**Министерство науки и высшего образования Российской Федерации**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

**(Университет ИТМО)**

Факультет **Инфокоммуникационных технологий**

Образовательная программа **Мобильные и сетевые технологии**

Направление подготовки(специальность) **09.03.03 Прикладная информатика**

**ПРАКТИЧЕСКАЯ РАБОТА №3**

**По дисциплине «Программирование»**

**Тема: Судоку.**

**Выполнил** Проскуряков Р. В. K3139

**Проверил** Терещенко В. В.

**Дата** 30.11.2023

**Санкт-Петербург 2023**

**Цель работы**

Познакомиться с функциональным программированием во время написания решателя Судоку.

**Ход работы**

Ход работы заключался в написании внутренностей функций:

1. Чтение пазлов

def read\_sudoku(path: tp.Union[str, pathlib.Path]) -> tp.List[tp.List[str]]:

""" Прочитать Судоку из указанного файла """

path = pathlib.Path(path)

with path.open() as f:

puzzle = f.read()

return create\_grid(puzzle)

def create\_grid(puzzle: str) -> tp.List[tp.List[str]]:

digits = [c for c in puzzle if c in "123456789."]

grid = group(digits, 9)

return grid

def display(grid: tp.List[tp.List[str]]) -> None:

"""Вывод Судоку """

width = 2

line = "+".join(["-" \* (width \* 3)] \* 3)

for row in range(9):

print(

"".join(

grid[row][col].center(width) + ("|" if str(col) in "25" else "") for col in range(9)

)

)

if str(row) in "25":

print(line)

print()

def group(values: tp.List[T], n: int) -> tp.List[tp.List[T]]:

"""

Сгруппировать значения values в список, состоящий из списков по n элементов

>>> group([1,2,3,4], 2)

[[1, 2], [3, 4]]

>>> group([1,2,3,4,5,6,7,8,9], 3)

[[1, 2, 3], [4, 5, 6], [7, 8, 9]]

"""

if n\*\*2 > len(values):

raise "group: мало значений для составления таблицы"

res = [[' ' for i in range(n)] for i in range(n)]

for i in range(n):

for j in range(n):

res[i][j] = values[i\*n + j]

return res

1. Разбивка на строки, колонки и блоки

def get\_row(grid: tp.List[tp.List[str]], pos: tp.Tuple[int, int]) -> tp.List[str]:

"""Возвращает все значения для номера строки, указанной в pos

>>> get\_row([['1', '2', '.'], ['4', '5', '6'], ['7', '8', '9']], (0, 0))

['1', '2', '.']

>>> get\_row([['1', '2', '3'], ['4', '.', '6'], ['7', '8', '9']], (1, 0))

['4', '.', '6']

>>> get\_row([['1', '2', '3'], ['4', '5', '6'], ['.', '8', '9']], (2, 0))

['.', '8', '9']

"""

return grid[pos[0]].copy()

def get\_col(grid: tp.List[tp.List[str]], pos: tp.Tuple[int, int]) -> tp.List[str]:

"""Возвращает все значения для номера столбца, указанного в pos

>>> get\_col([['1', '2', '.'], ['4', '5', '6'], ['7', '8', '9']], (0, 0))

['1', '4', '7']

>>> get\_col([['1', '2', '3'], ['4', '.', '6'], ['7', '8', '9']], (0, 1))

['2', '.', '8']

>>> get\_col([['1', '2', '3'], ['4', '5', '6'], ['.', '8', '9']], (0, 2))

['3', '6', '9']

"""

return [grid[i][pos[1]] for i in range(len(grid))]

def get\_block(grid: tp.List[tp.List[str]], pos: tp.Tuple[int, int]) -> tp.List[str]:

"""Возвращает все значения из квадрата, в который попадает позиция pos

>>> grid = read\_sudoku('src/lab3/puzzle1.txt')

>>> get\_block(grid, (0, 1))

['5', '3', '.', '6', '.', '.', '.', '9', '8']

>>> get\_block(grid, (4, 7))

['.', '.', '3', '.', '.', '1', '.', '.', '6']

>>> get\_block(grid, (8, 8))

['2', '8', '.', '.', '.', '5', '.', '7', '9']

"""

row, col = pos

row = row - row % 3

col = col - col % 3

res = []

for i in range(3):

res += grid[row + i][col:col+3]

return res

1. Алгоритм решения Судоку

def solve(grid: tp.List[tp.List[str]]) -> tp.Optional[tp.List[tp.List[str]]]:

""" Решение пазла, заданного в grid """

""" Как решать Судоку?

1. Найти свободную позицию

2. Найти все возможные значения, которые могут находиться на этой позиции

3. Для каждого возможного значения:

3.1. Поместить это значение на эту позицию

3.2. Продолжить решать оставшуюся часть пазла

>>> grid = read\_sudoku('puzzle1.txt')

>>> solve(grid)

[['5', '3', '4', '6', '7', '8', '9', '1', '2'], ['6', '7', '2', '1', '9', '5', '3', '4', '8'], ['1', '9', '8', '3', '4', '2', '5', '6', '7'], ['8', '5', '9', '7', '6', '1', '4', '2', '3'], ['4', '2', '6', '8', '5', '3', '7', '9', '1'], ['7', '1', '3', '9', '2', '4', '8', '5', '6'], ['9', '6', '1', '5', '3', '7', '2', '8', '4'], ['2', '8', '7', '4', '1', '9', '6', '3', '5'], ['3', '4', '5', '2', '8', '6', '1', '7', '9']]

"""

pos = find\_empty\_positions(grid)

if pos[0] == -1:

return grid

for possible\_values in find\_possible\_values(grid, pos):

grid[pos[0]][pos[1]] = possible\_values

res = solve(grid)

if res != [[]]:

return res

grid[pos[0]][pos[1]] = '.'

return [[]]

1. Поиск возможных решений

def find\_empty\_positions(grid: tp.List[tp.List[str]]) -> tp.Optional[tp.Tuple[int, int]]:

"""Найти первую свободную позицию в пазле

>>> find\_empty\_positions([['1', '2', '.'], ['4', '5', '6'], ['7', '8', '9']])

(0, 2)

>>> find\_empty\_positions([['1', '2', '3'], ['4', '.', '6'], ['7', '8', '9']])

(1, 1)

>>> find\_empty\_positions([['1', '2', '3'], ['4', '5', '6'], ['.', '8', '9']])

(2, 0)

"""

for i in range(len(grid)):

for j in range(len(grid)):

if grid[i][j] == '.':

return i, j

return -1, -1

def find\_possible\_values(grid: tp.List[tp.List[str]], pos: tp.Tuple[int, int]) -> tp.Set[str]:

"""Вернуть множество возможных значения для указанной позиции

>>> grid = read\_sudoku('puzzle1.txt')

>>> values = find\_possible\_values(grid, (0,2))

>>> values == {'1', '2', '4'}

True

>>> values = find\_possible\_values(grid, (4,7))

>>> values == {'2', '5', '9'}

True

"""

act\_row = set(get\_row(grid, pos))

act\_col = set(get\_col(grid, pos))

act\_block = set(get\_block(grid, pos))

act\_all = act\_row | act\_col | act\_block

return alphabet - act\_all

1. Проверка решения

def check\_solution(solution: tp.List[tp.List[str]]) -> bool:

""" Если решение solution верно, то вернуть True, в противном случае False

>>> solution = [["5", "3", "4", "6", "7", "8", "9", "1", "2"], ["6", "7", "2", "1", "9", "5", "3", "4", "8"], ["1", "9", "8", "3", "4", "2", "5", "6", "7"], ["8", "5", "9", "7", "6", "1", "4", "2", "3"], ["4", "2", "6", "8", "5", "3", "7", "9", "1"], ["7", "1", "3", "9", "2", "4", "8", "5", "6"], ["9", "6", "1", "5", "3", "7", "2", "8", "4"], ["2", "8", "7", "4", "1", "9", "6", "3", "5"], ["3", "4", "5", "2", "8", "6", "1", "7", "9"], ]

>>> check\_solution(solution)

True

>>> solution = [["5", "5", "4", "6", "7", "8", "9", "1", "2"], ["6", "7", "2", "1", "9", "5", "3", "4", "8"], ["1", "9", "8", "3", "4", "2", "5", "6", "7"], ["8", "5", "9", "7", "6", "1", "4", "2", "3"], ["4", "2", "6", "8", "5", "3", "7", "9", "1"], ["7", "1", "3", "9", "2", "4", "8", "5", "6"], ["9", "6", "1", "5", "3", "7", "2", "8", "4"], ["2", "8", "7", "4", "1", "9", "6", "3", "5"], ["3", "4", "5", "2", "8", "6", "1", "7", "9"], ]

>>> check\_solution(solution)

False

"""

for i in range(len(solution)):

if(set(get\_row(solution, (i, 0))) != alphabet

or set(get\_col(solution, (i, 0))) != alphabet

or set(get\_block(solution, (i \* 3 % 9, i // 3 \* 3))) != alphabet):

return False

return True

1. Генерация новых пазлов

def generate\_sudoku(N: int) -> tp.List[tp.List[str]]:

"""Генерация судоку заполненного на N элементов

>>> grid = generate\_sudoku(40)

>>> sum(1 for row in grid for e in row if e == '.')

41

>>> solution = solve(grid)

>>> check\_solution(solution)

True

>>> grid = generate\_sudoku(1000)

>>> sum(1 for row in grid for e in row if e == '.')

0

>>> solution = solve(grid)

>>> check\_solution(solution)

True

>>> grid = generate\_sudoku(0)

>>> sum(1 for row in grid for e in row if e == '.')

81

>>> solution = solve(grid)

>>> check\_solution(solution)

True

"""

N = min(N, 81)

res = [['.' for i in range(9)] for i in range(9)]

res = solve(res)

all\_poses = [[i, j] for i in range(9) for j in range(9)]

for pos in random.sample(all\_poses, 81 - N):

res[pos[0]][pos[1]] = '.'

return res

1. Замеры времени решений и многопроцессорная обработка

def run\_solve(filename: str) -> None:

grid = read\_sudoku(filename)

start = time.time()

solution = solve(grid)

end = time.time()

print()

print(filename, "ответ получен за:", end - start)

if not solution:

print("Puzzle can't be solved")

elif check\_solution(solution) == False:

print("Puzzle неправильно решён")

else:

display(solution)

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

for filename in ("puzzle1.txt", "puzzle2.txt", "puzzle3.txt"):

p = multiprocessing.Process(target=run\_solve, args=(filename,))

p.start()

**Вывод**

В результате проделанной работы мы реализовали познакомились с функциональным программированием, работой с файлами, замерами времени решений и многопроцессорной обработкой на языке python.