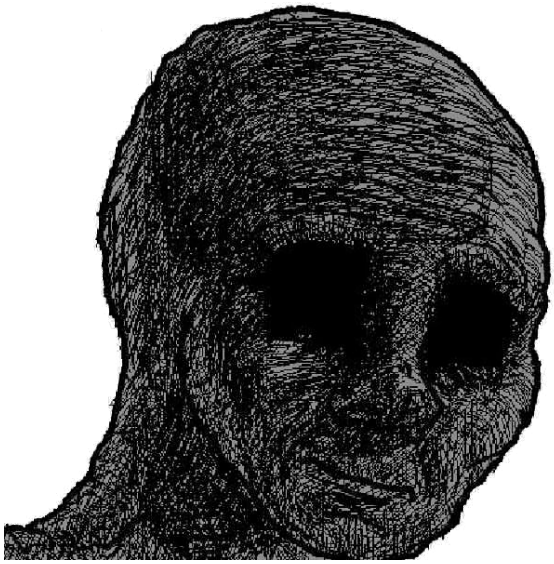
Tarea 2

CC4102-1 Análisis y diseño de algortimos



Profesor: Gonzalo Navarro

Estudiantes: Nicolás García
Javier Lavados
Lung Pang

Introducción

En este presente informe se estudian dos estructuras de datos distintas para realizar busquedas y autocompletado sobre un texto definido. Las estructuras estudiadas e implementadas son el arreglo de sufijos *suffix array*, y un arbol de sufijos *suffix tree*.

Los detalles de la implementación de estas estructuras se detallan y explican a lo largo del informe. Además, se desarrollaron dos programas de prueba interactivos con el usuario para poder estudiar y analizar ambas implementaciones.

El primer programa consiste en la implementación de un árbol de sufijos sobre un archivo de texto ingresado por el usuario en la ejecución. Este lee el archivo ingresado y crea su arreglo de sufijos correspondiente con el fin de poder buscar los sufijos pedidos por el usuario a través de la consola.

El segundo programa, por otro lado, crea primero el arreglo de sufijos del texto ingresado por el usuario con el fin de ingresar todos los sufijos a un Árbol Patricia. Con esto, tenemos un árbol de sufijos implementado y listo para realizar búsqueda de texto y otras funciones sobre la estructura.

Finalmente, analizaremos la construción, complejidad y tiempo de ejecución de ambos programas para compararlos y poder sacar conclusiones al respecto.

Implementación

Etapa 1: Arreglo de sufijos

El arreglo de sufijos se encuentra implementado en `suffix_array.py`, para esto se creó la clase `Suffix`, el cual tiene como parámetros index y el suffix, donde el primero corresponde a la posición en el cual parte el sufijo en text y el suffix contiene el string correspondiente. Además, se tiene la funcion `suffix_array(text)`, la cual se encarga de generar el arreglo de sufijos a partir del text que se le entregue.

Finalmente, se implementó la función `search_all(sa,phrase,fulltext)`, el cual dado un arreglo de sufijos, una frase y el texto fulltext, se obtiene mediante una búsqueda binaria todas las ocurrencias de la frase entregada con respecto a los sufijos.

Etapa 2: Ocurrencias de un usuario

Toda la lógica mencionada en la etapa anterior se lleva a cabo en el archivo `program_array.py`, donde dado un archivo de texto, se procede a crear su arreglo de sufijos correspondiente mediante las funciones mencionadas previamente. Luego el programa entra en un loop esperando las palabras a buscar dentro del texto, si encuentra n ocurrencias, el programa printeará en consola un *snippet* de estas. En cambio, en el caso de no haber encontrado ocurrencias, no se printeará nada y se le avisará al usuario que no existieron ocurrencias encontradas.

Para cortar el programa simplemente se deberá presionar Enter en consola, sin haber escrito nada en esta. Para ejecutar correctamente este archivo, se invita a leer la sección de Modo de usuario.

Etapa 3: Árbol de sufijos

El árbol de sufijos se implementa mediante un árbol Patricia, para ello se crean las clases `PatriciaNode()` y `Patricia()` correspondientes a los nodos y al árbol respectivamente. `PatriciaNode()` es una estructura que posee los campos `self.word`, `self.children`, `self.count_sons` y `self.most_popular` los cuales almacenan la palabra del nodo actual, los nodos hijos en un diccionario, la cantidad de hijos del nodo actual y la ruta al nodo terminal con la palabra más popular.

Para el cálculo de la cantidad de hijos del nodo actual y el camino a la palabra con más hijos(popular) se crean los métodos `pre_count_sons()` y `pre_most_popular()` los cuales actualizarán los campos `self.count_sons` y `self.most_popular` del mismo nodo. Estos deben ejecutarse una vez ya ingresados los valores necesarios al árbol, ya que estos deben leer el árbol guardado y poder calcular recursivamente los hijos y el nodo más popular.

La estructura `Patricia()` contiene toda la lógica del árbol patricia, incluidos los métodos de inserción y búsqueda `insert(word)`, `search(word)`. Además, para el autocompletado se crea el método `autocompletado(text)` que dado un string text, entregará sugerencias de autocompletado al usuario basándose en las indicaciones del enunciado, y entregando a lo más 3 opciones de autocompletado.

Etapa 4: Autocompletar

Como se menciona en la sección pasada, el autocompletado se implementa como un método del árbol patricia el cual mientras se va escribiendo una palabra o frase caracter a caracter, el método `autocompletar(text)` recomienda el siguiente caracter en base a la popularidad pues cada vez que el usuario typea se irá bajando por los nodos del árbol.

Modo de usuario

A continuación se muestran los requerimientos e instrucciones para ejecutar las pruebas de los programas detallados anteriormente.

1. Para ejectutar tanto el arreglo como el árbol de sufijos se debe tener previamente instalado `python 3`.
2. Para poder visualizar el widget del modo usuario del árbol de sufijos se necesita instalar la librería `PyQt5` ejecutando el siguiente comando en consola:

   ```
   pip install PyQt5
   ```

   No se preocupe si el editor de texto no detecta las importaciones, basta con la instrucción anterior para poder ejecutar el widget.

Arreglo de sufijos

Para ejecutar el programa de prueba del arrego de sufijos, se hace la primera llamada en consola:

```
python3 program_suffix.py texto.txt
```

Con esto, la consola comienza un programa donde espera los inputs del usuario para buscar dentro del arreglo de sufijos. Para buscar la palabra o frase, se escribe en la consola el texto y se apreta `Enter`, mientras que para salir del programa basta con presionar Enter sin haber escrito ningún texto, o bien podemos interrumpir la ejecución con la instrucción `Ctrl + C`.

Árbol de sufijos

Al igual que en el arreglo de sufijos, para hacer uso del árbol se debe llamar en consola

```
python3 program_tree.py texto.txt
```

Donde `program_tree.py` se encarga de toda la lógica de ejecución, desde crear el árbol de sufijos hasta el funcionamiento del widget y el autocompletado. Una vez ejecutado el comando, aparecerá el widget principal con un mensaje de bienvenida y que mostrará las instrucciones de búsqueda en el árbol de sufijos. En caso de encontrar el texto ingresado, se mostrará en el programa las opciones disponibles para el auto completado, mientras que de no existir el texto ingresado, no se mostrará nada.

El programa se va actualizando dinamicamente cada vez que el usuario typea un caracter.

Conclusiones

Las primeras impresiones sacadas del trabajo presentado son respecto a la dificultad de implementación entre ambas estructuras.

Si bien el Arreglo de Sufijos es simplemente un árbol Patricia al que se le ingresan los sufijos de un texto, su implementación no es trivial y trabaja con muchos casos para la implementación del autocompletado. Esto difiere por completo de la implementación del arreglo de sufijos, pues esta es más corta y más fácil de comprender para un usuario promedio.

Por otro lado, el árbol de sufijos permite métodos en tiempo constante por cada char que serían inmensamente más costosos de implementar en un arreglo de sufijos. En este caso, vimos que la función autocompletar tarda tiempo O(A*s) con A el tamaño del alfabeto y s el largo del string, lo cuál tardaría al menos O(Sx) con Sx la cantidad de sufijos en el array (que como lo estamos ejecutando, es de orden cuadrático).

Finalmente, también se notó que para textos de tamaños grandes (del órden de 100kB) el arbol de sufijos se comporta muy bien y permite una búsqueda en tiempo logarítmico, mientras que el arreglo de sufijos supera por mucho la cantidad de memoria requerida para su uso, haciendolo inviable en la práctica.