GRAPH EDITOR DISPLAY TOOL

USER DOCUMENTATION
GROUP H
CS 2370

May 1, 2002

INTRODUCTION

This document provides general information and guidelines for using the Graph Editor Display Tool. It includes a functional description of the software's capabilities and uses, installation instructions, an introductory manual to help you get started with this application, and a reference manual for more detailed descriptions of algorithms and messages. Following is a table of contents for easy reference.

TABLE OF CONTENTS:

SECTION 1: FUNCTIONAL DESCRIPTION

This software provides and easy-to-use tool for displaying standard graph ADT algorithms in action. Using an intuitive GUI environment, you can create several types of graphs and visually trace through the execution of various traversals, spanning trees, paths, and sorts. Graphs can be created according to your own design using the GUI or input using text format. Graphs can be saved as files and retrieved for editing or reuse.

Used in conjunction with courses in data structures and algorithms or graph theory, this software provides an invaluable learning resource. It is ideal for classroom instruction where multimedia tools are available. Further, it is instructive for students on an individual basis, helping them become familiar with graph structures.

SECTION 2: INSTALLATION DOCUMENT

This software is portable to any common system, and will run under Windows, MacOSX, and UNIX. As a java application, this program requires you have any java version 1.2 or higher on your machine prior to installation.

Upon successful installation, this user documentation will be available in the same directory where the application is installed.

Installation:
Insert the CD into the CD-ROM drive, and open the file *install.html*. Locate your particular operating system in this file for directions on how to install this software for your system. In cases where your system supports java applets, like Windows and MacOS, the installation is totally automated. In other formats, more work may be required.

***Un-installation*:**
1. If you are running Windows, you can uninstall this software using Add/Remove Programs in the control panel. This will uninstall the application, but you will still need to remove the application folder from the program files for any files that were written after installation (like log files).

2. For MacOSX, the system uninstaller can be used to remove this application. You will need to remove the application folder to delete any files written after installation (like log and text files).

3. For other operating systems, if system uninstallers are available, they can be used to remove this application. Otherwise, you will need to delete the directory where the application was installed manually.

SECTION 3: INTRODUCTORY MANUAL

To help you become familiar with the software, it is important that you understand the general options and features available to you.

3.1 Layout

3.1.1 Main Application Window

First, you will want to become familiar with the general layout of the software and its functions. When you first open and run the software, the main graph display window appears. This is the window where you will view the graphs you work with. This window will be blank. At the top there are six main menu selections: (1) a File Menu, (2) an Edit Menu, (3) an View Menu, (4) a Window Menu, (5) an Algorithms Menu, and (6) a Help Menu. Included in each menu are the various features you will need in working with graphs. (See Figure 1)

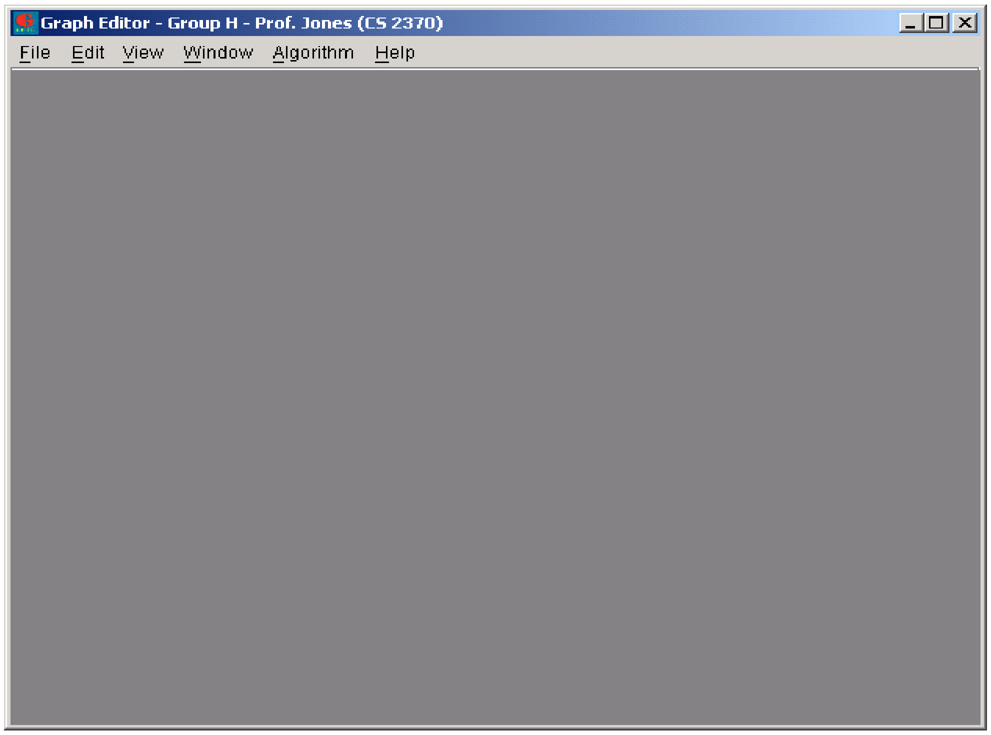


Figure 1. Main application window

Other primary windows used in the program include graph editing windows, a graph information window, a log window, a preferences window, and various help windows, all of which will be explained in greater detail in the following sections.

3.1.2 Connected Component Windows

Before you begin using this tool, it is important to note the way that connected components within a graph are represented. As you create and work with a graph, this program automatically separates individual connected components of the graph into separate windows. All of these windows are contained within the main application window. It is important to note that these windows do NOT denote separate graphs, but separate islands of the SAME graph.

This strategy helps to eliminate cumbersome graphs, as well as to provide the user with awareness of the connected components of a graph at all times. For example, the display below is a single graph that has 4 islands. (See Figure 2)

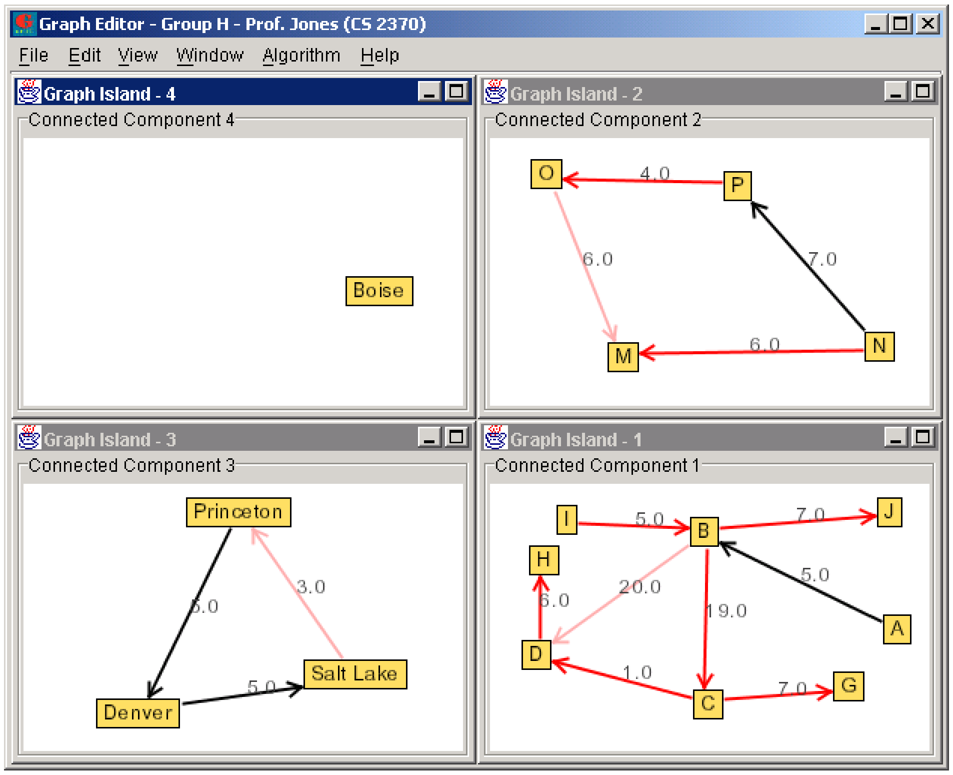


Figure 2. Example of four connected components within SAME graph

3.2 Creating/Editing a Graph

3.2.1 Creating a Graph

Probably the first thing that you will want to do is to create a graph. To do this, select the File Menu and click on 'New' in the pull down submenu that appears. Next a box will appear asking you to specify whether you want a directed or and undirected graph. Once you have selected what type it will be, you cannot change from directed to undirected or vice versa. (See Figure 3)

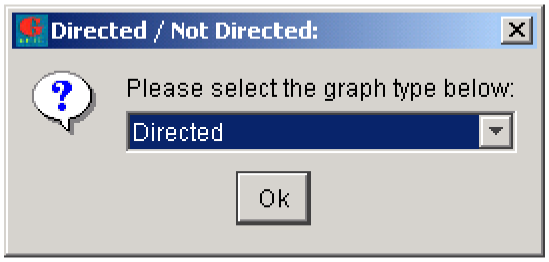


Figure 3. Dialog for selecting directed vs. undirected graph

3.2.2. Adding Nodes/Edges
After you select the directional state for your graph, a graph-island window will appear. This is where your graph will be displayed as you add nodes and edges. You are now ready to insert nodes and edges as desired to your graph. (*Note:* multiple island windows may appear and disappear frequently as you edit and change your graph. These changes will reflect the current connected-component identification of your graph at each step.)

Adding Nodes
To insert a node into your graph, select the 'Add Node' option in the Edit menu. This will bring up the addNode dialog box where you can enter the display name for the node. (See figure 4)

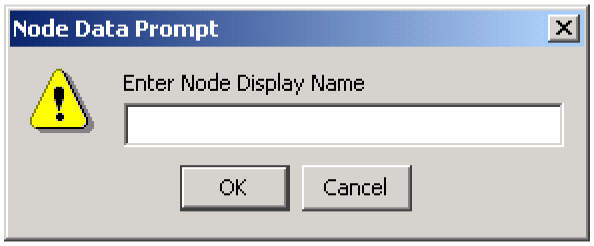


Figure 4: Add New Node dialog box

A more convenient way to add a node is to simply left click in any graph island window. (You may have to click twice depending on the operating system you are using.) This will bring up the add node dialog to create a node. You can insert as many nodes into the graph as you want.

Adding Edges
To insert edges into the graph, use the 'Add Edge' option in the Edit menu. After selecting 'Add Edge', a dialog box appears requesting edge information. There are three input fields in this dialog: the first for the source node, the second for the destination node, and the third for the edge weight. If the graph is undirected, it does not matter which node you identify as the source and which you identify as the destination. (See figure 5)

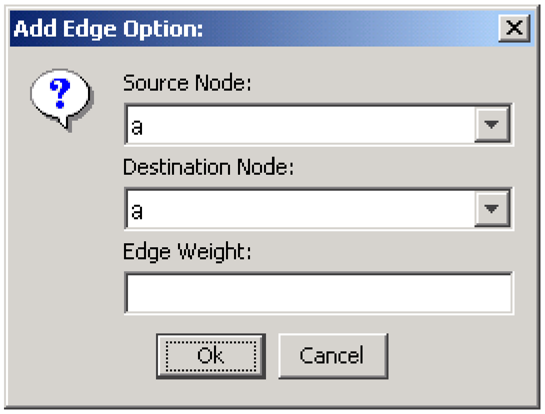


Figure 5: Add Edge dialog box, using menu selection method

A more convenient way to add an edge between two nodes works as follows. First, select the source node by left clicking a node. (You may have to click once or twice depending on your operating system. When a node is selected, it will change colors.) Immediately afterwards, select a destination node by left clicking it. These nodes are automatically inserted as the source and destination nodes for a new edge, and you will receive a dialog box prompting ONLY for the edge weight. (See figure 6)

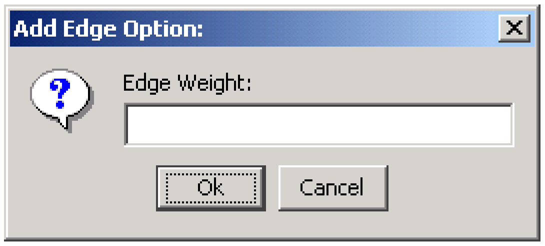
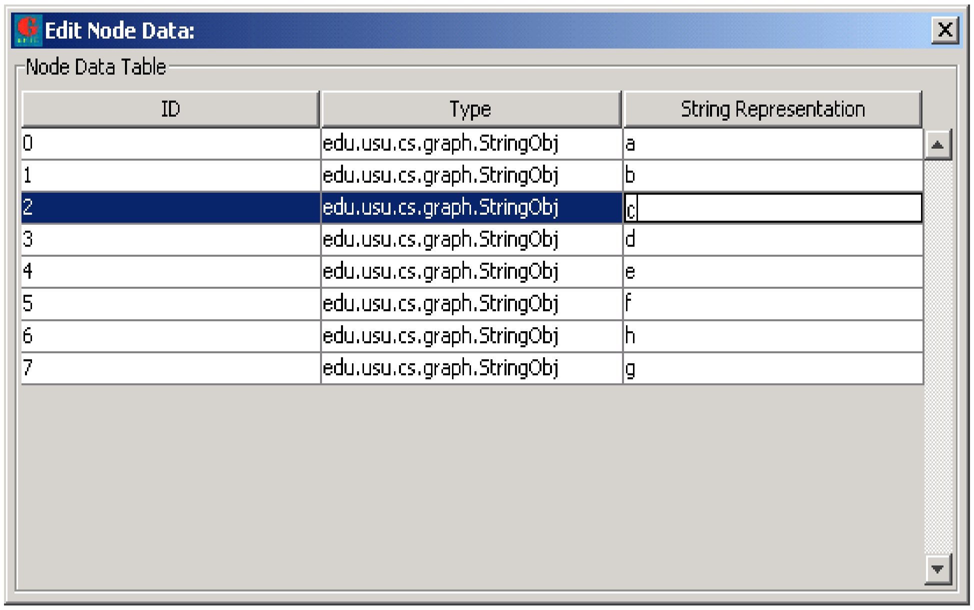


Figure 6: Add Edge dialog box, using mouse source-node and destination-node selection

You are prevented from inserting an edge that is not connected to a node, because the only nodes allowed for source and destination are those that already exist in the graph.

3.2.3 Editing Existing Nodes/Edges

Editing Node Data
To edit existing node data, select the 'Node Data' option in the Edit menu. This brings up node data window containing all of the nodes for the current graph. For any node that you want to change, double click the field containing its string representation (node name) and make changes accordingly. (See figure 7)

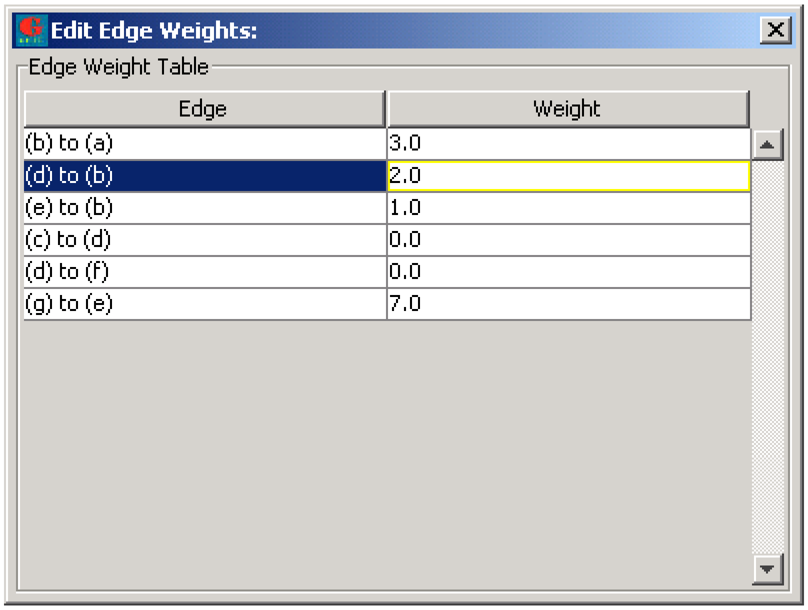


ID	Type	String Representation
0	edu.usu.cs.graph.StringObj	a
1	edu.usu.cs.graph.StringObj	b
2	edu.usu.cs.graph.StringObj	c
3	edu.usu.cs.graph.StringObj	d
4	edu.usu.cs.graph.StringObj	e
5	edu.usu.cs.graph.StringObj	f
6	edu.usu.cs.graph.StringObj	h
7	edu.usu.cs.graph.StringObj	g

Figure 7: Edit Node Data dialog window

Another way you can edit the data for a given node is to right-click on the node. A smaller dialog box pops up for editing only the single node's data. (Same dialog box as figure 4 above.)

Editing Edge Weights
To edit existing edge data, select the 'Edge Weights' option in the Edit menu. This brings up an edge weights window containing all edge data, including weights. For any edge weight you want to change, double click the field containing the edge weight and make changes accordingly. (See figure 8)



Edge	Weight
(b) to (a)	3.0
(d) to (b)	2.0
(e) to (b)	1.0
(c) to (d)	0.0
(d) to (f)	0.0
(g) to (e)	7.0

Figure 8: Edit Edge Weights dialog

3.2.4 Deleting Nodes/ Edges

Removing Nodes
In the case that you want to delete a node, first select a node and then choose the 'Remove Node' option in the Edit menu. This removes the node from the graph.

Removing Edges
In the case that you want to delete an edge, choose the 'Remove Edge' option in the Edit menu. A dialog box with a scrollable list of edges will appear. Select one of these edges and click "OK". This removes the edge from the graph. (See figure 9)

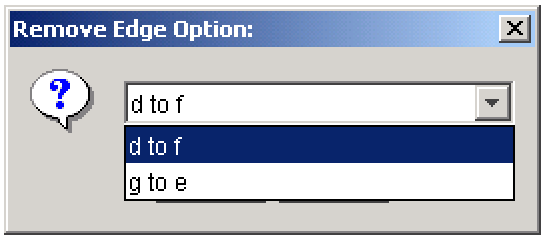


Figure 9: Remove Edge Dialog

3.2.5 Removing All Edges
To remove all edges from a graph, (with all nodes remaining), select the option 'Remove All Edges' in the Edit menu.

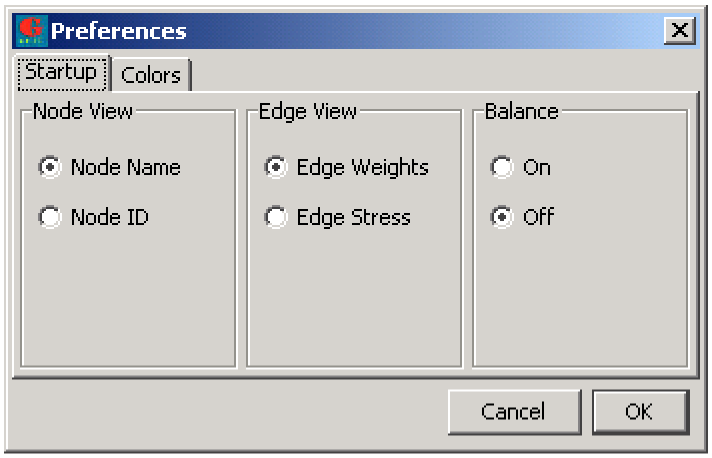
3.2.6 Deleting/Removing Entire Graph
To delete or clear an entire graph, select 'Close' or else 'New' in the File menu. In either case, before the current graph is removed, you will be prompted with a message about whether or not you want to save the current graph before clearing it, (if it is not already saved). If you choose to not save the graph, then the entire current graph will be deleted without opportunity for recovery.

3.2.7 Rearranging Nodes
At any time, you can select a node and drag it to wherever you want it within the graph island window where it is located. Thus, you can rearrange the graph to look how you want.

However, if you do not want to try to balance the graph yourself, select 'Balance Graph' in the View menu. This will cause the graph to try to reach an optimum balanced state. Such automatic balancing will continue at all times until you reselect 'Balance Graph', which acts as a toggle switch and disables automatic balancing. Even when auto balance is turned on, you can still drag and place nodes. Another way to control auto balancing is to set the 'Preferences' option in the Edit menu to the desired setting for balancing. (See section 3.3 and figure 10)

3.3 Preferences

The ‘Preferences’ option in the Edit menu allows you to control start-up and color preferences. Start-up preferences control the way nodes and edges are displayed, as well as whether auto-balancing is turned on or off. Color preferences select background, node, edge, and traversal colors as indicated. (See figure 10)



Node View: [✓ Node Name] Node ID

Edge View: [✓ Edge Weights] Edge Stress

Balance: On [✓ Off]

Figure 10: Preferences dialog

Following is a list of preferences with a description for each (refer to figure 10)

Node View Preferences
Nodes can be displayed by name or ID.

Edge View Preferences
Edges can be displayed by edge weight or by stress level. The stress level of a node has to do with the way the program automatically balances graphs. The longer the edge is on the display screen, the higher the stress value.

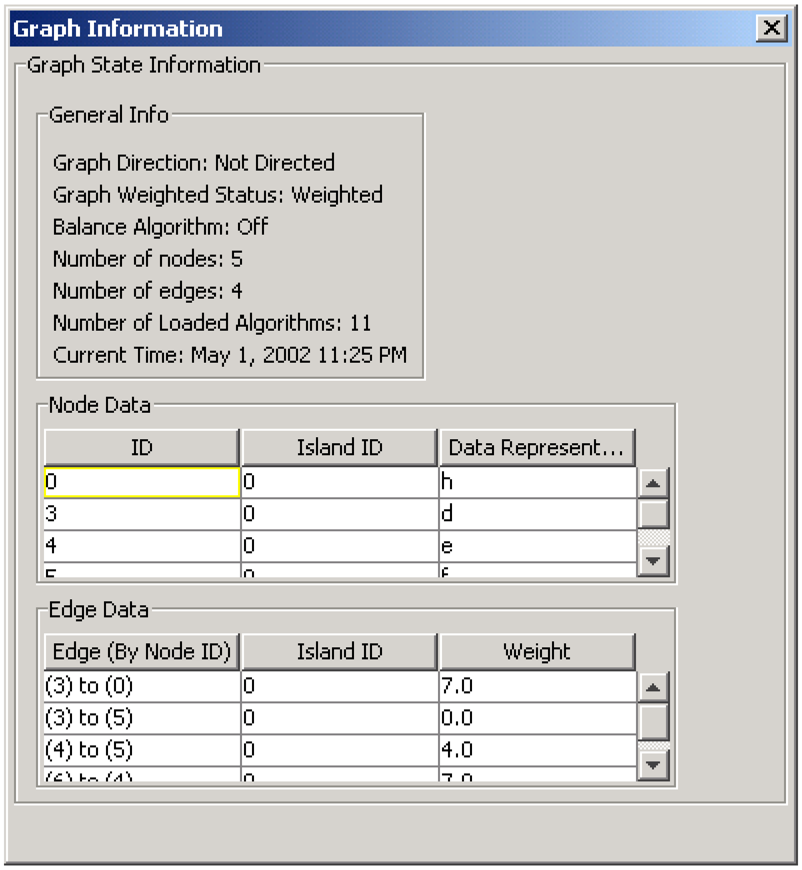
Balance Preferences
When balance is on, then the graph will balance itself to an optimum state, minimizing edge stress. When balance is off, the graph does not auto balance. Whether balance is on or off, at any time you can select a node and move it to any place you want in the graph island window where it is located.

Color Preferences
These preferences control colors for selected nodes, painting, backgrounds, etc. as indicated.

It is important to note that the preference window must be closed for graph editing.

3.4 Graph Information Window

To see a quick overview or summary of a current graph, select the ‘Graph Info’ option in the Help menu. This will bring up a graph information window containing a summary of the following graph attributes: graph direction, graph weighted-status, preference selection, and node and edge data information. (See figure 11)



Graph Direction: Not Directed
Graph Weighted Status: Weighted
Balance Algorithm: Off
Number of nodes: 5
Number of edges: 4
Number of Loaded Algorithms: 11
Current Time: May 1, 2002 11:25 PM

Node Data

ID	Island ID	Data Represent...
0	0	h
3	0	d
4	0	e

Edge Data

Edge (By Node ID)	Island ID	Weight
(3) to (0)	0	7.0
(3) to (5)	0	0.0
(4) to (5)	0	4.0

Figure 11: Graph Information window

3.5 Algorithms

3.5.1 Algorithm Tracing

Once you have created a graph (or opened a previous graph from file—see section 3.6 on file storage), you are now ready to trace through any algorithm that can be applied to a graph of the given type.

To execute an algorithm, select the Algorithms menu from the main menu. A pull down menu will appear containing the all algorithms provided by this software. Notice, only those algorithms that can be executed on the particular graph are available for selection. Not all algorithms can be executed on all types of graphs. (See section 3.5.2 for more information.)

To implement a given algorithm, simply select that algorithm from the pull down submenu. If you need to supply any information for the given algorithm, you will be prompted for it at this point. For example, you may be prompted to supply a source node for spanning tree. Figures 12 and 13 illustrate this.

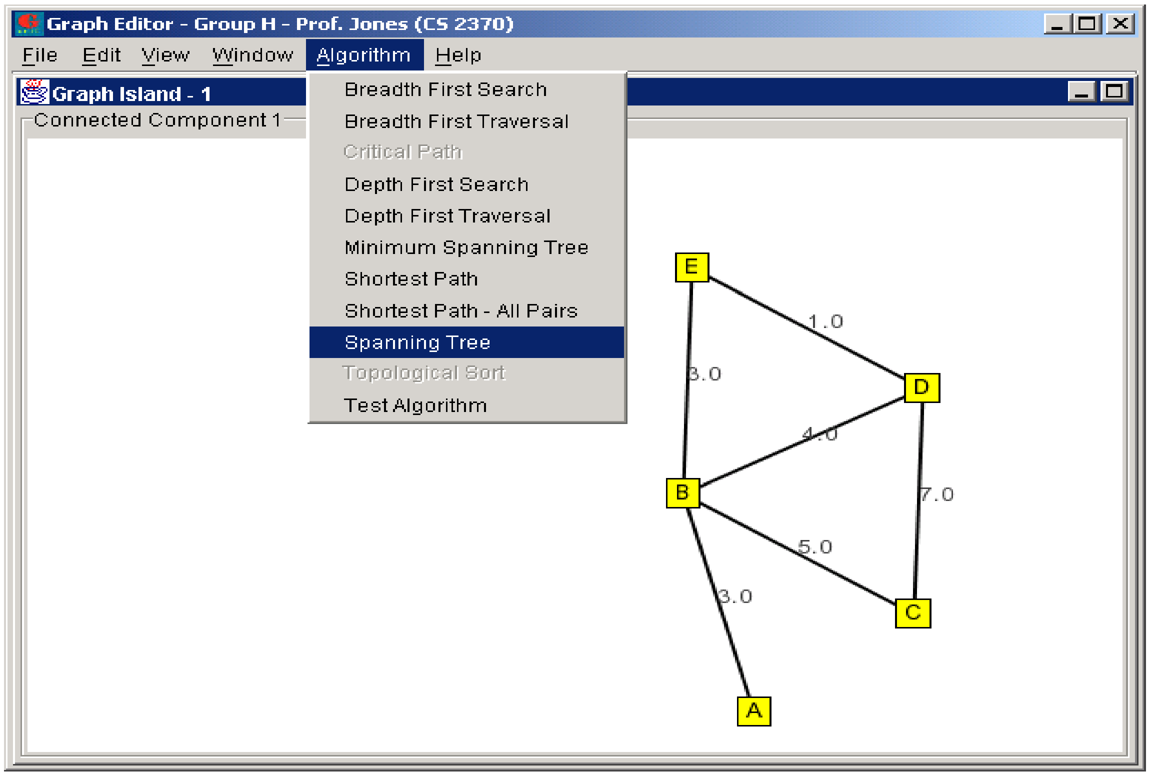


Figure 12: Selecting spanning tree algorithm for and undirected graph.

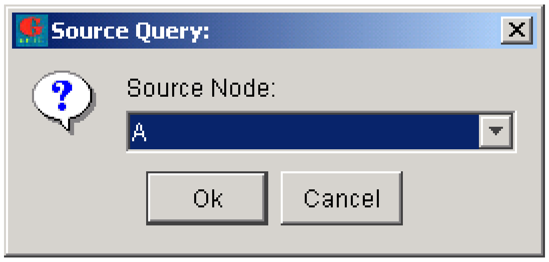


Figure 13: User prompt for source node for spanning tree.

After selecting a traversal and supplying any necessary initial algorithm information, the traversal controls window will now appear. See figure 14 below for an illustration of the algorithm traversal control window.

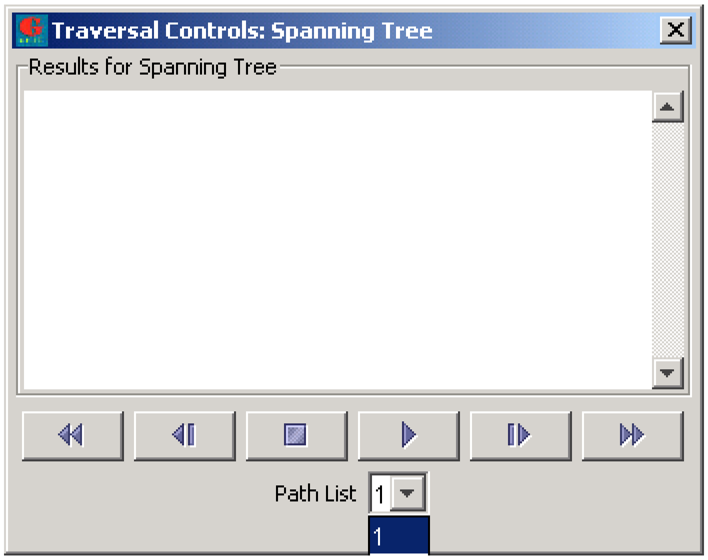
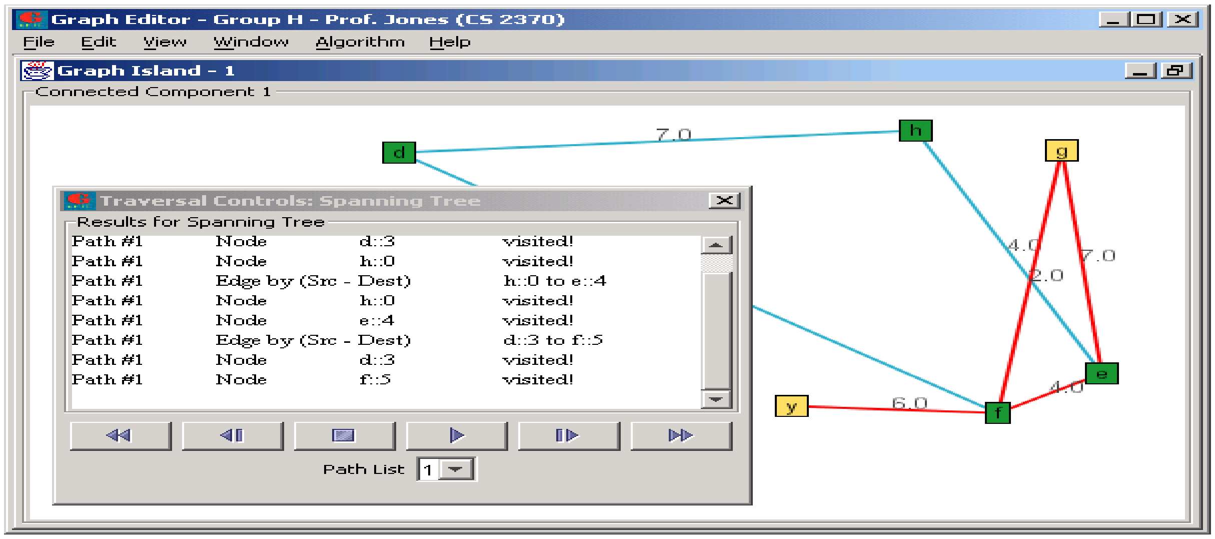


Figure 14: Traversal Control window

Notice the bottom field called ‘path list’ in figure 14. Clicking the arrow next to this field reveals the number of traversals or solutions that exist in the given graph for the algorithm you selected. You can scroll down and choose which of the several traversal solutions you would like to see traced on the graph. After viewing one solution, you can select another solution, and continue until you view them all.

The controls in this window act a lot like VCR controls. Clicking “|>” illustrates on the graph one step forward in algorithm execution. Similarly, “<|” moves one step backward. Clicking on “>>” paints the entire solution on the graph. Likewise “<<” returns the algorithm to the beginning before the traversal starts. The “>” button means “play.” Click it and you see an animation of the entire algorithm in real time execution, one step at a time, at a pace where you can follow the algorithm progression. Besides displaying the path on the graph, the algorithm traversal control window also reports the algorithm execution textually. See figure 15 for a snapshot of an algorithm traversal.


Graph Editor - Group H - Prof. Jones (CS 2370)
File Edit View Window Algorithm Help
Graph Island - 1
Connected Component 1
d
h
g
y
f
e
7.0
4.0
2.0
7.0
6.0
4.0
Traversal Controls: Spanning Tree
Results for Spanning Tree
Path List
1

```
Path #1        Node                    d::3           visited!
Path #1        Node                    h::0           visited!
Path #1        Edge by (Src - Dest)    h::0 to e::4
Path #1        Node                    h::0           visited!
Path #1        Node                    e::4           visited!
Path #1        Edge by (Src - Dest)    d::3 to f::5
Path #1        Node                    d::3           visited!
Path #1        Node                    f::5           visited!
```

Figure 15: Snapshot of spanning tree algorithm traversal

It is important to note that before graph editing can recommence, algorithm traversals must be discontinued and the traversal control window must be closed.

3.5.2 Algorithm – Graph State Compatibility

Any graph has two characteristics concerning the state of its edges. The first has to do with edge direction (1), and the second deals with edge weights (2).

1. Directed versus Undirected Graphs:
a. *Directed Graphs*: In a directed graph, each edge can be traversed in only one direction, specified by an arrow.
b. *Undirected Graphs*: In an undirected graph, each edge can be traversed in either direction between the two nodes it connects.

2. Weighted versus Non-weighted Graphs:
a. *Weighted Graphs*: In a weighted graph, associated with each edge is a value representing the cost or distance necessary to travel between the two nodes it connects.
b. *Non-weighted Graphs*: In a non-weighted graph, each edge merely denotes the fact that two edges are connected, with no associated weight value.

Each graph algorithm requires specific edge states in order to execute. For example, the spanning tree and minimum spanning tree algorithms require undirected graphs, while the shortest path and all-pairs shortest path algorithms require directed graphs. For this reason, only certain algorithms can be executed on any given graph, depending on its edge states. (For a listing of the states required for any specific algorithm, see Section 4.2 of this document.)

3.5.2 Adding/Removing Your Own Algorithms

It is possible to add your own graph algorithms to this application. You must meet specified requirements in order to do so. Because these requirements are somewhat complex and go beyond simple use of this application, they are not included in this introductory section. For details, see the Reference Manual (Section 4.3 of this document).

<u>3.6 File Storage</u>

3.6.1 Graph File Storage

Save to file
If you want to save the graph for future reference, select 'Save' or 'Save As' in the File menu, much as you would for most applications. A dialogue box will appear, requesting the pathname and filename for where you want the file stored. (See figure 16, on next page)

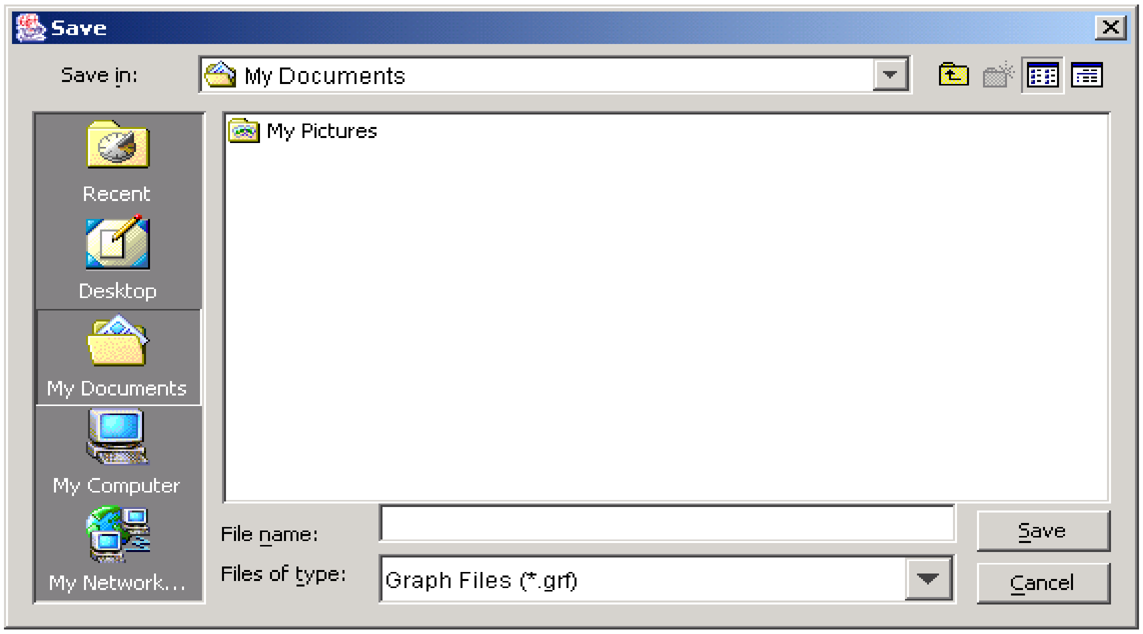


Figure 16: Save/Save As dialog window

Open from file
Similarly, to open any graphs already saved to file, choose 'Open' in the File menu and (1) either type in the path and file name of the graph you want to open, or (2) use the browse buttons to find and select the file containing the graph. (See figure 17)

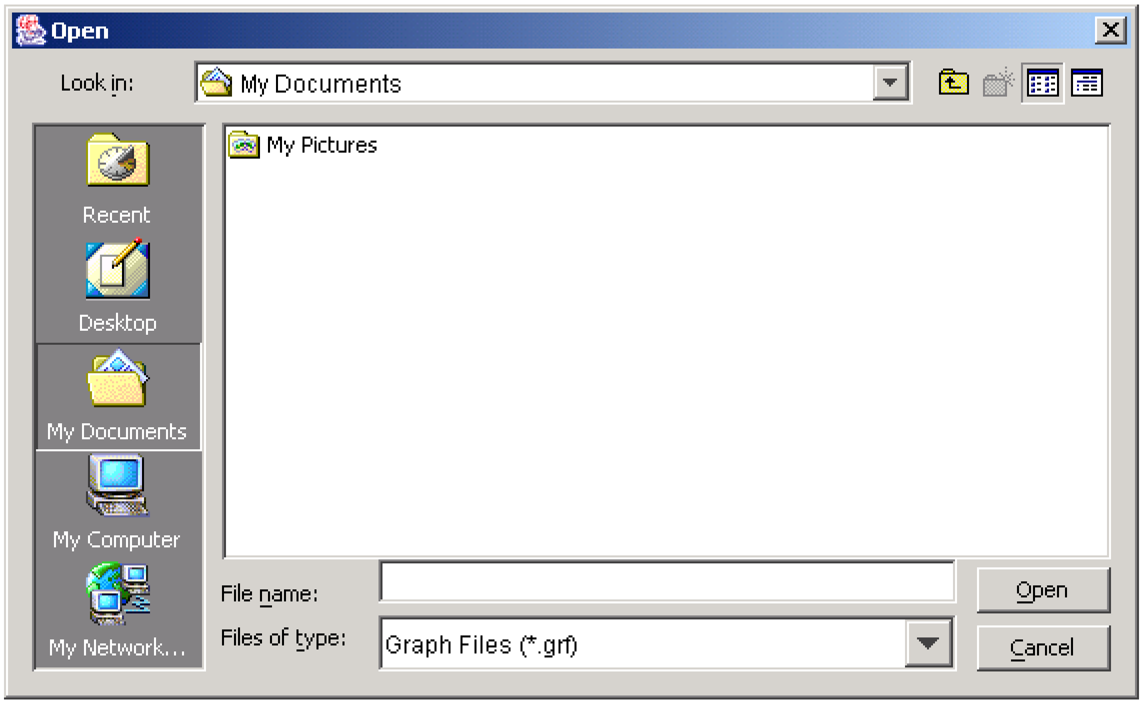


Figure 17: Open dialog window

3.6.2 Saving an Island as a Separate Graph

Any individual connected component of a graph can automatically be saved as a separate graph. To do this, right-click in any island window (away from any node) and select the 'Save Island as Graph' option that appears. (See figure 18 below) When you select this option the save/save as dialog box (figure 16) will appear, requesting a pathname for the file where the new graph will be contained. Enter a pathname and the island will be saved as a separate graph.

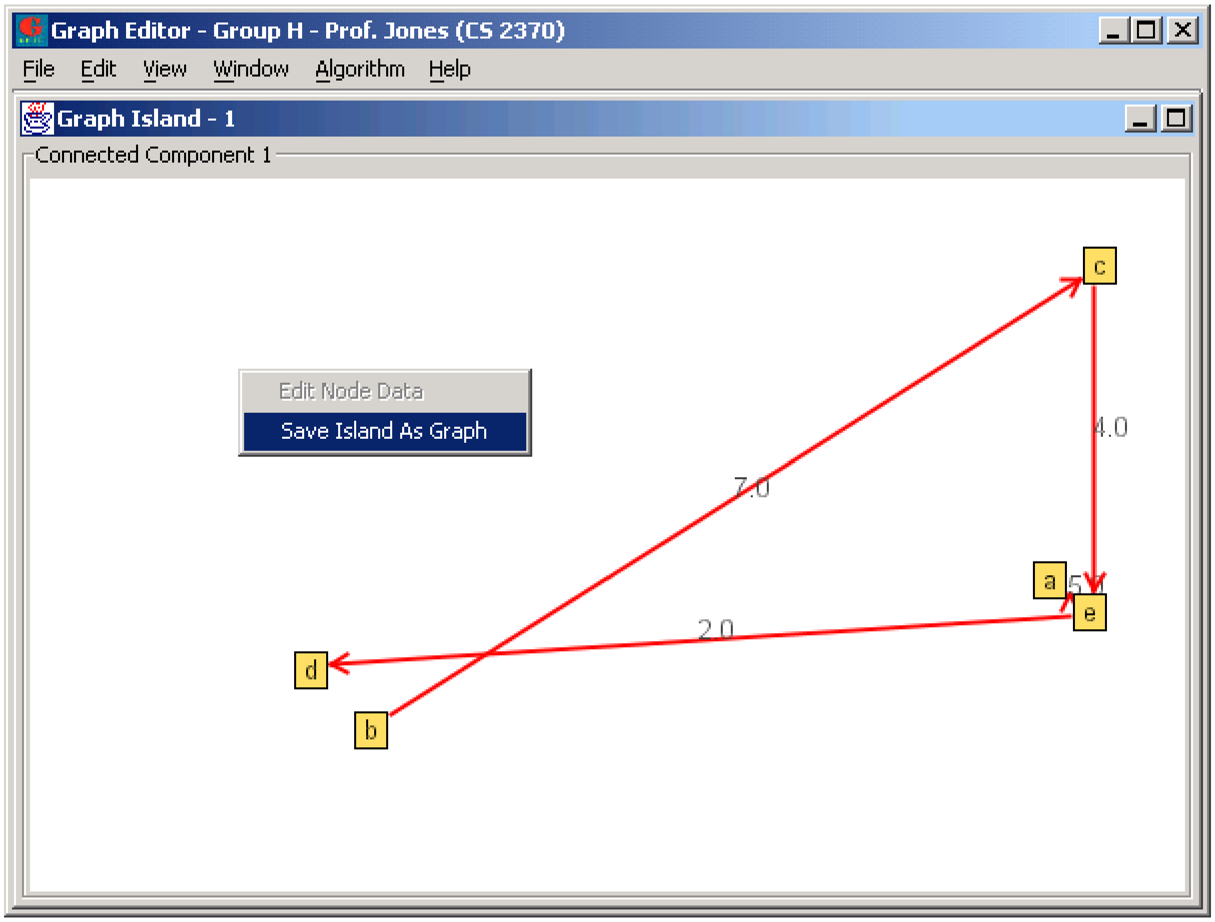
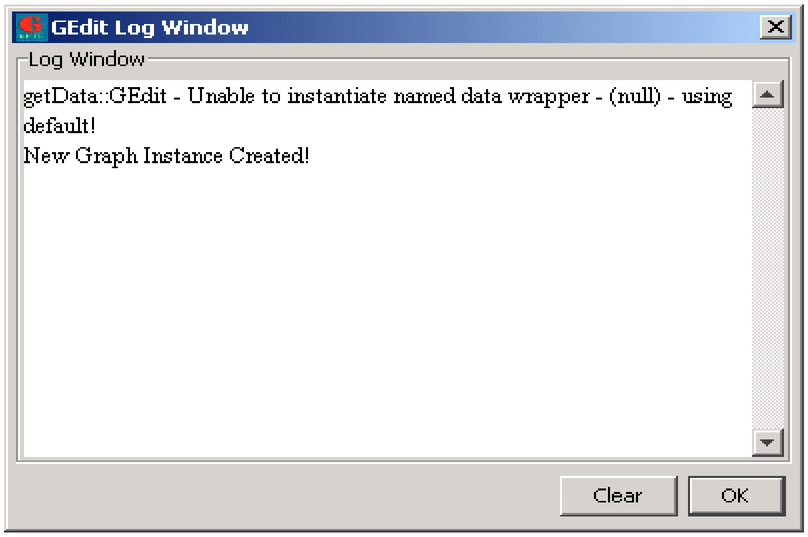


Figure 18: Save Island as Graph function

3.7 Log Window

A running log of application operations will be kept during execution of this program. Error messages, graph information, and user activities will be logged here. To view this window at any time, select 'Log Window' in the Windows Menu. (See figure 19)



```
getData::GEdit - Unable to instantiate named data wrapper - (null) - using
default!
New Graph Instance Created!
```

Figure 19: Log Window

3.8 Help Documentation

To access various help documentation for this program, select Help from the main menu. (See figure 20)

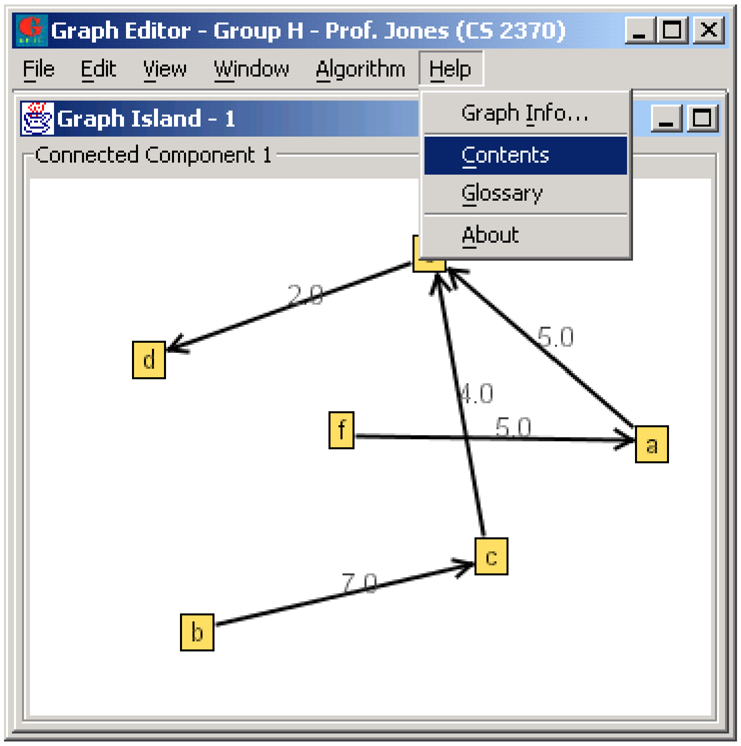


Figure 20: Help Menu selections

Options in this menu include the following:

1. *Graph Info*, which displays all current graph contents and state attributes in a summary window (see section 3.3).
2. *Contents,* which includes the main help files for assistance in running this application.
3. *Glossary*, which contains a glossary of common graph definitions and terminology.
4. *About*, for names of software developers involved in this project.

SECTION 4: REFERENCE MANUAL

This manual offers more in-depth descriptions of certain software features that may be important to you in using this program. It includes how to use any data type for nodes, more information about built-in graph algorithms as well as directions for adding new algorithms, and potential error messages that you may encounter.

4.1 User-Defined Data Types as Nodes

This graph data tool allows a programmer to put user-defined objects in nodes.

To integrate such user-defined nodes, a basic understanding of *java, including packages, and jar files* is necessary. Such knowledge is assumed throughout the rest of this explanation.

Directions for using user defined data types:
1.First, the user defined object must implement *edu.usu.cs.graph.Data*. This is an interface between your algorithm and our application. See the java doc, and make sure that you implement all the required methods as defined by this interface..

2. Run the application with a *-d <classname>* parameter. Example: If your class name is *com.yourcompany.YourObject*, then you would use
-d com.yourcompany.YourObject.

3. This allows your data type to correctly interface with the graph. Currently, this application only supports one user-defined data type per instance of the application.

To see an example of code for a data type with two fields that has successfully implemented this interface to work with this application, see the appendix of this document.

4.2 Built-in Algorithm Details

As the main goal for this software is to trace through graph algorithms, following is a list of the algorithms that are available in the Algorithm menu. Along with each algorithm is listed the appropriate graph type, as well as input you must provide (if necessary) for that algorithm. (Similar information is also available in the online-help.)

Algorithm	Graph Type	Input Required
Depth-first traversal	Undirected graphs	Starting Node
Breadth-first traversal	Undirected graphs	Starting Node
Depth-first search	Undirected graphs	Search Value and Starting Node
Breadth-first search	Undirected graphs	Search Value and Starting Node

Algorithm	Graph Type	Input Required
Topological sort	Directed graphs	None
Spanning tree	Undirected graphs	Root Node
Minimum spanning tree	Weighted undirected graphs	Root Node
Shortest path	Weighted directed graphs	Source and Destination Nodes
All-pairs shortest paths	Weighted directed graphs	None
Critical path	Weighted directed graphs	None
Connected component identification	Undirected graphs	None

Note: For all algorithms, when multiple solutions satisfy a given algorithm, only one solution is displayed at a time. You are given the option to run each of all possible traversal solutions for each algorithm. (See Section 3.5 in this document.)

4.3 Adding/Removing Your Own Algorithms

This graph data tool allows a programmer to add customized additional graph algorithms.

To add such user-defined algorithms, a basic understanding of *java, including packages, and jar files* is necessary. Such knowledge is assumed throughout the rest of this explanation.

Directions for algorithm addition:
1.First, create a new class that extends *edu.usu.cs.algorithms.Algorithm*. This is an abstract class that serves as an interface between your algorithm and our application. See the java doc, and make sure that you implement all the required methods as defined by this abstract class.

The method '*doAlgorithm*()' needs to return a PathContainer that contains at least one Path in order for the traversal window in the GUI to come up. Multiple Paths can be added to the PathContainer for execution of a single algorithm.

In order to create a Path, you must pass to the Path constructor two arrays of equal length: (1) an array of nodes, and (2) an array of edges. Corresponding array indices, one from the node array and one from the edge array, are considered a node-edge pair. Both node indices and edge indices can be set to NULL.

This software considers one step in an algorithm progression to be represented by the ordering of all consecutive node-edge pairs until both the node and the edge of a given pair are NULL. Thus, make sure you parse individual algorithm traversal steps in a path by a NULL-NULL pair. The GUI will paint everything it encounters as one step in a path until it runs into a node-edge pair where both are NULL.

Say, for example, that you want the first step of an algorithm to paint two nodes and one edge. The path you would create would contain a node-edge pair where the node is one of the nodes you want to paint, and the edge is the edge you want to paint. Then the next node-edge pair would contain the other node you want to paint, and a NULL edge. The next node-edge pair in the path would contain both a NULL node and a NULL edge. The GUI would then paint everything up to the NULL-NULL pair as the first step in the traversal, lighting up the two nodes and one edge as desired.

2.After you designed your algorithm according to these guidelines, make it a class in the package *edu.usu.cs.algorithms* and add it to Gedit.jar in the folder where this application is installed.

3. In the same folder as this application is a GEdit.properties file. You need to open this file and modify it by adding your algorithm class name to the property GEDIT.ALGORITHMS (with no package modifier). Save this file with these changes.

4. This completes the addition, and your algorithm will now show up in the menu bar and should execute according to your design.

Directions for algorithm removal:
In order to remove an algorithm, go back to the GEdit.properties file and delete your algorithm class name. This will remove your algorithm from the application.

If you have any problems or concerns adding or removing an algorithm, please contact us at the following email address: internet@secristfamily.com.

4.4 Error Messages
By design, many potential user errors have been eliminated. However, there are certain errors and error messages of which you should be aware when using this software.

1. *File input and output*: If you try to open a graph from file that is not at the specified location, or if you try to read a file that is not a graph file (or a graph file that is corrupted), you will receive the following error message:

Unable to access file.

Further information will be passed depending on the file error.

If you attempt to write to a file that is write protected or denied network permission, you will receive one of the following error messages:

File not found. Or

I/O exception

Further information will be passed depending on the file error.

2. *Illegal node insertions:* If you try to insert a node with the same name as a previously existing node, you receive the following error message:

We apologize, but there was an error adding a node.
Duplicate node.
The node will not be added. Add another node with a different name.

3. *General Application Errors:* Should the program encounter an unforeseen error, the following message will be displayed:
General Error.
Further information will be passed depending on the nature and the location of error.

It is useful to note that all error handling is recorded in the application log window. So if you ever wish to investigate an error that occurred during program run time without exiting the program, check the log window. (see Section 3.7)

SECTION 5: ADMINISTRATOR'S MANUAL

This application is not administrated software. So this manual is omitted in this version.

APPENDIX
Example of User-Defined Data Type Used for Graph Nodes

The following code for a user defined data type called MunicipalData successfully implements the interface necessary for to work with this application:

```
package edu.usu.cs.graph;
import java.util.StringTokenizer;
import javax.swing.*;

public final class MunicipalData implements Data, java.io.Serializable {
        private String cityName;
        private int cityPopulation;
/**
 * MunicipalData constructor comment.
 */
public MunicipalData(String delimited) {
        super();
        // Tokenize delimited string looking for two parameters.
        StringTokenizer token = new StringTokenizer(delimited, ",");

        // If this container does not contain 2 tokens, return an empty set of data.
        if (token.countTokens() != 2) return;

        // Please note that more robust code would better handle NumberFormateExceptions -
etc.
        this.cityName = token.nextToken();
        this.cityPopulation = new Integer(token.nextToken()).intValue();
}
        /**
         * Specifies how one MunicipalData is equals to another MuniciplaData
         */
        public boolean equals(Data x) {
                if (x instanceof MunicipalData) {
                        MunicipalData n = (MunicipalData) x;
                        boolean cityEqual = this.cityName.equalsIgnoreCase(n.cityName);
                        boolean popEqual = (this.cityPopulation == n.cityPopulation);
                        if (cityEqual && popEqual) return true;
                        else return false;
                }
                else {
                        return false;
                }
        }
        /**
         * Returns the display name for this object.
```

```
	 */
	public String getDisplayName() {
		return cityName;
	}
	/**
	 * The string representation of this object.
	 *
	 * This is used to reconstruct an instance of this object by
	 * calling the String Constructor.
	 */
	public String toString() {
		return cityName + "," + cityPopulation;
	}

/**
 * This constructor should never be called directly.
 * Creation date: (3/5/2002 8:46:37 AM)
 */
public MunicipalData() {
	super();
	this.cityName = "NULL";
	this.cityPopulation = 0;
}

	/**
	 * Specified how a new Data object can be created from a String.
	 */
	public Data getInstance(String val) {
		return new MunicipalData(val);
	}

	/**
	 * Displays a Swing JDialog Input for this Data Type.
	 */
	public String showInputDialog(javax.swing.JFrame parent) {
		// Messages
		Object[] message = new Object[4];
		message[0] = "City Name:";
		JTextField nameTxt = new JTextField();
		nameTxt.setDoubleBuffered(true);
		message[1] = nameTxt;

		message[2] = "City Population:";
		JTextField popTxt = new JTextField();
		popTxt.setDoubleBuffered(true);
		message[3] = popTxt;

		// Options
```

```
                String[] options = {
                        "Ok",
                        "Cancel"
                };
                int result = JOptionPane.showOptionDialog(
                        parent,                         // the parent that the dialog blocks
                        message,                        // the dialog message array
                        "New Municipal Data:",                  // the title of the dialog window
                        JOptionPane.DEFAULT_OPTION,             // option type
                        JOptionPane.QUESTION_MESSAGE,           // message type
                        null,                           // optional icon, use null to use the default
icon
                        options,                        // options string array, will be made into
buttons
                        options[0]                      // option that should be made into a default
button
                );


                // Ensure we got a number for population
                int population;
                if (popTxt == null || popTxt.getText().length() <= 0 ||
popTxt.getText().equalsIgnoreCase("")) population = 0;
                else population = Integer.parseInt(popTxt.getText());

                switch(result) {
                  case 0: // yes
                    return new String(nameTxt.getText() + "," + population);
                  case 1: // cancel
                    break;
                  default:
                    break;
                }
                return null;
        }
}
```