LOVELY PROFESSIONAL UNIVERSITY

Six week Summer Training Report

On

DSA Live with Project Building

Submitted By
Prakash Kumar

Registration Number
12018946

“ B.Tech. in Computer Science and Engineering ”

Under the Guidance of

Mr. Sandeep Jain

Founder & CEO, GeeksforGeeks

“School of Computer Science and Engineering”
Lovely Professional University
Phagwara, Punjab

(June – July 2022)

Declaration

I hereby declare that I have completed my six weeks summer training at GeekforGeeks - 5th Floor, A-118,Sector-136, Noida, Uttar Pradesh – 201305 from 6th of June 2022 to 21st of July 2022 under the guidance of Mr. Sandeep Jain. I have declare that I have worked with full dedication during these six weeks of training and my learning outcomes fulfill the requirements of training for the award of degree of B.Tech in Computer Science and Engineering, Lovely Professional University, Phagwara.

Prakash Kumar.

Name of Student : Prakash Kumar
Registration no : 12018946

Date : 22nd of July 2022

Acknowledgement

I would like to express my special thanks of gratitude to the teacher and instructor of the course “ DSA With Live Project Building ” from GeeksForGeeks Mr. Ishjot Singh Ahluwalia who provide me the golden opportunity to learn a new technology from home.

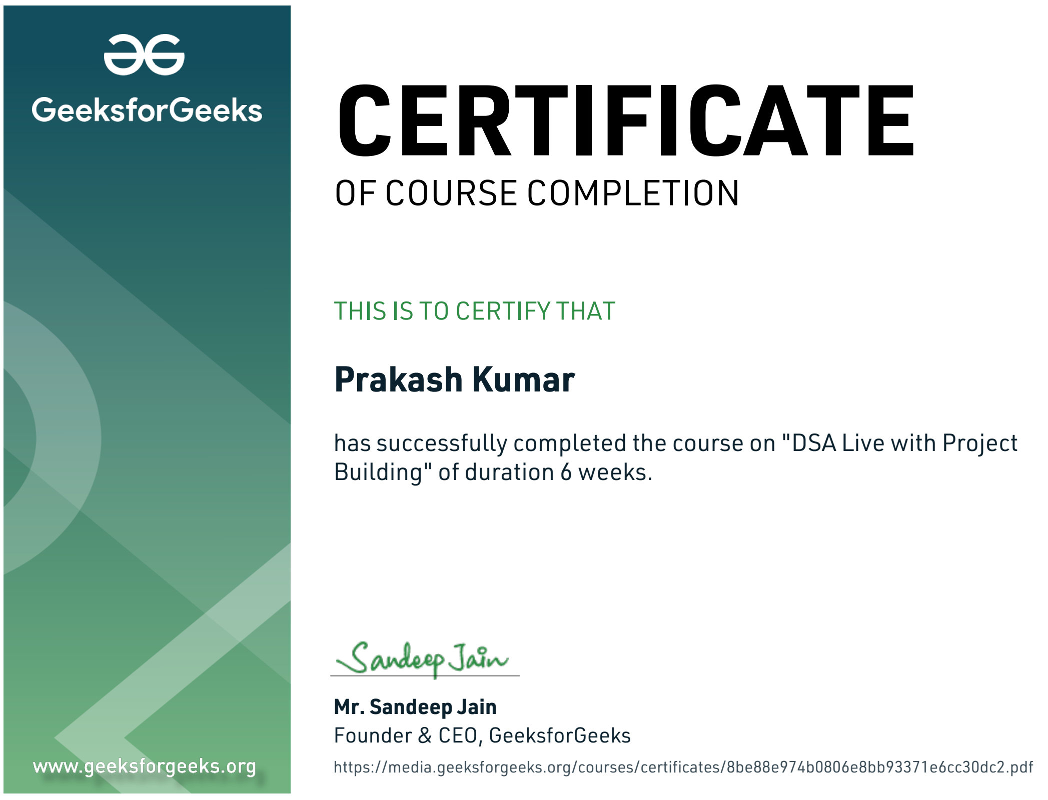
I would like to express my gratitude to my DSA faculty from lovely professional University Mrs. Amandeep Kaur for teaching me so effectively that I didn’t faced any problem while doing this project in association with GeeksForGeeks.

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 least I would like to thank my all classmates who have helped me a lot throughout this course

Summer Training Certificate



GeeksforGeeks

www.geeksforgeeks.org

CERTIFICATE

OF COURSE COMPLETION

THIS IS TO CERTIFY THAT

Prakash Kumar

has successfully completed the course on "DSA Live with Project Building" of duration 6 weeks.

Sandeep Jain

Mr. Sandeep Jain
Founder & CEO, GeeksforGeeks

https://media.geeksforgeeks.org/courses/certificates/8be88e974b0806e8bb93371e6cc30dc2.pdf

Table of Content

Format for identification of job profiles and required skill set

Student Name : Prakash Kumar   Reg No. : 12018946
Programme : B.Tech CSE
School : School of Computer Science and Engineering

Name of Organization : GeeksForGeeks
Organization Address : Noida, Utter Pradesh

A) Products developed by the company/Services provided by the company:
1 - Live Classes.
2 - MCQs and Coding Question Practice.
3 - Project Building using different DSA concepts.

B) Table for Job profile and Skill set identified

S No.	Job Profiles identified	Skill set required
1	Python Programmer	Python Language
2	Game Developer	GUI Knowledge using Python
3	Web Developer	Algorithm and programming for web Development
4	Data Analyst	Understanding of Algorithm

C) Skill Set attained during Summer Internship:

1)Problem Solving Technique
2)Deep knowledge of importance of DSA
3)Online Game Development

INTRODUCTION

Data Structure is the study related to data. Data Structure is a group of data elements that provides the easiest way to store and perform different actions on the data of the computer. A data structure is a particular way of organizing data in a computer so that it can be used effectively. The basic idea behind using DSA is to reduce the space and time complexities of different tasks.

The choice of good data structure makes it possible to perform a variety of critical operations efficiently. A data structure has also defined an instance of ADT(Abstract Data Type) which is formally defined as a triplet[D,F,A]

- D: Set of the domain
- F: the set of operations
- A: the set of axioms.

There are two types of Data structure mainly classified as:

1. Linear Data Structure
2. Non-Linear Data Structure

- Linear Data Structure:-

A linear Data structure is a structure in which the elements are stored sequentially, and the elements are connected to the previous and the next element. In a linear data structure, the data elements connect to each other sequentially. A user can transverse each element through a single run.
As the elements are stored sequentially, so they can be traversed or accessed in single run. The implementation of linear data structures is easier as the elements are sequentially organized in memory.
The data elements in an array are traversed one after another and can access only one element at a time.
There are different types of linear data structures, such as Array, Queue, Stack, Linked List.
Let’s discuss all these Linear data structures in detail:-

- Array: An array consists of data elements of a same data type. For example, if we want to store the roll numbers of 10 students, so instead of creating 10 integer type variables, we will create an array having size 10. Therefore, we can say that an array saves a lot of memory and reduces the length of the code.
- Stack: It is linear data structure that uses the LIFO (Last In-First Out) rule in which the data added last will be removed first. The addition of data element in a stack is known as a push operation, and the deletion of data element form the list is known as pop operation.

Mainly the following three basic operations are performed in the stack:
Initialize: Make a stack empty.
Push: Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.
Pop: Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.
Peek or Top : Returns top element of the stack.
isEmpty : Returns true if the stack is empty, else false.

- Queue: It is a data structure that uses the FIFO rule (First In-First Out). In this rule, the element which is added first will be removed first. There are two terms used in the queue front end and rear. The insertion operation performed at the back end is known ad enqueue, and the deletion operation performed at the front end is known as dequeue.

Mainly the following four basic operations are performed on queue:
Enqueue: Adds an item to the queue. If the queue is full, then it is said to be an Overflow condition.
Dequeue: Removes an item from the queue. The items are popped in the same order in which they are pushed. If the queue is empty, then it is said to be an Underflow condition.
Front: Get the front item from the queue.
Rear: Get the last item from the queue.

- Linked list: It is a collection of nodes that are made up of two parts, i.e., data element and reference to the next node in the sequence.

- **Non-Linear Data Structure:**

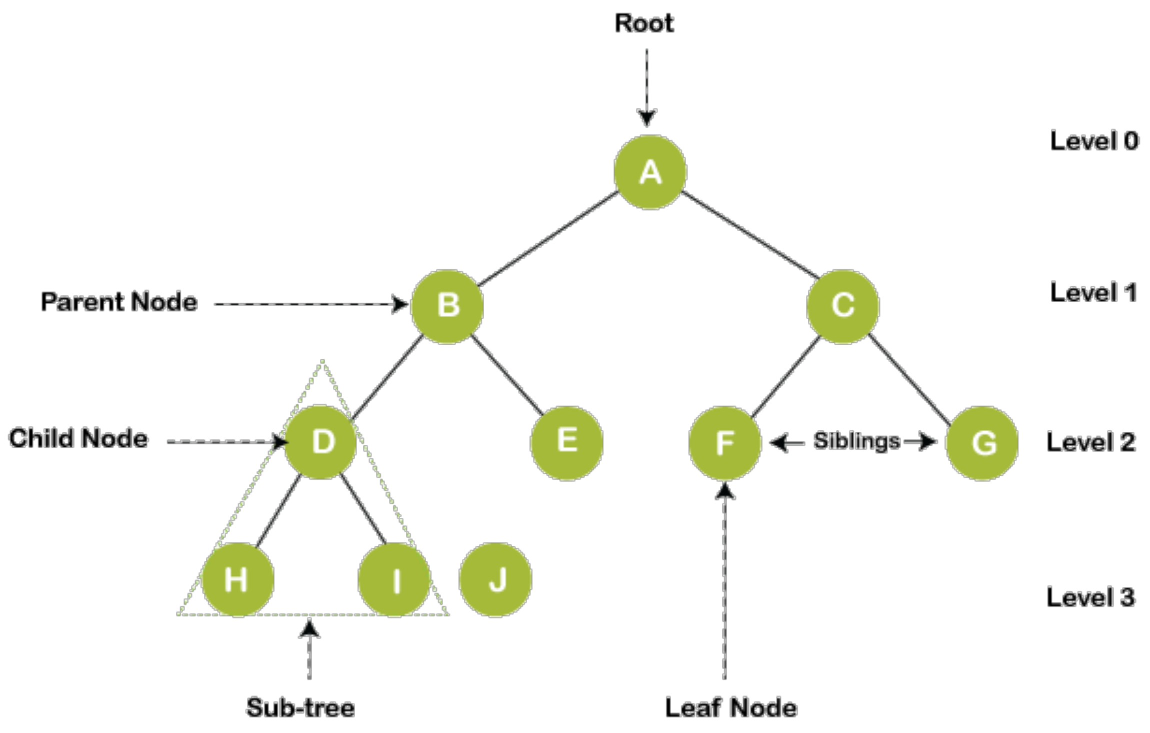
It is a form of data structure where the data elements don't stay arranged linearly or sequentially. Since the data structure is non-linear, it does not involve a single level. Therefore, a user can't traverse all of its elements in a single run. The non-linear data structure is not very easy to implement as compared to the linear data structure. The utilization of computer memory is more efficient in this case.
Examples of Non-Linear Data structure are Trees and Graphs.
In a non-linear data structure, the data elements connect to each other hierarchically. Thus, they are present at various levels.
The non-linear data structures are comparatively difficult to implement and understand as compared to the linear data structures.
Non-linear data structure's time complexity often remains the same with an increase in its input size.
One can find all the data elements at multiple levels in a non-linear data structure.

Let’s Discuss the types of Non-Linear data structure in details:-

1. Trees:- It is a non-linear data structure that consists of various linked nodes. It has a hierarchical tree structure that forms a parent-child relationship. A tree consists of root node, parent node, child node, leaf node, sub-tree, siblings. A tree consists of different levels starting from zero level as an top.

Root node:- The first node of the tree is called the root. A tree can have only one root node.
Parent node:- the node which is a predecessor of any node is called as parent node or the node adjacent to a given node on the path to the root node.
Child node:- The node below a given node connected by its edge downward is called its child node.
Leaf node:- the node which does not have a child is called as leaf node. In simple words, a leaf is a node with no child. In a tree data structure, the leaf nodes are also called as External Nodes.

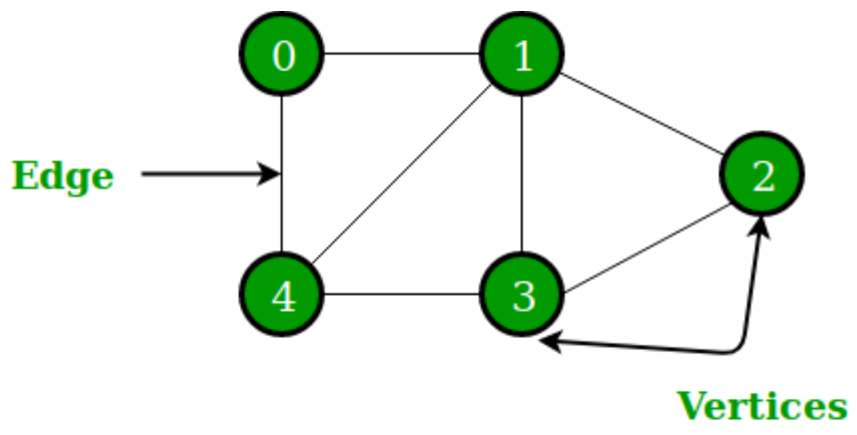
Sub-tree:- any node in a tree and its descendants.
Siblings:- the nodes with the same parent are called Sibling nodes.



- **Graph:-**

A graph is a non-linear data structure that has a finite number of vertices and edges, and these edges are used to connect the vertices. The vertices are used to store the data elements, while the edges represent the

relationship between the vertices. A graph is used in various real-world problems like telephone networks, circuit networks, social networks like LinkedIn, Facebook.
A Graph consists of a finite set of vertices(or nodes) and a set of Edges that connect a pair of nodes.



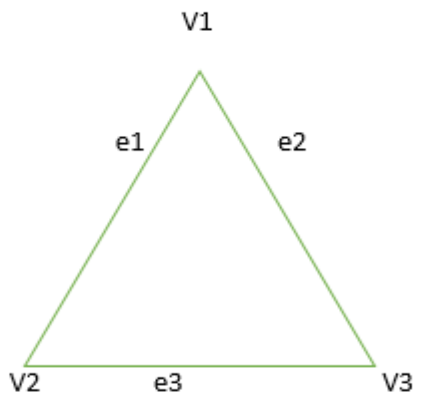
- **Vertices:** Vertices are the fundamental units of the graph.
- Sometimes, vertices are also known as vertex or nodes.
- Every node/vertex can be labelled or unlabelled.

- **Edges:** Edges are drawn or used to connect two nodes of the graph. It can be ordered pair of nodes in a directed graph.
- Edges can connect any two nodes in any possible way.
- Sometimes, edges are also known as arcs.
- Every edge can be labelled or unlabelled.

There are various types of graphs in DSA:-

- **Finite graph:-** These graphs contains finite number of vertices and edges respectively.
- **Infinite graph:-** These graphs contains infinite number of vertices as well as edges.
- **Trivial graph:-** If a finite graph contains only one vertex and no edes.
- **Simple graph:-** A simple graph is a graph that does not contain more than one edge between the pair of vertices.
- **Null graph:-** A graph of order n and size zero is a graph where there are only isolated vertices with no edges connecting any pair of vertices.
- **Pseudo graph:-** A graph G with a self-loop and some multiple edges is called a pseudo graph.

- **Regular graph:-** A simple graph is said to be regular if all vertices of graph G are of equal degree. All complete graphs are regular but vice versa is not possible.
- **Bipartite graph:-** A graph G = (V, E) is said to be a bipartite graph if its vertex set V(G) can be partitioned into two non-empty disjoint subsets.
- **Labelled Graph:-** The vertices and edges of a graph are labelled with name, date, or weight. It is also called as weighted graph.
- **Cyclic graph:-** A graph G consisting of n vertices and n> = 3 that is V1, V2, V3- – – – Vn and edges (V1, V2), (V2, V3), (V3, V4)- – – – (Vn, V1) are called cyclic graph.



Advantage of graphs are as follows:-

- By using graphs we can easily find the shortest path, neighbours of the nodes, and many more.
- Graphs are used to implement algorithms like DFS and BFS.
- It is used to find minimum spanning tree which has many practical applications.
- It helps in organizing data.
- Because of its non-linear structure, helps in understanding complex problems and their visualization.

Disadvantage of graphs are as follows:-

- Graphs use lots of pointers which can be complex to handle.
- It can have large memory complexity.
- If the graph is represented with an adjacency matrix then it does not allow parallel edges and multiplication of the graph is also difficult.

Asymptotic notations:- Asymptotic notations are the mathematical notations used to describe the running time of an algorithm when the input tends towards a particular value or a limiting value.
There are mainly three asymptotic notations:

- Big-O notation(it gives the worst-case complexity of an algorithm)
- Omega notation(it provides the best case complexity of an algorithm)
- Theta notation(it is used for analysing the average-case complexity of an algorithm).

Stack:- A stack is a linear data structure in which elements can be inserted and deleted only from one side of the list, called the top. A stack follows the LIFO (Last In First Out) principle.
Queue:- is a linear data structure in which elements can be inserted only from one side of the list called rear, and the elements can be deleted only from the other side called the front. The queue data structure follows the FIFO (First In First Out) principle.

TECHNOLOGY LEARNT

- 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, and A direct way of calculating the order of growth

• **Asymptotic Notations -** Best, Average and Worst-case explanation through a program.

• **Big O Notation -** Graphical and mathematical explanation. Calculation. Applications at Linear Search

• **Omega Notation -** Graphical and mathematical explanation. o Calculation.

• **Theta Notation -** Graphical and mathematical explanation. o 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 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

STACK

- **Understanding the Stack data structure**
- **Applications of Stack**
- **Implementation of Stack in Array and Linked List**

QUEUE

- **Introduction and Application**
- **Implementation of the queue using array and LinkedList**
  - In C++ STL
  - In Java
  - Stack using queue

DEQUE

- **Introduction and Application**
- **Implementation**

  - In C++ STL
  - In Java
- **Problems (With Video Solutions)**
  - Maximums of all subarrays of size k o Array Deque in Java
  - Design a DS with min max operations

TREE
• **Introduction**

  - Tree
  - Application
  - Binary Tree
  - Tree Traversal

• **Implementation of:**

  - In order Traversal
  - Pre order Traversal
  - Post order Traversal
  - Level Order Traversal (Line by Line) o 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

- **Heap Sort**
- **Priority Queue in C++**
- **Priority Queue in Java**

GRAPH

- **Introduction to Graph**
- **Graph Representation**

  - Adjacency Matrix
  - Adjacency List in CPP and Java o Adjacency Matrix VS List

- **Breadth-First Search** o

  - 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**

GREEDY

- **Introduction**
- **Activity Selection Problem**
- **Fractional Knapsack**
- **Job Sequencing Problem**

BACKTRACKING

- **Concepts of Backtracking**
- **Rat In a Maze**
- **N Queen Problem**

DYNAMIC PROGRAMMING

- **Introduction**
- **Dynamic Programming**
    - Memorization
    - Tabulation

TREE

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

SEGMENT TREE

- **Introduction**
- **Construction**
- **Range Query**
- **Update Query**

DISJOINT SET

- **Introduction**
- **Find and Union Operations**
- **Union by Rank**
- **Path Compression**
- **Kruskal's Algorithm**

MINI – PROJECT

SUDUKO SOLVER
(Using Graphical User Interface)

Language Used:- PYTHON Programming Language

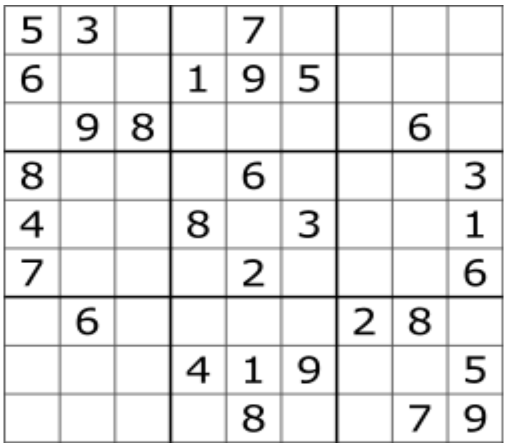
About

Python is a computer programming language often used to build websites and software, automate tasks, and conduct data analysis.
Python is a general-purpose language, meaning it can be used to create a variety of different programs and isn't specialized for any specific problems. Python is a high-level, general-purpose, interpreted object-oriented programming language.
Python has five standard Data Types:

- Numbers(contains numerical inputs)
- String(contains alphabetical inputs)
- List(Lists are mutable datatypes)
- Tuple(Tuples are immutable datatypes)
- Dictionary(contains link and info of datatypes)

SUDUKO ABSTRACT

- Sudoku also known as number place is basically a logical, combinational puzzle in which there is a 9 x 9 grid which are further divided into 3 x 3 sub-grids. Main objective of the game is to fill each of 9 the sub-grid, row and column contains digits from 1 to 9. The puzzle provided have been completed partially such that there is only one solution to the puzzle. The image attached below shows a typical Sudoku puzzle grid.



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

- Modern version of Sudoku is believed to be designed by Howard Grans of Indiana, USA. A scientist Jeremy Grabbed found that solving Sudoku involved an area of cognition called working memory. This verified that solving sudoku puzzle in routine can improve working memory.
- This project will solve sudoku automatically using backtracking.

Table Of Content

INTRODUCTION

This project has been done as part of my course Summer Training Internship Programme. Whole project is completed in almost 50 days. As we have use basic knowledge of GUI in Python, we decided to make a Sudoku puzzle game which looks quite attractive, and it has been proved that solving these puzzles will help us to increase our working memory. During this project we have gained experience of how to use GUI to make user friendly programs. This project gives us an insight to different aspects of GUI in python programming.

Throughout the document we refer the whole puzzle as grid and a 3 x 3 sub grid as a block and individual box containing/will be containing a number is referred as cell.

LIBRARY USED

Pygame: - Pygame is a cross platform used for creating games using graphics and sound library. It used for creating various small games for different operating systems using high level languages like python.

USEFULL CONCEPTS:

Backtracking:-

Backtracking is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time (by time, here, is referred to the time elapsed till reaching any level of the search tree). consider the Sudoku solving Problem, we try filling digits one by one. Whenever we find that current digit cannot lead to a solution, we remove it (backtrack) and try next digit. This is better than naive approach (generating all possible combinations of digits and then trying every combination one by one) as it drops a set of permutations whenever it backtracks.

SOLVING METHODS

Brute force solving method
This is the simplest way to solve Sudoku puzzle. In this method we fill each square blank with a number from 1 to 9 until a valid solution is found. Now I will describe 3 different ways to implement this brute force method.

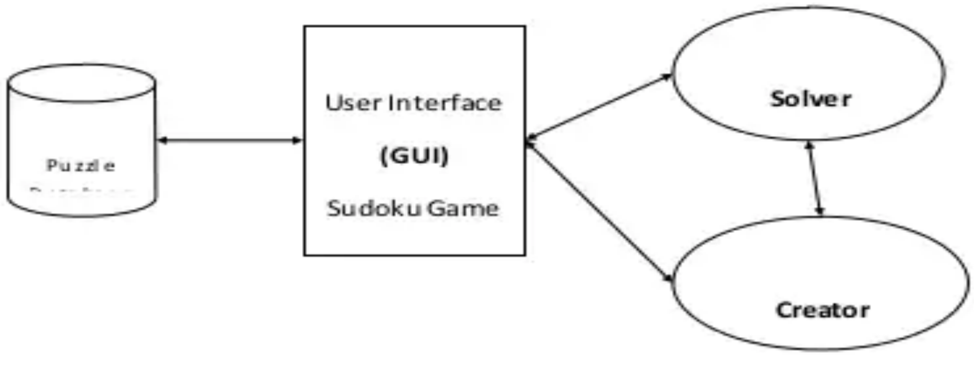
Way 1: -
Mostly a naïve solver starts at the top left square and moves from left to right, top to bottom, filling each blank box with a number from 1 to 9 until the grid is valid and continue froward again.

Way 2: -
This way uses the concept of domains. Each square in the grid has a domain of 1 to 9 that is reduced according to numbers already present in the sub-grid, row, and column. The solver acts in the same way as the previous implementation except that it restricts the domain of the grid before it starts, only using numbers present in the initial domain. While the domain restrictions require some computation, it should result in significantly less backtracking to find the solution of a grid.

RESEARCH METHODOLGY

The basic requirement was to create an interactive and simple sudoku game, where user will try their hand to solve the puzzle. Considering this requirement, a simple 9 x 9 grid layout has been made and discussed. With discussion we across the following questions about the project development: -

- Type of layout
- Interaction from mouse or Keyboard
- Resizable window or not
- Kind of help/support



IMPLEMENTATION TECHNIQUE

For this project we chose Python as the programming language as this was the basic requirement of the course. Prolong would probably have been the best option for designing this puzzle but since this course requires python as language and the fact that we haven’t done any work in prolong we went with python. Languages like prolong can perform some of the solving work automatically so they are preferred.

IMPLEMENTATION STEPS:-

1. Fill the pygame window with sudoku board i.e., construct a 9x9 grid.
2. Fill the board with default numbers.
3. Assign a specific key for each operations and listen it.
4. Integrate the backtracking algorithm into it.
5. Use set of colors to visualize auto solving.

Instructions

1. Press ‘Enter’ to auto solve and visualize.
2. To play game manually,
   Place the cursor in any cell you want and enter the number.
3. At any point, press enter to solve automatically.
4. Press R to empty the sudoku grid

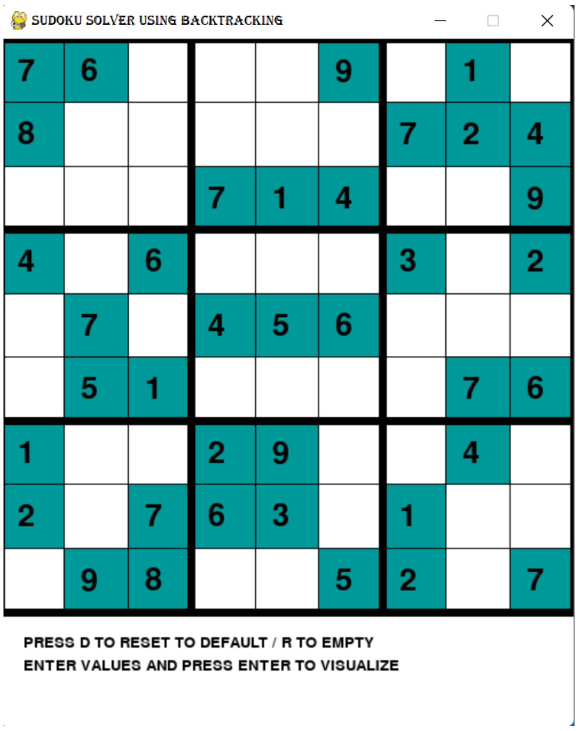
Implementation

```
# import pygame library-------->it is library used to make games in python
import pygame

# initialise the pygame font------------>initialise the font mdule
pygame.font.init()

# Total window-------------->helps to create window named screen
screen = pygame.display.set_mode((500, 600))

# Title and Icon------------>title named
pygame.display.set_caption("SUDOKU SOLVER USING BACKTRACKING")


x = 0
y = 0
dif = 500 / 9
val = 0
# Default Sudoku Board.
grid =[
        [7, 8, 0, 4, 0, 0, 1, 2, 0],
        [6, 0, 0, 0, 7, 5, 0, 0, 9],
        [0, 0, 0, 6, 0, 1, 0, 7, 8],
        [0, 0, 7, 0, 4, 0, 2, 6, 0],
        [0, 0, 1, 0, 5, 0, 9, 3, 0],
        [9, 0, 4, 0, 6, 0, 0, 0, 5],
        [0, 7, 0, 3, 0, 0, 0, 1, 2],
        [1, 2, 0, 0, 0, 7, 4, 0, 0],
        [0, 4, 9, 2, 0, 6, 0, 0, 7]
    ]

# Load test fonts for future use
font1 = pygame.font.SysFont("comicsans", 40)
font2 = pygame.font.SysFont("comicsans", 20)
def get_cord(pos):
    global x
    x = pos[0]//dif
    global y
    y = pos[1]//dif

# Highlight the cell selected
def draw_box():
    for i in range(2):
        pygame.draw.line(screen, (255, 0, 0), (x * dif-3, (y + i)*dif), (x *
dif + dif + 3, (y + i)*dif), 14)  #block selector color change,window name
,color,start position,end position,thick
```

```
        pygame.draw.line(screen, (255, 0, 0), ( (x + i)* dif, y * dif ),  ((x
+ i) * dif, y * dif + dif), 14)

# Function to draw required lines for making Sudoku grid
def draw():
    # Draw the lines

    for i in range (9):
        for j in range (9):
            if grid[i][j]!= 0:

                # Fill blue color in already numbered grid
                pygame.draw.rect(screen, (0, 153, 153), (i * dif, j * dif, dif
+ 1, dif + 1)) #rect ===rectangle

                # Fill grid with default numbers specified
                text1 = font1.render(str(grid[i][j]), 1, (0, 0, 0))
                screen.blit(text1, (i * dif + 15, j * dif + 15))            #,
(SCREEN_WIDTH/2, SCREEN_HEIGHT/2))    blit --->block transfer
    # Draw lines horizontally and verticallyto form grid
    for i in range(10):
        if i % 3 == 0 :
            thick = 7
        else:
            thick = 1
        pygame.draw.line(screen, (0, 0, 0), (0, i * dif), (500, i * dif),
thick)
        pygame.draw.line(screen, (0, 0, 0), (i * dif, 0), (i * dif, 500),
thick)

# Fill value entered in cell
def draw_val(val):
    text1 = font1.render(str(val), 1, (0, 0, 0))
    screen.blit(text1, (x * dif + 15, y * dif + 15))

# Raise error when wrong value entered
def raise_error1():
    text1 = font1.render("WRONG !!!", 1, (0, 0, 0))
    screen. BLit(text1, (20, 570))
def raise_error2():
    text1 = font1.render("Wrong !!! Not a valid Key", 1, (0, 0, 0))
    screen.blit(text1, (20, 570))

# Check if the value entered in board is valid
def valid(m, i, j, val):
    for it in range(9):
        if m[i][it]== val:
            return False
        if m[it][j]== val:
            return False
    it = i//3
    jt = j//3
    for i in range(it * 3, it * 3 + 3):
        for j in range (jt * 3, jt * 3 + 3):
            if m[i][j]== val:
                return False
    return True

# Solves the sudoku board using Backtracking Algorithm
def solve(grid, i, j):

    while grid[i][j]!= 0:
        if i<8:
            i+= 1
        elif i == 8 and j<8:
            i = 0
            j+= 1
        elif i == 8 and j == 8:
            return True
    pygame.event.pump()
    for it in range(1, 10):
        if valid(grid, i, j, it)== True:
            grid[i][j]= it
            global x, y
            x = i
            y = j
            # white color background\
            screen.fill((255, 255, 255))
            draw()
            draw_box()
            pygame.display.update()
            pygame.time.delay(20)
            if solve(grid, i, j)== 1:
                return True
            else:
                grid[i][j]= 0
            # white color background\
            screen.fill((255, 255, 255))

            draw()
            draw_box()
            pygame.display.update()
            pygame.time.delay(50)
    return False

# Display instruction for the game
```

```python
def instruction():
    text1 = font2.render("PRESS D TO RESET TO DEFAULT / R TO EMPTY", 1, (0, 0,
0))
    text2 = font2.render("ENTER VALUES AND PRESS ENTER TO VISUALIZE", 1, (0,
0, 0))
    screen.blit(text1, (20, 520))
    screen.blit(text2, (20, 540))

# Display options when solved
def result():
    text1 = font1.render("FINISHED PRESS R or D", 1, (0, 0, 0))
    screen.blit(text1, (20, 570))
run = True
flag1 = 0
flag2 = 0
rs = 0
error = 0
# The loop thats keep the window running
while run:

    # White color background
    screen.fill((255,255,255))
    # Loop through the events stored in event.get()
    for event in pygame.event.get():
        # Quit the game window
        if event.type == pygame.QUIT:
            run = False
        # Get the mouse position to insert number
        if event.type == pygame.MOUSEBUTTONDOWN:
            flag1 = 1
            pos = pygame.mouse.get_pos()
            get_cord(pos)
        # Get the number to be inserted if key pressed
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                x-= 1
                flag1 = 1
            if event.key == pygame.K_RIGHT:
                x+= 1
                flag1 = 1
            if event.key == pygame.K_UP:
                y-= 1
                flag1 = 1
            if event.key == pygame.K_DOWN:
                y+= 1
                flag1 = 1
            if event.key == pygame.K_1:
                val = 1
            if event.key == pygame.K_2:
                val = 2
            if event.key == pygame.K_3:
                val = 3
            if event.key == pygame.K_4:
                val = 4
            if event.key == pygame.K_5:
                val = 5
            if event.key == pygame.K_6:
                val = 6
            if event.key == pygame.K_7:
                val = 7
            if event.key == pygame.K_8:
                val = 8
            if event.key == pygame.K_9:
                val = 9
            if event.key == pygame.K_RETURN:
                flag2 = 1
            # If R pressed clear the sudoku board
            if event.key == pygame.K_r:
                rs = 0
                error = 0
                flag2 = 0
                grid =[
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0]
                ]
            # If D is pressed reset the board to default
            if event.key == pygame.K_d:
                rs = 0
                error = 0
                flag2 = 0
                grid =[
                    [7, 8, 0, 4, 0, 0, 1, 2, 0],
                    [6, 0, 0, 0, 7, 5, 0, 0, 9],
                    [0, 0, 0, 6, 0, 1, 0, 7, 8],
                    [0, 0, 7, 0, 4, 0, 2, 6, 0],
                    [0, 0, 1, 0, 5, 0, 9, 3, 0],
                    [9, 0, 4, 0, 6, 0, 0, 0, 5],
                    [0, 7, 0, 3, 0, 0, 0, 1, 2],
                    [1, 2, 0, 0, 0, 7, 4, 0, 0],
                    [0, 4, 9, 2, 0, 6, 0, 0, 7]
                ]
    if flag2 == 1:
        if solve(grid, 0, 0)== False:
            error = 1
        else:
            rs = 1
        flag2 = 0
    if val != 0:
        draw_val(val)
        # print(x)
        # print(y)
        if valid(grid, int(x), int(y), val)== True:
            grid[int(x)][int(y)]= val
            flag1 = 0
        else:
            grid[int(x)][int(y)]= 0
            raise_error2()
        val = 0

    if error == 1:
        raise_error1()
    if rs == 1:
        result()
    draw()
    if flag1 == 1:
        draw_box()
    instruction()

    # Update window
    pygame.display.update()

# Quit pygame window---------->attribute to quit game
pygame.quit()
```

Output :

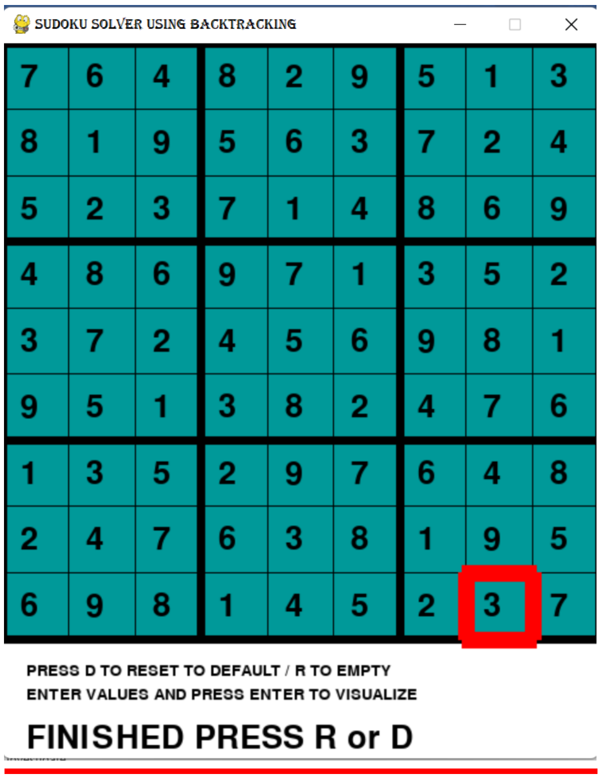


SUDOKU SOLVER USING BACKTRACKING

7	6				9		1	
8						7	2	4
			7	1	4			9
4		6				3		2
	7		4	5	6			
	5	1					7	6
1			2	9			4	
2		7	6	3		1		
	9	8			5	2		7

PRESS D TO RESET TO DEFAULT / R TO EMPTY
ENTER VALUES AND PRESS ENTER TO VISUALIZE

Solved output :



SUDOKU SOLVER USING BACKTRACKING

7	6	4	8	2	9	5	1	3
8	1	9	5	6	3	7	2	4
5	2	3	7	1	4	8	6	9
4	8	6	9	7	1	3	5	2
3	7	2	4	5	6	9	8	1
9	5	1	3	8	2	4	7	6
1	3	5	2	9	7	6	4	8
2	4	7	6	3	8	1	9	5
6	9	8	1	4	5	2	3	7

PRESS D TO RESET TO DEFAULT / R TO EMPTY
ENTER VALUES AND PRESS ENTER TO VISUALIZE

FINISHED PRESS R or D

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 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

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

Knowledge of data structures like Hash Tables, Trees, Tries, Graphs, and various 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.

BIBLIOGRAPHY

- GeeksForGeeks website
- GeeksForGeeks DSA self-paced Course
- Google