LeetCode 139

https://leetcode.com/problems/word-break/description/

Yifeng Zeng

Description

139. Word Break

Idea Report

We can devide the input string s into two halves, the left half and right half. Each time we actually check if left half and right half is in the dict. Because we are spliting s into two halves, and we want to try each pairs, we need a pointer i to split it. Either i from 0 to len or len to 0, one side of the substring is fixed. So we can store left half (or right half) into a boolean[] array. Each time we check is left half in boolean array is true, and right half is in the dict. If both true then this string s is breakable.

So our boolean[] array f is our dynamic function. f[i] means substring[0, i) is breakable. Then we need to find a j between 0 to i so that f[j] is true and substring[j,i) is in dict so that f[i] is breakable into [0,j) and [j,i). The base case is f[0] = true because an empty string "" is considered breakable.

Code

```
public class Solution {

    public boolean wordBreak(String s, List<String> dict) {
        boolean[] f = new boolean[s.length() + 1];
        f[0] = true;

        for (int i = 1; i <= s.length(); i++) {
            for (int j = 0; j < i; j++) {
                String substring = s.substring(j, i);
                if (f[j] && dict.contains(substring)) {
                    f[i] = true;
                }
            }
        }

        return f[s.length()];
    }
}
```

Summary

- Consider the base case
- Consider how a large problem can split into sub problems
- Consider how one state can transfer to another state

Follow up

Primitive DFS without memoization.
TLE case:
"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaab"
["a","aa","aaa","aaaa","aaaaa","aaaaaa","aaaaaaa","aaaaaaaa","aaaaaaaaa","aaaaaaaaaa"]

```
public class Solution {
    // TLE
    public boolean wordBreak(String s, List<String> dict) {
        return helper(s, dict);
    }

    private boolean helper(String s, List<String> dict) {
        if (s.equals("") || dict.contains(s)) {
            return true;
        }

        boolean res = false;
        for (int i = 0; i < s.length(); i++) {
            String substring = s.substring(0, i);
            if (dict.contains(substring) && helper(s.substring(i), dict)) {
                return true;
            }
        }
        return res;
    }
}
```

DFS with memoization AC.

```
public boolean wordBreak(String s, List<String> dict) {
    int[] memo = new int[s.length()];
    return helper(s, new HashSet<>(dict), 0, memo);
}

private boolean helper(String s, Set<String> dict, int index, int[] memo) {
    if (s.equals("") || dict.contains(s) || memo[index] == 1) {
        return true;
    }
    if (memo[index] == -1) {
        return false;
    }

    boolean res = false;
    for (int i = 0; i < s.length(); i++) {
        String substring = s.substring(0, i);
        if (dict.contains(substring)
            && helper(s.substring(i), dict, i + 1, memo)) {
            memo[index] = 1;
            return true;
        }
    }
    memo[index] = -1;
    return res;
}
```

DFS with memoization AC with map, easier to understand

```
public boolean wordBreak(String s, List<String> dict) {
    Map<String, Boolean> map = new HashMap<>();
    return helper(s, new HashSet<>(dict), 0, map);
}

private boolean helper(String s, Set<String> dict, int index,
                       Map<String, Boolean> map) {
    if (s.equals("") || dict.contains(s)) {
        return true;
    }
    if (map.containsKey(s)) {
        return map.get(s);
    }

    boolean res = false;
    for (int i = 0; i < s.length(); i++) {
        String substring = s.substring(0, i);
        if (dict.contains(substring)
            && helper(s.substring(i), dict, i + 1, map)) {
            map.put(s, true);
            return true;
        }
    }
    map.put(s, false);
    return res;
}
```