



دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)
دانشکده ریاضی و علوم کامپیوتر

گزارش سوم - سودو کو

نگار ابوالحسنی - 9833002

استاد قطعی - دکتر یوسفی مهر

فروردین هزار و چهار صد و دو

چکیده

سودوکو یک بازی پازلی است که در آن یک جدول $n \times n$ داریم که هر خانه آن باید توسط یک عدد بین 1 تا n پر شود، به صورتی که از هر عدد در هر سطر، ستون و زیرجدول مرتبطش دقیقاً یکی وجود داشته باشد.

در ادامه، ما با استفاده از الگوریتم `backtracking`، و با بهبود آن با `forward`، `MRV`، `LCV`، `checking` به حل سودوکو پرداختیم.

برنامه ما قادر به حل بسیار سریع جدول‌های سودوکو در چند صدم ثانیه است. در ادامه به توضیح الگوریتم‌ها و کد می‌پردازیم.

چکیده.....	أ
فصل اول مقدمه.....	1
فصل دوم منطق بازی و اجرای آن در پایتون.....	2
منطق بازی و اجرای آن در پایتون.....	2
2-1 solver.py.....	2
2-2 Utils.py.....	3
2-3 Property.py.....	5
2-4 Board.py.....	7
فصل سوم نگارش صحیح نگارش صحیح.....	Error! Bookmark not defined.
3-1 فارسی نویسی.....	Error! Bookmark not defined.
3-2 رعایت املاي صحیح فارسی.....	Error! Bookmark not defined.
3-3 رعایت قواعد نشانه گذاری.....	Error! Bookmark not defined.
3-3-1 ویرگول و نقطه.....	Error! Bookmark not defined.
3-3-2 دو نقطه.....	Error! Bookmark not defined.
3-3-3 گیومه.....	Error! Bookmark not defined.
3-3-4 نشانه پرشی.....	Error! Bookmark not defined.
3-3-5 خط تیره.....	Error! Bookmark not defined.
3-3-6 پرانتز.....	Error! Bookmark not defined.
فصل چهارم سبک ها و قلم ها سبک ها و قلم ها.....	Error! Bookmark not defined.
4-1 قلم های فارسی.....	Error! Bookmark not defined.
4-2 قلم های انگلیسی.....	Error! Bookmark not defined.
4-3 فرمول ها (روابط ریاضی).....	Error! Bookmark not defined.
4-4 فاصله های افقی و عمودی.....	Error! Bookmark not defined.
4-4-1 فاصله کلی از چهار طرف کاغذ.....	Error! Bookmark not defined.
4-4-2 فاصله خط ها.....	Error! Bookmark not defined.
4-4-3 فاصله های تفکیک کننده.....	Error! Bookmark not defined.
4-5 فواصل بین کلمات.....	Error! Bookmark not defined.
4-6 جدانویشتن کلمات بدون گذاشتن فاصله بین آنها.....	Error! Bookmark not defined.
4-7 فهرست گزارش، فهرست شکل ها و فهرست جداول.....	Error! Bookmark not defined.

8-4- سربرگ و ته‌برگ (Header and Footer).....Error! Bookmark not defined.

9-4- جداول، منحنی‌ها، شکل‌ها.....Error! Bookmark not defined.

10-4-ارجاع به جداول، شکل‌ها، روابط، مراجع و بخش‌ها.....Error! Bookmark not defined.

فصل پنجم بررسی ساختار پایان نامه‌بررسی ساختار پایان نامه.....Error! Bookmark not defined.

1-5- بررسی سرفصل‌ها.....Error! Bookmark not defined.

2-5- بررسی ساختار کلی.....Error! Bookmark not defined.

3-5- بررسی مفهومی.....Error! Bookmark not defined.

4-5- مطالعه مفهومی و جمله‌بندی.....Error! Bookmark not defined.

5-5- تنظیم بندها.....Error! Bookmark not defined.

6-5- بررسی قواعد نگارشی.....Error! Bookmark not defined.

7-5- بررسی روابط.....Error! Bookmark not defined.

8-5- بررسی شکل‌ها.....Error! Bookmark not defined.

1-8-5- بررسی کیفیت شکل و تطابق عنوان آن.....Error! Bookmark not defined.

2-8-5- بررسی تطابق روابط، برنامه و شکل.....Error! Bookmark not defined.

9-5- بررسی جداول.....Error! Bookmark not defined.

1-9-5- بررسی کیفیت جدول و تطابق عنوان آن.....Error! Bookmark not defined.

2-9-5- بررسی تطابق روابط، برنامه و جدول.....Error! Bookmark not defined.

10-5- به‌روزرسانی مراجع.....Error! Bookmark not defined.

11-5- صفحه‌بندی.....Error! Bookmark not defined.

12-5- سربرگ و ته‌برگ‌ها.....Error! Bookmark not defined.

فصل ششم جمع‌بندی و نتیجه‌گیری و پیشنهادهاجمع‌بندی و نتیجه‌گیری.....Error! Bookmark not defined.

منابع و مراجع.....Error! Bookmark not defined.

4-5- یا مطابق دستور العمل زیر :.....Error! Bookmark not defined.

پیوست‌ها.....Error! Bookmark not defined.

Abstract.....Error! Bookmark not defined.

فصل اول

مقدمه

فصل دوم

منطق بازی و اجرای آن در پایتون

منطق بازی و اجرای آن در پایتون

برنامه از فایل‌های `solver.py`, `solver.optimized.py`, `main.py`, `utils.py` تشکیل شده. همچنین یک فایل تست کیس برای حل سودوکو 9*9 به همراه آن آپلود شده است.

solver.py-1-2

این فایل شامل تابع `solve_sudoku` است. تابع `solve_sudoku` ورودی را به عنوان ورودی می‌پذیرد و وقتی معمای سودوکو حل شد، `True` را برمی‌گرداند. ابتدا این تابع با فراخوانی تابع `find_empty_cell` بررسی می‌کند که آیا هنوز هیچ سلول خالی در برد وجود دارد یا نه. اگر سلول خالی وجود ندارد، `True` برگردانده می‌شود. اگر هر سلولی خالی باشد، از عدد 1 شروع کرده و به تعداد سائز برد، مقدارهای مختلف را در آن سلول امتحان می‌کند. اگر یک مقدار معتبر برای سلول فعلی پیدا شد، آن مقدار را به سلول اختصاص داده و تابع `solve_sudoku` را به صورت بازگشتی فراخوانی می‌کند. اگر `solve_sudoku True` را برگرداند، `solve_sudoku True` برگردانده می‌شود. اگر `solve_sudoku False` را برگرداند، مقدار سلول فعلی را صفر قرار داده و به مقدار بعدی ادامه می‌دهد.

```

solver.py > ...
1  import numpy as np
2  from utils import *
3
4  #solve sudoku
5  def solve_sudoku(board):
6      #if there is no empty cell left, return true
7      if not find_empty_cell(board):
8          return True
9
10     #for empty cells try possible variables
11     row, col = find_empty_cell(board)
12     for value in range(1, len(board)+1):
13         if is_valid_move(board, row, col, value):
14             board[row][col] = value
15
16             if solve_sudoku(board):
17                 return True
18
19             board[row][col] = 0
20
21     return False
22

```

Solver_optimized.py-2-2

تابع `mrsv` با استفاده از تابع `get_valid_moves` (که در `utils` به توضیح آن می‌پردازیم)، سلول خالی با کمترین حرکت معتبر باقیمانده را پیدا می‌کند و یک تاپل از ردیف و ستون سلول انتخاب شده، برمیگرداند.

```

def mrsv(board):
    empty_cells = []
    for i in range(len(board)):
        for j in range(len(board)):
            if board[i][j] == 0:
                empty_cells.append((i, j))
    return min(empty_cells, key=lambda x: len(get_valid_moves(board, x[0], x[1])))

```

تابع `lcv` با دریافت برد و ردیف و ستون یک سلول به عنوان ورودی، لیستی از حرکت‌های معتبر برای آن سلول را برمیگرداند که بر اساس تعداد حرکت‌های معتبری که با حذف هر حرکت انجام می‌شود، مرتب شده است. این تابع از تابع `get_valid_moves` برای پیدا کردن حرکت‌های معتبر استفاده می‌کند.

```
def lcv(board, row, col):
    moves = list(get_valid_moves(board, row, col))
    return sorted(moves, key=lambda x: len(get_valid_moves(board, row, col) - set([x])))
```

تابع `forward_check` با گرفتن بورد، ردیف و ستون یک سلول و یک مقدار به عنوان ورودی، بررسی می‌کند که قرار دادن مقدار در سلول با محدودیت‌های پازل سازگار است یا خیر، با بررسی زیرشبکه، سطر و ستون برای تداخل با مقدار، اگر تداخلی یافت شود، `False` را برمیگرداند. اگر یک سلول پس از قرار دادن مقدار، هیچ حرکت معتبری باقی نمانده باشد، همچنین `False` را برمیگرداند. در غیر این صورت، `True` را برمیگرداند.

```
def forward_check(board, row, col, value):
    subgrid_row = (row // int(np.sqrt(len(board)))) * int(np.sqrt(len(board)))
    subgrid_col = (col // int(np.sqrt(len(board)))) * int(np.sqrt(len(board)))

    for i in range(subgrid_row, subgrid_row + int(np.sqrt(len(board)))):
        for j in range(subgrid_col, subgrid_col + int(np.sqrt(len(board)))):
            if board[i][j] == value:
                return False

            if board[i][j] == 0 and not get_valid_moves(board, i, j).difference({value}):
                return False

    return True
```

تابع `solve_sudoku_optimized` با استفاده از یک جستجوی عمقی بازگشتی، به دنبال حرکت‌های معتبر مختلف می‌گردد تا تا یافتن یک راه حل برسد. در ابتدا با فراخوانی تابع `find_empty_cell` (که در `utils` به توضیح آن می‌پردازیم)، بررسی می‌کند که آیا تخته قبلاً حل شده است یا خیر. اگر حل شده باشد، `True` را برمیگرداند. در غیر این صورت، یک سلول خالی برای پرکردن انتخاب می‌کند، با استف


```
def solve_sudoku_optimized(board):
    if not find_empty_cell(board):
        return True

    row, col = mrv(board)
    for value in lcv(board, row, col):
        if is_valid_move(board, row, col, value):
            board[row][col] = value

            if forward_check(board, row, col, value) and solve_sudoku_optimized(board):
                return True

            board[row][col] = 0

    return False
```

Util.py-3-2

تابع "print_board" یک تخته سودوکو $n \times n$ را به عنوان ورودی دریافت می‌کند و آن را به صورت قابل خواندن در کنسول چاپ می‌کند.

```
def print_board(grid):
    for row in grid:
        print(*row)
```

تابع "find_empty_cell" یک تخته سودوکو را به عنوان ورودی دریافت می‌کند و ردیف و ستون اولین خانه خالی (خانه با مقدار صفر) را که پیدا می‌کند، برمی‌گرداند. اگر هیچ خانه‌ی خالی وجود نداشته باشد، تابع None را برمی‌گرداند.

```

5
6
7
8 #find empty cell
9 def find_empty_cell(board):
10     for i in range(len(board)):
11         for j in range(len(board)):
12             if board[i][j] == 0:
13                 return (i, j)
14     return None
15
16
17

```

تابع "is_valid_move" یک تخته سودوکو، یک شاخص ردیف و ستون و یک مقدار را به عنوان ورودی دریافت می‌کند و اگر مقدار را بتوان در خانه‌ی مشخص شده با شاخص ردیف و ستون قرار داد بدون نقض هیچ یک از قوانین سودوکو (یعنی مقدار در همان ردیف، ستون و یا زیرشبکه‌ی $(n*1/2)*(n*1/2)$ خود قرار ندارد)، True را برمی‌گرداند. در غیر این صورت، تابع False را برمی‌گرداند.

```

#check the validation of value
def is_valid_move(board, row, col, value):
    # check if value is already in the row
    if value in board[row]:
        return False

    # check if value is already in the column
    if value in [board[i][col] for i in range(len(board))]:
        return False

    # check if value is already in the sub-grid
    subgrid_row = (row // int(np.sqrt(len(board)))) * int(np.sqrt(len(board)))
    subgrid_col = (col // int(np.sqrt(len(board)))) * int(np.sqrt(len(board)))

    for i in range(subgrid_row, subgrid_row + int(np.sqrt(len(board)))):
        for j in range(subgrid_col, subgrid_col + int(np.sqrt(len(board)))):
            if board[i][j] == value:
                return False

    return True

```

تابع "get_valid_moves" یک تخته سودوکو و یک شاخص ردیف و ستون را به عنوان ورودی دریافت می‌کند و مجموعه‌ای از تمام حرکت‌های معتبر (یعنی تمام مقادیری که می‌توانند در خانه‌ای با شاخص ردیف و ستون مشخص شده بدون نقض هیچ یک از قوانین سودوکو قرار داد)

```
#give all possible values
def get_valid_moves(board, row, col):
    moves = set(range(1, len(board)+1))

    # remove values already in the row
    moves -= set(board[row])

    # remove values already in the column
    moves -= set([board[i][col] for i in range(len(board))])

    # remove values already in the sub-grid
    subgrid_row = (row // int(np.sqrt(len(board)))) * int(np.sqrt(len(board)))
    subgrid_col = (col // int(np.sqrt(len(board)))) * int(np.sqrt(len(board)))

    for i in range(subgrid_row, subgrid_row + int(np.sqrt(len(board)))):
        for j in range(subgrid_col, subgrid_col + int(np.sqrt(len(board)))):
            moves.discard(board[i][j])

    return moves
```

main.py -4-2

در این فایل با استفاده از فانکشن‌هایی که توضیح دادیم، به اجرای سودوکو می‌پردازیم. می‌توانیم فایل تست کیس را مستقیم بخوانیم یا آن را به عنوان ورودی از کاربر بگیریم. در خط اول اندازه سودوکو، در خط دوم تعداد خانه‌های پر شده و در خط‌های بعدی مختصات و عدد آن خانه‌ها را می‌گیریم. با هردو فانکشن سولور و سولور آپتیمایز شده که پیش‌تر توضیح دادیم سودوکو را حل می‌کنیم و زمان استفاده شده برای حل آن را در کنسول نمایش می‌دهیم.

به عنوان مثال یک تست کیس از سودوکو 9×9 به عنوان ورودی به آن دادم:

خط اول 9 سایز سودوکو، خط دوم 38، تعداد خانه‌های پر شده و در 38 خط بعد مختصات و عدد خانه‌های پر شده را وارد کردیم.

```
3 8 9
solvable
2 9 6 3 4 5 8 7 1
5 8 1 7 2 6 3 4 9
7 4 3 8 9 1 2 5 6
6 5 8 9 1 3 4 2 7
4 3 2 6 8 7 9 1 5
9 1 7 2 5 4 6 8 3
3 7 4 1 6 2 5 9 8
1 2 9 5 3 8 7 6 4
8 6 5 4 7 9 1 3 2
Elapsed time: 0.05086469650268555 seconds
solvable
2 9 6 3 4 5 8 7 1
5 8 1 7 2 6 3 4 9
7 4 3 8 9 1 2 5 6
6 5 8 9 1 3 4 2 7
4 3 2 6 8 7 9 1 5
9 1 7 2 5 4 6 8 3
3 7 4 1 6 2 5 9 8
1 2 9 5 3 8 7 6 4
8 6 5 4 7 9 1 3 2
Elapsed time optimized: 0.025936365127563477 seconds
PS D:\daneshga\8\AI\Sudoku> █
```

منابع

در حل این پروژه از chatgpt استفاده کردم.