**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.

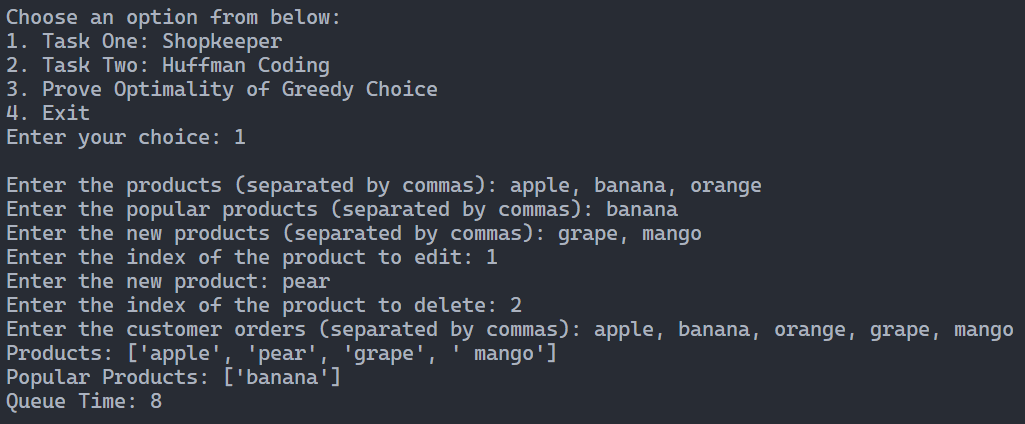
- Довести, що жадібний вибір для розглядаємої задачі є оптимальним рішенням або принаймні є частиною деякого оптимального рішення.

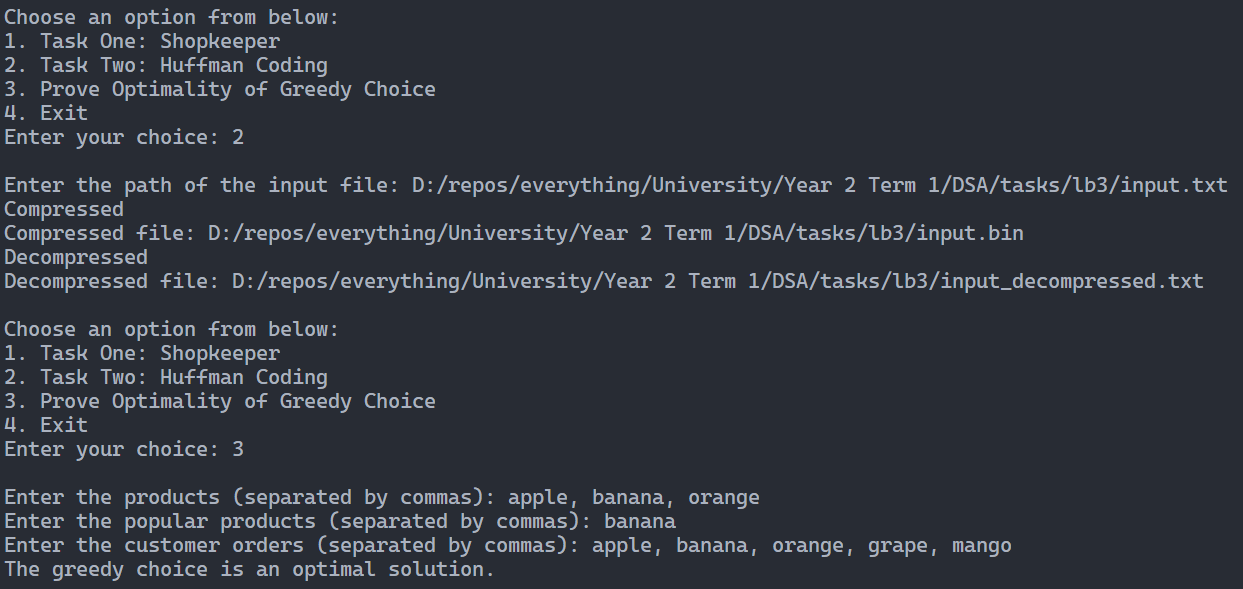
- Розробити програмне забезпечення, яке реалізує використання алгоритму Хаффмана для стискання даних текстового файлу у вигляді класу. Клас повинен мати методи, які дозволяють задати файл з даними, виконати стискання даних, визначити параметри виконаного стискання та зворотне перетворення, записати результати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





**Код**

*# 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.