Edsger W. Dijkstra

Edsger Wybe Dijkstra was born in 1930 in Rotterdam, The Netherlands. His father was a chemist and his mother was a mathematician. In 1942, at 12, Dr. Dijkstra entered an elite high school where he studied classical Greek and Latin, French, German, English, biology, mathematics, physics and chemistry.

During the German occupation his parents sent him away for a brief period to the countryside. He earned degrees in mathematics and theoretical physics from the University of Leyden and a Ph.D. in computing science from the University of Amsterdam.

He taught at the Technical University of Eindhoven from 1963 to 1973 and at the University of Texas from 1984. He was widely known for his 1959 solution to the shortest-path problem; his algorithm is still used to determine the fastest way between two points, as in the routing of communication networks and in flight planning.

His research on the idea of mutual exclusion in communications led him to suggest in 1968 the concept of computer semaphores, which are used in virtually every modern operating system. A letter he wrote in 1968 was extremely influential in the development of structured programming. He received the Turing Award in 1972.

Another concept due to Dijkstra in the field of distributed computing is that of self-stabilization – an alternative way to ensure the reliability of the system. Dijkstra’s algorithm is used in SPF, Shortest Path First, which is used in the routing protocol OSPF, Open Shortest Path First.

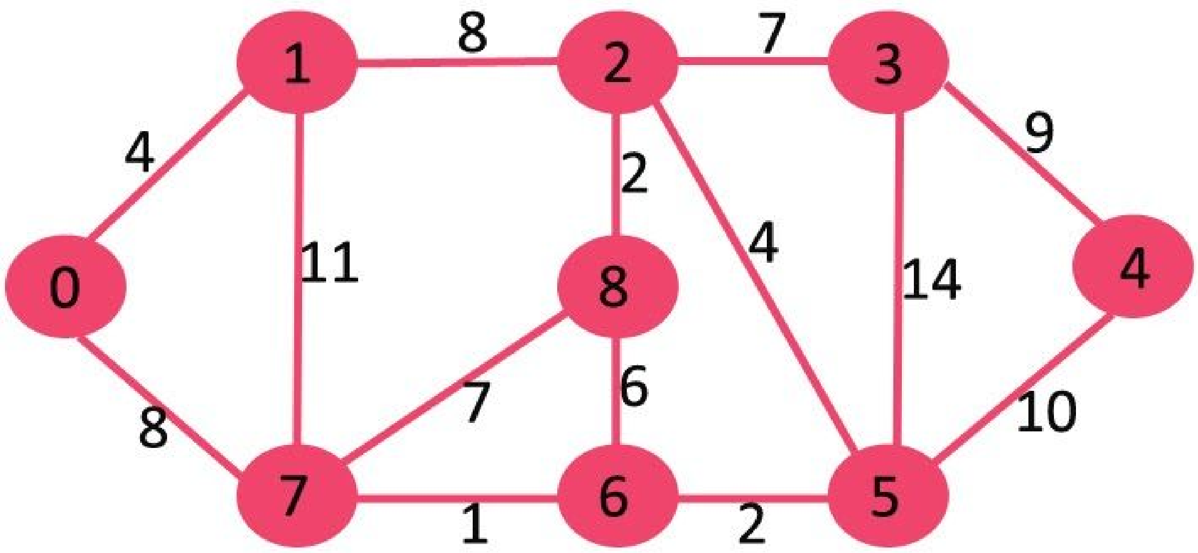
Of even greater importance was his solution to what he originally called the dining quintuple problem, but which later became known as the dining philosophers' problem.

Dr. Dijkstra, an advocate of an approach known as structured programming, wrote a short research note in the March 1968 edition of the journal Communications of the ACM that became legendary. Titled "The GO TO Considered Harmful," it argued against the complexity of a feature in programming languages like Fortran and Basic that permitted programmers to write convoluted programs that jump around haphazardly.

He is famed for coining the popular programming phrase “2 or more, use a for”, alluding to the fact that when you find yourself processing more than one instance of a data structure, it is time to encapsulate that logic inside a loop.

Dijkstra was known for his essays on programming; he was the first to make the claim that programming is so inherently difficult and complex that programmers need to harness every trick and abstraction possible in hopes of managing the complexity of it successfully.
Dijkstra believed that computer science was more abstract than programming; he once said, “Computer Science is no more about computers than astronomy is about telescopes.”

It took him three years to publish the method, which is now known simply as Dijkstra's algorithm. At the time, he said, algorithms were hardly considered a scientific topic.



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.

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, the current value will be kept.

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").