Question 1

Algorithm Description:

Input: Process id
Output: Computes the parents and children of each processor.

In the algorithm, two root processors (with ids "1" and "2") create an initial message that requests adoption from all neighbors in the first round. Each message contains a destination and three data items. The first item stores the processor's id, the second item indicates the type of request ('?' for adoption, 'Y' for an ACK response), and the third item ("1" or "2") identifies whether the processor is rooted at processor 1 or processor 2. The algorithm also utilizes a flag (*round_left)* to track the remaining rounds, initialized to -1.
During each round, the algorithm sends adoption requests ("?" messages) to neighbors, excluding the current node's parent if it hasn't adopted one yet, and sends ACK ("Y" messages) to the parent node. The *round_left* flag is set to 1 to make a two-round wait for responses to requests.
Upon receiving a message in each round, the algorithm processes messages as follows: if the received message is a request for adoption ("?"), the processor checks if it already has a parent. If not, it adopts the sender as its parent, sends adoption requests to other neighbors, and if the message is an ACK ("Y"), it records the sender as its child by adding it to a children vector. Finally, two rounds after the last message is sent from both BFS trees, the algorithm prints out the parent and children of the processor and terminates.

Java Implementation:

Algorithm is implemented in *DoubleBFS.java*

Proof of Correctness - correct output

Prove that the algorithm correctly computes the parents and children of each processor by proof by induction.

Base Case:
In the first round, the neighbors of each root located at distance of 1 receive a message ("?"). They adopt the sender as their parent if they have no parent. Additionally, they receive ACK message(s) ("Y") to agree to be their children. The

length of the path is 1, representing the shortest path between the processors and their root processor.

Induction Hypothesis
Assume that in the first k-1 rounds, the algorithm correctly computes the parents and children of each processor at a distance of k-1 from the root, and each path represents the shortest path from a processor to its root processor.

Induction Step
Show that in the k-th round, the algorithm correctly computes the parents and children of each processor at a distance k from the root, and the shortest paths have been chosen.
In the k-th round, processors at a distance of k from each root receive their message ("?") from processors at a distance of k-1 from each root. Consequently, each processor adopts the processors at a distance of k-1 from each root as their parent if they have no parent. Therefore, the algorithm correctly computes the parents and children of each processor at a distance of k.
By induction, we can conclude that the algorithm correctly computes the parents and children of each processor for all distances from the root in each round.

Proof of Correctness – Termination

When all processors have received the acknowledgment (ACK) response from their children in both trees, the algorithm correctly computes and prints the parents and children of each processor and then terminate.

Time Complexity

The time complexity is **O(d)**, where d is the diameter of the tree.

The time complexity is the number of rounds. In each round, the message travels a distance of 1. To reach the leaf processor in the worst case, the message needs to traverse to the farthest processor which travels the height of the tree. Plus, additional 2 round to wait to get responses to requests.
Therefore, $f(n) = number\ of\ round\ \leq diameter + 2 =$
$O(d)\ where\ d\ is\ the\ diameter$

Communication Complexity

The communication complexity is **O(m)**, where m is the number of edges in the tree.

The communication complexity is the number of messages sent. In this algorithm, there are two types of messages: the request message ("?") and the acknowledgment message ("Y"). Therefore, the communication complexity

$f(n) = number\ of\ messages = 2\ x\ number\ of\ edges = O(m)$ where m is the number of edges in the tree

Does your algorithm work on an asynchronous system (without using a synchronizer)? Explain your answer.

No. This algorithm is not designed for an asynchronous system. It assumes a synchronous environment where request messages ("?") and acknowledgment messages ("Y") are processed and exchanged in a specific order during each round. This assumption poses issues in an asynchronous system where processors may operate at different speeds, leading to the arrival of messages out of order or with delays, potentially causing incorrect behavior.

Additionally, the algorithm's termination relies on the number of rounds. In an asynchronous system, it cannot guarantee the detection of when all nodes have terminated, as there may be no synchronization mechanism to coordinate this information globally. Therefore, due to these considerations, this algorithm is not suitable for an asynchronous system.

Question 2

Algorithm Description:

Input: Process id
Output: The diameter of tree T and the diameter of every subtree of T.

In this algorithm, each processor sends a message to its parent containing two values: the height and diameter of its subtree. The leaf processor initiates the message with height=0 and diameter=0 and sends to its parent. Upon receiving the message, the parent processor first increments the height by 1 and then calculates the diameter based on two different cases, using the updated height value. In the first case, when the processor has only one child, the diameter is set equal to the height. In the second case, when the processor has more than one child, each child's height is saved to a vector. After receiving all child heights, the processor calculates its diameter by adding the top two heights from the vector. Once the height and diameter are calculated, the processor creates a new message with these values and sends it to its parent. When the algorithm reaches the root, it prints the diameter and terminates.

Java Implementation:

Algorithm is implemented in *Diameter.java*

Proof of Correctness - correct output

Prove that the algorithm correctly calculates and prints the diameter of the subtree rooted at each node.

Base Case (Leaf Nodes):

For leaf nodes, the algorithm sets the diameter to 0, and the height is increased by 1 when the parent receives the message. This is correct because there are no paths from a leaf to another node. So, the base case holds.

Induction Hypothesis:

Assume that the algorithm correctly computes the diameter for all nodes up to level k in the tree

Inductive Step:

Assume the algorithm works correctly for nodes up to level k. Now, let's prove that it works for nodes at level k+1.

Case 1: Node with one child

If a node at level k+1 has only one child, the diameter is set to the height. This is correct because the longest path in the subtree is the path to its only child. The algorithm calculates this correctly.

Case 2: Node with multiple children

If a node at level k+1 has more than one child, the algorithm correctly saves each child's diameter to a vector. The induction hypothesis ensures that the algorithm has correctly computed the diameter for each child up to level k. After receiving all child diameters, the algorithm computes the diameter by adding the top two heights from the vector. This is also correct because the top two heights represent the two longest paths in the subtree.

By the induction hypothesis and the correctness of the algorithm for the base case, we can conclude that the algorithm correctly computes the diameter for all nodes in the tree.

Proof of Correctness – Termination

The algorithm terminates when it reaches the root node. At this point, the root has correctly computed the diameter of the entire tree, and the algorithm prints the correct diameter and then terminate.

Time Complexity

The time complexity is **O(d)**, where d is the diameter of the tree.

The time complexity is determined by the number of rounds. To calculate the diameter, start from the leave nodes and traverse to the root node. The worse case occurs when the number of round equals to the farthest leave processor.

Therefore, $f(n) = number\ of\ rounds = height\ of\ tree\ \leq diameter = O(d) \quad where\ d\ is\ the\ diameter$

Communication Complexity

The communication complexity is **O(m)**, where m is the number of edges in the tree.

The communication complexity represents the number of messages sent. In this algorithm, each processor except the root will send a message to its parent. Therefore, $f(n) = number\ of\ messages = number\ of\ processors - 1 = number\ of\ edges = O(m)$ Where m is the number of edges in the tree.

Does your algorithm work on an asynchronous system (without using a synchronizer)? Explain your answer.

Yes. This algorithm works for an asynchronous system as well. In this algorithm, messages are sent in a unidirectional way, specifically from children to their parent in a bottom-up direction. The diameter is calculated by considering the diameters of all children. As long as the parent receives the diameter information from all its children, the diameter can be calculated correctly. The processor speed will not affect the calculation results. Therefore, it works for an asynchronous system.