Modern Distributed Computing

Theory and Applications

Ioannis Chatzigiannakis

Sapienza University of Rome

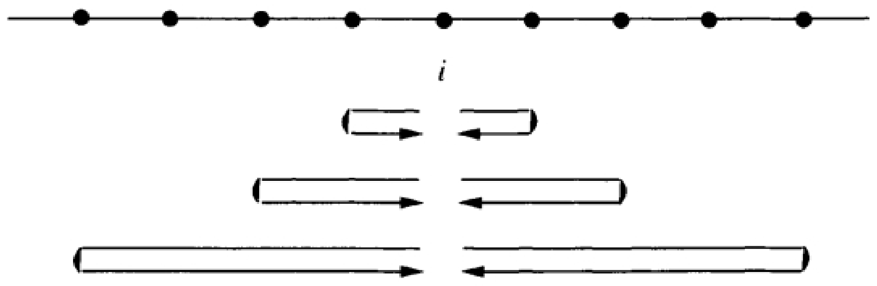
Lecture 3
Tuesday, March 19, 2013

Part 1: Static Synchronous Networks

Hirschberg's and Sinclair's Algorithm

Each process i operates in phases $(0, 1, 2, \ldots)$. In each phase l, process u sends out "tokens" containing its UID u_i. These are intended to travel distance 2^l, then return back to i. If both tokens make it back, i continues with the next phase. While a u_i token is proceeding in the outbound direction, each other process j on u_i's path compares u_i with UID u_j. If $u_i < u_j$, then j discards the token, if $u_i > u_j$, then j relays u_i. If $u_i = u_j$, then it means that process j has received its own UID and elects itself as the leader. All processes always relay all tokens in the inbound direction.

Trajectories of successive tokens of process i



Communication Complexity

Theorem (3.1)

Algorithm HS achieves a $\mathcal{O}(n \log n)$ message complexity.

Proof: Every process sends out a token in phase 0: a total of $4n$ messages are sent in both directions. For phase $l > 0$, a process sends a token in phase l ecatly if it receives both its phase $l-1$ tokens back. This is exactly if it has not been "defeated" by another process within distance $2^{l-1}+1$ in either direction along the ring.

Communication Complexity

Theorem (3.1)

Algorithm HS achieves a $\mathcal{O}(n \log n)$ message complexity.

Proof: Thus within any group of $2^{l-1}+1$ consecutive processes, at most one goes on to the next phase. Therefore at most

$$\left\lfloor \frac{n}{2^{l-1}+1} \right\rfloor$$

will be active in phase l.
Thus in phase l we have

$$4\left(\left\lfloor \frac{n}{2^{l-1}+1} \right\rfloor \times 2^l\right) \leq 8n$$

message exchanges.

Communication Complexity

Theorem (3.1)

Algorithm HS achieves a $\mathcal{O}(n \log n)$ message complexity.

Proof: The maximum number of phases is $1 + \lceil \log n \rceil$, thus the total number of messages is smaller than $8n(1 + \lceil \log n \rceil)$. ∎

Time Complexity

Theorem (3.2)

Algorithm HS requires $\mathcal{O}(n)$ rounds.

Proof: For each phase l a total of 2^{l+1} rounds are required so that the tokens go out and return. The final phase takes time n – it is an incomplete phase, with tokens only traveling outbound. Thus the total number of messages required are:

$$\sum_{l=0}^{\lceil \log n \rceil - 1} 2^{l+1} + n = 2^{\lceil \log n \rceil + 1} - 2 + n < 5n$$

∎

Breaking Symmetry using Non-determinism

- We re-examine the case where processes do not have UID.
- Ring Networks are symmetric - thus we need to weaken our model: we allow processes to use randomness.
- Itai & Rodeh solve the problem
  - Use the LCR algorithm as the basis.
  - Important difference: processes do not have UID.
- Each process chooses a random ID from a set $\{1, \ldots, k\}$
- We try to identify the process with the highest UID
  - If more than one process is identified, all processes with the highest UID re-execute the algorithm.

Avoiding a Deadlock

- No process should be allowed to unilaterally stop.
- We have to make sure that at least one process is still active, trying to win the election.
- If the algorithm does not guarantee this
  - It is possible to end up in a configuration where no process wishes to get elected.
  - The algorithm enters a deadlock.
- The Itai & Rodeh algorithms deals with this issue as follows:
  - In each phase, a processes can become inactive if and only if it identifies at least one other process with higher UID.
  - Thus, the are aware of at least one other process that may become a leader.
  - It is not possible for all processes to step-down all together.

Itai's and Rodeh's Algorithms

All processes maintain a variable phase$=0$. In each phase they pick uniformly randomly a UID from the set $\{1, \ldots, k\}$ and send a message to their clockwise neighbor of the form (phase,UID,counter,unique), where counter$=0$ and unique$=$`true`. When i receives a message from j, they compare its u_i with u_j. If $u_i > u_j$, i discards the message. If $u_i < u_j$, i forwards the message to their clockwise neighbor. If $u_i = u_j$, i checks,

- if $counter < n$ then it sets unique$=$`false` and forwards the message.
- If $counter \geq n$ and $unique =$ `false`, i moves to the next phase and repeat the algorithm.
- If $counter \geq n$ and $unique =$ `true`, i enters the state "elected".

Properties of Algorithm IR

- The algorithm evolves in phases
  - Each phase takes $\mathcal{O}(n)$ rounds.
  - Each phase, processes exchange $\mathcal{O}(n \log n)$ messages.
  - The required number of phases until a leader is elected is $\frac{en}{n-1}$.
    - A constant number of phases — $\mathcal{O}(1)$.
- Time complexity is $\mathcal{O}(n)$.
- Communication complexity is $\mathcal{O}(n \log n)$.

Communication Complexity

- Let's examine a single phase of the algorithm.
- For simplicity, without loss of generality, let $k = n$.
- Let $p(i, d)$ the probability that the message of process with UID i travels distance d

$$p(i,d) = \begin{cases} p_i^{n-1} & \text{if } d = n \\ p_i^{d-1}(1-p_i) & \text{otherwise} \end{cases}$$

- We wish to evaluate the average distance D that each message travels
    - This is related on the UID i
    - If $i = n$ then $\overline{D} = n$
    - otherwise, if $i < n$ then we compute it as follows

Communication Complexity

$$\begin{aligned} \overline{D}(i) &= \sum_{i=1}^{n} d \cdot p(i,d) \\ &= n \cdot p_i^{n-1} + (1-p_i)\sum_{i=1}^{n-1} d \cdot p_i^{d-1} \\ &= \frac{1 - np_i^{n-1} + (n-1)p_i^n}{1-p_i} + np_i^{n-1} \\ &= \frac{1-p_i^n}{1-p_i}, \quad 1 \le i \le n \end{aligned}$$

- Each process has probability $\frac{1}{n}$ to select UID i.
- Process with UID i has probability $\frac{i}{n}$ to be elected.

Communication Complexity

Thus the average number of messages N that are exchanged during each phase is bound by :

$$\begin{aligned} N &\le \sum_{i=1}^{n} \overline{D}(i) = n + \sum_{i=1}^{n-1} \frac{1-(\frac{i}{n})^n}{1-\frac{i}{n}} \\ &\le n + \sum_{i=1}^{n-1} \frac{1}{1-\frac{i}{n}} = n + n\sum_{i=1}^{n-1} \frac{1}{n-1} \\ &= n + n \cdot \mathcal{O}(\log n) \end{aligned}$$

Therefore the average number of messages for each phase is $\mathcal{O}(n \log n)$

TimeSlice Algorithm

Computation proceeds in phases $(0, 1, 2, \ldots)$, where each phases consists of n consecutive rounds. Each phase is devoted to the possible circulation, all the way around the ring, of a token carrying a particular UID. That is, in phase v (which consists of rounds $(v-1)n + 1, \ldots, uv)$ only a token carrying the UID v is permitted to circulate.
If a process v exists, and round $(v-1)n + 1$ is reached without v having previously received any non-null message, then v elects itself the leader and sends a token carrying its UID around the ring. As this token travels, all the other processes not that they have received it, which prevents them from electing themselves as leader or initiating the sending of a token at any later phase.

- Very Strong assumption: each process knows the ring size n.
- Non-uniform algorithm.

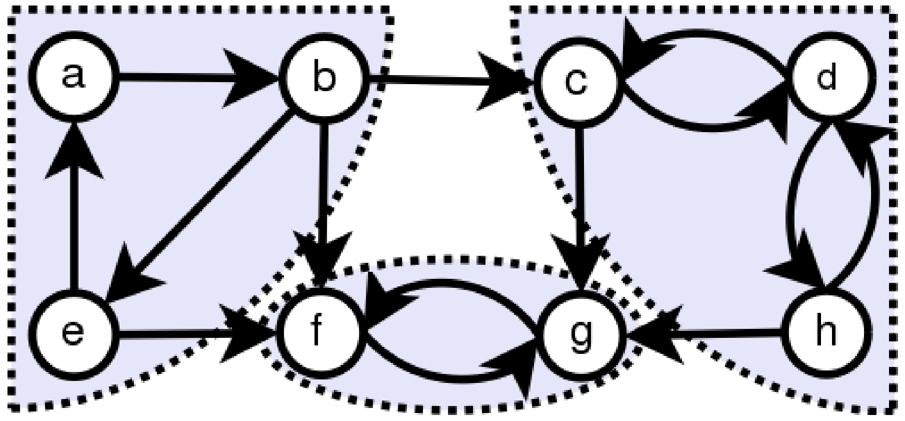
Properties of Algorithm TimeSlice

- The process with the lowest UID is elected (i_{min}).
- No process other than i_{min} enters state "elected".
- Process i_{min} elects itself leader at round $(i_{min} - 1)n + 1$.
- Until round $(i_{min} - 1)n + 1$ and after round $i_{min}n$ no messages are exchanged.
- Total number of messages n.
- Message complexity $\mathcal{O}(n)$.
- Time complexity cannot be expressed in a closed form. It requires ni_{min} rounds.

Seminal Papers

1. Daniel S. Hirschberg, J. B. Sinclair: Decentralized Extrema-Finding in Circular Configurations of Processors. Commun. ACM 23(11): 627-628 (1980)
2. Alon Itai, Michael Rodeh: Symmetry breaking in distributed networks. Inf. Comput. 88(1): 60-87 (1990)

Strongly Connected General Network



- The graph is strongly connected.
- Each process has UID – is not aware of the UID of the other processes.

Algorithm FloodMax

Every process maintains a record of the maximum UID it has seen so far (initially its own). At each round, each process propagates this maximum on all of its outgoing edges. After *diam*(G) rounds, if the maximum value seen is the process's own UID, the process elects itself the leader; otherwise, it is a non-leader.

- Processes are not aware of the total number of processes (n).
- Processes are aware of the network diameter — $\delta = diam(G)$
- Comparison-based algorithm.

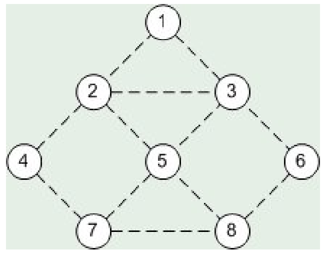
Pseudo-code for FloodMax

```
#DEFINE UID = <...>;
#DEFINE $\delta$ = <...>;
void main() {
   bool leader = false;
   int max_id = UID;
   for (int i = 0 ; i < $\delta$; i++ ) {
      sendMessage(max_id);
      while (int new_msg = readMessage()) {
         if (new_msg > max_id)
            max_id = new_msg;
      }
   }
   if (max_id == UID)
     leader = true;
}
```

Example of Execution for FloodMax Algorithm

- Let a synchronous distributed system of $n = 8$ processes..
  - General network where $\delta = 3$
  - Processes are index $1 \dots 8$

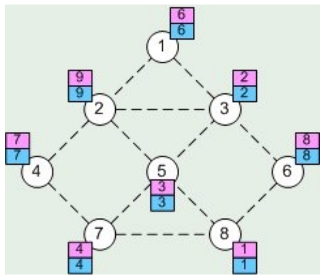
General Network



Example of Execution for FloodMax Algorithm

- Let a synchronous distributed system of $n = 8$ processes..
  - General network where $\delta = 3$
  - Processes are index $1 \dots 8$
- The processes have UID.
  - Not aware of the UID of the other processes.

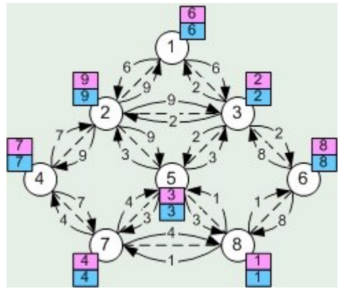
General Network



Example of Execution for FloodMax Algorithm

- Let a synchronous distributed system of $n = 8$ processes..
  - General network where $\delta = 3$
  - Processes are index $1 \dots 8$
- The processes have UID.
  - Not aware of the UID of the other processes.
- First Round

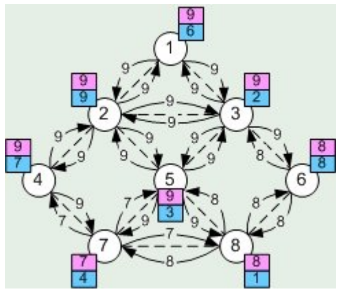
General Network



Example of Execution for FloodMax Algorithm

- Let a synchronous distributed system of $n = 8$ processes..
  - General network where $\delta = 3$
  - Processes are index $1 \dots 8$
- The processes have UID.
  - Not aware of the UID of the other processes.
- First Round
- Second Round

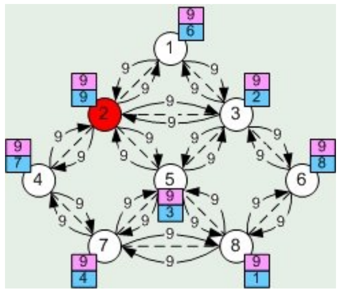
General Network



Example of Execution for FloodMax Algorithm

- Let a synchronous distributed system of $n = 8$ processes..
  - General network where $\delta = 3$
  - Processes are index $1 \dots 8$
- The processes have UID.
  - Not aware of the UID of the other processes.
- First Round
- Second Round
- Leader Election

General Network



Properties of FloodMax Algorithm

Let n processes and m channels, where the process with the highest UID is i_{max}.

- Process i_{max} is elected leader at the end of round δ.
- No other process is in state "elected".
- Time complexity is $\mathcal{O}\left(diam(G)\right)$.
- Message complexity $\mathcal{O}\left(diam(G) \cdot m\right)$.

Proof of Correctness

Theorem (3.3)

In the FloodMax algorithm, process i_{max} is elected at the end of round δ.

Proof: It is enough to prove that after δ rounds, $\text{leader}_{i_{max}} =$ `true`.
The key to the proof is the fact that after r rounds, the maximum UID has reached every process that is within distance r of i_{max}.
In view of the definition of the diameter of the graph, this implies that every process has the maximum UID by the end of δ rounds.
■

Directed spanning tree

A directed spanning tree of a directed graph $G = (V, E)$ is a rooted tree that consists entirely of directed edges in E, all edges directed from parents to children in the tree, and that contains every vertex of G.

Breadth-First directed spanning tree

A directed spanning tree of G with root i is breadth-first provided that each node at distance d from i in G appears at depth d in the tree.

- Every strongly connected digraph has a breadth-first directed spanning tree.
- Constructing a Breadth-First directed spanning tree is useful for efficient collection of information.

SynchBFS Algorithm

At any point during execution, there is some set of processes that is "marked", initially just i_0. Process i_0 sends out a search message at round 1, to all of its outgoing neighbors. At any round, if an unmarked process receives a search message, it marks itself and chooses one of the processes from which the search has arrived as its parent. At the first round after a process gets marked, it sends a search message to all of its outgoing neighbors.

- Processes are not aware of the total number of processes (n)
- All processes have UIDs.

Example of Execution for SyncBFS Algorithm

Initial Network

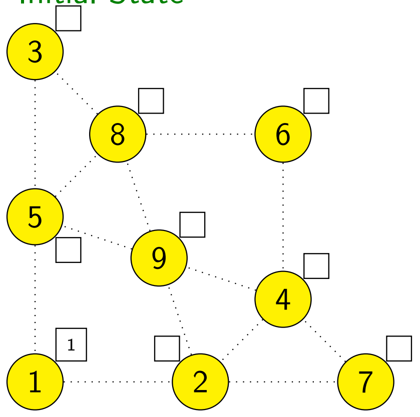
The network contains 9 processes, 14 channels

Process **1** initiates the execution.

Process **1** is marked.

All other processes are not marked.

Initial State

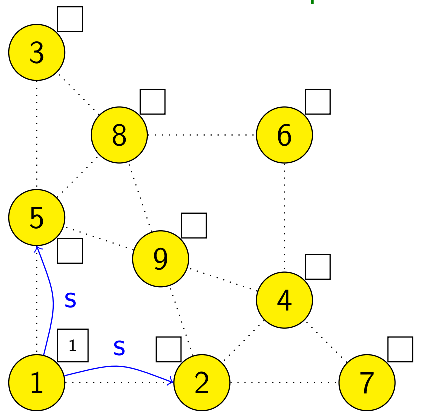


Example of Execution for SyncBFS Algorithm

1st Round – 1st Step

Process **1** sends search to its neighbors.

1st Round – 1st Step



Example of Execution for SyncBFS Algorithm

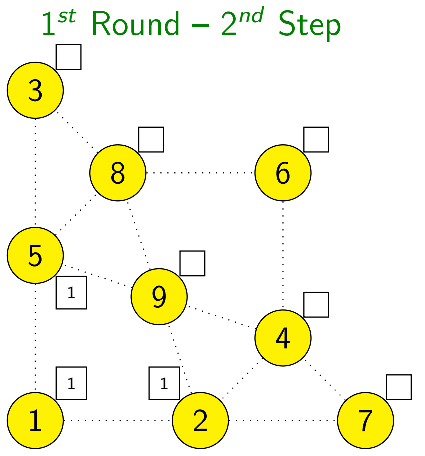
1st Round – 1st Step

Process **1** sends search to its neighbors.

1st Round – 2nd Step

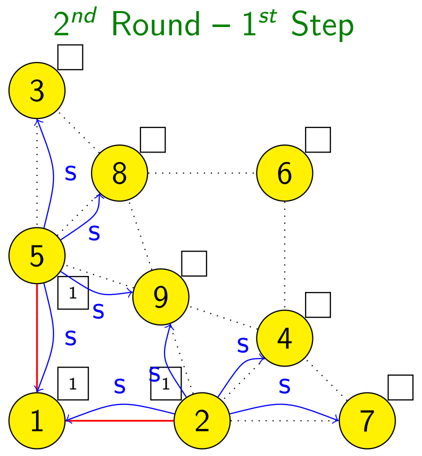
Processes **2, 5** are marked.

Processes **2, 5** select **1** as parent process.



Example of Execution for SyncBFS Algorithm

2nd Round – 1st Step Processes **2, 5** send search to all neighbors.



Example of Execution for SyncBFS Algorithm

2nd Round – 1st Step Processes **2, 5** send search to all neighbors.

2nd Round – 2nd Step

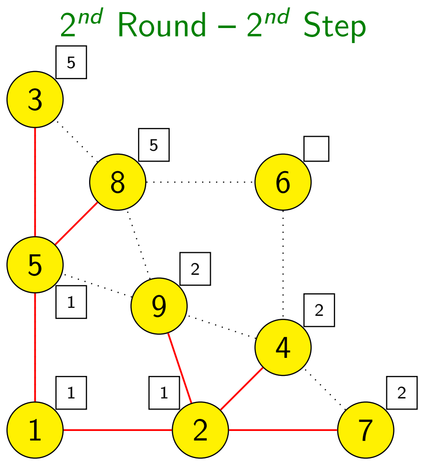
Process **1** ignores all search messages received.

Processes **3, 4, 7, 8, 9** are marked.

Processes **3, 8** set process **5** as parent process.

Processes **4, 7** set process **2** as parent process.

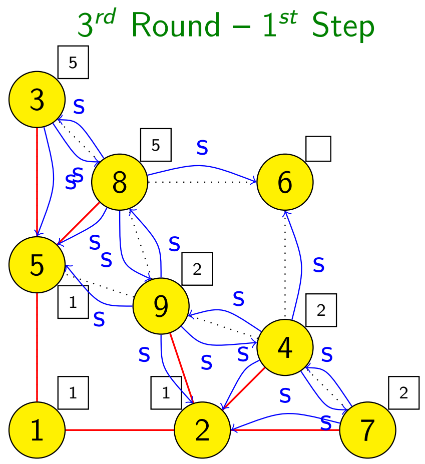
Process **9** chooses (randomly) process **2** as parent process.



Example of Execution for SyncBFS Algorithm

3rd Round – 1st Step

Processes **3, 4, 7, 8, 9** send search to all neighbors.



Example of Execution for SyncBFS Algorithm

3rd Round – 1st Step

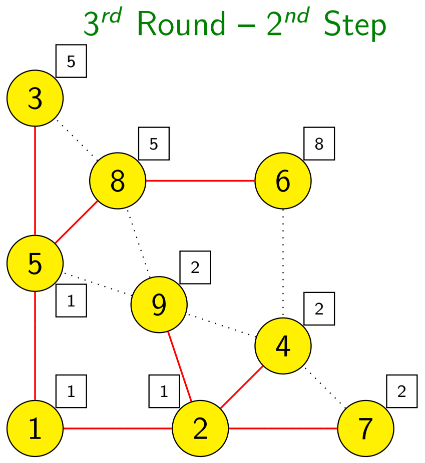
Processes **3, 4, 7, 8, 9** send search to all neighbors.

3rd Round – 2nd Step

Processes **2, 3, 4, 5, 7, 8, 9** ignore the search messages received.

Process **6** is marked.

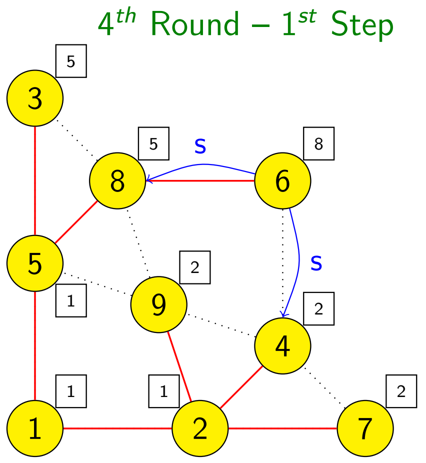
Process **6** chooses (randomly) process **8** as parent process.



Example of Execution for SyncBFS Algorithm

4th Round – 1st Step

Process **6** sends search to all neighbors.



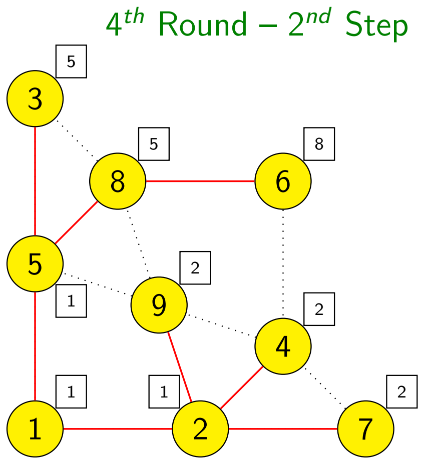
Example of Execution for SyncBFS Algorithm

4th Round – 1st Step

Process **6** sends search to all neighbors.

4th Round – 2nd Step

Processes **4, 8** ignore the search messages received.



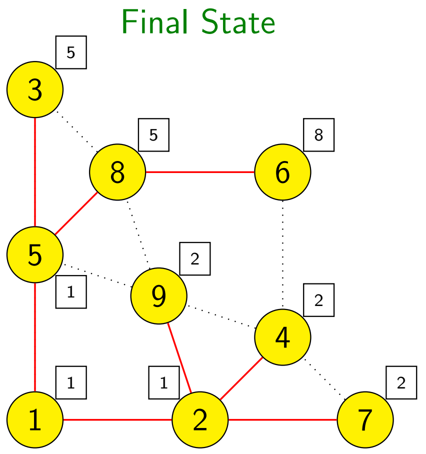
Example of Execution for SyncBFS Algorithm

Final Step

Breadth-first directed spanning tree is constructed.

Total number of rounds: 4

Total number of messages: 28



Properties of SynchBFS Algorithm

- Algorithm SynchBFS constructs a breadth-first directed spanning tree.
- The structure is not stored in some "centralized" process.
- The tree pointers are "distributed" across the network.
- The time complexity is $\mathcal{O}(diam(G))$
  - In practice it is the maximum distance from u_0
  - In the example, the diameter is 4 – maximum distance from u_0 is 3.
- Message complexity: $\mathcal{O}(m)$

Improving Message Complexity

We can reduce the total number of messages exchanged by the algorithm as follows:

- The processes can identify the channel from which they received a search message.
- The processes do not send search towards those channels.

In the example, messages are reduced to 10 (i.e., 18 less).

Message Broadcast

The algorithm can easily be augmented to implement message broadcast.

- A process has a message m that it wants to communicate to all of the processes in the network.
- It initiates an execution of SynchBFS with itself as the root.
- Piggybacks message m on the search message in round 1.
- Other processes continue to piggyback m on all their search messages as well.
- Since the tree eventually spans all the nodes, message m is eventually delivered to all the processes.

Convergecast

Message convergecast is the inverse of a broadcast in a message-passing system (see Flooding) – instead of a message propagating down from a single root to all nodes, data is collected from outlying nodes through a direct spanning tree to the root. Typically some function is applied to the incoming data at each node to summarize it, with the goal being that eventually the root obtains this function of all the data in the entire system. (Examples would be counting all the nodes or taking an average of input values at all the nodes.)

Child Pointers

In order to use SyncBFS for message broadcast it is required that each process learn not only who its parent in the tree is, but also who all of its children are.

- If bidirectional communication is allowed between all pairs of neighbors, i.e., the network is undirected, this is simple.
- Each unmarked process, upon receiving the first search message, it sends a message parent to the process from which the message was received.

When SynchBFS' terminates, all processes are aware of their “children” processes.

The modified algorithm SynchBFS' requires $diam(G) + 2$ rounds and uses $m + n - 1$ messages.

Termination

How can the source process i_0 tell when the construction of the tree has completed ?

- The diameter of the network is know known, neither the total number of processes n.

If each search message is answered with either a parent or non-parent message, then after any process has received responses from all of its search messages, it knows who all its children are and knows that they have all been marked.

Termination

How can the source process i_0 tell when the construction of the tree has completed ?

Starting from the leaves of the BFS tree, notification of completion can be “fanned in” to the source:

- each process can send notification of completion to its parent in the tree as soon as
  1. it has received responses for all its search messages (so that it knows who its children are and knows that they have been marked)
  2. it has received notification of completion from all its children.

This type of procedure is called a convergecast.

Example of Execution for SyncBFS$_c$ Algorithm

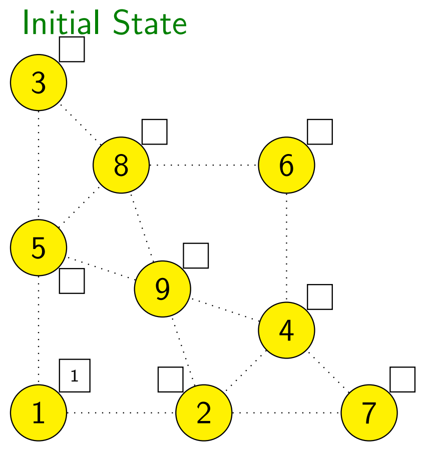
Initial Network

The network contains 9 processes, 14 channels

Process **1** initiates the execution.

Process **1** is marked.

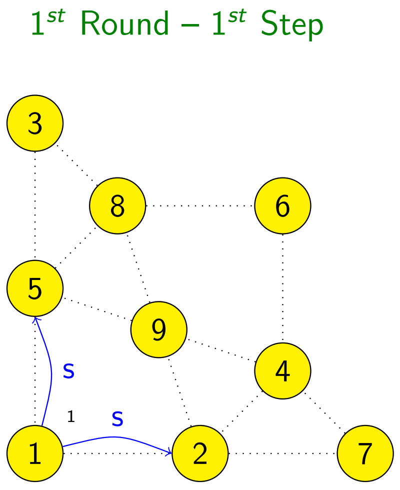
All other processes are not marked.



Example of Execution for SyncBFS$_c$ Algorithm

1st Round – 1st Step

Process **1** sends search to its neighbors.



Example of Execution for SyncBFS$_c$ Algorithm

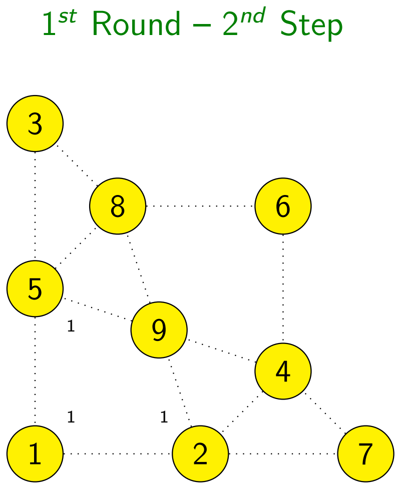
1st Round – 1st Step

Process **1** sends search to its neighbors.

1st Round – 2nd Step

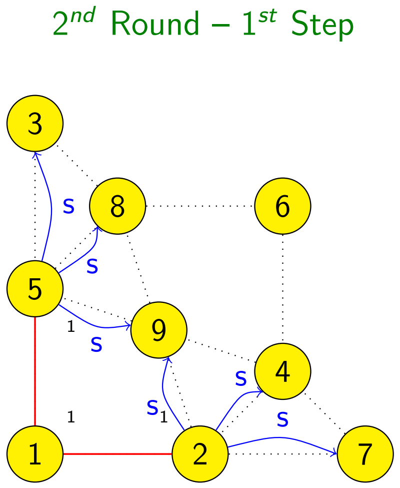
Processes **2, 5** are marked.

Processes **2, 5** select **1** as parent process.



Example of Execution for SyncBFS$_c$ Algorithm

2nd Round – 1st Step Processes **2, 5** send search to all neighbors.



Example of Execution for SyncBFS$_c$ Algorithm

2nd Round – 1st Step Processes **2, 5** send search to all neighbors.

2nd Round – 2nd Step

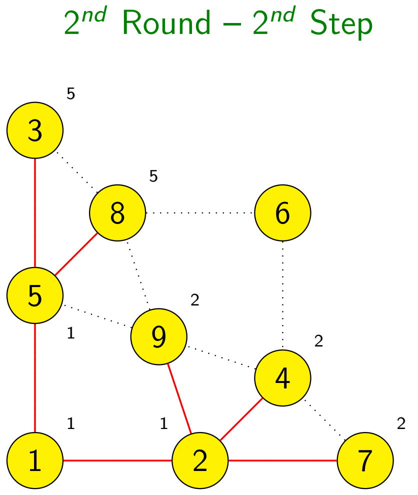
Process **1** ignores all search messages received.

Processes **3, 4, 7, 8, 9** are marked.

Processes **3, 8** set process **5** as parent process.

Processes **4, 7** set process **2** as parent process.

Process **9** chooses (randomly) process **2** as parent process.

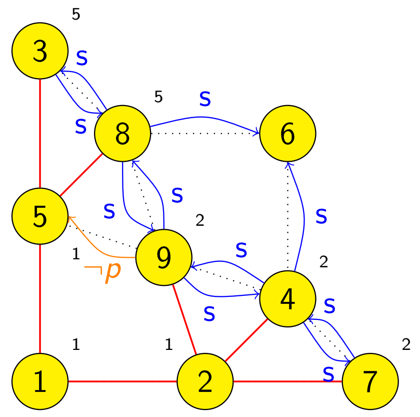


Example of Execution for SyncBFS$_c$ Algorithm

3rd Round – 1st Step

Processes **3, 4, 7, 8, 9** send search to all neighbors.

3rd Round – 1st Step



Example of Execution for SyncBFS$_c$ Algorithm

3rd Round – 1st Step

Processes **3, 4, 7, 8, 9** send search to all neighbors.

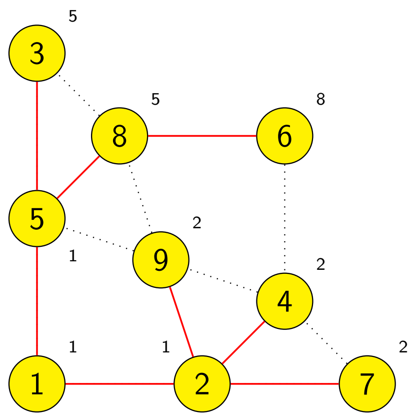
3rd Round – 2nd Step

Processes **2, 3, 4, 5, 7, 8, 9** ignore the search messages received.

Process **6** is marked.

Process **6** chooses (randomly) process **8** as parent process.

3rd Round – 2nd Step



Example of Execution for SyncBFS$_c$ Algorithm

4th Round – 1st Step

Process **3** sends non-parent to **8**

Process **8** sends non-parent to **3**

Process **8** sends non-parent to **9**

Process **9** sends non-parent to **4**

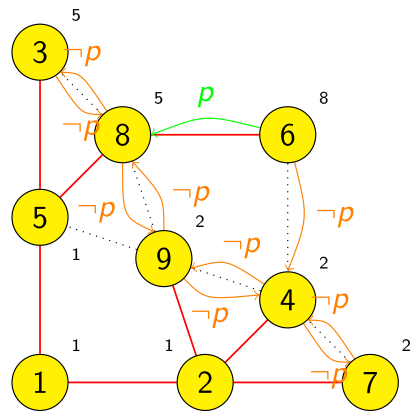
Process **4** sends non-parent to **7**

Process **7** sends non-parent to **4**

Process **6** sends non-parent to **4**

Process **6** st'elnei m'hnuma "gon'eas" sthn **8**

3rd Round – 1st Step



Example of Execution for SyncBFS$_c$ Algorithm

4th Round – 1st Step

Process **3** sends non-parent to **8**

Process **8** sends non-parent to **3**

Process **8** sends non-parent to **9**

Process **9** sends non-parent to **4**

Process **4** sends non-parent to **7**

Process **7** sends non-parent to **4**

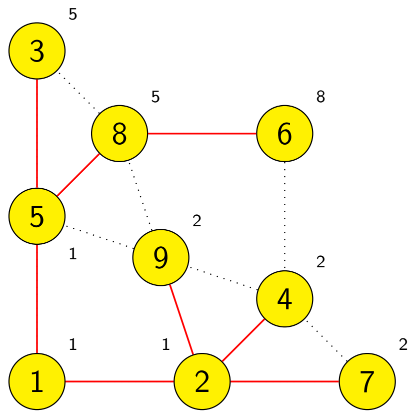
Process **6** sends non-parent to **4**

Process **6** st'elnei m'hnuma "gon'eas" sthn **8**

4th Round – 2nd Step

Process **8** detects the completion of the sub-tree of **6**

3rd Round – 2nd Step

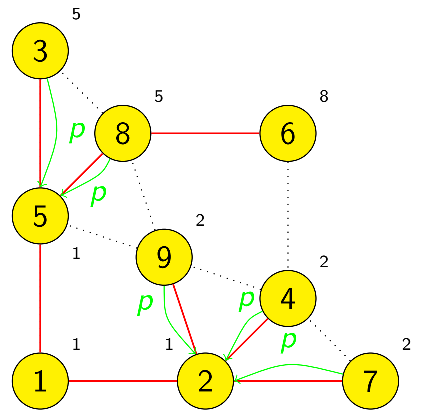


Example of Execution for SyncBFS$_c$ Algorithm

5th Round – 1st Step Processes **3, 8** send parent message to **5**

Processes **4, 7, 9** send parent message to **2**

5th Round – 1st Step



Example of Execution for SyncBFS$_c$ Algorithm

5th Round – 1st Step Processes **3, 8** send parent message to **5**

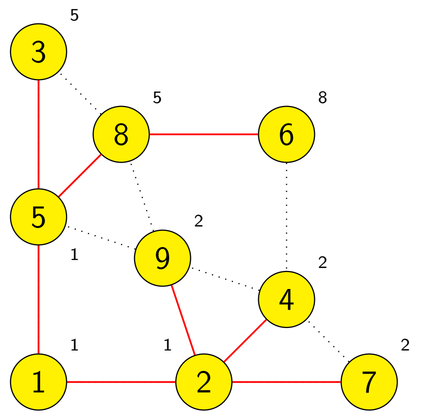
Processes **4, 7, 9** send parent message to **2**

5th Round – 2nd Step

Process **5** detects the completion of the sub-trees of **3, 8**

Process **2** detects the completion of the sub-trees of **4, 7, 9**

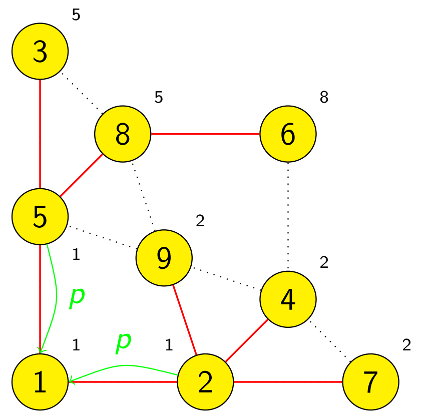
5th Round – 2nd Step



Example of Execution for SyncBFS$_c$ Algorithm

6th Round – 1st Step Processes **2, 5** send parent message to **1**

6th Round – 1st Step



Example of Execution for SyncBFS$_c$ Algorithm

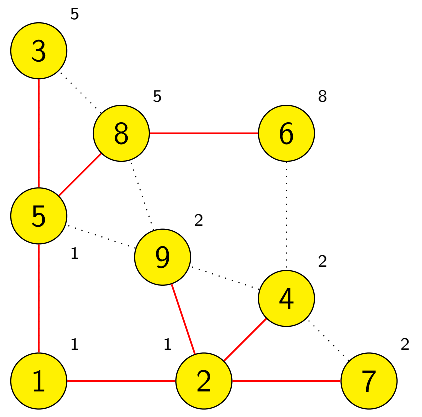
6th Round – 1st Step Processes **2, 5** send parent message to **1**

6th Round – 2nd Step

Process **1** detects the completion of the sub-trees of **2, 5**

Process **1** terminates

6th Round – 2nd Step



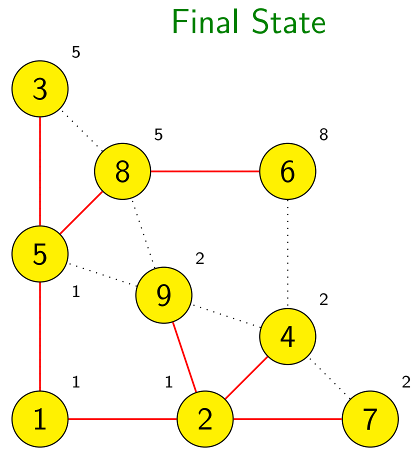
Example of Execution for SyncBFS$_c$ Algorithm

Final Step

Breadth-first directed spanning tree is constructed.

Total number of rounds: 6

Total number of messages: 36



Open Problem # 3

Design a synchronous distributed algorithm for general networks that computes the diameter of the network.

Open Problem # 3

Design a synchronous distributed algorithm for general networks that computes the diameter of the network.

Open Problem # 4

Prove the correctness of SyncBFS.

Open Problem # 3

Design a synchronous distributed algorithm for general networks that computes the diameter of the network.

Open Problem # 4

Prove the correctness of SyncBFS.

Open Problem # 5

Each process u receives an input value $i_u \in R$. Design a synchronous distributed algorithm for general networks that computes the average over all input values.

Open Problem # 3

Design a synchronous distributed algorithm for general networks that computes the diameter of the network.

Open Problem # 4

Prove the correctness of SyncBFS.

Open Problem # 5

Each process u receives an input value $i_u \in R$. Design a synchronous distributed algorithm for general networks that computes the average over all input values.

Open Problem # 6

Each process u receives an input value $i_u \in R$. Design a synchronous distributed algorithm for general networks that computes the median of all input numbers.