|  |  |
| --- | --- |
| 1: | **function** Dijkstra(Graph, source): |
| 2: | **for each** vertex v in Graph: | // Initialization |
| 3: | dist[v] := infinity | // initial distance from source to vertex v is set to infinite |
| 4: | previous[v] := undefined | // Previous node in optimal path from source |
| 5: | dist[source] := 0 | // Distance from source to source |
| 6: | Q := the set of all nodes in Graph | // all nodes in the graph are unoptimized - thus are in Q |
| 7: | **while** Q **is not** empty: | // main loop |
| 8: | u := node in Q with smallest dist[ ] |  |
| 9: | remove u from Q |  |
| 10: | **for each** neighbor v of u: | // where v has not yet been removed from Q. |
| 11: | alt := dist[u] + dist\_between(u, v) |  |
| 12: | **if** alt < dist[v] | // Relax (u,v) |
| 13: | dist[v] := alt |  |
| 14: | previous[v] := u |  |
| 15: | **return** previous[ ] |  |

Pseudocode of bellman ford algorithm:

BELLMAN-FORD(G,w,s)

INITIALIZE-SINGLE-SOURCE(G,s)

for i = 1 to |G.V| - 1

for each edge(u,v) ∈ G.E

RELAX(u,v,w)

for each edge(u,v) ∈ G.E

if v.d > u.d + w(u,v)

return FALSE

return TRUE

Algorithm details about Dijkstra algorithm:

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 (s=shortest p=path t=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(initialization).  
**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.

Example:



The set sptSe tis initially empty and distances assigned to vertices are {0, INF, INF, INF, INF, INF, INF, INF} where INF indicates infinite where 0 for the starting point because starting point to staring point that means own distance is 0.

Now there should pick the vertex having minimum distance value. The vertex 0 is picked, include it in sptSet. SosptSetbecomes {0} and only one element is inserted into set.

Now update distance values of its adjacent vertices.0’s adjacent vertices are 1 and 7. The distance values of 1 is updated as 4 and 7 is updated as 8.



This subgraph shows vertices and their distance values as indicated at the top of the vertices,. The vertices included in SPT are shown in another color here it is green.

Now it should pick the vertex having minimum distance value and not already included in the set of sptSET.The vertex 1 is picked and added to sptSet. So sptSet now becomes {0, 1} because at 1 the value is 4 and at 7 the value s is 8 and the minimum value is 4 that’s why it is selected.



Vertices 1’s adjacent value is updated and the distance value of vertex 2 is now 12;

As previous system now we should pick the vertex with minimum distance value that are not in the set.Vertex 7 is picked. So sptSet is now {0, 1, 7}.



Vertices 7’s adjacent value is updated and the distance value of vertex 6 is now 15 and 8 is now 9.

Pick the vertex having minimum distance value that is not already included in sptSET.Vertex 6 is picked. So sptSet is now becomes Update the distance values of adjacent vertices of 6. The distance value of vertex 5 and 8 are updated and their value is 11 and 15 respectively..



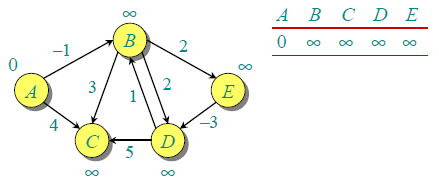
This process repeats this steps until sptSetdoesn’t include all vertices of given graph which shortest path is needed. The result is:

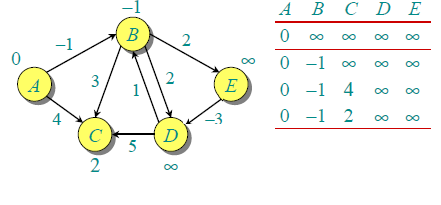
[](http://d1gjlxt8vb0knt.cloudfront.net/wp-content/uploads/DIJ5.jpg)

Example of bellman ford algorithm:

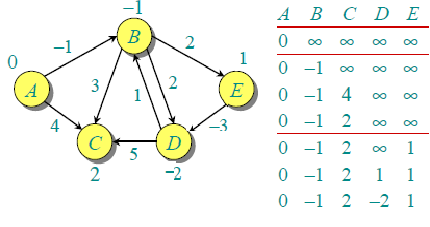
The images are taken from [this](http://www.cs.arizona.edu/classes/cs445/spring07/ShortestPath2.prn.pdf)source.

Let the given source vertex be 0. Initialize all distances as infinite, except the distance to source itself. Total number of vertices in the graph is 5, so all edges must be processed 4 times.

[](http://d1gjlxt8vb0knt.cloudfront.net/wp-content/uploads/bellman2.png)

Let all edges are processed in following order: (B,E), (D,B), (B,D), (A,B), (A,C), (D,C), (B,C), (E,D). We get following distances when all edges are processed first time. The first row in shows initial distances. The second row shows distances when edges (B,E), (D,B), (B,D) and (A,B) are processed. The third row shows distances when (A,C) is processed. The fourth row shows when (D,C), (B,C) and (E,D) are processed.  
[](http://d1gjlxt8vb0knt.cloudfront.net/wp-content/uploads/After1stIteration.png)

The first iteration guarantees to give all shortest paths which are at most 1 edge long. We get following distances when all edges are processed second time (The last row shows final values).

[](http://d1gjlxt8vb0knt.cloudfront.net/wp-content/uploads/seconditeration2.png)

The second iteration guarantees to give all shortest paths which are at most 2 edges long. The algorithm processes all edges 2 more times. The distances are minimized after the second iteration, so third and fourth iterations don’t update the distances.