CSE 100: Algorithm Design and Analysis
Final Exam

Spring 2020

- This is a take-home exam with no proctoring. You can start at any time once you get access to this exam. You have to submit your solutions by **11:00am, Tue, 5/12**, through catcourses (Final under Assignments). You can resubmit until the deadline. If there are any technical issues with uploading, it is extremely important to immediately contact the instructor or the TAs, and email solutions to them.
- It is your **responsibility** to check if your solutions have been uploaded successfully. In addition to submitting through **catcourses**, we strongly recommend you to email your solutions to **cse100.s20@gmail.com** to be safe – if we haven't received your submission via Catcourses, nor via email, you will earn 0 points. You will receive an automatic reply: "This is CSE 100, Spring 2020 email account. Please note that this reply only confirms that you emailed at the correct email address. Please keep the email you sent in your sent box for the record."
- This is an open-book exam. The **only** resources you can use during the exam are all **course materials** uploaded to catcourses plus the **textbook**. In particular, this means that you are **NOT** allowed to search for solutions on the internet. No calculator is allowed. You have to take the exam by yourself.
- There are 7 problems in total. The first 5 problems are of at least intermediate difficulty.
- Please make your answers concise as well as precise. If there is something in the question that you believe is open to interpretation, then please go ahead and interpret, but state your assumptions in your answer.
- If you have questions, you can email the instructor and the TAs. However, we may not answer your questions on time as we can't be available all the time. In particular, do not expect reply during non-business hours.

You're required to take the following honor pledge prior to taking the exam.

By completing this exam, I acknowledge and confirm that I will not give or receive any unauthorized assistance on this examination. I will conductmyself within the guidelines of the university academic integrity guidelines.

1. (20 points) Recall that the Dijkstra algorithm maintains the set of vertices S whose shortest distances have been determined, and grows it by adding one vertex to S in each iteration. For more details of the algorithm, you can see the algorithm in Ch 24.3 or the lecture slides. Answer if the following claim is **true or not**, and explain **why**.

   Claim: Let's assume that the directed input graph $G = (V, E)$ has no negative weight edges, and further all vertices are reachable from the given source vertex s. Suppose vertices are added to S in this order: $s = v_1, v_2, v_3, ...v_n$. Then, it must be the case that $\delta(s, v_1) \leq \delta(s, v_2) \leq ... \leq \delta(s, v_n)$. Here, $\delta(s, v)$ denotes the (shortest) distance from s to v.

   You will get 0 points without explanation.

   Sol. This is true. Suppose not for the sake of contradiction. It means that $\delta(s, v_k) > \delta(s, v_{k+1})$. Consider an arbitrary shortest path P from s to v_{k+1}. Clearly, v_k is not on P as otherwise it would be a contradiction due to the optimality of subpaths. Let $v \in \{v_{k+1}, v_{k+2}, ..., v_n\}$ be the first vertex we visit when traversing P from s. Then, again due to the suboptimality of the subpaths, it must be that $\delta(s, v) \leq \delta(s, v_{k+1})$. Since v is a 'neighbor' vertex from S when $S = \{v_1, v_2, ..., v_{k-1}\}$, at the moment, we had $v.d = \delta(s, v)$, which is less than $\delta(s, v_k) \leq v_k.d$. Then, the algorithm must have chosen v over v_k, which is a contradiction.

2. (15 points) We learned that assuming that all edges have distinct weights and the graph is connected, there is a unique MST and an edge is in the MST if and only if the edge is safe. Answer if the following claim is **true or not**, and explain **why**.

   Claim: Assume that all edges have distinct weights and the graph is connected. Suppose an edge e has the maximum weight among all edges on some cycle C of the graph. Then, the edge is **not** safe.

   You will get 0 points without explanation.

   Sol. Let (u, v) denote the edge. Consider any cut separating u and v – so, (u, v) is a an edge crossing the cut. Then, there must exist another edge on C that crosses the cut, and the edge must be cheaper. So, (u, v) is not a light edge for the cut. Therefore, (u, v) is not safe.

3. (15 points) Suppose we ran the Bellman-Ford algorithm on a directed graph $G = (V, E)$ with no negative weight cycle. As a result, we obtained $v.d = \delta(s, v)$ where s is the given source vertex. Now, you're asked to find a shortest **path** from s to a given vertex v in time $O(E+V)$. Describe your algorithm. No need to explain why it works. As usual, you can describe your algorithm in plain English or using pseudocode.

   Sol. Consider each edge (u, v). If $v.d = w(u, v) + u.d$, then let $v.\pi = u$. And let $s.\pi = NIL$. Then, we can use the following pseudocode.

```
Print($u$)
If u = NIL return
```

```
Print($u.\pi$)
output $u$
```

Now we just call Print(v).

4. (15 points) You're given a directed graph $G = (V, E)$ with a source vertex $s \in V$; assume that edges have non-negative weights. Dr. Sponge tells you that the shortest distance from s to each vertex v is $p(v)$. However, you're unsure. How can you test if Dr. Sponge is telling the true in $O(E + V)$ time? No proof is needed. Just explain how.

   Sol. The algorithm should have two components.

   (a) Checking no edge relaxations give further improvements: Relax all edges, pretending $v.d = p(v)$, which takes $O(E+V)$ time. If no vertex's distance estimate hasn't changed, then return true, otherwise false.

   (b) Checking if there is a path to each v of distance $p(v)$: Check if $p(v) = \min_{(u,v)\in E} w(u,v) + p(u)$ for all v. If so, return true, otherwise false.

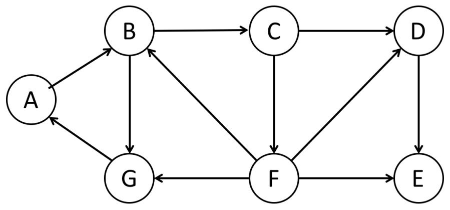
   He's telling the truth if and only if both tests return true.

   Note: The first step is worth 8 points and the second 7 points. No need to explain why. As the RT is easy to see $O(E + V)$, no need to explain the RT. But FYI, here's a quick explanation. The second step is needed. For example, say $p(v) = 0$ for all v. Then, it's easy to see that no edge relaxations give improvements. If the second test returns true, observe that there is a path from s to each vertex v of distance $p(v)$. Now the question is if $p(v)$ is indeed a shortest distance from s to v. From the analysis of BF, we know that if $v.d$ is the current best distance estimate to v, then when the first test returns true, we have found a shortest distance to every (reachable) vertex.

5. (10 points) We are given as input an undirected graph $G = (V, E)$, a source vertex $s \in V$, and an integer $k > 0$. We want to output all vertices within k hops from s. Here, we say that v is within k hops from s if there is a path from s to v consisting of at most k edges. Describe your algorithm. The faster your algorithm is, the more points you will earn.

   Sol. Run BFS. We just need to output vertices v with $v.d \leq k$. In fact, we don't have to completely run BFS. When we discover eachvertex of distance at most k, we immediately output it. And if we're about to discover a vertex of distance $k + 1$, we stop.

6. (15 points) Consider the following directed graph.

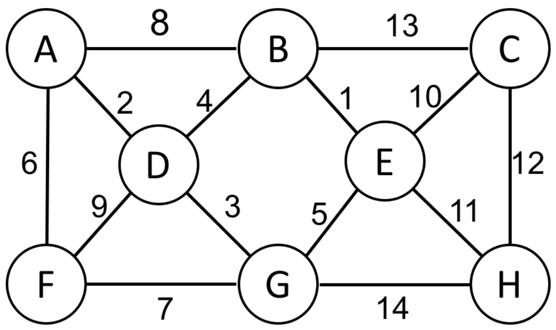


(a) (7 points) Do depth-first search in G, considering vertices in **increasing alphabetical** order. Also visit each vertex's neighbors in **increasing alphabetical** order. Show the final result, with vertices labeled with their starting (discovery) and finishing times, and edges labeled with their type (T/B/F/C); here, T, B, F, C stand for tree, back, forward and cross edges, respectively.

(b) (4 points) Based on your results, proceed to run the algorithm we learned in class to find the strongly connected components of G (show the result of the DFS in G^T, with vertices labeled with their starting (discovery) and finishing times. The DFFs must be clearly shown.)

(c) (4 points) Draw the component graph G^{SCC} of G.

7. (10 points) Consider the following weighted undirected graph.



(a) (3 points) Explain why edge (B, D) is safe. In other words, give a cut where the edge is the cheapest edge crossing the cut. **Sol.** A possible solution: $\{A, D, F, G\}$, and the others.

(b) (3 points) We would like to run the Kruskal's algorithm on this graph. List the edges appearing in the Minimum Spanning Tree (MST) in the order they are added to the MST. For simplicity, you can refer to each edge as its weight.

Sol. 1, 2, 3, 4, 6, 10, 11

(c) (4 points) We would like to run the Prim's algorithm on this graph using A as initial vertex. List the edges appearing in the Minimum Spanning Tree (MST) in the order they are added to the MST. For simplicity, you can refer to each edge as its weight. **Sol.** 2, 3, 4, 1, 6, 10, 11