BELLMAN- FORD ALGORITHM

BY GROUP 12

GROUP MEMBERS : K.VISHNU VARDHAN (21CE10031)

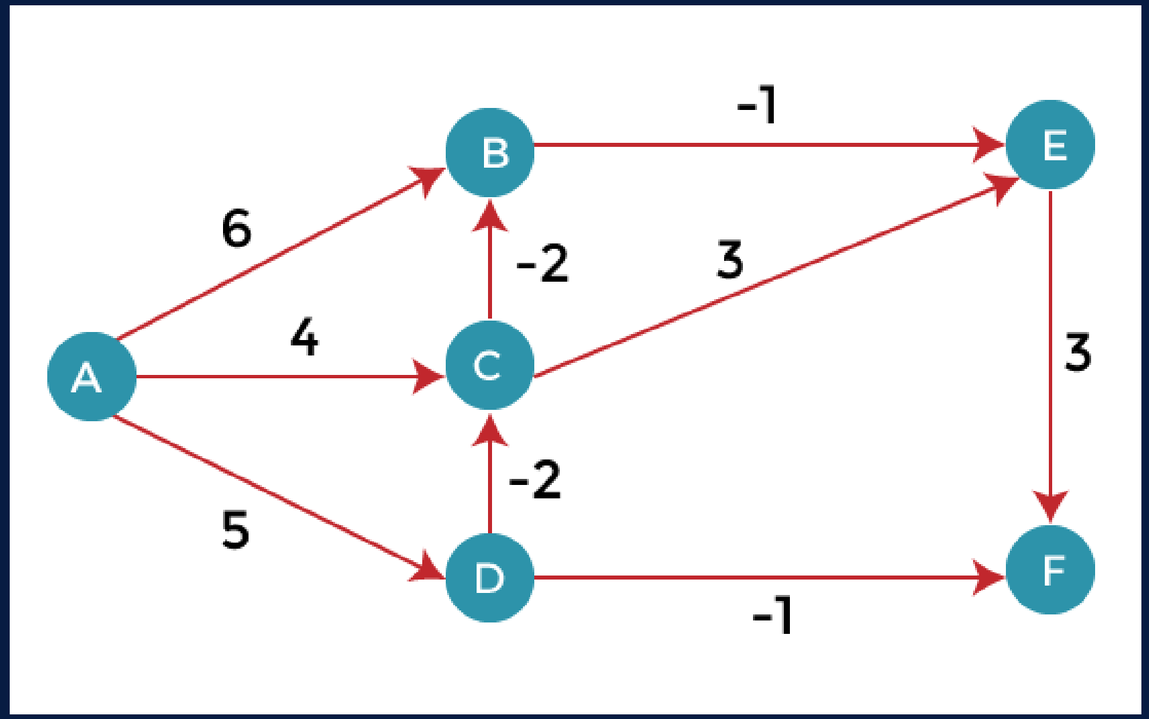
K.SHREYA REDDY (21CE10032)

What is Bellman-Ford Algorithm ?

- The Bellman–Ford algorithm is an algorithm that computes the shortest paths from a single source vertex to all of the other vertices in a weighted digraph.
- The algorithm was first proposed by Alfonso Shimbel (1955) but is instead named after Richard Bellman and Lester Ford Jr., who published it in 1958 and 1956, respectively.
- It is capable of handling graphs in which some of the edge weights are negative numbers.
- If a graph contains a "negative cycle" (i.e. a cycle whose edges sum to a negative value), the Bellman–Ford algorithm can detect and report the negative cycle.

How does it work ?

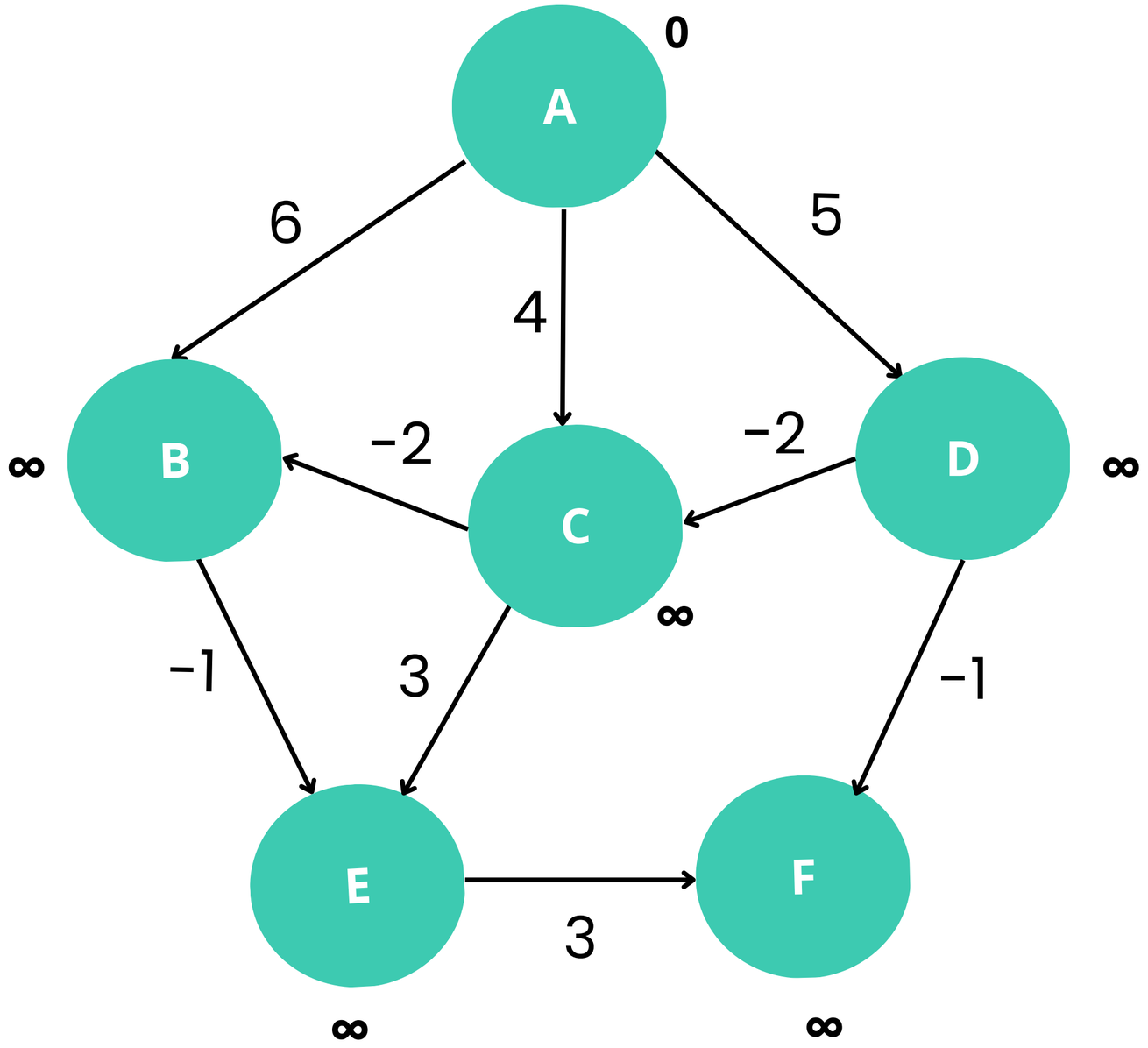
- Bellman-Ford algorithm is a single-source shortest path algorithm that finds the shortest path from a source node to every other node in a weighted graph. It works by iteratively relaxing the shortest path weights until all paths are found.
- The algorithm begins by assigning a distance of 0 to the source node and infinity to all other nodes. It then relaxes the edges of the graph one by one, updating the distance of each node until all of the distances have been updated.
- The algorithm then checks for any negative cycles in the graph, and if one is found, it returns an error. If no negative cycles are found, the shortest path from the source node to all other nodes is found.
- If there are V number of vertices in the graph, the number of iterations would be |V|-1.



STEP 1

- Let the given source vertex(here A) be 0. Initialize all distances as infinite, except the distance to the source itself.
- The total number of vertices in the graph is 6, so all edges must be processed 5 times.

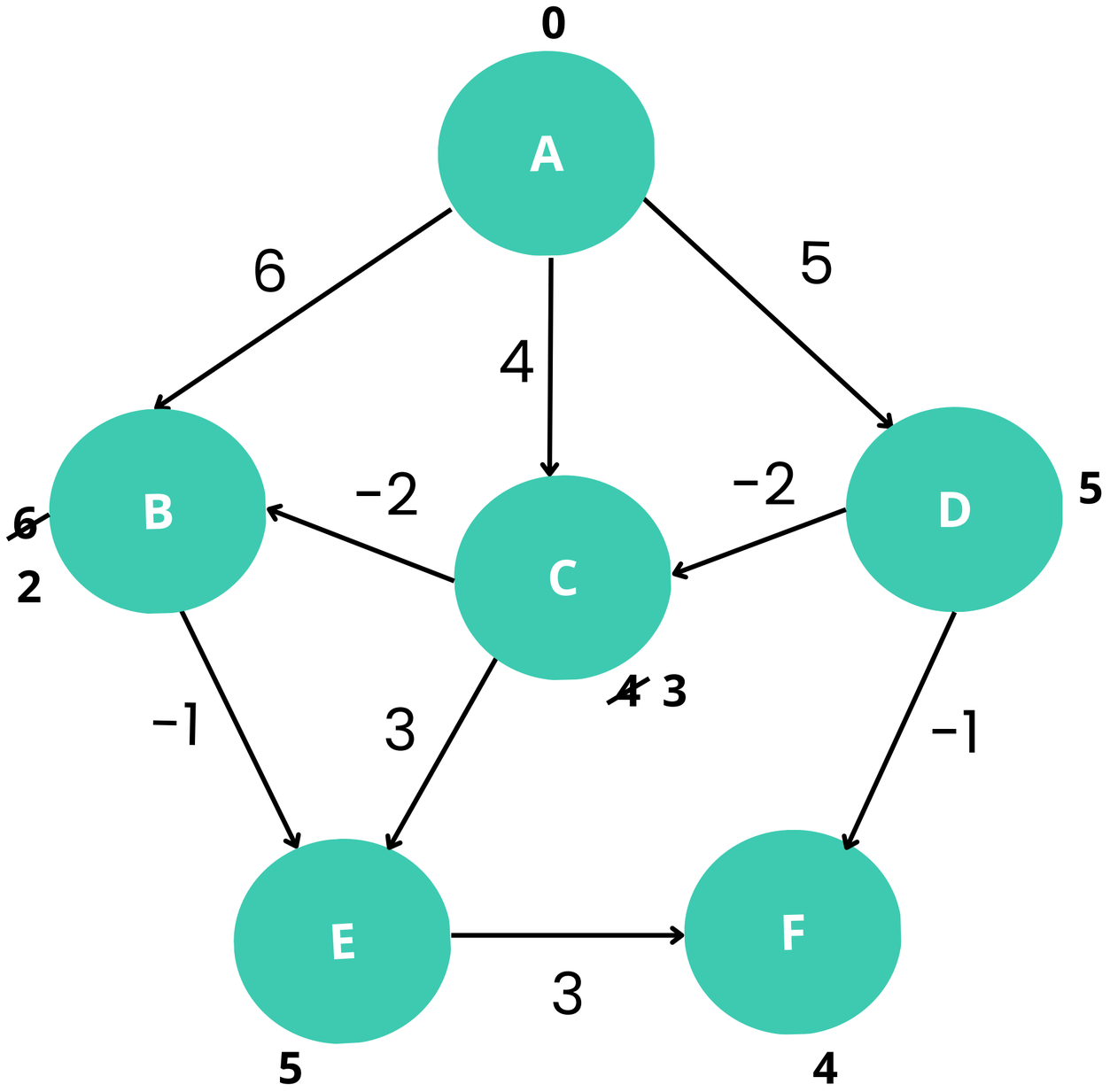
A	B	C	D	E	F
0	∞	∞	∞	∞	∞



STEP 2 : 1st iteration

- The paths possible are (A,B),(A,C), (A,D),(B,E),(C,B),(C,E),(D,C),(D,F),(E,F).
- Now the distances are updated following the rule:if **distance[u]+weight of edge uv < distance[v]** then distance [v] is replaced with distance[u]+weight of edge uv.
- After the first iteration, the values get altered/updated as follows.

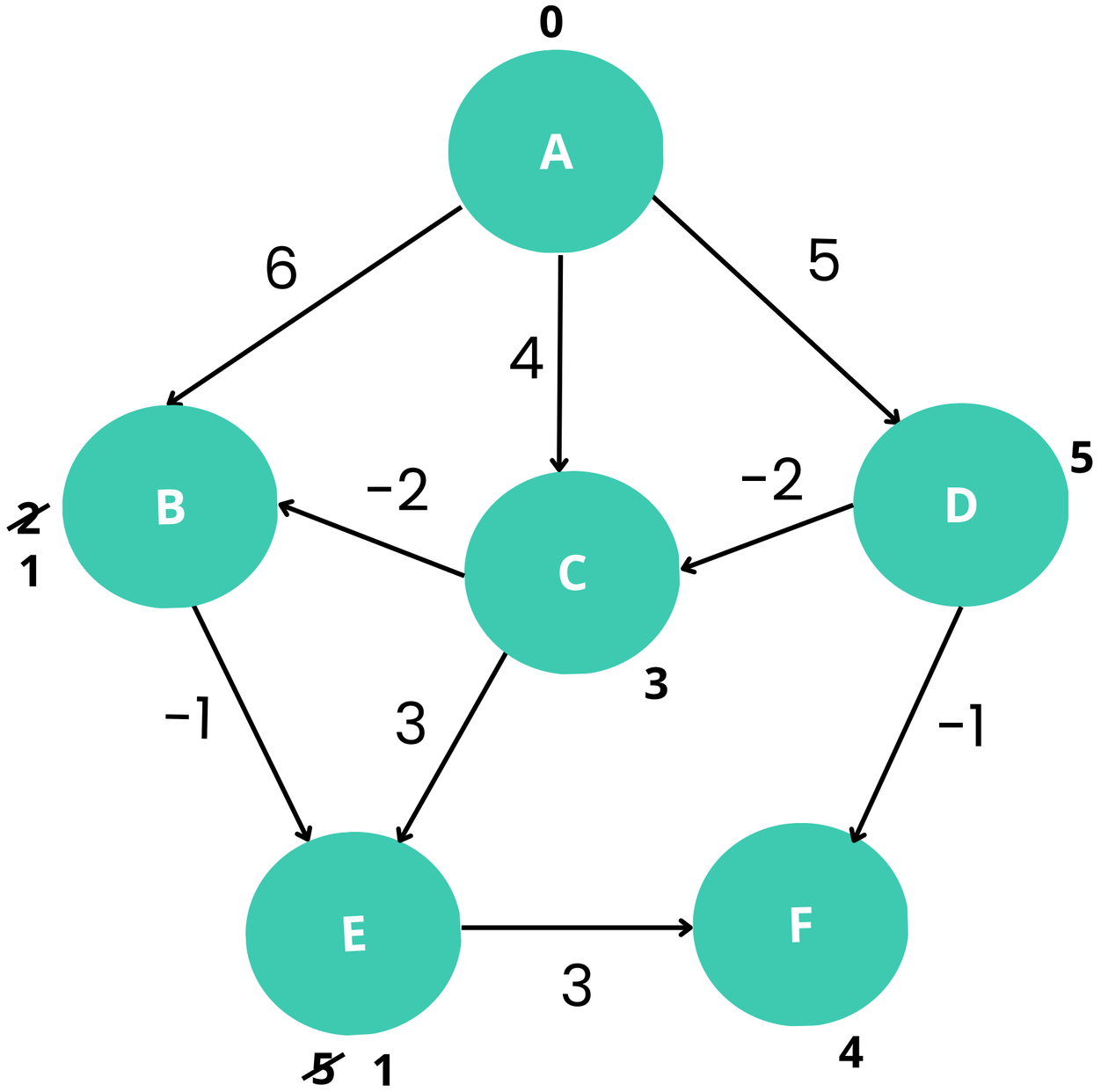
A	B	C	D	E	F
0	∞	∞	∞	∞	∞
0	2	3	5	5	4



STEP 3 : 2nd iteration

- Repeats the same and sees if the values change.

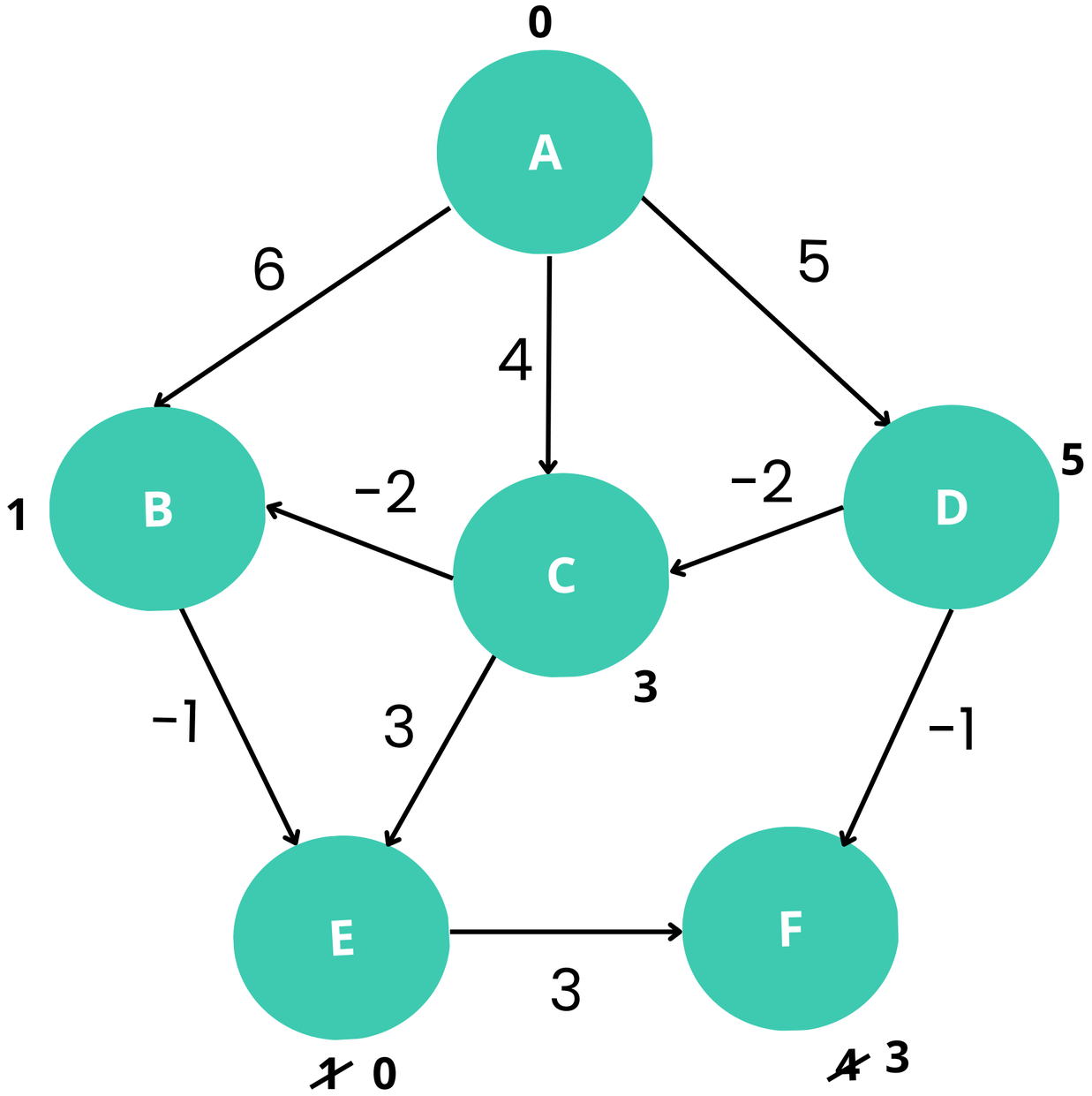
A	B	C	D	E	F
0	∞	∞	∞	∞	∞
0	2	3	5	5	4
0	1	3	5	1	4



STEP 3 : 3rd iteration

- Repeats the same and sees if the values change.

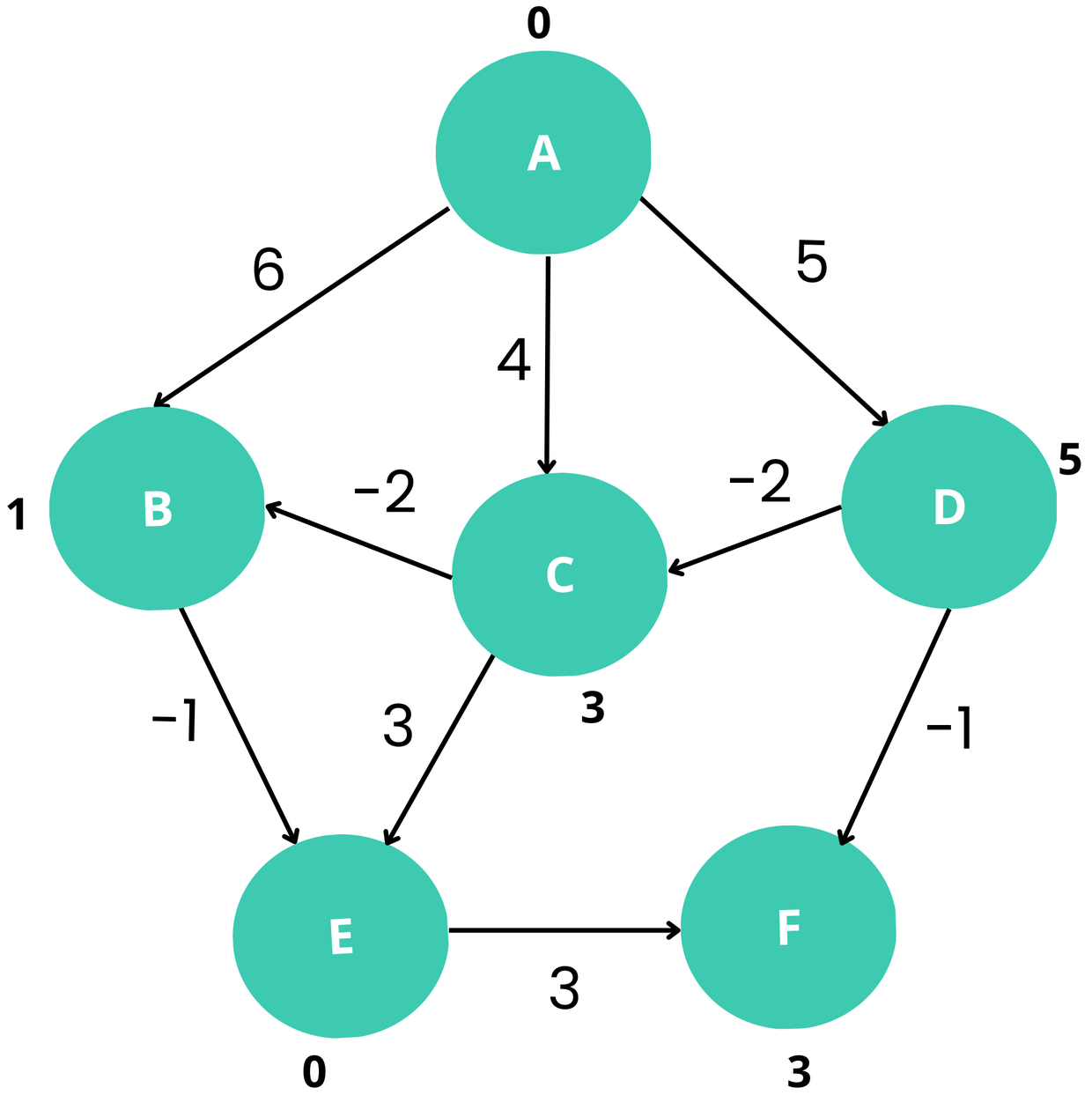
A	B	C	D	E	F
0	∞	∞	∞	∞	∞
0	2	3	5	5	4
0	1	3	5	1	4
0	1	3	5	0	3



STEP 4 : 4th iteration

- Repeats the same and sees if the values change.

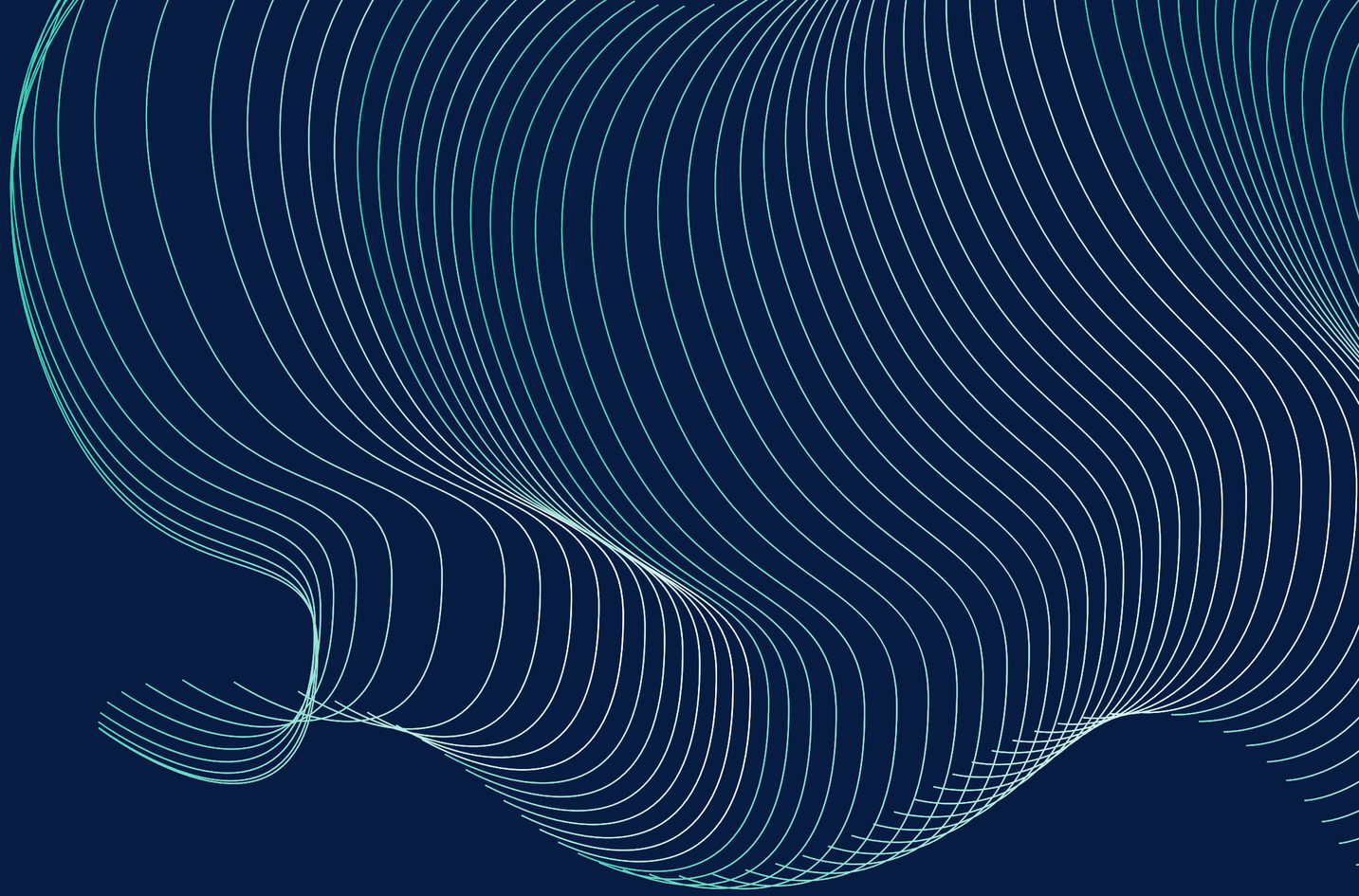
A	B	C	D	E	F
0	∞	∞	∞	∞	∞
0	2	3	5	5	4
0	1	3	5	1	4
0	1	3	5	0	3
0	1	3	5	0	3



- We observe that the distances have been minimized and didn't change after the third iteration.

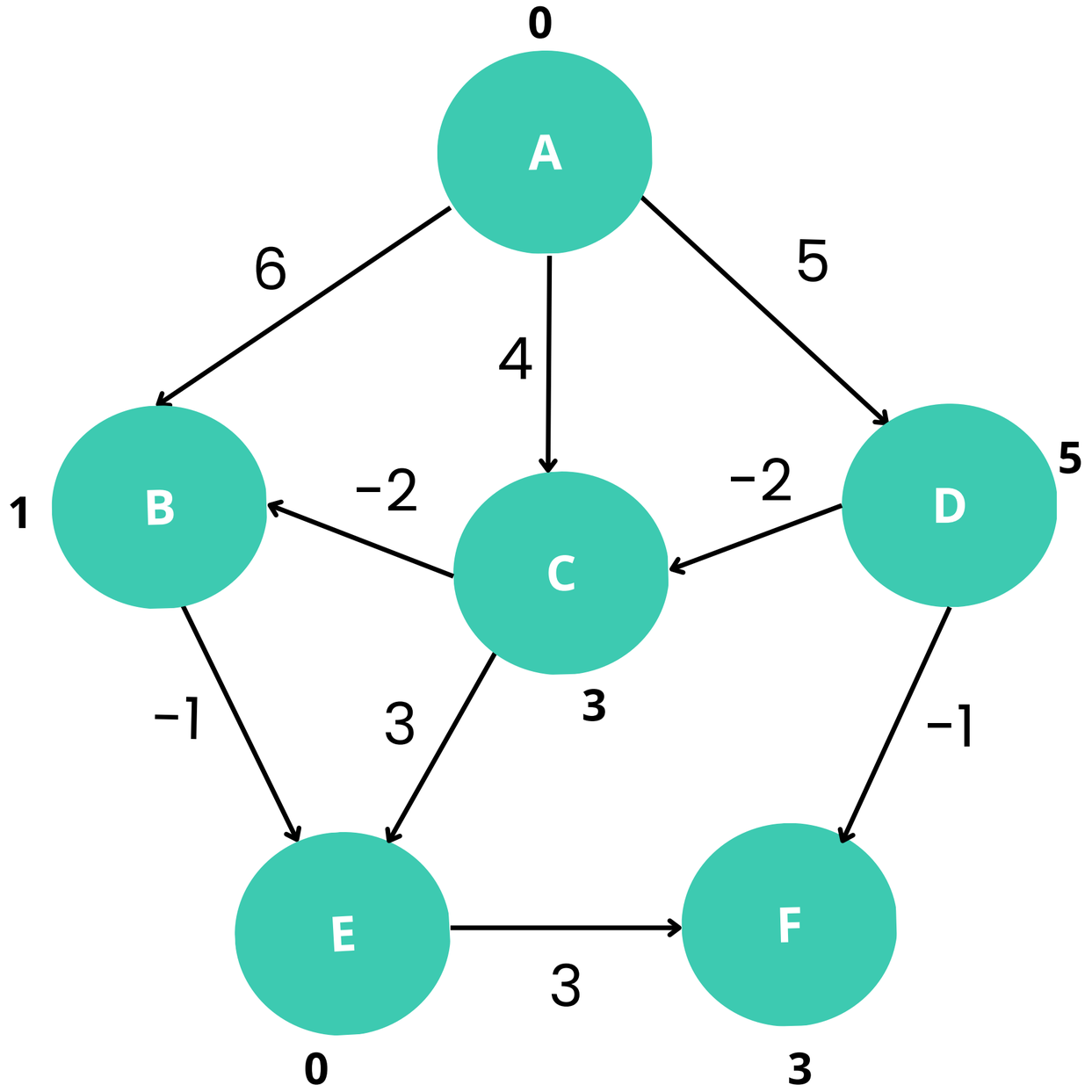
- The algorithm processes it once again but doesn't update the distances.

- **Time complexity of this algorithm: O(E.V),** where E is the number of edges and V is the number of vertices.



After 5th iteration

Vertex	Shortest distance from source
A	0
B	1
C	3
D	5
E	0
F	3



Applications of Bellman-Ford algorithm

The Bellman-Ford algorithm is used in various applications such as network routing protocols, finding the cheapest way to travel from one place to another, and finding the shortest path in a network.

It is also used in distributed systems, such as in distributed databases and distributed computing.

It can also be used in the analysis of financial markets, where it can be used to find the optimal portfolio.

Advantages :

- The Bellman-Ford Algorithm has several advantages. It is relatively easy to implement and can be used to solve the single-source shortest path problem for a graph with negative edge weights i.e. it is more versatile.

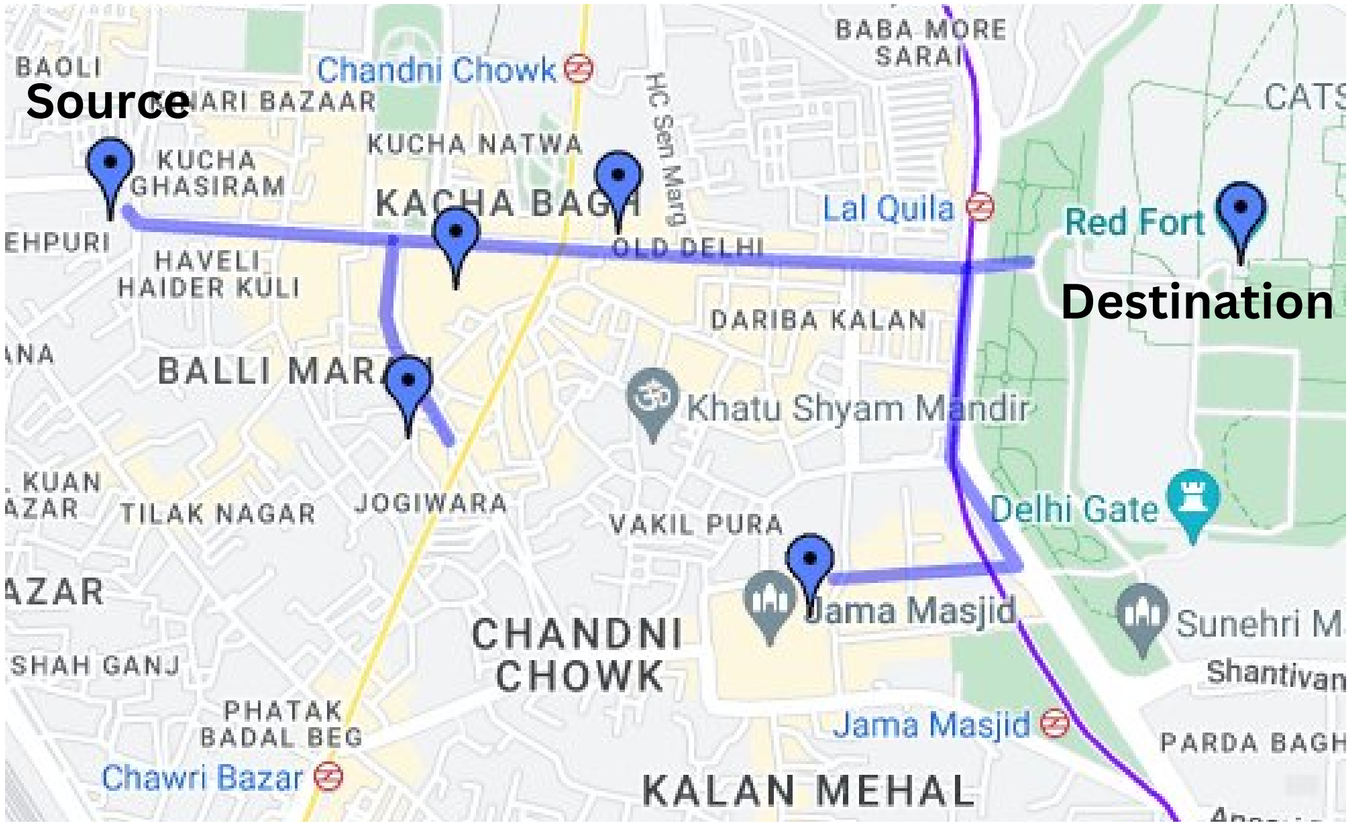
- It also has the ability to detect negative cycles in a graph.

Disadvantages :

- However, the Bellman-Ford Algorithm also has some disadvantages. It is not as efficient as other algorithms such as Dijkstra’s Algorithm, and it requires more time and memory to run. It is also not suitable for large graphs with a large number of edges.

- The Bellman-Ford algorithm is not suitable for graphs with negative weight cycles. Negative weight cycles are cycles of edges whose sum is negative. These cycles can cause the algorithm to loop indefinitely, as it can continue to decrease the distance to a vertex each time it passes through the cycle.

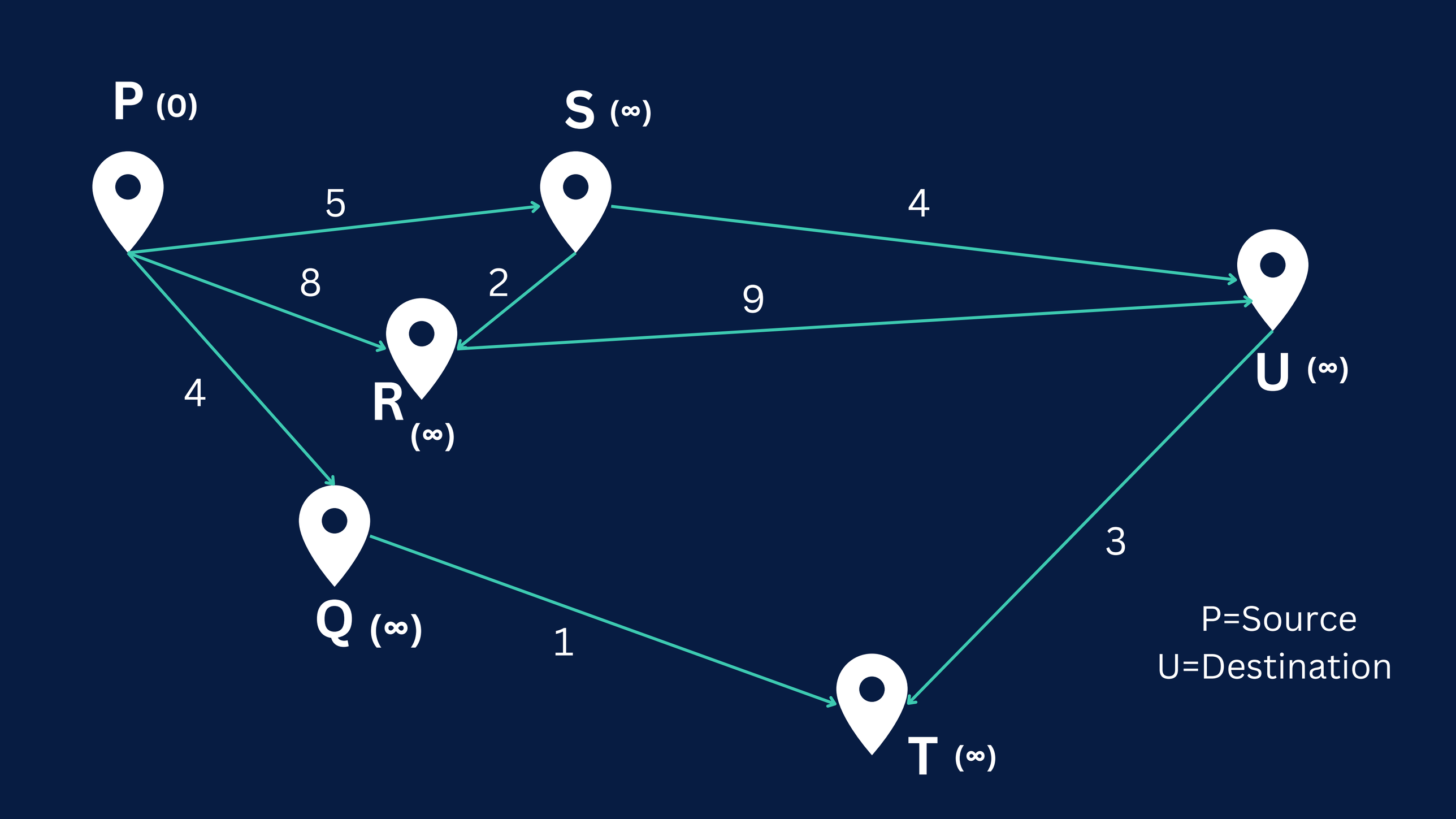
Problem :



This is the map of a locality and we are supposed to find the shortest path to reach our destination from the source in terms of travel time by considering all factors.

Using **Bellman-Ford Algorithm**

- Each pinned location on this map can be considered as a vertice.
- The map is a graph.
- Roads connecting various locations are the edges.

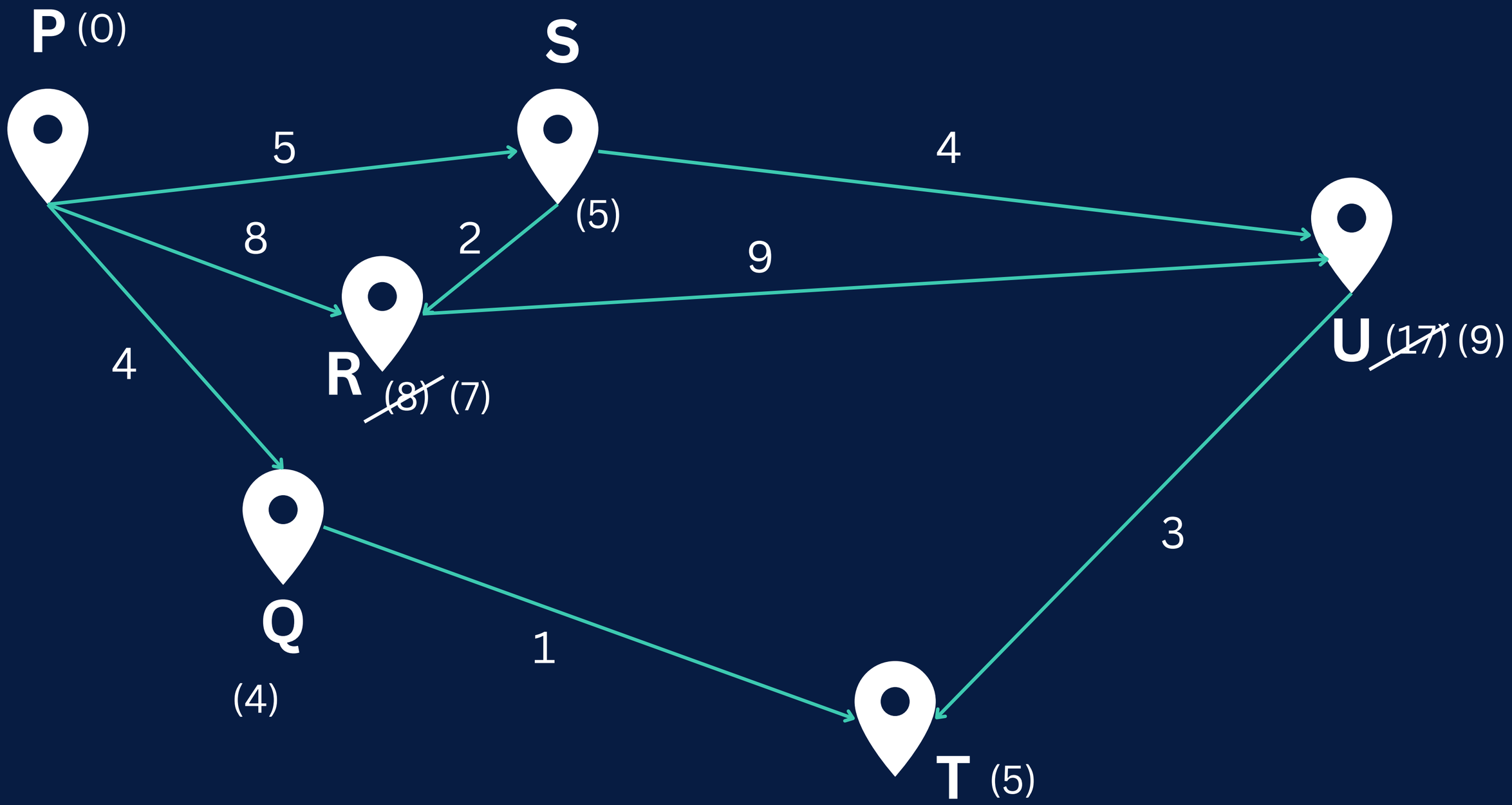

P (0)
S (∞)
5
4
8
2
9
R
(∞)
U (∞)
4
Q (∞)
1
3
T (∞)
P=Source
U=Destination

We start by assigning 0 to the source node and infinity to all other nodes/vertices.

Edges: (P,Q), (P,R), (P,S), (Q,T), (R,U), (S,R), (S,U), (U,T)

There are 6 vertices, so all edges must be processed 5 times.

P	Q	R	S	T	U
0	∞	∞	∞	∞	∞


P (0)
S
5
4
(5)
8
2
9
R
(8) (7)
U (17) (9)
4
Q
(4)
1
3
T (5)

After the first iteration, the distances get modified as follows.

P	Q	R	S	T	U
0	∞	∞	∞	∞	∞
0	4	7	5	5	9

The distances are now minimised and do not change in the further iterations. The algorithm gets processed for 4 more times but the distances don't get updated.

After five iterations :

Vertex	Shortest distance
P	0
Q	4
R	7
S	5
T	5
U	9

BUT ROAD CONDITIONS AREN'T ALWAYS THE SAME !

Traffic conditions and road conditions may not be the same always. Peak hours of all the roads may not be the same and sometimes even if the traffic volume is low uneven roads or roads under construction might lead to delays. We would like to consider this issue and make our code applicable to a wider range of situations by assuming different traffic and road conditions for all the segments like:

- High traffic (h), uneven roads(r): For segment P-Q
- Low traffic(l), even road(s): For segment P-R
- Moderate traffic(m), even road(s): For segment P-S
- Low traffic(l), uneven road(r): For segment P-T
- High traffic(h), even roads(s): From segment P-U

CODE :

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

// structure to represent an edge in the graph
struct Edge
{
   int source, destination, weight;
};

// structure to represent a graph
struct Graph
{
   int vertices, edges;
   struct Edge *edge;
};

// Function to create a graph with V vertices and E edges
struct Graph *createGraph(int vertices, int edges)
{
   struct Graph *graph = (struct Graph *)malloc(sizeof(struct Graph));
   graph->vertices = vertices;
   graph->edges = edges;
   graph->edge = (struct Edge *)malloc(edges * sizeof(struct Edge));
   return graph;
}
```

```
// Function to print the distance array returned by the Bellam-Ford algorithm
void printDistance(int dist[], int n)
{
 printf("Vertex\tShortest distance from the source\n");
 for (int i = 0; i < n; ++i)
 {
 if (i == 0)
 printf("Source\t\t0\n");
 else
 printf("%d\t\t%d\n", i, dist[i]);
 }
}
// Function to consider the dynamically varying parameters
// eg; Traffic congestion, condition of the road
void Traffic_And_Condition(int dist[], int n)
{
 int min_val = INT_MAX, min_vertex = -1;
 for (int i = 1; i < n; ++i)
 {
 char traffic, condition;
 printf("Enter the traffic(l/m/h) and road condition(s/r) from source to vertex %d (l/m/h):\n", i);
 scanf(" %c %c", &traffic, &condition);
 int t, c;
 if (traffic == 'l')
 t = -1;
```

```
 else if (traffic == 'm')
 t = 0;
 else if (traffic == 'h')
 t = 1;
 if (condition == 's')
 c = -1;
 else if (condition == 'r')
 c = 1;

 int new_dist = dist[i] + t + c;
 if (new_dist < min_val)
 {
 min_val = new_dist;
 min_vertex = i;
 }
 }
 printf("The path from the source to vertex %d is the quickest\n", min_vertex);
}
```

```
// Bellman-Ford algorithm to find the shortest path from a source vertex to all other vertices
void BellmanFord(struct Graph *graph, int source)
{
 int V = graph->vertices;
 int E = graph->edges;
 int dist[V];

 // Initialize distance array
 for (int i = 0; i < V; ++i)
 {
 dist[i] = INT_MAX;
 }
 dist[source] = 0;

 // Relax all edges V-1 times
 for (int i = 1; i <= V - 1; ++i)
 {
```

```
for (int j = 0; j < E; ++j)
{
int u = graph->edge[j].source;
int v = graph->edge[j].destination;
int weight = graph->edge[j].weight;
if (dist[u] != INT_MAX && dist[u] + weight < dist[v])
{
dist[v] = dist[u] + weight;
}
}
}

// Check for negative-weight cycles
for (int i = 0; i < E; ++i)
{
int u = graph->edge[i].source;
int v = graph->edge[i].destination;
int weight = graph->edge[i].weight;
if (dist[u] != INT_MAX && dist[u] + weight < dist[v])
```

```
{
 printf("The graph contains a negative-weight cycle\n");
 return;
 }
 }

 // Print distance array
 printDistance(dist, V);
 Traffic_And_Condition(dist, V);
}

int main()
{
 int V, E, s = 0;
 printf("Enter the number of vertices and edges in the graph :\n");
 scanf("%d%d", &V, &E);
```

```
struct Graph *graph = createGraph(V, E);
 for (int i = 0; i < E; ++i)
 {
 int u, v, w;
 printf("Enter the source, destination and weight of edge %d :\n", i + 1);
 scanf("%d%d%d", &u, &v, &w);
 graph->edge[i].source = u;
 graph->edge[i].destination = v;
 graph->edge[i].weight = w;
 }

 BellmanFord(graph, s);

 return 0;
}
```

Enter the number of vertices and edges in the graph :
6 8
Enter the source, destination and weight of edge 1 :
0 1 4
Enter the source, destination and weight of edge 2 :
0 2 8
Enter the source, destination and weight of edge 3 :
0 3 5
Enter the source, destination and weight of edge 4 :
1 4 1
Enter the source, destination and weight of edge 5 :
2 4 9
Enter the source, destination and weight of edge 6 :
3 2 2
Enter the source, destination and weight of edge 7 :
3 5 4
Enter the source, destination and weight of edge 8 :
5 4 3

Vertex	**Shortest distance from the source**
Source	**0**
1	**4**
2	**7**
3	**5**
4	**5**
5	**9**

Enter the traffic(l/m/h) and road condition(s/r) from source to vertex 1 (l/m/h):
h r
Enter the traffic(l/m/h) and road condition(s/r) from source to vertex 2 (l/m/h):
l s
Enter the traffic(l/m/h) and road condition(s/r) from source to vertex 3 (l/m/h):
m s
Enter the traffic(l/m/h) and road condition(s/r) from source to vertex 4 (l/m/h):
l r
Enter the traffic(l/m/h) and road condition(s/r) from source to vertex 5 (l/m/h):
h s

The path from the source to vertex 3 is the quickest

THANK YOU