Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Лабораторная работа №3
по дисциплине: «Теория Алгоритмов»
Тема: «Рекурсивные алгоритмы»

Выполнила ученица 223 группы
Никончук А.П.

Проверил
Сорокин Д.С. _____

Санкт-Петербург,
2015 г.

Задание 1: написать функцию group(), которая принимает пазл и размер доски n, а в качестве результата работы возвращает матрицу n*n

```
def group(values, n):
    r=[]
    for rows in range(n):
        r.append([])
        for element_index in range(n):
            r[rows].append(values[rows*n + element_index])
    return r
    pass
```

Задача 2: написать три функции get_row(), get_col() и get_block(), каждая из которых принимает два аргумента, пазл (values) и позицию (pos), для которой мы пытаемся найти верное число

```
def get_row(values, pos):
    ''' Возвращает все значения для номера строки, указанной в pos'''
    return values[pos[0]]

def get_col(values, pos):
    ''' Возвращает все значения для номера столбца, указанного в pos'''
    column = []
    for i in range(len(values)):
        column.append(values[i][pos[1]])
    return column

def get_block(values, pos):
    ''' Возвращает все значения из квадрата, в который попадает позиция pos '''
    x = pos[0]-pos[0] % 3
    y = pos[1]-pos[1] % 3
    m =[]
    for i in range(x, x+3):
        for j in range(y,y+3):
            m.append(values[i][j])
    return m
```

Задача 3.1: функция solve(), которая принимает один аргумент - пазл, а возвращает заполненную значениями доску

```
def solve(grid):

    pos = find_empty_positions(grid)
    possible_values = find_possible_values(grid, pos)
    if pos ==(-1, -1):
        return grid
    if possible_values:
        for i in possible_values:
            x = grid[pos[0]][:pos[1]] + [str(i)] + grid[pos[0]][pos[1] + 1:]
            solution = solve(grid[:pos[0]] + [x] + grid[pos[0] + 1:])
            if solution:
                return solution
```

Задача 3.2: написать функцию find_empty_positons(), которая принимает один аргумент - пазл и возвращает первую попавшуюся свободную позицию

```
def find_empty_positions(grid):
    ''' Найти первую свободную позицию в пазле'''
    for i in range(9):
        for j in range(9):
            if grid[i][j] == '.':
                return (i, j)
    return (-1, -1)
```

Задача 3.3: Кроме поиска свободных позиций, также необходимо искать значения, которые на эту позицию можно поставить

```
def find_possible_values(grid, pos):
    ''' Вернуть все возможные значения для указанной позиции '''
    m = []
    for k in range(1, 10):
        if not (str(k) in get_row(grid, pos) or str(k) in get_col(grid, pos) or str(k) in get_block(grid, pos)):
            m.append(k)
    return m
```

Задача 4: написать функцию check_solution(), которая проверяет наше решение

```
def check_solution(grid):

    def help_check_solution(L):
        for i in range(1, 10):
            if L.count(i) > 1:
                return False
        return True


    for i in range(9):
        if not help_check_solution(get_row(grid, (i, i))) or not help_check_solution(get_col(grid, (i, i))):
            return False
    for index in [(1, 1), (1, 4), (1, 7), (4, 1), (4, 4), (4, 7), (7, 1), (7, 4), (7,7)]:
        if not help_check_solution(get_block(grid, index)):
            return False
    return True
```


Администратор@TATYANNA ~/cs101myworks/lab_3 (master)
$ python -i sudoku.py
>>> grid = read_sudoku('s1.txt')
>>> grid
[['5', '3', '.', '.', '7', '.', '.', '.', '.'], ['6', '.', '.', '1', '9', '5', '.', '.', '.'], ['.', '9', '8', '.', '.', '.', '.', '6', '.'], ['8', '.', '.', '.', '6', '.', '.', '.', '3'], ['4', '.', '.', '8', '.', '3', '.', '.', '1'], ['7', '.', '.', '.', '2', '.', '.', '.', '6'], ['.', '6', '.', '.', '.', '.', '2', '8', '.'], ['.', '.', '.', '4', '1', '9', '.', '.', '5'], ['.', '.', '.', '.', '8', '.', '.', '7', '9']]
>>> display(grid)
5 3 . |. 7 . |. . .
6 . . |1 9 5 |. . .
. 9 8 |. . . |. 6 .
------+------+------
8 . . |. 6 . |. . 3
4 . . |8 . 3 |. . 1
7 . . |. 2 . |. . 6
------+------+------
. 6 . |. . . |2 8 .
. . . |4 1 9 |. . 5
. . . |. 8 . |. 7 9
>>> solution = solve(grid)
>>> display(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
>>>