CMPSC 465 Data Structures & Algorithms
Fall 2025 Ke Chen and Yana Safonova

HW 5

Due October 13th, 10:00 pm

Instructions: You are encouraged to solve the problem sets on your own, or in groups of three to five people, but you must write your solutions strictly by yourself. You must explicitly acknowledge in your write-up all your collaborators, as well as any books, papers, web pages, etc. you got ideas from.

Formatting: Each part of each problem should begin on a new page. Each page should be clearly labeled with the problem number and the problem part. The pages of your homework submissions must be in order. **When submitting in Gradescope, make sure that you assign pages to problems from the rubric. You risk receiving no credit for it if you do not adhere to these guidelines.**

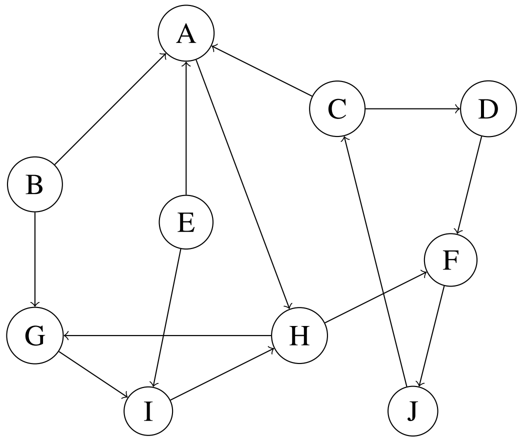
Late homework will not be accepted. Please, do not ask for extensions since we will provide solutions shortly after the due date. Remember that we will drop your lowest three scores.

This homework is due Monday, October 13, at 10:00 pm electronically. You need to submit it via Gradescope. Please ask on Canvas about any details concerning Gradescope.

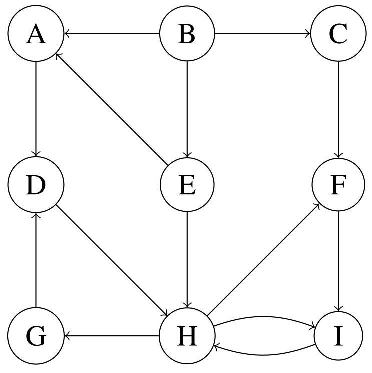
1. (20 pts.) **Strongly Connected Components**

Run the strongly connected components algorithm on the following directed graphs. When doing DFS on the reverse graph G^R: whenever there is a choice of vertices to explore, always pick the one that is alphabetically first.

(i)



(ii)



In each case answer the following questions.

(a) In what order are the strongly connected components (SCCs) found?

(b) Which are source SCCs and which are sink SCCs?

(c) Draw the "metagraph" (each "meta-node" is an SCC of G).

(d) What is the minimum number of edges you must add to this graph to make it strongly connected (namely, consisting of just a single SCC)?

2. (20 pts.) **Strongly Connected Components.** Give an efficient algorithm which takes as input a directed graph $G = (V, E)$ and determines whether or not there is a vertex $s \in V$ from which all other vertices are reachable. Explain why your algorithm is correct and analyze its running time.

3. (20 pts.) **Unique Shortest Path.** Shortest paths are not always unique: sometimes there are two or more different paths with the minimum possible length. Show how to solve the following problem in $O((|V| + |E|)\log|V|)$ time. Explain why your algorithm is correct and justify its running time.

Input: An undirected graph $G = (V, E)$; edge lengths $l(u, v) > 0$ for all $(u, v) \in E$; a starting vertex $s \in V$.

Output: A Boolean array `usp[·]`: for each node u, the entry `usp[`u`]` should be `true` if and only if there is a *unique* shortest path from s to u. (By convention, we set: `usp[`s`] = true`.)

4. (20 pts.) **Dijkstra's Algorithm.** Consider a directed graph in which the only negative edges are those that leave s; all other edges are positive. Can Dijkstra's algorithm, started at s, fail on such a graph? Assume that s is a source node, i.e., s has no incoming edges. Prove your answer.

5. (20 pts.) **BFS on Directed Graph.** Consider a directed graph $G = (V, E)$. Depth first search allows us to label edges as tree, forward, back and cross edges. We can make similar definitions for breadth first search on a directed graph:

1. A BFS tree edge is an edge that is present in the BFS tree.
2. A BFS forward edge leads from a node to a non-child descendant in the BFS tree.
3. A BFS back edge leads to an ancestor in the BFS tree.
4. All other edges are BFS cross edges.

(a) Explain why it is impossible to have BFS forward edges.

(b) Give an efficient algorithm that classifies all edges in G as BFS tree edges, back edges, or cross edges.