CSCI 3104 Fall 2021
Instructor: Profs. Grochow and Waggoner

Quiz- Standard 3

Due Date ..... October / 1^{st} / 2021
Name ..... **Michael Ghattas**
Student ID **109200649**

Contents

1 Instructions

- The solutions **should be typed**, using proper mathematical notation. We cannot accept hand-written solutions. Here's a short intro to LaTeX.
- You should submit your work through the **class Canvas page** only. Please submit one PDF file, compiled using this LaTeX template.
- You may not need a full page for your solutions; pagebreaks are there to help Gradescope automatically find where each problem is. Even if you do not attempt every problem, please submit this document with no fewer pages than the blank template (or Gradescope has issues with it).
- You **may not collaborate with other students**. **Copying from any source is an Honor Code violation. Furthermore, all submissions must be in your own words and reflect your understanding of the material.** If there is any confusion about this policy, it is your responsibility to clarify before the due date.
- Posting to **any** service including, but not limited to Chegg, Discord, Reddit, StackExchange, etc., for help on an assignment is a violation of the Honor Code.
- You **must** virtually sign the Honor Code (see Section 2). Failure to do so will result in your assignment not being graded.

2 Honor Code (Make Sure to Virtually Sign)

Problem 1.

- My submission is in my own words and reflects my understanding of the material.
- I have not collaborated with any other person.
- I have not posted to external services including, but not limited to Chegg, Discord, Reddit, StackExchange, etc.
- I have neither copied nor provided others solutions they can copy.

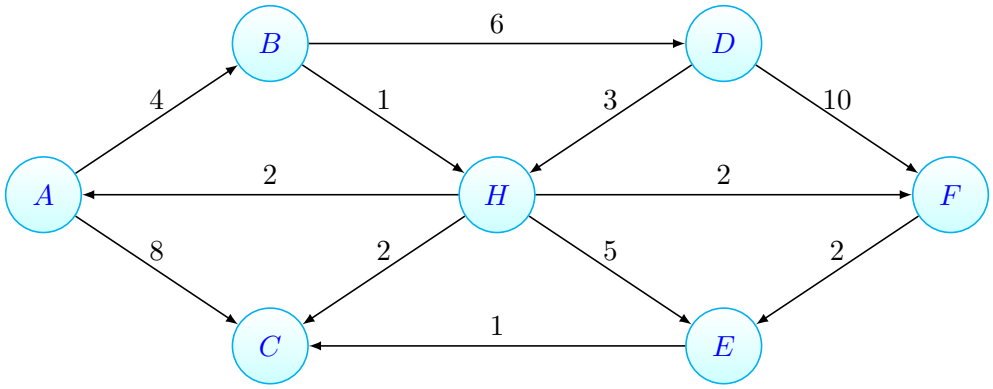
I agree to the above, Michael Ghattas. □

3 Standard Dijkstra

3.1 Problem 2

Problem 2. Consider the weighted graph $G(V, E, w)$ pictured below. Work through Dijkstra's algorithm on the following graph, using the source vertex A.

- Clearly include the contents of the priority queue, as well as the distance from A to each vertex at each iteration.
- If you use a table to store the distances, clearly label the keys according to the vertex names rather than numeric indices (i.e., `dist['B']` is more descriptive than `dist['1']`).
- You do **not** need to draw the graph at each iteration, though you are welcome to do so. [This may be helpful scratch work, which you do not need to include.]



Answer:

Below is the algorithm we will follow.

Start vertex = 0.

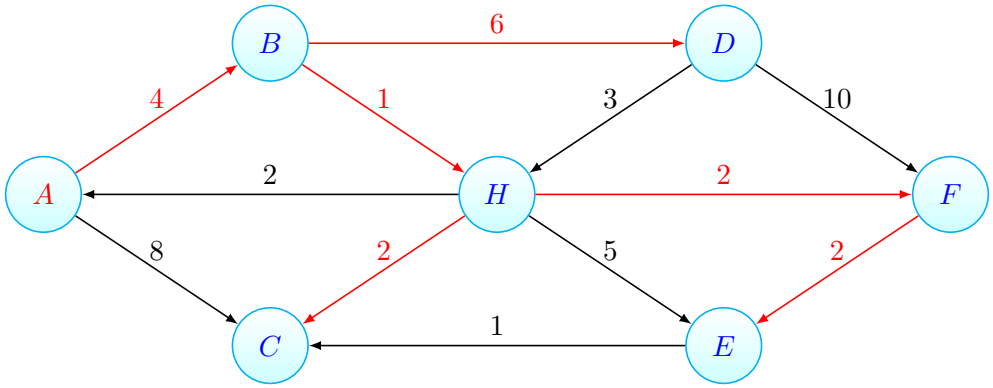
Distance to all other vertices from start vertex = ∞.

WHILE vertices remain unvisited
Visit unvisited vertex with smallest known distance from start to vertex, label it *current vertex*
FOR each unvisited neighbor of the "current vertex"
Calculate the distance from start vertex
If the calculated distance of this vertex is less than the known distance
Update the shortest distance for this vertex
Update the "previous" vertex with the current vertex
end If
NEXT unvisited neighbor
Add the current vertex to the list of visited vertices
end WHILE
END.

Below is the updated graph with the shortest path:

Table 1: **Priority Queue & Distance Weight**

	dist[**A**]	dist[**B**]	dist[**C**]	dist[**H**]	dist[**D**]	dist[**F**]	dist[**E**]	**Previous Vertex**
Initiate on $[A]$	(0)	(∞)	(∞)	(∞)	(∞)	(∞)	(∞)	**N/A**
Current $[A]$	(0)	(4)	(8)	(∞)	(∞)	(∞)	(∞)	**N/A**
Current $[B]$	(0)	(4)	(8)	(5)	(10)	(∞)	(∞)	**A**
Current $[C]$	(0)	(4)	(8)	(5)	(10)	(∞)	(∞)	**B**
Current $[H]$	(0)	(4)	(7)	(5)	(10)	(7)	(10)	**C**
Current $[D]$	(0)	(4)	(7)	(5)	(10)	(7)	(10)	**H**
Current $[F]$	(0)	(4)	(7)	(5)	(10)	(7)	(9)	**D**
Current $[E]$	(0)	(4)	(7)	(5)	(10)	(7)	(9)	**F**



□