HW 12. Introduction to algorithms

This is twelve homework for CS 566.

Task 1. Solve the problem "Word Search II" from https://leetcode.com/problems/word-search-ii/description/ using Python3.

Use the box below, to paste the working code. The format of the code should be identical to LeetCode platform. (4 points)

```
from typing import List

class TrieNode:
    def __init__(self):
        self.children = {}
        self.is_end = False

class Trie:
    def __init__(self):
        self.root = TrieNode()

    def insert(self, word: str):
        node = self.root
        for char in word:
            if char not in node.children:
                node.children[char] = TrieNode()
            node = node.children[char]
        node.is_end = True

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

        self.result = set()
        self.rows, self.cols = len(board), len(board[0])
        self.board = board

        def backtrack(r, c, node, path):
            if node.is_end:
                self.result.add(path)
                node.is_end = False
            if r < 0 or r >= self.rows or c < 0 or c >= self.cols:
                return
```

```
            tmp = self.board[r][c]
            if tmp == "#" or tmp not in node.children:
                return

            self.board[r][c] = "#"  # Mark as visited
            for dr, dc in [(0, 1), (1, 0), (0, -1), (-1, 0)]:
                backtrack(r + dr, c + dc, node.children[tmp], path + tmp)
            self.board[r][c] = tmp  # Restore the cell

        for r in range(self.rows):
            for c in range(self.cols):
                backtrack(r, c, trie.root, "")

        return list(self.result)
```

Do not modify the testing code below. If you get message "Mistake in test case #", it means that you algorithm is incorrect.

```
#test_case_1
board = [["o","a","a","n"],["e","t","a","e"],["i","h","k","r"],["i","f","l","v"]]
words = ["oath","pea","eat","rain"]
actual = list(sorted(Solution().findWords(board, words)))
expected = list(sorted(["eat","oath"]))
assert actual==expected, "Mistake in test case 1"

board = [["a","b"],["c","d"]]
words = ["abcb"]
actual = list(sorted(Solution().findWords(board, words)))
expected = []
assert actual==expected, "Mistake in test case 2"


print("OK")
```

```
OK
```

Write analysis of the Memory Complexity and Time Complexity using Aymptotic Notation O. (1 point)

Memory Analysis: O(w * l+m*n), where l is average length of a word, w is number of words, and m,n is the row and column of the board

Time Analysis: O(w * l +m^2*n^2), where l is average length of a word, w is number of words, and m,n is the row and column of the board

Task 2. Solve the problem "Prefix and Suffix Search" from https://leetcode.com/problems/prefix-and-suffix-search/

- https://leetcode.com/problems/prefix-and-suffix-search/description/ using Python3.

Use the box below, to paste the working code. The format of the code should be identical to LeetCode platform. (4 points)

```
from collections import defaultdict, deque


class WordFilter:
    def __init__(self, words: List[str]):
        self.prefix_suffix_index = defaultdict(lambda: -1)
        for idx, w in enumerate(words):
            for prefix in range(len(w)):
                for suffix in range(len(w)):
                    self.prefix_suffix_index[(w[:prefix+1], w[suffix:])] = idx

    def f(self, prefix: str, suffix: str) -> int:
        return self.prefix_suffix_index[(prefix, suffix)]



# Your WordFilter object will be instantiated and called as such:
# obj = WordFilter(words)
# param_1 = obj.f(pref,suff)
```

```
wordFilter = WordFilter(["apple"]);
actual = wordFilter.f("a", "e")
assert 0 == actual, "Mistake in test case 1"
print("OK")
```

```
OK
```

Write analysis of the Memory Complexity and Time Complexity using Asymptotic Notation O. (1 point)

Memory Analysis: O(m^3), where m is word length

Time Analysis: O(n * m^3), where n is words and m is word length

- Task 3. Solve the problem "Word Break II" from https://leetcode.com/problems/word-break-ii/description// using Python3.

Use the box below, to paste the working code. The format of the code should be identical to

LeetCode platform. (4 points)

```
from typing import List
class Solution:
    def wordBreak(self, s: str, wordDict: List[str]) -> List[str]:
        word_set = set(wordDict)
        memo = {}

        def backtrack(start):
            if start in memo:
                return memo[start]
            if start == len(s):
                return [""]

            results = []
            for end in range(start + 1, len(s) + 1):
                word = s[start:end]
                if word in word_set:
                    for sub_sentence in backtrack(end):
                        sentence = word + ("" if sub_sentence == "" else " " + s
                        results.append(sentence)

            memo[start] = results
            return results

        return backtrack(0)
```

```
s = "catsanddog"
wordDict = ["cat","cats","and","sand","dog"]
expected = ["cats and dog","cat sand dog"]
actual = Solution().wordBreak(s, wordDict)
assert set(expected) == set(actual), "Mistake in test case 1"
print("OK")
```

```
OK
```

Write analysis of the Memory Complexity and Time Complexity using Asymptotic Notation O. (1 point)

Memory Analysis: O(n)

Time Analysis: O(n^2 * m)