Number of subsequences of the form a^i b^j c^k

Given a string, count number of subsequences of the form aibjck, i.e., it consists of i ’a’ characters, followed by j ’b’ characters, followed by k ’c’ characters where i >= 1, j >=1 and k >= 1.

**Note:** Two subsequences are considered different if the set of array indexes picked for the 2 subsequences are different.

Expected Time Complexity: O(n)

**Examples:**

**Input :** abbc

**Output :** 3

Subsequences are abc, abc and abbc

**Input :** abcabc

**Output :** 7

Subsequences are abc, abc, abbc, aabc

abcc, abc and abc

[**Recommended: Please solve it on “*PRACTICE*” first, before moving on to the solution.**](https://practice.geeksforgeeks.org/problems/count-subsequences-of-type-ai-bj-ck/0)

**Approach:**  
We traverse given string. For every character encounter, we do the following:

**1)** Initialize counts of different subsequences caused by different combination of ‘a’. Let this count be aCount.

**2)** Initialize counts of different subsequences caused by different combination of ‘b’. Let this count be bCount.

**3)** Initialize counts of different subsequences caused by different combination of ‘c’. Let this count be cCount.

**4)** Traverse all characters of given string. Do following for current character **s[i]**  
    **If current character is ‘a’**, then there are following possibilities :  
    a) Current character begins a new subsequence.  
    b) Current character is part of aCount subsequences.  
    c) Current character is not part of aCount subsequences.  
    Therefore we do aCount = (1 + 2 \* aCount);

**If current character is ‘b’**, then there are following possibilities :  
    a) Current character begins a new subsequence of b’s with aCount subsequences.  
    b) Current character is part of bCount subsequences.  
    c) Current character is not part of bCount subsequences.  
    Therefore we do bCount = (aCount + 2 \* bCount);

**If current character is ‘c’**, then there are following possibilities :  
    a) Current character begins a new subsequence of c’s with bCount subsequences.  
    b) Current character is part of cCount subsequences.  
    c) Current character is not part of cCount subsequences.  
    Therefore we do cCount = (bCount + 2 \* cCount);

**5)** Finally we return cCount;

**Explanation of approach with help of example:**

* aCount is the number of subsequences of the letter ‘a’.
* Consider this example: aa.
* We can see that aCount for this is 3, because we can choose these possibilities: (xa, ax, aa) (x means we did not use that character). Note also that this is independent of characters in between, i.e. the aCount of aa and ccbabbbcac are the same because both have exactly 2 a’s.
* Now, adding 1 a, we now have the following new subsequences: each of the old subsequences, each of the old subsequences + the new a, and the new letter a, alone. So a total of **aCount + aCount + 1** subsequences.
* Now, let’s consider bCount, the number of subsequences with some a’s and then some b’s. in ‘aab’, we see that bCount should be 3 (axb, xab, aab) because it is just the number of ways we can choose subsequences of the first two a’s, and then b. So every time we add a b, the number of ways increases by aCount.
* Let’s find bCount for ‘aabb’. We have already determined that aab has 3 subsequences, so certainly we still have those. Additionally, we can add the new b onto any of these subsequences, to get 3 more. Finally, we have to count the subsequences that are made without using any other b’s, and by the logic in the last paragraph, that is just aCount. So, bCount after this is just the old bCount\*2 + aCount;
* cCount is similar.

Solution:

// C++ program to count subsequences of the

// form a^i b^j c^k

#include <bits/stdc++.h>

using namespace std;

// Returns count of subsequences of the form

// a^i b^j c^k

int countSubsequences(string s)

{

    // Initialize counts of different subsequences

    // caused by different combination of 'a'

    int aCount = 0;

    // Initialize counts of different subsequences

    // caused by different combination of 'a' and

    // different combination of 'b'

    int bCount = 0;

    // Initialize counts of different subsequences

    // caused by different combination of 'a', 'b'

    // and 'c'.

    int cCount = 0;

    // Traverse all characters of given string

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

        /\* If current character is 'a', then

           there are the following possibilities :

             a) Current character begins a new

                subsequence.

             b) Current character is part of aCount

                subsequences.

             c) Current character is not part of

                aCount subsequences. \*/

        if (s[i] == 'a')

            aCount = (1 + 2 \* aCount);

        /\* If current character is 'b', then

           there are following possibilities :

             a) Current character begins a new

                subsequence of b's with aCount

                subsequences.

             b) Current character is part of bCount

                subsequences.

             c) Current character is not part of

                bCount subsequences. \*/

        else if (s[i] == 'b')

            bCount = (aCount + 2 \* bCount);

        /\* If current character is 'c', then

           there are following possibilities :

             a) Current character begins a new

                subsequence of c's with bCount

                subsequences.

             b) Current character is part of cCount

                subsequences.

             c) Current character is not part of

                cCount subsequences. \*/

        else if (s[i] == 'c')

            cCount = (bCount + 2 \* cCount);

    }

    return cCount;

}

// Driver code

int main()

{

    string s = "abbc";

    cout << countSubsequences(s) << endl;

    return 0;

}

**Complexity Analysis:**

* **Time Complexity:** O(n).  
  One traversal of the string is needed.
* **Auxiliary Space:** O(1).  
  No extra space is needed.
* Here is my attempt in explaining this solution:  
    
  To understand it better, Lets take an example : s = "aaabbcc"  
    
  1. Let's take 3 variables aCount, bCount, and cCount. where,  
  aCount stores total number of subsequences so far ending with 'a'  
  bCount stores total number of subsequences so far ending with 'b'  
  cCount stores total number of subsequences so far ending with 'c'  
    
  initialize all 3 variables to zero, aCount = bCount = cCount = 0,   
  as initially we have 0 subsequences ending with 'a', 'b', and 'c'  
    
  2. Iterate through all characters of the input string  
  **when i = 0, s[i] = 'a'**  
  aCount = 1  
    
  **when i = 1, s[i] = 'a'**  
  What are the different subsequences ending with 'a' now ? They are as below:  
  a (at index 0)  
  aa (index 0, index 1)  
  a (at index 1)  
    
  so aCount = 3  
    
  Now what can this be summarized into:  
  1. first 'a' is from previous subsequences ending with 'a', i.e., when we don't consider current 'a'   
   as part of subsequence, and that count we already have stored in aCount.  
  2. second 'aa' is when we append current 'a' to previous total number of subsequences ending   
   with 'a', which we already have stored in aCount.  
  3. third 'a' is when we consider current 'a' as the beginning of a new subsequence, so this   
   contributes 1 to new total number of subsequences ending with 'a'.  
    
  so updated aCount = (1) + (2) + (3) = aCount + aCount + 1 = 2 \* aCount + 1 = 3  
    
  **when i = 2, s[i] = 'a'**  
  What are the different subsequences ending with 'a' now ? They are as below:  
  1. a, a, aa (from previous subsequences ending with 'a', i.e., when we don't consider current 'a'   
   as part of a subsequence, and that count we already have stored in aCount.)  
  2. aa, aa, aaa (when we append current 'a' to previous total number of subsequences ending   
   with 'a', which we already have stored in aCount.)  
  3. a (when we consider current 'a' as the beginning of a new subsequence, so this contributes   
   1 to new total number of subsequences ending with 'a'.)  
    
  so updated aCount = (1) + (2) + (3) = aCount + aCount + 1 = 2 \* aCount + 1 = 7  
    
  So generalized formula to calculate aCount is **aCount = 2 \* aCount + 1**  
    
    
    
  **when i = 3, s[i] = 'b'**  
  What are the different subsequences ending with 'b' ? They are as below:  
  ab, ab, aab, aab, aab, aaab, ab   
  (this is when we append current 'b' to previous total number of subsequences ending   
  with 'a', which we already have stored in aCount)  
    
  therefore bCount = aCount = 7  
    
  **when i = 4, s[i] = 'b'**  
  What are the different subsequences ending with 'b' now ? They are as below:  
  1. ab, ab, aab, aab, aab, aaab, ab (this is when we append current 'b' to previous total number   
   of subsequences ending with 'a', which we already have stored in aCount)  
  2. abb, abb, aabb, aabb, aabb, aaabb, abb (this is when we append current 'b' to previous total   
   number of subsequences ending with 'b', which we already have stored in bCount)  
  3. ab, ab, aab, aab, aab, aaab, ab (this is from previous total subsequences ending with 'b', i.e.,   
   when we don't consider current 'b' as part of a subsequence, and that count we already have   
   stored in bCount.)  
    
  so updated bCount = (1) + (2) + (3) = aCount + 2 \* bCount = 7 + 2\*7 = 21  
    
  So generalized formula to calculate bCount is **bCount = aCount + 2 \* bCount**  
    
    
    
  **when i = 5, s[i] = 'c'**  
  What are the different subsequences ending with 'c' ? They are as below:  
  abc, abc, aabc, aabc, aabc, aaabc, abc, abbc, abbc, aabbc, aabbc, aabbc, aaabbc, abbc, abc,   
  abc, aabc, aabc, aabc, aaabc, abc  
  (this is when we append current 'c' to previous total number of subsequences ending with 'b',   
  which we already have stored in bCount)  
    
  therefore cCount = bCount = 21  
    
  **when i = 6, s[i] = 'c'**  
  What are the different subsequences ending with 'c' now ? They are as below:  
  1. abc, abc, aabc, aabc, aabc, aaabc, abc, abbc, abbc, aabbc, aabbc, aabbc, aaabbc, abbc, abc,   
  abc, aabc, aabc, aabc, aaabc, abc  
  (this is when we append current 'c' to previous total number of subsequences ending with 'b',   
  which we already have stored in bCount)  
    
  2. abcc, abcc, aabcc, aabcc, aabcc, aaabcc, abcc, abbcc, abbcc, aabbcc, aabbcc, aabbcc, aaabbcc,   
  abbcc, abcc, abcc, aabcc, aabcc, aabcc, aaabcc, abcc   
  (this is when we append current 'c' to previous total number of subsequences ending with 'c',  
  which we already have stored in cCount)  
    
  3. abc, abc, aabc, aabc, aabc, aaabc, abc, abbc, abbc, aabbc, aabbc, aabbc, aaabbc, abbc, abc, abc,   
  aabc, aabc, aabc, aaabc, abc  
  (this is from previous total subsequences ending with 'c', i.e., when we don't consider current 'c' as part   
  of a subsequence, and that count we already have stored in cCount.)  
    
  so updated cCount = (1) + (2) + (3) = bCount + 2 \* cCount = 21 + 2\*21 = 63  
    
  So generalized formula to calculate cCount is **cCount = bCount + 2 \* cCount**  
    
  Final count is cCount = 63.