**Annexure-I: DSA- Self paced (Geeks for Geeks)**

Submitted to-



# A training report

Submitted in partial fulfillment of the requirements for the award of degree of

**SUMMER MOOC**

on

# (DATA STRUCTURES AND ALGORITHMS)

# Submitted by

**(Ashwani Pandey)**

**Registration No-12211155 Program Name- BCA**

(June – July 2024)

**School of Computer Application Lovely Professional University, Phagwara, Punjab**

**Annexure-II: Student Declaration**

# 

# To whom so ever it may concern

I, **Ashwani Pandey, 12211155,** hereby declare that the work done by me on “**DSA**” from **June 2024** to **July 2024**, is a record of original work for the partial fulfillment of the requirements for the award of the degree, **Bachelor of Computer Application.**

Ashwani Pandey (12211155)

Dated: 23/07/24

**ACKNOWLEDGEMENT**

Primarily I would like to thank God for being able to learn a new technology. Then I would like to express my special thanks of gratitude to the teacher and instructor of the course DSA Self-Paced from Geeks For Geeks who provide me the golden opportunity to learn a new technology from home.

I would like to also thank my own college Lovely Professional University for offering such a course which not only improve my programming skill but also taught me other new technology.

Then I would like to thank my parents and friends who have helped me with their valuable suggestions and guidance for choosing this course.

Last but not the least I would like to thank my all classmates who have helped me a lot.

[Ashwani Pandey]

[12211155]

**INTERNSHIP CERTIFICATE**

(As given by MOOC or Organization in original)



**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Title** | **Page** |
| 1 | Cover Page | 1 |
| 2 | Declaration of the student | 2 |
| 3 | Acknowledgement | 3 |
| 4 | Training Certification from organization | 4 |
| 5 | Table of Contents | 5 |
| 6 | Introduction | 7 |
| 7 | Technology learnt | 11 |
| 8 | Reason for choosing DSA | 18 |
| 9 | Learning Outcome | 34 |
| 10 | Introduction of Mini Project | 38 |
| 11 | Details of Mini Project | 39 |
| 12 | Interface Design | 42 |
| 13 | Code Snippet | 44 |
| 14 | Conclusion and Future Perspective | 45 |
| 15 | Bibliography and References | 46 |

**INTRODUCTION OF THE COURSE**

The **Data Structures and Algorithms - Self Paced** is a comprehensive program designed to provide learners with a rigorous understanding of fundamental and advanced concepts in data structures and algorithms (DSA). This course target both beginners and intermediate learners and this course is ideal for students like: aspiring software engineers, and competitive programmers.

The course-curriculum covers a broad range of topics, including arrays, linked lists, stacks, queues, trees, graphs, dynamic programming, and many more. Each module is followed by interactive coding exercises, quizzes, and assessments that ensure a hands-on learning experience. The self-paced nature of the course allows participants to progress at their own speed, making it convenient for those with varied learning schedules.

The course name DSA stands for “Data Structures and Algorithms” and Self-paced means, one can join the course anytime. All of the content will be available once one gets enrolled. One can finish it at his own decided speed.

**What is Data Structure?**

Before defining data structures, let’s define, “[What is data](https://www.simplilearn.com/what-is-data-article)**?”**

Data is information optimized for processing and movement, facts and figures stored on computers. Data structures are a specific way of organizing data in a specialized format on a computer so that the information can be organized, processed, stored, and retrieved quickly and effectively.

They are a means of handling information, rendering the data for easy use.

Every application, piece of software, or programs foundation consists of two components: algorithms and data. Data is information, and algorithms are rules and instructions that turn the data into something useful to [programming.](https://www.simplilearn.com/how-to-learn-programming-article)

Related data + Permissible operations on the data = Data Structures

Data structures + Algorithms = Programs

**What is Algorithm?**

An algorithm is a well-defined sequential computational technique that accepts a value or a collection of values as input and produces the output(s) needed to solve a problem.

Or we can say that an algorithm is said to be accurate if and only if it stops with the proper output for each input instance.

### **Example:**

Consider a box where no one can see what’s happening inside, we say a black box.

We give input to the box and it gives us the output we need but the procedure that we might need to know behind the conversion of input to desired output is an ALGORITHM.

An algorithm is independent of the language used. It tells the programmer the logic used to solve the problem. So, it is a logical step-by-step procedure that acts as a blueprint to programmers.

Chart of the Data-Structures:-

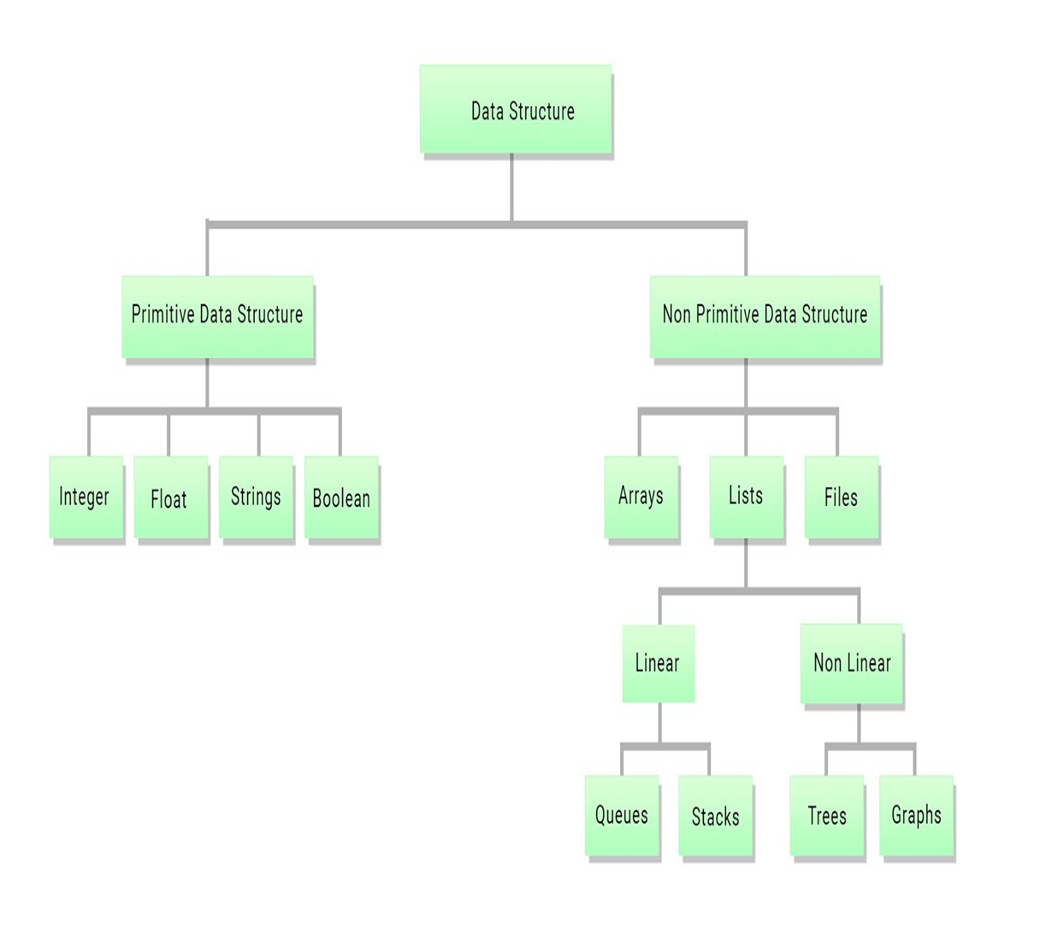


Figure : Data Structure Chart

Now, the introduction of various topics:

**1. Analysis of Algorithms**

This topic focuses on evaluating the efficiency of algorithms in terms of time and space complexity. It introduces concepts like Big O, Omega, and Theta notations, helping determine the performance and scalability of algorithms.

**2. Mathematics**

Mathematics forms the foundation for algorithm design and analysis. Topics like number theory, combinatorics, probability, and modular arithmetic are covered to build logic and problem-solving abilities essential for DSA.

**3. Bit Magic**

Bit manipulation techniques are powerful tools for optimizing algorithms. This topic explores operations like bitwise AND, OR, XOR, and shifting, along with their applications in problems like finding unique elements and efficient arithmetic operations.

**4. Recursion**

Recursion is the technique of solving problems where a function calls itself with a smaller sub-problem. This topic covers the basics of recursion, its use cases, and how it simplifies complex problems, along with understanding base cases and recursive cases.

**5. Array**

Arrays are linear data structures used to store elements in contiguous memory locations. This topic discusses array operations like insertion, deletion, traversal, and covers various problems like subarray sums and rotations.

**6. Searching**

Searching involves finding elements in a collection. This topic covers linear and binary search algorithms, their efficiency, and their practical applications in finding data within arrays and lists.

**7. Sorting**

Sorting algorithms arrange elements in a specific order. This section covers different algorithms like Bubble Sort, Merge Sort, Quick Sort, and more, with emphasis on their time complexity and stability.

**8. Matrix**

Matrices are 2D arrays widely used in computer science for representing grids, images, and graphs. This topic introduces operations on matrices, matrix traversal, and problems like rotation and diagonal traversal.

**9. Hashing**

Hashing involves mapping data to a fixed-size value using hash functions. This topic explains hash tables, collision resolution techniques like chaining and open addressing, and their applications in searching and indexing.

**10. Linked Lists**

Linked lists are dynamic data structures consisting of nodes linked together. This section covers types like singly, doubly, and circular linked lists, and operations like insertion, deletion, and traversal.

**11. Strings**

Strings are sequences of characters used to store and manipulate text. This topic covers string operations, pattern matching algorithms like KMP, and problems like anagram checks and palindrome verification.

**12. Stack**

Stacks follow a Last-In-First-Out (LIFO) order and are used in problems like expression evaluation and backtracking. This topic discusses stack operations, implementation, and applications like balancing parentheses.

**13. Queue**

Queues follow a First-In-First-Out (FIFO) order, making them useful in scenarios like scheduling and buffering. This section covers queue operations, types (circular and priority queues), and applications in breadth-first search (BFS).

**14. Dequeue**

Dequeue, or double-ended queue, allows insertion and deletion at both ends. This topic introduces the concept, its operations, and applications like implementing sliding window problems.

**15. Binary Search Tree (BST)**

BSTs are specialized tree structures that maintain a sorted order of elements. This topic covers insertion, deletion, searching, and traversal operations in BSTs, along with their applications in maintaining ordered data.

**16. Tree**

Trees are hierarchical data structures with nodes connected by edges. This section covers types of trees (binary, n-ary), tree traversal techniques (inorder, preorder, postorder), and tree-related problems like depth and diameter.

**17. Heap**

Heaps are specialized tree-based data structures used to implement priority queues. This topic discusses Min-Heap and Max-Heap, heap operations, and their applications in problems like finding the k-largest elements.

**18. Graph**

Graphs represent networks of connected nodes. This section introduces graph representations (adjacency list and matrix), traversal algorithms (BFS, DFS), and shortest path algorithms (Dijkstra, Bellman-Ford).

**19. Greedy**

The greedy approach involves making locally optimal choices to achieve a global solution. This topic covers classic greedy algorithms like Kruskal’s MST, Huffman Coding, and their applications in optimization problems.

**20. Backtracking**

Backtracking is a method for solving problems recursively by trying out all possible solutions and undoing choices that lead to dead ends. This section covers problems like N-Queens, Sudoku, and combinatorial generation.

**21. Dynamic Programming (DP)**

Dynamic programming is an optimization technique that solves complex problems by breaking them down into simpler subproblems. This topic covers classic DP problems like Knapsack, Longest Common Subsequence, and others.

**22. Trie**

Tries are tree-like data structures used for storing collections of strings, making them efficient for prefix-based searching. This section covers trie operations and applications like autocomplete and spell check.

**23. Segment Tree and Binary Indexed Tree (BIT)**

Segment Trees and BITs are advanced data structures used for efficient range queries and updates. This topic explains their construction, operations, and applications in problems like range sum queries.

**24. Disjoint Set**

Disjoint sets, also known as Union-Find, are used for managing partitioned sets and efficiently handling union and find operations. This topic covers their applications in graph algorithms like Kruskal’s MST and connectivity problems.

**TECHINOLOGY LEARNT FROM THIS COURSE**

* Learn Data Structures and Algorithms from basic to an advanced level like:
* Learn Topic-wise implementation of different Data Structures & Algorithms as follows

## **Analysis of Algorithm**

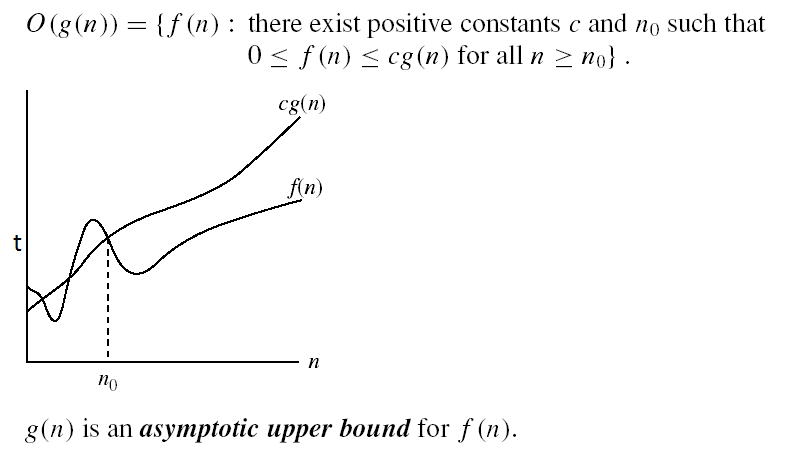
* + - In this I learned about background analysis through a Program and its functions.

## **Order of Growth**

* + - A mathematical explanation of the growth analysis through limits and functions.
    - A direct way of calculating the order of growth

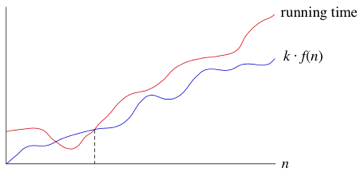
## **Asymptotic Notations**

* + - Best, Average and Worst-case explanation through a program.



## **See the source imageBig O Notation**

* + - Graphical and mathematical explanation.
    - Calculation
    - Applications at Linear Search



## **Omega Notation**

* + - Graphical and mathematical explanation.
    - Calculation.

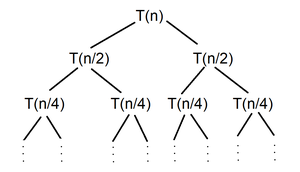
## **See the source imageTheta Notation**

* + - Graphical and mathematical explanation.
    - Calculation.

## **Analysis of common loops**

* + - Single, multiple and nested loops.

## **Analysis of Recursion**

* + - Various calculations through Recursion Tree method
  + 

## **Space Complexity**

* + - Basic Programs
    - Auxiliary Space
    - Space Analysis of Recursion
    - Space Analysis of Fibonacci number

## **MATHEMATICS**

* + **Finding the number of digits in a number.**

## Arithmetic and Geometric Progressions.

* + **Quadratic Equations.**

## Mean and Median.

* + **Prime Numbers.**

## LCM and HCF

* + **Factorials**

## Permutations and Combinations

* + **Modular Arithmetic**

**BITMAGIC**

## **Bitwise Operators in C++**

* + - Operation of AND, OR, XOR operators
    - Operation of Left Shift, Right Shift and Bitwise Not

## **Bitwise Operators in Java**

* + - Operation of AND, OR
    - Operation of Bitwise Not, Left Shift
    - Operation of Right Shift and unsigned Right Shift

## **Problem (With Video Solutions): Check Kth bit is set or not**

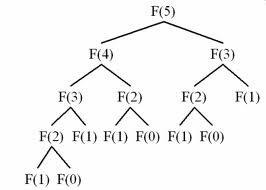
* + - Method 1: Using the left Shift.
    - Method 2: Using the right shift

## **RECURSION**

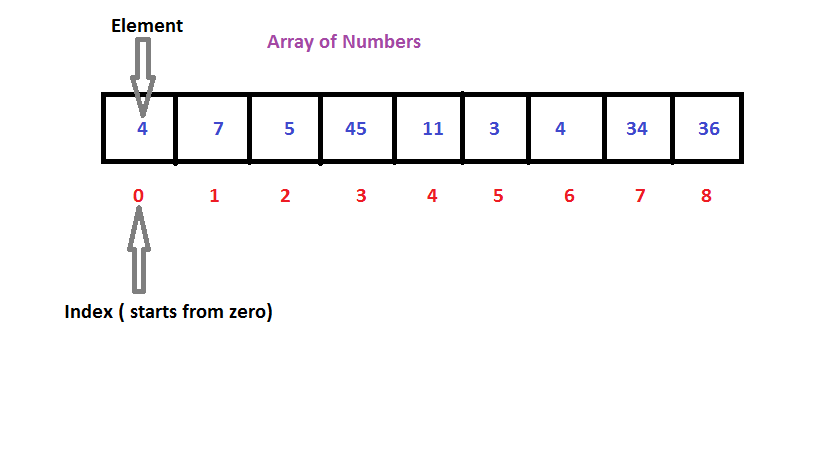
* **Introduction to Recursion**

## Applications of Recursion

* **Writing base cases in Recursion**
  + Factorial
  + N-th Fibonacci number



## **ARRAYS**



* **Introduction and Advantages**

## **Types of Arrays**

* + - Fixed-sized array
    - Dynamic-sized array

## **Operations on Arrays**

* + - Searching
    - Insertions
    - Deletion
    - Arrays vs other DS
    - Reversing - Explanation with complexity

## **SEARCHING**

* **Binary Search Iterative and Recursive**

## Binary Search and various associated problems

* **Two Pointer Approach Problems**

**SORTING**

## Implementation of C++ STL sort () function in Arrays and Vectors

* + - Time Complexities

## Sorting in Java

* + **Arrays. Sort() in Java**

## Collection.sort() in Java

* + **Stability in Sorting Algorithms**
    - Examples of Stable and Unstable Algos

## **Insertion Sort**

* + **Merge Sort**

## **Quick Sort**

* + - Using Lomuto and Hoare
    - Time and Space analysis
    - Choice of Pivot and Worst case

## **Overview of Sorting**

## See the source image

## **Algorithms MATRIX**

* **Introduction to Matrix in C++ and Java**

## Multidimensional Matrix

* **Pass Matrix as Argument**

## Printing matrix in a snake pattern

* **Transposing a matrix**

## Rotating a Matrix

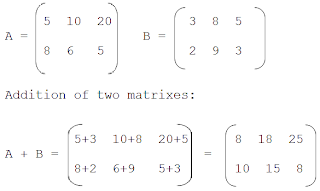
* **Check if the element is present in a row and column-wise sorted matrix.**

## Boundary Traversal

* **Spiral Traversal**

## Matrix Multiplication

## Search in row-wise and column-wise Sorted Matrix



**HASHING**

## Introduction and Time complexity analysis

* **Application of Hashing**

## Discussion on Direct Address Table

* **Working and examples on various Hash Functions**

## Introduction and Various techniques on Collision Handling

* **Chaining and its implementation**

## Open Addressing and its Implementation

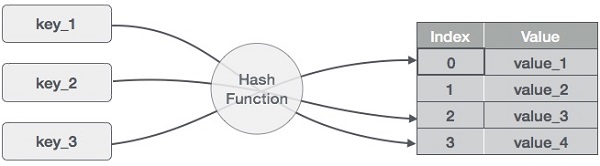
* **Chaining V/S Open Addressing**

## Double Hashing

* **C++**
  + Unordered Set
  + Unordered Map

## Java

* + HashSet
  + HashMap



## **STRINGS**

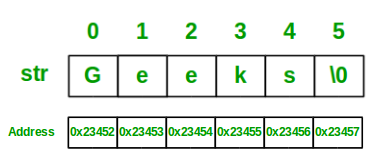
* + **Discussion of String DS**

## Strings in CPP

* + **Strings in Java**

## Rabin Karp Algorithm

* + **KMP Algorithm**



**LINKED LIST**

## **Introduction**

* + - Implementation in CPP
    - Implementation in Java
    - Comparison with Array DS

## **Doubly Linked List**

* **Circular Linked List**

## **Loop Problems**

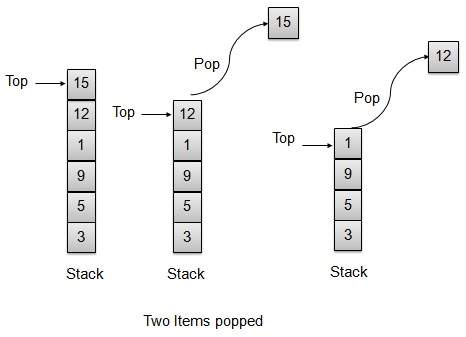
* + - Detecting Loops
    - Detecting loops using Floyd cycle detection
    - Detecting and Removing Loops in Linked List

## See the source image

## STACK

* **Understanding the Stack data structure**

## Applications of Stack

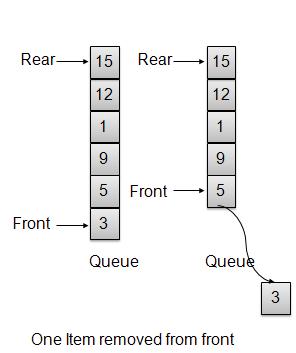
* **Implementation of Stack in Array and Linked List**
  + - In C++
    - In Java
    - 

## **QUEUE**

* **Introduction and Application**

## Implementation of the queue using array and LinkedList

* + In C++ STL
  + In Java
  + Stack using queue



## **DEQUE**

## **See the source image**

## 

* **Introduction and Application**

## **Implementation**

* + - In C++ STL
    - In Java

## **Problems (With Video Solutions)**

* + - Maximums of all subarrays of size k
    - ArrayDeque in Java.
    - Design a DS with min max operations

## **TREE**

## See the source image

* **Introduction** 
  + - Tree
    - Application
    - Binary Tree
    - Tree Traversal

## **Implementation of:**

* + - Inorder Traversal
    - Preorder Traversal
    - Postorder Traversal
    - Level Order Traversal (Line by Line)
    - Tree Traversal in Spiral Form

## **BINARY SEARCH TREE**

* **Background, Introduction and Application**

## Implementation of Search in BST

* **Insertion in BST**

## Deletion in BST

* **Floor in BST**

## Self-Balancing BST

* **AVL Tree**

## **HEAP**

* **Introduction & Implementation**

## **Binary Heap**

* + - Insertion
    - Heapify and Extract
    - Decrease Key, Delete and Build Heap
* **Priority Queue in C++, Java**

****

**GRAPH**

* **Introduction to Graph**

## Graph Representation

* + - Adjacency Matrix
    - Adjacency List in CPP and Java
    - Adjacency Matrix VS List

## Breadth-First Search

* + - Applications

## Depth First Search

* + - Applications

## Shortest Path in Directed Acyclic Graph

* **Prim's Algorithm/Minimum Spanning Tree**
  + - Implementation in CPP
    - Implementation in Java

## Dijkstra's Shortest Path Algorithm

* + - Implementation in CPP
    - Implementation in Java

## Bellman-Ford Shortest Path Algorithm

* **Kosaraju's Algorithm**

## Articulation Point

* **Bridges in Graph**

## Tarjan’s Algorithm

## See the source image

## **GREEDY**

* **Introduction**

## Activity Selection Problem

* **Fractional Knapsack**

## Job Sequencing

## **BACKTRACKING**

* **Concepts of Backtracking**

## Rat In a Maze

* **N Queen Problem**

**DYNAMIC PROGRAMMING**

## **Introduction**

* **Dynamic Programming**
  + - Memoization
    - Tabulation

## **TREE**

* **Introduction**
  + - Representation
    - Search
    - Insert
    - Delete

## Count Distinct Rows in a Binary

## **Matrix SEGMENT TREE**

* **Introduction**

## Construction

* **Range Query**

## Update Query

**DISJOINT SET**

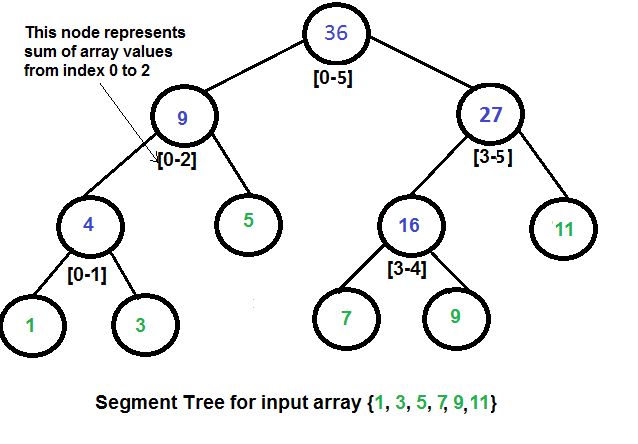
## Introduction

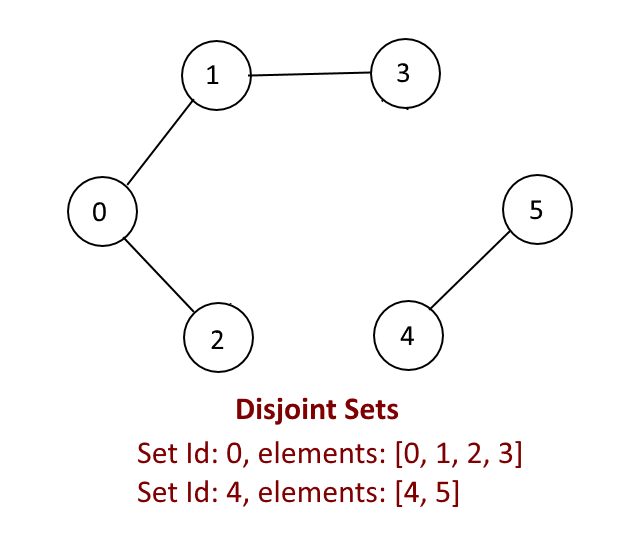
* **Find and Union Operations**

## Union by Rank

* **Path Compression**

Kruskal's Algorithm





* Improved my problem-solving skills by practicing problems to become a stronger developer

**Practice problems**

This track contains many practice problems for the users which are considered important and must-do as far as Data Structure and Algorithm is concerned

* Developed my analytical skills on Data Structures to use them efficiently
* Solved problems asked in product-based companies’ interviews
* Solved problems in contests similar to coding round for SDE role

**Reason for choosing this technology**

* + With advancement and innovation in technology, programming is becoming a highly in-demand skill for Software Developers. Everything you see around yourself from Smart TVs, ACs, Lights, Traffic Signals uses some kind of programming for executing user commands.



***“In order to be irreplaceable, one must always be eicient.”***

**Data Structures** and **Algorithms** are the identity of a good Software Developer. The interviews for technical roles in some of the tech giants like *Google, Facebook, Amazon, Flipkart* is more focused on measuring the knowledge of Data Structures and Algorithms of the candidates. The main reason behind this is Data Structures and Algorithms improves the problem-solving ability of a candidate to a great extent.

* + This course has video lectures of all the topics from which one can easily learn. I prefer learning from video rather than books and notes. I know books and notes and thesis have their own significance but still video lecture or face to face lectures make it easy to understand faster as we are involved Practically.
  + It has 200+ algorithmic coding problems with video explained solutions.
  + It has track based learning and weekly assessment to test my skills.
  + It was a great opportunity for me to invest my time in learning instead of wasting it here and there during my summer break in this Covid-19 pandemic.
  + This was a lifetime accessible course which I can use to learn even after my training whenever I want to revise.

**LEARNING OUTCOMES**

Programming is all about data structures and algorithms. Data structures are used to hold data while algorithms are used to solve the problem using that data.

Data structures and algorithms (DSA) goes through solutions to standard problems in detail and gives you an insight into how efficient it is to use each one of them. It also teaches you the science of evaluating the efficiency of an algorithm. This enables you to choose the best of various choices.

For example, you want to search your roll number in 30000 pages of documents, for that you have choices like Linear search, Binary search, etc. So, the more efficient way will be Binary search for searching something in a huge number of data.

So, if you know the DSA, you can solve any problem efficiently. The main use of DSA is to make your code scalable because

* + Time is precious
  + Memory is expensive

# One of the uses DSA is to crack the interviews to get into the product- based companies

In our daily life, we always go with that person who can complete the task in a short amount of time with efficiency and using fewer resources. The same things happen with these companies. The problem faced by these companies is much harder and at a much larger scale. Software developers also have to make the right decisions when it comes to solving the problems of these companies.

For example, in an interview, your given a problem to find the sum of first N natural numbers.

One candidate solves it by using loop like

## **Initialize sum = 0**

**for every natural number n in range 1 to N (inclusive): add n to sum**

## **sum is the answer**

And you solve it using the sum of first N natural numbers is given by the formula:

## **Sum=N\*(N+1)/2**

Obviously, they will choose you over other one because your solution is more efficient.

Knowledge of data structures like Hash Tables, Trees, Tries, [Graphs,](http://www.geeksforgeeks.org/graph-data-structure-and-algorithms/) and various [algorithms](https://www.geeksforgeeks.org/fundamentals-of-algorithms/) goes a long way in solving these problems efficiently and the

interviewers are more interested in seeing how candidates use these tools to solve a problem. Just like a car mechanic needs the right tool to fix a car and make it run properly, a programmer needs the right tool (algorithm and data structure) to make the software run properly. So, the interviewer wants to find a candidate who can apply the right set of tools to solve the given problem. If you know the characteristics of one data structure in contrast to another you will be able to make the right decision in choosing the right data structure to solve a problem.

# Another use of DSA, if you love to solve the real-world complex problems.

Let’s take the example of Library. If you need to find a book on Set Theory from a library, you will go to the math section first, then the Set Theory section. If these books are not organized in this manner and just distributed randomly then it will be frustrating to find a specific book. So, data structures refer to the way we organize information on our computer. Computer scientists process and look for the best way we can organize the data we have, so it can be better processed based on input provided.

A lot of newbie programmers have this question that where we use all the stuff of data structure and algorithm in our daily life and how it’s useful in solving the real-world complex problem. We need to mention that whether you are interested in getting into the top tech giant companies or not DSA still helps a lot in your day to day life.

Let’s consider some examples

* + In Facebook you can represent your friends on Facebook, friends of friends, mutual friends easily by Graph.
  + If you need to keep a deck of cards and arrange it properly, how would you do that? You will throw it randomly or you will arrange the cards one over another and from a proper deck. You can use Stack here to make a proper arrangement of cards one over another.
  + If you need to search a word in the dictionary, what would be your approach? Do you go page by page or you open some page and if the word is not found you open a page prior/later to one opened depending upon the order of word to the current page (Binary Search)?

The first two were a good example of choosing the right data structure for a real-world problem and the third one is a good example of choosing the right algorithm to solve a specific problem in less amount of time.



***“Data structure and algorithms help in understanding the nature of the problem at a deeper level and thereby a better understanding of the***

***world.”***

**INTRODUCTION OF MINI PROJECT**

### **Mini Project: - Sudoku Solver**

The Sudoku Solver project focuses on building an algorithm that solves Sudoku puzzles using concepts from Data Structures and Algorithms. Sudoku is a popular logic-based puzzle that involves filling a 9x9 grid with digits from 1 to 9, ensuring that each row, column, and 3x3 sub grid contains all digits exactly once. This project leverages techniques like backtracking to efficiently solve the puzzle.

The primary goal of this project is to create an algorithm that can automatically solve any valid Sudoku puzzle. The problem is not only an excellent test of logic and algorithmic skills but also serves as an introduction to backtracking, a common problem-solving approach in computer science.

**DETAILS OF MINI PROJECT**

# Details Of Mini Project

#### **Project Title:**

**Sudoku Solver using Backtracking Algorithm**

#### **2. Project Overview:**

This mini project focuses on designing and implementing an automated solution to solve any valid Sudoku puzzle using the backtracking algorithm. The project demonstrates how to tackle constraint-based problems using recursive approaches and provides hands-on experience with algorithmic problem-solving in Python (or any other programming language).

#### **3. Problem Statement:**

The objective of the project is to build a program that can take an incomplete Sudoku grid as input and fill it with numbers such that it adheres to all Sudoku rules:

* Each digit from 1 to 9 appears exactly once in each row.
* Each digit from 1 to 9 appears exactly once in each column.
* Each digit from 1 to 9 appears exactly once in each 3x3 subgrid.

#### **4. Tools and Technologies Used:**

* **Programming Language**: Python (or another language like C++, Java)
* **Algorithm**: Backtracking
* **Optional GUI**: Tkinter for a graphical interface (if implemented)

#### **5. Core Concepts Applied:**

* **Backtracking Algorithm**: A recursive approach that attempts every possible configuration while maintaining the problem constraints. When a conflict is detected, it backtracks and tries a different configuration.
* **Constraint Satisfaction**: Ensuring that each number placement conforms to the Sudoku rules, leading to a valid solution.

#### **6. Project Implementation Steps:**

**Step 1: Input Handling**

* The program starts by taking a 9x9 grid as input, where some cells are filled with numbers and others are empty.
* The grid can be input manually, read from a file, or through a GUI.

**Step 2: Grid Validation Function**

* A function is implemented to check if placing a particular number in a given cell is valid, considering the current state of the row, column, and 3x3 subgrid.

**Step 3: Backtracking Algorithm**

* A recursive function is developed to fill in the empty cells one by one.
* For each empty cell, numbers from 1 to 9 are attempted.
* If a number is valid according to the validation function, it is placed in the cell, and the algorithm moves to the next empty cell.
* If no valid number can be placed, the algorithm backtracks to the previous cell and tries the next number in sequence.
* The process continues until the entire grid is filled correctly.

**Step 4: Display the Solved Grid**

* Once the puzzle is solved, the solution is displayed in the console or GUI.

**Step 5: Optional GUI Implementation**

* For enhanced user interaction, a graphical interface can be added using Tkinter (in Python). The interface allows users to input a puzzle, visualize the solving process, and view the final solution.

#### **7. Key Challenges Faced:**

* **Handling Edge Cases**: Dealing with scenarios like grids with multiple solutions or unsolvable configurations.
* **Optimizing the Backtracking Process**: Reducing the search space by implementing smart heuristics like filling cells with the least possible candidates first.
* **Ensuring Code Efficiency**: Balancing readability and optimization, especially for large or complex puzzles.

#### **8. Testing and Validation:**

* The algorithm was tested against various difficulty levels of Sudoku puzzles (easy, medium, hard, and expert).
* It was validated to ensure correct results for all inputs, handling both standard and edge cases effectively.

#### **9. Learning Outcomes:**

* **Deep Understanding of Backtracking**: The project enhances knowledge of recursive algorithms and their applications in solving constraint-based problems.
* **Practical Coding Skills**: It improves coding practices related to recursion, input validation, and optimization.
* **Problem-Solving Strategies**: The project reinforces the importance of choosing appropriate data structures and algorithms for different problem types.
* **Project Development Lifecycle**: The end-to-end development process, from problem analysis to testing and optimization, provided insights into structuring and managing coding projects.

#### **10. Project Conclusion:**

The Sudoku Solver mini project is a successful implementation of algorithmic problem-solving using backtracking. It provides a strong foundation for understanding complex recursive problems and offers a real-world application of data structures and algorithms. The project also serves as an excellent entry point for exploring advanced concepts like constraint satisfaction and optimization in a user-friendly manner.

**INTERFACES DESIGN**

Sudoku Solver

1.

Table

Description automatically generated

2.

Table, calendar

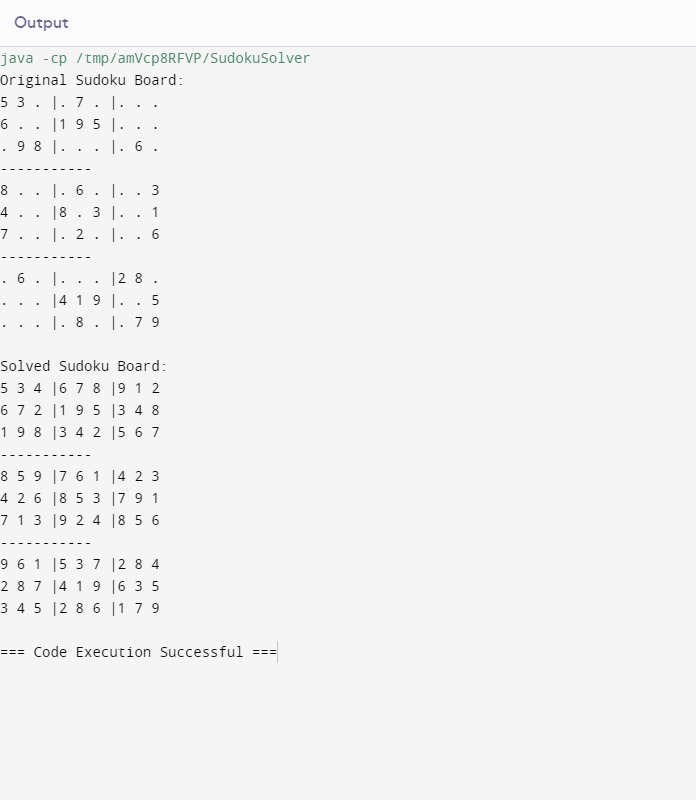
Description automatically generated

3.

Calendar

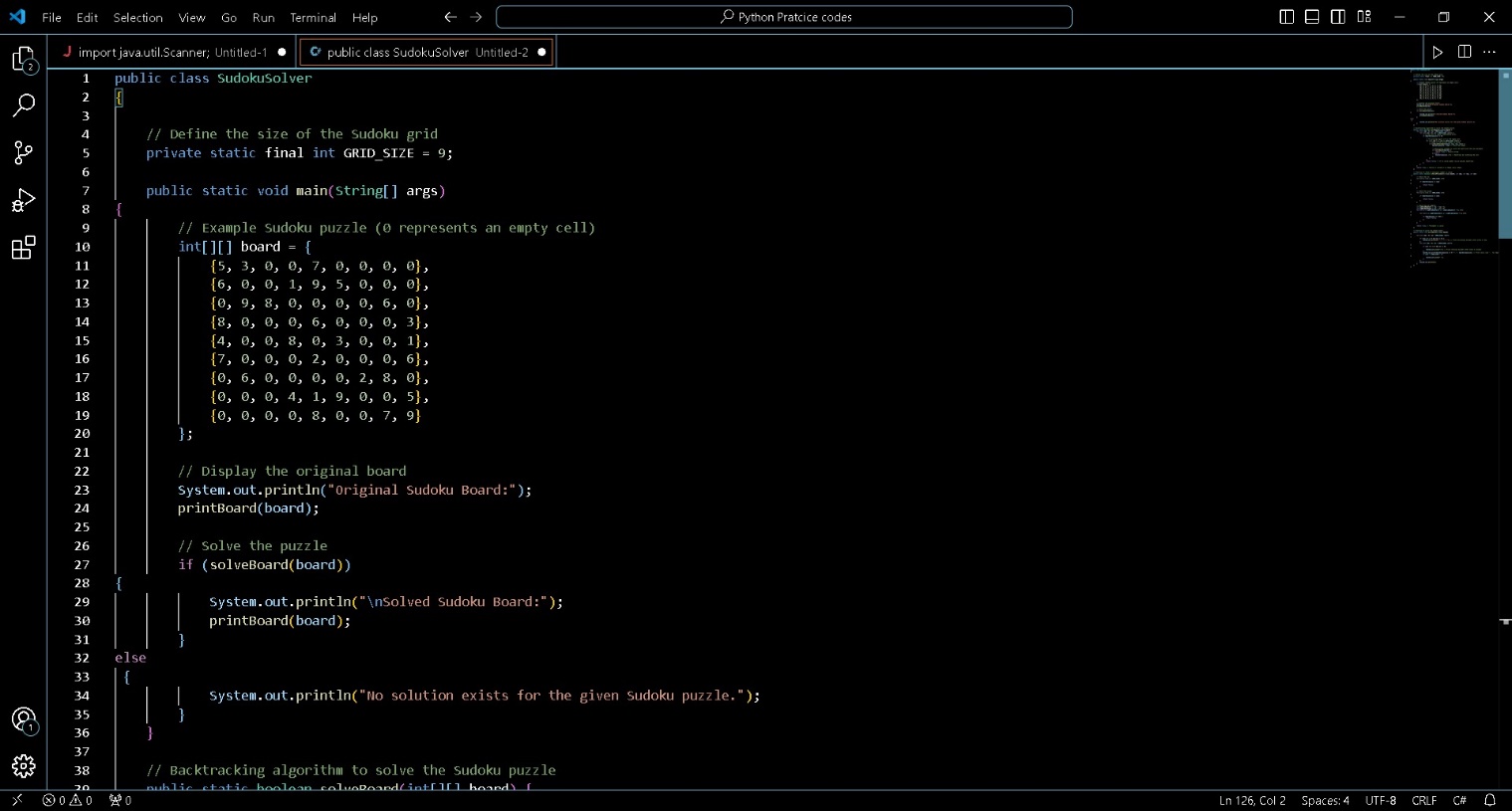
Description automatically generated

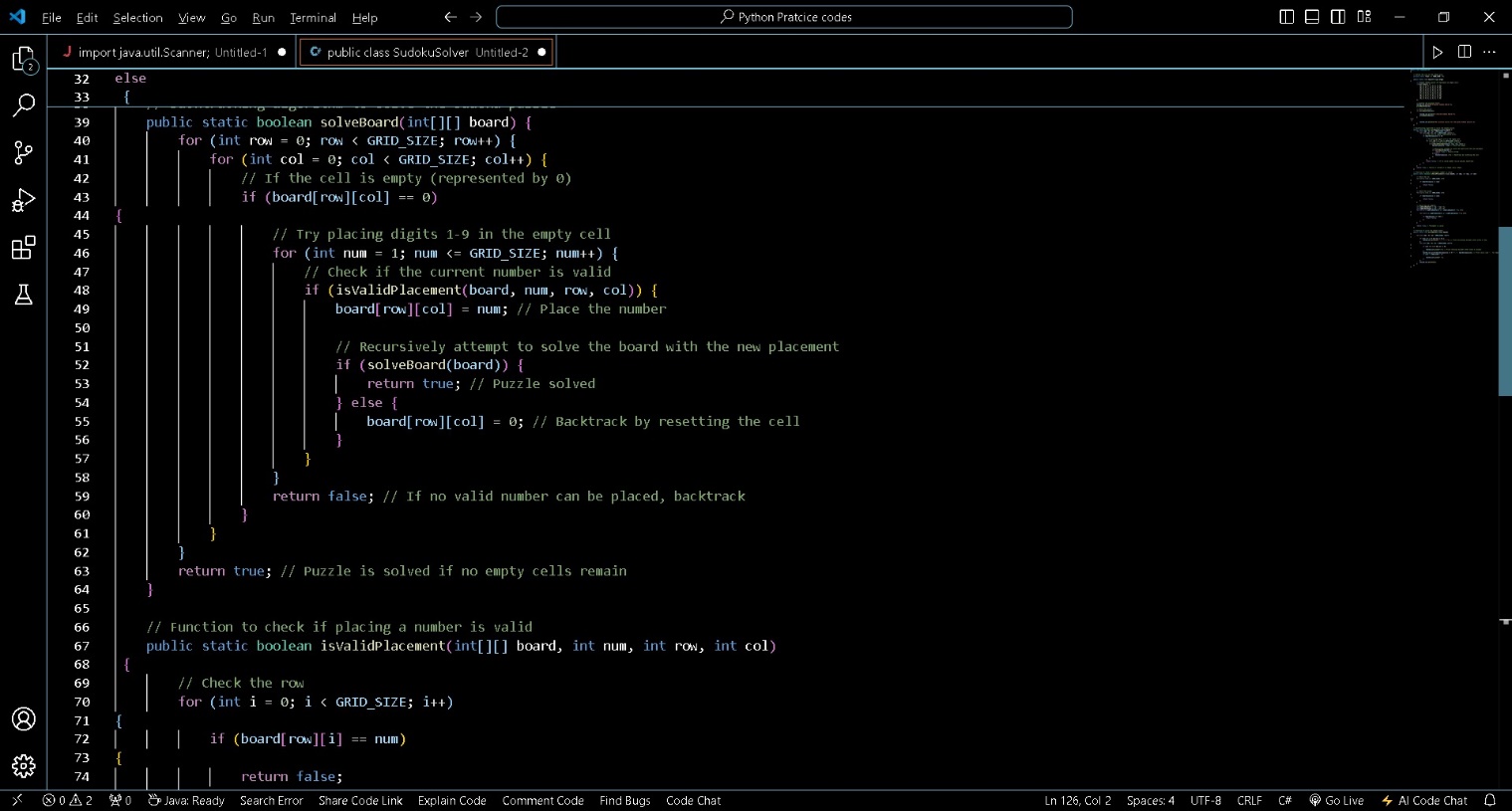
4.

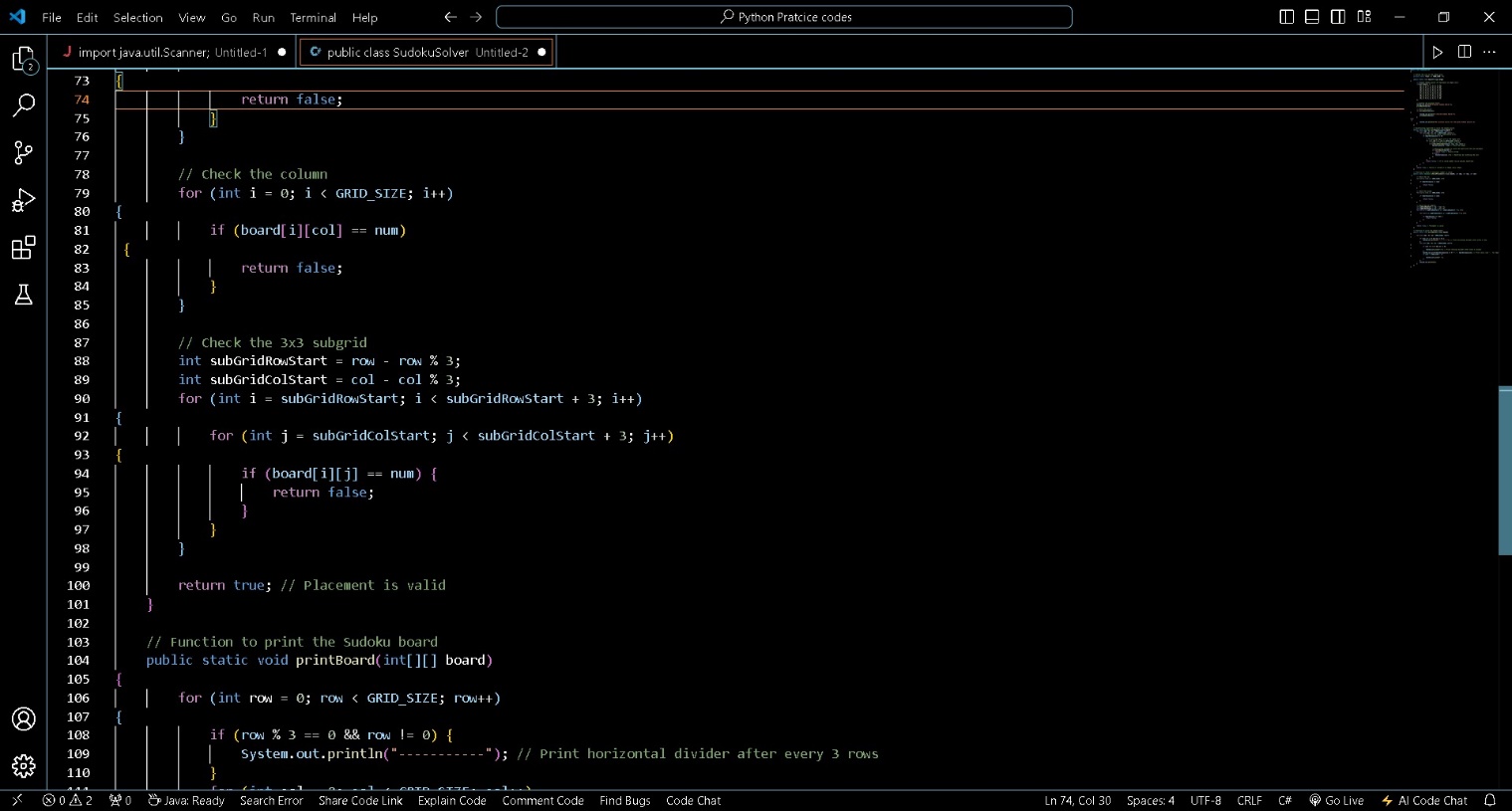


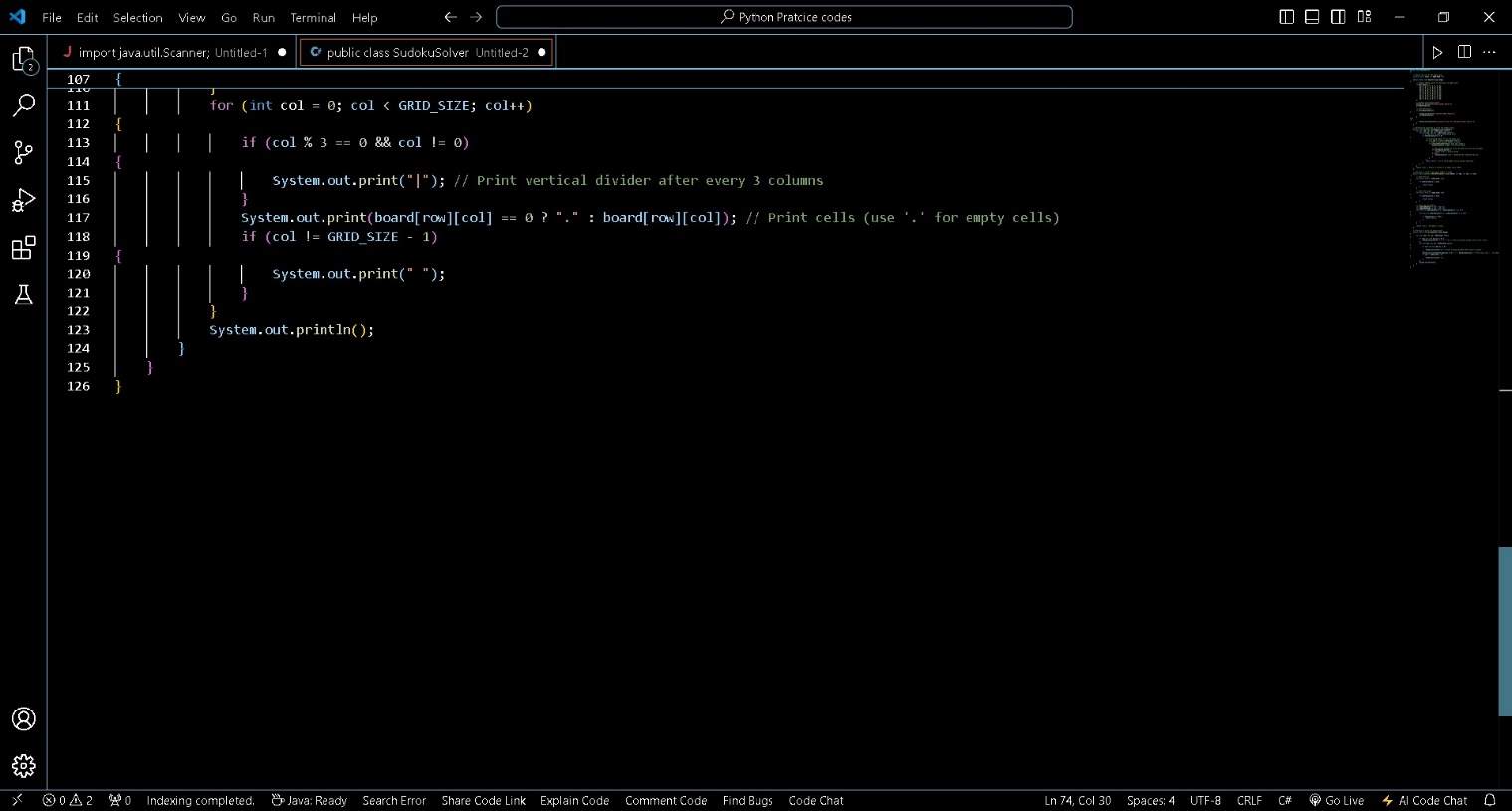
**CODE SNIPPET**

* 1. **In JAVA:-**









**FINAL CHAPTER:**

**CONCLUSION AND FUTURE PERSPECTIVE**

* Improved my problem-solving skills by practicing problems to become a stronger developer

**Practice problems**

This track contains many practice problems for the users which are considered important and must-do as far as Data Structure and Algorithm is concerned

* Developed my analytical skills on Data Structures to use them efficiently
* Solved problems asked in product-based companies’ interviews
* Solved problems in contests similar to coding round for SDE role

**Reason for choosing this technology:**

•With advancement and innovation in technology, programming is becoming a highly in-demand skill for Software Developers. Everything you see around yourself from Smart TVs, ACs, Lights, Traffic Signals uses some kind of programming for executing user commands.

Data Structures and Algorithms are the identity of a good Software Developer. The interviews for technical roles in some of the tech giants like Google, Facebook, Amazon, Flipkart is more focused on measuring the knowledge of Data Structures and Algorithms of the candidates. The main reason behind this is Data Structures and Algorithms improves the problem-solving ability of a candidate to a great extent.

•This course has video lectures of all the topics from which one can easily learn. I prefer learning from video rather than books and notes. I know books and notes and thesis have their own significance but still video lecture or face to face lectures make it easy to understand faster as we are involved Practically.

•It has 200 algorithmic coding problems with video explained solutions.

•It has track based learning and weekly assessment to test my skills.

•It was a great opportunity for me to invest my time in learning instead of wasting it here and there during my summer break.

•This was a lifetime accessible course which I can use to learn even after my training whenever I want to revise.

**BIBLIOGRAPHY OR REFRENCES**

* 1. GeeksforGeeks. (n.d.). *Sudoku Solver* [Java Implementation]. Retrieved from https://www.geeksforgeeks.org/sudoku-backtracking-7/
* Provides a detailed guide and code implementation of the backtracking algorithm used in solving Sudoku puzzles.
  1. LeetCode. (n.d.). *Sudoku Solver* [Problem and Solutions]. Retrieved from https://leetcode.com/problems/sudoku-solver/
* Offers various approaches to solving Sudoku puzzles, including optimized backtracking algorithms and their applications.
  1. Kothari, A. (2020). *Understanding Backtracking with Sudoku*. FreeCodeCamp. Retrieved from https://www.freecodecamp.org/news/backtracking-algorithm-to-solve-sudoku-puzzle/
* It explains how the backtracking algorithm works with a practical implementation example in Python.