1. 单词拆分

给定一个非空字符串 s 和一个包含非空单词列表的字典 wordDict，判定 s 是否可以被空格拆分为一个或多个在字典中出现的单词。

说明：

拆分时可以重复使用字典中的单词。

你可以假设字典中没有重复的单词。

示例 1：

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

输出: true

解释: 返回 true 因为 "leetcode" 可以被拆分成 "leet code"。

示例 2：

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

输出: true

解释: 返回 true 因为 "applepenapple" 可以被拆分成 "apple pen apple"。

  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

来源：力扣（LeetCode）

链接：https://leetcode-cn.com/problems/word-break

著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。

解法一：回溯带剪枝(即把已经搜索过的下标记忆起来，防止重复搜索。比如当前以下标i为开头，遍历过了s.substr(i,j-i)的所有字符串(j=[i:s.length])均失败，那么标记memo[i] = false

/\*\*

\* @param {string} s

\* @param {string[]} wordDict

\* @return {boolean}

\*/

var wordBreak = function(s, wordDict) {

let hash = new Set(wordDict),memo = [];

let dfs = function(index)

{

if(index>=s.length)

return true;

if (memo[index] != null) {

return memo[index];

}

for (let end = index + 1; end <= s.length; end++) {

if (hash.has(s.substr(index, end-index)) && dfs(end)) {

return memo[index] = true;

}

}

return memo[index] = false;

}

return dfs(0);

};

解法二：宽度优先搜索

var wordBreak = function(s, wordDict) {

let len = s.length,visited = Array(len).fill(false);

let dic = new Set(wordDict),root = [];

let start = 0;

root.push(0);

while(root.length!==0)

{

start = root.shift();

if(!visited[start])

{

visited[start]=true;

for(let i=start;i<=len;i++)

{

if(dic.has(s.substr(start,i-start)))

{

if(i===len)

return true;

root.push(i);

}

}

}

}

return false;

};

解法三：动态规划

Dp[i]表示s的子串s[0:i]是否可以拆分

var wordBreak = function(s, wordDict) {

let len = s.length,splitWord = "",indice = 0;

let dic = new Set(wordDict);

let dp = Array(len+1).fill(false);

dp[0] = true;

let search = function()

{

for(let i=1;i<=len;i++)

for(let j=0;j<i;j++)

{

if(dp[j] && dic.has(s.substr(j,i-j)))

{

dp[i] = true;

}

}

}

search(0);

return dp[len];

};