Problem 1  You are given a distributed system which is represented by an undirected graph in which the vertices represent agents (also called processes) and the undirected edges represent channels. Thus if \( i \) and \( j \) are neighbors in the graph then there is a message channel from \( i \) to \( j \), and another message channel from \( j \) to \( i \). The graph is strongly connected, i.e., there exists a path from every vertex to every other vertex.

(a) Develop an algorithm initiated by some agent, say agent 0, by which the agent discovers the number of vertices in the graph. Make sure you write out the program for both the agent 0 (which is not included in Sivilotti) and other agents in the UNITY format as in Section 6.5 of Sivilotti.

(b) Prove that your algorithm terminates execution eventually. If you use an algorithm that is proved in the book you don’t need to give the proof in your homework. For example, you don’t need to prove that a diffusing computation terminates.

(c) Prove that when it terminates, agent 0 has the correct counts. For this purpose it is helpful to define an invariant. You should provide similar level of rigor to Sivilotti 6.6.1.

Problem 2  In this problem, we will go into more details on Lamport’s algorithm for mutual exclusion, described in Section 7.5 of Sivilotti.

(a) In Lamport’s mutual exclusion algorithm, it is required that agent \( P_i \) has received messages from all other agents with timestamps larger than requested access time \( t_i \) for \( P_i \)’s request to enter the critical section. Give an example showing why this property is required to guarantee mutual exclusion.

(b) Show that for Lamport’s mutual exclusion algorithm if an agent \( P_i \) is executing the critical section, then \( P_i \)’s request need not be at the top of the request queue for another agent \( P_j \). Is this still true when there are no messages in transit? Justify your answer.

Problem 3  [Periodic Updating for Mutual Exclusion] Consider the following proposal for mutual exclusion which is a variant of the algorithm given in the book. In this problem, each agent has a local clock, which may not be synchronized (say when agent \( P \)’s clock reads 2, the clock of \( Q \) may read 3). Moreover, the only thing such clocks can do is to tick forward and the agents do NOT update the clocks based on message exchange as in Lamport’s logical clock. You can assume the clock ticks are accurate in the sense that it takes exactly the same amount of time for any clock to tick ahead by 1 second. We assume the agents only transit states when the clock reads as integers, but the delivery of messages can take an arbitrary floating number time.
When an agent transits from NC (not critical) to TRY (waiting to enter critical section) it broadcasts a request message to all agents where the request message has a field that is the value of the agent’s clock at the instant when the agent transits from NC to TRY. An agent sends exactly one request message each time it transits from NC to TRY. When an agent transits from CS (inside critical section) to NC it sends a release message with a field containing its clock value at the instant that it executed the transition.

Every agent, regardless of its state, periodically broadcasts a *my-time* message with a field containing the value of the agent’s clock at the instant the message is broadcast. The period is some positive integer \( \tau > 1 \). Thus an agent broadcasts a my-time message when the agent’s clock reads \( n \cdot \tau \) for \( n = 0, 1, 2, \ldots \). Each agent maintains a list \( L \) of request messages that it has received. When an agent \( P \) receives a release message from an agent \( Q \) it removes \( Q \)’s request messages (if any) from \( L \). We assume the agents do not send out request or release message when the local clock is at \( n \cdot \tau \) for \( n = 0, 1, 2, \ldots \).

Each agent also maintains an array \( \text{REMOTE\_TIMES} \) where for an agent \( Q \), \( \text{REMOTE\_TIMES}[Q] \) is the latest time field in any message (request, release or my-time) that it has received from \( Q \). An agent \( P \) transits from TRY to CS (in critical section) if and only if the agent’s last request message has a time field, \( z \), that is:

(a) earlier than the time fields of all the request messages that it has in \( L \)

(b) for every agent \( Q \) (other than \( P \) itself), \( z < \text{REMOTE\_TIMES}[Q] \)

Ties for clock values are broken by assuming that the agent with the smaller id is earlier. Assume that every agent has an integer id. Also assume all message deliveries are reliable and FIFO.

Is the algorithm correct? Prove or give a counterexample. If you want to prove the algorithm is correct, you need to prove the mutual exclusion as safety property, and prove that each agent that requested access to the critical section will eventually be granted. Make your proof clear and succinct. A similar level to the correctness proof for Lamport’s algorithm in Sivilotti is enough. If you decide to give a counterexample, please be specific about the example in terms of which actions lead the system to a state where the algorithm is no longer correct (a diagram based argument is NOT sufficient unless it is the graph representation of a specific program as in Sivilotti Section 2.5).