

**Department of Computer Engineering**

**Course: DAA**

**Course Code: BTECCE21501**

**Mini-Project – Report**

**PHASE - I**

**Guidance By - Prof. Trupti**

**Topic: Algorithmic Solutions for Solving Sudoku Puzzles**

**By**

|  |  |  |
| --- | --- | --- |
| **Roll No** | **SRN** | **Name of Student** |
| 38 | 202200930 | Shyamal S. Patil |
| 39 | 202200931 | Shaikh M. Asad |
| 33 | 202101389 | Tanmay Gawali |

**Project Descriptive**

Project Overview: This project aims to design, implement, and analyze a Sudoku puzzle solver using various algorithms and techniques, with a focus on the Design and Analysis of Algorithms (DAA) principles. Sudoku is a popular logic-based puzzle game, and this project will involve creating a Sudoku solver software that can efficiently solve Sudoku puzzles of varying complexity levels.

**Project Objectives:**

Algorithm Implementation:

* Develop a Sudoku solver using one or more algorithms, such as backtracking, constraint propagation, or Dancing Links (Algorithm X).
* User Interface:

Create a user-friendly graphical interface for users to input Sudoku puzzles and view the solutions.

* Performance Analysis:

Analyze and compare the time and space complexity of the implemented algorithms to understand their efficiency in solving Sudoku puzzles.

* Optimization:

Investigate and implement optimization techniques to enhance the solver's performance, especially for challenging Sudoku puzzles.

* User Experience:

Ensure the user interface is intuitive and responsive, providing a seamless user experience for inputting puzzles and viewing solutions.

* Error Handling:

Implement robust error-checking mechanisms to validate user input and provide clear and informative error messages.

* Difficulty Levels:

Develop a feature that generates Sudoku puzzles with different difficulty levels (easy, medium, hard) and allows users to select the level of challenge they prefer.

**Advanced Solving Techniques (Optional):**

Consider implementing advanced techniques such as naked pairs, naked triples, hidden singles, and other Sudoku-specific strategies to improve solving efficiency.

* Documentation:

Create comprehensive documentation, including algorithm explanations, software usage instructions, and an explanation of how the software works.

* Testing:

Rigorously test the software with a wide range of Sudoku puzzles to ensure correctness, accuracy, and robustness.

**Some algorithms**

Sudoku is a popular logic-based puzzle game that can be solved using various algorithms and techniques. Here are a few common algorithms and strategies for solving Sudoku puzzles:

Brute Force Algorithm:

Generate all possible combinations for each cell, starting from the first cell.

If a conflict is detected, backtrack and try the next possible value.

Repeat this process until a solution is found or until it's determined that the puzzle has no solution.

**Backtracking Algorithm:**

Similar to brute force, but it uses backtracking to efficiently explore the search space.

Start with the first empty cell and try filling it with a number.

Move to the next empty cell and repeat. If a conflict is detected, backtrack to the previous cell and try the next number.

Continue until a solution is found or until it's determined that the puzzle has no solution.

Constraint Propagation:

Apply the rules of Sudoku (each number must appear once in each row, column, and 3x3 subgrid) to eliminate possible values for each cell.

Use techniques like naked pairs, naked triples, and hidden singles to deduce cell values.

Dancing Links Algorithm (Algorithm X):

* Convert the Sudoku puzzle into an exact cover problem and use Donald Knuth's Dancing Links algorithm to find a solution.

This is a more advanced technique and is often used in solving Sudoku puzzles efficiently.

* Project Objectives for a Design and Analysis of Algorithms (DAA) Project involving Sudoku:
* Algorithm Implementation: Implement one or more Sudoku-solving algorithms such as backtracking, constraint propagation, or Dancing Links.
* User Interface: Create a user-friendly interface that allows users to input Sudoku puzzles and view the solutions.
* Performance Analysis: Analyze the time and space complexity of the chosen algorithm(s) and compare their efficiency in solving Sudoku puzzles of varying complexity.
* Optimization: Explore ways to optimize the chosen algorithm(s) for faster solving of Sudoku puzzles.

User Experience: Ensure that the user interface provides a smooth and intuitive experience for inputting puzzles and viewing solutions.

**Project Requirements:**

1. **Functional Requirements**:

* Sudoku Solver Algorithms:

The software must implement at least one Sudoku-solving algorithm, such as backtracking, constraint propagation, or Dancing Links.

* User Interface:

Create a graphical user interface that allows users to:

Input Sudoku puzzles.

* View solutions.

Choose difficulty levels for generated puzzles.

* Performance Analysis:

The software should analyze and display the time and space complexity of the solving algorithm(s).

* Optimization:

Implement optimization techniques to enhance the solver's performance.

* Error Handling:

Detect and handle invalid input or unsolvable puzzles gracefully by providing informative error messages.

**Non-Functional Requirements:**

* Performance:

The software should be efficient and capable of solving Sudoku puzzles in a reasonable time frame, even for challenging puzzles.

* User-Friendly Interface:

The graphical user interface should be intuitive and responsive, catering to users with varying levels of experience.

* Error Tolerance:

The software should gracefully handle user errors and unsolvable puzzles without crashing or freezing.

* Cross-Platform Compatibility:

If applicable, ensure that the software is compatible with multiple operating systems (e.g., Windows, macOS, Linux).

* Scalability:

The software should be able to handle Sudoku puzzles of varying sizes (e.g., 9x9, 16x16).