Distributed Approach:

↳ distributed approach is similar to centralized approach.

The main difference is the link cost of a link received by one node may be different from another node since the information is disseminated in an asynchronous manner.

$d_{km}^{i}(t)$ → link cost at node i at time t

is denoted by $d_{km}^{i}(t)$

(or) $d_{km}^{i}(t)$ → cost of the link from $k-m$ at node i at time t

$D_{ij}(t)$ → minimum distance from i to j, time-dependent

→ The communication of the link cost information in a distributed manner.

Dijkstra's Shortest path first algorithm:

(distributed approach)

1. Discover nodes in the network N.

   cost of link k-m. $d^i_{km}(t)$ → node i at its

   time of computation t.

2. Start with source node i in the Permanent list of

   nodes considered.

   (ie) $S = \{i\}$

   $S' = \{$ rest of the nodes $\}$ → tentative list.

   $D_{ij}(t) = d^i_j(t)$ for all $j \in S'$.

3. Identify a neighboring node (intermediary) k not in

   the current list S with the minimum cost

   path from node i.

   (ie) find $k \in S'$ such that $D_{ik}(t) = \min_{m \in S'} D_{im}(t)$

   Add k to the permanent list S (ie) $S = S \cup \{k\}$.

   drop k from the tentative list S' (ie) $S' = S' \setminus \{k\}$

   If S' is empty stop. ↓ all nodes in S' except node k.

4. Consider neighboring nodes N_k, of the intermediary k (but do not consider nodes already in S) to check for improvement in the minimum cost path

(c) for $j \in N_k \cap S'$

$$D_{ij}(t) = \min \{ D_{ij}(t), D_{ik}(t) + d_{kj}(t) \}$$

Goto Step 2.

Determination of the next hop is important in many networking environments.

Next hop → refers to the next directly connected node, to reach the destination j

→ Next hop should be the optimal path.

H_{ij} → Next hop from i for destination j

finally, in many situations, the shortest path to a specific destination j, instead of being to all destinations, is sufficient to compute.

Dijkstra's Shortest path first algorithm (with tracking of next hop)

Computation at time t

S = {i} // permanent list, start with source node i

S' = N \ {i} // tentative list of the nodes.

for (j in S') do

if ($d_{ij}(t) < \infty$) then // if i is directly connected to j

$D_{ij}(t) = d_{ij}(t)$

$H_{ij} = j$ // Set i's next hop to be j

else

$D_{ij}(t) = \infty$

$H_{ij} = -1$ // next hop not set

endif

endfor

while (S' is not empty) do // while tentative list is not empty.

$D_{temp} = \infty$ // find minimum cost neighbor k

for (m in S') do

if ($D_{im}(t) < D_{temp}$) then

$D_{temp} = D_{im}(t)$

endif $k = m$;

endfor

$S = S \cup \{k\}$ // add to permanent list

$S' = S' \setminus \{k\}$ // delete from tentative list

for (j in $N_k \cap S'$) do

if ($D_{ij}(t) > D_{ik}(t) + d_{kj}(t)$) then

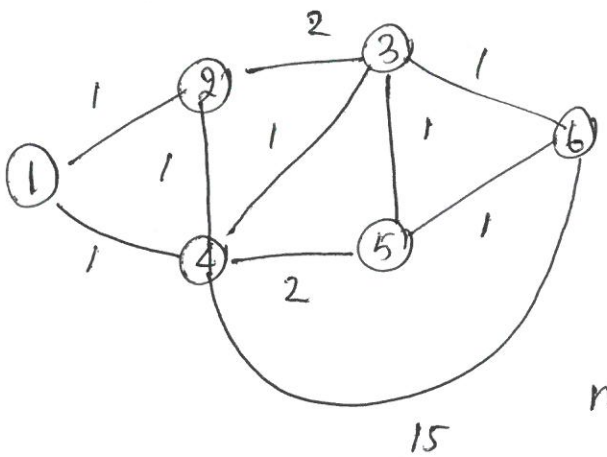
$D_{ij}(t) = D_{ik}(t) + d_{kj}(t)$

$H_{ij} = H_{ik}$ // next hop for destination j in hop from k.

end if

end for

end while.



neighboring nodes of node 1 is node 2 and 4

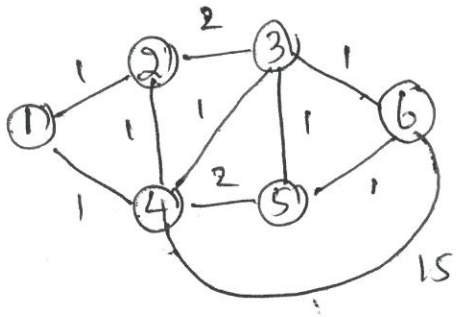
D_{12}, D_{14}

neighboring node of 2 and node 4 is $D_{23}, D_{43}, D_{45}, D_{46}$ D_{24}. .. etc.

The steps are same as the centralized approach.

$D_{ij}(t) = \min \{ D_{ij}(t), D_{ik}(t) + d_{kj}(t) \}$.

Iteration	List, S	D_{12}	path	D_{13}	Path	D_{14}	Path	D_{15}	path	D_{16}	path
1	{1}	1	1-2	∞	-	1	1-4	∞	-	∞	-
2	{1,2}	1	1-2	3	1-2-3	1	1-4	∞	-	∞	-
3	{1,2,4}	1	1-2	2	1-4-3	1	1-4	3	1-4-5	16	1-4-6
4	{1,2,4,3}	1	1-2	2	1-4-3	1	1-4	3	1-4-5	3	1-4-3-6
5	{1,2,4,3,5}	1	1-2	2	1-4-3	1	1-4	3	1-4-5	3	1-4-3-6
6	{1,2,4,3,5,6}	1	1-2	2	1-4-3	1	1-4	3	1-4-5	3	1-4-3-6



$D_{12} = \min \{ 1, D_{14}(t) + d_{42} \}$

$= \min \{ 1, 2 \} = \underline{1}$.

Comparison of the Bellman - Ford algorithm and Dijkstra's Algorithm

	Bellman ford algorithm	Dijkstra's algorithm
1.	Computes the shortest path to one destination at a time.	→ Computes shortest paths to all destinations. (Shortest path tree)
2.	Intermediary node k is the overall node to find the next hops to node j.	→ Node k is an intermediary first determined and fixed and then the shortest path computation is done to ~~all~~ all j not already covered.
3.	Computational complexity is $O(LN)$ N → total no of nodes. L → total no of links.	→ Computational complexity is $O(N^2)$ but can be improved to $O(L+N \log N)$ using good data structure.
4.	If the network is fully connected, the no of bidirectional links = $\frac{N(N-1)}{2}$ Complexity $O(N^3)$	→ If the network is fully connected, the complexity is $O(N^2)$