Toc71324433)

[3.3.3 PAQ 25](#_Toc71324434)

[РОЗДІЛ 4. ІМПЛЕМЕНТАЦІЯ МЕТОДІВ І АЛГОРИТМІВ СТИСНЕННЯ 25](#_Toc71324435)

[4.1 Перетворення Барроуза – Віллера 25](#_Toc71324436)

[4.2 Стандартний метод LZW 25](#_Toc71324437)

[4.3 Метод LZW з перетворенням Барроуза-Віллера 25](#_Toc71324438)

[4.4 Метод Хаффмана (Huffman coding) 25](#_Toc71324439)

[РОЗДІЛ 5. ПОРІВНЯННЯ РЕЗУЛЬТАТІВ СТИСКАННЯ 25](#_Toc71324440)

[5.1 Порівняння розроблених методів між собою 25](#_Toc71324441)

[5.2 Порівняння розроблених методів з алгоритмами та програмами, представленими на ринку 25](#_Toc71324442)

[ВИСНОВКИ 25](#_Toc71324443)

[ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 26](#_Toc71324444)

[ДОДАТКИ 28](#_Toc71324445)

# ВСТУП

**Оцінка сучасного стану об’єкта дослідження.** На сьогоднішній день, зі зростанням кількості користувачів та кількості даних в мережі інтернет гостро постає питання зберігання даних. Так як будь-яка людина розуміє, що просто зберігати файли не дуже правильно, люди замислюються щодо використання алгоритмів стиснення задля збереження дорогого та цінного місця на серверах, комп’ютерах, мережевих системах. Через це були придумані та розроблені різні алгоритми стиснення даних та допоміжні перетворення для збільшення якості стиснення. Наприклад великій продуктовій компанії Amazon не вигідно зберігати сотні мільйонів даних про користувачів, товари та історію у первісному вигляді. Це займає багато місця. Тому вони використовують алгоритми, які допомагають в такому же об’ємі пам’яті зберігати набагато більше файлів.

**Актуальність роботи та підстави для її виконання.** Застосування, розробка нових та модифікація існуючих алгоритмів стиснення має важливе значення зі збільшенням генерації інформації у сьогоднішній час. Задля раціонального використання пам’яті та попередження надлишкової інформаційної катастрофи, коли не вистачатиме пам’яті. Методи стиснення та модифіковані (наприклад з використанням алгоритму Барроуза–Віллера) допомагають зменшити розмір файлу тим самим зменшуючи витрати та попередження настання інформаційної катастрофи.

**Мета й завдання роботи.** Метою дипломної роботи є огляд алгоритму перетворення Барроуза–Віллера, модифікація алгоритму Лемпеля – Зіва – Велша використовуючи цей алгоритм та порівняльний аналіз результатів стиснення, які приведуть до висновку модифікування алгоритмів у такий спосіб. Також завданням є реалізація власне перетворення BWT(Барроуза – Віллера)[[9]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1), алгоритму стиснення LZW(Лемпеля – Зіва – Велша)[[12]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1), модифікація даного алгоритму використовуючи перетворення та реалізація алгоритму Huffman coding(алгоритм стиснення Хаффмана)[[8]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1). Порівняння даних результатів у розрізі часу виконання та результуючому розмірі файлу. Для досягнення цієї поставлено такі завдання:

* Огляд методу BWT[[9]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1).
* Огляд основних методів та алгоритмів стиснення.
* Провести роботу у дослідженні розробки бібліотек
* Розробити програмну реалізацію алгоритму BWT[[9]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1), LZW[[12]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1), LZW + BWT, Huffman coding[[8]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1) з можливістю використовувати у будь-якому проекті на базі мови C#[[2]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1).

**Об'єкт, методи й засоби розроблення**. Об’єктом порівняльного аналізу є стандартні та модифіковані алгоритми стиснення даних. Об’єктом розробки є декілька програмних реалізацій, які допомагають побачити та застосувати бажаний алгоритм стиснення та бібліотеку з алгоритмами, яка може бути використана та встановлена будь-яким розробником через NuGet[[4]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1) пакет у середовищі мови програмування C#[[2]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1)

**Можливі сфери застосування.** Дана реалізація доступна у вигляді розширюваної бібліотеки для вбудовування у будь – які додатки на розсуд розробника. Розроблені алгоритми допомагають зменшити розмір файлів не пошкоджуючи якість даного файлу.

# РОЗДІЛ 1. ЗАГАЛЬНІ ВІДОМОСТІ ПРО СТИСНЕННЯ ДАНИХ

## Що таке стиснення даних?

Стиснення даних – це зменшення кількості бітів, необхідних для репродукції даних. Такі алгоритми допомагають зберігати пам’ять, збільшити швидкість передачі даних та зменшити вартість накопичувачів та необхідну пропускну властивість інтернету для передачі. Іншими словами, це процес кодування, реструктуризації або модифікування даних для зменшення її розміру. Фундаментально, це включає повторне шифрування інформації, користуючись меншими бітами (шматками), ніж оригінальне представлення. Стиснення даних підлягає компромісу між кількістю пам’яті, яку займає та часом. Це широко застосовується в обчислювальних сервісах та рішеннях, особливо в комунікації даними, Стискання даних працює за допомогою декількох методів стиснення та програмних рішень, які використовують алгоритми стискання для зменшення обсягу даних. [[20]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1)

## Чому стиснення даних так важливо?

Стиснення даних може значно зменшити обсяг пам'яті, займаний файлом. Наприклад, при співвідношенні стиснення 2:1, файл розміром 20 мегабайт (МБ) займає 10 МБ місця. В результаті стиснення, компанії витрачають менше грошей і менше часу на зберігання. Стиснення оптимізує продуктивність сховища резервних копій і нещодавно проявилося в скороченні обсягу даних первинного сховища. Стиснення буде важливим методом скорочення даних, оскільки кількість даних продовжує зростати експоненційно.

Практично будь-який тип файлів може бути стиснутий, але важливо дотримуватися рекомендацій при виборі того, які з них стискати. Наприклад, деякі файли вже можуть бути стиснуті, тому стиснення цих файлів не матиме істотного впливу або може випливти в більший розмір файлу.

## Історія створення методів стиснення

Теоретичну базу стискання даних було розроблено Клодом Шеноном, який опублікував роботи по алгоритмічній теорії інформації(AIT) [[13]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1) для компресії без втрат та теорії швидкості спотворення[[14]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1) для компресії з втратами. Вони стали фундаментом започаткування стискання даних, які на сьогоднішній день досягли величезних здібностей та дуже часто застосовуються у машинному навчанні.

З народженням інтернету в 70 – х роках, взаємозв’язок між розміром файлів та швидкістю передачі даних став більш значущим. Видатні розуми всього світу боролися з цією проблемою роками, але безрезультатно допоки не з’явився у середині 80 – х роках універсальний алгоритм стиснення без втрат Лемпеля – Зіва – Велша (LZW)[[12]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1), який показав реальні переваги стискання. [[21]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1)

Метод LZW був першим широко вживаним алгоритмом стискання даних, який було розроблено на комп’ютерах та досі використовується сьогодні (у різноманітних варіаціях). Великий текст англійською мовою може бути стиснутий приблизно на 50% використовуючи LZW[[6]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1). Але історія методів починає задовго до цього. Азбука Морзе, яка була створена у 1838 році, являється предком такого поняття як «стискання даних». Тобто, це є одним із найперших методів стискання. Таким алгоритмом, в якому найпоширеніші літери англійської мові, такі як «е» і «t» мають коротші азбуки Морзе.

Пізніше, в 1949 році, коли почали освоюватися мейнфрейми, вже відомий Клод Шеннон разом з Робертом Фано розробили кодування Шеннона – Фано[[11]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1). Їх алгоритм призначав коди символам у заданих блоках базуючись на ймовірності появи символу. Ймовірність появи символу обернено пропорційна довжині коду, що призводить до більш короткого способу презентації даних[[5]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1).

Два роки потому, Девід Хаффман вивчав курс «Теорія Інформації» в Массачусетському Технологічному Інституті (МІТ) та мав заняття, викладачем яких був Робертом Фано. Викладач запропонував класу вибір: написати курсову робочу або складати випускний іспит. Авжеж Хаффман виправ курсову, тема якої була «Пошук найбільш ефективного методу двійкового кодування». Попрацювавши кілька місяців та нічого не придумавши, Хаффман вже збирався викинути всю свою роботу й почати готуватися до випускного іспиту замість здачі курсової. Саме в цей час йому в голову прийшла ідея, просвітництво та він придумав дещо схожу на техніку Шеннона – Фано[[11]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1), але набагато ефективнішу. Ключова відмінність між кодуванням Хаффмана та зазначеним вище було те, що у алгоритмі Шеннона – Фано[[11]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1) дерево ймовірностей будувалося знизу вверх, створюючи неоптимальний результат, а в цьому алгоритмі – зверху вниз. [[15]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1)

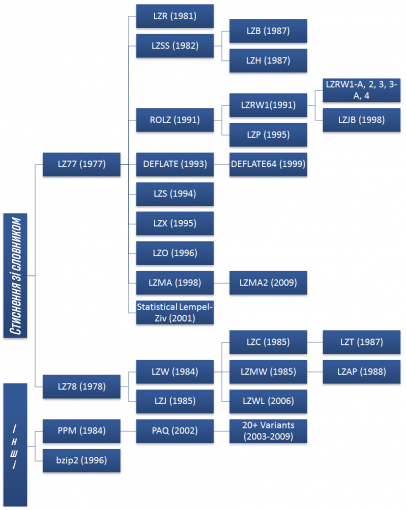


Рисунок 1. Ієрархія алгоритмів стиснення «без втрат»

Ранні імплементації алгоритмів Хаффмана[[8]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1) та Шеннона – Фано[[11]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1) були розроблені з використанням апаратних та жорстко закодованих (hard coded) кодів. Але коди Хаффмана до цього були статичними. Завдяки появі Інтернету в 1970 – х роках та сервісів, які надаюсь послуги онлайн – сховищ, розробники змогли вдосконалити алгоритм та реалізувати такий застосунок, який генерував коди Хаффмана динамічно застосовуючи вхідні дані як основу. Пізніше, в 1977 році, Авраам Лемпель та Якоб Зів опублікували свою інноваційний, новаторський алгоритм LZ77[[6]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1). Це був перший алгоритм, який для стиснення використовує словник. Конкретніше, LZ77[[6]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1) часто використовував динамічний словник, який називається «ковзаючим вікном» (sliding window)[[10]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1). У 1978 році цей же дует опублікував новий алгоритм LZ78[[7]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1), який теж використовує словник. На відміну від LZ77[[6]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1), цей алгоритм аналізує вхідні дані та генерує статичний словник, а не генерує його динамічно. [[16]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1)

## Зростання дефляції

З моменту виходу алгоритмів дуету Лемпеля – Зіва, компанії та інші великі організації використовують алгоритми стиснення даних, оскільки їхні потреби в зберіганні та сховищі продовжують зростати і стиснення даних може повністю задовольнити їх потреби. Однак алгоритми стиснення даних не набули такої широкої популярності, допоки всесвітня мережа Інтернет не набула ваги та значущості в кінці 1980 – х років. Саме в ці роки виникла необхідність в стисненні даних. В ті часи мережева пропускна здатність була коштовною, обмеженою або і тим й іншим одночасно. Саме стиснення допомогло вирішити ці проблеми та закрити недоліки. З появою Всесвітньої павутини, коли люди почали між собою обмінюватися різноманітною інформацією, від текстів до відео, стискаючі алгоритми стали особливо важливими. Адже це допомагало зберігати дорогоцінний трафік та (зважаючи на маленькі швидкості) швидко завантажувати необхідні дані. Постало питання задоволення потреб в стисненні різних даних. Саме для цього було розроблено кілька нових форматів файлів, що включають стиснення, включаючи ZIP, GIF, PNG.

Том Хендерсон опублікував та випустив перший комерційно успішний формат архівування під назвою ARC через свою компанію в 1985 році, System Enhancement Associates. ARC був особливо популярним серед товариства BBS, так як це була одна з перших програм, яка могла не тільки будувати, але й стискати файли. Також вона мала відкритий вихідний код. Формат ARC використовував модифікований алгоритм LZW. В цей час Філ Кац помітив шалену популярність формату ARC і вирішив поліпшити його, написавши процедури компресії та декомпресії мовою асемблера. Він опублікував PKARC та почав його продавати. Через копірайт, який було доведено, Кац був вимушений відректися від свого дітища через судові процеси. [[17]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1)

Так як він не міг більше продавати PKARC, в 1989 році він розробив модифіковану версію алгоритму, який зараз відомий як ZIP формат. Через використання запатентованого алгоритму LZW під капотом, було вирішено перейти на новий IMPLODE алгоритм. Формат було знову оновлено в 1993 році, коли Кац випустив PKZIP 2.0, який реалізовував DEFLATE алгоритм і деякі інші функції, наприклад поділ об’ємів. Ця версія формату ZIP і сьогодні повсюдно зустрічається, так як майже всі файли .zip слідують формату PKZIP 2.0, незважаючи на поважний вік.

GIF або Graphical Interchange Format, було розроблено компанією CompuServe аж в 1987 році щоб дозволити спільне використання растрових зображень без втрати даних (хоча формат обмежений 256 кольорами на кадр) при цьому істотно зменшивши розмір файлу, щоб забезпечити передачу по модемах віддаленого доступу (dialup modems). Однак, як і формат ZIP, GIF також базується на запатентованому алгоритмі LZW. Незважаючи на обтяженість патентами, компанія Unisys не змогла захистити та жорстко контролювати дотримання своїх патентів, щоб зупинити поширення формату. Навіть зараз, більше 20 років потому, GIF залишається у використанні, особливо через його здатність анімації. [[22]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1)

Хоча GIF не вдалося зупинити, CompuServe шукала формат, який не є обтяженим патентами. І в 1994 році представила формат Portable Network Graphics (PNG). Як і ZIP, стандарт PNG використовує алгоритм стискання DEFLATE. Хоча DEFLATE був запатентований Філом Кацом, патент ніколи не застосовувався і, таким чином, PNG та інші формати, засновані на алгоритмі DEFLATE, уникають порушення патентів. Незважаючи на те, що LZW сильно піднявся у рейтингу в перші дні започаткування стискання даних, через суперечливий характер Unisys він більш – менш помер на користь більш швидшого та ефективнішого алгоритму DEFLATE. Зазначений у минулому реченні алгоритм в даний час є найбільш використовуваним у стисненні даних, оскільки він ніби швейцарський армійський ніж серед усіх алгоритмів.

Крім використання у форматах PNG і ZIP, алгоритм DEFLATE також дуже часто використовується в інших обчислювальних системах. Наприклад, формат файлу gzip (.gz) використовує DEFLATE, оскільки він по суті є версією Zip з відкритим кодом. Інші види використання DEFLATE включають HTTP, SSL та інші технології, призначені для ефективного стиснення даних по мережі.

## Типи стиснення

Стиснення даних буває «без втрат»(lossless) та «з втратами»(lossy). Стискання «без втрат» допомагає відновлювати файл у його початковий стан без втрати жодного з бітів даних при розпакуванні. Таке стиснення – це типовий підхід до застосування з виконавчими файлами, а також текстовими та електронними таблицями, де втрата слів або чисел змінить інформацію. Стискання «без втрат»: видаляє біти, виявляючи статистичні надмірності. Саме через цю техніку ніяка інформація насправді не видаляється. Стиснення без втрат зазвичай має менший ступінь стиснення, що дозволяє не втратити будь-які дані у файлі. Це дуже важливо тоді, коли необхідно підтримувати абсолютну якість, як у випадку з інформацією бази даних або професійними медіа файлами. Такі формати, як FLAC і PNG, пропонують варіанти стиснення без втрат.

Таке стиснення, як lossy, назавжди видаляє ті біти даних, які непотрібні через надлишковість. Вони непотрібні й не впливають на вихідний результат декомпресії. Можливо деякі з них навіть непомітні. Таке корисно для графіки, аудіо, відео та зображень, де видалення деяких бітів даних практично не впливає на представлення вмісту. Стискання «з втратами»: зменшує розмір за рахунок видалення непотрібної інформації та зменшення складності існуючих бітів інформації. Стиснення з втратами може забезпечити набагато вищі коефіцієнти стиснення через можливе погіршення якості файлів. JPEG пропонує варіанти стиснення з втратами, а MP3 і то заснований на стисненні з втратами.

Стиснення графічних зображень може бути «з втратами» або «без втрат». Формати файлів графічних зображень зазвичай призначені для стиснення інформації, оскільки файли, як правило, великі. JPEG – це формат файлу зображення, який підтримує стиснення зображень з втратами. Такі формати, як GIF і PNG, використовують стиснення без втрат. [[18]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1) [[19]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ_1)

## Переваги та недоліки стискання даних

Основними перевагами стискання файлів є скорочення розміру файлу на носії, часу передачі даних і пропускної здатності зв'язку, а також економія витрат. Стиснутий файл вимагає меншого обсягу пам'яті, ніж нестиснений файл, і використання стиснення може привести до значного зниження витрат на дискові та/або твердотільні накопичувачі. Стиснутий файл також вимагає менше часу для передачі і споживає менше пропускної здатності мережі, ніж нестиснений файл.

Основним недоліком стиснення даних є вплив на продуктивність, що виникає в результаті використання ресурсів процесора і пам'яті для стиснення даних і виконання декомпресії. Багато компаній розробили і досі розробляють свої системи, щоб спробувати звести до мінімуму вплив інтенсивних обчислень, пов'язаних зі стисненням. Якщо стиснення виконується вбудовано, перед записом даних на диск, система може розвантажити або, іншими словами, вбити процес стиснення, щоб зберегти системні ресурси. Наприклад, IBM використовує окрему карту апаратного прискорення для обробки стиснення з деякими зі своїх корпоративних систем зберігання даних.

Якщо дані стискаються після запису на диск або після обробки, стиснення може виконуватися у фоновому режимі, щоб знизити вплив на продуктивність. Хоча стиснення після такої операції може скоротити час відгуку для кожного вводу - виводу (I/O), воно як і раніше споживає пам'ять, процесорні цикли і може вплинути на загальну кількість операцій вводу-виводу, з якими може впоратися система зберігання. Крім того, оскільки дані спочатку повинні бути записані на диск або флеш-накопичувач в нестисненому вигляді, економія фізичного сховища не така велика, як при вбудованому стисненні. [[18]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ)

## Поточні програми архівації

Формат ZIP та інші формати, засновані на алгоритмі DEFLATE, були королями аж до середини 1990-х років, коли почали з'являтися нові та поліпшені формати. У 1993 році Євген Рошаль випустив свій архіватор, відомий як WinRAR, який використовує власний формат RAR. Остання версія RAR використовує комбінацію алгоритмів PPM і LZSS, але про ранні реалізації відомо небагато. RAR став стандартним форматом для обміну файлами через Інтернет, особливо при поширенні піратських носіїв. Реалізація перетворення Барроуза-Віллера з відкритим вихідним кодом під назвою bzip2 була представлена в 1996 році і швидко набрала популярність на платформі UNIX в порівнянні з форматом GZIP на основі DEFLATE.

Ще одна програма стиснення з відкритим вихідним кодом була випущена в 1999 році у форматі 7-Zip або .7z. 7-Zip може стати першим форматом, який кине виклик домінуванню ZIP і RAR через його, як правило, високий ступінь стискання, модульність та відкритості формату. Цей формат не обмежується використанням одного алгоритму стиснення, але замість цього може вибирати між алгоритмами bzip2, LZMA, LZMA2 і PPMd серед інших. Нарешті, на передовій архівного програмного забезпечення знаходяться формати PAQ. Перший формат PAQ був випущений Метом Махоні в 2002 році під назвою PAQ1. Цей алгоритм істотно покращує алгоритм PPM використовуючи метод, відомий як змішування контексту, який об'єднує дві або більше статистичних моделей для отримання кращого передбачення наступного символу, ніж будь-яка з інших моделей.

## Майбутні розробки

Майбутнє ніколи не буває певним. Але, виходячи з поточних тенденцій, можна зробити деякі прогнози щодо того, що може статися в майбутньому при стисненні даних. Алгоритми змішування контексту, такі як PAQ і його варіанти, почали ставати дедалі популярнішими. І вони, як правило, досягають найвищих коефіцієнтів стиснення, але зазвичай працюють повільно. З експоненціальним збільшенням швидкості апаратного забезпечення відповідно до закону Мура, алгоритми змішування контексту, ймовірно, будуть процвітати, оскільки штрафи за швидкість нівелюються за рахунок більш швидкого обладнання і на виході отримуємо високий ступінь стиснення. Алгоритм, який PAQ прагнув поліпшити, називається прогнозом шляхом часткового зіставлення (PPM), також може обзавестися модифікаціями та нащадками. Нарешті, ланцюговий алгоритм Лемпеля – Зіва – Маркова (LZMA) незмінно демонструє відмінний компроміс між швидкістю компресії та високим ступенем стиснення. Ймовірно, в майбутньому цей алгоритм породить більше варіантів. LZMA може навіть стати "переможцем" серед всіх алгоритмів у подальшого розвитку, оскільки він вже був прийнятий на озброєння багатьма конкуруючими форматами стиснення з тих пір, як його було введено з форматом 7-Zip. Ще одним потенційним розвитком є використання стиснення за допомогою перерахування під рядків (CSE), який є перспективним методом стиснення, який ще не бачив багатьох програмних реалізацій. У своїй наївній формі він працює аналогічно bzip2 і PPM, і дослідники працюють над підвищенням його ефективності. [[23]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ)

# РОЗДІЛ 2. ОГЛЯД МЕТОДІВ СТИСНЕННЯ

## Вступ

Для стиснення даних використовується безліч різних методів. Більшість методів стиснення не можуть стояти самі по собі, але повинні бути об'єднані разом, щоб сформувати алгоритм стиснення. Навіть ті, які можуть стояти окремо, часто більш ефективні, коли вони об'єднані з іншими методами стиснення. Більшість з цих методів підпадають під категорію ентропійного кодування. Але є й інші, такі як кодування довжини пробігу і перетворення Барроуза-Віллера, які також широко використовуються. Також в даному розділі розглянемо такі методи, як ентропійне кодування, алгоритм Шеннона – Фано, алгоритм Хаффмана та арифметичне кодування.

## Кодування довжин серій (RLE)

Кодування довжин серій або Run-Length Encoding – це дуже простий метод стиснення, який замінює прогони двох або більше однакових символів числом, що представляє довжину прогону, за яким слід вихідний символ; окремі символи кодуються як прогони одиниці. RLE корисний для сильно надлишкових даних, індексованих зображень з великою кількістю пікселів одного кольору в рядку або в поєднанні з іншими методами стиснення, такими як перетворення Барроуза-Віллера.

Розглянемо приклад. Нехай у нас є чорно – біле зображення, яке складається з чорних або білих пікселів. Дано такий масив пікселів:

“WWWWBBBBBBBBBBBBBBWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWB”

Застосуємо до цього алгоритму RLE та продемонструємо вихідний масив та деяку статистику. Треба пам’ятати, що обсяг вихідного масиву може зрости якщо вхідний масив містить багато неповторюваних символів:

Вихідний масив: “4W14B34W1B”

Вхідна кількість символів: 53 символи

Вихідна кількість символів: 10 символів

Співвідношення: 81,1%

Проблему зростання кількості знаків досить легко вирішити. В цьому випадку алфавіт, який застосовувався для запису довжини серії розділяється навпіл. Наприклад розділити цілі числа на додатні, від’ємні та використати додатні для запису кількості повторюваних символів, а від’ємні – неоднакових.

## Перетворення Барроуза – Віллера (BWT)

Перетворення Барроуза-Віллера – це метод стиснення, винайдений в 1994 році Девідом Віллером та Майклом Барроузом, який був його студентом. Цей алгоритм спрямований на оборотне перетворення блоку вхідних даних таким чином, щоб кількість запусків ідентичних символів було максимальним. Сам BWT не виконує жодних операцій стиснення, він просто перетворює вхідні дані таким чином, щоб їх можна було більш ефективно кодувати за допомогою кодування довжини виконання або іншого методу вторинного стиснення.

Алгоритм для BWT простий:

* Створити масив рядків.
* Згенерувати всі можливі повороти вхідного рядка, зберігаючи кожен з них в масиві.
* Відсортувати масив за алфавітом.
* Повернути останній стовпчик масиву. [[24]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ)

BWT зазвичай найкраще працює на довгих вхідних даних з великою кількістю однакових символів, які чергуються. Наведемо приклад алгоритму, що виконується на ідеальному вході. Зверніть увагу, що & - це кінець символу файлу:

|  |  |  |  |
| --- | --- | --- | --- |
| Вхідні дані | Обертання | Альфа-Сортовані Обертання | Вихідні дані |
| HAHAHA& | HAHAHA& | AHAHA&**H** | **HHH&AAA** |
| &HAHAHA | AHA&HA**H** |
| A&HAHAH | A&HAHA**H** |
| HA&HAHA | HAHAHA**&** |
| AHA&HAH | HAHA&H**A** |
| HAHA&HA | HA&HAH**A** |
| AHAHA&H | &HAHAH**A** |

Таблиця 1. Приклад перетворення BWT

Через чергування однакових символів, виконання BWT на цьому вході генерує оптимальний результат, який інший алгоритм може додатково стиснути, наприклад RLE, який дасть "3H&3A". Хоча цей приклад дав оптимальний результат, він не дає оптимальних результатів для більшості реальних даних.

Ряд методів оптимізації можуть зробити цей алгоритм більш ефективним без зміни вихідних даних. Немає необхідності представляти таблицю ні в кодері, ні в декодері. У кодері кожен рядок таблиці може бути представлений одним покажчиком на рядки, а сортування виконується з використанням індексів. У декодері також немає необхідності зберігати таблицю, і насправді сортування взагалі не потрібно. У часі, пропорційному розміру алфавіту і довжині рядка, декодований рядок може генеруватися по одному символу за раз справа наліво. "Символ" в алгоритмі може бути байтом, бітом або будь-яким іншим зручним розміром. Можна також зробити зауваження, що математично закодований рядок може бути обчислений як проста модифікація масиву суфіксів, а масиви суфіксів можуть бути обчислені з лінійним часом і пам'яттю. BWT може бути визначений щодо масиву суфіксів SA тексту T як (індексування на основі 1):

Немає необхідності мати фактичний символ "EOF". Замість цього можна використовувати покажчик, який запам'ятовує, де в рядку був би "EOF", якби він існував. При такому підході вихідні дані BWT повинні включати як перетворений рядок, так і кінцеве значення покажчика. Зворотне перетворення потім стискає його назад до вихідного розміру: йому дається рядок, покажчик і повертається тільки рядок.

## Динамічний алгоритм BWT

При редагуванні тексту його перетворення Барроуза-Віллера зміниться. Мікаель Салсон у своїй статті пропонує алгоритм, який виводить перетворення Барроуза–Віллера відредагованого тексту з вихідного тексту, виконуючи обмежену кількість локальних переупорядкувань у вихідному перетворенні Барроуза–Віллера. Це може бути швидше, ніж безпосередньо побудова перетворення трансформації Барроуза–Віллера відредагованого тексту. [[26]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ)

## Ентропійне кодування (Entropy coding)

Ентропія при стисненні даних означає найменшу кількість бітів, необхідна в середньому для представлення символу або літералу. Базовий кодер ентропії поєднує в собі статистичну модель і кодер. Вхідний файл аналізується і використовується для створення статистичної моделі, яка складається з ймовірностей появи даного символу. Потім кодер буде використовувати статистичну модель, щоб визначити, які біти або байт-коди призначати кожному символу таким чином, щоб найбільш поширені символи мали найкоротші коди, а найменш поширені символи мали найдовші коди. [[24]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ)

Двома найпоширенішими методами ентропійного кодування є кодування Хаффмана та арифметичне кодування.[[25]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ) Якщо приблизні характеристики ентропії потоку даних відомі заздалегідь (особливо для стиснення сигналу), може бути корисний простіший статичний код. Ці статичні коди включають універсальні коди (такі як гамма-кодування Еліаса або кодування Фібоначчі) і коди Голомба (такі як унарне кодування або кодування рису).

З 2014 року Компресори даних почали використовувати сімейство методів ентропійного кодування асиметричних числових систем, що дозволяє поєднувати ступінь стиснення арифметичного кодування з витратами на обробку, аналогічними кодування Хаффмана.

## Алгоритм Шеннона – Фано (Shannon – Fano coding)

Це один з найбільш ранніх методів стиснення, винайдений в 1949 році Клодом Шенноном і Робертом Фано. Цей метод включає в себе створення двійкового дерева для представлення ймовірностей появи кожного символу. Символи впорядковані таким чином, що найбільш часто зустрічаються символи з'являються у верхній частині дерева, а найменш ймовірні – в нижній. Код для даного символу виходить шляхом пошуку його в дереві Шеннона – Фано і додавання до коду значення 0 або 1 для кожної взятої лівої або правої гілки відповідно. Наприклад, якщо" А "- це дві гілки зліва і одна справа, її код буде "0012". Кодування Шеннона – Фано не завжди дає оптимальні коди через те, що воно будує двійкове дерево знизу вгору. З цієї причини замість цього використовується кодування Хаффмана, оскільки воно генерує оптимальний код для будь-якого заданого введення.

Алгоритм генерації кодів Шеннона-Фано досить простий

* Проаналізувати вхідні дані, підрахувавши появу кожного символу.
* Визначити ймовірність кожного символу, використовуючи кількість символів.
* Відсортувати символи за ймовірністю, спочатку вибравши найбільш ймовірні.
* Створити кінцеві вузли для кожного символу.
* Розділити список на дві частини, зберігаючи ймовірність лівої гілки приблизно рівною ймовірності правої гілки.
* Додати 0 і 1 до кодів лівого і правого вузлів відповідно.
* Рекурсивно застосовувати кроки 5 і 6 до лівого і правого піддерев'я, поки кожен вузол не стане листом в дереві. [[27]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ)

## Алгоритм Хаффмана (Huffman coding)

Кодування Хаффмана – це ще один варіант ентропійного кодування, який працює дуже схоже на кодування Шеннона – Фано, але двійкове дерево будується зверху вниз для отримання оптимального результату.

Алгоритм генерації кодів Хаффмана схожий своїми першими кроками з Шенноном – Фано:

1. Проаналізувати вхідні дані, підрахувавши появу кожного символу.
2. Визначити ймовірність кожного символу, використовуючи кількість символів.
3. Відсортувати символи за ймовірністю, спочатку вибравши найбільш ймовірні.
4. Створити кінцеві вузли для кожного символу, включаючи P, і додати їх в чергу.
5. Допоки (вузли в черзі > 1)

* Видалити з черги два вузли з найменшою ймовірністю.
* Додати 0 і 1 до кодів лівого і правого вузлів відповідно.
* Створити новий вузол зі значенням, рівним сумі ймовірностей вузлів.
* Призначити перший вузол до лівої гілки, а другий вузол – до правої гілки.
* Додати вузол в чергу

1. Останній вузол, що залишився в черзі, є коренем дерева Хаффмана. [[28]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ)

## Арифметичне кодування (Arithmetic coding)

Цей метод було розроблено в 1979 році в IBM, яка вивчала методи стиснення даних для використання в своїх мейнфреймах. Арифметичне кодування, можливо, є найбільш оптимальним методом ентропійного кодування, якщо метою є найкраща ступінь стиснення, оскільки воно зазвичай досягає кращих результатів, ніж кодування Хаффмана. Однак це досить складно в порівнянні з іншими методами кодування. Замість того, щоб розбивати ймовірності символів на дерево, арифметичне кодування перетворює вхідні дані в одне раціональне число від 0 до 1, змінюючи базу і присвоюючи кожному унікальному символу одне значення від 0 до бази. Потім він далі перетворюється в двійкове число з фіксованою точкою, яке є закодованим результатом. Значення може бути декодовано в вихідний висновок, змінивши базу з двійковою назад на вихідну базу і замінивши значення символами, яким вони відповідають.

Загальний алгоритм обчислення арифметичного коду:

1. Обчислити кількість унікальних символів у вхідних даних. Це число являє собою базу b (наприклад, база 2 є двійковим) арифметичного коду.
2. Призначити значення від 0 до b кожному унікальному символу в порядку їх появи.
3. Використовуючи значення з кроку 2, замінити символи у вхідних даних їх кодами
4. Перетворити результат кроку 3 з бази b в досить довге двійкове число з фіксованою точкою, щоб зберегти точність.
5. Записати довжину вхідного рядка де – небудь в результаті, так як це необхідно для декодування. [[29]](#_ПЕРЕЛІК_ВИКОРИСТАНИХ_ДЖЕРЕЛ)

Наведемо приклад арифметичного кодування:

Дано масив " ABCDAABD”

1. Знайдено 4 унікальних символи у вхідних даних, отже, база = 4. Довжина = 8
2. Присвоєні значення символам: A=0, B=1, C=2, D=3
3. Замінено вхідні дані кодами: "0.012300134", де початковий 0 не є символом.
4. Перетворено " 0.012311234 "з бази 4 в базу 2:" 0.011011000001112”
5. Результат знайдено. Звертаємо увагу, що в результаті довжина вхідного сигналу дорівнює 8.

Припускаючи 8-бітові символи, вхідний сигнал має довжину 64 біта, в той час як його арифметичне кодування становить всього 15 біт, що призводить до відмінної ступеня стиснення у 24%. Цей приклад демонструє, як арифметичне кодування добре стискається при обмеженому наборі символів.

# РОЗДІЛ 3. ОГЛЯД АЛГОРИТМІВ СТИСНЕННЯ

## Алгоритми «Ковзаючого вікна» (Sliding window)

## Алгоритм Лемпеля – Зіва та його модифікацій

## DEFLATE та DEFLATE64

## Алгоритми з використанням словника

## Алгоритми без використання словників

## Передбачення щодо часткового збігу (PPM)

## Bzip2

## PAQ

# РОЗДІЛ 4. ІМПЛЕМЕНТАЦІЯ МЕТОДІВ І АЛГОРИТМІВ СТИСНЕННЯ

## Перетворення Барроуза – Віллера

## Стандартний метод LZW

## Метод LZW з перетворенням Барроуза-Віллера

## Метод Хаффмана (Huffman coding)

## РОЗДІЛ 5. ПОРІВНЯННЯ РЕЗУЛЬТАТІВ СТИСКАННЯ

# 5.1 Порівняння розроблених методів між собою

# 5.2 Порівняння розроблених методів з алгоритмами та програмами, представленими на ринку

# ВИСНОВКИ

# ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Jeb Brains Rider [Електронний ресурс]. – Режим доступу до ресурсу: <https://www.jetbrains.com/rider/>.
2. C# programming language [Електронний ресурс] – Режим доступу до ресурсу: <https://en.wikipedia.org/wiki/C_Sharp_(programming_language)>.
3. .NET Library [Електронний ресурс] // Microsoft. – 2018. – Режим доступу до ресурсу: <https://docs.microsoft.com/en-us/dotnet/standard/library-guidance/>.
4. ASP.NET Framework [Електронний ресурс] // Microsoft. – 2021. – Режим доступу до ресурсу: <https://dotnet.microsoft.com/apps/aspnet>.
5. Стівен В. Новий вид Науки / Вольфрам Стівен. – Іллінойс: Вольфрам Медіа, 2002. – 1069 с.
6. Lossless Data Compression: LZ77 [Електронний ресурс] – Режим доступу до ресурсу: <https://cs.stanford.edu/people/eroberts/courses/soco/projects/data-compression/lossless/lz77/index.htm>.
7. Lossless Data Compression: LZ78 [Електронний ресурс] – Режим доступу до ресурсу: <https://cs.stanford.edu/people/eroberts/courses/soco/projects/data-compression/lossless/lz78/index.htm>.
8. Huffman encoding [Електронний ресурс] – Режим доступу до ресурсу: <http://homes.sice.indiana.edu/yye/lab/teaching/spring2014-C343/huffman.php>.
9. Burrows-Wheeler Transform [Електронний ресурс] – Режим доступу до ресурсу: <https://web.stanford.edu/class/cs262/presentations/lecture4.pdf>.
10. Sliding Window Algorithm [Електронний ресурс] – Режим доступу до ресурсу: <https://www.baeldung.com/cs/sliding-window-algorithm>.
11. Lossless Data Compression: Shannon-Fano [Електронний ресурс] – Режим доступу до ресурсу: <https://cs.stanford.edu/people/eroberts/courses/soco/projects/data-compression/lossless/shannonfano/index.htm>.
12. LZW Encoding [Електронний ресурс] – Режим доступу до ресурсу: <http://www.cs.columbia.edu/~allen/S14/NOTES/lzw.pdf>.
13. Grunwald P. D. Algorithmic Information Theory: дис. докт. філос. наук / Grunwald Peter D. – Амстердам, 2007. – 37 с.
14. Rate-Distortion Theory [Електронний ресурс] – Режим доступу до ресурсу: <https://web.stanford.edu/class/ee368b/Handouts/04-RateDistortionTheory.pdf>.
15. Huffman D. Encoding the “Neatness” of Ones and Zeroes / David Huffman. // Scientific American. – 1991. – С. 54–58.
16. Ziv J. Compression of individual sequences via variable-rate coding: 10.1109/TIT.197 / Ziv Jacob, 1978. – 530 с.
17. Phil Katz [Електронний ресурс]. – 2002. – Режим доступу до ресурсу: <http://www.esva.net/~thom/philkatz.html>.
18. Crocetti P. Data Compression [Електронний ресурс] / Paul Crocetti – Режим доступу до ресурсу: <https://searchstorage.techtarget.com/definition/compression>.
19. What is Data Compression? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.barracuda.com/glossary/data-compression>.
20. Wayner P. Compression / Peter Wayner // Disappearing Cryptography (Third Edition) / Peter Wayner., 2009.
21. Mengyi Pu I. Dictionary-based compression / Ida Mengyi Pu // Fundamental Data Compression / Ida Mengyi Pu., 2006.
22. What about patents on data compression algorithms? [Електронний ресурс] // 2014 – Режим доступу до ресурсу: <http://www.faqs.org/faqs/compression-faq/part1/section-7.html>.
23. Iwata K. An Improvement in Lossless Data Compression via Substring Enumeration / Iwata K. – IEEE/ACIS 10th International Conference on Compute, 2011.
24. Wheeler D. J. A Block-sorting Lossless Data Compression Algorithm / D. J. Wheeler, M. Burrows. // SRC Research Report. – 1994. – №124. – С. 1–24.
25. Huffman D. A. A Method for the Construction of Minimum-Redundancy Code / David Huffman. // IEEE. – 1952. – №40. – С. 1098–1101.
26. A four-stage algorithm for updating a Burrows–Wheeler transform / M.Salson, T. Lecroq, M. Leonard, L. Mouchard. // String Algorithmics: Dedicated to Professor Maxime Crochemore on his 60th birthday. – 2009. – №410. – С. 4350–4359.
27. Shannon C. E. A Mathematical Theory of Communication / Shannon. // Bell System Technical Journal. – 1948. – №27. – С. 379–423.
28. Huffman D. A. A method for the construction of minimum-redundancy codes. / Huffman. // In Proceedings of the Institute of Electrical and Radio Engineers. – 1952. – №40. – С. 1098–1101.
29. Rissanen J. Arithmetic coding / J. Rissanen, G. G. Langdon. // IBM Journal of Research & Development. – 1979. – №23. – С. 149–162.

# ДОДАТКИ

ДОДАТОК А