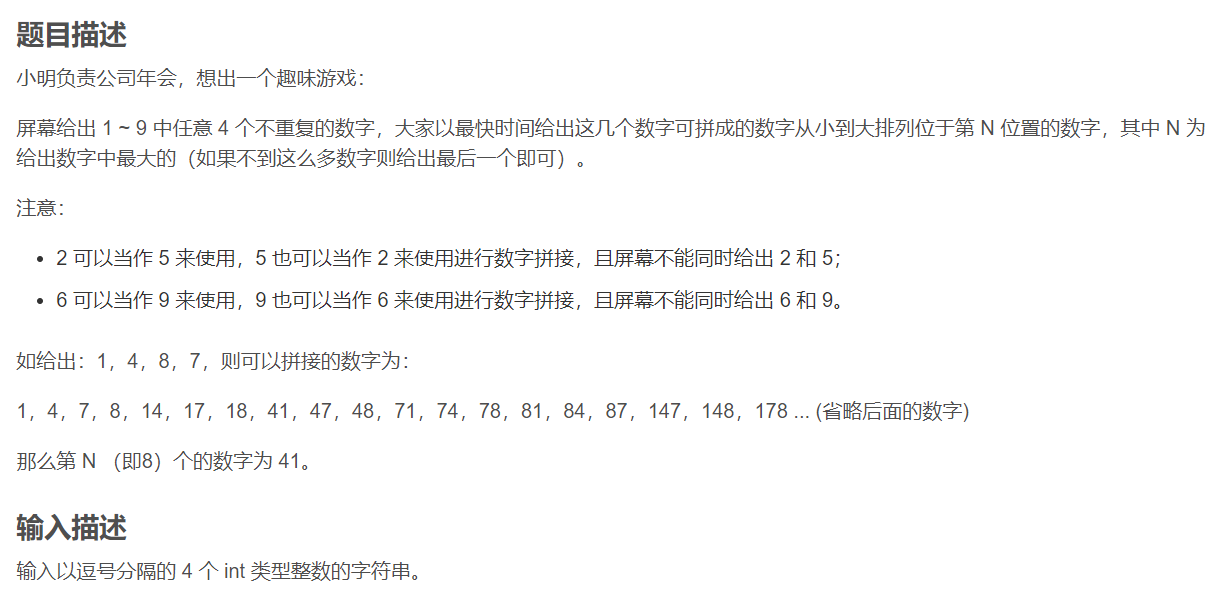
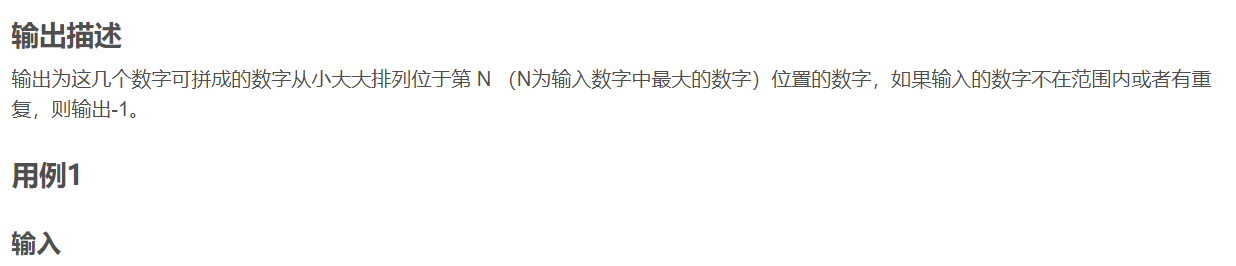
# **E卷-数字排列[200分]（ Java | Python3 | C++ | C语言 | JsNode | Go）**

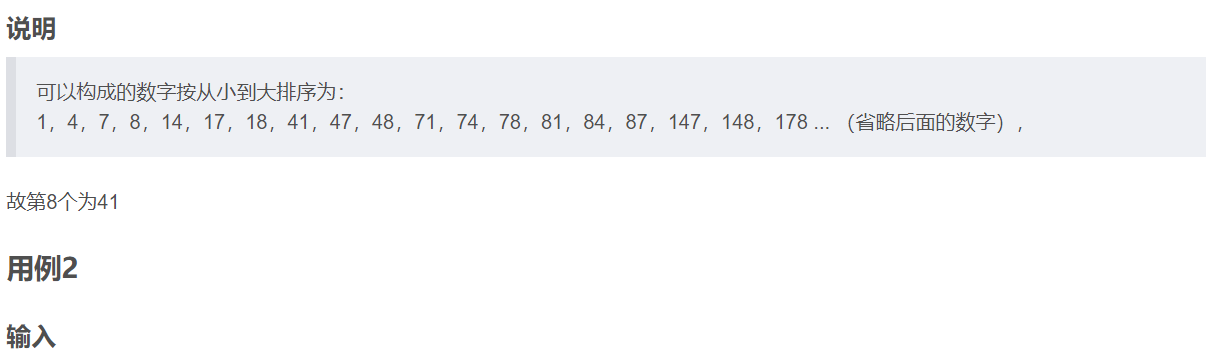




1,4,8,7



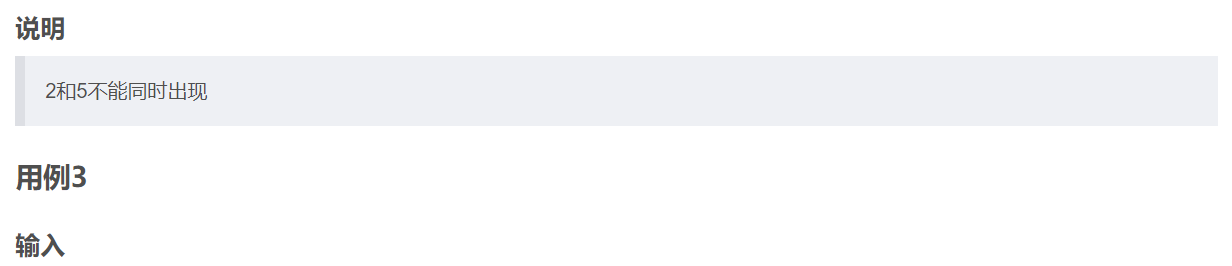
41



2,5,1



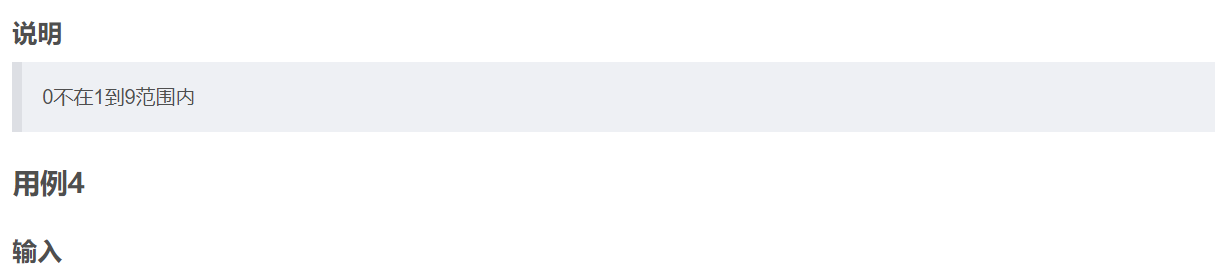
-1



3,0,9



-1

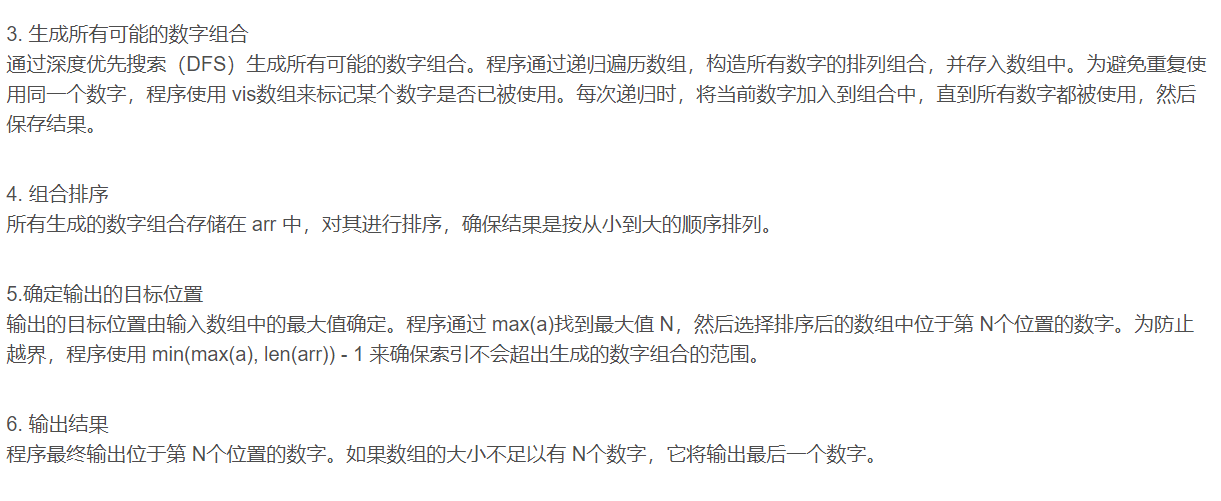


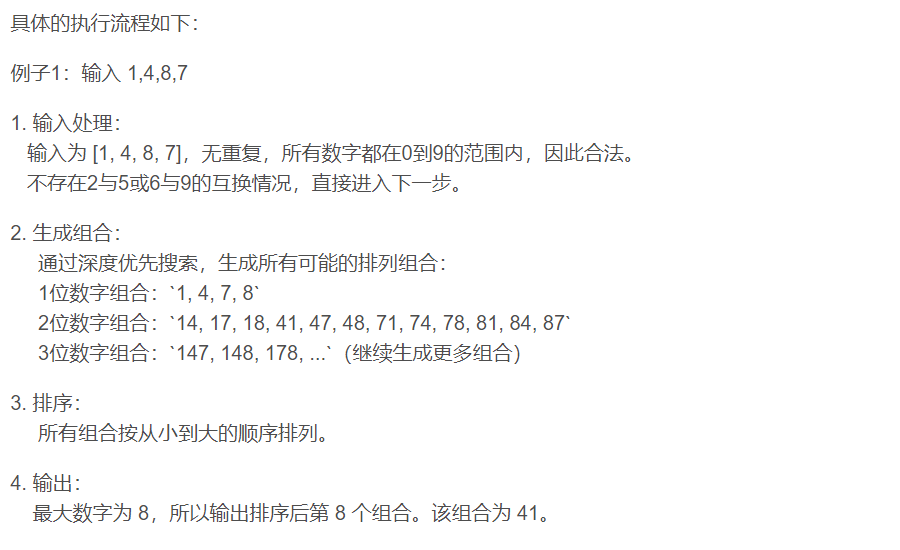
3,9,7,8

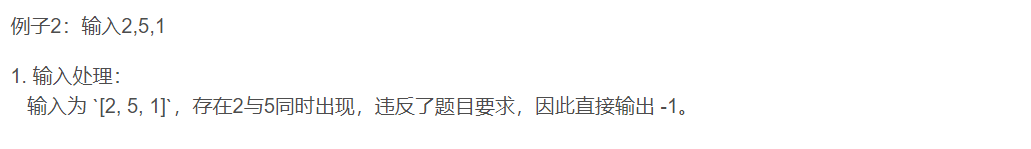


39











import java.util.\*; // 导入Java工具库

public class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in); // 创建扫描器从控制台读取输入

String[] input = sc.nextLine().split(","); // 按逗号分割输入

List<Integer> a = new

ArrayList<>(); // 创建一个整数列表存储输入的数字

// 将输入的字符串转换为整数并加入列表

for (String s : input) {

a.add(Integer.parseInt(s)); // 转换为整数并加入列表

}

Set<Integer> st = new HashSet<>(a); // 将列表转为集合，去重

if (st.size() !=

a.size()) { // 如果集合长度和列表长度不同，说明有重复元素

System.out.println(-1); // 输出-1，表示输入无效

return; // 结束程序

}

// 检查每个数字是否在0到9的范围内

for (int x : st) {

if (x < 0 || x > 9) {

System.out.println(-1); // 输出-1表示无效输入

return; // 结束程序

}

}

// 检查是否同时存在2和5

if (st.contains(2) && st.contains(5)) {

System.out.println(-1); // 输出-1表示无效输入

return; // 结束程序

}

// 检查是否同时存在6和9

if (st.contains(6) && st.contains(9)) {

System.out.println(-1); // 输出-1表示无效输入

return; // 结束程序

}

// 如果集合中有2，则添加5

if (st.contains(2)) a.add(5);

// 如果集合中有5，则添加2

if (st.contains(5)) a.add(2);

// 如果集合中有6，则添加9

if (st.contains(6)) a.add(9);

// 如果集合中有9，则添加6

if (st.contains(9)) a.add(6);

List<Integer> arr = new

ArrayList<>(); // 用于存储所有可能的数字组合

boolean[] vis = new boolean[a.size()]; // 用于标记数字是否被使用

// 深度优先搜索生成所有数字组合

dfs(0, 0, a, arr, vis);

Collections.sort(arr); // 对结果进行排序

int idx = Math.min(Collections.max(a), arr.size()) - 1; // 计算索引

System.out.println(arr.get(idx)); // 输出结果

}

// 深度优先搜索函数

public static void dfs(int idx, int x, List<Integer> a, List<Integer> arr,

boolean[] vis) {

if (x != 0) {

arr.add(x); // 将当前组合加入结果数组

}

if (idx == a.size())

return; // 如果已经遍历完所有数字，结束递归

// 遍历数组中的每个数字

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

if (!vis[i]) { // 如果当前数字没有被使用

vis[i] = true; // 标记数字已被使用

dfs(idx + 1, x \* 10 + a.get(i), a, arr, vis); // 递归调用

vis[i] = false; // 回溯，取消标记

}

}

}

}



# 将输入的字符串按照逗号分隔并转换为整数列表

a = list(map(int, input().split(",")))

# 将整数列表转换为集合，以检查是否有重复元素

st = set(a)

# 如果集合的长度与原数组的长度不同，说明有重复元素

if len(st) != len(a):

print(-1) # 输出-1表示无效输入

exit(0) # 结束程序

# 检查数组中的每个数字是否都在0到9的范围内

for x in st:

if x < 0 or x > 9: # 如果有不在范围内的数字

print(-1) # 输出-1表示无效输入

exit(0) # 结束程序

# 检查2和5是否同时存在，2和5不能同时存在

if 2 in st and 5 in st:

print(-1) # 输出-1表示无效输入

exit(0) # 结束程序

# 检查6和9是否同时存在，6和9不能同时存在

if 6 in st and 9 in st:

print(-1) # 输出-1表示无效输入

exit(0) # 结束程序

# 如果集合中有2，则可以将5也加入数组，因为2和5可以互换

if 2 in st:

a.append(5) # 追加5到数组

# 如果集合中有5，则可以将2也加入数组

if 5 in st:

a.append(2) # 追加2到数组

# 如果集合中有6，则可以将9也加入数组，因为6和9可以互换

if 6 in st:

a.append(9) # 追加9到数组

# 如果集合中有9，则可以将6也加入数组

if 9 in st:

a.append(6) # 追加6到数组

# 初始化一个空数组，用来存储所有可能的组合

arr = []

# 初始化一个标记数组，用来记录每个数字是否已被使用

vis = [0] \* len(a)

# 定义一个深度优先搜索函数，生成所有可能的数字组合

def dfs(idx, x):

# 如果x非空（即组合已经生成一部分），则将其加入arr数组

if x:

arr.append(x)

# 如果当前深度等于数组长度，说明递归结束，返回

if idx == len(a):

return

# 遍历数组中的每个元素

for i in range(len(a)):

# 如果当前元素没有被使用

if not vis[i]:

vis[i] = 1 # 标记该元素已被使用

dfs(idx + 1, x \* 10 + a[i]) # 递归调用，将当前元素加入组合

vis[i] = 0 # 回溯，取消标记

# 调用深度优先搜索函数，开始递归生成数字组合

dfs(0, 0)

# 对生成的数字组合进行排序

arr.sort()

# 计算第N个位置的数字，N为数组中的最大数字，同时确保索引不超出数组长度

idx = min(max(a), len(arr)) - 1

# 输出位于第N位置的数字

print(arr[idx])



#include <bits/stdc++.h>

using namespace std;

vector<int> arr; // 用于存储所有可能的组合

vector<int> a; // 存储输入的整数

bool vis[10]; // 标记数组，用于DFS中记录数字是否已被使用

// 深度优先搜索函数

void dfs(int idx, int x) {

if (x != 0) arr.push\_back(x); // 如果当前组合非空，将其存储

if (idx == a.size()) return; // 如果遍历完数组，结束递归

// 遍历数组

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

if (!vis[i]) { // 如果当前数字未被使用

vis[i] = true; // 标记已使用

dfs(idx + 1, x \* 10 + a[i]); // 递归调用，生成新组合

vis[i] = false; // 回溯，取消标记

}

}

}

int main() {

string input;

getline(cin, input); // 读取输入

stringstream ss(input); // 用于分割输入

string token;

while (getline(ss, token, ',')) {

a.push\_back(stoi(token)); // 分割并转换为整数

}

set<int> st(a.begin(), a.end()); // 去重检查

if (st.size() != a.size()) {

cout << -1 << endl; // 如果有重复，输出-1

return 0;

}

// 检查数字范围

for (int x : st) {

if (x < 0 || x > 9) {

cout << -1 << endl; // 如果超出范围，输出-1

return 0;

}

}

// 检查2和5

if (st.count(2) && st.count(5)) {

cout << -1 << endl;

return 0;

}

// 检查6和9

if (st.count(6) && st.count(9)) {

cout << -1 << endl;

return 0;

}

// 添加互换数字

if (st.count(2)) a.push\_back(5);

if (st.count(5)) a.push\_back(2);

if (st.count(6)) a.push\_back(9);

if (st.count(9)) a.push\_back(6);

dfs(0, 0); // 调用DFS函数

sort(arr.begin(), arr.end()); // 对结果排序

int idx = min(\*max\_element(a.begin(), a.end()),

(int)arr.size()) - 1; // 计算目标位置

cout << arr[idx] << endl; // 输出结果

return 0;

}



#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int min(int a,int b){

return a<b?a:b;

}

// 检查数组中是否有重复元素

int has\_duplicates(int\* arr, int len) {

for (int i = 0; i < len; i++) {

for (int j = i + 1; j < len; j++) {

if (arr[i] == arr[j]) {

return 1; // 如果有重复元素，返回1

}

}

}

return 0; // 否则返回0

}

// 检查是否所有数字都在0到9之间

int is\_valid\_range(int\* arr, int len) {

for (int i = 0; i < len; i++) {

if (arr[i] < 0 || arr[i] > 9) {

return 0; // 如果有数字超出范围，返回0

}

}

return 1; // 否则返回1

}

// 检查2和5是否同时存在，6和9是否同时存在

int check\_invalid\_combinations(int\* arr, int len) {

int has2 = 0, has5 = 0, has6 = 0, has9 = 0;

for (int i = 0; i < len; i++) {

if (arr[i] == 2) has2 = 1;

if (arr[i] == 5) has5 = 1;

if (arr[i] == 6) has6 = 1;

if (arr[i] == 9) has9 = 1;

}

// 如果2和5、6和9同时存在，返回0表示无效

if ((has2 && has5) || (has6 && has9)) return 0;

return 1; // 否则返回1表示有效

}

// 深度优先搜索生成所有组合

void dfs(int\* arr, int len, int\* vis, int idx, int x, int\* res, int\* res\_len) {

if (x > 0) {

res[(\*res\_len)++] = x; // 如果当前组合不为空，将其加入结果数组

}

if (idx == len) {

return; // 如果达到数组的长度，递归结束

}

for (int i = 0; i < len; i++) {

if (!vis[i]) {

vis[i] = 1; // 标记当前元素已被使用

dfs(arr, len, vis, idx + 1, x \* 10 + arr[i], res, res\_len);

vis[i] = 0; // 回溯，取消标记

}

}

}

int cmp(const void \*a, const void \*b) {

return (\*(int \*)a - \*(int \*)b);

}

int main() {

int a[100], n = 0, num;

char ch;

// 读取输入的数字并存储到数组a中

while (scanf("%d%c", &num, &ch)) {

a[n++] = num;

if (ch == '\n') break; // 读到换行符结束输入

}

// 检查是否有重复元素

if (has\_duplicates(a, n)) {

printf("-1\n");

return 0;

}

// 检查数字是否在0到9范围内

if (!is\_valid\_range(a, n)) {

printf("-1\n");

return 0;

}

// 检查无效组合（2和5，6和9不能同时存在）

if (!check\_invalid\_combinations(a, n)) {

printf("-1\n");

return 0;

}

int m=n;

// 如果存在2，将5也添加到数组中，反之亦然

for (int i = 0; i < m; i++) {

if (a[i] == 2) a[n++] = 5;

if (a[i] == 5) a[n++] = 2;

if (a[i] == 6) a[n++] = 9;

if (a[i] == 9) a[n++] = 6;

}

// 初始化结果数组和访问标记数组

int res[10000], res\_len = 0, vis[100] = {0};

// 调用深度优先搜索生成所有组合

dfs(a, n, vis, 0, 0, res, &res\_len);

// 对结果数组进行排序

qsort(res, res\_len, sizeof(int),cmp);

int mx=0;

for(int i=0;i<n;++i){

if(a[i]>mx){

mx=a[i];

}

}

// 计算N位置的索引

int idx = res[min(mx, res\_len) - 1];

// 输出位于第N位置的数字

printf("%d\n", idx);

return 0;

}



const readline = require("readline");

// 创建输入输出接口

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout,

});

let a = [];

// 读取输入

rl.on("line", (input) => {

// 将输入的字符串以逗号分隔并转换为整数数组

a = input.split(",").map(Number);

// 检查是否有重复元素

let st = new Set(a);

if (st.size !== a.length) {

console.log(-1); // 输出-1表示无效输入

rl.close();

return;

}

// 检查数组中每个数字是否都在0到9范围内

for (let x of st) {

if (x < 0 || x > 9) {

console.log(-1); // 输出-1表示无效输入

rl.close();

return;

}

}

// 检查2和5、6和9不能同时存在

if ((st.has(2) && st.has(5)) || (st.has(6) && st.has(9))) {

console.log(-1); // 输出-1表示无效输入

rl.close();

return;

}

// 如果集合中有2或5，互相加入数组

if (st.has(2)) a.push(5);

if (st.has(5)) a.push(2);

if (st.has(6)) a.push(9);

if (st.has(9)) a.push(6);

// 结果数组和访问标记数组

let arr = [],

vis = new Array(a.length).fill(0);

// 深度优先搜索生成所有组合

function dfs(idx, x) {

if (x !== 0) arr.push(x); // 如果x非空，加入结果数组

if (idx === a.length) return; // 递归结束条件

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

if (!vis[i]) {

vis[i] = 1; // 标记当前元素已使用

dfs(idx + 1, x \* 10 + a[i]); // 递归生成下一个组合

vis[i] = 0; // 回溯

}

}

}

// 开始深度优先搜索

dfs(0, 0);

// 对生成的数字组合排序

arr.sort((x, y) => x - y);

// 输出位于第N位置的数字

let idx = Math.min(Math.max(...a), arr.length) - 1;

console.log(arr[idx]);

rl.close();

});



package main

import (

"fmt" // 导入fmt包用于输入输出

"sort" // 导入sort包用于对生成的数字组合进行排序

"strconv"

"strings"

)

// 定义一个检查是否有重复元素的函数

func hasDuplicates(arr []int) bool {

// 使用一个map记录每个元素是否已经出现

set := make(map[int]bool)

for \_, v := range arr {

// 如果该元素已经存在于map中，说明有重复

if set[v] {

return true // 返回true表示有重复元素

}

set[v] = true // 标记该元素已出现

}

return false // 无重复元素返回false

}

// 定义一个检查数字是否在0到9范围内的函数

func isValidRange(arr []int) bool {

for \_, v := range arr {

// 如果有数字不在0到9的范围内，返回false

if v < 0 || v > 9 {

return false

}

}

return true // 如果所有数字都在范围内，返回true

}

// 定义一个检查无效组合的函数，2和5、6和9不能同时存在

func checkInvalidCombinations(arr []int) bool {

has2, has5, has6, has9 := false, false, false, false

for \_, v := range arr {

if v == 2 {

has2 = true

}

if v == 5 {

has5 = true

}

if v == 6 {

has6 = true

}

if v == 9 {

has9 = true

}

}

// 如果2和5同时存在，或者6和9同时存在，返回false表示无效组合

if (has2 && has5) || (has6 && has9) {

return false

}

return true // 否则返回true表示有效组合

}

// 定义深度优先搜索函数，用于生成所有数字组合

func dfs(arr []int, vis []bool, idx int, x int, res \*[]int) {

// 如果x不为0（即当前已生成部分组合），将其加入结果数组

if x != 0 {

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

}

// 如果当前深度达到数组长度，结束递归

if idx == len(arr) {

return

}

// 遍历数组中的每个元素

for i := 0; i < len(arr); i++ {

// 如果当前元素没有被使用

if !vis[i] {

vis[i] = true // 标记该元素已被使用

// 递归调用，将当前元素加入组合，进入下一层递归

dfs(arr, vis, idx+1, x\*10+arr[i], res)

vis[i] = false // 回溯，取消标记

}

}

}

func stringToArray(s string) []int {

var result []int

parts := strings.Split(s, ",")

for \_, part := range parts {

num, err := strconv.Atoi(part)

if err == nil {

result = append(result, num)

}

}

return result

}

func main() {

// 读取输入的数字并以逗号分隔存储在切片a中

var input string

fmt.Scan(&input)

a := stringToArray(input)

// 检查是否有重复元素

if hasDuplicates(a) {

fmt.Println(-1) // 输出-1表示无效输入

return

}

// 检查数字是否在0到9范围内

if !isValidRange(a) {

fmt.Println(-1) // 输出-1表示无效输入

return

}

// 检查无效组合（2和5、6和9不能同时存在）

if !checkInvalidCombinations(a) {

fmt.Println(-1) // 输出-1表示无效输入

return

}

// 如果存在2，将5也加入数组，反之亦然

c := len(a)

for i := 0; i < c; i++ {

if a[i] == 2 {

a = append(a, 5)

}

if a[i] == 5 {

a = append(a, 2)

}

if a[i] == 6 {

a = append(a, 9)

}

if a[i] == 9 {

a = append(a, 6)

}

}

// 初始化结果数组，用来存储所有可能的组合

var res []int

// 初始化访问标记数组，用来记录每个数字是否已被使用

vis := make([]bool, len(a))

// 调用深度优先搜索函数，开始递归生成数字组合

dfs(a, vis, 0, 0, &res)

// 对生成的数字组合进行排序

sort.Ints(res)

// 计算第N个位置的数字，N为数组中的最大数字，同时确保索引不超出数组长度

idx := min(max(a), len(res)) - 1

// 输出位于第N位置的数字

fmt.Println(res[idx])

}

// 定义一个返回数组中最大值的函数

func max(arr []int) int {

maxVal := arr[0]

for \_, v := range arr {

if v > maxVal {

maxVal = v

}

}

return maxVal

}

// 定义一个返回两个整数较小值的函数

func min(a, b int) int {

if a < b {

return a

}

return b

}