DSC 40B - Quiz 08

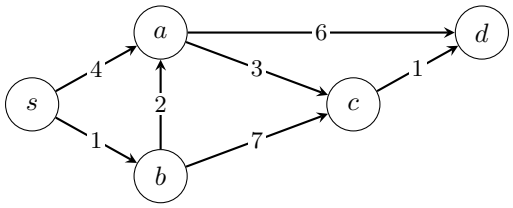
Name: Student ID:

About the quizzes:

- Quizzes in DSC 40B are *optional* and graded pass/fail.
- A score of 70% or higher earns a "pass" and 1.5 credits toward your final grade.
- If you don't pass, no credits are earned, but it doesn't hurt your grade.
- You have 30 minutes to complete the quiz.
- Unfortunately, we can't answer clarifying questions during the quiz. If you think a question has a bug or is unclear, please let us know in a private post on Campuswire after the quiz, and we'll take it into account when grading.

Problem 1.

Suppose the Bellman-Ford algorithm is run on the weighted, directed graph below using node s as the source.



Suppose `graph.edges` produces the edges in this order:

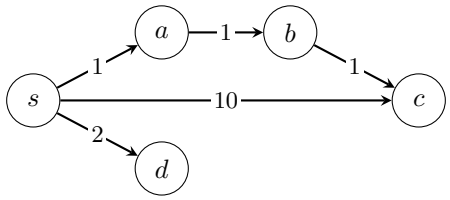
$$(a, c), (c, d), (s, a), (b, a), (s, b), (a, d), (b, c).$$

After **one** iteration of the outer loop, what is the estimated distance to node d?

◯ 7 ◯ 8 ◯ 10 ◯ ∞

Problem 2.

Suppose the Bellman-Ford algorithm is run on the weighted, directed graph below using node s as the source.

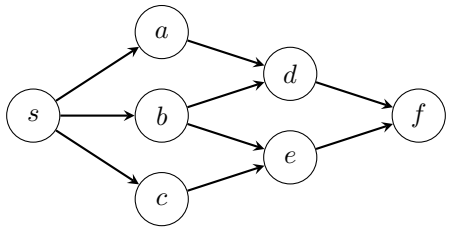


After 2 iterations of the outer loop, which one of these nodes is **not** guaranteed to have a correct estimated distance (no matter the order in which `graph.edges` produces the edges)?

◯ a ◯ b ◯ c ◯ d

Problem 3.

Suppose Bellman-Ford **with early stopping** is run on the directed graph below using node s as the source. The edge weights are not shown: they are unknown to us (but known to the algorithm), and the edges may be produced by `graph.edges` in any order.



a) What is the **fewest** number of iterations of the outer `for`-loop that can possibly be performed?

◯ 1 ◯ 2 ◯ 3 ◯ 4 ◯ 6

b) What is the **greatest** number of iterations of the outer `for`-loop that can possibly be performed?

◯ 2 ◯ 3 ◯ 4 ◯ 5 ◯ 6

Problem 4.

A **directed** graph has nodes $1, 2, \ldots, n$, and there is an edge from node u to node v whenever $u < v$. Suppose the Bellman-Ford algorithm *without* early stopping is run on this graph, using node 1 as the source. What is its time complexity, as a function of n?

◯ $\Theta(n)$ ◯ $\Theta(n^2)$ ◯ $\Theta(n^3)$ ◯ $\Theta(n^4)$

Problem 5.

The version of Bellman-Ford from lecture that detects negative cycles runs its outer loop up to $|V|$ times, with early stopping. Suppose it is run on a graph with 8 nodes, and in the **8th** iteration of the outer loop, one of the estimated distances changes. What can you conclude?

◯ The graph has a negative cycle that is reachable from the source.

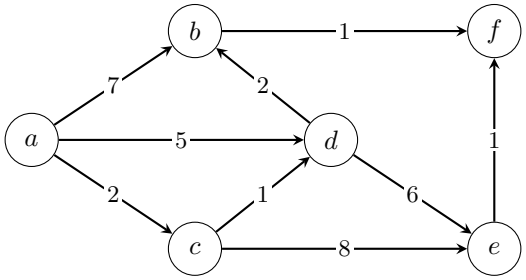
◯ The graph has no negative cycles.

◯ The graph has negative edge weights, but no negative cycles.

◯ The edges were produced in an unlucky order, and a 9th iteration would make every estimate correct.

Problem 6.

Suppose Dijkstra's algorithm is run on the weighted, directed graph below using node a as the source.

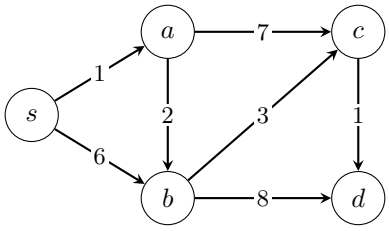


What will be the **fifth** node popped from the priority queue? (The first node popped is the source, a.)

◯ b ◯ c ◯ d ◯ e ◯ f

Problem 7.

Suppose Dijkstra's algorithm, as written in lecture, is run on the weighted, directed graph below using node s as the source.

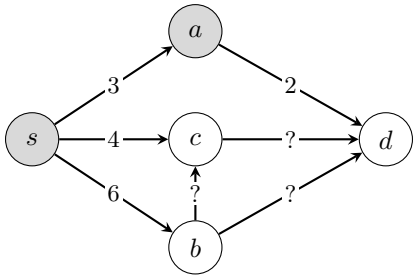


When the algorithm finishes, what is d's predecessor (that is, `pred['d']`)?

◯ s ◯ a ◯ b ◯ c

Problem 8.

Dijkstra's algorithm is run on the directed graph shown below using node s as the source. The weights marked "?" are unknown to us (but known to the algorithm), and are assumed to be **positive**. The algorithm has been run partway: the shaded nodes, s and a, have been popped from the priority queue, and the edges out of them have been updated.



a) At this point, the estimated distances of the other nodes are $b : 6$, $c : 4$, and $d : 5$. Which one of these statements must be true, whatever the unknown weights are?

- ◯ c's estimated distance is its actual distance from s.
- ◯ d's estimated distance is its actual distance from s.
- ◯ b's estimated distance is greater than its actual distance from s.
- ◯ The next node popped from the priority queue is d.

b) True or False: it is possible that the shortest path from s to d goes through b, depending on what the unknown weights are.

◯ True ◯ False

Problem 9.

Below is the priority queue implementation of Dijkstra's algorithm from lecture.

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

In lecture, the priority queue was a binary min-heap. Suppose your friend claims to have invented a new data structure for priority queues, where `PriorityQueue(est)` takes $\Theta(V)$ time, and both `.extract_min()` and `.change_priority()` take $\Theta(1)$ time.[1] If the claim were true, what would be the time complexity of the code above when it uses your friend's priority queue?

◯ $\Theta(V)$ ◯ $\Theta(E)$ ◯ $\Theta(V + E)$ ◯ $\Theta(V \cdot E)$

[1] Your friend, unfortunately, must have made a mistake: no such priority queue can possibly exist. With it, we could sort n numbers in $\Theta(n)$ time: put them in the priority queue, then call `.extract_min()` n times. But we know that no comparison-based algorithm can sort faster than $\Theta(n \log n)$.