Brief Algorithms Explaination

This document provides explanations for all the algorithms uploaded on this file, alongside with their complexity for both space and time.

- ## BFS Algorithm

Description: BFS is a graph traversal algorithm that explores a graph level by level. It starts at a source node and visits all its neighbors before moving to the next level of neighbors. It uses a queue to maintain the order of nodes to be visited. BFS is commonly used to find the shortest path in an unweighted graph.

Steps:

1. Enqueue the starting node.
2. While the queue is not empty:
   2.1. Dequeue a node.
   2.2 Mark the node as visited.
   2.3 Enqueue all unvisited neighbors of the dequeued node.

Complexity:

Time Complexity: O(V + E), where V is the number of vertices and E is the number of edges.
Space Complexity: O(V), as the queue can hold all vertices in the worst case.

- ## DFS Algorithm

Description: DFS is a graph traversal algorithm that explores as far as possible along each branch before backtracking. It uses a stack (implicitly through recursion) to maintain the order of nodes to be visited. DFS is commonly used for tasks like detecting cycles, finding connected components, and topological sorting.

Steps:

1. Mark the starting node as visited.
2. For each unvisited neighbor of the current node:
   Recursively call DFS on the neighbor.

Complexity:

Time Complexity: O(V+E).
Space Complexity: O(V) in the worst case (for the recursion stack).

- ## Bellman - Ford Algorithm

Description: The Bellman-Ford algorithm finds the shortest paths from a single source vertex to all other vertices in a weighted graph, even if the graph contains negative-weight edges. It can also detect negative-weight cycles.

Steps:

1. Initialize distances to all vertices as infinity, except for the source vertex, which is set to 0.
2. Relax all edges V - 1 times: For each edge (u, v) with weight w, if dist[u] + w < dist[v], update dist[v] to dist[u] + w.

3. Check for negative-weight cycles: If any edge can still be relaxed after V - 1 iterations, a negative-weight cycle exists.

Complexity:
Time Complexity: O(V•E).
Space Complexity: O(V).

- **Borůvka's Algorithm**

Description: Borůvka's algorithm is a greedy algorithm for finding a minimum spanning tree (MST) in a connected, weighted graph. It operates by repeatedly selecting the minimum-weight edge connecting distinct components until a single component remains.

Steps:

1. Initially, each vertex is its own component.
2. Repeat until only one component remains:
   2.1. For each component, find the minimum-weight edge connecting it to another component.
   2.2. Add these edges to the MST.
   2.3. Merge the connected components.

Complexity:
Time Complexity: O(E•logV).
Space Complexity: O(V+E)

- **Dijkstra's Algorithm**

Description: Dijkstra's algorithm finds the shortest paths from a single source vertex to all other vertices in a weighted graph with non-negative edge weights. It uses a priority queue to efficiently select the vertex with the smallest distance.

Steps:

1. Initialize distances to all vertices as infinity, except for the source vertex, which is set to 0.
2. While there are unvisited vertices:
   2.1. Select the unvisited vertex with the smallest distance.
   2.2. Mark the vertex as visited.
   2.3. For each unvisited neighbor of the selected vertex:
3. If the distance to the neighbor can be shortened, update the distance.

Complexity:
Time Complexity: O(E+V•logV) (using a min-heap).
Space Complexity: O(V).

- **Iterative Deepening Search (IDS)**

Description: IDS is a search algorithm that combines the space efficiency of DFS with the completeness of BFS. It performs a series of depth-limited DFS searches, gradually increasing the depth limit until the goal is found.

Steps:

1. For increasing depth limits:
   1.1. Perform a depth-limited DFS.
   1.2. If the goal is found, return the result.

Complexity:
Time Complexity: $O(b^d)$, where b is the branching factor and d is the depth of the solution. (Same asymptotic complexity as DFS, but with less memory)
Space Complexity: O(d), where d is the depth limit.

- **Kruskal's Algorithm**

Description: Kruskal's algorithm is a greedy algorithm for finding a minimum spanning tree (MST) in a connected, weighted graph. It sorts the edges by weight and adds them to the MST if they do not create a cycle.
Steps:
1. Sort all edges in non-decreasing order of weight.
2. Initialize an empty MST.
3. For each edge (u, v) in the sorted list:
   3.1. If adding (u, v) to the MST does not create a cycle, add it.

Complexity:
Time Complexity: O(E•logE) || O(E•logV) (sorting dominates).
Space Complexity: O(V).