Dijkstra’s Algorithm is a pathfinding algorithm that generates every single route through the graph, and then selects the route that has the lowest overall cost.

This works by iteratively calculating a distance for each node in the graph, starting from the start node, and continuing until we reach the end node. In every iteration we have a “current node,” and we compute a new best score for every node that can be reached from it.

A diagram of a program

Description automatically generated

Time complexity – as the algorithm each and every vertex the time complexity is O(V2)

**Bellman Ford Algo -**

The Bellman-Ford algorithm is a [single-source shortest path](https://en.wikipedia.org/wiki/Shortest_path_problem) algorithm. This means that, given a weighted graph, this algorithm will output the shortest distance from a selected node to all other nodes.

It is very similar to the [Dijkstra Algorithm](https://www.baeldung.com/java-dijkstra). However, unlike the Dijkstra Algorithm, **the Bellman-Ford algorithm can work on graphs with negative-weighted edges**. This capability makes the Bellman-Ford algorithm a popular choice.

1. The first step is to initialize the vertices. The algorithm initially set the distance from starting vertex to all other vertices to infinity. The distance between starting vertex to itself is 0. The variable *D[]* denotes the distances in this algorithm.
2. After the initialization step, the algorithm started calculating the shortest distance from the starting vertex to all other vertices. This step runs  times. Within this step, the algorithm tries to explore different paths to reach other vertices and calculates the distances. If the algorithm finds any distance of a vertex that is smaller then the previously stored value then it relaxes the edge and stores the new value.
3. Finally, when the algorithm iterates  times and relaxes all the required edges, the algorithm gives a last check to find out if there is any negative cycle in the graph.

If there exists a negative cycle then the distances will keep decreasing. In such a case, the algorithm terminates and gives an output that the graph contains a negative cycle hence the algorithm can’t compute the shortest path. If there is no negative cycle found, the algorithm returns the shortest distances.

The Bellman-Ford algorithm is an example of [Dynamic Programming](https://www.baeldung.com/cs/greedy-approach-vs-dynamic-programming). It starts with a starting vertex and calculates the distances of other vertices which can be reached by one edge. It then continues to find a path with two edges and so on. The Bellman-Ford algorithm follows the [bottom-up approach](https://www.investopedia.com/articles/investing/030116/topdown-vs-bottomup.asp).

Time complexity – as the algorithm each and every vertex and edge the time complexity is O(VE)

Examples –

**Dijkstra's Algorithm Example**: GPS Navigation System

GPS navigation system used in vehicles. The system needs to find the shortest path from the current location (source) to the destination while considering various factors such as road distance, traffic conditions, and travel time.

Dijkstra's algorithm can efficiently find the shortest path from the current location to the destination by treating road segments as edges and their lengths as weights. Since road distances are typically non-negative, Dijkstra's algorithm is well-suited for this task.

The navigation system can use Dijkstra's algorithm to continually update and provide the shortest route to the destination in real-time, considering traffic conditions and any new information received.

**Bellman-Ford's Algorithm Example:** Network Routing with Quality of Service (QoS) Constraints

In computer networks, routers need to determine the shortest path for packets to travel from a source to a destination while satisfying Quality of Service (QoS) constraints. QoS constraints may include factors like minimum bandwidth requirements, maximum latency thresholds, or maximum packet loss rates.

Bellman-Ford's algorithm can be used to find the shortest paths while considering these QoS constraints. It can handle scenarios where certain paths may have negative weights (e.g., indicating higher bandwidth availability or lower latency).

For example, in a corporate network where different links have varying bandwidth capacities and latency characteristics, Bellman-Ford's algorithm can efficiently find paths that optimize network performance while meeting QoS requirements.

Additionally, Bellman-Ford's algorithm is also used in distributed systems for resource allocation and load balancing, where negative weights may represent resource availability or optimization opportunities.