[**Department of Computer Science**](http://www.cs.umd.edu/)

[**CMSC132:**](http://www.cs.umd.edu/class/fall2017/cmsc132) Fall 2017

**Project:** Graphs

**Due Date:** Fri, Dec 8

## Overview

For this project you will implement the Breadth-First Search, Depth-First Search and Dijkstra's algorithms. Your project score will be based on submit server results. For this project you are not required to write student tests, but you are encourage to do so.

## Objectives

* Practice implementation of graphs.
* Implement Breadth-First Search (BFS) Traversal.
* Implement Depth-First Search (DFS) Traversal.
* Implement Dijkstra's algorithm for shortest path computation.

## Grading

* (100%) Tests
  + (35%) Public Tests
  + (65%) Release Tests

## Clarifications

## Code Distribution

The project's code distribution is available by checking out the project named **Graphs**. The code distribution provides you with the following:

* A package named **graphs** → Package where the class you need to implement can be found. Feel free to add any other classes you understand are necessary, but make sure you place them in the graphs package. Notice that you are not required to add any additional classes (i.e., you will NOT lose credit by implementing everything in the Graph class).
* A package named **tests**  → Includes the public tests (PublicTests.java).

## Specifications

This project has two main tasks: Implementing a graph representation and implementing the BFS/DFS/Dijkstra's algorithms. Notice that the Graph class is a generic class whose type parameter represents the elements of the graph we are defining (e.g., Graph<Student>, Graph<Double>, etc.). In order to represent the adjacency properties and the graph data we will use the following maps:

HashMap<String, HashMap<String, Integer>> adjacencyMap;  
HashMap<String, E> dataMap;

The actual methods you need to implement are described at [javadoc](http://docs.google.com/doc/index.html). You will find in the graphs package an interface named **CallBack**. A class implementing this interface represents a class that will process a vertex. An example of such a class is **PrintCallBack**  (which you will find in the graph package). This class is used to generate the string that represents the path we follow when performing a breadth-first search or a depth-first search. Each time we reach a vertex, the implementation of DFS and BFS is expected to call the **processVertex**  method to apply whatever processing needs to be done to the vertex. We have implemented **PrintCallBack** for you (don't modify it).

## Requirements

* The method getCost returns the cost of the directed edge that exist between startVertex and endVertex. You can assume that endVertex is adjacent to startVertex. Notice this method is NOT computing the cost between any two vertices. For this method ignore the @throws IllegalArgumentException information provided in the javadoc.
* If your doDepthFirstSearch and doBreadthFirstSearch methods work in Eclipse, but not in the submit server, your problem might be that you are defining the callback parameter as PrintCallBack instead of Callback. The submit server expects Callback.
* **Do not implement DFS using a recursive approach; implement DFS using an explicit stack as described in lecture.**
* **If no path is found while executing Dijkstra's algorithm, the ArrayList representing the path will have the entry "None". The doDijkstras method will return -1 in this case.**
* Verify that your project passes the submit server tests ([Submit Server](https://submit.cs.umd.edu/)). Those are the results we use to compute your grade.
* See [Student Tests](http://www.cs.umd.edu/class/fall2017/cmsc132/content/resources/StudentTests.html) for information regarding the implementation of student tests for this project.
* See [Style Guidelines](http://www.cs.umd.edu/~nelson/classes/resources/javastyleguide/) for information regarding style.
* Notice you will receive points for release tests even if you don't release test your project (e.g., you ran out of tokens.)
* **Process adjacent vertices in alphabetical order. This means that when processing a node you will add adjacent elements to a stack or queue by selecting adjacent nodes in alphabetical order. It does not mean the DFS or BFS result will show the nodes in alphabetical order. For example, if node B has nodes E and D as adjacents, we will add D to the stack/queue first, followed by E.**

## Academic Integrity

Please make sure you read the academic integrity section of the syllabus so you understand what is permissible in our programming projects. We want to remind you that we check your project against other students' projects and any case of academic dishonesty will be referred to the [Office of Student Conduct](https://osc.umd.edu/).