Count the paths

Given a directed acyclic graph(DAG) with n nodes labeled from 0 to n-1. Given edges, s and d ,count the number of ways to reach from s to d.There is a directed Edge from vertex edges[i][0] to the vertex edges[i][1].

Example:

```
Input: edges = {{0,1},{0,3},{1,2},{3,2}},
n = 4, s = 0, d = 2
Output: 2
Explanation: There are two ways to reach at
2 from 0. These are-
1. 0->1->2
2. 0->3->2
```

Your Task:
You don't need to read or print anything. Your task is to complete the function **possible_paths()** which takes edges, n, s and d as input parameter and returns the number of ways to reach from s to d.

Expected Time Compelxity: $O(2^n)$
Expected Space Complexity: O(n+e)

where e is the number of edges in the graph.

Constraints:
1 <= n <= 15
0 <= s, d <= n-1

```
class Solution:
    def possible_paths(self, edges, n, s, d):
        #Code here
        ans = [0]
        visited = [False]*n
        graph = {}
        for src,dest in edges:
            if src in graph:
                graph[src].append(dest)
            else:
                graph[src] = [dest]
        self.findAllPath(graph,s,d,visited,ans)
        return ans[0]
```

```
def findAllPath(self,graph,src,dest,visited,ans):
    if src==dest:
        ans[0]+=1
        return
    visited[src]=True
    for nbr in graph[src]:
        if visited[nbr]==False:
            self.findAllPath(graph,nbr,dest,visited,ans)
    visited[src]=False
```