# Introduction to Distributed Algorithms

John Augustine

IIT Madras
augustine@cse.iitm.ac.in

January 17, 2013

# Lets Start @ the Very Beginning . . .

**Pre-60's:** Telephone, telegram and postal networks.

→ Borůvka's Algorithm for MST invented in 20's

**60's and 70's:** Mostly small networks, simple topologies.

→ DARPA starts arpanet.

**80's:** Larger networks (mostly in-house in large organizations).

→ Email and other Internet technologies taking shape.

→ Fisher, Lynch and Paterson show that deterministic agreement is impossible in faulty asynchronous networks.

# The Big Bang

**90's:** The Internet breaks into the scene.

- $\rightarrow$ Cell phones become popular in many countries.
- $\rightarrow$ PageRank algorithm invented at Stanford.
  Google starts operation from a garage.
- $\rightarrow$ Distributed algorithms is established firmly.

# The Expanding Universe of Networks

**2000's:** Ubiquitously connected lifestyle

- $\rightarrow$ ~~Cell~~ Smart phones become popular in India.
- $\rightarrow$ Social Networks, P2P networks, ...
- $\rightarrow$ Cloud computing, big data, ...

# Overview

1. Agreement
2. Leader Election
3. Minimum Spanning Tree
4. Maximal Independent Set

# Audience Participation

→ Choose a number: 37 or 43.

→ You may communicate with your neighbours.

→ Goal: everybody must agree on same number.

Either number is fine. Just reach agreement!

What lessons can we learn?

# Model or Setting

*The number of models in distributed computing is only slightly smaller than the number of papers published.*

— *David Peleg*

# Model or Setting

**Distributed.** Graph $G = (V, E)$.

- $\rightarrow$ Each node is a computer
- $\rightarrow$ Nodes have unique ids (think ip address)
- $\rightarrow$ Edges indicate communicability
- $\rightarrow$ Possibly weighted — bandwidth, transmission cost, energy cost, etc.

**Synchronous.** Common clock tick. Rounds $1, 2, 3, \ldots$

- $\rightarrow$ Not realistic, but helpful in gaining traction.

# Model or Setting

**Message Passing.** If $e = (u, v) \in E$, then $u$ and $v$ can directly send messages to each other.

$\mathcal{CONGEST}$**.** Size of each message is $O(\log n)$, where $n = |V|$.

$\rightarrow$ Why $O(\log n)$? Why not $O(\sqrt{n})$?

In Each round $r$, each node executes following steps in sequence.

1. Perform local computation.
2. Send messages to neighbours.
3. Receive messages.

# The Agreement Problem

**Input.** Each node starts with a bit value from $\{0, 1\}$.

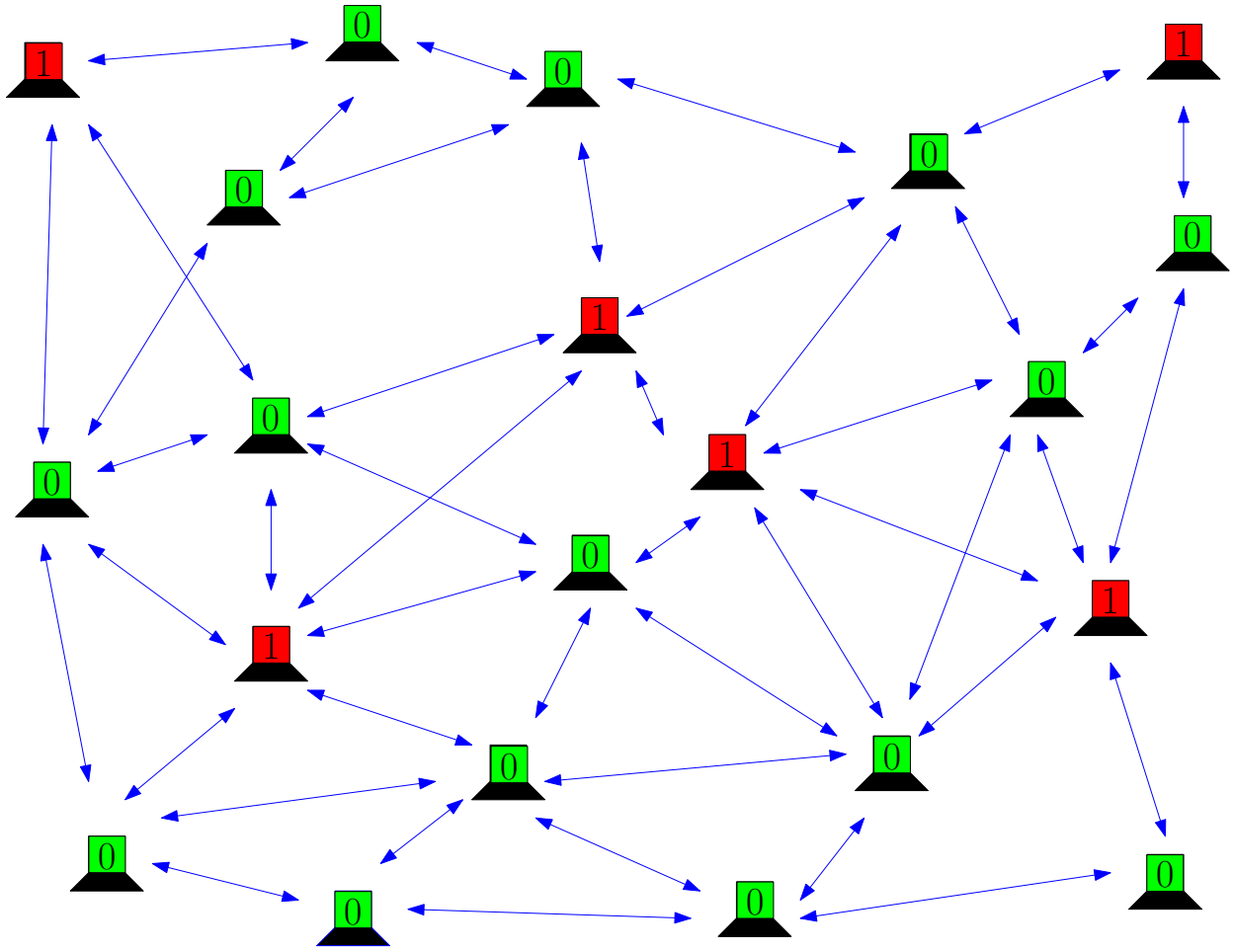
**Goal.** Achieve following conditions.

**Agreement.** Every node must "output" the same bit value.

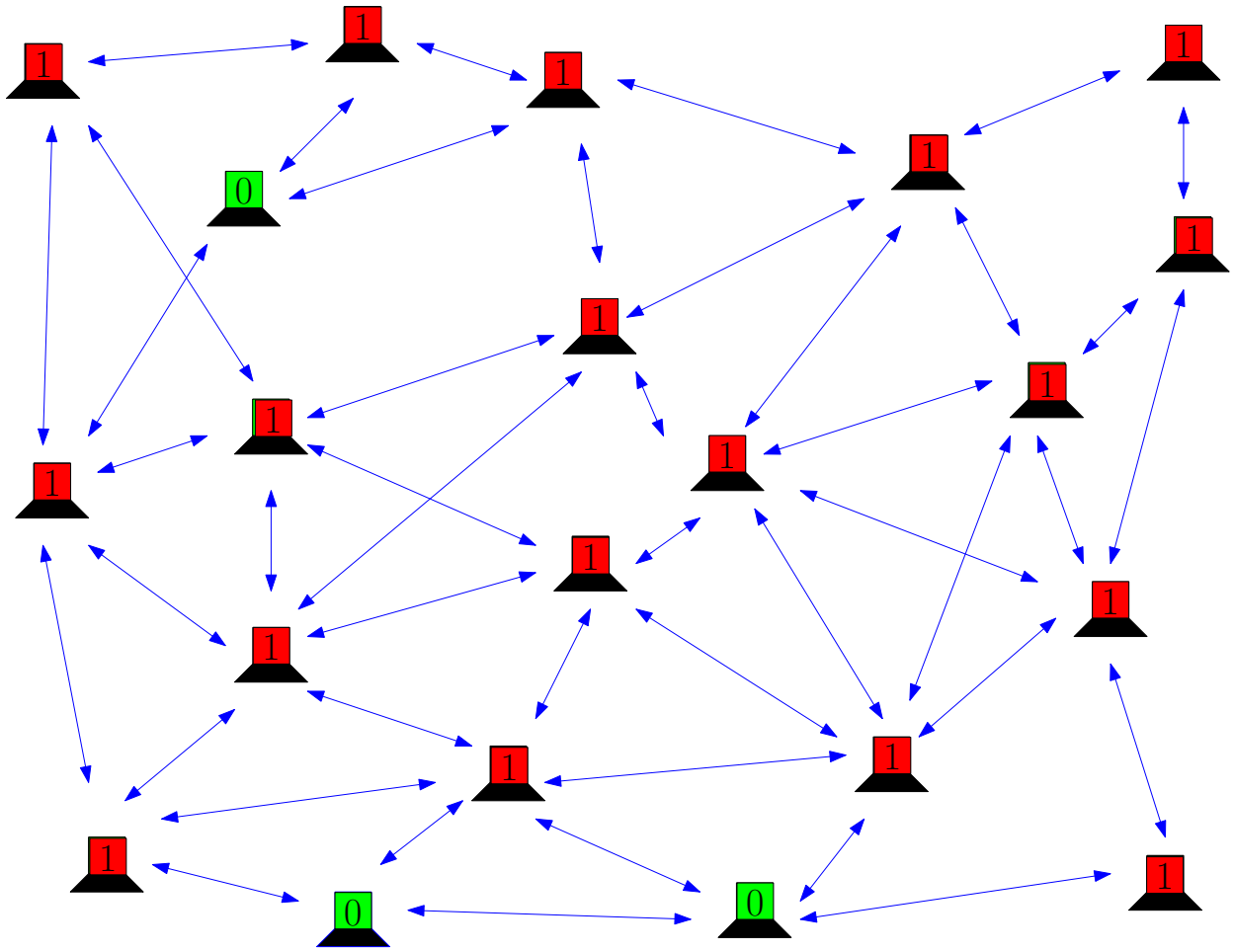
**Validity.** At least one node must have started with the agreed bit value.

**Termination.** Must terminate and nodes must know when algorithm terminates.

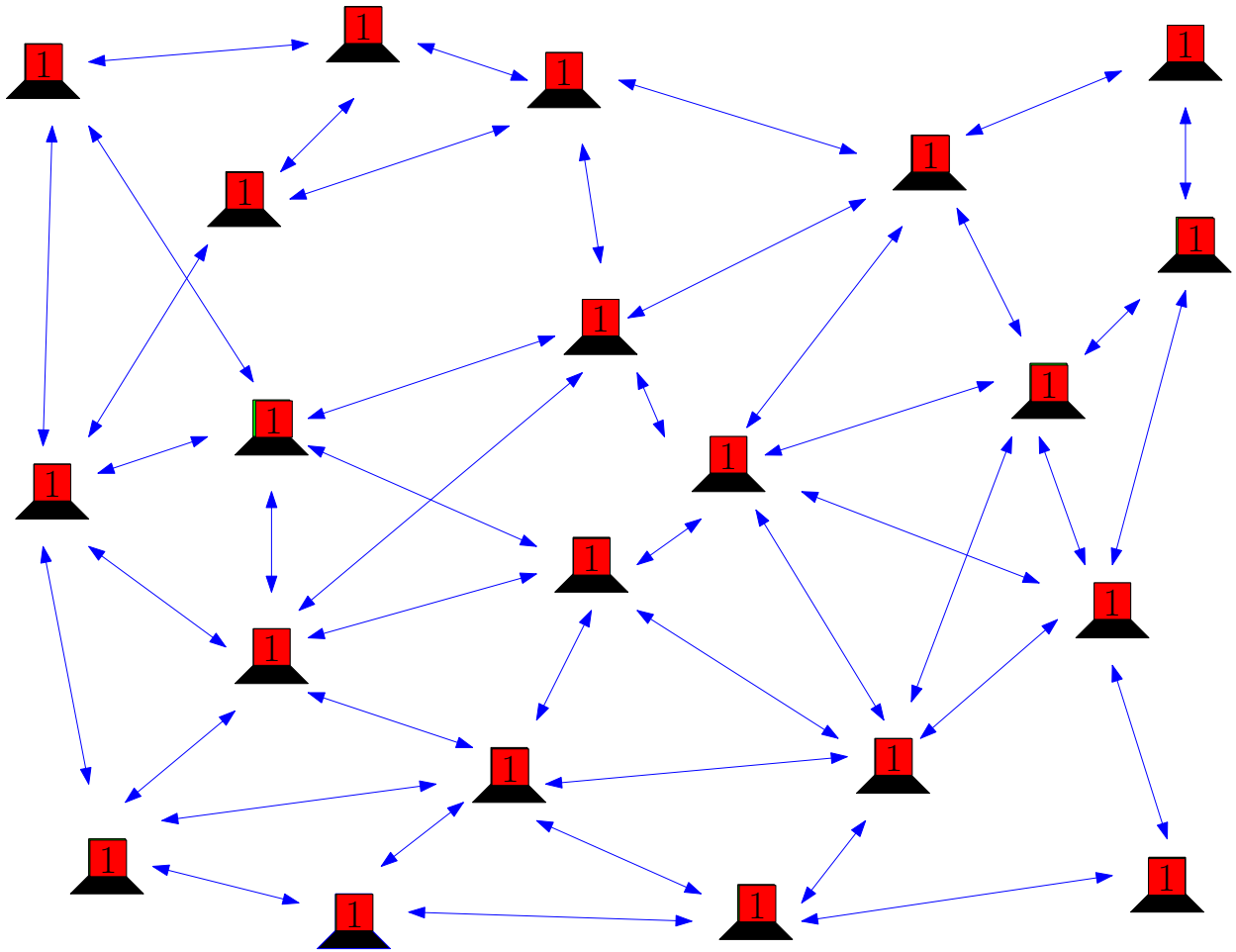
# Algorithm for the Agreement Problem



# Algorithm for the Agreement Problem



# Algorithm for the Agreement Problem



# Algorithm for the Agreement Problem

- How many rounds are required to reach agreement?

  $O$(diameter) rounds

  (Nodes need to know the diameter.)

- What happens when there are NO nodes starting at 1?

  Validity maintained.

# Algorithm for the Agreement Problem

- How many rounds are required to reach agreement?

  $O$(diameter) rounds
  (Nodes need to know the diameter.)

- What happens when there are NO nodes starting at 1?

  Validity maintained.

# Algorithm for the Agreement Problem

- How many rounds are required to reach agreement?

  $O$(diameter) rounds
  (Nodes need to know the diameter.)

- What happens when there are NO nodes starting at 1?

  Validity maintained.

# Algorithm for the Agreement Problem

- How many rounds are required to reach agreement?

  $O$(diameter) rounds
  (Nodes need to know the diameter.)

- What happens when there are NO nodes starting at 1?

  Validity maintained.

# How Good is the Algorithm?

**Time complexity.** $O(\text{diameter})$

**Message complexity.** ~~$O(\text{diameter} \times m)$~~ $O(m)$. (Here $m = |E|$.)

Can we achieve $O(n)$?

# How Good is the Algorithm?

**Time complexity.** $O(\text{diameter})$

**Message complexity.** ~~$O(\text{diameter} \times m)$~~ $O(m)$. (Here $m = |E|$.)

Can we achieve $O(n)$?

Use a spanning tree!

# How Good is the Algorithm?

**Time complexity.** $O(\text{diameter})$

**Message complexity.** ~~$O(\text{diameter} \times m)$~~ $O(m)$. (Here $m = |E|$.)

Can we achieve $O(n)$?

Use a spanning tree!

Will that affect running time?

# Agreement $\xrightarrow{?}$ Leader Election

# Agreement $\xrightarrow{?}$ Leader Election

Each node performs the following.

1: Generate a random number (with sufficiently many bits).

2: Initialize $(r, id)$ pair with $r$ being its own random number and $id$ being its own id.

3: **for** $i \leftarrow 1$ to diameter **do**
4: Send $(r, id)$ pair to neighbours.

5: Receive $(r, id)$ pairs from neighbours.

6: Update its $(r, id)$ pair if a bigger $r$ value is received.

7: **end for**

# Minimum Spanning Tree (MST)

→ Recall MST: spanning tree of $G$ whose total edge weights is minimum.

→ Goal: each node must know which of its incident edges belong to the MST and which do not.

→ W.L.O.G., assume that all edge weights are distinct.

- This implies a unique MST.

# MST Basics

- Let $T_M$ be the (unique MST) of $G$.
- A fragment is a connected subtree of $T_M$.
- An outgoing edge of a fragment is an edge in $E$ where one adjacent node to the edge is in the fragment and the other is not.
- The minimum weight outgoing edge (MOE) of fragment $T$ is called $M(T)$.
- The following rule produces a MST.
- **Blue rule:** Pick a fragment $T$ and add $M(T)$ to it (until possible).

# Synchronous GHS Algorithm (Gallager, Humblet, and Spira)

- → Each node begins as a fragment by itself
- → All nodes end as one fragment — the MST.
- → Every fragment will have exactly one root node.
- → Each nodes in the fragment will know its parent and children.
- → Each fragment is identified by the identifier of its root and each node knows this.

# Finding Minimum Weight Outgoing Edge

- → Root broadcasts a "command" to all nodes in the fragment using the edges in the fragment.
- → Each node, on receiving command, checks its neighbours (in increasing order of weight.)
- → (Once an edge is known to lead to another node in the same fragment, it is not considered again.)
- → Each node reports its candidate MOE to the root by a prioritized convergecast.
  - Only smaller weight edges progress toward root.

# Combining Fragments

→ Each fragment combines with the fragment at the other end of the edge.

→ If other fragment also selected the same edge:

- Two fragments agree to combine at that point.
- Endpoint with higher id becomes new root.

→ The (combined) root broadcasts a "new-fragment" message through the fragment edges and the MOE edges.

→ Each node updates its parent, children, root identifier.

# Correctness

→ The algorithm essentially follows the blue rule.

→ Can a cycle be formed when merging fragments?

→ A directed graph is weakly connected if its undirected version is connected.

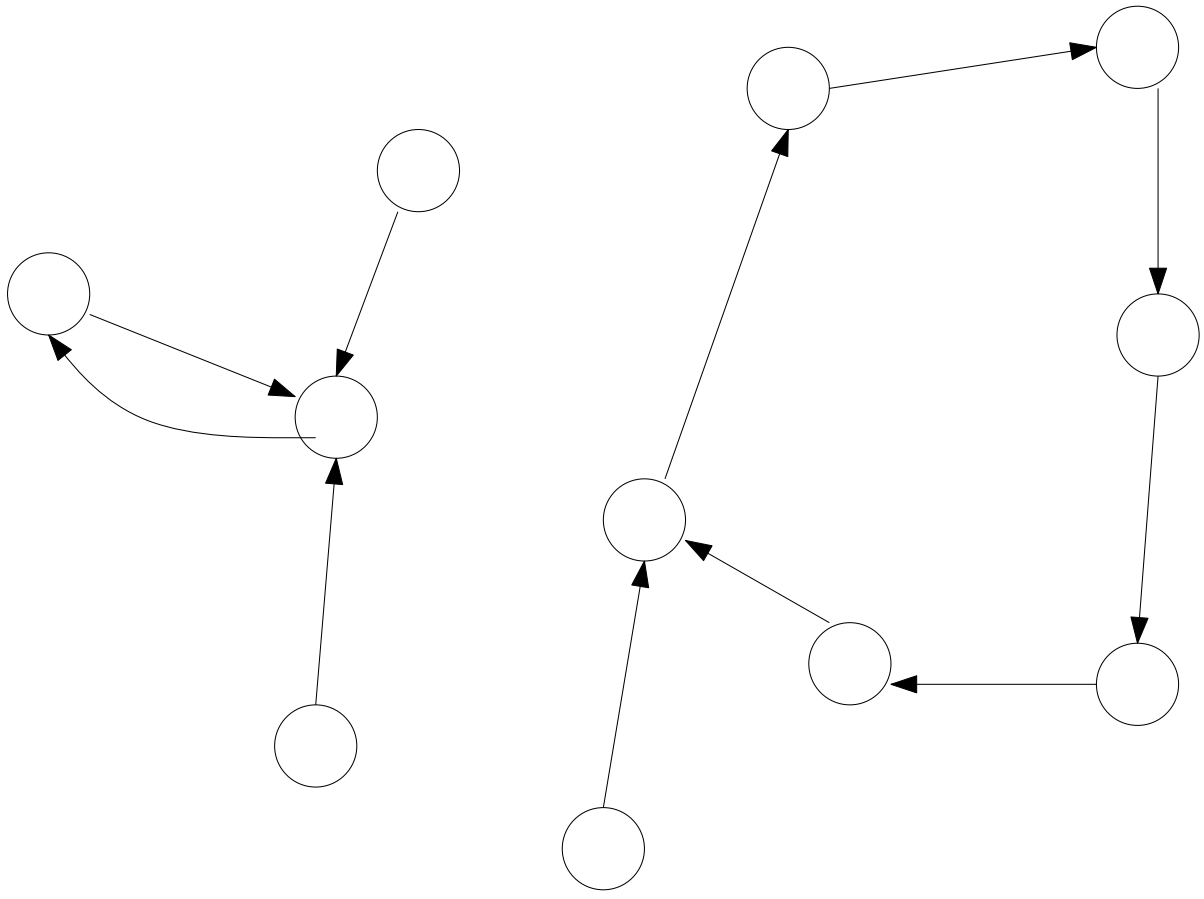
**Lemma**

*A weakly connected digraph in which each vertex has exactly one outgoing edge has exactly one cycle.*

**Proof Hint.**

Use induction. (See figure next slide.) □

# Illustration of Lemma



# Correctness

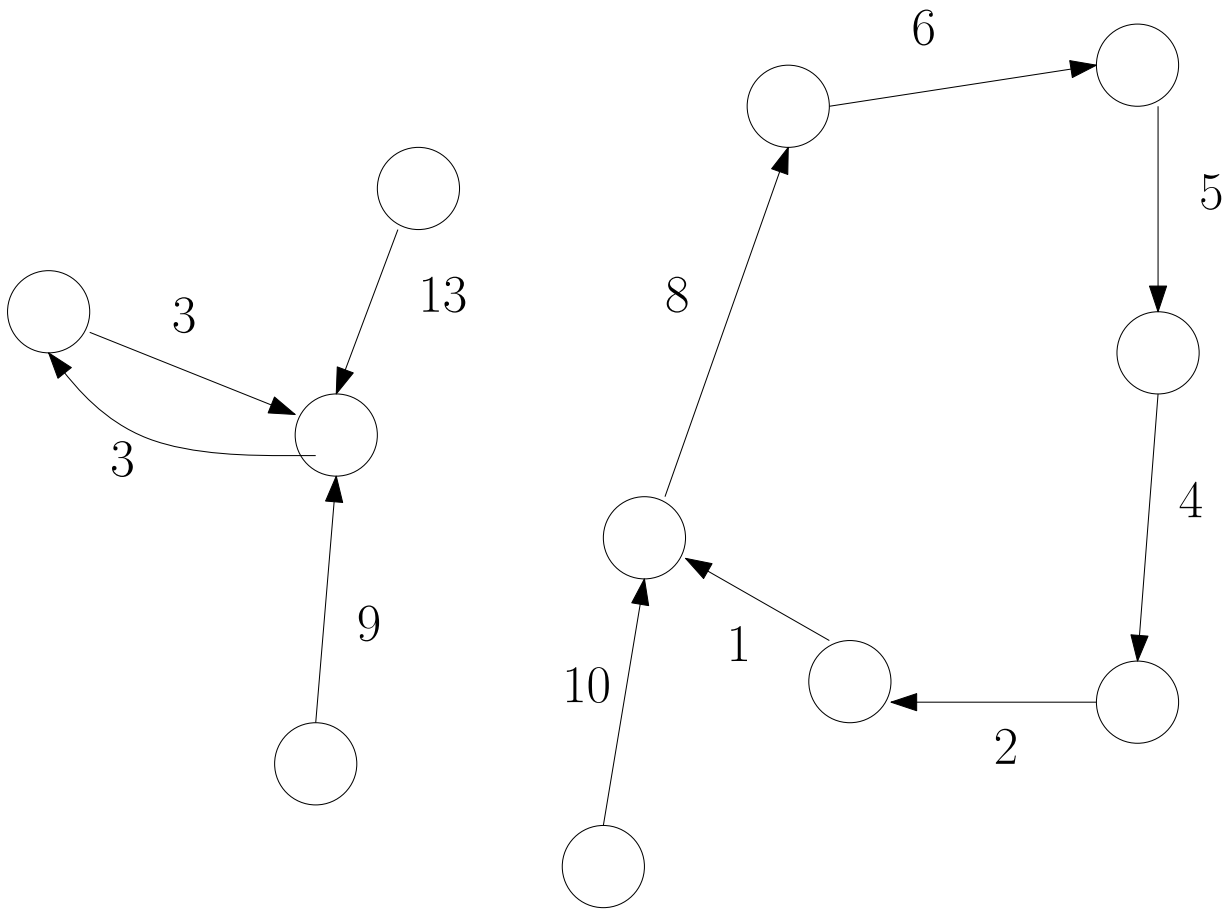
→ Consider the digraph $F$ on the fragments connected by MOE edges.

- The directions indicate which fragment wants to connect to which other fragment.
- Each fragment has exactly one outgoing edge.

→ Therefore, by Lemma 1, each weakly connected component of $F$ has exactly one cycle.

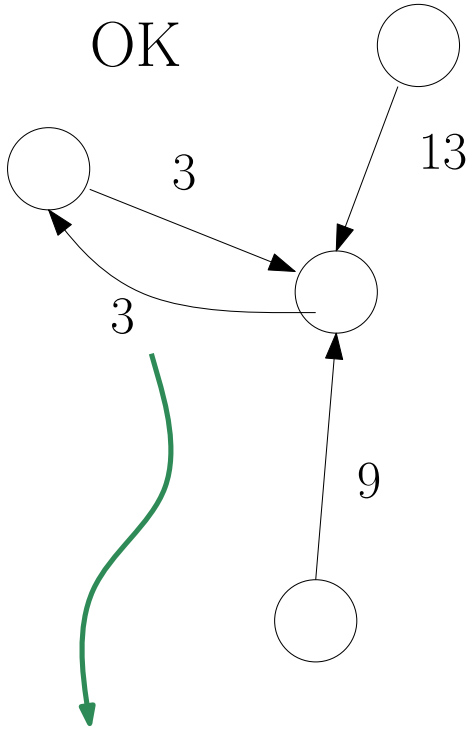
# Correctness

Consider cycles in $F$.

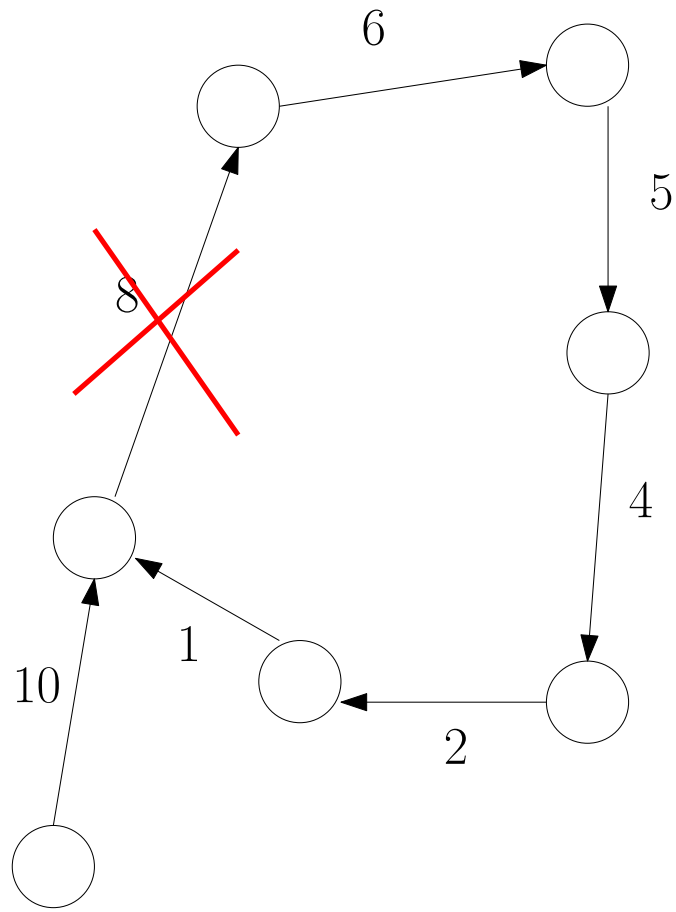


# Correctness

Consider cycles in $F$. They must contain exactly two fragments.




Note same weights



# Complexity Analysis

$\rightarrow$ $O(\log n)$ phases.

$\rightarrow$ Each phase takes $O(n)$ time.

$\Rightarrow$ Hence, time complexity is $O(n \log n)$.

$\rightarrow$ Each phase takes $O(n)$ messages (for convergecast and broadcast) plus the messages needed to find MOE.

$\rightarrow$ The latter takes a total of $O(|E|)$ for the entire algorithm, because, a node checks a neighbour at most once.

$\Rightarrow$ Hence total message complexity is $O(|E| + n \log n)$.

# Maximal Independent Set Problem

**Defn.** A set of nodes is called an independent set if it contains no pair of neighboring nodes.

**Defn.** An independent set is maximal if it cannot be increased to form a larger independent set by the addition of other nodes.

$\rightarrow$ Given an arbitrary network we want to find a maximal independent set of nodes.

$\rightarrow$ Each node must know if it is in MIS or not.

**Appln.** dominating sets, clusterhead selection ..

# High Level Idea

**Notation** Let $\Gamma(v)$ be the set of vertices in $V$ that are adjacent to $v$.

**Idea** The algorithm proceeds in rounds; In every round

1. find an independent set $S$,
2. add $S$ to $I$ (initially $I$ is empty) and
3. delete $S \cup \Gamma(S)$ from the graph.

# Luby's Algorithm I

1: Uncolour all vertices.

2: **for** $O(\log n)$ rounds **do**

3: Nodes with degree 0 are coloured red and removed.

4: Each node $v$ marks itself with prob $\frac{1}{2d(v)}$

5: If two endpoints of an edge are coloured, unmark lower degree endpoint (breaking ties arbitrarily).

6: Colour all marked nodes red

7: Colour neighbours of red nodes blue (in parallel)

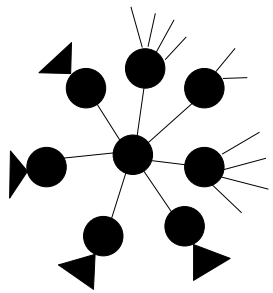
8: Remove coloured nodes.

9: **end for**

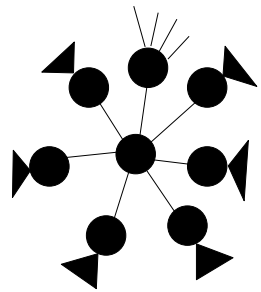
# Analysis

Key Idea: Constant fraction of edges removed each round (on exp.)

- $v \in V$ is good if at least $d(v)/3$ neighbors of degree no more than $d(v)$; otherwise the vertex is bad.



Good Vertex



Bad vertex

- An edge is good if at least one of its endpoints is a good vertex
- bad if both endpoints are bad vertices.
- We show that a good vertex is likely to have one of its lower degree neighbors in MIS and thereby deleted from $V$.

**Lemma**

*Let $v \in V$ be a good vertex with degree $d(v) > 0$. Then, the probability that some vertex $w \in \Gamma(v)$ gets marked is at least $1 - e^{-1/6}$.*

**Proof.**

- Each vertex $w \in \Gamma(v)$ is marked independently with probability $1/(2d(w))$.
- The probability that none of the neighbors of $v$ gets marked is at most

$$\left(1 - \frac{1}{2d(v)}\right)^{d(v)/3} \leq e^{-1/6}$$

□

Note: $1 - e^{-1/6} = 0.153518275$, which is small, but nevertheless a constant!

**Lemma**

*During any round, if a vertex $w$ is marked then it is coloured red with probability at least $1/2$.*

**Proof.**

- The only reason a marked vertex $w$ becomes unmarked is that one of its neighbors of degree at least $d(w)$ is also marked.
- The probability of this happening is

$$\sum_{x \in \Gamma(w)} \frac{1}{2d(x)} \leq \sum_{x \in \Gamma(w)} \frac{1}{2d(w)} = \frac{d(w)}{2d(w)} = 1/2$$

**Lemma**

*The probability that a good vertex is either coloured red or blue in one iteration is at least*

$$(1 - e^{-1/6})/2.$$

Note: $\frac{1-e^{-1/6}}{2} = 0.0767591376$, a constant!

**Lemma**

*In a graph $G = (V, E)$ the number of good edges is at least $|E|/2$.*

**Proof.**

- Direct the edges in $E$ from the lower degree end-point to the higher degree end-point breaking ties arbitrarily. Let $d_i(v)$ and $d_o(v)$ be the in-degree and out-degree of $v$.
- For each bad vertex $v$,

$$d_o(v) - d_i(v) \geq d(v)/3 = \frac{d_o(v) + d_i(v)}{3}$$

- Therefore,

$$d_o(v) \geq 2d_i(v). \tag{1}$$

- The number of edges going into a bad vertex is at most half the number of edges going out of that bad vertex.
- Therefore, the number of edges going into bad vertices is at most half the number of edges going out of bad vertices.

# Proof (contd.)

- Let $E(S, T)$ be the set of edges directed from vertices in $S$ to vertices in $T$; and $e(S, T) = |E(S, T)|$.
- Let $V_G$ and $V_B$ be the good and bad vertices, respectively.

$$|e(V_B, V_B)| + |e(V_G, V_B)| \leq \frac{1}{2}(|e(V_B, V_B)| + |e(V_B, V_G)|)$$

- Therefore,

$$|e(V_B, V_B)| \leq |e(V_B, V_G)|,$$

implying that # of bad edges are less than the # of good edges.

# The Final Touch

## Corollary

*On expectation, at least* $\frac{1-e^{-1/6}|E|}{4}$ *edges are deleted in each iteration. I.e.,* $|E| \times 0.0383795688$ *edges are deleted in each iteration.*

## Theorem

*Therefore, the expected number of iterations is* $O(\log n)$.

# Luby's Algorithm II

1: {Each node executes the following code.}

2: **while** not deleted **do**

3: Generate a random number $r$.

4: Send your random number to neighbours.

5: Receive neighbours' random numbers. Let $r'$ be the max among neighbours' random numbers.

6: **if** $r > r'$ **then**

7: Put self in MIS.

8: Send "delete" command to neighbours

9: **end if**

10: Delete self if "delete" command received.

11: **end while**


Thank you