International Journal of Scientific Research & Engineering Trends
Volume 12, Issue 4, Jul-Aug-2026, ISSN (Online): 2395-566X

Learning System for Tree Traversal Algorithms: An Enhanced Algorithm Visualization Tool

Samir Yau Nuhu[1], Jamilu Awwalu[2], Zaharaddeen Salele Iro[3], Zaharaddeen Sufyanu[4]
[1]Department of Computer Science Federal University Dutse
[2]Department: Computer Science Department, Federal University Dutse, Dutse 720211, Nigeria
[3]Department: Cybersecurity Department, Federal University Dutse, Dutse 720211, Nigeria
[4]Department: Computer Science Department, Federal University Dutse, Dutse 720211, Nigeria

Abstract- Tree traversal algorithms form a major part of undergraduate computer science education because they teach important concepts about how data is organized and processed in trees. However, students find these algorithms difficult to understand since they are abstract and involve following specific rules to visit each node in a particular sequence. Existing visualization tools fail to help students fully because they do not clearly separate or explain the differences between the three main Depth-First Search variants, and they do not show the step-by-step order of visited nodes in a simple and structured format. To solve these problems, this thesis presents a new interactive learning tool that runs directly in a web browser and is built using HTML, CSS, JavaScript and SVG. The tool allows students to create their own binary trees by adding nodes, gives them full control to move through each step of any traversal method, and includes a table-generation feature that automatically builds and displays a clear list showing exactly which nodes are visited and in what order. We carried out an analysis of the system's algorithms and showed that it remains efficient and scalable even when working with larger tree. In addition, we tested the tool in a real classroom setting with 100 undergraduate computer science students. Before using the tool, their average score on a test about tree traversals was only 40%. After use of the tool their average score increased to 68%. These results clearly demonstrate that the new system overcomes the main weaknesses of earlier visualization tools and helps students gain a better understanding of tree traversal algorithms.

Keywords- Tree, Traversal Algorithm, Breadth-First Search, Depth-First Search.

I. INTRODUCTION

In computer Science, tree is a hierarchical data structure that is made up of multiple nodes, connected by one or more edges [1]. It is one of the most important data structures in computer science [2]. It has its terminologies, types, operations as well as different algorithms for visiting its nodes. These make it a wider topic under data structures in Computer Science. However, understanding data structures and algorithms, particularly tree traversal algorithms such as Breadth-First Search (BFS) and Depth-First Search (DFS), is fundamental in computer science. These algorithms form the basis for solving complex computational problems, but they remain challenging for many students to master due to their abstract nature [3]. Traditional instructional approaches, including static diagrams, lectures, and textbooks, are not sufficient enough to convey the abstract nature of these algorithms [4].

To address these challenges, computer science educators and researchers have encouraged the use of visualization tools to aid in teaching data structures and algorithms. Visualization enables learners to interact with graphical representations of abstract concepts, for better understanding. Recent studies demonstrate that interactive visualizations not only improve students' understanding but also enhance their motivation and engagement with the subject matter.

However, despite the existence of several data structure visualization tools, limitations persist. Many tools are generalized, like focusing broadly on various data structures without taking target. Still, usability concerns and scalability issues in the existing systems reduce their effectiveness [5].

Recent advancements in Intelligent Tutoring Systems (ITS) and Adaptive Learning Technologies provide opportunities to enhance algorithm learning. Research shows that integrating

adaptive feedback, real-time assessments, and user-friendly interfaces can significantly improve student performance in algorithm-based tasks [6]. But despite these advancements, few systems target the step-by-step teaching of tree traversal algorithms through interactive visualization, real-time feedback, and learner-controlled tasks.

The present study addresses these limitations and gap, this research proposes the design, development, and evaluation of a web-based tree traversal tutoring system. The system integrates interactive tree creation, animated visualization of BFS and DFS algorithms, real-time traversal feedback, and a user-friendly interface to promote active learning of tree traversal processes. In line with recent research trends, the proposed system is upon established on the design principles of simplicity, interactivity and visual clarity, while using features that solve known limitations in current visualization tools.

The evaluation was conducted using undergraduate Computer Science students through a pre-test and post-test experimental design to determine the educational effectiveness of the proposed system. The study is limited to binary tree traversal algorithms and does not cover advanced tree structures such as AVL trees, B-trees, red-black trees or graph traversal algorithms beyond the implemented BFS and DFS operations. The findings are expected to contribute to algorithm education by providing a practical, accessible and platform-independent learning environment that enhances conceptual understanding and supports improved academic performance.

Related Works

Traversal algorithms are fundamental part of computer science. Two of the most common and widely used traversal algorithms are Breadth-First Search (BFS) and Depth-First Search (DFS). This literature review aims to provide an overview of tree data structure and its traversal algorithms, together with their principles and applications.

Tree Data Structure

A tree is a fundamental non-linear data structure that is used to represent hierarchical relationships. Technically, it is a finite set of nodes that is either empty or consists of a distinct node called the root, along with zero or more subtrees, each of which is itself a tree. Each may have multiple child nodes, forming a structure that is acyclic, rooted, and recursive [7].

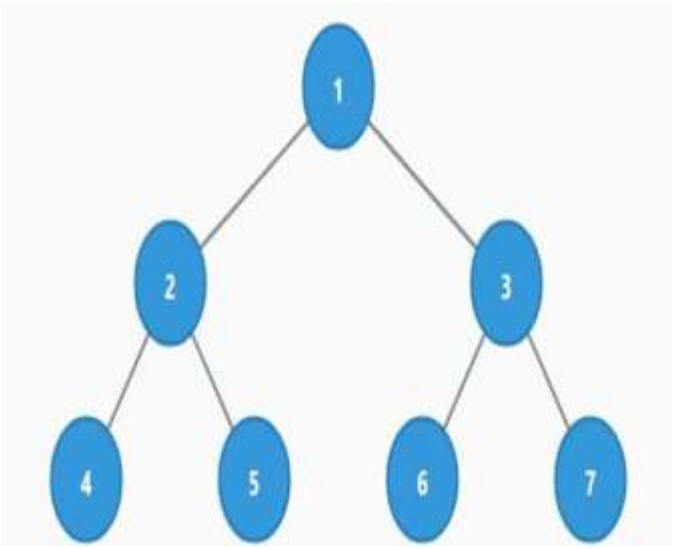

Fig. 2.0: Diagram of tree data structure

Trees are characterized by their properties like:

1. Root: the topmost node,
2. Edges: connections between nodes,
3. Parent/Child: hierarchical relationships,
4. Height and Depth: levels within the structure,
5. Leaf nodes: nodes without children.

Tree Traversal Algorithms (BFS & DFS)

Tree traversal is the algorithmic process of visiting every node in a tree exactly once according to a specified order, typically to inspect, compute on, or update node-associated data [1]. These are:

Breadth-First Search (BFS)

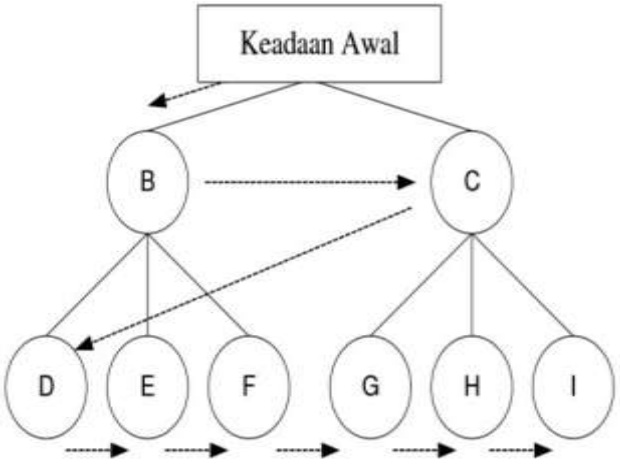

Figure 2.1: A Working Procedure of BFS

Depth-First Search (DFS)

DFS is another fundamental tree traversal algorithm that explores a tree by going as deep as possible along each branch before backtracking. It starts at the root node and explores one branch completely before moving to the next branch. DFS utilizes a stack data to keep track of the nodes to be visited. DFS is often used in problems where the goal is to explore all possible paths in a tree or tree, such as finding cycles.

Table 2.1 Comparison of Breadth-First Search and Depth-First Search

Feature	Breadth-First Search	Depth-First Search
Traversal Strategy	Level-by-level	Branch-by-branch
Data Structure	Queue	Stack/Recursion
Time Complexity		
Space Complexity		
Common Applications	Shortest path, routing	Tree processing, recursion
Main Strength	Complete level exploration	Lower memory in balanced trees
Main Limitation	Higher memory usage	Deep recursion in skewed trees

DFS Variations

DFS has three distinct variants, each following the same recursive pattern but with different traversal orders. These are:

1. **DFS Pre-order:** is a recursive tree traversal algorithm that follows the pattern: Node → Left → Right. Its operation:
2. Base Case: Check if the current node exists (not null)
3. Process First: Visit/process the current node immediately
4. Go Left: Recursively traverse the left subtree
5. Go Right: Recursively traverse the right subtree.

```
FUNCTION inorder(node):
    IF node exists:
        inorder(node.left)          // Go left first
        PROCESS node                // Visit root in middle
        inorder(node.right)         // Then go right
```

Fig. 2.2: DFS (Pre-order)

2. DFS In-order: is a recursive tree traversal algorithm that follows the pattern. Left → Node → Right. Its operation:

- **Base Case:** Check if the current node exists (not null)
- **Go Left First:** Recursively traverse the left subtree completely
- **Process Middle:** Visit/process the current node after left subtree is done
- **Go Right Last:** Recursively traverse the right subtree. as illustrated below:

```
FUNCTION inorder(node):
    IF node exists:
        inorder(node.left)          // Go left first
        PROCESS node                // Visit root in middle
        inorder(node.right)         // Then go right
```

Fig. 2.3: DFS (In-order)

3. DFS Post-order: is a recursive tree traversal algorithm that follows the pattern Left → Right → Node. Its operation:

- **Base Case:** Check if the current node exists (not null)
- **Go Left First:** Recursively traverse the left subtree completely
- **Go Right Second:** Recursively traverse the right subtree completely
- **Process Last:** Visit/process the current node after both subtrees are done [9].

```
FUNCTION postorder(node):
    IF node exists:
        postorder(node.left)        // Go left first
        postorder(node.right)       // Then go right
        PROCESS node                // Visit root last
```

Fig. 2.4: DFS (Post-order)

Table 2.2 Summary of Pre-order Traversal Characteristics

Feature	**Description**
Traversal Order	Root → Left → Right (NLR)
Traversal Category	Depth-First Search (DFS)
Data Structure	Stack or Recursion

Feature	Description
Time Complexity	
Space Complexity	
Common Applications	Tree copying, prefix notation, compiler design, XML processing
Main Advantage	Processes parent nodes before child nodes
Main Limitation	Recursive execution may be difficult for beginners to understand

Algorithmic Analysis

Algorithmic analysis is the process of studying the behavior of an algorithm to understand:

How much time it will take to run (time complexity)?

How much memory it will use (space complexity)?

How these factors grow as the input size increases?

Algorithmic analysis helps us to compare algorithms and choose the right approach for constraints like memory, speed, and power usage. When analyzing algorithms, focusing on how they scale is crucial. For example:

An algorithm that takes n^2 steps might be fine for small datasets.

But if n = 1,000,000, could be difficult to analyze.

The Big O Notation: It describes the upper bound of an algorithm's growth rate. Some common complexities:

Table 2.2 Common Complexities

Big O	Name	Growth Behavior
O(1)	Constant	Doesn't grow with input size
O(log n)	Logarithmic	Grows slowly, great for large inputs
O(n)	Linear	Grows proportionally with input size
O(n log n)	Log-linear	Common in efficient sorting algorithms
$O(n^2)$	Quadratic	Grows fast
$O(2^n)$	Exponential	Grows ridiculously fast
O(n!)	Factorial	Even worse than exponential.

Types of Analysis

Time Complexity: measures how the execution time of an algorithm grows as the input size (n) increases. It focuses on operations count, not the actual seconds on a clock.

Space complexity: measures how much extra memory an algorithm needs in addition to the input. It includes:

Variables

Data structures created during execution

Call stack (especially for recursion)

III. METHODOLOGY

The proposed system will be an interactive and platform-independent system for teaching and learning tree traversal algorithms, particularly Breadth-First Search (BFS) and Depth-First Search (DFS). The following theoretical tools and techniques will be used in the modelling and design of the system:

- Flowcharts
- Diagrams & tables
- Pseudocodes

Fig. 3.1, illustrate the flowchart of the proposed system. It will allow user to: generate tree, select a particular traversal algorithm and see the traversal both as a graphical tree diagram and as a sequential table of results. The steps involve:

1. **Initialization:** the system will begin by calling renderTree(). At this point, the system will check whether nodes exist (i.e., whether the user has provided or generated a tree input).
2. **Tree Validation:** if no nodes exist: the system will terminate (END). If nodes exist: the system will proceed to process and visualize them.
3. **Tree Visualization**: the visualization will have two sub-processes:

- **Node Rendering**: the system will draw circles to represent nodes. Each node is labeled with its corresponding value (number or character). This is will be handle by drawNodes(nodes).
- **Edge Rendering:** edges (connections between parent and child nodes) will be drawn. This will be handle by drawEdges(node).

4. **Coordinate Assignment:** to prevent overlapping and ensure a clear layout, the system will assign (X, Y) coordinates to each node. This will be done by calculatePosition(node, X, Y), which will typically: Places

the root node at the top-center. Recursively calculates child node positions relative to their parent, to ensure balanced spacing for left and right subtrees.

5. **Algorithm Execution:** user will select an algorithm (Pre-order, In-order, Post-order or BFS). The chosen algorithm will be executed on the tree structure. This produces an ordered traversal result, which will later be displayed in both graphical (tree) and tabular (sequence) formats.
6. **Result Generation**: after visualization: a tree diagram will be displayed to the user. A result table will also be generated, which will show the traversal sequence produced by the chosen algorithm.
7. **Termination:** once the results (visual + tabular) are displayed, the process ends.

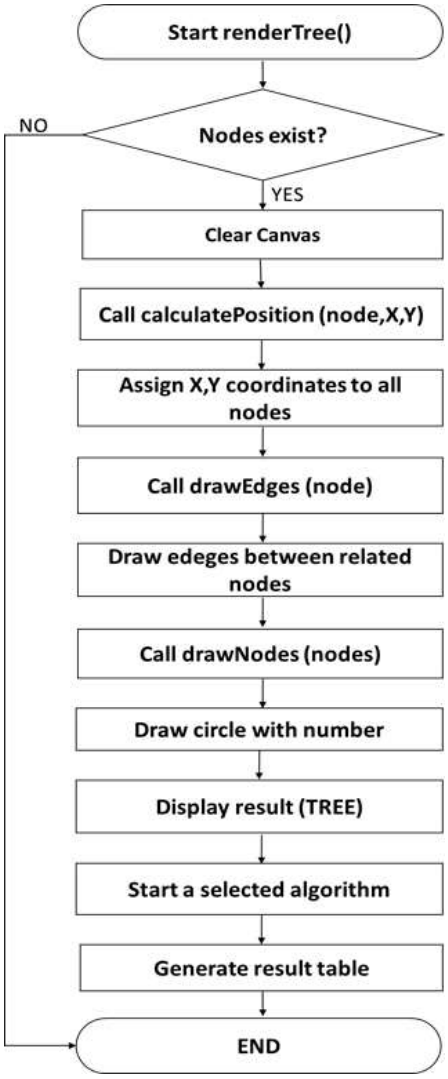

Fig. 3.0: Flowchart of the system

Use Case Diagram

Fig. 3.9, represent the use case diagram of the system. Everything inside the rectangle labeled Tutoring System represents the system's functions available to the students. Brief explanation:

- **Actor:** The actor is the Student, who interacts with the tutoring system.
- **Use Case 1:** Create tree: The student can generate a binary tree. This can be done in two ways: Random or Manual.
- **Use Case 2:** Learn: Once a tree is created, the student can learn traversal algorithms by selecting one: BFS or DFS (preorder, inorder, postorder). Pause/Resume: Control the animation of the traversal.

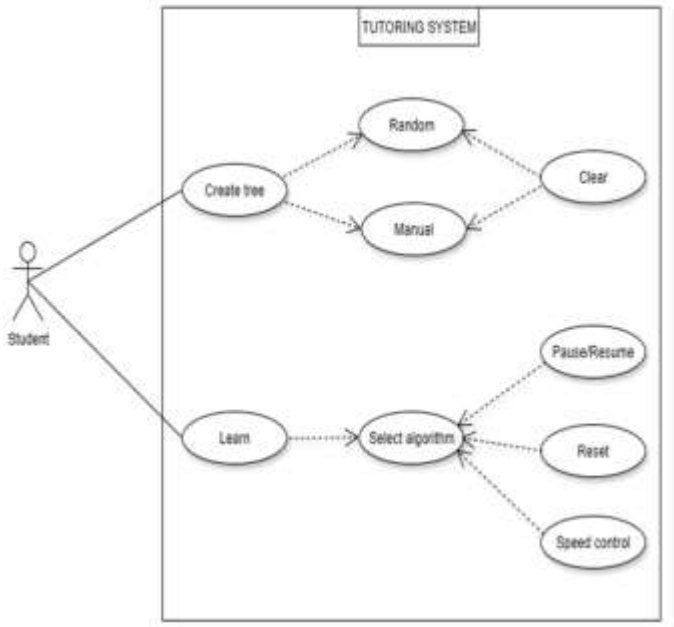

Fig. 3.1: Use case diagram of the system

System Requirements Specification

The proposed browser-based tutoring system was designed to satisfy both functional and non-functional requirements.

Functional Requirements

The system shall:
Allow users to register and log in securely.
Enable learners to construct binary trees interactively.
Support Breadth-First Search (BFS) traversal.
Support Depth-First Search (DFS) traversal.
Execute pre-order, in-order and post-order traversals.

Display animated traversal sequences.
Generate traversal outputs automatically.
Provide immediate feedback after algorithm execution.
Store learners' assessment records.
Allow administrators to manage learning content and user accounts.

Non-Functional Requirements

The system shall provide:
High usability through a user-friendly interface.
Fast response time during algorithm execution.
Reliability and consistent system performance.
Security through authenticated user access.
Compatibility with modern web browsers.
Scalability for future enhancements.
Maintainability through modular software design.

Software and Hardware Requirements

The successful operation of the proposed tutoring system depends on appropriate software and hardware resources.

Table 3.1: Software and Hardware Requirements

Category	Specification
Operating System	Windows 10/11 or Linux
Programming Languages	HTML5, CSS3, JavaScript, PHP
Database	MySQL
Web Server	Apache (XAMPP)
IDE	Visual Studio Code
Browser	Chrome, Firefox, Edge
Processor	Intel Core i3 or higher
RAM	Minimum 4 GB
Storage	Minimum 120 GB

Methods of Data Analysis

The data collected during the experimental evaluation were analysed using both descriptive and inferential statistical techniques. Descriptive statistics were employed to summarise participants' demographic characteristics and learning performance, while inferential statistics were used to determine whether the browser-based interactive tutoring system significantly improved students' understanding of binary tree traversal algorithms.

The analysis was performed using Statistical Package for the Social Sciences (SPSS) Version 27 and Microsoft Excel 2021. The results were presented using frequency tables, percentages, means, standard deviations, charts and hypothesis testing. Statistical significance was determined at the 0.05 significance level.

Descriptive Statistical Analysis

Descriptive statistics were used to summarise the characteristics of the respondents and describe their learning performance before and after using the proposed tutoring system. Frequency distributions and percentages were used to analyse demographic variables such as gender, age group and academic level, while measures of central tendency and dispersion were used to summarise assessment scores. The percentage of respondents was calculated using $P=f/N\times100$

Where:
P = Percentage
f = Frequency of responses
N = Total number of respondents
The arithmetic mean was computed as

$$\bar{X} = \frac{\sum X}{N}$$

Where:
$\bar{X}$ = Mean score
$\sum X$= Sum of observations
N = Number of observations
The standard deviation was calculated as

$$SD = \sqrt{\frac{\sum(X - \bar{X})^2}{N - 1}}$$

Where: SD = Standard deviation
X = Individual observation
SD = Mean
N = Sample size
These descriptive measures provided an overview of students' performance and facilitated comparison between the pre-test and post-test results.

Inferential Statistical Analysis

Inferential statistics were employed to determine whether the observed differences between students' pre-test and post-test scores were statistically significant. Since the same participants completed both assessments, the paired-samples t-test was adopted to compare the mean scores obtained before and after using the tutoring system.

The paired-samples t-test statistic was computed as

$$t = \frac{\bar{d}}{S_d/\sqrt{n}}$$

Where: t = Paired t-test statistic
d¯ = Mean difference between paired observations
S_d = Standard deviation of the differences
n = Number of paired observations

The decision rule was as follows:
Reject the null hypothesis () if the p-value < 0.05.
Fail to reject the null hypothesis if the p-value ≥ 0.05.
A statistically significant result indicates that the browser-based interactive tutoring system had a measurable effect on students' learning performance.
Where appropriate, the effect size was also determined using Cohen's d to evaluate the practical significance of the observed improvement.

$$d = \frac{\overline{X_{post}} - \overline{X_{pre}}}{SD_{pooled}}$$

Where:
d = Cohen's effect size
$\overline{X_{post}}$ = Mean post-test score

$\overline{X_{pre}}$ = Mean pre-test score

$\overline{X_{pre}}$= Pooled standard deviation

The effect size was interpreted as:
0.20 = Small effect
0.50 = Medium effect
0.80 or above = Large effect

IV. RESULTS AND DISCUSSION

This chapter presents system implementation details, algorithmic analysis, comparative evaluation of tree and table generation processes, and findings from the experimental study. The goal is to validate the effectiveness of the proposed browser-based tutoring system for tree traversal algorithms in terms of scalability and educational impact.

The Developed System
The developed system was designed and implemented as a browser-based interactive tutoring tool using web technologies—HTML for structural layout, CSS for interface styling, and JavaScript for functionality like algorithm execution. The frontend layout was structured into two main sections: a control panel for creating and manipulating trees, and a visualization area that renders the binary tree using SVG elements. CSS was used to provide a modern, responsive interface, incorporating components such as tabs, buttons, sliders, progress bars, and color-coded legends to achieve our goals.

At the development level, the system was architected around two custom JavaScript classes: TreeNode, which models the structure and attributes of each tree node, and TreeTutorSystem, which encapsulates all system behaviors, including tree construction, coordinate calculation, SVG rendering, and traversal animation. The system automatically converts user-supplied comma-separated values into a binary-tree array representation and then maps this structure into visual nodes and edges using SVG. Traversal algorithms (BFS, DFS pre-order, in-order, and post-order) were implemented as separate methods that generate ordered step sequences. These sequences are animated in real time, with each step updating node colors (current, visited, queued), edge states, progress indicators, and a step-by-step event. Additional interactive features—including adjustable animation speed, pausing and resuming, random tree generation, and automatic resetting—were also integrated.

Analysis of the main components of the Tutoring System
We started with the algorithmic analysis of the main system components. Table 4.1 shows the Big O analysis for the core components: Tree Rendering, Traversal Execution and Table Generation are all O(x) in time; which means they executed each node once. NB: 'x' is the number of nodes;

Table 4.1: Algorithmic Time and Space Complexity Analysis of the Core Components

Component	Operation	Time Complexity (T) Space Comp. (S)	
Tree Generation	Tree Rendering	O(x)	O(x)
Traversal Anim.	Algorithm Execu.	O(x)	O(x)
Tabular Const.	Table Row Gene.	O(x)	O(x)

Table 4.2: Key Performance Indicators of the main components of the tutoring tool

System components	Primary function	Educational benefit (KPI) Rating	
Tree Generation	Converts input data into visual tree structure (SVG).	Provides an immediate, accurate representation of data relationships.	4.2 / 5.0
Traversal Animation	Step-by-step visualization of BFS and DFS algorithms.	Enhances conceptual understanding of algorithmic flow.	4.8 / 5.0
Tabular Representation	Generates a final, consolidated summary of the traversal.	Reinforces learning through structured, textual	4.4 / 5.0

Figure 4.1: Key Performance Indicators

Comparison between Tree Generation and Table Generation

As we mentioned in chapter three that we are going to compare tree generation and table construction, below is the results of comparison between the two.

Table 4.3: Performance Metrics for Tree vs. Table Generation

No. of Nodes	Tree Rendering Time (ms)	Table Gen. Time (ms)	Performance Diff.
7	12.5	3.2	9.3
9	58.7	15.6	43.1
11	115.1	30.1	85.0
15	280.9	75.5	205.4
23	560.4	150.3	410.1

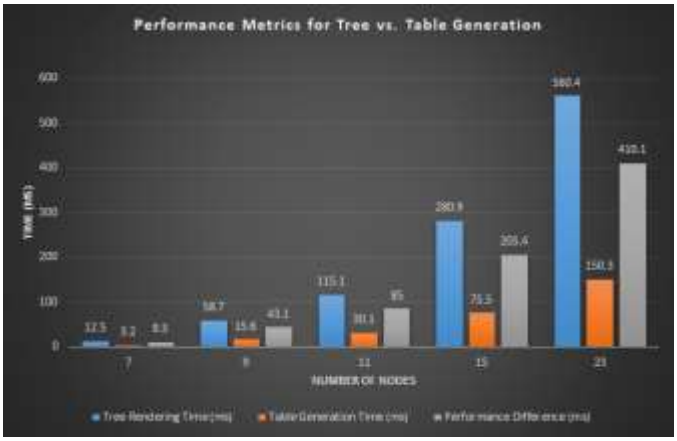

Figure 4.2: Performance Metrics for Tree vs. Table Generation

Table 4.3 and Figure 4.2 give measured timings: tree rendering goes from 12.5 ms at x=7 to 560.4 ms at x=23, while table generation goes from 3.2 ms to 150.3 ms. Although both processes are O(x), the per-node wall-clock cost grows with x due to animation.

Analysis of user experiment

The user experiment was done to evaluate the effectiveness of the browser-based tutoring system.

Table 4.4: Pre-Test Score Distribution (Student Frequency, N=100)

Score Interval (%)	Frequency	Percentage of Students
0 - 30 (Low)	20	20.0%
31 - 40 (Moderate)	45	45.0%
41 - 50 (High)	30	30.0%
51 - 100 (Very High)	5	5.0%
Total	100	100.0%

Table 4.4 shows the pre-test score distribution: 20% low (0–30), 45% moderate (31–40), 30% high (41–50), 5% very high.

Table 4.5: Post-Test Score Distribution (Student Frequency, N=100)

Score Interval (%)	Frequency	Percentage of Students
0 - 50 (Retained Low)	8	8.0%
51 - 60 (Moderate)	22	22.0%

61 - 70 (High)	35	35.0%
71 - 100 (Very High)	35	35.0%
Total	100	100.0%

Table 4.5 shows the post-test distribution: only 8% remain low, 22% moderate, 35% high and 35% very high. Those two tables together show a clear rightward shift in performance after the teaching session using the system.

Table 4.6: Pre-Test and Post-Test Score Summary Statistics (N=100)

Metric	Pre-Test Score	Post-Test Score	Improvement
Mean Score (%)	40.0	68.0	28.0 points
Median Score (%)	39.5	69.0	29.5 points
Standard Deviation	8.5	7.2	-1.3
Maximum Score (%)	55.0	85.0	30.0 points

Discussion

This section presents the results of the developed browser-based tutoring system and validates its effectiveness through algorithmic analysis and a user experiment with 100 undergraduate students. Technically, the system successfully delivers interactive tree construction, animated visualization of BFS and all three DFS variants (pre-order, in-order, post-order), full learner control, and a novel structured table output, with all core components achieving O(n) time and space complexity. The developed system was designed and implemented as a browser-based interactive tutoring tool using web technologies, below are some screenshots from the system. With these extensions, the tutoring system has the potential to evolve into widely adopted educational resource for data structures and algorithms.

Analysis of user experiment

The user experiment result in Table 4.6 was done to evaluate the effectiveness of the browser-based tutoring system.

Table 4.7: Pre-Test and Post-Test Score Summary Statistics (N=100)

Metric	Pre-Test Score	Post-Test Score	Improvement
Mean Score (%)	40.0	68.0	28.0 points
Median Score (%)	39.5	69.0	29.5 points
Standard Deviation	8.5	7.2	-1.3
Maximum Score (%)	55.0	85.0	30.0 points

Source: Experimental Results (2026).

Descriptive Statistics of Students' Performance

This section presents the descriptive statistics of students' pre-test and post-test scores obtained during the experimental evaluation of the browser-based interactive tutoring system. The analysis includes the mean, median and standard deviation of students' scores before and after interacting with the developed system. These statistics provide an overview of students' learning performance and indicate the extent of improvement achieved following the intervention.

Table 4.1 presents the descriptive statistics of students' academic performance before and after using the browser-based interactive tutoring system. The results indicate that the mean score increased from 40.0 in the pre-test to 68.0 in the post-test, representing a substantial improvement in students' understanding of binary tree traversal algorithms after the intervention. Similarly, the median score increased from 39.5 to 69.0, suggesting that the improvement was consistent across the participants and not limited to a few students.

The standard deviation increased slightly from 8.5 in the pre-test to 7.20 in the post-test. This indicates that although students' overall performance improved considerably, there was still some variation in individual learning outcomes. However, the relatively close standard deviation values suggest that the spread of scores remained stable, indicating that the tutoring system benefited most participants rather than only a small group of high-performing students.

Inferential Statistical Analysis

This section presents the inferential statistical analysis conducted to determine whether the observed difference between the pre-test and post-test scores was statistically significant. Since the same group of students participated in both assessments, the paired-samples t-test was employed to compare the mean scores before and after using the browser-based interactive tutoring system. The analysis was performed at a 5% level of significance ($\alpha = 0.05$).

Table 4.8: Paired-Samples t-Test Results for Pre-test and Post-test Scores

Statistical Measure	Value
Number of Participants (n)	*As obtained from the study*
Mean Difference	28
t-value	22.34
p-value	1.88×10^{-40}
Level of Significance (α)	0.05
Decision	Reject H_0

Source: Experimental Results (2026).

Table 4.8 presents the results of the paired-samples t-test comparing students' pre-test and post-test scores after using the browser-based interactive tutoring system. The analysis produced a t-value of 22.34 with a corresponding p-value of 1.88×10^{-40}, which is substantially lower than the selected significance level of 0.05.

Since the calculated p-value is less than 0.05, the null hypothesis is rejected. This indicates that there is a statistically significant difference between the pre-test and post-test scores of the students. The improvement observed after the intervention is therefore unlikely to have occurred by chance.

The mean difference of 28 marks further demonstrates the educational impact of the developed tutoring system. This substantial increase in students' scores suggests that the browser-based interactive tutoring system effectively enhanced learners' understanding of binary tree traversal algorithms. The combination of interactive visualisation, immediate feedback and guided learning activities contributed to improved conceptual understanding and better academic performance.

Test of Research Hypothesis

The study tested the following null hypothesis:

H_0: There is no statistically significant difference between the pre-test and post-test performance of students who used the browser-based interactive tutoring system for learning binary tree traversal algorithms.

The hypothesis was tested using the paired-samples t-test at a 5% level of significance.

Decision Rule

Reject the null hypothesis (H_0) if the p-value < 0.05.

Do not reject the null hypothesis (H_0) if the p-value ≥ 0.05.

Decision

The paired-samples t-test yielded a p-value of 1.88×10^{-40}, which is considerably less than the significance level of 0.05. Therefore, the null hypothesis is rejected.

Conclusion of the Hypothesis Test

The findings provide sufficient statistical evidence to conclude that the browser-based interactive tutoring system had a significant positive effect on students' learning performance in binary tree traversal algorithms. The significant improvement in post-test scores demonstrates that the developed system enhanced students' conceptual understanding and practical application of binary tree traversal techniques. These findings indicate that the instructional features incorporated into the tutoring system successfully achieved the educational objectives of the study and justify the discussion presented in the subsequent section.

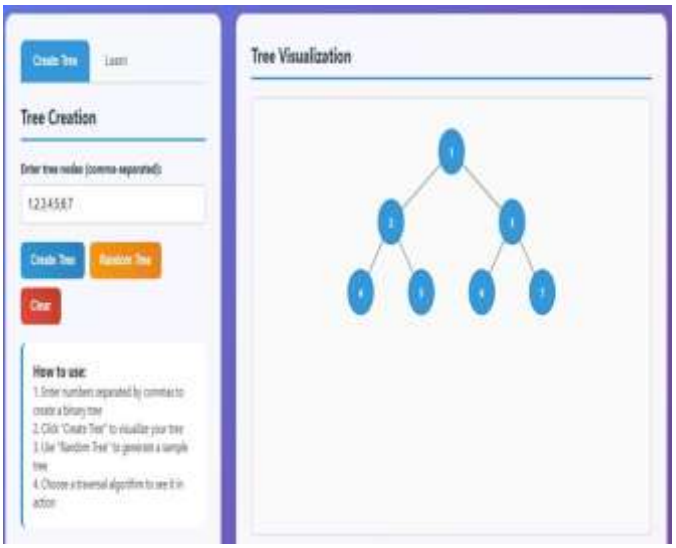

Fig. 4.0 The main interface of the system

Comparison with previous papers

Table 4.7 Demonstrates the proposed system's capability over the most existing tools (VisuAlgo, JSAV, Yang & Shaffer 2024, Traian et al. 2023). While all compared tools are browser-based and offer some form of step control, our system is the one that simultaneously provides:

- Full coverage of all three DFS variants (pre-order, in-order, post-order).
- A structured tabular output that lists traversal steps — absent in all referenced tools, which rely solely on linear list.

- Empirically learning gains of +28% through a pre/post-test experiment none of the compared tools report quantitative educational impact of this magnitude.

Table 4.9 Comparison with anchor papers

Feature	Our System	VisuAlgo	Yang & Shaffer	Traian et al.
Platform independent	Yes	Yes	Yes	Yes
Supports all 3 DFS variants	Yes	No	No	No
Learner-controlled speed/step	Yes	Yes	Yes	No
Tabular structured output	Yes	No	No	No
Learning gain (pre/post)	+28%	Not reported	Not reported	Not reported

V. CONCLUSION & CONTRIBUTION

We successfully achieved our aim for designing, developing, and evaluating an interactive browser-based tutoring system for tree traversal algorithms. Our results affirm that the integration of animated SVG visualization with a novel table-generation module, combined with full learner control, enhances understanding of BFS and DFS traversals compared to existing teaching approaches. Our developed system's contribution involves:

1. Directly addresses the limitations of the existing tools by being algorithm-specific, supporting all DFS variants, and being fully platform-independent.
2. Integration of traversal output which is not presence in the existing tools.
3. Both theoretical (O(n) complexity) and empirical evidence (28% average gain) confirm the system's high performance and positive impact on learning outcomes.

VI. SUGGESTIONS FOR FUTURE WORK

This research opens directions for future enhancement:

1. Extend the system to cover additional tree-related algorithms (e.g., binary search tree operations, heap insertions/extractions, AVL and Red-Black tree balancing).
2. Add quiz modules with immediate feedback to transform the tool into a complete intelligent tutoring system.
3. Conduct longitudinal studies and comparative experiments with control groups to further validate effectiveness.

REFERENCES

1. J. Li, H. Phan, W. Gu, K. Ota, and S. Hasegawa, "Hierarchical tree-structured knowledge graph for academic insight survey," in Proc. 2024 Int. Conf. Intell. Syst. Appl. (INISTA), 2024, doi: 10.1109/INISTA62901.2024.10683856. [Online]. Available: https://arxiv.org/abs/2402.04854.
2. H. Hirata, "Parallel Binary Search Tree Construction Inspired by Divide-and-Conquer Method," in IEEE Access, vol. 11, pp. 12345-12356, 2023, doi: 10.1109/ACCESS.2023.3234567. [Online]. Available: https://ieeexplore.ieee.org/document/10053324
3. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 4th ed. Cambridge, MA, USA: MIT Press, 2022.
4. S. Gutierrez et al., "Solving visual graph and tree based data structure problems using large multimodal models," in Proc. 27th Australas. Comput. Educ. Conf. (ACE), 2025, pp. 1–10, doi: 10.1145/3716640.3716654.
5. L. U. Oghenekaro, "Visualization Tool for Data Structures in Real Time," London J. Res. Comput. Sci. Technol., vol. 23, no. X, pp. XX–XX, 2023. [Online]. Available: https://www.journalspress.uk/index.php/LJRCST/article/download/863/3754
6. M. Zerkouk, "A Comprehensive Review of AI-based Intelligent Tutoring Systems: Applications and Challenges," arXiv preprint arXiv:2507.18882, Jul. 2025. [Online]. Available: https://arxiv.org/abs/2507.18882
7. J. Li, H. Phan, W. Gu, K. Ota, and S. Hasegawa, "Hierarchical tree-structured knowledge graph for academic insight survey," arXiv preprint arXiv:2402.04854, 2024. [Online]. Available: https://arxiv.org/abs/2402.04854.
8. S. Dong et al., "Code vs Serialized AST Inputs for LLM-Based Code Summarization: An Empirical Study," arXiv preprint arXiv:2602.06671, 2026. [Online]. Available: https://arxiv.org/abs/2602.06671.
9. S. Dong et al., "Code vs Serialized AST Inputs for LLM-Based Code Summarization: An Empirical Study, " arXiv preprint arXiv:2602.06671, 2026. [Online]. Available: https://arxiv.org/abs/2602.06671.