(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 №1. Вариант 12

Выполнил:

Колпаков А.С.

K3139

Проверил:

Афанасьев А.В

Санкт-Петербург 2024 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №1. Обмен монет	3
Задача №5. Наибольшая общая подпоследовательность последовательностей	трех 5
Дополнительные задачи	9
Задача №2. Примитивный калькулятор	9
Задача №4. Наибольшая общая подпоследовательность последовательностей	двух 11
Вывод	15

Задачи по варианту

Задача №1. Обмен монет

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.

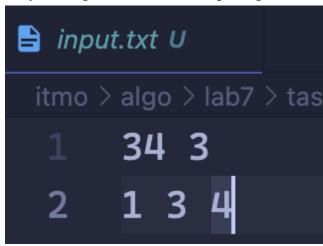
Листинг кода:

```
import utils
def min coins(money, coins):
   dp = [float('inf')] * (money + 1)
   dp[0] = 0
   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
if
          == ' main ':
     name
                       data
utils.read data('lab7/task1/textf/input.txt')
 res = min coins(data[0][0], data[1])
utils.print task data(7, 1, data, res)
    utils.write file("lab7/task1/textf/output.txt",
res)
```

Текстовое объяснение решения:

- 1. Создаем массив dp, который будет хранить минимальные значения для каждой суммы.
- 2. Для каждой суммы і и каждой монеты соіп из списка номиналов. Если і >= соіп, обновляем dp[i] = min(dp[i], dp[i - coin] + 1)
- 3. Если dp[money] == inf, значит, сумму нельзя разменять с использованием доступных монет.
- 4. В противном случае, dp[money] содержит минимальное количество монет.

Результат работы кода на примерах из текста задачи:





```
lab 7 task 1 input: [[34, 3], [1, 3, 4]] output: 9
```

	Время выполнения	Затраты памяти
Пример из задачи	0.000506292009958997 4 секунд	0.005377769470214844 МБ

Вывод по задаче:

Этот подход позволяет решить задачу для любых наборов монет, включая те, где жадный алгоритм не работает.

Задача №5. Наибольшая общая подпоследовательность трех последовательностей

Вычислить длину самой длинной общей подпоследовательности из $\underline{\textit{mpex}}$ последовательностей.

Даны три последовательности $A=(a_1,a_2,...,a_n),\,B=(b_1,b_2,...,b_m)$ и $C=(c_1,c_2,...,c_l)$, найти длину их самой длинной общей подпоследовательности, т.е. наибольшее неотрицатеьное целое число p такое, что существуют индексы $1\leq i_1< i_2<...< i_p\leq n,\,1\leq j_1< j_2<...< j_p\leq m$ и $1\leq k_1< k_2<...< k_p\leq l$ такие, что $a_{i_1}=b_{j_1}=c_{k_1},...,a_{i_p}=b_{j_p}=c_{k_p}$.

Листинг кода:

```
import utils

def longest_common_subsequence(a, b, c):
    n, m, l = len(a), len(b), len(c)

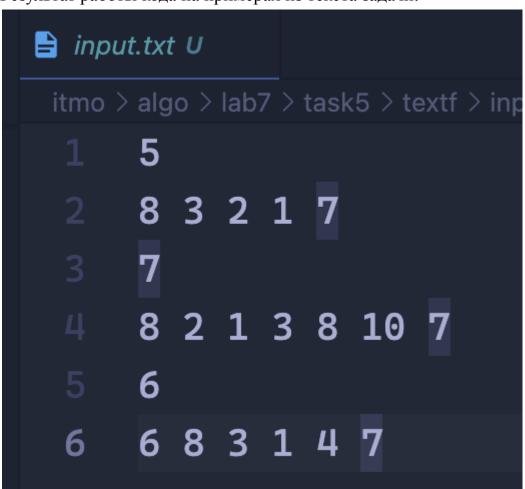
    dp = [[[0] * (l + 1) for x in range(m + 1)] for
i in range(n + 1)]

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]:
```

Текстовое объяснение решения:

- 1. dp[i][j][k] будет хранить длину самой длинной общей подпоследовательности для первых і элементов последовательности A, первых j элементов последовательности BB и первых k элементов последовательности C
- 2. Проходимся вложенными циклами по каждому элементу последовательностей, если текущие элементы послежовательностей совпадают, то записываем в соответствующую ячейку dp значение предыдущей + 1.
- 3. В ином случае записываем максимальный из элементов dp, убавляя индекс одной из ячеек.
- 4. В конце возвращаем длину самой длинной общей подпоследовательности.

Результат работы кода на примерах из текста задачи:





lab 7 task 2 input: [96234]

output: ['14\n', '1 3 9 10 11 22 66 198 594 1782 5346 16038 16039 32078 96234']

	Время выполнения	Затраты памяти
Пример из задачи	0.000299833016470074 65 секунд	0.005731582641601562 5 МБ

Вывод по задаче:

Программа эффективно реализует алгоритм нахождения длины самой длинной общей подпоследовательности трех последовательностей с помощью применения идеи динамического программирования.

Дополнительные задачи

Задача №2. Примитивный калькулятор

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 utils
def min operations(n):
   dp = [0] * (n + 1)
   for i in range (2, n + 1):
       dp[i] = dp[i - 1] + 1
       if i % 2 == 0:
           dp[i] = min(dp[i], dp[i // 2] + 1)
       if i % 3 == 0:
           dp[i] = min(dp[i], dp[i // 3] + 1)
  path = []
   current = n
  while current > 1:
       path.append(str(current))
          if current % 3 == 0 and dp[current]
dp[current // 3] + 1:
           current //= 3
         elif current % 2 == 0 and dp[current] ==
dp[current // 2] + 1:
```

Текстовое объяснение решения:

- 1. dp[i] минимальное количество операций, необходимых для получения числа і начиная с 1
- 2. Проходим циклом по по всем значениям. Если значение делится на 2, то минимальное количество операций будет равно либо текущему значению, либо значению dp деленному на 2+1.
- 3. Аналогично если значение делится на 3.
- 4. Чтобы восстановить последовательность операций, нужно отслеживать, откуда пришло значение dp[i] (например, из i-1, i//2 или i//3)

Результат работы кода:

lab 7 task 2 input: [96234]

output: ['14\n', '1 3 9 10 11 22 66 198 594 1782 5346 16038 16039 32078 96234']

	Время выполнения	Затраты памяти
Пример из задачи	0.0006839580019004643 секунд	0.0052337646484375 МБ

Вывод по задаче:

Программа успешно реализует алгоритм поиска минимального количества операций для получения определенного числа для примитивного калькулятора с помощью идеи динамического программирования.

Задача №4. Наибольшая общая подпоследовательность двух последовательностей

Вычислить длину самой длинной общей подпоследовательности из двух последовательностей.

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

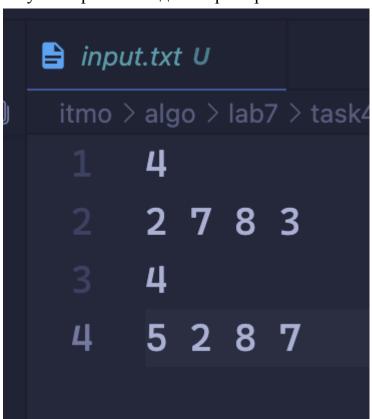
Листинг кода:

```
import utils
def longest common subsequence(a, b):
   n = len(a)
  m = len(b)
   dp = [[0] * (m + 1) for x 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]:
               dp[i][j] = dp[i - 1][j - 1] + 1
           else:
               dp[i][j] = max(dp[i - 1][j], dp[i][j]
- 1])
   return dp[n][m]
if name == " main ":
 data =
utils.read data('lab7/task4/textf/input.txt')
res = longest common subsequence(data[1], data[3])
utils.print task data(7, 4, data, res)
utils.write file("lab7/task4/textf/output.txt",
res)
```

Текстовое объяснение решения:

- 1. Создаем список dp, где dp[i][j] длина самой длинной общей подпоследовательности первых і элементов последовательности A и первых ј элементов последовательности B.
- 2. Проходим циклом по каждому элементу последовательностей, если элементы совпадают, то длина по текущим индексам будет равна предыдущей длине + 1.
- 3. В ином случае длина по текущим индексам будет равна максимальному из элементов dp[i-1][j], dp[i][j-1].
- 4. После заполнения списка dp, длина самой длинной общей подпоследовательности будет находиться в dp[n][m], где n и m длины последовательностей A и B.

Результат работы кода на примерах из текста задачи:





lab 7 task 4 input: [4, [2, 7, 8, 3], 4, [5, 2, 8, 7]] output: 2

	Время выполнения	Затраты памяти
Пример из задачи	0.000903249994735233 5 секунд	0.0052032470703125 МБ

Вывод по задаче:

Программа эффективно реализует алгоритм нахождения длины самой длинной общей подпоследовательности двух последовательностей с помощью применения идеи динамического программирования.

Вывод

В ходе лабораторной работы были изучены и реализованы различные алгоритмы с помощью применения идеи динамического программирования. Эта идея помогает эффективно управлять данными и решать задачи сортировки, поиска и упорядочивания.