L20 Palindrome Partitioning

Friday, May 26, 2023 10:20 AM

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

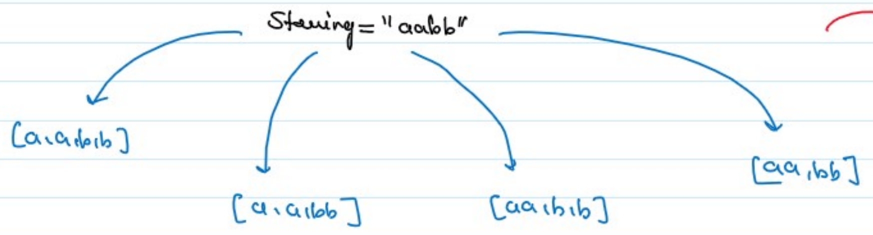
→ Means Not changed the Order of the String and find all the possible palindromic partation

Example 1:
Input: s = "aab"
Output: [["a","a","b"],["aa","b"]]
Example 2:
Input: s = "a"
Output: [["a"]]

Constraints:

- 1 <= s.length <= 16
- s contains only lowercase English letters.

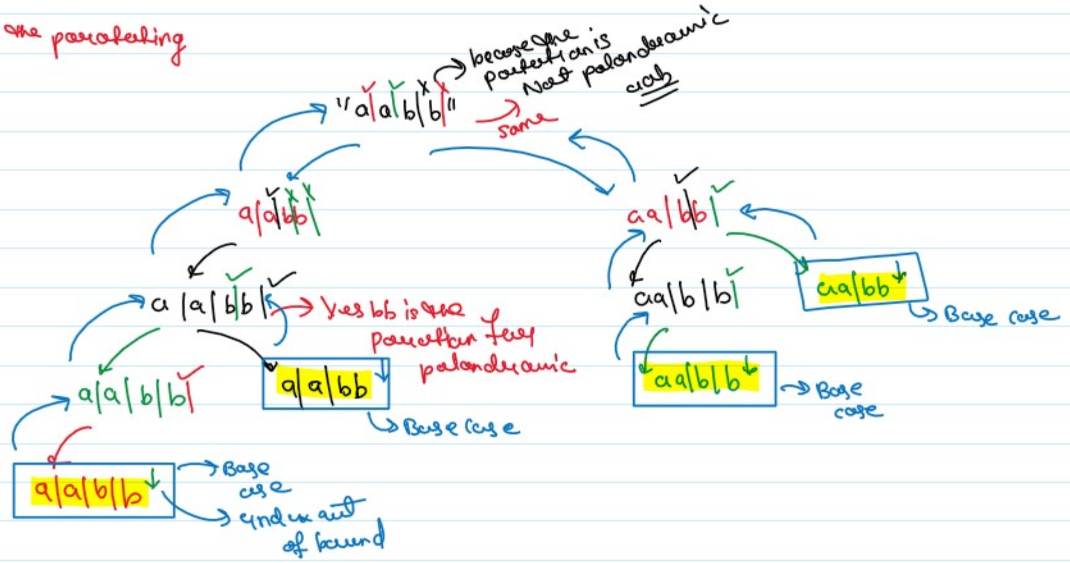
#lets Understand

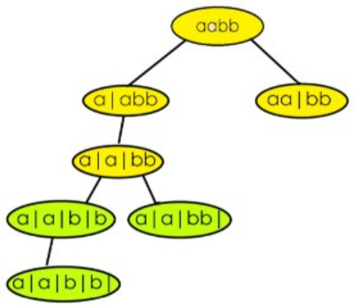


→ this is the (4) possible palindromic partation.

#thought Process

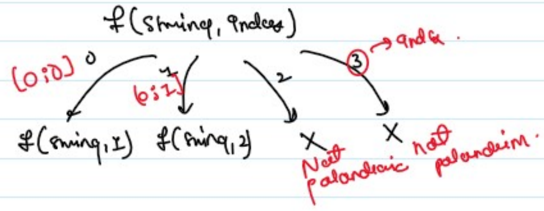
→ our first task is the partationing





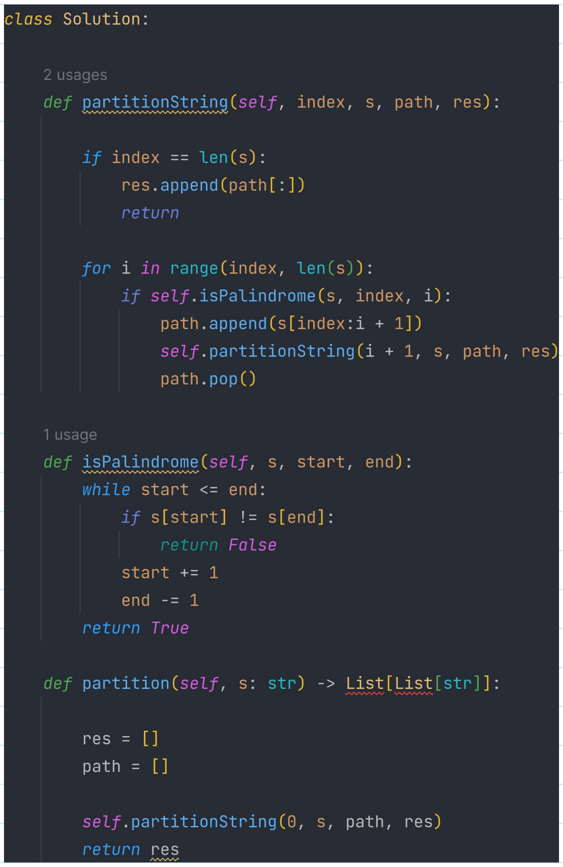
Sudo Code

String = "aabb"
index = 0

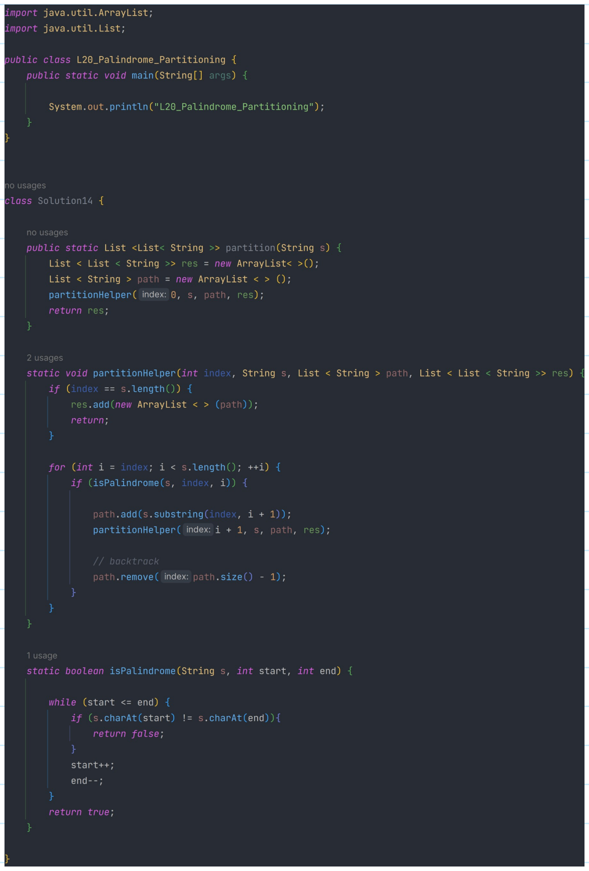


Time Complexity: O((2^n) *k*(n/2))
Reason: O(2^n) to generate every substring and **O(n/2)** to check if the substring generated is a palindrome. O(k) is for inserting the palindromes in another data structure, where k is the average length of the palindrome list.

Space Complexity: O(k * x)
Reason: The space complexity can vary depending upon the length of the answer. k is the average length of the list of palindromes and if we have x such list of palindromes in our final answer. The depth of the recursion tree is n, so the auxiliary space required is equal to the O(n).



```python
class Solution:

    2 usages
    def partitionString(self, index, s, path, res):

        if index == len(s):
            res.append(path[:])
            return

        for i in range(index, len(s)):
            if self.isPalindrome(s, index, i):
                path.append(s[index:i + 1])
                self.partitionString(i + 1, s, path, res)
                path.pop()

    1 usage
    def isPalindrome(self, s, start, end):
        while start <= end:
            if s[start] != s[end]:
                return False
            start += 1
            end -= 1
        return True

    def partition(self, s: str) -> List[List[str]]:

        res = []
        path = []

        self.partitionString(0, s, path, res)
        return res
```



```java
import java.util.ArrayList;
import java.util.List;

public class L20_Palindrome_Partitioning {
    public static void main(String[] args) {

        System.out.println("L20_Palindrome_Partitioning");
    }
}


no usages
class Solution14 {

    no usages
    public static List <List< String >> partition(String s) {
        List < List < String >> res = new ArrayList< >();
        List < String > path = new ArrayList < > ();
        partitionHelper( index: 0, s, path, res);
        return res;
    }

    2 usages
    static void partitionHelper(int index, String s, List < String > path, List < List < String >> res) {
        if (index == s.length()) {
            res.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));
                partitionHelper( index: i + 1, s, path, res);

                // backtrack
                path.remove( index: path.size() - 1);
            }
        }
    }

    1 usage
    static boolean isPalindrome(String s, int start, int end) {

        while (start <= end) {
            if (s.charAt(start) != s.charAt(end)){
                return false;
            }
            start++;
            end--;
        }
        return true;
    }

}
```