有 `N` 个房间，开始时你位于 `0` 号房间。每个房间有不同的号码：`0, 1, 2, ..., N-1`，并且房间里可能有一些钥匙能使你进入下一个房间。

在形式上，对于每个房间 `i` 都有一个钥匙列表 `rooms[i]`，每个钥匙 `rooms[i][j]` 由 `[0,1, ..., N-1]` 中的一个整数表示，其中 `N = rooms.length`。 钥匙 `rooms[i][j] = v` 可以打开编号为 `v` 的房间。

最初，除 `0` 号房间外的其余所有房间都被锁住。

你可以自由地在房间之间来回走动。

如果能进入每个房间返回 `true`，否则返回 `false`。

示例 1：

输入： `[[1],[2],[3],[]]`
输出： `true`
解释：
我们从 `0` 号房间开始，拿到钥匙 `1`。
之后我们去 `1` 号房间，拿到钥匙 `2`。
然后我们去 `2` 号房间，拿到钥匙 `3`。
最后我们去了 `3` 号房间。
由于我们能够进入每个房间，我们返回 `true`。

示例 2：

输入： `[[1,3],[3,0,1],[2],[0]]`
输出： `false`
解释： 我们不能进入 `2` 号房间。

提示：

1. `1 <= rooms.length <= 1000`
2. `0 <= rooms[i].length <= 1000`
3. 所有房间中的钥匙数量总计不超过 `3000`。

分析，直接用递归，模拟拿到钥匙，打开房门，继续拿钥匙的过程，用map来标记当前递归一共拿到了多少串钥匙，能开哪些房门，递归结束后，遍历数组，看看是否已经拿到所有房间的钥匙，拿到就返回true，否则返回false.

```
class Solution {
public:
    void dfs(vector<vector<int>>& rooms,int i,map<int,int> &keys){
        for(int j=0;j<rooms[i].size();j++){
            if(keys[rooms[i][j]]==0){
                keys[rooms[i][j]]=1;
```

```
                dfs(rooms,rooms[i][j],keys);
            }
        }
    }
    bool canVisitAllRooms(vector<vector<int>>& rooms) {
        map<int,int> keys;
        dfs(rooms,0,keys);
        for(int i=1;i<rooms.size();i++){
            if(keys[i]==0){
                return false;
            }
        }
        return true;
    }
};
```