MAC0345 - Desafios 2
Editorial Lista 2

Enrique Junchaya

April 20, 2023

A. Stammering Aliens.
Tópicos: Strings, Data structures

Iterando por intervalos de tamanho m no Suffix Array, temos m sufixos distintos do array ordenados alfabeticamente. Como qualquer substring é um prefixo de um sufixo da string, podemos ver que, para um intervalo qualquer do Suffix Array, a maior substring que aparece em todos os sufixos desse intervalo é dada pelo Longest Common Prefix (LCP) entre o primeiro e o último elemento do intervalo. Lembremos que o LCP do intervalo é o menor dos valores do vetor de LCP no intervalo. Assim, basta usar alguma estrutura de dados, por exemplo uma Segment Tree, para computar o mínimo do intervalo. Fazendo isso para todo intervalo, podemos encontrar o maior comprimento de uma substring da string dada.

Devemos tomar cuidado com o empate, pois para desempatar precisamos da substring mais à direita que atinge o máximo valor de repetições. Para conseguir isso, basta usar outra estrutura de dados (que pode ser outra Segment Tree) para devolver o índice mais à direita toda vez que encontramos um intervalo em que aparecem substrings de maior comprimento repetidas m vezes.

B. Burrows-Wheeler.
Tópicos: Strings

Seja S a string do input e $T = S + S$ a string formada concatenando duas vezes S. Veja que as rotações cíclicas de S estão ordenadas alfabeticamente no Suffix Array SA de T. Basta olhar apenas índices i tal que $SA[i] < \texttt{len}(S)$ para recuperar a ordem dessas rotações. Assim, basta apenas recuperar os últimos caráteres de tais rotações. Para isso, só temos que recuperar o caráter $T[SA[i] + \texttt{len}(S) - 1]$.

Devemos tomar cuidado com corner cases como strings que começam por espaços em branco, que só tem espaços em branco ou que estão compostas por apenas um caráter. Um detalhe importante é que o caráter \$ é maior que o espaço em branco no valor ASCII, então é melhor não usar \$ como o caráter especial do Suffix Array.

C. Cyclical Quest.
Tópicos: Strings

É possível resolver essa questão com Suffix Array, mas as soluções mais simples são muito lentas com essa estrutura. Mesmo assim, as soluções mais simples conseguem passar otimizando muito o código e usando algum template de Suffix Array $O(n)$. As soluções eficientes (que não precisam otimizações no código) com Suffix Array são muito complexas e provavelmente resolver a questão com Suffix Automata seja mais simples.

Seja A o Suffix Automata da string S. Considere uma consulta com a string x. Seja $T = x + x$. Como em outras questões, uma substring de T de tamanho $|x|$ é uma rotação de x. Assim, basta percorrer o suffix automata com a string T, adicionando um carátere na frente e tirando um do começo para manter sempre substrings de comprimento $|x|$. Note que para tirar um caráter de começo devemos usar os suffix links. Desta forma, estaremos visitando todos os estados do automata que representam as rotações de x.

Só falta detalhar como contar a quantidade de vezes em que aparece uma substring no Suffix Automata, mas para isso podemos usar fazez uma DP na árvore criada pelo suffix links, como foi mencionado em aula.

Um último detalhe que deve ser mencionado é que várias aparições de uma string nas rotações de x devem ser consideradas apenas uma vez, mas não é muito difícil lidar com isso.

Complexidade de tempo $O(N)$ onde N é a soma dos comprimentos de todas as strings que aparecem no enunciado.

D. String.
Tópicos: Strings

É fácil notar que as substrings aparecem na ordem alfabética no Suffix Array. Como $k \leq 10^5$ podemos simplesmente iterar pelas substrings nessa ordem. Assim, para cada posição do Suffix Array, vamos querer começar pelo primeiro substring que não tenha aparecido em posições anteriores. Note que tal substring para o índice i do Suffix Array é $SA[i]$ e tem comprimento igual a $lcp[i] + 1$. Portanto, basta começar a iterar desde essa posição até o final da string. Contudo, não devemos apenas contar as substrings distintas, senão também todas as aparições dela. Assim, lembrando que $k \leq 10^5$, podemos simplesmente iterar pelas seguintes posições do Suffix Array enquanto o lcp daquelas posições for maior ou igual ao comprimento da substring que estamos contando. Em aquelas posições estão todas as aparições da string.

Para resolver a questão, basta repetir esse processo até atingir a k-ésima string.

Desafio: resolva a questão para $k \leq 10^{10}$

E. Super Functional Strings.
Tópicos: Strings, Estruturas de dados, Brute Force, Matemática

O problema se simplifica fixando uma quantidade k distinta de letras. Como $1 \leq k \leq 26$, podemos iterar por todos os possíveis valores de k acumulando a resposta de cada solução fixando o valor dele.

Note que o problema pede para contar apenas os substring distintos. Podemos fazer isso com Suffix Array. Como vimos na solução da questão D, dada uma posição i do Suffix Array, dos substrings que começam em $SA[i]$, os repetidos são exatamente $lcp[i]$. Assim, os substrings que não aparecem em posições anteriores do Suffix Array são os que começam em $SA[i]$ e acabam entre $SA[i] + lcp[i]$ e n. Note que a quantidade de letras distintas desses substrings é não-decrescente. Desta forma, podemos fazer uma busca binária para encontrar o intervalo $[ini, fim]$ tal que os substrings que começam em $SA[i]$ e acabam numa posição desse intervalo tem exatamente k letras distintas. Com essa informação, podemos adicionar à resposta

$$\sum_{cur=ini}^{fim} (cur - SA[i] + 1)^k.$$

Podemos precalcular essas somas usando preffix sums das potências para cada k de 1 a 26. Assim, falta apenas detalhar como fazer as buscas binárias, ou seja, como saber quantas letras distintas tem uma substring. Como temos no máximo 26 letras distintas, uma forma de fazer isso é usar um bitmask onde cada bit representa uma letra diferente. Assim, podemos achar a resposta fazendo um OR dos bitmasks num segmento e contando quantos bits acesos tem esse bitmask. Contudo, fazer isso com uma Segment Tree é muito lento e daria TLE. Porém, note que ao fazer o OR de dois segmentos que se superpõem obtemos o OR da união de tais segmentos. Desta forma, podemos usar Sparse Table para responder em $O(1)$ as consultas de OR usando segmentos que se superpõem.

A complexidade total da solução é $O(26N \lg N)$.

F. A Story with Strings.
Tópicos: Strings, Estruturas de dados

Seja $T = S_1 + \$ + S_2$. Como vimos em aula, é fácil saber quais posições da Suffix Array de T correspondem a S_1 e quais a S_2. Assim, podemos iterar pelas posições i do Suffix Array de T que correspondem a S_2

mantendo dois ponteiros: um à última posição anterior a i que corresponde a S_1 e outro à primeira posição posterior a i que corresponde a S_1. Com isso, basta apenas calcular o LCP no segmento desde i até tais posições para saber a maior substring que aparece em S_1 e em S_2 e que em S_2 começa na posição i.

Para calcular o LCP num intervalo podemos usar uma Segment Tree como explicado na solução da questão A.