Synchronous single initiator spanning tree algorithm using flooding

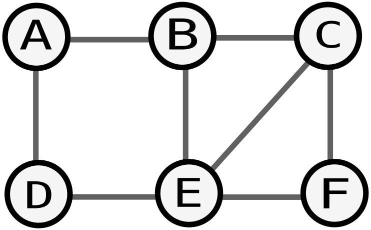
Siddharth Bhat, Anurag Chaturvedi, Hitesh Kaushik

March 13, 2020

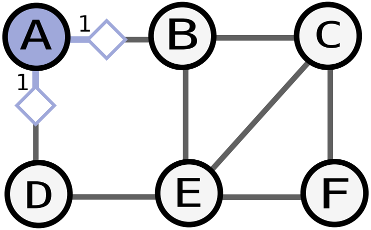
Introduction

- We use BFS to compute a spanning tree of a graph.

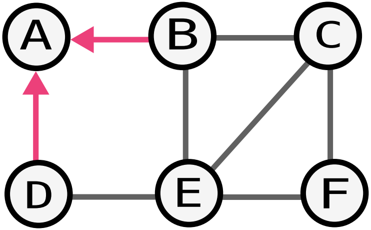
Example



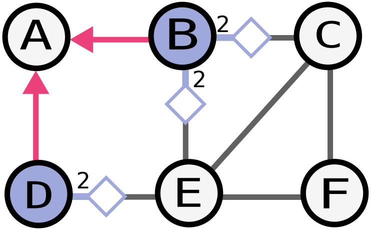
Example



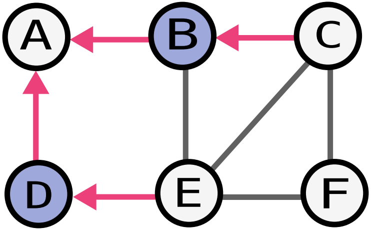
Example



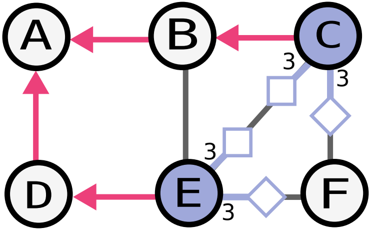
Example



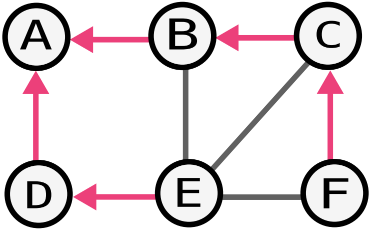
Example



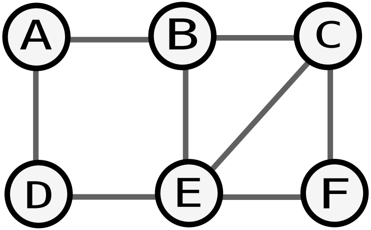
Example



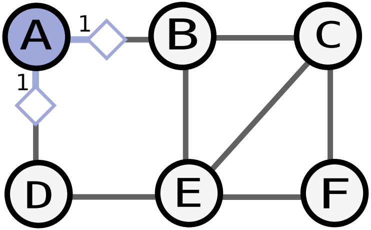
Example



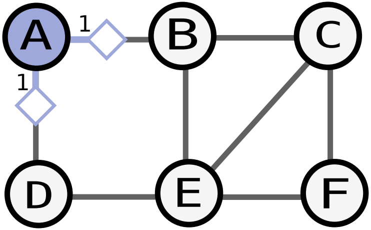
Why Synchrony?



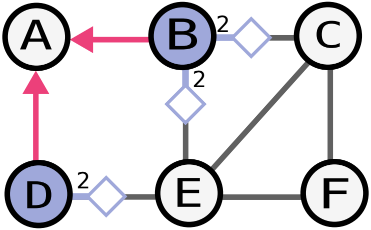
Why Synchrony?



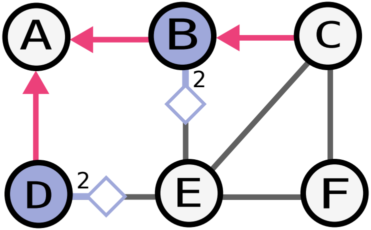
Why Synchrony?



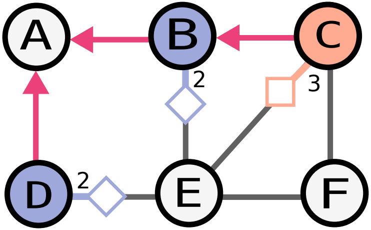
Why Synchrony?



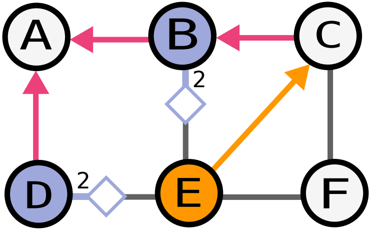
Why Synchrony?



Why Synchrony?



Why Synchrony?



Synchronous BFS (Pseudocode)

- Assume root begins computation.
- Algorithm is synchronous.

Synchronous BFS (Pseudocode)

- Assume root begins computation.
- Algorithm is synchronous.

```
def bfs_spanning_tree(self):
```

Synchronous BFS (Pseudocode)

- Assume root begins computation.
- Algorithm is synchronous.

```
def bfs_spanning_tree(self):

  if self.id == ROOT_ID:
    self.visited = True; self.depth = 0;
    for n in self.neighbours: n.send(self.id)
```

Synchronous BFS (Pseudocode)

- Assume root begins computation.
- Algorithm is synchronous.

```
def bfs_spanning_tree(self):

  if self.id == ROOT_ID:
    self.visited = True; self.depth = 0;
    for n in self.neighbours: n.send(self.id)

  for round in range(1, DIAMETER+1):
    if not self.visited: # if visited, skip
```

Synchronous BFS (Pseudocode)

- Assume root begins computation.
- Algorithm is synchronous.

```
def bfs_spanning_tree(self):

  if self.id == ROOT_ID:
    self.visited = True; self.depth = 0;
    for n in self.neighbours: n.send(self.id)

  for round in range(1, DIAMETER+1):
    if not self.visited: # if visited, skip

      if self.queries: # if we have a query
        # randomly choose from queries
        parent = random.choice(self.query)
        self.visited = True
        self.depth = round
```

Synchronous BFS (Pseudocode)

- Assume root begins computation.
- Algorithm is synchronous.

```
def bfs_spanning_tree(self):

  if self.id == ROOT_ID:
    self.visited = True; self.depth = 0;
    for n in self.neighbours: n.send(self.id)

  for round in range(1, DIAMETER+1):
    if not self.visited: # if visited, skip

      if self.queries: # if we have a query
        # randomly choose from queries
        parent = random.choice(self.query)
        self.visited = True
        self.depth = round

        # synchronous
        for n in self.neighbours: n.send(self.id)
    self.queries = [];
```

Synchronous BFS (Ending earlier if visited)

```
def bfs_spanning_tree(self):

  if self.id == ROOT_ID:
    self.depth = 0;
    for n in self.neighbours: n.send(self.id)

    return # early-exit for root node

  for round in range(1, DIAMETER+1):

      if self.queries: # if we have a query
        # randomly choose from queries
        parent = random.choice(self.query)
        self.visited = True
        self.depth = round

        # synchronous
        for n in self.neighbours: n.send(self.id)

        return # early-exit for child
```

Synchronous BFS (Learning children)

- Assume root begins computation.
- Algorithm is synchronous.

```
def bfs_spanning_tree(self):
  if self.id == ROOT_ID:
    self.visited = True; self.depth = 0;
    for n in self.neighbours: n.send(self.id)
  for round in range(1, DIAMETER+1):
    if self.visited: # if visited, wait for children
      for q in self.queries: self.children.append(q)
    else: # if not visited, run code
      if self.queries: # if we have a query
        # randomly choose from queries
        parent = random.choice(self.query)
        self.visited = True
        self.depth = round
        # synchronous
        for n in self.neighbours: n.send(self.id)
        parent.send(self.id) # send to parent
    self.queries = [];
```

Complexities

Complexities

- Local space for a: $|\{v : (a, v) \in E\}|$ (#of incident edges)

Complexities

- Local space for a: $|\{v : (a, v) \in E\}|$ (#of incident edges)
- $|Diameter|$ rounds.

Complexities

- Local space for a: $|\{v : (a, v) \in E\}|$ (#of incident edges)
- $|Diameter|$ rounds.
- 1 or 2 messages / edge. Message complexity $\leq 2|E|$.

Thank you!

Asynchronous Bounded Delay Network

- All processes have physical clocks: **need not be synchronized**.
- Message delivery time is bounded by constant $\mu \in \mathbb{R}$.

ABD Synchronizers: Bounded Delay $\rightarrow$ Synchronized

- All processes have physical clocks: **need not be synchronized**.
- Message delivery time is bounded by constant $\mu \in \mathbb{R}$.

Key idea

Chunk "real time" into units of μ. Each μ block of time behaves like a logical synchronized tick!