Source Code

# 

# Heart.cpp

**#include <iostream>**

**#include <chrono>**

**#include <thread>**

**#include "ohmcraft.cpp"**

**using namespace std;**

**void animateTitle2(const string& title) {**

**int length = title.length() + 10;**

**int animationDelay = 200;**

**for (int i = 0; i < 10; ++i) {**

**system("cls");**

**cout << "\033[1;31m"; // Set text color to blue and bold**

**for (int j = 0; j < length + i \* 2; ++j) {**

**cout << "-";**

**}**

**cout << endl;**

**for (int j = 0; j < i; ++j) {**

**cout << " ";**

**}**

**cout << "% " << title << " %" << endl;**

**for (int j = 0; j < length + i \* 2; ++j) {**

**cout << "-";**

**}**

**cout << "\033[0m"; // Reset text color to default**

**cout << endl;**

**this\_thread::sleep\_for(chrono::milliseconds(animationDelay));**

**}**

**}**

**void printWelcomeMessage() {**

**int animationDelay = 400;**

**cout << "\033[1;32m"; // Set text color to green and bold**

**cout << "----------------------------------------------------------------------------" << endl;**

**cout << "| Welcome (Please Read! Took So much to write....) |" << endl;**

**this\_thread::sleep\_for(chrono::milliseconds(animationDelay));**

**cout << "| ========================================================== |" << endl;**

**this\_thread::sleep\_for(chrono::milliseconds(animationDelay));**

**cout << "| This toolkit presents an API designed for handling Multi-Graphs, |" << endl;**

**cout << "| equipped with novel algorithms for node-based operations. |" << endl;**

**cout << "| Each node has the ability to establish connections with multiple |" << endl;**

**cout << "| other nodes, and it allows for the existence of multiple links between |" << endl;**

**cout << "| nodes. The underlying concept is highly adaptable and has been |" << endl;**

**cout << "| successfully applied to two distinct applications. Notably, this |" << endl;**

**cout << "| impressive toolkit is the sole creation of Chandru J. |" << endl;**

**cout << "| and can be reached via email at chandrukavin0503@gmail.com. |" << endl;**

**cout << "----------------------------------------------------------------------------" << endl;**

**cout << "\033[0m"; // Reset text color to default**

**cout << endl;**

**}**

**int whereToEnter(){**

**int animationDelay = 400;**

**cout << "\033[35m";**

**cout << "----------------------------------------------------------------------------" << endl;**

**cout << "| OHMCRAFT [Option: 1] |" << endl;**

**this\_thread::sleep\_for(chrono::milliseconds(animationDelay));**

**cout << "| ============================================================== |" << endl;**

**this\_thread::sleep\_for(chrono::milliseconds(animationDelay));**

**cout << "| This toolkit provides an API for a Graph Manipulation Model dedicated |" << endl;**

**cout << "| for Circuit Analysis With Peculiar New Algorithms for Nodal Fusion |" << endl;**

**cout << "| and my personal Circuit Solving Algorithm called Nodal Reduction |" << endl;**

**cout << "| using Prioritized Recursive Backtracking. |" << endl;**

**cout << "----------------------------------------------------------------------------" << endl;**

**cout << "\033[0m";**

**this\_thread::sleep\_for(chrono::milliseconds(animationDelay));**

**cout << "\033[1;34m";**

**cout << "----------------------------------------------------------------------------" << endl;**

**cout << "| Road Network Analysis [Option: 2] |" << endl;**

**this\_thread::sleep\_for(chrono::milliseconds(animationDelay));**

**cout << "| ============================================================== |" << endl;**

**this\_thread::sleep\_for(chrono::milliseconds(animationDelay));**

**cout << "| This toolkit provides an API for a Graph Manipulation Model dedicated |" << endl;**

**cout << "| to Road Network Analysis with Advanced Algorithms for Finding Shortest |" << endl;**

**cout << "| Distances and Paths between Multiple Locations. It stores data such as |" << endl;**

**cout << "| Road Distances between Nodes for comprehensive analysis. |" << endl;**

**cout << "----------------------------------------------------------------------------" << endl;**

**cout << "\033[0m";**

**this\_thread::sleep\_for(chrono::milliseconds(animationDelay));**

**this\_thread::sleep\_for(chrono::milliseconds(animationDelay));**

**cout << endl << ">>> Which one would you like to checkout? (1/2) > ";**

**int option;**

**cin >> option;**

**const int iterations = 10;**

**cout << endl;**

**for (int i = 0; i < iterations; ++i) {**

**cout << "\033[1;33mLoading................" << ((i % 2 == 0) ? "/" : "\\") << "\r";**

**cout.flush();**

**this\_thread::sleep\_for(chrono::milliseconds(300));**

**}**

**cout << "\033[0m" << endl;**

**return option;**

**}**

**int main(){**

**int option,opt;**

**do{**

**animateTitle2("Non-Directional Multi-Graph Manipulation Toolkit");**

**printWelcomeMessage();**

**option = whereToEnter();**

**if(option == 1){**

**main2();**

**cout << ">>> Want to Continue or Exit? (1/2) > ";**

**cin >> opt;**

**}**

**else if(option == 2){**

**cout << ">>> Want to Continue or Exit? (1/2) > ";**

**cin >> opt;**

**}**

**else{**

**cout << endl << ">>> Invalid Option (Enter Again) (1/2) > ";**

**cin>> option;**

**}**

**if(opt == 2){**

**break;**

**}**

**}**

**while(option != 0);**

**system("cls");**

**cout << "\033[1;31m"; // Set text color to red and bold**

**cout << "------------------------------------------------------------" << endl;**

**this\_thread::sleep\_for(chrono::milliseconds(200));**

**cout << "| Thank you for using the Non-Directional Multi-Graph |" << endl;**

**cout << "| Manipulation Toolkit! See you again! |" << endl;**

**this\_thread::sleep\_for(chrono::milliseconds(200));**

**cout << "------------------------------------------------------------" << endl;**

**cout << "\033[0m"; // Reset text color to default**

**return 0;**

**}**

# Ohmscraft.cpp

**#include <iostream>**

**#include <vector>**

**#include <list>**

**#include <iterator>**

**#include <limits>**

**#include <queue>**

**#include <fstream>**

**#include <sstream>**

**#include <string>**

**#include <list>**

**#include <unordered\_map>**

**#include <unordered\_set>**

**#include <chrono>**

**#include <thread>**

**#include <filesystem>**

**using namespace std;**

**class Link;**

**class Node;**

**class Link {**

**public:**

**int DestinationNodeID;**

**double weight;**

**Link() {}**

**Link(int destNodeID, double w) {**

**DestinationNodeID = destNodeID;**

**weight = w;**

**}**

**void setLinkValues(int destNodeID, double w) {**

**DestinationNodeID = destNodeID;**

**weight = w;**

**}**

**void setWeight(double w) {**

**weight = w;**

**}**

**int getDestinationNodeID() const {**

**return DestinationNodeID;**

**}**

**double getWeight() const {**

**return weight;**

**}**

**};**

**class Node {**

**public:**

**int state\_id;**

**string state\_name;**

**list<Link> linkList;**

**Node() {**

**state\_id = 0;**

**state\_name = "";**

**}**

**Node(int id, string sname) {**

**state\_id = id;**

**state\_name = sname;**

**}**

**int getStateID() const{**

**return state\_id;**

**}**

**string getStateName() const{**

**return state\_name;**

**}**

**void setID(int id) {**

**state\_id = id;**

**}**

**void setStateName(string sname) {**

**state\_name = sname;**

**}**

**list<Link> getLinkList() const{**

**return linkList;**

**}**

**void printLinkList() {**

**cout << "[";**

**for (auto it = linkList.begin(); it != linkList.end(); it++) {**

**cout << it->getDestinationNodeID() << "(" << it->getWeight() << ") --> ";**

**}**

**cout << "]";**

**cout << endl;**

**}**

**void updateNodeName(string sname) {**

**state\_name = sname;**

**cout << "Node Name Updated Successfully";**

**}**

**};**

**class Graph {**

**vector<Node> nodes;**

**public:**

**const vector<Node>& getNodes() const {**

**return nodes;**

**}**

**vector<Node>& getNodes() {**

**return nodes;**

**}**

**bool checkIfNodeExistByID(int nid) {**

**bool flag = false;**

**for (int i = 0; i < nodes.size(); i++) {**

**if (nodes.at(i).getStateID() == nid) {**

**return true;**

**}**

**}**

**return flag;**

**}**

**void addNode(Node newNode) {**

**bool check = checkIfNodeExistByID(newNode.getStateID());**

**if (check == true) {**

**cout << "Node with this ID already exists" << endl;**

**} else {**

**nodes.push\_back(newNode);**

**cout << "New Node Added Successfully" << endl;**

**}**

**}**

**Node getNodeByID(int nid) const{**

**Node temp;**

**for (int i = 0; i < nodes.size(); i++) {**

**temp = nodes.at(i);**

**if (temp.getStateID() == nid) {**

**return temp;**

**}**

**}**

**return temp;**

**}**

**bool checkIfLinkExistByID(int fromNode, int toNode) {**

**Node n = getNodeByID(fromNode);**

**list<Link> l;**

**l = n.getLinkList();**

**bool flag = false;**

**for (auto it = l.begin(); it != l.end(); it++) {**

**if (it->getDestinationNodeID() == toNode) {**

**flag = true;**

**return flag;**

**break;**

**}**

**}**

**return flag;**

**}**

**void updateNode(int oldNID, string nname) {**

**bool check = checkIfNodeExistByID(oldNID);**

**if (check == true) {**

**for (int i = 0; i < nodes.size(); i++) {**

**if (nodes.at(i).getStateID() == oldNID) {**

**nodes.at(i).setStateName(nname);**

**break;**

**}**

**}**

**cout << "Node(State) Updated Successfully " << endl;**

**}**

**}**

**double getLinkWeight(int fromNode, int toNode) {**

**for (int i = 0; i < nodes.size(); i++) {**

**if (nodes.at(i).getStateID() == fromNode) {**

**for (auto it = nodes.at(i).linkList.begin(); it != nodes.at(i).linkList.end(); it++) {**

**if (it->getDestinationNodeID() == toNode) {**

**return it->getWeight();**

**}**

**}**

**} else if (nodes.at(i).getStateID() == toNode) {**

**for (auto it = nodes.at(i).linkList.begin(); it != nodes.at(i).linkList.end(); it++) {**

**if (it->getDestinationNodeID() == fromNode) {**

**return it->getWeight();**

**}**

**}**

**}**

**}**

**return numeric\_limits<double>::infinity(); // Return infinity if link not found**

**}**

**void updateLinkWeight(int fromNode, int toNode, double newWeight) {**

**bool check = checkIfLinkExistByID(fromNode, toNode);**

**if (check == true) {**

**for (int i = 0; i < nodes.size(); i++) {**

**if (nodes.at(i).getStateID() == fromNode) {**

**for (auto it = nodes.at(i).linkList.begin(); it != nodes.at(i).linkList.end(); it++) {**

**if (it->getDestinationNodeID() == toNode) {**

**it->setWeight(newWeight);**

**break;**

**}**

**}**

**} else if (nodes.at(i).getStateID() == toNode) {**

**for (auto it = nodes.at(i).linkList.begin(); it != nodes.at(i).linkList.end(); it++) {**

**if (it->getDestinationNodeID() == fromNode) {**

**it->setWeight(newWeight);**

**break;**

**}**

**}**

**}**

**}**

**cout << "Link Weight Updated Successfully" << endl;**

**} else {**

**cout << "Link between " << getNodeByID(fromNode).getStateName() << "(" << fromNode << ") and "**

**<< getNodeByID(toNode).getStateName() << "(" << toNode << ") DOES NOT Exist" << endl;**

**}**

**}**

**void addLinkByID(int fromNode, int toNode, double weight) {**

**if (checkIfNodeExistByID(fromNode) && checkIfNodeExistByID(toNode)) {**

**bool linkExists = checkIfLinkExistByID(fromNode, toNode);**

**if (linkExists) {**

**// Link already exists, perform "parallel resistance" operation**

**double existingWeight = getLinkWeight(fromNode, toNode);**

**double newWeight = 1.0 / ((1.0 / existingWeight) + (1.0 / weight));**

**// Update the existing link's weight with the equivalent resistance**

**updateLinkWeight(fromNode, toNode, newWeight);**

**cout << "Link between " << getNodeByID(fromNode).getStateName() << "(" << fromNode << ") and "**

**<< getNodeByID(toNode).getStateName() << "(" << toNode << ") Updated Successfully" << endl;**

**} else {**

**// Link does not exist, add a new link**

**for (int i = 0; i < nodes.size(); i++) {**

**if (nodes.at(i).getStateID() == fromNode) {**

**Link l(toNode, weight);**

**nodes.at(i).linkList.push\_back(l);**

**} else if (nodes.at(i).getStateID() == toNode) {**

**Link l(fromNode, weight);**

**nodes.at(i).linkList.push\_back(l);**

**}**

**}**

**cout << "Link between " << fromNode << " and " << toNode << " added Successfully" << endl;**

**}**

**} else {**

**cout << "Invalid Node ID entered." << endl;**

**}**

**}**

**void updateLinkByID(int fromNode, int toNode, double newWeight) {**

**bool check = checkIfLinkExistByID(fromNode, toNode);**

**if (check == true) {**

**for (int i = 0; i < nodes.size(); i++) {**

**if (nodes.at(i).getStateID() == fromNode) {**

**for (auto it = nodes.at(i).linkList.begin(); it != nodes.at(i).linkList.end(); it++) {**

**if (it->getDestinationNodeID() == toNode) {**

**it->setWeight(newWeight);**

**break;**

**}**

**}**

**} else if (nodes.at(i).getStateID() == toNode) {**

**for (auto it = nodes.at(i).linkList.begin(); it != nodes.at(i).linkList.end(); it++) {**

**if (it->getDestinationNodeID() == fromNode) {**

**it->setWeight(newWeight);**

**break;**

**}**

**}**

**}**

**}**

**cout << "Link Weight Updated Successfully " << endl;**

**} else {**

**cout << "Link between " << getNodeByID(fromNode).getStateName() << "(" << fromNode << ") and "**

**<< getNodeByID(toNode).getStateName() << "(" << toNode << ") DOES NOT Exist" << endl;**

**}**

**}**

**void deleteLinkByID(int fromNode, int toNode) {**

**bool check = checkIfLinkExistByID(fromNode, toNode);**

**if (check == true) {**

**for (int i = 0; i < nodes.size(); i++) {**

**if (nodes.at(i).getStateID() == fromNode) {**

**for (auto it = nodes.at(i).linkList.begin(); it != nodes.at(i).linkList.end(); it++) {**

**if (it->getDestinationNodeID() == toNode) {**

**nodes.at(i).linkList.erase(it);**

**break;**

**}**

**}**

**}**

**if (nodes.at(i).getStateID() == toNode) {**

**for (auto it = nodes.at(i).linkList.begin(); it != nodes.at(i).linkList.end(); it++) {**

**if (it->getDestinationNodeID() == fromNode) {**

**nodes.at(i).linkList.erase(it);**

**break;**

**}**

**}**

**}**

**}**

**cout << "Link Between " << fromNode << " and " << toNode << " Deleted Successfully." << endl;**

**}**

**}**

**void deleteNodeByID(int nid) {**

**int nIndex = 0;**

**for (int i = 0; i < nodes.size(); i++) {**

**if (nodes.at(i).getStateID() == nid) {**

**nIndex = i;**

**}**

**}**

**for (int i = 0; i < nodes.size(); i++) {**

**for (auto it = nodes.at(i).linkList.begin(); it != nodes.at(i).linkList.end(); it++) {**

**if (it->getDestinationNodeID() == nid) {**

**nodes.at(i).linkList.erase(it);**

**break;**

**}**

**}**

**}**

**nodes.erase(nodes.begin() + nIndex);**

**cout << "Node Deleted Successfully" << endl;**

**}**

**void getAllNeighborsByID(int nid) {**

**cout << getNodeByID(nid).getStateName() << " (" << getNodeByID(nid).getStateID() << ") --> ";**

**for (int i = 0; i < nodes.size(); i++) {**

**if (nodes.at(i).getStateID() == nid) {**

**cout << "[";**

**for (auto it = nodes.at(i).linkList.begin(); it != nodes.at(i).linkList.end(); it++) {**

**cout << it->getDestinationNodeID() << "(" << it->getWeight() << ") --> ";**

**}**

**cout << "]";**

**}**

**}**

**}**

**void printGraph() {**

**for (int i = 0; i < nodes.size(); i++) {**

**Node temp;**

**temp = nodes.at(i);**

**cout << temp.getStateName() << " (" << temp.getStateID() << ") --> ";**

**temp.printLinkList();**

**}**

**}**

**void fuseNodes(int nodeID1, int nodeID2) {**

**// Check if both nodes exist in the graph**

**if (!checkIfNodeExistByID(nodeID1) || !checkIfNodeExistByID(nodeID2)) {**

**cout << "One or both of the nodes do not exist in the graph." << endl;**

**return;**

**}**

**// Find the nodes to be fused**

**Node\* node1 = nullptr;**

**Node\* node2 = nullptr;**

**for (int i = 0; i < nodes.size(); i++) {**

**if (nodes[i].getStateID() == nodeID1) {**

**node1 = &nodes[i];**

**} else if (nodes[i].getStateID() == nodeID2) {**

**node2 = &nodes[i];**

**}**

**}**

**// Check if the link between the two nodes holds a resistance value of -1**

**bool linkExists = false;**

**double resistanceValue = 0.0;**

**for (const Link& link : node1->getLinkList()) {**

**if (link.getDestinationNodeID() == nodeID2) {**

**linkExists = true;**

**resistanceValue = link.getWeight();**

**break;**

**}**

**}**

**if (!linkExists || resistanceValue != -1.0) {**

**cout << "The link between the nodes does not have a resistance value of -1." << endl;**

**return;**

**}**

**// Check if at least one of the nodes has at most two neighboring nodes**

**if ((node1->getLinkList().size() > 2 && node2->getLinkList().size() > 2)) {**

**cout << "At least one of the nodes should have at most two neighboring nodes." << endl;**

**return;**

**}**

**// Get the maximum node ID present in the graph**

**int maxNodeID = 0;**

**for (const Node& node : nodes) {**

**if (node.getStateID() > maxNodeID) {**

**maxNodeID = node.getStateID();**

**}**

**}**

**// Fuse the nodes and update the graph**

**int newNodeID = maxNodeID + 1; // Assign a new ID that is one greater than the maximum node ID**

**string fusedNodeName = node1->getStateName() + "\_" + node2->getStateName();**

**Node fusedNode(newNodeID, fusedNodeName);**

**addNode(fusedNode);**

**// Add links from the neighboring nodes of node1 to the fused node**

**for (const Link& link : node1->getLinkList()) {**

**int destNodeID = link.getDestinationNodeID();**

**if (destNodeID != nodeID2) {**

**fusedNode.linkList.push\_back(Link(destNodeID, link.getWeight()));**

**// Update the neighboring nodes to link to the fused node**

**addLinkByID(destNodeID, newNodeID, link.getWeight());**

**}**

**}**

**// Add links from the neighboring nodes of node2 to the fused node**

**for (const Link& link : node2->getLinkList()) {**

**int destNodeID = link.getDestinationNodeID();**

**if (destNodeID != nodeID1) {**

**fusedNode.linkList.push\_back(Link(destNodeID, link.getWeight()));**

**// Update the neighboring nodes to link to the fused node**

**addLinkByID(destNodeID, newNodeID, link.getWeight());**

**}**

**}**

**// Add the fused node to the graph and delete the original nodes**

**deleteNodeByID(nodeID1);**

**deleteNodeByID(nodeID2);**

**cout << "Nodes " << nodeID1 << " and " << nodeID2 << " fused into Node " << newNodeID << "." << endl;**

**}**

**void fuseChainOfLinks(int startNodeID, int endNodeID);**

**void clearGraph() {**

**nodes.clear();**

**cout << "Graph cleared successfully." << endl;**

**}**

**bool hasNoLinkWeightMinusOne() const {**

**for (const Node& node : nodes) {**

**const list<Link>& links = node.getLinkList();**

**for (const Link& link : links) {**

**if (link.getWeight() == -1) {**

**return false;**

**}**

**}**

**}**

**return true;**

**}**

**pair<int, int> findEndNodesForChainFusion() const {**

**int endNodeCount = 0;**

**int endNodeID1 = -1;**

**int endNodeID2 = -1;**

**for (const Node& node : nodes) {**

**const list<Link>& links = node.getLinkList();**

**if (links.size() == 1) {**

**endNodeCount++;**

**if (endNodeCount == 1) {**

**endNodeID1 = node.getStateID();**

**} else if (endNodeCount == 2) {**

**endNodeID2 = node.getStateID();**

**} else {**

**// There are more than two end nodes, return an invalid pair**

**return make\_pair(-1, -1);**

**}**

**}**

**}**

**// If there are exactly two end nodes, return them as a pair**

**if (endNodeCount == 2) {**

**return make\_pair(endNodeID1, endNodeID2);**

**} else {**

**// There are not exactly two end nodes, return an invalid pair**

**return make\_pair(-1, -1);**

**}**

**}**

**void autoFuseChainOfLinks() {**

**// Find the end nodes based on the criteria**

**pair<int, int> endNodes = findEndNodesForChainFusion();**

**int nodeID1 = endNodes.first;**

**int nodeID2 = endNodes.second;**

**// Check if the end nodes are valid for the chain fusion operation**

**if (nodeID1 == -1 || nodeID2 == -1) {**

**cout << "Invalid end nodes for chain fusion operation. Ensure there are exactly two end nodes with only one link each." << endl;**

**return;**

**}**

**// Call the existing fuseChainOfLinks function with the end nodes**

**fuseChainOfLinks(nodeID1, nodeID2);**

**}**

**};**

**void Graph::fuseChainOfLinks(int startNodeID, int endNodeID) {**

**// Check if the start and end nodes exist in the graph**

**bool startExists = checkIfNodeExistByID(startNodeID);**

**bool endExists = checkIfNodeExistByID(endNodeID);**

**if (!startExists || !endExists) {**

**cout << "One or both of the nodes do not exist in the graph." << endl;**

**return;**

**}**

**// Find the start and end nodes to fuse**

**Node\* startNode = nullptr;**

**Node\* endNode = nullptr;**

**for (int i = 0; i < nodes.size(); i++) {**

**if (nodes[i].getStateID() == startNodeID) {**

**startNode = &nodes[i];**

**} else if (nodes[i].getStateID() == endNodeID) {**

**endNode = &nodes[i];**

**}**

**}**

**// Check if there is a chain of links between the start and end nodes**

**if (startNode == nullptr || endNode == nullptr || startNode == endNode) {**

**cout << "No chain of links between the given nodes." << endl;**

**return;**

**}**

**// Fuse the chain of links into a single link with equivalent resistance**

**double equivalentResistance = 0.0;**

**for (const Link& link : startNode->getLinkList()) {**

**int nextNodeID = link.getDestinationNodeID();**

**if (nextNodeID == endNodeID) {**

**// Reached the end node, break the loop**

**break;**

**}**

**// Add the resistance of each link in the chain**

**equivalentResistance += link.getWeight();**

**// Remove the intermediate nodes and their links from the graph**

**for (int i = 0; i < nodes.size(); i++) {**

**if (nodes[i].getStateID() == nextNodeID) {**

**nodes.erase(nodes.begin() + i);**

**break;**

**}**

**}**

**}**

**for (const Link& link : endNode->getLinkList()) {**

**int nextNodeID = link.getDestinationNodeID();**

**if (nextNodeID == startNodeID) {**

**// Reached the end node, break the loop**

**break;**

**}**

**// Add the resistance of each link in the chain**

**equivalentResistance += link.getWeight();**

**// Remove the intermediate nodes and their links from the graph**

**for (int i = 0; i < nodes.size(); i++) {**

**if (nodes[i].getStateID() == nextNodeID) {**

**nodes.erase(nodes.begin() + i);**

**break;**

**}**

**}**

**}**

**// Update the link between the start and end nodes with the equivalent resistance**

**startNode->linkList.clear();**

**startNode->linkList.push\_back(Link(endNodeID, equivalentResistance));**

**// Update the reverse link from endNode to startNode (for two-nodal representation)**

**endNode->linkList.clear();**

**endNode->linkList.push\_back(Link(startNodeID, equivalentResistance));**

**cout << "Chain of links between nodes " << startNodeID << " and " << endNodeID**

**<< " fused into a two-nodal system with equivalent resistance: " << equivalentResistance << endl;**

**}**

**void setupGraphFromCSV(Graph& graph, const string& filename) {**

**graph.clearGraph();**

**ifstream file(filename);**

**if (!file.is\_open()) {**

**cerr << "Error: Unable to open file " << filename << endl;**

**return;**

**}**

**string line;**

**while (getline(file, line)) {**

**istringstream iss(line);**

**string sourceID, sourceName, destID, destName, weight;**

**if (getline(iss, sourceID, ',') &&**

**getline(iss, sourceName, ',') &&**

**getline(iss, destID, ',') &&**

**getline(iss, destName, ',') &&**

**getline(iss, weight, ',')) {**

**int sourceIDInt = stoi(sourceID);**

**int destIDInt = stoi(destID);**

**double weightInt = stod(weight);**

**bool sourceExists = graph.checkIfNodeExistByID(sourceIDInt);**

**if (!sourceExists) {**

**Node v(sourceIDInt, sourceName);**

**graph.addNode(v);**

**}**

**bool destExists = graph.checkIfNodeExistByID(destIDInt);**

**if (!destExists) {**

**Node v(destIDInt, destName);**

**graph.addNode(v);**

**}**

**graph.addLinkByID(sourceIDInt, destIDInt, weightInt);**

**}**

**}**

**file.close();**

**}**

**template <typename T>**

**void reverseVector(vector<T>& vec) {**

**size\_t start = 0;**

**size\_t end = vec.size() - 1;**

**while (start < end) {**

**swap(vec[start], vec[end]);**

**start++;**

**end--;**

**}**

**}**

**void animateTitle(const string& title) {**

**int length = title.length() + 10;**

**int animationDelay = 100;**

**for (int i = 0; i < 10; ++i) {**

**system("cls");**

**cout << "\033[1;34m"; // Set text color to blue and bold**

**for (int j = 0; j < length + i \* 2; ++j) {**

**cout << "-";**

**}**

**cout << endl;**

**for (int j = 0; j < i; ++j) {**

**cout << " ";**

**}**

**cout << "% " << title << " %" << endl;**

**for (int j = 0; j < length + i \* 2; ++j) {**

**cout << "-";**

**}**

**cout << "\033[0m"; // Reset text color to default**

**cout << endl;**

**this\_thread::sleep\_for(chrono::milliseconds(animationDelay));**

**}**

**}**

**void clearConsole() {**

**// For Windows**

**system("cls");**

**// For Linux and macOS**

**//system("clear");**

**}**

**void printMenu() {**

**cout << "\033[1;36m"; // Set text color to cyan and bold**

**this\_thread::sleep\_for(chrono::milliseconds(300));**

**cout << "------------------------------------" << endl;**

**cout << "| Node Operations |" << endl;**

**cout << "------------------------------------" << endl;**

**cout << "| [1] Add Node |" << endl;**

**cout << "| [2] Update Node |" << endl;**

**cout << "| [3] Delete Node |" << endl;**

**cout << "------------------------------------" << endl;**

**this\_thread::sleep\_for(chrono::milliseconds(300));**

**cout << "| Link Operations |" << endl;**

**cout << "------------------------------------" << endl;**

**cout << "| [4] Add Link |" << endl;**

**cout << "| [5] Update Link |" << endl;**

**cout << "| [6] Delete Link |" << endl;**

**cout << "------------------------------------" << endl;**

**this\_thread::sleep\_for(chrono::milliseconds(300));**

**cout << "| Neighbor Check |" << endl;**

**cout << "------------------------------------" << endl;**

**cout << "| [7] Check if Neighbors |" << endl;**

**cout << "| [8] Print Neighbors |" << endl;**

**cout << "------------------------------------" << endl;**

**this\_thread::sleep\_for(chrono::milliseconds(300));**

**cout << "| Graph Operations |" << endl;**

**cout << "------------------------------------" << endl;**

**cout << "| [9] Print Graph |" << endl;**

**cout << "| [10] Setup Graph from CSV |" << endl;**

**cout << "------------------------------------" << endl;**

**this\_thread::sleep\_for(chrono::milliseconds(300));**

**cout << "| Fusion Operations |" << endl;**

**cout << "------------------------------------" << endl;**

**cout << "| [11] Fuse Two Nodes |" << endl;**

**cout << "| [12] Auto Fuse Neighboring |" << endl;**

**cout << "| Nodes |" << endl;**

**cout << "| [13] Fuse Chain of Links |" << endl;**

**cout << "| between Nodes |" << endl;**

**cout << "| [14] Auto Fuse Chain of Links |" << endl;**

**cout << "|\033[1;33m [15] Execute Chandru's Nodal \033[0m\033[1;36m |" << endl;**

**cout << "| \033[1;33m Reduction Algorithm \033[0m \033[1;36m |" << endl;**

**cout << "------------------------------------" << endl;**

**cout << "| File Management |" << endl;**

**cout << "------------------------------------" << endl;**

**cout << "| [16] File Menu |" << endl;**

**cout << "| [17] Save Current file |" << endl;**

**cout << "| [18] Reload Current file |" << endl;**

**cout << "------------------------------------" << endl;**

**cout << "| [0] Exit Program |" << endl;**

**cout << "------------------------------------" << endl;**

**cout << "\033[0m";**

**}**

**void saveGraphToFile(const Graph& graph, const string& filename) {**

**ofstream file(filename);**

**if (!file.is\_open()) {**

**cerr << "Error: Unable to open file " << filename << endl;**

**return;**

**}**

**const vector<Node>& nodes = graph.getNodes();**

**for (const Node& node : nodes) {**

**int sourceNodeID = node.getStateID();**

**const string& sourceNodeName = node.getStateName();**

**const list<Link>& links = node.getLinkList();**

**for (const Link& link : links) {**

**int destNodeID = link.getDestinationNodeID();**

**double weight = link.getWeight();**

**file << sourceNodeID << "," << sourceNodeName << "," << destNodeID << "," << graph.getNodeByID(destNodeID).getStateName() << "," << weight << endl;**

**}**

**}**

**file.close();**

**cout << "Graph data has been saved to " << filename << endl;**

**}**

**namespace fs = filesystem;**

**bool hasCsvExtension(const string& filename) {**

**return filename.size() > 4 && filename.substr(filename.size() - 4) == ".csv";**

**}**

**void showAvailableCsvFiles(const string& directory) {**

**cout << "Available .csv files in " << directory << ":\n";**

**int count = 0;**

**for (const auto& entry : fs::directory\_iterator(directory)) {**

**if (entry.is\_regular\_file() && hasCsvExtension(entry.path().filename().string())) {**

**cout << entry.path().filename().string() << '\n';**

**count++;**

**}**

**}**

**if (count == 0) {**

**cout << "No .csv files found in the directory.\n";**

**}**

**}**

**bool fileExists(const string& filename) {**

**ifstream file(filename);**

**return file.good();**

**}**

**void createFile(const string& filename) {**

**ofstream file(filename);**

**if (file) {**

**cout << "File created successfully: " << filename << '\n';**

**} else {**

**cerr << "Error creating file: " << filename << '\n';**

**}**

**}**

**void deleteFile(const string& filename) {**

**if (remove(filename.c\_str()) == 0) {**

**cout << "File deleted successfully: " << filename << '\n';**

**} else {**

**cerr << "Error deleting file: " << filename << '\n';**

**}**

**}**

**void exportDataToCsv(const Graph& graph, const string& filename) {**

**ofstream file(filename);**

**if (file) {**

**const vector<Node>& nodes = graph.getNodes();**

**unordered\_set<string> visitedLinks; // To keep track of visited links to avoid duplicates**

**for (const Node& node : nodes) {**

**int sourceID = node.getStateID();**

**const string& sourceName = node.getStateName();**

**const list<Link>& links = node.getLinkList();**

**for (const Link& link : links) {**

**int destID = link.getDestinationNodeID();**

**double weight = link.getWeight();**

**// Check if the reverse link has already been visited**

**string linkKey1 = to\_string(sourceID) + "\_" + to\_string(destID);**

**string linkKey2 = to\_string(destID) + "\_" + to\_string(sourceID);**

**if (visitedLinks.find(linkKey1) == visitedLinks.end() && visitedLinks.find(linkKey2) == visitedLinks.end()) {**

**file << sourceID << "," << sourceName << "," << destID << "," << graph.getNodeByID(destID).getStateName() << "," << weight << '\n';**

**visitedLinks.insert(linkKey1);**

**}**

**}**

**}**

**cout << "Data exported to CSV successfully: " << filename << '\n';**

**} else {**

**cerr << "Error exporting data to CSV: " << filename << '\n';**

**}**

**}**

**int main2() {**

**Graph g;**

**string sname;**

**string filename;**

**int numNodes;**

**vector<int> nodeIDs;**

**int id1, id2, w;**

**int sourceID, destinationID;**

**vector<int> shortestPath;**

**string directory = "./";**

**int choice;**

**int closestNode;**

**int shortestDist;**

**int option;**

**bool check;**

**char h;**

**system("cls");**

**animateTitle(" OHMCRAFT - Multi-Graph Model for Circuit Analysis");**

**do {**

**printMenu();**

**cout << endl << ">>> Enter your option > " ;**

**cin >> option;**

**Node n1;**

**switch (option) {**

**case 0:**

**break;**

**case 1:**

**cout << "Add Node Operation -" << endl;**

**cout << "Enter State ID: ";**

**cin >> id1;**

**cout << "Enter State Name: ";**

**cin >> sname;**

**cout << endl << "\033[38;5;208m---------< Background Tasks -------------- \033[1m" << endl << endl;**

**n1.setID(id1);**

**n1.setStateName(sname);**

**g.addNode(n1);**

**cout << endl << "\033[38;5;208m---------- Background Tasks />------------ \033[0m" << endl << endl;**

**break;**

**case 2:**

**cout << "Update Node Operation -" << endl;**

**cout << "Enter State ID of Node(State) to update: ";**

**cin >> id1;**

**cout << "Enter State Name: ";**

**cin >> sname;**

**cout << endl << "\033[38;5;208m---------< Background Tasks -------------- \033[1m" << endl << endl;**

**g.updateNode(id1, sname);**

**cout << endl << "\033[38;5;208m---------- Background Tasks />------------ \033[0m" << endl << endl;**

**break;**

**case 3:**

**cout << "Delete Node Operation -" << endl;**

**cout << "Enter ID of Node(State) to Delete: ";**

**cin >> id1;**

**cout << endl << "\033[38;5;208m---------< Background Tasks -------------- \033[1m" << endl << endl;**

**g.deleteNodeByID(id1);**

**cout << endl << "\033[38;5;208m---------- Background Tasks />------------ \033[0m" << endl << endl;**

**break;**

**case 4:**

**cout << "Add Link Operation -" << endl;**

**cout << "Enter ID of Source Node(State): ";**

**cin >> id1;**

**cout << "Enter ID of Destination Node(State): ";**

**cin >> id2;**

**cout << "Enter Weight of Link: ";**

**cin >> w;**

**cout << endl << "\033[38;5;208m---------< Background Tasks -------------- \033[1m" << endl << endl;**

**g.addLinkByID(id1, id2, w);**

**cout << endl << "\033[38;5;208m---------- Background Tasks />------------ \033[0m" << endl << endl;**

**break;**

**case 5:**

**cout << "Update Link Operation -" << endl;**

**cout << "Enter ID of Source Node(State): ";**

**cin >> id1;**

**cout << "Enter ID of Destination Node(State): ";**

**cin >> id2;**

**cout << "Enter UPDATED Weight of Link: ";**

**cin >> w;**

**cout << endl << "\033[38;5;208m---------< Background Tasks -------------- \033[1m" << endl << endl;**

**g.updateLinkByID(id1, id2, w);**

**cout << endl << "\033[38;5;208m---------- Background Tasks />------------ \033[0m" << endl << endl;**

**break;**

**case 6:**

**cout << "Delete Link Operation -" << endl;**

**cout << "Enter ID of Source Node(State): ";**

**cin >> id1;**

**cout << "Enter ID of Destination Node(State): ";**

**cin >> id2;**

**cout << endl << "\033[38;5;208m---------< Background Tasks -------------- \033[1m" << endl << endl;**

**g.deleteLinkByID(id1, id2);**

**cout << endl << "\033[38;5;208m---------- Background Tasks />------------ \033[0m" << endl << endl;**

**break;**

**case 7:**

**cout << "Check if 2 Nodes are Neighbors -" << endl;**

**cout << "Enter ID of Source Node(State): ";**

**cin >> id1;**

**cout << "Enter ID of Destination Node(State): ";**

**cin >> id2;**

**cout << endl << "\033[38;5;208m---------< Background Tasks -------------- \033[1m" << endl << endl;**

**check = g.checkIfLinkExistByID(id1, id2);**

**if (check == true) {**

**cout << "Nodes are Neighbors (Link exists)" << endl;**

**} else {**

**cout << "Nodes are NOT Neighbors (Link does NOT exist)" << endl;**

**}**

**cout << endl << "\033[38;5;208m---------- Background Tasks />------------ \033[0m" << endl << endl;**

**break;**

**case 8:**

**cout << "Print All Neighbors of a Node -" << endl;**

**cout << "Enter ID of Node(State) to fetch all Neighbors: ";**

**cin >> id1;**

**cout << endl << "\033[38;5;208m---------< Background Tasks -------------- \033[1m" << endl << endl;**

**g.getAllNeighborsByID(id1);**

**cout << endl << "\033[38;5;208m---------- Background Tasks />------------ \033[0m" << endl << endl;**

**break;**

**case 9:**

**cout << endl << "\033[38;5;208m---------< printing Graph -------------- \033[1m" << endl << endl;**

**g.printGraph();**

**cout << endl << "\033[38;5;208m---------- Printing the Graph />------------ \033[0m" << endl << endl;**

**break;**

**case 10:**

**cout << "Setup Graph from CSV -" << endl;**

**filename = "E:\\OhmCraft\\circuitDatabase.csv";**

**do {**

**cout << "\nFile Handling Menu\n";**

**cout << "[1] Open a file\n";**

**cout << "[2] Show available files\n";**

**cout << "[0] Exit\n";**

**cout << endl << "Enter your choice (1-6): ";**

**cin >> choice;**

**switch (choice) {**

**case 1:**

**cout << endl << "Enter the filename to open: ";**

**cin >> filename;**

**if (fileExists(filename)) {**

**ifstream file(filename);**

**string content((istreambuf\_iterator<char>(file)), istreambuf\_iterator<char>());**

**cout << endl << "\033[38;5;208m---------< Setting Up Graph ---------- \033[1m" << endl << endl;**

**cout << "File content:\n" << content << '\n';**

**setupGraphFromCSV(g, filename);**

**cout << endl << "\033[38;5;208m---------- Setting Up Graph />------------ \033[0m" << endl << endl;**

**} else {**

**cerr << "Error: File not found.\n";**

**}**

**break;**

**case 2:**

**cout << endl << "\033[38;5;208m---------< File Menu ---------- \033[1m" << endl << endl;**

**showAvailableCsvFiles(directory);**

**cout << endl << "\033[38;5;208m---------- File Menu />------------ \033[0m" << endl << endl;**

**break;**

**case 0:**

**cout << "Exiting the program.\n";**

**break;**

**default:**

**cerr << "Invalid choice. Please enter a number from 1 to 5.\n";**

**break;**

**}**

**} while (choice != 0);**

**break;**

**case 11: {**

**cout << "Fuse Nodes Operation -" << endl;**

**int nodeID1, nodeID2;**

**cout << "Enter the ID of the first node to fuse: ";**

**cin >> nodeID1;**

**cout << "Enter the ID of the second node to fuse: ";**

**cin >> nodeID2;**

**cout << endl << "\033[38;5;208m---------< Fusing Nodes -------------- \033[1m" << endl << endl;**

**g.fuseNodes(nodeID1, nodeID2);**

**cout << endl << "\033[38;5;208m---------- Fusing Nodes />------------ \033[0m" << endl << endl;**

**break;**

**}**

**case 12: {**

**cout << "Fuse Neighboring Nodes Operation -" << endl;**

**do{**

**// Iterate through all nodes in the graph**

**for (const Node& node : g.getNodes()) {**

**int nodeID = node.getStateID();**

**const list<Link>& linkList = node.getLinkList();**

**// Iterate through the neighboring nodes (links) of the current node**

**for (const Link& link : linkList) {**

**int neighborNodeID = link.getDestinationNodeID();**

**// Check if the neighbor node exists in the graph**

**bool neighborExists = g.checkIfNodeExistByID(neighborNodeID);**

**if (neighborExists) {**

**// Execute the fuseNodes function for the current node and its neighboring node**

**g.fuseNodes(nodeID, neighborNodeID);**

**cout << endl << "\033[38;5;208m---------< Fusing Sub-Steps -------------- \033[1m" << endl << endl;**

**g.printGraph();**

**cout << endl << "\033[38;5;208m---------- Fussing Sub-Steps />------------ \033[0m" << endl << endl;**

**}**

**}**

**}**

**}**

**while(!g.hasNoLinkWeightMinusOne());**

**cout << "All neighboring nodes fused successfully." << endl;**

**break;**

**}**

**case 13: {**

**cout << "Fuse Chain of Links between Nodes Operation -" << endl;**

**int nodeID1, nodeID2;**

**cout << "Enter the ID of the start node: ";**

**cin >> nodeID1;**

**cout << "Enter the ID of the end node: ";**

**cin >> nodeID2;**

**cout << endl << "\033[38;5;208m---------< Fusing Chain of Nodes -------------- \033[1m" << endl << endl;**

**g.fuseChainOfLinks(nodeID1, nodeID2);**

**cout << endl << "\033[38;5;208m---------- Fusing Chain of Nodes />------------ \033[0m" << endl << endl;**

**break;**

**}**

**case 14: {**

**cout << endl << "\033[38;5;208m---------< Auto Fusing Chain of Nodes -------------- \033[1m" << endl << endl;**

**g.autoFuseChainOfLinks();**

**cout << endl << "\033[38;5;208m---------- Auto Fusing Chain of Nodes />------------ \033[0m" << endl << endl;**

**}**

**case 15: {**

**cout << "Executing Chandru's Nodal Reduction Algorithm" << endl;**

**do{**

**// Iterate through all nodes in the graph**

**for (const Node& node : g.getNodes()) {**

**int nodeID = node.getStateID();**

**const list<Link>& linkList = node.getLinkList();**

**// Iterate through the neighboring nodes (links) of the current node**

**for (const Link& link : linkList) {**

**int neighborNodeID = link.getDestinationNodeID();**

**// Check if the neighbor node exists in the graph**

**bool neighborExists = g.checkIfNodeExistByID(neighborNodeID);**

**if (neighborExists) {**

**// Execute the fuseNodes function for the current node and its neighboring node**

**g.fuseNodes(nodeID, neighborNodeID);**

**cout << endl << "\033[38;5;208m---------< Fusing Sub-Steps -------------- \033[1m" << endl << endl;**

**g.printGraph();**

**cout << endl << "\033[38;5;208m---------- Fusing Sub-Steps />------------ \033[0m" << endl << endl;**

**}**

**}**

**}**

**}**

**while(!g.hasNoLinkWeightMinusOne());**

**cout << "All neighboring nodes fused successfully." << endl;**

**cout << endl << "\033[38;5;208m---------< Final Result -------------- \033[1m" << endl << endl;**

**g.autoFuseChainOfLinks();**

**cout << endl << "\033[38;5;208m---------- Final Result />------------ \033[0m" << endl << endl;**

**break;**

**}**

**case 16:**

**cout << "Update File Operation -" << endl;**

**do {**

**cout << "\nFile Handling Menu\n";**

**cout << "1. Open a file\n";**

**cout << "2. Create a new file\n";**

**cout << "3. Delete a file\n";**

**cout << "4. Show available files\n";**

**cout << "5. Export data to CSV\n"; // New option added**

**cout << "6. Exit\n";**

**cout << "Enter your choice (1-6): ";**

**cin >> choice;**

**switch (choice) {**

**case 1:**

**cout << endl << "Enter the filename to open: ";**

**cin >> filename;**

**if (fileExists(filename)) {**

**ifstream file(filename);**

**string content((istreambuf\_iterator<char>(file)), istreambuf\_iterator<char>());**

**cout << endl << "\033[38;5;208m---------< File Operation ---------- \033[1m" << endl << endl;**

**cout << "File content:\n" << content << '\n';**

**cout << endl << "\033[38;5;208m---------- File Operation />------------ \033[0m" << endl << endl;**

**} else {**

**cerr << "Error: File not found.\n";**

**}**

**break;**

**case 2:**

**cout << "Enter the filename to create: ";**

**cin >> filename;**

**cout << endl << "\033[38;5;208m---------< File Operation ---------- \033[1m" << endl << endl;**

**createFile(filename);**

**cout << endl << "\033[38;5;208m---------- File Operation />------------ \033[0m" << endl << endl;**

**break;**

**case 3:**

**cout << "Enter the filename to delete: ";**

**cin >> filename;**

**cout << endl << "\033[38;5;208m---------< File Operation ---------- \033[1m" << endl << endl;**

**deleteFile(filename);**

**cout << endl << "\033[38;5;208m---------- File Operation />------------ \033[0m" << endl << endl;**

**break;**

**case 4:**

**cout << endl << "\033[38;5;208m---------< File Operation ---------- \033[1m" << endl << endl;**

**showAvailableCsvFiles(directory);**

**cout << endl << "\033[38;5;208m---------- File Operation />------------ \033[0m" << endl << endl;**

**break;**

**case 5: // New case for exporting data to CSV**

**cout << "Enter the filename to export data to CSV: ";**

**cin >> filename;**

**cout << endl << "\033[38;5;208m---------< File Operation ---------- \033[1m" << endl << endl;**

**exportDataToCsv(g,filename);**

**cout << endl << "\033[38;5;208m---------- File Operation />------------ \033[0m" << endl << endl;**

**break;**

**case 6:**

**cout << "Exiting the program.\n";**

**break;**

**default:**

**cerr << "Invalid choice. Please enter a number from 1 to 5.\n";**

**break;**

**}**

**} while (choice != 6);**

**saveGraphToFile(g, filename);**

**break;**

**case 17:**

**cout << endl << "\033[38;5;208m---------< File Operation ---------- \033[1m" << endl << endl;**

**exportDataToCsv(g,filename);**

**cout << endl << "\033[38;5;208m---------- File Operation />------------ \033[0m" << endl << endl;**

**break;**

**case 18:**

**if (fileExists(filename)) {**

**ifstream file(filename);**

**string content((istreambuf\_iterator<char>(file)), istreambuf\_iterator<char>());**

**cout << endl << "\033[38;5;208m---------< Reloading Graph ---------- \033[1m" << endl << endl;**

**cout << "File content:\n" << content << '\n';**

**setupGraphFromCSV(g, filename);**

**cout << endl << "\033[38;5;208m---------- Reloading Graph />------------ \033[0m" << endl << endl;**

**} else {**

**cerr << "Error: File not found.\n";**

**}**

**break;**

**default:**

**cout << "Enter Proper Option number " << endl;**

**}**

**if (option == 0){**

**break;**

**}**

**try{**

**cout << endl << ">>> Would you like to proceed (y/n) > " ;**

**cin >> h;**

**if(h != 'y' || h != 'n'){**

**throw('y');**

**}**

**cout << endl;**

**}**

**catch(char){**

**h == 'y';**

**}**

**} while (h != 'n');**

**return 0;**

**}**

# Roadcraft.cpp

**#include <iostream>**

**#include <vector>**

**#include <list>**

**#include <iterator>**

**#include <limits>**

**#include <queue>**

**#include <fstream>**

**#include <sstream>**

**#include <string>**

**using namespace std;**

**class Link;**

**class Node;**

**class Link {**

**public:**

**int DestinationNodeID;**

**double weight;**

**Link() {}**

**Link(int destNodeID, double w) {**

**DestinationNodeID = destNodeID;**

**weight = w;**

**}**

**void setLinkValues(int destNodeID, double w) {**

**DestinationNodeID = destNodeID;**

**weight = w;**

**}**

**void setWeight(double w) {**

**weight = w;**

**}**

**int getDestinationNodeID() {**

**return DestinationNodeID;**

**}**

**double getWeight() {**

**return weight;**

**}**

**};**

**class Node {**

**public:**

**int state\_id;**

**string state\_name;**

**list<Link> linkList;**

**Node() {**

**state\_id = 0;**

**state\_name = "";**

**}**

**Node(int id, string sname) {**

**state\_id = id;**

**state\_name = sname;**

**}**

**int getStateID() {**

**return state\_id;**

**}**

**string getStateName() {**

**return state\_name;**

**}**

**void setID(int id) {**

**state\_id = id;**

**}**

**void setStateName(string sname) {**

**state\_name = sname;**

**}**

**list<Link> getLinkList() {**

**return linkList;**

**}**

**void printLinkList() {**

**cout << "[";**

**for (auto it = linkList.begin(); it != linkList.end(); it++) {**

**cout << it->getDestinationNodeID() << "(" << it->getWeight() << ") --> ";**

**}**

**cout << "]";**

**cout << endl;**

**}**

**void updateNodeName(string sname) {**

**state\_name = sname;**

**cout << "Node Name Updated Successfully";**

**}**

**};**

**class Graph {**

**vector<Node> nodes;**

**public:**

**bool checkIfNodeExistByID(int nid) {**

**bool flag = false;**

**for (int i = 0; i < nodes.size(); i++) {**

**if (nodes.at(i).getStateID() == nid) {**

**return true;**

**}**

**}**

**return flag;**

**}**

**void addNode(Node newNode) {**

**bool check = checkIfNodeExistByID(newNode.getStateID());**

**if (check == true) {**

**cout << "Node with this ID already exists" << endl;**

**} else {**

**nodes.push\_back(newNode);**

**cout << "New Node Added Successfully" << endl;**

**}**

**}**

**Node getNodeByID(int nid) {**

**Node temp;**

**for (int i = 0; i < nodes.size(); i++) {**

**temp = nodes.at(i);**

**if (temp.getStateID() == nid) {**

**return temp;**

**}**

**}**

**return temp;**

**}**

**bool checkIfLinkExistByID(int fromNode, int toNode) {**

**Node n = getNodeByID(fromNode);**

**list<Link> l;**

**l = n.getLinkList();**

**bool flag = false;**

**for (auto it = l.begin(); it != l.end(); it++) {**

**if (it->getDestinationNodeID() == toNode) {**

**flag = true;**

**return flag;**

**break;**

**}**

**}**

**return flag;**

**}**

**void updateNode(int oldNID, string nname) {**

**bool check = checkIfNodeExistByID(oldNID);**

**if (check == true) {**

**for (int i = 0; i < nodes.size(); i++) {**

**if (nodes.at(i).getStateID() == oldNID) {**

**nodes.at(i).setStateName(nname);**

**break;**

**}**

**}**

**cout << "Node(State) Updated Successfully " << endl;**

**}**

**}**

**void addLinkByID(int fromNode, int toNode, double weight) {**

**bool check1 = checkIfNodeExistByID(fromNode);**

**bool check2 = checkIfNodeExistByID(toNode);**

**bool check3 = checkIfLinkExistByID(fromNode, toNode);**

**if ((check1 && check2 == true)) {**

**if (check3 == true) {**

**cout << "Link between " << getNodeByID(fromNode).getStateName() << "(" << fromNode << ") and "**

**<< getNodeByID(toNode).getStateName() << "(" << toNode << ") Already Exists" << endl;**

**} else {**

**for (int i = 0; i < nodes.size(); i++) {**

**if (nodes.at(i).getStateID() == fromNode) {**

**Link l(toNode, weight);**

**nodes.at(i).linkList.push\_back(l);**

**} else if (nodes.at(i).getStateID() == toNode) {**

**Link l(fromNode, weight);**

**nodes.at(i).linkList.push\_back(l);**

**}**

**}**

**cout << "Link between " << fromNode << " and " << toNode << " added Successfully" << endl;**

**}**

**} else {**

**cout << "Invalid Node ID entered." << endl;**

**}**

**}**

**void updateLinkByID(int fromNode, int toNode, double newWeight) {**

**bool check = checkIfLinkExistByID(fromNode, toNode);**

**if (check == true) {**

**for (int i = 0; i < nodes.size(); i++) {**

**if (nodes.at(i).getStateID() == fromNode) {**

**for (auto it = nodes.at(i).linkList.begin(); it != nodes.at(i).linkList.end(); it++) {**

**if (it->getDestinationNodeID() == toNode) {**

**it->setWeight(newWeight);**

**break;**

**}**

**}**

**} else if (nodes.at(i).getStateID() == toNode) {**

**for (auto it = nodes.at(i).linkList.begin(); it != nodes.at(i).linkList.end(); it++) {**

**if (it->getDestinationNodeID() == fromNode) {**

**it->setWeight(newWeight);**

**break;**

**}**

**}**

**}**

**}**

**cout << "Link Weight Updated Successfully " << endl;**

**} else {**

**cout << "Link between " << getNodeByID(fromNode).getStateName() << "(" << fromNode << ") and "**

**<< getNodeByID(toNode).getStateName() << "(" << toNode << ") DOES NOT Exist" << endl;**

**}**

**}**

**void deleteLinkByID(int fromNode, int toNode) {**

**bool check = checkIfLinkExistByID(fromNode, toNode);**

**if (check == true) {**

**for (int i = 0; i < nodes.size(); i++) {**

**if (nodes.at(i).getStateID() == fromNode) {**

**for (auto it = nodes.at(i).linkList.begin(); it != nodes.at(i).linkList.end(); it++) {**

**if (it->getDestinationNodeID() == toNode) {**

**nodes.at(i).linkList.erase(it);**

**break;**

**}**

**}**

**}**

**if (nodes.at(i).getStateID() == toNode) {**

**for (auto it = nodes.at(i).linkList.begin(); it != nodes.at(i).linkList.end(); it++) {**

**if (it->getDestinationNodeID() == fromNode) {**

**nodes.at(i).linkList.erase(it);**

**break;**

**}**

**}**

**}**

**}**

**cout << "Link Between " << fromNode << " and " << toNode << " Deleted Successfully." << endl;**

**}**

**}**

**void deleteNodeByID(int nid) {**

**int nIndex = 0;**

**for (int i = 0; i < nodes.size(); i++) {**

**if (nodes.at(i).getStateID() == nid) {**

**nIndex = i;**

**}**

**}**

**for (int i = 0; i < nodes.size(); i++) {**

**for (auto it = nodes.at(i).linkList.begin(); it != nodes.at(i).linkList.end(); it++) {**

**if (it->getDestinationNodeID() == nid) {**

**nodes.at(i).linkList.erase(it);**

**break;**

**}**

**}**

**}**

**nodes.erase(nodes.begin() + nIndex);**

**cout << "Node Deleted Successfully" << endl;**

**}**

**void getAllNeighborsByID(int nid) {**

**cout << getNodeByID(nid).getStateName() << " (" << getNodeByID(nid).getStateID() << ") --> ";**

**for (int i = 0; i < nodes.size(); i++) {**

**if (nodes.at(i).getStateID() == nid) {**

**cout << "[";**

**for (auto it = nodes.at(i).linkList.begin(); it != nodes.at(i).linkList.end(); it++) {**

**cout << it->getDestinationNodeID() << "(" << it->getWeight() << ") --> ";**

**}**

**cout << "]";**

**}**

**}**

**}**

**void shortestDistance(int sourceNode) {**

**vector<int> dist(nodes.size(), numeric\_limits<int>::max());**

**dist[sourceNode] = 0;**

**priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;**

**pq.push(make\_pair(0, sourceNode));**

**while (!pq.empty()) {**

**int u = pq.top().second;**

**pq.pop();**

**for (auto it = nodes[u].linkList.begin(); it != nodes[u].linkList.end(); it++) {**

**int v = it->getDestinationNodeID();**

**int weight = it->getWeight();**

**if (dist[u] + weight < dist[v]) {**

**dist[v] = dist[u] + weight;**

**pq.push(make\_pair(dist[v], v));**

**}**

**}**

**}**

**cout << "Shortest distances from node " << sourceNode << " to all other nodes:" << endl;**

**for (size\_t i = 0; i < dist.size(); i++) {**

**cout << "Node " << nodes[i].getStateName() << " (" << nodes[i].getStateID() << "): ";**

**if (dist[i] == numeric\_limits<int>::max()) {**

**cout << "Not reachable" << endl;**

**} else {**

**cout << dist[i] << endl;**

**}**

**}**

**}**

**int shortestDistanceBetweenIDs(int sourceID, int destinationID) {**

**vector<int> dist(nodes.size(), numeric\_limits<int>::max());**

**dist[sourceID] = 0;**

**priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;**

**pq.push(make\_pair(0, sourceID));**

**while (!pq.empty()) {**

**int u = pq.top().second;**

**pq.pop();**

**for (auto it = nodes[u].linkList.begin(); it != nodes[u].linkList.end(); it++) {**

**int v = it->getDestinationNodeID();**

**int weight = it->getWeight();**

**if (dist[u] + weight < dist[v]) {**

**dist[v] = dist[u] + weight;**

**pq.push(make\_pair(dist[v], v));**

**}**

**}**

**}**

**return dist[destinationID];**

**}**

**void printGraph() {**

**for (int i = 0; i < nodes.size(); i++) {**

**Node temp;**

**temp = nodes.at(i);**

**cout << temp.getStateName() << " (" << temp.getStateID() << ") --> ";**

**temp.printLinkList();**

**}**

**}**

**vector<int> shortestPath(int sourceNode, int destinationNode) {**

**vector<double> dist(nodes.size(), numeric\_limits<double>::max());**

**vector<int> prev(nodes.size(), -1);**

**dist[sourceNode] = 0;**

**priority\_queue<pair<double, int>, vector<pair<double, int>>, greater<pair<double, int>>> pq;**

**pq.push(make\_pair(0.0, sourceNode));**

**while (!pq.empty()) {**

**int u = pq.top().second;**

**double currDist = pq.top().first;**

**pq.pop();**

**if (u == destinationNode) {**

**break; // Stop once we reach the destination node**

**}**

**for (auto it = nodes[u].linkList.begin(); it != nodes[u].linkList.end(); it++) {**

**int v = it->getDestinationNodeID();**

**double weight = it->getWeight();**

**if (dist[u] + weight < dist[v]) {**

**dist[v] = dist[u] + weight;**

**prev[v] = u; // Store the previous node in the shortest path**

**pq.push(make\_pair(dist[v], v));**

**}**

**}**

**}**

**// Construct the path from the source to the destination**

**vector<int> path;**

**int curr = destinationNode;**

**while (curr != -1) {**

**path.push\_back(curr);**

**curr = prev[curr];**

**}**

**return path;**

**}**

**int findClosestNodeFromMultipleNodes(const vector<int>& nodeIDs) {**

**if (nodeIDs.empty()) {**

**cout << "Error: Empty input vector." << endl;**

**return -1;**

**}**

**vector<double> closestDistances(nodes.size(), numeric\_limits<double>::max());**

**for (int sourceNode : nodeIDs) {**

**vector<double> dist(nodes.size(), numeric\_limits<double>::max());**

**dist[sourceNode] = 0;**

**priority\_queue<pair<double, int>, vector<pair<double, int>>, greater<pair<double, int>>> pq;**

**pq.push(make\_pair(0.0, sourceNode));**

**while (!pq.empty()) {**

**int u = pq.top().second;**

**double currDist = pq.top().first;**

**pq.pop();**

**for (auto it = nodes[u].linkList.begin(); it != nodes[u].linkList.end(); it++) {**

**int v = it->getDestinationNodeID();**

**double weight = it->getWeight();**

**if (dist[u] + weight < dist[v]) {**

**dist[v] = dist[u] + weight;**

**pq.push(make\_pair(dist[v], v));**

**}**

**}**

**}**

**for (size\_t i = 0; i < dist.size(); i++) {**

**closestDistances[i] = min(closestDistances[i], dist[i]);**

**}**

**}**

**int closestNode = -1;**

**double minDistance = numeric\_limits<double>::max();**

**for (size\_t i = 0; i < closestDistances.size(); i++) {**

**if (closestDistances[i] < minDistance) {**

**minDistance = closestDistances[i];**

**closestNode = i;**

**}**

**}**

**return closestNode;**

**}**

**};**

**void setupGraphFromCSV(Graph& graph, const string& filename) {**

**ifstream file(filename);**

**if (!file.is\_open()) {**

**cerr << "Error: Unable to open file " << filename << endl;**

**return;**

**}**

**string line;**

**while (getline(file, line)) {**

**istringstream iss(line);**

**string sourceID, sourceName, destID, destName, weight;**

**if (getline(iss, sourceID, ',') &&**

**getline(iss, sourceName, ',') &&**

**getline(iss, destID, ',') &&**

**getline(iss, destName, ',') &&**

**getline(iss, weight, ',')) {**

**int sourceIDInt = stoi(sourceID);**

**int destIDInt = stoi(destID);**

**double weightInt = stod(weight);**

**bool sourceExists = graph.checkIfNodeExistByID(sourceIDInt);**

**if (!sourceExists) {**

**Node v(sourceIDInt, sourceName);**

**graph.addNode(v);**

**}**

**bool destExists = graph.checkIfNodeExistByID(destIDInt);**

**if (!destExists) {**

**Node v(destIDInt, destName);**

**graph.addNode(v);**

**}**

**graph.addLinkByID(sourceIDInt, destIDInt, weightInt);**

**}**

**}**

**file.close();**

**}**

**template <typename T>**

**void reverseVector(vector<T>& vec) {**

**size\_t start = 0;**

**size\_t end = vec.size() - 1;**

**while (start < end) {**

**swap(vec[start], vec[end]);**

**start++;**

**end--;**

**}**

**}**

**int main() {**

**Graph g;**

**string sname;**

**string filename;**

**int numNodes;**

**vector<int> nodeIDs;**

**int id1, id2, w;**

**int sourceID, destinationID;**

**vector<int> shortestPath;**

**int closestNode;**

**int shortestDist;**

**int option;**

**bool check;**

**do {**

**cout << "What operation do you want to perform? "**

**<< "Select Option number. Enter 0 to exit." << endl;**

**cout << "1. Add Node" << endl;**

**cout << "2. Update Node" << endl;**

**cout << "3. Delete Node" << endl;**

**cout << "4. Add Link" << endl;**

**cout << "5. Update Link" << endl;**

**cout << "6. Delete Link" << endl;**

**cout << "7. Check if 2 Nodes are Neighbors" << endl;**

**cout << "8. Print All Neighbors of a Node" << endl;**

**cout << "9. Print Graph" << endl;**

**cout << "10. Clear Screen" << endl;**

**cout << "11. Find Shortest Distance" << endl;**

**cout << "12. Find Shortest Distance Between Nodes" << endl;**

**cout << "13. Setup Graph from CSV" << endl;**

**cout << "14. Find Shortest Path" << endl;**

**cout << "15. Find Closest Node from Multiple Nodes" << endl;**

**cout << "0. Exit Program" << endl;**

**cout << endl << "Enter your option: ";**

**cin >> option;**

**Node n1;**

**switch (option) {**

**case 0:**

**break;**

**case 1:**

**cout << "Add Node Operation -" << endl;**

**cout << "Enter State ID: ";**

**cin >> id1;**

**cout << "Enter State Name: ";**

**cin >> sname;**

**n1.setID(id1);**

**n1.setStateName(sname);**

**g.addNode(n1);**

**break;**

**case 2:**

**cout << "Update Node Operation -" << endl;**

**cout << "Enter State ID of Node(State) to update: ";**

**cin >> id1;**

**cout << "Enter State Name: ";**

**cin >> sname;**

**g.updateNode(id1, sname);**

**break;**

**case 3:**

**cout << "Delete Node Operation -" << endl;**

**cout << "Enter ID of Node(State) to Delete: ";**

**cin >> id1;**

**g.deleteNodeByID(id1);**

**break;**

**case 4:**

**cout << "Add Link Operation -" << endl;**

**cout << "Enter ID of Source Node(State): ";**

**cin >> id1;**

**cout << "Enter ID of Destination Node(State): ";**

**cin >> id2;**

**cout << "Enter Weight of Link: ";**

**cin >> w;**

**g.addLinkByID(id1, id2, w);**

**break;**

**case 5:**

**cout << "Update Link Operation -" << endl;**

**cout << "Enter ID of Source Node(State): ";**

**cin >> id1;**

**cout << "Enter ID of Destination Node(State): ";**

**cin >> id2;**

**cout << "Enter UPDATED Weight of Link: ";**

**cin >> w;**

**g.updateLinkByID(id1, id2, w);**

**break;**

**case 6:**

**cout << "Delete Link Operation -" << endl;**

**cout << "Enter ID of Source Node(State): ";**

**cin >> id1;**

**cout << "Enter ID of Destination Node(State): ";**

**cin >> id2;**

**g.deleteLinkByID(id1, id2);**

**break;**

**case 7:**

**cout << "Check if 2 Nodes are Neighbors -" << endl;**

**cout << "Enter ID of Source Node(State): ";**

**cin >> id1;**

**cout << "Enter ID of Destination Node(State): ";**

**cin >> id2;**

**check = g.checkIfLinkExistByID(id1, id2);**

**if (check == true) {**

**cout << "Nodes are Neighbors (Link exists)" << endl;**

**} else {**

**cout << "Nodes are NOT Neighbors (Link does NOT exist)" << endl;**

**}**

**break;**

**case 8:**

**cout << "Print All Neighbors of a Node -" << endl;**

**cout << "Enter ID of Node(State) to fetch all Neighbors: ";**

**cin >> id1;**

**g.getAllNeighborsByID(id1);**

**break;**

**case 9:**

**cout << "Print Graph Operation -" << endl;**

**g.printGraph();**

**break;**

**case 10:**

**cout << "Clear Screen Operation -" << endl;**

**// Clearing the screen by printing a bunch of newlines**

**cout << string(100, '\n');**

**break;**

**case 11:**

**cout << "Find Shortest Distance -" << endl;**

**cout << "Enter the ID of the source node: ";**

**cin >> id1;**

**g.shortestDistance(id1);**

**break;**

**case 12:**

**cout << "Find Shortest Distance -" << endl;**

**cout << "Enter the ID of the source node: ";**

**cin >> sourceID;**

**cout << "Enter the ID of the destination node: ";**

**cin >> destinationID;**

**shortestDist = g.shortestDistanceBetweenIDs(sourceID, destinationID);**

**if (shortestDist == numeric\_limits<int>::max()) {**

**cout << "There is no path between the nodes." << endl;**

**} else {**

**cout << "Shortest distance between nodes with IDs " << sourceID << " and " << destinationID << " is: " << shortestDist << endl;**

**}**

**break;**

**case 13:**

**cout << "Setup Graph from CSV -" << endl;**

**filename = "E:\\OhmCraft\\chennaiCity.csv";**

**setupGraphFromCSV(g, filename);**

**break;**

**case 14:**

**cout << "Find Shortest Path -" << endl;**

**cout << "Enter the ID of the source node: ";**

**cin >> sourceID;**

**cout << "Enter the ID of the destination node: ";**

**cin >> destinationID;**

**shortestPath = g.shortestPath(sourceID, destinationID);**

**reverseVector(shortestPath);**

**if (shortestPath.empty()) {**

**cout << "There is no path between the nodes." << endl;**

**} else {**

**cout << "Shortest path between nodes with IDs " << sourceID << " and " << destinationID << ": ";**

**for (size\_t i = 0; i < shortestPath.size(); ++i) {**

**int node = shortestPath[i];**

**cout << g.getNodeByID(node).getStateName() << " (" << node << ")";**

**if (i != shortestPath.size() - 1) {**

**cout << " --> ";**

**}**

**}**

**cout << endl;**

**}**

**break;**

**case 15:**

**cout << "Find Closest Node from Multiple Nodes -" << endl;**

**cout << "Enter the number of nodes you want to consider: ";**

**cin >> numNodes;**

**for (int i = 0; i < numNodes; ++i) {**

**int nodeID;**

**cout << "Enter ID of node " << i + 1 << ": ";**

**cin >> nodeID;**

**nodeIDs.push\_back(nodeID);**

**}**

**closestNode = g.findClosestNodeFromMultipleNodes(nodeIDs);**

**if (closestNode != -1) {**

**cout << "Node " << g.getNodeByID(closestNode).getStateName() << " (" << closestNode << ") is closest to all the given nodes." << endl;**

**} else {**

**cout << "No nodes in the graph or all given nodes are disconnected." << endl;**

**}**

**break;**

**default:**

**cout << "Enter Proper Option number " << endl;**

**}**

**cout << endl;**

**} while (option != 0);**

**return 0;**

**}**

Output

**----------------------------------------------------------------------------**

**% Non-Directional Multi-Graph Manipulation Toolkit %**

**----------------------------------------------------------------------------**

**----------------------------------------------------------------------------**

**| Welcome (Please Read! Took So much to write....) |**

**| ========================================================== |**

**| This toolkit presents an API designed for handling Multi-Graphs, |**

**| equipped with novel algorithms for node-based operations. |**

**| Each node has the ability to establish connections with multiple |**

**| other nodes, and it allows for the existence of multiple links between |**

**| nodes. The underlying concept is highly adaptable and has been |**

**| successfully applied to two distinct applications. Notably, this |**

**| impressive toolkit is the sole creation of Chandru J. |**

**| and can be reached via email at chandrukavin0503@gmail.com. |**

**----------------------------------------------------------------------------**

**----------------------------------------------------------------------------**

**| OHMCRAFT [Option: 1] |**

**| ============================================================== |**

**| This toolkit provides an API for a Graph Manipulation Model dedicated |**

**| for Circuit Analysis With Peculiar New Algorithms for Nodal Fusion |**

**| and my personal Circuit Solving Algorithm called Nodal Reduction |**

**| using Prioritized Recursive Backtracking. |**

**----------------------------------------------------------------------------**

**----------------------------------------------------------------------------**

**| Road Network Analysis [Option: 2] |**

**| ============================================================== |**

**| This toolkit provides an API for a Graph Manipulation Model dedicated |**

**| to Road Network Analysis with Advanced Algorithms for Finding Shortest |**

**| Distances and Paths between Multiple Locations. It stores data such as |**

**| Road Distances between Nodes for comprehensive analysis. |**

**----------------------------------------------------------------------------**

**>>> Which one would you like to checkout? (1/2) >**

**Ohmcraft**

**----------------------------------------------------------------------------**

**% Non-Directional Multi-Graph Manipulation Toolkit %**

**----------------------------------------------------------------------------**

**-------------------------------------------------------------------------------**

**% OHMCRAFT - Multi-Graph Model for Circuit Analysis %**

**-------------------------------------------------------------------------------**

**------------------------------------**

**| Node Operations |**

**------------------------------------**

**| [1] Add Node |**

**| [2] Update Node |**

**| [3] Delete Node |**

**------------------------------------**

**| Link Operations |**

**------------------------------------**

**| [4] Add Link |**

**| [5] Update Link |**

**| [6] Delete Link |**

**------------------------------------**

**| Neighbor Check |**

**------------------------------------**

**| [7] Check if Neighbors |**

**| [8] Print Neighbors |**

**------------------------------------**

**| Graph Operations |**

**------------------------------------**

**| [9] Print Graph |**

**| [10] Setup Graph from CSV |**

**------------------------------------**

**| Fusion Operations |**

**------------------------------------**

**| [11] Fuse Two Nodes |**

**| [12] Auto Fuse Neighboring |**

**| Nodes |**

**| [13] Fuse Chain of Links |**

**| between Nodes |**

**| [14] Auto Fuse Chain of Links |**

**| [15] Execute Chandru's Nodal |**

**| Reduction Algorithm |**

**------------------------------------**

**| File Management |**

**------------------------------------**

**| [16] File Menu |**

**| [17] Save Current file |**

**| [18] Reload Current file |**

**------------------------------------**

**| [0] Exit Program |**

**------------------------------------**

**>>> Enter your option > 10**

**Setup Graph from CSV -**

**File Handling Menu**

**[1] Open a file**

**[2] Show available files**

**[0] Exit**

**Enter your choice (1-6): 1**

**Enter the filename to open: circuit1.csv**

**---------< Setting Up Graph ----------**

**File content:**

**1,A,2,B,-1**

**2,B,3,C,-1**

**2,B,4,D,-1**

**3,C,7,E,5**

**4,D,5,F,10**

**7,E,6,G,-1**

**5,F,6,G,-1**

**6,G,8,H,15**

**8,H,9,I,-1**

**Graph cleared successfully.**

**New Node Added Successfully**

**New Node Added Successfully**

**Link between 1 and 2 added Successfully**

**New Node Added Successfully**

**Link between 2 and 3 added Successfully**

**New Node Added Successfully**

**Link between 2 and 4 added Successfully**

**New Node Added Successfully**

**Link between 3 and 7 added Successfully**

**New Node Added Successfully**

**Link between 4 and 5 added Successfully**

**New Node Added Successfully**

**Link between 7 and 6 added Successfully**

**Link between 5 and 6 added Successfully**

**New Node Added Successfully**

**Link between 6 and 8 added Successfully**

**New Node Added Successfully**

**Link between 8 and 9 added Successfully**

**---------- Setting Up Graph />------------**

**File Handling Menu**

**[1] Open a file**

**[2] Show available files**

**[0] Exit**

**Enter your choice (1-6): 0**

**Exiting the program.**

**>>> Would you like to proceed (y/n) > y**

**>>> Enter your option > 15**

**Executing Chandru's Nodal Reduction Algorithm**

**New Node Added Successfully**

**Link between 3 and 10 added Successfully**

**Link between 4 and 10 added Successfully**

**Node Deleted Successfully**

**Node Deleted Successfully**

**Nodes 1 and 2 fused into Node 10.**

**---------< Fusing Sub-Steps --------------**

**C (3) --> [7(5) --> 10(-1) --> ]**

**D (4) --> [5(10) --> 10(-1) --> ]**

**E (7) --> [3(5) --> 6(-1) --> ]**

**F (5) --> [4(10) --> 6(-1) --> ]**

**G (6) --> [7(-1) --> 5(-1) --> 8(15) --> ]**

**H (8) --> [6(15) --> 9(-1) --> ]**

**I (9) --> [8(-1) --> ]**

**A\_B (10) --> [3(-1) --> 4(-1) --> ]**

**---------- Fusing Sub-Steps />------------**

**The link between the nodes does not have a resistance value of -1.**

**---------< Fusing Sub-Steps --------------**

**C (3) --> [7(5) --> 10(-1) --> ]**

**D (4) --> [5(10) --> 10(-1) --> ]**

**E (7) --> [3(5) --> 6(-1) --> ]**

**F (5) --> [4(10) --> 6(-1) --> ]**

**G (6) --> [7(-1) --> 5(-1) --> 8(15) --> ]**

**H (8) --> [6(15) --> 9(-1) --> ]**

**I (9) --> [8(-1) --> ]**

**A\_B (10) --> [3(-1) --> 4(-1) --> ]**

**---------- Fusing Sub-Steps />------------**

**New Node Added Successfully**

**Link between 5 and 11 added Successfully**

**Link between 3 and 11 added Successfully**

**Node Deleted Successfully**

**Node Deleted Successfully**

**Nodes 4 and 10 fused into Node 11.**

**---------< Fusing Sub-Steps --------------**

**C (3) --> [7(5) --> 11(-1) --> ]**

**E (7) --> [3(5) --> 6(-1) --> ]**

**F (5) --> [6(-1) --> 11(10) --> ]**

**G (6) --> [7(-1) --> 5(-1) --> 8(15) --> ]**

**H (8) --> [6(15) --> 9(-1) --> ]**

**I (9) --> [8(-1) --> ]**

**D\_A\_B (11) --> [5(10) --> 3(-1) --> ]**

**---------- Fusing Sub-Steps />------------**

**New Node Added Successfully**

**Link between 11 and 12 added Successfully**

**Link between 7 and 12 added Successfully**

**Link between 8 and 12 added Successfully**

**Node Deleted Successfully**

**Node Deleted Successfully**

**Nodes 5 and 6 fused into Node 12.**

**---------< Fusing Sub-Steps --------------**

**C (3) --> [7(5) --> 11(-1) --> ]**

**E (7) --> [3(5) --> 12(-1) --> ]**

**H (8) --> [9(-1) --> 12(15) --> ]**

**I (9) --> [8(-1) --> ]**

**D\_A\_B (11) --> [3(-1) --> 12(10) --> ]**

**F\_G (12) --> [11(10) --> 7(-1) --> 8(15) --> ]**

**---------- Fusing Sub-Steps />------------**

**One or both of the nodes do not exist in the graph.**

**---------< Fusing Sub-Steps --------------**

**C (3) --> [7(5) --> 11(-1) --> ]**

**E (7) --> [3(5) --> 12(-1) --> ]**

**H (8) --> [9(-1) --> 12(15) --> ]**

**I (9) --> [8(-1) --> ]**

**D\_A\_B (11) --> [3(-1) --> 12(10) --> ]**

**F\_G (12) --> [11(10) --> 7(-1) --> 8(15) --> ]**

**---------- Fusing Sub-Steps />------------**

**New Node Added Successfully**

**Link between 12 and 13 added Successfully**

**Node Deleted Successfully**

**Node Deleted Successfully**

**Nodes 9 and 8 fused into Node 13.**

**---------< Fusing Sub-Steps --------------**

**C (3) --> [7(5) --> 11(-1) --> ]**

**E (7) --> [3(5) --> 12(-1) --> ]**

**D\_A\_B (11) --> [3(-1) --> 12(10) --> ]**

**F\_G (12) --> [11(10) --> 7(-1) --> 13(15) --> ]**

**I\_H (13) --> [12(15) --> ]**

**---------- Fusing Sub-Steps />------------**

**The link between the nodes does not have a resistance value of -1.**

**---------< Fusing Sub-Steps --------------**

**C (3) --> [7(5) --> 11(-1) --> ]**

**E (7) --> [3(5) --> 12(-1) --> ]**

**D\_A\_B (11) --> [3(-1) --> 12(10) --> ]**

**F\_G (12) --> [11(10) --> 7(-1) --> 13(15) --> ]**

**I\_H (13) --> [12(15) --> ]**

**---------- Fusing Sub-Steps />------------**

**The link between the nodes does not have a resistance value of -1.**

**---------< Fusing Sub-Steps --------------**

**C (3) --> [7(5) --> 11(-1) --> ]**

**E (7) --> [3(5) --> 12(-1) --> ]**

**D\_A\_B (11) --> [3(-1) --> 12(10) --> ]**

**F\_G (12) --> [11(10) --> 7(-1) --> 13(15) --> ]**

**I\_H (13) --> [12(15) --> ]**

**---------- Fusing Sub-Steps />------------**

**New Node Added Successfully**

**Link between 7 and 14 added Successfully**

**Link between 12 and 14 added Successfully**

**Node Deleted Successfully**

**Node Deleted Successfully**

**Nodes 3 and 11 fused into Node 14.**

**---------< Fusing Sub-Steps --------------**

**E (7) --> [12(-1) --> 14(5) --> ]**

**F\_G (12) --> [7(-1) --> 13(15) --> 14(10) --> ]**

**I\_H (13) --> [12(15) --> ]**

**C\_D\_A\_B (14) --> [7(5) --> 12(10) --> ]**

**---------- Fusing Sub-Steps />------------**

**New Node Added Successfully**

**Link between 13 and 15 added Successfully**

**Link between 14 and 15 added Successfully**

**Link Weight Updated Successfully**

**Link between C\_D\_A\_B(14) and F\_G\_E(15) Updated Successfully**

**Node Deleted Successfully**

**Node Deleted Successfully**

**Nodes 12 and 7 fused into Node 15.**

**---------< Fusing Sub-Steps --------------**

**I\_H (13) --> [15(15) --> ]**

**C\_D\_A\_B (14) --> [15(3.33333) --> ]**

**F\_G\_E (15) --> [13(15) --> 14(3.33333) --> ]**

**---------- Fusing Sub-Steps />------------**

**One or both of the nodes do not exist in the graph.**

**---------< Fusing Sub-Steps --------------**

**I\_H (13) --> [15(15) --> ]**

**C\_D\_A\_B (14) --> [15(3.33333) --> ]**

**F\_G\_E (15) --> [13(15) --> 14(3.33333) --> ]**

**---------- Fusing Sub-Steps />------------**

**One or both of the nodes do not exist in the graph.**

**---------< Fusing Sub-Steps --------------**

**I\_H (13) --> [15(15) --> ]**

**C\_D\_A\_B (14) --> [15(3.33333) --> ]**

**F\_G\_E (15) --> [13(15) --> 14(3.33333) --> ]**

**---------- Fusing Sub-Steps />------------**

**The link between the nodes does not have a resistance value of -1.**

**---------< Fusing Sub-Steps --------------**

**I\_H (13) --> [15(15) --> ]**

**C\_D\_A\_B (14) --> [15(3.33333) --> ]**

**F\_G\_E (15) --> [13(15) --> 14(3.33333) --> ]**

**---------- Fusing Sub-Steps />------------**

**The link between the nodes does not have a resistance value of -1.**

**---------< Fusing Sub-Steps --------------**

**I\_H (13) --> [15(15) --> ]**

**C\_D\_A\_B (14) --> [15(3.33333) --> ]**

**F\_G\_E (15) --> [13(15) --> 14(3.33333) --> ]**

**---------- Fusing Sub-Steps />------------**

**All neighboring nodes fused successfully.**

**---------< Final Result --------------**

**Chain of links between nodes 13 and 14 fused into a two-nodal system with equivalent resistance: 18.3333**

**---------- Final Result />------------**

**roadcraft.cpp**

**Enter your option: 14**

**Find Shortest Path -**

**Enter the ID of the source node: 3**

**Enter the ID of the destination node: 23**

**Shortest path between nodes with IDs 3 and 23: Anna Nagar (3) --> Nungambakkam (10) --> Thiruvanmiyur (16) --> Chrompet (22) --> Thirumangalam (25) --> Aminjikarai (23)**