

# **Алгоритм и анализ сложности**

## Лабораторная работа 1

Крюков Никита А. (АТ-01)  
РИ-230916

Вариант: 14

## Задание 1. Ближайший ноль.

Ограничение по времени: 1.6 с. Ограничение по памяти: 400 Мб.

Улица, на которой хочет жить Тимофей, имеет длину  $n$ , то есть состоит из  $n$  одинаковых идущих подряд участков. На каждом участке либо уже построен дом, либо участок пустой. Тимофей ищет место для строительства своего дома. Он очень общителен и не хочет жить далеко от других людей, живущих на этой улице.

Чтобы оптимально выбрать место для строительства, Тимофей хочет для каждого участка знать расстояние до ближайшего пустого участка. (для пустого участка эта величина будет равна нулю – расстояние до самого себя)

Ваша задача – помочь Тимофею посчитать искомые расстояния. Для этого у вас есть карта улицы. Дома в городе Тимофея нумеровались в том порядке, в котором строились, поэтому их номера на карте никак не упорядочены. Пустые участки обозначены нулями.

Формат входных данных:

В первой строке дана длина улицы –  $n$  ( $1 \leq n \leq 10^6$ ). В следующей строке записаны  $n$  целых неотрицательных чисел – номера домов и обозначения пустых участков на карте (нули). Гарантируется, что в последовательности есть хотя бы один ноль. Номера домов (положительные числа) уникальны и не превосходят  $10^9$ .

Формат выходных данных:

Для каждого из участков выведите расстояние до ближайшего нуля. Числа выводите в одну строку, разделяя их пробелами.

Примеры:

Стандартный ввод	Стандартный вывод
5 0 1 4 9 0	0 1 2 1 0
6 0 7 9 4 8 20	0 1 2 3 4 5

Код:

```
1 usage
83  def main() -> None:
84      # Входные данные.
85
86      n = int(input())
87      houses = str(input())
88
89      start_time = time.perf_counter() # Таймер (для отслеживания затраченного времени).
90
91      # Выполнения основной программы.
92
93      distances = calculate_distances(n, houses)
94      print(distances)
95
96      # Вывод информации о времени и памяти.
97
98      print("=" * 42)
99      stop_time = time.perf_counter()
100     print(f"    Время выполнения: {stop_time - start_time:0.5f} секунд.")
101     final_memory = get_memory_usage()
102     print(f"    Использовано памяти: {final_memory:.2f} Mb.")
103     print("=" * 42)
104
105
106  if __name__ == "__main__":
107     main()
108
```

```

def calculate_distances(n: int, houses: str) -> str:
    """Вычисляет минимальные расстояния до ближайшего свободного участка для каждого дома."""
    if not (1 <= n <= 10 ** 6):
        raise ValueError("Переменная n должно быть в диапазоне от 1 до 10^6.")

    houses = format_houses(houses)

    distances = [n] * n
    left_distances = [n] * n
    right_distances = [n] * n

    for i in range(n):
        if houses[i] == 0:
            left_distances[i] = 0
        elif i > 0:
            if left_distances[i] > left_distances[i - 1] + 1:
                left_distances[i] = left_distances[i - 1] + 1

    for i in range(n - 1, -1, -1):
        if houses[i] == 0:
            right_distances[i] = 0
        elif i < n - 1:
            if right_distances[i] > right_distances[i + 1] + 1:
                right_distances[i] = right_distances[i + 1] + 1

    for i in range(n):
        if left_distances[i] < right_distances[i]:
            distances[i] = left_distances[i]
        else:
            distances[i] = right_distances[i]

    distances = format_distances(distances)

    return distances

```

2 usages

```
def calculate_left_distances(houses: List[int], n: int, i: int, left_distances: List[int]) -> None:
    if i < 0:
        return
    if houses[i] == 0:
        left_distances[i] = 0
    elif i > 0:
        calculate_left_distances(houses, n, i - 1, left_distances)
        left_distances[i] = left_distances[i - 1] + 1
```

2 usages

```
def calculate_right_distances(houses: List[int], n: int, i: int, right_distances: List[int]) -> None:
    if i >= n:
        return
    if houses[i] == 0:
        right_distances[i] = 0
    elif i < n - 1:
        calculate_right_distances(houses, n, i + 1, right_distances)
        right_distances[i] = right_distances[i + 1] + 1
```

1 usage

```
def calculate_distances_rec(n: int, houses: str) -> str:
    """Рекурсивно вычисляет минимальные расстояния до ближайшего свободного участка для каждого дома."""
    if not (1 <= n <= 10 ** 6):
        raise ValueError("Переменная n должна быть в диапазоне от 1 до 10^6.")

    houses = format_houses(houses)

    distances = [n] * n
    left_distances = [n] * n
    right_distances = [n] * n

    calculate_left_distances(houses, n, n - 1, left_distances)
    calculate_right_distances(houses, n, 0, right_distances)

    for i in range(n):
        if left_distances[i] < right_distances[i]:
            distances[i] = left_distances[i]
        else:
            distances[i] = right_distances[i]

    return format_distances(distances)
```

```

def format_distances(distances: List[int]) -> str:
    """Преобразует список расстояний в строку, где расстояния разделены пробелами."""
    result_array = ""

    for distance in distances:
        result_array += str(distance) + " "

    result_array = result_array.strip()

    return result_array

```

16 usages

```

def format_houses(houses: str) -> List[int]:
    """Преобразует строку, содержащую номера домов, в список целых чисел."""
    houses = houses.split() # Разделяем строку на список значений
    seen_houses = set()
    formatted_houses = [] # Список для хранения преобразованных номеров домов

    for house in houses:
        house_num = int(house) # Преобразуем строку в целое число
        if house_num > 10 ** 9:
            raise ValueError("Номера домов не могут превосходить 10^9.")
        if house_num != 0 and house_num in seen_houses:
            raise ValueError("Номера домов должны быть уникальными.")
        seen_houses.add(house_num)
        formatted_houses.append(house_num)

    if 0 not in formatted_houses:
        raise ValueError("В последовательности должен быть хотя бы один ноль.")

    return formatted_houses

```

```

from typing import List
import time
import os
import psutil

1 usage
def get_memory_usage() -> float:
    """Получение используемой памяти в мегабайтах."""
    process = psutil.Process(os.getpid())
    mem_info = process.memory_info()
    return mem_info.rss / (1024 * 1024)

```

## Тесты (Проходят успешно):

```
class TestCalculateDistances(unittest.TestCase):
    def test_format_houses_basic(self):
        """1. Проверка на правила форматирования."""
        self.assertEqual(format_houses("0 1 4 9 0"), second: [0, 1, 4, 9, 0])
        self.assertEqual(format_houses("0 7 9 4 8 2"), second: [0, 7, 9, 4, 8, 2])
        self.assertEqual(format_houses("1 0 2 3 0 5 4 7 8 11"), second: [1, 0, 2, 3, 0, 5, 4, 7, 8, 11])
        self.assertEqual(format_houses(" ".join(str(i) for i in range(100))), list(range(100)))
        self.assertEqual(format_houses(" ".join(str(i) for i in range(1000))), list(range(1000)))
        self.assertEqual(format_houses(" ".join(str(i) for i in range(1000000))), list(range(1000000)))

    def test_values_format_houses(self):
        """2. В последовательности должен быть ноль."""
        self.assertRaises(ValueError, format_houses, "8 4 3 1 6 7")
        self.assertRaises(ValueError, format_houses, "1 2 3 6 7 8")

    def test_number_format_houses(self):
        """3. Номера домов не могут превосходить 10^9."""
        self.assertRaises(ValueError, format_houses, "1 2 3 6 7 1000000000000")
        self.assertRaises(ValueError, format_houses, "1 2 900000000000 6 7 8")

    def test_seen_format_houses(self):
        """4. Номера домов должны быть уникальными."""
        self.assertRaises(ValueError, format_houses, "1 2 3 6 7 1")
        self.assertRaises(ValueError, format_houses, "6 5 4 1 2 4")
        self.assertRaises(ValueError, format_houses, "1 2 3 4 2 6")

    def test_calculate_distances(self):
        """5. Проверка расчёта дистанции."""
        self.assertEqual(calculate_distances(n: 5, houses: "0 1 4 9 0"), second: "0 1 2 1 0")
        self.assertEqual(calculate_distances(n: 6, houses: "0 7 9 4 8 20"), second: "0 1 2 3 4 5")
        self.assertEqual(calculate_distances(n: 13, houses: "0 2 3 1 11 13 14 7 9 4 8 20 19"), second: "0 1 2 3 4 5 6 7 8 9 10 11 12")
        self.assertEqual(calculate_distances(n: 13, houses: "0 2 3 1 11 13 14 7 0 4 8 20 19"), second: "0 1 2 3 4 3 2 1 0 1 2 3 4")
        self.assertEqual(calculate_distances(n: 100, houses: "1 2 3 4 5 6 7 8 0 9 10 11 12 13 14 15 16 17 18 19 20 21 0 22 23 24 25 26 27 28 29"), second: "0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30")
        # self.assertEqual(calculate_distances(100000, "1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32"), second: "0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32")

    def test_format_distances(self):
        """6. Проверка итогового форматирования."""
        self.assertEqual(format_distances([1, 2, 3, 1, 2, 3]), second: "1 2 3 1 2 3")
        self.assertEqual(format_distances([0, 1, 2, 0]), second: "0 1 2 0")
        self.assertEqual(format_distances([1, 3, 4, 6, 7, 10]), second: "1 3 4 6 7 10")
```

Ran 6 tests in 0.298s

OK

В данном задании на мой взгляд наилучшим является вариант с итеративным подходом, а не с рекурсивным.

Тесты на 1000 значений показали такие значения:

Итеративный способ:

```
=====
Время выполнения: 0.00101 секунд.
Использовано памяти: 16.18 Mb.
=====
```

## Рекурсивный:

Время выполнения: 0.00336 секунд.

Использовано памяти: 15.86 Мб.

Тесты на 100000 значений показали такие результаты:

### Итеративный способ:

Время выполнения: 0.05530 секунд.

Использовано памяти: 18.35 Мб.

### Рекурсивный способ:

Traceback (most recent call last): [Explain with AI](#)

```
File "D:\PycharmProjects\algorithms_and_complexity_analysis\quest_1\quest_1.py", line 156, in <module>
    main()
```

```
File "D:\PycharmProjects\algorithms_and_complexity_analysis\quest_1\quest_1.py", line 142, in main
    distances = calculate_distances_rec(n, houses)
```

```
File "D:\PycharmProjects\algorithms_and_complexity_analysis\quest_1\quest_1.py", line 85, in calculate_distances_rec
    calculate_left_distances(houses, n, n - 1, left_distances)
```

```
File "D:\PycharmProjects\algorithms_and_complexity_analysis\quest_1\quest_1.py", line 62, in calculate_left_distances
    calculate_left_distances(houses, n, i - 1, left_distances)
```

```
File "D:\PycharmProjects\algorithms_and_complexity_analysis\quest_1\quest_1.py", line 62, in calculate_left_distances
    calculate_left_distances(houses, n, i - 1, left_distances)
```

File "D:\PycharmProjects\algorithms\_and\_complexity\_analysis\quest\_1\quest\_1.py", line 62, in calculate\_left\_distances  
calculate\_left\_distances(houses, n, i - 1, left\_distances)

```
[Previous line repeated 994 more times]
```

```
RecursionError: maximum recursion depth exceeded
```

### Ход решения задачи:

Функция `main` – это точка входа. Считывает кол-во домов и строку с номером домов, вызывает функцию `calculate_distances`. Выводит результат и время выполнения программы.

Функция `calculate_distances` вычисляет минимальные расстояния до ближайшего свободного участка для каждого дома. Проверяется, что переменная `n` находится в диапазоне (по условию). В методе происходит вызов функции `format_houses`, которая принимает строку, содержащую номера домов, и преобразует её в список целых чисел. Создаются три массива `distances`, `left_distances` и `right_distances`. Проходим по массиву слева направо и вычисляем минимальные расстояния до ближайшего свободного участка (нуля) для каждого дома. Для каждого дома выбирается минимальное расстояние из `left_distances` и `right_distances`. В методе `calculate_distances` также используется функция `format_distances` для преобразования списка расстояний в строку. Функция `calculate_distances` возвращает в `main` строку, содержащую расстояния разделенные пробелами.

Если видно плохо или хочется протестировать, то можете посмотреть мой код по заданию тут: [https://github.com/ytkinroman/lab\\_1\\_complexity/tree/main/quest\\_1](https://github.com/ytkinroman/lab_1_complexity/tree/main/quest_1)



### Задание 3. Симметрическая разность.

#### Условие задачи:

Ограничение по времени: 2 с. Ограничение по памяти: 64 Мб.

На вход подается множество чисел в диапазоне от 1 до 20000, разделенных пробелом. Они образуют множество А. Затем идет разделитель – число 0 и на вход подается множество чисел В, разделенных пробелом, 0 – признак конца описания множества (во множество не входит). Необходимо вывести множество  $A \Delta B$  – симметрическую разность множеств А и В в порядке возрастания элементов. В качестве разделителя используйте пробел. В случае, если множество пусто, вывести 0.

Формат входных данных:

1 2 3 4 5 0 1 7 5 8 0

Формат выходных данных:

2 3 4 7 8

Примеры:

Стандартный ввод	Стандартный вывод
1 2 6 8 7 3 0 4 1 6 2 3 9 0	4 7 8 9

Замечание. Для вывода можно использовать любой алгоритм сортировки.

#### Ход решения задачи:

Функция main – это точка входа. Считывает строку, содержащую числа. Вызывает функцию find\_symmetric\_difference для вычисления симметрической разности, передаёт туда строку. После выполнения функции find\_symmetric\_difference выводит результат и время выполнения программы.

Функция find\_symmetric\_difference разделяет строку на две части (множества) с помощью метода split ("0"). Преобразует каждую часть в список целых чисел с помощью функции string\_to\_list. Находит симметрическую разность между двумя списками с помощью функции find\_symmetric\_difference\_lists. Если симметрическая разность пуста, возвращает "0" или сортирует симметрическую разность с помощью функции quick\_sort\_sym\_diff и преобразует результат в строку с помощью функции get\_list\_to\_string.

Функция find\_symmetric\_difference\_lists находит симметрическую разность между двумя списками целых чисел. Добавляет в результат элементы из первого списка, которые не встречаются во втором списке. Добавляет в результат элементы из второго списка, которые не встречаются в первом списке.

Функция quick\_sort\_sym\_diff реализует алгоритм быстрой сортировки для списка целых чисел. Рекурсивно сортирует левую и правую части и объединяет их.

Код:

```

1 usage
def main() -> None:
    input_str = input()

    start_time = time.perf_counter()

    sym_diff = find_symmetric_difference(input_str)
    print(sym_diff)

    print("")

    stop_time = time.perf_counter()

    print(f"Время выполнения: {stop_time - start_time:0.5f} секунд.")

    final_memory = get_memory_usage()
    print(f"Использовано памяти: {final_memory:.2f} Mb.")

> if __name__ == "__main__":
    main()

```

```

1 usage
def find_symmetric_difference_lists(list_a: List[int], list_b: List[int]) -> List[int]:
    """Нахождение симметрической разности."""
    sym_diff = []

    for item in list_a:
        if item not in list_b:
            sym_diff.append(item)

    for item in list_b:
        if item not in list_a:
            sym_diff.append(item)

    return sym_diff

```

```

10 usages
def find_symmetric_difference(string_values: str):
    """Нахождение симметрической разности."""
    parts = string_values.split("0")

    list_a = string_to_list(parts[0].strip())
    list_b = string_to_list(parts[1].strip())

    sym_diff = find_symmetric_difference_lists(list_a, list_b)

    if not sym_diff:
        return "0"
    else:
        sort_diff = quick_sort_sym_diff(sym_diff)

        result = get_list_to_string(sort_diff)

        return result

```

3 usages

```
def quick_sort_sym_diff(arr: List[int]) -> List[int]:
    """Алгоритм быстрой сортировки."""
    if len(arr) <= 1:
        return arr
    else:
        element = arr[0]

        left = []
        for x in arr:
            if x < element:
                left.append(x)

        middle = []
        for x in arr:
            if x == element:
                middle.append(x)

        right = []
        for x in arr:
            if x > element:
                right.append(x)

        return quick_sort_sym_diff(left) + middle + quick_sort_sym_diff(right)
```

▼ def get\_list\_to\_string(input\_list: List[int]) -> str:

"""Преобразует список целых чисел в строку."""  
result = ""

for distance in input\_list:  
 result += str(distance) + " "

result\_array = result.strip()

return result\_array

2 usages

▼ def string\_to\_list(input\_str: str) -> List[int]:

"""Преобразует строку, содержащую числа, разделенные пробелами, в множество целых чисел."""  
input\_str = input\_str.split()

result\_list = []  
for i in input\_str:  
 result\_list.append(int(i))

return result\_list

```

1 usage
2
3 def get_memory_usage():
4     """Получение используемой памяти в мегабайтах."""
5     process = psutil.Process(os.getpid())
6     mem_info = process.memory_info()
7
8     return mem_info.rss / (1024 * 1024)

```

## Тесты (Проходят успешно):

```

import unittest
from quest_3 import find_symmetric_difference

class TestFindSymmetricDifference(unittest.TestCase):
    def test_find_symmetric_difference(self):
        """Проверка расчёта симметрической разности."""
        self.assertEqual(find_symmetric_difference('1 2 3 4 0 3 4 5 6 0'), '1 2 5 6')
        self.assertEqual(find_symmetric_difference('11 1 2 3 4 5 0 1 7 5 8 0'), '2 3 4 7 8 11')
        self.assertEqual(find_symmetric_difference('1 2 6 8 7 3 0 4 1 6 2 3 9'), '4 7 8 9')
        self.assertEqual(find_symmetric_difference('1 0 3'), '1 3')
        self.assertEqual(find_symmetric_difference('1 2 3 0 1 2 3'), '0')
        self.assertEqual(find_symmetric_difference('0 1'), '1')
        self.assertEqual(find_symmetric_difference('0'), '0')
        self.assertEqual(find_symmetric_difference('3 5 6 7 8 9 11 13 16 17 19 20 23 24 26 28 29 30 31 32 33 34 36 37 42 43 44 45 46 47 48 49 50 51 52 55'
            '56 61 62 63 66 67 68 69 70 72 74 77 78 80 81 82 85 86 89 91 93 95 96 97 98 99 103 104 105 106 107 108 109'
            '110 111 114 115 116 117 120 122 123 124 125 127 128 129 130 131 138 140 141 142 143 144 145 146 148 149 150'
            '151 152 153 154 155 156 157 160 161 163 164 165 166 168 169 170 175 176 177 178 180 181 184 185 186 187 188'
            '189 192 193 194 195 196 199 200 202 203 205 206 207 209 210 211 212 213 214 215 217 218 219 220 221 225 226 228'
            '229 230 232 233 234 235 236 238 240 241 242 243 244 245 246 247 248 250 253 254 255 256 257 259 262 263 264 268 269'
            '270 271 273 274 275 276 279 281 282 284 287 288 289 294 295 296 297 298 299 300 0 1 2 3 5 6 7 8 10 11 14 15 16 18 19'
            '20 21 24 25 26 27 28 29 31 33 34 35 36 37 38 39 41 42 44 45 46 47 48 49 52 53 55 56 57 58 59 60 62 65 67 68 70 71 72'
            '73 74 76 77 79 80 82 83 84 86 87 88 89 90 91 96 97 98 99 100 102 103 105 106 107 109 110 111 113 114 115 116 117 118'
            '119 121 123 124 125 128 129 130 131 133 134 138 142 143 144 145 146 147 148 151 152 153 159 160 161 162 163 164 166 167'
            '168 169 170 172 174 175 176 177 179 180 181 182 183 184 185 186 187 188 192 193 195 197 199 200 201 202 203 204 205 206'
            '207 209 210 211 212 213 220 221 224 229 231 232 233 237 238 243 244 246 248 251 252 253 255 256 257 258 259 260 262 265 266'
            '268 271 272 274 275 276 277 278 281 283 285 286 288 289 290 291 292 293 294 298 299 300'), '2 5 6 7 8 9 11 13 16 17 19 23 24 26 28 29')

```

Ran 1 test in 0.001s

OK

Дополнительно результат на самый тяжёлый тест  
(Более 300 значений):

Время выполнения: 0.00010 секунд.  
Использовано памяти: 15.73 Mb.

Если видно плохо или хочется протестировать, то можете посмотреть мой код по заданию тут: [https://github.com/ytkinroman/lab\\_1\\_complexity/tree/main/quest\\_3](https://github.com/ytkinroman/lab_1_complexity/tree/main/quest_3)

## Задание 6. Вычисление полинома.

### Условие задачи:

Ограничение по времени: 1 с. Ограничение по памяти: 16 Мб.

Вычисление полинома – необходимая операция для многих алгоритмов. Нужно вычислить значение полинома

$$a_n x^n + a_{n-1} x^{n-1} + \dots + a_2 x^2 + a_1 x^1 + a_0$$

Так как число  $n$  может быть достаточно велико, требуется вычислить значение полинома по модулю  $M$ . Сделать это предлагается для нескольких значений аргумента.

#### Формат входных данных:

Первая строка файла содержит три числа – степень полинома  $2 \leq N \leq 100000$ , количество вычисляемых значений аргумента  $1 \leq M \leq 10000$  и модуль  $10 \leq MOD \leq 10^9$ .

Следующие  $N+1$  строк содержат значения коэффициентов полинома  $0 \leq a_i \leq 10^9$

В очередных  $M$  строках содержатся значения аргументов  $0 \leq x_i \leq 10^9$ .

#### Формат выходных данных:

Выходной файл должен состоять из ровно  $M$  строк – значений данного полинома при заданных значениях аргументов по модулю  $MOD$ .

#### Примеры:

Стандартный ввод	Стандартный вывод
2 5 10	4
1	0
5	8
4	8
0	0
1	
2	
3	
4	

### Ход решения задачи:

Функция `main` – это точка входа. Считывает и преобразует необходимые для вычислений значения (степень полинома, количество вычисляемых значений, модуль, значения коэффициентов). После чего вызывает метод `polynomial_values` и передаёт туда полученные значения.

Функция `polynomial_values` рекурсивно вычисляет значения полинома для каждого значения  $x$  из списка. Она использует функцию `poly_value` для вычисления значения полинома для каждого элемента списка. Функция `poly_value` использует рекурсию для вычисления значения полинома. Она последовательно умножает коэффициенты на соответствующие степени  $x$  и суммирует результаты.

Функция `get_result` в методе `main` просто выводит результаты, полученные из функции `polynomial_values`.

### Код

```

1 usage
def main() -> None:
    input_line = input().split()

    N = int(input_line[0])
    M = int(input_line[1])
    MOD = int(input_line[2])

    coefficients = []
    for i in range(N + 1):
        c = int(input())
        coefficients.append(c)

    x_values = []
    for i in range(M):
        x = int(input())
        x_values.append(x)

    start_time = time.perf_counter()

    results = polynomial_values(coefficients, x_values, MOD)

    get_result(results)

    stop_time = time.perf_counter()

    print(f"Время выполнения: {stop_time - start_time:0.5f} секунд.")

    final_memory = get_memory_usage()
    print(f"Использовано памяти: {final_memory:.2f} Mb.")

if __name__ == "__main__":
    main()

```

1 usage

```
def get_memory_usage():  
    """Получение используемой памяти в мегабайтах."""  
    process = psutil.Process(os.getpid())  
    mem_info = process.memory_info()  
    return mem_info.rss / (1024 * 1024)
```

2 usages

```
def poly_value(coefficients: list, x: int, mod: int, power_of_x: int = 1) -> int:  
    """Вычисляет значение полинома для заданного значения с использованием рекурсии."""  
    if not coefficients:  
        return 0  
    else:  
        return (coefficients[-1] * power_of_x + poly_value(coefficients[:-1], x, mod, (power_of_x * x) % mod)) % mod
```

5 usages

```
def polynomial_values(coefficients: list, x_values: list, mod: int) -> list:  
    """Вычисляет значения полинома для списка значений."""  
    if not x_values:  
        return []  
    else:  
        return [poly_value(coefficients, x_values[0], mod)] + polynomial_values(coefficients, x_values[1:], mod)
```

1 usage

```
def get_result(results: list) -> None:  
    """Выводит результаты из списка."""  
    for result in results:  
        print(result)
```

## Тесты

```
import unittest
from quest_6 import polynomial_values

> class TestComputePolynomialValues(unittest.TestCase):
>     def test_polynomial_values_1(self):
        N = 2
        M = 5
        MOD = 10
        coefficients = [1, 5, 4]
        x_values = [0, 1, 2, 3, 4]
        expected_results = [4, 0, 8, 8, 0]
        self.assertEqual(polynomial_values(coefficients, x_values, MOD), expected_results)

>     def test_polynomial_values_2(self):
        N = 5
        M = 9
        MOD = 10
        coefficients = [1, 0, 0, 0, 0, 0]
        x_values = [1, 2, 3, 4, 5, 6, 7, 8, 9]
        expected_results = [1, 2, 3, 4, 5, 6, 7, 8, 9]
        self.assertEqual(polynomial_values(coefficients, x_values, MOD), expected_results)
```

Ran 2 tests in 0.001s

Самый сложный тест:

Время выполнения: 0.00005 секунд.  
Использовано памяти: 15.37 Мб.

Если видно плохо или хочется протестировать, то можете посмотреть мой код по заданию тут: [https://github.com/ytkinroman/lab\\_1\\_complexity/tree/main/quest\\_6](https://github.com/ytkinroman/lab_1_complexity/tree/main/quest_6)



## Задание 8. Магараджа.

### Условие задачи:

Ограничение по времени: 1 с. Ограничение по памяти: 16 Mb.

Магараджа — это шахматная фигура, сочетающая возможности ферзя и коня. Таким образом, магараджа может ходить и бить на любое количество клеток по диагонали, горизонтали и вертикали (т.е. как ферзь), а также либо на две клетки по горизонтали и на одну по вертикали, либо на одну по горизонтали и на две по вертикали (как конь).

Ваша задача — найти число способов расставить на доске  $N$  на  $N$  ровно  $K$  магараджей так, чтобы они не били друг друга.

#### Формат входных данных:

Входной файл INPUT.TXT содержит два целых числа:  $N$  и  $K$  ( $1 \leq K \leq N \leq 10$ ).

#### Формат выходных данных:

В выходной файл OUTPUT.TXT выведите ответ на задачу.

#### Примеры:

INPUT.TXT	OUTPUT.TXT
3 1	9
4 2	20

### Ход решения задачи:

Функция `main` – это точка входа. Считывает входные данные из документа с помощью метода `read_file`. Функция `count_mag` создает пустую доску размером  $N \times N$  и вызывает `place_mag` для подсчета количества способов размещения магарадж. Записывает результат в файл с помощью `save_file`. Измеряет и выводит время выполнения и использованную память.

Функция `place_mag` рекурсивно размещает магарадж на доске. Если количество размещенных магарадж достигает значения, функция возвращает 1 (успешное размещение). В противном случае, функция пробует разместить магараджу на каждой позиции, проверяя безопасность с помощью `is_safe`. Если размещение безопасно, функция рекурсивно вызывает себя для следующей магараджи.

## Код

```
5 usages
def count_mag(N: int, K: int) -> int:
    """Функция для подсчета количества способов разместить магарадж на доске."""
    board = [[0] * N for _ in range(N)]
    return place_mag(board, row: 0, col: 0, count: 0, K)

1 usage
def main() -> None:
    start_time = time.perf_counter()
    filename = "input.txt"
    N, K = read_file(filename)
    # print(N, K)

    result = count_mag(N, K)
    # print(result)
    save_file(result)

    stop_time = time.perf_counter()

    print(f"Время выполнения: {stop_time - start_time:0.5f} секунд.")

    final_memory = get_memory_usage()
    print(f"Использовано памяти: {final_memory:.2f} Mb.")

if __name__ == "__main__":
    main()
```

```
55 def place_mag(board: list, row: int, col: int, count: int, K: int) -> int:
56     """Рекурсивная функция для размещения магарадж на доске."""
57     if count == K:
58         return 1
59
60     total = 0
61     for i in range(row, len(board)):
62         for j in range(len(board)):
63             if is_safe(board, i, j):
64                 board[i][j] = 1
65                 total += place_mag(board, i, j, count + 1, K)
66                 board[i][j] = 0
67
68     return total
69
```

1 usage

```
def save_file(result_num: int) -> None:
    """Запись данных в файл."""
    with open("output.txt", "w") as file:
        result_value = str(result_num)
        file.write(result_value)
```

1 usage

```
def is_safe(board: list, row: int, col: int) -> bool:
    """Проверка, можно ли поставить магараджу на позицию."""
    for i in range(len(board)):
        if board[i][col] == 1 or board[row][i] == 1:
            return False

    for i in range(len(board)):
        for j in range(len(board)):
            if (i + j == row + col) or (i - j == row - col):
                if board[i][j] == 1:
                    return False

    knight_moves = [
        (2, 1), (1, 2), (-1, 2), (-2, 1),
        (-2, -1), (-1, -2), (1, -2), (2, -1)
    ]

    for move in knight_moves:
        new_row, new_col = row + move[0], col + move[1]
        if 0 <= new_row < len(board) and 0 <= new_col < len(board):
            if board[new_row][new_col] == 1:
                return False

    return True
```

```

1 usage
def get_memory_usage():
    """Получение используемой памяти в мегабайтах."""
    process = psutil.Process(os.getpid())
    mem_info = process.memory_info()
    return mem_info.rss / (1024 * 1024)

1 usage
def read_file(filename: str) -> tuple:
    """Чтение данных из файла."""
    with open(filename, "r") as file:
        line = file.readline().strip()
        a, b = map(int, line.split())

    return a, b

```

Тесты (Проходят успешно):

```

import unittest
from quest_8 import count_mag

class TestCountMag(unittest.TestCase):
    def test_count_mag(self):
        self.assertEqual(count_mag(N: 3, K: 1), second: 9)
        self.assertEqual(count_mag(N: 4, K: 2), second: 20)
        self.assertEqual(count_mag(N: 5, K: 3), second: 48)

```

Время выполнения: 0.00065 секунд.  
Использовано памяти: 15.34 Mb.

Process finished with exit code 0

Если видно плохо или хочется протестировать, то можете посмотреть мой код по заданию тут: [https://github.com/ytkinroman/lab\\_1\\_complexity/tree/main/quest\\_8](https://github.com/ytkinroman/lab_1_complexity/tree/main/quest_8)

## Задание 10. Перетягивание каната.

### Условие задачи:

#### **Задача 10. Перетягивание каната.**

Ограничение по времени: 3 с. Ограничение по памяти: 16 Мб.

Для участия в соревнованиях по перетягиванию каната зарегистрировалось  $N$  человек. Некоторые из участников могут быть знакомы друг с другом. Причем, если двое из них имеют общего знакомого, то это не означает, что они обязательно знакомы друг с другом.

Организаторы соревнований заинтересованы в их качественном проведении. Они хотят разделить всех участников на две команды так, чтобы в первой команде было  $K$  человек, а во второй –  $N-K$  человек. Из всех возможных вариантов формирования команд, организаторы хотят выбрать такой вариант, при котором сумма сплоченностей обеих команд максимальна. Сплоченностью команды называется количество пар участников этой команды, знакомых друг с другом. Ваша задача – помочь организаторам найти требуемое разделение участников на две команды.

#### Формат входных данных:

В первой строке входного файла INPUT.TXT задаются три числа  $N$ ,  $K$ ,  $M$ , разделенные одиночными пробелами, где  $N$  – общее число зарегистрированных

участников,  $K$  – требуемое количество человек в первой команде,  $M$  – количество пар участников, знакомых друг с другом.

Каждая из следующих  $M$  строк содержит два различных числа, разделенные пробелом – номера двух участников, знакомых друг с другом. Все участники нумеруются от 1 до  $N$ .

Ограничения: все числа целые,  $0 < K < N < 25$ ,  $0 \leq M \leq N(N-1)/2$

#### Формат выходных данных:

Выходной файл OUTPUT.TXT должен содержать одну строку, состоящую из  $K$  чисел, каждое из которых задает номер участника, попавшего в первую команду. Числа должны быть разделены пробелами. Если существует несколько решений данной задачи, то выведите любое из них.

#### Примеры:

INPUT.TXT	OUTPUT.TXT
5 3 3 1 3 2 5 5 4	5 2 4

### Ход решения задачи:

Функция `main` – это точка входа. Считывает входные данные из документа с помощью метода `read_file`.

Функция `get_matrix` создает матрицу смежности. Матрица смежности используется для представления графа и последующего вычисления связности команд.

Функция `find_best_combination` инициализирует процесс поиска лучшей комбинации участников и возврата результата. Вызывается функция `generate_combinations` для генерации всех возможных комбинаций и поиска наиболее связанной комбинации.

Функция `generate_combinations` рекурсивно генерирует все возможные комбинации участников и находит наиболее связанную комбинацию. Если текущая комбинация достигла нужного размера `k`, вычисляется её связность с помощью функции `calculate_cohesion`. Если связность текущей комбинации больше, чем у лучшей найденной комбинации, обновляется лучшая комбинация. Рекурсивно генерируются все возможные комбинации, добавляя очередного участника из списка оставшихся участников.

### Код

```
4 usages
▼ def find_best_combination(N: int, K: int, matrix: list) -> str:
    """Находит лучшую комбинацию участников."""
    participants = list(range(1, N + 1))
    best_combination = [[]]

    generate_combinations(current_combination: [], participants, K, matrix, best_combination, best_cohesion: [0])

    return get_result_string(best_combination[0])

1 usage
▼ def get_result_string(result_list: list) -> str:
    """Преобразует список целых чисел в строку."""
    return " ".join(map(str, result_list))

1 usage
▼ def main() -> None:
    start_time = time.perf_counter()
    filename = "input.txt"
    N, K, M, edges = read_file(filename)

    matrix = get_matrix(N, edges)

    best_combination = find_best_combination(N, K, matrix)

    with open("output.txt", "w") as file:
        file.write(best_combination)

> if __name__ == "__main__":
    main()
```

1 usage

```
def get_matrix(party: int, edges: list) -> list:
```

```
    """Создание матрицы смежности."""
```

```
    matrix = []
```

```
    for i in range(party):
```

```
        row = [0] * party
```

```
        matrix.append(row)
```

```
    for edge in edges:
```

```
        u = edge[0] - 1
```

```
        v = edge[1] - 1
```

```
        matrix[u][v] = 1
```

```
        matrix[v][u] = 1
```

```
    return matrix
```

1 usage

```
def calculate_cohesion(team_list: list, matrix: list) -> int:
```

```
    """Вычисляет связность команды путем подсчета количества пар участников в команде."""
```

```
    team_length = len(team_list)
```

```
    ties = 0
```

```
    for i in range(team_length):
```

```
        for j in range(i + 1, team_length):
```

```
            if matrix[team_list[i] - 1][team_list[j] - 1] == 1:
```

```
                ties += 1
```

```
    return ties
```

2 usages

```
def generate_combinations(current_combination: list, remaining_participants: list, k: int, matrix: list, best_combination: list, best_cohesion: list) -> None:
```

```
    """Генерирует комбинации участников и находит лучшую комбинацию на основе связности."""
```

```
    if len(current_combination) == k:
```

```
        cohesion = calculate_cohesion(current_combination, matrix)
```

```
        if cohesion > best_cohesion[0]:
```

```
            best_cohesion[0] = cohesion
```

```
            best_combination[0] = current_combination
```

```
        return
```

```
    for i in range(len(remaining_participants)):
```

```
        generate_combinations(current_combination + [remaining_participants[i]], remaining_participants[i+1:], k, matrix, best_combination, best_cohesion)
```

```

def get_memory_usage():
    """Получение используемой памяти в мегабайтах."""
    process = psutil.Process(os.getpid())
    mem_info = process.memory_info()
    return mem_info.rss / (1024 * 1024)

1 usage
def read_file(filename: str) -> tuple:
    """Чтение данных из файла."""
    with open(filename, "r") as file:
        first_line = file.readline().strip()
        parts = first_line.split()

        N = int(parts[0]) # Общее число зарегистрированных участников.
        K = int(parts[1]) # Требуемое количество человек в первой команде.

        M = int(parts[2]) # Количество пар участников, знакомых друг с другом.

        edges = []
        for i in range(M):
            line = file.readline().strip()
            parts = line.split()

            a = int(parts[0])
            b = int(parts[1])
            edge = (a, b)

            edges.append(edge)

    return N, K, M, edges

```

Тесты (Проходят успешно):

```

class TestFunctions(unittest.TestCase):
    def test_get_matrix_1(self):
        N = 5
        edges = [(1, 3), (2, 5), (5, 4)]
        matrix = get_matrix(N, edges)
        self.assertEqual(matrix, (second) [[0, 0, 1, 0, 0], [0, 0, 0, 0, 1], [1, 0, 0, 0, 0], [0, 0, 0, 0, 1], [0, 1, 0, 1, 0]])

    def test_get_matrix_2(self):
        N = 7
        edges = [(1, 3), (2, 3), (5, 4)]
        matrix = get_matrix(N, edges)
        self.assertEqual(matrix, (second) [[0, 0, 1, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0], [1, 1, 0, 0, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0]])

    def test_find_best_combination_1(self):
        N = 5
        K = 3
        matrix = [[0, 0, 1, 0, 0], [0, 0, 0, 0, 1], [1, 0, 0, 0, 0], [0, 0, 0, 0, 1], [0, 1, 0, 1, 0]]
        best_combination = find_best_combination(N, K, matrix)
        self.assertEqual(best_combination, (second) "2 4 5")

    def test_find_best_combination_2(self):
        N = 7
        K = 4
        matrix = [[0, 0, 1, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0], [1, 1, 0, 0, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0]]
        best_combination = find_best_combination(N, K, matrix)
        self.assertEqual(best_combination, (second) "1 2 3 4")

```



```
D:\PycharmProjects\algorithms_and_complexity_analysis\.venv\Scripts\python  
Время выполнения: 0.00062 секунд.  
Использовано памяти: 15.33 Мб.  
  
Process finished with exit code 0
```

Если видно плохо или хочется протестировать, то можете посмотреть мой код по заданию тут:

[https://github.com/ytkinroman/lab\\_1\\_complexity/tree/main/quest\\_10](https://github.com/ytkinroman/lab_1_complexity/tree/main/quest_10)