COMP 2710

Software Construction

Lab 3

Robinson Davis

November 3, 2014

1. Analysis

This program will provide the functionality to traverse a graph in the shortest route possible. This graph specifically contains nodes with four neighbors. Users will be prompted to enter a configuration file from which this graph will be built. From there the user is able to choose a beginning and end node, and will be informed of the shortest path between those two nodes among the graph.  
  
2 Design

There will be numerous classes to help separate the functionality of this program. They are defined as follows:

1. **Node class** – Used to maintain the nodes on the graph.  
   1. Variables
      1. string name;
         1. Name of the current node.
      2. Node \*attachedNodes[4];
         1. All nodes attached to current node (using pointers.)
      3. int numberUsed;
         1. Number of sides used for a current node.
      4. Node \*previous;
         1. The previous node.
      5. int marked;
         1. Used to mark the node while travesing
   2. Functions
      1. void setNodeName(string newName);
         1. Sets the parameter name as the name of the current node.
      2. string getNodeName();
         1. Returns the current node’s name.
      3. void attatchNewNode(Node \*newNode, int index);
         1. Attaches a new parameter node to the current node at the parameter index.
      4. Node \*getAttachedNode(int index);
         1. Returns the attached node at the given index.
      5. int getNumberUsed();
         1. Returns the number of sides used.
      6. int setNumberUsed(int newNumberUsed);
         1. Sets a new parameter number of sides used.
      7. void setPrevious(Node \*previousNode);
         1. Sets the previous node to the parameter node.
      8. Node \*getPrevious();
         1. Returns the previous node of the current node.
      9. int setMarked();
         1. Set current node as marked.
2. **Solver class -** Handles functions to build and solve the maze.  
   1. Variables
      1. int numOfNodes;
         1. Number of nodes in the configuration file.
      2. Node\* nodeArray;
         1. Dynamic array that will maintain the node graph.
      3. String file;
         1. Current configuration file.
      4. String startNode;
         1. Starting node of maze.
      5. String endNode;
         1. Ending node of maze.
      6. Vector path;
         1. A vector that will maintain the shortest path.
      7. Vector visited;
         1. A vector that will maintain all nodes visited.
   2. Functions
      1. void getFile();
         1. Gets the current file from the user.
      2. void buildArray();
         1. Builds nodeArray from given input file.
      3. void getMazePath();
         1. Gets the startNode and endNode variables from the user.
      4. void traverse(startNode, endNode);
         1. Displays the shortest path between the parameter start and end nodes.
3. **System class -** Used to run the program.
   1. Variables
      1. None.
   2. Functions
      1. run();
         1. Used to run the program.
4. **Error class -** Used to maintain different error outputs.
   1. Variables
      1. None.
   2. Functions
      1. void noFileExists();
         1. Given configuration file does not exist.
      2. void noPathExists();
         1. No path exists between the two nodes.
      3. void noNodeExists();
         1. A given node does not exist.

2.1 Class Diagram

2.2 Data Flow Diagram

3 Tests  
  
 Effective testing for this message board can most easily be obtained through case testing. That is, testing each usage case and ensuring that proper output is achieved.

1. **Testing**
   1. Enter an invalid file, ensure that user is notified and re-prompted.
   2. Enter a valid file and ensure that information is read in correctly.
      1. This involves checking the file name as well as the dynamic array of nodes.
      2. Nodes must be connected by pointers.
         1. Check for proper pointer usage.
   3. Ensure that start and end nodes are received correctly.
      1. These nodes must both exist.
      2. These nodes must be connected.
   4. Traverse map
      1. Ensure that all visited nodes a properly marked and recorded.
      2. Ensure that the shortest path amongst these nodes is recorded.
      3. Ensure that both bits of information is correctly printed for the user.
   5. Repeat
      1. Ensure that each sequential file is read in, manipulated, and printed correctly.
   6. Quit
      1. Ensure that the file name “Quit” ends program execution.