**Documentation: Network routing tables using Floyd-Warshall algorithm**

# Floyd-Warshall algorithm

The Floyd-Warshall algorithm is a dynamic algorithm used to find the shortest paths between all pairs of vertices in a weighted graph. The algorithm is particularly useful for graphs with both positive and negative edge weights.

It is a versatile tool for network routing, offering a comprehensive solution for finding shortest paths and making routing decisions in dynamic and complex environments.

## How it works

1. Initialization
   1. Initialize the matrices for ‘next\_hop’ and ‘total\_cost’ with default values based on the graphs initial state
2. Main algorithm
   1. We use a triple nested loop to consider all possible intermediate vertices ‘k’, source vertices ‘I’ and destination vertices ‘j’
   2. Update the ‘total\_cost’ matrix if the shorter path is found through the intermediate vertex ‘k’
3. Routing information
   1. The ‘next\_hop’ matrix is keeping track of the next hop on the shortest path from each source to each destination

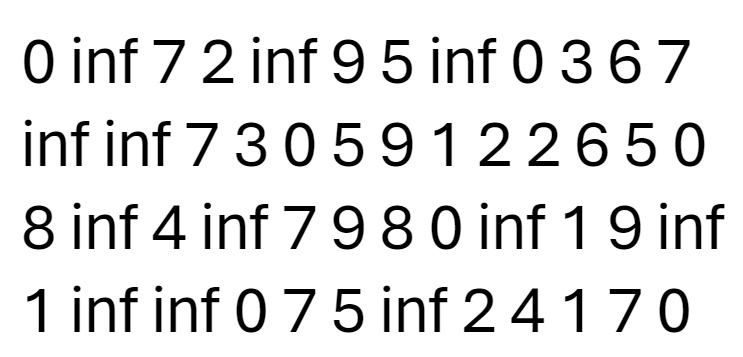
# Routing in networks

* Complete path information
  + Floyd-Warshall computes the shortest paths between all routers, providing complete path information for any source-destination pairs
* Dynamic network
  + It is suitable when the cost of connections can change over time, and the algorithm will be able to adapt to changes without recomputing the entire path
* Handling negative weights
  + Floyd-Warshall can handle graphs with negative weights, this is essential in some specific network scenarios
* Centralized routing decisions
  + When these scenarios have global knowledge, the Floyd-Warshall allows for centralized routing decisions to ensure that each router can efficiently determine the shortest path to any destination
* Robustness
  + Valuable tool for designing routing algorithms in network environments
* Next-hop information
  + Gives information about the next hop on the shortest path, makes efficient packet forwarding

# Test Cases

A number and a number

Description automatically generated with medium confidence



A number and a number

Description automatically generated with medium confidence

# Time complexity

In the Floyd-Warshall algorithm there are implemented 3 nested loops. These loops iterate through all the pairs of vertices so that it can find the shortest path possible. This causes a cubic time complexity to arise. This means the worst-time complexity of the Floyd-Warshall algorithm is O(n^3).

Since the algorithm must go through all potential pairs of vertices, its’ time complexities will all be O(n^3). The n represents the vertices, and since there are 3 loops that the algorithm goes through, we will put 3 as the exponent. To show this we can break down the time complexity for the best case, worst case and average case:

1. Best case time complexity:
   1. The algorithm doesn’t find any short paths, this means that the conditional check inside that triple nested loop is never true. As a result, the algorithm iterates through all pairs of vertices without making any updates to the graph
   2. Because of this the time complexity remains O(n^3), as all loops is still executed
2. Worst case time complexity:
   1. The algorithm finds shorter paths for many pairs of vertices, leading to multiple updates to the graph. The triple-nested loop will run for all pairs of vertices, and checks for all possible intermediate vertices.
   2. This means that the time complexity for worst case also remain to O(n^3)
3. Average case time complexity:
   1. This time complexity is also O(n^3), because you expect the algorithm to find shorter paths for a significant number of vertex pairs.

Regardless of the input, the Floyd-Warshall algorithm has a time complexity of O(n^3), which is making it suitable for relatively small graphs but then is less efficient for larger graphs due to its cubic complexity.