题目描述

如果一个字符串正读和反渎都一样（大小写敏感），则称它为一个「回文串」，例如：

- leVel是一个「回文串」，因为它的正读和反读都是leVel；同理a也是「回文串」
- art不是一个「回文串」，因为它的反读tra与正读不同
- Level不是一个「回文串」，因为它的反读leveL与正读不同（因大小写敏感）

给你一个仅包含大小写字母的字符串，请用这些字母构造出一个最长的回文串，若有多个最长的，返回其中字典序最小的回文串。

字符串中的每个位置的字母最多备用一次，也可以不用。

输入描述

无

输出描述

无

用例

输入	abczcccddzz
输出	ccdzazdcc
说明	无

输入	ABabBabA
输出	ABabbaBA
说明	无

题目解析

回文串必然是对称的，可以分为三部分，即： 左边部分， 中间部分， 右边部分

其中左边部分 和 右边部分 互为倒序

而中间部分 可以是空串，可以是单字母。

比如 aba，其中左边部分是a，右边部分也是a，而中间部分是b

再比如 aa，其中左边部分是a，右边部分也是a，而中间部分是”“

因此，我的解题思路如下：

统计输入字符串各字母出现的次数：

- 如果字母出现次数>=2：

1. 字母出现次数为偶数，则可以平均分到左边部分，和右边部分
2. 字母出现次数为奇数，则平均分到左，右部分后，必然还会剩余一个无法成对

- 如果字母出现次数 == 1，则无法成对

对于剩余无法成对的字母，我们记录字典序最小的到mid中（题目要求返回其中字典序最小的回文串）

对于可以平均分配到左，右部分的字母，我们可以将可以成对的字母记录到ans数组，将ans字典序升序，拼接字符串即为回文串左边部分，ans字典序降序，拼接字符串即为回文串右边部分。

最终拼接：回文串左边部分 + 中间部分 + 回文串右边部分，即为题解。

JavaScript算法源码

```
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
```

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

rl.on("line", (line) => {
  console.log(getResult(line));
});

function getResult(str) {
  const count = {};

  // 统计各字母个数
  for (let c of str) {
    count[c] ? count[c]++ : (count[c] = 1);
  }

  const half = [];
  let mid = "";

  for (let c in count) {
    // 如果字母数量大于等于2，则可以成对出现，
    if (count[c] >= 2) {
      let n = Math.floor(count[c] / 2); // 对数
      half.push(...new Array(n).fill(c));
    }

    // 如果字母数量只有1个，或者字母数量大于2但是为奇数，则最后必然只剩单个字母可用，此时我们应该在这些无法成对的单字母中选择一
    if (count[c] % 2 != 0 && (mid == "" || c < mid)) {
      mid = c;
    }
  }

  half.sort();

  return half.join("") + mid + half.reverse().join(""); // 回文串左边部分 + 中间单字母 + 回文串右边部分
}
```

Java算法源码

```java
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    String str = sc.next();
    System.out.println(getResult(str));
  }

  private static String getResult(String str) {
    HashMap<Character, Integer> count = new HashMap<>();

    // 统计各字母个数
    for (int i = 0; i < str.length(); i++) {
      char c = str.charAt(i);
      count.put(c, count.getOrDefault(c, 0) + 1);
    }

    ArrayList<Character> half = new ArrayList<>();
    String mid = "";

    for (char c : count.keySet()) {
      int n = count.get(c);
      // 如果字母数量大于等于2，则可以成对出现，
      if (n >= 2) {
        for (int i = 0; i < n / 2; i++) half.add(c);
      }

      // 如果字母数量只有1个，或者字母数量大于2但是为奇数，则最后必然只剩单个字母可用，此时我们应该在这些无法成对的单字母中选择
      if (n % 2 != 0 && ("".equals(mid) || mid.compareTo(c + "") > 0)) {
        mid = c + "";
      }
    }

    half.sort((a, b) -> a - b);
    StringBuilder sb = new StringBuilder();
    for (Character c : half) sb.append(c);

    return sb + mid + sb.reverse(); // 回文串左边部分 + 中间单字母 + 回文串右边部分
  }
}
```

Python算法源码

Java算法源码

```java
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    String str = sc.next();
    System.out.println(getResult(str));
  }

  private static String getResult(String str) {
    HashMap<Character, Integer> count = new HashMap<>();

    // 统计各字母个数
    for (int i = 0; i < str.length(); i++) {
      char c = str.charAt(i);
      count.put(c, count.getOrDefault(c, 0) + 1);
    }

    ArrayList<Character> half = new ArrayList<>();
    String mid = "";

    for (char c : count.keySet()) {
      int n = count.get(c);
      // 如果字母数量大于等于2，则可以成对出现，
      if (n >= 2) {
        for (int i = 0; i < n / 2; i++) half.add(c);
      }

      // 如果字母数量只有1个，或者字母数量大于2但是为奇数，则最后必然只剩单个字母可用，此时我们应该在这些无法成对的单字母中选择
      if (n % 2 != 0 && ("".equals(mid) || mid.compareTo(c + "") > 0)) {
        mid = c + "";
      }
    }

    half.sort((a, b) -> a - b);
    StringBuilder sb = new StringBuilder();
    for (Character c : half) sb.append(c);

    return sb + mid + sb.reverse(); // 回文串左边部分 + 中间单字母 + 回文串右边部分
  }
}
```

Python算法源码

```python
# 输入获取
s = input()


# 算法入口
def getResult(s):
    count = {}

    # 统计各字母个数
    for c in s:
        if count.get(c) is None:
            count[c] = 0
        count[c] += 1

    ans = []
    mid = ""

    for c in count.keys():
        # 如果字母数量大于等于2，则可以成对出现，
        if count[c] >= 2:
            n = count[c] // 2
            ans.extend([c] * n)

        # 如果字母数量只有1个，或者字母数量大于2但是为奇数，则最后必然只剩单个字母可用，此时我们应该在这些无法成对的单字母中选择
        if count[c] % 2 != 0 and (mid == "" or c < mid):
            mid = c

    ans.sort()
    left = "".join(ans)

    ans.reverse()
    right = "".join(ans)

    return left + mid + right  # 回文串左边部分 + 中间单字母 + 回文串右边部分


# 调用算法
print(getResult(s))
```