Data Structures and Algorithms II
Fall 2021 Section 1
Midterm

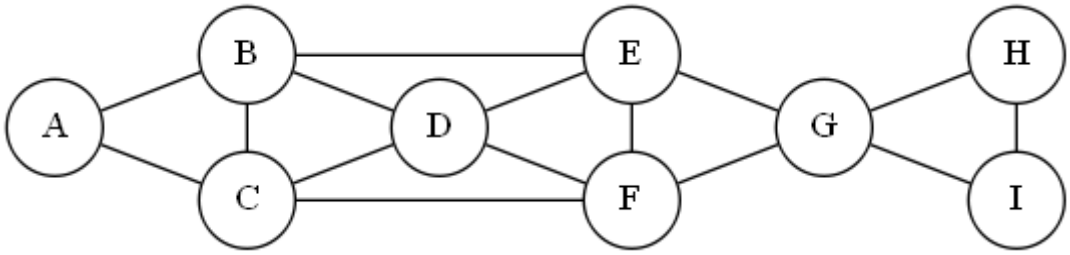
Name: _____

E-mail: _____

Write all of your answers directly within this test booklet. Use the backs of pages if necessary. This is an open-book, open-notes test. You should not use any electronic devices. You may use whatever scrap paper you like, but do not hand it in.

Please also write your name at the top of each page.

(1) Consider the following undirected graph:



(a) Without applying an algorithm to find an Euler circuit (a.k.a. Euler cycle), explain how you can examine the graph above to determine that it is possible to find an Euler circuit.

(b) Now suppose that you apply the algorithm discussed in class to find an Euler circuit in the above graph, starting at node B. Suppose the first depth-first search (DFS) stops after it detects the following cycle:

B → A → C → D → B → C → F → E → B

Explain why the algorithm would keep going after it returns to B the first time (in the middle of this cycle) but stop after it returns to B the second time (at the end of this cycle)?

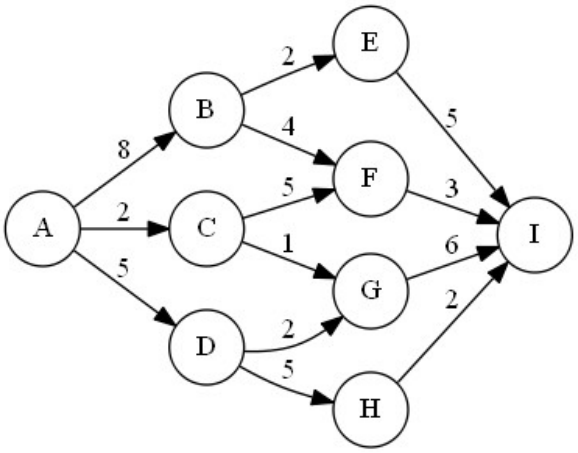
(c) Draw what the graph looks like after removing the appropriate edges up to this point in the algorithm:

(d) Where would the algorithm start its next DFS? Why would it start there?

(e) Starting at the location specified in (d), specify any cycle that might be found by the next DFS when the algorithm concludes this DFS (not necessarily ending the algorithm)?

(f) Specify the updated cycle after the result of the DFS indicated in part (e) is merged with the cycle indicated in part (b).

(2) Consider the following network flow graph (a.k.a. flow network):



The labels of edges represent their capacities.

(a) You want to solve the maximum-flow problem using the Ford-Fulkerson method. You decide to implement this using the Ford-Fulkerson algorithm (i.e., you choose the augmenting path that leads to the biggest increase in flow). At the start, you realize there is just one choice for your first augmenting path. What is the first path that you choose, and how much can you pump through it?

(b) Pump the amount that you indicated in part (a) through the path that you indicated in part (a). Draw the resulting residual graph below?

(c) Based on the original flow network as shown, use the max-flow, min-cut theorem to determine the largest flow that can be pumped through this flow network. In other words, without stepping through any algorithm, eyeball the graph to find the appropriate cut that is relevant to the theorem. Indicate the cut by drawing an appropriate curvy line over the figure above. Then state in the space below the value of the maximum possible flow.

(3) Draw all possible leftist heaps with exactly four total nodes, containing the keys 1, 2, 3, and 4. (Indicate the key value inside each node.)

(4) Answer the following questions related to binary heaps:

(a) Insert items with the following keys, in the order shown, into an initially empty binary heap: 69, 46, 20, 60, 35, 71, 27. Draw the resulting binary heap, with keys indicated inside nodes, *after each insertion*.

(b) Now assume that all the data from part (a) arrives at the same time, initially in an array in the same order that is shown: 69, 46, 20, 60, 35, 71, 27. This time, apply the worst-case linear routine for creating a binary heap, as discussed in class, and show the resulting heap.

(5) In class, we have examined the following pseudo-code for Dijkstra's algorithm:

```
WeightedPLDijkstra (Graph G, Vertex s)
   for each vertex v in G
     d_v ← ∞
     known_v ← FALSE
   d_s ← 0
   p_s ← NULL
   while there are still unknown vertices
     v ← the unknown vertex with the
         smallest d-value
     known_v ← TRUE
     for each edge from v to vertex w
       if d_v + c_v,w < d_w
         d_w ← d_v + c_v,w
         p_w ← v
```

For this question, you are going to consider making two modifications to this algorithm, one at a time. Answer parts (c) and (d) independently of (a) and (b).

(a) Suppose you want to modify the pseudo-code to also accept a destination vertex, t. The algorithm should stop as soon as the best possible path from s to t has been definitively determined (assuming the graph does not contain any negative-cost edges). In addition to correctly setting all the d-values and p-values, suppose that the algorithm should return TRUE if t is reachable from s (i.e., there is at least one path from s to t in the graph), and FALSE if t is not reachable from s. Next to the pseudo-code above, indicate all necessary changes, with arrows to make it clear where you are inserting the additional code.

(b) Using big-Theta notation, express the worst-case running time of the updated algorithm in terms of the number of vertices and/or edges. Assume that the algorithm is implemented efficiently, using a binary heap and adjacency lists.

The pseudo-code is re-printed here for convenience:

```
WeightedPLDijkstra (Graph G, Vertex s)
   for each vertex v in G
     d_v ← ∞
     known_v ← FALSE
   d_s ← 0
   p_s ← NULL
   while there are still unknown vertices
     v ← the unknown vertex with the
         smallest d-value
     known_v ← TRUE
     for each edge from v to vertex w
       if d_v + c_v,w < d_w
         d_w ← d_v + c_v,w
         p_w ← v
```

(c) Now suppose you want to modify the pseudo-code so that it does not make any assumptions about negative cost edges. The algorithm should run as usual. However, at the end, in addition to setting all the d-values and p-values as indicated above, it should also return TRUE if the graph does not contain any negative-cost edges (indicating that all discovered paths and distances are optimal), and FALSE if the graph does contain one or more negative-cost edges (indicating that some of the discovered paths and distances may not be optimal). Next to the pseudo-code above, indicate all necessary changes, with arrows to make it clear where you are inserting the additional code. (This version will compute a path from s to every other vertex in the graph; it is not expected to stop at a specified destination.)

(d) Assuming that there are negative cost edges, and that all vertices are reachable from s, will the algorithm find a valid path (although not necessarily optimal) from s to every other vertex in the graph? Briefly explain your answer.