Bring: chalk

Reading: 4.1-4.3

(Atiya - Welch Ch. 1+2)

Next: 4.4, 5.1, 6.1-6.2

Last time:
- Introducer model for synchronous networks.
- One introductory problem - leader election - and just in ring network
- User algorithms:
  1. LCR: pass tokens one way, elect max
     Complexity analysis: time \( m \) (\( n \) time, if we want halting)
     \( \frac{m}{n} \)  
     
  2. HTS: successive doubling alg
     
     \( \frac{m}{n} \)

     time? Still \( O(n) \), since last phase is \( O(n) \) +
     other phase bounds are dominated by last (geom)

Today: Consider the msg. complexity again - can it be lowered further?
- Gets us to the first nontrivial lower bound of the course - will take some time (hard).
- Then go over basic network search alg. (sanic, linear)
Theorem 2: Minimum \# of mugs to solve problem?
Can do in \(O(n)\)?

Theorem 1: \(\Omega(n \log n)\) with comparisons only, even if \(\Theta(n)\) known

But: \(O(n)\) possible with more powerful datatypes.
E.g.: If \(n\) known, unidirectional, \(n\) UIDs are positive integers,
allowing arithmetic.

Phases 1, 2, ... each \(n\) rounds
Phase \(i\) devoted to (potential) UID \(k_i\)
If proc with \(k_i\) exists, circulates at beg of phase \(k_i\).
Others who receive shut up permanently: Elect mini

\[ \text{in mugs} \]

Not practical, though, unless UIDs are very small int.
time \[ O(\min n) \]

But now consider comparisons only.
Claim that \(\Omega(n \log n)\) single-hop mugs are needed to break the symmetry.

Compare:
(1) Impossible with no UIDs
(2) With UIDs with comparisons only, it's possible, but indirectly hard

Q: What does it mean for the alg. to use comparisons only?
Roughly, all activity determined by relative order of UIDs.
All start in same state except for \(k\).

The procs manipulate the UIDs only by copying them, reading & receiving
them in mugs, comparing them for \((< = >)\).
Can store, resend, use results of comparisons to decide what to do next.

Decide on whether or not to send msgs to each user what to send whether or not to elect self-leader.

Formalized in 1960's research paper.

**Def:**

1. \((u_1, \ldots, u_k) \equiv (v_1, \ldots, v_k)\) order-equivalent provided 
\[ u_i \leq u_j \iff v_i \leq v_j \]

\[ (1365279) \equiv (279841011) \]

relationships all the same

2. Round of exec is active if at least one (nonnull) msg sent

3. \(k\)-neighbor: of \(i\) in \(R\)

\[ \rightarrow \]

4. Corresponding states \(s \equiv t\) of proc w.r.t.
\[ (u_1, \ldots, u_k) \equiv (v_1, \ldots, v_k) \]

are states that are identical exe. that the UIDs in \(s\) are closer for 
\[ s \equiv t \]

+ \(k\): \(i\) occurs in \(s\) exactly when \(v_i\) occurs in \(t\)

Coresy, messages analogous.

**Key Lemma 2:**

Assume \(A\) is a comparison-based algorithm.

Assume \(i + j\) have order-equivalent \(k\)-neighbors.

Then at any point after at most \(k\) active rounds,

\(i + j\) are in corresponding states w.r.t. the sequences of UIDs in their resp. \(k\)-neighbors.
say pairs with order-eq. k-nblds are indistinguishable at least until enough active rounds have happened. This means (essentially) that info has had an opportunity to propagate to these pairs from outside the given k-nbld.

In Ex. above, i+j correspond to 3 active rounds

Proof: Assume wlog that i ≠ j.
Proof by induction on # of rnds k in exec. (for each r, for all k)

Base: r = 0:
By def of comp-based algo., initial states identical exec. for UID there are corresponding states w.r.t. k-nblds (for any k).

Inductive: Assume for < r rounds, all k.
Show for r rounds, all k.

Fix k, i,j, + suppose i+j have order-eq. k-nblds.
+ also suppose the first r rounds have ≤ k active rounds.
Must show that after r rounds, i+j in compy states w.r.t. their k-nblds.

If neither i nor j receives any msg at round r, then by induction
(for r-1 + k) i+j are in compy states just before rd r w.r.t. their k-nblds.

No new input, so take compy. transitions, end up in compy states.

So assume at least one of i,j receives a msg at rd r.
Then rd r is active.
So, first r-1 rnds must include at most k-1 active rnds.
Cases based on whether $i$ receives msgs from

- both $cis$
- $L$ only
- $R$ only
- neither

just do "both" case here; others similar.

\[ i-1 \quad i \quad i+1 \]

By inductive hyp. \( i+j \) in corresp. states after \( n-1 \) rounds w.r.t. \( k\)-neighds. \( (\leq k \text{ active}, \text{ miss} k-1) \)

Also by ind. hyp.

\( i-1+j-1 \) in corresp. states after \( n-1 \) rounds, w.r.t. \( k-1\)-neighds

\( i+1+j+1 \ldots \)

(These last two hold because these pairs have only weak \( k-1\)-neighds, \( + \leq k-1 \) active rds up to that point.)

So, both \( i-1+j+1 \) send msgs to \( j \) at rd \( n \).

Moreover, the msgs they send correspond w.r.t. the \( k-1\)-neighds, + hence w.r.t. the recipient's \( k\)-neighds.

Since \( i+j \) in corresp. states (w.r.t \( k\)-neighds) after rd \( n \), they receive corresp. msgs (w.r.t \( k\)-neighds) at rd \( n \), hence w.r.t. the recipient's \( k\)-neighds.

\( i \) \& \( j \) in corresp. states (w.r.t \( k\)-neighds) after rd \( n \), they go to corresp. states after rd \( n \).

Other cases: Similar args.

So, we have shown that many active rds are needed to break symmetry if there are large order-equiv. neighds.

Next, show finding with lots of big order-equiv. neighds.

And then show that symmetry causes large comm. complexity.
Ex: $b^* - $reversal ring (dyslexic counting)

<table>
<thead>
<tr>
<th>15</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0111</td>
</tr>
<tr>
<td>11</td>
<td>0111</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
</tr>
<tr>
<td>13</td>
<td>0101</td>
</tr>
<tr>
<td>5</td>
<td>1001</td>
</tr>
<tr>
<td>9</td>
<td>0001</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
</tr>
<tr>
<td>0</td>
<td>1100</td>
</tr>
<tr>
<td>12</td>
<td>0010</td>
</tr>
<tr>
<td>8</td>
<td>1010</td>
</tr>
<tr>
<td>1</td>
<td>0110</td>
</tr>
<tr>
<td>6</td>
<td>1110</td>
</tr>
</tbody>
</table>

Every length $\frac{m}{2}$ seq has 2 eqns

$\frac{m}{4}$ eqn

etc.

Can quantify amount of symmetry:

$c$-symmetric: For every $l$, $\sqrt{m} \leq l \leq m$

a technicality - sounds ignore - doesn't affect anything

+ every seq $s$ of length $l$, $\exists$ at least $[c \cdot \frac{m}{2}]$ eqns seqs (incl. self)

(dyslexic ring is $\frac{1}{2}$-symmetric)

3 act $\mathbb{E}$: $\exists c$ (general count) $\forall m \exists c$-symmetric rings of size $m$

(in paper)

$c$-symmetry implies lots of active rounds before electing a leader.
Lemma 4:

Assume \( A \) is comp.-based alg. in c-symmetric ring. Then \( A \) has more than \( \left\lceil \frac{cm - 2}{4} \right\rceil \) active roles before a leader is elected.

Proof:

Let \( k = \left\lceil \frac{cm - 2}{4} \right\rceil \). Suppose elected leader in \( \leq k \) roles, get a contradiction.

Use Lemma 2.

Consider \( i \)'s k-sublist (length \( 2k + 1 \))

By c-symmetry, must be at least \( \left\lceil \frac{cm}{2k+1} \right\rceil \geq 2 \) equi seqs.

(adv LTTR)

Let \( j \) be multip of another.

Then Lemma 2 implies \( i + j \) remain in correct states throughout the exec., up to \( t \) including the election pt.

So also \( i \) elected.

Now pull it all together:

Theorem 5: Suppose \( A \) is comp.-based alg. that elects leader in (bidin) rings of size \( n \) (\( n \) known).

Then \( \exists \) execution in which \( \Omega (n \log n) \) rings are sent by the time the leader is elected.
Proof: Fix c, get c-symmetric ring.
Define k as above \( \left( \left\lfloor \frac{cm - 2}{4} \right\rfloor \right) \)

Lemma 4 implies \( r \geq k + 1 \) active rounds before leader is elected.

Add up messages for those rounds:

- In the active round: \( m \leq r \leq k + 1 \)
- Technically

Some process sends msg, say \( i \)
Consider \( r - 1 \) index of \( i \) (because \( r \) after first \( r - 1 \) rounds)

By \( c \)-symmetry, \( \exists \) at least \( \frac{cm}{2^{n-1}} \) such msg

Lemma 2 says the midpoints of these are in consensus states w.r.t. the \( r \)-th active round.

Then all send msgs at round \( r \)-th active round.
That is \( \frac{cm}{2^{n-1}} \) messages.

Total \# at least (approx) \( \leq \frac{cm}{2^{n-1}} \)

\( \approx \frac{cm}{2^m} \)

\( \approx \frac{m}{\log m} \)

(\( \text{ harmonic series, sum it using an integral approximation. } \)
Algorithms in Asynchronous Networks

Not just rings.

For basic tasks, like broadcasting messages & getting responses. Setting up communication structures.

Just algorithms, no laws, structural here.

These are basic algorithms, certainly.
Most are simplified versions of algo. Actually used in asynchronous networks.

Will return to them when we study asych algo. in a couple of weeks.

Assume: General digraph. Processes communicate only over digraph edges.

Local names for neighbors

No particular order for subs. In general.

Symmetry: If incoming + outgoing edges connect to same neighbor, assume the names are the same.

The node knows the two edges connect to the same node.

Not really a restriction: If UID's exist, which they do in all these algo's could find this out with simple tracing, etc.

Leader election:

Assume: Strongly connected digraph (define).

UID's, companions only.

No constraints on which id's actually appear.

Both leaders + non-leaders required to output status.

If nonleader, nonleader.

If leader, leader, nonleader.

Processes know upper bound of own claim.

Annoying, but we'll eliminate it later.
Informally:
Algorithm elects max "floated" max UID thru network.
Each process maintains max-uid, a state component that records
the max uid seen so far.
Initially, its own.
At each round, each process sends max-uid on all its outgoing
edges.

After d rounds (d ≥ diam), if max-uid = process' own uid,
then process elects self leader, else non-leader.

Then collects all the msgs it receives from its subs, uses
these to update their max-uid.

After d rounds (d ≥ diam), if max-uid = process' own uid,
then process elects self leader, else non-leader.

Write as local code for each process

Code in book: Sums this global phase into a local one, in model.

State info: u = received own UID
max-uid = max known UID
status ∈ \{ unknown, leader, non-leader \}
rounds = counts # of rounds that have passed.

Transfers straightforward.

Proof: Goal is to prove assertion:
\[
\{ \text{After d rounds,} \}
\begin{align*}
\text{status}_{i} & = \text{leader} \\
\text{status}_{j} & = \text{non-leader} \\
\forall j \in \text{max}
\end{align*}
\]

Strengthens for inductive proof: after d + 1 steps
\[
\forall 0 \leq r \leq d + 1 \forall i, j \text{ if dist}(\text{max}, i) \leq r \text{ then} \\
\text{max-uid}_{j} = \text{max}_{i}
\]
Props $v_{max}$ propagates to distance $r$ by end of $r$ rounds.

Can prove this by induction (see book)

**Inductive step:** $v_{max}$ moves 1 more step...

Consider after $r$ rounds, node at dist $r$

has msg at distance $r-1$, $v_{max}$

reaches there by msg of sol $r$, by alg.

**Complexity:**

Time: $d$  

Msgs: lots  

$O(d - 1 + 1)$

Send on every edge at every round.

The idea of flooding, whereby you send info.

**Improvement:** Don't keep sending the same value repeatedly.

only send a particular value when you first learn about it.

Modified code: include a boolean flag $new$, which tracks

whether the immediate prior round produced a new

max-val.

- Set $new = True$ when max-val increases.
- False when not.
- At each round, only send $new$ if max-val.

(some improve comm. cost drastically, though not worst-case order.)

Informally, it's easy to see this still works.

Can prove similarly to "optimized" alg.

Alternative approach to proof introduces another key idea that is very

important in practice for proving (or just understanding) distributed

algorithms: simulation relationships between algorithms.
Idea (simplified here, for synchronous algos):
Imagine running two dist. algos side-by-side, "impl" vs "spec.
One already known to be correct; the other not clear yet.
E.g., optimized vs non-optimized version of same alg.
Prove an invariant relating the states of the 2 algorithms.
Called a "simulation relation" between the 2 states.
Use this connection to infer correctness of impl from correctness of spec.

\[
\begin{array}{c}
\text{Spec} \\
\downarrow \text{relation} \\
\text{Impl}
\end{array}
\]

In this example:
Consider running the two algorithms: \( U \) (unopt.) vs \( O \) (opt.)
Same digraphs, starting with same VIDs.
Sim relation (just an props of states):
"The values of \( \max \text{-vid} \) (all the state vars) are the same in both states."

Proof sim reln by induction on length of execution, just as for any invariant.
Key is showing \( max \text{-vid} \) stay the same.
Inductive step: Assume the equality holds before \( \text{rd} \) \( r \) and consider what happens at \( \text{rd} \) \( r \)
Consider some \( f \), show \( max \text{-vid} \) same in both algos after \( \text{rd} \) \( r \)
By ind. step, \( max \text{-vid} \) same in the two algos just before \( \text{rd} \) \( r \)
Thus, for each \( \text{in} \text{-vid} i \) of \( f \), \( max \text{-vid} \) same in the two algos just before \( \text{rd} \) \( r \).
Want to argue that

eed a fact to help us:

Invariant of \(0\), which can be proved separately in the usual way.

\[
\text{If } \text{max}_{j \neq i} - \text{max}_{j < i} \text{ and } \text{max}_{j > i} \text{ then new } i = \text{true}
\]

for any \(i \neq j\)

If there's a difference then the flag is set.

Usual in this reln proofs to have auxiliary vars:

Now we're broken up the work into two pieces: one this hint, the next hint to finish the sum proof.

Let's finish the sum proof. Consider any particular \(i \neq j\). Cases:

\[
\text{max}_{j > i} - \text{max}_{j < i} \text{ just before and after} \]

In this case \(O\) sends no msg.

\[
\text{new } i = \text{true} \quad \text{before the step, in } O
\]

In this case, \(O\) needs to send msg with some info, to \(i\)

\[
\text{if have the same effect in both cases.}
\]

\[
\text{new } i = \text{false} \quad \text{before the step, in } O
\]

In this case, \(O\) sends msg in \(U\) but not in \(O\).

But by hint, it doesn't matter - \(\text{max}_{j > i} \text{ and } \text{max}_{j < i} \text{ already, so it has no effect in } U\).
Why is the invariant true?

If \( \text{max-wid} < \text{max-wid}_j \) then
\[
\text{new}_i = \text{true}.
\]

Proof: By induction.

Base: \( \text{new}_i = \text{true} \) everywhere.

Inductive step: round \( r \)

Assume \( \text{max-wid} < \text{max-wid}_j \) after \( \text{rd}_r \)

Cases:

1. \( \text{max-wid} \) increases at \( \text{rd}_r \)
   
   Then \( \text{new}_i \) gets set to \( \text{true} \), as needed.

2. \( \text{max-wid} \) doesn't increase at \( \text{rd}_r \)
   
   So \( \text{max-wid} < \text{max-wid}_j \) before \( \text{rd}_r \)
   
   This doesn't decrease in \( \text{rd}_r \).

   So by i.h., \( \text{new}_i = \text{true} \) before \( \text{rd}_r \).

   So \( i \) sends \( \text{msg} \) to \( j \) in \( \text{rd}_r \)

   Which raises \( \text{max-wid} \) to \( 2 \cdot \text{max-wid}_j \)

   \( \) contra, for case 2.

\( \)