ESTUDIANDO PREFIJOS

AUTOR: LAUTARO LASORSA

Solución Esperada

Para la Solución Esperada de este problema es necesario hacer 4 observaciones clave:

- Si armamos un **Trie** con las palabras, estando cada palabra vinculada a un nodo, el prefijo común mayor es la palabra vinculada al ancestro común menor (**LCA**) de dichos nodos. (el **Trie** es un árbol)
- Si uso el algoritmo explicado en la O.I.A. Wiki[1], si agrego un nodo al árbol puede calcular online su **tabla de ancestros.** Con esto, puedo mantener de forma dinámica un **Trie** en el que se puede calcular el **LCA** entre 2 nodos con complejidad **O(log(|Trie|).**
- Como la única función que puede crear nuevos nodos del **Trie** es **Agregar,** y puede crear a lo mucho un nodo, **|Trie| ≤ Q .**
- Como la tomar el **LCA** de 2 nodos es una **operación asociativa,** para tomar el **LCA** de un conjunto continuo de nodos puedo usar la estructura de datos **Segment Tree**[2][3], donde cada nodo contendrá el **LCA** del rango que representa. Notar que como el cálculo del **LCA** tiene complejidad **O(log(Q)),** cada consulta o actualización del **Segment Tree** tiene complejidad **O(log(Q) * log(N))**

Con estas observaciones, se puede representar al vector de palabras como un vector de enteros que indica a qué nodo del **Trie** está relacionada cada palabra.

A su vez, cada nodo del **Trie** contendrá la siguiente información:

- Sus hijos dentro del **Trie** (con un map o un array de enteros, por ejemplo)
- Su **tabla de ancestros** para el cálculo rápido del **LCA**
- Su profundidad, tomando la profundidad de la **raíz** como 0.

De esta forma, podemos implementar cada una de las funciones de la siguiente forma:

- **Inicializar**: Utilizaremos esta función para inicializar la **raíz** del **Trie** y lo que necesitemos del **Segment Tree.**

[1] http://wiki.oia.unsam.edu.ar/algoritmos-oia/grafos/arboles/lowest-common-ancestor
[2] http://wiki.oia.unsam.edu.ar/algoritmos-oia/estructuras/segment-tree
[3] Para ver los detalles de mi implementación de esta estructura, recomiendo ver esta explicación que hice para el curso O.I.A.-UNLaM https://docs.google.com/presentation/d/1lojZcLWkymB8_c2dTa6mKl012RaxFufKVe7aOqcSSzs/edit?usp=sharing

ESTUDIANDO PREFIJOS

AUTOR: LAUTARO LASORSA

- **Agregar:** Esta función es simplemente buscar cuál es el **hijo** en el **Trie** de la palabra en la posición **i** pasando por la arista vinculada al carácter **c**, creándolo si es necesario. Luego, igualamos la posición **i** a ese hijo y actualizamos el **Segment Tree**.
- **Borrar:** Esta función es buscar el **j-ésimo** ancestro del nodo asociado a la palabra **i**. Cosa que se puede hacer en **O(log(Q))** con la misma **tabla de ancestros** usada para el cálculo del **LCA**. Posteriormente, igualamos la posición **i** a ese valor y actualizamos el **Segment Tree**.
- **Cambiar:** Esta operación es simplemente intercambiar los valores del vector en las posiciones **i** y **j** y actualizar el **Segment Tree**.
- **Copiar:** Esta función es simplemente hacer que el valor en la posición **j** sea igual al valor en la posición **i** y actualizar el **Segment Tree**.
- **Preguntar:** Con la **query** propia del **Segment Tree** podemos devolver el **LCA** entre todas las palabras de ese rango, del cual debemos devolver su profundidad (que es su longitud como palabra)

Como todas las funciones (excepto Inicializar) implican **queries / updates** en el **Segment Tree**, todas tienen complejidad **O(log(Q)*log(N))**, con lo cual la complejidad total de esta solución resulta **O(N + Q*log(Q)*log(N))**.

Soluciones Parciales

Para la primera subtarea, se espera que se guarden explícitamente las palabras en un vector. Como no hay llamados a la función **Copiar**, la cantidad total de caracteres es a lo mucho **Q**.

Además, se espera que busque el prefijo común más largo iterando por todas las palabras y comparando una por una, con lo cual la complejidad total queda **O(Q*(Q+N))**.

(ya que como la suma de las longitudes es a lo mucho **Q**, la suma del costo de hacer las comparaciones contra todos los elementos del rango es el mínimo entre el **tamaño del rango** y **Q**)

En la segunda subtarea se espera también que guarde las palabras explícitamente, pero a su vez que arme el **Segment Tree** respecto de la operación **Prefijo Común Mayor**, guardando en cada nodo también de

ESTUDIANDO PREFIJOS

AUTOR: LAUTARO LASORSA

forma explícita este prefijo. Notar que la longitud máxima de una palabra está acotada por **Q**, por lo cual en este caso cada **query / update** del **Segment Tree** tiene una complejidad acotada por **O(Q*log(N))**. Al haber **Q** consultas, la complejidad total de esta solución resulta **O(Q^2*log(N))**.

En la tercera subtarea se espera que arme el **Trie** y guarde para cada palabra el nodo relacionado a esta, pero que no utilice el **Segment Tree** ni la forma rápida de calcular el **LCA**. Por esto último es que los llamados a la función **Borrar** solo borrarán una letra.

Se espera que, como en la primera subtarea, responda a los llamados de Preguntar recorriendo el vector de palabras, buscando en este caso el **LCA** entre los distintos nodos. Como la profundidad de cada nodo, es decir largo de la palabra que representa, es a lo mucho **Q**, la complejidad de está acción está acotada por **(N+Q)**. Como hay a lo mucho **100** llamadas a **Preguntar**, la complejidad total queda **O(100*(N+Q))**. (aunque en está notación no deberían incluirse constantes, esta es particularmente significativa para el tiempo real de ejecución)

En la cuarta subtarea se espera lo mismo que en la tercera, pero utilizando si la forma rápida de calcular **LCA**, lo que permite calcular rápidamente los llamados a **Borrar** cuando borran más de 1 letra, teniendo en este caso cada llamado una complejidad máxima de **O(log(Q))** . Por tanto, en este caso la complejidad es **O(100*(N+Q) + Q*log(Q))**.