A Dynamic and Static Analysis-Driven Data Structures Visualization System

Hongyu Wang
230141140
MSc Computer Science
ec23804@qmul.ac.uk
Project Supervisor: Dr Ying He

***Abstract*—Learning data structures and algorithms (DSAs) and applying them proficiently can be challenging for novice programmers. This project aims to develop and evaluate a data structure visualization system based on static and dynamic analysis techniques. Its primary function is to parse Java programs and generate content to help programmers understand DSAs intuitively. This paper includes a review of related work in the field, effective approaches for implementing the system, and a discussion of the evaluation results. We also briefly describe relevant future work.**

Keywords: Visualization, software reengineering, static and dynamic analysis, data structures, algorithms.

I Introduction

I-A Background

It is well known that data structures and algorithms (DSAs) form the foundation of programs (Wirth, 1985). Computing revolves around data processing, and problem-solving typically involves abstracting models, designing efficient DSAs, and implementing solutions in code. Thus, DSAs are essential for developing reliable, high-performing software.

Novice programmers often struggle with advanced DSAs despite understanding basic syntax (Luxton-Reilly et al., 2018). The challenge lies in practical application and limited resources. Understanding principles doesn't ensure proficient use. For instance, beginners may successfully compile a sorting algorithm but encounter unexpected results.

Popular learning resources include online video courses and in-person lectures. These usually teach DSAs using pseudocode or specific programming language explanations, which may not be intuitive enough for students. Therefore, some educators try to import graphical views of data structures in their lectures and even develop visualization software for learning data structures to enhance the quality of courses (Hundhausen et al., 2002; Naps et al., 2002).

However, existing solutions have limitations. Firstly, in both online and offline lectures, students primarily act as passive viewers rather than interacting with the system. Secondly, although some DSAs visualization systems can dynamically synchronize graphical displays with source code and allow students to interact with the system, the displayed content is usually preset and unrelated to the user-written code. This means that users have to undertake additional learning of the preset content to use the visualization system. Existing solutions will be discussed in detail in the following chapters.

I-B Aims and Objectives

This project aims to develop a DSA visualization tool for Java programming beginners. It addresses how to design and implement a DSA visualization system linked to user-written code, enhancing novice Java programmers' understanding. Objectives include reviewing existing systems, developing a new system, testing it, and evaluating it through user surveys.

II Related Work Survey

II-A Definition: Data Structures and Algorithms Visualisation

Stasko (1998) pioneered the definition of algorithm visualization (AV), categorizing it alongside program visualization (PV) as software visualization (See Figure 1). They defined AV as visualizing high-level software abstractions, distinguishing between static and dynamic forms. PV was described as a more concrete representation of software behaviour. The authors noted that dynamic AV is particularly useful in education for demonstrating algorithm processes to students.

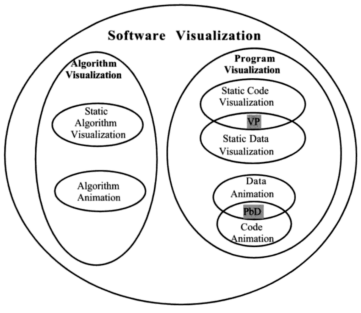


Fig. 1: shows that PV and AV are classified into Software Visualization and AV is further classified into static and dynamic (Stasko, 1998).

Subsequent research has further emphasized the educational purposes and functional aspects of DSAs visualization.

Naps et al. (2002) emphasized visualization's educational goal, proposing an 'engagement taxonomy' with levels from viewing to presenting, arguing higher engagement leads to better learning outcomes. Hundhausen et al. (2002) highlighted interactivity's importance, showing through meta-analysis that active participation in visualization is more effective than passive viewing for learning DSAs.

Regarding visualization content, Goodrich et al. (2014) emphasized using graphical elements to visualize abstract data structures, recommending appropriate visual metaphors like tree diagrams for hierarchies and node-edge representations for graphs. This approach aids students in building intuitive mental models. Shaffer et al. (2010) stressed the importance of dynamic visualization, introducing 'concept maps' that link algorithmic concepts with visual representations. This dynamic approach reveals both final results and intermediate steps, enhancing understanding of algorithm functionality.

In summary, DSA visualization uses graphics and dynamic demonstrations to illustrate abstract data structures and algorithms, primarily enhancing education of complex computational concepts.

II-B Existing Solution: VisuAlgo.net from NUS

The content of the subsection is based on the work of Halim et al. (2012).

VisualGo is a popular project tagged 221 times on GitHub (VisualGo, 2024). It is a powerful and intuitive DSA visualization website initially used by the National University of Singapore's Olympiad in Informatics team members for training. The interface of VisuAlgo provides step-by-step algorithm animations, where users can control the animation speed, pause, and view each step incrementally. To initiate this process, users need to first define the data structure operations, such as searching or removing a node, in the bottom left corner and click the button to start the algorithm animation. Subsequently, the pseudo-code for the algorithm will appear in the bottom right corner, with statements being highlighted as the algorithm executes, informing the user which statements cause changes in the data structure (See Figure 2).

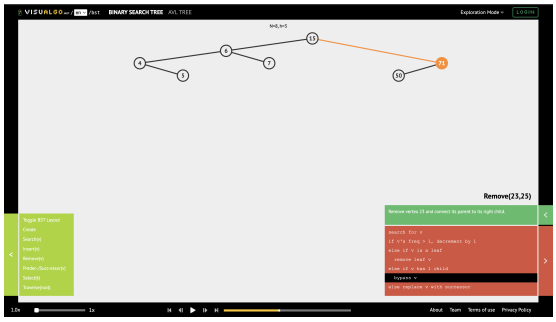


Fig. 2: shows an interface with all the elements for algorithm visualization, with the lower left corner being the area for defining binary tree operations, while the lower right corner shows the pseudocode (Halim et al., 2012).

VisuAlgo is a website developed using JavaScript, CSS, and HTML5 that offers excellent visualization, accessibility, and interactivity across various devices. However, it has significant limitations: users can't upload their own code, most content is preset, updates depend heavily on the maintenance team, and users can't interact by writing algorithm code. Additionally, it uses pseudocode rather than familiar programming languages, potentially hindering beginners' understanding and practical implementation of algorithms.

II-C Existing Solution: DS-PITON Algorithm Visualisation Tool

The content of the subsection is based on the work of Nathasya et al. (2019).

The DS-PITON algorithm visualization tool is developed based on a Python program visualization (PV) tool named PITON. In experimental studies, the system has demonstrated better learning outcomes compared to textbooks (See Figure 3). Unlike VisuAlgo, discussed earlier, DS-PITON allows users to write and visualize Python programs on its interface using seven built-in data structures. These Python programs are authored by users rather than preset.

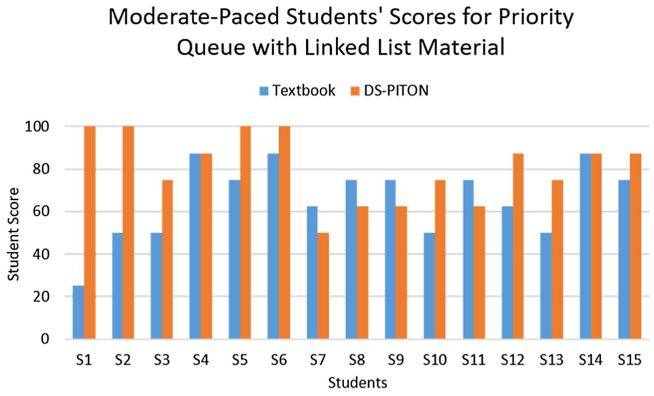


Fig. 3: demonstrates that moderate-paced students scored higher on average using the DS-PITON tool compared to traditional textbooks for learning about priority queues with linked lists (Nathasya et al., 2019).

The advantage of DS-PITON is its user-friendly interface, which integrates an output console, a file path display area, and a compilation error prompt area, making it function as a simple integrated development environment (IDE). This is particularly beneficial for users with some programming experience. DS-PITON can also display the data structure graphics and the source code on the interface. When the play button on the top left control panel is clicked, the algorithm animation starts, with the source code being highlighted alongside the graphical changes. Users can also use the fast-forward and rewind buttons to view different algorithm steps(See Figure 4 for reference).

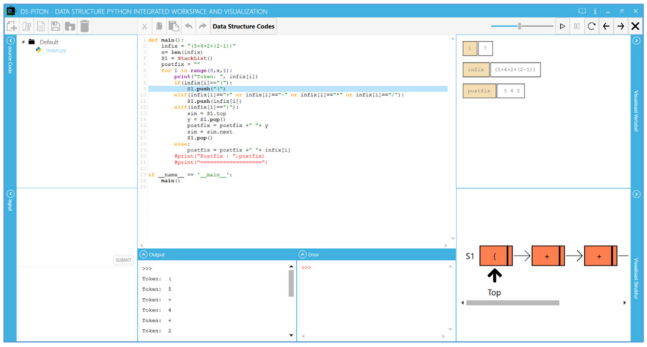


Fig. 4: shows a standard UI of DS-PITON (Nathasya et al., 2019).

However, DS-PITON's drawback is that it can only visualize the seven built-in basic data structures. Users cannot see visualizations of Python programs they write except for seven preset structures.

II-D Existing Solution: Algorithm-visualizer

The content of the subsection is based on the open-source project algorithm-visualizer in GitHub (algo-visualizer, 2024).

Algorithm-visualizer has been tagged 47,000 times on the open-source platform and is a leading DSA visualization project. It simultaneously displays code and data structures, visualizing changes during execution with UI controls. Unlike previous systems, it allows users to modify preset content or upload new algorithm code, directly visualizing user-written code. Supporting multiple languages (C++, Python, JavaScript, Java), it has fostered an active community sharing high-quality content. This enhanced interactivity and customization, coupled with multi-language support, sets it apart from earlier discussed projects (See Figure 5 for reference).

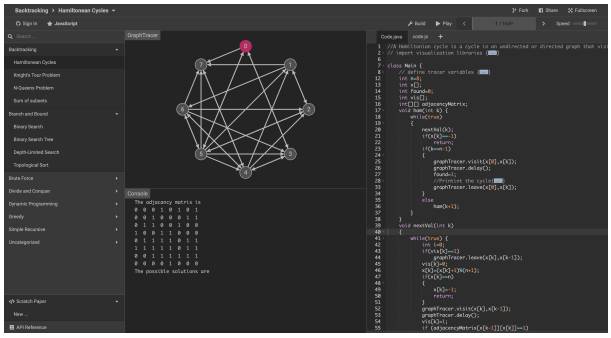


Fig. 5: shows the standard visualization interface of Algorithm-visualizer, simultaneously displaying graphical metaphors with varying degrees of code relevance in the left area (algo-visualizer, 2024).

Goodrich et al. (2014) believes that visual metaphors established by graphic elements in visualization software are crucial, as they help users build mental models to conceptualize data structures. However, these visual metaphors may be misleading, meaning the graphics used for visualization deviate from the actual implementation of data structures in code. For example, using arrow lines and circles to represent LinkedLists is accurate, as lines can be seen as metaphors for pointers. But when the same graphics are used to represent directed graphs, the lines do not represent pointers in the programming language. In fact, most graph structures are implemented using adjacency lists built with HashMap in the code(See Figure 6).

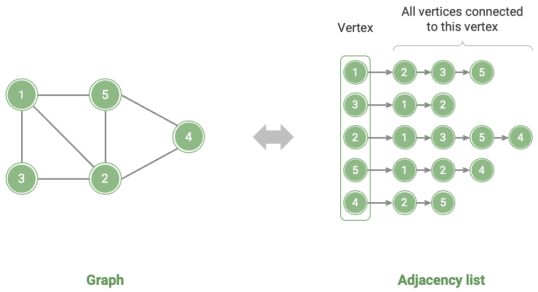


Fig. 6: demonstrates the interpretation of graph structures using HashMap to construct an adjacency list.

To address the issue of visual metaphor distortion, the Algorithm-visualizer simultaneously displays a visualization related to the data structure in the code in the lower-left corner of the interface. As shown in Figure 5, it directly visualizes the adjacency matrix from the code, while showing the graph implemented by this adjacency matrix above it(See Figure 5 for reference).

II-E Summary

This section defined DSAs visualization and examined three popular systems that simultaneously display algorithm source code and related graphics, showing the algorithm's execution process. The systems differ in their connection to user-written code: Algorithm-visualizer, Full user control, and data structure visualization. DS-PITON, Limited visualization for seven preset data structures. VisualGo has no connection to user-written code.

III Methodology

The methodology involves identifying requirements, describing design and implementation, conducting unit tests, and evaluating through user surveys.

III-A Requirements

Scenarios and Users

The system is software-based, aligning with most online DSA learning resources designed for self-supervised learning. Similar to the three representative projects discussed earlier and most visualization systems, this project's main usage scenario will also be self-supervised learning.

Features

Features are keywords about the software that can drive the creation of user-centric stories. Table I (See Table I in Appendix) includes the feature names, descriptions, and reasons for this software.

Working Platform and Developing Language

Most visualization systems are web applications accessible via browsers but lacking direct access to local resources. The system in this project needs to visualize user-written code, requiring closer integration with local development environments. Therefore, it will be an API or library configurable in users' local projects, functioning on desktop devices.

Regarding the programming language for system development, Java is chosen as the development language for direct processing of Java code from users without conversion. It also provides UI frameworks such as Swing and JavaFX, eliminating the need for additional front-end frameworks and simplifying testing.

User Stories

After determining the platform and features, user stories identify specific requirements from a user's perspective. The MoSCoW method prioritizes these stories into four categories: Must have (highest priority), Should have, Could have (lower priority), and Won't have (lowest priority, may be ignored). Table II in the Appendix shows the User Stories and their priorities(Kravchenko et al., 2022). Table II (See Appendix) shows the User Stories and their priorities.

Summary: What Should be Developed?

Following requirements analysis, the system will be developed as an API or library deployable in users' programming environments. It will identify, collect, and parse user-written code and data structures, displaying them through a simple, intuitive interface. The system will feature playback functionality for algorithm execution and data state changes, with user controls for the playback process, enabling easy visualization and understanding of code behaviour.

III-B Design

The design and implementation of this software are based on the following **concepts**:

Data structures present different states under the control of algorithms, and algorithms are implemented through user-written methods. Therefore, the software processes user-written target methods to achieve visualization.

The design section will include a software workflow diagram and the technical solutions choices for implementing the functionalities, followed by software architecture and user interface design selection.

Abstract Software Workflow

The following is an abstract workflow diagram(See Figure 7).

As shown in figure 7, the software system is divided into three abstract parts:

- **Data Collector** - accepts the "target method (source code containing visualization information)" from the user's project, then identifies and obtains the runtime data structure information within it.
- **Graph Generator** - processes the collected data into text or graphical views.
- **Visualizer Interface** - presents the processed information to the user.

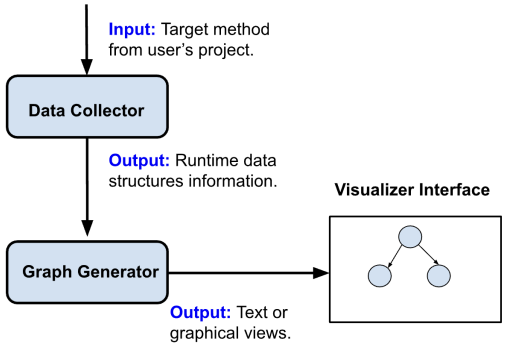


Fig. 7: shows the workflow of the three parts of the software.

Technical solutions: Principles of Static and Dynamic Analysis

The implementation of the 'Data Collector' component will employ principles of static and dynamic analysis, techniques commonly used in software reverse engineering. Static analysis examines code without execution, focusing on structure, syntax, and semantics to identify compilation errors. In contrast, dynamic analysis studies program behaviour during runtime, detecting potential 'bugs' that may pass compilation but produce unexpected results (Ernst, 2003).

To enable comprehensive data collection from user code, the project will utilize Abstract Syntax Tree (AST) for static analysis and code instrumentation for dynamic analysis. AST, derived from parsing source code, is a tree-like data structure that can be traversed to extract useful static information (Wang et al., 2020). Code instrumentation involves inserting additional code before executing the target program to gather runtime information (Tikir and Hollingsworth, 2002).

For the analysis of Java source code, JavaParser will be employed for static analysis. This tool can convert source code to AST and vice versa (Hosseini and Brusilovsky, 2013). Figure 8 illustrates the planned static analysis process in the system.

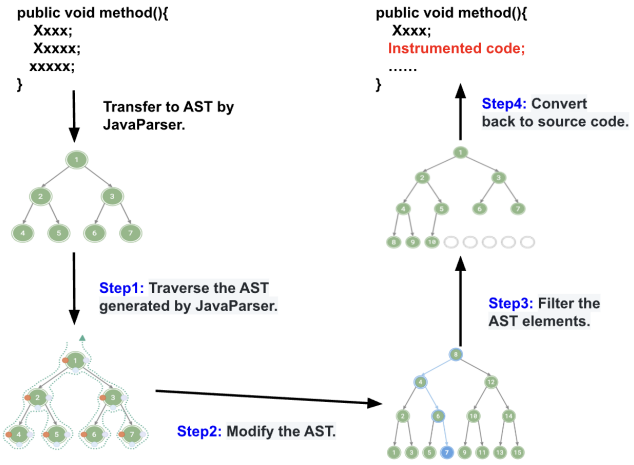


Fig. 8: shows the workflow of the static analysis functionality.

As shown in Figure 8, the working steps description for static analysis functionality in the system are:

- **Step 1** - First, traverse the AST generated by JavaParser to identify the locations of data structure variable declarations and initializations in the user's method's source code.
- **Step 2** - Then, insert the pre-written instrumented code as sub-trees or tree nodes to the AST according to the identification in Step 1.
- **Step 3** - Next, use filtering algorithms to remove any instrumented code that would cause compilation errors.
- **Step 4** - Finally, convert the modified AST back into an executable method source code.

The method with inserted instrumented codes will be used for dynamic analysis. The planned process is as follows:

- **Step 1** - First, compile and execute the instrumented method. The inserted instrumented code will collect runtime data during code execution.
- **Step 2** - Then, after the execution is complete, process the collected data.

Both the static and dynamic analysis functionalities in the system belong to the **Data Collector** part shown in Figure 7.

Software Design Pattern

The software will be implemented following the Model-View-Controller (MVC) design pattern. The three parts in this design pattern can be viewed as collections of programs. The Controller acts as an intermediary connecting the View and the Model, the Controller responds to interface interactions and updates the Model, and finally, the Model notifies the View to update when changes occur (Leff and Rayfield, 2001). The MVC pattern is simple to implement and can effectively handle software without large amounts of interface information display and complex interactions (Syromiatnikov and Weyns, 2014). This meets the needs of this project because the core requirement of the system interface is to display the visualization of data structures, so implementing this requirement does not require displaying massive information or complex interactions. The following Figure 9 demonstrates how to use the MVC design pattern to implement the **Data Collector**, **Graph Generator**, and **Visualizer Interface** parts of the software system.

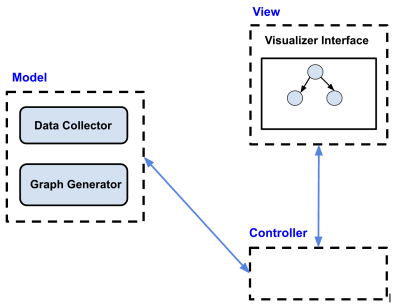


Fig. 9: demonstrates how to use the MVC design pattern to implement the Data Collector, Graph Generator, and Visualizer Interface parts of the software system.

As shown in Figure 9, the static and dynamic analysis functionalities of the Data Collector and the Graph Generator module are all concentrated in the Model. The programs in the Controller are used to respond to interactions from the UI and notify the Model to update, for example, when a user clicks a button and displays new data from the Model.

UI Design

The main UI of the software was designed using Figma, with the aim of achieving the main functions well. Figure 10 shows the design draft of the UI, which includes two display panels, a textbox, controls, and sliders.

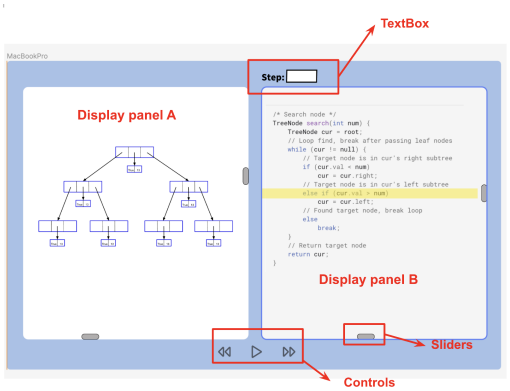


Fig. 10: shows the design of the UI with the visualization content and related method source code.

Display panels A and B are used to visualise data structures and related code snippets, respectively. The controls are responsible for playing, pausing, advancing, and rewinding the dynamic process. The textbox is used to display the current step, and the sliders are used to slide the two panels to show more content.

III-C Implementation

The implementation of this software is complex and challenging to fully discuss within a limited space. Therefore,

three core functional implementations have been selected for description, which effectively summarize the software's implementation.

Identification of Java Data Structure Objects

The static analysis functionality of the system is tasked with identifying data structure object variables used in user code. However, prior to identification, it is necessary to define the scope of data structures the system needs to collect and visualize. The selected Java data structures should be representative in terms of both functionality and structural features, without overlapping characteristics. Table 3 in the Appendix lists the selected data structures from the Java Collections Framework, along with the rationale for their selection.

Subsequently, a class named **MethodDSObjData** is implemented to serve as the variable information model. This class will be instantiated into objects representing data structure variables from user code. The fields and their corresponding types for this model class are illustrated in Figure 11.

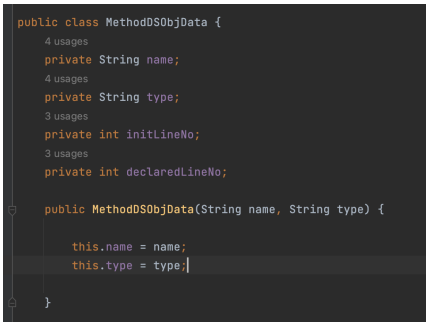


Fig. 11: shows the fields and their types for the variable information model MethodDSObjData.java.

As illustrated in the figure, the **MethodDSObjData** class contains four fields: the variable name, type, line number where the variable is declared, and line number where the variable is initialized, listed from top to bottom.

JavaParser will be utilized to convert user code snippets or target methods into strings and generate Abstract Syntax Trees (AST). The AST will be traversed to identify target variable names and types using JavaParser, instantiating objects of **MethodDSObjData** stored in a list. A second AST traversal will locate line numbers for variable declarations and initializations in the source code. This process results in a list of **MethodDSObjData** objects containing information about target variables from the user's code.

Subsequently, instrumented code will be generated based on the data in the list and inserted between the statements of the target method. Dynamic analysis will then be performed, as detailed in the following subsection.

Instrumented Code and Filter Algorithm

While information about data structure objects has been obtained, this data originates from static semantics rather than actual runtime data. Therefore, instrumentation will be employed to insert multiple individual code statements between the method's existing statements. Prior to this, the instrumented code must be implemented as a pre-written single statement. The instrumented code should be:

- **Concise:** - To minimize code bloat.
- **Purposeful:** - To collect and store runtime data in the system.
- **Efficient:** - To reduce the performance impact on the original method.

The instrumented code is implemented as a statement that calls a static method via its class name as follows in the figure:

Fig. 12: shows three parts of an instrumented code example.

Figure 12 illustrates an instrumented code, which is divided into three parts:

- **Blue Section:** - This is a method call that transfers the collected runtime data to the system as method parameters.
- **Red Section:** - This is the first parameter of the method, representing the line number where the collected runtime data is located. In this example, it indicates that the data comes from line 287 of the target method; the first parameter is generated using the "initLineNo" filed in each **MethodDSObjData** object.
- **Green Section:** - These are paired parameters that appear multiple times. They represent the data itself and its name. These are generated using the data from the **MethodDSObjData** object list in conjunction with string modification methods.

Each instrumented code-collecting task is placed in a new thread and executed in parallel, ensuring that tasks do not wait for each other to complete, thereby reducing performance loss.

The subsequent steps involve modifying the AST generated by JavaParser, inserting these instrumented codes as tree nodes to create a new AST. However, inserting statements solely based on the "initLineNo" field of the **MethodDSObjData** object leads to compilation errors in the generated method. These errors stem from Java statement blocks. In Java, a statement block is a group of statements enclosed within curly braces "{}", which are executed as a single unit Farrell (2013). The error occurs when variables declared inside a block are accessed by instrumented code outside that block, violating scope rules.

To address this issue, a filter algorithm is essential to remove parameter pairs from the instrumented code that would cause these errors. This filtering process ensures that the instrumented code only accesses variables within their valid scope, maintaining the integrity of the Java language's block structure and scoping rules.

As shown in Figure 15 in Appendix, filter algorithm's working mechanism includes the following steps:

- **Conversion:** - Transform the start and end line numbers of statement blocks in the method's source code into nodes of a linked list or tree structure.

- **Traversal and Comparison:** - Iterate through each instrumented code, comparing: a) The parameter pairs within the instrumented code. b) The line number represented by the first parameter.
- **Filtering:** - Remove parameter pairs that do not comply with the established rules based on the comparison results.

After filtering is complete, convert from AST back to instrumented source code. Figure 14 (See Appendix) shows a user-written method without errors after the system has completed instrumentation.

Generation of graphical views and dynamic process

After the detected method is compiled and executed, the runtime data structure information can be obtained. During the execution of the target method, the system concurrently distinguishes the type of data structure and deeply clones it. The cloned data structure will be used to obtain the data in it using the reflection mechanism in Java. Data structures often contain nested storage of other objects; for example, using arrays to store linked lists to simulate hash tables with collisions. Therefore, a reusable recursive algorithm (See Appendix C Algorithm 1) will be employed to store all data within the data structures and perform deep cloning of them. This algorithm operates between the **"Data Collector"** and **"Graph Generator"** (See Design section Figure 7). It is used to store data from the **"Data Collector"** in the system for visualization, making it a core algorithm of the system.

After obtaining and categorizing all data for visualization, the structural features of data structure variables listed in Appendix A Table III need to be identified. These features are identified in Appendix A Table IV. Based on Table IV, three types of graphical views are required:

- **1.** - Graphical view for array-based data structures.
- **2.** - Graphical view for linkedlists.
- **3.** - Graphical view for binary tree-based data structures.

All other data structure views should be derived from combinations of these three basic graphical views. The first version of the graphical views is shown in Figure 16 (See Appendix B). These views feature-rich graphical metaphors and display the data stored in the data structures.

However, this approach raises two issues. Firstly, the views are not based on a unified rule, which also makes it difficult for the program to lay out automatically. Secondly, these views occupy limited UI space. When the size of a data structure is large, users would need to frequently scroll using sliders to view the complete visualization. Consequently, we developed a second version of the graphical views set. In this version, the graphical views allow the system program to process the visualization line by line, using nodes as the smallest units. This approach maintains a degree of metaphorical representation while being as compact as possible to maximize the use of UI space, as shown in Figure 17 (See Appendix B), the second version views are more suitable for displaying larger size data structures.

The Data Collector module gathers line-by-line data from methods, collected through instrumented code. This data is then classified and stored in the system using the core algorithm described in Appendix C Algorithm 1. Subsequently, graphical views are generated for each line of data and stored in a list. The dynamic process is created by iterating through the list on the UI and refreshing the interface to display the list contents. Regarding the differences in data structures during method execution, the second version of graphical views facilitates easier programmatic processing. This enables a direct comparison between the graphical views generated at different steps of the execution process. By comparing these views, the system can identify differences and generate new comparative visualizations.

Final software implementation effect

The final software implementation is illustrated in Figure 18 of Appendix B. This figure demonstrates some capabilities of the software in two screenshots of the dynamic visualisation process.

- **Array Visualization:** - The software's ability to visually represent arrays.
- **Code Highlighting:** - Corresponding code sections are highlighted, establishing a clear link between the visualization and the source code.
- **Multiple Data Structures:** - The figure showcases scenarios where multiple data structures appear within a single method, illustrating the software's capacity to handle complex data relationships.
- **Change Representation:** - Changes in data structures are denoted using color coding - red and green - to visually indicate modifications.

III-D Testing and evaluation

The software's static analysis component, which identifies data and generates instrumented codes for insertion into user-written codes, requires a robust function-level unit test due to its foundational role and potential for causing compilation errors, while the overall system evaluation involves a functional test followed by a user survey to assess both technical reliability and user experience.

Regarding the user experience survey conducted via questionnaire, it is important to note that as the purpose of this survey is to collect user experience data, the software's authors are considered "service providers". The collected data is used solely for software evaluation, and all gathered information is circulated only within QMUL. Therefore, according to the EECS Devolved School Research Ethics Committee guidelines, this questionnaire survey qualifies as "Service evaluation" and does not require ethical review. For more details, please refer to: https://qmulprod.sharepoint.com/sites/EECS-DevolvedSchoolResearchEthicsCommittee [Accessed 15 Aug 2024].

The software will be distributed as a JAR file to volunteer users participating in the survey. Users are required to complete a questionnaire after using the software. The survey does not collect any personal information, ensuring user anonymity. The first seven questions of the survey are designed to determine whether the user falls into the category

of a novice programmer, which is the target user type for this system.

IV Results

The evaluation process began with a unit test of the functions within the static analysis component. A total of 20 test cases were executed across 8 functions. The results of these tests are illustrated in Figure 13.

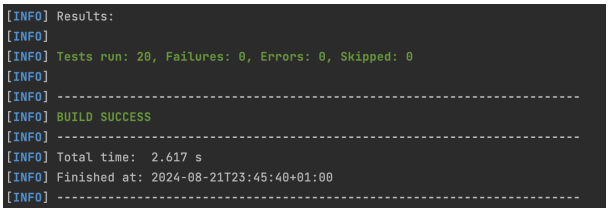


Fig. 13: shows the result of the unit test.

For the test results of the functional test, see Table V in Appendix A. Overall: 13/18 completed.

- **Must have:** - 9/9
- **Should have:** - 3/3
- **Could have:** - 1/4
- **Won't have:** - 0/2

There are 13 volunteer users who participated in the survey. For the results of the questionnaire survey, see Figure 19 and Figure 20 in Appendix B. Overall, more than 85% of the novice programmer-type users screened through the questionnaire were satisfied with the system. Please refer to Appendix D for the questionnaire and user consent form.

V Discussion

The user survey results indicate that the project's implemented features are generally satisfactory, including the visualization of multiple data structures and the presentation of a controllable dynamic process. However, some users identified reasonable functionalities that were not implemented, such as adjustable playback speed and a search feature. A portion of the survey participants found the system less user-friendly than anticipated(See Appendix B Figure 19b). Although they didn't need to modify their original algorithm source code, they still had to write code to launch the GUI and pass the target method's name to the system.

Regarding unit testing, only the static analysis component of the project was robustly tested. Other functions were not covered by unit tests. In fact, the "Graph Generator" part of the project is tightly coupled, especially the interface-related sections, making it challenging to conduct unit tests. It's also important to note that the sample size for the user testing was relatively small. While the results provide valuable insights, they should be considered more as a pilot study rather than a comprehensive survey. This preliminary feedback is useful for identifying potential issues and areas for improvement, but a larger-scale, more formal survey would be necessary to draw definitive conclusions.

In general, the project is satisfactory. It has made attempts and contributions in dynamic and static analysis at the source code level and obtaining runtime data, but there are still many areas that need improvement.

VI Conclusion

The report explores the definition of data structure visualization and analyzes three existing solutions, combining their strengths to address limitations. It presents the design and implementation of a software system based on static and dynamic analysis, which can be deployed in users' projects to directly visualize six common data structures from user-written Java code, bridging the gap between theory and practice. The system demonstrates algorithm execution processes while highlighting relevant code in real time, with user surveys indicating its effectiveness in helping novice programmers learn algorithms and data structures. Overall, the project successfully addresses the question of how to design and implement a DSA visualization system linked to user-written code, enhancing novice Java programmers' understanding. By effectively bridging theoretical concepts with practical application, this project provides a valuable tool for learning and understanding data structures and algorithms, making it an endeavour in the field of computer science education.

VII Future Work

The software still has significant room for improvement. The following are possible future work

1) **Time And Space Complexity Analysis:** Implement functionality to analyze and display the time and space complexity of algorithms, aiding novice programmers in writing more efficient code. In addition, these two analysis functions can also expand the system from the education field to the field of software analysis. In fact, since the instrumented code is densely inserted between all statements, including various loop statements, and the data therein are collected, the implementation of these two functions is realistic.
2) **Compatible with multiple language processing:** This project demonstrates the feasibility of Java code visualization based on static and dynamic analysis. As it obtains data solely through modifications at the source code level, this approach is potentially applicable to most programming languages.
3) **Non-conventional methods:** The software uses traditional analysis methods to solve the problem, but large language models trained on massive data may be used to more efficiently and accurately identify code locations that allow instrumentation.

These enhancements would significantly broaden the software's capabilities, making it a more comprehensive tool in different fields.

References

algo-visualizer (2024), 'algo-visualizer', https://github.com/algorithm-visualizer/algorithm-visualizer. Accessed: 2024-08-02.

Ernst, M. D. (2003), Static and dynamic analysis: Synergy and duality, *in* 'WODA 2003: ICSE Workshop on Dynamic Analysis', pp. 24–27.

Farrell, J. (2013), *Java programming*, Delmar Learning.

Goodrich, M. T., Tamassia, R. and Goldwasser, M. H. (2014), *Data structures and algorithms in Java*, John wiley & sons.

Halim, S., Koh, Z. C., Loh, V. B. H. and Halim, F. (2012), 'Learning algorithms with unified and interactive web-based visualization.', *Olympiads in Informatics* **6**.

Hosseini, R. and Brusilovsky, P. (2013), Javaparser: A fine-grain concept indexing tool for java problems, *in* 'CEUR Workshop Proceedings', Vol. 1009, University of Pittsburgh, pp. 60–63.

Hundhausen, C. D., Douglas, S. A. and Stasko, J. T. (2002), 'A meta-study of algorithm visualization effectiveness', *Journal of Visual Languages & Computing* **13**(3), 259–290.

Kravchenko, T., Bogdanova, T. and Shevgunov, T. (2022), Ranking requirements using moscow methodology in practice, *in* 'Computer Science On-line Conference', Springer, pp. 188–199.

Leff, A. and Rayfield, J. T. (2001), Web-application development using the model/view/controller design pattern, *in* 'Proceedings fifth ieee international enterprise distributed object computing conference', IEEE, pp. 118–127.

Luxton-Reilly, A., Simon, Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A. N., Ott, L., Paterson, J., Scott, M. J., Sheard, J. et al. (2018), A review of introductory programming research 2003–2017, *in* 'Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education', pp. 342–343.

Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S. et al. (2002), Exploring the role of visualization and engagement in computer science education, *in* 'Working group reports from ITiCSE on Innovation and technology in computer science education', pp. 131–152.

Nathasya, R. A., Karnalim, O. and Ayub, M. (2019), 'Integrating program and algorithm visualisation for learning data structure implementation', *Egyptian Informatics Journal* **20**(3), 193–204.

Shaffer, C. A., Cooper, M. L., Alon, A. J. D., Akbar, M., Stewart, M., Ponce, S. and Edwards, S. H. (2010), 'Algorithm visualization: The state of the field', *ACM Transactions on Computing Education (TOCE)* **10**(3), 1–22.

Stasko, J. (1998), *Software visualization: Programming as a multimedia experience*, MIT press.

Syromiatnikov, A. and Weyns, D. (2014), A journey through the land of model-view-design patterns, *in* '2014 IEEE/IFIP Conference on Software Architecture', IEEE, pp. 21–30.

Tikir, M. M. and Hollingsworth, J. K. (2002), 'Efficient instrumentation for code coverage testing', *ACM SIGSOFT Software Engineering Notes* **27**(4), 86–96.

VisualGo (2024), 'Visualgo-list', https://github.com/yulonglong/VisuAlgo-List. Accessed: 2024-07-08.

Wang, W., Li, G., Ma, B., Xia, X. and Jin, Z. (2020), Detecting code clones with graph neural network and flow-augmented abstract syntax tree, *in* '2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)', IEEE, pp. 261–271.

Wirth, N. (1985), *Algorithms & data structures*, Prentice-Hall, Inc.

Appendix A

TABLE I: Features

No.	Feature Names	Descriptions	Reasons
1	Data Identification	Identify data structure information from user-written Java code.	This functionality is a prerequisite for implementing data structure visualization. The information about data structures in the code must first be accurately identified before it can be collected.
2	Data Collection	Collect data structure information from user-written Java code.	This functionality is a prerequisite for data structure visualization; data structure visualization means displaying details and their structural characteristics, which implies the need to collect and then parse this information.
3	Compile and execute	The system needs to be able to compile and execute code containing visualizable content.	Only during the execution phase are the data involved in the code actually calculated, generated, or processed. Therefore, the generation of visualization depends on the successful compilation and execution of the code.
4	Data Parser	Parse the collected data structure information.	Visualization involves showing the details of the data structure and presenting it in a user-friendly way using graphics or text. This will require parsing the collected data structure data.
5	Visualization (static)	This feature means that the data structure should be displayed on the interface in a user-friendly way.	Help users understand the process of algorithm-solving problems.
6	Visualization (dynamic)	The system needs to display the different data structure states during code execution.	Visualization involves showing the details of the data structure and presenting it in a user-friendly way using graphics or text. This will require parsing the collected data structure data.
7	Source code display	Code related to the data structure can also be displayed on the interface.	Provide more references for users.
8	Play the algorithms process.	Users can control different content displays on the interface, pause or start playback, and move forward or backward.	N/A

TABLE II: User Stories

No.	Story	MoSCoW
1	As a user, I want to be able to easily deploy a visualization system in my project directory.	Must have
2	As a user, I want to be able to simply start the system to complete a visualization task.	Must have
3	As a user, I want to use the visualization system without writing additional code.	Must have
4	As a user, I want to see data structure information extracted from my Java code.	Must have
5	As a user, I want the visualization system to handle all common data structures in the Java standard library.	Must have
6	As a user, I want to see the data structure in my Java code presented in an easy-to-understand way.	Must have
7	As a user, I want to see all the data structures in my Java code presented in an easy-to-understand and non-confusing way.	Must have
8	As a user, I want to see the structural features of the data structure, the index, the data stored in it.	Must have
9	As a user, I want to see the visualization content and the source code related to it on the interface.	Should have
10	As a user, I want to see different states of the same data structure.	Must have
11	As a user, I want to see the difference between different states of the same data structure.	Could have
12	As a user, I want to see the changes in the data structures involved as the source code (algorithm) is executed.	Should have
13	As a user, I want to be able to view the data structure state corresponding to any line of code.	Should have
14	As a user, I want to be able to search for the visualization state of a data structure.	Could have
15	As a user, I hope to save the visualization record locally.	Could have
16	As a user, I want the data structures that I customize by instantiating objects to be visualized as well.	Could have
17	As a user, I want to see the time complexity of the algorithm I wrote.	Won't have
18	As a user, I want to see the space complexity of the algorithm I wrote.	Won't have

TABLE III: Data structure screening

No.	Data Structure	Selected	Reasons
1	Array	Yes	A fixed-size data structure that stores elements of the same type in a contiguous memory block. It provides fast random access (O(1)) but has a fixed size that cannot be changed after creation (Farrell, 2013).
2	ArrayList	Yes	A resizable array implementation. It allows dynamic resizing and provides fast random access (O(1)), but inserting and deleting elements (especially in the middle) can be slow (O(n)) (Farrell, 2013).
3	LinkedList	Yes	A doubly linked list implementation. It provides fast insertions and deletions (O(1)) but slower random access (O(n)). It's better suited for scenarios with frequent additions (Farrell, 2013).
4	PriorityQueue	Yes	A heap-based data structure that always gives priority to the smallest or largest element (depending on how it's implemented). It's used for scenarios where elements need to be processed based on priority, not in a strict insertion order (Farrell, 2013).
5	HashSet	Yes	Stores unique elements using hashing. It's unordered and offers O(1) access time (Farrell, 2013).
6	TreeSet	Yes	Stores unique elements in sorted order (natural order or via a comparator). Internally, it's based on a Red-Black Tree (Farrell, 2013).
7	HashMap	Yes	Stores key-value pairs using hashing. It offers fast access (O(1) time complexity) but does not maintain any order (Farrell, 2013).
8	TreeMap	Yes	Stores key-value pairs in sorted order based on the keys (natural order or using a comparator). It uses a Red-Black Tree structure (Farrell, 2013).
9	ArrayDeque	Yes	ArrayDeque is Java's implementation of a double-ended queue (deque) based on a dynamic array rather than a linked list. Although structurally based on an array, ArrayDeque has unique characteristics: it doesn't support null values and allows insertion and deletion operations at both ends of the queue. These features distinguish it from other data structures (Farrell, 2013).
10	Hashtable	No	Hashtable was introduced in JDK 1.0, predating the Java Collections Framework. It's considered a legacy data structure, with its functionality and structural characteristics now largely replaced by HashMap (Farrell, 2013).
11	LinkedHashMap	No	Same as LinkedHashSet.
12	LinkedHashSet	No	LinkedHashSet uses a linked list internally to maintain the order of elements in the HashSet (Goodrich et al., 2014). Its function and structure are repeated with a separate LinkedList and HashSet.
13	Vector	No	Vector was introduced in JDK 1.0, predating the Java Collections Framework. It's considered a legacy data structure, with its functionality and structural characteristics now largely replaced by arrays and ArrayList (Farrell, 2013).

TABLE IV: Data structure screening

No.	Data Structure	Structural Feature
1	Array	N/A
2	ArrayList	Array-based.
3	LinkedList	Linkedlist.
4	PriorityQueue	Array-based.
5	HashSet	Array-based and Linkedlist and binaryTree-based.
6	TreeSet	binaryTree-based.
7	HashMap	Array-based and Linkedlist and binaryTree-based.
8	TreeMap	binaryTree-based.
9	ArrayDeque	Array-based

TABLE V: Funtional Test

No.	Story	MoSCoW	Result
1	As a user, I want to be able to easily deploy a visualization system in my project directory.	Must have	Pass
2	As a user, I want to be able to simply start the system to complete a visualization task.	Must have	Pass
3	As a user, I want to use the visualization system without writing additional code.	Must have	Pass
4	As a user, I want to see data structure information extracted from my Java code.	Must have	Pass
5	As a user, I want the visualization system to handle all common data structures in the Java standard library.	Must have	Pass
6	As a user, I want to see the data structure in my Java code presented in an easy-to-understand way.	Must have	Pass
7	As a user, I want to see all the data structures in my Java code presented in an easy-to-understand and non-confusing way.	Must have	Pass
8	As a user, I want to see the structural features of the data structure, the index, the data stored in it.	Must have	Pass
9	As a user, I want to see the visualization content and the source code related to it on the interface.	Should have	Pass
10	As a user, I want to see different states of the same data structure.	Must have	Pass
11	As a user, I want to see the difference between different states of the same data structure.	Could have	Pass
12	As a user, I want to see the changes in the data structures involved as the source code (algorithm) is executed.	Should have	Pass
13	As a user, I want to be able to view the data structure state corresponding to any line of code.	Should have	Pass
14	As a user, I want to be able to search for the visualization state of a data structure.	Could have	Not Implemented
15	As a user, I hope to save the visualization record locally.	Could have	Not Implemented
16	As a user, I want the data structures that I customize by instantiating objects to be visualized as well.	Could have	Not Implemented
17	As a user, I want to see the time complexity of the algorithm I wrote.	Won't have	Not Implemented
18	As a user, I want to see the space complexity of the algorithm I wrote.	Won't have	Not Implemented

Appendix B

```java
public static void insertionSort(int[] arr) {
    MultiThreadedTracker.trackBasicData( ...params: 277, arr, "arr");
    int n = arr.length;
    MultiThreadedTracker.trackBasicData( ...params: 279, arr, "arr");
    for (int i = 1; i < n; ++i) {
        MultiThreadedTracker.trackBasicData( ...params: 280, arr, "arr");
        int key = arr[i];
        MultiThreadedTracker.trackBasicData( ...params: 281, arr, "arr");
        int j = i - 1;
        MultiThreadedTracker.trackBasicData( ...params: 282, arr, "arr");
        while (j >= 0 && arr[j] > key) {
            MultiThreadedTracker.trackBasicData( ...params: 283, arr, "arr");
            arr[j + 1] = arr[j];
            MultiThreadedTracker.trackBasicData( ...params: 284, arr, "arr");
            j = j - 1;
            MultiThreadedTracker.trackBasicData( ...params: 285, arr, "arr");
        }
        arr[j + 1] = key;
        MultiThreadedTracker.trackBasicData( ...params: 287, arr, "arr");
    }
}
```

Fig. 14: shows a user-written method after the system has completed instrumentation.

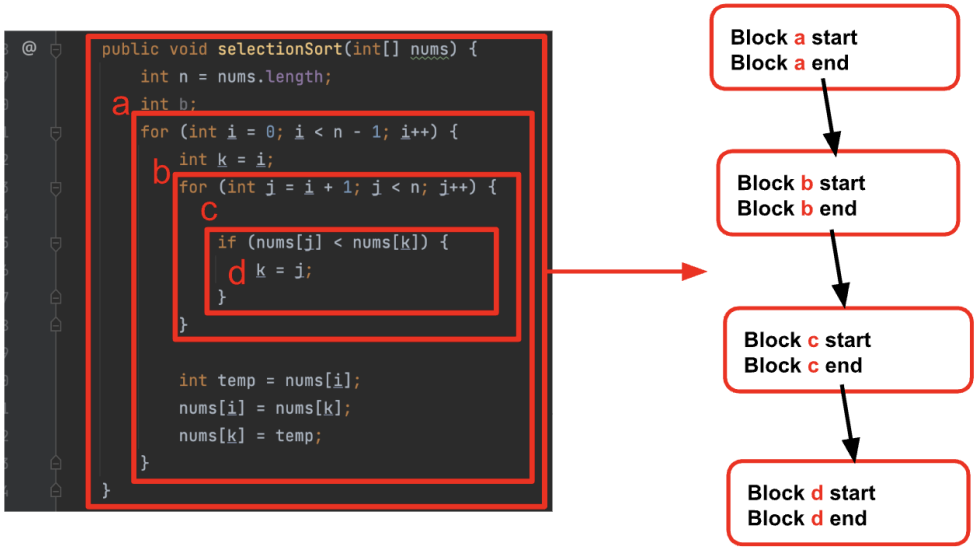
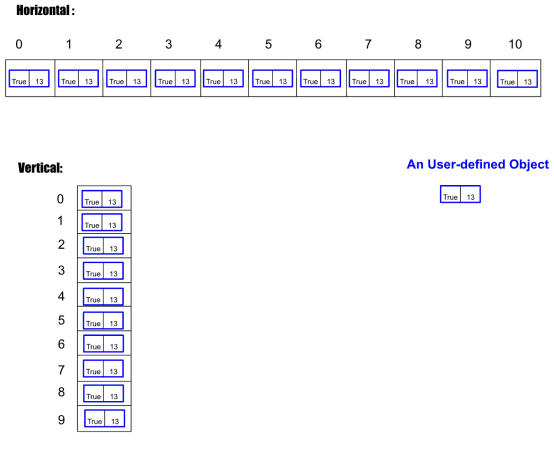
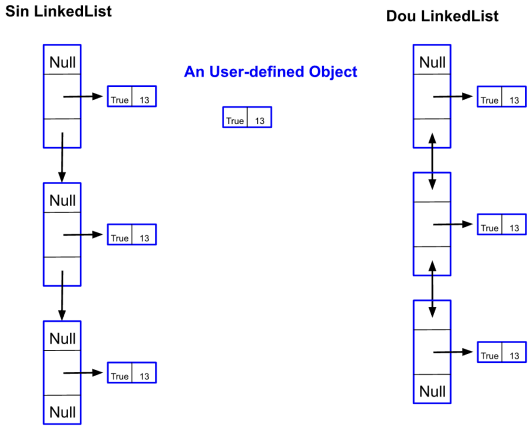


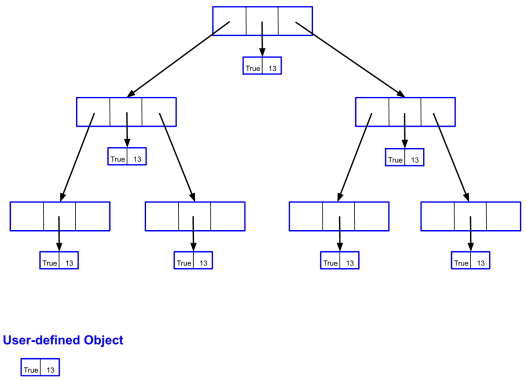
Fig. 15: shows the algorithm's working mechanism that transforms the start and end line numbers of statement blocks in the method's source code into nodes of a linked list or tree structure.



(a) Array-based data structures views with saved data.



(b) Linkedlist views with saved data.



(c) Binary tree-based data structures views with saved data.

Fig. 16: Three graphical views version 1.

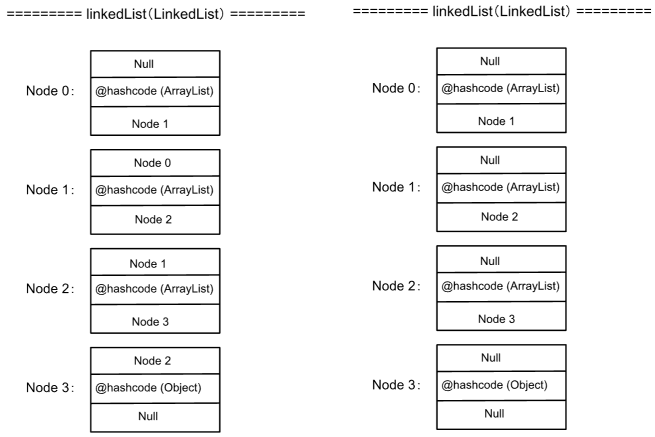
Horizontal :

0	1	2	3	4	5	6	7	8	9	10
12	10	23	34	0	10	11	12	17	45	3

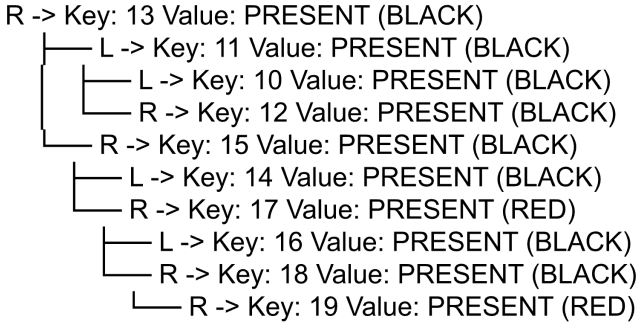
Vertical:

0	
1	
2	
3	
4	
5	
6	
7	
8	

(a) Array-based data structures views with saved data.

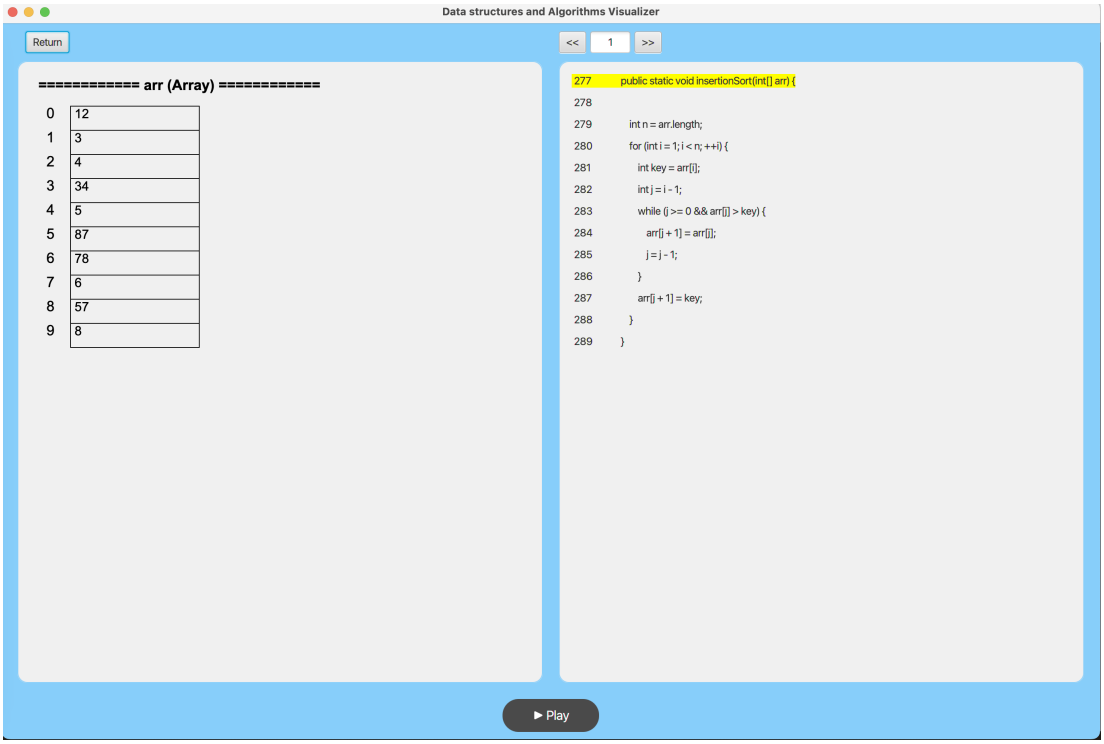


(b) Linkedlist views with saved data.

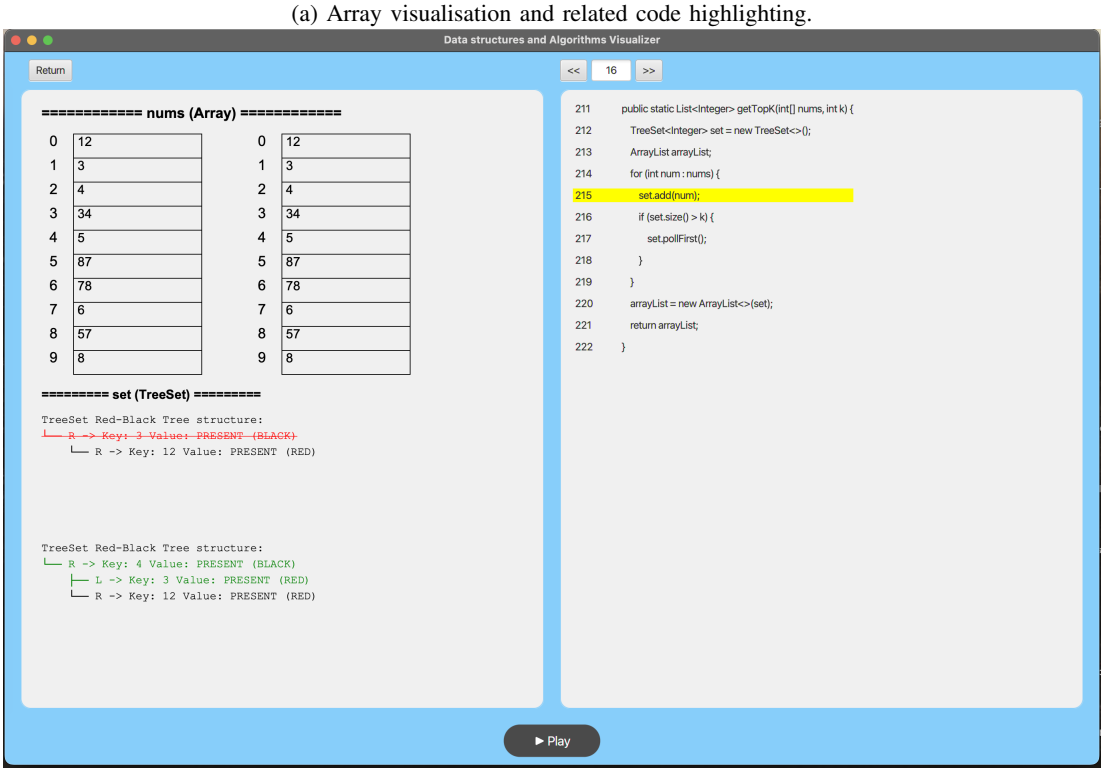


(c) Binary tree-based data structures views with saved data.

Fig. 17: Three graphical views version 2.

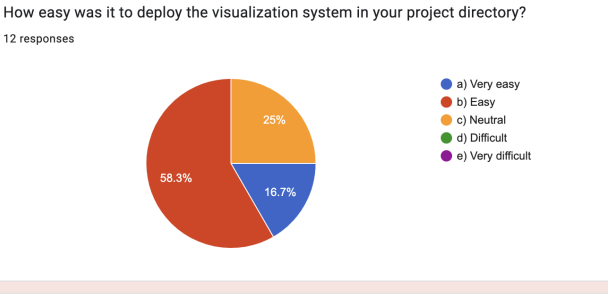


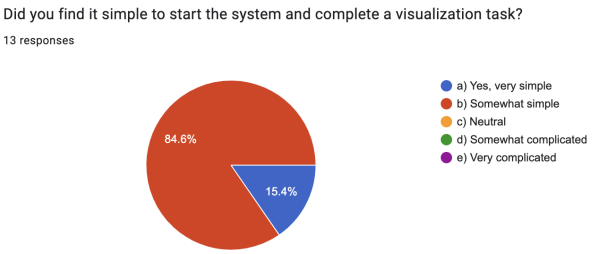
(a) Array visualisation and related code highlighting.



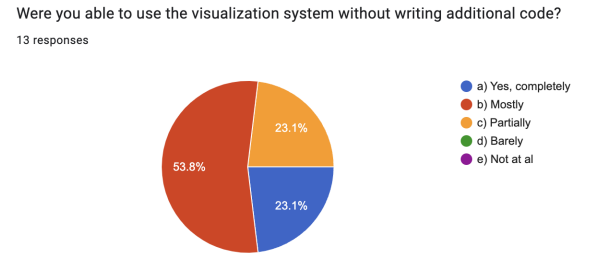
(b) Multiple Data Structures visualisation and change representation.

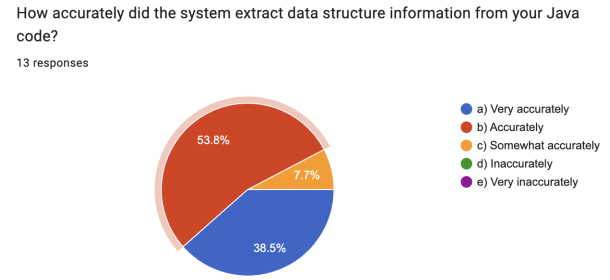
Fig. 18: final effects of the system.



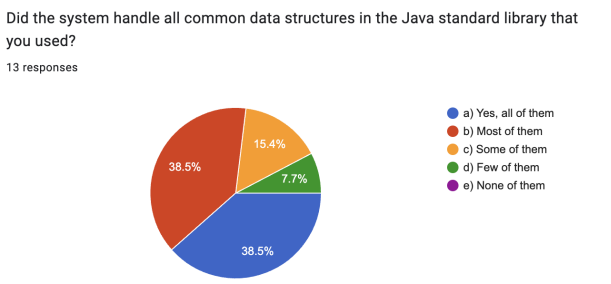


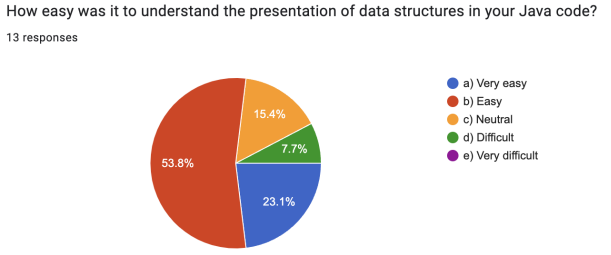
(a) The question 8 and 9 results.



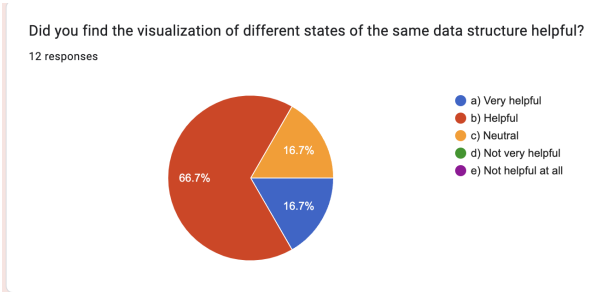


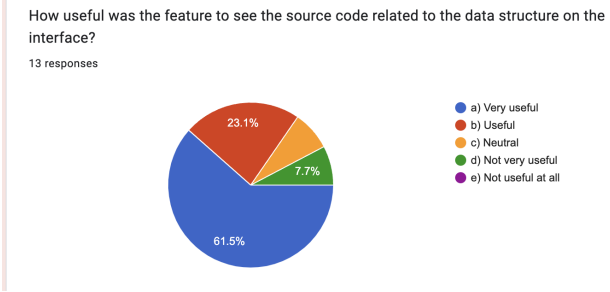
(b) The question 10 and 11 results.





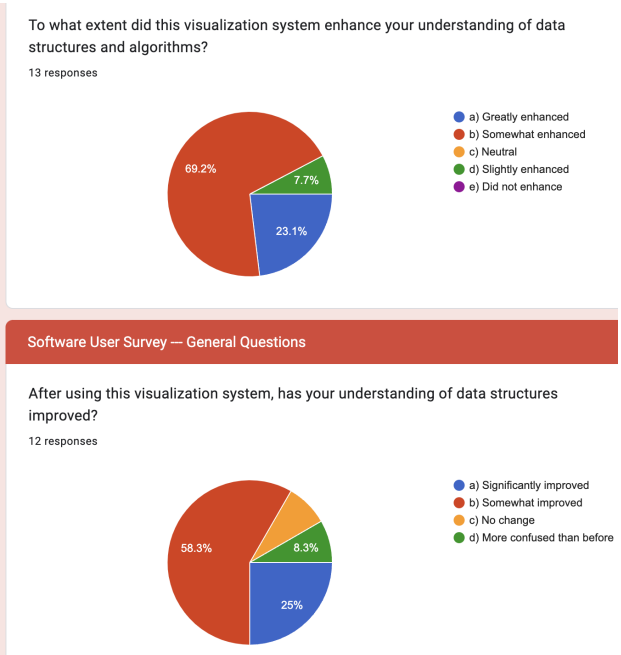
(c) The question 12 and 13 results.



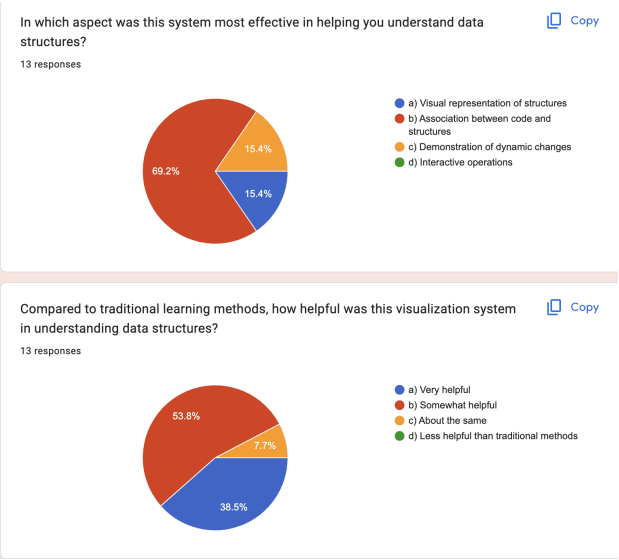


(d) The question 14 and 15 results.

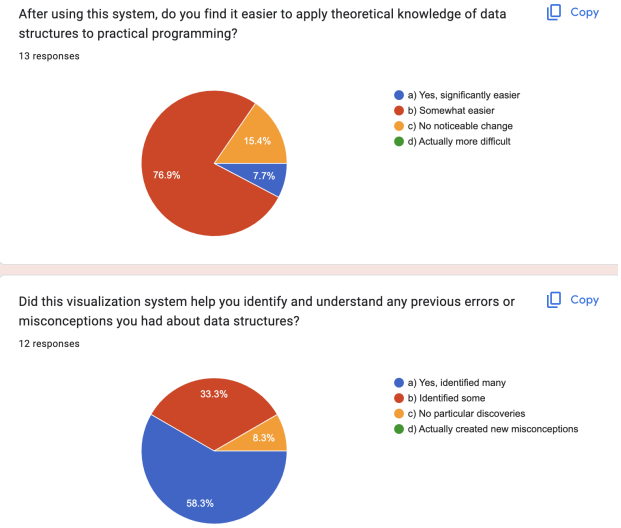
Fig. 19: The user questionnaire results 1.



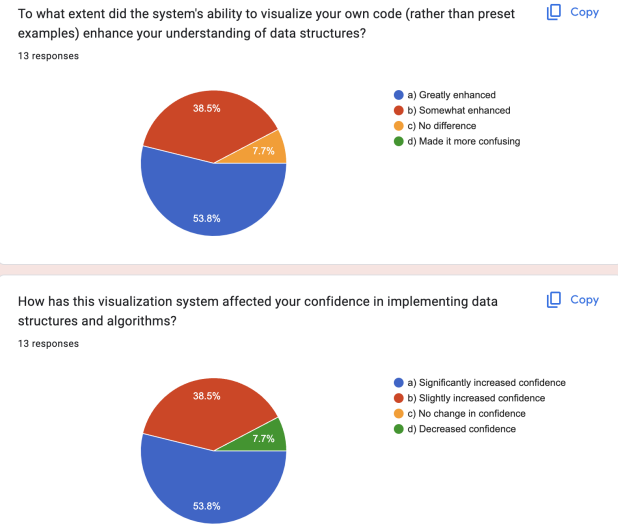
(a) The question 16 and 17 results.



(b) The question 18 and 19 results.



(c) The question 20 and 21 results.



(d) The question 22 and 23 results.

Fig. 20: The user questionnaire results 2.

Appendix C

Algorithm 1: Deep Clone and Store Algorithm

```
function (deepCloneAndStore) (data);
if data is null then
    return null;
else if data is a primitive type (int, float, string, boolean, etc.) then
    return data ;                                          // basic data type so return
else if data is an array then
    clonedArray ← create an empty array;
    for each element in data do
        clonedArray.add(deepCloneAndStore(element));
    end
    return clonedArray;
else if data is a list (ArrayList, LinkedList, etc.) then
    clonedList ← create an empty list;
    for each element in data do
        clonedList.add(deepCloneAndStore(element));
    end
    return clonedList;
else if data is a set (HashSet, TreeSet, etc.) then
    clonedSet ← create an empty set;
    for each element in data do
        clonedSet.add(deepCloneAndStore(element));
    end
    return clonedSet;
else if data is a map (HashMap, TreeMap, etc.) then
    clonedMap ← create an empty map;
    for each key-value pair in data do
        clonedKey ← deepCloneAndStore(key);
        clonedValue ← deepCloneAndStore(value);
        clonedMap.put(clonedKey, clonedValue);
    end
    return clonedMap;
else if data is an object (custom class) then
    clonedObject ← create a new instance of the same class as data;
    for each field in data do
        clonedField ← deepCloneAndStore(field);
        set the field of clonedObject to clonedField;
    end
    return clonedObject;
else if data is another data structure then
    clonedStructure ← create a new instance of the same type as data;
    for each element in data do
        clonedStructure.add(deepCloneAndStore(element));
    end
    return clonedStructure;
else
    return data ;                          // If none of the above, return the data as-is
end
```

Appendix D

User Questionaire Survey

* Indicates required question

Untitled section

Participants Information

Study title

A Dynamic and Static Analysis-Driven Data Structures Visualization System

Version number and date

Version 1.0: 07.08.2024

Researcher's name

Hongyu Wang; Dr Ying He

Queen Mary Ethics of Research Committee reference number:

N/A

Invitation paragraph

You are being invited to participate in a research study. Before you decide whether or not you wish to participate in this study, it is important for you to understand why the research is being done and what it will involve. Please take time to read the following information carefully and discuss it with others if you wish. Ask us questions if there is anything that is not clear or if you would like more information.

What is the purpose of the study and what would taking part involve?

The purpose of this study is to determine if a new interactive online tool can make learning about data structures and algorithms easier and more effective for computer science students. If you choose to participate, you'll try out this tool and answer a few questions about your experience. Participation involves:

- ***Using the educational tool for a set period, likely a few weeks.***
- ***Completing online questionnaires at the beginning and end of the period.***
- ***No in-person meetings are required; all interactions will be online.***
- ***Each session with the tool may last up to an hour, depending on your pace.***
- ***No follow-up plans beyond the initial study period.***
- ***Information collected will be about your use of the tool and your learning progress.***
- ***Data will be collected via online surveys and tool usage logs.***

Your involvement will help improve educational resources for future students.

Why am I being invited?

You are being invited to participate in this research study because you are a novice programmer in computer science, and this study is examining how a new interactive tool can assist in learning data structures and algorithms. You should have a basic understanding of these concepts as part of your coursework.

Do I have to take part?

This participant information sheet has been written to help you decide if you would like to take part. It is up to you whether you wish to take part. If you do decide to take part you will be free to withdraw at any time without needing to provide a reason, and with no penalties or detrimental effects.

What are the possible benefits of taking part?

By participating in this research, you may gain a deeper understanding of data structures and algorithms through the use of the new educational interface. Although there's no guarantee of individual benefits, your feedback can help improve the tool, potentially enhancing the learning experience for future students. Additionally, your participation gives you early access to innovative educational technology and a voice in shaping such tools. The broader benefits may include advancing educational methods and contributing to better academic outcomes for the computer science community.

What are the possible disadvantages and risks of taking part?

The possible disadvantages and risks of participating in this study are minimal. However, you might experience discomfort spending time on the educational interface or answering survey questions. There's a slight risk to confidentiality, as we collect data through online forms, but we'll use secure methods to protect your information. Anonymity is assured since personal identifiers are removed from the data. If you feel any distress during the study, we have

procedures in place to address it, including providing contact information for support services. Your well-being is our priority, and you can withdraw from the study at any time without penalty.

Expenses and payments

Participants will not incur any expenses as the study is conducted entirely online.

What information about me will you be collecting?

The personal data collected in this study will include information pertinent to your experience with the Data structures visualization system. This may consist of demographic details, academic background, and interaction logs with the educational tool. Specific data points could include your age, gender, academic year, course enrollments related to computer science, and feedback on the usability of the interface. All data will be handled confidentially and securely, with access restricted to the research team. This information helps tailor the educational tool to better suit user needs and improve learning outcomes.

How will my data be stored and who will have access to it?

Your data will be stored in a completely anonymous format in a secure Google Drive，accessible only to the study's principal investigator. This approach ensures that anyone viewing the data cannot link your information back to you, protecting your confidentiality in line with the University's strict data protection policy.

When and how will my data be destroyed?

Your data will be kept only for the duration necessary for the research analysis and as required by the university's data retention policy. After this period, typically not exceeding [insert specific time period], your data will be securely deleted in accordance with Queen Mary University's Disposal of Information policy. Data destruction will involve the secure erasure of digital files and any physical documents will be confidentially shredded. This process ensures that your data is not kept longer than necessary and is disposed of securely.

How will my data be used and shared?

Your data will be used for research purposes in a dissertation and will not be used in public. The data will not be stored in an open-access database and will be accessible only to the research team for analysis. The usage and sharing of your data will comply with the

university's Research Data Access and Management Policy, ensuring ethical handling and confidentiality.

Under what legal basis are you collecting this information?

Queen Mary University of London processes personal data for research purposes in accordance with the lawful basis of 'public task'.

Please read Queen Mary's privacy notice for research participants containing important information about your personal data and your rights in this respect. If you have any questions relating to data protection, please contact Queen Mary's Data Protection Officer, Queens' Building, Mile End Road, London, E1 4NS or data-protection@qmul.ac.uk or 020 7882 7596.

What will happen if I want to withdraw from this study?

You have the right to withdraw from this study at any time without providing a reason. If you decide to withdraw, your data collected up to that point can either be retained for analysis or destroyed, based on your preference. However, once the data has been anonymized or included in the overall study analysis, it may not be possible to withdraw it. For instance, you can request the destruction of your data any time prior to its anonymization or the submission of the dissertation. After these stages, it may only be possible to withdraw your participation without removing the already contributed data.

What should I do if I have any concerns about this study?

If you have any concerns about the manner in which the study was conducted, in the first instance, please contact the researcher(s) responsible for the study: *Hongyu Wang*; *Corey John Ford; Jane Reid*. If you have a complaint which you feel you cannot discuss with the researchers then you should contact the Research Ethics Facilitators by e-mail: research-ethics@qmul.ac.uk. When contacting the Research Ethics Facilitators, please provide details of the study title, description of the study and QMERC reference number (where possible), the researcher(s) involved, and details of the complaint you wish to make.

Who can I contact if I have any questions about this study?

- ***Hongyu Wang*** ec23804@qmul.ac.uk;
- ***Dr Ying He*** yinghe@qmul.ac.uk ;

Consent Form

Title of Research Study:

A Dynamic and Static Analysis-Driven Data Structures Visualization System

Principal Investigator:

Hongyu Wang; Dr Ying He

Queen Mary Ethics of Research Committee Ref: N/A

Thank you for your interest in this research.

Should you wish to participate in the study, please consider the following statements. Before signing the consent form, you should initial all or any of the statements that you agree with. Your signature confirms that you are willing to participate in this research, however you are reminded that you are free to withdraw your participation at any time.

Statement

1. I confirm that I have read the Participant Information Sheet dated **08 Aug 2024** version **V1.0** for the above study; or it has been read to me. I have had the opportunity to consider the information, ask questions and have had these answered satisfactorily.

YES/NO

2. I understand that my participation is voluntary and that I am free to stop taking part in the study at any time without giving any reason and without my rights being affected.

YES/NO

3. I understand that my data will be accessed by the research team members:

Hongyu Wang; Dr Ying He

YES/NO

4. I understand that my data will be securely stored in google drive and in accordance with the data protection guidelines of the Queen Mary University of London until 09 Sep 2024 in anonymous form..

YES/NO

5. I understand that I can access the information I have provided and request destruction of that information at any time prior to publication. I understand that following publication I will not be able to request withdrawal of the information I have provided.

YES/NO

6. I agree that personal information collected about me that can identify me will be used in publications and other study outputs.

Agree/Disagree

7. I understand that the researcher will not identify me in any publications and other study outputs using personal information obtained from this study.

YES/NO

8. I understand that the information collected about me will be used to support other research in the future, and it may be shared in anonymised form with other researchers.

YES/NO

9. I agree to take part in the above study.

Agree/Disagree

Participants should read Queen Mary's privacy notice for research participants which contains important information about your personal data and your rights in this respect. If you have any questions relating to data protection, please contact Data Protection Officer, Queens' Building, Mile End Road, London, E1 4NS or data-protection@qmul.ac.uk or 020 7882 7596.

I , Hongyu Wang, confirm that I have carefully explained the nature, demands and any foreseeable risks (where applicable) of the proposed research to the participant and provided a copy of this form.

Principal Investigator (or Supervisor for student projects)

Student Investigator (if applicable)

Dr Ying He yinghe@qmul.ac.uk

Hongyu Wang ec23804@qmul.ac.uk

1. I agree to participate in the research project under the * conditions described above. *

Mark only one oval.

- YES, I confirm that I have read and understand the Participant Information Sheet and voluntarily AGREE to participate in the project.
- NO, I do not wish to participate.

General Questions

2. Do you have any programming experience (Regardless of proficiency or languages.)? *

Mark only one oval.

- YES
- NO

3. Does your university major course involve learning about data structures and algorithms(DSA)? *

Mark only one oval.

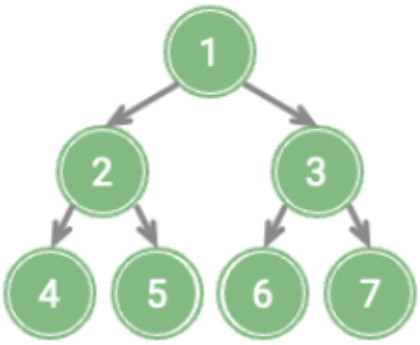
- ◯ Yes, I have known or learned about data structures and algorithms through some modules before.(Regardless of proficiency .)
- ◯ Yes, but I am currently learning data structures and algorithms through some modules.(Regardless of proficiency .)
- ◯ No, but I do teach myself.
- ◯ No, but I have not studied them but plan to study them in the future
- ◯ Know nothing about data structures and algorithms.

4. If you have previously studied or are currently studying DSA, what do you think about your level of proficiency? *

Mark only one oval.

- ◯ Beginner
- ◯ Intermediate
- ◯ Advanced
- ◯ Expert

5. Please match all the following data structure features with the image below to help us determine your level. *



Tick all that apply.

- [ ] contiguous
- [ ] dispersed
- [ ] Linear
- [ ] Non-Linear
- [ ] Other: _____

6. Please match all the following data structure features with the image below to help us determine your level. *

Tick all that apply.

- [ ] contiguous
- [ ] dispersed
- [ ] Linear
- [ ] Non-Linear
- [ ] Other: _____

7. Please match all the following data structure features with the image below to help us determine your level.

Tick all that apply.

- [ ] contiguous
- [ ] dispersed
- [ ] Linear
- [ ] Non-Linear
- [ ] Other: _____

Software User Survey --- Specific Questions

8. How easy was it to deploy the visualization system in your project directory?

Mark only one oval.

- ◯ a) Very easy
- ◯ b) Easy
- ◯ c) Neutral
- ◯ d) Difficult
- ◯ e) Very difficult

9. Did you find it simple to start the system and complete a visualization task?

Mark only one oval.

- a) Yes, very simple
- b) Somewhat simple
- c) Neutral
- d) Somewhat complicated
- e) Very complicated

10. Were you able to use the visualization system without writing additional code?

Mark only one oval.

- a) Yes, completely
- b) Mostly
- c) Partially
- d) Barely
- e) Not at al

11. How accurately did the system extract data structure information from your Java code?

Mark only one oval.

- a) Very accurately
- b) Accurately
- c) Somewhat accurately
- d) Inaccurately
- e) Very inaccurately

12. Did the system handle all common data structures in the Java standard library that you used?

Mark only one oval.

- a) Yes, all of them
- b) Most of them
- c) Some of them
- d) Few of them
- e) None of them

13. How easy was it to understand the presentation of data structures in your Java code?

Mark only one oval.

- a) Very easy
- b) Easy
- c) Neutral
- d) Difficult
- e) Very difficult

14. Did you find the visualization of different states of the same data structure helpful?

Mark only one oval.

- a) Very helpful
- b) Helpful
- c) Neutral
- d) Not very helpful
- e) Not helpful at all

15. How useful was the feature to see the source code related to the data structure on the interface?

Mark only one oval.

- a) Very useful
- b) Useful
- c) Neutral
- d) Not very useful
- e) Not useful at all

16. To what extent did this visualization system enhance your understanding of data structures and algorithms?

Mark only one oval.

- a) Greatly enhanced
- b) Somewhat enhanced
- c) Neutral
- d) Slightly enhanced
- e) Did not enhance

Software User Survey --- General Questions

17. After using this visualization system, has your understanding of data structures improved?

Mark only one oval.

- a) Significantly improved
- b) Somewhat improved
- c) No change
- d) More confused than before

18. In which aspect was this system most effective in helping you understand data structures?

Mark only one oval.

- a) Visual representation of structures
- b) Association between code and structures
- c) Demonstration of dynamic changes
- d) Interactive operations

19. Compared to traditional learning methods, how helpful was this visualization system in understanding data structures?

Mark only one oval.

- a) Very helpful
- b) Somewhat helpful
- c) About the same
- d) Less helpful than traditional methods

20. After using this system, do you find it easier to apply theoretical knowledge of data structures to practical programming?

Mark only one oval.

- a) Yes, significantly easier
- b) Somewhat easier
- c) No noticeable change
- d) Actually more difficult

21. Did this visualization system help you identify and understand any previous errors or misconceptions you had about data structures?

Mark only one oval.

- a) Yes, identified many
- b) Identified some
- c) No particular discoveries
- d) Actually created new misconceptions

22. To what extent did the system's ability to visualize your own code (rather than preset examples) enhance your understanding of data structures?

Mark only one oval.

- a) Greatly enhanced
- b) Somewhat enhanced
- c) No difference
- d) Made it more confusing

23. How has this visualization system affected your confidence in implementing data structures and algorithms?

Mark only one oval.

- a) Significantly increased confidence
- b) Slightly increased confidence
- c) No change in confidence
- d) Decreased confidence

MSc Project - Reflective Essay

Project Title:	A Dynamic and Static Analysis-Driven Data Structures Visualisation System
Student Name:	Hongyu Wang
Student Number:	230141140
Supervisor Name:	Dr Ying He
Programme of Study:	MSc Computer Science

Analysis of Strengths and Weaknesses

Strengths

The project's primary strengths and contributions can be summarised in four key points. Firstly, it introduces a novel approach to visualisation by employing static and dynamic analysis techniques, commonly used in software reverse engineering, to acquire runtime data.

Secondly, the project combines static and dynamic analysis techniques, which are often discussed separately in the other field. This integration is achieved by initially using static analysis to identify suitable line numbers for code instrumentation, followed by dynamic analysis.

Also, the project's underlying concept is valuable: it recognizes that data structures present different states under the control of algorithms, and these algorithms are implemented through user-written methods. Consequently, the software processes user-written target methods to achieve visualisation. This approach shifts researchers' focus onto the data structures themselves, arguing that once these structures are effectively visualised, the algorithmic processes naturally become clear and comprehensible. This conceptual framework not only guides the project's implementation but also offers a fresh perspective on understanding and visualising algorithms and data structures in computer science education and research.

Finally, this project makes a novel attempt compared to other visualisation systems. It does not preset any content; instead, all visualisation content is generated in real-time based on rules by executing user-written code. In fact, if we disregard the specific characteristics of programming languages, almost all data structures are composed of linked lists, arrays, or a combination of both[1]. Therefore, there exists a unified set of rules that, once written, can potentially address all data structures, both current and future. In the early stages of the project, efforts were dedicated to implementing these unified rules, but the results were not satisfactory. In addition, I have decently followed software engineering methodologies in this project, completing the process of requirements gathering, architecture and interface design, implementation, and then testing it using three different testing methods. Regarding my personal strengths, they include the ability to proficiently use the Java language to complete development tasks, and a willingness to combine multiple fields that I have previously encountered or studied, as well as to engage in further experimentation.

Weaknesses

Next I will talk about the project itself and my personal weaknesses while completing this project. Firstly, regarding the project's unimplemented features, playback speed adjustment and search functionality were not realised. In terms of project testing, unit tests only covered the static analysis part and did not encompass all functions in the project. Concerning the software interface, it was not well-designed. The interface only includes 4 buttons and two areas for displaying visualisation and source code respectively. Although users can click on visualisation content to view internal information and selectively observe specific steps, overall, it lacks interactivity. I collected responses from a total of 13 users for our survey, which is a very small sample size. Although we increased the number of questions to gather more data to address the issue of a small sample size, compared to a rigorous user questionnaire survey, the survey in this project should be more appropriately defined as a pilot study. This situation primarily resulted from the lack of an effective promotional strategy and the failure to disseminate information about the project through a wider range of media channels.

There's a more intricate issue concerning the handling of HashMap and HashSet data structures. In Java, HashMap and HashSet use linked lists to resolve hash collisions. However, when the amount of stored data becomes excessively large, the linked list used for resolving hash collisions actually transforms into a red-black tree, allowing for a graceful performance degradation[2]. Nevertheless, hash collisions themselves are rare occurrences, as Java strives to avoid them. Consequently, the HashMap visualised by the system ultimately appears indistinguishable from an array, and users cannot observe the phenomena of list conversion or tree conversion because the conditions for these occurrences are too stringent. I only realised this issue at the final moment and felt at a loss. Eventually, I attempted to recreate my own versions of HashMap and HashSet. These versions could demonstrate the list conversion and tree conversion during hash collisions by limiting the length of the hash array. However, the visualisation of this version was ultimately unsatisfactory and was therefore abandoned on the project's debug console (As shown in Figure1).

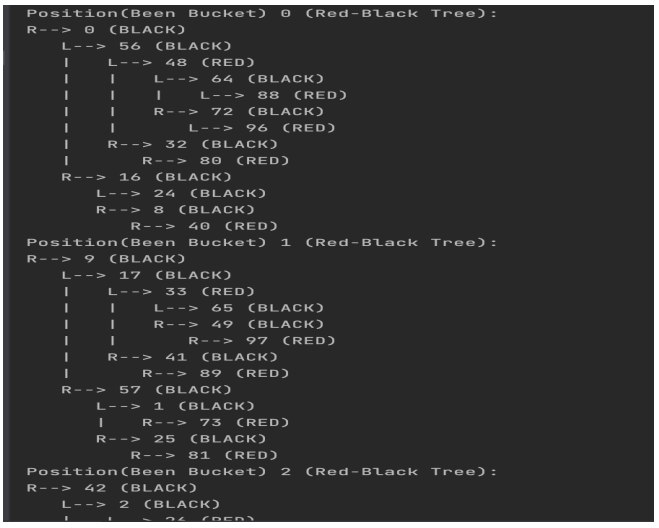


Figure1

Regarding my personal aspects, there are primarily four main weaknesses.

First, despite having long-term planning abilities, I lack the skill to establish detailed milestones or set short-term task deadlines, leading to frantic moments and anxiety due to time constraints. For instance, in the project's initial phase, I failed to set timely deadlines for experimenting with code instrumentation methods, wasting significant time on individual trial and error. Second, I frequently underestimate the time required for writing and debugging project code, often allocating insufficient programming time, which resulted in using much of the time originally intended for thesis writing on coding instead. I also struggled with excessive perfectionism during the project. While striving for perfection isn't inherently problematic, when personal resources are insufficient to achieve good results within a limited time, it often becomes counterproductive. Consequently, I often wasted considerable time obsessing over the implementation of single functions, highlighting the need to balance perfectionism with practical time management in future projects. Finally, I was slow to respond to the latest technologies such as AI models, and it was difficult for me to learn to apply them in a short period of time. These technologies might bring new ideas to the project.

Possible Further Work

Given sufficient time, there are four key tasks that should be completed to enhance the project.

- First, I should improve the graphical interface by adding a new feature for users to upload methods they want to visualise and incorporating a search window in the main interface to filter multiple visualisation views.
- Second, extend the visualisation capabilities to include custom Java objects, allowing for the visualisation of all custom data structures beyond the nine from the Java Collections Framework currently supported.
- Third, I will also focus on software architecture improvements, including further separation and encapsulation of components, developing a distinct frontend and backend architecture, and providing interface services in the form of an API.
- Lastly, I need to complete the task of transforming the collected HashMap or HashSet type data to ensure their distinctive features are accurately represented in the visualisation interface.

These improvements would significantly enhance the functionality, usability, and scalability of the project, providing a more comprehensive and user-friendly visualisation tool for Java data structures and algorithms.

Critical Analysis Between Theory & Practical Work

During this project, I became acutely aware of the gap between theory and practice. This project was self-purposed, essentially an extension of my undergraduate thesis. I believe that knowledge of algorithms and data structures forms the foundation and core of computer science, but they are abstract concepts. Therefore, I was fascinated by the idea of making them more concrete and visualizable.

However, the available resources in terms of academic papers and project references in this field are quite limited. As I mentioned earlier, people tend to create projects with preset

content because the number of frequently used data structures is very limited. While using preset content is a reliable development approach, it lacks innovation, which consequently leads to fewer published papers on the subject. Additionally, such tools are often used for higher-level educational research, and the tools themselves are often overlooked.

Although the concept underlying this project (presenting the algorithm execution process by continuously displaying different states of data structures) is appealing, it encountered problems in actual development. For instance, what would happen if I only performed a search or traversal algorithm on a data structure? Obviously, its state would not change at all during this process because I'm not modifying any elements, but merely searching. In such cases, the idea of 'presenting the algorithm execution process by continuously displaying different states of data structures' would prove ineffective.

Regarding more specific differences between theory and practice, the visualisation of HashMap and HashSet stands out. In theory, these data structures should demonstrate the conversion process between linked lists and red-black trees to handle hash collisions. However, in practice, due to Java's effective avoidance of hash collisions, I found it challenging to present these complex internal mechanisms in our visualisation. This resulted in a final visualisation that diverged from the ideal state, appearing more like a simple array structure.

Furthermore, while the project generally followed software engineering methods for design, implementation, and testing, we encountered practical challenges. The complex coupling between the interface and controller code made unit testing difficult. As a result, our unit tests only covered the static analysis portion and did not encompass all project functions. This discrepancy highlights the challenge of balancing test coverage and development progress in real-world projects.

Lastly, in terms of user survey, I initially planned to conduct a large-scale user questionnaire survey. However, due to limitations in our promotional strategy, we only collected 13 samples, making our study more akin to a pilot study rather than a comprehensive survey. This difference serves as a reminder of the need for better planning and execution of user research in future projects.

Awareness of Legal, Social, Ethical Issues and Sustainability

During the development of this Java data structure and algorithm visualisation tool, we encountered several direct legal and ethical issues, particularly in the final stage of the project when conducting user questionnaire surveys.

Firstly, regarding data privacy, although our survey did not require approval from an ethics committee, we only collected necessary information and ensured all data was anonymized. We needed to identify whether users were beginners, which was determined through the questionnaire questions themselves, without requiring users to provide any personal information. In terms of informed consent, we ensured that all survey participants were fully informed and gave their explicit consent. We provided a clear explanation of the survey's

purpose and data usage policy at the beginning of the questionnaire. Users could only proceed with answering questions after fully agreeing to these terms.

Concerning participant rights, we respected the autonomy of participants. The user consent form informed them of their right to withdraw from the survey at any time and request the deletion of their data. For data security, all data was stored in Google Cloud services that require password authentication. Regarding sustainability considerations, firstly, the project is scalable. We conducted unit tests, unit tests can ensure that old functionalities would not be broken when new features are added, increasing developer confidence[3].

These measures demonstrate our commitment to ethical research practices and sustainable development, ensuring that our project not only meets current legal and ethical standards but is also prepared for future growth and adaptation.

References

[1] Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C., 2022. *Introduction to algorithms*. MIT press.

[2] Bajracharya, D. and Kathmandu, N., A review on java hashmap and treemap.

[3] Zhu, H., Hall, P.A. and May, J.H., 1997. Software unit test coverage and adequacy. *Acm computing surveys (csur)*, *29*(4), pp.366-427.