Лабораторна робота 3. Реалізація коду Хаффмана

Мета роботи

- **Ознайомитися з принципами побудови оптимального префіксного кодування.**
- **Реалізувати алгоритм кодування Хаффмана.**
- **Провести аналіз ефективності кодування на різних вхідних даних.**

Код Хаффмана — оптимальний префіксний код, який мінімізує середню довжину кодового слова при заданому розподілі ймовірностей символів.

Алгоритм Хаффмана:

1. Побудова таблиці частот появи символів.

2. Формування бінарного дерева на основі мінімальних частот.

3. Генерація кодових слів при обході дерева.

Методи LZ77 та LZMA базуються на пошуку повторень у тексті та заміні їх короткими посиланнями.

Код:

```
import heapq
import os
import zlib
import lzma
from collections import Counter, defaultdict
import matplotlib.pyplot as plt

# Крок 1. Хаффманівське дерево
```

```
class Node:
        def __init__(self, char=None, freq=0):
        self.char = char
        self.freq = freq
        self.left = None
        self.right = None

        def __lt__(self, other):  # Для heapq
        return self.freq < other.freq

def build_huffman_tree(freq_table):
        heap = [Node(char, freq) for char, freq in freq_table.items()]
        heapq.heapify(heap)

        while len(heap) > 1:
        n1 = heapq.heappop(heap)
        n2 = heapq.heappop(heap)
        merged = Node(freq=n1.freq + n2.freq)
        merged.left = n1
        merged.right = n2
        heapq.heappush(heap, merged)

        return heap[0]

def generate_codes(node, prefix="", codebook=None):
        if codebook is None:
        codebook = dict()
        if node:
        if node.char is not None:
        codebook[node.char] = prefix
        generate_codes(node.left, prefix + "0", codebook)
        generate_codes(node.right, prefix + "1", codebook)
        return codebook

def huffman_encode(text, codebook):
        return ''.join(codebook[char] for char in text)
```

```
def decode_huffman(encoded, root):
        decoded = []
        node = root
        for bit in encoded:
        node = node.left if bit == '0' else node.right
        if node.char:
        decoded.append(node.char)
        node = root
        return ''.join(decoded)

# Крок 2. Стиснення іншими методами
def compress_zlib(text):
        return zlib.compress(text.encode())

def compress_lzma(text):
        return lzma.compress(text.encode())

# Крок 3. Основна логіка
def main():
        with open("input.txt", "r", encoding="utf-8") as f:
        text = f.read()

        freq = Counter(text)
        huff_root = build_huffman_tree(freq)
        codes = generate_codes(huff_root)
        encoded = huffman_encode(text, codes)

        # Статистика
        original_size = len(text.encode('utf-8')) * 8
        huff_size = len(encoded)
        zlib_size = len(compress_zlib(text)) * 8
        lzma_size = len(compress_lzma(text)) * 8

        # Таблиця кодів
        sorted_codes = sorted(codes.items(), key=lambda x: -freq[x[0]])
        code_table = "\n".join([f"{char!r}: {codes[char]} (частота:
{freq[char]})" for char, _ in sorted_codes])
```

```
    # Графік
    lengths = [len(codes[char]) for char in freq]
    f_values = [freq[char] for char in freq]
    plt.figure(figsize=(10, 6))
    plt.scatter(f_values, lengths, alpha=0.7)
    plt.title("Графік: Частота vs Довжина коду")
    plt.xlabel("Частота символу")
    plt.ylabel("Довжина Хаффман-коду")
    plt.grid(True)
    plt.savefig("code_lengths_distribution.png")
    plt.close()

    # Запис звіту
    with open("report.txt", "w", encoding="utf-8") as f:
    f.write("Лабораторна №3 — Порівняння методів стиснення\n")
    f.write("="*50 + "\n\n")
    f.write("1. Код Хаффмана\n")
    f.write(f"  Розмір: {huff_size} біт\n")
    f.write("2. Zlib (LZ77)\n")
    f.write(f"  Розмір: {zlib_size} біт\n")
    f.write("3. LZMA (LZW)\n")
    f.write(f"  Розмір: {lzma_size} біт\n")
    f.write("4. Початковий розмір: " + str(original_size) + " біт\n\n")
    f.write("Коефіцієнти стиснення:\n")
    f.write(f"  Хаффман: {original_size/huff_size:.2f}x\n")
    f.write(f"  Zlib: {original_size/zlib_size:.2f}x\n")
    f.write(f"  LZMA: {original_size/lzma_size:.2f}x\n\n")
    f.write("="*50 + "\n\n")
    f.write("Таблиця кодів Хаффмана:\n\n")
    f.write(code_table)

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

Дані для input.txt взяті з
https://uk.wikipedia.org/wiki/%D0%9C%D1%96%D1%81%D1%8F%D1%86

%D1%8C_(%D1%81%D1%83%D0%BF%D1%83%D1%82%D0%BD%D0%B8%D0%BA)

Вихідні дані:

==========

1. Код Хаффмана
  Розмір: 485205 біт
2. Zlib (LZ77)
  Розмір: 368840 біт
3. LZMA (LZW)
  Розмір: 308096 біт
4. Початковий розмір: 1330296 біт

Коефіцієнти стиснення:
Хаффман: 2.74x
Zlib: 3.61x
LZMA: 4.32x

==========

Таблиця кодів Хаффмана:

' ': 100 (частота: 12369)
'о': 1011 (частота: 6442)
'н': 0110 (частота: 5322)
'а': 0101 (частота: 5283)
'і': 0011 (частота: 5122)
'и': 11111 (частота: 3957)
'р': 11011 (частота: 3436)
'т': 11010 (частота: 3359)
'е': 11001 (частота: 3329)
'с': 11000 (частота: 3274)
'в': 10101 (частота: 3229)
'м': 01110 (частота: 2620)
'я': 01001 (частота: 2613)
'л': 01000 (частота: 2591)
'к': 00011 (частота: 2255)
'д': 00000 (частота: 2017)
'п': 111100 (частота: 1927)
'у': 111001 (частота: 1773)
'з': 011111 (частота: 1465)
'ь': 001010 (частота: 1212)
',': 000101 (частота: 1127)
'ц': 000010 (частота: 1026)
'г': 1111011 (частота: 995)
'б': 1111010 (частота: 987)
'ч': 1110110 (частота: 953)
'х': 1010011 (частота: 807)
'й': 1010010 (частота: 805)
'.': 0111101 (частота: 731)

'ю': 0111100 (частота: 690)
'М': 0010011 (частота: 609)
']': 0010000 (частота: 577)
'[': 0010001 (частота: 577)
'ж': 0001000 (частота: 557)
'1': 0000110 (частота: 512)
'ш': 11101011 (частота: 467)
'\n': 11101010 (частота: 459)
'2': 11101001 (частота: 454)
'є': 11101000 (частота: 446)
'ї': 11100000 (частота: 403)
'0': 10100011 (частота: 390)
'3': 00101101 (частота: 324)
'4': 00001111 (частота: 259)
'9': 111011111 (частота: 249)
'щ': 111011101 (частота: 238)
'»': 111000101 (частота: 222)
'«': 111000110 (частота: 222)
'З': 111000011 (частота: 205)
'ф': 101000101 (частота: 195)
'6': 101000100 (частота: 194)
'5': 101000011 (частота: 193)
'8': 101000001 (частота: 181)
'7': 001011110 (частота: 173)
'С': 001011111 (частота: 173)
'-': 001001001 (частота: 151)
')': 001001010 (частота: 151)
'(': 001001011 (частота: 151)
'е': 000100111 (частота: 147)
'А': 000100110 (частота: 146)
'п': 000011101 (частота: 132)
'П': 1110111100 (частота: 120)
'о': 1110001111 (частота: 112)
'а': 1110001001 (частота: 108)
'—': 1110000101 (частота: 104)
'К': 1010000100 (частота: 91)
'В': 1010000000 (частота: 89)

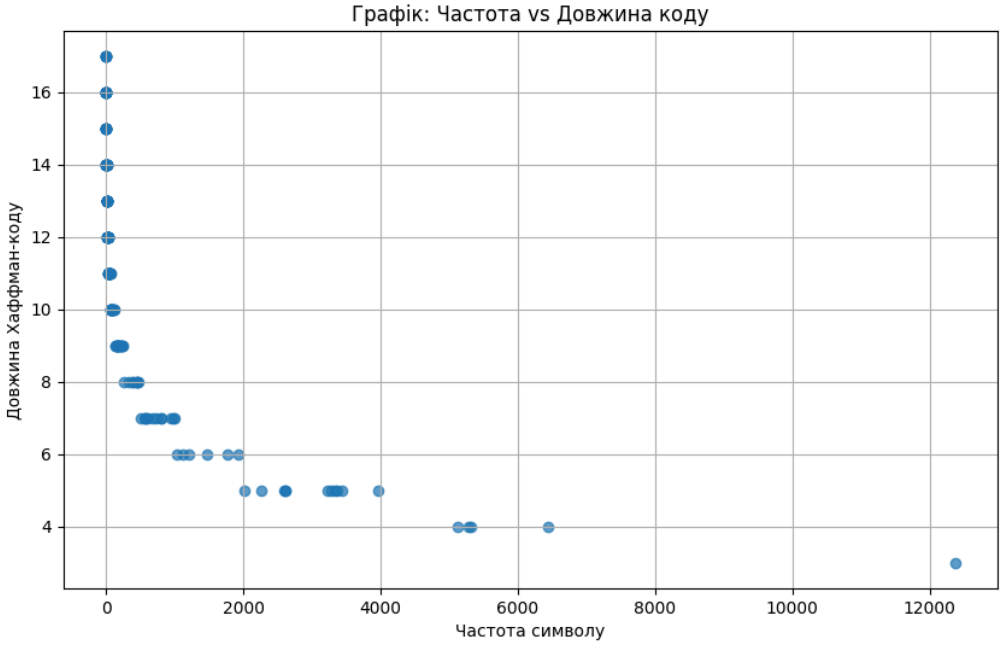
'r': 0010111001 (частота: 83)
'i': 0010110011 (частота: 80)
'Д': 0010110000 (частота: 76)
':': 0010010000 (частота: 74)
's': 0010010001 (частота: 74)
"'": 0001001011 (частота: 72)
'u': 0001001000 (частота: 65)
'Т': 0000111000 (частота: 61)
'У': 11101111010 (частота: 60)
'l': 11101110011 (частота: 59)
'О': 11101110001 (частота: 57)
'Н': 11100011101 (частота: 56)
'c': 11100010001 (частота: 54)
't': 10100000010 (частота: 45)
'М': 00101110110 (частота: 44)
'I': 00101110111 (частота: 44)
'Р': 00101110100 (частота: 42)
'Л': 00101110101 (частота: 42)
'\ufeff': 00101100101 (частота: 40)
'Г': 00101100011 (частота: 38)
'L': 00101100100 (частота: 38)
'Ц': 00010010101 (частота: 36)
'Ш': 00010010010 (частота: 34)
'Б': 00010010011 (частота: 34)
'S': 111011110110 (частота: 30)
'I': 111011100001 (частота: 29)
'С': 111011100100 (частота: 29)
'Ч': 111011100000 (частота: 28)
'h': 111000111000 (частота: 27)
'p': 111000010011 (частота: 26)
'd': 111000100000 (частота: 26)
'm': 111000100001 (частота: 26)
';': 101000000111 (частота: 23)
'R': 101000010100 (частота: 23)
'Е': 101000010101 (частота: 23)
'%': 001011100000 (частота: 20)
'g': 001011000100 (частота: 19)

'X': 001011000101 (частота: 19)
'O': 000100101001 (частота: 18)
'Я': 000100101000 (частота: 17)
'P': 000011100100 (частота: 16)
'y': 1110111001011 (частота: 15)
'T': 1110001110010 (частота: 14)
'k': 1110001110011 (частота: 14)
'г': 1110111001010 (частота: 14)
'A': 1110000100100 (частота: 13)
'X': 1110000100101 (частота: 13)
'V': 1010000101100 (частота: 12)
'f': 1010000101101 (частота: 12)
'°': 1010000101110 (частота: 12)
'v': 1110000100000 (частота: 12)
'N': 1110000100001 (частота: 12)
'D': 0010111000110 (частота: 11)
'Є': 0010111000111 (частота: 11)
'Ф': 1010000001100 (частота: 11)
'b': 0010111000100 (частота: 10)
'B': 0010111000101 (частота: 10)
'/': 0000111001011 (частота: 8)
'E': 11101111011101 (частота: 8)
'*': 11100001000111 (частота: 7)
'G': 11101111011100 (частота: 7)
'Й': 10100000011011 (частота: 6)
'³': 10100001011110 (частота: 6)
'F': 11100001000101 (частота: 6)
'H': 11100001000110 (частота: 6)
'Ю': 00101110000100 (частота: 5)
'ē': 00101110000101 (частота: 5)
'w': 00101110000111 (частота: 5)
'U': 10100000011010 (частота: 5)
'’': 00001110010101 (частота: 4)
'Щ': 00001110011001 (частота: 4)
'W': 00001110011010 (частота: 4)
'Ж': 00001110011100 (частота: 4)
'–': 00001110011111 (частота: 4)

ʹ': 0010111000011101 (частота: 3)
'…': 1010000101111110 (частота: 3)
'!': 1110000100010000 (частота: 3)
'K': 1110000100010001 (частота: 3)
'×': 0000111001010000 (частота: 2)
'–': 0000111001010001 (частота: 2)
'"': 0000111001100000 (частота: 2)
'\t': 0000111001100001 (частота: 2)
'Q': 0010111000011000 (частота: 2)
'ḫ': 1110111101111000 (частота: 2)
'ā': 1110111101111001 (частота: 2)
'ν': 1110111101111011 (частота: 2)
'ή': 1110111101111100 (частота: 2)
'μ': 1110111101111101 (частота: 2)
'j': 1110111101111111 (частота: 2)
'γ': 0000111001101100 (частота: 1)
'ę': 0000111001101101 (частота: 1)
'±': 0000111001101110 (частота: 1)
'Ï': 0000111001101111 (частота: 1)
'λ': 0000111001110100 (частота: 1)
'ε': 0000111001110101 (частота: 1)
'ě': 0000111001110110 (частота: 1)
'·': 0000111001110111 (частота: 1)
'Σ': 0000111001111000 (частота: 1)
'ê': 0000111001111001 (частота: 1)
'ą': 0000111001111010 (частота: 1)
'Y': 0000111001111011 (частота: 1)
'x': 1010000101111110 (частота: 1)
'ä': 10100001011111110 (частота: 1)
'η': 10100001011111111 (частота: 1)
'Z': 11101111011110100 (частота: 1)
'J': 11101111011110101 (частота: 1)
'q': 11101111011111100 (частота: 1)
'&': 11101111011111101 (частота: 1)

Графік:

На графіку видно, що символи з більшою частотою мають коротші коди, що відповідає принципу мінімізації середньої довжини.



Висновки:

Код Хаффмана показав добрі результати стиснення для текстів зі стабільним розподілом частот.

Методи Zlib та LZMA виявилися ефективнішими для великих текстів із багатьма повтореннями завдяки використанню фразового словника.

Найвищий коефіцієнт стиснення продемонстрував метод **LZMA**.

Візуалізація підтвердила властивість алгоритму Хаффмана: частіші символи отримують коротші коди.

Усі методи забезпечили повністю безвтратне декодування.

Контрольні запитання та відповіді

1. Які переваги та недоліки коду Хаффмана?

Переваги:

- **Оптимальність**: забезпечує найменшу середню довжину коду для заданого розподілу ймовірностей.
- **Простота реалізації**: алгоритм досить простий у реалізації.
- **Без втрат**: кодування не втрачає жодної інформації.
- **Префіксність**: жоден код не є префіксом іншого, що забезпечує однозначне декодування.

Недоліки:

- **Неадаптивність (у класичній версії)**: для нових символів необхідно перебудовувати дерево.
- **Неефективність при рівномірному розподілі**: якщо всі символи мають однакову частоту, ефект стискання мінімальний.
- **Потребує повного аналізу частот** перед кодуванням, що може бути проблематичним для потокових або дуже великих даних.
- **Низька ефективність при дуже коротких текстах**, де побудова дерева не дає значного виграшу.

2. Як змінюється ефективність кодування при різних вхідних текстах?

Ефективність кодування напряму залежить від **частоти символів у тексті**:

- Якщо **одні символи зустрічаються значно частіше**, ніж інші — ефективність зростає.

- Якщо **розподіл рівномірний**, Хаффман-код майже не стискає текст.

- **Довші тексти** з багатим частотним розподілом забезпечують краще стиснення.

- Якщо текст містить **багато рідковживаних символів або випадкових послідовностей**, ефективність зменшується.

3. Чому код Хаффмана є оптимальним у середньому?

Код Хаффмана будується на основі **ймовірностей появи символів** і гарантує мінімальну **середню довжину кодування** для кожного символу при відомому розподілі:

- Це досягається шляхом об'єднання найрідших символів у піддерева з довшими кодами, а частіших — у вузли з коротшими кодами.

- За теоремою, **ніякий інший префіксний код** не може дати меншої середньої довжини.

4. Як забезпечується декодування без втрат?

- Код Хаффмана — це **префіксний код**, тобто жоден код одного символу не є початком коду іншого.

- Завдяки цьому, **декодер не потребує роздільників** — він просто читає біт за бітом, доки не знайде відповідний код у дереві.

- **Повне відновлення тексту** гарантується, якщо маємо доступ до дерева або таблиці відповідностей.