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:

Халилев Владислав Виленович

Проверил:

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 4](#_Toc187676637)

[2.1.3. Листинг кода 4](#_Toc187676638)

[ЗАКЛЮЧЕНИЕ 5](#_Toc187676639)

[Список использованных источников 6](#_Toc187676640)

# Цели и формулировка задачи

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.

Работа совершалась только с использованием шахматной фигуры ”Алфин + Рыбка” (ходы данной фигуры через две клетки по диагонали и две по вертикали)

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 8, ее ходы обозначаются 1, а пустые клетки обозначаются 0.

# Результаты работы

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

## Ход работы

1. Считывем входные данные.
2. Создаем матрицу N\*N, заполненную нулями.
3. Позиционируем имеющиеся фигуры на доске, с указанием их возможных ходов.
4. Создание функции для проверки возможности постановки фигуры на x, y координаты. Создание функции для расстановки полей под боем от поставленной фигуры.
5. Создание и запуск рекурсивной функции для заполнения клеток поля фигурами, координаты которых представлены в входном файле. После проверки на количество расставленных фигур записывает их координаты в файл.

## Демонстрация работы программы

## 

<https://github.com/Cvif223/20121_Vlad_Khalilev.git>

## Листинг кода

import numpy as np  
  
def read\_input(file\_path):  
 with open(file\_path, 'r') as file:  
 N, L, K = map(int, file.readline().strip().split()) # где L - количество фигур, которые ещё предстоит разместить  
 #print(file.readline().strip().split())  
 existing\_positions = [tuple(map(int, file.readline().strip().split())) for \_ in range(K)]  
 print(existing\_positions)  
 return N, L, existing\_positions  
N, L, e = read\_input("input.txt")  
  
  
def create\_brd(n):  
 # Создаем шахматное поле  
 brd = np.zeros((n, n), dtype=int)  
 return brd  
chessboard = create\_brd(N)  
  
  
def position\_fgrs(field, coordinates):  
 #Позиционирует фигуру(ы) на шахматной доске  
 for x, y in coordinates:  
 if 0 <= x < field.shape[0] and 0 <= y < field.shape[1]:  
 field[x, y] = 8 # Устанавливаем фигуру  
 else:  
 print(f"Координаты ({x}, {y}) выходят за пределы доски.")  
 return chessboard  
placed\_figures = position\_fgrs(chessboard, e)  
  
def position\_trgts(field, coordinates, need\_moves=False):  
 #Позиционирует ходы на шахматной доске  
 for x, y in coordinates:  
 if 0 <= x < field.shape[0] and 0 <= y < field.shape[1]:  
 field[x, y] = 1 # Устанавливаем фигуру  
 else:  
 print(f"Координаты ({x}, {y}) выходят за пределы доски.")  
 if need\_moves:  
 return chessboard, coordinates  
 return chessboard  
  
def move\_piece(chessboard, placed\_fgr, need\_moves=False):  
 #Выполняет ходы для фигур на шахматной доске  
 moves = []  
  
 for x, y in placed\_fgr:  
 # Диагональные ходы  
 diagonal\_moves = [(x + 1, y + 1), (x + 2, y + 2), (x + 1, y - 1), (x + 2, y - 2), (x - 1, y + 1), (x - 2, y + 2), (x - 1, y - 1), (x - 2, y - 2)]  
 # Вертикальные ходы  
 vertical\_moves = [(x + 3, y), (x - 3, y)]  
 # Горизонтальные ходы  
 horizontal\_moves = [(x, y + 3), (x, y - 3)]  
  
 all\_moves = diagonal\_moves + vertical\_moves + horizontal\_moves  
  
 # Проверка допустимости ходов  
 for new\_x, new\_y in all\_moves:  
 if 0 <= new\_x < chessboard.shape[0] and 0 <= new\_y < chessboard.shape[1] and chessboard[new\_x, new\_y] == 0:  
 moves.append((new\_x, new\_y))  
 if need\_moves:  
 return position\_trgts(placed\_figures, moves, need\_moves=True)  
 return position\_trgts(placed\_figures, moves)  
new\_board = move\_piece(chessboard, e)  
"""  
На данном этапе имеем доску с единицами там куда уже точно не сможем поставить ни одну фигуру,  
ведь она будет находиться либо под боем имеющейся, либо на размещённой фигуре  
"""  
print(new\_board)  
  
  
def is\_safe(chessboard, x, y):  
 # Проверка на занятость клетки  
 if chessboard[x, y] in [1, 8]:  
 return False # Клетка занята  
  
 # Проверка на атаку других фигур (8)  
 directions = [  
 (1, 1), (2, 2), (1, -1), (2, -2), (-1, 1), (-2, 2), (-1, -1), (-2, -2),  
 (3, 0), (-3, 0),  
 (0, 3), (0, -3)  
 ]  
  
 for dx, dy in directions:  
 nx, ny = x, y  
 while 0 <= nx < chessboard.shape[0] and 0 <= ny < chessboard.shape[1]:  
 nx += dx  
 ny += dy  
 if 0 <= nx < chessboard.shape[0] and 0 <= ny < chessboard.shape[1]:  
 if chessboard[nx, ny] == 8: # Если находим фигуру 8, то клетка под атакой  
 return False  
  
 return True # Клетка безопасна для размещения новой фигуры  
  
file = open("output.txt", mode="w")  
def place\_figures(chessboard, L, current\_solution=[]):  
 if L == 0:  
 # Если все фигуры размещены, сохраняем решение  
 file.write(f"{current\_solution}\n")  
 return  
  
 # Перебираем все возможные позиции на доске  
 for x in range(chessboard.shape[0]):  
 for y in range(chessboard.shape[1]):  
 if is\_safe(chessboard, x, y):  
 # Если позиция безопасна, размещаем фигуру  
 chessboard[x, y] = 8 # Устанавливаем фигуру  
 current\_solution.append((x, y)) # Запоминаем позицию  
  
 # Получаем возможные ходы для размещенной фигуры  
 possible\_moves, save\_to\_del = move\_piece(chessboard, current\_solution, need\_moves=True)  
 # Рекурсивно пытаемся разместить оставшиеся фигуры  
 place\_figures(chessboard, L - 1, current\_solution)  
  
 # Убираем фигуру и откатываемся  
 chessboard[x, y] = 0  
 current\_solution.pop()  
 for x1, y1 in save\_to\_del:  
 chessboard[x1, y1] = 0  
  
# Запуск размещения фигур  
place\_figures(chessboard, L)

# ЗАКЛЮЧЕНИЕ

По результатам тестирования программы, поставленные задачи выполнены, все цели достигнуты.

# Список использованных источников

https://skillbox.ru/media/code/biblioteka-numpy-vsye-chto-nuzhno-znat-novichku/

Конспекты лекций с занятий