```
% Dijkstra algorithm

% Caleb Kennett

% input is weight matrix with rows as nodes
% and column as connecting node

% output is shortest distances to final node from start node
% and the previous node that lead to that distance.




n = 13; % matrix of weights should then be n by n


w_actual =[0 5 8 1 999 999 999 999 999 999 999 999 999;
           5 0 4 999 2 11 2 999 999 999 999 999 999;
           8 4 0 6 999 3 10 1 999 999 999 999 999;
           1 999 6 0 999 999 9 2 5 999 999 999 999;
           999 2 999 999 0 10 999 999 999 6 999 999 999;
           999  11 3 999 10 0 4 999 999 3 9 999 999;
           999 2 10 9 999 4 0 11 999 5 22 10 999;
           999 999 1 2 999 999 11 0 7 999 7 8 999;
           999 999 999 5 999 999 999 7 0 999 999 1 999;
           999 999 999 999 6 3 5 999 999 0 8 999 2;
           999 999 999 999 999 9 22 7 999 8 0 11 5;
           999 999 999 999 999 999 10 8 1 999 11 0 23;
           999 999 999 999 999 999 999 999 999 2 5 23 0];
w_actual( w_actual == 999) = Inf;  % 999 set to Inf

w= w_actual;
w(w==0)=Inf;


vertex = n;
source = 1;
destination = 13;


prev = ones(1,vertex);

visted = zeros(1,n);
% 0/1 binary variable indicating whether
%           the node has been visited.

dist = w(1,:);% the distance from node 1.
dist(source) = 0;

%%%%%%%%%
```

```
u =1;

while visted(destination) ~= 1

    %Find the unvisited node with the lowest distance.
    % Set it as current node and mark it as visited
    if visted(u) ~= 1 % if unvisted

        [~,lowest_dist_node]= min(w(u,2:n));
        visted(u) = 1; % mark as visted

    else

        VISTED = visted*999 + 1;

        weight = VISTED.*w(u,:);% mask

        [~,lowest_dist_node]= min(weight(2:n));
        visted(u) = 1; % mark as visted

    end
    u = lowest_dist_node+1;
    %%%%%%

    for v = 1:vertex
        if w(u,v) ~= Inf % find all links that join to that current
 node.
            alt = dist(u) + w(u,v) ; %  dist from source to u
                                     % + length from u to v
            if alt < dist(v)  %If this is less than the stored
 distance

                dist(v) = alt;% update the distance.
                prev(v) = u;

            end

        end
    end


end

shortest_path = [];
current = destination;

while current ~= source

    shortest_path = cat(2,current, shortest_path);

    current = prev(current );

end
```

```
all_nodes = [1:13];
output=[all_nodes;dist;prev;visted]

shortest_path = cat(2,source,shortest_path) % shortest path
```

```
output =

    1     2     3     4     5     6     7     8     9    10    11
 12    13
    0     5     4     1     7     7     7     3     6    10    10
 7    12
    1     1     8     1     2     3     2     4     4     6     8
 9    10
    1     1     1     1     1     1     1     1     1     0     1
 1     1


shortest_path =

    1     4     8     3     6    10    13
```

Published with MATLAB® R2018b