17. Palindrome Partitioning

131. Palindrome Partitioning

Medium 7418 233 Add to List Share

Given a string `s`, partition `s` such that every substring of the partition is a **palindrome**. Return all possible palindrome partitioning of `s`.

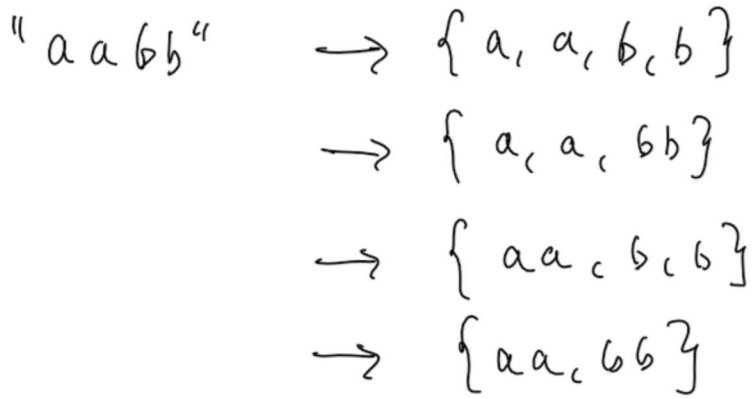
A **palindrome** string is a string that reads the same backward as forward.

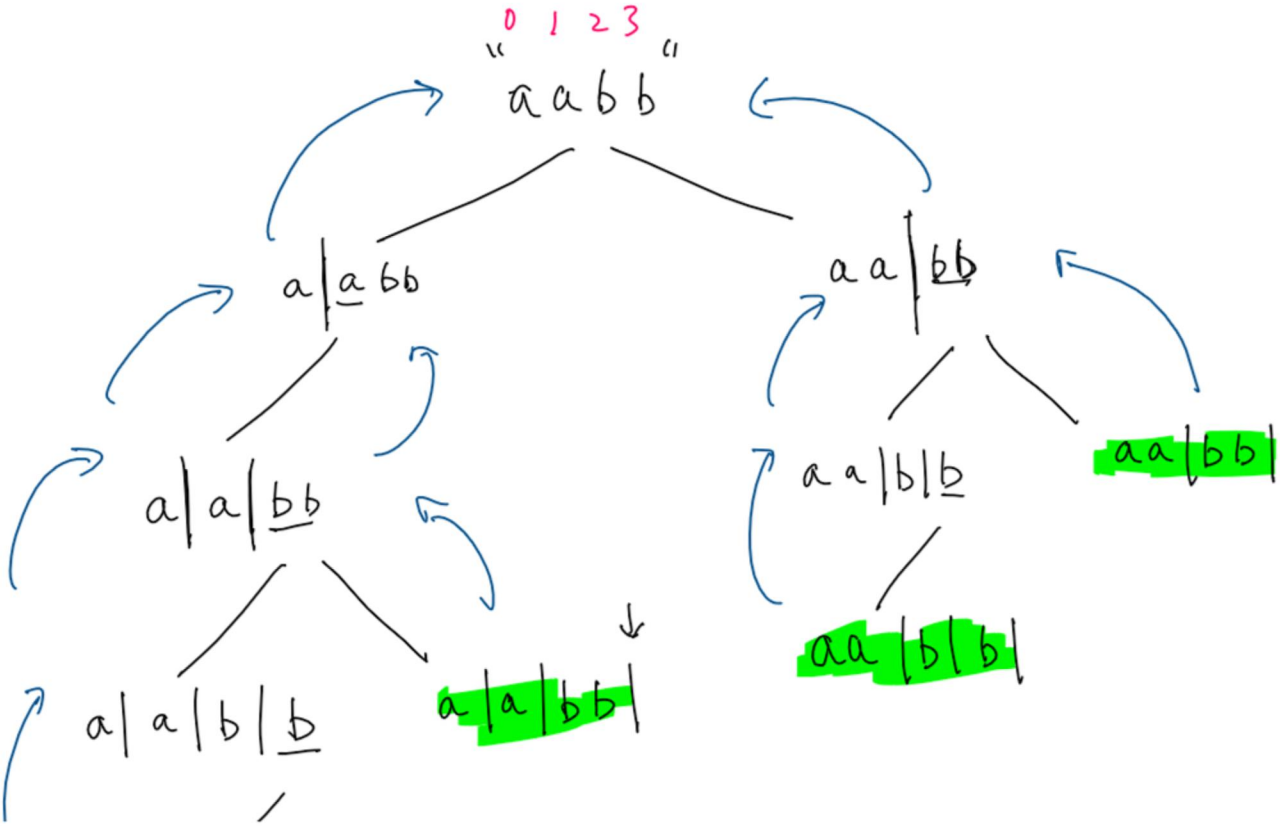
Example 1:

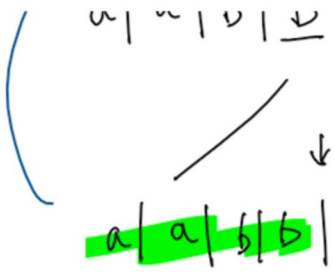
```
Input: s = "aab"
Output: [["a","a","b"],["aa","b"]]
```

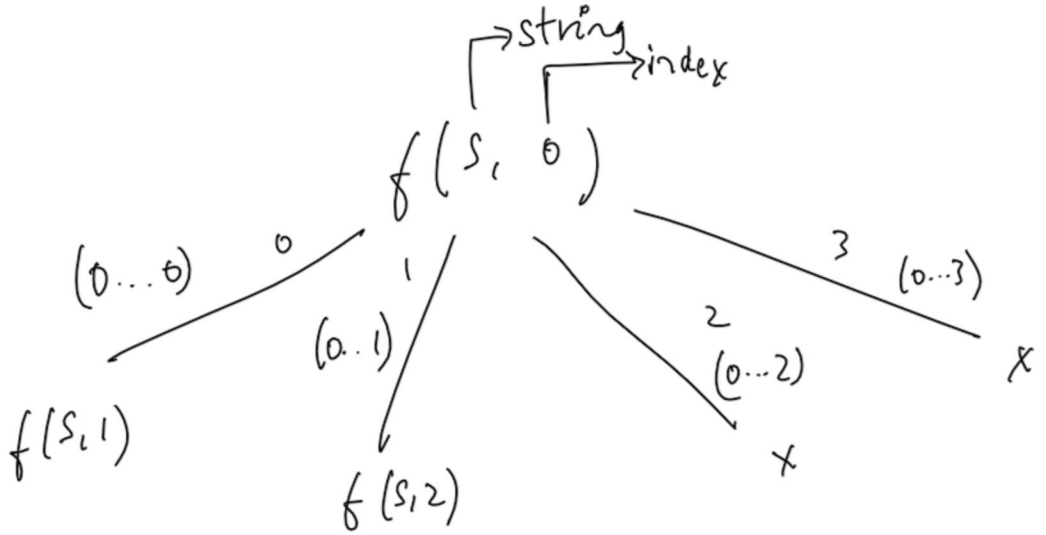
Example 2:

```
Input: s = "a"
Output: [["a"]]
```









f (S, ind)

↳ assuming i (0 to 3)

loop (ind → n-1) → i

When you will able to partion.

(ind ... i) → partion.

Base if (ind == n)

return

Java

Autocomplete

```java
class Solution {
    public List<List<String>> partition(String s) {
        List<List<String>> ans = new ArrayList<>();
        List<String> path = new ArrayList<>();
        func(0, s, path, ans);
        return ans;
    }

    void func(int index, String s, List<String> path, List<List<String>> ans){
        // base case
        if(index == s.length()){
            ans.add(new ArrayList<>(path));
            return;
        }
        for(int i = index; i < s.length(); ++i){
            if(isPalindrome(s, index, i)){
                path.add(s.substring(index, i + 1));
                func(i + 1, s, path, ans);
                path.remove(path.size() - 1);
            }
        }
    }

    boolean isPalindrome(String s, int start, int end){
        while(start <= end){
            if(s.charAt(start++) != s.charAt(end--)) return false;
        }
        return true;
    }
}
```

C++

Autocomplete

```cpp
class Solution {
public:
    vector<vector<string>> partition(string s) {
        vector<vector<string>> ans;
        vector<string> path; // it store the individual list of substring
        func(0, s, path, ans);
        return ans;
    }

    void func(int index, string s, vector<string> &path, vector<vector<string>> &ans){
        // base case
        if(index == s.size()){
            ans.push_back(path);
            return;
        }

        for(int i = index; i <= s.size(); ++i){
            if(isPalindrome(s, index, i)){
                path.push_back(s.substr(index, i - index + 1));
                func(i + 1, s, path, ans);
                path.pop_back();
            }
        }
    }

    bool isPalindrome(string s, int start, int end){
        while(start <= end){
            if(s[start++] != s[end--])
                return false;
        }
        return true;
    }
};
```