PA10-WeightedGraph

Generated by Doxygen 1.8.11

Contents

1 Main Page

This project contains the following items -create an implementation of the Weighted Graph ADT usin ga vertex list and an adjacency matrix. -Develop a routine that finds the shorted path between each pair of vertices in a graph. -add vertex coloring and implement a function that checks whether a graph has a proper coloring. -investiate the four color theorem by generating a graph for which no proper coloring can be creating using less than five colors.

2 Class Index

2.1 Class List

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

3 File Index

3.1 File List

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

4 Class Documentation

4.1 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**

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

- WeightedGraph.h

4.2 Vertex Class Reference

Public Attributes

- char **label** [vertexLabelLength]
- char **color**

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

- WeightedGraph2.h
- WeightedGraph3.h

4.3 WeightedGraph Class Reference

Classes

- class Vertex

Public Member Functions

- WeightedGraph (int maxNumber=defMaxGraphSize)
- WeightedGraph (const WeightedGraph &other)
- WeightedGraph & operator= (const WeightedGraph &other)
- ~WeightedGraph ()
- void insertVertex (const Vertex &newVertex) throw (logic_error)
- void insertEdge (const string &v1, const string &v2, int wt) throw (logic_error)
- bool retrieveVertex (const string &v, Vertex &vData) const
- bool getEdgeWeight (const string &v1, const string &v2, int &wt) const throw (logic_error)
- void removeVertex (const string &v) throw (logic_error)
- void removeEdge (const string &v1, const string &v2) throw (logic_error)
- void clear ()
- bool isEmpty () const
- bool isFull () const
- void showStructure () const
- void showShortestPaths ()
- bool hasProperColoring () const
- bool areAllEven () const
- **WeightedGraph** (int maxNumber=defMaxGraphSize)
- **WeightedGraph** (const WeightedGraph &other)
- WeightedGraph & **operator=** (const WeightedGraph &other)
- void **insertVertex** (Vertex newVertex) throw (logic_error)
- void **insertEdge** (char *v1, char *v2, int wt) throw (logic_error)
- bool **retrieveVertex** (char *v, Vertex &vData) const
- int **edgeWeight** (char *v1, char *v2, int &wt) const throw (logic_error)
- bool **getEdgeWeight** (char *v1, char *v2, int &wt) const throw (logic_error)
- void **removeVertex** (char *v) throw (logic_error)
- void **removeEdge** (char *v1, char *v2) throw (logic_error)
- void **clear** ()
- void **computePaths** ()
- bool **isEmpty** () const
- bool **isFull** () const
- void **showStructure** () const

Static Public Attributes

- static const int **defMaxGraphSize** = 10
- static const int **vertexLabelLength** = 11
- static const int **INFINITE_EDGE_WT** = INT_MAX
- static const int **DEF_MAX_GRAPH_SIZE** = 10
- static const int **VERTEX_LABEL_LENGTH** = 11

Private Member Functions

- int getIndex (const string &v) const
- int getEdge (int row, int col) const
- int getPath (int row, int col) const
- void setEdge (int row, int col, int wt)
- void setPath (int row, int col, int wt)
- int **getIndex** (char *v) const
- int **getEdge** (int row, int col) const
- int **getPath** (int row, int col) const
- void **setEdge** (int row, int col, int wt)
- void **setPath** (int row, int col, int wt)

Private Attributes

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

4.3.1 Constructor & Destructor Documentation

4.3.1.1 WeightedGraph::WeightedGraph (int *maxNumber* = `defMaxGraphSize`)

This function is the param constructor for the WeightedGraph class.

The function will set the maxSize equal to the parameter maxNumber passed in. The function will set the size to 0 since we are not putting in any data yet. The function will dynamically allocate memory for the vertexList to be equal to the maxSize. The function will dynamically allocate memory for the adjMatrix and pathMatrix equal to the maxSize * maxSize. The function will then loop through the maxSize of the array, acting like a 2D array and turning it into a 1D array for the adjacency matrix, and then set everything to Infinite weight.

Parameters

int	maxNumber, which is the largest possible size for the WeightedGraph.

Returns

This function does not return anything.

Precondition

none

Postcondition

The WeightedGraph is empty and was just created.

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

This function is the copy constructor for the WeightedGraph class.

The function will set the maxSize equal to the other WeightedGraph's maxSize. The function will set the size equal to the other WeightedGraph's size. The function will dynamically allocate memory for the vertexList to be equal to the maxSize. The function will dynamically allocate memory for the adjMatrix and pathMatrix equal to the maxSize ∗ maxSize. The function will then loop through the maxSize of the array, acting like a 2D array and turning it into a 1D array for the adjacency matrix, and then set copy the data from the other arrays to this array.

Parameters

const	WeightedGraph & otherm which is the other object to be copied to make this object.

Returns

This function does not return anything.

Precondition

none

Postcondition

The WeightedGraph is an exact copy of the other object that was passed in.

4.3.1.3 WeightedGraph::∼WeightedGraph ( )

This function is the destructor for the WeightedGraph class.

This function will call the clear function to clear to set the size to 0. This function will call the delete operator on pathMatrix, adjMatrix and vertexList. This function will set pathMatrix, adjMatrix and vertexList to NULL.

Parameters

none	

Returns

This function does not return anything.

Precondition

none

Postcondition

Size will be set to 0 and pathMatrix, adjMatrix and vertexList will be cleared and deleted and set to NULL.

4.3.2 Member Function Documentation

4.3.2.1 bool WeightedGraph::areAllEven ( ) const

This function is the areAllEven function for the WeightedGraph class

The function checks to see if every vertex in a graph is of even degree, meaning that the amount of vertices the vertex is connected to is an even number. The function does this by looping through an checks to see if there exists an edge between a vertix and another. If it does, it increments edges by one. After one vertex is done, it checks to see if the the number of edges of that vertex is not divisible by 2, if it isn't the function sets to_return to false. Otherwise, the function resets Edges to 0 and increments i and moves to the next vertex. The function returns to_return at the end.

Parameters

none	

Returns

If all of the vertices on the graph is even, the function returns true, else it returns false.

Precondition

none

Postcondition

If all of the vertices on the graph is even, the function returns true, else it returns false.

4.3.2.2 void WeightedGraph::clear ( )

This function is the clear function for the WightedGraph class.

This function sets the size of the WeightedGraph to be 0.

Parameters

none	

Returns

This function does not return anything

Precondition

none

Postcondition

This function will clear the WeightedGraph by setting size is 0.

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

This function is the getEdge for the WeightedGraph class

The function makes sure that row and size are within the maxSize of the array. it it is, the function returns the edge associated with the row and col of the adjacency matrix.

Parameters

int	row, which is the row of the edge to look for
int	col, which is the col of the edge to look for

Returns

returns the edge of the given row and col from the adjacency matrix, if it is within bounds. Otherwise it returns -10.

Precondition

none

Postcondition

returns the edge of the given row and col from the adjacency matrix, if it is within bounds. Otherwise it returns -10.

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

This function is the getEdgeWeight function for the WeightedGraph class.

The function first checks creates local variables to hold the 2 indexs to insert the edge for and sets them to -10 (invalid number) and a bool variable to return. The function then gets the 2 indexes by calling getIndex with string v1 and string v2 passed as parameters. If both of the indexes are not between 0 and (size $*$ size) the function throws a logic error saying vertex not found or bound error. Other wise, the function throws a logic error saying the vertex was not found or bound error. The function checks to see that the edge weight is not infinite, to make sure that an edge exists. If an edge exists, the weight of to the edge is set to wt and to_return is set to true.

Parameters

const	string& v1, which is the string to look for one of the vertex.
const	string& v2, which is the string to look for the second first.
int	wt, which is the variable to hold the weight from the edge between the two vertices.

Returns

This function returns return to_return based on if the vertices if found and the edge weight exists.

Precondition

Valid vertices containing v1 and v2.

Postcondition

The function will return to_return based on if the vertices is found and the edge weight exists, and then set wt equal to the weight of the ede.

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

This function is the getIndexfunction for the WeightedGraph class

The function goes through the size of the vertexList and checks to see if a vertex's label is equal to the string v. If it does, the function returns the index of the Vertex. If it doesn't the function returns -10, to show that the function could not find the vertex.

Parameters

const	string& v which is the string label to look for in a vertex.

Returns

returns the index of the vertex if it is found, or it returns -10 if the vertex is not found.

Precondition

none

Postcondition

returns the index of the vertex if it is found, or it returns -10 if the vertex is not found.

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

This function is the getPath for the WeightedGraph class

The function makes sure that row and size are within the maxSize of the array. it it is, the function returns the edge associated with the row and col of the pathMatrix.

Parameters

int	row, which is the row of the path to look for
int	col, which is the col of the path to look for

Returns

returns the path of the given row and col from the path matrix, if it is within bounds. Otherwise it returns -10.

Precondition

none

Postcondition

returns the path of the given row and col from the path matrix, if it is within bounds. Otherwise it returns -10.

4.3.2.7 bool WeightedGraph::hasProperColoring ( ) const

This function is the hasProperColoring function for the WeightedGraph class

The function checks to see if the vertice's colors in the graph are different from adjacent vertices. The function does this by checking the vertex next to it by making sure that the vertex in front of it and connected via an edge are not equal in color.

Parameters

none	

Returns

If the vertice's colors in the graph are different from adjacenct vertices, it returns, else it returns false.

Precondition

All verties have been assigned a color

Postcondition

If the vertice's colors in the graph are different from adjacenct vertices, it returns, else it returns false.

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

This function is the insertEdge function for the WeightedGraph class.

The function first checks creates local variables to hold the 2 indexs to insert the edge for and sets them to -10 (invalid number) The function then gets the 2 indexes by calling getIndex with string v1 and string v2 passed as parameters. If both of the indexes are between 0 and (size $*$ size) the function sets the edge by calling setEdge with V1_Index, V2_Index and the weight passed as the parameters. Other wise, the function throws a logic error saying the vertex was not found or bound error.

Parameters

const	string& v1, which is the string to look for one of the vertex.
const	string& v2, which is the string to look for the second first.
int	wt, which is the weight to insert for the edge.

Returns

This function does not return anything.

Precondition

Valid vertices containing v1 and v2.

Postcondition

The function will insert the edge with the given weight for the the two vertices.

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

This function is the insertVertex function for the WeightedGraph class.

The function first checks to see if the vertex already exists in the graph. If it does, the function will set is equal to the new Vertex and sets the edges with infinite weight and updates them. Otherwise, if the object is full, the function will throw a logic_error saying Graph Full. If not, the function will increment size and insert the newVertex and set the edges with infinite weight.

Parameters

const	Vertex& newVertex, which is the newVertex that will be either inserted or updated.

Returns

This function does not return anything.

Precondition

none

Postcondition

newVertex will be either be inserted into the graph, or the graph will be updated.

4.3.2.10 bool WeightedGraph::isEmpty ( ) const

This function is the isEmpty function for the WeightedGraph class.

This function checks to see if size is equal to 0. If it is, the function returns true. Otherwise the function returns false.

Parameters

none	

Returns

This function returns if the WeightedGraph is empty or not.

Precondition

none

Postcondition

This function will return if the WeightedGraph is empty or not.

4.3.2.11 bool WeightedGraph::isFull ( ) const

This function is the isFull function for the WeightedGraph class.

This function checks to see if size is equal to the maxSize. If it is, the function returns true. Otherwise the function returns false.

Parameters

none	

Returns

This function returns if the WeightedGraph is full or not.

Precondition

none

Postcondition

This function will return if the WeightedGraph is full or not.

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

This function is the overloaded assignment operator for the WeightedGraph class.

The function will first check to make sure that the other object is not the same as this object. The function will then set the maxSize equal to the other WeightedGraph's maxSize. The function will set the size equal to the other WeightedGraph's size. The function will dynamically allocate memory for the vertexList to be equal to the maxSize. The function will dynamically allocate memory for the adjMatrix and pathMatrix equal to the maxSize $*$ maxSize. The function will then loop through the maxSize of the array, acting like a 2D array and turning it into a 1D array for the adjacency matrix, and then set copy the data from the other arrays to this array.

Parameters

const	WeightedGraph & otherm which is the other object to be copied to make this object.

Returns

This function returns a pointer to this object.

Precondition

none

Postcondition

The WeightedGraph is an exact copy of the other object that was passed in.

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

This function is the removeEdge function for the WeightedGraph class.

The function first checks creates local variables to hold the 2 indexs to insert the edge for and sets them to -10 (invalid number. The function then gets the 2 indexes by calling getIndex with string v1 and string v2 passed as parameters. If both of the indexes are between 0 and (size $*$ size) the function sets the edge by calling setEdge with V1_Index, V2_Index and Infinite passed as the parameters. Other wise, the function throws a logic error saying the vertex was not found or bound error.

Parameters

const	string& v1, which is the string to look for one of the vertex.
const	string& v2, which is the string to look for the second first.

Returns

This function does not return anything.

Precondition

Valid vertices containing v1 and v2.

Postcondition

The function will remove the edge connecting between two vertices if they are found.

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

This function is the removeVertex function for the WeightedGraph class.

The function looks to see if the label of a vertex is equal to the string v. If it is found, the function fixes the adjacency matrix by moving everything to the left by 1. The function then fixes the adjacency matrix by by moving everything up by 1. The function has to fix by left and up because it is imagined as a 2d array, but it is represented as a 1d array. The funtion then moves everything from the vertexList to the left by 1 from the point it is at to the size and then decrements the size, which will then force the function to return. If none of that occurs, the function throws a logic error saying that either the vertex was not found or there was a bound error.

Parameters

const	string& v1, which is the string to look for the vertex label.

Returns

This function does not return anything.

Precondition

A valid vertex in the graph.

Postcondition

The function removs the vertex from the list and removes edges connected to it and fixes the adjacency matrix.

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

This function is the retrieveVertex function for the WeightedGraph class.

The function first declares a local bool variable called to_return and sets it as false. The function then iterates through the entire size of the vertexList and continues until it finds the label that is equal to the string v. The function sets vData to be equal to the vertex that was found to be equal to the label of string v. The function returns the variable to_return;

Parameters

const	string& v, which is the label to look for in the vertices.
Vertex&	vData, which holds the value of the vertex's data that was found.

Returns

This function returns the variable to_return indicating true of false if the vertex was found and retrieved.

Precondition

none

Postcondition

This function returns the variable to_return indicating true of false if the vertex was found and retrieved and then sets vData equal to the vertex that was found.

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

This function is the setEdge for the WeightedGraph class

The function makes sure that row and size are within the maxSize of the array. it it is, the function then sets the edge for the row and col. Sine we have a 1d array mapped as a 2d array, we have to set the edge weight for both index in the adjacency Matrix.

Parameters

int	row, which is the row of the edge to set for
int	col, which is the col of the edge to set for

Returns

This function does not return anything.

Precondition

none

Postcondition

sets the edge of a given row and col by setting the edge weight for the index in the adjacency Matrix.

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

This function is the setPath for the WeightedGraph class

The function makes sure that row and size are within the maxSize of the array. it it is, the function then sets the path for the row and col. Sine we have a 1d array mapped as a 2d array, we have to set the path for both index in the Path Matrix.

Parameters

int	row, which is the row of the path to set for
int	col, which is the col of the path to set for

Returns

This function does not return anything.

Precondition

none

Postcondition

sets the path of a given row and col by setting the path for the index in the adjacency Matrix.

4.3.2.18 void WeightedGraph::showShortestPaths ( )

This function is the showShortestPath function for the WeightedGraph class

We use Floyd's algorithm to make sure that a path from vertex k to vertex j and there exists a path from vertex j to vertex l and the sum of entries (k,j) abd (j,l) is less than entry (k,l) in the path matrix. Then we replace entry (k,l)with the sum of entries (k,j) and (j,l) We make sure that the paths exist at first, and then we check to see that Floyd's algorhtm is true, and if it is, we set the Path to be the smallest path.

Parameters

none	

Returns

This function does not return anything.

Precondition

none

Postcondition

The Path Matrix is printed just like shown above in the showStructure function for the Edge and Matrix, as that one was given and i modeled it to match that.

4.3.2.19 void WeightedGraph::showStructure ( ) const

This function is the showStructure function for the WeightedGraph class

Outputs the vertexList and adjacency Matrix of the WeightedGraph will be printed. If the size is 0, the function will just print out empty graph.

Parameters

none	

Returns

This function does not return anything.

Precondition

none

Postcondition

The vertexList and adjacency Matrix of the WeightedGraph will be printed.

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

- WeightedGraph.h
- WeightedGraph2.h
- show12.cpp
- WeightedGraph.cpp

4.4 WtGraph Class Reference

Public Member Functions

- **WtGraph** (int maxNumber=defMaxGraphSize) throw (bad_alloc)
- void **insertVertex** (Vertex newVertex) throw (logic_error)
- void **insertEdge** (char *v1, char *v2, int wt) throw (logic_error)
- bool **retrieveVertex** (char *v, Vertex &vData) const
- bool **edgeWeight** (char *v1, char *v2, int &wt) const throw (logic_error)
- bool **getEdgeWeight** (char *v1, char *v2, int &wt) const throw (logic_error)
- void **removeVertex** (char *v) throw (logic_error)
- void **removeEdge** (char *v1, char *v2) throw (logic_error)
- void **clear** ()
- bool **isEmpty** () const
- bool **isFull** () const
- bool **hasProperColoring** () const
- void **showStructure** () const

Private Member Functions

- int **index** (char *v) const
- int **getEdge** (int row, int col) const
- void **setEdge** (int row, int col, int wt)

Private Attributes

- int **maxSize**
- int **size**
- Vertex * **vertexList**
- int * **adjMatrix**

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

- WeightedGraph3.h
- WeightedGraph.cs

5 File Documentation

5.1 WeightedGraph.cpp File Reference

This program will create an implementation of the Weighted Graph ADT using a vertex list and adjacency matrix.

```
#include "WeightedGraph.h"
```

5.1.1 Detailed Description

This program will create an implementation of the Weighted Graph ADT using a vertex list and adjacency matrix.

Author

Kripash Shrestha

Version

1.2

The specifications of the program are instructed and documented on Lab 12 C++ Data Structures: A Laboratory Course Third Edition by Brandle, Geisler, Roberge and Whittington

Date

Wednesday, November 27, 2017

Index