**Dijkstra's Algorithm**

It differs from the minimum spanning tree because the shortest distance between two vertices might not include all the vertices of the graph.

## How Dijkstra's Algorithm works

Dijkstra's Algorithm works on the basis that any subpath B -> D of the shortest path A -> D between vertices A and D is also the shortest path between vertices B and D.

|  |
| --- |
| shortest subpath property is used by dijkstra's algorithm |
| Each subpath is the shortest path |

Djikstra used this property in the opposite direction i.e we overestimate the distance of each vertex from the starting vertex. Then we visit each node and its neighbors to find the shortest subpath to those neighbors.

The algorithm uses a greedy approach in the sense that we find the next best solution hoping that the end result is the best solution for the whole problem.

## Example of Dijkstra's algorithm

It is easier to start with an example and then think about the algorithm.

|  |
| --- |
| Start with a weighted graph |
| Start with a weighted graph |

|  |
| --- |
| Choose a starting vertex and assign infinity path values to all other devices |
| Choose a starting vertex and assign infinity path values to all other devices |

|  |
| --- |
| Go to each vertex and update its path length |
| Go to each vertex and update its path length |

|  |
| --- |
| If the path length of the adjacent vertex is lesser than new path length, don't update it |
| If the path length of the adjacent vertex is lesser than new path length, don't update it |

|  |
| --- |
| Avoid updating path lengths of already visited vertices |
| Avoid updating path lengths of already visited vertices |

|  |
| --- |
| After each iteration, we pick the unvisited vertex with the least path length. So we choose 5 before 7 |
| After each iteration, we pick the unvisited vertex with the least path length. So we choose 5 before 7 |

|  |
| --- |
| Notice how the rightmost vertex has its path length updated twice |
| Notice how the rightmost vertex has its path length updated twice |

|  |
| --- |
| Repeat until all the vertices have been visited |
| Repeat until all the vertices have been visited |

## Djikstra's algorithm pseudocode

We need to maintain the path distance of every vertex. We can store that in an array of size v, where v is the number of vertices.

We also want to be able to get the shortest path, not only know the length of the shortest path. For this, we map each vertex to the vertex that last updated its path length.

Once the algorithm is over, we can backtrack from the destination vertex to the source vertex to find the path.

A minimum priority queue can be used to efficiently receive the vertex with least path distance.

function dijkstra(G, S)

for each vertex V in G

distance[V] <- infinite

previous[V] <- NULL

If V != S, add V to Priority Queue Q

distance[S] <- 0

while Q IS NOT EMPTY

U <- Extract MIN from Q

for each unvisited neighbour V of U

tempDistance <- distance[U] + edge\_weight(U, V)

if tempDistance < distance[V]

distance[V] <- tempDistance

previous[V] <- U

return distance[], previous[]

## **Dijkstra's Algorithm Complexity**

Time Complexity: O(E Log V)

where, E is the number of edges and V is the number of vertices.

Space Complexity: O(V)

## **Dijkstra's Algorithm Applications**

* To find the shortest path
* In social networking applications
* In a telephone network
* To find the locations in the map