Lecture 2: Basic Graph Algorithms

CS 539 / ECE 526

Distributed Algorithms

Some slides borrowed from Jennifer Welch’s class CSCE 668 at Texas A&M
Outline

• Simple broadcast using flooding
• Broadcast & convergecast assuming a spanning tree
• Find a spanning tree
• Breadth-first search
The Broadcast Problem

- Distributed processes / nodes
- One node has a piece of information M
- Want to send M to all nodes
- Assumption today:
  - Message passing
  - Generic graph that is connected
  - No link or node faults
  - Sync or async
Flooding Broadcast (Async)

- Initially, broadcaster sends M to all neighbors
- Upon receiving M for the first time
  
  Send to all neighbors

![Diagram of a network with nodes a, b, c, d, e, f, g, h connected by lines to represent the communication network.]
Flooding Broadcast (Lockstep Sync)

• Round 1: broadcaster sends M to all neighbors

• Round $r > 1$: if receiving M for the first time in the last round

Send to all neighbors
Flooding Broadcast Correctness

• Graph connected $\rightarrow$ exist a path from broadcaster $\rightarrow$ induction based on distance

• "Obvious"
Recall Efficiency Metrics

• Time complexity
  – For synchrony: # of rounds (aka round complexity)
  – Can be extended to asynchrony

• Communication complexity
  – Can be measured in # of msgs or bits

• Computation and space complexity
  – Not too different from non-distributed
  – Often ignored, will discuss when important
Flooding Broadcast Efficiency

• Round complexity:
  – $D$ (graph diameter)

• Communication complexity:
  – $2|E|$ msgs ($E$ is the set of edges)
    • 1 msg per direction per edge
  – Do not send $M$ back to those who sent $M$ to me?
    • Useful in some cases, but not always, e.g., complete graph
Outline

- Simple broadcast using flooding
- Broadcast & convergecast assuming a spanning tree
- Find a spanning tree
- Breadth-first search
Spanning Tree

• Let $G = (V, E)$ be a connected undirected graph

• $T = (V, E')$ is a spanning tree of $G$
  – $E'$ is a subset of $E$
  – $T$ is connected
  – $T$ has no cycles
Broadcast using a Spanning Tree

• Assume a spanning tree has been build, i.e., each node maintains parent & list of children

• Similar to flooding, send to all children instead of all neighbors

• Correctness: same

• Round complexity: depth of tree

• Msg complexity: |V| - 1

  – No longer has the factor 2
The Convergecast Problem

• Distributed processes / nodes

• Each node has a piece of information

• Want to **collect** all info at one node
  – Variants: sum, max, ...

• Assumption today: same
Convergecast via a Spanning Tree

• Dotted lines: non-tree edges

• Solid arrows: tree edges
Convergecast via a Spanning Tree

• Initially, each leaf node sends input to parent

• Upon receiving from all children
  Send to parent $f(m_1, m_2, ..., m_c)$
  (Showing async version here, sync version similar)

• Correctness: similar induction

• Efficiency: D rounds, $|V| - 1$ msgs
  – Bits depend on shape of spanning tree in general
  – $(|V| - 1)|M|$ for functions such as sum and max
Outline

• Simple broadcast using flooding

• Broadcast & convergecast assuming a spanning tree

• Find a spanning tree from specified root

• Breadth-first search
Spanning Tree from Specified Root

• Can augment the flooding algorithm
  – Record parent
  – Reply to parent so parent can record children
Spanning Tree via Flooding (Sync)

• Round 1: root sends recruit msg to all neighbors
• Round \( r > 1 \):
  If receiving a recruit msg in round \( r-1 \)
    If this is first time (*ties broken arbitrarily*)
      record parent & send yes to parent
      send recruit to all neighbors
  If receiving yes from node \( j \)
    Add \( j \) as a child
Spanning Tree via Flooding (Async)

- Initially: root sends recruit msg to all neighbors
- Upon receiving a recruit msg
  If this is first time (ties never occur in async)
    record parent & send yes to parent
    send recruit to all neighbors

Upon receiving yes from node j
Add j as a child
Spanning Tree from Specified Root

- Can augment the flooding algorithm
  - Record parent
  - Reply to parent so parent can record children

- When do nodes terminate?
  - Reply to non-parent so that it does not wait forever
  - Not a problem in sync, lack of yes = no
    (observation from student)
Spanning Tree via Flooding (Async)

• Initially: root sends recruit msg to all neighbors

• Upon receiving a recruit msg
  If this is first time (ties never occur in async)
  record parent & send yes to parent
  send recruit to all neighbors

  Else:
  reply with no

  Upon receiving yes from node j
  Add j as a child

  Upon receiving yes or no from all neighbors
  Terminate
Spanning Tree via Flooding (Async Detailed)

Initially: parent = NULL, children = {}, pending = neighbors;
for each j in neighbors
  root sends “recruit” to j

Upon receiving “recruit” from j
  If parent == NULL
    parent = j
    Send “yes” to j
    for each j in neighbors:
      Send “recruit” to j
  Else: send “no” to j
Upon receiving “yes” from j
  children = children U {j}
  pending = pending \ {j}
Upon receiving “no”
  pending = pending \ {j}
Upon pending = {}
  Terminate
Spanning Tree via Flooding

• Correctness:
  – Every node has at most one parent
  – Every node has a parent
    • Graph connected, induction on distance
  – Node i considers j parent if and only if j considers i child

• Efficiency: same as flooding broadcast
  – Round: Diameter of graph (+1)
  – Communication: 2|E| msgs, O(|E|) bits
Spanning Tree via Flooding

- Extra nice property under sync: breadth-first search
- Does not hold under async
BFS in Async

• One idea: synchronized (yes, in async) “wavefronts”
  – Root recruit distance-1 nodes (root’s neighbors)
  – Wait for confirmation from all of these
  – Recruit distance-2 nodes (root \(\rightarrow\) dist-1 \(\rightarrow\) dist-2)
  – Convergecast confirmations back to root
  – Recruit distance-3 nodes (root \(\rightarrow\) dist-1 \(\rightarrow\) dist-2 \(\rightarrow\) dist 3)
  – Convergecast confirmations back to root
  – ...

```
       a
      / \     0
     b   c   1
    / \   |
   d   e  f
   | \   |
  g   h  2
      3
```
BFS in Async Efficiency

• One idea: synchronized (yes, in async) “wavefronts”

• Comm complexity:
  – $O(|E| + |V| \times D)$
  – Each edge sees recruit once and yes/no once
  – $D$ broadcast / convergecast each costing (up to) $V$

• Round complexity:
  – $O(D^2)$, $D$ broadcast / convergecast each up to $V$ rounds
  • More accurately: $1 + 2 + 3 + \ldots + D$
Summary

• Broadcast & convergecast via a spanning tree (sync and async)

• Find a spanning tree from specified root using flooding (sync and async)

• Breadth-first search
  – Easy in sync, need synchronization in async