Word-Search-2

Group: Ardrian Wong (UNI:aaw2179), Kevin Lo (UNI:kl3695), Sean Zhang (UNI:srz2116)

Problem Statement

Given an m x n board of characters and a list of string 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.

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"]

Reference: https://leetcode.com/problems/word-search-ii/description/

Sequential Algorithm:

Below is the python algorithm we plan to implement in Haskell.

Python

```python
class Solution:
    def findWords(self, board: List[List[str]], words: List[str]) -> List[str]:
        root = TrieNode()
        for word in words:
            curr = root
            for c in word:
                if c not in curr.children:
                    curr.children[c] = TrieNode()
                curr = curr.children[c]
            curr.end = True

        m = len(board)
        n = len(board[0])
        res = set()
        arr = []
        def dfs(r,c, node):
            if node.end:
                res.add("".join(arr))
```

```
            if r < 0 or r == m or c < 0 or c == n or board[r][c] == "MARKED" or
board[r][c] not in node.children:
                return

            arr.append(board[r][c])

            v = board[r][c]
            board[r][c] = "MARKED"

            for y, x in [(1,0),(0,1),(-1,0),(0,-1)]:
                newR = y + r
                newC = x + c
                dfs(newR, newC, node.children[v])

            arr.pop()
            board[r][c] = v
            if not node.children[v].children:
                node.children.pop(v)

            return

        for i in range(m):
            for j in range(n):
                if board[i][j] in root.children:
                    dfs(i, j, root)
        return list(res)

class TrieNode:
    def __init__(self):
        self.children = {} # "letter": node
        self.end = False
```

- Generate a trie for the target words
- For every position in the grid, perform dfs to find target words
    - Perform validation checks for the current (i,j) position in the grid
    - Mark the current position in the board to avoid reexploring it
    - For the current position, explore in all 4 directions
    - Unmark current position and prune trie

Methods of Parallelism:

- Break grid into subgrids and have a thread search dfs for each subgrid
- Parallelize recursive calls up to some depth N
- Parallelize search for each target word

We could utilize some combination of the Par, Eval, Strategies Monads to start sparks for different aspects we want to parallelize.

Evaluation Data:

We plan to either come up with test cases of our own as well as utilize existing test cases from websites like leetcode.com to compare performance of our sequential algorithm versus our parallelization attempts.