МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное образовательное учреждение высшего образования «Санкт-петербургский государственный морской технический университет»

ФАКУЛЬТЕТ ЦИФРОВЫХ ПРОМЫШЛЕННЫХ ТЕХНОЛОГИЙ

Кафедра Киберфизических систем

ЛАБОРАТОРНАЯ РАБОТА

По дисциплине «Программирование»

Выполнил: Гузненков Владислав Сергеевич группа 20121

Проверил: Поделенюк Павел Петрович

Санкт-Петербург

2024

Оглавление

[1. Цели и формулировка задачи 3](#_Toc187676633)

[2. Результаты работы 4](#_Toc187676634)

[2.1. Реализация программы с использованием функционального программирования языка Python 4](#_Toc187676635)

[2.1.1. Ход работы 4](#_Toc187676636)

[2.1.2. Демонстрация работы программы 7](#_Toc187676637)

[2.1.3. Листинг кода 8](#_Toc187676638)

[ЗАКЛЮЧЕНИЕ 10](#_Toc187676639)

[Список использованных источников 11](#_Toc187676640)

# Цели и формулировка задачи

Цель: При работе над данной лабораторной работой будут затронуты следующие вопросы:

1. Работа с текстовыми файлами,
2. Работа с локальными и глобальными переменными,
3. Работа с несколькими функциями, а не только с main,
4. Нисходящий метод разработки программ,
5. Анализ ходов.

Формулировка задачи:

Дана квадратная шахматная доска размером N x N. На доске уже размещено K фигур. Фигуры размещены так, что находятся не под боем друг друга.

Необходимо расставить на доске еще L фигур так, чтобы никакая из фигур на доске не находилась под боем любой другой фигуры. Необходимо найти все возможные решения.

Фигура: Смесь ферзя и коня

Входные данные в файле input.txt. На первой строке файла записаны три числа: N L K (через пробел). Далее следует K строк, содержащих числа x и y (через пробел) - координаты уже стоящей на доске фигуры (фигуры стоят правильно). Координаты отсчитываются от 0 до N-1. 1 <= N <= 20.

Выходные данные в файл output.txt. На каждое найденное решение необходимо записать в файл одну строку. Строка состоит из пар (x,y) - координаты фигур на доске. В решение следует вывести координаты всех фигур, находящихся на доске. Каждую фигуру необходимо записать в виде пары координат, разделенных запятой и обрамленных скобками. Координаты отсчитываются от 0 до N-1. Порядок, в котором фигуры перечислены в решении, не имеет значения. Если не было найдено ни одного решения, в файл необходимо записать no solutions.

Выходные данные на консоль — это доска N\*N, где фигура обозначается #, ее ходы обозначаются \*, а пустые клетки обозначаются 0.

# Результаты работы

## **Реализация программы с использованием функционального программирования языка Python**

Основная идея алгоритма заключалась в том, чтобы программа отбирала подходящие по условию координаты фигуры с помощью рекурсивной функции, которая бы сначала перебирала координаты циклом, потом проверяла, подходят ли эти координаты по условию, а затем начиналась бы рекурсия с поставленной фигурой

## Ход работы

Функция mark\_moves: функция принимает на вход:

* Доску в виде двумерного массива с помеченными фигурами и их ходами,
* Позицию фигуры, которую необходимо поставить на доску,
* Размер доски.

Эта функция помечает ходы фигуры символом «\*» на нашей доске (в нашем случае ходы конем и ферзем). Она помечает их в 2 этапа: сначала помечаются ходы ферзем, то есть проверяются координаты на валидность (не выходят ли за доску), а затем помечается ход, потом помечаются ходы конем следующим образом: возможные ходы записаны в переменной horse\_moves в виде координат, которые необходимо прибавить к координате фигуры, и если итоговая координата не выходит за рамки доски, то эта координата помечается как ход.

Функция unmark\_moves: функция принимает на вход:

* Доску в виде двумерного массива,
* Позицию фигуры, которую нужно убрать и её ходы соответственно,
* Размер доски.

Эта функция убирает выбранную нами фигуру и её ходы на доске, делается это следующим образом: сначала сама фигура заменяется на «0», а позже и все ходы остальных фигур на тот же символ. Далее, для каждой оставшейся фигуры работает функция mark\_moves и помечает ходы для каждой фигуры. Таким образом, у нас выходит доска без выбранной нами фигуры и её ходов.

Функция place\_figure: эта рекурсивная функция получает на вход:

Доску в виде двумерного массива,

Количество фигур, которые осталось расставить на доску,

Размер доски,

Список с координатами фигур, расставленных в последствии выполнения функции,

Переменную, которая равна x в части (x, y) координаты фигуры, поставленной в последствии выполнения предыдущего шага рекурсии,

Переменную, которая равна y в части (x, y) координаты фигуры, поставленной в последствии выполнения предыдущего шага рекурсии (в цикле она позже меняется на 0),

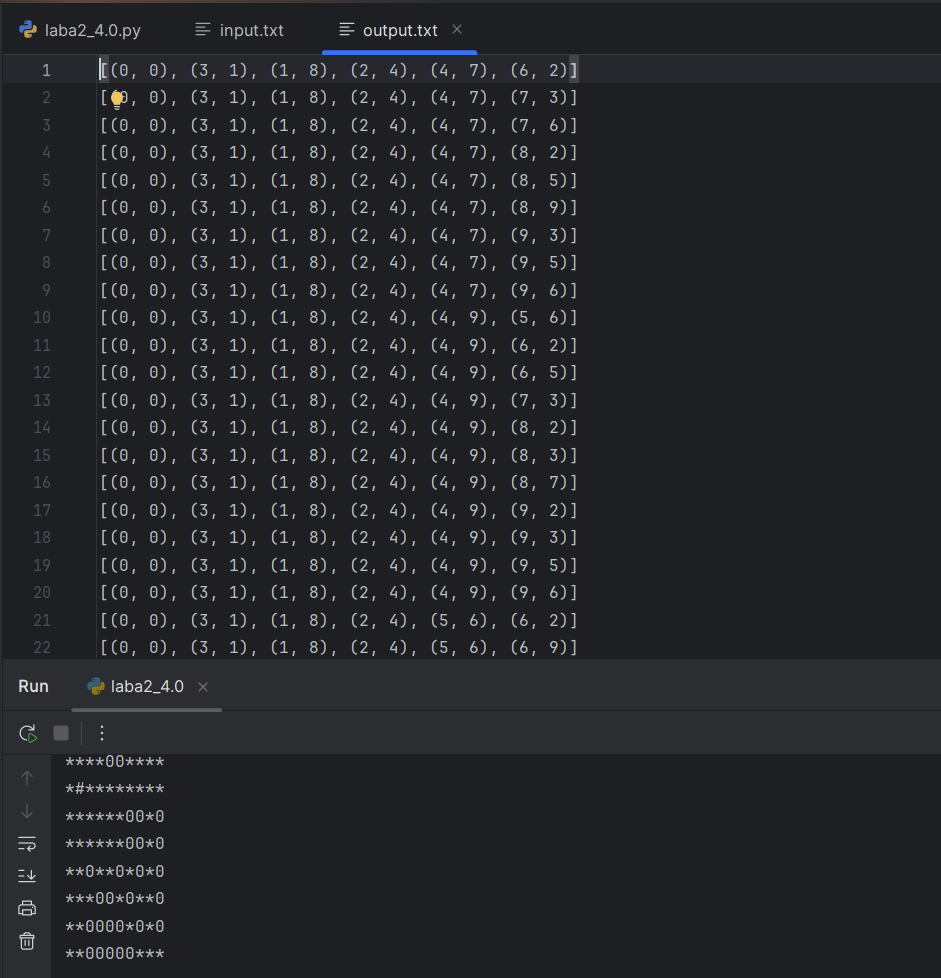
Массив, состоящий из предварительно расставленных на доску фигур до входа в функцию.

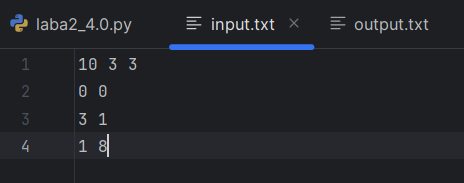
В этой функции выход из рекурсии реализован в виде условия: если число фигур, которые необходимо расставить равно 0, то в файл output.txt записываются координаты расставленных фигур условием и координаты фигур, расставленных в последствии работы функции, а также переменная a становится равной True, что значит, что было найдено хотя бы одно решение.

Сам алгоритм работает следующим образом: реализован двухэтапный цикл, перебирающий все возможные x и y на доске , который работает от переменной start\_x до конца доски и от start\_y, который равен 0, если первый x поменялся (сделано это для того, чтобы цикл не начинался сначала при следующем шаге рекурсии, иначе будут выведены повторы координат в файл). Сначала проверяется, равен ли символ на этой координате «0», если равен, то ставится на доску фигура с помощью функции mark\_moves и добавляется координата в positions, потом начинается новый шаг рекурсии с новыми доской и массивом positions. Далее, чтобы цикл не работал с этими же доской и массивом дальше, мы убираем координату из positions и фигуру и её ходы с помощью функции unmark\_moves.

Дальше код состоит из считывания с файла input.txt размера доски n, числа фигур, которые надо расставить l, числа фигур, уже расставленных на доску k и координат этих фигур. Далее, создается доска с расставленными фигурами с помощью цикла и функции mark\_moves. Потом первый вход в функцию place\_figure с исходными данными, запись в файл output.txt «no solutions» если не было найдено ни одного решения и закрытия обоих файлов.

## Демонстрация работы программы



Примерно так выглядит файл output.txt (337 строк) после работы программы со следующими данными: 

## Листинг кода

1. def mark\_moves(board: list, pos: tuple, n: int):  
    *"""  
    Функция, помечающая все возможные ходы фигуры на доске  
    :param board: доска  
    :param pos: позиция фигуры  
    :param n: размер доски (n\*\*2)  
    :return: не нужен тут  
    """* horse\_moves = [(-2, -1), (-2, 1), (-1, -2), (-1, 2), (1, -2), (1, 2), (2, -1), (2, 1)] # ходы конем  
    for fx in range(n): # помечаем ходы ферзем  
    board[fx][pos[1]] = '\*'  
    board[pos[0]][fx] = '\*'  
    if pos[0] - fx >= 0 and pos[1] - fx >= 0:  
    board[pos[0] - fx][pos[1] - fx] = '\*'  
    if pos[0] + fx < n and pos[1] + fx < n:  
    board[pos[0] + fx][pos[1] + fx] = '\*'  
    if pos[0] - fx >= 0 and pos[1] + fx < n:  
    board[pos[0] - fx][pos[1] + fx] = '\*'  
    if pos[0] + fx < n and pos[1] - fx >= 0:  
    board[pos[0] + fx][pos[1] - fx] = '\*'  
    for dx, dy in horse\_moves: # помечаем ходы конем  
    if 0 <= pos[0] + dx < n and 0 <= pos[1] + dy < n:  
    board[pos[0] + dx][pos[1] + dy] = '\*'  
    board[pos[0]][pos[1]] = '#' # ставим саму фигуру  
     
     
     
   def unmark\_moves(board: list, pos: tuple, n: int):  
    *"""  
    Функция, возвращающая доску в состояние до размещения новой фигуры  
    :param board: доска  
    :param pos: позиция фигуры, ходы которой нужно убрать (и её саму)  
    :param n: размер доски (n\*\*2)  
    :return: не нужен тут  
    """* board[pos[0]][pos[1]] = '0' # убираем фигуру с доски  
    for x in range(n): # убираем все возможные ходы фигур  
    for y in range(n):  
    if board[x][y] == '\*':  
    board[x][y] = '0'  
    for x in range(n): # помечаем все возможные ходы расставленных фигур на доске заново, но уже без убранной фигуры  
    for y in range(n):  
    if board[x][y] == '#':  
    mark\_moves(board, (x, y), n)  
     
     
   def place\_figure(board: list, l: int, n: int, positions: list, start\_x: int, start\_y: int, placed\_figures: list) -> None:  
    *"""  
    Рекурсивная функция, расстанавливающая фигуры на доске  
    :param board: доска  
    :param l: количество оставшихся фигур для расстановки  
    :param n: размер доски (n\*\*2)  
    :param positions: список с координатами расставленных фигур (в последствии выполнения функции)  
    :param start\_x: переменная, используемая в цикле перебора координат, чтобы при каждом новом входе в рекурсию отсчет  
    координат не начинался с нуля  
    :param start\_y: тоже самое, что и start\_x, но при втором и последующих циклах row изменяется на 0 (чтобы было как  
    (8,9), (9, 0), (9,1)..., а не (8,9), (9,9))  
    :param placed\_figures: уже поставленные на доску из условия фигуры  
    :return: выход из рекурсии чтобы комп не лопнул  
    """* global a  
    if l == 0: # все фигуры расставлены, выход из рекурсии  
    a = True  
    out.write(f'{placed\_figures + positions}\n')  
    return  
     
    for row in range(start\_x, n): # цикл, перебирающий все возможные координаты на доске  
    for col in range(start\_y if row == start\_x else 0, n):  
    if board[row][col] == '0': # проверка не под боем ли клетка  
    positions.append((row, col)) # добавляем координаты новой фигуры  
    mark\_moves(board, (row, col), n) # помещаем фигуру на доску  
     
    place\_figure(board, l - 1, n, positions, row, col, placed\_figures) # рекурсия  
     
    unmark\_moves(board, (row, col), n) # откатываем изменения  
    positions.pop()  
    return  
     
     
   f = open('input.txt', 'r') # открываем файл с входными данными и считываем с него информацию  
   nlk = list(map(int, f.readline().split()))  
   n, l, k = nlk[:3]  
   init\_pos = []  
   for \_ in range(k):  
    x, y = map(int, f.readline().split())  
    init\_pos.append((x, y))  
   a = False # переменная, показывающая, что функция нашла хотя бы 1 решение  
     
     
   board = [['0' for \_ in range(n)] for \_ in range(n)] # инициализация доски  
   for pos in init\_pos:  
    mark\_moves(board, pos, n)  
   print(''.join([''.join(x) + '\n' for x in board])) # вывод исходной доски (по приколу)  
     
   out = open('output.txt', 'w') # открываем файл вывода  
   place\_figure(board, l, n, [], 0, 0, init\_pos) # входим в функцию, расставляющую фигуры  
   if not a:  
    out.write('no solutions')  
   f.close() # закрываем файлы  
   out.close()

# ЗАКЛЮЧЕНИЕ

Были получены первичные знания о работе с файлом, с прочтением информации с него и записью в другой файл, о работе с несколькими функциями сразу и с рекурсивной в том числе, о разбиении одной крупной задачи на несколько подзадач.

# Список использованных источников

<https://pythonru.com/osnovy/rekursiya-python> - **Рекурсивная функция в python**

<https://pythonworld.ru/tipy-dannyx-v-python/fajly-rabota-s-fajlami.html> - **Файлы. Работа с файлами.**

<https://www.youtube.com/watch?v=jvFULnNpNLg> - **41 Рекурсия в Python. Рекурсивная функция Часть 1**

<https://www.youtube.com/watch?v=rzGCxtZdMuM> - **42 Рекурсия в Python. Рекурсивная функция Часть 2**

<https://pythonru.com/osnovy/globalnye-peremennye-python> **- Руководство по глобальным переменным**