CONGEST Model: SSSP, Electing a leader, APSP

Christian Wulff-Nilsen
Algorithmic Techniques for Modern Data Models
DTU

October 3, 2025

Overview for today

- CONGEST vs LOCAL Model
- Single-Source Shortest Path Distances (SSSP)
- Breadth-First Search Tree (BFS Tree)
- Electing a Leader
- All-Pairs Shortest Path (APSP)

CONGEST vs LOCAL Model

- LOCAL model: port-numbered network with unique node identifiers in $\{1, \ldots, n^d\}$, constant d, $n = |V|$
- Messages along edges can be arbitrarily large
- CONGEST model: same as LOCAL model but with small messages:
  - Each message is in the set $\{0, \ldots, n^c\}$, c constant
  - Thus, in each round, each message can only contain $O(\log n^c) = O(c \log n) = O(\log n)$ bits
- Communication round order:
  - Send messages to neighbors
  - Receive messages from neighbors
  - Local computation at any point in the round
- We want to solve instances of a graph problem with a distributed algorithm using few communication rounds
- The communication network *is* the graph instance
- Each node is a machine that should output its part of the solution

Single-Source Shortest Path Distance Problem (SSSP)

- $G = (V, E)$: unweighted undirected n-node connected graph
- One node s starts with input 1, all others with input 0
- SSSP: local output for each node v is its distance from/to root s:

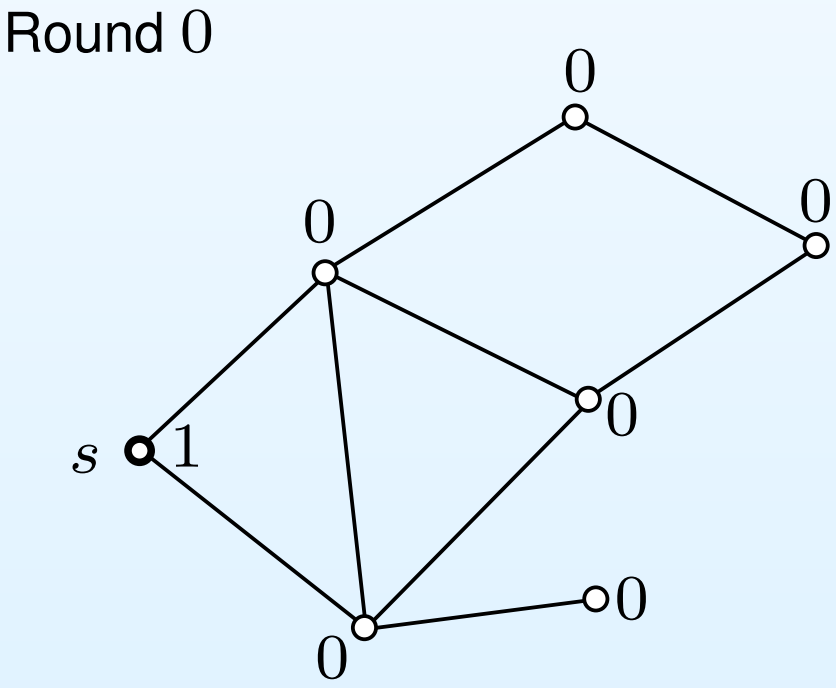
$$\mathrm{dist}_G(s, v) = \mathrm{dist}_G(v, s)$$

- We show that SSSP can be solved in $O(\mathrm{diam}(G))$ communication rounds where $\mathrm{diam}(G)$ is the diameter of G:

$$\mathrm{diam}(G) = \max_{u,v \in V} \mathrm{dist}_G(u, v)$$

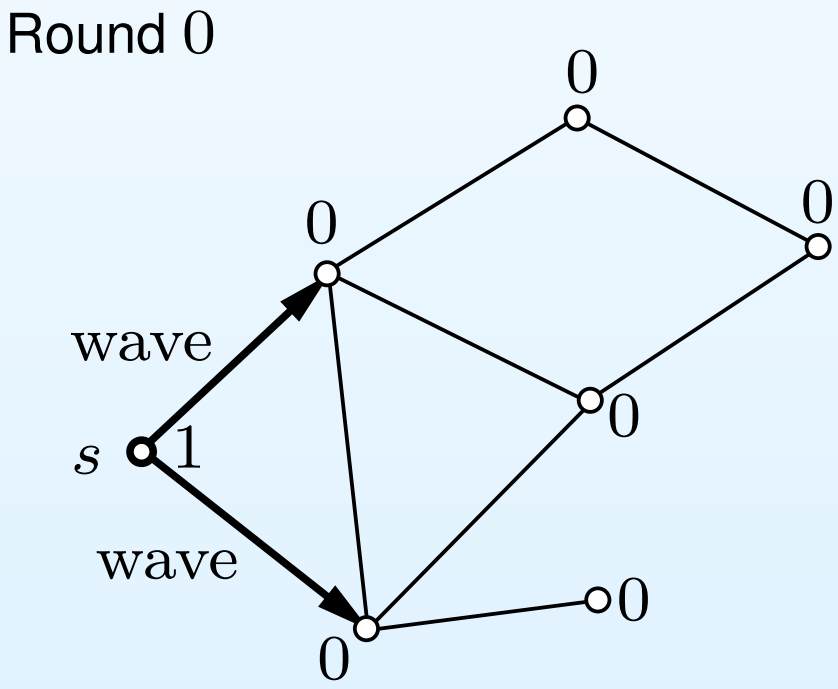
Solving SSSP:the `Wave` algorithm

- Round 0: root s sends message wave to its neighbors, switches to state 0 and stops
- Round $i > 0$: for each node v that has not stopped and has received at least one message wave in round $i - 1$:
  - v sends wave to its neighbors, switches to state i and stops



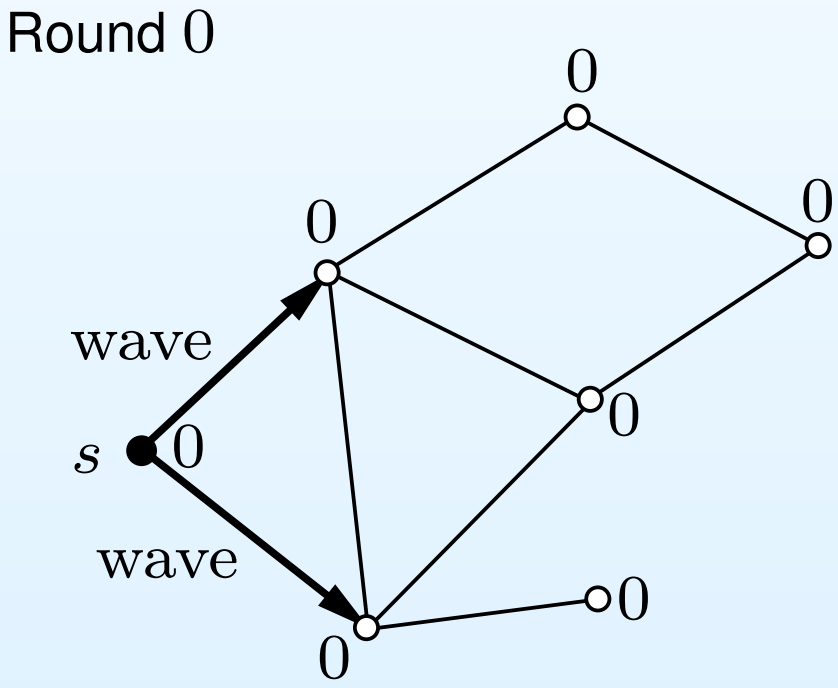
Solving SSSP:the `Wave` algorithm

- Round 0: root s sends message wave to its neighbors, switches to state 0 and stops
- Round $i > 0$: for each node v that has not stopped and has received at least one message wave in round $i - 1$:
  - v sends wave to its neighbors, switches to state i and stops



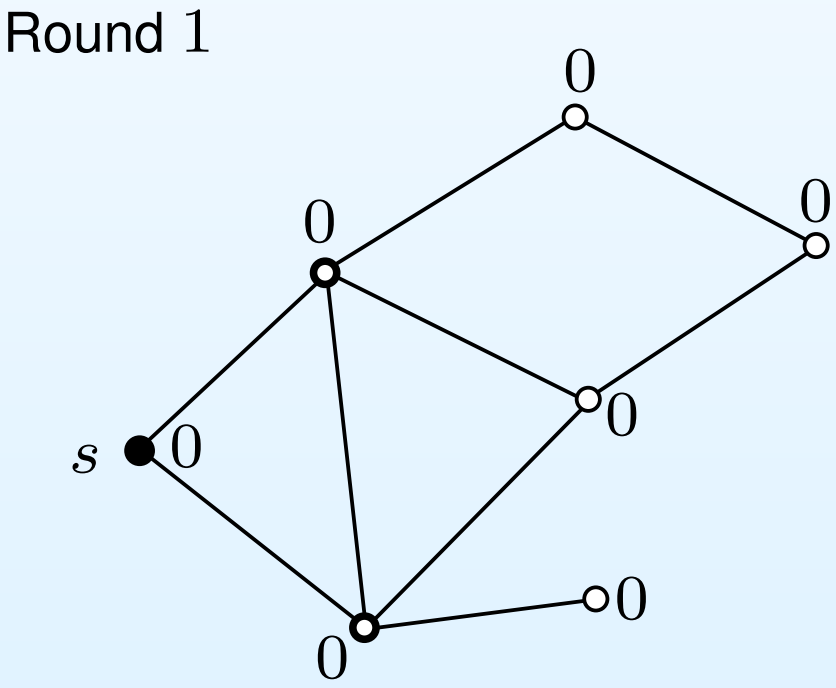
Solving SSSP:the `Wave` algorithm

- Round 0: root s sends message wave to its neighbors, switches to state 0 and stops
- Round $i > 0$: for each node v that has not stopped and has received at least one message wave in round $i - 1$:
  - v sends wave to its neighbors, switches to state i and stops



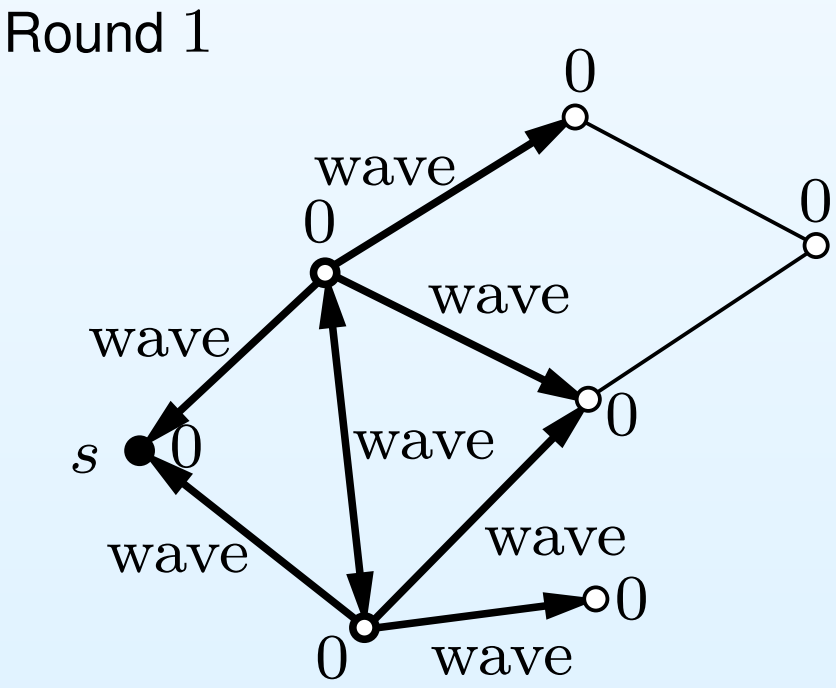
Solving SSSP:the `Wave` algorithm

- Round 0: root s sends message wave to its neighbors, switches to state 0 and stops
- Round $i > 0$: for each node v that has not stopped and has received at least one message wave in round $i - 1$:
  - v sends wave to its neighbors, switches to state i and stops



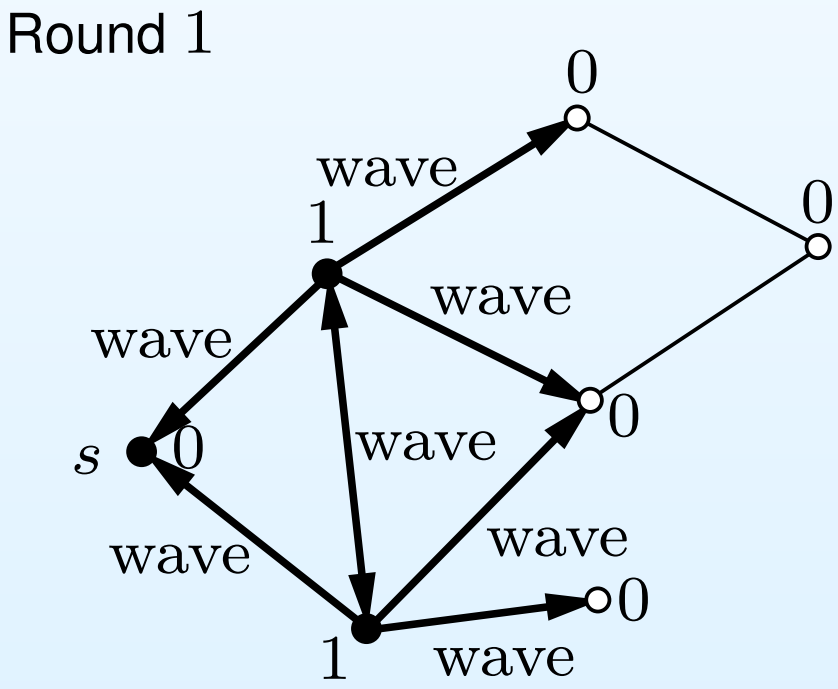
Solving SSSP:the `Wave` algorithm

- Round 0: root s sends message wave to its neighbors, switches to state 0 and stops
- Round $i > 0$: for each node v that has not stopped and has received at least one message wave in round $i - 1$:
  - v sends wave to its neighbors, switches to state i and stops



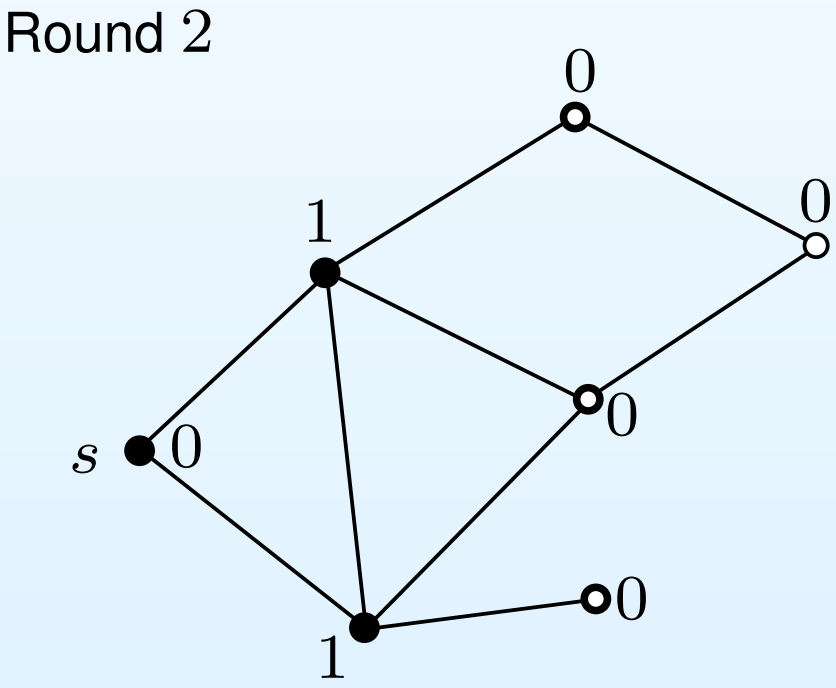
Solving SSSP:the `Wave` algorithm

- Round 0: root s sends message wave to its neighbors, switches to state 0 and stops
- Round $i > 0$: for each node v that has not stopped and has received at least one message wave in round $i - 1$:
  - v sends wave to its neighbors, switches to state i and stops



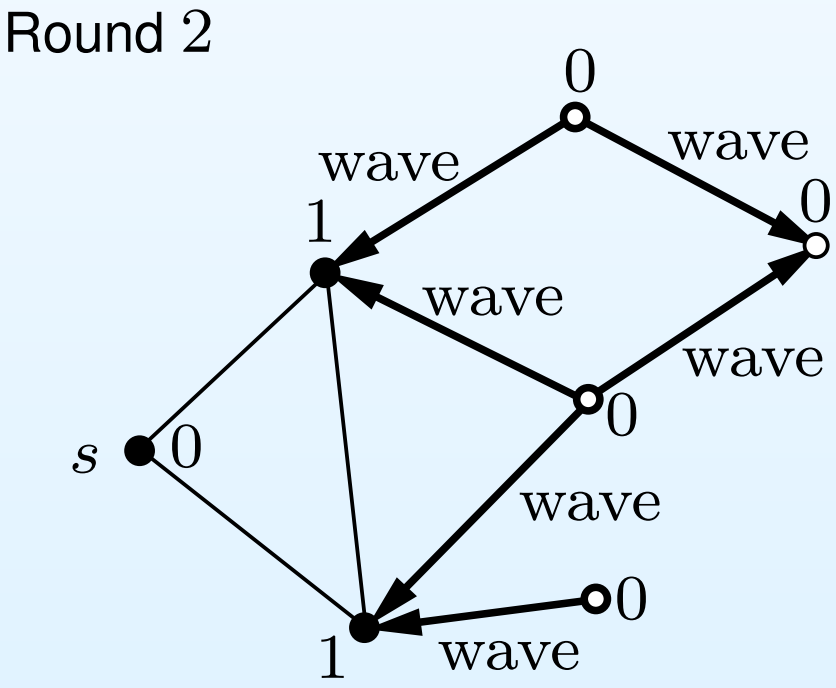
Solving SSSP:the `Wave` algorithm

- Round 0: root s sends message wave to its neighbors, switches to state 0 and stops
- Round $i > 0$: for each node v that has not stopped and has received at least one message wave in round $i - 1$:
  - v sends wave to its neighbors, switches to state i and stops



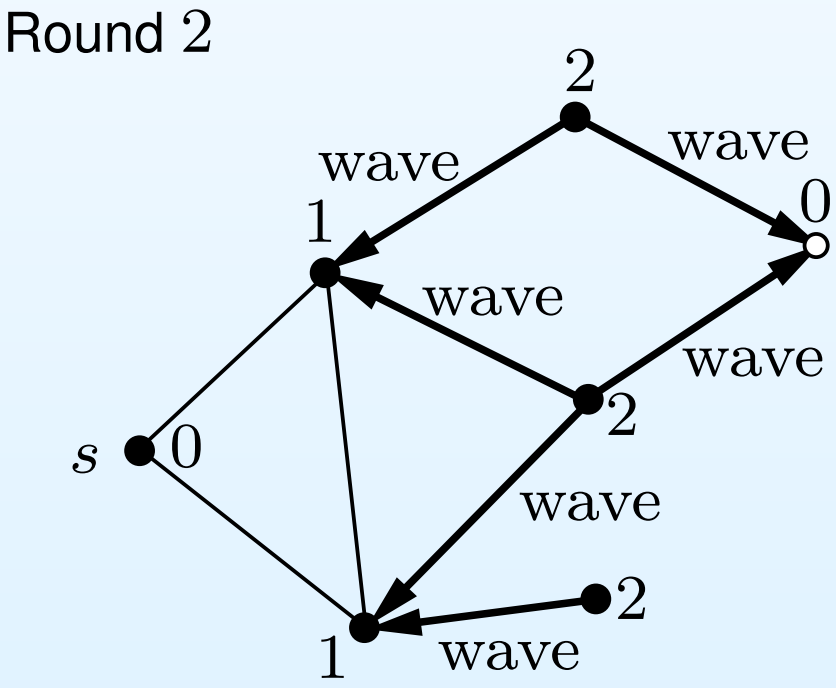
Solving SSSP:the `Wave` algorithm

- Round 0: root s sends message wave to its neighbors, switches to state 0 and stops
- Round $i > 0$: for each node v that has not stopped and has received at least one message wave in round $i - 1$:
  - v sends wave to its neighbors, switches to state i and stops



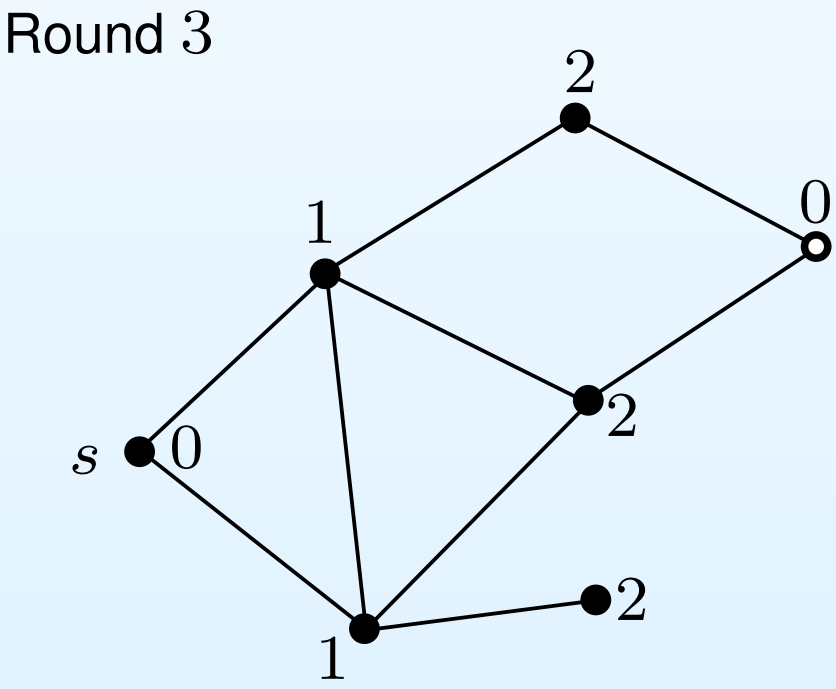
Solving SSSP:the `Wave` algorithm

- Round 0: root s sends message wave to its neighbors, switches to state 0 and stops
- Round $i > 0$: for each node v that has not stopped and has received at least one message wave in round $i - 1$:
  - v sends wave to its neighbors, switches to state i and stops



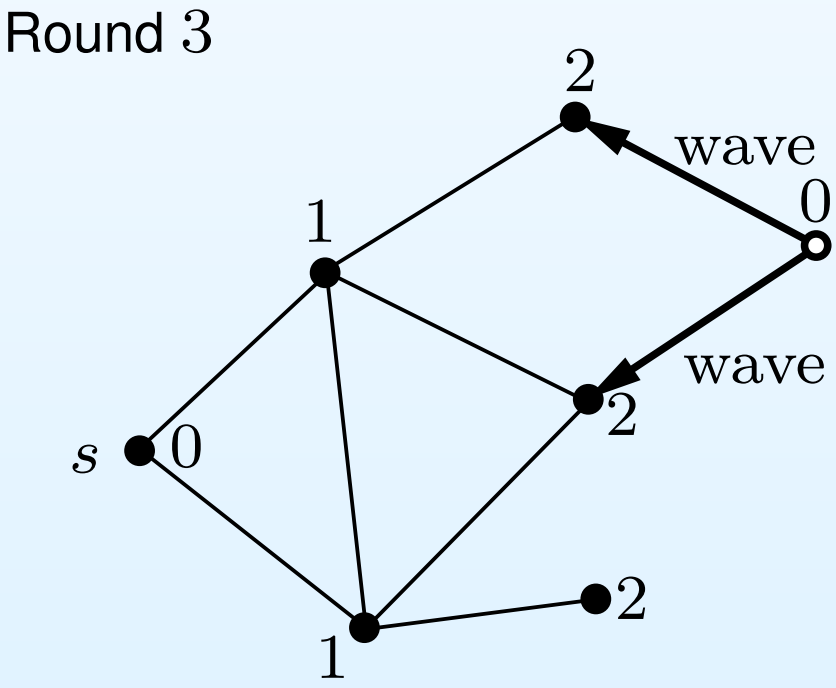
Solving SSSP:the `Wave` algorithm

- Round 0: root s sends message wave to its neighbors, switches to state 0 and stops
- Round $i > 0$: for each node v that has not stopped and has received at least one message wave in round $i - 1$:
  - v sends wave to its neighbors, switches to state i and stops



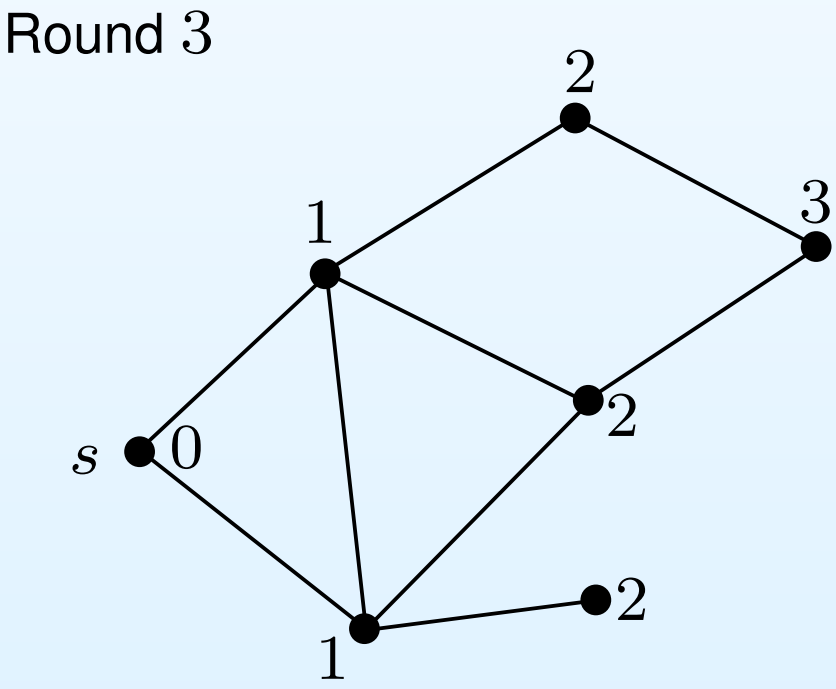
Solving SSSP:the `Wave` algorithm

- Round 0: root s sends message wave to its neighbors, switches to state 0 and stops
- Round $i > 0$: for each node v that has not stopped and has received at least one message wave in round $i - 1$:
  - v sends wave to its neighbors, switches to state i and stops



Solving SSSP:the `Wave` algorithm

- Round 0: root s sends message wave to its neighbors, switches to state 0 and stops
- Round $i > 0$: for each node v that has not stopped and has received at least one message wave in round $i - 1$:
  - v sends wave to its neighbors, switches to state i and stops



Correctness of the `Wave` algorithm

- For each $i \geq 0$, every node v with $\mathrm{dist}_G(s, v) = i$, outputs i and stops in round i
- Proof: by induction on $i \geq 0$

Number of communication rounds

- Each node v has distance $\mathrm{dist}_G(s, v) \leq \mathrm{diam}(G)$ from s
- Thus it stops in round $\mathrm{diam}(G)$ at the latest
- Number of communication rounds: $O(\mathrm{diam}(G))$

Breadth-First Search (BFS) Tree

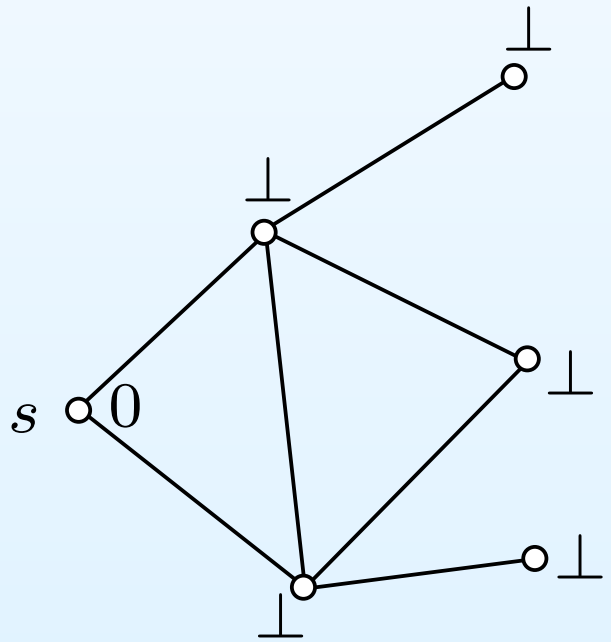
- SSSP: compute distances from s to each node $v \in V$
- BFS Tree Problem: compute shortest path tree T from s, labeling each $v \in V$ with:
  - $d(v) = \text{dist}_G(s, v) = \text{dist}_G(v, s)$
  - Parent $p(v)$ (provided $v \neq s$)
  - Children $C(v)$

BFS: Information maintained in each node

- During the BFS algorithm, each node $v \in V$ maintains the following for the partially constructed BFS tree T
  - $d(v)$: distance to root s
  - $p(v)$: parent node in T
  - $C(v)$: set of children of v in T
  - $a(v)$: acknowledgment, equals 1 when the subtree rooted at v is constructed; 0 otherwise

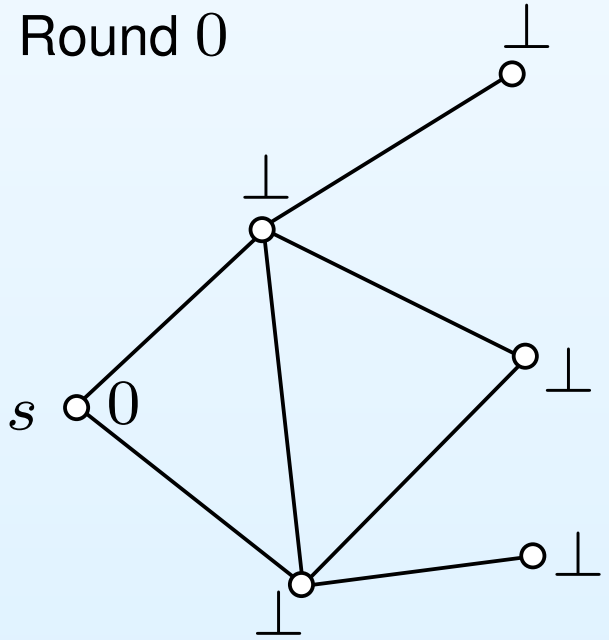
BFS: Initialization

- Initial T is empty so initialize for each v:
  - $d(v) \leftarrow \perp$ for $v \neq s$ and $d(v) \leftarrow 0$ for $v = s$
  - $p(v) \leftarrow \perp$
  - $C(v) \leftarrow \perp$
  - $a(v) \leftarrow 0$
- Example of an initial tree with d-values:



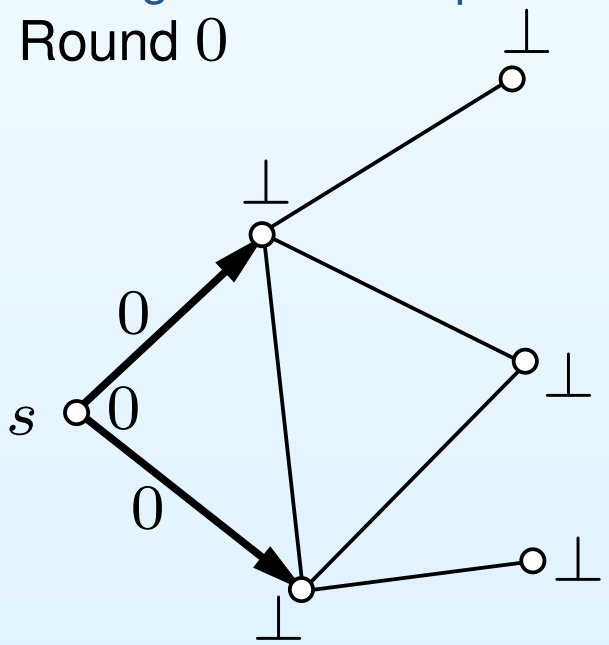
BFS: One communication round

- For each node u with $d(u) \neq \perp$:
  - u sends $d(u)$ to its neighbors
- Each node v with $d(v) = \perp$ receiving a value j from a node u:
  - Set $p(v) \leftarrow u$
  - Set $d(v) \leftarrow j + 1$
  - Send "accept" back to u (in the next round)
  - If v receives messages from multiple nodes, ignore all except one



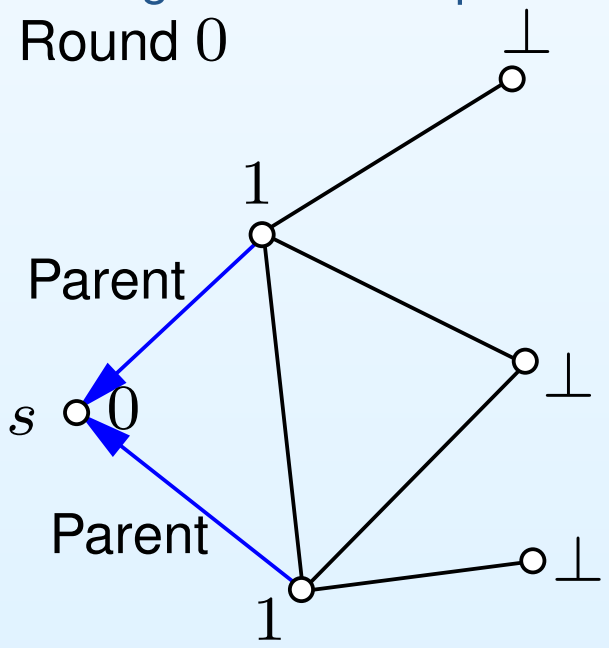
BFS: One communication round

- For each node u with $d(u) \neq \perp$:
  - u sends $d(u)$ to its neighbors
- Each node v with $d(v) = \perp$ receiving a value j from a node u:
  - Set $p(v) \leftarrow u$
  - Set $d(v) \leftarrow j + 1$
  - Send "accept" back to u (in the next round)
  - If v receives messages from multiple nodes, ignore all except one



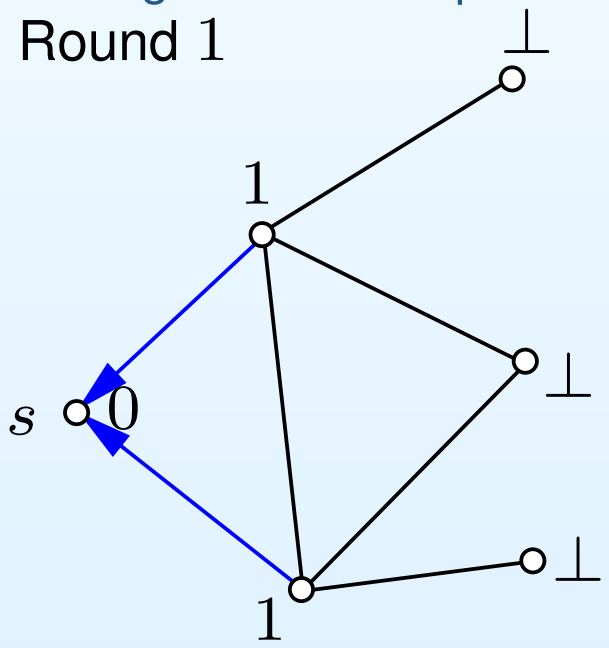
BFS: One communication round

- For each node u with $d(u) \neq \perp$:
  - u sends $d(u)$ to its neighbors
- Each node v with $d(v) = \perp$ receiving a value j from a node u:
  - Set $p(v) \leftarrow u$
  - Set $d(v) \leftarrow j + 1$
  - Send "accept" back to u (in the next round)
  - If v receives messages from multiple nodes, ignore all except one



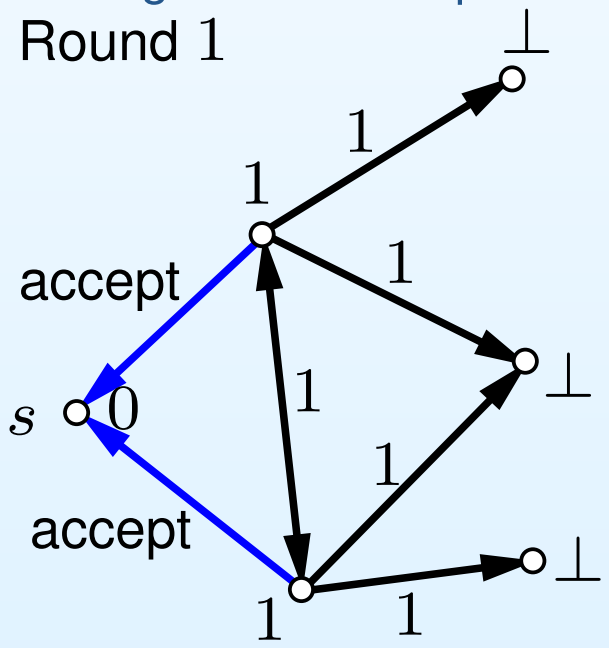
BFS: One communication round

- For each node u with $d(u) \neq \perp$:
  - u sends $d(u)$ to its neighbors
- Each node v with $d(v) = \perp$ receiving a value j from a node u:
  - Set $p(v) \leftarrow u$
  - Set $d(v) \leftarrow j + 1$
  - Send "accept" back to u (in the next round)
  - If v receives messages from multiple nodes, ignore all except one



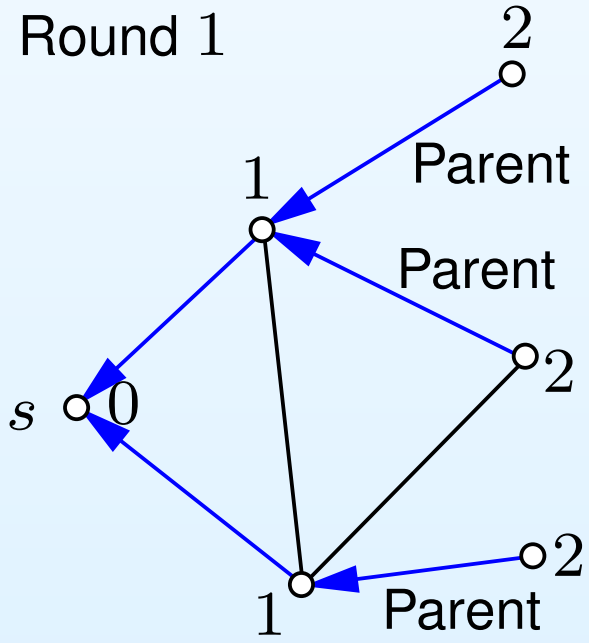
BFS: One communication round

- For each node u with $d(u) \neq \perp$:
  - u sends $d(u)$ to its neighbors
- Each node v with $d(v) = \perp$ receiving a value j from a node u:
  - Set $p(v) \leftarrow u$
  - Set $d(v) \leftarrow j + 1$
  - Send "accept" back to u (in the next round)
  - If v receives messages from multiple nodes, ignore all except one



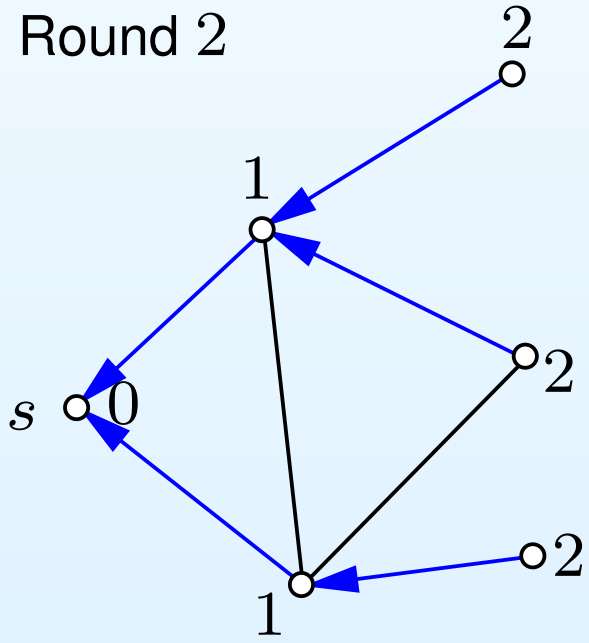
BFS: One communication round

- For each node u with $d(u) \neq \perp$:
  - u sends $d(u)$ to its neighbors
- Each node v with $d(v) = \perp$ receiving a value j from a node u:
  - Set $p(v) \leftarrow u$
  - Set $d(v) \leftarrow j + 1$
  - Send "accept" back to u (in the next round)
  - If v receives messages from multiple nodes, ignore all except one



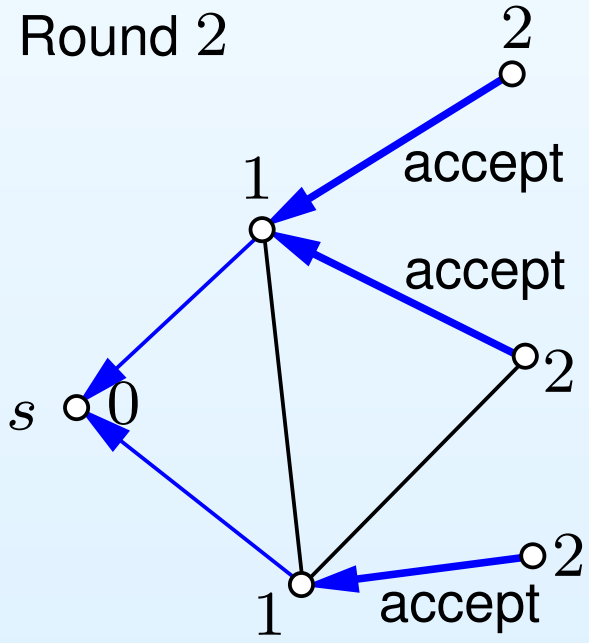
BFS: One communication round

- For each node u with $d(u) \neq \perp$:
  - u sends $d(u)$ to its neighbors
- Each node v with $d(v) = \perp$ receiving a value j from a node u:
  - Set $p(v) \leftarrow u$
  - Set $d(v) \leftarrow j + 1$
  - Send "accept" back to u (in the next round)
  - If v receives messages from multiple nodes, ignore all except one



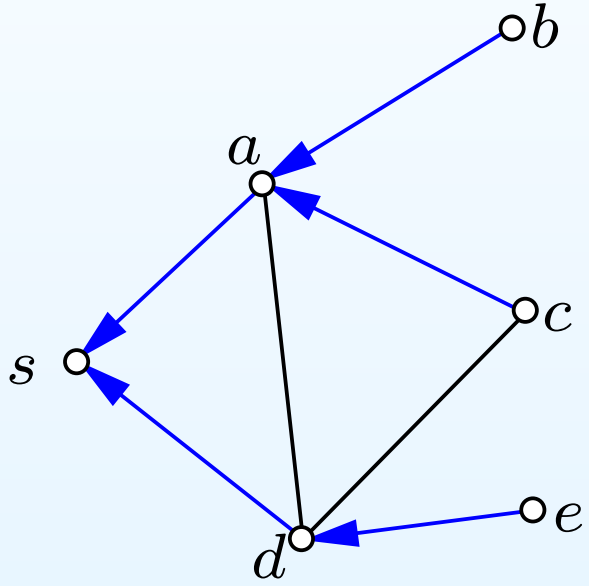
BFS: One communication round

- For each node u with $d(u) \neq \perp$:
  - u sends $d(u)$ to its neighbors
- Each node v with $d(v) = \perp$ receiving a value j from a node u:
  - Set $p(v) \leftarrow u$
  - Set $d(v) \leftarrow j + 1$
  - Send "accept" back to u (in the next round)
  - If v receives messages from multiple nodes, ignore all except one



BFS: One communication round

- Each node u sets $C(u)$ to the set of nodes it received "accept" from
  - $C(u)$ is exactly the set of nodes v with $p(v) = u$
  - $C(u) = \emptyset$ if u is a leaf

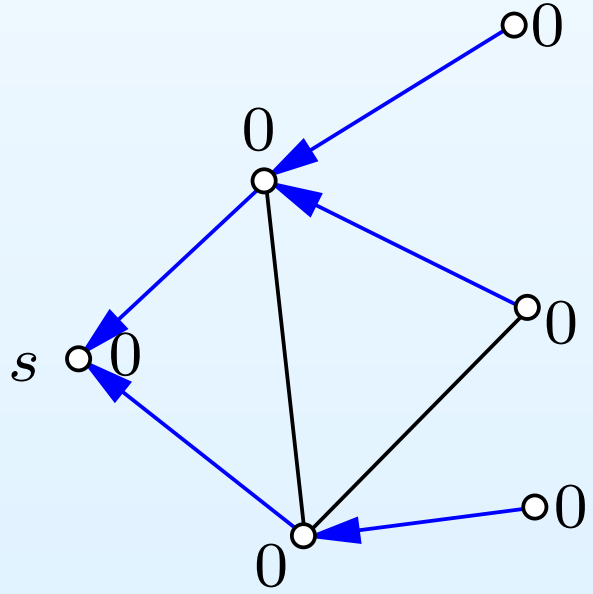


- In this example,

$$C(s) = \{a, d\}, \quad C(a) = \{b, c\}, \quad C(b) = \emptyset,$$
$$C(c) = \emptyset, \quad C(d) = \{e\}, \quad C(e) = \emptyset$$

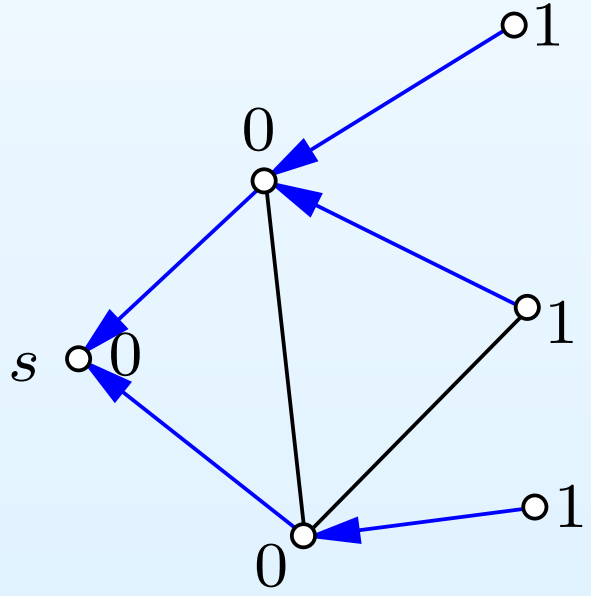
BFS: One communication round, acknowledgments

- "ack" message: sent from a node v to its parent once the subtree rooted at v is constructed
- For each node u with $C(u) \neq \perp$ and $a(u) = 0$:
  - Once u has received message "ack" from each $v \in C(u)$, it sets $a(u) \leftarrow 1$ since its subtree is constructed
  - If $p(v) \neq \perp$, v then sends "ack" to $p(v)$ (happens when $v \neq s$)
  - If u is a leaf, $C(u) = \emptyset$ so u does not wait for any "ack" messages
- Example with $a(u)$ indicated for each u:



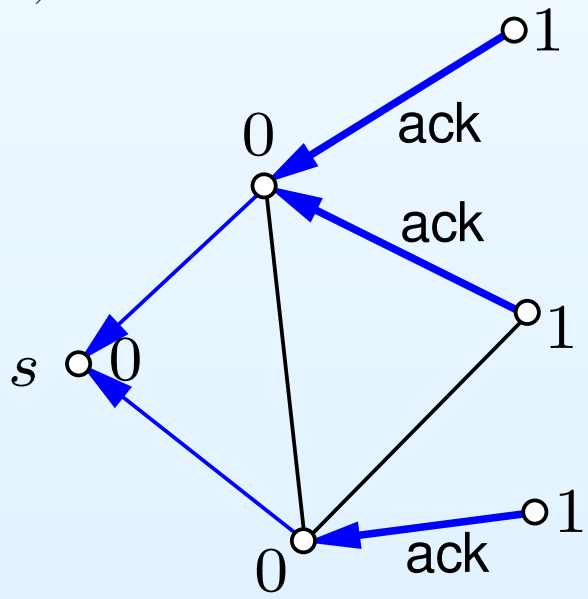
BFS: One communication round, acknowledgments

- "ack" message: sent from a node v to its parent once the subtree rooted at v is constructed
- For each node u with $C(u) \neq \perp$ and $a(u) = 0$:
  - Once u has received message "ack" from each $v \in C(u)$, it sets $a(u) \leftarrow 1$ since its subtree is constructed
  - If $p(v) \neq \perp$, v then sends "ack" to $p(v)$ (happens when $v \neq s$)
  - If u is a leaf, $C(u) = \emptyset$ so u does not wait for any "ack" messages
- Example with $a(u)$ indicated for each u:



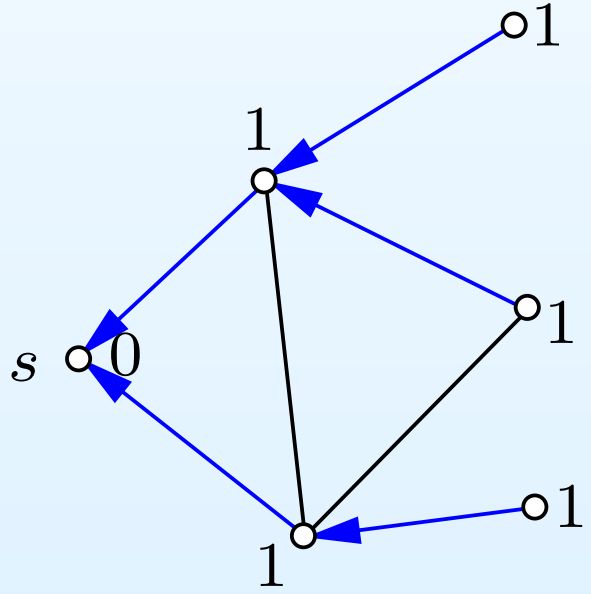
BFS: One communication round, acknowledgments

- "ack" message: sent from a node v to its parent once the subtree rooted at v is constructed
- For each node u with $C(u) \neq \perp$ and $a(u) = 0$:
  - Once u has received message "ack" from each $v \in C(u)$, it sets $a(u) \leftarrow 1$ since its subtree is constructed
  - If $p(v) \neq \perp$, v then sends "ack" to $p(v)$ (happens when $v \neq s$)
  - If u is a leaf, $C(u) = \emptyset$ so u does not wait for any "ack" messages
- Example with $a(u)$ indicated for each u:



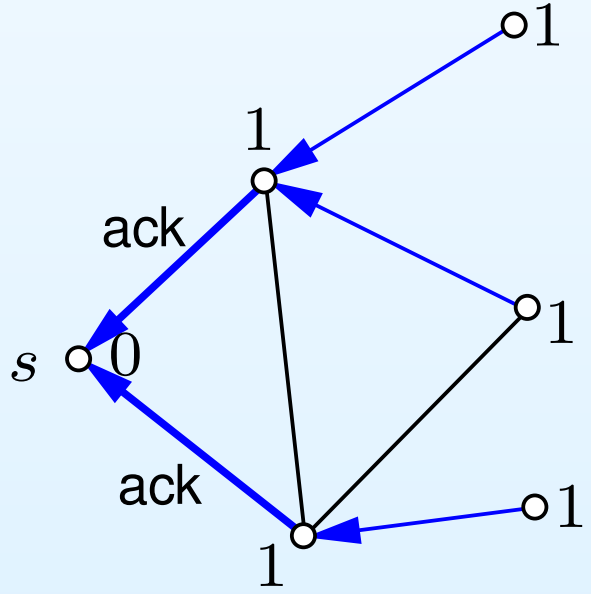
BFS: One communication round, acknowledgments

- "ack" message: sent from a node v to its parent once the subtree rooted at v is constructed
- For each node u with $C(u) \neq \perp$ and $a(u) = 0$:
  - Once u has received message "ack" from each $v \in C(u)$, it sets $a(u) \leftarrow 1$ since its subtree is constructed
  - If $p(v) \neq \perp$, v then sends "ack" to $p(v)$ (happens when $v \neq s$)
  - If u is a leaf, $C(u) = \emptyset$ so u does not wait for any "ack" messages
- Example with $a(u)$ indicated for each u:



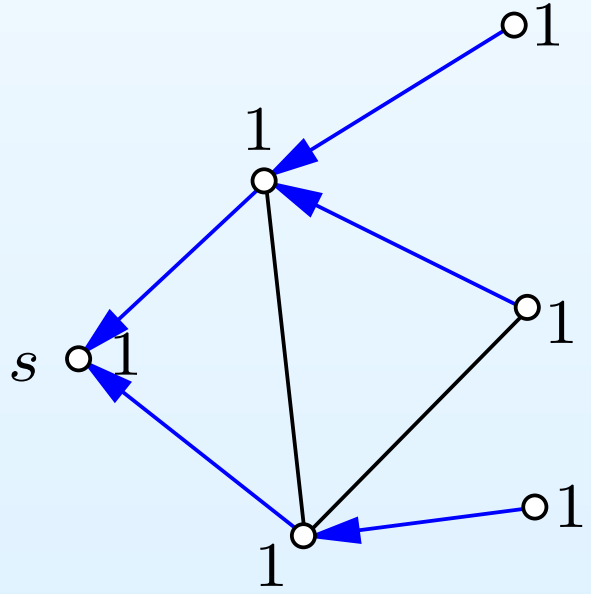
BFS: One communication round, acknowledgments

- "ack" message: sent from a node v to its parent once the subtree rooted at v is constructed
- For each node u with $C(u) \neq \perp$ and $a(u) = 0$:
  - Once u has received message "ack" from each $v \in C(u)$, it sets $a(u) \leftarrow 1$ since its subtree is constructed
  - If $p(v) \neq \perp$, v then sends "ack" to $p(v)$ (happens when $v \neq s$)
  - If u is a leaf, $C(u) = \emptyset$ so u does not wait for any "ack" messages
- Example with $a(u)$ indicated for each u:



BFS: One communication round, acknowledgments

- "ack" message: sent from a node v to its parent once the subtree rooted at v is constructed
- For each node u with $C(u) \neq \perp$ and $a(u) = 0$:
  - Once u has received message "ack" from each $v \in C(u)$, it sets $a(u) \leftarrow 1$ since its subtree is constructed
  - If $p(v) \neq \perp$, v then sends "ack" to $p(v)$ (happens when $v \neq s$)
  - If u is a leaf, $C(u) = \emptyset$ so u does not wait for any "ack" messages
- Example with $a(u)$ indicated for each u:



BFS: number of communication rounds

- It takes $O(\mathrm{diam}(G))$ rounds to compute $d(v)$ and $p(v)$ for each v:
  - Each round adds another BFS layer to T
- It takes an additional $O(\mathrm{diam}(G))$ rounds to set $a(v) \leftarrow 1$ and send "ack" messages back towards s
- Total number of rounds: $O(\mathrm{diam}(G))$

Electing a Leader, Problem Definition

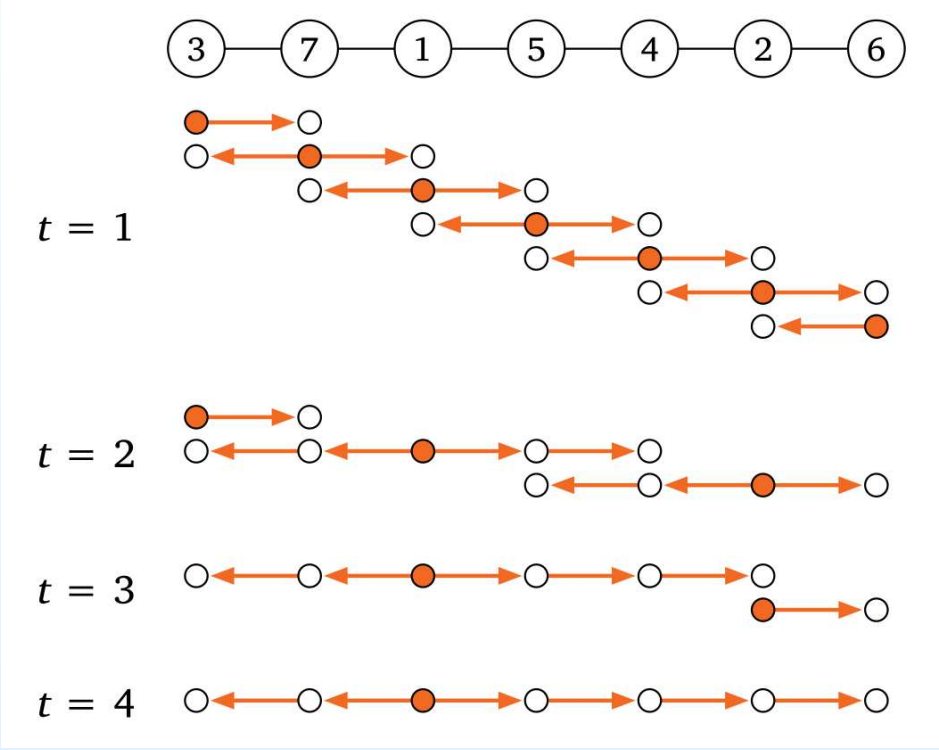
- Exactly one node s should output 1, all others should output 0
- Node s is called the *leader*
- We will solve this problem in $O(\mathrm{diam}(G))$ communication rounds
- Electing a leader is useful when we want one node to take charge in solving a problem
- Will be needed as part of our APSP algorithm

Electing a Leader

- We will give an algorithm for electing the leader with smallest node id
- Overall algorithm description:
  - Run BFS from each possible root node in parallel
  - Extend BFS algorithm's messages with the root node's id
  - Each node v keeps track of the smallest root node id among the BFS'es that have visited v
  - When all BFS'es have completed, all nodes will have stored the smallest node id in the graph and this is the elected leader
- Problem with this algorithm:
  - Up to $\Theta(n)$ messages sent along a single edge in one communication round ($n = |V|$)
  - The CONGEST model only allows $O(\log n)$ bit messages!

Electing a Leader: Small Messages

- Modification of algorithm to ensure small messages:
  - Suppose $\ell(v)$ is the current smallest root id for a node v
  - In each round, v only sends messages for the BFS of root id $\ell(v)$
  - Messages to v from all other BFS'es are simply ignored since they are not needed to elect the leader
  - Message size: $O(\log n)$ – allowed by the CONGEST model



Electing a Leader: Algorithm Analysis

- Let s be the node with smallest id
- The BFS from s eventually reaches each node v
- Thus, the final update to $\ell(v)$ is $\ell(v) \leftarrow s$
- The "ack" messages for the BFS of s will eventually reach s, indicating that s has grown a full BFS tree
- For each node $s' \neq s$, the BFS tree of s' will never be completed:
  - s (and possibly other nodes) ignore messages for the BFS of s'
  - Thus, s' will not receive "ack" messages for its own BFS
- At termination:
  - Once s knows it is the leader, it informs the other nodes of this with another BFS
  - s outputs 1, all other nodes output 0
- Number of communication rounds: $O(\mathrm{diam}(G))$

All-Pairs Shortest Path Problem (APSP)

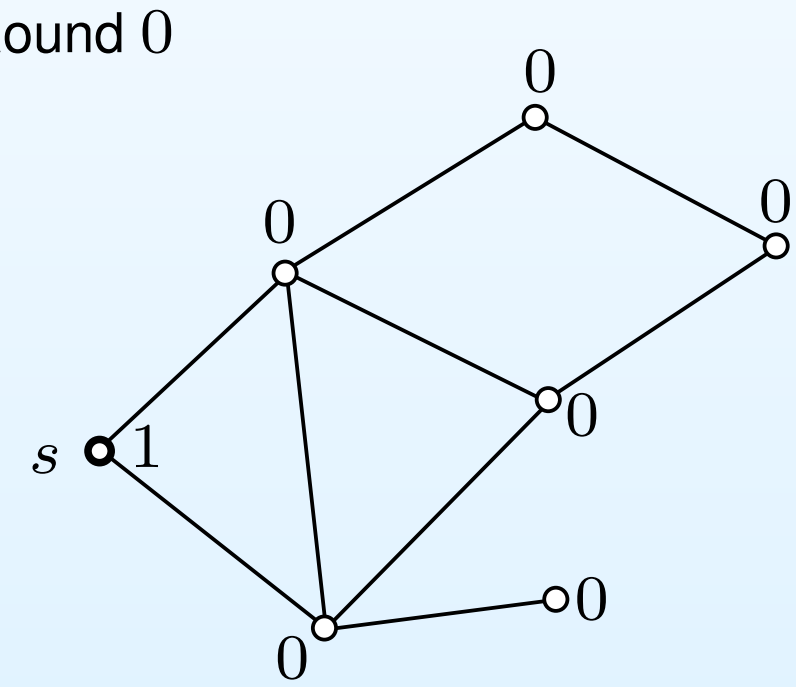
- $G = (V, E)$: unweighted undirected n-node connected graph
- APSP: compute $\mathrm{dist}_G(u, v)$ for all $u, v \in V$
- Local output of each $v \in V$:

$$\{(u, d) : u \in V, d = \mathrm{dist}_G(u, v)\}$$

- We show that APSP can be solved in $O(n)$ communication rounds
- This is asymptotically optimal as there is an $\Omega(n)$ lower bound (Exercise 5.7 in the book)

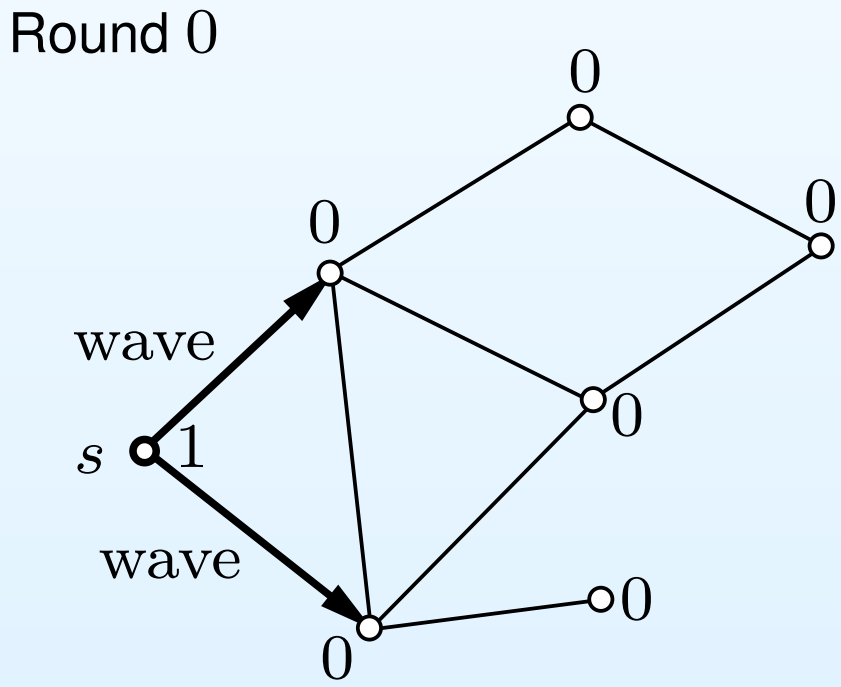
All-Pairs Shortest Paths (APSP)

- Recall the `Wave` algorithm for SSSP:
  - Round 0: root s sends message wave to its neighbors, switches to state 0 and stops
  - Round $i > 0$: for each node v that has not stopped and has received at least one message wave in round $i - 1$:
    - v sends wave to its neighbors, switches to state i and stops



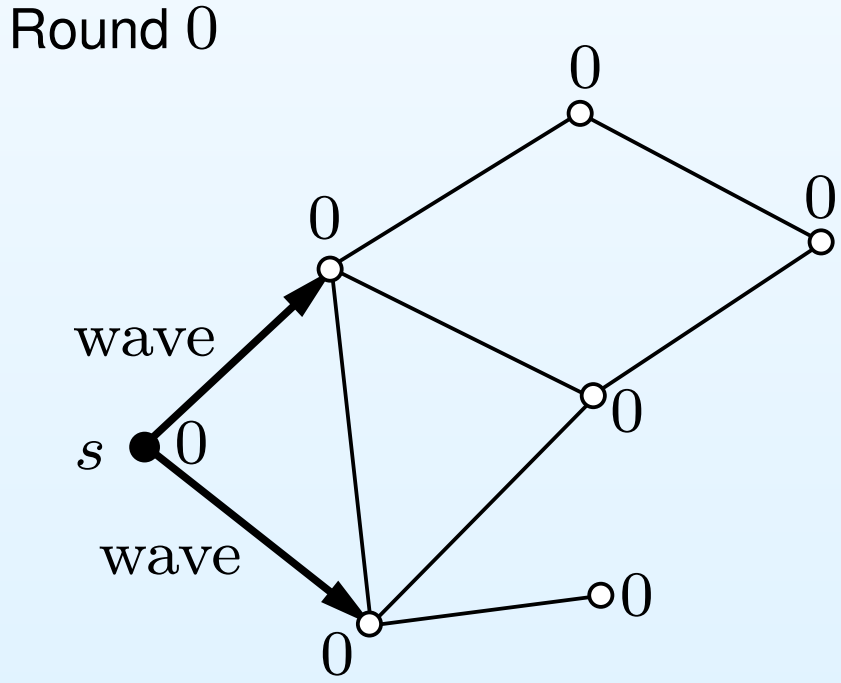
All-Pairs Shortest Paths (APSP)

- Recall the `Wave` algorithm for SSSP:
  - Round 0: root s sends message wave to its neighbors, switches to state 0 and stops
  - Round $i > 0$: for each node v that has not stopped and has received at least one message wave in round $i - 1$:
    - v sends wave to its neighbors, switches to state i and stops



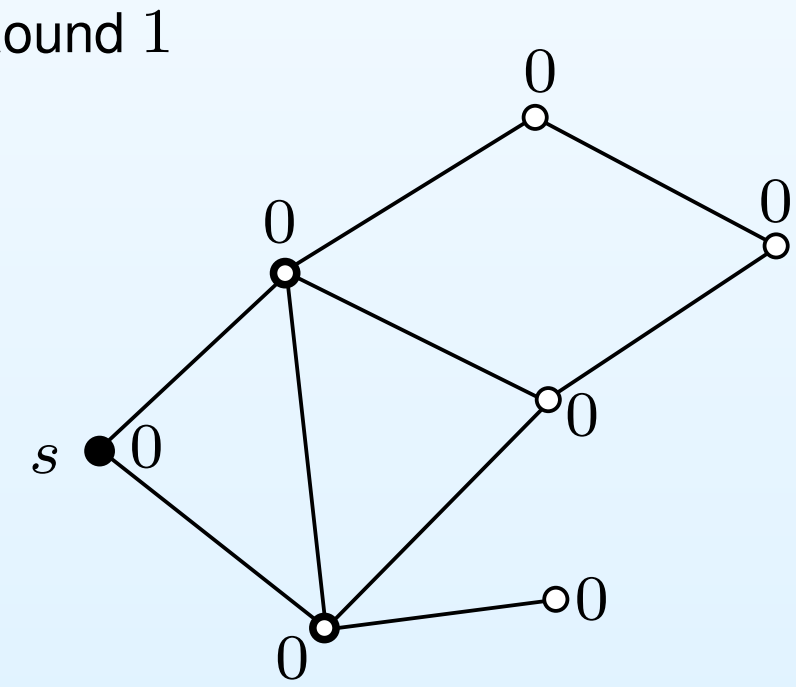
All-Pairs Shortest Paths (APSP)

- Recall the `Wave` algorithm for SSSP:
  - Round 0: root s sends message wave to its neighbors, switches to state 0 and stops
  - Round $i > 0$: for each node v that has not stopped and has received at least one message wave in round $i - 1$:
    - v sends wave to its neighbors, switches to state i and stops



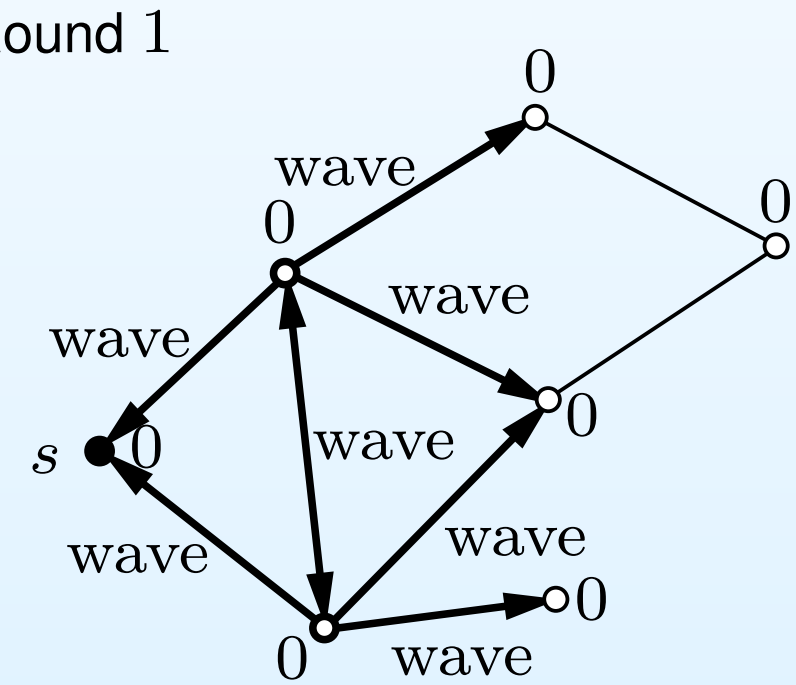
All-Pairs Shortest Paths (APSP)

- Recall the `Wave` algorithm for SSSP:
  - Round 0: root s sends message wave to its neighbors, switches to state 0 and stops
  - Round $i > 0$: for each node v that has not stopped and has received at least one message wave in round $i - 1$:
    - v sends wave to its neighbors, switches to state i and stops



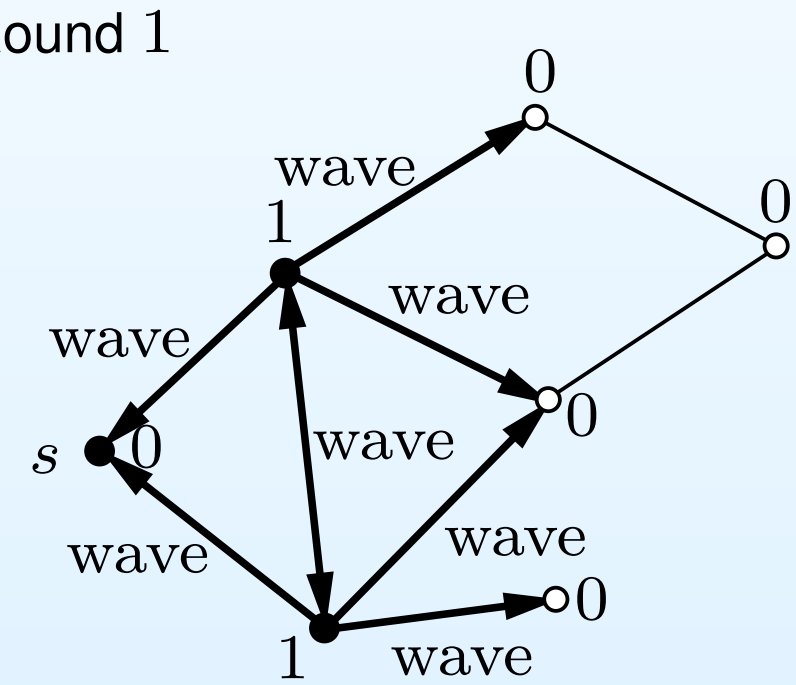
All-Pairs Shortest Paths (APSP)

- Recall the `Wave` algorithm for SSSP:
  - Round 0: root s sends message wave to its neighbors, switches to state 0 and stops
  - Round $i > 0$: for each node v that has not stopped and has received at least one message wave in round $i - 1$:
    - v sends wave to its neighbors, switches to state i and stops



All-Pairs Shortest Paths (APSP)

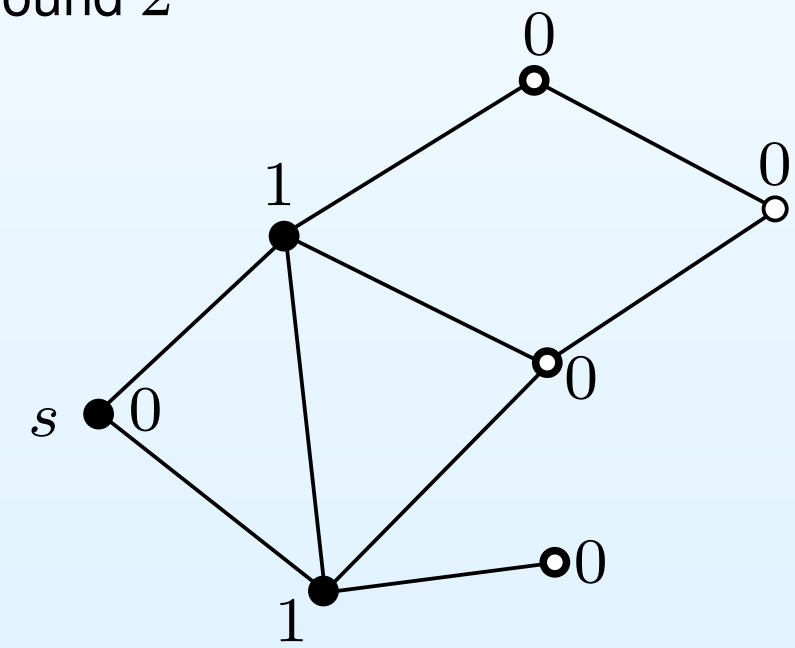
- Recall the `Wave` algorithm for SSSP:
  - Round 0: root s sends message wave to its neighbors, switches to state 0 and stops
  - Round $i > 0$: for each node v that has not stopped and has received at least one message wave in round $i - 1$:
    - v sends wave to its neighbors, switches to state i and stops



All-Pairs Shortest Paths (APSP)

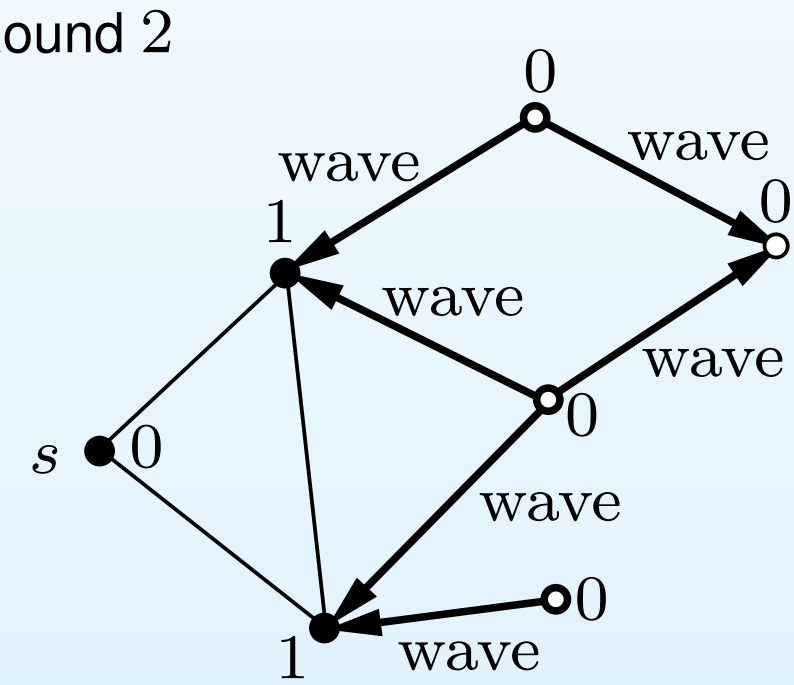
- Recall the `Wave` algorithm for SSSP:
  - Round 0: root s sends message wave to its neighbors, switches to state 0 and stops
  - Round $i > 0$: for each node v that has not stopped and has received at least one message wave in round $i - 1$:
    - v sends wave to its neighbors, switches to state i and stops

Round 2



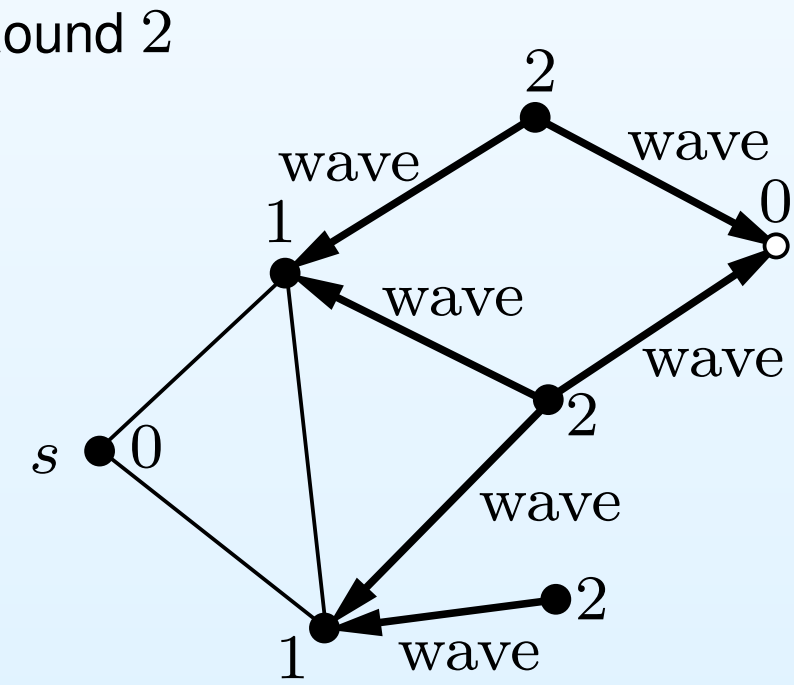
All-Pairs Shortest Paths (APSP)

- Recall the `Wave` algorithm for SSSP:
  - Round 0: root s sends message wave to its neighbors, switches to state 0 and stops
  - Round $i > 0$: for each node v that has not stopped and has received at least one message wave in round $i - 1$:
    - v sends wave to its neighbors, switches to state i and stops



All-Pairs Shortest Paths (APSP)

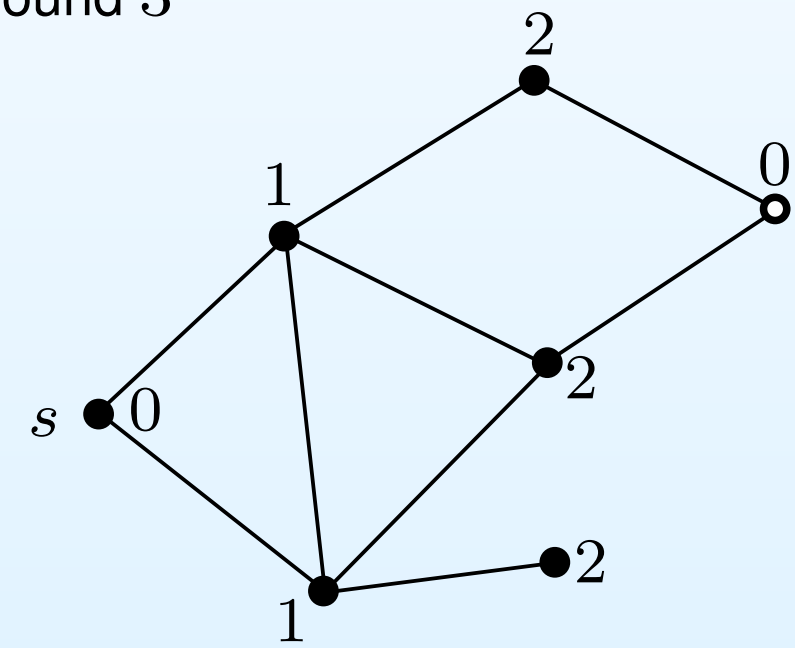
- Recall the `Wave` algorithm for SSSP:
  - Round 0: root s sends message wave to its neighbors, switches to state 0 and stops
  - Round $i > 0$: for each node v that has not stopped and has received at least one message wave in round $i - 1$:
    - v sends wave to its neighbors, switches to state i and stops



All-Pairs Shortest Paths (APSP)

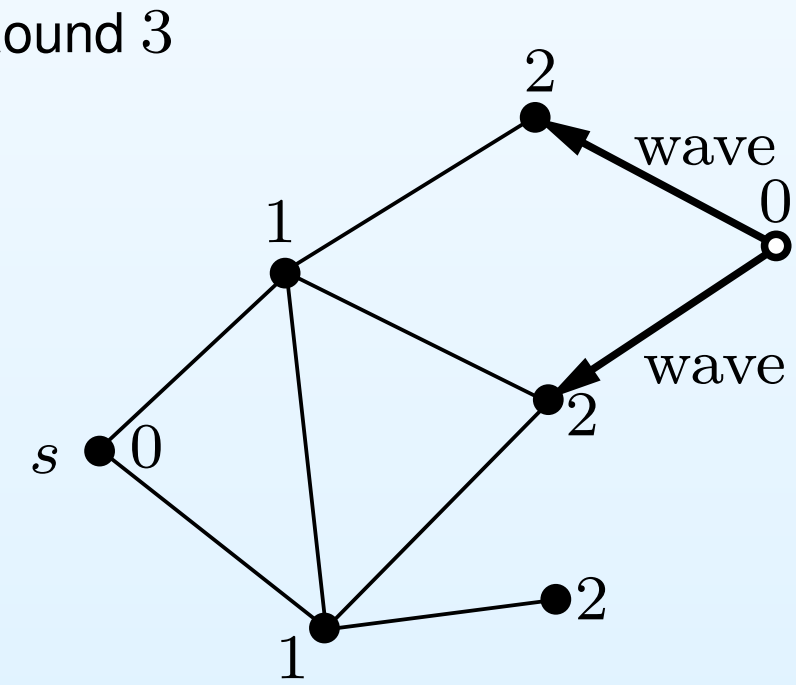
- Recall the `Wave` algorithm for SSSP:
  - Round 0: root s sends message wave to its neighbors, switches to state 0 and stops
  - Round $i > 0$: for each node v that has not stopped and has received at least one message wave in round $i - 1$:
    - v sends wave to its neighbors, switches to state i and stops

Round 3



All-Pairs Shortest Paths (APSP)

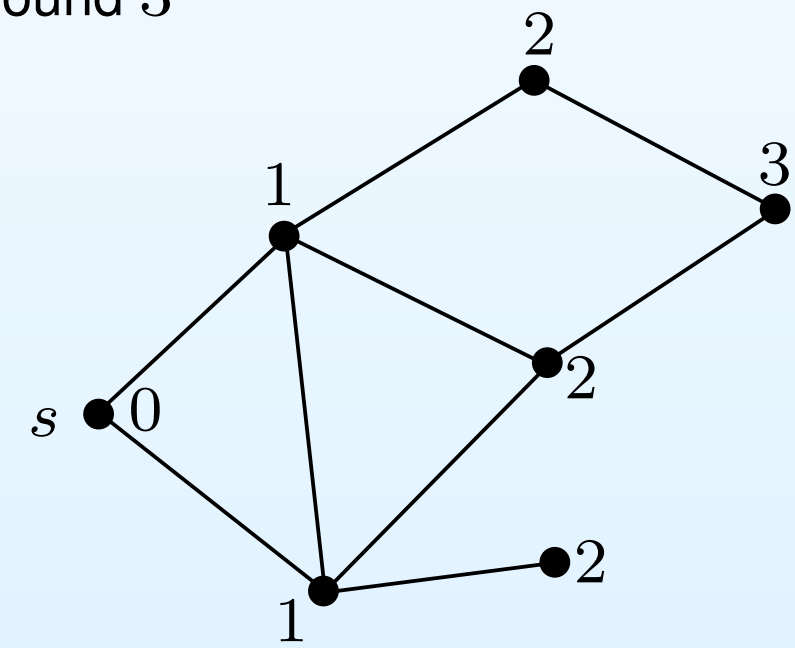
- Recall the `Wave` algorithm for SSSP:
  - Round 0: root s sends message wave to its neighbors, switches to state 0 and stops
  - Round $i > 0$: for each node v that has not stopped and has received at least one message wave in round $i - 1$:
    - v sends wave to its neighbors, switches to state i and stops



All-Pairs Shortest Paths (APSP)

- Recall the `Wave` algorithm for SSSP:
  - Round 0: root s sends message wave to its neighbors, switches to state 0 and stops
  - Round $i > 0$: for each node v that has not stopped and has received at least one message wave in round $i - 1$:
    - v sends wave to its neighbors, switches to state i and stops

Round 3



Non-interfering Waves

- Suppose we start the `Wave` algorithm at node u in communication round t_u and at node v in communication round $t_v > t_u$
- Claim: no node will send more than one message in any round for the two `Wave` algorithms if:

$$t_v - t_u > \mathrm{dist}_G(u, v)$$

- Intuition:
  - The two waves move distance 1 per round
  - v's wave starts too late to ever catch up with u's wave

Non-interfering Waves, Example

- $t_u = 1, t_v = 4, \mathrm{dist}_G(u, v) = 2$
- Inequality is satisfied:

$$t_v - t_u = 3 > 2 = \mathrm{dist}_G(u, v)$$

- The wave of v never catches up with the wave of u:

$t = 1$

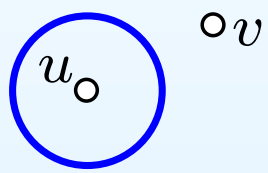
Non-interfering Waves, Example

- $t_u = 1, t_v = 4, \mathrm{dist}_G(u, v) = 2$
- Inequality is satisfied:

$$t_v - t_u = 3 > 2 = \mathrm{dist}_G(u, v)$$

- The wave of v never catches up with the wave of u:

$$t = 2$$



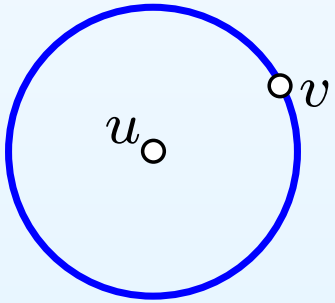
Non-interfering Waves, Example

- $t_u = 1, t_v = 4, \mathrm{dist}_G(u, v) = 2$
- Inequality is satisfied:

$$t_v - t_u = 3 > 2 = \mathrm{dist}_G(u, v)$$

- The wave of v never catches up with the wave of u:

$$t = 3$$



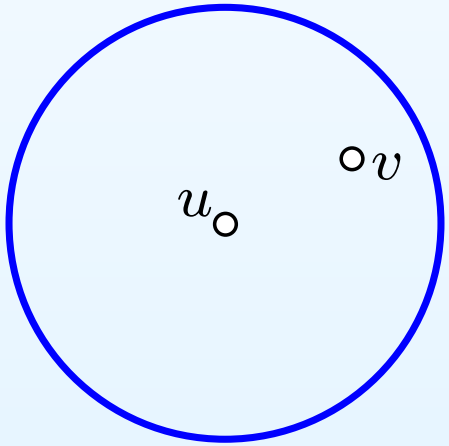
Non-interfering Waves, Example

- $t_u = 1, t_v = 4, \mathrm{dist}_G(u, v) = 2$
- Inequality is satisfied:

$$t_v - t_u = 3 > 2 = \mathrm{dist}_G(u, v)$$

- The wave of v never catches up with the wave of u:

$t = 4$

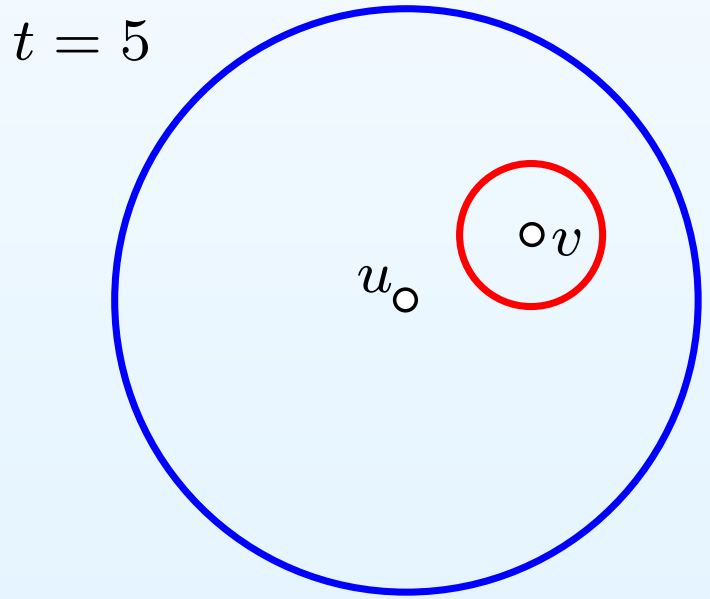


Non-interfering Waves, Example

- $t_u = 1, t_v = 4, \mathrm{dist}_G(u, v) = 2$
- Inequality is satisfied:

$$t_v - t_u = 3 > 2 = \mathrm{dist}_G(u, v)$$

- The wave of v never catches up with the wave of u:

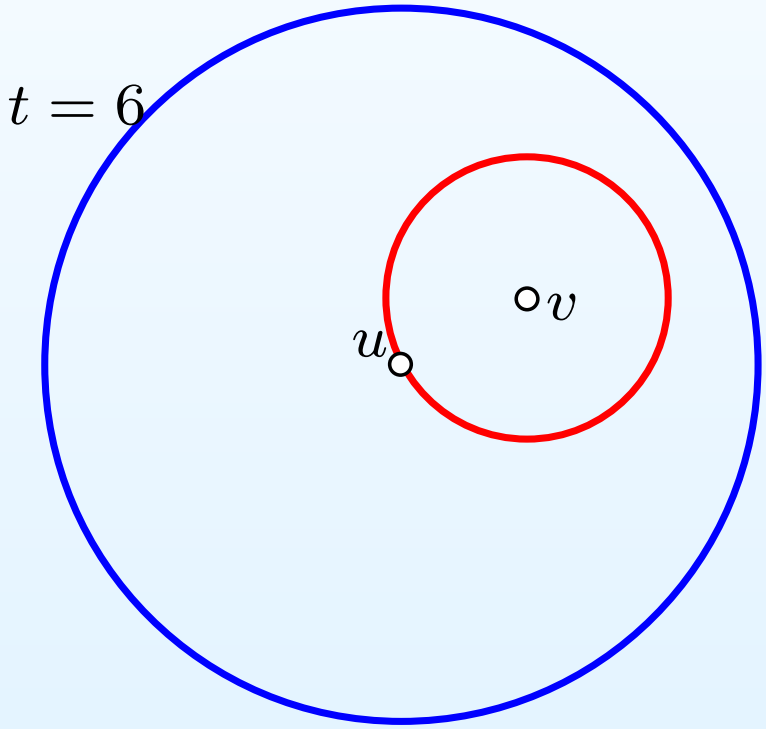


Non-interfering Waves, Example

- $t_u = 1, t_v = 4, \mathrm{dist}_G(u, v) = 2$
- Inequality is satisfied:

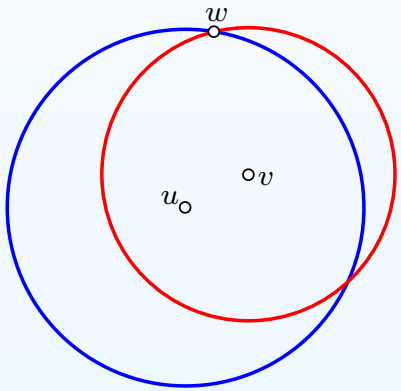
$$t_v - t_u = 3 > 2 = \mathrm{dist}_G(u, v)$$

- The wave of v never catches up with the wave of u:



Proof of Claim by Contraposition

- Assume some w sends "wave" twice in the same round t, one originating from u and one from v (waves interfere)
- Need to show the opposite inequality: $t_v - t_u \leq \mathrm{dist}_G(u, v)$



- By the triangle inequality,

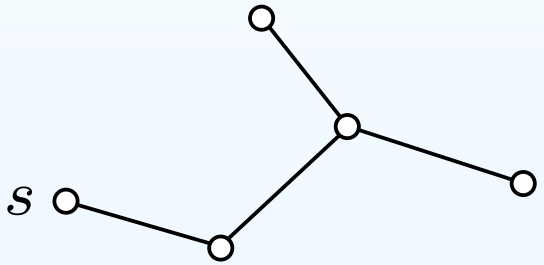
$$\mathrm{dist}_G(u, w) \leq \mathrm{dist}_G(u, v) + \mathrm{dist}_G(v, w) \Leftrightarrow$$
$$\mathrm{dist}_G(u, w) - \mathrm{dist}_G(v, w) \leq \mathrm{dist}_G(u, v)$$

- It takes $\mathrm{dist}_G(u, w)$ rounds for "wave" to reach w from u
- It takes $\mathrm{dist}_G(v, w)$ rounds for "wave" to reach w from v
- Since w sends "wave" originating from both u and v in round t:

$$t = t_u + \mathrm{dist}_G(u, w) = t_v + \mathrm{dist}_G(v, w)$$
$$\Leftrightarrow t_v - t_u = \mathrm{dist}_G(u, w) - \mathrm{dist}_G(v, w) \leq \mathrm{dist}_G(u, v)$$

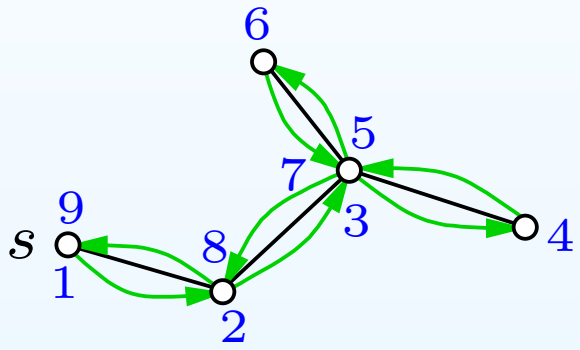
APSP Algorithm: Moving Token

- A BFS tree T is constructed from an elected leader s



APSP Algorithm: Moving Token

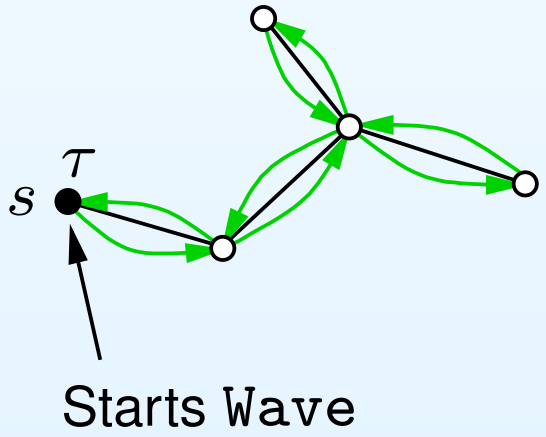
- Let W be a DFS walk of T, starting and ending in s



APSP Algorithm: Moving Token

- A *token* τ is then sent around T, following W
- When a vertex v has τ, it:
  - starts the `Wave` algorithm if it has not previously had the token,
  - waits a turn before passing τ on to the next vertex of W

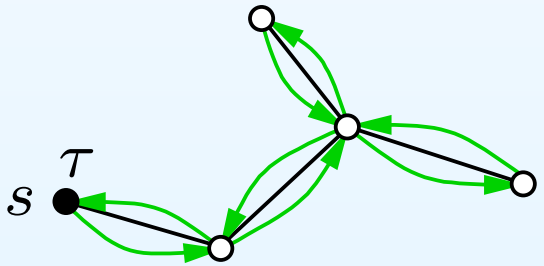
$t = 1$



APSP Algorithm: Moving Token

- A *token* τ is then sent around T, following W
- When a vertex v has τ, it:
  - starts the `Wave` algorithm if it has not previously had the token,
  - waits a turn before passing τ on to the next vertex of W

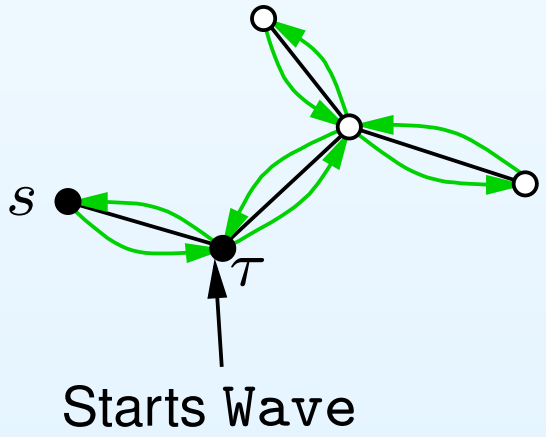
$$t = 2$$



APSP Algorithm: Moving Token

- A *token* τ is then sent around T, following W
- When a vertex v has τ, it:
  - starts the `Wave` algorithm if it has not previously had the token,
  - waits a turn before passing τ on to the next vertex of W

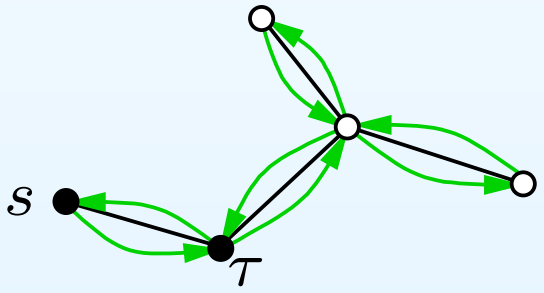
$$t = 3$$



APSP Algorithm: Moving Token

- A *token* τ is then sent around T, following W
- When a vertex v has τ, it:
  - starts the `Wave` algorithm if it has not previously had the token,
  - waits a turn before passing τ on to the next vertex of W

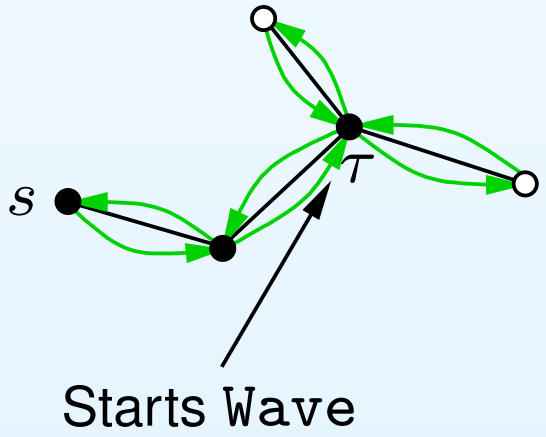
$$t = 4$$



APSP Algorithm: Moving Token

- A *token* τ is then sent around T, following W
- When a vertex v has τ, it:
  - starts the `Wave` algorithm if it has not previously had the token,
  - waits a turn before passing τ on to the next vertex of W

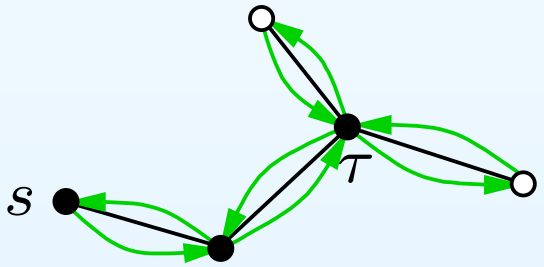
$t = 5$



APSP Algorithm: Moving Token

- A *token* τ is then sent around T, following W
- When a vertex v has τ, it:
  - starts the `Wave` algorithm if it has not previously had the token,
  - waits a turn before passing τ on to the next vertex of W

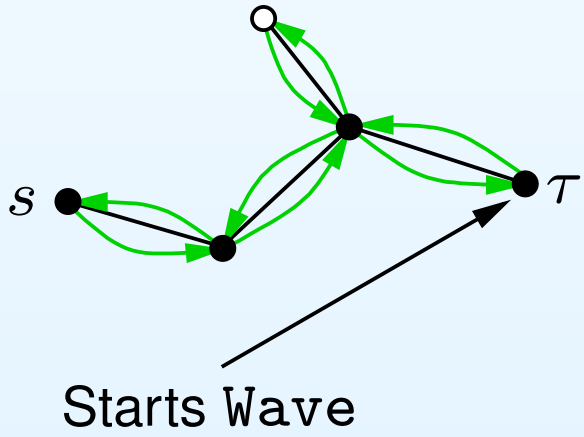
$$t = 6$$



APSP Algorithm: Moving Token

- A *token* τ is then sent around T, following W
- When a vertex v has τ, it:
  - starts the `Wave` algorithm if it has not previously had the token,
  - waits a turn before passing τ on to the next vertex of W

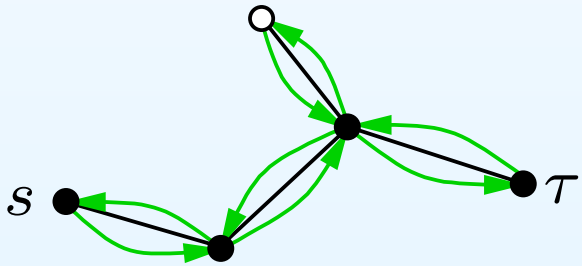
$$t = 7$$



APSP Algorithm: Moving Token

- A *token* τ is then sent around T, following W
- When a vertex v has τ, it:
  - starts the `Wave` algorithm if it has not previously had the token,
  - waits a turn before passing τ on to the next vertex of W

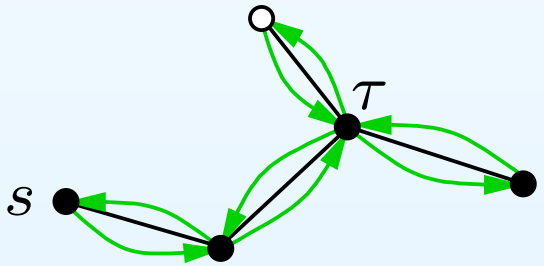
$$t = 8$$



APSP Algorithm: Moving Token

- A *token* τ is then sent around T, following W
- When a vertex v has τ, it:
  - starts the `Wave` algorithm if it has not previously had the token,
  - waits a turn before passing τ on to the next vertex of W

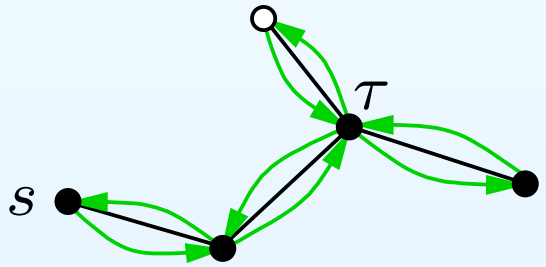
$$t = 9$$



APSP Algorithm: Moving Token

- A *token* τ is then sent around T, following W
- When a vertex v has τ, it:
  - starts the `Wave` algorithm if it has not previously had the token,
  - waits a turn before passing τ on to the next vertex of W

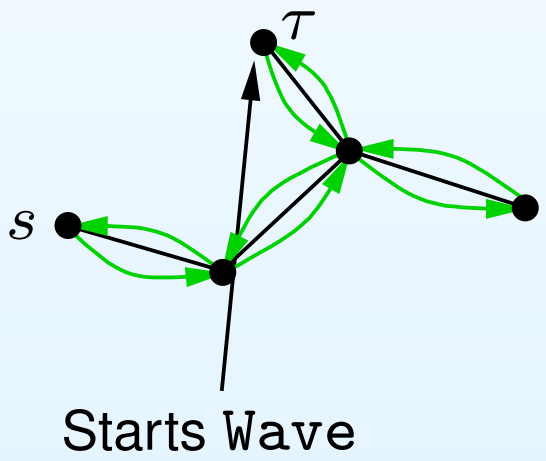
$$t = 10$$



APSP Algorithm: Moving Token

- A *token* τ is then sent around T, following W
- When a vertex v has τ, it:
  - starts the `Wave` algorithm if it has not previously had the token,
  - waits a turn before passing τ on to the next vertex of W

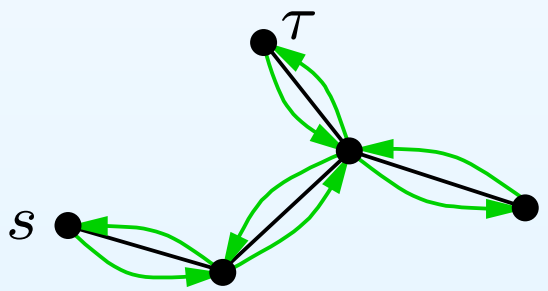
$$t = 11$$



APSP Algorithm: Moving Token

- A *token* τ is then sent around T, following W
- When a vertex v has τ, it:
  - starts the `Wave` algorithm if it has not previously had the token,
  - waits a turn before passing τ on to the next vertex of W

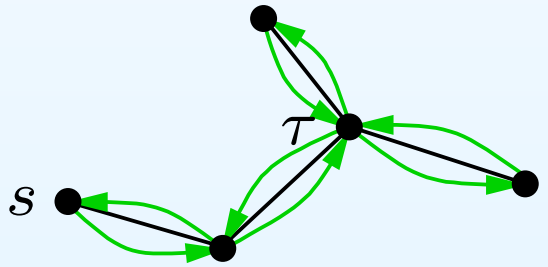
$$t = 12$$



APSP Algorithm: Moving Token

- A *token* τ is then sent around T, following W
- When a vertex v has τ, it:
  - starts the `Wave` algorithm if it has not previously had the token,
  - waits a turn before passing τ on to the next vertex of W

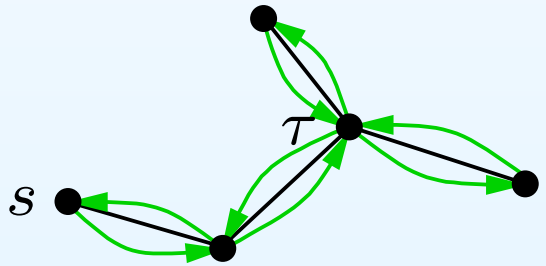
$$t = 13$$



APSP Algorithm: Moving Token

- A *token* τ is then sent around T, following W
- When a vertex v has τ, it:
  - starts the `Wave` algorithm if it has not previously had the token,
  - waits a turn before passing τ on to the next vertex of W

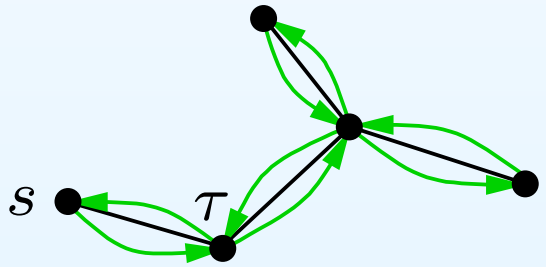
$$t = 14$$



APSP Algorithm: Moving Token

- A *token* τ is then sent around T, following W
- When a vertex v has τ, it:
  - starts the `Wave` algorithm if it has not previously had the token,
  - waits a turn before passing τ on to the next vertex of W

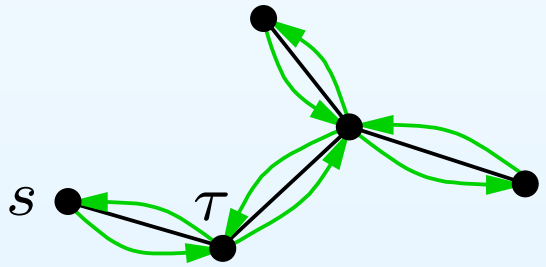
$$t = 15$$



APSP Algorithm: Moving Token

- A *token* τ is then sent around T, following W
- When a vertex v has τ, it:
  - starts the `Wave` algorithm if it has not previously had the token,
  - waits a turn before passing τ on to the next vertex of W

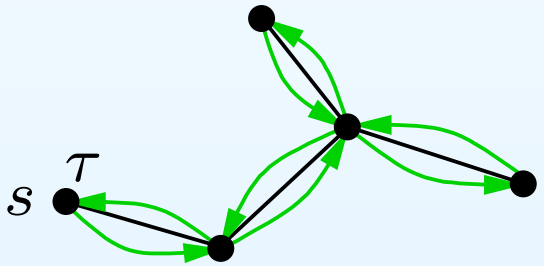
$$t = 16$$



APSP Algorithm: Moving Token

- A *token* τ is then sent around T, following W
- When a vertex v has τ, it:
  - starts the `Wave` algorithm if it has not previously had the token,
  - waits a turn before passing τ on to the next vertex of W

$$t = 17$$



APSP Algorithm Analysis: No Waves Interfere

- Want to show our earlier inequality: for any u and v with $t_v > t_u$, the waves for u and v do not interfere:

$$t_v - t_u > \mathrm{dist}_G(u, v)$$

- t_u is the earliest round when u has token τ
- t_v is the earliest round when v has token τ
- τ moves from u to v between rounds t_u and t_v
- Let $\mathrm{dist}_W(u, v)$ be number of edges traversed by τ between these rounds
- Round number increases by 2 each time τ moves by one edge, so

$$t_v - t_u = 2\,\mathrm{dist}_W(u, v) \geq 2\,\mathrm{dist}_G(u, v) > \mathrm{dist}_G(u, v)$$

- Since this holds for any u and v, no waves can interfere
- Thus, the bandwidth requirement of the CONGEST model is satisfied:
  - Each message is only $O(\log n)$ bits long

APSP Algorithm: Number of Communication Rounds

- Computing the initial BFS tree T takes $O(\mathrm{diam}(G))$ rounds
- Walk W visits each edge of T twice
- T has $n-1$ edges so the token moves $2n-2$ times
- It moves every second round so $O(n)$ rounds to complete the walk
- Thus, every execution of `Wave` starts within $O(n)$ rounds and takes an additional $O(\mathrm{diam}(G))$ rounds to finish
- Total number of rounds for our APSP algorithm:

$$O(n + \mathrm{diam}(G)) = O(n)$$