131. Palindrome Partitioning

Given a string *s*, partition *s* such that every substring of the partition is a palindrome.

Return all possible palindrome partitioning of *s*.

For example, given *s* = "aab",

Return

```
[
  ["aa","b"],
  ["a","a","b"]
]
```

C++:

```
class Solution {
private:
    void help( vector<vector<string>>& palindromes, vector<string>& buf, string& s, int idx ) {
        if( idx > s.length() - 1 ) palindromes.push_back(buf);
        else {
            for( int i = idx; i<s.length(); i++ ) {
                int j = idx, k = i;
                while( j<k && s[j] == s[k]) { j++; k--; };
                if( j>=k ) {
                    buf.push_back(s.substr(idx, i-idx+1));
                    help(palindromes, buf, s, i+1);
                    buf.pop_back();
                }
            }
        }
    }
public:
    vector<vector<string>> partition(string s) {
        vector<string> buf;
        vector<vector<string>> ans;
        help(ans, buf, s, 0);
        return ans;
    }
};
```

JAVA:

```
/*
    s[i...j] (isPalin[i][j])是不是palindromic? 通过(s[i]==s[j] && dp[i+1][j-1])来判断.

       a  a  b
    a  y  y  n
    a     y  n
    b        y

    剩下的事情就是做一个dfs.
*/
public class Solution {
    public List<List<String>> partition(String s) {
        List<List<String>> ans = new ArrayList<>();
        if (s==null || s.length()==0) { return ans; }
        int length = s.length();
        boolean[][] isPalin = new boolean[length][length];
        for (int i=0, count=length, cont; i<length; ++i, --count) {
            for (int j=0; j<count; ++j) {
```

```
                isPalin[j][i+j] = s.charAt(j)==s.charAt(i+j) && (j+1>=i+j-1 ? true :
isPalin[j+1][i+j-1]);
            }
        }
        dfs(ans, new ArrayList<String>(), s, isPalin, 0);
        return ans;
    }

    private void dfs(List<List<String>> ans, List<String> list, String s, boolean[][]
isPalin, int row) {
        int length = s.length();
        if (row == length) { ans.add(new ArrayList<String>(list)); return; }
        for (int i=row; i<length; ++i) {
            if (isPalin[row][i]) {
                list.add(s.substring(row, i+1));
                dfs(ans, list, s, isPalin, i+1);
                list.remove(list.size()-1);
            }
        }
    }
}
```

PYTHON :

```
def partition(self, s):
    res = []
    self.dfs(s, [], res)
    return res

def dfs(self, s, path, res):
    if not s:
        res.append(path)
        return
    for i in range(1, len(s)+1):
        if self.isPal(s[:i]):
            self.dfs(s[i:], path+[s[:i]], res)

def isPal(self, s):
    return s == s[::-1]
```