LeetCode Training Day 15 Valid Parentheses

Valid parenthess is a common string problem. The string contains some parentheses such as "(()))", if it is a valid parenthese string then '(' and ')' should match. For example the following string is valid parenthese, "()()" or "(())". The following string is invalid parenthese, such as ")(", "(()))" or ")()()".

There are two common solutions to detect the valid parentheses. One is using stack, when you see '(', you push to stack, when you see ')', you pop up from stack. By the end if the stack is empty and the stack is NOTE underflow (which means you try to pop up '(' but nothing left in stack) during process. Then it is a valid parentheses string. Another is scan from left to right when you see '(', you add count by 1, when you see ')' you deduct count by 1, any time you see count is negative you reset it to zero and claim it is invalid parentheses, after you scan from left to right, you scan from right to left again.

In some problems you are asked to fix the invalid parentheses by adding or removing some parentheses, any time you see scanning count as negative, you can put a fix and reset the scanning count. If you use stack, you can do at when stack is underflow or there is some partheses left in stack.

## 20. Valid Parentheses

Easy

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.

**Example 1:**

**Input:** s = "()"

**Output:** true

**Example 2:**

**Input:** s = "()[]{}"

**Output:** true

**Example 3:**

**Input:** s = "(]"

**Output:** false

**Constraints:**

* 1 <= s.length <= 104
* s consists of parentheses only '()[]{}'.

### Analysis:

Because we have 3 different kinds of parentheses, although it is fine to use scanning count, but using stack will make the code simple.

/// <summary>

/// Leet Code 20. Valid Parentheses

///

/// Easy

///

/// Given a string s containing just the characters '(', ')', '{', '}',

/// '[' and ']', determine if the input string is valid.

///

/// An input string is valid if:

///

/// Open brackets must be closed by the same type of brackets.

/// Open brackets must be closed in the correct order.

///

/// Example 1:

/// Input: s = "()"

/// Output: true

///

/// Example 2:

/// Input: s = "()[]{}"

/// Output: true

///

/// Example 3:

/// Input: s = "(]"

/// Output: false

///

/// Constraints:

/// 1. 1 <= s.length <= 10^4

/// 2. s consists of parentheses only '()[]{}'.

/// </summary>

bool LeetCodeString::isValidParentheses(string s)

{

stack<char> stack;

for (size\_t i = 0; i < s.size(); i++)

{

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

{

stack.push(s[i]);

}

else if (s[i] == ')')

{

if ((!stack.empty()) && (stack.top() == '('))

{

stack.pop();

}

else

{

return false;

}

}

else if (s[i] == ']')

{

if ((!stack.empty()) && (stack.top() == '['))

{

stack.pop();

}

else

{

return false;

}

}

else if (s[i] == '}')

{

if ((!stack.empty()) && (stack.top() == '{'))

{

stack.pop();

}

else

{

return false;

}

}

}

if (stack.size() == 0)

{

return true;

}

else

{

return false;

}

}

## 32. Longest Valid Parentheses

Hard

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

**Example 1:**

**Input:** s = "(()"

**Output:** 2

**Explanation:** The longest valid parentheses substring is "()".

**Example 2:**

**Input:** s = ")()())"

**Output:** 4

**Explanation:** The longest valid parentheses substring is "()()".

**Example 3:**

**Input:** s = ""

**Output:** 0

**Constraints:**

* 0 <= s.length <= 3 \* 104
* s[i] is '(', or ')'.

### Analysis:

We scan from left to right, at any position if we see the paretheses become invalid, we mark the position as -1, otherwise we keep it as 0. In the end we measure the longest consecutive zero.

/// <summary>

/// Leet Code 32. Longest Valid Parentheses

///

/// Hard

///

/// Given a string containing just the characters '(' and ')', find the

/// length of the longest valid (well-formed) parentheses substring.

///

/// Example 1:

/// Input: s = "(()"

/// Output: 2

/// Explanation: The longest valid parentheses substring is "()".

///

/// Example 2:

/// Input: s = ")()())"

/// Output: 4

/// Explanation: The longest valid parentheses substring is "()()".

///

/// Example 3:

/// Input: s = ""

/// Output: 0

///

/// Constraints:

/// 1. 0 <= s.length <= 3 \* 10^4

/// 2. s[i] is '(', or ')'

/// </summary>

int LeetCodeString::longestValidParentheses(string s)

{

vector<int> dp(s.size());

int count = 0;

for (size\_t i = 0; i < s.size(); i++)

{

if (s[i] == '(') count++;

else count--;

if (count < 0)

{

count = 0;

dp[i] = -1;

}

}

count = 0;

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

{

if (s[i] == ')') count++;

else count--;

if (count < 0)

{

count = 0;

dp[i] = -1;

}

}

int result = 0;

count = 0;

for (size\_t i = 0; i < dp.size(); i++)

{

if (dp[i] == -1) count = 0;

else

{

count++;

result = max(result, count);

}

}

return result;

}

## 678. Valid Parenthesis String

Medium

Given a string s containing only three types of characters: '(', ')' and '\*', return true *if* s *is****valid***.

The following rules define a **valid** string:

* Any left parenthesis '(' must have a corresponding right parenthesis ')'.
* Any right parenthesis ')' must have a corresponding left parenthesis '('.
* Left parenthesis '(' must go before the corresponding right parenthesis ')'.
* '\*' could be treated as a single right parenthesis ')' or a single left parenthesis '(' or an empty string "".

**Example 1:**

**Input:** s = "()"

**Output:** true

**Example 2:**

**Input:** s = "(\*)"

**Output:** true

**Example 3:**

**Input:** s = "(\*))"

**Output:** true

**Constraints:**

* 1 <= s.length <= 100
* s[i] is '(', ')' or '\*'.

### Analysis:

Scan from left to right, count "()" and '\*' separately. If we see count is negative, which means at some point ')' is more than '(', then check '\*', if we have some '\*' available we can convert one '\*' to '('. If no more '\*', we make it as invalid. After that we can scan from right to left to perform exactly same process.

/// <summary>

/// Leet code #678. Valid Parenthesis String

///

/// Given a string containing only three types of characters: '(', ')'

/// and '\*', write a function to check whether this string is valid.

/// We define the validity of a string by these rules:

/// Any left parenthesis '(' must have a corresponding right

/// parenthesis ')'.

/// Any right parenthesis ')' must have a corresponding left

/// parenthesis '('.

/// Left parenthesis '(' must go before the corresponding right

/// parenthesis ')'.

/// '\*' could be treated as a single right parenthesis ')' or a single

/// left parenthesis '(' or an empty string.

/// An empty string is also valid.

/// Example 1:

/// Input: "()"

/// Output: True

/// Example 2:

/// Input: "(\*)"

/// Output: True

/// Example 3:

/// Input: "(\*))"

/// Output: True

/// Note:

/// The string size will be in the range [1, 100].

/// </summary>

bool LeetCodeString::checkValidString(string s)

{

// scan left to right

int parenthesis\_count = 0;

int star\_count = 0;

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

{

if (s[i] == '\*')

{

star\_count++;

}

else if (s[i] == '(')

{

parenthesis\_count++;

}

else

{

parenthesis\_count--;

if (parenthesis\_count < 0 && star\_count > 0)

{

parenthesis\_count++;

star\_count--;

}

}

if (parenthesis\_count < 0) return false;

}

// scan right to left

parenthesis\_count = 0;

star\_count = 0;

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

{

if (s[i] == '\*')

{

star\_count++;

}

else if (s[i] == ')')

{

parenthesis\_count++;

}

else

{

parenthesis\_count--;

if (parenthesis\_count < 0 && star\_count > 0)

{

parenthesis\_count++;

star\_count--;

}

}

if (parenthesis\_count < 0) return false;

}

return true;

}

## 921. Minimum Add to Make Parentheses Valid

Medium

A parentheses string is valid if and only if:

* It is the empty string,
* It can be written as AB (A concatenated with B), where A and B are valid strings, or
* It can be written as (A), where A is a valid string.

You are given a parentheses string s. In one move, you can insert a parenthesis at any position of the string.

* For example, if s = "()))", you can insert an opening parenthesis to be "(**(**)))" or a closing parenthesis to be "())**)**)".

Return *the minimum number of moves required to make*s*valid*.

**Example 1:**

**Input:** s = "())"

**Output:** 1

**Example 2:**

**Input:** s = "((("

**Output:** 3

**Constraints:**

* 1 <= s.length <= 1000
* s[i] is either '(' or ')'.

### Analysis:

Scan from left to right, count '(' as +1 and ')' as -1. When we see negative count, we should add '(', add 1 to result, and reset count. After that, we reset the count and scan from right to left do the same.

/// <summary>

/// Leet code #921. Minimum Add to Make Parentheses Valid

///

/// Given a string S of '(' and ')' parentheses, we add the minimum number of

/// parentheses ( '(' or ')', and in any positions ) so that the resulting

/// parentheses string is valid.

///

/// Formally, a parentheses string is valid if and only if:

///

/// It is the empty string, or

/// It can be written as AB (A concatenated with B), where A and B are valid

/// strings, or

/// It can be written as (A), where A is a valid string.

/// Given a parentheses string, return the minimum number of parentheses we

/// must add to make the resulting string valid.

///

/// Example 1:

/// Input: "())"

/// Output: 1

///

/// Example 2:

/// Input: "((("

/// Output: 3

///

/// Example 3:

/// Input: "()"

/// Output: 0

/// Example 4:

///

/// Input: "()))(("

/// Output: 4

///

/// Note:

///

/// 1. S.length <= 1000

/// 2. S only consists of '(' and ')' characters.

/// </summary>

int LeetCodeString::minAddToMakeValid(string S)

{

int result = 0;

int count = 0;

for (int i = 0; i < (int)S.size(); i++)

{

if (S[i] == '(') count++;

else count--;

if (count < 0)

{

result++;

count = 0;

}

}

count = 0;

for (int i = S.size() - 1; i >= 0; i--)

{

if (S[i] == ')') count++;

else count--;

if (count < 0)

{

result++;

count = 0;

}

}

return result;

}

## 1249. Minimum Remove to Make Valid Parentheses

Medium

Given a string s of '(' , ')' and lowercase English characters.

Your task is to remove the minimum number of parentheses ( '(' or ')', in any positions ) so that the resulting *parentheses string* is valid and return **any** valid string.

Formally, a *parentheses string* is valid if and only if:

* It is the empty string, contains only lowercase characters, or
* It can be written as AB (A concatenated with B), where A and B are valid strings, or
* It can be written as (A), where A is a valid string.

**Example 1:**

**Input:** s = "lee(t(c)o)de)"

**Output:** "lee(t(c)o)de"

**Explanation:** "lee(t(co)de)" , "lee(t(c)ode)" would also be accepted.

**Example 2:**

**Input:** s = "a)b(c)d"

**Output:** "ab(c)d"

**Example 3:**

**Input:** s = "))(("

**Output:** ""

**Explanation:** An empty string is also valid.

**Constraints:**

* 1 <= s.length <= 105
* s[i] is either'(' , ')', or lowercase English letter.

### Analysis:

Scan from left to right, count '(' as +1 and ')' as -1. When we see negative count, we should remove ')' , add 1 to result, and reset count. After that, we reset the count and scan from right to left do the same.

/// <summary>

/// Leet code #1249. Minimum Remove to Make Valid Parentheses

///

/// Given a string s of '(' , ')' and lowercase English characters.

///

/// Your task is to remove the minimum number of parentheses ( '(' or ')',

/// in any positions ) so that the resulting parentheses string is valid

/// and return any valid string.

///

/// Formally, a parentheses string is valid if and only if:

///

/// It is the empty string, contains only lowercase characters, or

/// It can be written as AB (A concatenated with B), where A and B are valid

/// strings, or It can be written as (A), where A is a valid string.

///

/// Example 1:

///

/// Input: s = "lee(t(c)o)de)"

/// Output: "lee(t(c)o)de"

/// Explanation: "lee(t(co)de)" , "lee(t(c)ode)" would also be accepted.

///

/// Example 2:

///

/// Input: s = "a)b(c)d"

/// Output: "ab(c)d"

///

/// Example 3:

///

/// Input: s = "))(("

/// Output: ""

/// Explanation: An empty string is also valid.

///

/// Example 4:

///

/// Input: s = "(a(b(c)d)"

/// Output: "a(b(c)d)"

///

/// Constraints:

///

/// 1. 1 <= s.length <= 10^5

/// 2. s[i] is one of '(' , ')' and lowercase English letters.

/// </summary>

string LeetCodeString::minRemoveToMakeValid(string s)

{

int count = 0;

string str = s;

for (size\_t i = 0; i < str.size(); i++)

{

if (str[i] == '(') count++;

else if (str[i] == ')')

{

if (count == 0) str[i] = '\*';

else count--;

}

}

count = 0;

for (int i = (int)str.size() - 1; i >= 0; i--)

{

if (str[i] == ')') count++;

else if (str[i] == '(')

{

if (count == 0) str[i] = '\*';

else count--;

}

}

string result;

for (size\_t i = 0; i < str.size(); i++)

{

if (str[i] == '\*') continue;

result.push\_back(str[i]);

}

return result;

}

## 1541. Minimum Insertions to Balance a Parentheses String

Medium

Given a parentheses string s containing only the characters '(' and ')'. A parentheses string is **balanced** if:

* Any left parenthesis '(' must have a corresponding two consecutive right parenthesis '))'.
* Left parenthesis '(' must go before the corresponding two consecutive right parenthesis '))'.

In other words, we treat '(' as an opening parenthesis and '))' as a closing parenthesis.

* For example, "())", "())(())))" and "(())())))" are balanced, ")()", "()))" and "(()))" are not balanced.

You can insert the characters '(' and ')' at any position of the string to balance it if needed.

Return *the minimum number of insertions* needed to make s balanced.

**Example 1:**

**Input:** s = "(()))"

**Output:** 1

**Explanation:** The second '(' has two matching '))', but the first '(' has only ')' matching. We need to to add one more ')' at the end of the string to be "(())))" which is balanced.

**Example 2:**

**Input:** s = "())"

**Output:** 0

**Explanation:** The string is already balanced.

**Example 3:**

**Input:** s = "))())("

**Output:** 3

**Explanation:** Add '(' to match the first '))', Add '))' to match the last '('.

**Constraints:**

* 1 <= s.length <= 105
* s consists of '(' and ')' only.

### Analysis:

In this problem, one '(' count 2 and one ')' count -1. If we see any '(', the previous ')' must be even, if it is odd, we add 1 ')' first to make it even. If we see count negative, we add '(' and count 2. When we scan from right to left, if we see negative count we add '(' and 2 to count.

/// <summary>

/// Leet code #1541. Minimum Insertions to Balance a Parentheses String

///

/// Medium

///

/// Given a parentheses string s containing only the characters '('

/// and ')'. A parentheses string is balanced if:

///

/// Any left parenthesis '(' must have a corresponding two consecutive

/// right parenthesis '))'.

/// Left parenthesis '(' must go before the corresponding two consecutive

/// right parenthesis '))'.

/// For example, "())", "())(())))" and "(())())))" are balanced, ")()",

/// "()))" and "(()))" are not balanced.

///

/// You can insert the characters '(' and ')' at any position of the

/// string to balance it if needed.

///

/// Return the minimum number of insertions needed to make s balanced.

///

/// Example 1:

/// Input: s = "(()))"

/// Output: 1

/// Explanation: The second '(' has two matching '))', but the first

/// '(' has only ')' matching. We need to to add one more ')' at the

/// end of the string to be "(())))" which is balanced.

///

/// Example 2:

/// Input: s = "())"

/// Output: 0

/// Explanation: The string is already balanced.

///

/// Example 3:

/// Input: s = "))())("

/// Output: 3

/// Explanation: Add '(' to match the first '))', Add '))' to match the

/// last '('.

///

/// Example 4:

/// Input: s = "(((((("

/// Output: 12

/// Explanation: Add 12 ')' to balance the string.

///

/// Example 5:

/// Input: s = ")))))))"

/// Output: 5

/// Explanation: Add 4 '(' at the beginning of the string and one ')' at

/// the end. The string becomes "(((())))))))".

///

/// Constraints:

/// 1. 1 <= s.length <= 10^5

/// 2. s consists of '(' and ')' only.

/// </summary>

int LeetCodeString::minInsertions(string s)

{

int result = 0;

int count = 0;

for (size\_t i = 0; i < s.size(); i++)

{

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

{

if (count % 2 == 1)

{

result++;

count--;

}

count += 2;

}

else count--;

if (count < 0)

{

result++;

count += 2;

}

}

result += count;

return result;

}

## 1963. Minimum Number of Swaps to Make the String Balanced

Medium

You are given a **0-indexed** string s of **even** length n. The string consists of **exactly** n / 2 opening brackets '[' and n / 2 closing brackets ']'.

A string is called **balanced** if and only if:

* It is the empty string, or
* It can be written as AB, where both A and B are **balanced** strings, or
* It can be written as [C], where C is a **balanced** string.

You may swap the brackets at **any** two indices **any** number of times.

Return *the****minimum****number of swaps to make*s ***balanced***.

**Example 1:**

**Input:** s = "][]["

**Output:** 1

**Explanation:** You can make the string balanced by swapping index 0 with index 3.

The resulting string is "[[]]".

**Example 2:**

**Input:** s = "]]][[["

**Output:** 2

**Explanation:** You can do the following to make the string balanced:

- Swap index 0 with index 4. s = "[]][][".

- Swap index 1 with index 5. s = "[[][]]".

The resulting string is "[[][]]".

**Example 3:**

**Input:** s = "[]"

**Output:** 0

**Explanation:** The string is already balanced.

**Constraints:**

* n == s.length
* 2 <= n <= 106
* n is even.
* s[i] is either '[' or ']'.
* The number of opening brackets '[' equals n / 2, and the number of closing brackets ']' equals n / 2.

### Analysis:

We scan left to right, then right to left, accumulate negative count as k. the result should be (k+1) /2.

/// <summary>

/// Leet Code 1963. Minimum Number of Swaps to Make the String Balanced

///

/// Medium

///

/// You are given a 0-indexed string s of even length n. The string

/// consists of exactly n / 2 opening brackets '[' and n / 2 closing

/// brackets ']'.

///

/// A string is called balanced if and only if:

///

/// It is the empty string, or

/// It can be written as AB, where both A and B are balanced strings, or

/// It can be written as [C], where C is a balanced string.

/// You may swap the brackets at any two indices any number of times.

///

/// Return the minimum number of swaps to make s balanced.

///

/// Example 1:

/// Input: s = "][]["

/// Output: 1

/// Explanation: You can make the string balanced by swapping index 0 with

/// index 3.

/// The resulting string is "[[]]".

///

/// Example 2:

/// Input: s = "]]][[["

/// Output: 2

/// Explanation: You can do the following to make the string balanced:

/// - Swap index 0 with index 4. s = "[]][][".

/// - Swap index 1 with index 5. s = "[[][]]".

/// The resulting string is "[[][]]".

///

/// Example 3:

/// Input: s = "[]"

/// Output: 0

/// Explanation: The string is already balanced.

///

/// Constraints:

/// 1. n == s.length

/// 2. 2 <= n <= 10^6

/// 3. n is even.

/// 4. s[i] is either '[' or ']'.

/// 5. The number of opening brackets '[' equals n / 2, and the number

/// of closing brackets ']' equals n / 2.

/// </summary>

int LeetCodeString::minSwaps(string s)

{

int result = 0;

int count = 0;

for (size\_t i = 0; i < s.size(); i++)

{

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

{

count++;

}

else

{

count--;

}

if (count < 0)

{

count = 0;

result++;

}

}

return (result + 1) / 2;

}