Bellman Ford Algorithm

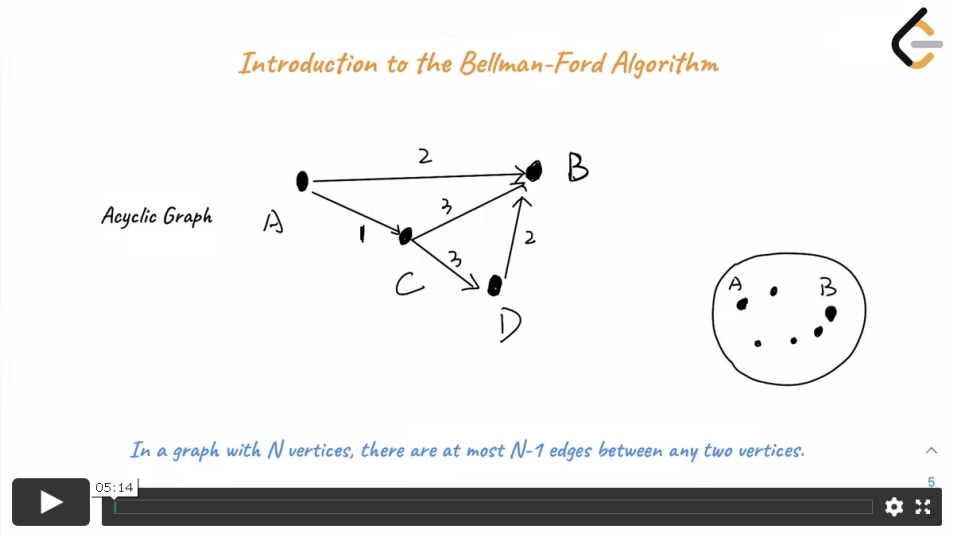
Report Issue

As discussed previously, the "Dijkstra algorithm" is restricted to solving the "single source shortest path" problem in graphs without negative weights. So, how could we solve the "single source shortest path" problem in graphs with negative weights? In this chapter, we will introduce the Bellman-Ford algorithm.

Basic Theorem

Theorem 1: In a "graph with no negative-weight cycles" with N vertices, the shortest path between any two vertices has at most N-1 edges.

In the following video, we're going to prove this Theorem.



▶ Clarifying Notes

Theorem 2: In a "graph with negative weight cycles", there is no shortest path.

In the next video, we'll show why there is no solution can be found when we have a "graph with negative weight cycles".

Using Dynamic Programming to Find the Shortest Path

Now we'll use Dynamic Programming technique to find the shortest path between the vertices in the given graph.

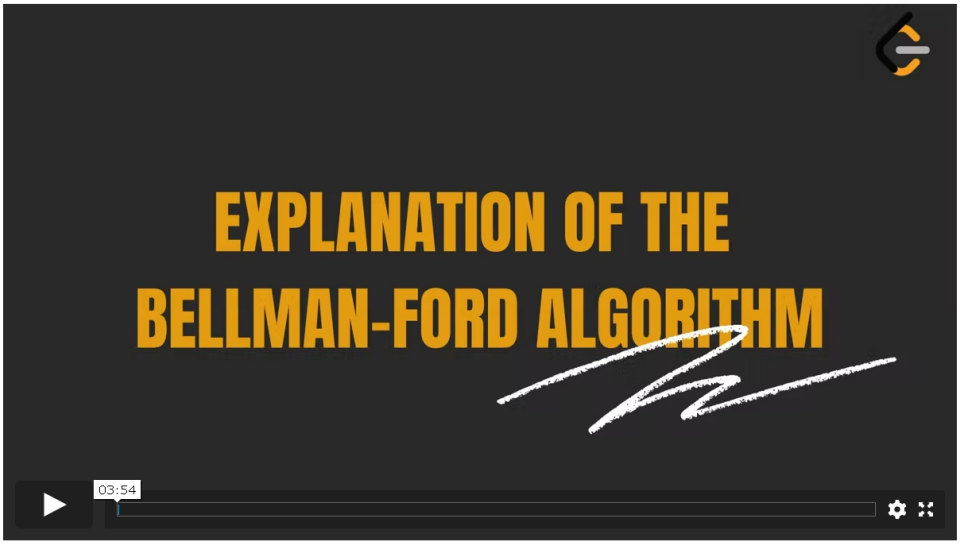
Complexity Analysis

V represents the number of vertices in the graph, and E represents the number of edges.

- Time Complexity: $O(V \cdot E)$. In the worst-case scenario, when all the vertices are connected with each other, we need to check every path from every vertex; this results in $O(V \cdot E)$ time complexity.
- Space Complexity: $O(V^2)$. We need to store a 2-dimensional DP matrix with the size of $V \cdot V$.

Explanation of the Bellman-Ford Algorithm:

We will solve the same problem in the following video, but now we'll use the Bellman-Ford Algorithm.

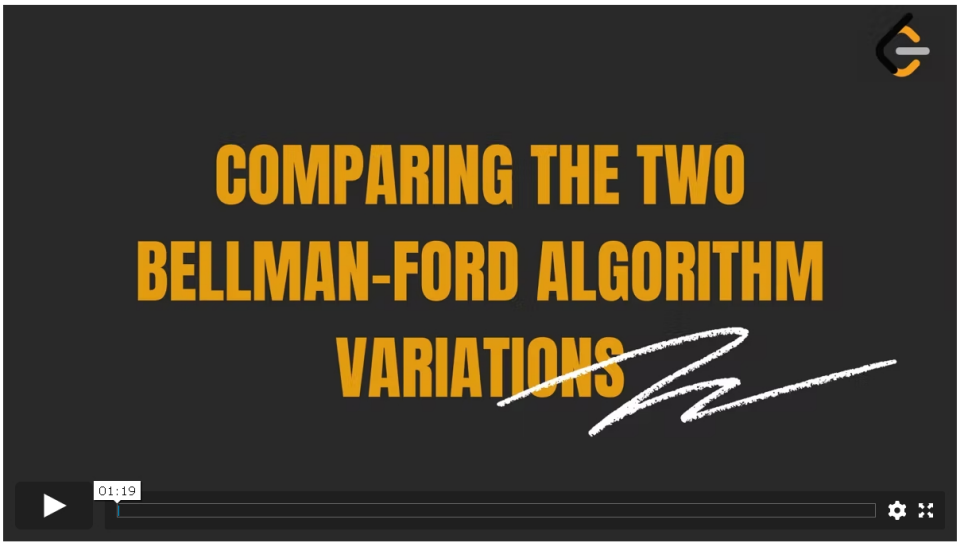


Optimizing the Bellman-Ford Algorithm

In the following video, we'll further optimize our algorithm.

Comparing the Two Bellman-Ford Algorithm Variations

Now let's check the differences between the two versions of the Bellman-Ford Algorithm that we explained earlier.



Limitation of the algorithm

"Bellman-Ford algorithm" is only applicable to "graphs" with no "negative weight cycles".

How does the Bellman-Ford algorithm detect "negative weight cycles"?

Although the "Bellman-Ford algorithm" cannot find the shortest path in a graph with "negative weight cycles", it can detect whether there exists a "negative weight cycle" in the "graph".

Detection method: After relaxing each edge `N-1` times, perform the `N`th relaxation. According to the "Bellman-Ford algorithm", all distances must be the shortest after relaxing each edge `N-1` times. However, after the `N`th relaxation, if there exists `distances[u] + weight(u, v) < distances(v)` for any `edge(u, v)`, it means there is a shorter path . At this point, we can conclude that there exists a "negative weight cycle".

Complexity Analysis

V represents the number of vertices, and E represents the number of edges.

- Time Complexity: we iterate through all the vertices, and in each iteration, we'll perform a relaxation operation for each appropriate edge. Therefore, the time complexity would be $O(V \cdot E)$.

- Space Complexity: $O(V)$. We use two arrays of length V. One to store the shortest distance from the source vertex using at most `k-1` edges. The other is to store the shortest distance from the source vertex using at most `k` edges.