【华为OD机考 统一考试机试C卷】构成指定长度字符串的个数 / 字符串拼接 （ C++ + Java JavaScript python C语言）

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

2023年11月份，华为官方已经将 华为OD机考：OD统一考试（A卷 / B卷）切换到 OD统一考试（C卷）和 OD统一考试（D卷） 。

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：构成指定长度字符串的个数 (本题分值100)

给定 M（0 < M ≤ 30）个字符（a-z），从中取出任意字符（每个字符只能用一次）拼接成长度为 N（0 < N ≤ 5）的字符串，

要求相同的字符不能相邻，计算出给定的字符列表能拼接出多少种满足条件的字符串，

输入非法或者无法拼接出满足条件的字符串则返回0。

输入描述

给定的字符列表和结果字符串长度，中间使用空格(" ")拼接

输出描述

满足条件的字符串个数

用例1

输入

```
aab 2
```

输出

```
2
```

说明

只能构成ab,ba。

用例2

输入

```
abc 2
```

输出

```
6
```

说明

可以构成：ab ac ba bc ca cb 。

解题思路

使用递归和回溯的思想来生成不同的字符串。具体的逻辑如下：

1. 首先，我们定义一个函数 `generateDistinctStrings` ，这个函数接收以下参数：可用字符集 `s` ，目标字符串长度 `length` ，当前已生成的字符串 `current` ，已生成的结果集 `result` ，以及一个标记数组 `used` ，用来记录每个字符是否已被使用。
2. 在 `generateDistinctStrings` 函数中，首先检查当前已生成的字符串 `current` 的长度是否等于目标长度 `length` 。如果等于，说明我们已经生成了一个满足长度要求的字符串，将其添加到结果集 `result` 中，然后返回。
3. 如果当前字符串 `current` 的长度还未达到目标长度 `length` ，我们就需要继续添加字符。此时，我们遍历可用字符集 `s` 中的每一个字符。对于每一个字符，我们首先检查它是否已经被使用（通过查看 `used` 数组），以及它是否与 `current` 的最后一个字符相同。如果字符已经被使用，或者与 `current` 的最后一个字符相同，我们就跳过这个字符，继续检查下一个字符。
4. 如果一个字符未被使用，且与 `current` 的最后一个字符不同，我们就将它添加到 `current` 的末尾，然后标记这个字符为已使用，接着递归调用 `generateDistinctStrings` 函数，以生成下一个字符。

5. 在递归调用返回后，我们需要取消对当前字符的使用标记，以便在后续的遍历中可以再次使用这个字符。这就是回溯的思想，即撤销之前的选择，尝试其他的选择。

以下是对应的中文伪代码：

```
函数 generateDistinctStrings(s, length, current, result, used)
    如果 current的长度 等于 length
        将 current 添加到 result
        返回
    对于 s中的每一个字符 c
        如果 c已被使用 或者 c与current的最后一个字符相同
            继续下一次循环
        标记 c为已使用
        generateDistinctStrings(s, length, current + c, result, used)
        取消标记 c的使用状态
```

C++

```cpp
#include <iostream>
#include <unordered_set>
#include <vector>
#include <sstream>

using namespace std;

// 递归生成满足条件的不同字符串
void generateDistinctStrings(string s, int length, string current, unordered_set<string>& result, vector<bool>& used) {
    // 当生成的字符串长度等于指定长度时，将其加入到结果集中
    if (current.length() == length) {
        result.insert(current);
        return;
    }

    // 遍历字符串中的字符
    for (int i = 0; i < s.length(); i++) {
        // 判断字符是否已经被使用，或者当前字符与前一个字符相同
        if (used[i] || (current.length() > 0 && current.back() == s[i])) {
            continue;  // 如果字符已被使用或与前一个字符相同，则跳过当前字符

```

```
        }
        used[i] = true;  // 标记当前字符为已使用
        // 递归调用生成下一个字符
        generateDistinctStrings(s, length, current + s[i], result, used);
        used[i] = false;  // 取消标记当前字符的使用状态，以便下一次遍历
    }
}

// 计算满足条件的不同字符串的数量
int countDistinctStrings(string s, int length) {
    // 创建一个集合来存储不同的字符串
    unordered_set<string> distinctStrings;
    // 创建一个列表来标记字符串中的字符是否已经被使用
    vector<bool> used(s.length(), false);
    // 调用generateDistinctStrings方法生成满足条件的不同字符串
    generateDistinctStrings(s, length, "", distinctStrings, used);
    // 打印生成的所有不同的字符串
    // for (auto& str : distinctStrings) {
       // cout << str << endl;
    // }
    // 返回不同字符串的数量
    return distinctStrings.size();
}

int main() {
    string input;
    getline(cin, input);
    // 将输入的字符串按空格分割为两部分，分别为字符串和长度
    string str;
    int length;
    istringstream iss(input);
    iss >> str >> length;

    // 调用countDistinctStrings方法计算满足条件的不同字符串的数量
    int count = countDistinctStrings(str, length);
    // 输出计算结果
    cout <<  count << endl;

    return 0;
}

```

```
63
64
```

Java

```java
import java.util.*;

public class Main {
    public static void main(String[] args) {
        // 创建一个Scanner对象来读取用户的输入
        Scanner sc = new Scanner(System.in);
        // 读取用户输入的字符串
        String input = sc.nextLine();
        // 将输入的字符串按空格分割为两部分，分别为字符串和长度
        String[] parts = input.split(" ");
        String str = parts[0]; // 获取输入的字符串
        int length = Integer.parseInt(parts[1]); // 将输入的长度部分转换为整数

        // 调用countDistinctStrings方法计算满足条件的不同字符串的数量
        int count = countDistinctStrings(str, length);
        // 输出计算结果
        System.out.println(count);
    }

    // 计算满足条件的不同字符串的数量
    public static int countDistinctStrings(String str, int length) {
        // 创建一个HashSet来存储不同的字符串
        HashSet<String> set = new HashSet<>();
        // 创建一个boolean数组来标记字符串中的字符是否已经被使用
        boolean[] used = new boolean[str.length()];
        // 调用generateDistinctStrings方法生成满足条件的不同字符串
        generateDistinctStrings(str, length, "", set, used);
        // 打印生成的所有不同的字符串
        // for(String str1 : set){
           // System.out.println(str1);
        // }
        // 返回不同字符串的数量
        return set.size();
```

```
    }

    // 递归生成满足条件的不同字符串
    public static void generateDistinctStrings(String str, int length, String current, HashSet<String> set, boolean[] used) {
        // 当生成的字符串长度等于指定长度时，将其加入到HashSet中
        if (current.length() == length) {
            set.add(current);
            return;
        }

        // 遍历字符串中的字符
        for (int i = 0; i < str.length(); i++) {
            // 判断字符是否已经被使用，或者当前字符与前一个字符相同
            if (used[i] || (current.length() > 0 && current.charAt(current.length() - 1) == str.charAt(i))) {
                continue; // 如果字符已被使用或与前一个字符相同，则跳过当前字符
            }
            used[i] = true; // 标记当前字符为已使用
            // 递归调用生成下一个字符
            generateDistinctStrings(str, length, current + str.charAt(i), set, used);
            used[i] = false; // 取消标记当前字符的使用状态，以便下一次遍历
        }
    }
}
```

javaScript

```
// 导入所需的模块
const readline = require('readline');

// 创建一个接口来读取用户的输入
const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

// 递归生成满足条件的不同字符串
function generateDistinctStrings(str, length, current, set, used) {
  // 当生成的字符串长度等于指定长度时，将其加入到集合中
  if (current.length === length) {
```

```
    set.add(current);
    return;
  }

  // 遍历字符串中的字符
  for (let i = 0; i < str.length; i++) {
    // 判断字符是否已经被使用，或者当前字符与前一个字符相同
    if (used[i] || (current.length > 0 && current.charAt(current.length - 1) === str.charAt(i))) {
      continue; // 如果字符已被使用或与前一个字符相同，则跳过当前字符
    }
    used[i] = true; // 标记当前字符为已使用
    // 递归调用生成下一个字符
    generateDistinctStrings(str, length, current + str.charAt(i), set, used);
    used[i] = false; // 取消标记当前字符的使用状态，以便下一次遍历
  }
}

// 计算满足条件的不同字符串的数量
function countDistinctStrings(str, length) {
  // 创建一个集合来存储不同的字符串
  const set = new Set();
  // 创建一个数组来标记字符串中的字符是否已经被使用
  const used = new Array(str.length).fill(false);
  // 调用generateDistinctStrings方法生成满足条件的不同字符串
  generateDistinctStrings(str, length, "", set, used);
  // 打印生成的所有不同的字符串
  // for (let string of set) {
    // console.log(string);
  // }
  // 返回不同字符串的数量
  return set.size;
}

// 读取用户输入的字符串
rl.on('line', (input) => {
  // 将输入的字符串按空格分割为两部分，分别为字符串和长度
  const parts = input.split(" ");
  const str = parts[0]; // 获取输入的字符串
  const length = parseInt(parts[1]); // 将输入的长度部分转换为整数
```

```
  // 调用countDistinctStrings方法计算满足条件的不同字符串的数量
  const count = countDistinctStrings(str, length);
  // 输出计算结果
  console.log(count);

  rl.close();
});
```

Python

```python
# 导入所需的模块
from collections import defaultdict

# 递归生成满足条件的不同字符串
def generate_distinct_strings(s, length, current, result, used):
    # 当生成的字符串长度等于指定长度时，将其加入到结果集中
    if len(current) == length:
        result.add(current)
        return

    # 遍历字符串中的字符
    for i in range(len(s)):
        # 判断字符是否已经被使用，或者当前字符与前一个字符相同
        if used[i] or (len(current) > 0 and current[-1] == s[i]):
            continue  # 如果字符已被使用或与前一个字符相同，则跳过当前字符
        used[i] = True  # 标记当前字符为已使用
        # 递归调用生成下一个字符
        generate_distinct_strings(s, length, current + s[i], result, used)
        used[i] = False  # 取消标记当前字符的使用状态，以便下一次遍历

# 计算满足条件的不同字符串的数量
def count_distinct_strings(s, length):
    # 创建一个集合来存储不同的字符串
    distinct_strings = set()
    # 创建一个列表来标记字符串中的字符是否已经被使用
    used = [False] * len(s)
    # 调用generate_distinct_strings方法生成满足条件的不同字符串
    generate_distinct_strings(s, length, "", distinct_strings, used)
    # 打印生成的所有不同的字符串
```

```
    # for string in distinct_strings:
       # print(string)
    # 返回不同字符串的数量
    return len(distinct_strings)

# 读取用户输入的字符串
input_str = input()
# 将输入的字符串按空格分割为两部分，分别为字符串和长度
parts = input_str.split(" ")
s = parts[0]  # 获取输入的字符串
length = int(parts[1])  # 将输入的长度部分转换为整数

# 调用count_distinct_strings方法计算满足条件的不同字符串的数量
count = count_distinct_strings(s, length)
# 输出计算结果
print(count)
```

C语言

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_SIZE 31

char inputString[MAX_SIZE];  // 存储输入的字符串
int stringLength;            // 存储输入字符串的长度
int targetLength;            // 目标排列的长度
int validCount = 0;          // 符合条件的排列个数

void generateDistinctStrings(int lastUsedIndex, int currentLength, int usedFlags[]) {
    // 当前排列长度达到目标长度时，增加计数并返回
    if (currentLength == targetLength) {
        validCount++;
        return;
    }

    // 遍历每个字符
    for (int i = 0; i < stringLength; i++) {
```

```
        // 跳过已使用的字符
        if (usedFlags[i]) continue;

        // 跳过与上一个字符相同的字符，避免相邻重复
        if (lastUsedIndex >= 0 && inputString[i] == inputString[lastUsedIndex]) continue;

        // 树层去重：跳过重复字符生成的相同排列
        if (i > 0 && inputString[i] == inputString[i - 1] && !usedFlags[i - 1]) continue;

        // 标记字符为已使用，并递归生成下一层排列
        usedFlags[i] = 1;
        generateDistinctStrings(i, currentLength + 1, usedFlags);
        // 回溯，标记字符为未使用
        usedFlags[i] = 0;
    }
}

int main() {
    // 读入字符串和目标排列长度
    scanf("%s %d", inputString, &targetLength);

    // 计算输入字符串的长度
    stringLength = strlen(inputString);

    // 初始化标记数组
    int usedFlags[MAX_SIZE] = {0};
    // 从空排列开始生成
    generateDistinctStrings(-1, 0, usedFlags);

    // 输出有效排列的数量
    printf("%d\n", validCount);

    return 0;
}
```

完整用例

用例1

aabc 2

用例2

aabb 4

用例3

aab 3

用例4

abcd 2

用例5

abcd 4

用例6

abc 4

用例7

a 2

用例8

a 1

用例9

aaabbb 3

用例10

abcdef 3

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：构成指定长度字符串的个数 (本题分值100)
输入描述

输出描述
用例1
用例2
解题思路
C++
Java
javaScript
Python
C语言
完整用例
    用例1
    用例2
    用例3
    用例4
    用例5
    用例6
    用例7
    用例8
    用例9
    用例10

机考C卷真题
华为OD