Algorithm Visualizer

Submitted in partial fulfillment of the requirements
for the degree of

B.E. Information Technology

By

Himanshu Chaurasiya	**191024**
Vikas Chaurasiya	**191023**
Mukesh Gupta	**191045**
Ashly John	**191051**

Supervisor

Ms. Alvina Alphonso

Assistant Professor

Department of Information Technology

St. Francis Institute of Technology

(Engineering College)

University of Mumbai

2022-2023

CERTIFICATE

This is to certify that the project entitled **"Algorithm Visualizer"** is a bonafide work of **"Himanshu Chaurasiya (191024), Vikas Chaurasiya (191023), Mukesh Gupta (191045) & Ashly John (191051)"** submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the degree of Bachelor of Engineering in Information Technology.

Ms. Alvina Alphonso

(Project Guide)

Dr. Prachi Raut

(Head Of Department)

Dr. Sincy George

(Principal)

Project Report Approval for B.E.

This project report entitled ***"Algorithm Visualizer"*** by ***"Himanshu Chaurasiya (191024), Vikas Chaurasiya (191023), Mukesh Gupta (191045) & Ashly John (191051)"*** is approved for the degree of ***Bachelors of Engineering in Information Technology***.

1. ———————————

2. ———————————

Date:

Place:

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

_____

(Signature)

Himanshu Chaurasiya (191024)

_____

(Signature)

Vikas Chaurasiya (191023)

_____

(Signature)

Mukesh Gupta (191045)

_____

(Signature)

Ashly John (191051)

Date:

ABSTRACT

Algorithms Visualizations contribute to improving computer science education. The method of teaching and learning algorithms is commonly complex to understand the problem. Visualization is a helpful technique for learning in any engineering course. Within the report we discuss the possibility of enriching the standard methods of teaching algorithms, with the algorithm visualizations. As a step in this direction, Our own algorithm visualization platform will visualize topics such as Sliding window problem, matrix chain multiplication, levenshtein distance, and Longest Common subsequence. Sliding windows is a very intriguing technique used to solve some of the complex problems involving an array or a string. Matrix chain multiplication is an optimization problem that seeks the most efficient method of multiplying a given sequence of matrices.The Levenshtein distance is a string metric used to compare two sequences.The longest common subsequence (LCS) problem is the problem of finding the longest subsequence common to all sequences in a set of sequences. Through this project every student can learn at their own pace with our three speeds of learning: slow, average and fast. This will form a clear understanding of the underlying mechanics behind some confusing data structures thus building the foundation for the concepts of advanced methodologies and implementations.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction to domain of project

Algorithm visualization illustrates how algorithms and Advance data structure work in a graphical way. It mainly aims to simplify and deepen the understanding of algorithm operation.Using professional articles of algorithms, animations, and responsive design , each algorithm can be learnt easily, Helps students identify their strengths and areas for improvement, Revisit topics multiple times and practice.Every software engineer should have a good understanding of Algorithms to develop efficient software. Visualizers have a good history of providing effective understanding to the users.

1.2 Major Challenges in said domain

Site responsiveness:- To bring responsiveness to the site, We have gone back to learn CSS styling. It is like going back to the time when we were learning the CS fundamentals and web technologies as a beginner.
Writing sort algorithms in JS:- And coming to the second challenge we faced writing algorithms in JS. When we want to write some code or learn algorithms for solving coding problems, we usually prefer C++/Java.
Visualizing and Controlling animation speed:- Adding features to the visualization so that users can go back and easily understand the previous steps as well the slowing down animation upon users command.

1.3 Motivation

To set up a clear framework:- To ensure that everyone viewing the visualization is on common ground about what it is representing. In order to do so, the designer needs to set up a clear framework, which involves the semantics and syntax under which the data information is

designed to be interpreted. The semantics involve the meaning of the words and graphics used, and the syntax involves the structure of the communication. For example, when using an icon, the element should bear resemblance to the thing it represents, with size, color and position all communicating meaning to the viewer.

It tells a story:- Visualization in its educational or informational role is really a dynamic form of persuasion.Data visualization lends itself well to being a communication medium for story-telling, in particular when the story also contains a lot of data.

1.4 Problem Statement

To build a visualization tool which intend to facilitate the understanding of advance algorithm concepts by means of animation and visualization.

Chapter 2

Literature Review

2.1 Existing Work

2.1.1 Literature review related to exiting system/methodology

Table 2.1: Comparison of methodology

Sr. No.	Title of Paper	Review	Analysis/Limitations
[1]	A Tool for Data Structure Visualization and User-defined Algorithm Animation[10].	In this paper[10], a software application that features the visualization of commonly used data structures and their associated insertion and deletion operations is introduced. In addition, this software can be used to animate user-defined algorithms.	This report[10] provides an e-learning tool for Pathfinder, Prime Numbers, Sorting Algorithms, N Queen, Convex Hull, Binary Search Game visualization is described.
[2]	Algorithm Visualizer: Its features and working[11].	In this paper[11], the concept of Time Complexity has also been introduced to the user through an interactive game.	This report[11] provides visualization of an algorithm such as path finding and sorting algorithm.

Table 2.2: Comparison of methodology

[3]	Sorting Algorithm Visualization[12].	In this paper[12], they have visualized various types of sorting algorithms with their distinct implementations and strategies. They are often not as easy to understand and absorb.	This report provides the visualization of sorting algorithm such as Merge Sort,Quick Sort, Bubble Sort,Insertion, Heap Sort and Selection Sort[12].

2.1.2 Literature review related to algorithms

Table 2.3: Comparison of algorithms

Sr. No.	Title of Paper	Review	Analysis/Limitations
[1]	Sliding Window Algorithms for k-Clustering Problems[5].	This paper[5] we can understand that they have implemented a k-clustering problem by using Sliding Window technique for grouping data into k clusters so that elements within the same cluster are similar to each other that were related to the unsupervised machine learning algorithms.A natural avenue for future work is to give a tighter analysis, and reduce this gap between theory and practice.	This report [5] provides simple and practical algorithms that offer stronger performance guarantees than previous results. Empirically, we show that our methods store only a small fraction of the data, are orders of magnitude faster, and find solutions with costs only slightly higher than those returned by algorithms with access to the full dataset.

Table 2.4: Comparison of methodology

[2]	Coded Matrix Chain Multiplication[1].	The research paper[5] focuses on the computation requirements for matrix chain multiplication and introduces a coding scheme that completes the multiplication in a single round. The conventional matrix multiplication method takes multiple rounds, which results in server load and stragglers. The paper proposes two methods to mitigate the issue of stragglers, including task replication and coded-based method. The coding scheme was implemented on Microsoft Azure and significantly reduced the time of distributed matrix chain multiplication.	This report[5] on Microsoft Azure demonstrates that it significantly saves the time of the distributed matrix chain multiplication and distributed linear regression, compared to multiplying two matrices each time.
[3]	Longest Common Subsequence as Private Search[2].	In this paper[2], they investigate a problem of significant importance to genomic computation: the longest common subsequence (LCS) problem. Their main objective is that the privacy notions from private search give a type of functional security, offering a strong definition for patient privacy while allowing specific questions posed by researchers to be answered.	The paper[2] discusses how the researchers approached the longest common subsequence (LCS) problem as a private search problem, where the objective is to find a string or embedding corresponding to an LCS while maintaining privacy. The paper reveals that deterministic selection strategies do not comply with privacy guarantees and may leak information proportional to the entire input.

2.1.3 Literature review related to tools/technology/framework

Table 2.5: Comparison of tools/technology/framework

Sr. No.	Title of Paper	Review	Analysis/Limitations
[1]	Sorting Algorithm Visualization[12].	In this paper[12], they have visualized various types of sorting algorithms with their distinct implementations and strategies. They are often not as easy to understand and absorb.	They have include the features like to select the size of the data (array of random numbers), visualization speed, sorting algorithm[12]. Along with that, an An extra feature for accessibility called color mode is also present for anyone with a medical condition that is affected by flashing colors.
[2]	Algorithm Visualizer: Its features and working[11].	In this paper[11], the concept of Time Complexity has also been introduced to the user through an interactive game.	The algorithms present in the navbar are chosen on the basis of their popularity and difficulty level.They have also provided tools like maze and Pattern generation between the start and destination node and users can also control the animation speed according to their need[11].
[3]	A Tool for Data Structure Visualization and User-defined Algorithm Animation[10].	In this paper[10], a software application that features the visualization of commonly used data structures and their associated insertion and deletion operations is introduced. In addition, this software can be used to animate user-defined algorithms. 6	This paper[10] Provides the animation of common operations associated with the data structures, such as inserting an element into and deleting an element from the specified data structures and Provides animation of simple user-defined algorithms.

2.2 Research Gap identified

- [10] While these existing research results and developed websites paved the foundation to build an effective and intelligent AV system, there are still many limitations associated with these AV websites which cannot meet the pedagogy, usability and accessibility needs of students: the existing web pages do not have a consistent interface.

- [11]Most AV systems provided promising results with their potential in demonstrating the algorithm and data structure step by step in animation in terms of changing values in the variables. However, the logic behind an algorithm cannot be revealed.

- [3][8][11][12]We also found out that the algorithm which we are going to implement is not visualized yet.

Chapter 3

Proposed Methodology

3.1 Problem Formulation

Through our research paper we found that there is no visualization for Longest common subsequence, sliding window, levenshtein distance and matrix chain multiplication algorithms. All the research was done on implementing the application of the algorithm,so we decided to choose the algorithm for visualization.

3.2 Problem Definition

The objective of this study is to style a system of algorithmic visualization, implement the system and visualize the run time for every implemented algorithmic programme aims to assist students to acknowledge algorithms effectively.

3.3 Scope

- Currently we are implementing algorithms like Sliding Window, Longest Common Subsequence,matrix chain multiplication and Levenshtein Distance.
- User information is not stored in our system.
- A backend database connectivity is also important to the system to keep a record of the existing users and new ones along with providing them the feature of tracking their progress in their respective accounts.
- Our proposed system can be used to enhance the traditional classroom education and textbook for Data Structures and Basic Geometric Algorithms courses

3.4 Proposed Methodology

The proposed system is an implementation of an algorithm in a visually appealing animated way using libraries of javascript. So the visualization tool which is being built is written in javascript and uses html5 canvas element which is very helpful in accessing the tool via any browser. Algorithms that are implemented are Sliding window problem, matrix chain multiplication, Levenshtein distance, and Longest Common subsequence.

3.4.1 Sliding Window Problem

Sliding window is an algorithmic technique for efficiently processing arrays or streams of data in a defined window or subset. Involves moving window along data, performing operation (max/sum/average) for current window and updating as it slides.

To solve the sliding window problem, the algorithm maintains a subarray or window of fixed size that moves through the input array or string. At each step, the subarray shifts by one element or character, and the solution is updated based on the new portion of the input covered by the subarray. This approach minimizes redundant computations and reduces the amount of data that needs to be processed, ultimately leading to more efficient algorithmic solutions.

Mathematically, the sliding window problem can be formulated as finding a subarray or substring of length ”k” in an array or string of ”n” elements that meets a specified criterion or condition. The solution to the problem can be computed by defining two pointers, ”start” and ”end,” and iterating over the input while maintaining the current subarray or window. The algorithm checks whether the current subarray meets the specified criterion, and if so, the solution is updated. The subarray continues to slide until the end of the input is reached.

The sliding window problem is a powerful and versatile technique that has broad applications in computer science and software development. It offers an efficient approach to solving problems that involve analyzing fixed-length portions of data and is a crucial tool for developers working on a wide range of projects.

3.4.2 Levenshtein distance

The Levenshtein distance, also known as edit distance, is a metric that measures the difference between two strings. It calculates the minimum number of operations (insertions, deletions, and substitutions) needed to transform one string into the other.

Given two strings A and B of length n and m respectively, Levenshtein's edit distance is defined as the minimum number of operations required to transform string A into string B, where each operation can be one of three types: insertion, deletion, or substitution.

The mathematical representation of Levenshtein's edit distance is as follows:

$$D(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ and } j = 0 \\ i & \text{if } j = 0 \\ j & \text{if } i = 0 \\ D(i-1,j-1) & \text{if } A_i = B_j \\ 1 + \min\{D(i,j-1), D(i-1,j), D(i-1,j-1)\} & \text{if } A_i \neq B_j \end{cases} \tag{3.1}$$

where $D(i,j)$ is the minimum edit distance between the first i characters of string A and the first j characters of string B. The first three conditions represent the base cases, where the distance is 0 if one of the strings is empty. The fourth condition states that if the i-th character of string A is the same as the j-th character of string B, then no operation is needed. The fifth condition represents the case where the i-th character of string A is different from the j-th character of string B. In this case, we have to choose the operation with the minimum cost (insertion, deletion, or substitution) to transform string A into string B.

The minimum edit distance between the two strings is given by $D(n,m)$, where n is the length of string A and m is the length of string B. Levenshtein's edit distance provides a useful tool for measuring the similarity between two strings of characters. Its mathematical representation allows for efficient computation and implementation in various applications, such as spell-checking and natural language processing.

3.4.3 Longest Common Subsequence

The Longest Common Subsequence (LCS) is a sequence of characters that appear in the same order in two or more strings. The problem of finding the LCS of two or more strings is a classic computer science problem and is used in various fields such as text comparison, DNA sequence analysis, and version control systems. The goal is to find the longest common subsequence of two or more strings, not necessarily contiguous.

Given two sequences, X and Y, the problem of finding their LCS can be defined recursively as follows:

Let m and n be the lengths of sequences X and Y respectively. We define $LCS(X_i, Y_j)$ to be the LCS of the first i elements of X and the first j elements of Y. Then we have:

$$LCS(X_i, Y_j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ LCS(X_{i-1}, Y_{j-1}) + 1 & \text{if } X_i = Y_j \\ \max\{LCS(X_i, Y_{j-1}), LCS(X_{i-1}, Y_j)\} & \text{if } X_i \neq Y_j \end{cases}$$

The first case represents the base case where one of the sequences is empty and there is no common subsequence. The second case states that if the last elements of both sequences match, we add one to the length of the LCS of the remaining sequences. The third case states that if the last elements of the sequences don't match, we take the maximum LCS between the remaining sequences.

The length of the LCS can be found by computing $LCS(X_m, Y_n)$.
The Longest Common Subsequence problem is an important problem in computer science with applications in various fields. Its recursive definition allows for efficient computation and implementation using dynamic programming.

3.4.4 Matrix chain multiplication

Matrix chain multiplication is a problem in computer science that involves finding the optimal way to multiply a sequence of matrices. The objective is to minimize the number of scalar multiplications required to compute the product of the matrices. The problem is formulated as a dynamic programming problem, where the solution is found by breaking down the problem into smaller subproblems and then using the solutions of the subproblems to compute the final solution.

Given a sequence of n matrices $A_1, A_2, \ldots, A_n$, where the dimensions of matrix A_i are $p_{i-1} \times p_i$ for $i = 1, 2, \ldots, n$, the goal is to find the minimum number of scalar multiplications required to compute the product: $A_i A_{i+1} \ldots A_j$ for $1 \leq i \leq j \leq n$.

The base case is when $i = j$, and the product is simply A_i. Thus, $m_{i,i} = 0$ for $1 \leq i \leq n$.

For $j > i$, the number of scalar multiplications required to compute the product $A_i A_{i+1} \ldots A_j$ can be split into two parts: the left subchain $A_i A_{i+1} \ldots A_k$ and the right subchain $A_{k+1} \ldots A_j$ for some $i \leq k < j$. The total number of scalar multiplications required is the sum of the number of scalar multiplications needed to compute the left subchain, the number of scalar multiplications needed to compute the right subchain, and the number of scalar multiplications needed to multiply the resulting matrices together.

The formula for $m_{i,j}$ is therefore: $m[i,i] = 0$ for $i = 1, 2, \ldots, n$

$m[i,j] = \min m[i,k] + m[k+1,j] + p[i-1] \times p[k] \times p[j]$ for $i \leq k < j$

where the base case is when $i = j$, and the matrix is already multiplied. For $i < j$, the minimum number of scalar multiplications required to compute the product of matrices $A_i \times A_{i+1} \times \cdots \times A_j$ can be computed by dividing the product into two sub-products, $A_i \times A_{i+1} \times \cdots \times A_k$ and $A_{k+1} \times A_{k+2} \times \cdots \times A_j$, and multiplying them separately before multiplying the two sub-products. The minimum number of scalar multiplications required for this can be computed by adding the number of scalar multiplications required for multiplying the two sub-products and the number of scalar multiplications required for multiplying the two matrices.

The time complexity of the dynamic programming solution is $O(n^3)$, and the space complexity is $O(n^2)$, where n is the number of matrices in the sequence. The matrix chain multiplication problem has many applications in computer science and engineering, such as in optimization problems, robotics, and computer graphics.

3.5 Proposed Algorithm

Algorithm Visualizer is an effective way to understand complex data structures. The proposed system has various interactive animations for a variety of algorithms. Our visualization tool is written in JavaScript using the HTML5 canvas element, and it can be accessed via any modern browser. The visualizations are meant to be fairly self- explanatory.

3.6 Features of proposed System

1. Interactive Visualization.
2. User can see the code corresponding to visualization.
3. Users can control the speed of the animation.
4. Simple User Interface.

Chapter 4

System Analysis

4.1 Functional Requirements

- Ask students to construct or to customize an AV.
- Allow users to provide input to the algorithm
- Manipulate and direct aspect of visualization
- Speed variation
- Restart/pause algorithm
- Structural view of algorithm
- Data input facilities

4.2 Non-Functional Requirements

- **Accessibility :** The application should be used on any modern browser.
- **Usability :** The application should be user-friendly.
- **Maintenance :** The application interface should be simple and concise so as it can be easily edited in future.
- **Acceptance :** It should meet the user's requirements.
- **Responsive :** The function response time should be smooth.
- **Modifiable :** The function should be modifiable .
- **Sustainable :** The function should be able to be maintained at a certain rate or level.
- User Friendly Graphical Interface.

4.3 Specific Requirements

Hardware Rwquirements :

- **RAM:** Minimum 4 GB
- **Hard Drive:** 32 GB and above
- **Clock speed:** 2.5 GHz or above
- **CPU:** Intel i3/Ryzen 3 or above

Software Requiremnets :

- **Operating System:** Windows 7 and above
- **Front end:** HTML5, CSS3, Bootstrap, Reactjs
- **Editor:** VS code

4.4 Use-Case Diagrams

Use-case diagrams describe the high-level functions and scope of a system. These diagrams also identify the interactions between the system and its actors. The use cases and actors in use-case diagrams describe what the system does and how the actors use it, but not how the system operates internally.

Table 4.1: Use Case Diagram Description

Actors	Description
User	user will open the application and he/she can see the welcome screen. Once the model appears on the screen the user has the option to close the algorithm or use extra features like the speed control and give suitable input data value

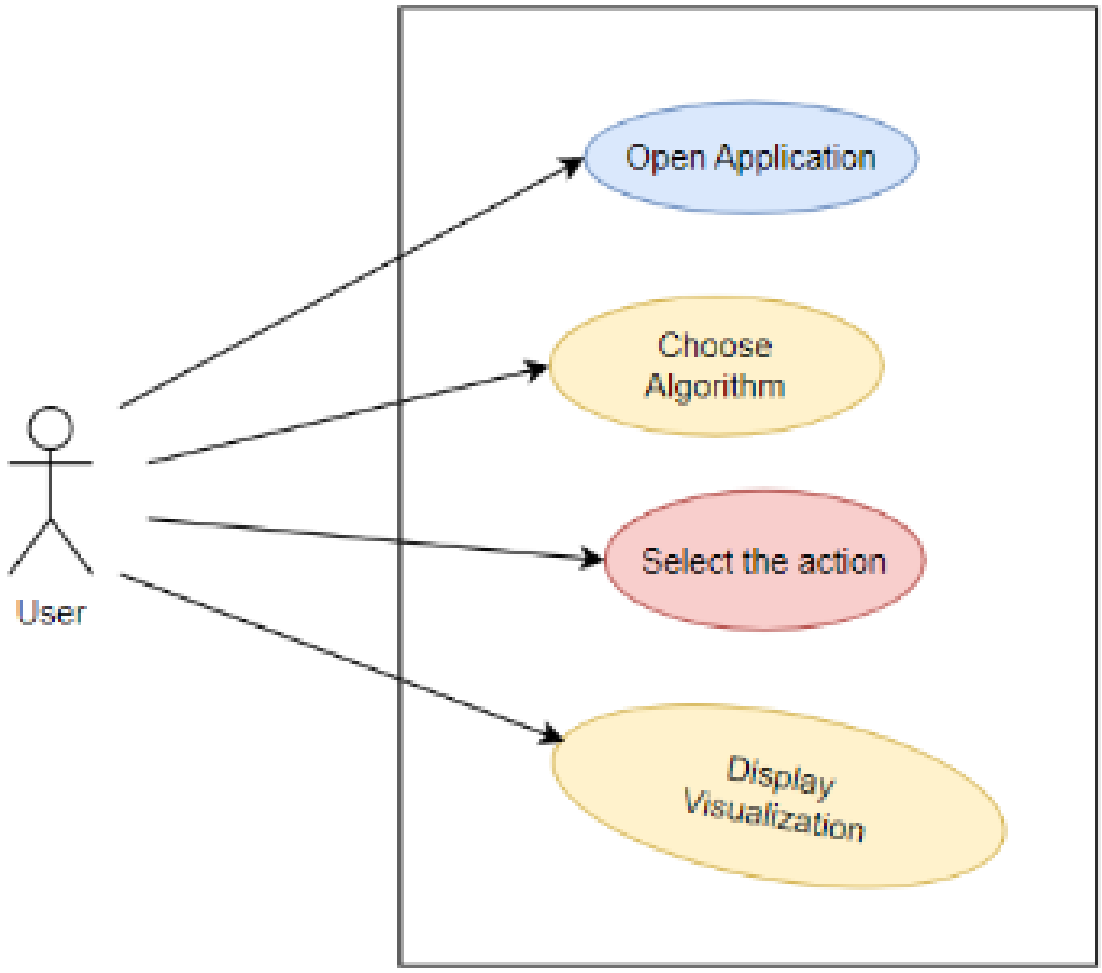

Figure 4.1: Use Case Diagram

Table 4.2: Use Case Diagram Operations Description

Operation	Description
Open Application	The First stage is Opening the application.
Choose Algorithm	Once the model appears on the screen the user has the option to close the algorithm.
Select the action	Use extra features like the speed control and give suitable input data value.
Visualization Software	Visualize Software takes the query which is selected by the user and performs some operation and displays the visualization of the selected algorithm.

Chapter 5

Analysis Modeling

5.1 Data Modeling : Class Diagram

A class diagram is a static diagram. It represents the application's static view. Class diagrams are used to create executable code for software applications as well as for visualizing, explaining, and documenting various elements of systems.

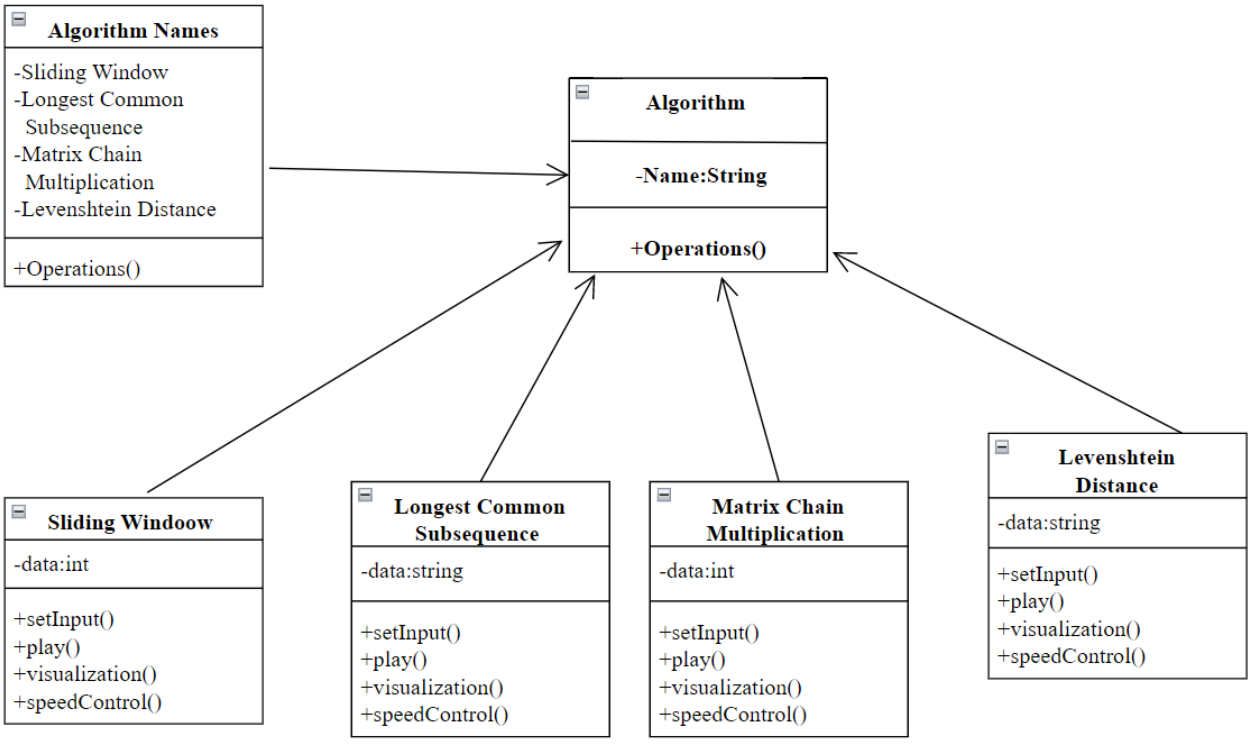

Figure 5.1: Class Diagram.

5.2 Activity Diagrams

An activity diagram visually presents a series of actions or flow of control in a system similar to a flowchart or a data flow diagram. Activity diagrams are often used in business process modeling. They can also describe the steps in a use case diagram. Activities modeled can be sequential and concurrent.

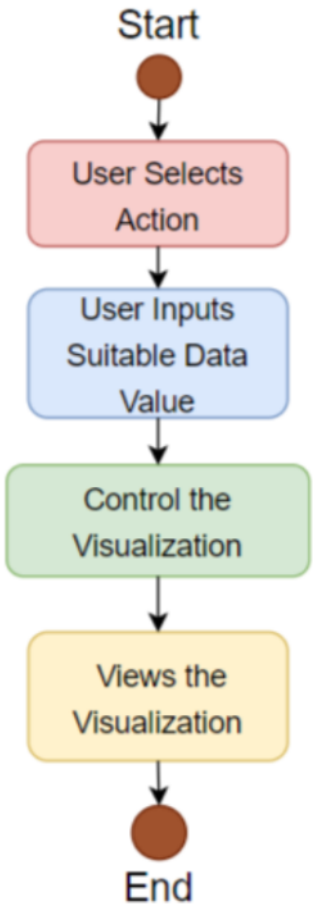

Figure 5.2: Activity Diagram for User.

The Activity Diagram gives us a gist of the stages in the application Process. The First stage is Opening the application. Once the model appears on the screen the user has the option to close the algorithm or use extra features like the speed control and give suitable input data value.

5.3 Functional Modeling

5.3.1 DFD:Level 0

DFD Level 0 is also called a Context Diagram. It's a basic overview of the whole system or process being analyzed or modeled. It's designed to be an at-a-glance view, showing the system as a single high-level process, with its relationship to external entities.

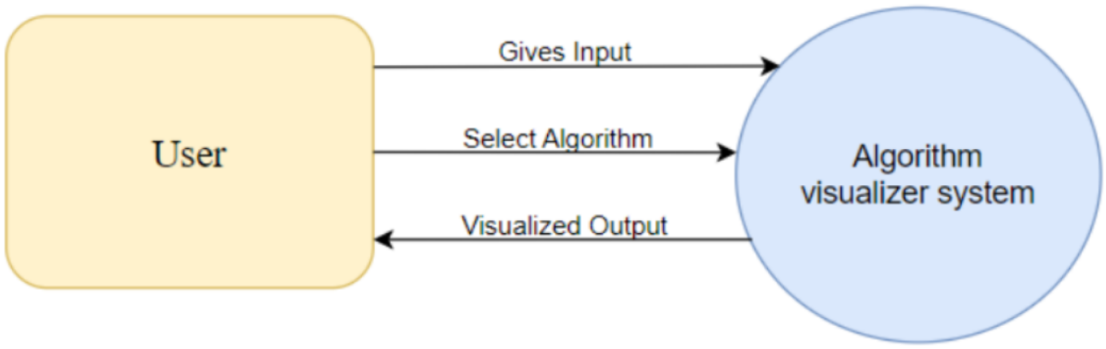

Figure 5.3: DFD Level 0.

In the above DFD the user gives the input and selects the algorithm. Then the algorithm visualizer system visualized output for the selected algorithm.

5.3.2 DFD:Level 1

Level 1 DFDs are still a general overview, but they go into more detail than a context diagram. In level 1 DFD, the single process node from the context diagram is broken down into sub-processes. As these processes are added, the diagram will need additional data flows and data stores to link them together.

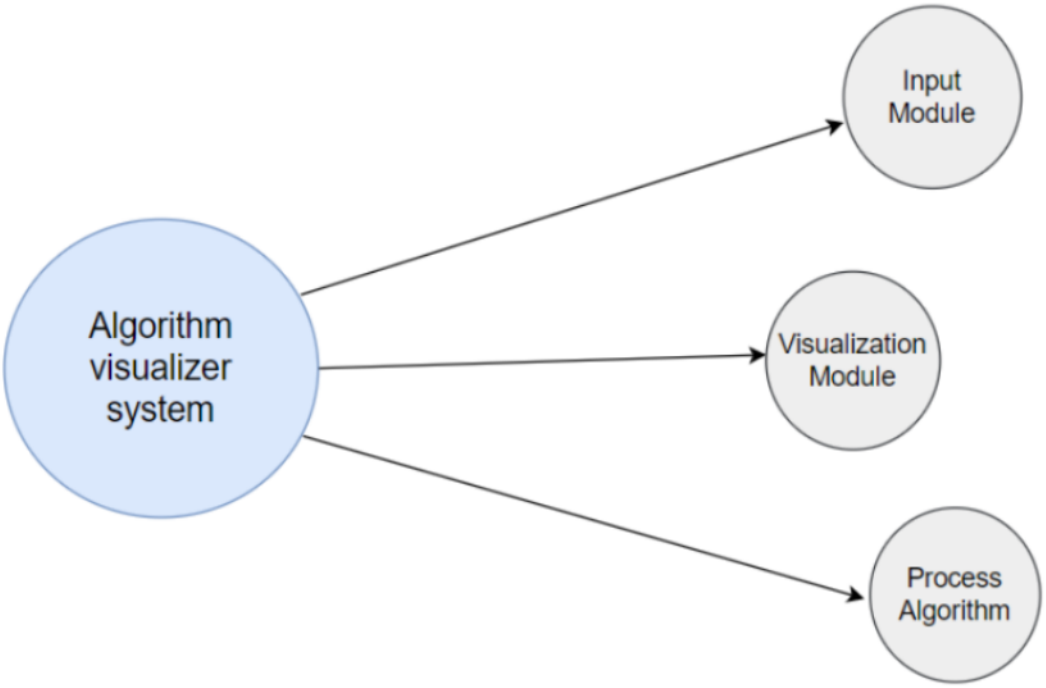

Figure 5.4: DFD Level 1.

In the above DFD level 1 Algorithm visualizer system takes the input module and selects the algorithm and processes it and displays the visualization.

5.4 TimeLine Chart

	Name	Duration	Start	Finish	...
1	Major Project sem 7	194 days	11/7/22 8:00 AM	6/4/23 5:00 PM	
2	Group Formation	2.75 days	11/7/22 8:00 AM	13/7/22 3:00 PM	
3	Finalization Group	2 days	14/7/22 8:00 AM	15/7/22 5:00 PM	2
4	Topic Formulation	5 days	18/7/22 8:00 AM	22/7/22 5:00 PM	3
5	Project Proposal	10 days	25/7/22 8:00 AM	5/8/22 5:00 PM	4
6	Proposal Submisssion	9.5 days	25/7/22 8:00 AM	5/8/22 1:00 PM	5
7	Proposal Presentation	1.75 days	8/8/22 8:00 AM	9/8/22 3:00 PM	6
8	Proposal Approval	3 days	10/8/22 8:00 AM	12/8/22 5:00 PM	7
9	Guide Allotment	2 days	15/8/22 8:00 AM	16/8/22 5:00 PM	8
10	Meeting Project Guide	10 days	17/8/22 8:00 AM	30/8/22 5:00 PM	9
11	Disscusion	10 days	17/8/22 8:00 AM	30/8/22 5:00 PM	10
12	Algorithm Finalize	5 days	31/8/22 8:00 AM	6/9/22 5:00 PM	11
13	Planning	5 days	7/9/22 8:00 AM	13/9/22 5:00 PM	12
14	Literature Review	5 days	7/9/22 8:00 AM	13/9/22 5:00 PM	13
15	Deciding Module	5 days	7/9/22 8:00 AM	13/9/22 5:00 PM	13
16	UML Diagram	5 days	7/9/22 8:00 AM	13/9/22 5:00 PM	15
17	Use Case Diagram	5 days	7/9/22 8:00 AM	13/9/22 5:00 PM	15
18	Approval of Diagram	5 days	14/9/22 8:00 AM	20/9/22 5:00 PM	17
19	Mid Term Evaluation	5 days	21/9/22 8:00 AM	27/9/22 5:00 PM	18
20	Report Building	10 days	28/9/22 8:00 AM	11/10/22 5:00 PM	19
21	Report Changes	10 days	12/10/22 8:00 AM	25/10/22 5:00 PM	20
22	End Sem Evaluation	3 days	26/10/22 8:00 AM	28/10/22 5:00 PM	21

Figure 5.5: Time Chart Tasks: Semester 7

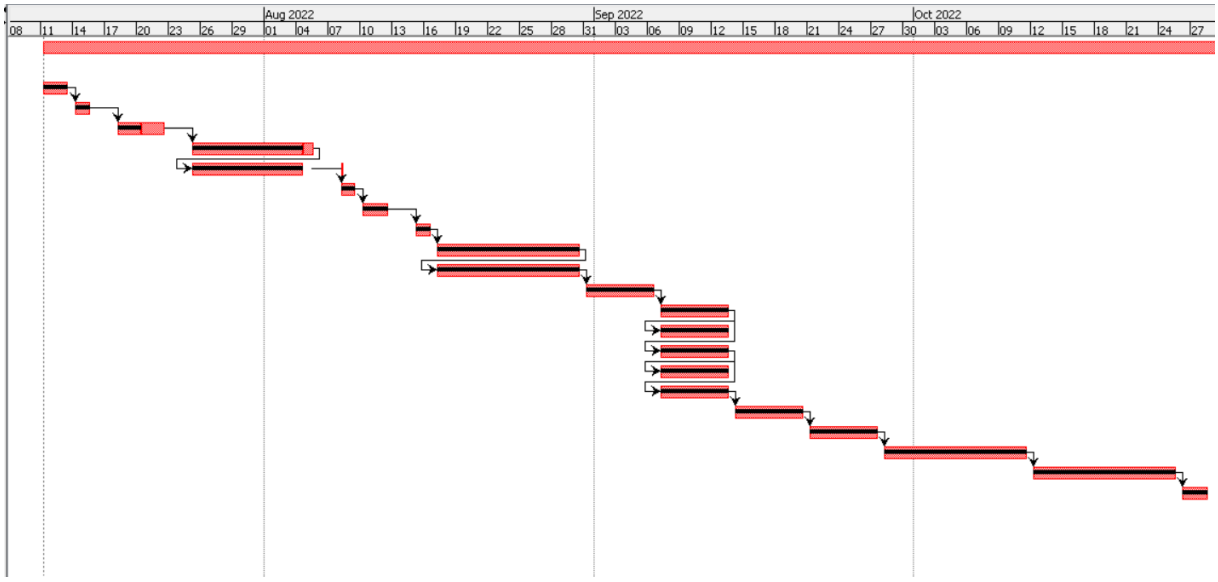

Figure 5.6: Gantt Chart From July 2022 - October 2022.

	Name	Duration	Start	Finish
1	MAJOR PROJECT	85 days	10/1/23 8:00 AM	8/5/23 5:00 PM
3	Discussion with Guide	1 day	10/1/23 8:00 AM	10/1/23 5:00 PM
4	**Paper**	11 days	**17/1/23 8:00 AM**	**31/1/23 5:00 PM**
5	Paper discussion with guide	4 days	17/1/23 8:00 AM	20/1/23 5:00 PM
6	Changes Implemented	4 days	24/1/23 8:00 AM	27/1/23 5:00 PM
7	Paper discussion with guide	3 days	27/1/23 8:00 AM	31/1/23 5:00 PM
8	Changes Implemented	1 day	31/1/23 8:00 AM	31/1/23 5:00 PM
9	**Implementation**	65 days	**1/2/23 8:00 AM**	**2/5/23 5:00 PM**
10	10% Implementation	8 days	1/2/23 8:00 AM	10/2/23 5:00 PM
11	30% Implemetation	12 days	13/2/23 8:00 AM	28/2/23 5:00 PM
12	40% Implemented	7 days	28/2/23 8:00 AM	8/3/23 5:00 PM
13	Discussion With Guide	1 day	7/3/23 8:00 AM	7/3/23 5:00 PM
14	50% Implemtation	3 days	8/3/23 8:00 AM	10/3/23 5:00 PM
15	75% Implementation	3 days	13/3/23 8:00 AM	15/3/23 5:00 PM
16	Poster Making	10 days	1/3/23 8:00 AM	14/3/23 5:00 PM
17	Mid Term Evaluation	1 day	15/3/23 8:00 AM	15/3/23 5:00 PM
18	Changes Implemented	2 days	17/3/23 8:00 AM	20/3/23 5:00 PM
19	90% Implemented	15 days	20/3/23 8:00 AM	7/4/23 5:00 PM
20	Poster Discussion	1 day	7/4/23 8:00 AM	7/4/23 5:00 PM
21	Poster Completed	2 days	10/4/23 8:00 AM	11/4/23 5:00 PM
22	Evaluation	1 day	12/4/23 8:00 AM	12/4/23 5:00 PM
23	Report Making	7 days	13/4/23 8:00 AM	21/4/23 5:00 PM
24	Full Project Completed	8 days	21/4/23 8:00 AM	2/5/23 5:00 PM

Figure 5.7: Time Chart Tasks: Semester 8

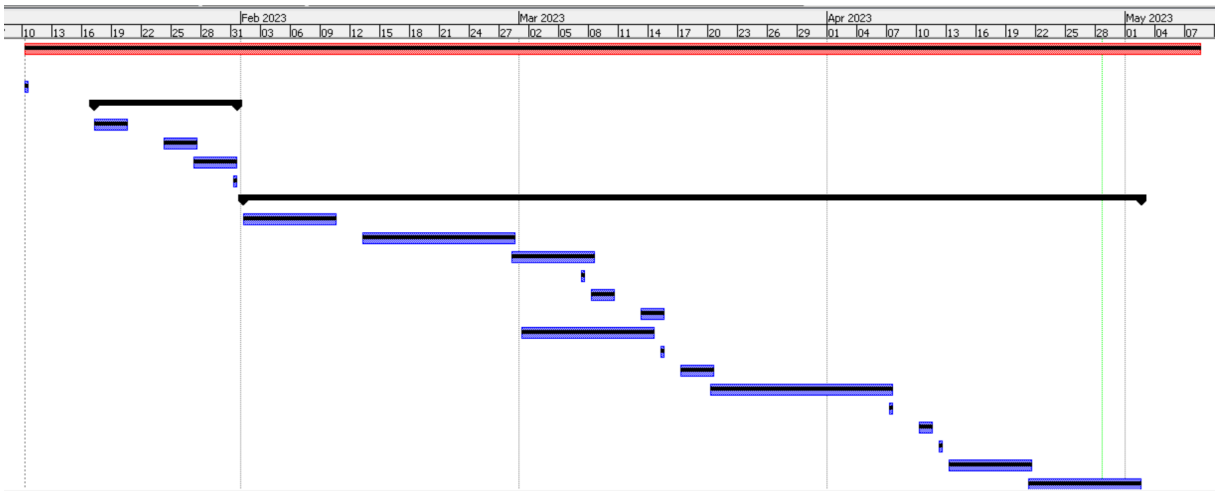

Figure 5.8: Gantt Chart From January 2023 - May 2023.

Chapter 6

Implementation

6.1 Working of the Project

The algorithm visualizer project is a web-based application that allows users to visualize how different algorithms work. The project is built using the ReactJS framework, which is a popular JavaScript library for building user interfaces. The project's front-end component comprises a user interface (UI) that presents a visualization of the algorithm, a code editor, and multiple controls that enable the user to interact with the visualization. These controls consist of a build button, play button, pause button, and speed controller, which provide the user with various options for controlling the visualization of the algorithm on the UI. The build button is used to build the input that is taken from the user. This input can be in the form of an array of numbers or a string of characters, depending on the algorithm being visualized. Once the user has inputted the data, they can click the build button to build the visualization. The play button is used to start the visualization, while the pause button is used to pause the visualization. The speed controller allows the user to control the speed of the visualization, so they can slow it down or speed it up as needed. The code editor within the project allows the user to view the algorithm's source code that is being visualized. By modifying the code in the editor and then rebuilding the visualization, the user can make minor adjustments to the algorithm's behavior and observe the changes in the visualization. To implement the project, the developers used ReactJS to build the frontend of the application. They used HTML, CSS, and JavaScript to create the UI and various controls. Overall, the technical implementation of the algorithm visualizer project required a deep understanding of ReactJS, HTML, CSS, JavaScript, and various libraries and frameworks. The developers had to carefully plan and execute each aspect of the project to ensure that it was both functional and visually appealing.

6.2 Performance Metrics Used

6.2.1 Run time

Runtime performance metrics refer to the measurements used to evaluate the speed and efficiency of a software program or algorithm during execution. This metric measures the time it takes for a program or algorithm to complete its execution, from the moment it starts running to the moment it finishes. There are several factors that can affect runtime performance, including the complexity of the algorithm, the size of the input data set, and the processing power and memory of the computer or device running the program. By monitoring and optimizing runtime performance metrics, developers can ensure that their software programs and algorithms are performing optimally and delivering the best possible user experience

6.2.2 Memory Usage

Memory usage performance metrics refer to the measurements used to evaluate the amount of memory consumed by a software program or algorithm during execution. This metric is important because excessive memory usage can lead to slow performance, crashes, and other issues that can impact the overall user experience.Memory usage can be affected by several factors, including the size and complexity of the data structures used by the program or algorithm, the number and type of variables stored in memory, and the efficiency of memory allocation and deallocation strategies. By monitoring and optimizing memory usage performance metrics, developers can ensure that their software programs and algorithms are performing optimally and delivering the best possible user experience.

6.3 Code

algorithm-visualizer\src\components\App\index.js

```
1  import React from 'react';
2  import Cookies from 'js-cookie';
3  import { connect } from 'react-redux';
4  import Promise from 'bluebird';
5  import { Helmet } from 'react-helmet';
6  import queryString from 'query-string';
7  import {
8    BaseComponent, CodeEditor,Header,  Navigator,ResizableContainer,
       TabContainer, ToastContainer,VisualizationViewer,
9  } from 'components';
10 import { AlgorithmApi, GitHubApi, VisualizationApi } from 'apis';
11 import { actions } from 'reducers';
12 import { createUserFile, extension, refineGist } from 'common/util';
13 import { exts, languages } from 'common/config';
14 import { SCRATCH_PAPER_README_MD } from 'files';
15 import styles from './App.module.scss';
16
17 class App extends BaseComponent {
18   constructor(props) {
19     super(props);
20
21     this.state = {
22       workspaceVisibles: [true, true, true],
23       workspaceWeights: [1, 2, 2],
24     };
25
26     this.codeEditorRef = React.createRef();
27
28     this.ignoreHistoryBlock = this.ignoreHistoryBlock.bind(this);
29     this.handleClickTitleBar = this.handleClickTitleBar.bind(this);
30     this.loadScratchPapers = this.loadScratchPapers.bind(this);
31     this.handleChangeWorkspaceWeights = this.
         handleChangeWorkspaceWeights.bind(this);
32   }
33
34   componentDidMount() {
35     window.signIn = this.signIn.bind(this);
36     window.signOut = this.signOut.bind(this);
37
38     const { params } = this.props.match;
39     const { search } = this.props.location;
40     this.loadAlgorithm(params, queryString.parse(search));
41
```

```
42      const accessToken = Cookies.get('access_token');
43      if (accessToken) this.signIn(accessToken);
44
45      AlgorithmApi.getCategories()
46        .then(({ categories }) => this.props.setCategories(categories))
47        .catch(this.handleError);
48
49      this.toggleHistoryBlock(true);
50    }
51
52    componentWillUnmount() {
53      delete window.signIn;
54      delete window.signOut;
55
56      this.toggleHistoryBlock(false);
57    }
58
59    componentWillReceiveProps(nextProps) {
60      const { params } = nextProps.match;
61      const { search } = nextProps.location;
62      if (params !== this.props.match.params || search !== this.props.
          location.search) {
63        const { categoryKey, algorithmKey, gistId } = params;
64        const { algorithm, scratchPaper } = nextProps.current;
65        if (algorithm && algorithm.categoryKey === categoryKey &&
            algorithm.algorithmKey === algorithmKey) return;
66        if (scratchPaper && scratchPaper.gistId === gistId) return;
67        this.loadAlgorithm(params, queryString.parse(search));
68      }
69    }
70
71    toggleHistoryBlock(enable = !this.unblock) {
72      if (enable) {
73        const warningMessage = 'Are you sure you want to discard changes?'
            ;
74        window.onbeforeunload = () => {
75          const { saved } = this.props.current;
76          if (!saved) return warningMessage;
77        };
78        this.unblock = this.props.history.block((location) => {
79          if (location.pathname === this.props.location.pathname) return;
80          const { saved } = this.props.current;
81          if (!saved) return warningMessage;
82        });
83      } else {
84        window.onbeforeunload = undefined;
85        this.unblock();
```

```
86        this.unblock = undefined;
87      }
88    }
89
90    ignoreHistoryBlock(process) {
91      this.toggleHistoryBlock(false);
92      process();
93      this.toggleHistoryBlock(true);
94    }
95
96    signIn(accessToken) {
97      Cookies.set('access_token', accessToken);
98      GitHubApi.auth(accessToken)
99        .then(() => GitHubApi.getUser())
100       .then(user => {
101         const { login, avatar_url } = user;
102         this.props.setUser({ login, avatar_url });
103       })
104       .then(() => this.loadScratchPapers())
105       .catch(() => this.signOut());
106   }
107
108   signOut() {
109     Cookies.remove('access_token');
110     GitHubApi.auth(undefined)
111       .then(() => {
112         this.props.setUser(undefined);
113       })
114       .then(() => this.props.setScratchPapers([]));
115   }
116
117   loadScratchPapers() {
118     const per_page = 100;
119     const paginateGists = (page = 1, scratchPapers = []) => GitHubApi.
          listGists({
120       per_page,
121       page,
122       timestamp: Date.now(),
123     }).then(gists => {
124       scratchPapers.push(...gists.filter(gist => 'algorithm-visualizer'
            in gist.files).map(gist => ({
125         key: gist.id,
126         name: gist.description,
127         files: Object.keys(gist.files),
128       })));
129       if (gists.length < per_page) {
130         return scratchPapers;
```

```
131       } else {
132         return paginateGists(page + 1, scratchPapers);
133       }
134     });
135     return paginateGists()
136       .then(scratchPapers => this.props.setScratchPapers(scratchPapers))
137       .catch(this.handleError);
138   }
139
140   loadAlgorithm({ categoryKey, algorithmKey, gistId }, { visualizationId
          }) {
141     const { ext } = this.props.env;
142     const fetch = () => {
143       if (window.__PRELOADED_ALGORITHM__) {
144         this.props.setAlgorithm(window.__PRELOADED_ALGORITHM__);
145         delete window.__PRELOADED_ALGORITHM__;
146       } else if (window.__PRELOADED_ALGORITHM__ === null) {
147         delete window.__PRELOADED_ALGORITHM__;
148         return Promise.reject(new Error('Algorithm Not Found'));
149       } else if (categoryKey && algorithmKey) {
150         return AlgorithmApi.getAlgorithm(categoryKey, algorithmKey)
151           .then(({ algorithm }) => this.props.setAlgorithm(algorithm));
152       } else if (gistId === 'new' && visualizationId) {
153         return VisualizationApi.getVisualization(visualizationId)
154           .then(content => {
155             this.props.setScratchPaper({
156               login: undefined,
157               gistId,
158               title: 'Untitled',
159               files: [SCRATCH_PAPER_README_MD, createUserFile('
                      visualization.json', JSON.stringify(content))],
160             });
161           });
162       } else if (gistId === 'new') {
163         const language = languages.find(language => language.ext === ext
              );
164         this.props.setScratchPaper({
165           login: undefined,
166           gistId,
167           title: 'Untitled',
168           files: [SCRATCH_PAPER_README_MD, language.skeleton],
169         });
170       } else if (gistId) {
171         return GitHubApi.getGist(gistId, { timestamp: Date.now() })
172           .then(refineGist)
173           .then(this.props.setScratchPaper);
174       } else {
```

```
175         this.props.setHome();
176       }
177       return Promise.resolve();
178     };
179     fetch()
180       .then(() => {
181         this.selectDefaultTab();
182         return null; // to suppress unnecessary bluebird warning
183       })
184       .catch(error => {
185         this.handleError(error);
186         this.props.history.push('/');
187       });
188   }
189
190   selectDefaultTab() {
191     const { ext } = this.props.env;
192     const { files } = this.props.current;
193     const editingFile = files.find(file => extension(file.name) === '
          json') ||
194       files.find(file => extension(file.name) === ext) ||
195       files.find(file => exts.includes(extension(file.name))) ||
196       files[files.length - 1];
197     this.props.setEditingFile(editingFile);
198   }
199
200   handleChangeWorkspaceWeights(workspaceWeights) {
201     this.setState({ workspaceWeights });
202     this.codeEditorRef.current.handleResize();
203   }
204
205   toggleNavigatorOpened(navigatorOpened = !this.state.workspaceVisibles
        [0]) {
206     const workspaceVisibles = [...this.state.workspaceVisibles];
207     workspaceVisibles[0] = navigatorOpened;
208     this.setState({ workspaceVisibles });
209   }
210
211   handleClickTitleBar() {
212     this.toggleNavigatorOpened();
213   }
214
215   render() {
216     const { workspaceVisibles, workspaceWeights } = this.state;
217     const { titles, description, saved } = this.props.current;
218
219     const title = `${saved ? '' : '(Unsaved) '}${titles.join(' - ')}`;
```

```
220     const [navigatorOpened] = workspaceVisibles;
221
222     return (
223       <div className={styles.app}>
224         <Helmet>
225           <title>{title}</title>
226           <meta name="description" content={description}/>
227         </Helmet>
228         <Header className={styles.header} onClickTitleBar={this.
              handleClickTitleBar}
229                 navigatorOpened={navigatorOpened} loadScratchPapers={
                      this.loadScratchPapers}
230                 ignoreHistoryBlock={this.ignoreHistoryBlock}/>
231         <ResizableContainer className={styles.workspace} horizontal
              weights={workspaceWeights}
232                             visibles={workspaceVisibles} onChangeWeights
                                ={this.handleChangeWorkspaceWeights}>
233           <Navigator/>
234           <VisualizationViewer className={styles.visualization_viewer}/>
235           <TabContainer className={styles.editor_tab_container}>
236             <CodeEditor ref={this.codeEditorRef}/>
237           </TabContainer>
238         </ResizableContainer>
239         <ToastContainer className={styles.toast_container}/>
240       </div>
241     );
242   }
243 }
244
245 export default connect(({ current, env }) => ({ current, env }), actions
      )(
246   App,
247 );
```

algorithm-visualizer\src\components\App\App.module.scss

```
1  @import "~common/stylesheet/index";
2
3  .app {
4    display: flex;
5    flex-direction: column;
6    align-items: stretch;
7    height: 100%;
8    background-color: $theme-normal;
9
10   .header {
11   }
12
```

```
13    .workspace {
14      flex: 1;
15
16      .visualization_viewer {
17        background-color: $theme-dark;
18      }
19
20      .editor_tab_container {
21      }
22    }
23
24    .toast_container {
25      position: absolute;
26      bottom: 0;
27      right: 0;
28      z-index: 99;
29    }
30  }
```

algorithm-visualizer\src\components\BaseComponent\index.js

```
1  import React from 'react';
2
3  class BaseComponent extends React.Component {
4    constructor(props) {
5      super(props);
6
7      this.handleError = this.handleError.bind(this);
8    }
9
10   handleError(error) {
11     if (error.response) {
12       const { data, statusText } = error.response;
13       const message = data ? typeof data === 'string' ? data : JSON.
             stringify(data) : statusText;
14       console.error(message);
15       this.props.showErrorToast(message);
16     } else {
17       console.error(error.message);
18       this.props.showErrorToast(error.message);
19     }
20   }
21 }
22
23 export default BaseComponent;
```

algorithm-visualizer\src\components\Button\index.js

```
1  import React from 'react';
```

```
2  import { Link } from 'react-router-dom';
3  import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
4  import faExclamationCircle from '@fortawesome/fontawesome-free-solid/
       faExclamationCircle';
5  import faSpinner from '@fortawesome/fontawesome-free-solid/faSpinner';
6  import { classes } from 'common/util';
7  import { Ellipsis } from 'components';
8  import styles from './Button.module.scss';
9
10 class Button extends React.Component {
11   constructor(props) {
12     super(props);
13
14     this.state = {
15       confirming: false,
16     };
17
18     this.timeout = null;
19   }
20
21   componentWillUnmount() {
22     if (this.timeout) {
23       window.clearTimeout(this.timeout);
24       this.timeout = undefined;
25     }
26   }
27
28   render() {
29     let { className, children, to, href, onClick, icon, reverse,
           selected, disabled, primary, active, confirmNeeded, inProgress,
           ...rest } = this.props;
30     const { confirming } = this.state;
31
32     if (confirmNeeded) {
33       if (confirming) {
34         className = classes(styles.confirming, className);
35         icon = faExclamationCircle;
36         children = <Ellipsis key="text">Click to Confirm</Ellipsis>;
37         const onClickOriginal = onClick;
38         onClick = () => {
39           if (onClickOriginal) onClickOriginal();
40           if (this.timeout) {
41             window.clearTimeout(this.timeout);
42             this.timeout = undefined;
43             this.setState({ confirming: false });
44           }
45         };
```

```
46        } else {
47          to = null;
48          href = null;
49          onClick = () => {
50            this.setState({ confirming: true });
51            this.timeout = window.setTimeout(() => {
52              this.timeout = undefined;
53              this.setState({ confirming: false });
54            }, 2000);
55          };
56        }
57      }
58
59      const iconOnly = !children;
60      const props = {
61        className: classes(styles.button, reverse && styles.reverse,
             selected && styles.selected, disabled && styles.disabled,
             primary && styles.primary, active && styles.active, iconOnly &&
              styles.icon_only, className),
62        to: disabled ? null : to,
63        href: disabled ? null : href,
64        onClick: disabled ? null : onClick,
65        children: [
66          icon && (
67            typeof icon === 'string' ?
68              <div className={classes(styles.icon, styles.image)} key="
                   icon"
69                    style={{ backgroundImage: `url(${icon})` }} /> :
70              <FontAwesomeIcon className={styles.icon} fixedWidth icon={
                   inProgress ? faSpinner : icon} spin={inProgress}
71                                key="icon" />
72          ),
73          children,
74        ],
75        ...rest,
76      };
77
78      return to ? (
79        <Link {...props} />
80      ) : href ? (
81        <a rel="noopener" target="_blank" {...props} />
82      ) : (
83        <div {...props} />
84      );
85    }
86  }
87
```

```
88 export default Button;
```

algorithm-visualizer\src\components\CodeEditor\index.js

```
1  import React from 'react';
2  import faTrashAlt from '@fortawesome/fontawesome-free-solid/faTrashAlt';
3  import faUser from '@fortawesome/fontawesome-free-solid/faUser';
4  import { classes, extension } from 'common/util';
5  import { actions } from 'reducers';
6  import { connect } from 'react-redux';
7  import { languages } from 'common/config';
8  import { Button, Ellipsis, FoldableAceEditor } from 'components';
9  import styles from './CodeEditor.module.scss';
10
11 class CodeEditor extends React.Component {
12   constructor(props) {
13     super(props);
14
15     this.aceEditorRef = React.createRef();
16   }
17
18   handleResize() {
19     this.aceEditorRef.current.resize();
20   }
21
22   render() {
23     const { className } = this.props;
24     const { editingFile } = this.props.current;
25     const { user } = this.props.env;
26     const { lineIndicator } = this.props.player;
27
28     if (!editingFile) return null;
29
30     const fileExt = extension(editingFile.name);
31     const language = languages.find(language => language.ext === fileExt
       );
32     const mode = language ? language.mode :
33       fileExt === 'md' ? 'markdown' :
34         fileExt === 'json' ? 'json' :
35           'plain_text';
36
37     return (
38       <div className={classes(styles.code_editor, className)}>
39         <FoldableAceEditor
40           className={styles.ace_editor}
41           ref={this.aceEditorRef}
42           mode={mode}
43           theme="tomorrow_night_eighties"
```

```
44            name="code_editor"
45            editorProps={{ $blockScrolling: true }}
46            onChange={code => this.props.modifyFile(editingFile, code)}
47            markers={lineIndicator ? [{
48              startRow: lineIndicator.lineNumber,
49              startCol: 0,
50              endRow: lineIndicator.lineNumber,
51              endCol: Infinity,
52              className: styles.current_line_marker,
53              type: 'line',
54              inFront: true,
55              _key: lineIndicator.cursor,
56            }] : []}
57            value={editingFile.content}/>
58          <div className={classes(styles.contributors_viewer, className)}>
59            <span className={classes(styles.contributor, styles.label)}>
                 Vikas Chaurasiya</span>
60            {
61              (editingFile.contributors || [user || { login: 'guest',
                   avatar_url: faUser }]).map(contributor => (
62                <Button className={styles.contributor} icon={contributor.
                     avatar_url} key={contributor.login}
63                        href={`https://github.com/${contributor.login}`}>
64                  {contributor.login}
65                </Button>
66              ))
67            }
68            <div className={styles.empty}>
69              <div className={styles.empty}/>
70              <Button className={styles.delete} icon={faTrashAlt} primary
                     confirmNeeded
71                        onClick={() => this.props.deleteFile(editingFile)}>
72                <Ellipsis>Delete File</Ellipsis>
73              </Button>
74            </div>
75          </div>
76        </div>
77      );
78    }
79  }
80
81  export default connect(({ current, env, player }) => ({ current, env,
       player }), actions, null, { forwardRef: true })(
82    CodeEditor,
83  );
```

algorithm-visualizer\src\components\Divider\index.js

```
import React from 'react';
import { classes } from 'common/util';
import styles from './Divider.module.scss';

class Divider extends React.Component {
  constructor(props) {
    super(props);

    this.handleMouseDown = this.handleMouseDown.bind(this);
    this.handleMouseMove = this.handleMouseMove.bind(this);
    this.handleMouseUp = this.handleMouseUp.bind(this);
  }

  handleMouseDown(e) {
    this.target = e.target;
    document.addEventListener('mousemove', this.handleMouseMove);
    document.addEventListener('mouseup', this.handleMouseUp);
  }

  handleMouseMove(e) {
    const { onResize } = this.props;
    if (onResize) onResize(this.target.parentElement, e.clientX, e.
        clientY);
  }

  handleMouseUp(e) {
    document.removeEventListener('mousemove', this.handleMouseMove);
    document.removeEventListener('mouseup', this.handleMouseUp);
  }

  render() {
    const { className, horizontal } = this.props;

    return (
      <div className={classes(styles.divider, horizontal ? styles.
          horizontal : styles.vertical, className)}
            onMouseDown={this.handleMouseDown} />
    );
  }
}

export default Divider;
```

algorithm-visualizer\src\components\VisualizationViewer\index.js

```
import React from 'react';
import { classes } from 'common/util';
import { Divider } from 'components';
```

```
4  import styles from './ResizableContainer.module.scss';
5
6  class ResizableContainer extends React.Component {
7    handleResize(prevIndex, index, targetElement, clientX, clientY) {
8      const { horizontal, visibles, onChangeWeights } = this.props;
9      const weights = [...this.props.weights];
10
11     const { left, top } = targetElement.getBoundingClientRect();
12     const { offsetWidth, offsetHeight } = targetElement.parentElement;
13     const position = horizontal ? clientX - left : clientY - top;
14     const containerSize = horizontal ? offsetWidth : offsetHeight;
15
16     let totalWeight = 0;
17     let subtotalWeight = 0;
18     weights.forEach((weight, i) => {
19       if (visibles && !visibles[i]) return;
20       totalWeight += weight;
21       if (i < index) subtotalWeight += weight;
22     });
23     const newWeight = position / containerSize * totalWeight;
24     let deltaWeight = newWeight - subtotalWeight;
25     deltaWeight = Math.max(deltaWeight, -weights[prevIndex]);
26     deltaWeight = Math.min(deltaWeight, weights[index]);
27     weights[prevIndex] += deltaWeight;
28     weights[index] -= deltaWeight;
29     onChangeWeights(weights);
30   }
31
32   render() {
33     const { className, children, horizontal, weights, visibles } = this.
         props;
34
35     const elements = [];
36     let lastIndex = -1;
37     const totalWeight = weights.filter((weight, i) => !visibles ||
         visibles[i])
38       .reduce((sumWeight, weight) => sumWeight + weight, 0);
39     children.forEach((child, i) => {
40       if (!visibles || visibles[i]) {
41         if (~lastIndex) {
42           const prevIndex = lastIndex;
43           elements.push(
44             <Divider key={`divider-${i}`} horizontal={horizontal}
45                      onResize={((target, dx, dy) => this.handleResize(
                           prevIndex, i, target, dx, dy))} />,
46           );
47         }
```

```
48          elements.push(
49            <div key={i} className={classes(styles.wrapper)} style={{
50              flexGrow: weights[i] / totalWeight,
51            }}>
52              {child}
53            </div>,
54          );
55          lastIndex = i;
56        }
57      });
58
59      return (
60        <div className={classes(styles.resizable_container, horizontal &&
              styles.horizontal, className)}>
61          {elements}
62        </div>
63      );
64    }
65  }
66
67  export default ResizableContainer;
```

Sliding Window.js

```
1  // import visualization libraries {
2  const { Tracer, Array1DTracer, LogTracer, Randomize, Layout,
       VerticalLayout } = require('algorithm-visualizer');
3  // }
4
5  // define tracer variables {
6  const tracer = new Array1DTracer();
7  const logger = new LogTracer();
8  Layout.setRoot(new VerticalLayout([tracer, logger]));
9  const D = Randomize.Array1D({ N: 20, value: () => Randomize.Integer({
       min: -5, max: 5 }) });
10 tracer.set(D);
11 Tracer.delay();
12 // }
13
14 let sum = D[0] + D[1] + D[2];
15 let max = sum;
16 // visualize {
17 tracer.select(0, 2);
18 logger.println(`sum = ${sum}`);
19 Tracer.delay();
20 // }
21 for (let i = 3; i < D.length; i++) {
22   sum += D[i] - D[i - 3];
```

```
23    if (max < sum) max = sum;
24    // visualize {
25    tracer.deselect(i - 3);
26    tracer.select(i);
27    logger.println(`sum = ${sum}`);
28    Tracer.delay();
29    // }
30  }
31  // visualize {
32  tracer.deselect(D.length - 3, D.length - 1);
33  logger.println(`max = ${max}`);
34  // }
```

Levenshtein's Edit Distance.js

```
1   // import visualization libraries {
2   const { Tracer, Array2DTracer, LogTracer, Layout, VerticalLayout } =
        require('algorithm-visualizer');
3   // }
4
5   const str1 = 'stack';
6   const str2 = 'racket';
7   const table = new Array(str1.length + 1);
8
9   for (let i = 0; i < str1.length + 1; i++) {
10    table[i] = new Array(str2.length + 1).fill(-1);
11    table[i][0] = i;
12  }
13  for (let i = 1; i < str2.length + 1; i++) {
14    table[0][i] = i;
15  }
16
17  // define tracer variables {
18  const tracer = new Array2DTracer('Distance Table');
19  const logger = new LogTracer();
20  Layout.setRoot(new VerticalLayout([tracer, logger]));
21  tracer.set(table);
22  Tracer.delay();
23  // }
24
25  // logger {
26  logger.println('Initialized DP Table');
27  logger.println(`Y-Axis (Top to Bottom): ${str1}`);
28  logger.println(`X-Axis (Left to Right): ${str2}`);
29  // }
30
31  const dist = (function editDistance(str1, str2, table) {
32    // display grid with words
```

```
33  // logger {
34  logger.println(`*** ${str2.split('').join(' ')}`);
35  table.forEach((item, index) => {
36    const character = (index === 0) ? '*' : str1[index - 1];
37    logger.println(`${character}\t${item}`);
38  });
39  // }
40
41  // begin ED execution
42  for (let i = 1; i < str1.length + 1; i++) {
43    for (let j = 1; j < str2.length + 1; j++) {
44      if (str1[i - 1] === str2[j - 1]) {
45        // visualize {
46        tracer.select(i - 1, j - 1);
47        Tracer.delay();
48        // }
49        table[i][j] = table[i - 1][j - 1];
50        // visualize {
51        tracer.patch(i, j, table[i][j]);
52        Tracer.delay();
53        tracer.depatch(i, j);
54        tracer.deselect(i - 1, j - 1);
55        // }
56      } else {
57        // visualize {
58        tracer.select(i - 1, j);
59        tracer.select(i, j - 1);
60        tracer.select(i - 1, j - 1);
61        Tracer.delay();
62        // }
63        table[i][j] = Math.min(table[i - 1][j], table[i][j - 1], table[i
              - 1][j - 1]) + 1;
64        // visualize {
65        tracer.patch(i, j, table[i][j]);
66        Tracer.delay();
67        tracer.depatch(i, j);
68        tracer.deselect(i - 1, j);
69        tracer.deselect(i, j - 1);
70        tracer.deselect(i - 1, j - 1);
71        // }
72      }
73    }
74  }
75
76  // visualize {
77  tracer.select(str1.length, str2.length);
78  // }
```

```
  return table[str1.length][str2.length];
}(str1, str2, table));

// logger {
logger.println(`Minimum Edit Distance: ${dist}`);
// }
```

Longest Common Subsequence.js

```
// import visualization libraries {
const { Tracer, Array1DTracer, Array2DTracer, LogTracer, Layout,
    VerticalLayout } = require('algorithm-visualizer');
// }

const string1 = 'AGGTAB';
const string2 = 'GXTXAYB';
const m = string1.length;
const n = string2.length;
const A = new Array(m + 1);
for (let i = 0; i < m + 1; i++) {
  A[i] = new Array(n + 1);
}

// define tracer variables {
const tracer1 = new Array1DTracer('String 1');
const tracer2 = new Array1DTracer('String 2');
const tracer3 = new Array2DTracer('Memo Table');
const logger = new LogTracer();
Layout.setRoot(new VerticalLayout([tracer1, tracer2, tracer3, logger]));
tracer1.set(string1);
tracer2.set(string2);
tracer3.set(A);
Tracer.delay();
// }

let i;
let j;

// Build the memo table in bottom up fashion
for (i = 0; i <= m; i++) {
  for (j = 0; j <= n; j++) {
    if (i === 0 || j === 0) {
      A[i][j] = 0;
    } else if (string1[i - 1] === string2[j - 1]) {
      // visualize {
      tracer1.select(i - 1);
      Tracer.delay();
      tracer2.select(j - 1);
```

```
39        Tracer.delay();
40        tracer3.select(i - 1, j - 1);
41        Tracer.delay();
42        // }
43
44        A[i][j] = A[i - 1][j - 1] + 1;
45
46        // visualize {
47        tracer1.deselect(i - 1);
48        tracer2.deselect(j - 1);
49        tracer3.deselect(i - 1, j - 1);
50        // }
51      } else {
52        // visualize {
53        tracer3.select(i - 1, j);
54        Tracer.delay();
55        tracer3.select(i, j - 1);
56        Tracer.delay();
57        // }
58
59        if (A[i - 1][j] > A[i][j - 1]) {
60          A[i][j] = A[i - 1][j];
61        } else {
62          A[i][j] = A[i][j - 1];
63        }
64
65        // visualize {
66        tracer3.deselect(i - 1, j);
67        tracer3.deselect(i, j - 1);
68        // }
69      }
70      // visualize {
71      tracer3.patch(i, j, A[i][j]);
72      Tracer.delay();
73      tracer3.depatch(i, j);
74      // }
75    }
76  }
77
78  let finalString = '';
79  i = m;
80  j = n;
81  while (i >= 1 && j >= 1) {
82    // visualize {
83    tracer3.select(i, j);
84    Tracer.delay();
85    // }
```

```
  if (string1[i - 1] === string2[j - 1]) {
    // visualize {
    tracer1.select(i - 1);
    Tracer.delay();
    tracer2.select(j - 1);
    Tracer.delay();
    // }

    finalString = string1[i - 1] + finalString;
    i--;
    j--;
  } else if (A[i - 1][j] > A[i][j - 1]) {
    i--;
  } else {
    j--;
  }
}

// logger {
logger.println(`Longest Common Subsequence Length is ${A[m][n]}`);
logger.println(`Longest Common Subsequence is ${finalString}`);
// }
```

Matrix Chain Multiplication.js

```
// import visualization libraries {
const { Tracer, Array1DTracer, Array2DTracer, LogTracer, Layout,
    VerticalLayout } = require('algorithm-visualizer');
// }

const dimensions = [10, 20, 30, 40, 30]; // the dimensions of matrices
    to be multiplied
const n = dimensions.length - 1;
const A = new Array(n);
for (let i = 0; i < n; i++) {
  A[i] = new Array(n).fill(0);
}

// define tracer variables {
const tracer1 = new Array1DTracer('Matrix Dimensions');
const tracer2 = new Array2DTracer('Memo Table');
const logger = new LogTracer();
Layout.setRoot(new VerticalLayout([tracer1, tracer2, logger]));
tracer1.set(dimensions);
tracer2.set(A);
Tracer.delay();
// }

```

```
22 let i;
23 let j;
24 let k;
25 let len;
26
27 // Build the memo table in bottom up fashion
28 for (len = 2; len <= n; len++) {
29   for (i = 1; i <= n - len + 1; i++) {
30     j = i + len - 1;
31     for (k = i; k <= j - 1; k++) {
32       // visualize {
33       tracer1.select(i - 1, j - 1);
34       Tracer.delay();
35       tracer2.select(i - 1, k - 1);
36       Tracer.delay();
37       tracer2.select(k, j - 1);
38       Tracer.delay();
39       // }
40
41       const q = A[i - 1][k - 1] + A[k][j - 1] + dimensions[i - 1] *
             dimensions[k] * dimensions[j];
42
43       // visualize {
44       tracer1.deselect(i - 1, j - 1);
45       tracer2.deselect(i - 1, k - 1);
46       tracer2.deselect(k, j - 1);
47       // }
48
49       if (A[i - 1][j - 1] === 0 || q < A[i - 1][j - 1]) {
50         A[i - 1][j - 1] = q;
51         // visualize {
52         tracer2.patch(i - 1, j - 1, A[i - 1][j - 1]);
53         Tracer.delay();
54         tracer2.depatch(i - 1, j - 1);
55         // }
56       }
57     }
58   }
59 }
60
61 // logger {
62 logger.println(`Minimum number of scalar multiplications required to
     multiply the matrices of dimensions ${dimensions.join('x')} is ${A
     [0][n-1]}`);
63 // }
```

Chapter 7

Testing

Table 7.1: Test Case

Test Id	Test Name	Input	Expected output	Actual output	Pass/Fail
[1]	Change Input	User can gives the input	Input has been changed.	Input has been changed.	Pass
[2]	Build Button	User can build the visualisation.	Visualisation has been build.	Visualisation has been build.	Pass
[3]	Play/pause Button	User can play/-pause the visualisation.	Play/pause button working properly.	Play/pause button working properly.	Pass
[4]	Backward/ Forward button	User can move to the previous/next step of visualisation.	Backward/ Forward button working properly.	Backward/ Forward button working properly.	Pass

Table 7.2: Test Case

[5]	Slider	User can change the animation speed.	Slider working properly.	Slider working properly.	Pass
[6]	Text Editor	User can change the code as per the requirement.	Code has been changed.	Code has been changed.	Pass
[7]	Display Visualisation	After building the code user can view the visualisation.	Visualisation is visible.	Visualisation is visible.	Pass

Chapter 8

Results and Discussions

8.1 Screenshots

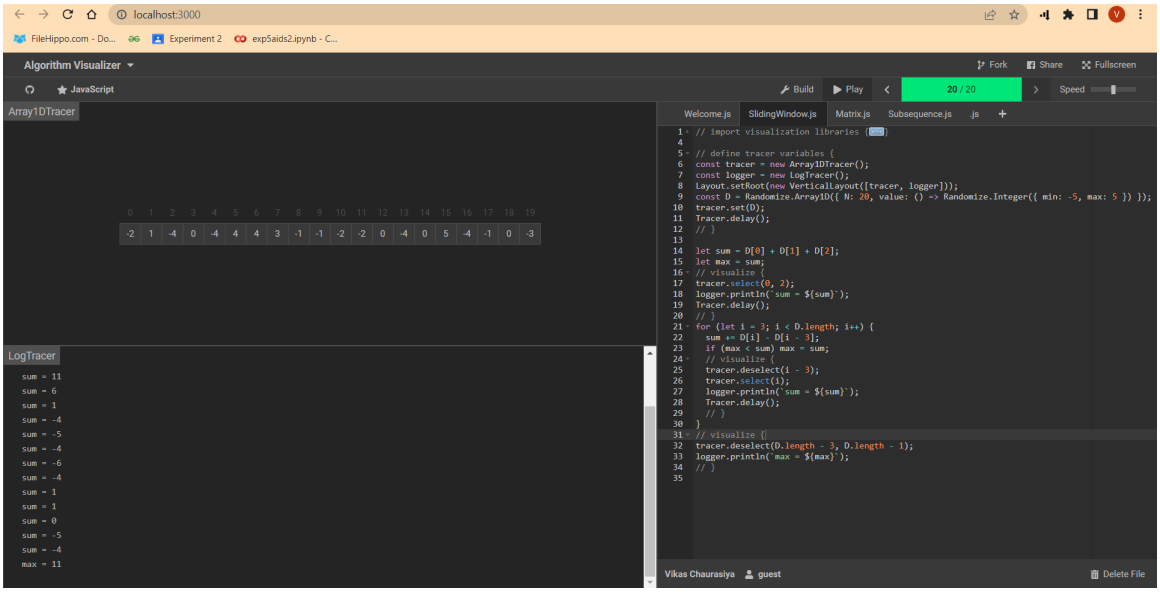

Figure 8.1: Sliding Window Visualization along with code.

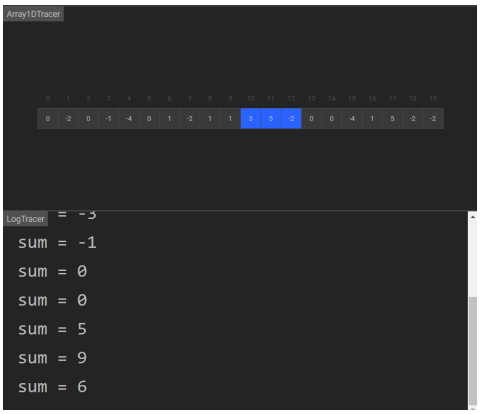

Figure 8.2: Sliding Window Visualization.

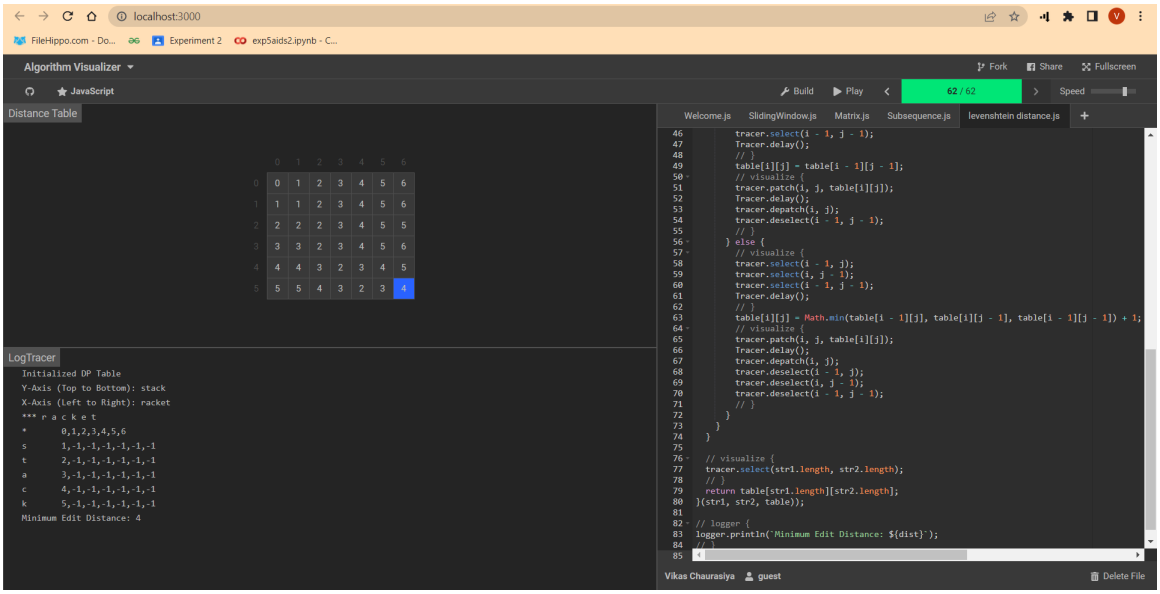

Figure 8.3: Levenshtein Distance Visualization along with code.

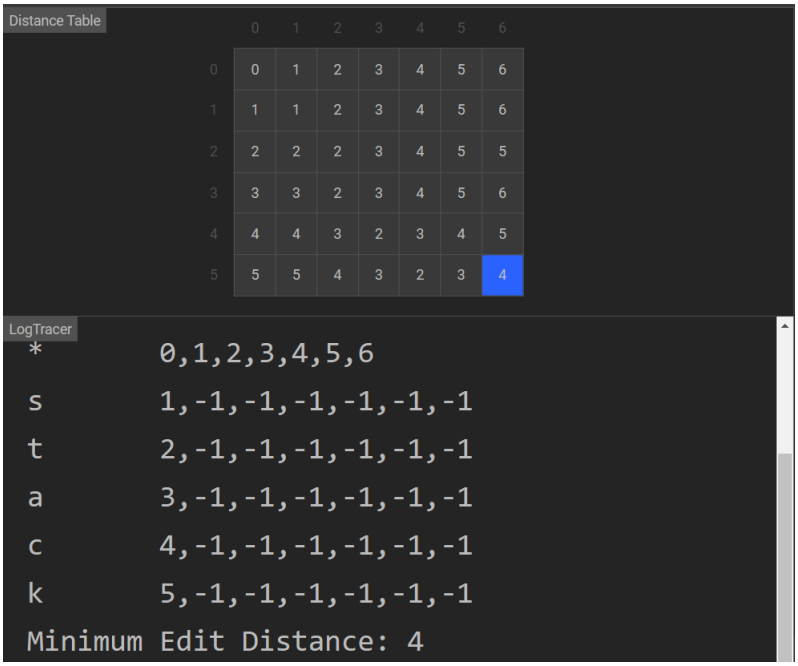

Figure 8.4: Levenshtein Distance Visualization.

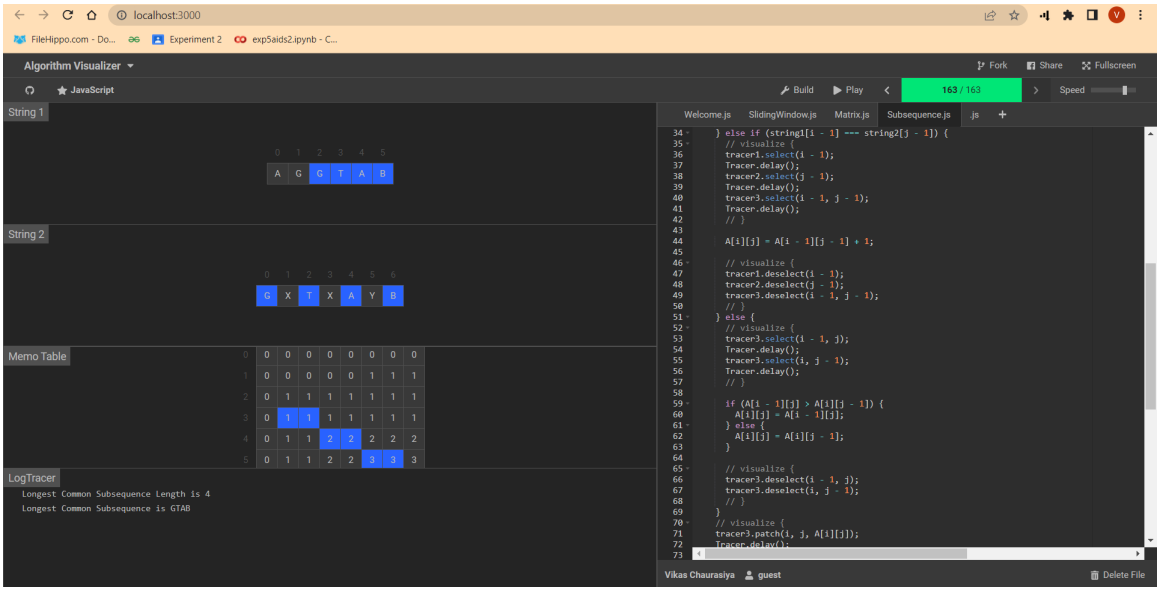

Figure 8.5: Longest Common Subsequence Visualization along with code.

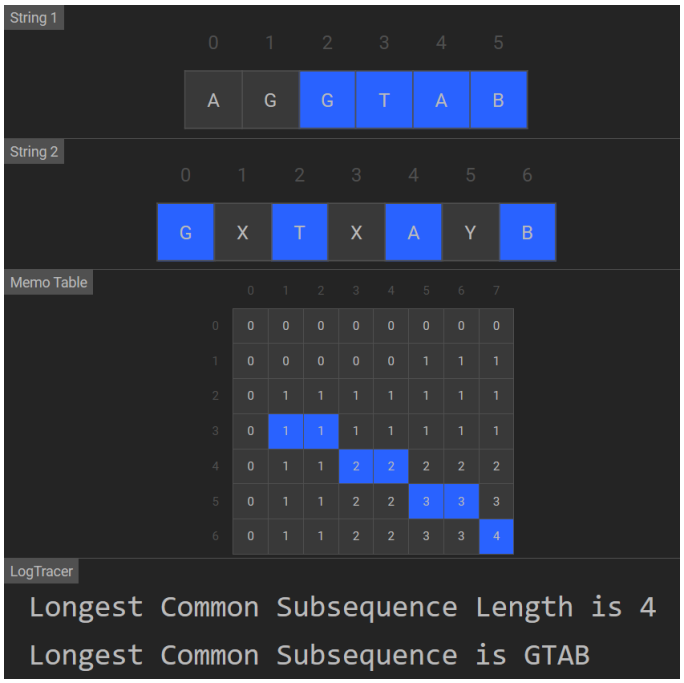

Figure 8.6: Longest Common Subsequence Visualization.

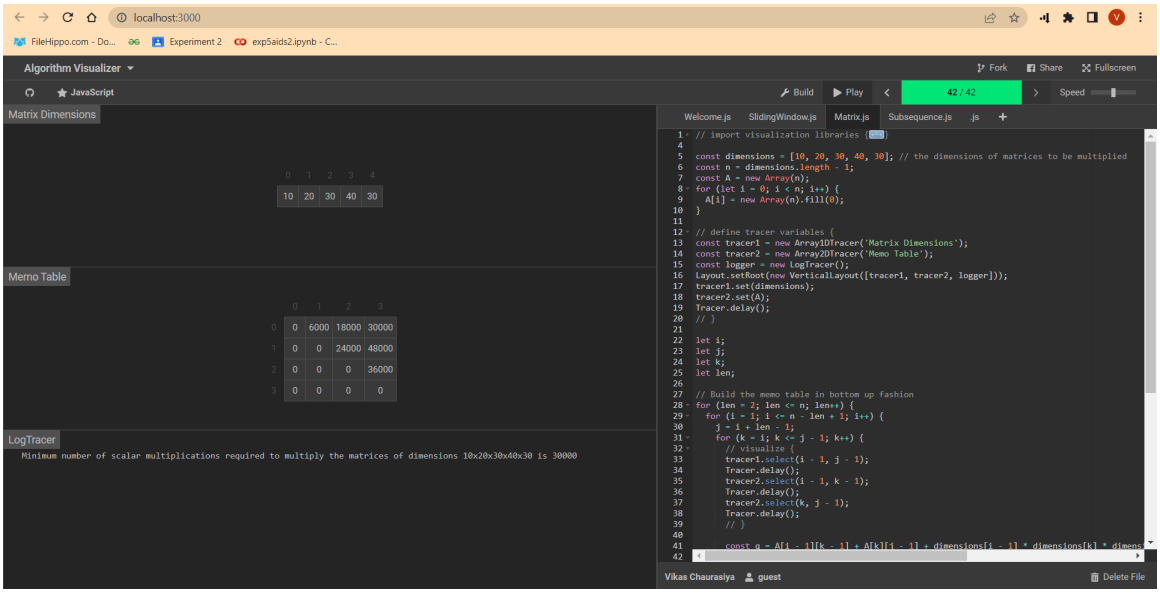

Figure 8.7: Matrix Chain Multiplication Visualization along with code.

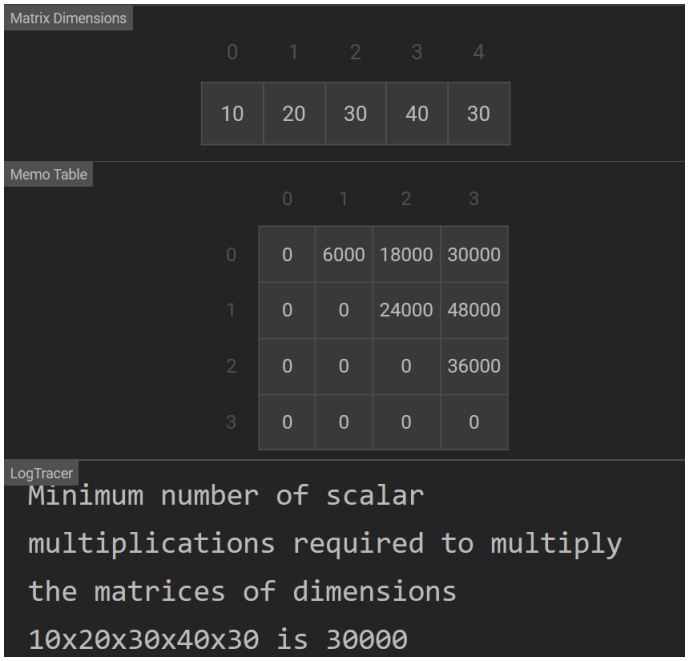

Figure 8.8: Matrix Chain Multiplication Visualization.

8.2 Performance Metrics

The performance metrics of an algorithm visualizer have been evaluated to assess its effectiveness in visualizing various algorithms. The visualizer was tested with four well-known algorithms - Longest Common Subsequence (LCS), Levenshtein Distance, Sliding Window, and Matrix Chain Multiplication - to determine its runtime and memory consumption. The results showed that the visualizer performed efficiently for all four algorithms. The LCS algorithm took 27ms to run and consumed 50.7MB of memory, while the Levenshtein Distance algorithm had a runtime of 6ms and consumed 42.7MB of memory. The Sliding Window algorithm had a slightly longer runtime of 38ms but still demonstrated efficient memory consumption at 57.1MB. The Matrix Chain Multiplication algorithm had an impressive runtime of 4ms and consumed 39.8MB of memory.

Table 8.1: Performance Metrics

Sr no	Algorithm	Run time	Memory Usage
[1]	Sliding Window	38ms	57.1MB
[2]	Levenshtein Distance	6ms	42.7MB
[3]	Longest Common Subsequence	27ms	50.7MB
[4]	Matrix Chain Multiplication	4ms	39.8MB

Chapter 9

Conclusion and Future Scope

9.1 Conclusion

Algorithm Visualizer is an essential tool for studying and understanding the functionality of algorithms in the field of computer science. Our implementation of algorithms in the visualizer provides a better understanding of the algorithms and can aid in building efficient software. Additionally, algorithm visualizers can aid in teaching and learning, as they provide a hands-on approach to understanding complex algorithms. Overall, incorporating an algorithm visualizer into the development process can lead to more efficient and effective algorithms, as well as a deeper understanding of how they work.

9.2 Future Scope

The proposed algorithm visualizer has promising potential for future development. The current version of the visualizer has a user-friendly interface, making it easy to use for beginners and experts alike. The ability to add more algorithms in the future is a significant advantage, as it provides flexibility for the tool to grow and expand. Hosting the visualizer on the internet and making it available to everyone would be a significant step forward in the field of algorithm visualization, as it would make the tool easily accessible to anyone with an internet connection. Improving the user interface is also an essential aspect of future development, as it can increase the user's engagement and overall experience while using the tool. Finally, adding more features, such as allowing users to create their algorithms and share them with others, can help take the visualizer to the next level. Thus, algorithm visualizer has a bright future ahead, with the potential to become an indispensable tool in the field of algorithm visualization.

Literature Cited

[1] Fan, Xiaodi, Angel Saldivia, Pedro Soto, and Jun Li, "Coded Matrix Chain Multiplication.", *IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, 2021, pp. 1-6.

[2] Gondree, Mark & Mohassel, Payman, "Longest Common Subsequence as Private Search.", *Proceedings of the ACM Conference on Computer and Communications Security*, 2009, pp. 81-90.

[3] Pedro Moraes and Leopoldo Teixeira, "Willow: A Tool for Interactive Programming Visualization to Help in the Data Structures and Algorithms Teaching-Learning Process", *SBES 2019: Proceedings of the XXXIII Brazilian Symposium on Software Engineering.* , 2019, pp . 553-558.

[4] Q. Gao and X. Xu, "The analysis and research on computational complexity",*The 26th Chinese Control and Decision Conference, (CCDC)*, 2014, pp. 3467-3472.

[5] Borassi, Michele & Epasto, Alessandro & Lattanzi, Silvio & Vassilvitskii, Sergei & Zadimoghaddam, Morteza, "Sliding Window Algorithms for k-Clustering Problems." *Advances in Neural Information Processing Systems, 33*, 2020, pp.8716-8727.

[6] Haldar, Rishin & Mukhopadhyay, Debajyoti, "Levenshtein Distance Technique in Dictionary Lookup Methods: An Improved Approach.", *Computing Research Repository, (CORR)*,2011,pp.1101-1232

[7] Clifford A. Shaffer Matthew L. Cooper Alexander Joel D. Alon Monika Akbar Michael Stewart Sean Ponce et al, "Algorithm Visualization: The State of the Field", *ACM Transactions on Computing Education*,vol. 10 no. 3, 2010.

[8] V. Karavirta and C. A. Shaffer, "Creating Engaging Online Learning Material with the JSAV JavaScript Algorithm Visualization Library," *IEEE Transactions on Learning Technologies*, vol. 9, no. 2, pp. 171-183, 2016.

[9] Jay Talekar, Jugal Suthar, Sanket Joshi and Jignesh Patel,"Review of Algorithm Visualization Methodologies" *International Research Journal of Modernization in Engineering Technology and Science*, vol. 04, Issue. 04, April-2022.

[10] Tao Chen and T. Sobh, "A tool for data structure visualization and user-defined algorithm animation," *31st Annual Frontiers in Education Conference. Impact on Engineering and Science Education. Conference Proceedings*, 2001, pp. TID-2.

[11] B. Goswami, A. Dhar, A. Gupta and A. Gupta, ”Algorithm Visualizer: Its features and working,”*2021 IEEE 8th Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON)*, 2021, pp. 1-5.

[12] Shubham Nath, Jatin Gupta, Abhinav Gupta and Teena Verma,”Sorting Algorithm Visualizer”,*International Research Journal of Modernization in Engineering Technology and Science*, 2021 Volume:03/Issue:07.

Acknowledgement

We are extremely grateful to our college St. Francis Institute of Technology for the confidence bestowed in us and entrusting out project entitled ”Algorithm Visualizer”. We express our sincere gratitude to our respected director Bro. Shantilal Kujur, our principal Dr. Sincy George and our HOD Dr. Prachi Raut for encouragement and facilities provided to us. We owe our profound gratitude to our project guide Ms. Alvina Alphonso , who introduced us to the methodology of work and timely completion of various stages of our project. Many people especially all team members and classmates have made valuable comments on this proposal which gave us inspiration to improve our assignment.