UNIVERSIDAD AUTONOMA GABRIEL RENE MORENO

FACULTAD DE INGENIERIA Y CIENCIAS DE LA COMPUTACION Y TELECOMUNICACIONES

ESTRUCTURA DE DATOS II

CONTENIDO:

TAREA-5. ABB, LISTA DE ABBs

PORCENTAJE TERMINADO: 80%

GRUPO: 14

INTEGRANTES	DT	HG	HI	EVAL
Flores Veizaga Eudenia Gandira	1	0	1	80
Garcia Taboada Brayan Albaro	1	0	3	100
Gedalge Cayhuara Cristian Gabriel	1	0	1	80
Haquin Serrano Rodrigo	1	0	1	80
Hernandez Lijeron Roly	1	0	1	80

//DT, días trabajados

//HG, horas grupo

//HI, horas individual

// Eval

Fecha de presentación : Domingo, 26 de mayo de 2024

Fecha Presentada : domingo, 26 de mayo de 2024

Días de Atraso : 0

LISTA DE ÁRBOLES.

Para ambos ejercicios, incluir en la estructura de Arbol, el atributo cantElem, para evitar navegar el árbol para saber la cantidad de nodos y poder realizar consultas en forma directa.

I. Árbol Binario de Búsqueda de Palabras.

Hacer un programa para leer un Archivo de Texto "Startup.txt", manipular las palabras, todos los caracteres en minúsculas. (Convertir todas las cadenas en minúsculas para tener mayor claridad en el análisis de los datos). Encontrar las siguientes salidas.

1. *Mostrar las palabras de menor a mayor con sus respectivas frecuencias.*
2. *Mostrar las palabras de mayor a menor con sus respectivas frecuencias.*
3. *Mostrar las palabras de menor a mayor frecuencia.*
4. *Mostrar las palabras de mayor a menor frecuencia.*

En base a las frecuencias de palabras, concluir un análisis de contenido del archivo de texto.

Para tener mayor claridad en el Análisis de Contenido, no incluir palabras irrelevantes o palabras que sean artículos, preposiciones y otros. Esta información pueden almacenar en alguna estructura de datos y/o leer de un Archivo de Texto de Palabras irrelevantes. Repetir el ejercicio para las consultas anteriores.

CONSULTAS ADICIONALES.

1. *Encontrar las palabras que tienen una frecuencia entre a y b inclusive.*
2. *Encontrar las palabras de mayor frecuencia.*
3. *Encontrar las palabras de menor frecuencia.*
4. *Encontrar las palabras de longitud entre a y b, inclusive.*
5. *Encontrar la cantidad de palabras menores a una palabra.*
6. *Encontrar la cantidad de palabras mayores a una palabra.*
7. *Encontrar la altura de la raíz a una palabra. (distancia de la raíz al nodo que contiene la palabra)*

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

public class Arbol {

   public Nodo raiz;
```

```
public int cantElem;

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);
   q.izq = copiar(p.izq);
   q.der = copiar(p.der);
   return q;
}

public void insertar(String elem) {
   raiz = insertarRec(raiz, elem);
}

private Nodo insertarRec(Nodo raiz, String elem) {
   if (raiz == null) {
      raiz = new Nodo(elem);
      cantElem++;
      return raiz;
   }
   if (elem.compareTo(raiz.elem) < 0) {
      raiz.izq = insertarRec(raiz.izq, elem);
   } else if (elem.compareTo(raiz.elem) > 0) {
      raiz.der = insertarRec(raiz.der, elem);
   } else {
      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);
      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;
            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);
                    }
                }
            }
```

```
        } 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);
    }
//Encontrar las palabras que tienen una frecuencia entre a y b inclusive.

    public void palFrecuencia(int a, int b) {
        inOrden(raiz, a, b);
    }

    private void inOrden(Nodo nodo, int a, int b) {
        if (nodo != null) {
            inOrden(nodo.izq, a, b);
            if (nodo.frec > a && nodo.frec <= b) {
                System.out.println(nodo.elem + ": " + nodo.frec);
            }
            inOrden(nodo.der, a, b);
        }
    }
//Encontrar las palabras de mayor frecuencia.

    public void mayorfrec() {
        lista.add(raiz);
        mayorFrec(raiz);
        for (Nodo nodo : lista) {
            System.out.println(nodo.elem + ": " + nodo.frec);
        }
    }
    public LinkedList<Nodo> lista = new LinkedList<>();

    private void mayorFrec(Nodo p) {
        if (p == null) {
            return;
        }
        if (lista.getFirst().frec < p.frec) {
            lista.clear();
            lista.add(p);
        } else if (lista.getFirst().frec == p.frec) {
            lista.add(p);
        }
        mayorFrec(p.izq);
        mayorFrec(p.der);
```

```
    }

  //Encontrar las palabras de menor frecuencia.
    public void menorfrec() {
      lista.add(raiz);
      menorfrec(raiz);
      for (Nodo nodo : lista) {
        System.out.println(nodo.elem + ": " + nodo.frec);
      }
    }

    private void menorfrec(Nodo p) {
      if (p == null) {
        return;
      }
      if (lista.getFirst().frec > p.frec) {
        lista.clear();
        lista.add(p);
      } else if (lista.getFirst().frec == p.frec) {
        lista.add(p);
      }
      menorfrec(p.izq);
      menorfrec(p.der);

    }
  //Encontrar las palabras de longitud entre a y b, inclusive.

    public void longitud(int a, int b) {
      longitud(raiz, a, b);
    }

    private void longitud(Nodo nodo, int a, int b) {
      if (nodo != null) {
        longitud(nodo.izq, a, b);
        if (nodo.elem.length() > a && nodo.elem.length() <= b) {
          System.out.println(nodo.elem + ": " + nodo.frec);
        }
        longitud(nodo.der, a, b);
      }
    }
  //Encontrar la cantidad de palabras menores a una palabra.

    public void longitud(String A) {
      longitud(raiz, A.length());
    }

    private void longitud(Nodo nodo, int pal) {
      if (nodo != null) {
        longitud(nodo.izq, pal);
        if (nodo.elem.length() < pal) {
          System.out.println(nodo.elem + ": " + nodo.frec);
        }
```

```
            longitud(nodo.der, pal);
        }
    }
//Encontrar la cantidad de palabras mayores a una palabra.

 public void cantPalMayoresA(String A) {
    int cantidad = cantPalMayoresA(raiz, A.length());
    System.out.println("Cantidad de palabras mayores a '" + A + "': " + cantidad);
}

private int cantPalMayoresA(Nodo nodo, int longPalabra) {
    if (nodo == null) {
        return 0;
    }
    int can = cantPalMayoresA(nodo.izq, longPalabra);
    if (nodo.elem.length() > longPalabra) {
        can++;
    }
    can += cantPalMayoresA(nodo.der, longPalabra);
    return can;
}
//Encontrar la altura de la raíz a una palabra. (distancia de la raíz al nodo que contiene la
palabra)
    public int alturaElemento(String elem) {
        return alturaElementoRec(raiz, elem);
    }

    private int alturaElementoRec(Nodo nodo, String elem) {
        if (nodo == null) {
            return -1; // El nodo no se encontró en el árbol
        }
        if (nodo.elem.equals(elem)) {
            return altura(nodo);
        }

        int alturaIzq = alturaElementoRec(nodo.izq, elem);
        if (alturaIzq != -1) {
            return alturaIzq;
        }

        return alturaElementoRec(nodo.der, elem);
    }
     private int altura(Nodo nodo) {
        if (nodo == null) {
            return -1; // Altura de un nodo nulo es -1
        } else {
            int alturaIzq = altura(nodo.izq);
            int alturaDer = altura(nodo.der);
            return 1 + Math.max(alturaIzq, alturaDer);
        }
    }
```