САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИТМО
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №7
по курсу «Алгоритмы и структуры данных»
Тема: Динамическое программирование №1
Вариант 17

Выполнил:
Останин А. С.
К3140

Проверил:
Афанасьев А. В.

Санкт-Петербург
2024 г.

Содержание отчета

Задачи по варианту

Задача №1. Множество

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.

```
import os
import sys

current_dir = os.path.dirname(__file__)
parent_dir = os.path.abspath(os.path.join(current_dir, '..', '..', '..'))
sys.path.insert(0, parent_dir)

from lab7.utils import read_data, write_data


def money_exchange(money: int, coins: list):
    """
    :param money: сумма, которую нужно набрать
    :param coins: список доступных номиналов монет
    :return: Минимальное количество монет, необходимое для набора суммы
money
    """
    MinNumCoins = [0] + [float('inf')] * money  # массив для хранения
минимального количества монет для каждой суммы

    for m in range(1, money + 1):
        for coin in coins:  # проверяем каждую монету
            if m >= coin:  # если текущая сумма больше или равна номиналу
монеты
                NumCoins = MinNumCoins[m - coin] + 1
                MinNumCoins[m] = min(NumCoins, MinNumCoins[m])

    return MinNumCoins[money]


def task1():
    PATH_INPUT = 'lab7/task1/txtf/input.txt'
    PATH_OUTPUT = 'lab7/task1/txtf/output.txt'

    origin_input = read_data(PATH_INPUT)

    data = read_data(PATH_INPUT)
    money_ = int(data[0].split()[0])
    coins_ = [int(i) for i in data[1].split()]

    result = money_exchange(money_, coins_)
    write_data(PATH_OUTPUT, str(result))
    print(origin_input)
    print(result)
if __name__ == "__main__":
    task1()
```

Сначала создаётся массив MinNumCoins длиной money + 1, где каждый элемент представляет минимальное количество монет, необходимое для набора соответствующей суммы. После инициализации массива выполняется его заполнение. Для каждой суммы m от 1 до money проверяются все доступные номиналы монет из списка coins. Если текущая сумма m больше или равна номиналу монеты coin, то вычисляется минимальное количество монет, необходимое для набора суммы m, через размен суммы m – coin. Затем минимальное количество монет для суммы m обновляется следующим образом: MinNumCoins[m] = min(NumCoins, MinNumCoins[m]. Таким образом, на каждой итерации массива проверяются все возможные варианты использования монет для минимизации числа монет, необходимых для набора текущей суммы.

```
Input:
2 3
1 3 4
Output:
2
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи		
Пример из задачи	9.550002869218588e-05 s	0.0004425048828125 Mb
Верхняя граница диапазона значений входных данных из текста задачи		

Вывод по задаче: решена задача, используя динамическое программирование

Задача №2. Редакционное расстояние

Редакционное расстояние между двумя строками – это минимальное количество операций (вставки, удаления и замены символов) для преобразования одной строки в другую. Это мера сходства двух строк. У редакционного расстояния есть применения, например, в вычислительной

биологии, обработке текстов на естественном языке и проверке орфографии. Ваша цель в этой задаче – вычислить расстояние редактирования между двумя строками.

```
import os
import sys

current_dir = os.path.dirname(__file__)
parent_dir = os.path.abspath(os.path.join(current_dir, '..', '..', '..'))
sys.path.insert(0, parent_dir)

from lab7.utils import read_data, write_data


def distance(s1: str, s2: str) -> int:
    s1_len = len(s1)
    s2_len = len(s2)

    D = [[0] * (s2_len + 1) for _ in range(s1_len + 1)]  # создаём
двумерный массив

    # Базовые случаи
    for i in range(s1_len + 1):
        D[i][0] = i
    for j in range(s2_len + 1):
        D[0][j] = j

    # Заполнение таблицы
    for i in range(1, s1_len + 1):
        for j in range(1, s2_len + 1):
            if s1[i - 1] == s2[j - 1]:
                D[i][j] = D[i - 1][j - 1]  # берётся значение из
диагональной ячейки
            else:
                D[i][j] = 1 + min(D[i - 1][j], D[i][j - 1], D[i - 1][j -
1])  # min(удаление, вставка, замена)

    return D[-1][-1]


def task2():
    PATH_INPUT = 'lab7/task3/txtf/input.txt'
    PATH_OUTPUT = 'lab7/task3/txtf/output.txt'

    origin_input = read_data(PATH_INPUT)

    data_ = read_data(PATH_INPUT)
    word_1 = data_[0]
    word_2 = data_[1]

    result = distance(word_1, word_2)

    write_data(PATH_OUTPUT, str(result))
    print(origin_input)
    print(result)


if __name__ == "__main__":
    task2()
```

Алгоритм вычисления редакционного расстояния между двумя строками реализован с использованием динамического программирования. Он основывается на построении двумерного массива D, где элемент D[i][j] представляет минимальное количество операций (вставки, удаления или замены символов), необходимых для преобразования первых i символов строки s1 в первые j символов строки s2. Создаётся двумерный массив D, где сначала заполняются базовые случаи:
Базовые случаи задаются следующим образом:

- D[i][0]=i, так как для преобразования первых i символов s1 в пустую строку требуется i операций удаления.
- D[0][j]=j, так как для преобразования пустой строки в первые j символов s2 требуется j операций вставки.

Далее заполняется таблица:

Для каждой пары индексов i и, начиная с 1:

- Если символы s1[i−1] и s2[j−1] совпадают, то D[i][j] = D[i-1][j-1] (значение берётся из диагональной ячейки, так как операция не требуется).
- Если символы не совпадают, то минимальное количество операций вычисляется как: D[i][j] = 1 + min(D[i-1][j], D[i][j-1], D[i-1][j-1}), где:

D[i−1][j] – операция удаления,

D[i][j−1] — операция вставки,

D[i−1][j−1] — операция замены.

После заполнения всей таблицы итоговое значение D[len(s1)][len(s2)] представляет минимальное количество операций для преобразования строки s1 в строку s2.

```
Input:
editing
distance

Output:
5
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи		
Пример из задачи	9.549997048452497e-05 s	0.0004425048828125 mb
Верхняя граница диапазона значений входных данных из текста задачи		

Вывод по задаче: решена задача, используя динамическое программирование

Вывод

Решены задачи, используя динамическое программирование