Auxiliar 10

Universos Discretos - Diccionarios de Strings

Profesores: Benjamín Bustos, Gonzalo Navarro
Auxiliares: Sergio Rojas, Pablo Skewes

P1. [Resumen Materia]

P2. [P2 - C2 - 2023-2]

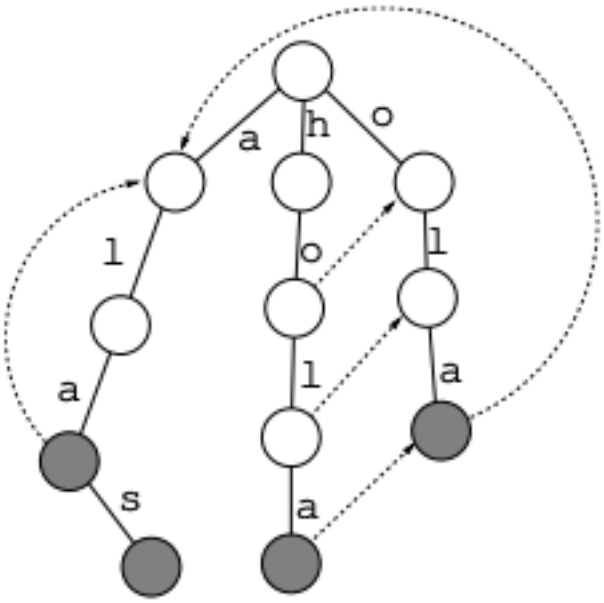
Se tiene una estructura de datos que almacena puntos en una grilla discreta de $t \times t$. Una vez construida, recibe rangos ortogonales $[x_1, x_2] \times [y_1, y_2]$, y responde en tiempo $O(\log t)$ si existe algún punto en ese rango, y de haberlos entrega uno de ellos.

Sabiendo de la existencia de esta estructura, encuentre una solución eficiente al siguiente problema: se tiene un conjunto S de strings y se quiere buscar por prefijo y sufijo, es decir, se entrega un prefijo $P[1..p]$ y un sufijo $S[1..s]$, y se desea saber si existe algún string en S que empiece con P y termine con S, entregando este string de existir.

Hint: Use una estructura que le permita agrupar en un rango todos los strings que comparten el mismo prefijo/sufijo. El tiempo de consulta debería ser $O(p + s + \log t)$.

P3. [P2 - C2 - 2022-2]

Considere un trie para un conjunto S de strings (la estructura básica, un nodo por letra). Para resolver un determinado problema, vamos a aumentar la estructura del trie con los así llamados **failure links**. Si un nodo interno v representa el string S, entonces su failure link, almacenado en v.flink, apunta al nodo $u \neq v$ que representa el máximo sufijo posible de S. La excepción es el failure link de la raíz, que apunta a la misma raíz. La siguiente figura muestra el trie aumentado para $S = \{ala, alas, hola, ola\}$, con el failure link de cada nodo en líneas punteadas (los que apuntan a la raíz se omiten). Por ejemplo, el failure link del nodo que representa «***hol***»" apunta al que representa su sufijo «***ol***». El failure link del nodo que representa «***ola***» apunta al que representa «***a***», pues no existe un nodo que represente «***la***». Note que no le agregamos el $ a los strings, porque lo que si una es prefijo de otra, terminará en un nodo interno. A cambio, pintamos de gris los nodos donde terminan strings, y decimos que son **terminales**.



Estos failure links sirven para búsqueda en texto. Supongamos que queremos encontrar todas las ocurrencias de los strings de S en un texto $T[1..n]$. Ejecutamos el siguiente algoritmo:

```
BÚSQUEDA EN TEXTO():
1  u ← raíz del trie
2  for i ← 1 to n:
3      if u es un nodo terminal then:
4          reportar i como la posición final de una ocurrencia
5      end
6      while u no es raíz y no tiene un hijo v por T[i] do u ← u.flink;
7      if u tiene un hijo v por T[i] then u ← v;
8  end
```

Para construir los failure links de un trie, los nodos se recorren en BFS desde la raíz. Para la raíz u, se pone simplemente u.flink $\leftarrow u$. Para cada otro nodo v en este recorrido, hijo de u por la letra a, se hace lo siguiente:

Se pide:

a) Demuestre que, una vez construido el trie, el tiempo de búsqueda de todos los patrones en $T[1..n]$ es $O(n)$.

b) **[Propuesto]** Demuestre que el tiempo total para crear los failure links en un trie de m nodos es $O(m)$.

Construcción de failure links():

1 $w \leftarrow u$.flink;
2 **while** w no es raíz y no tiene un hijo x por a **do** $w \leftarrow w$.flink;
3 **if** w tiene un hijo x por a **then** v.flink $\leftarrow x$;
4 **else** v.flink $\leftarrow$ raíz del trie;

Este algoritmo, de tiempo $O(m+n)$ para buscar un conjunto de strings de largo total m en un texto de largo n, fue presentado en 1975 por Alfred Aho y Margaret Corasick.