Constructing Spanning Trees

Constructing Spanning Trees

- Important applications
  - Broadcast
  - Convergecast
  - Building loop-free structures for communication
    - Spanning Tree Protocol (IEEE 802.1D/IEEE 802.1Q) for switches
- Types
  - Arbitrary
  - BFS Spanning Tree
  - DFS Spanning Tree
  - Shortest path tree
  - MST
  - ….

- Constructing arbitrary spanning trees
  - Can use flooding
- Constructing shortest path trees
  - Distributed Bellman Ford
- Constructing DFS spanning trees
  - An example of a straightforward transformation of a sequential algorithm to a distributed algorithm,
  - We will do this
- Constructing BFS spanning trees
  - Can grow the tree layer-by-layer with appropriate synchronization
  - You will do this in assignment
- Constructing MST
  - More complex

Constructing a DFS tree with given root

- Plain parallelization of the sequential algorithm by introducing synchronization
- Each node i has a set *unexplored*, initially contains all neighbors of i
- A node i (initiated by the root) considers nodes in *unexplored* one by one, sending a neighbor j a message M and then waiting for a response (*parent* or *reject*) before considering the next node in *unexplored*
- If j has already received M from some other node, j sends a *reject* to i

- Else, j sets i as its parent, and considers nodes in its unexplored set one by one
- j will send a *parent* message to i only when it has considered all nodes in its unexplored set
- i then considers the next node in its *unexplored* set
- Algorithm terminates when root has received *parent* or *reject* message from all its neighbours
- Worst case no. of messages = 4m
- Time complexity O(m)

Initially *parent* equals nil, *children* is empty
and *unexplored* includes all the neighbors of p_i

```
1:  upon receiving no message:
2:      if i = r and parent is nil then
3:          parent := i
4:          let p_j be a processor in unexplored
5:          remove p_j from unexplored
6:          send M to p_j

7:  upon receiving M from neighbor p_j:
8:      if parent is nil then                    // p_i has not received M before
9:          parent := j
10:         remove p_j from unexplored
11:         if unexplored ≠ ∅ then
12:             let p_k be a processor in unexplored
13:             remove p_k from unexplored
14:             send M to p_k
15:         else send ⟨parent⟩ to parent
16:     else send ⟨reject⟩ to p_j
```

```
17: upon receiving ⟨parent⟩ or ⟨reject⟩ from neighbor p_j:
18:     if received ⟨parent⟩ then add j to children
19:     if unexplored = ∅ then
20:         if parent ≠ i then send ⟨parent⟩ to parent
21:         terminate                    // DFS sub-tree rooted at p_i has been built
22:     else
23:         let p_k be a processor in unexplored
24:         remove p_k from unexplored
25:         send M to p_k
```

Taken from:
Distributed Computing – Fundamentals, Simulations, and Advanced Topics
by Hagit Attiya and Jennifer Welch

What if no root is given?

- Let all nodes think they are roots at first!!
- Each starts growing a DFS tree rooted at it
- When two trees collide, use the root id to decide which tree will continue to grow (say the one with the larger root id wins)
- Kind of vague, can you modify the pseudocode from the first algorithm to write the algorithm neatly?