DSC 40B Discussion 9 Worksheet Solutions

Friday, May 29, 2026

Imagine you are a tutor, and you need to explain something to someone who has only a vague understanding of what's going on. Do your best to get the point across. **ALTERNATE**.

If you're not sure, ask your peer if they know how to explain it, then REPEAT the explanation back. Remember, the point of the exercise is to **vocalize** your thoughts.

If you are both unsure, make sure to come to/tutor for help! Do not use ChatGPT or any other LLM! The point of discussion is to prepare you for the exams.

Have one person in the pair open the Discussion 1 Gradescope assignment. After both you **and** your partner finish a problem **and explain it**, answer the corresponding multiple choice/short answer questions for both versions.. You will receive instant feedback on the questions, whether you are correct or not. If your answer is incorrect, please go back to the question to understand it and feel free to ask tutors about anything you are confused about. Once you have finished the entire worksheet or discussion is ending, turn in your assignment and add your partner to the submission.

Quick Icebreaker

Introduce yourself to your partner by sharing name, year, major, etc. What are your summer plans? (Do not spend more than 2 minutes!)

Questions

Question 1:

You are starting your fitness journey and decide to start with hiking trails.

First Person:

You decide that distance hiked is the most important statistic to keep track of. Due to your extensive knowledge in graph theory, you decide to model your local hiking trails as a weighted graph.

What do the nodes, edges, and edge weights in your graph represent? Do you have negative edge weights?

Nodes could be hiking landmarks, turns in the trail or trail intersections, edges are the trails themselves, and edge weights are the distance of each trail. Since the weights are distance, there would most likely not be negative edge weights.

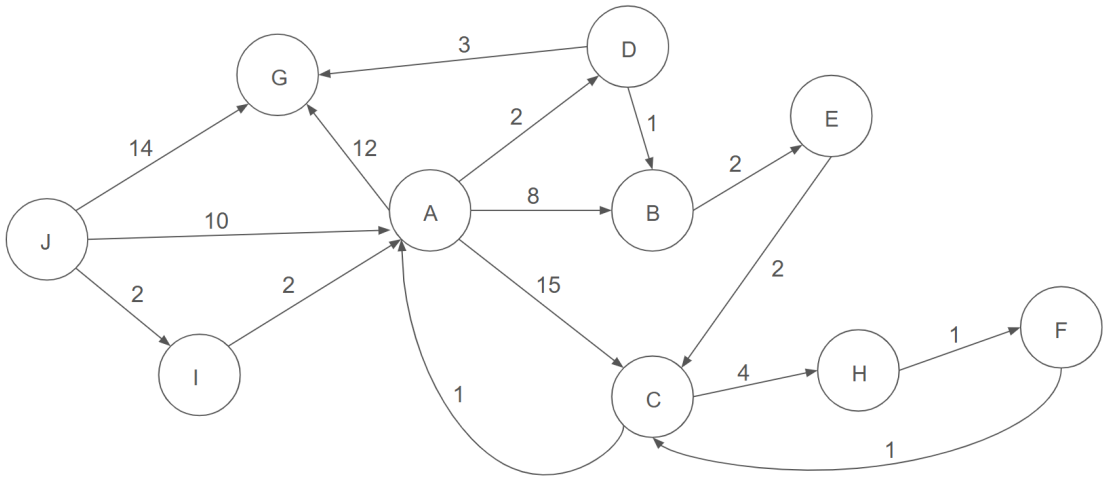
Second Person:

Your friend notes that distance is not the only factor, as elevation plays a role in how hard the hike really is. You decide to model a separate weighted graph to keep track of your elevation.

What do your nodes, edges, and edge weights represent? Do you have negative edge weights?

Nodes could be hiking landmarks, turns in the trail or trail intersections, edges are the trails themselves, and edge weights are the elevation gain or loss of each trail. Since the change in elevation can be negative or positive, there would most likely be negative edge weights.

Question 2:



Second Person:

Given the graph above, run Dijkstra's from the source of **node J**. Give the resulting distance and predecessor of each node. Use the convention that `graph.neighbors()` returns successors in alphabetical order.
Show your work and explain your reasoning.

A: 4, from I
B: 7, from D
C: 11, from E
D: 6, from A
E: 9, from B
F: 16, from H
G: 9, from D
H: 15, from C
I: 2, from J
J: 0, from None (source)

First Person:

Given the graph above, run Dijkstra's from the source of **node A**. Give the resulting distance and predecessor of each node. Use the convention that `graph.neighbors()` returns successors in alphabetical order.
Show your work and explain your reasoning.

A: 0, from None (source)
B: 3, from D
C: 7, from E
D: 2, from A
E: 5, from B
F: 12, from H
G: 5, from D
H: 11, from C
I: ∞, from None
J: ∞, from None

Question 3:

First Person:

Given the graph in Question 2, run BFS from **node A**. Give the resulting distance and predecessor of each node. Use the convention that `graph.neighbors()` returns successors in alphabetical order.

How many nodes have the incorrect distance and predecessor? Show your work and explain your reasoning.

3

A: 0, from None (source)
B: 8, from A (both incorrect)
C: 15, from A (both incorrect)
D: 2, from A
E: 10, from B (distance incorrect)
F: 20, from H (distance incorrect)
G: 12, from A (both incorrect)
H: 19, from C (distance incorrect)
I: ∞, from None
J: ∞, from None

Second Person:

Given the graph in Question 2, run BFS from **node J**. Give the resulting distance and predecessor of each node. Use the convention that `graph.neighbors()` returns successors in alphabetical order.

How many nodes have the incorrect distance and predecessor? Show your work and explain your reasoning.-

4

A: 10, from J (both incorrect)
B: 18, from A (both incorrect)
C: 25, from A (both incorrect)
D: 12, from A (distance incorrect)
E: 20, from B (distance incorrect)
F: 30, from H (distance incorrect)
G: 14, from J (both incorrect)
H: 19, from C (distance incorrect)
I: 2, from J
J: 0, from None (source)

Question 4:

Second Person:

True or False: Dijkstra’s algorithm can still return incorrect shortest paths even if there is only one negative edge and no negative cycle.

- ☑ ~~True~~
- ☐ False

Even one negative edge can make Dijkstra's finalize a node too early, then later discover a shorter path to that already-finalized node. The issue is not the number of negative edges or negative cycles; the issue is that negative edges can break Dijkstra’s “finalized means final” assumption.

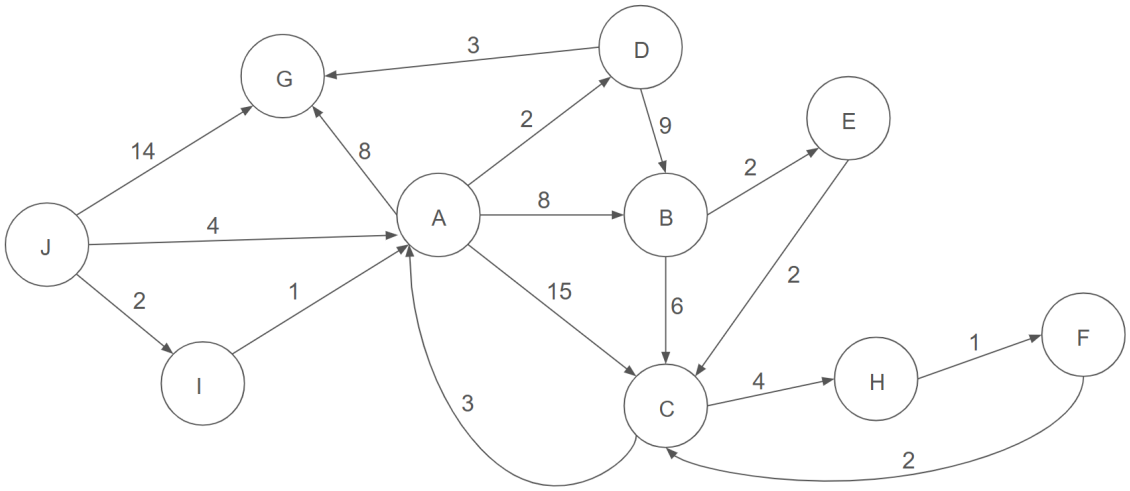
First Person:

True or False: Dijkstra’s algorithm can still return all the correct shortest paths even if there are more than one negative edge..

- ☑ ~~True~~
- ☐ False

Dijkstra's can still return correct shortest paths with multiple negative edges in some graphs, but it is not guaranteed to. Negative edges do not automatically make the output wrong every time; they just make Dijkstra's unsafe as a general algorithm.

Question 5:



First Person:

Consider the updated graph above. Running Dijkstra's with a source of **node A**, what is the most amount of times a node's distance is updated? Which node is this? Use the convention that `graph.neighbors()` returns successors in alphabetical order. Show you work and explain your reasoning.

Node C was updated 3 times. First update is from ∞ to 15 with a predecessor as A. Next update is from 15 to 14 with B as the predecessor. Finally, the last update is from 14 to 12, with E as the predecessor.

Second Person:

Consider the updated graph above. Running Dijkstra's with a source of **node J**, what is the most amount of times a node's distance is updated? Which node is this? Use the convention that `graph.neighbors()` returns successors in alphabetical order. Show you work and explain your reasoning.

Node G was updated 3 times. First update is from ∞ to 14 with a predecessor as J. Next update is from 14 to 11 with A as the predecessor. Finally, the last update is from 11 to 8, with D as the predecessor.

Question 6:

Second Person:

Given a sparsely connected graph, where each node has at most a degree of 2, what is the Big O runtime of Dijkstra's Algorithm in terms of $|V|$?

The runtime of Dijkstra's is $O(|V|\log(|V|)+|E|\log(|V|))$. Therefore, substituting $2|V|$ for $|E|$ yields a runtime of $O(|V|\log(|V|))$.

First Person:

Given a fully connected graph, where each node is connected to every other node, what is the Big O runtime of Dijkstra's Algorithm in terms of $|V|$?

The runtime of Dijkstra's is $O(|V|log(|V|)+|E|log(|V|))$. Therefore, substituting $|V|^2$ for $|E|$ yields a runtime of $O(|V|^2log(|V|))$.

Question 7:

First Person:

```
def dijkstra(graph, weights, source):
    est = {node: float('inf') for node in graph.nodes}
    est[source] = 0

    pred = {node: None for node in graph.nodes}

    priority_queue = PriorityQueue(est)

    while priority_queue:
        u = priority_queue.extract_min()

        for v in graph.neighbors(u):
            changed = update(u, v, weights, est, pred)

          # Added loop
          for x in graph.nodes:
              print("I love DSC40B!")

            if changed:
                priority_queue.change_priority(v, est[v])

    return est, pred
```

Given the modified version of Dijkstra's above, what is the new Big O runtime?

The loop is adding $|V|$ runtime to every edge check. Therefore the total runtime of the modified code is $O(|V|log(|V|)+|E||V|+|E|log(|V|))$, which simplifies to $O(|V|log(|V|)+|E||V|)$.

Second Person:

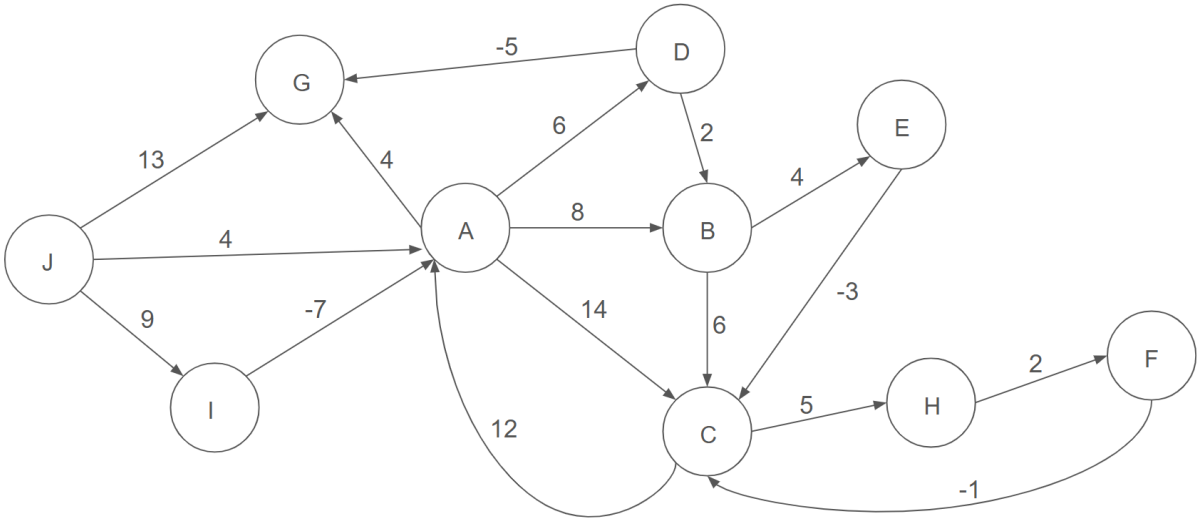
```
def dijkstra(graph, weights, source):
    est = {node: float('inf') for node in graph.nodes}
    est[source] = 0
```

```
    pred = {node: None for node in graph.nodes}

    priority_queue = PriorityQueue(est)

    while priority_queue:
        u = priority_queue.extract_min()

        # Added loop
        for x in graph.nodes:
            print("I love DSC40B!")

        for v in graph.neighbors(u):
            changed = update(u, v, weights, est, pred)
            if changed:
                priority_queue.change_priority(v, est[v])

    return est, pred
```

Given the modified version of Dijkstra's above, what is the new Big O runtime?

The loop is adding $|V|$ runtime to every time a node is popped from the priority queue. Therefore the total runtime of the modified code is $O(|V|^2+|V|\log(|V|)+|E|\log(|V|))$, which simplifies to $O(|V|^2+|E|\log(|V|))$.

Question 8:



Second Person:

Given the modified graph above with negative weights, run Dijkstra's with a source of **node J**. How many nodes have an incorrect distance assigned? Show your work and explain your reasoning.

A: 2 (correct)
B: 12, when actually 10
C: 13, when actually 11
D: 10, when actually 8
E: 16, when actually 14
F: 20, when actually 18
G: 8, when actually 3
H: 18, when actually 16
I: 9 (correct)
J: 0 (source)

FirstPerson:

Given the modified graph above with negative weights, run Dijkstra's with a source of **node A**. How many nodes have an incorrect distance assigned? Show your work and explain your reasoning.

Node G is the only node with an incorrect distance, assigned a distance of 4 when the real shortest distance is 1.

Question 9:

First Person:

Suppose we run Bellman Ford on a directed graph where every vertex has degree 4. What is the time complexity of the algorithm? (Here V is the number of vertices in the graph).

- ☐ A. O(V)
- ☐ B. O(V log V)
- ☑ ~~C. O(V^2)~~
- ☐ D. O(V^2 log V)
- ☐ E. O(V^3)

Given that the runtime of Bellman Ford is $O(|E||V|)$, substituting $4|V|$ for $|E|$ results in a runtime of $O(|V|^2)$).

Second Person:

Suppose we run Bellman Ford on a complete directed graph, i.e. every pair of distinct vertices is connected by a pair of unique edges. What is the time complexity of the algorithm? (Here V is the number of vertices in the graph).

- ☐ A. O(V)
- ☐ B. O(V log V)
- ☐ C. O(V^2)
- ☐ D. O(V^2 log V)

☑ ~~E. O(V^3)~~

Given that the runtime of Bellman Ford is O(|E||V|), substituting |E| results in a runtime of O(|V|³).

Question 10:

Second Person:

Given the graph in Question 8, run Bellman Ford with a source of **node J**. How many total times is a node's distance and predecessor updated? Do not count setting the source as a distance of 0 and all other nodes at ∞. Use the convention that `graph.edges` returns the edges in alphabetical order (A → B is first, then A → C, with J → I being last). Show your work and explain your reasoning.

Node's distance updated 22 times while predecessors updated 13 times.

First Person:

Given the graph in Question 8, run Bellman Ford with a source of **node A**. How many total times is a node's distance and predecessor updated? Do not count setting the source as a distance of 0 and all other nodes at ∞. Use the convention that `graph.edges` returns the edges in alphabetical order (A → B is first, then A → C, with J → I being last). Show your work and explain your reasoning.

Node's distance updated 11 times while predecessors updated 9 times.

Question 11:

First Person:

Given a fully connected graph with $|V|$ vertices, you run Bellman Ford with early stopping and negative cycle detection. What is the minimum and maximum number of iterations through `graph.edges` your algorithm can do, considering the fact that you do not know the order the edges are returned? Show your work and explain your reasoning.

The minimum case occurs when the first pass finds all the shortest paths, so there are no updates in the second pass, triggering an early stop. The maximum case occurs when the longest shortest path is $|V|-1$ edges long and only one edge is relaxed per iteration. The $|V|$th iteration is for negative cycle detection.

Second Person:

Given a sparsely connected graph with $|V|$ vertices, where each vertex has at most a degree of two, you run Bellman Ford with early stopping and negative cycle detection. What is the minimum and maximum number of iterations through `graph.edges` your algorithm can do, considering the fact that you do not know the order the edges are returned? Show your work and explain your reasoning.

The minimum case occurs when the first pass finds all the shortest paths, so there are no updates in the second pass, triggering an early stop. The maximum case occurs when the longest shortest path is $|V|-1$ edges long and only one edge is relaxed per iteration. The $|V|$th iteration is for negative cycle detection. There is no change between sparsely connected and fully connected best and worst case number of iterations. It is the runtime per iteration that differs between the two.