CSCI 102 assignment 9 – Other graph implementations

Dec 4

In this assignment you will implement `Graph` in two ways.

- Implement `Graph` with an adjacency map, `AdjacencyMapGraph`. In this implementation, you will still have a list of all vertices; but instead of a list of all edges, edges will be references by the vertices they connect. To do this, `InnerVertex` will have two maps keeping track of its incoming and outgoing edges. The maps should take in an opposite edge and return the connecting edge: `Map<Vertex<V>, Edge<E, V>>`. You may use any map implementation you like but assume `get`, `put`, and other methods are as efficient as our clever implementaiton of hash map (the one where each entry also referenced its position in a list).

  What are the asymptotic computational complexities of the methods in the `Graph` interface?

- Implement `Graph` with an adjacency matrix, `AdjacencyMatrixGraph`. In this implementation, in addition to lists of vertices and entries, keep a list of lists which contains the edge connecting vertex i and j in position (i, j): `Edge<E, V>[][] edgeMatrix`. Remake the entire matrix when adding or removing a vertex. Which methods need to be modified?

  What are the asymptotic computational complexities of the methods in the `Graph` interface?

- In both implementations, write a main method that adds nodes “A”, “B”, “C”, “D”, and 12 edges between the nodes that store the concatenation of the letters, i.e. the edge from “A” to “B” stores “AB” and that from “B” to “A” stores “BA”. Print all the vertices and edges. Then remove “B” and “C” and print all the vertices and edges.

Please submit your code and answers to the questions in a zipped folder on Brightspace by Dec 11.