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	0
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.

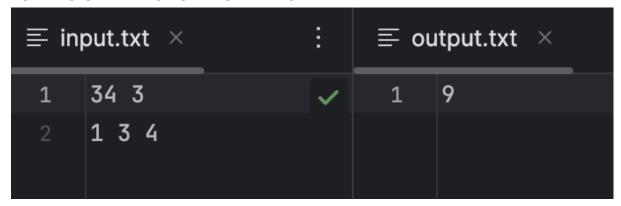
```
list with count for coints[i]
min(list with count for coints[i],
list with count for coints[i - coin] + 1)
   return list with count for coints[money]
CURRENT DIR
os.path.dirname(os.path.abspath( file ))
TXTF DIR
os.path.join(os.path.dirname(CURRENT DIR), "txtf")
INPUT PATH = os.path.join(TXTF DIR, "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 \le money \le 10 ** 3 and <math>1 \le k \le 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)



	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.

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

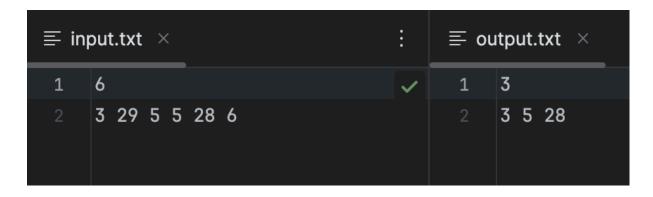
t_start = time.perf_counter()
tracemalloc.start()
```

```
def max increasing subsequence(numbers: list[int])
подпоследовательности
  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]:</pre>
               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())
   numders = list(map(int, lines[1].split()))
       result = max increasing subsequence(numders)
      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 п, заполняя его единицами. Для каждого элемента в массиве (начиная с 1-го индекса) мы сравниваем его с каждым предыдущим элементом: если элемент на позиции і больше элемента на позиции ј, то мы обновляем массив как max(counts[i], counts[j] + 1), что означает продолжение возрастающей последовательности. После завершения обработки всех элементов, максимальное значение в массиве ф будет длиной самой длинной возрастающей подпоследовательности. Массив ргеd помогает отслеживать, какой элемент предшествует текущему. Это нужно для восстановления самой последовательности в конце Пример работы(скрин файлов)



	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 = (a1,02,...,an) и B = (b1,02,...,bm), найти длину их самой длинной общей подпоследовательности, т.е. наибольшее неотрицатеьное целое число p такое, что существуют индексы $1 \le i1 < i2 < ... < ip$ In

11 ≤ j<j<... <jp ≤m такие, 4т0 ais = bj,..., ip = bp.

```
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
   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] +
```

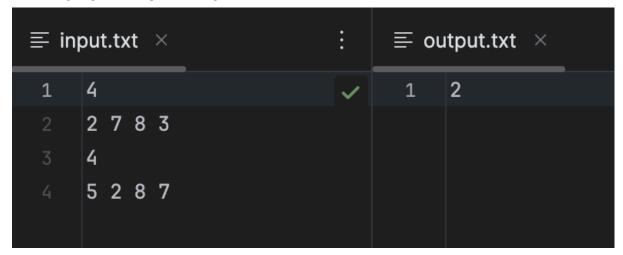
```
else:
               counts[i][j] = max(counts[i][j - 1],
counts[i - 1][j])
   return counts[len(number1)][len(number2)]
CURRENT DIR =
os.path.dirname(os.path.abspath( file ))
TXTF DIR =
os.path.join(os.path.dirname(CURRENT DIR), "txtf")
INPUT PATH = os.path.join(TXTF DIR, "input.txt")
OUTPUT PATH = os.path.join(TXTF_DIR, "output.txt")
if __name__ == " main <u>"</u>:
   lines = open file(INPUT PATH)
   n = int(lines[0].strip())
   numders1 = list(map(int, lines[1].split()))
   m = int(lines[2].strip())
   numders2 = list(map(int, lines[3].split()))
   if 1 \le n \le 100 and 1 \le m \le 100:
       result = max increasing subsequence(numders1,
numders2)
       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 і символов строки numbers1 и первых ј символов строки numbers2. Если символы текущих

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

Тесты(скрины файлов)



время	память
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.

```
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
   :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())
  numders1 = list(map(int, lines[1].split()))
  m = int(lines[2].strip())
  numders2 = list(map(int, lines[3].split()))
  k = int(lines[4].strip())
  numders3 = list(map(int, lines[5].split()))
100:
       result = max increasing subsequence (numders1,
numders2, numders3)
      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
      ✓
      1
      3

      3
      7
      ✓
      4
      8 2 1 3 8 10 7
      ✓
      6

      5
      6
      ✓
      6
      6 8 3 1 4 7
      ✓
      Image: Control of the control
```

	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.