Debugging Tool to Learn Algorithms: A Case Study Minimal Spanning Tree

https://doi.org/10.3991/ijet.v12i04.6442

Ahmed Y Khedr
Hail University, Hail, KSA.
Al-Azhar University, Cairo, Egypt.
a.khedr@uoh.edu.sa

Hazem M Bahig
Hail University, Hail, KSA.
Ain Shams University, Cairo, Egypt.
hazem.m.bahig@gmail.com

Abstract—This paper presents a visualization tool that works as a debugger to learn the minimal spanning tree. The tool allows the user to enter the graph as a matrix and then enable the user to visualize the execution of the algorithm step by step. During the visualization, the tool can handle and debug the errors that occurred by the user. Also, the tool gives the user feedback from the execution of the algorithm by storing the errors that occurred by the user. The teacher and students can use the tool inside and outside the class. The tool was evaluated by the students, and the results show that the tool enhances the understanding of algorithms.

Keywords—minimal spanning tree; education tool; visualizing tool; Kruskal's algorithm, Prim's algorithm.

1 Introduction

Design and analysis of algorithm is one of the core courses to the students in computer science. There are many issues in teaching algorithms need to illustrate in an attractive way to improve understanding the algorithms. Visualizing algorithms is one of the efficient methods used to achieve this goal. In visualizing algorithm, we use text, graphics, animation, and interaction to simulate the execution of each step of the algorithm.

Many different interactive systems have been designed to learn algorithms in an efficient way. One type of these systems is designed to teach and learn graph problems. Stasko [1] proposed XTango visualized system that is based on the path-transition animation paradigm. The animation system was designed as a general purpose to help the programmers to deign real-time animations for the algorithms. The system includes two problems for graphs: minimal spanning tree and shortest path. The ANIMAL system [1] is based on three roles of users: developers, visualizer and end users.

EVEGA [3] is designed for some graph problems. The system can create, edit, and display graphs. The system includes a maximum−flow algorithm. In [3], the authors present the design and implementation of a learning environment to the data structures and algorithms. The system includes B-tree and minimal spanning tree. GAIGS [5] is an algorithm visualization system based on constructing a sequence of discrete snapshots. Also by using the same strategy, another system was designed and called R-Zoom [6], which stands for Row-splitting Zoom. Algorithm Explorer [7] is a Microsoft Windows platform system that supports three-dimensional graphics and audio visualization. The GeoBuilder [8] is a Java platform that is based on Java. It has the capability to develop code for multiple users at the same time. The proposed system allows students to learn remotely without attending the classroom. PILOT [9] is a web visualization tool for common graph algorithms. JAVENGA [10] is a visualization system for graph and network algorithms. The system includes problems such as shortest path and minimal spanning tree. The system can be used step by step or at once. GraphTea [11] is an interactive tool for Graph Theory teaching courses. The tool provides a gateway to add new functionalities written in Java and MATLAB. In [12] a visualization program for graph algorithms that allows the user to construct a graph, enter algorithms, and watch the step-by-step effect of the algorithm acting on the graph. LAVES [13][13] is an open source used to help students for understanding some algorithms such as Vogel's approximation method and find an edge of minimum weight that are applied to solve problems arising in operations research.

Recently, two visualization tools are proposed. In [14], a visualization tool to construct four geometric spanners algorithms step by steps. In [15], a software tool was proposed to learn Dijkstra's algorithm and the graph dominant set using simple wireless network models.

In this paper, we propose a visualize software acts as debugging tool to learn and teach the student for one of the fundamental problems in the graph which is the Minimal Spanning Tree, MST. The system has the ability to run the algorithm step by step. During running the step of the algorithm by the student, the system has the ability to detect the error that done by the student. Also, the system can support the student when an error occurred during execution. Moreover, the system stores all the information of errors to give the student the feedback of this tracing.

The structure of the paper consists of an introduction and five sections. In Section 2, we present the definition of MST and its algorithms. We describe the details of the system in Section 3. The description includes the different modes of the system, the graphic user interface of the system, and the different layers of the system. The mechanism of debugging for the tool is given in Section 4. In Section 5, we evaluate the system to prove that the goals of the system can be achieved. Finally, the conclusion and future works appear in Section 6.

2 Problem Definition and its Algorithms

Given a connected undirected graph $G=(V,E)$ such that $|V|=n$, a spanning tree of that graph is a tree such that all the vertices are connected together. A *minimum span-*

ning tree (MST) for undirected weighted graph G is a spanning tree, $G'=(V,T)$, with weight less than or equal to the weight of every other spanning tree. Figure 1 represents one of the MST for the graph G.

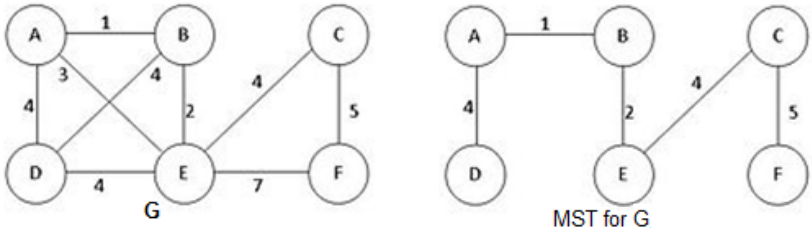

Fig. 1. Graph G and Its MST

MST is one of the fundamental problems in the graph that has many applications such as (1) Design network: telephone, road, and computer. (2) Find an approximation solution for NP-hard problems: traveling salesperson problem, and Steiner tree. (3) Find the solution of some problems by the indirect way such as maximum bottleneck paths and reducing data storage in sequencing amino acids in a protein.

There are many algorithms have been proposed to find MST [10][17][18][19][20]. Some of these algorithms are quite complicated or work in special cases such as dense graphs and graph with integer weights [16][17][18]. The simple and classical algorithms for MST are Kruskal and Prim algorithms [19][20]. Now, we describe the main steps of Kruskal and Prim algorithms.

The Kruskal's algorithm consists of three main steps as follows.

1. Sort the set of edges, E, of G by the weight in increasing order.
2. Start the MST by forest (V,T) such that $T=\emptyset$.
3. Repeat the following until the number of edges in the tree is n-1.
   (a) Select the current edge, e in E-T.
   (b) If adding the edge e to T does not create a cycle then adds the edge e in T. Otherwise, we discard the edge e.

The second algorithm is Prim's algorithm and consists of the following steps.

1. Create two sets of vertices: $X=\{1\}$ and $Y=\{2,3,\ldots,n\}$.
2. While $(Y\neq\emptyset)$ do
   (a) Find an edge $e=(x, y)$ of minimum weight such that $x\in X$ and $y\in Y$
   (b) Move y from Y to X.
   (c) Add the edge (x,y) to T.

3 Description of the System

In this section, we describe our system according to (1) system modes, (2) graphics user interface, and (3) components of the system.

3.1 System Modes

The system can be used as two main different modes during the execution. The first mode is using the system as a debugging system. The system debugs the errors occurred by the students during tracing the algorithm step by step. The system has the ability to (1) detect the error in the current step; (2) prevent the student to go the next step before correct the current step; (3) help the student to make the tracing correct, and (4) give the student many trials to trace the algorithm correctly. This mode of the system is the main objective of the system.

The second mode is using the system as a learning system. The system used by the teacher to learn the students during the class. This means that the system can be used as an auxiliary teaching method. We also can use this system by the students to understand the algorithm outside the class.

3.2 Graphics User Interface

The Graphics User Interface, GUI, of the system is designed based on Windows platform and we use C#.NET as a programming language to build the system. The GUI consists of three main parts as shown in Figure 2. These parts are:

1. *Algorithm* part. It used to display the algorithm and some explanations about each step of the algorithm. It consists of two sections. The first one is used to display the pseudocode of the algorithm. This section appears in the left of the main window. The second section is used to explain each step of the algorithm during the execution. This section appears below the pseudocode of the algorithm.
2. *Input* part. The part appears at the top of the middle of the window. It used to enter the input data for a weighted graph. It consists of two sections. The first section is used to enter the number of vertices of the graph *G*. The second section is used to enter the input graph using the matrix representation.
3. *Tracing* part. The part appears in the middle of the window. It used to trace the steps of the algorithm. In more details, this part is used to display the values of each input, output, and auxiliary variables of the algorithm.

Also, the GUI contains many buttons appear in different parts of the main window. These buttons are *Ok*, *Next*, and *Report*. The button *Ok* is used after the user enters the input of the problem. The button *Ok* will be changed to *Reset* after the user enters the data. This button is used to restart the execution of the algorithm. The button *Next* is used to execute the next step of the algorithm by the system. The button *Report* is used to display a report of errors during the execution of the algorithm.

3.3 Components of the System

The system consists of three main layers as follows. (1) User interface layer: It is responsible for interaction between the user and various components of GUI. The user may be teacher or student. The details of this layer in Section III.B (2) Debugging layer: it is responsible for validating the correctness of the action of the user. If the

action of the user is wrong, then the system will handle this error and the system prevents the user to go to the next step. Otherwise, the system allows the user to trace the next step if it exists. The details of this layer in Section 4. (3) Database layer: It is responsible for storing all information about the error. We use a simple database contains the identification number of the user (instructor/student) and all errors occurred during the tracing as a text message.

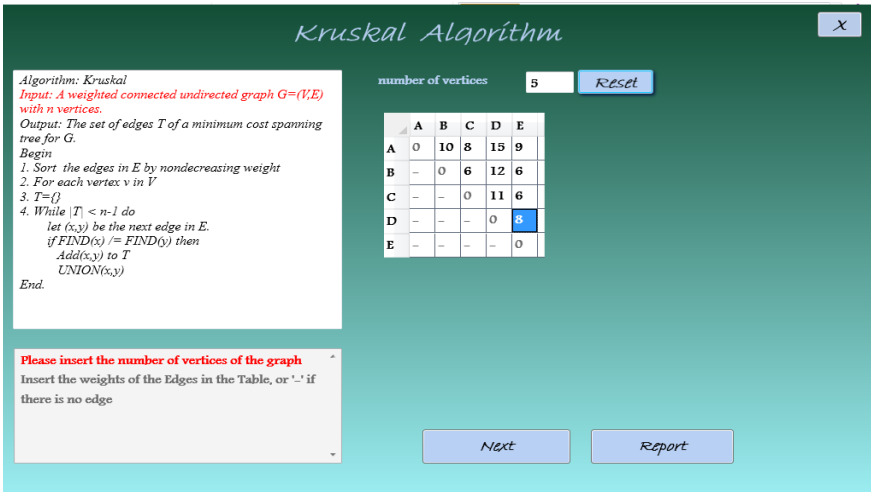

Fig. 2. Main window after entering the input data.

4 Debugging Methodology

The main idea behind the debugging system is to check every action done by the user to determine is this action correct or not. This leads to identifying all the expected errors that may be occurred during entering the input, executing the steps of the algorithm, and the output result of the algorithm.

To illustrate how this part of the system works, we apply the main idea of debugging methodology on Kruskal and Prim algorithms.

In Kruskal's algorithm, we can determine all the expected errors that may occur by the user during the tracing of the algorithm as follows.

For the input of the algorithm: (1) The number of vertices is not a positive integer number. (2) The weight of edges is not a positive integer, except the symbol "-" that is used to indicate no edges between two vertices. For the output of the algorithm: The set of edges for MST is not complete.

For the steps of the algorithm: (1) The set of edges is not sorted correctly. (2) The weight of the selected edge is minimal. (3) The selected edge makes cycle.

For Prim's algorithm, we can determine all the expected errors that may occur by the user during the tracing of the algorithm as follows. For the input of the algorithm, similar to Kruskal's algorithm. For the output of the algorithm, the set of edges for MST is not complete.

For the steps of the algorithm: (1) the initial values for the two sets, X and Y, are not correct. (2) The evaluation of the condition for while loop is not true. (3) The selected edge (x,y) is not minimal. (4) The end points of the edge (x,y) do not satisfy the condition $x \in X$ and $y \in Y$. (5) The value of Y after updating is not correct.

Figure 3 shows how the mechanism of debugging works in the case of Kruskal's algorithm. Also, the figure illustrates how the different layers of the system interact.

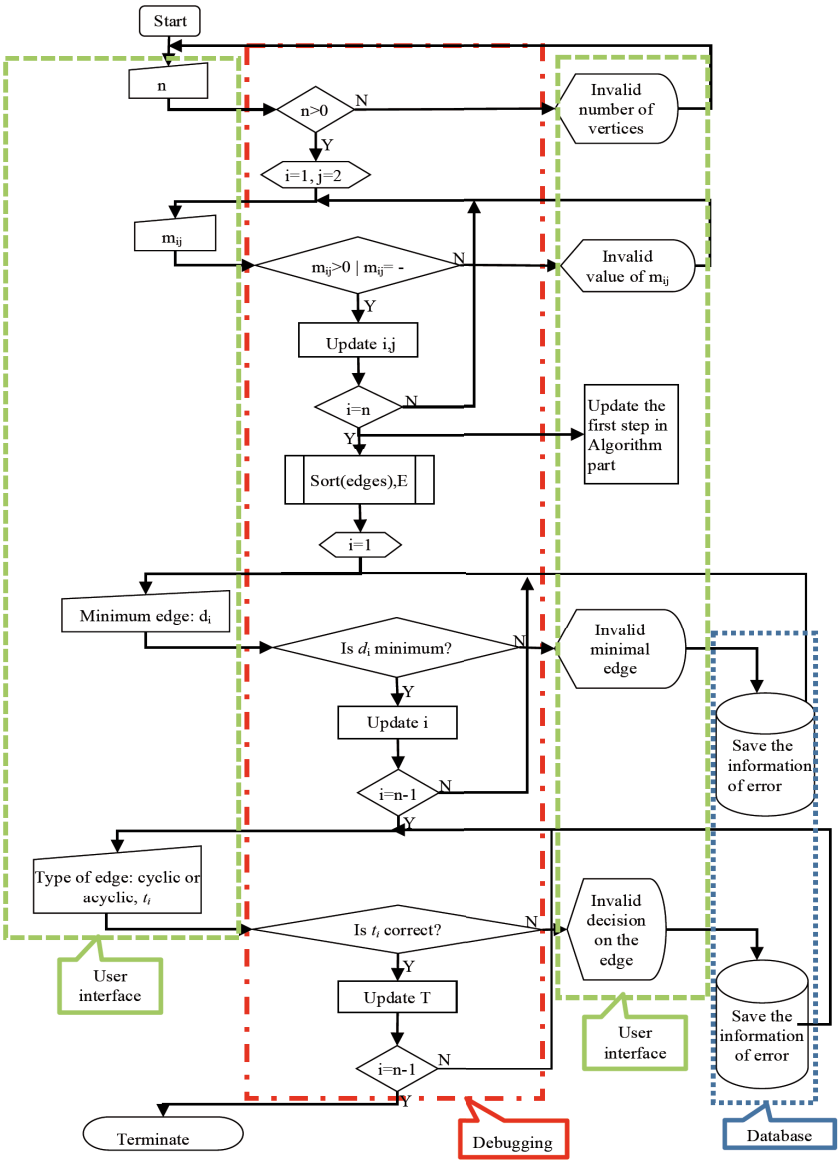

Fig. 3. Flowchart of the interaction between layers of the system in case of Kruskal's algorithm.

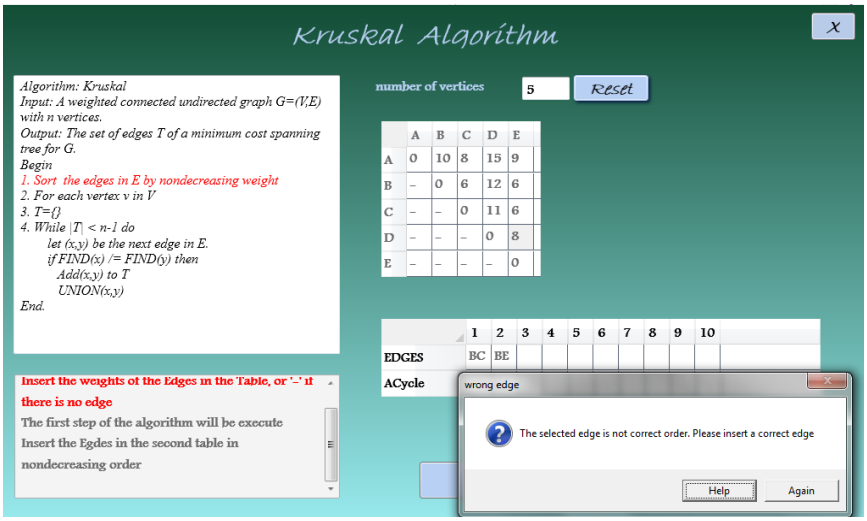

Fig. 4. Window message to indicate an error during sorting the set of edges.

The system starts with entering the user the number of vertices n. By clicking on *Ok* button (or enter), the system checks the correctness of n. If n is not correct, then the system will display a message to indicate this type of error. Otherwise (n is greater than 0) then the system will create a matrix, M, of dimension n x n to enter the weighted graph by the user. The user enters the value of m_{ij} and the system checks the correctness of this value. If the value of m_{ij} is not correct, the system will display a message and then allow the user to enter a new value for m_{ij} again. After entering a correct matrix for the input graph, the system creates a matrix of dimension 2 x |E|. The first row, EDGES, is used to enter a correct ordering for the set of edges according to the weight. The value of each cell in this row is a string of length 2 to represent the start and the end of each edge. The second row, ACycle, is used to determine if the edge makes a cycle or not. The value of each cell in this row is T (True) or F (False). If an error occurred in a cell for the first or second row, then the system will handle this error and a message will appears and then stored it in the database. The system will allow the user to re-enter a correct value by using "*Again*" button or help the user by entering a correct value in a valid position by using "*Help*" button.

In Figure 4, the user will execute the first step of the algorithm. In this step, the user will enter the edges in increasing order according to the weights. If the user entered incorrect data, the system would catch this error as shown in Figure 4. Figure 5 shows all edges of the graph in increasing order as in row "EDGES". Also in Figure 5, the user will take a decision about each edge: is it make a cyclic or not. In the figure, the user makes an error for the edge "CE" and the system catches this error by displaying a window message to indicate the decision of the user is wrong.

Remark: by the similar way we debug the Prim's algorithm.

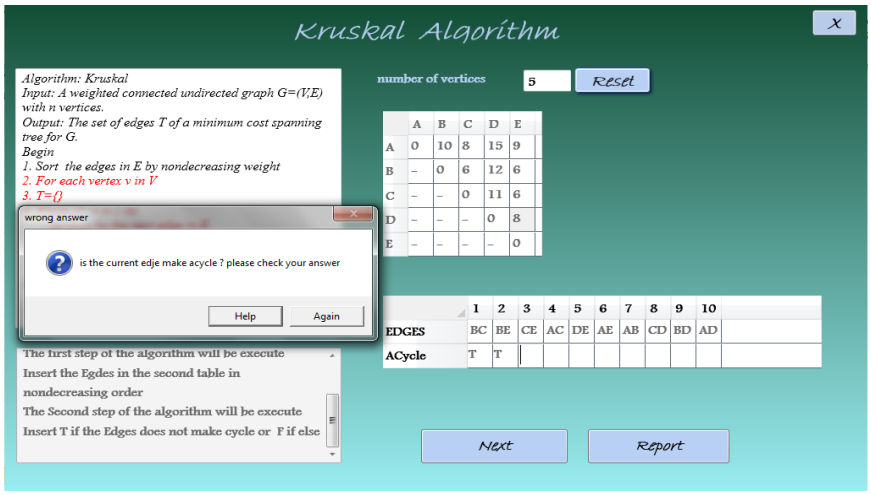

Fig. 5. Window message to indicate an error occurred during selected edge.

5 Assessment of the System

In this section, we studied the effectiveness of using the proposed system on learning. We assess the system on a class consists of 22 students from Computer Science and Software Engineering Department in the College of Computer Science and Engineering, Hail University, KSA. Th

e evaluation of the system is based on designing a questionnaire that consists of ten questions.

The questionnaire is designed according to Likert scale. In Likert scale, each question was given a score from 1 to 5. The value of 1, 2, 3, 4, and 5 indicated strongly rejected, rejected, neutral, accepted and strongly accepted respectively. The questionnaire consists of the following questions. (1) The system easy to use and the interface of the system is user-friendly. (2) The system allows the user to enter different graphs. (3) The system allows the user to trace the algorithm step by step with explanations. (4) The system can detect all possible errors. (5) The system can handle the error that occurred by the user. (6) The system allows the user to re-enter a correct answer if the attempt is wrong. (7) The system saves all errors occurred by the users. (8) The system saves the time by gave different examples in a short time. (9) The system can be considered as a mechanism for self-learning. (10) The system helps and guides the user to understand the MST algorithms.

The questionnaire was distributed after running the system in the class by the teacher and asked the students to work on the system outside the class. Twenty students are responded to fill the questionnaires from twenty-two students. The results and analysis of distributed these questionnaires on the sample data are given in Figure 6. When we analyzed the results,, we found that the minimum value for the average of "Accept" and "Strongly Accept" is 75%. Also, Table 1 shows the assessment results of Likert scale. The average results are in the "Accept" range. In general, the

students accept the system as a tool to help, guide, and learn the students by an interactive way and short time. Moreover, handling the errors and displaying the report of errors are a good way to give the students feedback on understanding the algorithms.

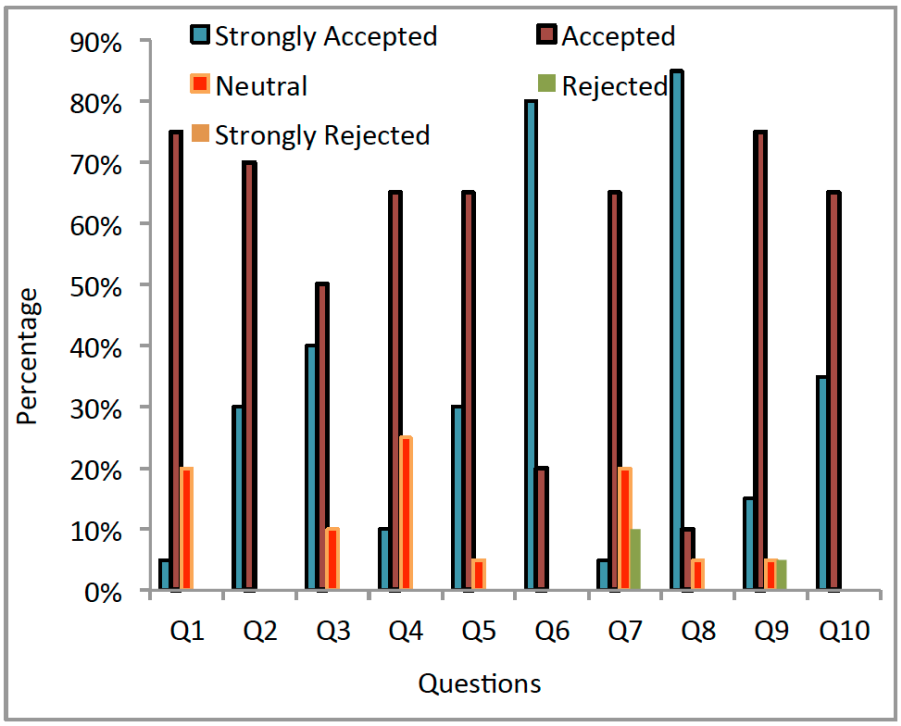

Fig. 6. Assessment results for ten questions

Table 1. Analysis of assessment results with Likert scale

Question	Assessment Results
Q1) The system easy to use and the interface of the system is user-friendly.	3.85
Q2) The system allows the user to enter different graphs.	4.3
Q3) The system allows the user to trace the algorithm step by step with explanations.	4.3
Q4) The system can detect all possible errors.	3.85
Q5) The system can handle the error that occurred by the user.	4.25
Q6) The system allows the user to re-enter a correct answer if the attempt is wrong.	4.8
Q7) The system saves all errors occurred by the users.	3.65
Q8) The system saves the time by gave different examples in a short time.	4.8
Q9) The system can be considered as a mechanism for self-learning.	4
Q10) The system helps and guides the user to understand the MST algorithms.	4.35
Mean	4.215

6 Conclusion

In this paper, we addressed how to learn the MST algorithms by using a debugging tool. The main objective of this tool is to debug the errors occurred by the students while tracing the algorithm step by step. The system can detect the error in the current step, prevent the student from going the next step before correcting the current step and help the student to make the tracing correct. Also, the system used as a tool to learn the students the MST. The system was evaluated by the students to measure the effect of the system in the learning. The results show that the system is accepted by the students as a learning tool for MST.

In the future, we extend the system to work on many different problems related to graphs. Also, we can extend this system for greedy technique. Finally, we apply this system on different platforms such as web and mobile.

7 Acknowledgment

This research was supported by Research Deanship, Hail University, KSA, on grant R2-2013-CS- 10.

8 References

[1] J. Stasko, Animating algorithms with XTANGO, ACM SIGACT News, vol. 23, no. 2, pp. 67-71, spring 1992.

[2] G. Roddling and B. Freisleben, ANIMAL: A system for supporting multiple roles in algorithm animation. J. of Visual Languages and Computing, vol. 13, no. 3, pp. 341 – 354, June 2002. https://doi.org/10.1006/jvlc.2002.0239

[3] S. Khuri and K. Holzapfel, EVEGA: An educational visualization environment for graph algorithms, ACM SIGCSE Bull vol. 33, no. 3, pp. 101-104, Sep. 2001. https://doi.org/10.1145/507758.377497

[4] A. Korhonen, "Visual Algorithm Simulation", Ph.D. dissertation, Dept. Comp. Sci. Eng., Helsinki Univ. of Technology, Espoo, Finland, 2003.

[5] T. Naps and B Swander, "An object-oriented approach to algorithm visualization—Easy, extensible, and dynamic," in Proc. of the 25th Tech. Symp. on Comput. Sci. Educ., SIGCSE, Phoenix, Arizona, USA, March 10-12, 1994, pp. 46-50. https://doi.org/10.1145/191029.191052

[6] J. Urquiza-Fuentes and Á. Velázquez-Iturbide, "R-Zoom: A Visualization technique for algorithm animation construction," in Int. Conf. on Appl. Computing, IADIS, Algarve, Portugal, Feb. 22-25, 2005, PP. 145-152.

[7] E. Carson *et al.*, "Algorithm explorer: Visualizing algorithm in a 3D multimedia environment", in Proc. 38th Tech. Symp. Comput. Sci. Educ., SIGCSE '07, Covington, Kentucky, USA, 2007, pp.155-159. https://doi.org/10.1145/1227310.1227367

[8] J. Wei *et al.* GeoBuilder: A geometric algorithm visualization and debugging system for 2D and 3D geometric computing, IEEE Trans. Vis. Comput. Graph., vol. 15, no. 2, pp. 234-248, March-April 2009. https://doi.org/10.1109/TVCG.2008.93

[9] S. Bridgeman *et al.* "PILOT: An interactive tool for learning and grading," in Proc. of the thirty-first techn. Symp. on Comput, Sci. Educ., ACM SIGCSE, Austin, Texas, USA, 2000, pp. 139-143. https://doi.org/10.1145/330908.331843
[10] B. Thanasis, JAVENGA: Java-based visualization environment for network and graph algorithms. J. Comput. Applicati. in Eng. Educ., vol 20,no. 2, pp. 255–268, 2012.
[11] M. A. Rostami *et al.* "Graphtea: Interactive graph self-teaching tool," in Proc 2014 Int Conf. Educ. and Educal. Technol., EET'14, Prague, Czech Republic, 2014, pp. 48–52.
[12] D. Pagels, "Graph algorithm visualization program," in Proc. of the 48th Ann. Midwest Instruction and Computing Symp., MICS, Grand Forks, North Dakota, USA, April 10-11, 2015, pp. 281-283.
[13] Dominik Kress and Jan Dornseifer, LAVES: An extensible visualization tool to facilitate the process of learning and teaching algorithms, J. INFORMS Trans. Educ., vol. 15, no. 3, pp. 201 –214, May 2015. https://doi.org/10.1287/ited.2015.0140
[14] M. Farshi and S. H. Hosseini, "Visualization of geometric spanner algorithms," in 32nd Int. Symp. on Computational Geometry, SoCG 2016, Boston, USA June 14-18, 2016, pp. 67:1–67:4.
[15] A. Dapena *et al.*, A framework to learn graph theory using simple wireless network models. J. Comput. App. in Eng. Educ., vol 24, no. 6, pp. 843–852, Nov. 2016.
[16] Karger *et al.*, A randomized linear-time algorithm to find minimum spanning trees, J. ACM, vol. 42, no. 2, pp. 321-328, March 1995. https://doi.org/10.1145/201019.201022
[17] M. L. Fredman and D. E.Willard, "Trans-dichotomous algorithms for minimum spanning trees and shortest paths", 31st IEEE Symp. Found. of Comp. Sci., pp. 719-725, 1990. https://doi.org/10.1109/fscs.1990.89594
[18] Gabow *et al.* Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. Combinatorica, vol. 6, pp. 109-122, June 1986. https://doi.org/10.1007/BF02579168
[19] A. Levitin, Introduction to the Design and Analysis of Algorithms. Pearson, 2012.
[20] T. Cormen *et al.* Stein. Introduction to Algorithms. MIT Press, 2009.

9 Authors

Ahmed Y. Khedr is an assistant professor in the Department of Systems and Computer Engineering in Al-Azhar University, Cairo, Egypt. Ahmed is working now in Computer Science and Engineering in Hail University, Hail, Saudi Arabia. Ahmed was funded by the Egyptian government to visit SMU at USA and conduct research in Mobile Computing with the PDA Mobile research group. Ahmed's research area is focused on wireless sensor networks, data mining, and e-learning algorithms.

Hazem M Bahig received the B.Sc. degree in Pure Mathematics and Computer Science from Ain Shams University, Faculty of Science in 1990. He also received M.Sc. and Ph. D. degrees in Computer Science in 1997 and 2003 respectively from the same university. Also, he is currently worked in College of Computer Science and Engineering, Hail University, KSA. His current research interests include high performance computing, algorithm, bioinformatics and e-learning systems for algorithms.

Article submitted 27 November 2016. Published as resubmitted by the authors 23 January 2017.