САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №7

по курсу «Алгоритмы и структуры данных»

Тема: последняявосеннем семестре

Вариант 16

Выполнил:
Субагио Сатрио
К3140

Проверила:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Содержание отчета

Задачи

Задача №1. Обмен монет

Текст задачи.

Как мы уже поняли из лекции, не всегда "жадное"решение задачи на обмен монет работает корректно для разных наборов номиналов монет. Например, если доступны номиналы 1, 3 и 4, жадный алгоритм поменяет 6 центов, используя три монеты (4 + 1 + 1), в то время как его можно изменить, используя всего две монеты (3 + 3). Теперь ваша цель - применить динамическое программирование для решения задачи про обмен монет для разных номиналов.

Листинг кода.

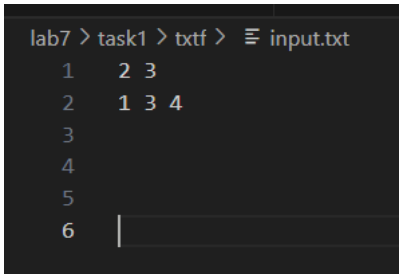
```
import sys
import os

base_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), '../../'))
sys.path.append(base_dir)

from utils import read_money_from_file, write_array_to_file,
measure_performance

def coin_exchange(money, k, coins, coin_limit=None):
    dp = [float('inf')] * (money + 1)
    dp[0] = 0

    if coin_limit:
        for i in range(1, money + 1):
            for coin, limit in zip(coins, coin_limit):
                for count in range(1, limit + 1):
                    if i >= coin * count:
                        dp[i] = min(dp[i], dp[i - coin * count] + count)
    else:
        for i in range(1, money + 1):
            for coin in coins:
                if i >= coin:
                    dp[i] = min(dp[i], dp[i - coin] + 1)

    return dp[money] if dp[money] != float('inf') else -1

def process_file(input_file_path, output_file_path):
    money, k, coins, coin_limit = read_money_from_file(input_file_path)

    result = coin_exchange(money, k, coins, coin_limit)

    write_array_to_file(output_file_path, [result])
```

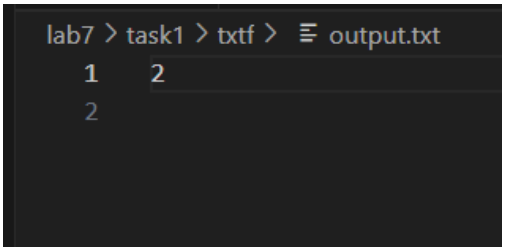
```
def main():
    script_dir = os.path.dirname(__file__)
    base_dir = os.path.abspath(os.path.join(script_dir, '..', '..', 'task1'))
    input_file_path = os.path.join(base_dir, 'txtf', 'input.txt')
    output_file_path = os.path.join(base_dir, 'txtf', 'output.txt')
    measure_performance(process_file, input_file_path, output_file_path)

if __name__ == "__main__":
    main()
```

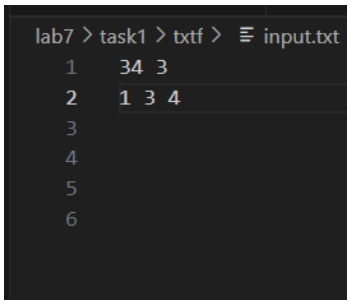
Текстовое объяснение решения.

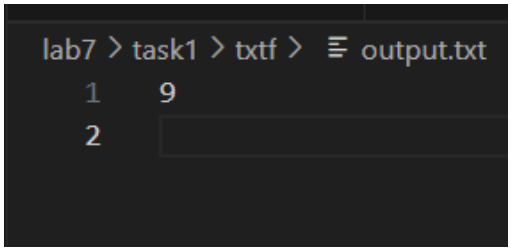
Функция `coin_exchange` использует динамическое программирование для нахождения минимального числа монет, необходимых для составления нужной суммы, учитывая два случая: без ограничения на количество монет каждого типа и с ограничением на их количество. Если ограничение применяется, алгоритм оценивает все возможные комбинации монет с учетом доступных ограничений. Функция `process_file` читает данные из входного файла, который содержит сумму денег, количество типов монет и список монет с ограничениями на их количество, затем вычисляет результат с помощью функции `coin_exchange` и записывает его в выходной файл. Функция `main` задает пути для входного и выходного файлов и измеряет производительность обработки файла с использованием функции `measure_performance`. Программа предназначена для решения задачи с использованием ввода и вывода через файлы.

Результат работы кода на примерах из текста задачи:









	Время выполнения	Затраты памяти
Пример из задачи 1	0.001015 секунд	2324 байт
Пример из задачи 2	0.000945 секунд	2324 байт

Вывод по задаче:
Применение динамического программирования для решения задачи обмена монет разного номинала. Результат проверки отображается правильно.

Задача №2. Примитивный калькулятор

Текст задачи.
Дан примитивный калькулятор, который может выполнять следующие три операции с текущим числом x: умножить x на 2, умножить x на 3 или прибавить 1 к x. Дано положительное целое число n, найдите минимальное количество операций, необходимых для получения числа n, начиная с числа 1.

Листинг кода.

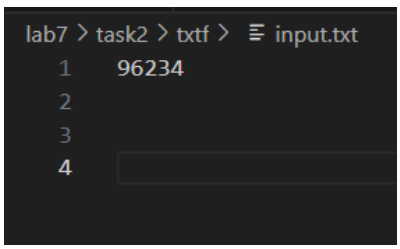
```
import sys
import os

base_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), '../../'))
sys.path.append(base_dir)

from utils import read_integers_from_file, write_array_to_file,
measure_performance

def optimal_sequence(n):
    operations = []
    while n > 1:
        operations.append(n)
        if n % 3 == 0:
            n = n // 3
        elif n % 2 == 0:
            n = n // 2
        else:
            n = n - 1
    operations.append(1)
    return list(reversed(operations))
```

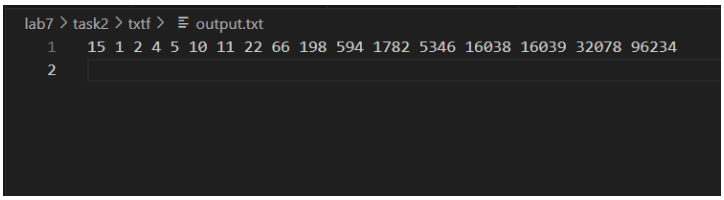
```python
def process_file(input_file_path, output_file_path):
    n = read_integers_from_file(input_file_path)[0]

    sequence = optimal_sequence(n)
    write_array_to_file(output_file_path, [len(sequence) - 1] + sequence)

def main():
    script_dir = os.path.dirname(__file__)
    base_dir = os.path.abspath(os.path.join(script_dir, '..', '..', 'task2'))
    input_file_path = os.path.join(base_dir, 'txtf', 'input.txt')
    output_file_path = os.path.join(base_dir, 'txtf', 'output.txt')
    measure_performance(process_file, input_file_path, output_file_path)

if __name__ == "__main__":
    main()
```

Текстовое объяснение решения.

Этот код решает задачу нахождения оптимальной последовательности операций для приведения числа `n` к единице с использованием операций деления на 3, деления на 2 или вычитания единицы. Функция `optimal_sequence` генерирует последовательность операций, начиная с числа `n` и повторяя операции, пока не дойдем до единицы. Для каждого числа выбирается операция в зависимости от того, делится ли оно на 3 или 2, иначе выполняется вычитание единицы. Результирующая последовательность операций сохраняется в список и возвращается в обратном порядке. Функция `process_file` считывает входное число из файла, затем вызывает функцию `optimal_sequence` для нахождения последовательности и записывает результат в выходной файл. Функция `main` задает пути для входного и выходного файлов и измеряет производительность выполнения задачи с использованием функции `measure_performance`. Код решает задачу с использованием файлов для ввода и вывода данных

Результат работы кода на примерах из текста задачи:





	Время выполнения	Затраты памяти
Пример из задачи	0.000971 секунд	1804 байт

Вывод по задаче:

Программа примитивного калькулятора проверяется в соответствии с заданием. Результат проверки отображается корректно.

Задача №6. Планировщик заданий

Текст задачи.

Дана последовательность, требуется найти ее наибольшую возрастающую подпоследовательность.

Листинг кода.

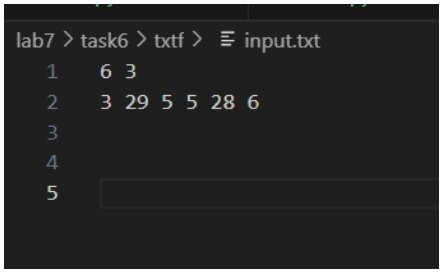
```
import sys
import os

base_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), '../../'))
sys.path.append(base_dir)

from utils import read_integers_space_from_file, write_array_to_file,
measure_performance

def longest_increasing_subsequence(sequence):
    n = len(sequence)
    dp = [1] * n
    prev = [-1] * n

    for i in range(1, n):
        for j in range(i):
            if sequence[i] > sequence[j] and dp[i] < dp[j] + 1:
                dp[i] = dp[j] + 1
                prev[i] = j

    max_length = max(dp)
    index = dp.index(max_length)

    subsequence = []
    while index != -1:
        subsequence.append(sequence[index])
        index = prev[index]

    subsequence.reverse()
    return max_length, subsequence

def process_file(input_file_path, output_file_path):
    n, sequence = read_integers_space_from_file(input_file_path)
```

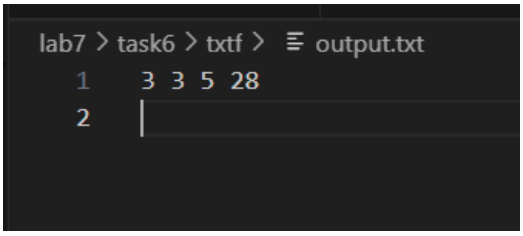
```
    if sequence:
        length, subsequence = longest_increasing_subsequence(sequence)
        write_array_to_file(output_file_path, [length] + subsequence)
    else:
        write_array_to_file(output_file_path, [0])


def main():
    script_dir = os.path.dirname(__file__)
    base_dir = os.path.abspath(os.path.join(script_dir, '..', '..', 'task6'))
    input_file_path = os.path.join(base_dir, 'txtf', 'input.txt')
    output_file_path = os.path.join(base_dir, 'txtf', 'output.txt')
    measure_performance(process_file, input_file_path, output_file_path)

if __name__ == "__main__":
    main()
```

Текстовое объяснение решения.

Функция `longest_increasing_subsequence` использует динамическое программирование для поиска длины самой длинной возрастающей подпоследовательности. Для этого она поддерживает два списка: `dp`, который хранит длину самой длинной возрастающей подпоследовательности, заканчивающейся в элементе, и `prev`, который хранит индекс предыдущего элемента в последовательности для восстановления самой подпоследовательности. Алгоритм проходит по всем парам элементов, проверяя, можно ли удлинить подпоследовательность. После вычисления длины самой длинной возрастающей подпоследовательности, функция восстанавливает саму последовательность, используя массив `prev`. Функция `process_file` читает входные данные из файла, вызывает `longest_increasing_subsequence` для получения результата, и записывает его в выходной файл. В функции `main` задаются пути для входного и выходного файлов, а также измеряется производительность работы с помощью функции `measure_performance`. Этот код решает задачу с использованием файлов для ввода и вывода данных, позволяя легко интегрировать его в более широкий проект.

Результат работы кода на примерах из текста задачи:





	Время выполнения	Затраты памяти
Нижняя граница диапазона значений	0.00782 секунд	1080 байт
Пример из задачи	0.001279 секунд	1804 байт
Верхняя граница диапазона значений входных данных из текста задачи	1.10294 секунд	1991 байт

Вывод по задаче:
Создание программы, сортирующей последовательность от наибольшей, было протестировано в соответствии с заданием. Результаты тестирования показаны правильно.

Задача №6. Шаблоны

Текст задачи.
Многие операционные системы используют шаблоны для ссылки на группы объектов: файлов, пользователей, и т. д. Ваша задача – реализовать простейший алгоритм проверки шаблонов для имен файлов.

В этой задаче алфавит состоит из маленьких букв английского алфавита и точки («.»). Шаблоны могут содержать произвольные символы алфавита, а также два специальных символа: «?» и «*». Знак вопроса («?») соответствует ровно одному произвольному символу. Звездочка «+» соответствует подстроке произвольной длины (возможно, нулевой). Символы алфавита, встречающиеся в шаблоне, отображаютсянаровноодинтакойжесимволвпроверяемойстрочке.Строкасч итается подходящей под шаблон, если символы шаблона можно последовательно отобразить на символы строки таким образом, как описано выше. Например, строчки «ab», «aab» и «beda.» подходят под шаблон «*a?», а строчки «bebe», «a» и «ba»
–нет.

Листинг кода.

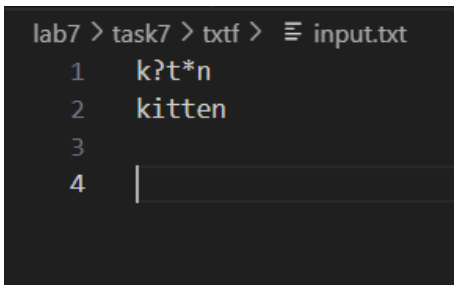
```
import sys
import os
import re

base_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), '../../'))
sys.path.append(base_dir)

from utils import read_strings_from_file, write_output_to_file2,
measure_performance

def is_match(pattern, string):
    regex = re.compile('^' + pattern.replace('?', '.').replace('*', '.*') +
'$')
    return regex.match(string) is not None

def process_file(input_file_path, output_file_path):
    pattern, string = read_strings_from_file(input_file_path)
    result = "YES" if is_match(pattern, string) else "NO"
    write_output_to_file2(output_file_path, result)

def main():
    script_dir = os.path.dirname(__file__)
    base_dir = os.path.abspath(os.path.join(script_dir, '..', '..', 'task7'))
    input_file_path = os.path.join(base_dir, 'txtf', 'input.txt')
    output_file_path = os.path.join(base_dir, 'txtf', 'output.txt')
    measure_performance(process_file, input_file_path, output_file_path)

if __name__ == "__main__":
    main()
```

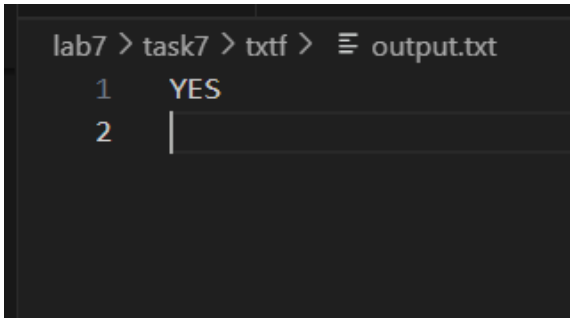
Текстовое объяснение решения.

Этот код решает задачу, которая включает проверку соответствия строки шаблону, где в шаблоне используются символы `?` (один любой символ) и `*` (любое количество символов, включая ноль). Функция `is_match` преобразует шаблон в регулярное выражение, где `?` заменяется на `.` (один любой символ), а `*` заменяется на `.*` (любое количество символов). Затем она пытается найти совпадение с данной строкой с помощью регулярного выражения. Если совпадение найдено, функция возвращает "YES", в противном случае — "NO". В функции `process_file` происходит чтение входных данных (шаблон и строка) из файла, проверка соответствия с помощью функции `is_match`, и запись результата в выходной файл. В функции `main` задаются пути для входных и выходных файлов, а также измеряется производительность работы с помощью функции

`measure_performance`. Это позволяет эффективно решить задачу с использованием файлов для ввода и вывода данных.

Результат работы кода на примерах из текста задачи:





	Время выполнения	Затраты памяти
Пример из задачи	0.001995 секунд	2645 байт

Вывод по задаче:

Создание программы, распознающей шаблоны символов в операционной системе, было протестировано в соответствии с заданием. Результаты тестирования отображаются корректно.

Вывод

Вывод из Практикума № 7 - Решения задач с использованием динамического программирования и алгоритмов оптимизации
Этот практикум включает в себя решение нескольких классических задач с использованием динамического программирования и других алгоритмов оптимизации, таких как рекурсия и бэктрекинг. Задачи охватывают широкий спектр проблем, начиная от обмена монет и вычисления наибольшей общей подпоследовательности до работы с шаблонами и редакционным расстоянием. Каждое решение сопровождается алгоритмом для поиска оптимального решения, включая работу с файлами для ввода и вывода данных, что углубляет знания в области эффективного решения задач с большими объемами данных.