CSCI 3104 Spring 2022
Instructor: Profs. Chen and Layer

Quiz 6 - Dijkstra's

Due Date ..... February 11
Name ..... **Alex Ojemann**
Student ID **109722375**

Contents

1 Instructions **1**

2 Honor Code (Make Sure to Virtually Sign) **2**

3 Standard 6 - Dijkstra's **3**
3.1 Problem 2 3

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.

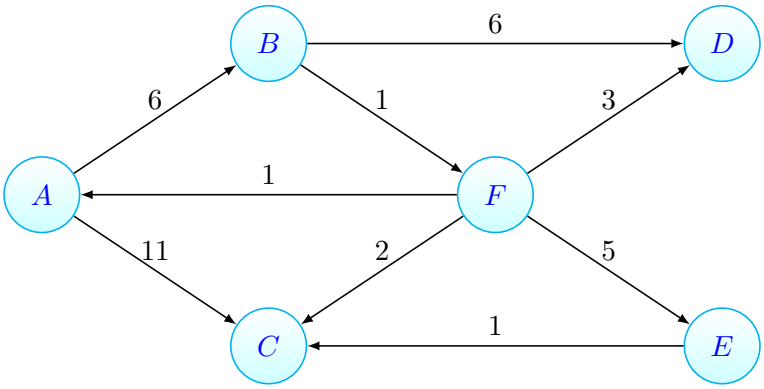
Agreed (Alex Ojemann). I agree to the above, Alex Ojemann □

3 Standard 6 - Dijkstra's

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. Q is a priority queue that holds tuples with the name of a vertex and its current shortest distance from the source vertex A.

Start: Q = []

Enqueue starting vertex A.

Q = [(A,0)]

Dequeue A and add A's unprocessed neighbors.

Q = [(B,6), (C,11)]

Dequeue B and add B's unprocessed neighbors.

Q = [(F,7), (C,11), (D,12)]

Dequeue F and add F's unprocessed neighbors. Since the paths through F to D and C have a lower total weight than their current shortest paths stored in Q, update their current shortest path lengths to those through F.

Q = [(C,9), (D,10), (E,12)]

Dequeue C. C has no neighbors so no new vertices are added.

Q = [(D,10), (E,12)]

Dequeue D. D has no neighbors so no new vertices are added.

Q = [(E,12)]

Dequeue E. E's only neighbor is C and C has already been dequeued thus it must have a shorter path from the source than that through F so no new vertices are added.

Q = []

Q is empty so the algorithm terminates.

The shortest paths to each vertex from source vertex A as returned by Dijkstra's algorithm are as follows: A:0 B:6 F:7 C:9 D:10 E:12. □