CS 232 - P2

Programmer Manual

By: Jeffrey Marron for B. Streller’s Spring 2016 CS232 class

Contents:

I. cs232\_p2\_client\_driver.cpp

II. graph.t

I. - cs232\_p2\_client\_driver.cpp

The driver first initializes the main Graph and then enters a switch case command system operated via user input.

II. - graph.t

This file specifies the templated implementations of the graph container class.

void Graph::isVertex()

Asks the user to input the name of a vertex to look up which is stored in string temp. The adj\_list is then iterated from the beginning through iterator Vit. If temp is equal to the name value of Vit then it will output “is a valid vertex”

void Graph::isUniEdge()

Asks the user to input the name of a starting and ending verticies which are then stored into strings temp1 and temp2. The adj\_list is then iterated from the beginning through iterator Vit. If temp1 is equal to the name value of Vit then it will iterate through the edgelist in that vertex through iterator ERit. If temp2 is equal to the name value for ERit then a uni directional edge exists.

void Graph::isBiDirEdge()

Asks the user to input the name of two verticies. Operates the same as void Graph::isUniEdge()

however the process is repeated with temp2 and temp1 switched. If there’s an edge from temp1 to temp2 and temp2 to temp1 then a bi directional edge exists.

void Graph::AddVertex()

Asks the user to input the name of a new vertex which is stored in string temp. Vit iterates through the adj\_list and compares temp to the name value of Vit. If its not in the graph, temp is inserted into tempVert and pushed into the adj\_list using vector push back.

void Graph::DeleteVertex()

Asks the user to input the name of a new vertex which is stored in string temp. Vit iterates through the adj\_list and compares temp to the name value of Vit. If temp is equal to the name value of Vit, Vit is erased from the adj\_list using vector erase.

void Graph::AddUniEdge()

Asks the user to input the name of a starting and ending verticies and weight which are then stored into strings temp1, temp2, and int weight. Vit and Erit iterate through the graph to determine if an edge already exists. If the starting vertex is found then temp2 and the weight are put into the tempEdgeRep which is then pushed back into the graph using the vector push back. Then Vit iterates again to determine if the second vector is in the graph and creates a new vector if necessary. If the start vertex was not found, then a new start vertex is created and inserted into the graph. Then Vit iterates again to determine if the second vector is in the graph and creates a new vector if necessary.

void Graph::DeleteUniEdge()

Asks the user to input the name of a starting and ending verticies which are then stored into strings temp1 and temp2. Vit and Erit iterate appropriately and then Vit is erased from the edgelist using vector erase. If the edge wasn’t found in the graph then an error message displays.

void Graph::AddBiDirEdge()

Asks the user to input the name of two verticies and weights which are then stored into strings temp1, temp2, and int weight1, weight2. Sub scripting is used instead of iterators in order to keep track of relative graph coordinates for later operations. If temp1 is in the graph then v1 is set to true. If temp2 is in the graph then v122 is true(edge from vector 1 to vector 2). The same test is performed again to test if v2 is in the graph and if an edge goes from vector 2 to vector 1. A series of if else statements evaluates the cases of vector 1 existing and not 2, vector 2 existing and not 1, if vectors 1 and 2 both exist, or if neither vectors exist. The statements also have cases for an edge from vector 1 to 2 but not 2 to 1, 2 to 1 but not 1 to 2, and if both 1 to 2 and 2 to 1 exist. Each case handles differently and adds new vectors to the graph where necessary.

void Graph::DeleteBiDirEdge()

Asks the user to input the name of two verticies which are then stored into strings temp1 and temp2. Vit and Erit iterate through the graph twice which alter the bool values v122 and v221. If there is an edge from vector 1 to 2 and there is an edge from vector 2 to 1, then it is a bidirectional edge. Both edges are then erased using the vector erase.

void Graph::SimplePrintGraph()

Prints out the graph by iterating using Vit and Erit.

void Graph::ShortestDistance()

Asks the user to input the name of a starting and ending vertex which are then placed in string temp1 and temp2. Attempts to use Dijkstra’s algorithm to determine the shortest distance between the two verticies.

void Graph::BFTraversal()

Asks the user to input a start vertex. A temporary vector visited is created to keep track of the visited nodes. The adj\_list is then iterated to where temp is equal to the specified node whose location is stored in int a. That location is then marked as visited. A while loop then checks the adj\_list location at “a” and walks down the edge list and marks each node as visited. The front is outputted and then the queue is popped. A new “a” value is then found and the while loop continues.

void Graph::DFTraversal()

Asks the user to input a start vertex. A temporary vector visited is created to keep track of the visited nodes. The adj\_list is then iterated to where temp is equal to the specified node whose location is stored in int a. A while loop then checks the adj\_list location at “a”. The adj\_list is iterated until the “a” is found which is then pushed into the stack. The new target is then set to the top, the top is printed to the console, and the top is then popped. The adj\_list is then iterated again to find the new “a” value to start from.

void Graph::GetGraph()

Parses the input file using the ‘#’ character as a delimiter. Assigns the appropriate values into tempEdgeRep and then pushes back into the edgelist using vector push back. Once the delimiter is reached the tempVert is pushed back into the adj\_list using vector push back. The infile is then closed.

string Graph::getFileName(ifstream &, string & inFilename)

Gets the input file name from the user.