Last Tue: Described a collection of standard network algos: for
bread, coast, & setting up various communication tree structures
BFS, shortest-paths, MST
Today finished with one more tree structure, then on to consensus in the
presence of failures.

Analysis of MST Last time was a bit rushed
MST: $O(1 + n \log n)$, for an optimized version of the

Sweep line** algorithm, in which each node checks edges
in order of weights, smallest first, one at a
time, to find MWOE. Once up an edge,
doesn't need to try it again.Repeat same component
When accepts an edge (different component), may
accept again later.

Most wign $O(n \log n)$, for
all up to # of phases
on tree edges.

Do Except: Test--accept--reject.

 amortized analysis:

  3 on each edge: Do test-reject at most once, total $O(1+1)$.

  3 on each node at each place: at most one test-accept,

  total $O(n \log n)$. 

  **Sweep line: topological order.
Time: $O \left( \frac{n \log m}{m} \right) \frac{1}{\# \text{ of phases}}$

$O(n)$ time per phase - including tie to breadth, cost

Cost for each node to explore all out-edges it needs to.

(Because each node starts this exploration)

as soon as it gets the breadth rug.

Back to this for anywhere else.
Using spanning tree to elect leader
Fan in from leaves, until meet at a node, a cross of edge.
Choose meeting pt as leader (highest in an edge)
Size $O(n)$
$log n$

Then under graph, no leader, can get leader in
$O(n \log n)$
$log n$ $O(n \log n + 1 + 1)$
Summary: We've seen direct algorithms for setting up various kinds of spanning trees: \( BF > S\text{-path} > \text{MST} \)

Example to show the difference:

- Weighted undirected graph

```
MST:
```

```
BF from root A:
```

```
S-P from A:
```

Return to all these in the asynchronous setting.

Depth-first search

Can also design a distributed algorithm to carry out a depth-first traversal of a connected undirected graph, to set up the tree, and along the way:

(a) Having each node know which of its edges are in the tree, and

(b) Having each node learn its number in the DF ordering
Although this is constructed by a distributed algorithm, the style of the algorithm is very "sequential," in that progress is only being made in one place at a time.

Assume a root $i$. It initiates a single token and passes it around, letting it carry the count of the number of nodes it has marked. The root (or even other nodes) keeps track of all the "unexplored" incident edges.

It starts by sending token on an unexplored edge, and marks that edge as explored.

Token is in "explore" mode.
Algorithm:

1. When node i receives token with tag "explore" from node j,
   - If i has already been numbered, then sends token back to j with "reject" tag.
   - Else (i hasn't been numbered yet)
     - Number i with the next number.
     - Set parent to j.
     - Make edge to j as explored.
     - If have any unexplored edges remaining then pick one, send then to edge + mark as explored.
     - Else send token to parent with "done" tag.

2. When node i receives token with tag "reject" from node j,
   - If have any unexplored edges, pick one, send then to edge with "explore" tag + mark as explored.
   - Else send token to parent with "done" tag.

3. When i receives token with tag "done" from node j,
   - Add j to set of known children.
   - If done! (same as *)

Remarks: when i, j has nothing left to explore.

Easy to see it constructs DFS tree. Like sequential alg.

Maps: \( K + \sum_{i} E_i \) (111)

- Can traverse from either end in "explore" mode.
- Get response back, either "reject" or "done".

4) \(|E| \) for # of undirected edges

(2) \(|E| \) if we count the way I do, where I undir edge

= 2 directed edges.
Time: $O(\sqrt{E})$ also

Result due to Chiang, in paper 1853.

that also considers max flow problems in directed networks.

By now, it should be clear we can come up with a distributed
version of practically any graph problem.
Most of these have been well studied.

Q: What about constructing a DFS tree in a network with no root?

Attiya-Welch describe a simple strategy: basically, everyone
starts a DFS of the entire network, labelled with its
own identifier, in parallel.

In the end, everyone has seen all—discard all but the
highest id's tree.

\[
\begin{align*}
\text{Size} & \quad O(\ell E) \\
\text{Mops} & \quad O(n \ell E)
\end{align*}
\]

Actually, they "optimize" a bit with another trick: extinction:

when any node

each node keeps track of the max-root it has seen,
\[ \text{if it gets a token from a smaller root, it just swallows it up, which ends that DFS. (cases mops)} \]

Dallagio get an algorithm for DFS with no root, which does

\[ O(n \log n \log E) \text{ mops} \]

Q: Improve the mops complexity for getting a DFS tree with no

initial root?

(Well, we could always construct some other a MST in $O(E + m \log n)$

mops, then run in to elect a leader, then leader starts make DFS con-

\[ E \]
Distributed Consensus

Abstract problem of reaching agreement among processes in a network, when all start initially with their own opinions.

Complications arise when there are failures (proc, link) or timing uncertainties.

Examples:
- **Transactions:** commit or abort
- **Airplanes:**
  - Agree on value of altimeter reading (SIFF)
  - Agree on who should go up or down, in trying to resolve encounters (TCAS)
- **Resource allocation:** Agree who gets priority to obtain a resource to do next database update

In this course, we'll study several versions of the problem:
- Synchronous, asynchronous, and failures
- Link failures, processor failures

We'll see algorithms and impossibility results.

**Synchronous, link failures**

**Inland scenario:**
- Several generals planning attack
  - Want all to agree to attack
  - Each has initial opinion about local readiness
  - Nearby generals can communicate via foot messengers
    - Unreliable, can get lost
    - Must agree
    - Shall attack if possible

(Assume comm graph is connected, undirected; all generals know graph; known u.d. on tree for successful messengers to deliver message.)
Can show no algorithm is possible!

Motivation: Transaction commit

Formal statement

\[ G = (V, E) \] undirected graph

Synchronous model

in 

Each process has input \([0, 1]\)

\[ \text{don't attack} \]
\[ \text{attack} \]

Any number of messages can be lost (see model, ch. 2)

all must eventually set their decision output was to 0 or 1.

In practice, of course, would need it by some time.)

Conditions:

Agreement: No 2 peers decide differently

Validity: 1. If all start with 0 then 0 is the only possible decision value.

2. If all start with 1 and all msgs are delivered then 1 is the only possible decision value.

Variations on the validity condition:

E.g., stronger: If anyone starts with 0 then 0 is the only possible decision value.

(Makes sense for transaction commit.)

In practice, for designing algorithms, we'd like to get the strongest properties. But there I'm going to show an impossibility result, so it's better to make the statement weak.
Proof: By contradiction, suppose we have a solution A (a process for each $i \in \{1, 2\}$)

Assume wlog that both proc send msgs at every round (can always make them send dummy msgs.)

The proof illustrates the limitations of local knowledge...

Consider $a$: Exes where both start with 1, all msgs delivered.
By termination cond. Both eventually decide, say by $r$ rounds.
By validity: Both decide on 1.

$x_1$: Same as $a$, but all msgs after $nd_r$ lost.
But they're already decided by end of round $r$, both decide 1.
(Different states may occur after $nd_{r+1}$ etc., but too late to change decision.)

Now continue, constructing a series of executions, each indistinguishable from the previous one w.r.t. some process (recall def).
We'll follow that all these execs have same decision value.

$x_2$: Same as $x_1$, but fail last $(nd_r)$ msg from 1 to 2.

Claim $x_1 \not\sim x_2$.
Looks same to proc. 1.
Since proc 1 decides 1 in $x_1$, it also decides 1 in $x_2$.
By term, proc 2 also decides in $x_2$.
By agreement, proc 2 decides 1 in $x_2$. 
Note that \( p_2 \) decides 1 in \( x_2 \) at some point. Its behavior might have changed completely from end of \( x_2 \) onward (\textit{e.g., decision could occur later. But still must decide 1}).

- \( x_3 \): Same as \( x_2 \) but will red \( x \) msg \( x \rightarrow 1 \) (also).
- \( x_2 \not\sim x_3 \), so proc \( p_2 \) decides 1 in \( x_3 \)
  
  So by term + agreement, proc \( p_1 \) decides 1 in \( x_3 \)

- \( x_4 \)

- \( x_{2n+1} \): Both start with 1, no msgs sent, both decide 1.

Now work toward a conflict by changing the initial values.

- \( x_1 \): \( p_1 \) has 1, \( p_2 \) has 0, no msgs.
  
  \( x_{2n+1} \not\sim x_1 \), so \( p_1 \) decides 1 in \( x_1 \)

  So also \( p_2 \) decides 1 in \( x_1 \)

- \( x_2^n \): Both start with 0, no msgs.

  \( x_1 \not\sim x_2^n \), so \( p_2 \) decides 1 in \( x_2^n \)

  so \( p_1 \)

  \( \ldots \)

  But \( x_2^n \) contradicts validity.

  \( \times \)
Now let's continue consensus problem, this time in the presence of node (process) failures rather than link/comm. failures.

(Could study both together, but we'll focus on impact of each separately.)

2 main models have been studied: stopping failures and model crashes

Byz. failures: arbitrary

LPS, general scenario, but I'll just present the abstract part here.

Agreement problem:

Input from a set \( V \)

Output from \( V \) by setting decision := \( v \)

Synch model, but now some number, \( k \), of failures.

Stopping case: Can stop at any point, including in middle of sending mosp at a round.

Subset of mosp are put in channels (or subset)

Could stop after sending + before doing state transition.

Properties:
Agreement: No 2 processes decide on ± values.

Validity: If all start with v then v is the only possible decision.

Termination: All nonfaulty processes eventually decide.

Byzantine case

"Arbitrary behavior" often described as "malicious" but really just means we have no assumptions restricting failures

- 3 faulty processes can start in arbitrary state
  - send arbitrary messages
  - perform arbitrary transitions

However: Can only affect its own state & its own outgoing msgs. (think of replacing the real process with another one.) Can't affect anything else in the network.

Agreement: No 2 nonfaulty processes decide on ± values.

Validity: If all nonfaulty proc's start with v, then v is the only possible decision for a nonfaulty proc.

Termination: As before

Note: Alternative, perhaps by now more common, validity condition is to say every (nonfaulty) proc's decision value must be initial value of some process.

Stronger: Used esp. for stopping case.

Stronger: Will use this later (k-agreement)