Weighted Graph

AUTHOR

Version 1.00

11/21/2014

Table of Contents

Class Index

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

File Index

File List

Here is a list of all documented files with brief descriptions:

Class Documentation

WeightedGraph::Vertex Class Reference

Public Member Functions

void **setLabel** (const string &newLabel)
string **getLabel** () const
void **setColor** (char newColor)
char **getColor** () const

Private Attributes

string **label**
char **color**

Detailed Description

Definition at line 27 of file WeightedGraph.h.

The documentation for this class was generated from the following file:

1 WeightedGraph.h

WeightedGraph Class Reference

Classes

class Vertex

Public Member Functions

WeightedGraph (int maxNumber=MAX_GRAPH_SIZE)

The parameterized constructor that creates an empty WeightedGraph that allocates enough memory for a graph containing maxNumber verticies.

WeightedGraph (const WeightedGraph &other)

The copy constructor that initializes this WeightedGraph to be equivalent to the other WeightedGraph parameter.

WeightedGraph & operator= (const WeightedGraph &other)

The overloaded assignment operator that sets this WeightedGraph to be equivalent to the other WeightedGraph object parameter.

~WeightedGraph ()

The destructor that deallocates the memory used to store this WeightedGraph.

void insertVertex (const Vertex &newVertex) throw (logic_error)

Inserts newVertex into this WeightedGraph.

void insertEdge (const string &v1, const string &v2, int wt) throw (logic_error)

Inserts an undirected edge containing vertices v1 and v2 into the graph.

bool retrieveVertex (const string &v, Vertex &vData) const

Searches graph for vertex v.

bool getEdgeWeight (const string &v1, const string &v2, int &wt) const throw (logic_error)

Searches the graph for the edge connecting vertices v1 and v2.

void removeVertex (const string &v) throw (logic_error)

Removes vertex v from the graph and any edges connected to v.

void removeEdge (const string &v1, const string &v2) throw (logic_error)

Removes the edge connecting vertices v1 and v2 form the graph.

void clear ()

Removes all the vertices and edges in the graph.

bool isEmpty () const

Returns true if the graph is empty (no vertices).

bool isFull () const

Returns true if the graph is full (cannot add any more vertices).

void **showStructure** () const

void showShortestPaths () const

Computes and display's the path matrix of the graph.

bool hasProperColoring () const

If no vertex has the same color as an adjacent vertex in the graph, returns true.

bool areAllEven () const

If every vertex is of even degree in the graph, returns true.

Static Public Attributes

static const int **MAX_GRAPH_SIZE** = 10

static const int **INFINITE_EDGE_WT** = INT_MAX

Private Member Functions

int getIndex (const string &v) const

Returns an adjacency matrix index from a vertex label.

int getEdge (int row, int col) const

Returns an edge weight from adjacency matrix indices.

int getPath (int row, int col) const

Returns the path weight from the path matrix indicies.

void setEdge (int row, int col, int wt)

Sets the edge weight from adjacency matrix indices.

void setPath (int row, int col, int wt) const

Sets the path weight from the path matrix indices.

Private Attributes

int **maxSize**
int **size**
Vertex * **vertexList**
int * **adjMatrix**
int * **pathMatrix**

Detailed Description

Definition at line 20 of file WeightedGraph.h.

Constructor & Destructor Documentation

WeightedGraph::WeightedGraph (int *maxNumber* = `MAX_GRAPH_SIZE`)

The parameterized constructor that creates an empty WeightedGraph that allocates enough memory for a graph containing maxNumber verticies.

Postcondition:

This WeightedGraph will be a valid empty WeightedGraph with enough memory for maxNumber verticies.

Parameters:

maxNumber	is how much memory to allocate for this WeightedGraph.

WeightedGraph::WeightedGraph (const WeightedGraph & *other*)

The copy constructor that initializes this WeightedGraph to be equivalent to the other WeightedGraph parameter.

Precondition:

other is a valid WeightedGraph.

Postcondition:

This WeightedGraph will be a deep copy of the other WeightedGraph.

Parameters:

Definition at line 30 of file WeightedGraph.cpp.

other	is the WeightedGraph that this WeightedGraph will be made equivalent to.

WeightedGraph::~WeightedGraph ()

The destructor that deallocates the memory used to store this WeightedGraph.

Postcondition:

This WeightedGraph will be an empty, deallocated, WeightedGraph.

Definition at line 99 of file WeightedGraph.cpp.

Member Function Documentation

bool WeightedGraph::areAllEven () const

If every vertex is of even degree in the graph, returns true.

Otherwise, returns false.

Returns:

True if every vertex is of even degree. False if every vertex is not of even degree.

Definition at line 386 of file WeightedGraph.cpp.

void WeightedGraph::clear ()

Removes all the vertices and edges in the graph.

Postcondition:

The graph will be cleared of all vertices and edges.

Definition at line 284 of file WeightedGraph.cpp.

int WeightedGraph::getEdge (int *row*, int *col*) const `[private]`

Returns an edge weight from adjacency matrix indices.

Precondition:

row and col are valid indices.

Parameters:

Definition at line 50 of file WeightedGraph.cpp.

row	is the row index of the adjacency matrix.
col	is the column index of the adjacency matrix.

The edge weight corresponding to the index locations.

Definition at line 429 of file WeightedGraph.cpp.

bool WeightedGraph::getEdgeWeight (const string & *v1*, const string & *v2*, int & *wt*) const throw (logic_error)

Searches the graph for the edge connecting vertices v1 and v2.

If this edge exists, then places the weight of the edge in wt and returns true. Otherwise, returns false with wt undefined.

Precondition:

The graph includes vertices v1 and v2.

Postcondition:

wt will either be the value of the edge's weight or undefined.

Parameters:

Returns:

v1	is a vertex already in the graph.
v2	is a vertex (not equivalent to v1) already in the graph.
wt	is the weight of the found edge will be placed into.

True if the edge is found. False if the edge is not found.

Exceptions:

Returns:

The	graph does not include vertex v1 or v2.

int WeightedGraph::getIndex (const string & *v*) const [private]

Returns an adjacency matrix index from a vertex label.

Precondition:

v is a valid vertex label.

Parameters:

Definition at line 202 of file WeightedGraph.cpp.

v	is the vertex label to get the adjacency matrix index from.

The index location in the adjacency matrix. -1 if not found.

Definition at line 412 of file WeightedGraph.cpp.

int WeightedGraph::getPath (int *row*, int *col*) const [private]

Returns the path weight from the path matrix indicies.

Precondition:

row and col are valid indices.

Parameters:

Returns:

row	is the row index of the path matrix.

col	is the column index of the path matrix.

The path weight corresponding to the index locations.

Definition at line 441 of file WeightedGraph.cpp.

bool WeightedGraph::hasProperColoring () const

If no vertex has the same color as an adjacent vertex in the graph, returns true.

Otherwise, returns false.

Precondition:

All vertices have been assigned a color.

Returns:

True if no vertex has the same color as an adjacent vertex. False if a vertex has the same color as an adjacent vertex.

Definition at line 367 of file WeightedGraph.cpp.

void WeightedGraph::insertEdge (const string & *v1*, const string & *v2*, int *wt*) throw (logic_error)

Inserts an undirected edge containing vertices v1 and v2 into the graph.

The weight of the edge is wt. If there is already an edge containing vertices, then updates the weight of the edge.

Precondition:

The graph includes vertices v1 and v2.

Postcondition:

The undirected edge is inserted into the graph.

Parameters:

Returns:

v1	is a vertex already in the graph.
v2	is a vertex (not equivalent to v1) already in the graph.
wt	is the weight of the edge.

Exceptions:

The	graph is full.

void WeightedGraph::insertVertex (const Vertex & *newVertex*) throw (logic_error)

Inserts newVertex into this WeightedGraph.

If the Vertex already exists in graph, then updates it.

Precondition:

This WeightedGraph is not full.

Postcondition:

newVertex will be added into this WeightedGraph or updated.

Parameters:

Definition at line 153 of file WeightedGraph.cpp.

newVertex	is the Vertex to be added to this WeightedGraph or updated.

Exceptions:

The	WeightedGraph is full.

bool WeightedGraph::isEmpty () const

Returns true if the graph is empty (no vertices).

Otherwise, returns false.

Returns:

True if the graph is empty. False if the graph is not empty.

Definition at line 293 of file WeightedGraph.cpp.

bool WeightedGraph::isFull () const

Returns true if the graph is full (cannot add any more vertices).

Otherwise returns false.

Returns:

True if the graph is full. False if the graph is not full.

Definition at line 302 of file WeightedGraph.cpp.

WeightedGraph & WeightedGraph::operator= (const WeightedGraph & *other*)

The overloaded assignment operator that sets this WeightedGraph to be equivalent to the other WeightedGraph object parameter.

Precondition:

other is a valid WeightedGraph.

Postcondition:

This WeightedGraph will be a deep copy of the other WeightedGraph.

Parameters:

Definition at line 114 of file WeightedGraph.cpp.

other	is the WeightedGraph that this WeightedGraph will be made equivalent to.

The reference to this WeightedGraph.

Definition at line 64 of file WeightedGraph.cpp.

void WeightedGraph::removeEdge (const string & *v1*, const string & *v2*) throw (logic_error)

Removes the edge connecting vertices v1 and v2 frorm the graph.

Precondition:

Graph includes vertices v1 and v2.

Postcondition:

The edge connecting v1 and v2 will no longer be in the graph.

Parameters:

Returns:

v1	is a vertex already in the graph. v2 is a vertex (not equivalent to v1) already in the graph.

Exceptions:

The	graph does not include vertex v1 or v2.

void WeightedGraph::removeVertex (const string & *v*) throw (logic_error)

Removes vertex v from the graph and any edges connected to v.

Precondition:

The graph includes vertex v.

Postcondition:

The graph will no longer include vertex v or any edges connected to it.

Parameters:

Definition at line 268 of file WeightedGraph.cpp.

v	is the vertex to be removed.

Exceptions:

The	graph does not include vertex v.

bool WeightedGraph::retrieveVertex (const string & *v*, Vertex & *vData*) const

Searches graph for vertex v.

If this vertex is found, then places the value of vertex's data in vData and returns true. Otherwise, returns false and vData is undefined.

Postcondition:

vData will either be value of the vertex's data or undefined.

Parameters:

Definition at line 228 of file WeightedGraph.cpp.

v	is the vertex to search the graph for.
vData	is the vertex that the found vertex will be placed into.

True if the vertex is found. False if the vertex is not found.

Definition at line 176 of file WeightedGraph.cpp.

void WeightedGraph::setEdge (int *row*, int *col*, int *wt*) `[private]`

Sets the edge weight from adjacency matrix indices.

Precondition:

row and col are valid indices and wt is a valid weight.

Postcondition:

The edge between the two indices will be set.

Parameters:

Returns:

row	is the row index of the adjacency matrix.
col	is the column index of the adjacency matrix.
wt	is the weight of the new edge.

void WeightedGraph::setPath (int *row*, int *col*, int *wt*) const `[private]`

Sets the path weight from the path matrix indices.

Precondition:

row and col are valid indices and wt is a valid weight.

Postcondition:

The path between the two indices will be set.

Parameters:

Definition at line 454 of file WeightedGraph.cpp.

row	is the row index of the path matrix.
col	is the column index of the path matrix.
wt	is the weight of the new path.

void WeightedGraph::showShortestPaths () const

Computes and display's the path matrix of the graph.

Postcondition:

Paths are changed to the shortest path and outputted to the console.

Definition at line 311 of file WeightedGraph.cpp.

The documentation for this class was generated from the following files:

2 WeightedGraph.h
3 show12.cpp
4 WeightedGraph.cpp

File Documentation

WeightedGraph.cpp File Reference

This program will implement a Weighted Graph.

```
#include "WeightedGraph.h"
#include "show12.cpp"
```

Detailed Description

This program will implement a Weighted Graph.

Author:

Saharath Kleips

The specifications of this project match those of the book C++ Data Structures - A Laboratory Course (3rd Edition) Project 11. A Weighted Graph is a linear data structure in which data items are maintained in descending order based on priority. You can only access data at the front of the queue, examining this data item entails removing it from the queue.

Definition in file WeightedGraph.cpp.

Index

INDE