I certify that every answer in this assignment is the result of my own work; that I have neither copied off the Internet nor from any one else's work; and I have not shared my answers or attempts at answers with anyone else

I would also like to use one of my late days for this assignment

---

1:

A weighted graph G is defined by the following (weighted) adjacency list:

node	*<(neighbor, weight)>*
u	$< (v,15), (w,4) >$
v	$< (y,3) >$
w	$< (v,9), (x,6), (y,4) >$
x	$< (v,2) >$
y	$< (x,3) >$

Trace the execution of DIJKSTRA-SSSP(G, u) by filling out the following tables for d and Π. Each table has one row per node.

In each column, show the value of $d[\ ]$ and $\Pi[\ ]$ respectively for that node at the beginning of the while-loop.

Circle the d-value for the node about to be extracted.

Indicate a node that is no longer in Q with a — (dash).

Node	$d[\,]$	$d[\,]$	...
u			
v			
w			
x			
y			

Node	$\Pi[\,]$	$\Pi[\,]$	...
u			
v			
w			
x			
y			

From the above tables, how would you find the shortest path from u to v?

Node	d[]	d[]	d[]	d[]	d[]	d[]
u	0	—	—	—	—	—
v	∞	15	13	13	**12**	—
w	∞	**4**	—	—	—	—
x	∞	∞	10	**10**	—	—
y	∞	∞	8	—	—	—

Node	Π[]	Π[]	Π[]	Π[]	Π[]	Π[]
u	—	—	—	—	—	—
v	—	u	w	w	**x**	—
w	—	**u**	—	—	—	—
x	—	—	w	**w**	—	—
y	—	—	w	—	—	—

I have indicated the path on both tables in bold font (instead of circling)
To determine the shortest path from u to v we must look at the Π[] table, as it determines the predecessor of the node based on the shortest path. We can start from node v, choose the last node indicated to the far right in v's row, and repeat our way backwards to u. Then reversing these results will give us our shortest path:

$$u \to w \to x \to v$$

2:
Prove that in Dijkstra's algorithm, once a vertex is returned by EXTRACTMIN (Line 7), no vertex will ever be returned in future iterations with a smaller $d[\,]$ value even though some $d[\,]$ values decrease in the relaxation step (Line 11).
(Line numbers are from slide 25/46 in graphs3.pdf.)

In Dijkstra's algorithm, EXTRACTMIN(Q) is first called, which gives us the node with the lowest-weight distance from Q.
Let's say that EXTRACTMIN(Q) gives us a node x with a weight of 0. The next iteration will call EXTRACTMIN(Q) on the adjacent nodes to x, so all adjacent nodes to x must a weight greater than 0.
This continues and means that all future call to EXTRACTMIN(Q) will return a node with a weight greater than u. The relaxation step will update the distances on paths to the adjacent nodes to the node returned from EXTRACTMIN(Q), however this does not affect that the weights of a returning node will always be greater than the weight of the previous path.

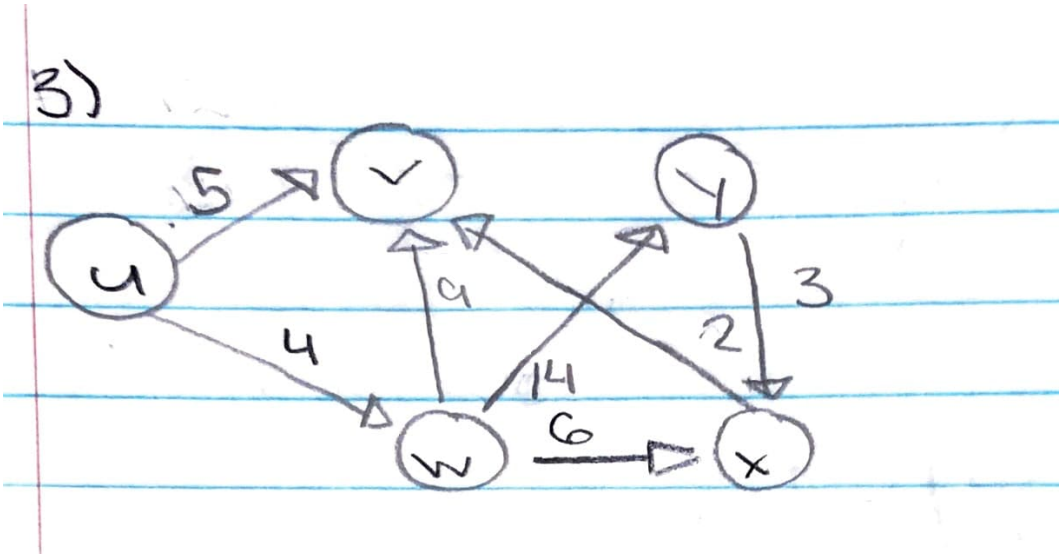
3:
Professor Flippy claims to have found an algorithm for the SSLSP (single-source longest simple path) problem. That algorithm modifies Dijkstra's algorithm (for SSSP) by flipping the sign in the initialization step (from positive infinity to negative infinity in Line 2), replacing $>$ by $<$ in the relaxation step (Line 10), and changing EXTRACTMIN to EXTRACTMAX (Line 7).
(Implementation: replace min-heap by max-heap.)

Show that the professor is incorrect by providing a counter-example: draw a graph and show the execution of the algorithm in each iteration using a table (as in the first question of this homework) and point out the error.

Note: A *simple* path means is one in which no vertex is repeated, i.e., the path doesn't traverse a loop.

(Line numbers are from slide 25/46 in graphs3.pdf.)

Let's consider the same problem from problem 1, but alter the weights of (u,v) $\to$ 5 and (w,y) $\to$ '4. Here is a rough graph of what that looks like:



Node	d[]	d[]	d[]	d[]	d[]	d[]
u	0	—	—	—	—	—
v	-∞	5	—	—	—	—
w	-∞	4	4	4	4	—
x	-∞	-∞	-∞	11	—	—
y	-∞	-∞	8	-∞	—	—

Node	Π[]	Π[]	Π[]	Π[]	Π[]	Π[]
u	—	—	—	—	—	—
v	—	u	—	—	—	—
w	—	u	u	u	u	—
x	—	—	—	y	—	—
y	—	—	v	—	—	—

From this data, the algorithm determines that the Longest Simple Path from u to v is 5 i.e. $< u, v > \rightarrow 5$. But the actual Longest Simple Path should be $u \rightarrow w \rightarrow y \rightarrow x \rightarrow v$ which is 23. This algorithm extracts v before w is processed because the initial values of d[v] and d[w], $5 > 4$ or $d[v] > d[w]$, so w has no outbound edges to calculate a Longest Simple Path for. So this is where the algorithm fails.