

Licenciatura en ciencia de la computación



# LABORATORIO 3

## Complejidad

**Profesor:**  
Nicolas Thériault

**Autor:**  
Sergio Salinas  
Danilo Abellá

# Contents

<b>1</b>	<b>Introducción</b>	<b>3</b>
<b>2</b>	<b>Estrategias utilizadas</b>	<b>4</b>
2.1	Arreglo ordenado con Mergesort . . . . .	4
2.2	Árbol AVL . . . . .	4
2.3	Tabla Hash . . . . .	4
<b>3</b>	<b>Soluciones a cada problema y tiempos</b>	<b>6</b>
3.1	Generar la estructura . . . . .	6
3.1.1	Mergesort . . . . .	6
3.1.2	Árbol AVL . . . . .	6
3.1.3	Tabla Hash . . . . .	6
3.2	Encontrar una palabra . . . . .	7
3.2.1	Mergesort . . . . .	7
3.2.2	Árbol AVL . . . . .	7
3.2.3	Tabla Hash . . . . .	7
3.3	Contar la cantidad de palabras distintas . . . . .	7
3.3.1	Mergesort . . . . .	7
3.3.2	Árbol AVL . . . . .	8
3.3.3	Tabla Hash . . . . .	8
3.4	Encontrar la palabra más utilizada . . . . .	8
3.4.1	Mergesort . . . . .	8
3.4.2	Árbol AVL . . . . .	9
3.4.3	Tabla Hash . . . . .	9
<b>4</b>	<b>Información de Hardware y Software</b>	<b>10</b>
4.1	Software . . . . .	10
4.2	Hardware . . . . .	10
<b>5</b>	<b>Conclusión</b>	<b>11</b>

# 1 Introducción

A continuación se muestran las estrategias utilizadas, los costos de generar las estrategias y como se resolvio cada pregunta pedida con cada estrategia y su tiempo.

A considerar que para encontrar cada palabra en el archivo se hizo una función que solo obtenera los caracteres del archivo y dice que es palabra cuando lo que le sigue no es caracter y todas las palabras siempre están en minúscula. Por eso es que palabra como The y the van a ser la misma y las palabras que lleven guion como that-that van a ser en realidad dos palabras distintas.

## 2 Estrategias utilizadas

### 2.1 Arreglo ordenado con Mergesort

Esta estrategia consiste en crear un arreglo de dos dimensiones, la primera dimensión tiene largo 21000 que es una estimación de la cantidad de palabras que tiene el archivo y cada arreglo asociado a este tiene un largo de 30, que también es una estimación de la cantidad de letras que tiene cada palabra. por lo que en total este método gasta 63000 espacios de memoria cada vez que se ejecuta.

### 2.2 Árbol AVL

Esta estrategia consiste en usar un árbol binario balanceado conocido como Árbol AVL, para la implementación se le hizo una ligera modificación que es agregar a la estructura de un nodo la frecuencia que este aparece, de esta forma en vez de apilar los elementos repetidos en nodos, solo aumenta el contador de frecuencia de este logrando una notoria mejora en memoria, ya que el árbol implementado solo va a ocupar tantos nodos como **palabras únicas** haya en el archivo. De esta forma la estructura del árbol es la que se muestra en la tabla 1.

Table 1: Estructura de un nodo de un árbol AVL

Nodo
Data
Altura
Frecuencia
Puntero Izquierdo
Puntero Derecho

### 2.3 Tabla Hash

Para implementar el algoritmo con funciones de hash se utilizar un arreglo de largo fijo para almacenar las cabeceras de las listas enlazadas que contienen las palabras que se desean almacenar, este arreglo en un principio solo almacena elementos nulos, cuando se quiere ingresar una palabra se pasa esta palabra por una función hash y luego el resultado es utilizado como índice y se agrega a la lista que corresponde a ese índice, cada elemento es ingresado al inicio de la lista.

La función hash que utiliza el algoritmo es simplemente sumar cada letra del string y calcular su mod MAX, donde MAX, es un número que puede ser modificado por el usuario, MAX también es el largo del arreglo de punteros.

La tabla hash siempre va a guardar todas las palabras que tenga el archivo, pero se puede hacer un intercambio de tiempo memoria en el largo del arreglo de punteros, entre más largo sea más memoria utilizara pero habrán menos colisiones lo que se traduce en menos tiempo en operaciones de búsqueda (la búsqueda llega a ser de costo 1 en este caso si no hay colisión) y conteo, entre más corto menos memoria utilizara pero se gasta más tiempo en hacer las operaciones de búsqueda y conteo dentro de las listas enlazadas.

Para propósitos de la medición de tiempo se decidió que el largo del arreglo de punteros sea de largo 800 debido a que más allá de 800 no hay crecimiento en los índices únicos que se pueden obtener por lo que es un desperdicio de memoria.

## 3 Soluciones a cada problema y tiempos

### 3.1 Generar la estructura

Al momento de generar la estructura se considera también el tiempo en leer el archivo, esto es debido a que la función que lee el archivo devuelve cada palabra que hay en el archivo en minúsculas y sin signos y está se ingresa en la estructura de inmediato.

#### 3.1.1 Mergesort

**Tiempo:** 0.084478 segundos

El tiempo total ocupado en el mergesort para ordenar un arreglo ya generado fue de 0.070521 segundos, haciendo del mergesort el peor algoritmo en lo que se refiere al tiempo de generar la estructura.

En lo que se refiere a la memoria mergesort va a ocupar tantos espacios de memoria como palabras haya en el archivo, además cada palabra va a usar 30 bloques de memoria por defecto, aunque se puede solucionar gastando más tiempo en proceso agregando un contador a cada palabra y que cuando se agregue la palabra arreglo se aumente el contador si es que está ya esta repetida, pero eso costaría mucho en implementación en este caso debido a que se está usando arreglos multidimensionales de largo fijo.

#### 3.1.2 Árbol AVL

**Tiempo:** 0.028204 segundos

La generación del árbol gasta más tiempo que en las tablas de hash debido a las comparaciones que se hacen a cada nodo y al balanceo que debe hacer cada vez que se inserta una palabra, aunque en lo que se refiere a memoria árbol AVL gana y por mucho debido a que la cantidad de nodos que usa es igual a la cantidad de palabras distintas que haya en el archivo y no a la cantidad total de palabras que tenga.

#### 3.1.3 Tabla Hash

**Tiempo:** 0.019277 segundos

La generación de la tabla hash es la más rápida debido a que no se hace ninguna comparación durante del proceso, una vez que se obtiene una palabra del archivo, a esta se le aplica una función de hash y está corresponde al índice que en que va esa palabra en el arreglo y la agrega a la lista asociada a ese índice, cuando hay colisión no hay gasto extra de tiempo debido a que la palabra se inserta siempre al inicio de la lista.

En lo que se refiere a la memoria está va a ser igual a la cantidad de palabras que tenga el archivo.

## 3.2 Encontrar una palabra

### 3.2.1 Mergesort

**Tiempo si la palabra está:** 0.000938 segundos

**Tiempo si la no palabra está:** 0.001291 segundos

La búsqueda en buscar una palabra en arreglo ordenado es lineal, debido a que se va a comparar cada palabra del arreglo desde el inicio hasta el final con la palabra que se desea encontrar. No se gasta memoria extra en el proceso.

### 3.2.2 Árbol AVL

**Tiempo si la palabra está:** 0.000004 segundos

**Tiempo si la no palabra está:** 0.000003 segundos

La búsqueda de una palabra en árbol es la más rápida ya que es de complejidad logarítmica, solo se va comparando los nodos desde la raíz y dependiendo si es mayor o menor pasa a la rama que le corresponde. No hay mayor gasto de memoria en la búsqueda.

### 3.2.3 Tabla Hash

**Tiempo si la palabra está:** 0.000002 segundos

**Tiempo si la no palabra está:** 0.000003 segundos

Para encontrar una palabra en la tabla en la tabla de hash se le aplica a la palabra la función hash y se obtiene el índice en donde se debe encontrar está, luego se compara cada palabra de la lista asociado a ese índice y se busca la palabra ahí. Por lo que el coste es lineal ya que el problema se reduce a buscar una palabra en una lista.

## 3.3 Contar la cantidad de palabras distintas

### 3.3.1 Mergesort

**Tiempo:** 0.084478 segundos

Para contar la cantidad de palabras distintas se compara cada palabra del arreglo con la que sigue y cuando son distintas se aumenta el contador a uno, como el arreglo está ordenado todas las palabras iguales están juntas y no dispersas a lo largo del arreglo. El arreglo se recorre 1 vez y se hacen  $n$  comparaciones por lo que la complejidad es lineal.

### 3.3.2 Árbol AVL

**Tiempo:** 0.000369 segundos

Contar la cantidad de palabras distintas en el árbol es más rápido debido a que el árbol solo contiene las palabras que son distintas, por lo que solo se deben contar la cantidad de elementos en el árbol. La complejidad es lineal ya que debe contar y comparar los  $n$  elementos del árbol.

### 3.3.3 Tabla Hash

**Tiempo:** 0.005666 segundos

Contar la cantidad distintas de elementos de una tabla hash es más complicado que en los algoritmos anteriores, esto es debido a que las coaliciones en la tabla de hash hace que tenga una cantidad grande de listas desordenadas. Por lo que el problema se trata de encontrar los elementos de una lista desordenadas  $k$  veces, donde  $k$  es el número de índices con colisiones en la tabla.

Para contar los elementos distintos en una lista desordenada es un principio se hizo por fuerza bruta que es comparando cada elemento con todos los siguen delante de él, pero luego de opto por hacer un intercambio tiempo memoria creando una una lista temporal en donde se guardan los elementos únicos, comparando cada elemento de la lista desordenada con los de la lista temporal y luego calculando el largo de la lista de elementos únicos. Se decidió hacer esto debido a que la listas por lo general tienen una gran cantidad de elementos repetidos por lo que la lista temporal no va a tener más de uno o dos elementos haciendo que en muchos casos la complejidad sea lineal ya que solo se termina comparando un elemento único en la lista temporal con el resto de elementos que están repetidos en la lista.

## 3.4 Encontrar la palabra más utilizada

### 3.4.1 Mergesort

**Tiempo:** 0.001357 segundos

Para encontrar la palabra más utilizada solo se recorre el arreglo ordenado contando la frecuencia de cada palabra, se compara cada palabra con la siguiente y si son distintas se compara la frecuencia máxima alcanza hasta el momento con la frecuencia de repeticiones que obtuvo esa palabra, si la palabra tiene más frecuencia entonces esa se convierte en la nueva frecuencia máxima y se se guarda esa palabra en un auxiliar. Luego se devuelve la frecuencia máxima y la palabra que se envió por referencia.



La complejidad es lineal debido a que solo se compara cada palabra con la que le sigue hasta el final del arreglo.

### 3.4.2 Árbol AVL

**Tiempo:** 0.000331 segundos

Para encontrar la palabra máxima utilizada se usan dos algoritmos, ambos aprovechan la estructura de los nodos que además guardar de la palabra almacenan la cantidad de veces que se repite esta, el primero busca la palabra con la mayor frecuencia en árbol de manera recursiva comparando los elementos que desde la hoja hasta la raíz y devolviendo siempre el mayor. El segundo algoritmo recorre todo el árbol buscando el nodo que tenga la frecuencia máxima ya encontrada anteriormente y la muestra. La complejidad es lineal ya que se recorre dos veces todo el árbol de  $n$  elementos, debido a que árbol está ordenado por el valor de las palabras y no su frecuencia.

### 3.4.3 Tabla Hash

**Tiempo:** 0.004188 segundos

Para encontrar el mayor elemento entre las listas de la tabla de hash se aprovecho de que las mayoría de las palabras que están repetidas están siempre juntas y la mayor siempre está en mayor cantidad así que se uso al algoritmo "Boyer-Moore majority vote algorithm" que logra encontrar el mayor elemento de una lista de manera lineal siempre y cuando este se repita una gran cantidad de veces. Se aplico el algoritmo a cada sublista de la tabla y cada vez que encuentra el mayor elemento se usa otro algoritmo que cuenta la cantidad de veces que se repite ese algoritmo, el mayor elemento es guardado en un auxiliar cuando a su frecuencia y cuando se encuentra otro mayor elemento se reemplaza, así hasta que se haya encontrado el mayor elemento de la tablad de hash.

## 4 Información de Hardware y Software

Se muestra información sobre la maquina usada para las pruebas mostradas en el informe.

### 4.1 Software

- SO: Xubuntu 16.04.1 LTS
- GMP Library
- Mousepad 0.4.0

### 4.2 Hardware

- AMD Turion(tm) X2 Dual-Core Mobile RM-72 2.10GHz
- Memoria (RAM): 4,00 GB(3,75 GB utilizable)
- Adaptador de pantalla: ATI Raedon HD 3200 Graphics

## 5 Conclusión

Se concluye que en lo que es generar la estructura la tabla hash es más eficiente ya que no requiere costos extras como lo son las operaciones de balance que usa el árbol o usar un algoritmo de ordamiento como lo hace el arreglo. Pero en lo que se refiere a memoria gana el árbol ya que permite guardar los elementos repetidos en el mismo nodo usando un contador sin usar tiempo extra.

En lo que se refiere a buscar dentro de la estructura gana tabla de hash, pero solo si no hay colisiones en el índice en que está palabra, si hay muchas colisiones entonces gana el árbol AVL ya que puede que haga menos comparaciones que lo que hace la tabla de hash. Con muchas colisiones de distintas palabras tabla hash y un arreglo ordenado tienen a ser parecidos, aunque siempre va a hash ya que mientras el arreglo tiene todas las palabras, una lista de tabla de hash va a tener las correspondan a esa colisión.

Para contar la cantidad de palabras distintas gana el árbol AVL ya que el problema se reduce a contar la cantidad de elementos del nodo, el cambio arreglo y tabla hash implica hacer una comparación completa de todos los elementos que tengan, hash ocupa más memoria que los dos ya que para cada sublista crea una lista auxiliar que tenga la cantidad de palabras únicas y gasta más tiempo en contar la cantidad de elementos de esa lista. Arreglo ordenado no usa memoria extra ya solo va comparando cada palabra con la que sigue.

Para contar la palabra más utilizada gana árbol AVL ya que al que tener la frecuencia en que aparece cada palabra solo busca la mayor frecuencia y luego recorre el árbol buscando la palabra que tenga esa frecuencia, todo sin usar memoria extra, el arreglo ordenado usa compara cada elemento con el que le sigue por lo que tampoco usa memoria extra y es más rápido, con tabla hash es más complicado ya que se debe buscar la mayor palabra en cada sublista y luego contar la cantidad de veces que aparece en esa sublista, por lo que usa mayor cantidad de tiempo que todas las demás, aunque no hay uso de memoria extra para esto.

En si, árbol AVL es más eficiente para operaciones que involucren usar todos los elementos que tengan, tabla hash es más eficiente cuando se hacer búsqueda de un solo elemento y un arreglo ordenado termina siendo más eficiente en general para lo que es la implementación.