

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 работа №2
по дисциплине “Программирование”




Студент 1 курса группы 20121
Очного отделения
Хохлов Д.Р.
Проверил:
Поделенюк П. П.

Цель работы - заключается в разработке программы на языке программирования Python, решающей задачу размещения шахматных фигур (в моём случае визирей) на доске размером NxN таким образом, чтобы они не находились под угрозой боя друг с другом.

Ход работы

Работа имеет 4 рабочих файла:

- main.py - основные принципы работы записаны в этом файле.
- config.py - в этом файле записаны функции для получения нужных данных (get)
- input.txt - тут записаны данные характеристик (например N - размер шахматной доски)
- output.txt - сюда будут записываться результаты вычислений

 config.py	29.01.2024 17:47	JetBrains PyCharm	1 КБ
 Input.txt	18.01.2024 3:38	Текстовый документ	1 КБ
 main.py	29.01.2024 18:49	JetBrains PyCharm	3 КБ
 output.txt	29.01.2024 18:50	Текстовый документ	89 246 КБ

Работа алгоритма состоит из нескольких этапов:

- Чтение данных
- Запуск рекурсивной функции
- Запись результатов и их вывод

Разберем каждый этап по-подробней

Чтение данных:

- импортируем файл “*config.py*” в главный файл “*main.py*”
- запускаем функцию “*init_data()*” инициализируем некоторые переменные
- запускаем функцию “*get_data()*”(она считывает данные из “*input.txt*”) из файла “*config.py*” и записываем в переменные N,L,K, board

```
main.py

if __name__ == "__main__":
    startTime, board, solutions, allSolutions, N, L, K = init_data()
```

```

main.py

def init_data():
    startTime = time.time() #запуск таймера

    solutions = []
    allSolutions = []

    N,L,K,board = get_data(solutions, make_move)

    return startTime, board, solutions, allSolutions, N, L, K

```

```

config.py

def get_data(solutions:list, make_move):
    with open("input.txt", "r") as input_file:
        N, L, K = map(int, input_file.readline().split())

        board = ["0" * N for _ in range(N)]

        for _ in range(K):
            row, col = map(int, input_file.readline().split())
            make_move(board, N, row, col, solutions)
    return N,L,K,board

```

Запуск рекурсии:

- Программа использует рекурсивную функцию “find_solutions” в файле *“main.py”*, которая принимает текущую доску, количество оставшихся ферзей для размещения, текущую позицию (строку и столбец) и текущий список решений.
- В каждом шаге рекурсии программа проверяет возможность размещения ферзя в текущей позиции. Если ферзь может быть размещен, то он добавляется на доску, и соответствующие клетки отмечаются, чтобы исключить возможность размещения других ферзей, которые могли бы быть угрозой.
- Рекурсия продолжается для следующей фигуры и так далее.

- Если все фигуры успешно размещены, программа добавляет текущее решение в список найденных решений.
- Если не удастся разместить фигуру в текущей позиции, программа откатывается назад и пытается другие варианты размещения.

```

def find_solution(L: int, N: int, board: list, row: int, col: int, solutions: list,
allSolutions: list) -> None:
    while True:
        col += 1
        if col >= N:
            col = 0
            row += 1
        if row >= N: break
        if board[row][col] != "0": continue
        now_board = list(board)
        now_solutions = list(solutions)
        make_move(now_board, N, row, col, now_solutions)
        if L - 1 == 0:
            allSolutions.append(now_solutions)
            if len(allSolutions) == 1:
                for i_row in now_board:
                    print(i_row)
            continue
        find_solution(L - 1, N, now_board, row, col, now_solutions, allSolutions)

```

Запись результатов и их вывод:

- После завершения выполнения алгоритма, программа выводит количество найденных решений и записывает их в файл "output.txt". А также выводит время работы программы и количество решений

```

def display_solutions(solutions: list, startTime) -> None:
    print(f"Всего решений:{len(solutions)}")

    with open("output.txt", "w") as output_file:
        if not solutions:
            output_file.write("no solutions")
        else:
            for solution in solutions:
                output_file.write(" ".join(map(str, solution)) + "\n")

    endTime = time.time()
    print(f"Время выполнения программы: {endTime - startTime}s")

```

Результат работы

Введем характеристики в файл "input.txt":

- $N = 16, L = 3, K = 4$
- Поставим некоторые фигуры на координаты:
 - (0,1)
 - (10,4)
 - (4,8)
 - (7,0)



Через небольшое количество времени получаем результат:

```
Размер доски: 16x16
Фигур на доске: 4
Требуется разместить: 3
*#####000
0*00*00*00*00000
0*00*00*0*00000
00000000*0000000
000000**#**00000
*0000000*0000000
*0000000*0000000
#**0000000000000
*000*00000000000
*000*00000000000
00**#**000000000
0000*00000000000
0000*00000000000
0000000000000000
0000000000000000
0000000000000000
Всего решений:1709809
Время выполнения программы: 10.410286903381348s
```

Вывод

Была написана программа на Python, построенная на рекурсии, размещающая шахматные фигуры на правильных клетка, оптимизированная для быстрой обработки большого количества входных данных

Весь код:



```
def get_moves(row: int, col: int) -> list[tuple[int, int]]:
    return [
        (row, col - 1), (row, col - 2), # Влево
        (row, col + 1), (row, col + 2), # Вправо
        (row - 1, col), (row - 2, col), # Вверх
        (row + 1, col), (row + 2, col) # Вниз
    ]

def get_data(solutions: list, make_move):
    with open("input.txt", "r") as input_file:
        N, L, K = map(int, input_file.readline().split())

        board = ["0" * N for _ in range(N)]

        for _ in range(K):
            row, col = map(int, input_file.readline().split())
            make_move(board, N, row, col, solutions)
    return N, L, K, board
```

```

main.py

import time
from config import get_moves, get_data

def display_solutions(solutions: list, startTime) -> None:
    print(f"Всего решений: {len(solutions)}")
    with open("output.txt", "w") as output_file:
        if not solutions:
            output_file.write("no solutions")
        else:
            for solution in solutions:
                output_file.write(" ".join(map(str, solution)) + "\n")
    endTime = time.time()
    print(f"Время выполнения программы: {endTime - startTime}s")

def make_move(board, N: int, row: int, col: int, solutions: list) -> None:
    solutions.append((row, col))
    board[row] = board[row][:col] + "#" + board[row][col+1:]
    moves = get_moves(row, col)
    for row_index, col_index in moves:
        if 0 <= row_index < N and 0 <= col_index < N and board[row_index][col_index]
        != "#":
            board[row_index] = board[row_index][:col_index] + "*" + board[row_index]
            [col_index+1:]

def find_solution(L: int, N: int, board: list, row: int, col: int, solutions: list,
allSolutions: list) -> None:
    while True:
        col += 1
        if col >= N:
            col = 0
            row += 1
        if row >= N: break
        if board[row][col] != "0": continue
        now_board = list(board)
        now_solutions = list(solutions)
        make_move(now_board, N, row, col, now_solutions)
        if L - 1 == 0:
            allSolutions.append(now_solutions)
            if len(allSolutions) == 1:
                for i_row in now_board:
                    print(i_row)
            continue
        find_solution(L - 1, N, now_board, row, col, now_solutions, allSolutions)

def init_data():
    startTime = time.time() #запуск таймера
    solutions = []
    allSolutions = []
    N, L, K, board = get_data(solutions, make_move)
    return startTime, board, solutions, allSolutions, N, L, K

if __name__ == "__main__":
    startTime, board, solutions, allSolutions, N, L, K = init_data()
    print(f"Размер доски: {N}x{N}\n"
          f"Фигур на доске: {K}\n"
          f"Требуется разместить: {L}")
    if L == 0:
        if not (len(solutions) == 0):
            allSolutions.append(solutions)
        for i_row in board:
            print(i_row)
        display_solutions(allSolutions, startTime)
    find_solution(L, N, board, 0, -1, solutions, allSolutions)
    display_solutions(allSolutions, startTime)

```

