Leader election

- message passing
- asynchronous

(Leader election)

- motivation
- who starts?
- Leader election, maximum finding, spanning tree
- Unidirectional ring
- Bidirectional rings
- Complete networks
- General networks

**Unidirectional ring phases, unique execution**

**Bidirectional ring sense of direction**
(Leader election)

Bidirectional ring

Sense of Direction:
For each process \( p \) in a bidirectional ring, its left and right neighbors are termed \( \text{left}(p) \) and \( \text{right}(p) \) respectively.

If \( \text{right}(\text{left}(p)) = p \) for every \( p \), then there is a sense of direction (otherwise – no sense of direction)

LeLann’s algorithm

\[
\begin{align*}
\text{state} & := \text{candidate}; \\
\text{send} & (\text{my_id}); \text{receive} (\text{nid}); \\
\text{while} & \text{ nid \neq my_id} \\
\text{do} & \text{if} \text{ nid > my_id} \\
\text{then} & \text{state} := \text{no_leader}; \\
\text{send} & (\text{nid}); \text{receive} (\text{nid}); \\
\text{od}; \\
\text{if} & \text{state} = \text{candidate} \text{ then } \text{state} := \text{leader}.
\end{align*}
\]
Algorithm: LeLann’s algorithm

- Messages: 64
- Time: 8

Theorem: LeLann’s algorithm terminates, and exactly one processor is in state = leader.

Message complexity: $O(n^2)$
(worst and average)
Time complexity: $O(n)$

Chang and Roberts algorithm:

```
state := candidate;
send (my_id); receive (nid);
while nid ≠ my_id
  do if nid > my_id
      then (state := no_leader;
            send (nid); receive (nid);
      od;
  if state = candidate then state := leader.
```
Theorem: Chang and Roberts’s algorithm terminates, and exactly one processor is in state=leader.

Message complexity: $O(n^2)$ (worst)

Time complexity: $O(n)$
Theorem: The average message complexity of Chang and Roberts’s algorithm is $O(n \log n)$.

Assume all rings equally probably (for the proof – assume ids are 1, 2, ..., n)

$P(i, k) = \frac{(i-1)_{k-1}}{(n-1)_{k-1}} \frac{n-i}{n-k}$

$P(i, k)$ – probability that id $i$ makes exactly $k$ steps

$n + \sum_{i=1}^{n-1} \sum_{k=1}^{i} k * P(i, k) = n + \sum_{k=1}^{n-1} \frac{n}{k+1} = n \left(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}\right) = 0.69n \log n + O(1)$
or: Consider all $n!$ rings
- Each id makes 1 step – $n!$ times
- Identity of $P_i$: makes 2nd step iff it is largest
  among $P_i, P_{i+1}$, which happens $\frac{n!}{2}$ times
- Identity of $P_i$: makes 3rd step iff it is largest among $P_i, P_{i+1}, P_{i+2}$, which happens $\frac{n!}{3}$ times,
- etc …

(Chang and Roberts’ algorithm)

Bidirectional rings

messages: ?
time: ?

Hirschberg and Sinclair’s algorithm

Phases 1, 2, …
- $active_k$ processors start phase $k$
- $active_k = n$
- $active_k \leq \left\lfloor \frac{active_k}{2} \right\rfloor$
- no. of phases $\leq \left\lfloor \log n \right\rfloor$
- messages $\leq 4n \left\lfloor \log n \right\rfloor$
- time = $O(n)$
Franklin's algorithm

messages: ?
time: ?

(Franklin's algorithm)

messages: \[ \Theta(n) \]

(Franklin's algorithm)

no. of phases \[ \leq \lceil \log n \rceil \]
messages \[ \leq 2n \lceil \log n \rceil \]
time \[ = O(n) \]

Exercise: what is the expected number of active processors after the first phase?
This algorithm is a modification of Franklin’s algorithm for unidirectional ring. The basic idea is, during a phase, each active process receives the temporary identifier of its nearest active neighbor and that neighbor’s nearest active neighbor’s temporary identifier, then applies Franklin’s strategy.

Each node maintains four variables:

- \( \text{state} \in \{\text{candidate, relay, leader}\} \)
- \( \text{tid} \) – temporary identity
- \( \text{ntid} \) – first id received
- \( \text{nntid} \) – second id received

```
state := candidate;
tid := id;
while state = relay do
    begin [start phase]
        send(tid); receive(ntid);
        if ntid = id then state := leader;
        if tid > ntid then send(tid);
        else send(ntid);
        receive(nntid);
        if nntid = id then state := leader;
        if ntid \geq \max(tid, nntid) then tid := ntid
        else state := relay;
    end;
    (now state = relay)
```
while state ≠ leader do
  begin
    receive(tid);
    if tid > id then state := leader;
    send(tid);
  end

(now state = relay)

[Diagram showing phase 1a and phase 1b with nodes and transitions]
\begin{align*}
\text{candidate:} & \quad \text{if } ntid \geq \max(tid, nntid) \\
& \quad \text{then } tid := ntid \\
& \quad \text{else state := relay;}
\end{align*}

\[ \text{send}(tid); \]
\[ \text{receive}(ntid); \]

\begin{align*}
\text{candidate:} & \quad \text{if } tid > ntid \\
& \quad \text{then send}(tid); \\
& \quad \text{else send}(nntid); \\
& \quad \text{receive}(ntid); \\
& \quad \text{relay};
\end{align*}
if \( ntid \geq \max(tid, nntid) \) then \( tid := ntid \) else state := relay;

\[\begin{align*}
\text{candidate:} & \\
\text{[start phase]}: & \\
\text{send}(tid); & \\
\text{receive}(ntid); & \\
\text{relay}. & 
\end{align*}\]

Exercises: 1. why send \( \max(tid, ntid) \)?
2. what happens if \( n=2 \)?
3. what happens if \( n=1 \)?
Theorem: Peterson’s 1st algorithm always determines a unique processor – the one holding the largest identity – as a leader.

Message complexity ≤ 2n log n
Time complexity ≤ 2n – 1

Exercise: show examples for worst cases and for best cases in terms of time and in terms of messages.
Peterson’s 2nd Algorithm
improvement of Peterson’s 1st algorithm

Instead of comparing its id with both neighbors in the same time, a process first compares itself with its left neighbor, then its right neighbor.

(Peterson’s 2nd Algorithms)

Each node maintains four variables:

\begin{itemize}
\item \textbf{state} \in \{candidate, relay, leader\}
\item \textbf{tid} – temporary identity
\item \textbf{ntid} – id received
\end{itemize}

\begin{verbatim}
state := candidate;
tid := id;
while state \neq relay do
begin [compare to left, odd phase]
    send(tid); receive(ntid);
    if ntid = id then state := leader;
    if tid < ntid then state := relay;
end;
begin [compare to right, even phase]
    send(tid); receive(ntid);
    if ntid = id then state := leader;
    if tid > ntid then state := relay
        else tid := ntid;
end;
end;
\end{verbatim}

(now state = relay)
(now state = relay)
while state ≠ leader do
  begin
    receive(tid);
    if tid = id then state = leader;
    send(tid);
  end

phase 1a

phase 1b
Theorem: Peterson's 2nd algorithm always determines a unique processor as a leader. Message complexity ≤ 1.44n log n

Exercise: show an example where a processor whose id is not the largest is elected as a leader.

phases \( p, p-1, \ldots, 1 \) (last phase)

\( t_1 \) - no. of processors that remain candidates after phase \( k \)

\( t_k \) - no. of processors that start phase \( k-1 \).

\[
\begin{align*}
t_{p+1} &= n \\
t_1 &= 1 \\
t_2 &\geq 2
\end{align*}
\]
**Lemma:** $t_k \leq \text{no. of processors that became relay during phase } k+1$

**Proof:**
We show that for each processor that remained active after phase $k$ there is a processor that became relay during phase $k+1$ (the previous phase).

### Case a: $k$ is odd

- Beginning of phase $k+1$
- End of phase $k+1 = \text{beginning of phase } k$
- End of phase $k$

---

### Case a: $k$ is odd

- $P$ survived phase $k$:
- End of phase $k+1 = \text{beginning of phase } k$
- End of phase $k$

Hence: $P \cdot q$
Case a: $k$ is odd

If in the beginning of phase $k+1$ all of these were already relays ...

end of phase $k+1 = \text{beginning of phase } k$

then $p$ would have become a relay, contradiction.

Lemma: $t_k \leq \text{no. of processors that became relay during phase } k+1$

Corollary: $t_k \leq t_{k-2} - t_{k-1}$

$t_k + t_{k-1} \leq t_{k-2}$

$t_k \geq \text{Fibonacci}_{k+1} = \frac{1}{\sqrt{5}} \left( \frac{1+\sqrt{5}}{2} \right)^{k+1} + O(1)$

$n = t_{p-1} \geq \text{Fibonacci}_{p-2} = \frac{1}{\sqrt{5}} \left( \frac{1+\sqrt{5}}{2} \right)^{p-2} + O(1)$

\[ \text{no. of phases } = p \leq 1.44... \log n \]

message complexity $\leq np \leq 1.44... n \log n$
<table>
<thead>
<tr>
<th>References</th>
</tr>
</thead>
</table>
| **E. Chang and R. Roberts,**  
*An improved algorithm for decentralized extrema-finding in circular configurations of processes,*  
*Communications of the ACM*, 22, 5, 1979, pp. 281-283. |

<table>
<thead>
<tr>
<th>References</th>
</tr>
</thead>
</table>
| **D. Dolev, M. Klawe and M. Rodeh,**  
*An O(n log n) unidirectional distributed algorithm for extrema finding in a circle,*  

<table>
<thead>
<tr>
<th>References</th>
</tr>
</thead>
</table>
| **W. R. Franklin,**  
*On an improved algorithm for decentralized extrema finding in circular configurations of processors,*  
<table>
<thead>
<tr>
<th>References</th>
</tr>
</thead>
</table>
References

N. Santoro,
Sense of direction, topological awareness and communication complexity,
SIGACT News, 16, 2, Summer 1984, pp. 50-56.