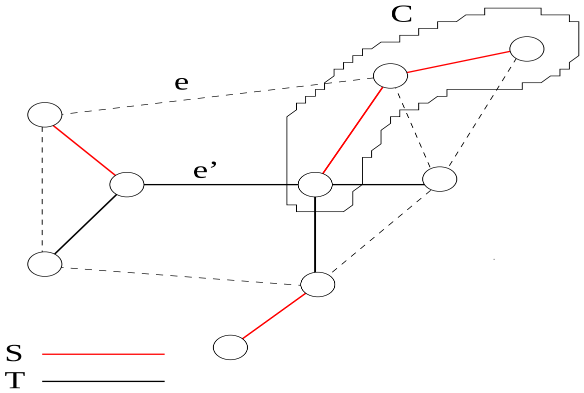
1 Spanning Tree Algorithm

Recall: Algorithm in Lecture 4 found a spanning tree using $O(nm)$ messages
Today: better algorithm. Reference: [Lyn96], Section 4.4.
Model:

Synchronous message passing system
processes have unique ids
processes don't know network topology

build MIN spanning tree (each edge has weight, which might represent the cost of using that edge) $\rightarrow$ looking for min total weight. From the known MST algorithms, we will adapt Prim's to distrib. system.
Assumption – edge weights are distinct (in case of ties, we break them according to process ids). This assumption allows us to claim that such MST is unique (Exercise; for solution see www.cs.yorku.ca/~ruppert/3101S-01/a4s.ps, question 2).

Lemma 1:
Let T be the MST
Let $S \subset T$ (proper subset)
Let C be connected component of S.
Then the lightest edge that has exactly 1 endpoint in C is in T



Lemma 1 Proof

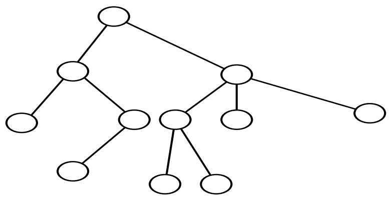
Proof:
Suppose e is the min weight edge with exactly 1 endpoint in C
Suppose $e \notin T$
There is at least one tree edge e' that has exactly 1 end point in C and is on cycle of $T \bigcup \{e\}$. Then change e' to e and we have a better spanning tree:
Let $T' = T \bigcup \{e\} - \{e'\}$. This is still a spanning tree.
$w(e) < w(e')$ since e is lightest edge with one endpoint in C
$wt(T') < wt(T) \Rightarrow$ contradiction

MST Algorithm

Proceed in phases. In each phase
Add one more edge to each tree in the forest computed so far

(the lightest edge with exactly one end pt in the tree)
Invariant throughout the algorithm:
At any time, we have forest $F \subseteq$ MST. For each tree in F, we'll have a leader

To figure out which edge to add, we must find the lightest edge that has one endpoint in the tree:

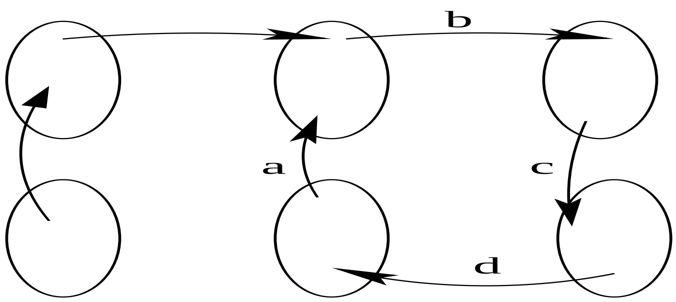


every proc p first finds lightest edge that has one end at p and other end in a different component of F (i.e. the other end pt has a different leader). p sends a message across each of its channels to ask whether the other endpoint is in a different component. Then,

$O(n)$ messages throughout whole graph	Send that info to your leader (using tree convergecast) Leader computes min. and broadcasts to everyone in tree

convergecast – opposite from the broadcast

<u>FACT</u>
In each new component created by new edges there is exactly one edge that is the edge chosen by both end pts.
Proof sketch: Suppose k components are combined into a single component.
Each adds one (directed) edge out of it to the forest.
So there is exactly one (directed) cycle. If cycle has length $> 2 \Rightarrow$ each edge on the cycle has bigger weight then the next one $\Rightarrow$ contradiction ($a > b > c > d > a$) (See figure below)



Forest

$\Rightarrow$ cycle must be of length 2 which is the same undirectional edge
$\rightarrow$ use the smaller of 2 endpts of that edge as new leader of the component (leader broadcasts his id to everyone in component using spanning tree).
This allows us to compute the new leader within each new component without having to do anything extra: it just turns out that there is a unique node in each component that satisfies a certain

property (it chose an edge that was also chosen by another process with a larger id) and that node can determine that it has that property just by seeing the local changes in the forest.
Use synchrony to start all phases at same time at every node. After i phases, how big are the components of F?
Each component has $\geq 2^i$ nodes (easy to prove by induction, since every component joins with at least one other component in each phase)
$\Rightarrow \leq \log(n)$ phases needed.

How many messages?
All the broadcasts and convergecasts in phase $i \rightarrow O(n)$
$\Rightarrow$ total over alg is $O(n \log n)$
"Probe" messages to determine whether other end pt is in same component $\rightarrow O(m)$ per phase.

Better analysis: we only get "yes" ("yes" meaning the other endpoint is in the same component as you are) for each edge once and after that there is no need to "ask" that edge again $\rightarrow$ Total number of "yes" answers is $O(m)$ throughout whole algorithm
Each node only gets "no" once in each phase if it starts by probing the smallest adjacent edge and probes in order by weight, so total number of messages used by probes that return "no" answers is $O(n)$ per phase.
Thus, in total we get $O(n \log n + m)$ message complexity
This is optimal: we need at least m to build a tree (ask every edge; see Jan 16 lecture) and we need at least $\Omega(n \log n)$ to select a leader in a ring.

Time complexity: synchronous system, so it is proportional to # of steps per phase $\times$ # of phases $\rightarrow O(n \log n)$ upper bound
NOTE: This alg can be adapted to run in asynchronous model too (still using $O(m + n \log n)$ messages); see [Lyn96], Section 15.5 for details.

2 Byzantine Agreement

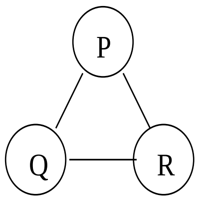
Historically comes from Albanian generals problem. *Byzantine Failures* – when processes act arbitrarily badly (even maliciously)
Model: perfect channels

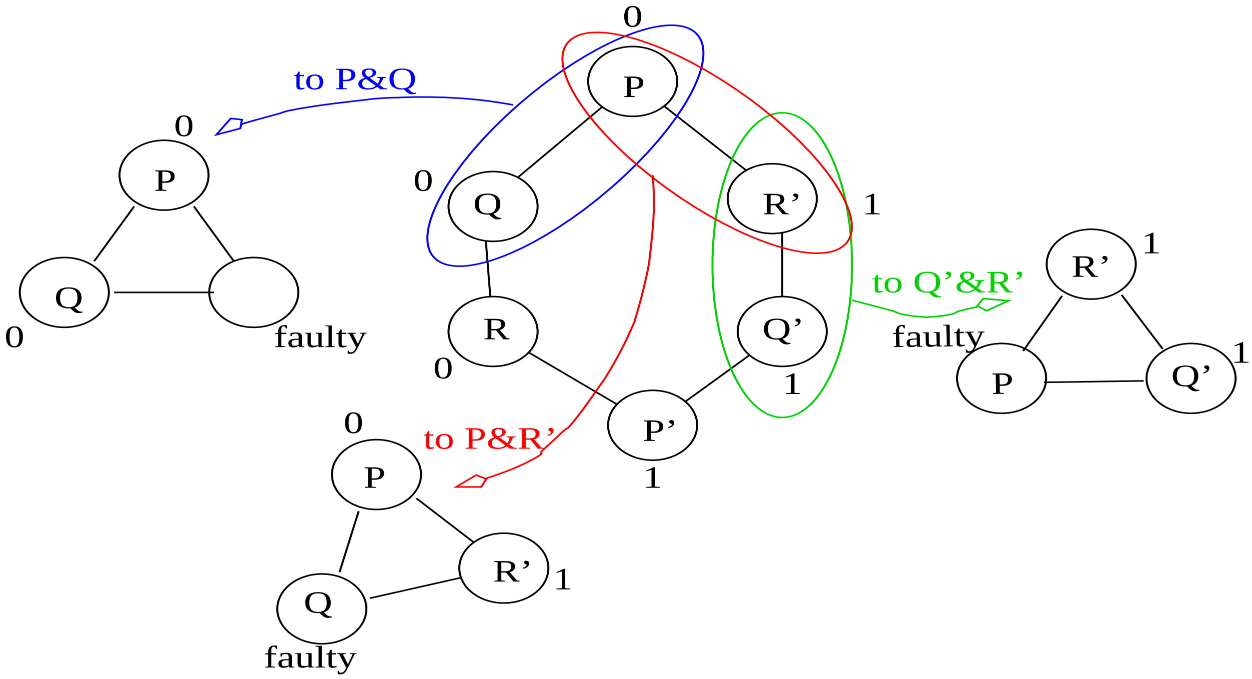
Agreement Problem

- Every proc starts with an input value
- Every non-faulty proc must produce an output (*termination*)
- Every non-faulty proc must produce the same output (*weak agreement*)
- If all inputs are same that should be the output value of non-faulty processes (*weak validity*)

THEOREM
Impossible if $n = 3$, # failures $= 1$ (even in synchronous model). Reference: [AW98], Section 5.2.3 or [Lyn96], Section 6.4. Originally proved by Pease, Shostak and Lamport (JACM, 1980).

Proof:
Suppose an alg exists that solves the Byzantine Agreement problem for this case, where $n = 3$ and $f = 1$. Imagine that we run the algorithm on the processes of a 6-process ring, giving 3 processes input 0 and the other 3 processes input 1 as shown below.





Running Algorithm for n=3 on 6 node system

To P and Q, it seems that they are running in the 3-process system with a faulty R, so they must output 0 (by weak validity condition). Similarly, it looks to R' and Q' as if they are running in a system with a faulty process P, so they must output 1 (again by weak validity).
In the third case (for P & R' triangle) we know that it looks to P & R' as if they are running in a 3-process system with a faulty process Q, so they must both output the same value (by weak agreement), we just don't know what. From the previous 2 triangles, however, we know that P & R' output different values
$\Rightarrow$ contradiction
$\Rightarrow$ impossible to do the Agreement Problem with 3 processes, if one of the three might experience a Byzantine failure.