Question 2:

\*Important Note About Code:

Due to issues with printing all values of philosophers and forks at the table at once, I’ve made the implementation such that only the philosophers that are eating are known. There are commented out code lines mentioned in the README that can provide you with an idea of the entire table state, but they are buggy, so I opted to only use what was fully functional.

As mentioned in the README:

//print\_state(); in Line 136 may be uncommented to determine which philosophers are EATING or HUNGRY

//print\_one\_state(phil\_num); in Line 120 may be uncommented to determine which philosophers are EATING or HUNGRY as well

//print\_state(); in Line 153 may be uncommented, as it shows when a philosopher is giving up their forks if new philosophers acquire them and begin eating, printing the states of all philosophers.

a) Only 3 forks are placed, but each philosopher still needs 2?

In the case where 5 philosophers are placed at a round table, and there are only 3 forks in the center of the table, for anyone to use, only one philosopher may eat at a time. This means that the remaining philosophers will be blocked while they wait for another fork. Initially, it is possible that all three forks will be initially taken by different philosophers, but since each philosopher needs to think, if there isn’t a second fork available, they will put down their one fork. In the time that they have their one fork, they will check if another one is available (one has been put down), and they will pick it up and begin eating. This situation increases starvation and contention for forks, reducing the frequency of philosophers that eat as there are less forks available amongst the philosophers.

b) Give one philosopher priority over others?

If one philosopher were to take priority over the others, it would mean they can pick up forks and begin eating before their adjacent philosophers. This problem would be remedied for the rest of the philosophers, as the philosopher with priority will pick up their forks, leaving only one fork option on the opposite side of the adjacent philosophers, and so on. It is necessary to implement fairness mechanisms to prevent the starvation of other philosophers in this case, where one philosopher may almost constantly eat while others are unable to find 2 forks.

c) Change which fork is acquired first on each pair of requests?

When the philosophers select a fork, there is a chance that they all select forks on one side. This introduces the possibility of deadlock as each philosopher would be waiting for one fork while possessing another.

If this logic were in place, philosophers would eat more often, as if one fork is picked up and the adjacent philosopher puts down there’s, someone can eat. This is possible if the philosophers can change which fork they acquire on each pair of requests. For example, if all odd-indexed philosophers pick up the right fork, and all even-indexed philosophers pick up the left fork, the circular wait that introduces deadlock and starvation is avoided.

About Edgar Dijkstra & the Philosophers Problem:

Edgar Dijkstra was a Dutch computer scientist and mathematician. He initially studied theoretical physics but chose programming. He’s the recipient of a Turing award for his advocation of structured programming, with a clear control flow from start to finish. He also developed the first compiler for ALGOL-60, and an old programming language. Dijkstra is well known for several reasons, but primarily his algorithm determines the shortest path between nodes in a weighted graph. He also taught at UT Austin for many years prior to his death.

The dining philosopher's problem was originally developed by Dijkstra as well and introduces important concepts in program concurrency and synchronization. A group of philosophers in a circle shared a limited number of forks and needed to avoid deadlock and starvation simultaneously while ensuring a fork isn’t grabbed twice. This problem is relevant in the real world in resource allocation for management of operating systems, networks, and other computer applications.

Dijkstra’s algorithm, mentioned above, is commonly used in network routing protocols and in services like Google Maps’ directions. In, for example, Google Map’s, there’s many paths available to reach an end destination from a start, but it requires an algorithm like Dijkstra’s to determine the minimum distance on different paths. Social networking may also use Dijkstra’s as well, where recommendations for what to watch or who to follow are given by the browsing of users that interact with each other often.

Sources:

<https://bradfieldcs.com/algos/graphs/dijkstras-algorithm/>

<https://www.researchgate.net/publication/373121211_Application_of_Dijkstra's_Algorithm_to_Determine_the_Shortest_Route_from_City_Center_to_Medan_City_Tourist_Attractions>

<https://runestone.academy/ns/books/published/pythonds/Graphs/DijkstrasAlgorithm.html>