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: Коткин Вадим Иванович

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 4](#_Toc187676637)

[2.1.3. Листинг кода 4](#_Toc187676638)

[ЗАКЛЮЧЕНИЕ 5](#_Toc187676639)

[Список использованных источников 6](#_Toc187676640)

# Цели и формулировка задачи

ЦЕЛИ РАБОТЫ  
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 ЗАДАНИЯ  
 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.  
Вы работаете с фигурой: \*(Задание по вариантам)   
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.  
Ваш алгоритм должен быть оптимизирован, для работы с доской 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

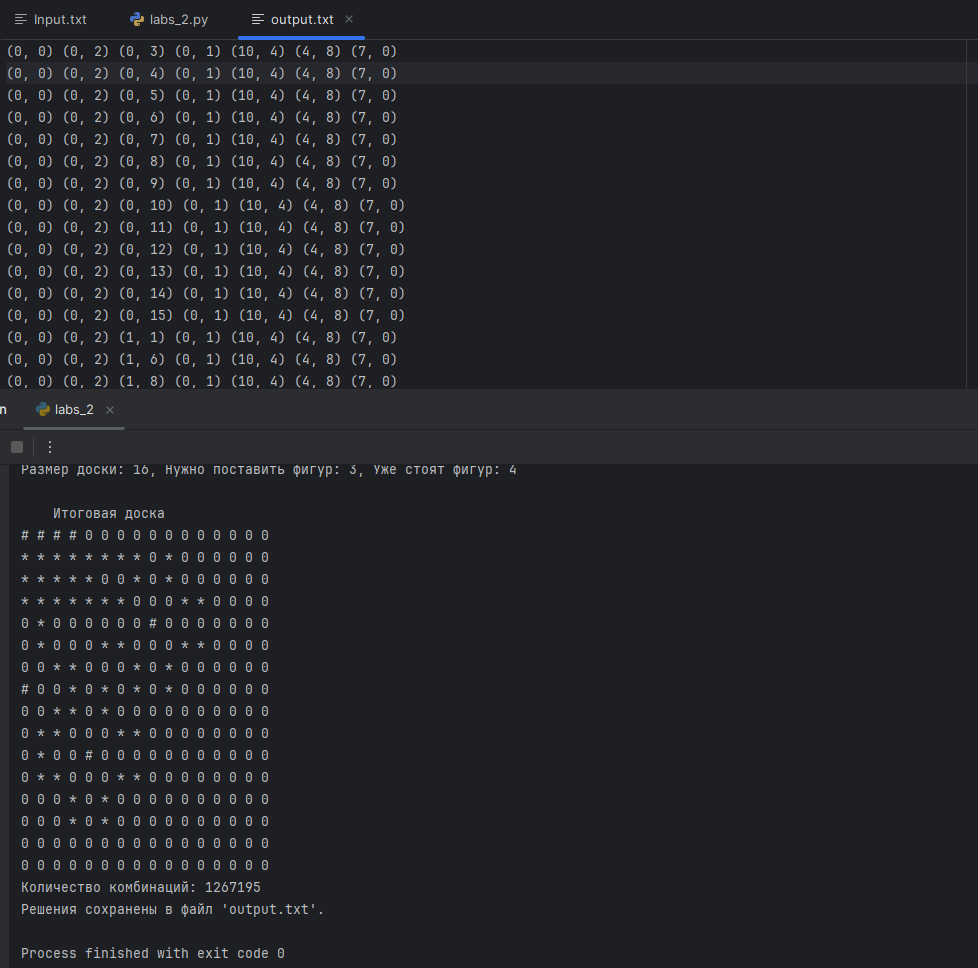
## **Реализация программы с использованием функционального программирования языка Python**

## Ход работы

1. **Чтение входных данных:**
   1. Программа начинает с вызова функции read\_file, которая открывает файл с именем Input.txt, считывает размер доски N, количество фигур, которые необходимо поставить, L, и количество уже стоящих фигур, K. Затем программа считывает координаты этих статичных фигур и сохраняет их в список coordinates\_const\_figure.
2. **Инициализация игрового поля:**
   1. Программа создаёт пустое игровое поле размером N x N, заполняя его нулями, что означает, что все клетки доступны для размещения фигур.
3. **Блокировка клеток для статичных фигур:**
   1. Для каждой из уже размещённых фигур (из списка coordinates\_const\_figure), программа вызывает функцию mark\_moves, которая помечает клетки, заблокированные этой фигурой и её возможными ходами. Фигуры в блокированных клетках не могут быть размещены.
4. **Поиск возможных решений с помощью бэктрекинга:**
   1. Затем начинается основной процесс решения задачи с помощью алгоритма бэктрекинга (обратного хода). Функция backtrack рекурсивно пытается найти все возможные решения, размещая на поле новые фигуры и блокируя клетки, на которые они могут попасть согласно возможным ходам.
   2. Для каждой клетки, куда можно поставить фигуру, программа вызывает mark\_moves, а затем рекурсивно пытается поставить оставшиеся фигуры. Если фигуры расставлены успешно, решение сохраняется.
5. **Подсчёт и вывод результатов:**
   1. После того как рекурсия завершена, количество найденных решений выводится на экран. Если решения найдены, программа выбирает первое решение из найденных и формирует итоговую доску, где будут отображены как уже стоящие, так и новые фигуры.
   2. Доска выводится в консоль с обозначениями: фигуры помечаются символом #, а доступные клетки для будущих ходов — символом \*.
6. **Сохранение решений в файл:**
   1. После вывода на экран, программа сохраняет все найденные решения в файл output.txt. Каждое решение записывается в формате координат фигур. Если решений нет, то в файл записывается строка "no solutions".
7. **Вывод итогов:**
   1. В конце программа выводит количество найденных решений, отображает итоговую доску и сообщает, что решения были сохранены в файл output.txt.

Программа ориентирована на решение задачи по размещению фигур на шахматной доске с учётом ограничений по движению и блокировке клеток.

## Демонстрация работы программы



## Листинг кода

1. # Функция для чтения данных из файла  
   def read\_file(file\_name):  
    global N, L, K  
    with open(file\_name, "r") as file:  
    first\_line = file.readline().strip()   
    N, L, K = map(int, first\_line.split())  
    coordinates\_const\_figure = []  
    for \_ in range(K):  
    x, y = map(int, file.readline().strip().split())  
    coordinates\_const\_figure.append((x, y))  
    return coordinates\_const\_figure  
     
   # Функция для пометки клеток, которые блокируются из-за хода фигуры  
   def mark\_moves(game\_field, x, y, moves\_of\_figure):  
    blocked\_cells = [] # Список заблокированных клеток  
    game\_field[x][y] = -1 # Помечаем клетку с фигурой как заблокированную  
    blocked\_cells.append((x, y))  
    for dx, dy in moves\_of\_figure: # Для каждого возможного хода фигуры  
    nx, ny = x + dx, y + dy  
    if 0 <= nx < N and 0 <= ny < N: # Проверяем, что новые координаты внутри доски  
    if game\_field[nx][ny] != -1: # Если клетка еще не заблокирована  
    game\_field[nx][ny] += 1 # Помечаем, что сюда можно ходить  
    blocked\_cells.append((nx, ny))  
    return blocked\_cells  
     
   # Функция для снятия блокировки с клеток после проверки текущего решения  
   def unmark\_moves(game\_field, blocked\_cells):  
    for bx, by in blocked\_cells:  
    if game\_field[bx][by] > 0: # Если клетка была доступной для хода  
    game\_field[bx][by] -= 1 # Уменьшаем ее "заблокированность"  
    elif game\_field[bx][by] == -1: # Если клетка была занята фигурой  
    game\_field[bx][by] = 0 # Снимаем блокировку  
     
   # Функция для поиска всех возможных решений с помощью рекурсии (обратного хода)  
   def backtrack(game\_field, remaining\_figures, moves\_of\_figure, start\_x, start\_y, solutions, current\_solution,  
    static\_coordinates):  
    if remaining\_figures == 0:  
    full\_solution = current\_solution + static\_coordinates # Составляем полное решение  
    solutions.append(full\_solution) # Добавляем решение в список  
    return 1 # Возвращаем 1, чтобы подсчитать количество решений  
    count = 0  
    for x in range(start\_x, N):  
    for y in range(start\_y if x == start\_x else 0, N):  
    if game\_field[x][y] == 0: # Если клетка свободна (не занята)  
    blocked\_cells = mark\_moves(game\_field, x, y, moves\_of\_figure) # Блокируем клетки, которые не доступны  
    current\_solution.append((x, y))  
    count += backtrack(game\_field, remaining\_figures - 1, moves\_of\_figure, x, y, solutions,  
    current\_solution, static\_coordinates) # Рекурсивно пробуем разместить следующую фигуру  
    unmark\_moves(game\_field, blocked\_cells) # Снимаем блокировки  
    current\_solution.pop()  
    start\_y = 0  
    return count  
     
   # Функция для сохранения решений в файл  
   def save\_solutions(solutions, file\_name="output.txt"):  
    with open(file\_name, "w") as file:  
    if not solutions:  
    file.write("no solutions\n")  
    else:  
    for solution in solutions: # Проходим по всем решениям  
    while len(solution) < (L + K): # Если решения не хватает до полного числа фигур  
    solution.append((-1, -1)) # Добавляем пустые клетки  
    file.write(" ".join(f"({x}, {y})" for x, y in solution) + "\n")  
     
   # Функция для вывода доски с обозначениями фигур и доступных клеток  
   def print\_board(game\_field, moves\_of\_figure):  
    for x in range(N):  
    for y in range(N):  
    if game\_field[x][y] == 1: # Если в клетке есть фигура  
    game\_field[x][y] = "#"  
    for dx, dy in moves\_of\_figure: # Помечаем клетки, на которые фигура может ходить  
    if 0 <= x+dx < N and 0 <= y+dy < N:  
    game\_field[x+dx][y+dy] = "\*"  
      
    # Выводим доску  
    for row in game\_field:  
    print(\*row) # Выводим каждую строку доски  
     
   # Основная функция  
   def main():  
    coordinates\_const\_figure = read\_file("Input.txt")  
    game\_field = [[0 for \_ in range(N)] for \_ in range(N)]  
    global moves\_of\_figure  
    moves\_of\_figure = [   
    (2, 1), (2, -1), (-2, 1), (-2, -1),  
    (1, 2), (1, -2), (-1, 2), (-1, -2),  
    (3, 1), (3, -1), (-3, 1), (-3, -1),  
    (1, 3), (1, -3), (-1, 3), (-1, -3)]  
     
    static\_coordinates = coordinates\_const\_figure # Список координат статичных фигур  
    for x, y in static\_coordinates: # Для каждой статичной фигуры блокируем клетки  
    mark\_moves(game\_field, x, y, moves\_of\_figure)  
     
    solutions = [] # Список для хранения решений  
    result = backtrack(game\_field, L, moves\_of\_figure, 0, 0, solutions, [], static\_coordinates) # Поиск решений  
    print(f"Размер доски: {N}, Нужно поставить фигур: {L}, Уже стоят фигур: {K}")  
    print()  
     
    if solutions:  
    chosen\_solution = solutions[0] # Выбираем первое решение  
    for x, y in chosen\_solution:  
    if (x, y) != (-1, -1):  
    mark\_moves(game\_field, x, y, moves\_of\_figure) # Помечаем клетки, занятые фигурами  
     
    output\_game\_field = [[0 for \_ in range(N)] for \_ in range(N)] # Создаем пустую доску для вывода  
    for x, y in static\_coordinates: # Размещаем статичные фигуры на доске  
    output\_game\_field[x][y] = 1  
     
    placed\_figures = 0 # Счетчик размещенных фигур  
    for x, y in chosen\_solution: # Размещаем остальные фигуры  
    if placed\_figures < L and (x, y) != (-1, -1):  
    output\_game\_field[x][y] = 1  
    placed\_figures += 1  
     
    print(f' Итоговая доска')  
    print\_board(output\_game\_field, moves\_of\_figure) # Выводим доску с фигурами и возможными ходами  
    print("Количество комбинаций:", result)  
    save\_solutions(solutions)  
    print("Решения сохранены в файл 'output.txt'.")  
     
     
   if \_\_name\_\_ == "\_\_main\_\_":  
    main()

# 

# ЗАКЛЮЧЕНИЕ

В ходе работы мы научились работать с текстовыми файлами. Научились работать с локальными и глобальными переменными. Научились работать с несколькими функциями, а не только с main. Научились нисходящему методу разработки программ. Написали программу, которая анализирует ходы.

# Список использованных источников