



Figure 1: A Spanning Tree

Last Day: broadcast and building a spanning tree

Initiator:

- send m to all neighbours
- wait for ack from all neighbours
- while waiting send ack to anyone who forwards message to him

Other: When receiving message for the first time (say from p)

- remember p as his parent
- forward msg to all neighbours except p
- wait for ack from all of them
- then send ack to parent
- Also, if you receive message again, send ack back to sender immediately

$\Theta(m)$ message for first broadcast

subsequent broadcast can use existing tree - $\Theta(n)$ messages

NOTE: $n = \#$ nodes, $m = \#$ edges

Why this works:

- message reaches everyone (argued last day) (spans graph)
- each of $n - 1$ nodes has a parent pointer: $n - 1$ edges in total (must be a spanning tree) Figure 1 shows an example spanning tree.

Why does initiator eventually get acks back?

- For every edge not in spanning tree ack is sent back right away

Also, within tree, each node eventually sends ack to its parent (can prove this by induction on height of node in tree)

Remark about Synchronous networks

- If system happens to be synchronous then the spanning tree constructed will be a BFS (breadth first search) tree
- This is most efficient way to do a broadcast from the root

What if lots of processes want to broadcast?

Option 1:

- Pick one process to initiate the building of a spanning tree
- Then everyone can use the tree for broadcasts
- The difficulty in this approach is in choosing a leader. This problem is called leader election

Option 2:

- Doesn't assume the network has a leader
- Does assume all processes have unique ids
- every process P starts running previous algorithm to build a spanning tree rooted at P
- Include in all the messages (including acks) the id of the root. (So that different executions don't interfere with one another)
- In the end, only one of the trees will get completely built
- The final tree will be the one whose root has the smallest id
- Whenever a node gets a message with an id smaller than any id it has previously seen, it throws away all previous work and starts working on tree of the message it has just received
- Ignore any message that arrives with larger id than the tree currently being worked on
- Process with smallest id will not participate in building anyone else's tree. So, only one tree will get completely built.
- When that one tree's root gets all its acks, it can broadcast a message saying the tree is ready for use by all processes.

(also solves leader election -> process with the smallest id knows it is the smallest)

Message Complexity ($n = \#$ nodes, $m = \#$ edges) $\Theta(nm)$

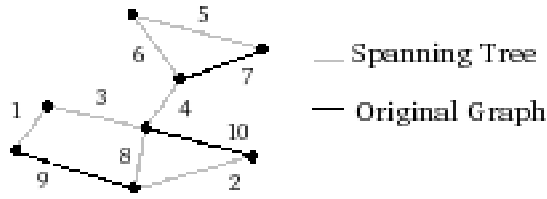


Figure 2: Creating A Spanning Tree

Can we build a spanning tree using fewer messages?

Goal: if edges are weighted with costs to use the edges, try to build spanning tree with smallest total cost. This is called the MST (Minimum spanning tree). Refer to figure 2 for an example.

LEMMA: If edge weights are distinct there exists a unique MST.

(Proof left as exercise for reader.)

NOTE: If edge weights are not distinct, we can make them distinct by taking the weight of edge (i,j) to be $(w(i,j), i, j)$

- this method breaks ties in a consistent way
- $(w(i, j), i, j)$ is distinct for all edges
- We compare edge weights using lexicographic ordering: $(x, y, z) < (x', y', z')$ if $(x < x')$ or $(x = x'$ and $y < y')$ or $(x = x'$ and $y = y'$ and $z < z')$

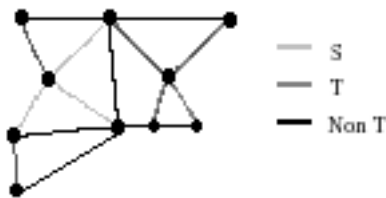
Distributed algorithm to find the (unique) MST

(Gallagher, Humblet, Spira 1983)

- A version of this algorithm works in asynchronous system, but we'll consider only special case of synchronous system

Basic Idea: Start building little pieces of the MST and slowly connect them together

LEMMA: If S is a proper subgraph of the MST T then the smallest weight edge with exactly one end in S is in T



PROOF: Suppose not

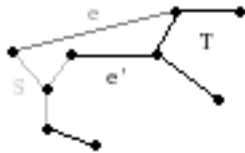
Let e be the smallest edge with exactly one end in S
Imagine adding e to T



This creates a cycle

Let e' be the other edge on the cycle that has exactly 1 end in S

Then $T' = T \cup \{e\} - \{e'\}$ is a spanning tree with smaller total weight than T



→ contradiction

This proves the lemma. Construction of algorithm to be continued next day.