August 2015

Effects of Visualization on Algorithm Comprehension

Matthew Mulvey
University of Wisconsin-Milwaukee

EFFECTS OF VISUALIZATION ON ALGORITHM COMPREHENSION

by

Matthew Mulvey

A Thesis Submitted in

Partial Fulfillment of the

Requirements for the Degree of

Master of Science

in Computer Science

at

The University of Wisconsin-Milwaukee

August 2015

ABSTRACT

EFFECTS OF VISUALIZATION ON ALGORITHM COMPREHENSION

by

Matthew Mulvey

The University of Wisconsin-Milwaukee, 2015
Under the Supervision of Dr. Susan McRoy

Computer science students are expected to learn and apply a variety of core algorithms which are an essential part of the field. Any one of these algorithms by itself is not necessarily extremely complex, but remembering the large variety of algorithms and the differences between them is challenging. To address this challenge, we present a novel algorithm visualization tool designed to enhance students understanding of Dijkstra's algorithm by allowing them to discover the rules of the algorithm for themselves. It is hoped that a deeper understanding of the algorithm will help students correctly select, adapt and apply the appropriate algorithm when presented with a problem to solve, and that what is learned here will be applicable to the design of other visualization tools designed to teach different algorithms. Our visualization tool is currently in the prototype stage, and this thesis will discuss the pedagogical approach that informs its design, as well as the results of some initial usability testing. Finally, to clarify the direction for further development of the tool, four different variations of the prototype were implemented, and the instructional effectiveness of each was assessed by having a small sample participants use the different versions of the prototype and then take a quiz to assess their comprehension of the algorithm.

TABLE OF CONTENTS

LIST OF FIGURES

1 Introduction

The study of algorithms is central to the field of Computer Science, and every computer science student is expected to learn and correctly apply a large number of algorithms. The quantity of algorithms and the sometimes subtle and nuanced differences between them can make it challenging to remember, select, and correctly apply the appropriate algorithm when faced with a problem on an exam or in a work setting. To address the challenge of teaching algorithms, we present an algorithm visualization tool designed to help students understand Dijkstra's algorithm. It is hoped that any knowledge gained by assessing this tool would be applicable to the design of visualizations for other algorithms as well. This thesis will begin by discussing existing research aimed at addressing the challenge of teaching algorithms, and will then describe the design and assessment of our algorithm visualization tool.

2 Related Research

2.1 Related Algorithm Visualization Research

This section examines some of the existing research related to algorithm visualization. C. A. Shaffer et al. asserts that the theoretical foundations for creating effective algorithm visualizations are strong and improving, and that much of the research indicates that algorithm visualizations can indeed enhance students understanding of core algorithms [1]. Reviewing this existing research provides some direction for our own visualization tool, and some justification for the choices made in its design.

Hundhausen and Brown have created a visualization tool called What you See is What you Code, which dynamically updates a visual representation of the operation of an algorithm as the student types the code that defines the algorithm [2]. Their assertion is that the interactive nature of the tool and the immediacy of the feedback will help students recognize errors and enhance their understanding of the algorithm.

Byrne, Catrambone, and Stasko found that having students watch an animation of an algorithm and then try to predict the subsequent behavior of the algorithm may have some positive effect on the students' performance on a post-test designed to assess comprehension [3].

Grissom, McNally and Naps compared different levels of student engagement in algorithm visualizations, and concluded that visualization that require students to actively engage and participate in some way are more effective than visualizations that only require the student to watch [4]. They assert that it is especially effective when students answer questions related to the algorithm while engaging in the visualization. The emphasis on interaction and on answering questions during the visualization is also advocated by other researchers, including Naps et al [7], and Hansen, Narayanan, and Schrimpsher [5]. In particular, the work of Hansen et al [5] suggests that the use of questions during the visualization can encourage critical thinking about the subject at hand (the algorithm) and avoid the problem of students getting too fixated on the interaction itself. Lawrence, Badre and Stasko advocate interactivity in the form of encouraging students to create their own animations of the operations of algorithms, as opposed to simply watching a premade animation [8].

Kehoe, Stasko and Taylor found that the presence of algorithm animations "seems to make a complicated algorithm more accessible and less intimidating" which can encourage engagement and facilitate learning [6].

Pierson and Rodger assert that the visual nature of algorithm animations provides an alternative way to understand algorithm operation, and may be easier to comprehend than textual information for some students [9].

Saraiya, Shaffer, Mccrickard, and North evaluated the effectiveness of different visualization features and concluded that several features are effective, including: allowing a student to control the pace of the visualization, minimizing distracting features, and providing a good example dataset for the algorithm to act on rather than allowing student to provide their own dataset [10].

Hundhausen and Douglass propose that encouraging students to construct their own algorithm visualizations and then present their work to the rest of the class may be an effective way to promote comprehension of the algorithm [11].

2.2 Related Instructional Research

The instructional theory that informs the design of our algorithm visualization more than any other is discovery learning. Discovery learning has been defined in a number of different ways. Jerome Bruner, considered to be one of the originators of the theory, characterizes discovery learning as follows: "…the very attitudes and activities that characterize 'figuring out' or 'discovering' things for oneself also seem to have the effect of making material more readily accessible in memory" [13]. Alfieri, Brooks, Aldrich, and

Tenenbaum surveyed existing definitions found in the literature and conclude that "discovery learning occurs whenever the learner is not provided with the target information or conceptual understanding and must find it independently and with only the provided materials" [12]. Wouter Van Joolingen asserts that the main advantage of discovery learning is that "the active involvement of the learner with the domain would result in a better structured base of knowledge in the learner as opposed to more traditional ways of learning, where knowledge is said to be merely transferred to the learner" [14]. It is clear that all these definitions have something in common - the idea that students gain a deeper understanding if they discover or figure something out on their own as opposed to simply being told the information. This idea is central to how we approached the design of the algorithm visualization tool, as we will describe in subsequent sections below.

3 Instructional Goal

The instructional goal of the algorithm visualization tool is to give students a deeper understanding of Dijkstra's algorithm and how it works, and to accomplish this in a way that would hopefully contribute to a solid foundation for studying graph algorithms in general. For convenience, Dijkstra's algorithm is presented below, in pseudo code:

```
1  function Dijkstra(Graph, start):
2
3      cost[start] ← 0          // Cost from start to start
4      prev[start] ← undefined  // Previous node initialization
5
6      for each node v in Graph:  // Initialization
7          if v ≠ start
8              cost[v] ← infinity  // Unknown distance from start to v
9              prev[v] ← undefined // Previous node in optimal path from start
10         end if
11         add v to Q  // All nodes initially in Q (unvisited nodes)
12     end for
13
14     while Q is not empty:
15         u ← node in Q with min cost[u]   // Start node in first case
16         remove u from Q
17
18         for each neighbor v of u:        // where v is still in Q.
19             alt ← cost[u] + length(u, v)
20             if alt < cost[v]:  // A shorter path to v has been found
21                 cost[v] ← alt
22                 prev[v] ← u
23             end if
24         end for
25     end while
26
27     return cost[], prev[]
28
29 end function
```

The pseudo code is from http://en.wikipedia.org/wiki/Dijkstra's_algorithm although we have changed some of the terminology for consistency with the terminology used in our tool. We won't present the complete background of the related graph concepts here, as the reader is assumed to already be familiar with this. The reader is simply asked to recall that the purpose of Dijkstra's algorithm is to find the lowest-cost paths from a start node to each other node in an undirected weighted graph. The output of the algorithm is two values assigned to each node in the graph (or two arrays, each of which has an element for each node in the graph). The two values are a cost value, which indicates the cost of the lowest cost path from the start node to a given node, and a "previous node" value (or "prev" for short) which indicates what node would be immediately before the given node on the lowest cost path from the start node. These

values are stored in the “cost” and “prev” arrays declared on lines 3 and 4 in the pseudo code. The algorithm generates this output by visiting each node in the graph exactly once, and assigning cost and prev values to the nodes adjacent to the currently visited node. What allows the algorithm to accomplish this goal while visiting each node only once is the rule which it uses to select the next node to visit. Specifically, the rule says to always visit the node that currently has the smallest cost entered into its cost field (line 15 in the pseudo code). Because this rule is central to why the algorithm works, we have chosen to design the tool to challenge the student to discover this rule through experimentation and reasoning. In fact, we have chosen to make discovering this rule the key challenge facing the student while using our tool, because understanding the basis for this rule is central to understanding the algorithm. The instructional goal then, is for students to understand the steps and the output of the algorithm, and the reasoning behind the rule for which node should be visited next during the algorithm.

4 High Level Design Approach

The design of any instructional tool must be informed by the goal of the tool. In this case, the goal is to give students a deeper understanding of Dijkstra’s algorithm and how it works. As mentioned above in the Related Research section, one effective way of promoting understanding is through the idea of discovery learning. Applying this to the challenge of teaching an algorithm suggests that the student should not be directly told what the rules of the algorithm are, but rather should discover the rules for themselves. Our algorithm visualization tool accomplishes this by allowing the student to manually go through the steps of Dijkstra’s Algorithm, by manipulating and editing a graph

diagram which is the visual centerpiece of the tool. Rather than instructing students about exactly how to perform each step in the algorithm, the tool challenges students to attempt to make the correct choices for themselves and see the results. The hope is that students will discover the correct algorithm rules by experimentation and reasoning, and thereby gain a deeper understanding of the algorithm than would result from simply reading about it or passively watching an animation, as suggested by the theory of discovery learning [12] [13] [14].

5 Design Details and Evolution

5.1 Early Stages of the Design

Figure 5-1 below shows a screenshot of the tool in its early stage:

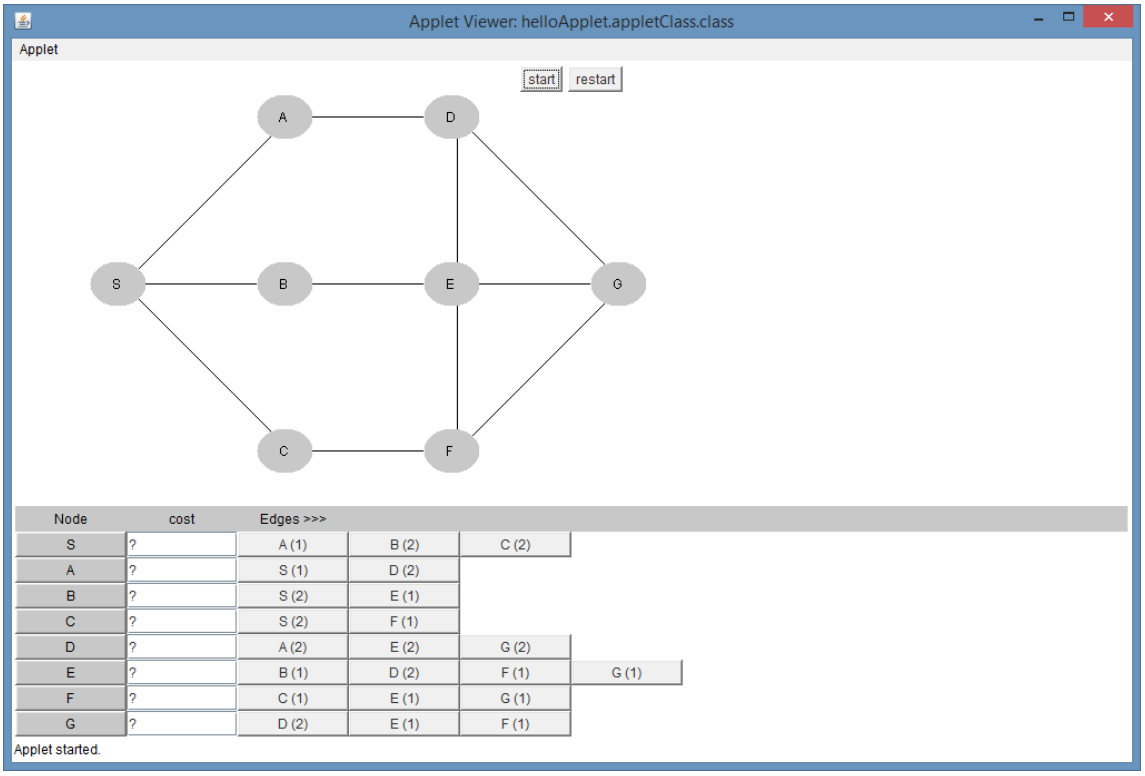


Figure 5-1: Early Stages of the Design

We can see that a graph is presented, as well as the associated graph data in the table in the bottom portion of the screen. The table has a “Node” column that indicates the name of each node, a “cost” column where the student can enter the cost of traversing to that node from the start node, and the list of connected edges for each node, labeled with the name of the adjacent node and the edge cost. The idea was for the student to perform the steps of Dijkstra’s algorithm by clicking on the “Node” buttons to choose which node to visit next, entering costs into the “cost” fields, and choosing each nodes’ “previous” node by clicking on the corresponding edge button in the list of connecting edges. However, we were dissatisfied with certain aspects of this design. One possible issue is that the graph nodes are displayed in a fixed arrangement that is completely visible to the user. This is a typical way to present a graph to students, but we became concerned that it may be misleading and may interfere with students’ understanding of how a computer actually performs an algorithm on a graph. For example, the following diagram is a typical example of how a graph may be illustrated to a student:

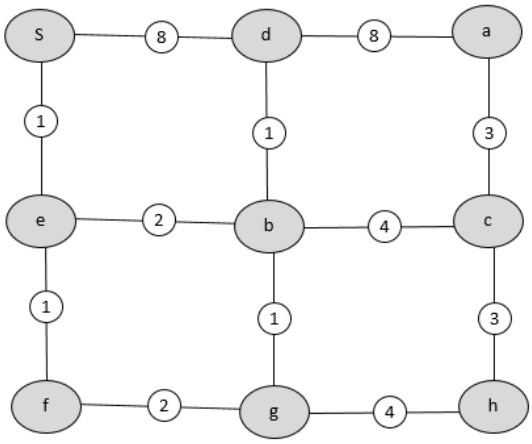


Figure 5-2: Typical Graph Diagram

When presented with a graph diagram like this, and challenged to think of an algorithm for finding the lowest cost path to a given node, the student may try to think of an algorithm that visits nodes in a direction that moves “towards” the target node. However, the notion of “towards” is only a result of how a human thinks of the graph when looking at a diagram like this, while a computer will consider one node at a time and it’s adjacency list, rather than a “big picture” view of the graph. In general, humans will tend to look at the graph as a whole and want to “eye-ball” a solution, which may create misconceptions about how computers process algorithms on a graph. The visualization tool is intended not only to teach students how to perform the steps of Dijkstra’s algorithm, but also to accomplish this in a way that gives them a better understanding of how a computer processes graph data using an algorithm, which in turn will hopefully prepare them to better understand and write graph algorithms in general. As mentioned above, the initial design of the tool, as seen in Figure 5-1, may not be conducive to this instructional goal.

Another dissatisfaction with the early design involved the table in the lower portion of the screen. The table is busy and complicated in appearance, and presents the user with a large amount of information at one time. The problem is aggravated by the fact that the two pieces of information associated with each node (cost, and previous node) are visually represented in two different ways (cost is a text field, while previous node is a list of buttons). According to the work of some researchers such as Mayer and Moreno [15] and Williams [16], it is possible that such a busy and overstuffed visual display may cause cognitive overload in a student, which can hamper learning.

5.2 Current State the Algorithm Visualization Tool

Our algorithm visualization tool is in a prototype stage, suitable for usability testing and some initial effectiveness evaluation, both of which could then lead to refinements in the design of the tool. The tool is implemented in Java, and makes use of the "swing" and "awt" libraries to display user interface elements and draw graph diagram elements on the screen. This section will describe the various features of the tool, in the order in which they would be encountered by the user, and the instructional basis behind the design of each feature. The student's experience with the tool would generally occur in two stages: first an orientation exercise that familiarizes the student with the concepts of weighted graphs and lowest-cost paths, and then the actual visualization exercise that takes them through the steps of the algorithm.

5.2.1 Orientation Exercise

The orientation exercise starts when the user launches the tool. It is designed to familiarize the student with weighted graphs and the concept of lowest-cost paths by having the student fill in the “prev” and “cost” values in the nodes table to indicate the lowest-cost paths in the given graph. Unlike the algorithm visualization exercise that comes later, the student can see the entire graph, and fill in the table in whatever sequence they choose (except for the first couple of steps, as seen below). In essence, the student accomplishes the goal of Dijkstra’s algorithm without actually having to follow the steps of the algorithm, so they can understand the basic context and goal of the algorithm first. Introducing the student to the challenges of understanding the algorithm gradually is intended to reduce the cognitive load placed on the student. We had originally intended to introduce students to the concept of lowest-cost paths with mostly written content which would also include one or two graph diagrams for reference. However, we ultimately felt that a presentation with more visual focus and less words would be more effective, as suggested by the work of Pierson and Rodger [9]. The decision to make the orientation exercise interactive (having the students enter the lowest-cost paths data, rather than passively watching the data appear) is supported by existing research that indicates that interactivity in algorithm visualization enhances learning [4] [5] [7] [8].

Below is a screenshot of what the student sees when they launch the tool, and the orientation exercise starts:

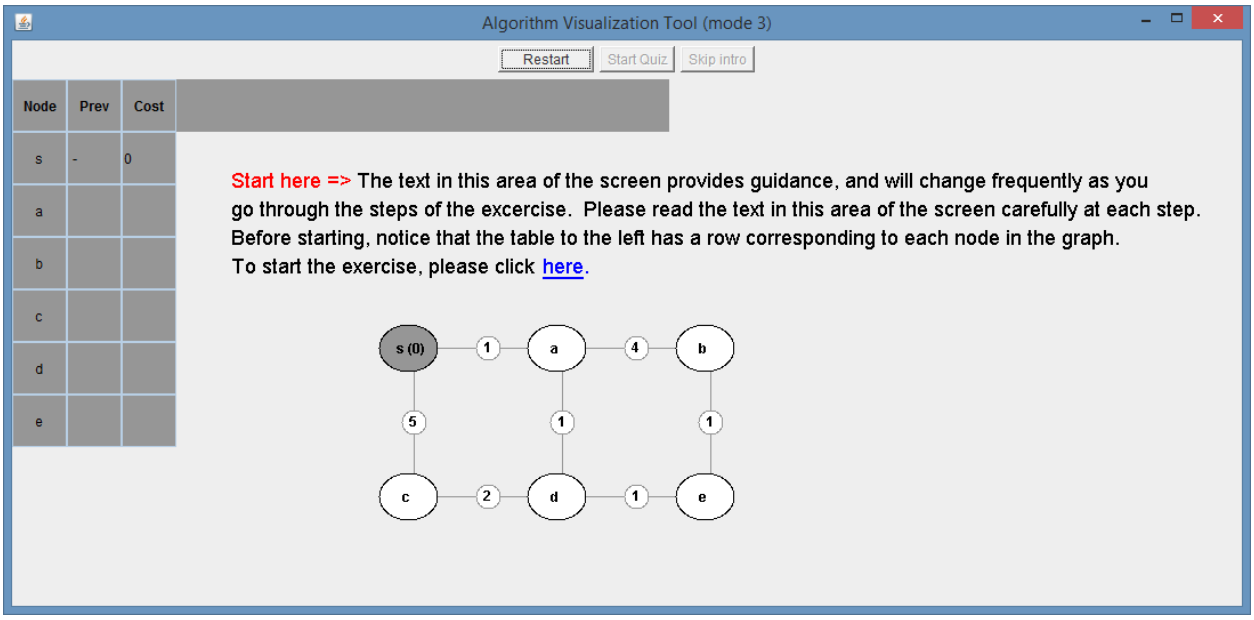


Figure 5-3: Launching the Algorithm Visualization Tool

In this screenshot, we can see that the table of nodes has been simplified compared to the earlier stages of the design – now the "prev" and "cost" columns look the same (the "prev" column consists of text fields, instead of a list of buttons). This decision was made in part to reduce the cognitive load placed on the student.

The interface design of the tool was informed in part by the human computer interaction (HCI) design principles described by Rogers, Sharp and Preece [17]. For example, the text that tells the student how to start the exercise begins with the words "Start here" highlighted in red. Since the rest of the window is muted grey/blue colors, this color-contrast will draw the user's eye so they know exactly where to start. This is an example of the HCI design principle of visibility, which suggests that the necessary information to use an interface should always be visible and clear to the user. Another HCI design principle we used is the principle of constraints, which suggests that the

possible actions a user can perform should be constrained to prevent actions that are not appropriate at a given time. For example, in Figure 5-3 above, the "prev" and "cost" fields in the nodes table are greyed out and disabled, to prevent the user from interacting with them before they understand how the exercise works. When the user clicks "here" to proceed, the screen changes to the image below:

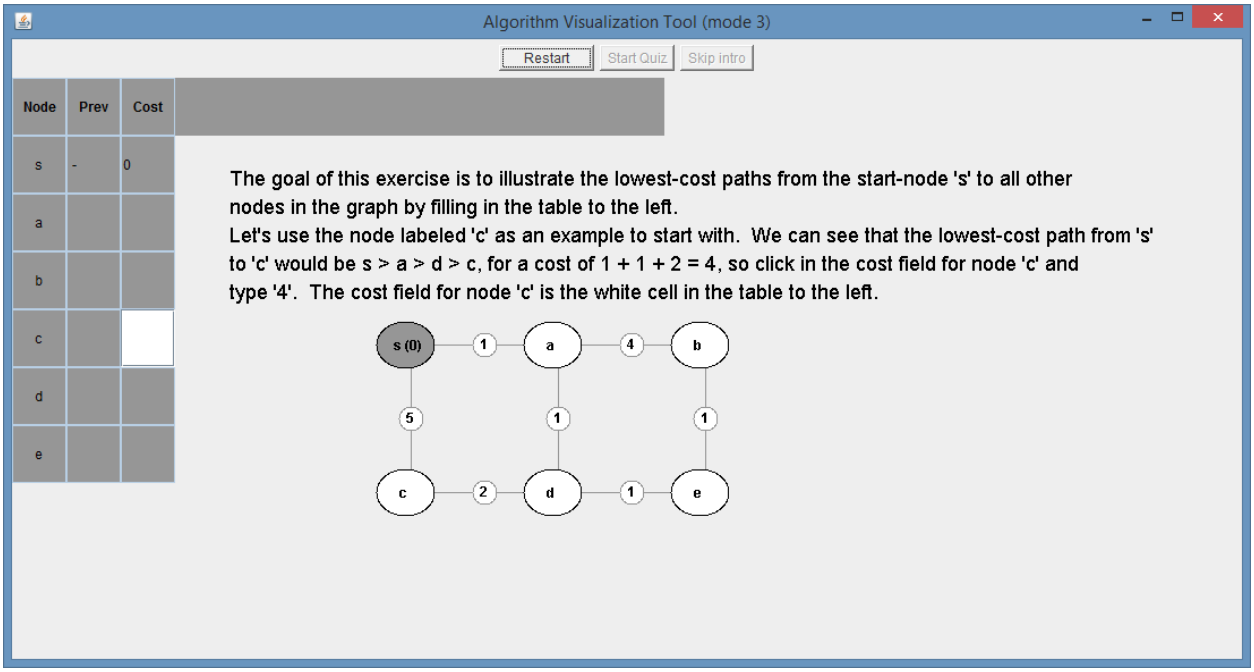


Figure 5-4: Consistency in User Interface Element Placement

We can see that, just as the text on the previous screen shown in Figure 5-3 indicated, the text that provides guidance to the student stays in the same area of the screen as shown in Figure 5-4. This is an example of the HCI design principle of consistency, which suggests that keeping user interface elements in a consistent location from one step to another improves usability. At this point, the student is given an example of the lowest cost path to reach a particular node from the start node, and instructed to enter the

cost into the cost field for that node. Starting with a specific example and allowing the student to only edit the relevant field is another example of reducing cognitive load and using the HCI design principle of constraints. When the user clicks the indicated cost field, the field and the associated node in the graph diagram both turn yellow, to indicate which node is being affected and also to illustrate the connections between the field in the table and the corresponding node in the graph diagram. When the student types the cost into the cost field, several things happen, as illustrated in the following screenshot:

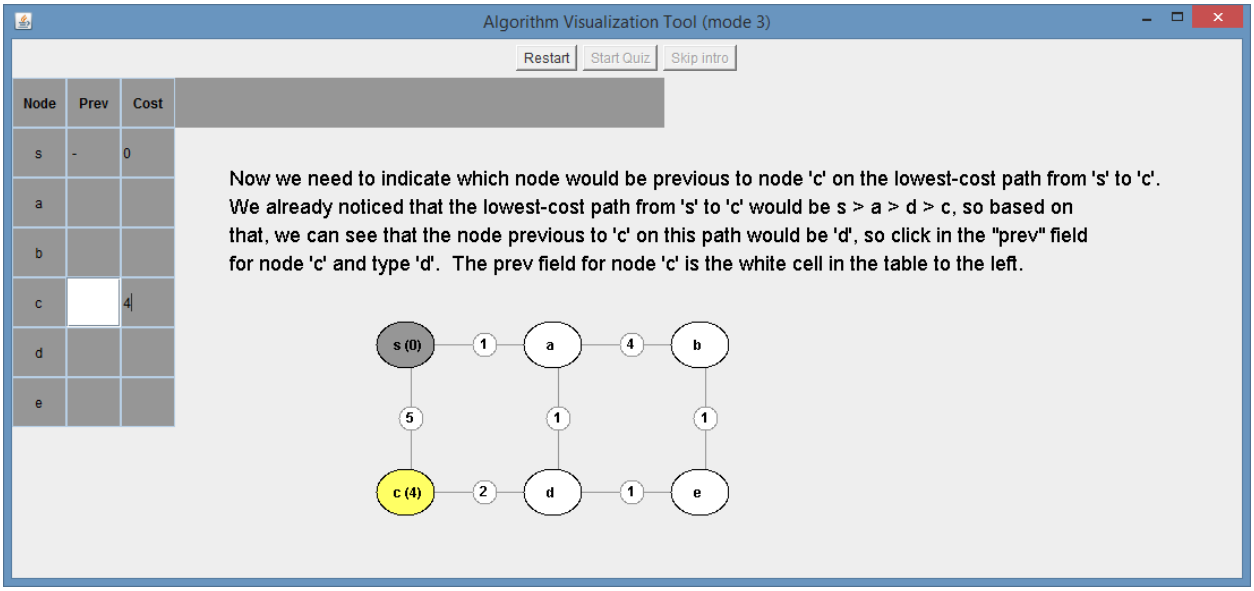


Figure 5-5: Using Feedback in User Interface Design

The label in the node being affected changes to show the cost (4 in this case), the “prev” field for the affected node becomes editable, and the text that provides guidance changes to explain the concept of the previous node in the path, and prompts the user to enter the previous node value. The immediate updating of the graph diagram in

response to the user input is an example of the HCI design principle of feedback, which asserts that clear and immediate feedback that shows the user the results of their actions improves the usability of the interface. The work of Hundhausen and Brown also emphasizes that immediate feedback improves the effectiveness of algorithm visualizations in particular [2]. When the user enters the previous node value, the graph diagram is updated again with an arrow, and a message appears as seen in the following screenshot:

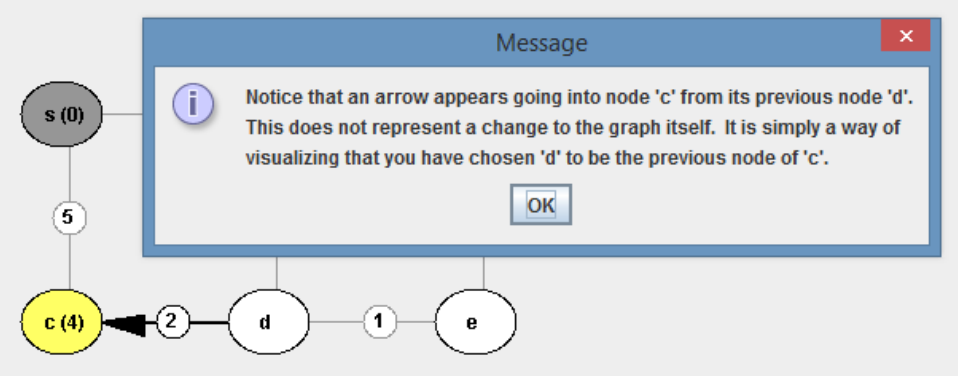


Figure 5-6: Another Example of Feedback

The arrow on the graph diagram is another example of feedback, and the message provides clarity as to what the arrow means. This message was added after usability testing showed that some users thought the arrow was perhaps part of the graph data being revealed at this moment, rather than a visualization of user input.

After entering the previous node data for the initial example node, the student is prompted to fill in the previous node and cost data for each other node in the graph. As suggested by the research of Saraiya, Shaffer, McCrickard, and North, it can be more

effective to design a particular dataset for the student to work with, rather than a random or student-provided dataset [10]. Accordingly, the edge costs of the graph were deliberately chosen to create "interesting" lowest-cost paths – specifically, lowest-cost paths that do not necessarily have the fewest number of edges compared to other paths. For example, in Figure 5-5, the lowest-cost path from node 'a' to node 'b' is (a -> d -> e -> b), rather than (a -> b). This is done to ensure that the student understands that lowest-cost path means the path with the smallest sum of edge costs, rather than the smallest number of edges.

When the student decides they are finished and clicks "done", any errors they might have made are highlighted in red, and they are prompted to correct the errors before proceeding, as shown in the screenshot below:

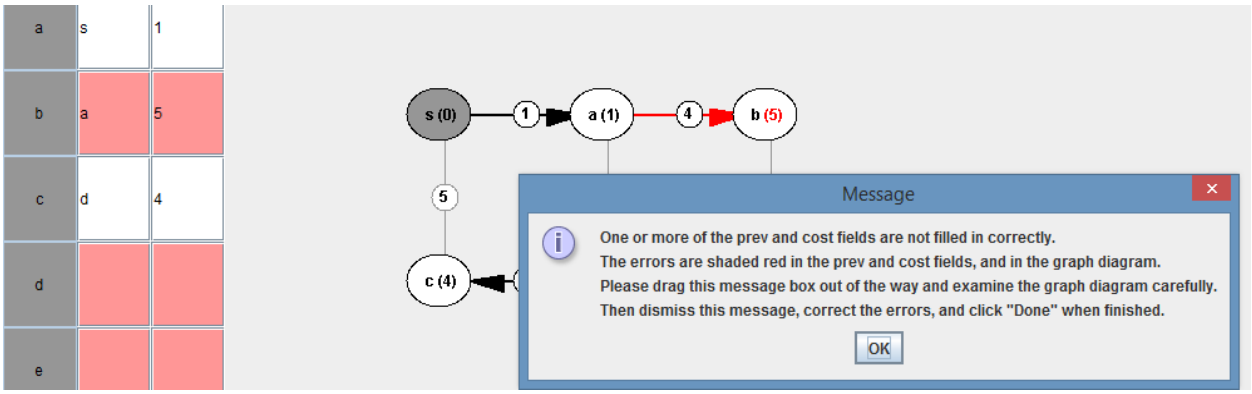


Figure 5-7: Highlighting User Errors

Correcting his or her errors ensures that the student has a solid understanding of lowest-cost paths before proceeding to the algorithm visualization. When the student has correctly filled in the table, the instructional text on the screen points out that the

lowest-cost paths to each node in the graph from the start node can be seen by following the arrows in the graph diagram, as seen below:

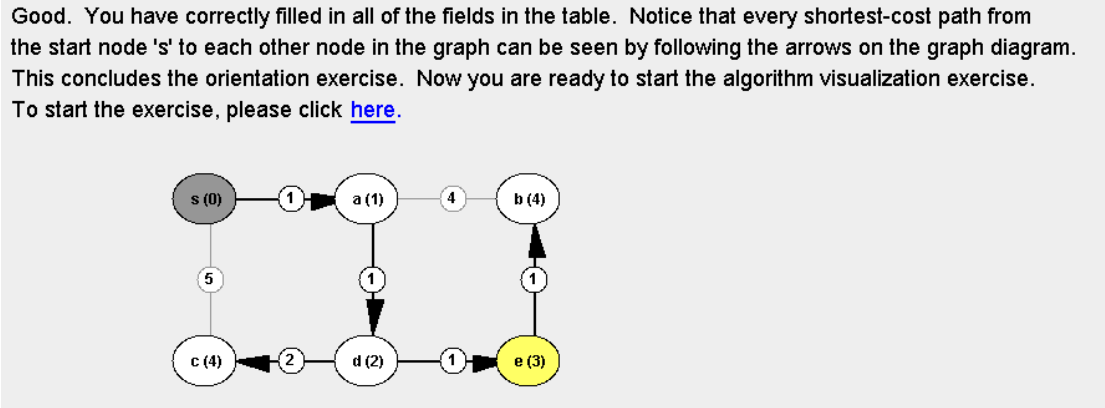


Figure 5-8: Orientation Exercise Complete

This is to emphasize that if the goal is to illustrate the lowest-cost path to each node in the graph, then it is sufficient to enter a previous node value into the table for each node (assuming the entered data is correct). This is an important concept to grasp, because it prepares the student to understand the output of Dijkstra's algorithm and why this output is sufficient to accomplish the goal of the algorithm.

5.2.2 Algorithm Visualization Exercise

Once the student has finished the orientation exercise, the tool starts the algorithm visualization exercise. As stated above, the goal of the algorithm visualization exercise is to provide the student with an enhanced understanding of Dijkstra's algorithm and how it works. To transition the student from the orientation exercise to the algorithm visualization exercise, the following text is displayed in a message box:

> "The goal of the algorithm visualization is the same as it was in the orientation exercise: to find and illustrate the lowest-cost paths from the start node to all other nodes in the graph. However, this time we will accomplish our goal by using an algorithm. An algorithm is a structured set of instructions or steps that can be followed to accomplish some task. The benefit of using an algorithm is that it can be programmed into a computer, to accomplish very large tasks efficiently. The algorithm visualization is also different from the orientation exercise in the sense that some graph information will be hidden from you and revealed only as you proceed through the algorithm. This simulates how a computer would perform the algorithm, focusing only on a few pieces of graph information at a time, rather than looking at the 'big picture' of the graph as a human would."

The purpose of this text is to make the student understand that what they learned about lowest-cost paths in the orientation exercise will apply to what comes next, and that they will now be approaching the same problem with a different method – using an algorithm. It is also intended to prepare the student for some of the differences that will occur because now the tool will be partly simulating how a computer would process graph data, while the orientation exercise was friendlier to how a human perceives a graph. The orientation exercise is designed to give the student some understanding of lowest-cost paths in a way that is comfortable and intuitive, so they have that foundation, which will hopefully help them understand the visualization exercise which may initially be less comfortable and intuitive, but provides insight into how the problem is approached algorithmically.

Below is a screenshot of what the student sees when starting the visualization exercise:

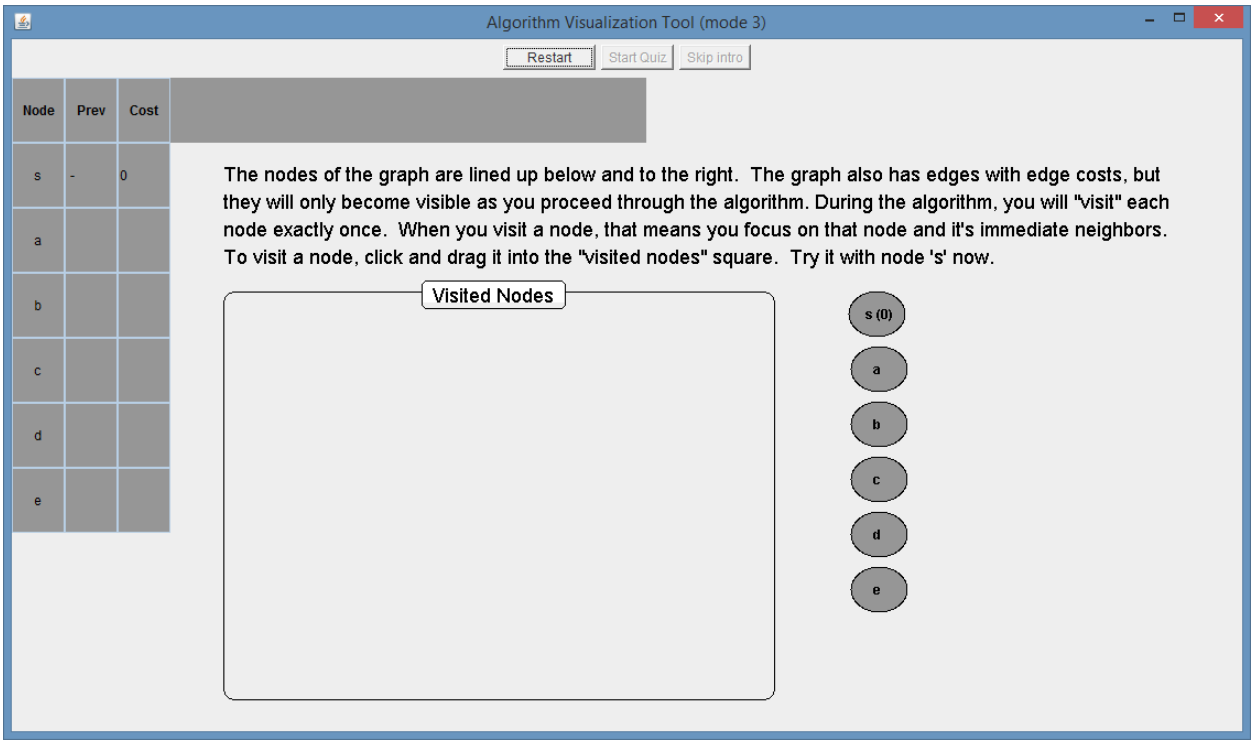


Figure 5-9: Initial Screen of the Visualization Exercise

The node table at the left and the location of the instructional text are the same as in the orientation exercise, which provides consistency to help the student transition into the visualization exercise. What is new is the "visited nodes" square, the arrangement of the graph nodes, and the absence of any visible edges. The "visited nodes" square is where nodes are placed when they have been visited, to strongly emphasize this distinction between visited and unvisited nodes which is an important part of Dijkstra's algorithm. The arrangement of the nodes and the absence of visible edges is intended to address one of the dissatisfactions described above in the "Early Stages of the Design" section. Specifically, the way the nodes are initially arranged is intended to prevent misconceptions that may occur when a graph is presented in a way that is more

visually intuitive to a human. The arrangement chosen for the visualization exercise is intended to suggest the way a computer would process a graph – as a list of nodes with no sense of physical "location" relative to each other, but rather with each node having separate adjacency information that is referenced through that node. In other words, a computer doesn't "look" at a graph visually and "eye-ball" a solution the way a human does, but rather processes it one node at time, in a sequence that is determined by the algorithmic logic applied to the adjacency information that is associated with each node. The design of the tool is intended to help students understand this, so that they can better understand and write graph algorithms in general. One possible concern with keeping edges hidden at this point may be that students might believe that edges are being created as the algorithm proceeds. To mitigate against this, the instructional text on the screen explains that the graph already has edges, but they only become visible as the student goes through the exercise.

We also implemented another measure designed to avoid the misconception that graphs have an inherent spatial arrangement – the node labels are randomly shuffled around each time the visualization is started (not the nodes themselves, just the labels). The reason for this is to avoid a situation where the student tries to anticipate how nodes are connected based on the node labels – or if the student does try this approach, he or she will discover that node connections are not generally predictable based on node labels, which will hopefully instill in them a more accurate concept of how computers process graphs.

As prompted by the instructional text on the screen, the student starts the visualization exercise by dragging the start node into the “visited nodes” square. The screen then appears as below:

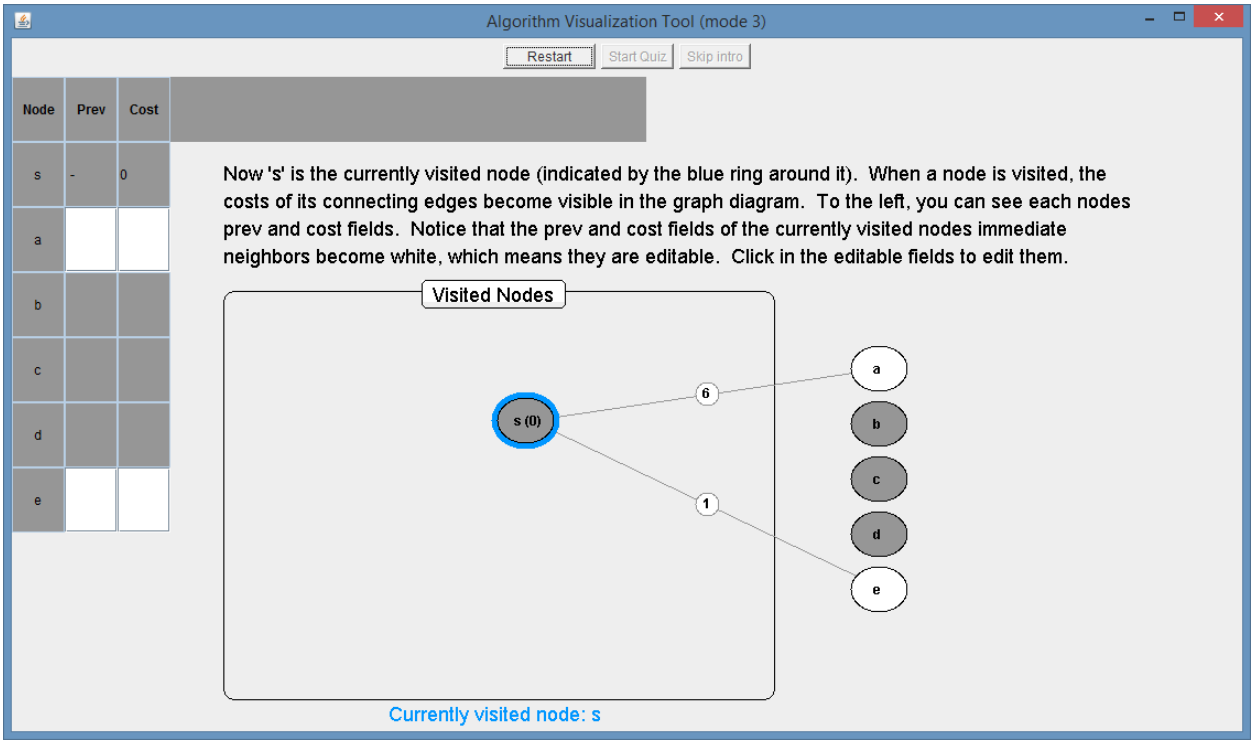


Figure 5-10: Visiting Nodes by Direct Manipulation

There are multiple reasons for having the student manipulate the nodes in this way. One is that dragging nodes into the visited nodes square emphasizes the strong distinction between visited and unvisited nodes in the algorithm – in Dijkstra’s algorithm, all nodes start as unvisited and then get reassigned into a list of visited nodes one at a time. Another reason is that it reduces user-interface clutter on the screen – if choosing which node to visit was not done by manipulating the nodes themselves, there would need to be some other user-interface element added to accomplish this. Finally,

allowing students to move nodes freely emphasizes the fact that the nodes of a graph do not generally have any inherent location information, which ties in with the goal of not introducing misconceptions about graph processing as describe above. One possible risk of allowing users to freely place the nodes is that they may create difficult to read graph arrangements. However, programing the tool to automatically adjust the positions of nodes in any way would run contrary to the purpose of our unconventional graph representation, as described above.

As seen in Figure 5-10 above, once a node is visited, its adjacent edges and edge-costs become visible, and the “prev” and “cost” fields of its adjacent nodes become editable (white). The adjacent nodes themselves also become white, to emphasize the correspondence between nodes in the table and nodes in the graph. The student is prompted to click in the editable fields to edit them, which results in the following screen:

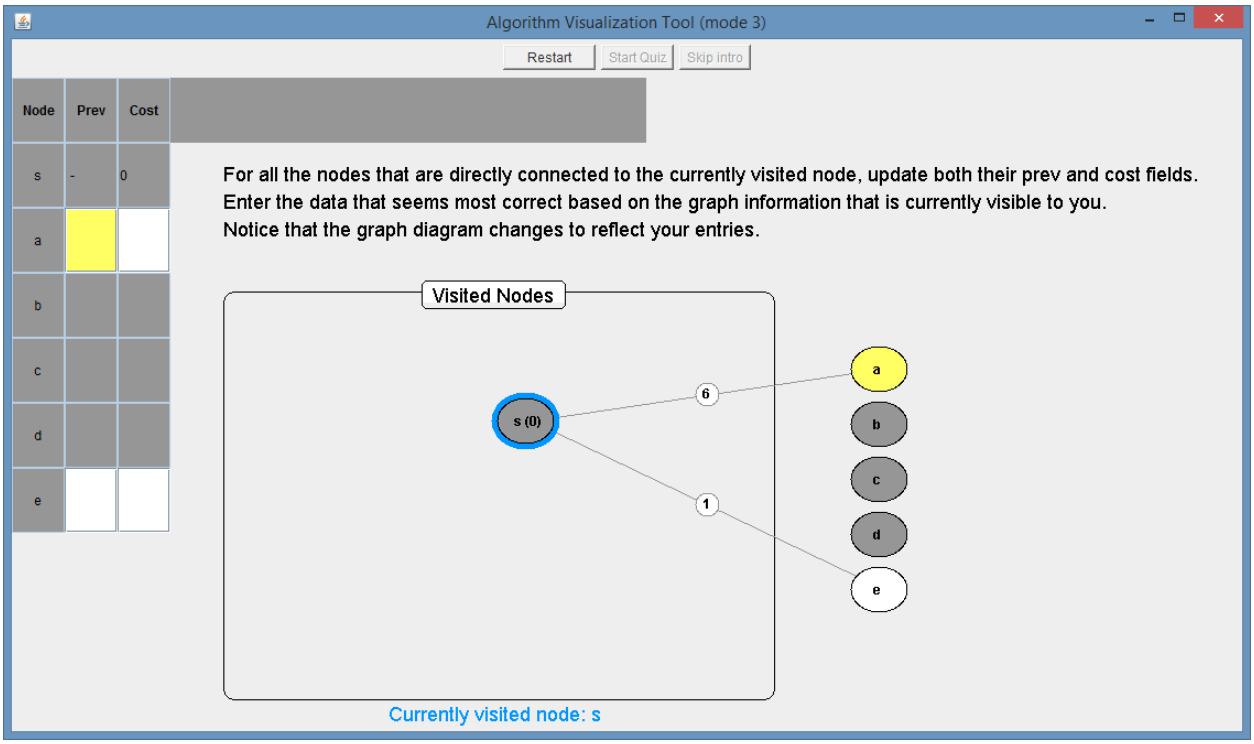


Figure 5-11: Entering Data into the Cost and Prev Fields

The instructional text automatically changes to provide guidance about what the student should do next. Also, the field clicked in becomes yellow, as does the corresponding node, so it is very clear to the student which node is being affected. The student is prompted to enter data into the "prev" and "cost" fields, which corresponds to lines 18-24 in the pseudo code for Dijkstra's algorithm presented above. The instructional text indicates to "Enter the data that seems most correct based on the graph information that is currently visible to you." This mirrors the way Dijkstra's algorithm works – updating the previous node and cost values to reflect the lowest-cost path found so far, based on the graph data that has been processed so far. Once the

student enters data into all of the editable “prev” and “cost” fields, the screen will appear as seen below:

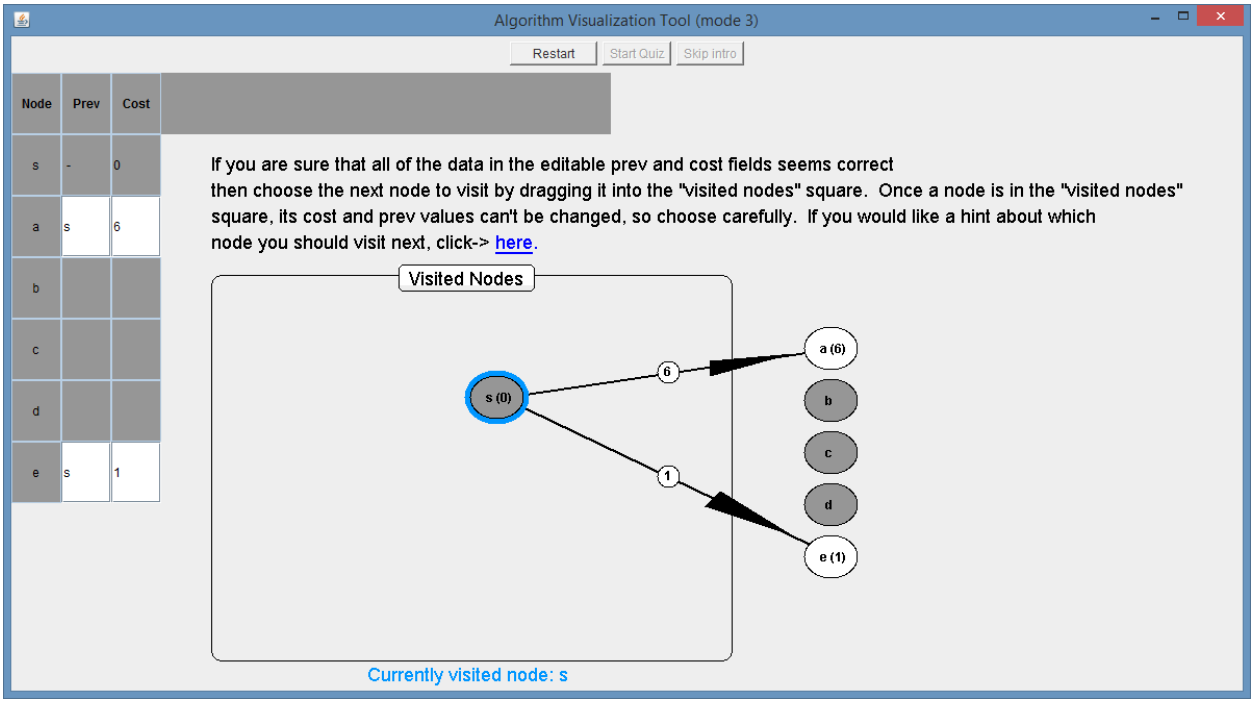


Figure 5-12: Immediate User Feedback during the Visualization Exercise

The graph diagram us automatically updated to reflect the student’s input. Constraining the editable fields to those associated with nodes adjacent to the currently visited node, the color cues that indicate which fields are editable and which node is currently being affected, the consistency in this regard between the visualization and the orientation exercise, and the automatic updating of the graph diagram, are all carefully chosen interface decisions that adhere to the HCI design principles of visibility, consistency, and feedback advocated by Rogers, Sharp and Preece [17].

As seen in Figure 5-12 above, once the student has entered data into all of the “prev” and “cost” fields, he or she is prompted to choose the next node to visit. This decision

corresponds to line 15 of the pseudo code for Dijkstra's algorithm shown in the "Instructional Goal" section above. As explained in that section, the rule for which node to visit next is central to why Dijkstra's algorithm works. For this reason, the tool does not tell the student at this point which node to visit, be rather reminds them that visited nodes can no longer be updated, and cautions them to choose carefully. There is also an optional hint available to the student at this point that reads as follows:

> "Remember, after you visit a node by dragging it into the visited nodes square, its cost and prev values can no longer be changed. So before you visit a node, look at the cost value that you have entered for it, and remember: this cost value represents the cost of the cheapest path you have discovered to that node so far. Then ask yourself: is it possible that you will discover a cheaper path to this node after some more of the edge costs in the graph have been revealed? If this looks possible, then don't visit that node. Instead, wait until you have revealed enough edge costs so you know for certain that the cost value entered for that node is as low as it can be for this particular graph."

The optional hint partially comes in the form of a question designed to get the student thinking about the algorithm and how it works, which is found to be an effective technique for teaching algorithms by the existing research [4] [5] [7]. By allowing the student to figure out the rule for which node to visit next for themselves through experimentation and reasoning, it is hoped that they will gain a deeper understanding of the algorithm, as would be predicted by the theory of discovery learning [12] [13] [14]. It is hoped that the student will realize that choosing any node other than the one which has the lowest cost value currently entered into its cost field is unsafe, because it is possible that a cheaper path to such a node will be discovered after more edges have

been revealed, while the node with the lowest entered cost value is guaranteed to be a safe choice.

After the student selects the next node to visit, he or she is prompted again to update the “prev” and “cost” fields for the adjacent nodes, and the pattern is repeated until all nodes are visited or the student makes an error. The screenshot below shows the exercise after the student has visited several nodes:

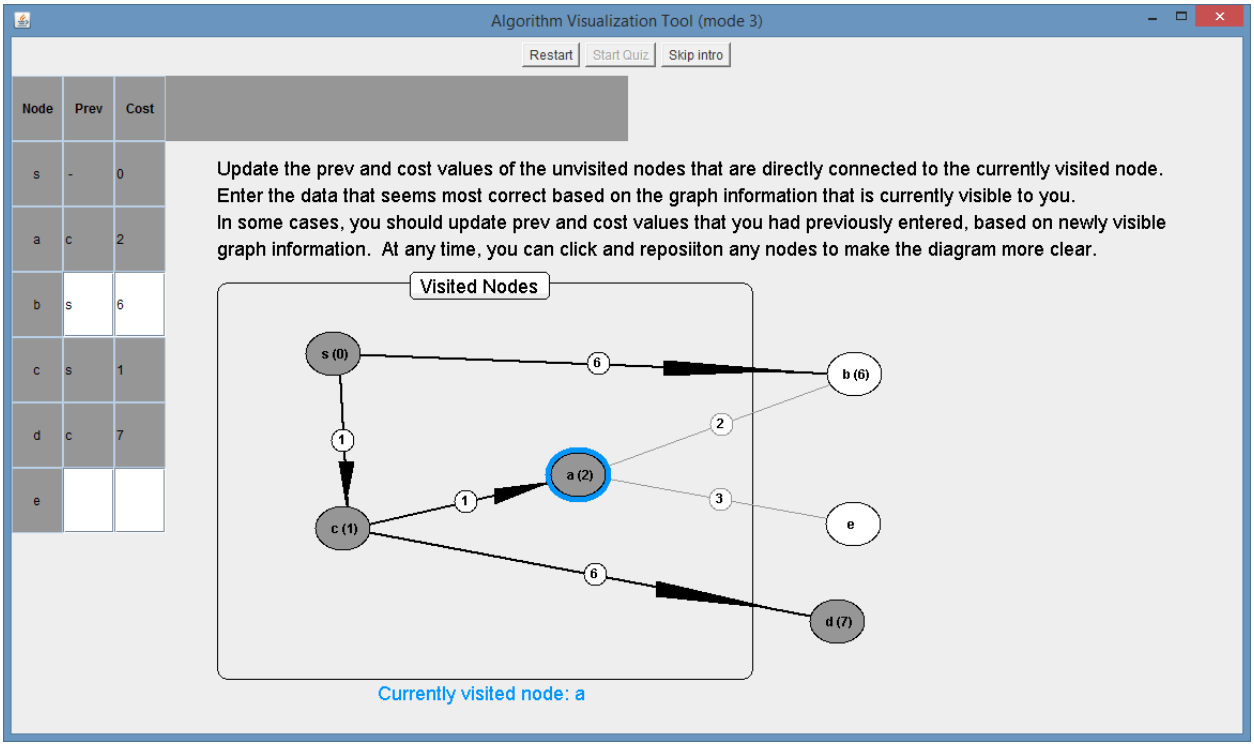


Figure 5-13: An Opportunity to Update Previously entered Data

This particular screenshot shows a situation that is important for the student to encounter. Specifically, we can see that node ‘b’ already has entered “prev” and “cost” data from when the currently visited node was ‘s’. Now the currently visited node is ‘a’, and since ‘b’ is also adjacent to ‘a’, there is an opportunity to update the data for node

‘b’ to reflect a lower-cost path which has now been discovered through the reveal of new edges. This situation highlights one of the keys to why Dijkstra’s algorithm works – it stores the lowest-cost path found to a given node so far, and then updates it if a lower-cost path is found after visiting additional nodes. Because of the importance of this concept, we specifically chose the edge costs in the graphs used in the visualization that would cause the situation described above to occur as often as possible. In other words, the graphs used in the visualization are not randomly or arbitrarily generated, but specifically designed to be pedagogically effective -- an approach supported by the research of Saraiya, Shaffer, McCrickard, and North [10]. There are three graphs for the student to perform the algorithm visualization with, and once they have completed all three, the visualization is done.

If the student makes an error during the exercise, they are prompted to try again with a different graph, as seen in the screenshot below:

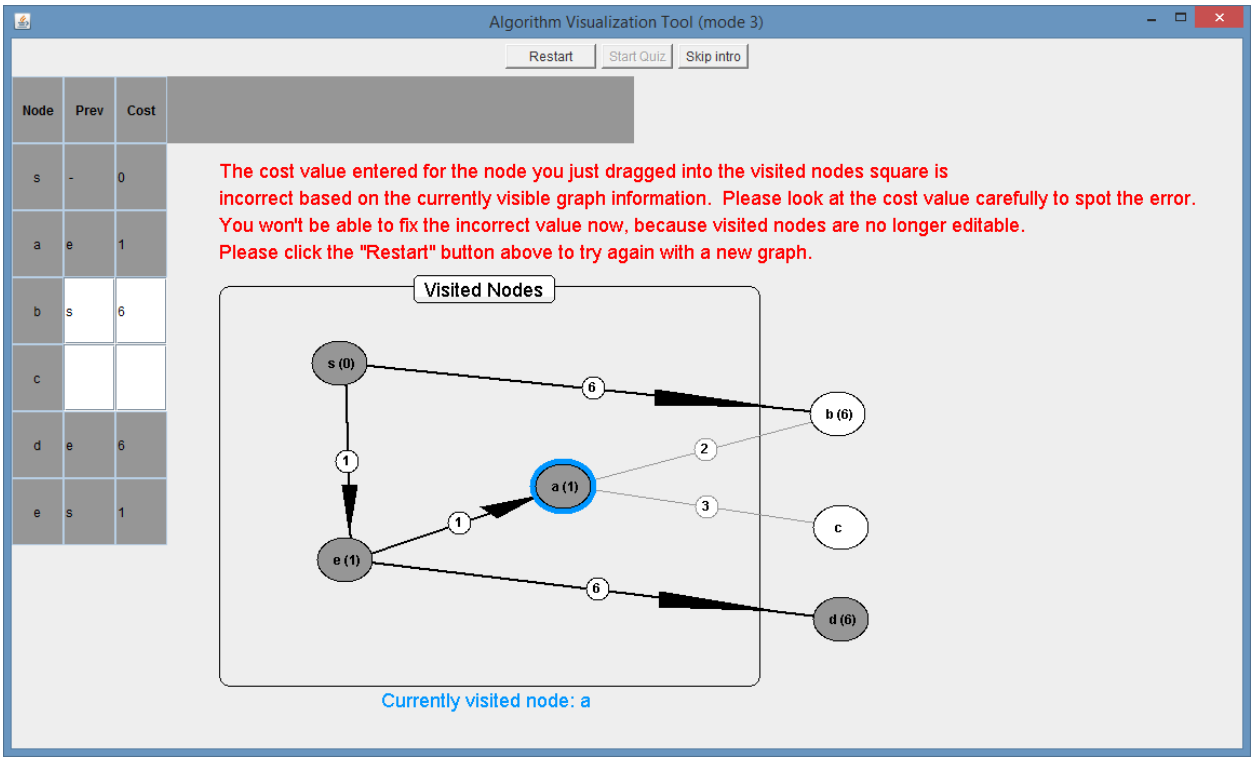


Figure 5-14: Handling User Errors during the Visualization Exercise

The visualization will restart, but the graph will not precisely be "new" as the onscreen text says. It will basically be the same graph with the node labels shuffled around, which makes it appear to the student as a new graph, while still giving him or her the opportunity to complete the intended challenge and corresponding learning outcome. The reason for keeping the underlying graph the same upon restart (as opposed to randomly generating a new one) ties back in with the fact that the visualization only has the three carefully designed graphs intended to provide enough practice with going through the algorithm while not become too repetitive.

6 Usability testing

After finishing a rough prototype of the algorithm visualization tool, we did usability testing with four volunteers. The volunteers were friends and coworkers. All of them have at least a bachelor's degree. Two of them have degrees at least somewhat related to the subject of our study (specifically, math and electrical engineering). The other two have degrees in art and education. The four usability test sessions were done separately, so the volunteers would not be influenced by each other's observations. We used the think aloud protocol [17] for the usability study. That is, we asked each volunteer to use the visualization tool from beginning to end while verbalizing any thoughts or confusions they might have. During the sessions, we observed and took note of the volunteers' choices and verbalizations while remaining quiet so as not to interfere with the user interactions that occurred.

The results of the usability testing were positive – the volunteers were able to proceed through the visualization with very little trouble. The most common problem that occurred related to entering data into the "cost" and "prev" fields in the nodes table during the algorithm visualization. In the initial prototype, the user needed to press 'enter' after typing data into these fields. The reason for this was that the cost value may be a single or double-digit number, so the program could not assume that a single digit typed by the user was intended to be the final input for that field. Although the program provided several reminders that it was necessary to press 'enter', volunteers would sometimes enter data into these fields, but forget to press 'enter', seeming to assume that the data would persist regardless. One solution to this problem may have

been to save the entered data as soon as the user interacted with some other user input element, indicating that they were done typing data into the field in question. However, this would be less than ideal, because the graph diagram changes to reflect entered data, and it may be confusing to the user if the graph didn't change when they typed data, but rather when they moved on to their next user input action. Instead we chose to constrain the edge costs of the graphs used in the visualization so that the cost to get to any node in the graph would never be more than 9. This way, the user could enter data into the cost fields simply by typing a single digit, and the program would immediately respond by reading in the entry and updating the graph diagram. Similarly, the input into the "prev" fields would naturally be a single letter node label, so these fields function in the same way.

Another issue encountered during usability testing involved the arrows which appeared superimposed over graph edges during the visualization. The purpose of these arrows is to illustrate the previous-node choices made by the user, and to show them how those choices could be visualized on the graph diagram. However, some volunteers interpreted the arrows as an inherent part of the graph itself, as if it were a directed graph rather than an undirected graph. To address this issue, we added a popup message that clarifies the cause and purpose of these arrows when they first appear, as seen in the "Orientation Exercise" section above. Another way of addressing this issue could be to program the tool to draw curved arrows that are spatially offset and separate from the graph edges to indicate previous node selections. Testing would

need to be done to discover if this would improve clarity, or if the extra arrows would introduce too much visual clutter an possibly reduce clarity.

7 Assessing Instructional Effectiveness

To assess the effectiveness of the algorithm visualization tool, we have designed a study in which a sample of volunteer participants use the tool, and then complete a short test, which is built into the tool, designed to assess their comprehension of certain key points. The design of the study is based on the instructional goals of the tool. As mentioned above in the "Instructional Goal" section, the goal of the algorithm visualization tool is for students to understand the steps and the output of the algorithm, and the reasoning behind the rule for which node should be visited next during the algorithm. One of the key assertions we are making, as discussed above, is that allowing students to figure out the rule for node selection through experimentation and reasoning should enhance student understanding of this rule, as would be suggested by the theory of discovery learning [12] [13] [14]. Another goal of the visualization tool, also discussed in the Instructional Goal section, is to avoid creating misconceptions about how a computer actually uses an algorithm to process a graph. As discussed in the Algorithm Visualization Exercise section, this is addressed by initially presenting the graph as a list of nodes with no visible edges, rather than as a typical graph diagram where nodes appear to have inherent locations in the plane. We can see then that the visualization tool has two main instructional features that need to be evaluated: the discovery Learning approach (implemented be carefully withholding some information during the exercise), and the unconventional choice to present the

graph as a list of nodes. To evaluate the effectiveness of these two features, we need to compare them to the effectiveness of the tool without them. We have done this by creating four different modes of the tool as follows:

	Guidance provided about which node to visit next during the exercise	**Initial presentation of graph diagram**
1	Explicit	Typical
2	Explicit	List of nodes with edges hidden
3	Hints only	Typical
4	Hints only	List of nodes with edges hidden

The tool is programed to randomly enter one of the four modes above when started. With a large enough sample, this will allow us to divide the participants into four different groups of roughly equal size, with the groups corresponding to what program mode the user happened to use (the participants do not know what mode the tool is in, or even that there is a mode). When analyzing the results, we will know what mode the tool was in for each participant, as explained in the methods section below. The test performance of these four groups will then be compared to assess the effectiveness of the two instructional features mentioned above.

Each question in the test is designed to assess a specific aspect of the participant's understanding and knowledge of Dijkstra's Algorithm. The first four questions are designed to assess whether or not the student knows the rule for how to choose the next node to visit while performing the algorithm:

> 1. When it is time to choose the next node to visit, you should always choose a node that is directly adjacent to the currently visited node. (True/false)
>
> 2. When it is time to choose the next node to visit, you should choose a node that is on a path towards the opposite corner of the graph. (True/false)

> 3. When it is time to choose the next node to visit, you should choose a node that does not yet have any cost value entered into its cost field. (True/false)
>
> 4. When it is time to choose the next node to visit, you should always choose the node that currently has the lowest cost value entered into its cost field. (True/false)

It would be expected that the participants who use either version of the tool that directly states the rule should do well on these questions. The more interesting assessment will be the performance of the participants who use the "discovery learning" versions, which only hint at how to discover this rule. If they do poorly on these questions, it could cast some doubt on the use of the discovery learning approach in the context of this visualization tool, or at least cast some doubt on how we applied the idea of discovery learning.

If the student incorrectly answers "false" to question 4 above, then the test stops, because the next two questions are basically irrelevant if the student has such a fundamental misconception about the algorithms rule for choosing which node to visit next. If the student correctly answers "true" to question 4 above, then the test proceeds to the following question:

> 5. When it is time to choose the next node to visit, why should you always choose the node that currently has the lowest cost value entered into its cost field? (choose one of the answers below)
>
> a) The goal is to keep the path costs as low as possible, so you should avoid visiting the nodes that are more costly to get to.
>
> b) Because, as more edge costs are revealed, you might discover less costly paths to some of the nodes that currently have higher costs entered into their cost fields.
>
> c) Because you might not have a chance to visit the lowest cost node later, while the higher cost nodes can always be visited later.

The purpose of this question is to assess whether the student understands the reasoning behind the rule for choosing which node to visit next, and why it works. This is the kind of deeper understanding that we hope the discovery learning approach might facilitate. By arriving at the rule themselves, through experimentation and reasoning, it is hoped that students will gain a deeper appreciation and understanding of the algorithm, which should in turn improve their ability to apply and adapt the algorithm to practical problems.

The final question is designed to assess whether the presentation of the graph diagram in the tool makes student more or less likely to develop misconceptions about how graphs are processed by computers using an algorithm, as we discussed in the "Early Stages of the Design" section above. The final question reads as follows:

> 6. Suppose we changed the goal of the algorithm – instead of trying to find the lowest cost paths to all nodes in the graph, suppose we are given the name of a single "target node" (like 'a' or 'c' for example), and our goal is to find the lowest cost path to that node only. Would that change the rule for choosing which node you should visit next in the graph?
>
> a) No, the rule would stay the same - always choose the node that currently has the lowest cost value entered into its cost field.
>
> b) Yes, the rule would change – always choose the node that gets you closer to the target node in the graph.

Specifically, we are concerned with the possibility that presenting the entire graph to the student right at the beginning of the exercise, with the nodes arranged in fixed locations and the edges visible, could lead to the misconception that the algorithm should "think" of the nodes spatial locations when traversing the graph, and "think" about moving towards some target in the graph. Answer "b" to question 6 represents

this erroneous way of thinking – it proposes that the algorithm should choose the next node to visit based on which node will move "closer" to the target node. It follows then, that if significantly more of the students who answer "b" to question 6 come from the group who used the "conventional graph representation" version of the tool as opposed to the "list of nodes" version of the tool, it may indicate that the "conventional graph representation" version could indeed cause the misconception described above.

7.1 Methods

This section will describe the methods used to carry out the assessment of instructional effectiveness described above. We recruited participants using Amazon Mechanical Turk, which is an online system for posting Human Intelligence Tasks (HITs). As the name implies, HITs are tasks that require some degree of human intelligence, and that can be done online. People or organizations that need this type of work done can create and post HITs, and then other people can select and complete HITs for some payment. Mechanical Turk does not display any identifying information about the people who complete the HITs. Using Mechanical Turk, we posted a HIT to allow participants to download and use the Algorithm Visualization tool, take a quiz, and return the results to us. The title of the HIT that appeared to people browsing HITs was "Evaluate an educational software program (30-60 minutes)." When someone selected our HIT, they saw the following:

Instructions

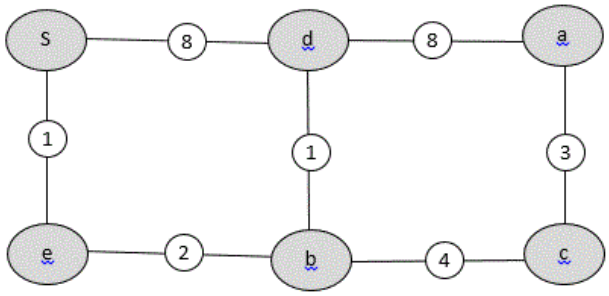
This is an academic study to evaluate the effectiveness of an educational software program. You will be asked to download and run the program, and then answer some questions.

Please read the background information below before starting.

Background information

You will need to be familiar with some basic concepts related to graphs in order to complete this exercise.

An undirected, weighted graph is a collection of nodes and edges, where the edges have no inherent direction, and each edge has a numeric weight (also called a cost) assigned to it. Below is a diagram of an undirected, weighted graph (each node is labeled with a letter; the numbers indicate edge costs):



Now suppose we want to determine the lowest-cost path to each node in the graph from a given start node. The cost of a path is the sum of all the edge costs along that path. For example, in the diagram above, we can see that the lowest-cost path from ‘s’ to ‘d’ is the path along the sequence of nodes s-e-b-d. The cost is 1 + 2 + 1 = 4. This path has a lower cost than the path s-d, which has a cost of 8, even though the path s-d goes through fewer nodes.

That is all the background you will need to complete this exercise.

Downloading the program

You can download the program here: http://www.av-tool-mm.com/AVrun_a.jar

If the program doesn't run, you will need to update the Java runtime files on your computer, which you can download for free here: http://java.com/en/download/

Please download and run the program by double-clicking on it. Then simply follow the instructions which will take you through an educational exercise. When you are done, the program will generate a log file in the same folder where the program resides. Please open this log file in note pad, and copy and paste the contents of the file into the text field below. Thank you for participating.

Please indicate any degrees you have (bachelors, masters, etc.) and what area of study they are in:

[_____]

The program will generate a log file, after you close the program, when you are finished with the exercise. Please paste the entire contents of the log file below:

[_____]

Please enter any comments or observations you have:

[_____]

As seen above, the HIT asks the participant to download and run the visualization tool. When the program is started, it randomly enters one of four different modes, as described above. When they are done and close the tool, a log file is generated which has recorded which of the four modes the tool was in, and some of the user actions while using the tool, as well as their quiz answers. The user is asked to paste the contents of the log file into a field, as well as indicate any degrees and any comments or observations they have. When the participant submits this information, it is then visible to us as the poster of the HIT on Mechanical Turk.

7.2 Results

The results of the instructional effectiveness assessment come in the form of quiz results recorded in the log file generated by the visualization tool. Fifteen people chose

our HIT on Mechanical Turk. All of them completed the visualization exercise and quiz and submitted their results. Below are the quiz questions and results. In additional to indicating the percentage who answered correctly out of all fifteen participants, we will also indicate the percentage who answered correctly out of those participants who used the different modes of the program (randomly determined when the program starts):

1. When it is time to choose the next node to visit, you should always choose a node that is directly adjacent to the currently visited node. (True/false)
Correct answer: false.
Percentage answering correctly:

- For all participants: 86.67%
- For participants who received explicit guidance about how to choose the next node: 100%
- For participants who only received hints about how to choose the next node: 75%

2. When it is time to choose the next node to visit, you should choose a node that is on a path towards the opposite corner of the graph. (True/false)
Correct answer: false.
Percentage answering correctly:

- For all participants: 93%
- For participants who received explicit guidance about how to choose the next node: 100%
- For participants who only received hints about how to choose the next node: 87.5%

3. When it is time to choose the next node to visit, you should choose a node that does not yet have any cost value entered into its cost field. (True/false)
Correct answer: false.
Percentage answering correctly:

- For all participants: 80%
- For participants who received explicit guidance about how to choose the next node: 85.7%
- For participants who only received hints about how to choose the next node: 75%

4. When it is time to choose the next node to visit, you should always choose the node that currently has the lowest cost value entered into its cost field. (True/false)
Correct answer: true.
Percentage answering correctly:

- For all participants: 93%
- For participants who received explicit guidance about how to choose the next node: 100%
- For participants who only received hints about how to choose the next node: 87.5%

5. When it is time to choose the next node to visit, why should you always choose the node that currently has the lowest cost value entered into its cost field? (choose one of the answers below)
a) The goal is to keep the path costs as low as possible, so you should avoid visiting the nodes that are more costly to get to.
b) Because, as more edge costs are revealed, you might discover less costly paths to some of the nodes that currently have higher costs entered into their cost fields.
c) Because you might not have a chance to visit the lowest cost node later, while the higher cost nodes can always be visited later.
Correct answer: b
Percentage answering correctly:

- For all participants: 60%
- For participants who received explicit guidance about how to choose the next node: 57.1%
- For participants who only received hints about how to choose the next node: 62.5%

6. Suppose we changed the goal of the algorithm – instead of trying to find the lowest cost paths to all nodes in the graph, suppose we are given the name of a single "target node" (like 'a' or 'c' for example), and our goal is to find the lowest cost path to that node only. Would that change the rule for choosing which node you should visit next in the graph?
a) No, the rule would stay the same - always choose the node that currently has the lowest cost value entered into its cost field.
b) Yes, the rule would change – always choose the node that gets you closer to the target node in the graph.
Correct answer: a
Percentage answering correctly:

- For all participants: 80%
- For participants who used version with typical graph diagram: 71.4%
- For participants who used version that shows graph as list of nodes: 87.5%

The other results that are potentially interesting are the comments/observations left by the participants, in the "comments and observations" field which we set up as part of the HIT in Mechanical Turk. Eight participants left comments, listed below:

- The directions were a bit hard to understand but once i got what I was doing it was much easier!
- I enjoyed this. Thank you

- I was unable to open the log file for the data above and had to attempt the quiz thrice to get it to work. Finally emailed the file and opened it another computer, after which I could open the file and get the data pasted above. The exercise was fun.

- I found the instructions somewhat unclear and confusing.

- It was fun, once I figured it out.

- At times it wasn't all that clear when I completed at the step, and at the end I wasn't sure if I were done. Perhaps it could have said 'quiz complete, please close this window now' or something to that effect.

- That was a fun HIT.

- It took a minute to figure out what to do, but it was easy after a bit of trying and errors. Also, the directions on the top of the screen were partially covered by a gray bar

7.3 Discussion of Results

The small sample size of participants prevents very concrete conclusions, but it was enough to give a general sense of the effectiveness of the prototype, and to suggest areas for further examination.

The most informative results are the quiz results when considering all fifteen participants as a group. As seen above in the results section, the percentage of participants answering correctly for each question are: 86.67, 93, 80, 93, 60, and 80. These relatively high percentages are encouraging, and indicate that people generally understand at least the basic idea of what the tool is intended to teach them. Even the one lower percentage of 60% for question 5 is not too bad when considering that this was the multiple choice question with three options.

It is also informative to consider the results of the first four questions when considering the difference between participants who received explicit guidance about how to choose which

node to visit next vs. those who received only hints. The first four questions are simply intended to determine if the user knows what the rule for choosing the next node is, not necessarily if they understand why the rule works. As seen above in the results section, here are the percentage of users who answered correctly for these two categories, for questions 1-4:

	Q1	Q2	Q3	Q4
Received explicit guidance for choosing the next node	100	100	85.7	100
Received only hints about how to choose the next node	75	87.5	75	87.5

Although the small sample size makes it difficult to come to a definite conclusion, the fact that the participants who received explicit guidance about how to choose the next node to visit did better on all four of these question does seem to suggest a pattern. However, this pattern could be a result of less than ideal quiz question wording. Specifically, the wording used to describe the rule for choosing which node to visit next in the quiz question is perhaps too similar to the wording used during the exercise. Since the study was designed such that users take the quiz right after using the tool, the quiz might be testing more for remembering the wording of a phrase rather than remembering the meaning. The study could be improved by modifying question 4 to use different wording while retaining the meaning. Another possibility would be to illustrate the rule for node visitation sequence during the exercise by animation rather than with words, to avoid the issue of similar phrasing between quiz questions and the exercise.

Considering how the different modes of the program affected the results of questions 5 and 6 does not provide any useful information. We hoped that the participants who did not receive explicit guidance about how to choose the node to visit next would do better on question 5, as would be suggested by the theory of discovery learning. However, considering that only six

people answered question 5 incorrectly, and of those, three of them received explicit guidance and three didn't, these results seem inconclusive. A larger sample size may have shown some effect of the mode of the tool in this case. Similarly, only three people answered question 6 incorrectly, and of those, two of them used the mode that shows the graph diagram in the typical way, while one of them used the mode that shows the graph as a list of nodes. This would certainly seem to not be enough data to draw any conclusions. In short, we don't have a clear sense of what difference (if any) out two most distinctive instructional features may have made on the effectiveness of the tool.

Finally, the comments made by the participants are worth considering. Three people indicated that they found the instructions to be confusing or unclear at times, but only one of these three offered any specific reason or suggestion for improvement. This suggests that further usability testing would be needed to understand where the confusions are and how to address them. One possible avenue for improving the understandability of the exercise instructions would be to break the text into bulleted style lists of more concise points, rather than the longer paragraph style instructions currently used in the prototype. It is possible that organizing the text in to clear and separate points would help students digest what they are reading more easily. One person mentioned that some text in the user-interface was covered by a gray bar. Testing the program on different platforms and different screen resolutions should be done to understand this issue and correct it. On the plus side, four participants described the exercise as being fun or enjoyable. It is not yet clear whether the program could have enough emotional or "fun" appeal to actually improve student motivation and learning, but it may be worth trying to determine (through usability testing) what features of the tool made people think it was fun, so that these features could be further developed.

One general limitation of our study that should be addressed is the lack of any controls for prior knowledge in the participants. Without such controls, it is more difficult to know why some participants score better on the quiz: is it because of what they learned from the tool, or is it because of prior knowledge related to the subject being tested? To address this limitation, any future studies should limit participants if possible to those with a specific level of computer science education, and a pretest should be used to measure any learning gains from a known starting point.

To summarize the discussion of results then, we have learned that the visualization tool does seem to have some potential, as seen by the positive overall quiz results and some of the comments indicating that the tool was fun or enjoyable to use. However, the sample size has not been large enough to determine which features of the tool have the most positive effect when different modes of the program (including or excluding different features) are compared. One issue that may have reduced the sample size is the complexity of the task description used in Mechanical Turk, which may have deterred some potential participants from attempting the HIT. Considering that some users said they thought the exercise was fun or enjoyable, we possibly could have recruited more participants by posting a much simpler task description, while relying on the "fun" factor to keep users motivated to finish the task even after they discover how involved it is.

8 Conclusion and Directions for Future Research

We have presented the early design stages of a unique algorithm visualization tool that has potential for further development. We are confident that the use of sound and carefully considered design principles, supported by existing research, has resulted in an application that has the potential to enhance algorithm instruction in a computer science education setting. In

particular, the tools design benefits from the use of established HCI design principles as well as the instructional theory of discovery learning. A small usability study also lead to refinements in the programs design. The results of the instructional effectiveness assessment study show that the participants did well on the post-test after using the tool, suggesting that the design methods we used did indeed have a positive effect.

The most useful next step in the development of the visualization tool would be to conduct further research to assess the usability and effectiveness of the different features of the tool. For example, some of the participants in the study left comments indicating that certain parts of the exercise were confusing. Further usability testing should be done to pinpoint exactly which parts caused the confusion, so that these issues can be addressed. Also, while the instructional effectiveness study we did showed generally positive results, the sample size was not large enough to compare the effectiveness of the different “modes” of the tool (where each mode includes/excludes a different combination of features), and the lack of controls for prior knowledge in the participants makes it harder to draw any conclusions from the results. A larger study, ideally using computer science students and an appropriate pretest, could potentially improve the study, and show us how to make the tool as effective as possible.

References

[1] Shaffer, C. A., Cooper, M. L., Alon, A. J. D., Akbar, M., Stewart, M., Ponce, S., and Edwards, S. H. Algorithm visualization: The state of the field. ACM Trans. Comput. Educ. 10, 3, Article 9 (August 2010).

[2] Hundhausen, C., and Jonathan Lee Brown, L. J. What You See Is What You Code: A Radically Dynamic Algorithm Visualization Development Model for Novice Learners. School of Electrical Engineering and Computer Science, Washington State University

[3] Byrne, M.D., Catrambone, R., and Stasko, J. T. 1996. Do algorithm animations aid learning? Tech. rep. GIT-GVU-96-18, Georgia Institute of Technology.

[4] Grissom, S., McNally, M., and Naps, T. 2003. Algorithm visualization in CS education: Comparing levels of student engagement. In Proceedings of the ACM Symposium on Software Visualization (SoftVis'03). 87–94.

[5] Hansen, S., Narayanan, N., and Schrimpsher, D. 2000. Helping learners visualize and comprehend algorithms. Interact. Multimedia Electron. J. Comput.-Enhanc. Learn. 13, 3, 291–317.

[6] Kehoe, C., Stasko, J., Taylor, A. Rethinking the evaluation of algorithm animations as learning aids: an observational study. International Journal of Human-Computer Studies, Volume 54, Issue 2, February 2001, Pages 265–284.

[7] Naps, T., Rossling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., Mcnally, M., Rodger, S., and Angel Velazquez-Iturbide, J. 2002. Exploring the role of visualization and engagement in computer science education. In Proceedings of the Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education (ITiCSE-WGR'02). 131–152.

[8] Lawrence, A. W., Stasko, J., and Badre, A. 1994. Empirically evaluating the use of animations to teach algorithms. In Proceedings of the IEEE Symposium on Visual Languages (VL'94).48–54

[9] Pierson, W. and Rodger, S. 1998. Web-based animation of data structures using jawaa. In Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education (SIGCSE'98). 267–271.

[10] Saraiya, P., Shaffer, C., McCrickard, D., and North, C. 2004b. Effective features of algorithm visualizations. In Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education (SIGCSE'04). 382–386.

[11] Hundhausen, C. D., Douglas, S. A., & Stasko, J. T. (2002). A meta-study of algorithm visualization effectiveness. Journal of Visual Languages & Computing, 13(3), 259-290.

[12] Alfieri, L., Brooks, P., Aldrich, N. J., Tenenbaum, H. R. Does discovery-based instruction enhance learning? Journal of Educational Psychology, Vol 103(1), Feb 2011, 1-18

[13] Bruner, J. S. (1961). The act of discovery. Harvard Educational Review 31 (1): 21–32.

[14] Van Joolingen, W. Cognitive tools for discovery learning. International Journal of Artificial Intelligence in Education (IJAIED), 1998, 10, pp.385-397. <hal-00197349>

[15] Mayer, R. E., & Moreno, R. (2003). Nine Ways to Reduce Cognitive Load in Multimedia Learning. Educational Psychologist, 38(1), 43–52. doi:10.1207/S15326985EP3801_6

[16] Williams, J. S. A Computer Learning Environment for Novice Java Programmers That Supports Cognitive Load Reducing Adaptations and Dynamic Visualizations of Computer Memory (2014). Theses and Dissertations. Paper 574. http://dc.uwm.edu/etd/574

[17] Wikipedia - Think aloud protocol: https://en.wikipedia.org/wiki/Think_aloud_protocol