Data Structures and Algorithms

COMP1002

**Project Report**

**Assignment Semester 2 2022**

Sean Anain

20324861

Contents

[User Guide 1](#_Toc115706151)

[Purpose 1](#_Toc115706152)

[Interactive Mode 1](#_Toc115706153)

[Exit Program 1](#_Toc115706154)

[Load Keyboard File 1](#_Toc115706155)

[Node Operations 1](#_Toc115706156)

[Edge Operations 2](#_Toc115706157)

[Display Graph 2](#_Toc115706158)

[Display Graph Information 2](#_Toc115706159)

[Enter a String for Finding a Path 2](#_Toc115706160)

[Generate Paths 2](#_Toc115706161)

[Display Paths 3](#_Toc115706162)

[Save Keyboard 3](#_Toc115706163)

[Silent Mode 3](#_Toc115706164)

[Description of Classes 4](#_Toc115706165)

[DSALinkedList 4](#_Toc115706166)

[DSAListNode 4](#_Toc115706167)

[DSALinkedListIterator 4](#_Toc115706168)

[DSAQueue 4](#_Toc115706169)

[DSAStack 4](#_Toc115706170)

[DSAGraph 4](#_Toc115706171)

[DSAGraphVertex 4](#_Toc115706172)

[DSAGraphEdge 4](#_Toc115706173)

[Justification of Decisions 5](#_Toc115706174)

[Use of DSAGraph 5](#_Toc115706175)

[Use of DSAGraph.DSAGraphVertex 5](#_Toc115706176)

[Use of DSALinkedList 5](#_Toc115706177)

[Use of DSAQueue 5](#_Toc115706178)

[Use of DSAStack 5](#_Toc115706179)

[Depth-First Search 5](#_Toc115706180)

[Breadth-First Search 5](#_Toc115706181)

[UML Class Diagram 5](#_Toc115706182)

[Traceability Matrix 5](#_Toc115706183)

[Showcase 5](#_Toc115706184)

[Introduction 5](#_Toc115706185)

[Scenario 1 5](#_Toc115706186)

[Scenario 2 5](#_Toc115706187)

[Scenario 3 5](#_Toc115706188)

[Conclusion 5](#_Toc115706189)

# User Guide

Ensure all source code and all supporting code and files are all within the same directory. Compile all code using “javac \*.java”. When starting program, there are 3 starting options:

* No command line arguments: provides usage information
* “-i”: interactive testing environment (java keyMeUp -i)
* “-s”: silent mode (java keyMeUp -s)

Note that vertices refer to the keys themselves and edges refer to the adjacency of two respective “keys”.

## Purpose

Purpose of this program to represent a keyboard in the form of a graph which allows the user to enter a string to which the program can determine the shortest path to “type” the entered string using different graph traversal methods.

## Interactive Mode

When starting in interactive mode, the following menu is presented:

1. Exit program
2. Load keyboard file
3. Node operations
4. Return to main menu
5. Find
6. Insert
7. Delete
8. Update
9. Edge operations
10. Return to main menu
11. Find
12. Add
13. Remove
14. Update
15. Display graph
16. Display graph information
17. Enter string for finding path
18. Generate paths
19. Display path(s)

y. Save results to file

1. Save keyboard

Each feature can be accessed by entering the respective integer associated with each menu choice into the terminal.

### Exit Program

This option promptly ends the program.

### Load Keyboard File

Selecting this option will prompt the user to enter a file name in the format <filename.fileType>. This file will then be read and converted into a graph to be used in the program.

### Node Operations

After selecting this option, the user will be presented a second menu to which they will be prompted to enter an integer associated with the desired option

#### Return to main menu

Selecting this option returns user to main menu.

#### Find

The user will be prompted to with a request to enter a label. This label will be associated with the vertex the user is trying to find. Enter a desired label into the terminal and the user will be informed if the associated vertex exists in the current graph or not.

#### Add

The user will be prompted to enter a label for a new vertex. They will then be requested to enter a value for that vertex. This vertex will then be added to the current graph.

#### Delete

The user will again be prompted to enter a label of the vertex they intend to remove. They will then be informed whether the associated vertex for that label exists. If the vertex exists, it will be removed from the current graph.

#### Update

The user will be prompted to enter the label of vertex they intend to edit. If such vertex exists, they are then prompted to enter a new label for the vertex. This vertex will not be referenced by that new label throughout the entire graph

### Edge Operations

After selecting this option, the user will be presented a second menu to which they will be prompted to enter an integer associated with the desired option

#### Return to main menu

Selecting this option returns user to main menu.

#### Find

The user will be prompted to with a request to enter two labels. These labels will be associated with the vertices containing the edge the user is trying to find. Enter a desired label into the terminal and the user will be informed if the associated vertex exists in the current graph or not. Note that the order of vertices entered will change whether a edge will be located or not.

#### Add

The user will be prompted to enter two labels associated with two existing vertices. If both vertices exist and an edge does not already exist between them in the direction chosen by the user, a edge will be added between the two vertices

#### Delete

The user will be prompted to enter two labels associated with two existing vertices. If both vertices exist and an edge does exist between them, this edge will be removed from the current graph

#### Update

CHANGE THIS

### Display Graph

Selecting this option will display the graph as both an adjacency matrix and list of adjacencies.

### Display Graph Information

Selecting this option will display relevant information regarding the current graph such a number of vertices and edges.

### Enter a String for Finding a Path

The user will be prompted to enter a string they intend to find a path for.

### Generate Paths

Selecting this option will internally generate the paths for the inputted string.

### Display Paths

Selecting this option will display the two paths generated by the program. Each path will be printed on several lines, with each line conveying the path between two “keys” within the respective string. Following this, the number of moves will also be displayed, allowing the user to easily rank the two paths against one another. Following this, the user will be prompted to enter ‘y’ or ‘n’ regarding whether they would like the paths saved. Should the user select ‘y’, the paths will be printed to “results.txt” in the same format it was displayed on the terminal.

### Save Keyboard

Selecting this option will the save the keyboard to “output.al” in the same format as the rest of the existing keyboard files.

## Silent Mode

When running the program in silent mode, all interaction with the user will be done in the command line. When running the program, run with the following command line arguments:

Java keyMeUp -s keyboardFile stringFile pathFile

Ensure all files are entered in the format <filename.fileType>. The paths can then be viewed by the user by opening the file entered for “pathFile”.

# Description of Classes

## DSALinkedList

The purpose of this class is to create the data structure that is the double ended, doubly linked, linked list. This class gives ability to store data as a linked list and allows for the ability to insert first, insert last, remove first, remove last, peek first, peek last, and to display. This class is very useful for the “keyMeUp” program as it allows data such as the generated paths to be stored as a linked list. This class is also the foundation for the data structures queues and stacks.

### DSAListNode

This class has an composition relationship with “DSALinkedList”. This class is responsible for the creation of the nodes used in the linked list such as the head and tail node. It also gives the ability to retrieve the value of a specific node within the list, set the value of a node, get the next node from the current, get the previous node from the current, and to set the next and previous nodes.

### DSALinkedListIterator

This class also has an composition relationship with “DSALinkedList”. This class has the purpose of creating an iterator to allow a program to iterate through the linked list. This class implements the default java class “Iterator”.

## DSAQueue

This class has the purpose of creating the data structure, queue. This class has an aggregation relationship with “DSALinkedList”. This class allows the linked list to be treated as queue by having methods that allow program to queue a value, dequeue a value, peek the front of the queue, check if the queue is empty, and the ability to iterate through the queue.

## DSAStack

This class has the purpose of creating the data structure, stack. This class has an aggregation relationship with “DSALinkedList”. This class allows the linked list to be treated as a stack by having methods that allow the program to push values onto the stack, pop values off of the stack, view the top of the stack, check if the stack is empty, and the ability to iterate through the stack.

## DSAGraph

This class has the purpose of creating the data structure, graph. This class allows the creation of several vertices (nodes) and to form edges between them. This class also has an aggregation relationship with “DSALinkedList”. This class allows programs to add vertices, remove vertices, edit vertices, add edges, remove edges, edit edges, check if a label has a vertex, get number of vertices, get number of edges, retrieve vertices using labels, return linked lists containing the adjacent vertices of one vertex, check if two vertices are adjacent, display the graph as an adjacency list, display graph as adjacency matrix, and perform breadth-first and depth-first search.

### DSAGraphVertex

This class has the purpose of creating the vertices for the DSAGraph class. This class has a composition relationship with “DSAGraphVertex”. The fields for this class include the DSALinkedList, links, Boolean variable visited, objects label and value. This class allows programs to get a vertex’s value, get all adjacent vertices, add edges, set as visited, get visited state, and clear visited.

### DSAGraphEdge

This class has the purpose of creating the vertices for the DSAGraph class. This class has a composition relationship with “DSAGraphEdge”. The fields for this class include two DSAGraphVertex “from” and “to”, objects label and value. This class allows programs to get the edge’s label, value, vertex it is coming from, vertex it is going towards, and its state of direction.

# Justification of Decisions

## Use of DSAGraph

## Use of DSAGraph.DSAGraphVertex

## Use of DSALinkedList

## Use of DSAQueue

## Use of DSAStack

## Depth-First Search

## Breadth-First Search

# UML Class Diagram

# Traceability Matrix

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Feature** | **Requirements** | **Design/Code** | **Test** |
| 1 | Driver/Menu & Modes | 1.1 System display |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# Showcase

## Introduction

## Scenario 1

## Scenario 2

## Scenario 3

# Conclusion