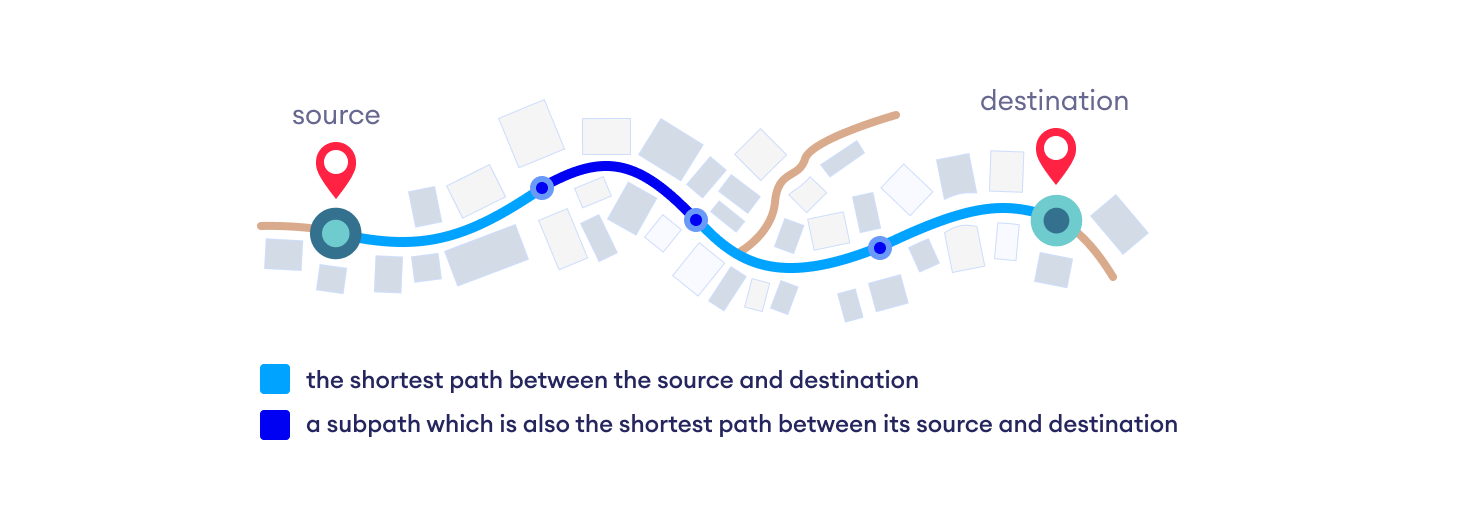
Dijkstra's Algorithm

Dijkstra's algorithm 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 -> D of the shortest path A -> D between vertices A and D is also the shortest path between vertices B and D.



Algorithm   
**1)** Create a set *sptSet* (shortest path tree set) that keeps track of vertices included in the shortest-path tree, i.e., whose minimum distance from the source is calculated and finalized. Initially, this set is empty.   
**2)** Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.   
**3)** While *sptSet* doesn’t include all vertices   
….**a)** Pick a vertex u which is not there in *sptSet* and has a minimum distance value.   
….**b)** Include u to *sptSet*.   
….**c)** Update distance value of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if the sum of distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

***How to implement the above algorithm?***

# Python program for Dijkstra's single

# source shortest path algorithm. The program is

# for adjacency matrix representation of the graph

# Library for INT\_MAX

**import** sys

**class** Graph():

**def** \_\_init\_\_(self, vertices):

        self.V **=** vertices

        self.graph **=** [[0 **for** column **in** range(vertices)]

**for** row **in** range(vertices)]

**def** printSolution(self, dist):

        print("Vertex \tDistance from Source")

**for** node **in** range(self.V):

**print**(node, "\t", dist[node])

    # A utility function to find the vertex with

    # minimum distance value, from the set of vertices

    # not yet included in shortest path tree

**def** minDistance(self, dist, sptSet):

        # Initialize minimum distance for next node

        min **=** sys.maxsize

        # Search not nearest vertex not in the

        # shortest path tree

**for** u **in** range(self.V):

**if** dist[u] < min **and** sptSet[u] **==** False:

                min **=** dist[u]

                min\_index **=** u

**return** min\_index

    # Function that implements Dijkstra's single source

    # shortest path algorithm for a graph represented

    # using adjacency matrix representation

**def** dijkstra(self, src):

        dist **=** [sys.maxsize] **\*** self.V

        dist[src] **=** 0

        sptSet **=** [False] **\*** self.V

**for** cout **in** range(self.V):

            # Pick the minimum distance vertex from

            # the set of vertices not yet processed.

            # x is always equal to src in first iteration

            x **=** self.minDistance(dist, sptSet)

            # Put the minimum distance vertex in the

            # shortest path tree

            sptSet[x] **=** True

            # Update dist value of the adjacent vertices

            # of the picked vertex only if the current

            # distance is greater than new distance and

            # the vertex in not in the shortest path tree

**for** y **in** range(self.V):

**if** self.graph[x][y] > 0 **and** sptSet[y] **==** False **and** \

                dist[y] > dist[x] **+** self.graph[x][y]:

                        dist[y] **=** dist[x] **+** self.graph[x][y]

        self.printSolution(dist)

# Driver program

g **=** Graph(9)

g.graph **=** [[0, 4, 0, 0, 0, 0, 0, 8, 0],

        [4, 0, 8, 0, 0, 0, 0, 11, 0],

        [0, 8, 0, 7, 0, 4, 0, 0, 2],

        [0, 0, 7, 0, 9, 14, 0, 0, 0],

        [0, 0, 0, 9, 0, 10, 0, 0, 0],

        [0, 0, 4, 14, 10, 0, 2, 0, 0],

        [0, 0, 0, 0, 0, 2, 0, 1, 6],

        [8, 11, 0, 0, 0, 0, 1, 0, 7],

        [0, 0, 2, 0, 0, 0, 6, 7, 0]

        ];

g.dijkstra(0);

# This code is contributed by Divyanshu Mehta and Updated by Pranav Singh Sambyal