Distributed Systems
COMP 212

Lecture 9

Othon Michail

UNIVERSITY OF LIVERPOOL

Leader Election

Problem Statement

- *Elect a unique leader processor from among all the processors in the distributed system*
- Leader to be interpreted as:
  - coordinator
  - master processor
- Special case of consensus/agreement
- Processors should agree eventually on who they elect

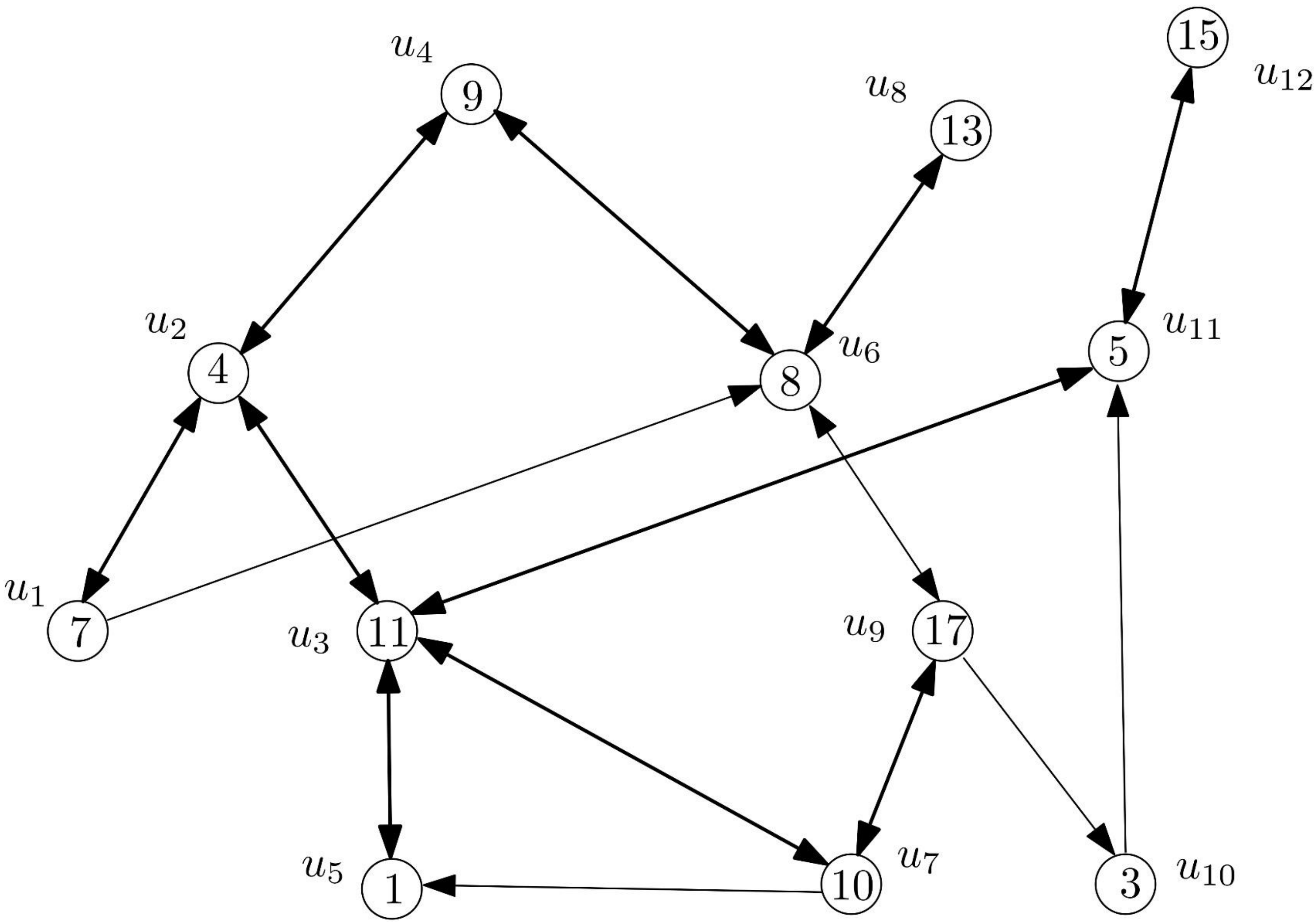
Beyond Rings
Leader Election in General Networks

Leader Election in General Networks

- *Elect a unique leader processor from among all the processors in the distributed system*
- Now the network can be any strongly connected directed network
  - Strongly connected: For every processors u, v there is a path from u to v and a path from v to u
  - e.g., the directed ring is just a special case
  - *Why can't we use LCR in this case?*
- Processors have unique ids

Leader Election in General Networks

- A strongly connected directed network



A Simple Algorithm based on Flooding

- Processors have unique ids, do not know *n* in advance, but do know the diameter *D* of the network
  - Diameter:
    - the distance between two nodes is given by the shortest path between them
    - Then the diameter of the network is determined by the pair of nodes at maximum distance (and is equal to that distance)
    - In other words, it is the maximum shortest path in the network
- FloodMax algorithm: solves the problem
- Uses transmission, comparison, and storage of ids
- Main idea: Flood the maximum id
  - LCR also does something like this but does not require knowledge of *D* and its termination condition works only for rings

FloodMax: Informal description

- All processors know the diameter *D* and their own id in advance
- All processors remember the greatest id that they have "heard" so far (initially their own)
- In every round all processors send the greatest known to all their out-neighbours
- After *D* rounds compare the largest heard to your own
  - if *greatest heard = own id*, declare yourself the leader
  - otherwise, declare yourself non-leader
- Intuitively:
  - The maximum id will manage to reach the whole network
  - So everyone non-maximum will know that there is a greater id and u_{max} can never receive a larger id

FloodMax: Pseudocode

Algorithm FloodMax

State of processor u_i :

- *myID$_i$*: holds the processor's unique id
- *maxID$_i$*: holds the greatest id "heard" so far
- *status$_i$* ∈ {"unknown", "leader", "non-leader"}: indicates whether u_i has been elected ("leader"), not elected ("non-leader") or doesn' know yet ("unknown")

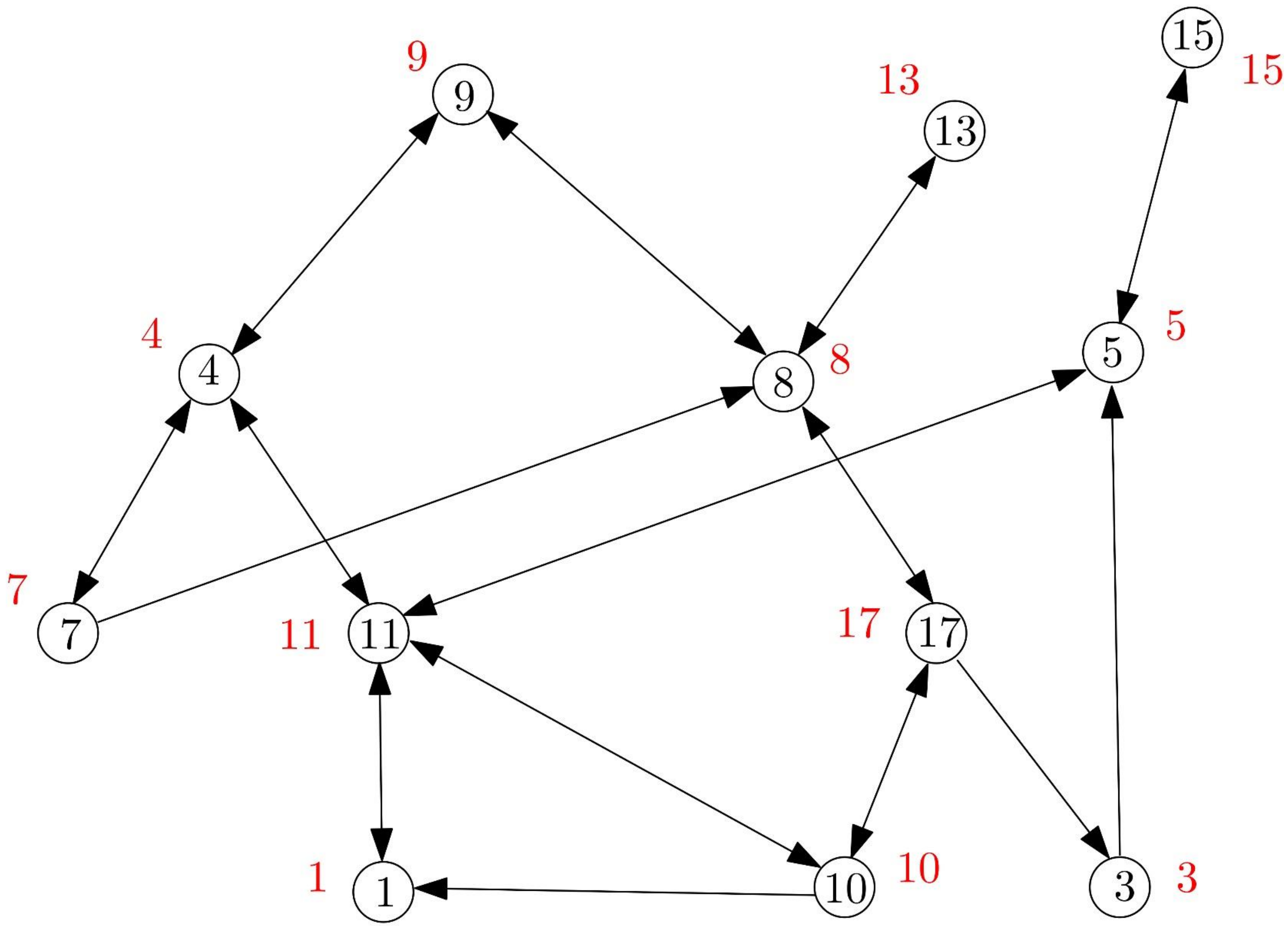
FloodMax: Pseudocode

Algorithm FloodMax
Code for processor u_i , i ∈ {1, 2,… , n}:

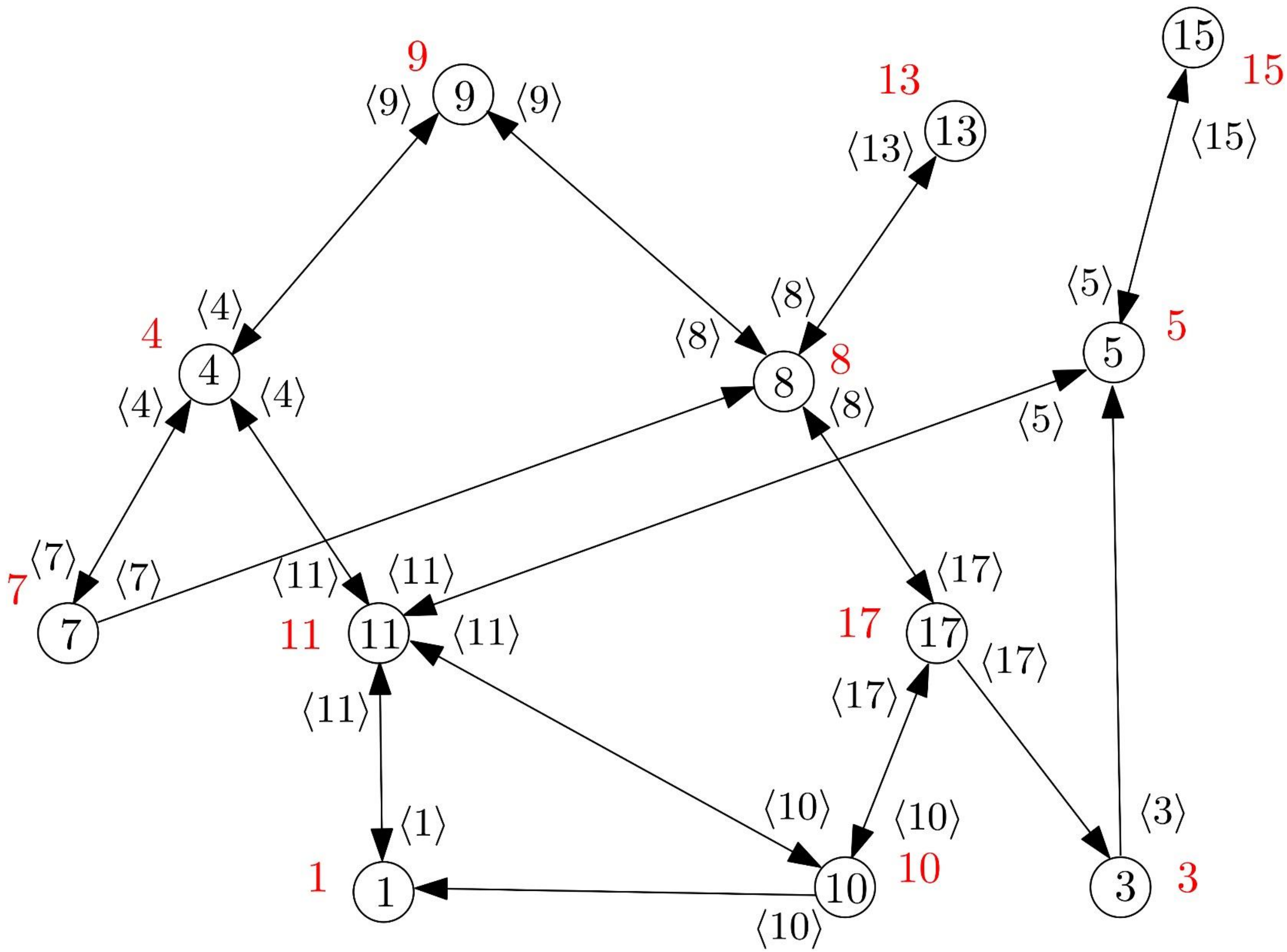
```
Initially:
   u_i knows its own unique id stored in myID_i
   maxID_i := myID_i
   status_i := "unknown"
   Also has access to the current round and knows the diameter D

if round = 1 then
   send ⟨maxID_i⟩ to all out-neighbours
else
   upon receiving ⟨inIDs⟩ from in-neighbours      // one or more ids arriving from neighbours
   maxID_i := max({maxID_i} ∪ inIDs)              // remember only the maximum "heard" so far
   if round ≤ D then    // 1 < round ≤ D
      send ⟨maxID_i⟩ to all out-neighbours
   else    // round = D + 1
      if maxID_i = myID_i then         // if equal to your own, no greater id exists in the network
         status_i := "leader"          // therefore, elect yourself a leader
      else                             // greater than own
         status_i := "non-leader"      // therefore, declare yourself a non-leader
// observe that in the end all processors know the id of the elected leader, stored in their maxID_i
// variable
```

Example Execution

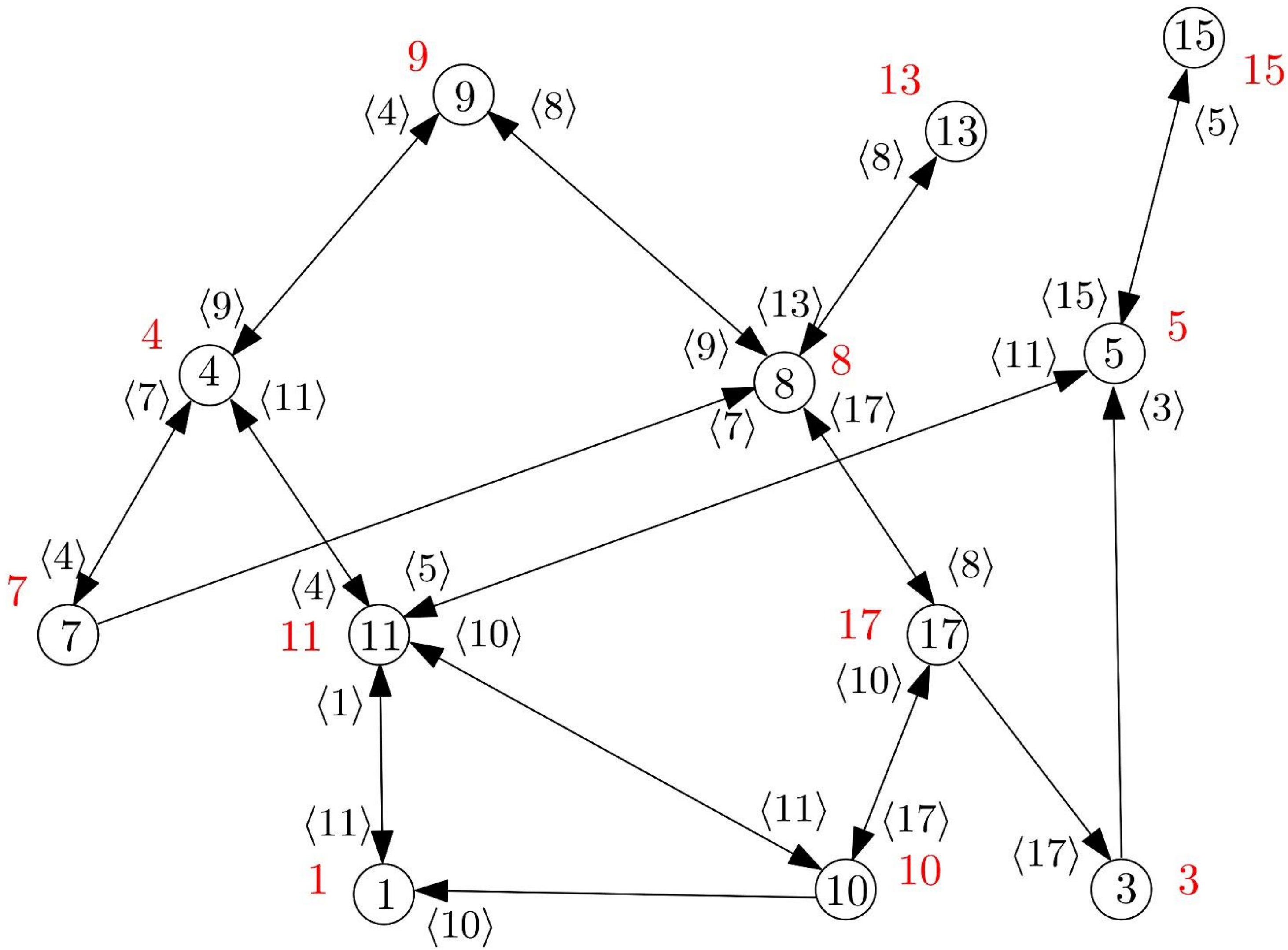


Example Execution



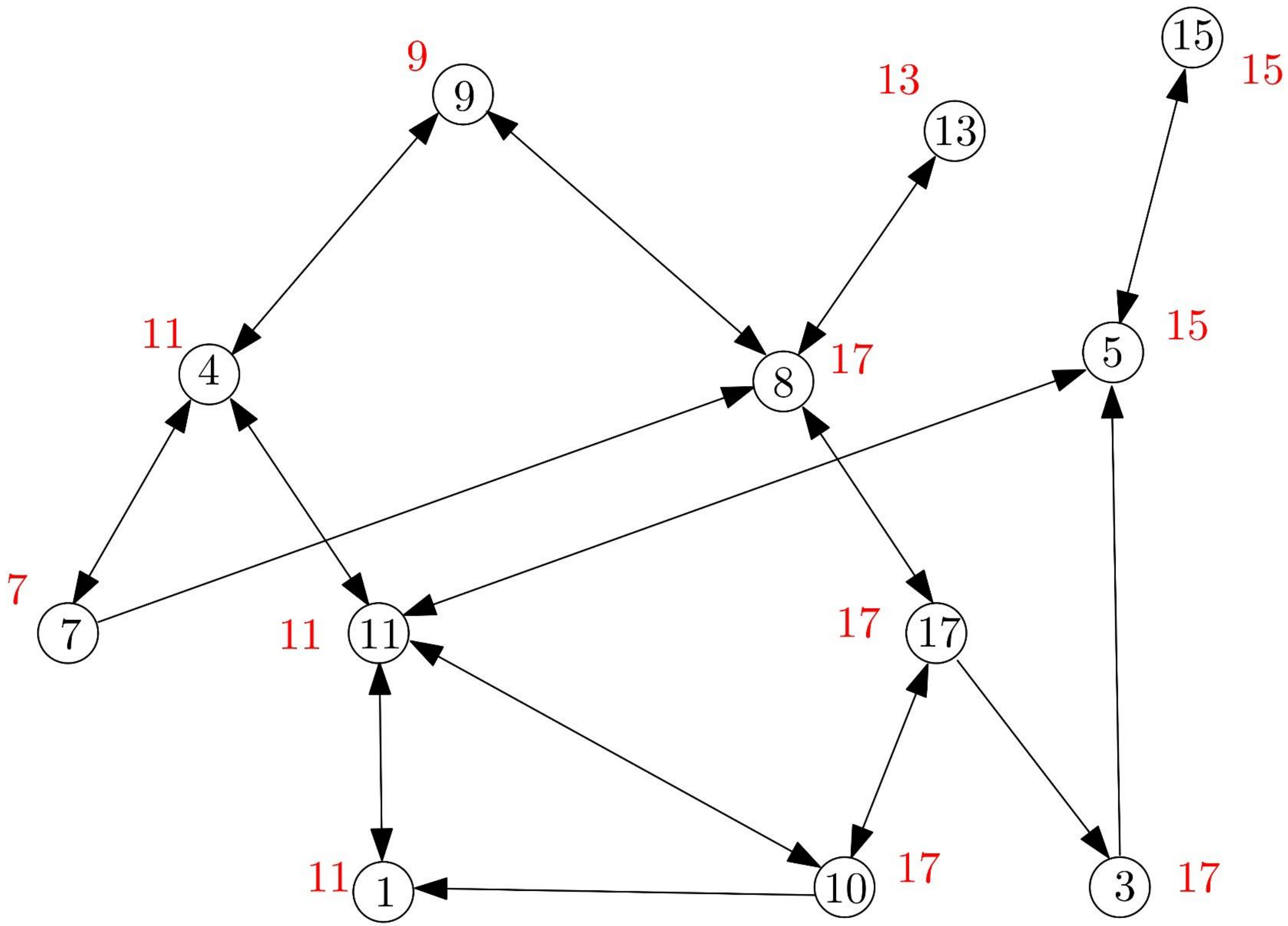
$round = 1$

Example Execution



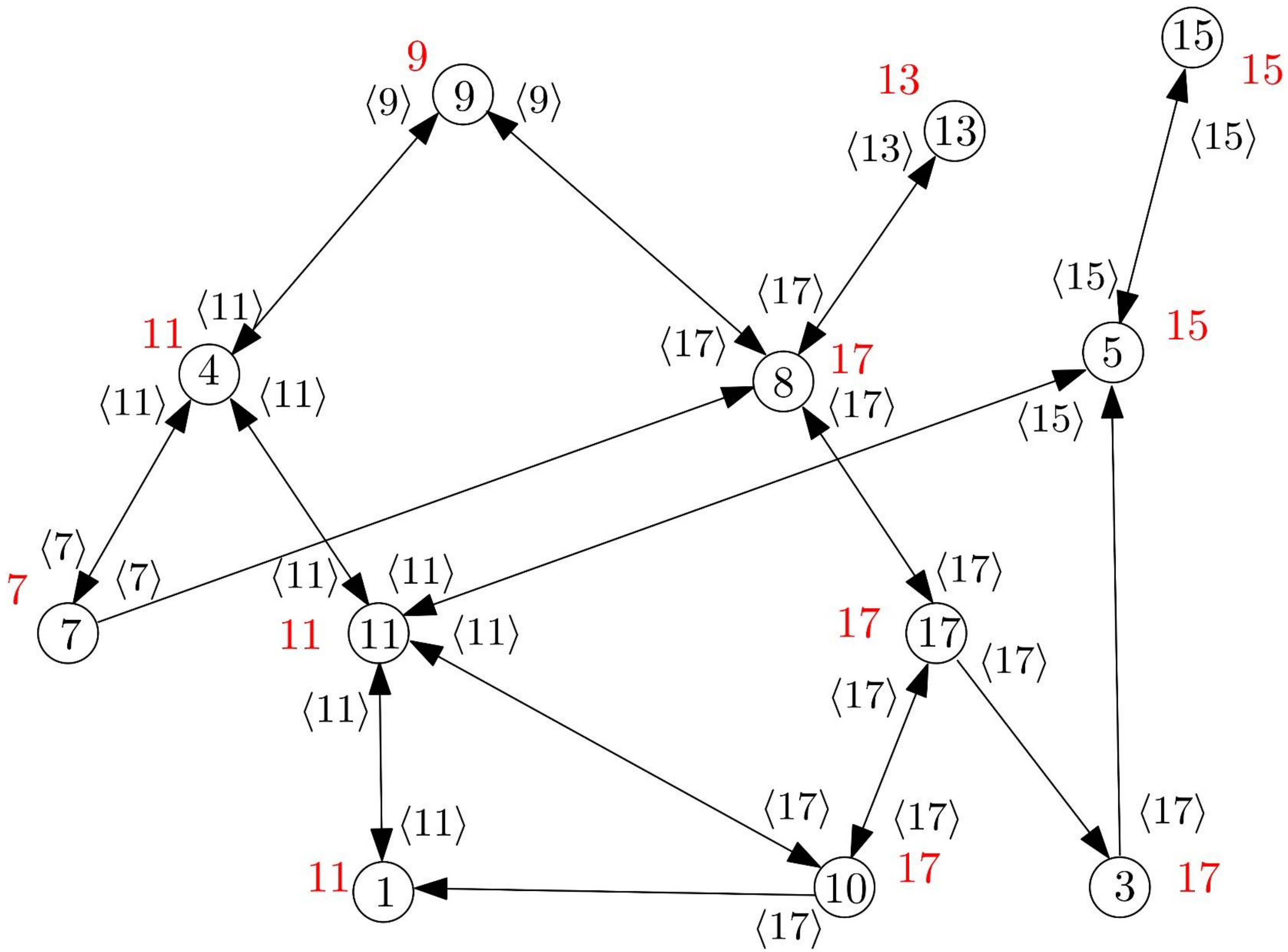
$round = 1$

Example Execution



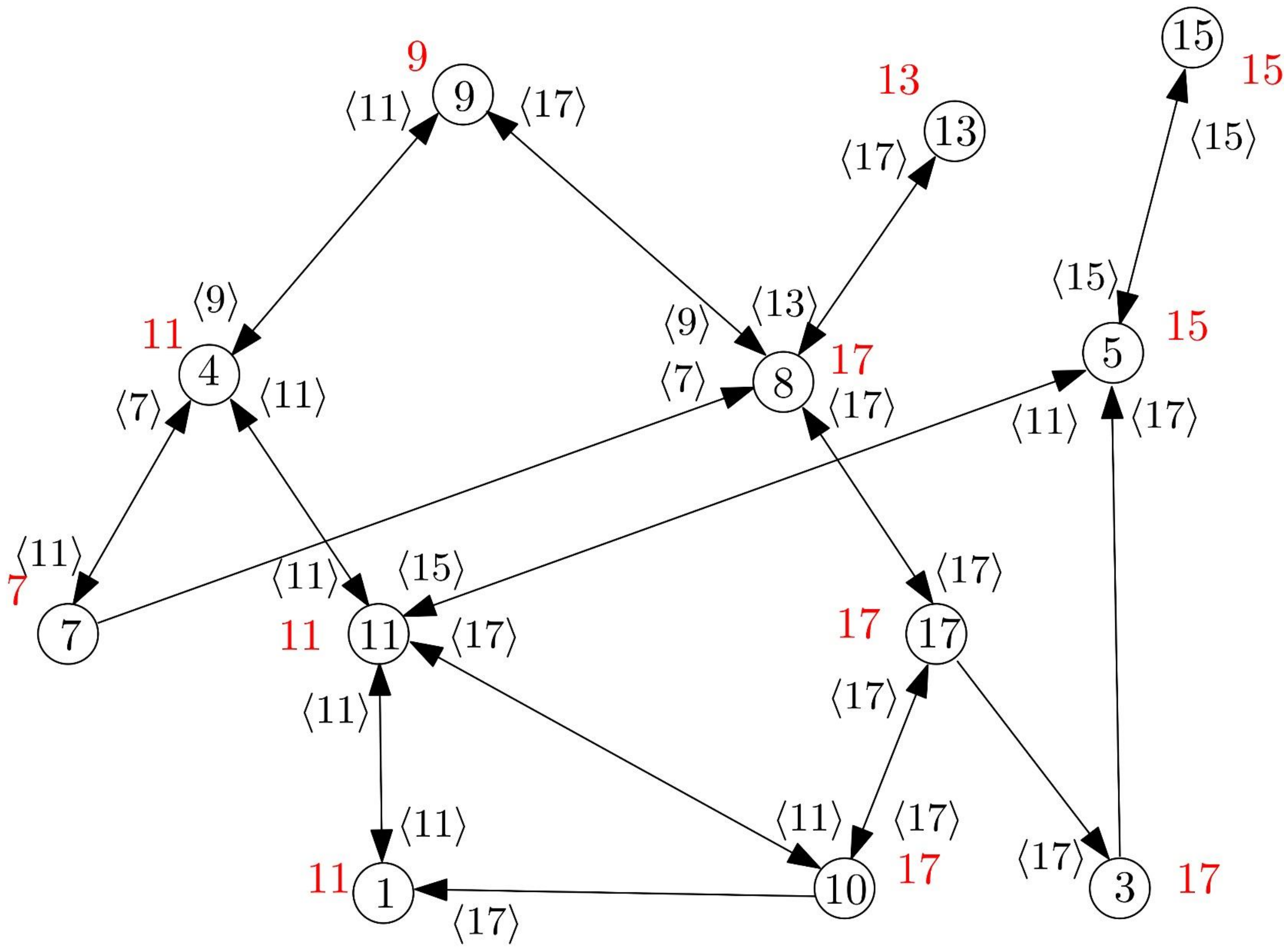
$round = 2$

Example Execution



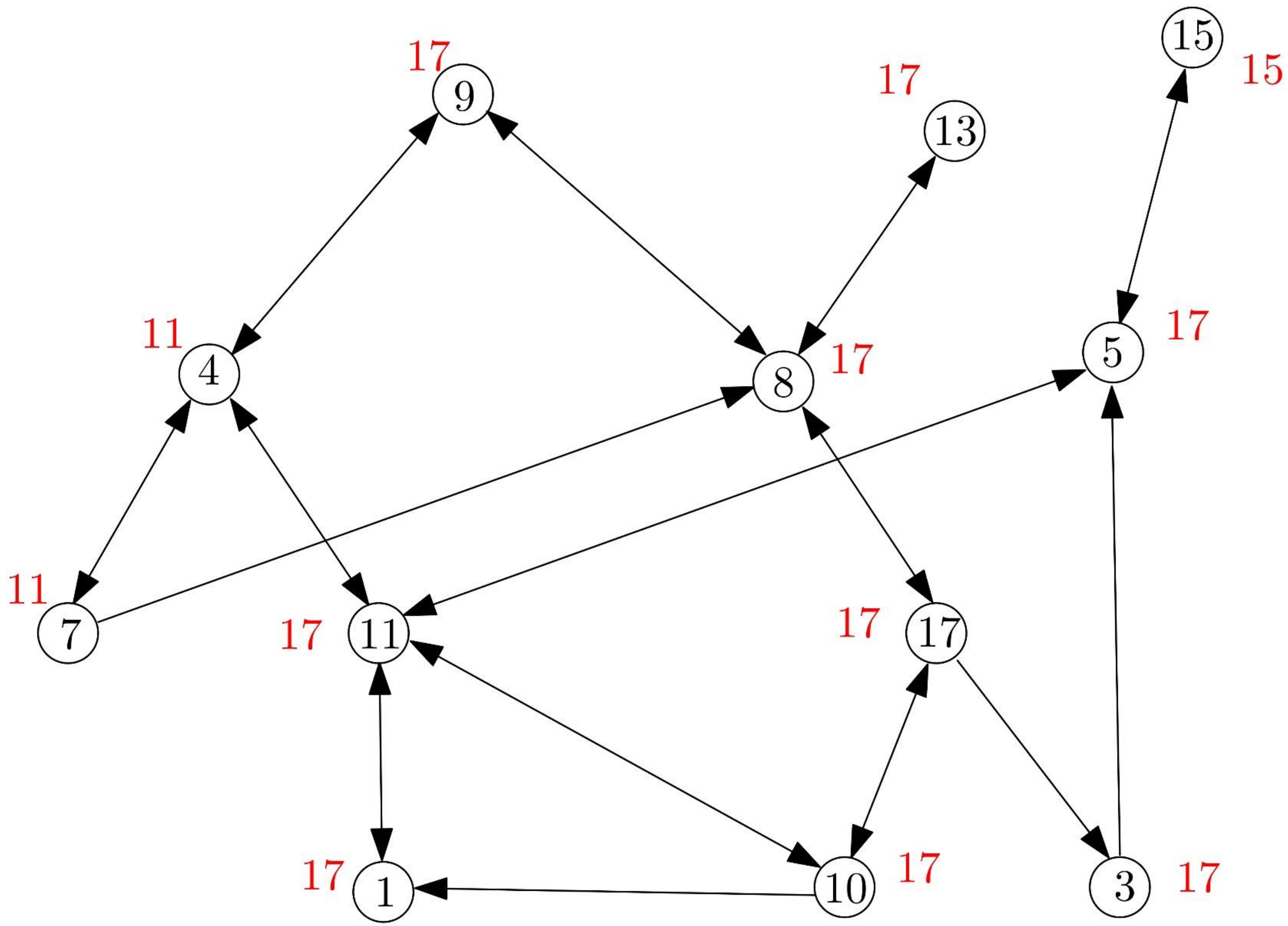
$round = 2$

Example Execution



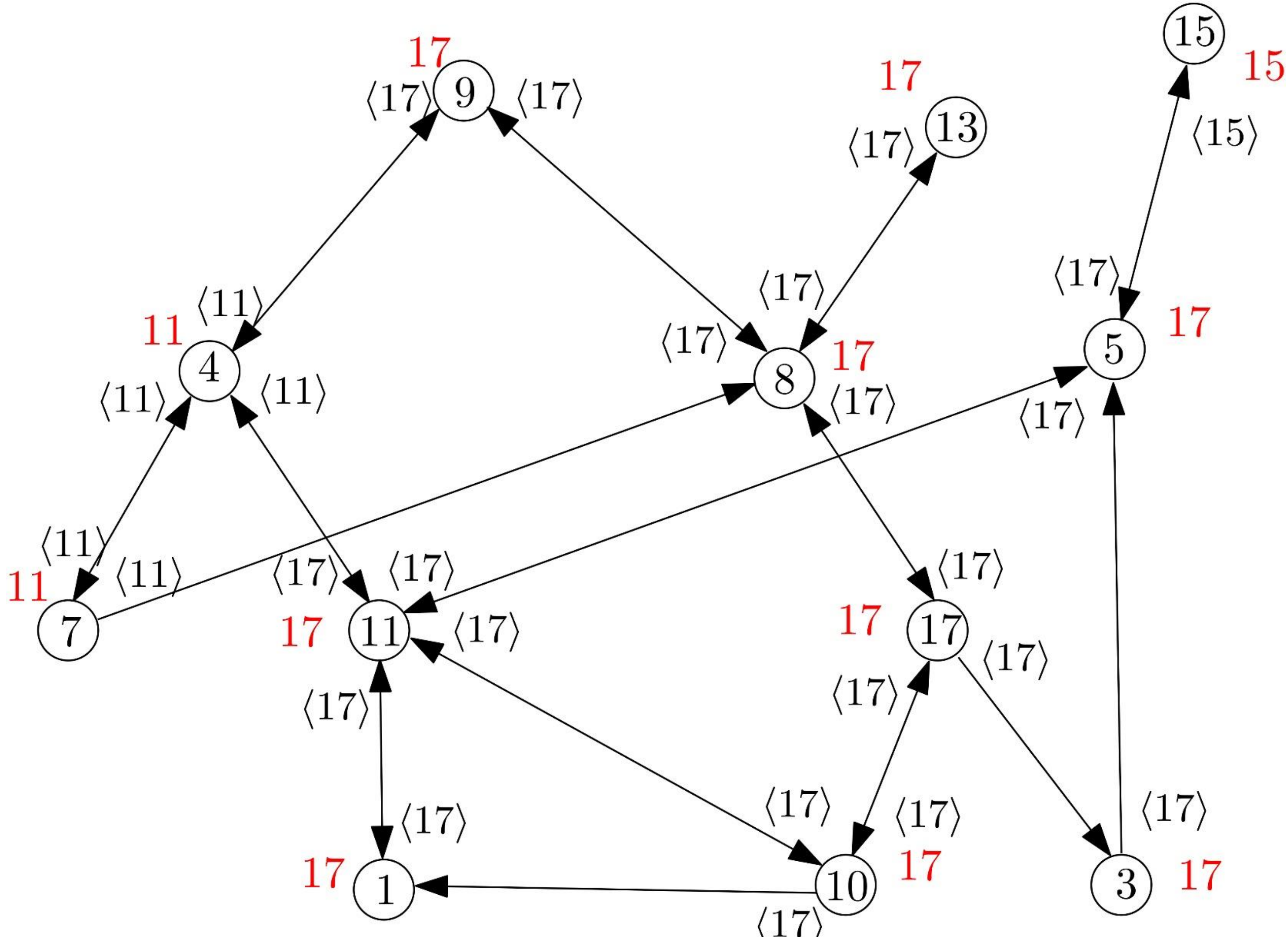
$round = 2$

Example Execution



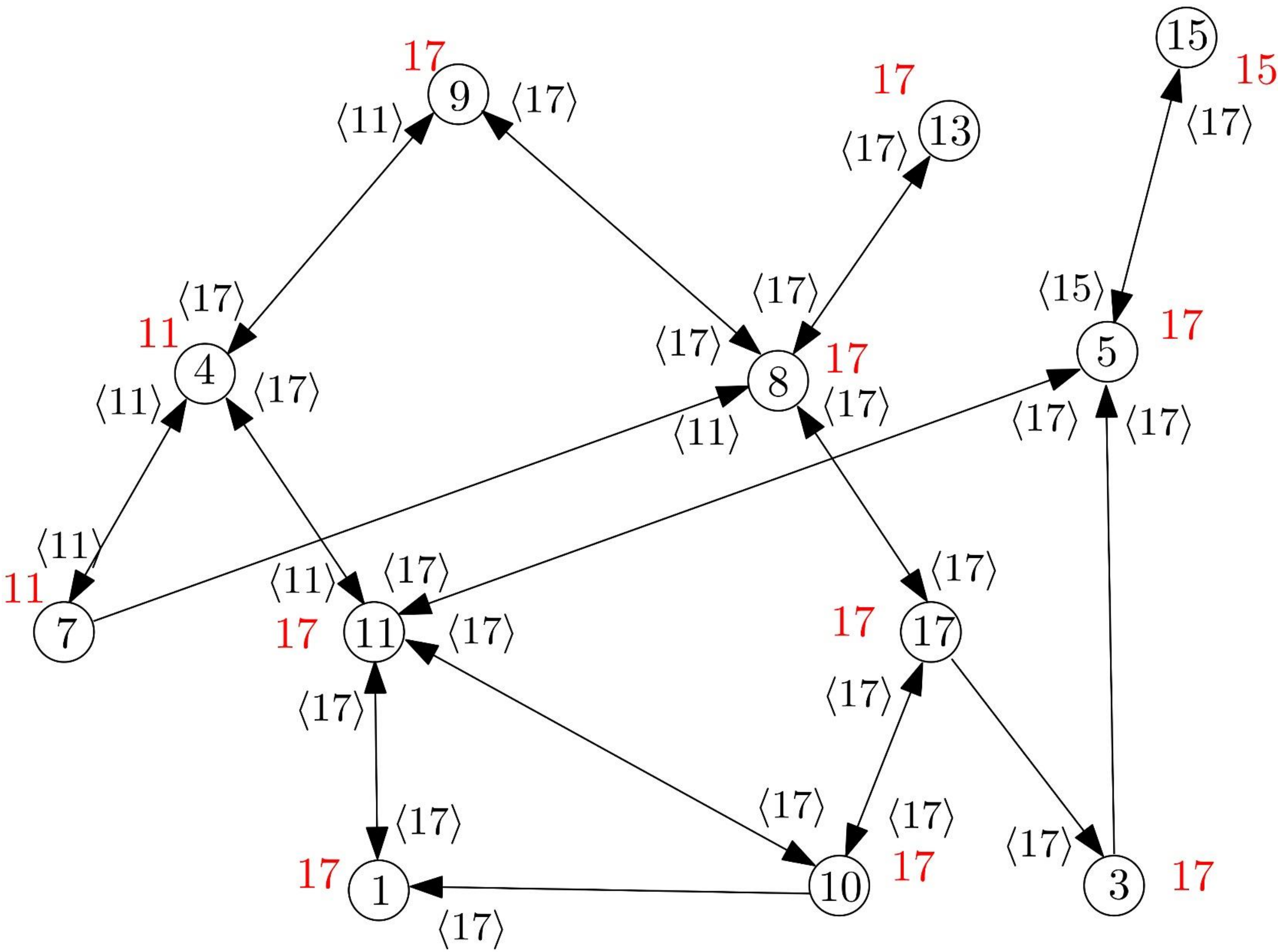
$round = 3$

Example Execution



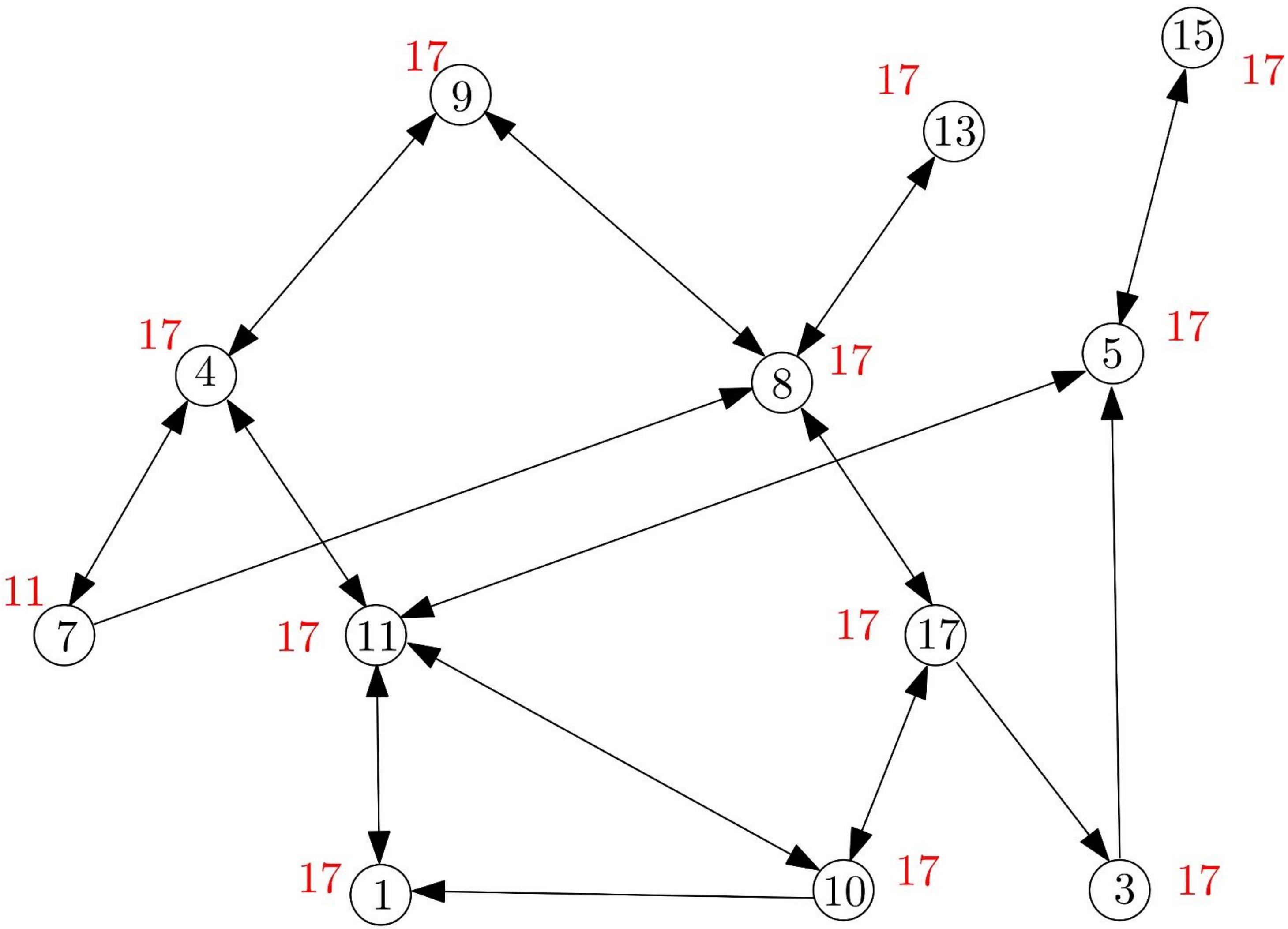
$round = 3$

Example Execution



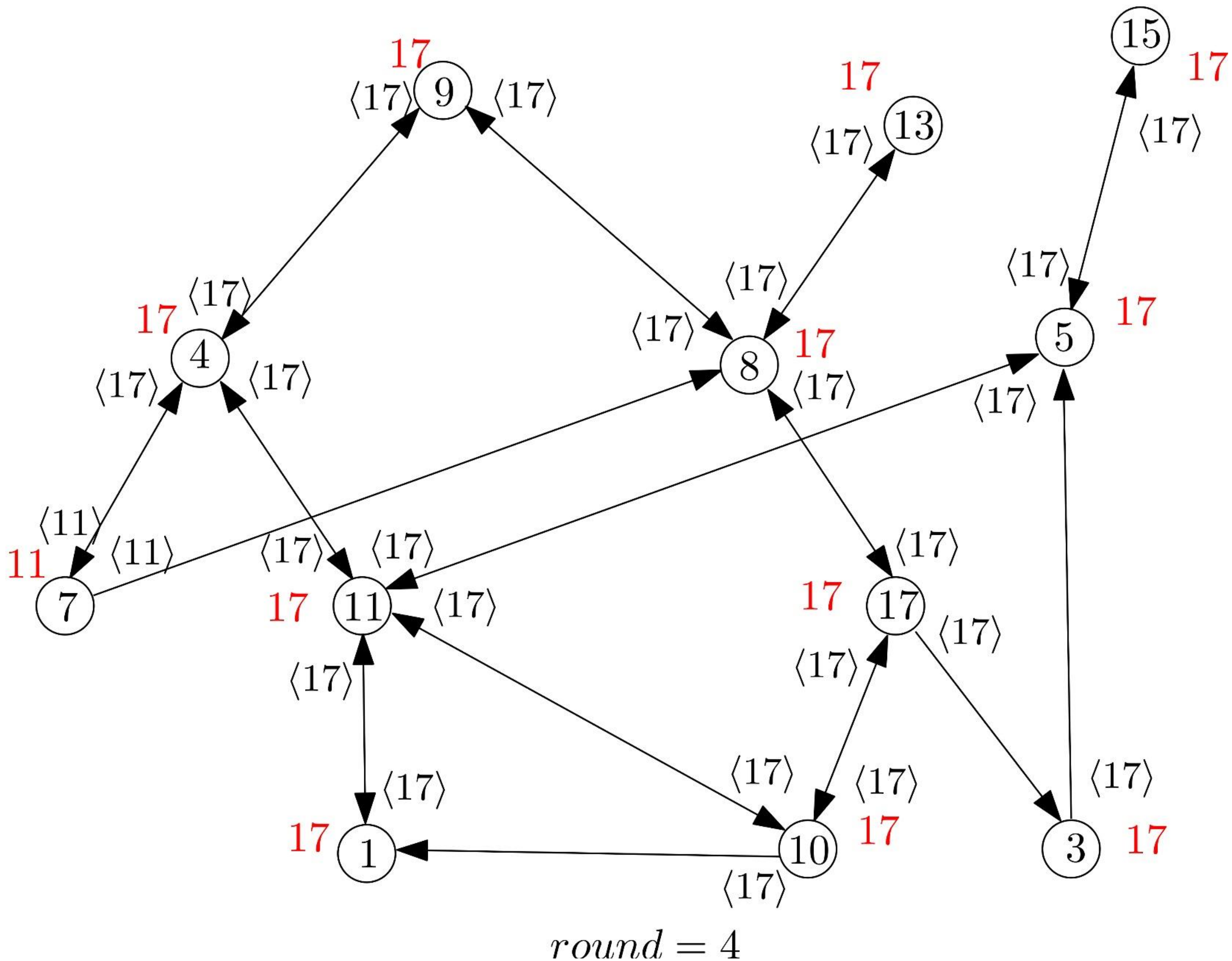
$round = 3$

Example Execution

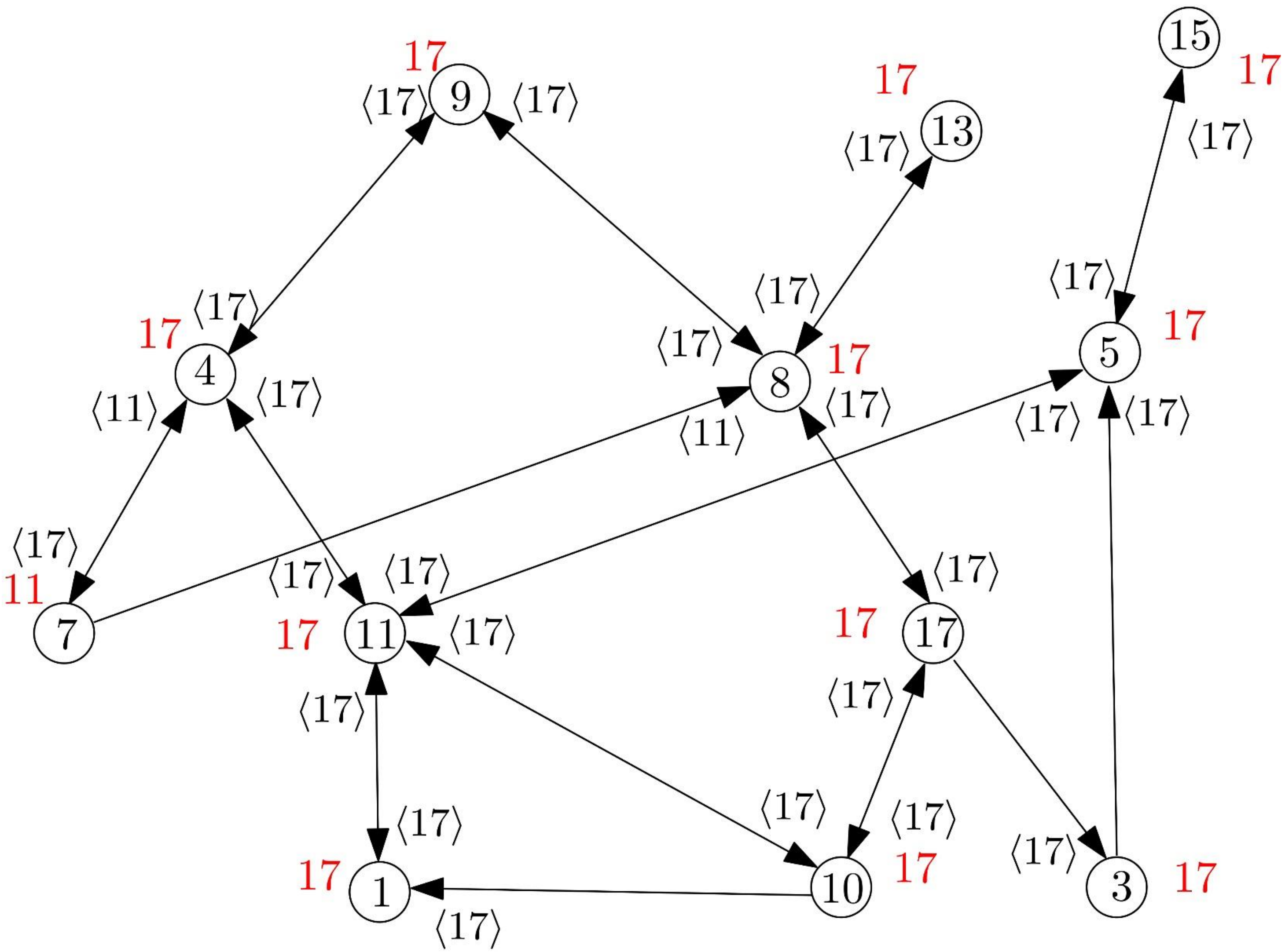


$round = 4$

Example Execution

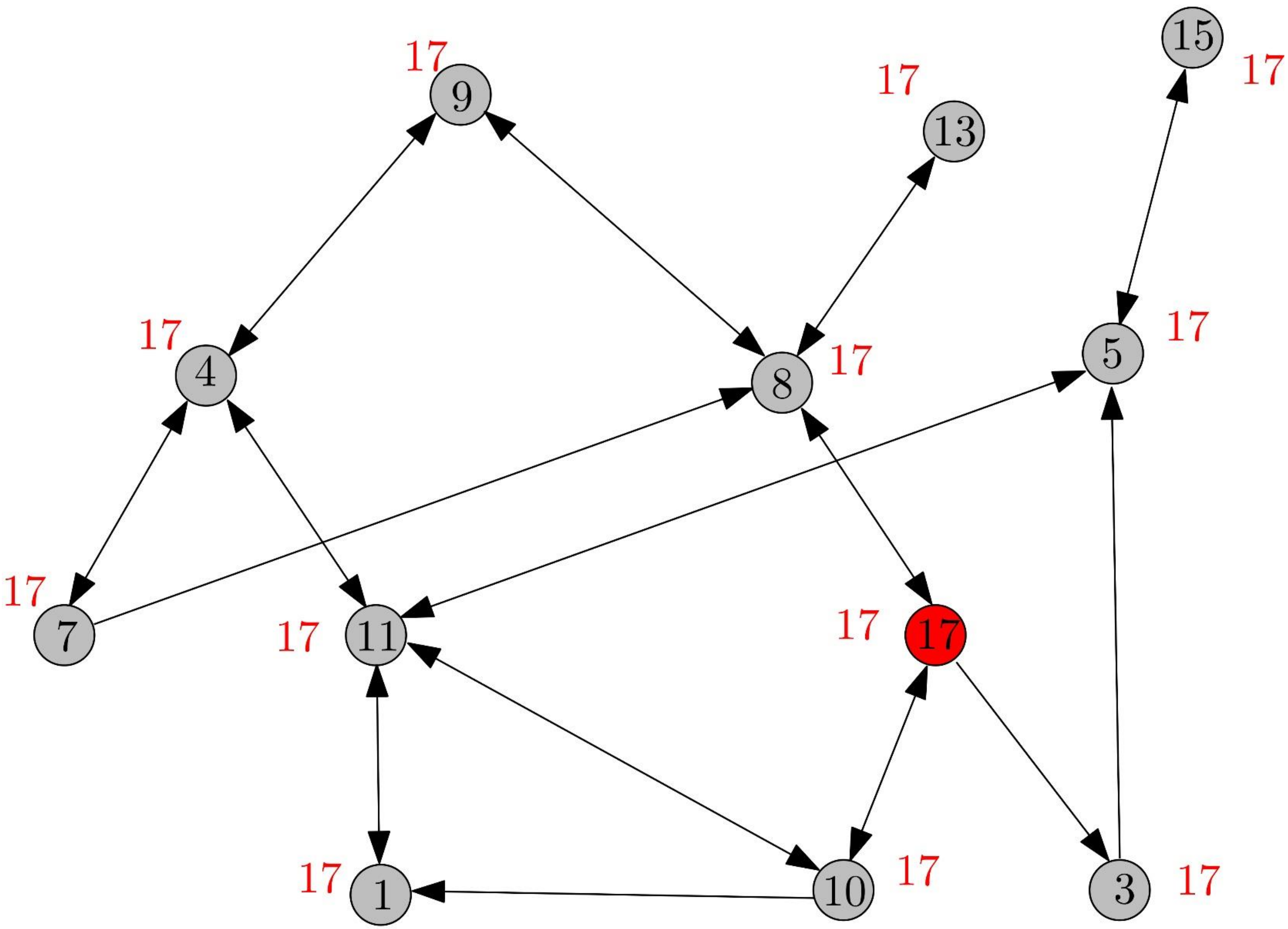


Example Execution



$round = 4$

Example Execution



$round = 5$

Correctness and Complexity

- Correctness:
  - exactly one processor is elected in the last round
- Time complexity:
  - $D + 1$ rounds (or D depending on the round model)
- Communication complexity:
  - size of messages: encoding in bits of the maximum id
  - $D \cdot m$ messages always
    - m is the number of directed links in the network

FloodMax Correctness

- We have to show that:
  - Exactly one processor u_i sets $status_i$:= "*leader*" in the last round
  - We know that the algorithm aims to elect the processor with the maximum id
    - Call it u_{max}
- Suffices to show that:
  - u_{max} outputs "*leader*" in the last round
  - Every other u_i outputs "*non-leader*" in the last round

FloodMax Correctness

Lemma. u_{max} outputs "*leader*" in round D + 1.

Proof. Trivial.

- We know that id_{max} (i.e., the id of u_{max}) is the greatest id in the network
- Therefore, in every round it will hold at u_{max} that
  - *maxID = myID*
  - as u_{max} will never hear an id greater than its own
- So, this will also hold in round D + 1
  - Then *maxID = myID evaluates to "true" at* u_{max}
  - Therefore, u_{max} outputs "*leader*" by setting *status* := "*leader*"

□

  - Essentially, by induction on the number of rounds *r, we show that maxID = myID holds for every r at* u_{max}

FloodMax Correctness

Lemma. Every processor u_i other than u_{max} outputs "non-*leader*" in round $D + 1$.

Proof. It suffices to show that by the beginning of round $D + 1$ every processor u_i has received id_{max} (i.e., the id of u_{max})

- Because then it must hold that in round $D + 1$
- $maxID_i = id_{max} > myID_i$
- and u_i will set $status_i :=$ "*non-leader*"

- We will prove that: In round r, any u_i at distance r from u_{max} receives id_{max}
  - By induction on r

FloodMax Correctness

Proof (continued).

- r = 1: Trivially, as u_{max} sends id_{max} to all its neighbours
- Assume it holds for any round r - 1 ≥ 1
  - that is, assume that all nodes at distance from u_{max} receive id_{max} in round r - 1
- Then it must hold also for round r
  - Because all those nodes at distance r - 1, in round r set $maxID_i$:= *idmax*
  - Therefore send id_{max} to all their neighbours
  - Implies that all nodes at distance r, receive id_{max} in round r

□

FloodMax Correctness

Theorem. The FloodMax algorithm solves the leader election problem in any strongly connected directed network (provided the availability of unique ids and knowledge of the diameter D).

- Observe that the diameter D concerns the maximum distance in the whole network
- *Can you think of a more precise parameter to replace D in this algorithm?*
  - In the worst case it will be equal to D but
  - In other cases it may be less

FloodMax Correctness

- Observe that the diameter *D* concerns the maximum distance in the whole network
- *Can you think of a more precise parameter to replace D in this algorithm?*
  - It is the maximum distance from u_{max} to any other processor
    - known as the eccentricity of that node
  - Observe that it would be a bit artificial to assume that the algorithm knows this in advance
  - Knowing the diameter of the network as a whole is quite natural to assume

FloodMax Time Complexity

- $D + 1$ rounds

- *Can you see why?*

FloodMax Time Complexity

- $D + 1$ rounds
  - All processors perform a check in round $D + 1$
  - From that point on they do nothing
  - We could have explicitly added a halt (or terminate) command at that point
  - At that point one processor has been elected and all other know that they have not been elected
- Question: *Do they also know who the elected one is?*

FloodMax Time Complexity

- *D* + 1 rounds
  - All processors perform a check in round *D* + 1
  - From that point on they do nothing
  - We could have explicitly added a halt (or terminate) command at that point
  - At that point 1 has been elected and all other know that they have not been elected
- Question: *Do they also know who the elected one is?*
  - *Yes*: In their *$maxID_i$ variable*

FloodMax Communication Complexity

- *$D \cdot m$* messages always
  - *m* (also denoted $|E|$) is the number of directed links in the network

- *Can you see why?*

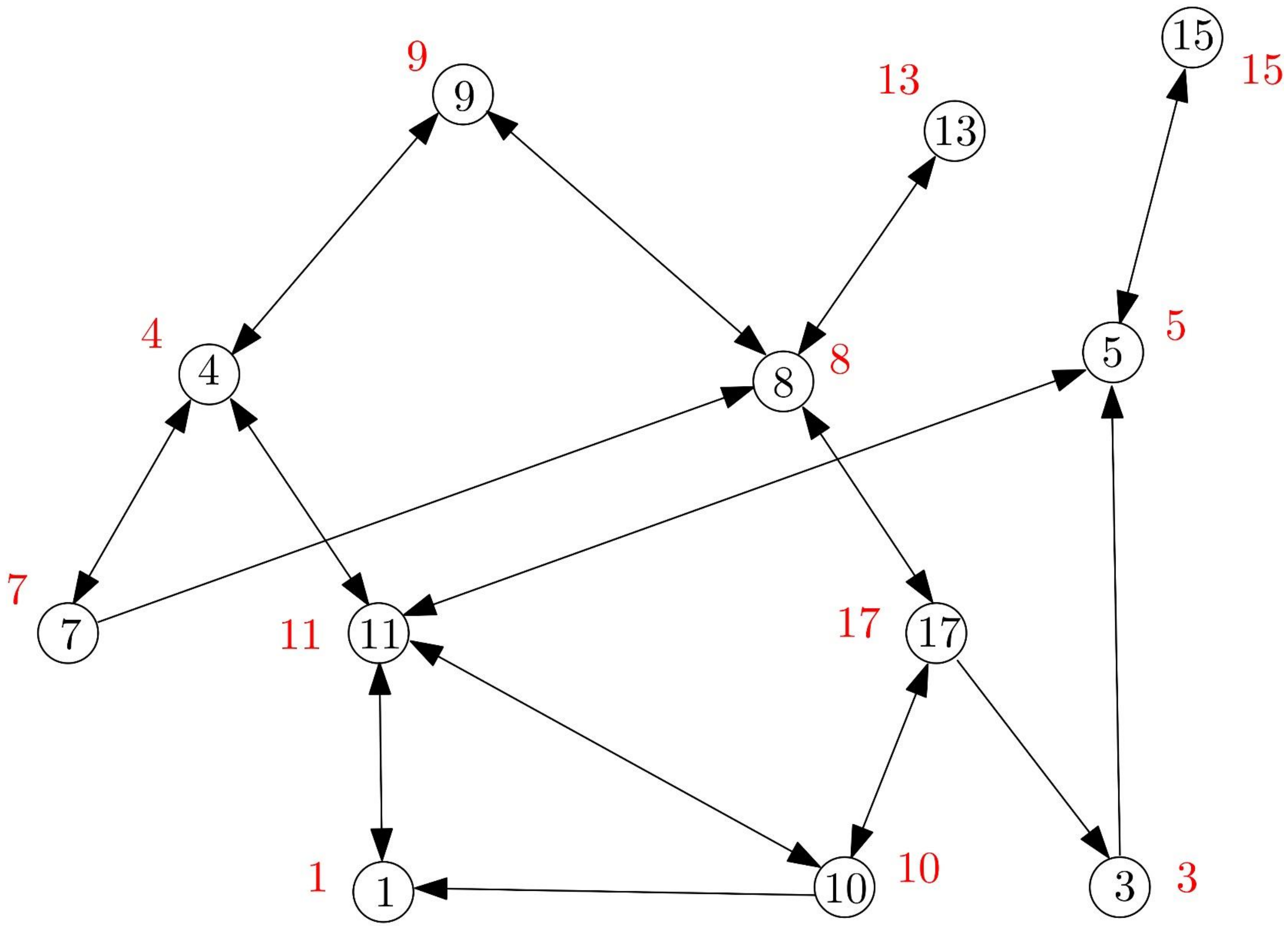
FloodMax Communication Complexity

- $D \cdot m$ messages always
  - m (also denoted $|E|$) is the number of directed links in the network
- In round $D + 1$ nothing is transmitted
  - Only local checks and termination decisions
- For the first D rounds though:
  - Every processor u_i sends $maxID_i$ to all its out-neighbours
  - Therefore, every link has one message in every round transmitted through it

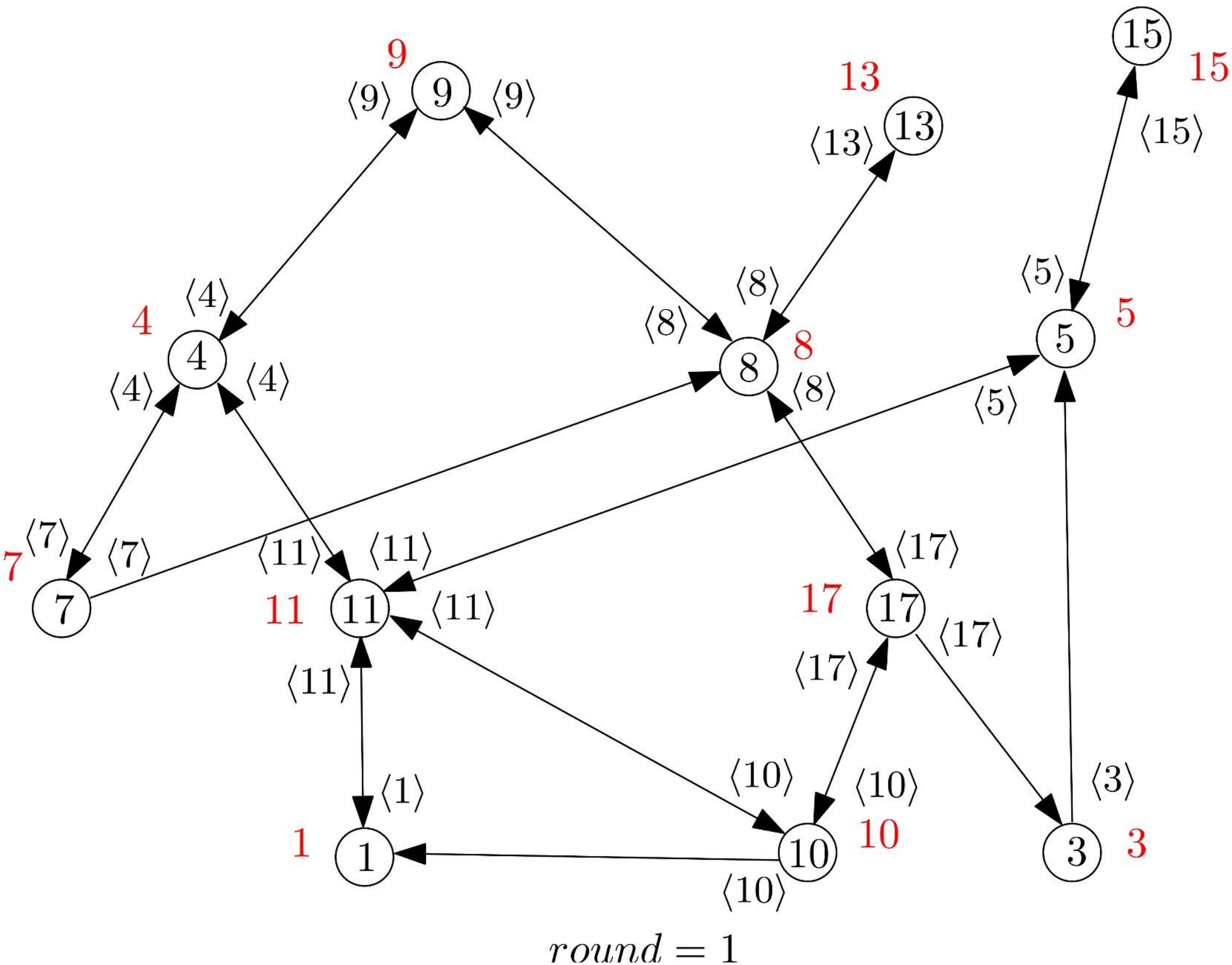
FloodMax Communication Complexity

- *Is this the most message-efficient solution?*
- *Can you observe any "waste" of messages in FloodMax?*

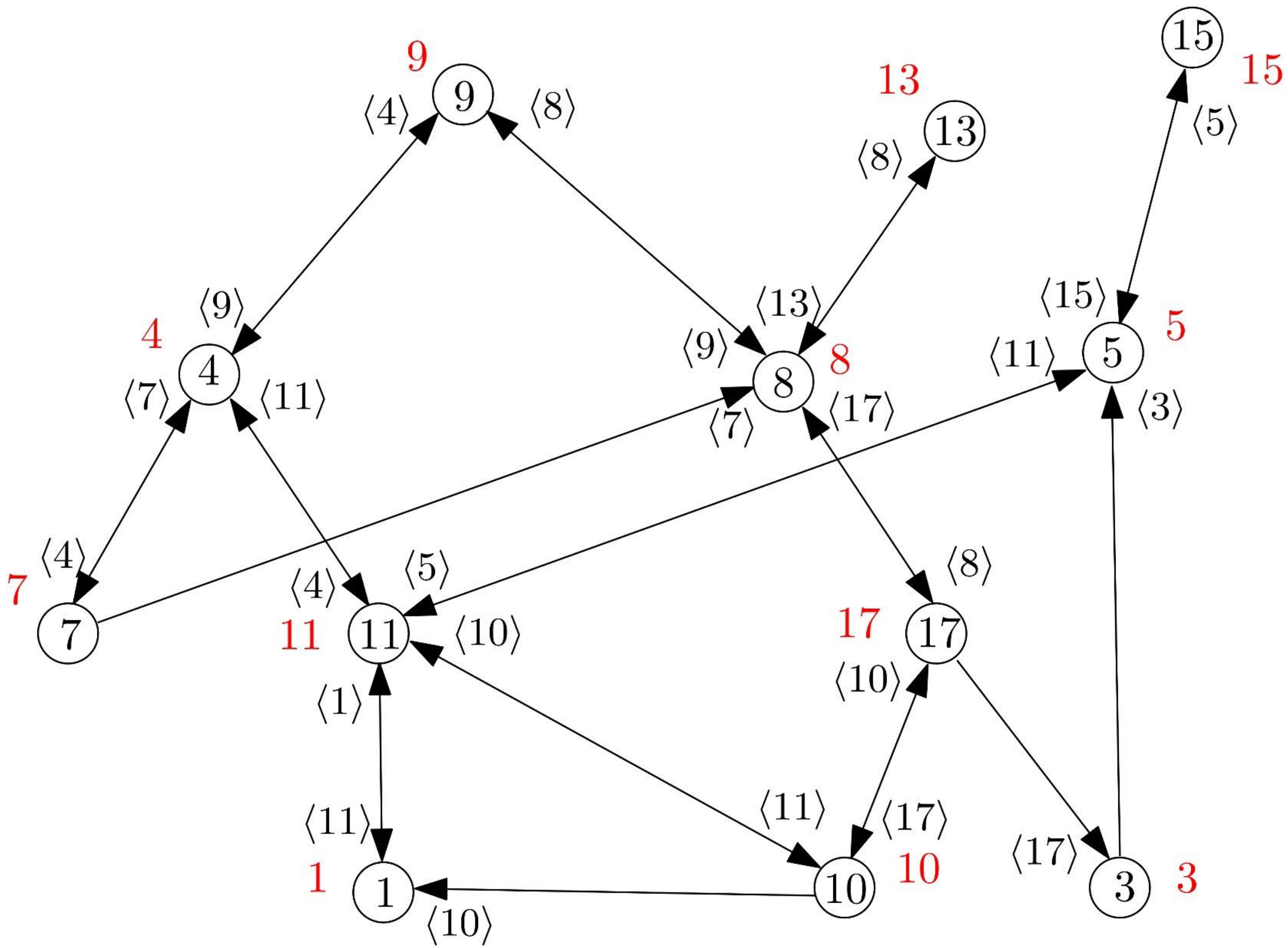
Example Execution



Example Execution

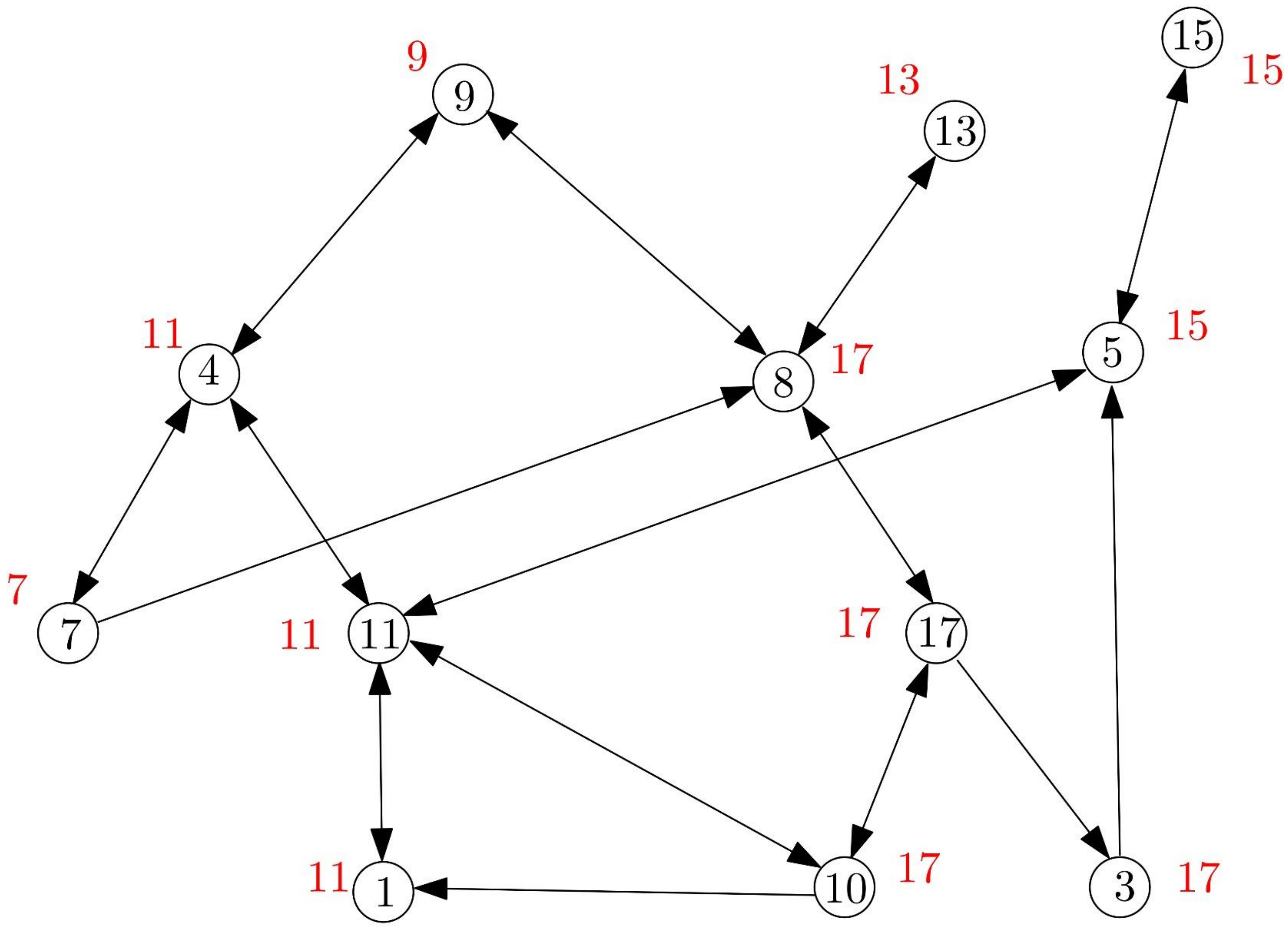


Example Execution



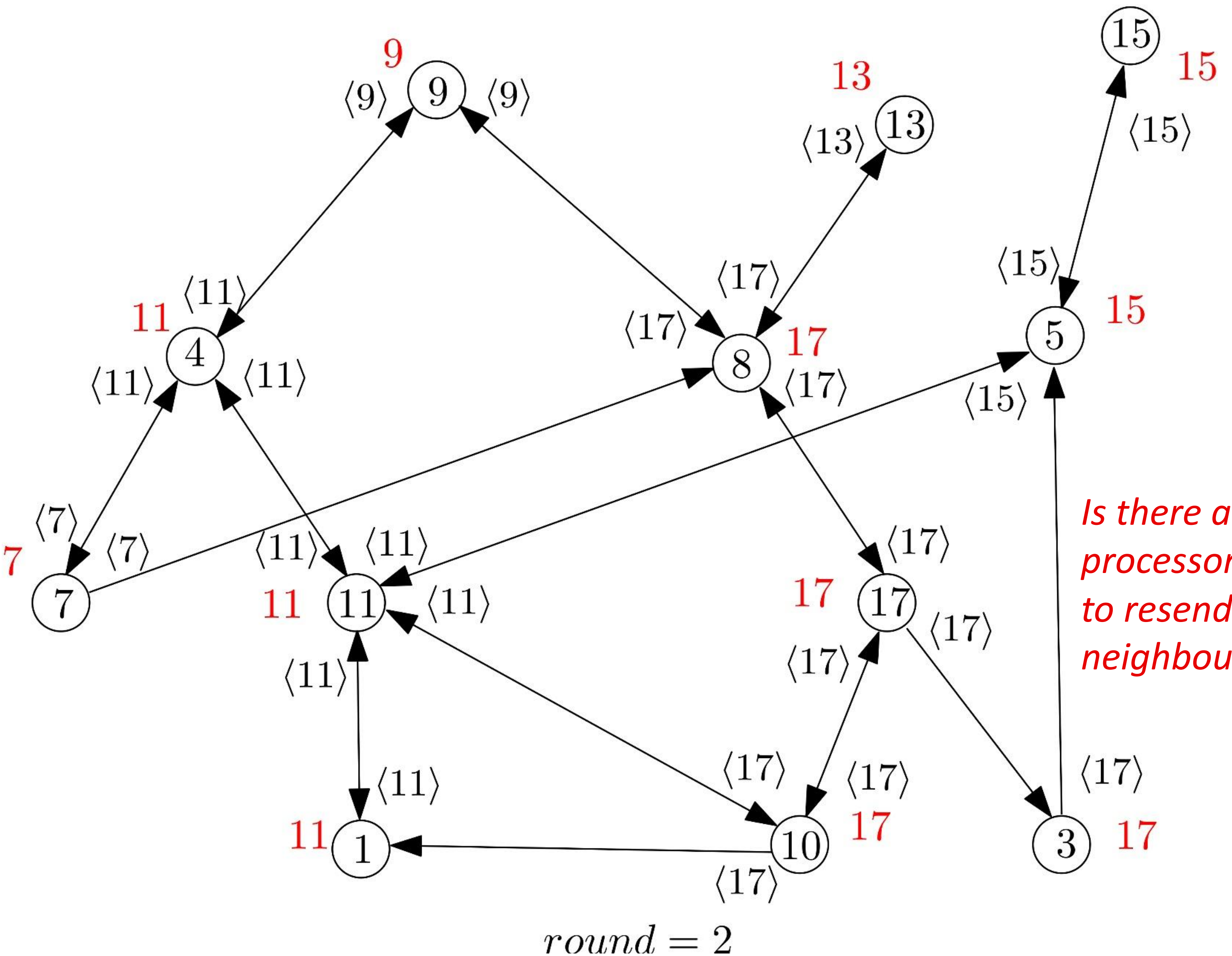
$round = 1$

Example Execution



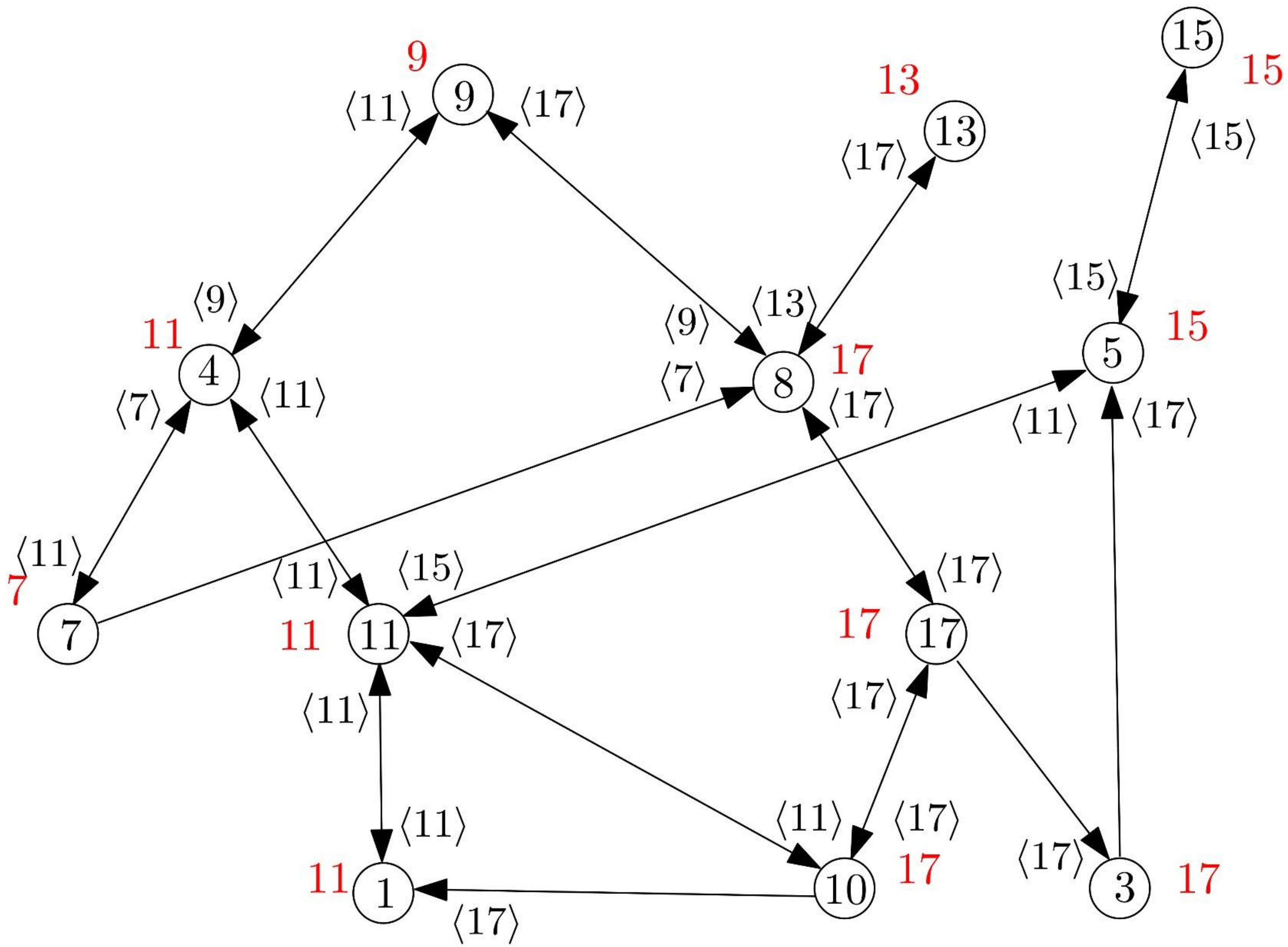
$round = 2$

Example Execution



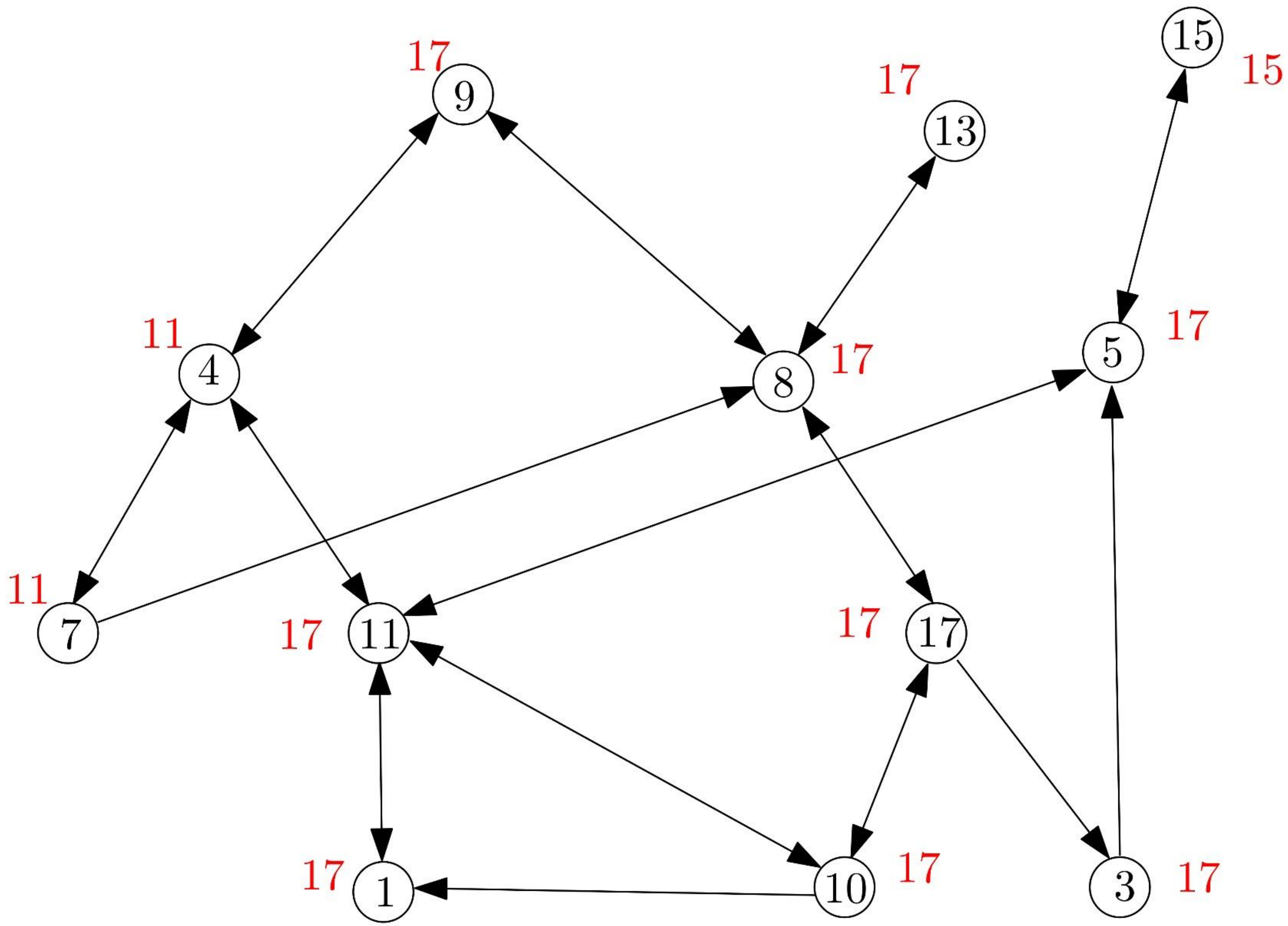
Is there any need for processor with id = 17 to resend ⟨17⟩ to its neighbours?

Example Execution



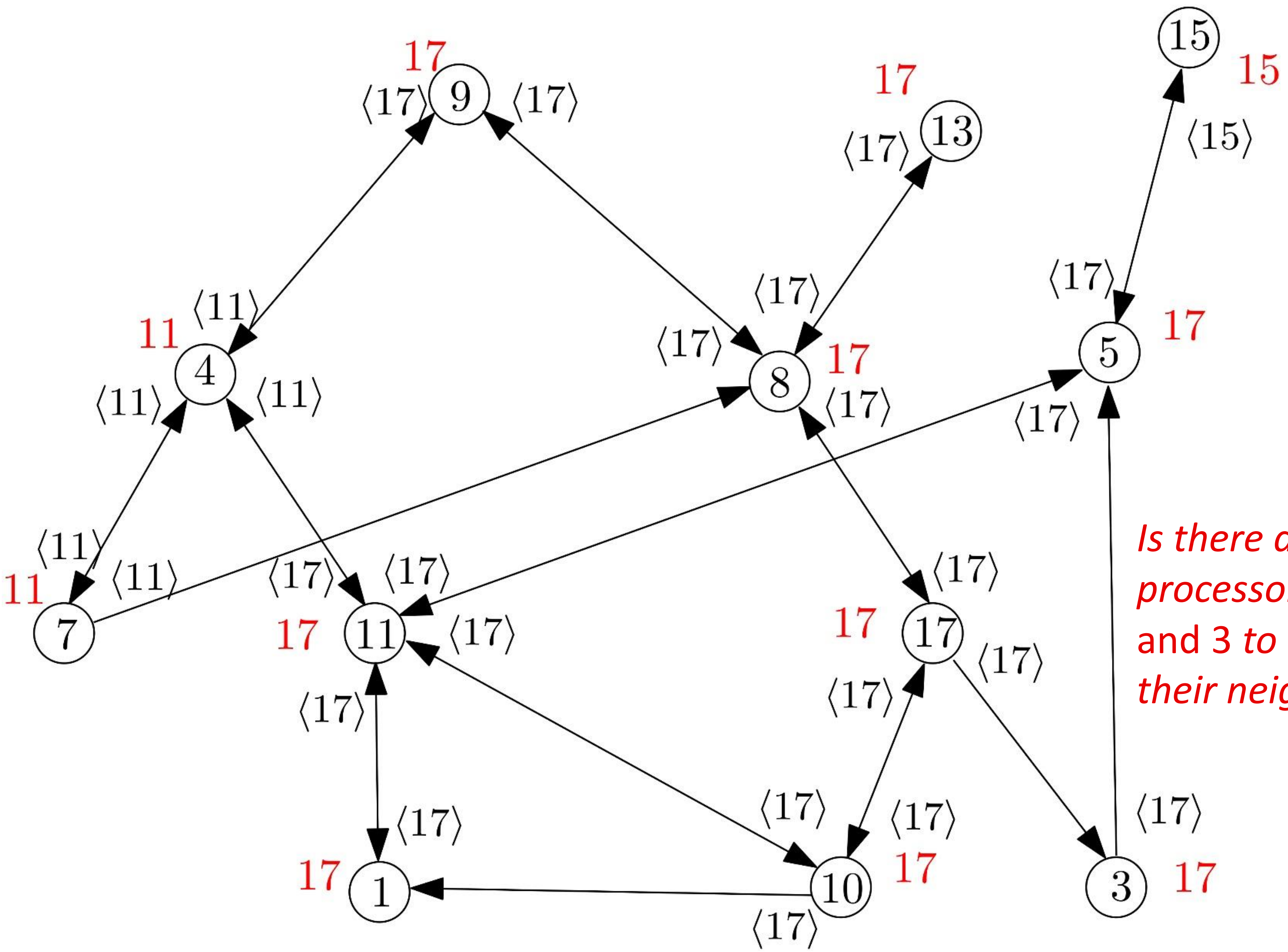
$round = 2$

Example Execution



$round = 3$

Example Execution



$round = 3$

Is there any need for processors 17, 8, 10, and 3 *to resend* ⟨17⟩ *to their neighbours?*

OptFloodMax: An Improvement of FloodMax

- Same as FloodMax but now
  - Processors do not send their $maxID_i$ in every round
  - They only send it whenever they hear a new maximum
  - That is, only in the rounds that they update their $maxID_i$

- Obvious to see that it reduces the #messages in various cases
- Not that obvious yet whether it improves the worst-case complexity

OptFloodMax: Pseudocode

Algorithm OptFloodMax
Code for processor u_i , i ∈ {1, 2,… , n}:

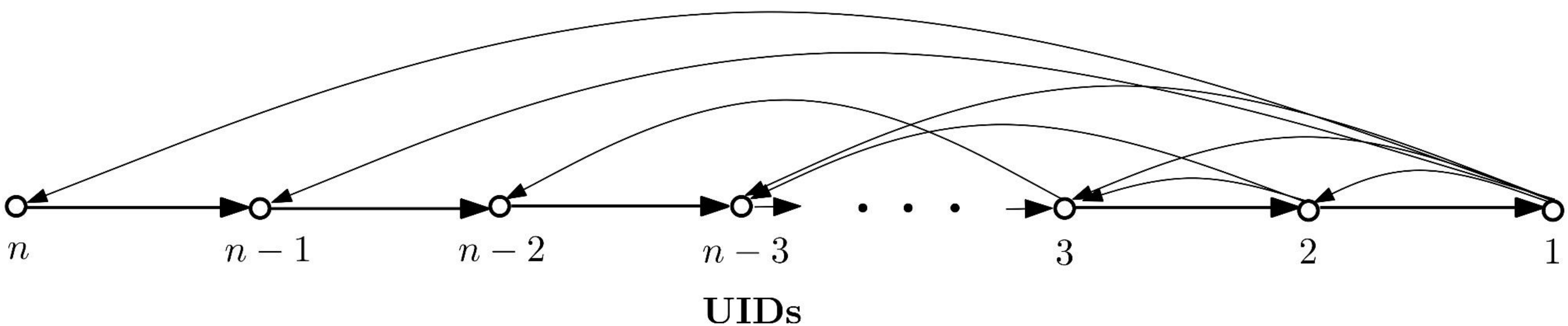
```
Initially:
   u_i knows its own unique id stored in myID_i
   maxID_i := myID_i
   status_i := "unknown"
   newInfo_i := true        // an additional Boolean variable
   Also has access to the current round and knows the diameter D

if round = 1 then
   send ⟨maxID_i⟩ to all out-neighbours
else
   upon receiving ⟨inIDs⟩ from in-neighbours        // one or more ids arriving from neighbours
   if max(inIDs) > maxID_i then
      maxID_i := max({maxID_i} ∪ inIDs)              // remember only the maximum "heard" so far
      newInfo_i := true
   else
      newInfo_i := false
   if round ≤ D and newInfo_i = true then    // 1 < round ≤ D
      send ⟨maxID_i⟩ to all out-neighbours
   else if round = D + 1 then
      if maxID_i = myID_i then        // if equal to your own, no greater id exists in the network
         status_i := "leader"         // therefore, elect yourself a leader
      else                            // greater than own
         status_i := "non-leader"     // therefore, declare yourself a non-leader
// observe that in the end all processors know the id of the elected leader, stored in their maxID_i
// variable
```

OptFloodMax: Correctness and Complexity

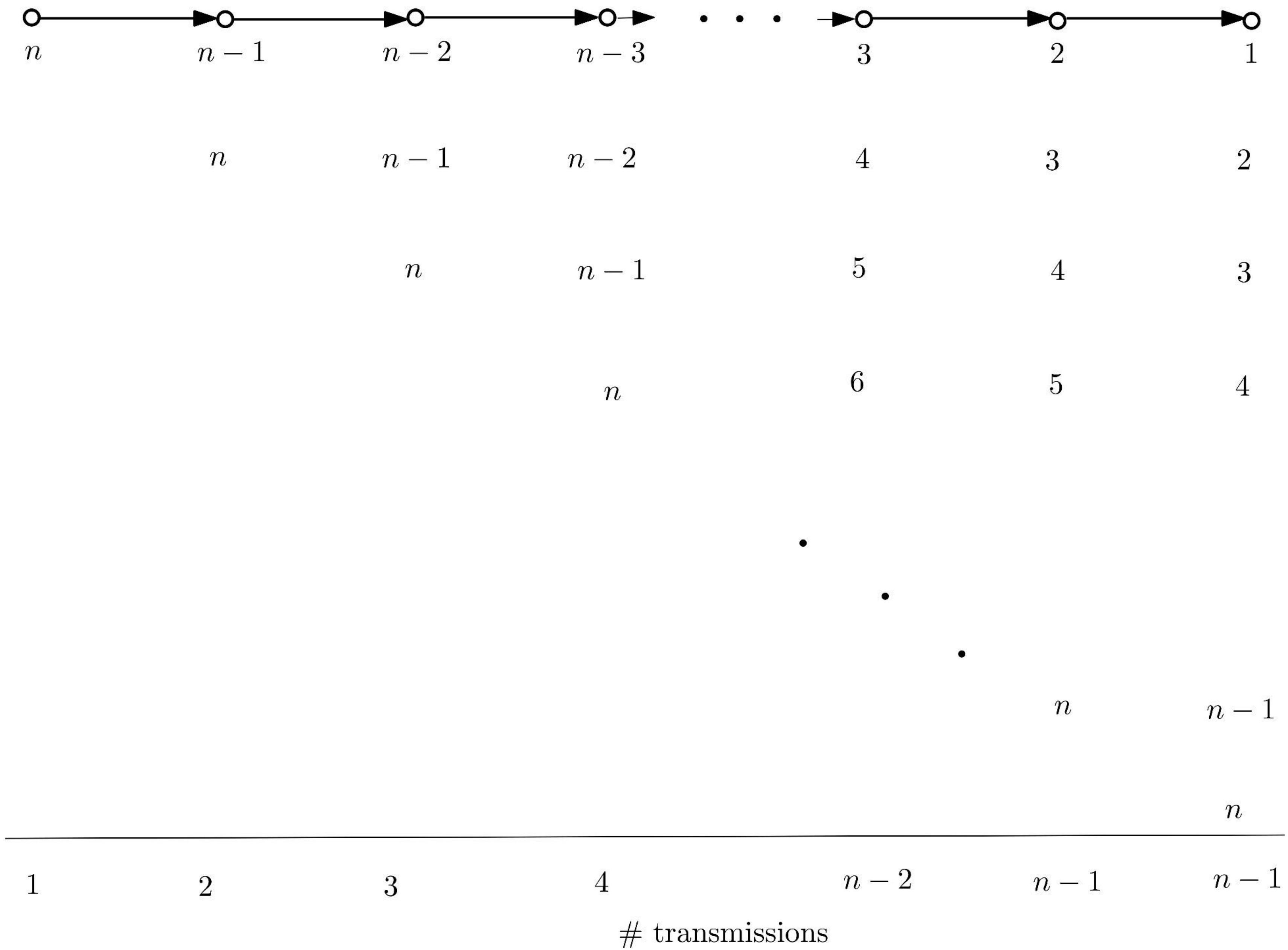
- Correctness:
  - remains correct (needs proof)
- Time complexity:
  - same as in FloodMax (immediate)
- Communication complexity:
  - *We can show that it does not improve the worst-case complexity compared to FloodMax*
  - FloodMax: $D \cdot m = O(n^3)$ messages
  - We can show that in some cases also OptFloodMax transmits $\Theta(n^3)$ messages

OptFloodMax: Communication Complexity



- It happens that UIDs here are 1 through n (consecutive) - not necessary
  - But their order in the network (combined with the specific structure of this network) is important for this result
- Remark: The network has **all** inverse links (to the left), not only the ones shown here

OptFloodMax: Communication Complexity



OptFloodMax: Communication Complexity

$$
\begin{aligned}
\#messages &= (n-1)^2 + \sum_{i=2}(n-i+1)^2 \\
&= \left[\sum_{i=1}^{n}(n-i+1)^2\right] - 2n + 1 \\
&= \left[\sum_{i=1}^{n} i^2\right] - 2n + 1 \\
&= \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6} - 2n + 1 \\
&= \Theta(n^3)
\end{aligned}
$$

(Opt)FloodMax Further Improvement

- *Can you think of any additional improvement?*
- *Any waste of messages that still remains?*

(Opt)FloodMax Further Improvement

- *Can you think of any additional improvement?*
- *Any waste of messages that still remains?*

- Yes: No need to send to a processor that just send you the new maximum
  - Again won't improve the worst case complexity
  - Still, we gain something in many cases

Summary

- Leader election is crucial for distributed systems
  - breaks symmetry
  - allows for coordination
- If all processors are initially identical then
  - impossible to elect a leader even in very simple networks
  - e.g., a ring
- Adding unique ids breaks this inconvenient initial symmetry
- The LCR algorithm elects a leader in any ring network
  - simple conceptually, assumes unique ids
  - n rounds (or $2n$ for all to terminate), $O(n^2)$ messages
- The FloodMax algorithm elects a leader in any strongly connected network
  - like a generalisation of LCR
  - simple, assumes unique ids and knowledge of the diameter D
  - D rounds, $D \cdot m$ messages
- The OptFloodMax algorithm is an improvement of FloodMax
  - Decreases the number of messages in many cases
  - Does not improve the worst-case complexity
  - Still such improvements may be very important for real systems and applications where we are not always faced with the worst cases