Class 10
6.852
Fall, '01

brig: {chalk
HW 3b

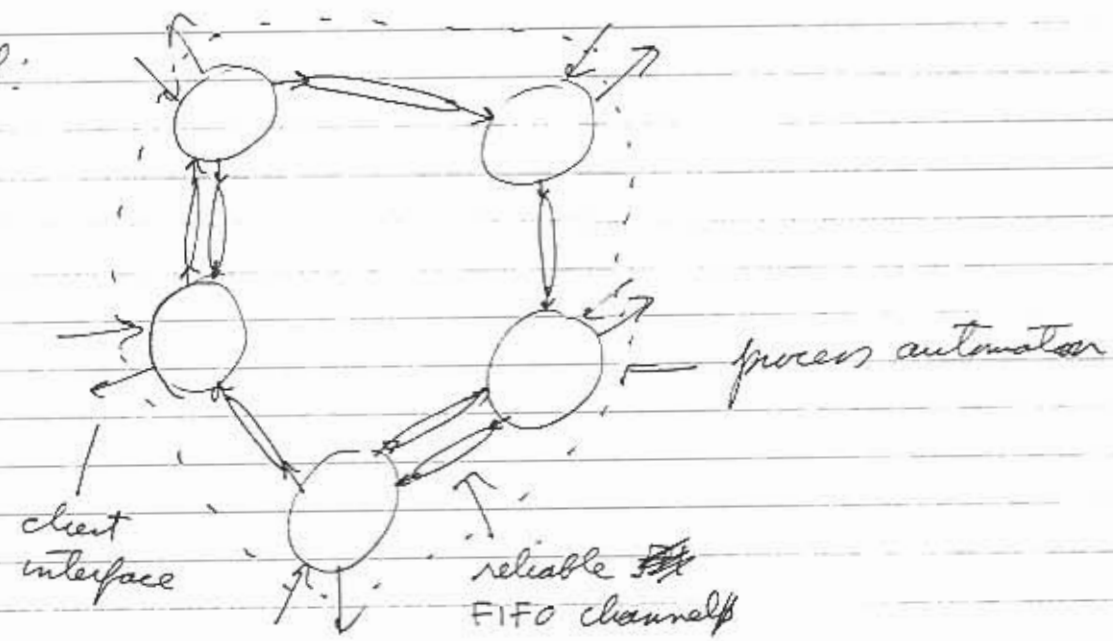
Reading: ~~[illegible]~~ Ch. 15

Next: Ch. 16, then 9, 10 (mutex)

Last time finished up basic asynch model ideas
Then started asynch network algs., by revisiting leader election in ring.

Today continue with {leader election, bcast, ccast,
spanning trees, → MST
in asynch networks.

Recall model:



Digraph with {process aut. at nodes
FIFO channel aut. on directed edges.

Last time showed LCR alg extends to asynch case. → *

This bound takes into acct ~~[illegible]~~ pileups (queuing delays) in channel + send buffers.

~~Still [illegible] at [illegible]~~

Total time still $O(n(l+d))$ – because pileups here simply mean that some UIDs have sped up. (see book)

Note style of measuring time bds: No rds, but using bounds on ~~[illegible]~~ duration ~~[illegible]~~ of time between low-level events.

---

* HS – Similarly, extends to asynch model

Peterson:

- $O(n \log n)$ msgs
- unidir comm
- D on't know n.
- Comparison-based

} hard – this was open Q for a while (HW prob?)

Any process, at any time, is active or relay. Relays just pass msgs on.

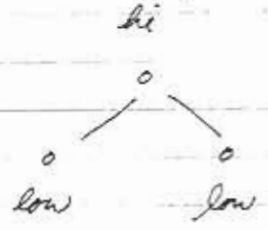
Asynch phases: In each "phase", # of active divided by 2 (at least).

So $\leq \log n$ phases till election.

Phase 1: Send UID 2 steps + collect 2 values.

Compare.

If



then remain active, adopting the UID of the predecessor (the high one).

else go inactive (+ remain inactive)

Claim someone remains active.
But no more ~~that~~ than half of those from the prior phase.

Subsequent phases:
Exactly the same, except now the inactive procs (relays) just pass on.

If ever see immediate pred's id same as yours, elect self leader.
(Know you're the only one left.)

$\boxed{p.484}$ Its over $\begin{cases} \text{advance-phase}_i \\ \text{become-relay}_i \end{cases}$

Easy to see solves.
Easy to see # active shrinks by at least $\frac{1}{2}$ each phase.
Not so obvious, but can show time to election is $O(n(l+d))$.
Technical arg, in book.

A Lower Bd:
On comm. complexity, $\Omega(n \log n)$, for election in asynch ring.
Follows from the synch l.b., in the special case of comparison-based alg.
Actually, there's also a synchronous l.b. for general algs, but it's pretty hard (starred section).
Here, I'll give an alternative elementary proof that depends heavily on the ~~nondeterminism~~ asynchrony.
Illustrates very well the problems that ~~can't~~ are introduced by asynchrony.

assume <u>ring size unknown</u> (alg. works in arb. size rings).

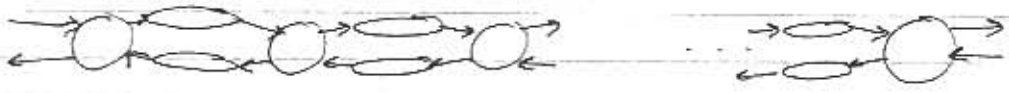
<u>UIDs</u>, infinite space of UIDs

all procs identical except for UIDs

Bidir. comm.

Consider combinations of procs to form <u>rings</u> as usual.

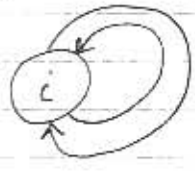
but also <u>lines</u>, where nothing is connected to the ends + no input arrives there:



<u>Lemma 1</u>: $\exists$ <u>inf set of process ant</u>, each of which can <u>send at least 1 msg without first receiving one</u>. (in some exec)

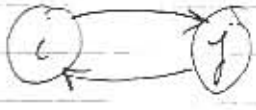
<u>Pf</u>: If not, $\exists$ 2 procs. i + j, neither can send msg without first receiving one.

Consider



must elect self, no msgs sent

"



both elect, contrad.

$\boxtimes$

<u>Lemma 2 (key)</u>: For every $r \geq 0$, $\exists$ inf. coll $\mathcal{L}_r$ of pairwise disjoint <u>lines</u>, s.t. for each $L \in \mathcal{L}_r$,

$$|L| = 2^r$$

$$\underbrace{c(L)}_{} \geq r\,2^{r-2}$$

defined to be the max (sup) # of msgs that can be sent in a single (input-free) exec. of L

Pf: Induction on r.

Base: $r=0$, $r=1$, LTTR

Inductive: Show for $r \geq 2$, assuming for $r-1$.

By ind. assumption, $\exists$ inf. coll. $\mathcal{L}_{r-1}$, all in $\mathcal{L}_{r-1}$ have

$$\begin{cases} |L| = 2^{r-1} \\ c(L) \geq (r-1)\, 2^{r-3} \end{cases}$$

Choose any 3, L, M, N, from $\mathcal{L}_{r-1}$

Claim at least one of the 6 combins $\left\{\begin{matrix} LM, ML \\ LN, NL \\ MN, NM \end{matrix}\right\}$ has $c(L) \geq r \cdot 2^{r-2}$.

That's sufficient (...)

Pf of claim: Suppose not, that is, can't force any of these 6 combins. to send $\geq \frac{n}{4} \log n$ (using $n = 2^r$)

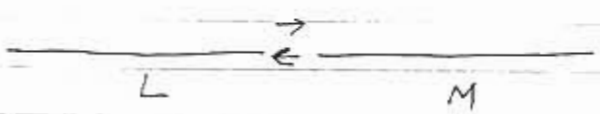
By ind. hyp, $\exists\, \alpha_L$ (finite, input-free) for L

$$c(\alpha_L) \geq (r-1)\, 2^{r-3} = \frac{n}{8} \log\left(\frac{n}{2}\right)$$

Same for α_M, α_N

assume WLOG all end in "quiescent" (silent) states, where no further messages can be sent without further input. (If not, extend...)

Now consider line



& construct new execution α_{LM}:

Run α_L, α_M, delaying msgs output along boundary from being delivered until 2 pieces quiesce.

Then let those msgs be delivered, + continue to run LM to a quiescent state.

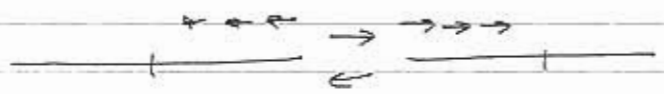
Number of msgs: $\exists$ a $\alpha_L + \alpha_M$, get $\geq 2\left(\frac{n}{8}\right)\log\left(\frac{n}{2}\right)$

$$= \frac{n}{4}(\log n - 1)$$

Claim in running to quiescence, number of add'l msgs $< \frac{n}{4}$.
(Since otherwise would total $\geq \frac{n}{4}\log n$.)

Thus, msgs can ~~at~~ propagate from boundary only to distance $< \frac{n}{4}$, before quiescence.

This says info about the junction doesn't reach the midpoint of either L or M, before quiescence.

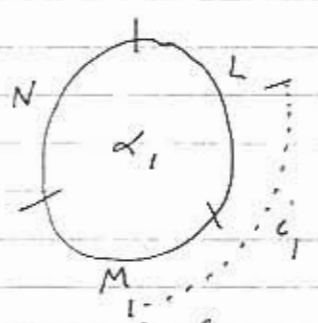


Same for ML, LN, NL, ...

Now get contrad by considering some rings:

(L → R on ~~ring~~ line corresponds to clockwise on ring)

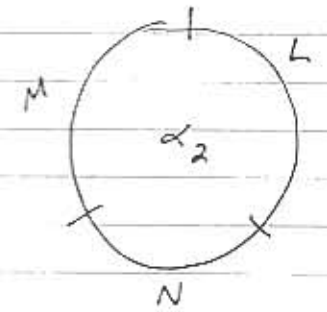
LMN:



Run $\alpha_L, \alpha_M, \alpha_N$ till quiescent
Then quiesce starting from boundaries.
No interference.

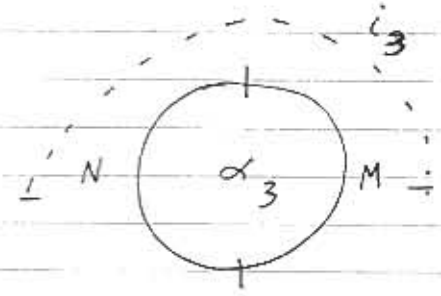
Must elect a leader, WLOG say i_1 ~~at~~ between midpts of L + M

LNM



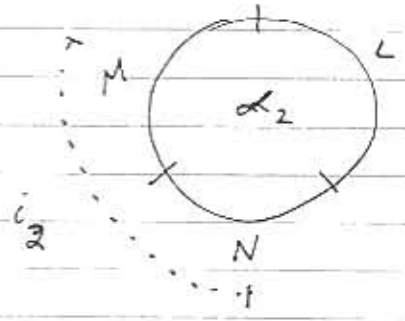
Some i_2 elected here.
Potentially anywhere.

MN

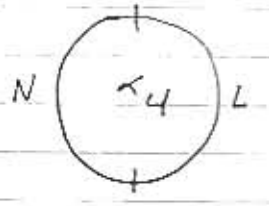


i_3 elected
Claim must be in ~~the~~ upper half.
Because if in lower half,
since α_3 + α_1 are
indistinguishable to i_3,
i_3 would also ~~be~~ get elected in α_1,
which would yield two leaders in
α_1. Contrad.

Then also i_3 elected in α_2, that is, $i_2 = i_3$



Now consider LN.



Must elect some i_4.
Where can it be?
If in upper half, also in α_1,
contrad.
If in lower half, also in α_2,
contrad.

Leader Election in Arbitrary Networks (skip)

Can get asynch version of alg. presented for synch case, but not very interesting. Can simulate ~~asy~~ synch rounds in asynch setting, with counters. Have to know diam to know when to terminate.

Network Searching

Fundamental tasks: Constructing spanning trees, ~~for~~ to use for communication, e.g. bcast & ccast.

Start with simple task of setting up some (arbitrary) spanning tree rooted at a given i_0.

Assume:
- Undirected, connected graph
- Procs don't know size or diameter
- UIDs, ~~[illegible]~~ can identify in & out edges that attach to same nbr. (don't actually need the UIDs)
- Each process should report, with parent output action, name of parent in tree.

Starting point ~~is~~ is the synch BFS algorithm.

Recall: Sets up a BF spanning tree, using search msgs. Each process chooses sender of first search it receives as parent.

Can run asynchronously, still gives spanning tree, but not necessarily B-F.

Go over code, $\boxed{\text{pp. } 496\text{-}7}$:
- search msgs
- only send in response to first search msg received
- tasks

Complexity: msgs: $O(|E|)$

time: all done in $(\text{diam})(l+d)+l$

But <u>note anomaly</u>: Paths produced could be longer than diam.
Because msgs ~~can~~ might <u>travel faster on longer paths than shorter</u>,
in asynch network.

Discuss

<u>Msg bcast</u>: Piggback on search msg

<u>Child pointers</u>: Easy to augment with responses to search msgs, determine children

<u>Using precomputed tree w/ child pointers for repeated bcasts + ccasts</u>:
Now note the <u>anomaly comes into play</u>.
Can take <u>$O(h(l+d))$ to bcast</u>, where h = height of the constructed tree. <u>Can be bigger than diam</u>
<u>Worst case n</u>

<u>Asynch Bcast + Ack</u>

Can also construct the sp. tree while using it to bcast + collect acks.
(To tell when the bcast is done, or to ccast an answer.)

Go over code $\boxed{p.\ 499\text{-}500}$

Report; lets leaves initiate ccast

<u>Complexity</u>: msgs: $O(|E|)$
time: $O(n(l+d))$, because of anomaly of ccast.

<u>Breadth-first spanning trees</u>

$G = (V, E)$, conn., undir., i_0
Procs don't know n or diam
~~Ho~~ UIDs as above
Each to report parent

} as before

~~Use sp. tree to elect leader?~~
~~network~~

*

In synch setting, Synch BFS worked to give BF spanning tree.
But when run asynchronously (as above), just produces some sp. tree, not rec. BF.

Can modify so processes correct erroneous info - when hear better dist, correct it. In this case, must inform other nbrs about the change, so they can also correct.

(Relaxation alg.)

See over code, $\boxed{p.502}$

Eventually stabilizes to BF spanning tree.
But no one knows when it's done, so can't produce parent outputs.

Can augment with acks for all search msgs, convergecast back to source i_0 to say done everywhere.

But this is tricky: Tree grows & shrinks, same process may need to participate many times. (bookkeeping needed)
But when acks finally all ccast back to source i_0, really done.
(Then i_0 can bcast signal to tell everyone else they're done.

Layered BFS

All this asynchrony leads to lots of corrections, which leads to extra communication.

~~It may be be~~

An idea: Slow down some of the computation, trying to keep the levels of the tree more synchronized.
~~Avoids tree~~ Reduces ~~as~~ needed corrections, which ~~lds~~ in turn reduces communication.

* Elect leader in unknown network: (with no sp. tree)
all initiate asynch-bcast-ack, ~~so~~ determine max, & max elects self leader

With a sp. tree:

* ~~the~~ Using <u>arb. sp. tree to elect leader</u>: convergecast, as in synch setting.

For example, could develop alternative BFS alg. that proceeds in synchronized layers

(unknown n, diam; parent announcements)

Each layer k sets up all the nodes at depth k in the tree.

Synchronize in between setting up each pair of layers, by comm. with i_0.

Phase 1: i_0 sends search msgs to nbrs, ~~[illegible]~~

Nbrs mark dist. as 1, send acks

i_0 gets acks.

Phase k+1: Assume phases 1, ..., k constructed. Each node at depth $\leq k$ knows parent, & also those at depth 1, ..., k-1 know children

Then i_0 bcasts a newphase msg along the tree edges, to depth k processes (current leaves).

Each of these sends out search msgs to all except their parent.

When non-i_0 process receives 1st search msg, makes sender its parent & sends positive ack.

Sends neg acks to all subsequent search msgs.

When ~~[illegible]~~ depth k process has acks for all its search msgs, calls the positive acks its children.

Then ~~[illegible]~~ depth k procs convergecast back to i_0, to say they're done.

(& also a bit saying if anything new was found).

Terminate when nothing new is found.

Produces BF tree.

~~Want to formal Cost~~

not modeling congestion delays

Complexity: Simplified: $\begin{cases} \text{N neglect local comp. time } l \\ \text{Assume every msg in channel delivered in time } \leq d \end{cases}$

Layered: msgs: $O(|E| + n \cdot diam)$

every edge explored at most twice, once in each direction

tree edges can be explored repeatedly, once per phase ($\leq$ diam phases)

time: $O(diam^2 d)$

because of diam layers

Compare with asynch BFS:

msgs: Much worse.

Even ignoring the acks for termination, best bd I have is

$O(n\,|E|)$

bd on # revised estimates

msgs sent as a result of each revision

Why: n is longest distance estimate anyone could get

time: better, $O(diam \cdot d)$

no back + forth

In book, you will also see a hybrid alg. with intermediate performance.

Several layers constructed in each phase.

Within each phase, construct all layers asynchronously.

LTTR

Shortest Paths

Same assumptions as BFS, plus locally-known edge weights

Now want to output shortest distance + parent.

Synch case, recall: Bellman-Ford

Relaxation, corrects estimates

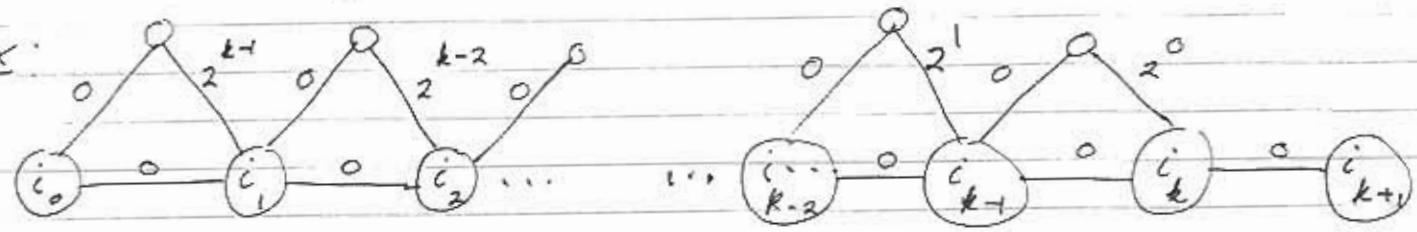
Can run asynchronously $\boxed{p.\ 507}$

Used in ArpaNet in 70s

Can augment with ccast as for BFS, for termination..

But complexity is <u>very bad</u> (in worst case)

<u>Ex</u>:



Succ. decreasing powers of 2

The <u>possible dist. estimates i_k can take on</u> are exactly the numbers

$2^k - 1,\ 2^k - 2,\ \ldots,\ 0.$

Moreover, can force i_k to take these on in sequence! How:

<u>Upper paths fastest</u>: $2^k - 1$

Then, <u>last lower msg arrives at i_k</u>, reduces i_k's estimate,

$2^k - 2$

Then next lower msg $i_{k-2} \to i_{k-1}$ arrives, i_{k-1} reduces estimate by 2, resends search, arrives at i_k first on <u>upper</u> link,

$2^k - 3$

Etc. Count down in binary.

Suppose all this happens fast - then can get queue of 2^k msgs in $C_{i_k, i_{k+1}}$

Exponentially many msgs, so (counting congestion) can take <u>exp. time</u> to arrive at i_{k+1} (exp. in n)

<u>Moral</u>: Unrestrained asynchrony can mess things up.