CISC 380 | **Instructor: Dr. Sarah Miracle**

Homework 6

Student: YOUR NAME HERE

Purpose:

- Understand and apply graph algorithms presented in class.
- Design and analyze your own graph algorithms.
- Learn about properties of shortest path trees and spanning trees.

General Homework Policies:

- This homework assignment is due by the start of class on Sunday, May 17 5:00PM. In order to submit your homework,

  1. Submit the `pdf` (it should generated by modifying the LaTeX file for this pdf) and the completed `DirectedGraph.java` and `UndirectedGraph.java` (and extra credit `PrimMST.java`) files through Canvas. Before submitting please zip all the files together and submit one `.zip` file. See the *Additional Programming Instructions* section below for more detailed instructions.

  Late assignments will not be accepted and will receive a 0!

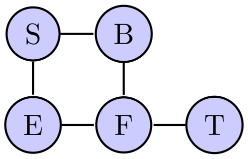
- You may choose to work with a partner on this assignment. If you choose to work with a partner, only one assignment should be submitted and you and your partner will both receive the same grade. Make sure to include your partner's name on the homework.
- If you choose to work with a partner, your assignment should be a true joint effort, equally created, and understood by both partners.
- You are not allowed to consult outside sources, other than the textbook, your notes, the Java API and the references linked from Canvas (i.e., no looking for answers on the internet).[1]
- Getting *ANY* solutions from the web, previous students etc. is *NOT* allowed.[1]
- You are not allowed to discuss this assignment with anyone except for your partner (if you have one) or the instructor.[1]
- Copying code from anywhere or anyone is not allowed (even previous code you have written). Allowing someone to copy your code is also considered cheating.[1]
- Your work will be graded on correctness and clarity. Write complete and precise answers and show all of your work.
- Questions marked (PRACTICE) will not be graded and do not need to be submitted. However it's highly recommended that you complete them.

[1]*See the section of the syllabus on academic dishonesty for more details.*

Homework Problems:

1. (5 points) Write a method to determine if a directed graph is acyclic. Implement your solution by adding code to the `isAcyclic` method in the `DirectedGraph.java` file.

2. (5 points) You are given as input an undirected and unweighted graph $G = (V, E)$, and two vertices $s, t \in V$. Give a *linear time* algorithm which will compute the *number* of distinct shortest paths from s to t. Note that two distinct paths can share multiple vertices in common. For example, in the graph below there are 2 distinct shortest paths $S-B-F-T$ and $S-E-F-T$.



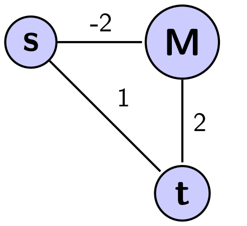
You will implement your solution by adding code to the `numShortestPaths` method within the `UndirectedGraph.java` file.

Note that if your code does not implement a solution with the required linear running time $O(n+m)$ you will lose credit regardless of whether the test cases execute correctly.

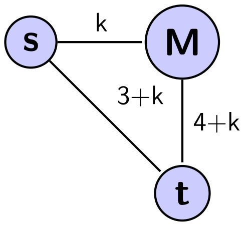
3. (3 points) Consider the following algorithm for finding the shortest path from node s to node t in a directed graph with some negative edges: add a large constant to each edge weight so that all the weights become positive, then run Dijkstra's algorithm starting at node s, and return the shortest path found to node t.

Is this a valid method (assuming the graph has no negative weight cycles- cycles whose total weight is negative)? Either explain why it works correctly, or give a counter-example (and explain clearly why your graph is a counter-example).

Solution: YOUR SOLUTION HERE



The shortest path in the above graph is s to t through M, and the total weight is zero.
If we add a constant $k \geq 0$, in this case k+2, we get below graph



It turns out that the shortest path is now s to t directly.
It fails the claim in the question and the method fails.

4. **(EXTRA CREDIT)** As described in class, you will write a method that finds (and prints) the unique *simple* path (no repeat vertices) between two vertices in a minimum spanning tree found by Prim's algorithm. Your method will be added to the existing implementation of Prim's algorithm given by Robert Sedgewick and Kevin Wayne and explained here. Specifically, you will add code to the `findPath` method that has been added to the version of `PrimMST.java`

that is posted to Canvas. This method does not return anything but as described in the documentation, it should print the unique path between the two input vertices given by the current minimum spanning tree. Your printed path should not contain any duplicate vertices and your method should run in time $O(n)$ where n is the number of vertices. I recommend you use one of the algorithms described in class.

In order to run the `PrimMST.java` file you will need to download several additional files from Sedgewick and Wayne's website. You are welcome to add additional private methods or data fields but you may not modify the method signature of the `findPath` method or any other methods or data fields. You also may not modify any files besides the `PrimMST.java` file. Your solution should not print anything aside from the path between the input vertices.

Instead of downloading just the files you need it may be easier to download the JAR file here. If you haven't used a JAR file before I encourage you to look up how to do this online. Here is one reference for Eclipse and one for the command-line.

5. **(PRACTICE)** For this problem, assume that the graph $G = (V, E)$ is undirected, connected and has at least 2 vertices. Do not assume that the edge weights are distinct. **PROVE** the following statements about minimum spanning trees.

   (a) There exists a graph G which has a cycle C, and e is the **unique lightest** edge in C so that e is a part of **at least one but not all** MSTs of G.

   Solution: YOUR SOLUTION HERE

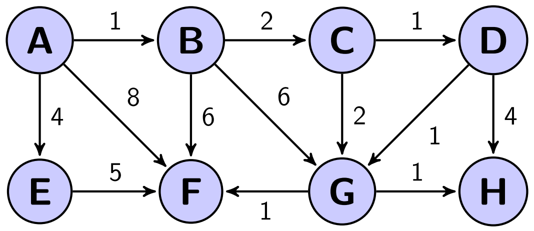
   (b) There exists a graph G which has a cycle C, and e is the **unique lightest** edge in C so that e is **never** a part of any of the MSTs of G.

   Solution: YOUR SOLUTION HERE

   (c) For any graph G, if G has a cycle C, and e is the **unique heaviest** edge in C then e is never a part of any MST on G.

   (d) For any graph G, let e be an edge of **minimal** weight then there is some MST which includes e.

6. **(PRACTICE)** Suppose Dijkstra's algorithm is run on the following graph, starting at node A.

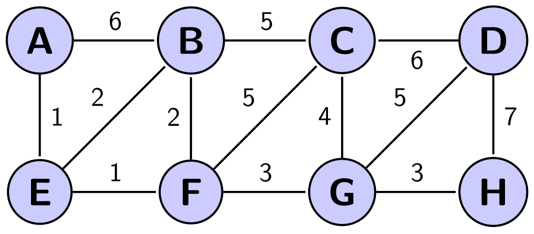


   (a) Draw a table showing the intermediate distance values of all the nodes after each iteration of the while loop.

   (b) Show the min-heap (or priority heap) after each iteration of the while loop. List the heap operations that were performed (e.g. DeleteMin(Q), DecreaseKey(Q, v, 5)).

   (c) Give the final shortest path tree.

7. **(PRACTICE)** Run **Prim's Algorithm** on the following graph.



(a) Show the tree at each step. Indicate the final spanning tree.

(b) Draw a table showing the intermediate cost values of all the nodes at each iteration of the algorithm.

(c) Show the min-heap (or priority heap) at each iteration of the algorithm.

(d) How many minimum spanning trees does the graph have? Show each one.

Additional Programming Instructions:

Note that your code will be automatically run on a standard set of test cases. In order to ensure that you do not lose points, follow the instructions below.

- Your code must compile without any errors using the version of Java on the lab computers. If your code does not compile you will not receive any points for the assignment.
- Do not modify any of the methods signatures (i.e. name, return type or input type). Note that you are always welcome (and encouraged) to add additional methods but these will not be run directly by the test code.
- You are not allowed to use packages (e.g. no statement `package ...` at the top of your file).
- No extra folders or files in your submission. Zip up only the files you need to submit not the folder they are in.
- Your solution should not print anything unless explicitly instructed to.

Grading Criteria:

Your work will be graded on both correctness **and clarity**. Write complete and precise answers and show all of your work. Your pseudo-code and proofs should follow the guidelines posted on Canvas and discussed in class.

Evaluation Rubric for Problem 1 and 2.

Component	Description
Style & Documentation (1 pts)	I'll be watching for style issues as well as correct output. See the syllabus for some general style guidelines (e.g. your code should be well-documented).
Correct Output on Test Cases* (4 pt)	Your code will be run on a standard set of test cases. Make sure to test your code thoroughly!

*If your code does not run in linear $O(n+m)$ time you will lose at least 2 points.

Evaluation Rubric for Problem 3.

Component	Requirement for Full Credit
Solution Correctness (1pt)	The answer is correct.
Written Explanation (2pt)	Your solution includes a logical argument explaining why your answer is correct. The explanation is clear, correct and complete. It includes full sentences and is easily understood by any student in the class. Think of this as a proof and remember what you learned in Math 128.

Evaluation Rubric for Problem 4:

Component	Description
Extra Credit (2 pt)	Your method will be run on a standard set of test cases. To receive any extra points this method must run in $O(n)$ time and follow the class style guidelines.