Graph

Programmer Manual
Graph

1. Problem Description

The Graph class consists of two structs, one representing a vertex of a graph, and one representing an edge connecting two vertices. The vertices are stored in a vector, and each vector contains a linked list which represents the adjacency list of each vertex. The class also contains all of the functions that the user would use to interact with the graph through.

2. Class Graph

Private data members:

bool populated	flag determining whether the graph has data or not

Private member functions:

DFUtility	recursive function for the depth first traversal

Public member functions:

Graph	constructor for a Graph object
~Graph	destructor for a Graph object
isVertex	tests if a vertex is in the Graph
isUniEdge	tests if a unidirectional edge is between two vertices in the Graph
isBiDirEdge	tests if a bidirectional edge is between two vertices
AddVertex	adds a vertex to the Graph
DeleteVertex	removes a vertex from the Graph
AddBiDirEdge	adds a bidirectional edge between two vertices
DeleteBiDirEdge	removes a bidirectional edge between two vertices
SimplePrintGraph	prints the Graph not using any specific traversal
ShortestDistance	calculates the shortest path between two vertices using Dijkstra's algorithm
GetGraph	reads in the Graph data from a file
BFTraversal	prints the breadth first traversal of the graph
DFTraversal	prints the depth first traversal of the graph

3. High Level Program Solution

Graph
  sets populated to false

isVertex
  returns the index location of the vertex, or -1 if the vertex is not in the graph

isUniEdge
  searches for an edge going from vertex 1 to vertex 2 and then from vertex 2 to vertex 1
  returns the XOR of these values so only a unidirectional path returns a 1, otherwise return 0

```
isBiDirEdge
    searches for an edge going from vertex 1 to vertex 2 and then from vertex 2 to vertex 1
    returns the AND of these values so only a bidirectional path returns a 1, otherwise return 0

AddVertex
    if a vertex does not exist, push it into the graph
    set the graph as populated if it is not already

DeleteVertex
    if a vertex exists, delete it from the graph
    delete the edges incident with the vertex in the rest of the vertex adjacency lists

AddUniEdge
    if the vertices the edge is to be connected to do not exist, create them
    if an edge already exists, delete them
    create the new edge and push it into the appropriate edgelists

DeleteUniEdge
    checks if a unidirectional edge exists between two vertices
    if so, remove it from the appropriate adjacency lists

AddBiDirEdge
    if the vertices the edge is to be connected to do not exist, create them
    if an edge already exists, delete them
    create the new edge and push it into the appropriate edgelists

DeleteBiDirEdge
    checks if a bidirectional edge exists between two vertices
    if so, remove it from the appropriate adjacency lists

SimplePrintGraph
    prints out a vertex
    prints out the adjacency list for that vertex
    repeat until no more vertices

ShortestDistance
    set minimum distances to infinity
    set starting vertex minimum distance to 0 and push into the queue
    while the queue is not empty
        pop from the queue
        mark the current vertex as visited
        look through the edgelist
        if a vertex is in the edgelist and not visited
        get the weight
        if the minimum distance plus the weight is less that the minimum distance to the vertices
            in the edgelist, push into the queue
        set the new minimum distance to the first vertex plus the weighted
        the previous node is now the node just considered
    if the minimum distance from the first vertex to the target vertex is INT_MAX, there is no path
    find the previous vertex from the final vertex
    put the final vertex into the stack
    put each previous into the stack until there are no more previous
    put the first vertex into the stack
    pop the stack until empty and print the shortest path
    return the distance of the shortest path

GetGraph
    if the graph has data, delete all of it
    get the file name from the user
    push a vertex into the graph's vector
    push that vertex's adjacency list into that vertex's edgelist
    continue until the file is empty
    set populated to true

BFTraversal
    set all of the vertices to unvisited
    mark the current vertex visited
    push the start in the queue
    while the queue is not empty
        pop the queue
        look through the edgelist and if the vertex is not visited, mark it visited and push it into the queue
    print any vertices unconnected with the starting vertex

DFUtility
    mark the starting vertex as visited
    look through the adjacency list
    call DFUtility recursively on the next vertex

DFTraversal
    mark all of the vertices as unvisited
    call DFUtility on the starting vertex
    print any vertices unconnected with the starting vertex
```