# Network : Routing Algorithm

Jae Hyeon Kim

# Reference

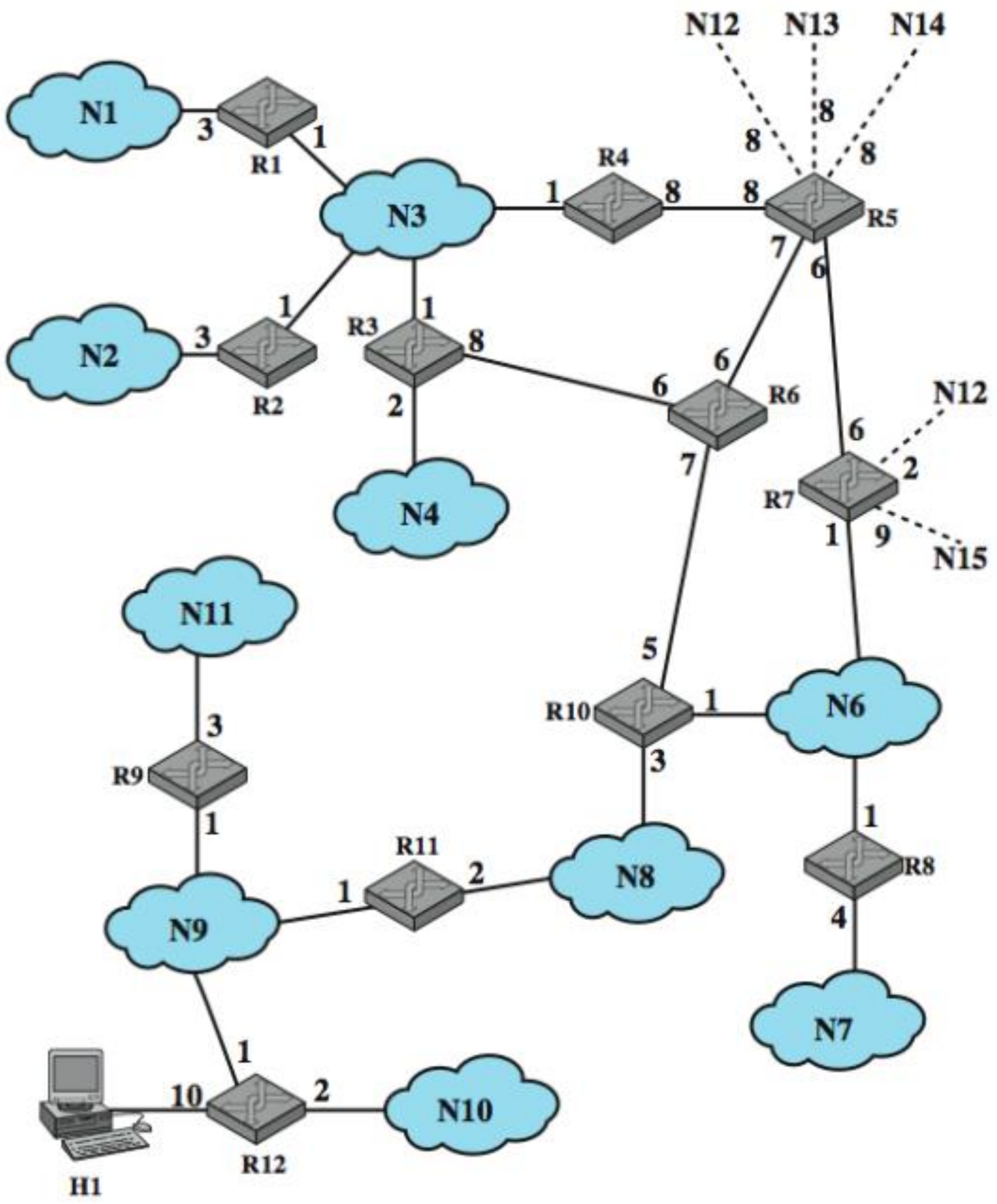
William Stalling, Data and Computer Communications 10/E, Prentice Hall

# OSPF

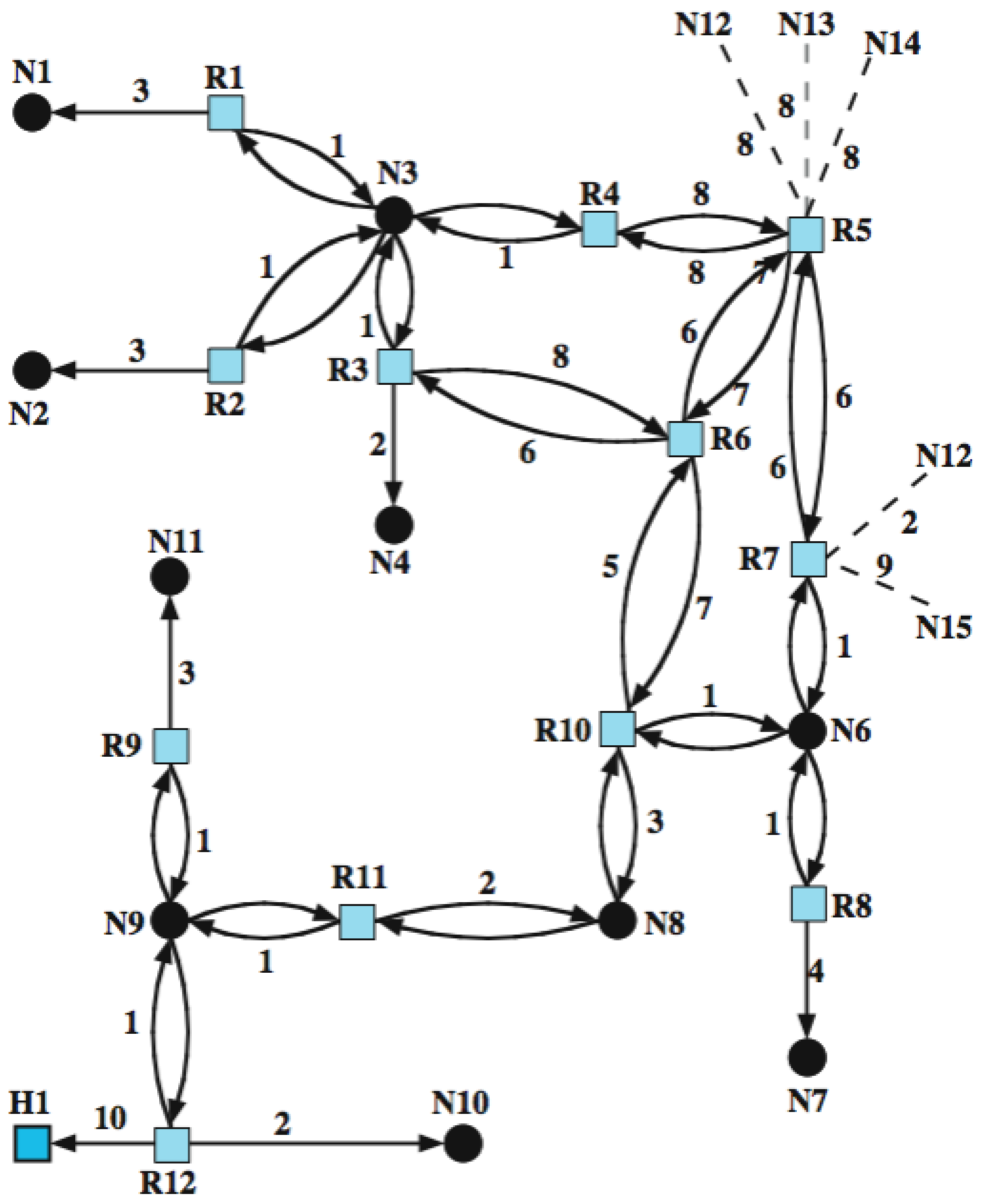
- IGP(Interior Gateway Protocol) in TCP/IP networks
  - Documented with RFC 2328
  - Replaced Routing Information Protocol (RIP)
- Computes a route through the Internet that incurs the least cost based on a user-configurable metric of cost
- Uses link state routing algorithm
  - Each router keeps list of state of local links to network
  - Transmits update state info to all routers
  - Little traffic as messages are small and not sent often
- Uses least cost based on User cost metric
  - E.g. Dijkstra's algorithm

# Sample OSPF AS

- Topology stored as directed graph
- Vertices or nodes
  - Router
  - network
- Edges
  - Connect two router
  - Connect router to network



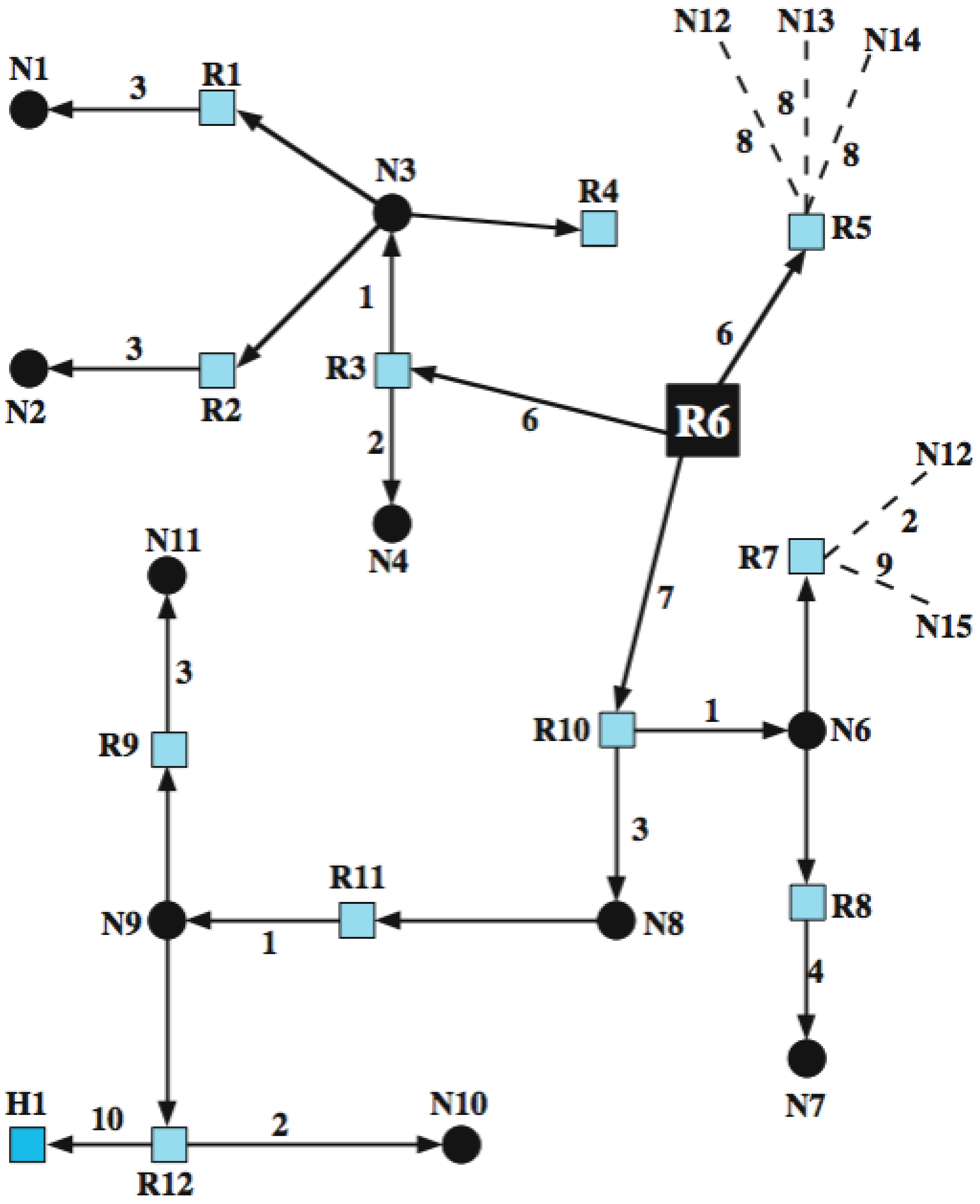
## – Directed Graph of AS



# – SPF Tree for Router 6

- Routing table for R6

| Destination | Next Hop | Distance |
|---|---|---|
| N1 | R3 | 10 |
| N2 | R3 | 10 |
| N3 | R3 | 7 |
| N4 | R3 | 8 |
| N6 | R10 | 8 |
| N7 | R10 | 12 |
| N8 | R10 | 10 |
| N9 | R10 | 11 |
| N10 | R10 | 13 |
| N11 | R10 | 14 |
| H1 | R10 | 21 |
| R5 | R5 | 6 |
| R7 | R10 | 8 |
| N12 | R10 | 10 |
| N13 | R5 | 14 |
| N14 | R5 | 14 |
| N15 | R10 | 17 |



# Least Cost Algorithms

- Basis for routing decisions
  - Minimize hop with each link cost 1
  - Have link cost value inversely proportional to capacity
- Defines cost of path between two nodes as sum of costs of links traversed
  - Network of nodes connected by bi-directional links
  - Link has a cost in each direction
- For each pair of nodes, find path with least cost
  - Link costs in different directions may be different
- Alternatives: Dijkstra of Bellman-Ford algorithms

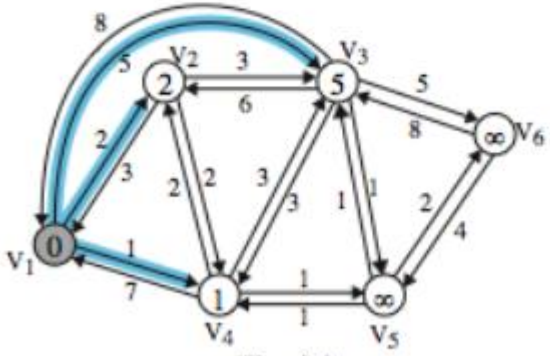
# Dijkstra's Algorithm

- Find shortest paths from given source to all other nodes, by developing paths in order of increasing path length
  - *N* = set of nodes in the network
  - *S* = source node
  - *T* = set of nodes so far incorporated by the algorithm
- W(*I*, *j* ) = link cost from node *i* to node *j*
  - w(*i*, *i*) = 0
  - w(*i*, *j*) = ∞ if the two nodes are not directly connected
  - w(*i*, *j*) >= 0 if the two nodes are directly connected
- *L*(n) = cost of least-cost path from node *s* to node *n* currently known
  - at termination, *L*(n) is cost of least-cost path from *s* to *n*

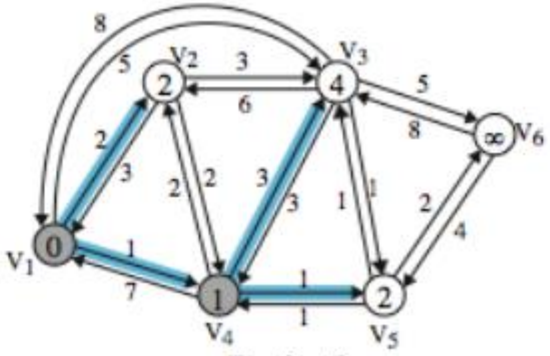
# Dijkstra's Algorithm Method

- Step 1 [initialization]
  - $T = \{s\}$ set of nodes so far incorporated consists of only source node
  - $L(n) = w(s, n)$ for $n \neq s$
  - initial path costs to neighboring nodes are simply link costs
- Step 2 [get next node]
  - find neighboring node not in T with least-cost path from s
  - incorporate node into T
  - also incorporate the edge that is incident on that node and a node in T that contributes to the path
- Step 3 [update least-cost paths]
  - $L(n) = \min[L(n), L(x) + w(x, n)]$ for all $n \notin T$
  - if latter term is minimum, path from s to n is path from s to x concatenated with edge from x to n
- Algorithm terminates when all nodes have been added to T

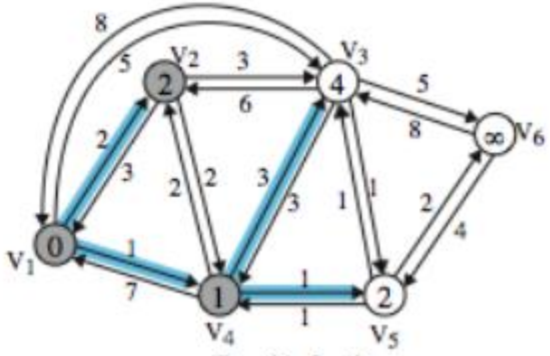
# Example of Dijkstra's Algorithm



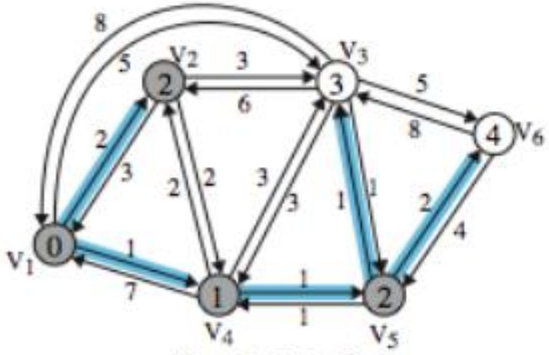
T = {1}



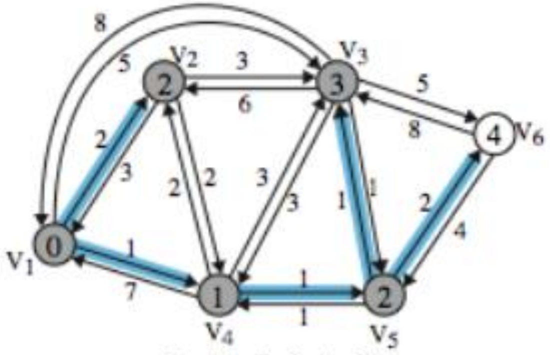
T = {1, 4}



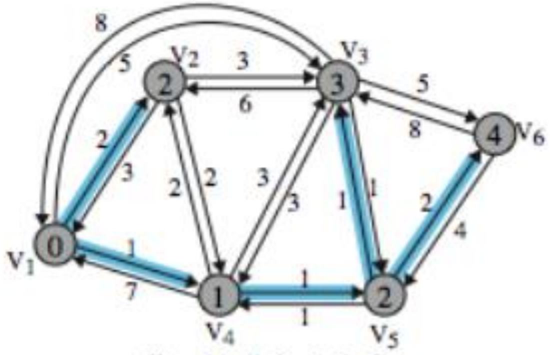
T = {1, 2, 4}



T = {1, 2, 4, 5}



T = {1, 2, 3, 4, 5}



T = {1, 2, 3, 4, 5, 6}

## Result of Example Dijkstra's Algorithm

| Iteration | T | L(2) | Path | L(3) | Path | L(4) | Path | L(5) | Path | L(6) | Path |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | {1} | 2 | 1–2 | 5 | 1-3 | 1 | 1–4 | ∞ | - | ∞ | - |
| 2 | {1,4} | 2 | 1–2 | 4 | 1-4-3 | 1 | 1–4 | 2 | 1-4–5 | ∞ | - |
| 3 | {1, 2, 4} | 2 | 1–2 | 4 | 1-4-3 | 1 | 1–4 | 2 | 1-4–5 | ∞ | - |
| 4 | {1, 2, 4, 5} | 2 | 1–2 | 3 | 1-4-5–3 | 1 | 1–4 | 2 | 1-4–5 | 4 | 1-4-5–6 |
| 5 | {1, 2, 3, 4, 5} | 2 | 1–2 | 3 | 1-4-5–3 | 1 | 1–4 | 2 | 1-4–5 | 4 | 1-4-5–6 |
| 6 | {1, 2, 3, 4,5, 6} | 2 | 1-2 | 3 | 1-4-5-3 | 1 | 1-4 | 2 | 1-4–5 | 4 | 1-4-5-6 |

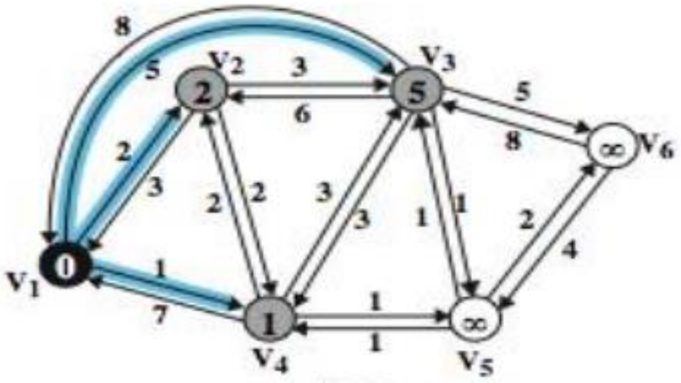
# Bellman-Ford Algorithm Definitions

- Idea
  - find shortest paths from given node subject to constraint that paths contain at most one link
  - find the shortest paths with a constraint of paths of at most two links
- S = source node
- W(I, j) = link cost from node I to node j
  - w(i, i) = 0
  - w(i, j) = ∞ if the two nodes are not directly connected
  - w(i, j) >= 0 if the two nodes are directly connected
- $L_h$ (n) = cost of least-cost path from s to n under constraint of no more than h links
  - h = maximum # of links in path at current stage of the algorithm

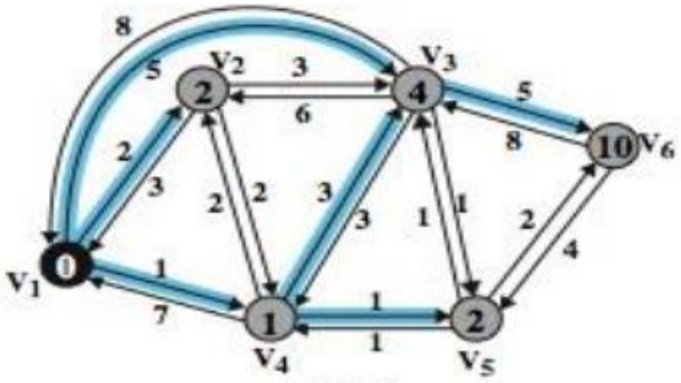
# – Bellman-Ford Algorithm Method

- Step 1 [initialization]
  - $L_0(n) = \infty$, for all $n \neq s$
  - $L_h(s) = 0$, for all $h$
- Step 2 [update]
  - for each successive $h \geq 0$, $n \neq s$
    - compute $L_{h+1}(n) = \min_j[L_h(j) + w(j,n)]$
  - connect $n$ with predecessor node $j$ that achieves minimum
  - eliminate any connection of $n$ with different predecessor node formed during an earlier iteration
  - path from $s$ to $n$ terminates with link from $j$ to $n$

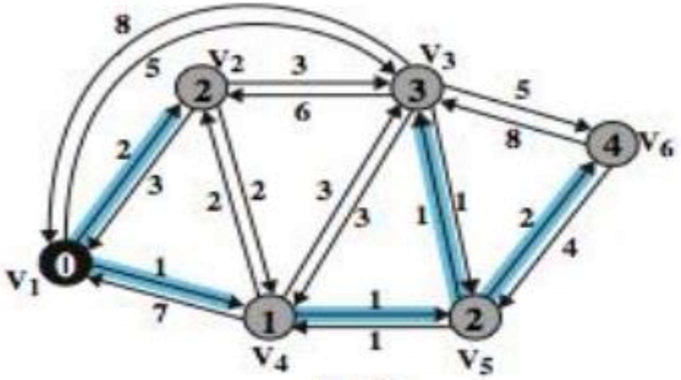
# – Bellman-Ford Algorithm Method



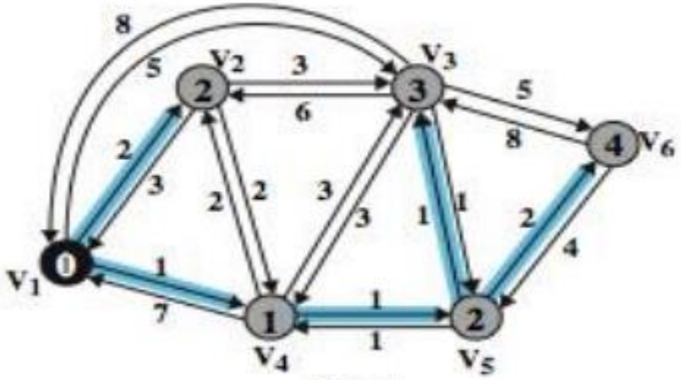

h = 1




h = 2




h = 3




h = 4

# – Bellman-Ford Algorithm Method

| h | $L_h(2)$ | Path | $L_h(3)$ | Path | $L_h(4)$ | Path | $L_h(5)$ | Path | $L_h(6)$ | Path |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | ∞ | - | ∞ | - | ∞ | - | ∞ | - | ∞ | - |
| 1 | 2 | 1-2 | 5 | 1-3 | 1 | 1-4 | ∞ | - | ∞ | - |
| 2 | 2 | 1-2 | 4 | 1-4-3 | 1 | 1-4 | 2 | 1-4-5 | 10 | 1-3-6 |
| 3 | 2 | 1-2 | 3 | 1-4-5-3 | 1 | 1-4 | 2 | 1-4-5 | 4 | 1-4-5-6 |
| 4 | 2 | 1-2 | 3 | 1-4-5-3 | 1 | 1-4 | 2 | 1-4-5 | 4 | 1-4-5-6 |

# Comparison

- Results from two algorithms agree each other
- Bellman-Ford
  - Route calculation for node n involves knowledge of link cost to all neighbor nodes plus total cost to each neighbor from s (step 2)
  - Each node can maintain set of costs and paths for other node
  - Can exchange information with direct neighbors
  - Can update costs and paths based on information from neighbors and knowledge of link costs
- Dijkstra
  - Each node needs complete topology
  - Must know link costs of all links in network (step 3)
  - Must exchange information with all other nodes

# Evaluation

- Dependent on
  - Processing time of algorithms
  - Amount of information required from other nodes
- Implementation specific
- Both converge under static topology and costs
- Both converge to same solution
- If link costs change, algorithms will attempt to catch up
- If link costs depend on traffic which depends of routes chosen, then feedback
  - If may result in instability