Distributed Algorithms
Algorithms in General Synchronous Networks

Maruth Goyal

UT Austin

Spring 2021

Table of Contents

Leader Election in General Networks

- Earlier, we looked at the problem of electing a leader in ring networks. What about general networks?

Leader Election in General Networks

- Earlier, we looked at the problem of electing a leader in ring networks. What about general networks?
- Multiple possible settings:

Leader Election in General Networks

- Earlier, we looked at the problem of electing a leader in ring networks. What about general networks?
- Multiple possible settings:
  1. Nodes know n, the number of nodes

Leader Election in General Networks

- Earlier, we looked at the problem of electing a leader in ring networks. What about general networks?
- Multiple possible settings:
  1. Nodes know n, the number of nodes
  2. Nodes know D, the "diameter" of the graph

Leader Election in General Networks

- Earlier, we looked at the problem of electing a leader in ring networks. What about general networks?
- Multiple possible settings:
  1. Nodes know n, the number of nodes
  2. Nodes know D, the "diameter" of the graph
  3. Nodes know some upper bound on these quantities

Leader Election in General Networks

Definition: Diameter

The diameter D of a graph $G = (V, E)$ is the maximum length of the shortest path between any pair of vertices $u, v \in V$.

- Consider setting in which diameter D is known to all vertices.

Leader Election in General Networks

Definition: Diameter

The diameter D of a graph $G = (V, E)$ is the maximum length of the shortest path between any pair of vertices $u, v \in V$.

- Consider setting in which diameter D is known to all vertices.
- Starting point: Can we adapt the LCR algorithm [Le Lann, 1977, Chang and Roberts, 1979] from last time?

Flooding

- How do we compute a distributed maximum in a **general network**?.

Flooding

- How do we compute a distributed maximum in a **general network**?.
- Idea: since we know D, just propagate ("flood") UIDs through network.

Flooding

- How do we compute a distributed maximum in a **general network**?.
- Idea: since we know D, just propagate ("flood") UIDs through network.

Algorithm: FloodMax

1. Every node keeps track of the max UID it has seen so far (initially its own)
2. At each round, send all outgoing neighbors max UID seen so far.
3. repeat for D rounds.
4. The node for whom the max UID is its own at the end elects itself as leader.

Flooding

- How do we compute a distributed maximum in a **general network**?.
- Idea: since we know D, just propagate ("flood") UIDs through network.

Algorithm: FloodMax

1. Every node keeps track of the max UID it has seen so far (initially its own)
2. At each round, send all outgoing neighbors max UID seen so far.
3. repeat for D rounds.
4. The node for whom the max UID is its own at the end elects itself as leader.

- Time complexity: $O(D)$
- Communication complexity: $O(n \cdot D)$

Table of Contents

BFS Setup

- Suppose a node wants to send a message to everyone on the network ("broadcast"), what's the most **time**-efficient way to do this?

BFS Setup

- Suppose a node wants to send a message to everyone on the network ("broadcast"), what's the most **time**-efficient way to do this?
  breadth-first spanning tree

Definition: Breadth-First Spanning Tree

Given a graph G and distinguished source node s, a breadth-first spanning tree is a spanning tree of G with root s such that every vertex at distance d from s in G is at depth d in the tree.

- Note: this is different from minimizing **communication** (that's **Minimum** spanning tree)

BFS Setup

1. Suppose in a network, a process n wants to compute the BFST with itself as root.

BFS Setup

1. Suppose in a network, a process n wants to compute the BFST with itself as root.
2. At the end, every node must have a populated `parent` field indicating its parent node in the tree. (the source node's is $\perp$)

BFS Setup

1. Suppose in a network, a process n wants to compute the BFST with itself as root.
2. At the end, every node must have a populated `parent` field indicating its parent node in the tree. (the source node's is $\perp$)
3. **Assumption**: graph is undirected, connected.
   - We will later relax undirectedness.

BFS Setup

1. Suppose in a network, a process n wants to compute the BFST with itself as root.
2. At the end, every node must have a populated `parent` field indicating its parent node in the tree. (the source node's is $\perp$)
3. **Assumption**: graph is undirected, connected.
   - We will later relax undirectedness.
4. Processes don't have knowledge of number of nodes, or diameter, etc.

BFS

1. Every node stores boolean state `marked`, initially false for everyone except source

BFS

1. Every node stores boolean state `marked`, initially false for everyone except source
2. At the start of round i, all nodes which were marked for the first time at round $i - 1$ send a $\textsc{Search}$ message to their neighbors.

Termination?

How does the source know the procedure has finished?

BFS

1. Every node stores boolean state `marked`, initially false for everyone except source
2. At the start of round i, all nodes which were marked for the first time at round $i-1$ send a $\textsc{Search}$ message to their neighbors.
3. Any node which receives a $\textsc{Search}$ message for the first time sets `marked` to `true`, and sets the sender as its `parent`.

Termination?

How does the source know the procedure has finished?

Idea

children send parents acknowledgements indicating they're done.

BFS

1. Every node stores boolean state `marked`, initially false for everyone except source

BFS

1. Every node stores boolean state `marked`, initially false for everyone except source
2. At the start of round i, all nodes which were marked for the first time at round $i - 1$ send a SEARCH message to their neighbors.
    1. If node was already marked, simply sends an ACK message back
    2. Once a node receives an ACK message from all of its children, it sends an ACK to its parent.

BFS

1. Every node stores boolean state `marked`, initially false for everyone except source
2. At the start of round i, all nodes which were marked for the first time at round $i - 1$ send a SEARCH message to their neighbors.
   1. If node was already marked, simply sends an ACK message back
   2. Once a node receives an ACK message from all of its children, it sends an ACK to its parent.
3. Any node which receives a SEARCH message for the first time sets `marked` to `true`, and sets the sender as its `parent`. The parent is then sent an ACK message.

BFS

1. Every node stores boolean state `marked`, initially false for everyone except source
2. At the start of round i, all nodes which were marked for the first time at round $i - 1$ send a SEARCH message to their neighbors.
   1. If node was already marked, simply sends an ACK message back
   2. Once a node receives an ACK message from all of its children, it sends an ACK to its parent.
3. Any node which receives a SEARCH message for the first time sets `marked` to `true`, and sets the sender as its `parent`. The parent is then sent an ACK message.
4. Terminate when root gets ACK from all children.

BFS

1. Every node stores boolean state marked, initially false for everyone except source
2. At the start of round i, all nodes which were marked for the first time at round $i - 1$ send a SEARCH message to their neighbors.
    1. If node was already marked, simply sends an ACK message back
    2. Once a node receives an ACK message from all of its children, it sends an ACK to its parent.
3. Any node which receives a SEARCH message for the first time sets marked to true, and sets the sender as its parent. The parent is then sent an ACK message.
4. Terminate when root gets ACK from all children.

- Time complexity: $O(D)$
- Communication complexity (# messages): $O(|E|)$

BFS: Directed graphs

What if the graph is directed?

BFS: Directed graphs

What if the graph is directed?

1. Every node stores boolean state `marked`, initially false for everyone except source
2. At the start of round i, all nodes which were marked for the first time at round $i - 1$ send a SEARCH message to their neighbors.
   1. If node was already marked, **simply sends an Ack message back**
   2. Once a node receives an ACK message from all of its children, it **sends an Ack to its parent**.
3. Any node which receives a SEARCH message for the first time sets `marked` to `true`, and sets the sender as its `parent`. **The parent is then sent an Ack message.**
4. Terminate when root gets ACK from all children.

BFS: Directed graphs

What if the graph is directed?

1. Every node stores boolean state `marked`, initially false for everyone except source
2. At the start of round i, all nodes which were marked for the first time at round $i-1$ send a SEARCH message to their neighbors.
   1. If node was already marked, **simply sends an Ack message back**
   2. Once a node receives an ACK message from all of its children, it **sends an Ack to its parent**.
3. Any node which receives a SEARCH message for the first time sets `marked` to `true`, and sets the sender as its `parent`. **The parent is then sent an Ack message.**
4. Terminate when root gets ACK from all children.

Issue

Cannot "simply" send message back in digraph

BFS: Directed Graph

- How do we send back message?
- What if we could broadcast a message to the network, but specify the intended recipient?
- Sounds oddly familiar ...

BFS: Directed Graph

- How do we send back message?
- What if we could broadcast a message to the network, but specify the intended recipient?
- Sounds oddly familiar . . . recursion! (kind of)

BFS: Directed Graph

- How do we send back message?
- What if we could broadcast a message to the network, but specify the intended recipient?
- Sounds oddly familiar ... recursion! (kind of)

Idea

Use BFS with node as root to send an ACK back to parent. Attach intended recipient to ACK message.

BFS: Directed Graphs

1. Every node stores boolean state `marked`, initially false for everyone except source
2. At the start of round i, all nodes which were marked for the first time at round $i-1$ send a $\textsc{Search}$ message to their neighbors.
    1. If node was already marked, send $\textsc{Ack}$ with sender as recipient using $\textrm{BFS}$
    2. Once a node receives an $\textsc{Ack}$ message from all of its children, it sends an $\textsc{Ack}$ with parent as recipient using $\textrm{BFS}$.
3. Any node which receives a $\textsc{Search}$ message for the first time sets `marked` to `true`, and sets the sender as its `parent`. The parent is then sent an $\textsc{Ack}$ message using $\textrm{BFS}$
4. Terminate when root gets $\textsc{Ack}$ from all children.

BFS: Directed Graphs

- Time Complexity: $O(D^2)$
  1. The diameter D corresponds to the height of the tree. Since we must now send messages back using calls to BFS, each level of the tree takes $O(D)$ extra time per call
  2. Note it's $O(D)$ per level since nodes at the same level can do the BFS in "parallel".
- Communication Complexity: $O(D^2 \cdot E)$
  1. In each round, at most E messages can be sent, and may be sent for the recursive BFS calls.
  2. We assume here the message vocabulary is expressive enough that concurrent BFS executions are combined into single messages for each channel.

Applications of BFS

- **Broadcast**: If a node wants to send a message to every other node on the network, it can simply attach it to the SEARCH messages it sends.

Applications of BFS

- **Broadcast**: If a node wants to send a message to every other node on the network, it can simply attach it to the $\textsc{Search}$ messages it sends.
- **Global Computaion**: To compute any *assosciative* and *commutative* function of the nodes' states, simply propagate state upwards from the leaves; parents then aggregate the inputs using the function to be computed and propagate the value upwards etc.

Applications of BFS

- **Broadcast**: If a node wants to send a message to every other node on the network, it can simply attach it to the SEARCH messages it sends.
- **Global Computaion**: To compute any *assosciative* and *commutative* function of the nodes' states, simply propagate state upwards from the leaves; parents then aggregate the inputs using the function to be computed and propagate the value upwards etc.
- **Leader Election**: All nodes run BFS, and then use Global Computation to compute the node with max UID. The node with max UID elects itself as leader.

Applications of BFS

- **Broadcast**: If a node wants to send a message to every other node on the network, it can simply attach it to the SEARCH messages it sends.
- **Global Computaion**: To compute any *assosciative* and *commutative* function of the nodes' states, simply propagate state upwards from the leaves; parents then aggregate the inputs using the function to be computed and propagate the value upwards etc.
- **Leader Election**: All nodes run BFS, and then use Global Computation to compute the node with max UID. The node with max UID elects itself as leader.
- **Diameter Computation**: All nodes run BFS, use the generated tree to find furthest node (can attach depth to SEARCH/ACK messages). Then use global computation to find global max (diameter). Can now use FLOODMAX.

Table of Contents

Shortest Path Setup

- Assume every edge now has a nonnegative weight.
- Every process knows the weight of all edges incident to it.
- Want to compute shortest-paths spanning tree with root node r.
  - Same as Breadth-first spanning tree in the case where all edge weights are equal.
- Minimizes longest communication time to any other node in network for broadcast.

Shrotest Paths

- Recall the classic **Bellman-Ford** algorithm

$$\delta(u, v) = \min_{u' \in \text{out}(u)} \left[\delta(u', v) + w(u, u')\right]$$

Shrotest Paths

- Recall the classic **Bellman-Ford** algorithm

$$\delta(u, v) = \min_{u' \in \text{out}(u)} \left[\delta(u', v) + w(u, u') \right]$$

- Consider almost natural extension to distributed algorithm:
    1. Each node stores $\delta_{(s,n)}$, its minimum known distance from source s to n. Initially $\delta_{(s,s)} = 0$ and $\delta_{(s,t)} = \infty$ for $s \neq t$.

Shrotest Paths

- Recall the classic **Bellman-Ford** algorithm

$$\delta(u, v) = \min_{u' \in \text{out}(u)} \left[\delta(u', v) + w(u, u')\right]$$

- Consider almost natural extension to distributed algorithm:
  1. Each node stores $\delta_{(s,n)}$, its minimum known distance from source s to n. Initially $\delta_{(s,s)} = 0$ and $\delta_{(s,t)} = \infty$ for $s \neq t$.
  2. At each round each node sends $\delta_{(s,n)}$ to all its neighbors.

Shrotest Paths

- Recall the classic **Bellman-Ford** algorithm

$$\delta(u, v) = \min_{u' \in \mathrm{out}(u)} \left[\delta(u', v) + w(u, u')\right]$$

- Consider almost natural extension to distributed algorithm:
  1. Each node stores $\delta_{(s,n)}$, its minimum known distance from source s to n. Initially $\delta_{(s,s)} = 0$ and $\delta_{(s,t)} = \infty$ for $s \neq t$.
  2. At each round each node sends $\delta_{(s,n)}$ to all its neighbors.
  3. At each round node performs relaxation

$$\delta_{(s,v)} := \min \left[\delta_{(s,v)}, \min_{u \in \in(v)} \left[\delta_{(s,u)} + w(u, v)\right]\right]$$

Accordingly, update the parent.

Shrotest Paths

- Recall the classic **Bellman-Ford** algorithm

$$\delta(u, v) = \min_{u' \in \text{out}(u)} \left[\delta(u', v) + w(u, u')\right]$$

- Consider almost natural extension to distributed algorithm:
  1. Each node stores $\delta_{(s,n)}$, its minimum known distance from source s to n. Initially $\delta_{(s,s)} = 0$ and $\delta_{(s,t)} = \infty$ for $s \neq t$.
  2. At each round each node sends $\delta_{(s,n)}$ to all its neighbors.
  3. At each round node performs relaxation

$$\delta_{(s,v)} := \min \left[\delta_{(s,v)}, \min_{u \in \in(v)} \left[\delta_{(s,u)} + w(u, v)\right]\right]$$

     Accordingly, update the parent.

  1. Time complexity: $O(V)$
  2. Communication complexity: $O(VE)$

Questions?

Thank You!

References

Chang, E. and Roberts, R. (1979).
An improved algorithm for decentralized extrema-finding in circular configurations of processes.
Commun. ACM, 22(5):281–283.

Le Lann, G. (1977).
Distributed systems-towards a formal approach.
In *IFIP congress*, volume 7, pages 155–160. Toronto.