Annexure-I

DSA Self-paced

GeeksForGeeks

A training report

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

Bachelor of Technology

(Computer Science and Engineering)

Submitted to

LOVELY PROFESSIONAL UNIVERSITY

PHAGWARA, PUNJAB

From May 4st , 2022 to

July 3th, 2022

SUBMITTED BY

Name: Piyush Sinha

Registration No. : 12004180

Piyush Sinha

Signature

Annexure-II: Student Declaration

<u>To whom so ever it may concern</u>

I, **Piyush Sinha, 12004180,** hereby declare that the work done by me on "**DSA**" from **May 2022** to **July 2022** is a record of original work for the partial fulfillment of the requirements for the award of the degree, **Bachelor of Technology.**

Piyush Sinha (12004180)

Piyush Sinha

Signature

Date: 6th July 2022.

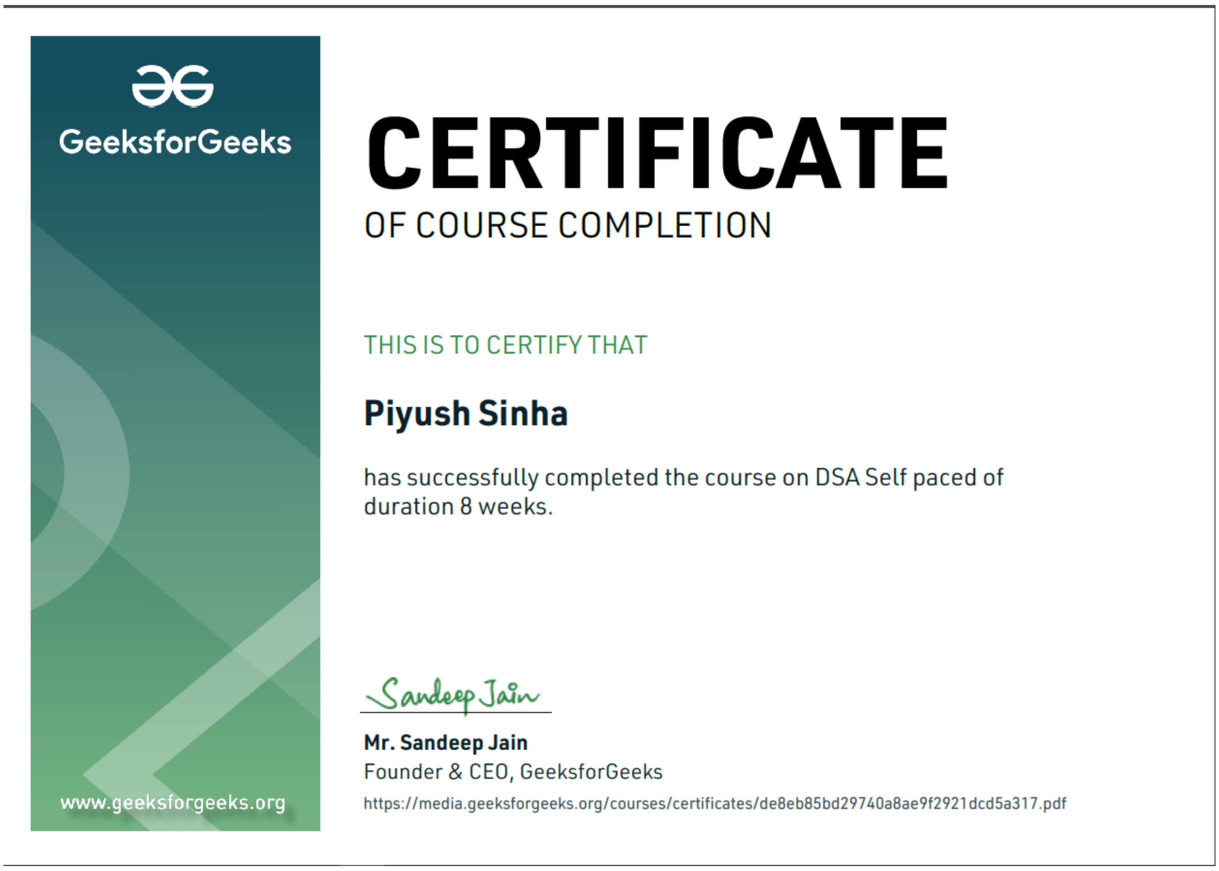
ACKNOWLEDGEMENT

I would like to express my gratitude towards my university as well as GeeksforGeeks for providing me the golden opportunity to do this wonderful summer training regarding DSA Self-Paced, which also helped me in doing a lot of homework and learning. As a result, I came to know about so many new things. So, I am really thankful to them. Moreover, I would like to thank my friends who helped me a lot whenever I got stuck in some problem related to my course. I am really thankful to have such a good support of them as they always have my back whenever I need. I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

Date: 06/07/2022 **Piyush Sinha**

Reg. no: 12004180

Summer Training Certificate



GeeksforGeeks

CERTIFICATE

OF COURSE COMPLETION

THIS IS TO CERTIFY THAT

Piyush Sinha

has successfully completed the course on DSA Self paced of duration 8 weeks.

Sandeep Jain

Mr. Sandeep Jain

Founder & CEO, GeeksforGeeks

https://media.geeksforgeeks.org/courses/certificates/de8eb85bd29740a8ae9f2921dcd5a317.pdf

www.geeksforgeeks.org

Table of Contents

Abbreviation

- **DSA**: Data Structures and Algorithms.
- **BFS** -Best First Search
- **DFS**- Depth First Search

INTRODUCTION

- 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?** Data Structure is a way of collecting and organizing data in such a way that we can perform operations on these data in an effective way. Data Structures is about rendering data elements in terms of some relationship, for better organization and storage. For example, we have some data which has, player's **name** "Virat" and **age** 26. Here "Virat" is of **String** data type and 26 is of **integer** data type.
- **What is Algorithm?** An algorithm is a finite set of instructions or logic, written in order, to accomplish a certain predefined task. Algorithm is not the complete code or program, it is just the core logic(solution) of a problem, which can be expressed either as an informal high-level description as **pseudocode** or using a **flowchart.**

- This course is a complete package that helped me learn Data Structures and Algorithms from basic to an advanced level. The course curriculum has been divided into 8 weeks where one can practice questions & attempt the assessment tests according to his own pace. The course offers me a wealth of programming challenges that will help me to prepare for interviews with top-notch companies like Microsoft, Amazon, Adobe etc.

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.
  - 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.
  - Calculation.
- **Theta 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

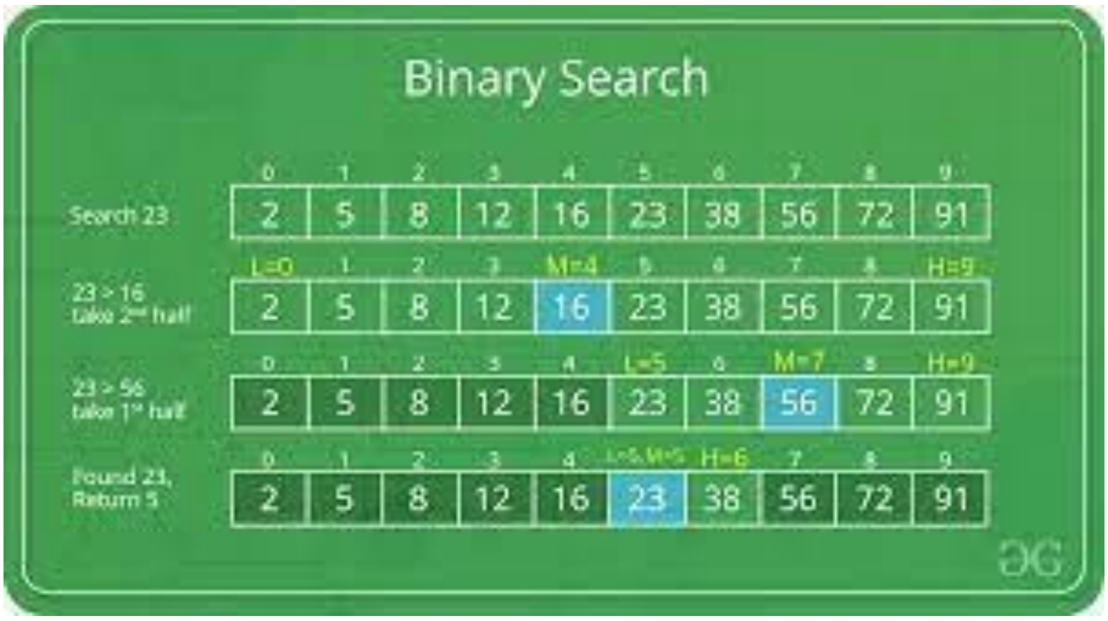
- **Introduction to Recursion**
- **Applications of Recursion**
- **Writing base cases in Recursion**
    - Factorial
    - N-th Fibonacci number

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

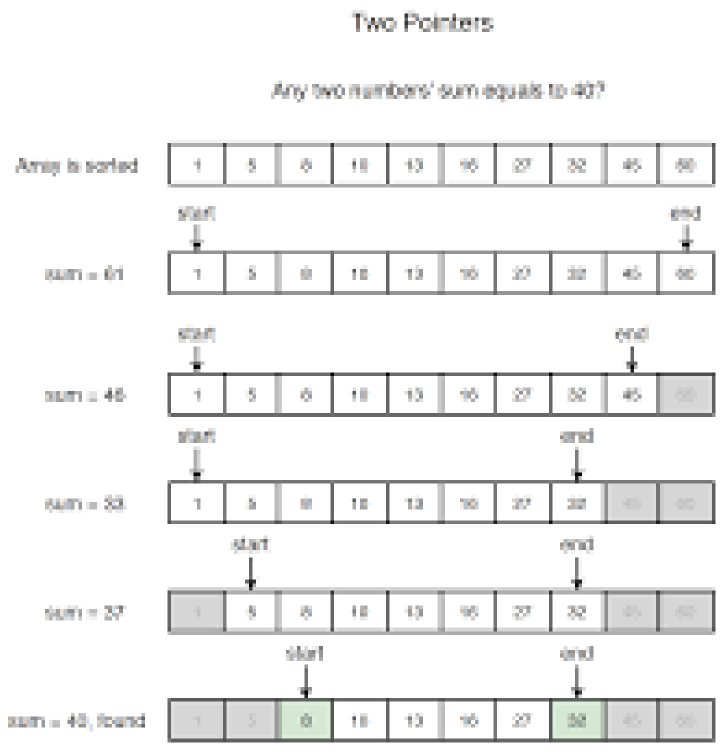
SEARCHING

- **Binary Search Iterative and Recursive**



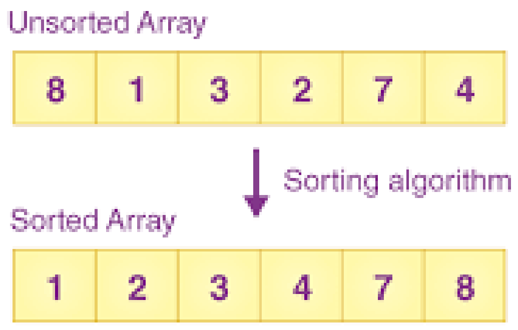
- **Binary Search and various associated problems**

- **Two Pointer Approach Problems**



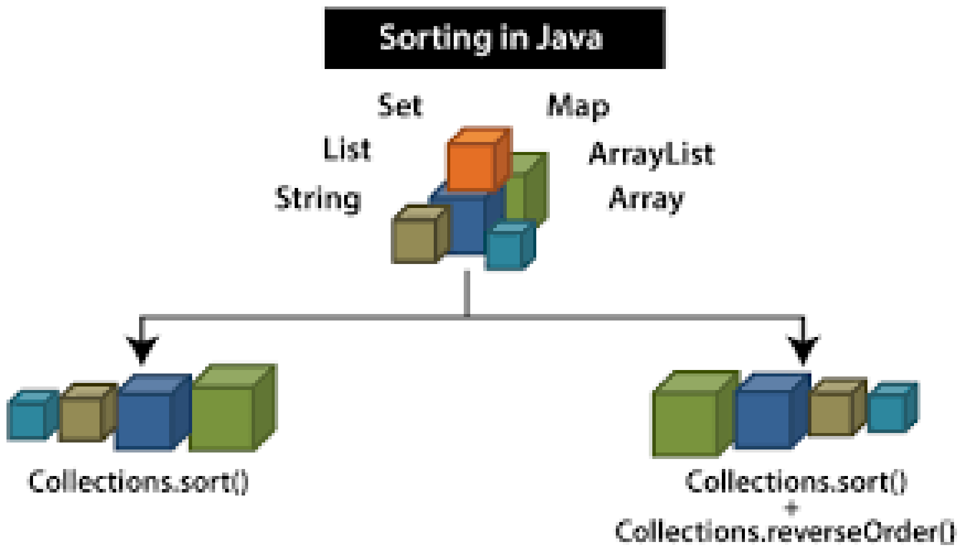
SORTING

- **Sorting in Java**
- **Arrays.sort() in Java**

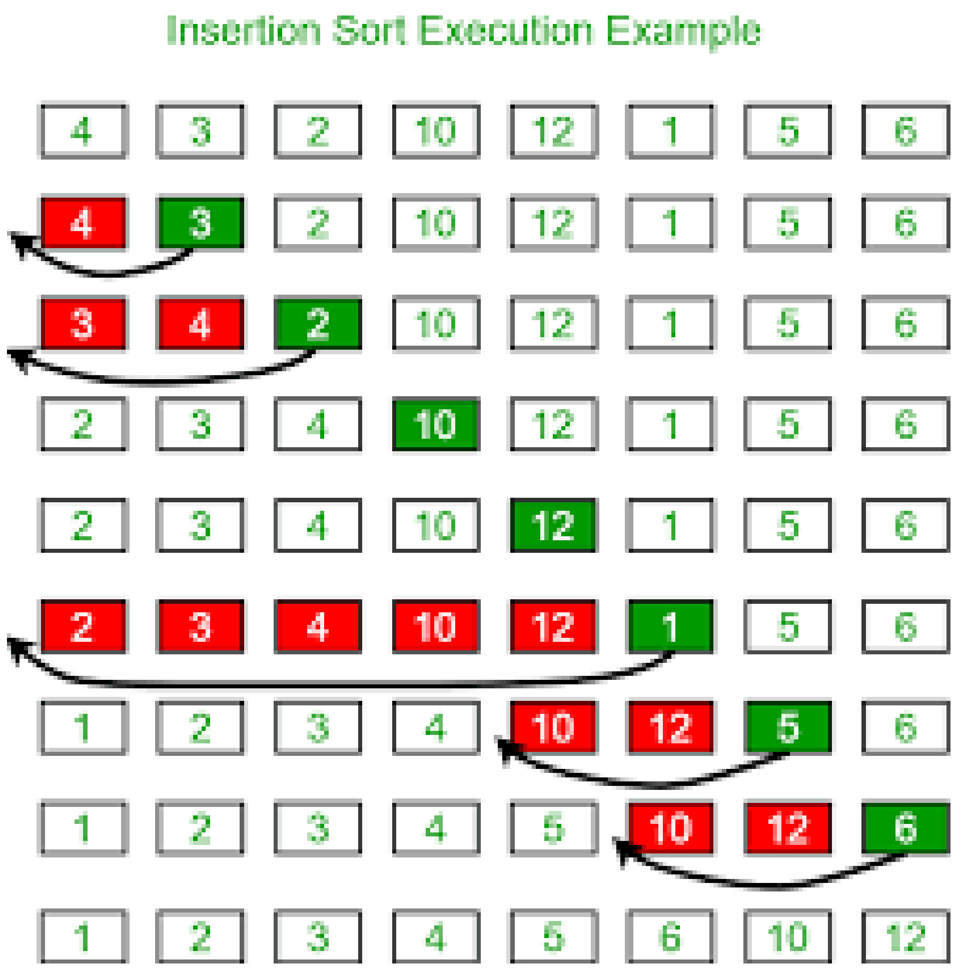


➢

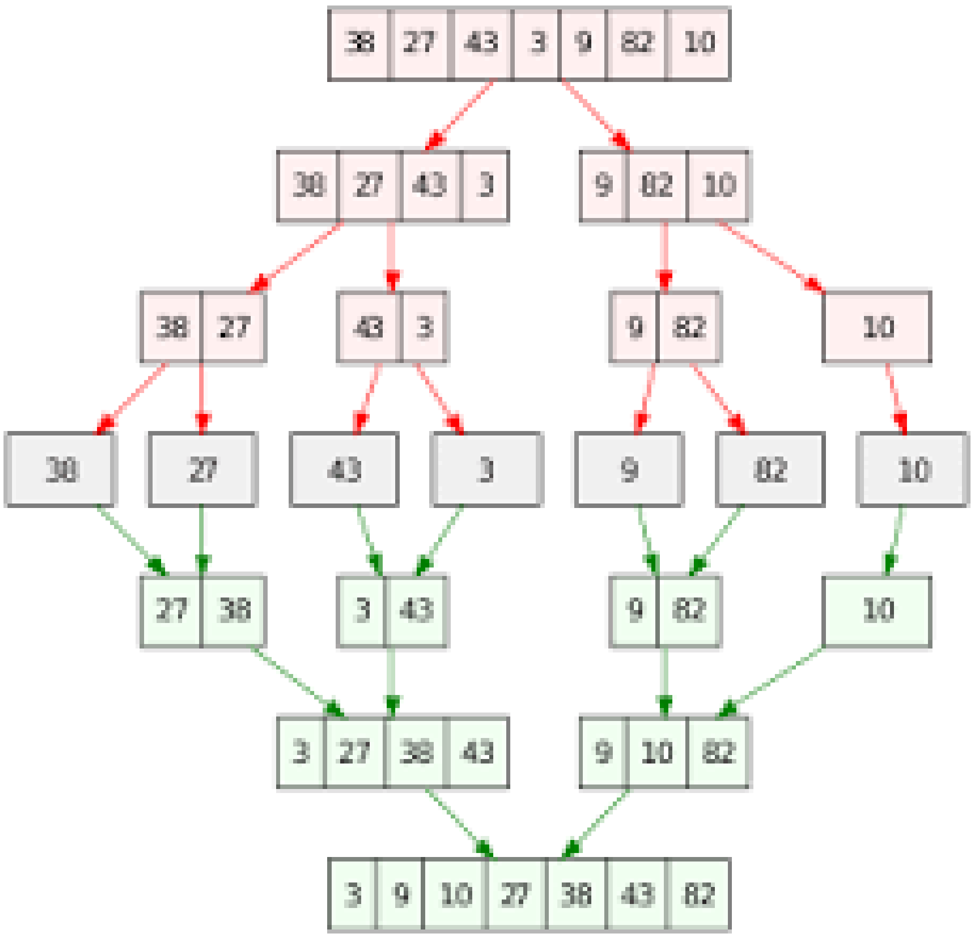
- **Collection.sort() in Java**



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


Insertion Sort Execution Example
4 3 2 10 12 1 5 6
4 3 2 10 12 1 5 6
3 4 2 10 12 1 5 6
2 3 4 10 12 1 5 6
2 3 4 10 12 1 5 6
2 3 4 10 12 1 5 6
1 2 3 4 10 12 5 6
1 2 3 4 5 10 12 6
1 2 3 4 5 6 10 12

Merge sort



- **Quick Sort**
  - Using Lomuto and Hoare
  - Time and Space analysis
  - Choice of Pivot and Worst case
- **Overview of Sorting Algorithms**
- **Implementation of C++ STL sort () function in Arrays and Vectors**
  - Time Complexities

MATRIX

- 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**
  - 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

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

- In C++ STL

- **Problems (With Video Solutions)**
  - Maximums of all subarrays of size k
  - ArrayDeque in Java
  - Design a DS with min max operations.

TREE

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

        - 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.

  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, 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.

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.

BIBLIOGRAPHY

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

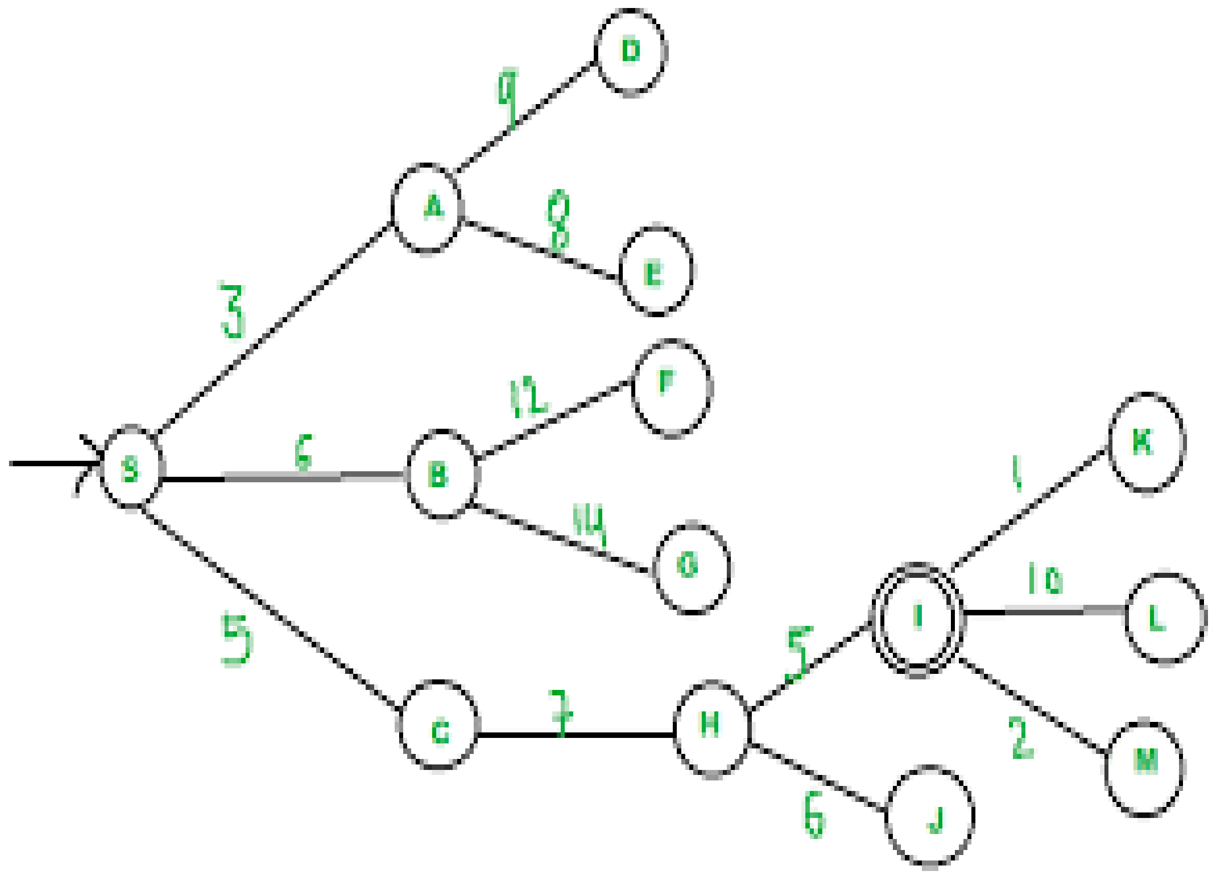
Best First Search

Best first search is a traversal technique that decides which node is to be visited next by checking which node is the most promising one and then check it. For this it uses an evaluation function to decide the traversal.

This best first search technique of tree traversal comes under the category of heuristic search or informed search technique.

The cost of nodes is stored in a priority queue. This makes implementation of best-first search is same as that of breadth First search. We will use the priority queue just like we use a queue for BFS.

```
Step 1 : Create a priorityQueue pqueue.
Step 2 : insert ‘start’ in pqueue : pqueue.insert(start)
Step 3 : delete all elements of pqueue one by one.
   Step 3.1 : if, the element is goal . Exit.
   Step 3.2 : else, traverse neighbours and mark the node
examined.
Step 4 :

 End.
```


S
A
B
C
D
E
F
G
H
I
J
K
L
M
3
6
5
9
8
12
14
7
5
6
1
10
2

PROJECT

Shortest Path finder using Dijkstra's algorithm in python

Technology used

1. **Python**
2. **Pycharm**
3. **Pygame**
4. **tkinter**

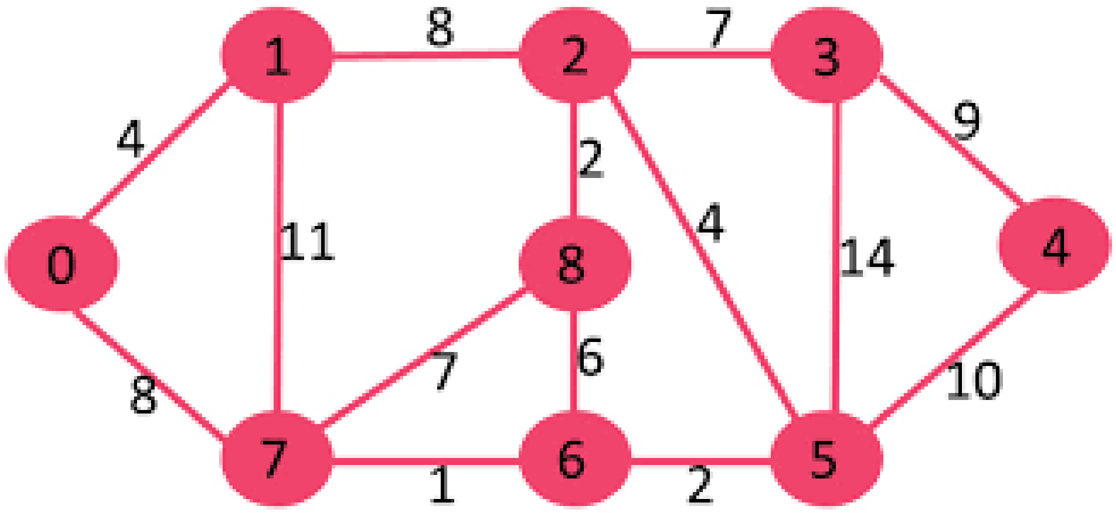
Dijkstra's Algorithm

we want to create a set of visited vertices, to keep track of all of the vertices that have been assigned their correct shortest path. We will also need to set "costs" of all vertices in the graph (lengths of the current shortest path that leads to it).

All of the costs will be set to ***'infinity'*** at the beginning, to make sure that every other cost we may compare it to would be smaller than the starting one. The only exception is the cost of the first, starting vertex - this vertex will have a cost of 0, because it has no path to itself - marked as node `s`.

Then, we repeat two main steps until the graph is traversed (as long as there are vertices without the shortest path assigned to them):

Since this might be a bit difficult to wrap one's head around - let's visualize the process before implementing it! Here's an undirected, weighted, connected graph:



0
1
2
3
4
5
6
7
8
4
8
11
8
7
2
7
6
1
2
4
14
9
10

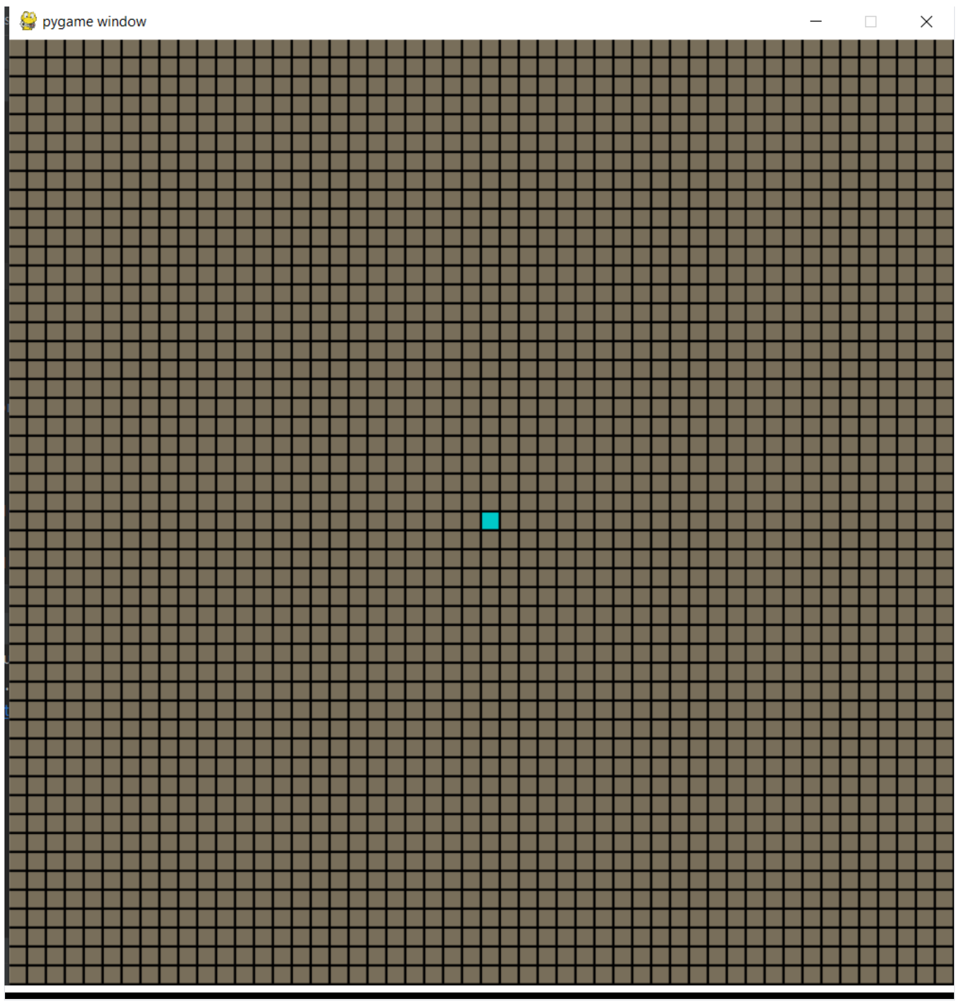
Algorithm

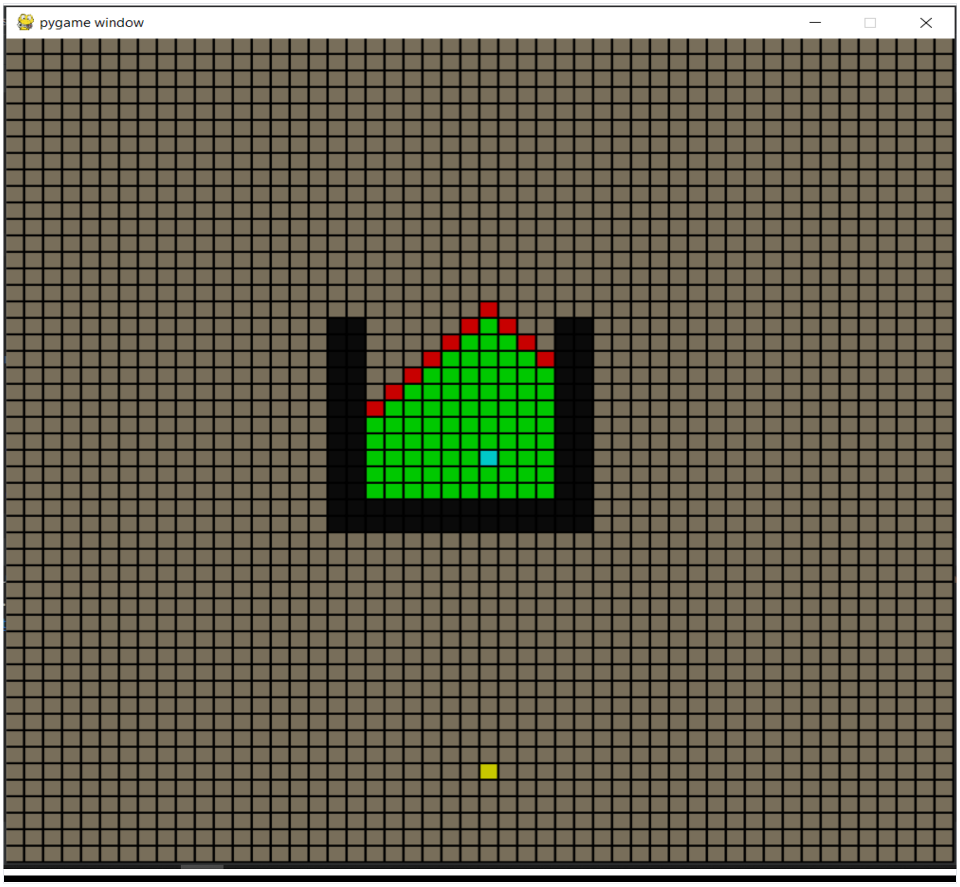
```
class Graph:
    def __init__(self, num_of_vertices):
        self.v = num_of_vertices
        self.edges = [[-1 for i in range(num_of_vertices)] for j in
range(num_of_vertices)]
        self.visited = []
def add_edge(self, u, v, weight):
        self.edges[u][v] = weight
        self.edges[v][u] = weight
def dijkstra(graph, start_vertex):
    D = {v:float('inf') for v in range(graph.v)}
    D[start_vertex] = 0

    pq = PriorityQueue()
    pq.put((0, start_vertex))

    while not pq.empty():
        (dist, current_vertex) = pq.get()
        graph.visited.append(current_vertex)

        for neighbor in range(graph.v):
            if graph.edges[current_vertex][neighbor] != -1:
                distance = graph.edges[current_vertex][neighbor]
                if neighbor not in graph.visited:
                    old_cost = D[neighbor]
                    new_cost = D[current_vertex] + distance
                    if new_cost < old_cost:
                        pq.put((new_cost, neighbor))
                        D[neighbor] = new_cost
    return D
```

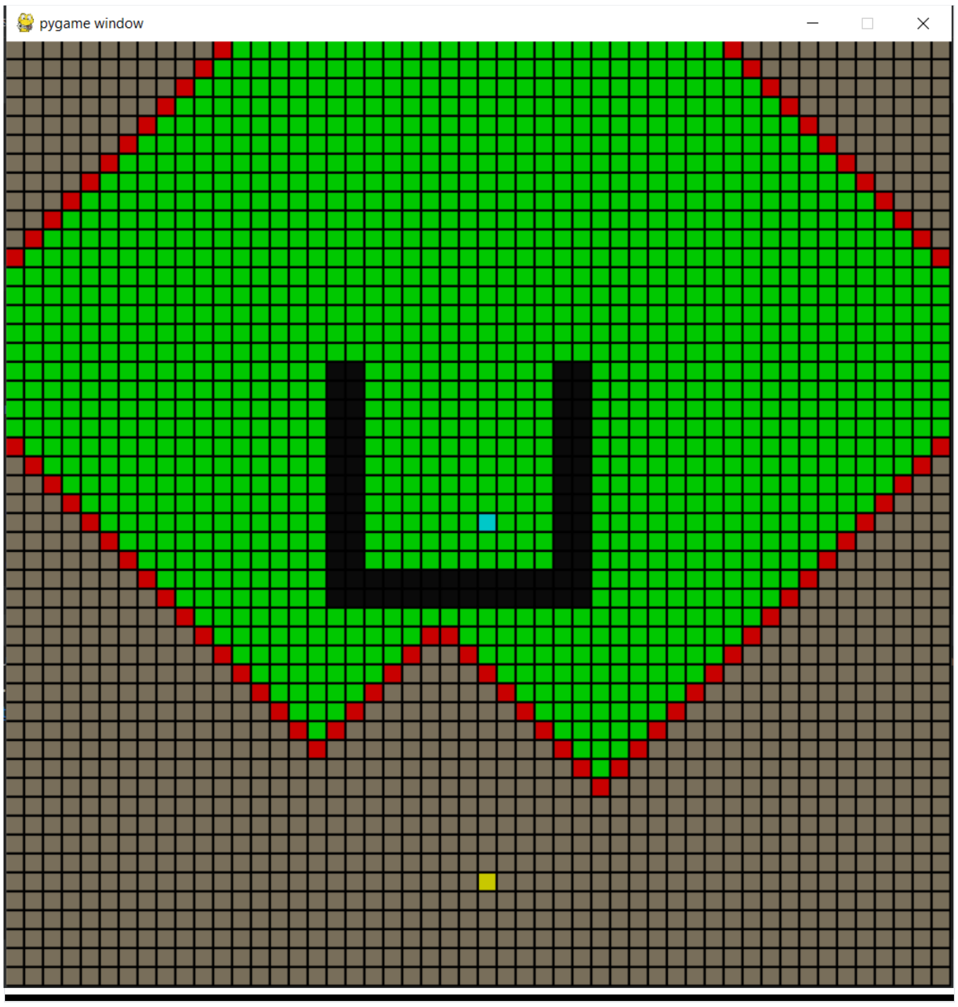
Project Code

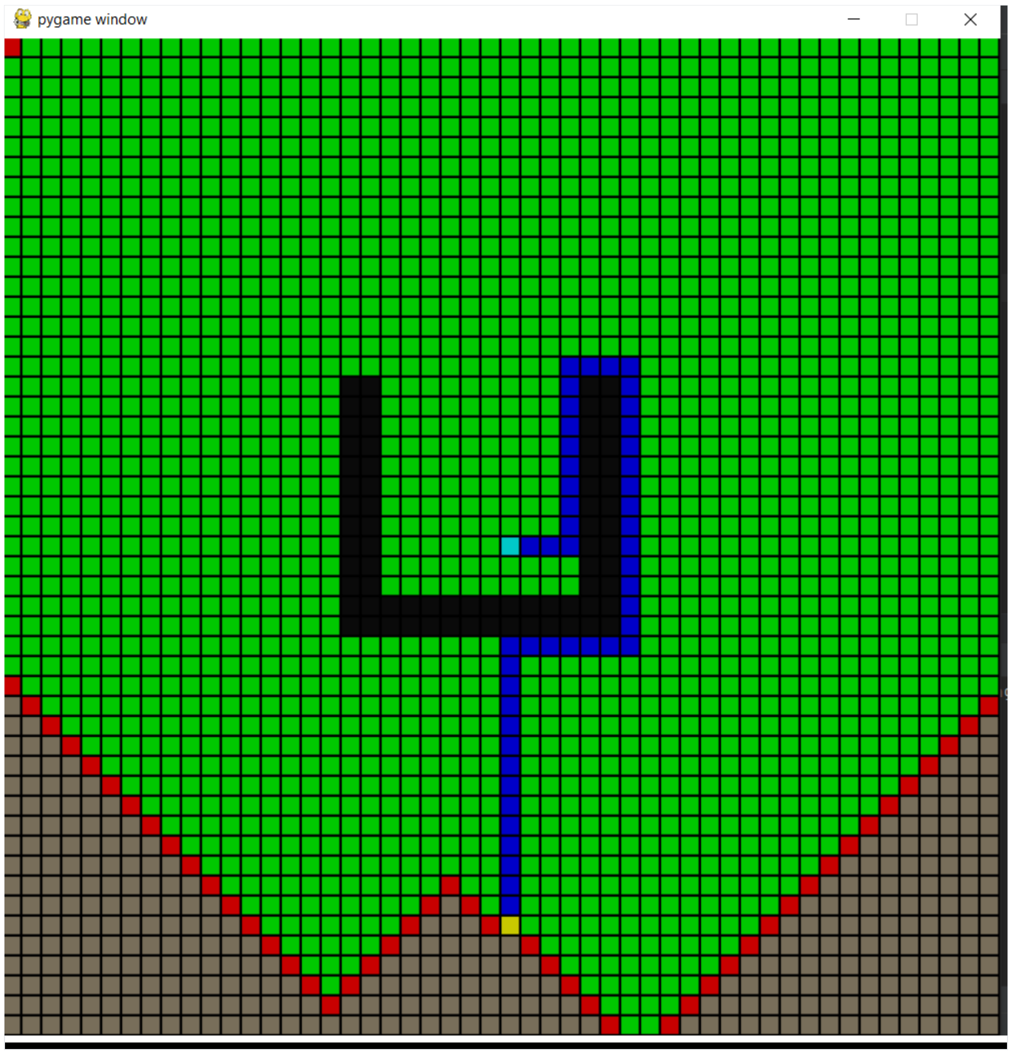
```
from tkinter import messagebox, Tk
import pygame
import sys

window_width = 800
window_height = 800

window = pygame.display.set_mode((window_width, window_height))

columns = 50
rows = 50

box_width = window_width // columns
box_height = window_height // rows

grid = []
queue = []
path = []


class Box:
    def __init__(self, i, j):
        self.x = i
        self.y = j
        self.start = False
        self.wall = False
        self.target = False
        self.queued = False
        self.visited = False
        self.neighbours = []
        self.prior = None

    def draw(self, win, color):
        pygame.draw.rect(win, color, (self.x * box_width, self.y *
box_height, box_width-2, box_height-2))

    def set_neighbours(self):
        if self.x > 0:
            self.neighbours.append(grid[self.x - 1][self.y])
        if self.x < columns - 1:
            self.neighbours.append(grid[self.x + 1][self.y])
        if self.y > 0:
            self.neighbours.append(grid[self.x][self.y - 1])
        if self.y < rows - 1:
            self.neighbours.append(grid[self.x][self.y + 1])


# Create Grid
for i in range(columns):
    arr = []
    for j in range(rows):
        arr.append(Box(i, j))
    grid.append(arr)

# Set Neighbours
for i in range(columns):
    for j in range(rows):
        grid[i][j].set_neighbours()

start_box = grid[25][25]
start_box.start = True
start_box.visited = True
queue.append(start_box)


def main():
    begin_search = False
    target_box_set = False
    searching = True
    target_box = None

    while True:
        for event in pygame.event.get():
            # Quit Window
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            # Mouse Controls
            elif event.type == pygame.MOUSEMOTION:
                x = pygame.mouse.get_pos()[0]
                y = pygame.mouse.get_pos()[1]
                # Draw Wall
                if event.buttons[0]:
                    i = x // box_width
                    j = y // box_height
                    grid[i][j].wall = True
                # Set Target
                if event.buttons[2] and not target_box_set:
                    i = x // box_width
                    j = y // box_height
                    target_box = grid[i][j]
                    target_box.target = True
                    target_box_set = True
            # Start Algorithm
            if event.type == pygame.KEYDOWN and target_box_set:
                begin_search = True

        if begin_search:
            if len(queue) > 0 and searching:
                current_box = queue.pop(0)
                current_box.visited = True
                if current_box == target_box:
                    searching = False
                    while current_box.prior != start_box:
                        path.append(current_box.prior)
                        current_box = current_box.prior
                else:
                    for neighbour in current_box.neighbours:
                        if not neighbour.queued and not neighbour.wall:
                            neighbour.queued = True
                            neighbour.prior = current_box
                            queue.append(neighbour)
            else:
                if searching:
                    Tk().wm_withdraw()
                    messagebox.showinfo("No Solution", "There is no
solution!")
                    searching = False

        window.fill((0, 0, 0))
        for i in range(columns):
            for j in range(rows):
                box = grid[i][j]
                box.draw(window, (120, 110, 90))

                if box.queued:
                    box.draw(window, (200, 0, 0))
                if box.visited:
                    box.draw(window, (0, 200, 0))
                if box in path:
                    box.draw(window, (0, 0, 200))

                if box.start:
                    box.draw(window, (0, 200, 200))
                if box.wall:
                    box.draw(window, (10, 10, 10))
                if box.target:
                    box.draw(window, (200, 200, 0))

        pygame.display.flip()


main()
```

Screenshots




pygame window


pygame window


pygame window

Thank You