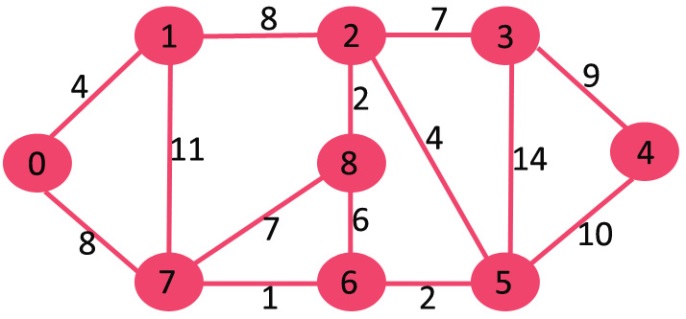
Dijkstra’s shortest path algorithm | Greedy Algo-7

Given a graph and a source vertex in the graph, find shortest paths from source to all vertices in the given graph.

Dijkstra’s algorithm is very similar to [Prim’s algorithm for minimum spanning tree](https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/). Like Prim’s MST, we generate a*SPT (shortest path tree)* with given source as root. We maintain two sets, one set contains vertices included in shortest path tree, other set includes vertices not yet included in shortest path tree. At every step of the algorithm, we find a vertex which is in the other set (set of not yet included) and has a minimum distance from the source.

Below are the detailed steps used in Dijkstra’s algorithm to find the shortest path from a single source vertex to all other vertices in the given graph.  
Algorithm  
**1)** Create a set *sptSet* (shortest path tree set) that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from 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 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 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.

Let us understand with the following example:  
[](https://www.geeksforgeeks.org/wp-content/uploads/Fig-11.jpg)

The set *sptSet* is initially empty and distances assigned to vertices are {0, INF, INF, INF, INF, INF, INF, INF} where INF indicates infinite. Now pick the vertex with minimum distance value. The vertex 0 is picked, include it in *sptSet*. So *sptSet*becomes {0}. After including 0 to *sptSet*, update distance values of its adjacent vertices. Adjacent vertices of 0 are 1 and 7. The distance values of 1 and 7 are updated as 4 and 8. Following subgraph shows vertices and their distance values, only the vertices with finite distance values are shown. The vertices included in SPT are shown in green colour.

[](https://www.geeksforgeeks.org/wp-content/uploads/MST1.jpg)

Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). The vertex 1 is picked and added to sptSet. So sptSet now becomes {0, 1}. Update the distance values of adjacent vertices of 1. The distance value of vertex 2 becomes 12.

[](https://www.geeksforgeeks.org/wp-content/uploads/DIJ2.jpg)

Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). Vertex 7 is picked. So sptSet now becomes {0, 1, 7}. Update the distance values of adjacent vertices of 7. The distance value of vertex 6 and 8 becomes finite (15 and 9 respectively).  
[](https://www.geeksforgeeks.org/wp-content/uploads/DIJ3.jpg)

Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). Vertex 6 is picked. So sptSet now becomes {0, 1, 7, 6}. Update the distance values of adjacent vertices of 6. The distance value of vertex 5 and 8 are updated.

[](https://www.geeksforgeeks.org/wp-content/uploads/DIJ4.jpg)

We repeat the above steps until *sptSet*does include all vertices of given graph. Finally, we get the following Shortest Path Tree (SPT).

[](https://www.geeksforgeeks.org/wp-content/uploads/DIJ5.jpg)

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

[**Recommended: Please solve it on “*PRACTICE* ” first, before moving on to the solution.**](https://practice.geeksforgeeks.org/problems/shortest-path-from-1-to-n/0)

We use a boolean array sptSet[] to represent the set of vertices included in SPT. If a value sptSet[v] is true, then vertex v is included in SPT, otherwise not. Array dist[] is used to store shortest distance values of all vertices.

* C++
* Java
* Python
* C#

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| # 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):            # Initilaize minimum distance for next node          min = sys.maxint            # Search not nearest vertex not in the          # shortest path tree          for v in range(self.V):              if dist[v] < min and sptSet[v] == False:                  min = dist[v]                  min\_index = v            return min\_index        # Funtion that implements Dijkstra's single source      # shortest path algorithm for a graph represented      # using adjacency matrix representation      def dijkstra(self, src):            dist = [sys.maxint] \* 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.              # u is always equal to src in first iteration              u = self.minDistance(dist, sptSet)                # Put the minimum distance vertex in the              # shotest path tree              sptSet[u] = 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 shotest path tree              for v in range(self.V):                  if self.graph[u][v] > 0 and sptSet[v] == False and \                  dist[v] > dist[u] + self.graph[u][v]:                          dist[v] = dist[u] + self.graph[u][v]            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 |

**Output:**

Vertex Distance from Source

0 0

1 4

2 12

3 19

4 21

5 11

6 9

7 8

8 14

**Notes:**  
**1)** The code calculates shortest distance, but doesn’t calculate the path information. We can create a parent array, update the parent array when distance is updated (like [prim’s implementation](https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/)) and use it show the shortest path from source to different vertices.

**2)** The code is for undirected graph, same dijkstra function can be used for directed graphs also.

**3)** The code finds shortest distances from source to all vertices. If we are interested only in shortest distance from the source to a single target, we can break the for the loop when the picked minimum distance vertex is equal to target (Step 3.a of the algorithm).

**4)** Time Complexity of the implementation is O(V^2). If the input [graph is represented using adjacency list](https://www.geeksforgeeks.org/graph-and-its-representations/), it can be reduced to O(E log V) with the help of binary heap. Please see  
[Dijkstra’s Algorithm for Adjacency List Representation](https://www.geeksforgeeks.org/greedy-algorithms-set-7-dijkstras-algorithm-for-adjacency-list-representation/) for more details.

**5)** Dijkstra’s algorithm doesn’t work for graphs with negative weight edges. For graphs with negative weight edges, [Bellman–Ford algorithm](http://en.wikipedia.org/wiki/Bellman-Ford_algorithm) can be used, we will soon be discussing it as a separate post.

[Dijkstra’s Algorithm for Adjacency List Representation](https://www.geeksforgeeks.org/greedy-algorithms-set-7-dijkstras-algorithm-for-adjacency-list-representation/)

[Printing Paths in Dijkstra’s Shortest Path Algorithm](https://www.geeksforgeeks.org/printing-paths-dijkstras-shortest-path-algorithm/)

[Dijkstra’s shortest path algorithm using set in STL](https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-using-set-in-stl/)

Please