解法一

Java代码

```
 class MagicDictionary {
    TrieNode root;
    public MagicDictionary() {root = new TrieNode(); }
    public void buildDict(String[] dict) {
        for (String word : dict) {
            TrieNode temp = root;
            for (int i = 0; i < word.length();; i++) {
                char c = word.charAt(i);
                if (temp.child[c-'a'] == null) temp.child[c-
'a'] = new TrieNode();
                temp = temp.child[c-'a'];
            }
            temp.isWord = true;
        }
    }

    public boolean search(String word) {
        TrieNode presentNode = root;
        for (int i = 0; i < word.length(); i++) {// 遍历待搜索字符
            char c = word.charAt(i);
            for (int j = 0; j < 26; j++) {// 遍历当前结点所有子结点
                if ((char)
(j+'a') == c || presentNode.child[j] == null) continue;
                if (partSearch(presentNode.child[j],word,i+1)) return true;//
如果剩余字符存在当前子结点为root的Trie中，返回true
            }
            if(presentNode.child[c-'a'] == null) return false;
            presentNode = presentNode.child[c-'a'];
        }
        return false;
    }
    public boolean partSearch(TrieNode temp, String word, int index) {
        for (int i = index; i < word.length(); i++) {
            char c = word.charAt(i);
            if (temp.child[c-'a'] == null) return false;
            temp = temp.child[c-'a'];
        }
        return temp.isWord;
    }

}
```

最优解法

Java代码

```
class MagicDictionary {
    Map<Integer, ArrayList<String>> wordMap;
    public MagicDictionary() {
        wordMap = new HashMap();
    }
•    public void buildDict(String[] words) {
        for (String word : words) {
            // 判断map中是否包含key,如果不包含，则将mapFunction(key)的值加入map
中 java8新特性
            wordMap.computeIfAbsent(word.length(), x -
> new ArrayList()).add(word);
        }
    }
    public boolean search(String word) {
        if (!wordMap.containsKey(word.length())) return false;// 根据词的长度先判
断
        for (String possibleWord: wordMap.get(word.length())) {
            int wordDiff=countWordDiff(word,possibleWord);// 换多少个字母才能将
word换成possibleWord
            if (wordDiff == 1) return true;
        }
        return false;
    }
    private int countWordDiff(String word1,String word2){
        int diff = 0;
        for (int i = 0; i < word1.length(); ++i) {
            if (word1.charAt(i) != word2.charAt(i)) {
                if (++diff > 1) break;
            }
        }
        return diff;
    }
}
```

C++代码

```
class MagicDictionary {
public:
    map<int,vector<string>> f;
    /** Initialize your data structure here. */
    MagicDictionary() {
```

```
    }

    void buildDict(vector<string> dictionary) {
        for(int i=0;i<dictionary.size();i++){
            f[dictionary[i].size()].push_back(dictionary[i]);
        }
    }

    bool search(string searchWord) {
        int wordLen=searchWord.size();
        for(auto &&[len,vec]:f){
            if(wordLen==len){
                vector<int> v(wordLen);
                for(auto &vecEntry:vec){
                    int cnt=0;
                    for(int i=0;i<wordLen;i++){
                        if(searchWord[i]!=vecEntry[i]){
                            cnt++;
                        }
                    }
                    if(cnt==1) return true;
                }
            }
        }
        return false;
    }
};
```

Python代码

```
class MagicDictionary:

    def __init__(self):
        """
        Initialize your data structure here.
        """
        self.hashdict = {}
        self.dictionary = {}

    def buildDict(self, dictionary: List[str]) -> None:
        self.dictionary = dictionary
        for d in dictionary:
            for j in range(len(d)):
                if (d[:j] + "_" + d[j+1:]) not in self.hashdict.keys():
                    self.hashdict[d[:j] + "_" + d[j+1:]] = 1
                else:
                    self.hashdict[d[:j] + "_" + d[j+1:]] += 1
```

```
    def search(self, searchWord: str) -> bool:
        for j in range(len(searchWord)):
            if searchWord in self.dictionary:
                if (searchWord[:j] + "_" + searchWord[j+1:]) in
self.hashdict.keys() and self.hashdict[searchWord[:j] + "_" + searchWord[j+1:]]
> 1:
                    return True
            if searchWord not in self.dictionary:
                if (searchWord[:j] + "_" + searchWord[j+1:]) in
self.hashdict.keys() and self.hashdict[searchWord[:j] + "_" + searchWord[j+1:]]
> 0:
                    return True
        return False
```