**Programming Project**

**Sudoku Solver with Backtracking**

1. Introduction

Sudoku is a popular number-based puzzle game. It is a logic-based game that requires you to fill in a 9x9 grid with digits so that each column, row, and 3x3 sub-grid contains all of the digits from 1 to 9 without any repetitions. The objective of the game is to fill in the grid with the correct numbers while following the rules mentioned above. The game begins with a partially filled grid and the player is required to fill in the missing numbers. There is only one correct solution to each puzzle, and the game is considered solved when the player correctly fills in all the cells on the grid. This project aims to create a solver for Sudoku using algorithm called backtracking.

2. Technical Details

A Sudoku solver with backtracking is a program that attempts to solve a given Sudoku puzzle using a backtracking algorithm. The backtracking algorithm works by trying out possible values for each empty cell in the Sudoku grid until a solution is found.

In this Python Project, a Sudoku solver with backtracking uses a recursive function that tries out values for each empty cell and backtracks if a solution cannot be found. The algorithm works as follows:

1. Find an empty cell in the Sudoku grid

2. Try out a value for the empty cell

3. Check if the value is valid according to the rules of Sudoku

4. If the value is valid, move on to the next empty cell and repeat the process from step 1

5. If the value is not valid, try out the next value for the current cell

6. If there are no valid values for the current cell, backtrack to the previous cell and try out the next value for that cell

7. If a solution is found, return it

8. If a solution cannot be found, backtrack to the previous cell and continue the search from there

This Project represents the Sudoku grid as nested list. Each row of the grid is represented by a sublist and each element of the sublist represents a cell in the row. The value of an empty cell is represented by a zero (0).

To check the validity of a value in a cell, program checks if the value satisfies the rules of Sudoku. The rules of Sudoku state that each row, column, and region (3x3 sub-grid) should contain unique values between 1 and 9. Therefore, we need to check if the value is not already present in the same row, column, or region as the current cell.

The recursive function that searches for a solution takes the Sudoku grid as input and return the solved grid as output.

3. Installation and Running Sudoku

1. Install Python 3.9

2. Download the project source code and input file

3. Open the command prompt

4. Run the file python Project.py < sudoku\_puzzle.txt

5. Get the required result

4. User Interface

The user interface of Sudoku solver with backtracking is simple and straightforward. The user can provide the input Sudoku puzzle in a specific format by storing it in a file sudoku\_puzzle.txt, and the output is printed out in the command line.

The user needs to put Sudoku puzzle in the following format:

0 0 0 2 6 0 7 0 1

6 8 0 0 7 0 0 9 0

1 9 0 0 0 4 5 0 0

8 2 0 1 0 0 0 4 0

0 0 4 6 0 2 9 0 0

0 5 0 0 0 3 0 2 8

0 0 9 3 0 0 0 7 4

0 4 0 0 5 0 0 3 6

7 0 3 0 1 8 0 0 0

Each row of the puzzle is a new line in the file, and the numbers are separated by spaces. The unknown cells are represented by 0.

Once the file is created, the user can run the Sudoku solver by calling the Python script from the command line and passing the filename as an argument:

python Project.py < sudoku\_puzzle.txt

The script reads the input file and solves the Sudoku puzzle using the backtracking algorithm. The solution is printed out in the command line in the same format as the input, with the solved cells replacing the 0 values

Overall, the user interface of Sudoku solver with backtracking is simple and intuitive, making it easy for users to input and solve Sudoku puzzles from the command line.

5. Test Cases

Here are some test examples of Sudoku Solver using Backtracking:

1. Valid puzzle: To ensure that the solver is able to correctly solve it.

5 3 0 0 7 0 0 0 0

6 0 0 1 9 5 0 0 0

0 9 8 0 0 0 0 6 0

8 0 0 0 6 0 0 0 3

4 0 0 8 0 3 0 0 1

7 0 0 0 2 0 0 0 6

0 6 0 0 0 0 2 8 0

0 0 0 4 1 9 0 0 5

0 0 0 0 8 0 0 7 9

This puzzle has a unique solution, and each row, column, and 3x3 sub-grid contains all the numbers 1 through 9 exactly once.

2. Invalid puzzle: To ensure that the solver is able to detect the puzzle is invalid and return an error message.

9 0 0 1 0 0 0 0 4

0 1 4 0 3 0 8 0 0

0 0 3 0 0 0 0 9 0

0 0 0 7 0 8 0 0 1

8 0 0 0 0 3 0 0 0

0 0 0 0 0 0 0 3 0

0 2 1 0 0 0 0 7 0

0 0 9 0 4 0 5 0 0

5 0 0 0 1 6 0 0 3

This puzzle cannot be solved, because the middle row (row 5) has no possible candidates for the value ‘1’.

3. Partially filled puzzle: Test the solver with a partially filled puzzle to ensure that it is able to complete the puzzle with the correct solution.

0 0 0 2 6 0 7 0 1

6 8 0 0 7 0 0 9 0

1 9 0 0 0 4 5 0 0

8 2 0 1 0 0 0 4 0

0 0 4 6 0 2 9 0 0

0 5 0 0 0 3 0 2 8

0 0 9 3 0 0 0 7 4

0 4 0 0 5 0 0 3 6

7 0 3 0 1 8 0 0 0

This puzzle has some initial numbers filled in, but not all. The solver will fill in the remaining cells in order to solve the puzzle.

6. Library Used

There was only one library used which is sys library for reading from standard input

7. Conclusion

In conclusion, this documentation has provided a comprehensive overview of a Sudoku solver implemented in Python using the backtracking algorithm. With the information provided in this documentation, users should be able to understand the program's functionality, limitations, and test it with various test cases. Overall this Sudoku solver Project was exciting and challenging experience. Through this project, we learned about backtracking algorithm more deeply. In the development process, we ran across a number of challenges, but we were able to resolve them through problem-solving.