

Lecture

NOTE: FOR FURTHER DETAILS AND MORE COMPREHENSIVE STUDY, PLEASE SEE RECOMMENDED BOOKS OR INTERNET.

Dijkstra's Algorithm

Dijkstra's algorithm is Single Source Shortest Path algorithm that allows us to find the shortest path between any two vertices of a graph. 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 $B \rightarrow D$ of the shortest path $A \rightarrow D$ between vertices A and D is also the shortest path between vertices B and D.



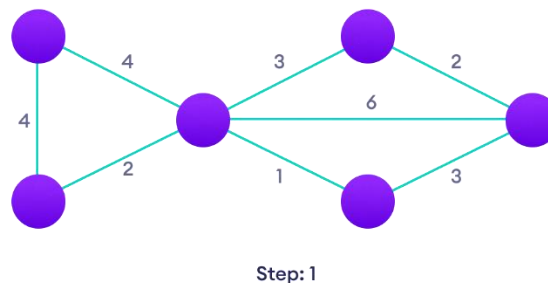
Each subpath is the shortest path

Dijkstra 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.

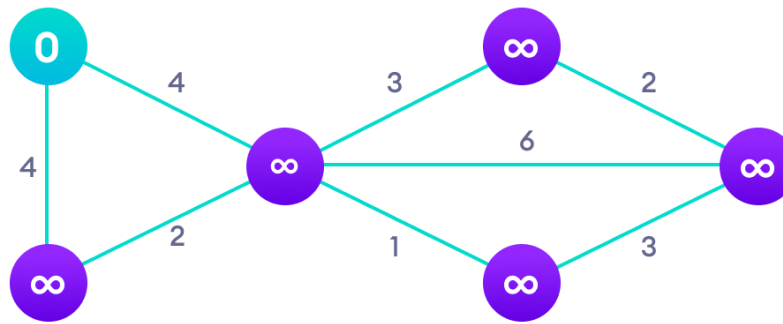
The algorithm uses a greedy approach in the sense that we find the next best solution hoping that the end result is the best solution for the whole problem.

Example of Dijkstra's algorithm

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

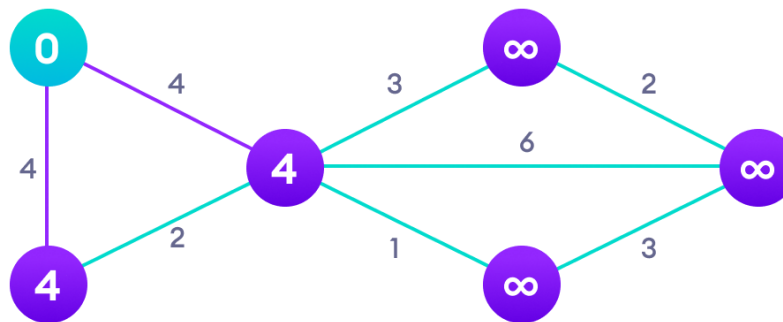


Start with a weighted graph



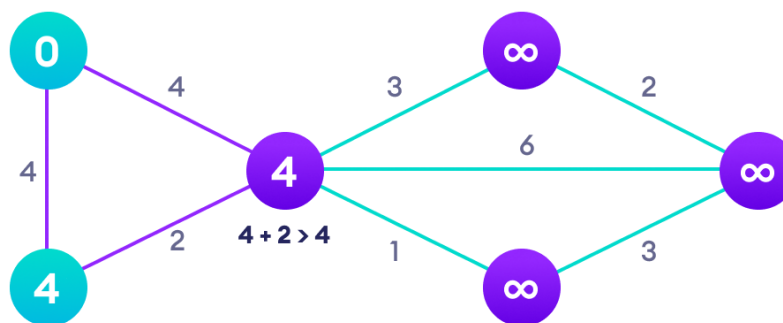
Step: 2

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



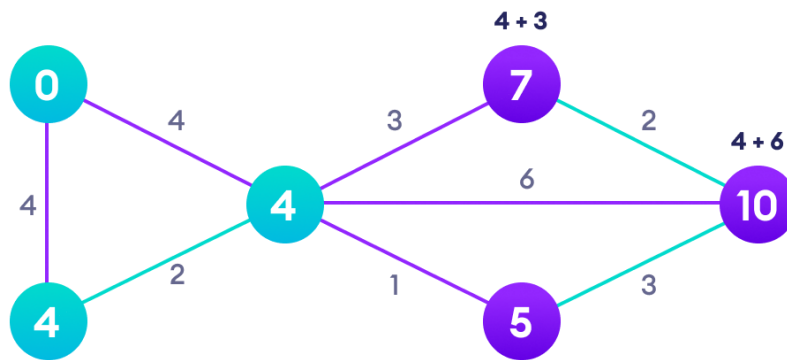
Step: 3

Go to each vertex and update its path length



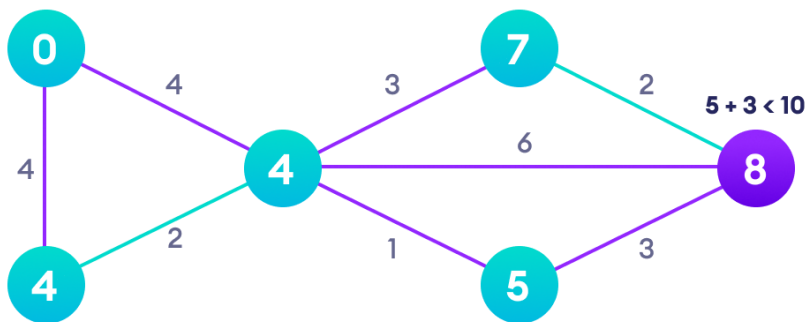
Step: 4

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



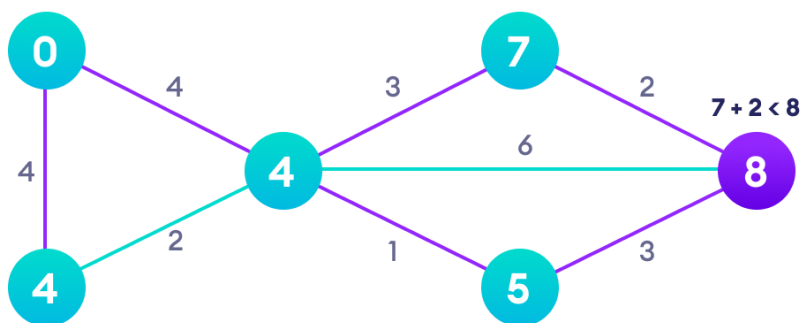
Step: 5

Avoid updating path lengths of already visited vertices



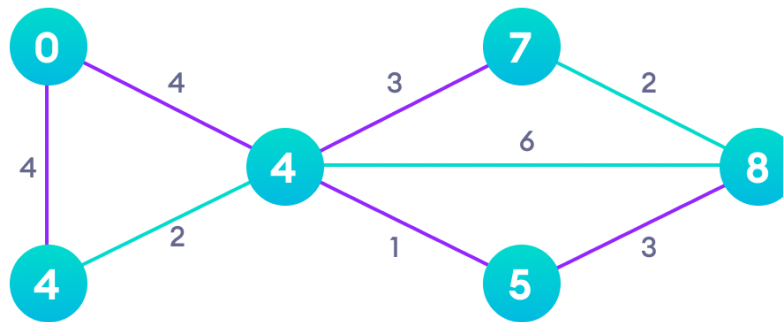
Step: 6

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



Step: 7

Notice how the rightmost vertex has its path length updated twice



Step: 8

Repeat until all the vertices have been visited

Dijkstra'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[]
```