COSC-373: HOMEWORK 3

LEE JIAEN, HYERY YOO, AND ALEXANDER LEE

Question 1. (a) We first use the LE/BFS/SSSP protocol from class to find the leader, BFS tree, and let the leader know the distance from the furthest node/nodes which is labeled as "Dist".

(b) Each node then does the following:

```
local_counter ← $t_v$
second_counter ← 0
status ← inactive
if leader then
    reset_cntdown ← Dist
else
    reset_cntdown ← 0
foreach round i do
    if $i = 0$ then
        if leader then
            status ← active
            send message "reset_cntdown − 1" to children in BFS tree
    if received messages then
        reset_cntdown ← message
        status ← active
        send message "reset_cntdown − 1" to children in BFS tree
    if status = active then
        if second_counter = reset_cntdown then
            local_counter ← 0
            status ← inactive
            halt
        else
            second_counter++
    local_counter++
return
```

(Correctness) The protocol above uses the LE/BFS/SSSP protocol discussed in class to elect a leader, find a BFS tree, and notify the leader of the distance from the leader to the furthest node/nodes in the BFS tree in $O(D)$ rounds. The algorithm finds the distance from the leader to the furthest node by using SSSP to get all of the nodes' distances from the leader and then have the leaves of the BFS tree send their distance to their parent nodes. At each of the next rounds, each internal node waits for all its children to send a message. Once each internal node has received messages from all its children, it takes the maximum distance, and sends it to its parent node. Eventually, the leader will receive messages from all of its children and note the maximum distance it encounters as "Dist" and set it's $reset\_cntdown$ to "Dist". The leader then knows that it will take "Dist" (which can be at most D) rounds to get from the leader to the furthest node/nodes in the BFS tree. Therefore, the leader decides to wait "Dist" rounds before resetting and then lets its children know to wait one less round and then those nodes let their children know to wait one less round and so on. The message that a node receives tells the node how many rounds to wait until resetting, i.e., how many rounds it will take until we reach the furthest node/nodes. Furthermore, $second\_counter$ and $reset\_cntdown$ help each node keep track of when to reset and $second\_counter$ only starts incriminating after the node has received a message and updated its $status$ to active. In $r_0 = O(D)$, we would have reached every node in the BFS tree and the furthest node/nodes from the leader in the BFS tree will be told to wait 0 rounds to reset. The furthest node/nodes will then have $second\_counter = reset\_cntdown$ and immediately reset $local\_counter$ to 0. In the same round, every other node in the BFS tree would also have $second\_counter = reset\_cntdown$ and reset $local\_counter$ to 0. The BFS tree ensures that we reach every node in the graph. Thus, the above protocol is correct because all nodes would reset $local\_counter$ to 0 at the same time in $r_0 = O(D)$.

(Runtime) All nodes reset after the furthest node/nodes received a message to reset. When we start from the leader, the furthest node/nodes receive a message to reset in at most D rounds. If we have less than D rounds, then the furthest away nodes in the BFS tree could potentially not be notified of the reset and fail to reset at the same time as the other nodes. Thus, the algorithm above will take $O(D)$ rounds. We know that the LE/BFS/SSSP algorithm will also take $O(D)$ rounds. Therefore, this protocol will take $O(D)$ rounds.

(Congestion) We already know that the CONGEST protocol LE/BFS/SSSP from class sends $O(\log n)$ bits. Furthermore, the algorithm above has nodes only send a message containing a constant that indicates how many rounds to wait until resetting, which only requires $O(1)$ bits. Therefore, this protocol is a CONGEST protocol.

Question 2. (a) Each node v does the following:

```
source ← ID of source node
current ← $\infty$
$n$ ← size of the network
$M \leftarrow \emptyset$
foreach round $r$ do
    if $myID = source$ then
        $current \leftarrow 0$
        send message $current + w(v,u)$ to each neighbor $u$
    foreach message $m$ received from neighbors in round $r-1$ do
        $M \leftarrow \{m\} \cup M$
    if $M \neq \emptyset$ then
        $new \leftarrow \min\{M\}$
        $M \leftarrow \emptyset$
        if $new < current$ then
            $current \leftarrow new$
            send message $current + w(v,u)$ to each neighbor $u$
    if $r = n$ then
        output $current$ and halt
```

(Correctness) Edge weight is added to the sum of weights each time a message is sent across an edge, so any message received by node u is the sum of weights along a path from the source node to node u. Since the weighted length along a path is independent of the unweighted length of the path, the minimal weighted length does not necessarily arrive at node u before larger weighted lengths. At round r, only the smallest message received from a neighbor in rounds 1 through $r-1$ is kept by node u, and a new message is sent to node u's neighbors if the minimal weighted length is updated. This is sufficient to guarantee that node u receives its true minimal weighted length from a neighbor because a least weight path from node v to node u through node k must go through the least weight path from node v to node k, and the least weight path from node v to node u must contain at least one of node u's neighbors.

(Termination and Runtime) All nodes terminate at round n, where n is the size of the network given as input. Messages propagate through the network one edge per round, so the sum of weights along a path from v_1 to v_k through $v_2, v_3, \ldots, v_{k-1}$ takes $k-1$ rounds to arrive at v_k from v_1. The longest possible least weight path from node v to node u is one that passes through every node in the network, so the correct minimal weighted distance is sent to node u from its neighbor at round $n-1$. In such case, node u's *current* is updated to the correct weighted distance at round n, and the process can terminate. For cases with shorter least weight paths, all nodes must have received the correct minimal weighted distance before round $n-1$, so their *current* has the correct value at round n. Since the algorithm terminates at round n, the runtime is $O(n)$.

(Congestion) The message sent by any node is the smallest sum of weights along the least weight path from the source node. Therefore, the messages are at most a sum of $n-1$ edge weights before the algorithm terminates at round n. Each edge weight can be encoded by $O(\log n)$ bits. The largest number that can be encoded by $\log n$ bits is $n-1$, so the sum of $n-1$ edge weights is at most $(n-1)(n-1) = O(n^2)$. This number can be encoded by $O(2\log n) = O(\log n)$ bits, so any message can be encoded within $O(\log n)$ bits. The condition for the CONGEST model is satisfied.

(b) Let G be a graph with n nodes, $V = \{v_1, v_2, \ldots, v_n\}$, where each node v_i is connected to every other node in G. It follows that the diameter of G is $D = 1 = O(1)$. Assume that $w(v_1, v_2) = w(v_2, v_3) = \cdots = w(v_{n-1}, v_n) = 1$ and the weight of every other edge in G is greater than n. Given source v_1, the weighted shortest path from v_1 to v_n, $P = (v_1, v_2, \ldots, v_{n-1}, v_n)$ because the weighted length of P is $1 \times (n-1) = n-1$, is necessarily smaller than any path containing an edge with weight larger than n. In each round, the sum of weights travels along one edge and adds the weight of one edge to the sum, so the sum of weights along path P takes $n-1$ rounds to arrive at v_n from v_1. Therefore, the algorithm must run for n rounds to solve the w-SSSP problem despite the small diameter of G.

Question 3. (a) We proceed with devising a greedy algorithm that is guaranteed to produce a proper $\Delta + 1$ coloring of G. The algorithm executes as follows:

(1) Initialize an empty set C, which represents the set of colors.
(2) Assign any vertex in V the color 1 and add 1 to C.
(3) Choose an uncolored vertex $v \in V$. Assign v the smallest color in C that has not been assigned to v's colored neighbors. If v's colored neighbors have been assigned all the colors in C, add a new color $\max\{C\} + 1$ to C and assign it to v.
(4) Repeat step (2) until all vertices are colored.

Since the algorithm assigns an uncolored vertex $v \in V$ with the lowest color in C that has not been assigned to v's neighbors or assigns v with a new color not yet in C, no two neighboring vertices are assigned the same color. Furthermore, because the algorithm only adds a new color to C when all neighbors of v have been assigned all the colors in C and v has at most Δ neighbors, the algorithm adds at most $\Delta + 1$ colors to C. Thus, only $\Delta + 1$ colors are required. Therefore, the algorithm produces a proper $\Delta + 1$ coloring of G.

(b) Consider the graph G_k of $\Delta + 1$ vertices, where each vertex is connected to every other vertex. In this case, every vertex has a degree of Δ, so the maximum degree is Δ. Now, suppose towards a contradiction that we have a Δ proper coloring of G_k. Thus, at most Δ colors have been assigned to $\Delta + 1$ vertices of G_k. By the Pigeonhole Principle, at least two vertices in G_k have been assigned the same color. Since every vertex is connected to every other vertex, these two vertices with the same color are neighbors. This is a contradiction since we assumed that we have a Δ proper coloring of G_k, where no two neighboring vertices are assigned the same color. Therefore, G_k does not admit a proper Δ coloring.

(c) Each phase i:

```
• If received message "halted" from neighbor in phase $i - 1$
    – Remove neighbor from set of neighbors
• Execute protocol $\Pi$
• If node is in the MIS found from $\Pi$
    – Output $i$, send message "halted" to neighbors, and halt
```

(Correctness) In each phase i, protocol Π finds a MIS in the network of remaining nodes that have not halted. The nodes found in the MIS from phase i output color i. Neighboring nodes will not output the same color since they belong to different maximal independent sets.

(Runtime) After each phase if a node is not in a MIS, then at least one of its neighbors becomes part of an MIS, by the maximality condition. As such, after each phase, a node's degree decreases by at least one. Since a node has at most Δ neighbors, at most Δ phases are needed so that all it's neighbors are assigned a color. Hence, each node requires at most $\Delta + 1$ phases to output a color i. Since each phase runs in $T(n)$ rounds, the protocol thus computes a property $\Delta + 1$ coloring of G in $O((\Delta + 1)T(n)) = O(\Delta T(n) + T(n)) = O(\Delta T(n))$ rounds.

(Congestion) Since the above protocol uses the CONGEST protocol Π and apart from in protocol Π, nodes only send the messages "halted" to neighbors, which only require $O(1)$ bits, the protocol above is therefore a CONGEST protocol.