Graph Implementation

1.0

Generated by Doxygen 1.8.8

Fri Nov 21 2014 08:53:46

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

Class Documentation

2.1 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

2.2 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

2.3 WeightedGraph Class Reference

Classes

- class Vertex

Public Member Functions

- WeightedGraph (int maxNumber=MAX_GRAPH_SIZE)

  Default Constructor, This will set the max number given by user or use the default of 10. Then allocates all memory needed for the arrays.
- WeightedGraph (const WeightedGraph &other)

  Copy Constructor - Creates a new graph with the exact contents of the one in the parameter.
- WeightedGraph & operator= (const WeightedGraph &other)

  Overloaded operator.
- ~WeightedGraph ()

  Destructor, deallocates all memory.
- void insertVertex (const Vertex &newVertex) throw (logic_error)

  Inserts a new vertex into the list.
- void insertEdge (const string &v1, const string &v2, int wt) throw (logic_error)

  Inserts new edge between two given verticies. Will find where to insert then insert in that spot in the matrix. If there is already an edge, it updates it.
- bool retrieveVertex (const string &v, Vertex &vData) const

  Looks for an input given by the user. If found saves the vertex in the parameter.
- bool getEdgeWeight (const string &v1, const string &v2, int &wt) const throw (logic_error)

  Finds the weight between two verticies then returns their weight by updating the parameter. Gets the two indicies and then gets their edge.
- void removeVertex (const string &v) throw (logic_error)

  Removes a vertex by furst deleting the whole correcsponding row and moving everything up. the does virtually the same with the colums.
- void removeEdge (const string &v1, const string &v2) throw (logic_error)

  This will just go to the associated edges from the parameters and set them to their initialized values.
- void clear ()

  Common Sense.
- bool isEmpty () const

  Checks if empty...
- bool isFull () const

  Checks if full...
- void **showStructure** () const
- void showShortestPaths ()

  Finds the shortest paths from every possible connection. I do not understand 100% how the algorithm works. I just took what was in the book and put it into code.
- bool hasProperColoring () const

  This checks that any two connected verticies dont have the same color. This is done by comparing each vertex to its connections through a loop.
- bool areAllEven () const

  This function checks if any given edge could be deleted and keep the graph completely intact. This checks if each vertex has an even number of connections.
- **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 **MAX_GRAPH_SIZE** = 10
- 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

  Finds where the vertex in the parameter is in the vertex list.
- int getEdge (int row, int col) const

  Finds a spot in the Edge array with the given coordinates.
- void setEdge (int row, int col, int wt)

  Sets a spot in the Edge array with the given info.
- int getPath (int row, int col) const

  Finds a spot in the Path array with the given coordinates.
- void setPath (int row, int col, int wt)

  Sets a spot in the Path array with the given info.
- 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**

2.3.1 Constructor & Destructor Documentation

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

Default Constructor, This will set the max number given by user or use the default of 10. Then allocates all memory needed for the arrays.

Postcondition

Everything will be initialized.

Parameters

The	maxnumber of the graph. Will be default 10.

Returns

Constructor.

set max and initialize size

allocate memory for graph

initialize "matrix"

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

Copy Constructor - Creates a new graph with the exact contents of the one in the parameter.

Precondition

Nothing

Postcondition

Exact copy of other graph.

Parameters

Graph	to copy from.

Returns

constructor.

copy sizes

iterate through both arrays and copy after allocating memory

2.3.1.3 WeightedGraph::~WeightedGraph ( )

Destructor, deallocates all memory.

Precondition

initialized graph

Postcondition

all memory will be deallocated

Parameters

none	

Returns

destructor

deallocate

2.3.2 Member Function Documentation

2.3.2.1 bool WeightedGraph::areAllEven ( ) const

This function checks if any given edge could be deleted and keep the graph completely intact. This checks if each vertex has an even number of connections.

Precondition

Initialized graph.

Postcondition

Same.

Parameters

None.	None.

Returns

True if all verticies has even numbers of connections false otherwise.

check if empty

loop through list of verticies

set index for corresponding matricies

make counter to check degree of vertex

prime loop and go through corresponding edges for current vertex

if the edge is not infinite increment

2.3.2.2 void WeightedGraph::clear ( )

Common Sense.

overwrite "matrix"

clear all values

reset size

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

Finds a spot in the Edge array with the given coordinates.

Returns

the index of the spot.

return length

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

Finds the weight between two verticies then returns their weight by updating the parameter. Gets the two indicies and then gets their edge.

Precondition

Initialized.

Postcondition

Nothing changes.

Parameters

The	two strings to search for a edge for and where to save the weight. If the indicies dont exist, if the indicies are the same, or the edge is not existant

Returns

true if found and false if not

check for empty

get indicies for boh strings

check if valid verticies

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

Finds where the vertex in the parameter is in the vertex list.

Returns

the index of the spot.

look for item in list

if equal return index

fail

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

Finds a spot in the Path array with the given coordinates.

Returns

the index of the spot.

return length

2.3.2.7 bool WeightedGraph::hasProperColoring ( ) const

This checks that any two connected verticies dont have the same color. This is done by comparing each vertex to its connections through a loop.

Precondition

Initialized.

Postcondition

Same. (Nothing changes)

Parameters

None.	None.

Returns

True if the coloring is ok and false if not.

check if empty

initialize color variable for comparison

loop through list of verticies

get color of first vertex

set index for corresponding matricies

prime loop and go through corresponding edges for current vertex

compare, if same colors return false

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

Inserts new edge between two given verticies. Will find where to insert then insert in that spot in the matrix. If there is already an edge, it updates it.

Precondition

initialized matrix

Postcondition

edge will be added in both correcsponding parts of the matrix.

Parameters

Two	strings which identify what the edges connect and an int for the edge's weight. If one of the strings do not exist, if the strings are the same.

Returns

void

find where to insert edge

check for errors, 1) doesnt exist 2) same place

print

update adjMatrix

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

Inserts a new vertex into the list.

Precondition

Initialized list.

Postcondition

New item in the list.

Parameters

The	new vertex to insert into the list. If the list is full cant add more.

Returns

void.

check for full

insert new vertex into list

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

Overloaded operator.

Precondition

An initialized graph to copy from.

Postcondition

Two equal graphs.

Parameters

Graph	to copy from.

Returns

The copy graph.

if same return

deallocate memory

copy sizes

allocate new memory of correct size and copy from otehr

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

This will just go to the associated edges from the parameters and set them to their initialized values.

Precondition

Edge with value

Postcondition

Edge will be "empty"

Parameters

the	two verticies to look for if the verticies are not valid

Returns

void

find locations

check for valid edges

replace current weight with infinite

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

Removes a vertex by furst deleting the whole correcsponding row and moving everything up. the does virtually the same with the colums.

Precondition

initialized graph

Postcondition

given vertex will be removed

Parameters

the	vertex identifer to delete if the graph is empty

Returns

void

check for empty

get index

remove from list

move up rows

move columns left

initialize left over spots to infinite edge wt

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

Looks for an input given by the user. If found saves the vertex in the parameter.

Precondition

Initialized vertex list.

Postcondition

Same

Parameters

What	to look for (string) and where to save if it is found (vertex) If if isnt found.

Returns

True if found, false if not.

get index of string

if the index is not found, return false

set other parameter to corresponding vertex and return

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

Sets a spot in the Edge array with the given info.

Returns

void.

set egdge if valid

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

Sets a spot in the Path array with the given info.

Returns

void.

set egdge if valid

2.3.2.16 void WeightedGraph::showShortestPaths ( )

Finds the shortest paths from every possible connection. I do not understand 100% how the algorithm works. I just took what was in the book and put it into code.

Precondition

Initialized.

Postcondition

A Path matrix will be created and printed.

Parameters

None	None

Returns

Void.

copy edge matrix

FLOYD

print the matrix

print actual data and - for infinite

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

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

2.4 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