

Licenciatura en ciencia de la computación



# ALGORITMO DE STRASSEN

## Complejidad

**Profesor:**  
Nicolas Thériault

**Autor:**  
Sergio Salinas  
Danilo Abellá

## Contents

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

## 1 Introducción

## 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 leer el archivo devuelve cada palabra que hay en el archivo en minúsculas y sin signos y está se ingresa en la estructura de inmediato.

El algoritmo que genero la estructura más rápido fue la tabla de hash.

#### 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 la 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:** 0.003609 segundos

**Tiempo si la no palabra está:** 0.003273 segundos

## 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

### 3.3.3 Tabla Hash

**Tiempo:** 0.005666 segundos

## 3.4 Encontrar la palabra más utilizada

### 3.4.1 Mergesort

**Tiempo:** 0.001357 segundos

### 3.4.2 Árbol AVL

**Tiempo:** 0.000331 segundos

### 3.4.3 Tabla Hash

**Tiempo:** 0.004188 segundos