

Distributed Algorithms

Homework 4

Name: Vahid
Surname: Khalilpour Akram

Student Number: 91120020516

Email: Vahid59@gmail.com

Contents

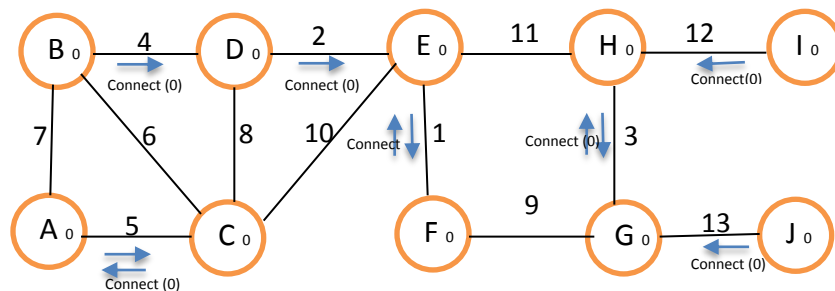
Question 1: Trace of asynchronous GHS algorithm.....	1
Question 2. Trace of asynchronous Petersen leader election algorithm	12

Question 1 (33 pts): Show an example trace of asynchronous GHS algorithm by giving all message transmissions, edge classifications and fragments step by step. In your trace, please show all fragment combination possibilities (fragments with the same level combine, a fragment with a smaller level wants to combine another fragment, fragments with a greater level wants to combine another fragment).

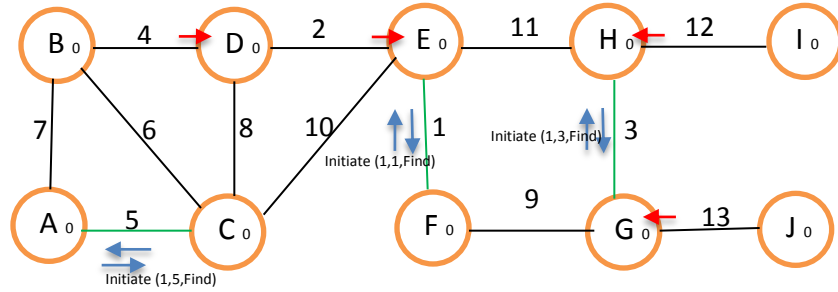
In this problem to simplify the tracing process, various colors are used to show different states and events. The following list shows the used colors and their meaning over all figures.



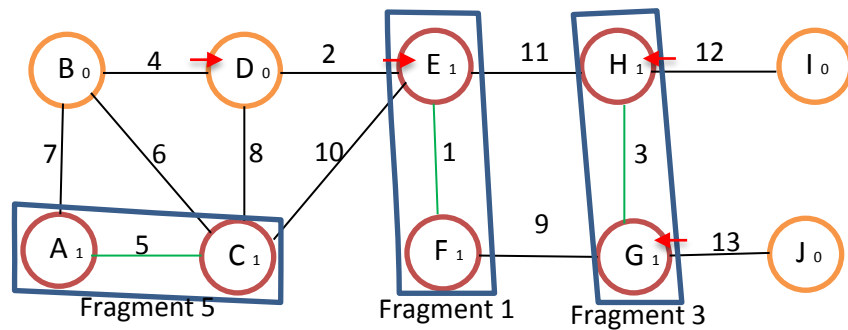
At the starting of algorithm each node first chooses its minimum-weight adjacent edge, marks this edge as a branch of the MST, sends a message called *Connect* over this edge, and goes into the state *Found*, waiting for a response from the fragment at the other end of the selected edge. The **Green** color is used to show *Branch* edges, the **Red** color is used to show *Rejected* edges and the **Black** color is used to show Basic edges. Also a little number in the right of each node shows the current Level of that node. Here is the starting situation in this algorithm.



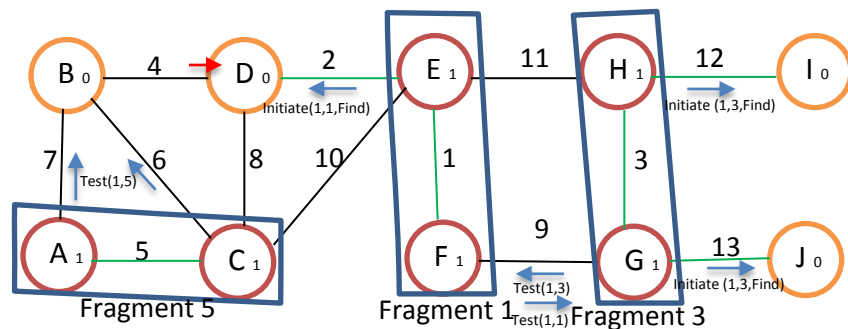
In GHS algorithm initiate messages are sent over branch edges. So at the starting of algorithm a process sends an initiate message to its neighbor if it send and also received a connect message from that neighbor in previous phase. As we can see in the previous figure this condition is satisfied between nodes A-C, E-F and H-G. So the initiate messages are exchanged between these nodes and other messages are queued by receiving nodes. In the following figure the queued messages are shown by a little red arrow.



Upon receiving initiate messages the fragment level of nodes are increased to 1 and three new fragments is constructed as following figure.

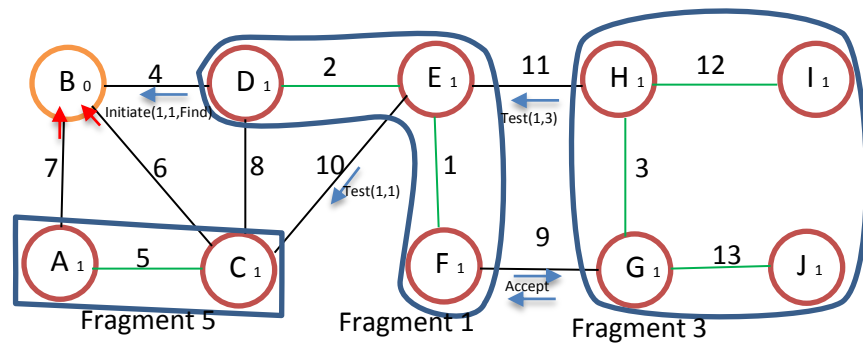


Note that the statuses of links between nodes in each fragment are changed to Branch. Because the level of new fragments are increased to 1, now they can response to previously received connect request. This is an example situation where a fragment with smaller level wants to connect to a fragment with greater level. So as shown in next figure, in response to the queued connect messages, nodes E, H and G send initiate message to nodes D, I and J respectively.

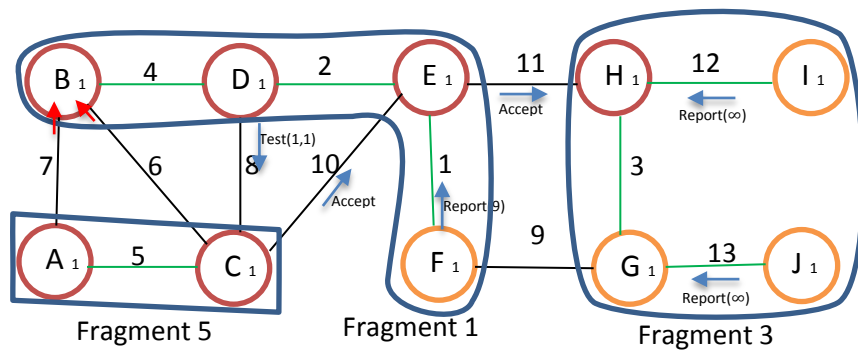


Also the roots of new fragments (includes A, C, F, G, H, E) start finding minimal outgoing edge by sending a test message over minimum weighted basic link. Note that nodes E, H and G send test messages with delay because of sending initiate messages.

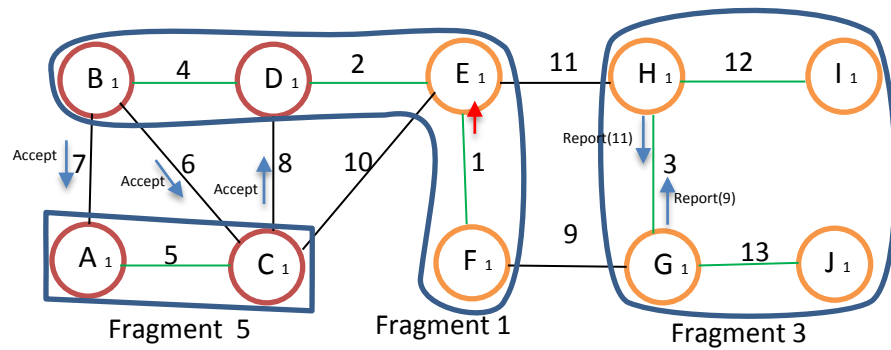
In response to test messages, nodes F and G send an Accept message to each other. In response to initiate message, node D merged to Fragment 1 and node I and J merged to fragment 3. Because the level of node D is increased to 1 it sends an initiate message to response node B queued connect request. Node B delays in responding to test messages from A and C because the level of their fragment is bigger than its level. Here is the example when a fragment with the bigger level wants to connect to fragment with smaller level. Node H send a test message to E because 11 is the only basic edge in H. Node E has a smaller basic link so it sends a test message over edge 10 to C. This events are shown in the following figure.



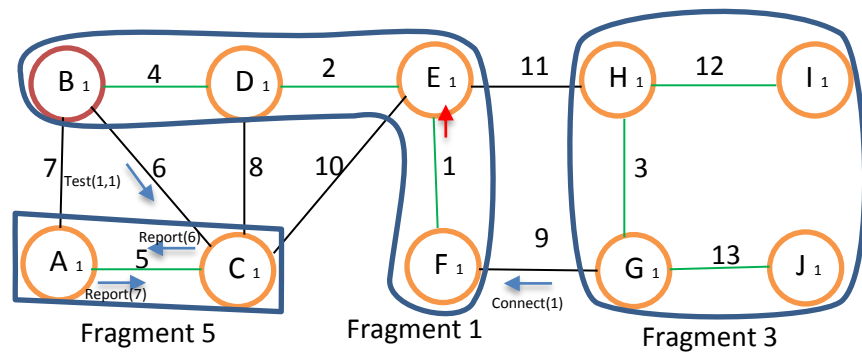
After receiving initiate message, Node B merged with Fragment 1. Also node D send a test to C to find minimum outgoing edge. Node C accept with E request and node E accept with H request because the levels of both fragments are equals. Nodes I and J don't have any outgoing edge hence they report a maximum number to fragment root. Because the nodes F, G, I and J decide on their minimum outgoing edges they go to Found state.



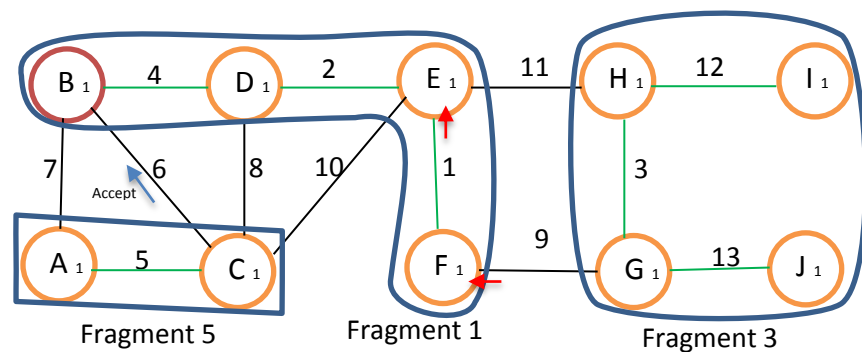
Node B merged with Fragment 1 and its level increased to 1. Hence it response to queued test messages from A and C. In response to test message node C sends an Accept to D. node E puts the received report from F to queue because it does not has the response from D. nodes G and H agree on minimum outgoing edge by exchanging a report message.



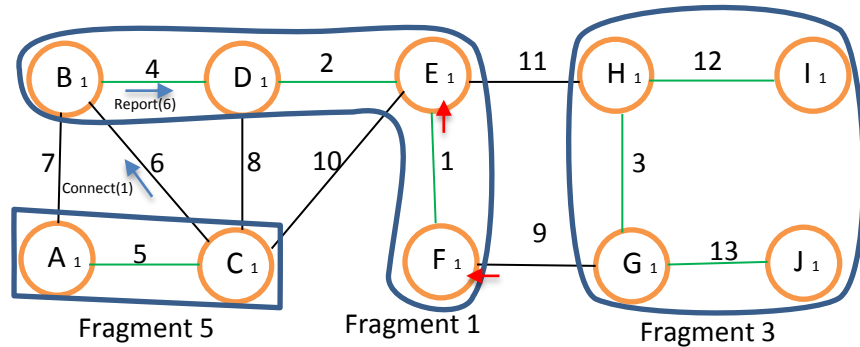
Fragment 3 sends a connect request to Fragment 1. Also node B sends a Test message to check its the outgoing edge. Node A and C agree about minimum outgoing edge by exchanging a report message.



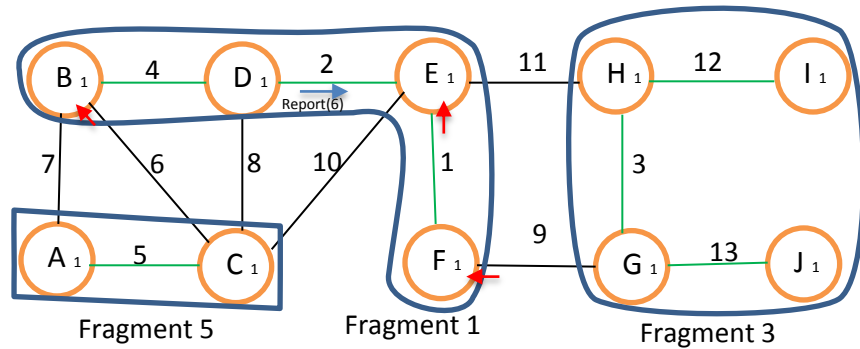
Node F puts the connect request in the queue because it is waiting for E to decide about link 9. At this moment this link type is Basic for F and is Branch G. Node C accept with B request because the level of two fragment are equals.



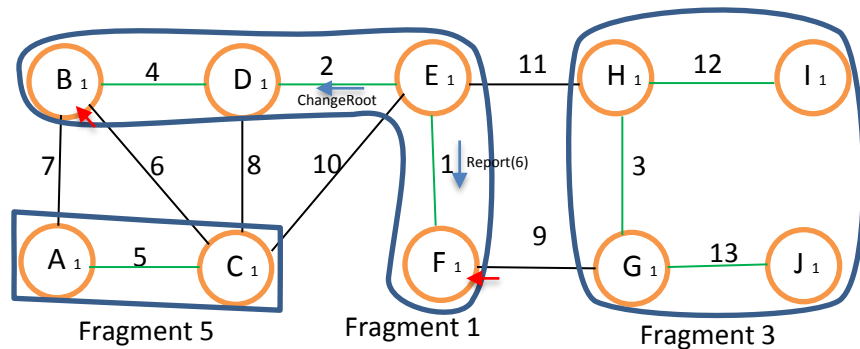
Node B sends its minimum outgoing edge to D and node C as a root of Fragment 5 sends a connect message to B.



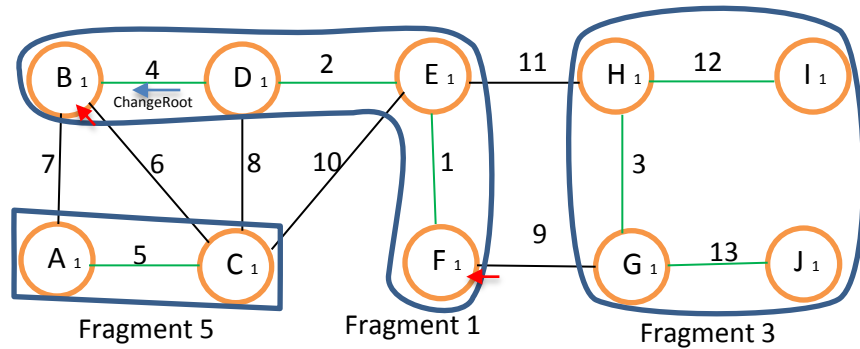
Node B puts the incoming connect message to queue because it needs to receive a change root message from root. Node D forward the minimum outgoing edge to E (because $6 < 8$).



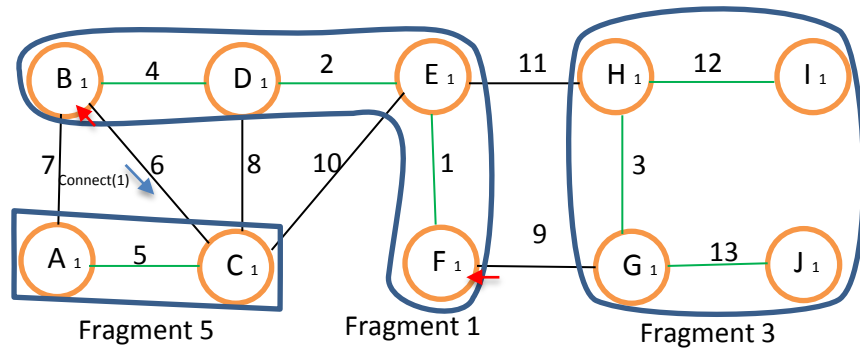
After receiving D response, node E decides about minimum outgoing edge and reports it to F. Also it sends a Change Root message toward D.



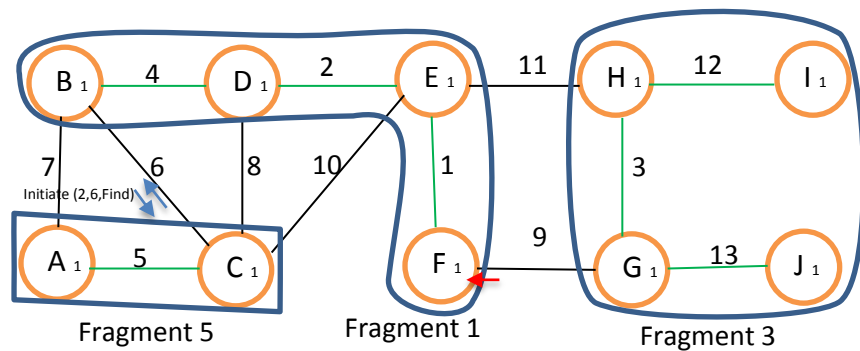
Node D forwards ChangeRoot message to B.



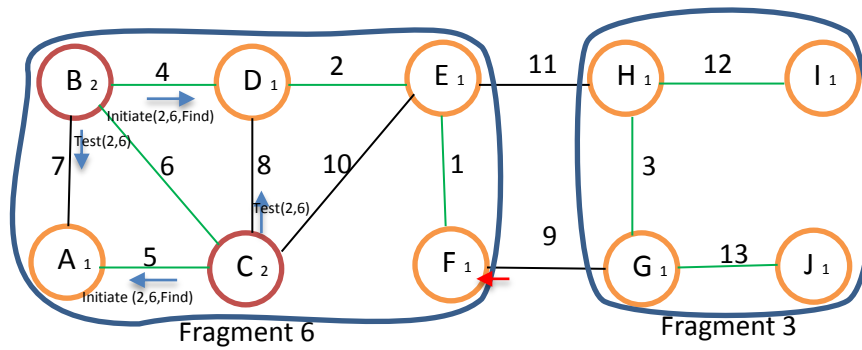
After receiving ChangeRoot node B sends a connect message to C.



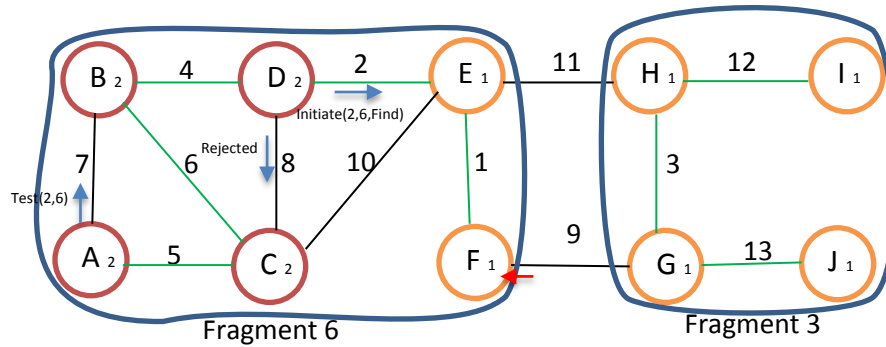
B and C exchange an initiate message to response their connect messages.



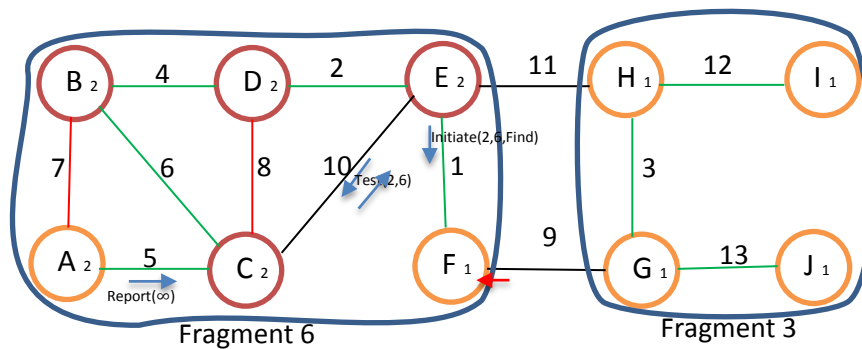
Two fragments are merged and the new level and fragment ID are sent over branch edges to announce other nodes. Also the roots of new fragment send test messages to find minimum outgoing edges.



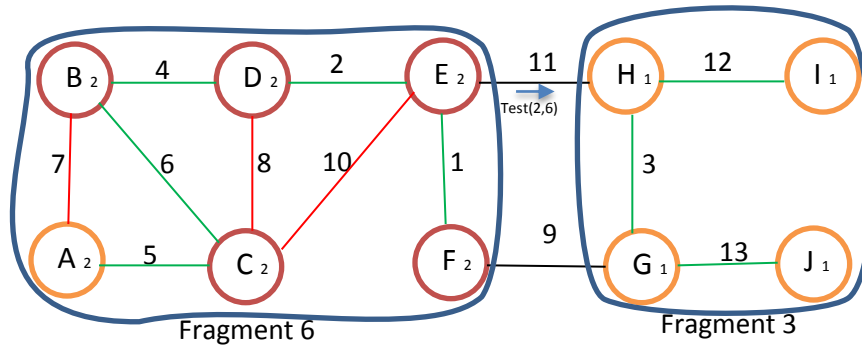
Node D reject C because they are in the same fragment.



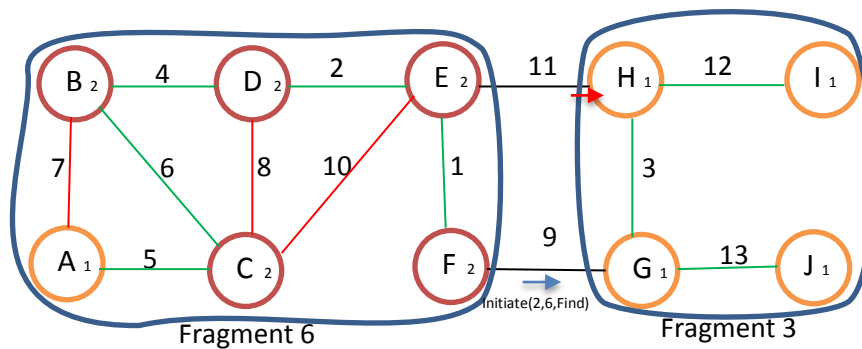
Nodes A and B reject each other because they receive test messages from same fragment ID from each other. Same case is acquired between nodes C and E.



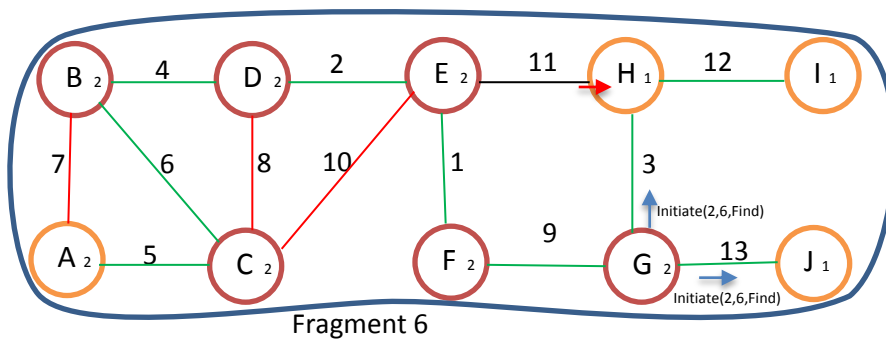
Node E tries link 11 by sending a test message.



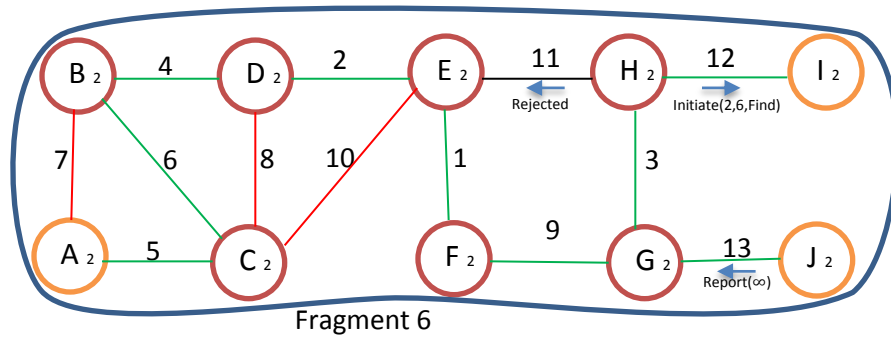
The level of Node F is increased to 2 hence it sends an initiate message to G. Node H put the test message in the queue because it has bigger fragment level.



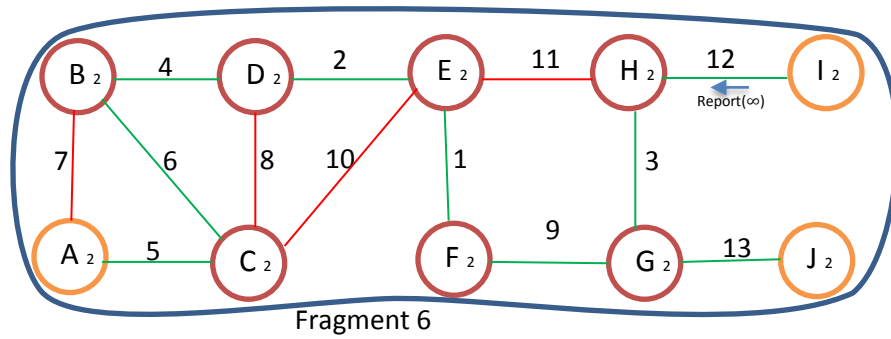
Node g receives initiate message and change its fragment ID , two fragment are merged. Node G sends initiate message to others to announce the level and ID of new fragment.



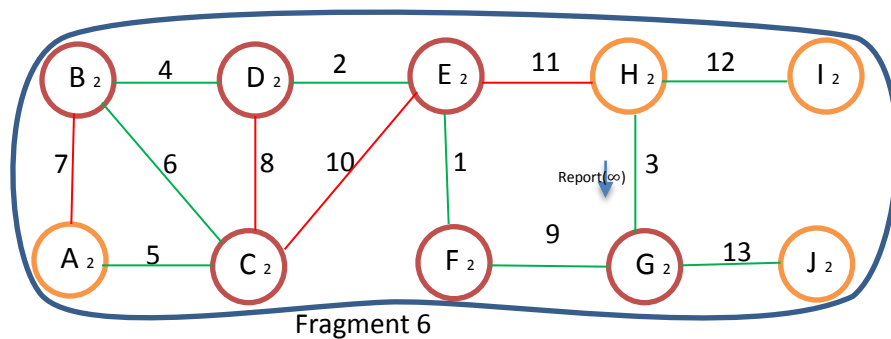
Node H rejects E because now they have same fragment ID.

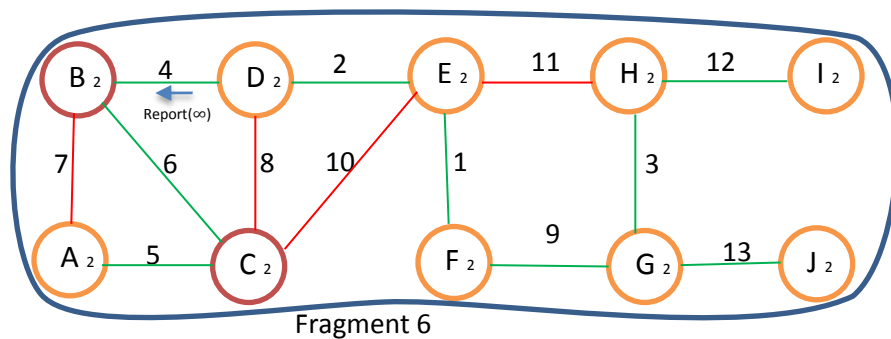
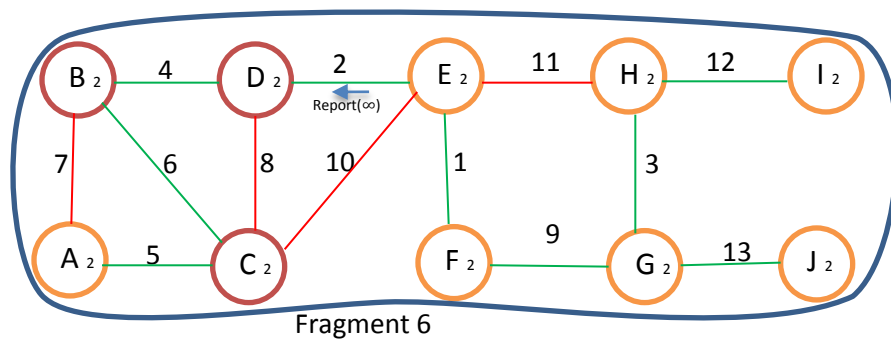
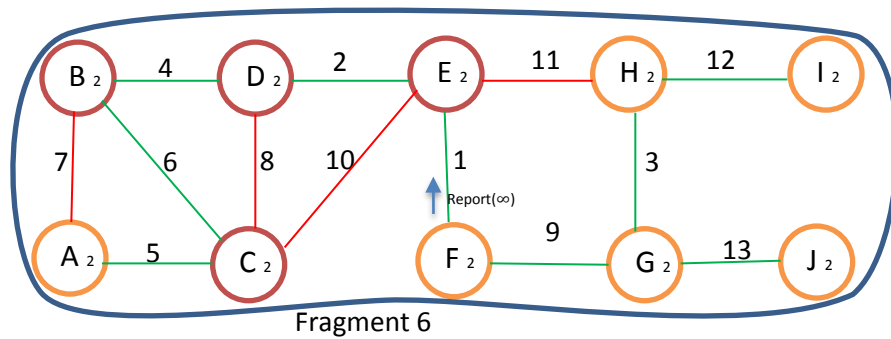
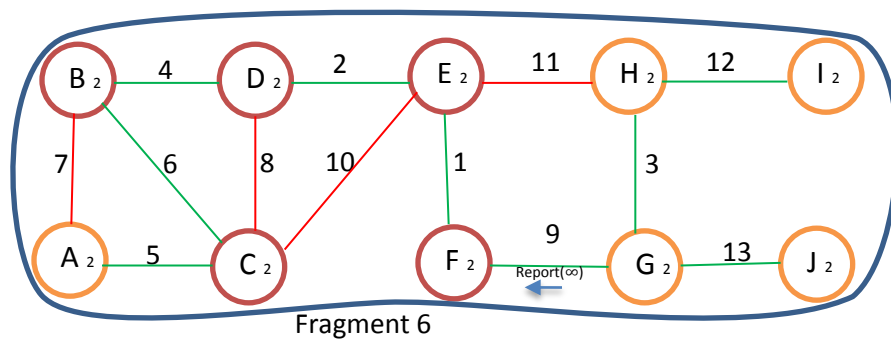


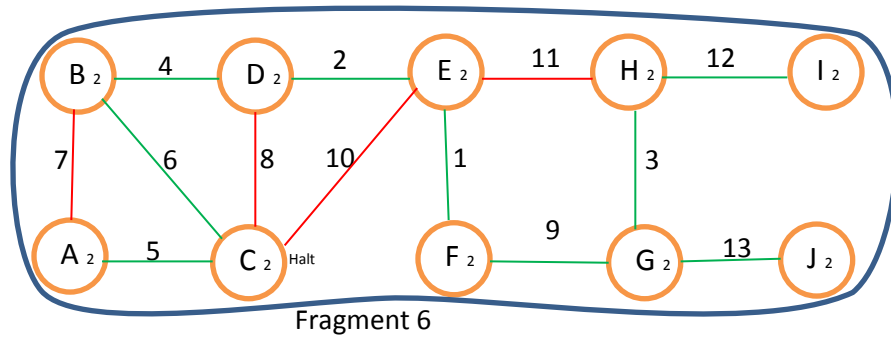
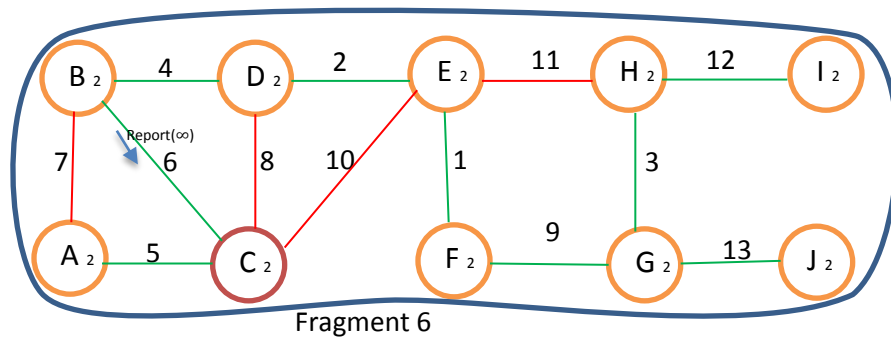
Leaf nodes send their reports with maximum numbers.



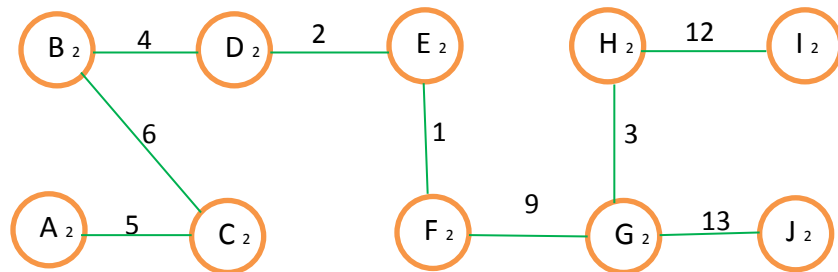
Each node forwards Report(∞) toward root and changes its state to found.





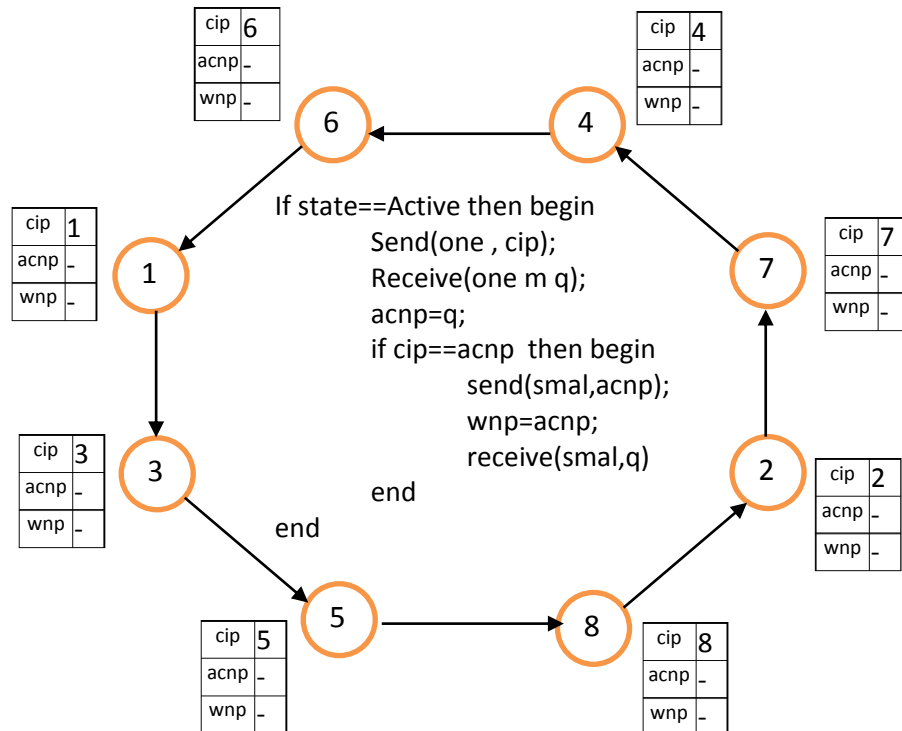


Final spanning tree:

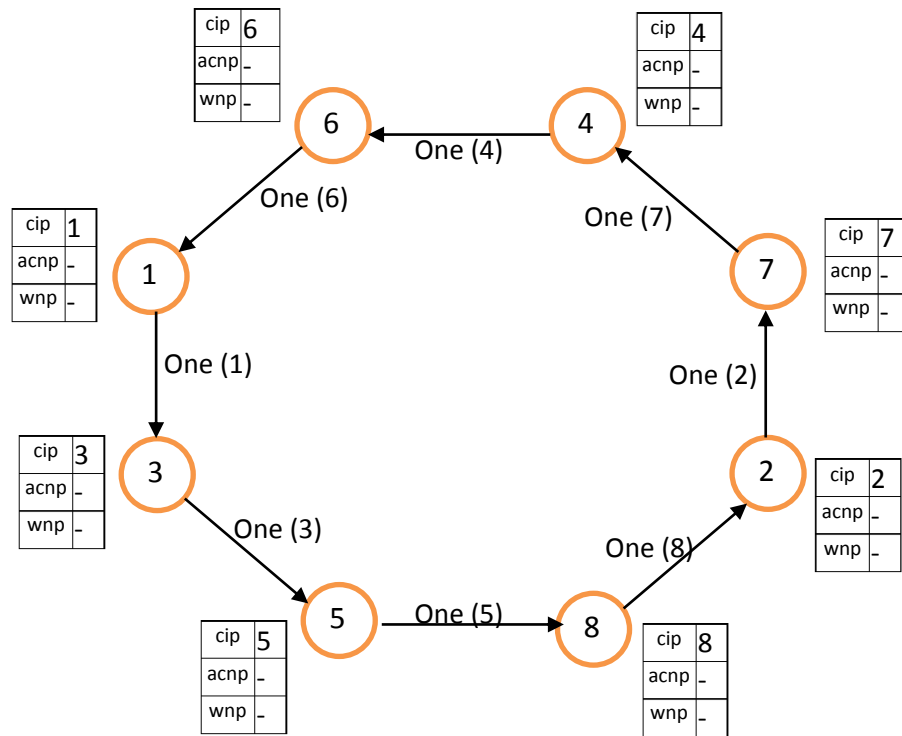


Question 2 (33 pts): Show an example trace of asynchronous Petersen leader election algorithm (Given in Gerard Tel's book) by giving all message transmissions and active/passive nodes step by step.

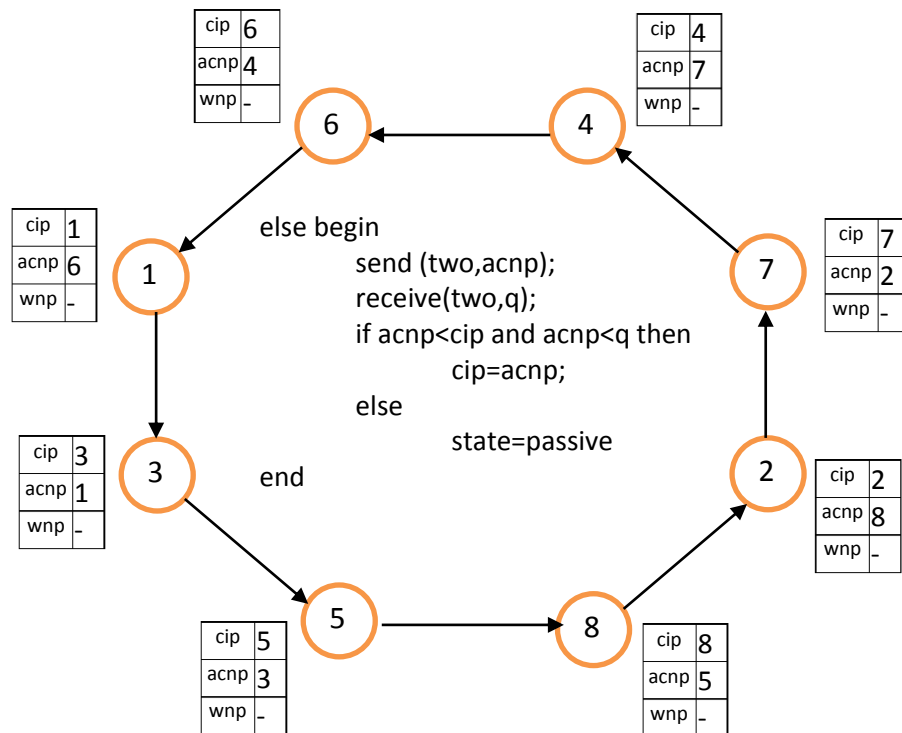
In this algorithm each process holds three variable named *cip*, *acnp* and *wnp*. The *cip* initially is process identifier. *acnp* is the identifier of first anti clockwise active process and *wnp* is the identifier of winner process which is determined at the end of algorithm. Algorithm continues until the value of *wnp* is determined. Here is the initial situation in this algorithm.



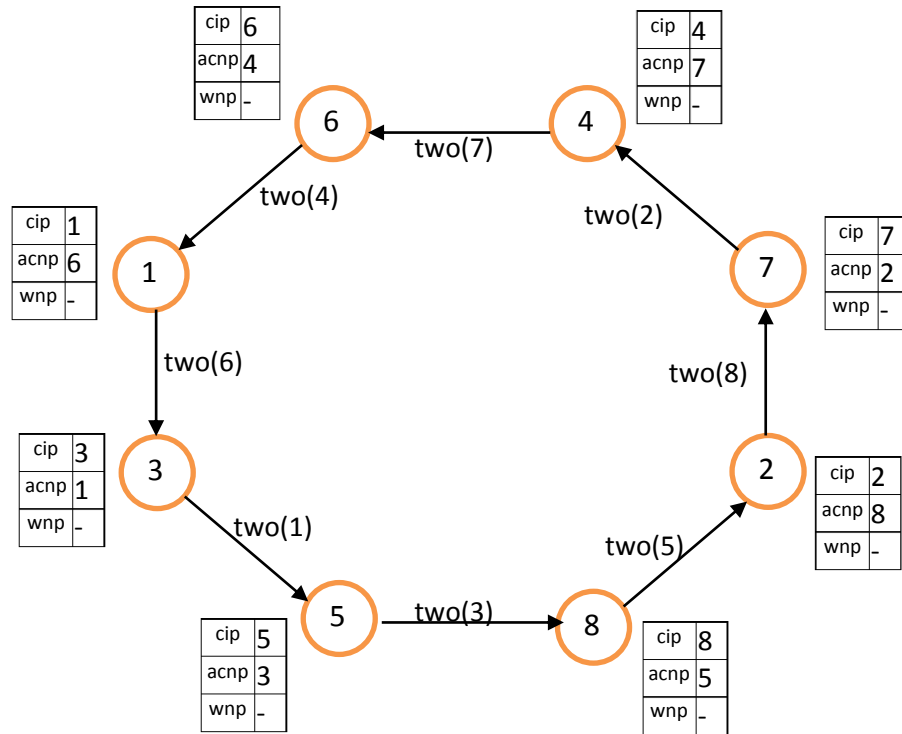
As we can see in the algorithm at the first phase each process sends its *cip* to its left neighbor and waits for incoming message from right neighbor. The incoming message brings the identifier of that neighbor. This value is stored in *acnp* variable. In this phase, messages are exchanged between processes as the following figure.



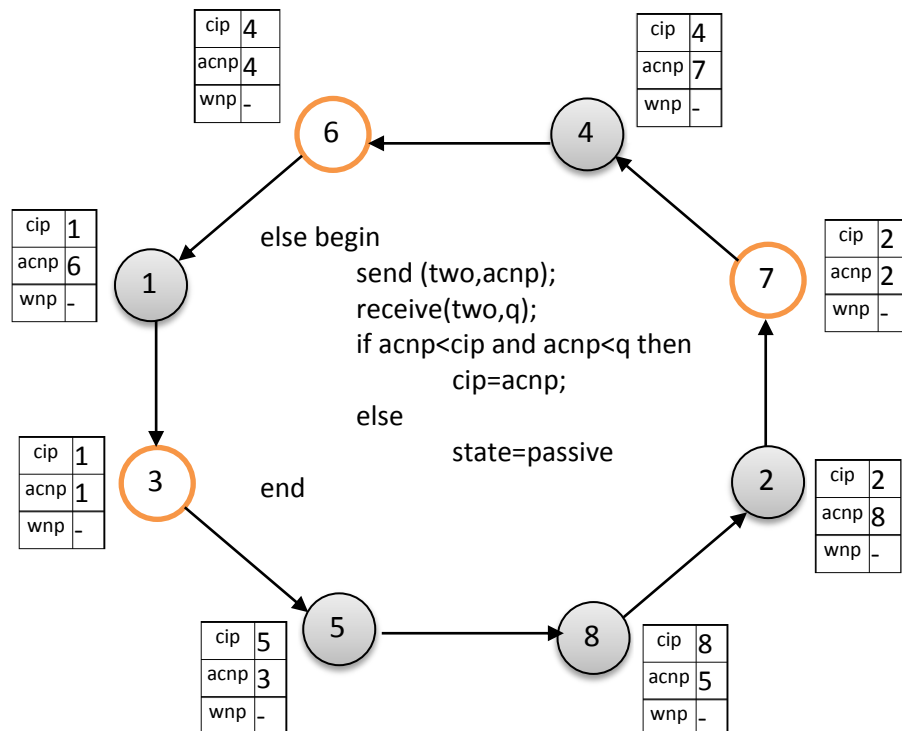
After receiving messages each process update its *acnp* as the following figure:



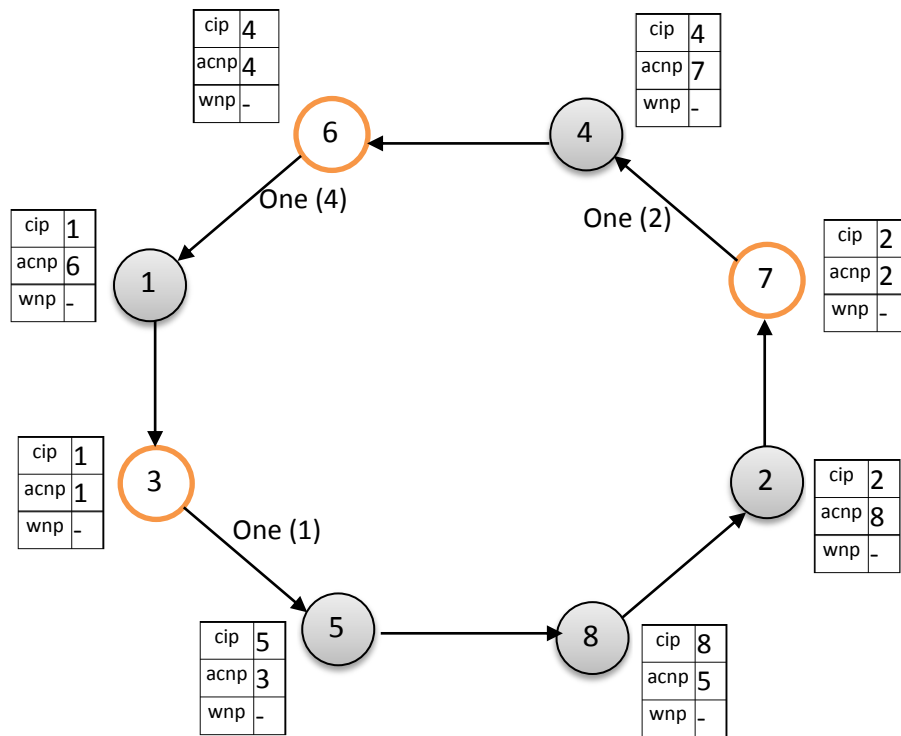
After updating *acnp* values all nodes send (two,acnp) messages to their neighbors .



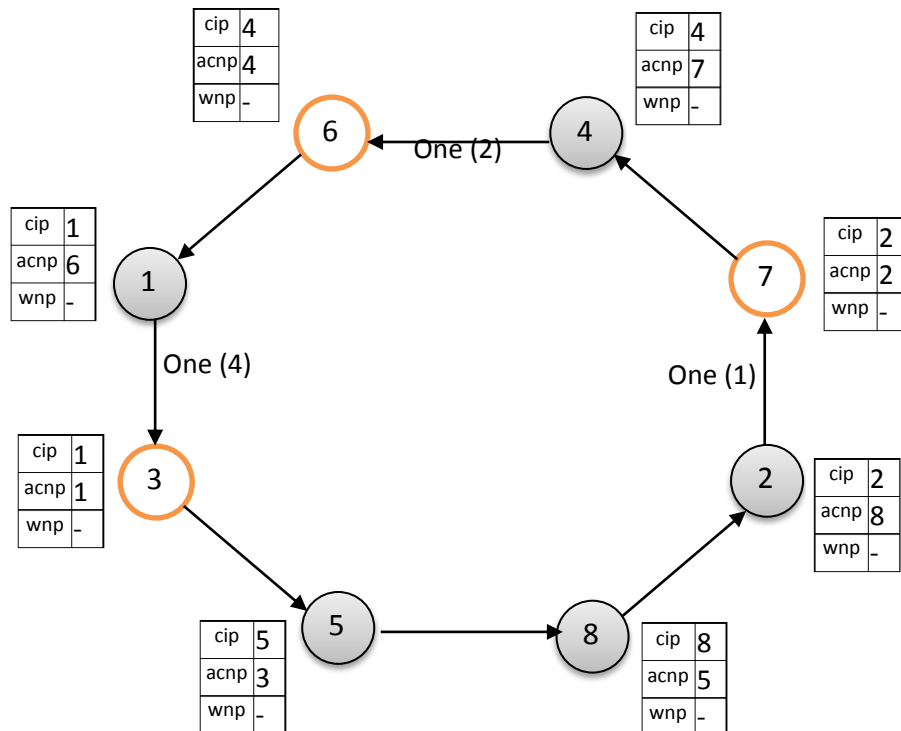
After receiving these messages, each process which its *acnp* is smaller than *cip* and received value, is remains active and other processes goes to passive state. The passive nodes are show with the gray color in the following figure.



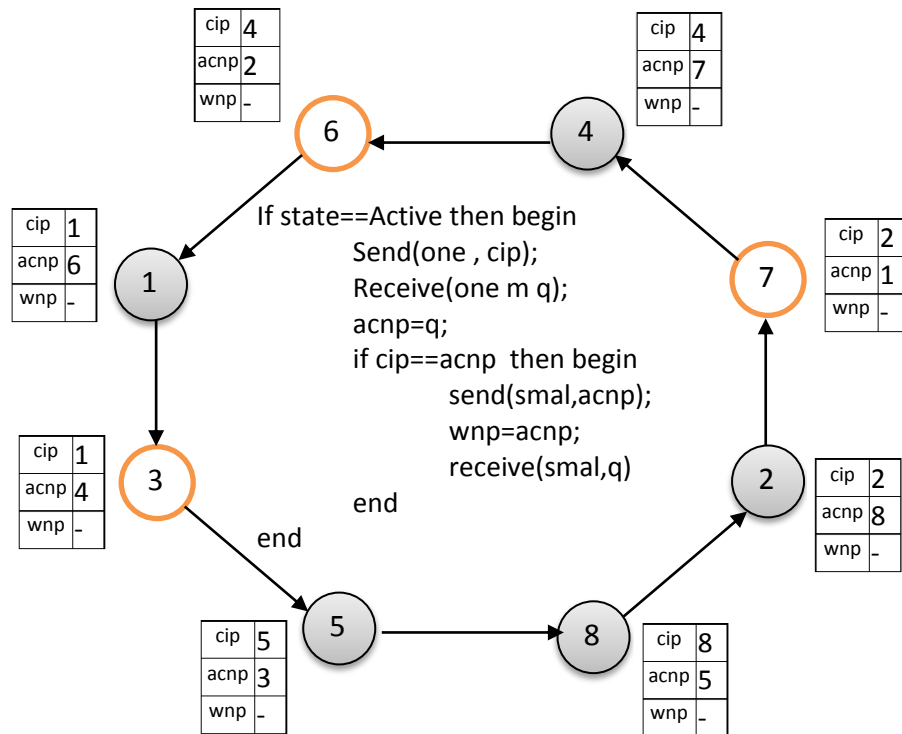
The active processes start a new round and send (One, cip) to their neighbors .



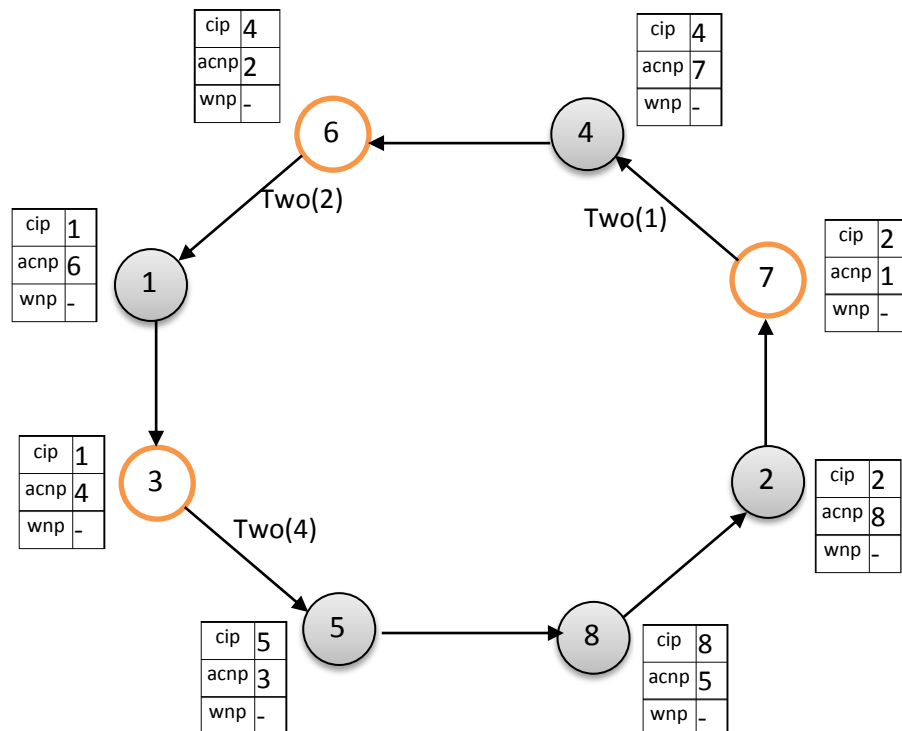
Passive nodes just forward incoming messages to next neighbors.



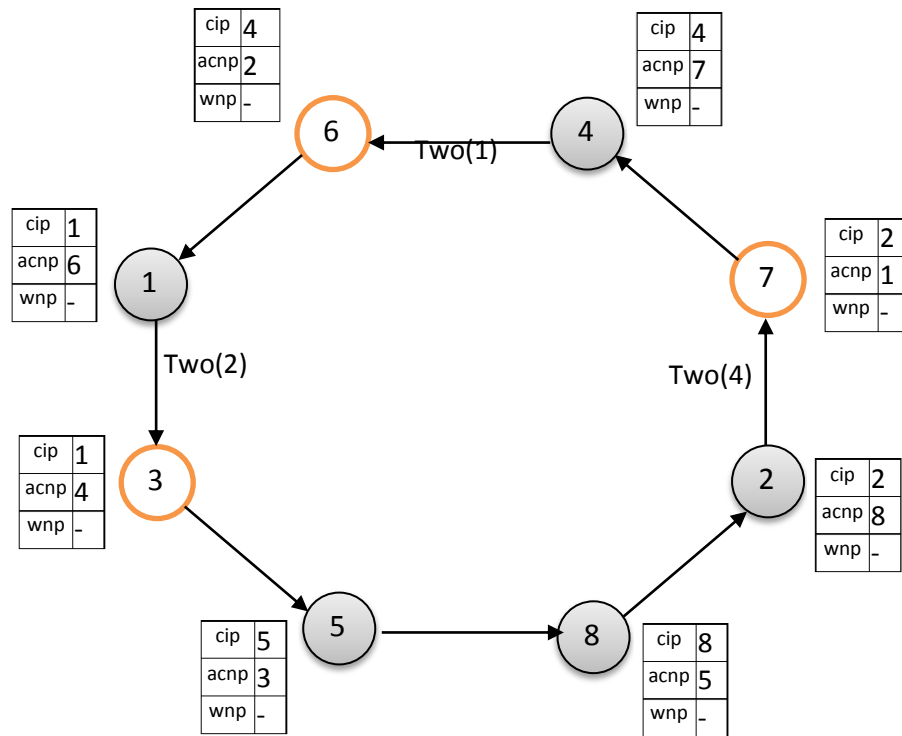
upon receiving (one , q) in active nodes, Their *acnp* values update with received q.



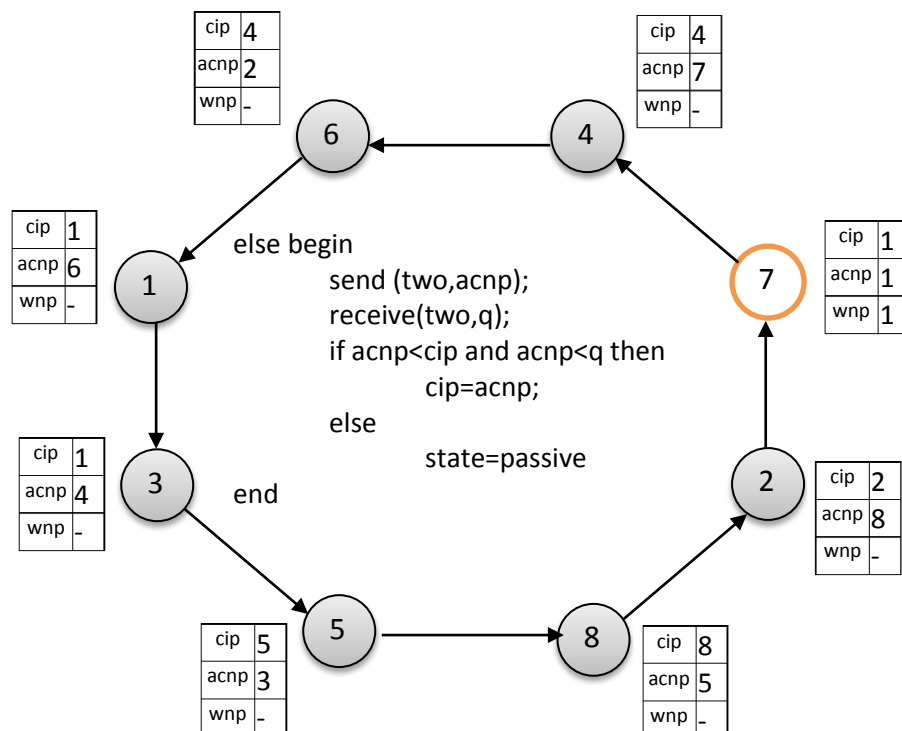
Again (two,acnp) messages are sent by active process.



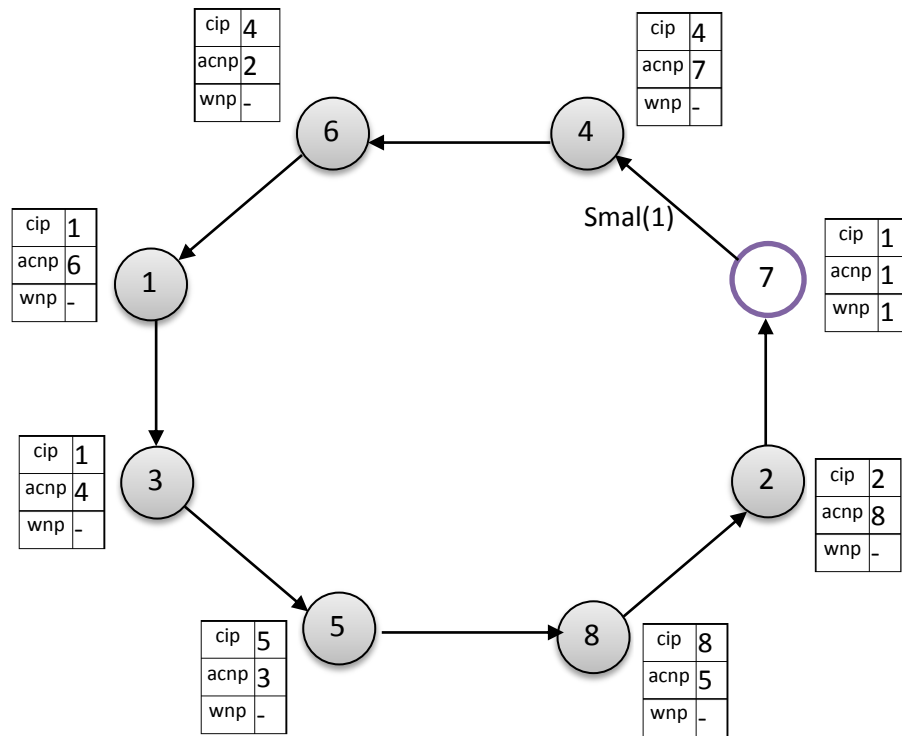
These messages are forwarded by passive nodes to active nodes.



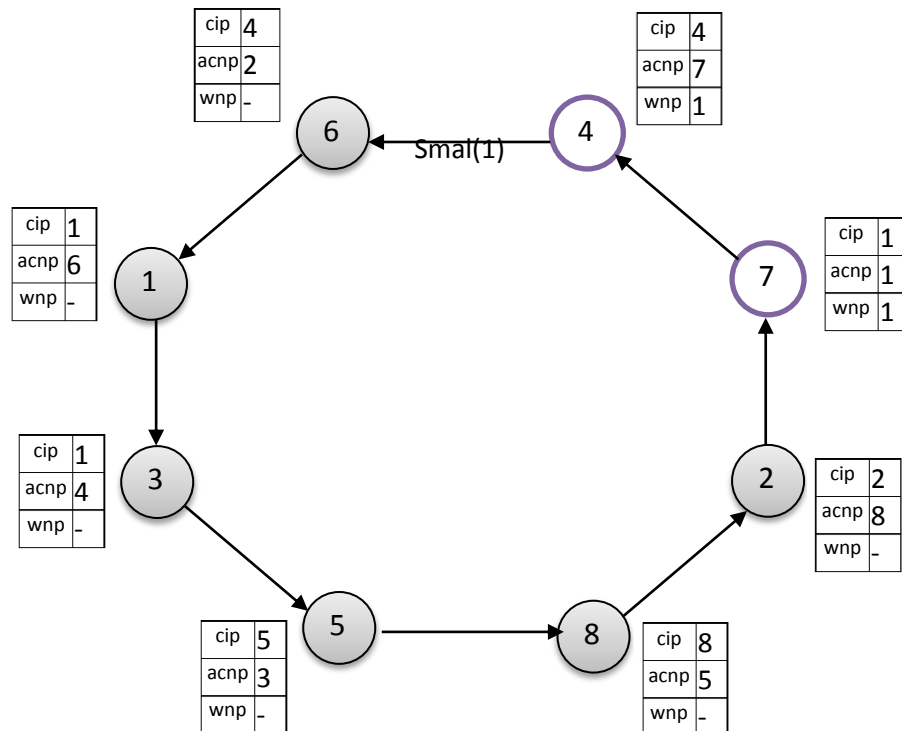
Node 6 has the bigger *acnp* than incoming *q* and node 3 has the bigger *acnp* than *cip* hence both nodes go to passive state. Now the only active node is 7.



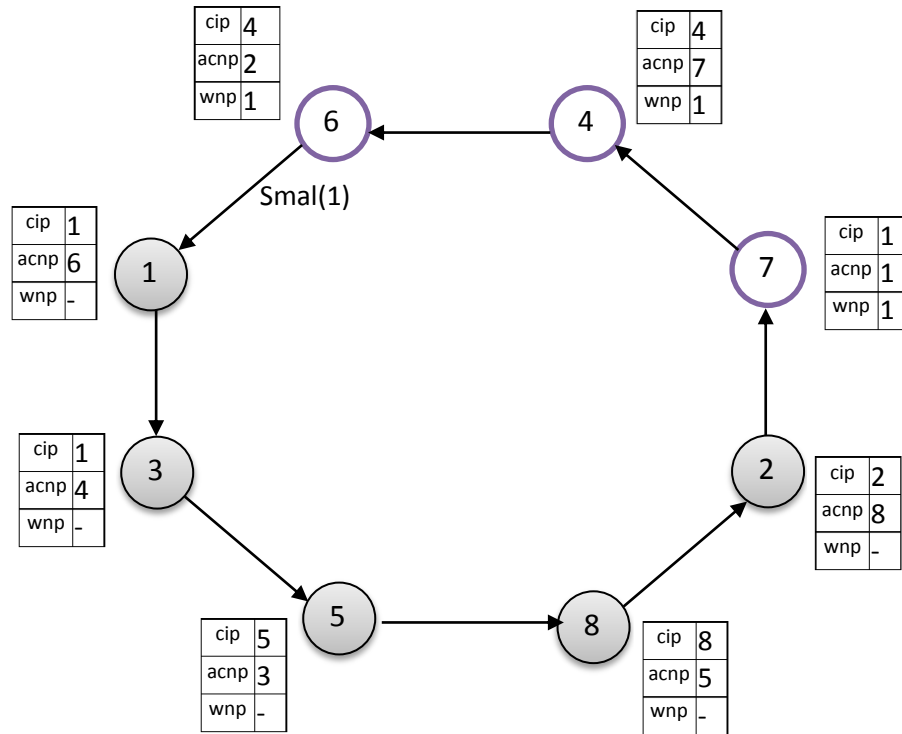
In node 7 the value of cip and acnp are equals so the winner can be selected in this node.



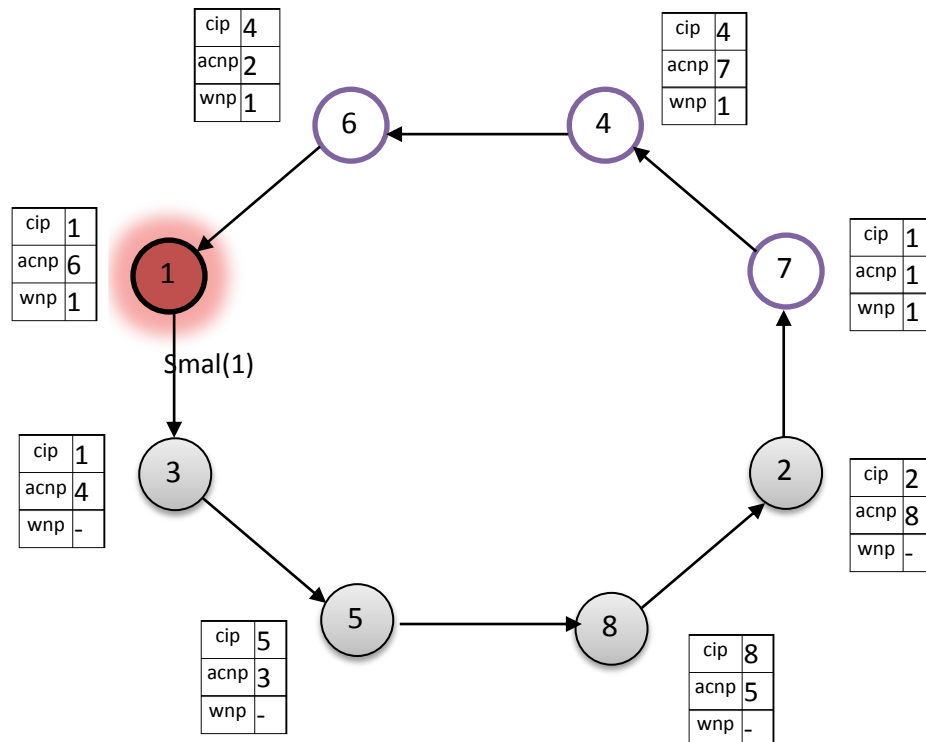
The winner identifier is announced to all nodes using a (small,wnp) message.



Upon receiving the (small,wnp) message, each node terminate algorithm execution.



Also the winner changes its state to *leader*.



Finally all nodes terminate their execution and node 1 becomes leader.

