Graph Traversal Algorithms

Table of Contents

[Depth First Search 2](#_Toc138883450)

[Breadth First Search 2](#_Toc138883451)

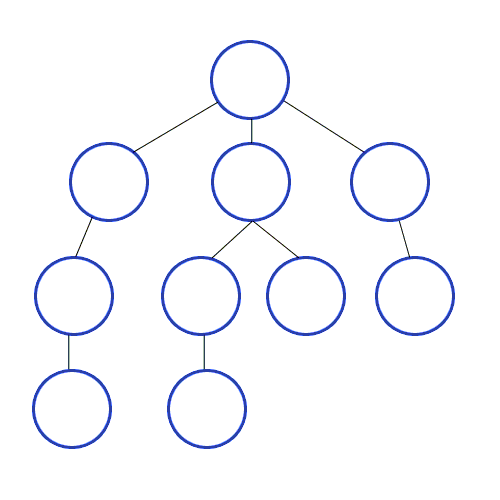
[Dijkstra’s Algorithm 3](#_Toc138883452)

[Bellman-Ford Algorithm 3](#_Toc138883453)

[Floyd-Warshall 4](#_Toc138883454)

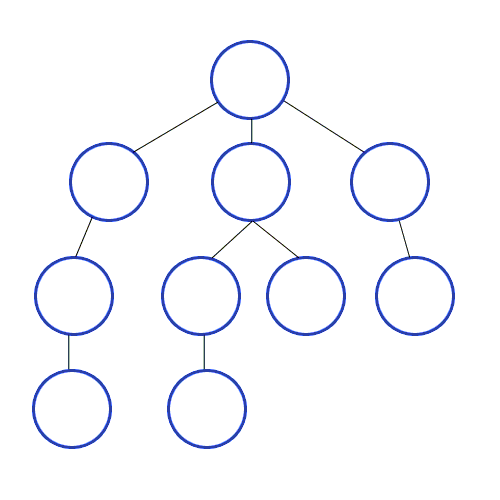
[Ford-Fulkerson Algorithm 5](#_Toc138883455)

## Depth First Search



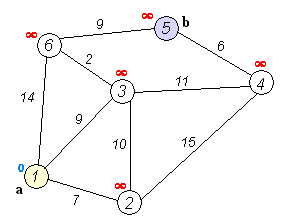
**Depth First Search** (DFS) explores as far as possible along each branch before backtracking. During the traversal, each node is marked as it is visited to prevent revisiting it again. It has a time complexity of .

## Breadth First Search



**Breadth First Search** (BFS) traverses the graph level-by-level. It has a time complexity of .

## Dijkstra’s Algorithm



**Dijkstra’s Algorithm** is used to find the shortest path between a starting node and all other nodes in a graph. It maintains a set of visited and unvisited nodes and assigns a distance to each node that is updated as the algorithm progresses. The edge weights must not be negative. The time complexity of the algorithm is .

## Bellman-Ford Algorithm

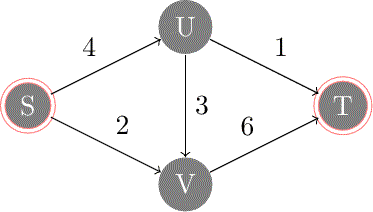


The **Bellman-Ford Algorithm** also finds the shortest distance between one node and every other node, but it can also work with negative edge weights. It maintains an array of distances to each node and ‘relaxes’ all of the edges of the graph times. Essentially, all of the edges are traversed times which results in the shortest paths being found. Finally, it relaxes the edges one more time and checks if the shortest path changes. If it does, it means there is a negative weight cycle and an error is thrown. The algorithm has a time complexity of .

## Floyd-Warshall

The **Floyd-Warshall** algorithm can find the shortest path between all pairs of nodes and also works with negative edge weights. The algorithm maintains a matrix of shortest path distances between any two nodes in the graph. The matrix is initialized with the weights of the edges between adjacent nodes, and the distances between non-adjacent nodes are set to infinity. The algorithm then considers all possible intermediate nodes between any two nodes and checks if the path through the intermediate node results in a shorter distance than the current shortest path between the two nodes. If it does, the shortest path distance is updated. The time complexity of the algorithm is .

## Ford-Fulkerson Algorithm



The **Ford-Fulkerson algorithm** is used to find the maximum flow in a flow network. It works by repeatedly finding an augmenting path in the residual network, which is a modified version of the original network that keeps track of the remaining capacity of the edges.

The algorithm starts by setting the flow on all edges to zero and finding an augmenting path from the source node to the sink node in the residual network using a search algorithm such as DFS or BFS. An augmenting path is a path from the source to the sink that has residual capacity (remaining capacity) on all its edges. It then increases the flow along the augmenting path by the smallest residual capacity on the path. This process is repeated until no more augmenting paths can be found in the residual network. At this point, the flow on each edge in the network represents the maximum flow that can be sent from the source to the sink.

The time complexity of the Ford-Fulkerson algorithm depends on the method used to find augmenting paths. The **Edmonds-Karp algorithm**, a variant of the Ford-Fulkerson algorithm which uses BFS to find augmenting paths, has a time complexity of , where V is the number of vertices and E is the number of edges in the graph.