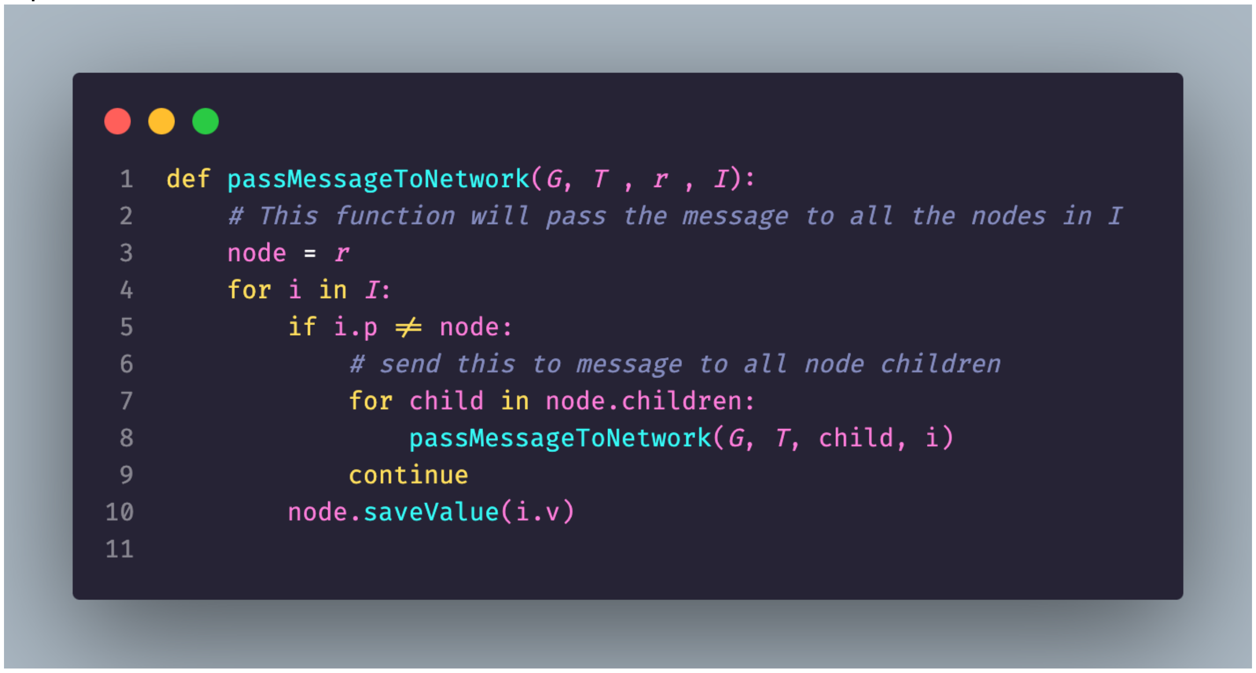
Distributed Computing

Question 1:

The first thing to understand is that we are not limited by the number of messages but on there size, we also need to find algorithm that will run in time (rounds) O(k + depth(T))
For each node we have the sub graph tree parent and children.
A pusdo code will be as follows



```
def passMessageToNetwork(G, T , r , I):
    # This function will pass the message to all the nodes in I
    node = r
    for i in I:
        if i.p ≠ node:
            # send this to message to all node children
            for child in node.children:
                passMessageToNetwork(G, T, child, i)
            continue
        node.saveValue(i.v)
```

Let’s anayse the code

1) Create a function to handle the logic
2) We go over all the items we got as a parameter, note if we are in the recursive step then we can any number between 0 and k. O(1)
3) For each message we get we check if the message is for this node
4) If not send This message to akk the children O(1)

Each message can be sent to max depth(T) this mean we can send a message down the tree, and this will take maxium of O(depth)
Because each round we can send only one message bound by the congest model, the maxmum rounds will be O(k) [one for each round until we get k rounds]
This mean the total run time in O(K + depth(T))

Question 2:

To design a **CONGEST algorithm** for computing a 2-approximation of the diameter of a network, we start by analyzing the relationship between the **radius** and the **diameter** of a graph.

Relation Between Radius and Diameter

- The **radius** r(G) of a graph is the smallest eccentricity among all nodes.
- The **diameter** D(G) is the largest eccentricity among all nodes.
- We aim to prove the following relationship:

$$r(G) \leq D(G) \leq 2 \cdot r(G)$$

Proof of the Relation:

1. **First Inequality ($\boldsymbol{r(G) \leq D(G)}$)**:
    - By definition, the radius r(G) is the smallest maximum distance from any node to all other nodes, while the diameter D(G) is the largest maximum distance. Therefore: $r(G) \leq D(G)$
2. **Second Inequality ($\boldsymbol{D(G) \leq 2 \cdot r(G)}$)**:
    - Let G be a connected graph, and let x be the **center node**, meaning ecc(x)=r(G) the smallest eccentricity in the graph.
    - Assume u and v are the two nodes such that $dist(u,v) = D(G)$ the diameter.
    - By the **triangle inequality**, we have: $dist(u,v) \leq dist(u,x) + dist(x,v)$
    - Since x is the center, $dist(u,x) \leq r(G)$ and$dist(x,v) \leq r(G)$. Substituting these into the inequality: $dist(u,v) \leq r(G) + r(G) = 2 \cdot r(G)$

From the above, we conclude:

$$r(G) \leq D(G) \leq 2 \cdot r(G) r(G)$$

Approach to Computing a 2-Approximation of Diameter

Using the above relation, we observe that to compute a 2-approximation of the diameter D(G), it suffices to compute the **radius** r(G) and return 2·r(G).

Algorithm Description

1. **Key Insight**:
    - The radius r(G) can be computed as the minimum eccentricity among all nodes.
    - Eccentricity for a node t is the maximum distance from t to any other node, i.e., ecc(t)=max v∈Vdist(t,v).
2. **Implementation**:

    - The designated node (root) initiates a **BFS (Breadth-First Search)** to compute the distance from itself to all other nodes.
    - Each node calculates its **eccentricity** using the distances gathered during the BFS.
    - The root collects the eccentricities of all nodes and determines the **minimum eccentricity** r(G).
3. **Complexity**:
    - BFS from the root to compute the maximum distance for a node takes O(D(G)) rounds, where D(G) is the diameter.
    - To compute eccentricities for all nodes, we perform this operation O(1) times in parallel using the CONGEST model, giving a total runtime of O(D(G)).
4. **Output**:
    - The designated root node computes the 2-approximation of the diameter as: 2·r(G)

Final Algorithm

Steps:

1. The designated node initiates the computation.
2. Perform BFS to compute distances from the root to all nodes.
3. Compute the eccentricity for each node ttt based on the BFS results.
4. Determine r(G) as the minimum eccentricity among all nodes.
5. Compute and output 2·r(G) the 2-approximation of the diameter.

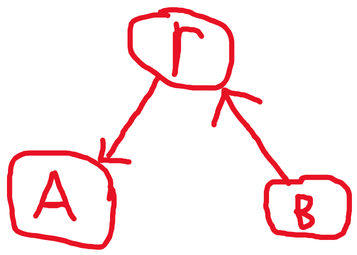
Complexity Analysis

- **Time Complexity**:
    - BFS takes O(D(G)), where D(G) is the diameter.
    - Computing the minimum eccentricity and the final 2-approximation adds negligible overhead, as it involves simple aggregation.
- **Message Complexity**:
    - Each edge transmits O(log n) bits per round (CONGEST model constraint).
    - BFS requires O(|E|) messages, where |E| is the number of edges.

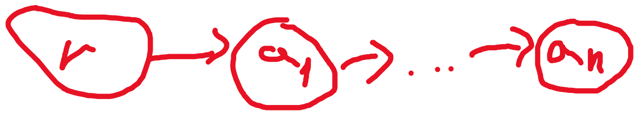
Thus, the total runtime of the algorithm is: O(D(G))

Question 3:

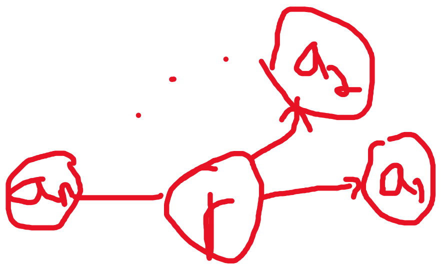
A) Yes given the example in the root (r) need to broadcast amd this can not happen cause r is not reacable from r.



B) Given a graph G we need to prove that broadcast which mean sending a messag from the source to all other nodes.
If all the nodes are reacable from r, and they are all in a line, each step will take at least $n-1$ because each message need to go over each item in the line, and the line have n-1 edegs.



C) No, given a start example broadcast will take O(1)



Question 4:

a) **Why the $\alpha-$Synchronizer Works for All Algorithms** The $\alpha-$Synchronizer ensures correctness because each node waits to receive responses from all its neighbors before proceeding to the next round.

- Assume we have already discovered an internal subtree G′, where messages must propagate through the entire subtree before reaching the root.
- Each message must receive a response acknowledgment from all nodes in its subtree to ensure they are all in sync.
- Now: consider a parent node connected to m subtrees where the induction hypothesis holds. The parent will not increment its round number until it receives a message from all its neighbors, ensuring all subtrees are synchronized.

This mechanism enforces that no node advances beyond a given round until all its neighbors have completed that round, making the $\alpha-$Synchronizer to work for any synchronous algorithm

b) Proof using unduction - We prove the correctness of the $\alpha-$Synchronizer using induction on the size of the graph.

1. **Base Case**:
   - Consider a graph G with one vertex and no edges.
   - In this case, there are no neighbors to communicate with, so the hypothesis trivially holds as the vertex completes each round without waiting
2. **Inductive Step**:
   - Assume the hypothesis holds for a graph G ′ with nnn vertices (all nodes in G′ synchronize correctly using the $\alpha-$Synchronizer).
   - Now consider a tree T with $n+1$ vertices, formed by connecting a new root node to m subtrees where the hypothesis holds.

   Synchronization in T:

   - When the root node sends a message for round i to its neighbors, the subtrees independently ensure (by the induction hypothesis) that no node advances to round i+1 until all nodes in the subtree have completed round i.
   - The root waits for acknowledgments from all its neighbors before proceeding to the next round.
   - Therefore, the hypothesis holds for T, ensuring all nodes are synchronized correctly.
3. **Synchronization Property**:
   - When the root sends a message with round number L to its neighbors, no neighbor can increment its round number beyond L until it has received a “safe” message for L.

- As a result, the difference in the round number i between neighboring nodes is at most ± 1.

c) Message complexity - The message complexity of the $\alpha - \text{Synchronizer}$ can be analyzed as follows:

1. **Messages per Round**:
    - Each node sends a message to all its neighbors.
    - Each neighbor replies with an acknowledgment (ACK) for every received message.
    - After receiving all acknowledgments, each node sends a SAFE message to its neighbors.
    - Total messages per round: $3 \cdot m$ (messages and ACKs)
2. **Total Rounds**:
    - If the algorithm A takes L synchronous rounds, the $\alpha - \text{Synchronizer}$ will also run for L rounds.
3. **Total Message Complexity**:
    - Over L rounds, the total number of messages is: Total Messages= $3 \cdot m \cdot L$
    - Since each message must travel both ways (sending and acknowledgment), the total complexity is: O(3·m·L)