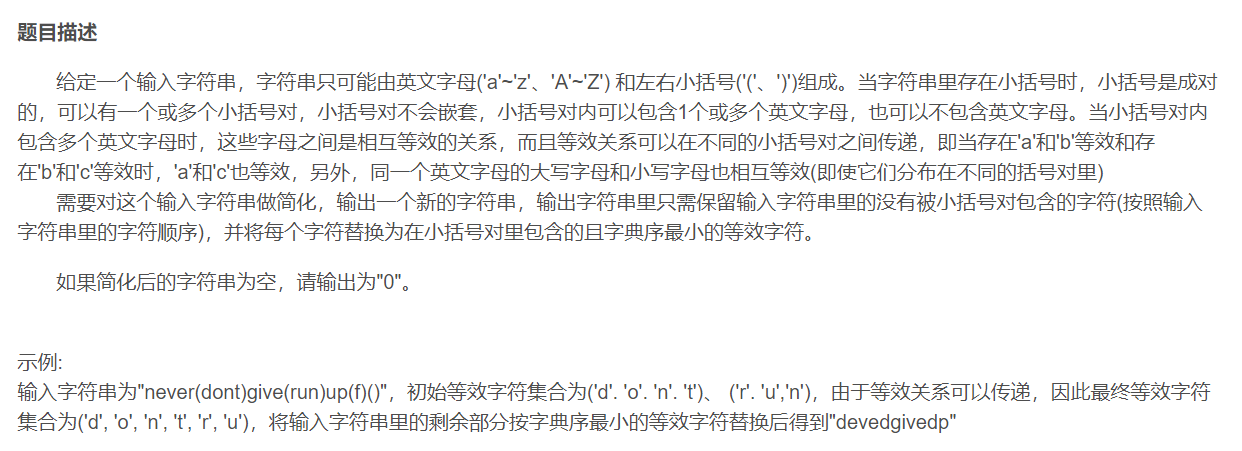
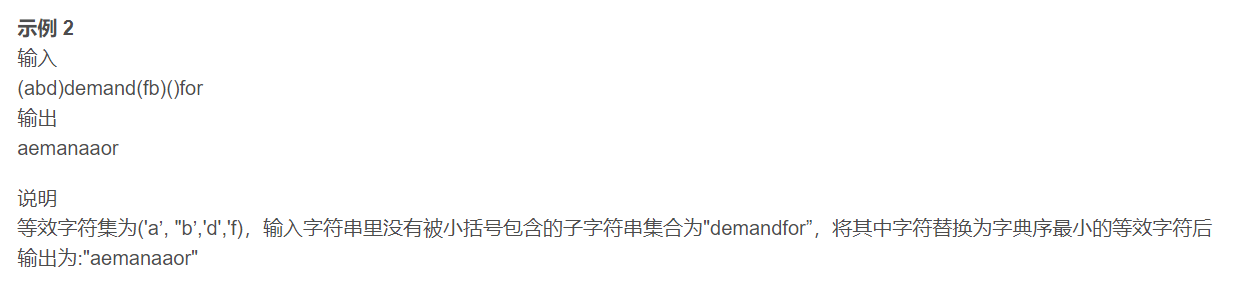
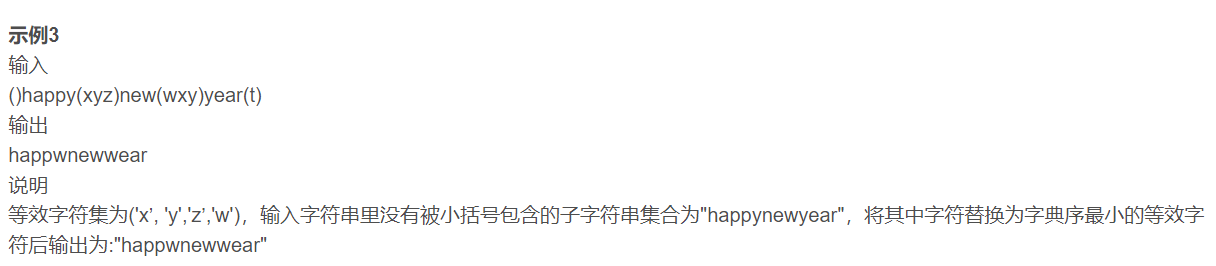
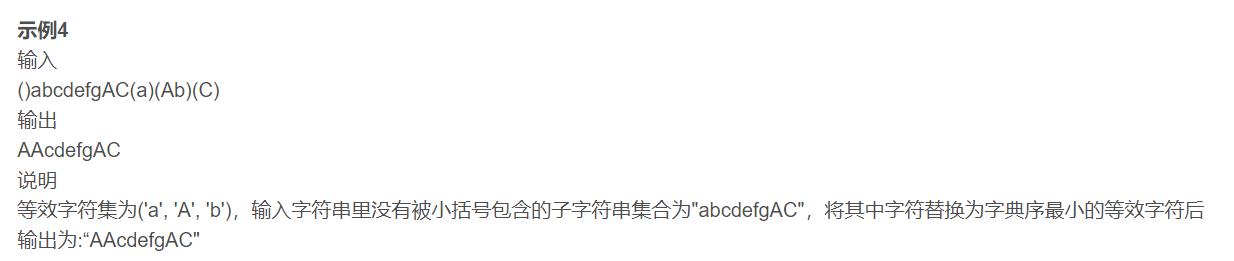
# **E卷-字符串化繁为简[200分]（ Java | Python3 | C++ | C语言 | JsNode | Go）**

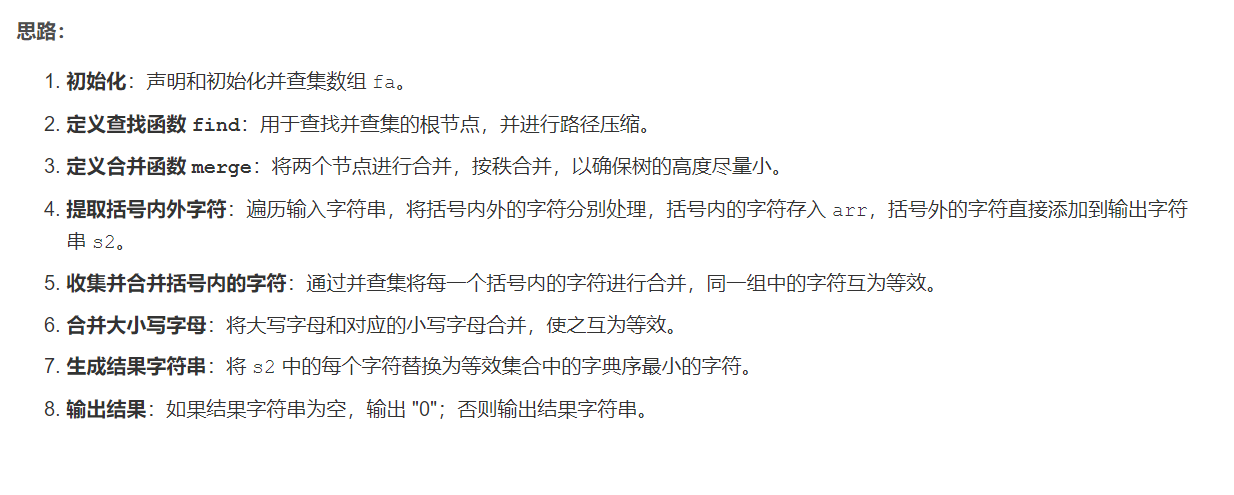














import java.util.\*;

public class Main {

public static int[] fa = new int[505]; // 并查集数组

// 并查集查找函数，带路径压缩

public static int find(int x) {

if (x == fa[x])

return x;

return fa[x] = find(fa[x]);

}

// 并查集合并函数，按秩合并

public static void merge(int x, int y) {

int f1 = find(x);

int f2 = find(y);

if (f1 > f2) {

int temp = f1;

f1 = f2;

f2 = temp;

}

fa[f2] = f1;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String s = scanner.nextLine();

for (int i = 0; i < 505; i++) {

fa[i] = i; // 初始化每个元素为自身的父节点

}

List<List<Character>> arr = new

ArrayList<>(); // 用于存储小括号中的字符组

StringBuilder s2 = new StringBuilder(); // 输出字符串

List<Character> same = new ArrayList<>(); // 暂存小括号内的字符

boolean f = false; // 标记是否在小括号内

// 遍历输入字符串，提取括号内外的字符

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

if (c == '(') {

f = true; // 进入小括号

} else if (c == ')') {

if (!same.isEmpty()) {

arr.add(new ArrayList<>(same)); // 将括号中的字符组存入arr

}

f = false; // 退出小括号

same.clear(); // 清空暂存字符组

} else {

if (f) {

same.add(c); // 在括号内，收集字符

} else {

s2.append(c); // 非括号内，直接添加到输出字符串

}

}

}

Set<Integer> st = new HashSet<>(); // 用于去重所有字符

for (List<Character> i : arr) { // 将所有括号内的字符加入集合

for (char j : i) {

st.add((int) j);

}

}

// 合并每组括号内的字符

for (List<Character> i : arr) {

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

merge(i.get(0), i.get(j)); // 将每组字符合并到一起

}

}

// 将大写字母和对应的小写字母合并

for (int i = 'A'; i <= 'Z'; i++) {

if (!st.contains(i))

continue;

int j = i - 'A' + 'a';

if (!st.contains(j))

continue;

merge(j, i);

}

StringBuilder res = new StringBuilder(); // 存储结果字符串

for (char i : s2.toString().toCharArray()) {

res.append((char) find(i)); // 替换为合并后的最小字典序字符

}

if (res.length() == 0) { // 如果结果字符串为空，输出0

System.out.println(0);

} else {

System.out.println(res.toString()); // 否则输出结果字符串

}

}

}



def main():

import sys

input = sys.stdin.read

s = input().strip()

fa = list(range(505)) # 并查集数组，最多505个元素

# 并查集查找函数，带路径压缩

def find(x):

if x == fa[x]:

return x

fa[x] = find(fa[x])

return fa[x]

# 并查集合并函数，按秩合并

def merge(x, y):

f1 = find(x)

f2 = find(y)

if f1 > f2:

f1, f2 = f2, f1

fa[f2] = f1

arr = [] # 用于存储小括号中的字符组

s2 = "" # 输出字符串

same = [] # 暂存小括号内的字符

f = False # 标记是否在小括号内

# 遍历输入字符串，提取括号内外的字符

for c in s:

if c == "(":

f = True # 进入小括号

elif c == ")":

if same:

arr.append(same.copy()) # 将括号中的字符组存入arr

f = False # 退出小括号

same.clear() # 清空暂存字符组

else:

if f:

same.append(c) # 在括号内，收集字符

else:

s2 += c # 非括号内，直接添加到输出字符串

st = set() # 用于去重所有字符

for i in arr: # 将所有括号内的字符加入集合

for j in i:

st.add(j)

# 合并每组括号内的字符

for i in arr:

for j in range(len(i)):

merge(ord(i[0]), ord(i[j])) # 将每组字符合并到一起

# 将大写字母和对应的小写书字母合并

for i in range(ord("A"), ord("Z") + 1):

if i not in st:

continue

j = i - ord("A") + ord("a")

if j not in st:

continue

merge(j, i)

res = "" # 存储结果字符串

for i in s2:

res += chr(find(ord(i))) # 替换为合并后的最小字典序字符

if not res: # 如果结果字符串为空，输出0

print(0)

else:

print(res) # 否则输出结果字符串

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

main()



#include <bits/stdc++.h>

using namespace std;

int main() {

string s;

getline(cin, s); // 读取输入字符串

vector<int> fa(505); // 并查集数组，最多512个元素

for (int i = 0; i < 505; i++) {

fa[i] = i; // 初始化每个元素为自身的父节点

}

// 并查集查找函数，带路径压缩

function<int(int)> find = [&](int x) {

if (x == fa[x])

return x;

return fa[x] = find(fa[x]);

};

// 并查集合并函数，按秩合并

auto merge = [&](int x, int y) {

int f1 = find(x);

int f2 = find(y);

if (f1 > f2)

swap(f1, f2); // 总是将较小的根作为父节点

fa[f2] = f1;

};

vector<vector<char>> arr; // 用于存储小括号中的字符组

string s2 = ""; // 输出字符串

vector<char> same; // 暂存小括号内的字符

bool f = false; // 标记是否在小括号内

// 遍历输入字符串，提取括号内外的字符

for (char c : s) {

if (c == '(') {

f = true; // 进入小括号

} else if (c == ')') {

if (!same.empty()) {

arr.push\_back(same); // 将括号中的字符组存入arr

}

f = false; // 退出小括号

same.clear(); // 清空暂存字符组

} else {

if (f) {

same.push\_back(c); // 在括号内，收集字符

} else {

s2 += c; // 非括号内，直接添加到输出字符串

}

}

}

set<int> st; // 用于去重所有字符

for (auto &i : arr) { // 将所有括号内的字符加入集合

for (auto &j : i) {

st.insert(j);

}

}

// 合并每组括号内的字符

for (auto &i : arr) {

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

merge(i[0], i[j]); // 将每组字符合并到一起

}

}

// 将大写字母和对应的小写字母合并

for (int i = 'A'; i <= 'Z'; i++) {

if (!st.count(i))

continue;

int j = i - 'A' + 'a';

if (!st.count(j))

continue;

merge(j, i);

}

string res = ""; // 存储结果字符串

for (char i : s2) {

res += char(find(i)); // 替换为合并后的最小字典序字符

}

if (res.empty()) { // 如果结果字符串为空，输出0

cout << 0;

} else {

cout << res; // 否则输出结果字符串

}

}



#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.h>

int fa[505]; // 并查集数组

// 并查集查找函数，带路径压缩

int find(int x) {

if (x == fa[x]) {

return x;

}

return fa[x] = find(fa[x]);

}

// 并查集合并函数，按秩合并

void merge(int x, int y) {

int f1 = find(x);

int f2 = find(y);

if (f1 > f2) {

int temp = f1;

f1 = f2;

f2 = temp;

}

fa[f2] = f1;

}

int main() {

char s[1005]; // 输入字符串，假设长度不会超过1000

fgets(s, 1005, stdin); // 读取输入字符串

for (int i = 0; i < 505; i++) {

fa[i] = i; // 初始化每个元素为自身的父节点

}

char arr[505][26]; // 用于存储小括号中的字符组，假设最多505个组，每组最多26个字符

int arr\_sizes[505]; // 用于存储每组的大小

int arr\_count = 0; // 当前字符组的数量

char s2[1005]; // 输出字符串

int s2\_len = 0;

bool f = false; // 标记是否在小括号内

int same\_len = 0; // 当前小括号内字符的数量

char same[26]; // 当前小括号内的字符，最多26个

// 遍历输入字符串，提取括号内外的字符

for (int i = 0; s[i] != '\0'; i++) {

char c = s[i];

if (c == '(') {

f = true; // 进入小括号

} else if (c == ')') {

if (same\_len > 0) {

memcpy(arr[arr\_count], same,

same\_len \* sizeof(char)); // 将暂存的字符组复制到数组中

arr\_sizes[arr\_count] = same\_len; // 存储当前字符组的大小

arr\_count++; // 字符组数目加1

}

same\_len = 0; // 清空暂存字符组

f = false; // 退出小括号

} else {

if (f) {

same[same\_len++] = c; // 在括号内，收集字符

} else {

s2[s2\_len++] = c; // 非括号内，直接添加到输出字符串

}

}

}

bool st[128] = {false}; // 用于去重所有字符

for (int i = 0; i < arr\_count; i++) { // 将所有括号内的字符加入集合

for (int j = 0; j < arr\_sizes[i]; j++) {

st[(int)arr[i][j]] = true;

}

}

// 合并每组括号内的字符

for (int i = 0; i < arr\_count; i++) {

for (int j = 0; j < arr\_sizes[i]; j++) {

merge(arr[i][0], arr[i][j]); // 将每组字符合并到一起

}

}

// 将大写字母和对应的小写字母合并

for (int i = 'A'; i <= 'Z'; i++) {

if (!st[i]) {

continue;

}

int j = i - 'A' + 'a';

if (!st[j]) {

continue;

}

merge(j, i);

}

char res[1005]; // 存储结果字符串

int res\_len = 0;

for (int i = 0; i < s2\_len; i++) {

res[res\_len++] = (char)find(

s2[i]); // 替换为合并后的最小字典序字符

}

res[res\_len] = '\0'; // 添加字符串结束符

if (res\_len == 0) { // 如果结果字符串为空，输出0

printf("0\n");

} else {

printf("%s\n", res); // 否则输出结果字符串

}

return 0;

}



const readline = require("readline");

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout,

});

rl.question("", (str) => {

const s = str.trim();

const fa = Array.from({ length: 505 }, (\_, i) => i); // 并查集数组，初始化为自身的父节点

// 并查集查找函数，带路径压缩

function find(x) {

if (x === fa[x]) return x;

return (fa[x] = find(fa[x]));

}

// 并查集合并函数，按秩合并

function merge(x, y) {

const f1 = find(x);

const f2 = find(y);

if (f1 > f2) {

const temp = f1;

f1 = f2;

f2 = temp;

}

fa[f2] = f1;

}

const arr = []; // 用于存储小括号中的字符组

let s2 = ""; // 输出字符串

let same = []; // 暂存小括号内的字符

let inBrackets = false; // 标记是否在小括号内

// 遍历输入字符串，提取括号内外的字符

for (const c of s) {

if (c === "(") {

inBrackets = true; // 进入小括号

} else if (c === ")") {

if (same.length) {

arr.push(same.slice()); // 将括号中的字符组存入arr

}

inBrackets = false; // 退出小括号

same = []; // 清空暂存字符组

} else {

if (inBrackets) {

same.push(c); // 在括号内，收集字符

} else {

s2 += c; // 非括号内，直接添加到输出字符串

}

}

}

const st = new Set(); // 用于去重所有字符

for (const grp of arr) {

// 将所有括号内的字符加入集合

for (const char of grp) {

st.add(char.charCodeAt(0));

}

}

// 合并每组括号内的字符

for (const grp of arr) {

for (const char of grp) {

merge(grp[0].charCodeAt(0), char.charCodeAt(0)); // 将每组字符合并到一起

}

}

// 将大写字母和对应的小写字母合并

for (let i = 65; i <= 90; i++) {

// 'A' 到 'Z'

if (!st.has(i)) continue;

const j = i - 65 + 97; // 对应的小写字母

if (!st.has(j)) continue;

merge(j, i);

}

let res = ""; // 存储结果字符串

for (const char of s2) {

res += String.fromCharCode(find(char.charCodeAt(0))); // 替换为合并后的最小字典序字符

}

if (res === "") {

// 如果结果字符串为空，输出0

console.log("0");

} else {

console.log(res); // 否则输出结果字符串

}

rl.close();

});



package main

import (

"bufio"

"fmt"

"os"

)

func find(fa []int, x int) int {

if x == fa[x] {

return x

}

fa[x] = find(fa, fa[x])

return fa[x]

}

func merge(fa []int, x, y int) {

f1 := find(fa, x)

f2 := find(fa, y)

if f1 > f2 {

f1, f2 = f2, f1

}

fa[f2] = f1

}

func main() {

reader := bufio.NewReader(os.Stdin)

s, \_ := reader.ReadString('\n')

s = s[:len(s)-1] // 去掉末尾的换行符

fa := make([]int, 505)

for i := range fa {

fa[i] = i // 初始化并查集

}

var arr [][]rune // 用于存储小括号中的字符组

var s2 []rune // 输出字符串

var same []rune // 暂存小括号内的字符

inBrackets := false // 标记是否在小括号内

// 遍历输入字符串，提取括号内外的字符

for \_, c := range s {

if c == '(' {

inBrackets = true // 进入小括号

} else if c == ')' {

if len(same) > 0 {

arr = append(arr, append([]rune(nil), same...)) // 将括号中的字符组存入arr

same = same[:0] // 清空暂存字符组

}

inBrackets = false // 退出小括号

} else {

if inBrackets {

same = append(same, c) // 在括号内，收集字符

} else {

s2 = append(s2, c) // 非括号内，直接添加到输出字符串

}

}

}

st := make(map[rune]bool) // 去重所有括号内的字符

for \_, group := range arr {

for \_, c := range group {

st[c] = true

}

}

// 合并每组括号内的字符

for \_, group := range arr {

for \_, c := range group {

merge(fa, int(group[0]), int(c)) // 将每组字符合并到一起

}

}

// 将大写字母和对应的小写字母合并

for i := 'A'; i <= 'Z'; i++ {

if !st[i] {

continue

}

j := i - 'A' + 'a'

if !st[j] {

continue

}

merge(fa, int(j), int(i))

}

res := []rune{} // 存储结果字符串

for \_, c := range s2 {

res = append(res, rune(find(fa, int(c)))) // 替换为合并后的最小字典序字符

}

if len(res) == 0 {

fmt.Println("0") // 如果结果字符串为空，输出0

} else {

fmt.Println(string(res)) // 否则输出结果字符串

}

}