0212. 单词搜索 II

ITCharge 大约 2 分钟

- 标签：字典树、数组、字符串、回溯、矩阵
- 难度：困难

题目链接

- 0212. 单词搜索 II - 力扣

题目大意

给定一个 `m * n` 二维字符网格 `board` 和一个单词（字符串）列表 `words` 。

要求：找出所有同时在二维网格和字典中出现的单词。

注意：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。

解题思路

- 先将单词列表 `words` 中的所有单词存入字典树中。
- 然后遍历二维字符网络 `board` 的每一个字符 `board[i][j]` 。
- 从当前单元格出发，从上下左右四个方向深度优先搜索遍历路径。每经过一个单元格，就将该单元格的字母修改为特殊字符，避免重复遍历，深度优先搜索完毕之后再恢复该单元格。
  - 如果当前路径恰好是 `words` 列表中的单词，则将结果添加到答案数组中。
  - 如果是 `words` 列表中单词的前缀，则继续搜索。
  - 如果不是 `words` 列表中单词的前缀，则停止搜索。
- 最后输出答案数组。

代码

```py
class Trie:

    def __init__(self):
        """
        Initialize your data structure here.
        """
        self.children = dict()
        self.isEnd = False
        self.word = ""


    def insert(self, word: str) -> None:
        """
        Inserts a word into the trie.
        """
        cur = self
        for ch in word:
            if ch not in cur.children:
                cur.children[ch] = Trie()
            cur = cur.children[ch]
        cur.isEnd = True
        cur.word = word


    def search(self, word: str) -> bool:
        """
        Returns if the word is in the trie.
        """
        cur = self
        for ch in word:
            if ch not in cur.children:
                return False
            cur = cur.children[ch]

        return cur is not None and cur.isEnd

class Solution:
```

```
def findWords(self, board: List[List[str]], words: List[str]) -> List[str]:
    trie_tree = Trie()
    for word in words:
        trie_tree.insert(word)

    directs = [(0, 1), (0, -1), (1, 0), (-1, 0)]
    rows = len(board)
    cols = len(board[0])

    def dfs(cur, row, col):
        ch = board[row][col]
        if ch not in cur.children:
            return

        cur = cur.children[ch]
        if cur.isEnd:
            ans.add(cur.word)

        board[row][col] = "#"
        for direct in directs:
            new_row = row + direct[0]
            new_col = col + direct[1]
            if 0 <= new_row < rows and 0 <= new_col < cols:
                dfs(cur, new    , new_col)
        board[row][col] = ch

    ans = set()
    for i in range(rows):
        for j in range(cols):
            dfs(trie_tree, i, j)

    return list(ans)
```