676. Implement Magic Dictionary

Description

Design a data structure that is initialized with a list of **different** words. Provided a string, you should determine if you can change exactly one character in this string to match any word in the data structure.

Implement the `MagicDictionary` class:

- `MagicDictionary()` Initializes the object.
- `void buildDict(String[] dictionary)` Sets the data structure with an array of distinct strings `dictionary`.
- `bool search(String searchWord)` Returns `true` if you can change **exactly one character** in `searchWord` to match any string in the data structure, otherwise returns `false`.

Example 1:

```
Input
["MagicDictionary", "buildDict", "search", "search", "search", "search"]
[[], [["hello", "leetcode"]], ["hello"], ["hhllo"], ["hell"], ["leetcoded"]]
Output
[null, null, false, true, false, false]

Explanation
MagicDictionary magicDictionary = new MagicDictionary();
magicDictionary.buildDict(["hello", "leetcode"]);
magicDictionary.search("hello"); // return False
magicDictionary.search("hhllo"); // We can change the second 'h' to 'e' to match "hello" so we return True
magicDictionary.search("hell"); // return False
magicDictionary.search("leetcoded"); // return False
```

Constraints:

- `1 <= dictionary.length <= 100`
- `1 <= dictionary[i].length <= 100`
- `dictionary[i]` consists of only lower-case English letters.
- All the strings in `dictionary` are **distinct**.
- `1 <= searchWord.length <= 100`
- `searchWord` consists of only lower-case English letters.
- `buildDict` will be called only once before `search`.
- At most `100` calls will be made to `search`.