Question 1:

Given a graph $G(u,v)$ that is undirected, we can find the number shortest path from u to v with a modified version of BFS.

Our algorithm starts at vertex $u \in V$, and our task is to find the shortest paths to $v \in V$. For each vertex, there will be a weight counter and a path counter. The weight is the number of steps required to reach that node so far, and the path will be updated depending on the number of paths to get there.

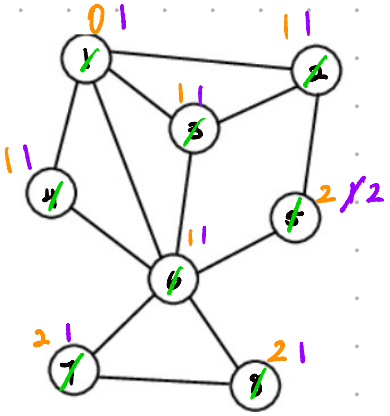
We call BFS at vertex u, all adjacent nodes to u will be initiated with their path counter to 1, and their weight counter to 1 (or the weight of the edge if it is weighted), BFS then does the same with each of the adjacent nodes to u's adjacent nodes, and so on...

Weight counter of a vertex is determined by the min of its adjacent nodes weight. Path counter is updated accordingly

① if another adjacent node also reaches me with the current weight I have, add its path counter to my path counter.
② if weight is less, change my current weight and set path back to 0
③ greater, just ignore

This algorithm terminates when BFS is complete. To find how many shortest paths exist from u to v, simply call v's path counter.

An example can walkthrough the algorithm:



given the following unweighted graph, we want to find the total number of shortest paths from 1 to 8

1. mark node 1's weight with 0, as we don't need to traverse at all to get to ourself and mark path to 1, as there is only one path to ourself
2. mark adjacent nodes weight, which is one more than before, there's also one path to them. Node 3 has two adjacent nodes that leads to it, however

cont...
there is only one path with weight 1. Same thing applies to node 6.

3. continue traversing other adjacent nodes of 4, 6, 3, 2
   node 4 has no adjacent nodes with greater weight
   node 6 can visit 5, 7, 8
   - mark their weight and path counter
   node 3 has no adjacent nodes with greater weight
   node 2 has a greater weight in 2, approaches 5 with the same weight
   - update path counter

4. our final node 5 has no adjacent nodes with a greater weight

5. we have marked the entire graph with its path and weight counter, now we can access to path counter for 8, and see that there exists only one shortest path

Question 2:

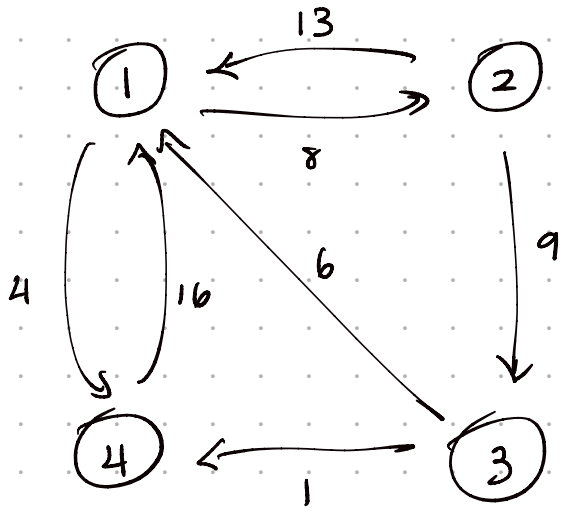
For this problem, we will work with matrices of vertices V. To find the shortest cycle in a direct, positive weighted graph, we will create |V| matrices and update the value of the shortest path between i, j in each new matrix.

The first matrix will be an n x n matrix that shows all the vertices that are connected with their weight at A[i, j], the value for A[i, i] will be set to infinity and will decrease by each new matrix.

The plan is to create n new matrices, and in each new matrix, the row K and column K will be copied from our previous matrix K-1, the matrix values at A[i, i] will be copied and edited in our current matrix K as well. When K=n, we can look at $A^K[i, i]$, and the shortest length cycle will be min $(A^{K=n}[0,0], A^{K=n}[1,1], \ldots A^{K=n}[n,n])$

Given following directed and positive weighted edges graph G, we can give the following example:

G =



Our initial matrix will be

$A^0 =$

	1	2	3	4
1	∞	8	∞	4
2	13	∞	9	∞
3	6	∞	∞	1
4	16	∞	∞	∞

A[i, i] = ∞ &&
if there is no edge between A[i, j], then A[i, j] = ∞

we can now build A'

	1	2	3	4
1	∞	8	∞	4
2	13	21	9	17
3	6	14	∞	1
4	16	24	∞	20

NOTE: copy A[K-1, j] and A[i, K-1] over

NOTE: calculate $A^K[i,j] = \min(A^{K-1}[i,j], A^{K-1}[i,k] + A^{K-1}[k,j])$

now let's build A^2

	1	2	3	4
1	21	8	17	4
2	13	21	9	17
3	6	14	23	1
4	16	24	33	20

now we copy row 2 and column 2 from A^1 and calculate missing values using the same formula

now we can build A^3

	1	2	3	4
1	21	8	17	4
2	13	21	9	10
3	6	14	23	1
4	16	24	33	20

copy r3 and c3 from A^2 and calculate others...

NOTE: we are still comparing all values $A[i,j]$ with formula

$$A^k[i,j] = \min\left(A^{k-1}[i,j], A^{k-1}[i,k], A^{k-1}[k,j]\right)$$

values are updated accordingly

finally at $k=n=4$, we build our last matrix

	1	2	3	4
1	20	8	17	4
2	13	21	9	10
3	6	14	23	1
4	16	24	33	20

do the same as before

now we have completed the graph
Assuming my calculations were correct...

we can now look at $A[i,i]$ for $i=1,\ldots,n$ and see that the shortest cycle exists at $A[1,1]$ with the shortest length of 20.

This algorithm achieves $O(|V|^3)$ complexity since looping through a $n \times n$ matrix takes $O(|V|^2)$ time, and we must loop n times until $k=n$, totaling $O(|V|^3)$.

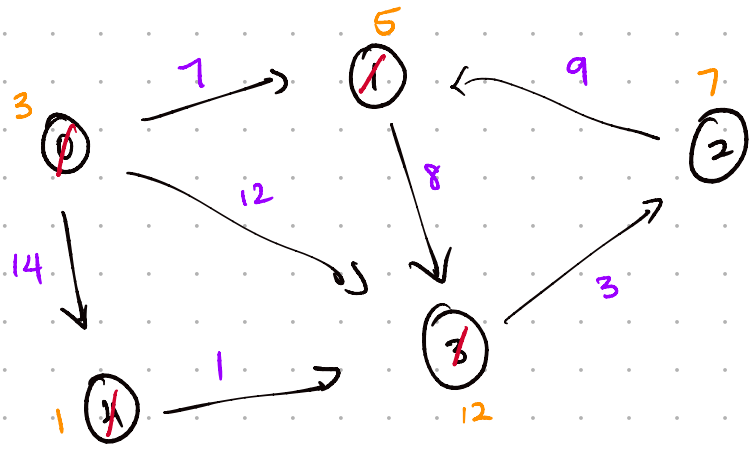
Additional, we loop through $A[i,i]$ at the end to find the min value, so this will take $O(|V|)$ as well. If all the diagonal values are infinity, then there exist no cycles and the graph is acyclic.

More accurate time complexity is $O(|V|^3 + |V|)$, but we can simply to $O(|V|^3)$

Question 3:

We can implement a BFS so that each node has a weight counter that tracks the current weight it takes to reach the graph G. If a node is already visited, we compare our current paths weight vs. the currently established weight, if our weight is smaller, the algorithm will update the node's weight counter. This algorithm will continue until all nodes have been visited

Consider following graph G (V, E) as an example



orange = node weights
purple = edge weights

In the beginning, all node weights are null and all node visited status = false

assume we have source node at 0, by BFS traversal, we visit

0 -> 1 : 15
0 -> 3 : 27
0 -> 4 : 18

} node 0 is completely visited, so we can cross it out

now we visit node 1

1 -> 3 : 35 => 3 already has a weight counter of 27, so we don't need to change anything

node 1 is completely visited

node 3

3 -> 2 : 37 node 3 is completely visited

node 4

4 -> 3 : 31 no change

node 4 completely visited

visit node 2 from node 3

2 -> 1 : 05 no change

node 2 completely visited

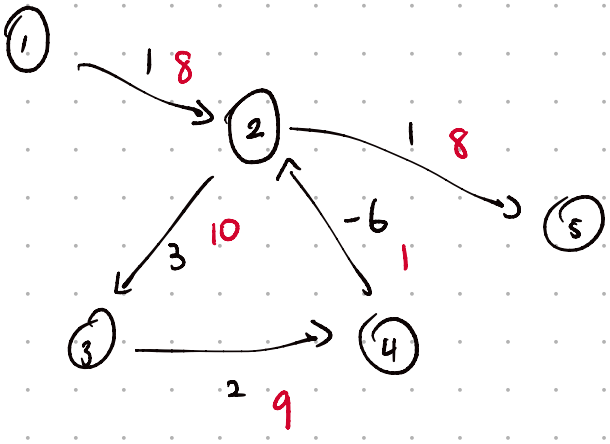
all nodes are completely visited

0 -> 1 : 15
0 -> 2 : 37
0 -> 3 : 27
0 -> 4 : 18

completing our algorithm

Question 4:

This is a valid solution <u>assuming</u> there are no negative cycles. Since all edges increase proportionally, it would be fine. However, in a graph with negative cycles like the following



- the following graph has a negative cycle at 2, 3, 4
- introduce a constant C=7 to increase all edge weights by 7
- we can clearly tell that the shortest path is 1 → 2 → 5 with these nodes

when looking back to our original values, this approach is incorrect. Since there is a negative cycle at 2->3->4, netting a total of -1, the shortest path would be to continue cycling the negative cycle, giving us a shortest path of

$$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow \dots \rightarrow 5$$

where each time we run through the cycle, our path weight decreases by 1. Our shortest path in the original graph weighs $-\infty$, whereas by adding a constant does not yield us the same path. Therefore we cannot run Dijkstra's on negative graphs by adding a constant.