Minimum Swaps for Bracket Balancing

Difficulty Level : Medium • Last Updated : 02 Jun, 2021

You are given a string of 2N characters consisting of N '[' brackets and N ']' brackets. A string is considered balanced if it can be represented in the for S2[S1] where S1 and S2 are balanced strings. We can make an unbalanced string balanced by swapping adjacent characters. Calculate the minimum number of swaps necessary to make a string balanced.

Examples:

```
Input  : []][][
Output : 2
First swap: Position 3 and 4
][][][
Second swap: Position 5 and 6
][][][

Input  : [][]]
```

```
Output : 0
The string is already balanced.
```

Recommended: Please solve it on "***PRACTICE***" first, before moving on to the solution.

We can solve this problem by using greedy strategies. If the first X characters form a balanced string, we can neglect these characters and continue on. If we encounter a ']' before the required '[', then we must start swapping elements to balance the string.

Naive Approach

Initialize sum = 0 where **sum** stores result. Go through the string maintaining a **count** of the number of '[' brackets encountered. Reduce this count when we encounter a ']' character. If the count hits negative, then we must start balancing the string.

Let index 'i' represent the position we are at. We now move forward to the next '[' at index j. Increase sum by j – i. Move the '[' at position j, to position i, and shift all other characters to the right. Set the count back to 0 and continue traversing the string. In the end, 'sum' will have the required value.

Time Complexity = O(N^2)

Extra Space = O(1)

Optimized approach

We can initially go through the string and store the positions of '[' in a vector say '**pos**'. Initialize 'p' to 0. We shall use p to traverse the vector 'pos'. Similar to the naive approach, we maintain a count of encountered '[' brackets. When we encounter a '[' we increase the count and increase 'p' by 1. When we encounter a ']' we decrease the count. If the count ever goes negative, this means we must start swapping. The element pos[p] tells us the index of the next '['. We increase the sum by pos[p] – i, where i is the current index. We can swap the elements in the current index and pos[p] and reset the count to 0 and increment p so that it pos[p] indicates to the next '['.
Since we have converted a step that was O(N) in the naive approach, to an O(1) step, our new time complexity reduces.

Time Complexity = O(N)
Extra Space = O(N)

C++

```
// C++ program to count swaps required to balance string
#include <iostream>
#include <vector>
#include <algorithm>
```

```
using namespace std;

// Function to calculate swaps required
long swapCount(string s)
{
    // Keep track of '['
    vector<int> pos;
    for (int i = 0; i < s.length(); ++i)
        if (s[i] == '[')
            pos.push_back(i);

    int count = 0; // To count number of encountered '['
    int p = 0;  // To track position of next '[' in pos
    long sum = 0; // To store result

    for (int i = 0; i < s.length(); ++i)
    {
        // Increment count and move p to next position
        if (s[i] == '[')
        {
            ++count;
            ++p;
        }
        else if (s[i] == ']')
            --count;

        // We have encountered an unbalanced part of string
        if (count < 0)
        {
            // Increment sum by number of swaps required
            // i.e. position of next '[' - current position
            sum += pos[p] - i;
            swap(s[i], s[pos[p]]);
            ++p;

            // Reset count to 1
            count = 1;
```

```
        }
    }
    return sum;
}

// Driver code
int main()
{
    string s = "[]][][";
    cout << swapCount(s) << "\n";

    s = "[[][]]";
    cout << swapCount(s) << "\n";
    return 0;
}
```

Java

```
// Java program to count swaps
// required to balance string
import java.util.*;

class GFG{

// Function to calculate swaps required
public static long swapCount(String s)
{

    // Keep track of '['
    Vector<Integer> pos = new Vector<Integer>();
    for(int i = 0; i < s.length(); ++i)
        if (s.charAt(i) == '[')
            pos.add(i);

    // To count number of encountered '['
```

```
    int count = 0;

    // To track position of next '[' in pos
    int p = 0;

    // To store result
    long sum = 0;

    char[] S = s.toCharArray();

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

        // Increment count and move p
        // to next position
        if (S[i] == '[')
        {
            ++count;
            ++p;
        }
        else if (S[i] == ']')
            --count;

        // We have encountered an
        // unbalanced part of string
        if (count < 0)
        {

            // Increment sum by number of
            // swaps required i.e. position
            // of next '[' - current position
            sum += pos.get(p) - i;
            char temp = S[i];
            S[i] = S[pos.get(p)];
            S[pos.get(p)] = temp;
            ++p;
```

```
            // Reset count to 1
            count = 1;
        }
    }
    return sum;
}

// Driver code
public static void main(String[] args)
{
    String s = "[]][][";
    System.out.println(swapCount(s));

    s = "[[][]]";
    System.out.println(swapCount(s));
}
}

// This code is contributed by divyesh072019
```

Python3

```
# Python3 Program to count
# swaps required to balance
# string

# Function to calculate
# swaps required
def swapCount(s):

    # Keep track of '['
    pos = []

    for i in range(len(s)):
        if(s[i] == '['):
```

```
            pos.append(i)

    # To count number
    # of encountered '['
    count = 0

    # To track position
    # of next '[' in pos
    p = 0

    # To store result
    sum = 0
    s = list(s)

    for i in range(len(s)):

        # Increment count and
        # move p to next position
        if(s[i] == '['):
            count += 1
            p += 1
        elif(s[i] == ']'):
            count -= 1

        # We have encountered an
        # unbalanced part of string
        if(count < 0):

            # Increment sum by number
            # of swaps required
            # i.e. position of next
            # '[' - current position
            sum += pos[p] - i
            s[i], s[pos[p]] = (s[pos[p]],
                               s[i])
            p += 1
```

```
            # Reset count to 1
            count = 1
    return sum

# Driver code
s = "[]][][["
print(swapCount(s))

s = "[[][]]"
print(swapCount(s))

# This code is contributed by avanitrachhadiya2155
```

C#

```
// C# program to count swaps
// required to balance string
using System.IO;
using System;
using System.Collections;
using System.Collections.Generic;

class GFG{

// Function to calculate swaps required
static long swapCount(string s)
{

    // Keep track of '['
    List<int> pos = new List<int>();
    for(int i = 0; i < s.Length; i++)
    {
        if (s[i] == '[')
        {
            pos.Add(i);
```

```
        }
    }

    // To count number of encountered '['
    int count = 0;

    // To track position of next '[' in pos
    int p = 0;

    // To store result
    long sum = 0;

    char[] S = s.ToCharArray();

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

        // Increment count and move p
        // to next position
        if (S[i] == '[')
        {
            ++count;
            ++p;
        }
        else if (S[i] == ']')
        {
            --count;
        }

        // We have encountered an
        // unbalanced part of string
        if (count < 0)
        {

            // Increment sum by number of
            // swaps required i.e. position
            // of next '[' - current position
```

```
            sum += pos[p]-i;
            char temp = S[i];
            S[i] = S[pos[p]];
            S[pos[p]] = temp;
            ++p;

            // Reset count to 1
            count = 1;
        }
    }
    return sum;
}

// Driver code
static void Main()
{
    string s = "[]][][";
    Console.WriteLine(swapCount(s));

    s = "[[][]]";
    Console.WriteLine(swapCount(s));
}
}

// This code is contributed by rag2127
```

Javascript

```
<script>

// JavaScript program to count swaps
// required to balance string

// Function to calculate swaps required
function swapCount(s)
```

```
{

    // Keep track of '['
    let pos = [];
    for(let i = 0; i < s.length; ++i)
        if (s[i] == '[')
            pos.push(i);

    // To count number of encountered '['
    let count = 0;

    // To track position of next '[' in pos
    let p = 0;

    // To store result
    let sum = 0;

    let S = s.split('');

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

        // Increment count and move p
        // to next position
        if (S[i] == '[')
        {
            ++count;
            ++p;
        }
        else if (S[i] == ']')
            --count;

        // We have encountered an
        // unbalanced part of string
        if (count < 0)
        {
```

```
            // Increment sum by number of
            // swaps required i.e. position
            // of next '[' - current position
            sum += pos[p] - i;
            let temp = S[i];
            S[i] = S[pos[p]];
            S[pos[p]] = temp;
            ++p;

            // Reset count to 1
            count = 1;
        }
    }
    return sum;
}

// Driver Code

    let s = "[]][][";
    document.write(swapCount(s) + "<br/>");

    s = "[[][]]";
    document.write(swapCount(s));

</script>
```

Output:

```
2
0
```

Another Method:

Time Complexity = O(N)

Extra Space = O(1)

We can do without having to store the positions of '['.

Below is the implementation :

C++

```
// C++ program to count swaps required
// to balance string
#include <bits/stdc++.h>
using namespace std;

long swapCount(string chars)
{

    // Stores total number of Left and
    // Right brackets encountered
    int countLeft = 0, countRight = 0;

    // swap stores the number of swaps
    // required imbalance maintains
    // the number of imbalance pair
    int swap = 0 , imbalance = 0;

    for(int i = 0; i < chars.length(); i++)
    {
        if (chars[i] == '[')
        {

            // Increment count of Left bracket
            countLeft++;
```

```
            if (imbalance > 0)
            {

                // swaps count is last swap count + total
                // number imbalanced brackets
                swap += imbalance;

                // imbalance decremented by 1 as it solved
                // only one imbalance of Left and Right
                imbalance--;
            }
        }
        else if(chars[i] == ']' )
        {

            // Increment count of Right bracket
            countRight++;

            // imbalance is reset to current difference
            // between Left and Right brackets
            imbalance = (countRight - countLeft);
        }
    }
    return swap;
}

// Driver code
int main()
{
    string s = "[]][][";
    cout << swapCount(s) << endl;

    s = "[[][]]";
    cout << swapCount(s) << endl;

    return 0;
}
```

```
// This code is contributed by divyeshrabadiya07
```

Java

```
// Java Program to count swaps required to balance string
public class BalanceParan
{

    static long swapCount(String s)
    {
        char[] chars = s.toCharArray();

        // stores total number of Left and Right
        // brackets encountered
        int countLeft = 0, countRight = 0;
                // swap stores the number of swaps required
        //imbalance maintains the number of imbalance pair
        int swap = 0 , imbalance = 0;

        for(int i =0; i< chars.length; i++)
        {
            if(chars[i] == '[')
            {
                // increment count of Left bracket
                countLeft++;
                if(imbalance > 0)
                {
                    // swaps count is last swap count + total
                    // number imbalanced brackets
                    swap += imbalance;
                    // imbalance decremented by 1 as it solved
                    // only one imbalance of Left and Right
                    imbalance--;
                }
```

```
            } else if(chars[i] == ']' )
            {
                // increment count of Right bracket
                countRight++;
                // imbalance is reset to current difference
                // between Left and Right brackets
                imbalance = (countRight-countLeft);
            }
        }
        return swap;
    }

// Driver code
    public static void main(String args[])
    {
        String s = "[]][][";
        System.out.println(swapCount(s) );

        s = "[[][]]";
        System.out.println(swapCount(s) );

    }
}
// This code is contributed by Janmejaya Das.
```

Python3

```
# Python3 program to count swaps required to
# balance string
def swapCount(s):

    chars = s

    # Stores total number of left and
    # right brackets encountered
```

```
    countLeft = 0
    countRight = 0

    # Swap stores the number of swaps
    # required imbalance maintains the
    # number of imbalance pair
    swap = 0
    imbalance = 0;

    for i in range(len(chars)):
        if chars[i] == '[':

            # Increment count of left bracket
            countLeft += 1

            if imbalance > 0:

                # Swaps count is last swap
                # count + total number
                # imbalanced brackets
                swap += imbalance

                # Imbalance decremented by 1
                # as it solved only one
                # imbalance of left and right
                imbalance -= 1

        elif chars[i] == ']':

            # Increment count of right bracket
            countRight += 1

            # Imbalance is reset to current
            # difference between left and
            # right brackets
            imbalance = (countRight - countLeft)
```

```
    return swap

# Driver code
s = "[]][][";
print(swapCount(s))

s = "[[][]]";
print(swapCount(s))

# This code is contributed by Prateek Gupta
```

C#

```
// C# Program to count swaps required
// to balance string
using System;

class GFG
{

public static long swapCount(string s)
{
    char[] chars = s.ToCharArray();

    // stores the total number of Left and
    // Right brackets encountered
    int countLeft = 0, countRight = 0;

    // swap stores the number of swaps
    // required imbalance maintains the
    // number of imbalance pair
    int swap = 0, imbalance = 0;

    for (int i = 0; i < chars.Length; i++)
    {
```

```
        if (chars[i] == '[')
        {
            // increment count of Left bracket
            countLeft++;
            if (imbalance > 0)
            {
                // swaps count is last swap count + total
                // number imbalanced brackets
                swap += imbalance;

                // imbalance decremented by 1 as it solved
                // only one imbalance of Left and Right
                imbalance--;
            }
        }
        else if (chars[i] == ']')
        {
            // increment count of Right bracket
            countRight++;

            // imbalance is reset to current difference
            // between Left and Right brackets
            imbalance = (countRight - countLeft);
        }
    }
    return swap;
}

// Driver code
public static void Main(string[] args)
{
    string s = "[]][][";
    Console.WriteLine(swapCount(s));

    s = "[[][]]";
    Console.WriteLine(swapCount(s));
}
```

```
}

// This code is contributed by Shrikant13
```

Javascript

```
<script>
    // Javascript Program to count swaps required
    // to balance string

    function swapCount(s)
    {
        let chars = s.split('');

        // stores the total number of Left and
        // Right brackets encountered
        let countLeft = 0, countRight = 0;

        // swap stores the number of swaps
        // required imbalance maintains the
        // number of imbalance pair
        let swap = 0, imbalance = 0;

        for (let i = 0; i < chars.length; i++)
        {
            if (chars[i] == '[')
            {
                // increment count of Left bracket
                countLeft++;
                if (imbalance > 0)
                {
                    // swaps count is last swap count + total
                    // number imbalanced brackets
                    swap += imbalance;
```

```
                    // imbalance decremented by 1 as it solved
                    // only one imbalance of Left and Right
                    imbalance--;
                }
            }
            else if (chars[i] == ']')
            {
                // increment count of Right bracket
                countRight++;

                // imbalance is reset to current difference
                // between Left and Right brackets
                imbalance = (countRight - countLeft);
            }
        }
        return swap;
    }

      let s = "[]][][";
    document.write(swapCount(s) + "</br>");

    s = "[[][]]";
    document.write(swapCount(s));

    // This code is contributed by suresh07.
</script>
```

Output:

```
2
0
```

This article is contributed by **Aditya Kamath**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.
Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Attention reader! Don't stop learning now. Get hold of all the important DSA concepts with the **DSA Self Paced Course** at a student-friendly price and become industry ready. To complete your preparation from learning a language to DS Algo and many more, please refer **Complete Interview Preparation Course**.

In case you wish to attend live classes with industry experts, please refer **Geeks Classes Live**

Like 26

Next
Fitting Shelves Problem

RECOMMENDED ARTICLES

Page : 1 2 3

01 Find index of closing bracket for a given opening bracket in an expression
23, Nov 17

02 Check if the bracket sequence can be balanced with at most one change in the position of a bracket
17, Sep 19

03 Find numbers of balancing positions in string
17, Jul 17

04 Minimum number of bracket reversals needed to make an expression balanced | Set - 2
05, Mar 19

05 Minimum Cost required to generate a balanced Bracket Sequence
21, Aug 20

06 Minimum number of bracket reversals needed to make an expression balanced
27, Oct 15

07 Number of closing brackets needed to complete a regular bracket sequence
15, Nov 18

08 Print Bracket Number
26, Feb 18

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Medium

Easy Normal Medium Hard Expert

Improved By : JANMEJAYA DAS, shrikanth13, PrateekGupta10, sanchayang98, avanitrachhadiya2155, divyeshrabadiya07, divyesh072019, rag2127, suresh07, susmitakundugoaldanga

Article Tags : Greedy, Strings

Practice Tags : Strings, Greedy

Improve Article

Report Issue

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

Load Comments