Computing and Informatics, Vol. 44, 2025, 336–365, doi: 10.31577/cai_2025_2_336

COMPARATIVE VISUALIZATION OF ALGORITHMS AND DATA STRUCTURES

Filip Vateha, Slavomír Šimoňák

Department of Computers and Informatics
Faculty of Electrical Engineering and Informatics
Technical University of Košice
Letná 9, 042 00 Košice, Slovak Republic
e-mail: vatehafilip@gmail.com, slavomir.simonak@tuke.sk

Abstract. Algorithms and data structures are principal parts of computer science education. For many students, however, it is not easy to master them due to their diversity and inherent complexity. The application of algorithm visualizations is a widely adopted approach, which can help to mitigate this difficulty. Within this work, we aim to improve the efficiency of the learning process in the field of algorithms and data structures. The main directions we use in this work to reach this goal are the introduction of comparative algorithm visualization and the implementation of the visualization tool based on contemporary standards. We analyze and compare several of the available solutions for algorithms and data structure visualization and evaluate them according to the provided functionalities. Further, we define a list of requirements, including the capability to compare selected algorithms visually. The practical outcome of this work is a web application that allows us to visualize and compare different algorithms and data structures in terms of their operation and efficiency. At the end of the paper, the proposed solution is evaluated in several ways.

Keywords: Algorithms, comparison of algorithms, data structures, JSAV, visualization, web application

Mathematics Subject Classification 2010: 68P05, 68P10, 68Q65, 97U50

1 INTRODUCTION AND MOTIVATION

The development of efficient software heavily relies on using appropriate data structures and algorithms [1, 2]. To use them correctly in a program, developers must first understand them properly. Algorithm visualizations appear to be an effective alternative or supplement to textual descriptions of algorithms supported by illustrations from textbooks or lectures [3].

Dynamic graphical visualizations can help users to learn how a particular algorithm works. To be efficient in the learning process, it is very important, how a user interacts with the visualization [4, 5]. So availability of options for enhanced interaction with a visualization (like adjusting playback speed, stepping, or providing input data) can help significantly in understanding the algorithm visualized.

Interesting hypotheses in this regard are proposed in [6] which state, that the higher level or more forms of engagement [7] are used, the more efficient learning process becomes.

In this work, we want to explore the idea of simultaneous or comparative algorithm visualization and also to implement it as one of the crucial features within our algorithm visualization tool. If we were able to display several visualizations of similar algorithms at once, we could better observe their differences in operation, speed, or memory requirements and understand when and if it is appropriate to use a particular algorithm for a given task.

Research shows that comparing multiple methods to solve a single problem promotes learning, procedural knowledge, and flexibility of thinking [8]. This can support students in the idea that there is not always only one 'right' solution to a problem, but that several options may be appropriate depending on the particular circumstances. The authors of the paper [9] based on the experiment found that students in the experimental sample, who were given code-comparison style tasks to solve a particular problem achieved better performance in procedural cognition (reading and writing code) and procedural flexibility (generating, recognizing, and evaluating multiple ways to solve a problem) than the control sample that received non-matched tasks. With these insights, it can be hypothesized that comparative visualization of algorithms can have a positive impact on learning and understanding of the algorithms and data structures presented in this application.

By the comparative algorithm visualization we mean that within a visualization tool we can choose and visualize two (or more) algorithms from the same category. We assume, that within the category are present only such algorithms, which can be compared in some reasonable way. The categorization is a crucial aspect of this approach, as it ensures that the algorithms and data structures being compared share enough similarities to provide meaningful insights. Categories for algorithms could be defined based on algorithmic tasks, such as sorting or graph traversal, and should include algorithms that are comparable in terms of complexity, performance, and underlying principles. For data structures, categories could be based on the structure of the data, such as trees or lists, and should include data structures that are comparable in terms of memory usage, performance, and the types of

operations they support. After the selection of algorithms (and maybe also the input for them) it would be possible to display and control their corresponding visualizations simultaneously.

Many tools can display only visualizations for one algorithm or data structure at a time, but simultaneous display of several algorithm visualizations at once is significantly less common. According to our actual knowledge, the only system that supports comparative algorithm visualization in the sense given above is to a certain limited extent ViSA [10]. Compared to this tool we would like to provide enhanced and generalized comparative visualization functionality.

Some of the tools can only be run on certain platforms or use specific technologies which can introduce problems in learning and accessibility [11]. Other tools may lack enhanced interaction with the visualization or the interaction can be considerably limited, for example, it is not possible to step back.

To avoid the problems described above in the development of our application, we analyzed a selection of existing tools and compared them in terms of usability, features, supported algorithms, and availability. Based on the results of the analysis, our intentions and actual trends in the design of visualization tools [12], we designed an application in a way, that took into account all the advantages and disadvantages of the compared tools, giving it features that were beneficial to users and at the same time as easy to implement as possible.

2 RELATED WORK

As mentioned in the introductory section of the paper, only a negligible part of the number of available tools for the visualization of algorithms supports the parallel display of several algorithms at the same time. We see as possible reasons that the development of such tools tends to be more demanding and also that such a tool requires more attention from the user because it has to follow several algorithms at the same time. Therefore, it is not suitable for complete beginners who are just learning the basic aspects of how algorithms and data structures work, but rather for advanced users who already have basic knowledge and want to expand their knowledge, e.g., about the efficiency of algorithms for a given set of inputs.

A total of eight visualization tools were analyzed, which can be categorized into two groups: desktop applications (Algomaster, ViSA, VizAlgo, AlgoViz) and web applications (VisuAlgo, Isvaus, Data Structure Visualizations, Algorithm Visualizer). The advantage of desktop applications is that they can fully function even without an Internet connection; however, they may not be compatible with all operating systems. In contrast, nowadays web applications are becoming more and more popular due to their easy accessibility on almost any device and the fact that they do not require installation.

Algomaster [13] is a desktop application developed using the .NET Framework. The program works in two modes: *View mode*, where it is possible to simulate the operation of an algorithm or data structure and possibly control it, and

Check mode, where we can test students' knowledge of the selected algorithm or data structure by automatically evaluating the answers. The application includes several algorithms and data structures, which are implemented as plugins, making the program extensible. Its biggest disadvantage is that it is only available for Windows OS.

- Advantages: relatively broad collection of available visualizations, extensible.
- Disadvantages: Windows OS only, no statistical information.

ViSA [10] is the only tool among those analyzed that supports comparative visualization using the so-called simultaneous simulation. With it, the user can choose several algorithms for their visualization, while he can choose the same set of data for all algorithms, or enter input data individually for each algorithm. The controls of the form (textboxes or array preview) are dynamic, which allows the user to change the input data at any time. If all algorithms are given the same input, a speed test can be run where the user can observe and compare the number of actions (comparisons, swaps, interval readjustment, pivot updates) for each algorithm in real-time. These actions are recorded in the queue during the algorithm and are performed individually during the visualizations, while the contents of the queue are displayed in the Listbox. The program also includes an algorithm analysis mode where, after selecting a data set, a graph of the performance of all sorting algorithms is displayed, so that the user can easily conclude what is the most suitable algorithm for the given data set. However, it only supports the visualization of sorting algorithms, it does not support stepping backwards during visualization, and as with the Algomaster tool, this program is only available for Windows OS. It was not possible to try this tool practically as we were not able to find the source codes or the executable file. One of the authors of the program, T. Orehovacki, confirmed in an e-mail communication that the ViSA tool was not further developed after 2012.

- Advantages: available comparative visualization, statistical information.
- Disadvantages: Windows only, limited scope, limited interaction.

VizAlgo [14, 5] is another desktop application created in the Java language. This ensures its cross-platform nature, nevertheless the appropriate Java environment is required. Similar to Algomaster, this tool can be extended with plugins. It supports a wide variety of visualizations, including up to twelve sorting algorithms, hashing visualization, tree structures such as BST, AVL and 2–3 trees, and many more. Additionally, a later update introduced a mode for testing students.

- Advantages: better accessibility (Java), display of complexity graphs for sorting algorithms.
- Disadvantages: limited interaction (does not support stepping backwards).

VisuAlgo [15] is a web-based visualization tool for animating a wide variety of algorithms and data structures. It uses modern web technologies such as HTML5, CSS3 and JavaScript. It includes a wide range of visualizations of algorithms and data structures, which are divided into several categories. A large number of visualizations are available, including sorting algorithms, hash table, linked list, graph traversal, finding the smallest graph skeleton, and many more. For each algorithm, the student can also view brief notes on its properties and functioning, including instructions for setting up the visualization of the given algorithm. In addition to the default browsing mode, there is also a training mode that allows students to test their knowledge on the selected topic. Please note that this tool is not extensible due to licensing restrictions.

- Advantages: accessibility, extensive collection of available visualizations.
- Disadvantages: licensing.

Isvaus [16] is a web application designed for interactive algorithm and data structure visualization. It was created in JavaScript and uses the NuxtJS framework built on VueJS. The tool does not offer a large number of visualizations, but it is easily extensible by modifying the configuration and adding a script controlling the visualization. After selecting a specific visualization, the user can choose the the available mode for that visualization (not all visualizations support all modes):

- Predefined: In this mode, the user can test the functioning of the algorithm on a predefined input (Figure 1).
- Interactive: In this mode, the user can test the functioning of the algorithm on his own input.
- Test: In this mode, the user can test his knowledge on an interactive exercise.

The visualizations are based on the JSAV library. This library [17] contains means to render and animate algorithms (sort, graph traversal) and data structures (array, list, stack, queue, trees, graphs). The main disadvantages of the Isvaus include the absence of automatic visualization playback (only manual stepping between states) and a small number of built-in visualizations (only 3 for data structures and 6 for algorithms).

- Advantages: accessibility.
- Disadvantages: limited collection of available visualizations, no statistical information.

Data Structure Visualizations (DSV)[1] is a web-based tool for visualizing data structures and algorithms. Of all the analyzed web tools, this is the oldest (the JavaScript version was released in 2011), and that is also reflected in its design. The tool provides a wide variety of visualizations, including sorting

[1] DSV: `https://www.cs.usfca.edu/~galles/visualization/Algorithms.html`

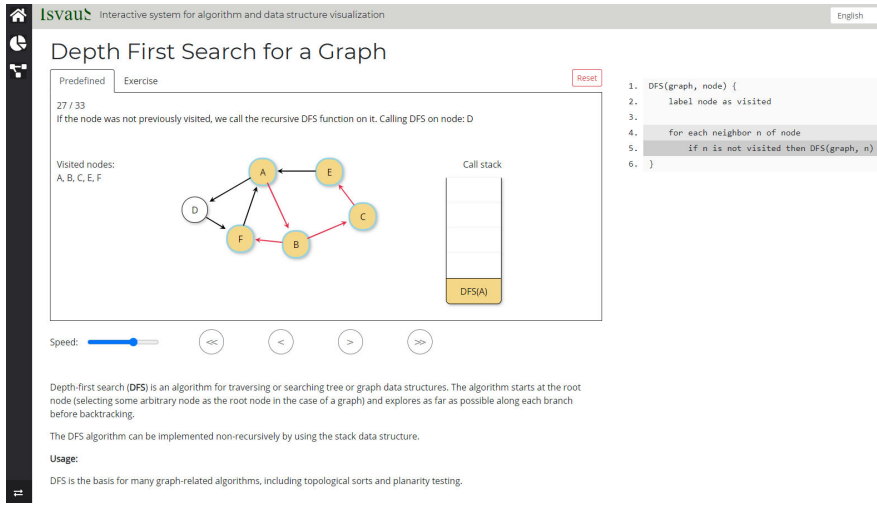


Figure 1. Visualization of the Depth-first search of a graph in the Isvaus tool using the predefined mode

algorithms, stack operations, queue functionality, list handling, tree structures, and graph traversal, among others. When visualizing data structures such as AVL tree allows the user to enter actions (add, remove, search) dynamically. However, when visualizing some algorithms, e.g. sorting, the user does not have the option to enter the input data. Only the options to generate a set of random data and choose the amount of data (small or large) are provided. This can be disadvantageous if the user would like to visualize and test the algorithm on specific data. The tool does not offer any description of the visualized algorithm or the option to display their pseudocode.

- Advantages: accessibility, broad collection of available visualizations.
- Disadvantages: limited descriptions of algorithms, no statistical information.

Algorithm Visualizer[2] is a web application containing a large number of visualizations of algorithms and data structures. It contains many examples of dynamic programming, divide and conquer, greedy and brute force algorithms. The user interface of the application resembles the interface of development environments such as Visual Studio Code, which can be difficult for beginners. A README.md file with a brief description of the visualization is available for each visualization provided. In addition to the graphical visualization, instead of the pseudocode, the real source code of the algorithm (in JavaScript, or possibly in Java or C++) is displayed with the currently executed line highlighted. This source code can be modified, which may be beneficial in certain

[2] Algorithm Visualizer system: `https://algorithm-visualizer.org/`

situations. However, for beginners, this method might be inadequate due to requiring knowledge of the specific programming language and libraries utilized by the tool. There is also a console window that displays log messages about the currently executing algorithm, which can serve to some extent as a substitute for a textual description of the current step of the algorithm.

- Advantages: accessibility, broad collection of available visualizations.
- Disadvantages: real source code may not be suitable for all users.

AlgoViz [18] – last analyzed tool – is a desktop application which is implemented in Python. It utilizes the Tkinter library for GUI rendering and Pygame, which enables rendering of graphics and animations, thereby making the application cross-platform. It deals with the visualization of sorting algorithms (Bubble and Insertion sort), search (Binary and Linear search)and path-finding algorithms (A*, Dijkstra). The application visualizes algorithms step by step, allowing only one algorithm to be visualized at a time. Before the visualization itself, the user enters the type and name of the algorithm he wants to visualize, the playback speed and the array of elements he can generate randomly. In path-finding algorithms, instead of an array of elements, the user selects a start and a destination point and draws barriers in a 2D grid. During the visualization, the user does not have the option to intervene in the visualization or change the input data. On the right side of the application, there is a panel with information about the algorithm with the time complexity, several comparisons as well as the pseudocode of the algorithm. E.g. with the Bubble sort, average, best and worst time complexity of the algorithm and the number of element comparisons are displayed in the information panel. One significant drawback of this application is that it does not allow to control the playback of the visualization or to manually step through the algorithm.

- Advantages: cross-platform, brief information about the algorithm.
- Disadvantages: limited interaction, no stepping through the algorithm.

3 DESIGN REQUIREMENTS

When designing the system, we formulated design requirements (DR) including a comparative visualization of algorithms. The requirements are based mainly on those found in [12], since we consider many of them still valid and suitable for our project. However, the list of requirements has been modified in several ways according to our purposes.

1. *Availability and Compatibility* – The system should be available on all relevant operating systems and platforms.
2. *Purpose and Extensibility* – The system should be general-purpose and should allow easy implementation of additional visualizations.

Requirement/ Tool	**Algo-master**	**ViSA**	**VisuAlgo**	**VizAlgo**	**Isvaus**	**DSV**	**Algorithm Visualizer**	**AlgoViz**
Availability and Compatibility	✗	✗	✓	✓	✓	✓	✓	✓
Expandability	✓	❍[1]	✗	✓	✓	✓	✓	❍[1]
Clarity	✓	✓	✓	✓	✓	✓	✓	✓
Comparability	✗	✓	✗	✗	✗	✗	✗	✗
Interactivity	✓	❍[2]	✓	✓	✓	✓	✓	✗
Pseudocode	✓	✗	✓	✓	✓	✗	❍[5]	✓
Verbal description of the steps	✗	✓	✓	✗	✓	✓	❍[6]	✗
Explanation of the principle of algorithms	✓	❍[3]	✓	✓	✓	✗	✓	❍[8]
Entering inputs	✓	✓	✓	✓	✓	✓[4]	✓[7]	✓
Results and statistics	✗	✓	✗	✗	✗	✗	✗	✗
Student training	✓	✗	✓	✓	✓	✗	✗	✗

Notes:

- 1 – not stated
- 2 – does not support stepping backwards
- 3 – it only displays information about the complexity of the algorithm and the number of comparisons and swaps of elements for the given input
- 4 – for some sorting visualizations there is only options to create a random array of numbers
- 5 – there is only the source code including logging commands
- 6 – it only shows essential information from the logs
- 7 – user can change inputs only from the source code
- 8 – it only displays information about the complexity of the algorithm and the number of comparisons of elements for the given input

Table 1. Comparison of analyzed tools according to requirements

3. *Clarity* – The system should offer a simple, clear, localizable and configurable user interface and intuitive interaction with visualizations. For example, in each step of the algorithm, changes compared to the previous state should be displayed.
4. *Comparability* – The system should allow the comparison of several similar algorithms from the same group.
5. *Interactivity* – The system should enable interactive manipulation of visualizations. The user should be able to move to the beginning of the visualization or move one step forward or backward [19], or adjust the speed of the animation.
6. *Pseudocode* – The system should give the option to display the pseudocode of the executed algorithm or algorithms in a clear form and to highlight the current and previous executed line in it.
7. *Verbal description of steps* – The system should verbally describe the currently executed step of the algorithm.
8. *Explanation of the principle of algorithms* – The system should be able to simply describe its properties and possibly its advantages and disadvantages for each algorithm.
9. *Entering inputs* – The system should allow entering inputs for algorithms. These inputs would then be used for all user-selected algorithms from the same group. In addition to user-defined input, the system should be able to generate random inputs or inputs that exhibit some specific property of the algorithm.
10. *Results and statistics* – The system should be able to display information about the runs and the results found after executing all the running algorithms. This information should be displayed in a table or graphical report. In addition, the system should be able to display statistical data on the performance of running algorithms, if possible.
11. *Student training* – The system should allow students to check their knowledge by the interactive exercises. The system should be able to automatically evaluate the results and display the results to the student.

Requirements R1–R4, R6, R9, R12, R16 and first part of R14 from [12] are incorporated in the above requirements. Requirement R11 was slightly reformulated, resulting in design requirement DR11. Requirements R5, R7, R8 and R10 may be considered in the future versions of the system. Requirements R13, R15 and second part of R14 are not included in the list because they are currently not important for our purposes. Requirement DR4 is a new requirement that was added in order to support the comparative algorithm visualization.

These requirements were analyzed and validated on the tools compared in Section 2. A summary of the comparison of the tools against the requirements is available in Table 1.

In addition to the above requirements (DR1–DR11), the system should also meet some additional features (DR12, DR13):

12. *Usability* – The system should be easy to use and every user should be able to work with it. It should be possible to verify the usability of the system on the basis of a questionnaire, the results of which are presented in Section 6.
13. *Demo mode* – In this mode, the system should be able to show the user how the implemented visualizations work on various inputs in such a way that it is easy to run and use this mode on the OpenLab [20] platform. OpenLab is a space in the vestibule of the department, equipped with cutting-edge technologies, intended for all students, allowing every interested person to try out created solutions and at the same time receive feedback on their own solutions. It contains display devices, cameras, microphones, speakers, smart sensors and many other technologies.

4 PRACTICAL COMPARATIVE ALGORITHM VISUALIZATION – ALGOCOMPARE

We decided to extend the existing Isvaus application because it met almost all our requirements, which allowed us to focus on adding the required functionalities. Isvaus provides a robust foundation with its existing features and stability, which reduces development time and effort. This strategic decision enables us to allocate more resources to innovating and enhancing specific aspects of our solution rather than rebuilding basic functionalities from scratch.

The extended system uses most of the technologies that were used in the original system, including the software framework NuxtJS and JSAV library for visualizing data structures and algorithms. Compared to the original work, the application uses the Vuetify library to implement the user interface, which is based on Material design principles and Progressive Web Application technology to enable the user to work with the application as a native application.

All parts of the application were written in the TypeScript language, which ensures static type checking, and thus increases the security and clarity of the code. Individual visualizations are implemented in separate directories, which enables easy implementation of new visualizations without modifying the main part of the application.

To support the variety of features identified within the requirements (Section 3) and still keep the user interface as simple as possible, we decided to introduce several basic modes in which application can operate:

- *Comparative mode* – The user chooses algorithms or data structures, that it wants to compare and specifies the input values to apply to each of them. Inputs cannot be changed during visualization.
- *Interactive mode* – Allows the user to dynamically define inputs during visualization. This mode was primarily developed for visualizations of data structures that the user is familiar with interacting using operations (actions). This mode is not directly accessible through the main menu of the application, but is available as a component of comparative mode for supported visualizations.

- *Training mode* – In this mode, the user can practice the functioning of some algorithms in the form of interactive exercises. This functionality is implemented in such a way that 2 visualizations are generated based on the generated inputs. One of them contains the correct solution, while the user does not see this visualization. The user sees and interacts with the second visualization. During the exercise, the user's solution is compared to the sample solution. Based on this comparison, the solution is evaluated for correctness. An example of the Training mode during the Depth-First Search exercise can be seen in Figure 2.
- *Demo mode* – This mode functions similarly to the comparative mode; however, the visualizations of the algorithms, corresponding data structures, and their input values are generated randomly. In this mode, the user is not required to interact with the application. This mode was developed for a visual demonstration of the operation of visualizations, especially for the OpenLab platform.

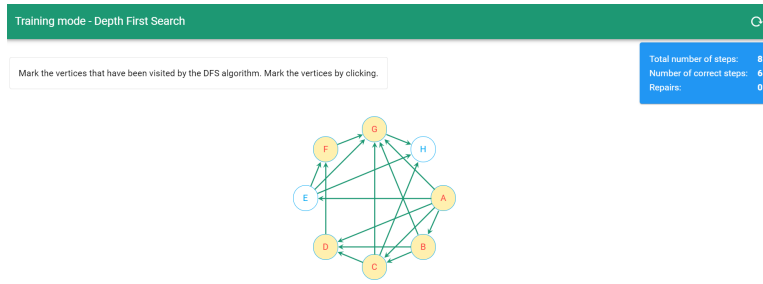


Figure 2. An example of the Training mode when exercising Depth-First Search

4.1 User Scenarios

The main scenario of using the application is to display the visualization of the algorithms or data structures to the user through a comparative mode (Figure 3):

1. The user opens the application.
2. The user selects a category of visualizations (e.g. sorting, tree or graph traversal).
3. The system will show the list of available visualizations in that category.
4. From the list, the user selects the visualizations he/she wants to compare, or chooses one visualization that they want to see.
5. The user enters the input data for the specified visualizations according to the selected category. If a category visualizing the data structure is selected, the user can select the set of operations he/she wants to perform on the data structure, or select the interactive mode. The inputs are applied to all visualizations in the same way.

6. The system displays the visualizations with the inputs specified.
7. If the user has specified a category that visualizes a data structure and has selected interactive mode in step 5, the user can interact with the visualization by entering the operations that are defined for the data structure.

Figure 3. The scenario of configuring the comparative mode of the application

The scenario of using the application in training mode is as follows (Figure 4):

1. The user opens the application.
2. The user selects the training mode.
3. The user selects an exercise category (e.g. sorting, searching trees or graphs), or can choose a specific type of exercise (e.g. Bubble sort, Depth-First Search).
4. The system will generate random inputs for the selected exercise and create 2 visualizations including the one the user will work with.
5. According to the assignment of the exercise, the user simulates the course of the algorithm by selecting visualization elements in the correct order.
6. If the user makes a mistake in the selection, the system warns him and returns him to the last correct step.
7. After completing all the steps, the system will show the user the result of the exercise with data on the total number of steps, the number of correct steps and the number of corrected errors.

4.2 User Interface

The application interface is divided into 3 main parts:

Side menu: Contains a list of basic actions above the application, including a link to the main page, selection of visualizations and application settings. The menu can be minimized and displayed only when the cursor is zoomed in with the mouse, or it can be pinned to the page.

Top menu: In the right part of the top menu there are buttons for the current control of displayed visualizations. In the left part, it is possible to edit the content that will be displayed in the visualization windows.

Main section: This section displays the main content of the page, e.g. selected visualizations. Individual visualizations are displayed in separate windows (Figure 5) that can be sorted, repositioned and resized.

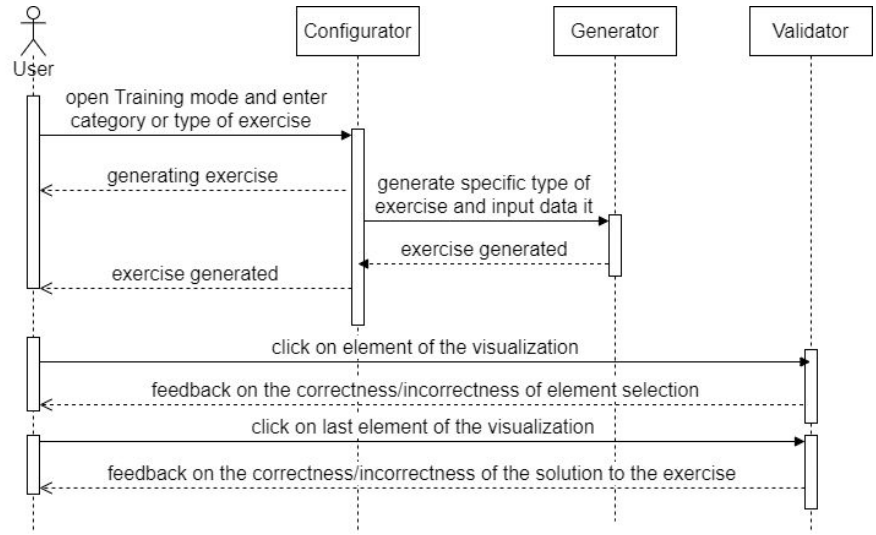


Figure 4. Sequence diagram of the training mode scenario

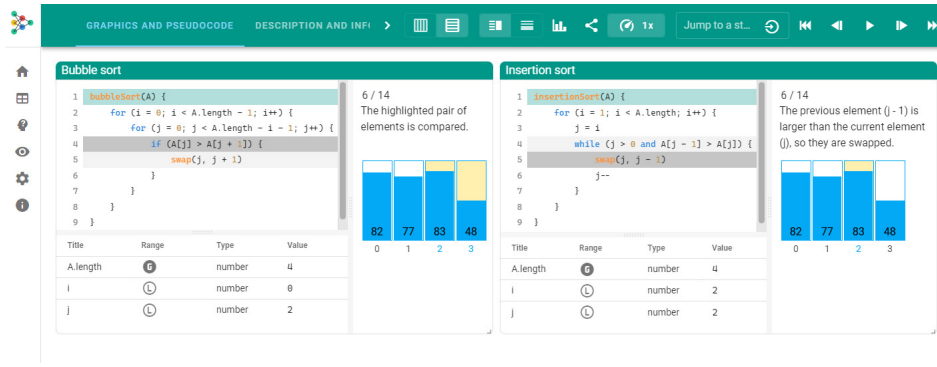


Figure 5. The layout of the content of the application together with the visualizations of Bubble sort and Insertion sort

The challenge when creating the user interface was to solve how to display a large amount of content in individual visualization windows. Specifically, for each visualization, it was necessary to display a graphic representation, the total number of steps, the order of the current step, a textual description of the current step, brief information about the algorithm or data structure, pseudocode with the currently executed step highlighted, and a list of the current variable values. This amount of information could not be displayed at the same time so that the comparative visualization was clear, therefore these elements were divided into display modes and a control element was added in the top menu to enable display and switching

between modes (all visualizations are always switched to the same display mode). The elements have been divided into the following display modes:

Graphical: Displays a graphical representation and a textual description of the current step. This mode is only available if the number of selected visualizations is greater than 3.

Pseudocode: Displays the pseudocode with the currently executed step highlighted and a list of the current variable values. This mode is only available if the number of selected visualizations is greater than 3.

Graphic and pseudocode: Displays all elements from the *Graphical* and *Pseudocode* modes. This mode is only available if the number of selected visualizations is at most 3. This mode was created because, with a maximum of three visualizations, it is still possible to clearly display several elements at the same time.

Description and information: Displays brief information about the algorithm or about the data structure. This mode is available for any number of selected visualizations (Figure 6).

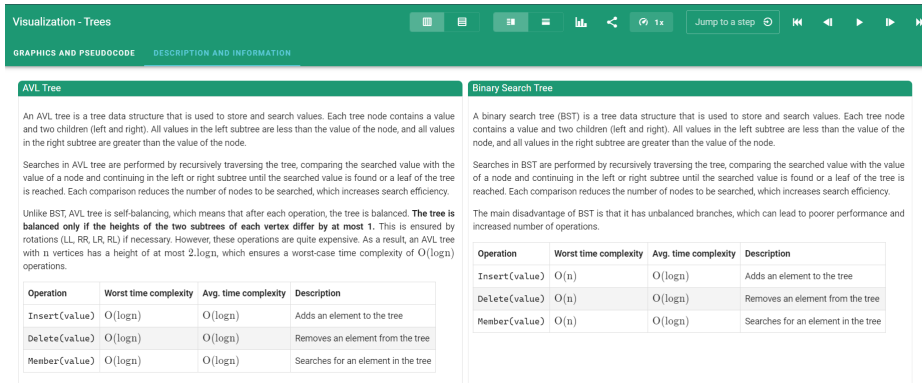


Figure 6. Display mode *Description and information* when visualizing AVL tree and Binary search tree

An important part of the application is also the setting of algorithms and structures to be visualized. After clicking on the button *Comparative mode* in the side menu, a dialog box will appear in which this setting takes place. The visualization setting itself is created in the form of a wizard, which is divided into 3 steps:

1. *Selecting the category of visualizations:* In this step, it is possible to select the category of visualizations to be displayed. By dividing visualizations into categories, it is ensured that when comparing, the user will be able to choose only those algorithms that can be compared in terms of their functionality. Thus,

the user will be able to select, for example, several sorting algorithms within the Sorting category, but will not be able to select algorithms for searching the tree.

2. *Setting visualizations:* In this step, it is possible to select 1 or more visualizations from the selected category. This step has two variants: for the category with simple or dynamic visualizations. More information about these types of visualizations is given in Section 5.
3. *Setting the input:* In this step, the inputs for the selected visualizations are set. The set inputs are applied to all selected visualizations equally.

In addition to the graphic visualization, pseudocode, a textual description of the steps and a brief textual description of the algorithm are available for each visualization in which its functioning, use and advantages or disadvantages are described. After the visualization, but also during it, the user can open the dialog *Output and Statistics*, which contains displays statistical information and outputs of the selected visualizations, such as the number of comparisons, exchanges during sorting, etc. The application fully supports localization using the i18n module[3] in Slovak and English, which can be switched between in the application settings. In the settings, it is also possible to choose a dark mode of the application, as well as a compact mode, which will reduce the height of the top menu.

4.3 Technologies Used

The basis of the application is the software framework NuxtJS[4] (version 2), which forms a superstructure on top of VueJS (also referred to as Vue). NuxtJS allows to create web applications with both client-side (CSR) and server-side (SSR) rendering. Both modes have their advantages and disadvantages, but we decided to use the rendering options on the client side, mainly due to the frequent updating of the application content (i.e. control of visualizations). The entire application works using the software system NodeJS[5], designed for the development of scalable web applications written in JavaScript.

The Vuetify[6] library was used to create the user interface of the application, which enables the simple implementation of the so-called Material design, which is currently very popular and easy to use. A library defines a set of components whose appearance and behavior can be modified to a large extent. It also allows for the easy implementation of the dark mode and defining base colours for the entire application.

The application uses the JSAV (JavaScript Visualization Library)[7] library, which allows easy implementation of visualizations of algorithms and data structures in the

[3] Module i18n: `https://i18n.nuxtjs.org/`
[4] NuxtJS: `https://nuxtjs.org`
[5] NodeJS: `https://nodejs.org/en`
[6] Vuetify: `https://vuetifyjs.com/en`
[7] JSAV: `http://jsav.io/`

form of a presentation. It thus implements the visualization as a list of steps, which ensures easy transition between steps in any direction. This list is implemented as a stack data structure, so to move to a specific position it is necessary to perform all previous steps from the beginning of the visualization. JSAV offers a set of pre-implemented data structures such as array, linked list, tree (including binary), graph (including directed) and others. Being designed as a modular library, it is easy to implement custom structures.

The application implements PWA (Progressive Web Application) technology, which makes it possible to use the web application also as a desktop or mobile application independent of the operating system. The user can easily and quickly install such an application from the page that offers the given application.

5 VISUALIZATIONS

The application allows easy addition of new visualizations only by editing the configuration file and implementation visualizations in the TypeScript language. The author of the visualization can decide which types of modes mentioned above should be supported by the visualization. The configuration of the visualizations is stored in the file `visualisations.yaml` in the directory `config`.

Each visualization is implemented as a separate directory that must contain the following files:

- `<dirname>.ts` – File containing visualization implementation. The file must have the same name as the directory it is in and must contain a default class of any name that inherits from the `Visualisation` class.
- `pcode.html` – A file showing the implementation of the algorithm using pseudocode. It can be a text or HTML file, but it is recommended to use an HTML file that contains predefined tags to highlight the syntax of the pseudocode.
- `info_sk.<html|md|txt>` and `info_en.<html|md|txt>` – Files containing visualization information for both language versions. They can be text, HTML or Markdown files.

The application allows to create 2 types of visualizations according to their use (they differ in implementation):

- *Simple visualizations* – When choosing algorithms in a given category, the user can choose only 1 instance of this type of visualization. An example can be the category *Sorting* with visualizations *Bubble sort* and *Insertion sort*. Both visualizations are implemented as simple visualizations in separate subdirectories, while the user can choose no more than 1 instance of each visualization.
- *Dynamic visualizations* – When choosing algorithms in a given category, the user can choose several examples of this type of visualization with different parameters. Since the user can select several visualizations of the same types, it is necessary to ensure that the title above each visualization for the user clearly

identifies the given visualization. Unlike simple visualizations, where their title was determined only according to the type of visualization, with dynamic ones, it is necessary to add parameters specified by the user to the identification. An example is the category *Hash Table* with the visualizations *Open Addressing* and *Separate Chaining*, where the user can choose several instances from each visualization at the same time with different inputs.

A total of 18 visualizations were created within the application, which are divided into six categories. For the Lists, Hash Table and Trees categories, when entering the inputs for the selected visualizations there is an option to choose the interactive mode.

The visualization interface was designed to respect the basic principles of Gestalt psychology [21]:

- *Law of Balance/Symmetry* – The compared visualizations have a similar/same structure and are symmetrically positioned.
- *Law of Continuation* – Arrows and dashes in graphs and in linked lists are used to indicate the direction of the transition between individual vertices/elements.
- *Law of Closure* – The frames around individual visualizations and their elements are closed, creating closed units.
- *Law of Figure-Ground* – The contrasting colours of the visualization elements are visually separated from the light/dark background (depends on the selected theme).
- *Law of Focal Point* – The currently processed element/elements are highlighted with a different background.
- *Law of Isomorphic Correspondence* – Use of "standard" symbols and shapes for individual visualization elements. Examples might be arrows for the edges of a graph, circles for the vertices of a graph, a tree pointing from top to bottom, etc.
- *Law of Proximity* – The elements of the pseudocode, the variable tables, and the visualization itself are shown close together, indicating their interrelationship.
- *Law of Similarity* – The similar design and colours of the visualization elements indicate that they belong to the same group (e.g. tree).
- *Law of Simplicity / Law of Prägnanz (Good Figure)* – The visualization elements (trees, graphs, callstacks, etc.) are designed as simple as possible, so that the user can easily understand them.
- *Law of Unity/Harmony* – All visualization elements are displayed in the same style (e.g. font, spacing between elements), which creates a uniform and consistent look.

AlgoCompare also applies the principles of Shneiderman's mantra [22]:

- *Overview* – The user can view a set of visualizations that are displayed on one screen.

- *Zoom* – The user can enlarge the window with the visualization that interests him and change its layout.
- *Filter* – The user can filter visualization elements by reducing or by hiding parts of the visualization (with pseudocode, table of variables or graphical representation), or filter the entire visualization by selecting a new set of visualizations.
- *Details on demand* – The user can view additional information about the visualization (e.g. statistics about the algorithm) on demand.
- *Relate* – The application displays relationships between individual visualization elements, such as edges between elements in graphs and trees, or the connection between an array and linked lists in a hash table (in the case of the separate chaining method).
- *History* – The application keeps a history of visualization states, so it is possible to return to them at any time.
- *Extract* – Visualizations, including their inputs and the currently displayed state, can be exported by copying the URL address that contains all the necessary information to display the given set of visualizations.

The application primarily works with one-dimensional data structures, for the list visualization, but also with the trees and networks. A two-dimensional data structure for hash tables is also used in the separate chaining method.

5.1 Lists

Within this category, 3 visualizations have been implemented, which are focused on working with the lists: Linked list, Doubly linked list and array with adjustable size (Array list). The following operations can be performed on each of these visualizations: *AddFirst(value)*, *AddLast(value)*, *Insert(index, value)*, *Remove(value)*, *RemoveAt(index)*, *Get(index)*, *Set(index, value)* and *IndexOf(value)*. The purpose of these visualizations is to demonstrate how each operation affects the lists and how the state of the list changes as a result.

5.2 Hash Table

In this category, 2 types of Hash Table (HT) visualizations have been implemented, which show 2 different collision resolution strategies, namely: *Open Addressing* and *Separate Chaining* (Figure 7). For separate chaining, the user enters the HT size (number of cells). In open addressing, the user specifies the number of cells in the HT, the type of approach (linear testing, quadratic testing, double hashing) and the step size if the user would choose linear testing. If the user chooses double hashing, they must enter a value modulo m for the second hashing function (default is 8). The following formulas are used to calculate the cell address for open addressing:

Linear testing: $h(k, i) = (h(k) + ci) \mod m$;

Quadratic testing: $h(k,i) = (h(k) + ci^2) \mod m; i \neq 0; c \neq 0;$

Double hashing: $h(k,i) = (h(k) + cih'(k)) \mod m;$

where $h(k)$ is the hash function, m is the size of the HT, i is the number of collisions, c is the step size and $h'(k) = 1 + (k \mod m)$ is the second hash function.

Above each of these visualizations, the operations *Insert(value)*, *Remove(value)* and *Member(value)*. This category has been implemented as a dynamic visualization, so the user can select multiple instances of the same type with different parameters. The purpose of these visualizations is to demonstrate to the user how the specified parameters can affect the resolution of collisions and also how the individual operations on the HT work with different strategies.

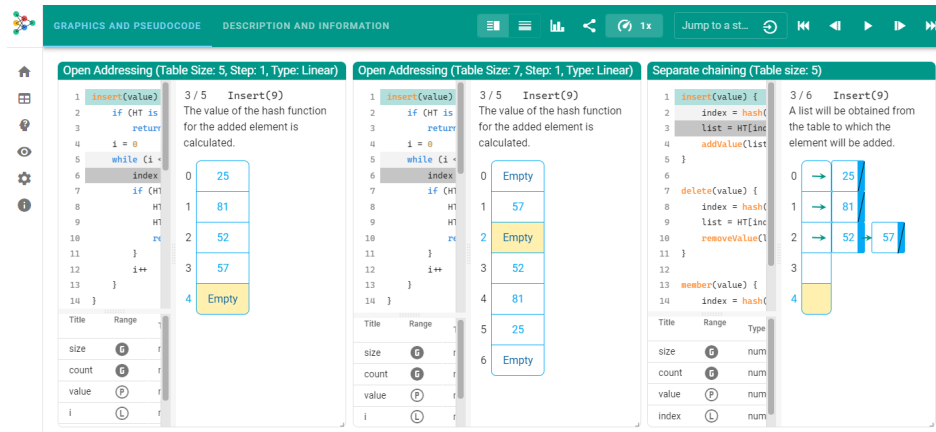


Figure 7. Visualizations of HT with open addressing with table size 5 and 7 and separate chaining with table size 5

5.3 Trees

The user has 2 types of visualizations available in this category representing 2 different types of trees, namely Binary search tree and AVL tree. The operations *Insert(value)*, *Remove(value)* and *Member(value)* can be performed on each of these visualizations. In the *Output and Statistics* dialog, the user can see information about the tree's height, the number of vertices, the rotations performed when balancing the AVL tree, and the number of elements added and removed. The purpose of these visualizations is to demonstrate how individual operations on the trees function, when and how AVL tree balancing is carried out, and how the state of the tree changes with each operation.

5.4 Sorting

This category contains visualization of five sorting algorithms: Bubble sort, Insertion sort, Selection sort, Merge sort and Quick sort. The dialog *Output and Statistics* provides information on the number of comparisons, moves and execution time in milliseconds. The purpose of these visualizations is to demonstrate some sorting methods and their advantages and disadvantages.

5.5 Graph Traversal

Graphs are data structures with many practical applications, as data presented in graphs are easier to interpret and comprehend [23]. There are 2 types of visualizations available in this category, which represent different ways of traversing the graph (Figure 8): *Depth-first search* (DFS) and *Breadth-first search* (BFS). Similar to the visualization of trees, they use the same graph structure for comparison, with colour differentiation of marked vertices, which can help to better understand their principle and better emphasize the difference between these strategies [24]. A call stack structure is available for the DFS strategy, and a queue structure is available for the BFS strategy. In the dialog *Output and Statistics*, the user can see the order of the vertices in which they were visited. Continuous information about the state of the visited vertices is also displayed directly in the graph visualization, where the visited vertices are marked with orange colour, but also in the table with variables.

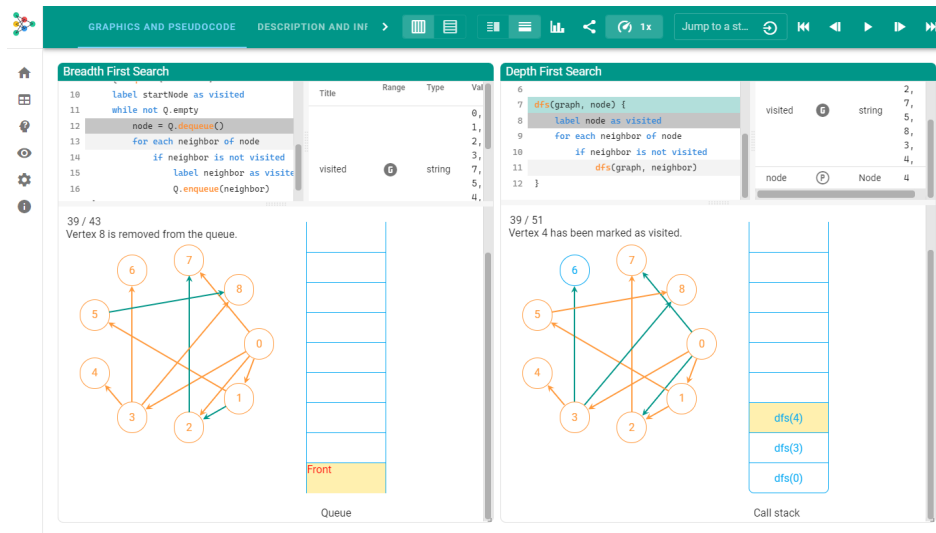


Figure 8. Visualizations of graph traversal using BFS and DFS strategies

5.6 Tree Traversal

The user has four different tree traversal strategies available in this category: *Preorder*, *Inorder*, *Postorder*, and *Levelorder*. The *Levelorder* strategy visualization was created after the realization of the questionnaire, so it is not included within the questionnaire evaluation (Subsection 6.4). Since these are recursive algorithms (except for the *Levelorder* strategy), a call stack view is available for these visualizations (with a queue being available for *Levelorder* instead). As when traversing the graph, the *Output and Statistics* dialog displays the order of the vertices in which they were visited. Continuous information about the state of the vertices is provided in the table of variables, but also graphically directly in the visualization.

6 EVALUATION AND DISCUSSION

Within this section we provide the results of evaluation of the *AlgoCompare* application. The evaluation procedure consists of the following steps:

1. Evaluation against the design requirements presented in Section 3,
2. Comparison with the systems analyzed in Section 2,
3. Testing on the OpenLab platform,
4. Evaluation using a questionnaire.

6.1 Fulfillment of Application Design Requirements

When comparing the *AlgoCompare* application with the design requirements formulated in the section 3, we can conclude that it covers all the specified requirements. It is implemented as a web-based application to meet the availability and compatibility requirements (DR1). The system provides visualizations from several areas, it can be expanded with new visualizations (DR2), it allows comparing multiple visualizations at the same time (DR4), it allows the user to control the stepping of the visualization in both directions, change the speed of the visualization and replay them by performing steps at a certain interval (DR5). The user interface of the application is localized in Slovak and English, the interface of the environment and visualization is configurable through the application settings. The visualization interface allows to display changes from the previous state by highlighting lines of pseudocode with the current and previous states distinguished by different colours (DR3).

AlgoCompare also allows to display additional information about the visualization such as pseudocode (DR6), a textual description of individual steps (DR7), a brief description of the algorithm (DR8), and also to display statistical data about the compared algorithms (DR10). The user has the option to set inputs for specific selected visualizations before they are launched (DR9), while the specified inputs are applied to all selected visualizations. In the interactive mode, it is possible to enter

inputs directly while performing operations. Like the original Isvaus system, *Algo-Compare* also supports student training (DR11) in the form of interactive exercises with a specific visualization.

One of the main features of the system – usability (DR12) was evaluated by a questionnaire (Subsection 6.4). Another established main feature of the system, namely the possibility to demonstrate the operation of visualizations, was implemented in the application as one of the four basic modes (DR13).

6.2 Comparison with Analyzed Systems

When analyzing data form the Table 2, we can see that none of the systems, except the *AlgoCompare*, fulfilled 11 design requirements (DRs), but some of them are quite close. None of the analyzed programs support similar functionality to the demo mode implemented in our application. Additionally, with the exception of the *ViSA* tool, none of the analyzed systems support the comparison of several algorithms at the same time and the display of statistical data about the compared algorithms.

Evaluation/ Tool	**Algo-Compare**	**Algo-master**	**ViSA**	**VisuAlgo**	**VizAlgo**	**Isvaus**	**DSV**	**Algorithm Visualizer**	**AlgoViz**
Fulfilled DRs	11	7	5	8	8	9	6	6	4
Available visualizations	18	25	8	53	25	9	53	71	6

Table 2. Evaluation by comparison with analyzed systems (their latest versions)

Given the number of available visualizations, we can conclude that there are applications with both lower and significantly higher counts of visualizations. Therefore, among other enhancements, we aim to address this issue in future extensions of our system.

The *Algomaster* application offers a larger number of visualization types than *AlgoCompare*. It can be used without an Internet connection; however, it is only compatible with Windows OS. Additionally, the application does not provide a verbal description of the steps involved. It also provides some more complex visualizations, like B-tree or B^+ tree.

The most interesting feature of the *ViSA* tool from our point of view is the ability of simultaneous algorithm visualization, similar to the Comparative mode in our application. However, several important features are missing in the *ViSA* tool, as it does not contain the equivalent of the student testing, it does not display pseudocode, and, like the *Algomaster* tool, it is only compatible with the Windows OS. It also offers a limited control, since the user does not have the option to step

backwards through the visualization if he already exceeded the step he wanted to display. *AlgoCompare* also supports the option to set to a specific visualization step to be displayed.

The *VisuAlgo* web application contains almost all the functionalities of our system, but apart from the unsupported functionalities mentioned above (comparability and statistics), it cannot be extended due to licensing conditions.

The *VizAlgo* application meets most of the requirements and can be used even without the Internet, but it lacks the functionalities mentioned at the beginning of this section.

The *Isvaus* web application meets almost all design requirements, except for the comparability and display of statistics. In addition, our application also supports random input generation and automatic visualization playback (not only manual stepping between states). *AlgoCompare* also contains more visualizations than the *Isvaus* application. Besides significantly enhancing the set of features, we also redesigned the user interface of the application, which can be observed by comparing the figure Figure 1 (*Isvaus*) with Figures 7 or 8 (*AlgoCompare*).

The web applications *Data Structure Visualizations* and *Algorithm Visualizer* are very similar in terms of requirements. As with most compared systems, they support requirements such as extensibility, availability, and clarity, and on the contrary, they do not support student testing, comparability, and summary of results after performing algorithms. The first named one does not allow the user to display information about the principle of the algorithms, and it does not display pseudocode. On the other hand it provides several advanced visualizations, like B-tree, B^+ tree, variety of graph algorithms or heap-like data structures.

The last analyzed application *Algoviz* fully supports only 4 out of 11 requirements, namely availability, clarity, entering inputs and displaying pseudocode. Other requirements are either not supported or their support is limited or not stated in the documentation.

It should be noted that if a system does not support all requirements, it does not necessarily mean that it is bad, but only that it does not meet our goals and requirements.

6.3 Application Testing on the OpenLab Platform

The application was tested on the *OpenLab* platform to verify the functionality of the demo mode on the various display devices available in the vestibule of the department. First, the application was opened in the browser (the browser window was displayed on the display panel) and then the demo mode was started, while the possibility of generating an unlimited number of groups was set (that is, until this mode is manually terminated). During the approximately 30-minute test, approximately 12 groups of visualizations of algorithms and data structures of different types and sizes of inputs were created successively on 2 display devices, namely: a small display panel consisting of 4 screens (2×2) with a total resolution of 4K and a vertical FullHD panel.

	Question	Avg. Response
1.	How would you rate the intuitiveness and clarity of the application's user interface?	4.54
2.	Is the control of visualizations in comparative mode simple and clear (playback, stepping, etc.)?	4.38
3.	In your opinion, is the possibility of comparing algorithms useful (helpful) or rather useless?	4.46
4.	Do you think random input generation is useful for comparative visualization?	4.69
5.	Is the used layout of the windows and their contents (visualization, variable table, pseudocode and algorithm/data structure information) in comparative mode appropriate?	4.31
6.	Did displaying the pseudocode with the currently executed line highlighted help you better understand the algorithm?	4.54
7.	Did the verbal description of the individual visualization steps help you better understand the algorithm?	4.62
8.	Did the verbally explained principle of algorithms/data structures together with additional information such as time and memory complexity, or the described advantages and disadvantages help you to clarify their functioning?	4.23
9.	Did the display of variables and their values in the form of a table help you better understand the algorithm?	4.31
10.	Is the selection of algorithms/data structures for visualization and subsequent entry of inputs clear and easy to use?	4.15
11.	In your opinion, is the display of results or of statistical data on the compared algorithms/data structures in the form of a clear table?	4.15
12.	Are the instructions for the interactive exercises in the training mode sufficient for the exercise to be completed successfully?	4.31
13.	Are interactive visualizations (at lists, search trees and HT) easy to control and clear?	4.07
14.	Did the application help you understand how individual data structures and algorithms work?	4.54
15.	Which visualizations do you think are the most useful?	–
16.	How would you rate this application overall?	4.15
17.	What other data structures and algorithms do you think would be appropriate to add to the application?	–
18.	What new functionalities would you welcome in the application?	–
19.	Do you have any other comments, suggestions to improve the application?	–

Table 3. Questions and results of the 3-part questionnaire evaluation (translated from Slovak)

A minor issue occurred when viewing on a small display panel, where the page and its elements appeared too small, which could make it difficult for the user to read the content from a greater distance. This problem was solved by modifying the CSS styles, when for screens with a resolution of 4K (at least 3 840 pixels wide) and larger, the content of the page is zoomed to 200 %. Otherwise, there were no other major bugs during the test and the application worked without any issues.

6.4 Questionnaire Evaluation

There are several types of questionnaires available for assessing the usability of products, like SUS (System Usability Scale), SUMI (Software Usability Measurement Inventory), QUIS (Questionnaire for User Interface Satisfaction) or TAM (Technology Acceptance Model) [25, 26]. In this work, however, we did not use

any of them, instead we created our own, with questions formulated in a way that would help us in further development and enhancing the application in the future. The questionnaire was partially inspired by the questionnaire used for evaluating the *Isvaus* application [16]. We used a combined questionnaire consisting of three parts, addressing different aspects of the application: usability of the application, visualization/application evaluation, and an optional part with opinions and ideas.

The respondents were 13 students who graduated from the *Data structures and algorithms* subject at the Faculty of Electrical Engineering and Informatics of the Technical University in Košice, aged between 18 and 25. The questionnaire contained 19 questions and was divided into three parts: general usability questions (14 questions); selection of the most useful visualizations and overall evaluation of the application (2 questions); opinions and ideas (3 questions).

In the first part, the respondents had to express to what extent on a scale from 1 to 5 (1 means strongly disagree, 5 means strongly agree) they agree with the statements about the usability of the application. The average rating of the questions from the first part along with their wording is shown in Table 3.

In the second part, the respondents could choose several visualizations which they found most useful and rate the application on a scale from 1 to 5. According to the answers (Figure 9), most respondents prefer sorting visualizations, specifically Quick sort and Merge sort along with the search tree visualizations namely Binary search tree and AVL tree (all four visualizations are preferred by 8 respondents). The smallest number of respondents prefer visualizations in the form of lists (only one for Doubly linked list and two for Array lists and a Linked list). Respondents evaluated the application positively, as the overall evaluation of the application reached a score of 4.15 on a scale from 1 to 5. According to the respondents, the application is clear and easy to use, thus fulfilling one of the main requirements for the system. Respondents evaluated the possibility of comparing different visualizations and algorithms as beneficial and the way this requirement was integrated into the system as intuitive and easy to use.

In the final optional section, respondents were invited to provide feedback on the application by answering three questions. They could suggest new visualizations and functionalities that the application should include, as well as add any additional notes or suggestions for improvement. The most frequently mentioned visualizations to add were the Heap sort visualization and the 2–3 tree visualization. As for new functionalities, the respondents would welcome a login system that would allow them, for example, to save the results achieved in the training mode or save created visualizations and load them again later. With the last question, there was a request to highlight the verbal assignment in the training mode, because it happened that the respondents did not notice it and did not know what they were supposed to do in the exercise. A solution to this requirement was later incorporated into the system.

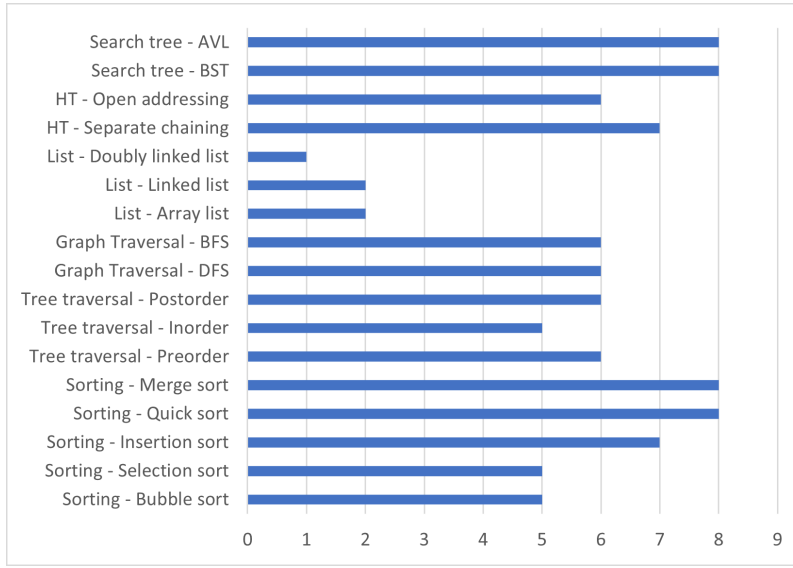


Figure 9. The most useful visualizations according to the survey

6.5 Limitations of the Validity of the Results

Regardless of our efforts there are some factors which can affect the validity of the results. We managed to involve only relatively low number of questionnaire participants, so the results may be unreliable. Also during the evaluation, no test scenarios where created to give more specific information about the usability problems of the application. However, users who tested the application could give feedback about individual parts of the application in the questions in the first part of the questionnaire, on individual visualizations in the 15th question, and with suggestions on how to improve the usability in the 19th question of the questionnaire.

Although the scores evaluating the usability of particular aspects of the application and the overall score were quite high, certainly there are some aspects which can be further improved in the future. E.g. the lowest score (4.07) for question 13 indicates opportunities to enhance the control and clarity of interactive visualizations.

7 CONCLUSION

The practical outcome of this work is the web application *AlgoCompare* intended for the visualization and comparison of algorithms and data structures. As the results of the evaluation suggest, the application can be particularly useful for studying the behavior of several fundamental algorithms and data structures or as a supporting tool for teaching related subjects. It was created by extending the *Isvaus* application

with the possibility of comparing multiple visualizations, completely redesigning the GUI and adding new visualizations of algorithms and data structures.

The application meets all the design requirements we formulated within the paper (Subsection 6.1) and provides unique features, when compared to the analyzed solutions (Subsection 6.2). Even if the comparability, one of the main features of the *AlgoCompare*, is not the completely new idea (as we found the *ViSA* tool supported it before in some limited form), we have taken this important concept to a new level. *AlgoCompare* was also successfully tested within the OpenLab platform (Subsection 6.3), so it is appropriate for deployment on the platform, potentially reaching a wider audience.

The application was evaluated positively by the questionnaire respondents (Subsection 6.4). According to the respondents, the possibility of mutual comparison of visualizations is useful and helpful. They also stated that the application has a clear user interface and is easy to use. Availability of additional information about visualizations was also evaluated positively. Potential limitations of the validity of the results are summarized in Subsection 6.5.

Despite receiving excellent evaluations from the questionnaire respondents, we continue to focus on further improvements and enhancements. In the future, we plan to add more visualizations, including some advanced ones. The system could also be enriched by a new mode (*Testing mode*) with the ability to record the results of individual tests completed by students. Within the mode, in the interface for the teacher, the teacher could have an overview of the results of previous testing, as well as tools for preparing and administering tests. Students could be able to start working on assigned tests or view the results of completed tests in the student interface.

REFERENCES

[1] Shaffer, C. A.: Data Structures & Algorithm Analysis. Edition 3.2 (C++ Version). Department of Computer Science Virginia Tech, Blacksburg, VA, 2013, `https://people.cs.vt.edu/shaffer/Book/C++3elatest.pdf`.

[2] Mehlhorn, K.—Sanders, P.: Algorithms and Data Structures: The Basic Toolbox. Springer, 2008, doi: 10.1007/978-3-540-77978-0.

[3] Shaffer, C. A.—Cooper, M.—Edwards, S. H.: Algorithm Visualization: A Report on the State of the Field. ACM SIGCSE Bulletin, Vol. 39, 2007, No. 1, pp. 150–154, doi: 10.1145/1227504.1227366.

[4] Hundhausen, C. D.—Douglas, S. A.—Stasko, J. T.: A Meta-Study of Algorithm Visualization Effectiveness. Journal of Visual Languages & Computing, Vol. 13, 2002, No. 3, pp. 259–290, doi: 10.1006/jvlc.2002.0237.

[5] Šimoňák, S.: Increasing the Engagement Level in Algorithms and Data Structures Course by Driving Algorithm Visualizations. Informatica, Vol. 44, 2020, No. 3, doi: 10.31449/inf.v44i3.2864.

[6] Naps, T. L.—Rössling, G.—Almstrum, V.—Dann, W.—Fleischer, R.—Hundhausen, C.—Korhonen, A.—Malmi, L.—McNally, M.—Rodger, S.—Velázquez-Iturbide, J. Á.: Exploring the Role of Visualization and Engagement in Computer Science Education. Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education (ITiCSE-WGR '02), ACM, 2002, pp. 131–152, doi: 10.1145/960568.782998.

[7] Urquiza-Fuentes, J.—Velázquez-Iturbide, J. Á.: Pedagogical Effectiveness of Engagement Levels – A Survey of Successful Experiences. Electronic Notes in Theoretical Computer Science, Vol. 224, 2009, pp. 169–178, doi: 10.1016/j.entcs.2008.12.061.

[8] Rittle-Johnson, B.—Star, J. R.: Does Comparing Solution Methods Facilitate Conceptual and Procedural Knowledge? An Experimental Study on Learning to Solve Equations. Journal of Educational Psychology, Vol. 99, 2007, No. 3, pp. 561–574, doi: 10.1037/0022-0663.99.3.561.

[9] Patitsas, E.—Craig, M.—Easterbrook, S.: Comparing and Contrasting Different Algorithms Leads to Increased Student Learning. Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research (ICER '13), 2013, pp. 145–152, doi: 10.1037/0022-0663.99.3.561.

[10] Reif, I.—Orehovacki, T.: ViSA: Visualization of Sorting Algorithms. 2012 Proceedings of the 35th International Convention MIPRO, IEEE, 2012, pp. 1146–1151, https://ieeexplore.ieee.org/document/6240816.

[11] Shaffer, C. A.—Karavirta, V.—Korhonen, A.—Naps, T. L.: OpenDSA: Beginning a Community Active-eBook Project. Proceedings of the 11th Koli Calling International Conference on Computing Education Research (Koli Calling '11), 2011, pp. 112–117, doi: 10.1145/2094131.2094154.

[12] Karavirta, V.—Shaffer, C. A.: Creating Engaging Online Learning Material with the JSAV JavaScript Algorithm Visualization Library. IEEE Transactions on Learning Technologies, Vol. 9, 2016, No. 2, pp. 171–183, doi: 10.1109/tlt.2015.2490673.

[13] Benej, M.—Šimoňák, S.: Algomaster Platform Extension for Improved Usability. Journal of Electrical and Electronics Engineering, Vol. 10, 2017, No. 1, pp. 27–30, https://electroinf.uoradea.ro/images/articles/CERCETARE/Reviste/JEEE/JEEE_V10_N1_MAY_2017/05paper0830SIMONAK_Slavomir.pdf.

[14] Šimoňák, S.: Using Algorithm Visualizations in Computer Science Education.

[15] Halim, S.: VisuAlgo – Visualising Data Structures and Algorithms Through Animation. Olympiads in Informatics, Vol. 9, 2015, pp. 243–245, doi: 10.15388/ioi.2015.20.

[16] Perháč, P.—Šimoňák, S.: Interactive System for Algorithm and Data Structure Visualization. Computer Science Journal of Moldova, Vol. 30, 2022, No. 1, pp. 28–48, doi: 10.56415/csjm.v30.02.

[17] Karavirta, V.—Shaffer, C. A.: JSAV: The JavaScript Algorithm Visualization Library. Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '13), 2013, pp. 159–164, doi: 10.1145/2462476.2462487.

[18] Gupta, A. S.—Vyawahare, M.: AlgoViz: Algorithm Visualization. 2023 5th Bi-

ennial International Conference on Nascent Technologies in Engineering (ICNTE), IEEE, 2023, pp. 1–5, doi: 10.1109/icnte56631.2023.10146719.

[19] Rössling, G.: A First Set of Design Patterns for Algorithm Animation. Electronic Notes in Theoretical Computer Science, Vol. 224, 2009, pp. 67–76, doi: 10.1016/j.entcs.2008.12.050.

[20] Porubän, J.: Challenging the Education in the Open Laboratory. 2018 16th International Conference on Emerging eLearning Technologies and Applications (ICETA), IEEE, 2018, pp. 439–444, doi: 10.1109/ICETA.2018.8572247.

[21] Chang, D.—Dooley, L.—Tuovinen, J. E.: Gestalt Theory in Visual Screen Design – A New Look at an Old Subject. WCCE2001 Australian Topics: Selected Papers from the Seventh World Conference on Computers in Education, Australian Computer Society, 2002, pp. 5–12, https://crpit.scem.westernsydney.edu.au/confpapers/CRPITV8Chang.pdf.

[22] Shneiderman, B.: The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In: Bederson, B. B., Shneiderman, B. (Eds.): The Craft of Information Visualization. Morgan Kaufmann, Interactive Technologies, 2003, pp. 364–371, doi: 10.1016/B978-155860915-0/50046-9.

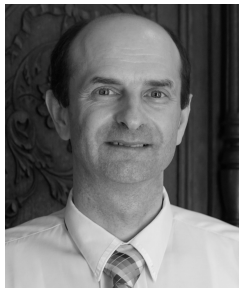
[23] Nesterenko, O.—Netesin, I.—Polischuk, V.—Selin, Y.: Graph-Based Decision Making for Varying Complexity Multicriteria Problems. Computer Science Journal of Moldova, Vol. 30, 2022, No. 3, pp. 391–412, doi: 10.56415/csjm.v30.21.

[24] Mocinecová, K.—Steingartner, W.: Software Support for Visualizing of the Graph Algorithms in a Novel Approach in Educating of Young IT Experts. IPSI Transactions on Internet Research, Vol. 16, 2020, No. 2, pp. 14–23, http://ipsitransactions.org/journals/papers/tir/2020jul/p3.pdf.

[25] Suharsih, R.—Febriani, R.—Triputra, S.: Usability of Jawara Sains Mobile Learning Application Using System Usability Scale (SUS). JOIN (Jurnal Online Informatika), Vol. 6, 2021, No. 1, pp. 41–52, doi: 10.15575/join.v6i1.700.

[26] Kogan, G.—Chassidim, H.—Rabaev, I.: The Efficacy of Animation and Visualization in Teaching Data Structures: A Case Study. Educational Technology Research and Development, Vol. 72, 2024, No. 4, pp. 2349–2372, doi: 10.1007/s11423-024-10382-w.

Filip Vateha received his B.Sc. degree in computer science in 2023 from the Technical University of Košice, Slovakia. He is currently a Master student at the Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Slovakia. His research interests include development of domain-specific languages, web development, algorithms and data structures.



Slavomír Šimoňák received his M.Sc. degree in computer science in 1998 and his Ph.D. degree in 2004, both from the Technical University of Košice, Slovakia. He is currently Associate Professor at the Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Slovakia. His research interests include formal methods integration and application, communication protocols, algorithms, and data structures.