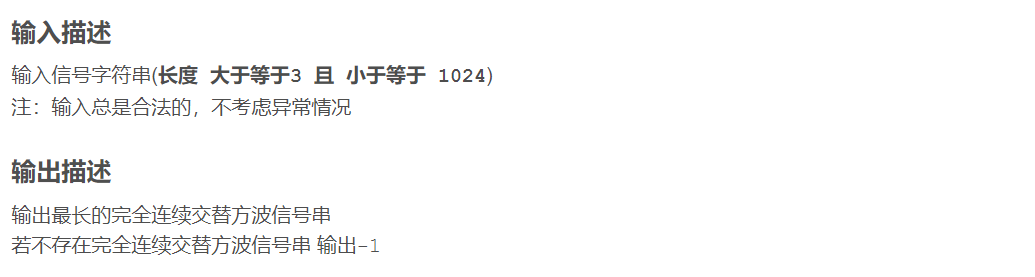
# **E卷-最长连续交替方波信号[200分]（ Java | Python3 | C++ | C语言 | JsNode | Go）**





输入

0010101010110000101000010

输出

01010



import java.util.Scanner;

public class Main {

public static void main(String[] args) {

// 读取用户输入的一行字符串 s

Scanner scanner = new Scanner(System.in);

String s = scanner.nextLine();

// 去除字符串两端的空白字符

s = s.trim();

// 获取字符串 s 的长度

int n = s.length();

// 初始化最大长度为 0

int MaxLen = 0;

// 遍历字符串的每一个字符

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

// 初始化标志位 flag，用于标识是否遇到连续的 "11"

int flag = 0;

// 如果当前字符是 '0'

if (s.charAt(i) == '0') {

int j = i + 1; // 设置 j 为 i 的下一个位置

// 向后遍历字符串，寻找最长的符合条件的子串

while (j < n) {

// 如果连续遇到两个 '0'，则退出循环，因为此时子串不符合要求

if (s.charAt(j) == '0' && s.charAt(j - 1) == '0') {

break;

}

// 如果连续遇到两个 '1'，则设置标志位 flag 为 1

if (s.charAt(j) == '1' && s.charAt(j - 1) == '1') {

flag = 1;

}

j++; // 继续向后遍历

}

// 如果找到的子串长度大于等于 3 且未遇到连续的 "11"

if (j - i >= 3 && flag == 0) {

// 更新最大长度 MaxLen

if (MaxLen < (j - i)) {

MaxLen = j - i;

}

}

// 将 i 移动到子串的最后一个位置

i = j - 1;

}

}

// 如果没有找到符合条件的子串，输出 -1

if (MaxLen == 0) {

System.out.println("-1");

return;

}

// 构建符合条件的最长子串

StringBuilder ans = new StringBuilder();

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

// 按照 "010101..." 的模式生成子串

if (i % 2 == 0) {

ans.append('0');

} else {

ans.append('1');

}

}

// 输出结果

System.out.println(ans.toString());

scanner.close();

}

}



# 读取用户输入的一行字符串 s

s = input()

s = s.strip() # 去除换行符等空白字符

# 获取字符串 s 的长度

n = len(s)

# 初始化最大长度为 0

MaxLen = 0

# 遍历字符串的每一个字符

for i in range(n):

# 初始化标志位 flag，用于标识是否遇到连续的 "11"

flag = 0

# 如果当前字符是 '0'

if s[i] == '0':

j = i + 1 # 设置 j 为 i 的下一个位置

# 向后遍历字符串，寻找最长的符合条件的子串

while j < n:

# 如果连续遇到两个 '0'，则退出循环，因为此时子串不符合要求

if s[j] == '0' and s[j - 1] == '0':

break

# 如果连续遇到两个 '1'，则设置标志位 flag 为 1

if s[j] == '1' and s[j - 1] == '1':

flag = 1

j += 1 # 继续向后遍历

# 如果找到的子串长度大于等于 3 且未遇到连续的 "11"

if j - i >= 3 and flag == 0:

# 更新最大长度 MaxLen

if MaxLen < (j - i):

MaxLen = j - i

# 将 i 移动到子串的最后一个位置

i = j - 1

# 如果没有找到符合条件的子串，输出 -1

if MaxLen == 0:

print("-1")

exit(0)

# 构建符合条件的最长子串

ans = ""

for i in range(MaxLen):

# 按照 "010101..." 的模式生成子串

if i % 2 == 0:

ans += "0"

else:

ans += "1"

# 输出结果

print(ans)



#include <iostream>

#include <string>

int main() {

// 读取用户输入的一行字符串 s

std::string s;

std::getline(std::cin, s);

// 去除字符串两端的空白字符

s.erase(0, s.find\_first\_not\_of(" \n\r\t"));

s.erase(s.find\_last\_not\_of(" \n\r\t") + 1);

// 获取字符串 s 的长度

int n = s.length();

// 初始化最大长度为 0

int MaxLen = 0;

// 遍历字符串的每一个字符

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

// 初始化标志位 flag，用于标识是否遇到连续的 "11"

int flag = 0;

// 如果当前字符是 '0'

if (s[i] == '0') {

int j = i + 1; // 设置 j 为 i 的下一个位置

// 向后遍历字符串，寻找最长的符合条件的子串

while (j < n) {

// 如果连续遇到两个 '0'，则退出循环，因为此时子串不符合要求

if (s[j] == '0' && s[j - 1] == '0') {

break;

}

// 如果连续遇到两个 '1'，则设置标志位 flag 为 1

if (s[j] == '1' && s[j - 1] == '1') {

flag = 1;

}

j++; // 继续向后遍历

}

// 如果找到的子串长度大于等于 3 且未遇到连续的 "11"

if (j - i >= 3 && flag == 0) {

// 更新最大长度 MaxLen

if (MaxLen < (j - i)) {

MaxLen = j - i;

}

}

// 将 i 移动到子串的最后一个位置

i = j - 1;

}

}

// 如果没有找到符合条件的子串，输出 -1

if (MaxLen == 0) {

std::cout << "-1" << std::endl;

return 0;

}

// 构建符合条件的最长子串

std::string ans = "";

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

// 按照 "010101..." 的模式生成子串

if (i % 2 == 0) {

ans += "0";

} else {

ans += "1";

}

}

// 输出结果

std::cout << ans << std::endl;

return 0;

}



#include <stdio.h>

#include <string.h>

int main() {

// 读取用户输入的一行字符串 s

char s[100005];

fgets(s, sizeof(s), stdin);

// 获取字符串 s 的长度

int n = strlen(s);

// 初始化最大长度为 0

int MaxLen = 0;

// 遍历字符串的每一个字符

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

// 初始化标志位 flag，用于标识是否遇到连续的 "11"

int flag = 0;

// 如果当前字符是 '0'

if (s[i] == '0') {

int j = i + 1; // 设置 j 为 i 的下一个位置

// 向后遍历字符串，寻找最长的符合条件的子串

while (j < n) {

// 如果连续遇到两个 '0'，则退出循环，因为此时子串不符合要求

if (s[j] == '0' && s[j - 1] == '0') {

break;

}

// 如果连续遇到两个 '1'，则设置标志位 flag 为 1

if (s[j] == '1' && s[j - 1] == '1') {

flag = 1;

}

j++; // 继续向后遍历

}

// 如果找到的子串长度大于等于 3 且未遇到连续的 "11"

if (j - i >= 3 && flag == 0) {

// 更新最大长度 MaxLen

if (MaxLen < (j - i)) {

MaxLen = j - i;

}

}

// 将 i 移动到子串的最后一个位置

i = j - 1;

}

}

// 如果没有找到符合条件的子串，输出 -1

if (MaxLen == 0) {

printf("-1\n");

return 0;

}

// 构建符合条件的最长子串

char ans[MaxLen + 1];

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

// 按照 "010101..." 的模式生成子串

if (i % 2 == 0) {

ans[i] = '0';

} else {

ans[i] = '1';

}

}

ans[MaxLen] = '\0'; // 加上字符串结尾的空字符

// 输出结果

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

return 0;

}



// 引入 Node.js 内置的 readline 模块，用于从标准输入输出获取数据。

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

input: process.stdin, // 标准输入

output: process.stdout, // 标准输出

});

// 获取 readline 接口的异步迭代器，以便在异步函数中使用。

var iter = rl[Symbol.asyncIterator]();

// 创建一个异步函数，用于读取用户输入的一行数据。

const readline = async () => (await iter.next()).value;

// 使用立即调用的异步函数表达式（IIFE）来运行异步代码。

void async function () {

// 读取用户输入的一行字符串 s

let s = (await readline());

// 获取字符串 s 的长度

let n = s.length;

// 初始化最大长度为 0

let MaxLen = 0;

// 遍历字符串的每一个字符

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

// 初始化标志位 flag，用于标识是否遇到连续的 "11"

let flag = 0;

// 如果当前字符是 '0'

if(s[i] == '0') {

let j = i + 1; // 设置 j 为 i 的下一个位置

// 向后遍历字符串，寻找最长的符合条件的子串

while(j < n) {

// 如果连续遇到两个 '0'，则退出循环，因为此时子串不符合要求

if(s[j] == '0' && s[j - 1] == '0') break;

// 如果连续遇到两个 '1'，则设置标志位 flag 为 1

if(s[j] == '1' && s[j - 1] == '1') flag = 1;

j++; // 继续向后遍历

}

// 如果找到的子串长度大于等于 3 且未遇到连续的 "11"

if(j - i >= 3 && flag == 0) {

// 更新最大长度 MaxLen

if(MaxLen < j - i) MaxLen = j - i;

}

// 将 i 移动到子串的最后一个位置

i = j - 1;

}

}

// 如果没有找到符合条件的子串，输出 -1

if(MaxLen == 0) {

console.log("-1");

return;

}

// 构建符合条件的最长子串

var ans = "";

for(let i = 0; i < MaxLen; i++) {

// 按照 "010101..." 的模式生成子串

if(i % 2 == 0) ans += "0";

else ans += "1";

}

// 输出结果

console.log(ans);

}();

// 00101010101100001010010

// 01010



package main

import (

"bufio"

"fmt"

"os"

"strings"

)

func main() {

// 使用bufio读取标准输入

reader := bufio.NewReader(os.Stdin)

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

s = strings.TrimSpace(s) // 去除换行符等空白字符

// 获取字符串s的长度

n := len(s)

// 初始化最大长度为0

MaxLen := 0

// 遍历字符串的每一个字符

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

// 初始化标志位flag，用于标识是否遇到连续的"11"

flag := 0

// 如果当前字符是'0'

if s[i] == '0' {

j := i + 1 // 设置j为i的下一个位置

// 向后遍历字符串，寻找最长的符合条件的子串

for j < n {

// 如果连续遇到两个'0'，则退出循环，因为此时子串不符合要求

if s[j] == '0' && s[j-1] == '0' {

break

}

// 如果连续遇到两个'1'，则设置标志位flag为1

if s[j] == '1' && s[j-1] == '1' {

flag = 1

}

j++ // 继续向后遍历

}

// 如果找到的子串长度大于等于3且未遇到连续的"11"

if j-i >= 3 && flag == 0 {

// 更新最大长度MaxLen

if MaxLen < j-i {

MaxLen = j - i

}

}

// 将i移动到子串的最后一个位置

i = j - 1

}

}

// 如果没有找到符合条件的子串，输出-1

if MaxLen == 0 {

fmt.Println("-1")

return

}

// 构建符合条件的最长子串

var ans strings.Builder

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

// 按照"010101..."的模式生成子串

if i%2 == 0 {

ans.WriteString("0")

} else {

ans.WriteString("1")

}

}

// 输出结果

fmt.Println(ans.String())

}