Graph Algorithms

Slides are based on the book chapter from Distributed Computing: Principles, Algorithms and Systems (Chapter 5) by Kshemkalyani and Singhal

Pradipta De
pradipta.de@sunykorea.ac.kr
Few Basic Concepts

• Symmetric and Asymmetric algorithm
  – all processes execute same logical functions, while different processes execute different functions

• Anonymous algorithm: process identifiers are not used to make any execution decision
  – Structurally elegant, easy to run on anonymous systems

• Uniform algorithm: does not use the number of processes in the system (n) in the code
  – Easy to scale; processes can join/leave easily

• Adaptive algorithm: In an adaptive algorithm, the complexity of the algorithm is expressible as a function of number of participating entities (k), not the total number of entities (n), where (k <= n)
Few Basic Concepts

• Centralized and Distributed Algorithms
  – Centralized: roles of processes are asymmetric
  – Distributed: more balanced roles of nodes

• Synchronous/Asynchronous:
  – Synchronous: execution proceeds in rounds
    • Known upper bounds on message delays $\Rightarrow$ known upper bounds on time
      to complete a round
  – Asynchronous: Different timing and execution models $\Rightarrow$ harder to assume any coordination

• On-line/Offline algorithm:
  – On-line: executes as data is generated
  – Offline: requires all trace data before execution
Spanning Tree

• What is a tree?
  – An undirected graph in which any two vertices are connected by exactly one path and without any cycle

• What is a forest?
  – A disjoint union of trees

• What is a spanning tree?
  – A tree $T$ is a spanning tree of a connected graph $G$, if $T$ is a subgraph of $G$, and $T$ contains all the vertices of $G$
Synchronous 1-initiator Spanning Tree

- Symmetric and proceeds in rounds
- Assumes a designated root node, which is the initiator
- Root floods QUERY messages to determine tree edges

- Parent of a node is the node from which first QUERY is received
- If multiple QUERY received in same round, then choose parent randomly
- Termination: rounds = diameter of the graph
- Spanning tree obtained in a Breadth-First-Tree (BFS)
Asynchronous 1-initiator Spanning Tree

- Root initiates flooding of QUERY messages to determine tree edges
- Parent of a node is the first node from which a QUERY is received
  - ACCEPT sent to first QUERY
  - REJECT sent to other QUERY messages from non-parent nodes
- Each node terminates algorithm when it has received a response to QUERY from all its neighbors

1. A sends QUERY to B and F
2. F receives QUERY from A and determines that AF is a tree edge
   i. F → E and C
3a. E receives QUERY from F and determines FE as tree edge
   i. E → B and D
3b. C receives QUERY from F, marks FC as tree edge, forwards to B and D
   ….
Asynchronous Concurrent Initiator Spanning Tree

Options to extend the 1-initiator algos

- Merge the partially computed trees
  - Cannot be done based on local knowledge at a node

- Suppress the instance initiated by one root when conflict detected at a node.
  - Messages must be enhanced with root’s process id, and process id can be used to determine which initiator survives

Spanning tree construction initiated by A, G, and J at the same time
Use of Spanning Tree

• Useful for distributing and collecting information from all nodes
  – Broadcast: root sends information to be sent to all its children, and non-root nodes replicates the information to its children, till leaf nodes
  – Convergecast: collects information from all nodes and collects at the root
    • Initiated by leaf nodes of a tree in response to a request sent by root
Minimum Weight Spanning Tree (MST)

- If a graph $G$ is a weighted graph, then weight of a spanning tree $T$ is the sum of weights of the edges.

- Among all the spanning trees, one with the smallest weight is the MST.

- MST minimizes the cost of transmission from any node to any other node.
Centralized Algorithm

• Kruskal’s Algorithm:
  – List all edges of G, with n vertices, in non-decreasing weight
  – Select the smallest edge of G
  – Repeat the selection, s.t. no cycle is formed with already selected edges, till n-1 edges

• Prim’s Algorithm:
  – Start with n isolated vertices (each is a forest component)
  – Create a matrix (v_n x v_n) with edge weights
  – Start from any vertex, say v_1, and pick the smallest edge, thus extending the forest by adding, say v_k
  – Next, look at rows for 1 and k, and pick the smallest component
  – Repeat this till n-1 edges are selected

• General Idea:
  – Begins with a spanning forest of n nodes, and without any edges
  – Add a “minimum weight outgoing edge” (MWOE) between two components
  – In distributed algo
    • the addition of the edges should be done concurrently
    • Each node knows only the weight of edge to its neighbors
Synchronous GHS Algo for MST

- Follows Kruskal’s strategy \( \Rightarrow \) begins with forest of graph components and extends by adding MWOE
- Each component has a leader node in an iteration

**Key Steps in an iteration:**

- Broadcast-convergecast: leader identifies MWOE
- Broadcast: (potential) leader for next iteration identified
- Broadcast: among merging components, 1 leader selected; leader identifies itself to other components

![Diagram showing phases within an iteration in a component.](image)
Leader = i;
For round = 1 to log(n) do
  if (leader = i) then Broadcast SEARCH_MWOE(leader) along marked edges

  On receiving SEARCH_MWOE(leader) msg,
  each process, including leader, sends EXAMINE msg
  (determine if the other end of the edge is part of the same component)
  pick MWOE from all the edges

  Leaf nodes within a component starts convergecast to inform MWOE
  (REPLY_MWOE)

  If (leader = i) then
    wait for convergecast replies
    select minimum weight edge from all replies
    broadcast ADD_MWOE

  If an MWOE edge gets marked by two different components,
  Define new leader as the process with higher id
  New leader identifies itself as leader for next round, and broadcast
  NEW_LEADER msg
Asynchronous MST: Challenges

- Two nodes could be in different rounds \(\Rightarrow\) there must be a way to determine this.
- Search for MWOE at different levels must be coordinated.

- Instead of designing the asynchronous algorithm from start, alternative approach:
  - Block at a round to force synchronization if the levels do not match.
  - When a node gets involved in a new round, it informs all its neighbors \(\Rightarrow\) helps in blocking.
Synchronizers

• Is there a general technique to convert an algorithm for synchronous system to run on an asynchronous system?

• Synchronizers: class of transformation algorithms that allow synchronous algorithms to run on asynchronous systems
  – Assumptions: failure-free system

• Main idea: Synchronizer is a mechanism to notify each process when it is “safe” to move to the next round
  – Process i is safe in round r if all messages sent by i is received
  – Synchronizer signals to each process when it is sure that all messages to be received in the current round have been received
Notations

• Message Complexity of asynchronous algo
  \[ M_a = M_s + (M_{init} + \text{rounds} \times M_{round}) \]

• Time Complexity of asynchronous algo
  \[ T_a = T_s + (T_{init} + \text{rounds} \times T_{round}) \]

• \( M_s \): number of messages in synchronous algo
• \( \text{rounds} \): number of rounds in synchronous algo
• \( T_s \): time for sync algo
  \[ \text{Assume 1 unit per round, hence equal to rounds} \]
• \( M_{round} \): number of messages needed to simulate a round
• \( T_{round} \): number of sequential message hops required to simulate a round
• \( M_{init} \) and \( T_{init} \) are the messages and message hops at initialization
Simple Synchronizer

• A process sends each neighbor 1 message/round
  – dummy msg sent, if sync version has no msg to send
  – Combine msgs into one msg, if sync version sends multiple msgs in one round
• After receiving a message from each neighbor, a process moves to the next round
• Neighbors $P_i$ and $P_j$ can be at most 1 round apart
• Any process can start a round

• Since in each round a msg is sent on each incident link in each direction,
  – $M_{\text{round}} = 2|L|$ and $T_{\text{round}} = 1$
$\alpha$-synchronizer

- Any process $i$ moves to next round $r+1$ if all the neighboring processes are safe for round $r$
- Uses message acknowledgement to learn about safety
- When $P_j$ has received ack for each msg it has sent out, then it informs $P_i$ that it is safe.
\(\alpha\)-synchronizer

- **Message complexity**
  - For every msg sent in a round, an ack required \(\Rightarrow 2p\) messages \((p \leq |L|)\)
  - \(2|L|\) msgs per round required to inform neighbors that the process is safe
  - \(2|L| + 2p = O(|L|)\) msgs

- **Time complexity**: \(T_{\text{round}} = O(1)\)
\( \beta \) - synchronizer

- Assumes a rooted spanning tree
  - Overhead in initialization
    - \( O(n \log n + |L|) \) msg complexity, and \( O(n) \) time complexity

- Operation
  - Safe leaf nodes initiate a convergecast
  - Intermediate node propagates convergecast to its parent when all nodes in subtree, including itself, is safe
  - When root is safe and received convergecast from all children, it sends a tree broadcast to move to next round

- Complexity:
  - \( M_{\text{round}} = 2(n-1) \)
  - \( T_{\text{round}} = 2 \log n \) (avg case); \( 2n \) (worst case)
\( \gamma - \) synchronizer

- Network is organized as a set of clusters
- Each cluster has a spanning tree
  – \( \beta \) synchronizer within a cluster
  – \( \alpha \) synchronizer over designated inter-cluster edges
Leader Election

• Model assumptions
  – Asynchronous Ring topology
  – All processes have unique ids
Leader Election: LCR algo

• Each process sends its identifier to the left neighbor (⇒ “I want to be leader” msg)
• When a node $P_i$ receives identifier $k$ from right neighbor, $P_j$
  – $i < k$: forward identifier $k$ to left neighbor
  – $i > k$: ignore message
  – $i = k$: non-anonymous ring, therefore, $P_i$’s identifier has circulated ⇒ $P_i$ is leader ... send a msg declaring itself as leader
• Message complexity: $O(n^2)$
Improvement

• Binary search in each direction
  – Assumes a bidirectional ring
• In round $k$, token circulated to $2^k$ neighbors on left and right side
• Ring covered in log steps
• Message complexity $O(n \log n)$
  – In each round $n$ msgs are sent
• Is there an algorithm for anonymous rings?