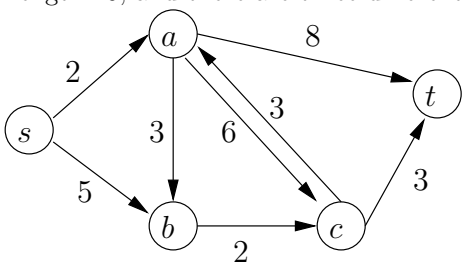
Homework Assignment #9
Due: April 3, 2023 at 7:00 p.m.

1. Given a directed graph with positive weights on the edges, we wish to find the *number* of different shortest paths from one node s to another node t. For example, in the following graph, the shortest path from s to t has length 10, and there are three different shortest paths from s to t.



We adapt Dijkstra's algorithm to accomplish this. Just as we keep track of the shortest path we have found so far from s to each node v in $d(v)$, we will also keep track of the *number* of such paths in $num(v)$.

```
1:  initialize an empty priority queue Q
2:  insert each node v into Q with priority d(v) = ∞
3:  change the priority d(s) to 0
4:  num(s) ← _____
5:  T ← ∅                          ▷ T stores set of elements that have been removed from Q
6:  loop
7:      ▷ Invariant: ∀v, d(v) is the length of the shortest path from s to v containing only nodes in T ∪ {v}
8:      ▷ (or ∞ if there no such path) and if d(v) < ∞ then num(v) is the number of different paths of
9:      ▷ length d(v) from s to v containing only nodes from T ∪ {v}.
10:     exit when all elements in Q have priority ∞
11:     u ← extract minimum element from Q
12:     add u to T
13:     for each node x such that (u, x) is an edge in the graph do
14:         if d(u) + weight(u, x) < d(x) then
15:             _____
16:         else if _____ then
17:             _____
18:         end if
19:     end for
20: end loop
21: _____                          ▷ Fill in code to output the number of shortest paths from s to t
```

[6] **(a)** Fill in the blanks to maintain the loop invariant. (You can replace blanks with multiple statements.)

You do not have to give a formal proof of correctness, but you should briefly explain your reasoning when you designed the code to maintain the loop invariant and to produce the correct output.

[2] **(b)** Fill in the a table showing the values stored in d and num after each iteration of the loop for the example graph shown above. There should be one row in your table for each iteration.

After iteration	d values					num values				
	s	a	b	c	t	s	a	b	c	t

[1] **(c)** For what types of input graphs would the loop exit before the priority queue Q is empty?

[1] **(d)** Use Θ notation to describe the worst-case running time of your algorithm if the input graph has n nodes and m edges. What implementation of the priority queue Q are you assuming for your bound?

[2] **(e)** Does the algorithm work if edge weights may also be 0? If so, explain why. If not, give an example of a graph where the algorithm computes an incorrect answer.