Міністерство освіти і науки, молоді та спорту України Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського" Фізико - технічний інститут

Проєкт

З теми “Коп’ютерна графіка”

LZW-стиснення зображень

Виконав:

Студент групи ФІ-21, НН ФТІ

Мелоян Мирослав

Зміст

Загальний огляд 3
Постановка задачі 3
Методи розв'язання задачі 3
Стиснення: 3
Декодування 4
Розрахунок ентропії 5
Відношення стиснення 6
Кількість біт на символ у стиснених даних 6
Розмір оригінальних даних у бітах 6
Розмір стиснених даних у бітах 7
Лістинг проєкту 7
Опис файлів 7
lzw.py 7
compress_image.py 8
decompress_image.py 11
main.py 12
Результати роботи 16
Висновок 18

Загальний огляд

Цей проект має на меті стиснення зображень у відтінках сірого за допомогою алгоритму **Лемпеля-Зіва-Велча (LZW)**. Алгоритм LZW - це алгоритм стиснення без втрат, який замінює підрядки, що часто зустрічаються, на коротші кодові слова.

Проект включає кодер, який стискає вхідне зображення в закодоване, і декодер, який приймає закодоване зображення і реконструює вихідне зображення. Створено функції для обчислення коефіцієнта стиснення, максимального стиснення та ентропії для зображення.

Постановка задачі

1. Реалізувати алгоритм LZW для стиснення зображень у відтінках сірого.
2. Створити кодер та декодер LZW алгоритму стиснення.
3. Отримати стиснене зображення з якістю оригіналу.
4. Обчислити показники стиснення.
5. Проаналізувати алгоритм для різних зображень.

Методи розв'язання задачі

Стиснення:

LZW-стиснення зчитує послідовність символів, групує їх у рядки і перетворює в коди. Коди займають менше місця, ніж рядки, які вони замінюють, тому ми отримуємо стиснення.

Високорівневий вигляд алгоритму кодування показано тут:

1. Створюється словник, щоб він містив усі рядки довжиною один символ;
2. Знаходження найдовшого рядка W в словнику, що відповідає поточному введенню;

3. Виведення індексу словника для W у вихідні дані та видалення W з введення;

4. Додавання W, за яким слідує наступний символ у введенні, до словника;

Словник ініціалізовано так, щоб він містив односимвольні рядки, які відповідають усім можливим вхідним символам (і нічого, окрім кодів очищення та зупинки, якщо вони використовуються). Алгоритм працює, скануючи вхідний рядок на наявність послідовно довших підрядків, поки не знайде той, якого немає у словнику. Коли такий рядок знайдено, індекс рядка без останнього символу (тобто найдовшого підрядка зі словника) витягується зі словника і надсилається на вивід, а новий рядок (включно з останнім символом) додається до словника з наступним доступним кодом. Останній введений символ використовується як наступна точка відліку для пошуку підрядків.

Таким чином, послідовно довші рядки заносяться до словника і стають доступними для подальшого кодування як окремі вихідні значення. Алгоритм найкраще працює на даних з повторюваними шаблонами, тому початкові частини повідомлення не зазнають значного стиснення. Однак, коли повідомлення зростає, коефіцієнт стиснення асимптотично прагне до максимуму (тобто коефіцієнт стиснення покращується по зростаючій кривій, а не лінійно, наближаючись до теоретичного максимуму в межах обмеженого періоду часу, а не протягом нескінченного часу).[1]

Декодування

LZW-декодування зчитує послідовність кодів, перетворює їх назад у рядки символів за допомогою словника і виводить ці рядки. Це дозволяє відновити початкову послідовність символів із стиснених даних:

1. Створення словника, щоб він містив усі рядки довжиною один.
2. Читання наступного закодованого символу. Чи закодований він у словнику?
    1. Так:
        1. Виведення відповідного рядка **W**
        2. Об'єднання попереднього виведеного рядка з першим символом **W**. Додавання його до словника.

[1] Ziv, J.; Lempel, A. (1978). "Compression of individual sequences via variable-rate coding"

2. Ні:
    1. Об'єднання попереднього виведеного рядка з його першим символом. Назвемо цей рядок **V**.
    2. Додавання **V** до словника і виведення **V** на екран.
3. Повторення кроку 2 до кінця вхідного рядка

Алгоритм декодування працює, зчитуючи значення з закодованого входу і виводячи відповідний рядок зі словника. Однак повний словник не потрібен, лише початковий словник, який містить односимвольні рядки (і який зазвичай жорстко кодується в програмі, а не надсилається разом із закодованими даними).

Замість цього повний словник перебудовується під час процесу декодування наступним чином: після декодування значення і виведення рядка, декодер об'єднує його з першим символом наступного декодованого рядка (або з першим символом поточного рядка, якщо наступний рядок не може бути декодований; оскільки якщо наступне значення невідоме, то це має бути значення, додане до словника на цій ітерації, і тому його перший символ збігається з першим символом поточного рядка), і оновлює словник новим рядком. Потім декодер переходить до наступного входу (який вже було прочитано на попередній ітерації) і обробляє його, як і раніше, і так до тих пір, поки не вичерпає вхідний потік.[2]

Розрахунок ентропії

В основі лежить теорія інформації **Клода Шеннона**, яка описує, як вимірювати інформацію та її невизначеність у повідомленнях:

$$H = -\sum_{i=1}^{m} P_i \log_2 P_i$$

Ентропія являє собою логарифмічну міру безладдя стану джерела повідомлень і характеризує середній ступінь невизначеності стану цього джерела.[3]

Вона базується на ймовірностях появи кожного символу P_i у даних. Формула використовує логарифм з основою 2, тому що ентропія вимірюється в бітах. Від’ємний знак використовується для того, щоб зробити значення ентропії

[2] https://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Welch#cite_note-2
[3] https://web.posibnyky.vntu.edu.ua/firen/6bilynskyj_elektronni_systemy/22.htm

додатним, оскільки ймовірності P_i менші за 1, і логарифм буде від'ємним. Сума всіх значень $P_i \log_2 P_i$ дає нам середню кількість біт інформації на символ.

Відношення стиснення

Відношення стиснення показує, наскільки ефективно дані були стиснуті:

$$R = \frac{S_O}{S_C}$$

S_O — це розмір оригінальних даних у бітах, а S_C — розмір стиснених даних у бітах. Якщо відношення більше 1, це означає, що стиснення було ефективним і розмір даних зменшився. Чим більше відношення, тим краще стиснення.

Кількість біт на символ у стиснених даних

$$bits = \lceil \log_2 n \rceil$$

n — це максимальний використовуваний код у словнику. Логарифм з основою 2 використовується, щоб знайти кількість біт, необхідних для кодування n різних символів (оскільки $\log_2 n$ — це кількість біт для кодування n різних значень). Округлення вгору ($\lceil - \rceil$) гарантує, що навіть якщо результат дробовий, ми отримуємо ціле число біт.[4]

Розмір оригінальних даних у бітах

$$S_O = w \times h \times b$$

w— ширина зображення у пікселях, h — висота зображення у пікселях, а b — кількість біт на піксель (для відтінків сірого це зазвичай 8 біт). Множення всіх цих значень дає загальну кількість біт, необхідних для збереження оригінального зображення.

[4] Compression Techniques in Action: Implementing Efficient compression Solution Using java

Розмір стиснених даних у бітах

$$S_C = L \times bits$$

L — це кількість символів у стиснених даних (довжина стисненого масиву даних), а *bits* — кількість біт на символ (визначена попередньою формулою). Множення цих значень дає загальну кількість біт, необхідних для збереження стиснених даних.

Лістинг проєкту

Проєкт включає функції для стиснення і декомпресії зображень, обчислення ентропії та коефіцієнта стиснення, а також графічний інтерфейс для вибору та відображення зображень.

Опис файлів

lzw.py

Базова реалізація алгоритму стиснення і декомпресії **LZW:**

```
def lzw_compress(uncompressed) :

    dictionary = { bytes([i]) : i for i in range(256) }
    dict_size = 256
    w = bytes()
    compressed_data = []
    max_code_used = 255

    for c in uncompressed :
wc = w + bytes([c])
if wc in dictionary :
w = wc
else :
    compressed_data.append(dictionary[w])
    if len(dictionary) < 4096 :
        dictionary[wc] = dict_size
        dict_size += 1
        w = bytes([c])
        max_code_used = max(max_code_used, dict_size - 1)

        if w:
```

```
compressed_data.append(dictionary[w])

return compressed_data, max_code_used

def lzw_decompress(compressed) :

    dictionary = { i: bytes([i]) for i in range(256) }
    dict_size = 256
    result = []

    w = bytes([compressed.pop(0)])
    result.append(w)

    for k in compressed :
if k in dictionary :
entry = dictionary[k]
elif k == dict_size :
    entry = w + bytes([w[0]])
    result.append(entry)

    if len(dictionary) < 4096 :
        dictionary[dict_size] = w + bytes([entry[0]])
        dict_size += 1

        w = entry

        return b''.join(result)
```

Функція `lzw_compress` стискає дані. Вона створює словник з усіма можливими байтами, потім обробляє вхідні дані, додаючи нові комбінації байтів до словника і записуючи коди цих комбінацій у стиснені дані. Наприкінці повертає стиснені дані та максимальний код у словнику.

Функція `lzw_decompress` відновлює стиснені дані. Вона зчитує коди з стиснених даних, знаходить відповідні комбінації байтів у словнику і відновлює початкові дані. Наприкінці повертає відновлені дані у вигляді байтового рядка.

compress_image.py

Файл **compress_image.py** відповідає за стиснення зображень, обчислення їх ентропії та коефіцієнта стиснення:

```
from PIL import Image

import numpy as np
```

```
import os
import math
from lzw import lzw_compress

def calculate_entropy(img):
    unique, counts = np.unique(img, return_counts=True)
    counts = counts.astype(float)
    total_pixels = img.size
    probabilities = counts / total_pixels
    entropy = 0.0
    for p in probabilities:
        if p > 0:
            entropy -= p * np.log2(p)
    return entropy

def calculate_compression_ratio(encoded_img, height, width,
max_dict_size):
    block_size = 1
    padded_height = height
    if height % block_size != 0:
        padded_height = height + block_size - (height % block_size)

    padded_width = width
    if width % block_size != 0:
        padded_width = width + block_size - (width % block_size)

    bits_in_original_img = padded_height * padded_width * 8
    bits_per_symbol = math.ceil(math.log2(max_dict_size))
    bits_in_encoded_img = len(encoded_img) * bits_per_symbol

    compression_ratio = bits_in_original_img / bits_in_encoded_img
```

```
    return compression_ratio

def compress_image(input_file, output_file):
    img = Image.open(input_file).convert('L')
    width, height = img.size
    img_data = np.array(img)

    max_dict_size = 4096

    flattened_data = img_data.flatten().astype(np.uint8)
    compressed_data, max_code_used = lzw_compress(flattened_data)
    entropy = calculate_entropy(img_data)

    output_dir = os.path.dirname(output_file)
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    with open(output_file, 'wb') as f:
        f.write(width.to_bytes(4, byteorder='big'))
        f.write(height.to_bytes(4, byteorder='big'))
        for data in compressed_data:
            f.write(data.to_bytes(2, byteorder='big'))

    compression_ratio = calculate_compression_ratio(compressed_data,
height, width, max_dict_size)

    return len(compressed_data) * 2, entropy, max_code_used,
compression_ratio
```

Функція `calculate_entropy` обчислює ентропію зображення, що вимірює середню кількість інформації на піксель, використовуючи частоту появи кожного значення пікселя.

Функція `calculate_compression_ratio` обчислює коефіцієнт стиснення, порівнюючи розмір оригінального зображення в бітах з розміром стисненого зображення, використовуючи максимальний розмір словника кодування.

Функція `compress_image` читає зображення з файлу, перетворює його у відтінки сірого, стискає дані за допомогою алгоритму LZW, обчислює ентропію та коефіцієнт стиснення, і зберігає стиснені дані у новий файл. Вона також створює необхідні директорії, якщо вони не існують, і повертає розмір стиснених даних, ентропію, максимальний використовуваний код у словнику та коефіцієнт стиснення.

decompress_image.py

Файл **decompress_image.py** відповідає за декомпресію стиснених зображень.

```
from PIL import Image
import numpy as np
from lzw import lzw_decompress
import os


def decompress_image(input_file, output_file):
    with open(input_file, 'rb') as f:

        width = int.from_bytes(f.read(4), byteorder='big')
        height = int.from_bytes(f.read(4), byteorder='big')

        compressed_data = []
        while True:
            byte = f.read(2)
            if not byte:
                break
            compressed_data.append(int.from_bytes(byte,
byteorder='big'))
```

```
    decompressed_data = lzw_decompress(compressed_data)
    img_flat = np.array(list(decompressed_data), dtype=np.uint8)
    img_data = img_flat.reshape((height, width))
    img = Image.fromarray(img_data)

    output_dir = os.path.dirname(output_file)
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    img.save(output_file)
```

Функція `decompress_image` зчитує стиснені дані з файлу, отримує ширину та висоту зображення, а потім декомпресує дані за допомогою алгоритму LZW. Після цього вона відновлює початкове зображення, перетворюючи декомпресовані дані у двовимірний масив пікселів. Відновлене зображення зберігається у вказаний файл. Якщо необхідні директорії не існують, функція створює їх.

main.py

Файл **main.py** забезпечує основну взаємодію користувача з програмою через графічний інтерфейс і виконує основні операції зі стиснення та декомпресії зображень:

```
import os
import tkinter as tk
from tkinter import filedialog
import matplotlib.pyplot as plt
from PIL import Image
import compress_image
import decompress_image
import math

def create_folders():
    if not os.path.exists('compressed_images'):
```

```
        os.makedirs('compressed_images')
    if not os.path.exists('decompressed_images'):
        os.makedirs('decompressed_images')
    if not os.path.exists('grayscale_images'):
        os.makedirs('grayscale_images')


def select_image():
    root = tk.Tk()
    root.withdraw()
    image_path = filedialog.askopenfilename(filetypes=[("Image
files", "*.png;*.jpg;*.jpeg;" "*.tif")])
    return image_path


def show_images(original_path, decompressed_path, compressed_size,
original_size, entropy, max_code_used, compression_ratio):
    original_img = Image.open(original_path)
    decompressed_img = Image.open(decompressed_path)
    max_achievable_compression = math.log2(256) / entropy if entropy
!= 0 else 0


    fig, axs = plt.subplots(1, 2, figsize=(12, 6))


    axs[0].imshow(original_img, cmap='gray')
    axs[0].set_title(f'Original Image\nSize: {original_size /
1024:.2f} KB')
    axs[0].axis('off')


    axs[1].imshow(decompressed_img, cmap='gray')
    axs[1].set_title(f'Decompressed Image\nSize:
{os.path.getsize(decompressed_path) / 1024:.2f} KB')
    axs[1].axis('off')


    info_text = (
```

```
        f'Compression Ratio: {compression_ratio:.2f}, Entropy:
{entropy:.2f}\n'
        f'Max Achievable Compression:
{max_achievable_compression:.2f}, Maximum Dictionary Code Used:
{max_code_used}'
    )
    plt.figtext(0.5, 0.03, info_text, ha='center', fontsize=12)
    plt.subplots_adjust(bottom=0.2)
    plt.show()


def main():
    create_folders()


    original_image_path = select_image()
    if not original_image_path:
        print("No image selected!")
        return


    filename = os.path.basename(original_image_path).rsplit('.',
1)[0]
    grayscale_image_path = os.path.join('grayscale_images', filename
+ "_grayscale.png")
    compressed_image_path = os.path.join('compressed_images',
filename + ".lzw")
    decompressed_image_path = os.path.join('decompressed_images',
filename + "_decompressed.png")


    img = Image.open(original_image_path).convert('L')
    img.save(grayscale_image_path)


    original_size = os.path.getsize(grayscale_image_path)
    compressed_size, entropy, max_code_used, compression_ratio =
compress_image.compress_image(grayscale_image_path,
compressed_image_path)
```

```
    decompress_image.decompress_image(compressed_image_path,
decompressed_image_path)


    show_images(grayscale_image_path, decompressed_image_path,
compressed_size, original_size, entropy, max_code_used,
compression_ratio)
    print(f"Image compressed to {compressed_image_path}")
    print(f"Image decompressed to {decompressed_image_path}")


if __name__ == "__main__":
    main()
```

Функція `create_folders` створює необхідні директорії для збереження зображень, якщо вони ще не існують.

Функція `show_images` відображає оригінальне і декомпресоване зображення поруч, а також виводить інформацію про розмір зображень, ентропію, коефіцієнт стиснення і максимальний код у словнику.

Функція `main` виконує основні кроки:

1. Створює необхідні директорії.
2. Відкриває діалогове вікно для вибору зображення.
3. Зберігає оригінальне зображення у відтінках сірого.
4. Стискає зображення та зберігає стиснені дані у файл.
5. Декомпресує зображення та зберігає його.
6. Відображає оригінальне та декомпресоване зображення з додатковою інформацією.

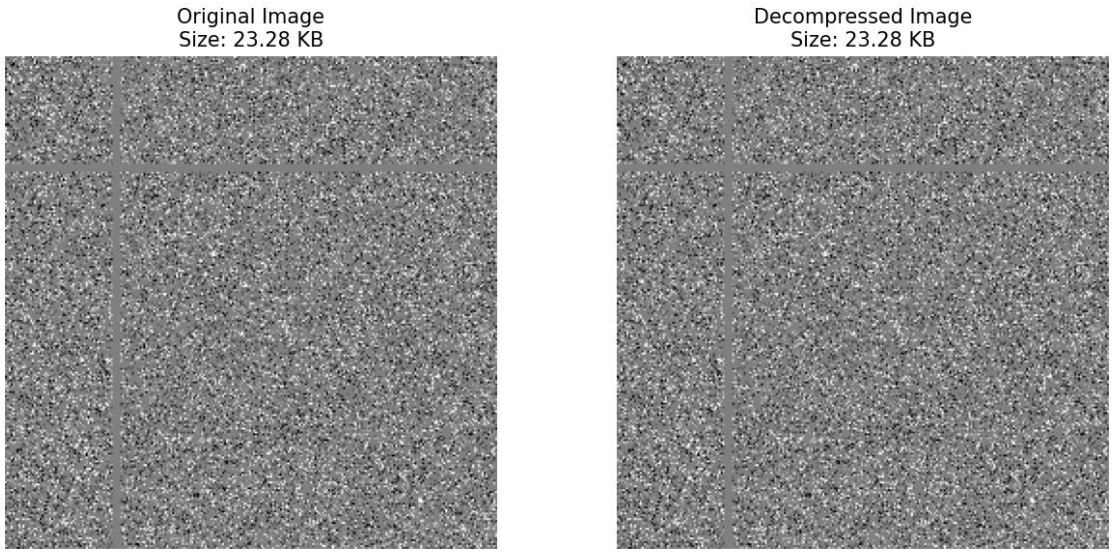
Результати роботи

Стиснення LZW в основному використовує два типи надмірності: просторову та статистичну.

Просторова надмірність означає, що в зображенні сусідні пікселі зазвичай мають схожі значення. LZW стиснення використовує цю надмірність, кодує повторювані шаблони пікселів як один код.

Статистична надмірність означає, що в будь-яких даних деякі символи або комбінації символів зустрічаються частіше за інші. LZW стиснення використовує цю надмірність, призначаючи коротші коди для частіше вживаних шаблонів, зменшуючи загальну кількість бітів, необхідних для представлення даних.

Для зображень з просторовою та статистичною надлишковістю алгоритм показав високі коефіцієнти стиснення:



Compression Ratio: 3.74, Entropy: 1.81
Max Achievable Compression: 4.41, Maximum Dictionary Code Used: 4095

Ось ще кілька вихідних файлів. З них видно, що стиснення LZW корисне лише для зображень з високою просторовою та статистичною надлишковістю.

Original Image
Size: 209.41 KB

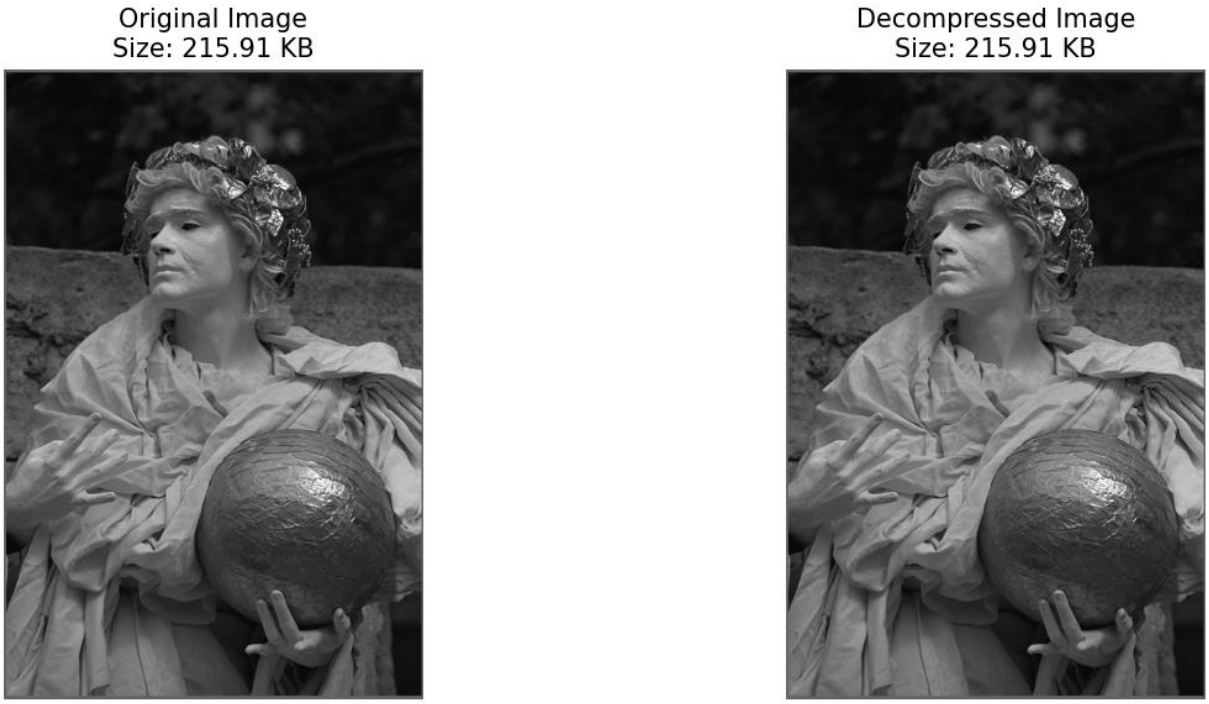
Decompressed Image
Size: 209.41 KB

Compression Ratio: 1.44, Entropy: 5.57
Max Achievable Compression: 1.44, Maximum Dictionary Code Used: 4095

Original Image
Size: 263.61 KB

Decompressed Image
Size: 263.61 KB

Compression Ratio: 1.03, Entropy: 7.62
Max Achievable Compression: 1.05, Maximum Dictionary Code Used: 4095



Compression Ratio: 1.08, Entropy: 7.26
Max Achievable Compression: 1.10, Maximum Dictionary Code Used: 4095

Висновок

У цьому проєкті реалізовано алгоритм LZW для стиснення зображень у відтінках сірого. Створені кодер та декодер забезпечують ефективне стиснення без втрати якості. Алгоритм LZW використовує схожість сусідніх пікселів і частоту появи певних шаблонів, що дозволяє зменшити розмір зображень. Для зображень з високою надмірністю алгоритм показав хороші результати стиснення. Однак для деяких завдань був недостатньо ефективним, особливо якщо просторової або статистичної надмірності в зображенні багато.