Design Add and Search Words Data Structure

Try to solve the Design Add and Search Words Data Structure problem.

We'll cover the following

Statement

Design a data structure called **WordDictionary** that supports the following functionalities:

- **Constructor:** This function will initialize the object.
- **Add Word(word):** This function will store the provided word in the data structure.
- **Search Word(word):** This function will return TRUE if any string in the **WordDictionary** object matches the query word. Otherwise, it will return FALSE. If the query word contains dots, `.`, each dot is free to match any letter of the alphabet.

  For example, the dot in the string ".ad" can have 26 possible search results like "aad", "bad", "cad", and so on.
- **Get Words():** This function will return all the words in the **WordDictionary** class.

Constraints:

- $1 \leq$ `word.length` ≤ 25
- Words passed to **Add Word()** consist of lowercase English letters.
- Words passed to **Search Word()** consist of `.` or lowercase English letters.
- There will be, at most, three dots in a word passed to **Search Word()**.
- At most, 10^3 calls will be made to **Add Word()** and **Search Word()**.

Examples

Sample example 1

Input

```
add word("bin")
```

Output

```
dictionary
bin
```

Understand the problem

Let’s take a moment to make sure you’ve correctly understood the problem. The quiz below helps you check if you’re solving the correct problem:

Design Add and Search Words Data Structure

1 Consider the list [“bin”, “pin”, “spin”, “pint”]. What will be the output of **Search Word(“bin”)**?

A) True

B) False

Submit Answer

Question 1 of 4
0 attempted

Reset Quiz

Figure it out!

We have a game for you to play. Rearrange the logical building blocks to develop a clearer understanding of how to solve this problem.

> **Note:** The game below is only for the **Add Word(word)** function.

Drag and drop the cards to rearrange them in the correct sequence.

Start from the root node of the trie, and traverse the word one character at a time.

If the character is not present in the trie, then create a new trie node for it, otherwise use the existing trie node for this character.

Once all the characters of the current word are added to the trie, set a boolean variable to

TRUE, marking this as the end of the word.

Calculate the index of the character between 1 and 26, and check whether the character already exists in the trie.

If we've reached the end of the word and the boolean flag is already set, this means that it is already present in the dictionary, so we return TRUE.

Reset | Show Solution | Submit

Try it yourself

Implement your solution in `WordDictionary.java` in the following coding playground. You will need the provided supporting code to implement your solution. We have also provided a useful code template that you may build on to solve this problem.

Java

WordDictionary.java
DepthFS.java
TrieNode.java
Trie.java

```
1  import java.util.*;
2
3  class WordDictionary {
4
5      public WordDictionary() {
6          // Write your code here
7      }
8
9      // get all words in the dictionary
10     public List<String> getWords() {
11
14     }
15
16     // adding a new word to the dictionary
17     public void addWord(String word) {
18
19         // Write your code here
20         return;
21     }
22
23     // searching for a word in the dictionary
24     public boolean search(String word) {
25
26         // Write your code here
27         return false;
28     }
```

Powered by AI | Submit

Test Cases | Results

Case 1 | Case 2 | Case 3

Input #1

["WordDictionary","addWord","addWord","addWord","getWords","searchWord","searchWord","searchWord","searchWord","getWords"]

Input #2

[[],["bad"],["dad"],["mad"],[],["pad"],["bad"],[".ad"],["b.."],[]]

Design Add and Search Words Data Structure

Back

Solution: Replace Wor...

Next

Solution: Design Add ...

Mark as Completed