

Average Rating: 4.21 (24 votes)

June 11, 2017 | 25.9K views

267. Palindrome Permutation II

Given a string s, return all the palindromic permutations (without duplicates) of it. Return an empty list if no palindromic permutation could be form. Example 1:

Input: "aabb"

```
Output: ["abba", "baab"]
Example 2:
  Input: "abc"
  Output: []
```

Approach #1 Brute Force [Time Limit Exceeded]

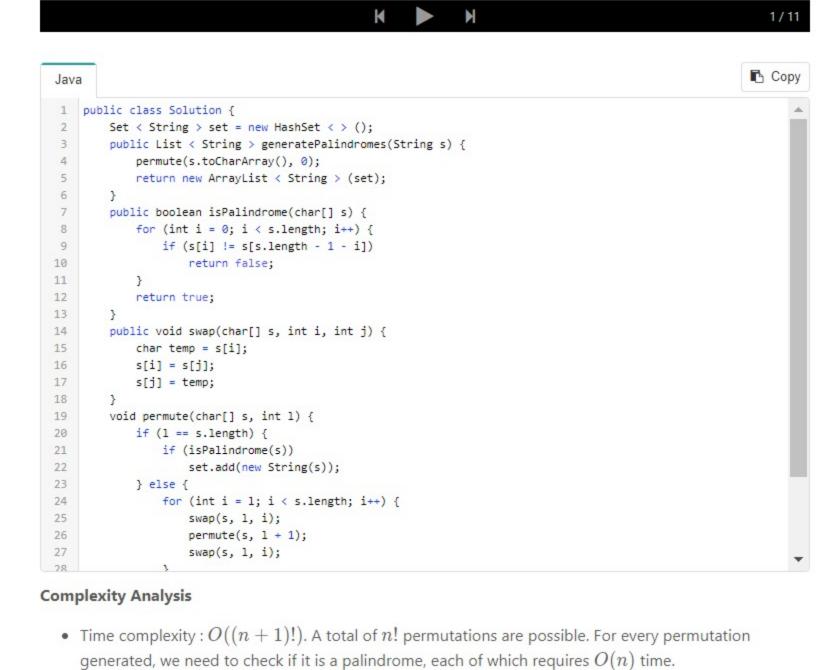
Solution

permutation is a palindrome. After this, the palindromic permuations can be added to a set in order to eliminate the duplicates. At the end, we can return an array comprised of the elements of this set as the

resultant array. Let's look at the way these permutations are generated. We make use of a recursive function permute which takes the index of the current element $current_index$ as one of the arguments. Then, it swaps the current element with every other element in the array, lying towards its right, so as to generate a new ordering of the array elements. After the swapping has been done, it makes another call to permute but this time with the

The simplest solution is generate every possible permutation of the given string s and check if the generated

index of the next element in the array. While returning back, we reverse the swapping done in the current function call. Thus, when we reach the end of the array, a new ordering of the array's elements is generated. The animation below depicts the ways the permutations are generated. В **Fixed Characters**



Approach #2 Backtracking [Accepted]

Algorithm

It might be possible that no palindromic permutation could be possible for the given string s. Thus, it is useless to generate the permutations in such a case. Taking this idea, firstly we check if a palindromic

ASCII characters possible). If the number of characters with odd number of occurences exceeds 1, it indicates that no palindromic permutation is possible for s. To look at this checking process in detail, look at Approach

permutation is possible for s. If yes, then only we proceed further with generating the permutations. To check this, we make use of a hashmap map which stores the number of occurences of each character(out of 128

• Space complexity : O(n). The depth of the recursion tree can go upto n.

Once we are sure that a palindromic permutation is possible for s, we go for the generation of the required

string to itself to create the palindromic permutations of s.

4 of the article for Palindrome Permutation.

generation of permutations i.e. we can generate only those permutations which are already palindromes. One idea to to do so is to generate only the first half of the palindromic string and to append its reverse string to itself to generate the full length palindromic string. Based on this idea, by making use of the number of occurrences of the characters in s stored in map, we create a string st which contains all the characters of s but with the number of occurrences of these characters in st reduced to half their original number of occurrences in s.

Thus, now we can generate all the permutations of this string st and append the reverse of this permuted

In case of a string s with odd length, whose palindromic permutations are possible, one of the characters in

s must be occurring an odd number of times. We keep a track of this character, ch, and it is kept separte from the string st. We again generate the permutations for st similarly and append the reverse of the generated permutation to itself, but we also place the character ch at the middle of the generated string.

permutations. But, instead of wildly generating all the permutations, we can include some smartness in the

In this way, only the required palindromic permutations will be generated. Even if we go with the above idea, a lot of duplicate strings will be generated.

In order to avoid generating duplicate palindromic permutations in the first place itself, as much as possible,

elements lying towards its right to generate the permutations. Before swapping, we can check if the elements

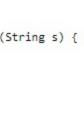
we can make use of this idea. As discussed in the last approach, we swap the current element with all the

being swapped are equal. If so, the permutations generated even after swapping the two will be

duplicates(redundant). Thus, we need not proceed further in such a case.

See this animation for a clearer understanding.

ch = "Y" s = "AAALYAL" st = "AAL" **Fixed Characters**



1/9

Сору

Next **①**

Sort By ▼

Post

Set < String > set = new HashSet < > (); public List < String > generatePalindromes(String s) { int[] map = new int[128]; char[] st = new char[s.length() / 2]; if (!canPermutePalindrome(s, map)) return new ArrayList < > (); char ch = 0; int k = 0; for (int i = 0; i < map.length; i++) { if (map[i] % 2 == 1) ch = (char) i; for (int j = 0; j < map[i] / 2; j++) { st[k++] = (char) i; } permute(st, 0, ch); return new ArrayList < String > (set); public boolean canPermutePalindrome(String s, int[] map) { int count = 0; for (int i = 0; i < s.length(); i++) { map[s.charAt(i)]++; if (map[s.charAt(i)] % 2 == 0)count--; else count++; **Complexity Analysis**

• Time complexity : $O\left(\left(\frac{n}{2}+1\right)!\right)$. Atmost $\frac{n}{2}!$ permutations need to be generated in the worst case.

Further, for each permutation generated, string.reverse() function will take n/4 time.

• Space complexity: O(n). The depth of recursion tree can go upto n/2 in the worst case.

Type comment here... (Markdown is supported)

sanketj * 49 @ February 12, 2020 2:06 PM

ashish53v 🛊 313 ② January 17, 2018 10:16 AM

36 ∧ ∨ ☑ Share ¬ Reply

SHOW 1 REPLY

class Solution {

SHOW 1 REPLY

String single = "";

0 A V C Share Share

0 A V 🗈 Share 🦘 Reply

List res;

My easy Java solution

11 A V Share Seply

jwzjwz2003cn ★2 ② April 29, 2020 5:45 PM

jimhorng * 5 @ January 23, 2018 6:48 AM

Preview

O Previous

Comments: 8

Rate this article: * * * * *

Java

9

10

11

12

13

14

15 16

17

18 19 20

21

22

23

24

25

26

27

public class Solution {

- jishenli1227 ★ 20 ② March 2, 2019 10:45 AM Thanks for the solution! I think one more thing that can make your code easier to read is to have a good var name. s, I, ch is very hard to remember 17 A V C Share Reply
- class Solution { public List<String> generatePalindromes(String s) { list(String) res = new Arravlist()():

Anyone else think this should be a hard problem rather than medium?

- I feel the problems in backtracking category are in general more difficult. Their easy seems like a medium problem, medium are more of hard problems, and hard are not more difficult, but takes more whiteboard space than medium ones 1 A V C Share Reply
- Since the Set < String > set = new HashSet < > (); need to store all the permutation before return, so Space Complexity is at most O((n/2+1)!), which is the same as time complexity, right? 1 A V 🗈 Share 🦘 Reply SHOW 1 REPLY RogerFederer 🖈 855 🗿 December 17, 2017 5:27 AM
 - class Solution(object): def permutePalindromesDetail(self, compressed, odd_char, result, tmp): if not compressed: palindrome = ''.join(tmp+[odd charl+tmp[::-1]) 1 A V C Share Share
 - SHOW 1 REPLY s961206 * 751 ② July 24, 2019 12:30 AM My solution that beats 99.83% (Skip duplicate branch)
 - Ewaolx 🛊 10 ② April 8, 2019 9:24 AM Thanks for the clear explanation. Just one query, in function "permute", what's the necessity of using set, if it is already handling the duplicate permutations. I might be missing something, a small clarification will be helpful. Thanks.

Read More