Problem 01: ***Sharmeen loves Mozahid loves Sharmeen***

Setter : Md. Mozahidul Islam Bhuiyan(kissu_pari_na)
Alternate:
1. Avik Sarkar(ssavi)
2. Reajul Haque Reayz(reayz)

Setter: Md. Mozahidul Islam Bhuiyan

Tag: Implementation.
Complexity: O(N)
Description: Use only one loop. Suppose the loop is iterated by i. We have to keep three things in mind. One is the count of ‘s’(cnt_s) at any moment, another is the count of subsequence of ‘sm’(cnt_sm) at any moment and the last one is the cnt of subsequence of ‘sms’(cnt_sms). When iterating through the string if s[i] = ‘s’ then we will increase cnt_s by 1 and also increase cnt_sms by cnt_sm at that time, as all subsequence of ‘sm’ till ith position can make a subsequence of ‘sms’ using this ‘s’. If s[i] = ‘m’, then we will increase cnt_sm by cnt_s, as all ‘s’ from position 1 to i will make a subsequence with this ‘m’. At last after iterating the whole string we have to output the cnt_sms.

Code: https://paste.ubuntu.com/p/s4j8WTVXXC/

Alternate 1: Avik Sarkar

Tag: Implementation.
Complexity: O(N)
Description: Take two array. The first one stores the amount of *‘s’* so far and the second one stores the amount of *‘m*’ so far from **0 - |S|** , where **|S|** is the length of string **S**. Now for each of i^{th} position that contains *‘m’* , retrieve number of *‘s’* before and after that position. Multiply amount of *‘s’* retrieved before i^{th} position and after i^{th} position. And sum up the multiplied result for each of the *‘m’* . That will be the final answer. For example, S = “*smsmss*” . For *‘m’* at position **2**, we have number of “*sms*” is **(1 x 3) = 3** and for position **4**, we have number of “*sms*” is **(2 x 2) = 4**. In total, we can make 7 “*sms*” .

Alternate 2: Reajul Haque Reayz
Code: https://paste.ubuntu.com/p/5TY46jsTdw/

Problem 02: ***Sharmeen loves substring***

Setter : Md. Mozahidul Islam Bhuiyan(kissu_pari_na)
Alternate: Avik Sarkar(ssavi)

Setter: Md. Mozahidul Islam Bhuiyan

Tag: Implementation.
Complexity: O(N+Q)
Description: Suppose, F(“x” , {L,R}) = number of subsequence of “x” from position L to R of a string.

First you have to observe that,

F(“sms” , {L,R}) = + F(“sms” , {1,R})
- F(“sms”,{1,L-1})
- (F(“s”,{1,L-1}) * F(“ms”,{L,R}))
- (F(“sm”,{1,L-1}) * F(“s”,{L,R}))

Means if we substruct the following 3 things

1. Number of subsequence of ‘sms’ from position 1 to L - 1
2. Multiplication of (Number of subsequence of ‘s’ from position 1 to L - 1) && (Number of subsequence of ‘ms’ from position L to R)
3. Multiplication of (Number of subsequence of ‘sm’ from position 1 to L - 1) && (Number of subsequence of ‘s’ from position L to R)

From (Number of subsequence of ‘sms’ from position 1 to R)
Then we can find the Number of subsequence of ‘sms’ from position L to R

How to calculate

1. F(“s” , {L,R})
2. F(“sm” , {1,any_pos})
3. F(“ms” , {L,R})
4. F(“sms” , {1,any_pos})

1 can be easily be calculated by cumulative of ‘s’.

2 can be solved using only one loop through the string. for(i = 1 ; i <= string_length ; i ++). In this we will keep two things: number of ‘s’ at any moment and number of total ‘sm’ at that moment. At any moment if s[i] == ‘s’, then we will increase the value of cnt_s. If s[i] == ‘m’, then we will increase cnt_sm by cnt_s, meant we will do cnt_sm += cnt_s, as all the ‘s’ left this i position will make a sequence of ‘sm’ using this ‘m’.

3 can be solved : F(“ms” , {L,R}) = F(“ms” , {1,R})
- F(“ms” , {1,L-1})
- F(“m” , {1,L-1}) * F(“s” , {L,R})

4 can be solved : Use only one loop. Suppose the loop is iterated by i. We have to keep three things in mind. One is the count of ‘s’(cnt_s) at any moment, another is the count of subsequence of ‘sm’(cnt_sm) at any moment and the last one is the cnt of subsequence of ‘sms’(cnt_sms). When iterating through the string if s[i] = ‘s’ then we will increase cnt_s by 1 and also increase cnt_sms by cnt_sm at that time, as all subsequence of ‘sm’ till ith position can make a subsequence of ‘sms’ using this ‘s’. If s[i] = ‘m’, then we will increase cnt_sm by cnt_s, as all ‘s’ from position 1 to i will make a subsequence with this ‘m’. For each position save the cnt_sms in an array.

Code: https://paste.ubuntu.com/p/kvZqnPp687/

Alternate: Avik Sarkar

Tag: Implementation.
Complexity: O(N+Q)
Description: Calculate the cumulative sum of amount
f ‘*s*’ through **0 - |S|** where **|S|** is the length of string. Let the cumulative sum array will be denoted as **A[]**. For any range **L…R** the amount of “sms” can be found by following rule –

```
For each index k that contains ‘m’ between L – R
      totalSMS = totalSMS + (A[k] - A[L-1]) * (A[R] - A[k])
```

If we look the formula thoroughly –

```
For a single ‘m’ the calculation is –
      (A[k] - A[L-1]) * (A[R] - A[k])
      = A[k] * (A[R] + A[L-1]) - A[k]^2 - A[R] * A[L-1]
      = A[k] * (Sum of A[L-1] and A[R]) - A[k]^2 - (Product of A[L-1] and A[R])
```

Now the point is we have to use the formula shown above for each appearance of ‘m’ which is k. So it can be written as –

```
  S = Sum(A[k] for each appearance of ‘m’) * (Sum of A[L-1] and A[R]) - Sum(A[k]² for
each appearance ‘m’) – (Product of A[L-1] and A[R]) * Amount_Of_‘m’_between_L_And_R.
```

Now, Sum of A[k] and Sum of $A[k]^2$ for each appearance of ‘m’ can be pre-calculated. Now for each query the amount of “*sms*” can easily be calculated at O(1).

Illustration: For a given string “smssmms’ and range **3 7**

```
  s m s s m m s  //string
0 1 1 2 3 3 3 4  //A[],amount of s
0 0 1 1 1 4 7 7  //B[],sum of A[k] for each appearance of ‘m’
0 0 1 1 1 10 19 19  // C[],sum of A[k]² for each appearance of ‘m’
0 0 1 1 1 2 3 2  // D[], amount of ‘m’
```

```
Result = (B[R]-B[L-1])*(A[R]+A[L-1])-(C[R]-C[L-1]) - A[R] * A[L-1] * (D[R] – D[L-1])
       = (7-1) * (4 + 1) - (19 - 1) - 4 * 1 * 2 = 4
```

Code: https://paste.ubuntu.com/p/DXGHW6vBKX/

Problem 03: *Can Sharmeen solve it?*

Setter : Md. Mozahidul Islam Bhuiyan(kissu_pari_na)
Alternate: Tahsin Masrur(Zeronfinity)

Setter: Md. Mozahidul Islam Bhuiyan

Tag: Binary Search , Two Pointer
Complexity: O(NlogN) [**Binary Search**] , O(N) [**Two Pointer**]
Description: Suppose, F(“x” , {L,R}) = number of subsequence of “x” from position L to R of a string.

First you have to observe that,

F(“sms” , {L,R}) = + F(“sms” , {1,R})
- F(“sms”,{1,L-1})
- (F(“s”,{1,L-1}) * F(“ms”,{L,R}))

- (F(“sm”,{1,L-1}) * F(“s”,{L,R}))

Means if we subtruct the following 3 things

1. Number of subsequence of ‘sms’ from position 1 to L - 1
2. Multiplication of (Number of subsequence of ‘s’ from position 1 to L - 1) && (Number of subsequence of ‘ms’ from position L to R)
3. Multiplication of (Number of subsequence of ‘sm’ from position 1 to L - 1) && (Number of subsequence of ‘s’ from position L to R)

From (Number of subsequence of ‘sms’ from position 1 to R)
Then we can find the Number of subsequence of ‘sms’ from position L to R

If we can do the above things then we can solve this problem using two approach:

1. **Binary Search(O(NlogN) complexity)**
2. **Two Pointer(O(N) complexity)**

Binary Search Approach:

From each position(Suppose i) of the string we will apply binary search where left = i, right = string_length and we will try to make x subsequences of ‘sms’. If count of ‘sms’ from i to mid (F(“sms”,{i,mid})) is greater than x then we will change the right to mid - 1. If count of ‘sms’ is less than x then we will change the right to mid + 1. If count of ‘sms’ is equal to x then we will store it as a result and change the left to mid + 1. In this way we can find the largest substring possible starting from position i that contains x subsequence of ‘sms’ if possible. We will do it for every position of the string and will find the largest one. Now you can easily write the rest of the code, I hope so :)

Two Pointer Approach:

Take two variable left_pointer and right_pointer and make their initial value is equal 1. Now find F(“sms”,{left_pointer , right_pointer }) means total subsequence of ‘sms’ from position left_pointer to position right_pointer. If it is equal to x then this substring(substring starts at position left_pointer and ends at position right_pointer) can be a answer and we will try to make it larger increasing the right_pointer by 1.If total subsequence of ‘sms’ is greater than x then we will increase the left_pointer. Continue this process until (left_pointer > right_pointer) or (right pointer > n(length of the substring)).

How to calculate

1. F(“s” , {L,R})
2. F(“sm” , {1,any_pos})
3. F(“ms” , {L,R})
4. F(“sms” , {1,any_pos})

1 can be easily be calculated by cumulative of ‘s’.

2 can be solved using only one loop through the string. for(i = 1 ; i <= string_length ; i ++). In this we will keep two things: number of ‘s’ at any moment and number of total ‘sm’ at that moment. At any moment if s[i] == ‘s’, then we will increase the value of cnt_s. If s[i] == ‘m’, then we will increase cnt_sm by cnt_s, meant we will do cnt_sm += cnt_s, as all the ‘s’ left this i position will make a sequence of ‘sm’ using this ‘m’.

3 can be solved : F(“ms” , {L,R}) =   F(“ms” , {1,R})
- F(“ms” , {1,L-1})
- F(“m” , {1,L-1}) * F(“s” , {L,R})

4 can be solved : Use only one loop. Suppose the loop is iterated by i. We have to keep three things in mind. One is the count of ‘s’(cnt_s) at any moment, another is the count of subsequence of ‘sm’(cnt_sm) at any moment and the last one is the cnt of subsequence of ‘sms’(cnt_sms). When iterating through the string if s[i] = ‘s’ then we will increase cnt_s by 1 and also increase cnt_sms by cnt_sm at that time, as all subsequence of ‘sm’ till ith position can make a subsequence of ‘sms’ using this ‘s’. If s[i] = ‘m’, then we will increase cnt_sm by cnt_s, as all ‘s’ from position 1 to i will make a subsequence with this ‘m’. For each position save the cnt_sms in an array.

Code :

1. Binary Search : https://paste.ubuntu.com/p/wBPPddRvbS/
2. Two Pointer : https://paste.ubuntu.com/p/CGC5PfBwQy/

Alternate: Tahsin Masrur

Tag: Two Pointer
Complexity: O(N)
Description: Let’s solve for the lower constraints subtask first. The idea is pretty bruteforce-ish. We will iterate over all substrings of the given string and find the number of “sms” subsequences in the substring. If it’s equal to X, we will compare it with our current answer. To iterate over all substrings, we will need O(n^2) complexity. Now how to efficiently find the number of desired subsequences in a substring?

Suppose, the substring is “ssymzmsms”. The number of “sms” subsequence here is 11. We can find it while building the substring itself. Let’s iterate over the characters of the substring to demonstrate that.

1st character - s:
f(s) = 1, f(s) denotes number of “s” so far.

2nd character - s:
f(s) = 2

3rd character - y:
No change needed

4th char - m:
f(s) = 2 (from previous characters)

How many “sm” subsequences do we have now then? We can append this new “m” to our 2 previous “s”.
So, f(sm) += f(s) = 2

5th char - z:
No change

6th char - m:
f(s) = 2
f(sm) += f(s) = 4

7th char - s:
From previous chars, we have f(sm) = 4. So we can append our new “s” to them to get “sms”.
f(sms) += f(sm) = 4
f(s) = 3
f(sm) = 4

8th char - m:
f(sm) += f(s) = 7
f(s) = 3
f(sms) = 4

9th char - s:
f(sms) += f(sm) = 11
f(s) = 4
f(sm) = 7

So, the number of “sms” subsequences is 11 in our substring. We can find it while building a substring itself.

Now how to do this efficiently for $n <= 10^5$?

The technique we will use is known as two pointers. Let’s say we have already found number of “sms” subsequences for [1, 5]. For “ssymzmsms”, it’s 0. Say, X = 1.

Now, let L = 1, R = 5. These are the two pointers, but don’t let the name scare you, they are just two normal variables pointing to the two boundaries of current substring.

Now, since f(sms) < X now, we need to make our current substring bigger, don’t we? But our answer substring might start from L, or it might not, we don’t know. But we are sure it’s end point is bigger than R, as f(sms) < X. If it wasn’t, f(sms) would be >= X, wouldn’t it be?

So, we will first try to increase R as long as f(sms) < X. We can find f(sms) while doing that same as before.

Now, suppose R = 7 now. So, f(sms) = 4 now. That’s larger than X. So what can we do? We can now increase L to decrease our substring in hope of decreasing f(sms).

It’s like this.

At first, L = 1, R = 1. f(sms) = 0 < X. So we increase R.
Now, L = 1, R = 2. f(sms) = 0 < X. Increase R.
L = 1, R = 3. f(sms) = 0 < X. Increase R.
L = 1, R = 4. f(sms) = 0 < X. Increase R.
L = 1, R = 5. f(sms) = 0 < X. Increase R.
L = 1, R = 6. f(sms) = 0 < X. Increase R.
L = 1, R = 7. f(sms) = 4 > X. Increase L.
L = 2, R = 7. f(sms) = 2 > X. Increase L.
L = 3, R = 7. f(sms) = 0 < X. Increase R.
L = 3, R = 8. f(sms) = 0 < X. Increase R.
L = 3, R = 9. f(sms) = 1 = X. This is a valid range (the only one in this substring). So consider it as an answer.

Now, how to find f(sms) when you increase L? Not that hard, similar to the concept of when you increase R.

Suppose L’th character is “s”. Then removing it will decrease f(sms) by f(ms), won’t it? So in this case, do f(sms) -= f(ms). It will also decrease f(sm) by f(m)

When L’th character is “m”, think a little yourself first.

Alter’s Code: https://paste.ubuntu.com/p/HC7tyd6J63/
Here, smsf denotes f(sms)