


题目描述

有一个考古学家发现一个石碑，但是很可惜，发现时其已经断成多段，原地发现n个断口整齐的石碑碎片。为了破解石碑内容，希望有程序能帮忙计算复原后的石碑文字 **组合数** ，你能帮忙吗？

输入描述

第一行输入n，n表示石碑碎片的个数。

第二行依次输入石碑碎片上的文字内容s，共有n组。

输出描述

输出石碑文字的组合（按照 **升序排列** ），行末无多余空格。

用例

输入	3 a b c
输出	abc acb bac bca cab cba
说明	无

输入	3 a b a
输出	aab aba baa
说明	无

题目解析

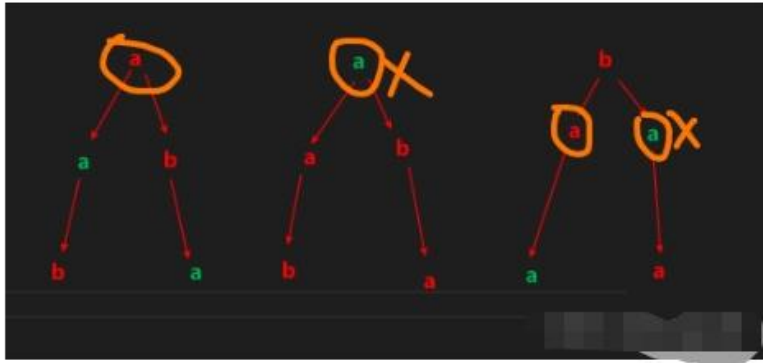
本题其实就是 

上面这篇博客提供了一种去重计数后求 **全排列**  的方案。

这里我介绍另一种方案，即树层去重。

如果我们将输入的数组进行排序，比如 [a,b,a] 升序后为 [a,a,b]

那么此时求全排列，必然会有如下过程：



我们发现，第二颗树和第一颗树完全重复，第三颗树的第二层的两个分支重复。

因此，按照本题要求，我们需要将上面两处重复去除。

其实这就是树层去重。

即全排列树的某一层兄弟节点重复了，则其以下分支必然重复。

因此，我们需要判断，如果全排列树的某一层兄弟节点发生重复，则不进行后续树枝生成。

那么如何判断同一层兄弟节点呢？

dfs的for循环中的操作就是当前树层的操作，而for循环中的递归dfs调用就是对下一树层的操作。

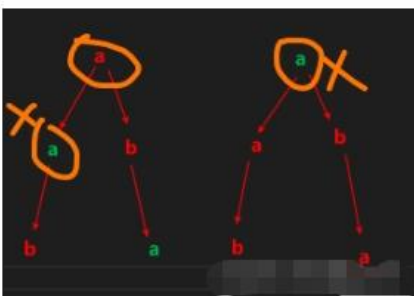
因此，我们判断当前树层的节点是否何其同一层的兄弟节点重复，即判断`arr[i] === arr[i-1]`

```

22 function dfs(arr, n, used, path) {
23   if (path.length === n) {
24     return console.log(path.join(""));
25   }
26
27   for (let i = 0; i < n; i++) {
28     if (i > 0 && arr[i] === arr[i - 1]) continue;
29     if (!used[i]) {
30       path.push(arr[i]);
31       used[i] = true;
32       dfs(arr, n, used, path);
33       used[i] = false;
34       path.pop();
35     }
36   }
37 }
38

```

但是这种方式，不仅会判断同一树层的兄弟节点，还会判断相邻树层的父子节点，如下图所示



因此，为了只对树层进行操作，我们需要增加一个过滤条件：

树枝父节点arr[i-1] 的 used[i-1] 是使用过的

而树层兄弟节点arr[i-1] 的 used[i-1]是未使用的

因此，最终判断如下

```
function dfs(arr, n, used, path) {  
  if (path.length === n) {  
    return console.log(path.join(""));  
  }  
  
  for (let i = 0; i < n; i++) {  
    if (i > 0 && arr[i] === arr[i - 1] && !used[i - 1]) continue;  
    if (!used[i]) {  
      path.push(arr[i]);  
      used[i] = true;  
      dfs(arr, n, used, path);  
      used[i] = false;  
      path.pop();  
    }  
  }  
}
```

2023.01.28，根据网友指正，本题中一个碎片上可能出现多个字母，比如下面用例：

```
3  
a b ab
```

此时，单纯依靠树层去重就不行了，还需缓存每一个排列结果到set中，利用set去重。

JavaScript算法源码

```
1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  const lines = [];
10 rl.on("line", (line) => {
11   lines.push(line);
12
13   if (lines.length === 2) {
14     const n = parseInt(lines[0]);
15     const arr = lines[1].split(" ").sort();
16
17     dfs(arr, n, new Array(n).fill(false), [], new Set());
18     lines.length = 0;
19   }
20 });
21
22 function dfs(arr, n, used, path, set) {
23   if (path.length === n) {
24     const ans = path.join("");
25
26     // 去重
27     if (!set.has(ans)) {
28       console.log(ans);
29       set.add(ans);
30     }
31
32     return
33   }
34
35   for (let i = 0; i < n; i++) {
36     if (i > 0 && arr[i] === arr[i - 1] && !used[i - 1]) continue;
37     if (!used[i]) {
38       path.push(arr[i]);
39       used[i] = true;
40       dfs(arr, n, used, path, set);
41       used[i] = false;
42       path.pop();
43     }
44   }
45 }
```

Java算法源码

```
1 import java.util.Arrays;
2 import java.util.HashSet;
3 import java.util.LinkedList;
4 import java.util.Scanner;
5
6 public class Main {
7     public static void main(String[] args) {
8         Scanner sc = new Scanner(System.in);
9
10        int n = sc.nextInt();
11
12        String[] arr = new String[n];
13        for (int i = 0; i < n; i++) {
14            arr[i] = sc.next();
15        }
16
17        Arrays.sort(arr);
18
19        HashSet<String> set = new HashSet<>();
20        dfs(arr, n, new boolean[n], new LinkedList<>(), set);
21    }
22
23    public static void dfs(
24        String[] arr, int n, boolean[] used, LinkedList<String> path, HashSet<String> set) {
25        if (path.size() == n) {
26            StringBuilder sb = new StringBuilder();
27
28            for (String node : path) {
29                sb.append(node);
30            }
31
32            String ans = sb.toString();
33
34            // 记录答案
35            if (!set.contains(ans)) {
36                System.out.println(ans);
37                set.add(ans);
38            }
39
40            return;
41        }
42
43        for (int i = 0; i < n; i++) {
44            if (i > 0 && arr[i].equals(arr[i - 1]) && !used[i - 1]) continue;
45            if (!used[i]) {
46                path.addLast(arr[i]);
47                used[i] = true;
48                dfs(arr, n, used, path, set);
49                used[i] = false;
50                path.removeLast();
51            }
52        }
53    }
54 }
```

Python算法源码

```
1  # 输入获取
2  n = int(input())
3  arr = input().split()
4
5
6  # 算法入口
7  def getResult(n, arr):
8      arr.sort()
9      dfs(arr, n, [False] * n, [], set())
10
11
12 def dfs(arr, n, used, path, cache):
13     if len(path) == n:
14         ans = "".join(path)
15         if ans not in cache:
16             print(ans)
17             cache.add(ans)
18         return
19
20     for i in range(n):
21         if i > 0 and arr[i] == arr[i-1] and not used[i-1]:
22             continue
23         if not used[i]:
24             path.append(arr[i])
25             used[i] = True
26             dfs(arr, n, used, path, cache)
27             used[i] = False
28             path.pop()
29
30
31 # 算法调用
32 getResult(n, arr)
```