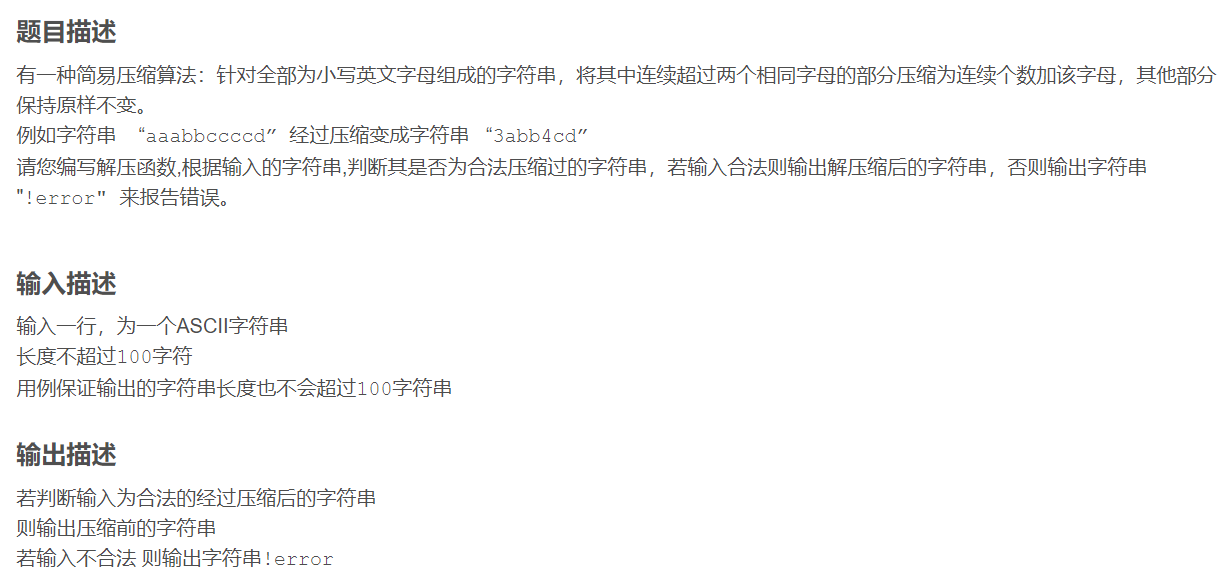
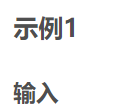
# **E卷-简易压缩算法[100分]（ Java | Python3 | C++ | C语言 | JsNode | Go）**





4dff



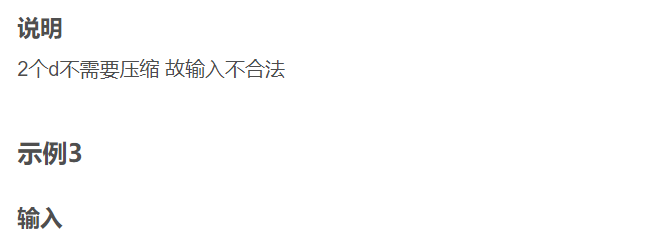
Ddddff



2dff



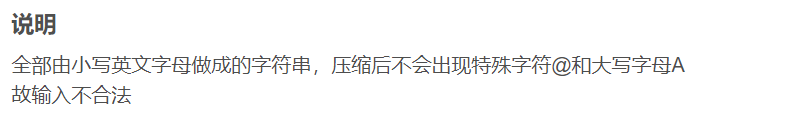
!error



4d@A



!error





import java.util.Scanner;

public class Main {

/\*\*

\* 编码函数，用于将字符串进行压缩处理

\* @param s 输入的原始字符串

\* @return 压缩后的字符串

\*/

public static String encode(String s) {

int i = 0;

StringBuilder res = new StringBuilder();

// 遍历整个字符串

while (i < s.length()) {

char c = s.charAt(i);

int cnt = 1;

int j = i + 1;

// 计算连续相同字符的数量

while (j < s.length() && s.charAt(j) == c) {

cnt++;

j++;

}

// 如果连续字符数量为2，则不进行压缩

if (cnt == 2) {

res.append(c).append(c);

} else if (cnt == 1) {

res.append(c);

} else {

res.append(cnt).append(c);

}

i = j;

}

return res.toString();

}

/\*\*

\* 解码函数，用于将压缩后的字符串解压缩成原始字符串

\* @param s 压缩后的字符串

\* @return 解压后的字符串

\*/

public static String decode(String s) {

int i = 0;

StringBuilder res = new StringBuilder();

// 遍历整个字符串

while (i < s.length()) {

// 判断当前字符是否为数字

if (Character.isDigit(s.charAt(i))) {

int now = s.charAt(i) - '0';

int j = i + 1;

// 处理多位数字的情况

while (j < s.length() && Character.isDigit(s.charAt(j))) {

now = now \* 10 + (s.charAt(j) - '0');

j++;

}

char c = 0;

// 获取数字后面的字符

if (j < s.length()) {

c = s.charAt(j);

}

i = j + 1;

// 根据数字重复相应的字符

for (int k = 0; k < now; k++) {

res.append(c);

}

} else {

// 如果不是数字，直接添加到结果中

res.append(s.charAt(i));

i++;

}

}

return res.toString();

}

/\*\*

\* 验证函数，用于检查输入字符串是否合法

\* @param s 输入的字符串

\* @return 合法则返回true，不合法则返回false

\*/

public static boolean valid(String s) {

// 检查每个字符是否为数字或小写字母

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

if (!Character.isDigit(c) && !Character.isLowerCase(c)) {

return false;

}

}

return true;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// 获取用户输入

String s = scanner.nextLine();

// 首先验证输入是否合法

if (!valid(s)) {

System.out.println("!error");

return;

}

// 解压缩输入字符串

String t = decode(s);

// 验证解压后的字符串再次压缩是否与原字符串一致

if (!encode(t).equals(s)) {

System.out.println("!error");

} else {

System.out.println(t);

}

}

}



def encode(s):

"""

编码函数，用于将字符串进行压缩处理

:param s: 输入的原始字符串

:return: 压缩后的字符串

"""

i = 0

res = ""

while i < len(s):

c = s[i]

cnt = 1

j = i + 1

# 计算连续相同字符的数量

while j < len(s) and s[j] == c:

cnt += 1

j += 1

# 如果连续字符数量为2，则保持原样

if cnt == 2:

res += c + c

# 如果连续字符数量为1，则保持原样

elif cnt == 1:

res += c

# 如果连续字符数量超过2，则进行压缩

else:

res += str(cnt) + c

i = j

return res

def decode(s):

"""

解码函数，用于将压缩后的字符串解压缩成原始字符串

:param s: 压缩后的字符串

:return: 解压后的字符串

"""

i = 0

res = ""

while i < len(s):

# 判断当前字符是否为数字

if s[i].isdigit():

now = int(s[i])

j = i + 1

# 处理多位数字的情况

while j < len(s) and s[j].isdigit():

now = now \* 10 + int(s[j])

j += 1

c = ""

# 获取数字后面的字母

if j < len(s):

c = s[j]

i = j + 1

# 根据数字重复相应的字母

res += c \* now

else:

# 如果不是数字，直接添加到结果中

res += s[i]

i += 1

return res

def valid(s):

"""

验证函数，用于检查输入字符串是否合法

:param s: 输入的字符串

:return: 合法则返回True，不合法则返回False

"""

for c in s:

# 检查字符是否为数字或小写字母

if not (c.isdigit() or c.islower()):

return False

return True

# 获取用户输入

s = input()

# 首先验证输入是否合法

if not valid(s):

print("!error")

exit(0)

# 解压缩输入字符串

t = decode(s)

# 验证解压后的字符串再次压缩是否与原字符串一致

if encode(t) != s:

print("!error")

else:

print(t)



#include <iostream>

#include <string>

using namespace std;

// 编码函数，用于将字符串进行压缩处理

string encode(const string& s) {

int i = 0;

string res = "";

while (i < s.length()) {

char c = s[i]; // 获取当前字符

int cnt = 1; // 初始化连续字符计数

int j = i + 1;

// 计算连续相同字符的数量

while (j < s.length() && s[j] == c) {

cnt++;

j++;

}

// 根据计数值决定如何处理结果

if (cnt == 2) {

res += c; // 如果连续字符数量为2，则保持原样

res += c;

} else if (cnt == 1) {

res += c; // 如果只有一个字符，也保持原样

} else {

res += to\_string(cnt) + c; // 如果超过2个字符，进行压缩处理

}

i = j;

}

return res;

}

// 解码函数，用于将压缩后的字符串解压缩成原始字符串

string decode(const string& s) {

int i = 0;

string res = "";

while (i < s.length()) {

if (isdigit(s[i])) {

int now = s[i] - '0'; // 把数字字符转换为整数

int j = i + 1;

// 处理多位数字的情况

while (j < s.length() && isdigit(s[j])) {

now = now \* 10 + (s[j] - '0');

j++;

}

char c = '\0';

// 获取数字后面的字母

if (j < s.length()) {

c = s[j];

}

i = j + 1;

// 根据数字重复相应的字母

res.append(now, c);

} else {

// 如果不是数字，直接添加到结果中

res += s[i];

i++;

}

}

return res;

}

// 验证函数，用于检查输入字符串是否合法

bool valid(const string& s) {

for (char c : s) {

// 检查字符是否为数字或小写字母

if (!isdigit(c) && !islower(c)) {

return false;

}

}

return true;

}

int main() {

string s;

cin >> s;

// 首先验证输入是否合法

if (!valid(s)) {

cout << "!error" << endl;

return 0;

}

// 解压缩输入字符串

string t = decode(s);

// 验证解压后的字符串再次压缩是否与原字符串一致

if (encode(t) != s) {

cout << "!error" << endl;

} else {

cout << t << endl;

}

return 0;

}



#include <stdio.h>

#include <string.h>

#include <ctype.h>

// 编码函数，用于将字符串进行压缩处理

void encode(const char\* s, char\* res) {

int i = 0, k = 0;

while (s[i] != '\0') {

char c = s[i]; // 获取当前字符

int cnt = 1; // 初始化连续字符计数

int j = i + 1;

// 计算连续相同字符的数量

while (s[j] != '\0' && s[j] == c) {

cnt++;

j++;

}

// 根据计数值决定如何处理结果

if (cnt == 2) {

res[k++] = c;

res[k++] = c; // 如果连续字符数量为2，则保持原样

} else if (cnt == 1) {

res[k++] = c; // 如果只有一个字符，也保持原样

} else {

k += sprintf(&res[k], "%d",

cnt); // 如果超过2个字符，进行压缩处理

res[k++] = c;

}

i = j;

}

res[k] = '\0'; // 给结果字符串添加终止符

}

// 解码函数，用于将压缩后的字符串解压缩成原始字符串

void decode(const char\* s, char\* res) {

int i = 0, k = 0;

while (s[i] != '\0') {

if (isdigit(s[i])) {

int now = s[i] - '0'; // 把数字字符转换为整数

int j = i + 1;

// 处理多位数字的情况

while (s[j] != '\0' && isdigit(s[j])) {

now = now \* 10 + (s[j] - '0');

j++;

}

char c = '\0';

// 获取数字后面的字母

if (s[j] != '\0') {

c = s[j];

}

i = j + 1;

// 根据数字重复相应的字母

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

res[k++] = c;

}

} else {

// 如果不是数字，直接添加到结果中

res[k++] = s[i];

i++;

}

}

res[k] = '\0'; // 给结果字符串添加终止符

}

// 验证函数，用于检查输入字符串是否合法

int valid(const char\* s) {

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

// 检查字符是否为数字或小写字母

if (!isdigit(s[i]) && !islower(s[i])) {

return 0;

}

}

return 1;

}

int main() {

char s[101], t[101], encoded[101];

// 获取用户输入

fgets(s, 101, stdin);

s[strcspn(s, "\n")] = 0; // 去掉换行符

// 首先验证输入是否合法

if (!valid(s)) {

printf("!error\n");

return 0;

}

// 解压缩输入字符串

decode(s, t);

// 验证解压后的字符串再次压缩是否与原字符串一致

encode(t, encoded);

if (strcmp(encoded, s) != 0) {

printf("!error\n");

} else {

printf("%s\n", t);

}

return 0;

}



const readline = require("readline");

// 创建接口以读取命令行输入

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout,

});

// 编码函数，用于将字符串进行压缩处理

const encode = (s) => {

let i = 0;

let res = "";

while (i < s.length) {

const c = s[i]; // 获取当前字符

let cnt = 1; // 初始化连续字符计数

let j = i + 1;

// 计算连续相同字符的数量

while (j < s.length && s[j] === c) {

cnt++;

j++;

}

// 根据计数值决定如何处理结果

if (cnt === 2) {

res += c + c; // 如果连续字符数量为2，则保持原样

} else if (cnt === 1) {

res += c; // 如果只有一个字符，也保持原样

} else {

res += cnt + c; // 如果超过2个字符，进行压缩处理

}

i = j;

}

return res;

};

// 解码函数，用于将压缩后的字符串解压缩成原始字符串

const decode = (s) => {

let i = 0;

let res = "";

while (i < s.length) {

if (/\d/.test(s[i])) {

let now = parseInt(s[i], 10); // 把数字字符转换为整数

let j = i + 1;

// 处理多位数字的情况

while (j < s.length && /\d/.test(s[j])) {

now = now \* 10 + parseInt(s[j], 10);

j++;

}

let c = "";

// 获取数字后面的字母

if (j < s.length) {

c = s[j];

}

i = j + 1;

// 根据数字重复相应的字母

res += c.repeat(now);

} else {

// 如果不是数字，直接添加到结果中

res += s[i];

i++;

}

}

return res;

};

// 验证函数，用于检查输入字符串是否合法

const valid = (s) => {

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

// 检查字符是否为数字或小写字母

if (!/\d/.test(s[i]) && !/[a-z]/.test(s[i])) {

return false;

}

}

return true;

};

// 读取用户输入的字符串

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

s = s.trim(); // 去除首尾空白字符

// 首先验证输入是否合法

if (!valid(s)) {

console.log("!error");

rl.close();

return;

}

// 解压缩输入字符串

const t = decode(s);

// 验证解压后的字符串再次压缩是否与原字符串一致

if (encode(t) !== s) {

console.log("!error");

} else {

console.log(t);

}

rl.close();

});



package main

import (

"bufio"

"fmt"

"os"

"strconv"

"strings"

)

// 编码函数，用于将字符串进行压缩处理

func encode(s string) string {

i := 0

var res strings.Builder

for i < len(s) {

c := s[i] // 获取当前字符

cnt := 1 // 初始化连续字符计数

j := i + 1

// 计算连续相同字符的数量

for j < len(s) && s[j] == c {

cnt++

j++

}

// 根据计数值决定如何处理结果

if cnt == 2 {

res.WriteString(string(c) + string(c)) // 如果连续字符数量为2，则保持原样

} else if cnt == 1 {

res.WriteByte(c) // 如果只有一个字符，也保持原样

} else {

res.WriteString(strconv.Itoa(cnt))

res.WriteByte(c) // 如果超过2个字符，进行压缩处理

}

i = j

}

return res.String()

}

// 解码函数，用于将压缩后的字符串解压缩成原始字符串

func decode(s string) string {

i := 0

var res strings.Builder

for i < len(s) {

if s[i] >= '0' && s[i] <= '9' {

now := int(s[i] - '0') // 把数字字符转换为整数

j := i + 1

// 处理多位数字的情况

for j < len(s) && s[j] >= '0' && s[j] <= '9' {

now = now\*10 + int(s[j]-'0')

j++

}

var c byte

// 获取数字后面的字母

if j < len(s) {

c = s[j]

}

i = j + 1

// 根据数字重复相应的字母

for k := 0; k < now; k++ {

res.WriteByte(c)

}

} else {

// 如果不是数字，直接添加到结果中

res.WriteByte(s[i])

i++

}

}

return res.String()

}

// 验证函数，用于检查输入字符串是否合法

func valid(s string) bool {

for \_, c := range s {

// 检查字符是否为数字或小写字母

if !(c >= '0' && c <= '9') && !(c >= 'a' && c <= 'z') {

return false

}

}

return true

}

func main() {

reader := bufio.NewReader(os.Stdin)

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

s = strings.TrimSpace(s) // 去掉换行符

// 首先验证输入是否合法

if !valid(s) {

fmt.Println("!error")

return

}

// 解压缩输入字符串

t := decode(s)

// 验证解压后的字符串再次压缩是否与原字符串一致

if encode(t) != s {

fmt.Println("!error")

} else {

fmt.Println(t)

}

}