САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №1
по курсу «Алгоритмы и структуры данных»

Вариант 14

Выполнила:
Лаузер Я.П. К3144

Проверил:

Санкт-Петербург
2024 г.

Содержание отчета

Лабораторная №1

Задача №1. Максимальная стоимость добычи

Задача №5. Максимальное количество призов

Задача №11. Максимальное количество золота

Задача №16. Продавец

Задача №20. Почти полидром

1 лаба “Жадные алгоритмы. Динамическое программирование No2”.

Задача №1.Максимальная стоимость добычи

Текст задачи

Вор находит гораздо больше добычи, чем может поместиться в его сумке. Помогите ему найти самую ценную комбинацию предметов, предполагая, что любая часть предмета добычи может быть помещена в его сумку.

Цель - реализовать алгоритм для задачи о дробном рюкзаке.

- **Формат ввода / входного файла (input.txt).** В первой строке входных данных задано целое число n - количество предметов, и W - вместимость сумки. Следующие n строк определяют значения веса и стоимости предметов. В i-ой строке содержатся целые числа p_i и w_i – стоимость и вес i-го предмета, соответственно.
- **Ограничения на входные данные.** $1 \leq n \leq 10^3, 0 \leq W \leq 2 \cdot 10^6, 0 \leq p_i \leq 2 \cdot 10^6, 0 \leq w_i \leq 2 \cdot 10^6$ для всех $1 \leq i \leq n$. Все числа - целые.
- **Формат вывода / выходного файла (output.txt).** Выведите максимальное значение стоимости долей предметов, которые помещаются в сумку. Абсолютная погрешность между ответом вашей программы и оптимальным значением должно быть не более 10^{-3}. Для этого выведите свой ответ как минимум с четырьмя знаками после запятой (иначе ваш ответ, хотя и будет рассчитан правильно, может оказаться неверным из-за проблем с округлением).

Листинг кода.

```
def fractional_knapsack(n, W, items):
    # Сортируем предметы по убыванию отношения
стоимости к весу
    items.sort(key=lambda x: x[0] / x[1], reverse=True)

    total_value = 0

    for value, weight in items:
        # Если предмет целиком помещается в сумку,
добавляем его полностью
        if weight <= W:
            total_value += value
```

```
            W -= weight
        else:
            # Если предмет не помещается целиком, берем
долю предмета,
            # которая поместится в сумку
            fraction = W / weight
            total_value += value * fraction
            break

    return total_value

# Чтение входных данных
with open('input.txt', 'r') as file:
    n, W = map(int, file.readline().split())
    items = [list(map(int, line.split())) for line in
file]

# Решение задачи и запись результата в файл вывода
result = fractional_knapsack(n, W, items)
with open('output.txt', 'w') as file:
    file.write("{:.4f}".format(result))
```

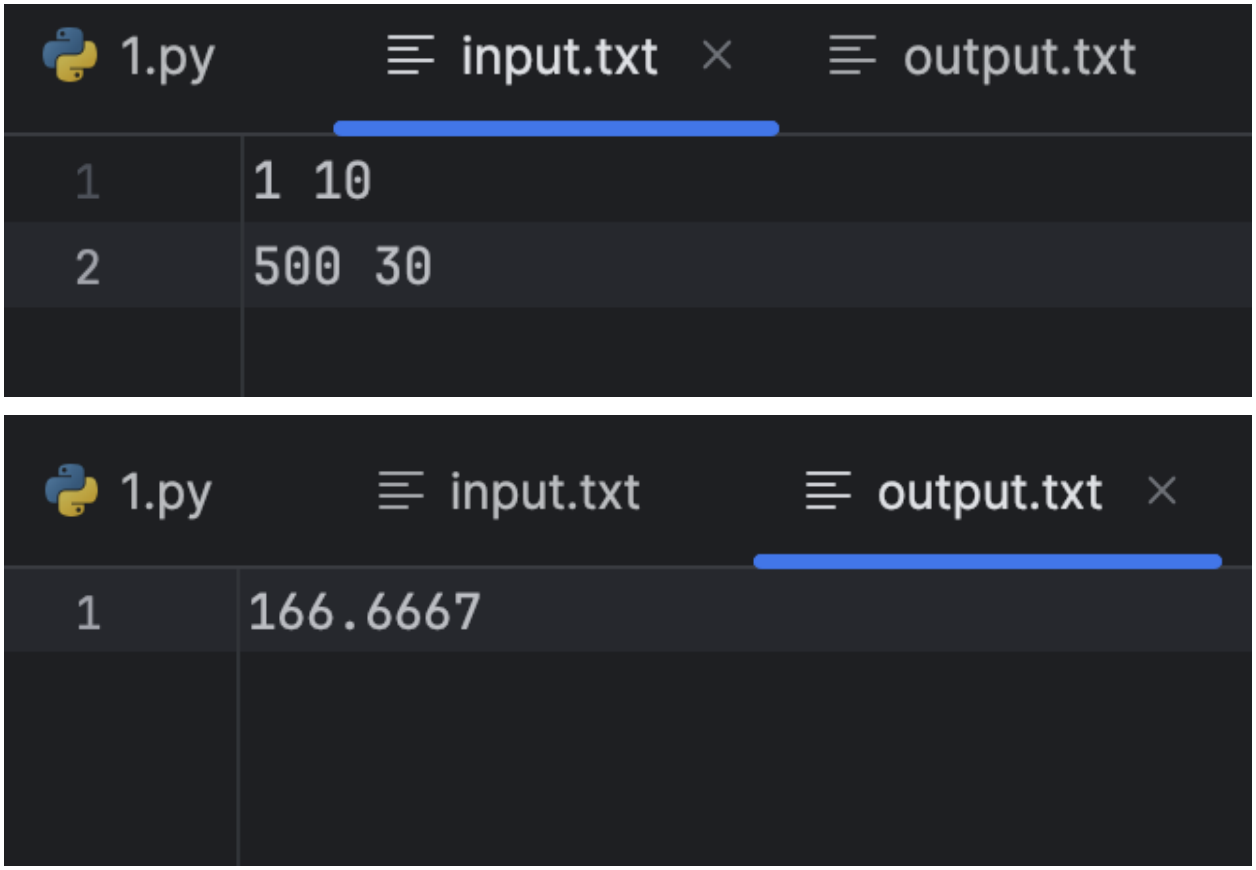
Текстовое объяснение решения.

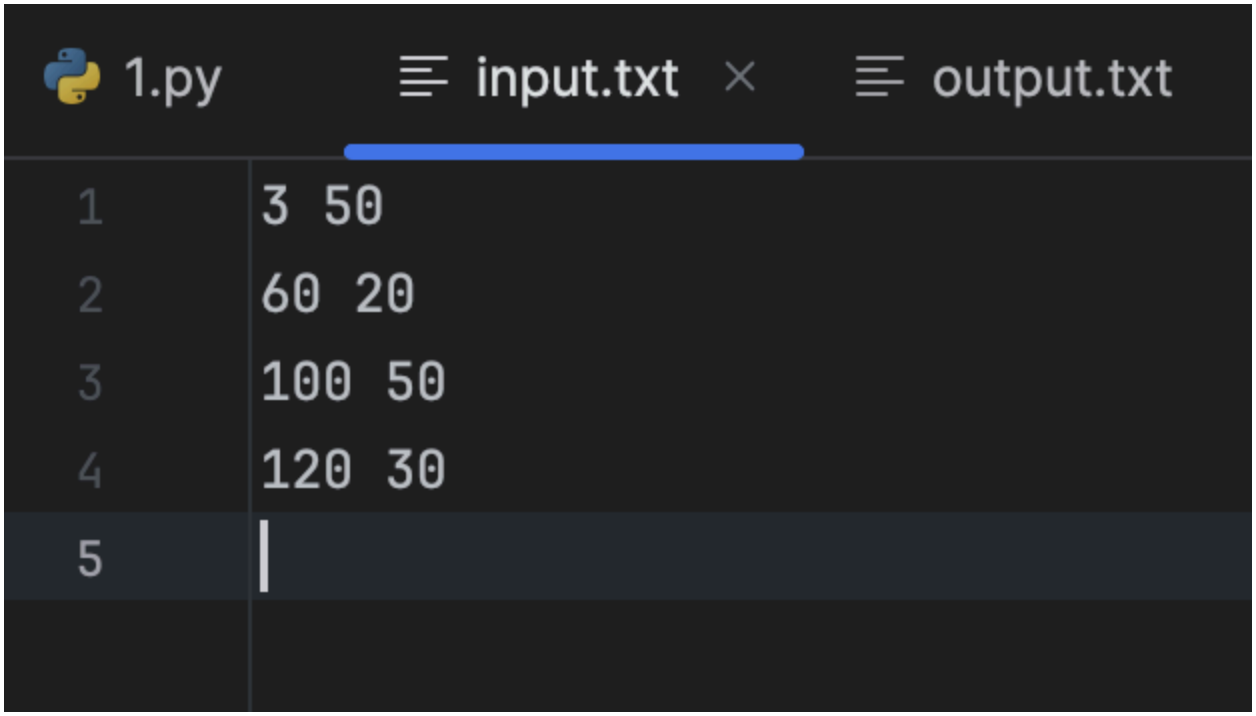
Код решает задачу о дробном рюкзаке, где нужно упаковать предметы с разными весами и стоимостями в рюкзак с ограниченной емкостью, чтобы максимально увеличить общую стоимость предметов в рюкзаке.

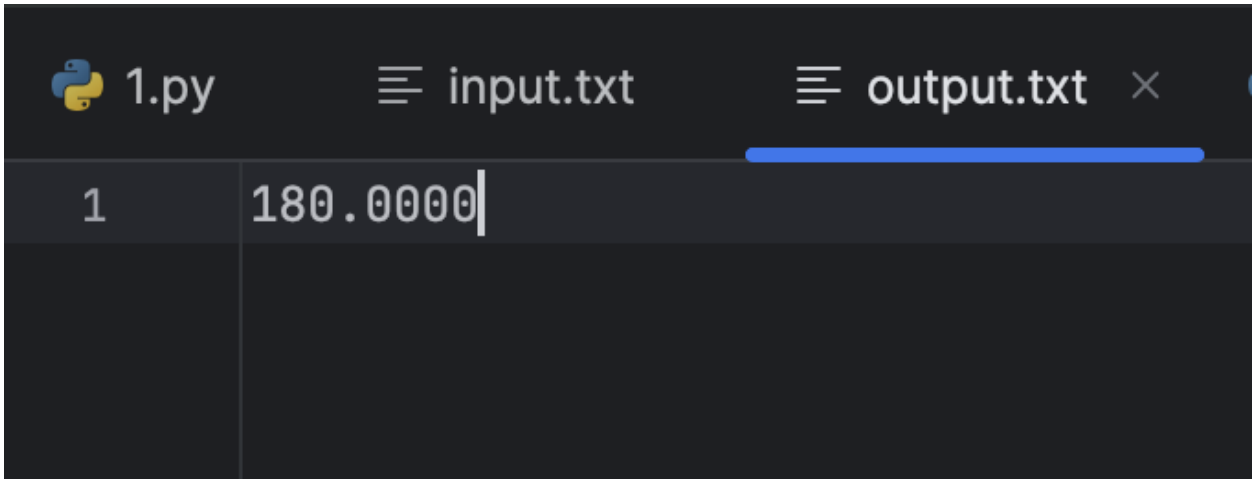
Функция fractional_knapsack сортирует предметы по убыванию отношения стоимости к весу, затем упаковывает предметы в рюкзак, начиная с самых дорогих и тяжелых. Если предмет не помещается целиком, берется доля предмета, которая поместится в рюкзак.

Код также читает входные данные из файла input.txt и записывает результат в файл output.txt с точностью до четырех знаков после запятой.

Результат работы кода на примерах из текста задачи:







	Время выполнения (сек)	Затраты памяти (Мб)
Пример из задачи	0.002499103546142578	5996544

Пример из задачи	00.00203628902819839	5984324

Задача №5 Максимальное количество призов

Текст задачи

Вы организуете веселый конкурс для детей. В качестве призового фонда у вас есть n конфет. Вы хотели бы использовать эти конфеты для раздачи k лучшим местам в конкурсе с естественным ограничением, заключающимся в том, что чем выше место, тем больше конфет. Чтобы осчастливить как можно больше детей, вам нужно найти наибольшее значение k, для которого это возможно.

- **Постановка задачи.** Необходимо представить заданное натуральное число n в виде суммы как можно большего числа попарно различных натуральных чисел. То есть найти максимальное k такое, что n можно записать как $a_1 + a_2 + ... + a_k$, где $a_1, ..., a_k$ - натуральные числа и $a_i \neq a_j$ для всех $1 \leq i < j \leq k$.
- **Формат ввода / входного файла (input.txt).** Входные данные состоят из одного целого числа n.
- **Ограничения на входные данные.** $1 \leq n \leq 10^9$.
- **Формат вывода / выходного файла (output.txt).** В первой строке выведите максимальное число k такое, что n можно представить в виде суммы k попарно различных натуральных чисел. Во второй строке выведите эти k попарно различных натуральных чисел, которые в сумме дают n (если таких представлений много, выведите любое из них).

Листинг кода

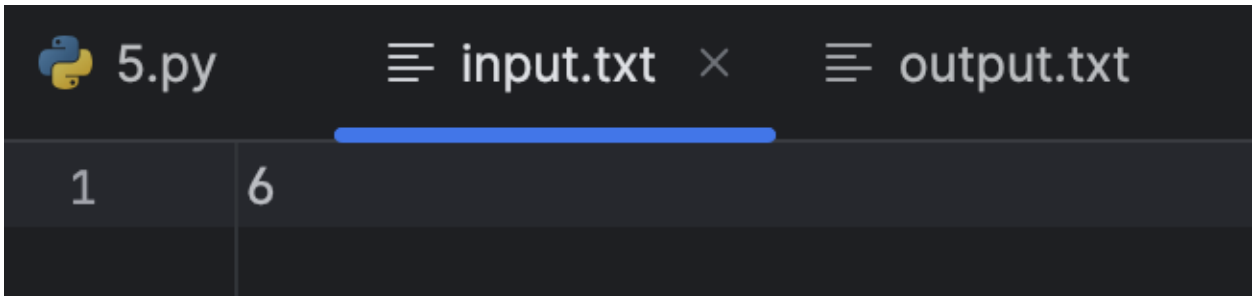
```
def find_maximum_summands(n):
    summands = []
    current_number = 1

    while n > 0:
        if n >= current_number:
            summands.append(current_number)
            n -= current_number
            current_number += 1
        else:
```

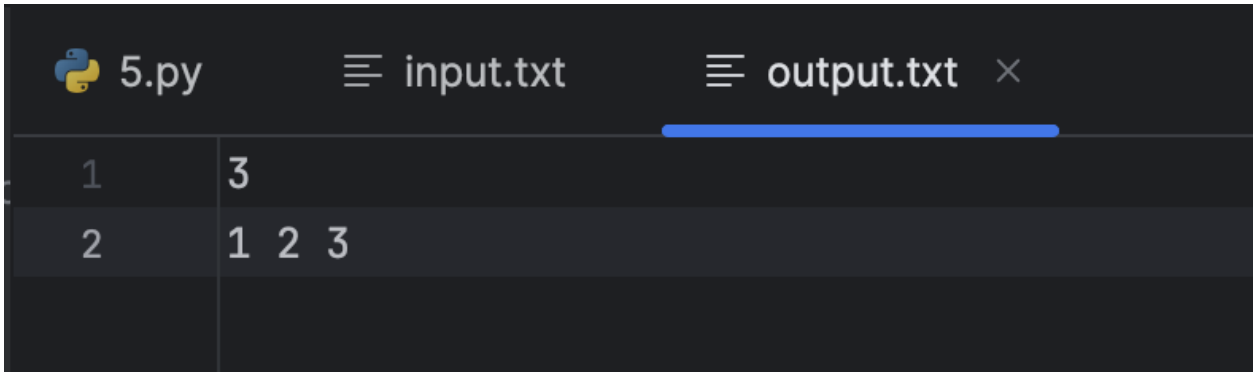
```
            summands[-1] += n
            n = 0

    return summands

# Чтение входных данных
with open('input.txt', 'r') as file:
    n = int(file.readline())

# Решение задачи и запись результата в файл вывода
summands = find_maximum_summands(n)
with open('output.txt', 'w') as file:
    file.write(str(len(summands)) + '\n')
    file.write(' '.join(map(str, summands)))
```

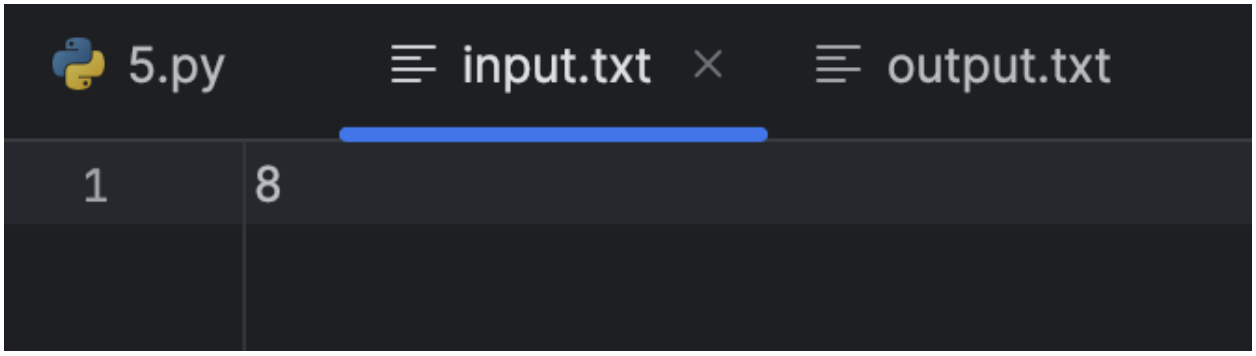
Объяснение кода:

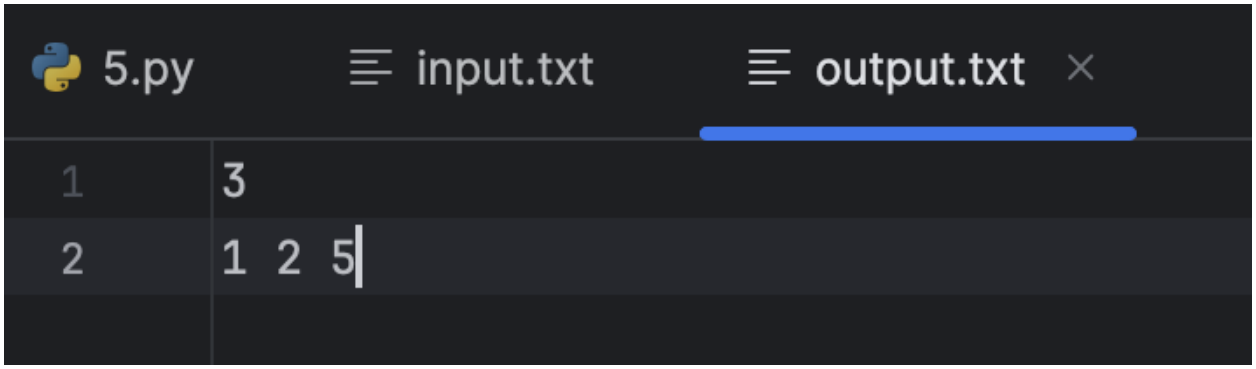
Код находит оптимальный способ разложения числа n на максимальное количество слагаемых. Он работает, добавляя к списку слагаемые, начиная с 1, пока не будет достигнуто значение n. Если оставшееся значение n меньше текущего слагаемого, оно добавляется к последнему слагаемому в списке. Результат записывается в файл output.txt в виде количества слагаемых и самих слагаемых, разделенных пробелами.

Результат работы кода на примерах из текста задачи:









	Время выполнения (сек)	Затраты памяти (Мб)
Пример из задачи	0.00254789207610109 8	610882
Пример из задачи	00.0023681908897319 0	609281

Задача №11 Максимальное количество золота

Текст задачи

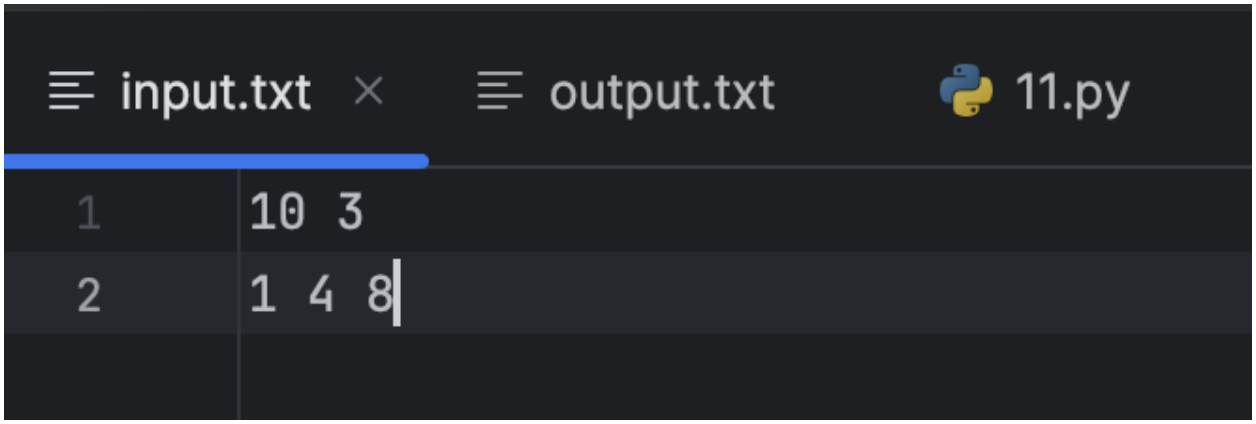
Вам дается набор золотых слитков, и ваша цель - набрать как можно больше золота в свою сумку. Существует только одна копия каждого слитка, и для каждого слитка вы можете либо взять его, либо нет (т.е. вы не можете взять часть слитка).

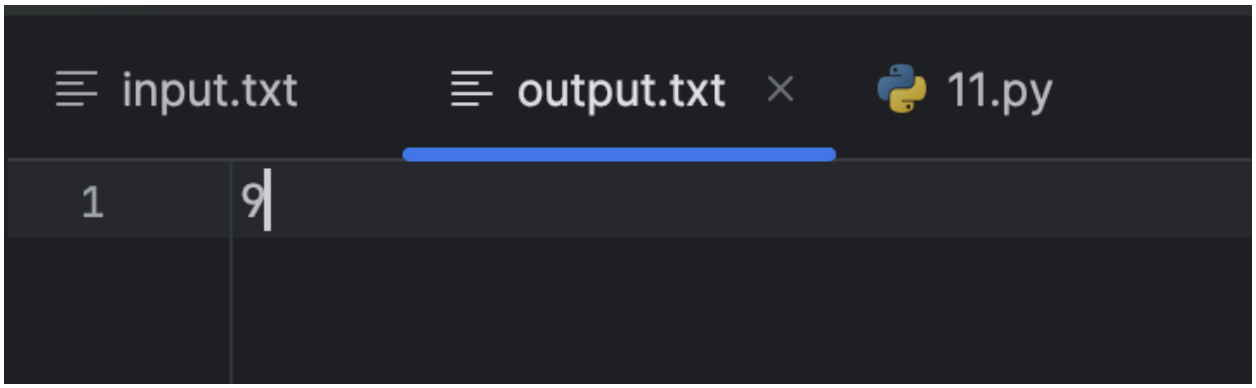
- **Постановка задачи.** Даны n золотых слитков, найдите максимальный вес золота, который поместится в сумку вместимостью W.
- **Формат ввода / входного файла (input.txt).** Первая строка входных данных содержит вместимость W сумки и количество n золотых слитков. В следующей строке записано n целых чисел $w_0, w_1, ..., w_{n-1}$, определяющие вес золотых слитков.
- **Ограничения на входные данные.** $1 \leq W \leq 10^4$, $1 \leq n \leq 300$, $0 \leq w_0, ..., w_{n-1} \leq 10^5$
- **Формат вывода / выходного файла (output.txt).** Выведите максимальный вес золота, который поместится в сумку вместимости W.

Объяснение кода

Код решает задачу о максимальном весе золота, который можно унести в рюкзаке с вместимостью W, используя n слитков золота с известными весами. Функция max_gold_weight использует динамическое программирование для поиска оптимального решения. Она создает двумерный массив dp для хранения результатов и заполняет его, выбирая максимум между взятием каждого слитка и не взятием. Результат записывается в файл output.txt.

Результат работы кода на примерах из текста задачи:





	Время выполнения (сек)	Затраты памяти (Мб)
Пример из задачи	0.00172400474548339 84	6041600

Задача №16. Продавец

Текст задачи.

- **Постановка задачи.** Продавец техники хочет объехать n городов, посетив каждый из них ровно один раз. Помогите ему найти кратчайший путь.
- **Формат ввода / входного файла (input.txt).** Первая строка входного файла содержит натуральное число n – количество городов. Следующие n строк содержат по n чисел – длины путей между городами. В i-й строке j-е число – $a_{i,j}$ – это расстояние между городами i и j.
- **Ограничения на входные данные.** $1 \leq n \leq 13, 0 \leq a_{i,j} \leq 10^6, a_{i,j} = a_{j,i}$, $a_{i,i} = 0$.
- **Формат вывода / выходного файла (output.txt).** В первой строке выходного файла выведите длину кратчайшего пути. Во второй строке выведите nчисел – порядок, в котором нужно посетить города.

Листинг кода:

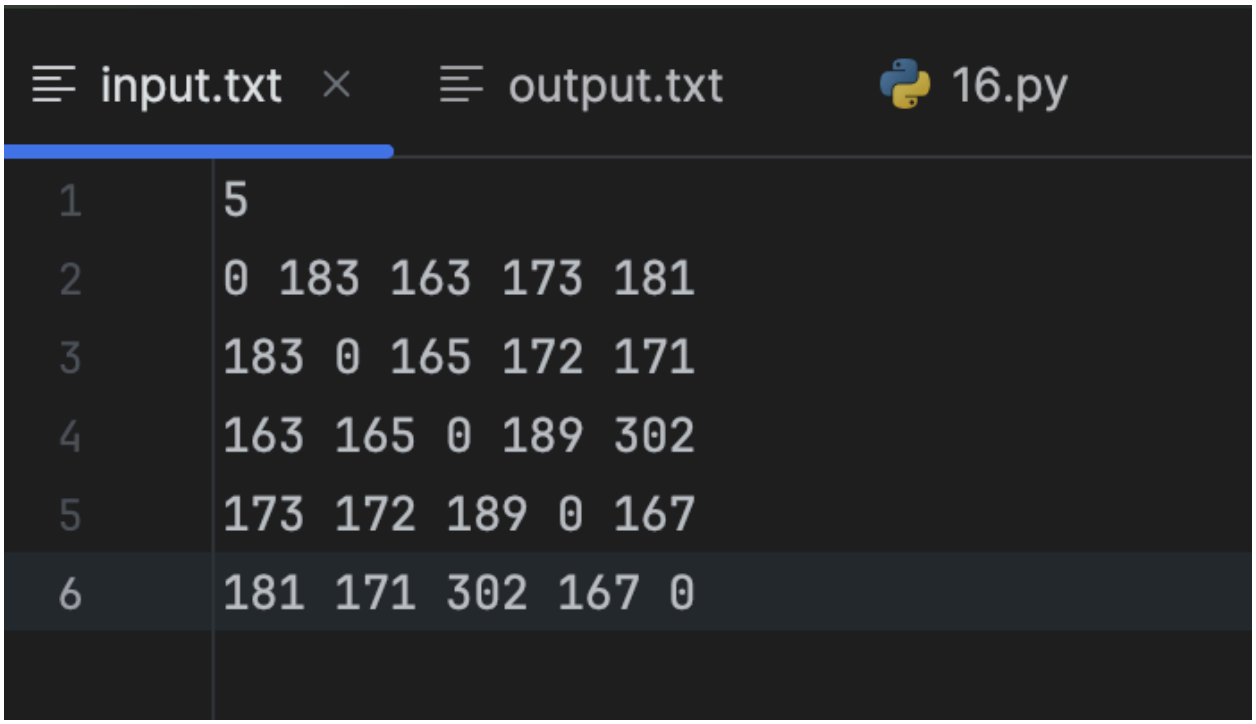
```
from itertools import permutations


def tsp(distances):

    n = len(distances) # Определяет количество городов
(число списков в distances).
    min_distance = float('inf') # Инициализирует
переменную min_distance максимально возможным
значением (бесконечность),
    # чтобы мы могли сравнивать ее с длинами маршрутов
и находить минимальное значение.
    best_order = None # хранится оптимальный порядок
посещения городов.


    for order in permutations(range(1, n)):  # цикл,
```

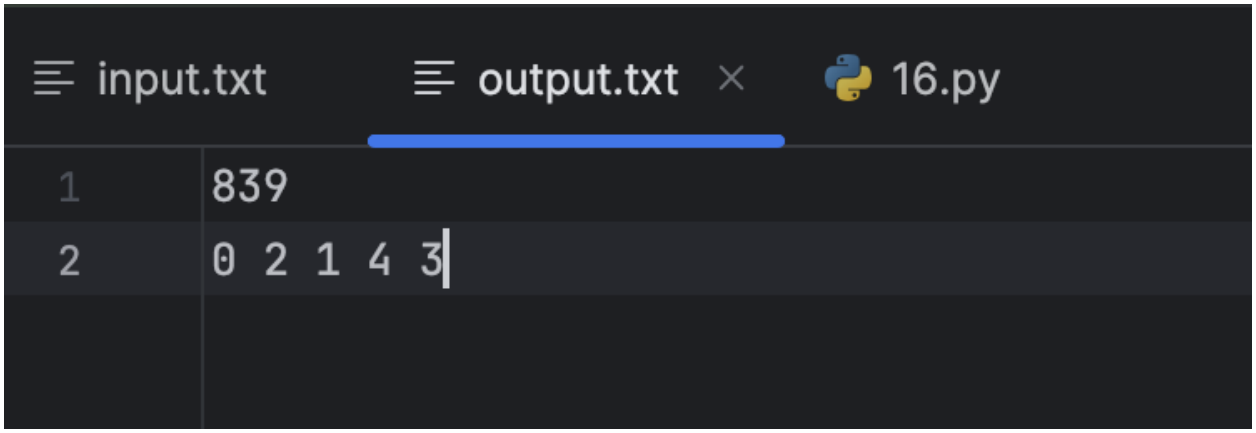
```
который перебирает все возможные перестановки номеров
городов
        # (от 1 до n - 1, так как 0 - это начальный
город).
        total_distance = distances[0][order[0]]  #
вычисляет начальное расстояние от первого города до
первого города в текущей перестановке.
        for i in range(len(order) - 1):
            total_distance +=
distances[order[i]][order[i + 1]] # вычисляет
суммарное расстояние по всем последовательным парам
городов в текущей перестановке.
        total_distance += distances[order[-1]][0] #
Добавляет расстояние от последнего города в
перестановке до исходного города.

        if total_distance < min_distance:    # является
ли текущее расстояние меньше предыдущего минимума.
            min_distance = total_distance
            best_order = (0,) + order

    return min_distance, best_order


with open("input.txt", "r") as f:
    n = int(f.readline())
    distances = []
    for _ in range(n):
        distances.append([int(x) for x in
f.readline().split()])

min_distance, best_order = tsp(distances)

with open("output.txt", "w") as f:
    f.write(str(min_distance) + "\n")
```

```
    f.write(" ".join([str(x) for x in best_order]))
```

Объяснение кода:
Функция permutations позволяет генерировать все возможные перестановки элементов из заданного списка.
Код читает матрицу расстояний между городами из файла input.txt и находит кратчайший путь, который проходит через каждый город ровно один раз и возвращается в исходный город.
Код использует функцию permutations из модуля itertools для перебора всех возможных перестановок городов. Для каждой перестановки вычисляется общая длина пути и сравнивается с минимальным значением, которое хранится в переменной min_distance. В конце функции возвращается минимальная длина пути и соответствующая оптимальная последовательность посещения городов.
Результат записывается в файл output.txt в двух строках: в первой строке — минимальная длина пути, а во второй — порядок посещения городов.

Результат работы кода на примерах из текста задачи:





	Время выполнения (сек)	Затраты памяти (Мб)
Пример из задачи	0.001893010737291012	6130272

Задача №20. Почти полидром.

- **Постановка задачи.** Слово называется палиндромом, если его первая буква совпадает с последней, вторая – с предпоследней и т.д. Например: «abba», «madam», «x».

  Для заданного числа K слово называется почти палиндромом, если в нем можно изменить не более K любых букв так, чтобы получился палиндром. Например, при $K = 2$ слова «$re\underline{a}ctor$», «$kolo\underline{b}ok$», «$madam$» являются почти палиндромами (подчеркнуты буквы, заменой которых можно получить палиндром).

  Подсловом данного слова являются все слова, получающиеся путем вычеркивания из данного нескольких (возможно, одной или нуля) первых букв и нескольких последних. Например, подсловами слова «cat» являются слова «c», «a», «t», «ca», «at» и само слово «cat» (а «ct» подсловом слова «cat» не является).

  Требуется для данного числа K определить, сколько подслов данного слова S являются почти палиндромами.

- **Формат входного файла (input.txt).** В первой строке входного файла вводятся два натуральных числа: N – длина слова и K. Во второй строке записано слово S, состоящее из N строчных английских букв.

- **Ограничения на входные данные.** $1 \leq N \leq 5000, 0 \leq K \leq N$.

- **Формат выходного файла (output.txt).** В выходной файл требуется вывести одно число – количество подслов слова S, являющихся почти палиндромами (для данного K).

Листинг кода:

```
def is_almost_palindrome(word, k):
    n = len(word)
    for i in range(n // 2):
        if word[i] != word[n - 1 - i]:
            k -= 1
            if k < 0:
                return False
    return True


def count_almost_palindromes(word, k):
```

```
    count = 0
    for i in range(len(word)):
        for j in range(i + 1, len(word) + 1):
            substring = word[i:j]
            if is_almost_palindrome(substring, k):
                count += 1
    return count


with open("input.txt", "r") as f:
    n, k = map(int, f.readline().split())
    word = f.readline().strip()

count = count_almost_palindromes(word, k)

with open("output.txt", "w") as f:
    f.write(str(count))
```

Объяснение кода:

1. Функция is_almost_palindrome(word, k) проверяет, является ли строка wordпочти палиндромом с максимальным количеством изменений k. Она сравнивает символы с начала и конца строки, уменьшая k при несовпадении. Если k становится отрицательным, строка не является почти палиндромом.
2. Функция count_almost_palindromes(word, k) считает количество почти палиндромов в строке word с максимальным количеством изменений k. Она перебирает все подстроки строки word и использует функцию is_almost_palindrome для проверки, является ли подстрока почти палиндромом.

Результат работы кода на примерах из текста задачи:

input.txt × output.txt 20.py

```
1  5 1
2  abcde
```

input.txt output.txt × 20.py

```
1  12
```

input.txt × output.txt 20.py

```
1  3 3
2  aaa
```

input.txt output.txt × 20.py

```
1  6
```

	Время выполнения (сек)	Затраты памяти
Пример из задачи	0.00253677368164062 5	5865472
Пример из задачи	0.00179886817932128 9	6045696