Homework 4: Graphs and Sorting

Due: August 6 2023 11:59pm PST (on Github)

Instructions

Over the last few weeks, we spent time talking about graphs, both directed and undirected. We also spent a couple lectures talking about sorting algorithms. This problem set will give you some practice around both these topics.

For each of the sections of the homework, we provide you with some test cases that you can use to verify the correctness of your code. Also, there are hidden test cases that we will not be releasing. However, by using the autograder we set up on Gradescope (see more details in the **Testing and Autograder** section), you'll have an idea of whether or not your code passes the hidden test cases.

Setting Up

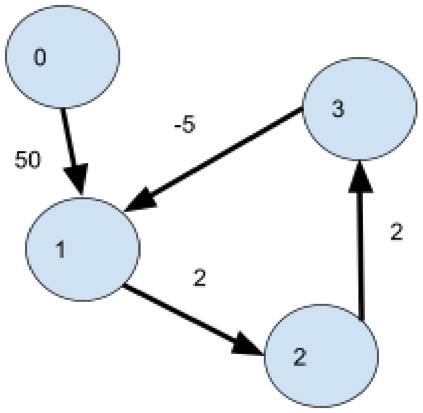
Pull the homework starter code from the Assignments repo by running **git pull origin main**. Make a copy of the homework4 folder and move it to your workspace. **You do not need to add any jar files for this assignment from the 'jars' folder from Canvas.**

Graphs

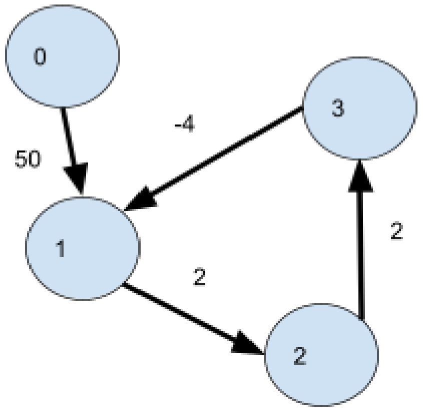
We initialize our graph by passing in the number of vertices our graph has to the Graph constructor. We then need to add edges to our graph.

You'll see that we have provided text files in the graphs directory. Each of these files represents a graph. Each line is a comma is a comma separated string that represents an edge. The first value is the source node and the second value is the second node. The third value, if present, represents the edge weight. We have provided a helper function called **createGraph that takes** three parameters: a graph, a string representing the file path that contains the graph data, and a boolean value indicating whether the graph is directed or not. This method reads the text file and adds the appropriate edges to the graph. Passing in the boolean true value as the third parameter to the createGraph function tells it to construct a directed graph. You do not need to modify this function in any way. You can use this method to come up with your own test cases to verify the correctness of your code. See the provided test cases to understand how you can develop your own test cases.

1. Implement the function **hasNegativeCycle in Graph.java** that returns true if the graph has a negative cycle and false otherwise (6 points).



Your function should return true for the graph above and false for the graph below

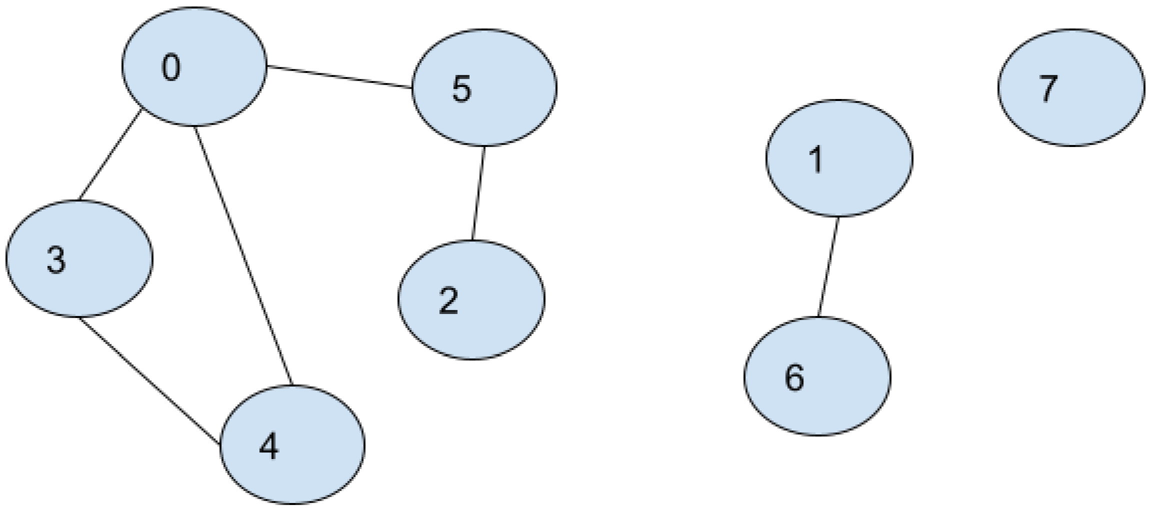


Test Cases:

1. testSmallNoNegativeCycle: 1 points
2. testSmallNoNegativeCycle: 1 point
3. Hidden Test Case 1: 2 points
4. Hidden Test Case 2: 2 points

2. In undirected graphs, a connected component is a subset of vertices where every vertex in that subset has a path to every other node in that subset.

The graph below has three connected components: {7}, {1, 6}, and {0,5,2,3,4}.



In Graph.java, implement the **connectedComponents** function that returns how many connected components there are in the graph. Your function should return 3 for the example above. (6 points).

Test Cases:

1. testSingleComponentSmall: 1 points
2. testMultipleComponentsSmall: 1 points
5. Hidden Test Case 1: 2 point
6. Hidden Test Case 2: 2 points

Sorting

1. Implement the function **triValueSort in Sort.java** that sorts an input array that consists of the values -1, 0 and 1. Your algorithm should run in linear time and not use any extra memory. Your function must modify the input array itself. Allocating extra memory or having runtimes that are worse than O(N) will only get you 1 point (5 points).

   Test Cases:

   1. testGeneral: 2 points
   2. testSmall: .5 points
   3. testAlreadySorted: 1 point
   4. Hidden Test Case 1: 1.5 points

Testing and Autograder

Create a folder called **hw** and put **only Graph.java and Sort.java**. Zip up this folder and upload it to the Homework 4 assignment on Gradescope.

Total: 17 points

Submissions

Upload your final code to your private Github repo.