```
class Solution {
    public List<List<String>> partition(String s) {
        int len = s.length();
        boolean[][] dp = new boolean[len][len];
        List<List<String>> res = new ArrayList<>();

        for (int i = len - 1; i >= 0; i--) {
            for (int j = i; j < len; j++) {
                if (s.charAt(i) == s.charAt(j) && (j - i < 3 || dp[i+1][j-1])) {
                    dp[i][j] = true;
                }
            }
        }
        h(dp, s, 0, res, new ArrayList<>());
        return res;
    }

    private void h(boolean[][] dp, String s, int idx, List<List<String>> res, List<String> tmp) {
        if (idx == s.length()) {
            res.add(new ArrayList<>(tmp));
            return;
        }
        for (int i = idx; i < s.length(); i++) {
            if (dp[idx][i]) {
                tmp.add(s.substring(idx, i+1));
                h(dp, s, i+1, res, tmp);
                tmp.remove(tmp.size()-1);
            }
        }
    }
}
```

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

Return all possible palindrome partitioning of *s*.

Example:

```
Input: "aab"
Output:
[
  ["aa","b"],
  ["a","a","b"]
]
```

```
class Solution {
    public int minCut(String s) {
        int len = s.length();
        boolean[][] dp = new boolean[len][len];
        int[] res = new int[len];
        for (int i = 0; i < len; i++) {
            res[i] = i;
        }
        for (int i = 0; i < len; i++) {
            for (int j = 0; j <= i; j++) {
                if (s.charAt(i) == s.charAt(j) && (i - j <
3 || dp[j+1][i-1])) {
                    dp[j][i] = true;
                    if (j == 0) {
                        res[i] = 0;
                    } else {
                        res[i] = Math.min(res[i], res[j-1]
+ 1);
                    }
                }
            }
        }

        return res[len - 1];
    }
}
```

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

Return the minimum cuts needed for a palindrome partitioning of *s*.

```
class Solution {
    public int minCut(String s) {
        List<List<String>> res = new ArrayList<>();
        int len = s.length();
        boolean dp[][] = new boolean[len][len];
        int[] arr = new int[len];
        for (int i = 0; i < arr.length; i++) {
            arr[i] = i;
        }
        for (int i = 0; i < len; i++) {
            for (int j = 0; j <= i; j++) {
                if (s.charAt(i) == s.charAt(j) && (i - j < 3 || dp[j+1][i-1])) {
                    dp[j][i] = true;
                    if (j == 0) {
                        arr[i] = 0;
                    } else {
                        arr[i] = Math.min(arr[i], 1 + arr[j-1]);
                    }
                }
            }
        }
        return arr[len-1];
    }}
```