CS242 - Spring 2020 - Assignment #4

Assigned: April 1st, 2020
Due: April 11th, 2020
(Extra credit: May 1st, 2020)
NO LATE SUBMISSIONS.
Non-coding answers may be resubmitted once, after receiving corrections on the first attempt. **That is, if you do not submit anything in the first attempt, you cannot resubmit.**

Collaboration policy: The goal of assignment is to give you practice in mastering the course material. Consequently, you are encouraged to collaborate with others. In fact, students who form study groups generally do better on exams than do students who work alone. If you do work in a study group, however, you owe it to yourself and your group to be prepared for your study-group meeting. Specifically, you should spend at least 30–45 minutes trying to solve each problem beforehand. If your study group is unable to solve a problem, it is your responsibility to get help from the instructor before the assignment is due.

For this assignment, you can form a team of up to three members. Each team must write up each problem solution and/or code any programming assignment without external assistance, even if you collaborate with others outside your team for discussions. You are asked to identify your collaborators outside your team. **If you did not work with anyone outside your team, you must write "Collaborators: none."** If you obtain a solution through research (e.g., on the web), acknowledge your source, but write up the solution in your own words. **It is a violation of this policy to submit a problem solution that any member of the team cannot orally explain to the instructor.** No other student or team may use your solutions; this includes your writing, code, tests, documentation, etc. It is

a violation of this policy to permit anyone other than the instructor and yourself read-access to the location where you keep your solutions.

Submission Guidelines: Your team has to submit your work on Blackboard (no email) by the due date. Only one submission per team is necessary. For each class in the programming assignments you must use the header template provided in Blackboard. Make sure that you identify your team members in the header, and any collaborators outside your team, if none, write "none". Your code must follow the Java formatting standards posted in Blackboard. Format will also be part of your grade. To complete the submission, you have to upload two files to Blackboard: your source file and your class file. Your answers to questions that do not require coding must be included in the remarks section of the header. **The submission will not be accepted in any other format.**

Style and Correctness: Keep in mind that your goal is to communicate. Full credit will be given only to the correct solution which is described clearly. Convoluted and obtuse descriptions might receive low marks, even when they are correct. Also, aim for concise solutions, as it will save you time spent on write-ups, and also help you conceptualize the key idea of the problem.

Assignment #4 Grading Rubric

Coding:

Program characteristic	Program feature	Credit possible		
Design 30%	Algorithm	30%		
Functionality 30%	Program runs without errors	20%		
	Correct result given	10%		
Input 15%	User friendly, typos, spacing	10%		
	Values read in correctly	5%		
Output 15%	Output provided	10%		
	Proper spelling, spacing, user friendly	5%		
Format 10%	Comments, name	5%		
	Indentation	5%		
	TOTAL	100%		

Non-coding:
Embedded in questions.

1(50)	2(25)	3(25)	**TOTAL**(100)

Assignment: Depth First Search (DFS) is a fundamental graph traversal used as building block to solve important graph problems such as Topological Sort, finding Strongly Connected Components, and others. Hence, the time efficiency of those algorithms depends heavily on implementing efficiently DFS. Given a directed graph $G = \{V, E\}$ encoded as an adjacency list, an efficient implementation of DFS runs in $O(|V| + |E|)$ steps. The purpose of this homework is to evaluate experimentally the performance of an efficient implementation of DFS.

1. **(50 points)** Write a program that, given the adjacency list of a directed graph, computes DFS efficiently. Your program should do the following.
   (a) Prompt the user to input the number of nodes and the number of edges.
   (b) Create an adjacency list choosing the edges at random (do not forget that it is a directed graph).
   (c) Compute DFS (including discovery and finishing times) following the pseudocode in the textbook.
   (d) Measure and display the running time

2. **(25 points)** Using the DFS program of part (1), fill in the following chart with the running times observed for different edge-set sizes.

	$\|E\| = \|V\| - 1$	$\|E\| = \lfloor(\|V\| - 1)^{3/2}\rfloor$	$\|E\| = (\|V\| - 1)^2$
$\|V\| = 10$			
$\|V\| = 100$			
$\|V\| = 1000$			

3. **(25 points)** Give an approximate formula (with constants, not big-O) for the asymptotic running time of DFS based on your experiments. How does this compare with the expected $O(|V| + |E|)$? If the results differ, overview the code of the data structures used for the adjacency list and explain what might have happened.

4. **Extra credit** DFS is used as a subroutine for more complex computations in graphs. One of them is Topological Sort. Topological Sort is defined on directed graphs that do not have cycles, customarily called *Directed Acyclic Graphs* (DAG). If the input graph of Topological Sort is not a DAG, the algorithm returns an incorrect output. So, it is useful

to detect cycles to stop the execution in that case. Fortunately, cycles can be detected while running DFS. Your task is to modify the DFS algorithm above to detect cycles. To test your program, write also a method that produces two large directed graphs (try $|V| > 1000$), one with cycles and the other without cycles, and runs the cycle detection DFS in both.