# 题目

一个只包含括号的字符串，判断该字符串所包含的括号匹配是否合法。

类似题目**：**Leetcode 921

# 分析

## 方法一：栈

1、首先对空字符串和仅含有一个字符的字符串做判断

2、如果字符串中出现左侧括号"{"，"("，"["入栈

3、如果字符串中出现右侧括号"}"，")"，"]"则与该字符前面的字符判断(即栈顶元素)，看是否能够配对，如果可以配对则将栈顶元素弹出，否则返回false

4、整个字符串判断完成后，如果最后栈为空，则表示全部配对，否则返回false

**代码：**

#include <iostream>

#include <vector>

#include <string>

#include <stack>

using namespace std;

/\*

判断一个括号字符串组成的序列是否合法

使用栈的思想 ,最后判断栈是否为空

\*/

bool isValidParenthess(string& str)

{

//首先对空等特殊情况判断

if(str.empty()) //note:需要先对输入判空，如果为空根据需求判断为true

return true;

if(1 == str.size())

//note：输入大小为1的情况需要特殊判断一下，这样平均执行时间大大提升

return false;

stack<char> st; //note:定义的是栈stack而不是vector

int i;

char tmp;

//开始借助栈这个数据结构的特性来遍历整个字符串

for(i=0;i<str.length();i++) //note:使用str.size()和str.length()都可以

{

if(str[i]=='(' || str[i]=='[' || str[i]=='{')

st.push(str[i]); //note:不是push\_back，是push

else if(str[i]==']' && !st.empty()) //note：判空不可少，否则进入函数core

{

tmp = st.top(); //现阶段判断的就是跟前一个元素比较，即栈顶元素

if(tmp != '[')

return false;

else

st.pop();

}

else if(str[i] ==')' && !st.empty())

{

tmp = st.top();

if(tmp != '(')

return false;

else

st.pop();

}

else if(str[i]=='}' && !st.empty()) //不要忘记栈判空

{

tmp = st.top();

if(tmp != '{')

return false;

else

st.pop();

}

else

return false; //note:这一步不能少，对于”]”这样的输入判断一下false

}

if(st.empty())

return true;

else

return false;

}

**复杂度分析：**

时间复杂度：O(n)，因为我们一次只遍历给定的字符串中的一个字符并在栈上进行O(1)的推入和弹出操作。

空间复杂度：O(n)，当我们将所有的开括号都推到栈上时以及在最糟糕的情况下，我们最终要把所有括号推到栈上。例如((((((((((。

或：

class Solution {

public:

bool isValid(string s) {

if(s.empty())

return true;

if(s.length()%2==1) //所有奇数个的字符串均返回false

return false;

stack<char> stk;

char tmp;

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

{

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

{

stk.push(s[i]);

}

else if(s[i] == ')' && !stk.empty())

{

tmp = stk.top();

if(tmp == '(')

{

stk.pop();

}

else

{

return false;

}

}

else if(s[i] == '}' && !stk.empty())

{

tmp = stk.top();

if(tmp == '{')

{

stk.pop();

}

else

{

return false;

}

}

else if(s[i] == ']' && !stk.empty())

{

tmp = stk.top();

if(tmp == '[')

{

stk.pop();

}

else

{

return false;

}

}

else

{

return false;

}

}

if(!stk.empty())

return false;

return true;

}

};

# 类似题目

**注：Leetcode20、921类似**

P21：给定一个只包括“（”，“）”，“{”，“}”，“[”，“]”的字符串，判断字符串是否有效。

有效字符串需满足：

1. 左括号必须用相同类型的右括号闭合。
2. 左括号必须以正确的顺序闭合。

注意：空字符串可以被认为是有效字符串。

分析：如果字符串中的字符是左括号（(/[/{），则入栈，如果是右括号，则判断是否与栈顶元素相等，如果相等则栈顶元素出栈，表示存在一对匹配的括号，如果不相等则不合法。

这里是匹配多种括号，**如果只匹配一种，则不需要使用栈（空间复杂度为O(n)），而使用一个变量标识是否匹配（这样空间复杂度为O(1)）**。

Java代码实现：

public class Stack{

public boolean isValid(String s) {

char[] stack = new char[s.;ength()];

int size = 0;

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

char c = s.charAt(i);

if(c=='(' || c=='{' || c=='[') { //如果是左括号则入栈

stack[size++] = c;

}else if(size>0 && c-stack[size-1]<2 && c-stack[size-1]>0) {

//如果是右括号则判断是否与栈顶元素匹配(注意空栈)

//查看ASCII码表可知码值相差1、2为左右括号

size--;

}

}

}

}

**L921：使括号有效的最少添加**

给定一个由“(”和“)”括号组成的字符串S，我们需要添加最少的括号(“(”或是“)”，可以在任何位置，以使得到的括号字符串有效。

从形式上讲，只有满足下面几点之一，括号字符串才是有效的：

1. 它是一个空字符串或者
2. 它可以被写成AB（A与B连接），其中A和B都是有效字符串，或者
3. 它可以被写成（A），其中A是有效字符串

给定一个括号字符串，返回为使结果字符串有效而必须添加的最少括号数。

分析：遇到左括号，则放入栈，遇到右括号，则检查是否与栈顶元素匹配，如果匹配则出栈，不匹配继续。最后，栈的长度就是最后额外需要匹配的括号数。