

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 $O(p + s + \log t)$.

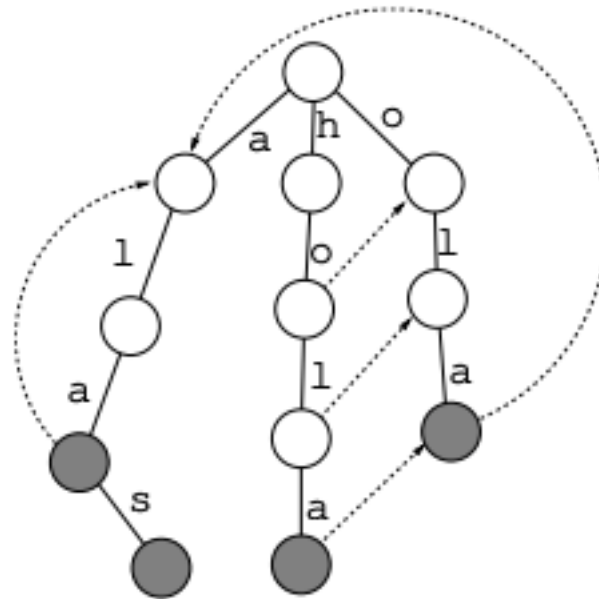
Solución:

Se crean dos tries, el T_P con los strings de S y el T_S con los strings de S reversos (leídos al revés). Cada nodo conoce el rango de hojas que abarca su subárbol (ej., de la 4ta a la 9na). En la grilla, cada string de S se guarda en el punto (x, y) si x es la posición de su hoja en T_P e y es la posición de su hoja en T_S .

Dada la query (P, S) , busco P en T_P y el reverso de S en T_S , con búsqueda de prefijos. Si en alguno no está, entonces no hay respuestas. Si las hay, tomo el rango $[x_1, x_2]$ de las hojas bajo el nodo al que llegué en T_P y el rango $[y_1, y_2]$ de las hojas bajo el nodo al que llegué en T_S . Si la grilla me dice que no hay puntos en ese rango, no hay respuestas. Si no, la grilla me dará un string de S que empieza con P y termina en S . El tiempo es el predicho.

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 $v.\text{flink}$, 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   $u \leftarrow$  raíz del trie
2  for  $i \leftarrow 1$  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  $u \leftarrow u.\text{flink}$ ;
7      if  $u$  tiene un hijo  $v$  por  $T[i]$  then  $u \leftarrow v$ ;
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 $u.\text{flink} \leftarrow u$. Para cada otro nodo v en este recorrido, hijo de u por la letra a , se hace lo siguiente:

Se pide:

- Demuestre que, una vez construido el trie, el tiempo de búsqueda de todos los patrones en $T[1..n]$ es $O(n)$.
- [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   $w \leftarrow u.\text{flink};$   
2  while  $w$  no es raíz y no tiene un hijo  $x$  por  $a$  do  $w \leftarrow w.\text{flink};$   
3  if  $w$  tiene un hijo  $x$  por  $a$  then  $v.\text{flink} \leftarrow x;$   
4  else  $v.\text{flink} \leftarrow$  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.

Solución:

a) Podemos tomar como función potencial la profundidad de u , que parte en cero. En cada paso, hay $O(1)$ operaciones (líneas 3–5 y 7) que diremos que cuestan $c_i = 1$, más el que el potencial aumenta en 1 en la línea 7, tenemos $\hat{c}_i = 2$. La línea 6 se ejecuta varias veces, a costo 1 por iteración, pero disminuye la profundidad de u en cada paso, por lo que su costo amortizado es cero. En total, $O(n)$. Otra idea que pueden usar es decir que hay dos cursores, uno donde comienza el posible alineamiento del match (que realmente es menos la profundidad de u) y otro donde está i . En cada paso de costo $O(1)$, o bien i crece, o bien el puntero donde empieza el alineamiento crece.

b) Podemos tomar una función potencial como

$$\varphi = \sum_{\{v \text{ con padre } u\}} d(v.\text{flink}) - d(u.\text{flink})$$

donde $d(x)$ es la profundidad del nodo x , y la suma se hace sobre los nodos v que ya tienen definido su failure link. Al comenzar, $\varphi = 0$. Al procesar un nodo v con padre u , el código primero intenta que $v.\text{flink}$ sea un hijo de $u.\text{flink}$, con lo cual φ crecería en 1 y trabajaríamos $O(1)$. Pero si no es así, iteramos en la línea 2. En cada iteración, el futuro $d(v.\text{flink})$ va decreciendo en 1. Si iteramos k veces, entonces $d(v.\text{flink}) - d(u.\text{flink}) \leq -k + 1$. Al final $d(v.\text{flink})$ puede decrecer uno más si llegamos a la raíz. En total, si iteramos k veces en la línea 2, será $c_i = k$ y $d(v.\text{flink}) - d(u.\text{flink}) \leq -k + 2$, con lo que $\hat{c}_i \leq 2$.