TRIES

No.	Problem Statement	Solution	Time complexity	Space complexity
1	**Implement Trie Prefix Tree**			
	Implement Trie (void Insert(), bool Search(), bool StartsWith())	- Each Trie node contains pointers to its children which are also Trie nodes & isEnd flag	O(n) Insert O(n) Search O(n) startsWith	O(n) Insert O(1) Search O(1) startsWith
2	**Design Add and Search Word Data Strucutre**			

	Design a data structure that supports
	- void addWords(): Adding new words
	- bool search(): Finding there is any string in the data structure that matches word
		- Use **Trie**
		- search(): If the current character is a dot (.), Recurse for all possible child nodes
			O(m*n) addWord
			O(m* 26^n) Search
			m = # words
			n = avg length of strings in words
				O(n) add Word
				O(n) Search

3	**Word Search II**			
	Given a board of characters & a list of words, return all words from the list that can be generated on the board	- Use **Trie** to store the list of words - For each cell on the board, call 'search' to explore possible words starting from that cell. - Search() - The character in the current cell is used to navigate the trie. - If the character does not exist in the current trie node's children, the search from this cell ends. - If the end of a word (isEnd) is reached in the trie, that word is added to the answer list, and its isEnd flag is reset to avoid duplicate words in the result. - Then recursively calls itself for all adjacent cells (up, down, left, and right). - After exploring all possibilities from a cell, the cell's original value is restored so that it can be used in other search paths.	O(m*n*k*4^l) k = words.size() l = avg length of strings in words	O(k*l) -- Trie O(l) -- Recursive