<http://buttercola.blogspot.com/2019/05/lintcode-680-split-string.html>

Give a string, you can choose to split the string after one character or two adjacent characters, and make the string to be composed of only one character or two characters. Output all possible results.

### Example

**Example1**

Input: "123"

Output: [["1","2","3"],["12","3"],["1","23"]]

**Example2**

Input: "12345"

Output: [["1","23","45"],["12","3","45"],["12","34","5"],["1","2","3","45"],["1","2","34","5"],["1","23","4","5"],["12","3","4","5"],["1","2","3","4","5"]]

**Attempt 1: 2022-12-27**

**Solution 1: Backtracking (10 min)**

**Style 1: Similar to 0-1 knapsack**

public class Solution {

/\*

\* @param : a string to be split

\* @return: all possible split string array

\*/

public List<List<String>> splitString(String s) {

List<List<String>> result = new ArrayList<List<String>>();

helper(s, result, new ArrayList<String>(), 0);

return result;

}

private void helper(String s, List<List<String>> result, List<String> list, int startIndex) {

if(startIndex == s.length()) {

// Deep copy since list is an object and go through backtracking

result.add(new ArrayList<String>(list));

return;

}

if(startIndex > s.length()) {

return;

}

// Attempt 1 digit for current level recursion

String s1 = s.substring(startIndex, startIndex + 1);

list.add(s1);

helper(s, result, list, startIndex + 1);

list.remove(list.size() - 1);

// Attempt 2 digits for current level recursion

if(startIndex + 1 < s.length()) {

String s2 = s.substring(startIndex, startIndex + 2);

list.add(s2);

helper(s, result, list, startIndex + 2);

list.remove(list.size() - 1);

}

}

}

Time Complexity : O(N \* 2^N), 递归树是二叉树，并且深度为n，同时，把list加入result的时候要把list拷贝一份

Space Complexity : O(N)

**Style 2: For loop**

public class Solution {

/\*

\* @param : a string to be split

\* @return: all possible split string array

\*/

public List<List<String>> splitString(String s) {

List<List<String>> result = new ArrayList<List<String>>();

helper(s, result, new ArrayList<String>(), 0);

return result;

}

private void helper(String s, List<List<String>> result, List<String> list, int startIndex) {

if(startIndex == s.length()) {

// Deep copy since list is an object and go through backtracking

result.add(new ArrayList<String>(list));

return;

}

if(startIndex > s.length()) {

return;

}

for(int i = startIndex + 1; i <= startIndex + 2 && i <= s.length(); i++) {

String substr = s.substring(startIndex, i);

list.add(substr);

helper(s, result, list, i);

list.remove(list.size() - 1);

}

}

}

Time Complexity : O(N \* 2^N), 递归树是二叉树，并且深度为n，同时，把list加入result的时候要把list拷贝一份

Space Complexity : O(N)