Dasha Gore

November 18, 2024

CS 499

Enhanced Project One

function main():

print "Welcome to the Pathfinding Program!"

filename = getUserInput("Enter the graph file name:")

graph = buildGraph(filename)

if graph is null:

print "Error: Failed to load the graph. Check the file format."

return

print "Graph loaded successfully."

startNode = getUserInput("Enter the starting node:")

endNode = getUserInput("Enter the ending node:")

if not validateNodes(graph, startNode, endNode):

print "Error: One or both nodes are invalid or not present in the graph."

return

print "Calculating the shortest path from", startNode, "to", endNode, "..."

shortestPath, totalCost = calculateShortestPath(graph, startNode, endNode)

if shortestPath is not null:

displayResults(shortestPath, totalCost)

else:

print "No path exists between", startNode, "and", endNode, "."

function buildGraph(filename):

try:

open file with filename

graph = initialize empty adjacency list

for each line in file:

parse line into (source, destination, weight)

if source or destination is invalid:

print "Warning: Skipping invalid line:", line

continue

add edge to graph[source] with (destination, weight)

add edge to graph[destination] with (source, weight) // For undirected graph

close file

return graph

catch error:

print "Error: Unable to process the graph file."

return null

function validateNodes(graph, startNode, endNode):

if startNode not in graph or endNode not in graph:

return false

return true

function calculateShortestPath(graph, startNode, endNode):

distances = initialize dictionary with infinity for all nodes

previousNodes = initialize dictionary with null for all nodes

distances[startNode] = 0

priorityQueue = initialize priority queue with (startNode, 0)

while priorityQueue is not empty:

currentNode, currentDistance = extractMin(priorityQueue)

if currentDistance > distances[currentNode]:

continue

for neighbor, weight in graph[currentNode]:

newDistance = currentDistance + weight

if newDistance < distances[neighbor]:

distances[neighbor] = newDistance

previousNodes[neighbor] = currentNode

insert (neighbor, newDistance) into priorityQueue

if distances[endNode] == infinity:

return null, null // No path exists

return reconstructPath(previousNodes, endNode), distances[endNode]

function reconstructPath(previousNodes, endNode):

path = initialize empty list

currentNode = endNode

while currentNode is not null:

insert currentNode at the beginning of path

currentNode = previousNodes[currentNode]

return path

function displayResults(shortestPath, totalCost):

print "Shortest path:"

for each node in shortestPath:

if node is not the last node:

print node, "->",

else:

print node

print "Total cost:", totalCost

function getUserInput(prompt):

print prompt

return input()

function extractMin(priorityQueue):

// Efficiently retrieves and removes the node with the smallest distance

return node with smallest distance