Algorithms

Fall 2018 **Graphs Part 1**

Dr. Gurka preliminary development

August 23

Overview

This project focuses on developing a graphs package that you can use in later projects. Develop it thinking of yourself as the future user, in the sense that you will be writing programs that need to use graphs, and will treat this code as a library resource.

Due September 8, 11:55 pm on Moodle, no paper copies, include a cover letter (template and discussion questions to be posted). Completion goal is one week or so. Submission is your claim that all test cases passed; do not hand in any output. Part of the grading will be a walkthrough with me on one of the graph implementations (adjacency matrix or adjacency list), whichever you finish first. Make an appointment for the walkthrough before the due date

Use your prior experience developing projects to have a well-organized approach for design, development, and test. Keep track of your process as you go, including time spent. Structure this information as an informal log to be turned in with the final project.

You may work together, but ensure that you fully understand each component of your system.

----

The first use of the package (a different assignment) will be to run performance analysis experiments comparing the different implementations. Later you’ll use the packages as the basis for various graph problems.

Everyone will create test plans, to be shared via a Moodle forum. Run your final project against all test cases, using both implementations (adjacency matrix or adjacency list). Leave a lot of time for testing

-----

There will be a Moodle forum for specifications questions, closing Sunday night. At that time I will post final specifications if there are any changes.

-----

Graph functions to implement:

* hasEdge, addEdge, deleteEdge
* addVertex, deleteVertex
* isSparse (use 15%), isDense (use 85%)
* countVertices, countEdges
* isConnected, isFullyConnected
* readGraph, printGraph (both text-based, format to be decided in class)

-----

Additional specifications.

* data file format will be correct, no error-checking needed
* graphs in the data file will be valid
* hasEdge asks whether a specific edge exists in the graph, not whether the entire graph has any edges
* weights, if any, will be positive floats
* vertex names will be a single uppercase letter (note the resulting limit of 26 vertices in a graph)
* adding a duplicate vertex has no effect on the graph (i.e., nothing is added) and returns false
* test files should include your name and a brief description of the case being tested (type of graph, order of operations, etc.) – what is the design of this specific test case?
* name your test file <your first name><k>.txt (Judy3.txt) for easy swapping
* edges will not be repeated in the graph edge list
* addEdge can overwrite an existing edge (applicable only to weighted graphs)
* an empty graph and a graph composed of a single node are both considered connected

-----

Required test file format (they will be exchanged).

<overall header comments, preceded by '\*' indicator>

<weighted or unweighted> (actual keywords)

<directed or undirected> (actual keywords)

begin

<list of vertices>

<edge 1>

<edge 2>

:

<edge n>

end

<function 1>

<correct result 1>

<function 2>

<correct result 2>

:

<function m>

<correct result m>

-----

Sample test file.

\* Graphs test file by J. Gurka

\* File: Judy7.txt

\* This file tests adding and deleting edges in a sparse graph, including

\* duplicate edges being added and non-edges being deleted.

weighted

undirected

begin

A F B C D

A B 2.3

C F 1.0

A D 6.2

end

hasEdge B Q

false

isSparse

false

deleteEdge C F

true (success)

-----

Notes

* ‘\*’ as the first character in the line means a comment
  + several lines of comments at the beginning will describe the graph being used and the test case – what, specifically, is being tested on what specific kind of graph?
  + comments can also occur between test cases for specific info on one test case
* file format is perfect (no error checking)
* each test file contains one graph and the operations on it (designed by the tester)
* edges listed in the file are in no particular order
* if the graph is directed, the edge goes *from* the first vertex listed *to* the second
* an empty graph has no lines between “begin” and “end”
* all text except comments and vertex names will be lowercase, except camel case for functions (see list)
* output: the program should echo the original graph, in the same form (checks correct input and documents output for easy checking)
* the number of edges is unknown
* edge data and vertex data are space separated
  + example: <vertex 1><space><vertex 2><space><weight>
* the graph is not a multi-graph (2 or more edges between 2 vertices)

-----

Function names, to be spelled exactly as shown (including case) for inclusion in the data file:

* hasEdge
* addEdge
* deleteEdge
* addVertex
* deleteVertex
* isSparse
* isDense
* countVertices
* countEdges
* isConnected
* isFullyConnected
* readGraph
* printGraph

-----