A Dutch computer scientist, ***Edsger Dijkstra***, in 1959, proposed an algorithm that can be applied to a weighted graph. The graph can either be directed or undirected with the condition that the graph needs to embrace a non-negative value on its every edge. He named this algorithm “Dijkstra’s Algorithm” at his name.

we will learn

1. Introduction to Graphs
1. What is Dijkstra’s Algorithm?
2. How to Implement the Dijkstra's Algorithm?
3. Working Example of Dijkstra's Algorithm
4. Applications of Dijkstra’s Algorithm
5. Advantages and Disadvantages of Dijkstra’s Algorithm

Introduction to Graphs

In simple words, graphs are data structures that are used to depict connections amidst a couple of elements where these elements are called nodes (or vertex) that generally real-time objects, persons or entities and connections amid nodes are termed as edges. Also, two nodes only get connected if there is an edge between them.

"A graph is essentially an interrelationship of nodes/vertices connected by edges."

Generally, graphs are suited to real-world applications, such as graphs can be used to illustrate a transportation system/network, where nodes represent facilities that transfer or obtain products and edges show routes or subways that connect nodes.

Graphs can be divided into two parts;

- **Undirected Graphs:** For every couple of associated nodes, if an individual could move from one node to another in both directions, then the graph is termed as an undirected graph.

- **Directed Graphs:** For every couple of associated graphs, if an individual could move from one node to another in a specific (single) direction, then the graph is known as the directed graph. In this case, arrows are implemented rather than simple lines in order to represent directed edges.

Weighted Graphs

The weight graphs are the graphs where edges of the graph have “a weight” or “cost” and also where weight could reflect distance, time, money or anything that displays the “association” amid a couple of nodes it links. These weights are an essential element under Dijkstra's Algorithm.

What is Dijkstra’s Algorithm?

What if you are provided with a graph of nodes where every node is linked to several other nodes with varying distance. Now, if you begin from one of the nodes in the graph, what is the shortest path to every other node in the graph?

Well simply explained, *an algorithm that is used for finding the shortest distance, or path, from starting node to target node in a weighted graph is known as Dijkstra's Algorithm*.

This algorithm makes a tree of the shortest path from the starting node, the source, to all other nodes (points) in the graph.

Dijkstra's algorithm makes use of weights of the edges for finding the path that minimizes the total distance (weight) among the source node and all other nodes. This algorithm is also known as the single-source shortest path algorithm.

Dijkstra's algorithm is the iterative algorithmic process to provide us with the shortest path from one specific starting node to all other nodes of a graph. It is different from the ***minimum spanning tree*** as the shortest distance among two vertices might not involve all the vertices of the graph.

It is important to note that Dijkstra's algorithm is only applicable when all weights are positive because, during the execution, the weights of the edges are added to find the shortest path.

And therefore if any of the weights are introduced to be negative on the edges of the graph, the algorithm would never work properly. However, some algorithms like the ***Bellman-Ford Algorithm*** can be used in such cases.

It is also a known fact that ***breadth-first search***(BFS) could be used for calculating the shortest path for an unweighted graph, or for a weighted graph that has the same cost at all its edges.

But if the weighted graph has unequal costs at all its edges, then BFS infers ***uniform-cost search*****.** Now what?

Instead of extending nodes in order of their depth from the root, uniform-cost search develops the nodes in order of their costs from the root. And a variant of this algorithm is accepted as Dijkstra's Algorithm.

Generally, Dijkstra's algorithm works on the principle of relaxation where an approximation of the accurate distance is steadily displaced by more suitable values until the shortest distance is achieved.

Also, the estimated distance to every node is always an overvalue of the true distance and is generally substituted by the least of its previous value with the distance of a recently determined path.

It uses a priority queue to greedily choose the nearest node that has not been visited yet and executes the relaxation process on all of its edges.

Example Involved

For example, an individual wants to calculate the shortest distance between the source, A, and the destination, D, while calculating a subpath which is also the shortest path between its source and destination. Let's see here how Dijkstra's algorithm works;

It works on the fact that any subpath, let say a subpath B to D of the shortest path between vertices A and D is also the shortest path between vertices B and D, i.e., each subpath is the shortest path.

Here, Dijkstra's algorithm uses this property in the reverse direction, that means, while determining distance, we overestimate the distance of each vertex from the starting vertex then inspect each node and its neighbours for detecting the shortest subpath to those neighbours.

This way the algorithm deploys a greedy approach by searching for the next plausible solution and expects that the end result would be the appropriate solution for the entire problem.

How to Implement the Dijkstra Algorithm?

Before proceeding the step-by-step process for implementing the algorithm, let us consider some essential characteristics of Dijkstra's algorithm;

- Basically, the Dijkstra's algorithm begins from the node to be selected, the source node, and it examines the entire graph to determine the shortest path among that node and all the other nodes in the graph.
- The algorithm maintains the track of the currently recognized shortest distance from each node to the source code and updates these values if it identifies another shortest path.
- Once the algorithm has determined the shortest path amid the source node to another node, the node is marked as "visited" and can be added to the path.
- This process is being continued till all the nodes in the graph have been added to the path, as this way, a path gets created that connects the source node to all the other nodes following the plausible shortest path to reach each node.

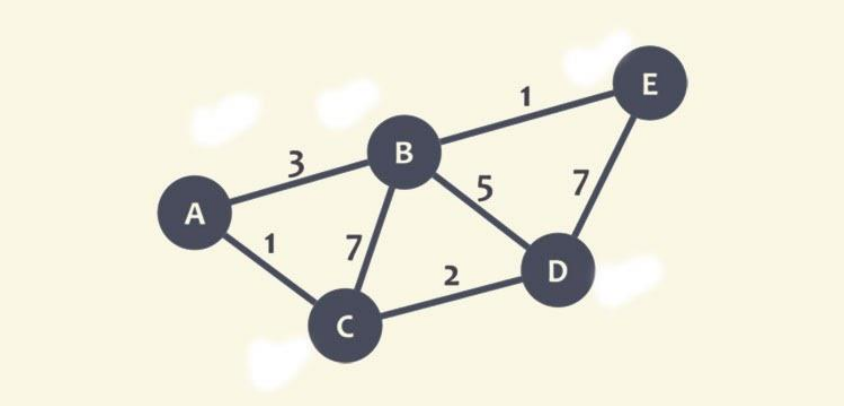
Now explaining the step-by-step process of algorithm implementation;

1. The very first step is to mark all nodes as unvisited,
2. Mark the picked starting node with a current distance of 0 and the rest nodes with infinity,
3. Now, fix the starting node as the current node,
4. For the current node, analyze all of its unvisited neighbours and measure their distances by adding the current distance of the current node to the weight of the edge that connects the neighbour node and current node,
5. Compare the recently measured distance with the current distance assigned to the neighbouring node and make it as the new current distance of the neighbouring node,
6. After that, consider all of the unvisited neighbours of the current node, mark the current node as visited,
7. If the destination node has been marked visited then stop, an algorithm has ended, and
8. Else, choose the unvisited node that is marked with the least distance, fix it as the new current node, and repeat the process again from step 4.

Working Example of Dijkstra's Algorithm

In the above section, you have gained the step by step process of Dijkstra's algorithm, now let's study the algorithm with an explained example.

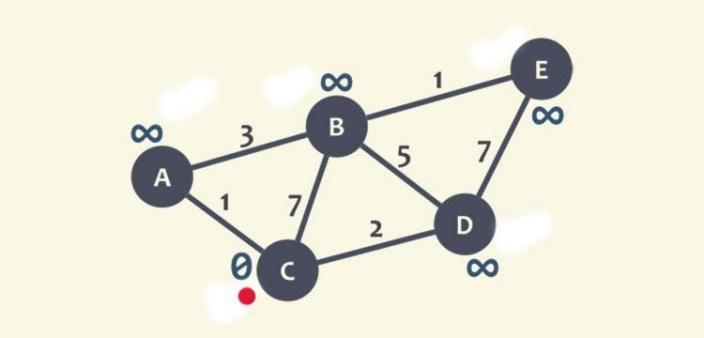
We will calculate the shortest path between node C and the other nodes in the graph.



Example of Dijkstra's Algorithm

1. During the execution of the algorithm, each node will be marked with its minimum distance to node C as we have selected node C.

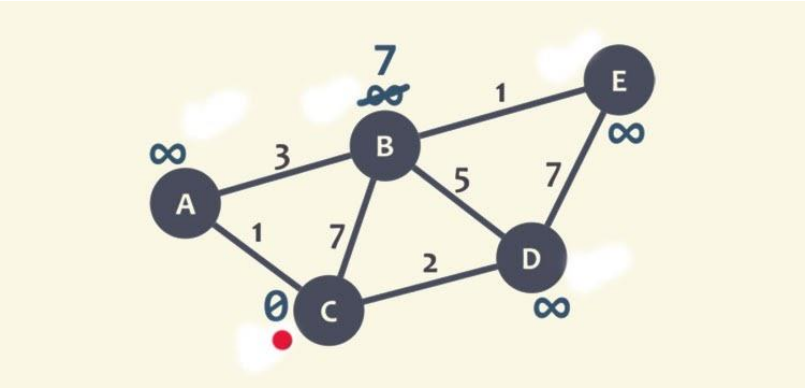
   In this case, the minimum distance is 0 for node C. Also, for the rest of the nodes, as we don't know this distance, they will be marked as infinity (∞), except node C (currently marked as red dot).



Graphical Representation of Node C as Current Node

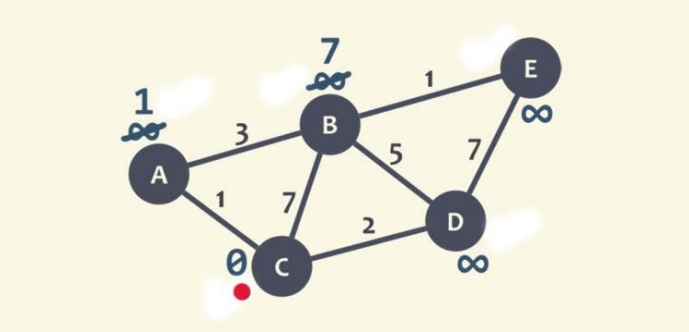
2. Now the neighbours of node C will be checked, i.e, node A, B, and D. We start with B, here we will add the minimum distance of current node (0) with the weight of the edge (7) that linked the node C to node B and get 0+ 7= 7.

Now, this value will be compared with the minimum distance of B (infinity), the least value is the one that remains the minimum distance of B, like in this case, 7 is less than infinity, and marks the least value to node B.



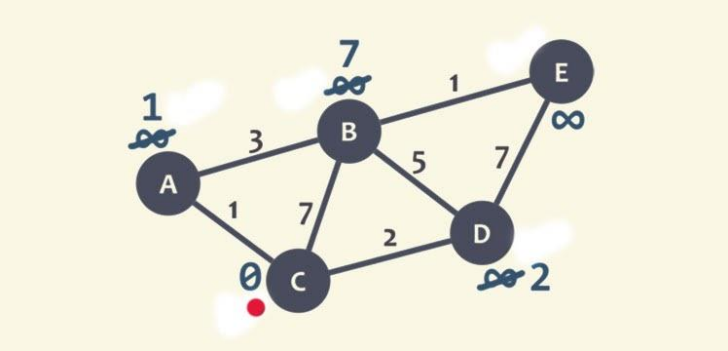
Assign Node B a minimum distance value

3. Now, the same process is checked with neighbour A. We add 0 with 1 (weight of edge that connects node C to A), and get 1. Again, 1 is compared with the minimum distance of A (infinity), and marks the lowest value.



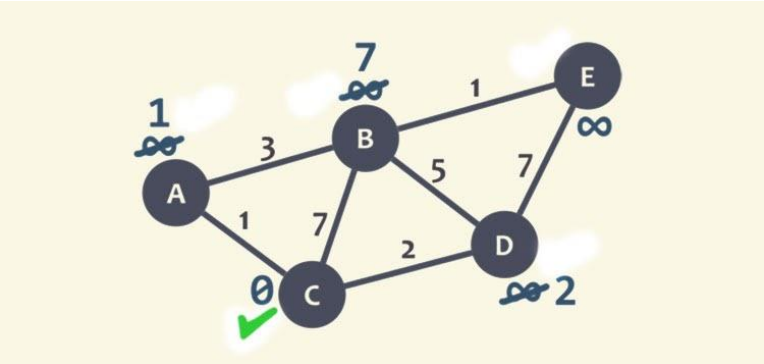
Assign Node A a minimum distance value

The same is repeated with node D, and marked 2 as lowest value at D.



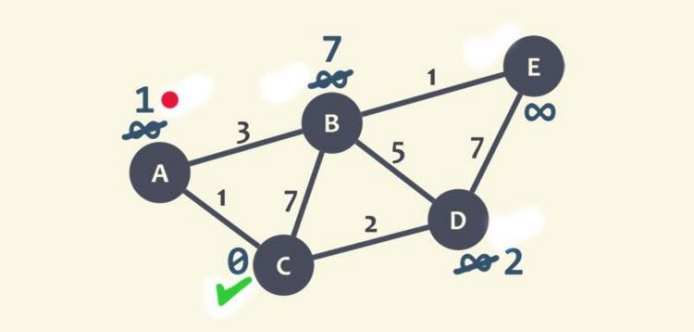
Assign Node D a minimum distance value

Since, all the neighbours of node C have checked, so node C is marked as visited with a green check mark.



Marked Node C as visited

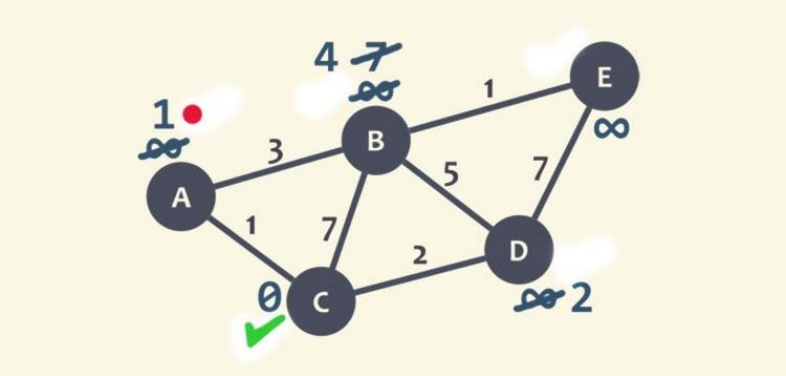
4. Now, we will select the new current node such that the node must be unvisited with the lowest minimum distance, or the node with the least number and no check mark. Here, node A is the unvisited with minimum distance 1, marked as current node with red dot.



Graphical Representation of Node A as Current Node

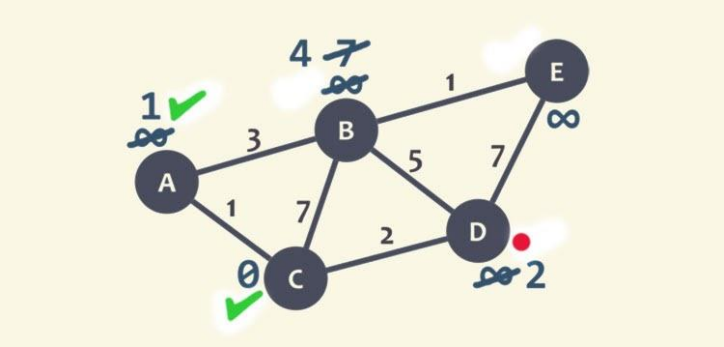
We repeat the algorithm, checking the neighbour of the current node while ignoring the visited node, so only node B will be checked.

For node B, we add 1 with 3 (weight of the edge connecting node A to B) and obtain 4. This value, 4, will be compared with the minimum distance of B, 7, and mark the lowest value at B as 4.



*Assign Node **B** a minimum distance value*

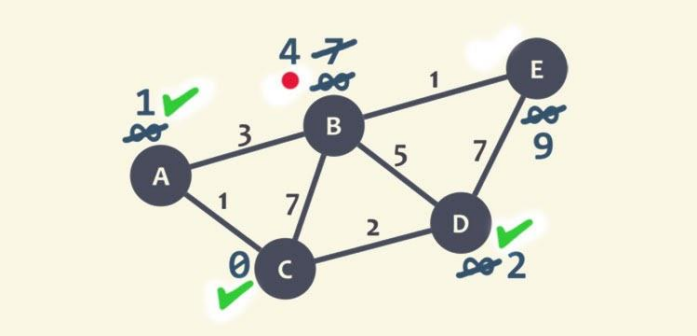
5. After this, node A marked as visited with a green check mark. The current node is selected as node D, it is unvisited and has a smallest recent distance. We repeat the algorithm and check for node B and E.



Graphical Representation of Node D as Current Node

For node B, we add 2 to 5, get 7 and compare it with the minimum distance value of B, since 7>4, so leave the smallest distance value at node B as 4.

For node E, we obtain 2+ 7= 9, and compare it with the minimum distance of E which is infinity, and mark the smallest value as node E as 9. The node D is marked as visited with a green check mark.

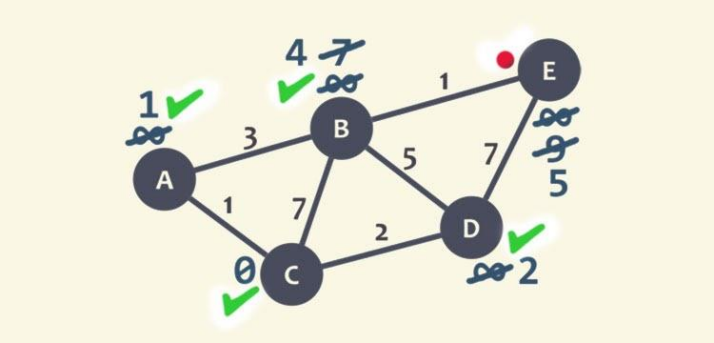


Marked Node D as visited

6. The current node is set as node B, here we need to check only node E as it is unvisited and the node D is visited. We obtain 4+ 1=5, compare it with the minimum distance of the node.

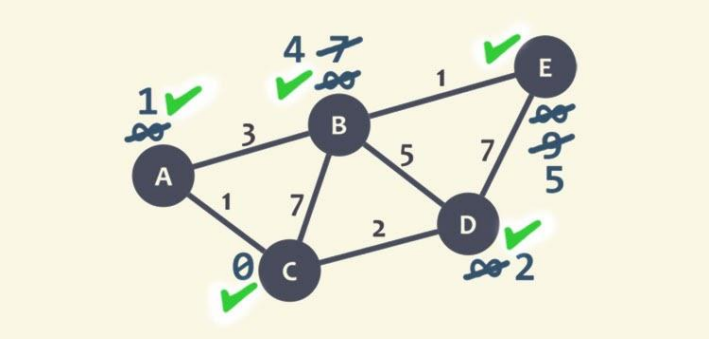
   As 9 > 5, leave the smallest value at node node E as 5.

   We mark D as visited node with a green check mark, and node E is set as the current node.



Marked Node B as visited

7. Since it doesn't have any unvisited neighbours, so there is not any requirement to check anything. Node E is marked as a visited node with a green mark.



Marked Node E as visited

So, we are done as no unvisited node is left. The minimum distance of each node is now representing the minimum distance of that node from node C.

Applications of Dijkstra's Algorithm

Before learning any algorithm, we should know the fundamental purpose of using an algorithm that could help us in real-world applications. Such as, for Dijkstra's algorithm, we are trying to find the solutions to least path-based problems.

For example, if a person wants to travel from city A to city B where both cities are connected with various routes. Which route commonly he/ she should choose?

Undoubtedly, we would adopt the route through which we could reach the destination with the least possible time, distance and even cost.

Further, with the discussion, it has various real-world use cases, some of the applications are the following:

- ***For map applications****, it is hugely deployed in measuring the least possible distance and check direction amidst two geographical regions like Google Maps, discovering map locations pointing to the vertices of a graph, calculating traffic and delay-timing, etc.*
- ***For telephone networks****, this is also extensively implemented in the conducting of data in networking and telecommunication domains for decreasing the obstacle taken place for transmission.*
- *Wherever addressing the need for shortest path explications either in the domain of robotics, transport, embedded systems, laboratory or production plants, etc, this algorithm is applied.*
- *Besides that, other applications are road conditions, road closures and construction, and IP routing to detect Open Shortest Path First.*

Advantages and Disadvantages of Dijkstra's Algorithm

Discussing some advantages of Dijkstra's algorithm;

1. One of the main advantages of it is its little complexity which is almost linear.
2. It can be used to calculate the shortest path between a single node to all other nodes and a single source node to a single destination node by stopping the algorithm once the shortest distance is achieved for the destination node.
3. It only works for directed-, weighted graphs and all edges should have non-negative values.

Despite various applications and advantages, Dijkstra's algorithm has disadvantages also, such as;

1. It does an obscured exploration that consumes a lot of time while processing,
2. It is unable to handle negative edges,
3. As it heads to the acyclic graph, so can't achieve the accurate shortest path, and
4. Also, there is a need to maintain tracking of vertices, have been visited.

Conclusion

Among many, we have discussed the Dijkstra algorithm used for finding the shortest path, however, one of the obstacles while implementing the algorithm on the internet is to provide a full representation of the graph to execute the algorithm as an individual router has a complete outline for all the routers on the internet.