ЗВІТ

Команда №4
Єлізавета Пілецька, Діана Лизенко, Анастасія Яблуновська, Гобела Максим
Ментори: Юлія Колодій, Андрій Музичук

Дослідження методів компресії даних та створення власної програми-кодека

ВСТУП

У даному проекті нам пропонується розробити кілька алгоритмів стиснення та проаналізувати їх роботу з текстовими файлами, зображеннями, відео, аудіо тощо. Для цього ми імплементували кілька алгоритмів, а саме: **LZ77, LZW, алгоритм Хаффмана, Deflate (LZ77 + Хаффмана)**. Варто також зазначити, що усі формати спочатку конвертуються у нас в бітові рядки. Для розподілу роботи ми вирішили зробити подальшу схему:

- **Для LZ77:** переважно тестуватимуться тексти та відео;
- **Для LZW:** та алгоритму Хаффмана - тексти та аудіофайли;
- **Для Deflate:** зображення.

ВИМОГИ

Для роботи алгоритму Deflate знадобиться встановити бібліотеки **heapq, pickle, datetime**. Також необхідна бібліотека **argparse**, яка буде запускати основну програму.

ЗАПУСК

Для запуску потрібно ввести у термінал наступну команду:

```
python3 main_argparse.py text.txt lz77
```

Тобто, як видно, це назва файлу + документ, що цікавить + алгоритм, за яким Ви хочете здійснити стиснення.

LZ77

Техніка стиснення LZ77 зменшує розмір даних, визначивши та кодуючи повторювані шаблони в вхідному потоці. Він використовує механізм "розсувного вікна"для виявлення найбільш розширених послідовностей подібності між нинішнім набором даних та тими, які були проаналізовані раніше. Ідентифіковані відповідності замінюються на кортежі (відстань, довжина), що означає розташування та ступінь повторюваної схеми. **Для файлів з широким повторюваним вмістом (наприклад, текст) алгоритм LZ77 працює з високою ефективністю. Тим не менш, при роботі з вже стислими даними (наприклад, відеофайлів MP4), може бути незначний вплив або потенційне збільшення розміру файлів через додаткові метадані**.
Псевдокод:

```
ФУНКЦІЯ lz77_compress(вхідні_дані, розмір_вікна=100):
    ініціалізувати порожній список стиснутих даних
    ініціалізувати пустий буфер вікна
    позиція = 0

    ПОКИ позиція < довжина_вхідних_даних:
        відстань = 0
        довжина = 0
        буфер_перегляду = вхідні_дані[позиція:]

        ДЛЯ j = 1 ДО довжина_буфера_перегляду:
            підрядок = буфер_перегляду[:j]
            позиція_у_вікні = остання_позиція(підрядок у вікні)

            ЯКЩО підрядок знайдено у вікні:
                відстань = довжина_вікна - позиція_у_вікні
                довжина = j
            ІНАКШЕ:
                вийти з циклу

        ЯКЩО довжина > 0:
            наступний_символ = символ після знайденої послідовності
        ІНАКШЕ:
            наступний_символ = поточний символ

        додати (відстань, довжина, наступний_символ) до стиснутих даних
        зсув = довжина + 1
        додати зсунуті дані до вікна
        обрізати вікно, якщо воно перевищує максимальний розмір
        позиція += зсув

    ПОВЕРНУТИ стиснуті дані

ФУНКЦІЯ lz77_decompress(стиснуті_дані):
    ініціалізувати порожній результат

    ДЛЯ КОЖНОГО (відстань, довжина, символ) у стиснутих_даних:
        ЯКЩО відстань = 0 І довжина = 0:
            додати символ до результату
        ІНАКШЕ:
            початок = довжина_результату - відстань
            ДЛЯ i = 0 ДО довжина-1:
                додати результат[початок + i] до результату
            додати символ до результату

    ПОВЕРНУТИ результат
```

LZW

Під час процесу стиснення алгоритм LZW конструює лексикон на основі зіткнених даних, замінюючи вихідну послідовність даних відповідними представленнями коду. **Він виявляється особливо вигідним для файлів документів, однак для аудіо файлів він дає неймовірно мале стиснення. Це так само може бути пов'язаним з форматом даних**.

Псевдокод:

```
ФУНКЦІЯ lzw_encode(дані):
    ініціалізувати таблицю з усіма можливими однобайтовими символами
    наступний_код = 256
    P = перший байт даних
    результат = []

    ДЛЯ КОЖНОГО наступного байту C в даних:
        ЯКЩО P + C є в таблиці:
            P = P + C
        ІНАКШЕ:
            додати код P до результату
            додати P + C до таблиці з кодом наступний_код
            збільшити наступний_код
            P = C

    додати код для P до результату
    ПОВЕРНУТИ результат

ФУНКЦІЯ lzw_decode(коди):
    ініціалізувати таблицю з усіма можливими однобайтовими символами
    наступний_код = 256
    СТАРИЙ = перший код
    S = таблиця[СТАРИЙ]
    додати S до результату
    C = перший байт S

    ДЛЯ КОЖНОГО НОВОГО коду в кодах:
        ЯКЩО НОВИЙ відсутній у таблиці:
            S = таблиця[СТАРИЙ] + C
        ІНАКШЕ:
            S = таблиця[НОВИЙ]

        додати S до результату
        C = перший байт S
        додати (таблиця[СТАРИЙ] + C) до таблиці з кодом наступний_код
        збільшити наступний_код
        СТАРИЙ = НОВИЙ

    ПОВЕРНУТИ результат
```

Алгоритм Хаффмана

Алгоритм Хаффмана використовує частоту символів у даних для побудови оптимального префіксного коду. Символи, які зустрічаються частіше, отримують короткі коди, а рідкісні — довші. **Цей метод добре працює для текстових файлів, де частота символів нерівномірна. Однак для аудіо або відео, які часто вже стиснуті (наприклад, MP3 або MP4), алгоритм Хаффмана дає незначний ефект, оскільки дані вже майже оптимізовані.**

Псевдокод:

```
ФУНКЦІЯ build_huffman_tree(частоти):
    створити список вузлів із байтів та їхніх частот
    відсортувати список за частотою (від найменшої)
    ПОКИ в списку більше 1 вузла:
        взяти два вузли з найменшою частотою
        створити новий вузол з частотою, рівною сумі частот цих вузлів
        встановити ці вузли як лівий та правий нащадки нового вузла
        додати новий вузол до списку
        відсортувати список
    ПОВЕРНУТИ корінь дерева

ФУНКЦІЯ build_codes(корінь):
    створити пустий словник кодів
    використати стек для обходу дерева
    ПОКИ стек не пустий:
        взяти вузол та його код зі стеку
        ЯКЩО вузол має символ:
            додати символ та код до словника кодів
        ІНАКШЕ:
            додати до стеку лівий нащадок з кодом + "0"
            додати до стеку правий нащадок з кодом + "1"
    ПОВЕРНУТИ словник кодів

ФУНКЦІЯ compress_file(шлях_до_файлу):
    прочитати дані з файлу
    побудувати словник частот
    побудувати дерево Хаффмана
    створити коди для символів
    закодувати дані, використовуючи створені коди
    додати паддінг до кінця, щоб мати цілу кількість байтів
    зберегти закодовані дані та коди у файл з розширенням .huff
    ПОВЕРНУТИ шлях до стисненого файлу

ФУНКЦІЯ decompress_file(шлях_до_стисненого_файлу):
    завантажити закодовані дані, словник кодів, довжину біт та ім'я оригінального файлу
    створити обернений словник кодів (код -> символ)
    перетворити байти на рядок бітів і обрізати надлишкові біти
    декодувати біти, використовуючи обернений словник кодів
    зберегти декодовані дані у файл
    ПОВЕРНУТИ шлях до відновленого файлу
```

Deflate (LZ77 + Хаффмана)

Deflate поєднує LZ77 для попереднього стиснення даних (шляхом усунення повторень) та алгоритм Хаффмана для подальшого кодування. Це робить його універсальним для різних типів даних. У нашій інтерпретації Deflate це LZ77 + алгоритм Хаффмана, комібнований з хешуванням, який робить процес швидшим та конвертуванням файла у байти і навпаки. **Тестування на зображеннях показали, що така версія алгоритму дуже добре підходить для нестиснутих форматів, (bmp, tiff) і є дуже проблемною для стиснутих, (png, jpg).**При взаємодії з отсанніми форматами алгоритм зробить об'єм пам'яті лише більшим.

Псевдокод:

```
ФУНКЦІЯ deflate_bit_compress(ім'я_файлу):
    прочитати дані з файлу

    // Перший етап: LZ77 з використанням хеш-таблиці для швидкого пошуку
    ДЛЯ КОЖНОЇ позиції у даних:
        знайти найдовшу відповідність у ковзному вікні, використовуючи хеш-таблицю
        ЯКЩО довжина відповідності >= 3:
            додати (відстань, довжина) до результату
            просунутись на довжину відповідності
        ІНАКШЕ:
            додати літеральний байт до результату
            просунутись на 1

    // Перетворити LZ77-стиснуті дані в послідовність байтів
    перетворити результат LZ77 у байти:
        для кортежів (відстань, довжина) використовувати маркер 0
        для літеральних байтів використовувати маркер 1

    // Другий етап: Стиснення Хаффмана
    побудувати дерево Хаффмана на основі частот байтів
    створити таблицю кодів Хаффмана
    закодувати LZ77-байти, використовуючи таблицю кодів
    перетворити закодовані біти в байти

    // Формування результату
    серіалізувати таблицю кодів
    додати розмір серіалізованої таблиці до заголовка
    повернути: заголовок + серіалізована таблиця + стиснуті дані

ФУНКЦІЯ inflate_bit_decompress(стиснуті_дані):
    прочитати розмір таблиці кодів із заголовка
    прочитати та десеріалізувати таблицю кодів
    прочитати стиснуті байти

    // Перший етап: Декодування Хаффмана
    перетворити стиснуті байти на бітовий рядок
    декодувати бітовий рядок за допомогою таблиці кодів

    // Другий етап: Декодування LZ77
    перетворити декодовані байти назад у формат LZ77
    декомпресувати дані LZ77:
        для кортежів (відстань, довжина) копіювати дані з попередньої позиції
        для літеральних байтів додати байт до результату

    повернути декомпресовані дані
```

ВИСНОВОК

Підсумовуючи, можна сказати, що алгоритми працюють добре, особливо для текстових файлів. Однак для відео аудіо та зображення можуть виникнути проблеми через їхній формат, оскільки стиснення уже стиснутих даних може призводити до збільшення об'єму пам'яті файлу.

РОЗПОДІЛ РОБОТИ:

- **Лизенко Діана**: реалізація LZ77, тестування;
- **Пілецька Єлізавета**: LZW, DEFLATE, тестування аудіо файлів, реалізація файлу README, презентація;
- **Яблуновська Анастасія**: Huffman Coding, тестування текстових файлів, реалізація argprase, презентація;
- **Гобела Максим**: DEFLATE, тестування файлів із зображенням.