

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: “Динамическое программирование”
Вариант 20

Выполнила:
Толстухина К.А.
К3139

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 год

Оглавление

Задачи по варианту	3
Задача №1. Обмен монет	3
Задача №6. Наибольшая возрастающая последовательность	5
Дополнительные задачи	9
Задача №4. Наибольшая общая подпоследовательность двух последовательностей	9
Задача №5. наибольшая общая подпоследовательность трех последовательностей	11

Задачи по варианту

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 tracemalloc
import time
from lab7.utils import *

t_start = time.perf_counter()
tracemalloc.start()

def min_coins(money: int, coins: list[int]) -> float:
    """
        подсчет минимального количества монет с
        использованием дп
    :param money: int
    :param coins: list[int]
    :return: float
    """
    list_with_count_for_coins = [float('inf')] * (money + 1)
    list_with_count_for_coins[0] = 0
    for coin in coins:
        for i in range(coin, money + 1):
```

```

        list_with_count_for_coins[i] =
min(list_with_count_for_coins[i],
list_with_count_for_coins[i - coin] + 1)

    return list_with_count_for_coins[money]

CURRENT_DIR =
os.path.dirname(os.path.abspath(__file__))
TXTF_DIR =
os.path.join(os.path.dirname(CURRENT_DIR), "txtf")
INPUT_PATH = os.path.join(TXTF_DIR, "input.txt")
OUTPUT_PATH = os.path.join(TXTF_DIR, "output.txt")

if __name__ == "__main__":
    lines = open_file(INPUT_PATH)
    money, k = map(int, lines[0].strip().split())
    coins = list(map(int, lines[1].split()))
    if 1 <= money <= 10 ** 3 and 1 <= k <= 100:
        result = min_coins(money, coins)
        write_file(str(result), OUTPUT_PATH)
    else:
        print("Введите корректные данные")

        print("Время работы: %s секунд" %
(time.perf_counter() - t_start))
        print("Затрачено памяти:",
tracemalloc.get_traced_memory()[1], "байт")
    tracemalloc.stop()

```

Текстовое объяснение

Подключаю две библиотеки для отслеживания памяти и времени. Идея алгоритма заключается в том, чтобы постепенно строить решение для всех возможных сумм от 0 до money, используя уже вычисленные минимальные количества монет для меньших сумм. Далее вне функции я прописываю пути к необходимым

директориям/файлам. И в основной части, я считываю данные, проверяю их валидность, вызываю функцию и вывожу время и память.

Пример работы(скрин файлов)

input.txt			:	output.txt		
1	34 3	✓		1	9	
2	1 3 4					

	Время	Память
пример	0.0014254160050768405 секунд	14685 байт

Вывод: был реализован алгоритм с использованием динамического программирования.

Задача №6. Наибольшая возрастающая последовательность

Текст задачи

Дана последовательность, требуется найти ее наибольшую возрастающую подпоследовательность.

Листинг кода

```
import tracemalloc
import time
from lab7.utils import *

t_start = time.perf_counter()
tracemalloc.start()
```

```

def max_increasing_subsequence(numbers: list[int])
-> tuple:
    """
    функция для нахождения максимальной возрастающей
    подпоследовательности
    :param numbers: list[int]
    :return: tuple
    """
    n = len(numbers)
    if n == 0:
        return 0

    counts = [1] * n
    pred = [-1] * n
    for i in range(1, n):
        for j in range(i):
            if numbers[j] < numbers[i]:
                counts[i] = max(counts[i], counts[j]
+ 1)
                pred[i] = j

    max_len = max(counts)
    index = counts.index(max_len)
    lis = []
    while index != -1:
        lis.append(numbers[index])
        index = pred[index]
    lis.reverse()

    return max(counts), lis

CURRENT_DIR =
os.path.dirname(os.path.abspath(__file__))
TXTF_DIR =
os.path.join(os.path.dirname(CURRENT_DIR), "txtf")

```

```

INPUT_PATH = os.path.join(TXTF_DIR, "input.txt")
OUTPUT_PATH = os.path.join(TXTF_DIR, "output.txt")

if __name__ == "__main__":
    lines = open_file(INPUT_PATH)
    n = int(lines[0].strip())
    numbers = list(map(int, lines[1].split()))
    if 1 <= n <= 300000:
        result = max_increasing_subsequence(numbers)
        write_file('\n'.join([str(result[0]), '
'.join(map(str, result[1]))]), OUTPUT_PATH)
    else:
        print("Введите корректные данные")

    print("Время работы: %s секунд" %
(time.perf_counter() - t_start))
    print("Затрачено памяти:",
tracemalloc.get_traced_memory()[1], "байт")
    tracemalloc.stop()

```

Текстовое объяснение

Работа с памятью, временем и файлами аналогична предыдущей задаче. Сначала создаю массив длиной n , заполняя его единицами. Для каждого элемента в массиве (начиная с 1-го индекса) мы сравниваем его с каждым предыдущим элементом: если элемент на позиции i больше элемента на позиции j , то мы обновляем массив как $\max(\text{counts}[i], \text{counts}[j] + 1)$, что означает продолжение возрастающей последовательности. После завершения обработки всех элементов, максимальное значение в массиве dp будет длиной самой длинной возрастающей подпоследовательности. Массив $pred$ помогает отслеживать, какой элемент предшествует текущему. Это нужно для восстановления самой последовательности в конце

Пример работы(скрин файлов)

input.txt ×			:	output.txt ×		
1	6	✓		1	3	
2	3 29 5 5 28 6			2	3 5 28	

	Время	Память
ПРИМЕР	0.002382083999691531 секунд	14690 байт

Вывод: я практиковала работу с дп

Дополнительные задачи

Задача №4. Наибольшая общая подпоследовательность двух последовательностей

Текст задачи

Даны две последовательности $A = (a_1, a_2, \dots, a_n)$ и $B = (b_1, b_2, \dots, b_m)$, найти длину их самой длинной общей подпоследовательности, т.е. наибольшее неотрицательное целое число p такое, что существуют индексы $1 \leq i_1 < i_2 < \dots < i_p \leq n$ и $1 \leq j_1 < j_2 < \dots < j_p \leq m$ такие, что $a_{i_1} = b_{j_1}, \dots, a_{i_p} = b_{j_p}$.

Листинг кода

```
import tracemalloc
import time
from lab7.utils import *

t_start = time.perf_counter()
tracemalloc.start()

def max_increasing_subsequence(number1: list[int],
                                number2: list[int]) -> int:
    """
    находит максимальную длину общей
    подпоследовательности двух последовательностей
    :param number1: list[int]
    :param number2: list[int]
    :return: int
    """
    counts = [[0] * (len(number2) + 1) for _ in
               range(len(number1) + 1)]

    for i in range(1, len(number1) + 1):
        for j in range(1, len(number2) + 1):
            if number1[i - 1] == number2[j - 1]:
                counts[i][j] = counts[i - 1][j - 1] + 1
```

```

        else:
            counts[i][j] = max(counts[i][j - 1],
counts[i - 1][j])
        return counts[len(number1)][len(number2)]

CURRENT_DIR =
os.path.dirname(os.path.abspath(__file__))
TXTF_DIR =
os.path.join(os.path.dirname(CURRENT_DIR), "txtf")
INPUT_PATH = os.path.join(TXTF_DIR, "input.txt")
OUTPUT_PATH = os.path.join(TXTF_DIR, "output.txt")

if __name__ == "__main__":
    lines = open_file(INPUT_PATH)
    n = int(lines[0].strip())
    numbers1 = list(map(int, lines[1].split()))
    m = int(lines[2].strip())
    numbers2 = list(map(int, lines[3].split()))
    if 1 <= n <= 100 and 1 <= m <= 100:
        result = max_increasing_subsequence(numbers1,
numbers2)
        write_file(str(result), OUTPUT_PATH)
    else:
        print("Введите корректные данные")

    print("Время работы: %s секунд" %
(time.perf_counter() - t_start))
    print("Затрачено памяти:",
tracemalloc.get_traced_memory()[1], "байт")
    tracemalloc.stop()

```

Текстовое объяснение

Я создаю таблицу, где будет храниться длина общей подпоследовательности для первых i символов строки `numbers1` и первых j символов строки `numbers2`. Если символы текущих

позиций в строках совпадают ($\text{numbers1}[i-1] == \text{numbers2}[j-1]$), то я увеличиваю длину, добавив 1 к значению $\text{counts}[i-1][j-1]$. Если символы не совпадают, то я выбираю максимальное значение из двух возможных вариантов: $\text{counts}[i-1][j]$ и $\text{counts}[i][j-1]$

Тесты(скрины файлов)

≡ input.txt ×		:	≡ output.txt ×	
1	4	✓	1	2
2	2 7 8 3			
3	4			
4	5 2 8 7			

	время	память
тест 1	0.0007209999894257 635 секунд	14856 байт

Вывод: была реализована работа с двумя последовательностями

Задача №5. наибольшая общая подпоследовательность трех последовательностей

Текст задачи

Вычислить длину самой длинной общей подпоследовательности из трех последовательностей.

Листинг кода

```
import tracemalloc
import time
from lab7.utils import *

t_start = time.perf_counter()
tracemalloc.start()

def max_increasing_subsequence(number1: list[int],
number2: list[int], number3: list[int]) -> int:
    """
    находит максимальную длину общей
    подпоследовательности трех последовательностей
    :param number1: list[int]
    :param number2: list[int]
    :param number3: list[int]
    :return: int
    """
    counts = [[[0] * (len(number3) + 1) for _ in
range(len(number2) + 1)] for _ in range(len(number1)
+ 1)]

    for i in range(1, len(number1) + 1):
        for j in range(1, len(number2) + 1):
            for k in range(1, len(number3) + 1):
                if number1[i - 1] == number2[j - 1]
== number3[k - 1]:
                    counts[i][j][k] = counts[i - 1][j
- 1][k - 1] + 1
                else:
                    counts[i][j][k] =
max(counts[i][j][k - 1], counts[i][j - 1][k],
counts[i - 1][j][k])
    return
counts[len(number1)][len(number2)][len(number3)]
```

```

CURRENT_DIR =
os.path.dirname(os.path.abspath(__file__))
TXTF_DIR =
os.path.join(os.path.dirname(CURRENT_DIR), "txtf")
INPUT_PATH = os.path.join(TXTF_DIR, "input.txt")
OUTPUT_PATH = os.path.join(TXTF_DIR, "output.txt")

if __name__ == "__main__":
    lines = open_file(INPUT_PATH)
    n = int(lines[0].strip())
    numbers1 = list(map(int, lines[1].split()))
    m = int(lines[2].strip())
    numbers2 = list(map(int, lines[3].split()))
    k = int(lines[4].strip())
    numbers3 = list(map(int, lines[5].split()))
    if 1 <= n <= 100 and 1 <= m <= 100 and 1 <= k <=
100:
        result = max_increasing_subsequence(numbers1,
numbers2, numbers3)
        write_file(str(result), OUTPUT_PATH)
    else:
        print("Введите корректные данные")

    print("Время работы: %s секунд" %
(time.perf_counter() - t_start))
    print("Затрачено памяти:",
tracemalloc.get_traced_memory()[1], "байт")
    tracemalloc.stop()

```

Объяснение

Аналогично предыдущей задаче, только изначально создается трехмерная таблица и сравниваются три элемента.

Скрины (файлов)

input.txt ×		✓	output.txt ×	
1	5		1	3
2	8 3 2 1 7			
3	7			
4	8 2 1 3 8 10 7			
5	6			
6	6 8 3 1 4 7			

	время	память
тест 1	0.0018262499943375 587 секунд	15041 байт

Вывод по всей работе: я познакомилась с динамическим программированием и поработала с ним на практике.