A Wordsearch

Here is a video walkthrough of the solutions.

Here is justification of the runtime.

Given an N by N wordsearch and N words, devise an algorithm to solve the wordsearch in O(N^3). For simplicity, assume no word is contained within another, i.e. if the word "bear" is given, "be" wouldn't also be given.

If you are unfamiliar with wordsearches or want to gain some wordsearch solving intuition, see below for an example wordsearch. Note that the below wordsearch doesn't follow the precise specification of an N by N wordsearch with N words, but your algorithm should work on this wordsearch regardless.

Example Wordsearch:

C	M	U	H	O	S	A	E	D
T	R	A	T	H	A	N	K	A
O	C	Y	E	S	R	T	U	T
N	I	R	S	A	I	O	L	S
Y	R	R	M	T	N	N	H	R
Y	E	A	E	V	A	R	U	E
A	A	A	I	M	E	L	C	R
N	H	D	J	Y	U	A	C	I
T	Y	S	A	A	R	S	U	C
A	R	S	I	G	Y	E	S	A

ajay	anton
crystal	eric
grace	isha
luke	naama
rica	sarina
sherry	shreyas
sohum	sumer
tony	vidya

Hint: Add the words to a Trie, and you may find the `longestPrefixOf` operation helpful. Recall that `longestPrefixOf` accepts a `String key` and returns the longest prefix of `key` that exists in the `Trie`, or **`null`** if no prefix exists.

Algorithm: Begin by adding all the words we are querying for into a `Trie`. Next, we will iterate through each letter in the wordsearch and see if any words *start* with that letter. For a word to start with a given letter, note that it can go in one of eight directions — N, NE, E, SE, S, SW, W, NW.

Looking at each direction, we will check if the string going in that direction has a prefix that exists in our `Trie`, which we can do using `longestPrefixOf`. Note that words are not nested inside of others, so *at most* one word can start from a given letter in a given direction. As such, if `longestPrefixOf` returns a word, we know it is the only word that goes in that direction from that letter.

For instance, if we are at the letter "S" in the middle of the top row of the wordsearch above and are considering the direction west, we would want to see if the string `"SOHUMC"` has a prefix that exists in the given wordsearch. To efficiently perform this query, we call `longestPrefixOf("SOHUMC")`, which, in this case, returns `"SOHUM"`, and we proceed by removing `"SOHUM"` from our `Trie` to signal that we found the word `"SOHUM"`.

We will repeat this process until the all the words have been found, i.e. when the `Trie` is empty. Finally, note that this is a very open ended problem, so this is one of *many* possible solutions.

Runtime: We look at N^2 letters. At each letter, we execute eight calls to `longestPrefixOf` which runs in time linear to the length of the inputted string, which can be of at most length N, since that is the height and width of the wordsearch. Thus, if we perform on the order of N work per letter and we look at N^2 letters, the runtime is $O(N^3)$.