

Programmer Manual

Programmer Manual

Graph

1. Problem Description

This program provides an ADT using adjacency lists to represent a directed or undirected graph. Data, such as cities, can be stored in a network of vertices connected with weighted edges. This graph can be added to and removed from, as well as traversed both breadth first and depth first. Dijkstra's algorithm is used to find the shortest path between any two of the vertices as well.

2. Data Types and Classes

The data types used in this program fall into two categories: predefined data types and programmer defined data types. The following subsections address the data types used.

A. Predefined Data Types

2.1 int

Variables:

choice	user input for the main menu
visited	flag determining whether a vertex has been visited
firstFound	first edge found
secondFound	second edge found
v1Index	index location of vertex 1
v2Index	index location of vertex 2

2.2 bool

Variables:

populated	flag determining whether a graph has data in it or not
-----------	--

2.3 double

Variables:

distance	shortest distance determined by Dijkstra's algorithm
minDist	the minimum distance to a particular vertex
prevDist	the minimum distance to the previous vertex

2.4 string

Variables:

filename	name of the file containing the graph data
----------	--

2.5 stack

Variables:

path	stack used to hold the shortest path
------	--------------------------------------

2.6 queue/deque

Variables:

q	queue used for the breadth first traversal, Dijkstra's algorithm, and a deque for Ford's algorithm
---	--

2.7 vector

Variables:

G	holds all of the vertices
---	---------------------------

2.8 list

Variables:

edgelist	the adjacency list of each vertex
----------	-----------------------------------

B. Programmer Defined Data Types

2.1 V

Variables:

name	vertex name
prev	previous vertex name
v	arbitrary vertex

2.2 W

Variables:

weight	weight of each edge
--------	---------------------

2.3 edgeRep

This struct has:

Data members:	V name
	W weight

See the programmer manual for the Graph class for more details.

2.3 vertex

This struct has:

Data members:	V name
	int visited
	list edgelist
	double minDist
	V prev

See the programmer manual for the Graph c lass for more details

2.4 Graph

This class has:

Data members:	bool populated
Member functions:	Graph
	~Graph
	isVertex
	isUniEdge
	isBiDirEdge
	AddVertex
	DeleteVertex
	AddUniEdge
	DeleteUniEdge
	AddBiDirEdge
	DeleteBiDirEdge
	SimplePrintGraph
	ShortestDistance
	GetGraph
	BFTraversal
	DFTraversal
	DFUtility
	MST
	FordShortestPath

See the programmer manual for the Graph class for more details.

3. High Level Program Solution

Main Program

Print the menu

Option 1. Read in an input file holding the graph data

Option 2. Print the graph

Option 3. Test if a vertex is in the graph

Option 4. Test if an edge exists between two vertices

Sub-Option 1. Test unidirectional edge

Sub-Option 2. Test bidirectional edge

Sub-Option 3. Return to main menu

Option 5. Add a vertex

Check if a vertex already exists

If not, push the new vertex into G

Option 6. Add an edge

Sub-Option 1. Add unidirectional edge

Sub-Option 2. Add bidirectional edge

Sub-Option 3. Return to main menu

Check if vertices exist and edge does not exist

If the vertices do not exist, create them

Push the new edge into the adjacency lists of the appropriate vertices

Option 7. Delete a vertex

Check if the vertex exists

If so, delete it from G

Delete all edges incident with the vertex from the rest of the adjacency lists

Option 8. Delete an edge

Sub-Option 1. Delete unidirectional edge

Sub-Option 2. Delete bidirectional edge

Sub-Option 3. Return to main menu

Check if the vertices and edge exists

If so, delete the edge from the edgelists

Option 9. Traverse the graph

Sub-Option 1. Breadth first traversal

Sub-Option 2. Depth first traversal

Sub-Option 3. Return to main menu

Option 10. Find the shortest path

Sub-Option 1. Dijkstra's algorithm between two vertices

Sub-Option 2. Ford's algorithm from one vertex to all other vertices

Sub-Option 3. Return to main menu

Option 11. Find the minimum spanning tree of the graph using Prim's algorithm

Option 12. Exit program

4. Limitations and Suggestions

The current program needs a specifically formatted file to read in the graph data. These restrictions could be relaxed and the program could read in different types of files. The program could also be modified in order to save the graph to another file which could be read back in. The minimum spanning tree could also be saved instead of just displayed when the option is selected