CS 9868/4438 Internet Algorithmics

Solution for Assignment 3

1. *Design and implement in Java a synchronous distributed algorithm that uses flooding to build a multicasting tree rooted at node s for the set R of processors that have even ID's.*
   Give an informal, high level description of the algorithm in English.

   In the first round the root processor *s* sends requests for adoption to all its neighbours, as is done in the algorithm for computing a breadth first search tree. When a processor *p* receives the first request for adoption it will temporarily set the processor *q* that sent the request as its parent; however, *p* will not send yet a reply to *q*. Further requests received by a processor that has already set its temporary parent will be rejected by sending them rejection messages in the next round.

   After a processor receives its first request, in the next round it will send requests for adoption to all its neighbours, except its parent and any other processor that also sent it a request. Each processor keeps track of how many requests it has sent and how many responses it has received to its requests. When a processor receives answers for all its requests then it will decide whether it will be part of the multicasting tree in the following manner:

   - If the processor has even ID then it will be part of the multicasting tree, or
   - if the processor has odd ID, but it has at least one child then it will be part of the multicasting tree;
   - otherwise the processor will not be part of the multicasting tree

   If a processor *p* determines that it will be part of the tree it sends an acceptance message to its parent (if any) in the next round and then; upon receiving this acceptance message from *p*, the parent processor adds *p* to its list of children. If a processor determines that it will not be part of the tree, it sends its temporary parent (if any) a rejection message in the next round, then it sets its parent to null and terminates.

- *Submit a java implementation of your algorithm.*

  See Multicast.java posted in the course's website.

- *Prove that your algorithm terminates.*

  The java code has some parts marked as (1), (2), and so on in the comments. Please refer to these portions of the code for the proofs below. In statement marked (1) the root processor *s* stores in message buffers the first set of requests for adoption that will be sent in the first round. These messages are sent in the first round. Note that every processor that first receives a request for adoption will set variable *firstMessage* to true and the block of code marked (2) will buffer request for adoption messages that in the next round will be sent to all its neighbours, except those that previously sent them requests.

  Therefore, in the first round the requests will arrive to all processors at distance 1 from *s*. In the second round those processors will send requests to their neighbours, hence every processor at distance 2 from *s* will receive at least one request. In the third round the requests will be received by the processors at distance 3 from *s* and so on. Since the network has a finite number of processors, after a finite number of rounds all processors will have received at least one request.

  When a processor *p* receives the first request for adoption, in the block of code (3), *p* sets its temporary parent and stores in vector *requestsDenied* the ID's of those processors that sent it further requests. Vector *adjacent* stores the ID's of the processors to whom *p* will send requests in the next round. This way *p* knows how many requests it sends and how many responses it must receive. Note that if *p* is a processor that is farthest from *s* then its vector *adjacent* will be empty and in the block of code (2) it will buffer a message of response to its potential parent; in the following round *p* will send the response to the parent and then it will terminate.

  From this point on processors will terminate in decreasing order of distance from *s*. Let *d* be the largest distance from any processor to *s*. After the processors at distance *d* from *s* send their responses to their (potential) parents, in the next round the processors at distance *d*-1 from s will buffer responses to their (potential) parents in blocks of code (2) and (4). In the following round these processors will send their responses in the blocks of code (5) and (6) and then they will terminate. After *d*-2 additional rounds all processors will terminate.

- *Prove that your algorithm produces the correct output.*

  Note that in the block of code (8) a processor with even ID will send a message to its parent indicating that it will be part of the multicasting tree. In (7) a processor will also decide to be part of the multicasting tree if it has at least one child. Since then the only leaf processors that are part of the multicasting tree have even ID's and the other processors that are part of the multicasting tree either have even ID's or are in the paths from even ID's leaves to *s*, then all processors in the multicasting tree are as required.
  Since a processor accepts only the first request for adoption that it receives, then in the resulting tree all paths are shortest paths from *s*. As shown above all processors receive requests for adoption and hence all of them execute (7) or (8) then all processors that must belong to the multicasting tree are part of the tree selected by the algorithm.

- *Compute the time complexity and communication complexity of your algorithm.*

  The algorithm needs O(diameter) rounds for the requests for adoption to be received by all processors in the system and O(diameter) additional rounds for all responses to be sent, received, and processed. Therefore, the time complexity of the algorithm is O(diameter).

  Since this is a flooding algorithm, each edge carries 2 messages, one in each direction. The total number of messages is then 2*m*, where *m* is the number of edges. Therefore, the communication complexity is O(*m*).

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

  Since each processor accepts the first request for adoption that it receives, in an asynchronous system we cannot guarantee that the request will arrive through a shortest path. Hence, in an asynchronous system we cannot guarantee that all paths in the tree produced by the algorithm are shortest paths.

2. *Design and implement in java a synchronous distributed algorithm for computing for each node u of a tree the sum of distances from u to each node in the subtree rooted at u. Give an informal, high level description of the algorithm in English.*

   The algorithm uses convergecast. Each node u will send to its parent two values: The sum of distances from *u* to all nodes in the subtree rooted at *u*, and the number of nodes in the subtree rooted at *u*. The leaves send values 0,1 to their parents in the first round. Each internal node of the tree keeps track of the number of messages that it has received. When a node *u* has received pairs of values (sumDistances,numNodes) from all its children, then it computes the total number of nodes in its subtree and the sum of distances from *u* to all nodes in its subtree as follows:

   - Let $(sumDistances_1,numNodes_1)$, $(sumDistances_2,numNodes_2, \ldots, sumDistances_k,numNodes_k)$ be the values received by *u* from its children.
   - Total number of nodes in the subtree rooted at u is $numNodes_1 + numNodes_2 + \ldots + numNodes_k + 1$
   - The sum of distances from *u* to all nodes in its subtree is $(sumDistances_1+numNodes_1) + (sumDistances_2+numNodes_2) + \ldots + (sumDistances_k+numNodes_k)$

     Notice that for each node *v* in the subtree rooted at the *i*-th child of u_i of *u*, the distance from *v* to *u* is 1 more than the distance from *v* to u_i. Hence the sum of distances from all the nodes in the subtree rooted at u_i to *u* is $sumDistances_i + numNodes_i$.

- *Submit a java implementation of your algorithm.*

  See SumDist.java posted in the course's website.

- *Prove that your algorithm terminates.*

  In the first round the leaf nodes send a message to their parents and terminate. After a processor receives messages from all its children, in the next round it will send a message to its parent (if any) and then it will terminate.

Let the largest distance from a node to *s* be *d*. In the first round all processors at distance *d* from *s* will terminate as all of them are leaves. In the second round all processors at distance *d*-1 from *s* will have terminated as all of them must have received messages from their children (if any, which are at distance *d* form *s*). After three rounds all processors at distance *d*-2 from *s* must have terminated because all of them must have received messages from their children (if any, which are at distance *d*-1 from *s*). Therefore, after *d* rounds all processors must have terminated.

- *Prove that your algorithm produces the correct output.*

  The leaves send in the first round messages to their parents containing two values: 0,1. As explained above the first number in each message is the sum of distances from a node *u* to all the nodes in its subtree and the second number is the total number of nodes in the subtree rooted at *u*. For each internal node the algorithm keeps track in variable *childrenLeft* of how many of its children have sent it messages; when *childrenLeft* = 0 it will have received messages from all of them and in the next round it will send its parent (if any) a message containing two values (sum of distances, total number of nodes) computed as explained in the first part of the question. Therefore, at the end of the algorithm each processor will correctly compute the sum of distances from itself to all the processors in its subtree.

- *Compute the time complexity and communication complexity of your algorithm.*

  The algorithm needs O(diameter) rounds for the messages to propagate from the farthest node from *s* all the way to *s*. Therefore, the time complexity is O(diameter). Each edge of the tree transmits one message from a child note to its parent. Therefore, the total number of messages that are sent by the algorithm is *n*-1 and so the communication complexity is O(*n*).

- *Does your algorithm work on an asynchronous system?*

  Yes. Note that a processor only sends a message to its parent when it has received messages from all its children. The values computer by the algorithm will be the correct ones as the only thing that the algorithm needs are the values computer by the children; the order in which these values are received does not matter.