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](#_fmjc2tazxxvi)

[Задание 1 4](#_2jzah6mlznqr)

[Задание 2 7](#_rnepktrnyk9)

[Задание 3 8](#_asw6g2a9gnmk)

[ЗАКЛЮЧЕНИЕ 9](#_ohelfv1lj554)

[СПИСОК ЛИТЕРАТУРЫ 10](#_1dqo8ded3fhd)

[ПРИЛОЖЕНИЕ 11](#_lbxrcchhzy39)

# **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 М заходов, каждый раз унося К 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 c размером *(n + 1) × (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.

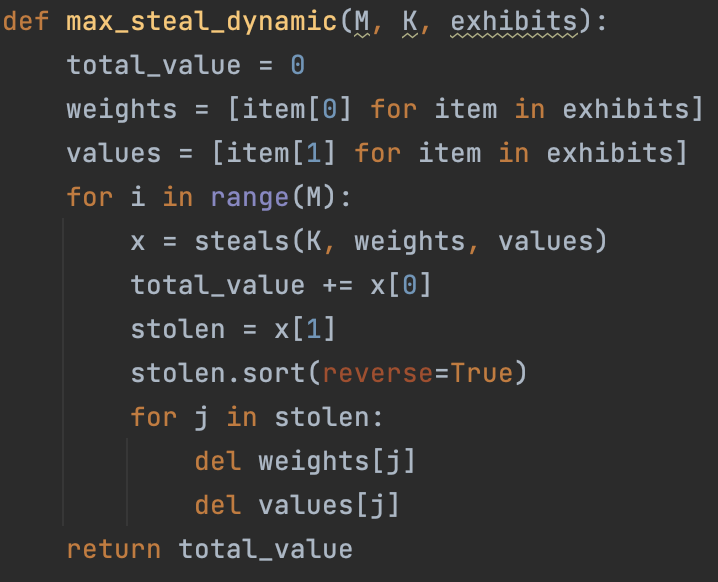


Рисунок 1 – Реализация функции, максимизирующей стоимость украденного в M заходов

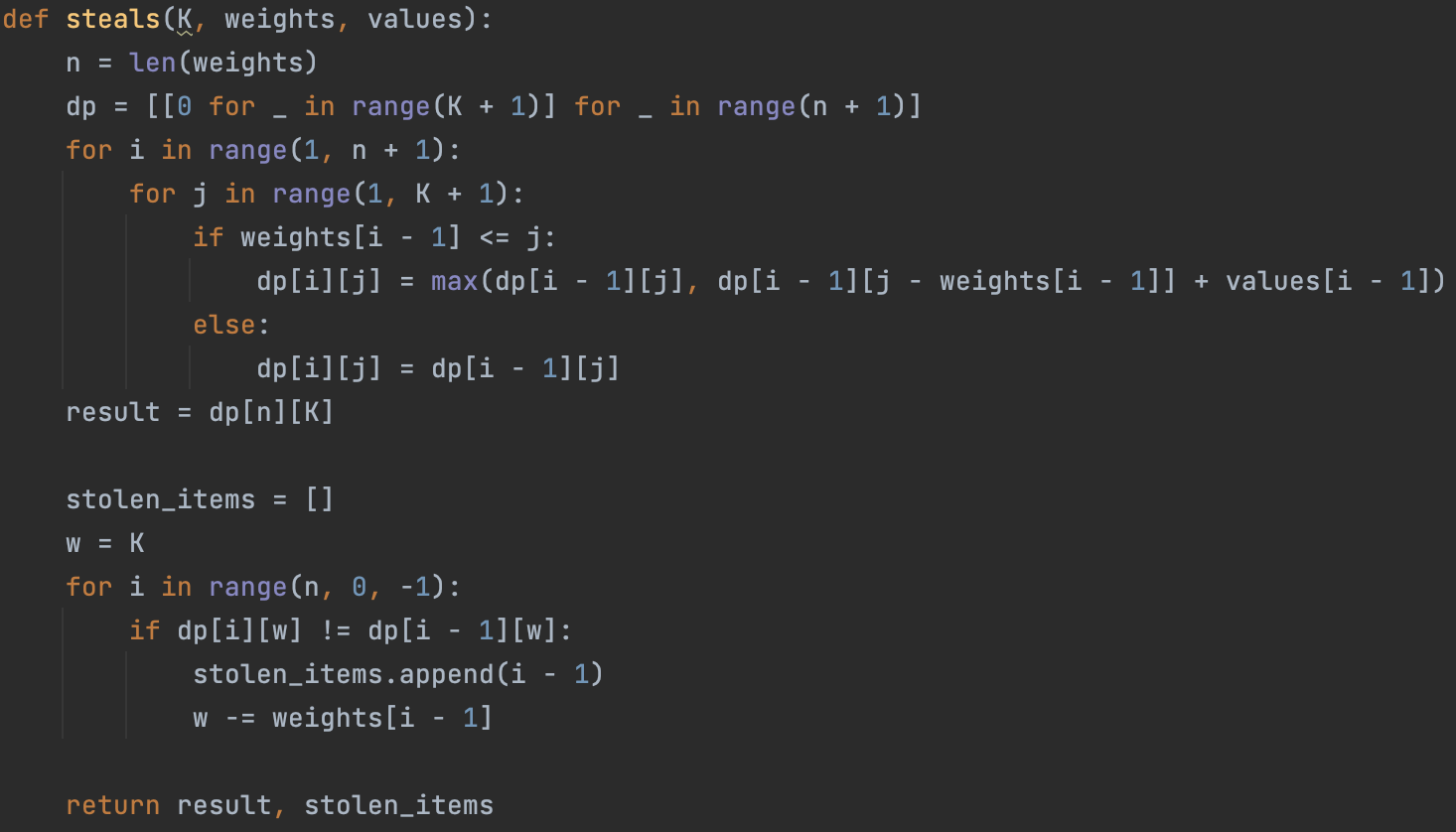


Рисунок 2 – Реализация функции, максимизирующей стоимость украденного в один заход

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

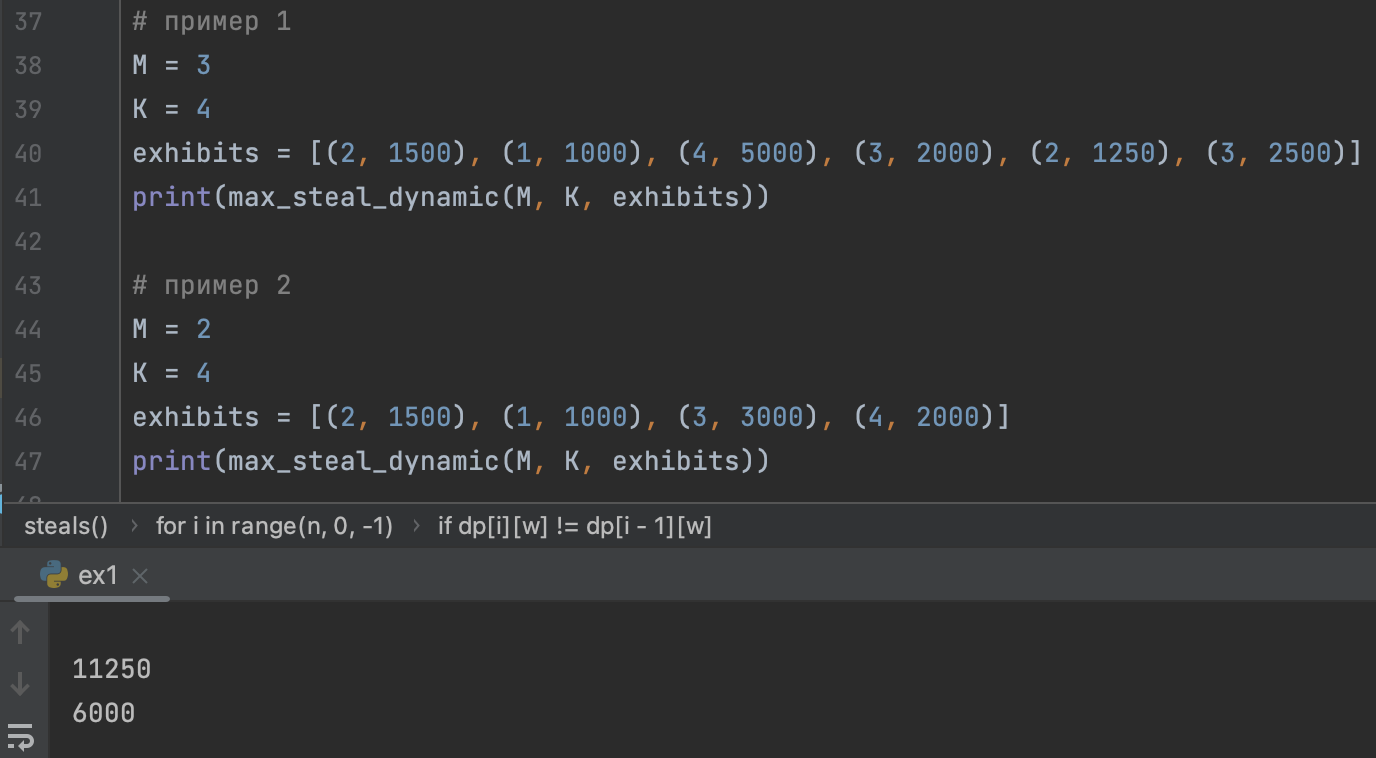


Рисунок 3 – Результат вывода программы, максимизирующей стоимость украденного в один заход, для двух примеров

# **Задание 2**

**Задание:** Дана последовательность матриц А, В, С, .. , 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 × 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 количество 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.

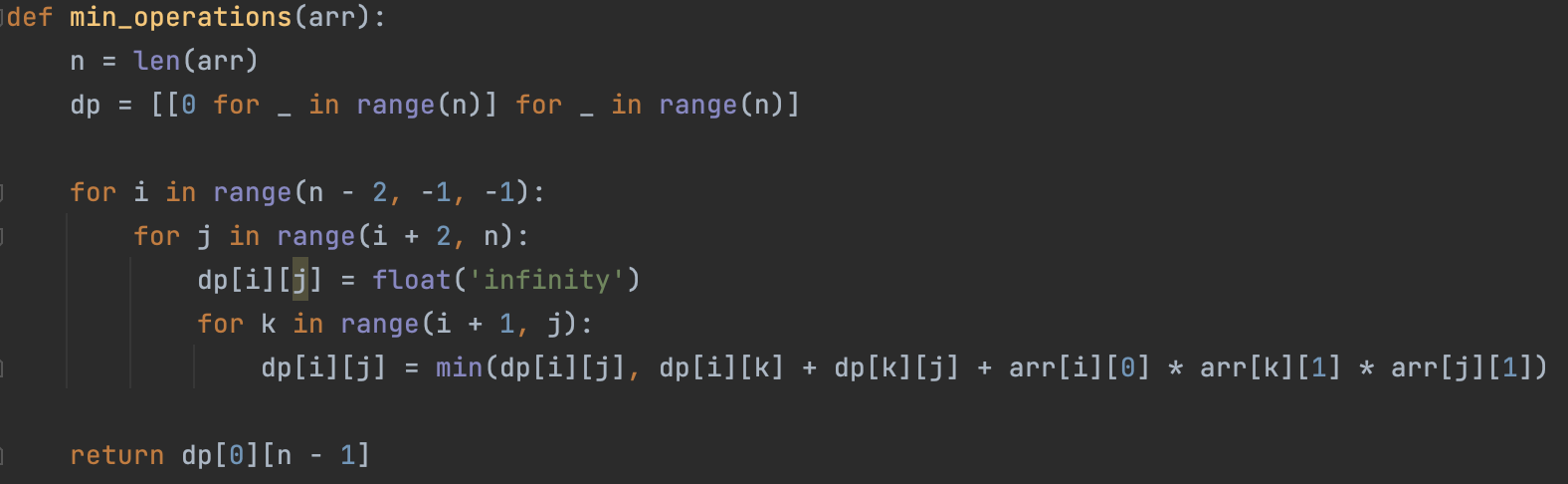


Рисунок 4 – Реализация программы, минимизирующей число операций перемножения матриц



Рисунок 5 – Вывод программы, минимизирующей число операций перемножения матриц

# **Задание 3**

**Задание:** Дан массив N, состоящий из п 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] < … < N[i+m], где i ≥ 0, а i + m ≤ 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 будет 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.

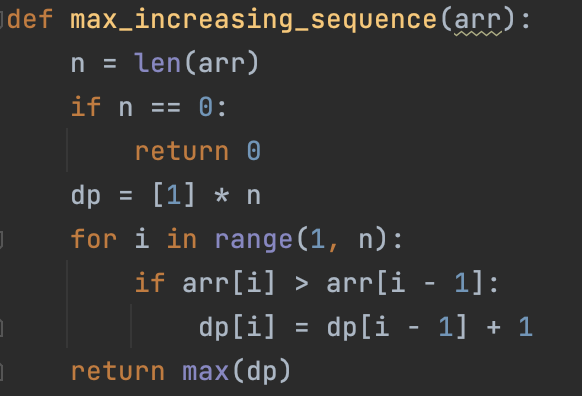


Рисунок 6 – Реализация программы, находящей максимальную длину возрастающей подпоследовательности массива

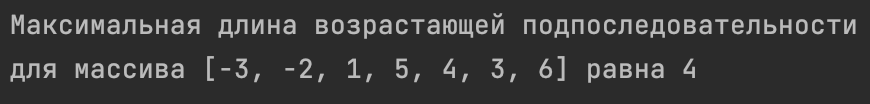


Рисунок 7 – Вывод программы, находящей максимальную длину возрастающей подпоследовательности массива

# 

# **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 данны, 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](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5#:~:text=%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5%20%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%20%D0%BE%D0%B1%D1%8B%D1%87%D0%BD%D0%BE%20%D0%BF%D1%80%D0%B8%D0%B4%D0%B5%D1%80%D0%B6%D0%B8%D0%B2%D0%B0%D0%B5%D1%82%D1%81%D1%8F%20%D0%B4%D0%B2%D1%83%D1%85%20%D0%BF%D0%BE%D0%B4%D1%85%D0%BE%D0%B4%D0%BE%D0%B2%20%D0%BA%20%D1%80%D0%B5%D1%88%D0%B5%D0%BD%D0%B8%D1%8E,%D0%B8%20%D0%B7%D0%B0%D1%82%D0%B5%D0%BC%20%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D1%83%D1%8E%D1%82%D1%81%D1%8F%20%D0%B4%D0%BB%D1%8F%20%D0%BF%D0%BE%D1%81%D1%82%D1%80%D0%BE%D0%B5%D0%BD%D0%B8%D1%8F%20%D1%80%D0%B5%D1%88%D0%B5%D0%BD%D0%B8%D1%8F%20%D0%B8%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D0%BE%D0%B9%20%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%B8.). [Электронный ресурс] – (Дата последнего обращения 19.05.2024);
2. Wikipedia. [Задача о порядке перемножения матриц](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%BE_%D0%BF%D0%BE%D1%80%D1%8F%D0%B4%D0%BA%D0%B5_%D0%BF%D0%B5%D1%80%D0%B5%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86). [Электронный ресурс] – (Дата последнего обращения 19.05.2024);
3. Wikipedia. [Наибольшая общая подпоследовательность](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D0%B8%D0%B1%D0%BE%D0%BB%D1%8C%D1%88%D0%B0%D1%8F_%D0%BE%D0%B1%D1%89%D0%B0%D1%8F_%D0%BF%D0%BE%D0%B4%D0%BF%D0%BE%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8C). [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>