Length of the longest valid substring

Difficulty Level : Medium • Last Updated : 28 Feb, 2022

Given a string consisting of opening and closing parenthesis, find the length of the longest valid parenthesis substring.

Examples:

```
Input : ((()
Output : 2
Explanation : ()

Input: )()())
Output : 4
Explanation: ()()

Input:  ()(()))))
Output: 6
Explanation:  ()(())
```

A **Simple Approach** is to find all the substrings of given string. For every whether a substring is valid or not in linear time using a stack (See this for details). Time complexity of this solution is O(n^2.

An **Efficient Solution** can solve this problem in O(n) time. The idea is to store indexes of previous starting brackets in a stack. The first element of the stack is a special element that provides index before the beginning of valid substring (base for next valid string).

```
1) Create an empty stack and push -1 to it.
   The first element of the stack is used
   to provide a base for the next valid string.

2) Initialize result as 0.

3) If the character is '(' i.e. str[i] == '('),
   push index'i' to the stack.

2) Else (if the character is ')')
   a) Pop an item from the stack (Most of the
      time an opening bracket)
   b) If the stack is not empty, then find the
      length of current valid substring by taking
      the difference between the current index and
      top of the stack. If current length is more
      than the result, then update the result.
   c) If the stack is empty, push the current index
      as a base for the next valid substring.

3) Return result.
```

Below is the implementation of the above algorithm.

C++

```
#include <bits/stdc++.h>
using namespace std;

int findMaxLen(string str)
{
    int n = str.length();

    // Create a stack and push -1 as
    // initial index to it.
    stack<int> stk;
    stk.push(-1);

    // Initialize result
    int result = 0;

    // Traverse all characters of given string
    for (int i = 0; i < n; i++)
    {
        // If opening bracket, push index of it
        if (str[i] == '(')
            stk.push(i);

        // If closing bracket, i.e.,str[i] = ')'
        else
        {
            // Pop the previous opening
            // bracket's index
            if (!stk.empty())
            {
                stk.pop();
            }

            // Check if this length formed with base of
            // current valid substring is more than max
            // so far
            if (!stk.empty())
                result = max(result, i - stk.top());

            // If stack is empty. push current index as
            // base for next valid substring (if any)
            else
                stk.push(i);
    return result;
}

// Driver code
int main()
{
    string str = "((()()";

    // Function call
    cout << findMaxLen(str) << endl;

    str = "()(()))))";

    // Function call
    cout << findMaxLen(str) << endl;

    return 0;
}
```

Java

```
// Java program to find length of the longest valid
// substring

import java.util.Stack;

class Test
{
    // method to get length of the longest valid
    static int findMaxLen(String str)
    {
        int n = str.length();

        // Create a stack and push -1
        // as initial index to it.
        Stack<Integer> stk = new Stack<>();
        stk.push(-1);

        // Initialize result
        int result = 0;
```

```
            // If opening bracket, push index of it
            if (str.charAt(i) == '(')
                stk.push(i);

            // // If closing bracket, i.e.,str[i] = ')'
            else
            {
                // Pop the previous
                // opening bracket's index
                if(!stk.empty())
                    stk.pop();

                // Check if this length
                // formed with base of
                // current valid substring
                // is more than max
                // so far
                if (!stk.empty())
                    result
                        = Math.max(result,
                                   i - stk.peek());

                // If stack is empty. push
                // current index as base
                // for next valid substring (if any)
                else
                    stk.push(i);
            }
        }

        return result;
    }

    // Driver code
    public static void main(String[] args)
    {
        String str = "((()()";

        // Function call
        System.out.println(findMaxLen(str));

        str = "()(()))))";
    }
}
```

Python3

```python
# Python program to find length of the longest valid
# substring


def findMaxLen(string):
    n = len(string)

    # Create a stack and push -1
    # as initial index to it.
    stk = []
    stk.append(-1)

    # Initialize result
    result = 0

    # Traverse all characters of given string
    for i in range(n):

        # If opening bracket, push index of it
        if string[i] == '(':
            stk.append(i)

        # If closing bracket, i.e., str[i] = ')'
        else:

            # Pop the previous opening bracket's index
            if len(stk) != 0:
               stk.pop()

            # Check if this length formed with base of
            # current valid substring is more than max
            # so far
            if len(stk) != 0:
                result = max(result,
                             i - stk[len(stk)-1])
                stk.append(i)

    return result


# Driver code
string = "((()()"

# Function call
print (findMaxLen(string))

string = "()(()))))"

# Function call
print (findMaxLen(string))

# This code is contributed by Bhavya Jain
```

C#

```
// C# program to find length of
// the longest valid substring
using System;
using System.Collections.Generic;

class GFG {
    // method to get length of
    // the longest valid
    public static int findMaxLen(string str)
    {
        int n = str.Length;

        // Create a stack and push -1 as
        // initial index to it.
        Stack<int> stk = new Stack<int>();
        stk.Push(-1);

        // Initialize result
        int result = 0;
```

```
            // If opening bracket, push
            // index of it
            if (str[i] == '(') {
                stk.Push(i);
            }

            else // If closing bracket,
                 // i.e.,str[i] = ')'
            {
                // Pop the previous opening
                // bracket's index
                if (stk.Count > 0)
                    stk.Pop();

                // Check if this length formed
                // with base of current valid
                // substring is more than max
                // so far
                if (stk.Count > 0)
                {
                    result
                        = Math.Max(result,
                                   i - stk.Peek());
                }

                // If stack is empty. push current
                // index as base for next valid
                // substring (if any)
                else {
                    stk.Push(i);
                }
            }
        }

        return result;
    }

    // Driver Code
    public static void Main(string[] args)
    {
        string str = "((()()";
        str = "()(()))))";

        // Function call
        Console.WriteLine(findMaxLen(str));
    }
}

// This code is contributed by Shrikant13
```

Javascript

```
<script>
       // JavaScript Program for the above approach
       function findMaxLen(str)
       {
           let n = str.length;

           // Create a stack and push -1 as
           // initial index to it.
           let stk = [];
           stk.push(-1);

           // Initialize result
           let result = 0;

           // Traverse all characters of given string
           for (let i = 0; i < n; i++)
           {
               // If opening bracket, push index of it
               if (str.charAt(i) == '(')
               {
                   stk.push(i);
               }

               // If closing bracket, i.e.,str[i] = ')'
               else
               {
                   // Pop the previous opening
                   // bracket's index
                   if (stk.length != 0) {
                   // current valid substring is more than max
                   // so far
                   if (stk.length != 0) {
                       result = Math.max(result, i - stk[stk.length - 1])
                   }

                   // If stack is empty. push current index as
                   // base for next valid substring (if any)
                   else {
                       stk.push(i);
                   }
               }
           }

           return result;
       }

       // Driver code
       let str = "((()()";

       // Function call
       document.write(findMaxLen(str) + "<br>");

       str = "()(()))))";

       // Function call
       document.write(findMaxLen(str) + "<br>");

   // This code is contributed by Potta Lokesh
   </script>
```

Output

```
4
6
```

```
Input: str = "(()()"

Initialize result as 0 and stack with one item -1.

For i = 0, str[0] = '(', we push 0 in stack

For i = 1, str[1] = '(', we push 1 in stack

For i = 2, str[2] = ')', currently stack has
[-1, 0, 1], we pop from the stack and the stack
now is [-1, 0] and length of current valid substring
becomes 2 (we get this 2 by subtracting stack top from
current index).

Since the current length is more than the current result,
we update the result.

For i = 3, str[3] = '(', we push again, stack is [-1, 0, 3].
For i = 4, str[4] = ')', we pop from the stack, stack
becomes [-1, 0] and length of current valid substring
becomes 4 (we get this 4 by subtracting stack top from
current index).
Since current length is more than current result,
we update result.
```

Another **Efficient Approach** can solve the problem in O(n) time. The idea is to maintain an array that stores the length of the longest valid substring ending at that index. We iterate through the array and return the maximum value.

```
1) Create an array longest of length n (size of the input
   string) initialized to zero.
   The array will store the length of the longest valid
   substring ending at that index.

2) Initialize result as 0.

3) Iterate through the string from second character
   a) If the character is '(' set longest[i]=0 as no
      valid sub-string will end with '('.
   b) Else
      i) if s[i-1] = '('
            set longest[i] = longest[i-2] + 2
      ii) else
            set longest[i] = longest[i-1] + 2 +
            longest[i-longest[i-1]-2]

4) In each iteration update result as the maximum of
   result and longest[i]

5) Return result.
```

Below is the implementations of the above algorithm.

C++

```
// C++ program to find length of the longest valid
// substring
#include <bits/stdc++.h>
using namespace std;
```

```
        return 0;

    // Initialize curMax to zero
    int curMax = 0;

    vector<int> longest(s.size(), 0);

    // Iterate over the string starting from second index
    for (int i = 1; i < s.length(); i++)
    {
        if (s[i] == ')' && i - longest[i - 1] - 1 >= 0
            && s[i - longest[i - 1] - 1] == '(')
        {
            longest[i]
                = longest[i - 1] + 2
                  + ((i - longest[i - 1] - 2 >= 0)
                  ? longest[i - longest[i - 1] - 2]
                  : 0);
            curMax = max(longest[i], curMax);
        }
    }
    return curMax;
}

// Driver code
int main()
{
    string str = "((()()";

    // Function call
    cout << findMaxLen(str) << endl;

    str = "()(()))))";

    // Function call
    cout << findMaxLen(str) << endl;

    return 0;
}
// This code is contributed by Vipul Lohani
```

```java
// Java program to find length of the longest valid
// subString
import java.util.*;
class GFG
{

  static int findMaxLen(String s)
  {
    if (s.length() <= 1)
      return 0;

    // Initialize curMax to zero
    int curMax = 0;
    int[] longest = new int[s.length()];

    // Iterate over the String starting from second index
    for (int i = 1; i < s.length(); i++)
    {
      if (s.charAt(i) == ')' && i - longest[i - 1] - 1 >= 0
          && s.charAt(i - longest[i - 1] - 1) == '(')
      {
        longest[i]
          = longest[i - 1] + 2
          + ((i - longest[i - 1] - 2 >= 0)
             ? longest[i - longest[i - 1] - 2]
             : 0);
        curMax = Math.max(longest[i], curMax);
      }
    }
    return curMax;
  }

  // Driver code
  public static void main(String[] args)
  {
    String str = "((()()";

    // Function call
    System.out.print(findMaxLen(str) +"\n");

    str = "()(()))))";
  }
}

// This code is contributed by aashish1995
```

Python3

```
# Python3 program to find length of
# the longest valid substring


def findMaxLen(s):
    if (len(s) <= 1):
        return 0

    # Initialize curMax to zero
    curMax = 0

    longest = [0] * (len(s))

    # Iterate over the string starting
    # from second index
    for i in range(1, len(s)):
        if ((s[i] == ')'
             and i - longest[i - 1] - 1 >= 0
             and s[i - longest[i - 1] - 1] == '(')):

            longest[i] = longest[i - 1] + 2
            if (i - longest[i - 1] - 2 >= 0):
                longest[i] += (longest[i -
                                       longest[i - 1] - 2])
            else:
                longest[i] += 0
            curMax = max(longest[i], curMax)
    return curMax


# Driver Code
if __name__ == '__main__':
    Str = "((()()"
    Str = "()(()))))"

    # Function call
    print(findMaxLen(Str))

# This code is contributed by PranchalK
```

C#

```
// C# program to find length of the longest valid
// subString
using System;

public class GFG {

    static int findMaxLen(String s) {
        if (s.Length <= 1)
            return 0;

        // Initialize curMax to zero
        int curMax = 0;
        int[] longest = new int[s.Length];

        // Iterate over the String starting from second index
        for (int i = 1; i < s.Length; i++) {
            if (s[i] == ')' && i - longest[i - 1] - 1 >= 0 && s[i - longe
                longest[i] = longest[i - 1] + 2 + ((i - longest[i - 1] -
                curMax = Math.Max(longest[i], curMax);
            }
        }
        return curMax;
    }

    // Driver code
    public static void Main(String[] args) {
        String str = "((()()";

        // Function call
        Console.Write(findMaxLen(str) + "\n");

        str = "()(()))))";
    }
}

// This code is contributed by aashish1995
```

Javascript

```
<script>
// javascript program to find length of the longest valid
// subString
    function findMaxLen( s) {
        if (s.length <= 1)
            return 0;

        // Initialize curMax to zero
        var curMax = 0;
        var longest = Array(s.length).fill(0);

        // Iterate over the String starting from second index
        for (var i = 1; i < s.length; i++) {
            if (s[i] == ')' && i - longest[i - 1] - 1 >= 0 && s[i - longe
                longest[i] = longest[i - 1] + 2 + ((i - longest[i - 1] -
                curMax = Math.max(longest[i], curMax);
            }
        }
        return curMax;
    }

    // Driver code
        var str = "((()()";

        // Function call
        document.write(findMaxLen(str) + "<br\>");

        str = "()(()))))";

        // Function call
        document.write(findMaxLen(str) + "<br\>");

// This code is contributed by umadevi9616
```

Output

```
4
6
```

Thanks to Gaurav Ahirwar and Ekta Goel for suggesting above approach.

Another approach in O(1) auxiliary space and O(N) Time complexity:

1. The idea to solve this problem is to traverse the string on and keep track of the count of open parentheses and close parentheses with the help of two counters **left** and **right** respectively.
2. First, the string is traversed from the left towards the right and for every "**(**" encountered, the **left counter** is incremented by 1 and for every "**)**" the **right counter** is incremented by 1.
3. Whenever the left becomes equal to right, the length of the current valid string is calculated and if it greater than the current longest substring, then value of required longest substring is updated with current string length.
4. If the right counter becomes greater than the left counter, then the set of parentheses has become **invalid** and hence the left and right counters are **set to 0**.
5. After the above process, the string is similarly traversed from right to left and similar procedure is applied.

Below is the implementation of the above approach:

C++

```
// C++ program to implement the above approach

#include <bits/stdc++.h>
using namespace std;
```

```
int solve(string s, int n)
{

    // Variables for left and right counter.
    // maxlength to store the maximum length found so far
    int left = 0, right = 0, maxlength = 0;

    // Iterating the string from left to right
    for (int i = 0; i < n; i++)
    {
        // If "(" is encountered,
        // then left counter is incremented
        // else right counter is incremented
        if (s[i] == '(')
            left++;
        else
            right++;

        // Whenever left is equal to right, it signifies
        // that the subsequence is valid and
        if (left == right)
            maxlength = max(maxlength, 2 * right);

        // Resetting the counters when the subsequence
        // becomes invalid
        else if (right > left)
            left = right = 0;
    }

    left = right = 0;

    // Iterating the string from right to left
    for (int i = n - 1; i >= 0; i--) {

        // If "(" is encountered,
        // then left counter is incremented
        // else right counter is incremented
        if (s[i] == '(')
            left++;
        else
            right++;
```

```
                maxlength = max(maxlength, 2 * left);

            // Resetting the counters when the subsequence
            // becomes invalid
            else if (left > right)
                left = right = 0;
        }
        return maxlength;
    }


    //A much shorter and concise version of the above code
    int solve2(string s, int n){
        int left=0,right=0,maxlength=0,t=2;
          while(t--){
            left=0;
            right=0;
            for(int i=0;i<n;i++){
                if(s[i]=='(')left++;
                else right++;

                if(left==right){
                    maxlength=max(maxlength,2*left);
                }
                //when travelling from 0 to n-1
                if(t%2==1 && right>left){
                    left=0;
                    right=0;
                  }
                  //when travelling from n-1 to 0
                  if(t%2==0 && left>right){
                      left=0;
                      right=0;
                  }
                }
                  //now we need to do the same thing from the other side;
                reverse(s.begin(),s.end());
            }
            return maxlength;
    }

    // Driver code
        // Function call
        cout << solve("((()()()()(((())", 16);
        return 0;
    }
```

Java

```
// Java program to implement the above approach
import java.util.Scanner;
import java.util.Arrays;

class GFG {

    // Function to return the length
    // of the longest valid substring
    public static int solve(String s, int n)
    {

        // Variables for left and right
        // counter maxlength to store
        // the maximum length found so far
        int left = 0, right = 0;
        int maxlength = 0;

        // Iterating the string from left to right
        for (int i = 0; i < n; i++) {

            // If "(" is encountered, then
            // left counter is incremented
            // else right counter is incremented
            if (s.charAt(i) == '(')
                left++;
            else
                right++;

            // Whenever left is equal to right,
            // it signifies that the subsequence
            // is valid and
            if (left == right)
                maxlength = Math.max(maxlength,
                                     2 * right);
            else if (right > left)
                left = right = 0;
        }

        left = right = 0;

        // Iterating the string from right to left
        for (int i = n - 1; i >= 0; i--) {

            // If "(" is encountered, then
            // left counter is incremented
            // else right counter is incremented
            if (s.charAt(i) == '(')
                left++;
            else
                right++;

            // Whenever left is equal to right,
            // it signifies that the subsequence
            // is valid and
            if (left == right)
                maxlength = Math.max(maxlength,
                                     2 * left);

            // Resetting the counters when the
            // subsequence becomes invalid
            else if (left > right)
                left = right = 0;
        }
        return maxlength;
    }

    // Driver code
    public static void main(String args[])
    {
        // Function call
        System.out.print(solve("((()()()()(((())", 16));
    }
}

// This code is contributed by SoumikMondal
```

```python
# Python3 program to implement the above approach

# Function to return the length of
# the longest valid substring


def solve(s, n):

    # Variables for left and right counter.
    # maxlength to store the maximum length found so far
    left = 0
    right = 0
    maxlength = 0

    # Iterating the string from left to right
    for i in range(n):

        # If "(" is encountered,
        # then left counter is incremented
        # else right counter is incremented
        if (s[i] == '('):
            left += 1
        else:
            right += 1

        # Whenever left is equal to right, it signifies
        # that the subsequence is valid and
        if (left == right):
            maxlength = max(maxlength, 2 * right)

        # Resetting the counters when the subsequence
        # becomes invalid
        elif (right > left):
            left = right = 0

    left = right = 0

    # Iterating the string from right to left
    for i in range(n - 1, -1, -1):

        # If "(" is encountered,
        # then left counter is incremented
```

```
            else:
                right += 1

            # Whenever left is equal to right, it signifies
            # that the subsequence is valid and
            if (left == right):
                maxlength = max(maxlength, 2 * left)

            # Resetting the counters when the subsequence
            # becomes invalid
            elif (left > right):
                left = right = 0
        return maxlength


# Driver code
# Function call
print(solve("((()()()()(((())", 16))

# This code is contributed by shubhamsingh10
```

C#

```
// C# program to implement the above approach
using System;

public class GFG {

  // Function to return the length
  // of the longest valid substring
  public static int solve(String s, int n)
  {

    // Variables for left and right
    // counter maxlength to store
    // the maximum length found so far
    int left = 0, right = 0;
    int maxlength = 0;

    // Iterating the string from left to right
    for (int i = 0; i < n; i++) {
      // else right counter is incremented
      if (s[i] == '(')
        left++;
      else
        right++;

      // Whenever left is equal to right,
      // it signifies that the subsequence
      // is valid and
      if (left == right)
        maxlength = Math.Max(maxlength,
                             2 * right);

      // Resetting the counters when the
      // subsequence becomes invalid
      else if (right > left)
        left = right = 0;
    }

    left = right = 0;

    // Iterating the string from right to left
    for (int i = n - 1; i >= 0; i--) {

      // If "(" is encountered, then
      // left counter is incremented
      // else right counter is incremented
      if (s[i] == '(')
        left++;
      else
        right++;

      // Whenever left is equal to right,
      // it signifies that the subsequence
      // is valid and
      if (left == right)
        maxlength = Math.Max(maxlength,
                             2 * left);

      // Resetting the counters when the
      // subsequence becomes invalid
      else if (left > right)
```

```
    }

    // Driver code
    public static void Main(String []args)
    {
      // Function call
      Console.Write(solve("((()()()()(((())", 16));
    }
  }


// This code is contributed by Rajput-Ji
```

Javascript

```
<script>

// JavaScript program to implement the above approach


// Function to return the length of
// the longest valid substring
function solve(s, n)
{

    // Variables for left and right counter.
    // maxlength to store the maximum length found so far
    let left = 0, right = 0, maxlength = 0;

    // Iterating the string from left to right
    for (let i = 0; i < n; i++)
    {
        // If "(" is encountered,
        // then left counter is incremented
        // else right counter is incremented
        if (s[i] == '(')
            left++;
        else
            right++;

        // Whenever left is equal to right, it signifies
```

```
            // Resetting the counters when the subsequence
            // becomes invalid
            else if (right > left)
                left = right = 0;
        }

        left = right = 0;

        // Iterating the string from right to left
        for (let i = n - 1; i >= 0; i--) {

            // If "(" is encountered,
            // then left counter is incremented
            // else right counter is incremented
            if (s[i] == '(')
                left++;
            else
                right++;

            // Whenever left is equal to right, it signifies
            // that the subsequence is valid and
            if (left == right)
                maxlength = Math.max(maxlength, 2 * left);

            // Resetting the counters when the subsequence
            // becomes invalid
            else if (left > right)
                left = right = 0;
        }
        return maxlength;
    }

    // Driver code


        // Function call
        document.write( solve("((()()()()(((())", 16));

    //This code is contributed by Manoj
    </script>
```

```
8
```

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Like 126

Next

Check for Balanced Brackets in an expression (well-formedness) using Stack

RECOMMENDED ARTICLES

Page : 1 2 3

01 **Find a valid parenthesis sequence of length K from a given valid parenthesis sequence**
10, Sep 20

02 **Generate an N-length string having longest palindromic substring of length K**
14, Dec 20

03 **Longest substring whose any non-empty substring not prefix or suffix of given String**
29, Mar 22

04 **Length of the largest substring which have character with frequency greater than or equal to half of the substring**
29, May 19

05 **Minimum length of substring whose rotation generates a palindromic substring**
21, May 20

06 **Length of the longest substring that do not contain any palindrome**
08, May 19

07 **Length of longest substring having all characters as K**
16, Jul 19

08 **Find length of longest substring with at most K normal characters**
26, Apr 20

Article Contributed By :

GeeksforGeeks

Easy Normal Medium Hard Expert

Improved By : Vipul Lohani, shrikanth13, PranchalKatiyar, guptaakshit21, SHUBHAMSINGH10, SoumikMondal, rsaini9416624750, Rajput-Ji, aashish1995, mank1083, lokeshpotta20, rkstrdee, umadevi9616, amartyaghoshgfg, simmytarika5

Article Tags : Amazon, Google, Dynamic Programming, Queue, Stack, Strings

Practice Tags : Amazon, Google, Strings, Dynamic Programming, Stack, Queue

Improve Article Report Issue

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Courses

News

Top News

Technology

Work & Career

Business

Finance

Lifestyle

Knowledge

Languages

Python

Java

CPP

Golang

C#

SQL

Kotlin

Web Development

Web Tutorials

Django Tutorial

HTML

JavaScript

Bootstrap

ReactJS

NodeJS

Contribute

Write an Article

Improve an Article

Pick Topics to Write

Write Interview Experience

Internships

Video Internship