PA08 1-3 Linked Implementation

Generated by Doxygen 1.8.6

Mon Dec 19 2016 22:37:01

Contents

1 Class Index

1.1 Class List

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

2 File Index

2.1 File List

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

3 Class Documentation

3.1 graphMatrix Class Reference

Public Member Functions

- graphMatrix (int, bool)
- ~graphMatrix ()
- int getNumVertices () const
- int getNumEdges () const
- int getEdgeWeight (int start, int end) const
- bool add (int start, int end, int edgeWeight=0)
- bool remove (int start, int end)
- void DFS (int start, void visit(int &))
- void BFS (int start, void visit(int &))
- void clearVisited ()

Private Attributes

- int **size**
- int **numEdges** = 0
- bool **isDirected**
- vector< list< Node > > **data**
- deque< int > **visited**
- deque< int > **BFSQ**

3.1.1 Constructor & Destructor Documentation

3.1.1.1 graphMatrix::graphMatrix (int *num*, bool *direc*)

Fully parameterized constructor for the graph.

Parameters

num	The number of possible elements (vertices) in the graph
direc	Whether the graph is directed or not

Postcondition

The graph is created by filling the vector with the specified number of lists

3.1.1.2 graphMatrix::~graphMatrix ( )

Deconstructor for the graph class.

Precondition

Graph object has dynamically allocated data

Postcondition

All dynamically allocated data in the object is freed

3.1.2 Member Function Documentation

3.1.2.1 bool graphMatrix::add (int *start,* int *end,* int *edgeWeight =* 0)

Function to add an edge between two vertices. If an edge is attempted to be made from a vertex to itself, it immediately fails. If an edge already exists, it will not be created and will fail. If the edge was successfully created, the function will check whether or not the graph is directed, and if not create an edge from the "end" vertex to the "start" vertex.

Precondition

Two vertices are given, regardless of existing edges

Postcondition

An edege is created between two unique vertices, if one doesn't exist already

Parameters

start	The vertex the edge should start at
end	The vertex the edge should end at
edgeWeight	The weight of the to-be edge

Returns

Whether or not a new edge was created

3.1.2.2 void graphMatrix::BFS (int *start,* void *visitint &*)

Function to traverse through the graph using a breadth first approach. The number being visited is pushed to the front of a deque outside of the function, because it can also be used by the recursive depth first approach. The number is then sent to another deque, BFSQ which is the deque of numbers to be visited next. The number is then sent to the visit function, to do various activities such as outputting the data. All the adjacent vertices are then placed into a temporary deque, sorted, then dumped into the BFSQ deque, which is outside the while function. If the number doesn't match with the current number being visited, has an edge, and is not in the visited deque, it is added to the visited deque and BFSQ deque, before being popped from the tmp deque.

Precondition

The "visited" and "BFSQ" deque should have been emptied beforehand

Postcondition

The graph has been traversed in a breadth first approach

Parameters

start	The vertex to start the breadth first traversal from
visit	The function to do stuff like outputting data values

3.1.2.3 void graphMatrix::clearVisited ( )

Function to clear the deques that may have been used before hand. These deques are important because they help keep track of whats been visited, thus needing to be emptied before being used again in another traversal.

Precondition

The "visited" and "BFSQ" deques may not be emptied

Postcondition

The two deques have been emptied

3.1.2.4 void graphMatrix::DFS (int *start,* void *visitint &*)

Function to traverse through the graph using a depth first approach. The number being visited is pushed to the front of a deque outside of the function, because the function is recursive. The number being visited is then sent to the visit function, which can do various things such as outputting the data value. For each vertex adjacent to the currently visited function will then be visited using this same function, which effectively visits all the connected vertices depth first. All the adjacent vertices will be visited in order numerically, by sending all the adjacent nodes to a list and sorting, before dumping them back into the recursive function.

Precondition

The "visited" deque should have been emptied beforehand

Postcondition

The "visited" deque should not be empty, and the graph has been traversed in a depth first approach.

Parameters

start	The vertex to start the iteration of DFS from
visit	The function to do stuff like outputting the data value

3.1.2.5 int graphMatrix::getEdgeWeight (int *start,* int *end*) const

Function to find the weight of the edge between two vertices

Precondition

Two vertices are given, and the edge between them unknown

Postcondition

The weight of the edge between two given vertices is known

Parameters

start	The starting vertex of the edge to be weighed
end	The ending vertex of the edge to be weighed

Returns

The value of the data in the 2d graph at the specified coordinates

3.1.2.6 int graphMatrix::getNumEdges ( ) const

Function to find the number of edges in the graph

Precondition

The number of vertices is unknown

Postcondition

The number of vertices is known

Returns

The integer representing the number of edges in the graph

3.1.2.7 int graphMatrix::getNumVertices ( ) const

Function to find the number of vertices in the graph. If the graph is not directed, it is relatively straightforward. However, if it is not, a temporary deque is created to store the unique, connected vertice numbers.

Precondition

number of vertices is unknown

Postcondition

The number of vertices is known @ return The number of vertices in the current graph

3.1.2.8 bool graphMatrix::remove (int *start,* int *end*)

Function to remove an edge between two vertices. If the two vertices are indentical, the function will fail instantly. If the edge does not exist, the function will also fail. If the graph is not directed, the reverse coordinate will also be removed. The function then checks if anything is connected to the two vertices, and removes them from the list of connected and unique vertices if they are not.

Precondition

Two vertices are given, regardless of existing edges

Postcondition

An edge is removed if one exists

Parameters

start	The vertex the edge starts from
end	The vertex the edge ends at

Returns

Returns whether or not an existing edge was removed

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

- graphMatrix.h
- graphMatrix.cpp

3.2 Node Struct Reference

Public Attributes

- int **pointTo**
- int **weight**

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

- node.h

4 File Documentation

4.1 graphMatrix.cpp File Reference

Implementation file for the graph.

```
#include "graphMatrix.h"
#include <algorithm>
```

Macros

- #define **INVALID_NUM** 2147483647

4.1.1 Detailed Description

Implementation file for the graph. I was not sure how to implement the infinity in the case that two vertices were not connected, so I set it to the maximum an integer could be on a 64-bit machine.

Version

1.00 Wei Tong (19 December 2016) Turned in version and initial development

4.2 graphMatrix.h File Reference

Header file for graph.

```
#include <iostream>
#include <vector>
#include <list>
#include <deque>
#include "node.h"
```

Classes

- class graphMatrix

4.2.1 Detailed Description

Header file for graph. The header file includes the using namespace standard because it was required for parts of the implementation file as well as the main, so it might as well have been put here. Vector, list and deque were also used because they had various properties that I could make use of. For example, vector is like an array, but can be of stuff other than integers or characters, so I used it to be an array of lists. std::list is a linked list that was pre-made, so I used it thinking it would be beneficial to be able to switch from nodes to another data type, without having to rewrite the entire program (for future usage)

Version

1.00 Wei Tong (19 December 2016) Turned in version and initial development

4.3 main.cpp File Reference

Test driver for graph matrix linked list implementation.

```
#include "graphMatrix.h"
#include <fstream>
```

Macros

- #define **isDirected** false

Functions

- void visit (int &data)
- int **main** ()

Variables

- ofstream **fout**

4.3.1 Detailed Description

Test driver for graph matrix linked list implementation. Main file for project. Will create a matrix, which is actually a vector of lists of nodes. This driver tests both directed and undirected, and can be switched by changing isDirected to either true or false. This also tests the number of vertices on a completed graph, and then removes the last connection to show that the number of vertices didn't change, but the number of edges did. Both depth first and breadth first search are performed, and the numbers are output to file as they are visited. Whether the graph is weighted or not depends on input when adding. If the third parameter is detected, it will set that as the weight. Otherwise, it will default to 0. Problems 1-3 of the optional project have been completed (pg. 663 of the textbook)

Version

1.00 Wei Tong (19 December 2016) Turned in version and initial development

Note

Adapted from Frank M. Carrano and Timothy M. Henry Copyright (c) 2012 Pearson Education, Hoboken, New Jersey.

4.3.2 Function Documentation

4.3.2.1 void visit (int & *data*)

Outputs what number was visited as the graph is traversed.

Parameters

data	An integer variable that will receive the data to be output

Precondition

Node in the grpah has been traversed to

Postcondition

Data in the node has been outputted

Returns

void

4.4 node.h File Reference

Header file for a simple node.

Classes

- struct Node

4.4.1 Detailed Description

Header file for a simple node. This class is extremely simple, and doesn't contain pointers and such because I had only needed a container that could container a data value and the next number the current number pointed to

Version

1.00 Wei Tong (19 December 2016) Turned in version and initial development

Index