Koç University
COMP202
Data Structures and Algorithms
Assignment 6

Instructor: Barış Akgün
Responsible TA: None
Due Date: June 10 2018, 23:59
Submission Trough: Blackboard
Relevant Programming Session: None

This programming assignment will test your knowledge and your implementation abilities of what you have learned in the Graph parts of the class.

This homework must be completed individually. Discussion about algorithms and data structures are allowed but group work is not. **Coming up with the same algorithm and talking about the ways of implementation leads to very similar code which is treated as plagiarism!** If you are unsure, you should not discuss. Any academic dishonesty, which includes taking someone else's code or having another person doing the assignment for you will not be tolerated. **By submitting your assignment, you agree to abide by the Koç University codes of conduct.**

Description

This assignment requires you to implement:

- A given graph ADT
- Graph search algorithms
- Graph traversal on an implicit graph (an image)

The files provided to you have comments that will help you with implementation. In addition, keep the slides handy as they include the pseudo-codes for the algorithms. The file descriptions are below. Bold ones are the ones you need to modify and they are placed under the *code* folder, with the rest under *given*. The homework consists of three parts:

Graph Implementation

This part of the homework requires you to implement a given graph ADT for four different types of graphs. There is no explicit `Edge` class or an explicit `Vertex` class. This is done to give you total freedom. If you need extra classes such as these, feel free to put them as nested classes but do not provide extra files.

- *iGraph.java*: The interface that defines the graph ADT. Make sure you pay attention to all the comments!
- ***BaseGraph.java***: The abstract base class for the graphs that you should implement. You can put the common functions here or ignore this file all together.
- ***UndirectedUnweightedGraph.java***: Implement an undirected-unweighted graph in this file.
- ***DirectedUnweightedGraph.java***: Implement a directed-unweighted graph in this file.
- ***UndirectedWeightedGraph.java***: Implement an undirected-weighted graph in this file.
- ***DirectedWeightedGraph.java***: Implement an directed-weighted graph in this file.
- *GraphTesting.java*: The file that includes the autograder implementation this part. Can be run by itself.

Graph Algorithms

This part of the homework requires you to implement depth first search (DFS), breadth first search (BFS), Dijkstra's single-source all-destinations shortest path (or actually smallest cost) algorithm and cycle finding algorithms (for both directed and undirected graphs).

- ***GraphAlgorithms.java***: Implement the algorithms. You can (and probably should) add more methods and fields.
- *AlgTesting.java*: The file that includes the autograder implementation this part. Can be run by itself.

Implicit Graphs: Image Segmentation

This part of the homework requires you to work on an implicit graph formed by the pixels of an image. Think of an image as a 2D array of floating point numbers (doubles in this case). These individual floating point numbers are called pixels. Pixels are indexed by their rows and columns. Pixels values range from 0.0 to 1.0. See Fig. 1

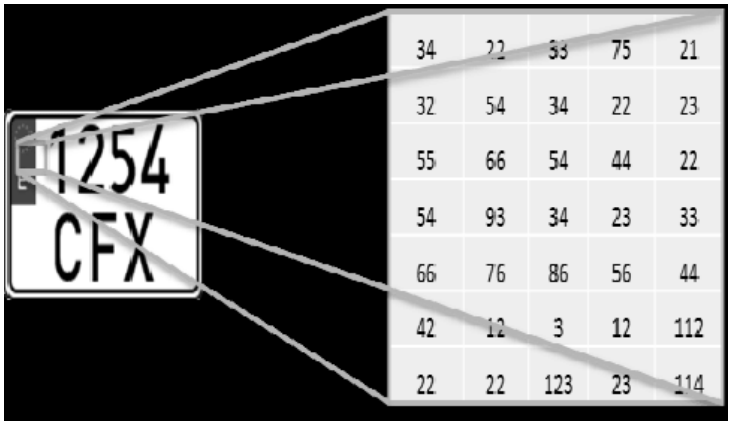

Figure 1: A grayscale image is very similar to a 2D array.

The task is to segment a given image by applying graph algorithms. The goal of image segmentation is to cluster pixels into image regions. In our segmentation approach, we want to group together pixels that have similar values. We can pose this as a problem of finding connected components in a graph where each pixel is a vertex and pixel value similarity decides whether there is an edge between neighboring vertices Each pixel has 4 possible neighbors as shown in Fig.2:

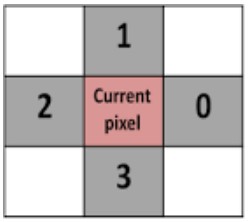

Figure 2: The pixel in the middle(red color) has 4 possible neighbors labeled 0, 1, 2, 3.

Look at the images under the *images/input* and *images/reference* folders to get an idea.

- ***ImageSegmenter.java***: The file where you need to implement the segmentation approach. There is a `dummyIteration` method to help you get around the image.
- *Image.java*: This files defines a wrapper for an image class. You need to go over it to be able to handle this part of the assignment.
- *GradeImagePart.java*: The file that includes the autograder implementation this part. Can be run by itself.

Grading

Your assignment will be graded through an autograder. Make sure to implement the code as instructed, use the same variable and method names. A version of the autograder is released to you. Our version will more or less be similar, potentially including more tests.
Run the main program in the *Grade.java* to get the autograder output and your grade.

Submission

You are going to submit a compressed archive through the blackboard site. The file should extract to a folder with your student ID without the leading zeros. This folder should only contain files that were in **boldface** in the previous section. Other files will be deleted and/or overwritten.

Important: Make sure to download your submission to make sure it is not corrupted and it has your latest code. You are only going to be graded by your blackboard submission.

Submission Instructions

- You are going to submit a compressed archive through the blackboard site. The file can have *zip*, *tar*, *rar*, *tar.gz* or *7z* format.
- This compressed file should extract to a folder with your student identification number with the two leading zeros removed which should have 5 digits. Multiple folders (apart from operating system ones such as MACOSX or DS Store) greatly slows us down and as such will result in penalties
- Code that does not compile will be penalized and may receive no credits.
- Do not trust the way that your operating system extracts your file. They will mostly put the contents inside a folder with the same name as the compressed file. We are going to call a program (based on your file extension) from the command line. The safest way is to put everyting inside a folder with your ID, then compress the folder and give it your ID as its name.
- One advice is after creating the compressed file, move it to your desktop and extract it. Then check if all the above criteria is met.
- Once you are sure about your assignment and the compressed file, submit it through Blackboard.
- After you submit your code, download it and check if it the one you intended to submit.
- **DO NOT SUBMIT CODE THAT DOES NOT TERMINATE OR THAT BLOWS UP THE MEMORY.**

Let us know if you need any help with setting up your compressed file. This is very important. We will put all of your compressed files into a folder and run multiple scripts to extract, cleanup, grade and do plagiarism checking. If you do not follow the above instructions, then scripts might fail. This will lead you to get a 0. Such structured submissions greatly help manual grading as well.