Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Санкт-Петербургский государственный морской технический университет»

(СПбГМТУ)

Факультет Цифровых Промышленных Технологий

Отчёт по лабораторной работе №2

Выполнил:

Студент группы 20121

Петров Е.М.

Проверил:

Поделенюк П.П.

Цели работы:

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

Ход работы:

1. Проанализировав задачу, обнаруживаем следующие полезные особенности:
   1. Если клетка не находится под боем другой фигуры и не занята ею, то мы можем поставит на эту клетку нашу фигуру и она не будет бить ни одну из установленных ранее фигур
   2. Все возможные ходы фигуры, поставленной на клетку с координатами x и y можно выразить как:  
       (x+1,y+1), (x+1,y), (x+1,y-1), (x,y-1), (x-1,y-1), (x-1,y), (x-1,y+1), (x,y+1), (x+1,y+3), (x-1,y+3), (x-3,y+1), (x-3,y-1), (x-1,y-3), (x+1,y-3), (x+3,y-1), (x+3,y+1)
   3. Двумерное пространство поля для простоты обхода мы можем представить как одномерный список размерностью N\*\*2, в такой ситуации y координату поля можно выразить как “Нынешняя позиция в списке//N”, а x координату поля можно выразить как “Нынешняя позиция в списке%N”
2. Создаём функцию FigureMoves, отсекающую ходы фигуры на указанной координате, выходящие за рамки нашего поля
3. Создаём функцию PlaceFigure, ставящую фигуру на клетку с указанными координатами.
4. Создаём функцию RemoveFigure, убирающую фигуру с клетки с указанными координатами.
5. Создаём рекуррентную функцию, реализующую следующий алгоритм:
   1. Ставим фигуру на поле, переходим на более низкий уровень рекурсии
   2. Если это была последняя фигура из тех, которые мы должны были поставить, то мы записываем получившуюся расстановку в файл и возвращаемся на предыдущий уровень рекурсии
   3. Если мы вернулись с более низкого уровня рекурсии, снимаем фигуру и переходим к следующей свободной клетке
6. Точками остановки в рекурсии из пункта пять будут две ситуации:
   1. Мы поставили все «дополнительные» фигуры
   2. Мы не можем поставить на поле больше ни одной фигуры
7. Обе точки остановки возвращают None, но в первой ситуации мы записываем получившуюся расстановку в файл
8. В функции main считываем данные их файла, расставляем константные фигуры на поле
9. Переменные side и File делаем глобальными для того чтобы не добавлять размерность игрового поля в параметры каждой функции, и для того чтобы записывать данные в файл в рамках рекурсии.  
   Переменные Flag и OutputField нужны для упрощения вывода ответа в терминал, они тоже являются глобальными переменными
10. Вызываем рекурсию
11. Выводим ответ в консоль, более чётко обозначаем ситуацию с отсутствием решений

Результат работы:

**Пример входного файла:**

  
Вывод:

  
Результат в файле:  


И ещё 51 различная строка

**Второй пример входного файла:**



Вывод:



Аналогичный результат в файле:



Вывод:

При работе было получено более глубокое понимание рекуррентных алгоритмов, отработано разбиение крупной задачи на более мелкие, работа с несколькими функциями, более чёткое представление о работе с глобальными переменными, углублены знания по работе с аннотацией в функциях, улучшены навыки анализа поставленной задачи.

Листинг Кода:

import copy

from typing import Generator

from typing import TypeAlias

# Глобальная аннотация шахматного поля, чтобы не пришлось писать из чего оно состоит в каждой функции, которая его использует

field:TypeAlias=list[list[int]]

# Функция для вывода нынешнего состояния доски: "-1" - фигура, "0" - свободная клетка, любое другое число - клетка под боем(число указывает сколько фигур бьют на эту клетку)

def PrintField(Field:field)-> None:

    for i in range(side):

        print(Field[i])

# Генератор, отсекающий ходы фигуры, выходящие за рамки игрового поля

def FigureMoves(x:int,y:int)-> Generator[tuple[int, int], None, None]:

    for i in ((x+1,y+1),(x+1,y),(x+1,y-1),(x,y-1),(x-1,y-1),(x-1,y),(x-1,y+1),(x,y+1),(x+1,y+3),(x-1,y+3),(x-3,y+1),(x-3,y-1),(x-1,y-3),(x+1,y-3),(x+3,y-1),(x+3,y+1)):

        if not(i[0]>side-1 or i[1]>side-1 or i[0]<0 or i[1]<0):

            yield i

# Функция, ставящая фигуру на указанную клетку

def PlaceFigure(x:int,y:int,Field:field)-> field:

    # Расстановка клеток боя на поле

    for i in FigureMoves(x,y):

        Field[i[1]][i[0]]+=1

    # Ставим саму фигуру

    Field[y][x]=-1

    return Field

# Функция, убирающая фигуру с указанной клетки

def RemoveFigure(x:int,y:int,Field:field)-> field:

    # Убираем бой этой фигуры с тех клеток, на которых он был

    for i in FigureMoves(x,y):

        Field[i[1]][i[0]]-=1

    # Убираем саму фигуру

    Field[y][x]=0

    return Field

# Рекуррентная функция, с точкой остановки "Все дополнительные фигуры поставлены"

def recursion(place:int,CurrentPos:int,Field:field,Figures:list=[])-> None:

    # Указываю на то что буду менять в этой функции значения глобальных переменных

    global Flag

    global OutputField

    # Точка остановки, если нам не нужно больше ставить фигуры на поле, мы записываем нынешнюю расстановку в файл и выходим из этой ветки рекурсии

    if place==0:

        # Здесь мы записываем первую удачную расстановку, чтобы потом её вывести в консоль, после чего больше не заходим в это условие

        if Flag==1:

            OutputField=copy.deepcopy(Field)

            Flag=False

        File.write(",".join(map(str,Figures))+'\n')

        return

    # Тело рекурсии

    else:

        # Представляем нашу шахматную доску в виде последовательности клеток, избавляясь от двумерности. Идти по доске мы начинаем с "Нулевой позиции"

        for pos in range(CurrentPos,side\*\*2):

            #Если мы находим пустую клетку, то ставим на неё фигуру, записываем её местоположение, \\

            # меняем значение "Нулевой позиции" на следующую клетку после той, на которую мы поставили фигуру

            if Field[pos//side][pos%side]==0:

                Field=PlaceFigure(pos%side,pos//side,Field)

                Figures.append((pos%side,pos//side))

                recursion(place-1,pos+1,Field,Figures)

                # По достижению точки остановки мы возвращаемся сюда и переходим в другую ветку, снимая поставленную ранее на эту клетку фигуру

                Figures.pop(len(Figures)-1)

                Field=RemoveFigure(pos%side,pos//side,Field)

def main():

    # Объявление глоббальных переменных. 1) Сторонаа поля 2) Само шахматное поле 3) Поле для вывода в консоль 4) Флаг(нужен для более простого вывода поля в консоль)

    global side

    global File

    global OutputField

    global Flag

    Flag=True

    OutputField=[]

    # Считывание данных из файла

    with open('input.txt','r') as f:

        side,place,OnField=map(int,f.readline().split())

        # Генерация поля, расстановка константных фигур

        Field=[[0]\*side for \_ in range(side)]

        for \_ in range(OnField):

            x,y=map(int,f.readline().split())

            Field=PlaceFigure(x,y,Field)

    # Рекуррентная магия(подробнее в самой функции рекурсии), запись всех возможных расстановок в файл

    File=open("output.txt",'w')

    recursion(place,0,Field)

    File.close()

    # Вывод количества расстановок и поля в консоль, более чёткое обозначение отсутствия решений

    NoSolution=False

    with open("output.txt",'r') as f:

        file=f.readlines()

        if len(file)==0:

            NoSolution=True

            print('No Solutions')

        else:

            print(len(file))

            PrintField(OutputField)

    if NoSolution:

        with open("output.txt",'w') as f:

            f.write('No Solutions')

if \_\_name\_\_=='\_\_main\_\_':

    main()