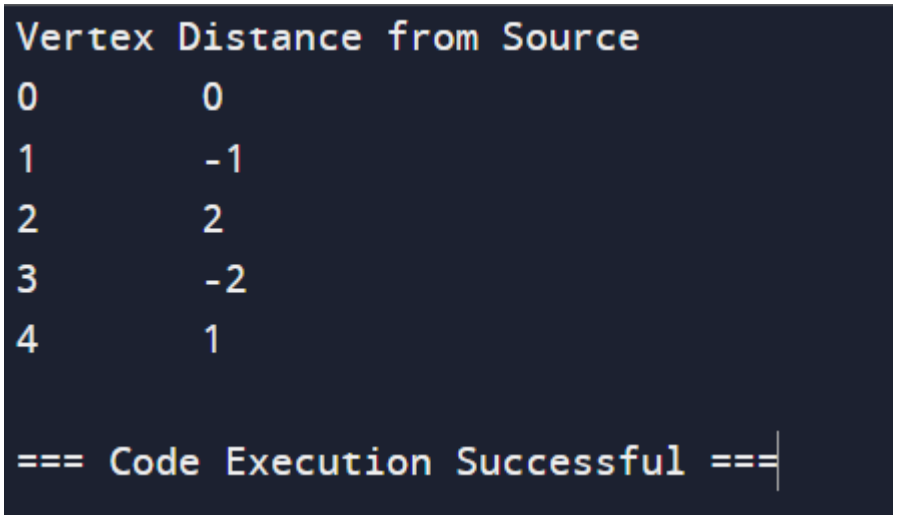
109. Bellman Ford algorithm

PROGRAM:-

```
class Graph:
    def __init__(self, vertices):
        self.vertices = vertices
        self.edges = []

    def add_edge(self, u, v, weight):
        self.edges.append((u, v, weight))

    def bellman_ford(self, source):
        # Step 1: Initialize distances from source to all other vertices as INFINITE
        distance = [float('inf')] * self.vertices
        distance[source] = 0

        # Step 2: Relax all edges |V| - 1 times
        for _ in range(self.vertices - 1):
            for u, v, weight in self.edges:
                if distance[u] != float('inf') and distance[u] + weight < distance[v]:
                    distance[v] = distance[u] + weight

        # Step 3: Check for negative-weight cycles.
        for u, v, weight in self.edges:
            if distance[u] != float('inf') and distance[u] + weight < distance[v]:
                print("Graph contains negative weight cycle")
                return None

        # Print all distances
        self.print_solution(distance)

    def print_solution(self, distance):
        print("Vertex Distance from Source")
        for i in range(self.vertices):
            print(f"{i}\t\t{distance[i]}")

# Example usage
if __name__ == "__main__":
    g = Graph(5)
    g.add_edge(0, 1, -1)
    g.add_edge(0, 2, 4)
    g.add_edge(1, 2, 3)
    g.add_edge(1, 3, 2)
    g.add_edge(1, 4, 2)
    g.add_edge(3, 2, 5)
    g.add_edge(3, 1, 1)
    g.add_edge(4, 3, -3)

    g.bellman_ford(0)
```

OUTPUT:-



TIME COMPLEXITY:-O(v*e)