# 题目

给你一个字符串 s 和一个字符串列表 wordDict 作为字典。如果可以利用字典中出现的一个或多个单词拼接出 s 则返回 true。

注意：不要求字典中出现的单词全部都使用，并且字典中的单词可以重复使用。

示例 1：

输入: s = "leetcode", wordDict = ["leet", "code"]

输出: true

解释: 返回 true 因为 "leetcode" 可以由 "leet" 和 "code" 拼接成。

示例 2：

输入: s = "applepenapple", wordDict = ["apple", "pen"]

输出: true

解释: 返回 true 因为 "applepenapple" 可以由 "apple" "pen" "apple" 拼接成。

注意，你可以重复使用字典中的单词。

示例 3：

输入: s = "catsandog", wordDict = ["cats", "dog", "sand", "and", "cat"]

输出: false

提示：

1 <= s.length <= 300

1 <= wordDict.length <= 1000

1 <= wordDict[i].length <= 20

s 和 wordDict[i] 仅由小写英文字母组成

wordDict 中的所有字符串 互不相同

# 分析

## 方法一：动态规划

思路：

可以使用动态规划的方法来解决。我们可以定义一个布尔类型的数组 dp，其中 dp[i] 表示字符串 s 的前 i 个字符是否可以被字典中的单词拼接而成。

首先，我们需要初始化 dp 数组。显然，空字符串可以被视为由空字符拼接而成，所以 dp[0] 应该为 true。

然后，我们遍历字符串 s 的每个位置 i，从 1 到 s.length()。对于每个位置 i，我们再遍历字典 wordDict 中的每个单词 word，检查是否可以将 word 与 s 的前 i - word.length() 个字符拼接成 s 的前 i 个字符。如果可以，那么 dp[i] 就应该为 true。

最后，返回 dp[s.length()]，即表示整个字符串 s 是否可以被字典中的单词拼接而成。

代码：

class Solution {

public:

bool wordBreak(string s, vector<string>& wordDict) {

int n = s.size();

vector<bool> dp(n + 1, false);

dp[0] = true; // 空字符串可以被视为由空字符拼接而成

// 将字典中的单词存入哈希集合中，方便快速查找

unordered\_set<string> wordSet(wordDict.begin(), wordDict.end());

for (int i = 1; i <= n; i++) {

for (int j = 0; j < i; j++) {

// 检查是否可以将 s[j:i-1] 与 s[0:j-1] 拼接成 s[0:i-1]

string word = s.substr(j, i - j);

if (dp[j] && wordSet.count(word)) {

dp[i] = true;

break;

}

}

}

return dp[n];

}

};

这段代码首先定义了一个布尔类型的数组 dp，并初始化了 dp[0] 为 true。然后，它将字典 wordDict 中的单词存入一个哈希集合 wordSet 中，以便快速查找。接下来，它使用两个嵌套的循环来遍历字符串 s 的每个位置 i 和每个可能的分割点 j。如果 s[j:i-1] 是一个字典中的单词，并且 s[0:j-1] 可以被字典中的单词拼接而成（即 dp[j] 为 true），那么 s[0:i-1] 也可以被字典中的单词拼接而成，因此将 dp[i] 设为 true。最后，返回 dp[n]，即整个字符串 s 是否可以被字典中的单词拼接而成。