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»

Выполнила: Подпорина Мария Тимофеевна группа 20121

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 5](#_Toc187676637)

[2.1.3. Листинг кода 6](#_Toc187676638)

[ЗАКЛЮЧЕНИЕ 8](#_Toc187676639)

[Список использованных источников 9](#_Toc187676640)

# Цели и формулировка задачи

Цели:

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 с фигурой: Алфил.

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. 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.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

1. Функция: read\_input(filename: str)

Открывает файл с помощью контекстного менеджера with. Считывает первую строку файла, которая содержит три числа: размер доски (N), количество фигур для размещения (L) и количество уже расставленных фигур (K). Далее считывает K строк, каждая из которых содержит координаты (x, y) для начально расставленных фигур. Преобразует эти данные в соответствующие типы и возвращает их в виде кортежа.

2. Функция: valid\_position(board: list[list[int]], x: int, y: int, N: int)

Определяет все возможные ходы для фигуры, которые могут угрожать текущей позиции. В данном случае используются смещения для проверки угрозы от других фигур. Для каждого возможного хода проверяет, находится ли угроза в пределах доски и занята ли эта клетка другой фигурой. Если хотя бы одна угроза обнаружена, позиция недопустима, возвращается False.

3. Функция check\_starting\_position(pieces: list[tuple[int, ...]], N: int)

Создает пустую доску размером N x N и заполняет ее нулями. Для каждой начальной позиции (x, y) устанавливает значение «1» на доске, обозначая размещение фигуры. Для каждой фигуры проверяет, допустима ли ее позиция с помощью функции valid\_position. Если хотя бы одна фигура находится в недопустимой позиции, функция возвращает False.

4. Функция: place\_pieces(board: list[list[int]], pieces: list[tuple[int, ...]], L: int, N: int, solutions: list, line: int, col: int)

Если количество фигур для размещения (L) равно нулю, добавляет текущее расположение фигур в список решений. Проходит по всем клеткам доски начиная с заданной строки и столбца. Для каждой клетки проверяет, свободна ли она и допустима ли для размещения новой фигуры. Если клетка допустима, помечает ее как занятую, устанавливает значение «1», добавляет координаты в список pieces и рекурсивно вызывает функцию для размещения оставшихся фигур. После возврата из рекурсии убираем фигуру с доски и из списка.

5. Функция: print\_board(board: list[list[int]], N: int)

Проходит по всем клеткам доски. Если клетка занята фигурой (значение 1), выводит символ "#". Если клетка свободна и допустима для размещения новой фигуры, выводит "0". Если клетка недопустима для размещения, выводит "\*".

6. Функция: write\_output(file: str, solutions: list)

Открывает файл для записи. Если список решений пустой, записывает "no solutions". В ином случае проходим по всем найденным решениям и записываем их в файл в формате "(x,y)".

7. Функция: main()

Объединяет и запускает все функцию. Считывает входные данные с помощью read\_input. Создает доску и проверяет начальное расположение фигур. Если начальные позиции корректны, запускает поиск возможных решений с помощью place\_pieces. Записывает результаты в выходной файл с помощью write\_output. Для демонстрации выводит одно из найденных решений на экран с использованием print\_board.

Программа запускается при выполнении скрипта через проверку

if \_\_name\_\_ == "\_\_main\_\_":

## Демонстрация работы программы

1. Создайте текстовый файл input.txt с содержимым:

На первой строке файла записаны три числа: N L K (через пробел). Далее следует K строк, содержащих числа x и y (через пробел)

1. При запуске программы выполняется функция main(), которая последовательно вызывает другие функции.

Функция read\_input("input.txt") считывает данные из файла. Функция check\_starting\_position() проверяет, корректно ли расставлены начальные позиции фигур. Функция place\_pieces() начинает рекурсивный поиск возможных позиций для размещения оставшихся фигур. Затем программа откатывает изменения и продолжает проверять другие клетки. Продолжаем этот процесс до тех пор, пока не будут проверены все возможные комбинации. После завершения поиска все найденные решения сохраняются в выходной файл output.txt. Если решений нет, записывается "no solutions". Функция print\_board() вызывается для отображения одного из найденных решений.

1. Программа завершает свою работу после записи результата в файл output.txt.

## Листинг кода

def read\_input(filename: any):  
 *''' Функция считывания данных из файла "input.txt"'''* 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 valid\_position(board: list[list[int]], x:int, y:int, N:int):  
 *''' Функция проверяющая допустима ли позиция для хода '''* moves = [(-2, -2), (-2, 2), (2, -2), (2, 2), (-1, -1), (-1, 1), (1, -1), (1, 1)]  
 for dx, dy in moves:  
 if 0 <= x + dx < N and 0 <= y + dy < N and board[x + dx][y + dy] == 1:  
 return False  
 return True  
  
def check\_starting\_position(pieces: list[tuple[int, ...]] , N:int):  
 *''' Функция проверяющая правильность изначального расположения фигур '''* board = [[0] \* N for \_ in range(N)]  
 for x, y in pieces:  
 board[x][y] = 1  
  
 for i in range(len(pieces)):  
 x, y = pieces[i]  
 if not valid\_position(board, x, y, N):  
 return False  
 return True  
  
def place\_pieces(board:list[list[int]], pieces: list[tuple[int, ...]], L:int, N:int, solutions:list, line:int, col:int):  
 *''' Функция для поиска всех возможных решений размещения фигуры '''* if L == 0:  
 solutions.append(pieces.copy())  
 return  
  
 for i in range(line, N):  
 for j in range(col if i == line else 0, N):  
 if board[i][j] == 0 and valid\_position(board, i, j, N):  
 board[i][j] = 1  
 pieces.append((i, j))  
 place\_pieces(board, pieces, L - 1, N, solutions, i, j)  
 pieces.pop()  
 board[i][j] = 0  
 col = 0  
  
def print\_board(board:list[list[int]], N:int):  
 *''' Функция выводящая в консоль доску '''* for i in range(N):  
 for j in range(N):  
 if board[i][j] == 1:  
 print("#", end=" ")  
 elif valid\_position(board, i, j, N):  
 print("0", end=" ")  
 else:  
 print("\*", end=" ")  
 print()  
  
def write\_output(file:any, solutions:list):  
 *''' Функция записывающая все решения в файл "output.txt" '''* with open(file, 'w') as f:  
 if not solutions:  
 f.write("no solutions\n")  
 else:  
 for solution in solutions:  
 f.write(' '.join(f"({x},{y})" for x, y in solution) + '\n')  
  
def main():  
 *''' Функция работающая со всеми функциями '''* N, L, K, initial\_positions = read\_input('input.txt')  
  
 board = [[0] \* N for \_ in range(N)]  
  
 for x, y in initial\_positions:  
 board[x][y] = 1  
 if not check\_starting\_position(initial\_positions, N):  
 write\_output('output.txt', [])  
 print("no solutions")  
 return  
  
 solutions = []  
 place\_pieces(board, initial\_positions, L, N, solutions, 0, 0)  
 write\_output('output.txt', solutions)  
  
 with open('output.txt', 'r') as file:  
 first\_line = file.readline().strip()  
 tuple\_strings = first\_line.replace("(", "").replace(")", "").split()  
 one\_solution = [tuple(map(int, s.split(','))) for s in tuple\_strings]  
 new\_board = [[0] \* N for \_ in range(N)]  
  
 for x, y in one\_solution:  
 new\_board[x][y] = 1  
 print\_board(new\_board, N)  
if \_\_name\_\_ == "\_\_main\_\_":  
 ''' Приводит в действие главную функцию '''  
 main()

# ЗАКЛЮЧЕНИЕ

В результате выполнения данной работы была разработана программа для решения задачи размещения фигур на доске с использованием функционального программирования на языке Python. Программа успешно обрабатывает входные данные и выводит результаты в требуемом формате.

# Список использованных источников

1. Учебные материалы и лекции.
2. Документация Python.