Data Structures and Algorithms II
Fall 2019
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) In class, we have examined the following pseudo-code for Dijkstra's algorithm:

```
WeightedPathLengthDijkstra ( 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 distance
          known_v ← TRUE
          for each outgoing 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 Dijkstra's algorithm, at the start of each iteration of the outer loop, the "known" vertices are the ones for which we have already found the optimal path from s (our specified starting vertex), and for each of these vertices, d_v is the optimal path length, and p_v is the previous vertex on the best path from s to v. The "unknown" vertices are the ones for which the we have not yet found the guaranteed optimal path from s, and for each of these vertices, d_v is the cost of the best path from s to v going through only known vertices, and p_v is the previous vertex on the best from s to v found so far.

When we covered Prim's algorithm for finding a minimum spanning tree of an undirected graph, we did not go over pseudo-code, because we said it was extremely similar to that of Dijkstra's algorithm (shown above). The name of the routine would change, of course, and "s" would represent an arbitrary specified vertex with which to start the spanning tree.

(a) For Prim's algorithm, at the start of each iteration of the outer loop, what is represented by the "known" and "unknown" vertices?

(b) For Prim's algorithm, what is represented by d_v and p_v for known vertices?

(c) For Prim's algorithm, what is represented by d_v and p_v for unknown vertices?

(d) Circle any part or parts of the pseudo-code that must change for Prim's algorithm; then write the modified part or parts below.

(e) Can a binary heap be used to store the unknown vertices for Prim's algorithm, as with Dijkstra's algorithm? If no, briefly explain why not? If yes, specify what operation is used to find the unknown vertex with the smallest distance and what operation is used to update values of d_w within the inner loop?

(2) Answer the following questions about binary heaps, leftist heaps, and binomial queues (each supporting deleteMin, as discussed in class).

(a) Insert the numbers 50, 40, 30, 20, and 10, in that order, into an initially empty binary heap. Show the state of the binary heap after each insertion.

(b) Insert the numbers 50, 40, 30, 20, 10, in that order, into an initially empty leftist heap. Show the state of the leftist heap after each insertion.

(c) Insert the numbers 50, 40, 30, 20, 10 in that order, into an initially empty binomial queue. Show the state of the binomial queue after each insertion.

(3) Briefly answer the following questions about biconnectivity and depth-first spanning trees.

(a) Is it possible for a graph that has two or more articulation points to become biconnected with the inclusion of one additional edge? Explain your answer.

(b) If a graph has only one articulation point, is it guaranteed that there is a single edge that can be added that would make the graph biconnected? Explain your answer.

(c) In a depth-first spanning tree, is it possible for a node to have a Low value that is greater than its Num value (the definitions of Low and Num were covered in class)? Explain your answer.

(d) Assume that a depth-first spanning tree is created for a connected, undirected graph with V vertices and E edges. In terms of V and E, how many tree edges will there be in the depth-first spanning tree? How many back edges will there be in the depth-first spanning tree?

(4) Briefly answer the following questions about various types of priority queues.

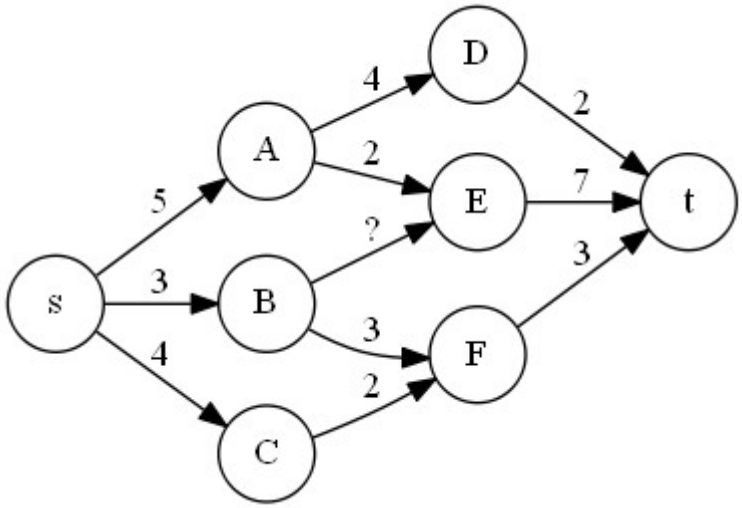
(a) For binary heaps (the type you implemented for your second programming assignment), we have learned that there is no logarithmic merge operation possible. Is there a guaranteed, worst-case linear merge operation? Explain your answer.

(b) For a leftist heap, explain how deleteMin can be considered a special case of merge?

(c) For a binomial queue, explain how deleteMin can be considered a special case of merge?

(d) For a well-implemented binomial queue, compare the actual time it would take to insert an additional node into a binomial queue with 31 nodes and the actual time it would take to insert an additional node into a binomial queue with 32 nodes. Would one take longer than the other? Explain your answer.

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



Note that one of the capacities, specifically of the edge from B to E, is unknown. However, you know it is a nonnegative real number (possibly zero).

(a) What is the smallest possible maximum flow that the given flow network might have?

(b) Redraw the flow network in the space below, filling in a capacity for edge (B, E) that leads to the value indicated in part (a). Then draw a curvy line through your drawn flow network indicating the cut that would be maxed out with this flow.

(c) Unrelated to parts (a) and (b), what is the largest possible maximum flow that the given flow network might have? Also indicate the minimum possible capacity that edge (B, E) would require to lead to this maximum flow.

(d) Redraw the flow network in the space below, filling in a capacity for edge (B, E) larger than the minimum required edge capacity indicated in part (c), thus being one possible value leading to the maximum flow indicated in part (c). Then draw a curvy line through your drawn flow network indicating the cut that would be maxed out with this flow.