题目描述

现有一字符串仅由‘(’，‘)’，‘{’，‘}’，‘[’，‘]’六种括号组成。

若字符串满足以下条件之一，则为无效字符串：

①任一类型的左右括号数量不相等；

②存在未按正确顺序（先左后右）闭合的括号。

输出括号的最大嵌套深度[?]，若字符串无效则输出0。

0<字符串长度<100000

输入描述

一个只包括‘(’，‘)’，‘{’，‘}’，‘[’，‘]’的字符串

输出描述

一个整数，最大的括号深度

用例

输入	[]
输出	1
说明	有效字符串，最大嵌套深度为1

输入	([]{})
输出	3
说明	有效字符串，最大嵌套深度为3

输入	{]
输出	0
说明	无效字符串，有两种类型的左右括号数量不相等

输入	((){
输出	0
说明	无效字符串，存在未按正确顺序闭合的括号

输入	){
输出	0
说明	无效字符串，存在未按正确顺序闭合的括号。

题目解析

本题可以通过栈结构来判断括号的对称性。

本题的难点在于最大深度计算，其实也不难，就是看是否形成了连续出栈

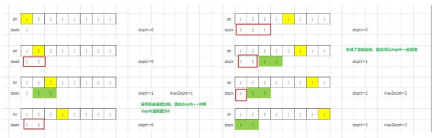
代码实现就是看入栈的一直是‘[’，‘{’，‘(’，则意味着必然要出栈一对括号，则深度一直自增。如果入栈的是‘)’，‘}’，‘]’，则意味着入栈元素不会和栈顶元素形成一队括号，也就不会出栈，此时深度重置为0

2023.05.21 根据网友指正，上面求解最大深度有问题，我们下应该将连续出栈次数作为最大深度，这是有漏洞的。比如用例：

```
({[]})
```

其实，本题在stack栈中，一旦遇到闭合的括号就会弹出，因此stack栈中能保留的只是左括号，而我们可以直接将stack栈中保留的左括号的最大数量作为最大深度，而stack栈的本身size就是对应的左括号数量。

比如下面图示中高红框的地方



JavaScript算法源码

```
// 注意类名必须为 Main, 不要有任何 package xxx 信息
const readline = require("readline");

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

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

function getMaxDepth(str) {
  const map = {
    ")": "(",
    "]": "[",
    "}": "{",
  };

  const stack = [];

  let maxDepth = 0;
  for (let i = 0; i < str.length; i++) {
    let c = str[i];

    if (stack.length > 0 && map[c] === stack.at(-1)) {
      stack.pop();
    } else {
      stack.push(c);
      maxDepth = Math.max(maxDepth, stack.length);
    }
  }

  if (stack.length) return 0;
  return maxDepth;
}
```

Java算法源码

```
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Scanner;

public class Main {
  // 输入获取
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    String s = sc.nextLine();
    System.out.println(getResult(s));
  }

  // 算法入口
  public static int getResult(String s) {
    HashMap<Character, Character> map = new HashMap<>();
    map.put(')', '(');
    map.put(']', '[');
    map.put('}', '{');

    LinkedList<Character> stack = new LinkedList<>();

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

      if (stack.size() > 0 && map.get(c) == stack.getLast()) {
        stack.removeLast();
      } else {
        stack.add(c);
        maxDepth = Math.max(maxDepth, stack.size());
      }
    }

    if (stack.size() > 0) return 0;
    return maxDepth;
  }
}
```

Python算法源码

```
# 输入获取
s = input()


# 算法入口
def getResult():
    map = {
        ")": "(",
        "]": "[",
        "}": "{"
    }

    stack = []

    maxDepth = 0
    for i in range(len(s)):
        c = s[i]

        if len(stack) > 0 and map.get(c) is not None and map[c] == stack[-1]:
            stack.pop()
        else:
            stack.append(c)
            maxDepth = max(maxDepth, len(stack))

    if len(stack) > 0:
        return 0

    return maxDepth


# 算法调用
print(getResult())
```