## 37、没有回文串。考点 or 实现——数为搜索+回文子串涵盖判断

**题目描述**

回文串的定义：正读和反读都一样的字符串。

现在已经存在一个不包含回文串的字符串，字符串的字符都是在英语字母的前N个,且字符串不包含任何长度大于等于2的回文串；

请找出下一个字典序的不包含回文串的、字符都是在英语字母的前N个、且长度相同的字符串。

如果不存在，请输出NO。

**输入描述**

输入包括两行。

第一行有一个整数:N（1<=N<=26），表示字符串的每个字符范围都是前N的英语字母。

第二行输入一个字符串（输入长度<=10000），输入保证这个字符串是合法的并且没有包含回文串。

**输出描述**

输出下一个字典序的不包含回文串的、字符都是在英语字母的前N个、且长度相同的字符串；

如果不存在,请输出"NO"。

**用例**

| | |
|---|---|
| 输入 | 3<br>cba |
| 输出 | NO |
| 说明 | 无 |

**题目解析**

本题还是比较难的。

首先，题目的意思是：

第二行会输入一个不含超过2位的回文子串的字符串。

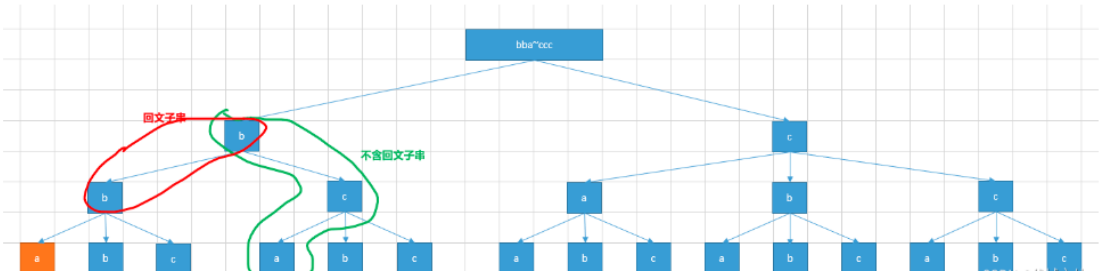
> 啥意思呢?
> 首先，我们需要了解下回文串概念：
> 回文串是指一个字符串，正序和反序是一样的，即回文串是中心对称的，比如aba，abba。通常而言，空串，比如""、单字符的字符串，比如"a"，都算是回文串。
> 这里题目说给定的字符串中不含有超过2位的回文子串，隐含意思是，让我们不要把空串和单字符的字符串当成回文串。比如"abc"是符合题目要求的不含回文子串的字符串，而"abbc"是不符合题目要求的不含有回文子串的字符串。

第一行会输入一个整数n，表示符合题目要求的不含回文子串的字符串的每一位字符：取自前n个的英语字母（小写）

> 啥意思呢?
> 这个是限定字符串每一位字符的取值，比如n=2，则每一位只能取"a"或"b"，比如n=3，则每一位只能在"a"、"b"、"c"字符中取值

输出下一个字典序的不包含回文串的、字符都是在英语字母的前N个、且长度相同的字符串；

> 啥意思呢?
> 我们假设现在输入是
> 3
> bba
> 那么bba后面的：字符都是在英语字母的前N个、且长度相同的字符串有哪些呢？如下图所示。
> 现在求bba下一个字典序的不包含回文串的字符串，从图中可以看出是bca。




因此，本题的求解可以分为两步：

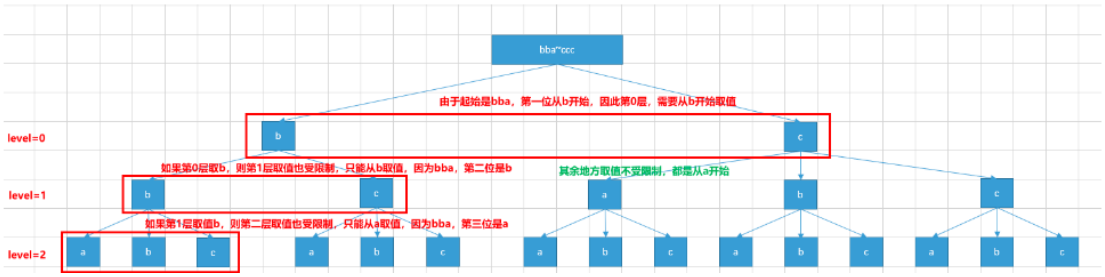
1、找出bba下一个字典序的、同范围、同长度的字符串F

2、判断找出的字符串F是否含有回文子串，若含有，则继续找下一个字符串，若不含，则输出找到的字符串

先看第一步，找下一个字典序的字符串F，该如何找呢？

从上面图树形结构，我们可以看出，这就是一个dfs的深度优先搜索的过程，那么该如何实现呢？

进行数位搜索时，如果是基于字符，那么就很难处理，我们可以预先将字符全部先转为数字，利用String(char).charCodeAt()方法得到字符char对应的ASCII码值，然后基于码值进行数位搜索。

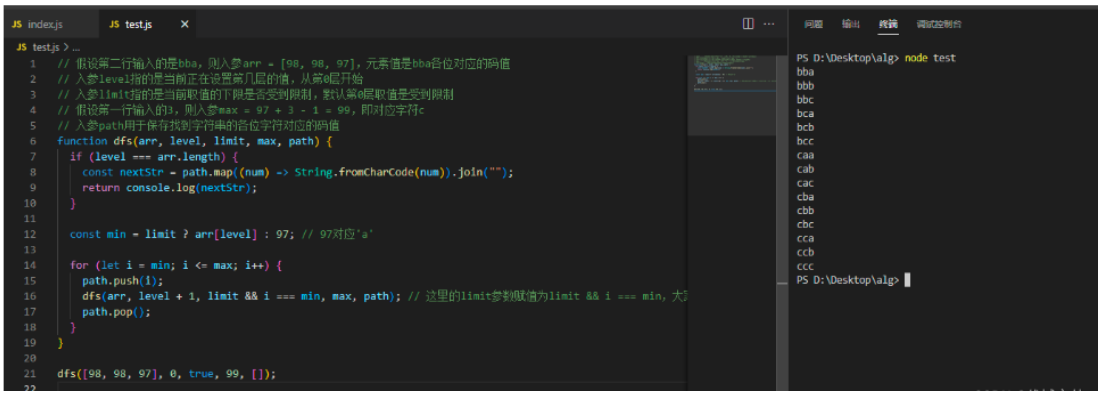



因此可以得到如下dfs逻辑

```
// 假设第二行输入的是bba，则入参arr = [98, 98, 97]，元素值是bba各位对应的码值
// 入参level指的是当前正在设置第几层的值，从第0层开始
// 入参limit指的是当前取值的下限是否受到限制，默认第0层取值是受到限制
// 假设第一行输入的3，则入参max = 97 + 3 - 1 = 99，即对应字符c
// 入参path用于保存找到字符串的各位字符对应的码值
function dfs(arr, level, limit, max, path){
    if(level === arr.length) {
        const nextStr = path.map(num => String.fromCharCode(num)).join('')
        return console.log(nextStr)
    }

    const min = limit ? arr[level] : 97; // 97对应'a'

    for(let i = min; i <= max; i++) {
        path.push(i)
        dfs(arr, level+1, limit && i === min, max, path) // 这里的limit参数赋值为limit && i === min，大家可以参考图示好好
        path.pop()
    }
}
```

上面就是基于数位搜索思想，来遍历bba~ccc之间所有字符串的方式

上面就是基于数位搜索思想，来遍历bba~ccc之间所有字符串的方式




接下来，我们就可以在遍历过程中，去做一些检测回文子串的动作。

我们可以很容易判断一个字符串是否为回文串（正序倒序相同），但是却无法轻易的判断一个字符串是否含有回文子串。

> 判断一个字符串str是否含有回文子串的方式，大致逻辑如下：
> 遍历字符串str的每一位，比如str[i]，然后将str[i]分别和str[i-1]、str[i+1]比较，若相同，则说明含有一个两位的回文子串，这其实是偶数位回文串判断方式，比如abbc，其中bb就是偶数位回文串。另外，我们还需要判断是否可能存在奇数位回文子串，即比较str[i-1]和str[i+1]是否相同，比如abac,其中aba就是奇数位回文串。
> 当遍历str每一位过程中，发现了符合要求的回文子串，则可以返回true，表示在str中发现了回文子串。

了解了如何判断字符串含有回文子串后，我们回到前面讨论，如何在dfs生成下一个字典序，且符合要求的字符串的过程中，去判断是否产生回文子串呢？

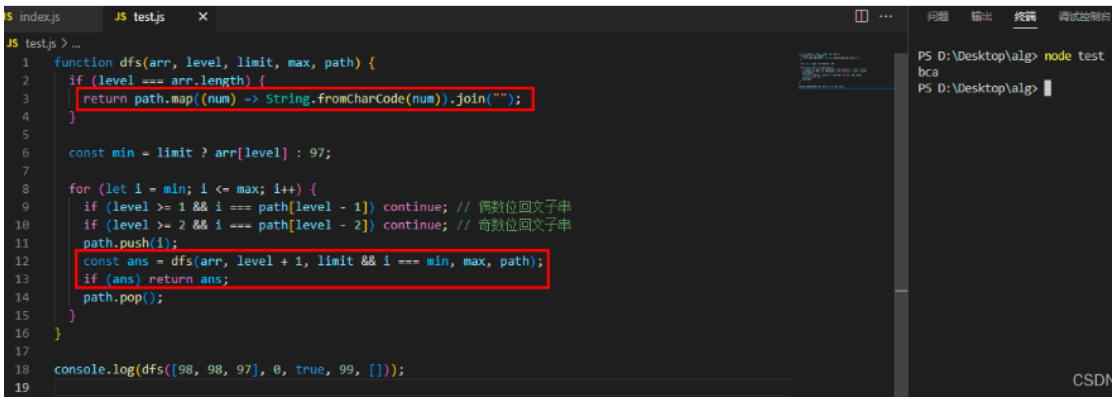
答：每当给一层取值时，比如给level=1层取值，则我们可以判断：

- arr[level - 1] === arr[level] 即判断是否出现偶数位回文子串
- arr[level - 2] === arr[level] 即判断是否出现奇数位回文子串

```
function dfs(arr, level, limit, max, path) {
  if (level === arr.length) {
    const nextStr = path.map((num) => String.fromCharCode(num)).join("");
    return console.log(nextStr);
  }

  const min = limit ? arr[level] : 97;

  for (let i = min; i <= max; i++) {
    if (level >= 1 && i === path[level - 1]) continue; // 偶数位回文子串
    if (level >= 2 && i === path[level - 2]) continue; // 奇数位回文子串
    path.push(i);
    dfs(arr, level + 1, limit && i === min, max, path);
    path.pop();
  }
}

dfs([98, 98, 97], 0, true, 99, []);
```


可以发现，答案给出的字符串少了很多，但都是不含回文子串的，字典序排后面的字符串。

但是我们只想要字典序下一个的，不想要下面全部的，因此：



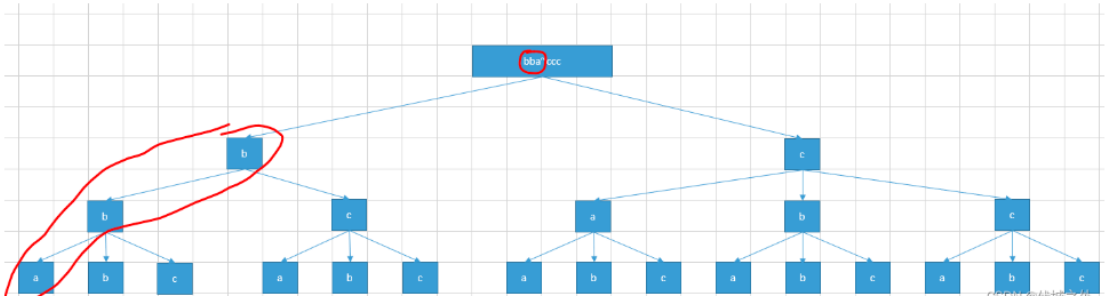

上面逻辑是，能走到dfs最后一步的肯定是符合要求的，不含回文子串的，处于下一个字典序的字符串，其他的都是中途失败的，返回undefined的，因此我们只要看dfs返回的是否不是undefined即可。

但是上面逻辑存在一个漏洞，可以尝试用例

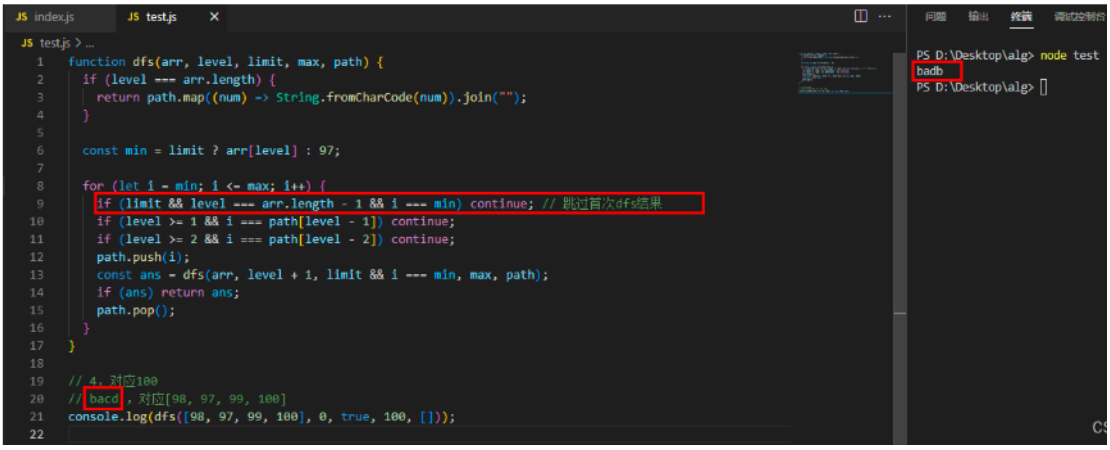
> 4
> bacd

可以发现，输出的还是bacd

原因是，我们在第一次dfs时，遍历的起始就是原始字符串




因此，我们要跳过第一次的遍历




```
function dfs(arr, level, limit, max, path) {
  if (level === arr.length) {
    return path.map((num) => String.fromCharCode(num)).join("");
  }

  const min = limit ? arr[level] : 97;

  for (let i = min; i <= max; i++) {
    if (limit && level === arr.length - 1 && i === min) continue; // 跳过首次dfs结果
    if (level >= 1 && i === path[level - 1]) continue;
    if (level >= 2 && i === path[level - 2]) continue;
    path.push(i);
    const ans = dfs(arr, level + 1, limit && i === min, max, path);
    if (ans) return ans;
    path.pop();
  }
}

// 4, 对应100
// bacd , 对应[98, 97, 99, 100]
console.log(dfs([98, 97, 99, 100], 0, true, 100, []));
```

## 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,
});

const lines = [];
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 2) {
    const n = lines[0] - 0;
    const str = lines[1];

    console.log(getResult(n, str));

    lines.length = 0;
  }
});

/**
 *
 * @param {*} n 字符串的每个字符范围都是前N的英语字母（1<=N<=26）
 * @param {*} str 合法的并且没有包含回文串的字符串（长度<=10000）
 */
function getResult(n, str) {
  const arr = [...str].map((char) => String(char).charCodeAt());
  const min = 97; // 对应字符'a'
  const max = min + n - 1;

  const ans = dfs(arr, 0, true, max, []);
  return ans ?? "NO";
}

function dfs(arr, level, limit, max, path) {
  if (level === arr.length) {
    return path.map((num) => String.fromCharCode(num)).join("");
  }

  const min = limit ? arr[level] : 97;

  for (let i = min; i <= max; i++) {
    if (limit && level === arr.length - 1 && i === min) continue; // 此步跳过原始字符串
    if (level >= 1 && i === path[level - 1]) continue; // 此步表示含有回文串，如abb这种情况
    if (level >= 2 && i === path[level - 2]) continue; // 此步表示含有回文串，如aba这种情况
    path.push(i);
    const ans = dfs(arr, level + 1, limit && i === min, max, path);
    if (ans) return ans; // 找到了不含回文串的字典序下一个字符串，则直接返回
    path.pop();
  }
}
```

## Java算法源码

```java
import java.util.LinkedList;
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int n = sc.nextInt();
    String s = sc.next();

    System.out.println(getResult(n, s));
  }

  /**
   * @param n 字符串的每个字符范围都是前N的英语字母（1<=N<=26）
   * @param s 合法的并且没有包含回文串的字符串（长度<=10000）
   */
  public static String getResult(int n, String s) {
    char[] tmp = s.toCharArray();
    int[] arr = new int[tmp.length];
    for (int i = 0; i < tmp.length; i++) arr[i] = tmp[i];
    int max = 97 + n - 1;

    String ans = dfs(arr, 0, true, max, new LinkedList<>());

    if (ans == null) {
      return "NO";
    } else {
```

```java
      return ans;
    }
  }

  public static String dfs(int[] arr, int level, boolean limit, int max, LinkedList<Integer> path) {
    if (level == arr.length) {
      StringBuilder sb = new StringBuilder();
      for (Integer num : path) sb.append((char) ((int) num));
      return sb.toString();
    }

    int min = limit ? arr[level] : 97;

    for (int i = min; i <= max; i++) {
      // 此步跳过原始字符串
      if (limit && level == arr.length - 1 && i == min) continue;
      //  此步表示含有回文串，如abb这种情况
      if (level >= 1 && i == path.get(level - 1)) continue;
      // 此步表示含有回文串，如aba这种情况
      if (level >= 2 && i == path.get(level - 2)) continue;
      path.add(i);
      String ans = dfs(arr, level + 1, limit && i == min, max, path);
      if (ans != null) return ans; // 找到了不含回文串的字典序下一个字符串，则直接返回
      path.removeLast();
    }

    return null;
  }
```

## Python算法源码

```python
# 输入获取
n = int(input())
s = input()


def dfs(arr, level, limit, maxV, path):
    if level == len(arr):
        return "".join(map(lambda x: chr(x), path))

    minV = arr[level] if limit else 97

    for i in range(minV, maxV + 1):
        # 此步跳过原始字符串
        if limit and level == len(arr) - 1 and i == minV:
            continue
        # 此步表示含有回文串，如abb这种情况
        if level >= 1 and i == path[level - 1]:
            continue
        # 此步表示含有回文串，如aba这种情况
        if level >= 2 and i == path[level - 2]:
            continue

        path.append(i)
        ans = dfs(arr, level + 1, limit and i == minV, maxV, path)
        # 找到了不含回文串的字典序下一个字符串，则直接返回
        if ans is not None:
```

```
            return ans
        path.pop()

    return None


# 算法入口
def getResult(n, s):
    """
    :param n: 字符串的每个字符范围都是前N的英语字母（1<=N<=26）
    :param s: 合法的并且没有包含回文串的字符串（长度<=10000）
    """
    arr = list(map(lambda x: ord(x), list(s)))
    maxV = 97 + n - 1

    ans = dfs(arr, 0, True, maxV, [])
    return ans or "NO"


# 算法调用
print(getResult(n, s))
```