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:

ст. гр. КНТ-113сп Ярослав ПАНЧЕНКО

Прийняв:

ст.викл. Лариса ДЕЙНЕГА

2023

# Мета роботи:

## Вивчити основні принципи та особливості жадібнихалгоритмів.

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

# Завдання до лабораторної роботи:

## Ознайомитися з літературою та основними теоретичними відомостями за темою роботи.

## Розробити програмне забезпечення, що розв’язує задачу у відповідності з індивідуальним завданням з пункту 3.3.2.3 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; – коригувати та видаляти існуючі; – розв’язувати задачу з використанням жадібного алгоритму.

## Індивідуальне завдання обрати у відповідності з номером варіанта на основі наступної формули: ((N – 1) % 5) + 1, де N – номер варіанта, що використовувався в попередніх лабораторних роботах.

## 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.

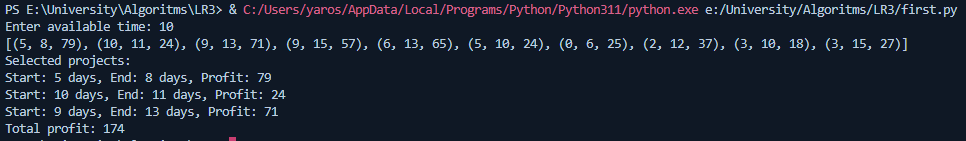
# Варіант 16: 1

Варіант №1

Інвестор оголосив конкурс на фінансування проєктів протягом заданого періоду часу. Після подачі заявок і попереднього відбору було сформовано відповідний пул заявок, кожну з яких визначено датою початку, датою завершення та прибутком, який даний проєкт принесе після виконання. Оцінивши надані заявки та власні ресурси, інвестор зрозумів, що зможе фінансувати не більше ніж один проєкт одночасно. Під час конкурсного відбору інвестор поставив задачу вибрати для виконання набір проєктів, які принесуть йому максимальну вигоду.

Задача 2: 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.

# Хід виконання самостійної роботи:



import random

class ProjectInvestor:

    def \_\_init\_\_(self, numProjects, availableTime):

        self.numProjects = numProjects

        self.availableTime = availableTime

        self.projects = [self.generateRandomProject() for \_ in range(numProjects)]

    def generateRandomProject(self):

        start = random.randint(0, 10)

        end = random.randint(start + 1, 15)

        profit = random.randint(10, 100)

        return (start, end, profit)

    def maximizeProfit(self):

        self.projects.sort(key=lambda x: x[2] / (x[1] - x[0]), reverse=True)

        print(self.projects)

        totalTime = 0

        totalProfit = 0

        selectedProjects = []

        for project in self.projects:

            start, end, profit = project

            if totalTime + (end - start) <= self.availableTime:

                totalTime += (end - start)

                totalProfit += profit

                selectedProjects.append(project)

        return selectedProjects, totalProfit

numProjects = 10

availableTime = int(input("Enter available time: "))

investor = ProjectInvestor(numProjects, availableTime)

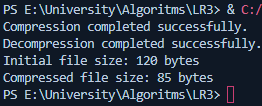
selectedProjects, totalProfit = investor.maximizeProfit()

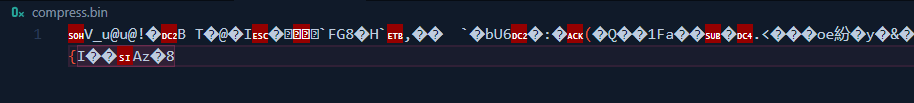
print("Selected projects:")

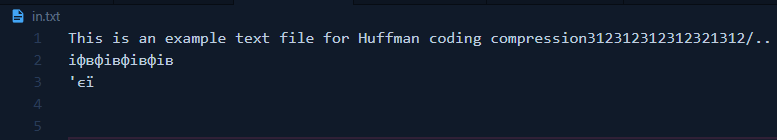
for project in selectedProjects:

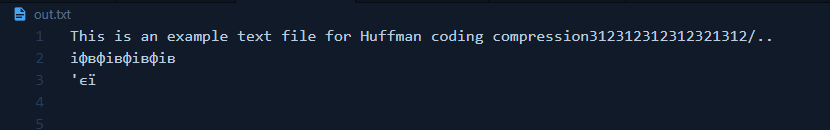
    print(f"Start: {project[0]} days, End: {project[1]} days, Profit: {project[2]}")

print(f"Total profit: {totalProfit}")









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 makeFrequencyDict(self, text):

        frequency = {}

        for char in text:

            if char in frequency:

                frequency[char] += 1

            else:

                frequency[char] = 1

        return frequency

    def makeHeap(self, frequency):

        for key in frequency:

            node = self.HeapNode(key, frequency[key])

            heapq.heappush(self.heap, node)

    def buildHuffmanTree(self):

        root = self.HeapNode(None, 0)

        root.left = self.heap[0]

        self.mergeNodes()

        return root

    def mergeNodes(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 buildHuffmanCodes(self, node, current\_code):

        if node is None:

            return

        if node.char is not None:

            self.codes[node.char] = current\_code

            self.reverse\_mapping[current\_code] = node.char

        self.buildHuffmanCodes(node.left, current\_code + "0")

        self.buildHuffmanCodes(node.right, current\_code + "1")

    def getEncodedText(self, text):

        encoded\_text = ""

        for char in text:

            encoded\_text += self.codes[char]

        return encoded\_text

    def padEncodedText(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 getByteArray(self, padded\_encoded\_text):

        if len(padded\_encoded\_text) % 8 != 0:

            raise ValueError("Encoded text is not properly padded")

        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, text, output\_file):

        frequency = self.makeFrequencyDict(text)

        self.makeHeap(frequency)

        self.mergeNodes()

        root = self.buildHuffmanTree()

        self.buildHuffmanCodes(root, "")

        encoded\_text = self.getEncodedText(text)

        padded\_encoded\_text = self.padEncodedText(encoded\_text)

        byte\_array = self.getByteArray(padded\_encoded\_text)

        with open(output\_file, "wb") as f:

            f.write(bytes(byte\_array))

        print("Compression completed successfully.")

    def decompress(self, input\_file, output\_file):

        with open(input\_file, "rb") as f:

            bit\_string = ""

            byte = f.read(1)

            while byte:

                byte = ord(byte)

                bits = bin(byte)[2:].rjust(8, '0')

                bit\_string += bits

                byte = f.read(1)

            extra\_padding = int(bit\_string[:8], 2)

            bit\_string = bit\_string[8:]

            encoded\_text = bit\_string[:-1 \* extra\_padding]

            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 = ""

        with open(output\_file, "w") as f:

            f.write(decoded\_text)

        print("Decompression completed successfully.")

if \_\_name\_\_ == "\_\_main\_\_":

    huffman = HuffmanCoding()

    input\_file = "in.txt"

    output\_file = "out.txt"

    compressed\_file = "compress.bin"

*# Read the input text file for compression*

    with open(input\_file, "r") as file:

        text = file.read()

*# Compress and save the compressed file*

    huffman.compress(text, compressed\_file)

*# Decompress the compressed file and save the resulting file*

    huffman.decompress(compressed\_file, output\_file)

*# Determine the file size before and after compression*

    initial\_size = os.path.getsize(input\_file)

    compressed\_size = os.path.getsize(compressed\_file)

    print(f"Initial file size: {initial\_size} bytes")

    print(f"Compressed file size: {compressed\_size} bytes")

# 4 Висновки:

Я вивчив основні принципи та особливості жадібних алгоритмів. Навчився використовувати жадібні алгоритми для розв’язання практичних завдань та обґрунтовувати прийняті рішення.