

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

Факультет инфокоммуникационных технологий

Домашнее задание №4

Вариант №1

Выполнил:

Бацанова Е. А.

Проверил

Мусаев А.А.

Санкт-Петербург,

2024

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ.....	3
Задание 1.....	4
Задание 2.....	7
Задание 3.....	8
ЗАКЛЮЧЕНИЕ.....	9
СПИСОК ЛИТЕРАТУРЫ.....	10
ПРИЛОЖЕНИЕ.....	11

ВВЕДЕНИЕ

Динамическое программирование является одним из основных методов оптимизации задач, которые можно разбить на подзадачи и решать их последовательно.

В данной работе мы рассмотрим несколько задач, которые можно эффективно решить с помощью динамического программирования.

Целью данной работы является исследование и применение методов динамического программирования для решения различных задач, направленных на оптимизацию (поиск глобального экстремума рассматриваемого в задаче параметра). В ходе работы были решены следующие задачи:

- 1) решение задачи о поиске максимально возможной цены экспонатов, которые вор может украсть за фиксированное число заходов, имея ограничение на суммарный вес украденных в каждый заход объектов;
- 2) сравнение вывода в случае решения данной задачи с помощью динамического метода и жадных алгоритмов;
- 3) решение задачи о минимизация количества операций для умножения матриц, обладающего свойством ассоциативности;
- 4) решение задачи о поиске наибольшей длины непрерывной возрастающей последовательности в массиве.

Задание 1

Задание: Вор пробрался в музей и хочет украсть N экспонатов. У каждого экспоната есть свой вес и цена. Вор может сделать M заходов, каждый раз унося K кг веса. Определить, что должен унести вор, чтобы сумма украденного была максимальной.

Решение:

Для решение данной задачи реализуем функцию *max_steal_dynamic*(M , K , *exhibits*) (рис. 1). Сначала в данной функции для удобства разделим список кортежей *exhibits* на два массива – *weights* (содержит вес экспонатов) и *values* (содержит их цену). Далее на каждом шаге (всего их количество равно числу заходов в музей) будет вызываться функция *steals*(K , *weights*, *values*) (рис. 2), которая ищет оптимальную комбинацию предметов для кражи из оставшихся в массиве *exhibits*. После каждого захода вора к переменной *total_value* прибавляется стоимость украденных экспонатов, после чего из массивов *weights* и *values* удаляются украденные предметы.

Функция *steals*(K , *weights*, *values*) (рис. 2) реализует поиск оптимального решения для одного захода в музей с помощью динамического программирования. В качестве аргументов функции передается максимально возможный вес украденного (K) и массивы, содержащие доступные для кражи предметы и их вес. Внутри функции создается двумерный массив с размером $(n + 1) \times (K + 1)$, где n – количество доступных для кражи объектов. Изначально все значения заполняются нулями. Далее в цикле перебираются все товары и веса. Решение в каждую итерацию получаем как максимум из двух вариантов: если предмет i не входит (тогда решение совпадает с решением на прошлом шаге по предметам), если входит – учитываем вес предмета и его стоимость. Функция возвращает максимальную стоимость украденного и номера украденных предметов в исходных массивах.

```
def max_steal_dynamic(M, K, exhibits):
    total_value = 0
    weights = [item[0] for item in exhibits]
    values = [item[1] for item in exhibits]
    for i in range(M):
        x = steals(K, weights, values)
        total_value += x[0]
        stolen = x[1]
        stolen.sort(reverse=True)
        for j in stolen:
            del weights[j]
            del values[j]
    return total_value
```

Рисунок 1 – Реализация функции, максимизирующей стоимость украденного в М заходов

```
def steals(K, weights, values):
    n = len(weights)
    dp = [[0 for _ in range(K + 1)] for _ in range(n + 1)]
    for i in range(1, n + 1):
        for j in range(1, K + 1):
            if weights[i - 1] <= j:
                dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - weights[i - 1]] + values[i - 1])
            else:
                dp[i][j] = dp[i - 1][j]
    result = dp[n][K]

    stolen_items = []
    w = K
    for i in range(n, 0, -1):
        if dp[i][w] != dp[i - 1][w]:
            stolen_items.append(i - 1)
            w -= weights[i - 1]

    return result, stolen_items
```

Рисунок 2 – Реализация функции, максимизирующей стоимость украденного в один заход

Протестируем работу программы (рис. 3) – видим, что в отличие от решения задачи, приведенного в прошлом домашнем задании, реализованного с использованием жадного алгоритма, данная функция в обоих случаях выдает оптимальный ответ.

```
37 # пример 1
38 M = 3
39 K = 4
40 exhibits = [(2, 1500), (1, 1000), (4, 5000), (3, 2000), (2, 1250), (3, 2500)]
41 print(max_steal_dynamic(M, K, exhibits))
42
43 # пример 2
44 M = 2
45 K = 4
46 exhibits = [(2, 1500), (1, 1000), (3, 3000), (4, 2000)]
47 print(max_steal_dynamic(M, K, exhibits))
48
steals() > for i in range(n, 0, -1) > if dp[i][w] != dp[i - 1][w]
```

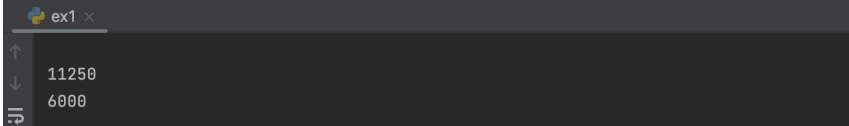


Рисунок 3 – Результат вывода программы, максимизирующей стоимость украденного в один заход, для двух примеров

Задание 2

Задание: Дана последовательность матриц A, B, C, \dots, Z таким образом, что с ними можно выполнить ассоциативные операции. Используя динамическое программирование, минимизируйте количество скалярных операций для нахождения их произведения.

Решение:

Реализуем минимизацию количества скалярных операций для нахождения произведения матриц с помощью функции `min_operations(arr)` (рис. 4). Алгоритм использует динамическое программирование – изначально создается двумерный массив dp размером $n \times n$, где n – количество перемножаемых матриц. Этот массив будет хранить минимальное количество операций умножения для каждой пары матриц. Затем алгоритм в цикле проходит по всем возможным парам матриц. Для вычисления минимального количества операций для данной пары матриц используется внутренний цикл, который перебирает все возможные точки разбиения между матрицами и выбирает наилучший вариант, минимизируя количество операций. В конце алгоритм возвращает значение $dp[0][n-1]$, которое представляет минимальное количество операций для умножения всех матриц в последовательности.

```
def min_operations(arr):
    n = len(arr)
    dp = [[0 for _ in range(n)] for _ in range(n)]

    for i in range(n - 2, -1, -1):
        for j in range(i + 2, n):
            dp[i][j] = float('infinity')
            for k in range(i + 1, j):
                dp[i][j] = min(dp[i][j], dp[i][k] + dp[k][j] + arr[i][0] * arr[k][1] * arr[j][1])

    return dp[0][n - 1]
```

Рисунок 4 – Реализация программы, минимизирующей число операций перемножения матриц

Для набора матриц [(3, 3), (3, 6), (6, 1), (1, 5), (5, 12)] минимальное число операций равно 213

Рисунок 5 – Вывод программы, минимизирующей число операций перемножения матриц

Задание 3

Задание: Дан массив N , состоящий из p случайных целых чисел, находящихся в диапазоне от -100 до 100 . Найти наибольшую непрерывную возрастающую последовательность из чисел внутри массива (длину серии, для которой верно $N[i] < N[i+1] < N[i+2] < \dots < N[i+m]$, где $i \geq 0$, а $i + m \leq n-1$).

Решение:

Решение для данной задачи реализуем с помощью функции `max_increasing_sequence(arr)` (рис. 6). Она принимает на вход массив `arr`, состоящий из целых чисел, и возвращает длину наибольшей возрастающей последовательности в этом массиве. Внутри функции создается массив `dp` длиной n , где n – длина входного массива `arr`. В этом массиве будут храниться длины максимальных возрастающих последовательностей. Проходим по массиву `arr` со второго элемента до последнего и для каждого элемента проверяем, больше ли он предыдущего элемента. Если это так, то увеличиваем значение `dp[i]` на 1 от значения, если нет – значение `dp[i]` остается равным 1, так как это начало новой возрастающей последовательности. В конце возвращается максимальное значение из массива `dp`. Если массив `arr` пустой – сразу возвращаем 0.

```
def max_increasing_sequence(arr):  
    n = len(arr)  
    if n == 0:  
        return 0  
    dp = [1] * n  
    for i in range(1, n):  
        if arr[i] > arr[i - 1]:  
            dp[i] = dp[i - 1] + 1  
    return max(dp)
```

Рисунок 6 – Реализация программы, находящей максимальную длину возрастающей подпоследовательности массива

```
Максимальная длина возрастающей подпоследовательности  
для массива [-3, -2, 1, 5, 4, 3, 6] равна 4
```

Рисунок 7 – Вывод программы, находящей максимальную длину возрастающей подпоследовательности массива

ЗАКЛЮЧЕНИЕ

В результате выполнения данной работы были разработаны и реализованы эффективные алгоритмы с применением метода динамического программирования для решения различных задач.

Была решена задача о поиске максимально возможной цены экспонатов, которые вор может украсть за фиксированное число заходов, имея ограничение на суммарный вес украденных в каждый заход объектов. После этого было проведено сравнение полученного динамического решения с решением через жадные алгоритмы (реализовывалось в домашней работе 4). Динамическое решение выдавало оптимальные ответы даже там, где решение через жадный алгоритм не работало. Из этого можно сделать вывод о том, что для решения данной задачи предпочтительно использование динамического программирования.

Далее были решены задачи о минимизация количества операций для умножения матриц, обладающего свойством ассоциативности и о поиске наибольшей длины непрерывной возрастающей последовательности в массиве. Для них также не удалось найти входных данные, при которых возникла бы ошибка и в выводе оказался неоптимальный ответ.

В итоге можно сказать, что динамическое программирование является мощным инструментом, который позволяет эффективно решать сложные задачи и оптимизировать процессы в различных областях.

СПИСОК ЛИТЕРАТУРЫ

- 1) Wikipedia. [Динамическое программирование](#). [Электронный ресурс] – (Дата последнего обращения 19.05.2024);
- 2) Wikipedia. [Задача о порядке перемножения матриц](#). [Электронный ресурс] – (Дата последнего обращения 19.05.2024);
- 3) Wikipedia. [Наибольшая общая подпоследовательность](#). [Электронный ресурс] – (Дата последнего обращения 19.05.2024).

ПРИЛОЖЕНИЕ

Для удобства все файлы выгружены на GitHub:
<https://github.com/kathykkKk/Algorithms-and-Data-Structures-ICT.git>