Дискретні структури

ІПЗ-21-2
Губарєв Ростислав Вадимович

Лабораторна робота №9.2

Створіть програму, яка будує коди Хаффмана та оцінює ступінь стиснення з їхньою допомогою для довільного заданого файлу. * Створіть програму, яка стискає файли за допомогою кодів Хаффмана.

Код

```
import heapq
import os


class HuffmanNode:
    """
    Представляє вузол дерева Хаффмана.
    """

    def __init__(self, char, freq):
        """
        Ініціалізує новий об'єкт HuffmanNode.

        Параметри:
            char (str): Символ, пов'язаний з вузлом.
            freq (int): Частота входження символу.
        """
        self.char = char
        self.freq = freq
        self.left = None
        self.right = None

    def __lt__(self, other):
        """
        Порівнює два вузли Хаффмана за їхніми частотами.

        Параметри:
            other (HuffmanNode): Інший вузол Хаффмана для порівняння.

        Повертає:
            bool: True, якщо поточний вузол має меншу частоту, ніж інший вузол.
        """
        return self.freq < other.freq


class HuffmanCoding:
    """
    Клас для створення кодів Хаффмана та стиснення даних.
    """

    def __init__(self):
        self.heap = []
        self.codes = {}
        self.reverse_mapping = {}
        self.root = None

    def build_frequency_dict(self, data):
        """
        Побудова словника частот символів у вхідних даних.

        Параметри:
```

```
            data (str): Вхідні дані.

        Повертає:
            dict: Словник з символами та їх частотами вхідних даних.
        """
        frequency_dict = {}
        for char in data:
            frequency_dict[char] = frequency_dict.get(char, 0) + 1
        return frequency_dict

    def build_heap(self, frequency_dict):
        """
        Побудова купи з вузлів Хаффмана на основі словника частот.

        Параметри:
            frequency_dict (dict): Словник з символами та їх частотами.

        Повертає:
            list: Купа вузлів Хаффмана.
        """
        heap = []
        for char, freq in frequency_dict.items():
            heapq.heappush(heap, HuffmanNode(char, freq))
        return heap

    def merge_nodes(self, heap):
        """
        Об'єднання вузлів Хаффмана у дерево.

        Параметри:
            heap (list): Купа вузлів Хаффмана.

        Повертає:
            HuffmanNode: Кореневий вузол дерева Хаффмана.
        """
        while len(heap) > 1:
            node1 = heapq.heappop(heap)
            node2 = heapq.heappop(heap)
            merged = HuffmanNode(None, node1.freq + node2.freq)
            merged.left = node1
            merged.right = node2
            heapq.heappush(heap, merged)
        return heap[0]

    def build_codes_helper(self, root, current_code):
        """
        Допоміжна функція для побудови кодів Хаффмана.

        Параметри:
            root (HuffmanNode): Поточний вузол дерева Хаффмана.
            current_code (str): Поточний код символу.
        """
        if root is None:
            return
        if root.char is not None:
            self.codes[root.char] = current_code
        self.build_codes_helper(root.left, current_code + "0")
        self.build_codes_helper(root.right, current_code + "1")

    def build_codes(self, root):
        """
        Побудова кодів Хаффмана на основі дерева Хаффмана.
```

```
        Параметри:
            root (HuffmanNode): Кореневий вузол дерева Хаффмана.
        """
        self.build_codes_helper(root, "")

    def get_encoded_data(self, data):
        """
        Отримання закодованих даних на основі кодів Хаффмана.

        Параметри:
            data (str): Вхідні дані.

        Повертає:
            str: Закодовані дані.
        """
        encoded_data = ""
        for char in data:
            encoded_data += self.codes[char]
        return encoded_data

    def pad_encoded_data(self, encoded_data):
        """
        Додавання доповнення до закодованих даних для вирівнювання.

        Параметри:
            encoded_data (str): Закодовані дані.

        Повертає:
            str: Закодовані дані з доданим доповненням.
        """
        extra_padding = 8 - len(encoded_data) % 8
        for i in range(extra_padding):
            encoded_data += "0"
        padded_info = "{0:08b}".format(extra_padding)
        encoded_data = padded_info + encoded_data
        return encoded_data

    def get_byte_array(self, padded_encoded_data):
        """
        Отримання байтового масиву з доповнених закодованих даних.

        Параметри:
            padded_encoded_data (str): Закодовані дані з доданим доповненням.

        Повертає:
            bytearray: Байтовий масив з закодованими даними.
        """
        if len(padded_encoded_data) % 8 != 0:
            print("Encoded data is not padded properly.")
            exit(0)
        b = bytearray()
        for i in range(0, len(padded_encoded_data), 8):
            byte = padded_encoded_data[i:i + 8]
            b.append(int(byte, 2))
        return b

    def compress(self, input_file, output_file):
        """
        Стиснення файлу за допомогою кодів Хаффмана.

        Параметри:
```

```
            input_file (str): Шлях до вхідного файлу.
            output_file (str): Шлях до стисненого файлу.

        Повертає:
            str: Шлях до стисненого файлу.
        """
        with open(input_file, 'r') as file:
            data = file.read()
            data = data.rstrip()

        frequency_dict = self.build_frequency_dict(data)
        heap = self.build_heap(frequency_dict)
        root = self.merge_nodes(heap)
        self.build_codes(root)
        encoded_data = self.get_encoded_data(data)
        padded_encoded_data = self.pad_encoded_data(encoded_data)
        byte_array = self.get_byte_array(padded_encoded_data)

        with open(output_file, 'wb') as file:
            file.write(bytes(byte_array))

        print("Compression successful!")
        return output_file

    def remove_padding(self, padded_encoded_data):
        """
        Видалення доповнення з закодованих даних.

        Параметри:
            padded_encoded_data (str): Закодовані дані з доданим доповненням.

        Повертає:
            str: Закодовані дані без доповнення.
        """
        padded_info = padded_encoded_data[:8]
        extra_padding = int(padded_info, 2)
        padded_encoded_data = padded_encoded_data[8:]
        encoded_data = padded_encoded_data[:-1 * extra_padding]
        return encoded_data

    def decode_data(self, encoded_data, root):
        """
        Розкодування закодованих даних на основі дерева Хаффмана.

        Параметри:
            encoded_data (str): Закодовані дані.
            root (HuffmanNode): Кореневий вузол дерева Хаффмана.

        Повертає:
            str: Розкодовані дані.
        """
        current_node = root
        decoded_data = ""
        for bit in encoded_data:
            if current_node is None:
                break
            if bit == '0':
                current_node = current_node.left
            else:
                current_node = current_node.right

            if current_node is not None and current_node.char is not None:
```

```
                decoded_data += current_node.char
                current_node = root

        return decoded_data

def calculate_compression_ratio(original_file, compressed_file):
    """
    Обчислення ступеня стиснення між оригінальним файлом та стисненим файлом.

    Параметри:
        original_file (str): Шлях до оригінального файлу.
        compressed_file (str): Шлях до стисненого файлу.

    Повертає:
        float: Ступінь стиснення у відсотках.
    """
    original_size = os.path.getsize(original_file)
    compressed_size = os.path.getsize(compressed_file)
    compression_ratio = (compressed_size / original_size) * 100
    return compression_ratio


# Застосовуємо кодування Хаффмана до заданого файлу
input_file = "input.txt"
output_file = "compressed.bin"

huffman = HuffmanCoding()
compressed_file = huffman.compress(input_file, output_file)

# Розраховуємо ступінь стиснення
compression_ratio = calculate_compression_ratio(input_file, compressed_file)
print("Compression ratio: {:.2f}%".format(compression_ratio))
```

Результат

```
"H:\University\2 курс\2 семестр\Дискр
Compression successful!
Compression ratio: 53.16%

Process finished with exit code 0
```