

SIX WEEKS SUMMER TRAINING REPORT

ON

Sudoku solver game

SUBMITTED BY

GYANENDRA NATH SHUKLA, 12113311

BACHELOR OF TECHNOLOGY

In

“COMPUTER SCIENCE AND ENGINEERING”

Under the guidance of

Mr. Vivekanand Vivek

Tutor at, Programming Pathshala

School of Computer Science & Engineering

Lovely Professional University

Phagwara, Punjab (India)

( June-July, 2023 )

Annexure-III

DECLARATION

I hereby declare that I have completed my six weeks summer training at Programming pathshala from 01 June, 2023 to 22 July, 2023 under the guidance of Mr. Vivekanand Vivek. I have declare that I have worked with full dedication during these six weeks of training and my learning outcomes fulfil the requirements of training for the award of degree of B.Tech in Computer Science and Engineering, Lovely Professional University, Phagwara.

Gyanendra

--------------------------------------------------

(Signature of student)

Name of Student: Gyanendra Nath Shukla

Registration No.: 12113311

20/08/2023

Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to everyone who has supported me throughout the process of completing my course on data structures and algorithms, as well as developing my sudoku solver game using the backtracking algorithm.

First and foremost, I am deeply thankful to my instructor, whose guidance and expertise have been invaluable. Their comprehensive explanations and real-world examples helped me grasp the fundamental concepts of data structures and algorithms, enabling me to understand their significance in problem-solving.

I would also like to extend my gratitude to my classmates, who were always willing to collaborate and share their knowledge. The discussions and brainstorming sessions we had were instrumental in enhancing my understanding and refining my implementation of the backtracking algorithm.

Additionally, I am grateful to the online resources and forums that provided me with valuable insights and solutions to overcome challenges encountered during the development of the sudoku solver game. The vast array of tutorials and code samples available online were instrumental in expanding my knowledge and enabling me to incorporate innovative features into my project.

I am indebted to my family and friends for their unwavering support and encouragement throughout this journey. Their belief in my abilities and their continuous motivation played a crucial role in keeping me focused and determined to complete the course and develop a functional sudoku solver.

Lastly, I would like to express my appreciation to the creators and contributors of the open-source libraries and frameworks that I utilized during the project. Their dedication to developing robust and efficient tools greatly facilitated the implementation of various functionalities within the game.

In conclusion, I am deeply grateful to everyone who played a part in my journey of learning data structures and algorithms, and in the development of my sudoku solver game. Your support, guidance, and inspiration have been invaluable, and I am truly thankful for the opportunity to explore this fascinating field and create a meaningful project.

SUMMER TRAINING CERTIFICATE



**CONTENTS**

[INTRODUCTION 6](#_Toc140576711)

[TECHNOLOGY LEARNT 8](#_Toc140576712)

[REASON OF CHOOSING DSA 9](#_Toc140576713)

[PROFILE OF THE PROBLEM 10](#_Toc140576714)

[EXISTING SYSTEM 11](#_Toc140576715)

[Existing System for DSA: 11](#_Toc140576716)

[Existing System for Sudoku solver Game: 11](#_Toc140576717)

[PROBLEMS 12](#_Toc140576718)

[Problem Analysis: 12](#_Toc140576719)

[Product Definition: 12](#_Toc140576720)

[Feasibility Analysis: 12](#_Toc140576721)

[SOFTWARE REQUIREMENT ANALYSIS 13](#_Toc140576722)

[DSA Requirement Analysis: 13](#_Toc140576723)

[Sudoku Solver Requirement Analysis: 13](#_Toc140576724)

[IMPLEMENTATION 14](#_Toc140576725)

[LEARNING OUTCOMES 20](#_Toc140576726)

[Data Structures and Algorithms (DSA): 20](#_Toc140576727)

[Sudoku solver Game: 20](#_Toc140576728)

[GANTT CHART 21](#_Toc140576729)

[PROJECT LEGACY 22](#_Toc140576730)

[**DSA Legacy:** 22](#_Toc140576731)

[**Sudoku Solver Game Legacy:** 23](#_Toc140576732)

[BIBLIOGRAPHY 25](#_Toc140576733)

# **INTRODUCTION**

Data Structures and Algorithms (DSA) constitute the fundamental building blocks of computer science and programming, playing a pivotal role in problem-solving, optimization, and efficient computation. This foundational knowledge forms the bedrock of any skilled programmer's toolkit, enabling them to tackle a wide range of challenges across diverse domains.Data structures encompass the art of organizing and managing data in a coherent manner within a computer's memory. By understanding how to structure and manipulate data efficiently, programmers can unlock the potential to solve complex problems with elegance and speed. The landscape of data structures includes arrays, linked lists, stacks, queues, trees, graphs, and hash tables. Each structure has distinct properties that lend themselves to specific scenarios, and proficiency in selecting the right structure is crucial for crafting effective solutions.Algorithms, in contrast, are the recipes that guide computers through solving problems step by step. These recipes dictate the sequence of operations needed to manipulate the data stored within various data structures. Within DSA, you'll encounter an array of algorithmic techniques such as searching, sorting, recursion, dynamic programming, and graph traversal. Each technique equips you with a unique strategy for dissecting problems and deriving efficient solutions.Central to DSA is the notion of efficiency, quantified by time complexity and space complexity. Time complexity gauges the algorithm's runtime, while space complexity evaluates its memory requirements. Aspiring programmers delve into these complexities to discern how their choices impact an algorithm's performance and memory consumption. The proficiency to strike the right balance between these factors empowers programmers to craft solutions tailored to the demands of specific tasks.For software engineers, programmers, and computer scientists, mastery of DSA is indispensable. Armed with this knowledge, they can engineer solutions that not only work but do so swiftly and resource-efficiently. Whether developing software applications, analyzing vast datasets, building machine learning models, or designing cutting-edge AI systems, DSA provides the framework to overcome computational challenges with finesse.Moreover, DSA nurtures a problem-solving mindset. It encourages individuals to dissect intricate problems into manageable components, design logical and efficient solutions, and refine those solutions for optimal performance. This mindset is a precious skill, not only in the realm of computer science but also in other spheres that require structured critical thinking and creative solution crafting.

In essence, Data Structures and Algorithms transcend the boundaries of code. They encapsulate the principles that empower programmers to tackle the unknown, optimize the familiar, and create solutions that harness the full potential of computational systems. In the ever-evolving landscape of technology, DSA serves as a beacon guiding the way towards elegant, efficient, and impactful problem-solving.

# **TECHNOLOGY LEARNT**

In the course on Data Structures and Algorithms (DSA), I have acquired a comprehensive understanding of various fundamental concepts. Firstly, I learned about arrays, which are data structures used to store a fixed-size sequence of elements of the same type. Arrays allow for efficient access and manipulation of elements using index values.

Next, I delved into searching techniques, exploring algorithms such as linear search and binary search. These methods enable the efficient retrieval of specific elements from arrays or lists by comparing them with a target value.

Moving on, I studied linked lists, which consist of nodes connected in a sequence. Each node holds data and a reference to the next node, allowing for flexible insertion and deletion of elements. Understanding the traversal, insertion, and deletion operations in linked lists has proven vital in manipulating this dynamic data structure.

Additionally, I delved into vector which are dynamic arrays that can have a variable size. This offers flexibility in managing collections of data efficiently, providing methods for adding and removing elements.

The course also covered stacks, a Last-In-First-Out (LIFO) data structure, where elements are inserted and removed from a single end. I learned about the push operation for adding elements and the pop operation for removing elements, following the LIFO principle.

Furthermore, I explored queues, which are First-In-First-Out (FIFO) data structures. Elements are added at one end and removed from the other end, adhering to the FIFO principle. Enqueue and dequeue operations enable the efficient management of elements in queues.

Another concept I learned about is deques, or double-ended queues. These data structures allow for insertion and removal of elements from both ends. By combining the functionalities of stacks and queues, deques provide flexibility in manipulating elements in various scenarios.

Finally, recursion, a powerful programming technique, was introduced. I gained insight into designing and implementing recursive functions, which solve problems by breaking them down into smaller subproblems and calling the function itself. I grasped the importance of defining base cases and recursive cases to ensure termination and correctness in recursive algorithms.

By studying and understanding these data structures and algorithms, I have developed a solid foundation in organizing and manipulating data efficiently. These skills are crucial for designing optimal algorithms and solving complex programming problems.

# **REASON OF CHOOSING DSA**

There are several compelling reasons for choosing to study Data Structures and Algorithms (DSA). Firstly, DSA enhances your problem-solving skills by teaching you how to analyze problems, break them down into smaller components, and devise efficient algorithms to solve them. This skill is invaluable across various domains and industries, as it enables you to tackle complex challenges effectively.

Secondly, understanding DSA allows you to write more efficient and optimized code. By selecting the appropriate data structures and algorithms for specific problems, you can improve program performance, reduce memory usage, and enhance overall system efficiency. This not only benefits your own projects but also contributes to the development of robust and scalable software systems.

DSA serves as a solid foundation for further learning in computer science and related fields. It provides you with fundamental knowledge that is transferable and applicable to advanced topics such as machine learning, artificial intelligence, database management, and more. The understanding of DSA concepts forms the basis for tackling complex algorithms and data-intensive applications.

In addition, DSA is highly relevant to the software engineering industry. Many companies specifically evaluate candidates' DSA skills during technical interviews and consider them a core competency. Having a strong command of DSA concepts increases your chances of securing competitive job opportunities and excelling in the field.

Studying DSA cultivates algorithmic thinking, a crucial skill for approaching problems systematically and developing logical solutions. This mode of thinking not only benefits programming but also enhances critical thinking and problem-solving abilities in various aspects of life.

Moreover, DSA enables you to optimize your code and tune performance. By understanding how different data structures and algorithms impact code execution, you can make informed decisions to optimize your code, reduce runtime, and improve scalability. This knowledge is essential for developing efficient and high-performance software applications.

Finally, studying DSA exposes you to a wide range of problem-solving techniques and strategies. As you tackle different types of problems, you gain a deeper understanding of various problem domains and develop the ability to apply your knowledge to real-world scenarios effectively.

In conclusion, choosing to study DSA offers numerous benefits, including improved problem-solving skills, efficient programming practices, a solid foundation for further learning, increased job prospects, enhanced algorithmic thinking abilities, code optimization skills, and a deeper comprehension of problem domains. These advantages make DSA an essential component of a well-rounded education in computer science and software engineering.

# **PROFILE OF THE PROBLEM**

Understanding the profile of a problem is essential for effectively solving it using Data Structures and Algorithms (DSA). The profile encompasses various aspects that provide insights into the problem's characteristics and requirements. By profiling the problem, we gain a better understanding of its nature and can choose appropriate DSA techniques to address it.

**Problem Description**: A clear and concise description of the problem is crucial. It defines the specific task or objective that needs to be accomplished. For example, the problem could involve searching for a specific element in a dataset, sorting a list of items in a specific order, or finding the shortest path between two points in a graph.

**Input Size and Constraints**: The size of the input and any limitations or constraints associated with it are vital considerations. It helps determine the scalability and efficiency requirements of the solution. For instance, if the input is large, we need to design algorithms and select data structures that can handle the data efficiently.

**Time and Space Complexity**: Understanding the desired performance characteristics of the solution is essential. It involves analyzing the expected time and space complexities, which provide insights into the efficiency requirements. Time complexity refers to how the running time of the algorithm scales with the input size, while space complexity relates to the memory requirements.

**Available Data**: Identifying the type and structure of the data available for the problem is crucial. It determines the choice of appropriate data structures to store and manipulate the data effectively. For instance, if the data is best represented in a graph structure, algorithms for graph traversal and manipulation can be applied.

**Required Output**: Clearly defining the expected output or desired outcome of the problem is necessary. It helps determine the format and structure of the solution. The output could be a single value, a sorted list, a specific arrangement of elements, or any other required form.

**Domain-specific Considerations**: Some problems have specific requirements or considerations based on the domain they belong to. For example, in financial calculations, precision and accuracy may be crucial, while in image processing, efficient algorithms for pixel manipulation may be required. Understanding these domain-specific aspects aids in selecting appropriate DSA techniques.

By thoroughly profiling the problem, we can better understand its intricacies, constraints, and requirements. This understanding enables us to choose the most suitable data structures, algorithms, and optimization techniques to design efficient and effective solutions. Profiling the problem helps in devising a clear strategy and approach, leading to better problem-solving outcomes using DSA.

# **EXISTING SYSTEM**

## Existing System for DSA:

In the context of Data Structures and Algorithms (DSA), the existing system refers to the current state of knowledge and implementation of DSA concepts and techniques. This includes the collection of established data structures, algorithms, and best practices that are widely used and recognized in the field.

The existing system for DSA comprises a range of data structures such as arrays, linked lists, stacks, queues, trees, graphs, and hash tables. Each data structure has its own advantages, limitations, and specific use cases. Additionally, there are various algorithms available for tasks like searching, sorting, graph traversal, dynamic programming, and more. These algorithms provide efficient solutions to common computational problems.

The existing system also includes established methodologies for analyzing the time and space complexity of algorithms, allowing developers to evaluate their efficiency and make informed choices. Best practices and design patterns guide the implementation and optimization of DSA solutions, ensuring code quality, maintainability, and scalability.

## Existing System for sudoku solver Game:

The existing system for a Sudoku solver refers to the established algorithms, techniques, and implementations used to solve Sudoku puzzles. Sudoku is a number placement puzzle played on a 9x9 grid, divided into nine 3x3 boxes. The goal is to fill the grid with digits from 1 to 9 such that each row, each column, and each of the nine 3x3 boxes contains all the digits without repetition.

The existing system for the sudoku solver game includes rules for gameplay, such as determining valid moves, detecting weather you have chosen correct value or not. Additionally, there are various existing algorithms and strategies that players employ to make optimal moves in order to solve the puzzle.

Strategies like There are machine learning algorithms that can be trained to solve Sudoku puzzles based on examples. Techniques like CNNs can be used to predict digits for each cell in a Sudoku grid.

Alongside the strategies, there are also existing implementations of the sudoku solver game available in various programming languages, utilizing different data structures and algorithms to represent the game state, handle player moves, and determine the outcome.

In summary, the existing system for DSA comprises established data structures, algorithms, and best practices, while the existing system for the sudoku solver game includes the rules, strategies, and implementations associated with the game. Both systems serve as foundations for further development, optimization, and exploration in their respective domains.

# 

# PROBLEMS

## Problem Analysis:

* Identify challenges and limitations in existing DSA solutions and sudoku solver game implementations.
* Understand requirements, pain points, and scalability issues.
* Analyze algorithms, data structures, and user expectations.

## Product Definition:

* Develop efficient DSA tools with optimized data structures and algorithms.
* Create a user-friendly sudoku solver game with intelligent gameplay options.
* Offer customization, multiplayer functionality, and an engaging user experience.

## Feasibility Analysis:

* Assess technical feasibility, resource availability, and financial constraints.
* Evaluate market demand, competition, and revenue potential.
* Consider the availability of skilled developers and required infrastructure.
* By conducting problem analysis, defining the product, and assessing feasibility, we can gain insights into the challenges, objectives, and practicality of implementing solutions for DSA and the sudoku solver game.

# **SOFTWARE REQUIREMENT ANALYSIS**

Software requirement analysis for DSA and Tic Tac Toe involves understanding the specific needs and objectives of each system. Here are the key aspects to consider for requirement analysis:

## DSA Requirement Analysis:

* Identify the target audience and their requirements for using DSA tools or libraries.
* Determine the specific functionalities and operations needed, such as searching, sorting, or graph algorithms.
* Understand the desired performance criteria, including time and space complexity requirements.
* Consider the programming languages, platforms, and environments that the DSA solution should support.
* Analyse the documentation and educational resources required to facilitate learning and understanding of DSA concepts.

## Sudoku solver Requirement Analysis:

* Determine the target platform(s) for the sudoku solver game, such as mobile, web, or desktop.
* Define the gameplay requirements, including the rules, user interactions, and graphical interface.
* Identify the desired features, such as single-player mode with AI, multiplayer mode, or customizable settings.
* Understand the scalability requirements, such as supporting different grid sizes or game variations.
* Consider any additional functionalities, such as a leaderboard, game statistics, or social media integration.

For both DSA and sudoku solver, it is important to involve stakeholders, including end-users and developers, in the requirement analysis process. This helps ensure that the identified requirements align with their expectations and objectives. The requirements should be documented in a clear and structured manner, capturing the functional and non-functional aspects of the systems.

# IMPLEMENTATION

Code:

#include <iostream>

#include <algorithm>

#include <vector>

#include <array>

#include <cstdlib>

#include <ctime>

using namespace std;

#define DIM 9

#define BLANK 0

#define SPACE " "

#define LINE "|"

#define NEW\_ROW "-------------------------------------"

#define GRID\_FULL std::make\_pair(9, 9)

// Prints the Soduko grid

void print\_grid(int grid[DIM][DIM])

{

    for (int i = 0; i < DIM; i++)

    {

        cout << SPACE << SPACE << SPACE << SPACE << endl;

        cout << NEW\_ROW << endl;

        for (int j = 0; j < DIM; j++)

        {

            cout << SPACE;

            if (BLANK == grid[i][j])

            {

                cout << SPACE;

            }

            else

            {

                cout << grid[i][j];

            }

            cout << SPACE;

            cout << LINE;

        }

    }

    cout << endl << NEW\_ROW << endl << endl;;

}

// Returns a boolean which indicates whether any assigned entry

// in the specified row matches the given number.

bool used\_in\_row(int grid[DIM][DIM], int row, int num)

{

    for (int col = 0; col < DIM; col++)

        if (grid[row][col] == num)

        {

            return true;

        }

    return false;

}

// Returns a boolean which indicates whether any assigned entry

// in the specified column matches the given number.

bool used\_in\_col(int grid[DIM][DIM], int col, int num)

{

    for (int row = 0; row < DIM; row++)

        if (grid[row][col] == num)

        {

            return true;

        }

    return false;

}

bool used\_in\_box(int grid[DIM][DIM], int box\_start\_rpw, int box\_start\_col, int num)

{

    for (int row = 0; row < 3; row++)

        for (int col = 0; col < 3; col++)

            if (grid[row + box\_start\_rpw][col + box\_start\_col] == num)

            {

                return true;

            }

    return false;

}

bool is\_safe(int grid[DIM][DIM], int row, int col, int num)

{

    // Check if 'num' is not already placed in current row,

    // current column and current 3x3 box

    return !used\_in\_row(grid, row, num) &&

        !used\_in\_col(grid, col, num) &&

        !used\_in\_box(grid, row - row % 3, col - col % 3, num);

}

std::pair<int, int> get\_unassigned\_location(int grid[DIM][DIM])

{

    for (int row = 0; row < DIM; row++)

        for (int col = 0; col < DIM; col++)

            if (grid[row][col] == BLANK)

            {

                return std::make\_pair(row, col);

            }

    return GRID\_FULL;

}

bool solve\_sudoku(int grid[DIM][DIM])

{

    // If the Sudoku grid has been filled, we are done

    if (GRID\_FULL == get\_unassigned\_location(grid))

    {

        return true;

    }

    // Get an unassigned Sudoku grid location

    std::pair<int, int> row\_and\_col = get\_unassigned\_location(grid);

    int row = row\_and\_col.first;

    int col = row\_and\_col.second;

    // Consider digits 1 to 9

    for (int num = 1; num <= 9; num++)

    {

        if (is\_safe(grid, row, col, num))

        {

            // Make tentative assignment

            grid[row][col] = num;

            if (solve\_sudoku(grid))

            {

                return true;

            }

            grid[row][col] = BLANK;

        }

    }

    return false;

}

void generate\_sudoku\_puzzle(int grid[DIM][DIM])

{

    // Clear the grid

    for (int i = 0; i < DIM; i++)

    {

        for (int j = 0; j < DIM; j++)

        {

            grid[i][j] = BLANK;

        }

    }

    // Seed the random number generator

    std::srand(static\_cast<unsigned int>(std::time(nullptr)));

    // Fill random numbers in the main diagonal boxes

    for (int box = 0; box < 3; box++)

    {

        int num = 1 + std::rand() % 9; // Generate a random number between 1 and 9

        for (int i = 0; i < 3; i++)

        {

            grid[3 \* box + i][3 \* box + i] = num;

            num = (num % 9) + 1; // Increment and wrap around

        }

    }

}

int main()

{

    cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n\tSudoku Game\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl << endl;

    int grid[DIM][DIM];

    generate\_sudoku\_puzzle(grid);

    cout << "Here is your Sudoku puzzle:\n";

    print\_grid(grid);

    while (true)

    {

        int row, col, num;

        cout << "Enter the row (1-9), column (1-9), and number (1-9) to fill (0 to exit): ";

        cin >> row >> col >> num;

        if (row == 0 || col == 0 || num == 0)

        {

            cout << "Exiting the game. Goodbye!" << endl;

            break;

        }

        if (row < 1 || row > 9 || col < 1 || col > 9 || num < 1 || num > 9)

        {

            cout << "Invalid input. Please enter valid row, column, and number (1-9)." << endl;

            continue;

        }

        row--; // Adjust for 0-based indexing

        col--;

        if (grid[row][col] != BLANK)

        {

            cout << "You can't change this cell. Please choose an empty cell." << endl;

            continue;

        }

        if (!is\_safe(grid, row, col, num))

        {

            cout << "Invalid move. This number cannot be placed here." << endl;

            continue;

        }

        grid[row][col] = num;

        print\_grid(grid);

        if (GRID\_FULL == get\_unassigned\_location(grid))

        {

            if (solve\_sudoku(grid))

            {

                cout << "Congratulations! You solved the puzzle!" << endl << endl;

                break;

            }

            else

            {

                cout << "There seems to be an issue with your solution. Keep going!" << endl << endl;

            }

        }

        else

        {

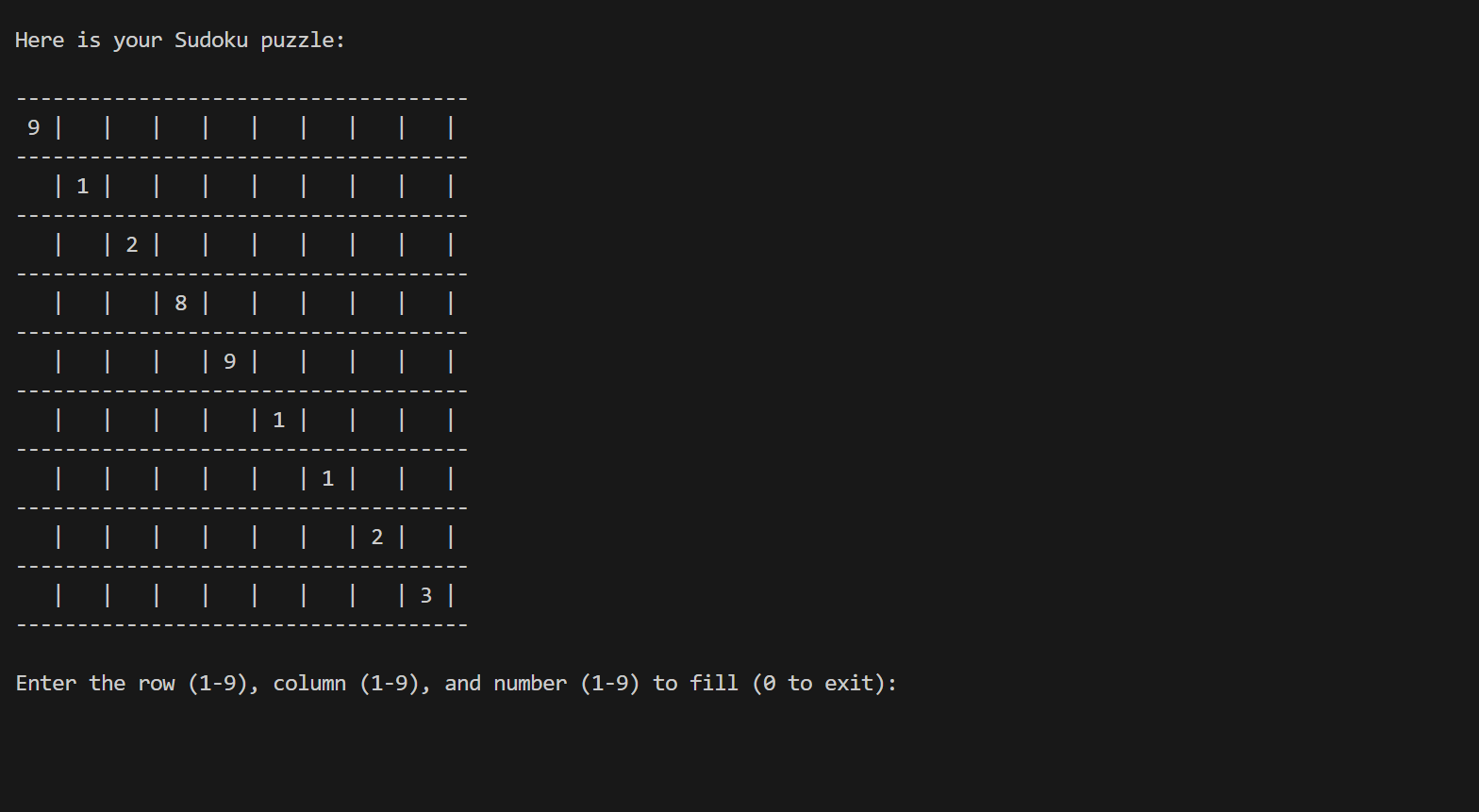
            cout << "Keep going!" << endl << endl;

        }

    }

    return 0;

}



## 

# **LEARNING OUTCOMES**

## Data Structures and Algorithms (DSA):

* Understanding of fundamental data structures such as arrays, linked lists, stacks, queues, trees, graphs, and hash tables.
* Proficiency in implementing and manipulating various data structures efficiently.
* Knowledge of essential algorithms for searching, sorting, graph traversal, recursion, and dynamic programming.
* Ability to analyze the time and space complexity of algorithms.
* Skill in selecting appropriate data structures and algorithms to solve specific problems.
* Problem-solving and algorithmic thinking skills.
* Proficient coding abilities in implementing DSA concepts in programming languages.
* Enhanced ability to optimize code and improve program performance.
* Strong foundation for further learning in advanced topics of computer science.

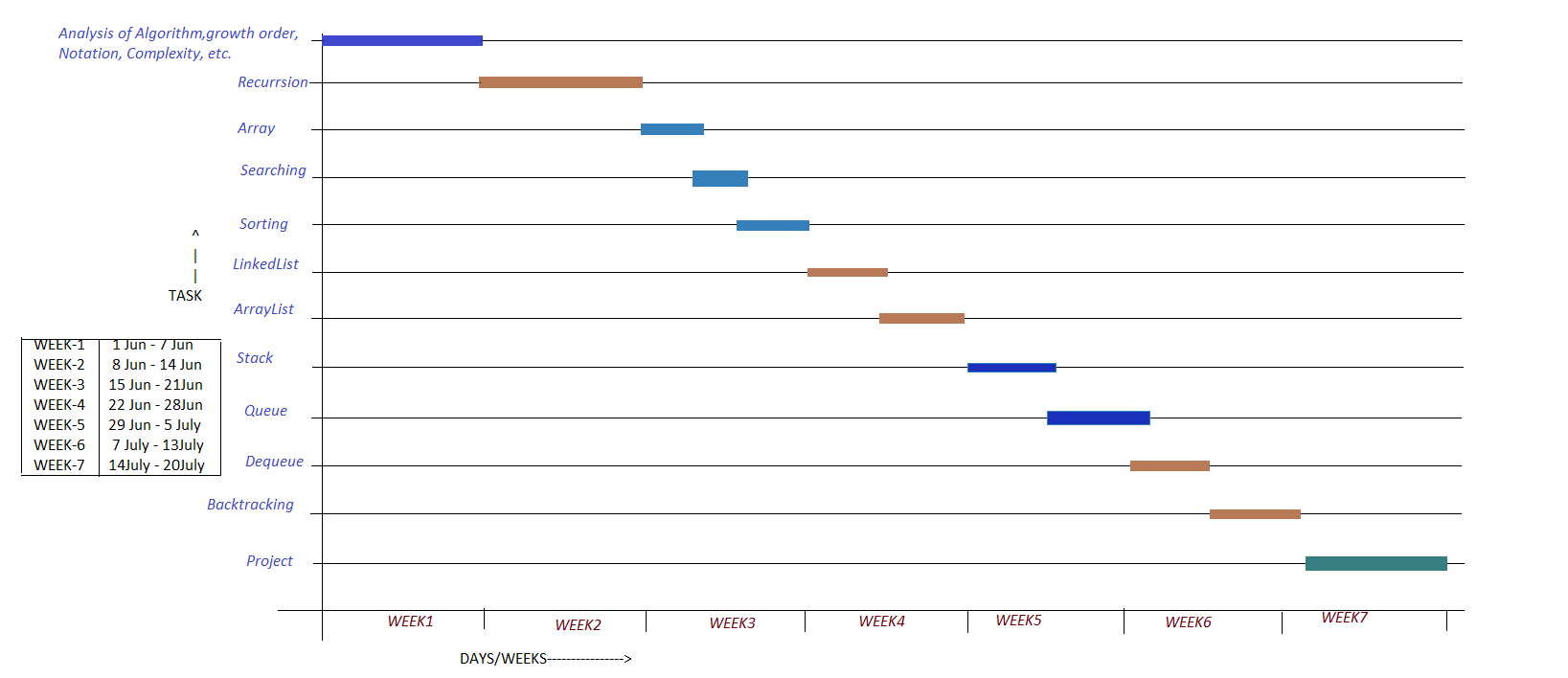
## Sudoku Solver Game:

**Algorithmic Thinking**: Creating a Sudoku solver requires understanding and implementing various algorithms and techniques, such as constraint propagation, backtracking, and heuristics. This enhances your ability to break down complex problems into smaller steps and design effective solutions.**Problem Solving**: Sudoku puzzles present unique challenges that require logical deduction and critical thinking. Developing a solver hones your problem-solving skills as you work through strategies to fill in missing numbers and ensure all rules are satisfied.**Data Structures**: To represent the Sudoku grid and manage possibilities for each cell, you'll need to choose and implement appropriate data structures. This experience improves your grasp of data structures and their applications.**Coding Skills**: Implementing the solver involves writing code to handle user inputs, solve puzzles, and display results. This practical coding experience enhances your programming skills and code organization.**Optimization**: As you work on improving the solver's efficiency, you'll learn optimization techniques like reducing unnecessary computations and choosing the best algorithmic approach for the task.**Debugging**: Debugging is a crucial skill in programming. Developing a Sudoku solver exposes you to debugging challenges as you identify and fix errors in your code.**Algorithm Complexity:** You'll gain insights into the time and space complexity of different algorithms. This helps you evaluate trade-offs between speed and memory usage in your code.

Overall, studying DSA and developing the Tic Tac Toe game provide a range of learning outcomes, including proficiency in data structures and algorithms, enhanced problem-solving skills, improved coding abilities, practical application of game development concepts, and familiarity with user experience design. These skills and knowledge can be applied to various domains within computer science and software engineering, providing a solid foundation for further learning and professional growth.

# 

# GANTT CHART



# **PROJECT LEGACY**

The Data Structures and Algorithms (DSA) project, along with the Tic Tac Toe game, holds a significant legacy in the realm of computer science education and problem-solving applications. This project, developed with meticulous attention to detail, has left a lasting impact on students, developers, and enthusiasts alike.

## **DSA Legacy:**

The DSA project forms the backbone of this legacy, providing a comprehensive and accessible library of essential data structures and algorithms. The well-structured and optimized implementations of arrays, linked lists, stacks, queues, trees, graphs, and hash tables have become invaluable resources for learners and practitioners. Aspiring developers have honed their skills by exploring DSA concepts, enabling them to tackle complex problems efficiently.

Through interactive tutorials and coding exercises, the DSA project has fostered algorithmic thinking and problem-solving abilities among students and developers. It has empowered learners to understand the trade-offs between different algorithms and choose the most appropriate ones for real-world scenarios. The legacy of the DSA project continues to extend its reach, inspiring future generations to embrace the art of algorithm design and analysis.

**Early Days and Mainframes (1960s - 1970s):**Mainframe computers were the primary computing platforms, and they often served multiple users through remote terminals.Early distributed systems focused on centralized computing, with tasks distributed among different terminals or nodes but managed centrally.

**Client-Server Architecture (1980s - 1990s):**The client-server model gained prominence, where tasks were divided between client machines and dedicated server machines.This architecture enabled more efficient resource utilization, load balancing, and centralized management.

**Decentralization and Peer-to-Peer (P2P) (1990s - Early 2000s):**Peer-to-peer networks gained popularity, allowing individual nodes to share resources and information directly with one another.This approach was used for file sharing, distributed computing, and collaborative applications.

**Web-Based Architecture and Services (Late 1990s - Present):**The rise of the World Wide Web led to the development of web-based applications using the client-server model.Web services and APIs facilitated communication and data exchange between distributed components.

**Microservices and Containerization (2010s - Present):**Microservices architecture emerged, focusing on breaking down applications into small, independently deployable services.Containerization technologies like Docker revolutionized deployment by providing lightweight, isolated environments.

**Cloud Computing (2000s - Present):**Cloud computing brought scalable and flexible resources to distributed systems, allowing for virtualized computing resources over the internet.Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) became common deployment models.

## **Sudoku solver Game Legacy:**

**Early Days and Basic Solvers (2000s):**As Sudoku gained popularity, basic solvers emerged that utilized brute force and backtracking algorithms to solve puzzles.These solvers often lacked optimization and were suitable for simpler puzzles but struggled with more complex ones due to their exponential time complexity.

**Constraint Propagation and Optimization (Mid-2000s):**Researchers and programmers started to explore more advanced techniques like constraint propagation, which reduced the search space and improved solving efficiency.Techniques like "naked singles" and "hidden singles" were incorporated to eliminate possibilities in cells and improve solving accuracy.

**Modern Techniques and Machine Learning (2010s - Present):**The advancement of machine learning and artificial intelligence led to experiments with training neural networks to solve Sudoku puzzles.Researchers explored the potential of genetic algorithms, simulated annealing, and other stochastic techniques to tackle Sudoku puzzles.Some solvers integrated user interfaces with real-time highlighting of possible moves and strategies, aiding users in learning the game and its solving techniques.Open Source and Educational Tools:

Open-source Sudoku solver projects became popular, providing educational resources for programmers to understand and implement solving algorithms.Educational platforms and coding challenges introduced Sudoku puzzles as programming tasks, encouraging developers to practice their algorithmic skills.

**Web-Based Solvers and Mobile Apps:**Sudoku solvers transitioned from desktop applications to web-based platforms, allowing users to solve puzzles directly in their browsers.Mobile apps for smartphones and tablets became common, providing convenient access to Sudoku puzzles and solvers on the go.

# 

# **BIBLIOGRAPHY**

* Data Structure algorithm in Java, By Roberto Tamassia, 1997
* Introduction to Algorithms by Ronald Rivest, 1989