**Week 6: Graph Algorithms**

1. **bfs**:

* **Guide**:
  + Breadth First Search starts at the root node and explores all neighboring nodes at the current depth before moving to nodes at the next depth level. Use a queue to keep track of nodes to be explored.
* **Pseudocode**:
* FUNCTION bfs(graph, start):  
   INITIALIZE a queue with start node  
   INITIALIZE a list to keep track of visited nodes  
   WHILE queue is not empty:  
   TAKE the front node from queue  
   IF node is not visited:  
   ADD node to visited list and ENQUEUE its neighbors  
   RETURN visited nodes list

1. **dfs**:

* **Guide**:
  + Depth First Search explores as far as possible along each branch before backtracking. You can use a stack (or even recursion) to achieve this behavior.
* **Pseudocode**:
* FUNCTION dfs(graph, start):  
   INITIALIZE a stack with start node  
   INITIALIZE a list to keep track of visited nodes  
   WHILE stack is not empty:  
   POP the top node from stack  
   IF node is not visited:  
   ADD node to visited list and PUSH its neighbors to stack  
   RETURN visited nodes list

1. **dijkstra**:

* **Guide**:
  + Dijkstra’s algorithm finds the shortest path from a source node to all other nodes. It uses a priority queue to select the next closest node and updates the distances to neighboring nodes.
* **Pseudocode**:
* FUNCTION dijkstra(graph, start):  
   n = NUMBER OF NODES in graph  
   INITIALIZE distance list 'dist' with infinite distances for all nodes and 0 for start node  
   INITIALIZE set 'unvisited' with all nodes  
    
   WHILE 'unvisited' is not empty:  
   SELECT 'current' node from 'unvisited' having the smallest distance in 'dist'  
   REMOVE 'current' node from 'unvisited'  
    
   FOR each 'neighbor' and its 'weight' from 'graph' corresponding to 'current':  
   IF there is an edge (i.e., non-zero wieght) AND 'neighbor' is in 'unvisited':  
   COMPUTE 'new\_dist' as sum of distance of 'current' and 'weight'  
   IF 'new\_dist' is less than the distance of 'neighbor' in 'dist':  
   UPDATE distance of 'neighbor' in 'dist' with 'new\_dist'  
   RETURN 'dist'