拉勾教育
—互联网人实战大学—

字典树+哈希表：实现一个魔法字典

中等/数据结构设计、字典树、哈希表

学习目标

拉勾教育
—互联网人实战大学—

- 掌握字典树的基本概念
- 掌握字典树的构造、遍历的方法
- 掌握字典树的思想的应用

题目描述

实现一个带有buildDict，以及search方法的魔法字典。

- 对于buidDict，你将被给定一串不重复的单词来构建一个字典
- 对于search，给定一个单词，请判断是否能将该单词中一个字母换成另一个字母，使得形成的新单词存在于你构建的字典中。

```
Input: buildDict(["hello", "leetcode"]), Output: Null
Input: search("hello"), Output: False
Input: search("hhllo"), Output: True
Input: search("hell"), Output: False
Input: search("leetcoded"), Output: False
```

一. Comprehend 理解题意

本题要求实现的魔法字典并不是常用的、学习过的数据结构

在面试中常常会要求给定功能定义新的数据结构，或改变已有的数据结构

这种题目一般高起点、低落点，需要大家仔细分析给定的功能

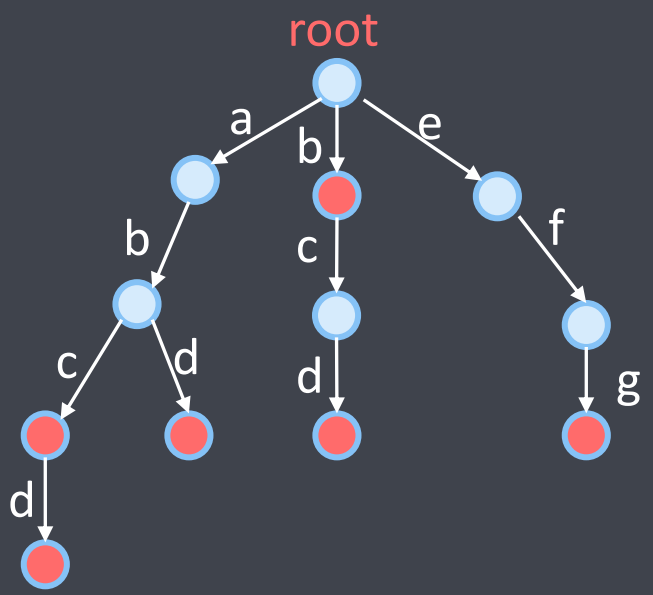
一. Comprehend 理解题意

拉勾教育
—互联网人实战大学—

字典树：

1. 又叫做trie树或前缀树。是一种有序的、用于统计、排序和存储字符串的数据结构。
2. 关键字不直接保存在结点中，而是由结点在树中的位置决定，每个结点代表一个字符
3. 根结点代表空字符串，第一层孩子结点到某个标记的结点代表了存储的字符串
4. 一个结点及其所有子孙结点都有相同的前缀，也就是这个结点对应的字符串
5. 只有叶子结点和某些被标记的内部结点，才存储了字符串
6. 字典树最大优点是利用字符串的公共前缀来减少存储空间和查询时间，从而最大限度的减少无谓的字符串比较，是非常高效的字符串查找结构，插入和查找的时间复杂度都可以达到O(1)

字符串：abc,abcd,abd,b,bcd,efg



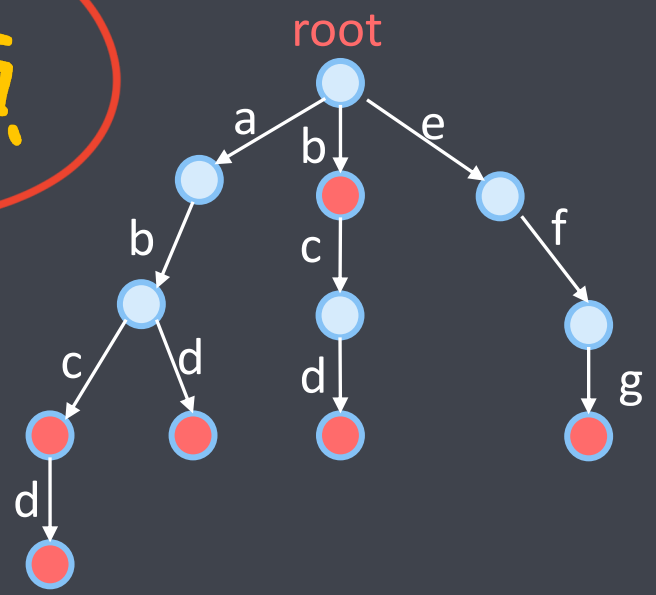
请同学们根据上面的要求思考该如何定义字典树的结构

一. Comprehend 理解题意

拉勾教育
—互联网人实战大学—

字典树的定义


```
public class TrieNode {
    private final int SIZE = 26;// 假设字典的元素都是小写字母
    TrieNode[] child;// 存储子结点
    boolean isWord;// 标记当前元素是否是单词的结尾
    public TrieNode() {
        isWord = false;// 初始化为false
        child = new TrieNode[SIZE];// 子结点初始化为空
    }
}
```



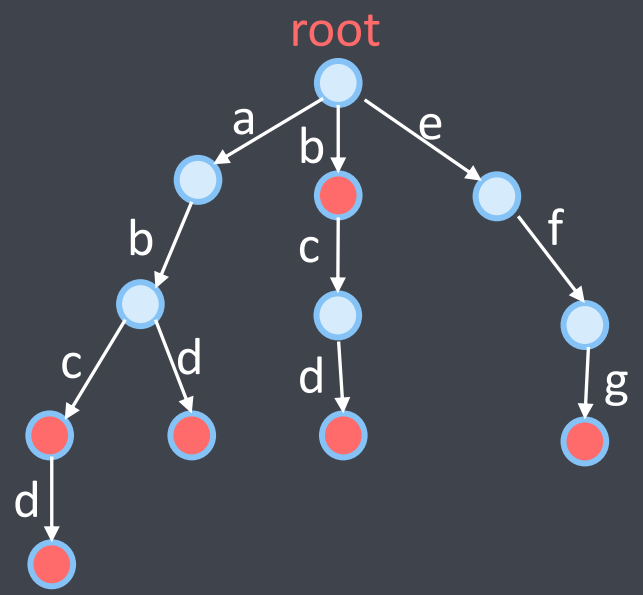
一. Comprehend 理解题意

拉勾教育
—互联网人实战大学—

字典树的前序遍历


```
public class TrieNode {
    private final int SIZE = 26;// 假设字典的元素都是小写字母
    TrieNode[] child;// 存储子结点
    boolean isWord;// 标记当前元素是否是单词的结尾
    public TrieNode() {
        isWord = false;// 初始化为false
        child = new TrieNode[SIZE];// 子结点初始化为空
    }
    public void preOrder(TrieNode root) {
        for (int i = 0; i < SIZE; i++) {// 访问根结点
            if (root.child[i] != null) {
                System.out.print(i + 'a');
                if (root.child[i].isWord) {
                    System.out.print("(end)");
                }
                System.out.print("\n");
                preOrder(root.child[i]);// 依次访问子树
            }

        }
    }
}
```



一. Comprehend 理解题意

拉勾教育
—互联网人实战大学—

字典树单词的插入

使用presentNode指向root

逐个遍历待插入字符串的字符：

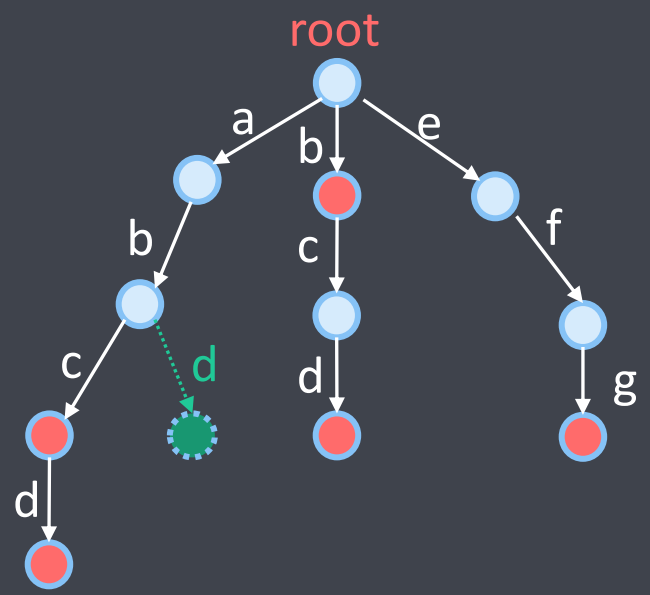
计算下标wordPos=当前遍历的字符 – ’ a’

如果presentNode指向的结点的第wordPos结点不存在：

创建一个新结点

presentNode指向该结点的第wordPos个结点

将presentNode指向结点标记为isWord=true



插入字符abd

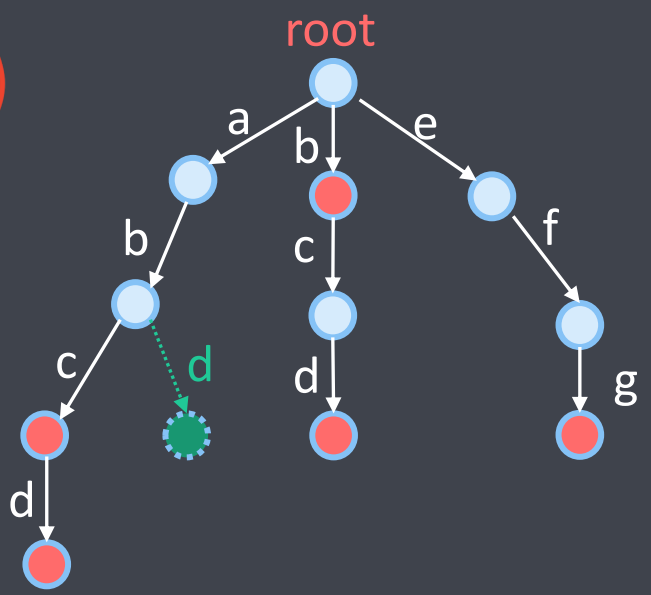
请大家根据算法流程实现代码！

一. Comprehend 理解题意

拉勾教育
—互联网人实战大学—

字典树单词的插入


```
public class TrieNode {
    private final int SIZE = 26;// 假设字典的元素都是小写字母
    TrieNode[] child;// 存储子结点
    boolean isWord;// 标记当前元素是否是单词的结尾
    public TrieNode() {
        isWord = false;// 初始化为false
        child = new TrieNode[SIZE];// 子结点初始化为空
    }
    public void insert(TrieNode root, String word) {
        char[] wordChar = word.toCharArray();
        TrieNode presentNode = root;
        for (int i = 0; i < wordChar.length; i++) {// 逐个遍历
            int wordPos = wordChar[i] - 'a'; // 计算坐标
            if (presentNode.child[wordPos] == null) {
                presentNode.child[wordPos] = new TrieNode(); // 不存在，则创建新结点
            }
            presentNode = presentNode.child[wordPos];// 存在，则当前字符已经存在，移动指针
        }
        presentNode.isWord = true;// 标记单词末尾
    }
}
```



插入字符abd

一. Comprehend 理解题意

拉勾教育
—互联网人实战大学—

字典树的单词的搜索

使用presentNode指向root

逐个遍历待插入字符串的字符：

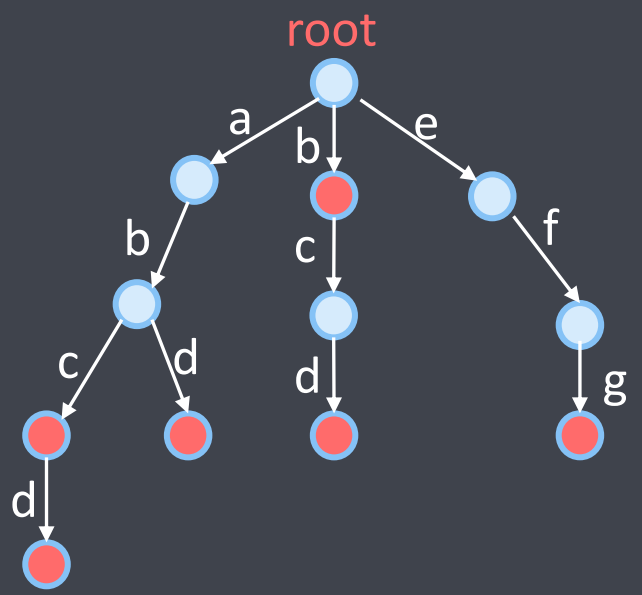
计算下标wordPos=当前遍历的字符 – ’ a’

如果presentNode指向的结点的第wordPos结点不存在：

搜索失败

presentNode指向该结点的第wordPos个结点

返回presentNode指向结点的标记



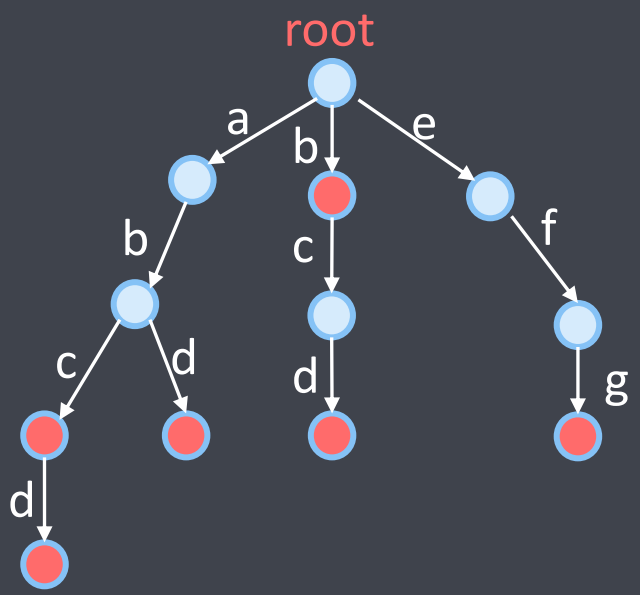
搜索 abd ex

一. Comprehend 理解题意

拉勾教育
—互联网人实战大学—

字典树的单词的搜索

```
public class TrieNode {
    private final int SIZE = 26;// 假设字典的元素都是小写字母
    TrieNode[] child;// 存储子结点
    boolean isWord;// 标记当前元素是否是单词的结尾
    public TrieNode() {
        isWord = false;// 初始化为false
        child = new TrieNode[SIZE];// 子结点初始化为空
    }
    public boolean search(TrieNode root, String word) {
        char[] wordChar = word.toCharArray();
        TrieNode presentNode = root;
        for (int i = 0; i < wordChar.length; i++) {// 逐个遍历待搜索字符
            int wordPos = wordChar[i] - 'a';// 计算坐标
            if (presentNode.child[wordPos] == null) {
                return false;// 如果当前字符在Trie中不存在，搜索失败
            }
            presentNode = presentNode.child[wordPos];// 继续向下搜索
        }
        return presentNode.isWord;// 返回当前结点的标记
    }
}
```



搜索 abd ex

一. Comprehend 理解题意

拉勾教育
—互联网人实战大学—

题目细节

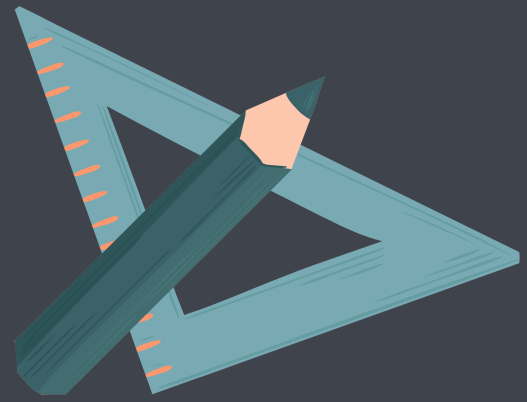
本题要求实现带有build + search 方法的魔法字典

- 选择何种数据结构来存储字典中的单词?
- 在构建字典时候添加什么额外信息有助于搜索？

二. Choose 数据结构及算法思维选择

方案一：基于Trie

- 用字典树来存储单词
- 数据结构：字典树
- 算法思维：基于Trie的搜索

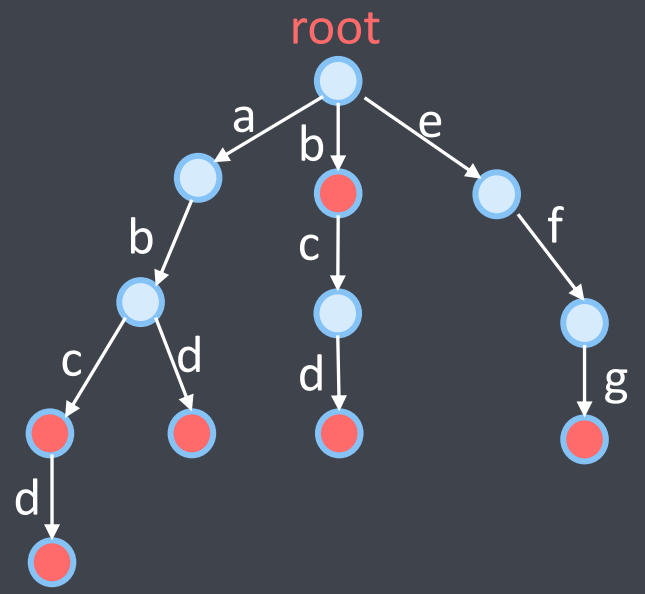


三. Code 基本解法及编码实现

解法一： 基于Trie

输入：[abc,abcd,abd,b,bcd,efg]

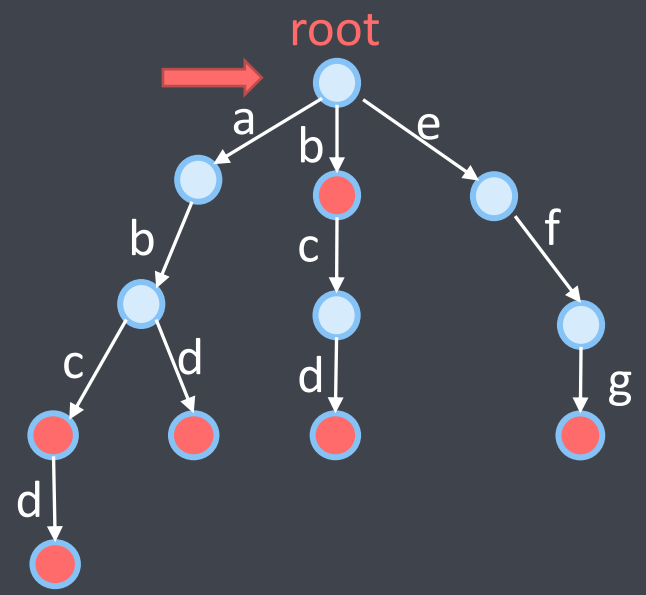
在build方法中，建立一棵字典树



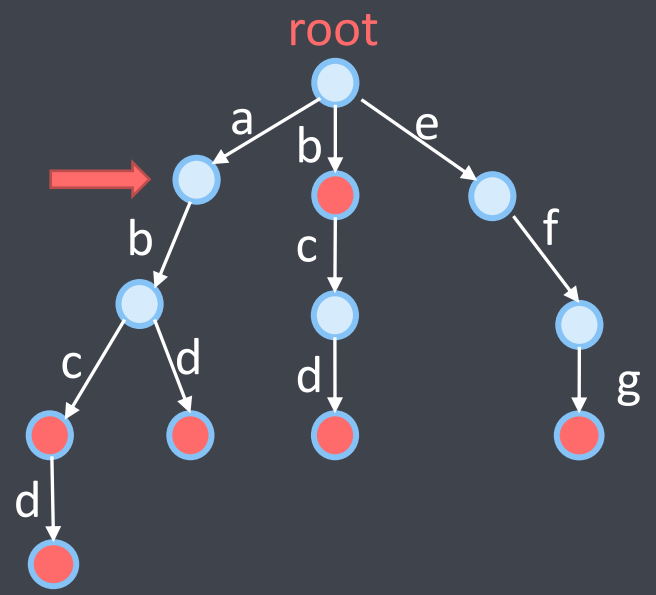
三. Code 基本解法及编码实现

解法一： 基于Trie

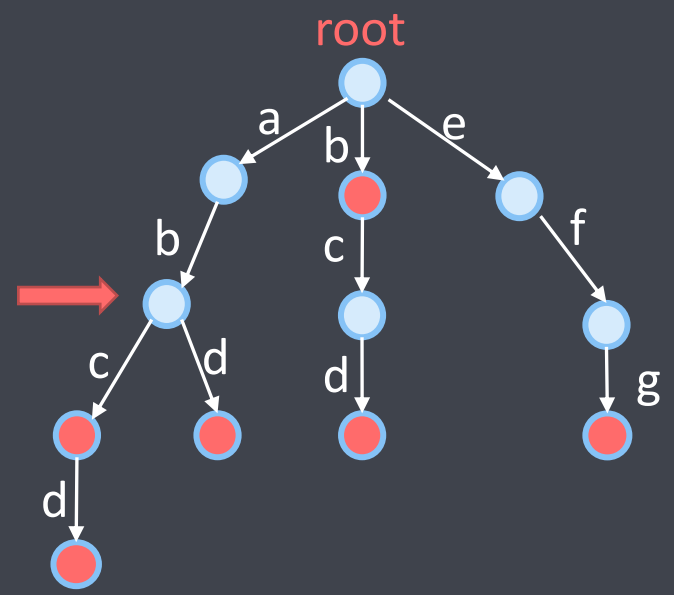
search(”ascd”)



step 1 字符a存在字典树中



step 2 字符s不存



step 3 判断后面字符串cd是否存在

三. Code 基本解法及编码实现

解法一： 基于Trie

使用presentNode指向root

遍历待插入字符串的字符：

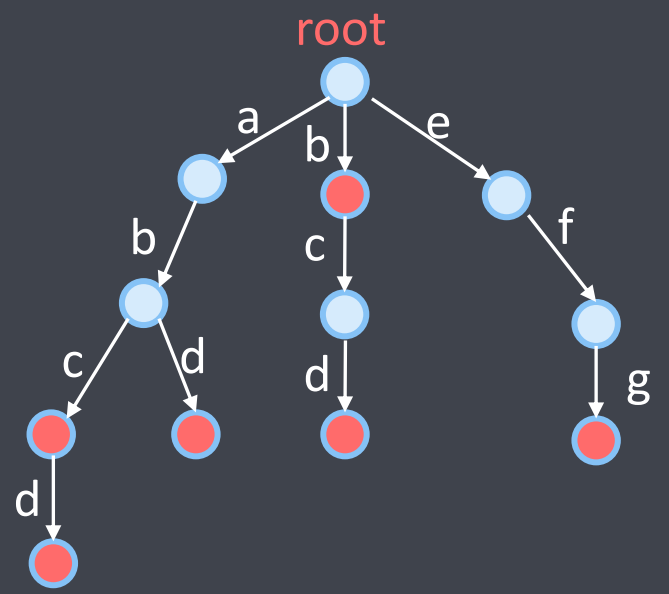
遍历 presentNode指向结点的子结点：

如果当前子结点的字符与当前字符相等：

向下遍历

如果当前子结点的字符与当前字符不相等：

判断剩下的字符串是否存在于当前子结点为root 的Trie中



search(”ascd”)

请大家根据算法流程实现代码！

三. Code 基本解法及编码实现

解法一：基于Trie参考代码

```
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;
    }
}
```

三. Code 基本解法及编码实现

拉勾教育
—互联网人实战大学—

解法一：基于Trie参考代码

```
class MagicDictionary {
    TrieNode root;
    public MagicDictionary() {root = new TrieNode(); }
    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;
    }
}
```

执行结果：通过 显示详情 >

执行用时：43 ms，在所有 Java 提交中击败了 26.63% 的用户

内存消耗：39.2 MB，在所有 Java 提交中击败了 53.57% 的用户

三. Code 基本解法及编码实现

解法一： 基于Trie复杂度分析

时间复杂度：

- s为输入字符串字符的个数
- k为待搜索单词长度
- build方法需要O(s)
- search 最差需要O（26*s*k）=O（sk）

空间复杂度：

- Q为去除公共前缀后字符个数，L<=Q<=S
- L为最长字符串的长度
- 字典树存储空间复杂度为O（Q）

四. Consider 思考更优解

解法一中使用了字典树来存储字符串

题目要求的搜索与字典树的搜索方法有差别

无法发挥字典树的长处

思考：是否可以选择别的数据结构来存储字符串加速搜索过程？

四. Consider 思考更优解

定义：换掉单词中的一个字母后得到的新词叫做广义邻居

比如 apple => [bpple,axple,apyle,appse,appld,...]

广义邻居所有词与原单词的长度相等

在建立魔法字典时候，如果将相同长度的词语为一组，

在搜索时根据词长快速缩小搜索范围

选择HashMap<integer,ArrayList<String>> 建立 词长 → 词组

五. Code 最优解思路及编码实现

最优解： 基于HashMap

buildDict

根据单词长度将输入词组进行分组即可

五. Code 最优解思路及编码实现

最优解： 基于HashMap

search

计算目标词的长度wordLen

如果wordLen在字典中不存在词组

返回false

如果wordLen在字典中存在词组

遍历词组中每一个词

计算其于目标词之间的不一样字符数wordDiff

if wordDiff == 1 则查找成功

五. Code 最优解思路及编码实现

编码实现

```
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;
    }
}
```

执行结果： 通过 显示详情 >

执行用时： 44 ms，在所有 Java 提交中击败了 26.32% 的用户

内存消耗： 39.3 MB，在所有 Java 提交中击败了 48.22% 的用户

五. Code 最优解思路及编码实现

复杂度分析

时间复杂度：

- S为所有输入单词的字符数
- K为目标词的长度
- buidDict 遍历词组计算词长需要O(S)
- search时候如果字典所有单词与目标词长度一致最差需要O(kS)
- Search最好达到O(1)

空间复杂度：O(S)

- S为所有输入单词的字符数

六. Change 变形延伸

拉勾教育
—互联网人实战大学—

题目变形

- （练习）词典中最长的单词

延伸扩展

- 字典树最大特点是利用单词的公共前缀节省存储空间和提高搜索效率

本题来源

- Leetcode 676 https://leetcode-cn.com/problems/implement-magic-dictionary/

总结

拉勾教育
—互联网人实战大学—

- 掌握字典树的基本概念
- 掌握字典树的构造、遍历的方法
- 掌握字典树的思想的应用

课后练习

拉勾教育
—互联网人实战大学—

1. 实现Trie(Leetcode208/中等)
2. 添加与搜索单词（Leetcode 211 /中等）
3. 单词替换（Leetcode 648 /中等）
4. 前缀和后缀搜索(Leetcode 755 /困难)

课后练习

1. 实现Trie(Leetcode208/中等)

   提示：实现一个 Trie (前缀树)，包含 insert, search, 和 startsWith 这三个操作。

```
Trie trie = new Trie();

trie.insert("apple");
trie.search("apple");   // 返回 true
trie.search("app");     // 返回 false
trie.startsWith("app"); // 返回 true
trie.insert("app");
trie.search("app");     // 返回 true
```

课后练习

2. 添加与搜索单词（Leetcode 211 /中等）

提示：如果数据结构中有任何与word匹配的字符串，则bool search（word）返回true，否则返回false。 单词可能包含点“.” 点可以与任何字母匹配的地方。

请你设计一个数据结构，支持添加新单词 和 查找字符串是否与任何先前添加的字符串匹配

输入:
["WordDictionary","addWord","addWord","addWord","search","search","
[[],["bad"],["dad"],["mad"],["pad"],["bad"],[".ad"],["b.."]]
输出:
[null,null,null,null,false,true,true,true]

解释:

```
WordDictionary wordDictionary = new WordDictionary();
wordDictionary.addWord("bad");
wordDictionary.addWord("dad");
wordDictionary.addWord("mad");
wordDictionary.search("pad"); // return False
wordDictionary.search("bad"); // return True
wordDictionary.search(".ad"); // return True
wordDictionary.search("b.."); // return True
```

课后练习

3. 单词替换（Leetcode 648 /中等）

提示：在英语中，我们有一个叫做 词根(root)的概念，它可以跟着其他一些词组成另一个较长的单词——我们称这个词为 继承词(successor)。例如，词根an，跟随着单词 other(其他)，可以形成新的单词 another(另一个)。

现在，给定一个由许多词根组成的词典和一个句子。你需要将句子中的所有继承词用词根替换掉。如果继承词有许多可以形成它的词根，则用最短的词根替换它。

你需要输出替换之后的句子。

```
输入: dictionary = ["cat","bat","rat"], sentence = "the cattle was rattled by the battery"
输出: "the cat was rat by the bat"
```

```
输入: dictionary = ["a","b","c"], sentence = "aadsfasf absbs bbab cadsfafs"
输出: "a a b c"
```

课后练习

4. 前缀和后缀搜索(Leetcode 755 /困难)

提示：在英语给定多个 words，words[i] 的权重为 i 。

设计一个类 WordFilter 实现函数WordFilter.f(String prefix, String suffix)。这个函数将返回具有前缀 prefix 和后缀suffix 的词的最大权重。如果没有这样的词，返回 -1。

```
输入:
WordFilter(["apple"])
WordFilter.f("a", "e") // 返回 0
WordFilter.f("b", "") // 返回 -1
```

拉勾教育

—互联网人实战大学—

下载「拉勾教育App」
获取更多内容