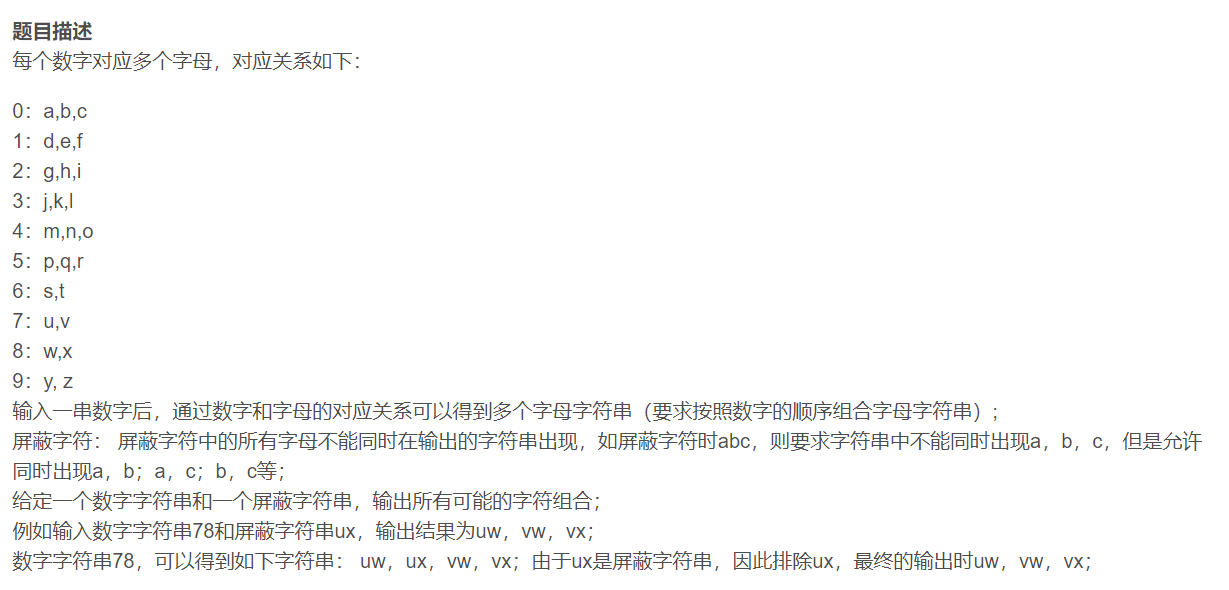
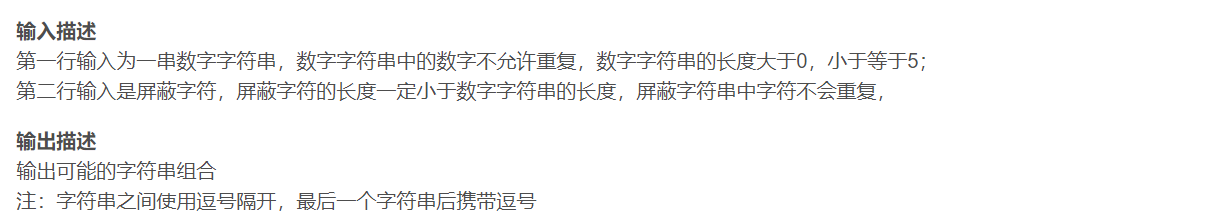
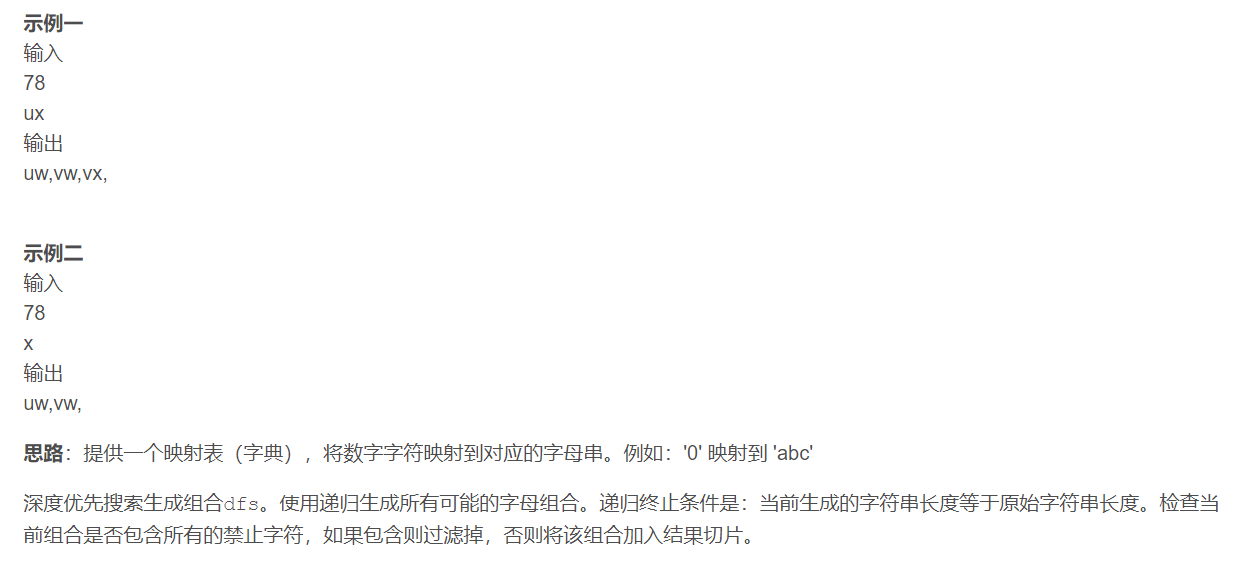
# **E卷-字母组合[200分]（ Java | Python3 | C++ | C语言 | JsNode | Go）**









import java.util.\*;

public class Main {

// 字典 mp 用于存储数字字符与对应字母串的映射

static Map<Character, String> mp = new HashMap<>();

// 集合 st 用于存储禁止出现的字符

static Set<Character> st = new HashSet<>();

// 列表 res 用于存储结果字符串

static List<String> res = new ArrayList<>();

// 初始化函数，建立数字与其对应字母的映射

public static void init() {

mp.put('0', "abc");

mp.put('1', "def");

mp.put('2', "ghi");

mp.put('3', "jkl");

mp.put('4', "mno");

mp.put('5', "pqr");

mp.put('6', "st");

mp.put('7', "uv");

mp.put('8', "wx");

mp.put('9', "yz");

}

// 深度优先搜索函数生成所有可能的字母组合

public static void dfs(int x, String now, int n, String s) {

// 递归终止条件：当拼接的字符串长度等于输入字符串长度时

if (x == n) {

// 集合 mk 用于存储当前组合中出现的禁止字符

Set<Character> mk = new HashSet<>();

for (char c : now.toCharArray()) {

// 检查并存储禁止字符

if (st.contains(c))

mk.add(c);

}

// 如果当前组合包含所有的禁止字符，则不加入结果集

if (mk.size() == st.size())

return;

// 当前组合有效，加入结果集

res.add(now);

return;

}

// 获取当前数字对应的字母串

String tmp = mp.get(s.charAt(x));

// 遍历字母串，递归处理后续字符

for (char c : tmp.toCharArray()) {

dfs(x + 1, now + c, n, s);

}

}

public static void main(String[] args) {

init(); // 初始化数字与字母的映射

Scanner scanner = new Scanner(System.in);

// 循环读取输入

while (scanner.hasNext()) {

String s = scanner.next(); // 读取数字字符串

String t = scanner.next(); // 读取禁止字符集合

res.clear(); // 清空上一次运行的结果

st.clear(); // 清空上一次运行的禁止字符集合

// 将禁止字符存入集合中

for (char c : t.toCharArray())

st.add(c);

// 进行深度优先搜索组合

dfs(0, "", s.length(), s);

// 输出结果

for (int i = 0; i < res.size(); i++) {

System.out.print(res.get(i) + ",");

}

System.out.println();

}

scanner.close(); // 关闭 Scanner 对象

}

}



# 初始化数字对应字母的字典

def init():

return {

"0": "abc",

"1": "def",

"2": "ghi",

"3": "jkl",

"4": "mno",

"5": "pqr",

"6": "st",

"7": "uv",

"8": "wx",

"9": "yz",

}

# 深度优先搜索生成组合

def dfs(x, now, n, s, mp, st, res):

if x == n:

mk = set() # 用于存储当前组合中的禁止字符

for char in now:

if char in st:

mk.add(char)

if len(mk) == len(st): # 组合中包含所有的禁止字符

return

res.append(now) # 加入有效组合

return

tmp = mp[s[x]] # 获取当前数字对应的字母串

for char in tmp:

dfs(x + 1, now + char, n, s, mp, st, res) # 递归处理后续字符

def main():

mp = init() # 初始化映射

while True:

try:

s = input().strip()

t = input().strip()

except EOFError:

break

res = []

st = set(t) # 将禁止字符存入集合

dfs(0, "", len(s), s, mp, st, res) # 进行深度优先搜索组合

print(",".join(res),end="")

print(",")

if \_\_name\_\_ == "\_\_main\_\_":

main()



#include<bits/stdc++.h>

using namespace std;

// 使用 typedef 定义 ll 为 long long 的简写

typedef long long ll;

string t; // 用于存储输入的禁止出现的字符集合的字符串

map<char, string> mp; // 键值对映射，每个数字字符对应一组字母

set<char> st; // 用于存储禁止出现的字符集合

// 初始化函数，建立数字与其对应字母的映射

void init() {

mp['0'] = "abc";

mp['1'] = "def";

mp['2'] = "ghi";

mp['3'] = "jkl";

mp['4'] = "mno";

mp['5'] = "pqr";

mp['6'] = "st";

mp['7'] = "uv";

mp['8'] = "wx";

mp['9'] = "yz";

}

string s; // 用于存储输入的数字字符串

vector<string> res; // 存储结果字符串的向量

// 深度优先搜索函数生成所有可能的字母组合

void dfs(int x, string now, int n) {

// 递归终止条件：当拼接的字符串长度等于输入字符串长度时

if (x == n) {

set<char> mk; // 临时集合用于存储当前组合中出现的禁止字符

// 检查当前组合是否包含禁止字符

for (int i = 0; i < now.length(); i++) {

if (st.count(now[i])) mk.insert(now[i]);

}

// 如果当前组合包含所有的禁止字符，则不加入结果集

if (mk.size() == st.size()) return;

res.push\_back(now); // 当前组合有效，加入结果集

return;

}

// 获取当前数字对应的字母串

string tmp = mp[s[x]];

// 遍历字母串，递归处理后续字符

for (int i = 0; i < tmp.length(); i++) {

dfs(x + 1, now + tmp[i], n);

}

}

int main() {

init(); // 初始化数字与字母的映射

// 循环每组输入

while (cin >> s) {

cin >> t; // 读取禁止字符集合

res.clear(); // 清空上一次的结果

st.clear(); // 清空上一次的禁止字符集合

// 将禁止字符存入集合中

for (int i = 0; i < t.size(); i++) st.insert(t[i]);

// 进行深度优先搜索组合

dfs(0, "", s.size());

// 输出结果

for (int i = 0; i < res.size(); i++) {

cout << res[i] << ",";

}

cout << endl;

}

}



#include <stdio.h>

#include <string.h>

#include <stdlib.h>

// 映射结构体，用于存储数字与对应字母的映射关系

typedef struct {

char key; // 数字字符

char value[4]; // 对应的字母串，最大长度为3，因而数组长度设为4

} MapEntry;

// 初始化数字与字母的映射表

MapEntry mp[] = {

{'0', "abc"}, {'1', "def"}, {'2', "ghi"},

{'3', "jkl"}, {'4', "mno"}, {'5', "pqr"},

{'6', "st"}, {'7', "uv"}, {'8', "wx"},

{'9', "yz"}

};

int mp\_size = 10; // 映射表的大小

// 深度优先搜索函数生成所有可能的字母组合

void dfs(int x, char\* s, char\* now, int n, char\* ans[], int\* ans\_size, char\* st,

int st\_size) {

// 递归终止条件：当拼接的字符串长度等于输入字符串长度时

if (x == n) {

int st\_hit\_count = 0; // 记录当前组合中出现了多少禁止字符

for (int i = 0; now[i]; i++) {

for (int j = 0; j < st\_size; j++) {

if (now[i] == st[j]) {

st\_hit\_count++;

break; // 找到一个禁止字符后立即跳出内层循环

}

}

}

// 如果当前组合包含所有的禁止字符，则不加入结果集

if (st\_hit\_count == st\_size) return;

// 当前组合有效，加入结果集

ans[\*ans\_size] = strdup(now); // 使用 strdup 复制当前字符串

(\*ans\_size)++;

return;

}

char\* tmp = ""; // 存储当前数字对应的字母串

// 找到当前数字对应的字母串

for (int i = 0; i < mp\_size; i++) {

if (mp[i].key == s[x]) {

tmp = mp[i].value;

break;

}

}

// 遍历字母串，递归处理后续字符

for (int i = 0; tmp[i]; i++) {

size\_t len = strlen(now); // 获取当前组合的长度

now[len] = tmp[i]; // 添加新字符到当前组合

now[len + 1] = '\0'; // 确保字符串以 '\0' 结尾

dfs(x + 1, s, now, n, ans, ans\_size, st,

st\_size); // 递归处理下一个位置

now[len] = '\0'; // 恢复字符串至原始状态以备下一次迭代

}

}

int main() {

char s[1000], t[1000]; // 存储输入的数字字符串和禁止字符集合

char\* ans[1000]; // 存储所有可能的有效组合

int ans\_size = 0; // 结果组合的数量

// 循环读取输入

while (scanf("%s %s", s, t) == 2) {

ans\_size = 0; // 重置结果组合数量

char st[1000]; // 存储禁止字符集合

int st\_size = 0; // 禁止字符集合的大小

// 将禁止字符加入集合

for (int i = 0; t[i]; i++) {

st[st\_size++] = t[i];

}

char now[1000] = ""; // 存储当前组合的字符串

// 进行深度优先搜索生成所有可能的组合

dfs(0, s, now, strlen(s), ans, &ans\_size, st, st\_size);

// 输出所有有效的结果组合并释放内存

for (int i = 0; i < ans\_size; i++) {

printf("%s,", ans[i]);

free(ans[i]); // 释放之前分配的内存

}

printf("\n"); // 每行的输出结束

}

return 0;

}



// 初始化数字对应字母的映射

function init() {

return {

0: "abc",

1: "def",

2: "ghi",

3: "jkl",

4: "mno",

5: "pqr",

6: "st",

7: "uv",

8: "wx",

9: "yz",

};

}

// 深度优先搜索生成组合

function dfs(x, now, n, s, mp, st, res) {

// 递归终止条件：当前生成的字符串长度等于原始字符串长度

if (x === n) {

let mk = new Set();

// 检查当前组合中是否包含禁止字符

for (let i = 0; i < now.length; i++) {

if (st.has(now[i])) {

mk.add(now[i]); // 存储禁止字符

}

}

// 如果当前组合包含所有的禁止字符，则不加入结果集

if (mk.size === st.size) {

return;

}

// 当前组合有效，加入结果集

res.push(now);

return;

}

let tmp = mp[s[x]]; // 获取当前数字对应的字母串

// 遍历字母串，递归处理后续字符

for (let i = 0; i < tmp.length; i++) {

dfs(x + 1, now + tmp[i], n, s, mp, st, res);

}

}

const readline = require("readline").createInterface({

input: process.stdin,

output: process.stdout,

});

let mp = init(); // 初始化数字与字母的映射关系

// 处理输入

readline.on("line", (line) => {

if (!global.s) {

// 读取第一个输入字符串

global.s = line.trim();

} else {

// 读取第二个输入字符串

global.t = line.trim();

let res = []; // 用于存储结果的数组

let st = new Set(global.t); // 将禁止字符存入集合中

// 进行深度优先搜索组合

dfs(0, "", global.s.length, global.s, mp, st, res);

// 输出结果

console.log(res.join(",") + ",");

// 删除全局变量，以便处理新的输入

delete global.s;

delete global.t;

}

});



package main

import (

"bufio"

"fmt"

"os"

"strings"

)

// 初始化数字与字母的映射

// 返回一个映射表，键为数字字符，值为对应的字母串

func initMp() map[rune]string {

return map[rune]string{

'0': "abc",

'1': "def",

'2': "ghi",

'3': "jkl",

'4': "mno",

'5': "pqr",

'6': "st",

'7': "uv",

'8': "wx",

'9': "yz",

}

}

// 深度优先搜索生成组合

// x: 当前递归深度

// now: 当前生成的字符串

// n: 原始字符串的长度

// s: 输入的数字字符串

// mp: 数字与字母映射表

// st: 禁止字符集合

// res: 存储结果的切片指针

func dfs(x int, now string, n int, s string, mp map[rune]string, st map[rune]bool, res \*[]string) {

if x == n {

// mk 用来记录当前组合中包含的禁止字符

mk := make(map[rune]bool)

for \_, char := range now {

if st[char] {

mk[char] = true

}

}

// 如果当前组合包含所有的禁止字符，则不添加到结果集

if len(mk) == len(st) {

return

}

\*res = append(\*res, now)

return

}

// 获取当前数字对应的字母串

tmp := mp[rune(s[x])]

// 遍历字母串，递归处理后续字符

for \_, char := range tmp {

dfs(x+1, now+string(char), n, s, mp, st, res)

}

}

func main() {

mp := initMp() // 初始化数字与字母的映射关系

scanner := bufio.NewScanner(os.Stdin)

for scanner.Scan() {

s := strings.TrimSpace(scanner.Text()) // 读取第一个输入字符串，去除首尾空格

scanner.Scan()

t := strings.TrimSpace(scanner.Text()) // 读取第二个输入字符串，去除首尾空格

// 将禁止字符存入集合中

st := make(map[rune]bool)

for \_, char := range t {

st[char] = true

}

res := make([]string, 0) // 用于存储结果的切片

// 进行深度优先搜索组合

dfs(0, "", len(s), s, mp, st, &res)

// 输出结果，使用逗号分隔

fmt.Println(strings.Join(res, ",") + ",")

}

}