```
/*
给定一个二维网格 board 和一个字典中的单词列表 words，找出所有同时在二维网格和字典中出现的单词。

单词必须按照字母顺序，通过相邻的单元格内的字母构成，其中“相邻”单元格是那些水平相邻或垂直相邻的单元格。同一个单元格内的字母在一个单词中不允许被重复使用。

示例:

输入:
words = ["oath","pea","eat","rain"] and board =
[
  ['o','a','a','n'],
  ['e','t','a','e'],
  ['i','h','k','r'],
  ['i','f','l','v']
]

输出: ["eat","oath"]

说明:
你可以假设所有输入都由小写字母 a-z 组成。

提示:

    你需要优化回溯算法以通过更大数据量的测试。你能否早点停止回溯？
    如果当前单词不存在于所有单词的前缀中，则可以立即停止回溯。什么样的数据结构可以有效地执行这样的操作？散列表是否可行？为什么？ 前缀树如何？如果你想学习如何实现一个基本的前缀树，请先查看这个问题： 实现Trie（前缀树）。

来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/word-search-ii
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:

- 普通回溯法,去重可以考虑在words或在结果中做.
- 回溯法+前缀树,未实现,todo.

方法一:C++_普通回溯法+去重

```
class Solution
{

private:
    int arr[8] = { -1,0,+1,0,0,-1,0,+1 };
    bool helper(    vector<vector<char>>&       board       ,
                    string&                     word        ,
                    int                         cur_idx     ,
                    int                         cur_row     ,
                    int                         cur_col     ,
```

```
                    vector<vector<int>>&        visited
    )
    {
        if(cur_idx == word.size())
        {
            return true;
        }

        if(     cur_row < 0
            ||  cur_row >= board.size()
            ||  cur_col < 0
            ||  cur_col >= board[0].size()
            ||  visited[cur_row][cur_col]
        )
        {
            return false;
        }

        if(board[cur_row][cur_col] == word[cur_idx])
        {
            visited[cur_row][cur_col] = 1;
            for (int i = 0; i < 4; i++)
            {
                if (helper(board, word, cur_idx + 1, cur_row + arr[2 * i],
cur_col + arr[2 * i + 1], visited))
                {
                    return true;
                }
            }
            visited[cur_row][cur_col] = 0;
            return false;

        }
        else
        {
            return false;
        }


    }

public:
    vector<string> findWords(vector<vector<char>>& board, vector<string>&
words)
    {
        vector<string>  vs;
        set<string>     svc;

        for (int k = 0; k < words.size(); k++)
        {
            vector<vector<int>> visited = vector<vector<int>>(board.size(),
vector<int>(board[0].size(), 0));
            for (int i = 0; i < board.size();i++)
            {
                for (int j = 0; j < board[0].size(); j++)
                {
                    if (helper(board, words[k], 0, i, j, visited))
                    {
```

```
                        //vs.push_back(words[k]);
                        svc.insert(string(words[k]));
                    }
                }
            }

        }

        set<string>::iterator it;
        for(it = svc.begin();it != svc.end();it++)
        {
            vs.push_back(*it);
        }
        return vs;
    }
};
/*
执行结果:
通过
显示详情
执行用时 :1548 ms, 在所有 cpp 提交中击败了6.72% 的用户
内存消耗 :32.1 MB, 在所有 cpp 提交中击败了61.75%的用户
*/
```

AlimyBreak
2019.12.05