/\*

Daniel Frey

CIS 350

Prog 3

This program will take an undirected graph with weighted edges, create a MST and keep it updated with user directives

\*/

#include<iostream>

#include<fstream>

#include<string>

#include<unordered\_map>

#include<utility>

#include<vector>

#include<conio.h>

#include<cfloat>

#include<algorithm>

using namespace std;

ofstream outFile;

class Prim;

class Graph

{

friend class Prim;

private:

float \*\*adjacencyMatrix;

int vertexCount;

public:

~Graph();

unordered\_map<string, int> vertices;

void buildGraph(int size);

int getSize();

void addEdge(string V1, string V2, float weight);

int lookupVertex(string vertex);

bool isEdge(string V1, string V2);

void printGraph();

void showDirectives();

void directivesInput(Prim MST);

void directiveFile(ifstream &directiveFile, Prim MST);

void addVertex(string vertex);

void deleteVertex(string vertex);

};

Graph::~Graph()

{

int i;

for (i = 0; i < vertexCount; i++)

{

delete[] adjacencyMatrix[i];

}

delete[] adjacencyMatrix;

}

void Graph::buildGraph(int size)

{

int i, j;

vertexCount = size;

adjacencyMatrix = new float\*[size];

for (i = 0; i < size; i++)

{

adjacencyMatrix[i] = new float[size];

for (j = 0; j < size; j++)

adjacencyMatrix[i][j] = 0;

}

}

int Graph::getSize()

{

return vertexCount;

}

void Graph::addEdge(string V1, string V2, float weight)

{

int i, j;

i = lookupVertex(V1);

j = lookupVertex(V2);

if ((i >vertexCount || i < 0) || (j > vertexCount || j < 0))

{

cout << "Edge contains invalid vertex." << endl;

outFile << "Edge contains invalid vertex." << endl;

}

else if (adjacencyMatrix[i][j] == 0 && adjacencyMatrix[j][i] == 0)

{

adjacencyMatrix[i][j] = weight;

adjacencyMatrix[j][i] = weight;

cout << "Added edge (" << V1 << "," << V2 << ") with weight " << weight << "." << endl;

outFile << "Added edge (" << V1 << "," << V2 << ") with weight " << weight << "." << endl;

}

else

{

cout << "Edge (" << V1 << "," << V2 << ") already exists." << endl;

outFile << "Edge (" << V1 << "," << V2 << ") already exists." << endl;

}

//printGraph();

}

int Graph::lookupVertex(string vertex)

{

for (unordered\_map<string, int>::iterator ii = vertices.begin(); ii != vertices.end(); ++ii)

{

if ((\*ii).first == vertex)

return (\*ii).second;

}

return -1;

}

bool Graph::isEdge(string V1, string V2)

{

int i, j;

i = lookupVertex(V1);

j = lookupVertex(V2);

if (i == -1 || j == -1)

{

cout << "Invalid vertex." << endl;

outFile << "Invalid vertex." << endl;

return false;

}

if (adjacencyMatrix[i][j] != 0 && adjacencyMatrix[i][j] != 0)

return true;

else

return false;

}

void Graph::printGraph()

{

cout << endl;

outFile << endl;

unordered\_map<string, int>::iterator ii = vertices.begin();

cout << "Adjacency Matrix Graph" << endl;

cout << "(Only left side identifiers are displayed)\n" << endl;

outFile << "Adjacency Matrix Graph" << endl;

outFile << "(Only left side identifiers are displayed)\n" << endl;

for (int i = 0; i < vertexCount; ++i)

{

cout << (\*ii++).first << " ";

outFile << (\*ii++).first << " ";

for (int j = 0; j < vertexCount; ++j)

{

cout << adjacencyMatrix[i][j] << ' ';

outFile << adjacencyMatrix[i][j] << ' ';

}

cout << endl;

outFile << endl;

}

}

void Graph::addVertex(string vertex)

{

int i, j, exists;

float \*\*arrInc;

exists = lookupVertex(vertex);

if (exists == -1) //if vertex does not exist

{

if (vertexCount + 1 <= 100) //if adding 1 will go over max

{

vertexCount = vertexCount + 1;

arrInc = new float\*[vertexCount];

for (i = 0; i < vertexCount; i++) //allocate space of new size

{

arrInc[i] = new float[vertexCount];

}

for (i = 0; i < vertexCount - 1; i++) //copy from old to new

{

for (j = 0; j < vertexCount - 1; j++)

arrInc[i][j] = adjacencyMatrix[i][j];

}

for (i = 0; i < vertexCount; i++) //fill brand new cells

{

arrInc[i][vertexCount - 1] = 0;

arrInc[vertexCount - 1][i] = 0;

}

for (i = 0; i < vertexCount - 1; i++) //delete old

{

delete[] adjacencyMatrix[i];

}

delete[] adjacencyMatrix;

adjacencyMatrix = arrInc; //update pointer

unordered\_map<string, int>::iterator ii = vertices.end();

--ii;

vertices.insert(make\_pair(vertex, (\*ii).second + 1)); //link new vertex after last

cout << "\nAdded vertex " << vertex << "." << endl;

outFile << "\nAdded vertex " << vertex << "." << endl;

}

else

{

cout << "\nAdding another vertex will exceed the maximum amount of 100 vertices." << endl;

outFile << "\nAdding another vertex will exceed the maximum amount of 100 vertices." << endl;

}

}

else

{

cout << "\nVertex " << vertex << " already exists." << endl;

outFile << "\nVertex " << vertex << " already exists." << endl;

}

}

void Graph::deleteVertex(string vertex)

{

int i, j, exists, num;

float \*\*arrDec;

unordered\_map<string, int> vertices2;

exists = lookupVertex(vertex);

if (exists != -1) //if vertex does exist

{

vertexCount = vertexCount - 1;

arrDec = new float\*[vertexCount];

for (i = 0; i < vertexCount; i++) //allocate space of new size

{

arrDec[i] = new float[vertexCount];

}

num = lookupVertex(vertex);

int num2 = vertexCount - num;

if (num != vertexCount + 1)

{

//swap rows/cols of old to outer most edge

for (i = 0; i < num2; i++) //swap columns

{

for (j = 0; j < vertexCount + 1; j++)

{

swap(adjacencyMatrix[j][num], adjacencyMatrix[j][num + 1]);

}

num++;

}

num = lookupVertex(vertex);

num2 = vertexCount - num;

for (i = 0; i < num2; i++) //swap rows

{

for (j = 0; j < vertexCount + 1; j++)

{

swap(adjacencyMatrix[num][j], adjacencyMatrix[num + 1][j]);

}

num++;

}

}

for (i = 0; i < vertexCount; i++) //copy from old to new

{

for (j = 0; j < vertexCount; j++)

arrDec[i][j] = adjacencyMatrix[i][j];

}

for (i = 0; i < vertexCount + 1; i++) //delete old

{

delete[] adjacencyMatrix[i];

}

delete[] adjacencyMatrix;

vertices.erase(vertex); //remove vertex from mapping

i = 0;

for (unordered\_map<string, int>::iterator ii = vertices.begin(); ii != vertices.end(); ++ii) //redo numbering

{

vertices2.insert(make\_pair((\*ii).first, i++));

}

vertices.swap(vertices2);

vertices2.clear();

adjacencyMatrix = arrDec; //update pointer

cout << "\nDeleted vertex " << vertex << "." << endl;

outFile << "\nDeleted vertex " << vertex << "." << endl;

}

else

{

cout << "\nVertex " << vertex << " does not exist." << endl;

outFile << "\nVertex " << vertex << " does not exist." << endl;

}

}

class BinaryHeap

{

private:

vector <int> heap;

int left(int parent);

int right(int parent);

int parent(int child);

void heapifyUp(int index);

void heapifyDown(int index);

public:

BinaryHeap(){}

void Insert(int element);

void DeleteMin();

int ExtractMin();

void DisplayHeap();

int Size();

};

int BinaryHeap::Size()

{

return heap.size();

}

void BinaryHeap::Insert(int element)

{

heap.push\_back(element);

heapifyUp(heap.size() - 1);

}

void BinaryHeap::DeleteMin()

{

if (heap.size() == 0)

{

cout << "Heap is Empty" << endl;

return;

}

heap[0] = heap.at(heap.size() - 1);

heap.pop\_back();

heapifyDown(0);

cout << "Element Deleted" << endl;

}

int BinaryHeap::ExtractMin()

{

if (heap.size() == 0)

{

return -1;

}

else

return heap.front();

}

void BinaryHeap::DisplayHeap()

{

vector <int>::iterator pos = heap.begin();

cout << "Heap --> ";

while (pos != heap.end())

{

cout << \*pos << " ";

pos++;

}

cout << endl;

}

int BinaryHeap::left(int parent)

{

unsigned int l = 2 \* parent + 1;

if (l < heap.size())

return l;

else

return -1;

}

int BinaryHeap::right(int parent)

{

unsigned int r = 2 \* parent + 2;

if (r < heap.size())

return r;

else

return -1;

}

int BinaryHeap::parent(int child)

{

int p = (child - 1) / 2;

if (child == 0)

return -1;

else

return p;

}

void BinaryHeap::heapifyUp(int in)

{

if (in >= 0 && parent(in) >= 0 && heap[parent(in)] > heap[in])

{

int temp = heap[in];

heap[in] = heap[parent(in)];

heap[parent(in)] = temp;

heapifyUp(parent(in));

}

}

void BinaryHeap::heapifyDown(int in)

{

int child = left(in);

int child1 = right(in);

if (child >= 0 && child1 >= 0 && heap[child] > heap[child1])

{

child = child1;

}

if (child > 0)

{

int temp = heap[in];

heap[in] = heap[child];

heap[child] = temp;

heapifyDown(child);

}

}

class Prim

{

friend class Graph;

private:

int num;

public:

void setNum(int size);

void setNum(int size, Graph &multiGraph);

float minKey(float key[], bool mstSet[]);

void primMST(float \*\*graph);

void printMST(int \*parent, float \*\*graph);

};

void Prim::setNum(int size, Graph &multiGraph)

{

num = multiGraph.getSize();

//primMST(multiGraph.adjacencyMatrix);

}

void Prim::setNum(int size)

{

num = size;

}

float Prim::minKey(float key[], bool mstSet[])

{

// Initialize min value

float min = FLT\_MAX, min\_index;

int v;

for (v = 0; v < num; v++)

if (mstSet[v] == false && key[v] < min)

min = key[v], min\_index = (float)v;

return min\_index;

}

void Prim::primMST(float \*\*graph)

{

int i, u, v, count;

int \*parent = new int[num]; // Array to store constructed MST

float \*key = new float[num]; // Key values used to pick minimum weight edge in cut

bool \*mstSet = new bool[num]; // To represent set of vertices not yet included in MST

cout << endl;

outFile << endl;

// Initialize all keys as INFINITE

for (i = 0; i < num; i++)

key[i] = FLT\_MAX, mstSet[i] = false;

// Always include first 1st vertex in MST.

key[0] = 0; // Make key 0 so that this vertex is picked as first vertex

parent[0] = -1; // First node is always root of MST

// The MST will have num vertices

for (count = 0; count < num - 1; count++)

{

// Pick the minimum key vertex from the set of vertices not yet included in MST

u = int(minKey(key, mstSet));

// Add the picked vertex to the MST Set

mstSet[u] = true;

// Update key value and parent index of the adjacent vertices of the picked vertex.

//Consider only those vertices which are not yet included in MST

for (v = 0; v < num; v++)

{

// graph[u][v] is non zero only for adjacent vertices of m

// mstSet[v] is false for vertices not yet included in MST

// Update the key only if graph[u][v] is smaller than key[v]

if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])

{

parent[v] = u, key[v] = graph[u][v];

cout << "Checking path to (" << u + 1 << "," << v + 1 << ")." << endl;

outFile << "Checking path to (" << u + 1 << "," << v + 1 << ")." << endl;

}

}

}

printMST(parent, graph);

}

void Prim::printMST(int \*parent, float \*\*graph)

{

int i;

cout << "\nEdge Weight" << endl;

outFile << "\nEdge Weight" << endl;

for (i = 1; i < num; i++)

{

cout << parent[i] + 1 << " - " << i + 1 << " " << graph[i][parent[i]] << endl;

outFile << parent[i] + 1 << " - " << i + 1 << " " << graph[i][parent[i]] << endl;

}

cout << endl;

outFile << endl;

}

void Graph::showDirectives()

{

cout << "\nThe directives are as follows:\n"

<< "print-graph Print the graph\n"

<< "print-mst Print the MST(s)\n"

<< "path u v Print the weight and path from u to v in the MST\n"

<< "insert-vertex u Insert vertex u in the graph\n"

<< "insert-edge u v w Insert edge (u,v) with weight w in the graph\n"

<< "decrease-weight u v w Decrease the weight of edge (u,v) by w units\n"

<< "delete-vertex u Delete vertex u from the graph\n"

<< "delete-edge u v Delete edge (u,v) from the graph\n"

<< "increase-weight u v w Increase the weight of edge (u,v) by w units\n"

<< "quit Quits the program\n"

<< "/? Show this list" << endl;

outFile << "\nThe directives are as follows:\n"

<< "print-graph Print the graph\n"

<< "print-mst Print the MST(s)\n"

<< "path u v Print the weight and path from u to v in the MST\n"

<< "insert-vertex u Insert vertex u in the graph\n"

<< "insert-edge u v w Insert edge (u,v) with weight w in the graph\n"

<< "decrease-weight u v w Decrease the weight of edge (u,v) by w units\n"

<< "delete-vertex u Delete vertex u from the graph\n"

<< "delete-edge u v Delete edge (u,v) from the graph\n"

<< "increase-weight u v w Increase the weight of edge (u,v) by w units\n"

<< "quit Quits the program\n"

<< "/? Show this list" << endl;

}

void Graph::directivesInput(Prim MST)

{

string directive, V1, V2;

int i, j;

float weight;

showDirectives();

do

{

cout << "\nPlease enter your directive: ";

outFile << "\nPlease enter your directive: ";

cin >> directive;

outFile << "Input: " << directive << endl;

if (directive == "print-graph")

printGraph(); //print graph

else if (directive == "print-mst")

MST.primMST(adjacencyMatrix); //print mst

else if (directive == "path")

{

cin >> V1 >> V2;

outFile << "Input: " << V1 << ' ' << V2 << endl;

//path

cout << "\nSorry, this function does not currently work." << endl;

outFile << "\nSorry, this function does not currently work." << endl;

}

else if (directive == "insert-vertex")

{

cin >> V1;

outFile << "Input: " << V1 << endl;

//insert vertex

addVertex(V1);

MST.setNum(vertexCount);

}

else if (directive == "insert-edge")

{

cin >> V1 >> V2 >> weight;

outFile << "Input: " << V1 << ' ' << V2 << ' ' << weight << endl;

if (isEdge(V1, V2))

{

cout << "\n(" << V1 << "," << V2 << ")" << " is already an edge." << endl;

outFile << "\n(" << V1 << "," << V2 << ")" << " is already an edge." << endl;

}

else

addEdge(V1, V2, weight);

}

else if (directive == "decrease-weight")

{

cin >> V1 >> V2 >> weight;

outFile << "Input: " << V1 << ' ' << V2 << ' ' << weight << endl;

if (isEdge(V1, V2))

{

i = lookupVertex(V1);

j = lookupVertex(V2);

adjacencyMatrix[i][j] = adjacencyMatrix[i][j] - weight;

adjacencyMatrix[j][i] = adjacencyMatrix[j][i] - weight;

cout << "\nEdge " << "(" << V1 << "," << V2 << ")" << " decreased by " << weight << "." << endl;

outFile << "\nEdge " << "(" << V1 << "," << V2 << ")" << " decreased by " << weight << "." << endl;

}

else

{

cout << "\n(" << V1 << "," << V2 << ")" << " is not a valid edge." << endl;

outFile << "\n(" << V1 << "," << V2 << ")" << " is not a valid edge." << endl;

}

}

else if (directive == "delete-vertex")

{

cin >> V1;

outFile << "Input: " << V1 << endl;

//delete vertex

deleteVertex(V1);

MST.setNum(vertexCount);

}

else if (directive == "delete-edge")

{

cin >> V1 >> V2;

outFile << "Input: " << V1 << ' ' << V2 << endl;

if (isEdge(V1, V2))

{

i = lookupVertex(V1);

j = lookupVertex(V2);

adjacencyMatrix[i][j] = 0;

adjacencyMatrix[j][i] = 0;

cout << "\nEdge " << "(" << V1 << "," << V2 << ")" << " has been removed." << endl;

outFile << "\nEdge " << "(" << V1 << "," << V2 << ")" << " has been removed." << endl;

}

}

else if (directive == "increase-weight")

{

cin >> V1 >> V2 >> weight;

outFile << "Input: " << V1 << ' ' << V2 << ' ' << weight << endl;

if (isEdge(V1, V2))

{

i = lookupVertex(V1);

j = lookupVertex(V2);

adjacencyMatrix[i][j] = adjacencyMatrix[i][j] + weight;

adjacencyMatrix[j][i] = adjacencyMatrix[j][i] + weight;

cout << "\nEdge " << "(" << V1 << "," << V2 << ")" << " increased by " << weight << "." << endl;

outFile << "\nEdge " << "(" << V1 << "," << V2 << ")" << " increased by " << weight << "." << endl;

}

else

{

cout << "\n(" << V1 << "," << V2 << ")" << " is not a valid edge." << endl;

outFile << "\n(" << V1 << "," << V2 << ")" << " is not a valid edge." << endl;

}

}

else if (directive == "/?")

showDirectives();

else

{

if (directive != "quit")

{

cout << "\nInvalid selection." << endl;

outFile << "\nInvalid selection." << endl;

}

}

} while (directive != "quit");

}

void Graph::directiveFile(ifstream &directiveFile, Prim MST)

{

string directive, V1, V2;

int i, j;

float weight;

while (!directiveFile.eof())

{

directiveFile >> directive;

outFile << "Input: " << directive;

if (directive == "print-graph")

printGraph(); //print graph

else if (directive == "print-mst")

MST.primMST(adjacencyMatrix); //print mst

else if (directive == "path")

{

directiveFile >> V1 >> V2;

outFile << "Input: " << V1 << ' ' << V2 << endl;

//path

cout << "\nSorry, this function does not currently work." << endl;

outFile << "\nSorry, this function does not currently work." << endl;

}

else if (directive == "insert-vertex")

{

directiveFile >> V1;

outFile << "Input: " << V1 << endl;

//insert vertex

addVertex(V1);

MST.setNum(vertexCount);

}

else if (directive == "insert-edge")

{

directiveFile >> V1 >> V2 >> weight;

outFile << "Input: " << V1 << ' ' << V2 << ' ' << weight << endl;

if (isEdge(V1, V2))

{

cout << "\n(" << V1 << "," << V2 << ")" << " is already an edge." << endl;

outFile << "\n(" << V1 << "," << V2 << ")" << " is already an edge." << endl;

}

else

addEdge(V1, V2, weight);

}

else if (directive == "decrease-weight")

{

directiveFile >> V1 >> V2 >> weight;

outFile << "Input: " << V1 << ' ' << V2 << ' ' << weight << endl;

if (isEdge(V1, V2))

{

i = lookupVertex(V1);

j = lookupVertex(V2);

adjacencyMatrix[i][j] = adjacencyMatrix[i][j] - weight;

adjacencyMatrix[j][i] = adjacencyMatrix[j][i] - weight;

cout << "\nEdge " << "(" << V1 << "," << V2 << ")" << " decreased by " << weight << "." << endl;

outFile << "\nEdge " << "(" << V1 << "," << V2 << ")" << " decreased by " << weight << "." << endl;

}

else

{

cout << "\n(" << V1 << "," << V2 << ")" << " is not a valid edge." << endl;

outFile << "\n(" << V1 << "," << V2 << ")" << " is not a valid edge." << endl;

}

}

else if (directive == "delete-vertex")

{

directiveFile >> V1;

outFile << "Input: " << V1 << endl;

//delete vertex

deleteVertex(V1);

MST.setNum(vertexCount);

}

else if (directive == "delete-edge")

{

directiveFile >> V1 >> V2;

outFile << "Input: " << V1 << ' ' << V2 << endl;

if (isEdge(V1, V2))

{

i = lookupVertex(V1);

j = lookupVertex(V2);

adjacencyMatrix[i][j] = 0;

adjacencyMatrix[j][i] = 0;

cout << "\nEdge " << "(" << V1 << "," << V2 << ")" << " has been removed." << endl;

outFile << "\nEdge " << "(" << V1 << "," << V2 << ")" << " has been removed." << endl;

}

}

else if (directive == "increase-weight")

{

directiveFile >> V1 >> V2 >> weight;

outFile << "Input: " << V1 << ' ' << V2 << ' ' << weight << endl;

if (isEdge(V1, V2))

{

i = lookupVertex(V1);

j = lookupVertex(V2);

adjacencyMatrix[i][j] = adjacencyMatrix[i][j] + weight;

adjacencyMatrix[j][i] = adjacencyMatrix[j][i] + weight;

cout << "\nEdge " << "(" << V1 << "," << V2 << ")" << " increased by " << weight << "." << endl;

outFile << "\nEdge " << "(" << V1 << "," << V2 << ")" << " increased by " << weight << "." << endl;

}

else

{

cout << "\n(" << V1 << "," << V2 << ")" << " is not a valid edge." << endl;

outFile << "\n(" << V1 << "," << V2 << ")" << " is not a valid edge." << endl;

}

}

else

{

if (directive != "quit")

{

cout << "\nInvalid selection." << endl;

outFile << "\nInvalid selection." << endl;

}

}

}

}

bool readVertex(ifstream &vertexFile, Graph &multiGraph)

{

int numVertices, count = 0, i = 0;

string fileLine;

vertexFile >> numVertices;

getline(vertexFile, fileLine); //get rest of blank line

if (numVertices > 100)

goto FAIL;

while (!vertexFile.eof())

{

vertexFile >> fileLine;

count++;

multiGraph.vertices.insert(make\_pair(fileLine, i++));

}

/\*

cout << "Before" << endl;

for (unordered\_map<string, int>::iterator ii = multiGraph.vertices.begin(); ii != multiGraph.vertices.end(); ++ii)

{

cout << (\*ii).first << " " << (\*ii).second << endl;

}

\*/

i = 0;

for (unordered\_map<string, int>::iterator ii = multiGraph.vertices.begin(); ii != multiGraph.vertices.end(); ++ii)

{

(\*ii).second = i++;

}

for (unordered\_map<string, int>::iterator ii = multiGraph.vertices.begin(); ii != multiGraph.vertices.end(); ++ii)

{

cout << "Vertex " << (\*ii).first << " added." << endl;

}

if (numVertices == count)

{

multiGraph.buildGraph(multiGraph.vertices.size());

return true;

}

else

{

FAIL:

return false;

}

}

bool readEdge(ifstream &edgeFile, Graph &multiGraph)

{

string V1, V2, fileLine;

float weight;

int numEdges, count = 0, i = 0;

edgeFile >> numEdges;

getline(edgeFile, fileLine); //get rest of blank line

while (!edgeFile.eof())

{

edgeFile >> V1 >> V2 >> weight;

multiGraph.addEdge(V1, V2, weight);

count++;

}

if (numEdges != count)

{

cout << "Error encountered. Number of edges do not match." << endl;

outFile << "Error encountered. Number of edges do not match." << endl;

return false;

}

else

return true;

}

int main()

{

string vertexName, edgeName, \*vertexTable = NULL, answer, dirName;

bool vertexCheck, edgeCheck;

ifstream vertexFile, edgeFile, directiveFile;

Graph multiGraph;

Prim MST;

cout << "The program has started." << endl;

outFile << "The program has started." << endl;

do //Determines if file is valid, if not, then repeats

{

cout << "\nPlease enter the name of the vertex file: ";

outFile << "\nPlease enter the name of the vertex file: ";

getline(cin, vertexName);

outFile << "Input: " << vertexName << endl;

vertexFile.open(vertexName);

if (vertexFile)

{

vertexCheck = readVertex(vertexFile, multiGraph);

break;

}

cout << "Invalid file. Please enter a valid file name.\n\n";

outFile << "Invalid file. Please enter a valid file name.\n\n";

} while (true);

if (vertexCheck == true)

{

do //Determines if file is valid, if not, then repeats

{

cout << "\nPlease enter the name of the edge file: ";

outFile << "\nPlease enter the name of the edge file: ";

getline(cin, edgeName);

outFile << "Input: " << edgeName << endl;

edgeFile.open(edgeName);

if (edgeFile)

{

cout << endl;

outFile << endl;

edgeCheck = readEdge(edgeFile, multiGraph);

break;

}

cout << "Invalid file. Please enter a valid file name.\n\n";

outFile << "Invalid file. Please enter a valid file name.\n\n";

} while (true);

}

else

{

cout << "ERROR! Incorrect number of vertices." << endl;

outFile << "ERROR! Incorrect number of vertices." << endl;

goto END;

}

if (edgeCheck == false)

{

cout << "ERROR! Incorrect number of edges." << endl;

outFile << "ERROR! Incorrect number of edges." << endl;

goto END;

}

MST.setNum(multiGraph.getSize(), multiGraph);

cout << "\nWill your directives come from input or file? (Please enter 'input' or 'file')" << endl;

outFile << "\nWill your directives come from input or file? (Please enter 'input' or 'file')" << endl;

getline(cin, answer);

outFile << "Input: " << answer << endl;

if (answer == "input")

multiGraph.directivesInput(MST);

else if (answer == "file")

{

do //Determines if file is valid, if not, then repeats

{

cout << "\nPlease enter the name of the directive file: ";

outFile << "\nPlease enter the name of the directive file: ";

getline(cin, dirName);

outFile << "Input: " << dirName << endl;

directiveFile.open(dirName);

if (directiveFile)

break;

cout << "Invalid file. Please enter a valid file name.\n";

outFile << "Invalid file. Please enter a valid file name.\n";

} while (true);

multiGraph.directiveFile(directiveFile, MST);

}

if (answer == "file")

{

cout << "\nWould you like to enter directives yourself? (Y/N): ";

outFile << "\nWould you like to enter directives yourself? (Y/N): ";

cin >> answer;

outFile << "Input: " << answer << endl;

cin.get();

if (answer == "Y" || answer == "y")

multiGraph.directivesInput(MST);

}

else if (answer == "input")

{

cout << "\nWould you like to enter a directive file as well? (Y/N): ";

outFile << "\nWould you like to enter a directive file as well? (Y/N): ";

cin >> answer;

outFile << "Input: " << answer << endl;

cin.get();

if (answer == "Y" || answer == "y")

{

do //Determines if file is valid, if not, then repeats

{

cout << "\nPlease enter the name of the directive file: ";

getline(cin, dirName);

outFile << "Input: " << dirName << endl;

directiveFile.open(dirName);

if (directiveFile)

break;

cout << "Invalid file. Please enter a valid file name.\n";

outFile << "Invalid file. Please enter a valid file name.\n";

} while (true);

multiGraph.directiveFile(directiveFile, MST);

}

}

vertexFile.close();

edgeFile.close();

END:

cout << "The program has ended." << endl;

outFile << "The program has ended." << endl;

\_getch();

return 0;

}