All Paths From Source to Target

Given a directed, acyclic graph of `N` nodes. Find all possible paths from node `0` to node `N-1`, and return them in any order.

The graph is given as follows: the nodes are 0, 1, ..., graph.length - 1. graph[i] is a list of all nodes j for which the edge (i, j) exists.

```
Example:
Input: [[1,2], [3], [3], []]
Output: [[0,1,3],[0,2,3]]
Explanation: The graph looks like this:
0--->1
|    |
v    v
2--->3
There are two paths: 0 -> 1 -> 3 and 0 -> 2 -> 3.
```

Note:

- The number of nodes in the graph will be in the range `[2, 15]`.
- You can print different paths in any order, but you should keep the order of nodes inside one path.

Solution 1

```
void dfs(vector<vector<int>>& graph, vector<vector<int>>& result, vector<int> path,
int src, int dst) {
    path.push_back(src);
    if(src == dst) {
        result.push_back(path);
 return;
    }

    for(auto it = graph[src].begin(); it != graph[src].end(); it++)
        dfs(graph, result, path, *it, dst);
}
vector<vector<int>> allPathsSourceTarget(vector<vector<int>>& graph) {
    vector<vector<int>> paths; vector<int> path;
    int nodes = graph.size();
    if(nodes == 0) return paths;
    dfs(graph, paths, path, 0, nodes - 1);
    return paths;
}
```

written by DDev original link here

Solution 2

```
class Solution {
    public List<List<Integer>> allPathsSourceTarget(int[][] graph) {
        List<List<Integer>> res = new ArrayList<>();
        List<Integer> path = new ArrayList<>();

        path.add(0);
        dfsSearch(graph, 0, res, path);

        return res;
    }

    private void dfsSearch(int[][] graph, int node, List<List<Integer>> res, List<In
teger> path) {
        if (node == graph.length - 1) {
            res.add(new ArrayList<Integer>(path));
            return;
        }

        for (int nextNode : graph[node]) {
            path.add(nextNode);
            dfsSearch(graph, nextNode, res, path);
            path.remove(path.size() - 1);
        }
    }
}
```

written by stevenlli original link here

Solution 3

```
    vector<vector<int>> res;
    vector<vector<int>> allPathsSourceTarget(vector<vector<int>>& graph) {
        DFS(graph, {0});
        return res;
    }

    void DFS(vector<vector<int>>& graph, vector<int> path) {
        int node=path.back();
        if(node==graph.size()-1) {
            res.push_back(path);
            return;
        }
        for(int i=0;i<graph[node].size();i++) {
            path.push_back(graph[node][i]);
            DFS(graph, path);
            path.pop_back();
        }
    }
```

written by MichaelZ original link here