Rishad Rahman (Deepak's Recitation) May 8, 2015

6.046 Problem 10-1

Collaborators: *None*

(a) No such algorithm exists. We can prove that the states of all the processes are always identical at any point in time via induction on the number of rounds. Note that all the processes are in the same state at $r = 0$. Assume that all the processes are at the same state at round $r = k$. As a result, the messages the processes send to the left are all identical and the ones sent to the right are also identical, let them be m_l and m_r respectively. However this means each process receives message m_l on its right port, and m_r on its left port hence all the processes perform identical transitions so the states at round $r = k + 1$ are all identical as well. This implies if one leader gets elected, all of them get elected, a contradiction.

(b) We first note that if each node had a unique UID, then a leader can be elected by selecting the maximum. To do this, on the beginning of each round, a node will send the maximum UID it has seen thus far to the left and right. On the first round, each node sends its own UID to the left and right. If a node receives a UID that is greater than its own, then it can confirm that it is not the leader and use the maximum of what it has received to be the next UID to propagate to the left and right. We claim that after $\frac{n}{2}$ rounds, there will only be one node remaining which has not ruled itself out as a leader. This follows easily as the maximum UID will propagate through the entire ring after $\frac{n}{2}$ rounds. To generate random UIDs, we borrow the randomization scheme from Lecture 19. Each process chooses an id uniformly at random from $\{1, 2, ..., \lceil \frac{n^2}{2\epsilon} \rceil\}$. The Lecture shows that via the union bound that the probability all generated numbers are different is at least $1 - \epsilon$. Each trial has $2n$ messages per round for $\frac{n}{2}$ rounds the number of messages sent is n^2. We have that with probability at least $1-\epsilon$ the protocol works after 1 trial i.e. time complexity of $\frac{n}{2}$ rounds and message complexity of n^2.

(c) Suppose we have a ring $(x_1, x_2, ..., x_n)$. We connect this with a copy of itself through a vertex i.e. x_1 so that one side of the ring is $(x_1, x_2, ..., x_n)$ and the other side is $(x'_1, x'_2, ..., x'_n)$. Basically you mesh the two rings together to form one super ring of $2n$ vertices. We claim that if the algorithm exists, there is a possibility of obtaining 2 leaders in the super ring. The existence of an algorithm implies a sequences of messages in $(x_1, x_2, ..., x_n)$ that elect a leader and by symmetry a sequence of messages in $(x'_1, x'_2, ..., x'_n)$ which elect a leader. Therefore the algorithm has a non-zero chance of selecting messages in the super ring which selects a leader in the original half and the copied half, giving 2 leaders, a contradiction so such an algorithm cannot exist.

Rishad Rahman (Deepak's Recitation) May 8, 2015

6.046 Problem 10-2

Collaborators: *None*

(a) The nodes of the tree during its creation will obey the following message protocol:

- A leaf will send a *search* message to all of its neighbors (excluding the parent) if and only if it receives a *ready* message from the parent, excluding the root which sends a *search* message regardless
- A non-leaf, non-root node will send a *ready* message to its children if and only if it receives a *ready* message from its parent
- The root will send a *ready* message to its children if and only if it receives a *ready* message from all of its children
- A non-leaf, non-root node will send a *ready* message to its parent if and only if it receives a *ready* message from all of its children
- If a node sends a *parent*(b) message, it sends a *ready* message directly afterwards
- A non-leaf, non-root node will send a *done* message to its parent if and only if it receives a *done* message from all of its children
- A leaf will send a *done* message to its parent if and only if it is a leaf of the entire BFS-tree i.e. does not gain any children/does not obtain any *parent*(*true*) messages.

The BFS-Algorithm follows by initializing a *search* messages from the root and then waiting until the root receives a *done* message from all of its children. To argue correctness, we just have to argue that the tree progresses one level at a time just like in BFS. More formally, we argue that nodes of shortest distance k from the root are reached before those of distance $k + 1$. This follows since a leaf cannot send a search message unless it receives a ready message from its parent which propagates upwards, which leads to requiring the root sending a ready message. However the root cannot send a ready message until it receives a ready message from all of its children, which propagates downwards until acquiring ready messages from all the recently searched nodes. Therefore the tree does not continue the search unless all its leaves are ready as expected from a BFS algorithm.

(b) The level of the BFS-tree increases by 1 each time *ready* messages propagate upwards, reaching the root, then downwards reaching the leaves before continuing the search. This means that each level in the current BFS-tree is accessed twice. This happens until the BFS-tree terminates. If the BFS-tree is of height h, then the first level gets accessed $2h$ times, the second $2(h - 1)$ times, etc. Since each channel in a level takes maximum d time to transmit a message we have that the amount of time for the ready messages is $O(2d(h + h - 1 + h - 2 + ... + 1) = O(dh^2) = O(diam^2 \cdot d)$ since $h = O(diam)$. The communication complexity on the other hand explodes since we need to count the number of branches in our BFS-tree. The number of branches in the jth level is bounded by n^j and the level gets accessed $2(h + 1 - j)$ times so the total number of messages is $O(\sum_{j=1}^{h}(h + 1 - j)n^j) = O(hn^{h+1}) = O(diam \cdot n^{diam})$.