Algorithms for Synchronous Networks

Etienne Renault

2 octobre 2020

https://www.lrde.epita.fr/~renault/teaching/algorep/

Problem Statement

In a previous lecture we only focused on the leader election problem in very simple synchronous networks (rings) . . .

Problem Statement

In a previous lecture we only focused on the leader election problem in very simple synchronous networks (rings) . . .

Let us complicate things !

More complex networks

More complex algorithms

Problem Statement

In a previous lecture we only focused on the leader election problem in very simple synchronous networks (rings) . . .

Let us complicate things !

More complex networks

More complex algorithms

In this lecture we consider any strongly connected network digraph with n nodes.

1 Leader Election

2 Breadth-First Search

3 Shortest Path

4 Minimum Spanning Trees

1 Leader Election

2 Breadth-First Search

3 Shortest Path

4 Minimum Spanning Trees

Flooding Algorithm : Informal

Pre-requisite :

- Processes have UIDs and use them only for comparison
- Processes known graph diameter D, i.e. $\forall i, j \; max \; distance(i, j)$

Flooding Algorithm : Informal

Pre-requisite :

- Processes have UIDs and use them only for comparison
- Processes known graph diameter D, i.e. $\forall i, j$ $max\ distance(i, j)$

Basic flooding algorithm :

- Every process maintains a record of the maximum UID it has seen so far
- At each round each process propagates this maximum on all its outgoing edge

Flooding Algorithm : Informal

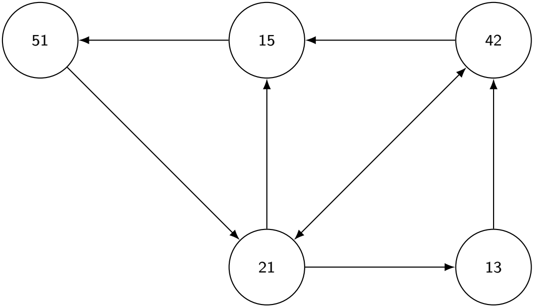
Pre-requisite :

- Processes have UIDs and use them only for comparison
- Processes known graph diameter D, i.e. $\forall i, j$ *max distance*(i, j)

Basic flooding algorithm :

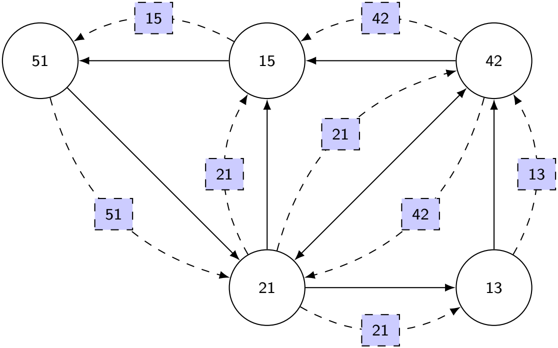
- Every process maintains a record of the maximum UID it has seen so far
- At each round each process propagates this maximum on all its outgoing edge
- After D rounds each process knows if it is the leader

Flooding Algorithm : Example



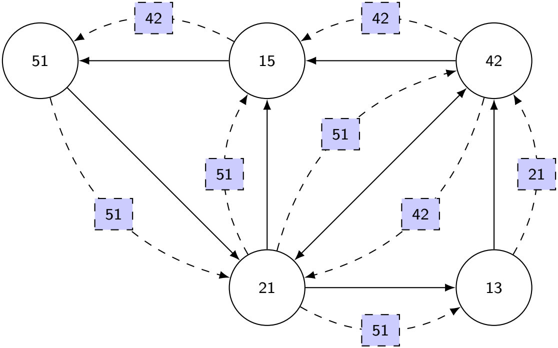
Initial State, $D = 3$, $n = 5$

Flooding Algorithm : Example



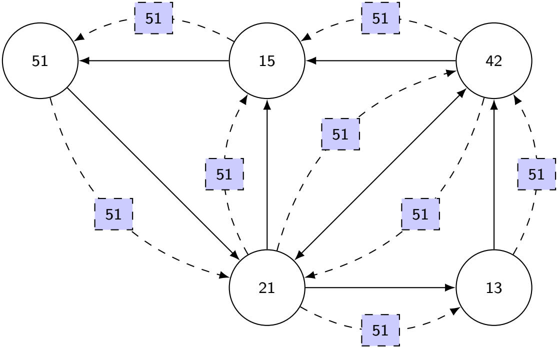
Round 1

Flooding Algorithm : Example



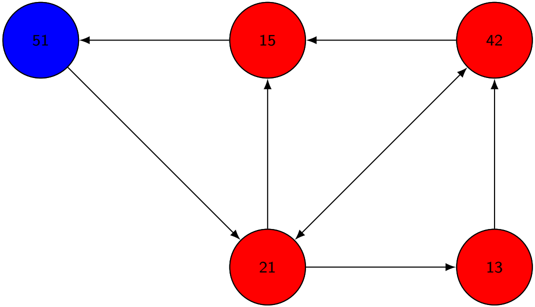
Round 2

Flooding Algorithm : Example



Round 3

Flooding Algorithm : Example



Complexity

- Time Complexity : D rounds, with D diameter
- Communication Complexity : $D \times |E|$, with $|E|$ the number of edges in the network

Complexity

- Time Complexity : D rounds, with D diameter
- Communication Complexity : $D \times |E|$, with $|E|$ the number of edges in the network

Upper Bound on the diameter

The algorithm still works if we consider D' as an upper bound of the diameter D

Improvements

- Process can send their max UID value only when they first learn about it
- When learning a new value from a bidirectionnal edge, there is no need to propagate back the information

Improvements

- Process can send their max UID value only when they first learn about it
- When learning a new value from a bidirectionnal edge, there is no need to propagate back the information

How to build a leader election without knowing the diameter ?

Improvements

- Process can send their max UID value only when they first learn about it
- When learning a new value from a bidirectionnal edge, there is no need to propagate back the information

How to build a leader election without knowing the diameter ?
In other words, how to detect the diameter ?

1 Leader Election

2 Breadth-First Search

3 Shortest Path

4 Minimum Spanning Trees

Breadth-First Search : Informal

We want to build the directed spanning tree for the network

Breadth-First Search : Informal

We want to build the directed spanning tree for the network

Pre-requisite :

- The network is strongly connected
- A source node is distinguish i_0

Breadth-First Search : Informal

We want to build the directed spanning tree for the network

Pre-requisite :

- The network is strongly connected
- A source node is distinguish i_0

Basic breadth-first search algorithm :

- Mark i_0
- Process i_0 send a *search* message at round 1 to all its neighbours
- At every round if a non-marked process receive a *send* message

Breadth-First Search : Informal

We want to build the directed spanning tree for the network

Pre-requisite :

- The network is strongly connected
- A source node is distinguish i_0

Basic breadth-first search algorithm :

- Mark i_0
- Process i_0 send a *search* message at round 1 to all its neighbours
- At every round if a non-marked process receive a *send* message
  - Mark itself

Breadth-First Search : Informal

We want to build the directed spanning tree for the network

Pre-requisite :

- The network is strongly connected
- A source node is distinguish i_0

Basic breadth-first search algorithm :

- Mark i_0
- Process i_0 send a *search* message at round 1 to all its neighbours
- At every round if a non-marked process receive a *send* message
  - Mark itself
  - Designates one process from which it received search as its parent

Breadth-First Search : Informal

We want to build the directed spanning tree for the network

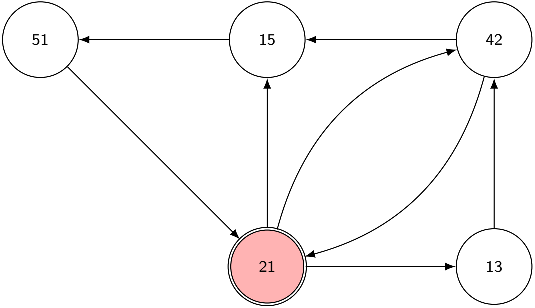
Pre-requisite :

- The network is strongly connected
- A source node is distinguish i_0

Basic breadth-first search algorithm :

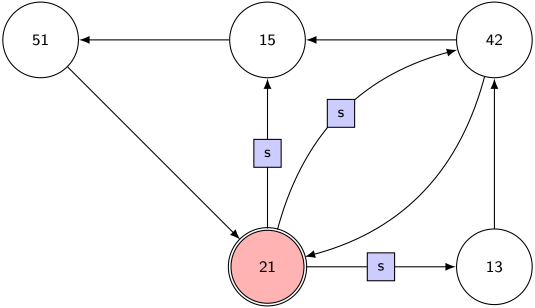
- Mark i_0
- Process i_0 send a *search* message at round 1 to all its neighbours
- At every round if a non-marked process receive a *send* message
  - Mark itself
  - Designates one process from which it received search as its parent
  - send a *search* message to all its neighbours

Example



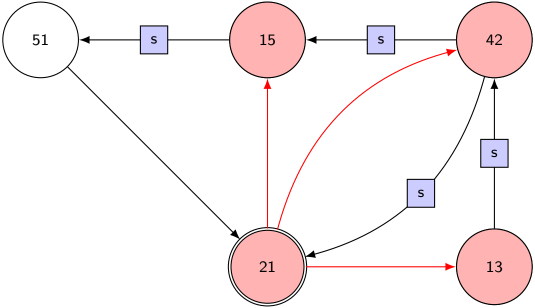
Initial State

Example



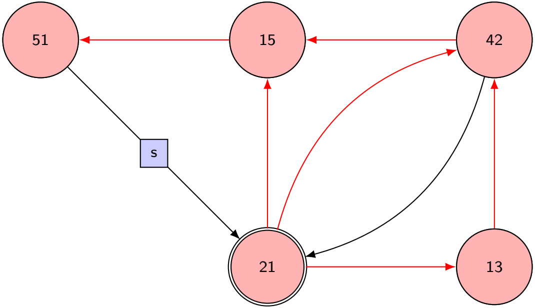
Round 0

Example



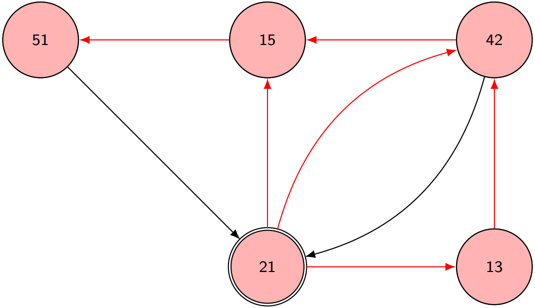
Round 1

Example



Round 2

Example



Complexity

- Time : D rounds, with D the diameter of the network
- Number of messages $|E|$, with $|E|$ the number of edges

Broadcast (and Piggybacking)

If a process wants to broadcast an information *m* :

- Initiate an execution of the Breadth-First Search algorithm
- Add to the *search* message the information *m*
- Other processes propagates *m* with their own *search* messages

All process get eventually notified !

Children Pointers

Sometimes knowing its children is important !
(Reuse the spanning tree)

Children Pointers

Sometimes knowing its children is important !
(Reuse the spanning tree)

Bi-directional communications :

- Easy : When choosing a parent, notify it
- Overhead : One message per son/child link

Uni-directional communications :

- Tricky since messages can be send via indirect routes
- Messages should carry the UID of the sender and information about *parent/non-parent*

Termination

How a process is notified that its BFS is finished ?

Termination

How a process is notified that its BFS is finished ?

- Every *send* message contains the UID of the process that has initiated the BFS
- Every *send* message is answered with an *parent/non-parent* message.

Termination

How a process is notified that its BFS is finished ?

- Every *send* message contains the UID of the process that has initiated the BFS
- Every *send* message is answered with an *parent/non-parent* message.
- The process wait that :

Termination

How a process is notified that its BFS is finished ?

- Every *send* message contains the UID of the process that has initiated the BFS
- Every *send* message is answered with an *parent/non-parent* message.
- The process wait that :
  - all its *send* messages have been replied

Termination

How a process is notified that its BFS is finished ?

- Every *send* message contains the UID of the process that has initiated the BFS
- Every *send* message is answered with an *parent/non-parent* message.
- The process wait that :
  - all its *send* messages have been replied
  - all its children have sent a completion message from all its children.

Termination

How a process is notified that its BFS is finished ?

- Every *send* message contains the UID of the process that has initiated the BFS
- Every *send* message is answered with an *parent/non-parent* message.
- The process wait that :
  - all its *send* messages have been replied
  - all its children have sent a completion message from all its children.
- then it sends a *completion* message to its parents

Termination

How a process is notified that its BFS is finished ?

- Every *send* message contains the UID of the process that has initiated the BFS
- Every *send* message is answered with an *parent/non-parent* message.
- The process wait that :
  - all its *send* messages have been replied
  - all its children have sent a completion message from all its children.
- then it sends a *completion* message to its parents

Homework

What is the complexity with these new messages ?

Breadth-First Search : Leader Election

1. Every process initiate a Breadth-First Search with itself as i_0

Breadth-First Search : Leader Election

1. Every process initiate a Breadth-First Search with itself as i_0
2. Every *search* message carry it's own UID

Breadth-First Search : Leader Election

1. Every process initiate a Breadth-First Search with itself as i_0
2. Every *search* message carry it's own UID
3. Every process uses its own BFS tree, child computation, termination detection to detect the maximum UID

Breadth-First Search : Leader Election

1. Every process initiate a Breadth-First Search with itself as i_0
2. Every *search* message carry it's own UID
3. Every process uses its own BFS tree, child computation, termination detection to detect the maximum UID
4. The process with the maximum UID declares itself as the leader

Breadth-First Search : Leader Election

1. Every process initiate a Breadth-First Search with itself as i_0
2. Every *search* message carry it's own UID
3. Every process uses its own BFS tree, child computation, termination detection to detect the maximum UID
4. The process with the maximum UID declares itself as the leader
5. All the other processes declare themselves as non-leader

Breadth-First Search : Leader Election

1. Every process initiate a Breadth-First Search with itself as i_0
2. Every *search* message carry it's own UID
3. Every process uses its own BFS tree, child computation, termination detection to detect the maximum UID
4. The process with the maximum UID declares itself as the leader
5. All the other processes declare themselves as non-leader

Complexity

- **undirected graph** : Time :$O(|D|)$; messages $O(n \times |D|)$
- **directed graph** :
  - $O(D)$ time : many BFS can run in parallel
  - $O(n \times |D|)$ messages : messages can be collapsed together

Breadth-First Search : Computing diameter

Breadth-First Search : Computing diameter

Each process construct the tree and use the termination acknowledgement to propagate distance from leaf to root

1 Leader Election

2 Breadth-First Search

3 Shortest Path

4 Minimum Spanning Trees

Problem Statement

- Generalisation of the BFS problem
- Communications can be uni-directional or bi-directional
- Each edge is associated to a weight
- The weight of a path is defined as the weight of each edge along the path

How to find the shortest path between two nodes ?

In other words, with some i_0 designated process,
what is the minimal distance of a node ?

Bellman-Ford : Informal

- Each process i keeps track of *dist* the shortest distance from i_0 it knows

Bellman-Ford : Informal

- Each process *i* keeps track of *dist* the shortest distance from i_0 it knows
- Initially $dist_{i_0} = 0$ and for $i \neq 0$, $dist_i = \infty$

Bellman-Ford : Informal

- Each process i keeps track of *dist* the shortest distance from i_0 it knows
- Initially $dist_{i_0} = 0$ and for $i \neq 0$, $dist_i = \infty$
- At each round, each process sends its *dist* to all its outgoing egdes

Bellman-Ford : Informal

- Each process i keeps track of $dist$ the shortest distance from i_0 it knows
- Initially $dist_{i_0} = 0$ and for $i \neq 0$, $dist_i = \infty$
- At each round, each process sends its $dist$ to all its outgoing egdes
- Then each process i updates $dist$ with all the $dist_j + weight_j$ received

Bellman-Ford : Informal

- Each process i keeps track of $dist$ the shortest distance from i_0 it knows
- Initially $dist_{i_0} = 0$ and for $i \neq 0$, $dist_i = \infty$
- At each round, each process sends its $dist$ to all its outgoing egdes
- Then each process i updates $dist$ with all the $dist_j + weight_j$ received
- When dist changed, each node update its parent field

Bellman-Ford : Informal

- Each process i keeps track of *dist* the shortest distance from i_0 it knows
- Initially $dist_{i_0} = 0$ and for $i \neq 0$, $dist_i = \infty$
- At each round, each process sends its *dist* to all its outgoing egdes
- Then each process i updates *dist* with all the $dist_j + weight_j$ received
- When dist changed, each node update its parent field
- After $n - 1$ rounds *dist* contains the shortest distance, and the field *parent* contains the parent in the shortest path tree

Example

Complexity

- Time : $n-1$
- Message : $(n-1) \times |E|$, with $|E|$ the number of edges in the network

A false assumption is to believe that
the algorithm stabilize after $|D|$ rounds

1 Leader Election

2 Breadth-First Search

3 Shortest Path

4 Minimum Spanning Trees

Problem Statement

How to find the minimum/maximum spanning tree ?

A minimum-weight spanning tree minimizes the total cost for any source process to communicate with all the other process in the network.

Let us assume that n is known

Some Definitions

Consider a graph $G = (V, E)$

Spanning Forest

A forest that consists of undirected edges of E such that every vertex V of G is covered.

Spanning Tree

A spanning forest of G such with every edges connected

Basic (non-distributed) construction

1. Start we a trivial spanning forest with n nodes and no edges
2. Repeatedly
   - Select a component C in the forest
   - Chose arbitrary outgoing edge e from C having minimum weight among all outgoing edges of C
   - Combine C with the component at the other end of e
   - Include e in the new combined-component
3. Stop when there is only one component

Basic (non-distributed) construction

1. Start we a trivial spanning forest with n nodes and no edges
2. Repeatedly
   - Select a component C in the forest
   - Chose arbitrary outgoing edge e from C having minimum weight among all outgoing edges of C
   - Combine C with the component at the other end of e
   - Include e in the new combined-component
3. Stop when there is only one component

Cycle Problem

If all edges have distinct weights there is exactly one MST for G

Distributed Algorithm : Intuition

Assume every edge have different weights, and undirected graph

- Tribute to Gallager, Humblet and Spira
- The algorithm builds the components in *levels*
- For each k, the level k components constitute a spanning forest that is a sub-graph of the MST
- Each level k component have at least 2^k nodes

Distributed Algorithm : Informal

1. Level 0 : components consisting of individual nodes and no edges
2. To get level $k + 1$:
   - Each component search along its spanning tree the outgoing edge with the minimum weight
   - The leader of each component communicate with the selected component to mark the edge as being in the new tree
   - A new leader must be chosen for the new component and propagate through the new component
3. Stop when there only one component

To detect if two nodes are in the same component
tests messages are sent along edges

Complexity

- **Time** : $O(n \log(n))$
  - At most $\log n$ level
  - Each level takes $O(n)$

- **Communication** : $O((n + |E|) \times \log(n))$
  - $O(n)$ messages sent per level
  - $O(|E|)$ additional messages to find local minimum weights per level

Communication can be reduced to $O(n \log(n) + |E|)$ by using a more careful strategy to find minimum weight edges.
Idea : remember edges going into the same component.

Remarks

Non-unique edge weight

We can define a lexicographic order using UID of processes

Leader Election

When building the MST a leader is elected naturally !