2023-09-16 - Handout – Tries

Q1. Word Break

Link:: https://leetcode.com/problems/word-break/

Given a string s and a dictionary of strings wordDict, return true if s can be segmented into a space-separated sequence of one or more dictionary words.

Note that the same word in the dictionary may be reused multiple times in the segmentation.

Constraints:

- 1 <= s.length <= 300
- 1 <= wordDict.length <= 1000
- 1 <= wordDict[i].length <= 20
- s and wordDict[i] consist of only lowercase English letters
- All strings of wordDict are **unique**

Example 1:

Input: s = "leetcode", wordDict = ["leet","code"]

Output: true

Explanation: Return true because "leetcode" can be segmented as "leet code".

Example 2:

Input: s = "applepenapple", wordDict = ["apple","pen"]

Output: true

Explanation: Return true because "applepenapple" can be segmented as "apple pen apple".

Note that you are allowed to reuse a dictionary word.

Example 3:

Input: s = "catsandog", wordDict = ["cats","dog","sand","and","cat"]

Output: false

Q2. Design Add and Search Words Data Structure

Link: https://leetcode.com/problems/design-add-and-search-words-data-structure

Design a data structure that supports adding new words and finding if a string matches any previously added string.

Implement the WordDictionary class:

- WordDictionary () Initializes the object.
- Void addword(word) Adds word to the data structure, it can be matched later.
- Bool search(word) Returns true if there is any string in the data structure that matches word or false

otherwise. Word may contain dots '.' where dots can be matched with any letter.

Constraints:

- 1 <= word.length <= 25
- word in addWord consists of lowercase English letters
- word in search consist of '.' or lowercase English letters.
- There will be at most 2 dots in word for search queries
- At most 104 calls will be made to addWord and search.

Example:

Input

```
["WordDictionary","addWord","addWord","addWord","search","search","search","search"
]
[[],["bad"],["dad"],["mad"],["pad"],["bad"],[".ad"],["b.."]]
```

Output

```
[null,null,null,null,false,true,true,true]
```

Explanation

```
WordDictionary wordDictionary = new WordDictionary();
wordDictionary.addWord("bad");
wordDictionary.addWord("dad");
wordDictionary.addWord("mad");
wordDictionary.search("pad"); // return False
wordDictionary.search("bad"); // return True
wordDictionary.search(".ad"); // return True
wordDictionary.search("b.."); // return True
```

Q3. Word Search II

Link: https://leetcode.com/problems/word-search-ii

Given an mxn board of characters and a list of strings words, return *all words on the board*.

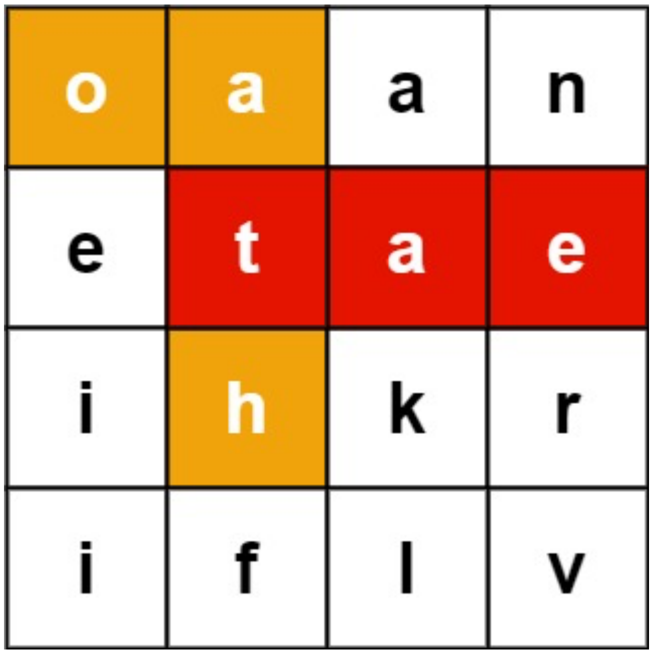
Each word must be constructed from letters of sequentially adjacent cells, where **adjacent cells** are horizontally or vertically neighboring. The same letter cell may not be used more than once in a word.

Constraints:

- m == board.length
- n == board[i].length
- 1 <= m, n <= 12
- board[i][j] is a lowercase English letter
- 1 <= words.length <= 3*104
- 1 <= words[i].length <= 10

- words[i] consists of lowercase English letters
- All the strings of words are unique.

Example 1:



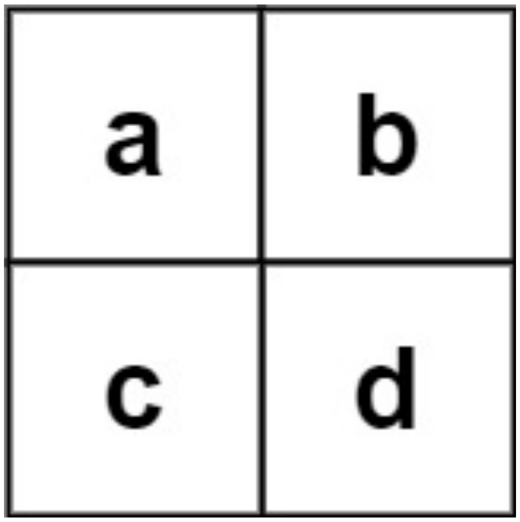
Input:

```
board                                            =
[["o","a","a","n"],["e","t","a","e"],["i"
,"h","k","r"],["i","f","l","v"]],

words = ["oath","pea","eat","rain"]
```

Output: `["eat","oath"]`

Example 2:



Input:

```
board = [["a","b"],["c","d"]],

words = ["abcb"]
```

Output: `[]`