Distributed and Parallel Computing

Lecture 13

Alan P. Sexton

University of Birmingham

Spring 2018

Wave Algorithms

A *wave* algorithm sends requests through the whole network to gather information.

Applications include:

- Termination detection
- Routing
- Leader election
- Transaction commit voting in the case of network partitions

To be a wave algorithm, it must meed 3 conditions:

- It must be finite
- It contains one or more *decide* events
- For each decide event a, and process p, $\exists$ event b in p . $b \prec a$

Wave Algorithms

A *wave* algorithm sends requests through the whole network to gather information.

Applications include:

- Termination detection
- Routing
- Leader election
- Transaction commit voting in the case of network partitions

To be a wave algorithm, it must meed 3 conditions:

- It must be finite
- It contains one or more *decide* events
- For each decide event a, and process p, $\exists$ event b in p . $b \prec a$
  - i.e. every process must participate in each decide event

Traversal Algorithms

A *traversal* algorithm is a type of *wave* algorithm

- An *initiator* sends a *token* to visit each process in the network
- The *token* may collect and/or distribute information on the way
- The *token* eventually returns to the *initiator* with the accumulated information
- The *initiator* makes the *decision*.
- Note that a *token* can only be at one process at any one time

Traversal algorithms can be used to build a *spanning tree* of the network

- The initiator becomes the *root* of the spanning tree
- For every other node, its *parent* is the node from which it received the token for the first time

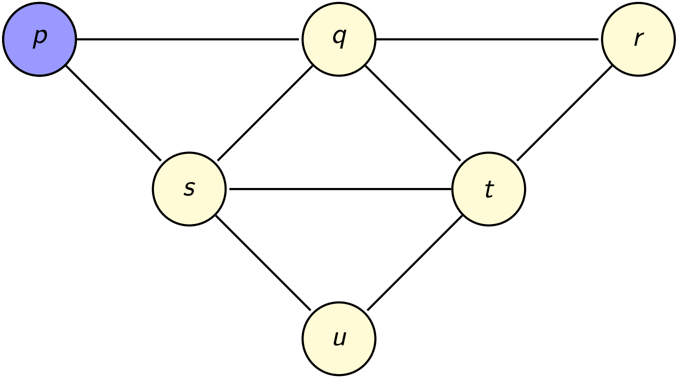
Tarry's Algorithm

Tarry's algorithm is a traversal algorithm for undirected networks. It is based on two rules:

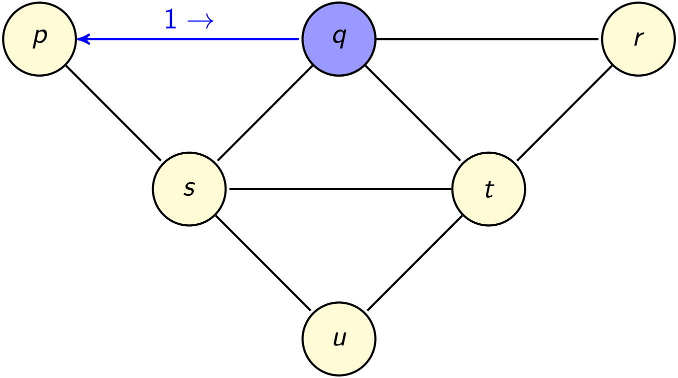
1. A process never forwards the token through the same channel twice
2. A process only forwards the token to its parent when there is no other option

These rules ensure that the token travels through every channel exactly twice, once in each direction, and ends up back at the initiator

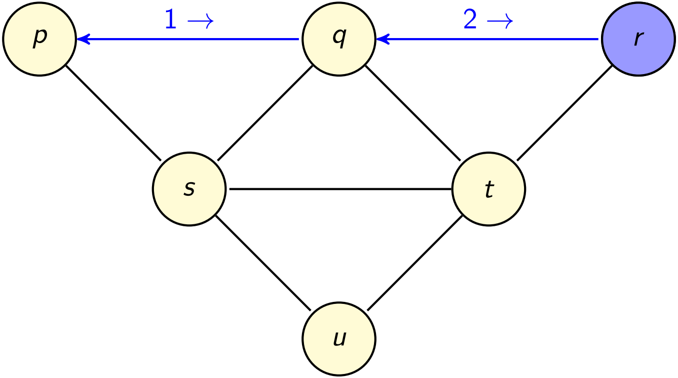
Example Execution of Tarry's Algorithm



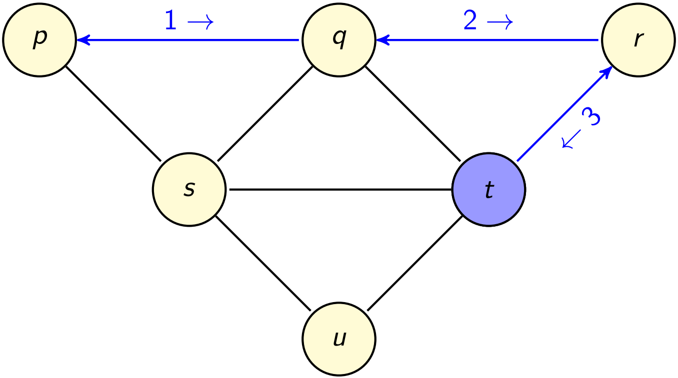
Example Execution of Tarry's Algorithm



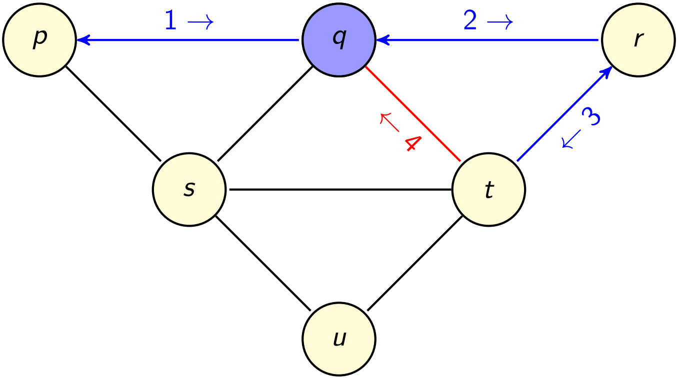
Example Execution of Tarry's Algorithm



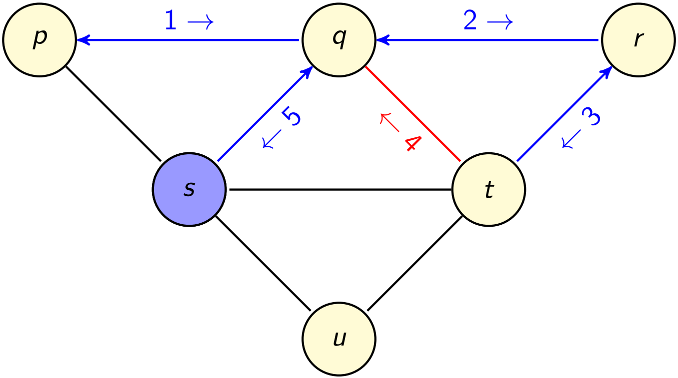
Example Execution of Tarry's Algorithm



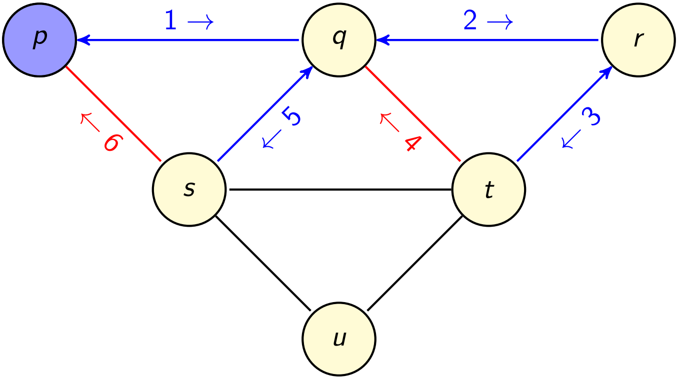
Example Execution of Tarry's Algorithm



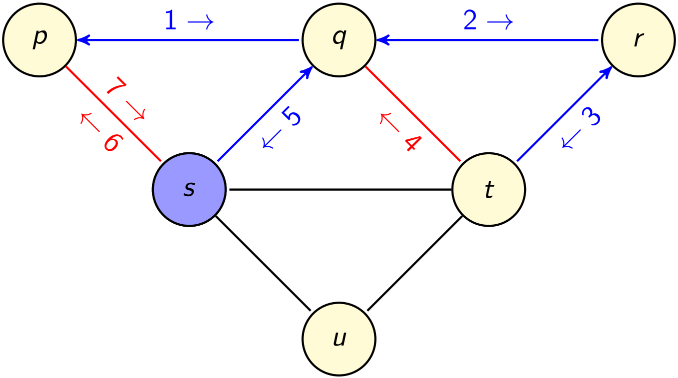
Example Execution of Tarry's Algorithm



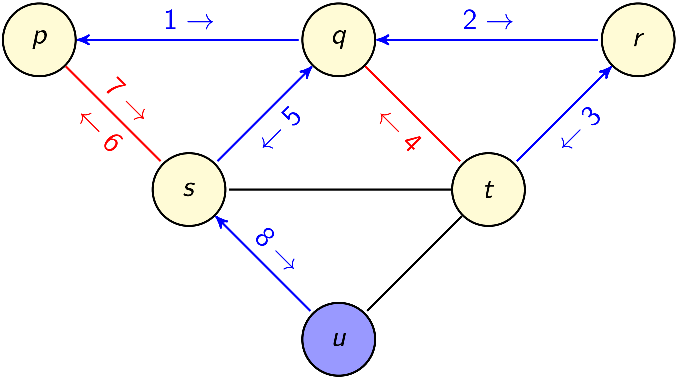
Example Execution of Tarry's Algorithm



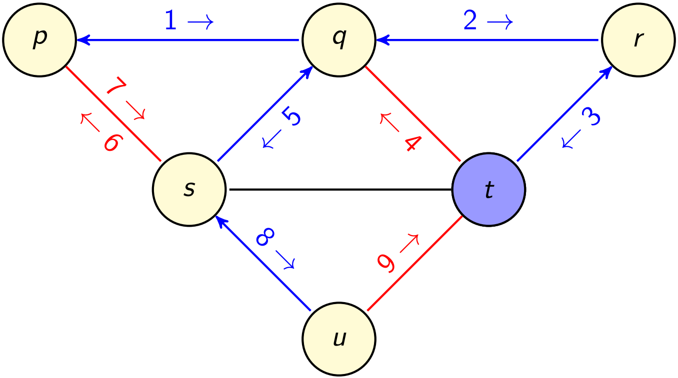
Example Execution of Tarry's Algorithm



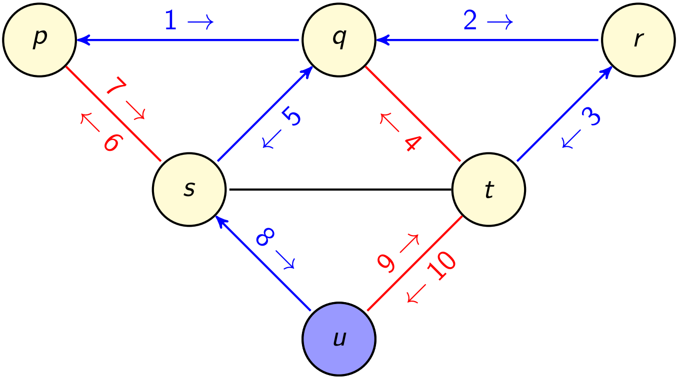
Example Execution of Tarry's Algorithm



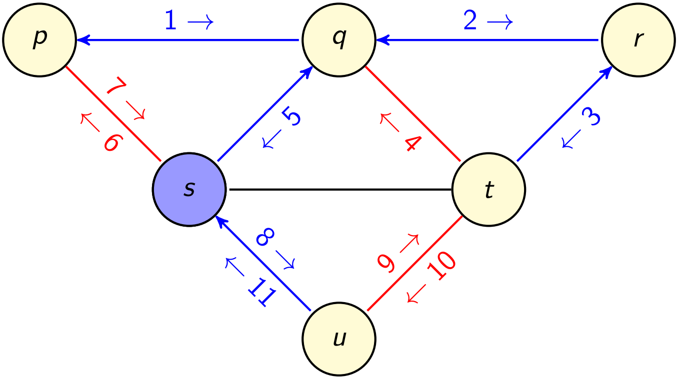
Example Execution of Tarry's Algorithm



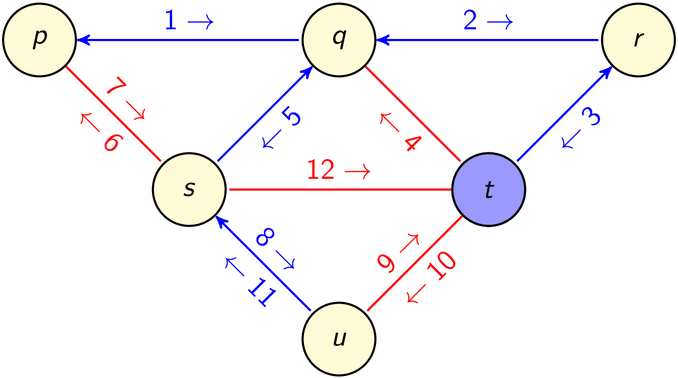
Example Execution of Tarry's Algorithm



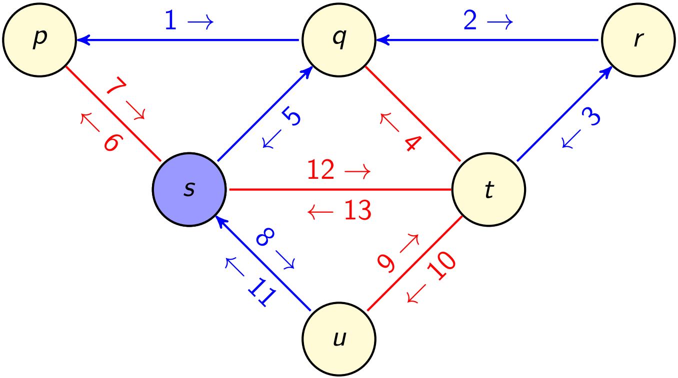
Example Execution of Tarry's Algorithm



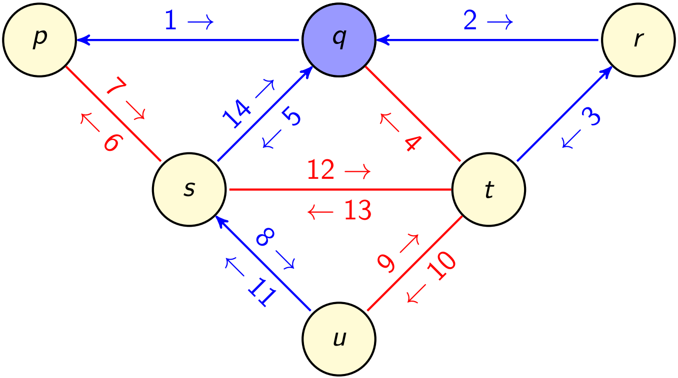
Example Execution of Tarry's Algorithm



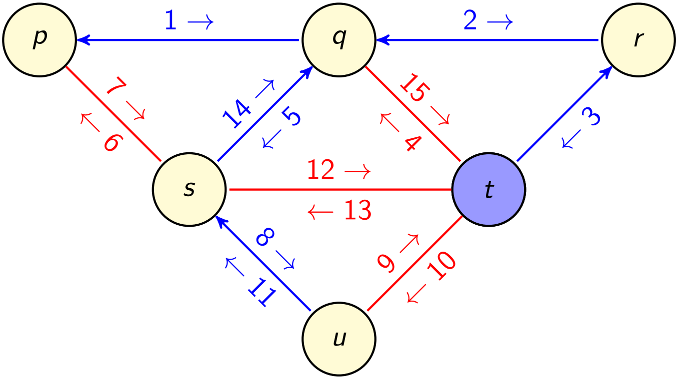
Example Execution of Tarry's Algorithm



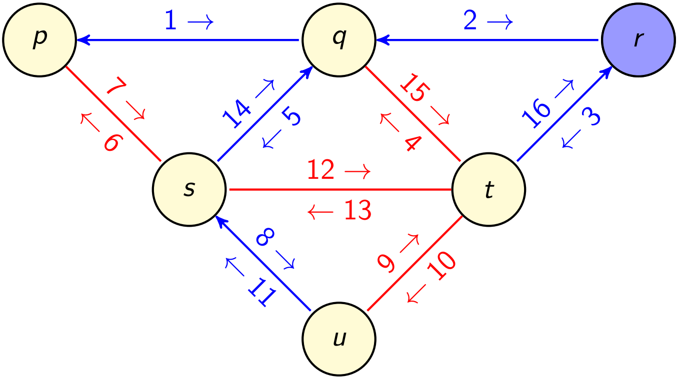
Example Execution of Tarry's Algorithm



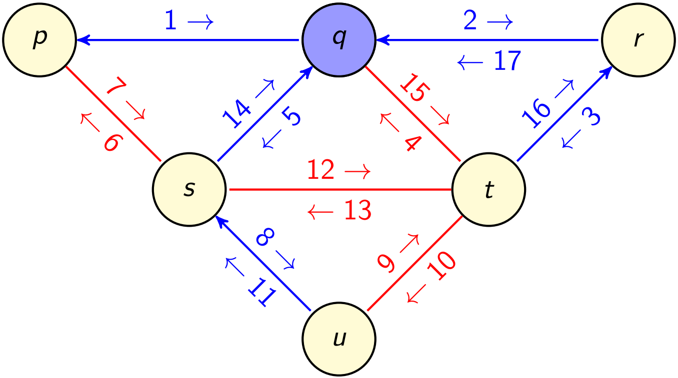
Example Execution of Tarry's Algorithm



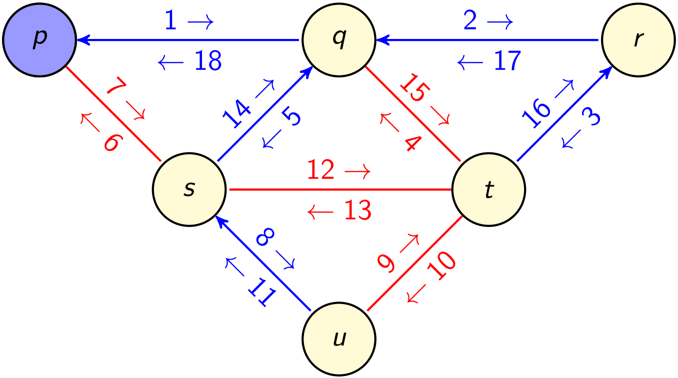
Example Execution of Tarry's Algorithm



Example Execution of Tarry's Algorithm



Example Execution of Tarry's Algorithm



Correctness of Tarry's Algorithm, part 2

To prove:

1. The token travels through each channel twice, once in each direction
2. The token ends up at the initiator

Correctness of Tarry's Algorithm, part 2

To prove:

1. The token travels through each channel twice, once in each direction
2. The token ends up at the initiator

Proof that the token ends up at the initiator:

- Rule 1 $\Rightarrow$ the token is never sent through the same channel in the same direction twice

Correctness of Tarry's Algorithm, part 2

To prove:

1. The token travels through each channel twice, once in each direction
2. The token ends up at the initiator

Proof that the token ends up at the initiator:

- Rule 1 $\Rightarrow$ the token is never sent through the same channel in the same direction twice
- Each time a non-initiator holds the token, it has received it one more time than it has sent it to at least one neighbour

Correctness of Tarry's Algorithm, part 2

To prove:

1. The token travels through each channel twice, once in each direction
2. The token ends up at the initiator

Proof that the token ends up at the initiator:

- Rule 1 $\Rightarrow$ the token is never sent through the same channel in the same direction twice
- Each time a non-initiator holds the token, it has received it one more time than it has sent it to at least one neighbour
- Hence there is still a channel into which it has not yet sent the token

Correctness of Tarry's Algorithm, part 2

To prove:

1. The token travels through each channel twice, once in each direction
2. The token ends up at the initiator

Proof that the token ends up at the initiator:

- Rule 1 $\Rightarrow$ the token is never sent through the same channel in the same direction twice
- Each time a non-initiator holds the token, it has received it one more time than it has sent it to at least one neighbour
- Hence there is still a channel into which it has not yet sent the token
- Therefore, by rule 1, it can send the token into this channel

Correctness of Tarry's Algorithm, part 2

To prove:

1. The token travels through each channel twice, once in each direction
2. The token ends up at the initiator

Proof that the token ends up at the initiator:

- Rule 1 $\Rightarrow$ the token is never sent through the same channel in the same direction twice
- Each time a non-initiator holds the token, it has received it one more time than it has sent it to at least one neighbour
- Hence there is still a channel into which it has not yet sent the token
- Therefore, by rule 1, it can send the token into this channel
- Hence, at the end of the algorithm, no non-initiator can be holding the token

Correctness of Tarry's Algorithm, part 2

To prove:

1. The token travels through each channel twice, once in each direction
2. The token ends up at the initiator

Proof that the token ends up at the initiator:

- Rule 1 $\Rightarrow$ the token is never sent through the same channel in the same direction twice
- Each time a non-initiator holds the token, it has received it one more time than it has sent it to at least one neighbour
- Hence there is still a channel into which it has not yet sent the token
- Therefore, by rule 1, it can send the token into this channel
- Hence, at the end of the algorithm, no non-initiator can be holding the token
- Hence, at the end of the algorithm, the token is with the initiator

Correctness of Tarry's Algorithm, part 2

Proof that the token travels through each channel twice, once in each direction:

Correctness of Tarry's Algorithm, part 2

Proof that the token travels through each channel twice, once in each direction:

- Assume that, at termination, some channel C_{pq} has not been traversed by the token in each direction.

Correctness of Tarry's Algorithm, part 2

Proof that the token travels through each channel twice, once in each direction:

- Assume that, at termination, some channel C_{pq} has not been traversed by the token in each direction.
- Let p be the earliest visited (non-initiator) process for which such a channel exists and s be the parent of p

Correctness of Tarry's Algorithm, part 2

Proof that the token travels through each channel twice, once in each direction:

- Assume that, at termination, some channel C_{pq} has not been traversed by the token in each direction.
- Let p be the earliest visited (non-initiator) process for which such a channel exists and s be the parent of p
- Therefore s was visited before p

Correctness of Tarry's Algorithm, part 2

Proof that the token travels through each channel twice, once in each direction:

- Assume that, at termination, some channel C_{pq} has not been traversed by the token in each direction.
- Let p be the earliest visited (non-initiator) process for which such a channel exists and s be the parent of p
- Therefore s was visited before p
- Hence all channels of s were traversed in both directions

Correctness of Tarry's Algorithm, part 2

Proof that the token travels through each channel twice, once in each direction:

- Assume that, at termination, some channel C_{pq} has not been traversed by the token in each direction.
- Let p be the earliest visited (non-initiator) process for which such a channel exists and s be the parent of p
- Therefore s was visited before p
- Hence all channels of s were traversed in both directions
- Hence p must have sent the token to its parent s

Correctness of Tarry's Algorithm, part 2

Proof that the token travels through each channel twice, once in each direction:

- Assume that, at termination, some channel C_{pq} has not been traversed by the token in each direction.
- Let p be the earliest visited (non-initiator) process for which such a channel exists and s be the parent of p
- Therefore s was visited before p
- Hence all channels of s were traversed in both directions
- Hence p must have sent the token to its parent s
- But, by rule 2, p must have sent the token into all its other channels before sending it to its parent

Correctness of Tarry's Algorithm, part 2

Proof that the token travels through each channel twice, once in each direction:

- Assume that, at termination, some channel C_{pq} has not been traversed by the token in each direction.
- Let p be the earliest visited (non-initiator) process for which such a channel exists and s be the parent of p
- Therefore s was visited before p
- Hence all channels of s were traversed in both directions
- Hence p must have sent the token to its parent s
- But, by rule 2, p must have sent the token into all its other channels before sending it to its parent
- Since p has sent and received the token an even number of times, it must have received it back through all its channels

Correctness of Tarry's Algorithm, part 2

Proof that the token travels through each channel twice, once in each direction:

- Assume that, at termination, some channel C_{pq} has not been traversed by the token in each direction.
- Let p be the earliest visited (non-initiator) process for which such a channel exists and s be the parent of p
- Therefore s was visited before p
- Hence all channels of s were traversed in both directions
- Hence p must have sent the token to its parent s
- But, by rule 2, p must have sent the token into all its other channels before sending it to its parent
- Since p has sent and received the token an even number of times, it must have received it back through all its channels
- Contradiction

More on Tarry's Algorithm

Performance:

- Number of messages:
- Time to complete:

More on Tarry's Algorithm

Performance:

- Number of messages: $2E$
- Time to complete:

More on Tarry's Algorithm

Performance:

- Number of messages: $2E$
- Time to complete: $2E$ time units

More on Tarry's Algorithm

Performance:

- Number of messages: $2E$
- Time to complete: $2E$ time units
- Note that this is a serial algorithm: There is only one token and only one process can send the token down one channel at a time — think of football players passing the ball

Tarry's algorithm and Depth First Search

In a depth first search, the token is forwarded to a process that has not yet held the token in preference to one that has.

A depth first spanning tree is one that could have been created by a depth-first search.

A depth first spanning tree will have its frond edges connecting nodes only to their ancestors or descendents in the spanning tree.

- Edges are frond edges if they connect a node to a node that has already been visited.
- In a depth first search, all nodes in a subtree are searched before any nodes in a sibling subtree
- Hence in a depth first search, frond edges will only connect ancestor-descendent pairs

Tarry's algorithm and Depth First Search

We can make Tarry's algorithm generate a depth-first spanning tree by adding an extra rule:

- When a process receives the token, it immediately sends it back through the same channel if allowed by rules 1 and 2.

Note: this does **NOT** make the search in Tarry's algorithm depth-first, but when it diverges off depth-first, it puts it back onto the depth-first track before any more parent-child edges are added to the spanning tree.

There is no extra cost to this change.

Benefits of Depth-First Search

If a depth-first spanning tree is created by the modified Tarry's algorithm:

- We can optimise the algorithm by letting the token carry the information of all processes that have held it:
- Avoid sending the token along frond edges at the cost of extra memory required in the token
- Messages only travel along spanning tree edges, so $2E$ messages reduced to $2N-2$
- Time complexity reduced from $2E$ to $2N-2$ time units.
- Bit complexity increases from $O(1)$ to $O(N \log N)$, where $O(\log N)$ bits are needed to represent the process identifiers.