Given a string containing just the characters '(' and ')', find the length of the longest valid (well-formed) parentheses substring.

For "(()", the longest valid parentheses substring is "()", which has length = 2.

Another example is ")()())", where the longest valid parentheses substring is "()()", which has length = 4.

题目：

给定一个只有括号的字符串，判断该字符串中最长的有效括号。

注意：这里是最长的有效括号的意思是，有效的括号的子字符串长度，不是有效括号的个数。

例如：）（）（（（）） 这个字符串的最长的有效括号是4 🡪（（））

这个字符串的有效括号个数为6（题目上说一对括号算2）

（1）借用栈，来做括号匹配。

当栈为空的时候，无论字符为左括号还有右括号就直接入栈。进行下一个括号判断。

记录有几个有效括号：

当栈不为空的时候，当遇到一个）的时候，取栈顶元素进行匹配。匹配成功，则弹出栈顶的（，计数器+2.当遇到（的时候直接入栈等待匹配。

代码

public static int validParentheses(String s) {

int counter = 0;

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

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

if(stack.isEmpty()){ //当前为空栈，直接入栈

stack.push(s.charAt(i));

continue; //记住continue很重要，空栈的时候，无论是（ 还是 ）入栈，并不做以下的判断

}

if(s.charAt(i)=='(') //( 压入栈中，等待匹配

stack.push(s.charAt(i));

if(s.charAt(i)==')'){ //当 ） 的时候，栈顶元素进行匹配，成功则 （ 出栈

if(stack.peek()=='('){ //匹配成功

counter+=2;

stack.pop(); //弹出栈顶元素

}

}

}

return counter;

}

但是题目要求的是最长的有效括号的子字符串长度。

我们这里就需要对这种办法进行修改。我们要是知道每个括号的位置就好了，这样匹配成功的时候，用右括号的位置减去左括号的位置正好等于长度。那么我们如何记录括号的位置呢？

我们这里把入栈的括号变成括号的位置。同样是左括号入栈，但是入的是该括号的位置，然后我们通过s.charAt(i)还能够找到该括号。同样还是右括号匹配，首先右边括号的位置就是当前遍历的位置是知道的，然后于栈顶元素进行匹配，因为栈记录的是位置信息，所以只需要通过位置找到左括号进行匹配就行了，匹配成功肯定是里右括号最近的那个没被匹配的左括号，用右括号的位置减去左括号的位置然后+1就是该匹配成功的字符串的长度。然后于记录的长度对比，大于就替换。

代码：

public static int longestValidParenttheses(String s){

int maxLength=0;

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

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

if(stack.isEmpty()){ //栈为空

stack.push(i); //当前元素位置i直接入栈

continue; //跳过匹配，下一个元素再进行匹配

}

if(s.charAt(i)=='(')// （ 直接把位置存入栈中

stack.push(i);

if(s.charAt(i)==')'){ // ） 弹出栈顶元素

Integer j=stack.pop(); //记录当前弹出元素的位置

maxLength=Math.max(maxLength,i-j+1);//当前匹配成功的括号位置差i-j+1,为当前括号的有效长度

//与之前记录进行对比，长则替换。

}

}

return maxLength;

}

（2）不使用外部空间

要想得出最长的有效括号长度，就要知道最开始的左括号是否能匹配成功

例如：

1 2 3 4 5 6 7

（ ） （ （ （ ） ）

我们用lcounter记录左括号的个数，rcounter记录右括号的个数

只要lcounter=rcounter，说明最开始的左括号匹配成功

我们从左往右开始遍历匹配，当然从左往右匹配的是左括号

首先1是最开始的左括号，lcounter=0+1=1

然后2是右括号，rcounter=0+1=1

此时lcounter=rcounter，说明1位置的括号匹配成功，length=rounter\*2=2

再次开始

3变成了最开始的左括号，lcounter=1+1=2，后边4，5也是lcounter=2+2=4

6开始是右括号，7也是右括号，rcounter=1+2=3.

4>3,说明3这个最远的左括号位置匹配失败

然后我们换个方向，匹配右括号

最远的右括号位置是7，然后开始遍历计数匹配

当遍历到4的时候，rcounter=lcounter，匹配成功，length=lcounter\*2=4

然后继续匹配

这里注意到的是3是下一轮开始匹配的括号，但是跟右括号反着，直接就匹配失败，因为反着的括号就不可能匹配成功，所以直接从0开始从新匹配。那么怎么判断呢？因为我们这次反向遍历匹配的是右括号，需要的是左括号，只有当右边括号多的时候才是需要匹配的状态，当左括号多了，说明之前有的位置已经匹配完毕，出现了根本就不需要匹配的左括号。所以当lcounter>rcounter的时候就全部清0从新开始匹配。从左到右也同理，左括号少于右括号的时候，需要匹配，当右括号多余左括号的时候不需要匹配。

代码如下：

public static int longestValidParentthesesNoSpace(String s){

int lcounter = 0;

int rcounter = 0;

int maxLength = 0;

for(int i=0;i<s.length();i++){ //从左往右，左括号匹配右括号

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

lcounter++;

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

rcounter++;

if(lcounter==rcounter) //最远未匹配成功的左边括号个数等于右边括号个数的时候，匹配成功

maxLength=Math.max(maxLength,rcounter\*2);//记录长度

if(rcounter>lcounter){//当右括号个数大于左括号个数的时候说明当前位置不需要匹配

lcounter=0; //清零从新开始匹配

rcounter=0;

}

}

lcounter=rcounter=0; //从右往左，右括号匹配左括号

for(int i=s.length()-1;i>=0;i--){

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

lcounter++;

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

rcounter++;

if(lcounter==rcounter)

maxLength=Math.max(maxLength,lcounter\*2);

if(lcounter>rcounter){

lcounter=0;

rcounter=0;

}

}

return maxLength;

}

体会：当以后我们做到括号匹配的时候，可以用额外空间的时候，我们优先想到使用栈，可以方便的匹配好任何一个括号。当需要连续的匹配成功括号的时候，我们换个方式来匹配，记录位置，通过位置来检索信息，然后进行匹配。这样还能够输出相关的位置信息