Programmer Manual

Digraph Class

A. Class Description

The Diagraph class is a templated implementation of a directed graph using an adjacency list. It provides member functions for adding and deleting vertices and edges, checking for their existence, and performing searches. The class also includes functions for reading graphs from input files and prompting the user for input.

B. Class standings

1. Protected Section:

- Int num_of_verticies: integer representing the number of vertices in the graph

- std::vector< vertex<V, W> > graph_array: graph being a vector

- Diagraph<V, W>* minTree: Pointer to a Diagraph

- W disOf(const int& v, const int& w, const std::vector< vertex<V, W>& g): takes in a graph represented as a vector and two vertex indices "v" and "w", calculates the total distance between them in the minimum spanning tree.

- W sumOf(const std::vector<W>& dist): takes in a vector "dist" representing edge weights or distances, calculates the sum of all the weights in the vector

- void printPath(vertex<V, W>* vertex):

- void printMst(const std::vector<bool>& comp, const std::vector<W>& dist): takes in a pointer to a vector type of a graph "vertex<V, W>*", evaluates the predecessor of a given vertex until no predecessor is found (meaning the predecessor is itself), and prints the path in the graph leading up to that vertex

- const std::vector<V>& neigh, int& count): prints trace information related to the "mst()" function.

2. Public Section:

- Digraph() and ~Digraph(): constructor and destructor for the Digraph class.
- set_size(ifstream& inFile): sets the number of vertices in the graph by reading from an input file.
- getDigraph(ifstream& inFile): reads in the vertices and edges of the graph from an input file.
- get_size(): returns the number of vertices in the graph.
- isVertex(V& v): checks if a vertex with name v exists in the graph.

- isUniEdge(V& v1, V& v2): checks if there is a unidirectional edge from vertex v1 to vertex v2.
- isBiDirEdge(V& v1, V& v2): checks if there is a bidirectional edge between vertices v1 and v2.
- addVertex(V& v): adds a new vertex with name v to the graph.
- deleteVertex(V& v): removes the vertex with name v from the graph.
- addUniEdge(V& v1, V& v2, W& w): adds a new unidirectional edge from vertex v1 to vertex v2 with weight w.
- addBiDirEdge(V& v1, V& v2, W& w): adds a new bidirectional edge between vertices v1 and v2 with weight w.
- deleteUniEdge(V & v1, V& v2, W& w): removes the unidirectional edge from vertex v1 to vertex v2 with weight w.
- deleteBiDirEdge(V& v1, V& v2, W& w): removes the bidirectional edge between vertices v1 and v2 with weight w.
- printDigraph(): prints out the vertices and edges of the graph.
- breadth(V& start_vertex): performs a breadth-first search starting from the vertex start_vertex.
- depth(V& start_vertex): performs a depth-first search starting from the vertex start_vertex.
- getOneVertex(V& v1): reads in a single vertex name from the user.
- getTwoVerticies(V& v1, V& v2): reads in two vertex names from the user.
- Mst(): Prim's algorithm to determine the minimum spanning tree of an undirected graph, prints trace information during the process, and prints the weight and adjacency list of the resulting MST.
- FordShortestPath(V& v): shortest paths to all other vertices from a given vertex, prints some information along the way.
- SimplePrintGraph() const: prints the list of vertices and their edges in proper parenthesized notation.

C. High Level Program Solution

The Digraph class is a templated implementation of a directed graph using an adjacency list. It provides member functions for adding and deleting vertices and edges, checking for their existence, and performing searches. The class also includes functions for reading graphs from input files and prompting the user for input.

To use the Digraph class, first create an instance of the class by calling the constructor. Then, add vertices and edges to the graph using the appropriate member functions. You can check if a vertex or edge exists in the graph using the "isVertex", "isUniEdge", and "isBiDirEdge" functions.

To perform a breadth-first or depth-first search, call the "breadth" or "depth" functions, respectively, with the starting vertex as a parameter.

To determine the minimum spanning tree of an undirected graph, call the "Mst" function. This function uses Prim's algorithm and prints trace information during the process, as well as the weight and adjacency list of the resulting MST.

To find the shortest path from a given vertex to all other vertices, call the "FordShortestPath" function with the starting vertex as a parameter. This function prints some information along the way. To print out the vertices and edges of the graph in proper parenthesized notation, call the "SimplePrintGraph" function.

D. Limitations and Suggestions

Limitations:

- The implementation assumes that an Adjacency List representation is being used for the graph.
- The class only supports weighted edges.
- The graph does not have a way to store data associated with each vertex or edge.

Suggestions:

- Add support for other types of graph representations (e.g., Adjacency Matrix).
- Add support for unweighted edges.
- Consider adding member functions to store and retrieve data associated with vertices and edges.