Algorithm

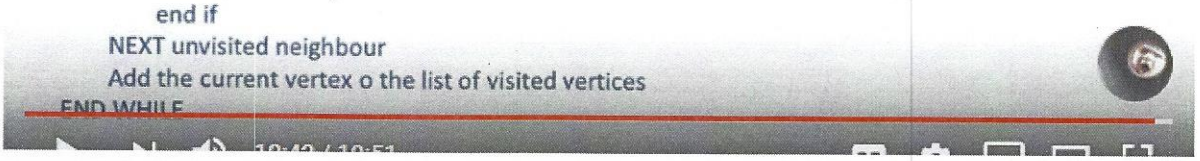
Let distance of start vertex from start vertex = 0
Let distance of all other vertices from start = ∞ (infinity)

Repeat
    Visit the unvisited vertex with the smallest known distance from the start vertex
    For the current vertex, examine its unvisited neighbours
    For the current vertex, calculate distance of each neighbour from start vertex
    If the calculated distance of a vertex is less than the known distance, update the shortest distance
    Update the previous vertex for each of the updated distances
    Add the current vertex to the list of visited vertices
Until all vertices visited

Algorithm

Let distance of start vertex from start vertex = 0
Let distance of all other vertices from start = ∞ (infinity)

WHILE vertices remain unvisited
    Visit unvisited vertex with smallest known distance from start vertex (call this 'current vertex')
    FOR each unvisited neighbour of the current vertex
        Calculate the distance from start vertex
        If the calculated distance of this vertex is less than the known distance
            Update shortest distance to this vertex
            Update the previous vertex with the current vertex
        end if
    NEXT unvisited neighbour
    Add the current vertex o the list of visited vertices
END WHILE



Algorithm [edit]

Let the node at which we are starting be called the **initial node**. Let the **distance of node *Y*** be the distance from the **initial node** to *Y*. Dijkstra's algorithm will assign some initial distance values and will try to improve them step by step.

1. Mark all nodes unvisited. Create a set of all the unvisited nodes called the *unvisited set*.
2. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes. Set the initial node as current.[13]
3. For the current node, consider all of its unvisited neighbours and calculate their *tentative* distances through the current node. Compare the newly calculated *tentative* distance to the current assigned value and assign the smaller one. For example, if the current node *A* is marked with a distance of 6, and the edge connecting it with a neighbour *B* has length 2, then the distance to *B* through *A* will be 6 + 2 = 8. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, keep the current value.
4. When we are done considering all of the unvisited neighbours of the current node, mark the current node as visited and remove it from the *unvisited set*. A visited node will never be checked again.
5. If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the *unvisited set* is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3.

When planning a route, it is actually not necessary to wait until the destination node is "visited" as above: the algorithm can stop once the destination node has the smallest tentative distance among all "unvisited" nodes (and thus could be selected as the next "current").