6.852: Distributed Algorithms
Fall, 2015

Lecture 3

Today's plan

- Algorithms in general synchronous networks, cont'd:
  - Shortest paths spanning trees
  - Minimum-weight spanning trees
  - Maximal independent Sets (MIS)
- Reading: Sections 4.3-4.5
  - [Gallager, Humblet, Spira]
  - [Luby]
  - [Metivier, Robson,…]

- Next time: (Stephan Holzer):
  - Maximal Independent Sets, cont'd
  - Graph coloring
  - Reading: Peleg, Chapters 7 and 8; related papers

Last time

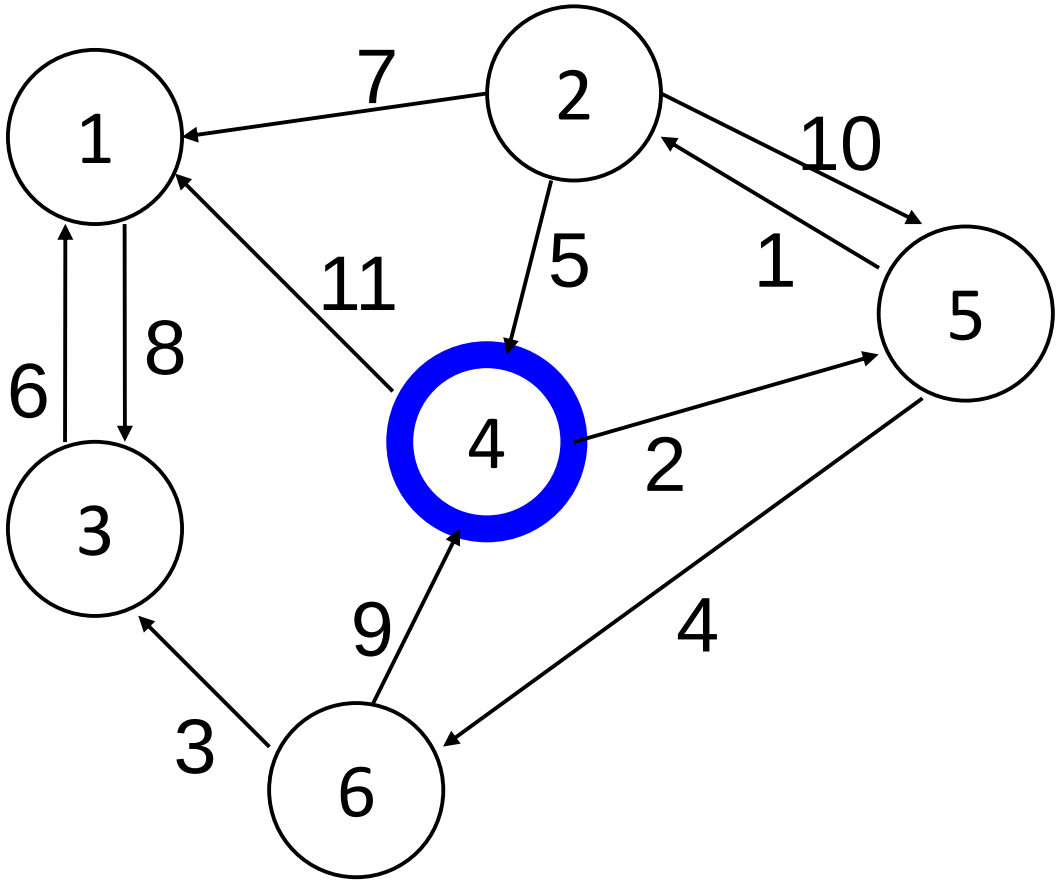
- $\Omega(n \log n)$ lower bound on the number of messages for comparison-based leader election in a ring.
- Leader election in general synchronous networks:
  - Flooding algorithm
  - Improved algorithm, with smaller message complexity
  - Simulation relation proof
- Breadth-first search in general synchronous networks:
  - Simple marking algorithm
  - Applications:
    - Broadcast, convergecast
    - Data aggregation (computation in networks)
    - Leader election in unknown networks
    - Determining the diameter

Shortest Paths

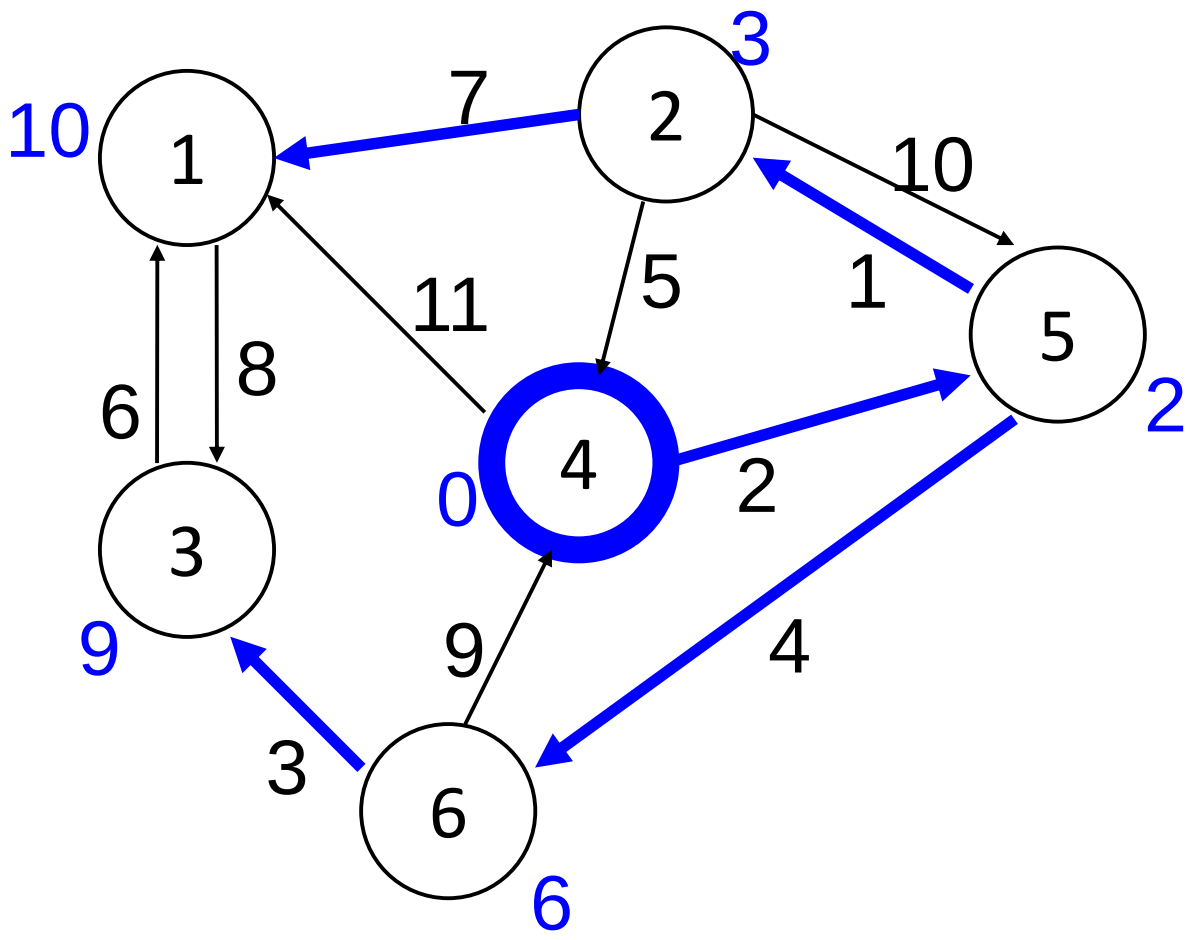
Shortest paths

- Motivation: Establish a structure for efficient communication.
  - Generalization of Breadth-First Search.
  - Now edges have associated costs (weights), w_{ij} for edge (i, j).
- Assume:
  - Strongly connected digraph, root i_0.
  - Weights (nonnegative reals) on edges.
    - Weights represent some type of communication cost, e.g. latency.
  - UIDs.
  - Nodes know weights of incident edges.
  - Nodes know n (use this just for termination).
- Required:
  - Shortest-paths tree, giving shortest path from i_0 to every other node.
  - Shortest path = path with minimum total weight.
  - Each node should output:
    - Its weighted distance from i_0, and
    - Its parent on a shortest path from i_0.

Shortest paths



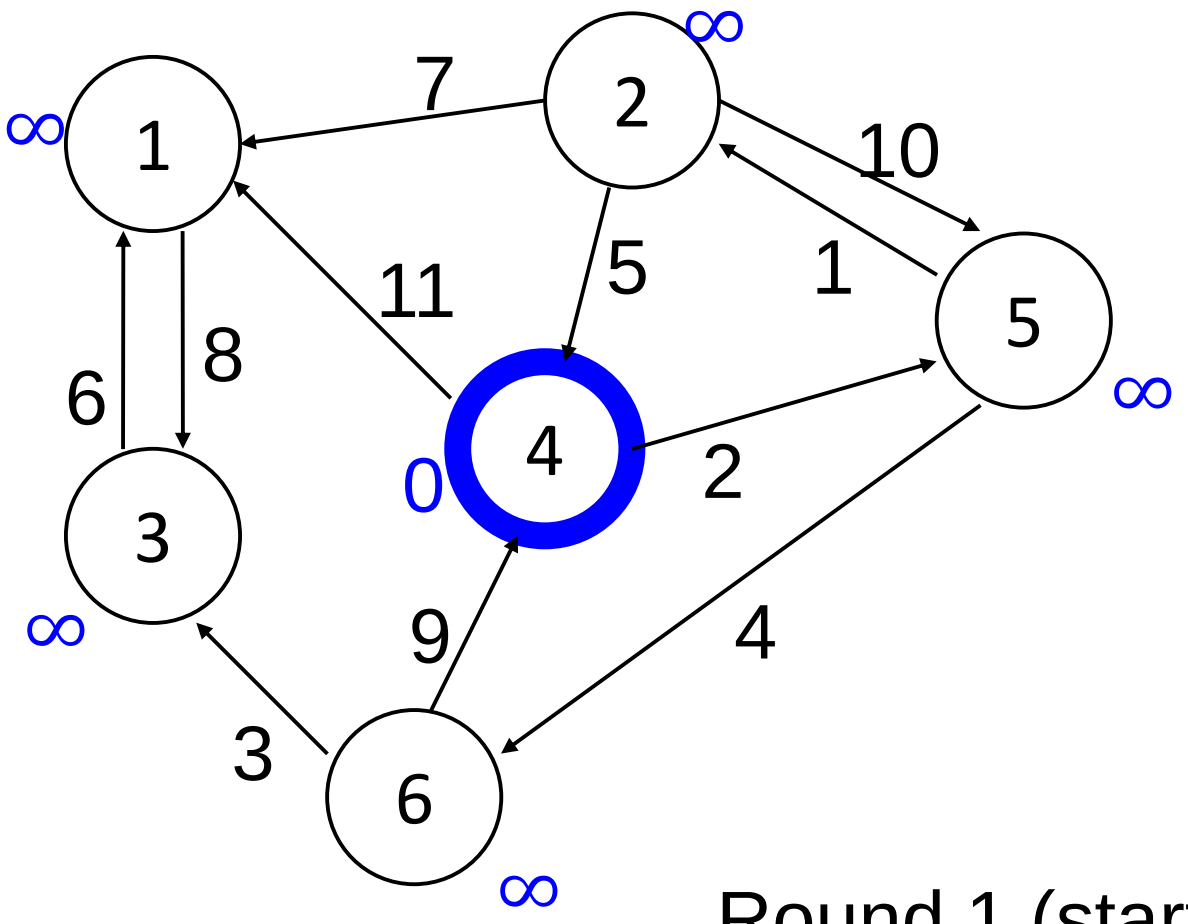
Shortest paths



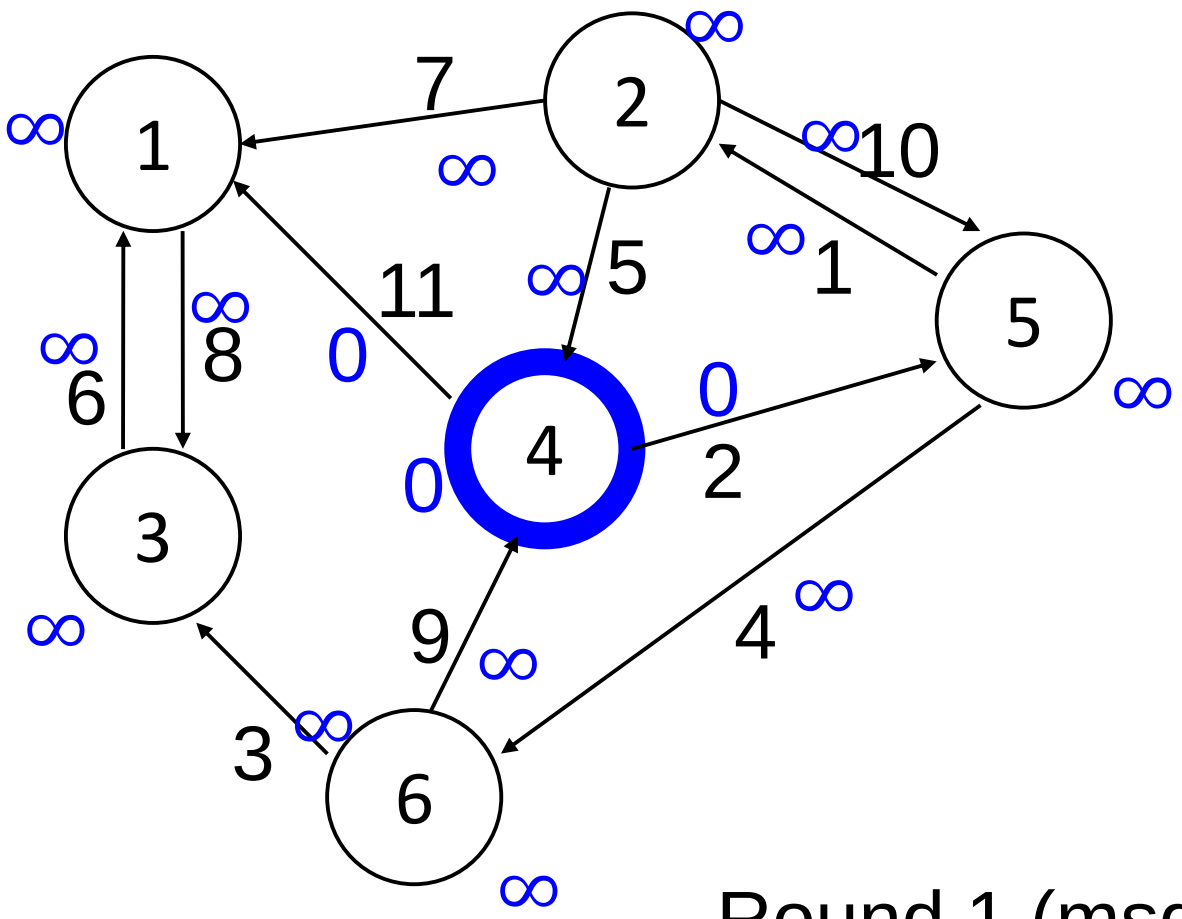
Shortest paths algorithm

- Bellman-Ford (adapted from sequential Bellman-Ford algorithm)
- Each process maintains:
  - $dist$, shortest distance it knows about so far, from i_0
  - $parent$, its parent in some path with total weight $=$ $dist$
  - $round$
- Initially:
  - i_0 has $dist = 0$, all others have $dist = \infty$.
  - Everyone's $parent = \perp$.
- At each round, each process:
  - Sends $dist$ to all $outnbrs$
  - Relaxation step:
    - Compute new $dist = \min(dist, min_j(dist_j + w_{ji}))$.
    - If $dist$ decreases then reset $parent$ to the corresponding $innbr$.
- Stop after $n - 1$ rounds.
- Then (claim) each process's $dist$ contains its distance from i_0, $parent$ contains the parent on a shortest path from i_0.

Shortest paths

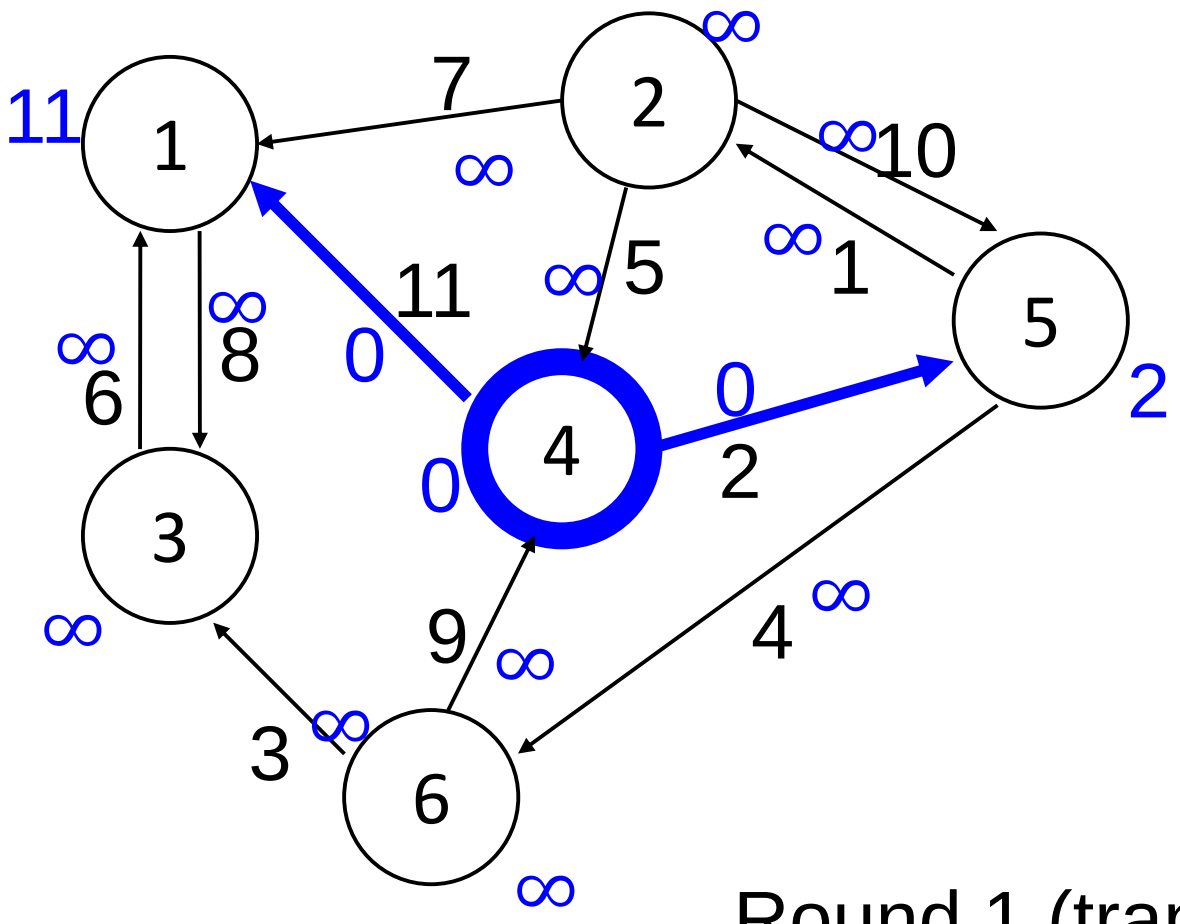


Shortest paths



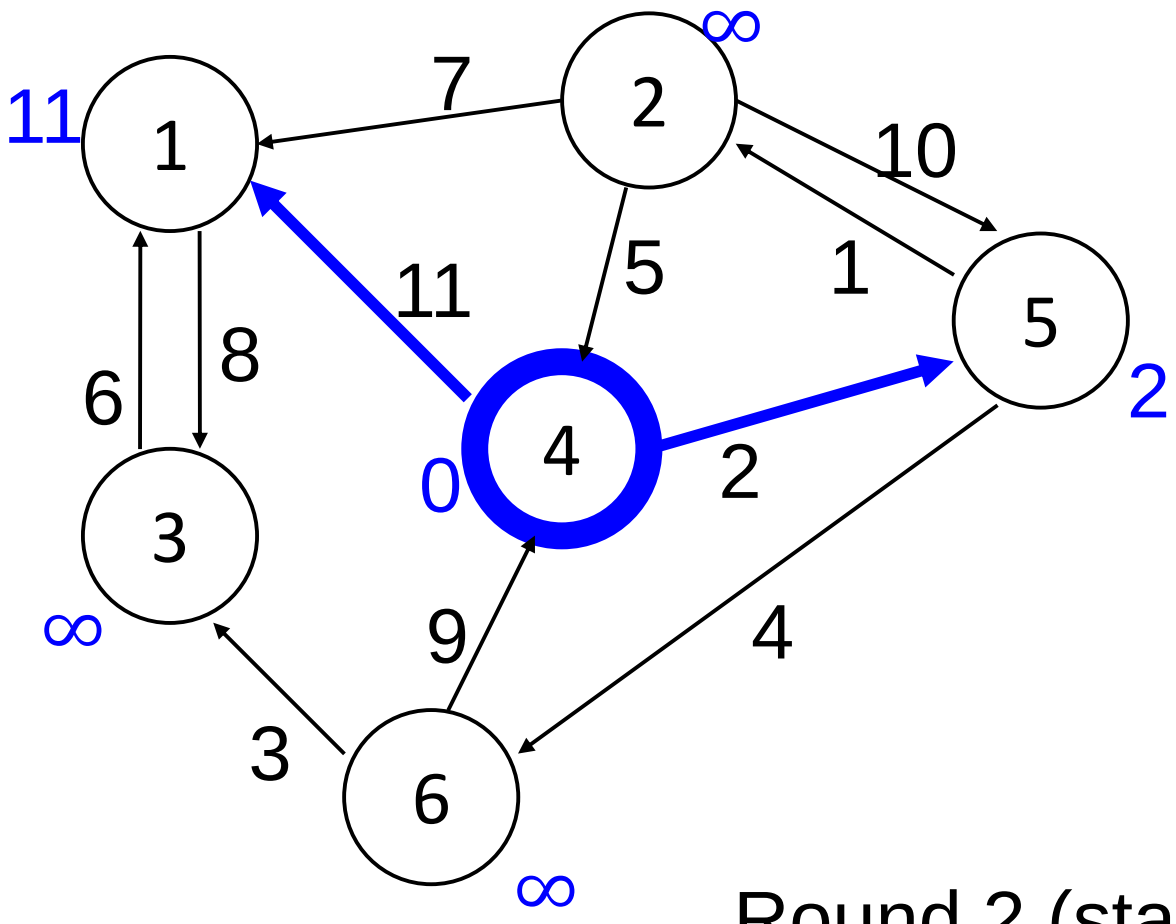
Round 1 (msgs)

Shortest paths



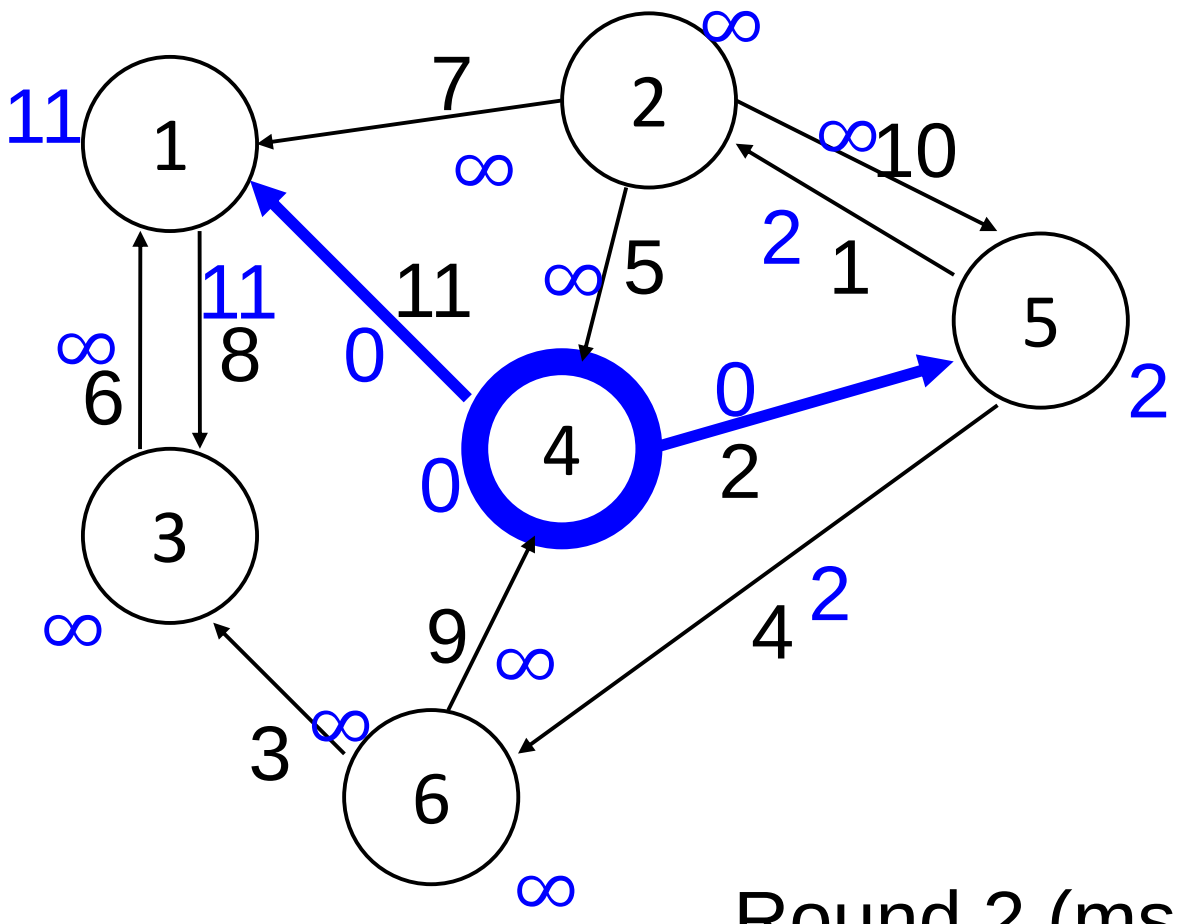
Round 1 (trans)

Shortest paths

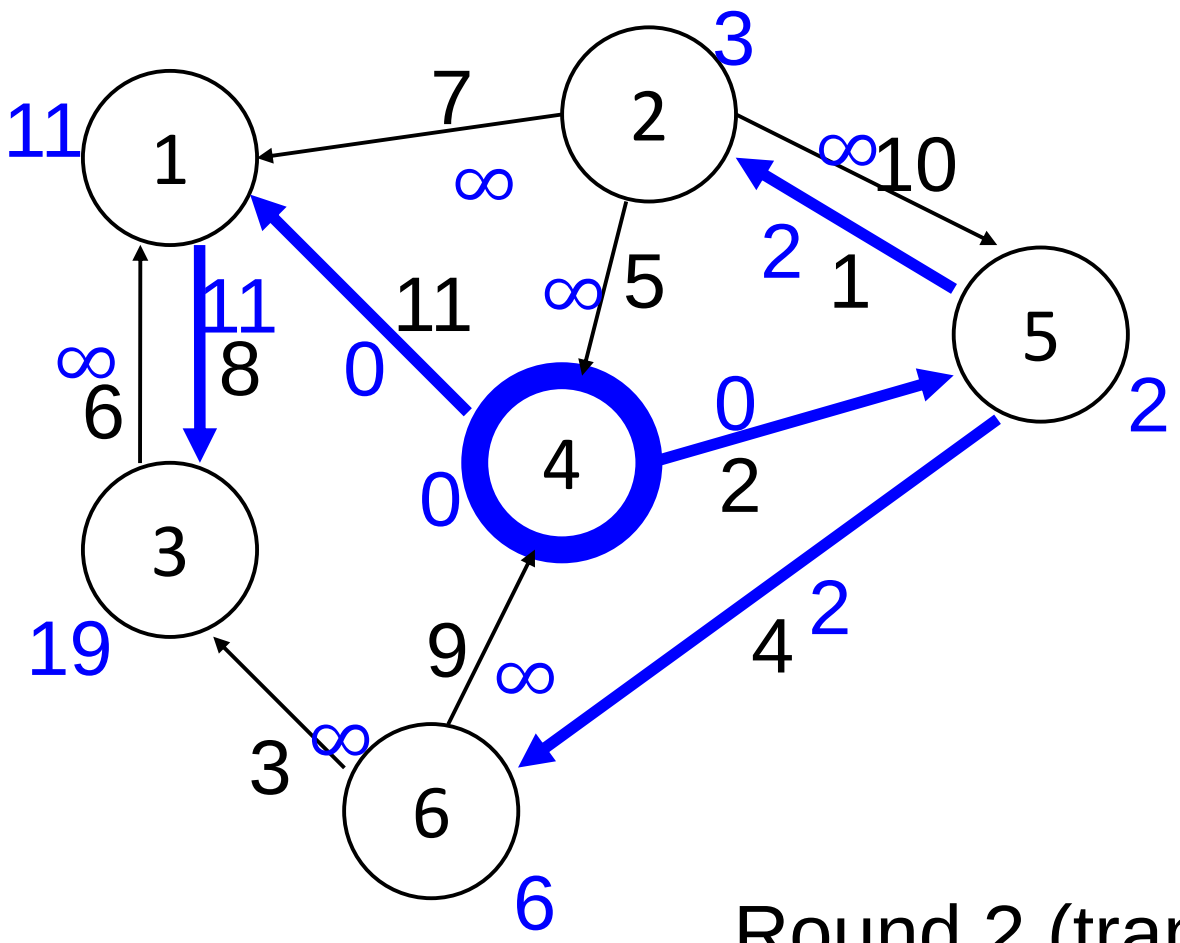


Round 2 (start)

Shortest paths

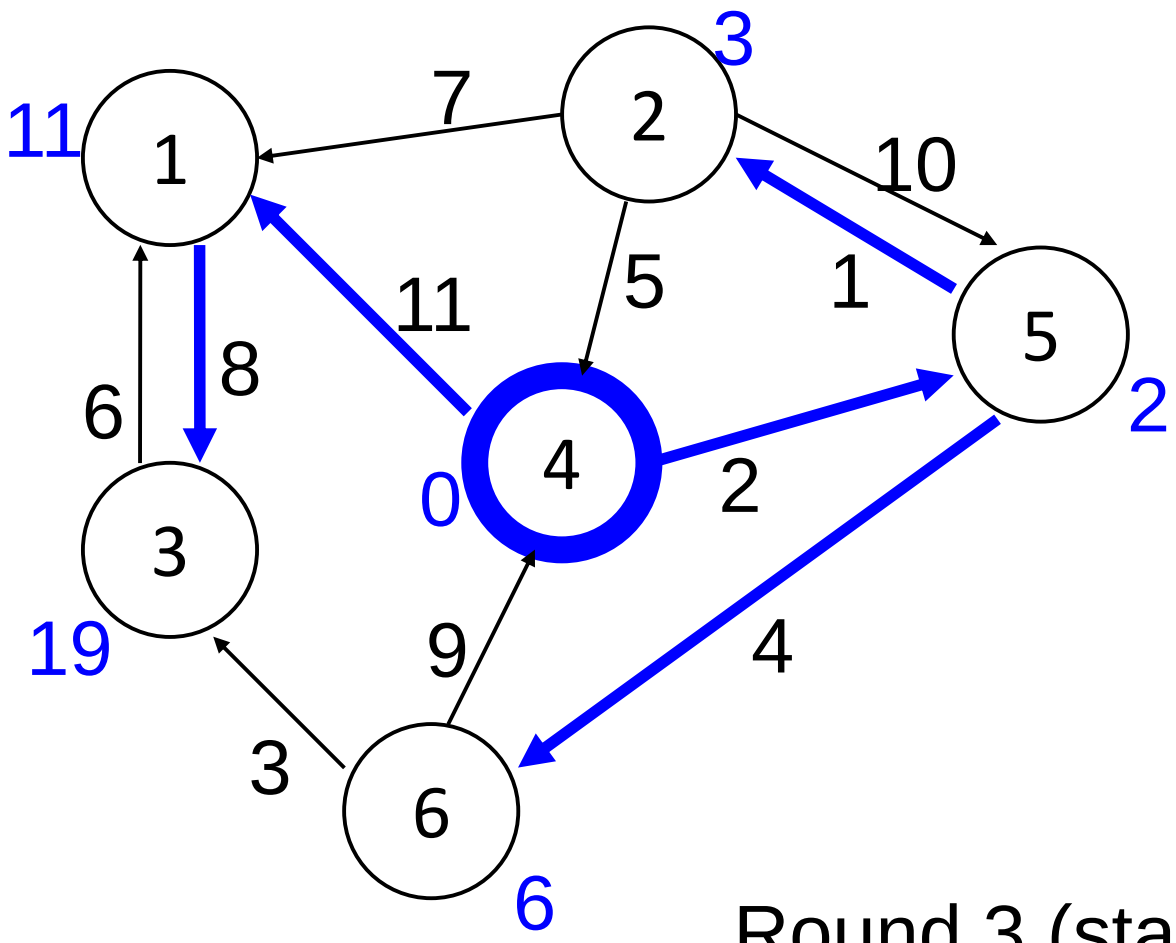


Shortest paths



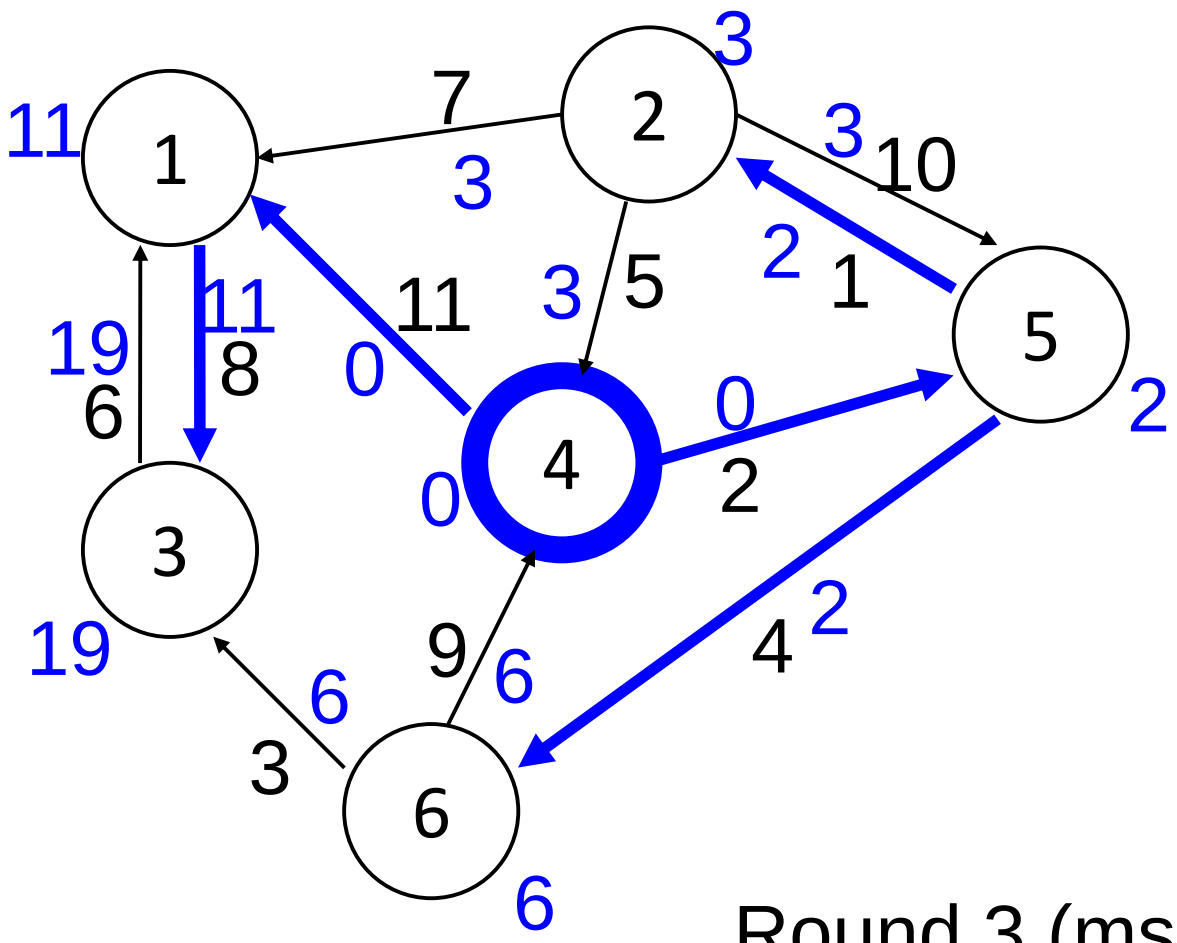
Round 2 (trans)

Shortest paths



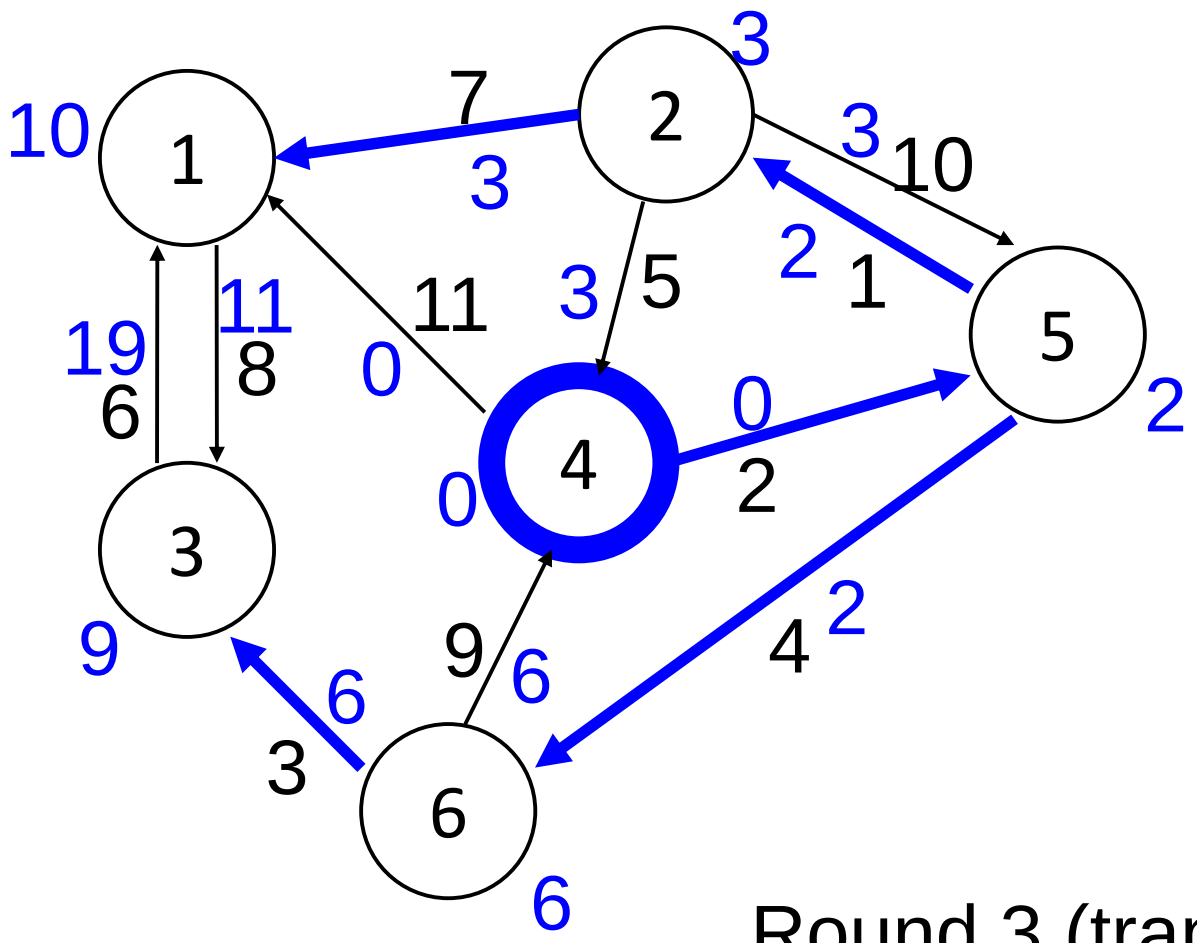
Round 3 (start)

Shortest paths



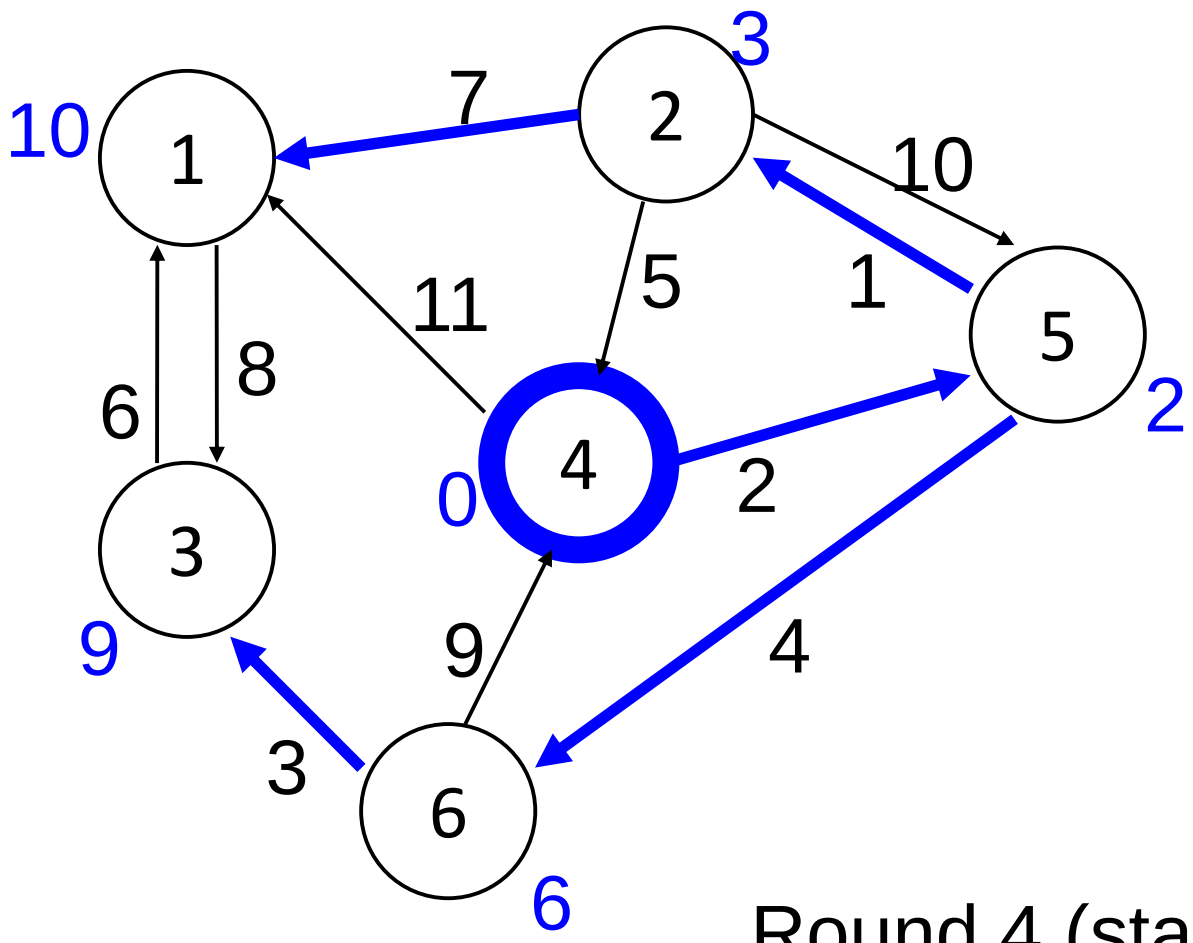
Round 3 (msgs)

Shortest paths



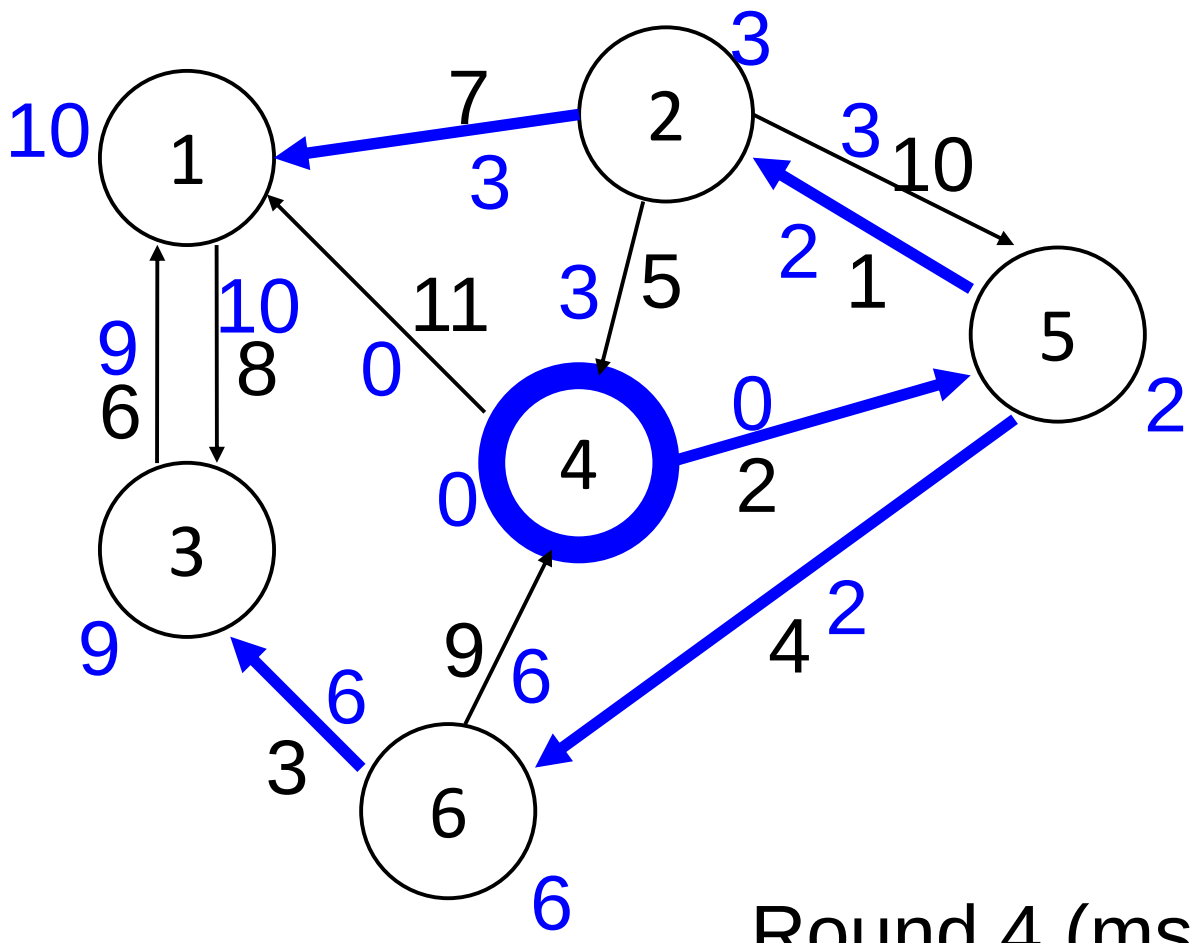
Round 3 (trans)

Shortest paths



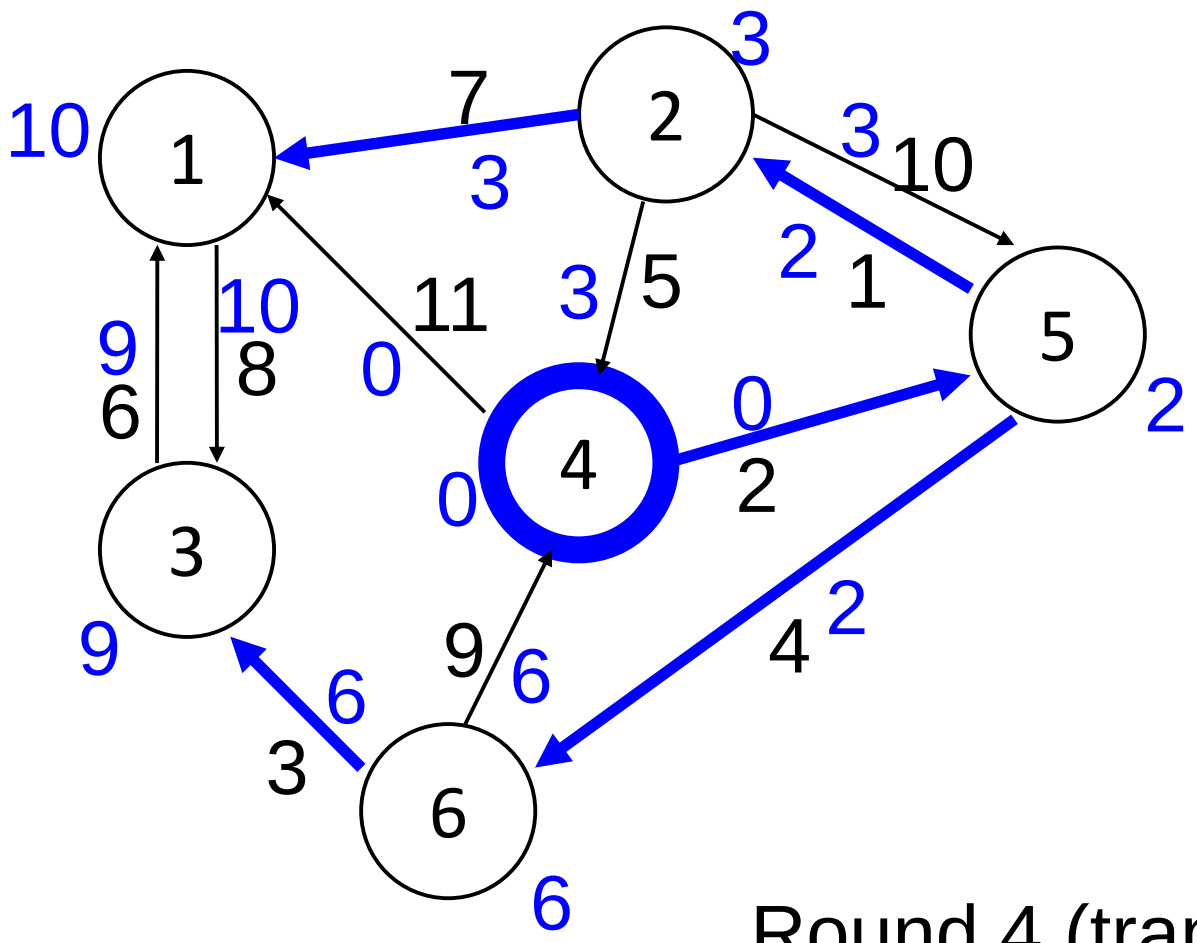
Round 4 (start)

Shortest paths



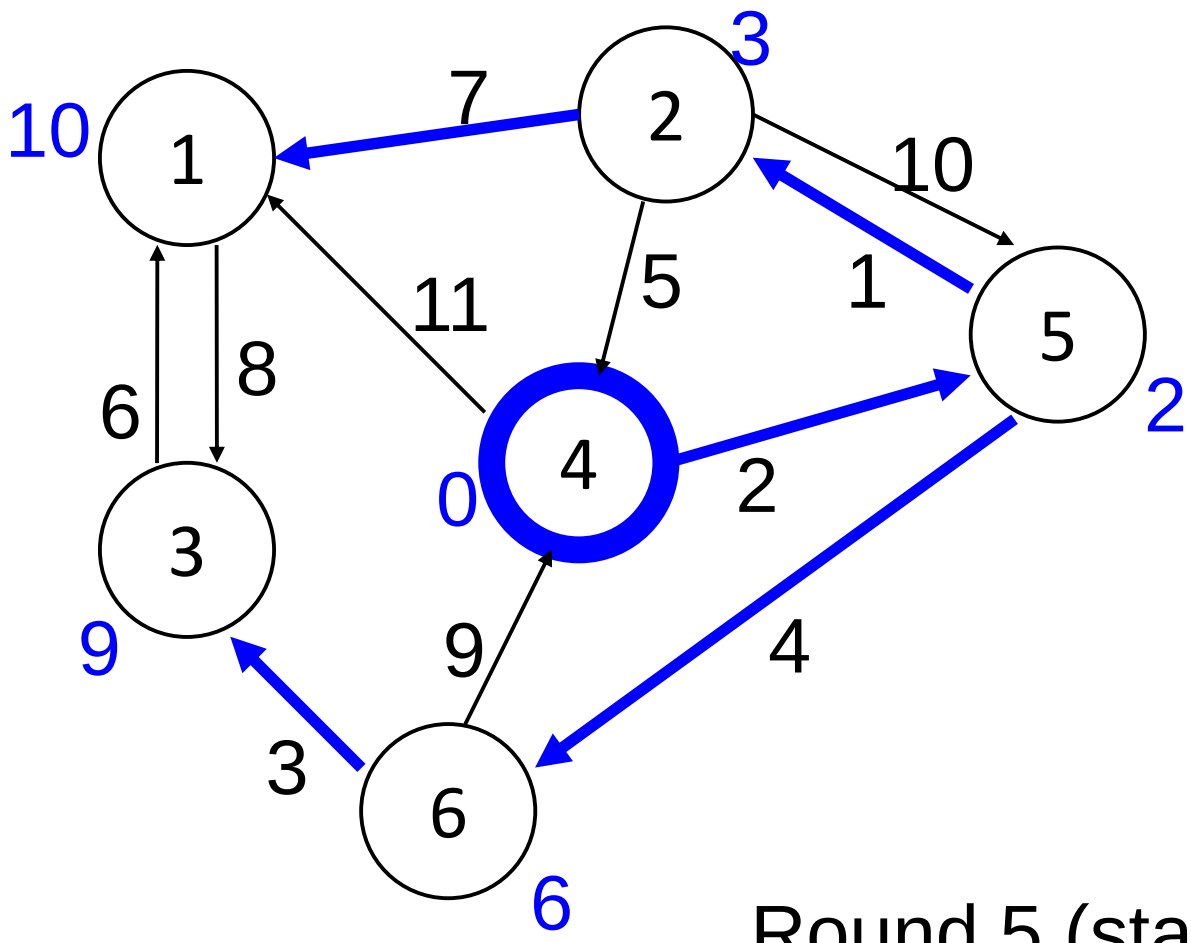
Round 4 (msgs)

Shortest paths



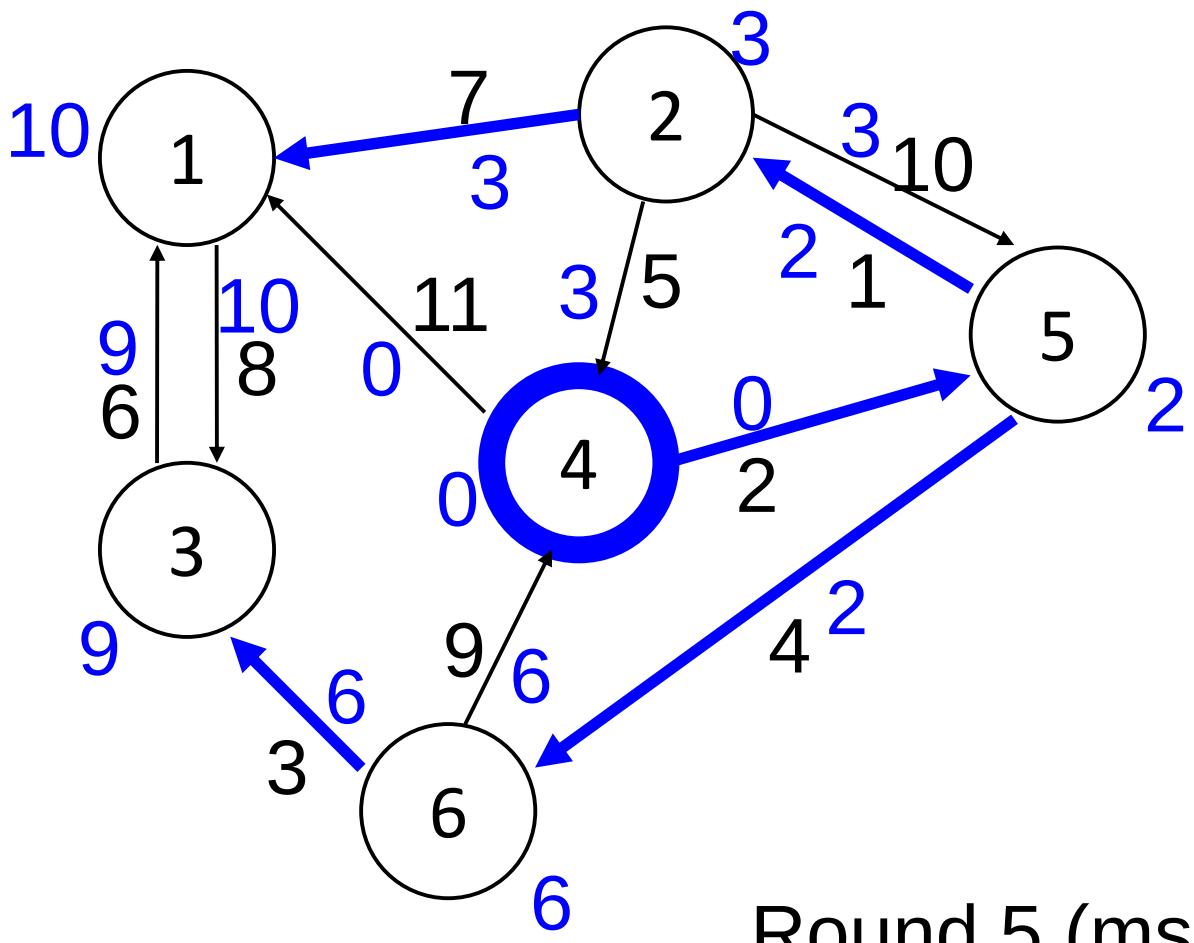
Round 4 (trans)

Shortest paths

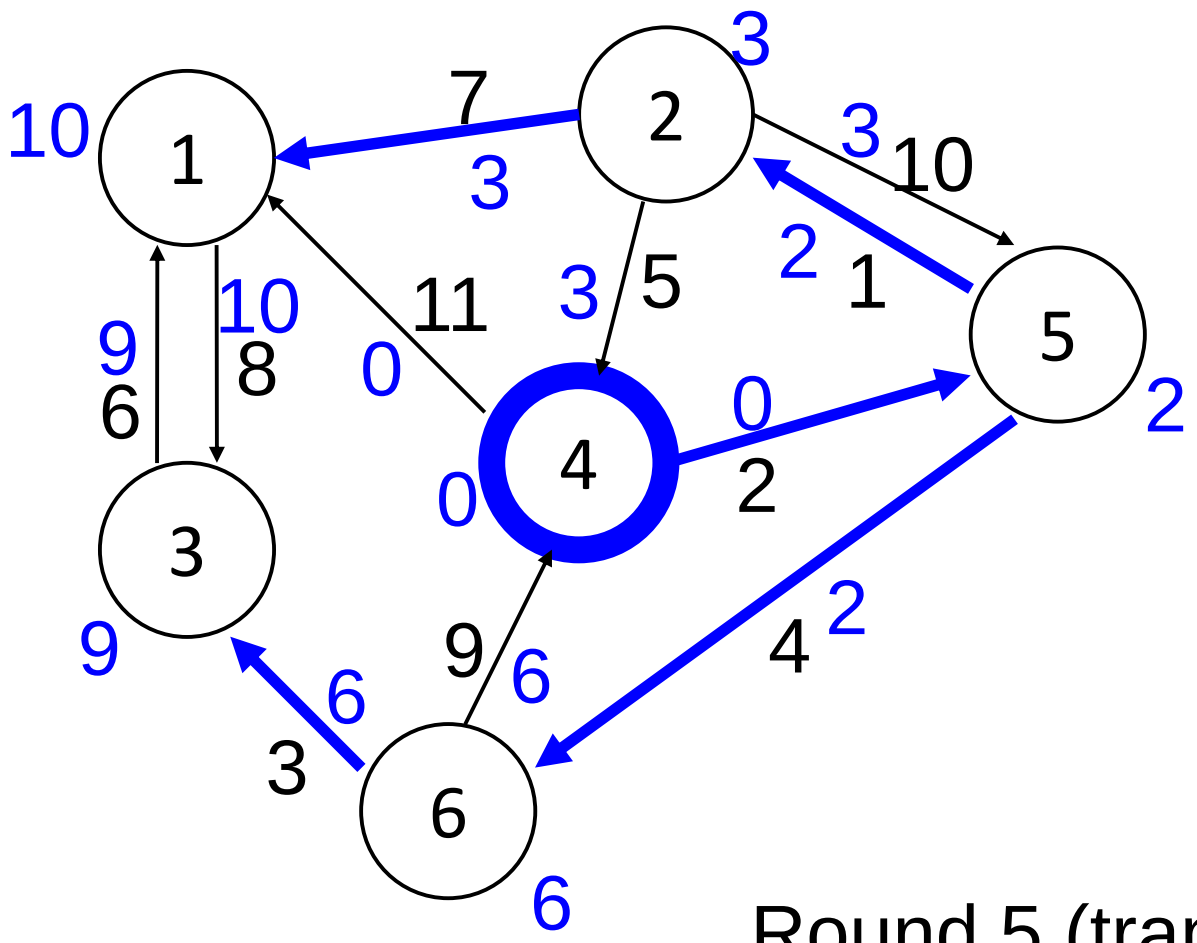


Round 5 (start)

Shortest paths

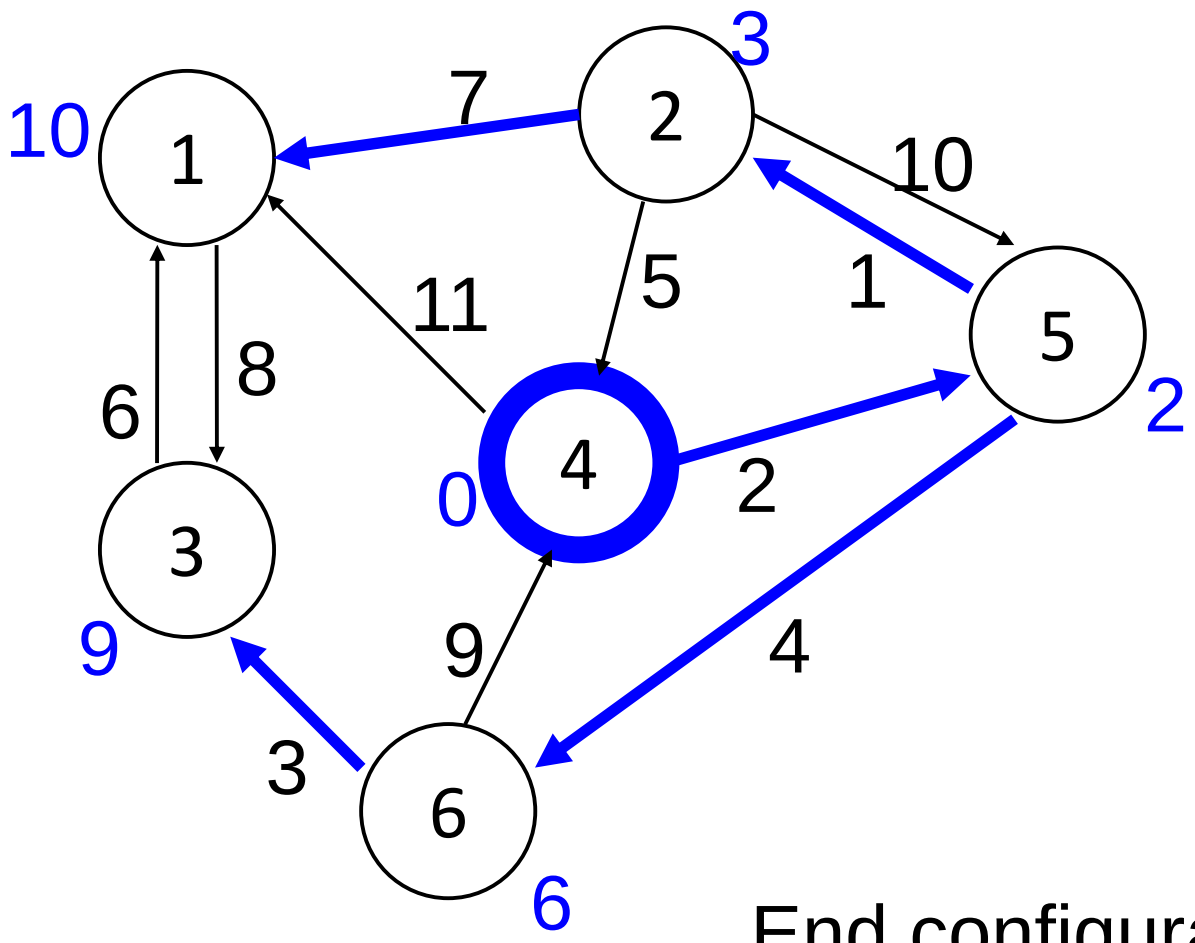


Shortest paths



Round 5 (trans)

Shortest paths



End configuration

Correctness

- Show that, just after round $n - 1$, for each process i:
  - $dist_i$ = distance from i_0 to i.
  - $parent_i$ = predecessor on a shortest path from i_0 to i.

- Proof:
  - Induction on the number r of rounds?
  - But what statement should we prove about the situation after r rounds?

Correctness

- Key invariant: After r rounds:
  - Every process i has its $dist$ and $parent$ corresponding to a shortest path from i_0 to i among those paths that consist of at most r hops (edges).
  - If there is no such path, then $dist = \infty$ and $parent = \perp$.
- Follows from two claims:
- Claim 1: After r rounds, if $dist_i$ is finite or $parent_i \neq \perp$, then $dist_i$ is actually the distance on some at-most-r-hop path from i_0 to i, and $parent_i$ is i's parent on such a path.
- Proof: LTTR---easy induction on r.
- Claim 2: After r rounds, if p is any at-most-r-hop path from i_0 to i, then $dist_i$ and $parent_i$ correspond to a path that is no longer than p.

Correctness

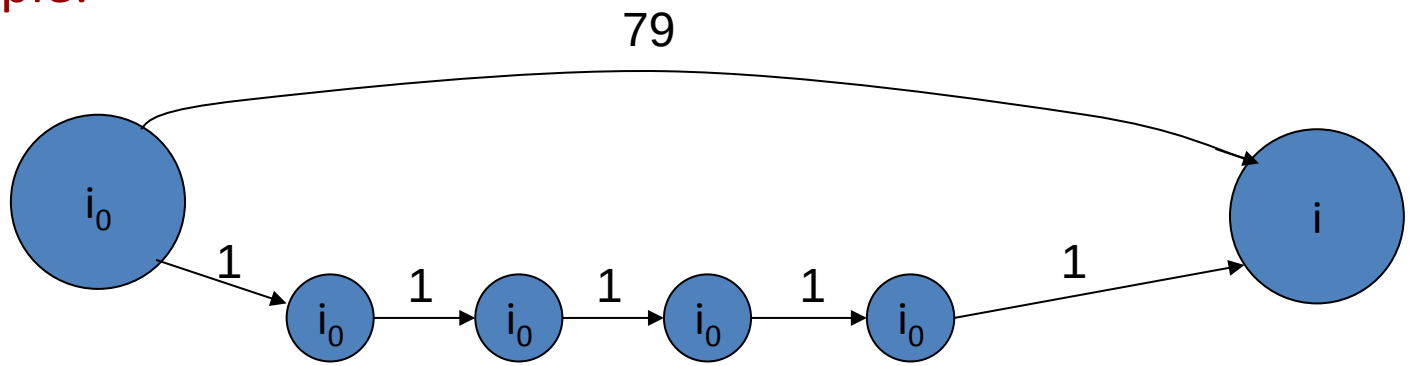
- Claim 2: After r rounds, if p is any at-most-r-hop path from i_0 to i, then $dist_i$ and $parent_i$ correspond to a path that is no longer than p.
- Proof:
  - By induction on r,
  - Base: $r = 0$: Immediate from initializations.
  - Inductive step: Assume for $r - 1$, show for r.
  - Fix at-most-r-hop path p from i_0 to i; we must show that after round r, $dist_i$ and $parent_i$ correspond to a path that is no longer than p.
  - Let j be i's predecessor on path p.
  - Let q be the path from i_0 to j that results from cutting off p at j.
  - Then q is an at-most-$(r-1)$-hop path from i_0 to j.
  - By inductive hypothesis, after round $r - 1$, $dist_j$ and $parent_j$ correspond to a path that is no longer than q.
  - At round r, j sends i its $dist_j$ information and process i takes this into account in calculating $dist_i$.
  - So after round r, $dist_i$ and $parent_i$ correspond to a path that is no longer than p, as needed.

Complexity

- Complexity:
  - Time: $n - 1$ rounds
  - Messages: $(n-1)\,|E|$
- Worse than BFS, which has:
  - Time: $diam$ rounds
  - Messages: $|E|$

- Q: Does the time bound really depend on n, or is it actually $O(diam)$?

- It's really n, since "shortest path" can be a path with many links.
- Example:



Remarks

- We will revisit Bellman-Ford for asynchronous networks.
- Gets even more expensive there.
- Similar to an old Arpanet routing algorithm.
- Consider more interesting termination strategy then.

Minimum Spanning Trees
[Gallager, Humblet, Spira]

Minimum Spanning Tree (MST)

- Another classical problem.
- Many sequential algorithms.
- Construct a spanning tree, minimizing the total weight of all edges in the tree.
- Assume:
  - Weighted undirected graph (bidirectional communication).
    - Weights are nonnegative reals.
    - Each node knows weights of incident edges.
  - Processes have UIDs.
  - Nodes know (a good upper bound on) n.
- Required:
  - Each process should decide which of its incident edges are in the MST and which are not.

Minimum spanning tree theory

- Graph theory definitions (for undirected graphs)
  - Tree: Connected acyclic graph
  - Forest: An acyclic graph (not necessarily connected)
  - Spanning subgraph of a graph G: Subgraph that includes all nodes of G.
    - Spanning tree, spanning forest.
  - Component of a graph: A maximal connected subgraph.

- Common strategy for computing MST:
  - Start with trivial spanning forest, n isolated nodes.
  - Repeat ($n - 1$ times):
    - Merge two components along an edge that connects them.
    - Specifically, add the minimum-weight outgoing edge ($MWOE$) of some component to the edge set of the current forest.

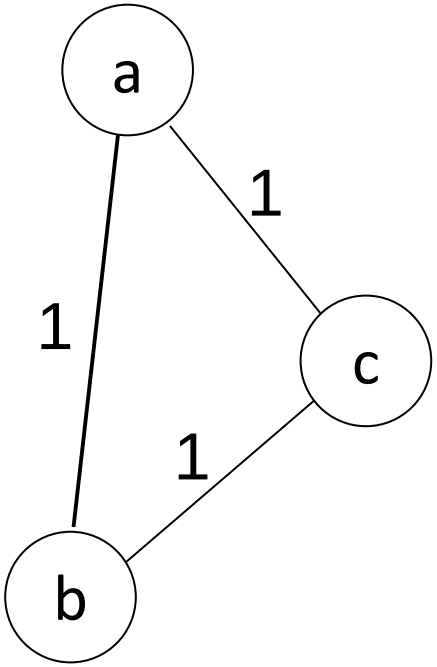
Why this works

- Lemma 1: Let $\{T_i: 1 \leq i \leq k\}$ be a spanning forest of G. Let e be a minimum weight outgoing edge of some T_j. Then there is a spanning tree for G that includes all the T_i and e, and has min weight among all spanning trees for G that include all the T_i.
- Proof:
  - Suppose not: then there is a spanning tree T for G that includes all the T_i and excludes e, and whose total weight is strictly less than that of any spanning tree that includes all the T_i and e.
  - Construct a new graph T' (not a tree) by adding e to T.
  - T' contains a cycle, which must contain another outgoing edge, e', of T_j.
  - $weight(e') \geq weight(e)$, by choice of e (it's a $MWOE$).
  - Construct a new tree T'' by removing e' from T'.
  - Then T'' is a spanning tree, and it contains all the T_i and e.
  - Furthermore, $weight(T'') \leq weight(T)$.
  - Contradicts assumed properties of T.

Minimum spanning tree algorithms

- General strategy:
  - Start with n isolated nodes.
  - Repeat ($n - 1$ times):
    - Choose some component j.
    - Add a minimum-weight outgoing edge of component j.
- Repeated use of Lemma 1 implies that this produces a MST.
- Sequential MST algorithms follow (special cases of) this general strategy:
  - Dijkstra/Prim: Grows one big component by adding one more node at each step, based on a min-weight outgoing edge of the component.
  - Kruskal: Always add a globally-min-weight edge.
- Distributed?
  - All components can choose simultaneously.
  - But there is a problem…

We can get cycles:



Minimum spanning tree

- Avoid such cycles by assuming that all weights are distinct.
- Not a serious restriction---we could break ties with UIDs.
- Lemma 2: If all weights are distinct, then the MST is unique.
- Proof: Another cycle-style argument (LTTR, in book p. 66).

- Justifies the following concurrent strategy:
  - At each stage, suppose (inductively) that the current forest contains only edges from the (unique) MST.
  - Now several components choose $MWOE$s concurrently.
  - Each of these edges is in the unique MST, by Lemma 1.
  - So it's OK to add them all (no cycles, since all are in the same MST).

- [Gallager, Humblet, Spira] algorithm (Dijkstra prize paper).
  - Designed for asynchronous networks, but simplified here.
  - We will revisit it in asynchronous networks.

GHS distributed MST algorithm

- Proceeds in phases (levels), each with $O(n)$ rounds.
  - Length of phases is fixed, and known to everyone.
  - This is all that we use n is for.
  - We'll remove this use of n for the asynchronous algorithm.

- For each $k \geq 0$, level k components form a spanning forest that is a subgraph of the unique MST.
- Each component is a tree rooted at a leader node.
  - Component identified by UID of leader.
  - Nodes in the component know which incident edges are in the tree.
- Every level $k+1$ component is constructed from two or more level k components.
  - So each level k component has at least 2^k nodes.

- Level 0 components: Single nodes.
- Level $k \rightarrow$ Level $k+1$:

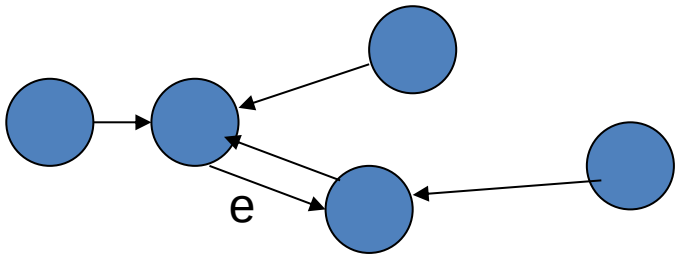
Level $k \rightarrow$ Level $k+1$

- Each level k component leader finds the $MWOE$ of its component:
  - Broadcasts $search$ (via tree edges).
  - Each process finds the $mwoe$ among its own incident edges.
    - Sends $test$ messages along non-tree edges, asking if the node at the other end is in the same component (compare component ids).
  - Convergecast the min back to the leader (via tree edges).
  - Leader determines $MWOE$ of the component.

- Combine level k components using $MWOE$s, to obtain level $k+1$ components:
  - Wait long enough for all components to find $MWOE$s.
  - Leader of each level k component tells endpoint nodes of its $MWOE$ to add the edge for level $k+1$.
  - Each new component has $\geq 2^{k+1}$ nodes, as claimed.

Level $k \rightarrow$ Level $k+1$

- Each level k component leader finds the $MWOE$ of its component.
- Combine level k components using $MWOE$s, to obtain level $k+1$ components.
- Choose new leaders:
  - For each new, level $k+1$ component, there is a unique edge e that is the $MWOE$ of two level k sub-components:



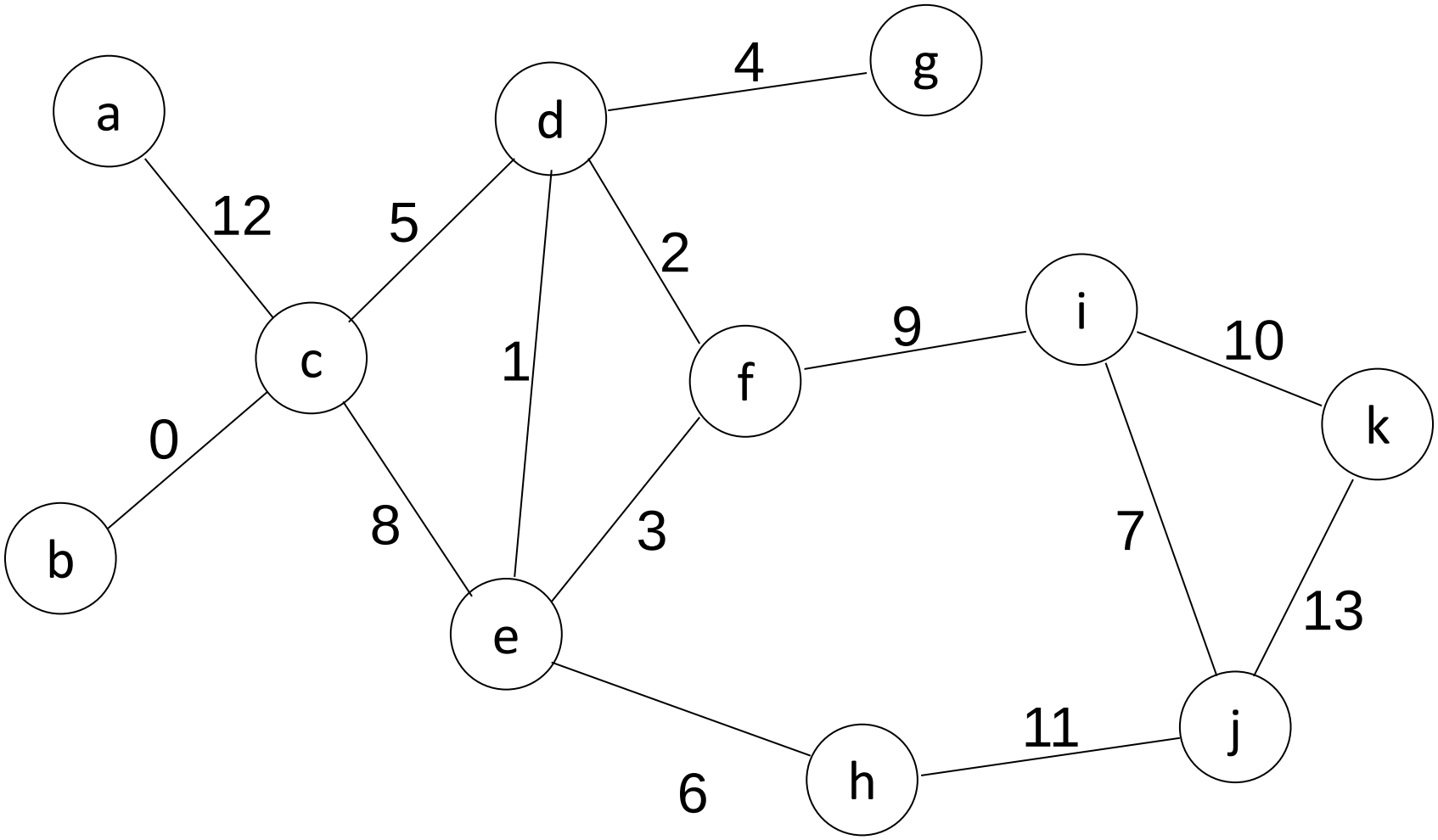
Must have a cycle.
Cycle can't have > 2 edges because weights of edges on the cycle must decrease around the cycle.

  - Choose the new leader to be the endpoint of e with the larger UID.
  - Broadcast leader UID throughout new (merged) component.
- GHS terminates when a leader finds no outgoing edges.

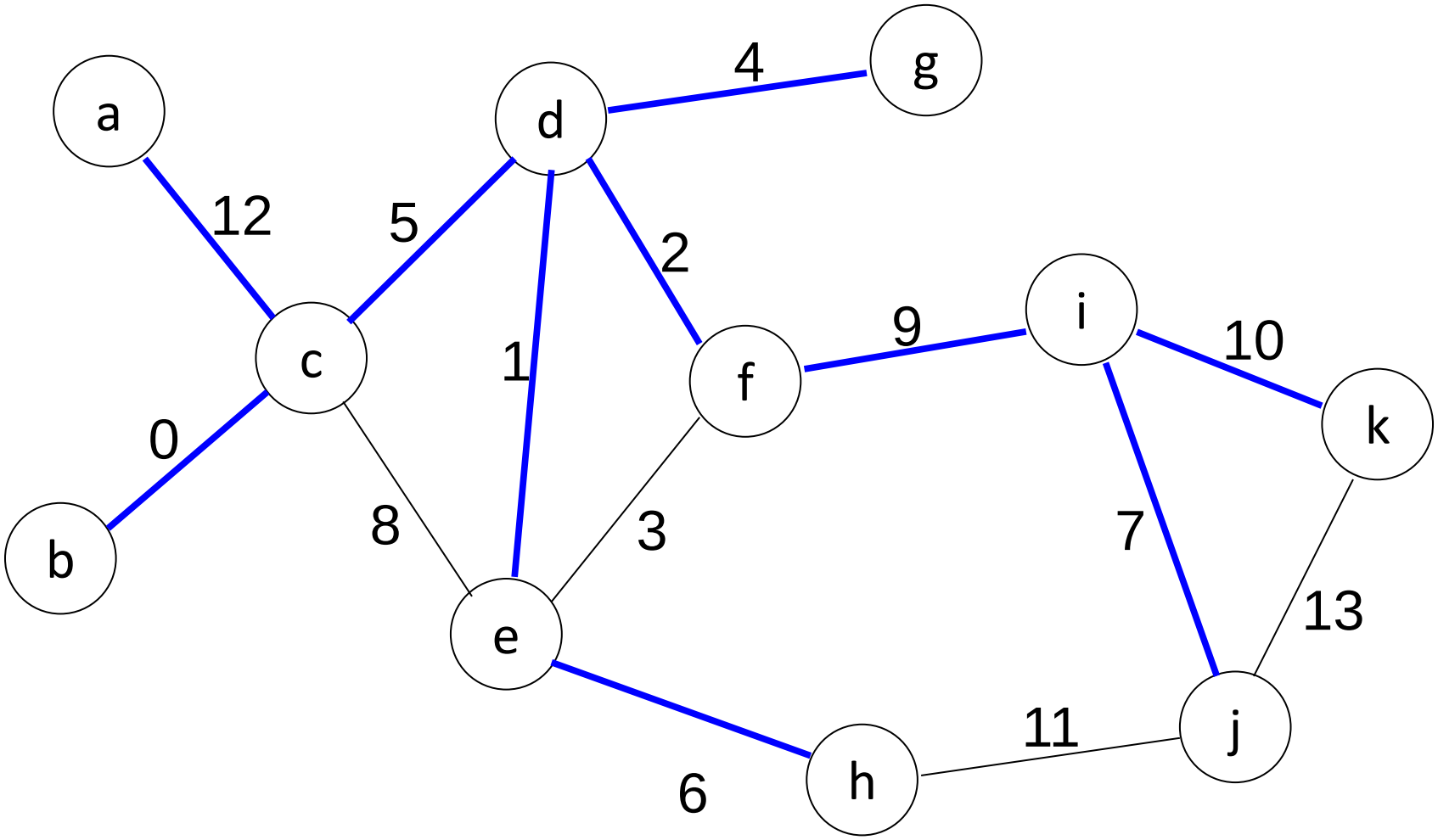
Note on synchronization

- This simplified version of GHS is designed to work with component levels synchronized.
- Difficulties can arise when they get out of synch (as we'll see later).
- In particular, $test$ messages are supposed to compare leader UIDs to determine whether endpoints are in the same component.
- This requires that the node being queried has up-to-date UID information.

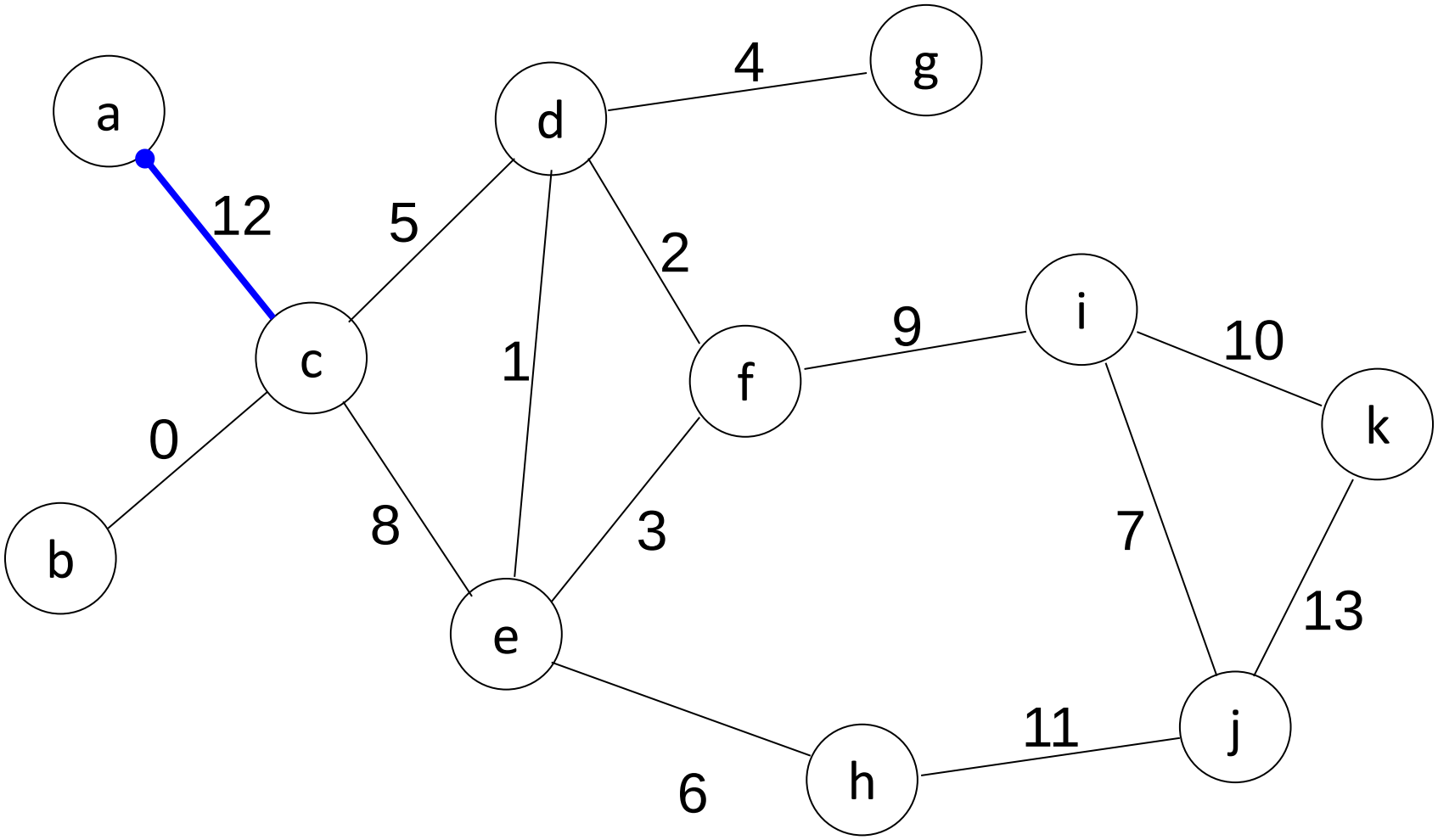
Minimum spanning tree



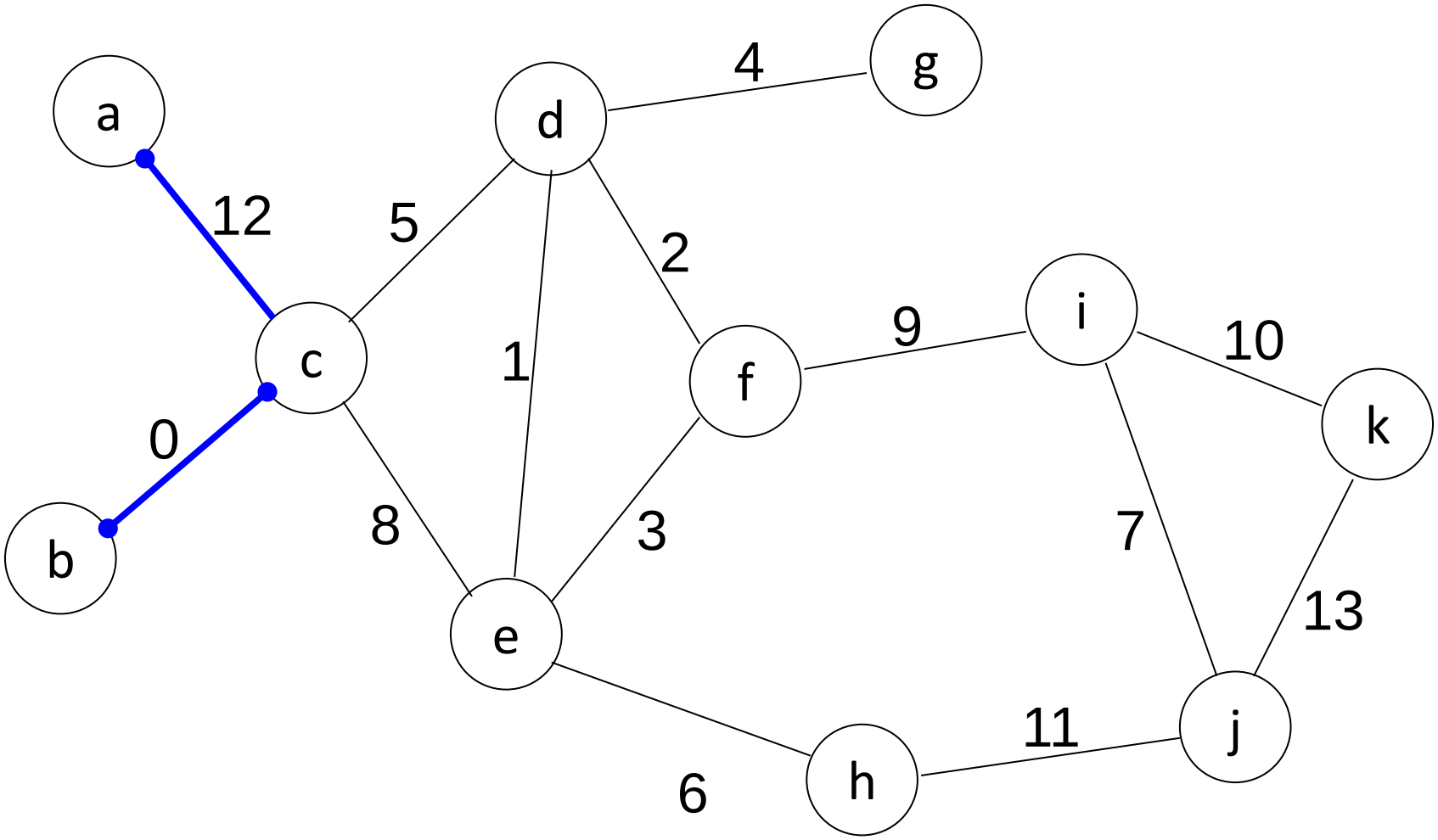
Minimum spanning tree



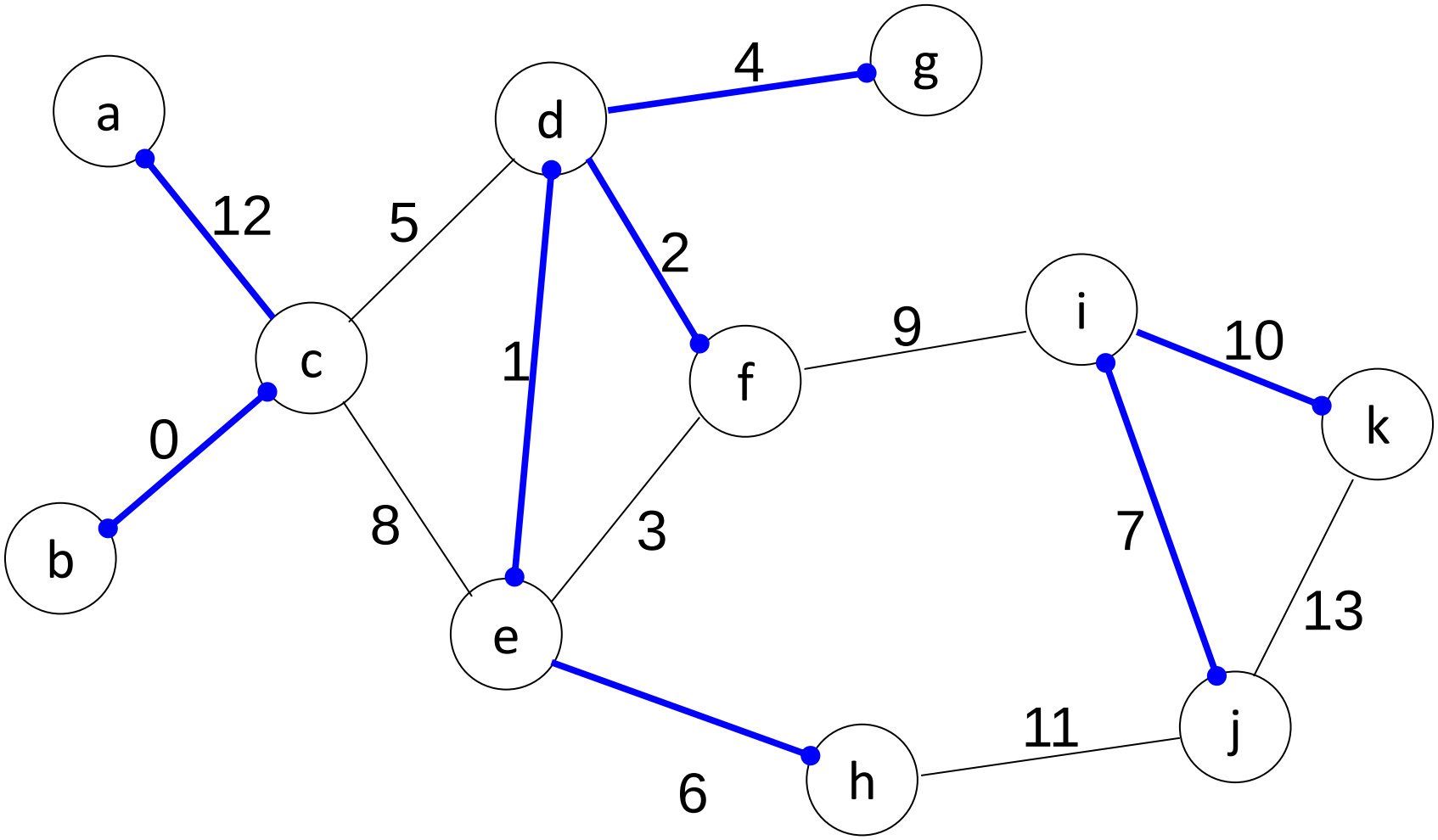
Minimum spanning tree



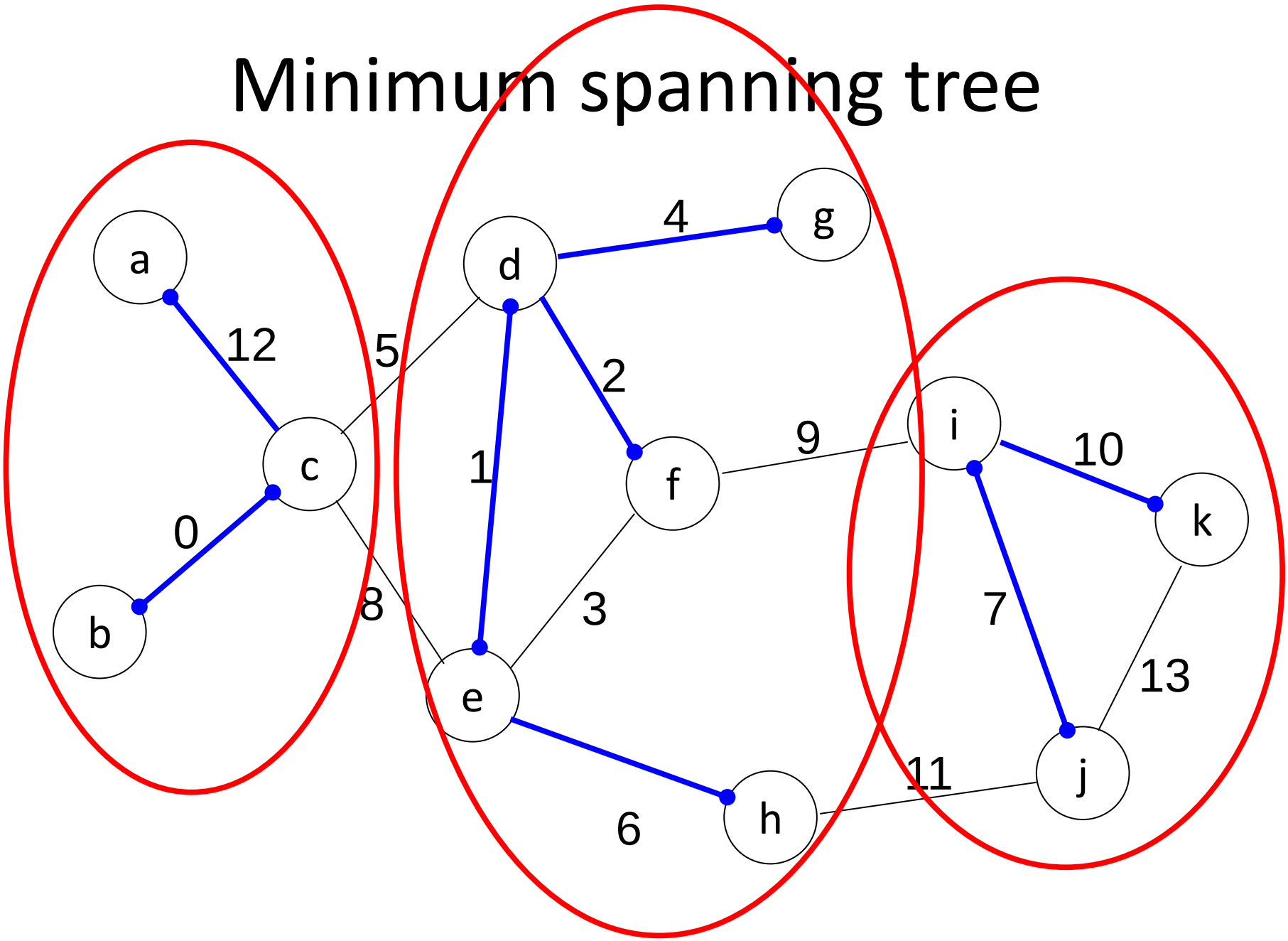
Minimum spanning tree



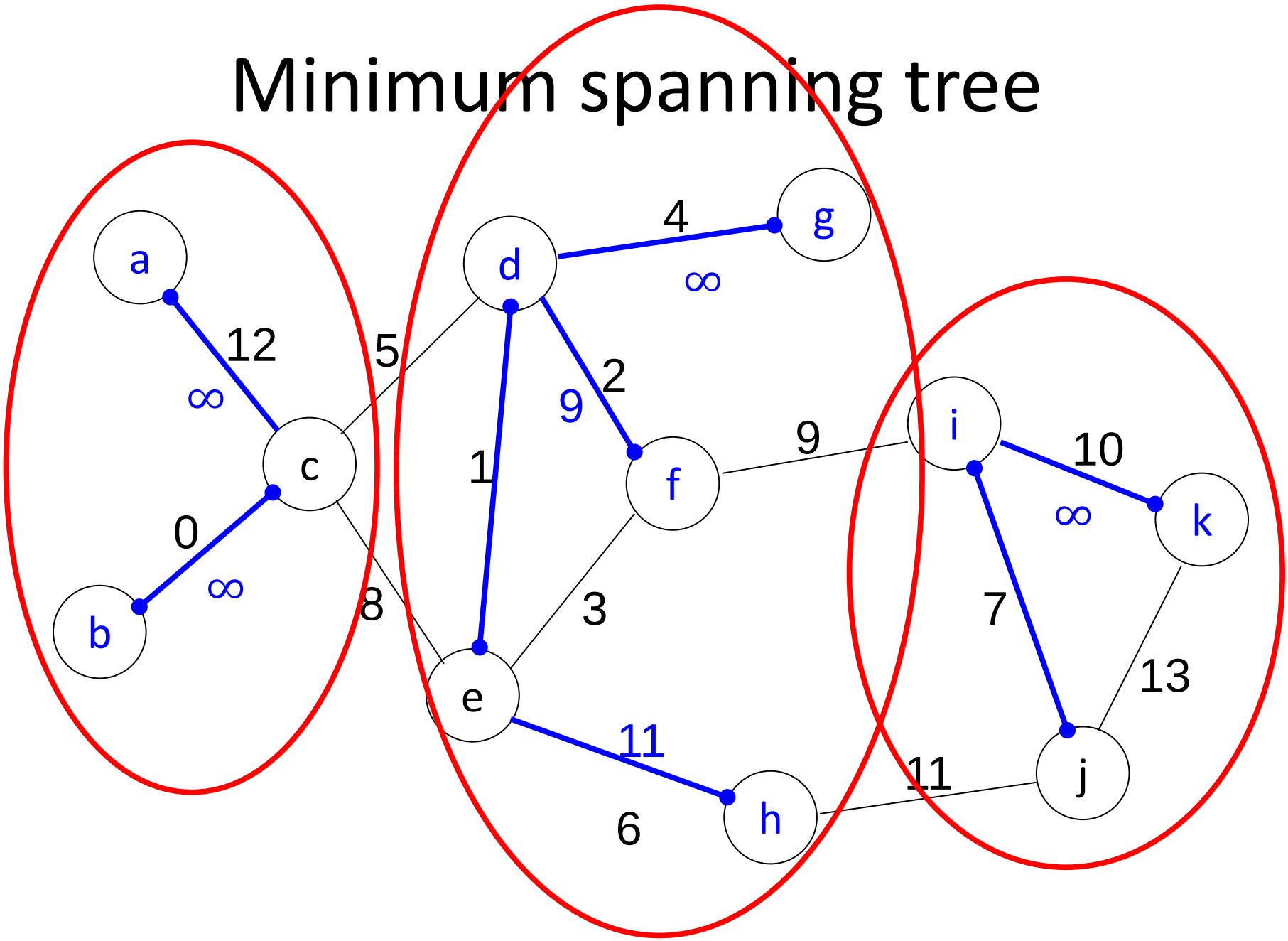
Minimum spanning tree



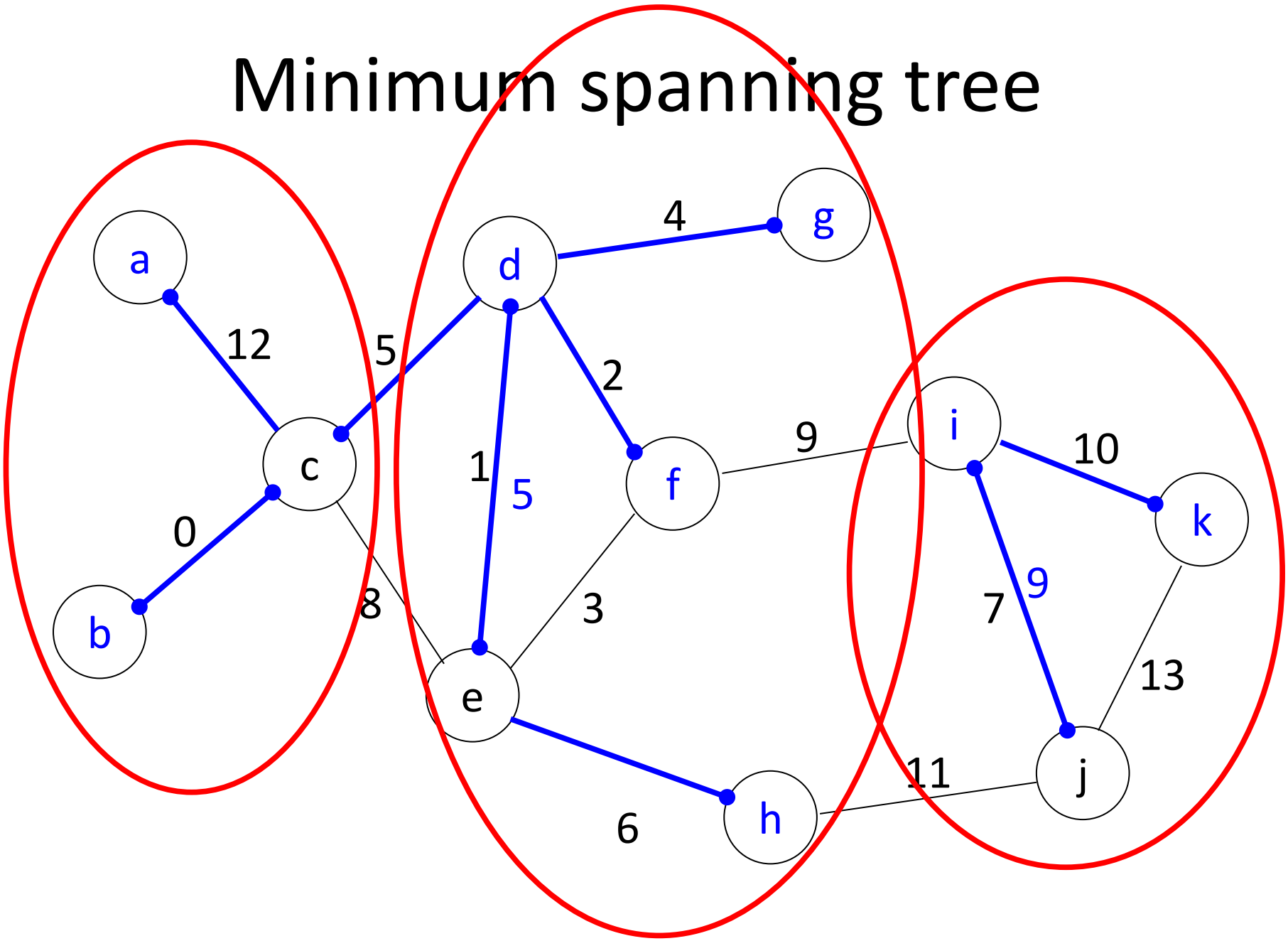
Minimum spanning tree



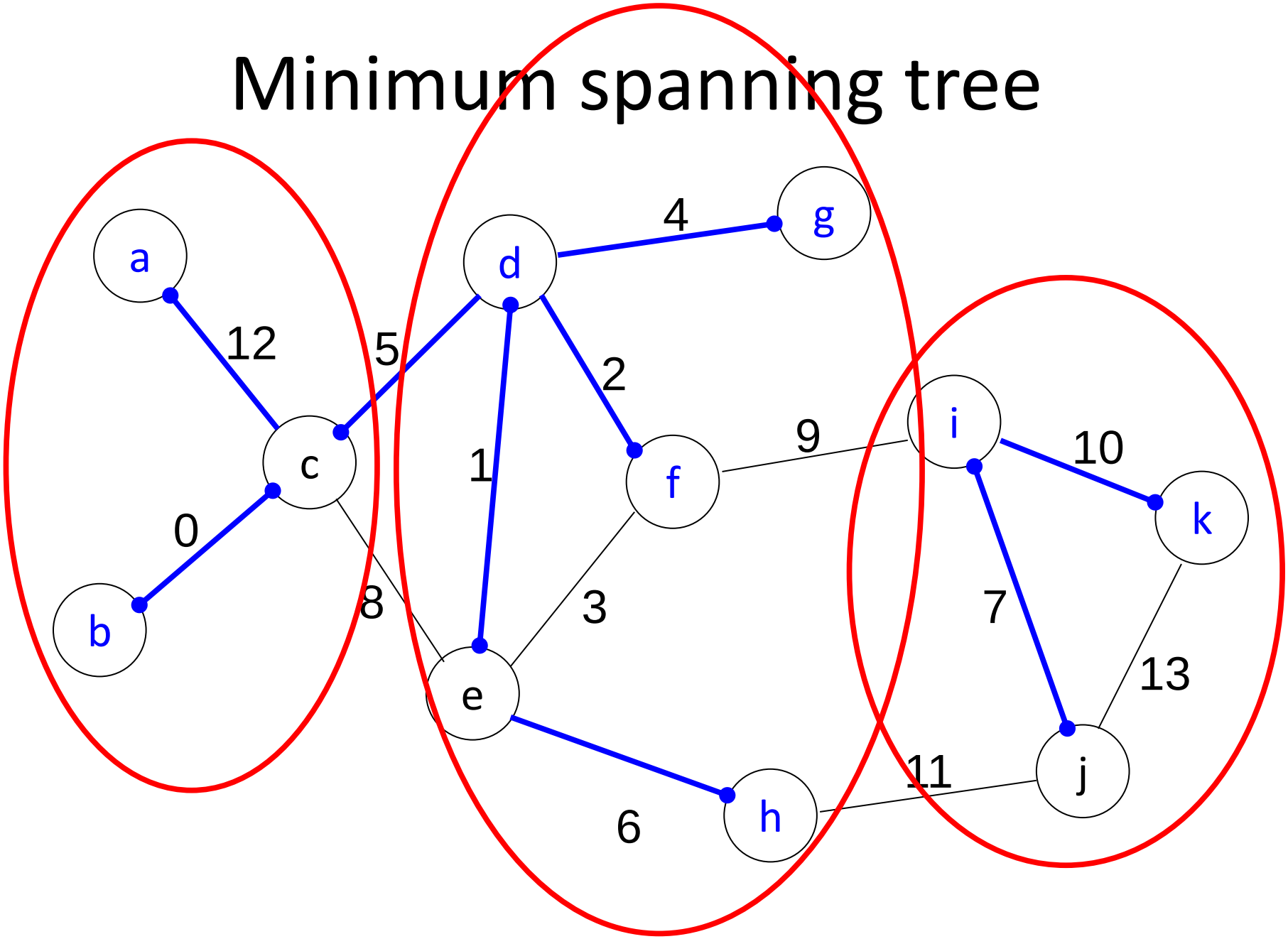
Minimum spanning tree



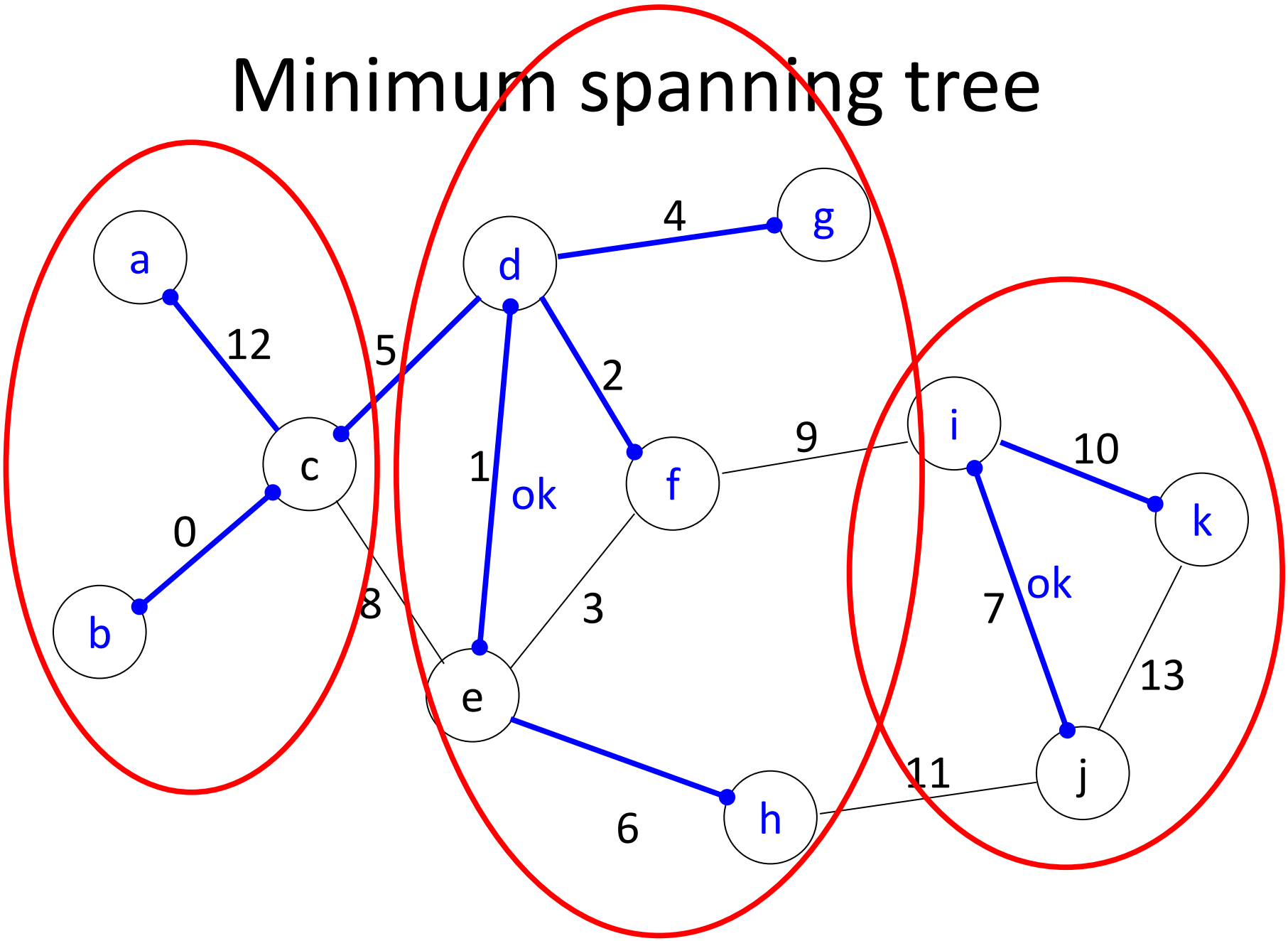
Minimum spanning tree



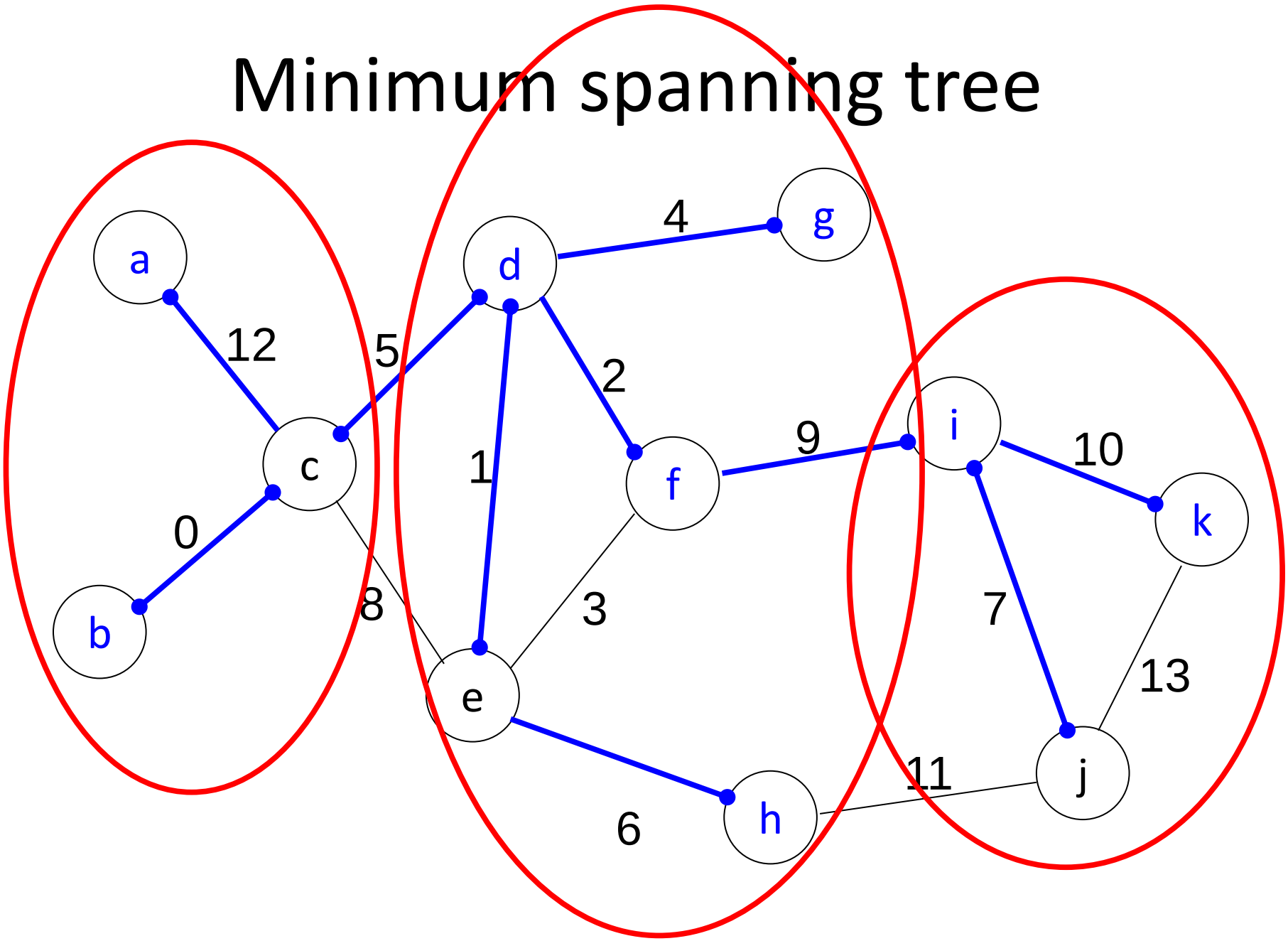
Minimum spanning tree



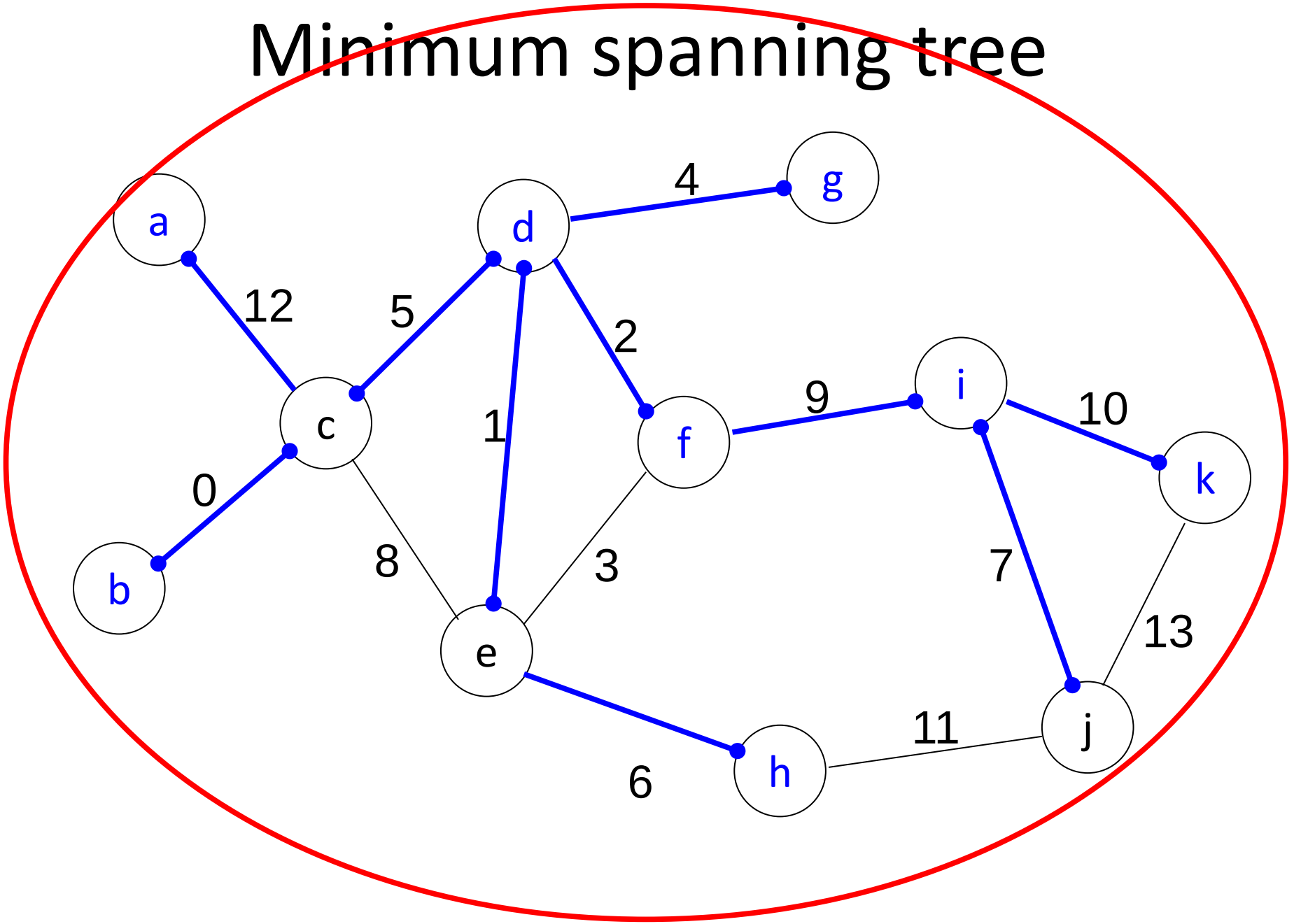
Minimum spanning tree



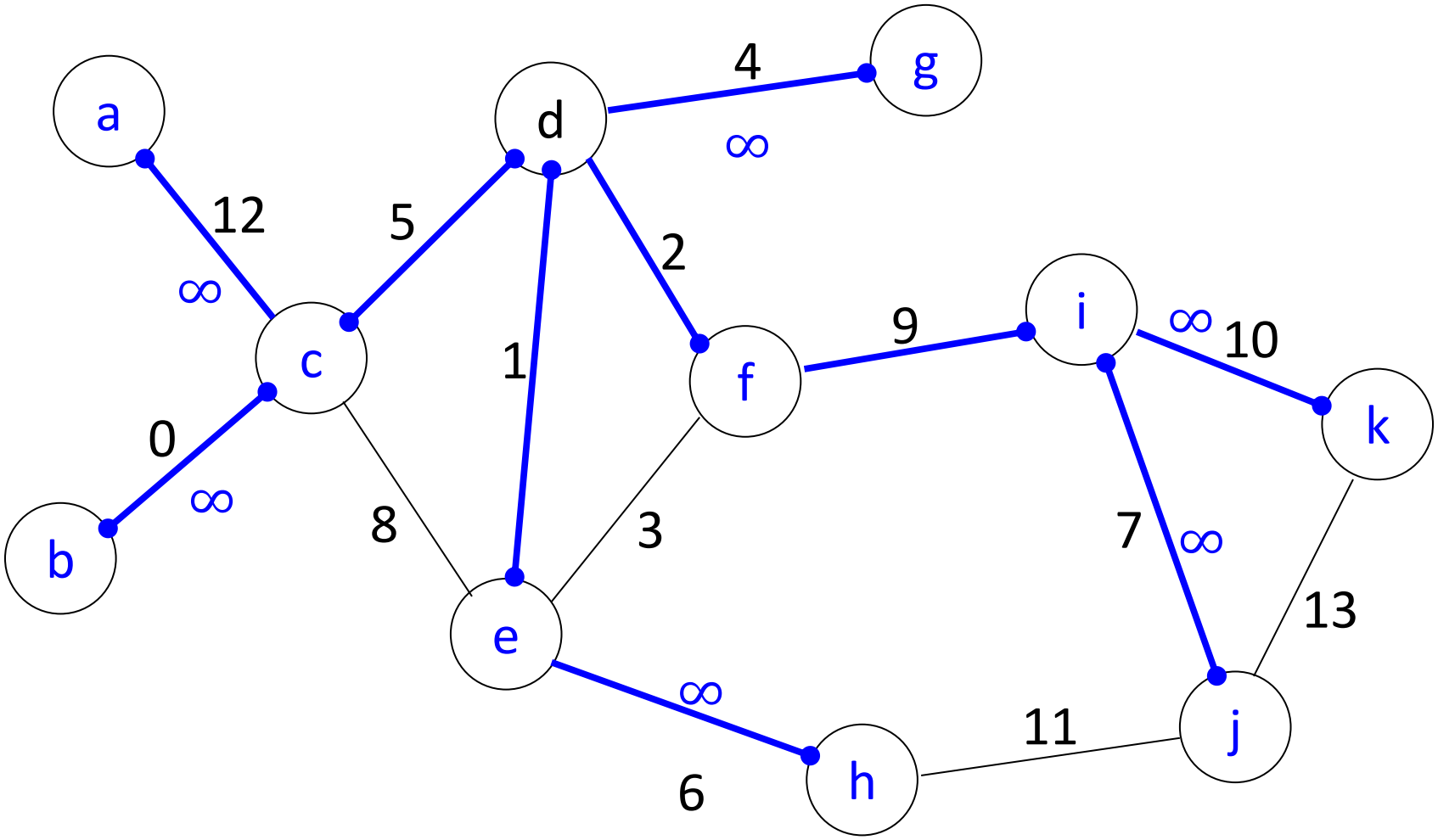
Minimum spanning tree



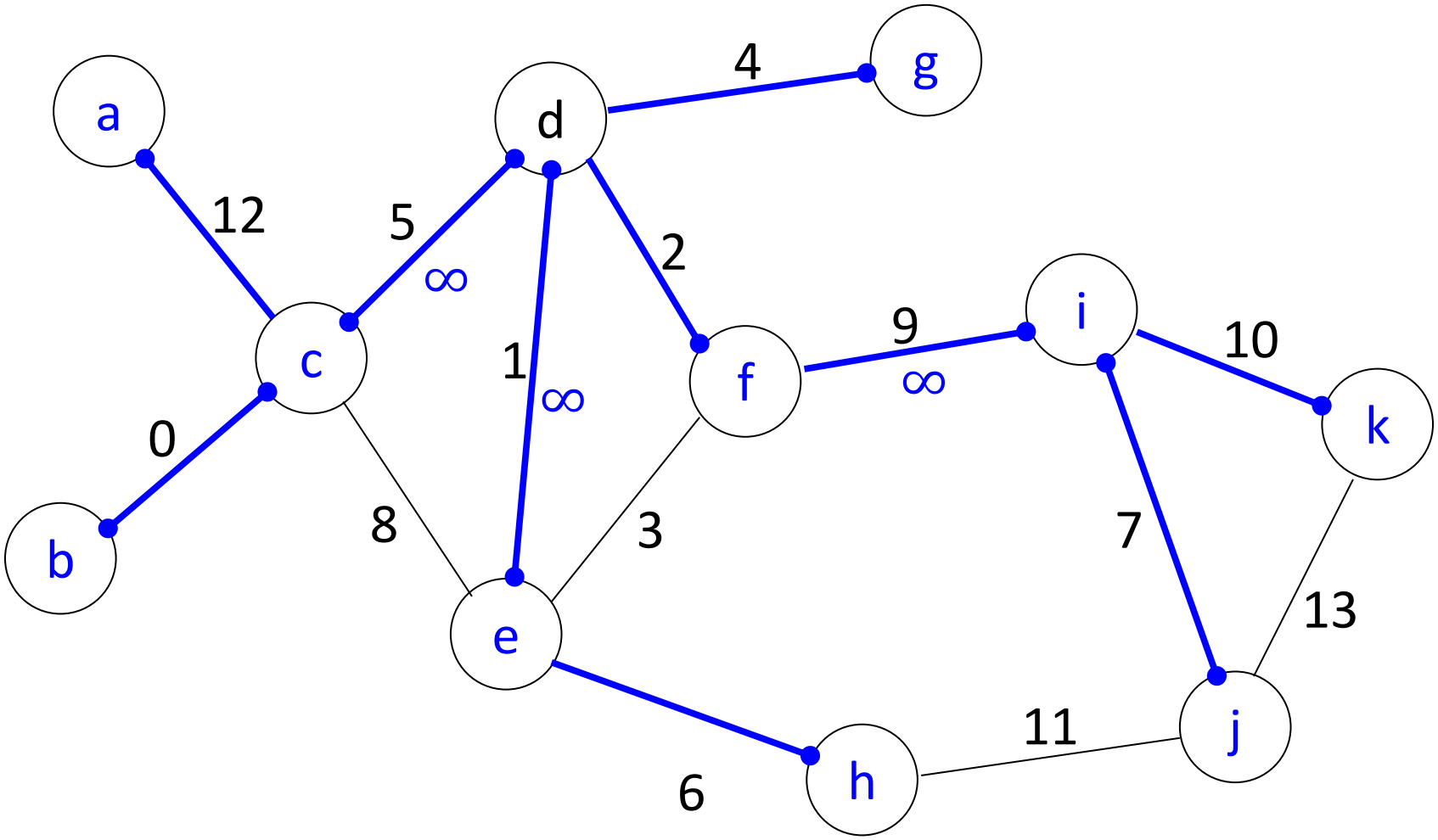
Minimum spanning tree



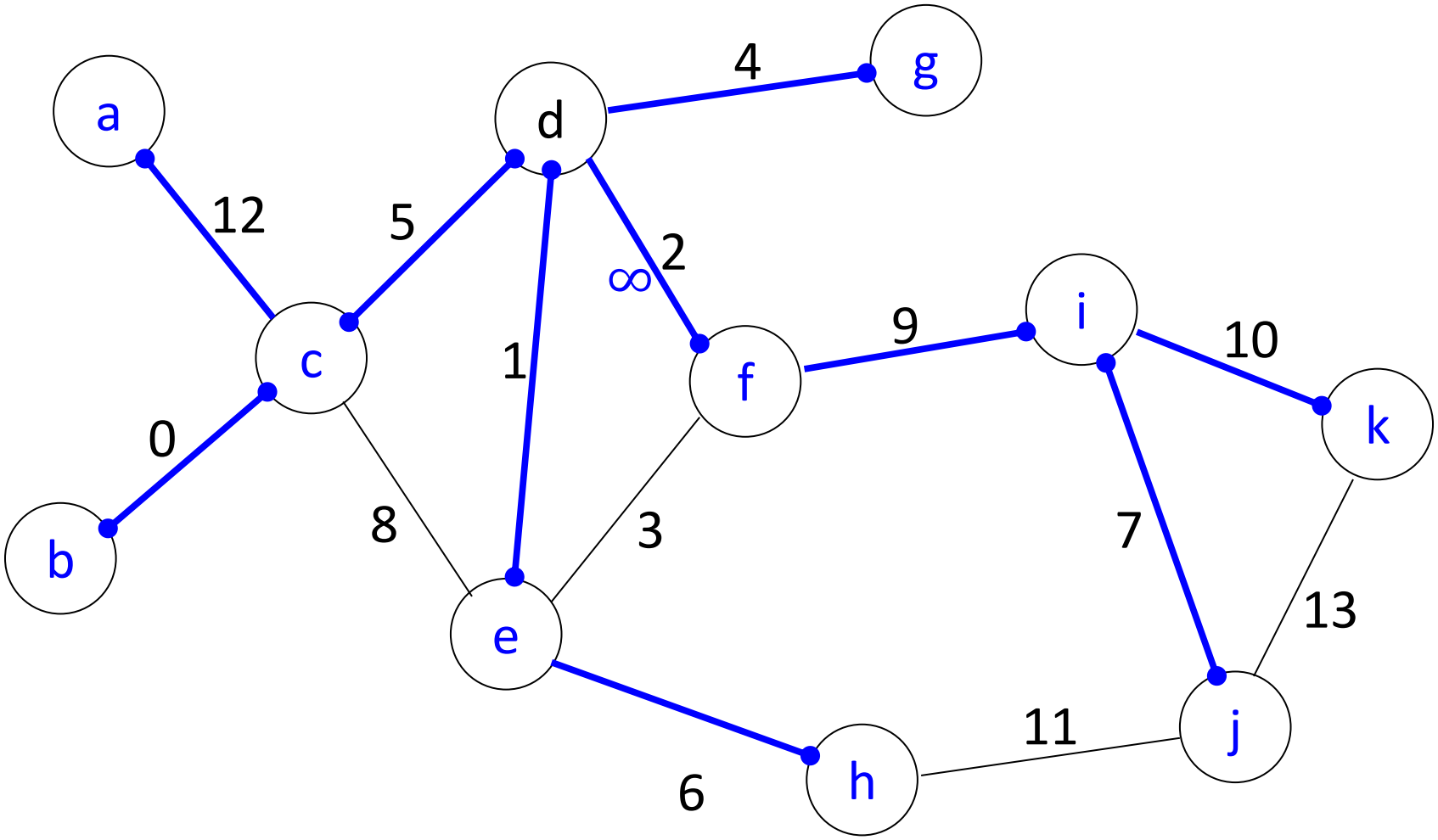
Minimum spanning tree



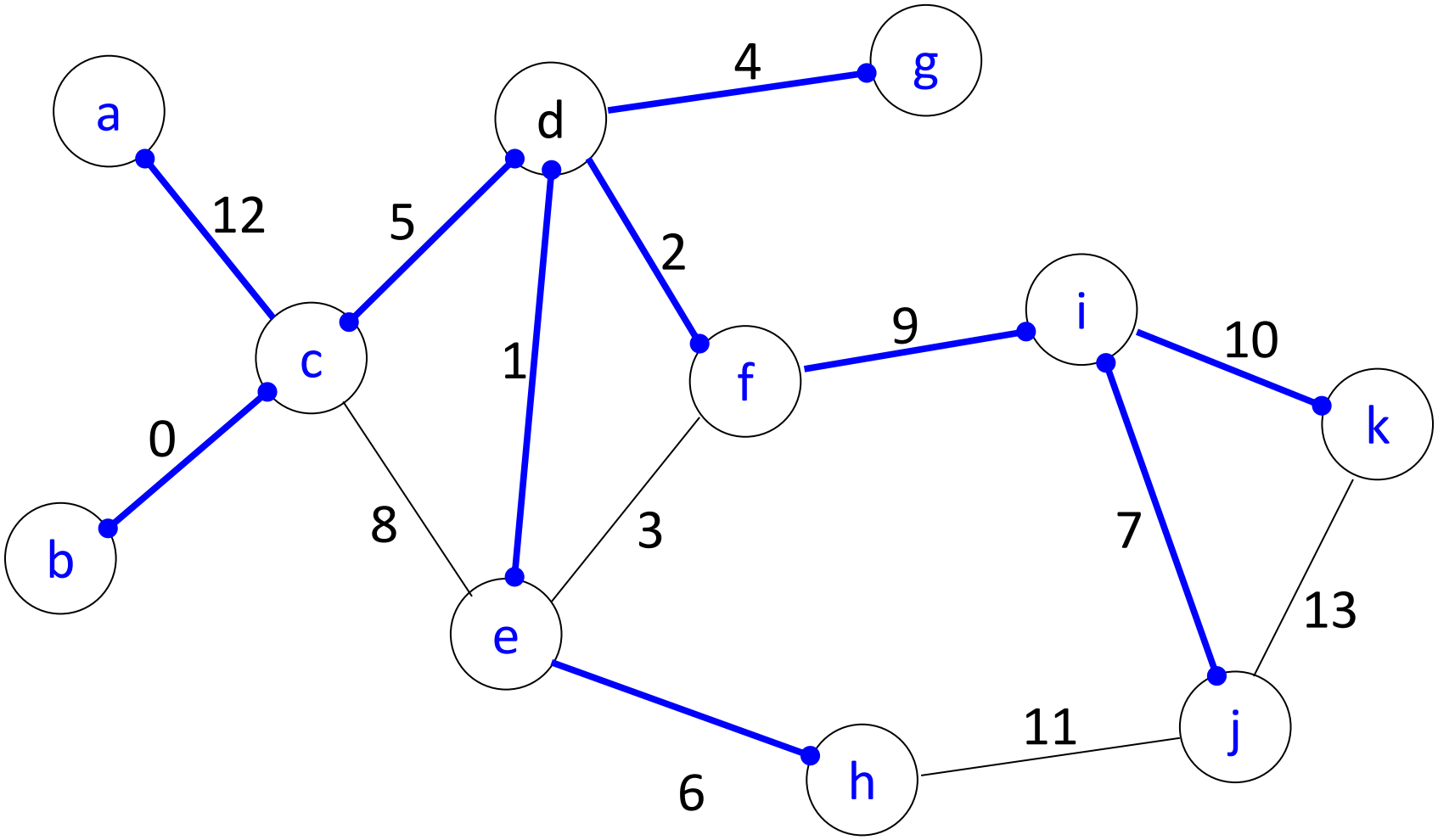
Minimum spanning tree



Minimum spanning tree



Minimum spanning tree



Proof, Analysis

- Proof:
  - Uses invariants.
  - Long and complicated because the algorithm is complicated, lots of multi-part invariants.
- Time complexity: $O(n \log n)$
  - n rounds for each level
  - $\log n$ levels, because there are $\geq 2^k$ nodes in each level k component.
- Messages: $O\big((n + |E|) \log n\big)$
  - Naïve analysis.
  - At each level, $O(n)$ messages are sent on tree edges, $O(|E|)$ messages for all the test messages and their responses.
- Can do better: $O(n \log n + |E|)$

Message Complexity Bound

- Messages: $O(\ (n\ +\ |E|) \log n)$
  - At each level, $O(n)$ messages sent on tree edges, $O(|E|)$ messages overall for all the $test$ messages and their responses.
- Messages: $O(n \log n\ +\ |E|)$
  - A surprising, significant reduction.
  - Trick also works in the asynchronous setting.
  - Has implications for other problems, such as leader election.

$O(n \log n + |E|)$ message complexity

- Each process i marks its incident edges as $rejected$ when they are first discovered to lead to the same component; no need to retest them.
- At each level, process i tests candidate edges one at a time, in order of increasing weight, until it finds the first one that leads outside the component (or until it exhausts the candidates).
- Rejects all edges that are found to lead to same component.
- At next level, resumes where it left off.
- $O(n \log n + |E|)$ bound:
  - $O(n)$ for messages on tree edges at each phase, $O(n \log n)$ total.
  - $test, accept$ (different component), $reject$ (same component):
    - Amortized analysis.
    - $Test - reject$: Each (directed) edge has at most one $test - reject$ pair ever, for $O(|E|)$ total.
    - $Test - accept$: Can accept the same directed edge several times; but at most one $test - accept$ per node per level, $O(n \log n)$ total.

Some implications

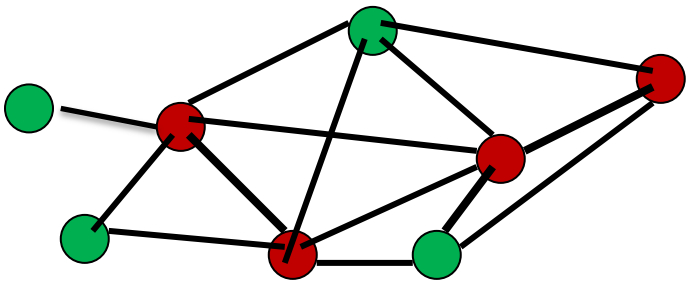
- Given any spanning tree of an undirected graph, elect a leader:
  - Convergecast from the leaves, until messages meet at a node (which can become the leader) or cross on an edge (choose endpoint with the larger UID).
  - Complexity: Time $O(n)$; Messages $O(n)$

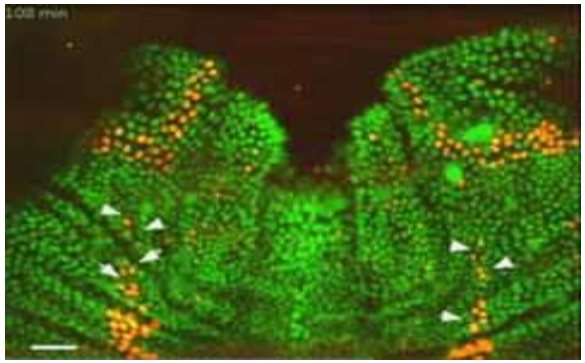
- Given any weighted connected undirected graph, with known n, but no leader, elect a leader:
  - First use GHS MST to get a spanning tree, then use the spanning tree to elect a leader.
  - Complexity: Time $O(n \log n)$; Messages $O(n \log n + |E|)$.
- Example: In a ring, $O(n \log n)$ time and messages.

Other graph problems…

- We can define a distributed version of practically any graph problem: Maximal Independent Set (MIS), dominating set, graph coloring,…
- Many of these have been well studied.
- One more example today…
- More next week…

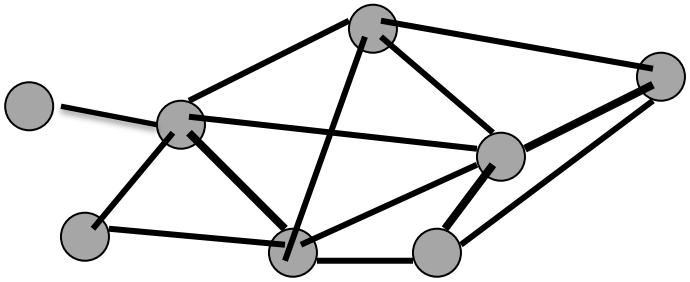
Maximal Independent Set



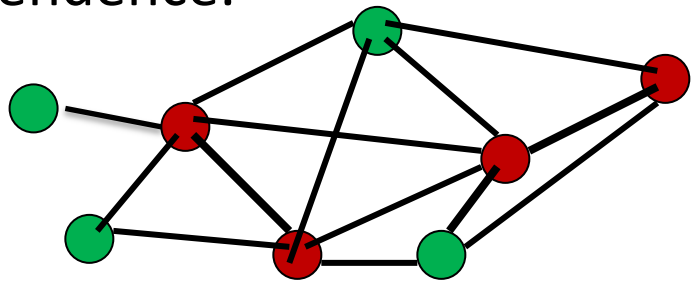


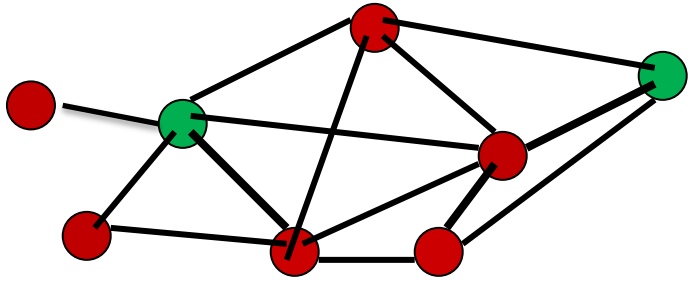
Maximal Independent Set (MIS)

- Assume a general undirected graph network, with processes at the nodes:
- Problem: Select a subset S of the processes, so that they form a Maximal Independent Set.



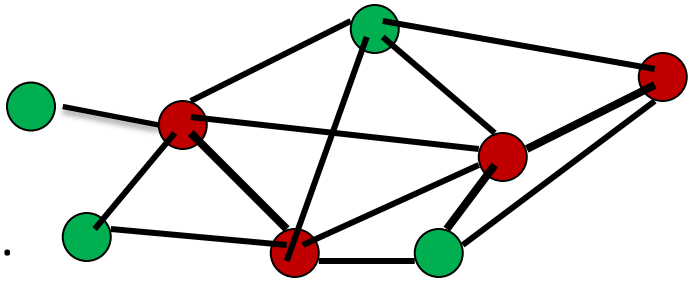
- Independent: No two neighbors are both in the set.
- Maximal: We can't add any more nodes without violating independence.





- An MIS is a Dominating Set: Every node i is either in the set S or has a neighbor in S.

Distributed MIS Problem

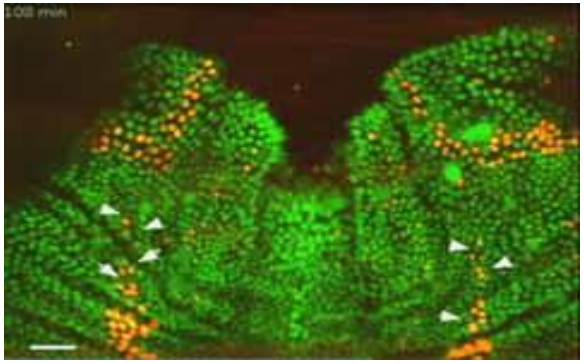


- Assume:
  - No UIDs
  - Processes know a good upper bound on n.
- Require:
  - Compute an MIS S of the network graph.
  - Each process in S should output $winner$, others output $loser$.

- Unsolvable by deterministic algorithms, in some graphs.
- So consider probabilistic algorithms:
  - Processes can make random choices.
  - Formally, in each round, a process executes a random step (choosing a new state from a probability distribution of states).
  - Leads to a probability distribution on executions.
  - Which allows us to talk about probabilistic properties of executions.

Applications of Distributed MIS

- Communication networks:
  - Selected processes can take charge of communication, convey information to all the other processes.
- Wireless network transmission:
  - Suppose MIS processes transmit messages, others listen.
  - A transmitted message reaches neighbors in the graph.
  - Assume a receiver actually receives everything that reaches it (ignoring the possibility of message collisions).
  - Independence guarantees that no two neighbors transmit, so all messages are received by all transmitters' neighbors.
  - Maximality implies that everyone either transmits or receives a message.
- Developmental biology:
  - Distinguish cells in fruit fly's nervous system to become "Sensory Organ Precursor" cells [Afek, Alon,…Bar-Joseph].



Luby's MIS Algorithm

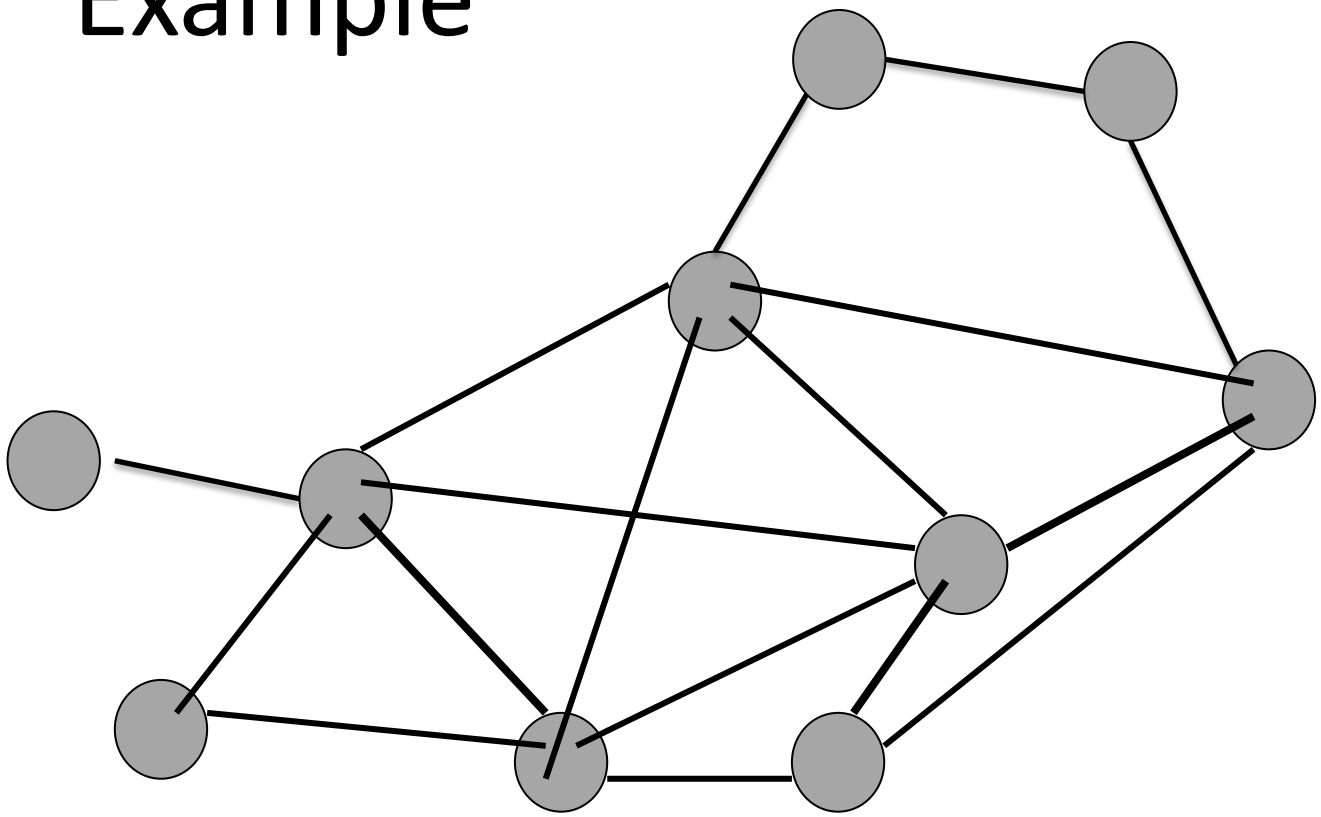
- Executes in 2-round phases.
- Initially all nodes are $active$.
- At each phase, some active nodes decide to be $winners$, others decide to be $losers$, algorithm continues to the next phase with a smaller graph (removing decided nodes and all their incident edges)
- Repeat until all nodes have decided.

- Behavior of active node i at phase ph:
- Round 1:
  - Choose a random value r in $\{1,2, \dots, n^5\}$, send it to all neighbors.
  - Receive values from all active neighbors.
  - If r is strictly greater than all received values, then join the MIS, output $winner$.
- Round 2:
  - If you joined the MIS, announce it in messages to all (active) neighbors.
  - If you receive such an announcement, decide not to join the MIS, output $loser$.
  - If you decided one way or the other at this phase, become $inactive$.

Properties of Luby's Algorithm

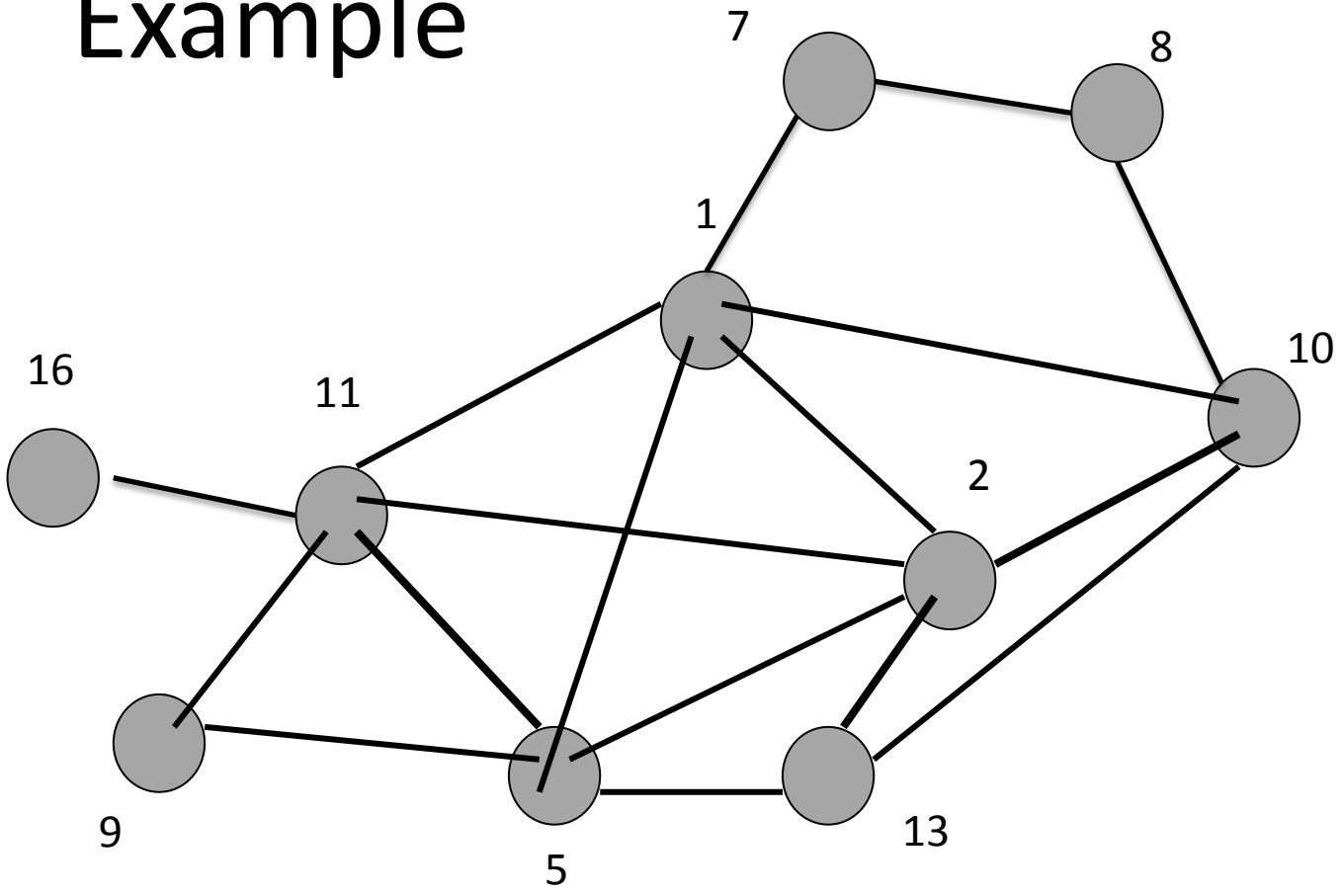
- Theorem: If LubyMIS ever terminates, it produces an MIS.
- Theorem: With probability 1, it eventually terminates; the expected number of rounds until termination is $O(\log n)$.
- Proofs: In a minute…

Example



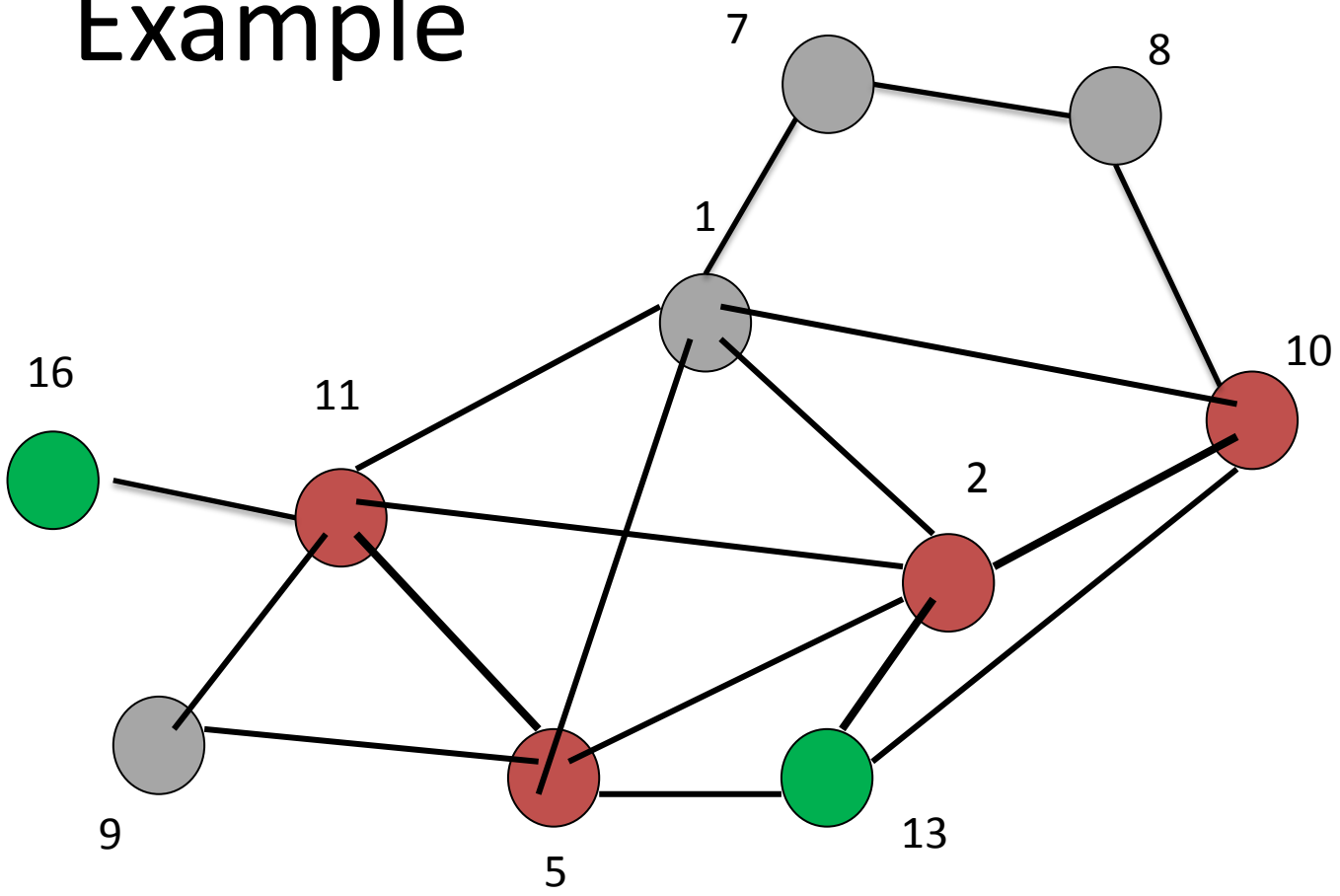
- All nodes start out identical.

Example

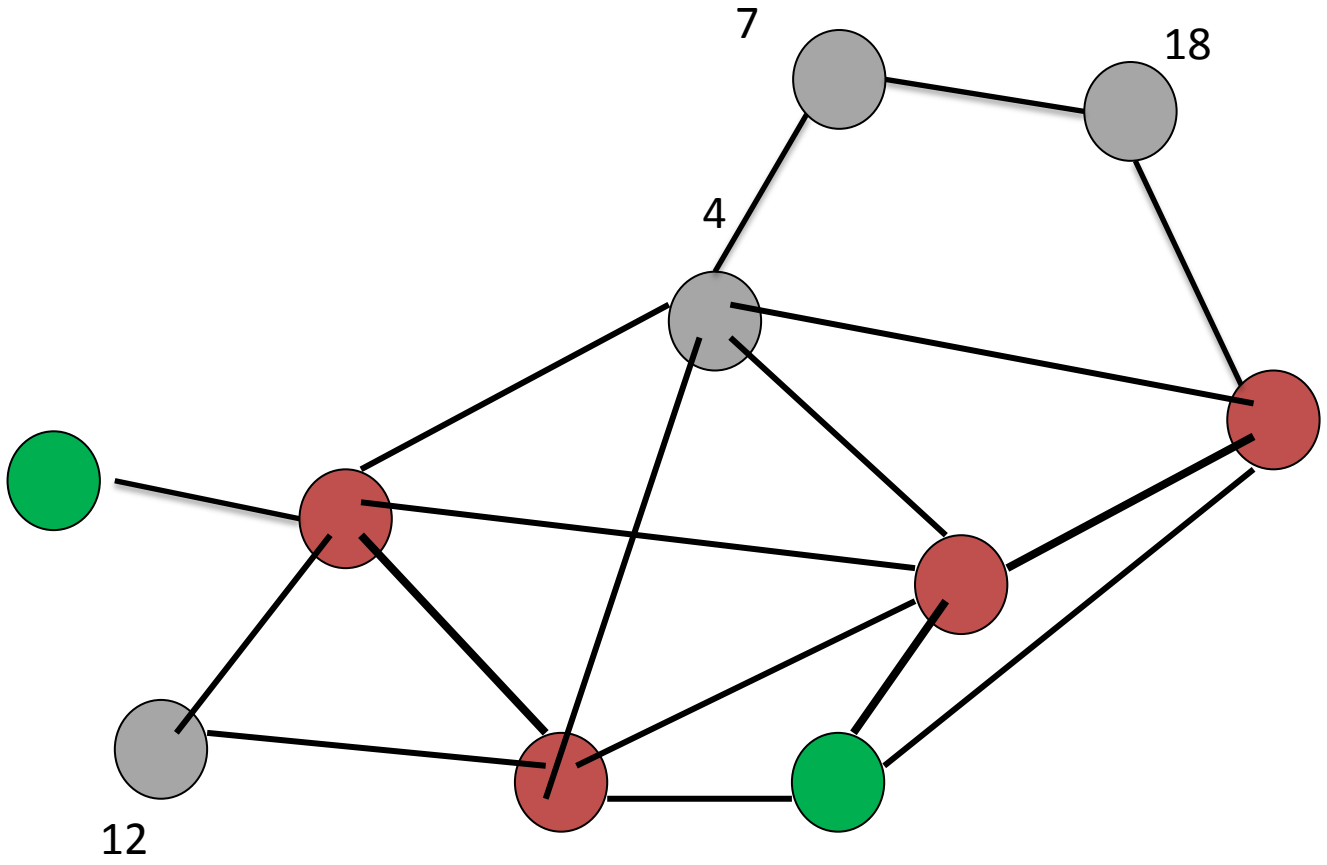


- Everyone chooses an ID.

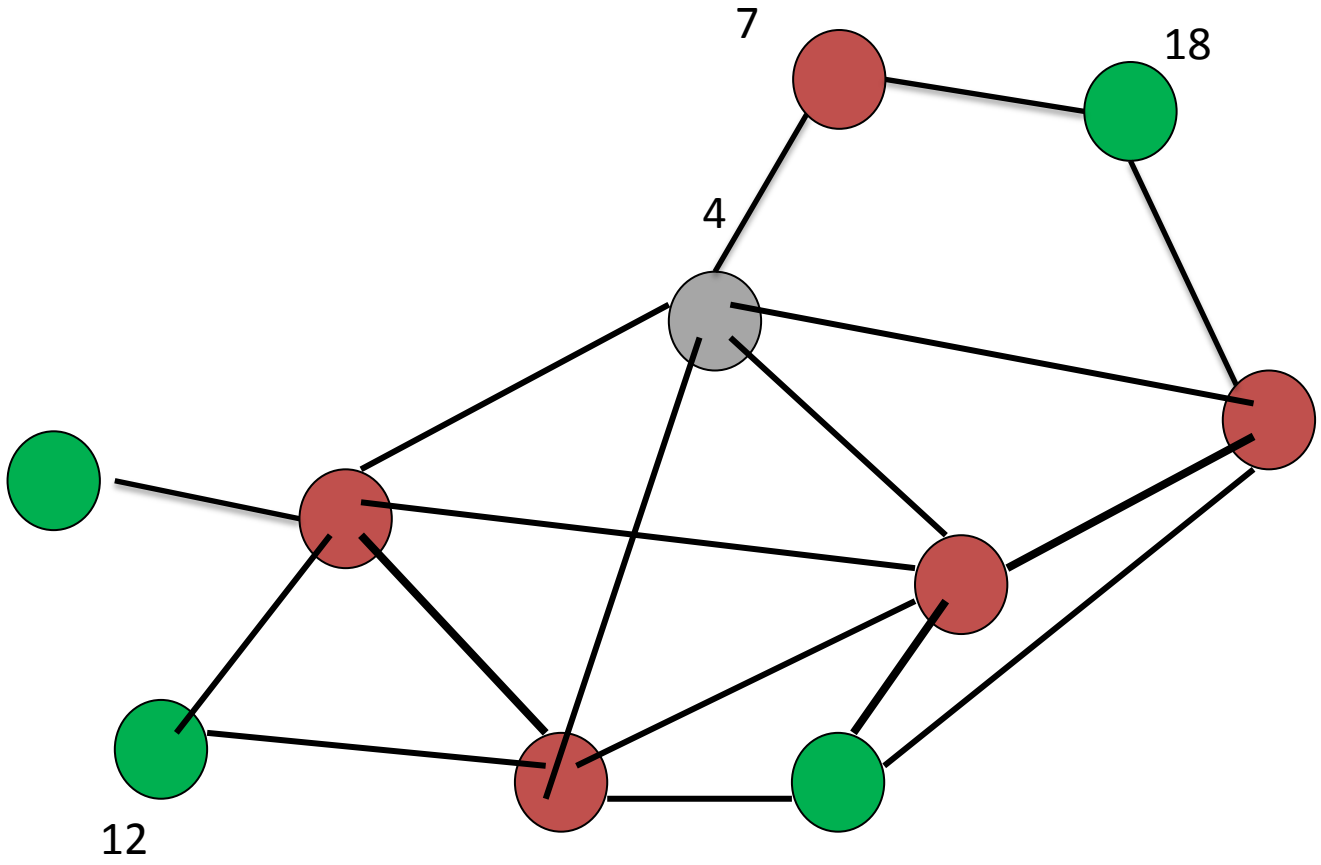
Example



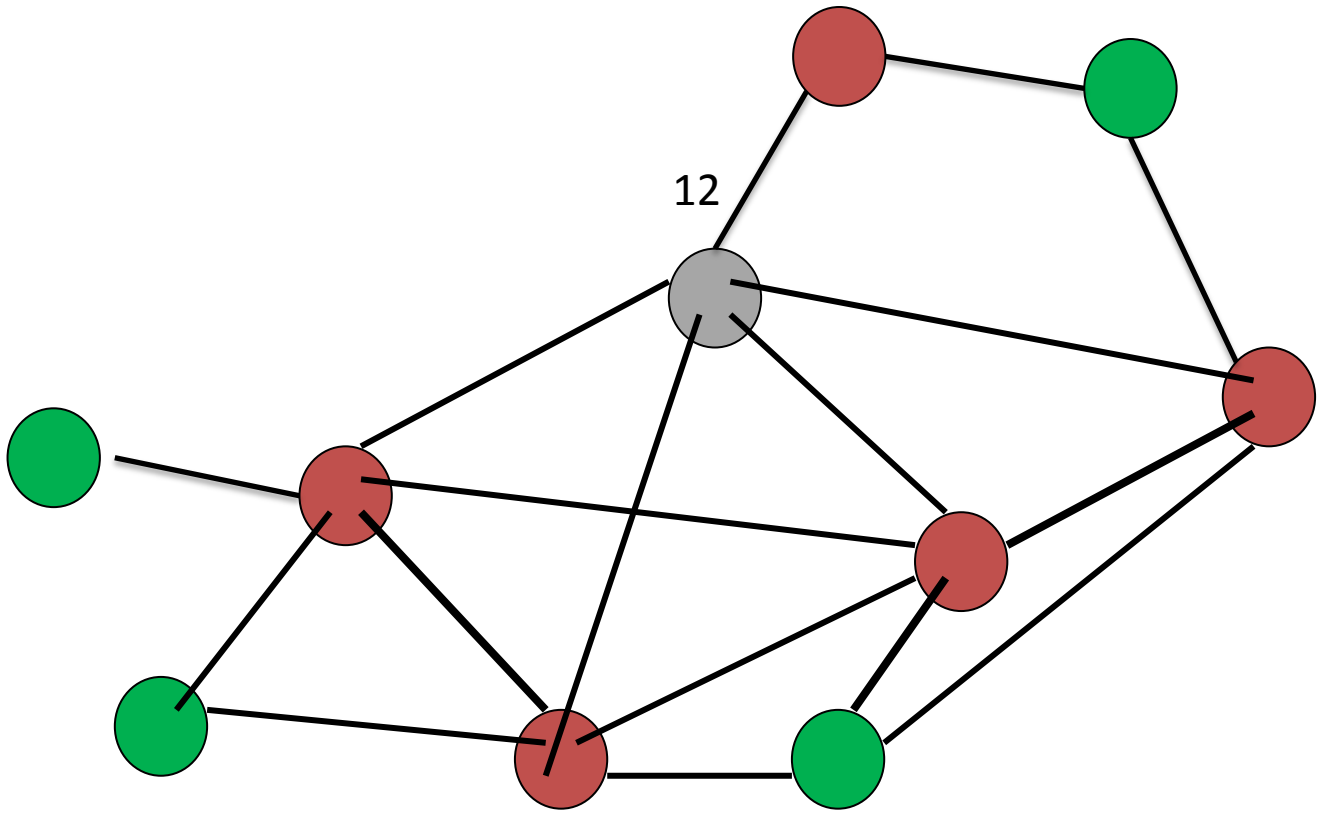
- Processes that chose 16 and 13 are in.
- Processes that chose 11, 5, 2, and 10 are out.



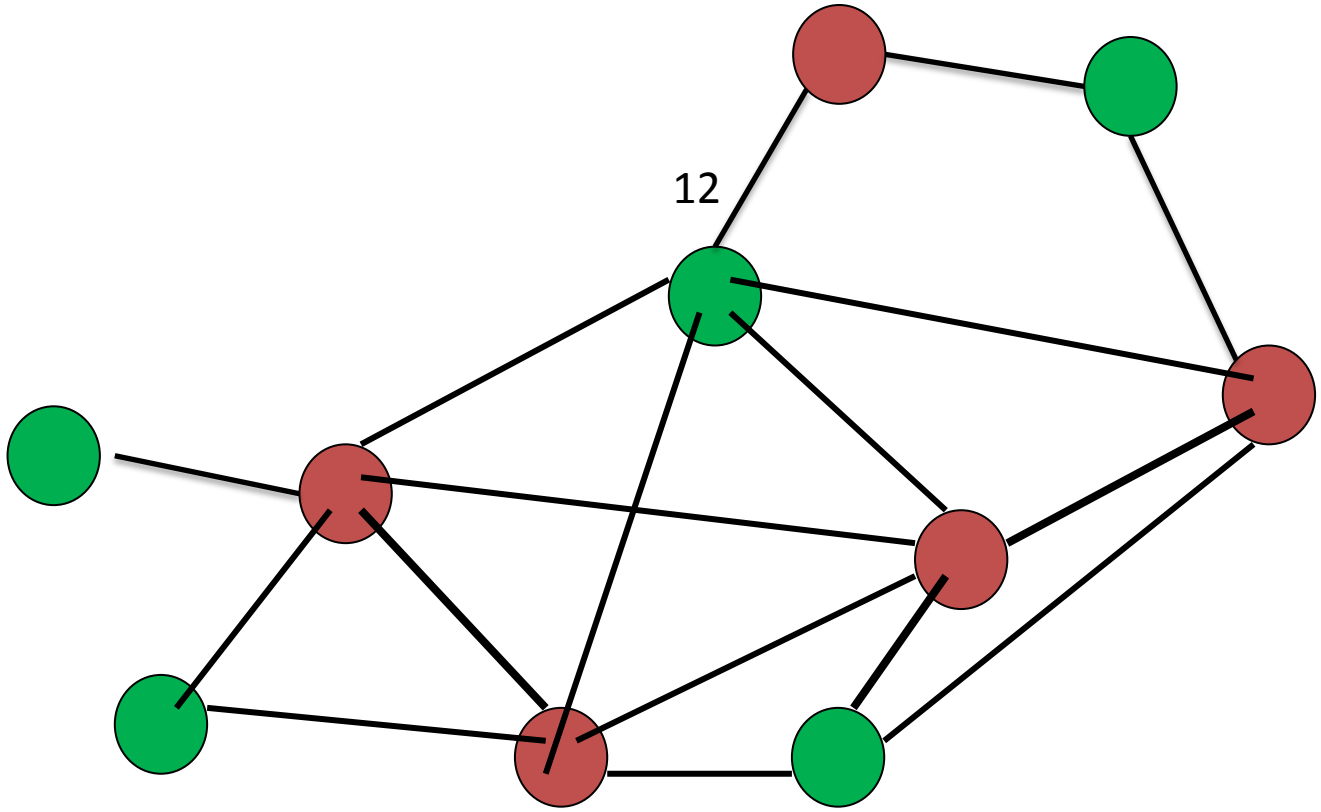
- Undecided (gray) processes choose new IDs.



- Processes that chose 12 and 18 are in.
- Process that chose 7 is out.



- Undecided (gray) process chooses a new ID.



- It’s in.

Properties of Luby's algorithm

- If it ever finishes, it produces a Maximal Independent Set.
- It eventually finishes (with probability 1).
- The expected number of rounds until it finishes is $O(\log n)$.

Independence

- Theorem 1: If Luby's algorithm ever terminates, then the final set S satisfies the independence property.
- Proof:
  - Each node joins S only if it has the unique maximum value in its neighborhood, at some phase.
  - When it does, all its neighbors become inactive.

Maximality

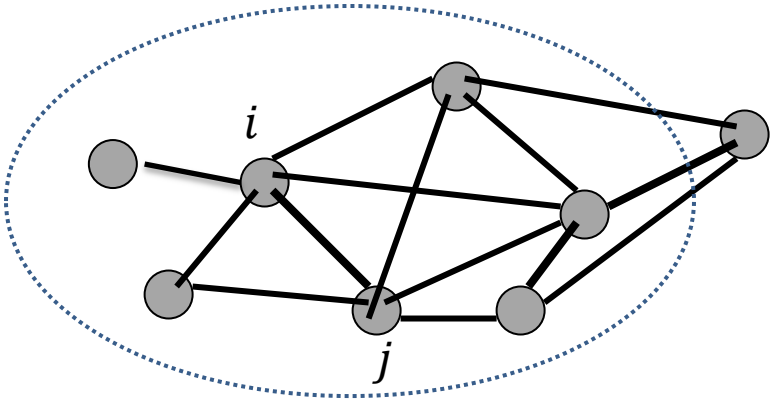
- Theorem 2: If Luby's algorithm ever terminates, then the final set S satisfies the maximality property.
- Proof:
  - A node becomes inactive only if it joins S or a neighbor joins S.
  - We continue until all nodes are inactive.

Termination

- With probability 1, Luby's MIS algorithm eventually terminates.
- Theorem 3: With probability at least $1 - \frac{1}{n}$, all nodes decide within $4 \log n$ phases.
- Proof uses a lemma:
- Lemma 4: With probability at least $1 - \frac{1}{n^2}$, in each phase $1, \ldots, 4 \log n$, all nodes choose different random values.
- So we can essentially pretend that, in each phase, all the random numbers chosen are different.
- Key idea: Show the graph gets sufficiently "smaller" in each phase.
- Lemma 5: For each phase ph, the expected number of edges that are live (connect two active nodes) at the end of the phase is at most half the number that were live at the beginning of the phase.

Termination

- Lemma 5: For each phase ph, the expected number of edges that are live (connect two active nodes) after the phase is at most half the number that were live before the phase.
- Proof:
  - If node i has some neighbor j whose chosen value is greater than those of all of j's neighbors and all of i's other neighbors, then i must become a $loser$ in phase ph.
  - The probability that j chooses such a value is at least $\frac{1}{\deg(i)+\deg(j)}$.
  - Then the probability node i is "killed" by some neighbor in this way is at least $\Sigma_{j\in\Gamma(i)} \frac{1}{\deg(i)+\deg(j)}$.
  - Now consider an undirected edge $\{i, j\}$.



Termination, cont'd

- Proof:
  - Probability i killed $\geq \Sigma_{j \in \Gamma(i)} \frac{1}{\deg(i)+\deg(j)}$.
  - Probability that edge $\{i, j\}$ "dies"

    $\geq \frac{1}{2}$ (probability i killed + probability j killed).
  - So the expected number of edges that die

    $\geq \frac{1}{2} \Sigma_{\{i,j\}}$ (probability i killed + probability j killed).
  - The sum includes the "kill probability" for each node i exactly $\deg(i)$ times.
  - So rewrite the sum as:

    $\frac{1}{2} \Sigma_i \deg(i)$ (probability i killed).
  - Plug in the probability lower bound:

    $\geq \frac{1}{2} \Sigma_i \deg(i) \; \Sigma_{j \in \Gamma(i)} \frac{1}{\deg(i)+\deg(j)}$.

    $= \frac{1}{2} \Sigma_i \Sigma_{j \in \Gamma(i)} \frac{\deg(i)}{\deg(i)+\deg(j)}$.

Termination, cont'd

- Proof:
  - Expected number of edges that die $\geq \frac{1}{2} \Sigma_i \Sigma_{j \in \Gamma(i)} \frac{\deg(i)}{\deg(i)+\deg(j)}$.
  - Write this expression equivalently as a sum over directed edges (i, j):
    $\frac{1}{2} \Sigma_{(i,j)} \frac{\deg(i)}{\deg(i)+\deg(j)}$.
  - Here each undirected edge is counted twice, once for each direction, so this is the same as the following sum over undirected edges $\{i, j\}$.
    $\frac{1}{2} \Sigma_{\{i,j\}} \frac{\deg(i)+\deg(j)}{\deg(i)+\deg(j)} = \frac{1}{2} \Sigma_{\{i,j\}} 1.$
  - This is half the total number of undirected edges, ½ $|E|$, as needed!
- Thus we have:
- Lemma 5: For each phase ph, the expected number of edges that are live (connect two active nodes) at the end of the phase is at most half the number that were live at the beginning of the phase.

Termination, cont'd

- Lemma 5: For each phase ph, the expected number of edges that are live (connect two active nodes) at the end of the phase is at most half the number that were live before the phase.
- Theorem 3: With probability at least $1 - \frac{1}{n}$, all nodes decide within $4 \log n$ phases.
- Proof sketch:
  - Lemma 5 implies that the expected number of edges still live after $4 \log n$ phases is at most $\frac{n^2}{2} \div 2^{4 \log n} = \frac{1}{2n^2}$.
  - Then the probability that *any* edges remain live is $\leq \frac{1}{2n^2}$ (by Markov).
  - The probability that the algorithm doesn't terminate within $4 \log n$ phases $\leq \frac{1}{2n^2} + \frac{1}{n^2} < \frac{1}{\mathrm{n}}$.

Next time

- Maximal Independent Sets, continued
- Graph coloring
- Reading:
  - Peleg book, Chapters 7 and 8
  - Cole-Vishkin paper (optional)
  - Linial paper (optional)