# 题目

在英语中，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，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。

需要输出替换之后的句子。

示例 1：

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

输出："the cat was rat by the bat"

示例 2：

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

输出："a a b c"

示例 3：

输入：dictionary = ["a", "aa", "aaa", "aaaa"], sentence = "a aa a aaaa aaa aaa aaa aaaaaa bbb baba ababa"

输出："a a a a a a a a bbb baba a"

示例 4：

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

输出："the cat was rat by the bat"

示例 5：

输入：dictionary = ["ac","ab"], sentence = "it is abnormal that this solution is accepted"

输出："it is ab that this solution is ac"

提示：

1 <= dictionary.length <= 1000

1 <= dictionary[i].length <= 100

dictionary[i] 仅由小写字母组成。

1 <= sentence.length <= 10^6

sentence 仅由小写字母和空格组成。

sentence 中单词的总量在范围 [1, 1000] 内。

sentence 中每个单词的长度在范围 [1, 1000] 内。

sentence 中单词之间由一个空格隔开。

sentence 没有前导或尾随空格。

# 分析

// 构造前缀树节点

class Trie {

private:

bool isWord;

vector<Trie\*> children;

public:

Trie () : isWord(false), children(26, nullptr) {}

void insert(const string& str) {

Trie\* node = this;

for (auto& ch : str) {

if (node->children[ch - 'a'] == nullptr) {

node->children[ch - 'a'] = new Trie();

}

node = node->children[ch - 'a'];

}

node->isWord = true;

}

int countPreFixLen(const string& str) {

Trie\* node = this;

int len = 0;

for (auto& ch : str) {

if (node->children[ch - 'a'] == nullptr) {

return 0;

}

node = node->children[ch - 'a'];

len++;

if (node->isWord) {

return len;

}

}

return 0;

}

};

class Solution {

public:

string replaceWords(vector<string>& dictionary, string sentence) {

Trie\* root = new Trie();

for (string& word : dictionary) {

root->insert(word);

}

// 分割

vector<string> words{""};

for (auto& ch : sentence) {

if (ch != ' ') {

words.back().push\_back(ch);

}

else {

words.push\_back("");

}

}

// 处理

vector<string> ret;

for (auto& word : words) {

int len = root->countPreFixLen(word);

if (len == 0) {

ret.emplace\_back(word);

}

else {

ret.push\_back(word.substr(0, len));

}

}

// 拼接

string ans{""};

for (auto& word : ret) {

ans += word;

ans += " ";

}

ans.pop\_back();

return ans;

}

};