Angel Avlear-Bonilla

4/21/21

Data Structures

PSET 5

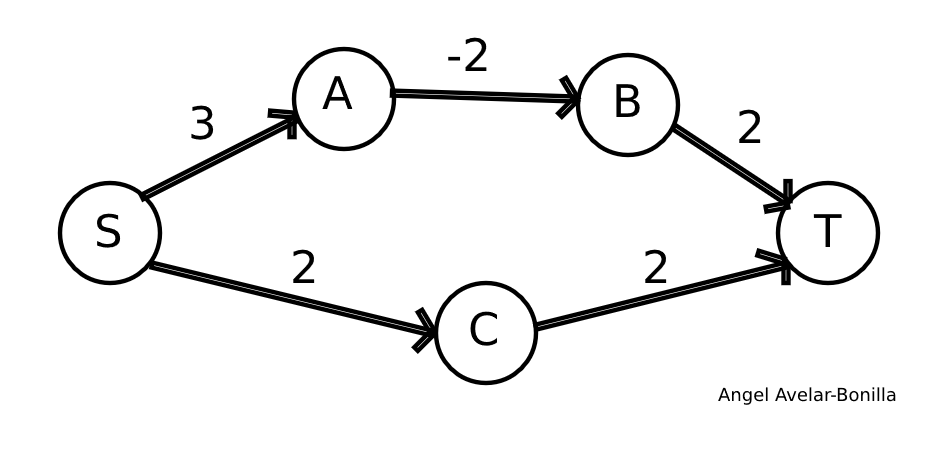
1a) **FALSE**

**In the below graph there are two paths from S -> T**

**S -> A -> B -> T = 3**

**S - > C -> T = 4**

**Here the shortest path is the top one with a length of 3.**

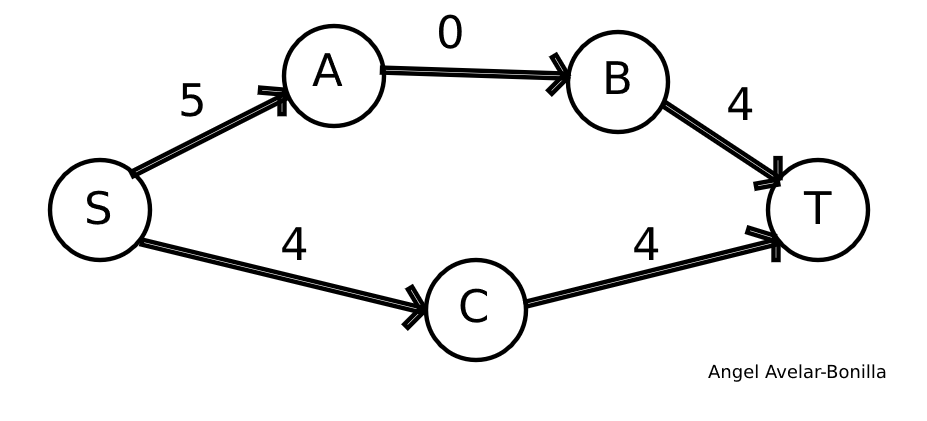


**If we add 2 so that all the edge weights are nonnegative, we get the below graph.**

**Here the path lengths are as follows:**

**S -> A -> B -> T = 9**

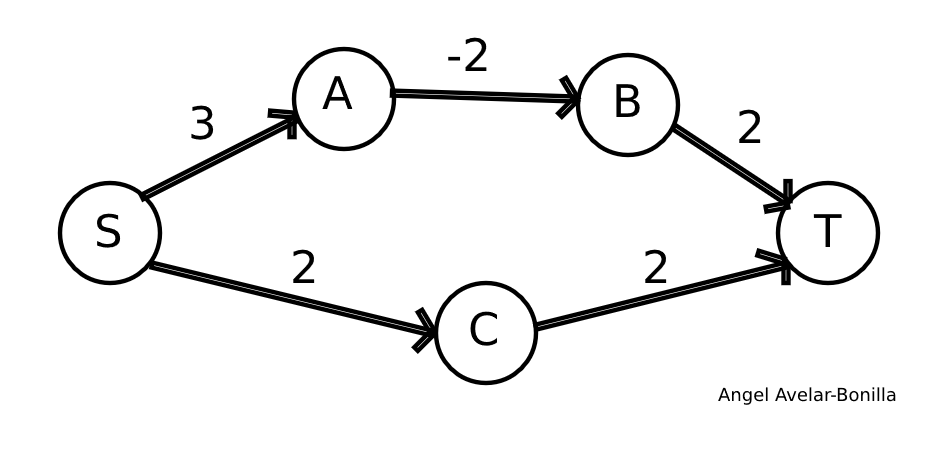
**S - > C -> T = 8**



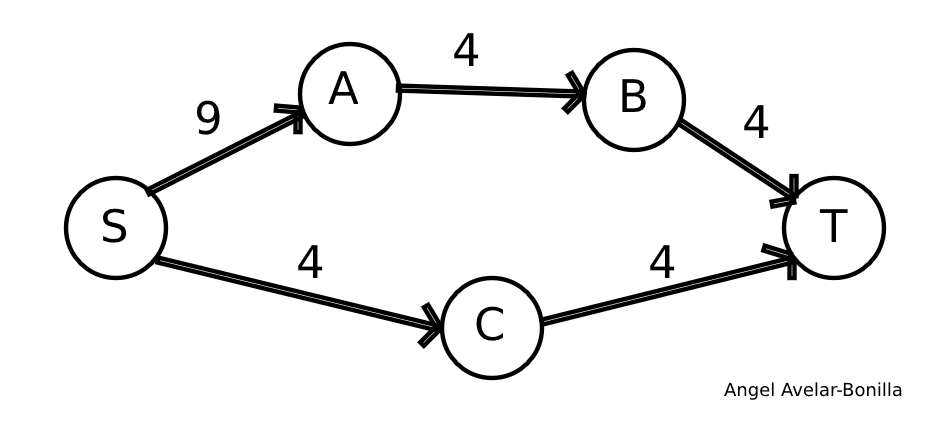
**As you can see the shortest path changes to the bottom one when we add a constant weight. Therefore, the answer is False.**

**1b) FALSE**

**In the below graph our top path is our shortest path P with a length of 3**



**If we square the edge weights, we get the below graph:**



**As you can see now the shortest path is S -> C - > T with an edge weight of 8.**

**So the answer is False.**

**1c) TRUE**

**Bellman-Ford requires that a graph have no negative cycles, if we are negating all our weights, we should also negate this requirement. Therefore, our graph should have no positive cycles. Since we are told that directed Graph G has no nonnegative edge weights it meets the requirement of no negative cycles, so negating all the edge weights would mean the graph has no positive cycles. Thereby still meeting the requirements for the Bellman-Ford algorithm.**

**2a) If there is a shortest path from s to v that ends at (u, v) then (u, v) MUST be on any shortest path to v since it too is also a shortest path.**

**If we assume that (u, v) is downwards critical but NOT on the shortest path, then (the distance from s to v) is less than (the distance from s to u) + (the distance from u, v) since (u, v) is not on the shortest path. If we decrease w(u , v) then we will only be changing the cost of (the distance from s to u) + (the distance from u, v) and not the cost of (the distance from s to v) thereby contradicting our claim that (u, v) is downwards critical.**

**If (u, v) is on the shortest path, then decreasing w(u, v) decreases the cost of this path. Since (u, v) is already on the shortest path we know that no other path is shorter, so it remains the shortest path. Since we decreased the cost of the path, (u, v) is downward critical.**

**Thus, when (u, v) is downwards critical, it MUST be on the shortest path. And when (u, v) is on the shortest path, it MUST be downward critical. QED**

**2b) We are claiming that an edge (u, v) is upwards critical if and only if there is a shortest path from s to v that ends at (u, v).**

**Like in the previous claim, if there is a shortest path from s to v that ends at (u, v) then (u, v) MUST be on any shortest path to v since it too is also a shortest path.**

**If we assume that (u, v) is upwards critical but NOT on the shortest path, then increasing w(u, v) would not affect our shortest path weight since it is not on the shortest path. This is a contradiction since we said (u, v) is upwards critical. Therefore if (u, v) is upwards critical then (u, v) must be on the shortest path.**

**If every shortest path includes (u ,v) then increasing its weight would affect every shortest path, thereby making it upwards critical.**

**Thus, when (u, v) is upwards critical, it must be on the shortest path. And when (u, v) is on the shortest path, it MUST be upwards critical. QED**

**2c) Using Djikstra’s: 🡪 Time O(E log V) using the min-heap method from class**

**d[] has shortest distances to all our edges**

**p[] has our parent nodes**

**Iterate through all edges (u, v) 🡪Time O(E) since we are just going thru edges**

**Check if w(u, v) + d(u) == d(v)**

**If yes then add (u, v) to our downwards critical edges. We can do this because if the above check passes, we know (u, v) is on the shortest path thus making it downward critical.**

**dc\_count[v]++ this stores the number of critical edges going into v, we increment whenever we find a critical edge going in**

**Iterate thru all vertices ‘v’ 🡪Time O(V) since we are just going thru edges**

**Check If dc\_count[v] == 1**

**If yes, add (p(v), v) to our upwards critical edge. We can do this since we know that v has only one incoming downwards critical edge so that edge must be on the shortest path from s to v.**

**O((E log V) + E + V) == O(E log V)**