#Лабораторна робота №2#

алгоритми кодування Виконували: Ковальчук Катерина, Гуменюк Денис

Алгоритм Гаффмана

Виконала Ковальчук Катерина

*у мене для зчитки файлу треба задавати повний шлях до файла, тому при перевірці це треба фрахувати і змінити за потреби

```
file_path0 = "huffman_test_files/Haffman.txt"
file_path1 = "huffman_test_files/ChornaRada.txt"
file_path2 = "huffman_test_files/PerehresniStezky.txt"
file_path3 = "huffman_test_files/test.txt"
file_path4 = "huffman_test_files/text.txt"
lst_files = [file_path0, file_path3, file_path4, file_path1,
file_path2]
```

Для реалізації алгоритму Гаффмана нам потрібно зчитати дані з файлу, що робить функція read_file. Оскільки наприкінці лабораторної нам треба буде порівняти декодоване і оригінальне повідомлення, то я оригінал зберігаю у змінну file_text, а у data записую текст посимвольно, щоб в подальшому мати змогу порахувати ймовірність кожного символу в тексті.

```
def read(path):
    file_text = ""
    data = []
    with open(path, "r", encoding="utf-8") as file:
        for line in file:
            file_text += line
            for character in line:
                data.append(character)
    return file_text, data

def read_file():
    """
    Reads data from file
    """
    file_text = ""
    data = []
    with open(file_path0, "r", encoding="utf-8") as file:
        for line in file:
            file_text += line
            for character in line:
                data.append(character)
    return file_text, data

file_text, the_data = read_file()
print(file_text,"\n", the_data)
```

Натуральну ваніль часто використовують у випічці та десертах. Дізнайтеся, як вибрати найкращі стручки, як їх зберігати та використовувати.

Що таке ваніль?

Ваніль має неповторний м’який, солодкий смак та аромат. Вона належить до сімейства орхідейних. Стручок ванілі — це плід квітки, що утворився після завершення цвітіння. Ваніль з Мадагаскару становить 75 відсотків від усієї ванілі на ринку. Решта — це ваніль з Таїті та Мексики, але її значно важче знайти.

Довгі, чорні, тонкі та зморшкуваті стручки ванілі містять тисячі дрібних чорних насінин, які використовуються для ароматизації переважно солодких страв і особливо добре поєднуються з шоколадом. Наявність крихітних чорних цяточок у страві є підтвердженням того, що була використана справжня ваніль.

Як вибрати найкращу ваніль

Шукайте ароматні, темно-коричневі, майже чорні стручки, злегка зморшкуваті, але пружні, з трохи маслянистою блискучою поверхнею. Довжина є показником якості — найкраще вибирати стручки 15-20 сантиметрів завдовжки.

Як підготувати ваніль до використання

Розріжте стручок уздовж, а потім вишкребіть дрібне липке насіння за допомогою кінчика маленького гострого ножа.

Як зберігати ваніль

У герметичному контейнері в прохолодному темному місці — там стручки можуть зберігатися до двох років.

Як готувати з ваніллю

Додавайте насіння безпосередньо до страв, щоб ароматизувати їх, або опустіть стручки в кипляче молоко, щоб наповнити його ванільним ароматом, а потім використовуйте для приготування пудингів на молочній основі.

Дайте стручку висохнути пару днів, а потім помістіть його в банку з цукром. Приблизно за тиждень ароматизований цукор можна використовувати для випічки.


```
 ['Н', 'а', 'т', 'у', 'р', 'а', 'л', 'ь', 'н', 'у', ' ', 'в', 'а',
'н', 'і', 'л', 'ь', ' ', 'ч', 'а', 'с', 'т', 'о', ' ', 'в', 'и', 'к',
'о', 'р', 'и', 'с', 'т', 'о', 'в', 'у', 'ю', 'т', 'ь', ' ', 'у', ' ',
'в', 'и', 'п', 'і', 'ч', 'ц', 'і', ' ', 'т', 'а', ' ', 'д', 'е', 'с',
'е', 'р', 'т', 'а', 'х', '.', ' ', 'Д', 'і', 'з', 'н', 'а', 'й', 'т',
'е', 'с', 'я', ',', ' ', 'я', 'к', ' ', 'в', 'и', 'б', 'р', 'а', 'т',
'и', ' ', 'н', 'а', 'й', 'к', 'р', 'а', 'щ', 'і', ' ', 'с', 'т', 'р',
'у', 'ч', 'к', 'и', ',', ' ', 'я', 'к', ' ', 'ї', 'х', ' ', 'з', 'б',
'е', 'р', 'і', 'г', 'а', 'т', 'и', ' ', 'т', 'а', ' ', 'в', 'и', 'к',
'о', 'р', 'и', 'с', 'т', 'о', 'в', 'у', 'в', 'а', 'т', 'и', '.', '\n',
'Щ', 'о', ' ', 'т', 'а', 'к', 'е', ' ', 'в', 'а', 'н', 'і', 'л', 'ь',
'?', '\n', 'В', 'а', 'н', 'і', 'л', 'ь', ' ', 'м', 'а', 'є', ' ', 'н',
'е', 'п', 'о', 'в', 'т', 'о', 'р', 'н', 'и', 'й', ' ', 'м', '’', 'я',
'к', 'и', 'й', ',', ' ', 'с', 'о', 'л', 'о', 'д', 'к', 'и', 'й', ' ',
'с', 'м', 'а', 'к', ' ', 'т', 'а', ' ', 'а', 'р', 'о', 'м', 'а', 'т',
'.', ' ', 'В', 'о', 'н', 'а', ' ', 'н', 'а', 'л', 'е', 'ж', 'и', 'т',
'ь', ' ', 'д', 'о', ' ', 'с', 'і', 'м', 'е', 'й', 'с', 'т', 'в', 'а',
```

' ', 'о', 'р', 'х', 'і', 'д', 'е', 'й', 'н', 'и', 'х', '.', ' ', 'С',
'т', 'р', 'у', 'ч', 'о', 'к', ' ', 'в', 'а', 'н', 'і', 'л', 'і', ' ',
'—', ' ', 'ц', 'е', ' ', 'п', 'л', 'і', 'д', ' ', 'к', 'в', 'і', 'т',
'к', 'и', ',', ' ', 'щ', 'о', ' ', 'у', 'т', 'в', 'о', 'р', 'и', 'в',
'с', 'я', ' ', 'п', 'і', 'с', 'л', 'я', ' ', 'з', 'а', 'в', 'е', 'р',
'ш', 'е', 'н', 'н', 'я', ' ', 'ц', 'в', 'і', 'т', 'і', 'н', 'н', 'я',
'.', ' ', 'В', 'а', 'н', 'і', 'л', 'ь', ' ', 'з', ' ', 'М', 'а', 'д',
'а', 'г', 'а', 'с', 'к', 'а', 'р', 'у', ' ', 'с', 'т', 'а', 'н', 'о',
'в', 'и', 'т', 'ь', ' ', '7', '5', ' ', 'в', 'і', 'д', 'с', 'о', 'т',
'к', 'і', 'в', ' ', 'в', 'і', 'д', ' ', 'у', 'с', 'і', 'є', 'ї', ' ',
'в', 'а', 'н', 'і', 'л', 'і', ' ', 'н', 'а', ' ', 'р', 'и', 'н', 'к',
'у', '.', ' ', 'Р', 'е', 'ш', 'т', 'а', ' ', '—', ' ', 'ц', 'е', ' ',
'в', 'а', 'н', 'і', 'л', 'ь', ' ', 'з', ' ', 'Т', 'а', 'ї', 'т', 'і',
' ', 'т', 'а', ' ', 'М', 'е', 'к', 'с', 'и', 'к', 'и', ',', ' ', 'а',
'л', 'е', ' ', 'ї', 'ї', ' ', 'з', 'н', 'а', 'ч', 'н', 'о', ' ', 'в',
'а', 'ж', 'ч', 'е', ' ', 'з', 'н', 'а', 'й', 'т', 'и', '.', ' ', '\n',
'Д', 'о', 'в', 'г', 'і', ',', ' ', 'ч', 'о', 'р', 'н', 'і', ',', ' ',
'т', 'о', 'н', 'к', 'і', ' ', 'т', 'а', ' ', 'з', 'м', 'о', 'р', 'ш',
'к', 'у', 'в', 'а', 'т', 'і', ' ', 'с', 'т', 'р', 'у', 'ч', 'к', 'и',
' ', 'в', 'а', 'н', 'і', 'л', 'і', ' ', 'м', 'і', 'с', 'т', 'я', 'т',
'ь', ' ', 'т', 'и', 'с', 'я', 'ч', 'і', ' ', 'д', 'р', 'і', 'б', 'н',
'и', 'х', ' ', 'ч', 'о', 'р', 'н', 'и', 'х', ' ', 'н', 'а', 'с', 'і',
'н', 'и', 'н', ',', ' ', 'я', 'к', 'і', ' ', 'в', 'и', 'к', 'о', 'р',
'и', 'с', 'т', 'о', 'в', 'у', 'ю', 'т', 'ь', 'с', 'я', ' ', 'д', 'л',
'я', ' ', 'а', 'р', 'о', 'м', 'а', 'т', 'и', 'з', 'а', 'ц', 'і', 'ї',
' ', 'п', 'е', 'р', 'е', 'в', 'а', 'ж', 'н', 'о', ' ', 'с', 'о', 'л',
'о', 'д', 'к', 'и', 'х', ' ', 'с', 'т', 'р', 'а', 'в', ' ', 'і', ' ',
'о', 'с', 'о', 'б', 'л', 'и', 'в', 'о', ' ', 'д', 'о', 'б', 'р', 'е',
' ', 'п', 'о', 'є', 'д', 'н', 'у', 'ю', 'т', 'ь', 'с', 'я', ' ', 'з',
' ', 'ш', 'о', 'к', 'о', 'л', 'а', 'д', 'о', 'м', '.', '\n', 'Н', 'а',
'я', 'в', 'н', 'і', 'с', 'т', 'ь', ' ', 'к', 'р', 'и', 'х', 'і', 'т',
'н', 'и', 'х', ' ', 'ч', 'о', 'р', 'н', 'и', 'х', ' ', 'ц', 'я', 'т',
'о', 'ч', 'о', 'к', ' ', 'у', ' ', 'с', 'т', 'р', 'а', 'в', 'і', ' ',
'є', ' ', 'п', 'і', 'д', 'т', 'в', 'е', 'р', 'д', 'ж', 'е', 'н', 'н',
'я', 'м', ' ', 'т', 'о', 'г', 'о', ',', ' ', 'щ', 'о', ' ', 'б', 'у',
'л', 'а', ' ', 'в', 'и', 'к', 'о', 'р', 'и', 'с', 'т', 'а', 'н', 'а',
' ', 'с', 'п', 'р', 'а', 'в', 'ж', 'н', 'я', ' ', 'в', 'а', 'н', 'і',
'л', 'ь', '.', '\n', 'Я', 'к', ' ', 'в', 'и', 'б', 'р', 'а', 'т', 'и',
' ', 'н', 'а', 'й', 'к', 'р', 'а', 'щ', 'у', ' ', 'в', 'а', 'н', 'і',
'л', 'ь', '\n', 'Ш', 'у', 'к', 'а', 'й', 'т', 'е', ' ', 'а', 'р', 'о',
'м', 'а', 'т', 'н', 'і', ',', ' ', 'т', 'е', 'м', 'н', 'о', '-', 'к',
'о', 'р', 'и', 'ч', 'н', 'е', 'в', 'і', ',', ' ', 'м', 'а', 'й', 'ж',
'е', ' ', 'ч', 'о', 'р', 'н', 'і', ' ', 'с', 'т', 'р', 'у', 'ч', 'к',
'и', ',', ' ', 'з', 'л', 'е', 'г', 'к', 'а', ' ', 'з', 'м', 'о', 'р',
'ш', 'к', 'у', 'в', 'а', 'т', 'і', ',', ' ', 'а', 'л', 'е', ' ', 'п',
'р', 'у', 'ж', 'н', 'і', ',', ' ', 'з', ' ', 'т', 'р', 'о', 'х', 'и',
' ', 'м', 'а', 'с', 'л', 'я', 'н', 'и', 'с', 'т', 'о', 'ю', ' ', 'б',
'л', 'и', 'с', 'к', 'у', 'ч', 'о', 'ю', ' ', 'п', 'о', 'в', 'е', 'р',
'х', 'н', 'е', 'ю', '.', ' ', 'Д', 'о', 'в', 'ж', 'и', 'н', 'а', ' ',
'є', ' ', 'п', 'о', 'к', 'а', 'з', 'н', 'и', 'к', 'о', 'м', ' ', 'я',

'к', 'о', 'с', 'т', 'і', ' ', '—', ' ', 'н', 'а', 'й', 'к', 'р', 'а',
'щ', 'е', ' ', 'в', 'и', 'б', 'и', 'р', 'а', 'т', 'и', ' ', 'с', 'т',
'р', 'у', 'ч', 'к', 'и', ' ', '1', '5', '-', '2', '0', ' ', 'с', 'а',
'н', 'т', 'и', 'м', 'е', 'т', 'р', 'і', 'в', ' ', 'з', 'а', 'в', 'д',
'о', 'в', 'ж', 'к', 'и', '.', '\n', 'Я', 'к', ' ', 'п', 'і', 'д', 'г',
'о', 'т', 'у', 'в', 'а', 'т', 'и', ' ', 'в', 'а', 'н', 'і', 'л', 'ь',
' ', 'д', 'о', ' ', 'в', 'и', 'к', 'о', 'р', 'и', 'с', 'т', 'а', 'н',
'н', 'я', '\n', 'Р', 'о', 'з', 'р', 'і', 'ж', 'т', 'е', ' ', 'с', 'т',
'р', 'у', 'ч', 'о', 'к', ' ', 'у', 'з', 'д', 'о', 'в', 'ж', ',', ' ',
'а', ' ', 'п', 'о', 'т', 'і', 'м', ' ', 'в', 'и', 'ш', 'к', 'р', 'е',
'б', 'і', 'т', 'ь', ' ', 'д', 'р', 'і', 'б', 'н', 'е', ' ', 'л', 'и',
'п', 'к', 'е', ' ', 'н', 'а', 'с', 'і', 'н', 'н', 'я', ' ', 'з', 'а',
' ', 'д', 'о', 'п', 'о', 'м', 'о', 'г', 'о', 'ю', ' ', 'к', 'і', 'н',
'ч', 'и', 'к', 'а', ' ', 'м', 'а', 'л', 'е', 'н', 'ь', 'к', 'о', 'г',
'о', ' ', 'г', 'о', 'с', 'т', 'р', 'о', 'г', 'о', ' ', 'н', 'о', 'ж',
'а', '.', '\n', 'Я', 'к', ' ', 'з', 'б', 'е', 'р', 'і', 'г', 'а', 'т',
'и', ' ', 'в', 'а', 'н', 'і', 'л', 'ь', '\n', 'У', ' ', 'г', 'е', 'р',
'м', 'е', 'т', 'и', 'ч', 'н', 'о', 'м', 'у', ' ', 'к', 'о', 'н', 'т',
'е', 'й', 'н', 'е', 'р', 'і', ' ', 'в', ' ', 'п', 'р', 'о', 'х', 'о',
'л', 'о', 'д', 'н', 'о', 'м', 'у', ' ', 'т', 'е', 'м', 'н', 'о', 'м',
'у', ' ', 'м', 'і', 'с', 'ц', 'і', ' ', '—', ' ', 'т', 'а', 'м', ' ',
'с', 'т', 'р', 'у', 'ч', 'к', 'и', ' ', 'м', 'о', 'ж', 'у', 'т', 'ь',
' ', 'з', 'б', 'е', 'р', 'і', 'г', 'а', 'т', 'и', 'с', 'я', ' ', 'д',
'о', ' ', 'д', 'в', 'о', 'х', ' ', 'р', 'о', 'к', 'і', 'в', '.', '\n',
'Я', 'к', ' ', 'г', 'о', 'т', 'у', 'в', 'а', 'т', 'и', ' ', 'з', ' ',
'в', 'а', 'н', 'і', 'л', 'л', 'ю', '\n', 'Д', 'о', 'д', 'а', 'в', 'а',
'й', 'т', 'е', ' ', 'н', 'а', 'с', 'і', 'н', 'н', 'я', ' ', 'б', 'е',
'з', 'п', 'о', 'с', 'е', 'р', 'е', 'д', 'н', 'ь', 'о', ' ', 'д', 'о',
' ', 'с', 'т', 'р', 'а', 'в', ',', ' ', 'щ', 'о', 'б', ' ', 'а', 'р',
'о', 'м', 'а', 'т', 'и', 'з', 'у', 'в', 'а', 'т', 'и', ' ', 'ї', 'х',
',', ' ', 'а', 'б', 'о', ' ', 'о', 'п', 'у', 'с', 'т', 'і', 'т', 'ь',
' ', 'с', 'т', 'р', 'у', 'ч', 'к', 'и', ' ', 'в', ' ', 'к', 'и', 'п',
'л', 'я', 'ч', 'е', ' ', 'м', 'о', 'л', 'о', 'к', 'о', ',', ' ', 'щ',
'о', 'б', ' ', 'н', 'а', 'п', 'о', 'в', 'н', 'и', 'т', 'и', ' ', 'й',
'о', 'г', 'о', ' ', 'в', 'а', 'н', 'і', 'л', 'ь', 'н', 'и', 'м', ' ',
'а', 'р', 'о', 'м', 'а', 'т', 'о', 'м', ',', ' ', 'а', ' ', 'п', 'о',
'т', 'і', 'м', ' ', 'в', 'и', 'к', 'о', 'р', 'и', 'с', 'т', 'о', 'в',
'у', 'й', 'т', 'е', ' ', 'д', 'л', 'я', ' ', 'п', 'р', 'и', 'г', 'о',
'т', 'у', 'в', 'а', 'н', 'н', 'я', ' ', 'п', 'у', 'д', 'и', 'н', 'г',
'і', 'в', ' ', 'н', 'а', ' ', 'м', 'о', 'л', 'о', 'ч', 'н', 'і', 'й',
' ', 'о', 'с', 'н', 'о', 'в', 'і', '.', '\n', 'Д', 'а', 'й', 'т', 'е',
' ', 'с', 'т', 'р', 'у', 'ч', 'к', 'у', ' ', 'в', 'и', 'с', 'о', 'х',
'н', 'у', 'т', 'и', ' ', 'п', 'а', 'р', 'у', ' ', 'д', 'н', 'і', 'в',
',', ' ', 'а', ' ', 'п', 'о', 'т', 'і', 'м', ' ', 'п', 'о', 'м', 'і',
'с', 'т', 'і', 'т', 'ь', ' ', 'й', 'о', 'г', 'о', ' ', 'в', ' ', 'б',
'а', 'н', 'к', 'у', ' ', 'з', ' ', 'ц', 'у', 'к', 'р', 'о', 'м', '.',
' ', 'П', 'р', 'и', 'б', 'л', 'и', 'з', 'н', 'о', ' ', 'з', 'а', ' ',
'т', 'и', 'ж', 'д', 'е', 'н', 'ь', ' ', 'а', 'р', 'о', 'м', 'а', 'т',
'и', 'з', 'о', 'в', 'а', 'н', 'и', 'й', ' ', 'ц', 'у', 'к', 'о', 'р',
' ', 'м', 'о', 'ж', 'н', 'а', ' ', 'в', 'и', 'к', 'о', 'р', 'и', 'с',

```
'т', 'о', 'в', 'у', 'в', 'а', 'т', 'и', ' ', 'д', 'л', 'я', ' ', 'в',
'и', 'п', 'і', 'ч', 'к', 'и', '.']
```

Функція calculate_probability приймає посимвольний список символів із файлу (data із фунції read_file) і рахує ймовірність кожного із них у тексті, записує це у словник {Symbol: Number}. Вкінці цей словник сортується по зростанню. Також створюється окремий словник bin_dict у форматі {Symbol:"empty str"}, в який в подалтшому будуть записуватися бінарні коди символів.

```
def calculate_probability(text):
    """
    Calculate probability of each symbol and sorts it by 0 -> 230
    """
    bin_dict = {}
    the_symbols = {}
    for item in text:
        if the_symbols.get(item) is None:
            the_symbols[item] = 1
        else:
            the_symbols[item] += 1
    sorted_symbols = dict(sorted(the_symbols.items(), key=lambda x:
x[1]))
    for value in sorted_symbols:
        bin_dict[value] = ""
    return sorted_symbols, bin_dict

prob, em_bin_dict = calculate_probability(the_data)
print(prob,"\n",em_bin_dict)
```

```
{'Щ': 1, '?': 1, '’': 1, 'С': 1, '7': 1, 'Т': 1, 'Ш': 1, '1': 1, '2':
1, '0': 1, 'У': 1, 'П': 1, 'Н': 2, 'М': 2, '5': 2, 'Р': 2, '-': 2,
'В': 3, '—': 4, 'Я': 4, 'Д': 5, 'є': 5, 'ш': 6, 'щ': 7, 'ї': 7, 'ю':
8, 'ц': 9, '\n': 13, 'ж': 15, 'х': 16, '.': 16, 'г': 18, 'б': 19, 'й':
20, ',': 20, 'ь': 25, 'ч': 26, 'п': 27, 'з': 27, 'я': 28, 'д': 33,
'л': 39, 'м': 40, 'у': 47, 'е': 52, 'с': 56, 'к': 65, 'р': 73, 'в':
78, 'і': 80, 'и': 86, 'н': 94, 'т': 103, 'а': 118, 'о': 127, ' ': 217}

 {'Щ': '', '?': '', '’': '', 'С': '', '7': '', 'Т': '', 'Ш': '', '1':
'', '2': '', '0': '', 'У': '', 'П': '', 'Н': '', 'М': '', '5': '',
'Р': '', '-': '', 'В': '', '—': '', 'Я': '', 'Д': '', 'є': '', 'ш':
'', 'щ': '', 'ї': '', 'ю': '', 'ц': '', '\n': '', 'ж': '', 'х': '',
'.': '', 'г': '', 'б': '', 'й': '', ',': '', 'ь': '', 'ч': '', 'п':
'', 'з': '', 'я': '', 'д': '', 'л': '', 'м': '', 'у': '', 'е': '',
'с': '', 'к': '', 'р': '', 'в': '', 'і': '', 'и': '', 'н': '', 'т':
'', 'а': '', 'о': '', ' ': ''}
```

Функція create_code заповнює bin_dict бітовими презентаціями кожного символу і додає ймовірності та сортує список за зростанням.

```
def create_code():
    """
    Creates binar codes of each symbol
    """
    prob, bin_dict = calculate_probability(the_data)
    item = list(prob.items())
    for i, value in enumerate(item):
        elem1 = item[0][0] + item[1][0]
        for j in item[0][0]:
            bin_dict[j] = "0" + bin_dict[j]
        for k in item[1][0]:
            bin_dict[k] = "1" + bin_dict[k]
        elem2 = item[0][1] + item[1][1]
        tup = (elem1, elem2)
        del item[0]
        del item[0]
        item.append(tup)
        item = sorted(item, key=lambda x: x[1])
    return item, bin_dict

item, bin_code = create_code()
print(item, "\n", bin_code)
```

```
[('влШ120ДєУПВйім,ьинует ч\nшщпзсяжїюаокх.д—ЯНМ5Р-Щ?’С7Тцгбр', 1658)]
 {'Щ': '11110010010', '?': '11110010011', '’': '11110010100', 'С':
'11110010101', '7': '11110010110', 'Т': '11110010111', 'Ш':
'0001100000', '1': '0001100001', '2': '0001100010', '0': '0001100011',
'У': '0001101100', 'П': '0001101101', 'Н': '1111000100', 'М':
'1111000101', '5': '1111000110', 'Р': '1111000111', '-': '1111001000',
'В': '000110111', '—': '111100000', 'Я': '111100001', 'Д': '00011001',
'є': '00011010', 'ш': '10100110', 'щ': '10100111', 'ї': '10111110',
'ю': '10111111', 'ц': '11110011', '\n': '1010010', 'ж': '1011110',
'х': '1110100', '.': '1110101', 'г': '1111010', 'б': '1111011', 'й':
'000111', ',': '001110', 'ь': '001111', 'ч': '101000', 'п': '101010',
'з': '101011', 'я': '101110', 'д': '111011', 'л': '00010', 'м':
'00110', 'у': '01100', 'е': '01101', 'с': '10110', 'к': '11100', 'р':
'11111', 'в': '0000', 'і': '0010', 'и': '0100', 'н': '0101', 'т':
'0111', 'а': '1100', 'о': '1101', ' ': '100'}
```

Функція huffman_encode приймає як аргумент текст, який ми хочемо закодувати (це той самий посимвольний текст, що ми читаємо з файлу), проходиться по цьому списку і створює повідомлення.

```
def huffman_encode(the_data):
    """
    Codes message
    """
    item, bin_dict = create_code()
    text = ""
    for i, elem in enumerate(the_data):
        if elem in bin_dict:
```

```
            text += str(bin_dict[elem])
    return text

compressed = huffman_encode(the_data)
print(compressed)
```

```
1111000100110001110110011111110000010001111010101100100000011000101001
0000100011111001010001100101100111110110000000100111001101111110100101
1001111101000001100101111111011100111110001100100000001001010100010101010
0011110011001010001111100100111011011011011001101111110111110011101001
1101011000001100100101010110101110000011101110110110110101110001110100
1011101110010000000100111101111111100011101001000101110000011111110011
1111100101001110010100101100111111111011001010001110001000011101001011 7
0111001001011111011101001001010111111011011011111100101111010110001110
1001000111110010000000100111001101111110100101100111110100000110000001
1000111010011101011010010111100100101101100011111001110001101100000011
0001010010000100011111111100100111010010000110111110001010010000100011 1
1100001101100000110101000101011011010101101000001111101111110101010000
0111100001101111001010010111011100010000011100111010010110110100010110
1111011111000100000111100101100011011001110010001111100100110011111110
1001101100011111101011000001101111101010111001000101110000010011011011
1100100011100111100111011111011001011000100011001101000111101100111000
0110010011011111111110100001011101101101000111010101001110100111010110 0
1111001010101111111110110010100011011110010000001100010100100001000101 0
0111100000100111100110110110010101000010001011101110011100000000100111
1110001000011101001010011111011000110001110000110111111010000001011010
1110100101010001010110000010101110100101011110000000110111111101001100 1
1010101010110111010011110011000000100111001001010101101110111010110000
0110111110001010010000100011111001010111001111000101110011101111001111
0101100101101110011001111101100100101100111111000101110100000100011100 1
1111001111001011011100011010000000010111011101101101011111100001000000
1000000001011101110001100101100010000110101011111010000001100010100100
0010001010001011100100111110100010111100011001110101100111100011101101
1010011001111100100111100000100111100110110110000001100010100100001000
1111100101011100111100101111100101111100111001010001111100100111100010
1011011110010110010011100010000011101001100000100110110010111110101111
0100101011010111001010000101110110000001100101111010100001101100101011
0101110000011101110100111010110010100100001100111010000111101000100011
1010010100011011111101010010001110100011111010101111000010100011111001
0010101100110110111111110100110111000110000001100011100101001011001111
1110110010100011100010010000001100010100100001000101000011000101011001
1110111001110011111000111010010110101110101000001010011101111111001011
1101101010100111010010010100011011111101010100111010010001011100101100
0100101010001010011101001011101110000101000000010011100110111111010010
1100111110100000110010111111011100111111011010111010011101100010101110 1
0011001111111010011011000111010010101111001111001100101011111010010101
0011011111110110100001100101111001011101100101101101000101101111011110
0010011101001001011001111111111000000100001010011011011011011111011000
1001000000110110011101111011111011111111011011001010101101000110101110 1
1010101100101111110111001111101101011101001010111001010011011011110011
0100010110011101111010011011101011010010111100010011001011100000010100
```

1010110011100111110011100111110100111010000100111010101001110100100101
0001101111110101010011101001001111001110111001111101101000110111100100
0110010010110011111111110000000010100000110101001010100010111011011100
0001101111111110111011110011010101010110111000110100011111011111010110
1001110100101001111101100111101101100000101100100000000100111001101111
1010010110011111000101110010010110101010111111100000010111100101101110
1000000110001010010000100011111110101101001011110000111100100000001001
1110111111111100011101001000101110000011111100111111001010011101100100
0000110001010010000100011111010010000110000001100111001100000111011101
1011001100111111101001101100011101010010001110100011101101001100101110
1111100100011100110111111010010100001010110100000001000111010000110110
0001111011110011011001010001101111110101001010010110011111111011001010
0011100010000111010010101100010011011111010111001100100101011001101101
1111110100110111000110000001100011100100011101001100000100110110010101
0111110110010111100101001000111010010101110001111111111011110100010010
0001101100101100001010111001010100101100111110110111111100111101100010
0100101101110001100101000110110111111001010101101000001101111111111010
0010101101101111111110101100000110011101000010111100100010111001000001
1010100101010110111100110010101101010100111001101001101001011101110011
0110110011100101001111000001000101110000011111100111111100101001110110
1100000000100111101101001111111000111010010010110011111111011001010001
1000100100000011000011111000110111100100000011000100001100011100101101
0001010111010000110011010111111110010000010010101111000000111011110100
0010111101110001001110101101001011110000111100100101010001011101111110
1011010111011000000110001110100100000001100010100100001000111110011101
1101100000000100111001101111110100101100111110001010101101110101001011
1000111110110101111111001010111100111011011001011001111111101100101000
1101111001000110010101111101111010000101111000111010011001001010101101
0111001000110100000000100101001101110011111011011111011001001110011111
0111011111110010111101101010110110000010010010101011100011011000101110
0101100010010101011011110100101011110010011101111011010101101001101101
1110101101101111111001110000100101101000010011100110010000110110000010
0110101010011111110011011111010110110011110101101101100111111111110111
1010110110001011101101111011001110101101001011110000111100100101011111
1011011011111100101111010110001110100100000011000101001000010001111101
0010000110110010011110100110111111001100110101110100101000010111010011
0011001001110011010101011101101000111010101101111110010100000010010101
0111111101111010011010001011011110110101110100110011001000111011010011
0010111010011001100100001100010101101111001100101001111000001000111110
0001101001011001111111101100101000111000100100001101101101111001100011
1001111100101011111110110110111111001011110101100011101001011010111010
1110111101100111011000011011110100100111111101111000010000011101011010
0101111000011110010011110101101011101100000011000111010010010101110000
0011000101001000010000101011111110100100001100111011110111100000011000
0011101110110110001011100101100010010101011011101001111011011011010111
0101011011011001101111110110111101101010011111101100111011110110010110
0111111111100000000111010010100111110111110111001100111111101001101100
0111010010101101100000001100011101001001011111011101000011101001100111
0111101100110110101001100101100111001001110011111001011001111111101100
1010001110001001000000100111000100101010000101011101010000110110000110

```
11010001011011110011010011101001010011111011111011100010111001010101101
10000010101000111010010000011111011111010110110000001100010100100001000
01111010101000011010011001111111010011011000111110100110001110100110010
01010101101011100100011010000000100111001101111110100101100111110100000
01100000111011101101100111011000101011101001010101111101001111010110101
11011000000110001010101101110100101010011001110110100010111110100010000
01000101110010000110110100010110110100001010010000111100110110110010111
01000000101110101101001000011001110000011101110110110010110011111111011
10010100011100011001000000001001011011011110100010101100011101001001010
10110011111011001001110110101001000000011101001100100101010110101110010
00110100101010110100110001010110011100100111001111100000011111011111010
11011000000100111101111000101111000110010010101110011110011011001110011
11111010011011101011000000110110111111010011110110001001001010110101110
11001010111100100011101001011110111011011010101001111100110011111110100
11011000111010010101111010000110001010100000111100111100110110011100110
11111110000110110110111100101110010000000100111001101111110100101100111
11010000001100000011000111010010011101100010101110100000001001
01010001010100011100010011101011
```

Функція huffman_decode приймає як аргумент закодоване повідомлення, проходиться по ньому, шукає потрібний ключ у словнику бінарних значень символів, додає потрібний символ до декодованого повідомлення і обрізає закодоване повідомлення на довжину ключа, щоб не декодувати постійно один і той самий символ.

```python
def huffman_decode(coded_message):
    """
    Decodes message
    """
    item, bin_dict = create_code()
    decoded_text = ""
    while coded_message:
        for key, value in bin_dict.items():
            if coded_message.startswith(value):
                decoded_text += key
                coded_message = coded_message[len(value) :]
    return decoded_text

decompressed = huffman_decode(compressed)
print(decompressed)
```

```
Натуральну ваніль часто використовують у випічці та десертах.
Дізнайтеся, як вибрати найкращі стручки, як їх зберігати та
використовувати.
Що таке ваніль?
Ваніль має неповторний м’який, солодкий смак та аромат. Вона належить
до сімейства орхідейних. Стручок ванілі – це плід квітки, що утворився
після завершення цвітіння. Ваніль з Мадагаскару становить 75 відсотків
від усієї ванілі на ринку. Решта – це ваніль з Таїті та Мексики, але
її значно важче знайти.
Довгі, чорні, тонкі та зморшкуваті стручки ванілі містять тисячі
```

дрібних чорних насінин, які використовуються для ароматизації переважно солодких страв і особливо добре поєднуються з шоколадом. Наявність крихітних чорних цяточок у страві є підтвердженням того, що була використана справжня ваніль.
Як вибрати найкращу ваніль
Шукайте ароматні, темно-коричневі, майже чорні стручки, злегка зморшкуваті, але пружні, з трохи маслянистою блискучою поверхнею. Довжина є показником якості — найкраще вибирати стручки 15-20 сантиметрів завдовжки.
Як підготувати ваніль до використання
Розріжте стручок уздовж, а потім вишкребіть дрібне липке насіння за допомогою кінчика маленького гострого ножа.
Як зберігати ваніль
У герметичному контейнері в прохолодному темному місці — там стручки можуть зберігатися до двох років.
Як готувати з ваніллю
Додавайте насіння безпосередньо до страв, щоб ароматизувати їх, або опустіть стручки в кипляче молоко, щоб наповнити його ванільним ароматом, а потім використовуйте для приготування пудингів на молочній основі.
Дайте стручку висохнути пару днів, а потім помістіть його в банку з цукром. Приблизно за тиждень ароматизований цукор можна використовувати для випічки.

Функція calculate_len приймає посимвольний список і рахує довжину закодованого і декодованого повідомлення

```
def calculate_len(the_data):
    """
    Calculates len of encoded and decoded message
    """
    encoded_message = huffman_encode(the_data)
    decoded_message = huffman_decode(encoded_message)
    len_coded_message = len(encoded_message) // 8
    len_decoded_message = len(decoded_message) // 8
    return len_coded_message, len_decoded_message

len_coded_message, len_decoded_message = calculate_len(the_data)
print(len_coded_message, len_decoded_message)
```

```
983 207
```

```
assert decompressed == file_text
```

```
import sys
compression_percentage = (1 -
sys.getsizeof(compressed)/sys.getsizeof(the_data))*100
# print(f"Original message length: {sys.getsizeof(the_data)} bits")
# print(f"Encoded message length: {sys.getsizeof(compressed)} bits")
# print(f"Compression percentage: {compression_percentage:.2f}%")
print(f"Original message length: {sys.getsizeof(the_data)}")
```

```
print(f"Encoded message length: {sys.getsizeof(compressed)}")
print(f"Compression percentage: {compression_percentage:.2f}%")
```

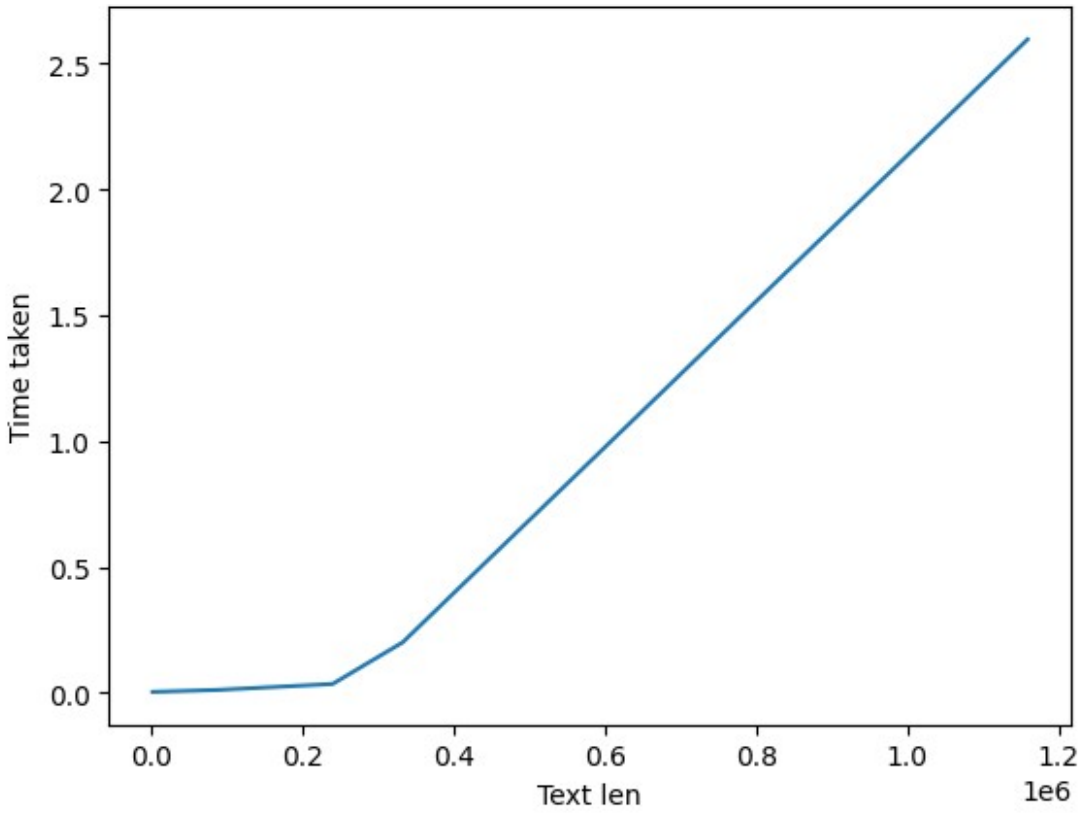
```
Original message length: 14360
Encoded message length: 7920
Compression percentage: 44.85%
```

Клас HuffmanCode використовує всі функції описані вище

```
class HuffmanCode:
    def __init__(self, file_path: str):
        self.file_path = file_path
        self.file_text, self.the_data = self.read_file()
        self.prob, self.bin_dict =
self.calculate_probability(self.the_data)
        self.item, self.bin_dict = self.create_code()

    def read_file(self):
        """
        Reads data from file
        """
        file_text = ""
        data = []
        with open(self.file_path, "r", encoding="utf-8") as file:
            for line in file:
                file_text += line
                for character in line:
                    data.append(character)
        return file_text, data

    def calculate_probability(self, text: list):
        """
        Calculate probability of each symbol and sorts it by 0 -> 230
        """
        bin_dict = {}
        the_symbols = {}
        for item in text:
            if the_symbols.get(item) is None:
                the_symbols[item] = 1
            else:
                the_symbols[item] += 1
        sorted_symbols = dict(sorted(the_symbols.items(), key=lambda
x: x[1]))
        for value in sorted_symbols:
            bin_dict[value] = ""
        return sorted_symbols, bin_dict

    def create_code(self):
        """
        Creates binary codes of each symbol
```

```
    """
    prob, bin_dict = self.prob, self.bin_dict
    item = list(prob.items())
    for i, value in enumerate(item):
        elem1 = item[0][0] + item[1][0]
        for j in item[0][0]:
            bin_dict[j] = "0" + bin_dict[j]
        for k in item[1][0]:
            bin_dict[k] = "1" + bin_dict[k]
        elem2 = item[0][1] + item[1][1]
        tup = (elem1, elem2)
        del item[0]
        del item[0]
        item.append(tup)
        item = sorted(item, key=lambda x: x[1])
    return item, bin_dict

def huffman_encode(self) -> str:
    """
    Codes message
    """
    item, bin_dict = self.item, self.bin_dict
    text = ""
    for i, elem in enumerate(self.the_data):
        if elem in bin_dict:
            text += str(bin_dict[elem])
    return text

def huffman_decode(self, coded_message: str) -> str:
    """
    Decodes message
    """
    item, bin_dict = self.item, self.bin_dict
    decoded_text = ""
    while coded_message:
        for key, value in bin_dict.items():
            if coded_message.startswith(value):
                decoded_text += key
                coded_message = coded_message[len(value) :]
    return decoded_text

def calculate_len(self):
    """
    Calculates length of encoded and decoded message
    """
    encoded_message_f = self.huffman_encode()
    decoded_message_f = self.huffman_decode(encoded_message_f)
    len_coded_message_f = len(encoded_message_f) // 8
    len_decoded_message_f = len(decoded_message_f) // 8
    return len_coded_message_f, len_decoded_message_f
```

```
    def calculate_compression_percentage(self):
        """
        Calculates the compression percentage of the Huffman
        encoded message relative to the original message
        """
        len_coded_message, len_original_message = self.calculate_len()
        compression_percentage = (1 - (len_coded_message /
len_original_message)) * 100
        return compression_percentage

import matplotlib.pyplot as plt
import time
from tqdm import tqdm

def test_huffman():
    time_taken = []
    lengh = []
    for file_path in lst_files:
        _, data = read(file_path)
        lengh.append(len(data))
        start = time.time()
        huffman = huffman_encode(data)
        end = time.time()
        time_taken.append(abs(start - end))
    plt.xlabel("Text len")
    plt.ylabel("Time taken")
    x = lengh
    y = time_taken
    plt.plot(x, y)

test_huffman()
```



З даного графіку ми можемо зробити висновок, що алгоритм Гаффмана краще працює на файлах меншого розміру і на файлах з великою кількістю повторюваних елементів.