Shortest Path Algorithm for Unweighted Graphs using BFS

* [Finding shortest path in unweighted graphs - BFS](https://www.youtube.com/watch?v=T_m27bhVQQQ)
  + Description
    - Unweighted graph is a special case of the weighted shortest-path problem, with all edges a weight of I
    - Shortest path using BFS only works for unweighted graphs
  + Required data
    - A distance table with three columns (each row corresponds to a vertex)
      * Vertex
      * Previous vertex
      * Distance from the source vertex
    - Queue: (to, from, distance)
      * Vertex we came from (from)
      * Vertex we reached (to)
      * Total cost of the path to the vertex (distance)
  + Algorithm
    - Create the distance table with all the vertices
      * Initialize the distance of the source to 0
      * Initialize the distance of all the other vertices to -1
      * Code: distanceTable = {vertex:(None, 0 if vertex == source else -1) for vertex in graph}
      * Initial State 
        Path and Cost Arrays 
        Queue 
        From 
        To 
        Cost 
    - Add all the neighboring vertices of the source vertex to the queue
      * Queue format: (from = source, to = source.neighbors, distance = 1)
      * Adding the source node to the queue: queue.append((from=None, to=source, distance=-1))
      * Queue 
        A 
        From 
        To 
        Cost 
        A 
        1 
        A 
    - While queue is not empty
      * Current = queue.pop() which will give (from, to, distance)
      * If that vertex we reached is not in the distance table which is if the distance is == -1:
        + distance+1
        + Update the distance table with the vertex we reached
        + For each neighbor of that vertex AND they are not in the distance table:

Add the neighbor with distance+1 to the queue

* + - To get the path from A to B
      * Find B in distance table
        + If the previous vertex is empty then there is no path from A to B
        + If it has a previous vertex, go to that vertex
      * Repeat the process until the path is A
      * Now we have a path from B to A, just reverse to get the path from A to B