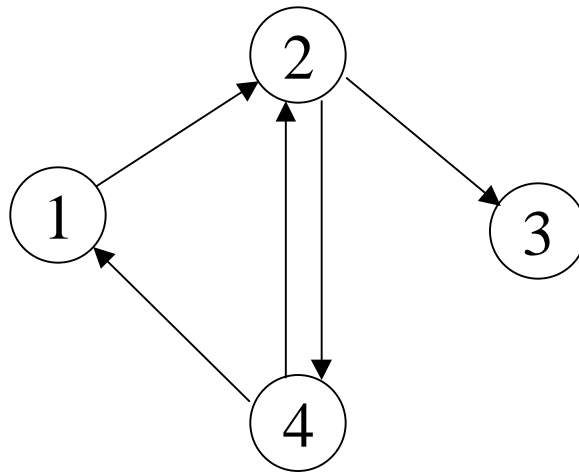


Agenda

- Graph theory notation
- Trees
- Shortest path algorithms
- Distributed, asynchronous algorithms

Notation

- $G = (V, E)$ denotes a graph consisting of a set of vertices V (i.e., nodes) interconnected by a set of edges E (i.e., links).
- An edge $e \in E$ of a directed graph is represented as an *ordered pair* (u, v) , where $u \in V$ and $v \in V$. Here, u is the initial vertex and v is the terminal vertex. For the purposes here, it is assumed that $u \neq v$.
- Example, a *connected*, "unweighted" and *directed* graph.

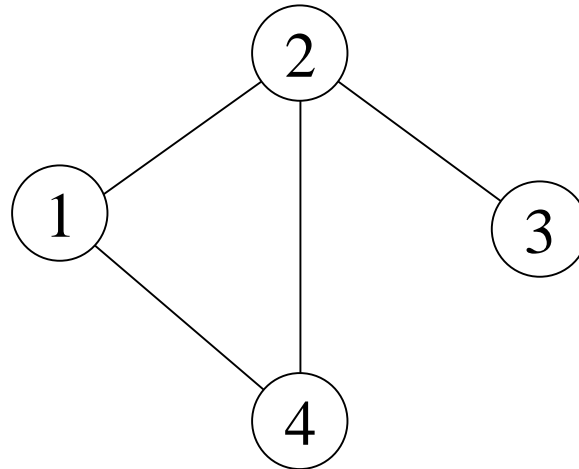


- $V = \{1, 2, 3, 4\}$, $E = \{(1, 2), (2, 3), (2, 4), (4, 1), (4, 2)\}$
- Although E is unweighted, uniform weights might be assigned to the edges. Thus, E may be represented by a weight, or *cost*, matrix W , where $(u, v) \notin E \leftrightarrow w(u, v) = \infty$.

$$W = \begin{bmatrix} \infty & 1 & \infty & \infty \\ \infty & \infty & 1 & 1 \\ \infty & \infty & \infty & \infty \\ 1 & 1 & \infty & \infty \end{bmatrix}$$

Notation (cont.)

- Example, a connected, unweighted and *undirected* graph.



- $V = \{1,2,3,4\}$
- $E = \{(1,2),(1,4),(2,3),(2,4)\} \leftrightarrow \{(2,1),(4,1),(3,2),(4,2)\}$
- An edge $e \in E$ of an undirected graph can be represented as an *unordered pair* $(u,v) = (v,u)$.
- Although G is undirected, each undirected edge $e = (u,v)$, is equivalent to a pair of directed edges (u,v) and (v,u) . A symmetric weight matrix may, therefore, also represent an unweighted and undirected edge set, E .

$$W = \begin{bmatrix} \infty & 1 & \infty & 1 \\ 1 & \infty & 1 & 1 \\ \infty & 1 & \infty & \infty \\ 1 & 1 & \infty & \infty \end{bmatrix}$$

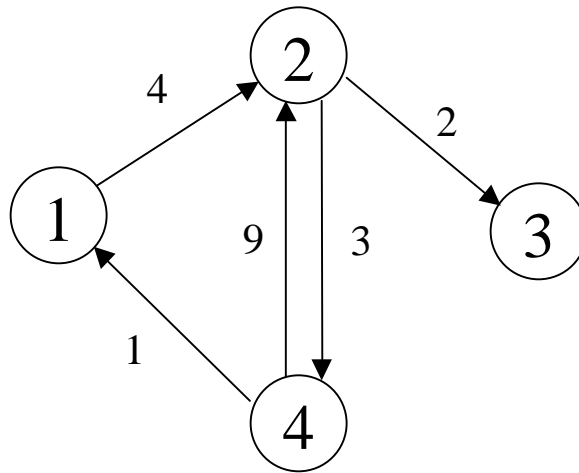
Q: Ignoring edges that form loops, how many distinct edges can there possibly be in an *undirected* graph?

Notation (Continued)

A: For an undirected graph with no loop edges, $|E| \leq \frac{|V| \cdot (|V| - 1)}{2}$.

Where, $|S| \equiv$ Cardinality of the set S .

- Example: Connected, directed graph with *non-uniform* costs.



- $V = \{1,2,3,4\}$
- $E = \{(1,2),(2,3),(2,4),(4,1),(4,2)\}$
- E can also be conveniently expressed in terms of a weight matrix.

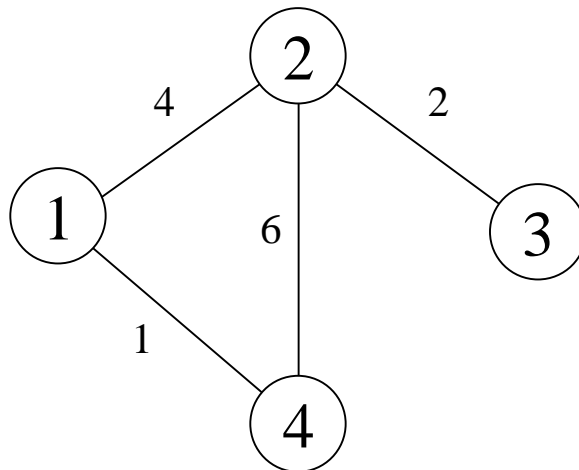
$$W = \begin{bmatrix} \infty & 4 & \infty & \infty \\ \infty & \infty & 2 & 3 \\ \infty & \infty & \infty & \infty \\ 1 & 9 & \infty & \infty \end{bmatrix}$$

Q: Why might link costs be asymmetric?

Notation (continued)

A: Traffic volume might be asymmetric between a pair of nodes, thereby, resulting in an asymmetric cost metric. Or, a policy metric may be in effect that sets asymmetric costs.

- Example: Connected, undirected graph with non-uniform edge costs.



- $V = \{1,2,3,4\}$
- $E = \{(1,2),(2,3),(2,4),(4,1)\}$
- The symmetric weight matrix associated with E is as follows.

$$W = \begin{bmatrix} \infty & 4 & \infty & 1 \\ 4 & \infty & 2 & 6 \\ \infty & 2 & \infty & \infty \\ 1 & 6 & \infty & \infty \end{bmatrix}$$

Notation (continued)

- Other definitions for *undirected* graphs [Bertsekas and Gallager, p. 387]:

- *Walk* \equiv An ordered sequence of nodes (v_1, v_2, \dots, v_L) such that $\{(v_1, v_2), (v_2, v_3), \dots, (v_{L-1}, v_L)\} \subseteq E$.

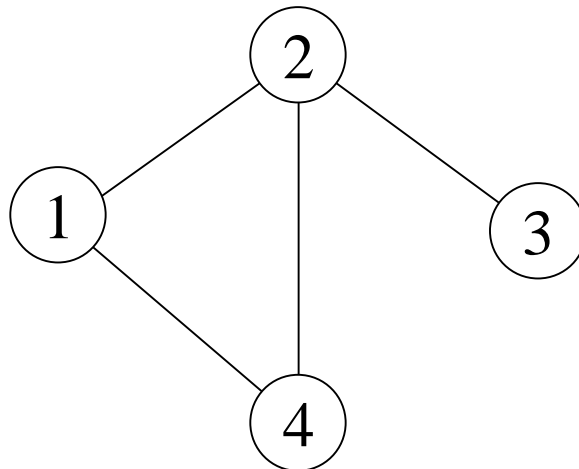
E.g., (3,2,4,2) is a walk.

- *Path* \equiv A walk with no repeated nodes.

E.g., (3,2,4,1) is a path

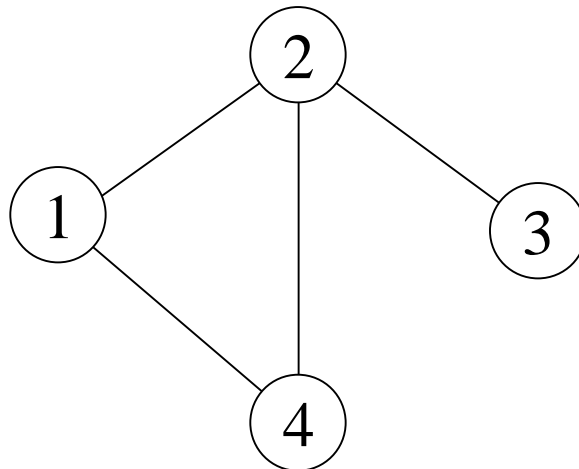
- *Cycle* \equiv A walk (v_1, v_2, \dots, v_L) where $v_1 = v_L$ with no other nodes repeated and $L > 3$.

E.g., (4,1,2,4) is a cycle.



Trees

- Definition: A graph $G = (V, E)$ is a tree if and only if G is connected and $|E| = |V| - 1$.
- Definition: A tree (T) is said to *span* $G = (V, E)$ if $T = (V, E')$ and $E' \subseteq E$.
- Constructing a spanning tree, without any additional requirements, is easily achieved in polynomial time.
- E.g., following the simple algorithm from [Bertsekas and Gallager, p. 388]. Initially, $G' = (V', E')$ where $V' = \emptyset = E'$.



- **Step 1:** Let $V' = \{4\}$. $E' = \emptyset$. **Step 2:** $V' \neq V \rightarrow$ Continue.
- **Step 3:** Add edge $(2,4)$ to $E' \rightarrow V' = \{2,4\}$, $E' = \{(2,4)\}$.
Step 2: $V' \neq V \rightarrow$ Continue.
- **Step 3:** Add edge $(1,4)$ to $E' \rightarrow V' = \{1,2,4\}$, $E' = \{(1,4), (2,4)\}$. **Step 2:** $V' \neq V \rightarrow$ Continue.
- **Step 3:** Add edge $(2,3)$ to $E' \rightarrow V' = \{1,2,3,4\}$, $E' = \{(1,4), (2,4), (2,3)\}$. **Step 2:** $V' = V \rightarrow$ STOP.

Q: What are two key ideas of a proof to validate this algorithm?

Trees (continued)

A: G is connected and G' after each iteration is tree. The connectivity of G ensures that $\forall S \subset G, S \neq \emptyset, S$ is connected to $G-S$. The fact that each iteration of the algorithm yields a tree allows a proof by induction.

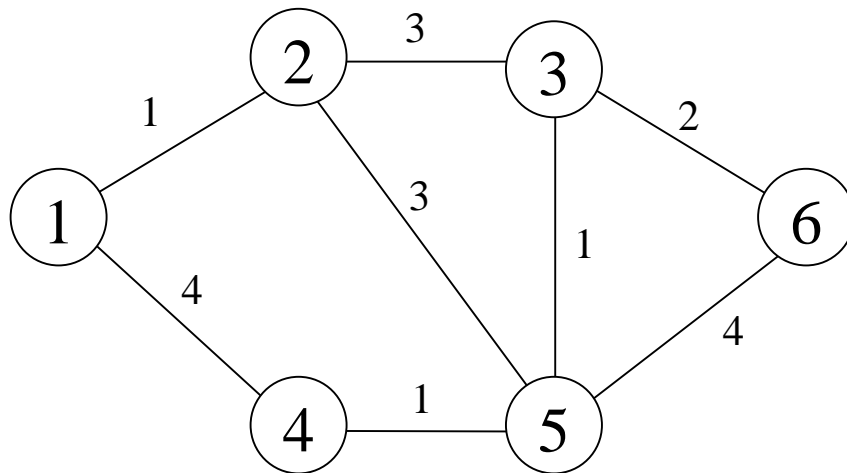
- A *minimum weight spanning tree* (MST) for G is a spanning tree such that the sum of its edge weights is minimum among all spanning trees of G .
- A solution for the MST problem can be obtained via a so-called greedy algorithm.
- Prim's algorithm:
 - Select an arbitrary node as the initial tree (T).
 - Enlarge T in an iterative fashion by adding the outgoing edge (u,v) , (i.e., $u \in T$ and $v \in G-T$) with minimum cost (i.e., weight).
 - The algorithm stops after $|V|-1$ iterations.
 - Computational complexity = $O(|V|^2)$ [Jungnickel].
- Kruskal's algorithm:
 - Select the edge $e \in E$ of minimum weight $\rightarrow E' = \{e\}$.
 - Continue to add the edge $e \in E-E'$ of minimum weight that, when added to E' , does *not* form a cycle.
 - Computational complexity = $O(|E| \times \log |E|)$ [Jungnickel].

Q: How can the MST problem be converted to a problem solvable by a greedy algorithm?

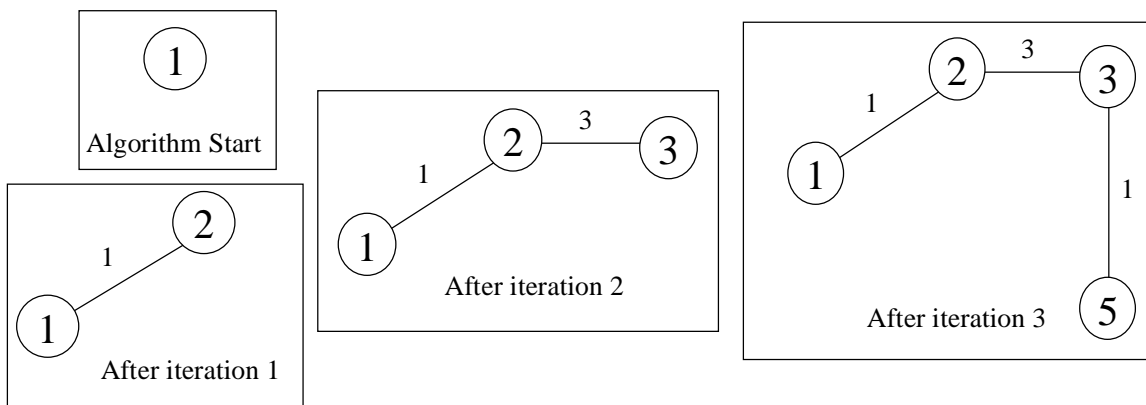
Trees (continued)

A: Multiply edge weights by -1 . Then modify Prim's or Kruskal's algorithm to obtain a *maximum* weight spanning tree.

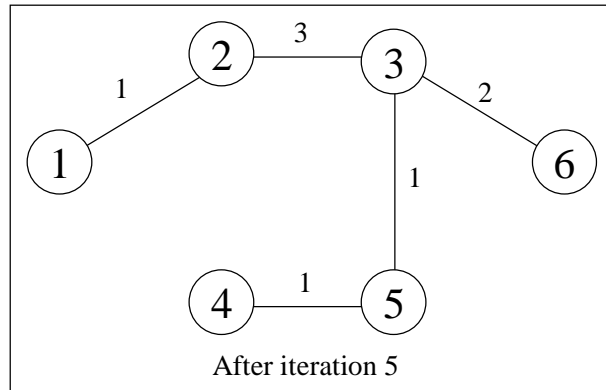
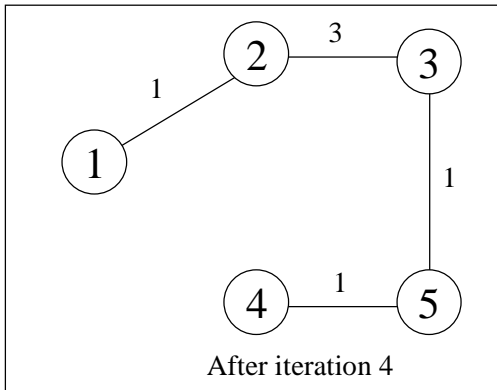
- E.g., [Bertsekas and Gallager, Fig. 5.35].



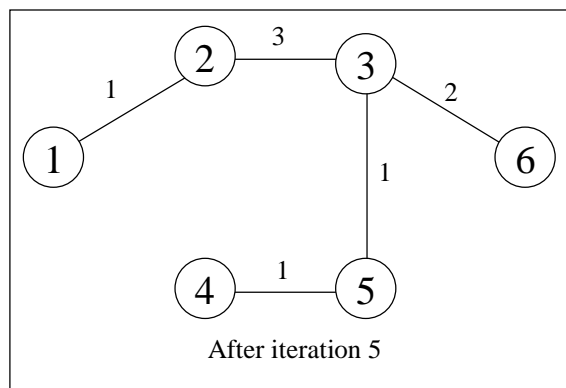
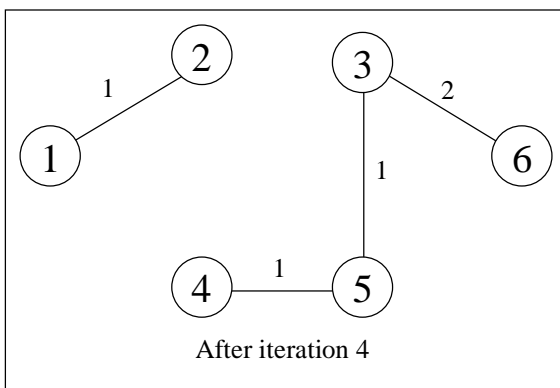
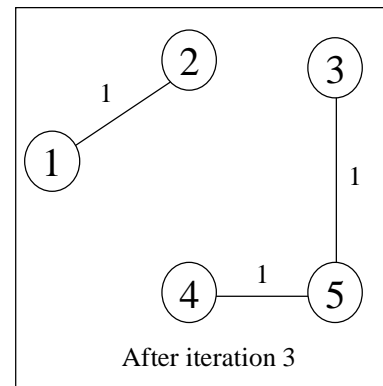
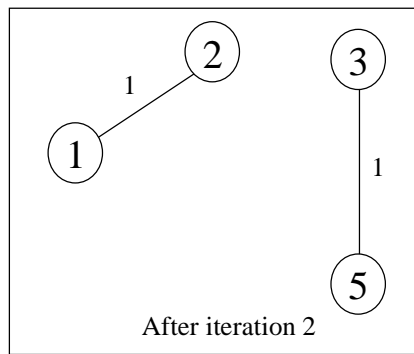
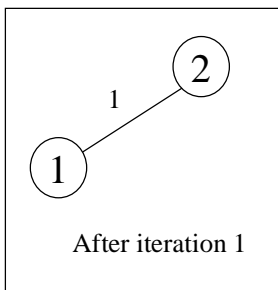
– Solution via Prim's algorithm:



Trees (continued)



– Solution via Kruskal's algorithm:



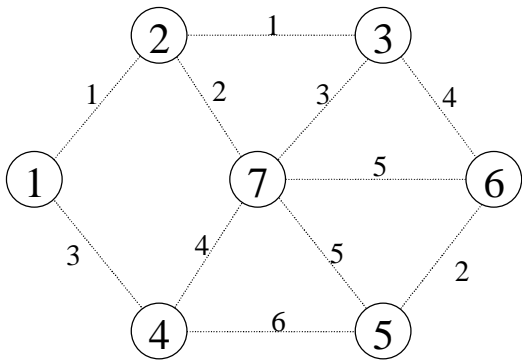
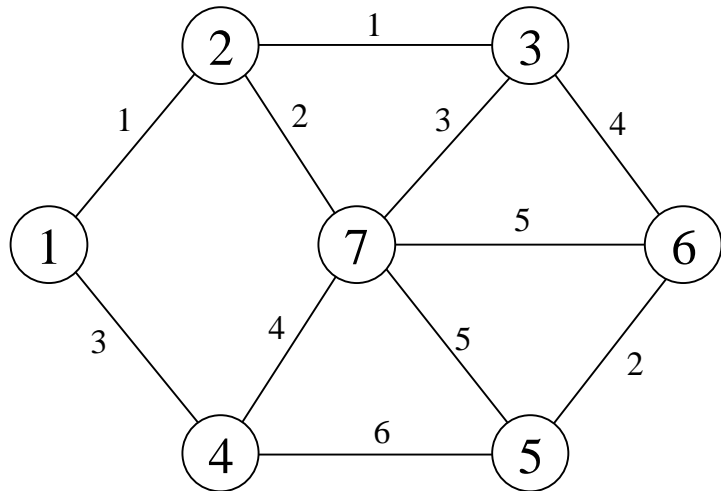
Trees (Continued)

- Distributed MST algorithms include the algorithm proposed in [Gallager, Humblet and Spira].
 - Advantage of this distributed approach is that a MST can be formed without each node having a complete topology map.
 - Fragments of a MST are created in a distributed fashion. Initially, G consists of $|V|$ fragments. Each fragment selects its minimum weight outgoing link, and via *control messaging*, each fragment arranges to merge with a neighboring fragment over its minimum weight outgoing link.
 - The algorithm is shown to produce a MST in $O(|V| \times \log |V|)$ time provided the edge weights are unique.
 - Algorithm can work even when \exists non-unique edge weights by using node IDs to "break ties" between edges with equal weight.
 - The algorithm requires only $O(|V| \times \log |V| + |E|)$ message overhead.

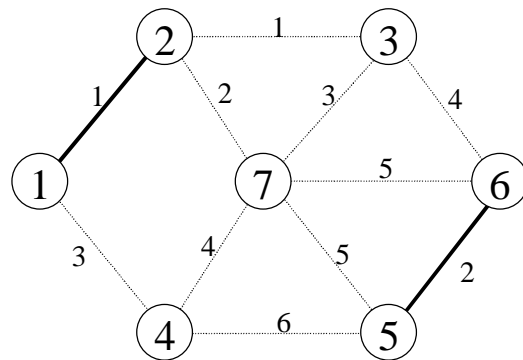
Note: $|E| = \frac{1}{2} \cdot d(G) \cdot |V|$, where $d(G)$ is the *average degree* (i.e., the average number links incident at an arbitrary node) of the network [Diestal].

Trees (continued)

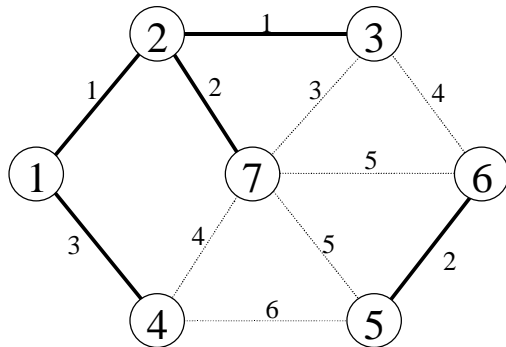
- Distributed MST construction example:



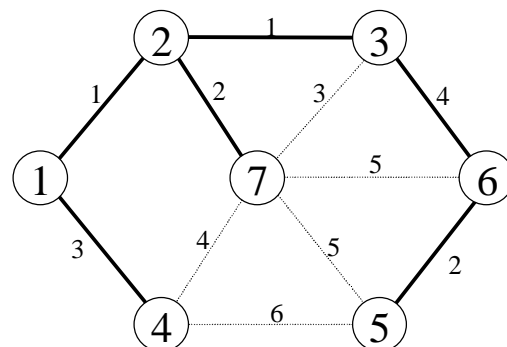
Initial State (V zero-level fragments)



1st level fragments $\{1,2\}$ and $\{5,6\}$ formed



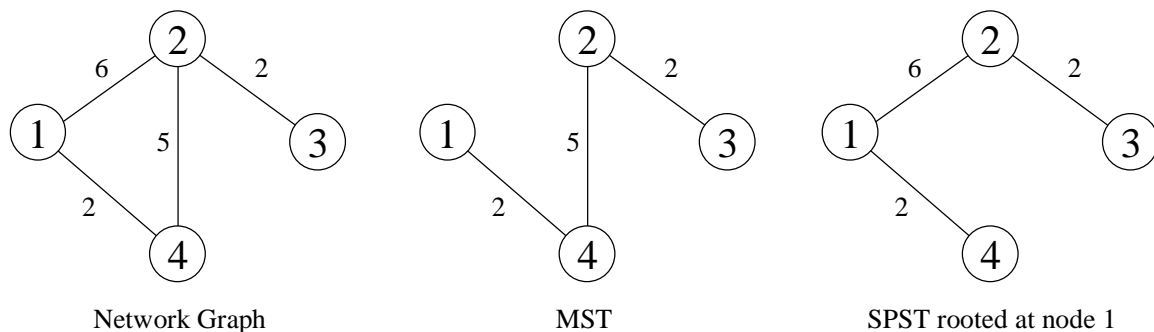
Nodes 3, 4 and 7 join fragment $\{1,2\}$



Fragments $\{1,2,3,4,7\}$ and $\{5,6\}$ join to form 2nd level fragment that is also the MST.

Trees (continued)

- A shortest path spanning tree (SPST), T , is a spanning tree *rooted* at a particular node, say node s , such that the $|V|-1$ minimum weight paths from s to each of the other network nodes is contained in T .
 - As will be addressed in the discussion of shortest path algorithms, the SPST can be constructed in polynomial time.
 - Note, the SPST is not necessarily the same as the MST. E.g.,



- Other trees [Jungnickel]:
 - *Maximum weight* spanning tree.
 - Most uniform spanning tree.
 - Minimum connected dominating set (MCDS) → A spanning tree where the number of *leaf nodes* has been maximized.
 - Bounded diameter spanning tree.
 - Degree constrained spanning tree.

Aside: The MCDS, bounded diameter spanning tree and degree constrained spanning tree are NP-complete problems. [Jungnickel]

Q: What are some of the network applications of trees?

Trees (continued)

A: Applications of trees include the following.

- Multicast routing (1-to-some communications).
- 1-to-1 (unicast) communications → SPST.
- 1-to-all communications for wired networks → MST.
- Maximum probability of reliable 1-to-all communications → Maximum weight spanning tree.
- 1-to-all communications in broadcast-based packet radio networks → MCDS.
- Load balancing → Degree constrained spanning tree.

Shortest Path Algorithms

- Include algorithms that effectively construct a SPST rooted at a given node (e.g., Dijkstra and Bellman-Ford algorithms) as well as algorithms that find the shortest path between all pairs of nodes (e.g., Floyd-Warshall).
- Dijkstra's algorithm:
 - Non-negative edge weights are assumed.
 - Effectively constructs a SPST rooted at an arbitrary node s .

Procedure *Dijkstra-like*($G, W, s; G', D, P$)

$V' = \{s\}; U = V - \{s\};$

$E' = \emptyset;$

For $v \in U$ **do**

$D_v = w(s, v);$

$P_v = s;$

EndFor

While $U \neq \emptyset$ **do**

Find $v \in U$ such that D_v is minimal;

$V' = V' \cup \{v\}; U = U - \{v\};$

$E' = E' \cup (P_v, v);$

For $x \in U$ **do**

If $D_v + w(v, x) < D_x$ **then**

$D_x = D_v + w(v, x);$

$P_x = v;$

EndIf

EndFor

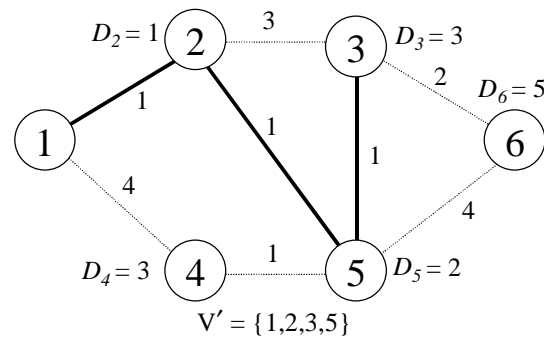
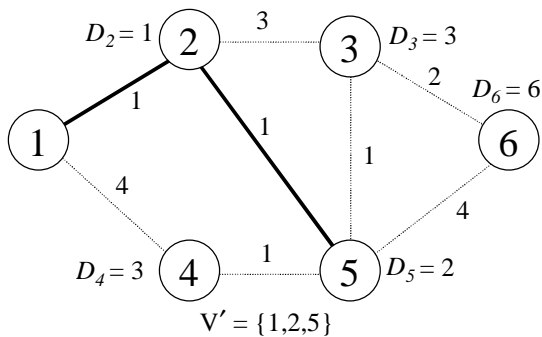
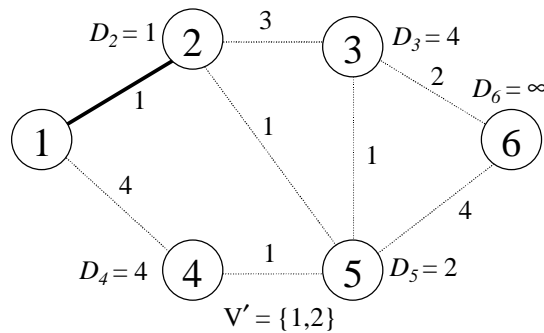
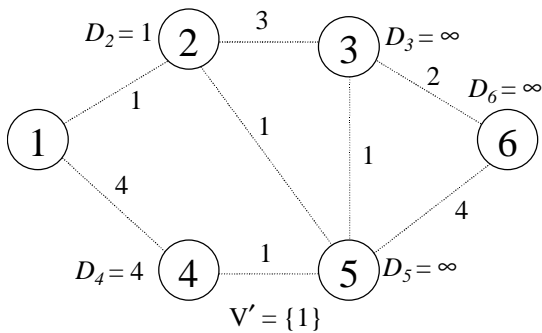
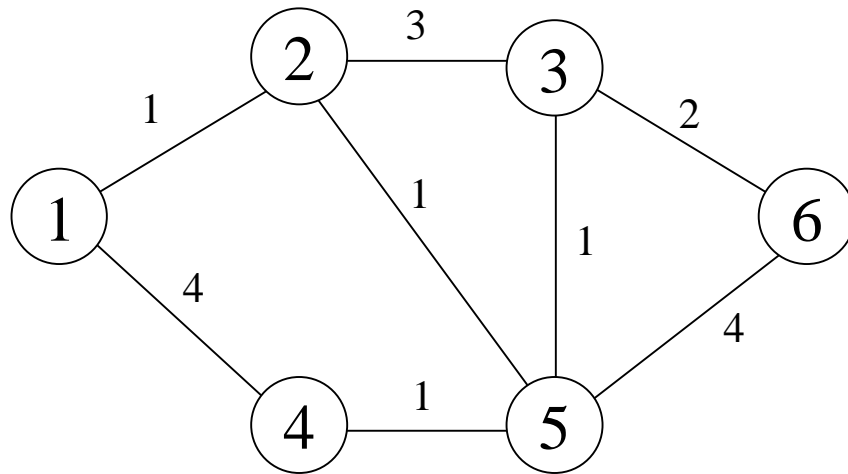
EndWhile

- Computational cost is $O(|V|^2)$.

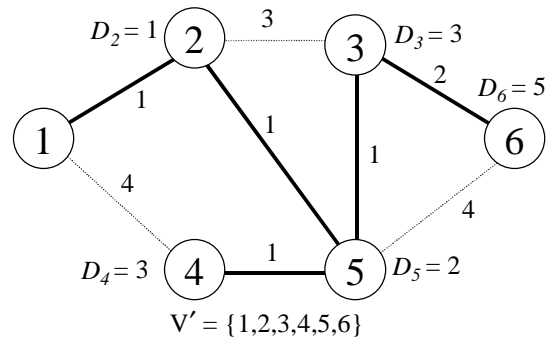
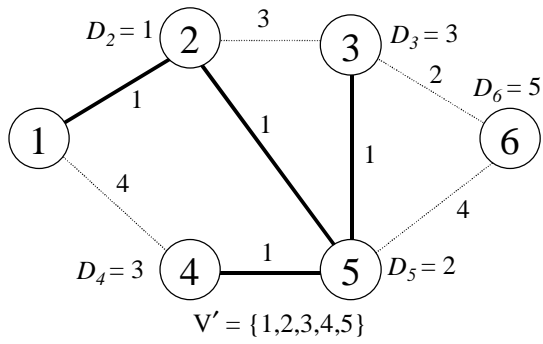
Shortest Path Algorithms (continued)

Note: For the sake of brevity here, unless otherwise noted in an algorithm's description, $D_v \equiv D_{s,v}$ (i.e., the distance from node s to node v).

- Example [Bertsekas and Gallager, Fig. 5.35] with $s = 1$.



Shortest Path Algorithms (continued)



Shortest Path Algorithms (continued)

- **Bellman-Ford algorithm**

- Finds the shortest *walk* from a source node s to an arbitrary destination node v subject to the constraints that the walk consists of at most h hops and goes through node v only once.

Procedure *Bellman-Ford-like*($G, W, s; D$)

$$D_v^{-1} = \infty \quad \forall v \in V.$$

$$D_s^0 = 0 \quad \text{and} \quad D_v^0 = \infty \quad \forall v \neq s, v \in V.$$

$$h = 0$$

Until ($D_v^h = D_v^{h-1} \quad \forall v \in V$) **or** ($h = |V|$) **do**

$$h = h + 1$$

For $v \in V$ **do** $D_v^{h+1} = \min_{u \in V} \{D_u^h + w(u, v)\}$ **EndFor**

EndUntil

- Note, the shortest (multi-hop) path from s to v contains as a last hop, an edge (u, v) . Removing the edge (u, v) from the (directed) shortest path between s and v yields the shortest path from s to u . [Jungnickel]
- In the worst case, the above simplistic implementation has $O(|V|^3)$ complexity.

Q: In [Bertsekas and Gallager, p. 399] it is claimed that the computational cost of the Bellman-Ford algorithm is $O(h_{max} \times |E|)$ where h_{max} is the maximum number hops contained in a shortest path. How might the above pseudo code be modified to achieve this improved performance?

Q: Why might the above procedure fail to terminate after $|V|$ iterations of the **Until** loop?

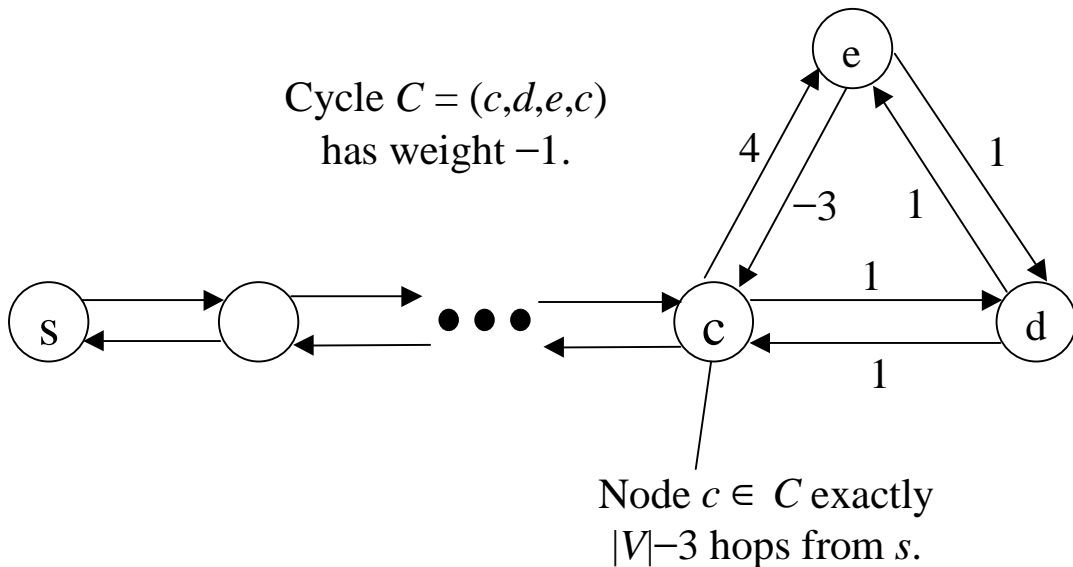
Shortest Path Algorithms (continued)

A: First, by defining for each node v an adjacency list, $A(v)$, the computational cost of the operation within the **For** loop can be reduced, *on average*, from $O(|V|)$ to $O(|E|/|V|)$, by replacing

$$D_v^{h+1} = \min_{u \in V} \{D_u^h + w(u, v)\} \text{ with } D_v^{h+1} = \min_{u \in A(v)} \{D_u^h + w(u, v)\}.$$

Thus, the total cost of the **For** loop is reduce from $O(|V|^2)$ to $O(|E|)$. Second, a more *precise* assessment than $O(|V|)$ for the number of iterations of the **Until** loop is h_{max} . Combining these two points yields a computational cost of $O(h_{max} \times |E|) \leq O(|V| \times |E|) \leq O(|V|^3)$.

A: If G contains a cycle C of negative length, such that at least one node in C lies not more than $|V|-|C|$ hops from s , then for at least one node $c \in C$, $D_c^{|V|} \neq D_c^{|V|-1}$. For example:



- Fortunately, for communication networks, non-negative edge weights are typically assumed.

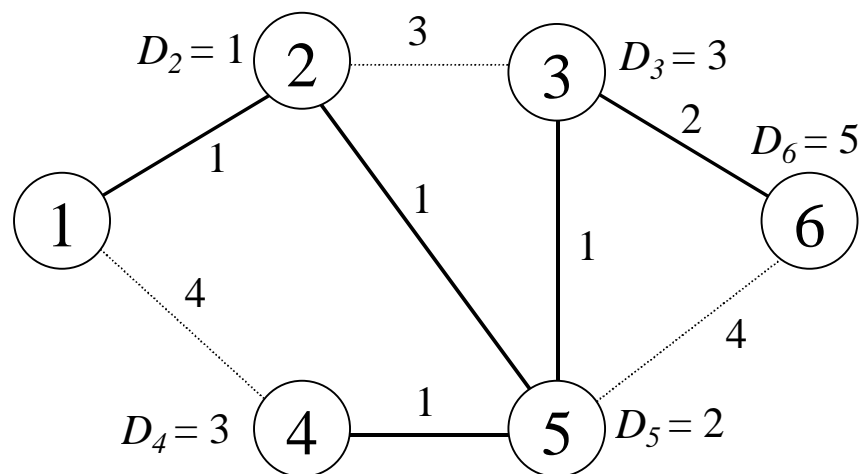
Shortest Path Algorithms (continued)

Bellman's equation and shortest path construction

$$D_s = 0$$

$$D_v = \min_{u \in V} \{w(u, v) + D_u\}, \forall v \in V - \{s\}.$$

Find shortest paths, given D , provided all cycles have non-negative weight. E.g., solution to [Bertsekas and Gallager, Fig. 5.35].



$$D_1 = 0$$

$$D_2 = 1 = \min\{w(1,2) + D_1, w(3,2) + D_3, w(5,2) + D_5\}$$

$$D_3 = 3 = \min\{w(2,3) + D_2, w(5,3) + D_5, w(6,3) + D_6\}$$

$$D_4 = 3 = \min\{w(1,4) + D_1, w(5,4) + D_5\}$$

$$D_5 = 2 = \min\{w(2,5) + D_2, w(3,5) + D_3, w(4,5) + D_4, w(6,5) + D_6\}$$

$$D_6 = 5 = \min\{w(3,6) + D_3, w(5,6) + D_5\}$$

$$D_1 = 0$$

$$D_2 = 1 = \min\{w(1,2), w(3,2) + 3, w(5,2) + 2\} = w(1,2)$$

$$D_3 = 3 = \min\{w(2,3) + 1, w(5,3) + 2, w(6,3) + 5\} = w(5,3) + 2$$

$$D_4 = 3 = \min\{w(1,4), w(5,4) + 2\} = w(5,4) + 2$$

$$D_5 = 2 = \min\{w(2,5) + 1, w(3,5) + 3, w(4,5) + 3, w(6,5) + 5\}$$

$$= w(2,5) + 1$$

$$D_6 = 5 = \min\{w(3,6) + 3, w(5,6) + 2\} = w(3,6) + 3$$

Shortest Path Algorithms (continued)

- Floyd-Warshall algorithm
 - Finds shortest path between *all ordered* pairs of nodes (s,v) , $\{s,v\} \in V$.
 - Let $V = \{1,2,\dots,|V|\}$.
 - Each iteration of Floyd-Warshall yields the shortest path weights between all pairs of nodes under the constraint that only nodes $\{1,2,\dots,n\}$, $n \leq |V|$, can be used as intermediary nodes on the shortest paths.

Procedure *Floyd-Warshall-like*($G,W;D$)

$D = W;$

For $u = 1$ to $|V|$ **do**

For $s = 1$ to $|V|$ **do**

For $v = 1$ to $|V|$ **do**

$D_{s,v} = \min\{D_{s,v}, D_{s,u} + W_{u,v}\}$

EndFor

EndFor

EndFor

Note: $W_{u,v}$ corresponds to the (u,v) entry of the weight matrix, W .

- Clearly, Floyd-Warshall completes in $O(|V|^3)$ time.

Q: What advantage might the Floyd-Warshall algorithm have over simply computing Dijkstra's algorithm for every node in V ?

Q: Besides for the purpose of centralized routing, why might it be useful to compute the shortest paths between all pairs of nodes?

Shortest Path Algorithms (continued)

A: Suppose there is a critical node, $x \in V$, for which it is desired to know both how its presence (i.e., node okay) and absence (e.g., node failure) affects network shortest paths. Let $x = |V|$. Thus, completion of $|V|-1$ iterations of the outermost **For** loop corresponds to the shortest paths when x is out of order and the completion of $|V|$ iterations of the loop corresponds to when x is operational.

A: Computation of D (distances between all pairs of nodes) can be used to find the *center* of the network. That is, node v such that:

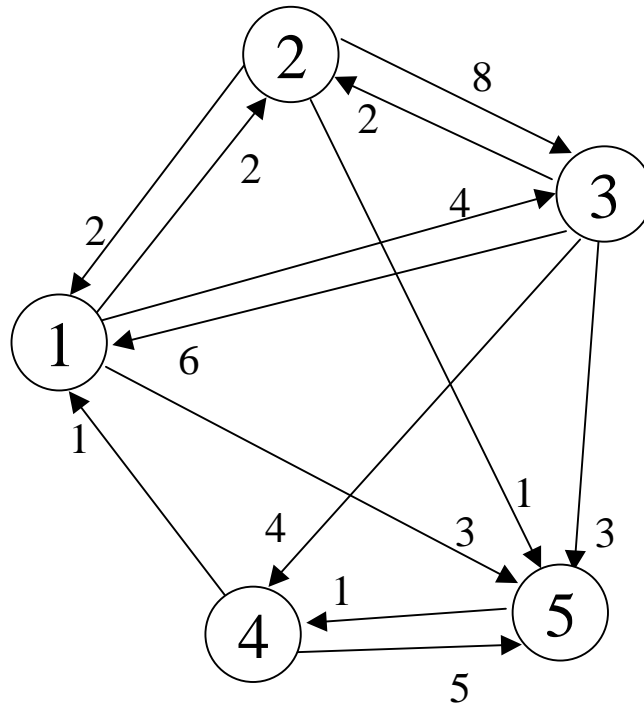
$$v = \min_{v \in V} \left\{ \max_{u \in V} \{D_{u,v}\} \right\}$$

Further, by setting the edge weights for all directly connected nodes to unity, the network diameter, Δ , can be computed:

$$\Delta = \max_{\{u,v\} \in V} \{D_{u,v}\}$$

Shortest Path Algorithms (continued)

Floyd-Warshall example:



$$\text{Start: } D = \begin{bmatrix} 0 & 2 & 4 & \infty & 3 \\ 2 & 0 & 8 & \infty & 1 \\ 6 & 2 & 0 & 4 & 3 \\ 1 & \infty & \infty & 0 & 5 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix} \rightarrow D = \begin{bmatrix} 0 & 2 & 4 & \infty & 3 \\ 2 & 0 & 6 & \infty & 1 \\ 6 & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

$$\rightarrow D = \begin{bmatrix} 0 & 2 & 4 & \infty & 3 \\ 2 & 0 & 6 & \infty & 1 \\ 4 & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix} \rightarrow D = \begin{bmatrix} 0 & 2 & 4 & 8 & 3 \\ 2 & 0 & 6 & 10 & 1 \\ 4 & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

Shortest Path Algorithms (continued)

$$\rightarrow D = \begin{bmatrix} 0 & 2 & 4 & 8 & 3 \\ 2 & 0 & 6 & 10 & 1 \\ 4 & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ 2 & 4 & 6 & 1 & 0 \end{bmatrix} \rightarrow D = \begin{bmatrix} 0 & 2 & 4 & 4 & 3 \\ 2 & 0 & 6 & 2 & 1 \\ 4 & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ 2 & 4 & 6 & 1 & 0 \end{bmatrix} \text{ Finish.}$$

Distributed Asynchronous Shortest Path Techniques

- *Distributed* in the sense that *each node* computes the shortest path *weights* to every network node.
 - That is, no centralized computation is required.
 - As in the distributed MST algorithm of [Gallager, Humblet and Spira], control messaging is required to support the distributed computation.
- *Asynchronous* in the sense that neither the distributed computation performed at each node, or the control messaging between nodes, require inter-node synchronization.
- Distributed shortest path computation via Dijkstra's algorithm.
 - No change to the algorithm structure is required.
 - Control messaging involves each node originating a link state packet (LSP) that is flooded throughout the network. → Up to $O(|V| \times |E|)$ LSP transmission overhead, on average, per LSP update period.
 - Entire topology map must be maintained at each node. → Requires data structure of $O(|E|)$ size.
 - Flooding of updated LSP upon link state change allows for timely dissemination of topology information. Thus, each node typically has an accurate topology map, which affords correct shortest computation at each node.

Distributed Asynchronous Techniques (continued)

- Distributed Bellman-Ford algorithm
 - In [Bertsekas and Gallager, Section 5.2.4], the distributed Bellman-Ford algorithm is shown to converge to the shortest path weights in networks that satisfy the following conditions.

C-1: As before, G contains only cycles of *non-negative* length.

C-2: If $(u,v) \in E$ then so is (v,u) . $\rightarrow G$ is *strongly connected*.

C-3: Although D may initially be arbitrary, $w(u,v) \forall (u,v) \in E$ is *fixed*.

- A modest change to the notation of Bellman's equation is helpful:

$$D_{s,s} = 0$$

$$D_{s,v} = \min_{u \in N(s)} \{w(s,u) + D_{u,v}\}, \forall v \in V - \{s\}.$$

$N(s) \equiv$ Neighborhood of s , the set of nodes lying exactly one hop away from s . $\rightarrow \forall u \in N(s), (s,u) \in E$.

Two subscripts are included here for D to distinguish $D_{s,v}$ from $D_{u,v}$.

- From the above form of Bellman's equation, it is evident that each node need not know the complete topology map. Each node only knows the weights of the links that are incident to it, the identity of all network nodes and estimates (received from its neighbors) of the distances to all network nodes. $\rightarrow O(d(G) \times |V|)$ storage overhead, on average.

Distributed Asynchronous Techniques (continued)

- Messaging consists of each node s transmitting to its neighbors a copy of its *distance vector*, $D_{s,v}$. That is, a current list of the shortest path distances it has computed, for *every* network node. Similarly, each neighboring node $u \in N(s)$ transmits to s a copy of its distance vector, $D_{u,v}$.
- A simplified overview of some possible events at a node s , for the distributed Bellman-Ford algorithm, is as follows.
 - Node s updates $D_{s,v}$, $\forall v \in V - \{s\}$ in accordance with:

$$D_{s,v} = \min_{u \in N(s)} \{w(s,u) + D_{u,v}\}, \forall v \in V - \{s\}$$
 If any of the updates change the distance vector at s , s sends an updated version of $D_{s,v}$ to each of its neighbors.
 - Node s receives a copy of a distance vector from one or more of its neighbors. \rightarrow Update $D_{s,v}$.
 - An internal trigger at s (e.g., a periodic timer) prompts s to recompute $D_{s,v}$, or to transmit a copy of $D_{s,v}$ to each of its neighbors, or both.

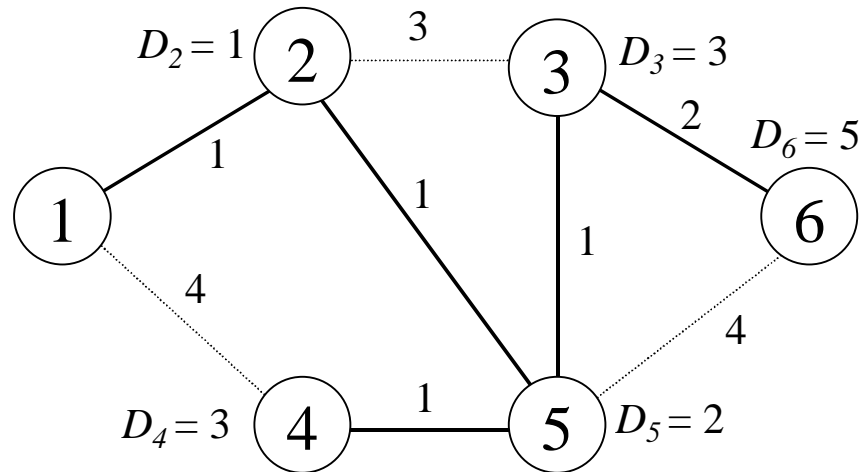
Q: Once a distance vector is computed at s , how can this information be used to make packet forwarding decisions?

A: Let $u_v \in N(s)$ be the node to which s forwards packets that are destined for v . Determine the appropriate outgoing edge that solves Bellman's equation for node v . Node u_v is the terminal vertex of this outgoing edge. That is:

$$u_v = u \text{ such that } D_{s,v} = w(s,u) + D_{u,v}.$$

Distributed Asynchronous Techniques (continued)

- Example of packet forwarding table generation. [Bertsekas and Gallager, Fig. 5.35] revisited.



$$D_{1,1} = 0$$

$$D_{1,2} = 1 = \min\{w(1,2) + D_{2,2}, w(1,4) + D_{4,2}\}$$

$$D_{1,3} = 3 = \min\{w(1,2) + D_{2,3}, w(1,4) + D_{4,3}\}$$

$$D_{1,4} = 3 = \min\{w(1,2) + D_{2,4}, w(1,4) + D_{4,4}\}$$

$$D_{1,5} = 2 = \min\{w(1,2) + D_{2,5}, w(1,4) + D_{4,5}\}$$

$$D_{1,6} = 5 = \min\{w(1,2) + D_{2,6}, w(1,4) + D_{4,6}\}$$

$$D_1 = 0$$

$$D_2 = 1 = \min\{w(1,2), w(1,4) + 2\} = w(1,2)$$

$$D_3 = 3 = \min\{w(1,2) + 2, w(1,4) + 2\} = w(1,2) + 2$$

$$D_4 = 3 = \min\{w(1,2) + 2, w(1,4)\} = w(1,2) + 2$$

$$D_5 = 2 = \min\{w(1,2) + 1, w(1,4) + 1\} = w(1,2) + 1$$

$$D_6 = 5 = \min\{w(1,2) + 4, w(1,4) + 4\} = w(1,2) + 4$$

Thus, node 1 forwards all outgoing packets to node 2.

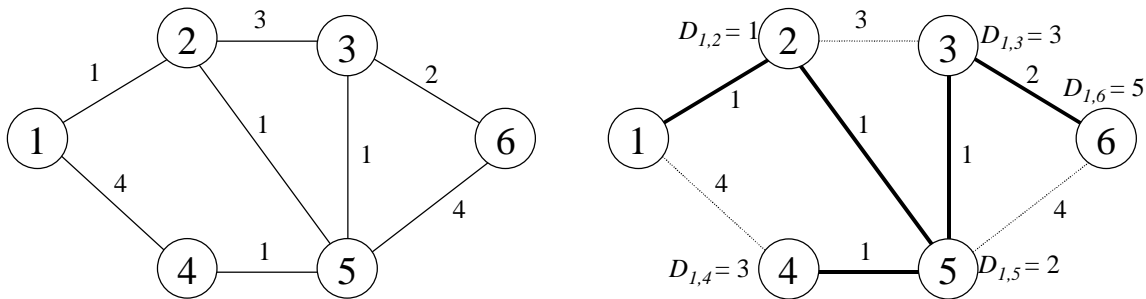
Q: How does the distributed Bellman-Ford algorithm compare with the distributed implementation of Dijkstra's algorithm?

Distributed Asynchronous Techniques (continued)

A: Distributed implementation of Dijkstra's algorithm requires complete topology knowledge. A distributed Bellman-Ford implementation does not.

However, distributed implementation of Dijkstra's algorithm affords computation of other useful structures such as trees for multicast routing.

Further, the distributed Bellman-Ford algorithm may converge to the correct shortest path weights much slower than a distributed implementation of the Dijkstra algorithm. E.g.



In a distributed implementation of Dijkstra's algorithm, each node floods its LSP. Thus, node 1 quickly acquires the topology information needed to compute the shortest path to node 6.

In the distributed Bellman-Ford implementation, the following events must occur before node 1 can compute the weight of the shortest path to node 6.

- 1) Node 3 communicates $D_{3,6} = 2$ to node 5.
- 2) Node 5 updates $D_{5,6} \leftarrow 3$.
- 3) Node 5 sends $D_{3,6} = 3$ to node 2.
- 4) Node 2 updates $D_{2,6} \leftarrow 4$.
- 5) Node 2 sends $D_{2,6} = 4$ to node 1.

References

- [Bertsekas and Gallager] D. Bertsekas and R. Gallager, *Data Networks*, Prentice-Hall, Inc., 2nd ed., 1992.
- [Diestel] R. Diestel, *Graph Theory*, Springer-Verlag New York, Inc., 1997.
- [Gallager, Humblet and Spira] R. G. Gallager, P. A. Humblet and P. M. Spira, "A distributed algorithm for minimum-weight spanning trees," *ACM Trans. Programm. Lang.*, Vol. 5, No. 1, January 1983, pp. 66-77.
- [Jungnickel] D. Jungnickel, *Graphs, Networks and Algorithms*, Springer-Verlag Berlin Heidelberg, 1999.