Sudoku Solver

Release 1.2.0

Will Purvis

Dec 10, 2023

CONTENTS:

CHAPTER

ONE

SUDOKU SOLVER MODULES

Here is a list of all the modules used in the Sudoku Solver package:

1.1 Main module

This script is the entry point for the Sudoku Solver package. It utilizes the backtracking algorithm with minimum remaining value (MRV) heuristics. The algorithm is from the `backtracking_mrv` module and various utility functions from the `utils` module are used to to solve sudoku puzzles. Usage: `src/main.py input.txt` where `input.txt` is the path to the file containing the sudoku puzzle to be solved.

Author: William Purvis
Created: 25/11/2023
Last updated: 10/12/2023

main.**get_user_input**()

Prompt user to select whether they want to solve the uploaded sudoku.

`Do you want to solve the uploaded sudoku? [y/n]:` is displayed to the user. If user enters `y`, `True` is returned. If user enters `n`, `False` is returned.

Returns
True if the user wants to solve sudoku, False otherwise.

Return type
bool

main.**is_valid_file**(*filename*)

Check if a given file is valid based on whether file exists and whether it is a text file (.txt).

Parameters
(str) (*filename*) –

Raises

- **FileNotFoundError** – If the file does not exist.
- **ValueError** – If the file format is not valid.

main.**main**()

Main function that handles the execution of the Sudoku solver program.

It reads the input Sudoku file (given as a .txt file via the CL),solves the Sudoku puzzle using backtracking algorithm,and displays the solved Sudoku along with the time taken for solving.

If the user chooses not to solve the Sudoku, the program exits.

Raises

- **FileNotFoundError** – If the input Sudoku file is not found.
- **ValueError** – If there is an error in the input Sudoku file.

main.**parse_arguments**()

Parse command line arguments.

Returns
args (argparse.Namespace)

Return type
Parsed command line arguments.

Raises
ValueError – If more than 1 argument is passed on CL.

1.2 Backtracking

This module contains the backtracking algorithm for solving sudoku puzzles. The backtracking algorithm works recursively by trying values 1-9 in empty cells and 'backtracking' if the value is invalid. Validity of values is checked using `validateCell()`, which checks if the value is valid for the cell according to sudoku rules.

References:

- Norvig, P. (2013). Solving Every Sudoku Puzzle
- GeeksforGeeks (2020). Sudoku | Backtracking-7

backtracking.**find_empty_cell**(*sudoku_board: list[list[int]]*) → tuple[int, int]

Find the first empty cell in `sudoku_board` and return its row & column indices.

Parameters
sudoku_board (*list[list[int]]*) – List of lists representing sudoku board

Returns
Tuple containing row & column indices of empty cell or None if no empty cells are found

Return type
tuple[int, int]

backtracking.**solve_backtrack**(*grid: list[list[int]], i: [<class 'int'>], j: [<class 'int'>]*) → list[list[int]]

Solve sudoku using backtracking algorithm (recursive implementation). The algorithm works as follows:

1) Find empty cell using findEmptyCell()
2) If there are no empty cells, the sudoku is solved and the resulting grid is returned
3) Try values 1-9
4) Validate value using validateCell()
5) Repeat steps 1-4 until sudoku is solved

Parameters

- **grid** (*list[list[int]]*) – List of list with dimensions 9x9 representing sudoku board
- **i** (*int*) – Row index of cell
- **j** (*int*) – Column index of cell

Returns
List of list with dimensions 9x9 representing solved sudoku board

Return type
list[list[int]]

backtracking.**validate_cell**(*sudoku_board: list[list[int]]*, *val: int*, *i: int*, *j: int*) → bool

Check if a value is valid for cell `[i][j]` in `sudoku_board`. Validity of sudoku is based off the following rules:

1) Each row must contain the digits 1-9 without repetition.
2) Each column must contain the digits 1-9 without repetition.
3) Each of the nine 3 x 3 sub-boxes of the grid must contain the digits 1-9 without repetition

Parameters

- **sudoku_board** (*list[list[int]]*) – List of list representing sudoku board
- **val** (*int*) – Value to be checked
- **i** (*int*) – Row index of cell
- **j** (*int*) – Column index of cell

Returns
True if value is valid according to sudoku rules, False otherwise

Return type
bool

Raises

- **TypeError** – If sudoku board input is not a list of lists
- **TypeError** – If value input is not an int
- **TypeError** – If row/column indices are not ints
- **ValueError** – If value input is not between 1-9
- **ValueError** – If row/column indices are not between 0-8

1.3 Backtracking with MRV heuristics

This module is an improved implementation of the backtracking algorithm in backtracking.py. The `validate_cell()` function is unchanged, but the `find_empty_cell()` and `solve_backtrack()` functions have been modified to include minimum remaining values (MRV) heuristics. This means that the backtracking algorithm will always choose the cell with the fewest possible values to try first (instead of the next empty cell). This is done using the `n_possible_values()` function.

backtracking_mrv.**find_empty_cell_MRV**(*sudoku_board: list[list[int]]*) → tuple[int, int]

Iterates through given Sudoku board and uses `n_possible_values()` to find the cell with the fewest possible values. Returns the row and column indices of that cell as a tuple to be used in the backtracking algorithm. This technique is called the Minimum Remaining Value (MRV) heuristic.

Parameters
sudoku_board (*list[list[int]]*) – List of lists representing sudoku board

Returns
min_cell – Row and column indices of the cell with the fewest possible values.

Return type
tuple[int, int]:

backtracking_mrv.**n_possible_values**(*sudoku_board: list[list[int]]*, *i: int*, *j: int*) → int

Returns the number of possible values for a given cell in a Sudoku board. Uses `validate_cell()`.

Parameters

- **sudoku_board** (*list[list[int]]*) – List of lists representing sudoku board
- **i** (*int*) – Row index of cell to be checked
- **j** (*int*) – Column index of cell to be checked

Returns
Number of possible values for the given cell (`sudoku_board[i][j]`).

Return type
int

backtracking_mrv.**solve_backtrack_MRV**(*sudoku_board*, *i*, *j*) → list[list[int]]

Backtracking algorithm with MRV

This function solves a Sudoku puzzle using the backtracking algorithm with Minimum Remaining Values (MRV) heuristic. It takes a Sudoku board represented as a 2D list (list of lists), along with the indices (i, j) of the current cell being considered. Returns the solved sudoku board as a 2D list of lists.

Parameters

- **sudoku_board** (*(list[list[int]])*) – List of list with dimensions 9x9 representing sudoku board
- **i** (*int*) – Row index of cell
- **j** (*int*) – Column index of cell

Returns
List of list with dimensions 9x9 representing solved sudoku board

Return type
list[list[int]]

backtracking_mrv.**validate_cell**(*sudoku_board: list[list[int]]*, *val: int*, *i: int*, *j: int*) → bool

Check if a value is valid for cell `[i][j]` in `sudoku_board`. Validity of sudoku is based off the following rules:

1) Each row must contain the digits 1-9 without repetition.
2) Each column must contain the digits 1-9 without repetition.
3) Each of the nine 3 x 3 sub-grids must contain the digits 1-9 without repetition

Parameters

- **sudoku_board** (*list[list[int]]*) – List of lists representing the sudoku board
- **val** (*int*) – Value to be checked
- **i** (*int*) – Row index of cell
- **j** (*int*) – Column index of cell

Returns
True if value is valid according to sudoku rules, **False** otherwise

Return type
bool

Raises

- **TypeError** – If sudoku board input is not a list of lists
- **TypeError** – If value input is not an int
- **TypeError** – If row/column indices are not ints
- **ValueError** – If value input is not between 1-9
- **ValueError** – If row/column indices are not between 0-8

1.4 Utilities

This module contains utility functions that are used throughout the project.
`parse_grid()` converts a text-based suduko grid into a numpy array ready for processing. `validate_board()` checks if a sudoku board is valid (*note*: a valid sudoku does not necessarily mean that the sudoku is solvable). If the board is invalid, the row and column of the invalid cells are returned. `highlight_errors()` highlights the invalid cells in red. `display_sudoku()` converts a list of lists into a text-based suduko grid for display purposes.

`utils.`**display_sudoku**(*board: list[list[int]]*) → str

Reverse of parse_grid: displays sudoku board given as a list in a readable format.

Parameters
board (*list of list*) – List of list where elements can be accessed by row [i] & column [j]

Returns

sudoku – Text-based grid where empty cells are represented by 0, each subgrid is seperated by | and each row is seperated by - and +

Example:

```
000|007|000
000|009|504
000|050|169
---+---+---
080|000|305
075|000|290
406|000|080
---+---+---
762|080|000
103|900|000
000|600|000
```

Return type
str

Raises

- **TypeError** – If input is not a list of lists of ints
- **ValueError** – If input is not 9x9 or if cells contains values outside of 0-9

utils.**highlight_errors**(*sudoku: array*, *invalid_cells: list[tuple[int, int]]*) → None

Given a 2D numpy array representing a sudoku board and a list of invalid cells for the given board, highlight the invalid cells in red.

Parameters

- **sudoku** (*np.array*) – 2D numpy array representing a sudoku board
- **invalid_cells** (*list[tuple[int, int]]*) – List of tuples containing the row and column of the invalid cells.

utils.**parse_grid**(*sudoku: str*) → array

Convert text-based suduko grid into a numpy array. This function allows for easier manipulation of the sudoku grid by accessing numbers via their row ([i]) and column ([j]) index.

Parameters
sudoku (*str*) – Text-based grid representation of Sudoku. Empty cells should be denoted with a 0, with | and - used to seperate subgrid columns and rows, respectively. The intersections of each subgrid should be marked with +.

Example:

```
000|007|000
000|009|504
000|050|169
---+---+---
080|000|305
075|000|290
406|000|080
---+---+---
762|080|000
103|900|000
000|600|000
```

Returns
sudoku_list – A numpy array where elements can be accessed by row [i] & column [j].

Return type
np.darray

Raises

- **TypeError** – If input is not a string.
- **ValueError** – If input is not an 11x11 grid, including '|', '-', and '+' characters.

utils.**validate_board**(*sudoku: array*) → bool | list[tuple[int, int]]

Given a 2D numpy array representing a sudoku board, check if the board is valid. Validity of sudoku is based off the following rules:

1) Each row must contain the digits 1-9 without repetition.
2) Each column must contain the digits 1-9 without repetition.

3) Each of the nine 3 x 3 sub-boxes of the grid must contain the digits 1-9 without repetition

Note: a valid sudoku does not necessarily mean that the sudoku is solvable.

If the board is valid, True is returned. If the board is invalid, False and the row and column of the invalid cells are returned.

Parameters

sudoku (*np.array*) – 2D numpy array representing a sudoku board

Returns

- *bool* – True if board is valid, False otherwise
- *list[tuple[int, int]]* – List of tuples containing the row and column of the invalid cells. Empty list if board is valid.

Raises

TypeError – If input is not a numpy array

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX