| Picture 1 | **МИНОБРНАУКИ РОССИИ**  **федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  «Санкт-Петербургский государственный морской технический университет» (СПбГМТУ) |
| --- | --- |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Факультет цифровых промышленных технологий

Направление подготовки 09.03.01.03 "Интеллектуальные технологии киберфизических систем"

**Лабораторная №2**

Студент 1 курса группы 20121

Очного отделения

Лаптев И. А.

Проверил:

Поделенюк П.П.

2023

# **Цель работы**

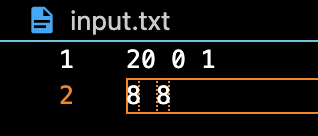
# Цель данной программы состоит в решении задачи о расстановке фигур на доске размером NxN так, чтобы полученное распределение соответствовало заданным правилам. В данном случае, программа находит всевозможные уникальные расстановки L фигур на доске размером N, выводит их в консоль, подсчитывает количество решений и записывает их в выходной файл "output.txt".

# **Ход работы**

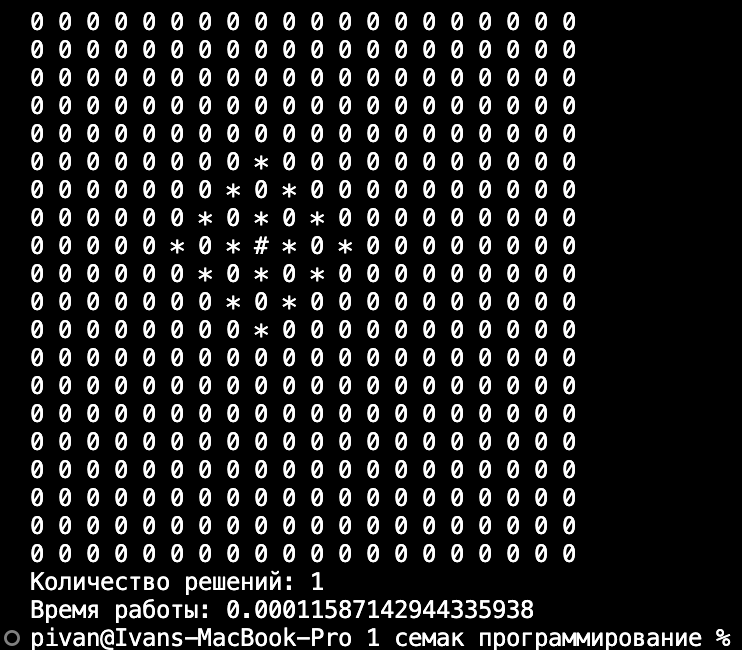
1. Функция possibleMoves: Определяет возможные ходы фигуры с текущими координатами (x, y) на доске.
2. Функция figureDislocation: Помечает текущую клетку (x, y) на доске фигурой и обозначает возможные ходы фигуры.
3. Функция otherFiguresDislocation: Помещает фигуры на доску.
4. Функция boardInitializer: Инициализирует доску NxN, заполняя ее нулями.
5. Функция boardPrinter: Выводит текущее состояние доски в консоль.
6. Функция recursia: Рекурсивная функция для нахождения всех расстановок фигур на доске. Выводит найденные решения в консоль.
7. Блок \_\_main\_\_:
   1. Считывает входные данные из файла "input.txt".
   2. Инициализирует переменные и вызывает рекурсивную функцию для нахождения решений.
   3. Записывает найденные решения в файл "output.txt".
   4. Выводит количество решений и время работы программы.

# **Результат работы**

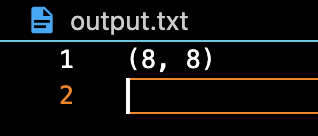
input.txt до запуска программы:



Терминал после запуска программы:



output.txt после запуска программы:



# **Вывод**

В рамках данной лабораторной работы была решена задача о расстановке фигур на шахматной доске таким образом, чтобы соблюсти правила хода каждой фигуры и при этом достичь заданного распределения.

Также были приобретены практические навыки по решению задачи комбинаторной оптимизации, применении рекурсивных функций и организации работы с файлами в языке программирования Python. Программа успешно решает поставленную задачу и предоставляет пользователям не только количество возможных вариантов, но и сами конфигурации доски для каждого решения.

# **Листинг кода**

import time

def possibleMoves(x, y):

# Функция возвращает множество возможных ходов для фигуры на доске с текущими координатами (x, y).

moves = {

(x - 3, y),

(x - 2, y - 1), (x - 2, y + 1),

(x - 1, y - 2), (x - 1, y), (x - 1, y + 2),

(x, y - 3), (x, y - 1), (x, y + 1), (x, y + 3),

(x + 1, y - 2), (x + 1, y), (x + 1, y + 2),

(x + 2, y - 1), (x + 2, y + 1),

(x + 3, y)

}

return moves

def figureDislocation(x, y, matrix):

# Функция помечает текущую клетку (x, y) фигурой и обозначает возможные ходы фигуры на доске matrix.

matrix[x][y] = '#'

for i, j in possibleMoves(x, y):

if 0 <= i < len(matrix) and 0 <= j < len(matrix):

matrix[i][j] = '\*'

return matrix

def otherFiguresDislocation(matrix, figures):

# Функция помещает фигуры на доску matrix.

for x, y in figures:

figureDislocation(x, y, matrix)

return matrix

def boardInitializer(N):

# Функция инициализирует доску NxN, заполняя ее нулями.

return [['0' for \_ in range(N)] for \_ in range(N)]

def boardPrinter(matrix):

# Функция выводит текущее состояние доски в консоль.

for row in matrix:

print(" ".join(row))

def recursia(N, L, solutions, currentSolution, figureCount):

# Рекурсивная функция для нахождения расстановок фигур на доске.

if figureCount == L:

# Проверяем, не добавлено ли уже текущее решение в множество решений

if tuple(sorted(currentSolution)) not in solutions:

solutions.add(tuple(sorted(currentSolution)))

# Выводим доску с текущим решением в консоль

boardPrinter(otherFiguresDislocation(boardInitializer(N), tuple(sorted(currentSolution))))

return

for i in range(N):

for j in range(N):

# Проверяем, что текущая клетка не занята другой фигурой и нет пересечений с возможными ходами

if all((moves not in currentSolution) for moves in possibleMoves(i, j)):

# Добавляем текущую клетку в решение и рекурсивно вызываем функцию для следующей фигуры

currentSolution.append((i, j))

recursia(N, L, solutions, currentSolution, figureCount + 1)

# Отменяем последний выбор для backtracking

currentSolution.pop()

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

# Чтение входных данных из файла

file = open("input.txt", "r")

N, L, K = map(int, file.readline().split())

figuresOnTheBoard = []

solutions = set()

# Чтение координат фигур из файла

for line in file.readlines():

x, y = map(int, line.split())

figuresOnTheBoard.append((x, y))

file.close()

# Замер времени начала выполнения программы

start\_time = time.time()

# Вызов рекурсивной функции для нахождения решений

recursia(N, L, solutions, figuresOnTheBoard, 0)

# Замер времени завершения выполнения программы

end\_time = time.time()

print(f"Количество решений: {len(solutions)}")

print(f"Время работы: {end\_time - start\_time}")

# Если найдены решения, записываем их в файл

if solutions:

uniqueSolutions = {tuple(sorted(solution)) for solution in solutions}

stringSolution = [" ".join(map(str, solution)) + "\n" for solution in uniqueSolutions]

with open("output.txt", "w") as output\_file:

output\_file.writelines(stringSolution)

else:

print("no solutions")