Document version: 1.1 (2015-11-15)

Curtin University – Department of Computing

Assignment Cover Sheet / Declaration of Originality

Complete this form if/as directed by your unit coordinator, lecturer or the assignment specification.

Last name:	Kannangara	Student ID:	20672308
Other name(s):	Don Sahas Sandaru		
Unit name:	Data Structures and Algorithms	Unit ID:	COMP1002
Lecturer / unit coordinator:	Valerie Maxville		
Assessment:	Assignment		

I declare that:

- The above information is complete and accurate.
- The work I am submitting is *entirely my own*, except where clearly indicated otherwise and correctly referenced.
- I have taken (and will continue to take) all reasonable steps to ensure my work is *not accessible* to any other students who may gain unfair advantage from it.
- I have *not previously submitted* this work for any other unit, whether at Curtin University or elsewhere, or for prior attempts at this unit, except where clearly indicated otherwise.

I understand that:

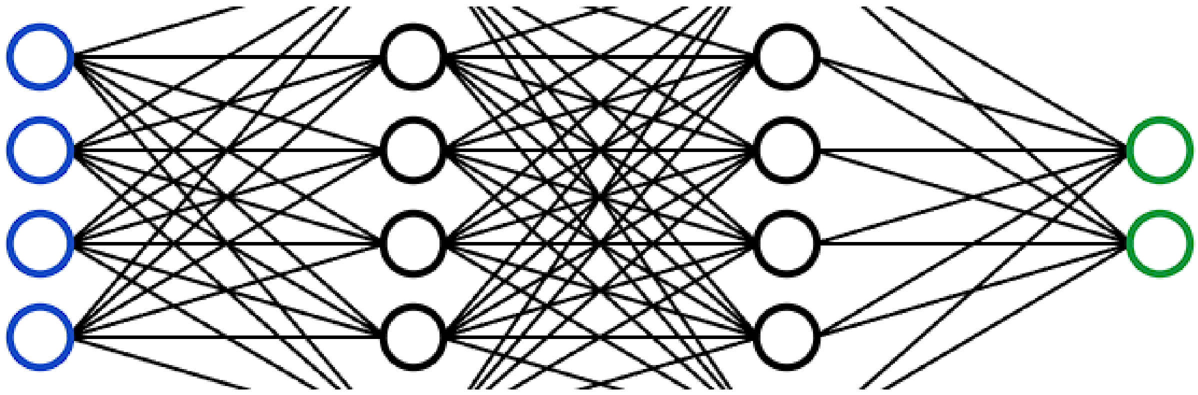
- Plagiarism and collusion are dishonest, and unfair to all other students.
- Detection of plagiarism and collusion may be done manually or by using tools (such as Turnitin).
- If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which will remain permanently on my academic record. I also risk termination from my course and other penalties.
- Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.
- It is my responsibility to ensure that my submission is complete, correct and not corrupted.

Signature: sandaru

Date of signature: 10/13/2022

(By submitting this form, you indicate that you agree with all the above text.)

DSA Assessment Report



Don Sahas Sandaru Kannangara

Curtin ID: 20672308

User Guide

BEFORE COMPILING:

Please make sure all the string files are in the present in the relevent directory and are accessible.

HOW TO COMPILE:

Complile everything with the command:

"javac *.java"

HOW TO EXECUTE:

"java keyMeUp"

FILES/DEPENDENCIES:

DSALinkedList (code for LL)

QueueUsingLinkedList - uses DSALinkedList (ADT queue has been implemented using LL)

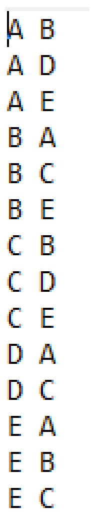
DSAGraph - uses QueueUsingLinkedList (Queue is used in graph for BFS)

keyMeUp - uses DSAGraph

UnitTestGraph - uses DSAGraph (test code to test the graph methods)

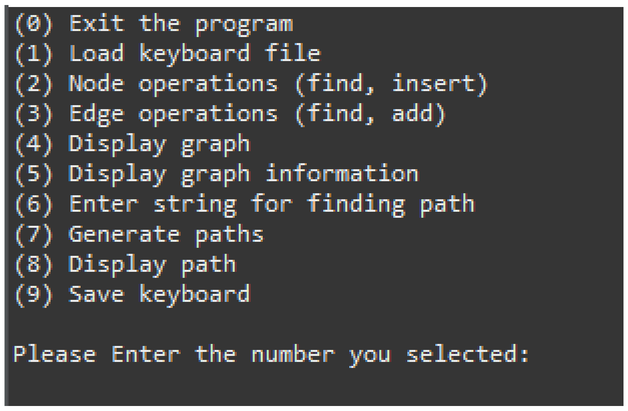
Test Scenario 1

Contents of “text.txt” :



```
A B
A D
A E
B A
B C
B E
C B
C D
C E
D A
D C
E A
E B
E C
```

As soon as the program starts, the user is greeted with a Menu.

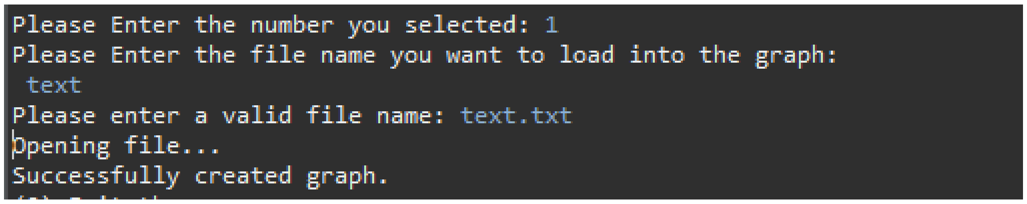


```
(0) Exit the program
(1) Load keyboard file
(2) Node operations (find, insert)
(3) Edge operations (find, add)
(4) Display graph
(5) Display graph information
(6) Enter string for finding path
(7) Generate paths
(8) Display path
(9) Save keyboard

Please Enter the number you selected:
```

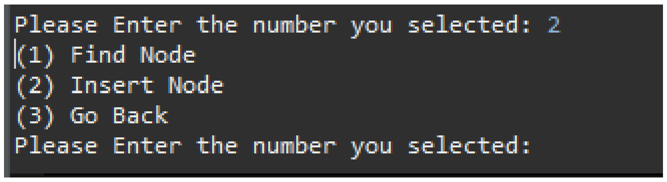
The user’s input is recorded and the action described for each number is executed or loads up another menu.

EXIT: User can stop the program if 0 is selected.

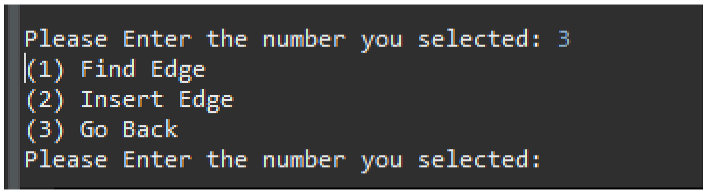
1) : user can enter the file name from which the graph has to be loaded into.


```
Please Enter the number you selected: 1
Please Enter the file name you want to load into the graph:
 text
Please enter a valid file name: text.txt
Opening file...
Successfully created graph.
```

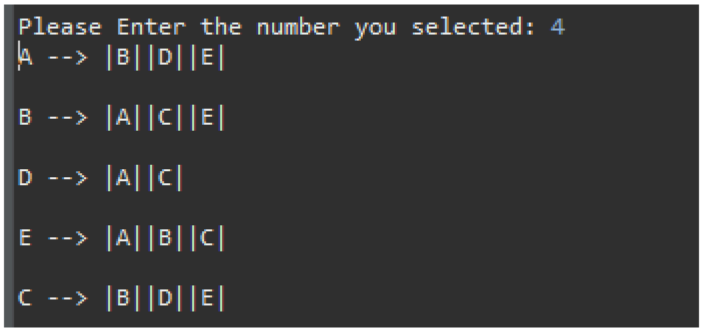
2) : vertices of the graph can be found or added


```
Please Enter the number you selected: 2
(1) Find Node
(2) Insert Node
(3) Go Back
Please Enter the number you selected:
```

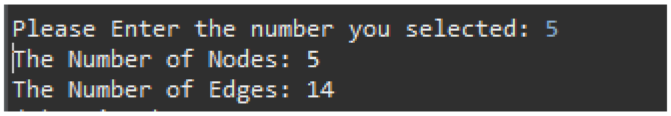
3) : edges of the graph can be found or added


```
Please Enter the number you selected: 3
(1) Find Edge
(2) Insert Edge
(3) Go Back
Please Enter the number you selected:
```

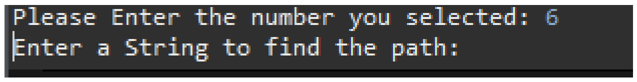
4) : Displays the graph


```
Please Enter the number you selected: 4
A --> |B||D||E|

B --> |A||C||E|

D --> |A||C|

E --> |A||B||C|

C --> |B||D||E|
```

5) : Displays the number of vertices and edges

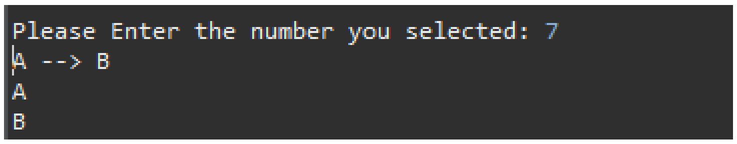

```
Please Enter the number you selected: 5
The Number of Nodes: 5
The Number of Edges: 14
```

6) : Accept the input for the string to search



```
Please Enter the number you selected: 6
Enter a String to find the path:
```

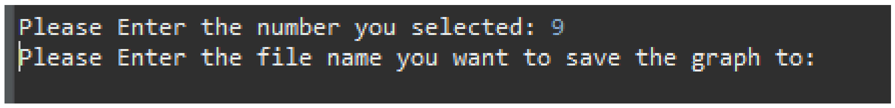
7) : Generate Paths for the entered string



```
Please Enter the number you selected: 7
A --> B
A
B
```

8) : Displays the path that was generated

9) : Save the keyboard



```
Please Enter the number you selected: 9
Please Enter the file name you want to save the graph to:
```

Description of Classes

DSALinkedList

The graph nodes are stored in the Linked List, it allows to dynamically change capacity of the storage.

DSAListNode

This DSAListNode class allows us to change the node according to the requirements which allows more flexibility when working with LinkedLists.

DSAGraph

The DSAgraph class is used to keep track of all the keys in the keyboard, as each key represents a node and each proximity to different keys are recorded as edges.

UnitTestDSAGraph

This class includes a main class of its own, and is used to test all the functionalities of the graph which includes all edge and node functions as well as BFS traversal methods. It does not require any text files in order to execute so it is more easier to isolate errors if found.

Justification of Decisions

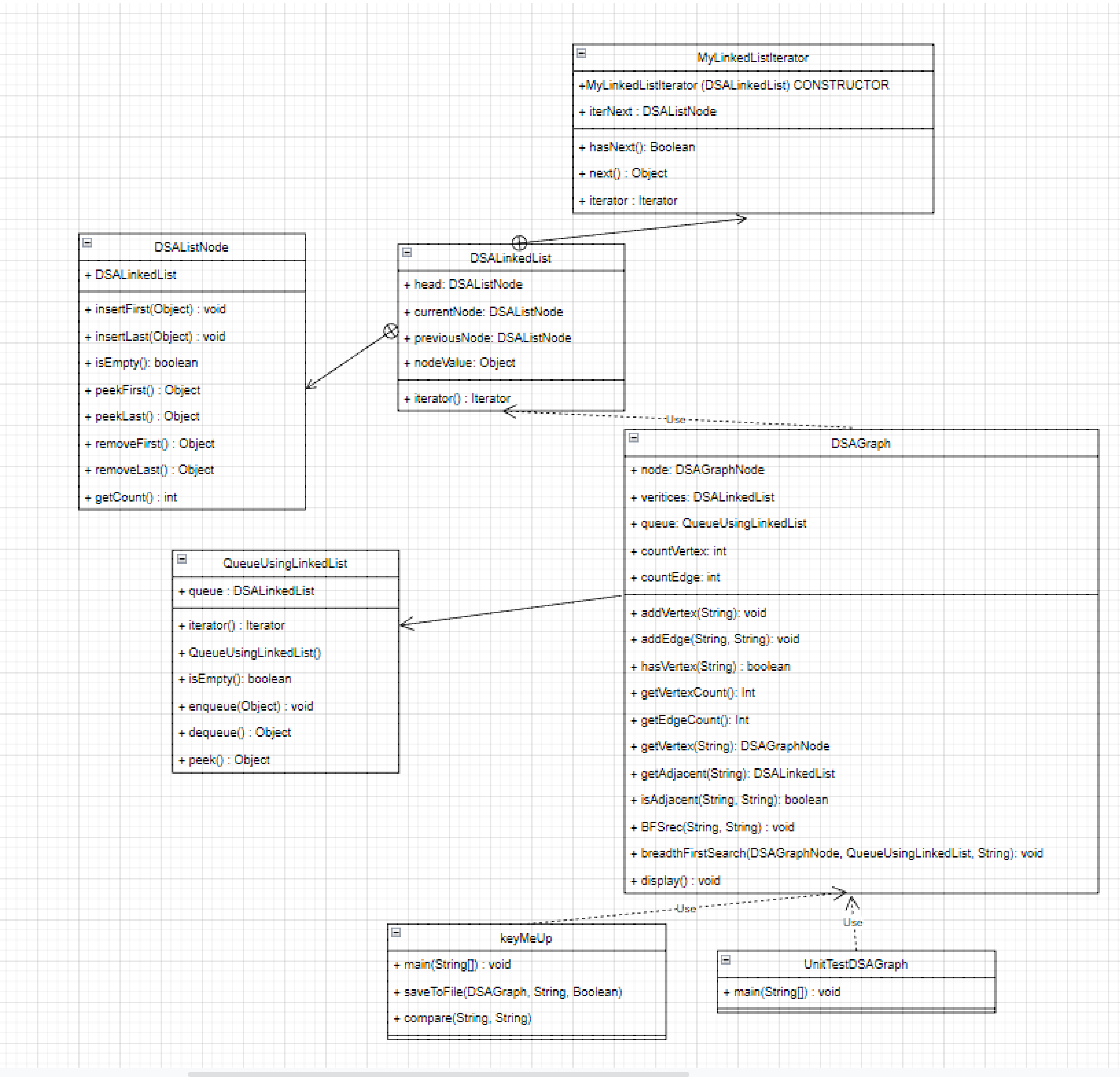
I implemented a graph for this program as it allowed me to implement Breadth First Search as I understood it is a convenient method to find the shortest path through the edges which was the main requirement for this program.

Linked list was also used for the implementation of the graph.

Queue using Linked List was used for the breadth First implementation as well.

Alternate ADT that I was considering of using was a tree but when I found graph implementation to be easier.

UML Class diagram



Traceability Matrix

Feature	Code	Test
0. Mode & Menu		
0.1 Interactive Mode	keyMeUp.java | main() | lines 14 - 254	[PASSED] Interactive mode menu is displayed.
0.2 Silent Mode	keyMeUp .java | main() | -	[FAILED] Silent mode has not been implemented
1. Load Input File	keyMeUp .java | main() lines 45 - 91	[PASSED] It prompts to enter a file name which iterates till a valid name is entered.
2. Node Operations		
2.0 Find	keyMeUp .java | main() lines 110 - 120	[PASSED] Prompts to enter a node, displays true of false if the node is found in the graph or not accordingly.
2.1 Insert	keyMeUp .java | main() lines 121 - 126	[PASSED] It prompts the user to enter a Label and adds it as a vertex to the graph.
2.2 Delete	keyMeUp .java | main() --	[FAILED] Delete not implemented.
2.3 Update	keyMeUp .java | main() --	[FAILED] Update not implemented.
3. Edge Operations		
3.0 Find	keyMeUp .java | main() lines 151- 163	[PASSED] Prompts to enter two nodes, displays true of false if the edge is found in the graph or not accordingly between the two nodes.
3.1 Add Edge	keyMeUp .java | main() lines 164- 171	[PASSED] It prompts the user to enter the two labels for the two edges and enters them into the graph.
3.2 Delete Edge	keyMeUp .java | main() --	[FAILED] Delete not implemented.
3.3 Update Edge	keyMeUp .java | main() --	[FAILED] Update not implemented.
4. Display Graph	keyMeUp .java | main() lines 180 - 188	[PASSED] The display function works and displays edges entered the system.

5. Display graph information	keyMeUp.java | main() lines 191 - 193	[PASSED] The edge count and the vertex count successfully displays according to the current graph loaded in.
6. Enter string for finding path	keyMeUp.java | main() lines 196 - 198	[PASSED] The program successfully takes a string input form that user.
7. Generate paths	keyMeUp.java | main() lines 203 - 217	[FAILED] The program always traverses the entire graph instead of stopping at the desired node, thus the shortest distance is not found.
8. Display path	keyMeUp.java | main() lines 219 - 232	[FAILED] The code has been implemented to save either the keyboard layout or the shortest path accordingly but an exception is met whenever either save function is executed.
9 &10. Save layout and path	keyMeUp.java | main() lines 219 - 247	[FAILED] The code has been implemented to save either the keyboard layout or the shortest path accordingly, but an exception is met whenever either save function is executed.
(0) Exit	keyMeUp.java | main() lines 42 - 44	[PASSED] All exit methods work across the program which allow the user to go back to the main meu without restarting the entire program.

Conclusion and Future Work

Issue in BNF:

The Breadth First search is going through the entire graph even though a condition is in place to check if the searched vertex is matched so as a result, I cannot generate a shortest path. As for future work that method needs to be altered so that it accurately gives out the shortest distance throughout the graph.

GUI:

For future upgrades to this program, I recommend a implementing a Graphical User interface as the Command Line interface is confusing.