1 An Easy Problem

For a string t, let $endpos(t) = \{p \mid p \geq |t|, S[p-|t|+1, p] = t\}$.

1.1 Algorithm 1

Simulate according to the problem's description. Time complexity is $O(qn^2)$, passes Subtask 1 and gets a total of 5 points.

1.2 Algorithm 2

We create an SAM (suffix automation) consisting of S and all of s_i, and create a segment tree consisting of the set of endpos'es for every vertex.

For every query of form L_i, R_i, s_i, we look for the corresponding vertex of s_i on the SAM. We iterate from L_i to R_i to enumerate $S[l_k, r_k]$. By querying the number of elements in the segment $[l_k + |s_i| - 1, r_k]$ we can find out the value of $c(s_i, S[l_k, r_k])$.

Time complexity is $O(qn \log n)$, passes Subtask 1, Subtask 2, Subtask 4 for a total of 30 points.

1.3 Algorithm 3

$$\sum_{k=L_i}^{R_i} c(s_i, S[l_k, r_k]) = \sum_{k \leq R_i} c(S_i, S[l_k, r_k]) - \sum_{k < L_i} c(s_i, S[l_k, r_k])$$

With the above equation we are able to split one query into two prefix queries where $L_i = 1$. From here on we will focus on the $L_i = 1$ case.

Again, we create an SAM (suffix automation) consisting of S and all of s_i. In ascending order of i, we add $S[l_i, r_i]$ into some data structure, such that for each vertex in the SAM, we maintain the total number of occurrences of the string corresponding to that vertex in previously added $S[l_i, r_i]$'s. Since all s_i's appear in SAM, it would be sufficient to maintain just the vertices in the SAM. In this way, after adding $S[l_i, r_i]$ we are able to answer queries satisfying $L = 1, R = i$ directly.

We define $tr(t)$ as follows: if t happens to be the longest string corresponding to some vertex, then $tr(t)$ equals that vertex, otherwise $tr(t)$ is the parent of that vertex.

Now let's consider how to maintain the number of occurrences for each vertex. The number of occurrences is initially 0 for each vertex. Let's consider how to update that number while adding $S[l_i, r_i]$. We consider the so-called "parent tree" of the SAM, which is a tree where the parent of every vertex is equal to its suffix link . For every prefix of $S[l_i, r_i]$, which is $S[l_i, p]$, we increase the number by 1 on all the ancestors of $tr(S[l_i, p])$ on the parent tree. This can be done with a combination of Fenwick Tree and DFS order, where we increase the value of the vertex by one in the Fenwick Tree, and query for the sum of the subtree.

1.4 Algorithm 4

For the sake of convenience, we assume $q, |S|, \sum|s_i| \leq n$.

For queries satisfying $|s_i| > \sqrt{n}$, we find out the endpos set of s_i by brute force, and for each k in segment $[L_i, R_i]$ we count $c(s_i, S[l_k, r_k])$ by the prefix sum technique. Since the number of such queries cannot exceed $\sqrt{n}$, the total time complexity for these queries is $O(n\sqrt{n})$.

For queries satisfying $|s_i| \leq \sqrt{n}$,

$$\begin{aligned} c(s_i, S[l_k, r_k]) &= |endpos(s_i) \bigcap [l_k + |s_i| - 1, r_k]| \\ &= |endpos(s_i) \bigcap [l_k, r_k]| - |endpos(s_i) \bigcap [l_k, \min(r_k, l_k + |s_i| - 2)]| \end{aligned}$$

We maintain the contribution of two terms separately. We follow the techniques described in Algorithm 3 where we add $S[l_i, r_i]$ by order and maintain $\sum|endpos(t) \bigcap [l_i, r_i]|$ and $\sum|endpos(t) \bigcap [l_k, \min(r_k, l_k + |s_i| - 2)]|$ separately.

1.4.1 Maintaining $\sum|endpos(t) \bigcap [l_i, r_i]|$

After adding $S[l_k, r_k]$, for each p satisfying $l_k \leq p \leq r_k$ we need to increase the values of all ancestors of $tr(S[1, p])$. Let's consider how to maintain these increments.

We can project all prefix vertices on the SAM onto a plane, where the x coordinate equals the length of the prefix of the vertex and the y coordinate equals the DFS order of the vertex. We sort all vertices by x and y coordinates respectively, and for each axis we merge every consecutive $\sqrt{n}$ vertices into a single block. The blocks on y axis would support $O(1)$ increments and $O(\sqrt{n})$ queries of segment sum. For each block on the x axis, we maintain the number of times that the block has been incremented by one as a whole, and prepare an array of length n where the i-th element equals the number of vertices whose y coordinate doesn't exceed i in that block. For queries, we enumerate the blocks and calculate its contribution with the array and the number of that block. For updates, we push the marks on both endpoints onto the y axis blocks, and update the contributions on these blocks. For all x axis blocks in between we just need to mark it and increment them by one as a whole.

1.4.2 Maintaining $\sum|endpos(t)\bigcap[l_k, \min(r_k, l_k + |s_i| - 2)]|$

After adding $S[l_k, r_k]$, for each p satisfying $l_k \leq p \leq \min(r_k, l_k + \sqrt{n})$ we need to increase the suffices of $S[1, p]$ whose length is greater than $p - l_k + 1$ by one. This is equivalent to increasing all ancestors of $tr(S[1, p])$ by one and decreasing all ancestors of $tr(S[l_k, p])$ by one. Similar to above, this can be converted into a problem about single-point updates and segment queries. By partitioning into $\sqrt{n}$ blocks we can achieve $O(1)$ updates and $O(\sqrt{n})$ queries. The time complexity of this part is $O(n\sqrt{n})$.

The standard program follows another approach. We enumerate a positive integer d (which does not exceed $\sqrt{n}$) and consider the contribution of each suffix $S[1, l_k + d - 1]$ (skipping if $l_k + d - 1 > r_k$). If a query string has length above d and is a suffix of $S[1, l_k + d - 1]$, the contribution is increased by one. We enumerate k in ascending order and maintain the contribution by partitioning SAM into $\sqrt{n}$ blocks with respect to the DFS order of all its vertices, answering only the queries where the string's length is above d. Since a query string s_i does not get processed more than $|s_i|$ times, the total complexity is $O(n\sqrt{n})$.

Time complexity of this method is $O(n\sqrt{n})$, passes all subtasks for a

total of 100 points.