Northern Michigan University (Marquette Co, MI)

CS422-01-25W: Algorithms (Andrew A. Poe) Name: _____
Quiz 10 Friday 18 April 2025 9:00 A.M. EDT

Time: 15 minutes

Imagine a trie stores only strings consisting entirely of capital letters. Imagine as well that a search string consists only of capital letters and *'s, * being a wildcard character. The wildcard character * can expand to any string of capital letters, including the empty string. A search string can as many *'s as it wants to have, and maybe it won't have any.

So, A*B will match ACB, ACATB, AZYXWVB, AAABBB, and AB (and many others).

Given a trie and a search string, design a recursive algorithm that will print all strings in the trie that match the search string. Your algorithm should use recursion to move down the trie; it should never move up the trie. Your algorithm should make use of the first character in the search string to decide which path or paths to take in the trie, and you should remove the first character in the search string as you move down the trie, but you shouldn't look at any character beyond the first character at any given step.

Be sure to list all relevant base cases and what you should do when these base cases are encountered. Be clear what happens at each recursive case.

Do not use any extra aggregate data structures for this. No arrays or linked lists or anything. Just move down the trie one node at a time and remove characters as appropriate from the beginning of the search string.

You do not have to write code for this. A clear English description is all I'm looking for.

If the search string is empty,
if you're at an end-of-string marker, you found a string, print it and return! (BASE CASE)
if you're not at an end-of-string marker, the search wasn't found (on this branch). Simply return.

If the first character in the search string is not *
If you do not have a child node corresponding to the first character, there is no string on this branch that works. Simply return. (BASE CASE)
If you have a child node corresponding to the first character, recursively run the algorithm starting from that child node, using the search string minus the first character.

If the first character in the string is *
Recursively run the algorithm on every child node using an unmodified search string (keep the * in there)!
Also recursively run the algorithm on your own node with the initial * removed from the search string.