题目描述（困难难度）

140. Word Break II

Description | Hints | Submissions | Discuss | Solution

Pick One

Given a **non-empty** string *s* and a dictionary *wordDict* containing a list of **non-empty** words, add spaces in *s* to construct a sentence where each word is a valid dictionary word. Return all such possible sentences.

Note:

- The same word in the dictionary may be reused multiple times in the segmentation.
- You may assume the dictionary does not contain duplicate words.

Example 1:

```
Input:
s = "catsanddog"
wordDict = ["cat", "cats", "and", "sand", "dog"]
Output:
[
  "cats and dog",
  "cat sand dog"
]
```

Example 2:

```
Input:
s = "pineapplepenapple"
wordDict = ["apple", "pen", "applepen", "pine", "pineapple"]
Output:
[
  "pine apple pen apple",
  "pineapple pen apple",
  "pine applepen apple"
]
Explanation: Note that you are allowed to reuse a dictionary word.
```

Example 3:

```
Input:
s = "catsandog"
wordDict = ["cats", "dog", "sand", "and", "cat"]
Output:
[]
```

139 题 的升级版。给一个字符串，和一些单词，找出由这些单词组成该字符串的所有可能，每个单词可以用多次，也可以不用。

完全按照 139 题 的思路做了，大家可以先过去看一下。

解法一 动态规划

先考虑 `139 题` 最后一个解法，动态规划，看起来也比较简单粗暴。

> 用 `dp[i]` 表示字符串 `s[0,i)` 能否由 `wordDict` 构成。
>
> ```
> public boolean wordBreak(String s, List<String> wordDict) {
>     HashSet<String> set = new HashSet<>();
>     for (int i = 0; i < wordDict.size(); i++) {
>         set.add(wordDict.get(i));
>     }
>     boolean[] dp = new boolean[s.length() + 1];
>     dp[0] = true;
>     for (int i = 1; i <= s.length(); i++) {
>         for (int j = 0; j < i; j++) {
>             dp[i] = dp[j] && set.contains(s.substring(j, i));
>             if (dp[i]) {
>                 break;
>             }
>         }
>     }
>     return dp[s.length()];
> }
> ```

这里修改的话，我们只需要用 `dp[i]` 表示字符串 `s[0,i)` 由 `wordDict` 构成的所有情况。

总体思想还是和之前一样的。

```
X X X X X X
^     ^   ^
0     j   i
先判断 j 到 i 的字符串在没在 wordDict 中
然后把 0 到 j 的字符串由 wordDict 构成所有情况后边加空格再加上 j 到 i 的字符串即可
```

结合上边的思想，然后把它放到循环中，考虑所有情况即可。

```
public List<String> wordBreak(String s, List<String> wordDict) {
    HashSet<String> set = new HashSet<>();
    for (int i = 0; i < wordDict.size(); i++) {
        set.add(wordDict.get(i));
    }
    List<List<String>> dp = new ArrayList<>();
    List<String> temp = new ArrayList<>();
    //空串的情况
    temp.add("");
    dp.add(temp);
    for (int i = 1; i <= s.length(); i++) {
        temp = new ArrayList<>();
        for (int j = 0; j < i; j++) {
            if (set.contains(s.substring(j, i))) {
                //得到前半部分的所有情况然后和当前单词相加
                for (int k = 0; k < dp.get(j).size(); k++) {
                    String t = dp.get(j).get(k);
                    //空串的时候不加空格，也就是 j = 0 的时候
                    if (t.equals("")) {
                        temp.add(s.substring(j, i));
                    } else {
                        temp.add(t + " " + s.substring(j, i));
                    }

                }

            }
        }
        dp.add(temp);
    }
    return dp.get(s.length());
}
```

遗憾的是，熟悉的问题又来了。

Submission Detail

31 / 39 test cases passed. Status: Time Limit Exceeded

Submitted: 3 days, 9 hours ago

```
Last executed input: "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"
["a","aa","aaa","aaaa","aaaaa","aaaaaa","aaaaaaa","aaaaaaaa","aaaaaaaaa","aaaaaaaaaa"]
```

由于 `s` 中的 `b` 字母在 `wordDict` 中并没有出现，所以其实我们并不需要做那么多循环，直接返回空列表即可。

和之前一样，所以我们可以先遍历一遍 `s` 和 `wordDict` ，从而确定 `s` 中的字符是否在 `wordDict` 中存在，如果不存在可以提前返回空列表。

```
public List<String> wordBreak(String s, List<String> wordDict) {
    //提前进行一次判断
    HashSet<Character> set2 = new HashSet<>();
    for (int i = 0; i < wordDict.size(); i++) {
        String t = wordDict.get(i);
        for (int j = 0; j < t.length(); j++) {
            set2.add(t.charAt(j));
        }
    }
    for (int i = 0; i < s.length(); i++) {
        if (!set2.contains(s.charAt(i))) {
            return new ArrayList<>();
        }
    }

    HashSet<String> set = new HashSet<>();
    for (int i = 0; i < wordDict.size(); i++) {
        set.add(wordDict.get(i));
    }
    List<List<String>> dp = new ArrayList<>();
    List<String> temp = new ArrayList<>();
    temp.add("");
    dp.add(temp);
    for (int i = 1; i <= s.length(); i++) {
        temp = new ArrayList<>();
        for (int j = 0; j < i; j++) {
            if (set.contains(s.substring(j, i))) {
                for (int k = 0; k < dp.get(j).size(); k++) {
                    String t = dp.get(j).get(k);
                    if (t.equals("")) {
                        temp.add(s.substring(j, i));
                    } else {
                        temp.add(t + " " + s.substring(j, i));
                    }

                }

            }
        }
        dp.add(temp);
    }
    return dp.get(s.length());
}
```

遗憾的是，刚刚那个例子通过了，又出现了新的问题。

Submission Detail

32 / 39 test cases passed. Status: Time Limit Exceeded

Submitted: 3 days, 9 hours ago

```
Last executed input: "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabaabaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"
["aa","aaa","aaaa","aaaaa","aaaaaa","aaaaaaa","aaaaaaaa","aaaaaaaaa","aaaaaaaaaa","ba"]
```

由于 `wordDict` 有 `b` 字母了，所以并没有提前结束，而是进了 `for` 循环。

再优化也想不到方法了，是我们的算法出问题了。因为 `139` 题中找到一个解以后就 `break` 了，而这里我们要考虑所有子串，所有的解，极端情况下时间复杂度达到了 `O(n³)` 。还有一点致命的是，我们之前求的解最后可能并不需要。举个例子。

```
"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabad"
["aa","aaa","aaaa","aaaaa","aaaaaa","aaaaaaa","aaaaaaaa","aaaaaaaaa","aaaaaaaaaa","b","ba","de"]

我们之前求了 "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa" 的所有组成可能，但由于剩余字符串 "

又比如，我们之前求了 "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaab" 的所有组成可能，但由于剩余
```

针对这个问题，我们可以优化一下，也就是下边的解法二

解法二

我们直接用递归的方法，先判断当前字符串在不在 `wordDict` 中，如果在的话就递归的去求剩余字符串的所有组成可能。此外，吸取之前的教训，直接使用 `memoization` 技术，将递归过程中求出来的解缓存起来，便于之后直接用。

```
public List<String> wordBreak(String s, List<String> wordDict) {
    HashSet<String> set = new HashSet<>();
    for (int i = 0; i < wordDict.size(); i++) {
        set.add(wordDict.get(i));
    }
    return wordBreakHelper(s, set, new HashMap<String, List<String>>());
}

private List<String> wordBreakHelper(String s, HashSet<String> set, HashMap<String, List<String>> map) {
    if (s.length() == 0) {
        return new ArrayList<>();
    }
    if (map.containsKey(s)) {
        return map.get(s);
    }
    List<String> res = new ArrayList<>();
    for (int j = 0; j < s.length(); j++) {
        //判断当前字符串是否存在
        if (set.contains(s.substring(j, s.length()))) {
            //空串的情况，直接加入
            if (j == 0) {
                res.add(s.substring(j, s.length()));
            } else {
                //递归得到剩余字符串的所有组成可能，然后和当前字符串分别用空格连起来加到结果中
                List<String> temp = wordBreakHelper(s.substring(0, j), set, map);
                for (int k = 0; k < temp.size(); k++) {
                    String t = temp.get(k);
                    res.add(t + " " + s.substring(j, s.length()));
                }
            }

        }
    }
    //缓存结果
    map.put(s, res);
    return res;
}
```

总

按理说其实可以直接就想到解法二，但受之前写的题的影响，这种分治的问题，都最终能转成动态规划，所以先写了动态规划的思路，想直接一步到位，没想到反而遇到了问题，很有意思，哈哈。原因就是你求子问题的代价很大，而动态规划就是要求所有的子问题。而解决最终问题的时候，一些子问题其实是没有必要的。