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.

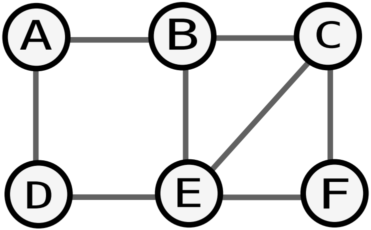
Introduction

- We use BFS to compute a spanning tree of a graph.
- We can distribute the sequential algorithm assuming *synchronous communication*.

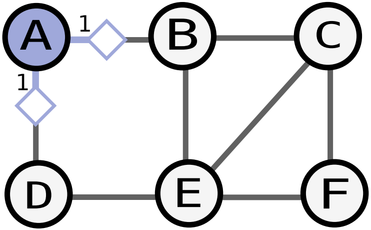
Introduction

- We use BFS to compute a spanning tree of a graph.
- We can distribute the sequential algorithm assuming *synchronous communication*.
- **Refresher:** Algorithm proceeds in rounds. All messages sent at round i are received at round i.

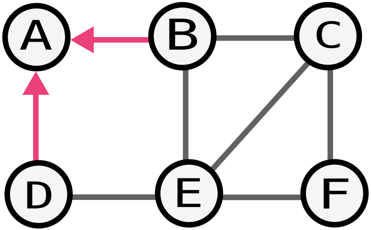
Example



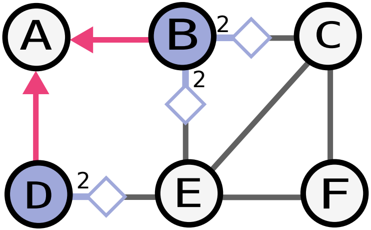
Example



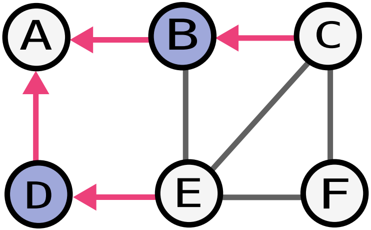
Example



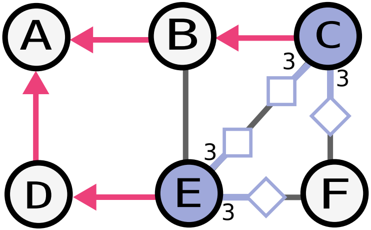
Example



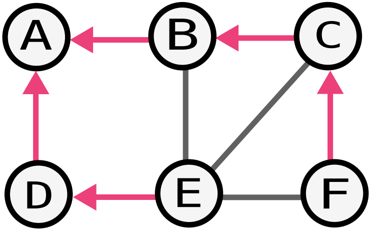
Example



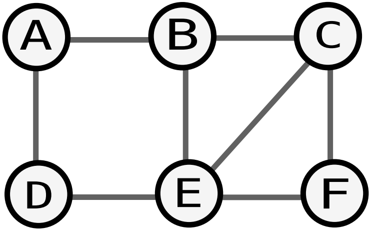
Example



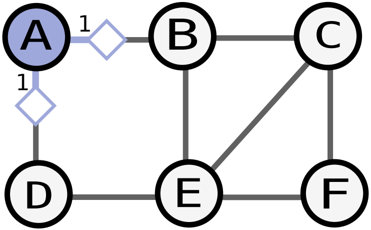
Example



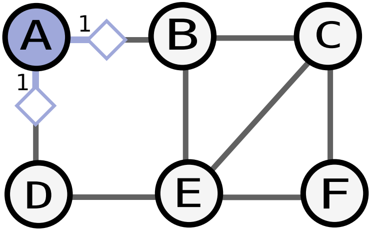
Why Synchrony?



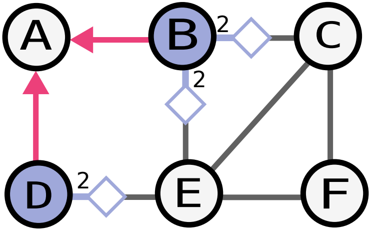
Why Synchrony?



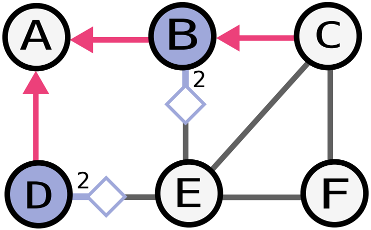
Why Synchrony?



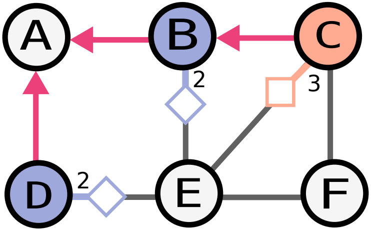
Why Synchrony?



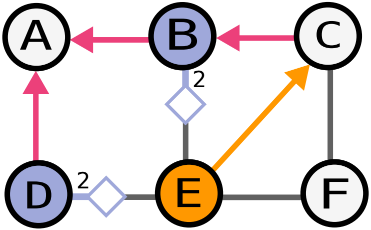
Why Synchrony?



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!

Formal definition of synchronous processes p

- $p \equiv (S, I, M\mathcal{G}, \vdash)$

Formal definition of synchronous processes p

- $p \equiv (S, I, M\mathcal{G}, \vdash)$
- $S \equiv$ set of states.

Formal definition of synchronous processes p

- $p \equiv (S, I, M\mathcal{G}, \vdash)$
- $S \equiv$ set of states.
- $I \subseteq S \equiv$ set of initial states.

Formal definition of synchronous processes p

- $p \equiv (S, I, M\mathcal{G}, \vdash)$
- $S \equiv$ set of states.
- $I \subseteq S \equiv$ set of initial states.
- $M\mathcal{G} : S \to 2^M \equiv$ message generating **function**.

Formal definition of synchronous processes p

- $p \equiv (S, I, M\mathcal{G}, \vdash)$
- $S \equiv$ set of states.
- $I \subseteq S \equiv$ set of initial states.
- $M\mathcal{G} : S \rightarrow 2^M \equiv$ message generating **function**.
- $\vdash \quad \subseteq S \times 2^M \subset S \equiv$ transition **relation**. We write $(s, M) \vdash s'$ for $(s, M, s') \in \vdash$.

Formal definition of synchronous processes p

- $p \equiv (S, I, M\mathcal{G}, \vdash)$
- $S \equiv$ set of states.
- $I \subseteq S \equiv$ set of initial states.
- $M\mathcal{G} : S \to 2^M \equiv$ message generating **function**.
- $\vdash \;\; \subseteq S \times 2^M \subset S \equiv$ transition **relation**. We write $(s, M) \vdash s'$ for $(s, M, s') \in \vdash$.
- Process starts at state $s_0 \in I$.

Formal definition of synchronous processes p

- $p \equiv (S, I, M\mathcal{G}, \vdash)$
- $S \equiv$ set of states.
- $I \subseteq S \equiv$ set of initial states.
- $M\mathcal{G} : S \to 2^M \equiv$ message generating **function**.
- $\vdash \;\subseteq S \times 2^M \subset S \equiv$ transition **relation**. We write $(s, M) \vdash s'$ for $(s, M, s') \in \vdash$.
- Process starts at state $s_0 \in I$.
- in state s, process sends a collection of messages $M\mathcal{G}(s)$.

Formal definition of synchronous processes p

- $p \equiv (S, I, M\mathcal{G}, \vdash)$
- $S \equiv$ set of states.
- $I \subseteq S \equiv$ set of initial states.
- $M\mathcal{G} : S \to 2^M \equiv$ message generating **function**.
- $\vdash \;\; \subseteq S \times 2^M \subset S \equiv$ transition **relation**. We write $(s, M) \vdash s'$ for $(s, M, s') \in \vdash$.
- Process starts at state $s_0 \in I$.
- in state s, process sends a collection of messages $M\mathcal{G}(s)$.
- **Before** next pulse, it receives *all* messages M sent to it in this pulse.

Formal definition of synchronous processes p

- $p \equiv (S, I, M\mathcal{G}, \vdash)$
- $S \equiv$ set of states.
- $I \subseteq S \equiv$ set of initial states.
- $M\mathcal{G} : S \to 2^M \equiv$ message generating **function**.
- $\vdash \ \subseteq S \times 2^M \subset S \equiv$ transition **relation**. We write $(s, M) \vdash s'$ for $(s, M, s') \in \vdash$.
- Process starts at state $s_0 \in I$.
- in state s, process sends a collection of messages $M\mathcal{G}(s)$.
- **Before** next pulse, it receives *all* messages M sent to it in this pulse.
- It then enters a state s' such that $(s, M) \vdash s'$.

Formal definition of synchronous processes p

- $p \equiv (S, I, M\mathcal{G}, \vdash)$
- $S \equiv$ set of states.
- $I \subseteq S \equiv$ set of initial states.
- $M\mathcal{G} : S \to 2^M \equiv$ message generating **function**.
- $\vdash \ \subseteq S \times 2^M \subset S \equiv$ transition **relation**. We write $(s, M) \vdash s'$ for $(s, M, s') \in \vdash$.
- Process starts at state $s_0 \in I$.
- in state s, process sends a collection of messages $M\mathcal{G}(s)$.
- **Before** next pulse, it receives *all* messages M sent to it in this pulse.
- It then enters a state s' such that $(s, M) \vdash s'$.
- Note that $M\mathcal{G}$ is **deterministic**. All the nondeterminism is in the relation $(s, M) \vdash s'$.

Formal definition of synchronous system $\mathbb{P}$

- $\mathbb{P} \equiv (p_1, p_2, \ldots, p_n)$ where each $p_i \equiv (S_i, I_i, M\mathcal{G}_i, \vdash_i)$ is a synchronous process, is a synchronous system.

Formal definition of synchronous system $\mathbb{P}$

- $\mathbb{P} \equiv (p_1, p_2, \ldots, p_n)$ where each $p_i \equiv (S_i, I_i, M\mathcal{G}_i, \vdash_i)$ is a synchronous process, is a synchronous system.
- Transition system $Tr \equiv (S, \mathcal{I}, \rightarrow)$.

Formal definition of synchronous system $\mathbb{P}$

- $\mathbb{P} \equiv (p_1, p_2, \ldots, p_n)$ where each $p_i \equiv (S_i, I_i, M\mathcal{G}_i, \vdash_i)$ is a synchronous process, is a synchronous system.
- Transition system $Tr \equiv (S, \mathcal{I}, \rightarrow)$.
- $S \equiv \prod_i S_i$ is the state space of the transition system.

Formal definition of synchronous system $\mathbb{P}$

- $\mathbb{P} \equiv (p_1, p_2, \ldots, p_n)$ where each $p_i \equiv (S_i, I_i, M\mathcal{G}_i, \vdash_i)$ is a synchronous process, is a synchronous system.
- Transition system $Tr \equiv (S, \mathcal{I}, \rightarrow)$.
- $S \equiv \prod_i S_i$ is the state space of the transition system.
- $\mathcal{I} \equiv \prod_i I_i$ is the set of initial states.

Formal definition of synchronous system $\mathbb{P}$

- $\mathbb{P} \equiv (p_1, p_2, \ldots, p_n)$ where each $p_i \equiv (S_i, I_i, M\mathcal{G}_i, \vdash_i)$ is a synchronous process, is a synchronous system.
- Transition system $Tr \equiv (S, \mathcal{I}, \rightarrow)$.
- $S \equiv \prod_i S_i$ is the state space of the transition system.
- $\mathcal{I} \equiv \prod_i I_i$ is the set of initial states.
- $\rightarrow \subseteq S \times S$ is the transition relation.

$$
\begin{aligned}
&(s_1, s_2, \ldots s_n) \rightarrow (s_1', s_2', \ldots, s_n') \\
&\quad \text{iff} \quad \forall p_i \in \mathbb{P}; (s_i, \{\text{messages to } p_i \text{ from } p_{-i}\}) \vdash s_i'
\end{aligned}
$$