Dijsktra's algorithm

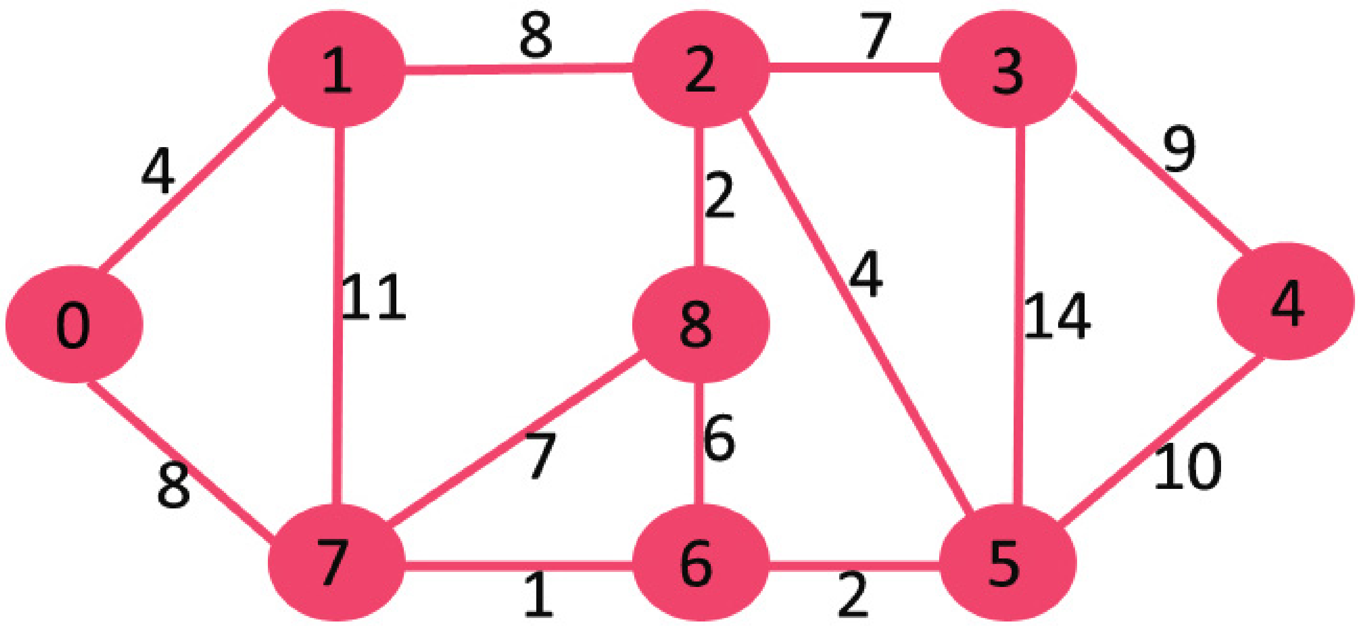
Greedy Algorithms | Set 7 (Dijkstra’s shortest path algorithm)

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

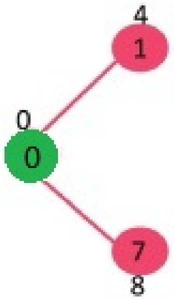
Dijkstra’s algorithm is very similar to [Prim’s algorithm for minimum spanning tree]. Like Prim’s MST, we generate a *SPT (shortest path tree)* with given source as root. We maintain two sets, one set contains vertices included in shortest path tree, other set includes vertices not yet included in shortest 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 minimum distance from 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.

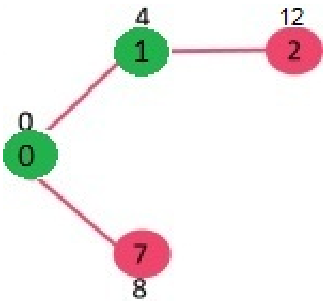
Let us understand with the following example:



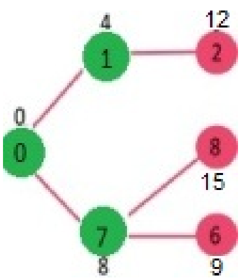
The set *sptSet*is initially empty and distances assigned to vertices are {0, INF, INF, INF, INF, INF, INF, INF} where INF indicates infinite. Now pick the vertex with minimum distance value. The vertex 0 is picked, include it in *sptSet*. So *sptSet* becomes {0}. After including 0 to *sptSet*, update 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 color.



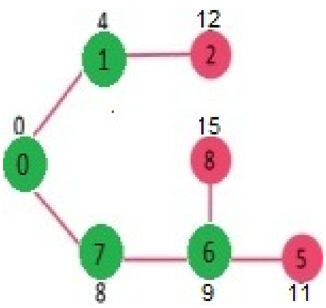
Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). The vertex 1 is picked and added to sptSet. So sptSet now becomes {0, 1}. Update the distance values of adjacent vertices of 1. The distance value of vertex 2 becomes 12.



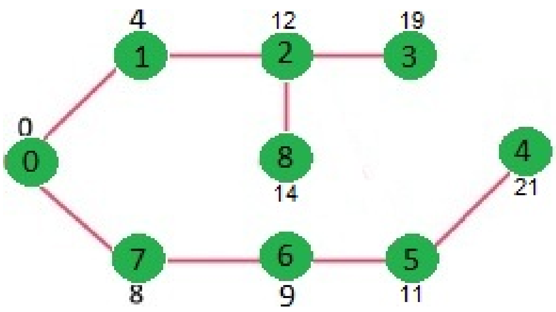
Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). Vertex 7 is picked. So sptSet now becomes {0, 1, 7}. Update the distance values of adjacent vertices of 7. The distance value of vertex 6 and 8 becomes finite (15 and 9 respectively).



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.



We repeat the above steps until *sptSet* doesn't include all vertices of given graph. Finally, we get the following Shortest Path Tree (SPT).



How to implement the above algorithm?

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/C++
- Java

```
// A C / C++ program for Dijkstra's single source shortest path
algorithm.// The program is for adjacency matrix representation of the
graph#include <stdio.h>#include <limits.h>// Number of vertices in the
graph#define V 9// A utility function to find the vertex with minimum
distance value, from// the set of vertices not yet included in shortest
path tree
int minDistance(int dist[], bool sptSet[])
{
    // Initialize min value
    int min = INT_MAX, min_index;
    for (int v = 0; v < V; v++)
      if (sptSet[v] == false && dist[v] <= min)
          min = dist[v], min_index = v;
    return min_index;
}// A utility function to print the constructed distance array
int printSolution(int dist[], int n)
{
    printf("Vertex   Distance from Source\n");
    for (int i = 0; i < V; i++)
        printf("%d \t\t %d\n", i, dist[i]);
}// Funtion that implements Dijkstra's single source shortest path
algorithm// for a graph represented using adjacency matrix representation
void dijkstra(int graph[V][V], int src)
{
     int dist[V];     // The output array.  dist[i] will hold the shortest
                      // distance from src to i
```

```
     bool sptSet[V]; // sptSet[i] will true if vertex i is included in
shortest
                     // path tree or shortest distance from src to i is
finalized
     // Initialize all distances as INFINITE and stpSet[] as false
     for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;
     // Distance of source vertex from itself is always 0
     dist[src] = 0;
     // Find shortest path for all vertices
     for (int count = 0; count < V-1; count++)
     {
       // Pick the minimum distance vertex from the set of vertices not
       // yet processed. u is always equal to src in first iteration.
       int u = minDistance(dist, sptSet);
       // Mark the picked vertex as processed
       sptSet[u] = true;
       // Update dist value of the adjacent vertices of the picked vertex.
       for (int v = 0; v < V; v++)
         // Update dist[v] only if is not in sptSet, there is an edge from
         // u to v, and total weight of path from src to  v through u is
         // smaller than current value of dist[v]
         if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
                                       && dist[u]+graph[u][v] < dist[v])
            dist[v] = dist[u] + graph[u][v];
     }
     // print the constructed distance array
     printSolution(dist, V);
}// driver program to test above function
int main()
{
   /* Let us create the example graph discussed above */
   int graph[V][V] = {{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, 0, 10, 0, 2, 0, 0},
                      {0, 0, 0, 14, 0, 2, 0, 1, 6},
                      {8, 11, 0, 0, 0, 0, 1, 0, 7},
                      {0, 0, 2, 0, 0, 0, 6, 7, 0}
                     };
```

```
    dijkstra(graph, 0);
    return 0;
}
```