

Problem: Implement a trie with find, insert, remove, and list_words methods.

Constraints

- Can we assume we are working with strings?
 - Yes
- Are the strings in ASCII?
 - Yes
- Should find only match exact words with a terminating character?
 - Yes
- Should list_words only return words with a terminating character?
 - Yes
- Can we assume this fits memory?
 - Yes

Test Cases

root node is denoted by "

```
"
 / | \
h  a* m
 /\  \ \
a  e* t* e*
 /\      /\
s* t*   n* t*
 /
s*
```

find

- * Find on an empty trie
- * Find non-matching
- * Find matching

insert

- * Insert on empty trie
- * Insert to make a leaf terminator char
- * Insert to extend an existing terminator char

remove

- * Remove me
- * Remove mens
- * Remove a
- * Remove has

list_words

- * List empty
- * List general case

Algorithm

find

- Set node to the root
- For each char in the input word
 - Check the current node's children to see if it contains the char
 - If a child has the char, set node to the child
 - Else, return None
- Return the last child node if it has a terminator, else None

Complexity:

- Time: $O(m)$, where m is the length of the word
- Space: $O(h)$ for the recursion depth (tree height), or $O(1)$ if using an iterative approach

insert

- set node to the root
- For each char in the input word
 - Check the current node's children to see if it contains the char
 - If a child has the char, set node to the child
 - Else, insert a new node with the char
 - Update children and parents
- Set the last node as a terminating node

Complexity:

- Time: $O(m)$, where m is the length of the word
- Space: $O(h)$ for the recursion depth (tree height), or $O(1)$ if using an iterative approach

remove

- Determine the matching terminating node by calling the find method

- If the matching node has children, remove the terminator to prevent orphaning its children
- Set the parent node to the matching node's parent
- We'll be looping up the parent chain to propagate the delete
- While the parent is valid
 - If the node has children
 - Return to prevent orphaning its remaining children
 - If the node is a terminating node and it isn't the original matching node from the find call
 - Return to prevent deleting this additional valid word
 - Remove the parent node's child entry matching the node
 - Set the node to the parent
 - Set the parent to the parent's parent

Complexity:

- Time: $O(m+h)$, where m is the length of the word and h is the tree height
- Space: $O(h)$ for the recursion depth (tree height), or $O(1)$ if using an iterative approach

list_words

- Do a pre-order traversal, passing down the current word
 - When you reach a terminating node, add it to the list of results

Complexity:

- Time: $O(n)$
- Space: $O(h)$ for the recursion depth (tree height), or $O(1)$ if using an iterative approach