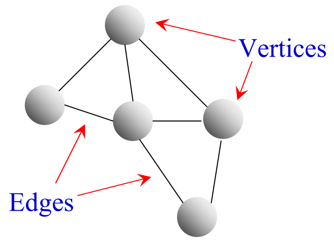
Graph Theory

Some applications of Graph Theory

- Models for communications and electrical networks
- Models for computer architectures
- Network optimization models for operations analysis, including scheduling and job assignment
- Analysis of Finite State Machines
- Parsing and code optimization in compilers

Application to Ad Hoc Networking

- Networks can be represented by graphs
- The mobile nodes are vertices
- The communication links are edges



- Routing protocols often use shortest path algorithms
- This lecture is background material to the routing algorithms

Elementary Concepts

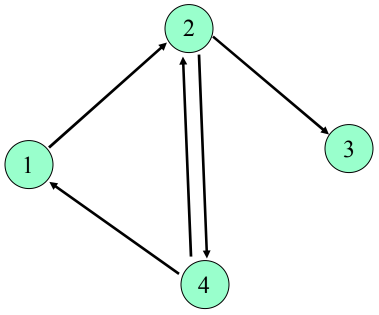
- A graph *G(V,E)* is two sets of object
  - ❖Vertices (or nodes) , set *V*
  - ❖Edges, set *E*
- A graph is represented with dots or circles (vertices) joined by lines (edges)
- The magnitude of graph *G* is characterized by number of vertices |*V*| (called the order of *G*) and number of edges |*E*| (size of *G)*
- The running time of algorithms are measured in terms of the order and size

Graphs ↔ Networks

Graph (Network)	Vertexes (Nodes)	Edges (Arcs)	Flow
Communications	Telephones exchanges, computers, satellites	Cables, fiber optics, microwave relays	Voice, video, packets
Circuits	Gates, registers, processors	Wires	Current
Mechanical	Joints	Rods, beams, springs	Heat, energy
Hydraulic	Reservoirs, pumping stations, lakes	Pipelines	Fluid, oil
Financial	Stocks, currency	Transactions	Money
Transportation	Airports, rail yards, street intersections	Highways, railbeds, airway routes	Freight, vehicles, passengers

Directed Graph

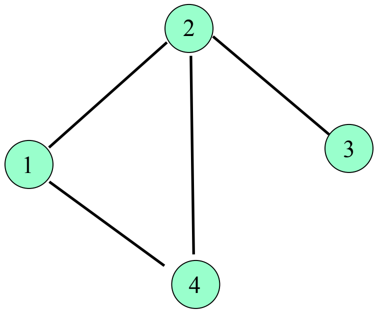
An edge $e \in E$ of a directed graph is represented as an ordered pair (u,v), where $u, v \in V$. Here u is the initial vertex and v is the terminal vertex. Also assume here that $u \neq v$



$V = \{1, 2, 3, 4\}, |V| = 4$
$E = \{(1,2), (2,3), (2,4), (4,1), (4,2)\}, |E| = 5$

Undirected Graph

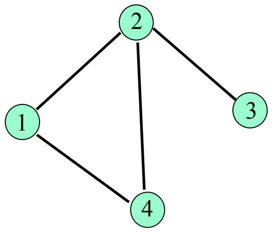
An edge $e \in E$ of an undirected graph is represented as an unordered pair $(u,v)=(v,u)$, where $u, v \in V$. Also assume that $u \neq v$

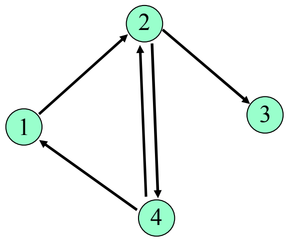


V = { 1, 2, 3, 4}, | V | = 4
E = {(1,2), (2,3), (2,4), (4,1)}, | E |=4

Degree of a Vertex

Degree of a vertex in an undirected graph is the number of edges incident on it. In a directed graph, the *out degree* of a vertex is the number of edges leaving it and the *in degree* is the number of edges entering it



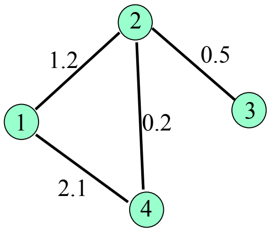


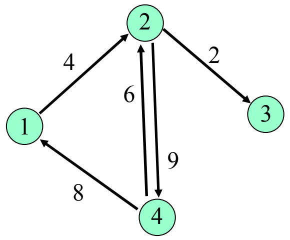
The *degree* of vertex 2 is 3

The *in degree* of vertex 2 is 2 and the *in degree* of vertex 4 is 1

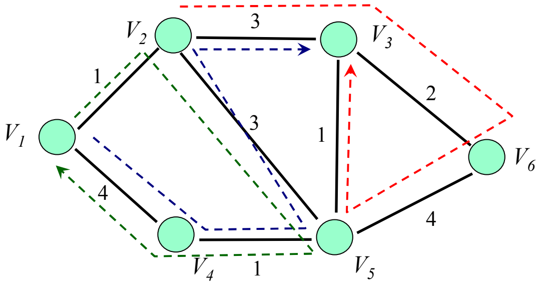
Weighted Graph

A *weighted graph* is a graph for which each edge has an associated *weight*, usually given by a *weight function* $w: E \rightarrow \mathrm{R}$





Walks and Paths



A *walk* is an sequence of nodes $(v_1, v_2, \ldots, v_L)$ such that $\{(v_1, v_2), (v_1, v_2), \ldots, (v_1, v_2)\} \subseteq E$, *e.g.* $(V_2, V_3, V_6, V_5, V_3)$

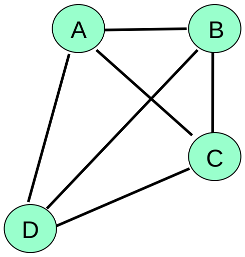
A *simple path* is a walk with no repeated nodes, e.g. $(V_1, V_4, V_5, V_2, V_3)$

A *cycle* is an walk $(v_1, v_2, \ldots, v_L)$ where $v_1 = v_L$ with no other nodes repeated and $L > 3$, e.g. $(V_1, V_2, V_5, V_4, V_1)$

A graph is called *cyclic* if it contains a cycle; otherwise it is called *acyclic*

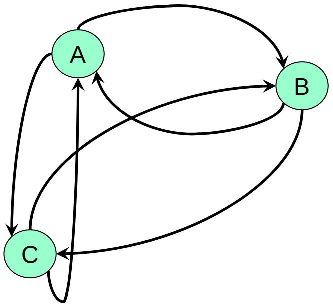
Complete Graphs

A *complete graph* is an undirected/directed graph in which every pair of vertices is *adjacent*. If (u, v) is an edge in a graph G, we say that vertex v is *adjacent* to vertex u.



4 nodes and (4*3)/2 edges

V nodes and V*(V-1)/2 edges

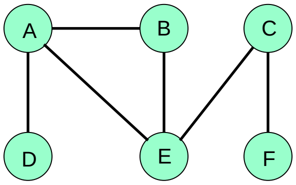


3 nodes and 3*2 edges

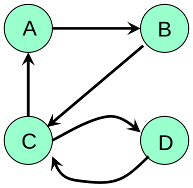
V nodes and V*(V-1) edges

Connected Graphs

An undirected graph is *connected* if you can get from any node to any other by following a sequence of edges OR any two nodes are connected by a path



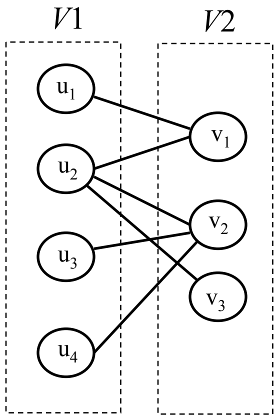
A directed graph is *strongly connected* if there is a directed path from any node to any other node



- A graph is *sparse* if $| E | \approx | V |$
- A graph is *dense* if $| E | \approx | V |^2$

Bipartite Graph

A *bipartite graph* is an undirected graph $G = (V,E)$ in which V can be partitioned into 2 sets $V1$ and $V2$ such that $(u,v) \in E$ implies either
$u \in V1$ and $v \in V2$
OR
$v \in V1$ and $u \in V2$.



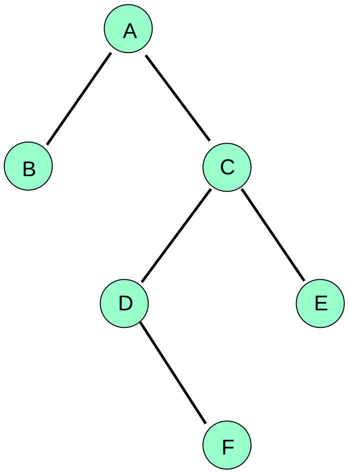
An example of bipartite graph application to telecommunication problems can be found in,
C.A. Pomalaza-Ráez, "A Note on Efficient SS/TDMA Assignment Algorithms," *IEEE Transactions on Communications*, September 1988, pp. 1078-1082.

For another example of bipartite graph applications see the slides in the Addendum section

Trees

Let $G = (V, E)$ be an undirected graph.
The following statements are equivalent,

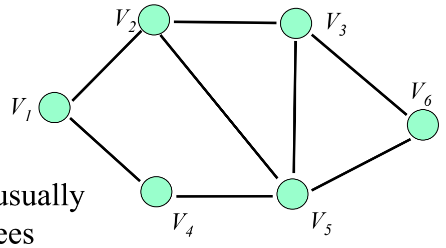
1. G is a tree
2. Any two vertices in G are connected by unique simple path
3. G is connected, but if any edge is removed from E, the resulting graph is disconnected
4. G is connected, and $|E| = |V| - 1$
5. G is acyclic, and $|E| = |V| - 1$
6. G is acyclic, but if any edge is added to E, the resulting graph contains a cycle



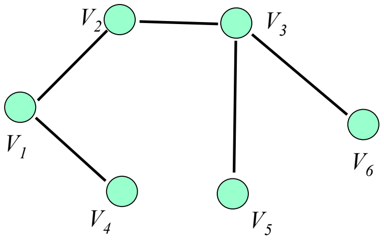
Spanning Tree

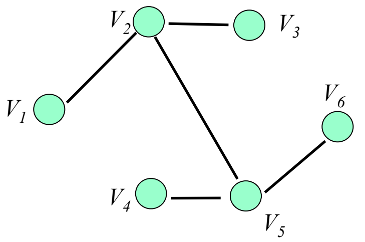
A tree (T) is said to span $G = (V,E)$ if $T = (V,E')$ and $E' \subseteq E$

For the graph shown on the right two possible spanning trees are shown below



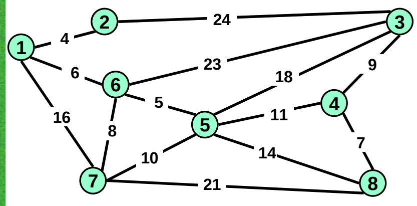
For a given graph there are usually several possible spanning trees



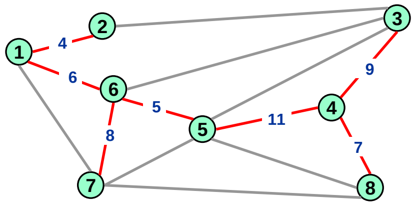


Minimum Spanning Tree

Given connected graph G with real-valued edge weights c_e, a Minimum Spanning Tree (MST*)* is a spanning tree of G whose sum of edge weights is minimized



$G = (V, E)$



$T = (V, F)$ $\boxed{w(T) = 50}$

Cayley's Theorem (1889)

There are n^{n-2} spanning trees of a complete graph K_n

- ❖ $n = |V|$, $m = |E|$
- ❖ Can't solve MST by brute force (because of n^{n-2})

Applications of MST

MST is central combinatorial problem with diverse applications

- Designing physical networks
  - telephone, electrical, hydraulic, TV cable, computer, road
- Cluster analysis
  - delete long edges leaves connected components
  - finding clusters of quasars and Seyfert galaxies
  - analyzing fungal spore spatial patterns
- Approximate solutions to NP-hard problems
  - metric TSP (Traveling Salesman Problem), Steiner tree
- Indirect applications.
  - describing arrangements of nuclei in skin cells for cancer research
  - learning salient features for real-time face verification
  - modeling locality of particle interactions in turbulent fluid flow
  - reducing data storage in sequencing amino acids in a protein

MST Computation

Prim's Algorithm

□ Select an arbitrary node as the initial tree (T)

□ Augment T in an iterative fashion by adding the outgoing edge (u,v), (i.e., $u \in T$ and $v \in G\text{-}T$) with minimum cost (i.e., weight)

□ The algorithm stops after |V | - 1 iterations

□ Computational complexity = O ($|V|^2$)

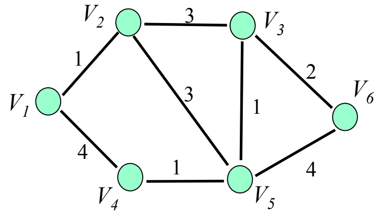
Kruskal's Algorithm

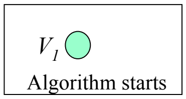
□ Select the edge $e \in E$ of minimum weight → $E' = \{e\}$

□ Continue to add the edge $e \in E - E'$ of minimum weight that when added to E', does not form a cycle

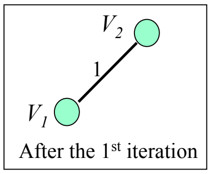
□ Computational complexity = O (|E|xlog|E|)

Prim's Algorithm (example)

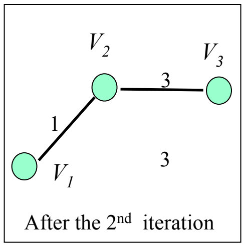




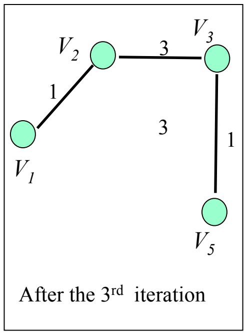
Algorithm starts



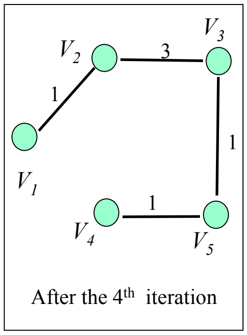
After the 1st iteration



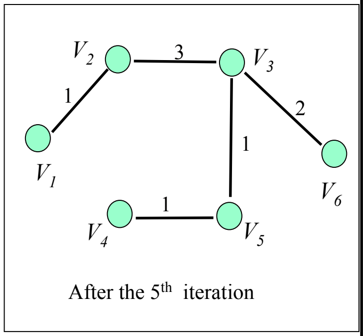
After the 2nd iteration



After the 3rd iteration

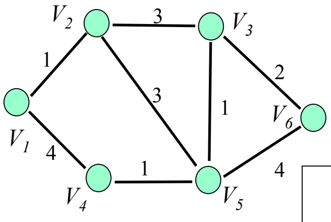


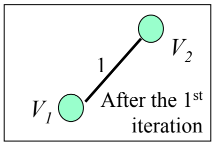
After the 4th iteration



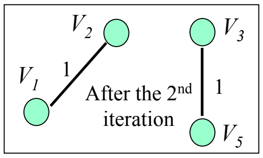
After the 5th iteration

Kruskal's Algorithm (example)

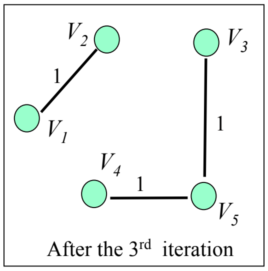




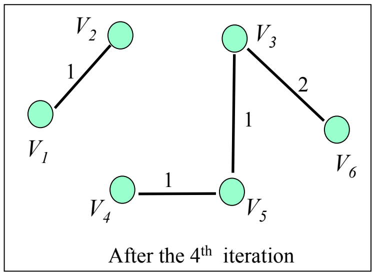
After the 1st iteration



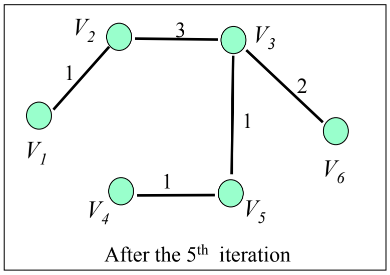
After the 2nd iteration



After the 3rd iteration



After the 4th iteration

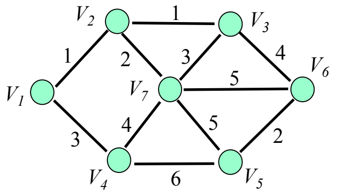


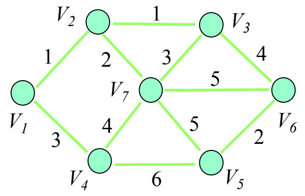
After the 5th iteration

Distributed Algorithms

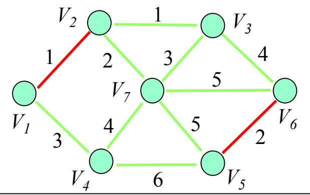
- Each node does not need complete knowledge of the topology
- The MST is created in a distributed manner
- Example of this type of algorithms is the one proposed by Gallager, Humblet, and Spira (“Distributed Algorithm for Minimum-Weight Spanning Trees,” ACM Transactions on Programming Languages and Systems, January 1983, pp. 66-67).
- Starts with one or more fragments consisting of single nodes
- Each fragment selects its minimum weight outgoing edge and using control messaging fragments coordinate to merge with a neighboring fragment over its minimum weight outgoing edge
- The algorithm can produce a MST in O($|V|$x$|V|$) time provided that the edge weights are unique
- If these weights are not unique the algorithm still works by using the nodes IDs to break ties between edges with equal weight
- The algorithm requires O($|V|$xlog$|V|$) + $|E|$) message overhead

Distributed Algorithm- Example



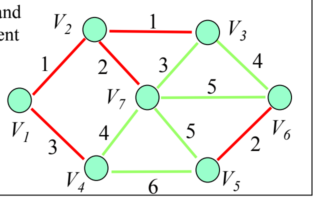


Zero level fragments

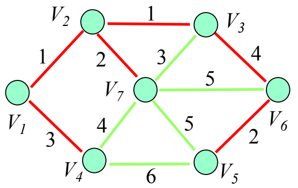


1st level fragments {1,2} and {5,6} are formed

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

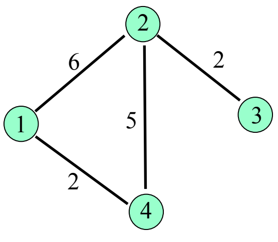


Fragments {1,2,3,4,7} and {5,6} join to form 2nd level fragment that is the MST

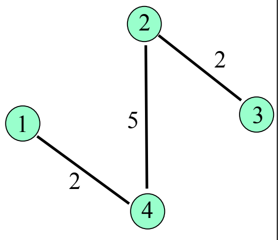


Shortest Path Spanning Tree

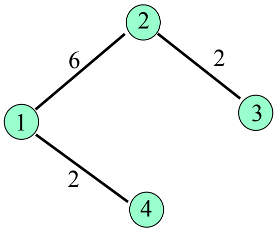
A shortest path spanning tree (SPTS), *T*, is a spanning tree rooted at a particular node such that the |*V*|-1 minimum weight paths from that node to each of the other network nodes is contained in *T*



Graph



Minimum Spanning Tree

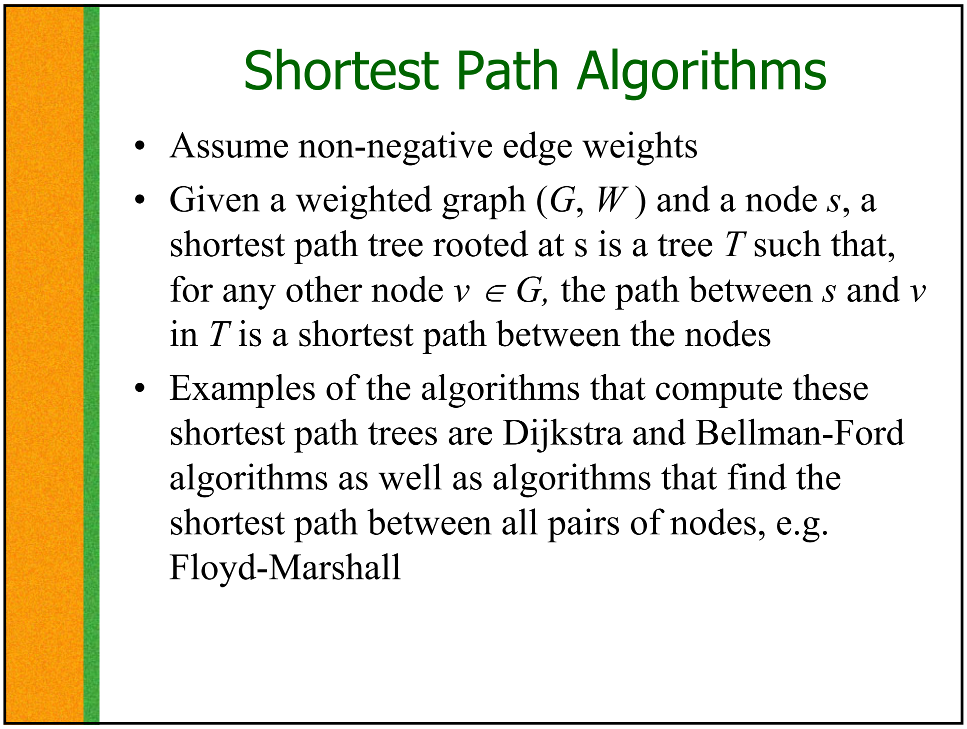


Shortest Path Spanning Tree
rooted at vertex 1

Note that the SPST is not the same as the MST

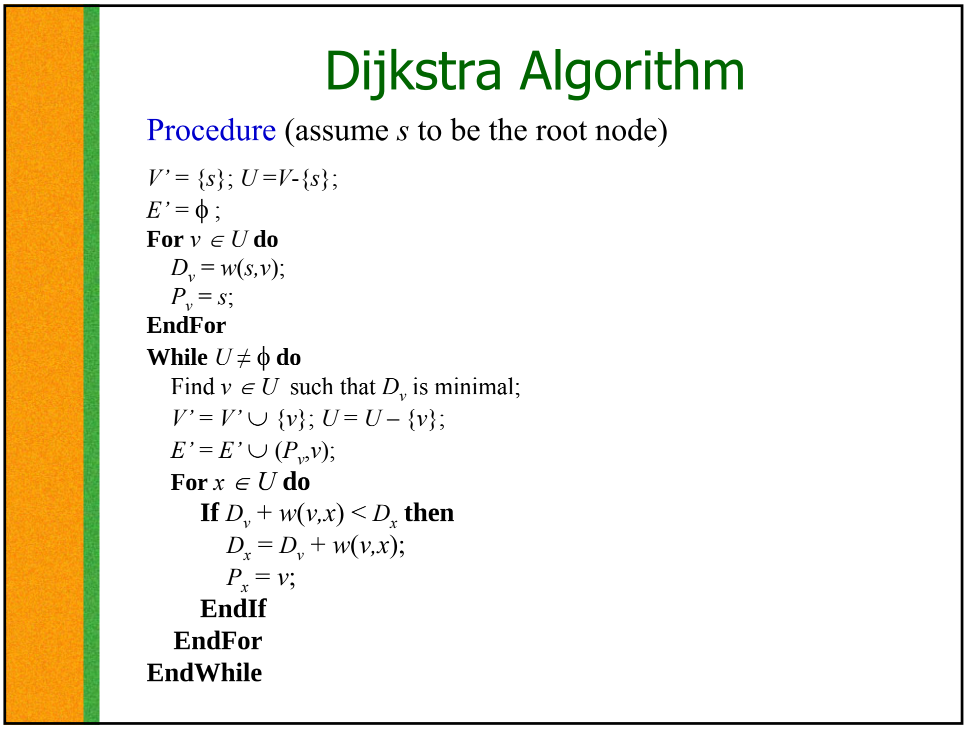
Applications of Trees

- Unicast routing (one to one) → SPST
- Multicast routing (one to several)
- Maximum probability of reliable one to all communications → maximum weight spanning tree
- Load balancing → Degree constrained spanning tree



Shortest Path Algorithms

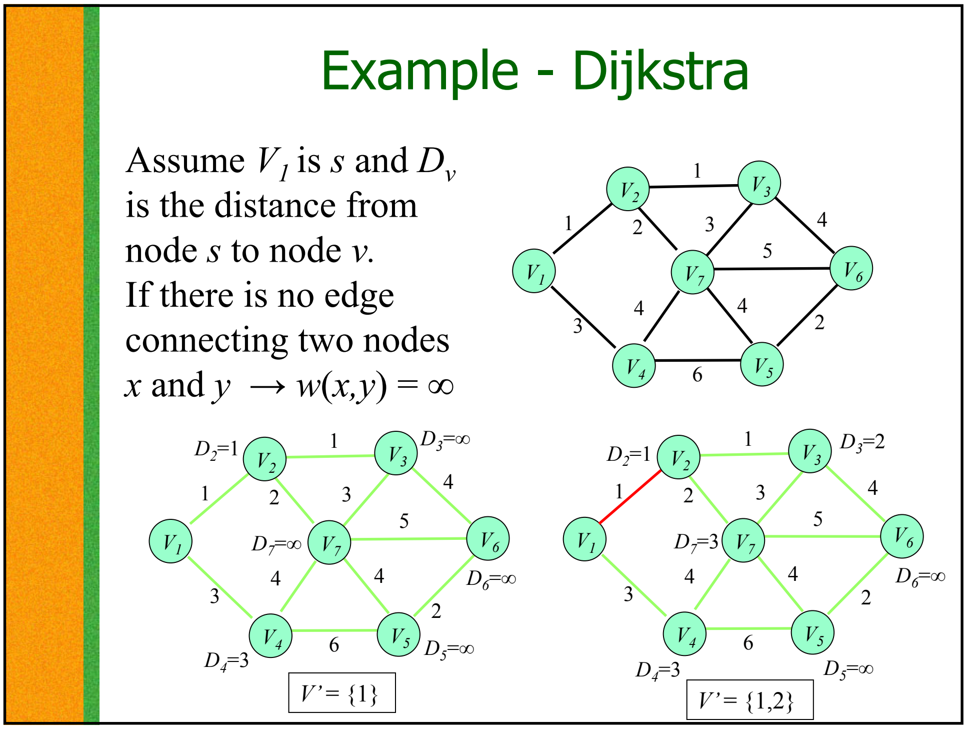
- Assume non-negative edge weights
- Given a weighted graph (*G*, *W*) and a node *s*, a shortest path tree rooted at s is a tree *T* such that, for any other node $v \in G$, the path between *s* and *v* in *T* is a shortest path between the nodes
- Examples of the algorithms that compute these shortest path trees are Dijkstra and Bellman-Ford algorithms as well as algorithms that find the shortest path between all pairs of nodes, e.g. Floyd-Marshall



Dijkstra Algorithm

Procedure (assume *s* to be the root node)

```
$V' = \{s\}$; $U = V - \{s\}$;
$E' = \phi$ ;
For $v \in U$ do
   $D_v = w(s,v)$;
   $P_v = s$;
EndFor
While $U \neq \phi$ 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
```

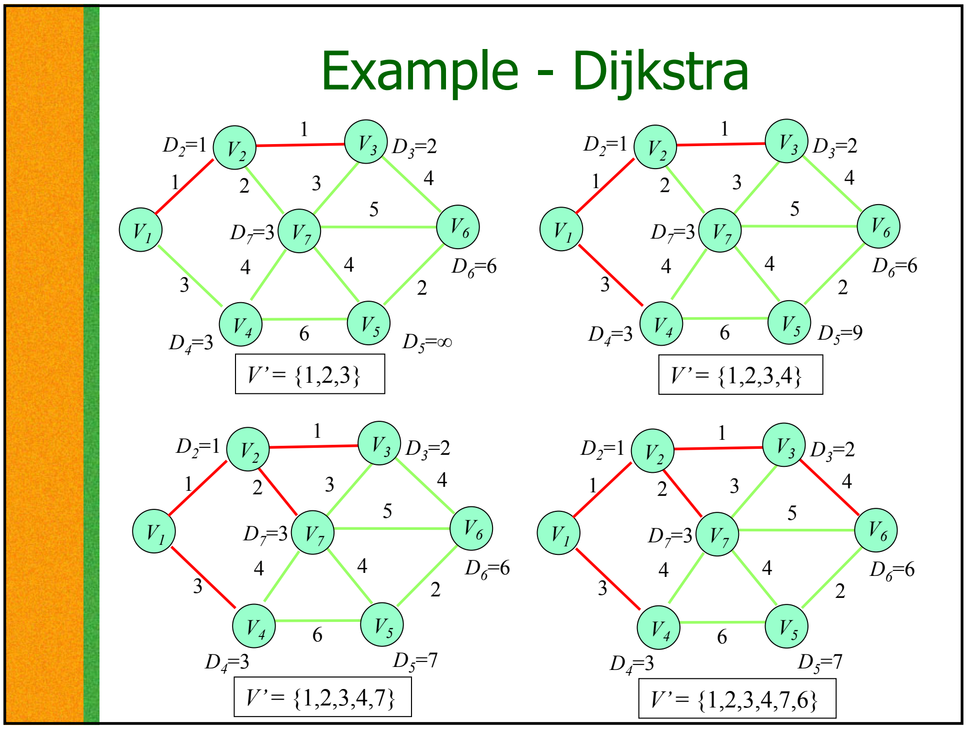


Example - Dijkstra

Assume V_1 is s and D_v is the distance from node s to node v. If there is no edge connecting two nodes x and y → $w(x,y) = \infty$

V'= {1}

V'= {1,2}



Example - Dijkstra

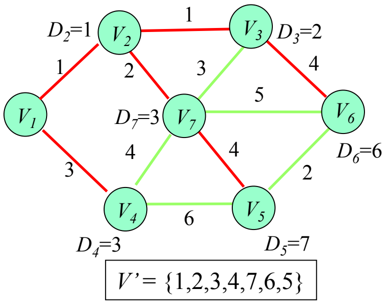
V'= {1,2,3}

V'= {1,2,3,4}

V'= {1,2,3,4,7}

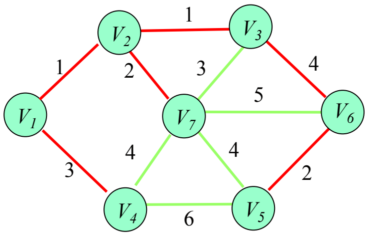
V'= {1,2,3,4,7,6}

Example - Dijkstra



The algorithm terminates when all the nodes have been processed and their shortest distance to node 1 has been computed

Note that the tree computed is not a minimum weight spanning tree. A MST for the given graph is →



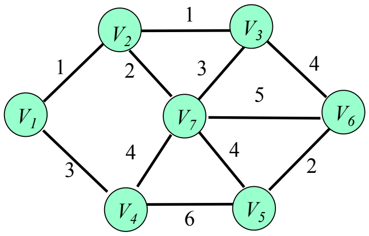
Bellman-Ford Algorithm

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

Procedure

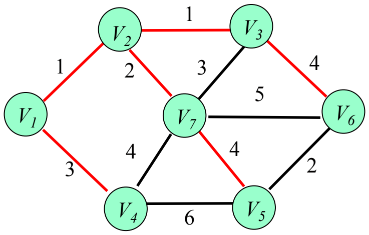
```
$D_v^{-1} = \infty \ \forall \ v \in V$;
$D_s^0 = 0$ and $D_v^0 = \infty \ \forall \ v \neq s, \ v \in V$;
$h = 0$;
Until ($D_v^h = D_v^{h-1} \ \forall \ v \in V$) or ($h = |V|$) do
    $h = h + 1$;
    For $v \in V$ do
        $D_v^{h+1} = \min\{D_u^h + w(u,v)\} \ u \in V$;
    EndFor
EndUntil
```

Bellman-Ford Algorithm (Example)



```
Until ($D_v^h = D_v^{h\text{-}1}\ \forall\ v \in V$) or ($h = |V|$) do
    $h = h + 1$;
    For $v \in V$ do
        $D_v^{h+1} = \min\{D_u^h + w(u,v)\}\ u \in V$;
    EndFor
EndUntil
```

	h=1	*h*=2	*h*=3	*h*=4
D_2^h	1	1	1	1
D_3^h	∞	2	2	2
D_4^h	3	3	3	3
D_5^h	∞	9	7	7
D_6^h	∞	∞	6	6
D_7^h	∞	3	3	3



Floyd-Warshall Algorithm

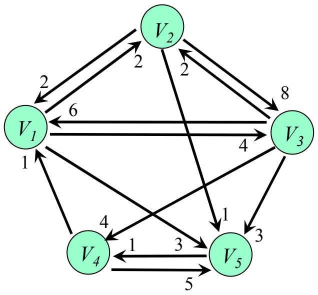
Find the shortest path between all ordered pairs of nodes (s,v), {s,v} $v \in V$. Each iteration yields the path with the shortest weight between all pair of nodes under the constraint that only nodes {1,2,…n}, $n \in |V|$, can be used as intermediary nodes on the computed paths.

Procedure

```
$D = W$; ($W$ is the matrix representation of the edge weights)
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 that this algorithm completes in $O(|V|^3)$ time

Floyd-Warshall Algorithm (Example)



```
$D = W$
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
```

$$D_0 = \begin{array}{cc} & \begin{array}{ccccc} V_1 & V_2 & V_3 & V_4 & V_5 \end{array} \\ & \left[\begin{array}{ccccc} 0 & 2 & 4 & \infty & 3 \\ 2 & 0 & 8 & \infty & 1 \\ 6 & 2 & 0 & 4 & 3 \\ 1 & \infty & \infty & 0 & 5 \\ \infty & \infty & \infty & 1 & 0 \end{array}\right] \begin{array}{c} V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \end{array} \end{array}$$

$$D_1 = \begin{bmatrix} 0 & 2 & 4 & \infty & 3 \\ 2 & 0 & 8 & \infty & 1 \\ 6 & 2 & 0 & 4 & 3 \\ 1 & \boxed{3} & \boxed{5} & 0 & \boxed{4} \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

Floyd-Warshall Algorithm (Example

$$D_2 = \begin{bmatrix} 0 & 2 & 4 & \infty & 3 \\ 2 & 0 & \boxed{6} & \infty & 1 \\ \boxed{4} & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

$$D_3 = \begin{bmatrix} 0 & 2 & 4 & \boxed{8} & 3 \\ 2 & 0 & 6 & \boxed{10} & 1 \\ 4 & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

$$D_4 = \begin{bmatrix} 0 & 2 & 4 & 8 & 3 \\ 2 & 0 & 6 & 10 & 1 \\ 4 & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ \boxed{2} & \boxed{4} & \boxed{6} & 1 & 0 \end{bmatrix}$$

$$D_5 = \begin{bmatrix} 0 & 2 & 4 & \boxed{4} & 3 \\ 2 & 0 & 6 & \boxed{2} & 1 \\ 4 & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ 2 & 4 & 6 & 1 & 0 \end{bmatrix}$$

Distributed Asynchronous Shortest Path Algorithms

- Each node computes the path with the shortest weight to every network node
- There is no centralized computation
- As for the distributed MST algorithm described in [Gallager, Humblet, and Spiral], control messaging is required to distributed computation
- Asynchronous means here that there is no requirement of inter-node synchronization for the computation performed at each node of for the exchange of messages between nodes

Distributed Dijkstra Algorithm

- There is no need to change the algorithm
- Each node floods periodically a control message throughout the network containing link state information → transmission overhead is $O(|V|\text{x}|E|)$
- Entire topology knowledge must be maintained at each node
- Flooding of the link state information allows for timely dissemination of the topology as perceived by each node. Each node has typically accurate information to be able to compute the shortest paths

Distributed Bellman-Ford Algorithm

- Assume G contains only cycles of *non-negative* weight
- If $(u,v) \in E$ then so is (v,u)
- The update equation is

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

$N(s)$ = Neighbors of s → $\forall u \in N(s), (s,u) \in E$

- Each node only needs to know the weights of the edges that are incident to it, the identity of all the network nodes and estimates (received from its neighbors) of the distances to all network nodes

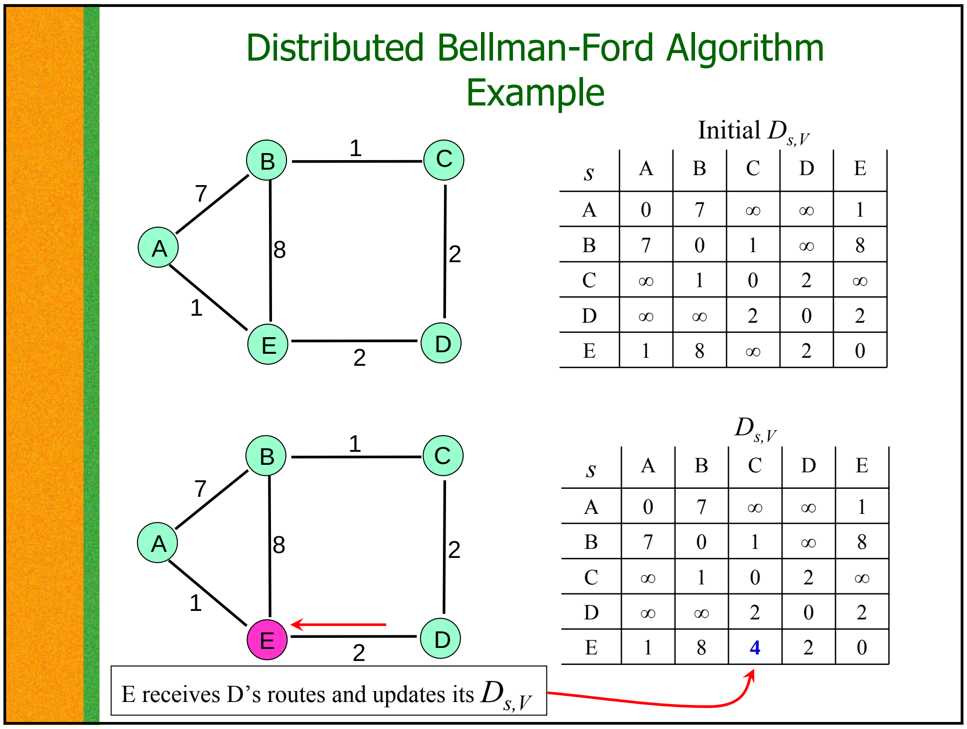
Distributed Bellman-Ford Algorithm

- Each node s transmits to its neighbors its current distance vector $D_{s,V}$
- Likewise each neighbor node $u \in N(s)$ transmits to s its distance vector $D_{u,V}$
- 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 update changes a distance value then s sends the current version of $D_{s,v}$ to its neighbors

- Node s updates $D_{s,v}$ every time that it receives a distance vector information from any of its neighbors
- A periodic timer prompts node s to recompute $D_{s,V}$ or to transmit a copy of $D_{s,V}$ to each of its neighbors



Distributed Bellman-Ford Algorithm Example

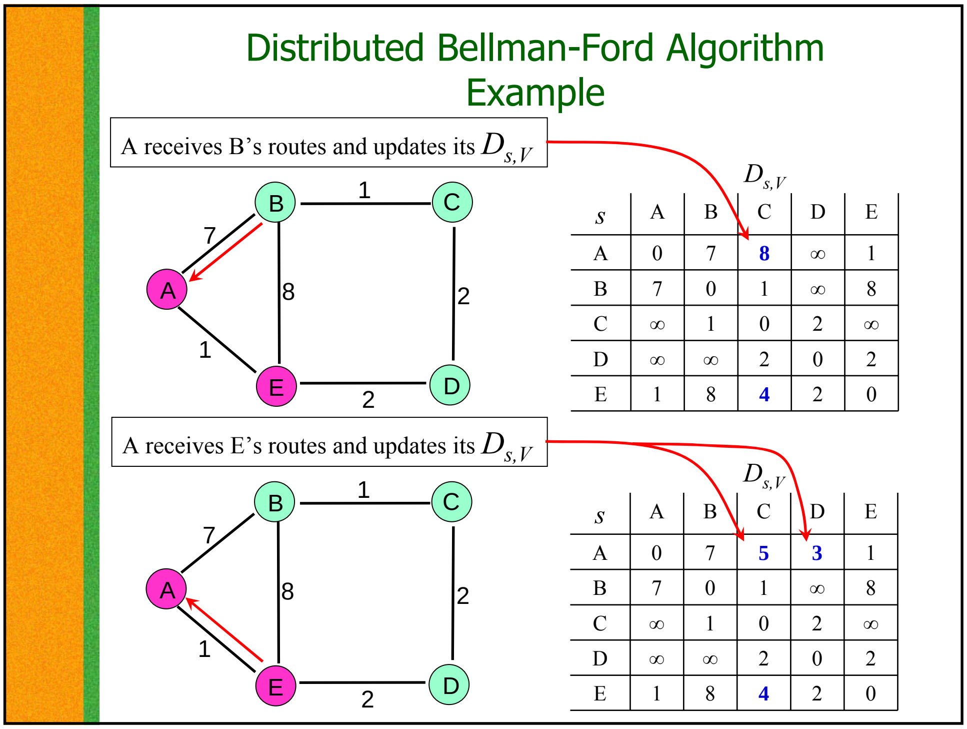
Initial $D_{s,V}$

s	A	B	C	D	E
A	0	7	∞	∞	1
B	7	0	1	∞	8
C	∞	1	0	2	∞
D	∞	∞	2	0	2
E	1	8	∞	2	0

$D_{s,V}$

s	A	B	C	D	E
A	0	7	∞	∞	1
B	7	0	1	∞	8
C	∞	1	0	2	∞
D	∞	∞	2	0	2
E	1	8	**4**	2	0

E receives D’s routes and updates its $D_{s,V}$



Distributed Bellman-Ford Algorithm Example

A receives B’s routes and updates its $D_{s,V}$

$D_{s,V}$

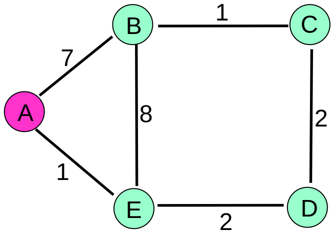
s	A	B	C	D	E
A	0	7	**8**	∞	1
B	7	0	1	∞	8
C	∞	1	0	2	∞
D	∞	∞	2	0	2
E	1	8	**4**	2	0

A receives E’s routes and updates its $D_{s,V}$

$D_{s,V}$

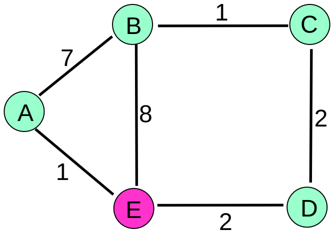
s	A	B	C	D	E
A	0	7	**5**	**3**	1
B	7	0	1	∞	8
C	∞	1	0	2	∞
D	∞	∞	2	0	2
E	1	8	**4**	2	0

Distributed Bellman-Ford Algorithm Example



A’s routing table

Destination	Next Hop	Distance
B	E	**6**
C	E	**5**
D	E	**3**
E	E	**1**



E’s routing table

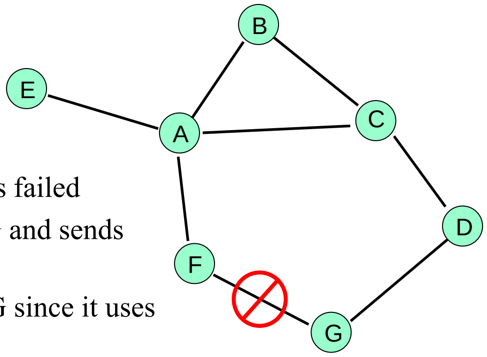
Destination	Next Hop	Distance
A	A	**1**
B	D	**5**
C	D	**4**
D	D	**2**

Distance Vector Protocols

- Each node maintains a routing table with entries {Destination, Next Hop, Distance (cost)}
- Nodes exchange routing table information with neighbors
  - Whenever table changes
  - Periodically
- Upon reception of a routing table from a neighbor a node updates its routing table if finds a “better” route
- Entries in the routing table are deleted if they are too old, i.e. they are not “refreshed” within certain time interval by the reception of a routing table

Link Failure

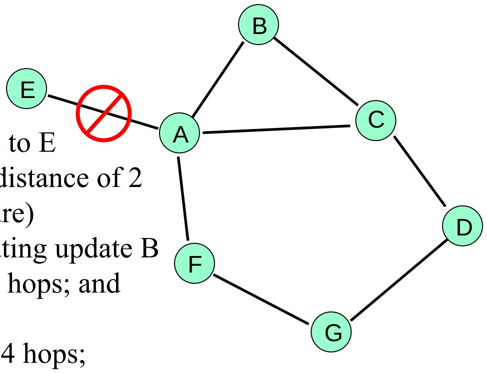
Simple rerouting case



- F detects that link to G has failed
- F sets a distance of ∞ to G and sends update to A
- A sets a distance of ∞ to G since it uses F to reach G
- A receives periodic update from C with 2-hop path to G (via D)
- A sets distance to G to 3 and sends update to F
- F decides it can reach G in 4 hops via A

Link Failure

Routing loop case

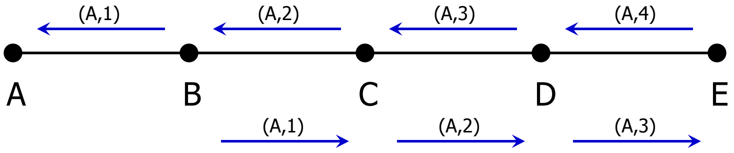


- Link from A to E fails
- A advertises distance of ∞ to E
- B and C had advertised a distance of 2 to E (prior to the link failure)
- Upon reception of A’s routing update B decides it can reach E in 3 hops; and advertises this to A
- A decides it can read E in 4 hops; advertises this to C
- C decides that it can reach E in 5 hops…

This behavior is called *count-to-infinity*

Count-to-Infinity Problem

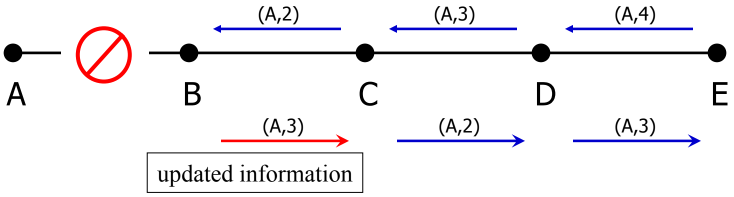
Example: routers working in stable state



Routing updates with distances to A are shown

Count-to-Infinity Problem

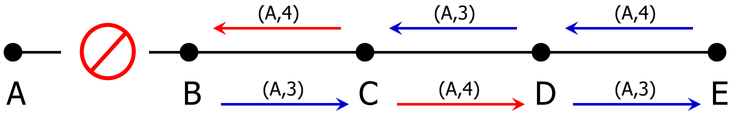
Example: link from A to B fails



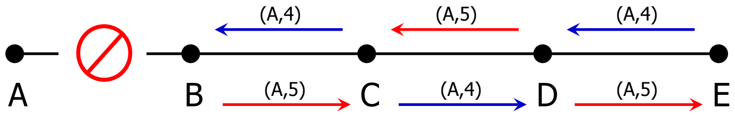
B can no longer reach A directly, but C advertises a distance of 2 to A and thus B now believes it can reach A via C and advertises it

Count-to-Infinity Problem

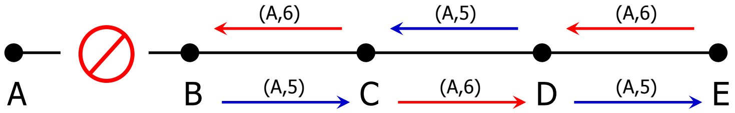
After 2 exchanges of updates



After 3 exchanges of updates

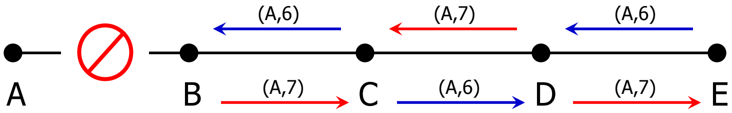


After 4 exchanges of updates

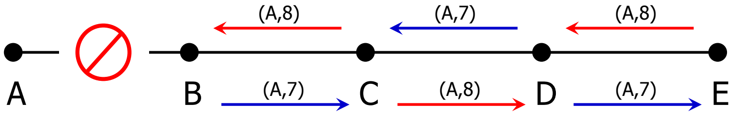


Count-to-Infinity Problem

After 5 exchanges of updates



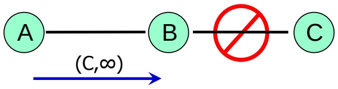
After 6 exchanges of updates



This continues until the distance to A reaches infinity

Split Horizon Algorithm

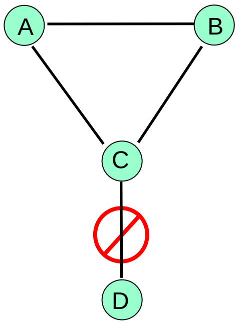
- Used to avoid (not always) the count-to-infinity problem
- If A routes to C via B, then A tells B that its distance to C is ∞



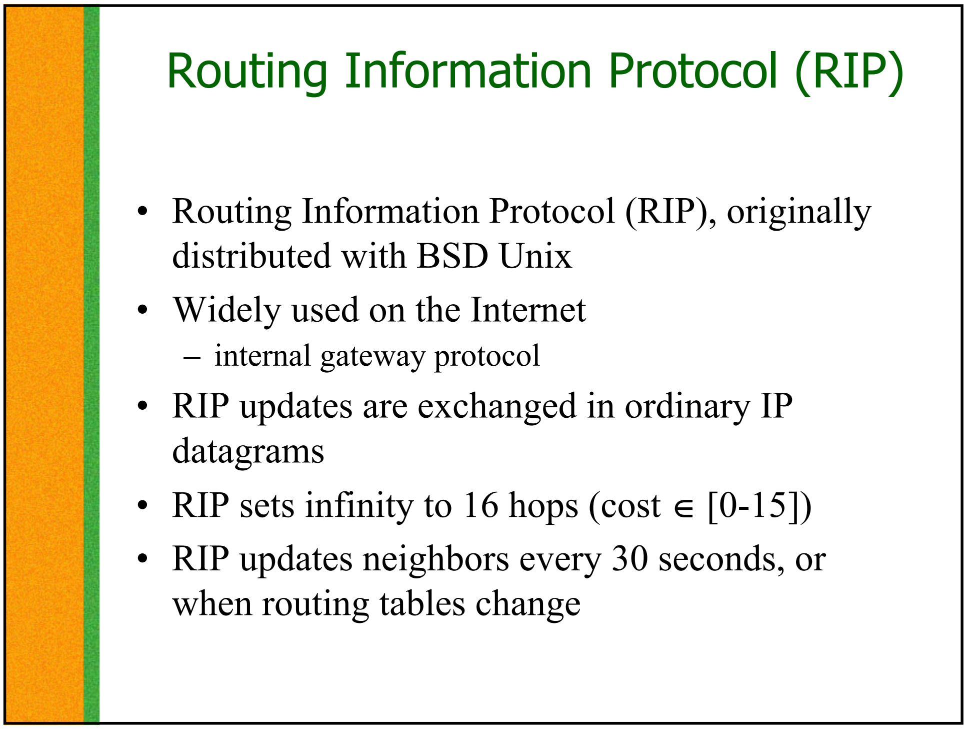
B will not route to C via A if the link B to C fails

- Works for two node loops
- Does not work for loops with more than two nodes

Example Where Split Horizon Fails



- When link C to D breaks, C marks D as unreachable and reports that to A and B.
- Suppose A learns it first
- A now thinks best path to D is through B
- A reports D unreachable to B and a route of cost 3 to C
- C thinks D is reachable through A at cost 4 and reports that to B.
- B reports a cost 5 to A who reports new cost to C.
- etc...



Routing Information Protocol (RIP)

- Routing Information Protocol (RIP), originally distributed with BSD Unix
- Widely used on the Internet
  - internal gateway protocol
- RIP updates are exchanged in ordinary IP datagrams
- RIP sets infinity to 16 hops (cost ∊ [0-15])
- RIP updates neighbors every 30 seconds, or when routing tables change