# Shortest Path Routing

**Team Members:**

Abhirup Mandal(RA2011003011105)

Mukundaan S(RA2011003011099)

Shantanu Tripathi(RA2011003011085)

**Aim:**To implement shortest path routing.

**Explanation:**Consider that a network comprises of N vertices (nodes or network devices) that are connected by M edges (transmission lines). Each edge is associated with a weight, representing the physical distance or the transmission delay of the transmission line. The target of shortest path algorithms is to find a route between any pair of vertices along the edges, so the sum of weights of edges is minimum. If the edges are of equal weights, the shortest path algorithm aims to find a route having minimum number of hops.In computer networks, the shortest path algorithms aim to find the optimal paths between the network nodes so that routing cost is minimized. They are direct applications of the shortest path algorithms proposed in graph theory.

**Common Shortest Path Algorithms:**

Bellman Ford's Algorithm,

Dijiktra's Algorithm,

Floyd Warshall Algorithm.

**Bellman Ford Algorithm:**

Input − A graph representing the network; and a source node, s

Output − Shortest path from s to all other nodes.

Initialize distances from s to all nodes as infinite (∞); distance to itself as 0; an array dist[] of size |V| (number of nodes) with all values as ∞ except dist[s].

Calculate the shortest distances iteratively. Repeat |V|- 1 times for each node except s −

Repeat for each edge connecting vertices u and v −

If dist[v] > (dist[u] + weight of edge u-v), Then

Update dist[v] = dist[u] + weight of edge u-v

The array dist[] contains the shortest path from s to every other node.

**Dijkstra's Algorithm**

Input − A graph representing the network; and a source node, s

Output − A shortest path tree, spt[], with s as the root node

Initializations −An array of distances dist[] of size |V| (number of nodes), where dist[s] = 0 and dist[u] = ∞ (infinite), where u represents a node in the graph except s.

An array, Q, containing all nodes in the graph. When the algorithm runs into completion, Q will become empty.

An empty set, S, to which the visited nodes will be added. When the algorithm runs into completion, S will contain all the nodes in the graph.

Repeat while Q is not empty −

Remove from Q, the node, u having the smallest dist[u] and which is not in S. In the first run, dist[s] is removed.

Add u to S, marking u as visited.

For each node v which is adjacent to u, update dist[v] as −

If (dist[u] + weight of edge u-v) < dist[v], Then

Update dist[v] = dist[u] + weight of edge u-v

The array dist[] contains the shortest path from s to every other node.

2

**Floyd Warshall Algorithm**

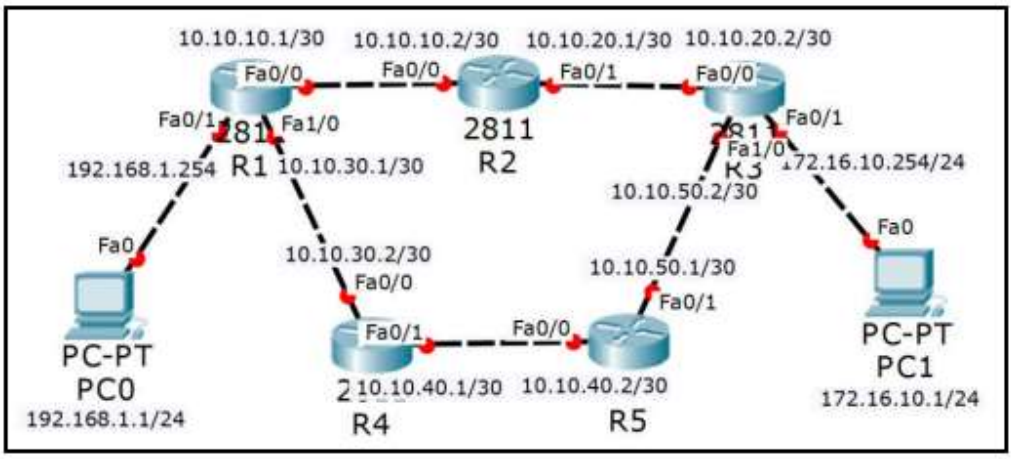
Input – A cost adjacency matrix, adj[][], representing the paths between the nodes in the network.Output – A shortest path cost matrix, cost[][], showing the shortest paths in terms of cost between each pair of nodes in the graph.

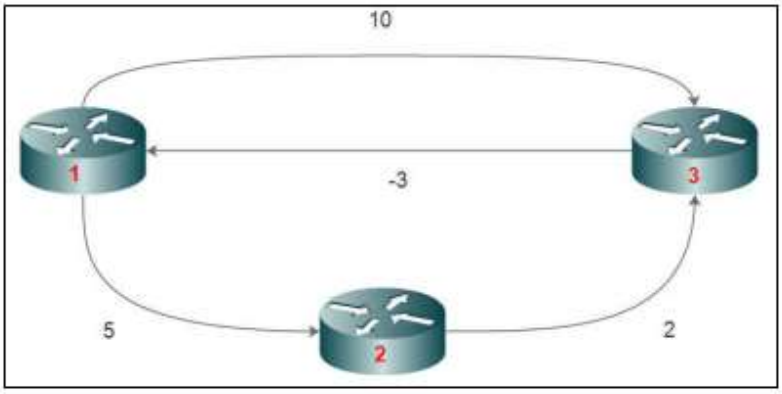
Populate cost[][] as follows:If adj[][] is empty Then cost[][] = ∞ (infinite)

Else cost[][] = adj[][]

N = |V|, where V represents the set of nodes in the network.Repeat for k = 1 to N –Repeat for i = 1 to N –Repeat for j = 1 to N –If cost[i][k] + cost[k][j] < cost[i][j], Then Update cost[i][j] := cost[i][k] + cost[k][j]The matrix cost[][] contains the shortest cost from each node, i , to every other node, j.

**Diagram:**





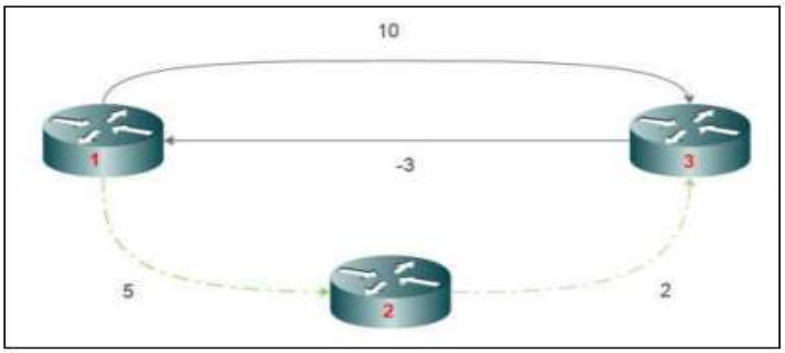




**Figure 5.** The shortest path is obtained.

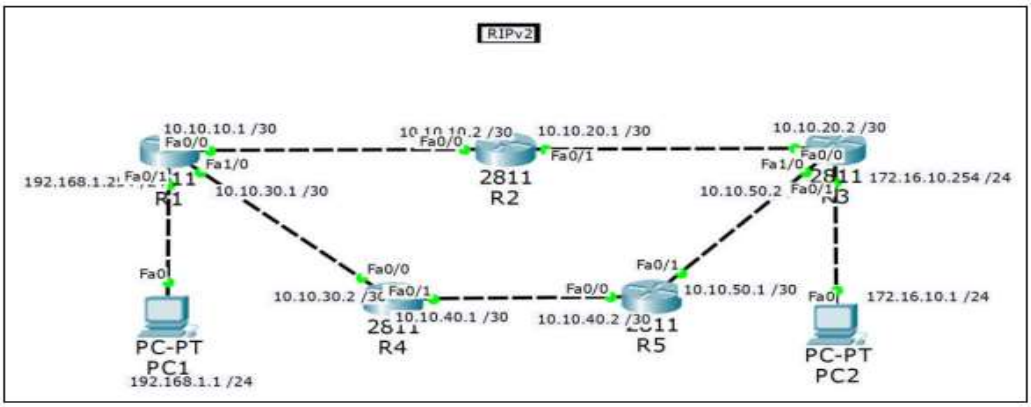



**Figure 6.** Shortest path.

## Code:

```
import java.util.Scanner;
public class Bellmanford {
private int distance[];
private int numb_vert;

public static final int MAX_VALUE=999;

public Bellmanford(int numb_vert)

{this.numb_vert=numb_vert;distance=new int[numb_vert+1];}

public void BellmanfordpEvaluation(int source,int adj_matrix[][])

{for(int node=1;node<=numb_vert;node++){     distance[node]=MAX_VALUE;}

distance[source]=0;

for(int node=1;node<=numb_vert;node++)

{ for(int src_node=1;src_node<=numb_vert;src_node++)

{

for(int dest_node=1;dest_node<=numb_vert;dest_node++)                        {
if(adj_matrix[src_node][dest_node]!=MAX_VALUE)

{if(distance[dest_node]>distance[src_node]+adj_matrix[src_node][dest_node])

distance[dest_node]=distance[src_node]+adj_matrix[src_node][dest_node];}

}}}

for(int src_node=1;src_node<=numb_vert;src_node++)

{

for(int dest_node=1;dest_node<=numb_vert;dest_node++)
```

```
{
if(adj_matrix[src_node][dest_node]!=MAX_VALUE)
{if(distance[dest_node]>distance[src_node]+adj_matrix[src_node][dest_node])
System.out.println("the graph contains negative edge cycle");
}}}
System.out.println("Routing table for Router"+source+"i");
System.out.println("destination distance/cost\t");
for(int vertex=1;vertex<=numb_vert;vertex++){
System.out.println(+vertex+"\t\t\t"+distance[vertex]);}}
public static void main(String...arg){
int numb_vert=0;Scanner scanner=new Scanner(System.in);
System.out.println("enter the number of vertices");
numb_vert=scanner.nextInt();
int adj_matrix[][]=new int[numb_vert+1][numb_vert+1];
System.out.println("enter the adjacency matrix");
for(int src_node=1;src_node<=numb_vert;src_node++)

                {
for(int dest_node=1;dest_node<=numb_vert;dest_node++)
{adj_matrix[src_node][dest_node]=scanner.nextInt();
if(src_node==dest_node){
adj_matrix[src_node][dest_node]=0;
continue;
if(adj_matrix[src_node][dest_node]==0)
{adj_matrix[src_node][dest_node]=MAX_VALUE;
}       }}
for(int i=1;i<=numb_vert;i++)
{       Bellmanford bellmanford=new Bellmanford(numb_vert);
bellmanford.BellmanfordpEvaluation(i,adj_matrix);
}scanner.close();}}
```

## Result:

```
OUTPUT:
1)
enter the number of vertices
6
enter the adjacency matrix
0    2    999    999    4    999
2    0     3     999    1     5
999  3     0      4    999    2
999 999    4      0    999    3
4    1    999    999    0     1
999  5     2      3     1     0
Routing table for Router1i
destination distance/cost
1                                   0
2                                   2
3                                   5
4                                   7
5                                   3
6                                   4
```

```
Routing table for Router2i
destination distance/cost
1                                   2
2                                   0
3                                   3
4                                   5
5                                   1
6                                   2
Routing table for Router3i
destination distance/cost
1                                   5
2                                   3
3                                   0
4                                   4
5                                   3
6                                   2
```

```
Routing table for Router4i
destination distance/cost
1                                   7
2                                   5
3                                   4
4                                   0
5                                   4
6                                   3
Routing table for Router5i
destination distance/cost
1                                   3
2                                   1
3                                   3
4                                   4
5                                   0
6                                   1
Routing table for Router6i
destination distance/cost
1                                   4
2                                   2
3                                   2
4                                   3
5                                   1
```

**Result:**Shortest Path routing has been implemented