**The Link-State (LS) Routing Algorithm**

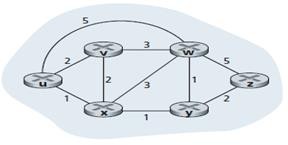
**Recall that in a link-state algorithm, the network topology and all link costs are known, that is, available as input to the LS algorithm. In practice this is accomplished by having each node broadcast link-state packets to *all* other nodes in the network, with each link-state packet containing the identities and costs of its attached links.**

**The link-state routing algorithm we present below is known as Dijkstra’s algorithm, named after its inventor.**

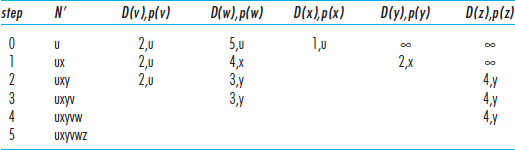
**Dijkstra’s algorithm computes the least-cost path from one node (the source, which we will refer to as *u*) to all other nodes in the network. Dijkstra’s algorithm is iterative and has the property that after the *k*th iteration of the algorithm, the least-cost paths are known to *k* destination nodes, and among the least-cost paths to all destination nodes, these *k* paths will have the *k* smallest costs. Let us define the following notation:**

* ***D*(*v*): cost of the least-cost path from the source node to destination *v* as of this iteration of the algorithm.**
* ***p*(*v*): previous node (neighbor of *v*) along the current least-cost path from the source to *v.***
* ***N*: subset of nodes; *v* is in *N*\_ if the least-cost path from the source to *v* is definitively known.**

**Example:** consider a network and compute the least-cost paths from ***u*** to all possible destinations



Solution:



Let’s consider the few first steps in detail.

* **In the initialization step,** the currently known least-cost paths from *u* to its directly attached neighbors,

*v, x,* and *w,* are initialized to 2, 1, and 5, respectively.

Note in particular that the cost to *w* is set to 5 (even though we will soon see that a lesser-cost path does indeed exist) since this is the cost of the direct (one hop) link from *u* to *w.* The costs to *y* and *z* are set to infinity because they are not directly connected to *u.*

* **In the first iteration:**we look among those nodes not yet added to the set *N* and find that node with the least cost as of the end of the previous iteration. That node is *x,* with a cost of 1, and thus *x* is added to the set *N*.. The cost of the path to *v* is unchanged. The cost of the path to *w* (which was 5 at the end of the initialization) through node *x* is found to have a cost of 4. Hence this lower-cost path is selected and *w’*s predecessor along the shortest path from *u* is set to *x*. Similarly, the cost to *y* (through *x*) is computed to be 2, and the table is updated accordingly.
* **In the second iteration**:nodes *v* and *y* are found to have the least-cost paths (2), and we break the tie arbitrarily and add *y* to the set *N*\_ so that *N*\_ now contains *u, x,* and *y.*

**The forwarding table in a node, say node *u*,** can then be constructed from this information by storing, for each destination, the next-hop node on the least-cost path from *u* to the destination

