

University of Manchester

Dijkstra's Algorithm Learning Tool

Author:

Pedro Jorge

Supervisor:

Dr. Milan Mihajlovic

Third Year Project Report

BSc (Hons) Computer Science with Industrial Experience

School of Computer Science

May 3, 2016



The University of Manchester

Contents

Abstract	1
Acknowledgements	2
1. Introduction	3
1.1. Aims	3
1.2. Motivation	3
2. Background	4
2.1. Learning Tools	4
2.2. Graph Theory	4
2.2.1. Applications	5
2.2.2. Dijkstra's Algorithm	6
2.3. Previous Work	7
2.3.1. VisuAlgo: Single-Source Shortest Path	7
2.3.2. Dijkstra Shortest Path - Algorithm Visualizations	7
2.3.3. Example Networks1: Dijkstra's Algorithm for Shortest Route Problems	8
3. Design & Implementation	9
3.1. Design	9
3.1.1. Technical Approach	9
3.1.2. Functionality	10
3.1.3. GUI	11
3.2. Implementation	13
3.2.1. GUI	13
3.2.2. WalkThrough	13
3.2.2.1. Set WalkThrough	14
3.2.2.2. Interactive WalkThrough	15
3.2.3. Exercises	16
3.2.3.1. Multiple Choice Questions	16
3.2.3.2. Interactive Exercises	16
4. Testing and Evaluation	18
4.1. Testing	18
4.1.1. Functionality Testing	18
4.1.2. User Interface and Compatibility Testing	18
4.2. Evaluation	19
5. Conclusions	21
5.1. Planning	21
5.2. Future Work	22
5.3. Summary	22
Bibliography	23
A Functional and Non-functional Requirements	26
B Incremental Progression of Dijkstra's Algorithm	27
C User Feedback Survey	29

Abstract

Learning tools offer visually appealing representations of difficult problems in an attempt to demystify sophisticated concepts. This is especially true in fields such as computer science, where complex theoretical subjects, such as the study of Algorithms, are predominant.

Dijkstra's algorithm is one of the most fundamental algorithms in graph theory, nevertheless, understanding how it operates is not trivial. The learning tool presented here, aspires to provide simple and concise visualisations of the algorithm in an attempt to improve its perception.

In this report, in addition to a thorough description of the development process of such a tool, an analysis of its usefulness after it was endorsed as a learning resource for students in the School of Computer Science at the University of Manchester is conducted.

Acknowledgements

I would like to extend a specially thank you to my supervisor Dr. Milan Mihajlovic, for his continuous support and advice throughout the project. I also thank Dr. David Rydeheard and Dr. Ian Pratt-Hartmann for their valuable feedback, and without whose support, the learning tool would not be available to second year students as part of their COMP26120 learning resources.

Finally, a special word of thanks to my employers, friends and most notably to my family for their support throughout these years and without whom it would not have been possible for me to complete this degree.

Chapter 1

Introduction

Dijkstra's algorithm is one of the most significant algorithms in graph theory and it is commonly used for solving the single-source shortest path problem (*See section 2.2*). Although understanding the general principle of the algorithm and the pseudocode is quite simple, the inner workings of the associated data structure are somewhat perplexing. The learning tool developed as part of this project aims to demystify this process by providing clear and concise graphics to aid students in understanding and consolidating their knowledge of Dijkstra's algorithm.

1.1 Aims

The main goal of this project was to produce a learning tool capable of highlighting the algorithm's most important aspects when solving the single-source shortest path problem given a weighted undirected graph. Such capabilities include:

- Show the incremental progression of the algorithm through the graph, the associated data structure and the pseudocode.
- Allow users to test their knowledge of Dijkstra's algorithm by providing exercises.

The proposed solution that was developed during this project is a web application that is a learning tool for Dijkstra's algorithm. It sets itself apart from existing learning tools (*see chapter 2*) by providing a complete package that allows for a more interactive learning experience. Additionally, it has been made available for public use.

1.2 Motivation

The main motivation behind this project was to be able to develop a tool that would aid students enrolled in the *Algorithms and Imperative Programming (COMP26120)* [1] course unit to get a comprehensive understanding of how Dijkstra's algorithm works along with its associated data structure, enabling them to implement the algorithm themselves.

There have been many attempts at making a thorough learning tool that targets graph theory, more specifically the shortest path problem. These often look at a wide range of algorithms providing a couple of set examples for each one. As mentioned in section 1.1, the aim was to produce a learning tool with a higher degree of interactivity with both the learning and testing aspects, making it a more complete application.

Chapter 2

Background

In this chapter the fundamental aspects of learning tools and graph theory are first discussed. Existing tools are then analysed and compared to what this project was trying to achieve. This chapter covers the theoretical foundation upon which this project was developed.

2.1 Learning Tools

Learning tools, as the name indicates, are used to facilitate the learning process by employing progressive disclosure [2] in an attempt to present comprehensive material in an understandable manner. Invariably, they require a higher degree of interactivity and users are expected to learn by doing:

“Generally speaking, a learning tool is something a student uses to work through big ideas, concepts or processes while demonstrating his or her thinking, planning and/or decision-making” [3]

Graphical and dynamic representation of concepts has been proved [4] to be more successful in grasping attention and increasing comprehension of students than traditional learning materials. Furthermore, students who are actively engaged with the visualisation have consistently outperformed those who passively viewed them [5].

We are then able to infer that in addition to delivering learning material in a clear, concise and visually appealing way, learning tools also need to prompt user interaction, involving the student in the learning process, in order to avoid them adopting a passive attitude.

The challenge of discovering new ways to motivate students in active learning encouraged the development of a new kind of tool, sufficiently innovative, that not only delivers learning material in an understandable and seamless way but also encourages user interaction and involvement so that they are able to learn more effectively.

2.2 Graph Theory

In computer science, graph theory is concerned with the study of graphs and is a concept that was introduced by the Swiss mathematician Leonhard Euler. Graphs are used to represent relationships within a set of objects.

*“A graph is a collection of **vertices** (points) that are connected by **edges** (lines).” [6]*

Formally, a graph G is an ordered pair $G = (V, E)$ comprising a set of V (nodes) and a set E (edges) [7].

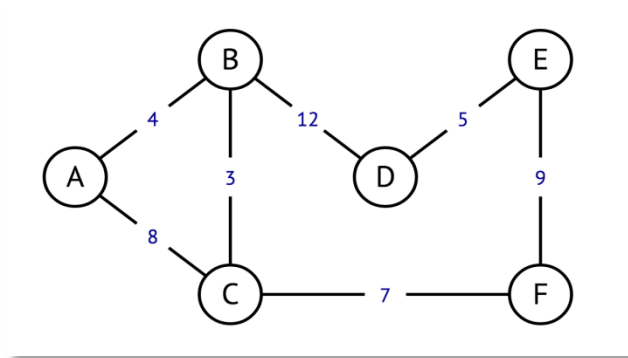


Figure 2.1: A weighted undirected graph with 6 vertices and 7 edges

There is a significant amount of vocabulary associated with graph theory, however, only the terms deemed necessary to the problem at hand are listed below.

Undirected graph: is a graph in which edges have no orientation, therefore the edge (x, y) is identical to the edge (y, x) . This means that the edges can be traversed in both directions.

Weighted graph: is a graph where edges have an associated numerical value called the weight. As an example, the weight might refer to the length of a route in the context of route planning applications.

Simple graph: is an unweighted, undirected graph containing no graph loops, an edge of a graph that joins a vertex to itself, or multiple edges, two or more edges connecting the same two vertices.

Path: is a finite sequence of edges which connect a sequence of vertices. In a weighted graph, the weight/cost/length of a path is the sum of the weights of the traversed edges.

Connectivity: if there is at least one path between all pairs of vertices, the graph is said to be connected.

2.2.1 Applications

In computer science, graphs are commonly used to represent data structures and to model relationships between entities. “The World Wide Web can be thought of as a directed graph, in which the vertices represent web pages, and the directed edges hyperlinks.” [8]. This rationale can be applied to many other fields such as telecommunication networks and biology. Algorithms such as Dijkstra’s help us to analyse, understand and optimise these data organizations.

2.2.2 Dijkstra's Algorithm

Provides a solution to the single-source shortest path problem in graph theory by using a greedy approach, it was devised by Edsger Dijkstra and published in 1959. The algorithm has various uses in diverse areas such as robot navigation, route planning applications and network routing protocols, most notably IS-IS [9] and OSPF [10]. For example, if we are trying to find the shortest route between one city and all the other cities within a county, we can represent each city as a vertex and the distances between them as edge weights. We can then resort to Dijkstra's algorithm to find the shortest path.

Shortest Path Problem

“Given a weighted graph $G = (V, E)$, and a source vertex s , find the minimum weighted path from s to every other vertex in G ” [11]

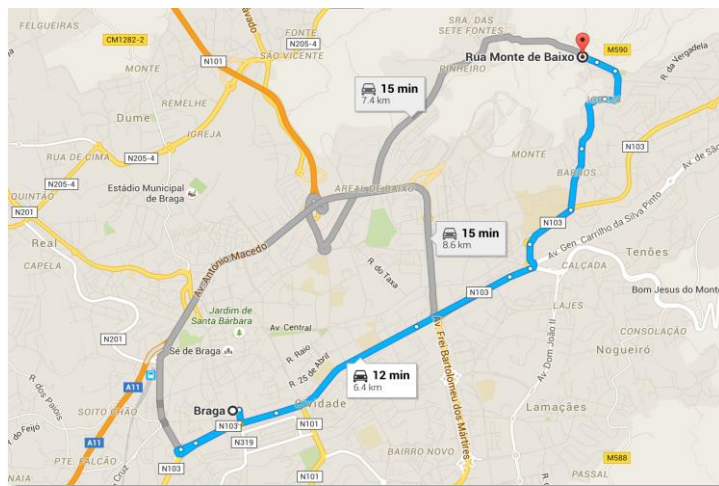


Figure 2.2: Google Maps Route Planner

The algorithm works as follows:

1. Initialize the source vertex distance to 0 and all the others to infinity.
2. Mark the source vertex as current and insert all the others in the unvisited set.
3. Store all the vertices in a *min-oriented* priority queue.
4. For the current vertex
 - a. Calculate *provisional* distances to all of its unvisited neighbours
 - b. Compare the *provisional* distances to the current distances and keep the smaller value
 - c. When all the neighbours' distances have been settled, the vertex is considered visited and is removed from the unvisited set.
 - d. If the priority queue is not empty, remove the next element and make it the current vertex.
5. When all vertices have been marked as visited and the priority queue is empty, the algorithm is finished.

For this project, the implementation of Dijkstra’s algorithm used in the “WalkThrough” sections (see Chapter 3), builds upon the pseudocode developed by Goodrich and Tamassia [12] that can be seen in Figure 2.3.

```

Algorithm DijkstraShortestPaths( $G, v$ ):
  Input: A simple undirected weighted graph  $G$  with nonnegative edge weights,
    and a distinguished vertex  $v$  of  $G$ 
  Output: A label  $D[u]$ , for each vertex  $u$  of  $G$ , such that  $D[u]$  is the distance from
     $v$  to  $u$  in  $G$ 
   $D[v] \leftarrow 0$ 
  for each vertex  $u \neq v$  of  $G$  do
     $D[u] \leftarrow +\infty$ 
  Let a priority queue  $Q$  contain all the vertices of  $G$  using the  $D$  labels as keys.
  while  $Q$  is not empty do
    {pull a new vertex  $u$  into the cloud}
     $u \leftarrow Q.\text{removeMin}()$ 
    for each vertex  $z$  adjacent to  $u$  such that  $z$  is in  $Q$  do
      {perform the relaxation procedure on edge  $(u, z)$ }
      if  $D[u] + w((u, z)) < D[z]$  then
         $D[z] \leftarrow D[u] + w((u, z))$ 
        Change to  $D[z]$  the key of vertex  $z$  in  $Q$ .
  return the label  $D[u]$  of each vertex  $u$ 

```

Figure 2.3: Pseudocode for Dijkstra’s algorithm for the single-source shortest path problem

2.3 Previous Work

There is an extensive list of tools, both web-based and standalone applications that enable the visualisation of the progression of Dijkstra’s algorithm, the one below is therefore by no means exhaustive.

2.3.1 VisuAlgo: Single-Source Shortest Path

VisuAlgo is a learning tool conceptualised in 2011 by Dr Steven Halim that aims to help students get a better understanding of data structures and algorithms [13]. It consists of a web application covering a wide range of topics, but in what pertains to the single-source shortest path problem it enables the visualisation of Dijkstra’s algorithm progression through a graph at a step-by-step level.

It allows users to create their own graph or select one out of a list of four graphs, however, it contains a few limitations, such as the fact that it does not allow manipulation of the graph, it only allows directed graphs, the pseudocode available is very basic and there is no display of the data structure.

2.3.2 Dijkstra Shortest Path - Algorithm Visualizations

This tool [14] is a web application developed by professor David Galles. It provides several set graphs of increasing complexity, allows the user to select a *source* vertex, whether the graph is directed or undirected and three different representations of the graph, “*logical*”, “*adjacency list*” and “*adjacency matrix*”. It shows a step-by-step progression of Dijkstra’s algorithm and it displays a table with information relating to the vertices that have been visited and the current distance to them.

However, this tool does not allow users to input their own graph or make changes to the set graphs, furthermore, it does not display a data structure and/or pseudocode progression.

2.3.3 Example Networks1: Dijkstra's Algorithm for Shortest Route Problems

This tool [15] provides a comprehensive explanation of the progression of Dijkstra's algorithm with good graphics and insightful comments. It is however, too informal for users wanting to gain knowledge pertaining to the technical aspects of the algorithm as it fails to show any pseudocode or data structure updates.

When comparing the learning tool developed for this project with the ones mentioned above, as it can be seen in Table 2.1, it is clear that it consists of a much more complete application covering the issue of explaining how the algorithm works by providing the necessary graphical display alongside pseudocode and data structure representation. It also provides means for users to test their knowledge by supplying multiple choice questions and interactive exercises.

Features	This Tool	VisuAlgo	Algorithms Visualizations	Example Networks1
Show Incremental Progression	✓	✓	✓	✓
Display of Graph	✓	✓	✓	✓
Display of Data Structure	✓	-	-	-
Display of Pseudocode	✓	✓	-	-
Display of Comments	✓	✓	-	✓
Allow Customisation of Set Graph	✓	-	-	-
Provide Multiple Set Graphs	-	✓	✓	-
Provide Multiple Representations of Graph	-	-	✓	-
Provide Directed and Undirected Graphs	-	-	✓	-
Allow Creation of Graph	✓	✓	-	-
Allow Choice of Source Node	✓	✓	✓	-
Exercises	✓	-	-	-

Table 2.1: Comparison of features present in Dijkstra's algorithm learning tools

Chapter 3

Design and Implementation

In this chapter an analysis of the design and implementation of the learning tool is conducted with the main focus being on the requirements gathering, the graphical user interface and the main functionality.

3.1 Design

When developing this learning tool, one of the main considerations was to build a flexible system that not only accommodated the needs of novice users, with some experience with graph theory, but little to no experience with Dijkstra's algorithm, but also more experienced users who desired to experiment with more complex graphs.

3.1.1 Technical Approach

One of the first decisions that needed to be made had to do with the technologies necessary to produce a successful learning tool. There were several aspects that impacted this decision but ultimately, the technologies needed to provide two keys properties:

- Smart and functional graphics
 - Clear and concise presentation of results, and visually appealing.
- Powerful user interface
 - Handle high volume of user input and interaction.

Initially there were two options: build a standalone application or a web application. After considerable and careful consideration, the development of a web application instead of a standalone application was chosen primarily due to convenience to the user, therefore, avoiding the need for installing software. There is also the added benefit of web applications having inherent cross-platform compatibility and HTML5 providing native support and handling of graphical content [16].

Once the decision to develop a web application instead of a standalone program was made, a careful analysis of the tool's functionality was necessary in order to pin down a programming language. Since it essentially consisted of a front-end application [17], JavaScript was the obvious choice since it is used to develop the client-side of 14 of the 15 most popular websites [18].

Additionally, JavaScript has an extensive number of libraries (*discussed in detail in section 3.2.1*) that provide excellent support when it comes to graphical manipulation on web sites and handling of complex processing.

3.1.2 Functionality

There are a vast number of features that have been implemented as part of this learning tool. Ultimately, they had to combine in order to provide **two** that are crucial and unique pieces of functionality:

- **The WalkThrough Section**
 - Show incremental progression of the algorithm over graph, heap and pseudocode.
- **The Exercises Section**
 - Provide test questions.

One of the key intentions, which also relates to the effectiveness of learning tools, and was clearly a focus point throughout the project, was to make the tool as interactive and flexible as possible. To this end, additional functionality was thought of:

- **The Set WalkThrough**
 - Ability to update edge weights and choose source node of set graph.
- **The Interactive WalkThrough**
 - Ability to create own graph.
- **Multiple Choice Questions (MCQs)**
 - With appropriate answer checking.
- **Interactive Exercises**
 - Manually progress through the algorithm, by updating graph and heap
 - Appropriate answer checking.
 - Give hints if requested.
 - The premise was to encourage users to test their knowledge by performing updates to the graph and the heap. They are essentially executing the step-by-step incremental progression of Dijkstra's algorithm manually.

The functional and non-functional requirements can be seen in Appendix A.

3.1.3 GUI

For the “WalkThrough Section”, the graphical user interface had to display four key elements: A graph, the data structure, pseudocode and comments explaining each step. Three initial designs were considered as can be seen in Figure 3.1.

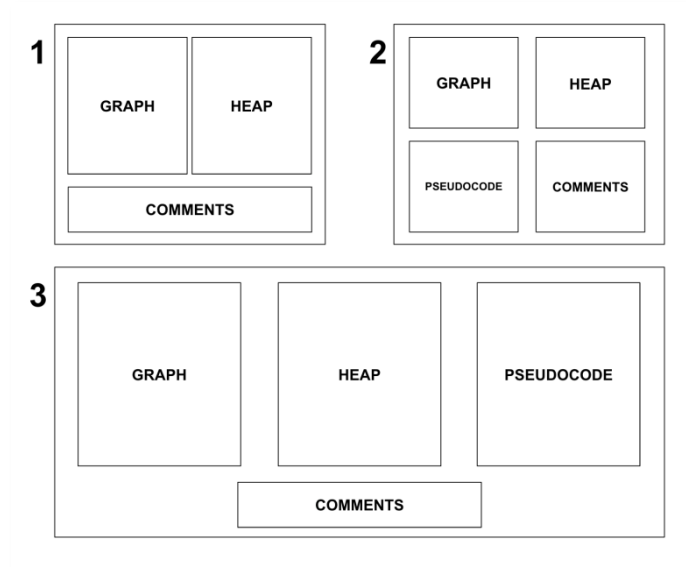
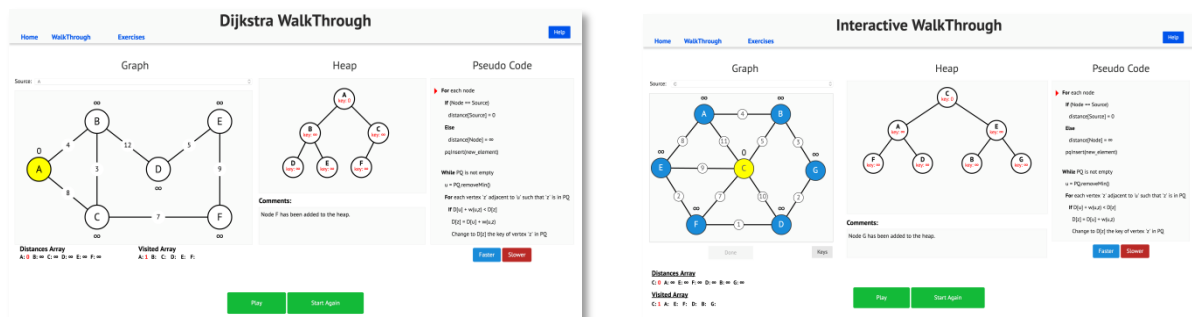


Figure 3.1: Initial designs for the “WalkThrough Section”

The decision to implement the third option stemmed from the fact that it allowed for a more seamless experience by facilitating shifting attention between elements and was also the one that received the most positive feedback. Both the “Set WalkThrough” and the “Interactive WalkThrough” (Figure 3.2) follow the same design, the difference is that the latter displays an empty canvas where users can create their own graph, instead of a set graph.

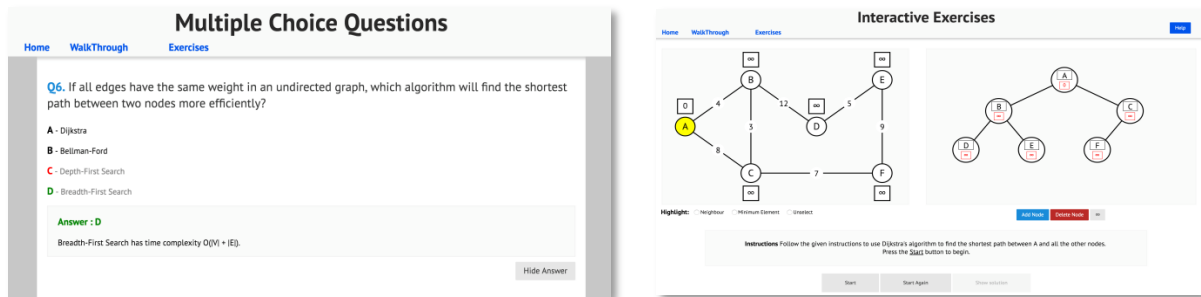


(a) “Set WalkThrough”

(b) “Interactive WalkThrough”

Figure 3.2: Final GUIs for the “WalkThrough Section”

For the “Exercises Section”, there are two separate parts that needed to be addressed, the multiple choice questions that cover book work and the interactive exercises. For the MCQs, the essential aspects concerning design had to do with the feedback given to the user once a question was answered. When a user selects an answer they obtain the result immediately, whether it is correct (*green*) or incorrect (*red*), if they wish to see the answer and its respective explanation they are able to do so by pressing a button on the bottom right of the question.



(a) Multiple Choice Questions

(b) “Simple Interactive Exercise”

Figure 3.3: Final GUIs for the “Exercises” section

For the “Interactive Exercises”, the initial plan consisted of displaying two empty HTML canvases side-by-side with the layout similar to the one displayed in Figure 3.3 (b), with instructions that requested that the user created their own graph and heap. This initial design was altered to the one seen on Figure 3.3 (b) because most user feedback inferred that there was too much wasted time on aspects that were not central to understanding how the algorithm worked. The new design was well received and allows the user to focus on the essential aspects: understanding the inner workings of the data structure and how the algorithm progresses through the graph in an interactive and straightforward manner. The graph and the heap are the essential elements with instructions being displayed below. The user can also use several buttons (*discussed in detail in section 3.2*) with the key ones being the “next step”, show solution, and the ones related to edits that can be performed to the graph and to the heap.

3.2 Implementation

The development of the learning tool consisted of building four main sections: the set walkthrough, the interactive walkthrough, the multiple choice questions and the interactive exercises. There was also a great concern with the look and usability of the GUI.

3.2.1 GUI

The basic functionality of the tool, such as the actions that take place when the buttons are pressed, for example, the drop down list to select the source node or the play button, is achieved by using JavaScript's built in functionality. Scalable Vector Graphics (SVG) [19] were used to produce the graphs and the heaps, and their manipulation is possible through the use of an external library, D3.js [20]. jQuery [21] is also used for Document Object Model (DOM) manipulation and jQuery UI [22] is used for visual effects such as displaying the help message or the keys to create the graph. A CSS library, Pure.CSS [23], was also used to give a modern look to the application. The external libraries are vital in what pertains to the display and dynamic behaviour of the graphics, namely the graph, the heap and the pseudocode.

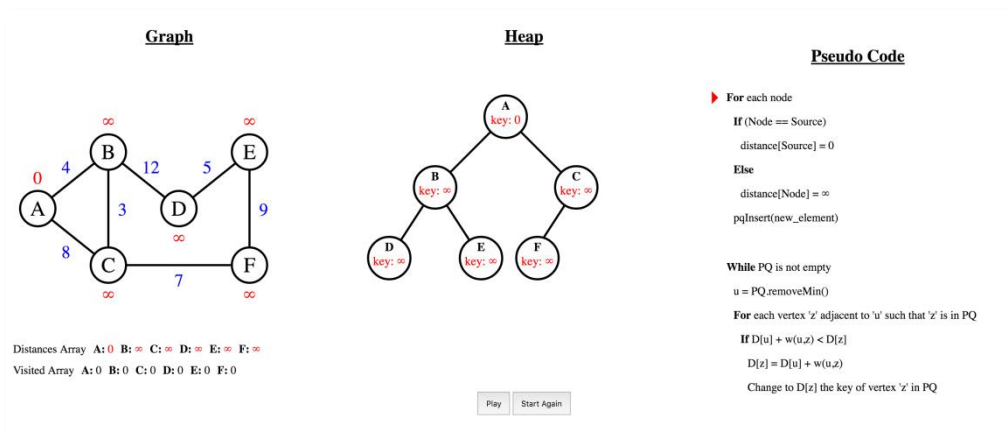


Figure 3.4: Original Version of the “Simple WalkThrough”

3.2.2 WalkThrough

For the WalkThrough parts of the learning tool there are three main components that needed to be developed: the implementation of Dijkstra's Algorithm, user input and the manipulation of the graph, and the display of the heap.

The implementation of Dijkstra's algorithm follows that of the one shown by Figure 2.3 and is modified slightly to handle a defined data structure (see Figure 3.5) as input and to call functions responsible for updating the heap, the comments and the pseudocode when required. To increase performance, the heap is pre-constructed and its elements are hidden at the start, once the progression is started, then each element is made visible incrementally. The progression of the algorithm only works in a step-by-step fashion and it does not allow for an automatic simulation, this was a clear design choice as it fostered the principles detailed in section 2.1. However, the users have access to buttons that can change the speed at which each step is executed in a manner that fits their needs.

```

// Node Object
function Node (label, index, neighbours, previous) {
  this.label = label;
  this.index = index;
  this.neighbours = neighbours;
  this.previous = previous;
} // Node

// Edges Object
function Edge (source, end, weight) {
  this.source = source;
  this.end = end;
  this.weight = weight;
} // Edge

```

Figure 3.5: Graph data structure

3.2.2.1 Set WalkThrough

Several designs and functionalities were considered for this version of the WalkThrough, the first version consisted of a fixed graph where all the user could alter was the starting node. However, after careful consideration and analysis of user feedback, the option to change the edge weights was included so that users can visualise different progressions using the same graph.

In terms of development, the main concern is to deal with user input, graph and heap graphical updates and running Dijkstra's algorithm. Once the user presses play, the graph is constructed by using jQuery to obtain the edge weight since, in this case, all the other information can be precomputed. The graph is then input to a *Dijkstra function* that runs through the algorithm and when necessary calls functions that update the graph, heap and pseudocode. Figure 3.6 describes the functional model. And Figure 3.7 is an example of a function within the “Graphical Dynamics” block, and is called when a minimum element's neighbour distance has been calculated and the graph needs to be updated accordingly. A full progression of the “Set WalkThrough” can be viewed in Appendix B.

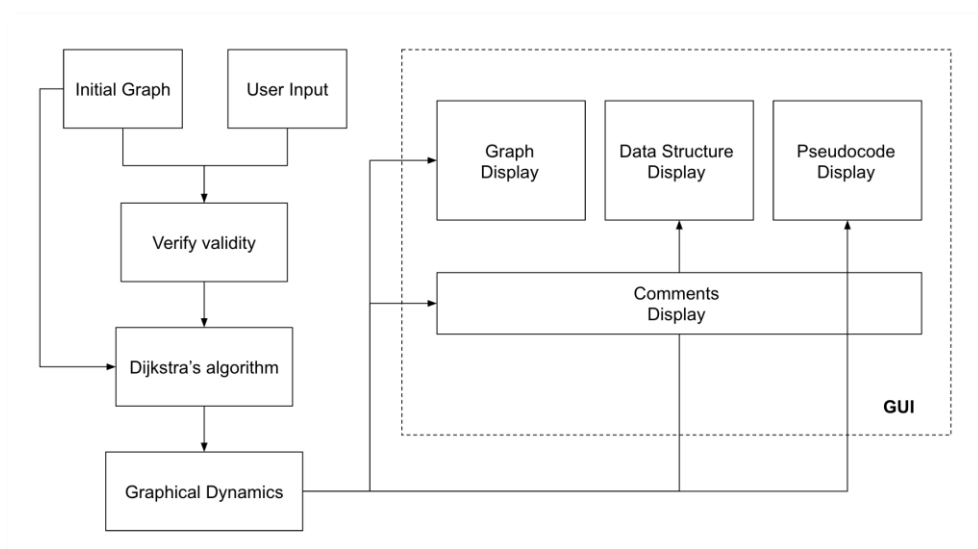


Figure 3.6: Functional block diagram for the “Set WalkThrough”


```
function paintNeighbours_confirm (index, result) {
  // Output the distance
  setTimeout(function() {
    d3.select(d3.select("#distance"+index)[0][0].parentNode).attr("class", "node neighbour");
    document.getElementById("distance"+index).innerHTML = result;
    document.getElementById("distance_array"+index).innerHTML = result;
  }, delay);
  delay += increment;
} // paintNeighbours_confirm
```

Figure 3.7: “Paint Neighbours” function for the “WalkThrough Section”

3.2.2.2 Interactive WalkThrough

This section has essentially the same functionality as the one covered previously with two distinctions: users have the ability to input their own graph and the ability to start again using the same graph. In order to allow the graph to be created and manipulated dynamically, I devised a set of keys that grant such actions (see Figure 3.8). All of these commands require complex operations in order to update the graph data structure.

In order to create a node, the coordinates of the mouse pointer need to be captured, a SVG circle object is created in its place, a label and index are assigned and are inputted into the graph data structure. When a node label is edited, the new input is captured, a verification is performed to ensure that there are no other nodes with the same label or that the label is not left empty. Consequently, the data structure is updated.

When an edge is created, both the index and label of the source and target nodes are recorded, a verification is performed to ensure the edge is not a duplicate, the components of the graph data structure concerning the edges are updated and a SVG path object is created connecting the two nodes. When an edge is deleted, the graph data structure gets updated and the SVG path deleted. When a node is deleted, more operations are required, the SVG circle is deleted but also the SVG paths that connect to it. Furthermore, the graph data structure is updated by removing the concerned node and any edges that have the deleted node as either the source or the target.

Once the user is satisfied with the graph and proceeds to the next step, several verifications are required in order to ensure that the graph is valid. These include checking that the edge weights are not left empty, that the graph exists and that the graph is connected, this is verified by running a simpler version of Dijkstra’s algorithm that certifies that all the nodes can be reached. If all verifications are successful, edits to the graph are disabled, with only dragging nodes being allowed, and the user can select the source node and begin the progression.

At the end of the progression, if the user wishes to re-use the graph, the graph data structure is kept unaltered and the canvas is the only part of the application that does not get updated, all the other variables are set to their initial values. The user is then able to edit the saved graph if they wish and proceed as before.

Keys	
Draw Node	SHIFT + CLICK
Delete Node	D + CLICK (NODE)
Draw Edge	E + CLICK (NODE)
Edit Node Label	W + CLICK (NODE)
Delete Edge	R + CLICK (EDGE)

Figure 3.8: Keys for graph creation

3.2.3 Exercises

This section of the learning tool aims to complement the “WalkThrough” by encouraging users to apply and consolidate the knowledge they have acquired.

3.2.3.1 Multiple Choice Questions

The main concern with this section was to devise appropriate questions that cover fundamental topics of graph theory such as time complexity, comparison of different graph traversal algorithms, and the basics of Dijkstra’s algorithm (*paradigm and progression*). The purpose of these questions was to allow students to gain an understanding of how Dijkstra’s algorithm fits within the realm of graph theory.

Regarding the functionality, jQuery is used to highlight the selected answer converting it into green or red if the answer is correct or incorrect respectively, and D3.js is used for the “Show Answer”/“Hide Answer” button that displays an explanation as it can be seen in Figure 3.3 (a).

3.2.3.2 Interactive Exercises

This section focused on allowing users to carry out Dijkstra’s algorithm manually, this can be achieved by various edits to the graph and the heap. The initial state of the exercises can be seen in Figure 3.3 (b). The user is able to perform four actions:

- Colour the graph
 - With the “Neighbour”, “Minimum Element” or “Unselect” options.
- Change the distances of each node on the graph.
- Change the key and label of each element on the heap.
- Add and delete nodes from the heap.

From a development point of view the challenge is to check if the answer is correct, as handling the animations (*add/remove nodes and colour graph*) and user input (*graph weights and heap labels and keys*) is quite straightforward with jQuery.

For the graph, an array stores the state of the graph (*node, distance, edge state*) for each step and then compares what the user has inputted with the correct array elements for that step. If an error occurs at this stage, a hint message will be prepared stating that there is a problem with the graph. At stages where the heap needs to be checked, this process will happen after the graph is checked. Here three processes take place, the heap size is checked, and this is done by matching the number of existing nodes with a dynamic variable that gets reduced by one at each step. The second verification pertains to the heap property, again an array with the state of the priority queue at each step is compared with what the user has inputted. Finally, the order of the elements in the heap is verified. If an error occurs at this stage, a hint message is produced stating that there is a problem with the heap. The functional models for the graph and heap verification procedure can be viewed in Figure 3.9 and Figure 3.10 respectively. An example of a function call to these procedures can be seen in the code snippet in Figure 3.11.

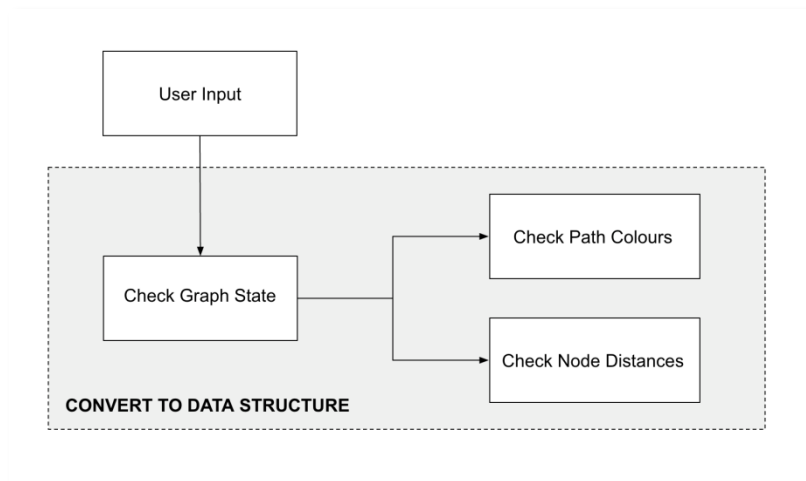


Figure 3.9: Functional Block Diagram for Checking the Graph State

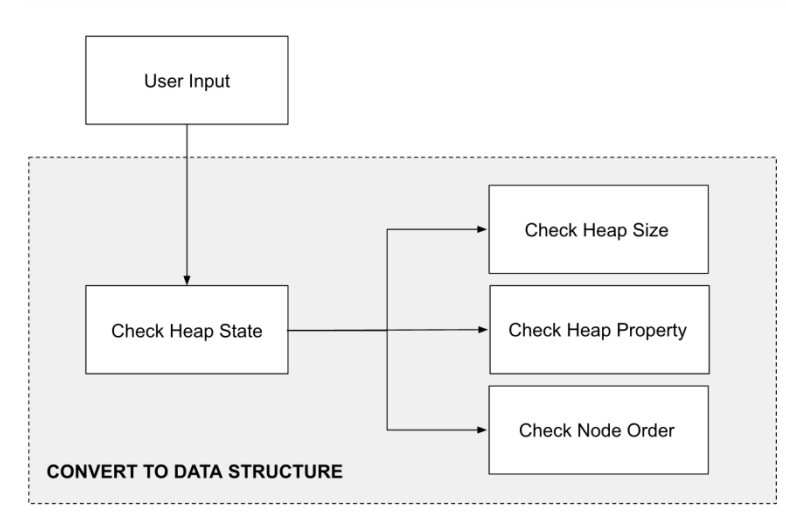


Figure 3.10: Functional Block Diagram for Checking the Heap State

```

// STEP 5 - Select neighbours & Check heap
else if (next_step == 5) {
    if (!checkGraphState(graph_check_step)) { incorrectAnswer(3); return false; }
    var PQ = [{"C", "7"}, {"D", "16"}, {"E", "∞"}, {"F", "∞"}];
    if (!checkHeap(PQ, 4)) { incorrectAnswer(1); return false; }
    graph_check_step++; return true;
} // if
    
```

Figure 3.11: Code Snippet for “Interactive Exercises” Step Verification

Chapter 4

Testing and Evaluation

This chapter is concerned with describing the testing paradigms utilized, additionally, an analysis of the user feedback and the obtained results is conducted.

4.1 Testing

The project consists of a web application meaning that it needs to be as adaptable as possible because there is an inherent requirement for it to be accessible on various browsers. The graphical components, including animations, were not testable by automated means, nevertheless, there are two main aspects of the learning tool that needed to be tested:

4.1.1 Functionality Testing

Test Driven Development [24] and Unit tests were the main testing methodologies used to ensure that the individual components of the web application worked as expected. As an example, in the “WalkThrough Section” the main testing that had to be done concerned the implementation of Dijkstra’s algorithm. Before any of the graphical functionalities were added, there was a need to guarantee that the algorithm was implemented correctly, this was done by developing unit tests for the respective functions. These focused on running the algorithm with graphs that covered most edge cases such as all edges having the same weight and two paths with the same distance to a node. Each step was outputted to the JavaScript Console with an evaluation stating whether it passed or failed.

In regards to functionality testing, four main outcomes needed to be ensured [25]:

1. Functions need to calculate correct output.
2. Animations need to be smooth and consistent.
3. Components should be stable and reliable.
4. Functions and scripts need to possess cross-browser compatibility.

4.1.2 User Interface and Compatibility Testing

To ensure maximum cross-browser compatibility, the technologies used: SVG and the JavaScript libraries, are all supported by the major web browsers. The full application is accessible by browser versions later than Firefox 40.0.3, Chrome 45.0 and Safari 9.1. These correspond to 3 of the top 4 most used web browsers [26]. Web browser versions earlier than the aforementioned, may not fully display the correct fonts and graphics and some of the functionality such as node creation or drag-and-drop may not be available.

The minimum comfortable display size is 1620x720 (*below this level, not all elements: graph, heap and pseudocode are fully visible*) with the optimal display size being 1680x1050. Navigation was also tested by ensuring that all links are pointed at the correspondent web pages. To enhance the user-friendliness of the application, several techniques were used starting with the minimalistic design and the concept of progressive disclosure [2]. For users less familiar with the learning tool, there is a

help button on the top right corner of each page that provides a detailed description of the functionality of that particular section.

In regards to usability testing, three main outcomes needed to be ensured [25]:

1. Seamless Navigation.
2. User Friendliness.
3. Cross-Browser Compatibility.

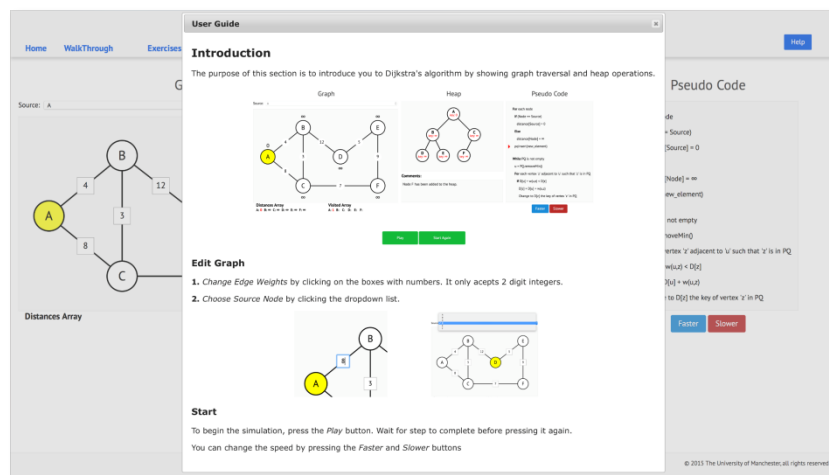


Figure 4.1: Help message for “Set WalkThrough”

4.2 Evaluation

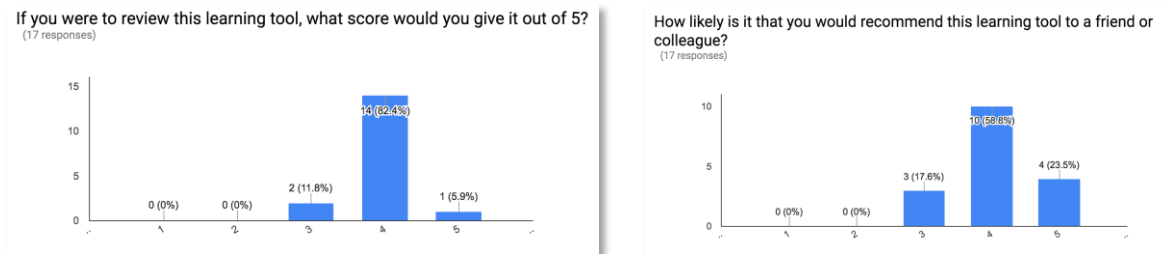
As mentioned previously, this learning tool was made available as a learning resource for students undertaking the *COMP26120 - Algorithms and Imperative Programming* course unit that aims to “Provide a practical introduction to computational thinking and developing algorithmic literacy” [1].

The tool was first reviewed and consequently approved by the course lecturers Dr. David Rydeheard, Dr. Ian Pratt-Hartmann and Dr. Milan Mihajlovic and was then published on the course website where students were encouraged to use it for both coursework and revision. To complete the evaluation of the project, a survey to gather user feedback was conducted and can be viewed in Appendix C.

The survey aimed to establish the target population (field of study related to Computer Science and familiarity with Dijkstra’s algorithm), the usefulness of each section, the overall experience with the tool (recommend to friends and overall score) and whether users felt anything was missing or could be improved.

The survey was advertised in online forums, more specifically at the *Computer Science Manchester University* [27], *Computer Science* [28] and *School of Computer Science, UoM - UG Students* [29] Facebook groups. Overall, 17 responses were obtained which should enable us to conclude that all usability problems in the design have been found [30].

The results collected are very promising, with 88.3% of users awarding the learning tool an overall score of 4 or higher out of 5 when questioned “If you were to review this learning tool, what score would you give it out of 5?” and 82.3% of users choosing 4 or higher out of 5 when questioned “How likely is it that you would recommend this learning tool to a friend or colleague?”. The “WalkThrough” section was found to be the most useful, with 64.7% of users selecting the “Set WalkThrough” and the “Interactive WalkThrough” when asked “What section(s) of the learning tool did you find the most useful in aiding your understanding of Dijkstra's Algorithm?”.

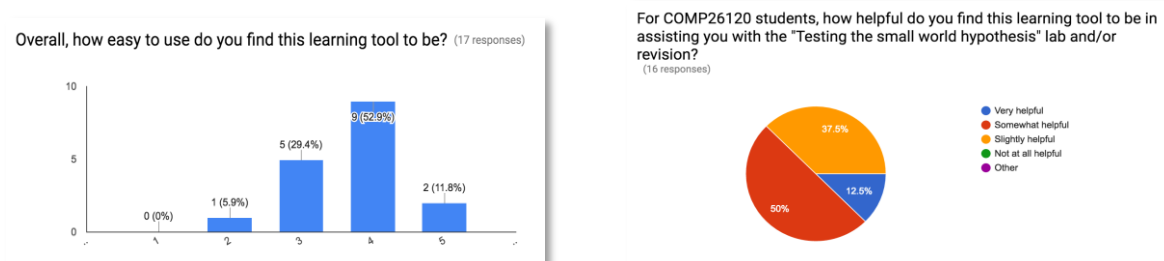


(a) Overall Result

(b) Recommendations

Figure 4.2: “Overall Experience” Feedback

The main concerns pertain to usability, with 35.3% of users answering 3 or less out of 5 when questioned “Overall, how easy to use do you find this learning tool to be?” and the relevance of the learning tool in the context of the COMP26120 course unit, with 37.5% of users finding the tool “slightly helpful” when questioned “For COMP26120 students, how helpful do you find this learning tool to be in assisting you with the ‘Testing the small world hypothesis’ lab and/or revision?”. The full responses to all the questions can be viewed in Appendix C.



(a) Ease of Use

(b) Helpfulness with Coursework

Figure 4.3: Usability Feedback

We can deduce from the obtained answers that the main goal of this project has hence been substantially fulfilled by the fact that it was made available to students with the backing from the course lecturers and that it received favourable feedback from its primary users.

Chapter 5

Conclusions

This chapter describes the initial planning and how changes were made to accommodate new requirements, as well as, a brief discussion is held concerning the ways in which the learning tool could be improved and finally a conclusion is drawn.

5.1 Planning

An initial plan was conceived at the beginning of the project with the main elements of the learning tool being a “WalkThrough” section with 3-5 set graphs of increasing complexity and an “Exercises” section that included 2-3 exercises where the user had to create their own graph and heap using a drag-and-drop system and then keep updating them as necessary.

In terms of the development process, I used the short iterations agile methodology [31] to plan the functionality that needed to be gradually developed. By using one week iterations, I was able to implement a list of features each week and then analyse my progress and what needed to be done next. This practice allowed me to realise that instead of creating 2-3 set graphs, I could create one set graph that can be updated, for novice users, and then allow more experienced users to create their own graph.

In regards to the “Exercises” phase of development, there was a complete overhaul of the initial plan, this was in part because I felt the need to include a section that focused on graph theory as a whole and the theoretical aspects of Dijkstra’s algorithm, but also based on user feedback. For the “Interactive Exercises”, the initial plan of requesting that the user created their own graph and heap, required too much wasted time on aspects that were not central to understanding how the algorithm worked. Therefore, I designed and implemented a system that allows the user to focus on the essential aspects: understanding the inner working of the data structure and how the algorithm progresses through the graph in an interactive and straightforward manner. Having gathered experience with manipulating SVGs and handling graph and heap display updates when developing the “WalkThrough”, the time needed to implement this section was significantly reduced, allowing me to further include a more complex exercise as seen in Figure 5.1.

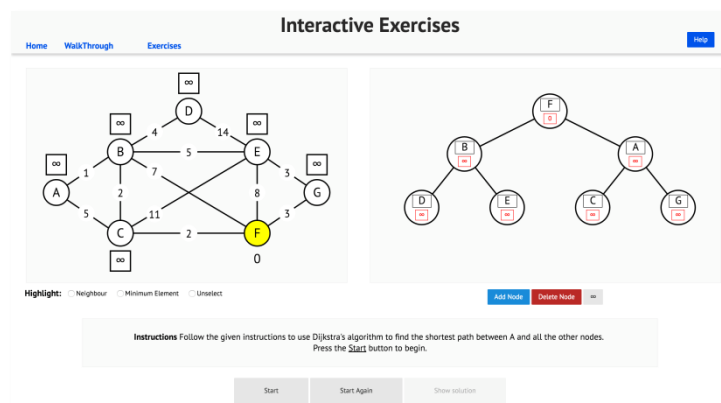


Figure 5.1: GUI for the “Complex Interactive Exercise”

5.2 Future Work

There are several improvements that can be made, and focusing on user feedback, the following features could be added:

- Provide more detailed comments in regards to what is happening at each step
 - Focus on explaining graph operations.
- Break it down into more steps
 - Possibly include an option to select level of detail (*simple* -> *thorough*) of steps at the beginning of the progression.
- Include test feature for Multiple Choice Questions
 - Conceive 3-5 different tests of increasing difficulty where user could receive feedback on performance.
- Develop another type of exercise that does not require users to complete all the steps and rather runs the progression and requests user input at some of them, for example, “input updated edge weight for node C” or “input new root for heap”.
- Improve the layout.

However, the biggest development that could be made is to convert the application into a framework for graph traversal algorithms that provides the main functionality and all that needs to be supplied are the algorithms, comments and answers. All the graphics, animations and functionality would be provided by the framework. This would allow for the creation of a truly *complete* tool that could perhaps be further extended to include other types of algorithms.

5.3 Summary

Overall, this project was able to achieve the aims it set out to accomplish and delivers key pieces of functionality:

1. Show incremental progression of Dijkstra’s algorithm in a clear and concise way.
2. Provide clarification of data structure evolution as algorithm progresses.
3. Allow users to input their own graph.
4. Allow users to test their knowledge of graph theory.
5. Allow users to manually carry out Dijkstra’s algorithm.

The ultimate goal of this project was for the learning tool to be made available to students as a learning resource for the *COMP26120 - Algorithms and Imperative Programming* course unit, this has been fulfilled, and together with the positive feedback obtained from both lecturers and students, I am confident that this project constitutes a success.

Bibliography

- [1] COMP26120 (2015). *COMP26120 - Algorithms and Imperative Programming (2015/16)*.
URL: <http://studentnet.cs.manchester.ac.uk/ugt/2015/COMP26120/> (visited on 29/04/2016).
- [2] Barone-Nugent, B. (2014). *Progressive Content for Progressive Reduction*.
URL: www.uxbooth.com/articles/progressive-content/ (visited on 30/04/2016).
- [3] PCAE. *Learning Tool Definition & Checklist*.
URL: http://www.mnartseducation.org/docs/03/_pdf/03_01.pdf (visited on 13/04/2016).
- [4] Gilakjani, A., et al. (2011). *The Effect of Multimodal Learning Models on Language Teaching and Learning*. In: Theory and Practice in Language Studies, Vol. 1, No. 10, pp. 1321-1327. ISSN: 1799-2591. URL: <http://www.academypublication.com/issues/past/tpls/vol01/10/07.pdf> (visited on 15/04/2016).
- [5] Hundhausen, C. (2002). *Integrating algorithm visualization technology into an undergraduate algorithms course: ethnographic studies of a social constructivist approach*. In: Computers & Education 39, pp.237-260.
URL:<http://www.sciencedirect.com/science/article/pii/S0360131502000441?np=y> (visited on 27/04/2016).
- [6] Adamchik, V. (2005). *Graph Theory*.
URL: https://www.cs.cmu.edu/~adamchik/21-127/lectures/graphs_1_print.pdf (visited on 15/04/2016).
- [7] Buell, D. (2013). *A first look at graphs*. In: Data Structures Using Java, Chapter 9, pg. 204.
- [8] Zhou, D., et al. (2005). *Learning from Labeled and Unlabeled Data on a Directed Graph*. URL: research.microsoft.com/en-us/um/people/denzho/papers/LLUD.pdf (visited on 15/04/2016).
- [9] IS-IS Protocol. *Intermediate System-to-Intermediate System Protocol*.
URL:www.cisco.com/en/US/products/ps6599/products_white_paper09186a00800a3e6f.shtml (visited on 19/04/2016).
- [10] OSPF (2005). *OSPF Design Guide*.
URL:www.cisco.com/c/en/us/support/docs/ip/open-shortest-path-first-ospf/7039-1.html (visited on 19/04/2016).
- [11] Erickson, J. (2014). *Lecture 21: Shortest Paths*.
URL:web.engr.illinois.edu/~jeffe/teaching/algorithms/notes/21-sssp.pdf (visited on 20/04/2016).
- [12] Goodrich, M. T., Tamassia, R. (2001). *Algorithm Design: Foundations, Analysis, and Internet Examples*. London: John Wiley & Sons. p343.

- [13] VisuAlgo (2001). *VisuAlgo: Single-Source Shortest Path*.
URL: <http://visualgo.net/sssp.html> (visited on 18/04/2016).
- [14] Galles, D. (2001). *Dijkstra Shortest Path - Algorithm Visualizations*.
URL: <https://www.cs.usfca.edu/~galles/visualization/Dijkstra.html> (visited on 18/04/2016).
- [15] Kinahan, K. and Pryor, J. (2007). *Example Networks1: Dijkstra's Algorithm for Shortest Route Problems*.
URL: <http://optlab-server.sce.carleton.ca/POAnimations2007/DijkstrasAlgo.html> (visited on 18/04/2016).
- [16] HTML5 (2014). *HTML5 Differences from HTML4*.
URL: <https://www.w3.org/TR/html5-diff/> (visited on 25/04/2016).
- [17] Front-end (2006). *What is front-end? - Definition from WhatIs.com*.
URL: <http://whatis.techtarget.com/definition/front-end> (visited on 25/04/2016).
- [18] Top Sites in United Kingdom (2016). Alexa - *Top Sites in United Kingdom*
URL: <http://www.alexa.com/topsites/countries/GB> (visited on 25/04/2016).
- [19] SVG (2016). *Scalable Vector Graphics (SVG)*.
URL: <https://developer.mozilla.org/en-US/docs/Web/SVG> (visited on 25/04/2016).
- [20] D3.js (2015). *D3.js - Data-Driven Documents*. URL: <https://d3js.org> (visited on 25/04/2016).
- [21] jQuery (2016). *jQuery - write less, do more*. URL: <https://jquery.com> (visited on 25/04/2016).
- [22] jQuery UI (2016). *jQuery - user interface*. URL: <https://jqueryui.com> (visited on 25/04/2016).
- [23] Pure (2014). *Pure.css*. URL: purecss.io (visited on 25/04/2016).
- [24] Palermo, J. (2006). *Guidelines for Test-Driven Development*.
URL: [https://msdn.microsoft.com/en-us/library/aa730844\(v=vs.80\).aspx](https://msdn.microsoft.com/en-us/library/aa730844(v=vs.80).aspx) (visited on 30/04/2016).
- [25] Ghoshal, A. (2014). *Web application testing: How to get the most out of your sessions*. URL: <http://thenextweb.com/apps/2013/11/28/guide-testing-web-app-steps-approach-testing-get-sessions/> (visited on 30/04/2016).
- [26] StatCounter Global Stats. (2016). *Top 5 Desktop, Tablet & Console Browsers from Apr 2015 to Apr 2016*. URL: gs.statcounter.com/#browser-ww-monthly-201504-201604-bar (visited on 30/04/2016).
- [27] *Computer Science Manchester University*. URL: <https://www.facebook.com/groups/ComputerScienceManchesterUniversity/> (visited on 25/04/2016).

- [28] *Computer Science*. URL: <https://www.facebook.com/groups/UnofficialUoMCS/> (visited on 25/04/2016).
- [29] *School of Computer Science, UoM - UG Students*. URL: <https://www.facebook.com/groups/CSUGMan/> (visited on 25/04/2016).
- [30] Nielsen, J. (2000). *Why You Only Need to Test with 5 Users*. URL: <https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/> (visited on 29/04/2016).
- [31] Leffingwell, D. (2016). *Iterations Abstract*. URL: <http://www.scaledagileframework.com/iterations/> (visited on 30/04/2016).

Appendix A

Functional and Non-functional Requirements

Functional requirements

A functional requirement describes the behaviour that a software system should exhibit.

- The Learning tool should show the incremental progression of the graph
- The Learning tool should show the incremental progression of the data structure
- The Learning tool should show the incremental progression of the pseudo-code
- The Learning tool should allow the user to progress in a step-by-step manner
- The Learning tool should allow the user to create/edit graph
- The Learning tool should allow the user to reuse a graph
- The Learning tool should allow the user to control the speed of the progression
- The Learning tool should provide insightful comments at each step of the progression
- The Learning tool should provide questions regarding theoretical aspects of Dijkstra's algorithm
- The Learning tool should provide users with feedback regarding their answer
- The Learning tool should provide exercises related to the progression of the algorithm through the graph and the data structure.

Non-functional requirements

A non-functional requirement describes how a software system will exhibit a particular behaviour.

- The Learning Tool must be easy to use
- The Web Application must be extensible/flexible, allowing for future updates and features
- The Web Application must be stable over time
- The Web application must have a fast response time, quick to load and refresh
- The Web application must have functions that are quick to produce output.

Appendix B

Incremental Progression of Dijkstra's Algorithm

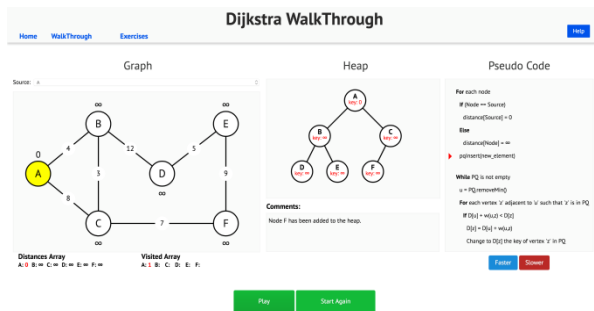


Figure B.1

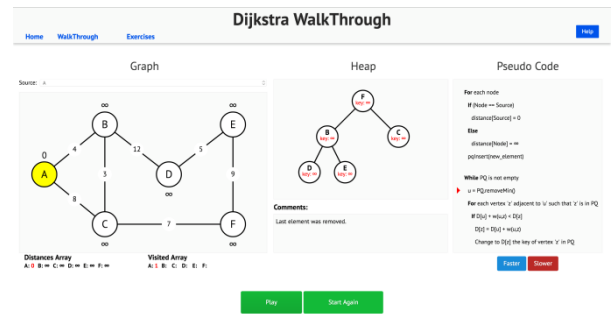


Figure B.2

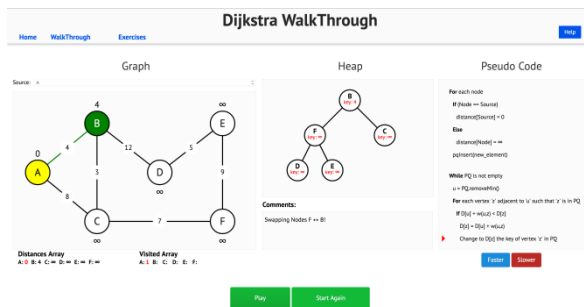


Figure B.3

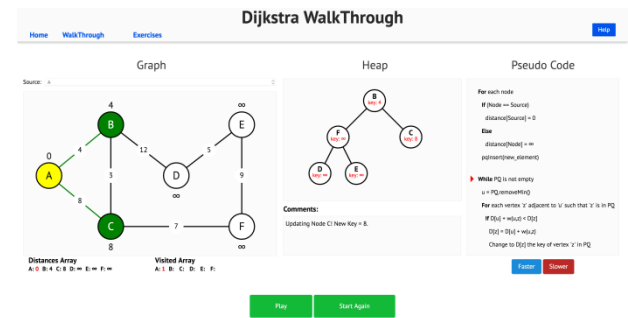


Figure B.4

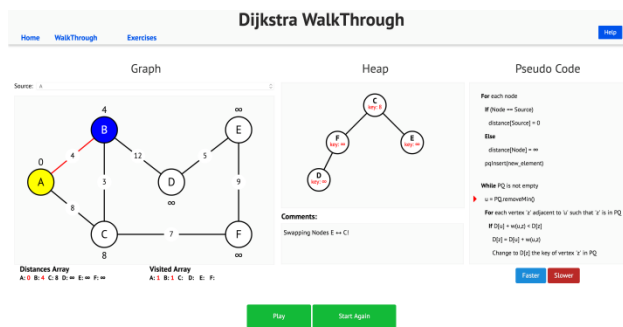


Figure B.5

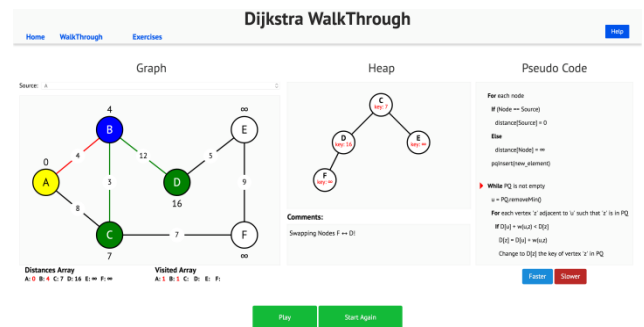


Figure B.6

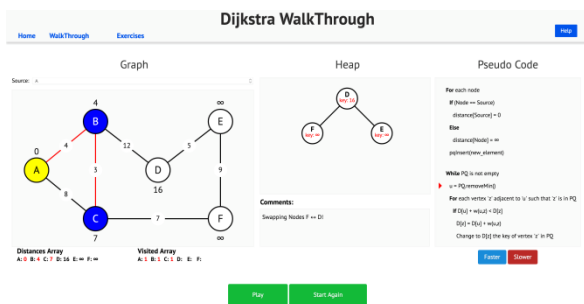


Figure B.7

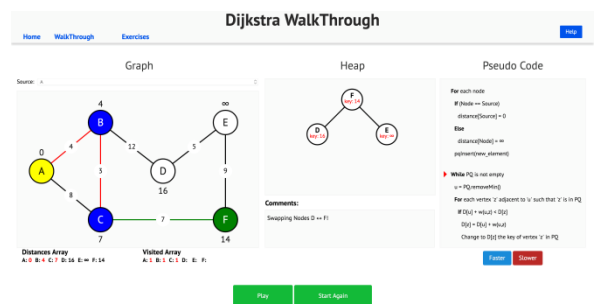


Figure B.8

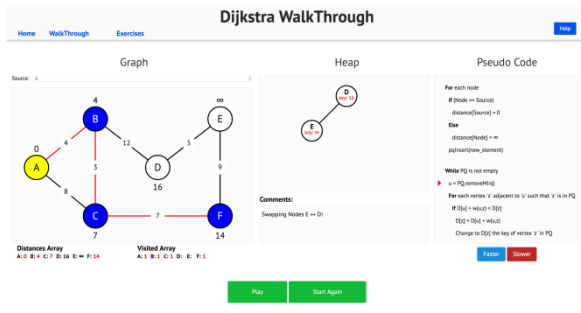


Figure B.9

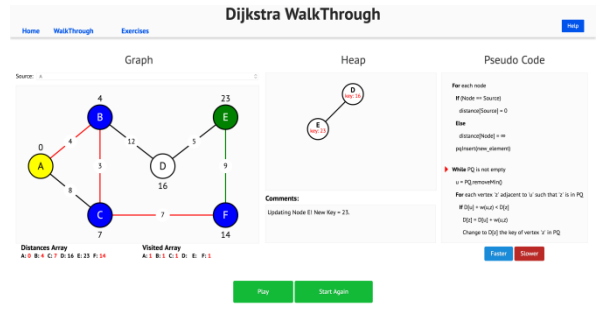


Figure B.10

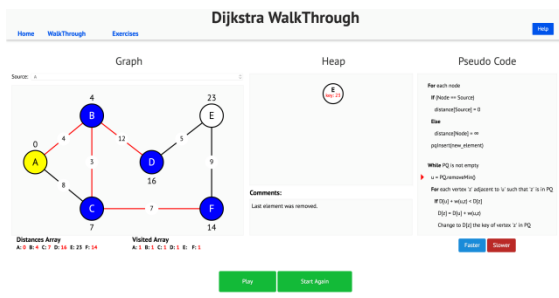


Figure B.11

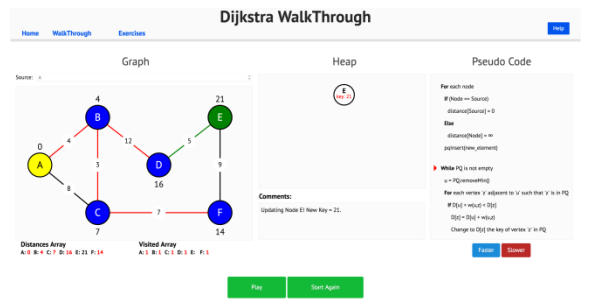


Figure B.12

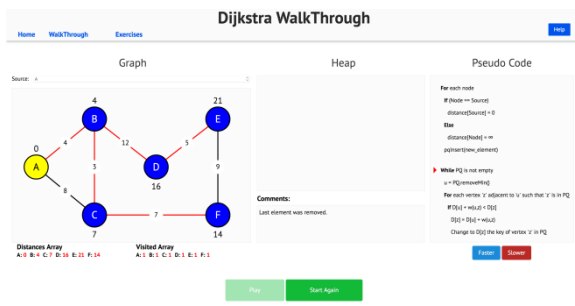



Figure B.13

Appendix C

User Feedback Survey

Edit this form

User Feedback Survey

Please take the time to let me know how useful you found the learning tool to be. Your cooperation is appreciated!

***Required**

Are you studying Computer Science or a related subject? (If you're a graduate or self-taught please answer yes) *

☐ Yes

☐ No

How familiar were you with Dijkstra's Algorithm when you first encountered this learning tool? *

☐ Very familiar

☐ Moderately familiar

☐ Slightly familiar

☐ Not at all familiar

How helpful did you find the WalkThrough section (Set/Interactive) to be in describing the progression of the algorithm? *

	1	2	3	4	5	
Not at all helpful	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very helpful

How useful did you find the Interactive Exercises to be in consolidating your knowledge of Dijkstra's algorithm? *

	1	2	3	4	5	
Not at all useful	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very useful

What section(s) of the learning tool did you find the most useful in aiding your understanding of Dijkstra's Algorithm? *

☐ Set WalkThrough

☐ Interactive WalkThrough

☐ Multiple Choice Questions

☐ Interactive Exercises

☐ Other: _____

For COMP26120 students, how helpful do you find this learning tool to be in assisting you with the "Testing the small world hypothesis" lab and/or revision?

☐ Very helpful

☐ Somewhat helpful

☐ Slightly helpful

☐ Not at all helpful

☐ Other: _____

Overall, how easy to use do you find this learning tool to be? *

	1	2	3	4	5	
Very difficult to use	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very easy to use

How likely is it that you would recommend this learning tool to a friend or colleague? *

	1	2	3	4	5	
Not at all likely	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Extremely likely

If you were to review this learning tool, what score would you give it out of 5? *

	1	2	3	4	5	
Not at all satisfied	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very satisfied

Is there anything that you wish this learning tool allowed you to do that it doesn't allow now?

Your answer

Anything else you care to share? Please feel free to add any comments about improvements or/and concerns you may have.

Your answer

SUBMIT

Never submit passwords through Google Forms.

This content is neither created nor endorsed by Google. Report Abuse - Terms of Service - Additional Terms

Google Forms

Responses to User Feedback Survey

