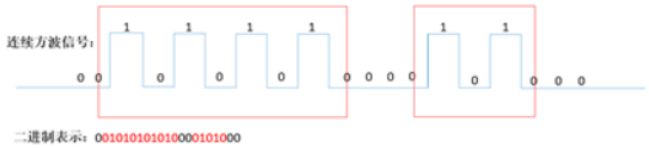
41、最长的完全交替连接方波信号，考点 or 实现——数据结构/栈

题目描述

输入一串方波信号，求取最长的完全连续交替方波信号，并将其输出，

如果有相同长度的交替方波信号，输出任一即可，

方波信号高位用1标识，低位用0标识，如图：



说明：

1. 一个完整的信号一定以0开始然后以0结尾，即010是一个完整信号，但101，1010，0101不是
2. 输入的一串方波信号是由一个或多个完整信号组成
3. 两个相邻信号之间可能有0个或多个低位，如0110010，011000010
4. 同一个信号中可以有连续的高位，如01110101011110001010，前14位是一个具有连续高位的信号
5. 完全连续交替方波是指10交替，如01010是完全连续交替方波，0110不是

输入描述

输入信号字符串（长度 >= 3 且 <= 1024）：
00101010101100001010000010
注：输入总是合法的，不用考虑异常情况

输出描述

输出最长的完全连续交替方波信号串：01010
若不存在完全连续交替方波信号串，输出 -1。

用例

输入	00101010101100001010010
输出	01010
说明	输入信号串中有三个信号： 0 010101010110(第一个信号段) 00 01010(第二个信号段) 010(第三个信号段) 第一个信号虽然有交替的方波信号段，但出现了11部分的连续高位，不算完全连续交替方波， 在剩下的连续方波信号串中01010最长

题目解析

本题要求“最长的完全交替连续方波信号”，

其中形成“完全交替连续方波信号”由两个要点：

- “完全交替连续方波”：完全连续交替方波是指10交替
- “信号”：一个完整的信号一定以0开始然后以0结尾

因此，“完全交替连续方波信号”，可以有两种描述方式：

1. 一个以0开头，之后有一个或多个10的字符串，用正则表示的话，即为 ^0(10)+$
2. 开头是一个或多个01，结尾是0的字符串，用正则表示的话，即为 ^(01)+0$

下面代码我们用了第二种描述方式的正则来判断一个字符串是否为“完全交替连续方波信号”

题目中还描述：

> 输入的一串方波信号是由一个或多个完整信号组成

> 两个相邻信号之间可能有0个或多个低位

因此，当遇到相邻的两个0时，就是出现的两个信号（输入总是合法的，不用考虑异常情况）

另外，输入中的信号，不仅有可能是“完全交替信号”，也可能是“不完全交替信号”，如0110，是一个完整信号，但是是非完全交替信号。

因此，我的解题思路如下，定义一个栈stack，然后遍历输入字符串的字符c：

- 如果stack为空，则c直接压入stack
- 如果stack不为空，假设stack栈顶字符为top

1. 如果c == '0'，且 top == '0'，则说明c可能是新信号组成，而stack中已保存的字符组成的字符串可能是一个完全交替信号，我们用之前的正则来验证即可，如果验证OK，则记录此完全交替信号的长度，和本身。完成记录后，清空stack，记录新信号的c。
2. 如果c == '0'，而 top == '1'，则c直接压入stack
3. 如果c == '1' ，则c直接压入satck

Java算法源码

```java
import java.util.Scanner;
import java.util.regex.Pattern;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    String s = sc.nextLine();

    System.out.println(getResult(s));
  }

  public static String getResult(String s) {
    Pattern reg = Pattern.compile("^(01)+0$");

    int maxLen = 0;
    String ans = "-1";

    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < s.length(); i++) {
      char c = s.charAt(i);

      if (c == '0') {
        if (sb.length() > 0 && sb.charAt(sb.length() - 1) == '0') {
          if (reg.matcher(sb.toString()).find() && sb.length() > maxLen) {
            maxLen = sb.length();
            ans = sb.toString();
          }
          sb = new StringBuilder();
        }
      }

      sb.append(c);
    }

    if (sb.length() > 0) {
      if (reg.matcher(sb.toString()).find() && sb.length() > maxLen) {
        return sb.toString();
      }
    }

    return ans;
  }
}
```

JS算法源码

```
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

rl.on("line", (line) => {
  console.log(getResult(line));
});

function getResult(s) {
  const reg = /^(01)+0$/;

  let maxLen = 0;
  let ans = "-1";

  const stack = [];

  for (let c of s) {
    if (c == "0") {
      const len = stack.length;
      if (len > 0 && stack.at(-1) == "0") {
        const str = stack.join("");
        if (reg.test(str) && len > maxLen) {
          maxLen = len;
```

```
          ans = str;
        }
        stack.length = 0;
      }
    }

    stack.push(c);
  }

  if (stack.length > 0) {
    const str = stack.join("");
    if (reg.test(str) && stack.length > maxLen) {
      return stack.join("");
    }
  }

  return ans;
}
```

Python算法源码

```
import re

# 输入获取
s = input()


# 算法入口
def getResult():
    reg = re.compile(r"^(01)+0$")

    maxLen = 0
    ans = "-1"

    stack = []

    for c in s:
        if c == "0":
            if len(stack) > 0 and stack[-1] == "0":
                tmp = "".join(stack)
                if reg.match(tmp) is not None and len(stack) > maxLen:
                    maxLen = len(stack)
                    ans = tmp
                stack.clear()
        stack.append(c)

    if len(stack) > 0:
```

```
    if len(stack) > 0:
        tmp = "".join(stack)
        if reg.match(tmp) is not None and len(stack) > maxLen:
            return tmp

    return ans


# 算法调用
print(getResult())
```