Class 3
6.852
Fall, 2001

Bring: chalk
HW 2

Reading: 4.2-4.4, ~~5.1~~
(Attiya-Welch Ch. 2.4, 2.5)
5.1

Next: Ch. 6 (all)

Last time: Started after showing a lower bd on comm for leader el. in ring,
Basic algs in general synchronous networks.

1. Simple leader election by flooding
~~[illegible]~~ + used to illustrate a basic simulation relation

2. Started BFS

Today: Continue - BFS, shortest paths, MST, DFS } Setting up basic comm structures in (synch) networks.

All these can be extended to asynch - will see shortly.

~~and start distributed~~

There are all analogues of standard problems ~~[illegible]~~ studied for sequential setting.
(& the algorithms have a lot in common too).

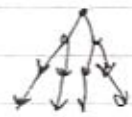
Then start distributed consensus.

Procs have UIDs, but no knowledge of size or diam. of network.

Breadth-first Search

issue: Strongly connected digraph with distinguished source node i_0.

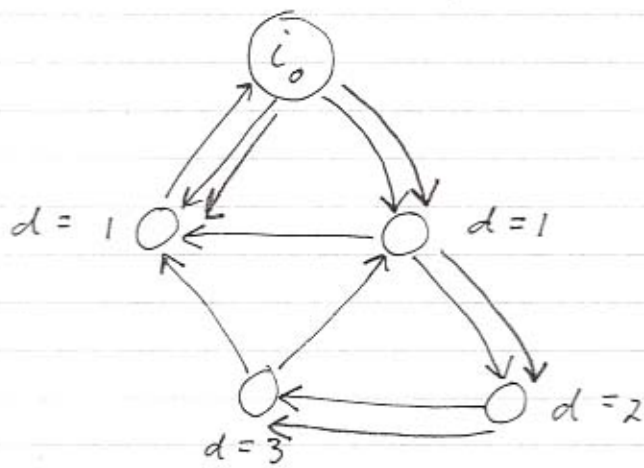
Want to set up a B-F spanning tree for digraph, starting at root i_0; branches are directed paths in the given graph.



Spanning: contains every node

B-F: each node at dist d appears at depth d in the tree
↑ from i_0 in the digraph

Ex:



Output: Each node other than i_0 should set a parent variable to indicate the node that is its parent in the tree. (named by a UID)

Output tree representation is distributed.

Informal descr: Mark nodes that are already incorporated into the tree. ~~Initial~~ ("mark" flag in state).
Initially, only i_0 is marked.

Rd 1: Proc. i_0 sends out "search" msgs to all nbrs.

at any round: If unmarked ~~msg~~ process receives "search" msg, marks self, chooses one of the processes that sent msg as its parent.

at next round (only) sends "search" msg to ~~all~~ all its neighbors.

So, state needs to contain flag to say when to send, etc.

Pf: LTTR B-F tree because created synchronously.

Complexity: Time $\leq$ diam to have everyone set "parent"

actually, it's $\max_j \text{dist}(i_0, j)$

for the particular i_0

Msgs $\leq |E|$

(one per directed edge)

Efficient.

(Can reduce slightly: processes needn't "reflect" msg back to any process that sent it a msg.)

Child pointers: Often, for applications, want process to know its children in tree, not just its parent.

Can accomplish this by having each node respond to its parent, telling it that it has been chosen.

Easy if bidirectional comm on all edges

(If not, must send responses indirectly, on whatever paths connect.

But this is complicated + expensive – initially, don't know which paths these are. So have to do something like send on all paths (flood)

And have to tell other nodes that they have not been chosen as parents, if node is supposed to ever figure out that it knows its entire set of children.)

~~So arrived here this~~

Termination: Suppose i_0 wants to know when the whole tree is completely set up?

Let each "search" msg get a response, "parent" or "non-parent"

after any process has received responses ~~from~~ to all its "search" msgs, it knows who its children are, + knows they are all marked.

So, starting from leaves of the tree (these discover who they are), fan in "completion" msgs to i_0. (convergecast)

Node can send completion msg after

(a) It has received responses ~~f~~ to all its search msgs (so it knows its children)

+ (b) It has received completion msgs from all its children.

Complexity: (assuming bidir. comm.)

Time: $O(\text{diam})$

Msgs: $O(|E|)$ Including responses to search msgs. (Completion msgs just $O(n)$).

Applications of BFS

1. Broadcast

Can broadcast a msg while setting up the tree (piggyback)

or

First produce tree, with child pointers, then use the tree for broadcasting.

Can reuse the tree for many bcasts; each takes only

$$\begin{cases} O(n) \text{ msgs} \\ O(\text{diam}) \text{ time} \end{cases}$$

Now assume bidirectional comm.

2. Global computation, given a BF tree

Involves collecting info from everyone in the network, possibly
the root
using it in some computation.

E.g. sum values held by all procs.

Convergecast, combining at each stage (start at leaves).

time: $O(\text{diam})$

msgs: $O(n)$

---

3. Leader election, with unknown $\begin{cases} n \\ \text{diam} \end{cases}$

All procs initiate BFS in parallel, use tree to compute max uid in network (convergecast)

Then process with this max uid declares self leader.

time: $O(\text{diam})$

msgs: $O(n \cdot |E|)$ msgs

for each BFS

~~each~~ number of different BFS execs

(actually, $O(\text{diam}\,|E|)$ is an upper bound, since there are only diam rds, $\leq$ one msg per edge per rd. But there are ~~[illegible]~~ big msgs, obtained by combining a lot of smaller primitive msgs)

Q: Cut down # of msgs by ~~[illegible]~~ discontinuing participation in BFS's for sources that aren't the max one you know about. ??

4. Computing diameter

E.g., all i can do BFS in parallel, use tree (convergecast) to determine max distance from i to any node. (height of tree)

Then all reuse BF trees for global comp. to determine max of all these max distances

Complexity as for 3. above.

Shortest Paths

also motivated by communication - setting up structure for shortest latency communication ~~that~~ where edges have associated "costs"

Generalization of BFS

Digraph, strongly connected, now with weights $\in R^{\geq 0}$ on edges

Source i_0

Want to determine shortest path (= path with smallest total weight) from i_0 to all nodes.

a tree again

Every node should output $\begin{cases} \text{parent in tree} \\ \text{distance from } i_0 \text{ in sp. tree.} \end{cases}$ (total of weights)

Node knows weights of incident edges.

Assume nodes know n. (again for termination)

Bellman-Ford algorithm:

Each i remembers $\begin{cases} \text{dist, shortest dist it knows (so far) from } i_0 \\ \text{parent, its parent in some path with total wt} \\ \quad = \text{dist.} \end{cases}$ (+ counts rounds)

Initially:

i_0	others
dist = 0	dist = ∞
parent = null	parent = null

At each rd, each process sends its dist to all its out-nbrs.

Then each process i updates its dist with a relaxation step, where it does:

$$\text{dist} := \min(\text{dist}, \underbrace{\text{all values}}_{\text{for all in-nbrs } j}(\underbrace{\text{dist}_j}_{\text{as received in new msgs}} + \underbrace{\text{wt}_{j,i}}_{\text{known}}))$$

If dist changes, parent also updated.

Done after $n-1$ rounds.

After $n-1$ rds, $\begin{cases} \text{dist contains shortest dist} \\ \text{parent contains parent in s-p tree.} \end{cases}$

~~Complexity: $\begin{cases} \text{time } \le n-1 \\ \text{msgs} \le (n-1)\,|E| \end{cases}$~~

Correctness:

Key invariant says that after r rounds:
Every process i has its dist + parent corresponding to a shortest path among paths from i_0 to i that consist of $\le r$ edges.

(If no such, then dist $= \infty$, parent = null.)

Proof: Induction. ~~Key is that a shortest path to i on $\le r$ path must ?~~

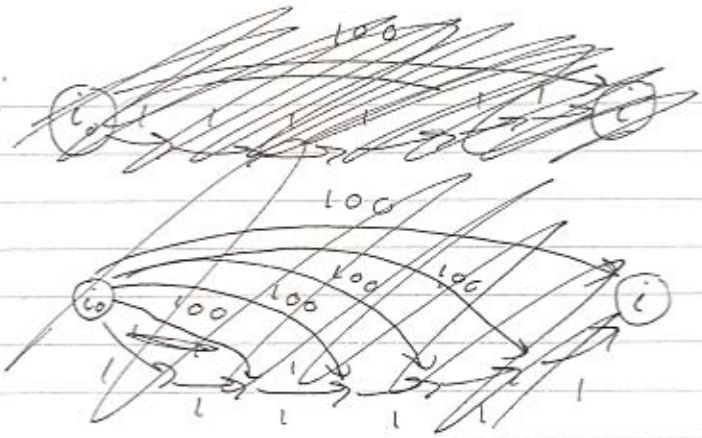
Key is that, after rd $r-1$, each pred j of i on any path from i_0 had its best dist, taken over paths of $\le r-1$ edges.
And shortest path to i on $\le r$ path, when restricted to its pred j on this path, yields a shortest path to j on $\le r-1$ path.

Complexity: $\begin{cases} \text{time } n-1 \\ \text{msgs} \le (n-1)\,|E| \end{cases}$

Worse than BFS, which has ~~the~~ $\begin{cases} \text{time diam} \\ \text{msgs } |E| \end{cases}$

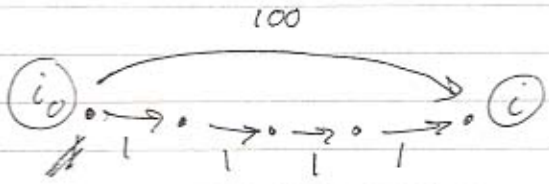
You might think this algorithm's time should depend on diam, not n.
But not so, because "shortest path" can actually be over a path with more links:

Ex:



Shortest path on bottom.

Time Complexity depends on n, not diam



~~(+ all other~~

+ make graph complete, with all other edges labeled by 100.

Shortest path = 4

though diam = 1.

Later will revisit Bellman-Ford in asynch networks (like old Arpanet)

Gets even more expensive.

Minimum Spanning Tree

Another basic tree structure, this one minimizing total wt of all edges in the tree

(Total cost, e.g. money to charge, for all communications.)

Show synchronous algorithm here, mainly as a stepping-stone to a more complicated asynch algorithm we'll see later.

Assume:
- $G = (V, E)$ undirected $\rightleftarrows$ on each edge
- weights, known by adjacent procs (e.g. nonneg. reals)
- UIDs,
- known n (actually, the asynch algorithm doesn't need this. But the simplified version uses it (technicality).

Each node to decide which adjacent edges are + aren't in the tree.

The theory (same general theory as for sequential MST algs)

Spanning tree (of G): a tree (acyclic graph) connecting all nodes, with edges selected from G.

Spanning forest: a collection of trees spanning all the nodes (again, edges selected from G).

All MST algs, both sequential + distributed, use special cases of a general strategy.

Start with trivial spanning forest of n separate nodes.

Repeatedly: Merge components along edges that connect components, until all are connected.

The trick is to make sure that we only merge along edges that are min weight outgoing edges of some component.

Justification for why this is OK: (Same argument as in sequential case):

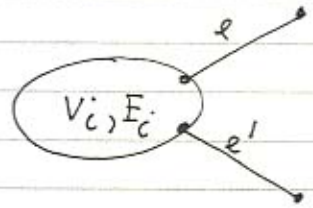
<u>Lemma 1</u>: Let (V_i, E_i), $1 \le i \le k$, be sp. forest, $k > 1$
Fix any i.
Let e be an edge of smallest weight among set of edges with ~~one~~ exactly one endpoint in V_i.
Then $\exists$ sp. tree for G that

(a) includes $\bigcup_j E_j$

(b) includes e

(c) has min weight among all sp. trees for G that include $\bigcup_j E_j$

<u>Pf</u>: Like the standard arg. for the sequential theory.
Suppose false, ~~$\exists T$ containing $\bigcup_j E_j$, not containing~~
~~let T be the a sp. tree with min wt among those incl. $\bigcup_j E_j$.~~
~~Fix & assume~~
that there's some T that is a sp. tree for G ~~&~~ & includes $\bigcup_j E_j$, that doesn't contain e, & ~~in fact~~ has weight strictly smaller than the weight of any sp. tree that includes $\bigcup_j E_j$ & e.
Now construct graph T' (not a tree) by <u>adding e to T.</u>
<u>Contains cycle</u>, which must contain another edge outgoing from the same component (V_i, E_i).



$wt(e') \ge wt(e)$ by choice of e.
Let $T'' = T' - \{e'\}$
T'' is sp. tree, & has $wt \le wt(T)$.
contains $\bigcup_j E_j$ and e
Contradicts choice of T.

$\boxtimes$

General strategy for MST:

Repeatedly: [Choose some component i
Choose any least-weight outgoing edge from i.
+ add (merging 2 components)

Sequential MST algs do special cases of this:

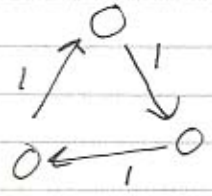
Dijkstra: Grows 1 component by adding 1 more node each time
(Prim)

Kruskal: Always add min wt edge globally.

Distributed version?

Would like to choose edges concurrently for several components.

Problem:



Can get cycles.

Avoid by assuming all weights are distinct (can achieve same effect by breaking ties with VIDs).

Lemma 2: If all wts are distinct, then there is exactly one MST.

Pf: another cycle-style argument, similar to the previous one. ☒

This allows a concurrent strategy:

At each stage, suppose (inductively) that the forest produced so far is part of the unique MST.

Then suppose several components choose least-cost outgoing edges concurrently.

Claim each of these is in the (unique) MST, by Lemma 1.

So all are, + OK to add them all.

No cycles, since all are in the MST.

Distributed algorithm:
assumes edge wts distinct.

Build fast in "levels" (phases).
For this presentation, for the synchronous model, the levels will be ~~created synchronously~~ kept synchronized throughout the network.
(asynchronous version lets them get out of synch)

For each k, level k components form a spanning forest, which is a subgraph of the unique MST.

Moreover, each level k component has $\begin{cases} \geq 2^k \text{ nodes} \\ \text{a distinguished leader node.} \end{cases}$

In this synchronous description, procs allow a fixed number ($O(n)$) of rounds for each level.
This requires that they know n. (This funny restriction is removed later in the asynchronous version)

~~When~~
~~at the end of~~
When we have level k components, each process knows:
$\begin{cases} \text{the UID of the leader - used as ID for the component} \\ \text{which of its adjacent edges are in the ~~component tree~~ tree for its component.} \end{cases}$

Level 0: Single nodes
Level $k \to$ level $k+1$:
Each component conducts a search (along tree edges) for MWOE of component:
Leader bcasts requests along tree edges (as before)
Each process finds the ~~least wt~~ min wt edge among its incident edges that are outgoing from the component.

Does this by sending test messages along non-tree edges, ~~that are outgoing from component~~ asking if other end is in same component - compare component ids.

Convergecast the min information.

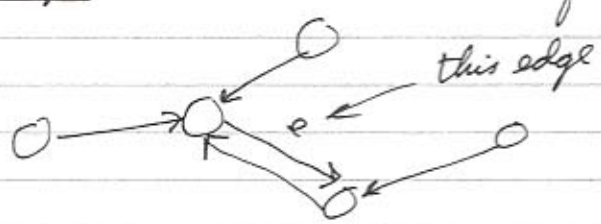
Min obtained by leader is min for entire component.

after all level k components have found MWOEs
(wait suff. many rds, $O(n)$, to be sure)
Need to combine all components using all these MWOEs to form new level $k+1$ fragt.

Need new leaders, etc.

So, leader of each level k component sends msg (along tree) to MWOE node (endpoint in this component), telling it to record that this edge is in the new tree at level $k+1$ (tells other end also).

So choose new leader: In each new component, there's a unique edge e that is MWOE of two level k components:



(Based on graph theory, see p. 68.)
Choose new leader to be one of the endpoints of e, e.g. larger UID.

Note new component has $\geq 2^{k+1}$ nodes.

Terminates: When attempt to find MWOE fails. This means a component has no outgoing edges, which means it's the entire ~~set~~ set of G nodes.
Then leader bcasts msg saying done.

This algorithm designed to work with component levels synchronized.
More difficulties arise when they can get out of synch (we'll see later)
In particular, test messages are supposed to determine if ~~same~~ it's the same component by comparing UIDs.
Important that the node being queried has up-to-date UID info.
(That's why we needed the procs to know n.)

Complexity:

Time: $O(\underbrace{n}_{\text{time per level}} \underbrace{\log n}_{\text{number of levels}})$

(because $\geq 2^k$ nodes in each level k component)

Msgs: $O((\underbrace{n + |E|}) \log n)$ if done naively:

at each level, $\begin{cases} O(n) \text{ msgs sent on tree edges} \\ O(|E|) \text{ for doing all the tests} \end{cases}$
(explore all edges)

Reducing communication:
a major contrib of the asynch algorithm is reducing the comm to $O(n \log n + |E|)$.
I'll show their trick here, in the synch setting:

Idea

Process marks its incident edges as rejected when they're discovered to lead to same component — no need to retest these.

At each level, test candidate edges 1 at a time, in order of increasing weight, until the first one is found that leads outside (or until exhaust candidate edges).
Rejects all those that are found to lead inside.

Msgs: $O(|E| + n \log n)$ total:
$O(n)$ for # of tree edges at each phase, for $O(n \log n)$ total.
Msg for test, accept, + reject :
"different component" "same component"

amortized analysis.
Each edge (directed) has test-reject at most once ~~in each direction~~ for total of $O(|E|)$.

If tested + accepted, might not be MWOE of entire component. So msg be tested again. Can test + accept several times.

However, at most 1 T+A for each node at each level, for total of $\boxed{O(n \log n)}$

Back to this for asynchronous case.