

Tarea 2

Integrantes: Esteban Ramirez
Valentina Esteban
Profesor: Gonzalo Navarro
Auxiliar: Antonia Labarca Sánchez
Ayudantes: Claudio Zamorano
Javier Oliva
Matías Rojas
Nicolás Canales

Fecha de realización: 13 de diciembre de 2021
Fecha de entrega: 13 de diciembre de 2021
Santiago de Chile

Índice de Contenidos

Implementación	1
1. Arreglo de Sufijos	1
Creación arreglo	1
Búsqueda	2
2. Árbol de sufijos Patricia	2
Búsqueda	3
Inserción	4
Resultados	5
1. Arreglo de Sufijos	5
2. Árbol de sufijos Patricia	7
Interfaz	10
Uso	10

Índice de Figuras

1. ejemplo arbol sufijos para el texto "ANANA_AN\$"	3
2. Buscando palabra 'and'.	6
3. Buscando palabra 'the'.	6
4. Buscando palabra 'the'.	7
5. Ingreesando palabra 'th'.	8
6. Ingreesando palabra 'the ba'.	8
7. Ingreesando palabra 'the ba'.	9
8. Index interfaz.	10
9. Se agrega frutas.txt	11
10. Formulario de búsqueda.	11
11. Buscando 'A' en el arreglo de sufijos.	12
12. Buscando 'AN' en el arreglo de sufijos.	12
13. Buscando 'A' en el árbol de sufijos.	13
14. Buscando 'AN' en el árbol de sufijos.	13

Índice de Tablas

1.	1
2.	1

Implementación

1. Arreglo de Sufijos

Un arreglo de sufijos corresponde a una estructura que contiene todos los sufijos de un texto de forma ordenada usando la referencia de estos en el texto. Es una forma de reducir el espacio que ocuparía un árbol de sufijos pero a cambio sacrificando información. Toma tiempo lineal tanto para su construcción como en el espacio que ocupan. No nos permiten resolver problemas muy complejos.

Por ejemplo sea un texto $T = \text{"ANANA_AN\$"}$, en las tablas podemos ver la indexación de texto y luego los índices que representa a todos los posibles sufijos del texto.

Tabla 1

index	0	1	2	3	4	5	6	7	8
$T[i]$	A	N	A	N	A	_	N	A	\$

Tabla 2

Sufijo	index
ANANA_AN\$	0
NANA_AN\$	1
ANA_AN\$	2
NA_AN\$	3
A_AN\$	4
_AN\$	5
AN\$	6
N\$	7
\$	8

Para la implementación del arreglo de sufijos se crea una clase *SuffixArray* la cual contiene las variables *file* , *n* y *arr* correspondientes a un texto, el largo del texto y un arreglo con valores que van del 0 al largo del texto respectivamente. Por otro lado la clase comprende los métodos: *getArr* para obtener el arreglo de sufijos de la clase, *searchSuffix(s)* la cual busca las ocurrencias de un string en el texto , *occurrenceLine(index)*.que retorna una línea del texto a partir del índice entregado y *estebanSort(first,last)* la cual es la encargada de ordenar la referencias de los sufijos del texto lexicográficamente.

Creación arreglo

En cuanto a la creación de este arreglo de sufijos se debe crear un objeto de la clase dándole el parámetro *file* correspondiente a la ubicación del texto. Con esto el constructor asigna la variable *file* y *n*. Luego para la variable *arr* crea un arreglo con índices que van del 0 al largo del texto menos1, para luego llamar al método *estebanSort* el cual se encarga de ordenar correctamente las referencias

de los sufijos del texto lexicográficamente.

El método *estebanSort(first, last)* está basado en el algoritmo Quicksort. El cual elige un pivote en la posición media del arreglo y con un puntero en la parte inicial del arreglo y otro en la parte final se va comparando los sufijos que corresponden a cada referencia hasta encontrar una posición tal que el puntero del lado izquierdo queda apuntando a una referencia cuyo sufijo es mayor que el pivote y el puntero final a uno menor al pivote, donde finalmente se intercambian de posición. Esto se repite mientras el puntero que comienza al principio del arreglo no sobrepase al que empieza al final. Finalmente se vuelve a llamar a la función recursivamente cambiando los valores de comienzo y termino del arreglo.

Búsqueda

El método comienza con un contador y arreglo vacío los cuales se van actualizando a medida que se revisa el arreglo de sufijos. Para ello se recorre el arreglo de sufijos de la clase comparando si el string a buscar comienza igual que el sufijo al cual se está referenciando en el arreglo. De ser así el contador aumenta y se agrega al arreglo del método la línea en donde aparece el string. Para esto se llama al método *occurrenceLine(index)* al cual le pasamos el índice del sufijo que se está comparando y retorna hasta encontrar un final de línea. Esto lo hace hasta que termina de recorrer el arreglo de sufijos.

2. Árbol de sufijos Patricia

Un árbol de sufijos patricia es una estructura de datos útil para encontrar eficientemente las ocurrencias de un patrón, en tiempo proporcional al largo del patrón.

Para en este trabajo se implementa el árbol en cuestión, en Python, usando una clase *PatriciaTree* y la clase *Node*. La primera recibe el archivo para el cual debe crear un árbol de sufijos, si bien se podría haber guardado este archivo en una variable global (para no cargarlo a una instancia de *PatriciaTree*), se decidió guardarlo así ya que la estructura modular del código lo hace más cómodo.

Con el archivo como variable interna se realizan operaciones de inserción, búsqueda y además un método *printTree* y *preprocessing* (el cual agrega la cantidad de descendientes a los nodos de un árbol ya construido).

Para entender los siguientes métodos, se debe considerar que una instancia de *Node* tiene una lista de hijos, un char que indica el char leído para llegar a él, el largo del string que empieza en el char contenido y termina en algún char del hijo, un índice que es seteado en caso de ser hoja y una variable que indica si es hoja o nodo interno, y una referencia a su padre.

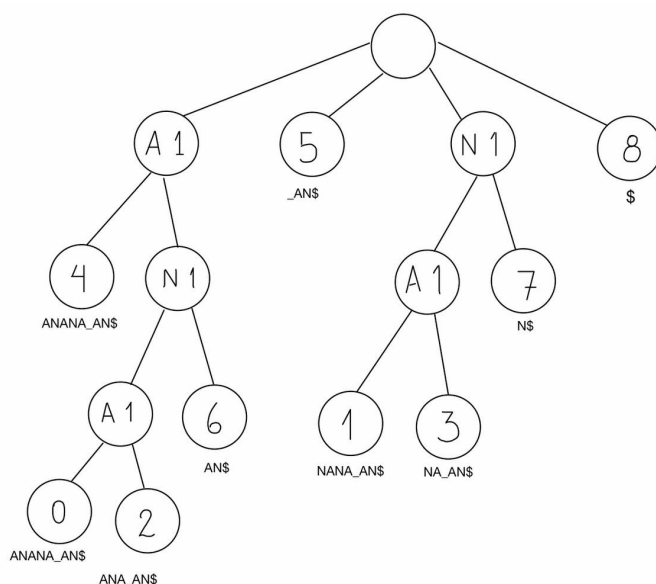


Figura 1: ejemplo árbol sufijos para el texto "ANANA_AN\$"

Búsqueda

Los métodos de búsqueda en el árbol están creados de forma tal que retorna una lista de a lo más tres elementos, estos elementos son los *strings* que debo sugerir como autocompletado.

Para realizar la búsqueda primero se identifica el caso base, que en la clase del árbol es que la raíz sea un *Node* sin hijos, es decir, un árbol vacío, en cuyo caso retorna una lista vacía. Si no está vacía, entonces delega la tarea al método *search(root)* de la clase *Node* que sabe como tratar la búsqueda en nodos. Éste último método primero verifica que lo ingresado no sea una hoja, si llega a ser una hoja verifica si lo que sea ha tipeado es substring de sufijo que contiene dicha hoja, de ser así, entonces retorna un arreglo con un elemento: el sufijo completo que contiene la hoja. Si a *search* ingresa un nodo que no es hoja, entonces va a identificar si el largo del string tipeado es mayor que lo que debo pagar ($|camino| \leq len(string)$) para avanzar a un hijo del nodo actual, si lo es significa que me falta seguir buscando en el árbol, por lo que va a revisar todos los hijos del nodo actual y para decidir por qué camino debe ir, una vez haya un match, sigue la recursión en ese nodo actualizando los parámetros de camino recorrido. Si no hay match, retorna una lista vacía.

Por otro lado, si no puede seguir avanzando en el árbol ($|camino| > len(string)$), significa que está a medio camino o que justo el string tipeado terminó en un nodo, en ambos casos lo que debe hacer es revisar todos los hijos del nodo actual, ordenar por cantidad de descendientes y capturar los caracteres que le faltan al string tipeado para llegar a cada uno de los nodos. Comprueba que lo tipeado efectivamente sea substring de lo que debería haber leído hasta el nodo actual, de ser así retorna una lista (de largo máx 3) ordenada por descendientes de el string tipeado + lo que le falta para llegar a cada hijo. Pero si no era substring, significa que no hay match en el árbol, por lo que retorna una lista vacía.

Inserción

La inserción recibe el índice del sufijo del texto que se quiere ingresar, captura el string a ingresar.

Primero identifica el caso base: si el nodo es una hoja, de ser así, identifica 2 casos en función de donde difieren el string tipeado con el sufijo de la hoja. Si difieren en una posición mayor o igual al meno que he recorrido para llegar al nodo actual, significa que se debe generar una especie de bifurcación que va a comenzar justo en la posición en que se identifica que difieren. Un hijo va a ser el del sufijo al que llegué originalmente y el otro hijo va a ser una hoja que complete el sufijo que se está ingresando al árbol. Por otro lado, si difieren en una posición menor al camino recorrido, tenemos el caso 3 mencionado en el apunte, en el cual se debe retroceder hasta encontrar el camino de v_r a v_s donde ocurrió la diferencia, y se repite el proceso anterior.

Si el nodo que llama a la inserción no es una hoja, entonces identifica si se ha pasado o no en la posición donde debería ingresar el sufijo en cuestión comparando la cantidad de chars recorridos (camino recorrido) con el largo del sufijo a ingresar.

Si no se ha pasado busca hijos para seguir avanzando, si lo encuentra, llama a *insert* recursivamente actualizando el valor de la variable que indica el índice del char en el sufijo ingresado (camino recorrido). Si no encuentra un match en los hijos, entonces debe ingresar aquí el sufijo, para lo cual tiene 2 casos, si el sufijo actual difiere con todos los hijos del nodo actual en la misma posición entonces agrego el sufijo como otro hijo, pero si no, entonces debo hacer lo que indica en el caso 2 del apunte. Si se ha pasado, entonces debe de la misma forma que en último caso mencionado (caso 2 del apunte).

Resultados

Se ingresan dos textos para probar el comportamiento de del arreglo y árbol de sufijos:

El primero correspondiente a los dos primeros párrafos del texto alice29.txt. No se probó completo ya que la interfaz gráfica implementada para mostrar las líneas donde una palabra aparece en el texto no soportó tantas apariciones.

Mientras que el segundo fue un extracto del texto aaa.txt. Al igual que el caso anterior se decidió no correr completo debido a que no se logró soportar la construcción de las estructuras con un largo tan grande.

A continuación se mostraran 3 imágenes para el arreglo y árbol de sufijos donde las dos primera corresponden a búsquedas en el texto alice29.txt y la tercera en el texto aaa.txt.

1. Arreglo de Sufijos

Buscando la palabra “the” en el arreglo de sufijos podemos observar en la figura 2 que nos da 8 apariciones en sufijos. Dado que el string debe ser igual al comienzo del sufijo para ser contado como aparición en el texto no tenemos repeticiones.

Por otro lado en la figura 3 tenemos la realización de la búsqueda de la palabra “the bank” la cual tiene un solo resultado. Esto se da a causa de que a medida que un palabra o texto va creciendo es menos probable que se encuentra en el texto repetida.

En la tercera imagen podemos ver como un string de solamente a’s es buscado en el arreglo de sufijo y dando como resultados varias líneas que corresponden a sus apariciones en el texto. Esto se da ya que al tener un texto compuesto completamente de a’s la cantidad de apariciones en el texto corresponderá al largo del texto menos el largo del string, haciendo entonces que se tenga que recorrer todo el texto linealmente y por lo tanto tomando un tiempo mayor al momento de la búsqueda.

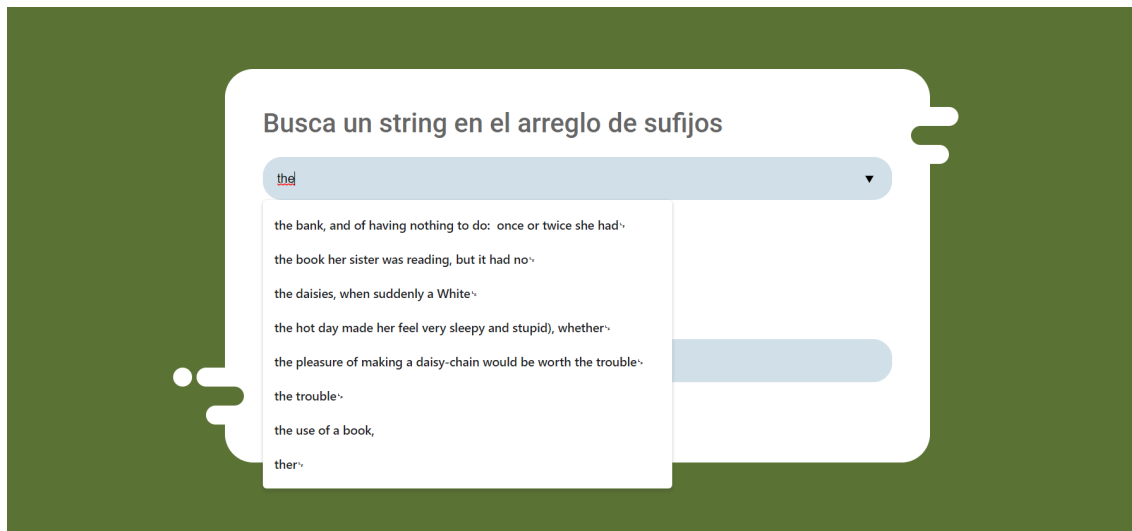


Figura 2: Buscando palabra 'and'.

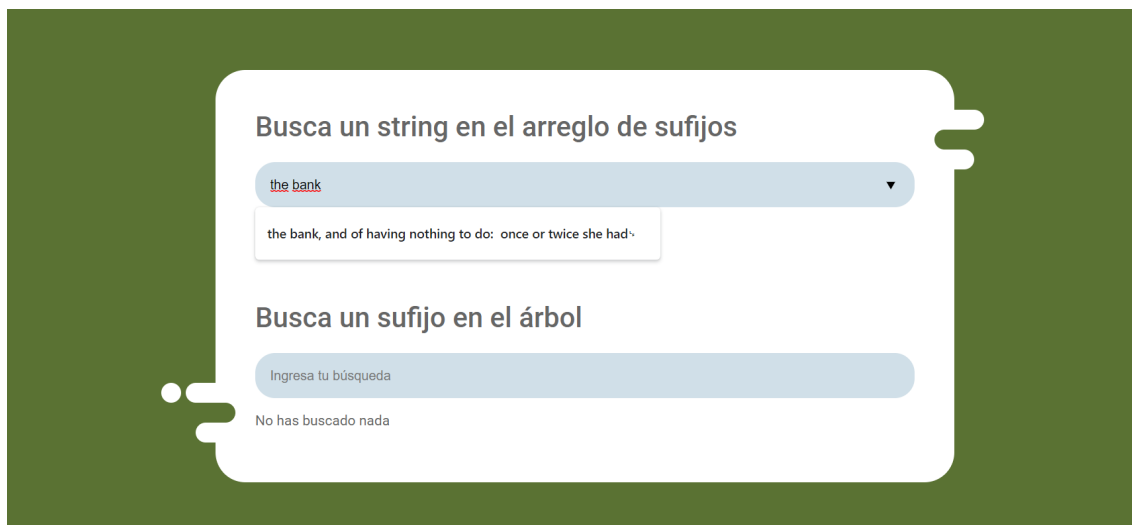


Figura 3: Buscando palabra 'the'.

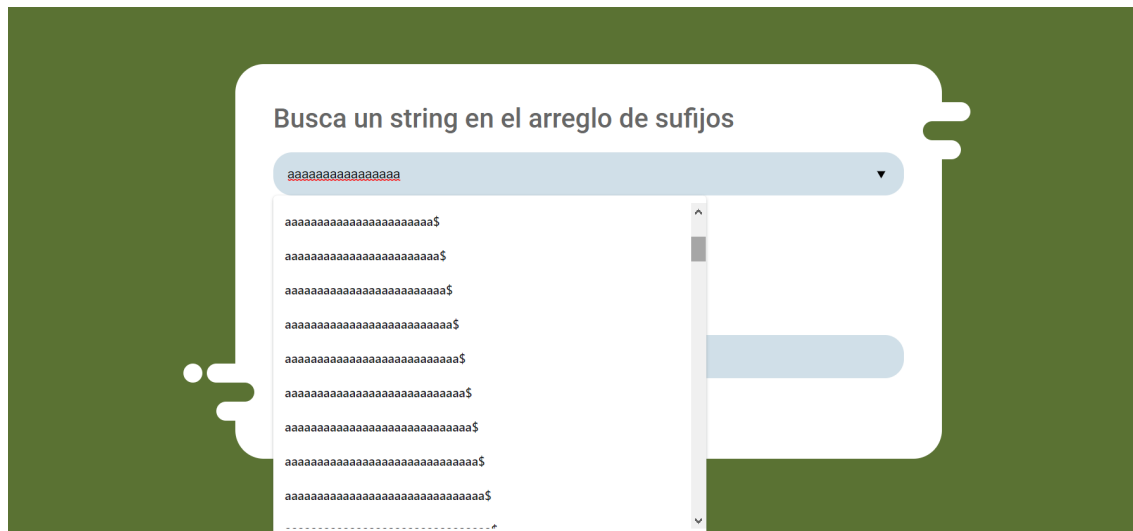


Figura 4: Buscando palabra 'the'.

2. Árbol de sufijos Patricia

Para el primer caso tenemos el ingreso de la palabra “th” que como se ve en la figura 5 nos da 5 posibles autocompletados. Es decir en nuestro árbol de sufijos tipo patricia tenemos un nodo cuyo character guardado corresponde a “t” y que contiene 3 hijos con characters “e”, “o” y un espacio.

Luego si seguimos completando dicha palabra hasta completar la palabra “the ba” podemos ver que solo se tiene una sugerencia de autocompletado. Esto debido a que del nodo que esta solo hay un hijo y como es un árbol de sufijos se tiene una sugerencia desde ese punto hasta el final del texto.

Finalmente en la ultima imagen tenemos el ingreso de la palabra “aaa” con solo dos sugerencias de autocomplpetado correspondientes a una “a” más o el termino del texto. Esto se da producto de que al tener un texto de solamente a’s, el árbol de sufijos construido será parecido a una lista enlazada pero en donde cada nodo tendrá dos hijos uno con que almacena el character de final de texto y otro con una “a”. Cada sufijo del texto difiere del anterior solamente en la cantidad de a’s que tien.

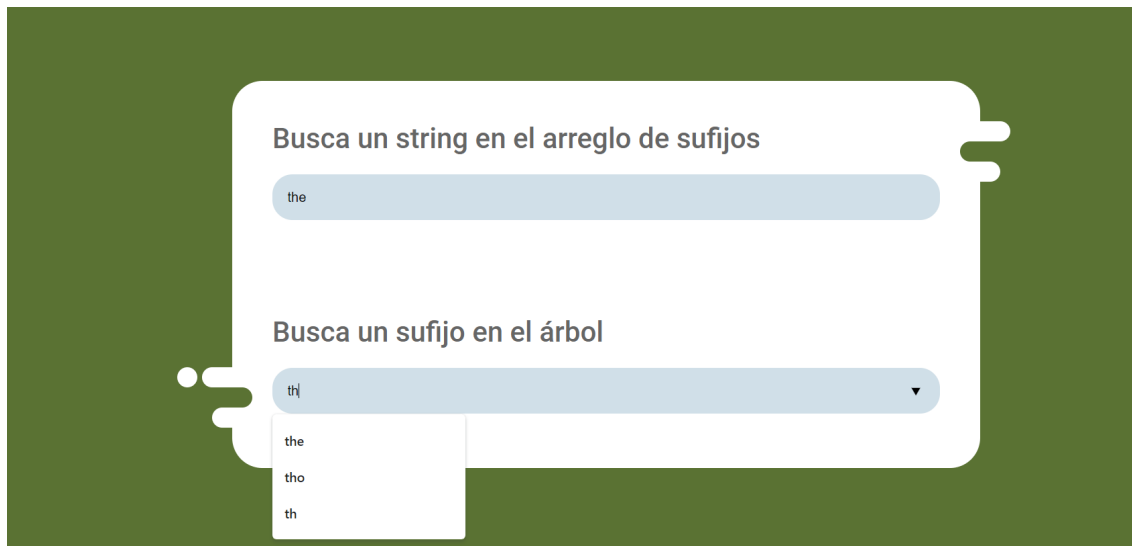


Figura 5: Ingresando palabra 'th'.

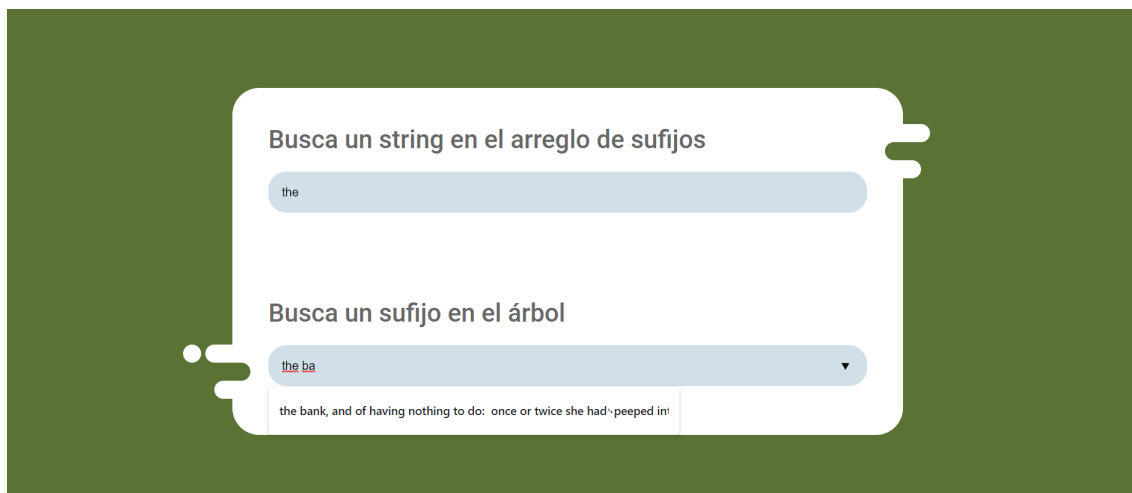


Figura 6: Ingresando palabra 'the ba'.

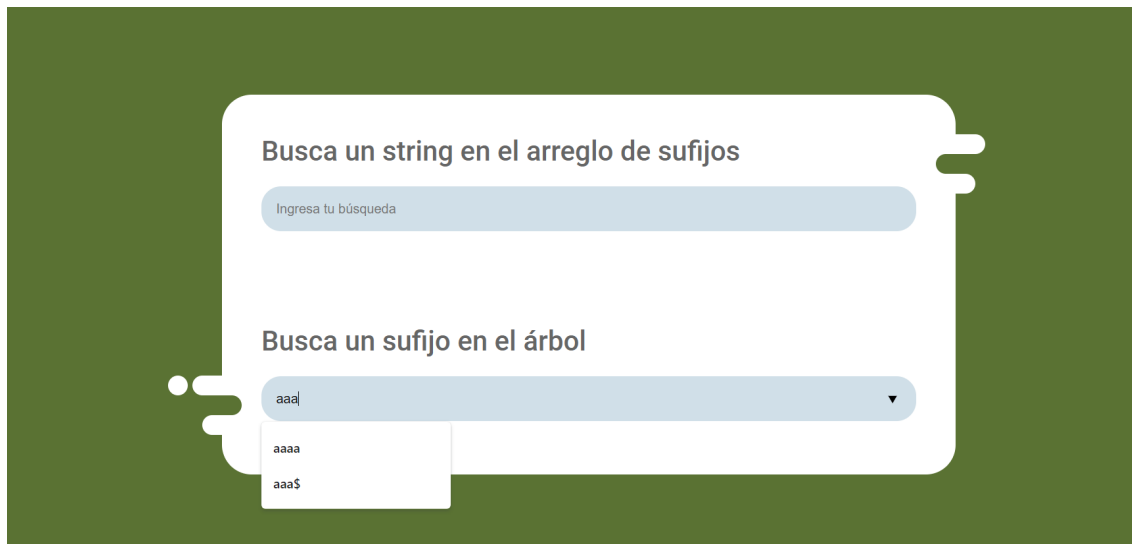


Figura 7: Ingresando palabra 'the ba'.

Link de github: <https://github.com/DidoTau/Tarea2-CC4102>

Nota: En la carpeta app/cgi-bin/ se podrá encontrar los archivos `arbol_patrici.py`, `arreglo_sufijos.py`, `correspondiente`

Interfaz

Para los puntos 2 y 4 de la tarea se creó una aplicación web que cumple con lo pedido.

La aplicación está en la carpeta *app* del repositorio y se hizo con modulo CGI de python por simplicidad.

La tecnología que permite sugerir al usuario en tiempo de tipeo es AJAX.

Uso

Para correr la aplicación, desde la terminal, estando ubicado en el directorio *app* debe correr el comando:

```
python -m http.server --bind localhost --cgi puerto
```

En **puerto** debe ingresar el número del puerto donde desea hacer el deployment de la aplicación. Luego, ingresando en [http://\[::1\]/](http://[::1]/) verá lo siguiente:



Figura 8: Index interfaz.

Aquí deberá ingresar in texto plano, al apretar *enviar*, enviará un mensaje por POST a un script de python que va a crear el arreglo de sufijos y el árbol de sufijos siguiendo los pasos antes mencionados.

Para mostrar la app ingresaremos el archivo *frutas.txt*, cabe mencionar que nuestro programa espera que los archivos terminen con \$. El archivo contiene:

```
BANANA AMARILLA  
MANZANA ROJA  
PERA VERDE$
```

Un formulario con un fondo verde oscuro. En el centro hay un recuadro blanco con esquinas redondeadas. Dentro del recuadro, al principio, está el título "Ingresa un texto". Debajo del título hay un botón rectangular con el texto "Seleccionar archivo" y, a su derecha, el texto "frutas.txt". Debajo de este botón hay un botón rectangular más largo con el texto "Enviar".

Figura 9: Se agrega frutas.txt

Al terminar de crear el arreglo y árbol de sufijos, va a mostrar el siguiente formulario:

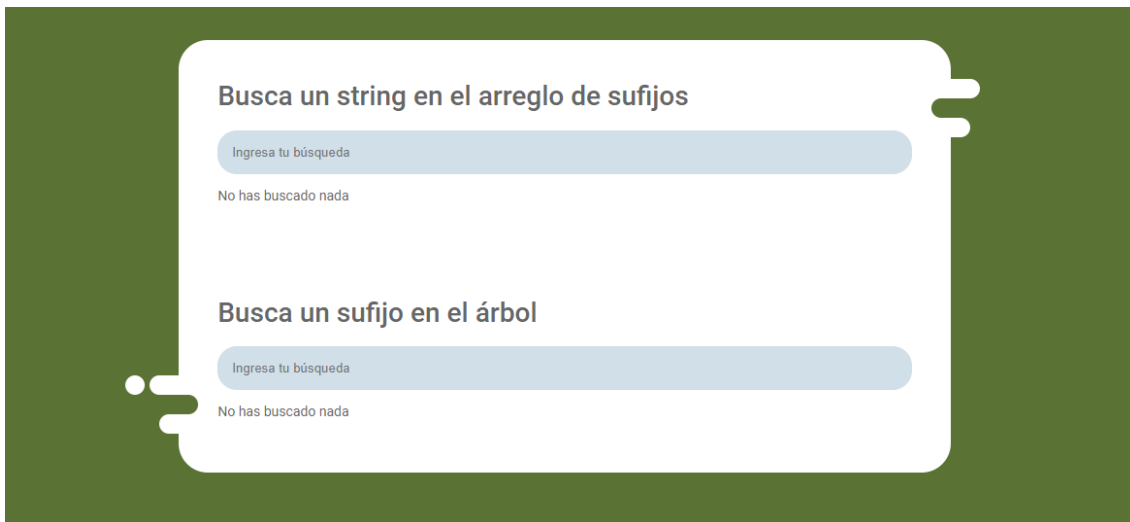
Un formulario con un fondo verde oscuro. En el centro hay un recuadro blanco con esquinas redondeadas. Dentro del recuadro, al principio, está el título "Busca un string en el arreglo de sufijos". Debajo del título hay un botón rectangular con el texto "Ingresa tu búsqueda". Debajo de este botón está el texto "No has buscado nada". Más abajo, dentro del mismo recuadro, está el título "Busca un sufijo en el árbol". Debajo de este título hay otro botón rectangular con el texto "Ingresa tu búsqueda". Debajo de este segundo botón está el texto "No has buscado nada".

Figura 10: Formulario de búsqueda.

Donde el primer input permite buscar en el arreglo de sufijos, para el cual muestra como una lista de sugerencias las líneas donde el input ingresado es substring (note que muestra desde que empieza el input ingresado hasta que termina la línea):

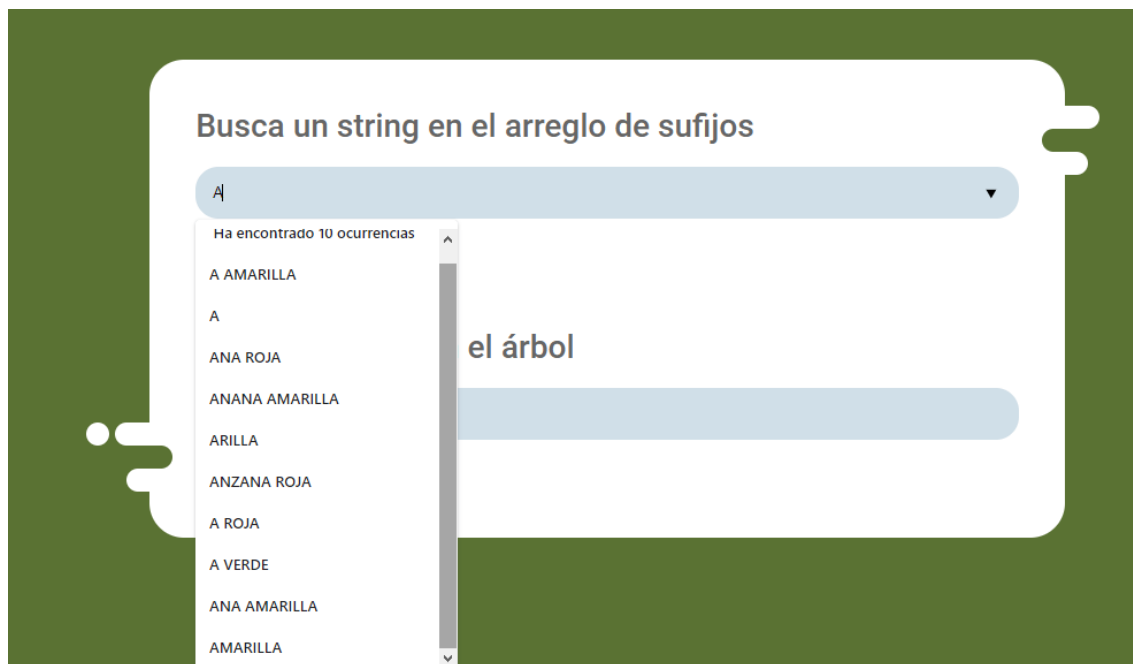


Figura 11: Buscando 'A' en el arreglo de sufijos.

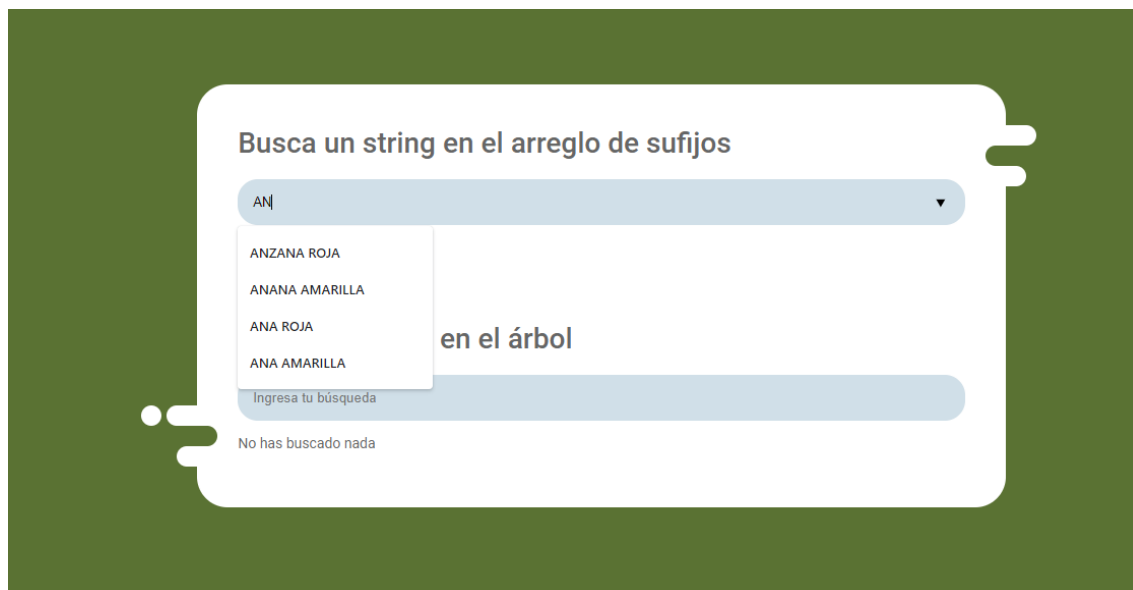


Figura 12: Buscando 'AN' en el arreglo de sufijos.

El segundo input permite buscar en el árbol de sufijos, al ingresar un string lanza una lista de a los más 3 elementos, sugiriendo caminos existentes en el árbol:

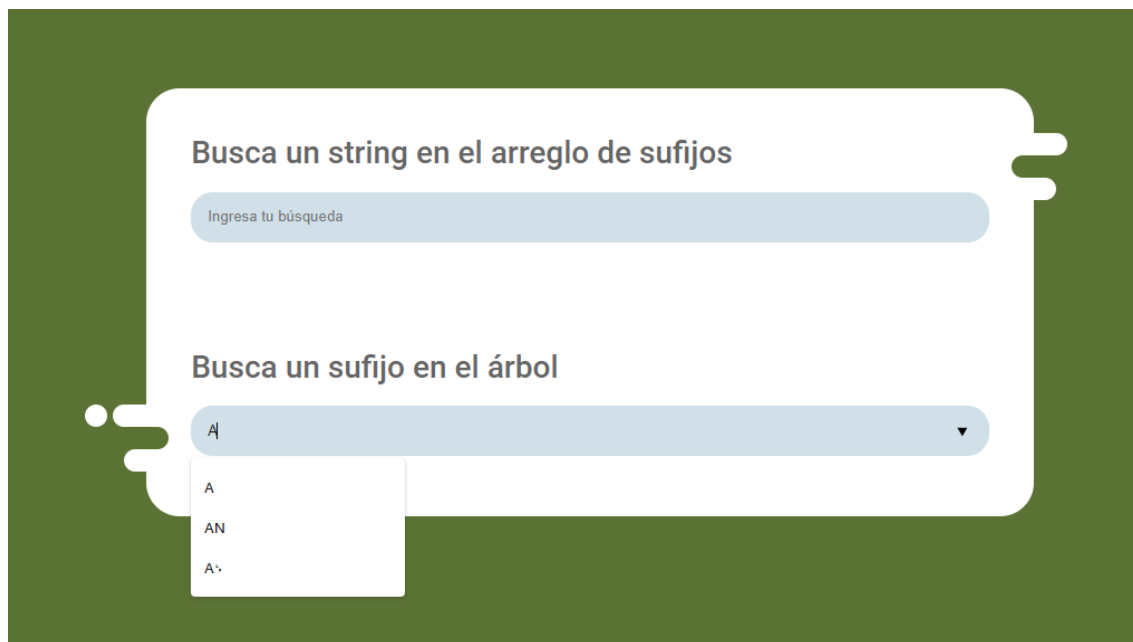


Figura 13: Buscando 'A' en el árbol de sufijos.

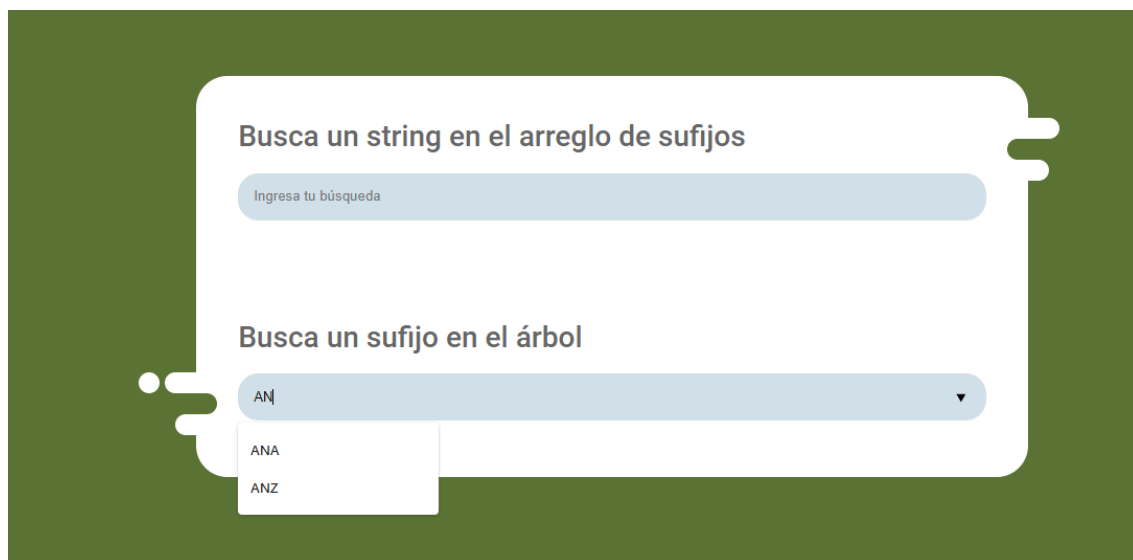


Figura 14: Buscando 'AN' en el árbol de sufijos.

Cada vez que se ingresa tipea va a ir a consultar la lista o el árbol, ya que por detrás se usa AJAX.

Nota: En algunos casos, al final de una sugerencia, se ve una expresión extraña que representa el salto de línea. Elegimos no tratar dicho caso porque no es posible escribir un salto de línea, si apretamos 'enter' se envía el form y todo muere.