CS4231 Notes.

Date No.

Parallel Algorithms.
Processes share information by shared memory or message passing. On a single computer.

Distributed Algorithms.
Processes share information by message passing. On different computers!

I Synchronization Problems.
Properties needed:
- Mutual Exclusion
  Prove that no two processes can be in the critical section.
- Progress
  Prove that no two processes cannot enter the critical section while it's unoccupied.
- No starvation.
  Prove that not a single process cannot wait for the critical section forever, A process eventually leaves the CS.

Peterson's Algorithm.
Make use of turn and wantCS.
The rule: To be courtesy:
① Wait if the other wants the CS & it is their turn
② Shift to the other's turn before you need it.
The order of wantCS and turn matters.
It is re-usable.

Lamport's Bakery Algorithm.
Procedure:
① Get a number first.
② Get served when all lower numbers are served.

Date No.

Two supporting data structures,
① choosing [i].
② number [i]. // if 0, then the i-th process is not interested.

Special Machine-level Instruction.
openDoor = true == the CS is unoccupied.

Semaphores:
RequestCS() = P() → set to false
ReleaseCS() = V(). → set to true.

Monitors:
① Monitor-queue: has nothing to do with wait() and notify().
② Wait-queue: processes wait here after calling wait().
Synchronized can modify ~~an object~~, a block of code, or a function. If a two functions of the same object are synchronized, then all processes can enter only one at a time.
Notifications do not have lasting effects.
- Hoare Style. (only applicable to 2 processes).
  The one that gets notified gets the monitor lock.
- Java-style
  The one that calls notify() gets the monitor lock. The one that gets notified still needs to acquire the monitor lock.

Need to avoid nested monitors.

Date No.

Producer-Consumer Problem:

- Producer waits when full, notifies when used to be empty.
- Consumer waits when empty, notifies when used to be full.

II Orders in Parallel Systems.

History: A sequence of invocations and responses ordered by real time. Just a partial order.

Sequential History: A sequence of events where $\forall e$, $inv(e)$ immediately precedes $resp(e)$. A total order.

Concurrent History: Not a sequential history.

Legal Sequential History: Satisfies the semantics of all objects involved.

Subhistory: $H|P$ (always sequential), $H|o$.

Sequential Consistency: Equivalent to some legal sequential history that preserves process order. The history itself may not be legal.

External Order: $o1 < o2 \iff resp(o1) < inv(o2)$.

Linearizability:

Def #1. The history is equivalent to a legal sequential execution such that all events happen instantaneously at some point between the inv and the resp

Def #2: The history H is equivalent to a legal sequential history S and S preserves the partial order (external order) induced by H. (The external order induced by H is a subset of that induced by S).

Date No.

Local Property.

H is linearizable $\Leftrightarrow \forall x \in \{Objects\};\ H|x$ is linearizable.

Proof ($\Leftarrow$).

① $o1 <_x o2$ if $o1$ happens before $o2$ on the same object x at $H|x$. $o1 \to o2$ "due to x"
$o1 <_H o2$ in external order. $o1 \to o2$ "due to H"

② Legal subhistories add up to legal history because objects are independent of e.o.

③ Since $<_x$ and $<_H$ are partial orders, the directed graph $<_H \cup_{\forall x} <_x$ is acyclic only if the cycle interleaves $<_H$ and $<_x$. Proof by contradiction.

④ Any topological sorting of the DAG is a linearization of H.

#: Trick; If $A <_x B$, then $inv(A) < resp(B)$.

Registers:

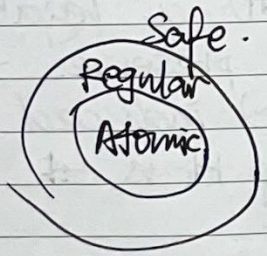
Atomic: Impl. always ensures linearizability.

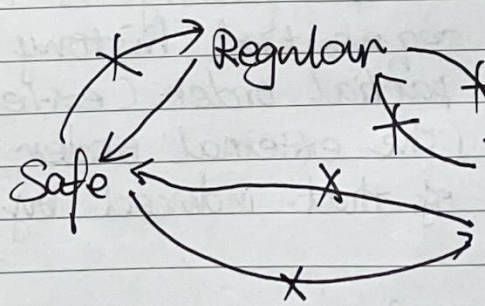
Regular: No overlapping writes → read one of the most recent values written.

Subset. Overlapping writes → read one of the most recent writes or a overlapping value. Possible that if 2 reads overlap with a write, then the later read returns the old value and the earlier read returns the new value.

Safe; No overlapping → one of the most recent.
Overlapping → anything

Not possible that if 2 reads overlap with a write, then the earlier read returns the new value and the later read returns the old value.





NUS National University of Singapore

Date No.

III Distributed Systems.

Three kinds of atomic actions/events.

① Local computation.
② Send a message.
③ Receive a message.

- No atomic broadcast.

Communication Model.

- PTP. (Point-to-point)
- Error-free, infinite buffer
- Potentially out of order.

Constraint: No (perfectly) sufficiently accurate physical clock.

Happened-Before Relation.

- A partial order among events, consisting of:

① Process order.
② Send-receive order
③ Transitivity.

Logical Clock.

- Increment the local counter on local computation or sending a message.
- Take the max of incoming clock value and its own local counter value when receiving a message. Increment as well.

$$s \to t \Rightarrow s.c < t.c$$

Vector clock.

- An (vector) array of N integers for N processes.

$V_1 \leq V_2 \iff \forall i,\ V_1[i] \leq V_2[i]$,

Now, $<$ is only a partial order among vectors.

- Increment the i-th value on local computation.
- Take element-wise max of own vector and the incoming vector when receiving a message.

Date No.

- Increment the i-th value before sending or after receiving.

$$s \to t \iff s.\underset{\sim}{V} < t.\underset{\sim}{V}$$

Trick: only the i-th process can update its i-th value.

$$s \neq t \wedge s \not\to t \Rightarrow t.\underset{\sim}{V}[s.p] < s.\underset{\sim}{V}[s.p].$$

If we know the processes the vectors come from,

$$s \to t \iff (s.\underset{\sim}{V}[s.p] \leq t.\underset{\sim}{V}[s.p]) \wedge (s.\underset{\sim}{V}[t.p] < t.\underset{\sim}{V}[t.p]).$$

Matrix Clock.

- $s.\underset{\sim}{V}[i,j]$ represents what $s.p$ knows about what i knows about j.
- Compare the happened-before relation by comparing the principle vectors.
- For principle vector, take pairwise max of it and the incoming principle vector of the sender.
- For non-principle vectors, take pairwise max.

This procedure updates "my view" with information from the sender.

At process j, $M_j[j,k] \geq M_j[i,k] \quad \forall i,k$.

Don't forget to increment $M[s.p, s.p]$.

Date No.

IV Global Snapshot.

A set F s.t. $f \in F \wedge e \prec f \Rightarrow e \in F$ (↓ process order.)

Consistent Global Snapshot.

$f \in F \wedge e \rightarrow f \Rightarrow e \in F$

Theorem:

For a process P with events e_i, for any integer m, $\exists$ a consistent global snapshot S, s.t. $\begin{cases} e_i \in S & \text{for } i \leq m \\ e_i \notin S & \text{for } i > m. \end{cases}$

Communication Model

- No loss, no error
- Unidirectional communication channels.
- FIFO order of message delivery.

Messages:

- Application messages: If their receive events are captured, so are their send events.
- Control messages: They won't be included in the snapshot.

Chandy & Lamport's Protocol.

- Each process is either red or white.
- Upon turning red, take a local snapshot and send a marker to all neighbours.
- Upon receiving a marker, turn red immediately.
- Upon telling neighbours to turn red (sending a marker out), include subsequent receive events of messages from neighbours until received a marker from them. (unless they turned red earlier)
- "Capturing the messages on the fly". Messages on the fly must be in the opposite direction of the triggering marker.

Date No.

V. Message Ordering

Causal Order VS FIFO.

FIFO: $s_1 \prec s_2 \Rightarrow \neg(r_2 \prec r_1)$

Causal Order (stronger): $s_1 \rightarrow s_2 \Rightarrow \neg(r_2 \prec r_1)$.

Note that r_1 and r_2 may be on different processes.

In this chapter, FIFO is not an underlying property.

Causal Order Protocol → Not a matrix clock.

- Matrix M[i,j]: # messages from i to j, as known by the underlying process.
- i sends to j: M[i,j]++ and piggyback M_i
- j delivers the message and set the pairwise max, if
  1. $T[k,j] \leq M[k,j] \ \forall k \neq i$ AND
     If not satisfied, there are some messages from another process k to j but not known by j, only known by i.
  2. $T[i,j] = M[i,j] + 1$.
     If not satisfied, there are some messages from i to j not yet received by j.
- M[i,j] on j takes consecutive values.

Correctness Proof: If $s_1 \rightarrow s_2$, assume r_1 and r_2 are on the same process k.

1. r2 is not before r1.
2. r_1 and r_2 will be delivered eventually.

Proof 1:

- If s_1 and s_2 are on the same process, then trivial.
- They are on different i,j. Consider $M_{s_1}, M_{s_2}, M_{r_1}, M_{r_2}$.
  There must be a chain of messages from i to j.
  $M_{s_1} = M_{r_1}$ by piggybacking.
  ∴ Before s_2 and after s_1 there is a matrix on j $M_j[j,k] + 1 = M_{s_2}[j,k]$ and $M_{s_1} \leq M_j$.
  ∴ $M_{r_1} \leq M_{r_2}$ and $M_{r_1}[j,k] < M_{r_2}[j,k]$.
- If r_2 is delivered first, by the monotone of $M_k[j,k]$, r_1 can never deliver.

Proof ②:

- Consider j and $M_j[i,j]$.
- Consider the non-empty set of undelivered messages. $\forall i$, $\exists$ a message whose $M[i,j] = M_j[i,j]+1$. We are assuming such messages eventually arrive, so their existence is presumed. These are called successor messages
- There is at least one sent event of successor messages that is not after any other sent events, by the partial order property. Those corresponding messages are call top successor messages, which can be delivered.
- For a TSM, t, we already know $M_t[i,j] = 1 + M_j[i,j]$. Need to prove $M_t[k,j] \leq M_j[k,j]$ $\forall k$.
  - Proof by contradiction, suppose $M_t[k,j] > M_j[k,j]$.
    Let l be the last delivered message from k to j. (If not existing, then just plug in $M_j[k,j] = 0$). $\therefore M_t[k,j] > M_l[k,j]$.
    $\therefore$ There must be a chain from l to t
    Since $M_j[k,j]$ is consecutive on j, there must be an undelivered message, u, from k to j.
    u must be before the chain from l to t, otherwise its increase in $[k,j]$ won't be reflected in $M_t[k,j]$.
    $\therefore u \to t$ and is undelivered.
    t is not a TSM. Contradiction.

Broadcast.

A message is sent to all processes (including the sender).

Total Order of Broadcast Messages / Atomic Broadcast

All messages are delivered to all processes in exactly the same order.
If $x <_p y$, then $x <_q y$

Date No.

Note that messages can be sent in different order of processes, but all processes deliver messages in the same order of messages.

Coordinator:
Forwards messages to all processes.
Has overly much control.

Skeen's Algorithm for Totally Ordered Broadcast.
- Each process maintains a logical clock and a(n) (infinite) buffer.
- Deliver a message if:
  AND ① ALL messages buffered are assigned a number.
  ② It has the smallest number.
- The initiator assigns a broadcast message with the highest logical clock value it has ever received.

CS4231 Notes II

Date No.

Leader Election.

Anonymous Ring. (NO unique IDs).

Impossible to solve using deterministic algorithm.
→ All steps and states are the same
→ Symmetry not broken.

General Ring. (Unique IDs) = Chang-Robert Algorithm

- Every nodes sends its ID to the next node.
- Only forward bigger IDs than its own.
- The node who received its own ID becomes the leader.
- Performance: $O(n \lg n)$ messages

Chang-Robert: Analysis.

- $(n-1)!$ Total orderings on a ring.
- Define x_k = # messages caused by node k.
- $\Pr[x_k = 1] = \frac{n-k}{n-1}$

$\Pr[x_k = 2 \mid x_k > 1] = \frac{n-k}{n-2}$. $= \frac{n-k}{n-1}$

$\exists p$, s.t. $\Pr[x_k = i+1 \mid x_k > i] \geq p$. $\forall i$

$E[x_i] \leq E[y_p] = \frac{1}{p} = \frac{n-1}{n-k}$

$\therefore \sum_{k=1}^{n} E[X_k] = n + \sum_{k=1}^{n-1} \frac{n-1}{n-k} = n + (n-1) \sum_{k=1}^{n} \frac{1}{k}$

$= n + (n-1)\, O(\lg n) = O(n \lg n)$

Leader Election on General Graph (n known).

- Send own id to all other nodes.
- Wait for all ids. → The biggest wins.

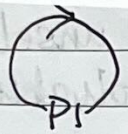
How to "flood" msg to other nodes and get n?
We have spanning tree construction.

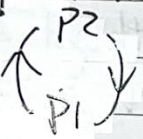
- Send an invitation to neighbours, inviting them to be own children, starting from the root to every node received msg.
- Each receipient either accepts or rejects
- Every node does not send a msg to its parent until it has received msg from all children.

Date No.

- We can use this spanning tree to broadcast/ aggregate info (max/min/avg ...).

Randomized Algorithms cannot Deal with Unkown Size.

- 



Indistinguishable for P1.

Randomized Algorithms for Known Size.

- Use a phase #. Attach phase # when sending, increment hop # when forwarding.
- Randomly pick an ID and run Chang-Robert.
- Losers from a phase onwards can only forward. Winners will see a msg after n hops. and proceed to the next round.
- Proof of termination with prob 1. (No proof by contradiction).
  - The i-th good phase results in the i-th loser. We need at most n-1 of which.
  - r := # phases. x_i := # phases btw the i-th and (i+1)-th good phase. [m] := the range of ID.

$$\Pr[x_i > \frac{r}{n-1}] = \underbrace{\underbrace{\left(\left(\frac{1}{m}\right)^{n-i-1}\right)}_{\text{prob of } n-i-1 \text{ nodes picking the same ID.}}^{\frac{r}{n-1}}}_{\text{happens } \frac{r}{n-1} \text{ times.}} \xrightarrow{n\to\infty} 0.$$

- Note: Picked IDs in one phase don't have to be greater than all IDs from all previous phases.

Consensus.

Ver	Problem Setting	Solution
0	No link or node crashes	Trivial solution.
1	Nodes can fail; Channels are reliable; Synchronous	Solvable.
2	Nodes won't fail; Channels are UNreliable; Synchronous (the coordinated attack problem)	Without error; unsolvable With error; solvable
3	Nodes may fail; Channels are reliable; Asynchronous	Unsolvable.
4	Nodes can have Byzantine failures; Channels are reliable; Synchronous (The Byzantine Generals)	Solvable with a lower bound of resources.

Ver 1

Synchronous.

- There is an upper bound for message delay and node processing
- Implies the notion of rounds and failure detection.

Goals:

- Termination: All nodes (that did not crash) eventually decide
- Validity: If all nodes have the same input, they should decide on the only possible decision.
- Agreement: All nodes that decide should decide on the same value.

Idea:

- The problem rooted from that if some nodes crash, some non-faulty (NF) nodes only get a subset of all values.
- Solution: Keep forwarding values to NF nodes
- For f failures, we need f+1 rounds.

Date No.

- Good round: A round is good if there is no node failure during that round.

Proof:
- At the end of a good round r, all NF nodes have the same set S.
- After r in any round t, all NF nodes' S does not change.
- All NF nodes will have the same S upon termination.

Theorem on Lower Bound of # Rounds.
- $\Omega(f)$
- Any consensus protocol has at least $f+1$ rounds.

Ver 2 (Determinstic without Error).
Goals: the same as in Ver 1, except for termination: all nodes eventually decide (no faulty nodes)

Impossibility Proof:
- Use indistinguishability in a scenario where all msg are dropped
- Alternatingly find the last msg from A to B and from B to A. Before and after the removal, the scenario is indistinguishable for who sends the msg.

↑ For Weakened Validity:
- All inputs are 0 → All nodes decide on 0
- All inputs are 1 and no msg dropped → on 1.
- Otherwise → decide on anything satisfying agreement.

Indistinguishability:
- Execution α is indistinguishable from β for node X if X sees the same input and msg in both executions.

6

Ver 2 (Randomized with error)

Goals:

- Termination: All nodes eventually decide.
- Agreement: All nodes decide on the same value with prob $1 - r$
- Validity: Weakened validity

Adaptive Adversary VS Oblivious Adversary. (in scope).

Algorithm:

- r rounds,
- Pick bar $\in [r]$ randomly, store it in P1
- Each node maintains a level. Attach it when sending out, set own level := received level + 1.
- Level initialized to 0.
- Levels differ at most by 1.
- All msg contain bar / sender level / input.
- A node decides on 1 if:

and
- It knows all nodes' input is 1. (including itself)
- Its level $\geq$ bar.
- It knows bar.

Proof:

- Error only occurs when bar = ℓ_{max}.
- Since bar $\in [r]$, error prob = $\frac{1}{r}$.

Ver 3.

Asynchronous: Process delay & msg delay are finite but unbounded.

→ We no longer have the notion of a round.

→ But msg are eventually received and processed at NF nodes.

Goals: as in Ver 1. Except Agreement:

No reachable global state from any initial state has more than one decision.

NUS National University of Singapore

Date No.

FLP Theorem

The distributed consensus problem under the asynchronous communication model is impossible to solve even with only one node failure.

Input: {0, 1}, decision ∈ {0, 1, null}.
↳ initial, can be written only once

Terminology

- Msg: (p, m) m sent to p, on the fly. Distinct.
- Send: add (p, m) to the msg system.
- Receive: Remove a msg from the system OR return null.
- Global state: Includes all processes' states and the message system's.
- Step: takes the global state from one to another. Including
  Atomic {
  - Receive a msg (can be null)
  - Send an arbitrary but finite number of msg;
  - Change a process's local state to another
  }
- Event: (p, m), applicable to a global state G if m is null or (p, m) is in the msg system of G.
- Execution: An infinite sequence of events.
- Schedule: σ, a sequence of events. σ can be applied to G if all events in σ are applicable to G in the same order as in σ.
- Reachable: G' is reachable from G if $\exists \sigma$ s.t. $G' = \sigma(G)$.
- 0-valent: G is 0-valent if 0 is the only reachable decision value.
- 1-valent: Similar.
- Univalent: Either 0- or 1-valent.
- Bivalent: Otherwise.

possible that $G = \sigma(G)$.

Observation.

- NF processes take infinite steps
- F processes take finite steps

Prefix of schedule: from early events to later events. from right to left

Date No.

Lemma 1: ("Bivalent initial state")
For every protocal A, ∃ a bivalent initial state.

Lemma 2: ("Disjoint Processes").
Let $\vec{\sigma}_1$ and $\vec{\sigma}_2$ be two schedules s.t. the set of processes executing steps in $\vec{\sigma}_1$ is disjoint with that of processes executing steps in $\vec{\sigma}_2$. Then for any G that $\vec{\sigma}_1$ and $\vec{\sigma}_2$ can be applied, we have $\vec{\sigma}_1(\vec{\sigma}_2(G)) = \vec{\sigma}_2(\vec{\sigma}_1(G))$.
Hint: prove by induction on $k = \max(|\vec{\sigma}_1|, |\vec{\sigma}_2|)$.

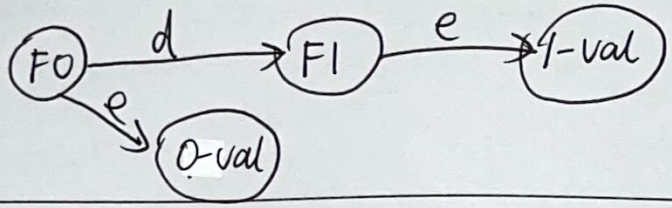
Lemma 3: ("Postponing an event")
Let G be a global state, and $e = (p, m)$ is an event that can be applied G. Let W be the set of global states reachable from G without applying e, then e can be applied to any state in W.

Lemma 4: ("Keeping the state bivalent").
Let G be a bivalent state, $e = (p, m)$ applicable to G. $W := \{$states reachable from G without applying $e\}$. $V := e(W)$. V contains a bivalent state.
Proof by Contradiction: V does not.
Claim 1:
There exists some schedule $\vec{\sigma}$ s.t. $e \in \vec{\sigma}$ and $\vec{\sigma}(G)$ is 0-valent.

Claim 2:
There must be a 0-valent state in V.

Claim 3:
There must be a 1-valent state in V.

Claim 4:
There must be F0 and F1 in W s.t. e(F0) is 0-valent, e(F1) is 1-valent. Either $F0 = d(F1)$ or $F1 = d(F0)$ for some event d.



Date No.

※ Trick: d and e must on the same process p.
By the termination requirement, nodes decide within a finite time period ⇒ hold p until the end of the period. Then $T := \delta(F0)$
$\therefore e(d(\delta(F0))) = \delta(e(d(F0))) = \delta(e(F1)) = \delta(G1) \Rightarrow 1\text{-val}$.
But $e(\delta(F0)) = \delta(G0) \Rightarrow 0\text{-val}$.
(As the adversary, we can control which of e and d will come first)

Ver 4.

Setting: A faulty node does not just halt, but it can also misbehave.

Goals:
slightly different from in ver 1.
- Termination: All NF nodes eventually decide.
- Validity: If all non-faulty nodes have the same inputs they should decide on the only possible value.
- Agreement: All NF nodes decide on the same value.

Byzantine Consensus Threshold.
$n \leq 3f \Rightarrow$ cannot be solved. In this case $n \geq 4f + 1$

Intuition: Rotating Coordinator + Notion of Phases.
- Each phase has a coordinator round + an all-to-all broadcast round.
- A phase is a deciding phase if the coordinator is NF.

※ Note: • 0 is the default value. A node can propose anything.
• It is possible that the coordinator propose a value that $n/2 + 1 - f$ good nodes and f bad nodes send to it. These f bad nodes may send something else to others, but all good nodes will still accept the proposal.

- It is even possible to reach a consensus on a value proposed by a bad node!
- If the vast majority pick the same value, the final y value may be the same as coordinator Proposal, but $> n/2 + f$.

Lemma 1
- If all NF processes P_i have $V[i] = y$ at the beginning of phase k, then this remains true at the end of phase k.
-

Lemma 2.
- If the coordinator in phase k is NF, then all NF processes P_i have the same $V[i]$ at the end of phase k.
- Case 1: coordinator has proposal $= x$.
- Case 2: coordinator has proposal $= 0$

Self-Stabilization.

Causes of Errors.
- Topology Changes.
- Failures/Reboots.
- Generally called "faults"

Notion of "Illegal state"
- Depends on application semantics.
- The code itself won't make mistake.

Definition of "self-stabilizing"
- A distributed system is S-S if the protocol will eventually reach a legal state (in finite time/steps) if there are no more faults.
- Once the system is in a legal state, it will only

Date No.

transit to other legal states unless there are faults.

The Rotating Privilege Problem.

- A token is passed in a round-robin fashion.
- $0 < V_i \leq K-1$ for $K \geq n$ and $\forall i \in [n]$.
- A fault refers to an accidental change of V_i
- Assume there is no interleaving.

Execution:

- For the only red node: increment V_s if its clockwise neighbour has $V_l = V_s$
- For other blue nodes: set $V_i = V_l$ if $V_i \neq V_l$.

Lemma: "A legal state looks like ..."

The following are the only two legal states:

① All V_i are the same

② Only 2 different values forming two consecutive bands, one from the red process.

Legal states ⇒ Legal states.

Case 1: V = L (the red V)

Case 2: V ≠ L (the red V)

Illegal states ⇒ Legal states.

Lemma 1: Let P be blue, Q be the clockwise neighbor of P. If Q makes i moves, then P can make at most $i+1$ moves.

Lemma 2: If the red process makes i moves the there are at most $ni + \frac{(n-1)n}{2}$ moves.

Lemma 3: A sequence of $\frac{(n-1)n}{2} + 1$ moves has at least one move of the red.

Lemma 4: In any (illegal/legal) state, at least one move is possible/applicable.

Date No.

Lemma 5: Regardless of the starting state, the system eventually reaches a state T where the red process has a distinct value from all other V_i.
- Use the Pigeonhole Principle.
- Use the idea that only Q can introduce new values. Only can others adopt values.

Lemma 6: If the system is in such a state T, then eventually all V_i are equal including Q's.
- Note that: if the final value is x, then a blue node cannot take x if its clockwise neighbour has yet to do so. → This is important to ensure that Q does not change in the middle.

Self-Stabilizing Spanning Tree Algorithm.
- Given n processes and a root P1, construct an ST at P1.
- A general undirected graph.
- Each process maintains parent, (non-negative) dist.
- There CAN be interleaving.

Execution: # ~~hops~~ nodes to dist 11
On P1: Periodically set parent := null, dist = 0.
On others: Retrieve dist from all neighbours and pick the one with the smallest as parent.

*Note: A node may not correct itself in one single move.

Terminology:
- Phase: The min time within which all processes have taken an action / executed the whole code once.
- A_i: Level of process A → the optimal value of dist.

NUS National University of Singapore

Date No.

Properties:

A node at level $X \geq 1$ has a neighbour at level $X-1$.

A node only has neighbours from level $X-1$, X, $X+1$.

Lemma 1:

At the end of phase 1, $dist_1 = 0$ and $dist_i \geq 1$ for $i \geq 2$.

Lemma 2:

At the end of r, $dist_i = A_i$ for $A_i \leq r-1$,

$dist_i \geq r$ for $A \geq r$.

General Strategy:

Proof by Induction on t (the time, which is quantized within a phase)

① At t, actions will not roll-back what is already achieved by phase r.

② At some t', more will be achieved by each node.

③ The forward moves at t' will not be later retracted.