**Problem Name: All path from source to target**

**Topics:**

**Companies:**

**Level:** Easy

**Language:** C++

**Problem Statement**:

**Input Format:**

**Output Format:**

**Constraints:**

**Examples:**

**Approach one Solution:**

**Explanation:**

We can start from the starting node 0 and traverse every possible next node from the current node. Whenever we reach the last node n-1, we will add the path till now into the final answer. This process can be implemented using a BFS traversal as -

1. Initialize a queue of path of nodes with the node 0 inserted into it initially denoting the starting node in our traversal path from 0 to n-1
2. Pop an element path from the queue
3. Explore every child node of last node in the path. That is, we try every possible edge in graph from node at the end of current path. Each edge added to end of path gives us another path which will be added to queue for further exploration
4. If the last node in path is n-1, we know that this is a valid source to target path in the graph. So, we add it to final list of paths
5. We repeat this process until the queue is not empty, that is, until there are paths remaining to be explored.

**Code:**

**Time Complexity**: O(N\*2N), where N is the number of nodes in the graph. Every node except the start and end node of graph can either be part of a path or not be part of a path. For a path consisting of k (3 <= k <= n) nodes, we have k-2 intermediate nodes and we can choose from n-2 available nodes. So, the total number of paths are Σ n-2Ck-2 for all 3 <= k <= n, which comes out to be O(2N-2) = O(2N). Each path requires O(N) to be computed. Thus resulting time complexity becomes O(N\*2N)

**Space Complexity:** O(N\*2N), as seen in analysis above, there can be O(2N-2) paths and each path can have a maximum of N nodes in it. The BFS will store all these paths till it reaches the final node which leads to time complexity of O(2N-2 \* (N-1)) = O(N\*2N)

**Optimized Solution:**

Explanation: We can also solve this problem using DFS traversal. This traversal should also be more efficient in terms of space usage as compared to BFS traversal since we are only required to keep a single path in memory at a given time. Note that we dont need to maintain a data structure such as seen to keep track of visited nodes since this is a DAG and thus no recursive dfs call will end up visiting same node twice.

The process of finding all paths using DFS can be implemented as -

1. Begin DFS traversal from the node 0
2. At each step of DFS, add the current node i to the path.
3. If the current node i in dfs call is n-1, we know that this is a valid source to target path in the graph. So, we add it to final list of paths.
4. Otherwise, we explore further path by trying each possible next nodes, that is, we recursively call DFS for every child node of i in the graph.
5. The above process continues till every possible path is tried out by dfs.

**Code:**

**Time Complexity**: O(N\*2N), same as above

**Space Complexity:** O(N), required by max recursive stack depth and for storing path. Generally output space is not considered towards overall space complexity.