Class 6
6.852
Fall, 01

bring: chalk

(skim 7.1.3)

Reading: 6.3-6.6, ~~Ch. 7 (skim)~~ Start Ch. 7: 7.1 ~~7.2 (skim)~~

Next: ~~Ch. 7~~ ~~7.2 (skim)~~, 7.3-the → Ch. 8
7.2 (skim), Asynch dist. computing.

Last time: Distributed agreement with stopping failures

Simple algs - { send whole set
send first 2 elts
EIG

all take $f+1$ rds worst-case

Lower bd: $f+1$ rds.

Proved by (assuming f-rd alg.) constructing a chain of execs, each with $\leq f$ failures, 1st with dec. value 0, last 1, + any consec. pair looks same to some nonfaulty process.

(or, look same to all-but-one process)

Today: On to Byzantine (worst case failures)

Recall problem: ~~Byz~~ n-node ~~complete~~ undirected graph
synchronous

Agreement: No 2 nonf procs decide differently.

Val: If all nonf have same init then ~~all nonf~~ that is the only possible decision value

Term: all nonf. procs eventually terminate

<u>Byzantine failures:</u> (worst-case failures)

n-node complete graph

We'll see

<u>EIG algorithm:</u> Exponential comm (in f)
$f+1$ rds
also $n > 3f$

} expensive; comm + # of procs worse than in stopping case.

~~Exam~~

We've seen that the $f+1$ rd bound is inherent.
Turns out the <u>processor bound is inherent too</u>.
Let's get an idea why:

<u>Example:</u> Suppose p_1, p_2 & p_3 solve BA & tolerate 1 failure.
Consider (in this example only) <u>restricted case</u> where

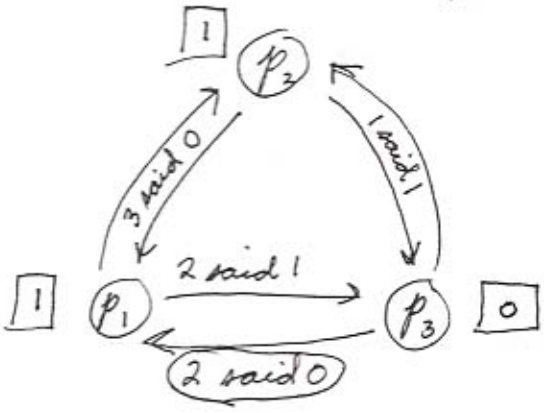
{ algorithm consists of <u>2 rounds</u>
each sends its own val in rd 1 & what the other process told it in rd 2.

(<u>similar to EIG structure</u>)

<u>Exec. α_1:</u> 1 + 2 nonfaulty, init val 1
3 faulty, start with 0

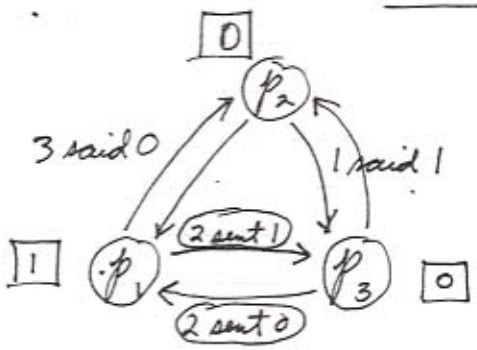
<u>rd 1</u>: all act correct

<u>rd 2</u>: <u>p_3 lies + tells p_1 that p_2 sent 0 in rd 1</u>



Then validity requires that both 1 + 2 decide 1.

α_2: Symmetric: 2 + 3 nonfaulty, start with 0
1 faulty, start with 1
rd 1: all act correct
rd 2: p_1 lies, tells 3 that 2 sent 1

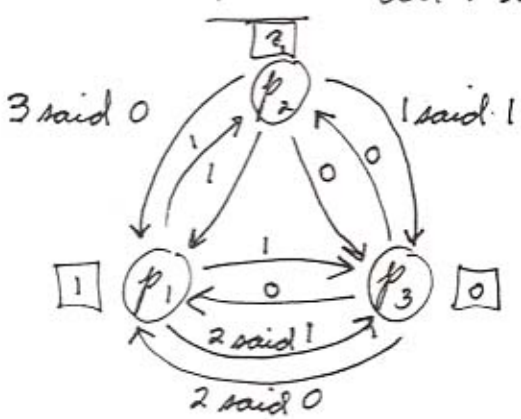


all symmetric, now validity requires that both 2 + 3 decide 0.

Now get a contradiction by mixing up these 2 executions:

α_3: 1 + 3 nonf, start with 1 + 0 resp.
2 faulty
rd 1: 2 { tells 1 its initial value = 1
tells 3 " " " " 0 } inconsistent

rd 2: all relay truthfully.



Notice

$\alpha_3 \overset{1}{\sim} \alpha_1$ So p_1 behaves same in both, decides 1 in α_3

$\alpha_3 \overset{3}{\sim} \alpha_2$ So p_3 decides 0 in α_3.

Contradicts agreement.

This wasn't a formal proof, but could be cast that way.
It shows that no alg. of this form { 3 procs, 2 rds
send + relay
can solve BA with 1 failure. — what can vary is the decision rule - says no rule will work

Note: The correct procs can tell something is wrong

E.g. in α_1, p_1 sees that $\begin{cases} 2 \text{ sends } 1 \\ \text{but} \\ 3 \text{ tells it } 2 \text{ sent } 0 \end{cases}$

So p_1 can tell that either p_2 or p_3 is faulty.

But doesn't know which.
Since alg. has to tolerate 1 fault, has to decide something, but nothing works right in all possible cases.

Can extend the idea of this detailed construction to prove that 3 procs can't solve BA for 1 failure (any form of alg., any number of rds).

Come back to this after showing the EIG algorithm.

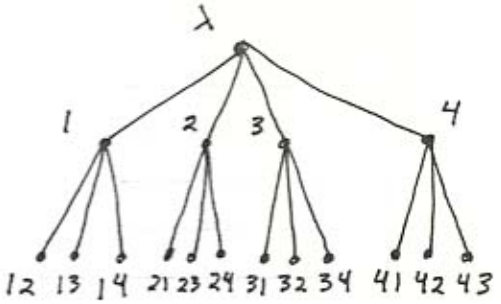
EIG alg for BA

Assumes $n > 3f$ (recall didn't need this for stopping alg)
Same EIG tree, propagate values as before, $f+1$ rds
(but now procs should "throw away" any "ill-formed" msgs + replace with $\perp$)

Now use different decision rule:
Replace all $\perp$ with default value v_0.
Decorate nodes again, this time bottom-up, with new-vals:
Leaf: newval (x) = val (x)
Non-leaf: newval $(x) = \begin{cases} \text{newval of strict majority of children, if exists} \\ v_0 \quad \text{else.} \end{cases}$

Final decision = newval (λ) (root label)

<u>Example</u>: $T_{4,1}$

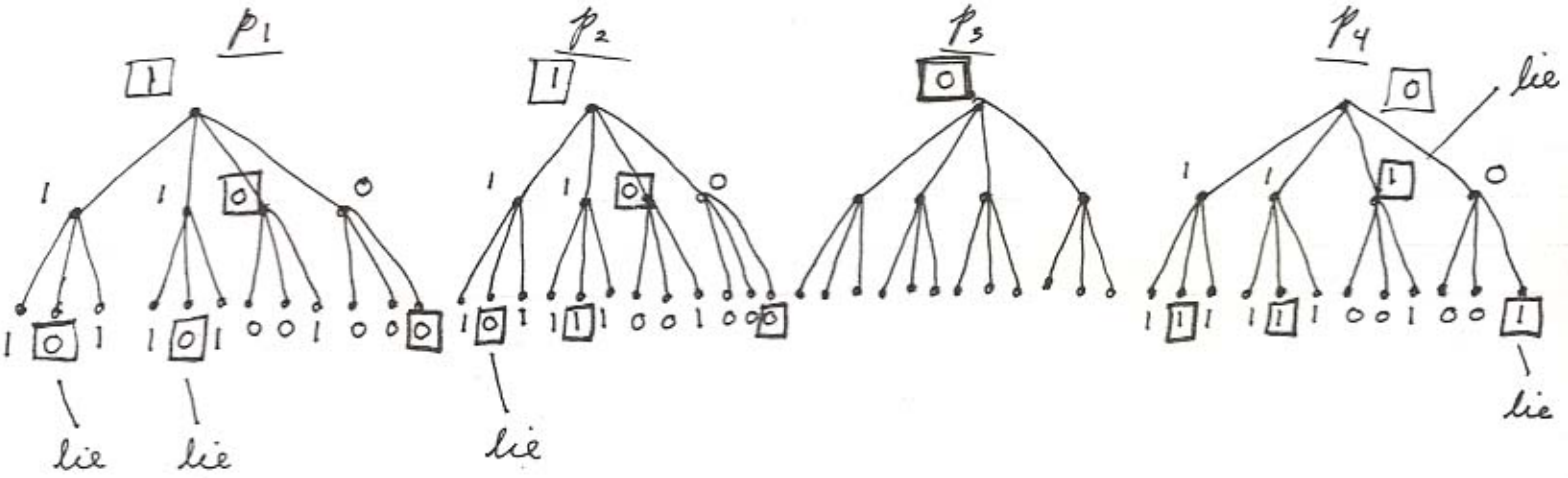


Label with <u>vals</u>, in several (here, 2) rounds

E.g. if start with 1 1 0 0

~~&~~ p_3 faulty,

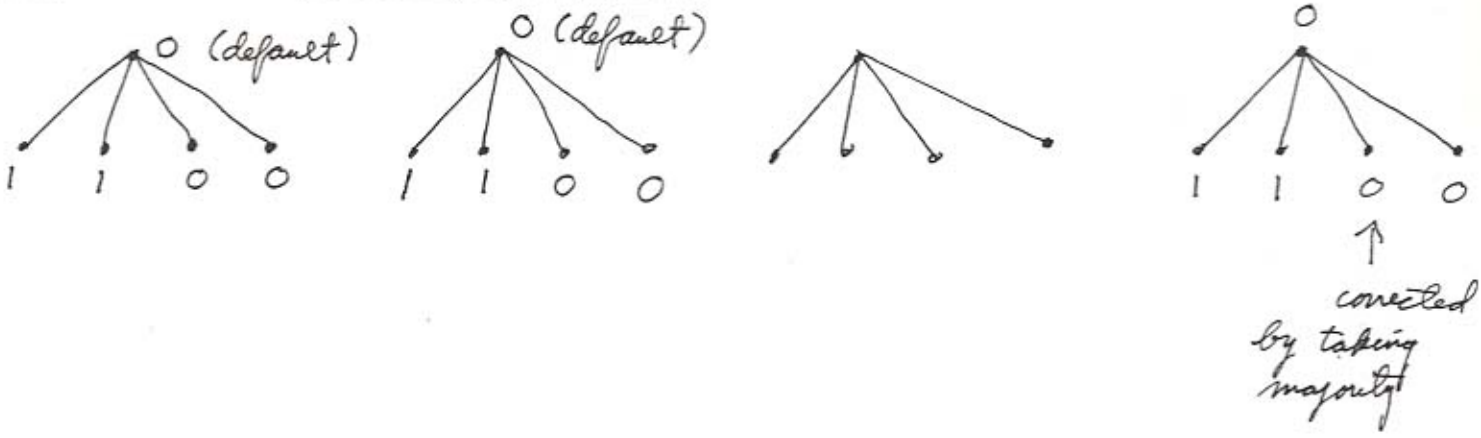
can get:



Then <u>each separately calculates new-val</u>, bottom-up, choosing <u>majority values</u>, default if no strict majority.

Use new-val at top as decision.

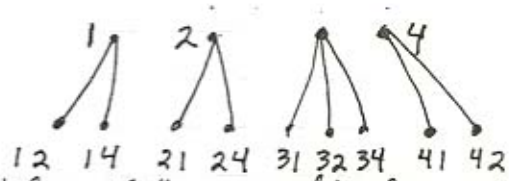
<u>Here</u>: Use <u>$v_0 = 0$ (default)</u>



Correctness: Simple facts saying nonfaulty procs agree about nonf. proc. index:

Lemma 1: If i, j, k nonfaulty, then $val(x)_i = val(x)_j$ for every label x ending in k.

Above, such nodes x are



Pf: k sends same to both, + they "decorate" accordingly. (on the way down)

Lemma 2: If x ends with nonf. process index, then $\exists v$ such that $val(x)_i = newval(x)_i = v$ for all nonfaulty i.

Above, any of the x nodes have val = newval = same everywhere.

Pf: Induction on lengths of tree labels, from leaves up.

Basis: Leaf.

Then Lemma 1 implies all nonfaulty have same $val(x)$, say v.
+ newval = val for leaves.

Inductive: $|x| = r \le f$ ($|x| = f+1$ at the leaves)

Again, Lemma 1 implies all nonfaulty have same $val(x)$, say v.
But need $newval(x)$ same too.
Every nonf. proc. beats same v for x at rd $r+1$, so
$val(xl)_i = v$ for all nonfaulty l and i.
By inductive hypothesis, also $newval(xl)_i = v$ for all nonf l + i.
Claim majority of labels of x's children end with nonf process indices: Counting: # of children $\ge n - f > 3f - f = 2f$
+ only $\le f$ faulty
So majority rule applied by i leads to $newval(x)_i = v$, for all nonfaulty i.

$\boxtimes$

This is already enough to see validity:

Validity: If all begin with v, then nonf. bcast v at rd 1, so $val(j)_i = v$ for all nonf i, j.

By Lemma 2, also $neuval(j)_i = v$ for all nonf. i, j.
Majority rule implies $neuval(\lambda)_i = v$ for all nonf. i.
So i decides v.

⊠

Agreement: Needs a little more work.

Path covering: (of a tree) Subset of nodes containing at least one node on each path from root to leaf:

Ex:

Common node: One for which all nonf. procs have same neuval.
(doesn't necessarily end in nonf proc. index)
label

Lemma 3: ∃ path covering all of whose nodes are common.

Pf: Let C = all nodes with labels of the form xi, i nonfaulty.
Lemma 2 implies these are common.
≤ f failures means these form a path covering
(each path contains f+1 distinct indices, + at least one must be nonfaulty)

⊠

Now show that common nodes "propogate up the tree":

Lemma 4: If there is a common path covering of subtree rooted at x (any node) then x is common.

Pf: Induction, from leaves up.

Base: If x is a leaf, the only p.c. of subtree is x itself.
So x is common, as needed.

Inductive step: $|x| = r \le f$

Suppose $\exists$ common p.c. C of x's subtree.

If $x \in C$ done.

If $x \notin C$, then C includes a p.c. for each top-level subtree (rooted at a child of x)

say xl

By ind. hypothesis, each such xl is common.
So, all children of x are common.
Then def of neural implies x is common.
(all use same data to compute it)

Therefore, λ is common.

Theorem: BA correct (recap)

Pf: Term: Obvious

Validity: already argued

agreement: because λ is common

$\boxtimes$

Complexity: as for EIG, in stopping model.
Plus $n > 3f$ requirement.

Now show why we have to have $n > 3f$ to solve BA.

Number of procs for BA:

Algs require $n > 3f$

Can prove as a lower bd - holds for any graph with n nodes.

But for graphs with low connectivity, may not even be able to tolerate this many failures.

Number of failures that can be tolerated for BA in an undirected graph is completely characterized

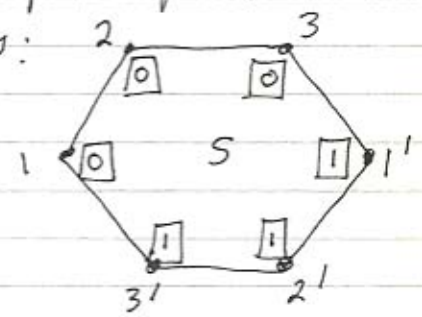
depends on combination of # of nodes + connectivity.

Start by showing 3 procs can't solve BA with 1 fault.

Pf: By contradiction.

Suppose $\exists$ alg. A, procs 1, 2 + 3, solves BA w 1 fault

Construct new system S of 2 copies, + start off with init values as follows:

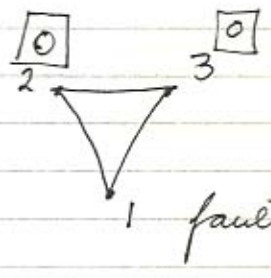


What is S? a synchronous system of some kind, but not required to satisfy any special corr. conds. —— but we can use it in getting a contradiction anyway.

Start with 0's, 1's as above.

Runs, does something.

Consider (2 + 3). Looks like:

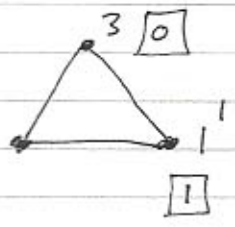


In Δ, must decide 0.

So they do in S also.

Consider $1' + 2'$: Decide 1 in Δ so in S

Consider $3 + 1'$: Must agree in Δ so in S.
But one dec. 0 + the other 1, in S.
Contrad.



but recall what that means

Discuss

Even get the contrad if the original algorithm is allowed to "know n".
This just means the procs in A have 3 written someplace...
+ their validity + agreement + termination conditions are only required to hold if they actually are placed in a Δ.
But that's all we used in the proof!

That's 3 vs 1.
But the same idea extends to 3f vs f for any f.
Can do a similar construction, with f procs playing the role of each 1.
Or, can do a reduction: Show how to transform a 3f vs f solution to a 3 vs 1 solution.
Since we already ~~know~~ know 3 vs 1 doesn't exist, that yields a contradiction.

Pf: If $n = 2$, easy to see impossible, LTTR

$$0 \rightleftarrows 1$$

Each can be faulty, requiring 0, 1 resp.
Or both nonfaulty, requiring =

So assume $3 \le n \le 3f$, soln A

Transform to B, solving 3 vs 1.

Partition A-procs into 3 groups, each $1 \le \quad \le f$ $\quad I_1, I_2, I_3$.

Each of the 3 procs simulates 1 group

Initializes all with his init value.

Each round: Simulate sending of msgs: $\begin{cases} \text{local: just sim} \\ \text{remote: package + send} \end{cases}$

If any simulated process decides, decide same.

Consider exec. of B with ≤ 1 fault.

"Mimics" execution of A with $\le f$ faults.

So agreement, validity, termination must hold for the emulated execution.

Show properties carry over to B's exec.

Termination: If i is nonf. proc of B, then simulates at least 1 (nonf) process of A. That terminates, so i does.

Validity: If all nonfaulty procs of B start with v, then so do all nonf procs of A. Validity for A then implies all nonfaulty decide v, so same in B.

Agreement: If i, j nonf procs of B, simulate only nonf procs of A. Agreement in A implies these all agree, so i + j agree.