

【华为OD机考 统一考试机试C卷】考古学家考古问题（C++ Java JavaScript Python C语言）

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

目前在考C卷，经过两个月的收集整理，**C卷真题已基本整理完毕**
抽到原题的概率为2/3到3/3，**也就是最少抽到两道原题。请注意：大家刷完C卷真题，最好要把B卷的真题刷一下，因为C卷的部分真题来自B卷。**
另外订阅专栏还可以联系笔者开通在线 **OJ** 进行刷题，提高刷题效率。
真题目录：华为OD机考机试 真题目录（C卷 + D卷 + B卷 + A卷） + 考点说明
专栏：2023华为OD机试(B卷+C卷+D卷) (C++JavaJSPy)
华为OD面试真题精选：华为OD面试真题精选
在线OJ：点击立即刷题，模拟真实机考环境

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，考古学家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

输出石碑文字的组数（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

全排列问题!!!

原题参考：[47. 全排列 II - 力扣 \(LeetCode\)](#)

解体思路

解决这个问题的方法是使用深度优先搜索（DFS）遍历所有可能的组合。以下是详细的思路：

- 首先，读取输入的石碑碎片个数 `n` 和石碑碎片上的文字内容 `s`。
- 将输入的石碑碎片内容存入一个列表 `charArray`，并对其进行排序。排序的目的是为了在遍历过程中方便地跳过重复的组合。
- 定义一个深度优先搜索函数 `dfs`，其中包含以下参数：
 - `charArray`：存储石碑碎片内容的列表。
 - `depth`：当前搜索的深度。
 - `path`：存储已经使用过的碎片。
 - `used`：记录每个碎片是否被使用过。
 - `result`：存储所有可能的组合。
- 在 `dfs` 函数中，首先检查当前搜索的深度是否等于石碑碎片的个数。如果是，则将当前组合加入结果列表 `result`。
- 遍历 `charArray` 中的每个碎片。对于每个碎片，检查它是否已经被使用过，以及它是否与前一个碎片相同且前一个碎片未被使用。如果满足这些条件，则跳过当前碎片。

6. 将当前碎片添加到 `path` 中, 并标记它为已使用。然后递归地搜索下一个碎片。
7. 在递归返回后, 执行回溯操作: 将当前碎片从 `path` 中移除, 并标记它为未使用。
8. 调用 `dfs` 函数, 开始搜索所有可能的组合。
9. 最后, 输出所有找到的组合。

这种方法可以有效地遍历所有可能的石碑文字组合, 并通过跳过重复的组合来减少搜索空间。

C++

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  #include <deque>
5  #include <string>
6
7  using namespace std;
8
9  // 深度优先搜索函数
10 void dfs(vector<string>& charArray, int depth, deque<string>& path, vector<bool>& used, vector<vector<string>>& result);
11
12 int main() {
13     // 处理输入
14     int n;
15     cin >> n;
16     cin.ignore(); // 忽略第一行剩余的换行符
17     string inputLine;
18     getline(cin, inputLine);
19     vector<string> charArray;
20     string tempStr = "";
21     for(int i = 0; i < inputLine.size(); i++) {
22         if(inputLine[i] == ' ') {
23             charArray.push_back(tempStr); // 将输入的碎片存入 charArray 中
24             tempStr = "";
25         }
26         else {
27             tempStr += inputLine[i];
28         }
29     }
30 }
```

```
~
31 charArray.push_back(tempStr); // 存入最后一个碎片
32 vector<vector<string>> result;
33
34 // 先对碎片进行排序
35 sort(charArray.begin(), charArray.end());
36 // path 中存储已经使用过的碎片
37 deque<string> path;
38 // 记录每个碎片是否被使用过
39 vector<bool> used(n, false);
40 dfs(charArray, 0, path, used, result);
41
42 // 输出所有组合
43 for (int i = 0; i < result.size(); i++) {
44     string outputStr = "";
45     for(int j = 0; j < result[i].size(); j++) {
46         outputStr += result[i][j];
47     }
48     cout << outputStr << endl;
49 }
50
51 return 0;
52 }
53
54 // 深度优先搜索函数
55 void dfs(vector<string>& charArray, int depth, deque<string>& path, vector<bool>& used, vector<vector<string>>& result) {
56     // 如果碎片都已经被使用过, 将当前组合加入结果中
57     if (depth == charArray.size()) {
58         result.push_back(vector<string>(path.begin(), path.end()));
59         return;
60     }
61     for (int i = 0; i < charArray.size(); i++) {
62         // 如果碎片已经被使用过, 则跳过
63         if (used[i]) {
64             continue;
65         }
66         // 如果当前碎片和前一个碎片相同, 并且前一个碎片还没有被使用, 则跳过
67         if (i > 0 && charArray[i] == charArray[i - 1] && !used[i - 1]) {
68             continue;
69         }
70         path.push_back(charArray[i]); // 将当前碎片存入 path 中
71     }
```

```
11      used[i] = true; // 标记当前碎片已被使用
12
13      dfs(charArray, depth + 1, path, used, result); // 递归搜索下一个碎片
14      path.pop_back(); // 回溯, 将当前碎片从 path 中移除
15      used[i] = false; // 标记当前碎片未被使用
16  }
17 }
```

JavaScript

```
1  const readline = require('readline');
2  const rl = readline.createInterface({
3      input: process.stdin,
4      output: process.stdout
5  });
6
7  // 深度优先搜索函数
8  function dfs(charArray, depth, path, used, result) {
9      // 如果碎片都被使用过, 将当前组合加入结果中
10     if (depth === charArray.length) {
11         result.push([...path]);
12         return;
13     }
14     for (let i = 0; i < charArray.length; i++) {
15         // 如果碎片已经被使用过, 则跳过
16         if (used[i]) {
17             continue;
18         }
19         // 如果当前碎片和前一个碎片相同, 并且前一个碎片还没有被使用, 则跳过
20         if (i > 0 && charArray[i] === charArray[i - 1] && !used[i - 1]) {
21             continue;
22         }
23         path.push(charArray[i]); // 将当前碎片存入 path 中
24         used[i] = true; // 标记当前碎片已被使用
25         dfs(charArray, depth + 1, path, used, result); // 递归搜索下一个碎片
26         path.pop(); // 回溯, 将当前碎片从 path 中移除
27         used[i] = false; // 标记当前碎片未被使用
28     }
29 }
30
31 rl.on('line', function (input) {
32     // ...
33 })
```

```
32 let n = parseInt(input);
33 rl.on('line', function (input) {
34   let charArray = input.split(' ');
35   let result = [];
36
37   // 先对碎片进行排序
38   charArray.sort();
39   // path 中存储已经使用过的碎片
40   let path = [];
41   // 记录每个碎片是否被使用过
42   let used = new Array(n).fill(false);
43   dfs(charArray, 0, path, used, result);
44
45   // 输出所有组合
46   for (let i = 0; i < result.length; i++) {
47     console.log(result[i].join(''));
48   }
49 });
50 });
```

Java

```
1 import java.util.*;
2
3 public class Main {
4   // 深度优先搜索函数
5   public static void dfs(String[] charArray, int depth, StringBuilder path, boolean[] used, List<String> result) {
6     // 如果碎片都已经被使用过, 将当前组合加入结果中
7     if (depth == charArray.length) {
8       result.add(path.toString());
9       return;
10    }
11    for (int i = 0; i < charArray.length; i++) {
12      // 如果碎片已经被使用过, 则跳过
13      if (used[i]) {
14        continue;
15      }
16      // 如果当前碎片和前一个碎片相同, 并且前一个碎片还没有被使用, 则跳过
17      if (i > 0 && charArray[i].equals(charArray[i - 1]) && !used[i - 1]) {
18        continue;
19      }
20    }
```

```
19     }
20     path.append(charArray[i]); // 将当前碎片存入 path 中
21     used[i] = true; // 标记当前碎片已被使用
22     dfs(charArray, depth + 1, path, used, result); // 递归搜索下一个碎片
23     path.setLength(path.length() - 1); // 回溯, 将当前碎片从 path 中移除
24     used[i] = false; // 标记当前碎片未被使用
25 }
26 }
27
28 public static void main(String[] args) {
29     Scanner scanner = new Scanner(System.in);
30     // 处理输入
31     int n = scanner.nextInt();
32     scanner.nextLine(); // 忽略第一行剩余的换行符
33     String inputLine = scanner.nextLine();
34     String[] charArray = inputLine.split(" ");
35     List<String> result = new ArrayList<>();
36
37     // 先对碎片进行排序
38     Arrays.sort(charArray);
39     // path 中存储已经使用过的碎片
40     StringBuilder path = new StringBuilder();
41     // 记录每个碎片是否被使用过
42     boolean[] used = new boolean[n];
43     dfs(charArray, 0, path, used, result);
44
45     // 输出所有组合
46     for (String s : result) {
47         System.out.println(s);
48     }
49 }
50 }
```

Python

```
1
2 import sys
3
4 # 深度优先搜索函数
5 def dfs(charArray, depth, path, used, result):
6     if depth == len(charArray):
7         result.append(path)
```



```
6 # 如果碎片都被使用过，将当前组合加入结果中
7
8 if depth == len(charArray):
9     result.append(list(path))
10    return
11 for i in range(len(charArray)):
12     # 如果碎片已经被使用过，则跳过
13     if used[i]:
14         continue
15     # 如果当前碎片和前一个碎片相同，并且前一个碎片还没有被使用，则跳过
16     if i > 0 and charArray[i] == charArray[i - 1] and not used[i - 1]:
17         continue
18     path.append(charArray[i]) # 将当前碎片存入 path 中
19     used[i] = True # 标记当前碎片已被使用
20     dfs(charArray, depth + 1, path, used, result) # 递归搜索下一个碎片
21     path.pop() # 回溯，将当前碎片从 path 中移除
22     used[i] = False # 标记当前碎片未被使用
23
24 if __name__ == '__main__':
25     # 处理输入
26     n = int(input())
27     inputLine = input()
28     charArray = inputLine.split()
29     result = []
30
31     # 先对碎片进行排序
32     charArray.sort()
33     # path 中存储已经使用过的碎片
34     path = []
35     # 记录每个碎片是否被使用过
36     used = [False] * n
37     dfs(charArray, 0, path, used, result)
38
39     # 输出所有组合
40     for i in range(len(result)):
41         outputStr = ''.join(result[i])
42         print(outputStr)
```

C语言

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 // 定义石碑碎片的最大个数
6 #define MAX_N 100
7
8 // 比较函数, 用于qsort排序
9 int compare(const void *a, const void *b) {
10     return strcmp(*(const char**)a, *(const char**)b);
11 }
12
13 // 深度优先搜索函数
14 void dfs(char *charArray[], int n, int depth, char *path, int *pathIndex, int *used, char result[][MAX_N], int *resultIndex) {
15     // 如果深度等于n, 将当前path存入result
16     if (depth == n) {
17         path[*pathIndex] = '\0';
18         strcpy(result[*resultIndex], path);
19         (*resultIndex)++;
20         return;
21     }
22
23     for (int i = 0; i < n; i++) {
24         // 跳过已使用的碎片
25         if (used[i]) {
26             continue;
27         }
28         // 跳过重复的碎片
29         if (i > 0 && strcmp(charArray[i], charArray[i - 1]) == 0 && !used[i - 1]) {
30             continue;
31         }
32         // 添加当前碎片到path
33         path[*pathIndex] = *charArray[i];
34         (*pathIndex)++;
35         used[i] = 1; // 标记为已使用
36         dfs(charArray, n, depth + 1, path, pathIndex, used, result, resultIndex); // 递归
37         (*pathIndex)--; // 回溯
38         used[i] = 0; // 标记为未使用
39     }
40 }
```

```
42
43 int main() {
44     int n;
45     scanf("%d", &n); // 读取碎片个数
46     char *charArray[MAX_N];
47     char buffer[MAX_N];
48
49     // 读取碎片
50     for (int i = 0; i < n; i++) {
51         charArray[i] = (char*)malloc(MAX_N * sizeof(char));
52         scanf("%s", charArray[i]);
53     }
54
55     // 对碎片排序
56     qsort(charArray, n, sizeof(char*), compare);
57
58     // 结果数组, path数组, 和辅助数组
59     char result[MAX_N][MAX_N];
60     char path[MAX_N];
61     int pathIndex = 0;
62     int used[MAX_N] = {0};
63     int resultIndex = 0;
64
65     // 执行深度优先搜索
66     dfs(charArray, n, 0, path, &pathIndex, used, result, &resultIndex);
67
68     // 输出所有组合
69     for (int i = 0; i < resultIndex; i++) {
70         printf("%s\n", result[i]);
71     }
72
73     // 释放内存
74     for (int i = 0; i < n; i++) {
75         free(charArray[i]);
76     }
77
78     return 0;
79 }
```

文章目录

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

题目描述

输入描述

输出描述

用例

题目解析

解体思路

C++

JavaScript

Java

Python

C语言

机考真题 华为OD



CSDN @算法大师