BELLMAN FORD ALGORITHM

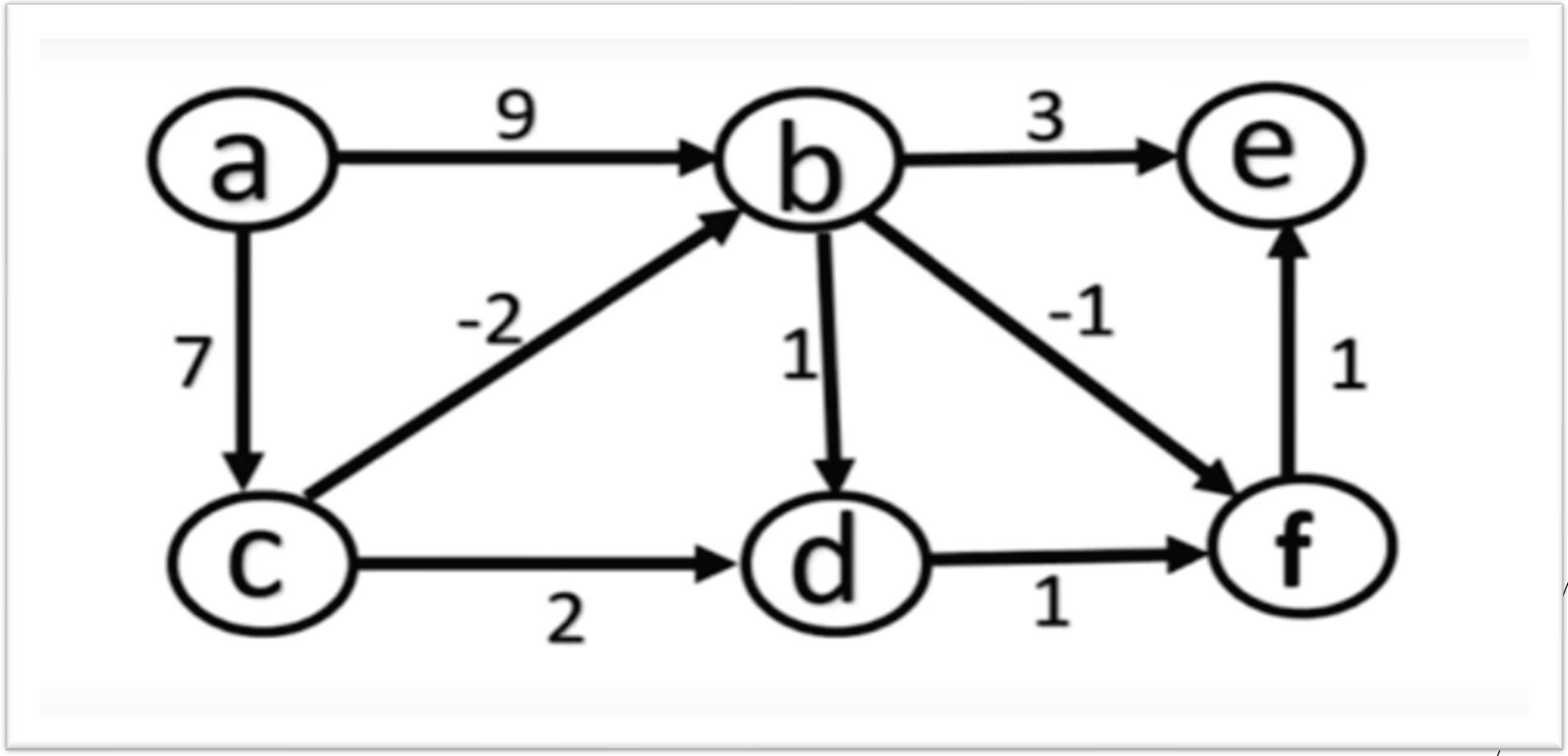
THE GOAL OF THE BELLMAN FORD

To find the shortest paths from a single source node to all other nodes in a weighted graph.

Key Purposes:

1. Find shortest path in a graph even if the graph contains negative weight edges.
2. Detect negative weight cycles if a negative weight cycle is reachable from the source, the algorithm can detect it.
3. The relaxation process is repeated V−1 times to ensure that the shortest paths are correctly calculated.
4. Why V – 1 times? Because every node has at most V – 1 edges.

LET'S SOLVE THIS EXAMPLE TOGETHER



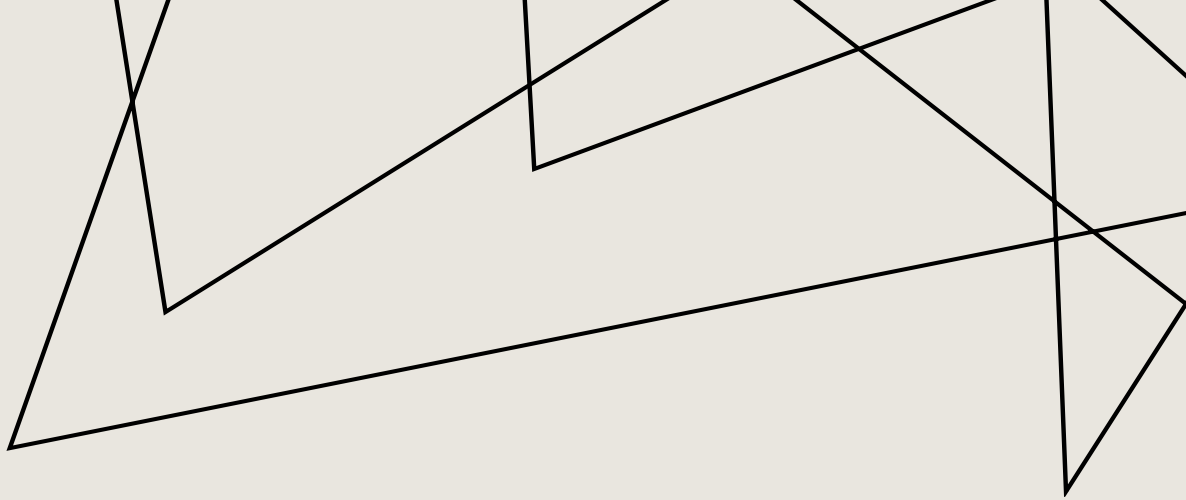
ALGORITHM

```
function BellmanFord(Graph, source):
    // Step 1: Initialize distances
    for each vertex v in Graph:
        distance[v] := ∞
        predecessor[v] := null
    distance[source] := 0
    // Step 2: Relax edges repeatedly
    for i from 1 to |V| - 1:
        for each edge (u, v) with weight w in Graph:
            if distance[u] + w < distance[v]:
                distance[v] := distance[u] + w
                predecessor[v] := u
    // Step 3: Check for negative-weight cycles
    for each edge (u, v) with weight w in Graph:
        if distance[u] + w < distance[v]:
            error "Graph contains a negative-weight cycle"
    return distance[], predecessor[]
```

PYTHON CODE

```
def bellman_ford(graph, vertices, source):
    # Step 1: Initialize distances
    distance = {v: float('inf') for v in vertices}
    predecessor = {v: None for v in vertices}
    distance[source] = 0
    # Step 2: Relax edges |V|-1 times
    for _ in range(len(vertices) - 1):
        for u, v, w in graph:  # graph is a list of edges (u, v, w)
            if distance[u] + w < distance[v]:
                distance[v] = distance[u] + w
                predecessor[v] = u
    # Step 3: Check for negative-weight cycles
    for u, v, w in graph:
        if distance[u] + w < distance[v]:
            raise Exception("Graph contains a negative-weight
cycle")
    return distance, predecessor
```

INPUT, OUTPUT & TIME COMPLEXITY



Input: weighted directed graph, source node, vertices.

output: shortest possible path from the source node to each vertex.

Worst case: O(E.V)

THANK YOU

GHENA GHUSEIN
TAMER OMAR