Distributed Computing Summative Coursework

wtxd25

1 Question 1

(a) The Flooding algorithm works by broadcasting a message to all neighbours and waiting for a response from them to know whether they are a child or not. In this way the algorithm is able to construct a spanning tree T of the input graph G. A necessary definition for this proof is the diameter, d, of the graph G. This is the maximum distance between any two vertices of the graph.

Let the root processor p_r send message M:

- *After k rounds the message M will have reached processors at distance k from p_r*:
  We shall prove this inductively; in the first round p_r sends the message M to its immediate neighbours which are distance 1 from the root. Therefore, this holds for the base case. Assume this is true for the k-th round; the message M has reached all nodes u at distance k from p_r. Now let it be the k+1-th round. All nodes u at distance k from p_r now send the message M to all immediate neighbours, some of which will be distance k+1 from the root processor $\Rightarrow$ this holds for the inductive step, and so the above statement is true.
- *After all seeing the message M, the processors will take a constant time to terminate*:
  The vertices which have received M simultaneously send M to the next vertex in the path and a message informing whether previous vertex is a parent or other to the previous vertex. In the next round, the vertex will either receive a response and terminate or receive no response, realise it is the final node online, and terminate. This process therefore takes at most 2 round which is $O(1)$.
- *The time complexity of flooding algorithm for synchronous model is $O(d)$*
  In our first argument we showed that in round k, our message M has reached nodes of distance at most k from p_r. The algorithm does not terminate until the message M has reached every node. Therefore, the greatest number of rounds this will take is d, when p_r is one of the nodes in the definition of the diameter of G. Furthermore, in our second argument we showed that once all processors have seen

M they will take a constant time to terminate. As such, the time complexity of the flooding algorithm is $O(d) + O(1) = O(d)$ $\square$.

(b) The proof of this is very similar to the above proof for the synchronous model. In the asynchronous model there is usually a maximum message delay, for convenience this message delay is equal to 1 unit of time. The time complexity of the asynchronous model is the number of units of time taken to solve a problem in the worst case. In the previous answer we gave a brief overview of the Flooding algorithm in natural language and in the same way we make use of the diameter of the graph G, d.

In the worst case the root of the graph, p_r is one of the vertices in the diameter, u. We know that the maximum distance leaf from $p_r = u$ is the other vertex in the diameter definition, v, and that the length of this path is d. Messages are passed in both directions along this longest path meaning that the length of the walk is $2d$. Assume that every edge has maximum message delay, and so we use 1 unit of time for each edge traversal. Therefore, the total number of units of time used in the worst case is $2d \Rightarrow$ the time complexity of the flooding algorithm in asynchronous model is $O(d)$ $\square$.

2 Question 2

(a) Let's assume such an algorithm exists. This would mean that each processor p_i would have a bit input b_i and the outcome of the algorithm would be $b_1 \wedge b_2 \wedge ... \wedge b_n$ for some number of processors n which is unknown. One way we could approach this is to coordinate the AND operation using a leader processor, however, the issue here is that no **anonymous** leader election algorithm in rings exists. This means that the AND computation would be performed by all of the processors. If the processors each had knowledge of the number of processors in the ring (non-uniformity) then they could communicate for a set number of rounds, and terminate after this because messages from most distant nodes would have propagated around the ring to them, trivially. However, if this number is unknown, and the ring is anonymous, there is no way for a processor to know when it has the bits from all processors. As such, there is no possible way that a uniform, synchronous algorithm on anonymous rings exists to compute the AND of input bits distributed across processors in a ring $\square$.

(b) The algorithm in natural language for each processor p_i is as follows:

1. Each processor takes some bit as input, b_i and stores this value.
2. count takes the value of 1, initially.
3. While count is less than number of processors, n, do the following:
4. Send b_i value to both left and right neighbours in the ring.

5. The values that are received from the neighbours in the ring should be AND-ed with the value b_i and the result stored in b_i.
6. Add 2 to the count value

We add 2 to the count value each iteration because the value of b_i has the value of two other processors AND-ed with it; coming from the processor's two neighbours in the ring. Duplicate ANDs are inconsequential because, for example, $a \wedge b \wedge a \equiv a \wedge b$. In the end, the result on each processor will be the AND of all bit inputs. The number of messages sent are 2 from each of the n processors each round, hence $2n$ in total. The max number of rounds of the ring is less than $\frac{n}{2}$. Therefore, $2n \cdot \frac{n}{2} = n^2 \Rightarrow$ algorithm sends $O(n^2)$ messages.

(c) The algorithm in natural language for each processor p_i is as follows:

1. Each processor takes some bit as input, b_i and stores this value
2. If the value of b_i is 0 then send a <terminate> message to the left, return 0 and terminate.
3. If the value of b_i is 1 then wait for n rounds or until a message is received. If no message has been received then return 1, and terminate.
4. If the value of b_i is 1 and wait process is interrupted by receiving a <terminate> message from the right then send a <terminate> message to the left, return 0, and terminate.

The algorithm works because a 0 will instantly nullify a conjunction operation; obviously return 0. The return value will only be 1 if all values of b_i are 1 which is true if no messages are sent meaning all processors terminate in n turns. If one of the processors has a bit input of 0 then all processors with value 1 will know that there is a 0 in the ring and return 0, before terminating. The algorithm will not know the number of 0s as this is inconsequential, but will be aware of the presence of a 0. To prove message complexity we do as follows: (label all processors as p_i for $i = 1, ..., n$ for convenience)

- If every processor has $b_i = 1$ for $i = 1, ..., n$ then no messages are sent.
- Otherwise, we decompose the ring into sections which are terminated on each end by processors which have $b_i = 0$. Once a 0 message reaches the start of a new section it cannot pass through: it is stopped by the processor holding 0 which denotes one end of the section.
- This means that the total messages that will be sent will be less than or equal to the sum of messages sent within each section of the ring. This in turn will be less than the sum of the number of nodes in each section. This is upper bounded by 2n, because in the worst case,

when every processor has value 0, there will be n sections of size 2 $\Rightarrow$ most endpoints of the sections will be double counted.

- As such, $2n = O(n)$ which is the worst case message complexity $\square$.