Vivian Marino Sánchez
Ingeniería de Sistemas Computacionales

Algorítmica II

Investigación #1

Primer Parcial

E-mail: marino6733784@gmail.com - Telf. 72584173

Investigación #1: Suffix Trie

1 Trie

Un Trie es un árbol compuesto por una colección de Strings. Los Strings se dividen en los caracteres que los componen y cada caracter es un nodo. El árbol se estructura acomodando los caracteres de cada String en orden. Cada nodo solamente puede bifurcarse en un nodo por caracter, esto quiere decir que si dos Strings inician con 'a', compatirán el nodo inicial, y si dos Strings inician con 'pa', compartirán los dos primeros nodos. Los finales de palabra son marcados como 'hojas'.

Se representa con arrays o maps.

2 Suffix Trie

2.1 Definición

Habiendo dado una breve explicación de Trie, la definición de un Suffix Trie no es difícil de entender. En este caso tomamos un String -o un conjunto de Strings- e incluimos en el Trie todos los posibles sufijos que podemos obtener. Esto quiere decir que tomaremos el String entero, lo posicionaremos, y luego tomaremos el String desde la posición i hasta el caracter final en un bucle hasta que lleguemos al vacío. En este caso el vacío será representado por un caracter extra que añadiremos al String original. Esto siempre nos dará un árbol con m hojas, donde m es el número

de caracteres que tiene el String más el caracter especial añadido.

2.2 Construcción

Para construir un Suffix Trie se utiliza el algoritmo de Ukkonen. Este algoritmo indica que construiremos un Suffix Trie para cada prefijo del String. Para realizar cada sub Suffix Trie existen tres reglas:

1. Si ya existe el principio del sufijo a introducir, revisamos las letras consecutivas y las comparamos con las del sufijo. Si faltan letras para formar el sufijo, simplemente se añaden los caracteres correspondientes.
2. Si no existe el principio del sufijo, se crea un nuevo nodo.
3. Si el sufijo ya existe como prefijo de alguna de las ramas del trie, no se hace nada.

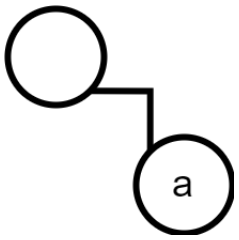
Veamos un ejemplo con el String $A = abbcba$:

Primero, como dijimos en la definición, añadiremos un caracter especial. Entonces, tenemos: $A = abbcba\$$.

A continuación utilizaremos el algoritmo de Ukkonen:

1. Tomamos el prefijo a e introduciremos todos sus sufijos: $S = [a]$.

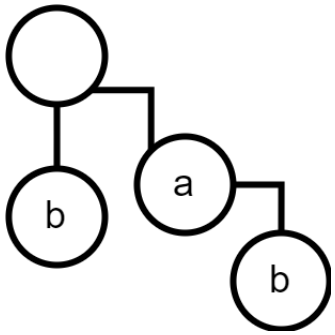
   Aplicamos la segunda regla con a.



2. Tomamos el prefijo ab e introduciremos todos sus sufijos: $S = [ab, b]$.

   Aplicamos la primera regla con ab.

   Aplicamos la segunda regla con b.

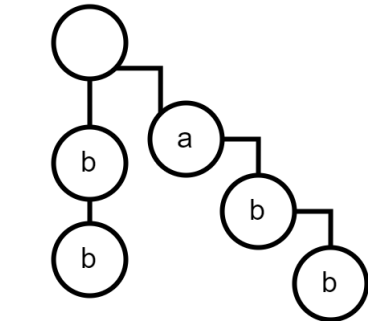


3. Tomamos el prefijo abb e introduciremos todos sus sufijos:

   $S = [abb, bb, b]$.

   Aplicamos la primera regla con abb y bb.

   Aplicamos la tercera regla con b.

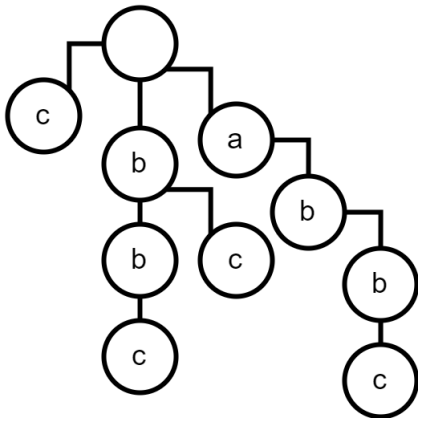


4. Tomamos el prefijo $abbc$ e introduciremos todos sus sufijos:

   $S = [abbc, bbc, bc, c]$.

   Aplicamos la primera regla con $abbc$, bbc y bc.

   Aplicamos la segunda regla con c.

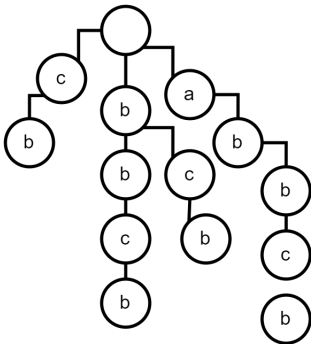


5. Tomamos el prefijo $abbcb$ e introduciremos todos sus sufijos:

   $S = [abbcb, bbcb, bcb, cb, b]$.

   Aplicamos la primera regla con $abbcb$, $bbcb$, bcb y cb.

   Aplicamos la tercera regla con b.

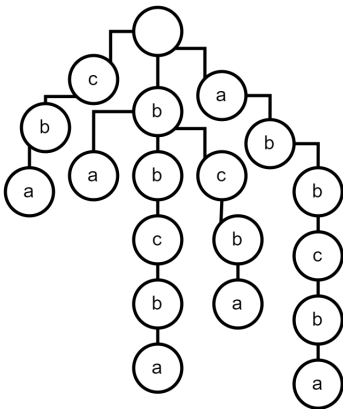


6. Tomamos el prefijo $abbcba$ e introduciremos todos sus sufijos:

   $S = [abbcba, bbcba, bcba, cba, ba, a]$.

   Aplicamos la primera regla con $abbcba$, $bbcba$, $bcba$, cba y ba.

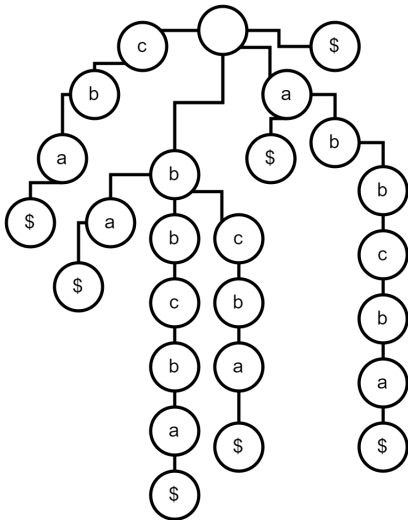
   Aplicamos la tercera regla con a.



7. Tomamos el prefijo $abbcba\$$ e introduciremos todos sus sufijos:

   $S = [abbcba\$, bbcba\$, bcba\$, cba\$, ba\$, a\$, \$]$.

   Aplicamos la primera regla con todos los sufijos, excepto con $\$$ con el cual aplicamos la segunda regla.



2.3 Implementación

Si implementamos el algoritmo de Ukkonen como lo explicamos, nos encontramos con el problema de que puede llegar a $O(n^3)$. Esto quiere decir que el algoritmo es pesado y no es eficiente. Entonces, para hacer un programa que construya un Suffix Trie, hay que realizar ciertas modificaciones.

Todavía trabajaremos con los sufijos de los prefijos del String a insertar. También le añadiremos un caracter especial al final del String original. Sin embargo, esta vez las incersiones se realizarán de manera más directa,

recordando los sufijos implícitos (que se encuentran como prefijos de ramas) y uniendo bifurcaciones de distintas ramas.

Cada vez que cambiemos un prefijo, diremos que hemos cambiado de fase y cada vez que añadamos el sufijo más básico de un prefijo o un acarreo diremos que hemos cambiado de extensión. Las bifurcaciones serán renombradas a nodos internos. Además, no todos los caracteres tendrán un nodo correspondiente, más bien, la mayoría de los nodos tendrá una línea de texto. También tendremos un artefacto 'activo' compuesto por el nodo del que partiremos -nodo activo-, la posición desde el nodo después de la que haremos una incersión -distancia activa- y el caracter del nodo en la posición equivalente a la distancia activa -caracter activo-. Por último, será necesario el uso de un contador de sufijos a añadir.

Las reglas a aplicar serán:

1. Cada cambio de fase sumamos uno al contador de sufijos a añadir. Cada cambio de extensión restamos uno al contador de sufijos a añadir.

2. Cada fase añadimos el nuevo caracter tomado al final de todos los caminos existentes.

3. Cada cambio de fase añadimos el caracter del String original en la posición equivalente a la fase en que nos encontramos a los sufijos acarreados.

4. Cada que creemos un nodo interno en una fase, el nodo interno creado

anteriormente apuntará a este nuevo nodo, a menos que el nodo interno anterior sea nulo.

5. Si el sufijo a insertar no existe después del caracter activo en la distancia activa dentro del nodo activo, lo añadiremos y hacemos un cambio de extensión. Aquí, si nos encontramos en el nodo inicial -root-, el nuevo caracter activo será el primer caracter del siguiente sufijo a ingresar y la distancia activa reduce en uno.

6. Si ya existe un caracter distinto en el lugar en que debería ir este nuevo sufijo, crearemos un nodo interno que marque la bifurcación.

7. Si una parte del sufijo a insertar existe después del caracter activo en la distancia activa del nodo activo, sumamos uno a la distancia activa por cada caracter ya presente. Si de pronto la distancia excede en número de caracteres dentro del nodo, saltamos al nodo interno que sigue.

8. Si el sufijo a insertar ya existe por completo en una rama, pasamos a la siguiente fase sin restar el contador de sufijos a añadir.

Veamos un ejemplo utilizando el mismo String que vimos en el apartado anterior: $A = abbcba$

Primero, como dijimos en la definición, añadiremos un caracter especial. Entonces, tenemos: $A = abbcba\$$.

Por esta ocasión, la fase será equivalente a $i + 1$, donde i es la posición partiendo de 0 que marca el final del prefijo que estamos usando.

1. Tomamos a.

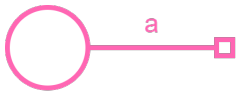
   Debemos añadir $[a]$.

   Nuestro activo es: $(root, , 0)$.

   Nuestro contador de sufijos es 1.

   Entonces, desde root, nuestro nodo inicial, contamos 0 y añadimos a como un nuevo nodo.

   Restamos uno al contador de sufijos.



2. Tomamos ab.

   Debemos añadir $[ab, b]$.

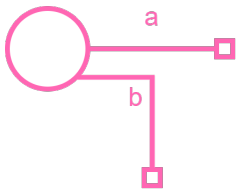
   Nuestro activo es: $(root, , 0)$.

   Nuestro contador de sufijos es 1.

   Entonces, desde root, contamos 0 y añadimos b en un nuevo nodo.

   Añadimos b al nodo que ya teníamos con a.

   Restamos uno al contador de sufijos.



3. Tomamos abb.

   Debemos añadir $[abb, bb, b]$.

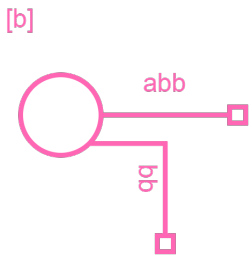
   Nuestro activo es: $(root, , 0)$.

   Nuestro contador de sufijos es 1.

   Añadimos b al nodo que ya teníamos con ab.

   Añadimos b al nodo que ya teníamos con b.

   Como ya tenemos un nodo que inicia con b, nuestro activo pasa a: $(root, b, 1)$.



4. Tomamos $abbc$.

   Debemos añadir $[abbc, bbc, bc, c]$.

   Nuestro activo es: $(root, b, 1)$.

   Nuestro contador de sufijos es 2.

   Añadimos c al nodo que ya teníamos con abb.

   Añadimos c al nodo que ya teníamos con bb.

   Desde el nodo activo, contamos la distancia activa e insertamos el caracter que le sigue: c.

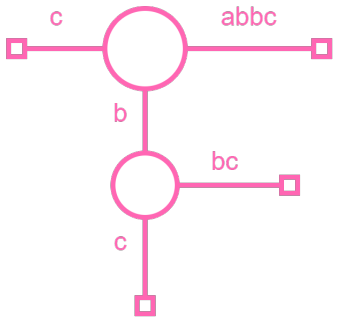
   Como ya insertamos el nodo b que acarreamos de la anterior fase

en la forma de bc, nuestro activo pasa a: $(root, c, 0)$. También creamos un nodo interno para denotar la bifurcación.

Restamos uno al contador de sufijos.

Desde root, contamos 0 y añadimos c en un nuevo nodo que inicie con c.

Restamos uno al contador de sufijos.



5. Tomamos $abbcb$.

   Debemos añadir $[abbcb, bbcb, bcb, cb, b]$.

   Nuestro activo es: $(root, c, 0)$.

   Nuestro contador de sufijos es 1.

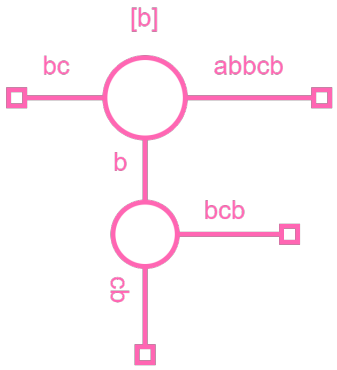
   Añadimos b al nodo que ya teníamos con $abbc$.

   Añadimos b al nodo que ya teníamos con bbc.

   Añadimos b al nodo que ya teníamos con bc.

   Añadimos b al nodo que ya teníamos con c.

   Como ya tenemos un nodo que inicia con b, nuestro activo pasa a: $(root, b, 1)$.



6. Tomamos $abbcba$.

   Debemos añadir $[abbcba, bbcba, bcba, cba, ba, a]$.

   Nuestro activo es: $(root, b, 1)$.

   Nuestro contador de sufijos es 2.

   Añadimos a al nodo que ya teníamos con $abbcb$.

   Añadimos a al nodo que ya teníamos con $bbcb$.

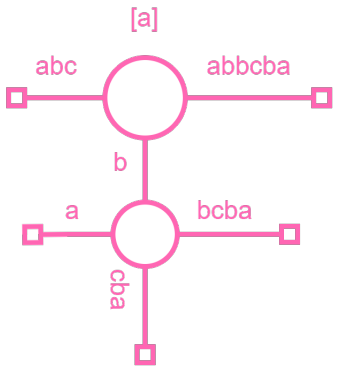
   Añadimos a al nodo que ya teníamos con bcb.

   Añadimos a al nodo que ya teníamos con cb.

   Desde el nodo activo, contamos la distancia activa e insertamos el caracter que le sigue: a. Como ya insertamos el nodo b que acarreamos de la anterior fase en la forma de ba, nuestro activo pasa a: $(root, a, 0)$.

   Restamos uno al contador de sufijos.

   Como ya tenemos un nodo que inicia con a, nuestro activo pasa a: $(root, a, 1)$.



7. Tomamos $abbcba\$$.

   Debemos añadir $[abbcba\$, bbcba\$, bcba\$, cba\$, ba\$, a\$, \$]$.

   Nuestro activo es: $(root, a, 1)$.

   Nuestro contador de sufijos es 2.

   Añadimos $\$$ al nodo que ya teníamos con $abbcba$.

   Añadimos $\$$ al nodo que ya teníamos con $bbcba$.

   Añadimos $\$$ al nodo que ya teníamos con $bcba$.

   Añadimos $\$$ al nodo que ya teníamos con cba.

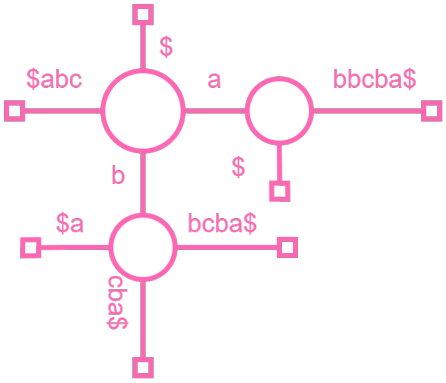
   Desde el nodo activo, contamos la distancia activa e insertamos el caracter que le sigue: $\$$. Como ya insertamos el nodo a que acarreamos de la anterior fase en la forma de $a\$$, nuestro activo pasa a: $(root, \$, 0)$. También creamos un nodo interno para denotar la bifurcación.

   Restamos uno al contador de sufijos.

   Desde root, contamos 0 y añadimos $\$$ en un nuevo nodo que inicie

con $.

Restamos uno al contador de sufijos.



Y concluimos. Podemos verificar que todo fue hecho de manera correcta porque el contador de sufijos es 0.

Bibliografía

[1] Enjoy Computer Science Staff. (s. f.).*Suffix Tree Data Structure*. Code Algorithms Pvt. Ltd. : Enjoy Computer Science. https://www.enjoyalgorithms.com/blog/suffix-tree-data-structure

[2] Geeks for Geeks Staff. (2021).*Ukkonen's Suffix Tree Construction – Part 1*. Geeks for Geeks. https://www.geeksforgeeks.org/ukkonens-suffix-tree-construction-part-1/

[3] Langmead, B. (s.f.). *Tries and suffix tries*. Johns Hopskins Whiting School of Engineering. https://www.cs.jhu.edu/~langmea/resources/lecture_notes/tries_and_suffix_tries.pdf

[4] StackOverflow User. (2019). *Ukkonen's suffix tree algorithm in plain English*. Stack Exchange, Inc.: StackOverflow. https://stackoverflow.com/questions/9452701/ukkonens-suffix-tree-algorithm-in-plain-english

[5] Tushar Roy - Coding Made Simple. (2015). *Suffix Tree using Ukkonen's algorithm*. YouTube. https://www.youtube.com/watch?v=aPRqocoBsFQ