CS211: Data Structures and Algorithm Analysis

Assignment 1 - Word Ladder Generator

Connor Luke Goddard

Student No: 110024253

Aber ID: clg11

Introduction

For this assignment I have been tasked with designing a Java application that can generate, and analyse word ladders. A word ladder is a list of words that differ from each other by having a one letter difference. The application must include a Graph data structure capable of holding a list of words read in from a file, along with data that represents a 'link' between words that have a one-letter difference.

This data structure must then be analysed by appropriate algorithms to generate word ladders when a start word is inputted, and also to find the shortest path between two words, before outputting this path as a word ladder. I must design my application using tools such as UML, and thoroughly test my completed system using both automated 'JUnit' tests, and test tables to ensure the system is robust and full fills its requirements.

The purpose of this document is to describe and justify my design for the application, including my choice of data structure and algorithms, and also to display evidence of my testing procedures for the system. This document also includes an appendix which contains the full Java source code for the entire application.

Notes

- Please be aware that the justification for my chosen designs has been incorporated into the class /design approach descriptions.

Design of the System

Before I can produce designs for my application, I have had to research methods of implementing the Graph data structure and the possible algorithms that could be used to traverse the graph, and algorithms that could locate the shortest path between two nodes within my graph.

Data Structure

A graph data structure consists of a series of elements (also known as Nodes/Vertices) that have links (otherwise known as Edges/Arcs) to other nodes within the same structure. A single edge can only be made between two nodes, however multiple pairs of nodes can be linked together (to form a graph).

In this application, I have identified that links between nodes will depend on a common criteria, this criteria being that two words (represented as a Node in the graph) differ by one letter. This common criterion for creating edges also defines that the graph structure in my application will be un-weighted, meaning that all edges between nodes will represent an equal value. This fact is crucial, when identifying a suitable algorithm for finding the shortest path between two nodes within the graph.

After performing research into methods of representing graph data structures within programs **(Sheard, 2009),** I have decided to implement my graph using an **Adjacency List**. An adjacency list is a data structure that stores a list of all nodes within a graph, and for every node, contains another list of all the nodes it currently has an edge to (neighbour) **(Wakeman, 2011)**.

I have chosen to use an adjacency list due to its simplicity to implement, and its natural ability to dynamically expand as new nodes are added and edges are created between them. I feel this ability is important to the application, as thousands of nodes are going to be contained within the graph (as words are read in from a file) and as a result many edges will be created. It is therefore vital that the graph structure can cope with this large amount of data.

Adjacency lists are commonly implemented using a **Hash table**. These data structures are a viable solution to creating an adjacency list as they contain a '*Key*' field, which is used to represent a particular node within the graph, and a '*Value*' field which is used to store a list of all other nodes that it is linked to.

I have decided to use a Hash table to implement my adjacency list graph representation because they are extremely easy to implement as the Java JDK library already contains a 'Hash table' object [3] (Oracle Inc.) which can be used "straight out of the box", they are extremely fast when accessing data and this speed is more apparent when the size of the Hash table is relatively large(as this one will be), and it is also supports dynamic allocation, which again is going to be vital when the application is automatically "reading-in" thousands of words in from a file.

Algorithms

Generation:

For generation, the application is required to traverse the graph over a selected number of linked nodes from a given 'start word'. I therefore identified that this task would require an algorithm that was quick, efficient and was always able to produce a solution (as long as there was at least one node that contained an edge). With this in mind, I have concluded that a **Depth Limited Search (DLS)** algorithm would be most suitable.

DLS is an extension of the widely used **Depth First Search** algorithm, and as a result contains the useful characteristics associated with DFS which makes it a suitable choice for this task:

- The nature of DFS is to travel as "deep" as it can down the particular path it is currently searching, expanding the nodes it meets while it travels. This behaviour is suitable for generation, as the primary objective is to traverse the graph as far as possible from the start node within the limit set by the user.

- The behaviour of DLS is exactly that of DFS, with the exception being that DLS sets a limit to how "deep" DFS searches to before returning a solution (if found). This therefore will enable the user to set how long the word ladder should be, by forcing the algorithm to terminate after it reaches a certain depth.

- The time complexity of DFS varies depending on the type of data it is using and the size of the graph it is searching. However with this particular task, DFS is significantly more likely to produce a solution faster than a Breadth First Search for example, because of its natural behaviour to traverse deep into a graph as far as it can possibly go before backtracking.

- The space complexity required by DFS is significantly smaller than similar algorithms such as Breadth First Search (BFS). The reason for this is that unlike BFS, as it only searches one path/branch at a time, DFS does not have to store a list of all the child-node pointers it encounters, and so as a result requires less storage space in memory.

Pseudo-code of DLS Algorithm:

```
DLS(searchTerm, currentDepth, depthLimit) {
Stack s = []; //Start with an empty stack
      if (currentDepth < depthLimit) {
            for (every node n unvisited) {
                  set n = visited;
                  for (every neighbour u of n unvisited) {
                        set u = visited;
                  DLS(u, currentDepth + 1, depthLimit);
                  if (search has traversed as far as it can go) {
                        s.push(u);
```

```
                        return; //Returns up to the previous recursion of DFS to
push that node element to stack

                        }}}}}
```

(Russell & Norvig, 2003, Second Edition)

Discovery:

The discovery task requires a slightly different approach to generation because instead of simply traversing the graph to a specified depth, the application is asked to return the shortest word ladder between two words inputted by the user.

This task can be broken down to a "find the shortest path" problem, and as a result requires an algorithm that can traverse and search the entire graph, to allow it to check all possible paths from the start word, to the target word, and so return the shortest possible path between them.

After performing some initial research, I discovered that when using weighted graphs Dijkstra's algorithm would be the most optimal solution for finding the shortest path. However, with this particular problem as every word is differentiated by the same criteria (one letter difference), this would result in the weighting between every node in the graph being equal and so as a result Dijkstra's algorithm would not be able to differentiate between edges to find the shortest path.

After further research, I concluded that a **Breadth First Search (BFS)** (a primary component of Dijkstra's algorthim) would be the most viable algorithm for discovery.

BFS works in a similar fashion to DFS, with the difference being that instead of traversing a single path as deep as it possibly can before backtracking, BFS traverses across all the nodes at a given depth before incrementing deeper and repeating the process.

The behaviour and time/space complexities of BFS make it suitable for finding the shortest path between the two specified nodes:

- The natural behaviour of BFS ensures the algorithm gathers information about the graph in its entirety, which is vital when comparing the different paths between nodes to return the shortest path. DFS/DLS for example would not be suitable as it only searches a select area of the graph at any one time, and does not keep a record of the paths it follows.

- The time complexity of BFS is the same as that of DFS. The reason for this is that they both share the same behaviour of traversing across all the nodes in a graph. However due to the number of words being read in (~3000-8000), the average size of the graph will be relatively small and so realistically the time taken should be minimal.

- The space complexity of BFS is larger than similar algorithms such as DFS, as it has to store a queue of all the nodes in the graph. However, this difference in space complexity is fundamental in allowing BFS to return the shortest path, and with the size of the graph being relatively small, the space complexity would have a minimal impact on the memory space in typical host computer.

Pseudo-code of BFS Algorithm:

```
BFS(startWord, targetWord) {
	Queue q = [];
	Stack s = [];
	q.add(startWord);
	set startWord = visited;
	while(q is not empty) {
		Node n = q.remove();
		for (every neighbour u of n unvisited) {
			set u = visited;
			set u.predecessor = n;
			q.add(u);
		}
	}
	Node c = targetWord;
	while(c is not startWord) {
		if {c does not contain a predecessor) {
			return;
		}
		set c = c.predeccessor;
		s.push(c);
	}
}
```

(Russell & Norvig, Artificial Intelligence, A Modern Approach, 2003, Second Edition)

System Class Diagram

I am now able to use my conclusions from the research I conducted on graph data structures and search algorithms to produce a UML class diagram of my proposed application solution. Figure 1.3 describes the proposed structure of my word ladder application including the data model, and GUI classes.

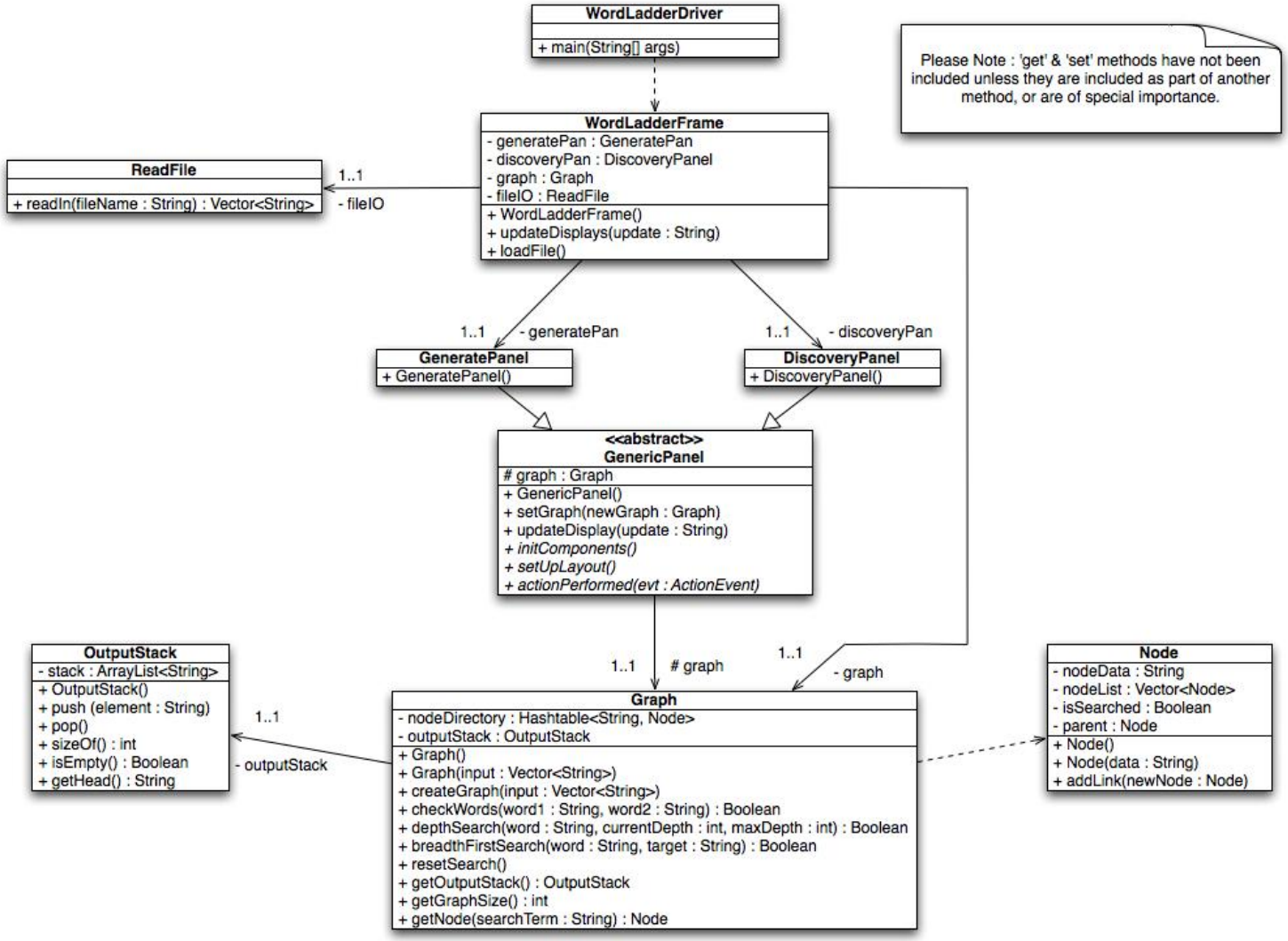


Figure 1.1 – Application Class Diagram

Design Description and Justification – Data Model

Graph

The Graph class is fundamental to the graph data structure within the application, as it contains all the objects and methods that are used in the implementation, manipulation and analysis of the graph.

The hash table used for the adjacency list is contained within the class along with the methods to add new elements ('createGraph()') and retrieve elements ('getNode()') to/from the hash table. The graph also contains both the implementations of the BFS, and DLS algorithms as these are specifically designed to be used with a graph data structure, and so in the interests of encapsulation I felt it was best that these methods remain contained within their working environment.

I have also decided to contain the method for comparing the letter difference in words ('checkWords()') within the Graph class. I chose to do this because this method will always be called when a new node is being added to the graph (as the sole purpose of the method is to determine if an edge should be created between two nodes) and it is not required by any other part of the application, so by keeping the method within the Graph class I maintain high cohesion, by keeping all similar methods/variables together, and low coupling by preventing the class having to create a reference to a different class that contains the 'checkWords()' method.

The class also contains a reference to the OutputStack class. This is to allow the two search algorithms ('breadthFirstSearch()' and 'depthSearch()') to push the results of their search (i.e. the node elements of a word ladder) to a stack that can then be passed to the GUI classes to be displayed to the user.

I believe that I have designed a class that allows a complete, and easy to configure implementation of a graph data structure. I feel that it conforms to good programming concepts such as "high cohesion and low coupling", and could easily be transferred and integrated into future projects for re-use. Given more time however, I would have attempted to use generics within the class to allow it to become completely versatile and re-usable.

Node

This is a light-weight class that is used to represent a particular word within the graph data structure. This class contains four variables which are utilised by different processes within the application.

'nodeData' contains the identity of the word that is being represented by that Node object. '*nodeList*' is a Vector that contains a list of all neighbours of that Node, and so is used as part of the adjacency list implementation. '*isSearched()*' is used by both search algorithms in the Graph class as a mechanism to prevent nodes that have been checked once in a search, being checked again before the search terminates. This is important as it prevents word ladders containing the same word twice, which is a requirement of the assignment brief. Finally the '*parent*' variable is used by the BFS as a means of outputting the word ladder once it has finished searching. It uses this variable to traverse along the shortest path once it has been identified to print out the resulting word ladder.

OutputStack

This implementation of Stack data structure is used by the Graph class to output the result of searches performed by the two search algorithms.

The stack is important to the generation and discovery of word ladders because of its "LIFO" structure. Once the two algorithms have completed their search, to return each node in the result 'path' they have to traverse from the last element in the path (i.e. the final iteration node in DLS and target node in BFS) up to the root/starting node. As they do this, they push the current node to the stack. Therefore, once all the nodes in the path have been pushed onto the stack and the application wishes to display this word ladder, the last element in the stack (i.e. the start/root node) is displayed first, and then the second is displayed and so forth. This behaviour could not be achieved by any other data structure that was as quick and easy to implement as a stack.

I decided to implement this stack in its own class as it enforces re-usability and encapsulation, and would allow this class to be easily integrated into future applications with ease.

ReadFile

Responsible for accessing and parsing an external dictionary file (i.e. 'dat', 'txt') before returning the contents which is then used to generate a graph. This light-weight class is referenced within the GUI, and the '*readIn()*' method is called when the application starts to extract the data that is then used to populate the graph data structure.

I decided to create a separate class for file parsing as it is a common and popular process that is used in a variety of different applications and software solutions. Therefore by creating a separate class, it ensures this method can be re-used however many times as is required by the application, and allows easy integration into future projects.

Design Description and Justification – GUI

WordLadderDriver

Bootstrap for the application. Contains the 'main' method that creates an instance of the main GUI frame, WordLadderFrame.

WordLadderFrame

Main GUI frame in which the panels that contain the interactive/display elements are created. These panels are added to the NORTH and SOUTH sections of the frame.

The Graph data class is initialized here, and is created using words extracted from an external file via the ReadFile class. This Graph object is passed as a parameter to both the GeneratePanel, and the DiscoveryPanel to allow them to access the data structure and hence provide the user with a means of utilising and manipulating the data.

GenericPanel

Abstract class extended by the GeneratePanel and DiscoveryPanel classes used to contain shared methods and variables between the two panels. As both sub-panels share all the same methods and variables I felt an abstract class would be the most efficient, and OO approach to preventing repetition of code.

As a result of this, if any future GUI development on this application was to take place, it would be a trivial task to implement a new panel and simply extend the abstract class from it, giving it access to all the shared methods that are available to the current two panels.

Another advantage of using an abstract class is that it ensures that the number of references to the Graph class is kept to a minimum, which is crucial when preventing un-authorised methods and processes from accessing the data model.

GeneratePanel

One of two GUI panels displayed within WordLadderFrame, designed to allow a user to generate word ladders of a certain length. This class extends from the abstract GenericPanel class, and as a result has access to the Graph object generated by WordLadderFrame at runtime.

This panel contains two textboxes, a button and a scrollable output window. One textbox is to allow a user to specify a start word from which the system will then attempt to generate a word ladder from, and the second is to allow the user to specify the length of that word ladder. The button triggers an ActionListener contained within the panel that calls the '*depthSearch()*' method within the Graph class. The result of the depth limited search is then displayed in the output window for the user.

I decided to implement the button listener within the panel as I felt it I would be over-complicating the system design by creating a new class for a listener that only handles two button calls. I felt it would be a more object-oriented and modular approach to specify the listener within the abstract class and then share this between the two sub-panels.

DiscoveryPanel

Second of two GUI panels displayed within WordLadderFrame, designed to allow a user to enter two words with the application then returning the shortest path between those two words. This class extends from the abstract GenericPanel class, and as a result has access to the Graph object generated by WordLadderFrame at runtime.

This panel also contains two textboxes, a button and a scrollable output window. One textbox is to allow a user to specify a start word from which the system will then attempt to generate a word ladder from, and the second is to specify a target word which the system will search for. The button triggers an ActionListener contained within the panel that calls the '*breadthFirstSearch()*' method within the Graph class. The search algorithm will then attempt to find the shortest path between the two specified words before displaying the word ladder in the output window.

System Testing

To ensure that the software I have designed and implemented is functioning as expected, I have performed both data model and GUI testing designed to test the software in variety of different situations to help highlight any errors in the code that can then be rectified accordingly.

Data Model & Algorithms

To test the data model and graph search algorithms in my application, I designed a selection of automated tests using the 'JUnit' testing framework. This has allowed me to design bespoke tests that execute the methods inside the data classes using specified test data, before checking the results of those methods against criteria set by myself to test that they working as expected. The source code for these tests is available in Appendix 1.2.

In addition to the JUnit tests, I have devised a testing table (Figure 1.4) which details the structure and plan of additional tests designed to ensure that both modes of operation (generation and discovery) work as expected.

Test ID	Description	Test Input	Expected Outcome	Actual Outcome	P/F	Comments
1	Generation (4 letter words) – Run using valid start word	"coil", 5	[coil, boil, bail,bait, bast]	[Coil, boil, bail,bait, bast, bass]	F	Generated word ladder one node too long - Rectified
2	Generation (4 letter words) – Run using invalid start word	"qwer", 5	No word ladder generated	No word ladder was generated	P	
3	Generation (4 letter words) – Valid word, maximum ladder length possible < specified length	"bank", 30	Largest possible word ladder generated (6 words) – User informed that ladder is not as large as specified	Word ladder of maximum length (6 words) generated – GUI informs user of this	P	
4	Generation (5 letter words) – Valid word	"print" , 7	[print, paint, faint, fains, fails, bails, baits]	[print, paint, faint, fains, fails, bails, baits]	P	
5	Generation (5 letter words) – Invalid word	"hytgy", 2	No word ladder generated	No word ladder was generated	P	
6	Generation (5 letter words) – Invalid word (6 letters)	"abacus", 4	No word ladder will be created as start word has too many letters	No word ladder was created as start word was too long	P	

7	Generation (6 letter words) – Valid word	“cooker”, 3	[cooker, cooked, choked]	[cooker, cooked, choked]	P	
8	Generation (6 letter words) – Invalid word	“zpoj4h”, 10	No word ladder generated	No word ladder was generated	P	
9	Generation (6 letter words) – Valid word that contains no edges to other nodes	“school”, 4	Word recognised within graph, however no word ladder generated	Performed as expected however system still displayed empty word ladder	F	Re-positioned method that displays stack contents to window - Rectified
10	Discovery (4 letter words) – Valid start word, valid end word	“head”, “foot”	Shortest word ladder generated. Ladder length: 6 words.	[head, bead, beat, boat, boot, foot]	P	
11	Discovery (4 letter words) – Valid start word, valid end word (SWAPPED)	“foot”, “head”	Identical word ladder generated as Test 9, arranged in opposite order	[foot, boot, boat, beat, bead, head]	P	
12	Discovery (4 letter words) – Valid start word, invalid end word	“coil”, “pytf”	Word ladder cannot be generated	Word ladder was not generated	P	
13	Discovery (4 letter words) – Invalid start word, valid end word	“gqwd”, “bank”	Word ladder cannot be generated	Word ladder was not generated	P	
14	Discovery (5 letter words) – Valid start word, valid end word	“print”, “break”	Shortest word ladder generated. Ladder length: 12 words.	[print, paint, taint, taunt, gaunt, grunt, brunt, blunt, blent, bleat, bleak, break]	P	
15	Discovery (5 letter words) – Valid start word, valid end word (SWAPPED)	“break”, “print”	Identical word ladder generated as Test 13, arranged in opposite order	[break, bleak, bleat, blent, blunt, brunt, brunt, grunt, gaint, taunt, taint, paint, print]	P	
16	Discovery (5 letter words) – Valid start word, valid end word. No possible path available	“light”, “zebra”	No word ladder can be created as no complete path exists. GUI informs the user	No word ladder generated. Message appears in output window informing user that	P	

	between words.		of this fact.	a shortest path is not possible.		
17	Discovery (6 letter words) – Valid start word, valid end word	“cooked”, “lively”	Shortest word ladder generated. Ladder length: 14 words.	[cooked, booked, booted, bolted, belted, belied, belies, bevies, levies, levees, levers, livers, livery, lively]	P	
18	Discovery (6 letter words) – Valid start word, valid end word (SWAPPED)	“lively”, “cooked”	Identical word ladder generated as Test 16, arranged in opposite order	[lively, livery, livers, levers, levee, levies, bevies, belies, belied, belted, bolted, booted, booked, cooked]	P	
19	Discovery (6 letter words) – Valid start word, valid end word. No possible path available between words.	“school”, “cooked”	No word ladder can be created as no complete path exists. GUI informs the user of this fact.	No word ladder generated. Message appears in output window informing user that a shortest path is not possible.	P	
20	Discovery (6 letter words) – Valid start word, In valid end word (7 letter)	“cooked”, “singles”	No word ladder will be created as target word has too many letters	No word ladder was created as words target word was too long	P	
21	Discovery & Generation – All inputs valid	G = “light”, 4 D = “jolly”, “split”	Both modes generate word ladders successfully	Word ladders generates successfully. Shortest path created in 16 steps.	P	
22	Discovery & Generation – Generation inputs valid, Discovery input in valid	G = “back”, 5 D = “boot”, “swed”	Generate creates word ladder successfully. Discovery unable to create word ladder.	Generate creates word ladder successfully. Discovery unable to create word ladder.	P	
23	Discovery & Generation – Generation inputs in valid, Discovery input valid	G = “vbh8tg”, 2 D = “socket”, “groins”	Discovery creates word ladder successfully. Generate unable to create word ladder.	Discovery creates shortest path word ladder in 15 steps. Generate unable to create word ladder.	P	

Graphical User Interface

I have created a series of tests for the application GUI to ensure that it successfully performs user validation checks designed to prevent a user entering erroneous or extreme values into the data model and search algorithms. To do this I have produced another test table detailing the structure, and results of these tests.

Test ID	Description	Test Input	Expected Outcome	P/F	Comments
1	Application 'main' class executed	N/A	Application GUI loads and manual load dialog appears.	P	
2	Dialog appears asking user if they wish to manual open dictionary file	User clicks "Yes"	'File load' dialog shows on screen	P	
		User clicks "No"	Dialog appears asking user to select		
		User clicks "Cancel"	Application closes		
3	Verify external file can be loaded into graph	User selects external file using 'File load' dialog and clicks "Open"	User can only select 'DAT' or 'TXT' files due to file filter	P	Users can select any 'DAT' or 'TXT' file to generate a graph from
4	Verify pre-defined, internal dictionary files can be loaded into graph	User selects 4 letter word length	Application loads 4 letter word file into Graph	P	
		User selects 5 letter word length	Application loads 5 letter word file into Graph		
		User selects 6 letter word length	Application loads 5 letter word file into Graph		
5	Cancel generating graph	User cancels loading external file	Application closes as no graph has yet been created.	P	
		User cancels selecting pre-defined graph	Application asks if the user would like to load from external file	F	Cancellation triggered NullPointerException - **Rectified**
6	Generation – Valid data in both textboxes	User enters a valid string into 'word' textbox, and valid integer in 'length' textbox	System attempts to generate word ladder using data specified in both textboxes	P	

7	Generation – Required data missing from textboxes	'Start word' textbox empty 'Length' textbox empty Both textboxes empty	System detects one or more textboxes are empty and displays dialog informing user to enter all required data.		
8	Generation – Check format of length value	User enters non-integer value in 'length' textbox	System detects 'length' textbox contains non-integer value and displays message box asking user to enter integer value	P	
9	Discovery – Valid data start and target words entered	User enters a valid string into 'start' textbox, and valid string into 'target' textbox	System attempts to generate shortest path word ladder using entered values	P	
10	Discovery – Required data missing from textboxes	'Start word' textbox empty 'Target word' textbox empty Both textboxes empty	System detects one or more textboxes are empty and displays dialog informing user to enter all required data.	P	

Appendix

- Appendix 1: Java Application Source Code
- Appendix 2 : 'JUnit' Test Source Code
- Appendix 3: Testing Table Screenshots

References

1. Oracle Inc. (n.d.). *Hashtable (Java Platform SE 6)*. Retrieved from Java Platform 6 API: http://docs.oracle.com/javase/6/docs/api/java/util/Hastable.html
2. Russell, S., & Norvig, P. (2003, Second Edition). Artificial Intelligence, A Modern Approach. In *Uninformed Search Strategies, Breadth-first Search* (pp. 73-74). Prentice Hall.
3. Russell, S., & Norvig, P. (2003, Second Edition). Artificial Intelligence, A Modern Approach. In *Uninformed Search Strategies, Depth-Limited Search* (pp. 77, 75-76). Prentice Hall.

4. Sheard, T. (2009). *Graphs in Computer Science*. Retrieved from web.cecs.pdx.edu: web.cecs.pdx.edu/~sheard/course/Cs163/index.html

5. Wakeman, I. (2011). *Implementing Graphs II.* Retrieved from informatics.sussex.ac.uk: http://www.informatics.sussex.ac.uk/courses/FP/graphs/implementingGraphsII4.pdf

Appendix 1.1

WordLadderDriver

```
import aber.dcs.clg11.wordladder.gui.WordLadderFrame;

/**
 * Bootstrap class - Initialises the application
 *
 * @author Connor Goddard (clg11)
 */
public class WordLadderDriver {

	/**
	 * The main method used to initialise the main application.
	 *
	 * @param args The arguments passed on initialisation.
	 */
	public static void main(String[] args) {

		new WordLadderFrame();

	}

}
```

WordLadderFrame

```
package aber.dcs.clg11.wordladder.gui;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Toolkit;

import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.filechooser.FileFilter;
import javax.swing.filechooser.FileNameExtensionFilter;


import aber.dcs.clg11.wordladder.fileIO.ReadFile;
import aber.dcs.clg11.wordladder.model.Graph;

/**
 * Main JFrame for displaying program GUI. Responsible for initialising the
 * main data structure class {@link aber.dcs.clg11.wordladder.model.Graph}, as
well as
 * the two GUI sub-panels {{@link aber.dcs.clg11.wordladder.gui.GeneratePanel} &
 * {@link aber.dcs.clg11.wordladder.gui.DiscoveryPanel}. Passes the new Graph
object to the base panels
```

```
 * as a parameter to allow access to the data model from the sub-panels.
 */
@SuppressWarnings("serial")
public class WordLadderFrame extends JFrame {

	/** The new GeneratePanel component. */
	private GeneratePanel generatePan;

	/** The new DiscoveryPanel component. */
	private DiscoveryPanel discoveryPan;

	/** FileChooser dialog object used to select external dictionary files. */
	private JFileChooser fc = new JFileChooser();

	/** The {@link aber.dcs.clg11.wordladder.model.Graph} data structure
object. */
	private Graph graph;

	/** The {@link aber.dcs.clg11.wordladder.fileIO.ReadFile} object used to
parse external files. */
	private ReadFile fileIO = new ReadFile();


	/**
	 * Instantiates a new WordLadderFrame
	 */
	public WordLadderFrame() {

		//Initialise and set up frame
		this.setTitle("WordLadder - By Connor Goddard (clg11)");
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

		//Prevent user resizing frame
		this.setResizable(false);

		//Initialise the two sub-panels
		generatePan = new GeneratePanel();
		discoveryPan = new DiscoveryPanel();

		//Add the two panels to the north and south of the frame
respectively.
		this.add(generatePan, BorderLayout.NORTH);
		this.add(discoveryPan, BorderLayout.SOUTH);

		//Fit frame to ensure all panels/components are visible
		this.pack();

		//Determine centre of user's screen and position frame accordingly
		Toolkit k=Toolkit.getDefaultToolkit();
		Dimension d=k.getScreenSize();
		this.setLocation(d.width/2-this.getWidth()/2,d.height/2-
this.getHeight()/2);

		//Display frame on screen
		this.setVisible(true);

		//Update both output window displays and inform user that the system
is currently genertaing the graph
		updateDisplays("Welcome to the Word Ladder App - By Connor Goddard
(clg11)");
```

```
	updateDisplays("**********************************************************
**********");
		updateDisplays("Generating graph...");

		//Run method to load dictionary file and generate graph
		loadFile();

		if (graph.getGraphSize() > 0) {

			updateDisplays("Graph created successfully. Total nodes: " +
graph.getGraphSize());
		}

	}


	/**
	 * Update displays.
	 *
	 * @param update the update
	 */
	public void updateDisplays(String update) {

		generatePan.updateDisplay(update);
		discoveryPan.updateDisplay(update);

	}

	/**
	 * Load file.
	 */
	public void loadFile() {

		//Display dialog asking user if they wish to use a internal
dictionary file,
		//or load a new external file
		int exit = JOptionPane.YES_NO_CANCEL_OPTION;
		exit = JOptionPane.showConfirmDialog(null, "Would you like to use an
internal dictionary file?", "WordLadder | Start-Up",
JOptionPane.YES_NO_CANCEL_OPTION);

		//If user selects "No"..
		if (exit == JOptionPane.NO_OPTION) {

			//Ensure user can only select files (not directories)
			fc.setFileSelectionMode(JFileChooser.FILES_ONLY);

			//Removes any existing file filters
			for (FileFilter f : fc.getChoosableFileFilters()){
				fc.removeChoosableFileFilter(f);
			}

			//Create and add pre-defined file filters
			fc.addChoosableFileFilter(new FileNameExtensionFilter("DAT
files", "dat"));
			fc.addChoosableFileFilter(new FileNameExtensionFilter("TXT
files", "txt"));

			//Remove the "All Files" filter
			fc.setAcceptAllFileFilterUsed(false);
```

```
                    //Display the file chooser dialog
                    int returnVal = fc.showOpenDialog(null);

                    //Check if user has approved save
                    if (returnVal == JFileChooser.APPROVE_OPTION) {

                          //Generate a new Graph object using data obtained from
the selected external file.
                          graph = new
Graph(fileIO.readIn(fc.getSelectedFile().toString()));

                          //If the generated graph is empty
                          if (graph.getGraphSize() == 0) {

                                //Display dialog asking user to select a file to
load
                                JOptionPane.showMessageDialog(new JFrame(),
"Please ensure a file is selected to continue.", "WordLadder | Start-Up",
JOptionPane.ERROR_MESSAGE);

                                //Return to original loading dialog
                                loadFile();

                          }

                          //Otherwise update sub-panel Graph objects with newly
generated graph
                          generatePan.setGraph(graph);
                          discoveryPan.setGraph(graph);

                          //Otherwise of the user cancels/closes the file chooser
dialog
                    } else {

                          //Terminate the program as no graph has been generated
                          System.exit(0);
                    }

             //If the user selects the "Yes" button when  asked if they wish to
load an internal dictionary file
             } else if (exit == JOptionPane.YES_OPTION) {

                    try {

                    //Initialise and display checkbox input dialog
                    Object[] possibilities = {"4 Letters", "5 Letters", "6
Letters"};
                    String s = (String)JOptionPane.showInputDialog(null,"Select a
word length:","Select Dictionary File |
WordLadder",JOptionPane.PLAIN_MESSAGE,null, possibilities, "WordLadder | Start-
Up");

                    //If the user selects the "4 letters" option
                    if (s.equals("4 Letters")) {

                          //Create a graph using the 4 letter dictionary file
                          graph = new Graph(fileIO.readIn("dict4.dat"));

                    } else if (s.equals("5 Letters")) {
```

```
                        graph = new Graph(fileIO.readIn("dict5.txt"));

                    } else if (s.equals("6 Letters")) {

                        graph = new Graph(fileIO.readIn("dict6.dat"));

                    }

                    //If the internal file cannot be located for any reason
                    } catch (NullPointerException nP) {

                        //Display message box asking the user to manually
select a dictionary file
                        JOptionPane.showMessageDialog(new JFrame(), "Internal
file not located - please manually select dictionary file.", "WordLadder | Start-
Up", JOptionPane.ERROR_MESSAGE);

                        //Return to original loading dialog
                        loadFile();
                    }

                    //Otherwise update sub-panel Graph objects with newly
generated graph
                    generatePan.setGraph(graph);
                    discoveryPan.setGraph(graph);


                    //Otherwise if attempts to close the dialog box or clicks
"Cancel"
            } else {

                    //Terminate the program
                    System.exit(0);
            }

      }

}
```

GenericPanel

```
package aber.dcs.clg11.wordladder.gui;

import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.SpringLayout;

import aber.dcs.clg11.wordladder.model.Graph;


/**
 * Abstract class extended by {@link aber.dcs.clg11.wordladder.gui.GeneratePanel}
and
 * {@link aber.dcs.clg11.wordladder.gui.DiscoveryPanel}.
 * Used to contain shared variables and method utilised by both sub-panels.
```

```java
 *
 * @author Connor Goddard (clg11)
 */
@SuppressWarnings("serial")
public abstract class GenericPanel extends JPanel implements ActionListener {

	/** {@link aber.dcs.clg11.wordladder.model.Graph} object passed from
	 * {@link aber.dcs.clg11.wordladder.gui.WordLadderFrame}.*/
	protected Graph graph;

	/** The layout manager used by the panel. */
	protected SpringLayout layout;

	/** The JTextArea output window used in both sub-panels. */
	protected JTextArea outputArea;

	/** The JScrollPane overlay used by both sub-panel output windows. */
	protected JScrollPane outputScroll;

	/**
	 * Instantiates a new GenericPanel.
	 */
	public GenericPanel() {

		//Initalise the layout manager
		layout = new SpringLayout();
		this.setLayout(layout);

		//Define the size of the panel
		this.setPreferredSize(new Dimension(480, 210));
	}

	/**
	 * Sets the local {@link aber.dcs.clg11.wordladder.model.Graph} object
	 * to a new object.
	 *
	 * @param newGraph New {@link aber.dcs.clg11.wordladder.model.Graph}
object.
	 */
	public void setGraph(Graph newGraph) {
		this.graph = newGraph;
	}

	/**
	 * Appends and updates the JTextPane output window with new information.
	 *
	 * @param update The new message to be appended to the output window.
	 */
	public void updateDisplay(String update) {

		//Appends the new message to the output window before adding a new
line.
		this.outputArea.append(update + "\n");

	}

	/**
	 * Abstract method implemented by {@link
aber.dcs.clg11.wordladder.gui.GeneratePanel} and
	 * {@link aber.dcs.clg11.wordladder.gui.DiscoverPanel}.
	 * Initialises and adds the GUI components to the sub-panel.
```

```
	 */
	public abstract void initComponents();

	/**
	 * Abstract method implemented by {@link
aber.dcs.clg11.wordladder.gui.GeneratePanel} and
	 * {@link aber.dcs.clg11.wordladder.gui.DiscoverPanel}.
	 * Configures the layout manager to organise all components on the panel.
	 */
	public abstract void setUpLayout();

	/**
	 * Abstract listener for actions from sub-panel buttons, to allow
operations to be run when clicked.
	 * @see
java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
	 * @param evt - ActionEvent called from button components in the sub-panels
that require an action to be performed.
	 */
	@Override
	public abstract void actionPerformed(ActionEvent evt);

}
```

GeneratePanel

```
package aber.dcs.clg11.wordladder.gui;

import java.awt.Dimension;
import java.awt.event.ActionEvent;
import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.SpringLayout;

import aber.dcs.clg11.wordladder.model.OutputStack;

/**
 * Contains the GUI elements utilised by the user to run the 'Generation' mode of
the application.
 * Extends from {@link aber.dcs.clg11.wordladder.gui.GenericPanel}.
 *
 * @author Connor Goddard (clg11)
 */
@SuppressWarnings("serial")
public class GeneratePanel extends GenericPanel {

	/** Button used to run the Breadth first search algorithm. */
	private JButton buttonGen;

	/** The component title labels. */
	private JLabel inputTitle, lengthTitle;

	/** The two input text boxes (start node and word ladder length) */
	private JTextField textWord, textLength;
```

```
	/**
	 * Instantiates a new GeneratePanel.
	 */
	public GeneratePanel() {

		//Create a titled border around the panel
		this.setBorder(BorderFactory.createTitledBorder("Generation"));

		//Initialise all panel components
		this.initComponents();

		//Define panel layout settings for layout manager
		this.setUpLayout();

	}

	/**
	 * Initialises the panel components (including linking components to
listeners where required) before
	 * adding the components to the panel.
	 */
	@Override
	public void initComponents() {

		//Create new instance of JLabel with specified display text
		inputTitle = new JLabel("Search term:");
		lengthTitle = new JLabel("Length:");

		//Set maximum size of text box characters
		textWord = new JTextField(7);
		textLength = new JTextField(3);

		//Create new instance of JButton with specified display text
		buttonGen = new JButton("Generate Ladder");

		//Link action listener to the button component
		buttonGen.addActionListener(this);

		//Create new instance of JTextArea (used for output window)
		outputArea = new JTextArea();

		//Prevent the user from editing this text area
		outputArea.setEditable(false);

		//Create new instance of scroll-able window which overlays the
'outputArea' window
		outputScroll = new JScrollPane(outputArea);
		outputScroll.setPreferredSize(new Dimension(400, 100));

		//Add the components to the panel
		this.add(inputTitle);
		this.add(textWord);
		this.add(lengthTitle);
		this.add(textLength);
		this.add(buttonGen);
		this.add(outputScroll);
	}

	/**
```

```
	 * Configures the 'SpringLayout' layout manager to organise all components
on the panel.
	 */
	@Override
	public void setUpLayout() {

		//Set NORTH edge of 'inputTitle' label, 10 pixels down from NORTH
edge of panel

	layout.putConstraint(SpringLayout.NORTH,inputTitle,10,SpringLayout.NORTH,
this);

	layout.putConstraint(SpringLayout.WEST,inputTitle,10,SpringLayout.WEST,
this);


	layout.putConstraint(SpringLayout.NORTH,textWord,10,SpringLayout.NORTH,
this);

	layout.putConstraint(SpringLayout.WEST,textWord,10,SpringLayout.EAST,
inputTitle);


	layout.putConstraint(SpringLayout.NORTH,lengthTitle,10,SpringLayout.NORTH,
this);

	layout.putConstraint(SpringLayout.WEST,lengthTitle,10,SpringLayout.EAST,
textWord);


	layout.putConstraint(SpringLayout.NORTH,textLength,10,SpringLayout.NORTH,
this);

	layout.putConstraint(SpringLayout.WEST,textLength,10,SpringLayout.EAST,
lengthTitle);


	layout.putConstraint(SpringLayout.NORTH,buttonGen,5,SpringLayout.NORTH,
this);

	layout.putConstraint(SpringLayout.WEST,buttonGen,10,SpringLayout.EAST,
textLength);


	layout.putConstraint(SpringLayout.NORTH,outputScroll,30,SpringLayout.NORTH,
textWord);

	layout.putConstraint(SpringLayout.WEST,outputScroll,35,SpringLayout.WEST,
this);

	}

	/**
	 * @see
aber.dcs.clg11.wordladder.gui.GenericPanel#actionPerformed(java.awt.event.ActionEv
ent)
	 */
	@Override
	public void actionPerformed(ActionEvent evt) {
		String actionCommand = evt.getActionCommand();
```

```
            //If the button contained within the panel is clicked..
            if (actionCommand.equals("Generate Ladder")) {

                  try {

                        //Check if any of the input textboxes are empty..
                        if (textWord.getText().equals("") ||
textLength.getText().equals("")) {

                              //If they are, display an error message box
informing the user of this
                              JOptionPane.showMessageDialog(new JFrame(),
"Please ensure all fields are completed.", "WordLadder | Generator",
JOptionPane.ERROR_MESSAGE);

                              //Otherwise verify that the graph has been
generated..
                        } else if (graph == null) {

                              //If it has not, inform the user of this by
displaying a message box
                              JOptionPane.showMessageDialog(new JFrame(),
"Graph is still generating - please wait.", "WordLadder | Generator",
JOptionPane.ERROR_MESSAGE);

                              //Otherwise..
                        } else {

                              //Retrieve the maximum word ladder length entered
by the user
                              //Deduct one from the value to account for the
start node within the ladder
                              int iterate =
(Integer.parseInt(textLength.getText()) - 1);

                              //If the depth limited search was successful
(using the data entered by the user)
                              if (graph.depthSearch(this.textWord.getText(), 0,
iterate)) {

                                    //If it was, initialise a new local stack
populated by the OutputStack contained within the Graph object
                                    OutputStack output =
graph.getOutputStack();

                                    //Obtain the total size of the generated
word ladder
                                    int length = output.sizeOf();

                                    //Initialise output window display string
                                    String ladder = "Word ladder: " +
textWord.getText();

                                    //For all elements contained within the
stack
                                    while (! output.isEmpty()) {

                                          //Retrieve the top element and
append that to the word ladder display string
                                          ladder = ladder + " --> " +
output.getHead();
```

```
                                        //Remove this top element from the
stack
                                        output.pop();
                                    }

                                    //Check if a word ladder was able to be
produced
                                    if (length == 0) {

                                        //If not, display a message to the
GUI output window informing the user
                                        updateDisplay("FAILURE: No words
link from " + textWord.getText() + ".");

                                        //Otherwise check if the maximum
possible length of the word ladder < the specified ladder length
                                    } else if (length < iterate) {

                                        //If this is the case, inform the
user of this fact before outputting the generated word ladder
                                        updateDisplay("WARNING: Maximum
ladder length available - " + (length + 1) + " words.");
                                        updateDisplay(ladder);

                                        //Otherwise, if a word ladder of the
speified length was able to the generated
                                    } else {

                                        //Display message to GUI output
window informing user
                                        updateDisplay("SUCCESS: Generation
completed successfully.");

                                        //Display the generated word ladder
in the GUI output window
                                        updateDisplay(ladder);
                                    }


                                    //Otherwise if the DLS returned false
(i.e. the entered word does not exist within the graph)
                                } else {

                                    //Display message to user via GUI output
window informing them of this
                                    updateDisplay("FAILURE: Ladder cannot be
generated - entered value does not exist.");
                                }

                            }

                            //If the user enters a sting into the 'word ladder
length' text box..
                        } catch (NumberFormatException nF) {

                            //Display error message dialog requesting them to
change the value
                            JOptionPane.showMessageDialog(new JFrame(), "Please
enter an integer value for ladder length", "WordLadder | Generator",
JOptionPane.ERROR_MESSAGE);
```

```
                }

            }
        }

}
```

DiscoveryPanel

```
package aber.dcs.clg11.wordladder.gui;

import java.awt.Dimension;
import java.awt.event.ActionEvent;
import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.SpringLayout;

import aber.dcs.clg11.wordladder.model.OutputStack;


/**
 * Contains the GUI elements utilised by the user to run the 'Discovery' mode of
the application.
 * Extends from {@link aber.dcs.clg11.wordladder.gui.GenericPanel}.
 *
 * @author Connor Goddard (clg11)
 */
@SuppressWarnings("serial")
public class DiscoveryPanel extends GenericPanel {

        /** Button used to run the Breadth first search algorithm. */
        private JButton buttonDisc;

        /** The component title labels. */
        private JLabel startTitle, targetTitle;

        /** The two word input textboxes (start word and target word) */
        private JTextField textStart, textTarget;


        /**
         * Instantiates a new DiscoveryPanel.
         */
        public DiscoveryPanel() {

                //Create a tiled border around the panel
                this.setBorder(BorderFactory.createTitledBorder("Discovery"));

                //Initialise all panel components
                this.initComponents();

                //Define panel layout settings for layout manager
                this.setUpLayout();
        }
```

```
	/**
	 * Initialises the panel components (including linking components to
listeners where required) before
	 * adding the components to the panel.
	 */
	@Override
	public void initComponents() {

		//Create new instance of JLabel with specified display text
		startTitle = new JLabel("Start:");
		targetTitle = new JLabel("Target:");

		//Set maximum size of amount text box to 15 characters
		textStart = new JTextField(15);
		textTarget = new JTextField(15);

		//Create new instance of JButton with specified display text
		buttonDisc = new JButton("Run Search");

		//Link action listener to the button component
		buttonDisc.addActionListener(this);

		//Create new instance of JTextArea (used for output window)
		outputArea = new JTextArea();

		//Prevent the user from editing this text area
		outputArea.setEditable(false);

		//Create new instance of scroll-able window which overlays the
'outputArea' window
		outputScroll = new JScrollPane(outputArea);
		outputScroll.setPreferredSize(new Dimension(400, 100));

		//Add the components to the panel
		this.add(startTitle);
		this.add(targetTitle);
		this.add(textStart);
		this.add(textTarget);
		this.add(buttonDisc);
		this.add(outputScroll);
	}

	/**
	 *  Configures the 'SpringLayout' layout manager to organise all components
on the panel.
	 */
	@Override
	public void setUpLayout() {

		//Set NORTH edge of 'startTitle' label, 10 pixels down from NORTH
edge of panel

	layout.putConstraint(SpringLayout.NORTH,startTitle,10,SpringLayout.NORTH,
this);

	layout.putConstraint(SpringLayout.WEST,startTitle,10,SpringLayout.WEST,
this);
```

```
	layout.putConstraint(SpringLayout.NORTH,textStart,10,SpringLayout.NORTH,
this);

	layout.putConstraint(SpringLayout.WEST,textStart,10,SpringLayout.EAST,
startTitle);


	layout.putConstraint(SpringLayout.NORTH,targetTitle,10,SpringLayout.NORTH,
this);

	layout.putConstraint(SpringLayout.WEST,targetTitle,10,SpringLayout.EAST,
textStart);


	layout.putConstraint(SpringLayout.NORTH,textTarget,10,SpringLayout.NORTH,
this);

	layout.putConstraint(SpringLayout.WEST,textTarget,10,SpringLayout.EAST,
targetTitle);


	layout.putConstraint(SpringLayout.NORTH,buttonDisc,10,SpringLayout.SOUTH,
textStart);

	layout.putConstraint(SpringLayout.WEST,buttonDisc,180,SpringLayout.WEST,
this);


	layout.putConstraint(SpringLayout.NORTH,outputScroll,10,SpringLayout.SOUTH,
buttonDisc);

	layout.putConstraint(SpringLayout.WEST,outputScroll,35,SpringLayout.WEST,
this);

	}

	/**
	 * @see
aber.dcs.clg11.wordladder.gui.GenericPanel#actionPerformed(java.awt.event.ActionEv
ent)
	 */
	@Override
	public void actionPerformed(ActionEvent evt) {
		String actionCommand = evt.getActionCommand();

		//If the button contained within the panel is clicked..
		if (actionCommand.equals("Run Search")) {

			//Check if any of the textboxes are empty
			if (textStart.getText().equals("") ||
textTarget.getText().equals("")) {

				//If they are, display an error message box informing
the user
				JOptionPane.showMessageDialog(new JFrame(), "Please
ensure all fields are completed.", "WordLadder | Generator",
JOptionPane.ERROR_MESSAGE);

				//Otherwise..
			} else {
```

```
                        //Create new instance of an OutputStack
                        OutputStack output = null;

                        //Check that a breadth first search has successfully
run..
                        if (graph.breadthFirstSearch(textStart.getText(),
textTarget.getText())) {

                                //If it has, set the local stack to the
OutputStack contained within the Graph object
                                output = graph.getOutputStack();

                                //Retrieve the total size of the stack (i.e the
size of the shortest path word ladder)
                                int length = output.sizeOf();

                                //Initialise output window display string
                                String ladder = "Word ladder: ";

                                //Update the GUI output window informing the user
that a shortest path between the two specified words has been retrieved
                                //and display the size of the shortest path
                                updateDisplay("SUCCESS: Shortest path between " +
textStart.getText() + " and " + textTarget.getText() + " located in " + length + "
steps.");

                                //For all elements contained within the stack
                                while (! output.isEmpty()) {

                                        //Retrieve the top element and append that
to the word ladder display string
                                        ladder = ladder + " --> " +
output.getHead();

                                        //Remove this top element from the stack
                                        output.pop();
                                }

                                //Update the GUI output window with the complete
word ladder
                                updateDisplay(ladder);

                                //Otherwise if the BFS returns false (i.e it
cannot locate the shortest path between the two nodes)
                        } else {

                                //Update the GUI output window informing that a
path cannot be retrieved.
                                updateDisplay("FAILURE: Shortest path cannot be
established.");

                        }
                }

        }

}
```

Graph

```
package aber.dcs.clg11.wordladder.model;

import java.util.Enumeration;
import java.util.Hashtable;
import java.util.LinkedList;
import java.util.Vector;


/**
 * Main data class within the application used to represent the Graph data
structure.
 * Contains both search algorithms (BFS and DLS) and the adjacency list
implementation via a hash table.
 *
 * @author Connor Goddard (clg11)
 */
public class Graph {

	/** Adjacency list implementation by using a Hash table. */
	private Hashtable<String, Node> nodeDirectory = new Hashtable<String,
Node>();

	/** OutputStack used to store search results. */
	private OutputStack outputStack = new OutputStack();

	/**
	 * Default constructor for Graph.
	 */
	public Graph() {

	}

	/**
	 * Instantiates a new Graph, with the ability to generate the graph using a
	 * Vector of strings.
	 *
	 * @param input Vector containing data to populate the graph.
	 */
	public Graph(Vector<String> input) {

		createGraph(input);
	}

	/**
	 * Populates the adjacency list structure using data obtained
	 * via the {@link aber.dcs.clg11.wordladder.model.ReadFile} class.
	 *
	 * @param input Vector containing data to populate the graph
	 */
	public void createGraph(Vector<String> input) {

		//For every element in the vector of words
		for (String s : input) {

			//Create a new Node object to represent the current word
			Node newNode = new Node(s);

			//Iterate through existing elements in hash table
			Enumeration<String> e = nodeDirectory.keys();
```

```
                        //Check that the has table size > 0
                        if (nodeDirectory.size() > 0) {

                                //For every existing Node in the hash table
                                while (e.hasMoreElements()) {

                                        String nextElement = e.nextElement();

                                        //Compare the new word with the current node in
the hash table
                                        //Check if the two words are one letter apart
                                        if (checkWords(newNode.getNodeData(),
nextElement)) {

                                                //If they are one letter apart..

                                                //Create a temporary object to represent
the identified node within the hash table
                                                Node otherNode =
nodeDirectory.get(nextElement);

                                                //Create an edge between the two nodes
                                                newNode.addLink(otherNode);
                                                otherNode.addLink(newNode);

                                        }
                                }

                        }

                        //Add the new Node object to the hash table
                        nodeDirectory.put(s, newNode);
                }

        }

        /**
         * Compares two strings to identify if they are one letter apart.
         *
         * @param word1 The first word to compare
         * @param word2 The second word to compare
         * @return true If the two words are one letter apart
         */
        public boolean checkWords(String word1, String word2) {

                //If the lengths of the two words are different
                if (word1.length() != word2.length()) {

                        //Return false as the words are going to be different
                        return false;

                        //Otherwise..
                } else {

                        int differ = 0;

                        //For every character that makes up the first word..
                        for (int i = 0; i < word1.length(); i++) {
```

```
                        //Check if the letter at the position in word one,
matches the letter at the same position in word 2
                        if (word1.charAt(i) != word2.charAt(i)) {

                                //If the letters do not match, increment the
value
                                differ++;
                        }
                }

                //Return true if the value is 1 (i.e. there is a one letter
difference)
                return (differ == 1);

        }
    }


    /**
     * Depth limited search that traverses through the graph from a specified
root node to
     * create a word ladder of specified length.
     *
     * @param word The specified root node (also used as part of the recursive
function).
     * @param currentDepth The current depth of the search within the graph
(used as part of the recursive function)
     * @param maxDepth The maximum depth that the search can traverse to (i.e.
the specified word ladder length)
     * @return true Returns if path is able to be retrieved.
     */
    public boolean depthSearch(String word, int currentDepth, int maxDepth) {

        //Declare boolean used within recursive algorithm
        Boolean resultFound;

        //Node object used to represent the current Node being accessed
        Node searchNode;

        //Check if the specified root node exists in the graph
        if (getNode(word) == null) {

                //If it does not exist, return false.
                return false;

                //Otherwise if the word does exist..
        } else {

                //Set the local Node variable to the current node being
accessed by the algorithm
                searchNode = getNode(word);

                //Set the Node as being searched to prevent it being searched
again and loops forming
                searchNode.setIsSearched(true);
        }

        //Check if the algorithm has searched to the maximum specified depth
        if (currentDepth < maxDepth) {

                //If it has not..
```

```
                        //For every node that has an edge to the current node
                        for (Node neighbour : searchNode.getNodeList()) {

                                //If the neighbour has not already been searched
                                if (neighbour.getIsSearched().equals(false)) {

                                        //Set the neighbour to 'isSearched' value to true
                                        neighbour.setIsSearched(true);

                                        //Recursively run the algorithm again for this
neighbour node
                                        resultFound =
depthSearch(neighbour.getNodeData(), currentDepth + 1, maxDepth);

                                        //If the recursive algorithm returns true..
                                        if (resultFound) {

                                                //Push this neighbour to the output stack
                                                outputStack.push(neighbour.getNodeData());

                                                //Return true to trigger the previous
recursive algorithm to terminate
                                                return true;

                                        }

                                }

                        }

                //Reset all node search variables
                resetSearch();

                //Return true to exit algorithm
                return true;

        }

        /**
         * Returns the {@link aber.dcs.clg11.wordladder.model.Node} object that
         * represents the search term entered by the user.
         *
         * @param searchTerm The word that is to be returned.
         * @return The node object that represents the specified word.
         */
        public Node getNode(String searchTerm) {

                return nodeDirectory.get(searchTerm);
        }

        /**
         * Resets search variables contained within every {@link
aber.dcs.clg11.wordladder.model.Node} object
         * currently within the {@link aber.dcs.clg11.wordladder.model.Graph}.
         */
        public void resetSearch() {

                //Iterate through the hash table
```

```
		Enumeration<String> e = nodeDirectory.keys();

		//For every Node object..
		while (e.hasMoreElements()) {

			Node reset = nodeDirectory.get(e.nextElement());

			//Reset the search variables
			reset.setIsSearched(false);
			reset.setParent(null);
		}

	}

	/**
	 * Breadth first search algorithm that traverses from the specified root
node through every
	 * level of the {@link aber.dcs.clg11.wordladder.model.Graph} until it
locates the specified target node.
	 * The algorithm then returns the shortest possible path (i.e. word ladder)
between the root and target nodes.
	 *
	 * Code modified from original source code available at:
http://www.codeproject.com/script/Articles/ViewDownloads.aspx?aid=32212
	 *
	 * @param word The specified root node.
	 * @param target The specified target node.
	 * @return true Returns if a path is able to be retrieved.
	 */
	public boolean breadthFirstSearch(String word, String target) {

		//Create a new queue used for storing nodes that are still to be
checked
		LinkedList<Node> q = new LinkedList<Node>();

		//If the either the root, or target word cannot be located as a node
within the graph..
		if (getNode(word) == null || getNode(target) == null) {

			//Terminate the search algorithm
			return false;
		}

		//Otherwise, retrieve the start word node from the graph
		Node root = getNode(word);

		//Add this node to the process queue
		q.add(root);

		//Set the 'isSearched' value to true to prevent the Node being
checked again
		root.setIsSearched(true);

		//While there are still Nodes to be checked
		while(! q.isEmpty()) {

			//Remove the next Node to be checked from the queue and set to
local variable
			Node n = q.remove();

			//For every Node with an edge to the current Node
```

```
                        for (Node child : n.getNodeList()) {

                                //If that Node has not already been checked..
                                if (child.getIsSearched().equals(false)) {

                                        //Prevent it being checked again for this search
                                        child.setIsSearched(true);

                                        //Set the parent node of this Node to the Node
that it is "child" of (i.e neighbour)
                                        //Used to return the path (word ladder) once a
search has completed
                                        child.setParent(n);

                                        //Add this Node to the process queue to allow its
neighbours to be searched
                                        q.add(child);

                                }

                        }
                }

                //Once all nodes have been searched..

                //Retrieve the Node representing the target node
                Node current = getNode(target);

                //Push this Node to the stack (word ladder)
                outputStack.push(current.getNodeData());

                //Until the start word node is reached
                while(! current.equals(getNode(word))) {

                        //Retrieve the parent of the 'current' node
                        current = current.getParent();

                        //If one of the nodes does not have a parent
                        //There is no word ladder available between the two words
                        if (current == null) {

                                //Therefore reset the node search variables
                                resetSearch();

                                //Terminate the search algorithm informing the GUI that
a ladder cannot be found
                                return false;
                        }

                        //Otherwise push the 'current' node to the output stack
                        outputStack.push(current.getNodeData());
                }

                //Reset the node search variables
                resetSearch();

                //Terminate the algorithm informing the GUI that a ladder was
located
                return true;
        }
```

```
	/**
	 * Retrieves the {@link aber.dcs.clg11.wordladder.model.OutputStack} object
used to store the result
	 * path of either search algorithm
	 *
	 * @return the output stack containing the word ladder path generated by
the search algorithms.
	 */
	public OutputStack getOutputStack() {
		return outputStack;
	}

	/**
	 * Retrieves the total size of the graph by counting all the elements
(Nodes) contained within the hash table.
	 *
	 * @return the number of elements within the hash table (adjacency list).
	 */
	public int getGraphSize() {

		return nodeDirectory.size();

	}

}
```

Node

```
package aber.dcs.clg11.wordladder.model;

import java.util.Vector;

/**
 * Data class used to represent each word read in from a dictionary file as a
 * 'Node' element within the {@link aber.dcs.clg11.wordladder.model.Graph} data
structure.
 *
 * @author Connor Goddard (clg11)
 */
public class Node {

	/** The name of the word that the Node represents */
	private String nodeData;

	/** Vector that contains a list of all other nodes that has an edge to this
node. */
	private Vector<Node> nodeList = new Vector<Node>();

	/** Used by the search algorithm to determine if this node has already been
searched */
	private boolean isSearched = false;

	/** Used by BFS to traverse through an identified path (word ladder) */
	private Node parent = null;

	/**
	 * Default constructor for a new Node.
	 */
	public Node() {

	}
```

```java
	/**
	 * Constructor for a new Node that allows the name of the node to be
specified.
	 *
	 * @param data the data
	 */
	public Node(String data) {
		setNodeData(data);
	}

	/**
	 * Returns the parent Node of this Node.
	 *
	 * @return the parent node
	 */
	public Node getParent() {
		return parent;
	}

	/**
	 * Sets the parent Node of this Node.
	 *
	 * @param newParent The new parent node to be set.
	 */
	public void setParent(Node newParent) {
		parent = newParent;
	}

	/**
	 * Returns the name of the node (i.e. the word this node represents)
	 *
	 * @return the name of the node
	 */
	public String getNodeData() {
		return nodeData;
	}

	/**
	 * Sets the name of the node.
	 *
	 * @param nodeData The new name for the node.
	 */
	public void setNodeData(String nodeData) {
		this.nodeData = nodeData;
	}

	/**
	 * Returns the vector that contains a list of all neighbour Nodes.
	 *
	 * @return the vector of neighbour nodes.
	 */
	public Vector<Node> getNodeList() {
		return nodeList;
	}

	/**
	 * Sets the list of neighbour nodes.
	 *
	 * @param newNodeList The new vector of neighbour nodes to be set.
	 */
```

```
	public void setNodeList(Vector<Node> newNodeList) {
		nodeList = newNodeList;
	}

	/**
	 * Creates a reference to a new neighbour node in the list of neighbour
nodes.
	 *
	 * @param newNode the new node to be added.
	 */
	public void addLink(Node newNode) {
		nodeList.add(newNode);
	}

	/**
	 * Sets the isSearched boolean which is used by the search algorithms.
	 *
	 * @param search The boolean to set the isSearched boolean.
	 */
	public void setIsSearched(Boolean search) {
		isSearched = search;
	}

	/**
	 * Returns the isSearched boolean.
	 *
	 * @return the boolean the determines if the node has already been
searched.
	 */
	public Boolean getIsSearched() {
		return isSearched;
	}
}
```

OutputStack

```
package aber.dcs.clg11.wordladder.model;

import java.util.ArrayList;

/**
 * Class used to represent a Stack data structure. Utilised DLS and BFS by the
{@link aber.dcs.clg11.wordladder.model.Graph}
 * class to output the result of word ladder searches.
 *
 * @author Connor Goddard (clg11)
 */
public class OutputStack {

	/** Array list used as the data structure for the stack.  */
	private ArrayList<String> stack;

	/**
	 * Instantiates a new OutputStack.
	 */
	public OutputStack() {

		stack = new ArrayList<String>();

	}
```

```java
	/**
	 * Pushes a new element to the top of the stack.
	 *
	 * @param element The element to be pushed onto the stack.
	 */
	public void push(String element) {

		stack.add(element);

	}

	/**
	 * Removes the top element from the stack.
	 */
	public void pop() {

		if (stack.isEmpty()) {

			throw new StackOverflowError("Stack is empty");

		} else {

			stack.remove(stack.size() - 1);

		}
	}

	/**
	 * Returns the total size of the stack.
	 *
	 * @return the size of the stack.
	 */
	public int sizeOf() {

		return stack.size();

	}

	/**
	 * Checks if the stack is empty.
	 *
	 * @return true if the stack is empty.
	 */
	public boolean isEmpty() {

		return stack.isEmpty();
	}

	/**
	 * Returns the top element of the stack.
	 *
	 * @return the top element of the stack.
	 */
	public String getHead() {

		return stack.get(stack.size() - 1);
	}

}
```

ReadFile

```
package aber.dcs.clg11.wordladder.fileIO;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Vector;


/**
 * Accesses and parses internal/external dictionary files. Takes contents of
dictionary file and returns the contents
 * as a Vector.
 *
 * @author Connor Goddard (clg11)
 */
public class ReadFile {

	/**
	 * Reads in the contents of internal/external dictionary files and places
the contents into a Vector
	 * which is then returned, and used to populate the {@link
aber.dcs.clg11.wordladder.model.Graph}.
	 *
	 * @param fileName The directory of the file to be parsed.
	 * @return Vector containing the contents of the parsed file.
	 */
	public Vector<String> readIn(String fileName) {

		try {

			//Create File IO objects
			FileReader fileReader;
			BufferedReader bufferedReader;

			//Initialise Vector used to store file contents (i.e.
collection of words)
			Vector<String> words = new Vector<String>();

			//Initialise the File IO objects, passing in the selected file
path
			fileReader = new FileReader(fileName);
			bufferedReader = new BufferedReader(fileReader);

			//Initialise local variable used to store the current line
being read in
			String line = null;


			//While there are still lines to read in from the file (i.e.
read in every line in the file)
			while ((line = bufferedReader.readLine()) != null) {

				//Add the current word to the Vector of words
				words.add(line);

			}

			//Once completed, safely close the file reader
			bufferedReader.close();
```

```
                        //Return the vector of words obtained from the file
                        return words;

                //If any IO exceptions occur...
                } catch (IOException iOE) {

                        //.. return nothing.
                        return null;
                }

        }
}
```

Appendix 1.2

AllTests

```
package aber.dcs.clg11.wordladder.tests;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

/**
 * Test suite used to execute all JUnit tests.
 *
 * @author Connor Goddard (clg11)
 */
@RunWith(Suite.class)
@SuiteClasses({ GraphTest.class, NodeTest.class, OutputStackTest.class,
                ReadFileTest.class })
public class AllTests {

}
```

GraphTest

```
package aber.dcs.clg11.wordladder.tests;

import static org.junit.Assert.*;

import java.util.ArrayList;
import java.util.Vector;

import org.junit.Before;
import org.junit.Test;

import aber.dcs.clg11.wordladder.fileIO.ReadFile;
import aber.dcs.clg11.wordladder.model.Graph;
import aber.dcs.clg11.wordladder.model.Node;
import aber.dcs.clg11.wordladder.model.OutputStack;


/**
 * JUnit test class for scrutinising the {@link
aber.dcs.clg11.wordladder.model.Graph} class.
```

```
 *
 * @author Connor Goddard (clg11)
 */
public class GraphTest {

	private Graph testGraph;

	private ReadFile fileIO = new ReadFile();

	/**
	 * Sets up the testGraph variable for each test.
	 *
	 * @throws Exception the exception
	 */
	@Before
	public void setUp() throws Exception {

		testGraph = new Graph();
	}

	/**
	 * Method used to generate the testGraph object used within the JUnit
tests.
	 *
	 * @param filePath The file path of the file to load in.
	 * @return the vector of the contents read in from the file.
	 */
	public Vector<String> createGraph(String filePath) {

		Vector<String> output = new Vector<String>();

		for (String s : fileIO.readIn(filePath)) {

			output.add(s);

		}

		return output;
	}


	/**
	 * Loads the file containing four words.
	 */
	public void loadFourLetterWords() {

		testGraph = new Graph(createGraph("dict4.dat"));

	}


	/**
	 * Loads the file containing five words.
	 */
	public void loadFiveLetterWords() {

		testGraph = new Graph(createGraph("dict5.txt"));
	}


	/**
```

```
	 * Loads the file containing six words.
	 */
	public void loadSixLetterWords() {

		testGraph = new Graph(createGraph("dict6.dat"));

	}

	/**
	 * Loads the file containing a sample of test words.
	 */
	public void loadTestWords() {

		testGraph = new Graph(createGraph("testWords.txt"));

	}

	/**
	 * Test method for {@link aber.dcs.clg11.wordladder.model.Graph#Graph()}.
	 */
	@Test
	public void testDefaultConstructor() {

		testGraph = new Graph();

		assertEquals("Graph size should equal 0", 0,
testGraph.getGraphSize());


	}

	/**
	 * Test method for {@link
aber.dcs.clg11.wordladder.model.Graph#Graph(java.lang.String)}.
	 */
	@Test
	public void testFilenameConstructor() {

		loadTestWords();

		assertEquals("Graph size should be 5", 5, testGraph.getGraphSize());
	}

	/**
	 * Test method for {@link
aber.dcs.clg11.wordladder.model.Graph#createGraph(java.lang.String)}.
	 */
	@Test
	public void testCreateGraph() {

		loadTestWords();

		assertEquals("Graph size should be 5", 5, testGraph.getGraphSize());
	}

	/**
	 * Test method for {@link
aber.dcs.clg11.wordladder.model.Graph#checkWords(java.lang.String,
java.lang.String)}.
	 */
	@Test
```

```
	public void testCheckWords() {

		assertTrue("Should return true", testGraph.checkWords("test",
"teat"));

		assertFalse("Should return false as words are more than one letter
apart", testGraph.checkWords("test", "feat"));

		assertFalse("Should return false as words are equal and so not one
letter apart", testGraph.checkWords("test", "test"));

		assertTrue("Should return true", testGraph.checkWords("setter",
"getter"));

	}

	/**
	 * Test method for checking invalid four letter words within the graph.
	 */
	@Test
	public void testDepthSearchFourLetterInvalidTerm() {

		loadFourLetterWords();

		assertFalse("Algortihm should not have returned a result as this
word does not exist", testGraph.depthSearch("breat", 0, 3));

		assertEquals("Stack should be empty as no search can take place", 0,
testGraph.getOutputStack().sizeOf());

	}

	/**
	 * Test method for {@link
aber.dcs.clg11.wordladder.model.Graph#depthSearch(java.lang.String, int, int)}.
	 */
	@Test
	public void testDepthSearchFourLetterValidTerm() {

		loadFourLetterWords();

		assertTrue("Method should have found result and so returns true",
testGraph.depthSearch("test", 0, 4));

		OutputStack testStack = testGraph.getOutputStack();

		ArrayList<String> testArray = new ArrayList<String>();
		testArray.add("teat");
		testArray.add("teas");
		testArray.add("tear");
		testArray.add("team");

		for (int i = 0; i < testArray.size(); i++) {

			assertEquals(testArray.get(i), testStack.getHead());
			testStack.pop();

		}

	}
```

```
	/**
	 * Test method for checking invalid five letter words within the graph.
	 */
	@Test
	public void testDepthSearchFiveLetterInvalidTerm() {

		loadFiveLetterWords();

		assertFalse("Algortihm should not have returned a result as this
word does not exist", testGraph.depthSearch("teff", 0, 4));

		assertEquals("Stack should be empty as no search can take place", 0,
testGraph.getOutputStack().sizeOf());

	}

	/**
	 * Test method for {@link
aber.dcs.clg11.wordladder.model.Graph#depthSearch(java.lang.String, int, int)}.
	 */
	@Test
	public void testDepthSearchFiveLetterValidTerm() {

		loadFiveLetterWords();

		assertTrue("Method should have found result and so returns true",
testGraph.depthSearch("light", 0, 3));

		OutputStack testStack = testGraph.getOutputStack();

		ArrayList<String> testArray = new ArrayList<String>();
		testArray.add("eight");
		testArray.add("bight");
		testArray.add("bigot");


		for (int i = 0; i < testArray.size(); i++) {

			assertEquals(testArray.get(i), testStack.getHead());
			testStack.pop();

		}

	}

	/**
	 * Test method for checking invalid six letter words within the graph.
	 */
	@Test
	public void testDepthSearchSixLetterInvalidTerm() {

		loadSixLetterWords();

		assertFalse("Algortihm should not have returned a result as this
word does not exist", testGraph.depthSearch("costos", 0, 4));

		assertEquals("Stack should be empty as no search can take place", 0,
testGraph.getOutputStack().sizeOf());

	}
```

```
	/**
	 * Test method for {@link
aber.dcs.clg11.wordladder.model.Graph#depthSearch(java.lang.String, int, int)}.
	 */
	@Test
	public void testDepthSearchSixLetterValidTerm() {

		loadSixLetterWords();

		assertTrue("Method should have found result and so returns true",
testGraph.depthSearch("pelves", 0, 5));

		OutputStack testStack = testGraph.getOutputStack();

		ArrayList<String> testArray = new ArrayList<String>();
		testArray.add("delves");
		testArray.add("deaves");
		testArray.add("heaves");
		testArray.add("heaven");
		testArray.add("heaved");


		for (int i = 0; i < testArray.size(); i++) {

			assertEquals(testArray.get(i), testStack.getHead());
			testStack.pop();

		}

	}


	/**
	 * Test method for {@link
aber.dcs.clg11.wordladder.model.Graph#getNode(java.lang.String)}.
	 */
	@Test
	public void testGetNode() {

		loadFourLetterWords();

		assertEquals("Returned node should be Coil",
testGraph.getNode("coil").getNodeData(), "coil");
	}

	/**
	 * Test method for {@link
aber.dcs.clg11.wordladder.model.Graph#resetSearch()}.
	 */
	@Test
	public void testResetSearch() {

		loadFourLetterWords();

		Node testNode = new Node("Parent");

		testGraph.getNode("test").setIsSearched(true);
		testGraph.getNode("test").setParent(testNode);

		assertTrue(testGraph.getNode("test").getIsSearched());
		assertEquals(testGraph.getNode("test").getParent(), testNode);
```

```
            testGraph.resetSearch();

            assertFalse(testGraph.getNode("test").getIsSearched());
            assertEquals(testGraph.getNode("test").getParent(), null);

      }

      /**
       * Test method for {@link
aber.dcs.clg11.wordladder.model.Graph#breadthFirstSearch(java.lang.String,
java.lang.String)}.
       */
      @Test
      public void testBreadthFirstSearchFourLetter() {

            loadFourLetterWords();

            assertTrue("Path can be obtained so BFS should return
true",testGraph.breadthFirstSearch("head", "foot"));

            assertEquals("Total path size should be 6 as this is the shortest
path", 6, testGraph.getOutputStack().sizeOf());

      }

      /**
       * Test method for {@link
aber.dcs.clg11.wordladder.model.Graph#breadthFirstSearch(java.lang.String,
java.lang.String)}.
       */
      @Test
      public void testBreadthFirstSearchFourLetterInvalidValue() {

            loadFourLetterWords();

            assertFalse("Origin word contains too many characters so BFS should
return false", testGraph.breadthFirstSearch("trail", "foot"));

            assertEquals("Total path size should be 0 as this is the shortest
path", 0, testGraph.getOutputStack().sizeOf());
      }

      /**
       * Test method for {@link
aber.dcs.clg11.wordladder.model.Graph#breadthFirstSearch(java.lang.String,
java.lang.String)}.
       */
      @Test
      public void testBreadthFirstSearchFiveLetter() {

            loadFiveLetterWords();

            assertTrue("Path can be obtained so BFS should return true",
testGraph.breadthFirstSearch("train", "plate"));

            assertEquals("Total path size should be 8 as this is the shortest
path", 8, testGraph.getOutputStack().sizeOf());

      }
```

```
	/**
	 * Test method for {@link
aber.dcs.clg11.wordladder.model.Graph#breadthFirstSearch(java.lang.String,
java.lang.String)}.
	 */
	@Test
	public void testBreadthFirstSearchFiveLetterInvalidValue() {

		loadFiveLetterWords();

		assertFalse("Target word does not exist, so BFS should return
false",testGraph.breadthFirstSearch("train", "qwert"));

		assertEquals("Total path size should be 0", 0,
testGraph.getOutputStack().sizeOf());
	}

	/**
	 * Test method for {@link
aber.dcs.clg11.wordladder.model.Graph#breadthFirstSearch(java.lang.String,
java.lang.String)}.
	 */
	@Test
	public void testBreadthFirstSearchSixLetter() {

		loadSixLetterWords();

		assertTrue("Path can be obtained so BFS should return true",
testGraph.breadthFirstSearch("dunned", "grocer"));

		assertEquals("Total path size should be 13 as this is the shortest
path", 13, testGraph.getOutputStack().sizeOf());

	}

	/**
	 * Test method for {@link
aber.dcs.clg11.wordladder.model.Graph#breadthFirstSearch(java.lang.String,
java.lang.String)}.
	 */
	@Test
	public void testBreadthFirstSearchSixLetterInvalidValue() {

		loadSixLetterWords();

		assertFalse("BFS has no target word so should return false",
testGraph.breadthFirstSearch("abacus", ""));

		assertEquals("Total path size should be 0", 0,
testGraph.getOutputStack().sizeOf());
	}


	/**
	 * Test method for {@link
aber.dcs.clg11.wordladder.model.Graph#getOutputStack()}.
	 */
	@Test
	public void testGetOutputStack() {

		testGraph.getOutputStack().push("testElement1");
```

```
		testGraph.getOutputStack().push("testElement2");

		OutputStack testStack = testGraph.getOutputStack();

		assertEquals("testElement2", testStack.getHead());

		testStack.pop();

		assertEquals("testElement1", testStack.getHead());

	}

	/**
	 * Test method for {@link
aber.dcs.clg11.wordladder.model.Graph#getGraphSize()}.
	 */
	@Test
	public void testGetGraphSize() {

		loadTestWords();

		assertEquals("Graph size should match number of words stored in
testWords.txt", 5, testGraph.getGraphSize());
	}

}
```

NodeTest

```
package aber.dcs.clg11.wordladder.tests;

import static org.junit.Assert.*;

import java.util.Vector;

import org.junit.Before;
import org.junit.Test;

import aber.dcs.clg11.wordladder.model.Node;


/**
 * JUnit test class for scrutinising the {@link
aber.dcs.clg11.wordladder.model.Node} class.
 *
 * @author Connor Goddard (clg11)
 */
public class NodeTest {


	Node testNode;

	/**
	 * Sets up the testNode variable for each test.
	 *
	 * @throws Exception the exception
	 */
	@Before
	public void setUp() throws Exception {
```

```
		testNode = new Node();
	}

	/**
	 * Tests the Node default constructor.
	 */
	@Test
	public void testNodeDefaultConstructor() {

		assertNull("No name has been set, so should return null",
testNode.getNodeData());
	}

	/**
	 * Tests the Node constructor that allows the node name to be specified.
	 */
	@Test
	public void testNodeDataConstructor() {

		testNode = new Node("testNode");

		assertEquals("Node data should equal 'testNode'", "testNode",
testNode.getNodeData());
	}

	/**
	 * Tests the get/set predeccessor methods.
	 */
	@Test
	public void testGetSetPredeccessor() {

		Node testNeighbourNode = new Node("testNeighbourNode");

		testNode.setParent(testNeighbourNode);

		assertEquals("Returned parent should be 'testNeighbourNode",
testNeighbourNode, testNode.getParent());

	}

	/**
	 * Tests the get/set nodeData methods.
	 */
	@Test
	public void testGetSetNodeData() {

		testNode.setNodeData("testNodeDataValue");

		assertEquals("Node data should return 'testNodeDataValue'",
"testNodeDataValue", testNode.getNodeData());
	}


	/**
	 * Tests the get/set nodeList methods.
	 */
	@Test
	public void testGetSetNodeList() {

		Vector<Node> testNodeList = new Vector<Node>();
```

```
		for (int i = 0; i < 10; i++) {

			testNodeList.add(new Node("testNodeElement" + i));
		}

		testNode.setNodeList(testNodeList);

		for (int i = 0; i > 9; i++) {

			assertEquals("Node list element should match",
"testNodeElement" + i, testNode.getNodeList().get(i));

		}

	}

	/**
	 * Tests the addLink method to ensure neighbour nodes can be referenced.
	 */
	@Test
	public void testAddLink() {

		assertEquals("List of neighbour nodes should equal 0", 0,
testNode.getNodeList().size());

		for (int i = 0; i < 25; i++) {

			testNode.addLink(new Node("testNodeElement" + i));
		}

		assertEquals("List of neighbour nodes should equal 25", 25,
testNode.getNodeList().size());
	}

	/**
	 * Tests the get/set isSearched methods.
	 */
	@Test
	public void testGetSetSearched() {

		assertFalse(testNode.getIsSearched());

		testNode.setIsSearched(true);

		assertTrue(testNode.getIsSearched());
	}


}
```

OutputStackTest

```
package aber.dcs.clg11.wordladder.tests;

import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

import aber.dcs.clg11.wordladder.model.OutputStack;
```

```java
/**
 * JUnit test class for scrutinising the {@link
aber.dcs.clg11.wordladder.model.OutputStack} class.
 *
 * @author Connor Goddard (clg11)
 */
public class OutputStackTest {

	OutputStack testStack;

	/**
	 * Sets up the testGraph variable for each test.
	 *
	 * @throws Exception the exception
	 */
	@Before
	public void setUp() throws Exception {

		testStack = new OutputStack();
	}

	/**
	 * Tests the OutputStack default constructor.
	 */
	@Test
	public void testStack() {

		assertEquals("Stack size should be 0 as new OutputStack object has
been created.", 0, testStack.sizeOf());
	}

	/**
	 * Tests that a new element can be pushed onto the stack.
	 */
	@Test
	public void testPush() {

		testStack.push("testElement1");
		testStack.push("testElement2");

		assertEquals(2, testStack.sizeOf());
		assertEquals("testElement2", testStack.getHead());

	}

	/**
	 * Tests that an element can be removed from the top of the stack.
	 */
	@Test
	public void testPop() {

		testStack.push("testElement1");
		testStack.push("testElement2");

		assertEquals(2, testStack.sizeOf());

		testStack.pop();
		assertEquals(1, testStack.sizeOf());

		testStack.pop();
		assertEquals(0, testStack.sizeOf());
```

```
	}

	/**
	 * Tests that the correct size of the stack can be returned.
	 */
	@Test
	public void testSizeOf() {

		assertEquals("Size should return 0, as the stack is new", 0,
testStack.sizeOf());

		testStack.push("testElement1");
		testStack.push("testElement2");

		assertEquals("Size should return 2, as there are now two elements in
the stack", 2, testStack.sizeOf());
	}

	/**
	 * Tests that the status of the stack being empty can be returned
correctly.
	 */
	@Test
	public void testIsEmpty() {

		assertTrue("Stack should be empty as it has been newly created",
testStack.isEmpty());

		testStack.push("testElement1");
		testStack.push("testElement2");

		assertFalse("Stack should not be empty as two elements are now in
the stack", testStack.isEmpty());

		testStack.pop();
		testStack.pop();

		assertTrue("Stack should be empty as all elements have been
removed.", testStack.isEmpty());

	}

/**
  * Tests that the element at the top of the stack can be returned correctly.
  */
 @Test
	public void testGetHead() {

		for (int i = 0; i < 5; i++) {

			testStack.push("testElement" + i);

		}

		for (int i = 4; i > 0; i--) {

			assertEquals("testElement" + i, testStack.getHead());
			testStack.pop();

		}
	}
```

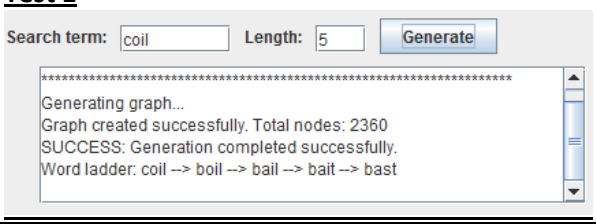
ReadFileTest

```
package aber.dcs.clg11.wordladder.tests;

import static org.junit.Assert.*;

import java.util.Vector;

import org.junit.Test;

import aber.dcs.clg11.wordladder.fileIO.ReadFile;

/**
 * JUnit test class for scrutinising the {@link
aber.dcs.clg11.wordladder.fileIO.ReadFile} class.
 *
 * @author Connor Goddard (clg11)
 */
public class ReadFileTest {

	ReadFile testReadFile = new ReadFile();

	/**
	 * Tests that a valid file can be parsed and red-in correctly.
	 */
	@Test
	public void testReadInValidFile() {

		Vector<String> testVector = testReadFile.readIn("testWords.txt");

		assertEquals("Vector size should be 5 to match number of words in
file.", 5, testVector.size());
	}

	/**
	 * Tests that an invalid file does not get read in, and null is returned
instead.
	 */
	@Test
	public void testReadInInvalidFile() {

		Vector<String> testVector = null;

		testVector = testReadFile.readIn("fake.txt");
		assertNull("Vector size should be 0 to match number of words in
file.", testVector);
	}

}
```

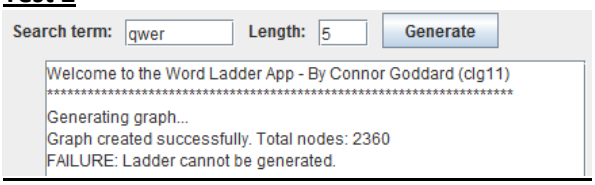
Appendix 1.3

These screenshots document the output generated by the application under testing set out in the application testing table (Figure 1.3).

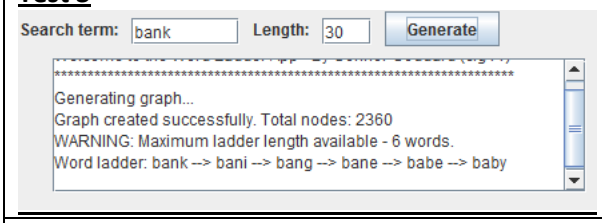
Test 1



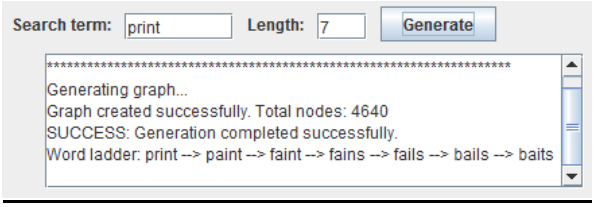
Test 2



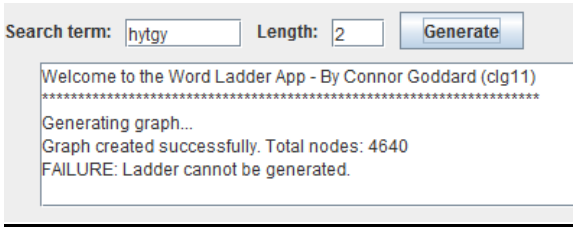
Test 3



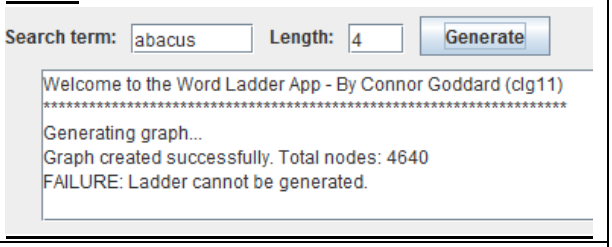
Test 4



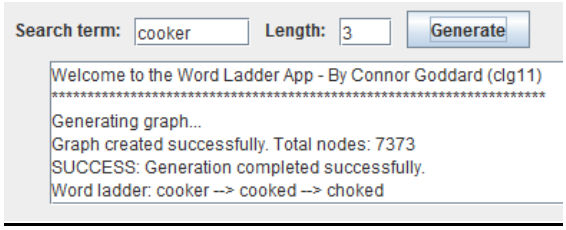
Test 5



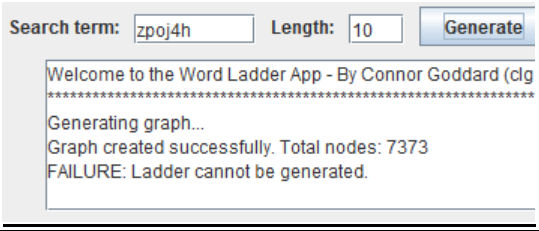
Test 6



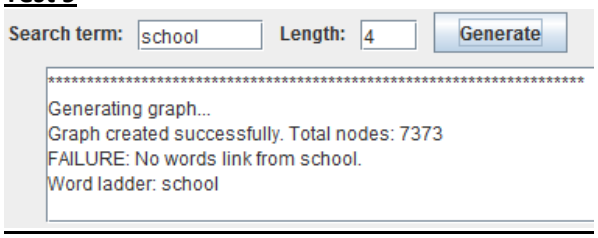
Test 7



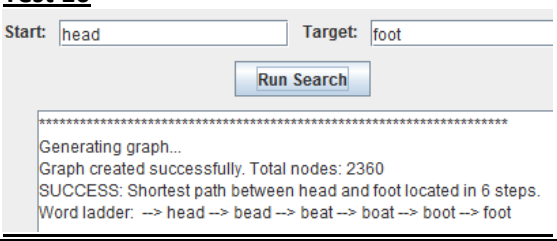
Test 8



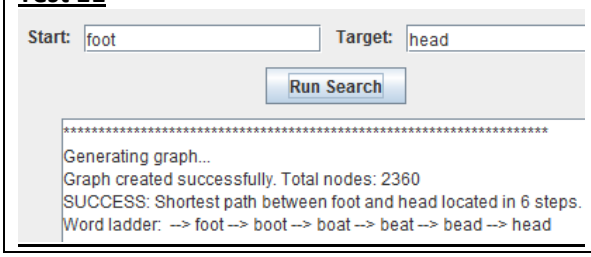
Test 9



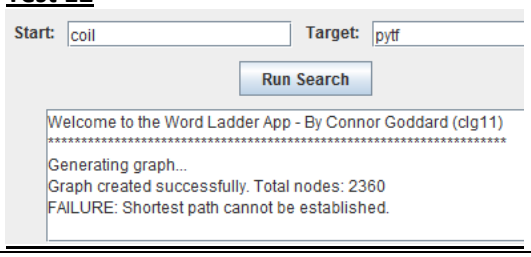
Test 10



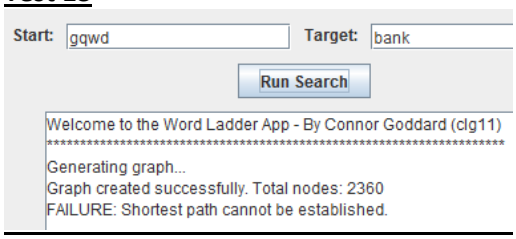
Test 11



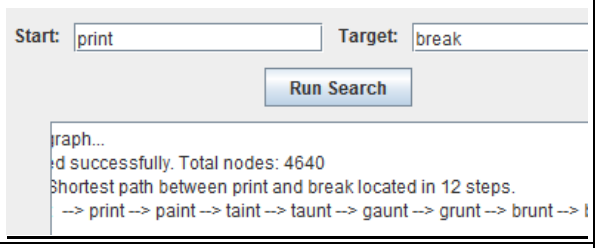
Test 12



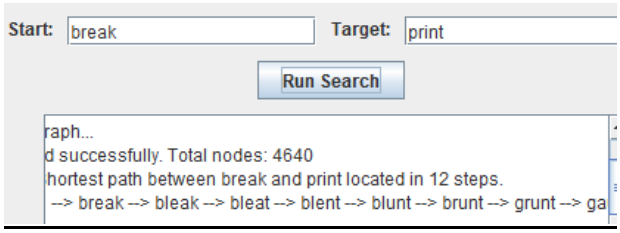
Test 13



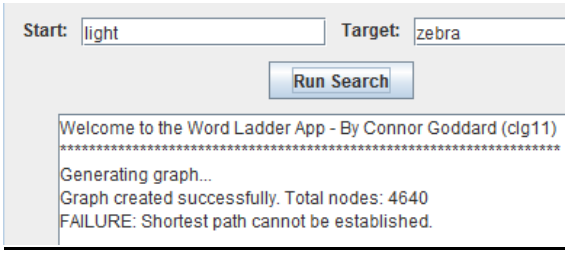
Test 14



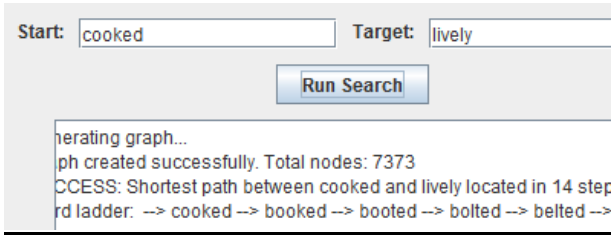
Test 15



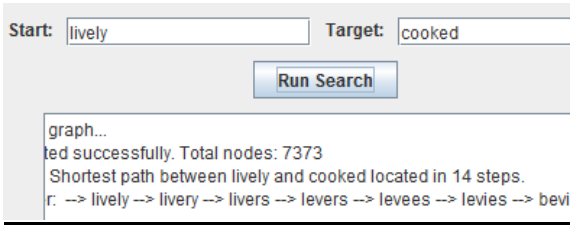
Test 16



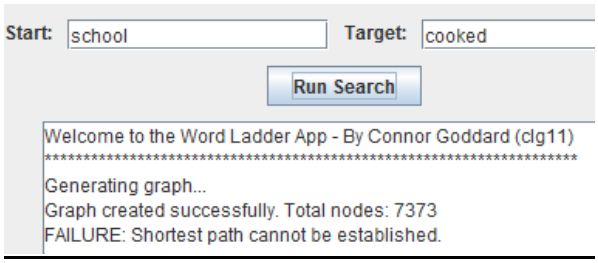
Test 17



Test 18



Test 19



Test 20