# CSE 373: Homework 4

Dictionaries and Testing

Due: May 10th, 11:59 PM to Canvas

## Introduction

For this assignment, you will develop a graph representation. You will be using this graph representation for the next homework assignment to implement Dijkstra's algorithm for finding shortest paths.

**Different for this assignment:** You may use anything in the Java standard collections (or anything else in the standard library) for any part of this assignment. Take a look at the Java API as you are thinking about your solutions. At the very least, look at the Collection and List interfaces to see what operations are allowable on them and what classes implement those interfaces. .

## 1 Provided Files

The following files are provided:

- `Graph.java`: Graph interface. *Do not modify*
- `Vertex.java`: Vertex Class: You may add to this if you want
- `Edge.java`: Edge Class: You may add to this if you want
- `MyGraph.java`: Your implementation of the `Graph` interface, you will need to follow the **fill in code** there.

## 2 Functionality

In this assignment, you will implement a graph representation that you will use in the next homework assignment. Add code to the provided-but-incomplete MyGraph class to implement the Graph interface. Do not change the arguments to the constructor of MyGraph and do not add other constructors. Otherwise, you are free to add things to the Vertex, Edge, and MyGraph classes, but please do not remove code already there and do not modify the Graph.java interface. You may also create other classes if you find it helpful.

As always, your code should be correct (implement a graph) and efficient (in particular, good asymptotic complexity for the requested operations), so choose a good graph representation for computing shortest paths next week. You do not know the sparsity or density of the Graph data you might represent with your graph.

**Throw an exception in the following cases:**

- The edges should involve only vertices with labels that are in the vertices of the graph. That is, there should be no edge from or to a vertex labeled A if there is no vertex with label A.
- Edge weights should not be negative.
- Do not throw an exception if the collection of vertices has repeats in it: If two vertices in the collection have the same label, just ignore the second one encountered as redundant information.
- Do throw an exception if the collection of edges has the same directed edge more than once with a different weight. Remember in a directed graph an edge from A to B is not the same as an edge from B to A. Do not throw an exception if an edge appears redundantly with the same weight; just ignore the redundant edge information.

**Other useful information**:

- The Vertex and Edge classes have already defined an appropriate equals method (and, therefore, as we discussed in class, they also define hashCode appropriately). If you need to decide if two Vertex objects are "the same", you probably want to use the equals method and not ==.
- You will likely want some sort of Java Map in your program so you can easily and efficiently look up information stored about some Vertex. (This would be much more efficient than, for example, having a Vertex[] and iterating through it every time you needed to look for a particular Vertex.)
- As you are debugging your program, you may find it useful to print out your data structures. There are toString methods for Edge and Vertex. Remember that things like ArrayLists and Sets can also be printed.

Use these graphs to answer the following questions.

1. Which of the implementations seems to be the best overall? Explain why?

2. Are there any surprising findings? Do all O(n) functions perform at the same speed? Why or why not?

3. Create inputs for the BSTDict to exhibit best case and worst case scenarios. Show the timing difference for the find function for large values of n.

# 3 Writeup

Answer the following questions thoroughly and completely.

1. Describe the worst-case asymptotic running times of your methods adjacentVertices and edgeCost. In your answers, use `|E|` for the number of edges and `|V|` for the number of vertices. *Explain and justify your answers.*

2. Describe how you tested your code. Include code snippets here if you think it would be helpful.

3. Provide pseudo-code that would provide a breadth-first search of the graph where each vertex is processed exactly once. Give an explanation of the tight, asympotic runtime for this function and explain why you feel this is the case

# Deliverables

For this assignment, there will be two submissions on Canvas.

- Part 1 consists of a zip of your Edge, Vertex and MyGraph classes. Only submit those 3 java files.

- Part 2 is the pdf of the writeup. Make sure all questions are answered.