```
/*
二维数组的第 i 个数组中的单元都表示有向图中 i 号结点所能到达的下一些结点（译者注：有向图是有方向的，即规定了a→b你就不能从b→a）空就是没有下一个结点了。

示例:
输入: [[1,2], [3], [3], []]
输出: [[0,1,3],[0,2,3]]
解释: 图是这样的:
0--->1
|    |
v    v
2--->3
这有两条路: 0 -> 1 -> 3 和 0 -> 2 -> 3.

提示:

    结点的数量会在范围 [2, 15] 内。
    你可以把路径以任意顺序输出，但在路径内的结点的顺序必须保证。

来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/all-paths-from-source-to-target
著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。
*/
```

分析:

- 对邻接表存储的有向图进行深度优先遍历.
  - 为防止成环情况,可以对容器长度进行检测(能屏蔽一些成环情况,但用例没有测试出来).
  - 不过也没说带环的怎么处理，先这样吧.
  - 为防止成环情况,也可以申请和维护visted数组(未实现,todo).

方法一:C++_DFS+检测容器长度.

```
class Solution
{
    private:
        vector<vector<int>> ret_val;
        int max_level;
        void __visited(vector<vector<int>>& graph, int level, vector<int> last_vec)
        {

            vector<int> cur_vec(last_vec);

            cur_vec.push_back(level);

            if (level == max_level)
            {
                ret_val.push_back(cur_vec);
                return;
            }

```

```
            /*防止成环的死循环*/
            if(cur_vec.size() > max_level)
            {
                return ;
            }
            for (int i = 0; i < graph[level].size(); i++)
            {
                __visited(graph, graph[level][i], cur_vec);
            }
        }

    public:
        /* graph 是邻接表保存的*/
        vector<vector<int>> allPathsSourceTarget(vector<vector<int>>&
graph)
        {

            vector<int> temp;
            ret_val.clear();
            max_level = graph.size() - 1;
            if(max_level >= 0)
            {
                __visited(graph,0,temp);
            }
            return ret_val;
        }
};
/*
执行结果:
通过
显示详情
执行用时 :276 ms, 在所有 C++ 提交中击败了6.94% 的用户
内存消耗 :20.4 MB, 在所有 C++ 提交中击败了12.00%的用户
*/
```

AlimyBreak
2019.09.23