Minimum bracket reversals to make an expression balanced - GeeksforGeeks

Source: https://www.geeksforgeeks.org/minimum-number-of-bracket-reversals-needed-to-make-an-expression-balanced/

Courses Tutorials Practice Jobs DSA Tutorial Interview Questions Quizzes Must Do Advanced DSA System Design Aptitude Puzzles Interview Corner DSA Python Technical Scripter 2026 Explore DSA Fundamentals Logic Building Problems Analysis of Algorithms Data Structures Array Data Structure String in Data Structure Hashing in Data Structure Linked List Data Structure Stack Data Structure Queue Data Structure Tree Data Structure Graph Data Structure Trie Data Structure Algorithms Searching Algorithms Sorting Algorithms Introduction to Recursion Greedy Algorithms Tutorial Graph Algorithms Dynamic Programming or DP Bitwise Algorithms Advanced Segment Tree Binary Indexed Tree or Fenwick Tree Square Root (Sqrt) Decomposition Algorithm Binary Lifting Geometry Interview Preparation Interview Corner GfG160 Practice Problem GeeksforGeeks Practice - Leading Online Coding Platform Problem of The Day - Develop the Habit of Coding DSA Course 90% Refund Minimum bracket reversals to make an expression balanced Last Updated : 26 Mar, 2025 Given an expression with only '}' and '{'. The expression may not be balanced. Find minimum number of bracket reversals to make the expression balanced. Examples: Input: s = "}{{}}{{{" Output: 3 Explanation : We need to reverse minimum 3 brackets to make " { {{}}{ }} ". Input: s = "{{" Output: 1 Explanation : We need 1 reversal Input : s = "{{}{{{}{{}}{{" Output: -1 Explanation : There's no way we can balance this sequence of braces as the length is odd. Try it on GfG Practice [Naive Approach] Recursive Bracket Reversal - O(2^n * n) time and (n) space This approach recursively checks all possible ways to reverse brackets to make the expression balanced. The algorithm explores two options for each bracket: keeping it as it is or reversing it. The goal is to find the minimum number of reversals needed to balance the string. Steps: Check if the string length is even : If it's odd, return -1 because a balanced string requires an even number of brackets. Recursive function : For each bracket, either keep it as is or reverse it, then check if the resulting string is balanced. Balance check : Traverse the string and maintain a count variable to track the balance of { and } . If at any point the count becomes negative, the string is unbalanced. Return the minimum number of reversals : If a balanced string is found, compare the number of reversals and store the minimum. C++ #include <bits/stdc++.h> using namespace std ; //Function to check that expression is balanced or not bool isBalanced (string expr) { // Initialising Variables bool flag = true ; int count = 0 ; // Traversing the Expression for (int i = 0 ; i < expr . length (); i ++) { if (expr [i] == '{') { count ++ ; } else { // It is a closing bracket count -- ; } if (count < 0) { // This means there are more // Closing brackets than opening ones flag = false ; break ; } } // If count is not zero, It means // there are more opening brackets if (count != 0) { flag = false ; } return flag ; } void recur (string expr , int n , int ind , int change , int & ans){ // When generated expression is balanced if (isBalanced (expr)){ ans = min (ans , change);} // When we covered whole string if (ind == n){ return ;} // Keep bracket as it is recur (expr , n , ind + 1 , change , ans); // Reverse the bracket if (expr [ind] == '{'){ expr [ind] = '}' ;} else { expr [ind] = '{' ;} recur (expr , n , ind + 1 , change + 1 , ans); } int countMinReversals (string expr) { // Length of expression int n = expr . length (); // To store answer int ans = INT_MAX ; // When total number of brackets are odd if (n % 2 == 1){ return -1 ; } else { recur (expr , n , 0 , 0 , ans); return ans ; } } int main () { string expr = "}{{}}{{{" ; cout << countMinReversals (expr); return 0 ; } Java import java.util.* ; public class GfG { public static boolean isBalanced (String expr) { int count = 0 ; // Traversing the expression for (int i = 0 ; i < expr . length (); i ++) { if (expr . charAt (i) == '{') { count ++ ; } else { // It is a closing bracket count -- ; } if (count < 0) { // This means there are more closing brackets than opening ones return false ; } } // If count is not zero, it means there are more opening brackets return count == 0 ; } // Recursive function to find the number of reversals public static void recur (String expr , int n , int ind , int change , int [] ans) { // When generated expression is balanced if (isBalanced (expr)) { ans [0] = Math . min (ans [0] , change); } // When we have covered the whole string if (ind == n) { return ; } // Keep bracket as it is recur (expr , n , ind + 1 , change , ans); // Reverse the bracket char [] exprArray = expr . toCharArray (); if (exprArray [ind] == '{') { exprArray [ind] = '}' ; } else { exprArray [ind] = '{' ; } recur (new String (exprArray), n , ind + 1 , change + 1 , ans); } public static int countMinReversals (String expr) { // Length of expression int n =

expr . length (); // To store answer int [] ans = { Integer . MAX_VALUE }; // When the total number of brackets is odd if (n % 2 == 1) { return - 1 ; } else { // Function call for finding answer recur (expr , n , 0 , 0 , ans); return ans [0] ; } } public static void main (String [] args) { String expr = "}{{}}{{{" ; System . out . println (countMinReversals (expr)); } } Python def is_balanced (expr): # Initialising Variables flag = True count = 0 # Traversing the Expression for char in expr : if char == '{' : count += 1 else : # It is a closing bracket count -= 1 if count < 0 : # This means there are more Closing brackets than opening ones flag = False break # If count is not zero, It means there are more opening brackets if count != 0 : flag = False return flag def recur (expr , n , ind , change , ans): # When generated expression is balanced if is_balanced (expr): ans [0] = min (ans [0], change) # When we covered whole string if ind == n : return # Keep bracket as it is recur (expr , n , ind + 1 , change , ans) # Reverse the bracket if expr [ind] == '{' : expr [ind] = '}' else : expr [ind] = '{' recur (expr , n , ind + 1 , change + 1 , ans) def countMinReversals (expr): # Length of expression n = len (expr) # To store answer ans = [float ('inf')] # When total number of brackets are odd if n % 2 == 1 : return - 1 else : # Function call for finding answer recur (list (expr), n , 0 , 0 , ans) return ans [0] if __name__ == '__main__' : expr = '}{{}}{{{' print (countMinReversals (expr)) C# using System ; class GfG { // Function to check if the expression is balanced or not public static bool IsBalanced (string expr) { int count = 0 ; // Traversing the expression for (int i = 0 ; i < expr . Length ; i ++) { if (expr [i] == '{') { count ++ ; } else { // It is a closing bracket count -- ; } if (count < 0) { // This means there are more closing brackets than opening ones return false ; } } // If count is not zero, it means there are more opening brackets return count == 0 ; } // Recursive function to find the number of reversals public static void recur (string expr , int n , int ind , int change , ref int ans) { // When generated expression is balanced if (IsBalanced (expr)) { ans = Math . Min (ans , change); } // When we have covered the whole string if (ind == n) { return ; } // Keep bracket as it is recur (expr , n , ind + 1 , change , ref ans); // Reverse the bracket char [] exprArray = expr . ToCharArray (); if (exprArray [ind] == '{') { exprArray [ind] = '}' ; } else { exprArray [ind] = '{' ; } recur (new string (exprArray), n , ind + 1 , change + 1 , ref ans); } public static int countMinReversals (string expr) { int n = expr . Length ; // To store answer int ans = int . MaxValue ; // When the total number of brackets is odd if (n % 2 == 1) { return - 1 ; } else { // Function call for finding answer recur (expr , n , 0 , 0 , ref ans); return ans ; } } // Entry point of the program public static void Main (string [] args) { string expr = "}{{}}{{{" ; Console . WriteLine (countMinReversals (expr)); } } JavaScript // Function to check if the expression is balanced or not function isBalanced (expr) { let count = 0 ; // Traversing the expression for (let i = 0 ; i < expr . length ; i ++) { if (expr [i] === '{') { count ++ ; } else { // It is a closing bracket count -- ; } if (count < 0) { // This means there are more closing brackets than opening ones return false ; } } // If count is not zero, it means there are more opening brackets return count === 0 ; } // Recursive function to find the number of reversals function recur (expr , n , ind , change , ans) { // When generated expression is balanced if (isBalanced (expr)) { ans [0] = Math . min (ans [0], change); } // When we have covered the whole string if (ind === n) { return ; } // Keep bracket as it is recur (expr , n , ind + 1 , change , ans); // Reverse the bracket let exprArray = expr . split (''); if (exprArray [ind] === '{') { exprArray [ind] = '}' ; } else { exprArray [ind] = '{' ; } recur (exprArray . join (''), n , ind + 1 , change + 1 , ans); } // Returns the count of minimum reversals for making expr balanced. Returns -1 if expr cannot be balanced. function countMinReversals (expr) { // Length of expression const n = expr . length ; // To store answer let ans = [Infinity]; // When the total number of brackets is odd if (n % 2 === 1) { return - 1 ; } else { // Function call for finding answer recur (expr , n , 0 , 0 , ans); return ans [0] === Infinity ? - 1 : ans [0]; } } // Test the function const expr = "}{{}}{{{" ; console . log (countMinReversals (expr)); Output 3 Time complexity : O(2 n *n), because O(2 n) in recursive function and O(n) in checking the expression generated after every recursive call is balanced or not. Auxiliary space O(n), because of the Auxillary space of recursion [Better Approach] Stack Based Bracket Reversal - O(n) time and O(n) space The solution involves first removing all balanced parts of the expression. For example, converting "}{{}}{{{" to "}}{{{" by eliminating the balanced part. After this removal, the remaining expression will always have the form }}...}{{...{ , consisting of some closing brackets followed by some opening brackets. To balance this remaining expression, we need to calculate the minimum number of reversals required. Let m be the number of closing brackets and n be the number of opening brackets. The minimum reversals required is given by ceil(m / 2) + ceil(n / 2) . C++ #include <bits/stdc++.h> using namespace std ; int countMinReversals (string expr) { int len = expr . length (); // length of expression must be even to make, it balanced by using reversals. if (len % 2) return -1 ; stack < char > s ; for (int i = 0 ; i < len ; i ++) { if (expr [i] == '}' && ! s . empty ()) { if (s . top () == '{') s . pop (); else s . push (expr [i]); } else s . push (expr [i]); } // Length of the reduced expression red_len = (m+n) int red_len = s . size (); // count opening brackets at the end of stack int n = 0 ; while (! s . empty () && s . top () == '{') { s . pop (); n ++ ; } // return ceil(m/2) + ceil(n/2) which i actually equal to (m+n)/2 + n%2 when m+n is even. return (red_len / 2 + n % 2); } int main () { string s = "}{{}}{{{" ; cout << countMinReversals (s); return 0 ; } Java import java.util.Stack ; public class GfG { public static int countMinReversals (String expr) { int len = expr . length (); // length of expression must be even to make, it balanced by using reversals. if (len % 2 != 0) return - 1 ; Stack < Character > s = new Stack <> (); for (int i = 0 ; i < len ; i ++) { if (expr . charAt (i) == '}' && ! s . isEmpty ()) { if (s . peek () == '{') s . pop (); else s . push (expr . charAt (i)); } else { s . push (expr . charAt (i)); } } // Length of the reduced expression red_len = (m+n) int red_len = s . size (); // count opening brackets at the end of stack int n = 0 ; while (! s . isEmpty () && s . peek () == '{') { s . pop (); n ++ ; } // return ceil(m/2) + ceil(n/2) which i actually equal to (m+n)/2 + n%2 when m+n is even. return (red_len / 2 + n % 2); } public static void main (String [] args) { String s = "}{{}}{{{" ; System . out . println (countMinReversals (s)); } } Python def countMinReversals (expr): length = len (expr) # length of expression must be even to make, it balanced by using reversals. if length % 2 : return - 1 stack = [] for char in expr : if char == '}' and stack : if stack [- 1] == '{' : stack . pop () else : stack . append (char) else : stack . append (char) # Length of the reduced expression red_len = (m+n) red_len = len (stack) # count opening brackets at the end of stack n = 0 while stack and stack [- 1] == '{' : stack . pop () n += 1 # return ceil(m/2) + ceil(n/2) which is actually equal to (m+n)/2 + n%2 when m+n is even. return (red_len // 2 + n % 2) if __name__ == '__main__' : s = '}{{}}{{{' print (countMinReversals (s)) C# using System ; using System.Collections.Generic ; class GfG { public static int CountMinReversals (string expr) { int len = expr . Length ; // length of expression must be even to make, it balanced by using reversals. if (len % 2 != 0) return - 1 ; Stack < char > s = new Stack < char > (); for (int i = 0 ; i < len ; i ++) { if (expr [i] == '}' && s . Count > 0) { if (s . Peek () == '{') s . Pop (); else s . Push (expr [i]); } else { s . Push (expr [i]); } } // Length of the reduced expression red_len = (m+n) int red_len = s . Count ; // count opening brackets at the end of stack int n = 0 ; while (s . Count > 0 && s . Peek () == '{') { s . Pop (); n ++ ; } // return ceil(m/2) + ceil(n/2) which is actually equal to (m+n)/2 + n%2 when m+n is even. return (red_len / 2 + n % 2); } static void Main () { string s = "}{{}}{{{" ; Console . WriteLine (CountMinReversals (s)); } } JavaScript // Function to count minimum reversals to make the expression balanced function countMinReversals (expr) { let len = expr . length ; // length of expression must be even to make, it balanced by using reversals. if (len % 2) return - 1 ; let stack = []; for (let i = 0 ; i < len ; i ++) { if (expr [i] === '}' && stack . length > 0) { if (stack [stack . length - 1] === '{') { stack . pop (); } else { stack . push (expr [i]); } } else { stack . push (expr [i]); } } // Length of the reduced expression red_len = (m+n) let red_len = stack . length ; // count opening brackets at the end of stack let n = 0 ; while (stack . length > 0 && stack [stack . length - 1] === '{') { stack . pop (); n ++ ; } // return ceil(m/2) + ceil(n/2) which is actually equal to (m+n)/2 + n%2 when m+n is even. return Math . floor (red_len / 2) + (n % 2); } let s = '}{{}}{{{' ; console . log (countMinReversals (s)); Output 3 [Expected Approach] Two-Pointer Bracket Reversal - O(n) time and O(1) space Since the expression only contains one type of bracket, we maintain two variables to keep track of the counts of left ({) and right (}) brackets as we process the string. If the expression has balanced brackets, we decrement the left variable. Otherwise, we increment the right variable. Finally, the minimum number of reversals required is calculated as ceil(left / 2) + ceil(right / 2) . C++ #include <bits/stdc++.h> using namespace std ; // Returns count of minimum reversals for making // expr balanced. Returns -1 if expr cannot be // balanced. int countMinReversals (string expr) { int len = expr . length (); // Expressions of odd lengths cannot be balanced if (len % 2 != 0) { return -1 ; } int left_brace = 0 , right_brace = 0 ; int ans ; for (int i = 0 ; i < len ; i ++) { // If we find a left bracket then we // simply increment the left bracket if (expr [i] == '{') { left_brace ++ ; } // Else if left bracket is 0 then we find // unbalanced right bracket and increment // right bracket or if the expression // is balanced then we decrement left else { if (left_brace == 0) { right_brace ++ ; } else { left_brace -- ; } } } ans = ceil (left_brace / 2.0) + ceil (right_brace / 2.0); return ans ; } int main () { string s = "}{{}}{{{" ; cout << countMinReversals (s); return 0 ; } Java // Returns count of minimum reversals for making // expr balanced. Returns -1 if expr cannot be // balanced. class Solution { public int countMinReversals (String expr) { int len = expr . length (); // Expressions of odd lengths cannot be balanced if (len % 2 != 0) { return - 1 ; } int left_brace = 0 , right_brace = 0 ; int ans ; for (int i = 0 ; i < len ; i ++) { // If we find a left bracket then we // simply increment the left bracket if (expr . charAt (i) == '{') { left_brace ++ ; } // Else if left bracket is 0 then we find // unbalanced right bracket and increment // right bracket or if the expression // is balanced then we decrement left else { if (left_brace == 0) { right_brace ++ ; } else { left_brace -- ; } } } ans = (int) Math . ceil (left_brace / 2.0) + (int) Math . ceil (right_brace / 2.0); return ans ; } public static void main (String [] args) { String s = "}{{}}{{{" ; Solution sol = new Solution (); System . out . println (sol . countMinReversals (s)); } } Python

```
# Returns count of minimum reversals for making # expr balanced. Returns -1 if expr cannot be #
balanced. def countMinReversals ( expr ): length = len ( expr ) # Expressions of odd lengths cannot be
balanced if length % 2 != 0 : return - 1 left_brace = 0 right_brace = 0 for char in expr : # If we find a left
bracket then # we simply increment the left bracket if char == '{' : left_brace += 1 # Else if left bracket is
0 then we find # unbalanced right bracket and increment # right bracket or if the expression # is
balanced then we decrement left else : if left_brace == 0 : right_brace += 1 else : left_brace -= 1 ans = (
left_brace + 1 ) // 2 + ( right_brace + 1 ) // 2 return ans s = "}{{}}{{{" print ( countMinReversals ( s )) C# //
Returns count of minimum reversals for making // expr balanced. Returns -1 if expr cannot be //
balanced. using System ; using System.Linq ; class GfG { static int CountMinReversals ( string expr ) {
int len = expr . Length ; // Expressions of odd lengths cannot be balanced if ( len % 2 != 0 ) { return - 1 ;
} int leftBrace = 0 , rightBrace = 0 ; int ans ; for ( int i = 0 ; i < len ; i ++ ) { // If we find a left bracket then
we // simply increment the left bracket if ( expr [ i ] == '{' ) { leftBrace ++ ; } // Else if left bracket is 0 then
we find // unbalanced right bracket and increment // right bracket or if the expression // is balanced then
we decrement left else { if ( leftBrace == 0 ) { rightBrace ++ ; } else { leftBrace -- ; } } } ans = ( int ) Math .
Ceiling ( leftBrace / 2.0 ) + ( int ) Math . Ceiling ( rightBrace / 2.0 ); return ans ; } static void Main () {
string s = "}{{}}{{{" ; Console . WriteLine ( CountMinReversals ( s )); } } JavaScript // Returns count of
minimum reversals for making // expr balanced. Returns -1 if expr cannot be // balanced. function
countMinReversals ( expr ) { let len = expr . length ; // Expressions of odd lengths cannot be balanced if
( len % 2 !== 0 ) { return - 1 ; } let leftBrace = 0 , rightBrace = 0 ; let ans ; for ( let i = 0 ; i < len ; i ++ ) { //
If we find a left bracket then we // simply increment the left bracket if ( expr [ i ] === '{' ) { leftBrace ++ ; }
// Else if left bracket is 0 then we find // unbalanced right bracket and increment // right bracket or if the
expression // is balanced then we decrement left else { if ( leftBrace === 0 ) { rightBrace ++ ; } else {
leftBrace -- ; } } } ans = Math . ceil ( leftBrace / 2 ) + Math . ceil ( rightBrace / 2 ); return ans ; } let s =
"}{{}}{{{" ; console . log ( countMinReversals ( s ));
```

Output 3 Comment Article Tags: Article Tags: Strings Stack Queue DSA Amazon + 1 More