Koneru Lakshmaiah Education Foundation
(Deemed to be University estd. u/s. 3 of the UGC Act, 1956)
Off-Campus: Bachupally-Gandimaisamma Road, Bowrampet, Hyderabad, Telangana - 500 043.
Phone No: 7815926816, www.klh.edu.in

Department of Computer Science and Engineering

2024-2025

DESIGN AND ANALYSIS ALGORITHMS (23CS2205R)

ALM - CASE STUDY

Case Study on Bellman-Ford Algorithm for Shortest Path in Weighted Graphs

BSR CHAITANYA KOUSIK 2320030142

26/9/24

COURSE INSTRUCTOR

Dr. J SIRISHA DEVI

Bellman-Ford Algorithm

The Bellman-Ford algorithm is used to compute the shortest paths from a single source to all other vertices in a weighted graph, and it is capable of handling graphs with negative-weight edges. Given a weighted graph G=(V,E) with V vertices and E edges, the goal is to find the shortest path from a source vertex to all other vertices. The algorithm also detects negative-weight cycles in the graph.

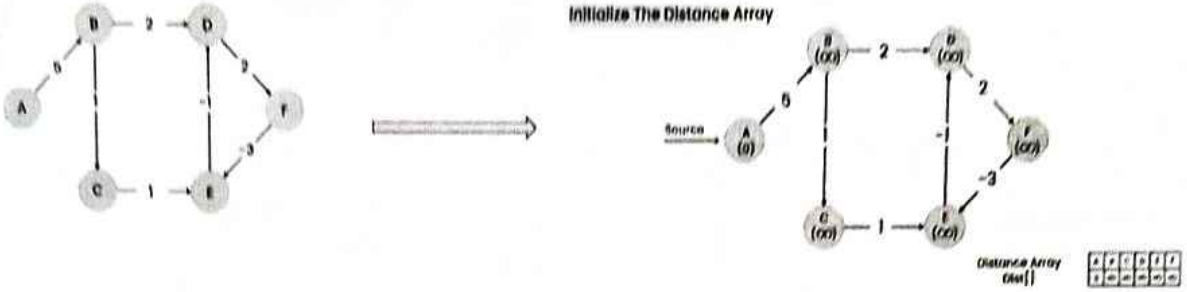
Application/Example:

To provide a practical case study for the Bellman-Ford algorithm, consider its application in a public transportation network. This network is represented as a weighted graph where vertices symbolize bus stops and the edges denote routes between these stops with weights as travel times. Some routes may have negative weights due to subsidies or special discounts during off-peak hours. The objective is to determine the shortest travel time from a central bus station to all other stations.

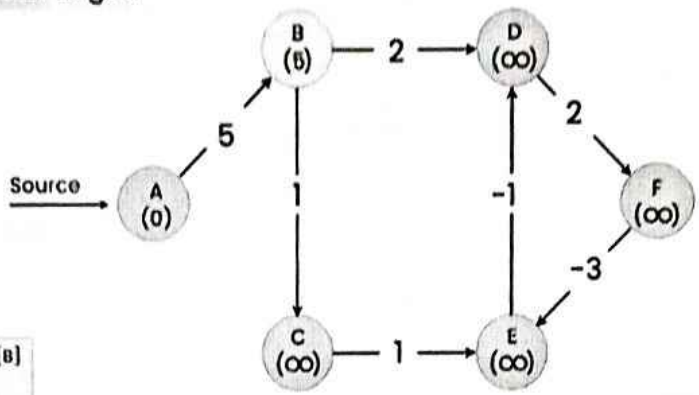
In this scenario, the Bellman-Ford algorithm is critical for two reasons. Firstly, it provides the shortest travel times even in the presence of negative weights, unlike other shortest path algorithms such as Dijkstra's, which only work with non-negative weights. Secondly, the algorithm's ability to detect negative-weight cycles is invaluable because such cycles could represent a scheduling or routing anomaly in the transportation system, like a route that continually decreases total travel time, indicating a potential flaw in route planning.

By applying the Bellman-Ford algorithm to the transportation network, transit planners can optimize travel schedules, ensure fair and efficient route management, and detect and rectify any discrepancies in the routing system that could lead to impractical or inefficient travel times. This application showcases the algorithm's practical utility in managing complex systems with varied route conditions.

ALGORITHM / PSEUDO CODE



1st Relaxation Of Edges



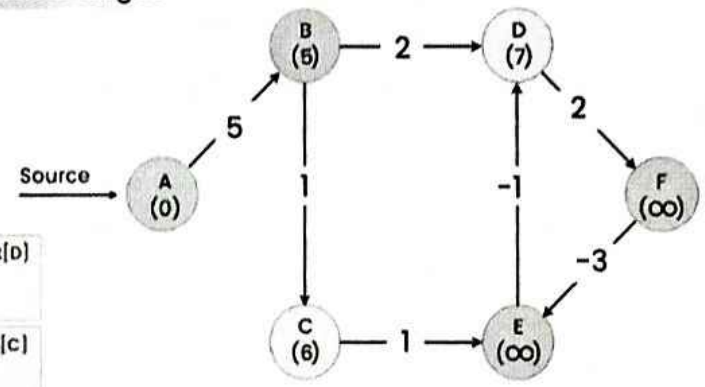
Dist [A] + 5 <Dist[B]
0+5<(∞)
Dist[B] = 5

Distance Array

A	B	C	D	E	F
0	∞	∞	∞	∞	∞

A	B	C	D	E	F
0	5	∞	∞	∞	∞

2nd Relaxation Of Edges



Dist [B] + 2 <Dist[D]
5+2<(∞)
Dist[D] = 7

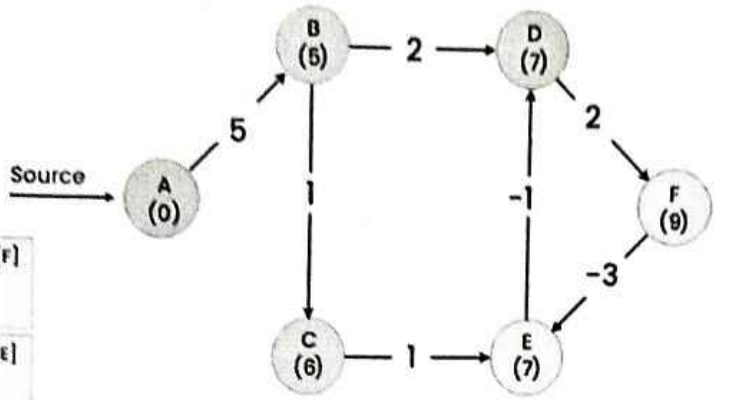
Dist [B] + 1 <Dist[C]
5+1 <(∞)
Dist[C] = 6

Distance Array

A	B	C	D	E	F
0	5	∞	∞	∞	∞

A	B	C	D	E	F
0	5	6	7	∞	∞

3rd Relaxation Of Edges



Dist [D] + 2 <Dist[F]
7+2<(∞)
Dist[F] = 9

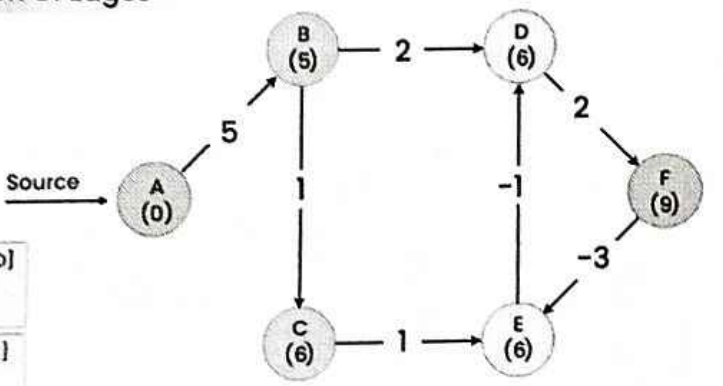
Dist [C] + 1 <Dist[E]
6+1 <(∞)
Dist[E] = 7

Distance Array

A	B	C	D	E	F
0	5	6	7	∞	∞

A	B	C	D	E	F
0	5	6	7	7	9

4th Relaxation Of Edges



Dist [E] + 2 <Dist[D]
7+(-1)< 7
Dist[D] = 6

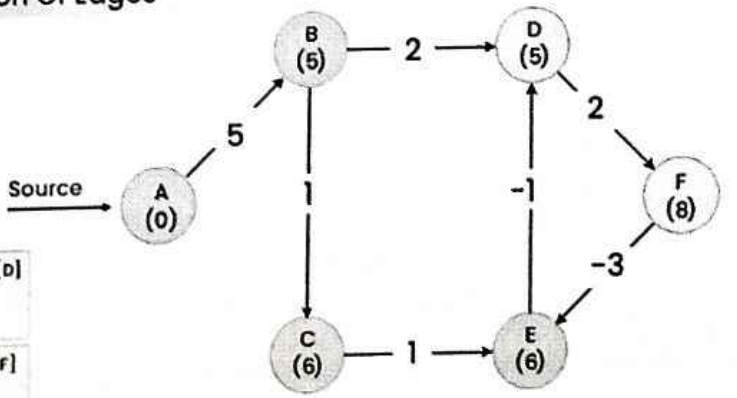
Dist [F] + 1 <Dist[E]
9+(-3) < 6
Dist[E] = 6

Distance Array

A	B	C	D	E	F
0	5	6	7	7	9

A	B	C	D	E	F
0	5	6	6	6	9

5th Relaxation Of Edges



Dist [E] + (-1) <Dist[D]
6+(-1)<6
Dist[D] = 5

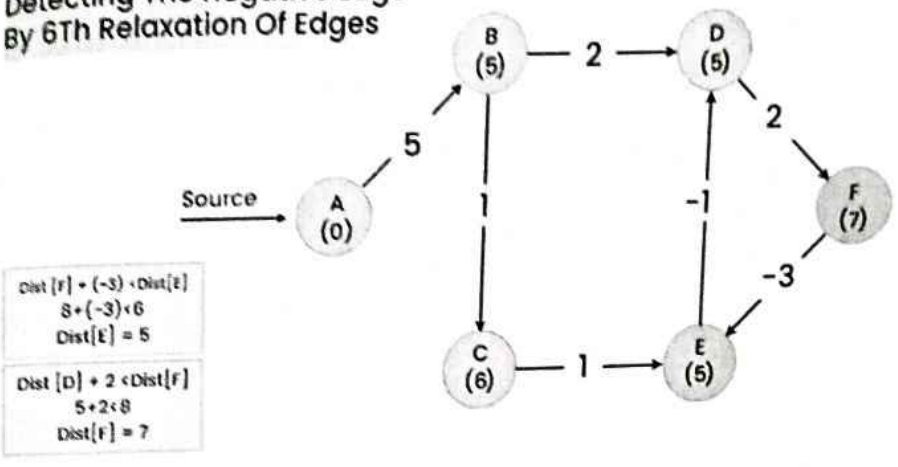
Dist [D] + 2 <Dist[F]
6+2<9
Dist[F] = 8

Distance Array

A	B	C	D	E	F
0	5	6	6	6	9

A	B	C	D	E	F
0	5	6	5	6	8

Detecting The Negative Edge By 6Th Relaxation Of Edges




```
Algorithm Bellman-Ford(Graph, source)
    Input: Graph G = (V, E) where V is the set of vertices and E is the set of edges, and
    Output: Shortest path from source to all vertices or detection of a negative-weight cy

    Step 1: Initialize distance to all vertices
        for each vertex v in V
            if v is source
                distance[v] = 0
            else
                distance[v] = infinity

    Step 2: Relax edges repeatedly
        for i from 1 to |V|-1
            for each edge (u, v) in E
                if distance[u] + weight(u, v) < distance[v]
                    distance[v] = distance[u] + weight(u, v)

    Step 3: Check for negative-weight cycles
        for each edge (u, v) in E
            if distance[u] + weight(u, v) < distance[v]
                report "Graph contains a negative-weight cycle"

    Step 4: Return the distances
        return distance
```

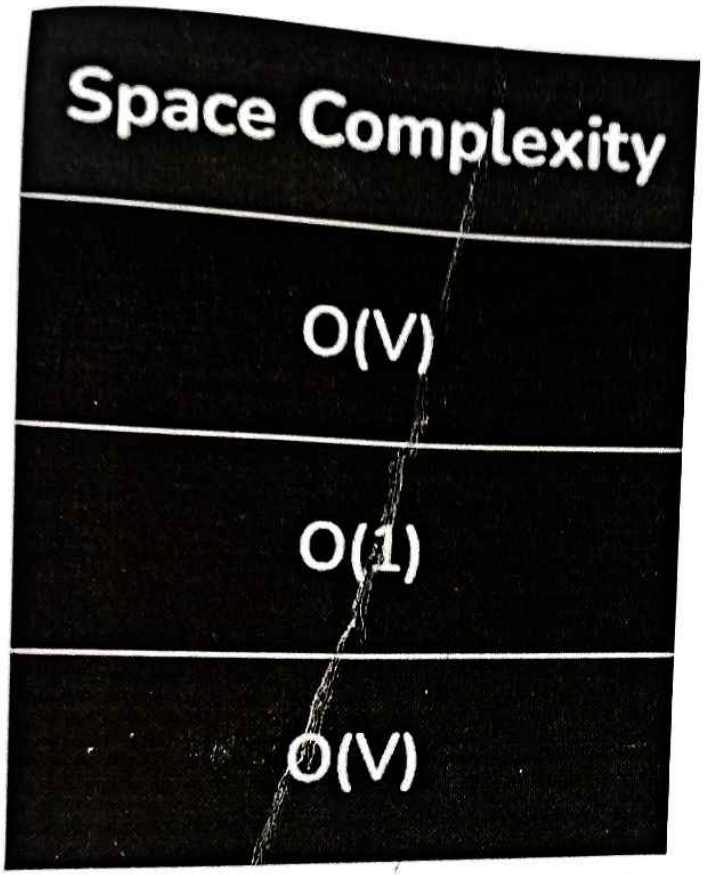
TIME COMPLEXITY

Operation	Time Complexity
Initialization	O(V)
Relaxation	O(V*E)
Overall Complexity	O(V*E)

The time complexity of the Bellman-Ford algorithm is primarily determined by its two nested loops:

1. The outer loop runs |V| - 1 times, where |V| is the number of vertices in the graph. This loop accounts for the fact that in the worst case, the shortest path through the graph could traverse up to |V| - 1 edges (the maximum number of edges in any simple path, preventing cycles).
2. The inner loop iterates over all edges, |E|, in the graph. For each iteration of the outer loop, every edge is examined to check if a shorter path can be established through that edge.

SPACE COMPLEXITY



The space complexity of the Bellman-Ford algorithm is O(V), where V is the number of vertices in the graph. This is because the algorithm primarily requires space to store the distance values for each vertex from the source vertex. Additional minimal space may be used for auxiliary data structures such as arrays or lists to manage the edges and vertices during the computation, but the primary storage requirement remains linear in terms of the number of vertices.