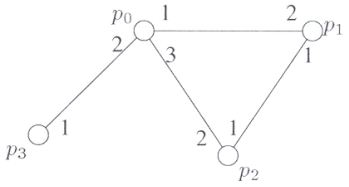
PART II
Algorithms for Message-Passing Systems

Section 7
Introduction – Model – Basic Graph Algorithms

Formal Model for Message-Passing Systems

- There are n processes in the system: p_0, .., p_{n-1}
- Each process is modeled as a state machine.
- The state of each process is comprised by its local variables and a set of arrays. For instance, for p_0, the state includes six arrays:
  - $inbuf_0[1]$, …, $inbuf_0[3]$: contain messages that have been sent to p_0 by p_1, p_2 and p_3, respectively, but p_0 has not yet processed.
  - $outbuf_0[1]$, …, $outbuf_0[3]$: messages that have been sent by p_0 to p_1, p_2, and p_3, respectively, but have not yet been delivered to them.



Formal Model of Message-Passing Systems

- The state of process p_i excluding the $outbuf_i[l]$ components, comprises the accessible state of p_i.
- Each process has an initial state in which all inbuf arrays are empty.
- At each step of a process, all messages stored in the inbuf arrays of the process are processed, the state of the process changes and a message to each other neighboring process can be sent.

Formal Model of Message-Passing Systems

- A configuration is a vector $C = (q_0, .., q_{n-1})$ where q_i represents the state of p_i.
  - The states of the outbuf variables in a configuration represent the messages that are in transit on the communication channels.
  - In an initial configuration all processes are in initial states.

Formal Model of Message-Passing Systems

- **Computation event, comp(i)**
  - Represents a computation step of process p_i in which p_i' s transition function is applied to its current accessible state.
- **Delivery Event, del(i,j,m)**
  - Represents the delivery of message m from processor p_i to processor p_j (i.e., message m is placed in one of the inbuf buffers of p_j)
- The behavior of a system over time is modeled as an **execution**, which is a sequence of configurations alternating with events.

Formal Model of Message-Passing Systems

- An execution must satisfy a variety of conditions.
  - Safety condition
    - Holds in every finite prefix of the execution (it states that nothing bad has happened yet)
  - Liveness condition
    - Holds a certain number of times (it states that eventually something good must happen)

Formal Model of Message-Passing Systems Complexity Measures

- The message complexity of an algorithm for either a synchronous or an asynchronous message-passing system is the maximum, over all executions of the algorithm, of the total number of messages sent.
- The time complexity of an algorithm for a *synchronous message-passing system* is the maximum number of rounds, in any execution of the algorithm, until the algorithm has terminated.

Formal Model of Message-Passing Systems Complexity Measures

Measuring the time complexity of asynchronous algorithms

- A timed execution is an execution that has a nonnegative real number associated with each event, which illustrates the time at which that event occurs.
- The times must start at 0, must be strictly increasing for each individual processor, and must increase without bound if the execution is infinite.
- We define the delay of a message to be the time that elapses between the computation event that sends the message and the computation event that processes the message.
- **Assumption**: The maximum message delay in any execution is one unit of time.
- The time complexity of an *asynchronous algorithm* is the maximum time until termination among all timed executions of the algorithm in which every message delay is at most one time unit.

Broadcast on a Spanning Tree

- A distinguished processor, p_r, has a message <M> it wishes to send to all other processors.
- Copies of the message are to be sent along a tree which is rooted at p_r, and spans all the processors in the network.
- The spanning tree is maintained in a distributed fashion:
  - Each processor has a distinguished channel that leads to its parent, as well as a set of channels that lead to its children.

Algorithm 1 Spanning tree broadcast algorithm.

Initially $\langle M \rangle$ is in transit from p_r to all its children in the spanning tree.

Code for p_r:
1: upon receiving no message: // first computation event by p_r
2: terminate

Code for p_i, $0 \leq i \leq n-1$, $i \neq r$:
3: upon receiving $\langle M \rangle$ from parent:
4: send $\langle M \rangle$ to all children
5: terminate

Broadcast on a Spanning Tree

State of process p_i, $i \in \{0, \ldots, n-1\}$

- a variable $parent_i$,which holds either a processor index or nil
- a variable $children_i$,which holds a set of processor indices
- a variable $terminated_i$,which indicates whether p_i is in a terminated state
- the inbuf and outbuf tables of p_i

Initial State

- all terminated variables are false.
- The inbuf tables are empty, for all processes.
- The outbuf tables are empty for all processes other than p_r; $outbuf_r[j]$ contains M for all $j \in children_r$.

Complexities?

- Communication Complexity?
- Time Complexity?

Broadcast on a spanning tree - Time Complexity

Synchronous System

- **Lemma**: In every execution of the broadcast algorithm in the synchronous model, every process at distance t from p_r in the spanning tree receives <M> in round t.
- **Proof**: By induction on the distance t of a process from p_r.
- t = 1. Each child of p_r receives <M> from p_r in the first round.
- Assume that every process at distance $t-1 \geq 1$ from p_r receives the message <M> in round t-1.
- Let p be any process in distance t from p_r. Let p' be the parent of p in the spanning tree. Since p' is at distance t-1 from p_r, by the induction hypothesis, p' receives <M> in round t-1. By the description of the algorithm, p receives <M> from p' in the next round.

Broadcast on a spanning tree - Time Complexity

Asynchronous System

- **Lemma**: In every execution of the broadcast algorithm in an asynchronous model, every process at distance t from p_r in the spanning tree receives <M> in time t.
- **Proof**: By induction on the distance t of a process from p_r.
- t = 1. From the description of the algorithm, <M> is initially in transit to each process p_i at distance 1 from p_r. By the definition of time complexity for the asynchronous model, p_i receives <M> by time 1.
- Assume that every process at distance t-1 ≥ 1 from p_r receives the message <M> by time t-1.
- Let p be any process in distance t from p_r. Let p' be the parent of p in the spanning tree. Since p' is at distance t-1 from p_r, by the induction hypothesis, p' receives <M> by time t-1. By the description of the algorithm, p receives <M> from p' by time t.

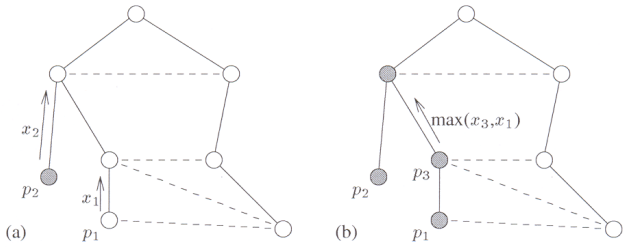
Broadcast on a spanning tree

- **Theorem 1**: There is a synchronous broadcast algorithm with message complexity n-1 and time complexity d, when a rooted spanning tree with depth d is known in advance.
- **Theorem 2**: There is an asynchronous broadcast algorithm with message complexity n-1 and time complexity d, when a rooted spanning tree with depth d is known in advance.

Convergecast

Problem

- Collect information from the nodes of the tree to the root.
- Each processor p_i starts with a value x_i.
- We wish to forward the maximum value among these values to the root p_r.



- **Theorem:** There is an asynchronous convergecast algorithm with message complexity n-1 and time complexity d, when a rooted spanning tree with depth d is known in advance.

Flooding and Building a Spanning Tree

Problem

- Broadcast without a preexisting spanning tree, starting from a distinguished processor p_r.

Solution

- **Flooding**
- Assume that m is the number of edges and n is the number of processes. How many messages does the flooding algorithm send?
- Can we modify the flooding algorithm to construct a spanning tree?

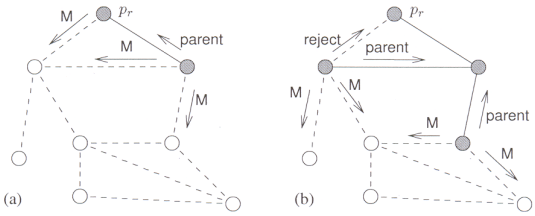
Modified flooding algorithm to construct a spanning tree: code for processor p_i, $0 \leq i \leq n-1$.

Initially $parent = \perp$, $children = \emptyset$, and $other = \emptyset$.

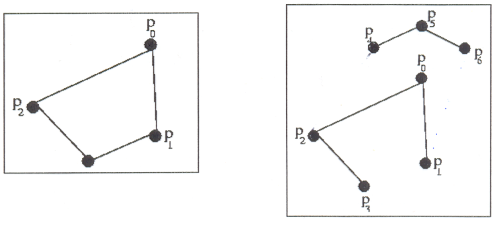
```
1:  upon receiving no message:
2:      if p_i = p_r and parent = ⊥ then
3:          send ⟨M⟩ to all neighbors
4:          parent := p_i

5:  upon receiving ⟨M⟩ from neighbor p_j:
6:      if parent = ⊥ then
7:          parent := p_j
8:          send ⟨parent⟩ to p_j
9:          send ⟨M⟩ to all neighbors except p_j
10:     else send ⟨already⟩ to p_j

11: upon receiving ⟨parent⟩ from neighbor p_j:
12:     add p_j to children
13:     if children ∪ other contains all neighbors except parent then
14:         terminate

15: upon receiving ⟨already⟩ from neighbor p_j:
16:     add p_j to other
17:     if children ∪ other contains all neighbors except parent then
18:         terminate
```

Flooding and Building a Spanning Tree: The F-SpanningTree Algorithm

The F-SpanningTree Algorithm



Two steps in the construction of the spanning tree.



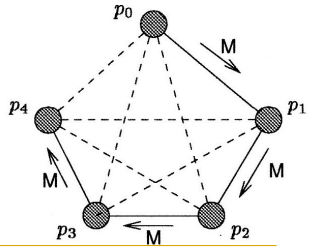
Correctness

Why is every node reachable from the root?

Why is there no cycle?

The F-SpanningTree Algorithm

- **Theorem:** There is an asynchronous algorithm to find a spanning tree of a network with m edges and diameter D, given a distinguished node, with message complexity O(m) and time complexity O(D).
- What kind of tree is the output of F-SpanningTree when the system is synchronous?
- **Theorem:** In every execution of F-SpanningTree in the synchronous model, the algorithm constructs a BFS tree rooted at p_r.
- What kind of tree can be the output of F-SpanningTree when the system is asynchronous?



Synchronous Systems

- We define a directed spanning tree of a directed graph G = (V,E) to be 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.
- A directed spanning tree of G with root node p_r is breadth-first provided that each node at distance d from p_r in G appears at depth d in the tree (that is at distance d from p_r in the tree).
- ✓ Every strongly connected digraph has a breadth-first directed spanning tree.
- ➢ Given that the G is a strongly connected directed graph and given that we have a distinguished node p_r, how can we design a synchronous algorithm that computes the directed BFS tree?
- ➢ How can a process learn which nodes are its children?
- ➢ What is the communication complexity of the algorithm in this case?
- ➢ What is the time complexity of the algorithm in this case?
- ➢ How can p_r learn that the construction of the spanning tree has terminated?

Constructing a Depth-First Search Spanning Tree for a Specified Root

Brief Description

- Each node maintains a set, called unexplored, of "unexplored" neighboring nodes and a set of nodes that will be its children in the constructed spanning tree.
- Initially, the root sends <M> to one of its neighbors and deletes this neighbor from unexplored.
- When a node p_i receives <M> for the first time from some node p_j, p_i marks p_j as its parent node in the spanning tree. Then, p_i chooses one of the nodes in unexplored and forwards <M> to it. If p_i does not receive <M> for the first time, it sends a message of type <already> to p_j and removes p_j from unexplored. If unexplored is empty, p_i sends a message of type <parent> to its parent node.
- When a node p_i receives a message of type <parent> or <already>, it sends <M> to one of the nodes in unexplored. If p_i has received <M> or a message of type <parent> or <already> from all its neighbors, p_i terminates.

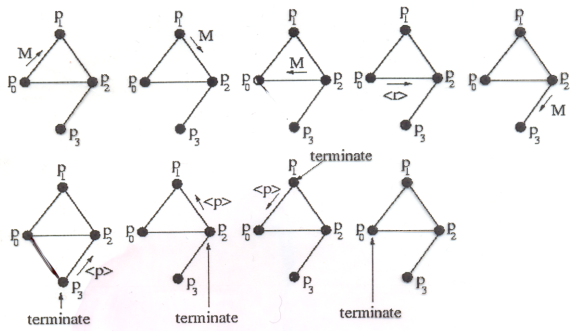
Algorithm 3 Depth-first search spanning tree algorithm for a specified root: code for processor p_i, $0 \leq i \leq n-1$.

Initially $parent = \perp$, $children = \emptyset$, $unexplored =$ all neighbors of p_i

```
1:  upon receiving no message:
2:      if p_i = p_r and parent = ⊥ then                    // root wakes up
3:      parent := p_i
4:      explore()

5:  upon receiving ⟨M⟩ from p_j:
6:      if parent = ⊥ then                                  // p_i has not received ⟨M⟩ before
8:          parent := p_j
9:          remove p_j from unexplored
10:         explore()
11:     else
12:         send ⟨already⟩ to p_j                           // already in tree
            remove p_j from unexplored
14: upon receiving ⟨already⟩ from p_j:
15:     explore()

16: upon receiving ⟨parent⟩ from p_j:
17:     add p_j to children
18:     explore()

19: procedure explore():
20:     if unexplored ≠ ∅ then
21:         let p_k be a processor in unexplored
22:         remove p_k from unexplored
23:         send ⟨M⟩ to p_k
24:     else
25:         if parent ≠ p_i then send ⟨parent⟩ to parent
26:         terminate                                       // DFS subtree rooted at p_i has been built
```

Constructing a Depth-First Search Spanning Tree for a Specified Root: The DFS-ST Algorithm

The DFS-ST Algorithm



$unexplored_0 = \{2\}$	$parent_0 = nil$	$children_0 = \{\}$
$unexplored_1 = \{0,2\}$	$parent_1 = nil$	$children_1 = \{\}$
$unexplored_2 = \{0,1,3\}$	$parent_2 = nil$	$children_2 = \{\}$
$unexplored_3 = \{2\}$	$parent_3 = nil$	$children_3 = \{\}$

The DFS-ST Algorithm

Correctness

- **Lemma**: In every execution of DFS-ST in the asynchronous model, DFS-ST constructs a DFS tree of the network rooted at p_r.

Communication Complexity

- **Lemma**: The communication complexity of DFS-ST is $O(m)$.
- **Proof**: Each node/process sends <M> at most once in each of the edges that are incident to it.
- Each node that receives <M> sends at most one message as a response on each of the edges that are incident to it.
- Thus, the number of messages sent is at most $2m$.

The DFS-ST Algorithm

Time Complexity

Lemma: The time complexity of DFS-ST is O(m).

Proof

- Since the time p_r executes its first step and before p_r terminates, there is always exactly one message in transit.
- No more than two messages are ever send on each edge.
- There are m edges in the graph.

Theorem: There is an asynchronous algorithm to find a depth-first search spanning tree of a network with m edges and n nodes, given a distinguished node, with message complexity O(m) and time complexity O(m).

Constructing a DFS Spanning Tree without a Specified Root

- How can we build a spanning tree when there is no distinguished node?

Brief Description

- Each processor that wakes up spontaneously attempts to build a DFS spanning tree with itself as the root, using a separate copy of DFS-ST.
- If two DFS trees try to connect to the same node, the node will join the DFS tree whose root has the higher identifier.
- p_m: the node with the maximal identifier among the nodes that wake up spontaneously.

Algorithm 4 Spanning tree construction: code for processor p_i, $0 \leq i \leq n-1$.

Initially $parent = \perp$, $leader = -1$, $children = \emptyset$, $unexplored =$ all neighbors of p_i

```
1:  upon receiving no message:
2:      if parent = ⊥ then                          // wake up spontaneously
3:          leader := id
4:          parent := p_i
5:          explore()

6:  upon receiving ⟨leader,new-id⟩ from p_j:
7:      if leader < new-id then                     // switch to new tree
8:          leader := new-id
9:          parent := p_j
10:         children := ∅
11:         unexplored := all neighbors of p_i except p_j
12:         explore()
13:     else if leader = new-id then
14:         send ⟨already,leader⟩ to p_j            // already in same tree
                // otherwise, leader > new-id and the DFS for new-id is stalled

15: upon receiving ⟨already,new-id⟩ from p_j:
16:     if new-id = leader then explore()

17: upon receiving ⟨parent,new-id⟩ from p_j:
18:     if new-id = leader then                     // otherwise ignore message
19:         add p_j to children
20:         explore()

21: procedure explore():
22:     if unexplored ≠ ∅ then
23:         let p_k be a processor in unexplored
24:         remove p_k from unexplored
25:         send ⟨leader,leader⟩ to p_k
26:     else
27:         if parent ≠ p_i then send ⟨parent,leader⟩ to parent
28:         else terminate as root of spanning tree
```

Constructing a DFS Spanning Tree without a Specified Root

Constructing a DFS Spanning Tree without a Specified Root

Correctness

- <leader> messages with leader id m are never dropped because of discovering a larger leader id, by definition of m.
- <already> messages with leader id m are never dropped because they have the wrong leader id.
- <parent> messages with leader id m are never dropped because they have the wrong leader id.
- messages with leader id m are never dropped because the recipient has terminated.
- Thus, the instance of DFS-ST for leader id m completes, and correctness of DFS-ST implies correctness of Algorithm 4.

- Message complexity?
- Time complexity?

Constructing a DFS Spanning Tree without a Specified Root

- Theorem: Algorithm 4 finds a spanning tree of a network with m edges and n nodes, with message complexity $O(nm)$ and time complexity $O(m)$.

Bibliography

These slides are based on material that appears in the following books:

- H. Attiya & J. Welch, Distributed Computing: Fundamentals, Simulations and Advanced Topics, Morgan Kaufmann, 1998 (Chapter 1)
- N. Lynch, Distributed Algorithms, Morgan Kaufmann, 1996 (Chapters 1 & 9).