Санкт-Петербургский Национальный Исследовательский Университет
Информационных Технологий, Механики и Оптики

Факультет инфокоммуникационных технологий

Лабораторная работа №4

Выполнили:

Петрова Н. Г

Садовая А. Р

Оншин Д. Н

Проверил:

Мусаев А.А.

Санкт-Петербург,

2023

СОДЕРЖАНИЕ

Стр.

ВВЕДЕНИЕ

Данная работа представляет собой отчет о выполненных заданиях:

1. Задача 1 (вор в музее).
2. Задача 2 (произведение матриц)
3. Задча 3 (поиск наибольшей непрерывной возрастающей последовательности из чисел внутри массива)

Динамическое программирование (Dynamic Programming) - это метод решения задач оптимизации, который заключается в разбиении исходной задачи на более мелкие подзадачи и решении их вплоть до базового случая.

В отличие от жадных алгоритмов, которые выбирают локально оптимальные решения на каждом шаге в надежде, что их сумма приведет к оптимальному решению задачи в целом, динамическое программирование использует подзадачи, которые могут повторяться в разных ветвях решения, сохраняя их решение и повторно используя его для ускорения работы. В данном отчёте будут продемонстрированы примеры решения подобных задач.

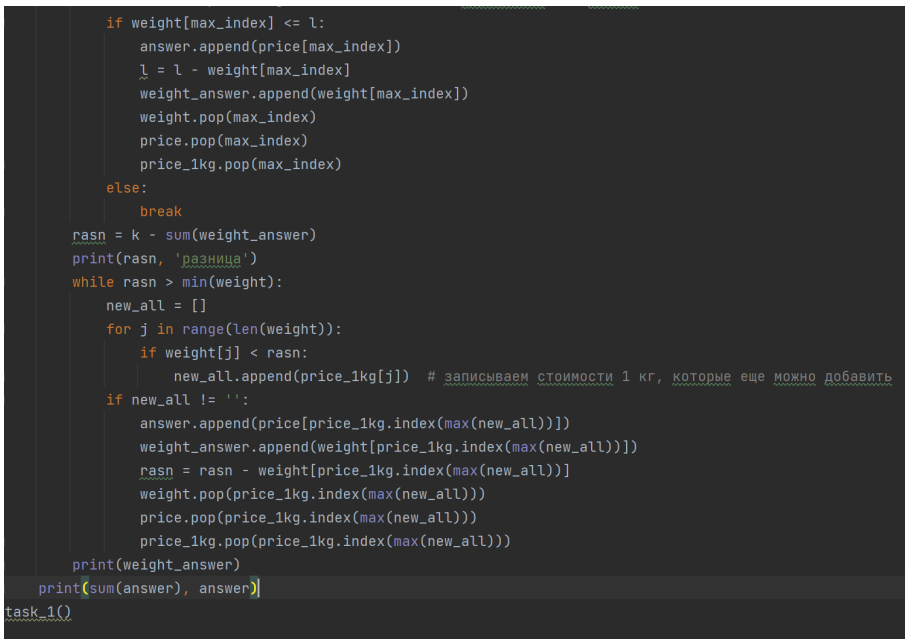
1 Задача 1

Самым оптимальным алгоритмом с использованием динамического программирования для решения данной задачи будет алгоритм в котором нужно рассчитать отношение цены к весу для каждого экспоната и в каждый заход брать самые "ценные"экспонаты по отношению цены к весу, а в оставшееся по весу место будут определяться не самые выгодные, но максимальные по стоимости и возможные для расположения по весу. То есть с помощью первого цикла while мы берем самые "ценные"экспонаты, пока у нас есть такая возможность. Затем мы считаем разность по весу, которая осталась в текущем заходе и отбираем из экспонатов те, которые потенциально могут войти в заход, то есть их вес меньше или равен оставшегося свободного места. И соответственно из отобранных экспанатов мы берем те, которые принесут наибольшую стоимость. Таким образом, вор будт извлекать наибольшую выгоду от суммарно вынесенного веса экспонатов.

```python
def task_1():
    exhibits = {'Mona Lisa': '1 10', 'David': '20 15', 'Madonna': '1 7', 'Peacock': '10 20', 'Dance': '3 5',
                'Spring': '10 10', 'Eggs': '3 30', '1': '23 99', '2': '3 10',
                '3': '12 50', '4': '3 1', '5': '16 23', '6': '2 5'}  # exhibits: weight price
    while True:
        n = int(input(f'Input N (not more than {len(exhibits)}): '))
        if n <= len(exhibits):
            break
        else:
            print('Wrong input!')
    k = int(input('Input K: '))  # вес за один заход
    m = int(input('Input M: '))  # количество заходов
    weight = [1, 20, 1, 10, 3, 10, 3, 23, 3, 12, 3, 16, 2]
    price = [10, 15, 7, 20, 5, 10, 30, 99, 10, 50, 1, 23, 5]
    price_1kg = []
    for i in range(len(price)):
        price_1kg.append(round(price[i] / weight[i], 2))
    print(price_1kg)
    answer = []

    for i in range(m):
        print(i, 'заход')
        l = k
        weight_answer = []
        while True:
            print(l)
            max_value = max(price_1kg)  # берем максимальную стоимость 1 кг(самый ценный)
            max_index = price_1kg.index(max_value)  # определяем его индекс
```

Рисунок 1.1 — Код функции для задачи 1



```
            if weight[max_index] <= l:
                answer.append(price[max_index])
                l = l - weight[max_index]
                weight_answer.append(weight[max_index])
                weight.pop(max_index)
                price.pop(max_index)
                price_1kg.pop(max_index)
            else:
                break
        rasn = k - sum(weight_answer)
        print(rasn, 'разница')
        while rasn > min(weight):
            new_all = []
            for j in range(len(weight)):
                if weight[j] < rasn:
                    new_all.append(price_1kg[j])  # записываем стоимости 1 кг, которые еще можно добавить
            if new_all != '':
                answer.append(price[price_1kg.index(max(new_all))])
                weight_answer.append(weight[price_1kg.index(max(new_all))])
                rasn = rasn - weight[price_1kg.index(max(new_all))]
                weight.pop(price_1kg.index(max(new_all)))
                price.pop(price_1kg.index(max(new_all)))
                price_1kg.pop(price_1kg.index(max(new_all)))
        print(weight_answer)
    print(sum(answer), answer)
task_1()
```

Рисунок 1.2 — Продолжение кода функции для задачи 1

```
Input N (not more than 13): 10
Input K: 5
Input M: 3
[10.0, 0.75, 7.0, 2.0, 1.67, 1.0, 10.0, 4.3, 3.33, 4.17, 0.33, 1.44, 2.5]
0 заход
5
4
1
0
0 разница
[1, 3, 1]
1 заход
5
5 разница
[3]
2 заход
5
5 разница
[2]
62 [10, 30, 7, 10, 5]
```

Рисунок 1.3 — Пример вывода функции для задачи 1

Переменная "N"обозначает количество экспонатов в музее. Переменная "K"обозначает максимальный вес, который вор может унести за один раз. Переменная "M"обозначает количество заходов, которые вор может сделать в музей.

2 Задача 2

Алгоритм по шагам:

1. Передаем в функцию список размерностей матриц в последовательности, где p[i] - размерность i-й матрицы, а n = len(p) - 1, так как мы рассматриваем n матриц, но количество элементов в списке p равно n+1.
2. Далее мы создаем таблицу или двумерный массив размером (n x n), где будут в каждой ячейке минимальное количество скалярных операций для умножения матриц от i до j.
3. Затем мы заполняем диагональные элементы нулями, так как нам не нужно умножать матрицу саму на себя.
4. Затем мы заполняем остальные элементы таблицы, используя циклы, чтоюы пройтись по всем возможным цепочкам матриц. Внешний цикл проходится по всем длинам цепочек матриц (от двух до n, так как для умножанеия матриц нужно минимум 2), а затем также с помощью циклов проходимся по всем возможным начальным и конечным индексам цепочек матриц. То есть, другими словами, для каждой цепочки матриц мы проходимся по всем возможным разбиениям на две подцепочки матриц и выбираем наименьшее количество скалярных операций, которое требуется для умножения двух подцепочек матриц на текущем шаге цикла.
5. В конце функция возвращает значение из крайней ячейки таблицы, которое содержит минимальное количество скалярных операций для умножения всех матриц в последовательности.

```python
def matrix(p):
    n = len(p) - 1
    our_matrix = [[float('inf') for i in range(n)] for j in range(n)]
    for i in range(n):
        our_matrix[i][i] = 0 #по диагонали ставим 0, так как матрицу на эту же матрицу умножать не нужно
    for w in range(2, n+1):
        for i in range(n-w+1):
            j = i + w - 1
            for k in range(i, j):#проходимся по всем возможным разбиениям на две подцепочки матриц
                our_matrix[i][j] = min(our_matrix[i][j], our_matrix[i][k] + our_matrix[k+1][j] + p[i]*p[k+1]*p[j+1])

    return our_matrix[0][n-1]
m = [5, 10, 15]
min_op = matrix(m)
print(min_op, 'скалярных действий')
```

Рисунок 2.1 — Код функции для задачи 2

Рисунок 2.2 — Пример вывода функции для задачи 2

3 Задача 3

```python
def longest_sequence(arr):
    n = len(arr)
    max_len = 1
    max_seq = [arr[0]]

    for i in range(n):
        current_seq_len = 1
        current_seq = [arr[i]]

        for j in range(i + 1, n):
            if arr[j] > current_seq[-1]:
                current_seq.append(arr[j])
                current_seq_len += 1
            else:
                break

        if current_seq_len > max_len:
            max_len = current_seq_len
            max_seq = current_seq

    return max_len, max_seq
import random
n = random.randint(10, 20) # случайное число от 10 до 20 включительно
arr = [random.randint(-100, 100) for i in range(n)] #генерируется массив из n чисел
max_len, max_seq = longest_sequence(arr)
print("Длина наибольшей возрастающей последовательности:", max_len)
print("Наибольшая возрастающая последовательность:", max_seq)
print("Исходный массив чисел:",arr)
```

Рисунок 3.1 — Код функции для задания 3

```
Длина наибольшей возрастающей последовательности: 3
Наибольшая возрастающая последовательность: [-85, 9, 16]
Исходный массив чисел: [37, -67, 78, 71, -94, 52, -85, 9, 16, 10]
```

Рисунок 3.2 — Пример вывода функции для задания 3

1) Инициализируется переменная max len с начальным значением 1 (любая последовательность длины 1 является возрастающей). 2) Инициализируется переменная max seq с начальным значением, равным первому элементу массива arr. 3) Происходит перебор всех элементов массива arr с помощью цикла for i in range(n), где n — длина массива. 4) Для каждого элемента arr[i] начинается внутренний цикл for j in range(i + 1, n), который перебирает все оставшиеся элементы массива arr, начиная с индекса i + 1. 5) Если очередной элемент arr[j] больше последнего элемента текущей последовательности current eq[-1], то он добавляется в текущую последовательность current seq, а длина текущей последовательности увеличивается на 1.А если очередной

элемент arr[j] меньше или равен последнему элементу текущей последовательности current seq[-1], то внутренний цикл прерывается с помощью оператора break. 6) Если длина текущей последовательности current seq len больше длины наибольшей найденной последовательности max len, то значения переменных max len и max seq обновляются значениями текущей последовательности current seq и её длиной current seq len. 7) Затем все найденные значения выводятся на экран

ЗАКЛЮЧЕНИЕ

Таким образом, были разработаны программы на языке Python с применением методов динамического программирования для решения различных задач. Изучены новые подходы и методы. В результате, для каждой задачи были представлены программы и их вывод. Все программы можно найти на репозитории в GitHub [1].

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. GitHub [Электронный ресурс]: `https://github.com/NatalyaPetrova/Algoritms` (дата обращения 01.05.2023).