Distributed Systems
COMP 212

Lecture 10

Othon Michail

UNIVERSITY OF LIVERPOOL

Breadth-First Search
and
Construction of a BFS Tree

So Far

- Broadcast from a root u_0 given a spanning tree
- Broadcast without a spanning tree but again from a root u_0
- Leader election in special networks
- Leader election in general networks, knowing the diameter D

- *Question: Can we still solve these without any assumptions but unique ids?*

Broadcast without a given Spanning Tree

Problem:

- u_0 has some information it wishes to send to all processors
  - e.g., a message $\langle M \rangle$
  - additionally all nodes must have terminated at the end
- No spanning tree of the network G is given in advance
- The algorithm should also output a constructed spanning tree of G

Solution: Informal description

- All nodes awake initially
- If awake and have just received $\langle M \rangle$ from some neighbours,
  - Choose one of those neighbours as your *parent* and let that processor know
  - forward $\langle M \rangle$ to the rest of the neighbours
  - Wait for 1 round to collect children (if any) and then sleep
- If neighbours inform you that you are their parent,
  - add those processors to your *children* list
  - sleep
- If you are asleep, do nothing

Solution: Pseudocode

Algorithm Broadcast & Spanning tree construction
Code for processor u_i, $i \in \{0, 1, \ldots, n - 1\}$:

Initially *parent* = $\perp$ and *children* = $\emptyset$

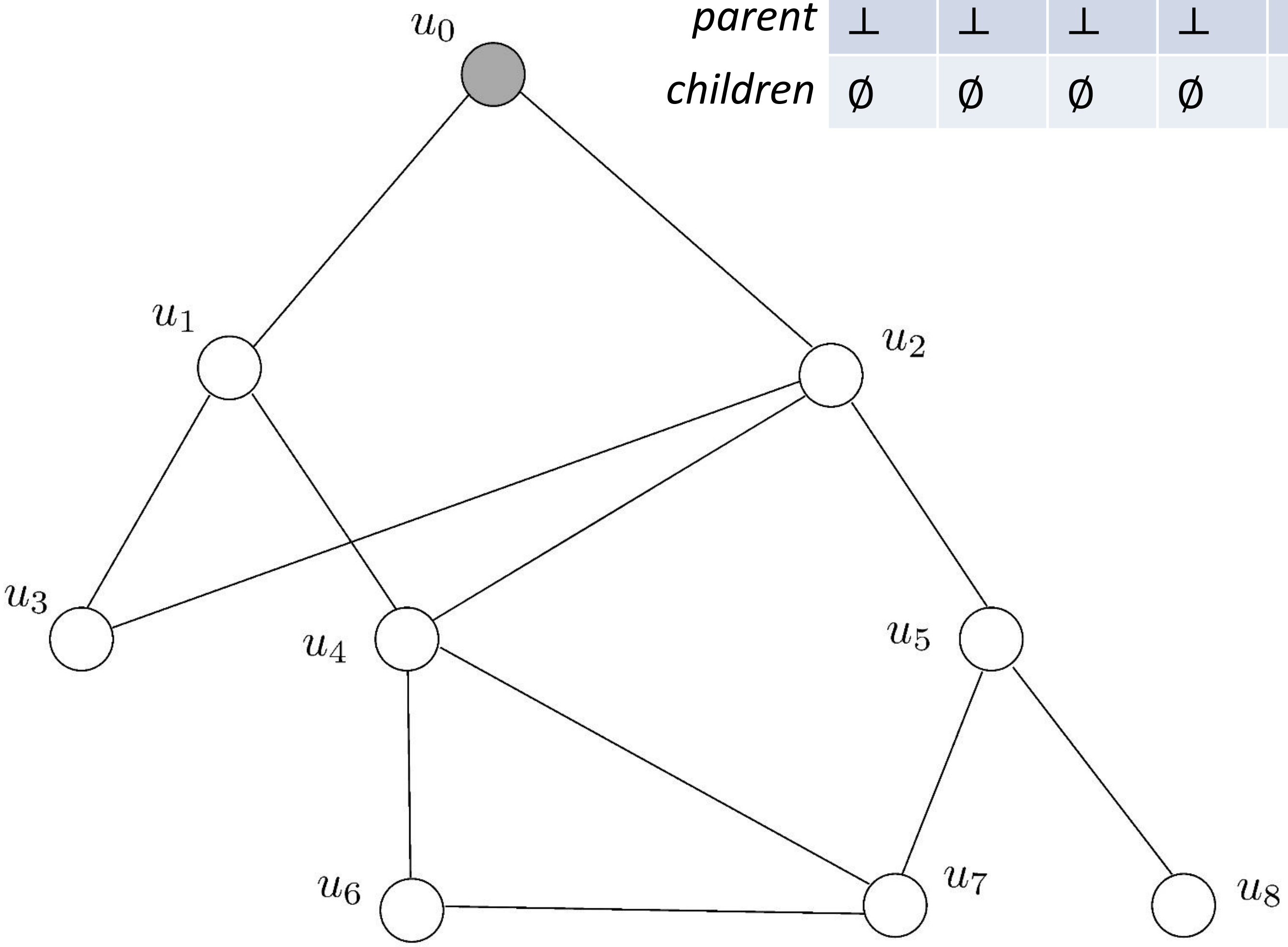
```
if u_i = u_0 and parent = ⊥ then                 // root has not yet sent ⟨M⟩
   send ⟨M⟩ to all neighbours
   parent := u_i

upon receiving ⟨M⟩ from neighbours N:
   if parent = ⊥ then                            // u_i has not received ⟨M⟩ before
      parent := u_j ∈ N                          // select one arbitrarily as parent
      send ⟨"parent"⟩ to u_j
      send ⟨M⟩ to all neighbours except those in N
      wait for one round to collect children if any and then terminate
    else terminate

upon receiving ⟨"parent"⟩ from neighbours N:
   add all u_j ∈ N to children
   terminate
```

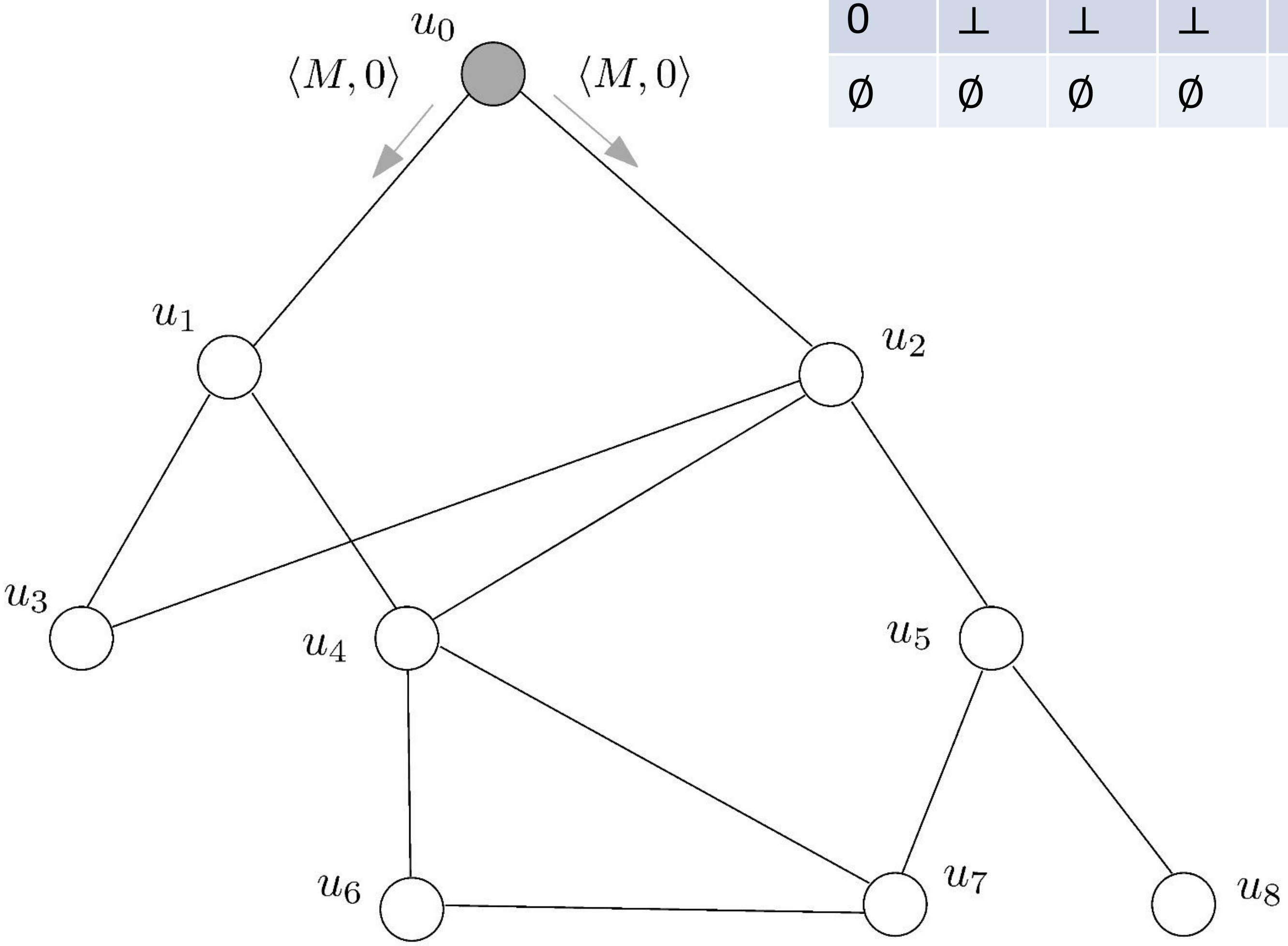
Example Execution

	u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8
parent	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
children	∅	∅	∅	∅	∅	∅	∅	∅	∅



Example Execution

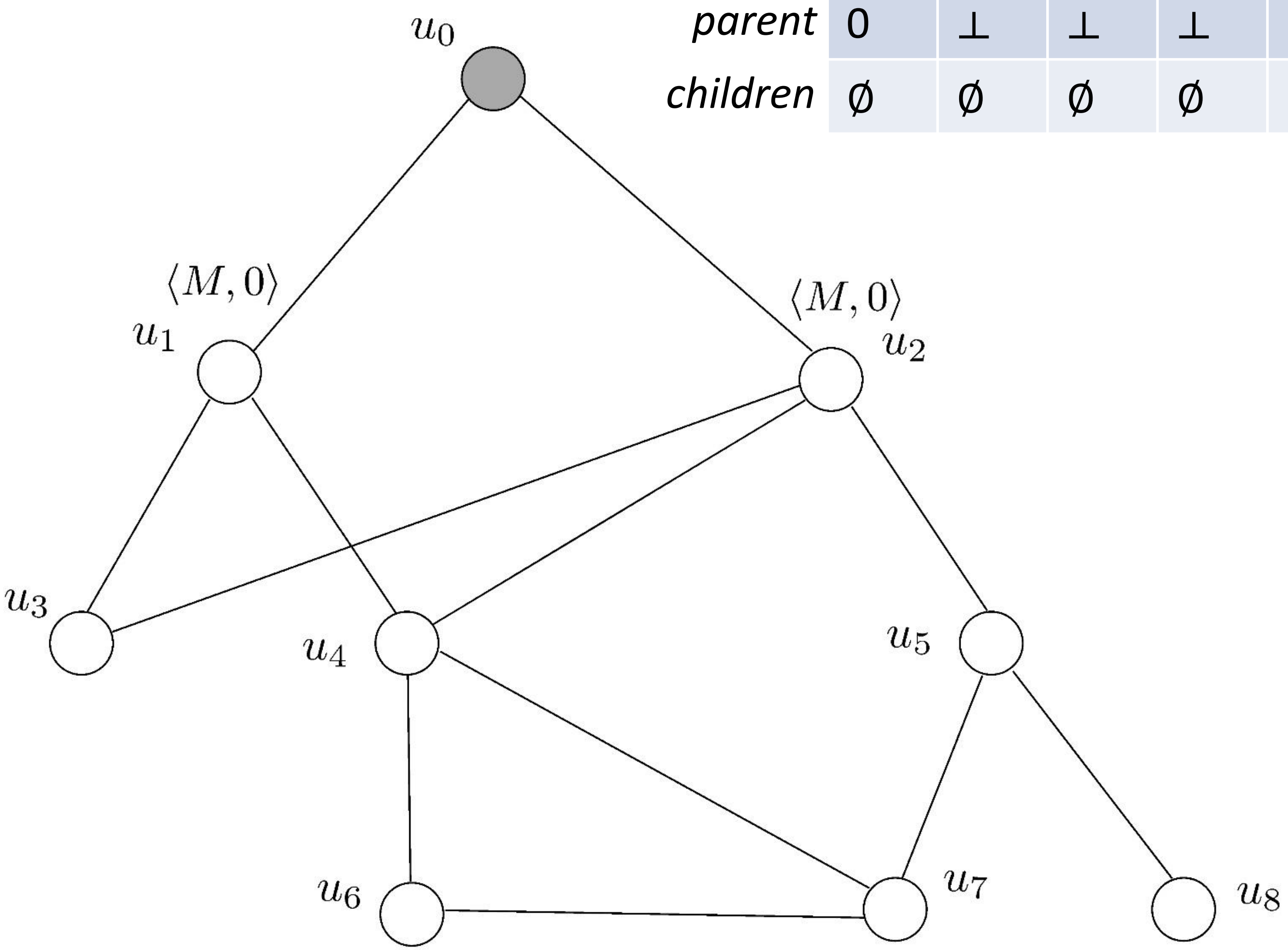
u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8
0	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
∅	∅	∅	∅	∅	∅	∅	∅	∅



$round = 1$

Example Execution

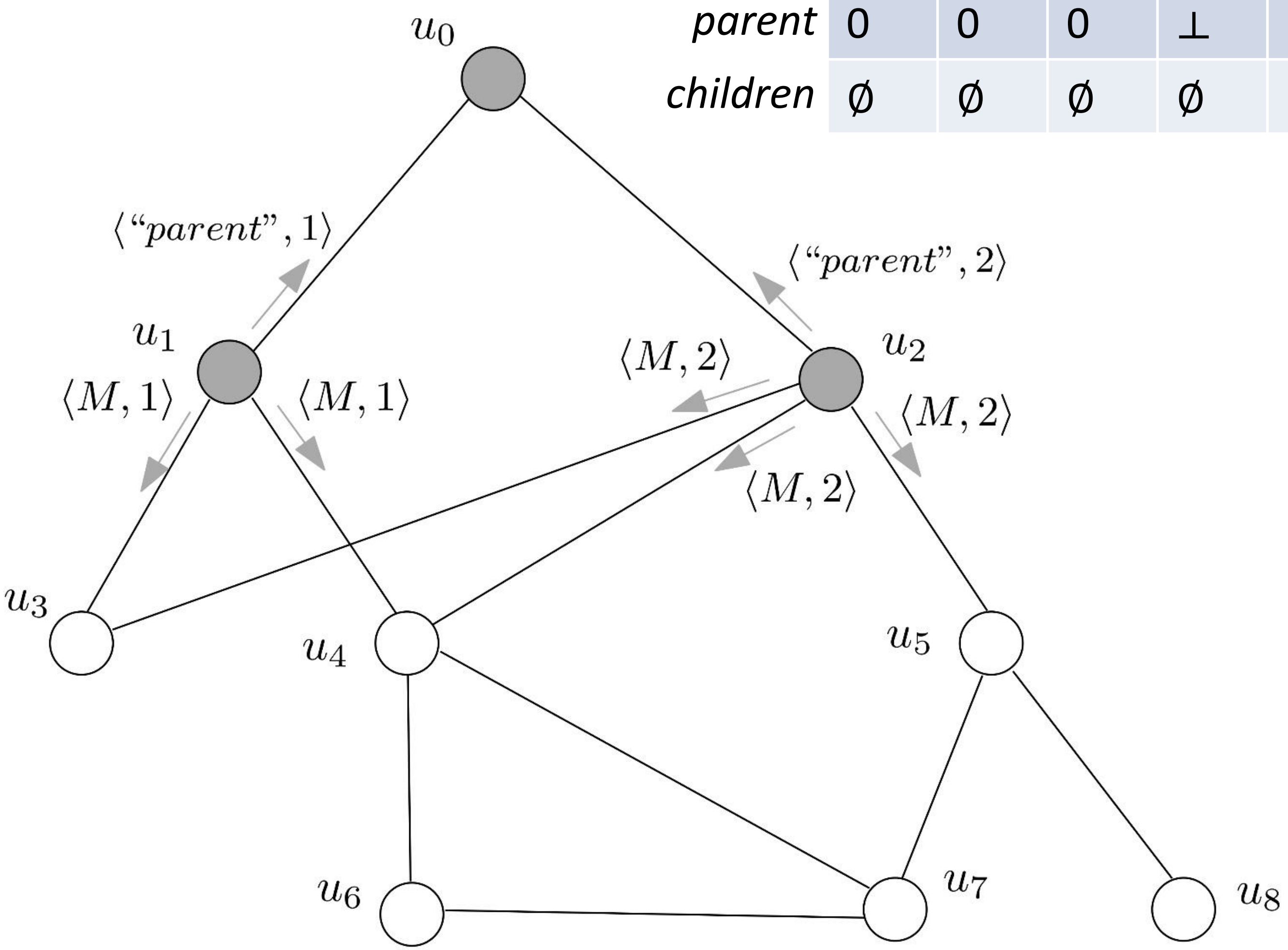
	u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8
parent	0	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
children	∅	∅	∅	∅	∅	∅	∅	∅	∅



$round = 1$

Example Execution

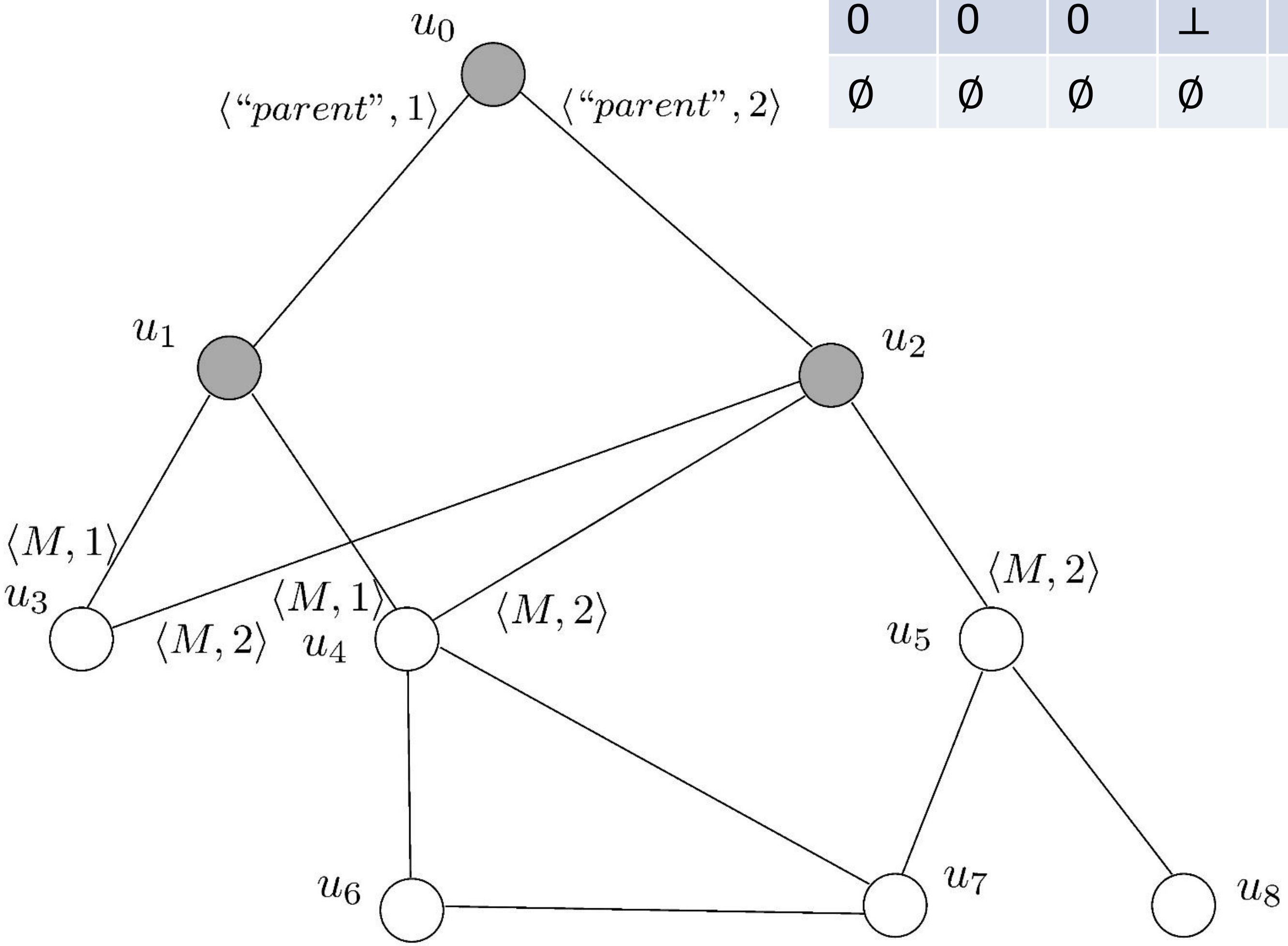
	u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8
parent	0	0	0	⊥	⊥	⊥	⊥	⊥	⊥
children	∅	∅	∅	∅	∅	∅	∅	∅	∅



$round = 2$

Example Execution

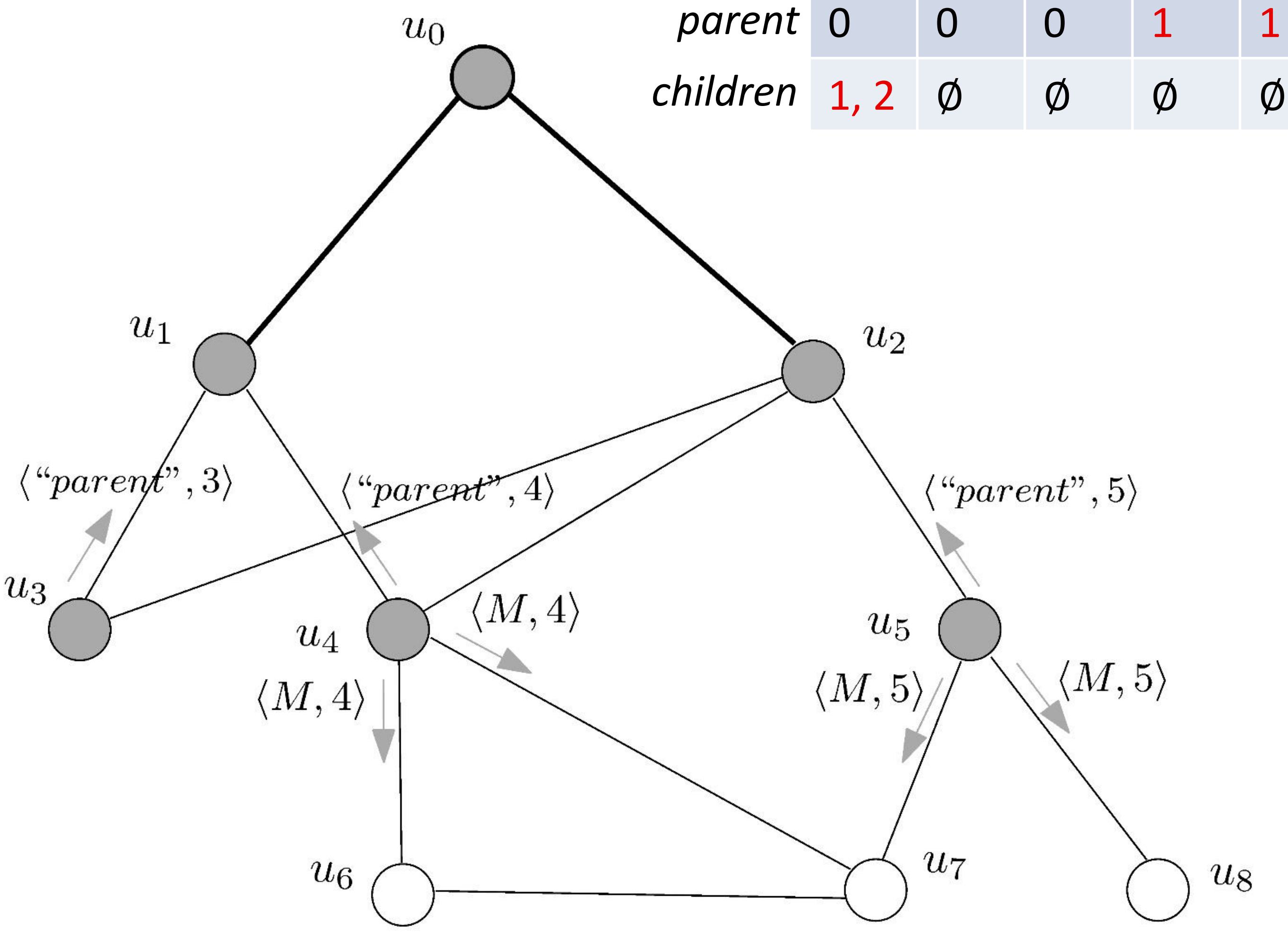
u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8
0	0	0	⊥	⊥	⊥	⊥	⊥	⊥
∅	∅	∅	∅	∅	∅	∅	∅	∅



$$round = 2$$

Example Execution

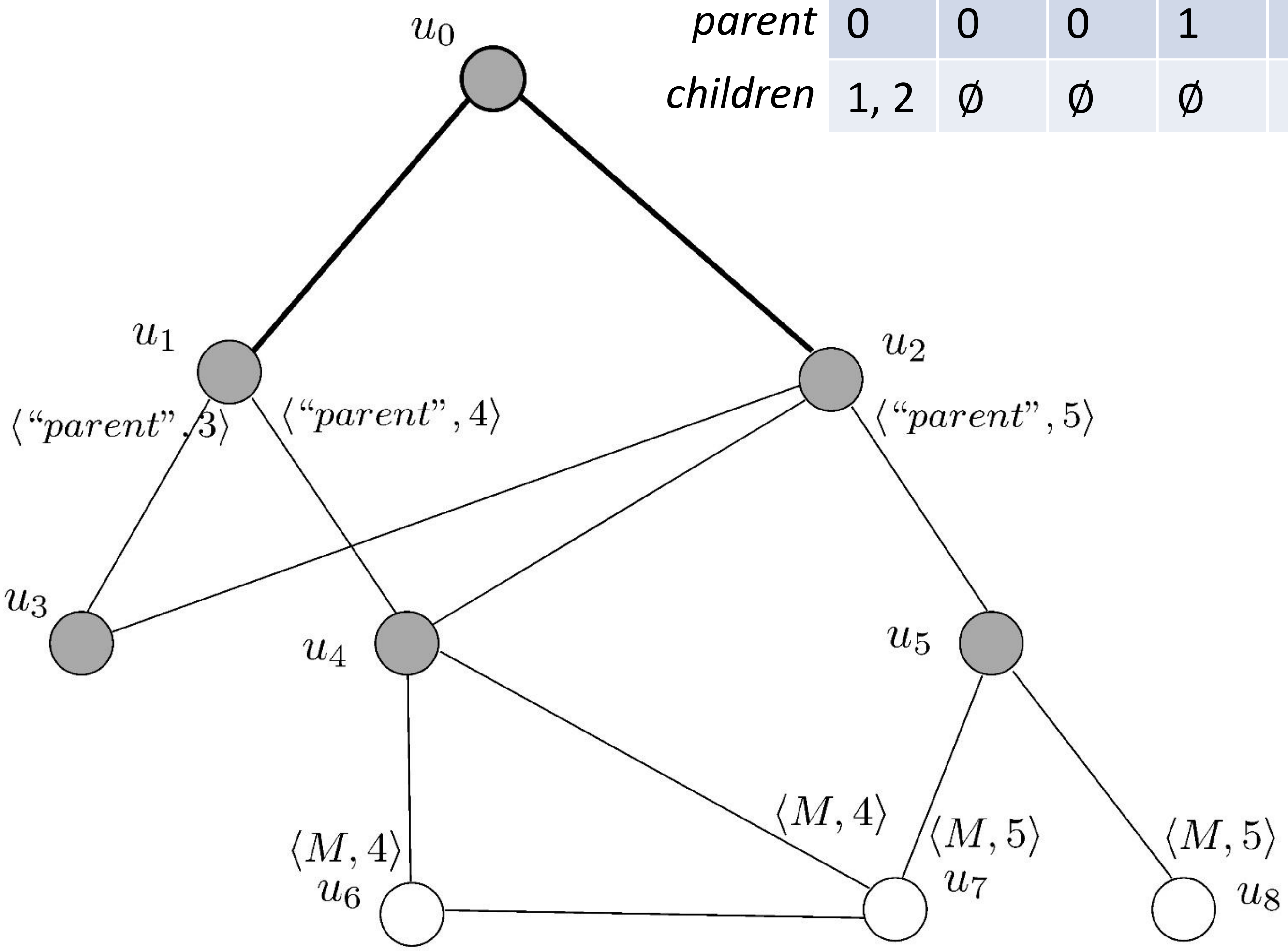
	u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8
parent	0	0	0	1	1	2	⊥	⊥	⊥
children	1, 2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø



$round = 3$

Example Execution

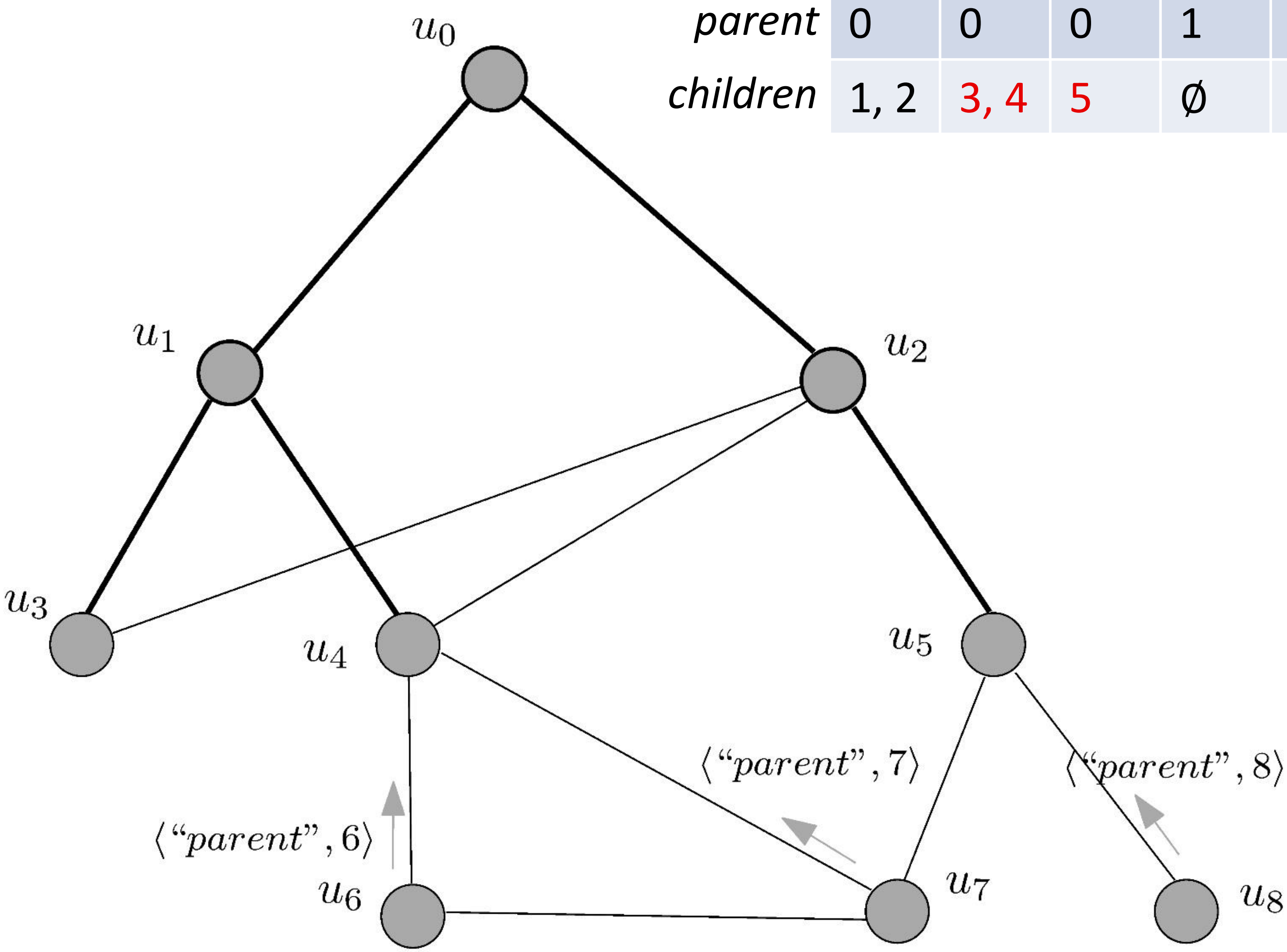
	u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8
parent	0	0	0	1	1	2	⊥	⊥	⊥
children	1, 2	∅	∅	∅	∅	∅	∅	∅	∅



$round = 3$

Example Execution

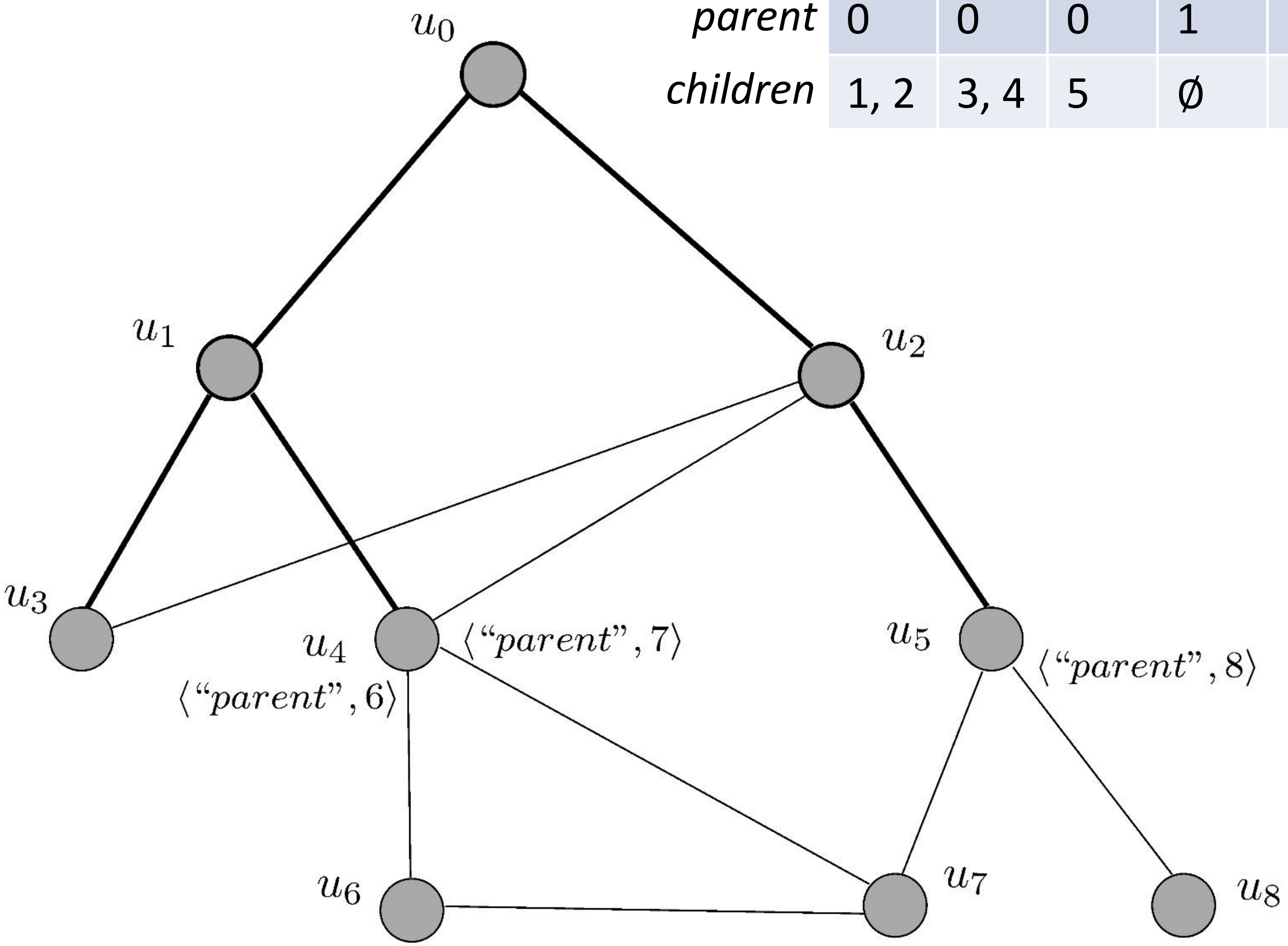
	u_0	u_1	u_2	u_3	u_4	u_5	U_6	u_7	u_8
parent	0	0	0	1	1	2	4	4	5
children	1, 2	3, 4	5	Ø	Ø	Ø	Ø	Ø	Ø



$round = 4$

Example Execution

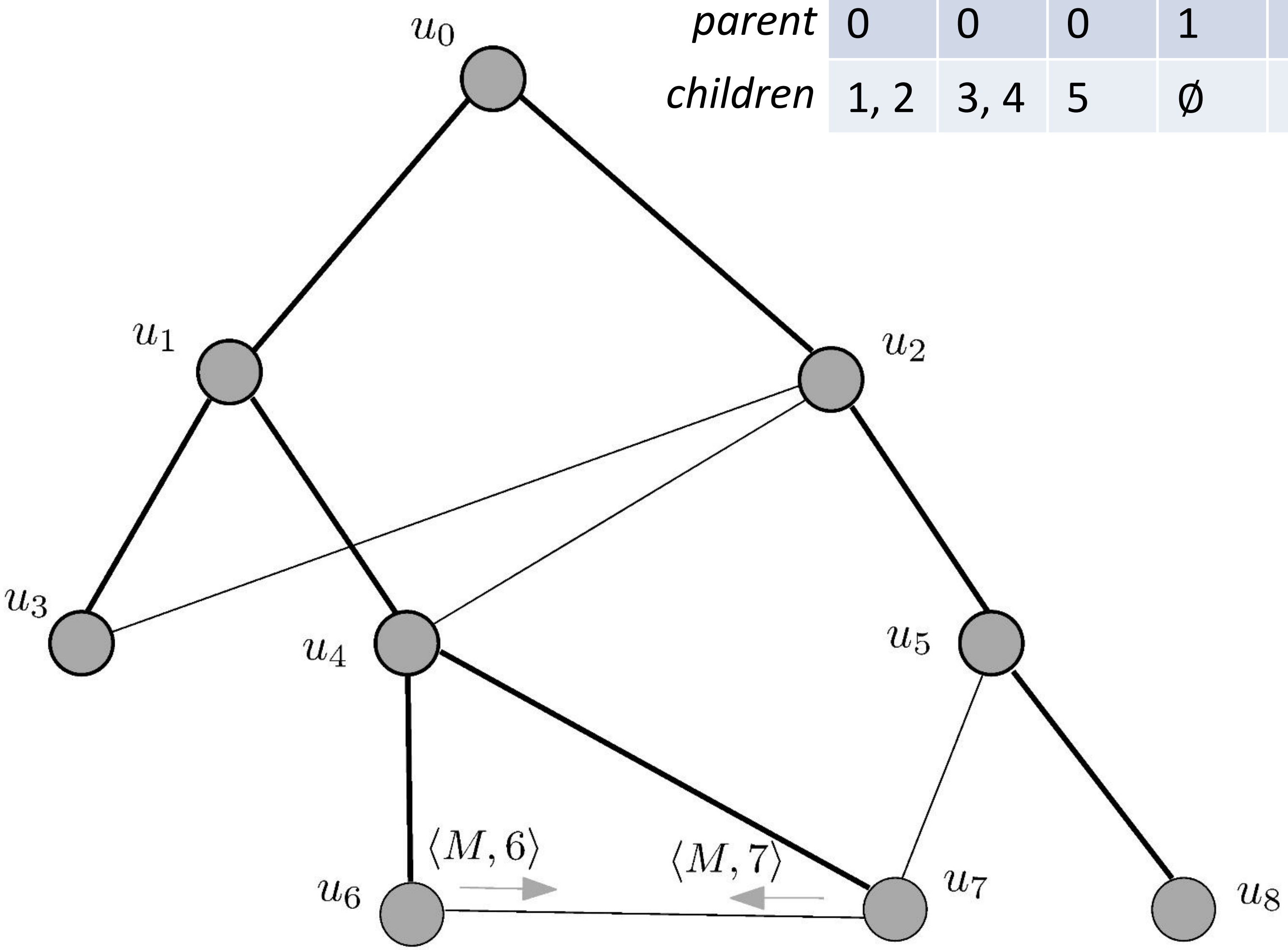
	u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8
parent	0	0	0	1	1	2	4	4	5
children	1, 2	3, 4	5	∅	∅	∅	∅	∅	∅



$round = 4$

Example Execution

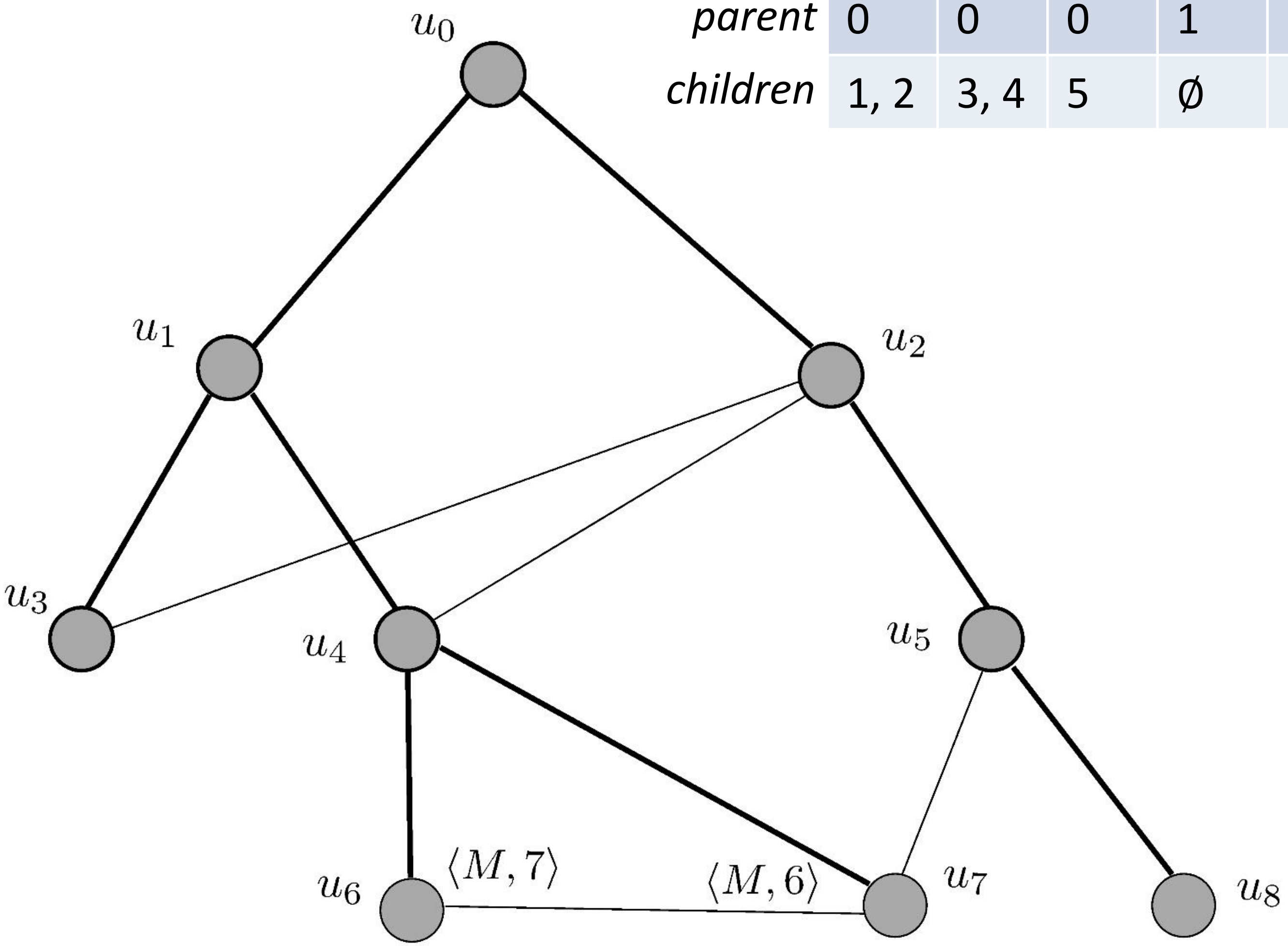
	u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8
parent	0	0	0	1	1	2	4	4	5
children	1, 2	3, 4	5	∅	6, 7	8	∅	∅	∅



$round = 5$

Example Execution

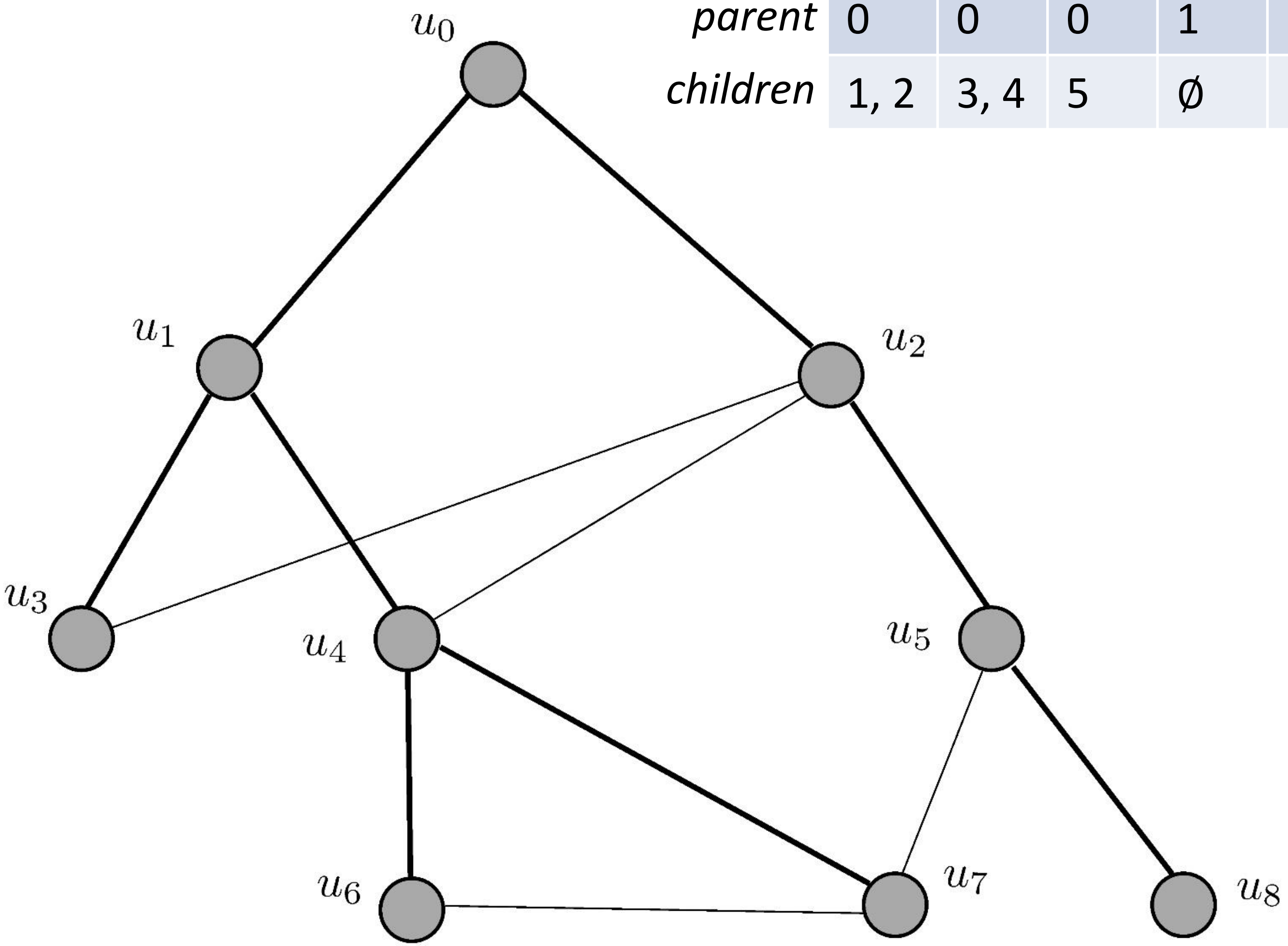
	u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8
parent	0	0	0	1	1	2	4	4	5
children	1, 2	3, 4	5	∅	6, 7	8	∅	∅	∅



$round = 5$

Example Execution

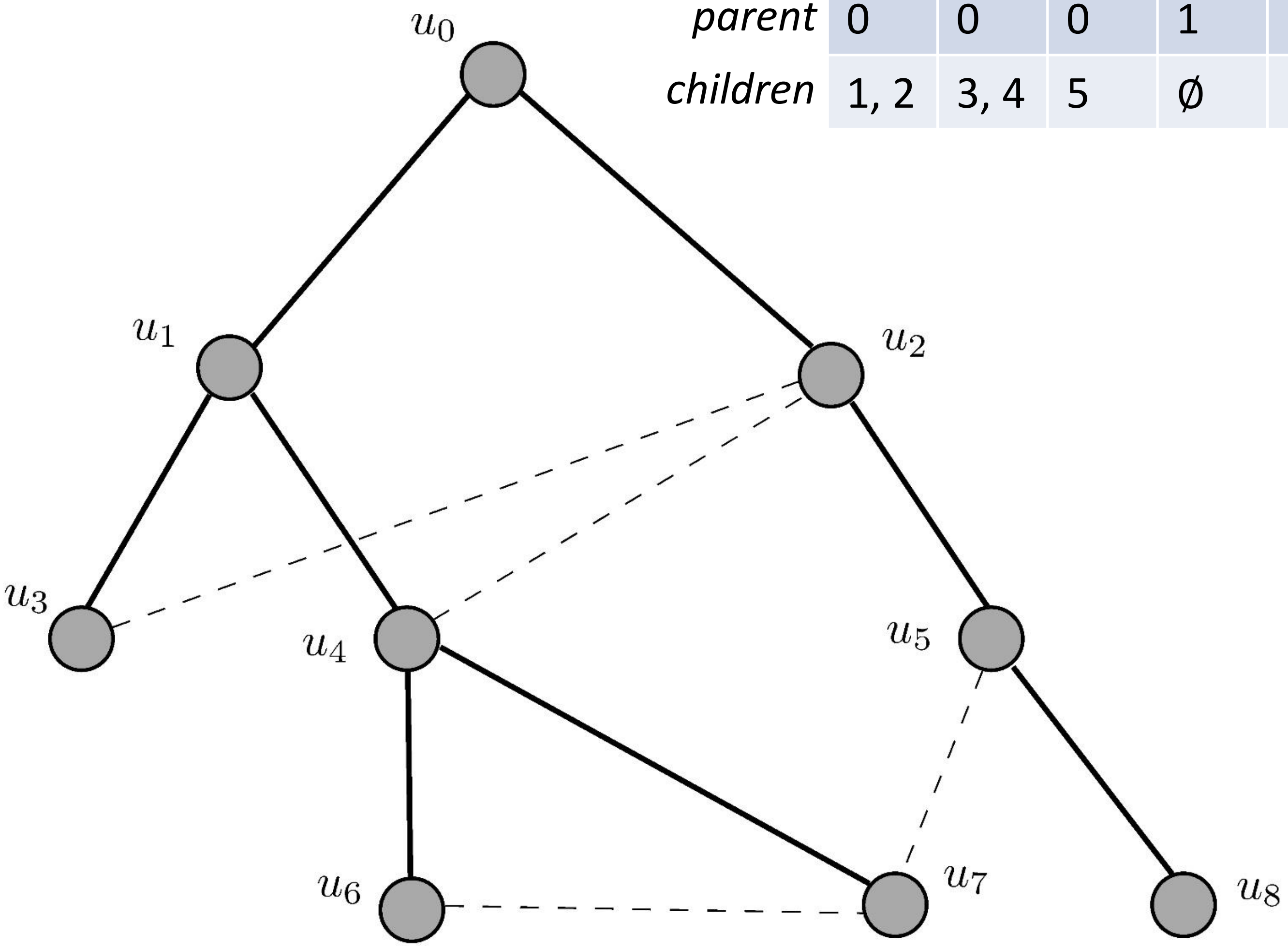
	u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8
parent	0	0	0	1	1	2	4	4	5
children	1, 2	3, 4	5	∅	6, 7	8	∅	∅	∅



$round = 6$

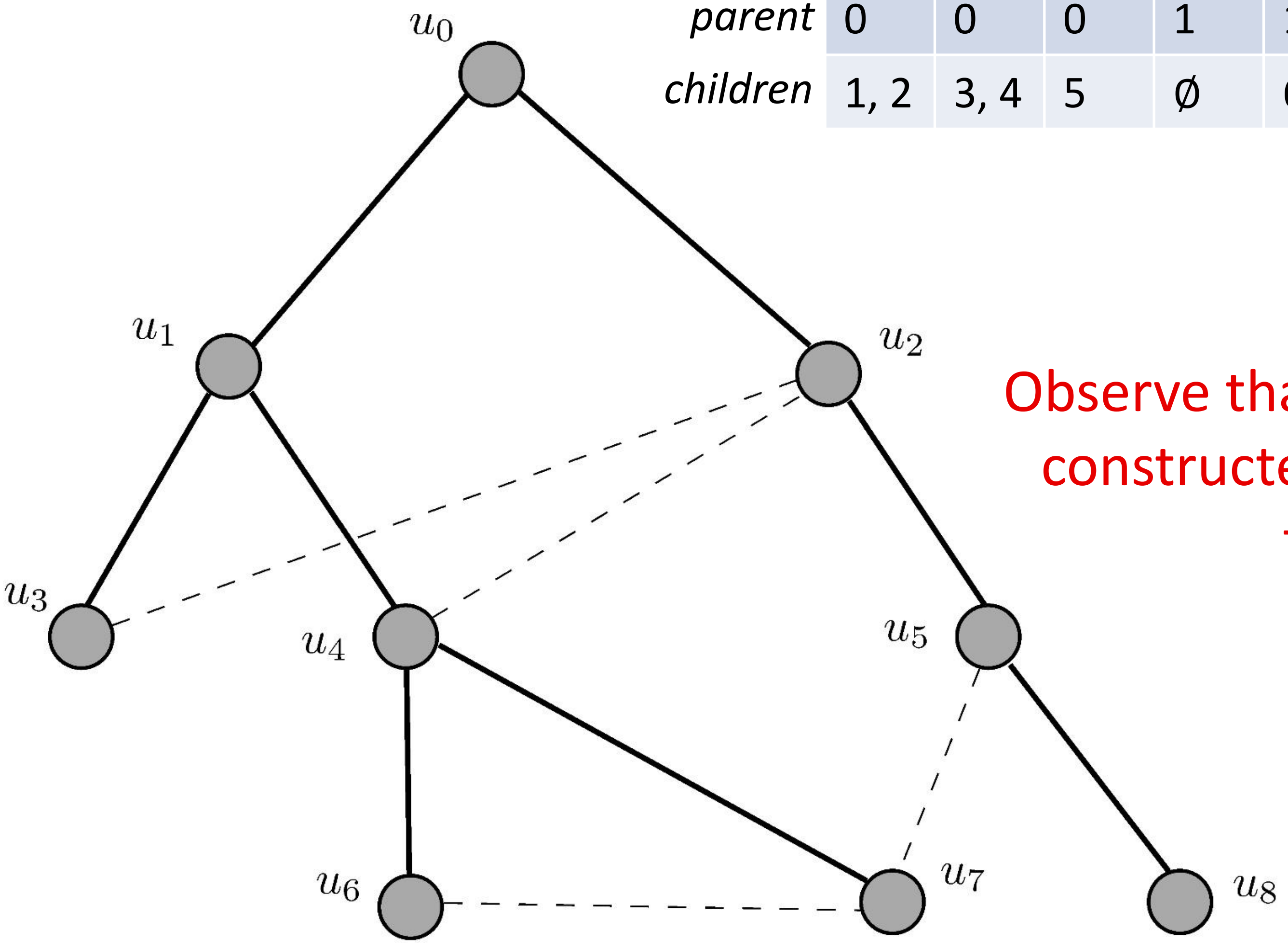
Example Execution

	u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8
parent	0	0	0	1	1	2	4	4	5
children	1, 2	3, 4	5	∅	6, 7	8	∅	∅	∅



Example Execution

	u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8
parent	0	0	0	1	1	2	4	4	5
children	1, 2	3, 4	5	∅	6, 7	8	∅	∅	∅



Observe that the spanning tree constructed is actually a BFS tree of u_0

Beyond a Predetermined Root

- But the previous algorithm essentially requires a predetermined root u_0
  - u_0 starts and ends the BFS/Broadcast
  - All other processors simply "follow" whenever they receive information that has originated at u_0
- But in a "truly" distributed system we may not have such a u_0
- Question: *Can we drop this assumption?*
- That is, *can we construct a single BFS (spanning) tree, with a designated root despite the fact that initially no such root exists?*

Beyond a Predetermined Root

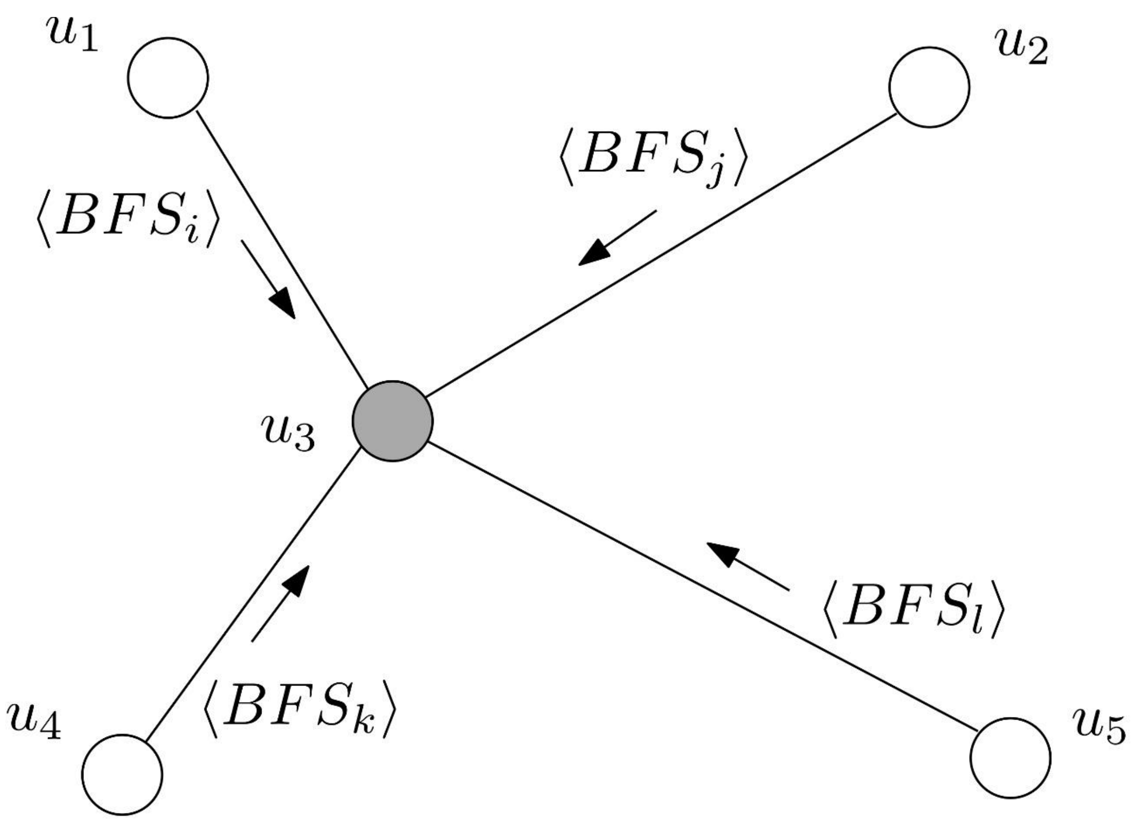
- *Can we construct a single BFS (spanning) tree, with a designated root despite the fact that initially no such root exists?*
- Yes we can!

All in Parallel

- Main idea: Each processor pretends to be the root and initiates its own BFS/broadcast
- So, initially *n* BFSs start in parallel
- To avoid "interference", each root broadcasts its unique id
- But how does an "internal" node of one or more BFSs (which btw is also the root of its own BFS) chooses what to store and forward?

All in Parallel

- But how does an "internal" node chooses what to store and forward?



Two Main Approaches

1. Store and forward "everything"
2. Store and forward only the BFS of the root with the maximum id (among those heard so far)

1. Everything

Main Idea:

- All nodes transmit initially their id to all their neighbours
- Every node u that receives a ⟨*rootID*⟩ message from neighbours v_i,
  - Sets one of those, call it v, as its parent in the T_{RootID} tree
  - Responds ⟨“*parent*”, *rootID*⟩ to v
  - Forwards ⟨*rootID*⟩ to all its neighbours apart from v
- Upon receipt of ⟨“*parent*”, *rootID*⟩ from neighbours
  - Add those neighbours to your $children_{RootID}$ list

1. Everything

- Therefore, every processor
  - Is the root of its own tree
  - Will be a non-root node in n - 1 other processors' trees
  - Has to also play the role of an internal or leaf node in n - 1 other trees
- Does this by
  - maintaining for each such tree one *parent* variable ($parent_{rootID}$) and one *children* variable ($children_{rootID}$)
  - forwarding in every round all information that needs to be forwarded for every such tree (all in parallel)

2. Maximum Prevails

Main Idea:

- A node always maintains and forwards the information related to a single tree
  - despite the fact that many such trees are trying to evolve in parallel
  - the tree of the root with the maximum id from those heard so far is preferred

Pseudocode

Algorithm MaxBFS
Code for processor u_i, $i \in \{0, 1, \ldots, n - 1\}$:

Initially *parent* = ⊥, *children* = ∅, *maxRoot* := *myID*, *root* := true

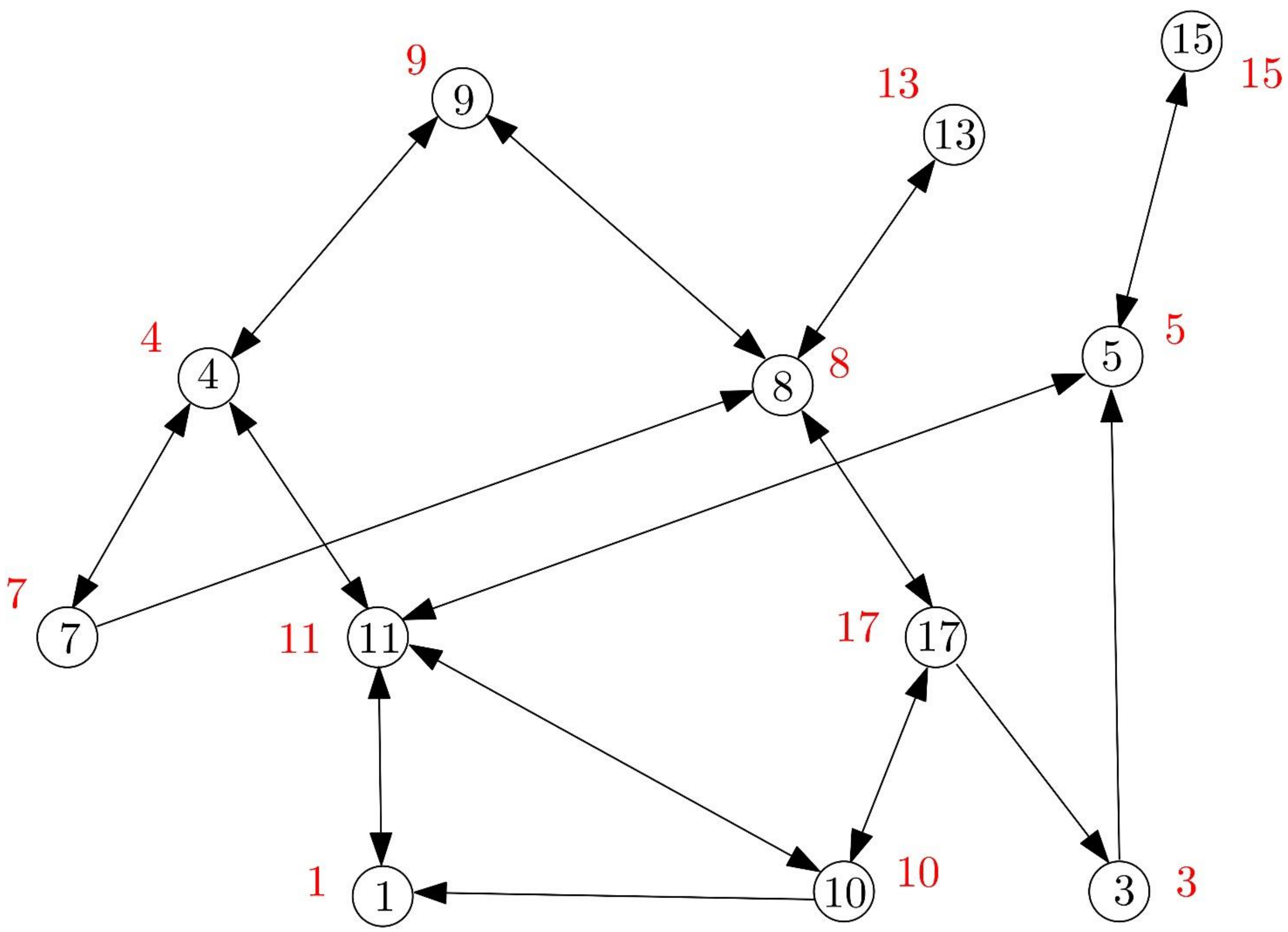
if *round* = 1 then // all trees start
  send ⟨*myID*⟩ to all neighbours
  parent := *myID*

upon receiving ⟨$rootID_j$⟩ from neighbours *N*:
  if *maxRoot* < $\max_j\{rootID_j\}$ then // u_i hears of a new prevailing root
    if *root* := true then *root* := false
    maxRoot := $\max_j\{rootID_j\}$
    parent := $u \in N$ where *u* just sent $\max_j\{rootID_j\}$ // select one of those that sent
    children := ∅ // get rid of previous children // the max, arbitrarily as parent
    send ⟨"parent"⟩ to *u*
    send ⟨*maxRoot*⟩ to all neighbours except those from which you just received it
  else do nothing

upon receiving ⟨"parent"⟩ from neighbours *N*:
  add all $u_j \in N$ to *children*

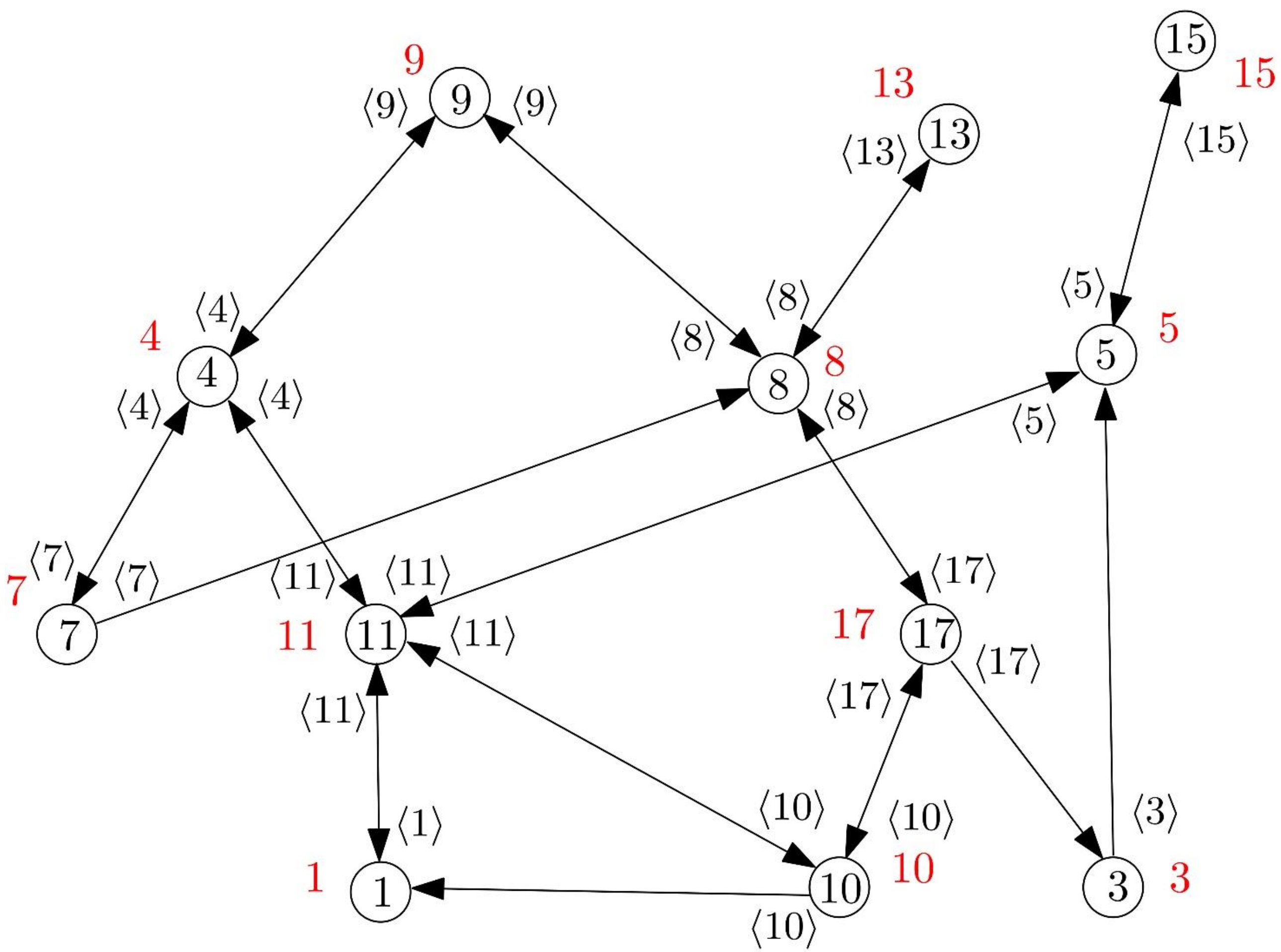
Example Execution

	1	3	4	5	7	8	9	10	11	13	15	17
parent	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥
children	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅



Example Execution

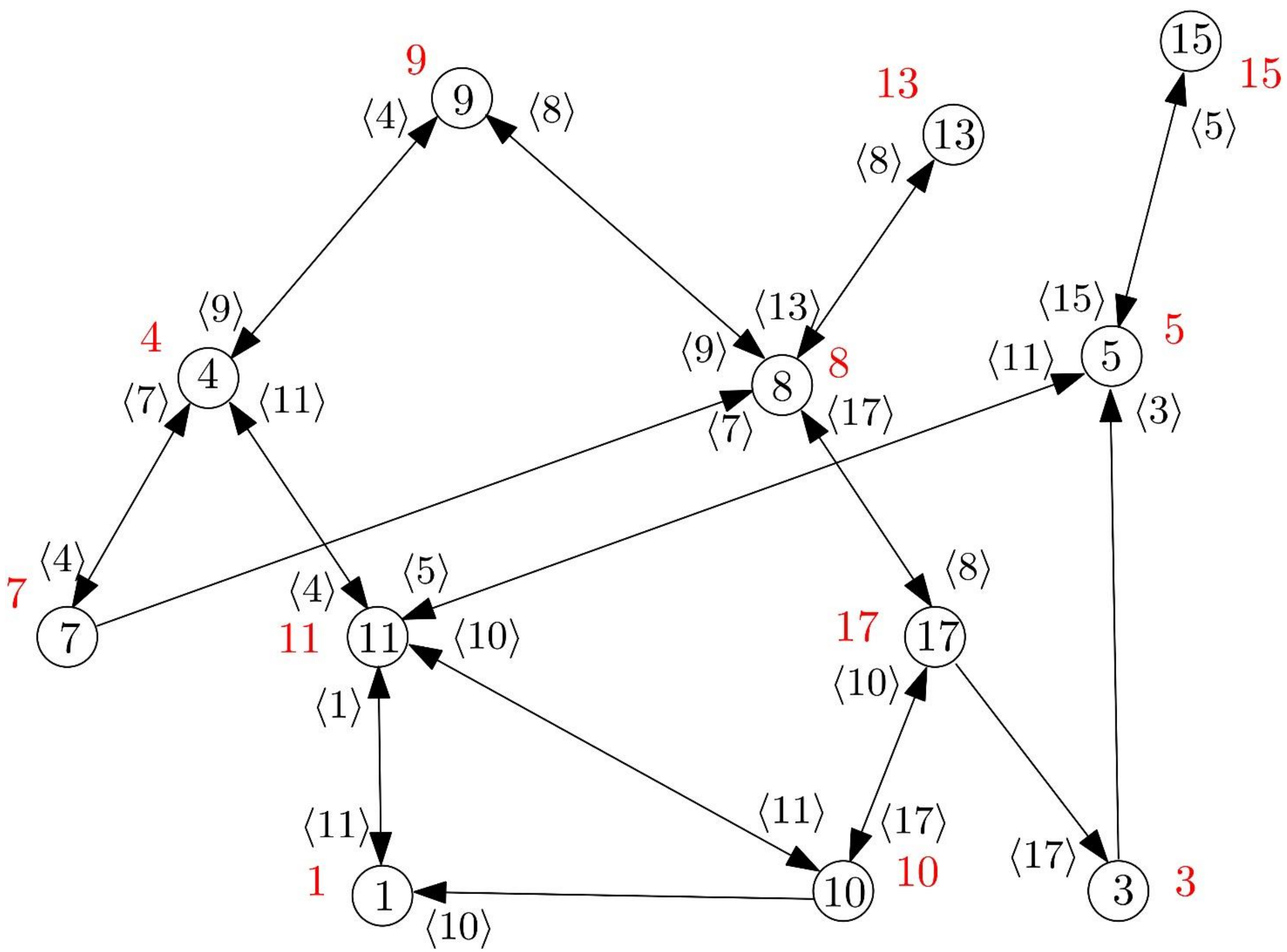
	1	3	4	5	7	8	9	10	11	13	15	17
parent	1	3	4	5	7	8	9	10	11	13	15	17
children	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅



$round = 1$

Example Execution

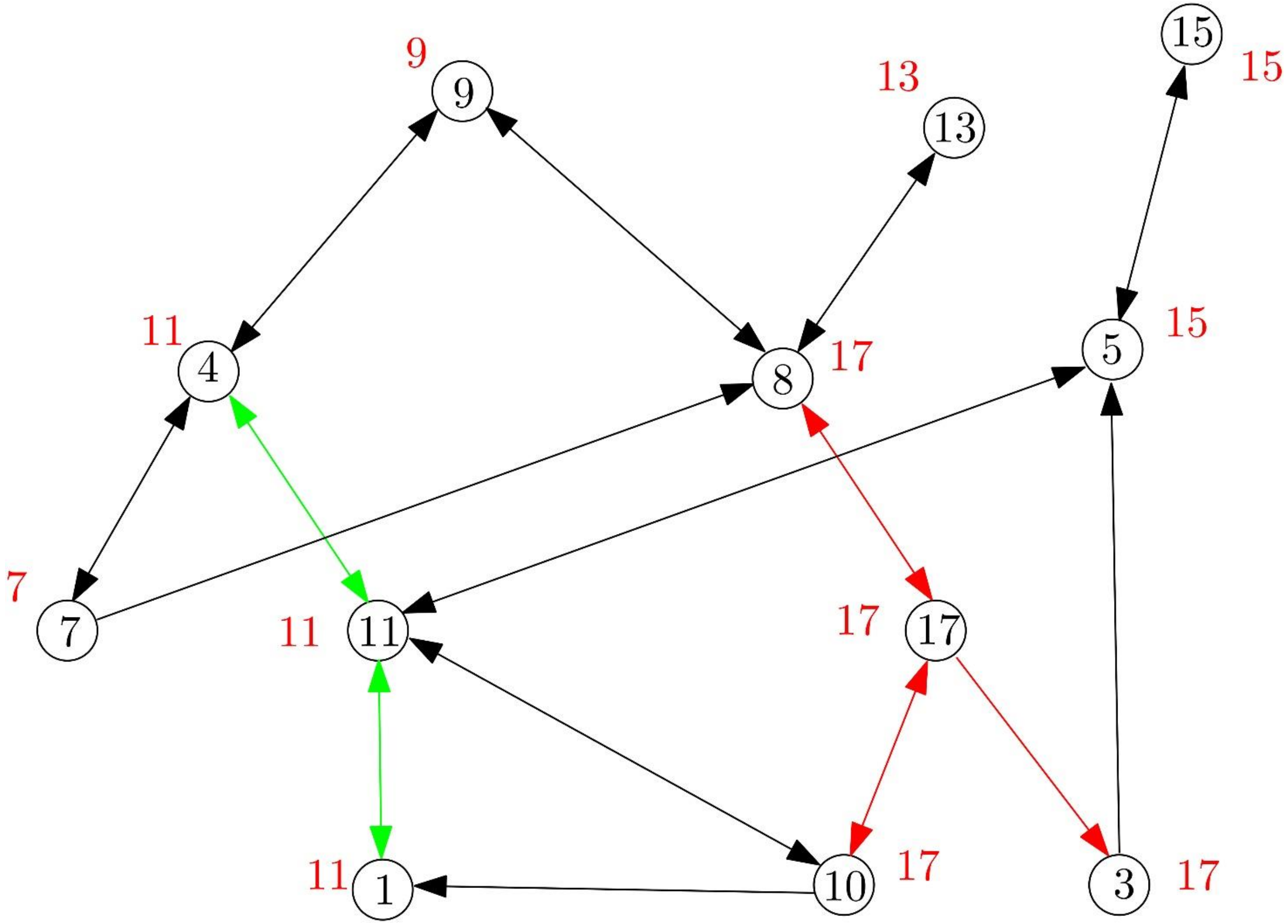
	1	3	4	5	7	8	9	10	11	13	15	17
parent	1	3	4	5	7	8	9	10	11	13	15	17
children	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅



$round = 1$

Example Execution

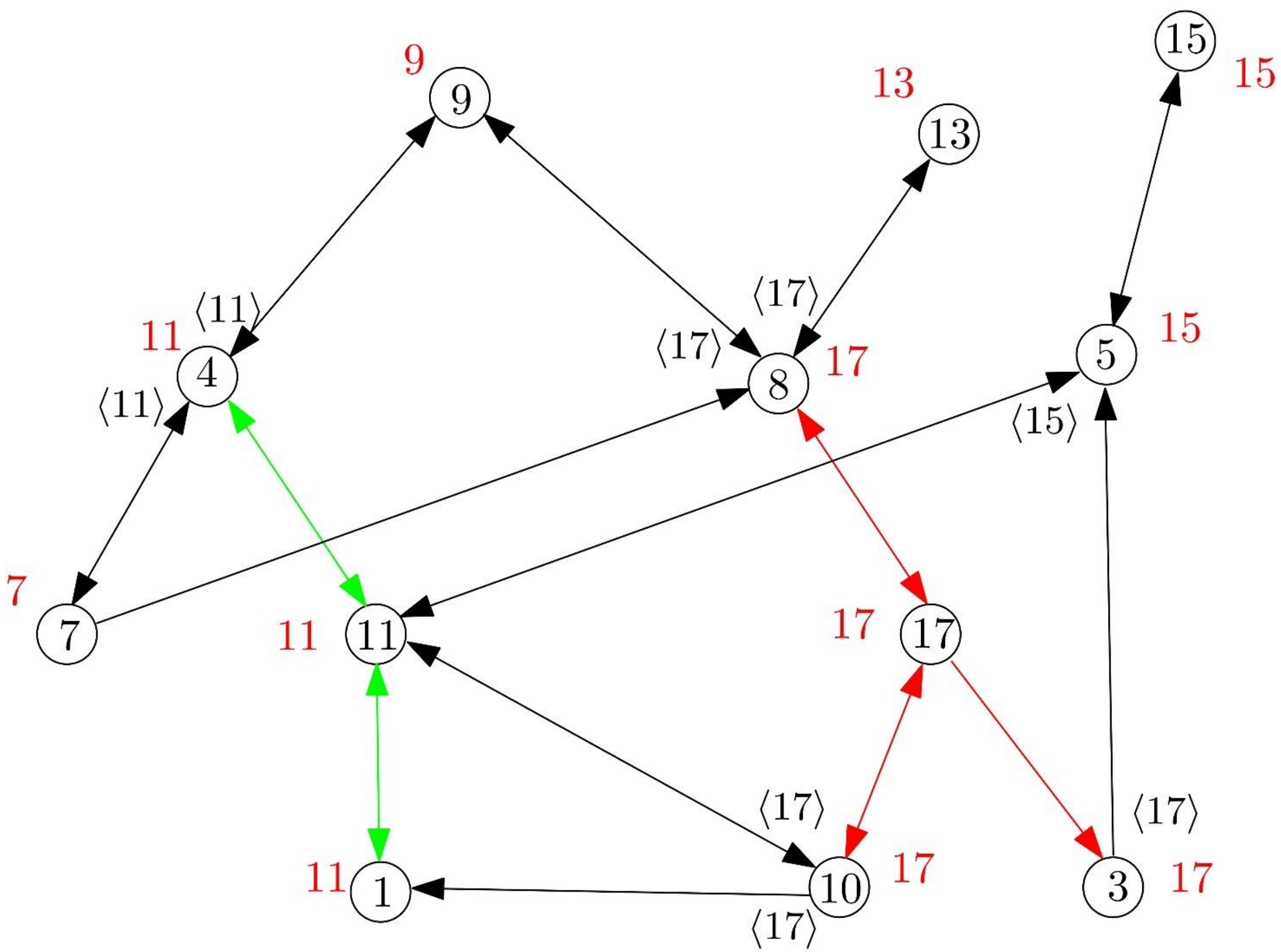
	1	3	4	5	7	8	9	10	11	13	15	17
parent	11	17	11	15	7	17	9	17	11	13	15	17
children	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅



$round = 2$

Example Execution

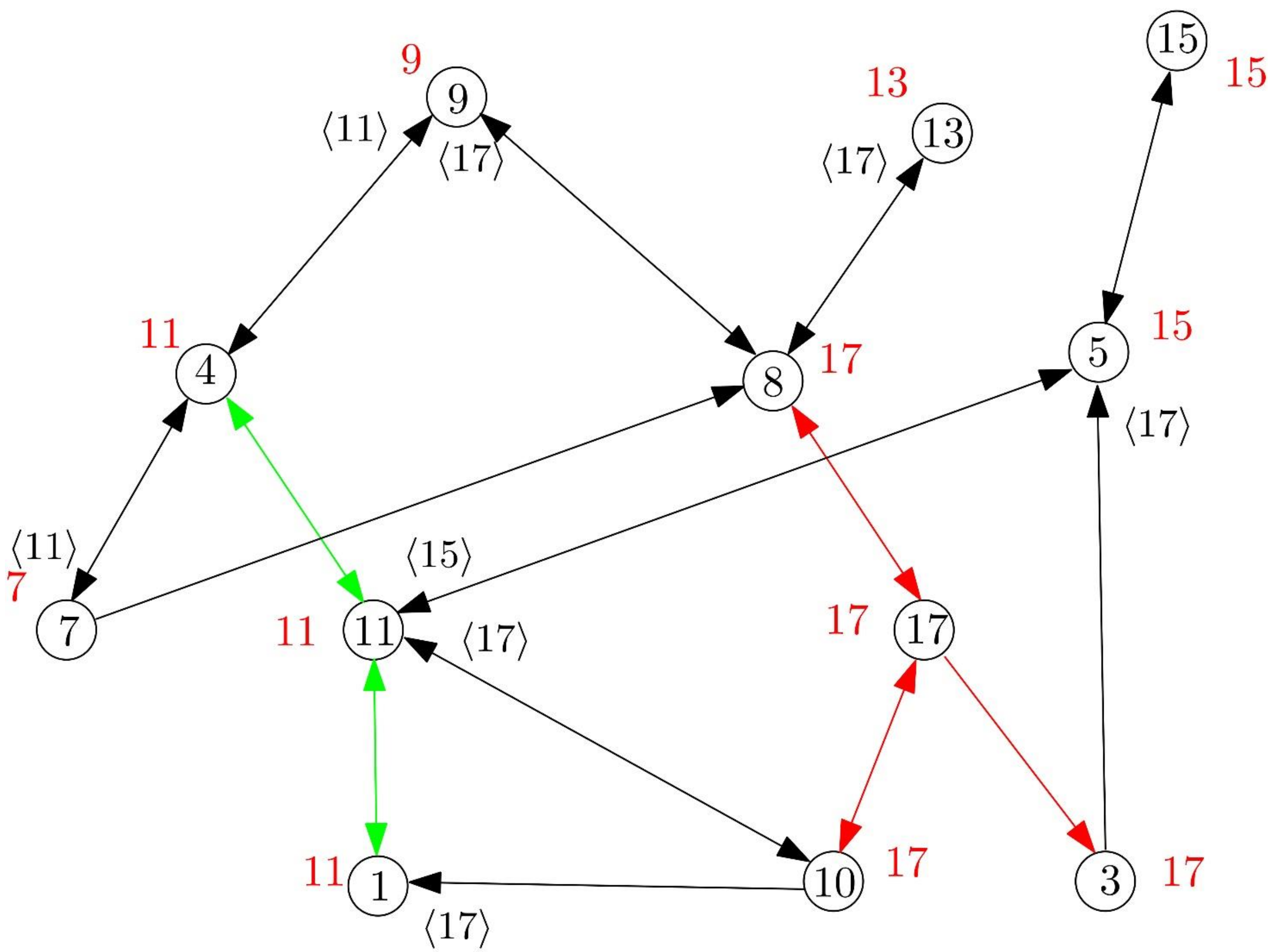
	1	3	4	5	7	8	9	10	11	13	15	17
parent	11	17	11	15	7	17	9	17	11	13	15	17
children	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅



$round = 2$

Example Execution

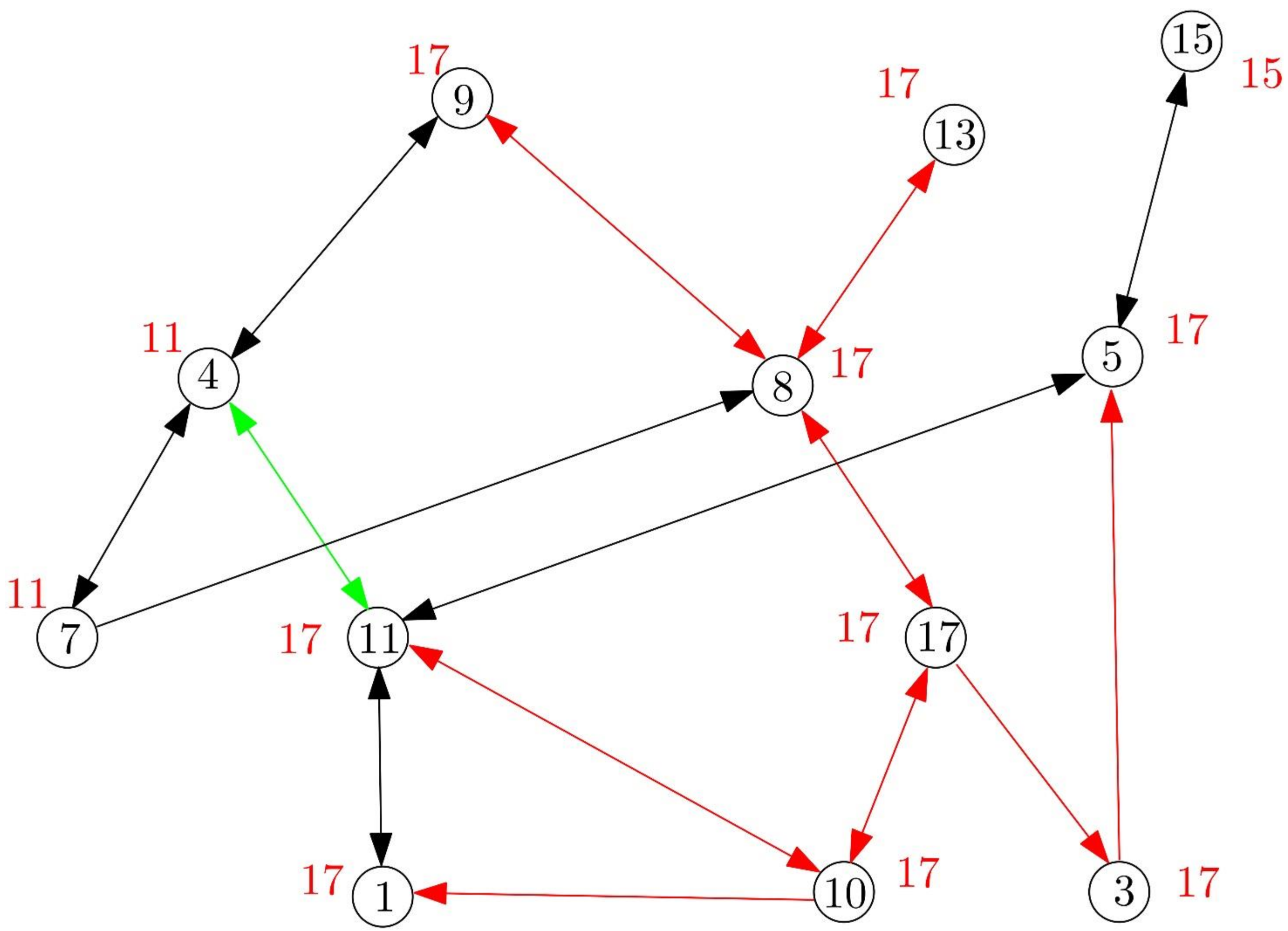
	1	3	4	5	7	8	9	10	11	13	15	17
parent	11	17	11	15	7	17	9	17	11	13	15	17
children	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅



$round = 2$

Example Execution

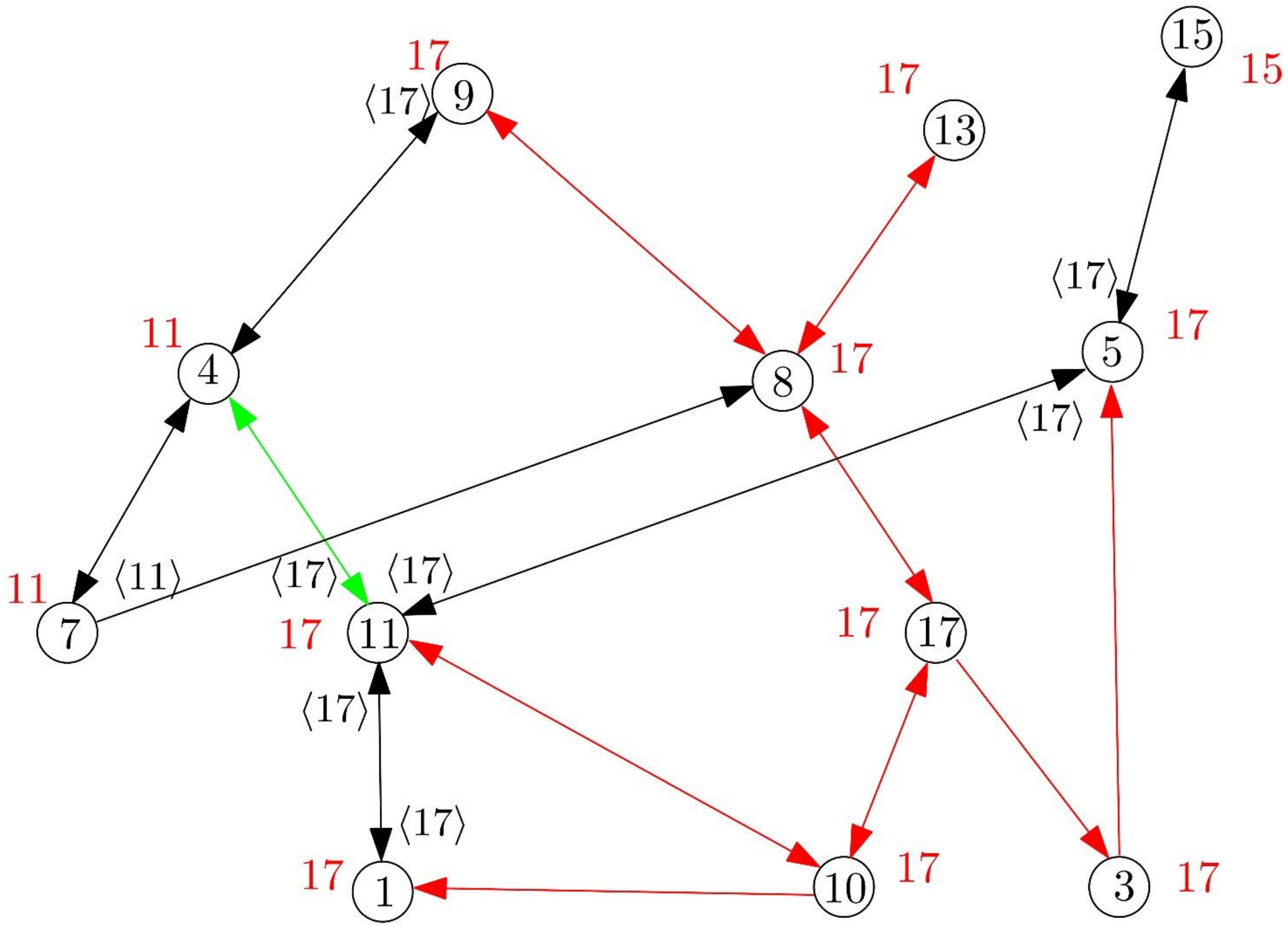
	1	3	4	5	7	8	9	10	11	13	15	17
parent	10	17	11	3	7	17	8	17	10	8	15	17
children	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	5	3,8,10



$round = 3$

Example Execution

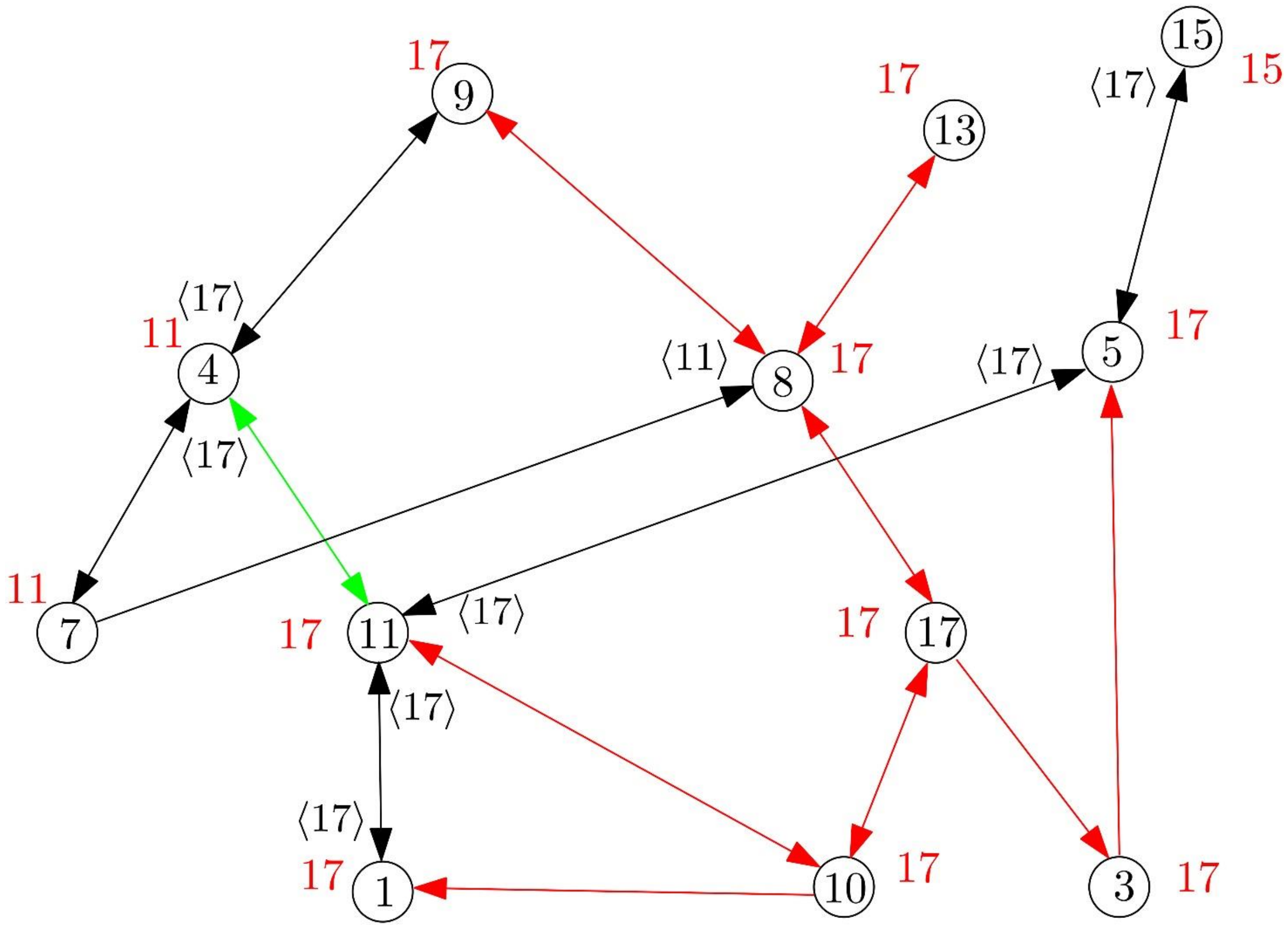
	1	3	4	5	7	8	9	10	11	13	15	17
parent	10	17	11	3	7	17	8	17	10	8	15	17
children	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	5	3,8,10



$round = 3$

Example Execution

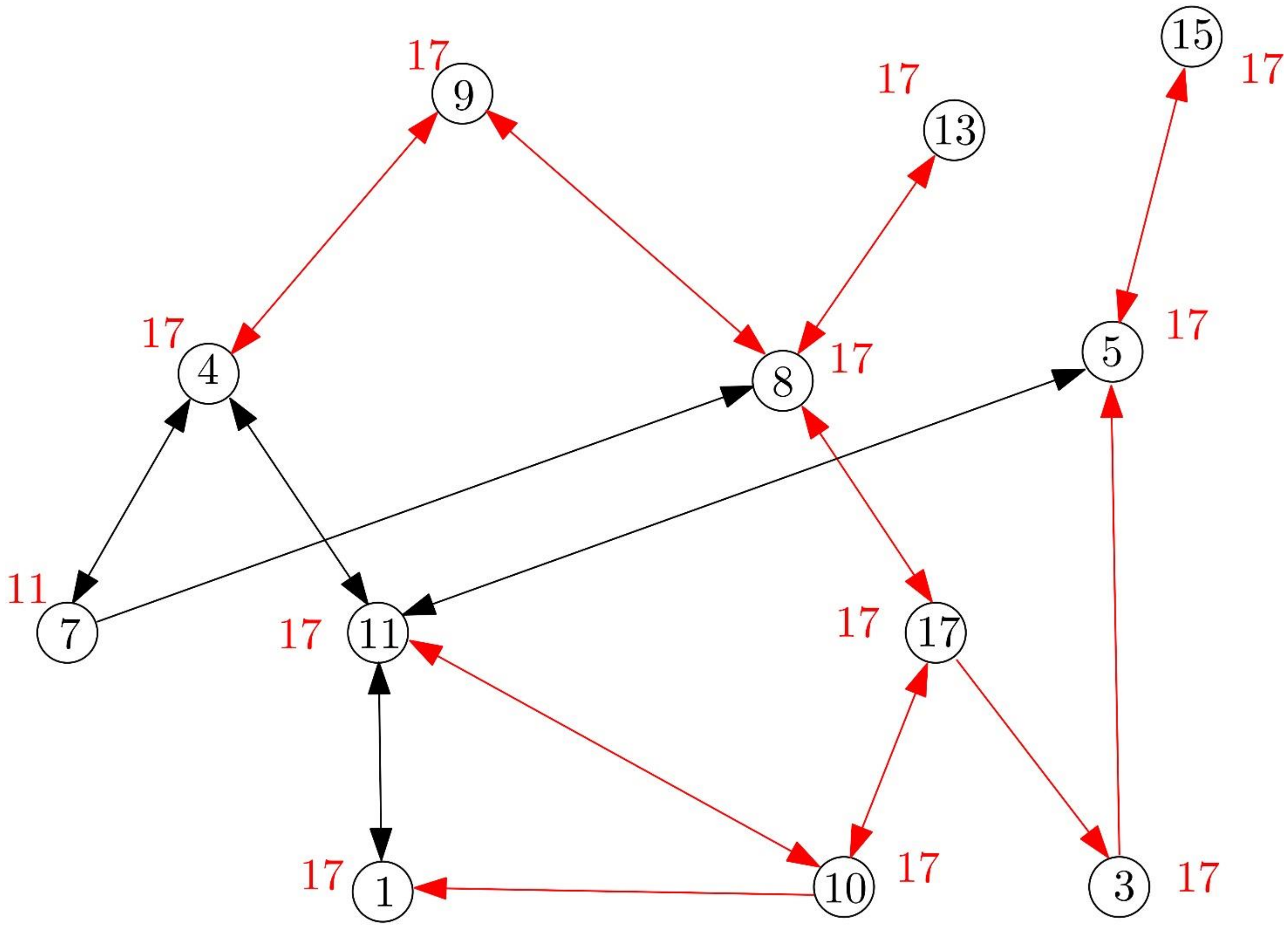
	1	3	4	5	7	8	9	10	11	13	15	17
parent	10	17	11	3	7	17	8	17	10	8	15	17
children	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	5	3,8,10



$round = 3$

Example Execution

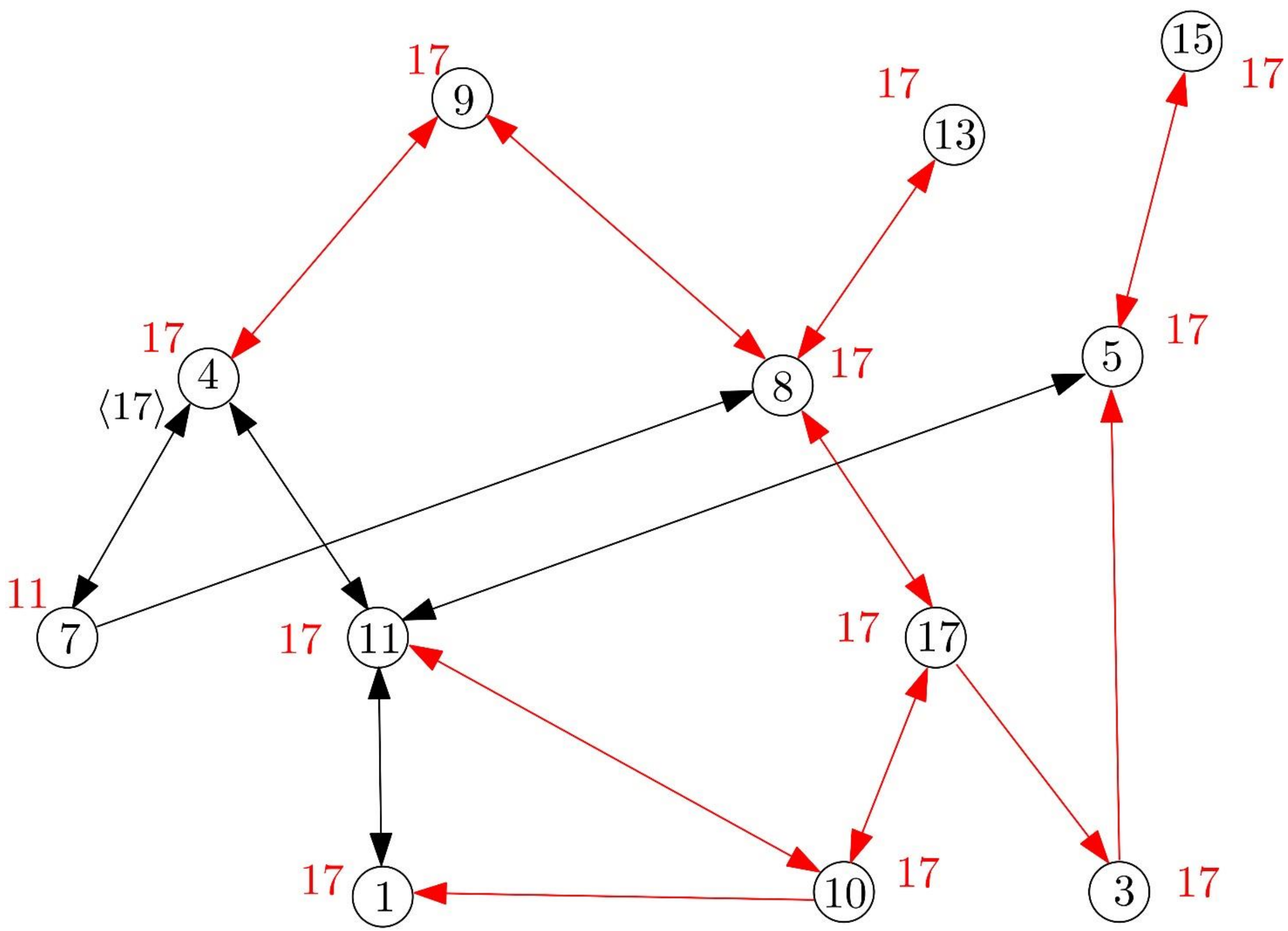
	1	3	4	5	7	8	9	10	11	13	15	17
parent	10	17	9	3	7	17	8	17	10	8	5	17
children	∅	5	∅	∅	∅	9,13	∅	1,11	∅	∅	∅	3,8,10



$round = 4$

Example Execution

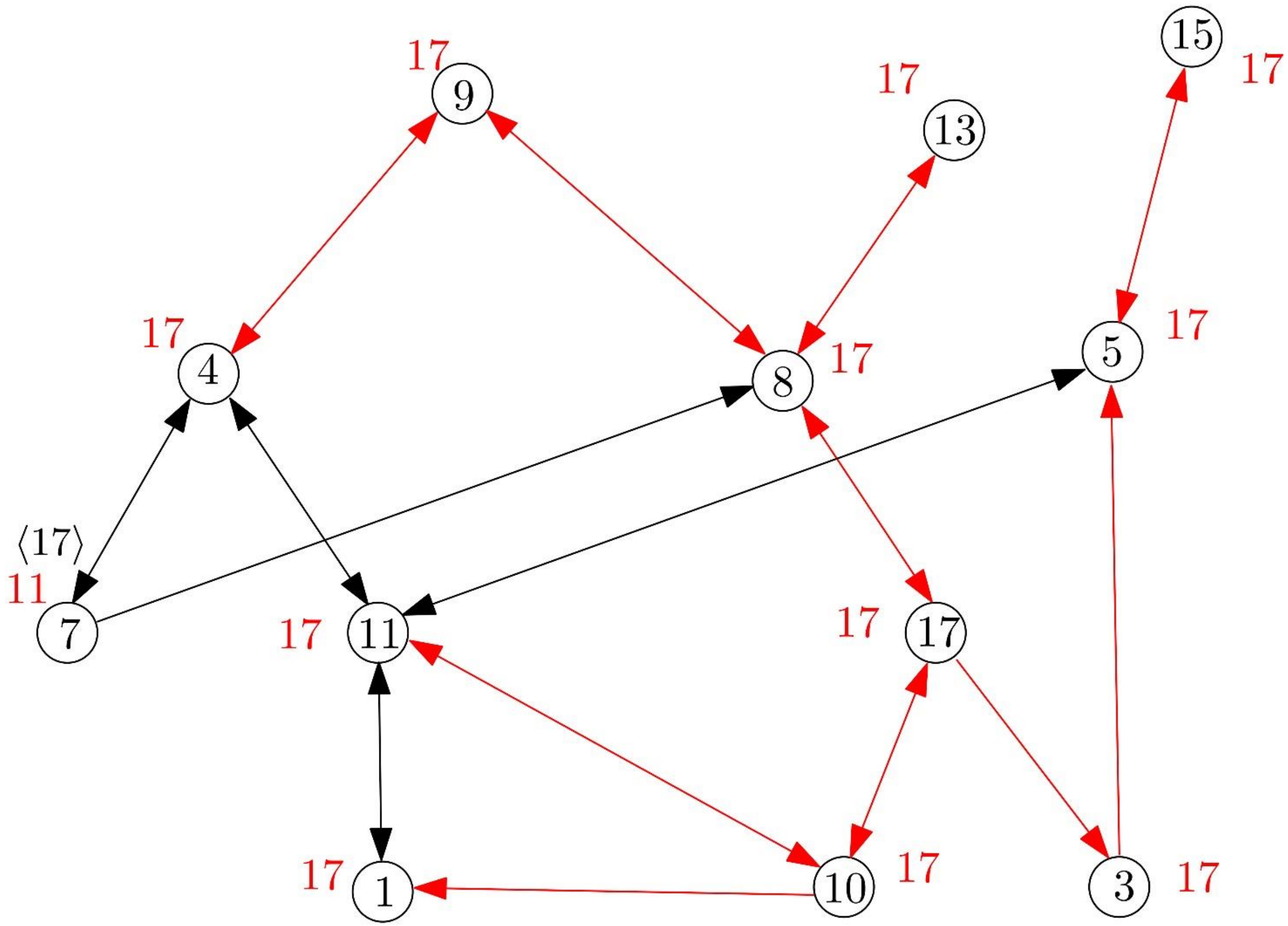
	1	3	4	5	7	8	9	10	11	13	15	17
parent	10	17	9	3	7	17	8	17	10	8	5	17
children	Ø	5	Ø	Ø	Ø	9,13	Ø	1,11	Ø	Ø	Ø	3,8,10



$round = 4$

Example Execution

	1	3	4	5	7	8	9	10	11	13	15	17
parent	10	17	9	3	7	17	8	17	10	8	5	17
children	∅	5	∅	∅	∅	9,13	∅	1,11	∅	∅	∅	3,8,10

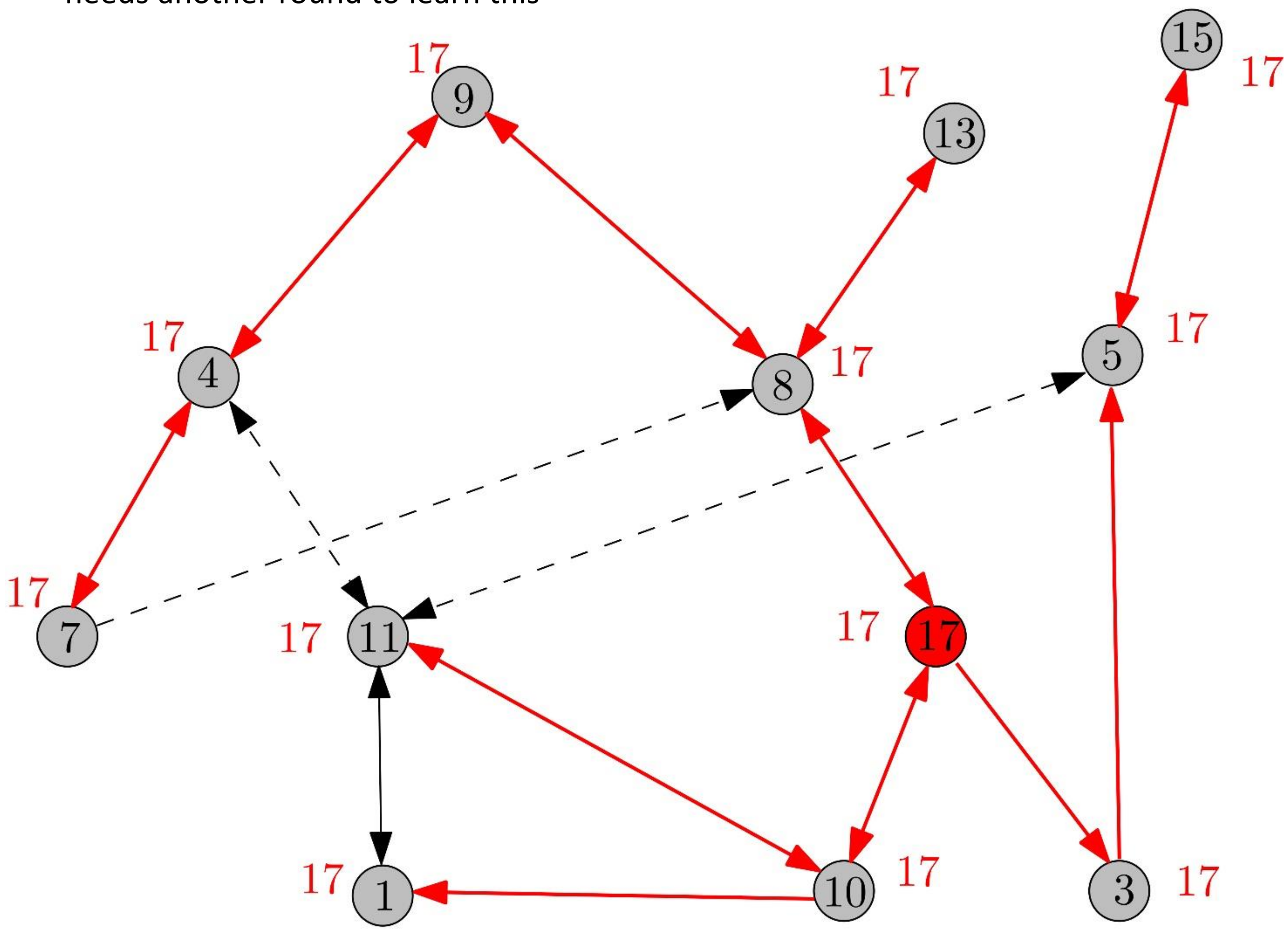


$round = 4$

Example Execution

	1	3	4	5	7	8	9	10	11	13	15	17
parent	10	17	9	3	4	17	8	17	10	8	5	17
children	Ø	5	7 *	15	Ø	9,13	4	1,11	Ø	Ø	Ø	3,8,10

* needs another round to learn this



$round = 5$

Applications (assuming bidirectional networks)

- Broadcast
  - We pay the tree construction once
  - Then, we can keep using it for broadcast that assumes a given spanning tree and a root
- Global computation
  - The root can collect all information contained in other processors
  - Convergecast towards the root
  - Once the root has this, can compute any function on those values, e.g., find max, min, average, …
  - Can then broadcast the result to all processors

Applications

- Leader Election
  - So far, were assuming some additional knowledge (e.g., diameter D)
  - The algorithm we described elects a unique leader without knowing n or D
  - It is the root whose tree prevails
  - *How can we make it aware that it has been elected and allow it to terminate?* (simpler to answer if we keep all trees in parallel)
- Computing the Diameter
  - Again, more straightforward if we keep all trees
  - Every root initiates a procedure with which shall estimate the depth d_i of its own BFS tree
  - Then a convergecast of the d_is of all other processors can allow a processor to compute $D := \max\{d_i\}$ which is the diameter
  - All processors do the same in parallel, therefore all learn the diameter

Summary

- Broadcast can be solved
  - given a root and a spanning tree
  - given a root only and constructing a BFS tree
- Leader election can be solved
  - special networks (e.g., rings)
  - general networks by assuming knowledge of the network diameter *D*
  - *if we knew only n instead?*
- Fortunately, both problems can be solved in general networks without any of these assumptions (only unique ids)
  - Parallel broadcasts (BFS trees) of all processors
  - Keep all and decide at the end
  - Maintain and forward only the maximum heard so far