# ICSI213 – Assignment 5 – Graphs – The Last One!

**You must submit .java files. Any other file type will be ignored. Especially “.class” files.**

**You must not zip or otherwise compress your assignment. Blackboard will allow you to submit multiple files.**

***You must submit buildable .java files for credit.***

## Introduction

Graphs are used in many different applications. One of the most important algorithms in graphs is Dijkstra’s Algorithm - finding the shortest path from a start node to every other node. This is done by exploring – keeping track of the shortest distance we have seen (so far) to each node. As we explore (that is, visit unseen nodes), we look at all of the places we can get to and see if this path is shorter than the best one we have seen so far.

I am giving you an implementation of Dijkstra’s Algorithm. I am also giving you a simple graph, vertex and edge interface. From this you will need to implement two different graph designs – one using objects (AdjacencyListGraph) and one using a 2-D array of integer (AdjacencyMatrix).

Before you start, take a look at the Wikipedia page for Dijkstra’s algorithm ( <https://en.wikipedia.org/wiki/Dijkstra's_algorithm> ) and the PowerPoint presentation from class on Graphs.

## Details

An edge in this problem consists only of an integer (distance) and a Vertex (destination). Not having “source” might seem odd, but the only way to find out about an edge is by looking at a Vertex to see its neighbors. Edges have no methods other than accessors for the members.

A vertex has a name (with an accessor) and a method to get its neighbors:

**List<Edge> getNeighbors()** – get all of the edges connected to this vertex.

The graph has two methods:

**List<Verex> getVertices ()** – get a complete list of the vertices in the graph.

**void addEdge(String name1, String name2, int distance)** – adds a vertex with a name of name1 and a vertex with a name of name2 (if they don’t already exist) and makes an edge between them.

### The Implementations:

**AdjacencyListGraph** is the most familiar form of this problem. Make classes that implement each of the three interfaces (Graph, Edge, Vertex). Adding an edge makes two new instances of the edge class – one from name1🡪 name2, the other from name2🡪name1.

This implementation is very straightforward; mine is 70 lines of Java, but only about 18 of them are actual “real” methods – the rest is class and member definitions and constructors.

**AdjecencyMatrix** is the slightly more challenging version of this assignment. In this version, you may **not** make permanent edges.

Your graph will have vertices and a matrix (2D array) of integers. The matrix of integers will indicate the edges (look at the Graphs PowerPoint, but replace the booleans with integers). You will also need an array of Vertex. When addEdge is called, either find each vertex in the array OR make a new one. Note the indices of the vertices. Use those indices to indicate the distance in the matrix (remember to set the distance both from name1 🡪 name2 and name2 🡪 name1).

When “getNeighbors()” is called on a vertex, find the edge indices in the matrix that are >0 and make edges.

For reference, my implementation is 90 lines, but only about 40 are “real” code.

My test file is provided, along with my output.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Rubric | Poor | OK | Good | Great |
| Comments | None/Excessive (0) | “What” not “Why”, few (5) | Some “what” comments or missing some (7) | Anything not obvious has reasoning (10) |
| Variable/Function naming | Single letters everywhere (0) | Lots of abbreviations (5) | Full words most of the time (8) | Full words, descriptive (10) |
| AdjacencyList – Edge Class | None (0) |  | Holds distance, destination, has accessors, members are not private (3) | Holds distance, destination, has accessors, members are private (5) |
| AdjacencyList – Vertex Class | None (0) |  | Holds name, edges, has accessors, members are not private (3) | Holds name, edges, has accessors, members are private (5) |
| AdjacencyList – Graph Class - efficiency | None (0) |  | Inefficiently finds vertices by name (5) | Efficiently finds vertices by name (10) |
| AdjacencyList – Graph Class - addEdge | None (0) |  | Incorrectly adds edges and vertices (5) | Correctly adds edges and vertices (10) |
| AdjacencyMatrix – Edge Class | None (0) |  | Holds distance, destination, has accessors, members are not private (3) | Holds distance, destination, has accessors, members are private (5) |
| AdjacencyMatrix – Vertex Class | None (0) |  | Holds name, has accessor, members are not private (3) | Holds name, has accessor, members are private (5) |
| AdjacencyMatrix – Graph class | None (0) |  | Has 2d matrix for edge data, array of vertex, NOT all fields private (3) | Has 2d matrix for edge data, array of vertex, all fields private (5) |
| AdjacencyMatrix – addEdge | None (0) |  | Repeats vertices or doesn’t populate matrix (10) | Finds existing vertices, populates matrix with distances (20) |
| AdjacencyMatrix – getNeighbors | None (0) |  |  | Dynamically generates edges as needed by vertex call (15) |