Network Routing

- **A major component of the network layer routing protocol.**
- **Routing protocols use routing algorithms.**
- **Job of a routing algorithm: Given a set of routers with links connecting the routers, find a "good" path from the source to the destination.**

Modeling a Network

- A network can be modeled by a graph.
  - Routers/switches are represented by nodes.
  - Physical links between routers/switches are represented by edges.
  - Attached computers are ignored.
  - Each edge is assigned a weight representing the "cost" of sending a packet across that link.
- The total cost of a path is the sum of the costs of the edges.
- The problem is to find the least-cost path.

Routing Algorithms

- Routing algorithms that solve a routing problem are based on shortest-path algorithms.
- Two common shortest-path algorithms are Dijkstra's Algorithm and the Bellman-Ford Algorithm.
- Routing algorithms fall into two general categories.

Link-State Algorithms

- **The network topology and all link costs are known.**
- **Example: Dijkstra's Algorithm.**
- **More complex of the two types.**
- **Nodes perform independent computations.**
- **Used in Open Shortest Path First (OSPF) protocol, a protocol intended to replace RIP.**

Distance-Vector Algorithms

- **Nodes receive information from their directly attached neighbors.**
- **Example: Bellman-Ford Algorithm.**
- **Simpler of the two types.**
- **May have convergence problems.**
- **Used in Routing Information Protocol (RIP).**

Dijkstra’s Algorithm

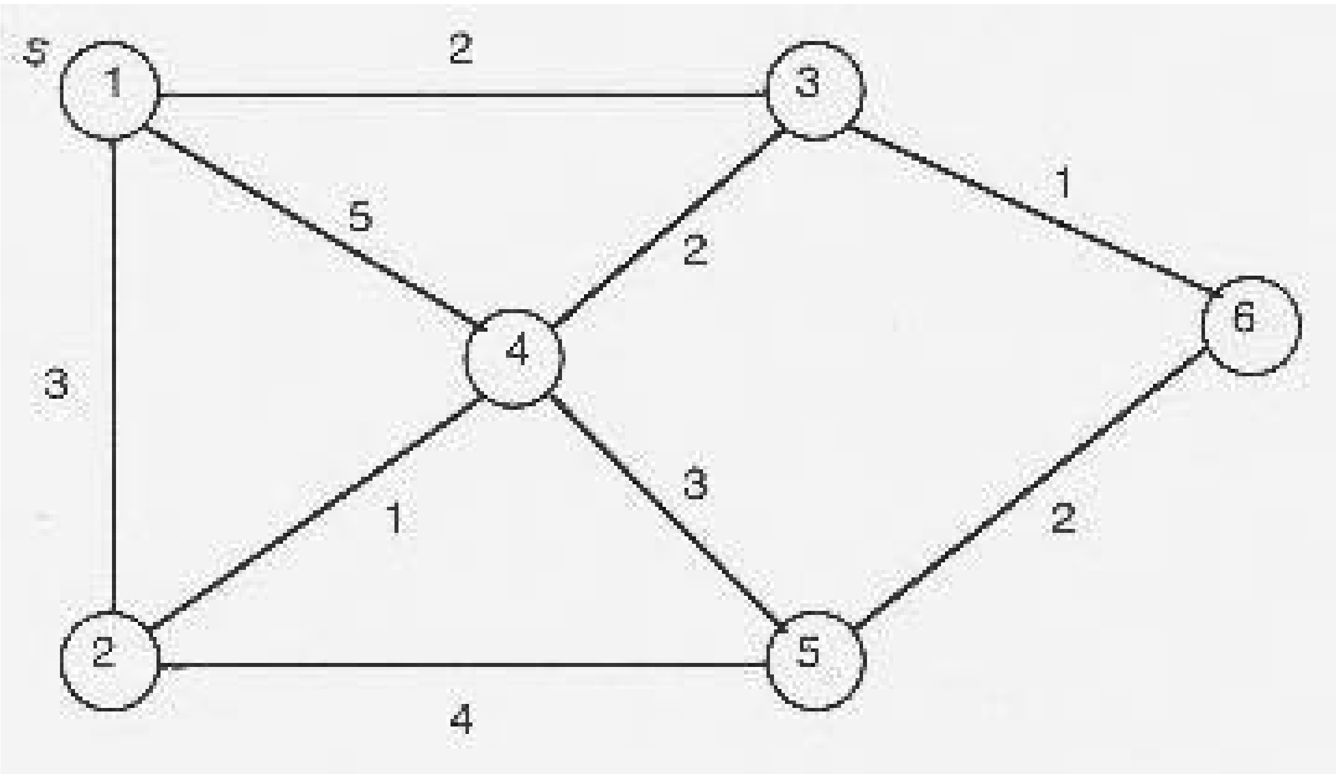
- Named after E. W. Dijkstra.
- Fairly efficient.
- Iterative algorithm.
- At the first iteration, the algorithm finds the closest node from the source node which must be a neighbor of the source node.
- At the second iteration, the algorithm finds the second-closest node from the source node. This node must be a neighbor of either the source node or the closest node found in the first iteration.

Dijkstra's Algorithm

- At the third iteration, the algorithm finds the third-closest node from the source node. This node must be a neighbor of either the source node or one of the first two closest nodes.
- The process continues. At the k^{th} iteration, the algorithm finds the first k closest nodes from the source node.

Example

The source node is s = 1.



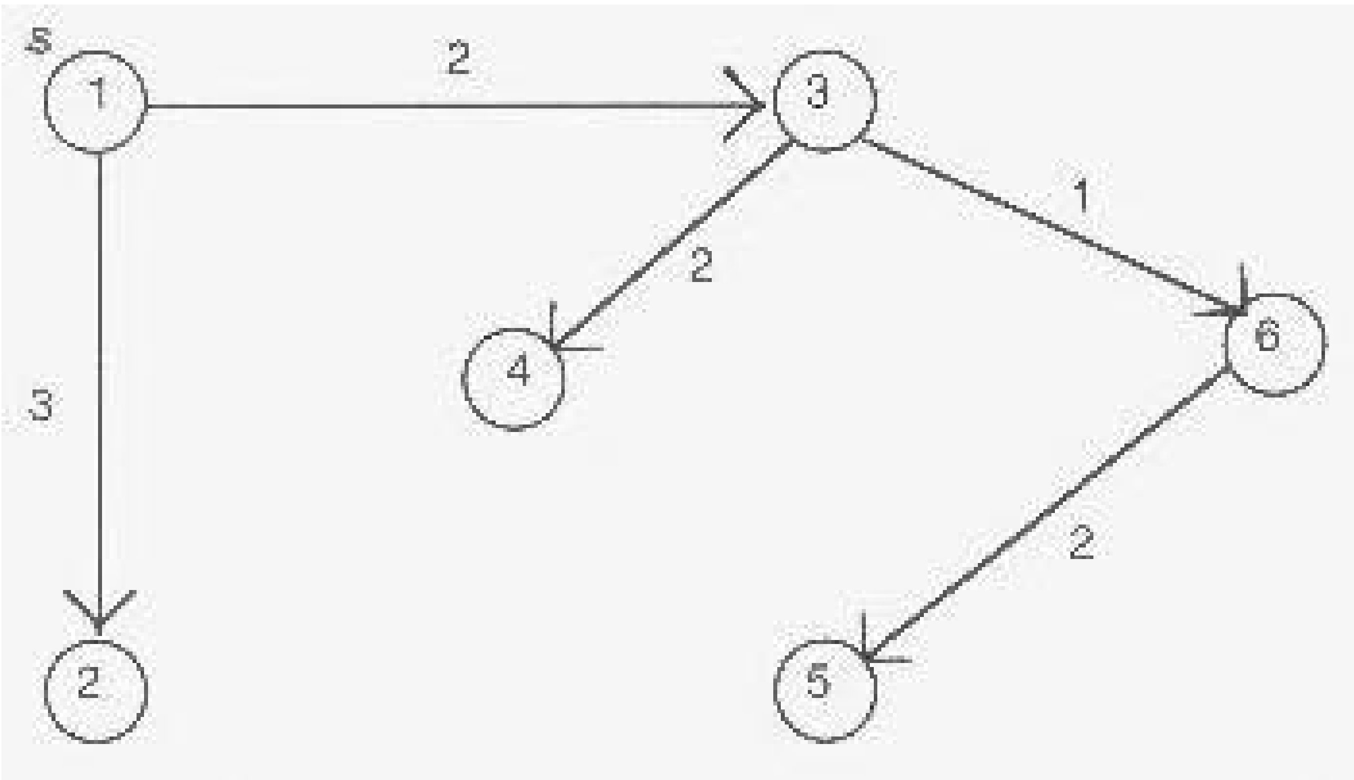
Example

Iteration	N	D_2	D_3	D_4	D_5	D_6
0	{1}	3	2	5	∞	∞
1	{1,3}	3		4	∞	3
2	{1,2,3}			4	7	3
3	{1,2,3,6}			4	5	
4	{1,2,3,4,6}				5	
5	{1,2,3,4,5,6}					

The bottom entry in each D-column is the minimum cost to go from the start node 1 to that node.

- **Question: How can you determine the path which gives the minimum cost to a destination node?**
- **Answer: The table not only gives the minimum costs. It also gives the predecessor node of each node along a least-cost path from the source node. By keeping track of the predecessor nodes, we can construct a least-cost path.**

Least-Cost Path Tree



Routing Table for Source Node 1

Destination	Next Node	Cost
2	2	3
3	3	2
4	3	4
5	3	5
6	3	3

Complexity of Dijkstra’s Algorithm

- Suppose there are n nodes not counting the source node.
- In the first iteration, we need to search through n nodes to determine the node not in N with minimum cost.
- In the second iteration, we need to check n–1 nodes.
- In the third iteration, n–2 nodes. And so on.

Complexity of Dijkstra's Algorithm

- The total number of nodes we need to examine is

$$1 + 2 + 3 + \cdots + n = n(n+1)/2$$

- Thus, Dijkstra's Algorithm as presented is

$$O(n^2)$$

- A more sophisticated implementation of the second step using a heap would find the minimum in logarithmic instead of linear time. This improves the performance to

$$O(n \log n)$$