Алгоритми перетворення інформації.
Завдання 1, звіт

Михайло Голуб

27 лютого 2025 р.

Реалізація функції `encode(in_bytes)` для RLE:

На вхід приймається масив байтів `in_bytes`. Створюються пусті масиви байтів `output` та `bufer`.

Для кожного байту з вхідного масиву:

1. **Перевірка на переповнення буфера:**
   Якщо довжина буферу більша за 127 і буфер містить різні символи – додати службовий байт "переписати 128 символів"до вихідного масиву (дялі – ВМ), переписати буфер до ВМ та очистити буфер;

   Якщо довжина буферу рівна 129 і буфер містить однакові символи – додати службовий байт "повторення 129 разів"і перший байт з буфера до Вихідного масиву та очистити буфер.

2. **Перевірка на закінчення послідовності різних байтів:**
   Якщо довжина буфера більша за 1 і поточний обраний байт рівний останньому байту буфера, але не рівний передостанньому – завершено послідовність різних байтів. До ВМ додається службовий байт виду `0*******`, де значення молодщих бітів плюс 1 – кількість байтів які треба переписати декодеру після даного службового байту. Після службового байту, до ВМ додається буфер, після чого буфер очищується.

3. **Перевірка на закінчення послідовності однакових байтів:**
   Якщо довжина буфера більша за 1 і поточний обраний байт не рівний останньому байту буфера і останній та передостанній байти буфера рівні – завершено послідовність однакових байтів. До ВМ додається службовий байт виду `1*******`, де значення молодших бітів плюс 2 – кількість повторень наступного байту які має зробити декодер. Після службового байту, до ВМ додається перший байт з буфера, після чого буфер очищується.

4. **Додати поточний байт до буфера**

Після завершення циклу, якщо буфер не порожній, виконуються кроки аналогічні крокам 2 та 3.

Функція повертає сформований вихідний масив байтів `output`

Реалізація функції `decode(in_bytes)` для RLE:

На вхід приймається масив байтів `in_bytes`. Створюється пустий масив байтів `output` та змінна `pointer` Доки значення `pointer` менше за довжину вхідного масиву:

1. Якщо байт на позиції `pointer` у вхідному масиві більший за 127 – записати у ВМ `байт-126` разів байт на позиції `pointer+1`. Збільшити `pointer` на 2.

2. Інакше – переписати наступні `байт+1` байтів у ВМ. Збільшити `pointer` на байт+2.

Функція повертає сформований вихідний масив байтів.

Взаємодія з реалізованими функціями RLE:

Функції `encode` та `decode` знаходяться у файлах `rle_encode.py` та `rle_decode.py`. При запуску файлу з бажаною функцією як головного файлу (відкрити за допомогою інтерпритатора) у консоль виводяться короткі підказки та користувач може ввести бажані шляхи до початкового файлу та кінцевого результату.

Також створено програму `rle_main.py` яка приймає на вхід команди виду `encode;source;destination` та `decode;source;destination` усі символи (у т.ч. пробіли) після ; будуть вважатись частиною шляху.

Приклади правильного використання команд:
`encode;file.txt;` – вихідний файл буде мати шлях file.txt.rle
`decode;file.txt.rle;` – вихідний файл буде мати шлях file.txt
`encode;file.txt;example.com`
`decode;example.com;file.txt`

Тестування RLE:

Програму протестовано на декількох різних файлах: текстових, зображеннях, коротких відео та документах .docs:

- Текстовий файл що містив Гамлета зменшився на менш ніж 900 байтів при початковому розмірі 187 КБайт.

- Текстові файли стереолітографічного (.stl) змісту зменшились на 5-10% через наявність певних ділянок однакових цифер в числах

- Зображення "чорний текст, білий фон"зменшили свій розмір на 15-20%

- Зображення пейзажів збільшили свій розмір на 1-2% через неповторюваність їх змісту

- Файли з короткими відео збільшились на 1-2%, оскільки вже мають якесь певне кодування та повторення байтів в них – рідкість

Висновки:

- RLE ефективний для файлів з великою кількістю повторюванних байтів;
- RLE не ефективний для файлів які вже стиснуті дуже добре, та/або не мають однакових байтів підряд.

Посилання на код на GitHub:
`https://github.com/MINIAProgramStudio/algorythms_of_data_transformation/tree/main/task1`