Suffix Array

Aplicaciones de Suffix Array y LCP

Mateo Carranza Velez

Universidad Nacional de Córdoba - FAMAF

Training Camp 2024

Gracias Sponsors!

Organizador

Diamond

Gold

¿Qué es un Suffix Array?

- Es un arreglo ordenado de los sufijos de un string.

¿Qué es un Suffix Array?

- Es un arreglo ordenado de los sufijos de un string.
- Por ejemplo, para el string `"banana"`, sus sufijos son:
  `["banana", "anana", "nana", "ana", "na", "a", ""]`

¿Qué es un Suffix Array?

- Es un arreglo ordenado de los sufijos de un string.
- Por ejemplo, para el string `"banana"`, sus sufijos son:
  `["banana", "anana", "nana", "ana", "na", "a", ""]`
- Si los ordenamos lexicográficamente obtenemos:
  `["", "a", "ana", "anana", "banana", "na", "nana"]`

¿Qué es un Suffix Array?

- Es un arreglo ordenado de los sufijos de un string.
- Por ejemplo, para el string `"banana"`, sus sufijos son:
  `["banana", "anana", "nana", "ana", "na", "a", ""]`
- Si los ordenamos lexicográficamente obtenemos:
  `["", "a", "ana", "anana", "banana", "na", "nana"]`
- Nos conviene solo guardar los índices de los sufijos:
  `[6, 5, 3, 1, 0, 4, 2]`

¿Cómo calcularlo eficientemente?

- El algoritmo es incremental: luego de k pasos, quedan ordenados los substrings de largo 2^k. En $\log n$ pasos se ordenan todos los sufijos.

¿Cómo calcularlo eficientemente?

- El algoritmo es incremental: luego de k pasos, quedan ordenados los substrings de largo 2^k. En $\log n$ pasos se ordenan todos los sufijos.
- Para el paso $2^k \to 2^{k+1}$ si ya tenemos identificados los strings de largo 2^k con un número entre 0 y la cantidad-1, con hacer un sort de pares es suficiente. Los iguales se identifican con numeros iguales.

¿Cómo calcularlo eficientemente?

- El algoritmo es incremental: luego de k pasos, quedan ordenados los substrings de largo 2^k. En $\log n$ pasos se ordenan todos los sufijos.
- Para el paso $2^k \to 2^{k+1}$ si ya tenemos identificados los strings de largo 2^k con un número entre 0 y la cantidad-1, con hacer un sort de pares es suficiente. Los iguales se identifican con numeros iguales.
- Por ejemplo, para el string `"banana."`, los substrings de largo 2 son: `["ba", "an", "na", "an", "na", "a.", "."]`

¿Cómo calcularlo eficientemente?

- El algoritmo es incremental: luego de k pasos, quedan ordenados los substrings de largo 2^k. En $\log n$ pasos se ordenan todos los sufijos.
- Para el paso $2^k \to 2^{k+1}$ si ya tenemos identificados los strings de largo 2^k con un número entre 0 y la cantidad-1, con hacer un sort de pares es suficiente. Los iguales se identifican con numeros iguales.
- Por ejemplo, para el string `"banana."`, los substrings de largo 2 son: `["ba", "an", "na", "an", "na", "a.", "."]`
- Y los identificadores son: `[3, 2, 4, 2, 4, 1, 0]`

¿Cómo calcularlo eficientemente?

- El algoritmo es incremental: luego de k pasos, quedan ordenados los substrings de largo 2^k. En $\log n$ pasos se ordenan todos los sufijos.
- Para el paso $2^k \to 2^{k+1}$ si ya tenemos identificados los strings de largo 2^k con un número entre 0 y la cantidad-1, con hacer un sort de pares es suficiente. Los iguales se identifican con numeros iguales.
- Por ejemplo, para el string `"banana."`, los substrings de largo 2 son: `["ba", "an", "na", "an", "na", "a.", "."]`
- Y los identificadores son: `[3, 2, 4, 2, 4, 1, 0]`
- Entonces los pares son: `[(3,4), (2,2), (4,4), (2,1), (4,0), (1,-1), (0,-1)]`

¿Cómo calcularlo eficientemente?

- El algoritmo es incremental: luego de k pasos, quedan ordenados los substrings de largo 2^k. En $\log n$ pasos se ordenan todos los sufijos.
- Para el paso $2^k \to 2^{k+1}$ si ya tenemos identificados los strings de largo 2^k con un número entre 0 y la cantidad-1, con hacer un sort de pares es suficiente. Los iguales se identifican con numeros iguales.
- Por ejemplo, para el string `"banana."`, los substrings de largo 2 son: `["ba", "an", "na", "an", "na", "a.", "."]`
- Y los identificadores son: `[3, 2, 4, 2, 4, 1, 0]`
- Entonces los pares son: `[(3,4), (2,2), (4,4), (2,1), (4,0), (1,-1), (0,-1)]`
- Y si volvemos a comprimir queda: `[4, 3, 6, 2, 5, 1, 0]`

¿Cómo calcularlo eficientemente? (Continuación)

- Si ordenamos los pares usando un algoritmo de ordenación basado en comparaciones, tenemos una complejidad de $\mathcal{O}(n(\log n)^2)$

¿Cómo calcularlo eficientemente? (Continuación)

- Si ordenamos los pares usando un algoritmo de ordenación basado en comparaciones, tenemos una complejidad de $\mathcal{O}(n(\log n)^2)$
- Si usamos radix sort para ordenar los pares, conseguimos una complejidad de $\mathcal{O}(n \log n)$.

Problema de ejemplo: Minimal Rotation

Minimal Rotation

A rotation of a string can be generated by moving characters one after another from beginning to end. For example, the rotations of acab are acab, caba, abac, and baca.
Your task is to determine the lexicographically minimal rotation of a string.
Constraints: $1 \leq n \leq 10^6$

`https://cses.fi/problemset/task/1110`

Problema de ejemplo: Minimal Rotation

Minimal Rotation

A rotation of a string can be generated by moving characters one after another from beginning to end. For example, the rotations of acab are acab, caba, abac, and baca.
Your task is to determine the lexicographically minimal rotation of a string.
Constraints: $1 \leq n \leq 10^6$

`https://cses.fi/problemset/task/1110`

Solución en el pizarrón.

Longest Common Prefix

- Es un arreglo que nos da el máximo prefijo en común entre los sufijos consecutivos en el suffix array.

Longest Common Prefix

- Es un arreglo que nos da el máximo prefijo en común entre los sufijos consecutivos en el suffix array.
- Por ejemplo, para el string `"banana."`, el suffix array es: `[".","a.","ana.","anana.","banana.","na.","nana."]`

Longest Common Prefix

- Es un arreglo que nos da el máximo prefijo en común entre los sufijos consecutivos en el suffix array.
- Por ejemplo, para el string `"banana."`, el suffix array es:
  `[".","a.","ana.","anana.","banana.","na.","nana."]`
- Y el arreglo de LCP es:
  `[0, 1, 3, 0, 0, 2]`

Longest Common Prefix

- Es un arreglo que nos da el máximo prefijo en común entre los sufijos consecutivos en el suffix array.
- Por ejemplo, para el string `"banana."`, el suffix array es:
  `[".","a.","ana.","anana.","banana.","na.","nana."]`
- Y el arreglo de LCP es:
  `[0, 1, 3, 0, 0, 2]`
- Observar que como están ordenados:
  $\text{LCP}(i,j) = \min(\text{LCP}(i,i+1), \text{LCP}(i+1,i+2), \ldots, \text{LCP}(j-1,j))$

Longest Common Prefix

- Es un arreglo que nos da el máximo prefijo en común entre los sufijos consecutivos en el suffix array.
- Por ejemplo, para el string `"banana."`, el suffix array es:
  `[".","a.","ana.","anana.","banana.","na.","nana."]`
- Y el arreglo de LCP es:
  `[0, 1, 3, 0, 0, 2]`
- Observar que como están ordenados:
  $\text{LCP}(i, j) = \min(\text{LCP}(i, i+1), \text{LCP}(i+1, i+2), \ldots, \text{LCP}(j-1, j))$
- Para calcular LCP de dos sufijos no consecutivos en el suffix array, podemos usar una sparse table.

¿Cómo calcularlo eficientemente?

Se puede calcular en $\mathcal{O}(n)$ observando que:

¿Cómo calcularlo eficientemente?

Se puede calcular en $\mathcal{O}(n)$ observando que:

- Si tenemos el LCP entre las posiciones i e $i+1$ del SA, al borrar una letra de cada uno el LCP nos da exctamente uno menos.

¿Cómo calcularlo eficientemente?

Se puede calcular en $\mathcal{O}(n)$ observando que:

- Si tenemos el LCP entre las posiciones i e $i+1$ del SA, al borrar una letra de cada uno el LCP nos da exctamente uno menos.
- Si j es la posición en el SA del sufijo que se obtiene al borrar un caracter del sufijo de la posición i y j' del de borrar $i+1$, entonces $j+1 \leq j'$

¿Cómo calcularlo eficientemente?

Se puede calcular en $\mathcal{O}(n)$ observando que:

- Si tenemos el LCP entre las posiciones i e $i+1$ del SA, al borrar una letra de cada uno el LCP nos da exctamente uno menos.
- Si j es la posición en el SA del sufijo que se obtiene al borrar un caracter del sufijo de la posición i y j' del de borrar $i+1$, entonces $j+1 \leq j'$
- $LCP(j, j+1) \geq LCP(j, j') = \max(0, LCP(i, i+1) - 1)$

¿Cómo calcularlo eficientemente?

Se puede calcular en $\mathcal{O}(n)$ observando que:

- Si tenemos el LCP entre las posiciones i e $i+1$ del SA, al borrar una letra de cada uno el LCP nos da exctamente uno menos.
- Si j es la posición en el SA del sufijo que se obtiene al borrar un caracter del sufijo de la posición i y j' del de borrar $i+1$, entonces $j+1 \leq j'$
- $LCP(j, j+1) \geq LCP(j, j') = \max(0, LCP(i, i+1) - 1)$
- Entonces podemos calcular el arreglo de LCP en el orden en el que los sufijos aparecen en el string, en cada paso restamos uno e iteramos de la forma obvia. Solo volvemos a empezar máximo una vez cuando vemos el último sufijo del SA.

Problema de ejemplo: Distinct Substrings

Distinct Substrings

Count the number of distinct substrings that appear in a string.
Constraints: $1 \leq n \leq 10^5$

`https://cses.fi/problemset/task/2105`

Problema de ejemplo: Distinct Substrings

Distinct Substrings

Count the number of distinct substrings that appear in a string.
Constraints: $1 \leq n \leq 10^5$

`https://cses.fi/problemset/task/2105`

Solución en el pizarrón.

Problema de ejemplo: Substring Order I

Substring Order I

You are given a string of length n. If all of its distinct substrings are ordered lexicographically, what is the kth smallest of them?
Constraints: $1 \leq n \leq 10^5$, $1 \leq k \leq \frac{n(n+1)}{2}$
It is guaranteed that k does not exceed the number of distinct substrings.

`https://cses.fi/problemset/task/2108`

Problema de ejemplo: Substring Order I

Substring Order I

You are given a string of length n. If all of its distinct substrings are ordered lexicographically, what is the kth smallest of them?
Constraints: $1 \leq n \leq 10^5$, $1 \leq k \leq \frac{n(n+1)}{2}$
It is guaranteed that k does not exceed the number of distinct substrings.

`https://cses.fi/problemset/task/2108`

Solución en el pizarrón.

Problema de ejemplo: Repeating Substring

Repeating Substring

A repeating substring is a substring that occurs in two (or more) locations in the string. Your task is to find the longest repeating substring in a given string.
Constraints: $1 \leq n \leq 10^5$

`https://cses.fi/problemset/task/2106`

Problema de ejemplo: Repeating Substring

Repeating Substring

A repeating substring is a substring that occurs in two (or more) locations in the string. Your task is to find the longest repeating substring in a given string.
Constraints: $1 \leq n \leq 10^5$

`https://cses.fi/problemset/task/2106`

Solución en el pizarrón.

Problema de ejemplo: Longest Common Substring II

Longest Common Substring II

A string is finite sequence of characters over a non-empty finite set Σ.
In this problem, Σ is the set of lowercase letters. Substring, also called factor, is a consecutive sequence of characters occurrences at least once in a string. Now your task is a bit harder, for some given strings, find the length of the longest common substring of them.
Here common substring means a substring of all the considered strings.
The input contains at most 10 lines, each line consists of no more than 100000 lowercase letters, representing a string.

`https://www.spoj.com/problems/LCS2/en/`
(ADVERTENCIA: Time Limit muy ajustado)

Problema de ejemplo: Longest Common Substring II

Longest Common Substring II

A string is finite sequence of characters over a non-empty finite set Σ.
In this problem, Σ is the set of lowercase letters. Substring, also called factor, is a consecutive sequence of characters occurrences at least once in a string. Now your task is a bit harder, for some given strings, find the length of the longest common substring of them.
Here common substring means a substring of all the considered strings.
The input contains at most 10 lines, each line consists of no more than 100000 lowercase letters, representing a string.

`https://www.spoj.com/problems/LCS2/en/`
(ADVERTENCIA: Time Limit muy ajustado)

Solución en el pizarrón.

Material de referencia

Implementaciones del vasito de algoritmos dados:

- Suffix Array: Link
- Longest Common Prefix: Link

Problemas de práctica: (Además de los explicados durante la clase)

- `https://cses.fi/problemset/task/2109`
- `https://cses.fi/problemset/task/2110`
- `https://codeforces.com/contest/1562/problem/E`

Pueden consultarme durante esta semana, o me pueden enviar un mail a:

- mateocvelez@mi.unc.edu.ar