Bellman-Ford Algorithm

What is BFA:

An algorithm that gives the shortest path from the source node to every other node in the graph.

Advantages of Bellman-Ford:

- Can be used in distributed systems or where the graph is not fully connected.
- Enables efficient and reliable pathfinding.
- Detects negative weight cycles.

Disadvantages of Bellman-Ford:

- Does not work with an *undirected graph* or undirected graph with *negative edges*
- Does not scale well: Slow performance on large graphs with many edges
- Non-optimal performance on non-negative graphs with non-negative edge weights

Note: Usefulness depends on the specific requirements of the application

Bellman-Ford Pseudocode:

```
function BellmanFord(vertices, edges, source):
   // Step 1: Initialize distances from source to all other vertices as infinity
   dist = {}
   for each vertex in vertices:
      dist[vertex] = infinity
   dist[source] = 0

   // Step 2: Relax edges repeatedly
   for i from 1 to |vertices|-1:
      for each edge (u, v, w) in edges:
         if dist[u] + w < dist[v]:
            dist[v] = dist[u] + w

   // Step 3: Check for negative-weight cycles
   for each edge (u, v, w) in edges:
      if dist[u] + w < dist[v]:
         throw "Graph contains a negative-weight cycle"

   return dist
```

Time & Space complexity

Time Complexity:

- Worst case: **O (V^3)**
  - V = total number of vertices
  - E = total number of edges = V * V - 1 = V^2 // V choices for start vertex, V-1 choices for end vertex
  - Therefore, (O(|V| * |E|) = O(V * V^2) = O(V^3)
- Average case: **O(|V| * |E|)**
  - V = total number of vertices
  - E = total number of edges
  - Varies depending on the graph
  - Graph can be dense and have many edges, algorithm will need to relax more edges
  - Graph can be sparse and have few edges, algorithm does not need to relax as many edges
- Best case: **O(E)**
  - Outer loop would only need to run once // relax all edges on the graph 1 time only
  - Occurs when the vertices are connected to each other in linear fashion // only 1 edge going out of each vertex

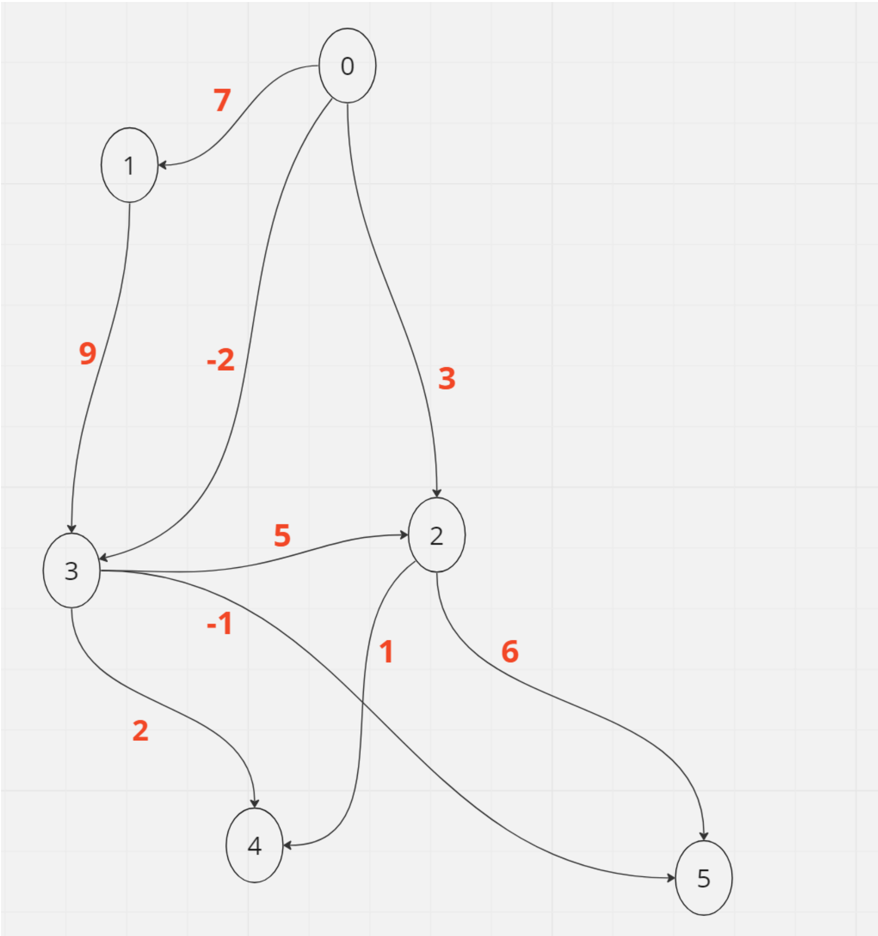
Space Complexity: **O(V + E) for all cases**

- The algorithm requires an array to store the distance from the starting vertex to each of the other vertices in the graph, which has a size of V.
- It requires an array to store the predecessor of each vertex along the shortest path, which also has a size of V.
- It also requires to store all the edges and the weights of those edges

Key Functionality Take-Away

- Performs shortest path traversal on a source node from a weighted digraph.
- Advantageous as it calculates the shortest path negative weights exist.
- Algorithm is able to detect negative weight cycles, indicating that there is no shortest path if there exists one.
- Real life practices include but are not limited to network routing, flight path optimizations, and traffic routing.

Practice:



Directions: Find the shortest path off to all nodes from from the source using the graph above and fill this table

Edges:	Nodes:	Min Distance:
	0	0
	1	
	2	
	3	
	4	
	5	
	6	