

MANGALORE INSTITUTE OF TECHNOLOGY & ENGINEERING

Accredited by NAAC with A+ Grade, An ISO 9001: 2015 Certified Institution

(A Unit of Rajalaxmi Education Trust®, Mangalore - 575001)

Affiliated to V.T.U., Belagavi, Approved by AICTE, New Delhi



TECHNICAL TRAINING PROJECT

Group-cp044

TOPIC : SUDOKU PUZZLE

GROUP MEMBERS:

Shravya :4MT21CS188

Shreya : 4MT21CS152

Shantika : 4MT21CS139

Sparsitha:4MT21CS161

Yamuna :4MT21CS188

Vikas : 4MT21CS180

Problem statement :

Sudoku solver Implement a function to solve Sudoku puzzles programmatically apply Sudoku solver algorithm and display the solve Sudoku grid to the user.

Introduction to Sudoku.:

Sudoku is a puzzle that has enjoyed worldwide popularity since 2005. To solve a Sudoku puzzle, one needs to use a combination of logic and trial-and-error. More maths is involved behind the scenes: combinatorics used in counting valid Sudoku grids, group theory used to describe ideas of when two grids are equivalent, and computational complexity with regards to solving Sudoku. Although sudoku-type patterns had been used earlier in agricultural design, their first appearance in puzzle form was in 1979 in a New York-based puzzle magazine, which called them Number Place puzzles. They next appeared in 1984 in a magazine in Japan, where they acquire the name Sudoku (abbreviated from suuji wa dokushin ni kagiru, meaning “the numbers

must remain single”). In spite of the puzzle’s popularity in Japan, the worldwide sudoku explosion had to wait another 20 years.

IDEA OF SUDOKU:

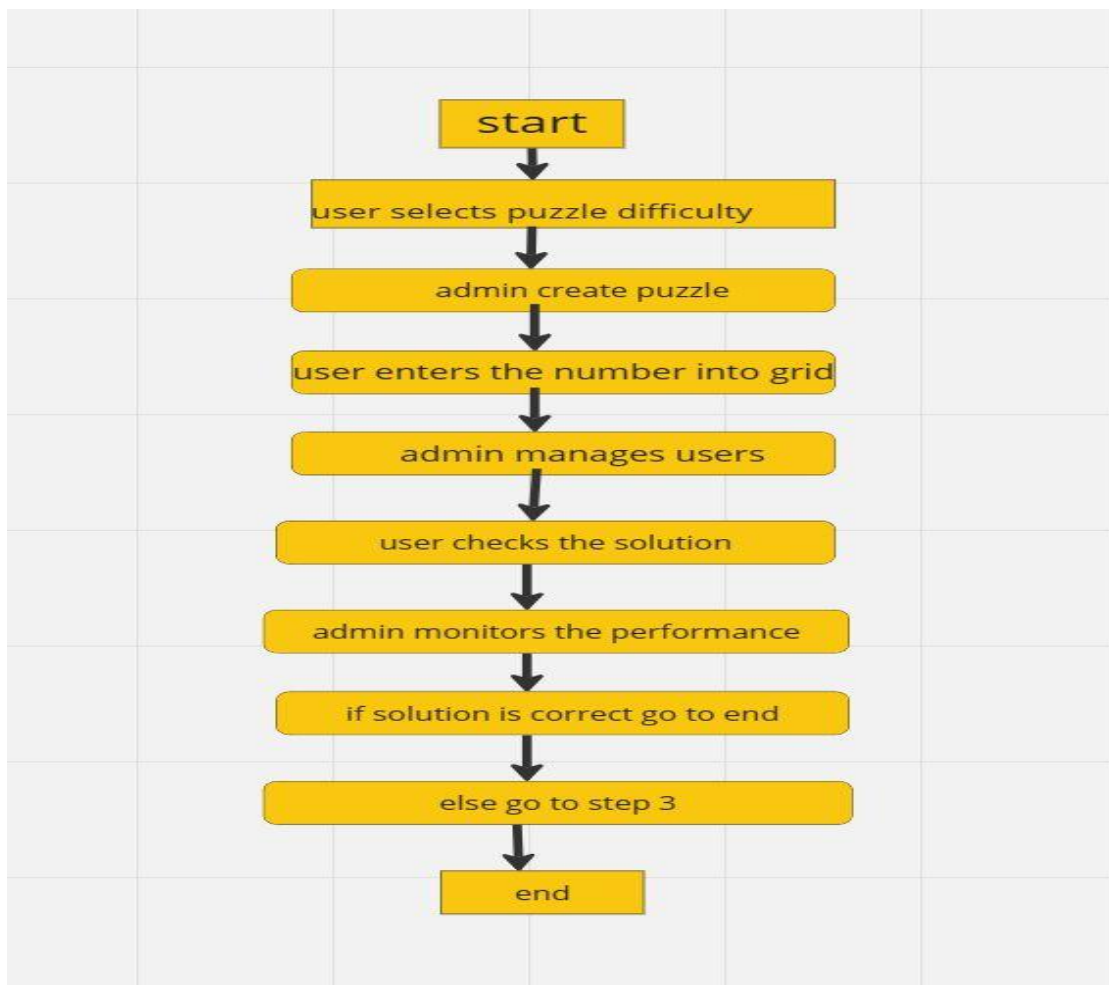
1. **Grid Structure:** A standard Sudoku grid consists of 81 cells arranged in a 9x9 grid. This grid is divided into nine 3x3 regions, each containing nine cells. These regions are typically shaded or outlined to distinguish them from the rest of the grid.
2. **Initial Numbers:** A Sudoku puzzle begins with some of the cells already filled with numbers. These numbers are called "given numbers" or "clues." The placement of these initial numbers is such that the puzzle has a unique solution.
3. **Number Placement:** To solve a Sudoku puzzle, you must fill in the empty cells with numbers from 1 to 9. The placement of numbers must adhere to the following rules:
 - Each row must contain all the numbers from 1 to 9, with no repetitions.
 - Each column must also contain all the numbers from 1 to 9, with no repetitions.
 - Each 3x3 region must contain all the numbers from 1 to 9, with no repetitions.
4. **Solving Techniques:** Sudoku puzzles can vary in difficulty. Solving techniques range from simple logical deductions to more advanced strategies. Some common solving techniques include:
 - **Scanning:** Identifying the only possible candidate for a cell based on the numbers already present in the row, column, and region.
 - **Crosshatching:** Focusing on rows and columns with fewer missing numbers to narrow down possibilities.
 - **Penciling In:** Keeping track of possible candidates for each cell.
 - **Naked Pairs, Triples, and Quads:** Identifying sets of two, three, or four numbers that can only appear in certain cells within a region.
 - **X-Wing, Swordfish, and other advanced techniques:** These involve patterns that can eliminate candidates in specific cells.
5. **Solvability:** A well-constructed Sudoku puzzle should have a unique solution that can be reached through logic and deduction alone. Some puzzles are designed to be easier, while others are more challenging and require more advanced solving techniques.
6. **Variations:** Sudoku has spawned numerous variations and hybrid puzzles, including:
 - **Samurai Sudoku:** Multiple overlapping Sudoku grids in a single puzzle.
 - **Killer Sudoku:** Involves not only number placement but also arithmetic operations.
 - **Sudoku-X:** The diagonal cells in addition to the rows, columns, and regions must contain unique numbers.

Sudoku Solver																	
Puzzle									Solution								
8	5	6		1	4	7	3		8	5	6	2	1	4	7	3	9
	9								1	9	3	5	7	6	8	4	2
2	4					1	6		2	4	7	9	8	3	1	5	6
	6	2		5	9	3			4	6	2	7	5	9	3	8	1
	3	1	8		2	4	5		9	3	1	8	6	2	4	5	7
		5	3	4			9	2	7	8	5	3	4	1	9	2	6
	2	4					7	3	6	2	4	1	9	8	5	7	3
							1		3	7	9	4	2	5	6	1	8
1	8	6	3		2	9	4		5	1	8	6	3	7	2	9	4

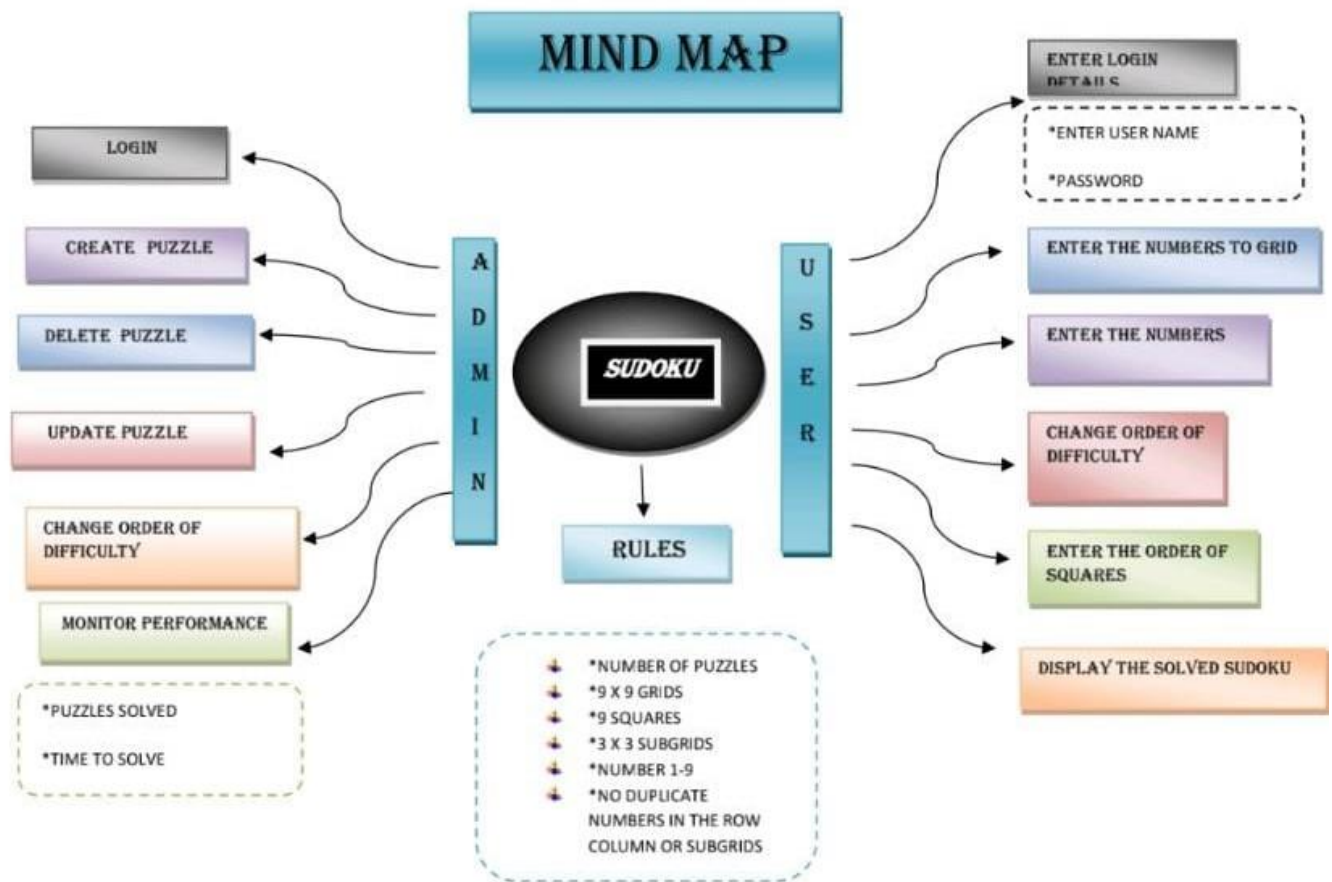
ABOUT OUR PROJECT:

Our project is typically is a Sudoku game .A game which you all are familiar with, Here we have a user and admin. Sudoku game is all about filling the numbers with some certain rules and technique.Some permissions are only for the the admin and some are unique for the users. The permissions which Users have is to: Login To enter the order of puzzles which they have to play To set the time they wish complete their puzzle in Change the order of difficulty To delete the puzzle or to enter the numbers The permissions which Admin have is to: Login Maintain ,monitoring and moderation Bux fixes and updates Game configuration Custom and support Security.

FLOWCHART:



MINDMAP:



CODE :

```

#include <stdio.h>
#include <stdbool.h>
#include <string.h>
// Constants
#define GRID_SIZE 9
#define EMPTY_CELL 0
// Function prototypes
bool solveSudoku(int grid[GRID_SIZE][GRID_SIZE]);
bool isValidMove(int grid[GRID_SIZE][GRID_SIZE], int row, int col, int num);
void printGrid(int grid[GRID_SIZE][GRID_SIZE]);
bool loginUser();
bool isAdminLoggedIn = false;
int main() {
int sudokuGrid[GRID_SIZE][GRID_SIZE] = {
{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}
};
int choice;
printf("Welcome to Sudoku Game\n");
do {
printf("Main Menu\n");
printf("1. Login\n");
printf("2. Play Sudoku\n");
printf("3. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 1:
if (loginUser()) {
printf("Login successful.\n");
} else {
printf("Login failed.\n");
}
break;
case 2:
if (isAdminLoggedIn) {
// Play Sudoku
printf("Sudoku Puzzle:\n");
printGrid(sudokuGrid);
printf("Solving Sudoku...\n");
if (solveSudoku(sudokuGrid)) {
printf("Solved Sudoku:\n");
printGrid(sudokuGrid);
} else {
printf("No solution found.\n");
}
} else {
printf("You must log in first.\n");
}
break;
case 3:
printf("Exiting...\n");
break;
default:
printf("Invalid choice. Please try again.\n");
break;
}
} while (choice != 3);
return 0;
}

// Sudoku Solver Functions
bool solveSudoku(int grid[GRID_SIZE][GRID_SIZE]) {
int row, col;
// Find an empty cell
bool isEmpty = false;
for (row = 0; row < GRID_SIZE; row++) {
for (col = 0; col < GRID_SIZE; col++) {
if (grid[row][col] == EMPTY_CELL) {

```

```

isEmpty = true;
break;
}
}
if (isEmpty)
break;
}
// If no empty cell is found, the puzzle is solved
if (!isEmpty)
return true;
// Try placing numbers 1-9 in the empty cell
for (int num = 1; num <= 9; num++) {
if (isValidMove(grid, row, col, num)) {
grid[row][col] = num;
if (solveSudoku(grid))
return true;
// If the current configuration does not lead to a solution, backtrack
grid[row][col] = EMPTY_CELL;
}
}
return false;
}
bool isValidMove(int grid[GRID_SIZE][GRID_SIZE], int row, int col, int num) {
// Check if 'num' is not already present in the current row, column, and 3x3 grid
for (int i = 0; i < GRID_SIZE; i++) {
if (grid[row][i] == num || grid[i][col] == num || grid[row - row % 3 + i / 3][col - col %
3 + i
% 3] == num) {
return false;
}
}
return true;
}
void printGrid(int grid[GRID_SIZE][GRID_SIZE]) {
for (int i = 0; i < GRID_SIZE; i++) {
for (int j = 0; j < GRID_SIZE; j++) {
printf("%2d ", grid[i][j]);
}
printf("\n");
}
}
// Login Function
bool loginUser() {
char username[20];
char password[20];
printf("Enter username: ");
scanf("%s", username);
printf("Enter password: ");
scanf("%s", password);
// Simulated username and password check (Replace with secure authentication)
if (strcmp(username, "user") == 0 && strcmp(password, "password") == 0) {
isAdminLoggedIn = true;
return true;
} else {

```

```

isAdminLoggedIn = false;
return false;
}
} #include <stdio.h>
#include <stdbool.h>
#include <string.h>
// Constants
#define GRID_SIZE 9
#define EMPTY_CELL 0
// Function prototypes
bool solveSudoku(int grid[GRID_SIZE][GRID_SIZE]);
bool isValidMove(int grid[GRID_SIZE][GRID_SIZE], int row, int col, int num);
void printGrid(int grid[GRID_SIZE][GRID_SIZE]);
bool loginUser();
bool isAdminLoggedIn = false;
int main() {
int sudokuGrid[GRID_SIZE][GRID_SIZE] = {
{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}
};
int choice;
printf("Welcome to Sudoku Game\n");
do {
printf("Main Menu\n");
printf("1. Login\n");
printf("2. Play Sudoku\n");
printf("3. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 1:
if (loginUser()) {
printf("Login successful.\n");
} else {
printf("Login failed.\n");
}
break;
case 2:
if (isAdminLoggedIn) {
// Play Sudoku
printf("Sudoku Puzzle:\n");
printGrid(sudokuGrid);
printf("Solving Sudoku...\n");
if (solveSudoku(sudokuGrid)) {
printf("Solved Sudoku:\n");
printGrid(sudokuGrid);
} else {

```

```

printf("No solution found.\n");
}
} else {
printf("You must log in first.\n");
}
break;
case 3:
printf("Exiting...\n");
break;
default:
printf("Invalid choice. Please try again.\n");
break;
}
} while (choice != 3);
return 0;
}

// Sudoku Solver Functions
bool solveSudoku(int grid[GRID_SIZE][GRID_SIZE]) {
int row, col;
// Find an empty cell
bool isEmpty = false;
for (row = 0; row < GRID_SIZE; row++) {
for (col = 0; col < GRID_SIZE; col++) {
if (grid[row][col] == EMPTY_CELL) {
isEmpty = true;
break;
}
}
}
if (isEmpty)
break;
}
// If no empty cell is found, the puzzle is solved
if (!isEmpty)
return true;
// Try placing numbers 1-9 in the empty cell
for (int num = 1; num <= 9; num++) {
if (isValidMove(grid, row, col, num)) {
grid[row][col] = num;
if (solveSudoku(grid))
return true;
// If the current configuration does not lead to a solution, backtrack
grid[row][col] = EMPTY_CELL;
}
}
return false;
}

bool isValidMove(int grid[GRID_SIZE][GRID_SIZE], int row, int col, int num) {
// Check if 'num' is not already present in the current row, column, and 3x3 grid
for (int i = 0; i < GRID_SIZE; i++) {
if (grid[row][i] == num || grid[i][col] == num || grid[row - row % 3 + i / 3][col - col %
3 + i
% 3] == num) {
return false;
}
}
}

```



```

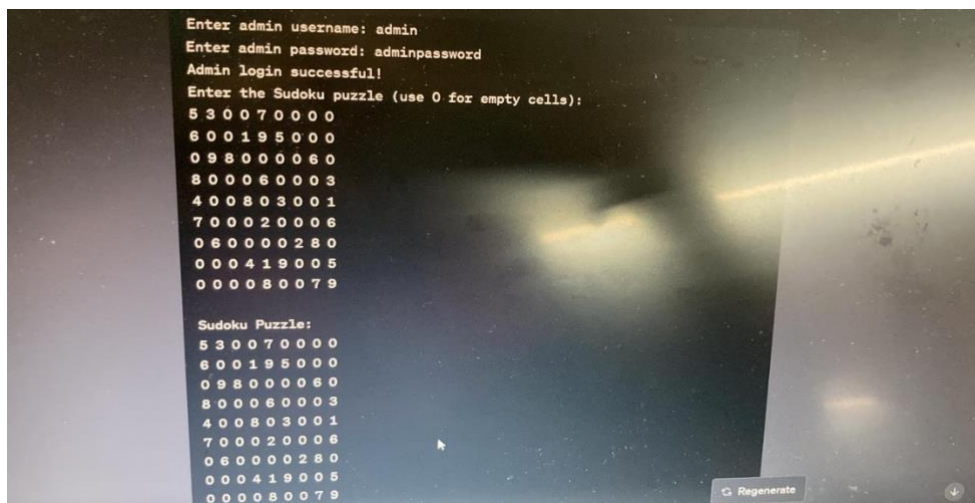
}
return true;
}

void printGrid(int grid[GRID_SIZE][GRID_SIZE]) {
for (int i = 0; i < GRID_SIZE; i++) {
for (int j = 0; j < GRID_SIZE; j++) {
printf("%2d ", grid[i][j]);
}
printf("\n");
}
}

// Login Function
bool loginUser() {
char username[20];
char password[20];
printf("Enter username: ");
scanf("%s", username);
printf("Enter password: ");
scanf("%s", password);
// Simulated username and password check (Replace with secure authentication)
if (strcmp(username, "user") == 0 && strcmp(password, "password") == 0) {
isAdminLoggedIn = true;
return true;
} else {
isAdminLoggedIn = false;
return false;
}
}
}

```

Output:

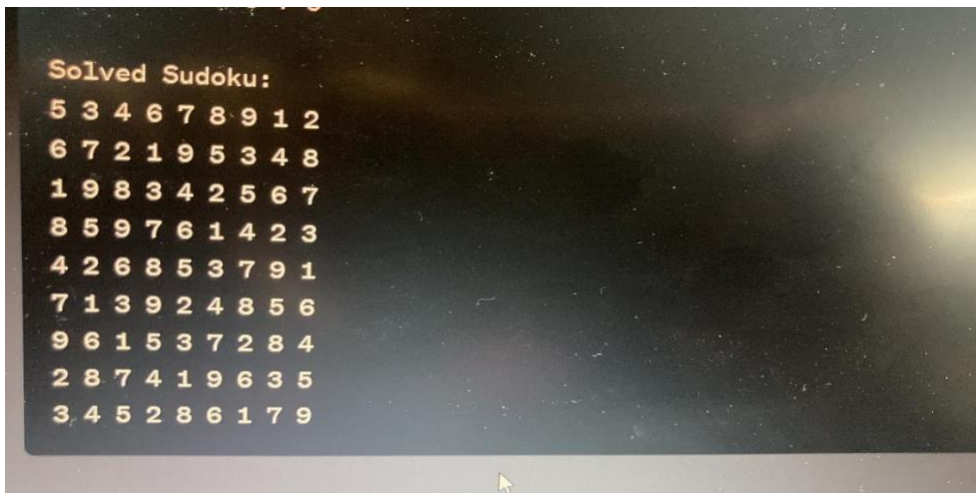


```

Enter admin username: admin
Enter admin password: adminpassword
Admin login successful!
Enter the Sudoku puzzle (use 0 for empty cells):
5 3 0 0 7 0 0 0 0
6 0 0 1 9 5 0 0 0
0 9 8 0 0 0 6 0
8 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

Sudoku Puzzle:
5 3 0 0 7 0 0 0 0
6 0 0 1 9 5 0 0 0
0 9 8 0 0 0 6 0
8 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

```



In conclusion:

Sudoku is a challenging and engaging puzzle game that has captivated the minds of millions worldwide. It's a game of logic, patience, and problem-solving, where players must fill a grid with numbers while adhering to strict rules. Sudoku offers a mental workout that can sharpen cognitive skills such as deductive reasoning and pattern recognition. Sudoku remains a beloved puzzle game that has stood the test of time, offering a unique blend of entertainment, mental stimulation, and educational value. Whether played casually or pursued as a serious endeavor, Sudoku continues to captivate minds and bring joy to countless individuals around the world.

REFERENCES:

You can find various tutorials, articles, and example code online to help you implement a Sudoku solver in C. Here are some resources:

- Sudoku Solver Tutorial by Norvig: [Link](#) (Python-based, but the logic can be adapted to C)
- GitHub Repositories with C Sudoku Solvers: Search for open-source Sudoku solver implementations on GitHub.
- C Programming Books: Consider referring to C programming books for algorithmic and data structure concepts.

WORK DISTRIBUTION:

> MINDMAP:

1.SHRAVYA 2.SHREYA >

CODE CREATION:

1.SPASHITHA 2.SHANTHIKA 3.YAMUNA

REPORT:

1.SPASHITHA 2.SHANTHIKA 3.YAMUNA 4.SHRAVYA 5.SHREYA 6.VIKAS