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 1

Выполнила:

Аксянова А.Р

К3140

Проверил:

Афанасьев А.В.

Санкт-Петербург

2024 г.

# Содержание отчета

[Содержание отчета 2](#_Toc185028867)

[Задачи 3](#_Toc185028868)

[Задача №1. Обмен монет 3](#_Toc185028869)

[Задача №4. Наибольшая общая подпоследовательность двух по- следовательностей 4](#_Toc185028870)

[Задача №5. Наибольшая общая последовательность трех подпоследовательностей. 6](#_Toc185028871)

[Задача №6 . Наибольшая возрастающая последовательность. 7](#_Toc185028872)

[Результат работы всех задач 10](#_Toc185028873)

[Вывод 10](#_Toc185028874)

# Задачи

## Задача №1. Обмен монет

1. 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.

**Формат ввода / входного файла (input.txt).** Целое число money (1 ≤ money ≤ 103). Набор монет: количество возможных монет k и сам набор coins = {coin1, ..., coink}. 1 ≤ k ≤ 100, 1 ≤ coini ≤ 103. Проверку можно сделать на наборе {1,3,4}. Формат ввода: первая строка содержит через пробел money и k; вторая - coin1coin2...coink.

**Форматвывода/выходногофайла(output.txt).**Вывестиодночисло–ми- нимальное количество необходимых монет для размена money доступным набором монет coins.

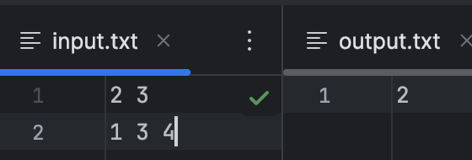
1. Листинг кода

import time  
import os  
from lab7.utils import inp, outp, caption  
  
CURRENT\_SCRIPT\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))  
os.chdir(CURRENT\_SCRIPT\_DIR)  
  
PATH\_INPUT = os.path.join('..', '..', 'files', 'input.txt')  
PATH\_OUTPUT = os.path.join('..', '..', 'files', 'output.txt')  
  
def min\_coins(money, coins):  
 l = [float('inf')] \* (money + 1)  
 l[0] = 0  
  
 for coin in coins:  
 for j in range(coin, money + 1):  
 l[j] = min(l[j], l[j - coin] + 1)  
  
 return l[money] if l[money] != float('inf') else -1  
  
def task1():  
 n, p, b = inp(PATH\_INPUT, 0, 'task1')  
 res = str(min\_coins(n, b))  
 global task\_numb  
 task\_numb = 1  
 outp(PATH\_OUTPUT, res)  
 print(caption(task\_numb, res))  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 start = time.perf\_counter()  
 task1()  
 time = float(time.perf\_counter() - start)  
 print(caption(task\_numb, time))

1. Текстовое объяснение решения.

Функция min\_coins строит массив, где каждый индекс представляет сумму, а значение в этом индексе — минимальное количество монет, необходимое для ее достижения. Если конкретная сумма не может быть составлена из доступных монет, результат для этой суммы остается бесконечным. Возвращается либо минимальное количество монет для размена суммы money, либо -1, если это невозможно.

1. Результат работы кода на примерах:



\*Результат работы кода (скрин терминала) так же будет приложен ниже

1. Вывод по задаче:

В результате выполнения задачи я научилась применять динамическое программирование для решения задачи про обмен монет для разных номиналов.

Задача №4. Наибольшая общая подпоследовательность двух по- следовательностей

1. Текст задачи

Вычислить длину самой длинной общей подпоследовательности из двух по- следовательностей.

Даны две последовательности A = (a1, a2, ..., an) и B = (b1, b2, ..., bm), найти

длину их самой длинной общей подпоследовательности, т.е. наибольшее неотри-

цатеьное целое число p такое, что существуют индексы 1 ≤ i1 < i2 < ... < ip ≤ n

и1≤j <j <...<j ≤mтакие,чтоa =b ,...,a =b . 12p i1j1ipjp

• **Формат ввода / входного файла (input.txt).**

**–** Первая строка: n - длина первой последовательности. **–** Вторая строка: a1, a2, ..., an через пробел.  
**–** Третья строка: m - длина второй последовательности.

**–** Четвертая строка: b1, b2, ..., bm через пробел.

* Ограничения: 1 ≤ n,m ≤ 100; −109 < ai,bi < 109.
* **Формат вывода / выходного файла (output.txt).** Выведите число p.

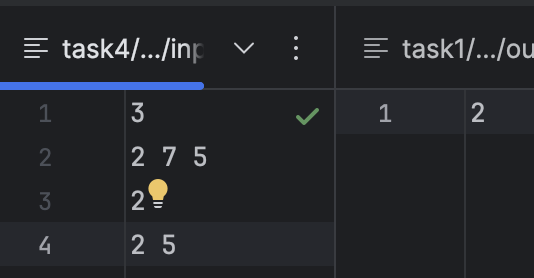
1. Листинг кода

import time  
import os  
from lab7.utils import inp, outp, caption  
  
CURRENT\_SCRIPT\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))  
os.chdir(CURRENT\_SCRIPT\_DIR)  
  
PATH\_INPUT = os.path.join('..', '..', 'files', 'input.txt')  
PATH\_OUTPUT = os.path.join('..', '..', 'files', 'output.txt')  
  
  
def longest\_common\_subsequence(n, a, m, b):  
 n = len(a)  
 m = len(b)  
 l = [[0] \* (m + 1) for \_ in range(n + 1)]  
  
 for i in range(1, n + 1):  
 for j in range(1, m + 1):  
 if a[i - 1] == b[j - 1]:  
 l[i][j] = l[i - 1][j - 1] + 1  
 else:  
 l[i][j] = max(l[i - 1][j], l[i][j - 1])  
  
 return l[n][m]  
  
def task1():  
 n, a, m, b = inp(PATH\_INPUT, 0, 'task4')  
 res = str(longest\_common\_subsequence(n, a, m, b))  
 global task\_numb  
 task\_numb = 4  
 outp(PATH\_OUTPUT, res)  
 print(caption(task\_numb, res))  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 start = time.perf\_counter()  
 task1()  
 time = float(time.perf\_counter() - start)  
 print(caption(task\_numb, time))

1. Текстовое объяснение решения.

Функция longest\_common\_subsequence заполняет этот массив в два этапа: если элементы a[i - 1] и b[j - 1] равны, то длина подпоследовательности увеличивается на 1, иначе выбирается максимальное значение из предыдущих шагов. В конце функции возвращается значение в правом нижнем углу массива l[n][m], которое и является длиной наибольшей общей подпоследовательности.

1. Результат работы кода на примерах:



\*Результат работы кода (скрин терминала) так же будет приложен ниже

1. Вывод по задаче:

В результате выполнения задачи я научилась применять динамическое программирование для нахождения подпоследовательности.

## Задача №5. Наибольшая общая последовательность трех подпоследовательностей.

1. Текст задачи

Вычислить длину самой длинной общей подпоследовательности из *трех* по- следовательностей.

Даны три последовательности A = (a1, a2, ..., an), B = (b1, b2, ..., bm) и C =

(c1, c2, ..., cl), найти длину их самой длинной общей подпоследовательности, т.е.

наибольшее неотрицатеьное целое число p такое, что существуют индексы 1 ≤

i1 <i2 <...<ip ≤n,1≤j1 <j2 <...<jp ≤mи1≤k1 <k2 <...<kp ≤l

такие,чтоa =b =c ,...,a =b =c . i1 j1 k1 ip jp kp

* 1. **Формат ввода / входного файла (input.txt).**

**–** Первая строка: n - длина первой последовательности. **–** Вторая строка: a1, a2, ..., an через пробел.  
**–** Третья строка: m - длина второй последовательности. **–** Четвертая строка: b1, b2, ..., bm через пробел.

**–** Пятая строка: l - длина второй последовательности. **–** Шестая строка: c1, c2, ..., cl через пробел.

* 1. Ограничения: 1 ≤ n,m,l ≤ 100; −109 < ai,bi,ci < 109.
  2. **Формат вывода / выходного файла (output.txt).** Выведите число p.

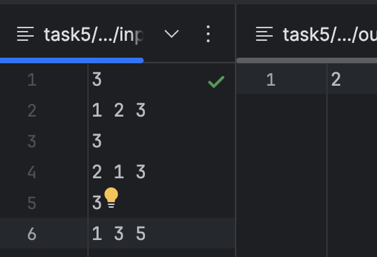
1. Листинг кода

import time  
import os  
from lab7.utils import inp, outp, caption  
  
CURRENT\_SCRIPT\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))  
os.chdir(CURRENT\_SCRIPT\_DIR)  
  
PATH\_INPUT = os.path.join('..', '..', 'files', 'input.txt')  
PATH\_OUTPUT = os.path.join('..', '..', 'files', 'output.txt')  
  
  
def subsequence(n, a, m, b, l, c):  
  
 posl = [[[0] \* (l + 1) for \_ in range(m + 1)] for \_\_ in range(n + 1)]  
  
 for i in range(1, n + 1):  
 for j in range(1, m + 1):  
 for k in range(1, l + 1):  
 if a[i - 1] == b[j - 1] == c[k - 1]:  
 posl[i][j][k] = posl[i - 1][j - 1][k - 1] + 1  
 else:  
 posl[i][j][k] = max(posl[i - 1][j][k], posl[i][j - 1][k], posl[i][j][k - 1])  
  
 return posl[n][m][l]  
  
def task1():  
 n, a, m, b, l, c = inp(PATH\_INPUT, 0, 'task5')  
 res = str(subsequence(n, a, m, b, l, c))  
 global task\_numb  
 task\_numb = 5  
 outp(PATH\_OUTPUT, res)  
 print(caption(task\_numb, res))  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 start = time.perf\_counter()  
 task1()  
 time = float(time.perf\_counter() - start)  
 print(caption(task\_numb, time))

1. Текстовое объяснение решения.

Функция subsequence использует трехмерный список posl для хранения результатов вычислений на каждом шаге. В этом списке posl[i][j][k] хранится длина наибольшей общей подпоследовательности. Тройной цикл перебирает все возможные индексы этих последовательностей и заполняет массив. Если текущие элементы всех трех последовательностей совпадают, то значение на соответствующем индексе увеличивается на 1. Если элементы не совпадают, то выбирается максимальное значение из предыдущих шагов.

1. Результат работы кода на примерах из текста задачи:



1. Вывод по задаче:

В результате выполнения задачи я научилась применять динамическое программирование для нахождения подпоследовательности.

## Задача №6 . Наибольшая возрастающая последовательность.

1. Текст задачи

Дана последовательность, требуется найти ее наибольшую возрастающаю под- последовательность.

* 1. **Формат ввода / входного файла (input.txt).** В первой строке входных данных задано целое число n – длина последовательности (1 ≤ n ≤ 300000).

Во второй строке задается сама последовательность. Числа разделяются пробелом.

Элементы последовательности – целые числа, не превосходящие по модулю 109 .

**–** Подзадача 1 (полегче). n ≤ 5000. **–** Общая подзадача. n ≤ 300000.

* 1. **Формат вывода / выходного файла (output.txt).** В первой строке выве- дите длину наибольшей возрастающей подпоследовательности, а во второй строке выведите через пробел саму наибольшую возрастающаю подпоследо- вательность данной последовательности. Если ответов несколько - выведите любой.

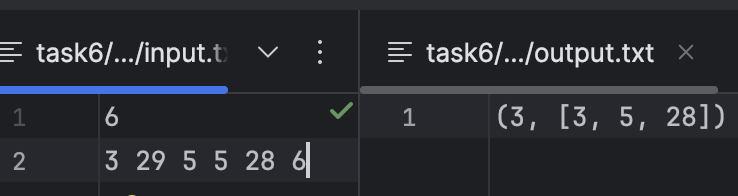
1. Листинг кода

import time  
import os  
from lab7.utils import inp, outp, caption  
  
CURRENT\_SCRIPT\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))  
os.chdir(CURRENT\_SCRIPT\_DIR)  
  
PATH\_INPUT = os.path.join('..', '..', 'files', 'input.txt')  
PATH\_OUTPUT = os.path.join('..', '..', 'files', 'output.txt')  
  
  
def increasing\_sequence(n, sequence):  
 n = len(sequence)  
 l = [float('inf')] \* n  
 parent = [-1] \* n  
 pos = [-1] \* n  
  
 max\_len = 0  
 max\_index = -1  
  
 for i, value in enumerate(sequence):  
 pos\_index = 0  
 while pos\_index < max\_len and l[pos\_index] < value:  
 pos\_index += 1  
  
 l[pos\_index] = value  
 pos[pos\_index] = i  
  
 if pos\_index > 0:  
 parent[i] = pos[pos\_index - 1]  
  
 if pos\_index + 1 > max\_len:  
 max\_len = pos\_index + 1  
 max\_index = i  
  
 lis = []  
 while max\_index != -1:  
 lis.append(sequence[max\_index])  
 max\_index = parent[max\_index]  
  
 lis.reverse()  
 return max\_len, lis  
  
def task1():  
 n, b = inp(PATH\_INPUT, 0, 'task6')  
 res = str(increasing\_sequence(n, b))  
 global task\_numb  
 task\_numb = 6  
 outp(PATH\_OUTPUT, res)  
 print(caption(task\_numb, res))  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 start = time.perf\_counter()  
 task1()  
 time = float(time.perf\_counter() - start)  
 print(caption(task\_numb, time))

1. Текстовое объяснение решения.

increasing\_sequence проходит по каждому элементу последовательности и находит, в какую позицию его можно вставить в возрастающую подпоследовательность, используя бинарный поиск. Это осуществляется с помощью переменной pos\_index, которая определяет индекс, на который следует вставить текущий элемент в список l. При этом также обновляются индексы в parent для отслеживания пути подпоследовательности.

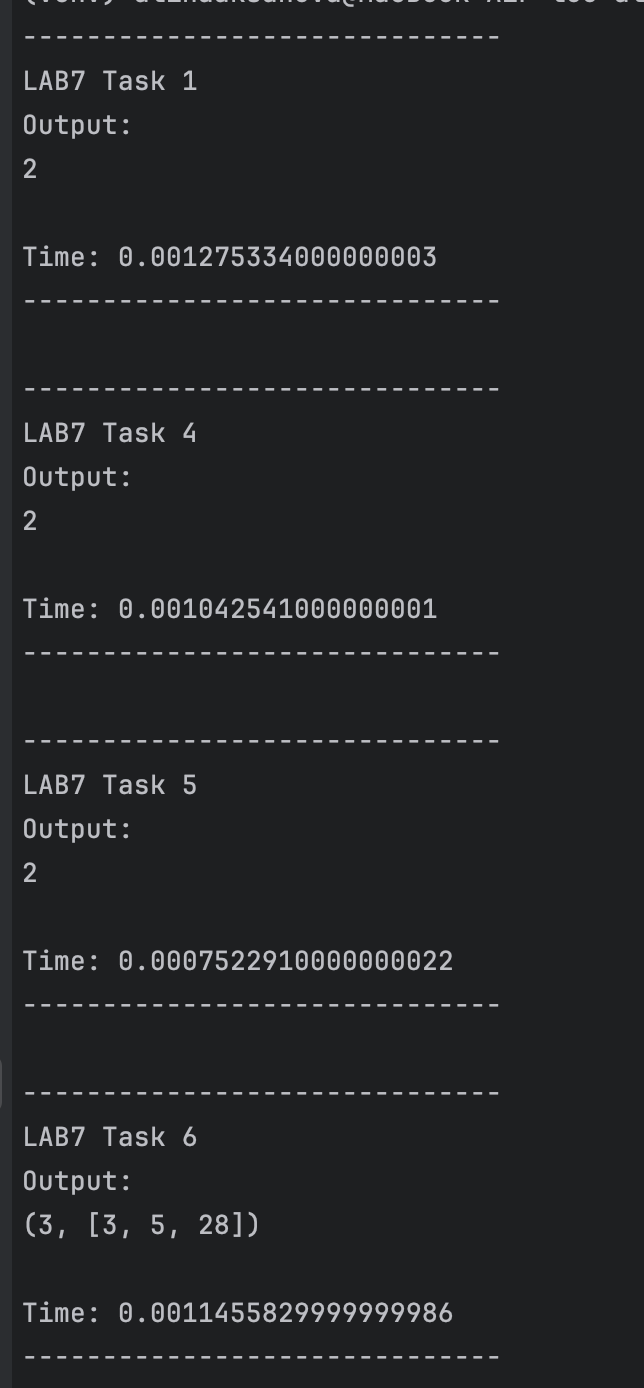
1. Результат работы кода на примерах из текста задачи:



1. Вывод по задаче:

В результате выполнения задачи я научилась применять динамическое программирование для нахождения наибольшей возрастающей последовательности.

## Результат работы всех задач

****

# Вывод

В результате лабораторной я изучила динамическое программирование.