*UNIVERSIDAD AUTONOMA GABRIEL RENE MORENO*

*FACULTAD DE INGENIERIA Y CIENCIAS DE LA COMPUTACION Y TELECOMUNICACIONES*

ESTRUCTURA DE DATOS II

CONTENIDO:

TAREA-6. ABB. FRECUENCIA DE PALABRAS Y REFERENCIAS CRUZADAS.PORCENTAJE TERMINADO: 100%

GRUPO: 14

| INTEGRANTES | DT | HG | HI | EVAL |
|---|---|---|---|---|
| Flores Veizaga Eudenia Gandira | 1 | 1 | 1 | 100 |
| ***Garcia Taboada Brayan Albaro*** | 1 | 1 | 1 | 100 |
| Gedalge Cayhuara Cristian Gabriel | 1 | 1 | 1 | 100 |
| Haquin Serrano Rodrigo | 1 | 1 | 1 | 100 |
| Hernandez Lijeron Roly | 1 | 1 | 1 | 100 |

//DT, días trabajados

//HG, horas grupo

//HI, horas individual

// Eval

**Fecha de presentación** : jueves, 30 de mayo de 2024

**Fecha Presentada :** jueves, 30 de mayo de 2024

**Días de Atraso : 0**

# Contenido



## ÁRBOLES BINARIOS DE BÚSQUEDA.

***FRECUENCIA DE PALABRAS Y REFERENCIAS CRUZADAS DE UN ARCHIVO DE TEXTO.***

***Trabajo Grupal.***

*Hacer un programa para leer el archivo de texto* ***"Startup.txt"****, mostrar las* ***palabras relevantes*** *ordenadas de menor a mayor con sus respectivas frecuencias y el número de línea en la que aparecen.*

*Para resolver este problema, cada nodo del ABB debe contener una Lista de números de línea, que siempre se adicionarán elementos al final de la lista.*

*Modificar, su programa para guardar el número de página en el que se encuentra la palabra. La salida es similar al índice de palabras de un libro. (referencias cruzadas). Definir, la cantidad máxima de líneas de texto que puede contener una página.*

*Realizar otras formas de consultas interesantes sobre las palabras de un archivo de texto, pueden consultar sobre sus mismas estructuras de datos o modificar o adicionar otras estructuras de datos. Aplicar estas consultas sobre otros Archivos de Texto representativos de alguna área.*

## Comentarios

Solo fue necesario un cambio en la clase nodo añadiendo dos nuevos atributos y modificando insertar y leerArchivo.

## Clase Nodo

```
public class Nodo {

   String elem;
   int frec;
   LinkedList<Integer> lineas;
   LinkedList<Integer> paginas;
   Nodo izq, der;
```

```
    public Nodo(String elem) {
        this.elem = elem;
        this.frec = 1;
        this.lineas = new LinkedList<>();
        this.paginas = new LinkedList<>();
        izq = der = null;
    }

    public Nodo(String elem, int frec, LinkedList<Integer> lineas, LinkedList<Integer> paginas) {
        this.elem = elem;
        this.frec = frec;
        this.lineas = new LinkedList<>(lineas);
        this.paginas = new LinkedList<>(paginas);
        izq = der = null;
    }

}
```

---

## Clase Arbol

```
import java.io.*;
import java.util.StringTokenizer;
import java.util.*;

public class Arbol {

    public Nodo raiz;
    public int cantElem;
    public int LINEAS_POR_PAGINA = 50;

    public Arbol() {
        raiz = null;
        cantElem = 0;
    }

    public Arbol(Arbol A1) {
        raiz = copiar(A1.raiz);
        cantElem = A1.cantElem;
    }

    private Nodo copiar(Nodo p) {
        if (p == null) {
            return null;
        }
        Nodo q = new Nodo(p.elem, p.frec, p.lineas, p.paginas);
        q.izq = copiar(p.izq);
        q.der = copiar(p.der);
        return q;
    }
```

```
    public void insertar(String elem, int lin, int pag) {
        raiz = insertarRec(raiz, elem, lin, pag);
    }

    private Nodo insertarRec(Nodo raiz, String elem, int lin, int pag) {
        if (raiz == null) {
            raiz = new Nodo(elem);
            if (!raiz.lineas.contains(lin)) {
                raiz.lineas.add(lin);
            }
            if (!raiz.paginas.contains(pag)) {
                raiz.paginas.add(pag);
            }
            cantElem++;
            return raiz;
        }
        if (elem.compareTo(raiz.elem) < 0) {
            raiz.izq = insertarRec(raiz.izq, elem, lin, pag);
        } else if (elem.compareTo(raiz.elem) > 0) {
            raiz.der = insertarRec(raiz.der, elem, lin, pag);
        } else {
            if (!raiz.lineas.contains(lin)) {
                raiz.lineas.add(lin);
            }

            if (!raiz.paginas.contains(pag)) {
                raiz.paginas.add(pag);
            }
            raiz.frec++;
        }
        return raiz;
    }

    public void inOrden() {
        inOrden(raiz);
    }

    private void inOrden(Nodo nodo) {
        if (nodo != null) {
            inOrden(nodo.izq);
            System.out.println(nodo.elem + ": " + nodo.frec + " | Líneas: " + nodo.lineas + " |
Páginas: " + nodo.paginas);
            inOrden(nodo.der);
        }
    }

    public void inOrdenDesc(Nodo nodo) {
        if (nodo != null) {
            inOrdenDesc(nodo.der);
            System.out.println(nodo.elem + ": " + nodo.frec);
            inOrdenDesc(nodo.izq);
        }
```

```
    }

    public void inordenMenMay() {
        inordenMenMay(raiz);
    }

    private void inordenMenMay(Nodo p) {
        if (p == null) {
            return;
        }
        inordenMenMay(p.izq);
        System.out.println(p.elem + " | " + p.frec);
        inordenMenMay(p.der);
    }

    public void inOrdenMayMen() {
        inOrdenMayMen(this.raiz);
    }

    private void inOrdenMayMen(Nodo p) {
        if (p == null) {
            return;
        }
        inOrdenMayMen(p.der);
        System.out.println(p.elem + " | " + p.frec);
        inOrdenMayMen(p.izq);

    }

    public void leerDesdeArchivo(String nombreArchivo) {
        try (BufferedReader br = new BufferedReader(new FileReader(nombreArchivo))) {
            String linea;
            int NLin = 0;
            int pagina = 1;
            while ((linea = br.readLine()) != null) {
                // Convertir la línea a minúsculas y eliminar signos de puntuación
                linea = linea.toLowerCase().replaceAll("[+*=$%&.,\"\'()\\[\\]{};:!¡¿?1234567890-]", "
");
                StringTokenizer st = new StringTokenizer(linea);
                while (st.hasMoreTokens()) {
                    String palabra = st.nextToken();
                    if (!esPalabraIrrelevante(palabra)) {
                        insertar(palabra, NLin, pagina);
                    }
                }
                NLin++;
                if (NLin % LINEAS_POR_PAGINA == 0) {
                    pagina++;
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
```

```
        }
    }

    private boolean esPalabraIrrelevante(String palabra) {
        Set<String> palabrasIrrelevantes = new HashSet<>(Arrays.asList(
                "y", "o", "el", "la", "los", "las", "un", "una", "unos", "unas",
                "de", "del", "a", "en", "con", "por", "para", "sin", "sobre",
                "entre", "pero", "si", "no", "lo", "al"
        ));
        return palabrasIrrelevantes.contains(palabra);
    }
}
```