Unit 5 Shortest Path Algorithm

What are the Shortest Path Algorithms?

*The shortest path algorithms are the ones that focuses on calculating the minimum travelling cost from **source node** to **destination node** of a graph in optimal time and space complexities.*

Types of Shortest Path Algorithms:

As we know there are various types of graphs (weighted, unweighted, negative, cyclic, etc.) therefore having a single algorithm that handles all of them efficiently is not possible. In order to tackle different problems, we have different shortest-path algorithms, which can be categorised into two categories:

Single Source Shortest Path Algorithms:

In this algorithm we determine the shortest path of all the nodes in the graph with respect to a single node i.e. we have only one source node in this algorithms.

- Depth-First Search (DFS)
- Breadth-First Search (BFS)
- Dijkstra's algorithm

Let us discuss these algorithms one by one.

Shortest Path Algorithm using Depth-First Search(DFS):

DFS algorithm recursively explores the adjacent nodes untill it reaches to the depth where no more valid recursive calls are possible.

*For DFS to be used as a shortest path algorithm, the graph needs to be acyclic i.e. a **TREE,** the reason it won't work for cyclic graphs is because due to cycles, the destination node can have multiple paths from the source node and dfs will not be able to choose the best path.*

If there does not exist a path between source node and destination node then we can store -1 as the shortest path between those nodes.

Algorithm:

- Distance of source node to source node is initialized to 0.
- Start the DFS from the source node.
- As we go to a neighbouring nodes we set it's distance from source node = edge weight + distance of parent node.
- DFS ends when all the leaf nodes are visited.

Complexity Analysis:

- **Time Complexity**: O(N) where N is the number of nodes in the tree.
- **Auxiliary Space**: O(N) call stacks in case of skewed tree.

Breadth-First Search (BFS) for Shortest Path Algorithm:

BFS is a great shortest path algorithm for all graphs, the path found by breadth first search to any node is the shortest path to that node, i.e the path that contains the smallest number of edges in unweighted graphs.
***Reason**: This works due to the fact that unlike DFS, BFS visits all the neighbouring nodes before exploring a single node to its neighbour. As a result, all nodes with distance 'd' from the source are visited after all the nodes with distance **smaller than d.** In simple terms all the nodes with distance 1 from the node will be visited first, then distance 2 nodes, then 3 and so on.*

Algorithm:

- Use a queue to store a pair of values **{node, distance}**
- Distance of source node to source node is initialized to **0** i.e. push **{src, 0}** to the queue
- While queue is not empty do the following:
    - Get the **{node, distance}** value of top of the queue
    - Pop the top element of the queue

- Store the distance for the node as the shortest distance for that node from the source
- For all non visited neighbours of the current node, push **{neighbour, distance+1}** into the queue and mark it visited

Complexity Analysis:

- **Time Complexity**: O(N) where N is the number of nodes in the graph
- **Auxiliary Space:** O(N) for storing distance for each node.

Multi-Source BFS for Shortest Path Algorithm:

Like BFS, it is also applicable to unweighted graph. To perform a Multi-source search, we basically start BFS from multiple nodes at the same time. When all the BFS meet, we've found the shortest path.

Algorithm:

- Push all source nodes in a Queue with distance 0.
- While the queue is not empty:
  - Get the **{node, distance}** value of top of the queue
  - Pop the top element of the queue
  - Store the distance for the node as the shortest distance for that node from the source
  - For all non visited neighbours of the current node, push **{neighbour, distance+1}** into the queue and mark it visited

Complexity Analysis:

- **Time Complexity:** O(N) where N is the number of nodes in the tree.
- **Auxiliary Space:** O(N) call stacks in case of skewed tree.

Dijkstra's Algorithm for Shortest Path Algorithm:

Dijkstra's algorithm finds the shortest path from a source node to all other nodes in a weighted graph by iteratively selecting the node with the smallest tentative distance and updating the distances to its neighbours. It ensures the shortest path is progressively discovered and is based on the principle of ***greedy optimization***.

Algorithm:

- Create a Set **sptSet** (shortest path tree set) that keeps track of vertices included in the shortest path tree, i.e., whose minimum distance from the source is calculated and finalized. Initially, this set is empty.
- Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign the distance value as **0** for the source vertex so that it is picked first.
- While Set doesn't include all vertices
    - Pick a vertex '**u**' that is not there in Set and has a minimum distance value.
    - Include '**u**' to **sptSet**.
    - Then update the distance value of all adjacent vertices of **u**.
        - To update the distance values, iterate through all adjacent vertices.
        - For every adjacent vertex **v**, if the sum of the distance value of **u** (from source) and weight of edge **u-v**, is less than the distance value of **v**, then update the distance value of **v.**

Complexity Analysis:

- **Time Complexity**: O(E*log(V)), where E is the number of edges and V is the number of vertices in the graph:
- **Auxiliary Space**: O(V)