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.

For example:

Given `s = "aabb"`, return `["abba", "baab"]`.

Given `s = "abc"`, return `[]`.

1. If a palindromic permutation exists, we just need to generate the first half of the string.
2. To generate all distinct permutations of a (half of) string, use a similar approach from: Permutations II or Next Permutation.

Solution 1

Basically, the idea is to perform permutation on half of the palindromic string and then form the full palindromic result.

```
public List<String> generatePalindromes(String s) {
    int odd = 0;
    String mid = "";
    List<String> res = new ArrayList<>();
    List<Character> list = new ArrayList<>();
    Map<Character, Integer> map = new HashMap<>();

    // step 1. build character count map and count odds
    for (int i = 0; i < s.length(); i++) {
        char c = s.charAt(i);
        map.put(c, map.containsKey(c) ? map.get(c) + 1 : 1);
        odd += map.get(c) % 2 != 0 ? 1 : -1;
    }

    // cannot form any palindromic string
    if (odd > 1) return res;

    // step 2. add half count of each character to list
    for (Map.Entry<Character, Integer> entry : map.entrySet()) {
        char key = entry.getKey();
        int val = entry.getValue();

        if (val % 2 != 0) mid += key;

        for (int i = 0; i < val / 2; i++) list.add(key);
    }

    // step 3. generate all the permutations
    getPerm(list, mid, new boolean[list.size()], new StringBuilder(), res);

    return res;
}

// generate all unique permutation from list
void getPerm(List<Character> list, String mid, boolean[] used, StringBuilder sb,
List<String> res) {
    if (sb.length() == list.size()) {
        // form the palindromic string
        res.add(sb.toString() + mid + sb.reverse().toString());
        sb.reverse();
        return;
    }

    for (int i = 0; i < list.size(); i++) {
        // avoid duplication
        if (i > 0 && list.get(i) == list.get(i - 1) && !used[i - 1]) continue;

        if (!used[i]) {
            used[i] = true; sb.append(list.get(i));
            // recursion
            getPerm(list, mid, used, sb, res);
```

```
                // backtracking
                used[i] = false; sb.deleteCharAt(sb.length() - 1);
            }
        }
    }
```

written by jeantimex original link here

Solution 2

```
public List<String> generatePalindromes(String s) {
    int[] map = new int[256];
    for(int i=0;i<s.length();i++){
        map[s.charAt(i)]++;
    }
    int j=0,count=0;
    for(int i=0;i<256;i++){
        if(count== 0 && map[i] %2 == 1){
            j= i;
            count++;
        }else if(map[i] % 2==1){
            return new ArrayList<String>();
        }
    }
    String cur = "";
    if(j != 0){
        cur = ""+ (char)j;
        map[j]--;
    }
    List<String> res = new ArrayList<String>();
    DFS(res,cur,map,s.length());
    return res;
}
public void DFS(List<String> res,String cur,int[] map,int len){
    if(cur.length()== len){
        res.add(new String(cur));
    }else {
        for(int i=0;i<map.length;i++){
            if(map[i] <= 0) continue;
            map[i] = map[i] - 2;
            cur = (char)i + cur + (char)i;
            DFS(res,cur,map,len);
            cur = cur.substring(1,cur.length()-1);
            map[i] = map[i]+2;
        }
    }
}
```

written by zq670067 original link here

Solution 3

We only need to generate the first part of palindrome string, and the remaining part will be a middle character with the reverse of first part.

```
private List<String> list = new ArrayList<>();

public List<String> generatePalindromes(String s) {
    int numOdds = 0; // How many characters that have odd number of count
    int[] map = new int[256]; // Map from character to its frequency
    for (char c: s.toCharArray()) {
        map[c]++;
        numOdds = (map[c] & 1) == 1 ? numOdds+1 : numOdds-1;
    }
    if (numOdds > 1)   return list;

    String mid = "";
    int length = 0;
    for (int i = 0; i < 256; i++) {
        if (map[i] > 0) {
            if ((map[i] & 1) == 1) { // Char with odd count will be in the middle
                mid = "" + (char)i;
                map[i]--;
            }
            map[i] /= 2; // Cut in half since we only generate half string
            length += map[i]; // The length of half string
        }
    }
    generatePalindromesHelper(map, length, "", mid);
    return list;
}
private void generatePalindromesHelper(int[] map, int length, String s, String mi
d) {
    if (s.length() == length) {
        StringBuilder reverse = new StringBuilder(s).reverse(); // Second half
        list.add(s + mid + reverse);
        return;
    }
    for (int i = 0; i < 256; i++) { // backtracking just like permutation
        if (map[i] > 0) {
            map[i]--;
            generatePalindromesHelper(map, length, s + (char)i, mid);
            map[i]++;
        }
    }
}
```

written by lianngg original link here