Leader Election

Distributed Systems, 3rd Semester, BSc

Christian Fischer Pedersen
cfp@eng.au.dk

Section of Electrical and Computer Engineering
Department of Engineering
Aarhus University

Revised on October 12, 2020

Outline

Outline

The objective

The objective

- Getting a set of processes to elect a **single** leader

Relevance of leader

- Nodes cooperate in solving common task
- Most often, cooperation requires a leader, e.g.
  assign sub-tasks to participants and gather sub-results

Prerequisites

- **All** participating processes can **lead** and **call** for an election
- **Each** process P has a **unique** identifier

General approach

When to elect a leader

- When the system is **initialized**
- When a leader **fails**
- When a leader **retires** on purpose

General Approach

- Locate and designate the process with highest ID as leader
- Election algorithms differ in how they do this

Election costs

- Time complexity, space complexity, message complexity

Outline

BE: Background

Proposed by

- H. Garcia-Molina, "Elections in a distributed computing system," IEEE Trans. on Computers, vol. 31, no. 1, pp. 48–59, January 1982

Leader election process begins

- Leader halts processing
- If a process that was previously down comes back up, it holds an election.

End result

- One and **only one** coordinator
- The **bully** with highest ID **bullies** those with lower IDs

BE: Assumptions

- Each process has a unique ID
- Processes know each other's process ID
- Processes do not know which ones are currently alive
- Each process can compare IDs (e.g. to find the highest)
- All processes can inter-communicate
- A failed process is always detectable
- Any process can initiate an election
- Upon failure recovery, the process knows it failed

BE: Message types

- *Election*: Message to announce election
- *OK*: Msg in response to election msg:
  "OK I am alive, have a higher ID and will take over the election"
- *Coordinator*: Message to announce ID of new coordinator

BE: The algorithm illustrated

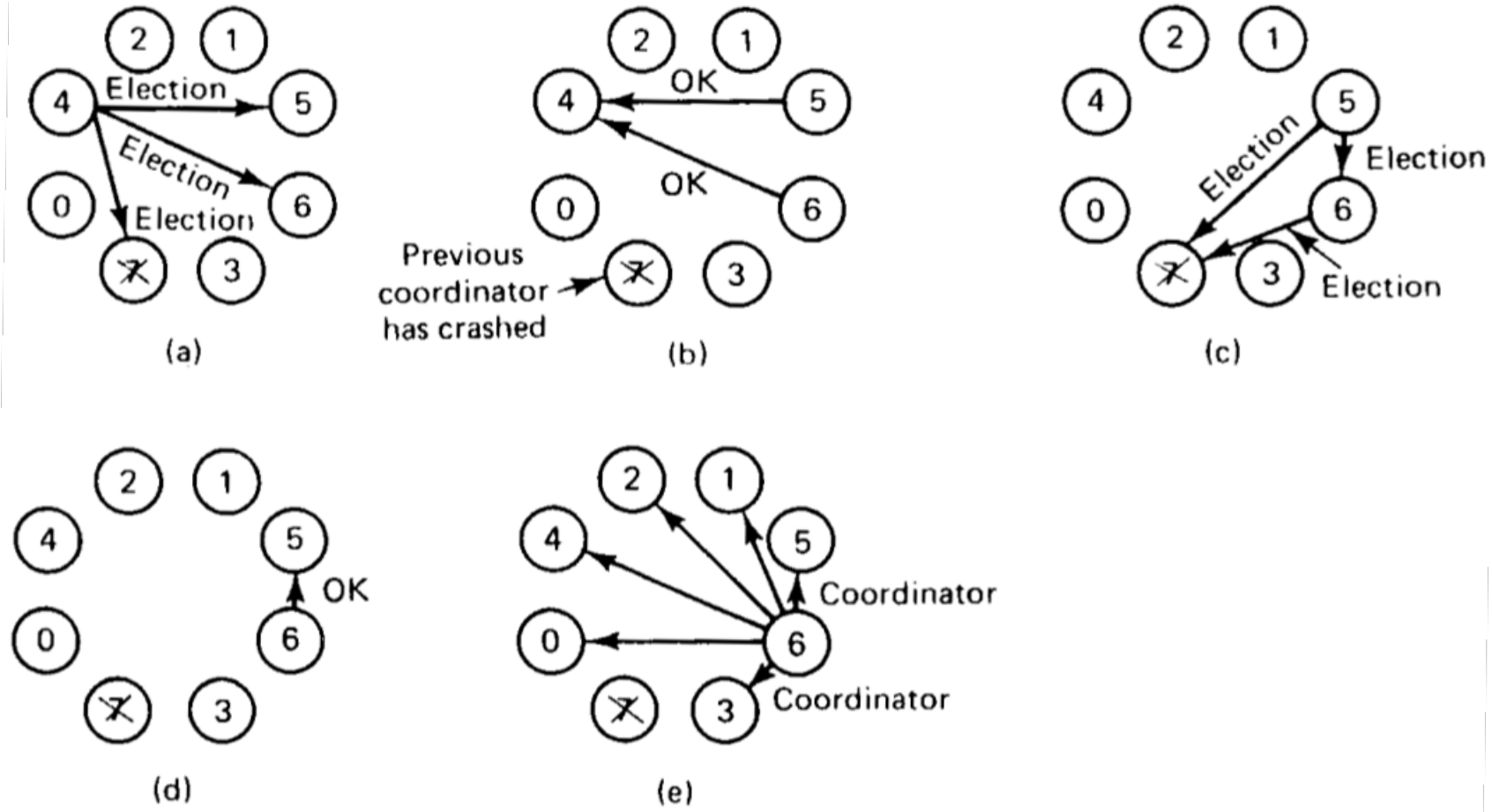


Fig. 3-12. The bully election algorithm. (a) Process 4 holds an election. (b) Processes 5 and 6 respond, telling 4 to stop. (c) Now 5 and 6 each hold an election. (d) Process 6 tells 5 to stop. (e) Process 6 wins and tells everyone.

BE: The algorithm in words

Consider processes $\{P_i\}_{i=0}^{N-1}$ and let $\text{id}(P_k) = k$. If P_k notices a non-responding coordinator, it initiates an election.

1. P_k sends an *Election* msg to all processes with higher IDs
2. P_k awaits *OK* message(s)
   - If **no** msg: P_k becomes leader and sends *Coordinator* messages to all processes with lower IDs
   - If msg: P_k drops out and awaits a *Coordinator* message

- If a process receives an *Election* message
  - Respond w/ *Coordinator* msg if it is the process w/ highest ID
  - Otherwise, respond w/ *OK* and take over the election
- If a process receives a *Coordinator* message, it treats sender as the coordinator

BE: Message complexity

Assume

- N processes and **one** election in progress

Best case: Failed process was leader

- It sends *Coordinator* msg to all other processes on recovery
- Immediate election: $N-1$ messages

Worst case: Initiator is process with lowest ID, i.e. P_0

- *Election* messages
  - P_0 sends $N-1$ and P_1 sends $N-2$ and ... $\Rightarrow$ $(N-1)+(N-2)+...+(N-(N-1))=$ Left for exercise...
- *OK* messages: Left for exercise...
- *Coordinator* messages: Left for exercise...
- Overall message complexity: Left for exercise...

Outline

LCR: Background

Proposed by

- E. G. Chang and R. Robert, "An improved algorithm for decentralized extreme-finding in circular configurations of processors," Communications of ACM, vol. 22, no. 9, pp. 281-283, 1979
- Note: The explanation in the course book by Marten van Steen is not fully equivalent with the explanation given in the paper by Chang and Robert. We will adhere to the book.

Leader election process begins

- Leader halts processing
- If a process that was previously down comes back up, it holds an election.

End result

- One and **only one** leader
- Process with highest ID elected leader

LCR: Assumptions

- Processes are arranged in a logical unidirectional ring
- All processes knows the structure of the ring
- Each process has a unique ID
- Processes know each other's process ID
- Processes do not know which ones are currently alive
- Each process can compare IDs (e.g. to find the highest)
- All processes can communicate with clockwise neighbor
- A failed process is always detectable
- Any process can initiate an election
- Upon failure recovery, the process knows it failed

LCR: Message types

- *Election*: Message to announce election
- *Coordinator*: Message to announce ID of new coordinator

LCR: The algorithm illustrated

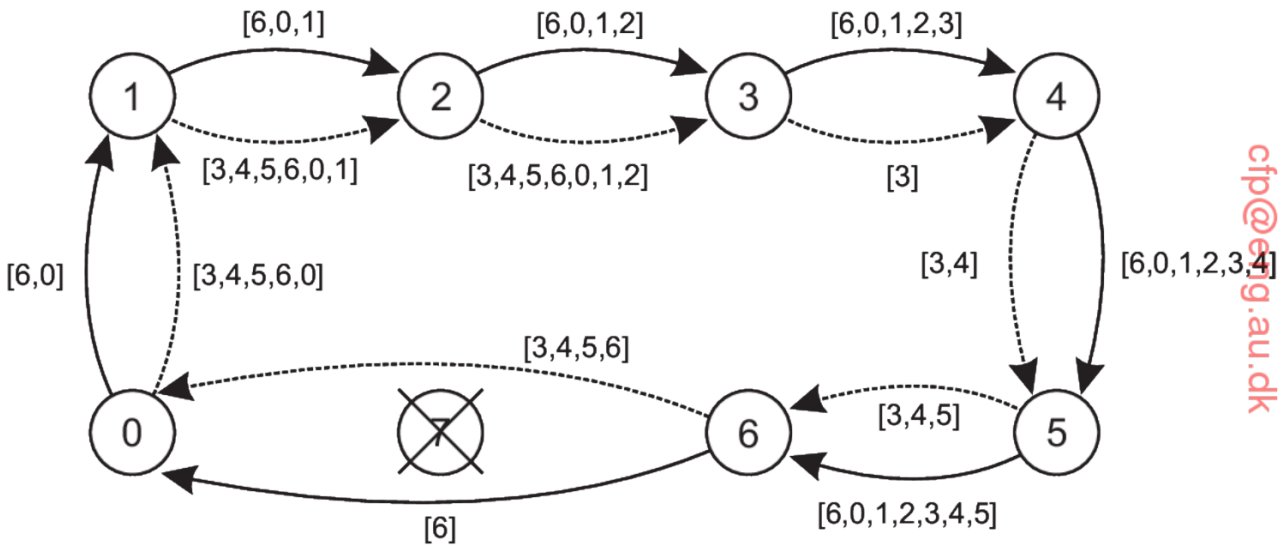


Figure 6.21: Election algorithm using a ring. The solid line shows the election messages initiated by P_6; the dashed one those by P_3.

LCR: The algorithm in words

- A process initiates an election if
  - it just recovered from failure or
  - if it notices that the leader has failed
- Initiator sends *Election* msg to closest live clockwise neighbor
  - Election message is forwarded around the ring
  - Each process adds its own ID to the *Election* message
- When *Election* message returns
  - Initiator picks node with highest ID
  - Sends a *Coordinator* message specifying the election winner
  - Also, IDs with all living members are attached in order to update all members of this
  - Coordinator message is removed when it has circulated once
- Multiple elections can be in progress simultaneously

LCR: Message complexity

Assume

- N processes and **one** election in progress

Best case $=$ Worst case

- Always $2N$ messages, i.e. $\mathcal{O}(N)$
- One round for the election message
- One round for the coordinator message

Outline

MAH: Background

Proposed by

- Vasudevan S., Kurose J.F. and Towsley D.F., "Design and Analysis of a Leader Election Algorithm for Mobile Ad Hoc Networks". In 12th International Conference on Network Protocols, pp. 350–360, Los Alamitos, CA., Oct. 2004. IEEE Computer Society Press.

Properties

- For wireless/mobile ad hoc networks
- Handles failing nodes and network partitions

End result

- The best (wrt. **resources**) leader elected

MAH: Assumptions

- Each process has a unique ID
- Processes know each other's process ID
- Processes do not know which ones are currently alive
- Each process can compare IDs (e.g. to find the highest)
- All processes can inter-communicate
- A failed process is always detectable
- Any process can initiate an election
- Upon failure recovery, the process knows it failed
- The course book disregards that
  - nodes can move physically, i.e. topology is not static
  - message passing is unreliable particularly in wireless net

MAH: Message types

- *Election*: Message to announce election
  - To neighbors within range
  - *Election-ID*: Each election process is tagged with an unique ID. When multiple elections are initiated, each node will decide to join only one election, i.e. the one with the highest ID (stopping any running participation in other elections.)
- *Acknowledgement*: Msg in response to election msg
  - Acknowledge the receipt of *Election* msg.

MAH: The algorithm illustrated

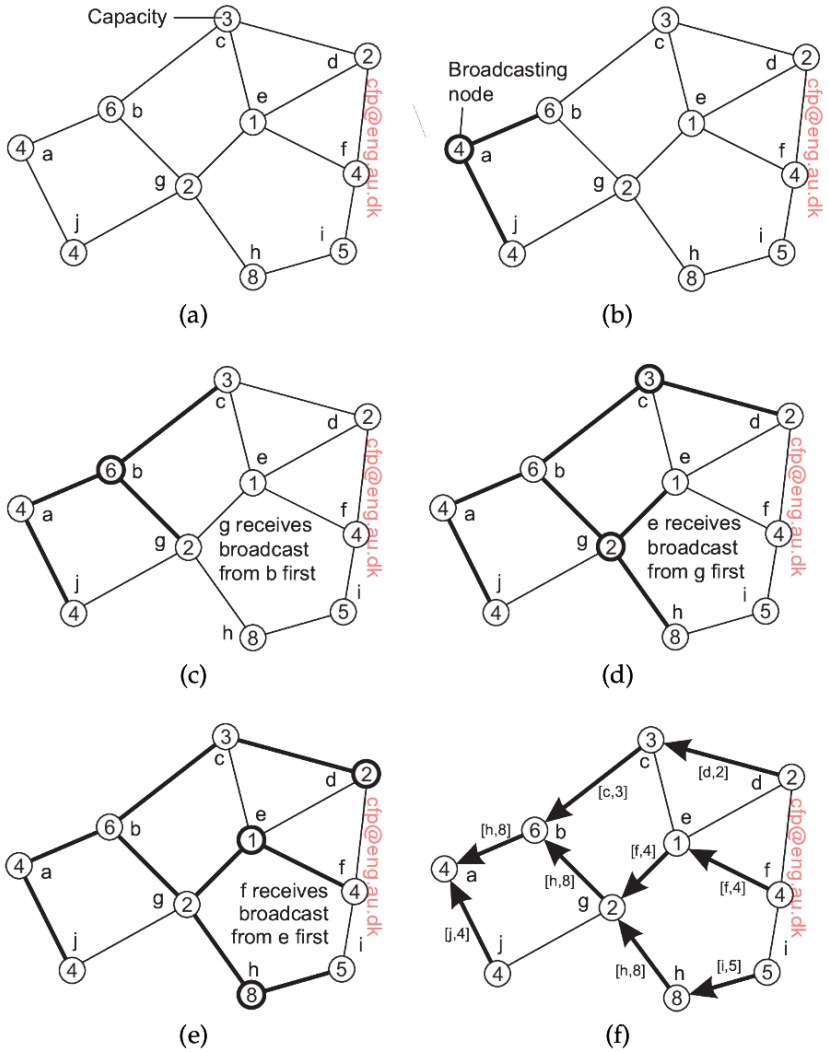


Figure 6.22: Election algorithm in a wireless network, with node a as the source. (a) Initial network. (b)–(e) The build-tree phase (last broadcast step by nodes f and i not shown). (f) Reporting of best node to source.

MAH: The algorithm in words

- Any node can initiate an election by broadcasting *Election* msg
- Nodes within range receive the *Election* msg
- When a node receives an *Election* msg for the first time
  1. it designates the sender as its *parent*
  2. sends out *Election* msg to nodes within range (except parent)
  3. waits for ack's to return before ack'ing *Election* from parent.
     The *Acknowledgement* **also** contain a resource status.
     The resource status reported back is of the **best** (by some metric and comparison) of the incoming statuses.
- When a node receives an *Election* msg from a non-parent node, it merely acknowledges the receipt.
  The *Acknowledgement* **does not** contain a resource status

Outline

HS: Background

Proposed by

- D.S. Hirschberg and J.B. Sinclair, "Decentralized extrema-finding in circular configurations of processors", Communications of the ACM, 23 (11), 1980

Leader election process begins

- Leader halts processing
- If a process that was previously down comes back up, it holds an election.

End result

- One and **only one** coordinator
- Leader (highest ID) election in bidirectional rings
- Message complexity $\mathcal{O}_m = (n \log n)$

HS: Assumptions

- Each process has a unique ID
- Processes do not know each other's process ID
- The process with highest ID is elected
- Ring size (no. of processes) initially unknown to the processes
- Communication is asynchronous

HS: Message types

- `SendBoth`: Send messages in both directions
- `SendPass`: Pass received message on
- `SendEcho`: Send response message back

HS: Neighbourhood elections

Elections are performed in neighbourhoods

- The 2^k-**neighbourhood** of a process p is the **set of processes** that are at distance at most 2^k from p (2^k left-neighbours plus 2^k right-neighbours)
- Phases $k = 0, 1, 2, ...$
- For each k, process i sends its ID through 2^k bidirectionally
- If both ID tokens return, then i continues to phase $k + 1$

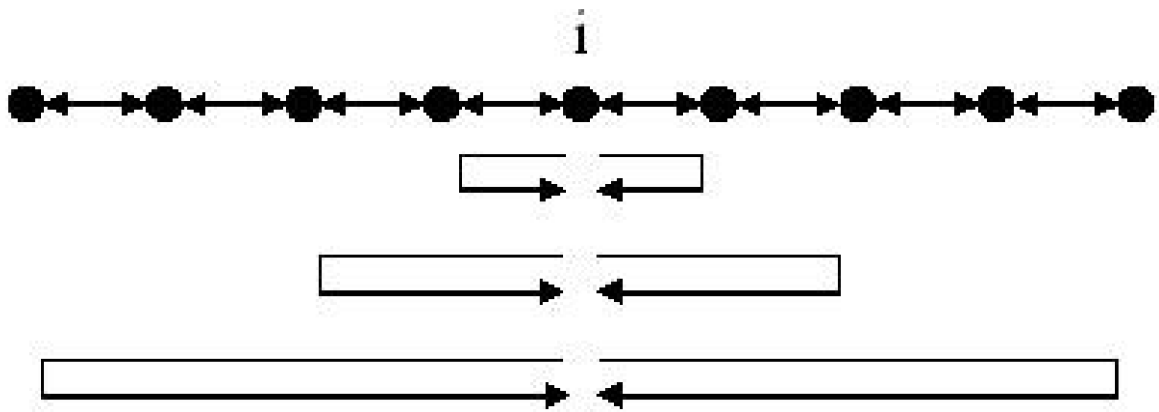


Figure: Execution of HS (by courtesy of Mitsou Valia)

HS algorithm: Election phases

Elections are carried out in (asynchronous) phases

- Phases $k = 0, 1, 2, ...$
- Process p_i tries to become a leader in phase k in its 2^k-neighbourhood
- Only if p_i is the winner, i.e. it has the highest ID in its 2^k-neighbourhood, it can proceed to phase $k + 1$
- In each phase, the process p_i sends out tokens containing ID_i in both directions
- Tokens travel distance 2^k and return to p_i
- The size of the neighbourhood doubles in each phase (i.e. 2^k)
- Fewer p's proceed to higher phases, until a single winner gets elected in the whole ring.

HS algorithm: Sending messages 1/2

- Initially, all p's initiate a candidancy (phase 0), e.g. after having received a broadcasted request for electing a leader
- The **election-messages** sent by candidates contain 3 fields:
  - The ID of the candidate.
  - The current phase number k
  - A hop counter d, which is initially 0 and is incremented by 1 whenever the message is forwarded to the next process p
- If a p_j receives a msg with `(r, k, d)` where $d = 2^k$, then it is the last process in the 2^k -neighbourhood of p_r with ID $= r$

HS algorithm: Sending messages 2/2

- If the p_i receiving the election message has a greater ID, then it swallows the message, otherwise it relays it to the same direction, after incrementing d by 1.
- If the message makes it till the 2^k-distance p_i, then p_i sends back a **reply-message**, which is forwarded till it reaches the candidate p_r
- If the candidate receives replies from both directions, then it is the winner of its 2^k-neigbourhood
- A p_i that receives an election message with its own ID is the leader of the ring.
- The leader should also announce itself to all other nodes

HS algorithm: Correctness

- Messages from process with highest ID are never discarded
- Thus, the correct leader is elected
- No other process ID can traverse the entire ring
- Therefore, no one else is elected

HS algorithm: Example (phase 0)

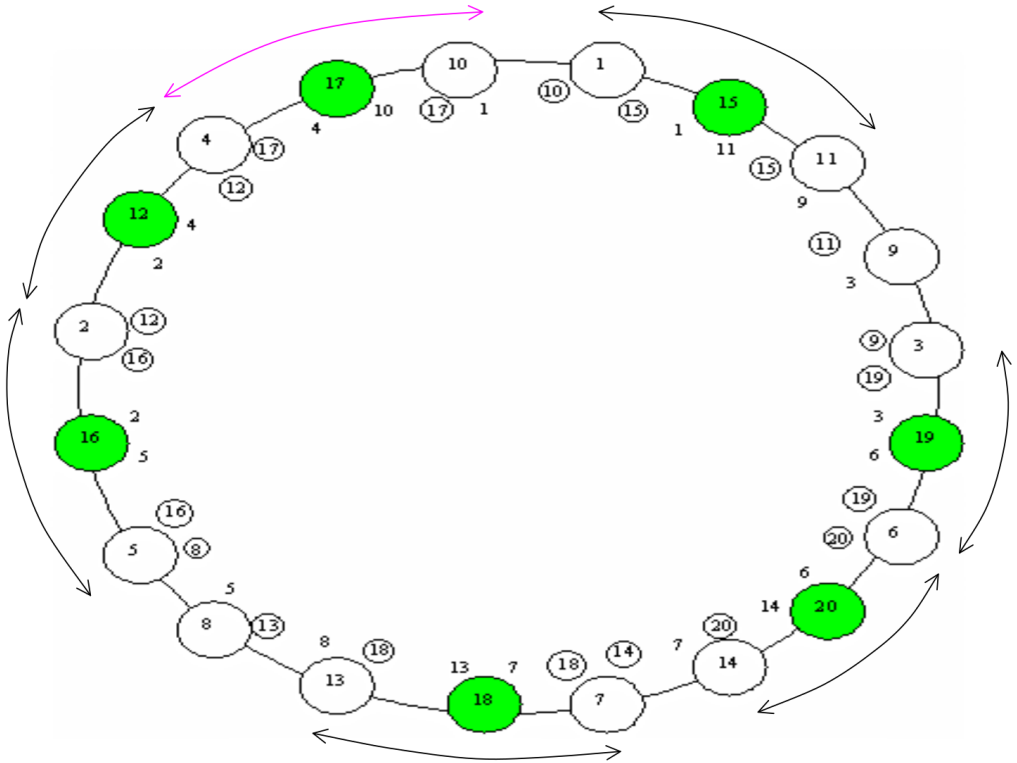
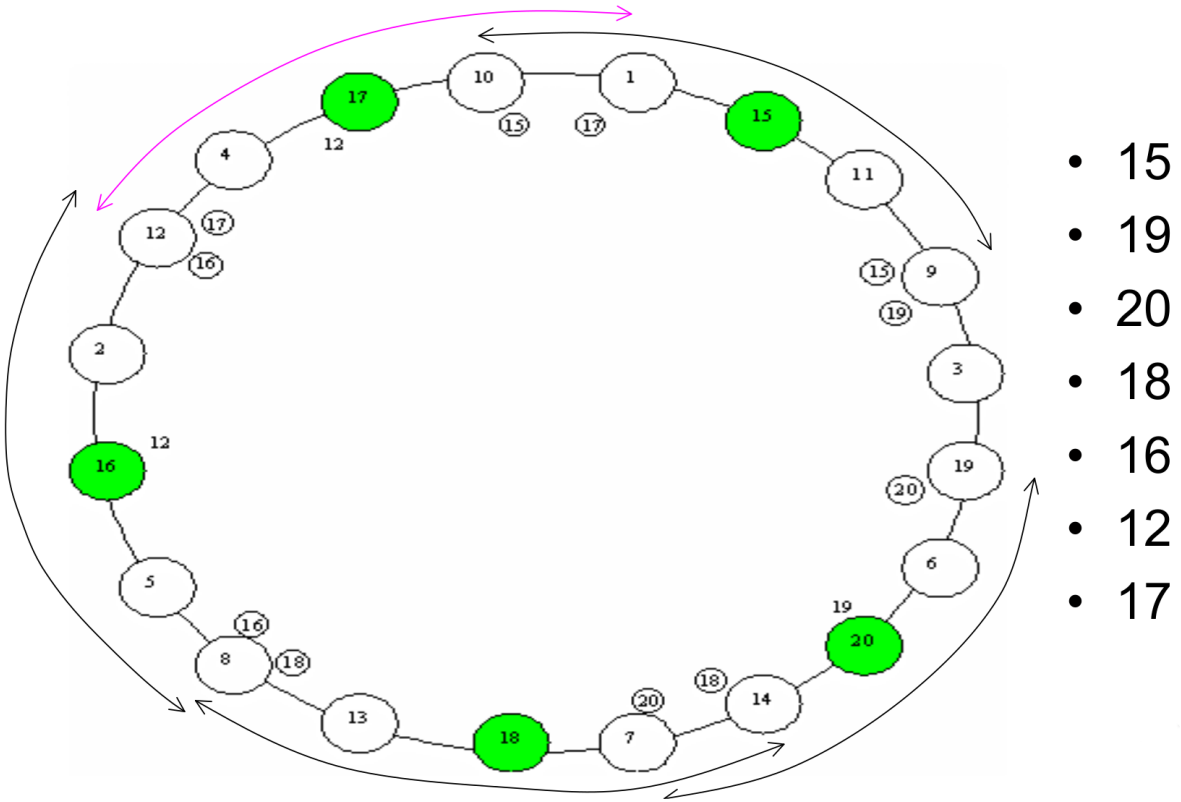


Figure: Fig. by courtesy of Mitsou Valia

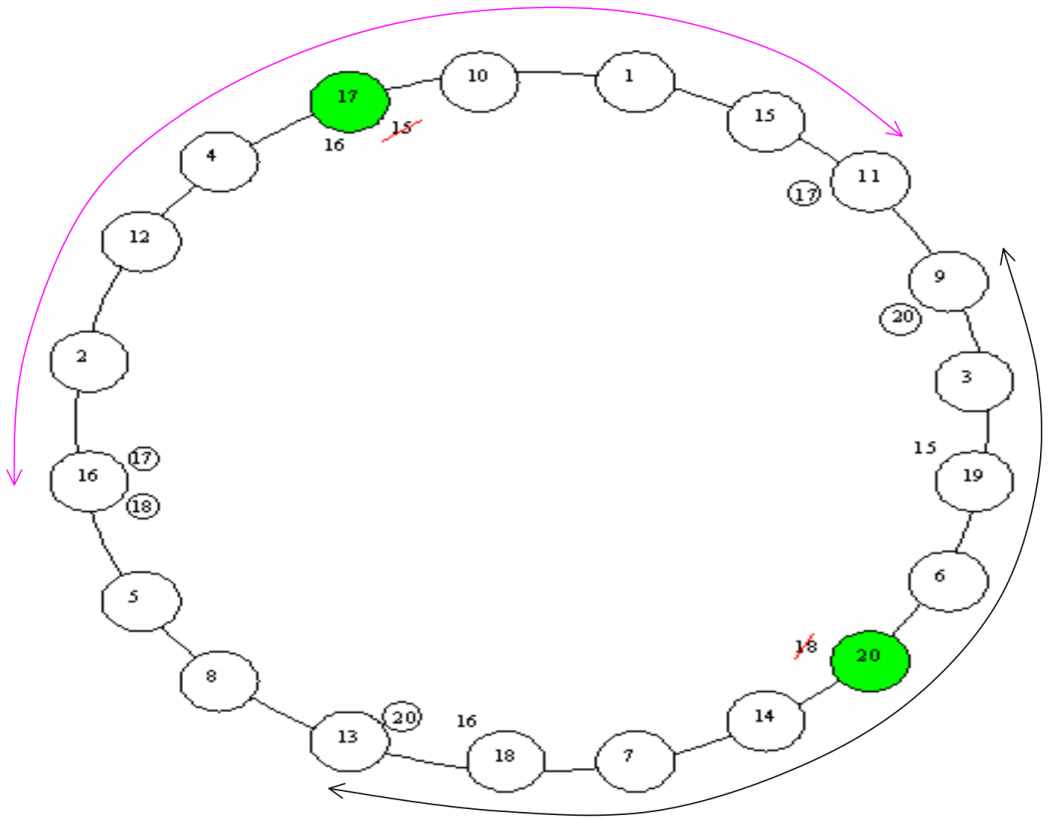
HS algorithm: Example (phase 1)



- 15
- 19
- 20
- 18
- 16
- 12
- 17

Figure: Fig. by courtesy of Mitsou Valia

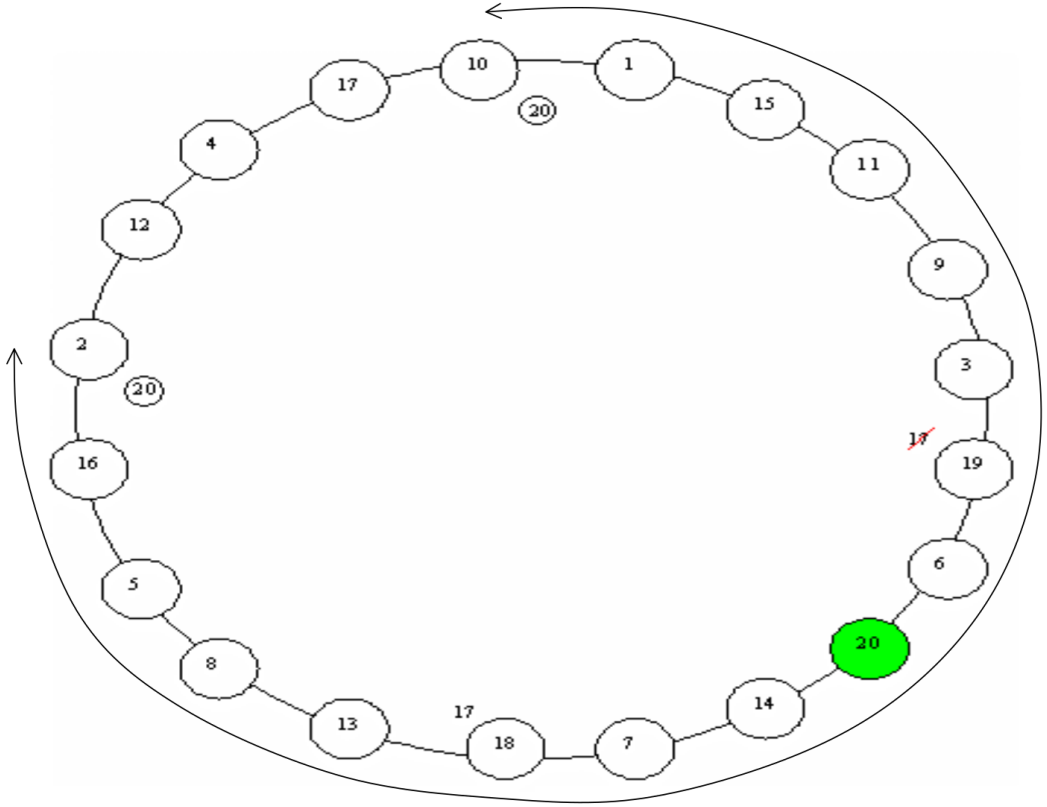
HS algorithm: Example (phase 2)



- 15
- 20
- 18
- 16
- 17

Figure: Fig. by courtesy of Mitsou Valia

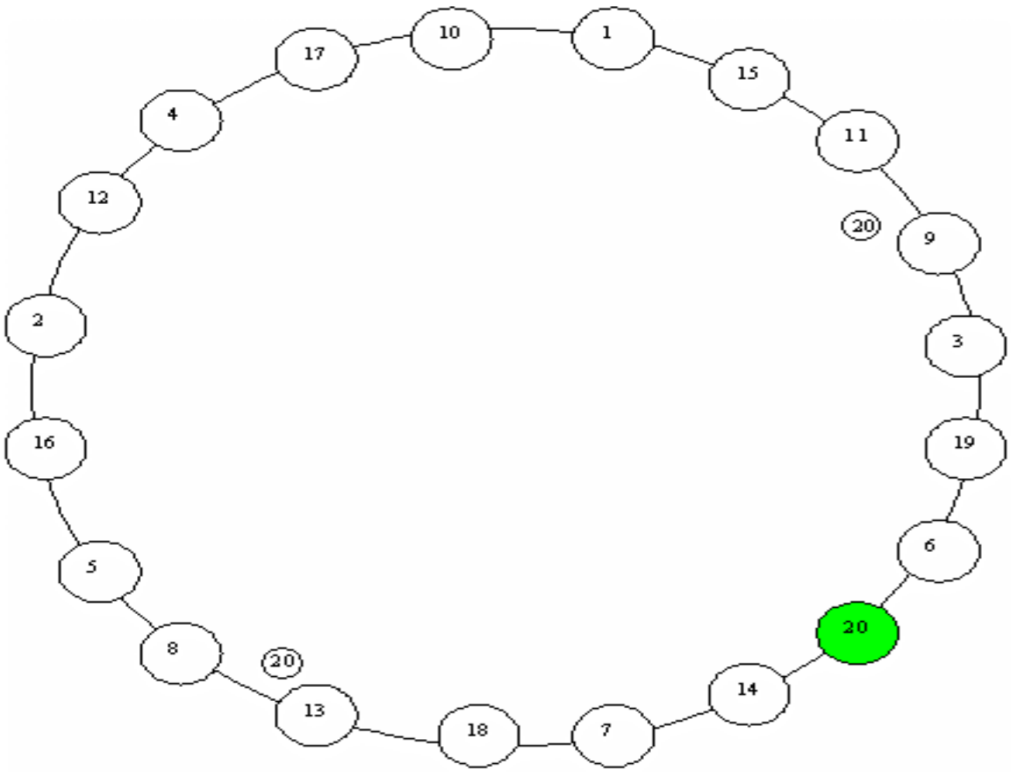
HS algorithm: Example (phase 3)



- 20
- 17

Figure: Fig. by courtesy of Mitsou Valia

HS algorithm: Example (phase 4)



- 20

Figure: Fig. by courtesy of Mitsou Valia

HS algorithm: Example (phase 5)

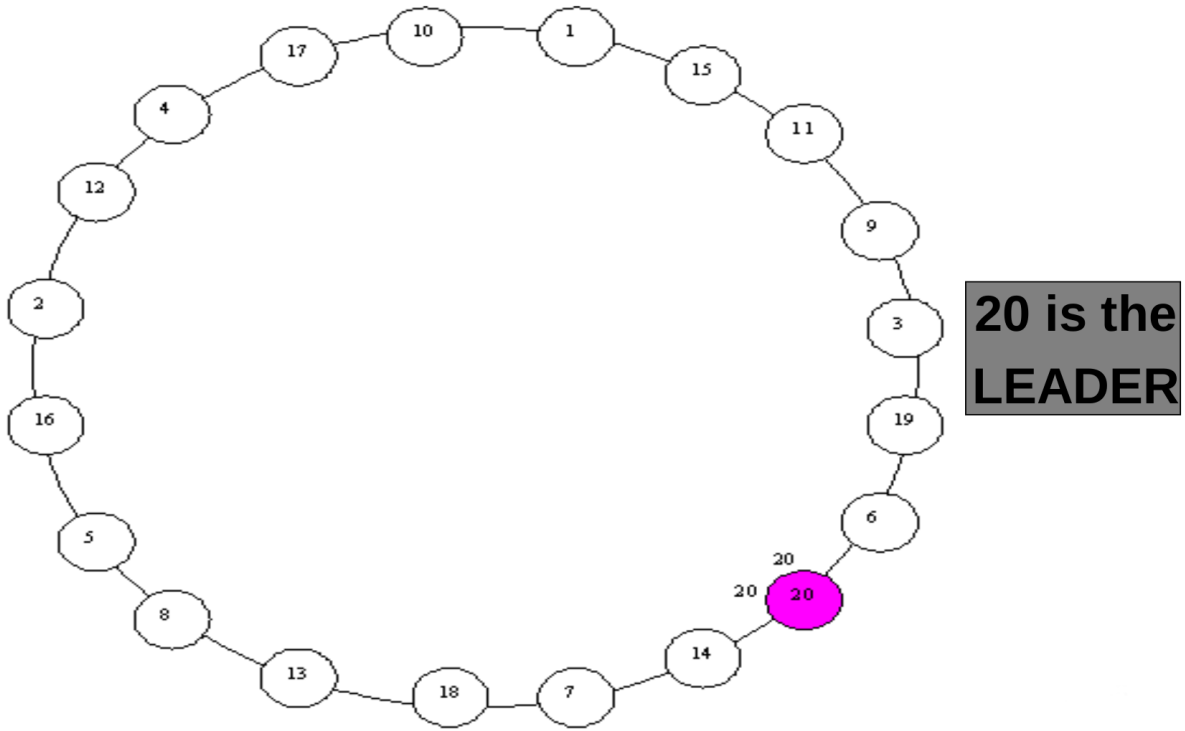


Figure: Fig. by courtesy of Mitsou Valia