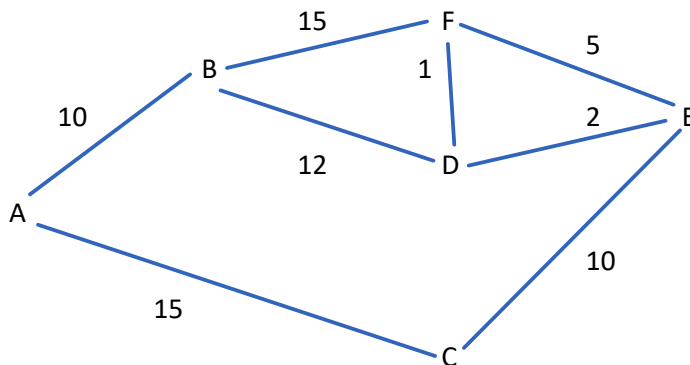


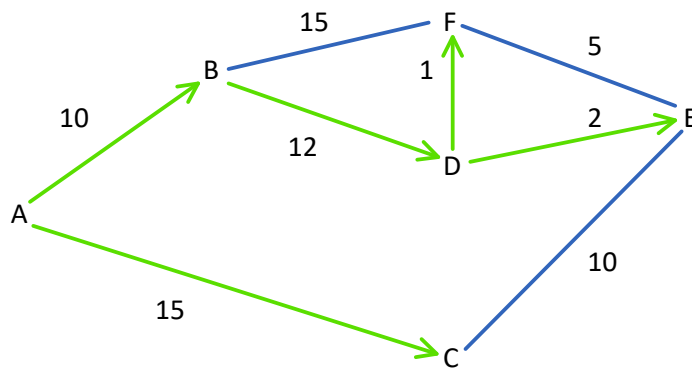
# Dijkstra's algorithm

For each vertex (or node) in a graph, Dijkstra's algorithm finds the shortest path from the source node to that vertex. Figure 1. shows a graph with vertices A - F and edges with distances of 10,15,12,1,2 and 5.



**Figure 1.** The graph in its initial state before the algorithm is applied.

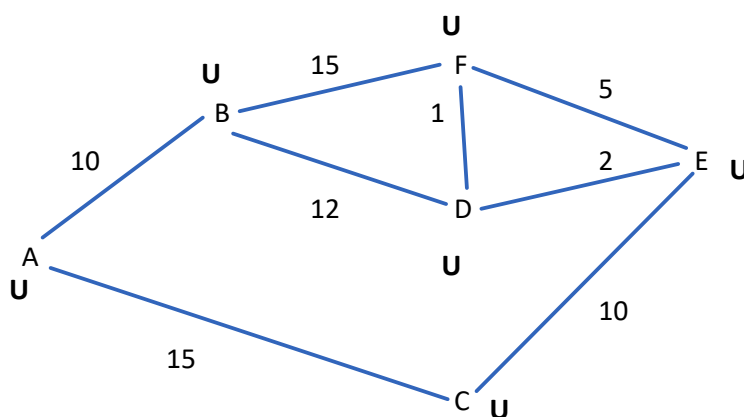
When the algorithm is finished, the graph will have calculated the shortest paths.



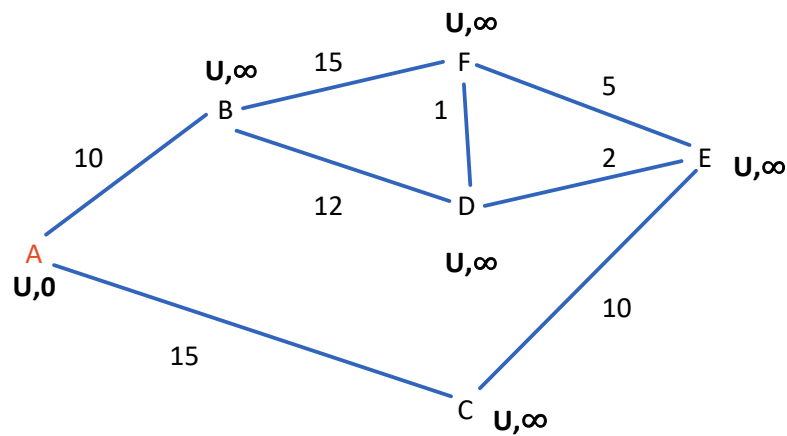
**Figure 2.** Graph with the shortest paths from A to all other vertices.

## Running the algorithm

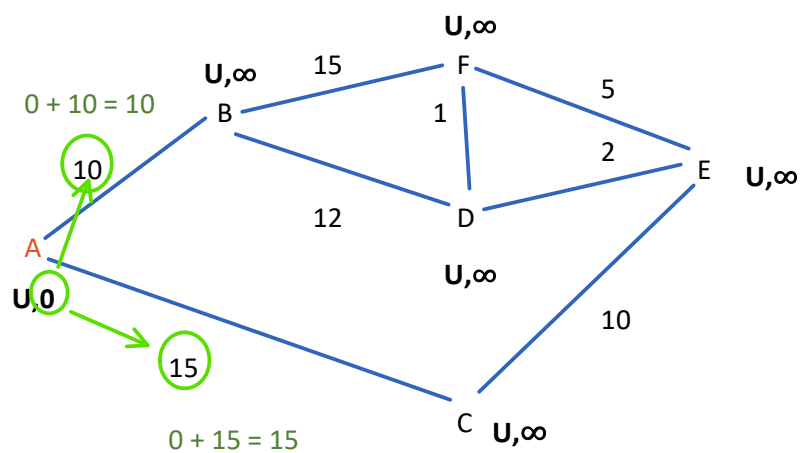
Iteration1.STEP1 - Set all the nodes to unvisited, marking them with a U.



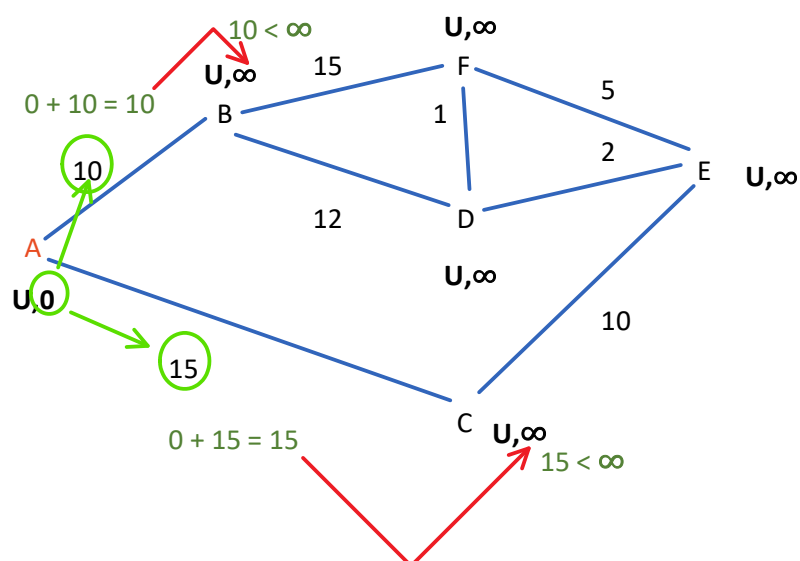
Iteration1.STEP2 - Assign every node a temporary distance - set it to zero for the source node, A, and set all other nodes to infinity. Set the current node to A.



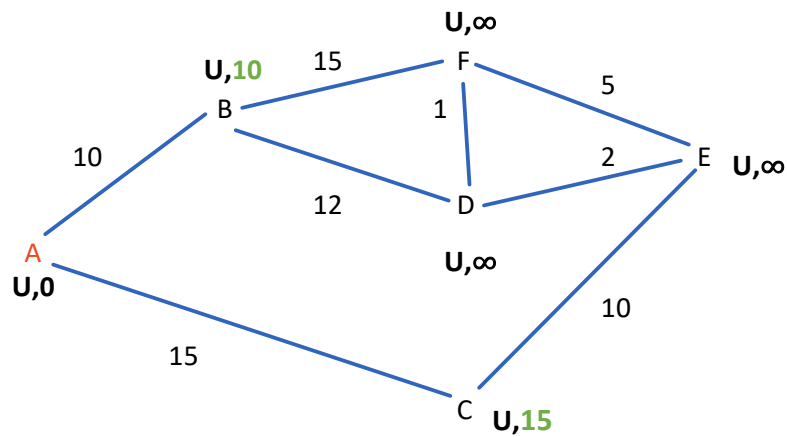
Iteration1.STEP3 - For each **unvisited** node *directly attached* to the current node, calculate the new temporary distance (follow the sequence of Figures 3a - 3c).



**Figure 3a.** Add the temporary distance on the current node to the edges of the unvisited nodes.

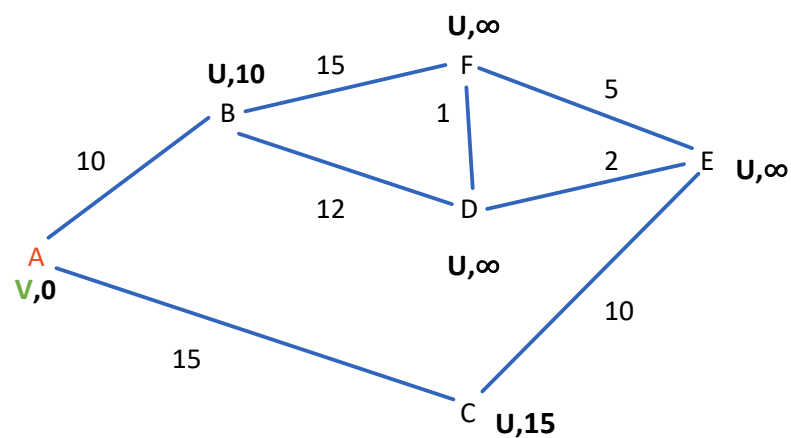


**Figure 3b.** Compare the new temporary distance with the temporary distance on the unvisited node. For each node, record the smaller value.



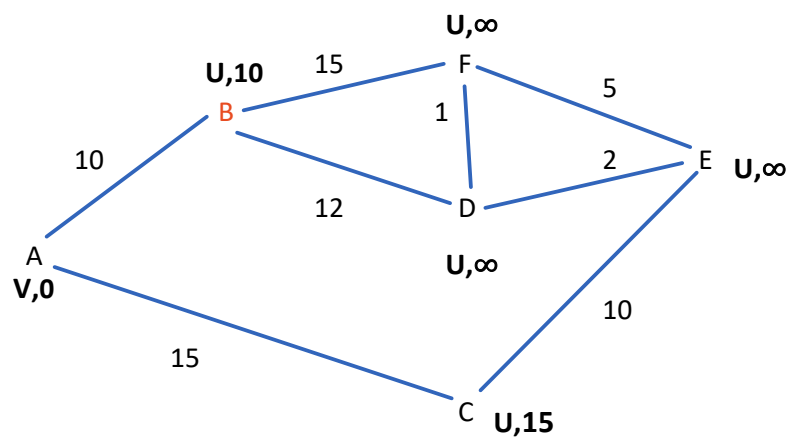
**Figure 3c.** Update the temporary distance on the unvisited node with the smaller value.

Iteration1.STEP4 - Mark the current node as visited.

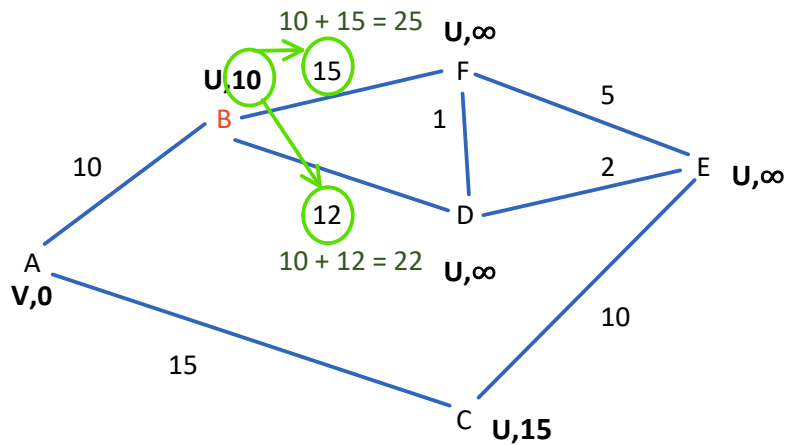


Iteration1.STEP5 - Stop the algorithm if all the nodes are visited. There are still unvisited nodes. Continue algorithm.

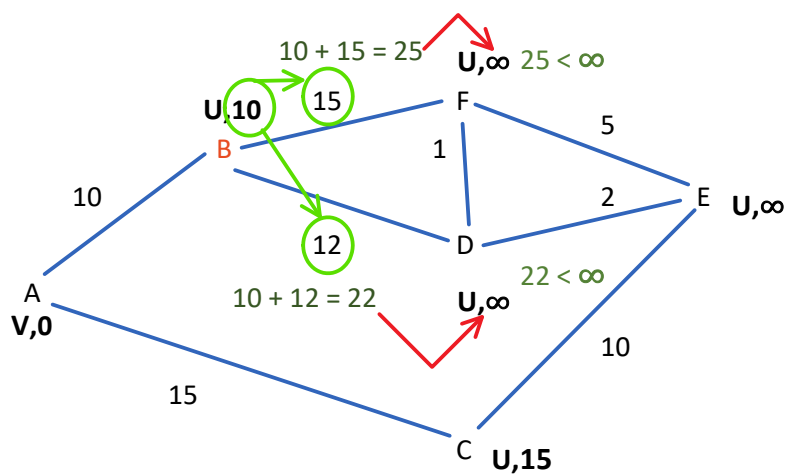
Iteration1.STEP6 - Set the current node to the node with the smallest temporary distance in the unvisited set. Start a new iteration, jumping to STEP3.



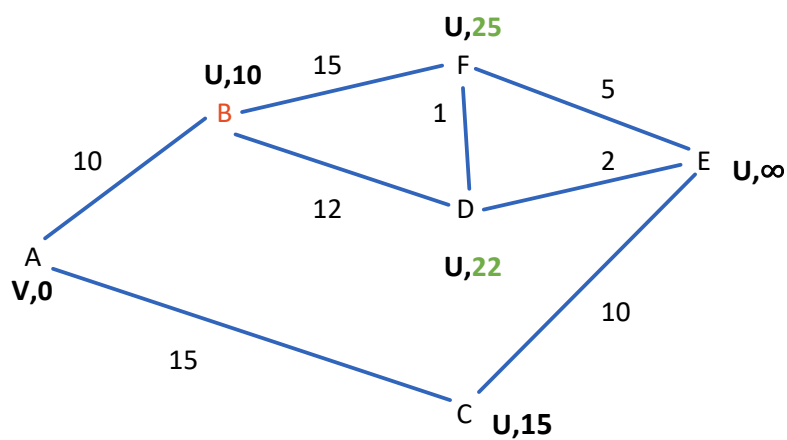
Iteration2.STEP3 - For each unvisited node directly attached to the current node, calculate the new temporary distance (follow the sequence of Figures 4a - 4c).



**Figure 4a.** Add the temporary distance on the current node to the edges of the unvisited nodes.

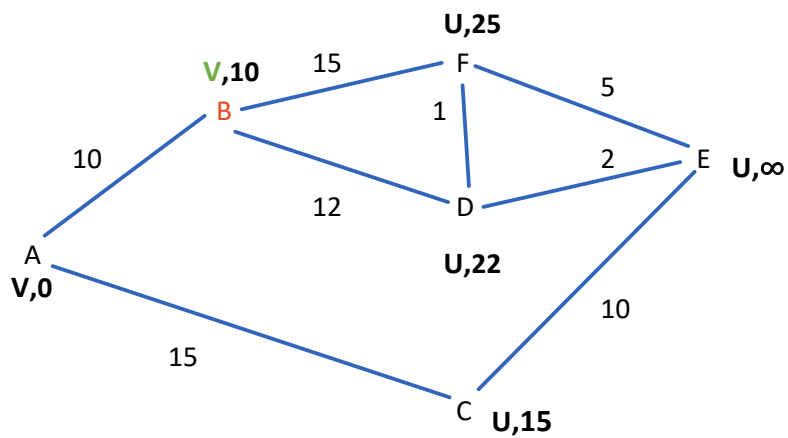


**Figure 4b.** Compare the new temporary distance with the temporary distance on the unvisited node. For each node, record the smaller value.



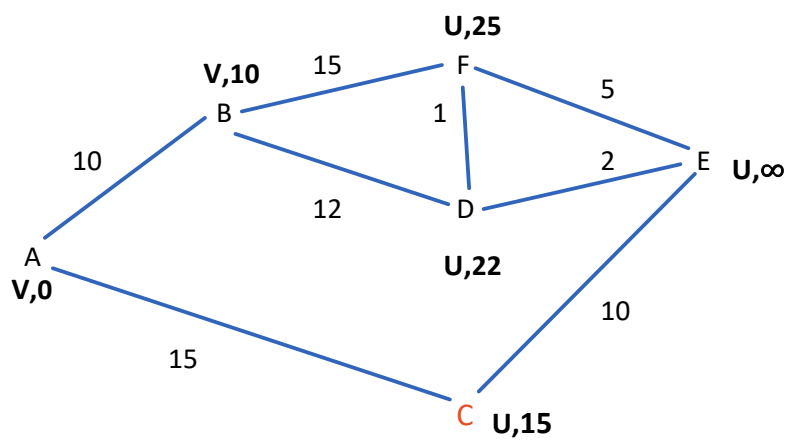
**Figure 4c.** Update the temporary distance on the unvisited node with the smaller value.

Iteration2.STEP4 - Mark the current node as visited.

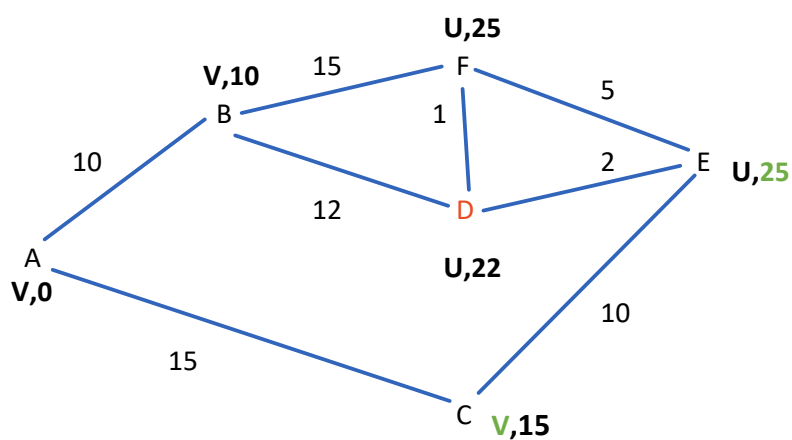


Iteration2.STEP5 - Stop the algorithm if all the nodes are visited. There are still unvisited nodes. Continue algorithm.

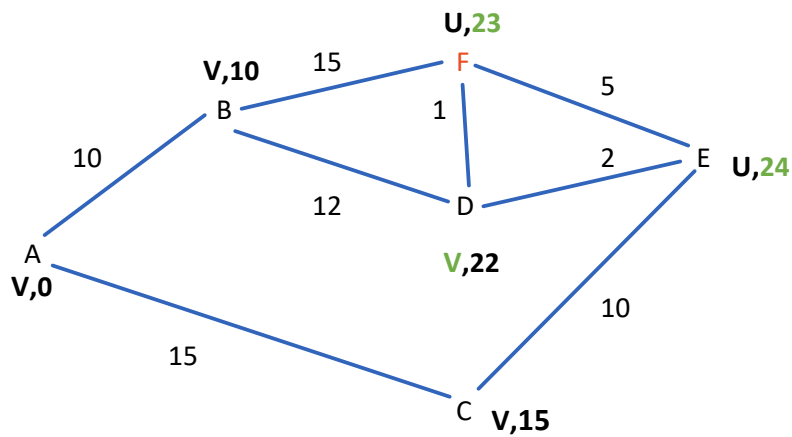
Iteration2.STEP6 - Set the current node to the node with the smallest temporary distance in the unvisited set. Start a new iteration, jumping to STEP3.



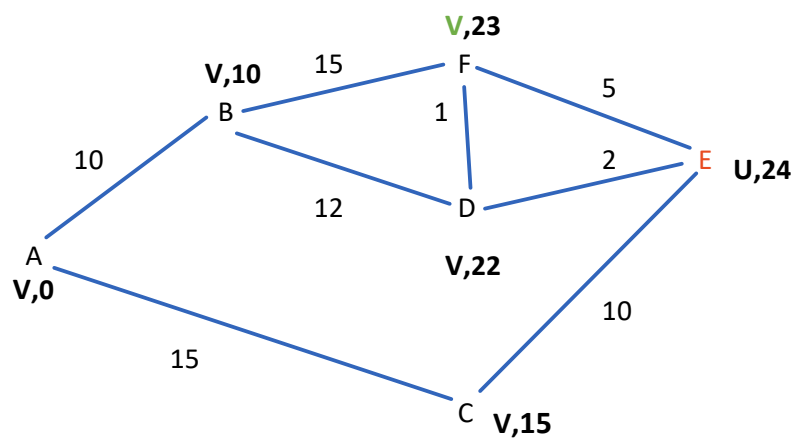
Iteration 3 - Run the same steps (3 - 6) as before. Updates are in green; the new current node is in red.



Iteration 4 - Run steps 3 - 6 to get the following result.



Iteration 5 - Run steps 3 - 6 again; the temporary distance on E is not updated ( $24 < 23 + 5$ ).



Iteration 6 - Run steps 3 - 5; the algorithm will stop. The temporary distance on a vertex represents the distance from A to it, along its shortest path.