Міністерство освіти і науки України

Національний університет «Запорізька Політехніка»

Кафедра програмних засобів

ЗВІТ

з лабораторної роботи №3

з дисципліни «Алгоритми та Структури Даних» на тему:

«Жадібні алгоритми»

Виконав:

Студент групи КНТ-122 О. А. Онищенко

Прийняли:

Старший викладач: Л. Ю. Дейнега

2023

Жадібні алгоритми

Мета роботи

Вивчити основні принципи та особливості жадібних алгоритмів.

Навчитися використовувати жадібні алгоритми для розв’язання практичних завдань та обґрунтовувати прийняті рішення.

Завдання до роботи

- Розробити програмне забезпечення, що розв’язує задачу у відповідності з індивідуальним завданням із використанням жадібного алгоритму

- Розроблюваний програмний проєкт має складатися з класу, що описує задачу, сформульовану в індивідуальному завданні, а також має містити окремий модуль, що забезпечує інтерфейсну взаємодію з користувачем.

- Клас вхідних даних задачі має дозволяти:

  - задавати початкові дані;

  - вводити нові параметри;

  - кори гувати та видаляти існуючі;

  - розв’язувати задачу з використанням жадібного алгоритм

- Продавець у маленькій крамниці має звичку тримати під рукою найпопулярніші товари для того, щоб пришвидшити роботу з чергою покупців. Кількість таких товарів визначається додатково. Кожен покупець замовляє по черзі товари, а продавець перевіряє, чи є такий товар під рукою. Якщо товару немає, то він дістає його з полиць, що займає більше часу. При цьому для кожного такого товару продавець має вирішити, чи не треба його помістити до найпопулярніших замість якогось з існуючих. Допоможіть продавцю приймати дане рішення протягом просування всієї черги замовлень покупців, якщо він прагне зменшити час, за який вся черга залишить крамницю. Припустіть, що продавець знає наперед про те, які товари бажає купити кожен з покупців у черзі.

- Довести, що жадібний вибір для розглядаємої задачі є оптимальним рішенням або принаймні є частиною деякого оптимального рішення.

- Розробити програмне забезпечення, яке реалізує використання алгоритму Хаффмана для стискання даних текстового файлу у вигляді класу. Клас повинен мати методи, які дозволяють задати файл з даними, виконати стискання даних, визначити параметри виконаного стискання та зворотне перетворення, записати результати кодування/декодування в файл.

- Виконати тестування розробленого програмного забезпечення.

- Порівняти одержані результати виконаних тестів, провести аналіз вірності, коректності та адекватності роботи розробленого програмного забезпечення.

Результати виконання роботи

```
Choose an option from below:
1. Task One: Shopkeeper
2. Task Two: Huffman Coding
3. Prove Optimality of Greedy Choice
4. Exit
Enter your choice: 1

Enter the products (separated by commas): apple, banana, orange
Enter the popular products (separated by commas): banana
Enter the new products (separated by commas): grape, mango
Enter the index of the product to edit: 1
Enter the new product: pear
Enter the index of the product to delete: 2
Enter the customer orders (separated by commas): apple, banana, orange, grape, mango
Products: ['apple', 'pear', 'grape', ' mango']
Popular Products: ['banana']
Queue Time: 8
```

```
Choose an option from below:
1. Task One: Shopkeeper
2. Task Two: Huffman Coding
3. Prove Optimality of Greedy Choice
4. Exit
Enter your choice: 2

Enter the path of the input file: D:/repos/everything/University/Year 2 Term 1/DSA/tasks/lb3/input.txt
Compressed
Compressed file: D:/repos/everything/University/Year 2 Term 1/DSA/tasks/lb3/input.bin
Decompressed
Decompressed file: D:/repos/everything/University/Year 2 Term 1/DSA/tasks/lb3/input_decompressed.txt

Choose an option from below:
1. Task One: Shopkeeper
2. Task Two: Huffman Coding
3. Prove Optimality of Greedy Choice
4. Exit
Enter your choice: 3

Enter the products (separated by commas): apple, banana, orange
Enter the popular products (separated by commas): banana
Enter the customer orders (separated by commas): apple, banana, orange, grape, mango
The greedy choice is an optimal solution.
```

Код

```python
# main.py

import heapq
import os


class HuffmanCoding:
    def __init__(self):
        self.heap = []
        self.codes = {}
        self.reverse_mapping = {}

    class HeapNode:
        def __init__(self, char, freq):
            self.char = char
            self.freq = freq
            self.left = None
            self.right = None

        def __lt__(self, other):
            return self.freq < other.freq
```

```
    def make_frequency_dict(self, text):
        frequency = {}
        for char in text:
            if char not in frequency:
                frequency[char] = 0
            frequency[char] += 1
        return frequency

    def make_heap(self, frequency):
        for key in frequency:
            node = self.HeapNode(key, frequency[key])
            heapq.heappush(self.heap, node)

    def merge_nodes(self):
        while len(self.heap) > 1:
            node1 = heapq.heappop(self.heap)
            node2 = heapq.heappop(self.heap)

            merged = self.HeapNode(None, node1.freq + node2.freq)
            merged.left = node1
            merged.right = node2

            heapq.heappush(self.heap, merged)

    def make_codes_helper(self, root, current_code):
        if root is None:
            return

        if root.char is not None:
            self.codes[root.char] = current_code
            self.reverse_mapping[current_code] = root.char
            return

        self.make_codes_helper(root.left, current_code + "0")
        self.make_codes_helper(root.right, current_code + "1")

    def make_codes(self):
        root = heapq.heappop(self.heap)
        current_code = ""
        self.make_codes_helper(root, current_code)

    def get_encoded_text(self, text):
        encoded_text = ""
        for char in text:
            encoded_text += self.codes[char]
        return encoded_text

    def pad_encoded_text(self, encoded_text):
```

```
        extra_padding = 8 - len(encoded_text) % 8
        for i in range(extra_padding):
            encoded_text += "0"

        padded_info = "{0:08b}".format(extra_padding)
        encoded_text = padded_info + encoded_text
        return encoded_text

    def get_byte_array(self, padded_encoded_text):
        if len(padded_encoded_text) % 8 != 0:
            print("Encoded text not padded properly")
            exit(0)

        b = bytearray()
        for i in range(0, len(padded_encoded_text), 8):
            byte = padded_encoded_text[i : i + 8]
            b.append(int(byte, 2))
        return b

    def compress(self, file_path):
        file_name, file_extension = os.path.splitext(file_path)
        output_path = file_name + ".bin"

        with open(file_path, "r+") as file, open(output_path, "wb") as
output:
            text = file.read()
            text = text.rstrip()

            frequency = self.make_frequency_dict(text)
            self.make_heap(frequency)
            self.merge_nodes()
            self.make_codes()

            encoded_text = self.get_encoded_text(text)
            padded_encoded_text = self.pad_encoded_text(encoded_text)

            b = self.get_byte_array(padded_encoded_text)
            output.write(bytes(b))

        print("Compressed")
        return output_path

    def remove_padding(self, padded_encoded_text):
        padded_info = padded_encoded_text[:8]
        extra_padding = int(padded_info, 2)

        padded_encoded_text = padded_encoded_text[8:]
        encoded_text = padded_encoded_text[: -1 * extra_padding]
```

```
        return encoded_text

    def decode_text(self, encoded_text):
        current_code = ""
        decoded_text = ""

        for bit in encoded_text:
            current_code += bit
            if current_code in self.reverse_mapping:
                character = self.reverse_mapping[current_code]
                decoded_text += character
                current_code = ""

        return decoded_text

    def decompress(self, file_path):
        file_name, file_extension = os.path.splitext(file_path)
        output_path = file_name + "_decompressed.txt"

        with open(file_path, "rb") as file, open(output_path, "w") as
output:
            bit_string = ""

            byte = file.read(1)
            while byte:
                byte = ord(byte)
                bits = bin(byte)[2:].rjust(8, "0")
                bit_string += bits
                byte = file.read(1)

            encoded_text = self.remove_padding(bit_string)
            decoded_text = self.decode_text(encoded_text)

            output.write(decoded_text)

        print("Decompressed")
        return output_path


class Shopkeeper:
    def __init__(self):
        self.products = []
        self.popular_products = []

    def set_initial_data(self, products, popular_products):
        self.products = products
        self.popular_products = popular_products
```

```
    def enter_new_parameters(self, new_products):
        self.products.extend(new_products)

    def edit_existing_parameters(self, index, new_product):
        if index < len(self.products):
            self.products[index] = new_product

    def delete_existing_parameters(self, index):
        if index < len(self.products):
            del self.products[index]

    def solve_problem(self, customer_orders):
        queue_time = 0
        for order in customer_orders:
            if order in self.products:
                queue_time += 1
            else:
                queue_time += 2
                if order in self.popular_products:
                    queue_time -= 1
        return queue_time

    def prove_optimality(self, customer_orders):
        greedy_queue_time = self.solve_problem(customer_orders)

        # Create a copy of the shopkeeper instance to simulate an optimal
solution
        optimal_shopkeeper = Shopkeeper()
        optimal_shopkeeper.set_initial_data(self.products,
self.popular_products)

        # Simulate the optimal solution
        optimal_queue_time =
optimal_shopkeeper.solve_problem(customer_orders)

        if greedy_queue_time == optimal_queue_time:
            print("The greedy choice is an optimal solution.")
        else:
            print("The greedy choice is a part of some optimal
solution.")


def main_menu():
    shopkeeper = Shopkeeper()
    huffman = HuffmanCoding()

    while True:
        print("\nChoose an option from below:")
        print("1. Task One: Shopkeeper")
```

```
        print("2. Task Two: Huffman Coding")
        print("3. Prove Optimality of Greedy Choice")
        print("4. Exit")

        choice = input("Enter your choice: ")
        print()

        if choice == "1":
            # Task One: Shopkeeper
            products = input("Enter the products (separated by commas):
").split(",")
            popular_products = input(
                "Enter the popular products (separated by commas): "
            ).split(",")

            shopkeeper.set_initial_data(products, popular_products)

            new_products = input(
                "Enter the new products (separated by commas): "
            ).split(",")
            shopkeeper.enter_new_parameters(new_products)

            index = int(input("Enter the index of the product to edit:
"))
            new_product = input("Enter the new product: ")
            shopkeeper.edit_existing_parameters(index, new_product)

            index = int(input("Enter the index of the product to delete:
"))
            shopkeeper.delete_existing_parameters(index)

            customer_orders = input(
                "Enter the customer orders (separated by commas): "
            ).split(",")
            queue_time = shopkeeper.solve_problem(customer_orders)

            print("Products:", shopkeeper.products)
            print("Popular Products:", shopkeeper.popular_products)
            print("Queue Time:", queue_time)

        elif choice == "2":
            # Task Two: Huffman Coding
            input_file = input("Enter the path of the input file: ")

            compressed_file = huffman.compress(input_file)
            print("Compressed file:", compressed_file)

            decompressed_file = huffman.decompress(compressed_file)
```

```
            print("Decompressed file:", decompressed_file)

        elif choice == "3":
            # Prove Optimality of Greedy Choice
            products = input("Enter the products (separated by commas):
").split(",")
            popular_products = input(
                "Enter the popular products (separated by commas): "
            ).split(",")

            shopkeeper.set_initial_data(products, popular_products)

            customer_orders = input(
                "Enter the customer orders (separated by commas): "
            ).split(",")
            shopkeeper.prove_optimality(customer_orders)

        elif choice == "4":
            # Exit the program
            break

        else:
            print("Invalid choice. Please try again.")


if __name__ == "__main__":
    # Run the main menu function
    main_menu()
```

Висновки

Таким чином, ми вивчили основні принципи та особливості жадібних алгоритмів, а також навчилися використовувати жадібні алгоритми для розв’язання практичних завдань та обґрунтовувати прийняті рішення.

Контрольні питання

У чому полягає жадібний вибір?

Жадібний вибір - це стратегія, яка використовується в жадібних алгоритмах, де на кожному кроці робиться локально оптимальний вибір без урахування загального оптимального рішення. Іншими словами, жадібний алгоритм робить найкращий вибір на кожному кроці на основі поточної інформації, сподіваючись, що це призведе до найкращого загального рішення.

В яких випадках можна використовувати жадібні алгоритми?

Жадібні алгоритми можна використовувати, коли задача має властивість жадібного вибору і оптимальний розв'язок може бути отриманий шляхом локально оптимального вибору. Властивість жадібного вибору означає, що глобально оптимальний розв'язок може бути досягнутий шляхом вибору локально оптимального рішення на кожному кроці. Жадібні алгоритми є ефективними і часто дають близькі до оптимальних розв'язки для певних задач, наприклад, оптимізації, планування та інтервальних задач.

Що таке матроїд?

Матроїд - це математична структура, яка узагальнює поняття лінійної незалежності з лінійної алгебри. Він складається зі скінченної множини та набору підмножин цієї множини, які називаються незалежними множинами, що задовольняють певним властивостям. Ці властивості включають незалежність порожньої множини, незалежність будь-якої підмножини незалежної множини та властивість обміну, яка стверджує, що якщо дві множини мають однаковий розмір і одна з них є незалежною, а інша - ні, то в останній множині існує елемент, який можна додати до першої множини, зберігаючи при цьому незалежність.

Матроїди застосовуються в різних галузях комп'ютерних наук, включаючи теорію графів, оптимізацію та розробку алгоритмів. Вони забезпечують основу для розв'язання задач комбінаторної оптимізації і можуть бути використані для розробки ефективних алгоритмів з доведеними гарантіями.