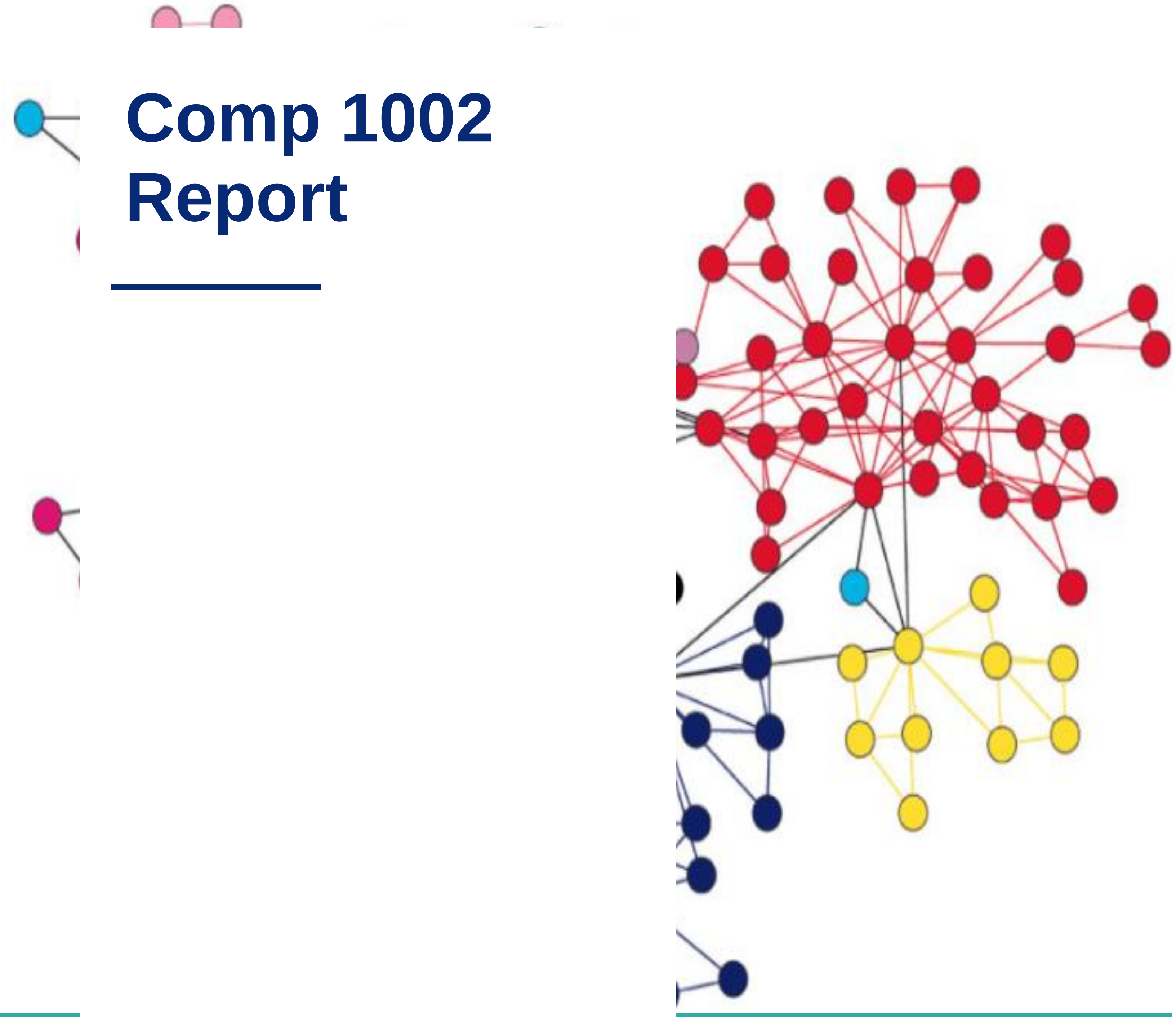


Comp 1002 Report

OCTOBER 10

Curtin University

Authored by: Jawagar Prabaharan

Curtin ID - 20944793

Assignment 1 Report

Table of contents

1. Introduction
2. User guide
3. Description on the classes and methods used
4. Justification of decisions
5. UML Diagrams
6. Traceability matrix
7. Showcase of scenarios
    - Introduction
    - Scenario 1
    - Scenario 2
8. Conclusion

Introduction

This assignment is the first take home assignment that we received in this module(Comp 1002). The main focus of the assignment was to test the knowledge students possess in regard to ADT's(abstract data types) namely graphs, linked lists and queues. This assignment required us to continue developing on the graph code that we submitted in our 6th practical, and wanted us to add certain functionalities as requested. Now I move onto the next segment the user guide.

User guide

This program that I have created can be run in three modes as mentioned in the assignment.

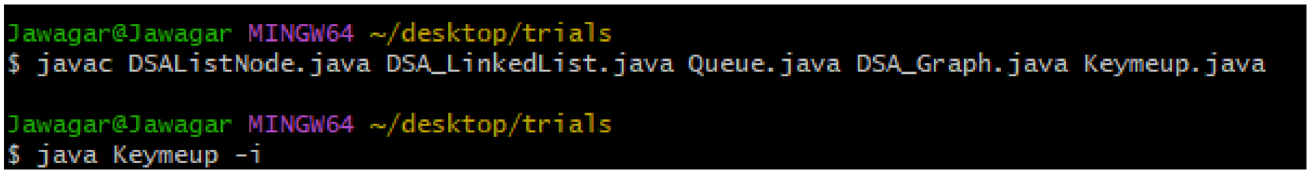
1. No command line arguments
2. Interactive testing environment
3. Silent mode

1. No command line arguments

in this method there will be no command line arguments given, just the manual compilation would take place and then the options to run the program such as the interactive testing environment and the silent mode would be displayed and the user would be given an idea on how to run the program.

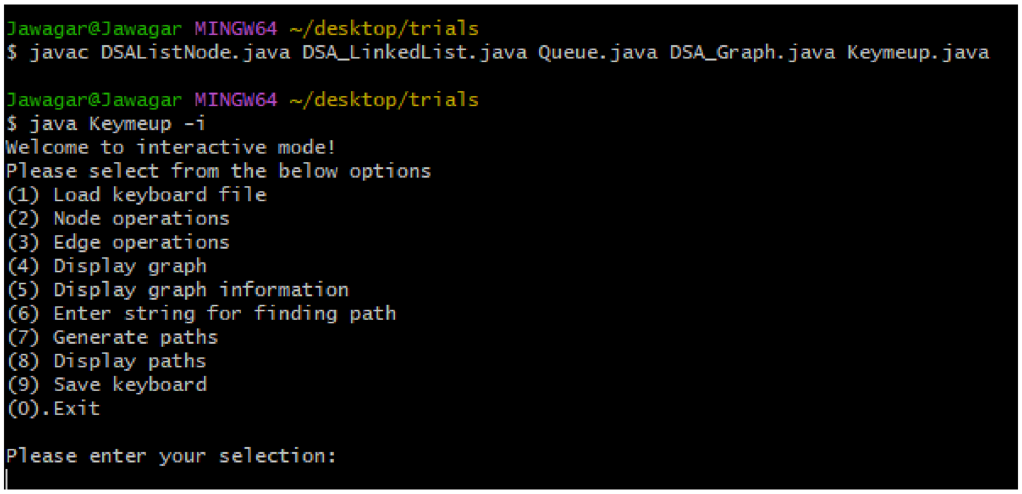
2. Interactive testing environment

In this method the user should use the command "-i" along the file name when compiling it which would lead to the program displaying the interactive menu to the user from which the user can select what operations to conduct.



```
Jawagar@Jawagar MINGW64 ~/desktop/trials
$ javac DSAListNode.java DSA_LinkedList.java Queue.java DSA_Graph.java Keymeup.java

Jawagar@Jawagar MINGW64 ~/desktop/trials
$ java Keymeup -i
```

The program would start in the interactive mode when compiled in the manner mentioned in the picture above.
Then this would lead to a menu being presented to the users only in the interactive mode as mentioned in the picture below.



```
Jawagar@Jawagar MINGW64 ~/desktop/trials
$ javac DSAListNode.java DSA_LinkedList.java Queue.java DSA_Graph.java Keymeup.java

Jawagar@Jawagar MINGW64 ~/desktop/trials
$ java Keymeup -i
Welcome to interactive mode!
Please select from the below options
(1) Load keyboard file
(2) Node operations
(3) Edge operations
(4) Display graph
(5) Display graph information
(6) Enter string for finding path
(7) Generate paths
(8) Display paths
(9) Save keyboard
(0).Exit

Please enter your selection:
```

the user can continue using the program by selecting the options they like to do

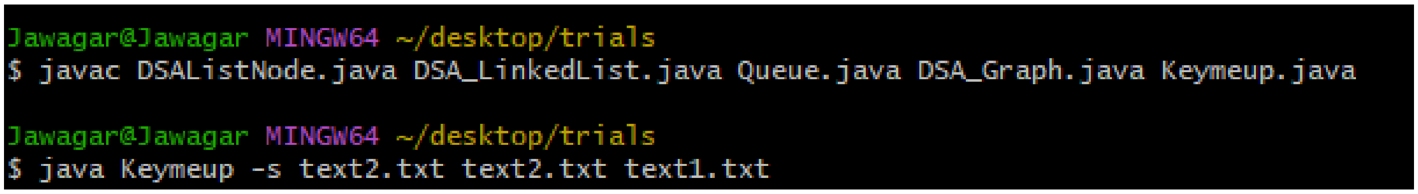
3.Silent mode

in this mode user will have to add “-s” following the name of the file in order to enter into silent mode. In addition, users will have to enter the details of three other files in this order -

1.The keyboard file

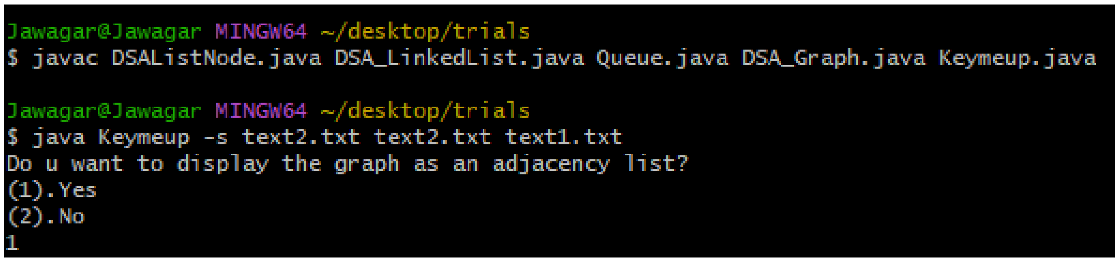
2.the file containing strings to generate path

3.the file to write the paths of a file



```
Jawagar@Jawagar MINGW64 ~/desktop/trials
$ javac DSAListNode.java DSA_LinkedList.java Queue.java DSA_Graph.java Keymeup.java

Jawagar@Jawagar MINGW64 ~/desktop/trials
$ java Keymeup -s text2.txt text2.txt text1.txt
```

This is how the silent mode will be triggered when compiling and running it.
Then a menu is displayed as mentioned below.



```
Jawagar@Jawagar MINGW64 ~/desktop/trials
$ javac DSAListNode.java DSA_LinkedList.java Queue.java DSA_Graph.java Keymeup.java

Jawagar@Jawagar MINGW64 ~/desktop/trials
$ java Keymeup -s text2.txt text2.txt text1.txt
Do u want to display the graph as an adjacency list?
(1).Yes
(2).No
1
```

Then it will be displayed and written onto the file name specified in the command line.

Description of classes and methods used

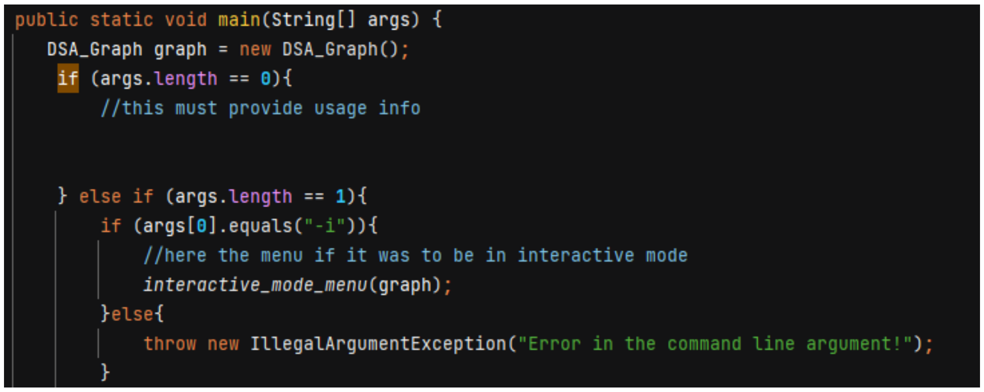
I have used 5 separate files in my assignment program.

1. DSAListNode.java - this class includes all the methods and instance variables needed to create a linked list node.
2. DSA_LinkedList.java – this class includes all the methods and variables needed to make a linked list work by using the DSAListNode class.
3. Queue.java – this class includes the methods and variables needed to implement a queue by using linked list
4. DSA_Graph.java - this class includes the methods and variables needed to implement a graph by using linked list and BFS using a queue.
5. keyMeUp.java – this is the driver code that has the codes for the keyboard program and has the main method.

DSAListNode class is just an independent class with fields such as value, next and prev that will be used to make a linked list node. This is an independent class and is been implemented on a separate file in order to reduce code complexity.
DSA_LinkedList class is just an independent class with fields such as head and tail that are a reference of the list node class. The purpose of this class is to create a proper linked list with the help of the **DSAListNode** class**.** This is an independent class and is been implemented on a separate file in order to reduce code complexity.
Queue class is just an independent class with fields such as DSAQueue that will be used to make a reference to the linked list code in the **DSA_LinkedList class**. This is an independent class and is been implemented on a separate file in order to reduce code complexity.
DSA_Graph class is just an independent class with fields such as vertices that is a reference of linked lists and value that is of object data type. Then this class has 2 other inner classes one for the graph vertex and the graph edge. The graph vertex has class fields namely label, value, links which is a linked list reference variable and visited a Boolean variable. The graph edge class has class fields namely from, to, label, value and directed a Boolean variable. The private inner classes could also have been implemented separately by 2 separate files but since these are only used within the graphs file, they have been made into one file.

keyMeUp – this file is the driver code as it is the file that has the main method and trigger to start the keyboard program. This file can be accessed in 2 modes the interactive mode and the silent mode. If neither of these methods are used then information regarding the methods namely the interactive mode and the silent mode will be displayed. Now lets look into how the 2 methods will work and how the file is developed to run this.
The command line arguments namely the interactive and rhe silent mode are implemented by passing arguments in the command line so the main method that actually takes arguments as

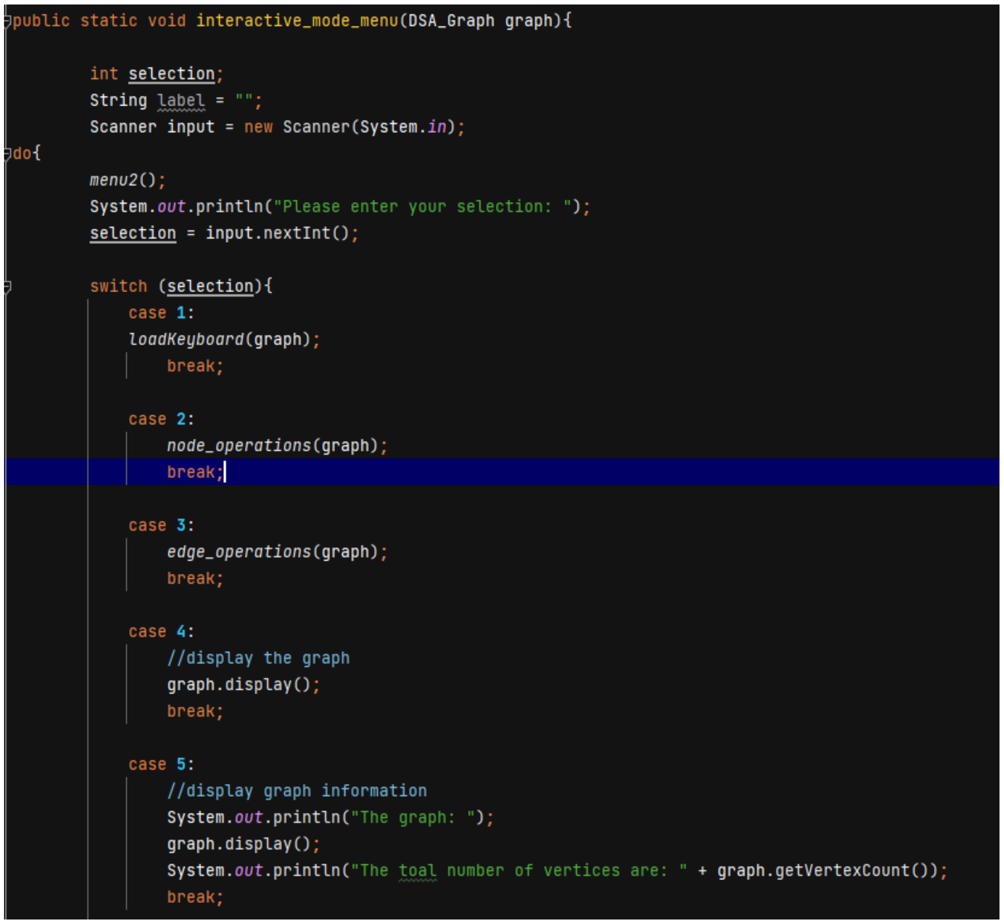
parameters is passed in arguments and then this is used in if statements to differentiate it as interactive menu and code run in silent mode.



```
public static void main(String[] args) {
   DSA_Graph graph = new DSA_Graph();
    if (args.length == 0){
        //this must provide usage info


    } else if (args.length == 1){
        if (args[0].equals("-i")){
            //here the menu if it was to be in interactive mode
            interactive_mode_menu(graph);
        }else{
            throw new IllegalArgumentException("Error in the command line argument!");
        }
```

The above attached picture represents the piece of coding that would make the program an interactive testing environment one.

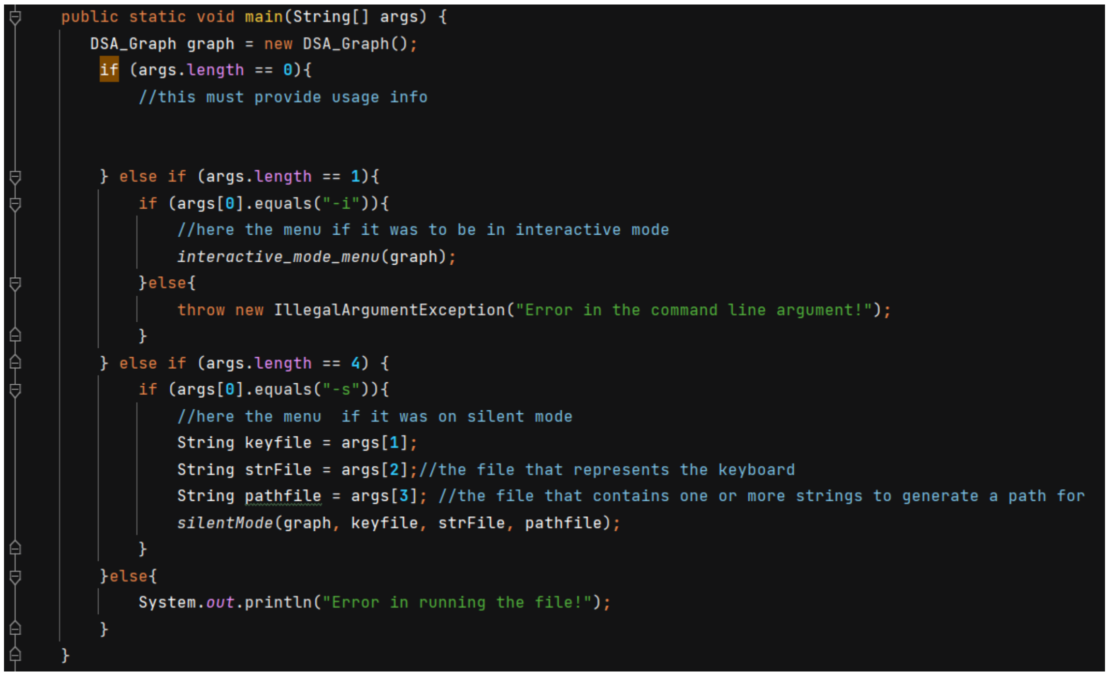
Interactive testing environment coding – an if statement is used inorder to differentiate it from silent mode and no argument mode. The if statement then calls another module the Interactive_mode_menu that has all the switch statements needed to make the interactive menu work. This method also calls a separate method coded to keep the options separately that would make the Interactive_mode_menu less confusing.



```
public static void interactive_mode_menu(DSA_Graph graph){

        int selection;
        String label = "";
        Scanner input = new Scanner(System.in);
do{
        menu2();
        System.out.println("Please enter your selection: ");
        selection = input.nextInt();

        switch (selection){
            case 1:
            loadKeyboard(graph);
                break;

            case 2:
                node_operations(graph);
                break;

            case 3:
                edge_operations(graph);
                break;

            case 4:
                //display the graph
                graph.display();
                break;

            case 5:
                //display graph information
                System.out.println("The graph: ");
                graph.display();
                System.out.println("The toal number of vertices are: " + graph.getVertexCount());
                break;
```

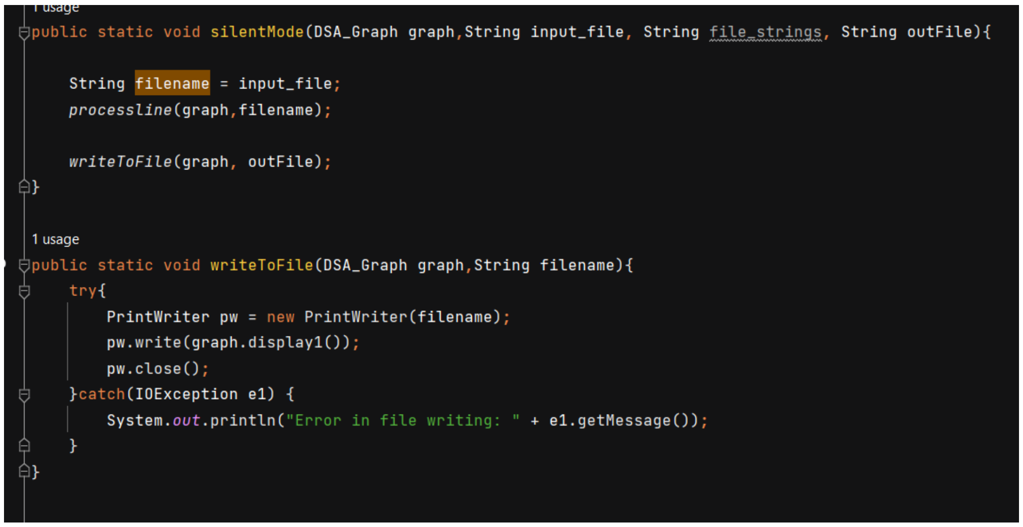
The above attached picture represents a snapshot of the Interactive_mode_menu that has the case statements required to make the interactive mode work.

Silent mode – once again an if statement is used to separate the interactive testing environment and the silent mode coding. The silent mode coding requires a range of information to be passed in the command line namely a symbol - “-s”, name of the keyboard file, then the file with strings from which to find paths from and finally the file to which to write to the paths of the entered strings. So it takes 4 arguments in the command line to successfully get into the silent mode version of the program. Then the details of the files to be specific the names of the files are passed as parameters into another method called silent mode that will carry out the silent mode code implementation. The coding to take input is represented by the picture given below.



```
public static void main(String[] args) {
   DSA_Graph graph = new DSA_Graph();
    if (args.length == 0){
        //this must provide usage info


    } else if (args.length == 1){
        if (args[0].equals("-i")){
            //here the menu if it was to be in interactive mode
            interactive_mode_menu(graph);
        }else{
            throw new IllegalArgumentException("Error in the command line argument!");
        }
    } else if (args.length == 4) {
        if (args[0].equals("-s")){
            //here the menu  if it was on silent mode
            String keyfile = args[1];
            String strFile = args[2];//the file that represents the keyboard
            String pathfile = args[3]; //the file that contains one or more strings to generate a path for
            silentMode(graph, keyfile, strFile, pathfile);
        }
    }else{
        System.out.println("Error in running the file!");
    }
}
```

The silent mode method then calls other file reading and writing methods inorder to carryout the silent mode function. The file is read and added to the edges and then the strings function would take place and then write it to the file.



```
1 usage
public static void silentMode(DSA_Graph graph,String input_file, String file_strings, String outFile){

    String filename = input_file;
    processline(graph,filename);

    writeToFile(graph, outFile);
}

1 usage
public static void writeToFile(DSA_Graph graph,String filename){
    try{
        PrintWriter pw = new PrintWriter(filename);
        pw.write(graph.display1());
        pw.close();
    }catch(IOException e1) {
        System.out.println("Error in file writing: " + e1.getMessage());
    }
}
```

Justification of decisions

I have used a range of abstract data types in my code. I have used linked lists to make my graph code as it is the most suitable ADT to make it. Linked lists allows the graph nodes to be connected through vertices and edges easily as the concept of linked lists is basically a set of nodes with a head and an optional tail that are connected, thereby allowing a graph to be created successfully. Then I have used queues in my breadth first search piece of code which is recommended by lecturers and online sources too.

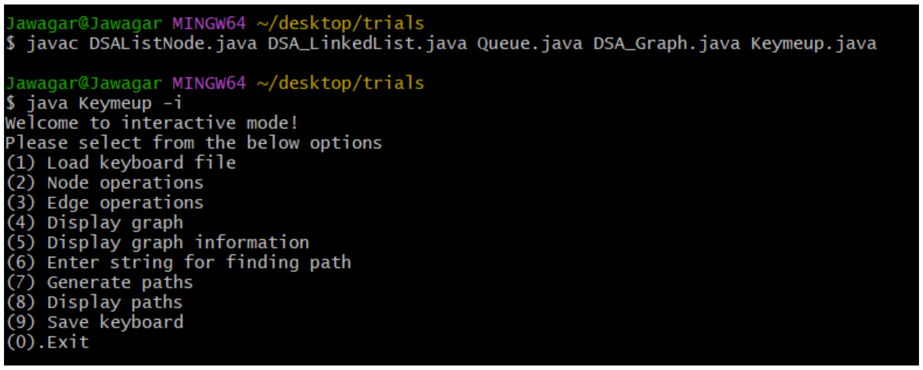
UML Diagrams

Traceability matrix

Showcase

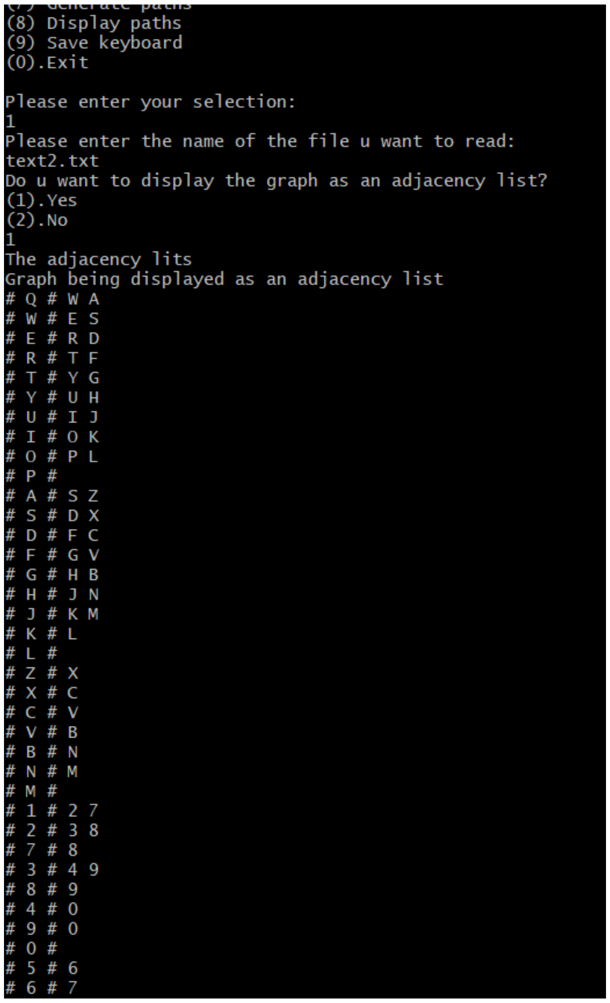
The operations of the program will run as follows:

1. **Load keyboard file –** when the program is run through the interactive testing method then the following menu is presented.



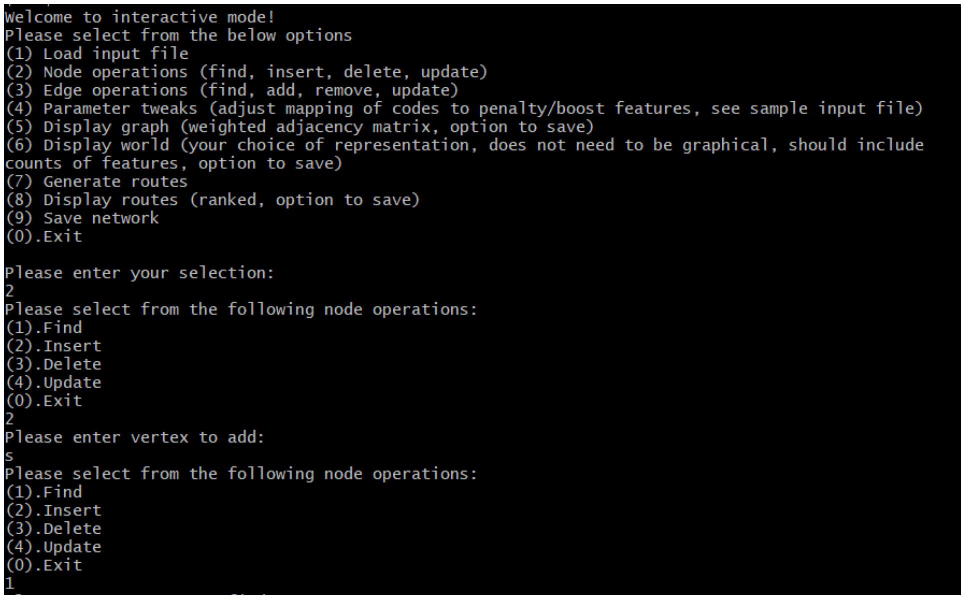
```
Jawagar@Jawagar MINGW64 ~/desktop/trials
$ javac DSAListNode.java DSA_LinkedList.java Queue.java DSA_Graph.java Keymeup.java

Jawagar@Jawagar MINGW64 ~/desktop/trials
$ java Keymeup -i
Welcome to interactive mode!
Please select from the below options
(1) Load keyboard file
(2) Node operations
(3) Edge operations
(4) Display graph
(5) Display graph information
(6) Enter string for finding path
(7) Generate paths
(8) Display paths
(9) Save keyboard
(0).Exit
```

Then when the users select option 1 that is to load the keyboard file then the name of the file to be read is requested and then it will be read and another question will come up on whether to print the graph as a list or not and then the graph will be printed otherwise it will return to the first menu.



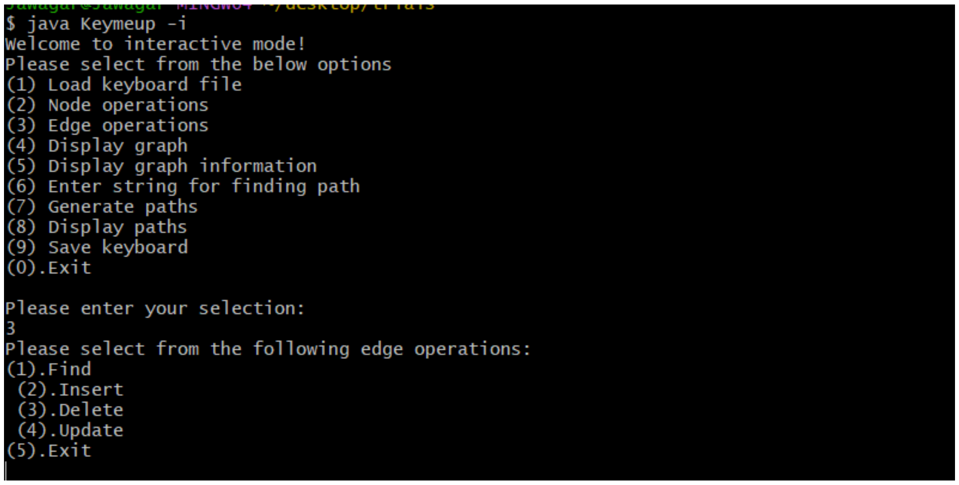
```
(8) Display paths
(9) Save keyboard
(0).Exit

Please enter your selection:
1
Please enter the name of the file u want to read:
text2.txt
Do u want to display the graph as an adjacency list?
(1).Yes
(2).No
1
The adjacency lits
Graph being displayed as an adjacency list
# Q # W A
# W # E S
# E # R D
# R # T F
# T # Y G
# Y # U H
# U # I J
# I # O K
# O # P L
# P #
# A # S Z
# S # D X
# D # F C
# F # G V
# G # H B
# H # J N
# J # K M
# K # L
# L #
# Z # X
# X # C
# C # V
# V # B
# B # N
# N # M
# M #
# 1 # 2 7
# 2 # 3 8
# 7 # 8
# 3 # 4 9
# 8 # 9
# 4 # 0
# 9 # 0
# 0 #
# 5 # 6
# 6 # 7
```

2. **Node operations** - this will ask the user on what types of node operations they would wish to continue with



```
Welcome to interactive mode!
Please select from the below options
(1) Load input file
(2) Node operations (find, insert, delete, update)
(3) Edge operations (find, add, remove, update)
(4) Parameter tweaks (adjust mapping of codes to penalty/boost features, see sample input file)
(5) Display graph (weighted adjacency matrix, option to save)
(6) Display world (your choice of representation, does not need to be graphical, should include
counts of features, option to save)
(7) Generate routes
(8) Display routes (ranked, option to save)
(9) Save network
(0).Exit

Please enter your selection:
2
Please select from the following node operations:
(1).Find
(2).Insert
(3).Delete
(4).Update
(0).Exit
2
Please enter vertex to add:
s
Please select from the following node operations:
(1).Find
(2).Insert
(3).Delete
(4).Update
(0).Exit
1
```

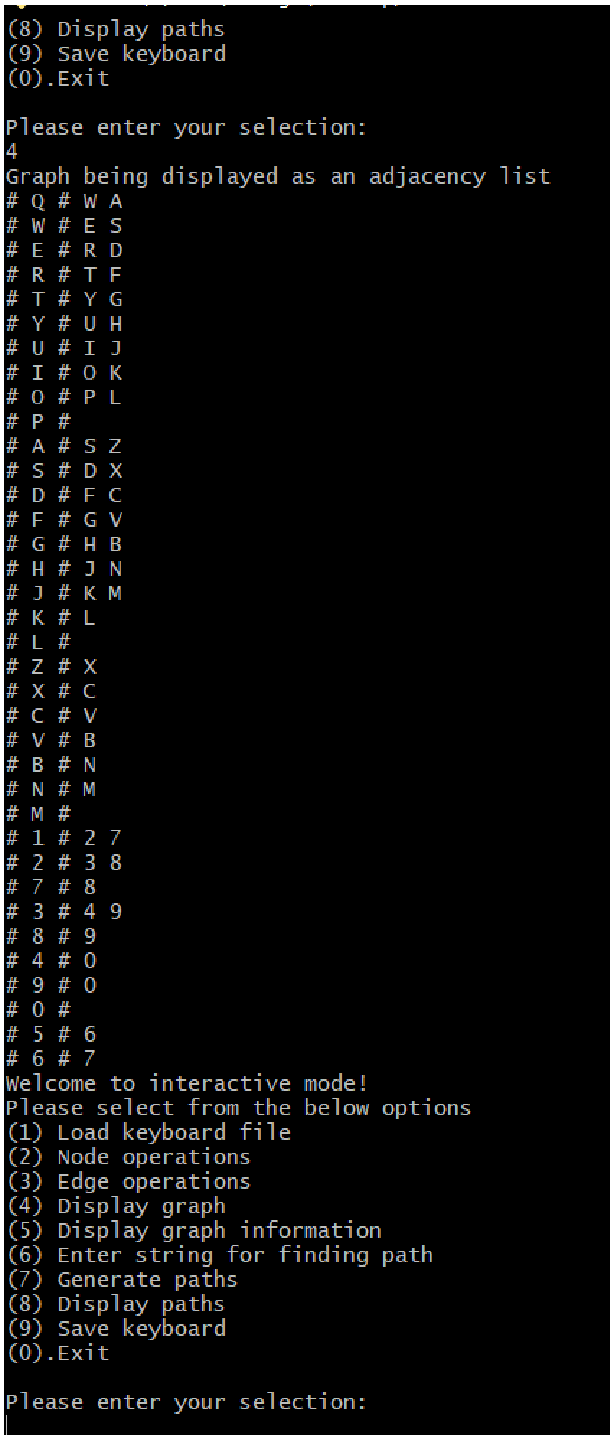
3. **Edge operations -** this will ask the user on what types of node operations they would wish to



```
$ java Keymeup -i
Welcome to interactive mode!
Please select from the below options
(1) Load keyboard file
(2) Node operations
(3) Edge operations
(4) Display graph
(5) Display graph information
(6) Enter string for finding path
(7) Generate paths
(8) Display paths
(9) Save keyboard
(0).Exit

Please enter your selection:
3
Please select from the following edge operations:
(1).Find
 (2).Insert
 (3).Delete
 (4).Update
(5).Exit
```

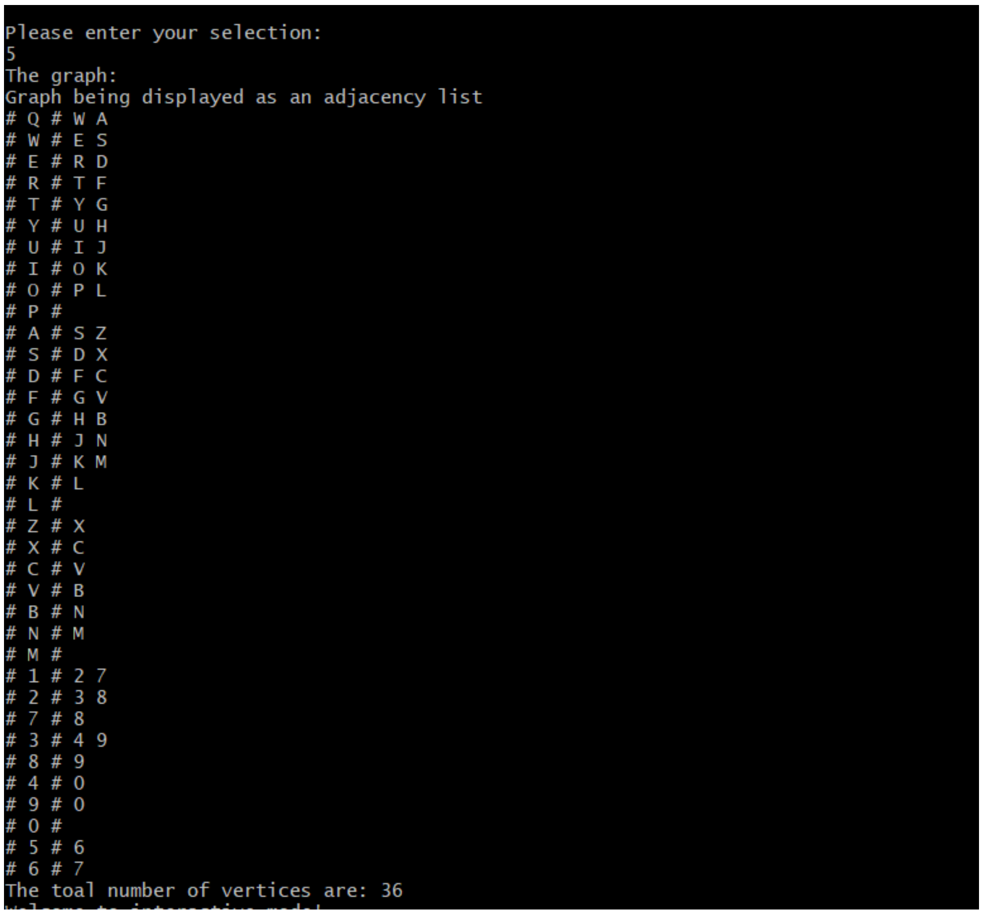
continue with

4. **Display graph –** this code displays the graph as an adjacency list



```
(8) Display paths
(9) Save keyboard
(0).Exit

Please enter your selection:
4
Graph being displayed as an adjacency list
# Q # W A
# W # E S
# E # R D
# R # T F
# T # Y G
# Y # U H
# U # I J
# I # O K
# O # P L
# P #
# A # S Z
# S # D X
# D # F C
# F # G V
# G # H B
# H # J N
# J # K M
# K # L
# L #
# Z # X
# X # C
# C # V
# V # B
# B # N
# N # M
# M #
# 1 # 2 7
# 2 # 3 8
# 7 # 8
# 3 # 4 9
# 8 # 9
# 4 # 0
# 9 # 0
# 0 #
# 5 # 6
# 6 # 7
Welcome to interactive mode!
Please select from the below options
(1) Load keyboard file
(2) Node operations
(3) Edge operations
(4) Display graph
(5) Display graph information
(6) Enter string for finding path
(7) Generate paths
(8) Display paths
(9) Save keyboard
(0).Exit

Please enter your selection:
```

5.Display graph information – this displays the vertices count and the adjacency list of the graph.



```
Please enter your selection:
5
The graph:
Graph being displayed as an adjacency list
# Q # W A
# W # E S
# E # R D
# R # T F
# T # Y G
# Y # U H
# U # I J
# I # O K
# O # P L
# P #
# A # S Z
# S # D X
# D # F C
# F # G V
# G # H B
# H # J N
# J # K M
# K # L
# L #
# Z # X
# X # C
# C # V
# V # B
# B # N
# N # M
# M #
# 1 # 2 7
# 2 # 3 8
# 7 # 8
# 3 # 4 9
# 8 # 9
# 4 # 0
# 9 # 0
# 0 #
# 5 # 6
# 6 # 7
The toal number of vertices are: 36
```

Scenarios

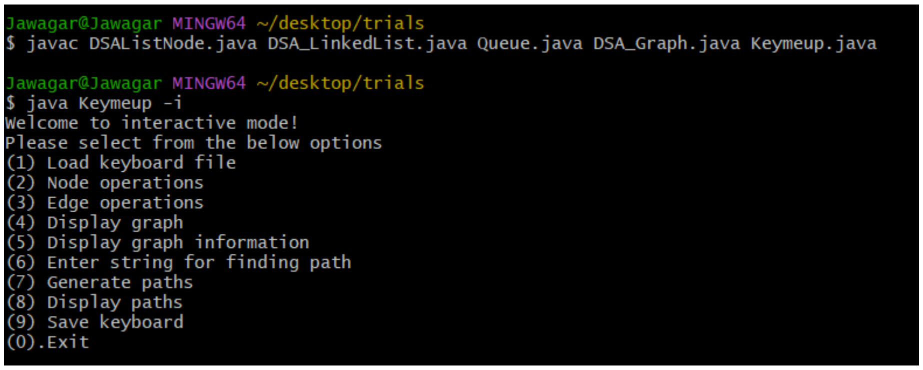
Scenario – 1

User now wants to load a keyboard file and then display the graph as an adjacency list.

User can commence the program either by using the interactive mode or through silent mode

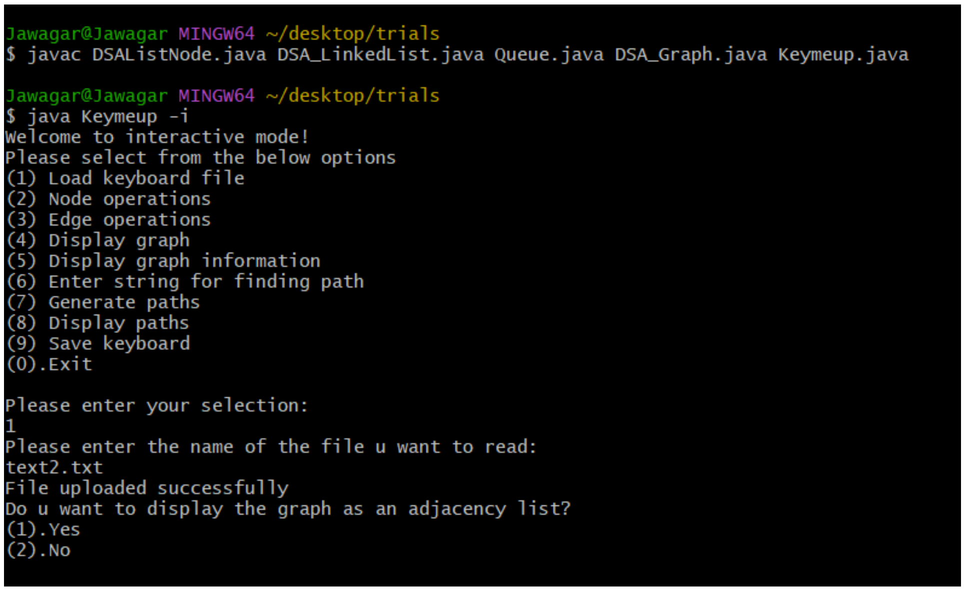
- ✓ Interactive testing mode – keyMeUp -i
- ✓ Silent mode - keyMeUp -s text2.txt string.txt out.txt

Then in the interactive mode a menu would appear as follows:

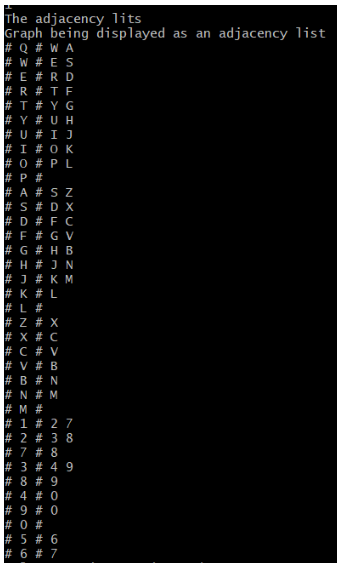


```
Jawagar@Jawagar MINGW64 ~/desktop/trials
$ javac DSAListNode.java DSA_LinkedList.java Queue.java DSA_Graph.java Keymeup.java

Jawagar@Jawagar MINGW64 ~/desktop/trials
$ java Keymeup -i
Welcome to interactive mode!
Please select from the below options
(1) Load keyboard file
(2) Node operations
(3) Edge operations
(4) Display graph
(5) Display graph information
(6) Enter string for finding path
(7) Generate paths
(8) Display paths
(9) Save keyboard
(0).Exit
```

Then the user would select option 1 as they should load the keyboard file. then they would be asked for the name of the file to read data from. User would enter that detail and then the file would be read. After that user would be asked if they want to display it as a list or return to main menu. If they chose the first option they can display the graph.



```
Jawagar@Jawagar MINGW64 ~/desktop/trials
$ javac DSAListNode.java DSA_LinkedList.java Queue.java DSA_Graph.java Keymeup.java

Jawagar@Jawagar MINGW64 ~/desktop/trials
$ java Keymeup -i
Welcome to interactive mode!
Please select from the below options
(1) Load keyboard file
(2) Node operations
(3) Edge operations
(4) Display graph
(5) Display graph information
(6) Enter string for finding path
(7) Generate paths
(8) Display paths
(9) Save keyboard
(0).Exit

Please enter your selection:
1
Please enter the name of the file u want to read:
text2.txt
File uploaded successfully
Do u want to display the graph as an adjacency list?
(1).Yes
(2).No
```



```
The adjacency lits
Graph being displayed as an adjacency list
# Q # W A
# W # E S
# E # R D
# R # T F
# T # Y G
# Y # U H
# U # I J
# I # O K
# O # P L
# P #
# A # S Z
# S # D X
# D # F C
# F # G V
# G # H B
# H # J N
# J # K M
# K # L
# L #
# Z # X
# X # C
# C # V
# V # B
# B # N
# N # M
# M #
# 1 # 2 7
# 2 # 3 8
# 7 # 8
# 3 # 4 9
# 8 # 9
# 4 # 0
# 9 # 0
# 0 #
# 5 # 6
# 6 # 7
```

Conclusion

This assignment was a challenging one unlike the other assignments. Overall I have implemented all the functions and operations with my knowledge and I have tried my level best in completing the assignment, but I have not been able to as I am not able to figure out certain operations or activities.