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: Башкатов Артемий Валерьевич 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: Заключается в изучении основ функционального программирования на языке Python и реализации алгоритма для размещения фигур на шахматной доске с учетом правил атаки. Задача состоит в том, чтобы разработать программу, которая принимает на вход размеры доски и позиции заранее заданных фигур, а затем находит все возможные способы размещения дополнительных фигур, не подвергая их атаке.

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: Биг Ромб

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**

## Ход работы

Функция read\_input: функция для парсинга входных данных из файла.

Функция mark\_tiles: функция для отметки ячеек под ударом, также отмечает саму фигуру.

* x, y — координаты фигуры
* board — шахматная доска

Функция clear\_tiles: функция обратная mark\_tiles, убирает ячейки под боем.

* x, y — координаты фигуры
* board — шахматная доска

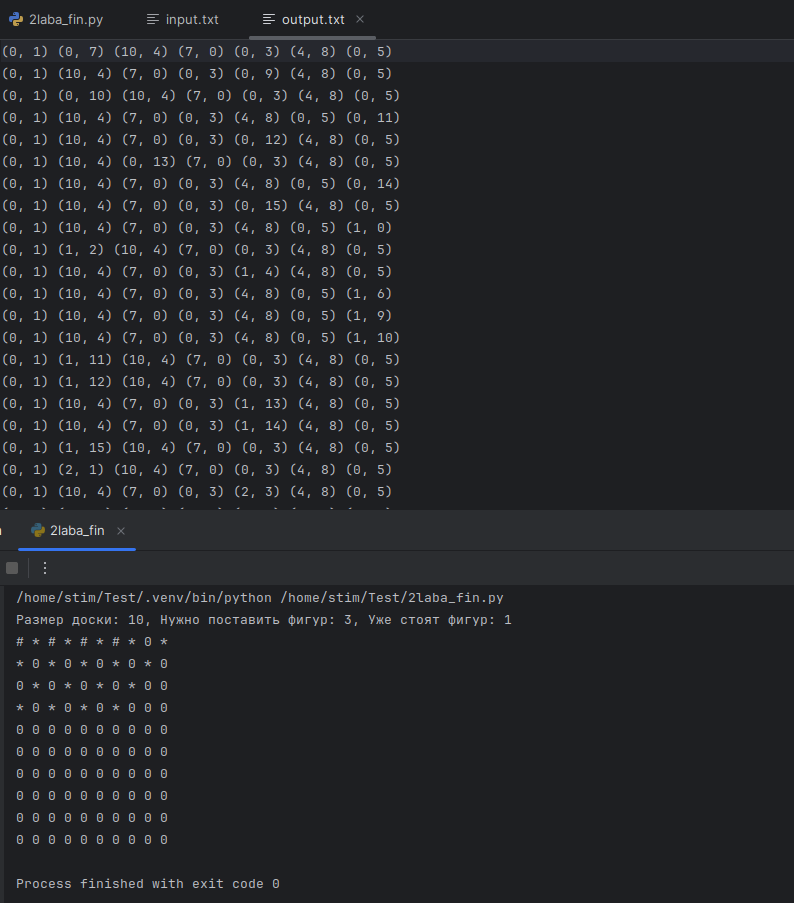
Функция can\_be\_placed: функция возвращающее булевое значение True, если фигуру можно разместить на эту ячейку и она не под ударом, иначе False

* x, y — координаты фигуры
* board — шахматная доска

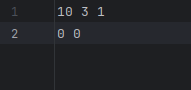
Функция place\_figure: Основная рекурсивная функция программы. Алгоритм сначала проверяет были ли поставленны все фигуры, если да и решение уникальное, то записывает значение в файл и в кэш, если решение уже было в кэше то ничего не записывается.

* x, y — координаты с которых алгоритм начинает работу
* l — количество фигур которых надо расставить
* cur\_solution — список состоящий из кортежей в которых сохранены координаты фигур
* placed\_figures — список кортежей с координатами уже поставленных фигур на доске
* board — щахматная доска
* cache — множество состоящее из кортежей с кортежами, в которых сохранены координаты правильных решений

## Демонстрация работы программы



Входные данные:



## Листинг кода

def read\_input(filename: str) -> tuple:  
 with open(filename, 'r') as f:  
 N, L, K = map(int, f.readline().strip().split())  
 pieces = [tuple(map(int, f.readline().strip().split())) for \_ in range(K)]  
 return N, L, K, pieces  
  
def mark\_tiles(x: int, y: int, board: list):  
 moves = ((0, -1), (0, 1), (3, 0), (-3, 0), (0, -3), (0, 3), (-1, -2), (-1, 2), (-1, 0), (1, -2), (1, 2), (1, 0), (-2, -1), (-2, 1), (2, -1), (2, 1))  
 for i in moves:  
 if x + i[0] < 0 or y + i[1] < 0 or x + i[0] > n-1 or y + i[1] > n-1:  
 continue  
 else:  
 board[x+i[0]][y+i[1]] = '\*'  
 board[x][y] = '#'  
  
def clear\_tiles(x: int, y: int, board: list):  
 board[x][y] = '0'  
 for x in range(n):  
 for y in range(n):  
 if board[x][y] == '\*':  
 board[x][y] = '0'  
 for x in range(n):  
 for y in range(n):  
 if board[x][y] == '#':  
 mark\_tiles(x, y, board)  
  
def can\_be\_placed(x: int, y: int, board) -> bool:  
 moves = ((0, -1), (0, 1), (3, 0), (-3, 0), (0, -3), (0, 3), (-1, -2), (-1, 2), (-1, 0), (1, -2), (1, 2), (1, 0), (-2, -1), (-2, 1), (2, -1), (2, 1))  
 for i in moves:  
 if x + i[0] < 0 or y + i[1] < 0 or x + i[0] > n - 1 or y + i[1] > n - 1:  
 continue  
 else:  
 if board[x + i[0]][y+i[1]] == '#' or board[x][y] == '#':  
 return False  
 else:  
 continue  
 return True  
  
def place\_figure(x: int, y: int, l: int, cur\_solution: list, placed\_figures: list, board: list, cache: set):  
 if l == 0:  
 cur\_solution.sort()  
 cur\_solution\_t = tuple(cur\_solution)  
 if cur\_solution\_t in cache:  
 return None  
 cache.update(cur\_solution\_t)  
 answ = cur\_solution + placed\_figures  
 #print(\*answ)  
 for el in answ:  
 f.write(str(el) + " ")  
 #f.write(f"{cur\_solution + placed\_figures}"[1:-1] + '\n')  
 f.write('\n')  
 return None  
 for i in range(x, n):  
 for j in range(y if i == x else 0, n):  
 if can\_be\_placed(i, j, board):  
 cur\_solution.append((i, j))  
 mark\_tiles(i, j, board)  
 place\_figure(i, j, l - 1, cur\_solution, placed\_figures, board, cache)  
 clear\_tiles(i, j, board)  
 cur\_solution.pop()  
  
def main():  
 global n, l, f, const\_pieces  
 n, l, k, const\_pieces = read\_input('input.txt')  
 board = [['0' for i in range(n)] for j in range(n)]  
 for i in const\_pieces:  
 mark\_tiles(i[0], i[1], board)  
 f = open('output2.txt', 'w')  
 cache = set()  
 print(f'Размер доски: {n}, Нужно поставить фигур: {l}, Уже стоят фигур: {k}')  
 place\_figure(0, 0, l, [], const\_pieces, board, cache)  
 f.close()  
 f = open('output2.txt', 'r+')  
 a = f.readline()  
 if a == '':  
 f.write('no solution')  
 else:  
 a = str(a).strip().split(sep=' ')  
 c = [ele.replace(',', '') for ele in a]  
 b = [ele.replace('(', '') for ele in c]  
 d = [ele.replace(')', '') for ele in b]  
 p = []  
 for i in range(1, len(d), 2):  
 p.append((d[i-1], d[i]))  
 d = []  
 for i in p:  
 d.append((int(i[0]), int(i[1])))  
 for i in d:  
 mark\_tiles(i[0], i[1], board)  
 for i in board:  
 print(\*i)  
 f.close()  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

# 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