Electrical Engg. Department

Third Year, 1st Semester

Advanced Programming Languages (Java)

Aswan University

Faculty of Engineering

DSA Visualizer

Supervised by:

Dr. Ali AbdElAziz

Student Names:

Moaaz Mahmoud

Ziad Hussien

AbdElRahman Yasser

Mohamed AbdElRahim

Abstract

This project is mainly an application that performs visualization—that is, presents a view of the content—for some well-known data structures, namely the linked list, stack, queue, and heap. For every data structure it supports, it begins with an empty instance of that structure, and allows the user to insert and remove values into and from it. After every operation, either insertion or removal, the application provides a view of the state of that instance.

The application also performs performance analysis for some of the data structures. This is by viewing number-of-operations–time plots for the basic operations of a certain data structure. It is also possible to export the analysis result to a csv file.

1. Introduction

In this project, we have focused on three main goals:

- Visualize at least one sequential data structure, and we did for three.
- Visualize at least one hierarchical data structure, and we did.
- Display performance plots for at least two data structures, and we did for three, or in a sense, five; as each of the stack and the queue has two implementations: array-based and link-based.

We have used several tools and technologies in order to get this project finished. Here is a list of them:

- JavaFX v11.0.2.
- Scene Builder v8.5.
- IntelliJ IDEA 2021.2.3.

and we used JDK v17.0.1.

2. Project Description

As mentioned before, this project performs two main functions — visualization and performance analysis. We will next discuss the project in more details. We will break the discussion into four parts: project structure, layout details, visualization functionality, and performance analysis functionality.

2.1. Project Structure

This project has five categories of files:

- The main files, Main.java and Application.java. The file Main.java contains the `main` methods which calls the `start` method to begin the application.

- Implementation files. Implement the data structures used in the application like the array-based stack.
- Controller files. Perform event handling and connect the implementation files to the layout FXML files. The implementation and controller files are listed below:
  - ArrayQueue.java
  - ArrayStack.java
  - Controller.java
  - Heap.java
  - HeapPage.java
  - LinkedListPage.java
  - LinkedQueue.java
  - LinkedStack.java
  - LogarithmicAxis.java
  - Node.java
  - PerformanceAnalysisPage.java
  - QueuePage.java
  - StackPage.java

- Layout files. FXML files that contain the GUI components that realize the layout of the project. The layout files are listed below:
  - heap-page.fxml
  - home-page.fxml
  - linked-list-page.fxml
  - performance-analysis-page.fxml
  - queue-page.fxml
  - stack-page.fxml

- CSS stylesheets. A single CSS file for styling various layout components.
  - component-styles.css

2.2. Layout Details and User Experience

The Home Page

When the user first opens the application, the home page shown in Fig. 1 appears. This page redirects the user to one of five pages, four of which are for visualization and the other is for performance analysis.

Fig. 1. The home page of the application.

The Visualization Pages

The pages where the user can visualize the linked list, stack, queue, and heap are shown figures 2, 3, 4, and 5 respectively with visualizations for sample structure states. Each of these pages contains textboxes and buttons as needed to perform the basic operations on the structure related to the page.

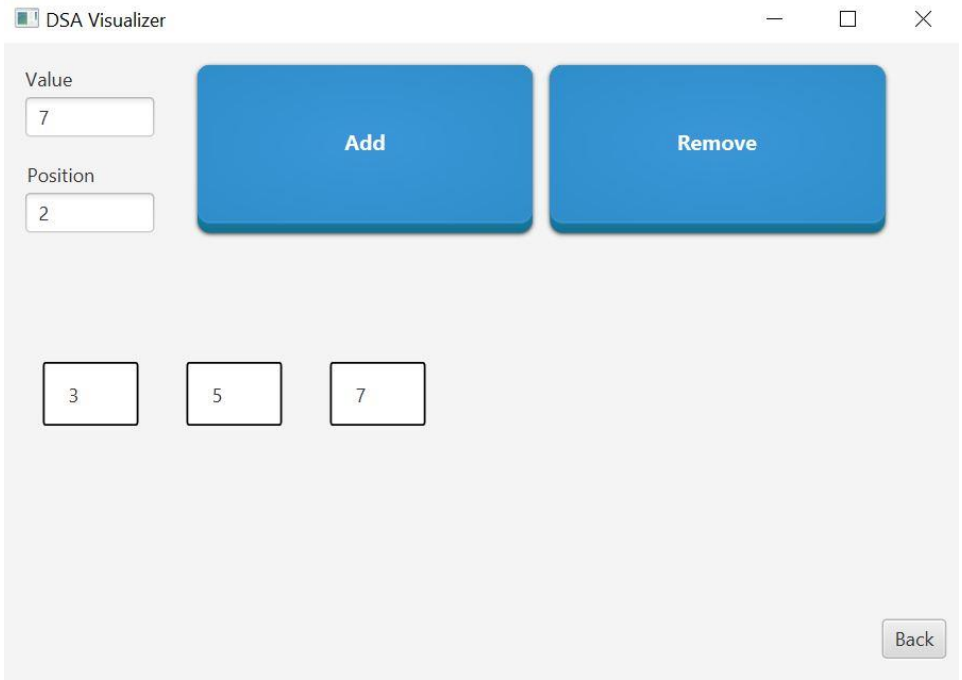

Fig. 2. The linked list visualization page.

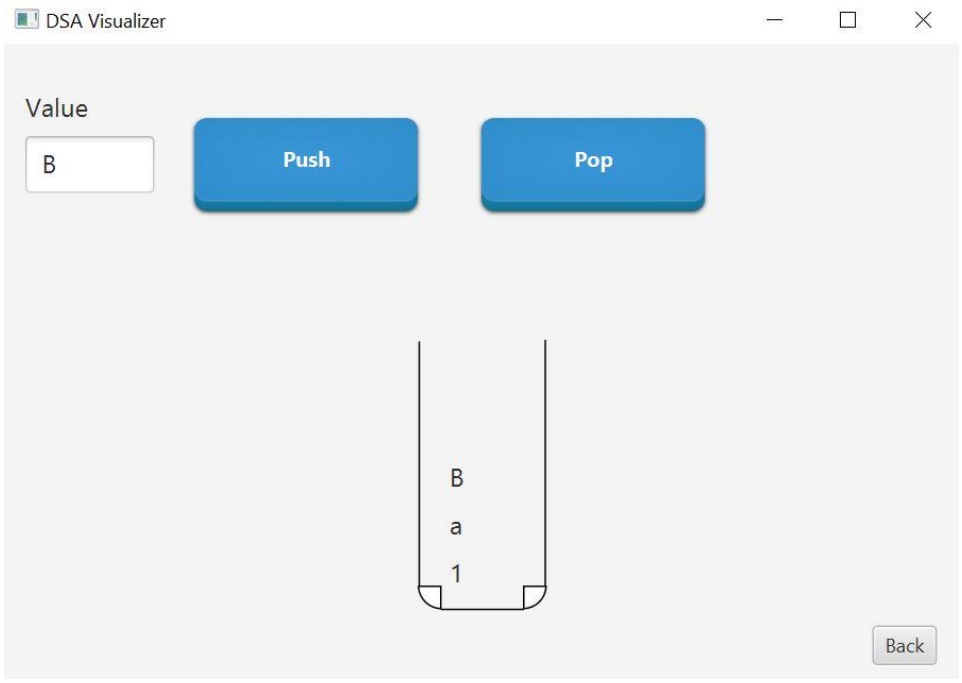

Fig. 3. The stack visualization page.

Fig. 4. The queue visualization page.

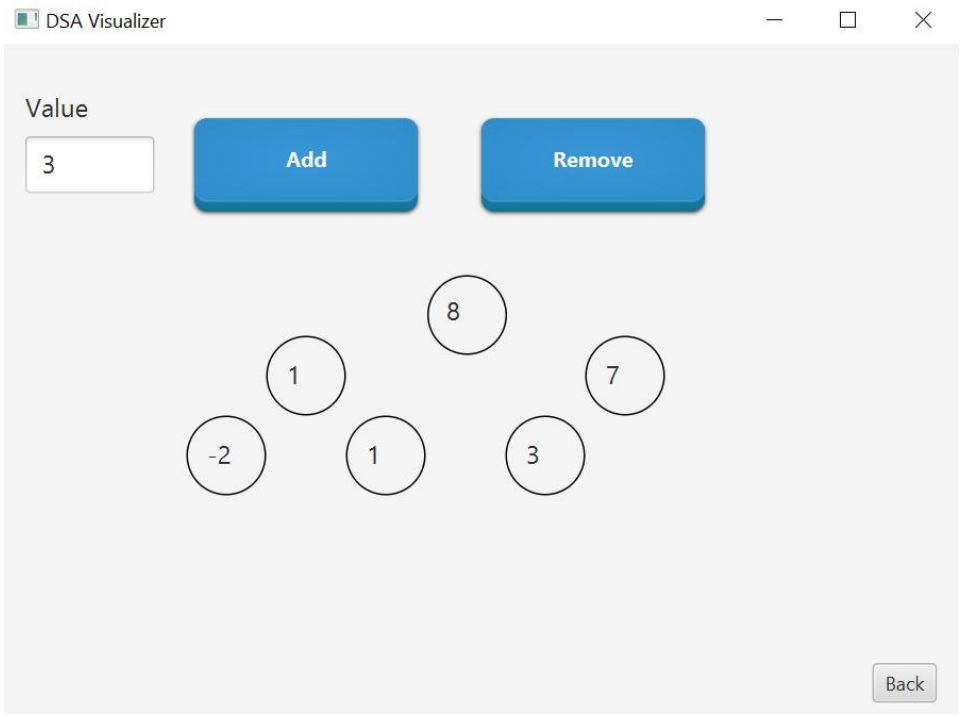

Fig. 5. The heap visualization page.

The Performance Analysis Page

The performance analysis page, as mentioned before, allows the user to test the performance of some of the structures supported by the application. The main components of this page are:

- A chart for displaying the number-of-operations–time plots.
- For every data structure:
    - A combo box to specify the implementation—whether link- or array-based (if applicable).
    - A button that triggers the insertion operation performance test.
    - A button that triggers the removal operation performance test.

A snapshot of the page is shown in Fig. 6 and sample visualizations are shown in Fig. 7.

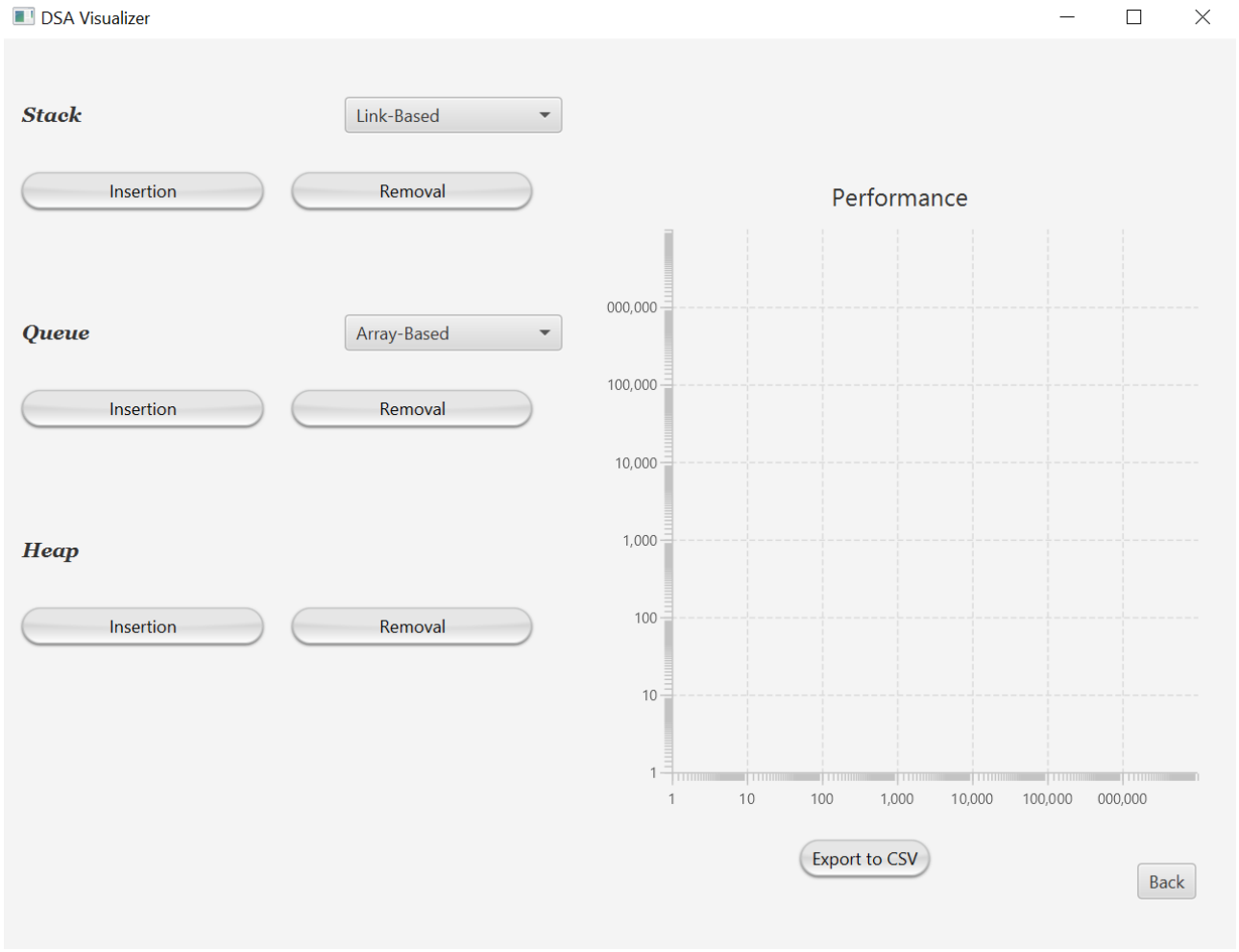

Fig. 6. The performance analysis page.

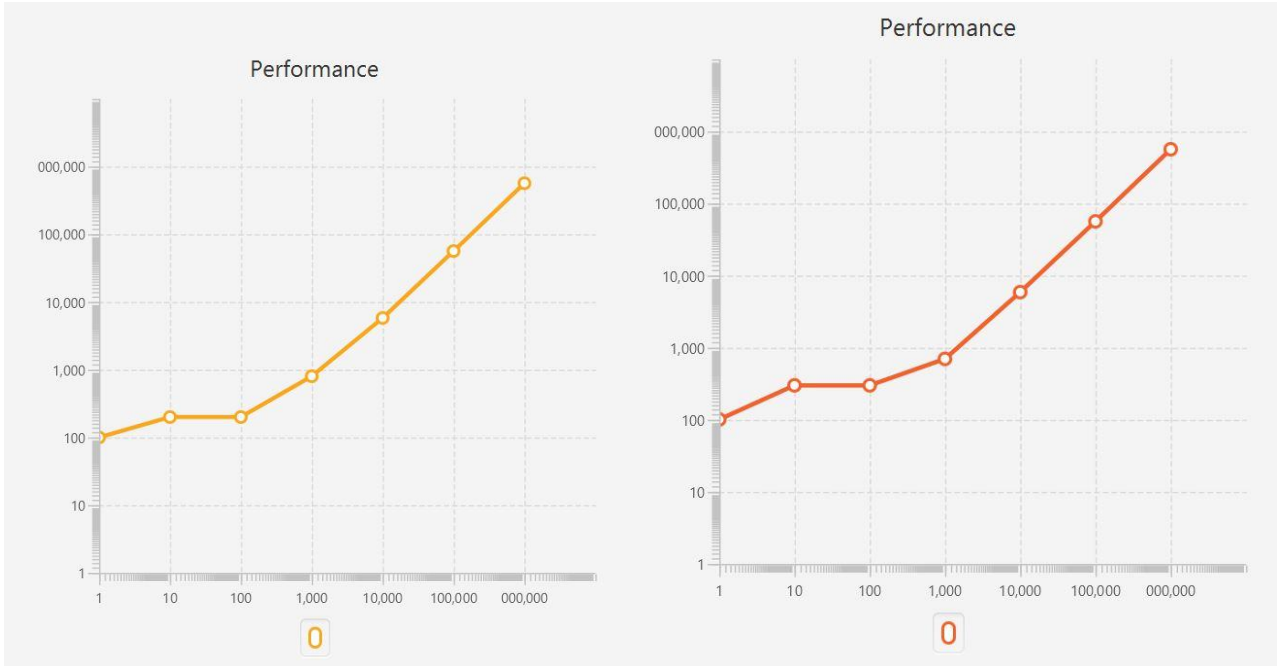

Fig. 7. Sample plots by the application.

2. 3. Visualization

In this section, we present the algorithms (or the general strategies) we followed in order to visualize the data structures in this application.

The Linked List

For the linked list, we used an array `Rectangle`s and an array of `Label`s to hold the values of the list elements. The user specifies the value and position of the element to be inserted or the position of the element to be removed and then specifies the operation. After that, assuming the list has n elements, we set the first n `Label`s and n `Rectangle`s to be visible and display the elements on them.

The Stack

In order to visualize the stack, we keep track of the number of elements currently in the stack and keep an array of `Label`s vertically stacked on one another.

Whenever the user performs a push operation, we set the first invisible `Label` to visible and display the new value on it. If the number of elements on the stack is equal to the size of the array, we do nothing.

Whenever the user performs a pop operation, we set the last visible `Label` to be invisible. If the stack is empty, we do nothing.

The Queue

We visualize the queue in pretty much the same way we visualize the linked list. When the user performs an enqueue or a dequeue operation, we visualize the queue with an array of `Label`s and `Oval`s by shifting the elements as necessary and setting some `Label`s to visible/in visible.

The Heap

Due to the hierarchical nature of the heap, the task of visualizing it was a rather challenging one. Fortunately, because the heap is by definition a complete binary tree, we could use a similar approach.

We added a method to the heap class we implemented that returns an array containing the elements of the heap. We then added a group of `Label`s to the heap page in a way that resembles a binary tree and enumerated them top-down and from right to left. In the end, it was pretty straightforward to reflect the heap content on the `Label`s in a similar way as we did before with the sequential data structures.

2.4. Performance Analysis

We followed the following strategy to generate our plots:

1. Initialize an empty structure $struc$.
2. Initialize an empty array $times$.
3. Set $operationCount = 1$
4. While $operationCount < 10$
    i. Let the $start$ be the value of the current time.
    ii. Perform $operationCount$ operations on the structure.
    iii. Let end be the value of the current time.
    iv. Store the difference between $start$ and end in $times$.
5. Visualize $times$ on the chart.

3. Individual Team Members' Roles

Moaaz Mahmoud and Ziad Hussien

- Implemented the data structures used
    - Linked list
    - Stack (array-based and link-based)
    - Queue (array-based and link-based)
    - Heap
- Designed the visualization algorithms for the data structures
- Implemented the heap page layout
- Implemented the heap visualization functionality
- Implemented the performance analysis page layout
- Identified the strategy used to test the performance of the data structures
- Implemented the performance analysis functionality
- Refactored some CSS stylesheets from the web and used them for styling some UI components
- Added the 'Export to CSV' feature

AbdElRahman Yasser

- Implemented the stack page layout
- Implemented the stack visualization functionality relying on the Stack class from the Java collections API
- Implemented the queue page layout
- Implemented the queue visualization functionality relying on the Queue class from the Java collections API

Mohamed AbdElRaheem

- Implemented the home page layout

- Implemented the home page functionality
- Implemented the linked list page layout
- Implemented the linked list visualization functionality using the ArrayList collection from the Java collections API