ISSN No. 0976-5697

Volume 4, No. 3, March 2013 (Special Issue)

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Learner Motivation and E-Learning: Fabricating and Analysing Graphs through JGraphEd Tool

Jitendra Sharma
M.Tech. Scholar, Dept. of Computer Science & Engg.
Swami Keshvanand Institute of Technology
Jaipur, India
jitendra0511@gmail.com

Shubhra Saxena
Reader, Dept. of Computer Science & Engg.
Swami Keshvanand Institute of Technology
Jaipur, India
shubhrasaxena123@gmail.com

Abstract: At present, learning and understanding graph algorithms is a great confrontation for researchers. Instructors are always looking for such kind of application that is especially designed to attest the algorithms and enable users to learn algorithms efficiently. Although, there are few software's which are capable of solving these problems as the availability and compatibility with various environments has been quite a hard task. The graph drawing and analyzing software JGraphEd is easily available and provides better environment, so that users are capable to study and review the algorithm, resolve a hard-headed practical problem and study the functional process via graphical display environment. This can be accessed via any Internet browser anytime, anywhere, without downloading and setting up any software.

Keywords: Graph, Algorithm, Environments, JGraphEd, Software.

I. INTRODUCTION

This paper describes the design, implantation and some development to the Java Graph Editing application and Graph Drawing framework called JGraphEd. [1] It was designed to allow user to draw a graph step by step by adding, removing and modifying nodes and edges. It has a variety of independent algorithms provided for manipulating and visualizing graphs. This paper also suggest what more can be added to JGraphEd to make it better. Section II describes the introduction about menu, toolbar, graph editor modes and visualization of the graph. Section III describes about the code structure of JGraphEd. Section IV gives fairly extensive description of the operation and algorithms that are packaged with JGraphEd. Section V, describes about new application implementation.

II. JGRAPHED

JGraphEd is a Java graph redaction software and graph drawing model. It is contrived for users to create graphs stepwise by adding, removing or modifying nodes or edges. There are many reasons for the question that why JGraphEd was chosen. Some of the reasons are listed below: [1]

a. Most important reason for choosing JGraphEd is that it can be executed online which makes the software platform independent.
b. It is concerned with Java and is compatible with JDK 1.5 or later version of it.
c. A variety of algorithms are implemented in JGraphEd.
d. The code structure is very neat and clean which makes it extensible.
e. Last but not the least**,** the documentation and graph data structures used in JGraphEd is easy to understand.

JGraphEd operates for simple graphs. It has various features which were found more prominent than other graph drawing tools which include modifying graphs in any way. Graphs can be rotated, resized, even one node can be selected and shifted to some other place, nodes can be labeled, edges can be selected and edges can be curved for all the algorithms to implement on them. [1][2]

A. Usage and User Interface:

JGraphEd is useful in analyzing graph algorithm by implementing it on various graphs. It is helpful in drawing a graph and applying different algorithms to the graph to understand the basic properties of the algorithm. Figure 1. shows the user interface which allows the user to interact and handle JGraphEd easily and also shows different algorithms in the form of menu and icon which is used by the user.

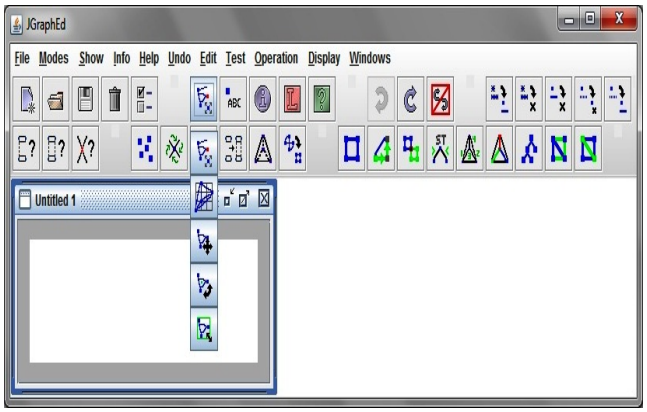

Figure 1. User Interface of JGraphEd Menu and Toolbar

The below Table I shows various types of identifiers used in JGraphEd to provide amend functionality between its algorithms.

Table 1 Various Types of Identifiers used in JGraphEd

M	Describes a menu header.
S	Describes a single action for creating individual menu items and toolbar buttons.
P	Describes a group of actions, for creating a group of radio button menu items and button chooser buttons.
Separator	Denotes that a separator should be placed on the toolbar.
cursorLocationLabel	Used to determine the relative location of the cursor location label.

B. Graph Editor:

This section describes all the features which JGraphEd has for creating a graph on the graph editor. It is responsible for displaying what has been done by the user at any point of time. For example, when the user is making an edge, the graph editor will delineate a line from starting node to ending node. The following list of things shows the user can do with graph editor to draw a graph: [1]

a. There is a feature like rotating a graph and resizing a graph. So graph editor is responsible for putting a center point along which the graph is being rotated and shows a boundary box when the graph is being resized.
b. It is used to store the image which is created by user.
c. It is also used to store special kind of nodes that has been used as input to operations because some algorithm requires some input nodes to be selected. For example, for calculating shortest path between two nodes, the start node and destination node should be selected before applying the algorithm.[5]
d. Changing the cursor of the mouse when moved to graph editor area. For example, when user wants to rotate the graph, the cursor changes to a circular double arrow rather than a single arrow.

C. Graph Editor Modes:

A graph editor operates in one of the five modes namely:

a. ***Edit Mode*** **-** The edit mode is designed to allow the user to use extensive range of options for making and modifying graphs. It mainly uses a graph structure namely "Node-split tree" which is carried out in KD-Tree to speed up the searching of nodes at specific editing locations. [3][4] The set of things that can be done with Edit Mode includes node creation, node selection, node movement, node deletion, node labeling, node colorings, edge initiation, edge sequential display, edge veering, edge unbending, edge directing, edge undirecting, edge pick, edge deletion and edge coloring.

b. ***Grid Mode*** **-** The grid mode is like edit mode, except that nodes are not allowed to move in any direction. It is only allowed to move along the grids. When grid mode is activated, users insert the number of rows and columns in the grid field, and also insert the height and width of the grid cells. [3] In grid mode, it allows application involves node creation, edge orthogonalization, and node selection.

c. ***Move, Resize and Rotate Modes*** **-** The three modes move, resize and rotate are all similar in property with few different mappings. It mainly allows the functionality such as move mode for translating, resize mode for scaling and rotate mode for rotating the graph severally. These are the sole functions that the user can perform with these three modes respectively. The move mode mainly changes the display of mouse cursor to hand icon, and interpret the whole graph in the direction as the user wants it to be. Rotate mode puts a circular arrow mark on the center of the graph and the graph can be rotated along with the circular arrow.

III. CODE STRUCTURE

The code structure is basically demonstration of Java codes of JGraphEd; in other words it's higher than program label. [3][4] The following section describes the basic packages that form the source code of JGraphEd, describes the inner java files, the relationship between them, methods inside the java files. There are some packages and sub packages in JGraphEd source code. Figure 2. shows the UML diagram of all the packages of JGraphEd and the relationship between them.

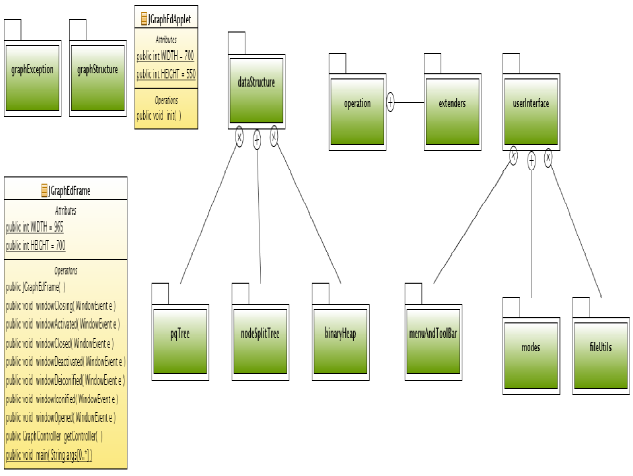

Figure 2. UML Diagram showing all packages of JGraphEd

A. Operation:

The operation class provides static methods that take a graph as input and any other required data as parameters. The pqTree, nodeSplitTree, binaryHeap are sub-packages of Data Structure package and menuAndToolbar, modes, fileUtils are sub-packages of user Interface. [4]

B. DataStructure:

The package dataStructure contains two files, DoublyLinkedList.java and Queue.java. Since JGraphEd requires to the doubly linked list data structure it has been written, because there is nothing like java predefined doublylinkedlist. The UML diagram of dataStructure package is shown in Figure 3. As shown in diagram the package contains two classes DoublyLinkedList and Queue.[6]

CONFERENCE PAPER
II International Conference on
"Advance Computing and Creating Entrepreneurs (ACCE2013)"
On 19-20 Feb 2013
Organized by
2nd SIG-WNs, Div IV & Udaipur Chapter , CSI , IEEE Computer Society Chapter India Council ,
IEEE Student Chapter Geetanjali Institute of Technical Studies, Udaipur, Rajasthan, India

As written in the Queue.java file, the private variables are QueueNode head which is a QueueNode appearing at the head of the queue, QueueNode tail which is also a QueueNode appearing at the tail of the queue and an integer variable called size which is of the size of the queue. QueueNode is one of the sequences of object in Queue class which contains one Object and pointer to the next Object. [5] The method enqueue is responsible for adding an element to the queue and the method dequeue is responsible for removing the element which appears at the head of the queue and the method size returns the size of the queue.

The file DoublyLinkedList.java has three DoublyLinkedListNode variable called head, tail and current and their meaning are obvious from their names. There exists standard enqueue and dequeue operations in DoublyLinkedList.java file which is responsible for adding and removing the element. [5] The enqueue operation takes an object as an input, then creates a doublylinkedlist with the input alone which is of length zero, and then checks the length of the doublylinkedlist for which enqueue operation is called and if the length is zero then it just inserts the newly created doublylinkedlist of length zero in between the head and tail. If the length is not zero it inserts the newly created doublylinkedlist at the end of the given doublylinkedlist

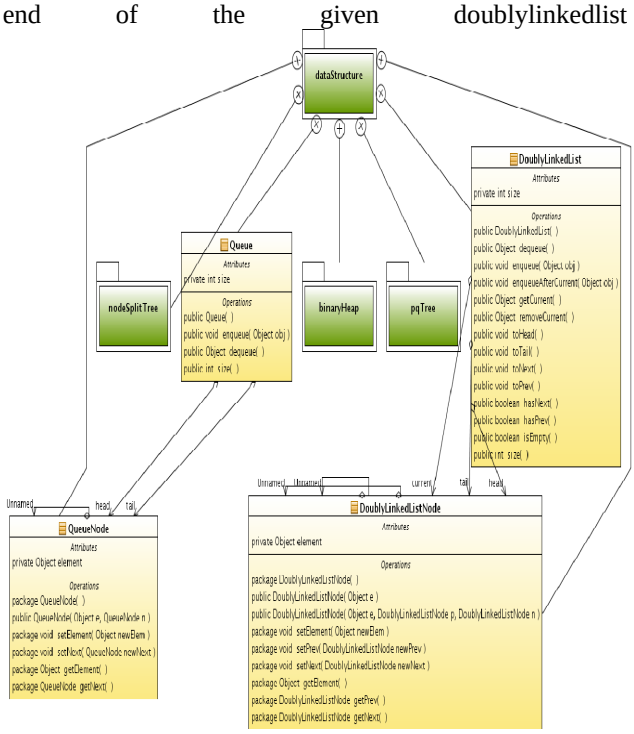

Figure 3. UML Diagram of the package dataStructure

IV. ALGORITHMS IMPLEMENTED ON JGRAPHED

There are many applications implemented on JGraphEd which makes it one of the most useful graph algorithm tools. The applications which are implemented on JGraphEd are creating a random graph, depth first search on a graph, Checking connectivity of a graph, Checking the Biconnectivity of a graph, Making a graph Maximal Planar, Checking the planarity of the graph, performing an embedding of a planar graph, canonical ordering of a graph, normal labelling of a graph, straight line grid embedding of a graph, making tree of a graph, Calculating Dijkstra's shortest path between a pair of vertices of a graph, displaying minimum spanning tree of a graph. There are three types of applications of JGraphEd. [1]

a. ***Test Application-*** It tests various properties of a graph and gives a Boolean result.

b. ***Operation Application-*** This applies some algorithm to a Graph.

c. ***Display Application*** **-** This displays the graph after applying some algorithm and giving the desired result.

A. Random Graph Creation:

Create random graph operation is defined in the CreateRandomGraphOperation class. It is a primitive operation that distributes a number of nodes at random locations in the graph. The user is prompted for the number of nodes to create, and these nodes are subsequently distributed at uniform x coordinate intervals, and random y coordinates across the current graph editor canvas. The intent is that the user may then create edges between these nodes using other operations such as make connected, make biconnected or make maximal.

B. Depth First Search:

The depth first search operation is defined in the DepthFirstSearchOperation class. This operation builds a tree of the nodes, based on the order that they are visited by the search. Table II shows the depth first search operation requires the following additional fields for the nodes and edges: [5]

Table 2 DFS Operation with Nodes & Edges

Node- Integer	Depth First Search Number
Node- Integer	Depth First Search Low
Node-Node	Depth First Search Parent
Edge-Boolean	Is Back Edge
Edge-Boolean	Is Used (has been traversed)

These fields are defined in the DFSNodeEx and DFSEdgeEx extender classes. The depth first search number of a node is set incrementally as it is visited by the search. The low number of a node is the smallest depth first search number that can be reached by the node using non-back edges followed by at most one back edge. There is back edge' flag is used to mark edges that are traversed during the search and result in the visitation of a node that was already searched. The "is used" flag is used to keep track of the edges that have already been traversed during the search.

C. Connectivity

The ConnectivityOperation class provides a variety of methods for performing connectivity operations related to graphs, such as returning all of the connected sub-graphs of a graph, returning all of the nodes connected to

a node, testing a graph for connectivity, and making a graph connected. A graph is connected if all of its nodes can be reached from every other node using a sequence of the graph's edges. [6]

Input: A graph G.

Output: whether the graph is connected or not? How many connected components the graph has?

```
For every connected subgraph g of the graph G
{
Apply Depth first search on g
   If ∃ an edge e ∈ E where E = set of edges of
g such that e is a backedge, the graph G
contains one or more cycle
   Else G doesn't have any cycle.
}
```

To make the graph connected in case it's not connected. [3] For example there is a function which takes a graph as Input and returns the number of connected parts of the graph which the user can see in case it's not a huge graph. To get all of the connected sub-graphs of a graph, the Connectivity class makes repeated use of the depth first search node operation. The graph's copy method is invoked on all of the nodes visited by a depth first search from an arbitrary node. This process is then repeated using a depth first search from any node that was not visited by the last depth first search.

Testing a graph for connectivity is also a trivial operation, it simply determines whether or not there is exactly one connected sub-graph of the input graph. [3] Making a graph connected involves retrieving all of the connected sub-graphs of the input graph. Each of the connected sub-graphs is linked with a new edge to the next connected sub-graph until all of the sub-graphs are linked.

D. Biconnectivity:

The BiconnectivityOperation class provides a variety of method for performing biconnectivity operations related to graph, such as returning all of the biconnected sub-graphs of a graph, finding all separator nodes, testing a graph for biconnectivity and making a graph biconnected. A biconnected graph is a graph which does not contain any separator nodes. A separator node is a node whose removal from the graph results in the graph being split into two non-connected graphs. [3]

Input: A graph G.

Output: whether the graph is biconnected or not?

To make the graph connected in case it's not connected.

V. NEW ALGORITHMS IMPLEMENTED

This section describes about the new applications that is added to JGraphEd, The below three sub-sections describes about some of the additional applications that has been made and implemented in JGraphEd.

A. Checking for Cycles:

There is always a difference in property between a graph having cycles and a graph having no cycles. It affects the bi-partiteness of the graph, Minimum Spanning Tree of the graph and also few others. It is always important to know whether the graph contains a cycle or not, although it is obvious for graphs having less size, the problem may occur for huge size graphs. The CycleCheckOperation.java file in Operation Package checks for every connected graph of a graph G, and applies depth first search and eventually checks for any back edges giving desired result. Figure 4 shows the demo of checking cycles in the graph and the resulted graph contains cycle.

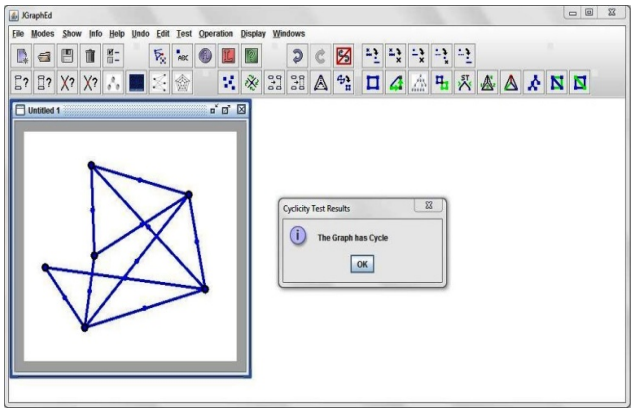

Figure 4. Result showing the Graph contains Cycle

B. Checking of Bi-Partiteness of a Graph:

A graph is said to be bipartite graph only when it does not contain any odd-length cycles in a graph. Because if effort is made to put one node of the cycle of odd length in one set and then on adding the next node in the other set, the final node will appear in the same set, which is not allowed as per the definition of Bi-partiteness which says there can't be any node between the elements of same set.

Input: A Graph G

Output: Whether G is bipartite or not.

Equivalently, a bipartite graph is a graph that does not contain any odd-length cycles. That's because if we try to put one node of the cycle of odd length in one set and then go on adding the next node in the other set the final node will appear in the same set, which is not allowed as per the definition of Bi-partiteness which says there can't be any node between the elements of same set. Figure 5 shows the demo of bi-partiteness in a graph and the given graph is not Bi-Partite.

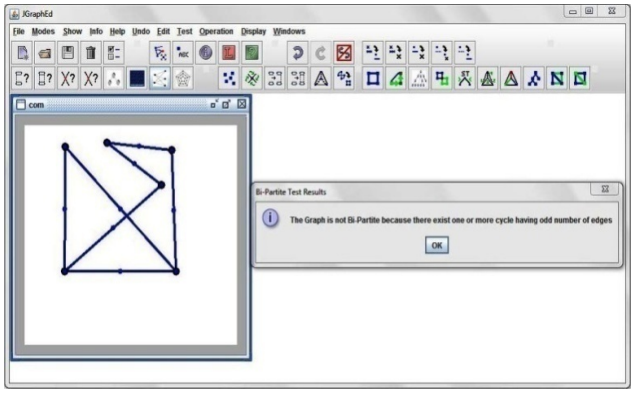

Figure 5. Result showing the Graph is not Bi-Partite

CONFERENCE PAPER
II International Conference on
"Advance Computing and Creating Entrepreneurs (ACCE2013)"
On 19-20 Feb 2013
Organized by
2nd SIG-WNs, Div IV & Udaipur Chapter , CSI , IEEE Computer Society Chapter India Council ,
IEEE Student Chapter Geetanjali Institute of Technical Studies, Udaipur, Rajasthan, India

Algorithm of finding whether a graph is bi-partite or not exactly finding whether a graph contains a graph of odd length or not. Also there exist various algorithms to check whether the graph is bi-partite or not. To find whether there exists a graph with cycles having odd length, first of all we have to check whether there exists any circle for which we have to check whether there exists any back edge in dfs tree of the graph. [6]

To calculate the length of a circle one has to traverse from starting node of back edge through the parents of each node until it reaches the ending node of the back edge.

C. Checking for Isomorphism of Two Graphs:

It is more difficult to identify the isomorphism of two graphs because there are n! potential ways to find out one-to-one correspondence relation between the vertex sets of the given two graphs with 'n' vertices. The checking of one-to-one correspondence relation between the graphs is more difficult when the value of 'n' is large.

The proof that the two simple graphs are not isomorphic is by giving the detail description about the common property that is not shared by both the simple two graphs, if they do not share any property between them, the graph is not isomorphic.

Input: Two graphs G1 and G2

Output: Whether G1 and G2 are isomers or not?

For example, if there are same numbers of vertex in the given isomorphic graphs, there must be a one-to-one relation between the sets of vertices of the graph. Isomorphic graphs also must have the same number of edges, because the one-to-one relation between vertices establishes a one-to-one relation between edges. In addition, the degree of the vertices in isomorphic simple graphs must be same.

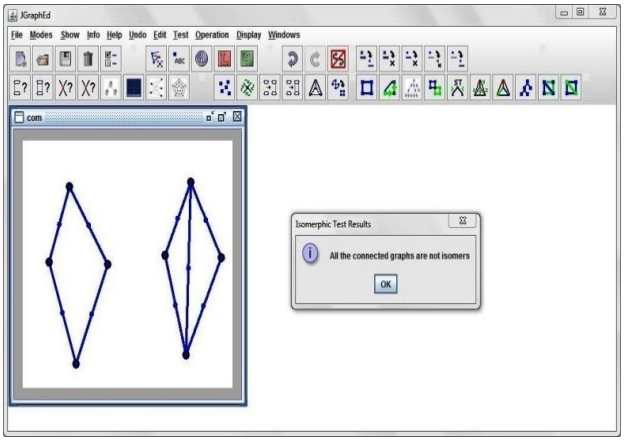

Figure 6. Result showing the Graph is Not Isomorphism

As seen in Fig. 5.3 the two graphs have same number of vertices and edges but they are not isomorphic because the first graph has a vertex of degree one and the second one doesn't. The number of vertices, the number of edges, and the degree of the vertices are all invariants under isomorphism, if any of these quantities differ in two simple graphs, these graphs can't be called isomorphic.

VI. CONCLUSION

This paper has depicted the entire designed views and characteristics of JGraphEd. It also describes the structure and framework for its redaction and drawing potentialities, the execution of algorithms or operations, the different data structures which are furnished with JGraphEd. JGraphEd also assist its use in different graphs and structure and also provides detailing of various algorithms.

VII. FUTURE SCOPE

In the future work some more test and analysis algorithm for example making a 'non-planar graph' planar by deleting some selected edges, implementation of 'Minimum spanning tree' using krushal's algorithm, converting a 'Graph' into orthogonal, calculation of 'Breadth first search' of a graph could be included in present JGraphEd to provide better understanding of graph algorithm.

VIII.REFERENCES

[1] Jon Harris: JGraphEd - A Java Graph Editor and Graph Drawing Framework

[2] Rosen: Discrete Mathematics and its Applications New Delhi: Tata McGraw-Hill Publishing Company Limited Chapter 8: Graphs: 560 – 563

[3] Gary Chartrand, Ping Zhang: Introduction to Graph Theory

[4] Herman, Ivan, Melançon, Guy, Marshall, M. Scott, "Graph Visualization and Navigation in Information Visualization: A Survey", IEEE Transactions on Visualization and Computer Graphics, 2000

[5] Di Battista, Giuseppe, Eades, Peter, Tamassia, Roberto, Tollis, Ioannis G., "Algorithms for Drawing Graphs: an Annotated Bibliography", Computational Geometry: Theory and Applications 4, pp. 235–282, 1994

[6] Caccetta and K. Vijayan, "Applications of Graph Theory", Fourteenth Australasian Conference on Combinatorial Mathematics and Computing, vol.-23, pp. 21-77, 1987