Homework 9: Due November 20 (11:59 p.m.)

Instructions

- **Answer each question on a separate page.**
- **Honors questions are optional. They will not count towards your grade in the course. However you are encouraged to submit your solutions to these problems to receive feedback on your attempts. Our estimation of the difficulty level of these problems is expressed through an indicative number of stars ($'*' =$ easiest) to ($'*****' =$ hardest).**
- **You must enter the names of your collaborators or other sources as a response to Question 0. Do NOT leave this blank; if you worked on the homework entirely on your own, please write "None" here. Even though collaborations in groups of up to 3 people are encouraged, you are required to write your own solution.**

Question 0: List all your collaborators and sources: ($-\infty$ points if left blank)

Question 1: (5 points)

The disjoint set data structure maintains a set of disjoint sets and supports the following operations.

1. createSet(x): creates a set containing the single element x.
2. findSet(x): returns a pointer to the unique set containing x.
3. merge(x, y): merges the unique sets containing x and y.

See **CLRS 4th Edition (available via NYU Libraries), Chapter 19 (Data Structures for Disjoint Sets)** for a more detailed description. The following is an example which tracks how a given disjoint set data structure changes corresponding to the sequence of operations applied to it.

Operation performed	Collection of disjoint sets								
Initial sets	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$
merge(a, b)	$\{a, b\}$		$\{c\}$	$\{d\}$	$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$
merge(c, d)	$\{a, b\}$		$\{c, d\}$		$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$
merge(d, f)	$\{a, b\}$		$\{c, d, f\}$		$\{e\}$		$\{g\}$	$\{h\}$	$\{i\}$
merge(a, h)	$\{a, b, h\}$		$\{c, d, f\}$		$\{e\}$		$\{g\}$		$\{i\}$
merge(e, g)	$\{a, b, h\}$		$\{c, d, f\}$		$\{e, g\}$				$\{i\}$
merge(b, e)	$\{a, b, e, g, h\}$		$\{c, d, f\}$						$\{i\}$

Now, fill out the following table and state how many distinct sets are present after all the operations have been applied.

Operation performed	Collection of disjoint sets									
Initial sets	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
merge(a, c)										
merge(b, d)										
merge(b, e)										
merge(f, i)										
merge(f, j)										
merge(j, h)										
merge(g, e)										
merge(a, h)										
merge(h, d)										

Question 2: (5 points)

Using the disjoint set data structure from Question 1, design an algorithm that computes the connected components of a given graph $G = (V, E)$. The algorithm should output a disjoint set data structure such that for any $v \in V$, the query findSet(v) returns a pointer to the connected component containing v. (Hint: read CLRS Chapter 19. Data Structures for Disjoint Sets).

Question 3: (10 points)

Recall that the *adjacency matrix* representation of a directed graph $G = (V, E)$ is the $|V| \times |V|$ matrix A, where $A[i][j]$ is 1 if there is an edge from vertex i to vertex j, and 0 otherwise.

Describe an algorithm that, given the adjacency matrix A, checks if there is any vertex in G that has edges coming to it from *all other* vertices but no edges going out from it (this is known as a **universal sink**). Your algorithm should return TRUE if G has a universal sink, and FALSE otherwise. You may assume no vertex in the graph has an edge going into itself (i.e., no self-loops). Your algorithm should run in $O(|V|)$ time. Justify why it is correct, as well as why it satisfies this run-time bound. (Hint: Notice that for any pair of vertices i, j, the edge $i \to j$ is either present or absent. If it is present, then i cannot be a universal sink. If it is absent, then j cannot be a universal sink. Justify this fact and use it to create your algorithm.)

Question 4: (5+10=15 points)

Let P be a program which, given as input a directed graph $G = (V, E)$ as well as two vertices $s, t \in V$, outputs the shortest path between them. The run-time of P is $O(|V| + |E|)$, but assume that P is highly optimized so that it runs significantly faster than your custom shortest path algorithm. We will use P to solve the following problem.

We are given a directed graph G where each of its edges is colored either red or blue. We want to find the shortest *separable* path from some vertex s to some other vertex t. A separable path consists of first some number (possibly 0) of red edges followed by some number (possibly 0) of blue edges. In other words, once your path uses a blue edge, all the following edges must be blue.

1. Design an algorithm to find the shortest separable path from s to t. Your algorithm must invoke P *exactly once* on a carefully constructed graph, and it should run in time $O(|V| + |E|)$. Do NOT try to modify BFS so that it outputs a separable path; use the optimized program P.

(Hint: Consider the subgraph formed by only taking the red edges. Next, consider the subgraph formed by only taking the blue edges. Notice that you are to spend some number of steps in the first part and then move to the second part and stay there. Can you combine the two graphs somehow?)

2. Justify the correctness of your algorithm, and show that it runs in time $O(|V| + |E|)$.

Question 5: (5+5=10 points)

1. Consider the graph in Figure 1. Say we begin a DFS traversal starting at node A. List the discovery (i.e., when we mark a node "gray") and finishing times (i.e., when we mark a node "black") of all the vertices (you may assume each successive step in the traversal takes time 1). Assume that the adjacency lists in the representation of the input graph are in alphabetical order.

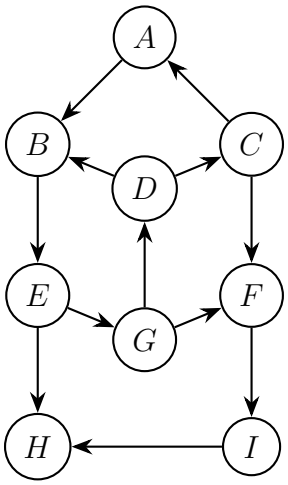

Figure 1: Graph for Question 4.1

2. Consider the directed acyclic graph in Figure 2. Consider a DFS traversal of this graph starting from vertex A. List the discovery and finishing times of all the vertices. Then, draw the vertices on a *line* in decreasing order of finish time. Now draw all the edges from the original graph on this line of "sorted" vertices. Does this way of sorting vertices have a special name? (Assume that the adjacency lists in the representation of the input graph are in alphabetical order.)

Question 6: (5+5=10 points)

For any vertex v in directed graph G, we denote by $v.\mathsf{d}$ the time at which DFS on G discovers (i.e., marks as "gray") the vertex v. Similarly, denote by $v.\mathsf{f}$ the time at which DFS fully explores (i.e., marks as "black") the vertex v. Give counterexamples to the following statements:

1. If a directed graph G contains a path from v to w, and if $v.\mathsf{d} < w.\mathsf{d}$ in a depth-first search of G, then w is a descendant of v in the depth-first forest produced.

2. If a directed graph G contains a path from v to w, then any depth-first search must result in $w.\mathsf{d} < v.\mathsf{f}$.

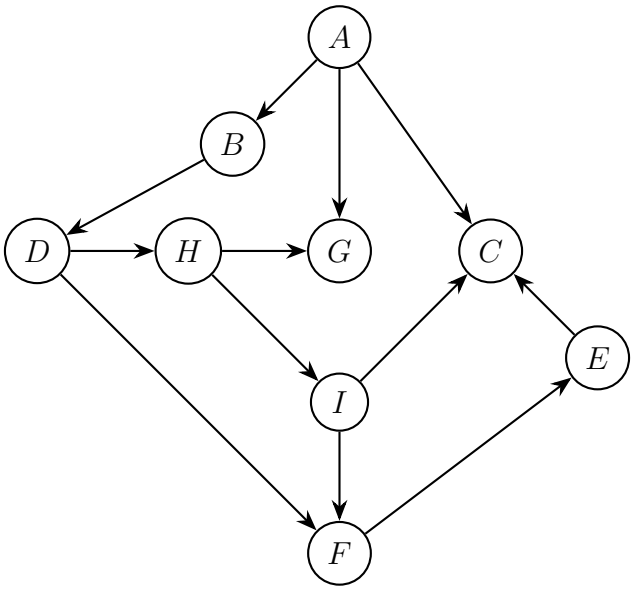

Figure 2: Graph for Question 4.2