

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра ПЗ

Лабораторна робота №2
з дисципліни «Основи теорії інформації та кодування»

Виконав: ст. 2ПІ-22Б

Чорний О. В.

Перевірів: доцент

Майданюк В. П.

ЛАБОРАТОРНА РОБОТА №2

Мета: отримати практичні навички застосування словникових методів

Завдання: 1. Ущільнити файл з використанням 9-бітної LZW.

Код програми:

```
import os
from collections import Counter
import math
import struct

def calculate_entropy(data):
    frequency = Counter(data)
    total_symbols = len(data)
    entropy = -sum((freq / total_symbols) * math.log2(freq / total_symbols) for freq in
frequency.values())
    return entropy

def lzw_compress(data):
    max_table_size = 2 ** 9 #
    dictionary = {chr(i): i for i in range(256)} # Початкова таблиця символів
    next_code = 256

    current = ""
    compressed_data = []

    for symbol in data:
        current_plus_symbol = current + symbol
        if current_plus_symbol in dictionary:
            current = current_plus_symbol
        else:
            compressed_data.append(dictionary[current])
            if len(dictionary) < max_table_size:
                dictionary[current_plus_symbol] = next_code
                next_code += 1
```

```
current = symbol
```

```
if current:
```

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

```
return compressed_data
```

```
def lzw_decompress(compressed_data):
```

```
    max_table_size = 2 ** 9 # 9-бітне кодування
```

```
    dictionary = {i: chr(i) for i in range(256)} # Початкова таблиця
```

```
    next_code = 256
```

```
    current = chr(compressed_data[0])
```

```
    decompressed_data = [current]
```

```
    for code in compressed_data[1:]:
```

```
        if code in dictionary:
```

```
            entry = dictionary[code]
```

```
        elif code == next_code:
```

```
            entry = current + current[0]
```

```
        else:
```

```
            raise ValueError("Некоректні стиснені дані")
```

```
    decompressed_data.append(entry)
```

```
    if len(dictionary) < max_table_size:
```

```
        dictionary[next_code] = current + entry[0]
```

```
        next_code += 1
```

```
    current = entry
```

```
    return "".join(decompressed_data)
```

```
def process_file(file_path):
```

```
    with open(file_path, 'r', encoding='utf-8', errors='ignore') as file:
```

```
        data = file.read()
```

```
    entropy = calculate_entropy(data)
```

```
    compressed_data = lzw_compress(data)
```

```

original_size = len(data) * 8
compressed_size = len(compressed_data) * 9 # 9 біт на кожен код
avg_bits_per_symbol = compressed_size / len(data)

compressed_file_path = file_path + ".lzw"
with open(compressed_file_path, 'wb') as file:
    for code in compressed_data:
        file.write(code.to_bytes(2, byteorder='big'))

print(f"Файл: {file_path}")
print(f"Ентропія вихідного файлу: {entropy:.4f} біт/символ")
print(f"Оригінальний розмір файлу: {original_size} біт")
print(f"Стиснений розмір файлу: {compressed_size} біт")
print(f"Середня кількість біт на символ після ущільнення: {avg_bits_per_symbol:.4f} біт/символ")
print(f"Стиснений файл збережено як: {compressed_file_path}")

def decompress_file(compressed_file_path, output_file_path):
    with open(compressed_file_path, 'rb') as file:
        compressed_data = []
        while byte := file.read(2):
            compressed_data.append(int.from_bytes(byte, byteorder='big'))

    decompressed_data = lzw_decompress(compressed_data)

    with open(output_file_path, 'w', encoding='utf-8') as file:
        file.write(decompressed_data)

    print(f"Розпакований файл збережено як: {output_file_path}")

file_path = "example.txt"
process_file(file_path)

decompressed_file_path = "example_decompressed.txt"
decompress_file(file_path + ".lzw", decompressed_file_path)

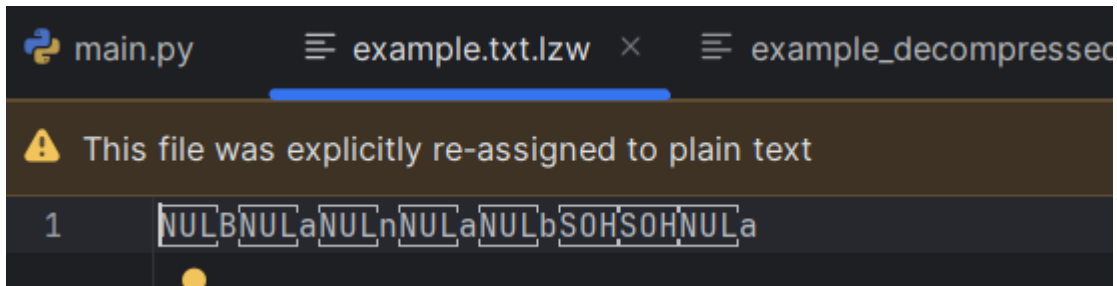
```

Результат роботи програми:

Вхідні дані: Banabana

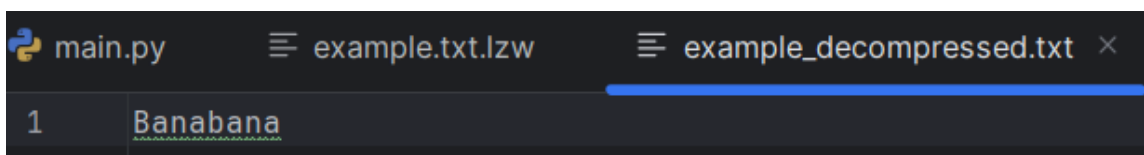
Вихідні дані:

- **Стиснений файл:**



```
main.py example.txt.lzw example_decompressed.txt
⚠ This file was explicitly re-assigned to plain text
1 NULBNULaNULnNULaNULbSOH$OHNULa
```

- **Видобутий файл:**



```
main.py example.txt.lzw example_decompressed.txt
1 Banabana
```

Контрольні питання:

1. Охарактеризуйте словникові методи ущільнення даних.

Словникові методи ущільнення даних базуються на заміні повторюваних послідовностей символів на коротші представлення — коди або індекси. Вони будують таблицю (словник) з унікальних фрагментів вхідних даних і замість кожного фрагмента записують його індекс у словнику. Словник може бути статичним (заздалегідь відомим) або динамічним (побудованим під час кодування). Ці методи ефективні для даних з повторюваними шаблонами, наприклад, у текстах або зображеннях з однаковими пікселями.

2. Наведіть алгоритм кодування методом LZW.

Алгоритм LZW (Lempel–Ziv–Welch) починає з ініціалізації словника всіма можливими окремими символами алфавіту. Далі зчитується поточний символ і будується рядок, який розширюється посимвольно доти, доки він міститься у словнику. Коли новий рядок більше не знаходиться у словнику, до вихідного потоку додається код попереднього рядка, а новий рядок записується до словника з новим кодом. Процес повторюється до кінця вхідних даних.

3. Наведіть алгоритм декодування методом LZW.

Декодування LZW починається з тим самим словником, що й при кодуванні. Зчитується перший код, і відповідний йому символ виводиться. Потім зчитується наступний код, і до словника додається новий рядок, що складається з попереднього рядка та першого символу поточного. Якщо коду ще немає в словнику (особливий випадок), його значення обчислюється на основі попереднього рядка. Розшифровані символи виводяться, а словник постійно оновлюється в процесі декодування.

4. Які недоліки алгоритму LZW?

Серед основних недоліків LZW — поступове зростання розміру словника, що може призвести до надмірного використання пам'яті. Якщо дані не мають достатньої повторюваності, алгоритм може навіть збільшити розмір файлу. Також ефективність LZW знижується при роботі з шумовими або вже стисненими файлами. У деяких реалізаціях потрібне обмеження на розмір словника, інакше можливе зниження продуктивності або переповнення пам'яті.

5. Чи забезпечують алгоритми ущільнення без втрат високий коефіцієнт ущільнення зображень?

Алгоритми ущільнення без втрат зазвичай не забезпечують високого коефіцієнта стиснення для зображень, особливо фотографій або складних сцен, де пікселі змінюються часто. Вони ефективні лише для зображень із великою кількістю повторюваних пікселів, наприклад, графіки або скріншоти з великими однотонними ділянками. Для фотографій частіше застосовують методи зі втратами (наприклад, JPEG), які дозволяють досягти значно вищого ступеня стиснення за рахунок допустимого зниження якості.