6.852 Lecture 9

- Lower bound on leader election
- Basic asynchronous network algorithms
  - constructing a spanning tree
  - breadth-first search
  - shortest paths
  - minimum spanning tree
- Reading: Chapter 15 (continued)
- Next lecture: Chapter 16.

Last lecture

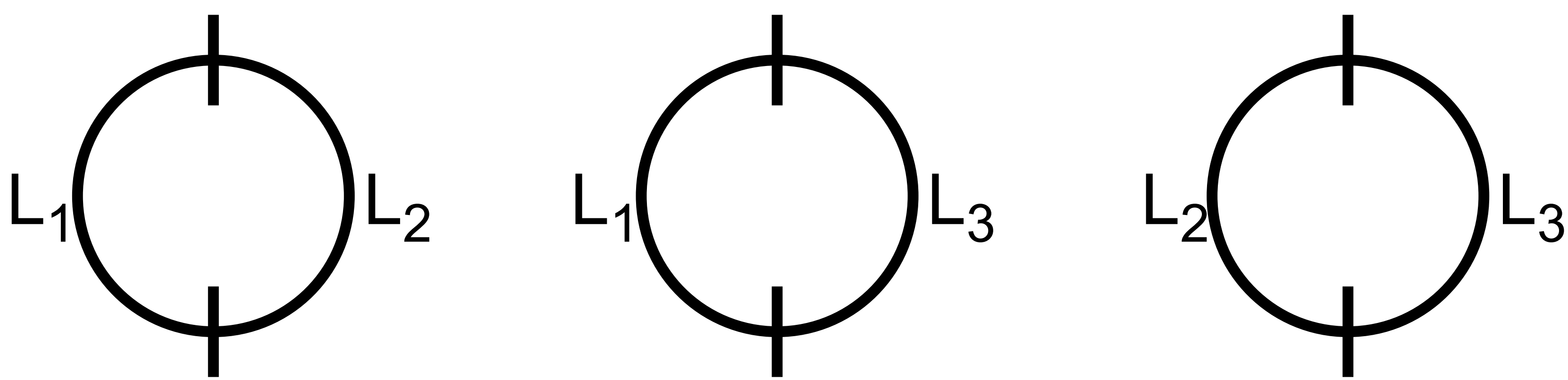
- Finished defining formal model
- Leader election algorithm for asynchronous networks
- Described lower bound for leader election

Leader election in a ring

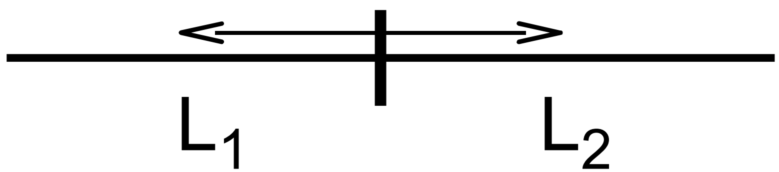
- Lower bound in asynchronous network if n is unknown
  - Key: “assemble” ring from pieces which delay communication
    - **silent** state: no more messages will be sent without input
    - ring looks like “line” if communication delayed across ends
    - some lines may send $\Omega(n \log n)$ msgs before becoming silent
    - connect ends of such a line to make a ring
      - delay communication across ends of line

Lower bound on leader election

- $C(\alpha)$ = number of messages sent in α
- $C(A) = \sup\{ C(\alpha) \mid \alpha$ is an input-free execution of $A \}$
- Lemma 1: If L_1, L_2, L_3 are three line graphs of length l such that $C(L_i) \geq k$ for all i then $C(L_i \text{ join } L_j) \geq 2k + l/2$ for some $i \neq j$
  - Suppose not. Consider three rings:

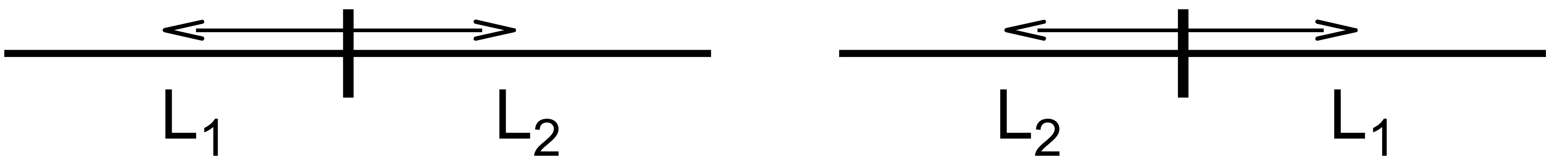


Lower bound on leader election



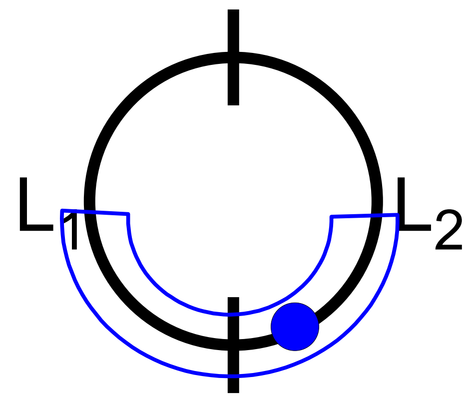
- Let α_i be finite execution of L_i with ≥ k msgs.
- Run α_1 then α_2 then $\alpha_{1,2}$, with msgs across boundary
  - since fewer than l/2 add'l msgs, middles of L_1 & L_2 still silent
    - not enough msgs to reach them

Lower bound on leader election



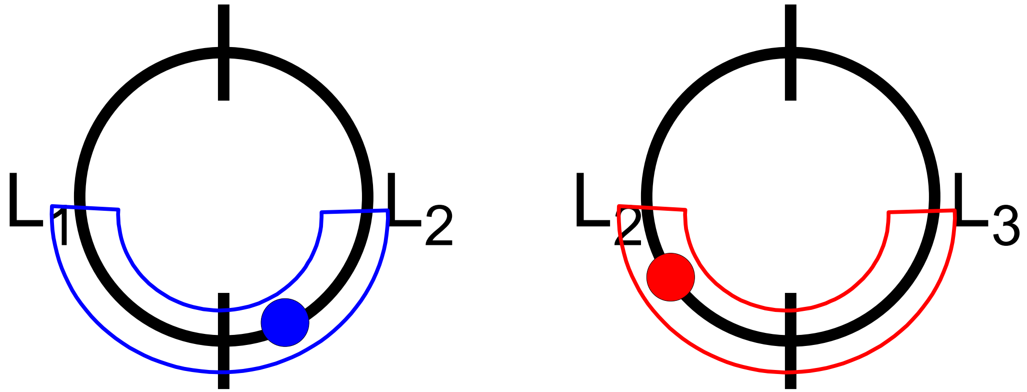
- Let α_i be finite execution of L_i with ≥ k msgs.
- Run α_1 then α_2 then $\alpha_{1,2}$, with msgs across boundary
  - since fewer than l/2 add'l msgs, middles of L_1 & L_2 still no input
    - not enough msgs to reach them
- Similarly for $\alpha_{2,1}$.
  - no interference between $\alpha_{1,2}$ and $\alpha_{2,1}$

Lower bound on leader election



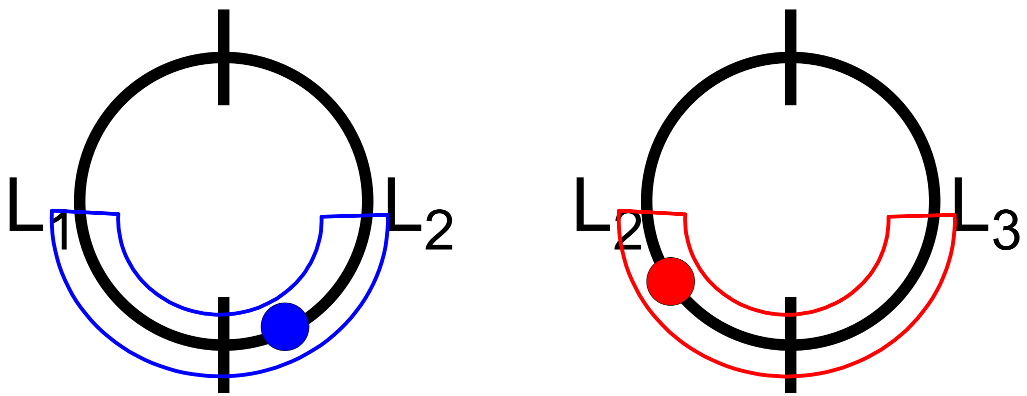
- Connect both ends into ring
  - left neighbor is clockwise around ring
- Run α_1 then α_2 then $\alpha_{1,2}$ then $\alpha_{2,1}$.
  - must be silent in final state
  - must elect leader (possibly in extension, but no more msgs)
- Assume WLOG that elected leader is in “bottom half”
  - can't be midpoint of either L_1 or L_2

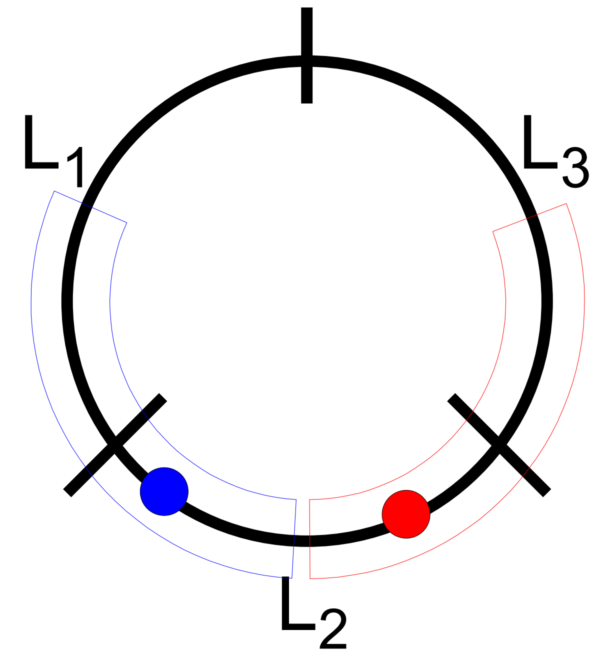
Lower bound on leader election



- Same argument for ring(L_2 join L_3)
  - Can leader be in bottom half?

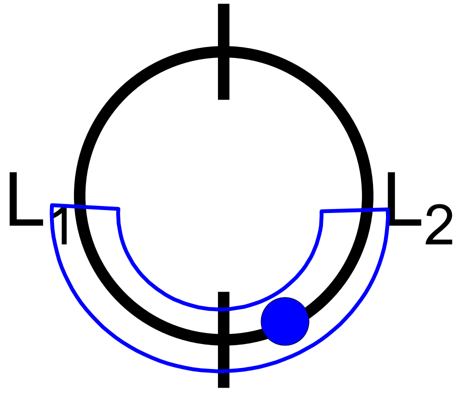
Lower bound on leader election

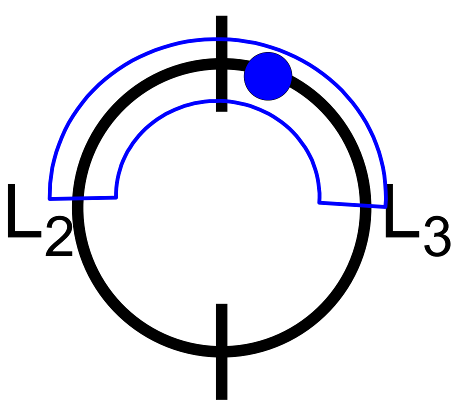




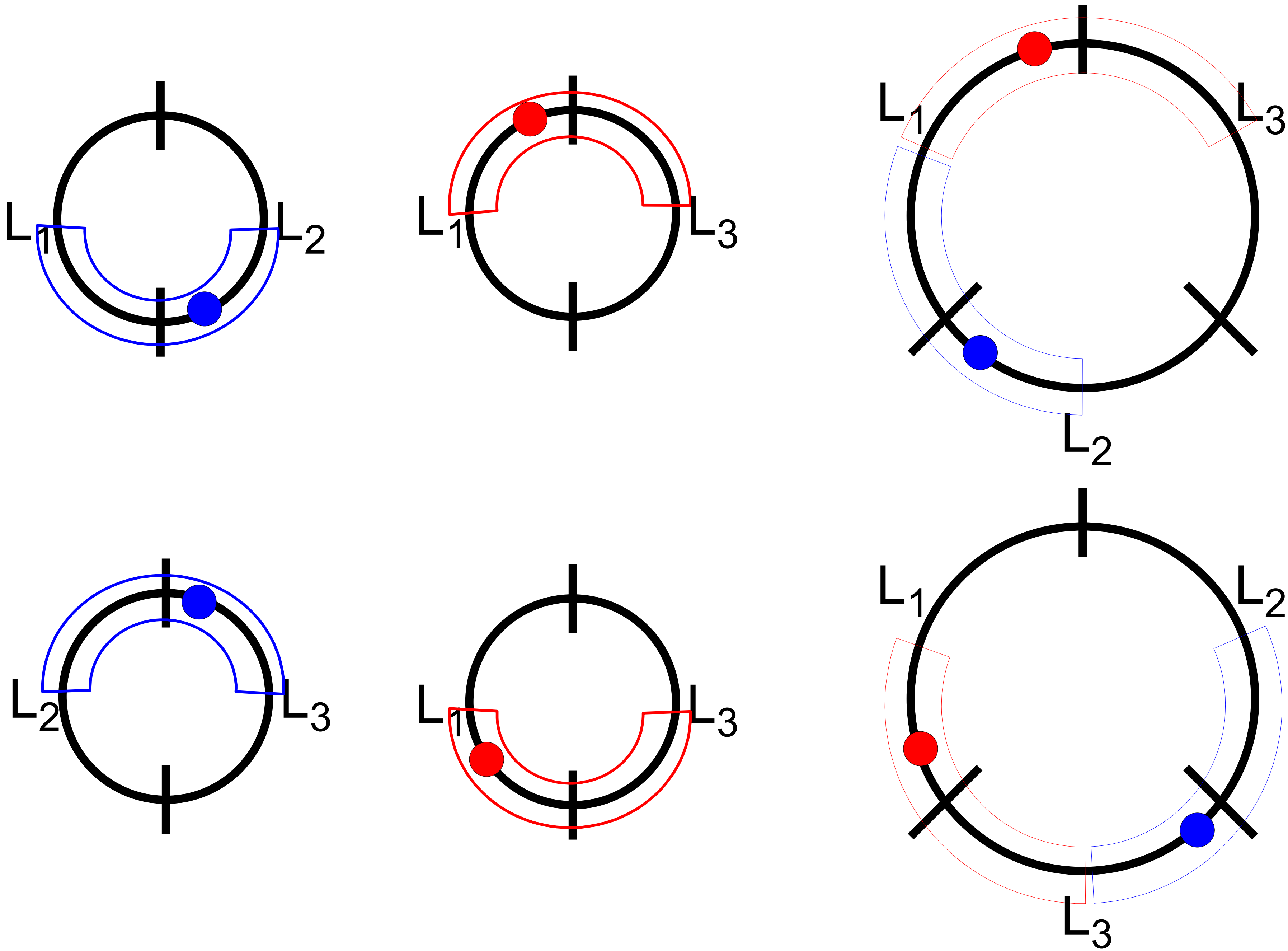
- Same argument for ring(L_2 join L_3)
  - Can leader be in bottom half? **No!**
  - so must be in top half

Lower bound on leader election





Lower bound on leader election



Lower bound on leader election

- Lemma 2: There are an infinite number of processes that can send a message before receiving any.

Lower bound on leader election

- Lemma 1: If L_1, L_2, L_3 are three line graphs of length l such that $C(L_i) \geq k$ for all i, then $C(L_i \text{ join } L_j) \geq 2k + l/2$ for some $i \neq j$.
- Lemma 2: There are an infinite number of processes that can send a message before receiving any.
- Lemma 3: For any $r \geq 0$, there are infinitely many disjoint line graphs L of length 2^r such that $C(L) \geq r\, 2^{r-2}$.
  - base case ($r = 0$): Trivial.
  - base case ($r = 1$): Use Lemma 1.
  - inductive case ($r \geq 2$):
    - Choose L_1, L_2, L_3 of length 2^{r-1} with $C(L_i) \geq (r-1)\, 2^{r-3}$.
    - By Lemma 2, for some i,j, $C(L_i \text{ join } L_j) \geq 2(r-1)2^{r-3} + 2^{r-1}/2 = r\, 2^{r-2}$.

Lower bound on leader election

- Lemma 3: For any $r \geq 0$, there are infinitely many disjoint line graphs L of length 2^r such that $C(L) \geq r\ 2^{r-2}$.
- Theorem: For any $r \geq 0$, there is a ring R of size $n = 2^r$ such that $C(R) = \Omega(n \log n)$.
  - Choose L of length 2^r such that $C(L) \geq r\ 2^{r-2}$.
  - Connect ends, but delay communication across boundary.
    - line graph by itself must never elect leader
- Corollary: For any $n \geq 0$, there is a ring R of size n such that $C(R) = \Omega(n \log n)$.

Leader election in general network

- Can get asynchronous version of synchronous alg
  - can simulate rounds with counters
  - need to know diameter for termination
- Better algorithms later
  - no need to know diameter
  - lower message complexity

Spanning trees and searching

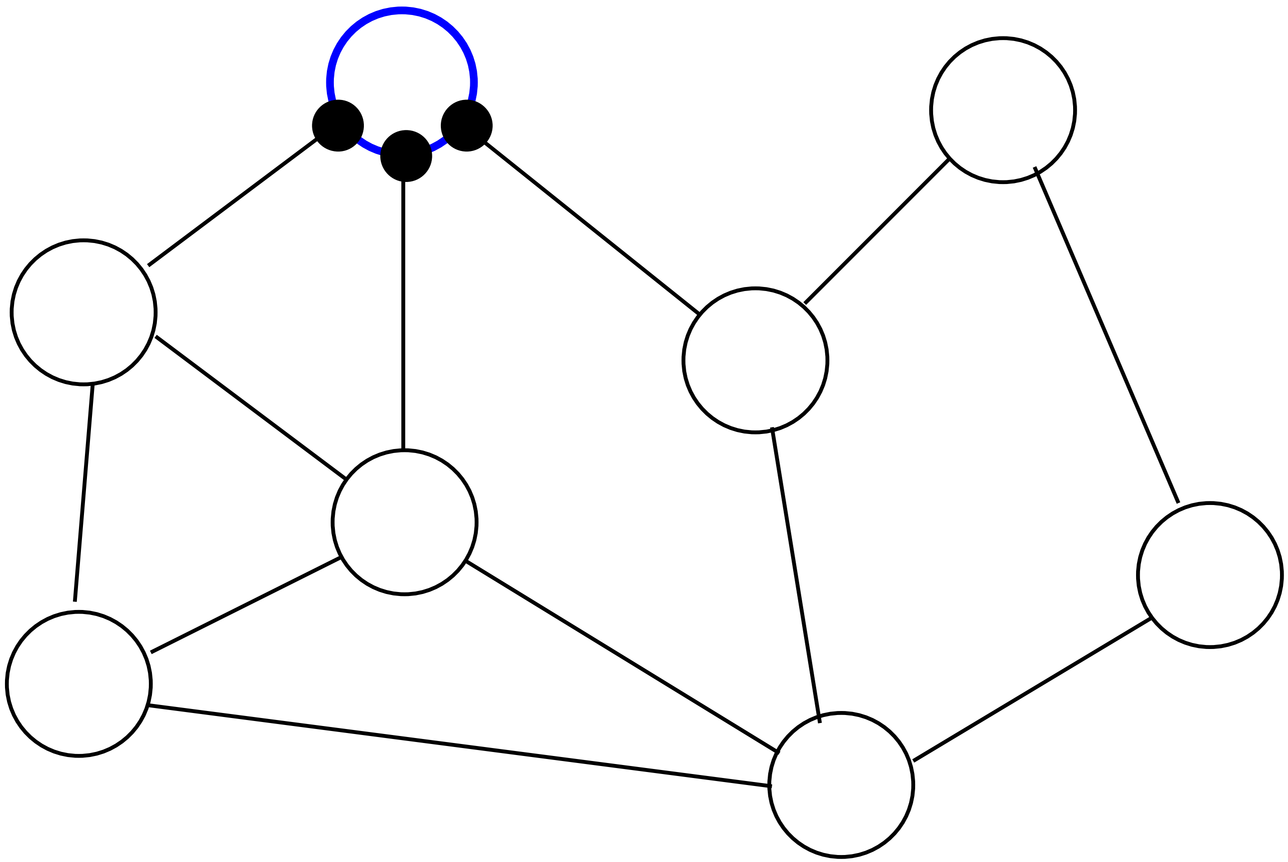
- Spanning trees used for broadcast/convergecast
- Assume (for rest of these algorithms)
  - undirected graph (i.e., bidirectional communication)
  - root i_0
  - size and diameter unknown
  - can identify in- and out-edges to same neighbor
- Problem: each process outputs parent in tree
- Start from SynchBFS algorithm
  - i_0 “flood” search msg; parent is first that sends it to process
  - still yields spanning tree in asynchronous network, but not necessarily breadth-first tree

AsynchSpanningTree

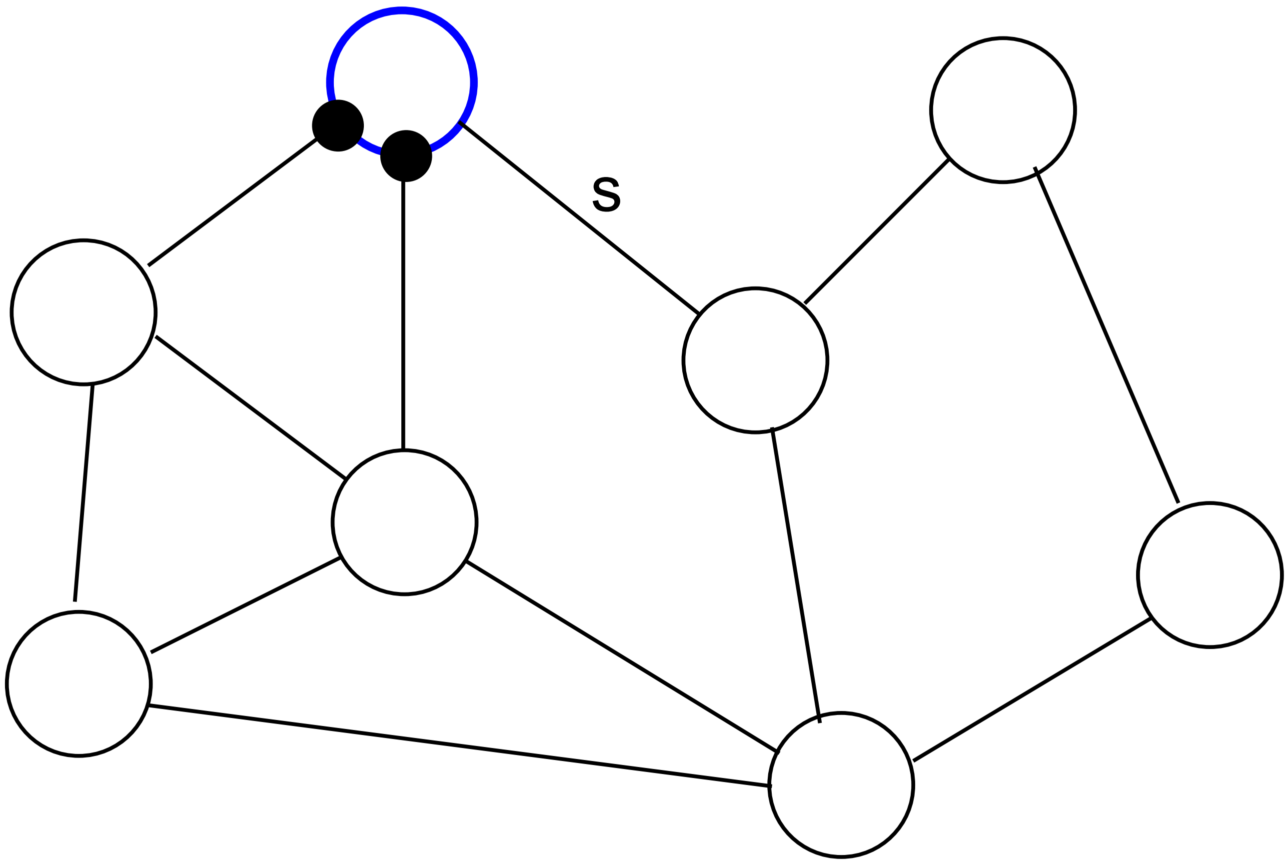
- Signature
  - ***in*** receive(“search”)$_{j,i}$, $j \in$ nbrs
  - ***out*** send(“search”)$_{i,j}$, $j \in$ nbrs
  - ***out*** parent(j)$_i$, $j \in$ nbrs
- State
  - **parent**: nbrs U { null }; init null
  - **reported**: Boolean; init false
  - for each $j \in$ nbrs
    - **send**(j) $\in$ { search, null }; init search iff $i = i_0$

- send(“search”)$_{i,j}$
  pre: **send**(j) = search
  eff: **send**(j) := null
- receive(“search”)$_{j,i}$
  eff: if $i \neq i_0$ and **parent** = null then
  parent := j
  for $k \in$ nbrs - { j } do
  send(k) := search
- parent(j)$_i$
  pre: **parent** = j
  reported = false
  eff: **reported** := true

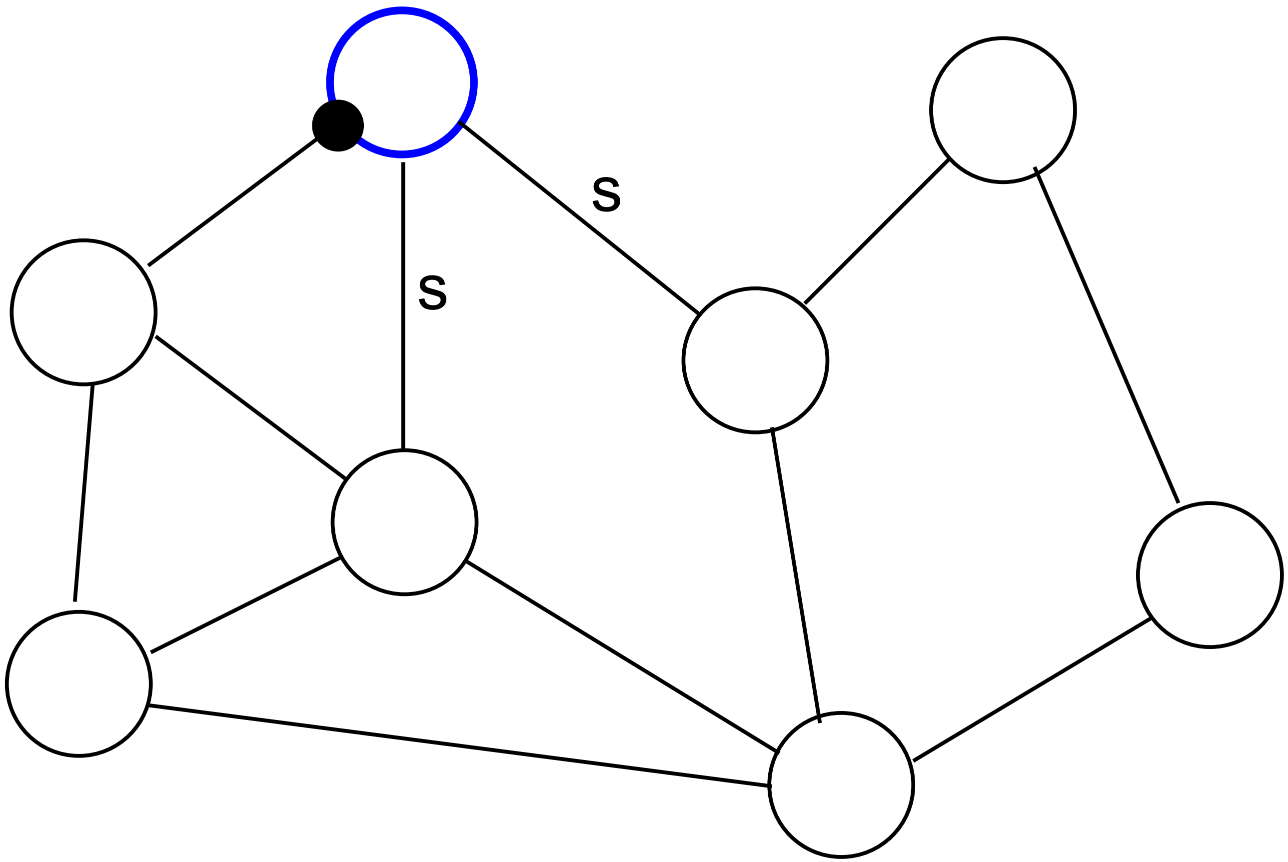
AsynchSpanningTree



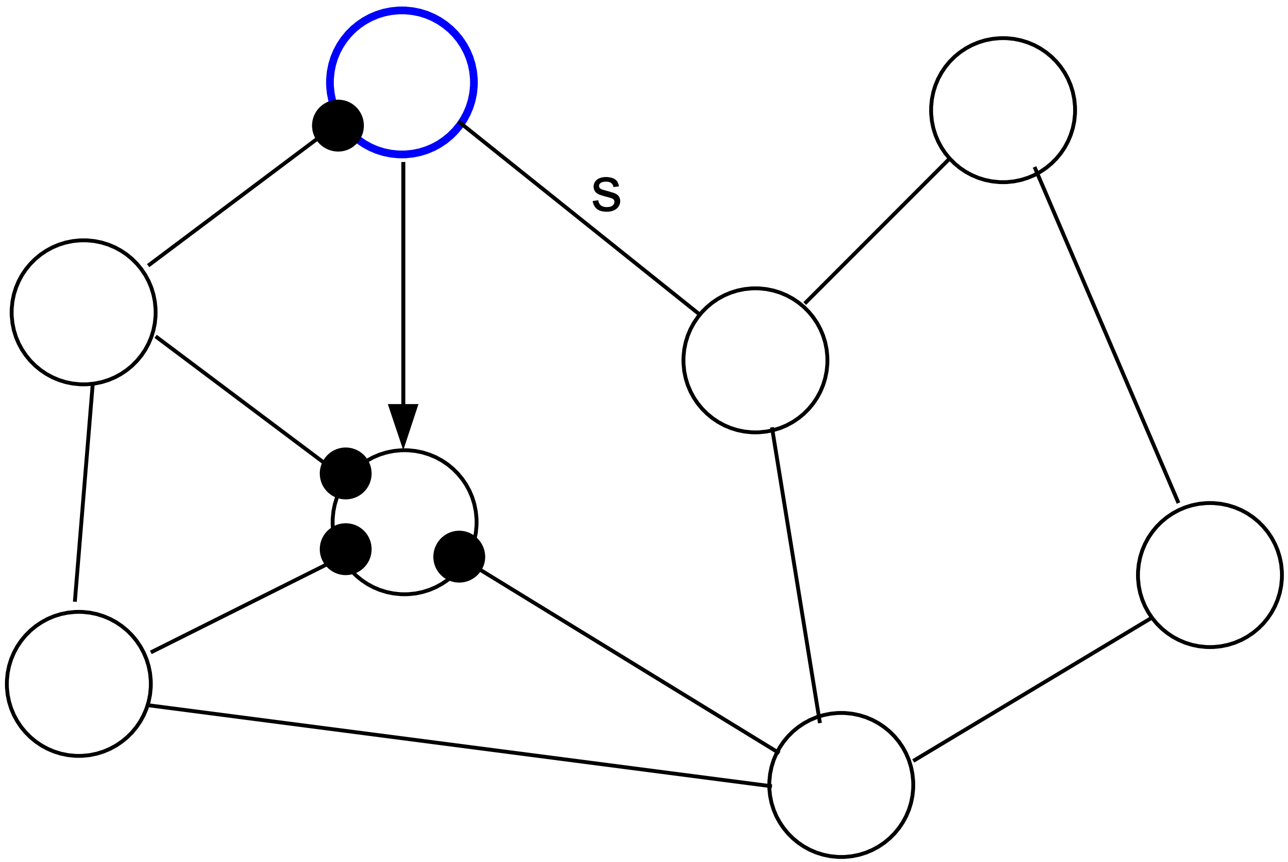
AsynchSpanningTree



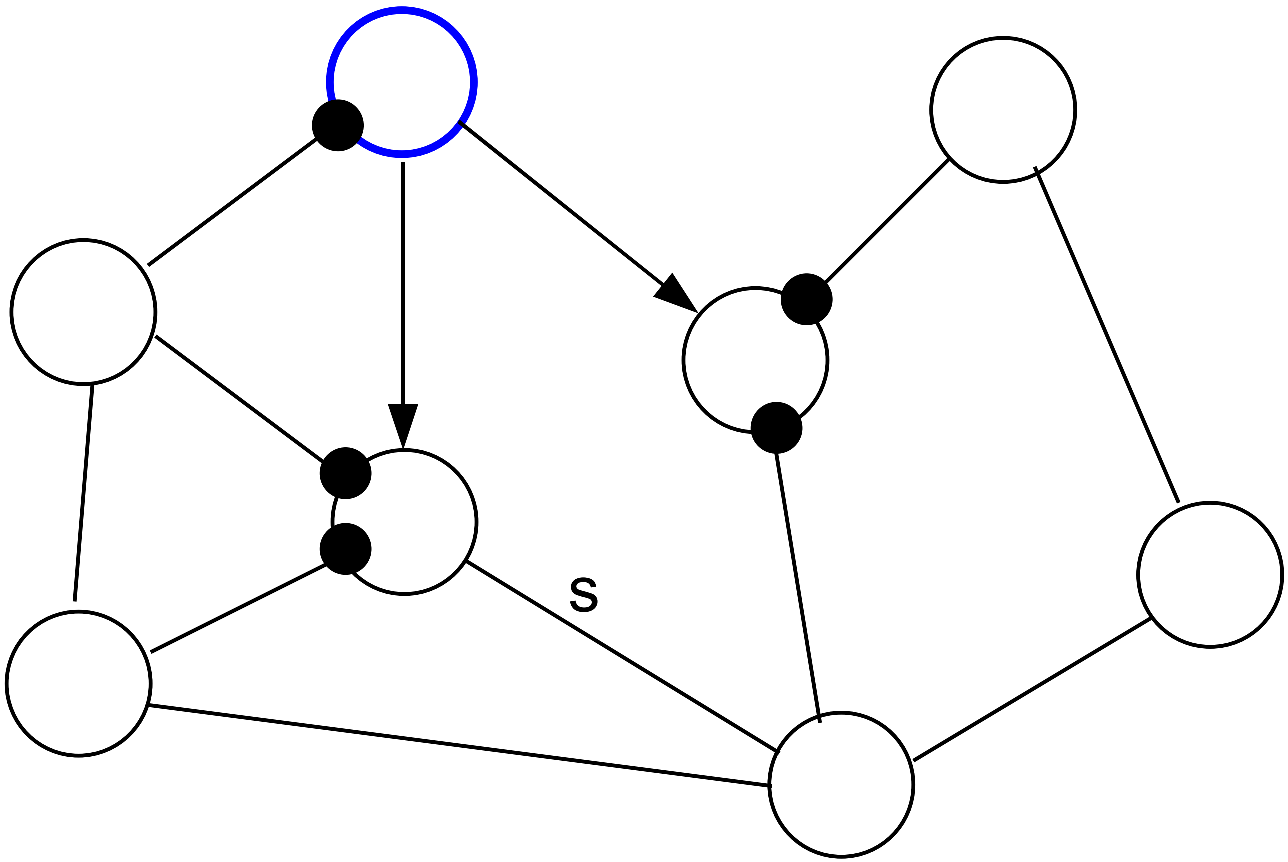
AsynchSpanningTree



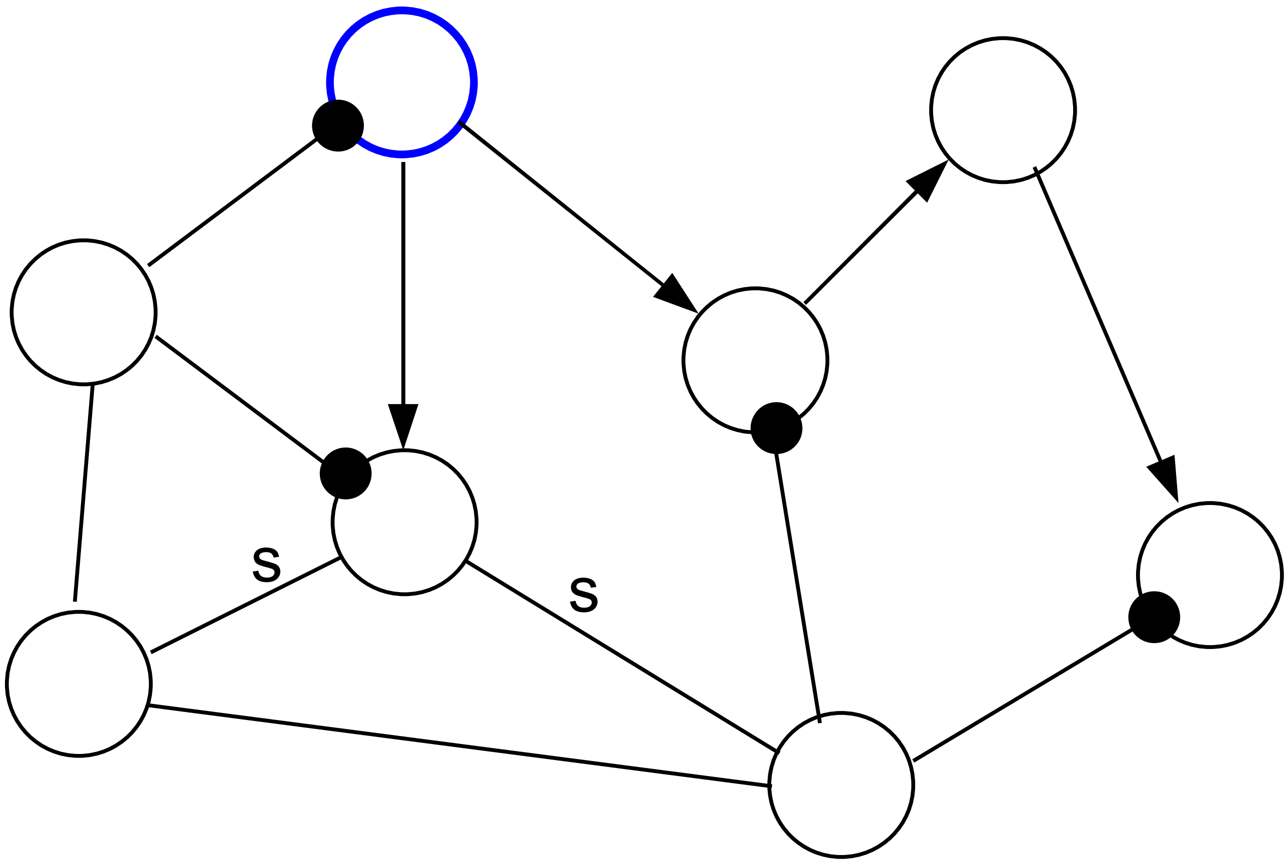
AsynchSpanningTree



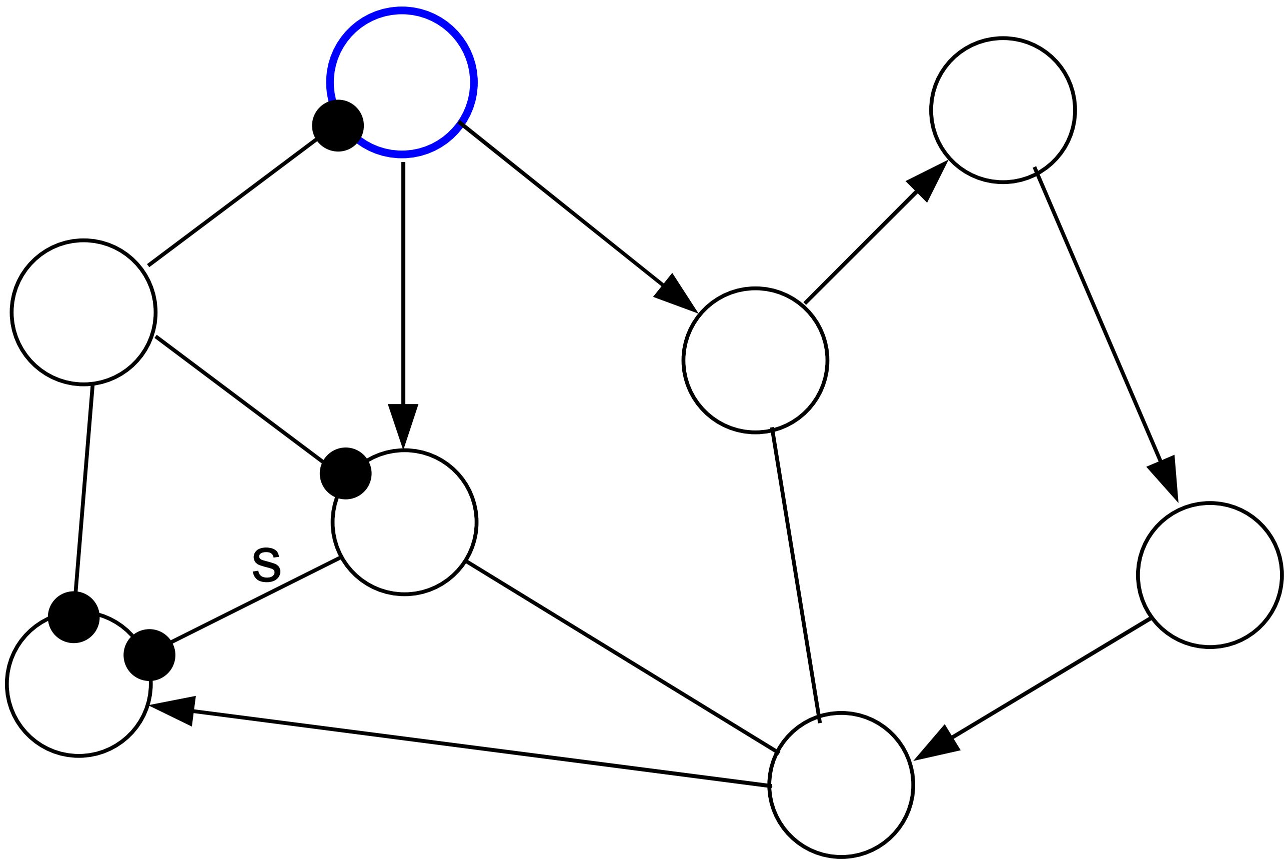
AsynchSpanningTree



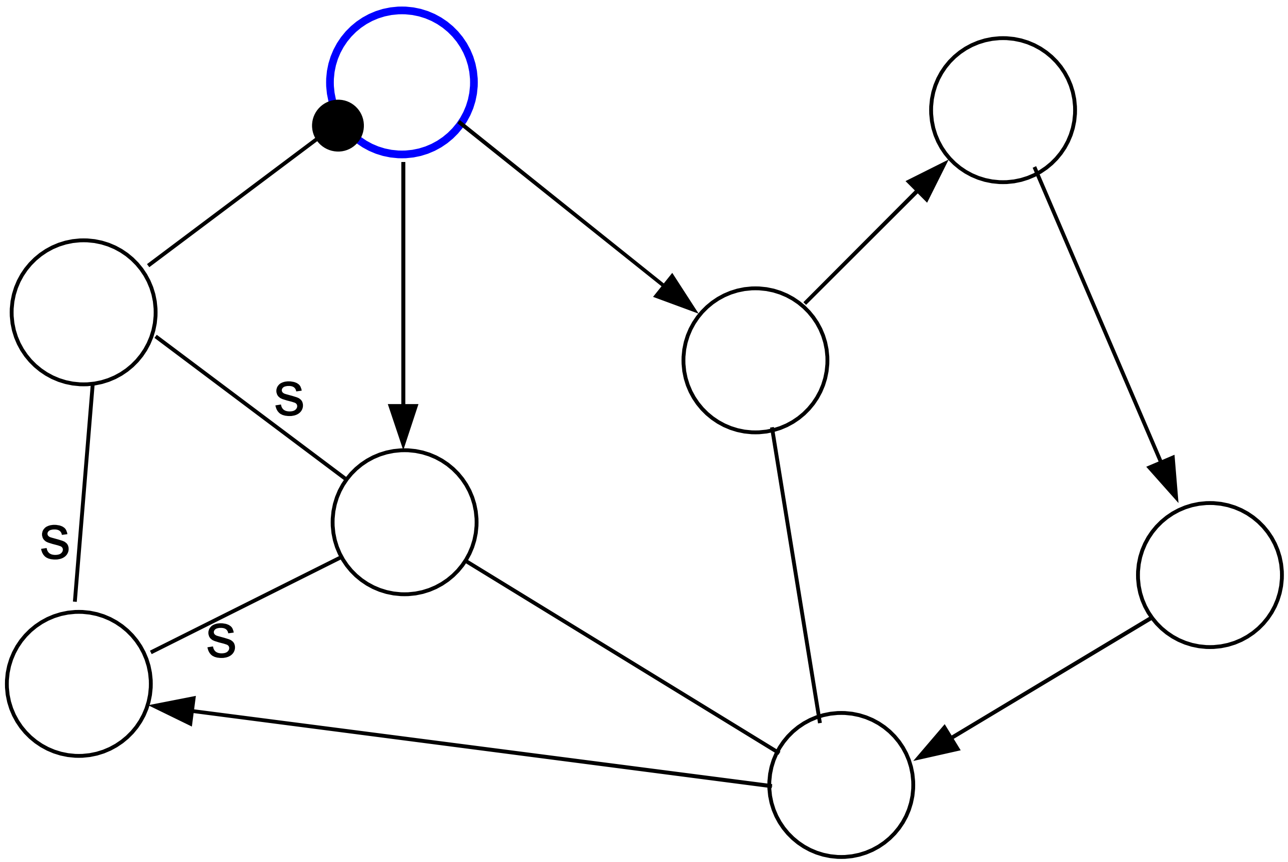
AsynchSpanningTree



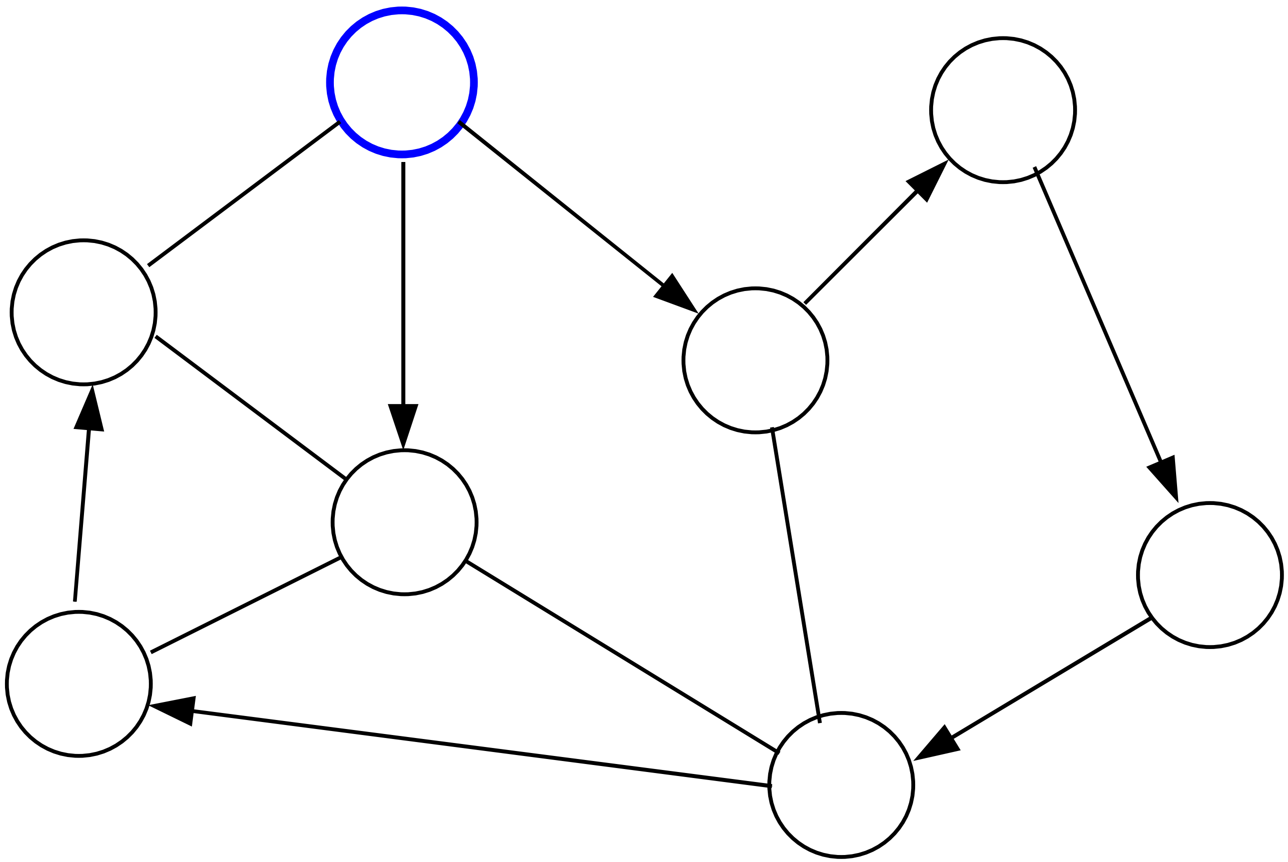
AsynchSpanningTree



AsynchSpanningTree

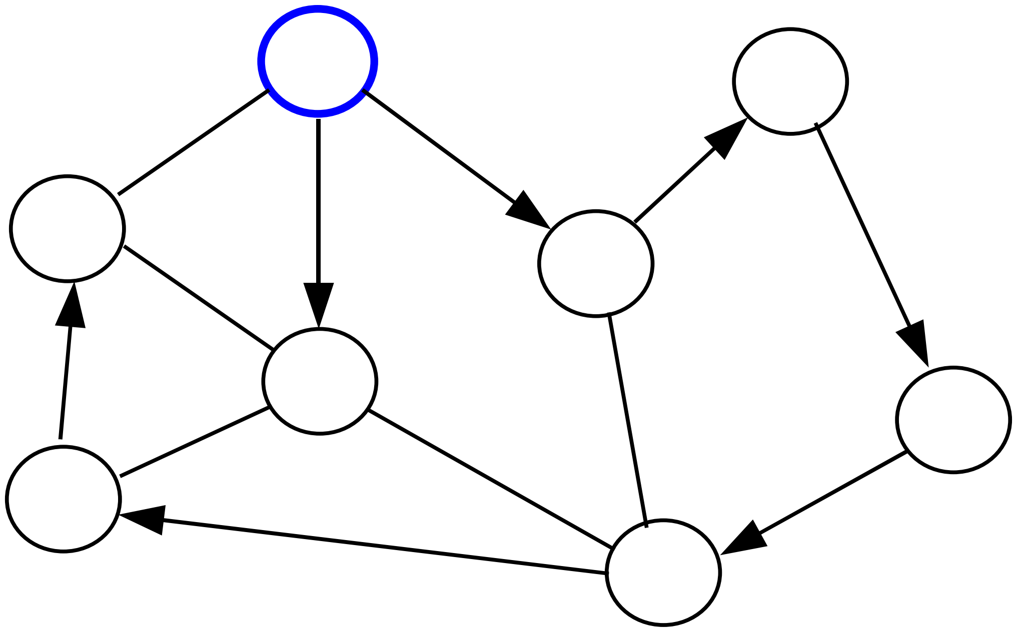


AsynchSpanningTree



AsynchSpanningTree

- Complexity
  - msg: O(|E|)
  - time: (diam) (l+d) + l
- Anomaly: Paths may be longer than diameter!
  - messages may travel faster along longer paths



AsynchSpanningTree

- Applications of spanning tree (as in synchronous alg)
  - message broadcast: piggyback on search msg
  - child pointers: easy because of bidirectional communication
  - use precomputed tree to do broadcast/convergecast
    - O(n) msg complexity; O(h(l+d)) time complexity
  - see book for details

h = height of tree; may be n

Breadth-first search

- In asynchronous networks, “SynchBFS” does not guarantee spanning tree constructed is breadth-first
  - long paths may be traversed faster than short ones
- We can modify each process to keep track of distance, change parent when it hears of shorter path.
  - relaxation algorithm (like Bellman-Ford)
  - must inform neighbors of change
  - eventually tree stabilizes into breadth-first spanning tree

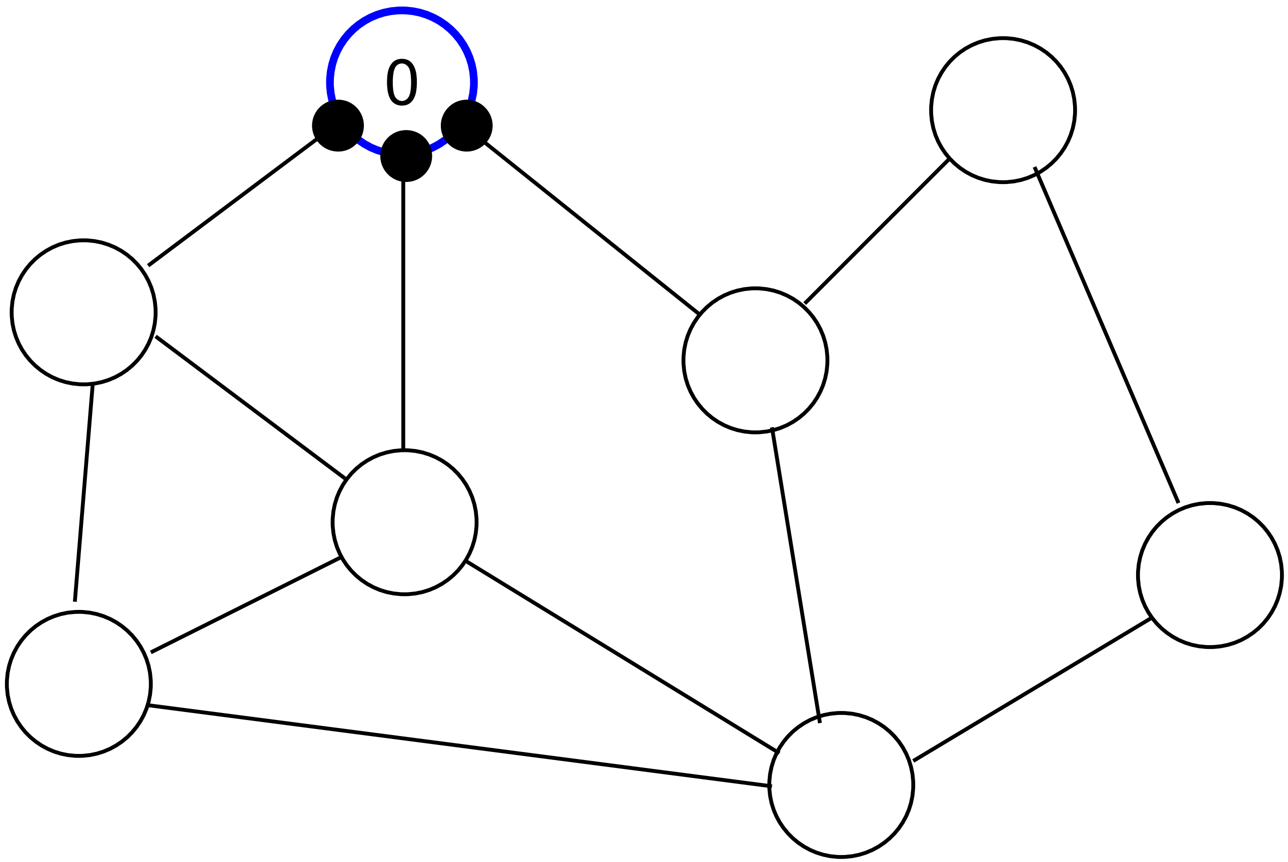
AsynchBFS

- Signature
  - ***in*** receive$(m)_{j,i}$, $m \in$ **N,** $j \in$ nbrs
  - ***out*** send$(m)_{i,j}$, $m \in$ **N**, $j \in$ nbrs
- State
  - **dist**: **N** U { ∞ };
    init 0 if $i = i_0$, else ∞
  - **parent**: nbrs U { null }
  - for each $j \in$ nbrs
    - **send**(j): FIFO queue of **N**;
      init { 0 } if $i = i_0$, else ∅

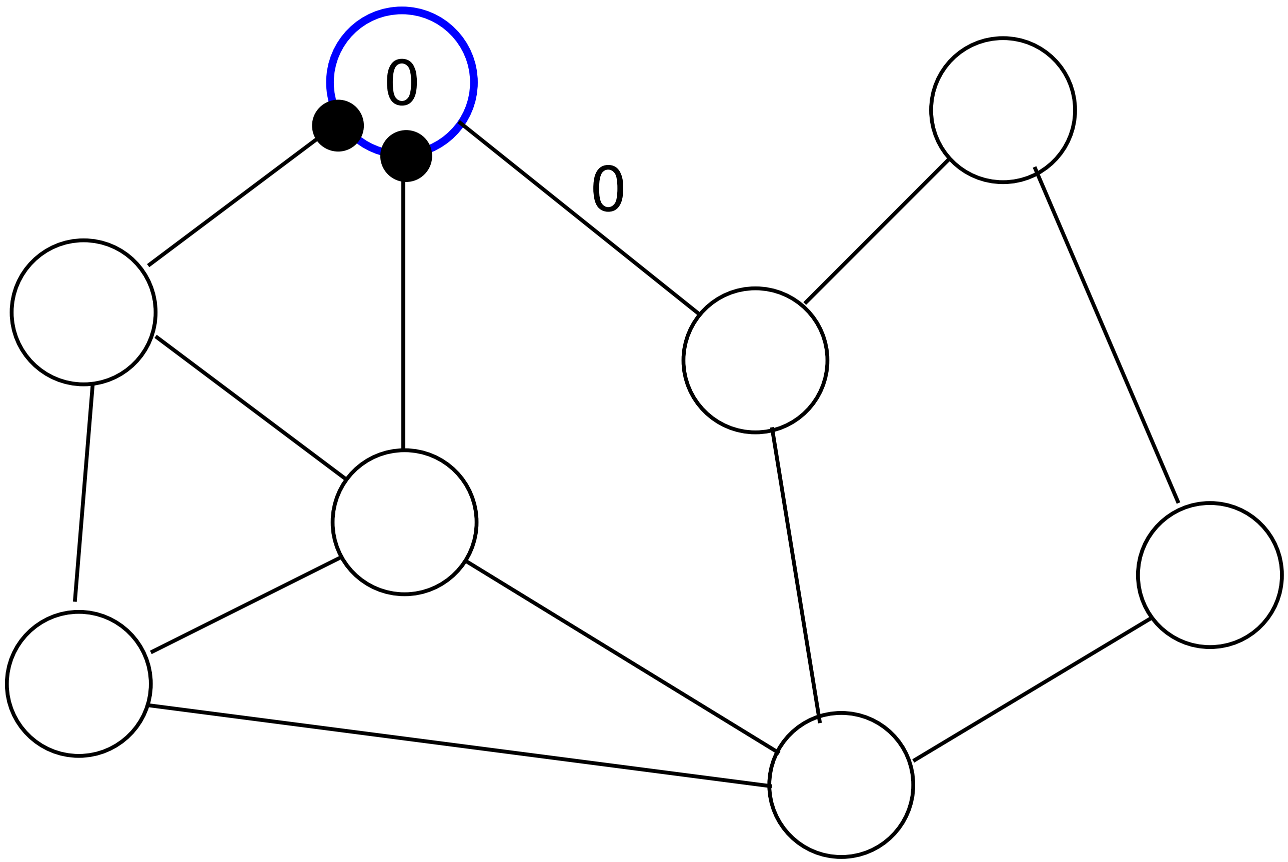
- send$(m)_{i,j}$
  pre: m is head of **send**(j)
  eff: remove head of **send**(j)
- receive$(m)_{j,i}$
  eff: if m+1 < **dist** then
  dist := m+1
  parent := j
  for $k \in$ nbrs - { j } do
  add **dist** to **send**(k)

- No parent actions.
  - no one knows when it's done

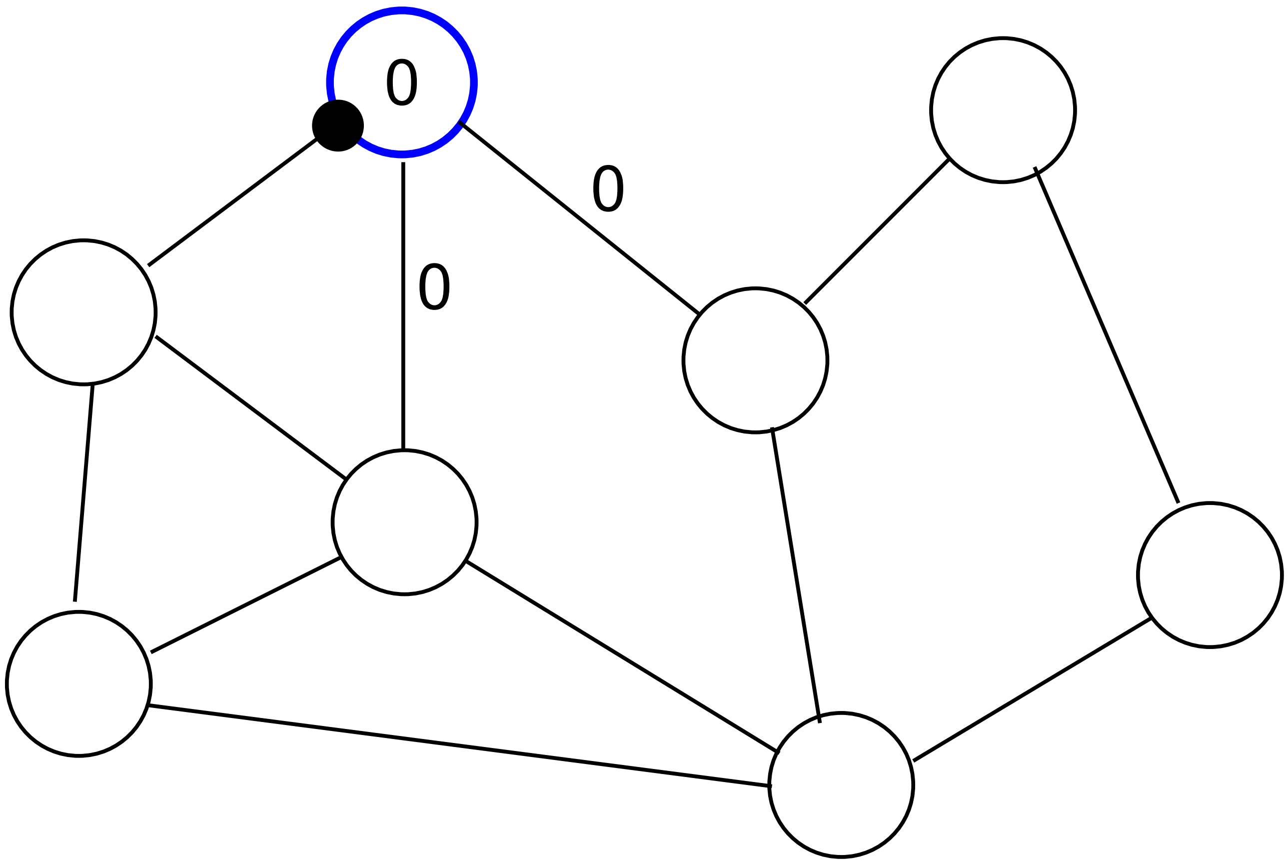
AsynchBFS



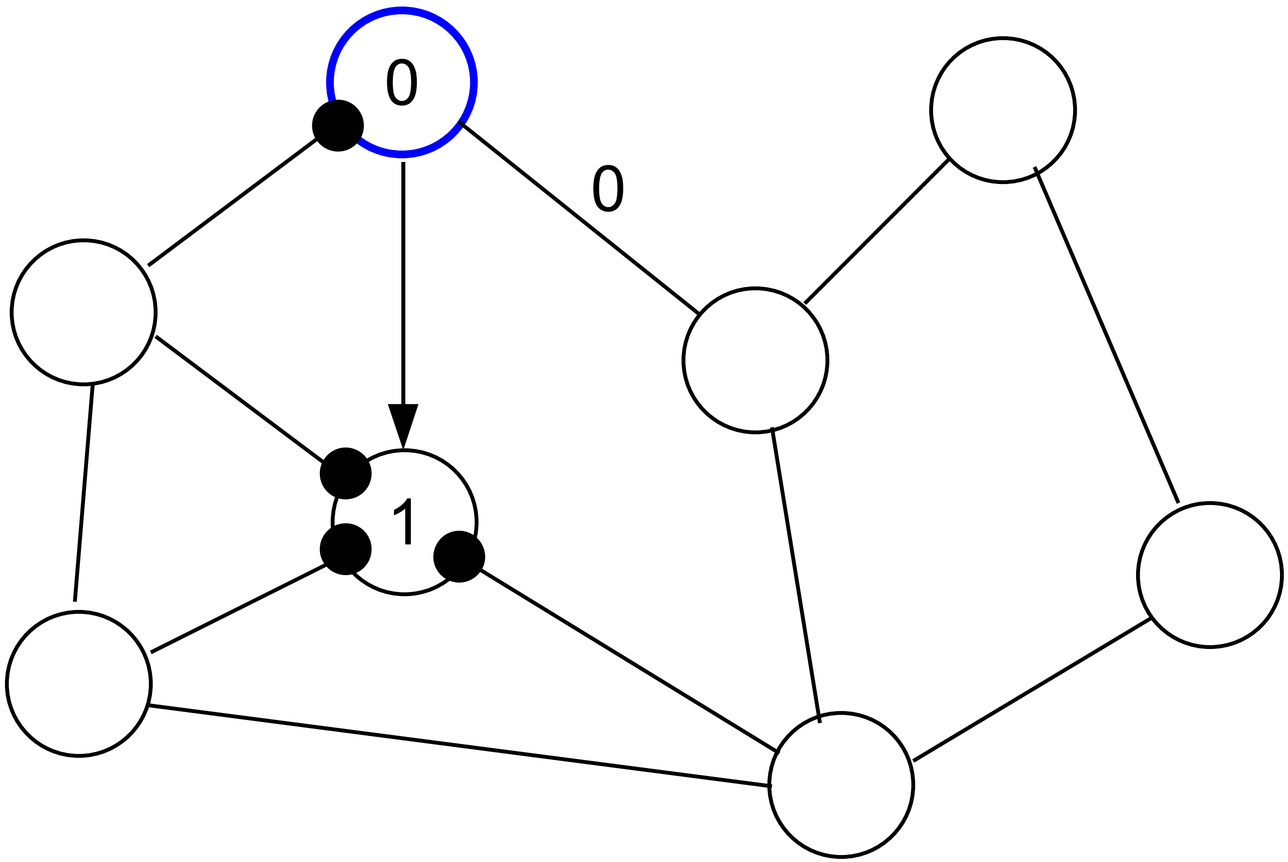
AsynchBFS



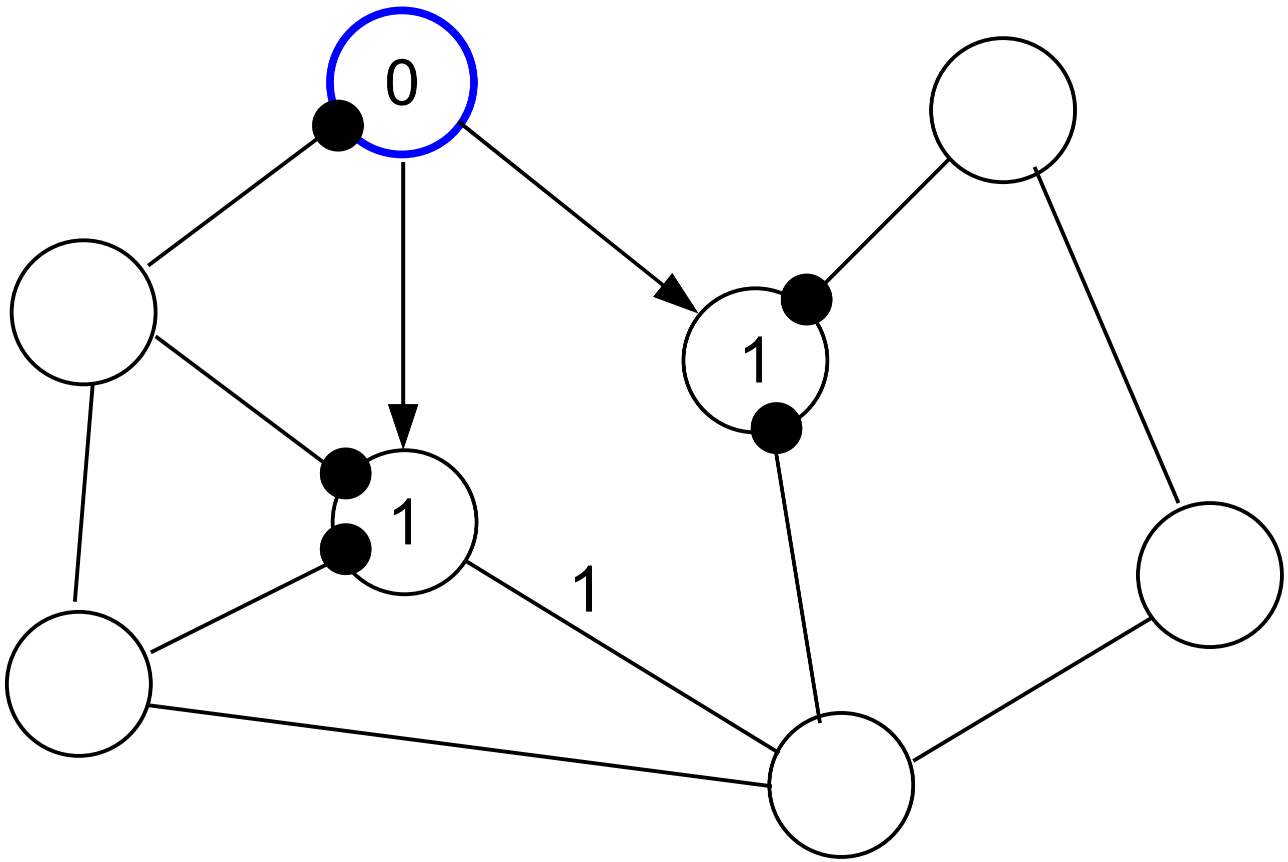
AsynchBFS



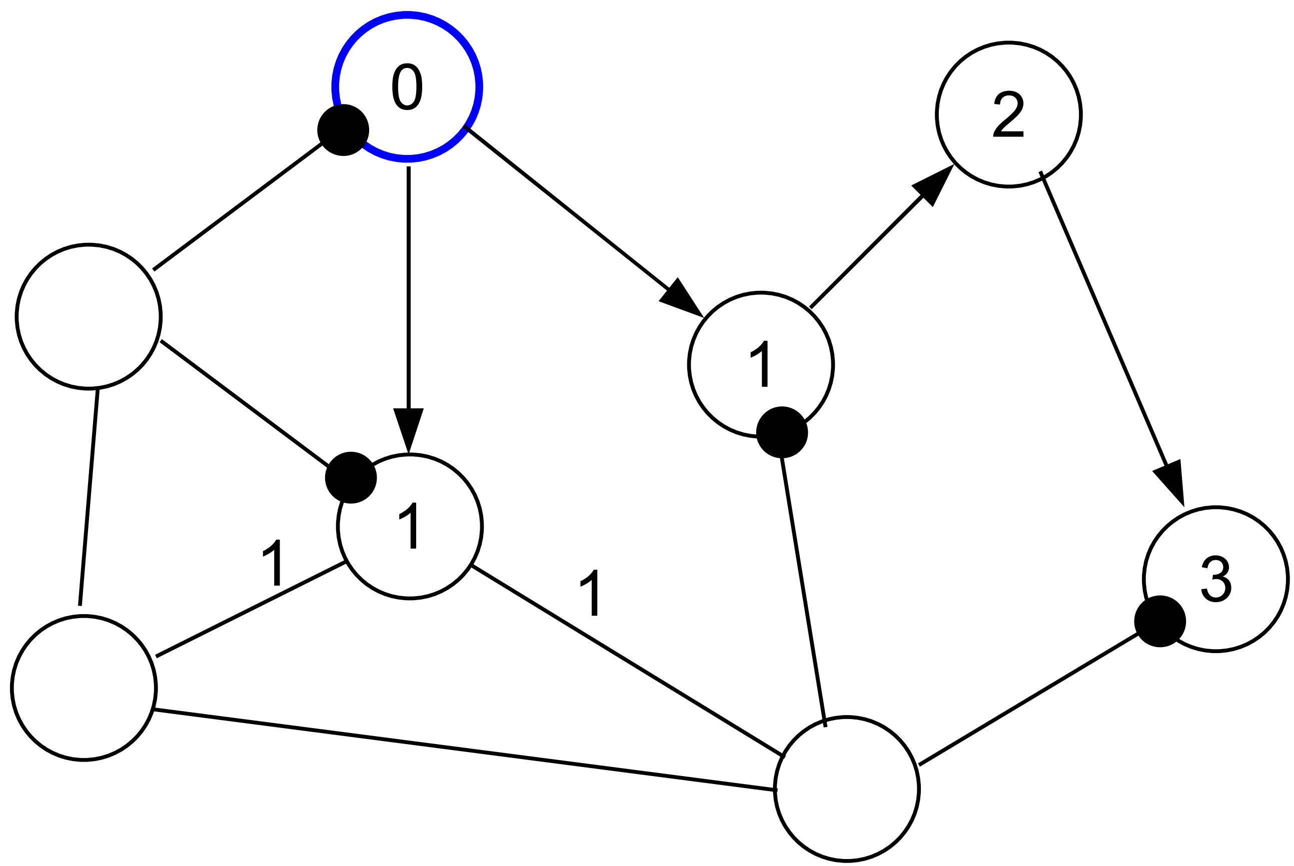
AsynchBFS



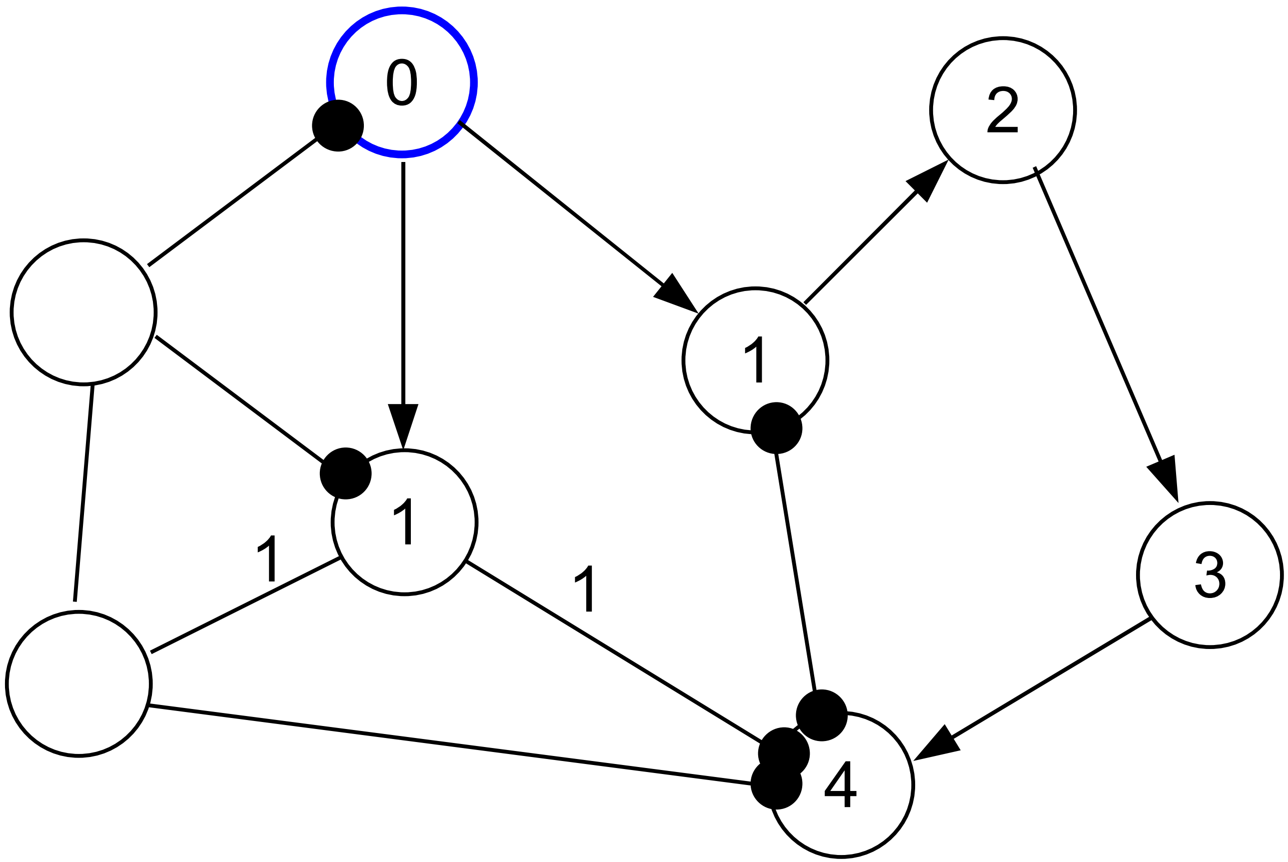
AsynchBFS



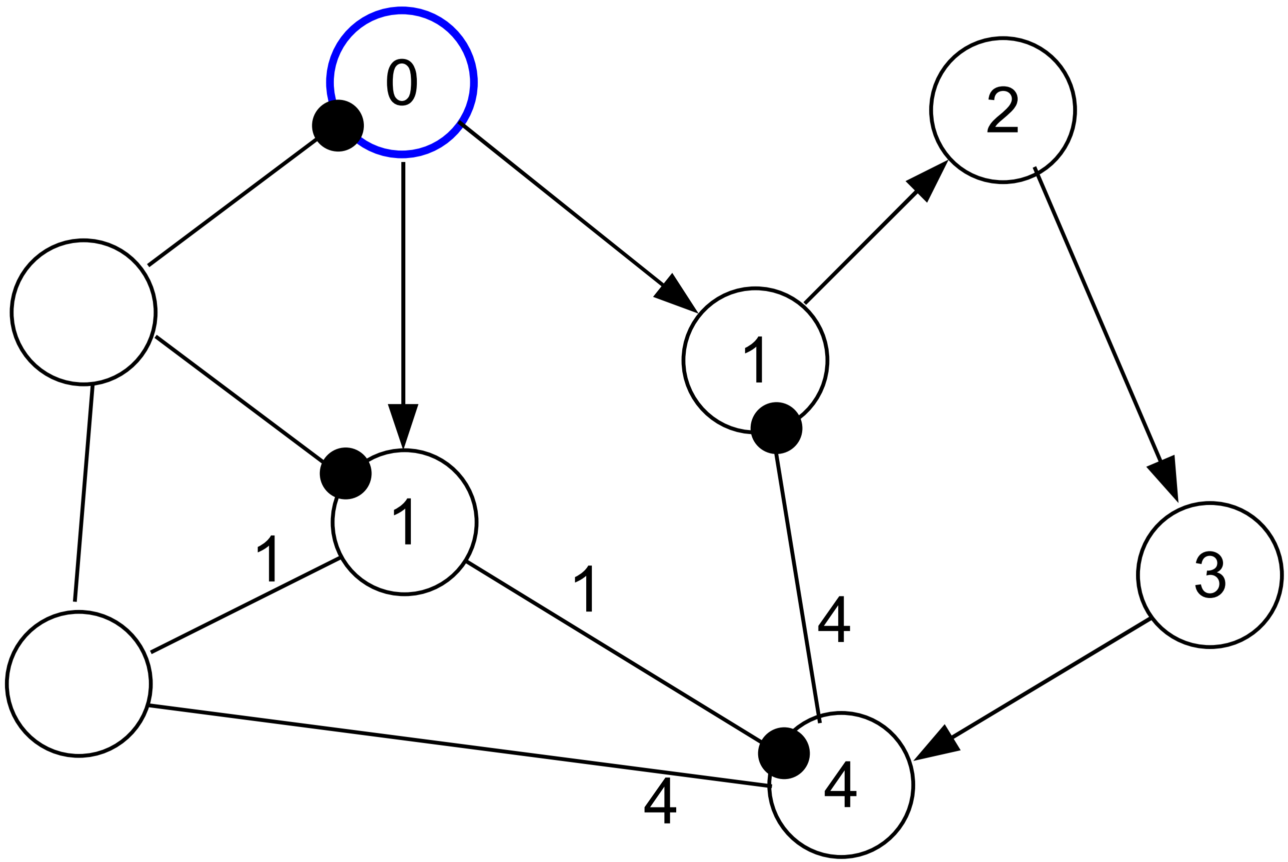
AsynchBFS



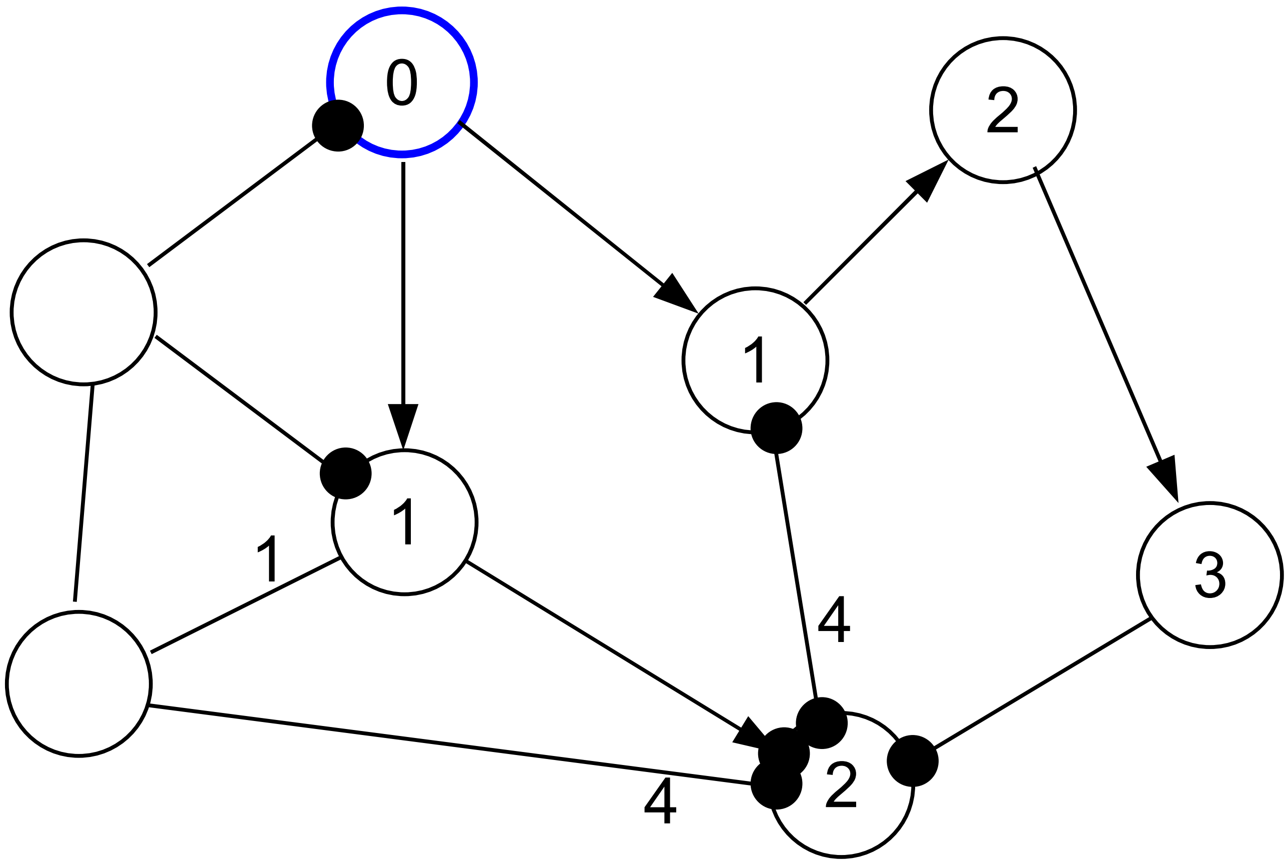
AsynchBFS



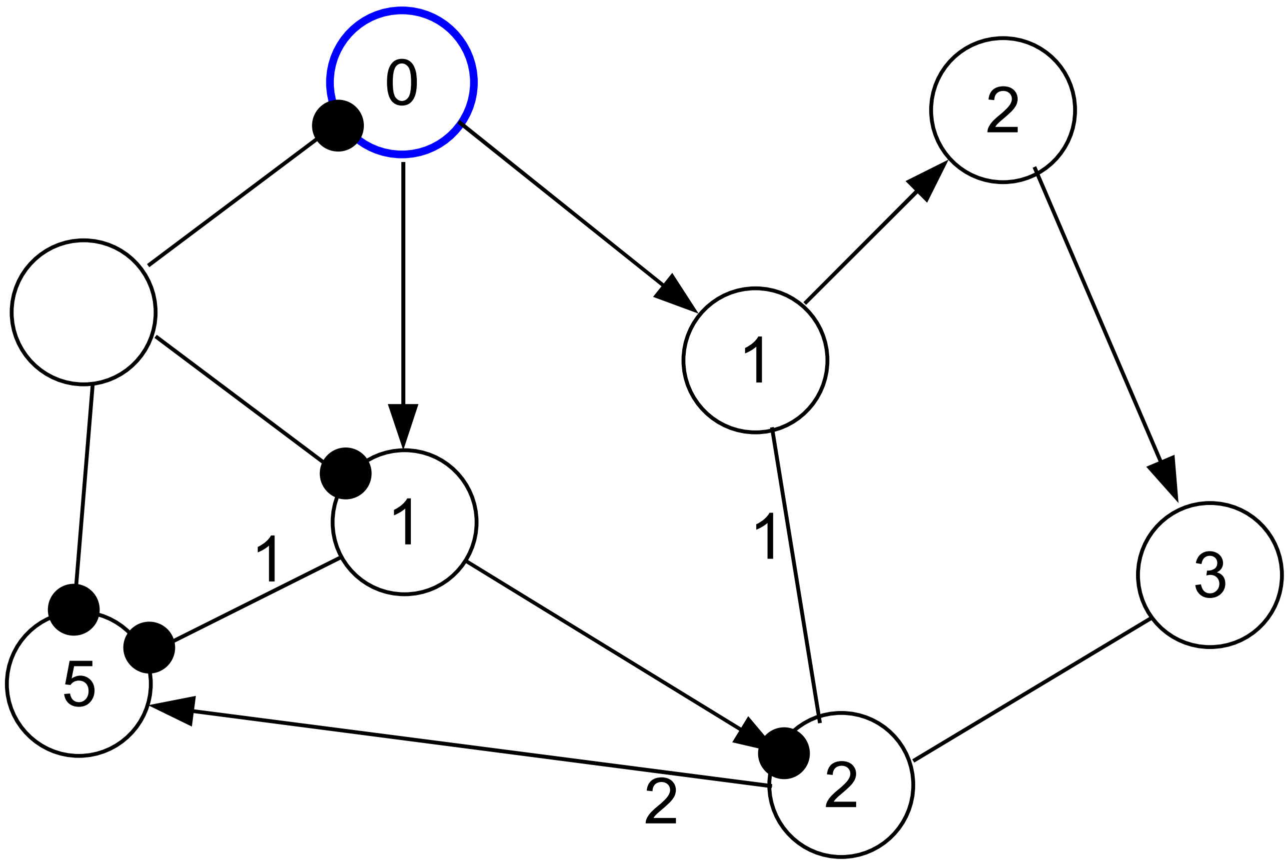
AsynchBFS



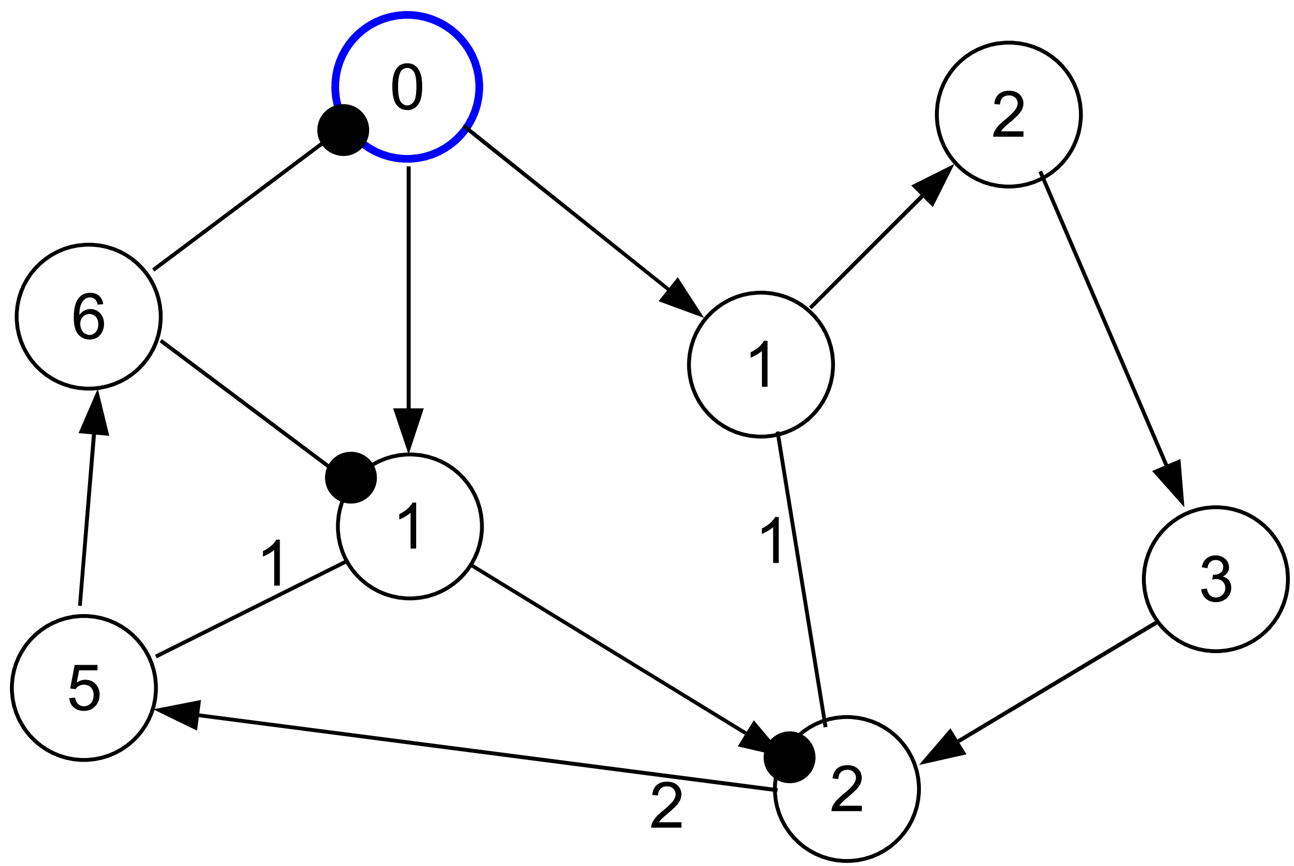
AsynchBFS



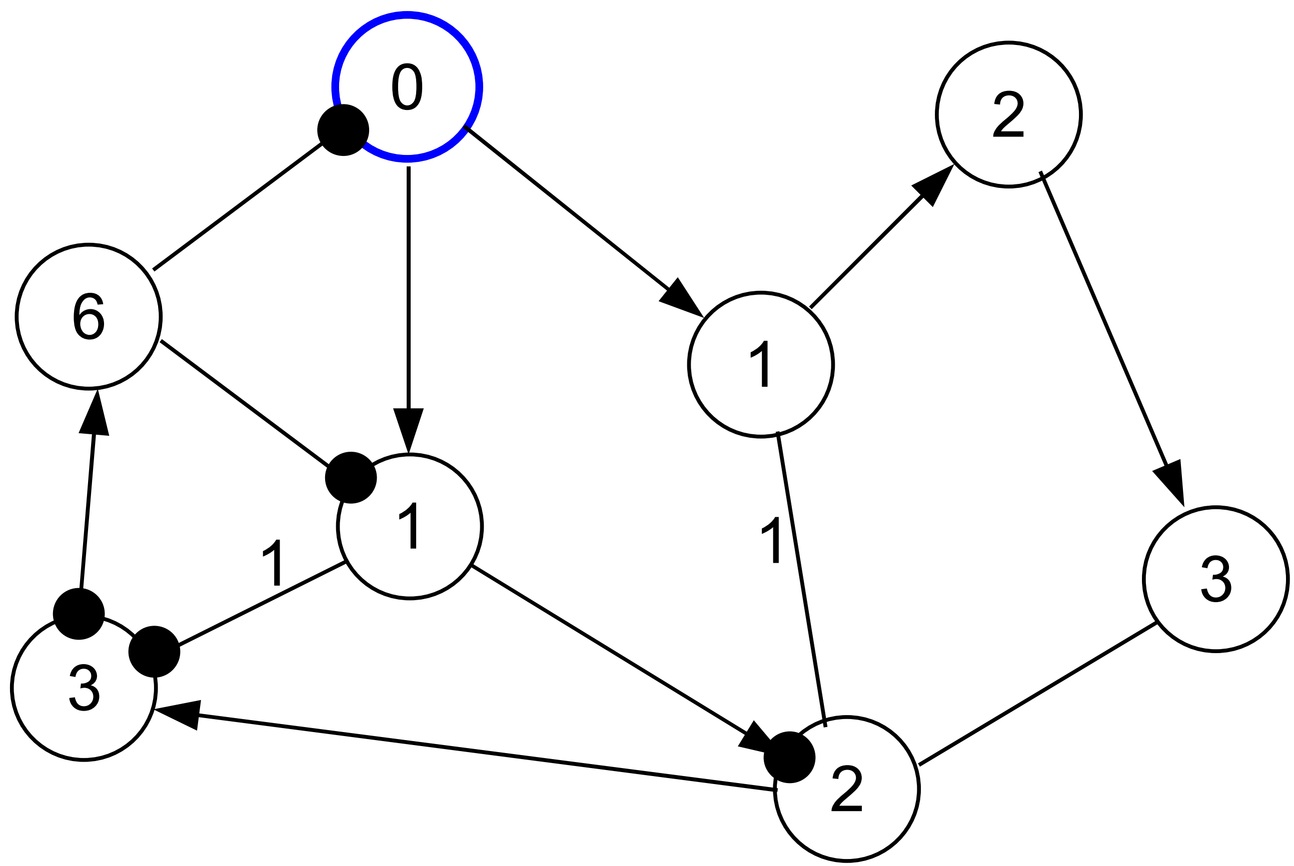
AsynchBFS



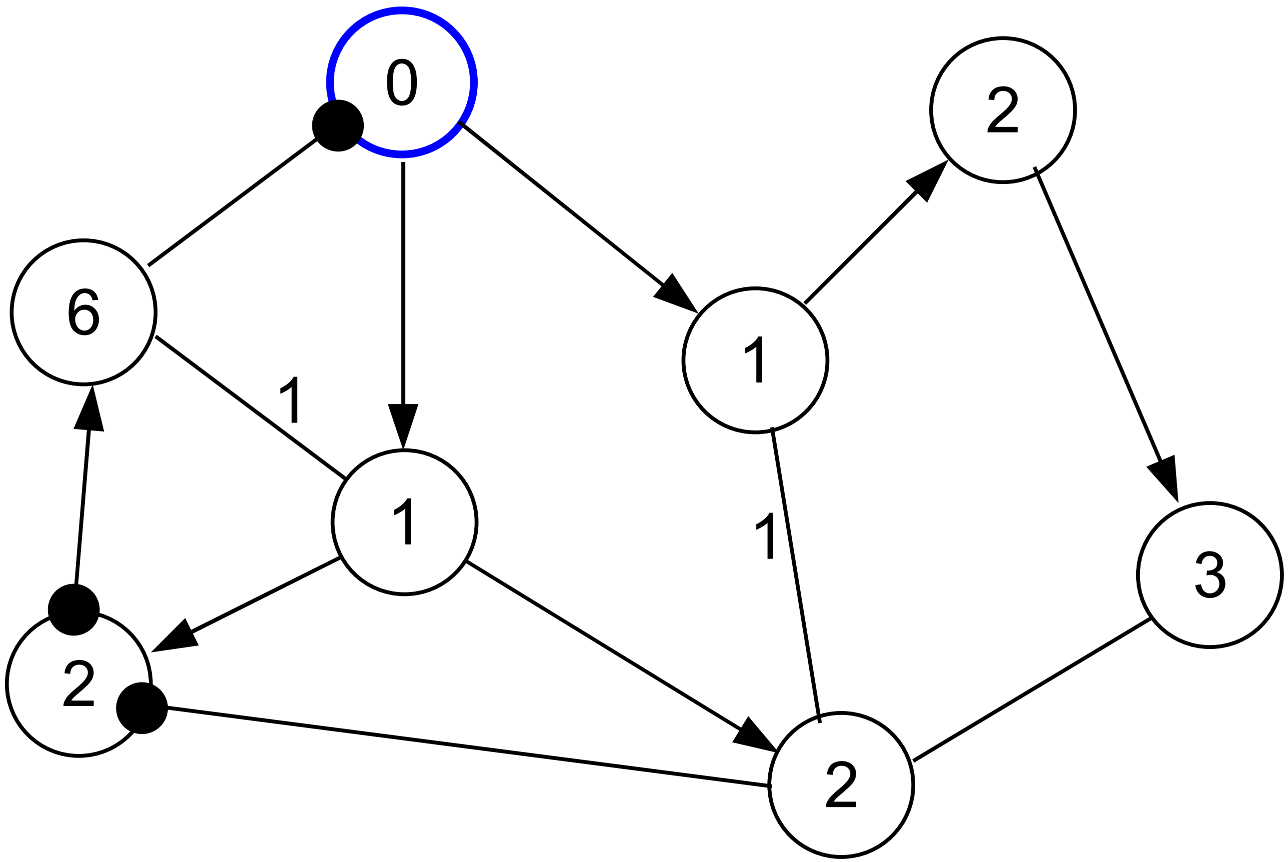
AsynchBFS



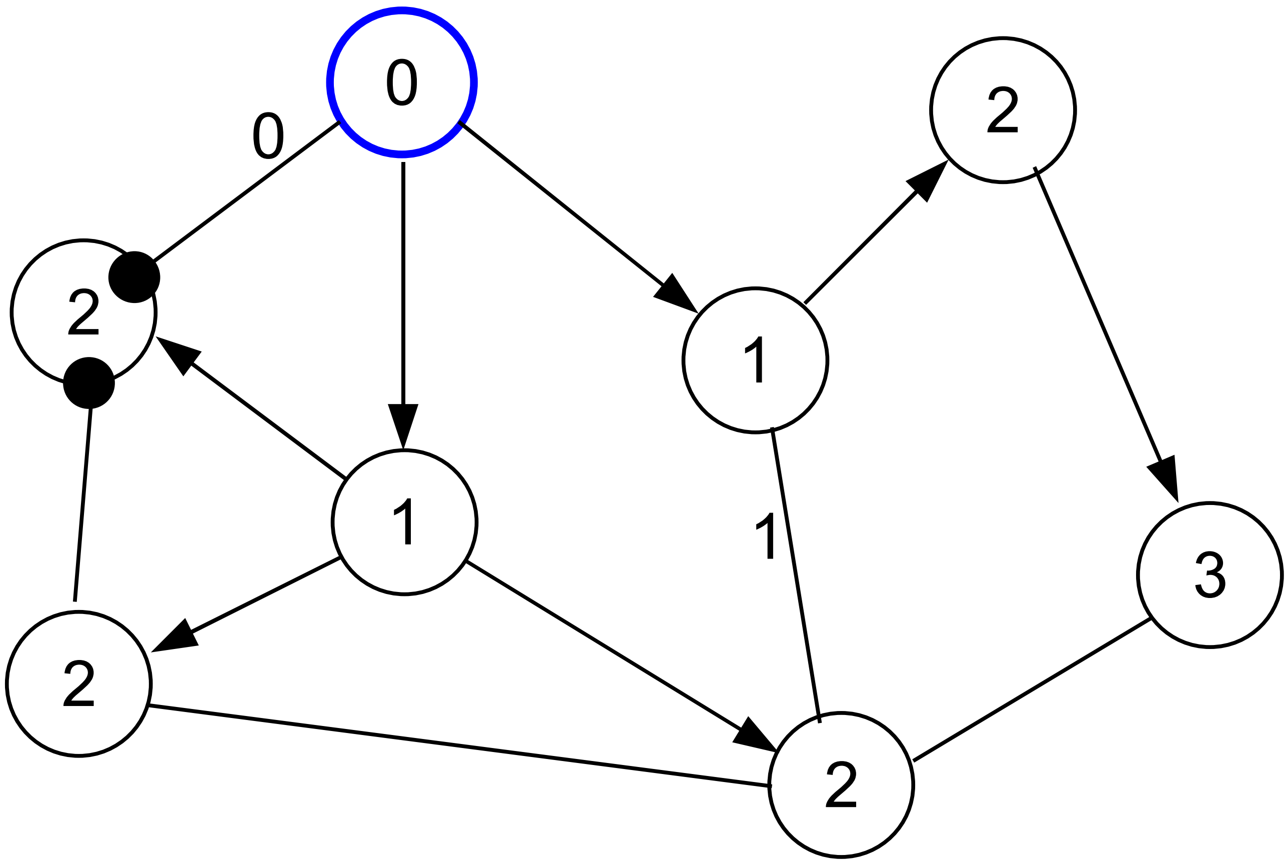
AsynchBFS



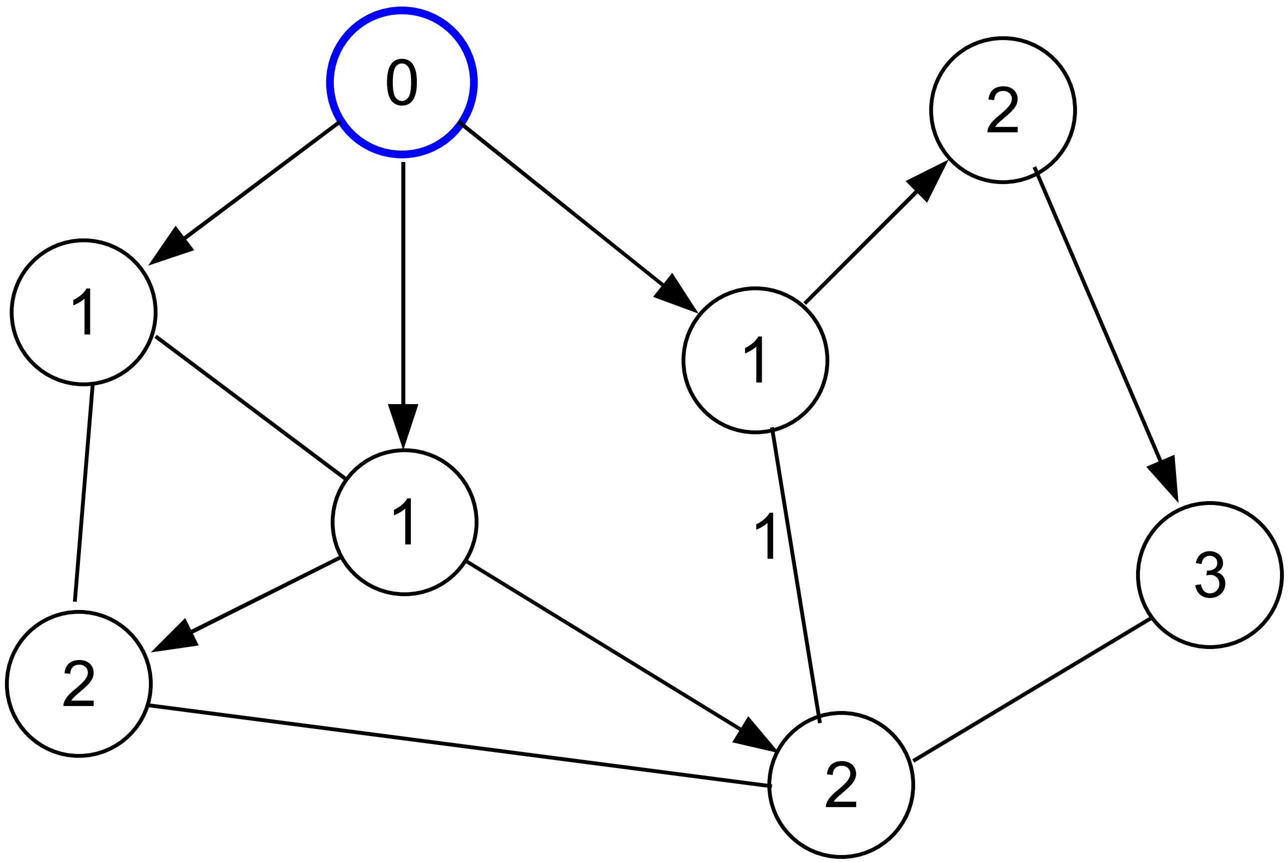
AsynchBFS



AsynchBFS



AsynchBFS



AsynchBFS

- Complexity
  - msg: O(n |E|)
    - may send O(n) msgs on each link (one for each distance estimate)
  - time: O(diam n (l+d))
- Reduce complexity if know bound D on diameter
  - msg: O(D |E|); time: O(diam D (l+d))
- To determine parents, use convergecast
  - ack receipt after "children" ack receipt (of forwarded message)
    - may have several messages "pending" (i.e., awaiting acks)
  - when i_0 gets ack from all its neighbors, everyone is done (why?)
  - complexity?

Layered BFS

- Run in phases
  - in phase k,find nodes at depth k
  - start by i_0 sending newphase (broadcast along tree)
  - end by convergecast (from nodes at depth k)
- Complexity
  - msg: O(|E| + n diam)
  - time: $O(diam^2\ (l+d))$

LayeredBFS vs AsynchBFS

- Alternative timing assumption
  - local computation negligible
  - Δ is bound on time from send to receive (i.e., no “pile-up”)
- Message complexity
  - AsynchBFS: O(D |E|)
  - LayeredBFS: O(|E| + n diam)
- Time complexity
  - AsynchBFS: O(diam Δ)
  - LayeredBFS: O(diam^2 Δ)
- Can make “hybrid” algorithm (in book)
  - add m layers in each phase

Shortest paths

- Same assumptions as before, but add edge weights
  - use weight function: weight(i,j)
  - assume nonnegative weights; same in each direction
- Output shortest distance and parent
- Use Bellman-Ford asynchronously
  - used to establish routes in ARPANET 1969-1980
  - where was synchrony used? what other assumptions?

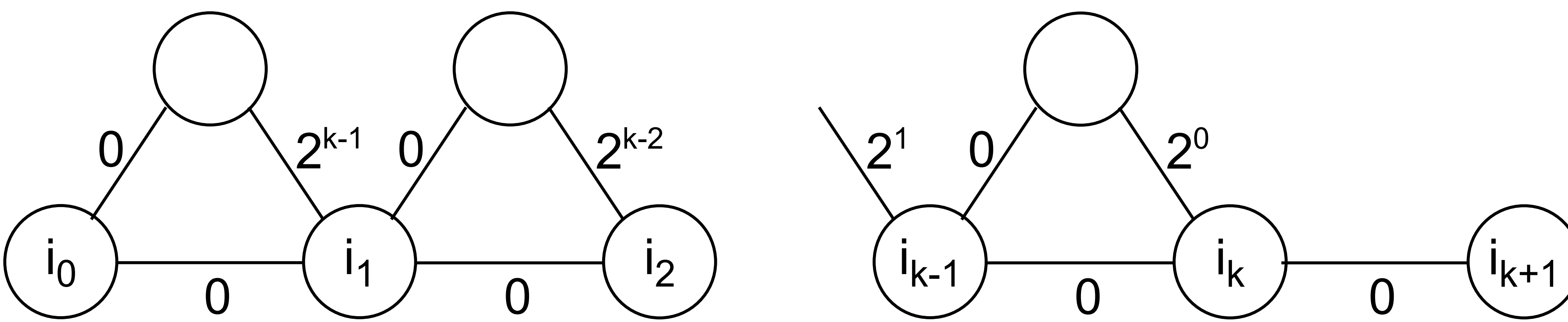
AsynchBellmanFord

- Signature
  - ***in*** receive$(w)_{j,i}$, $m \in \mathbf{R}^{\geq 0}$**,** $j \in$ nbrs
  - ***out*** send$(w)_{i,j}$, $m \in \mathbf{R}^{\geq 0}$**,** $j \in$ nbrs
- State
  - **dist**: $\mathbf{R}^{\geq 0}$ U { ∞ };
    init 0 if i = i_0, else ∞
  - **parent**: nbrs U { null }
  - for each $j \in$ nbrs
    - **send**(j): FIFO queue of $\mathbf{R}^{\geq 0}$;
      init { 0 } if i = i_0, else ∅

- send$(w)_{i,j}$
  pre: m is head of **send**(j)
  eff: remove head of **send**(j)
- receive$(w)_{j,i}$
  eff: if w+weight(j,i) < **dist** then
  dist := w+weight(j,i)
  parent := j
  for $k \in$ nbrs - { j } do
  add **dist** to **send**(k)

AsynchBellmanFord

- Termination
  - use broadcast/convergecast (as in AsynchBFS)
- Complexity
  - (n-1)! simple paths from i0,
    so msg complexity = O((n-1)! |E|), time complexity = O(n!(l+d))
    - time complexity = O(n Δ)?
  - in some exec of network below, i_k sends 2^k messages to i_{k+1},
    so msg complexity is $\Omega(2^{n/2})$ and time complexity is $\Omega(2^{n/2}$ d)



Minimum spanning tree

- Assumptions as before, and edge weights distinct
- Problem:
  Input: wakeup actions, asynchronous at one or more nodes
- Gallager-Humblet-Spira algorithm
  - read this paper!
  - recall synchronous variant
    - grow tree in levels

Next lecture

- Gallager-Humblet-Spira algorithm (Chapter 15.5)
- Synchronizers (Chapter 16)