UNIDAD TEMÁTICA 5: Arboles Binarios II

PRACTICOS DOMICILIARIOS INDIVIDUALES -2

Ejercicio #1

Programa JAVA: Cálculo del "costo" de diferentes árboles binarios de búsqueda

Se trata del siguiente escenario: se desea desarrollar un analizador sintáctico para un cierto lenguaje de programación. Para ello, se han de almacenar en un árbol binario de búsqueda las **palabras reservadas** del lenguaje, junto con sus **frecuencias relativas** de búsqueda. En tiempo de compilación, el analizador, entre otras cosas, recorrerá el código fuente del programa desarrollado y necesitará acceder a las palabras reservadas, **pero también a todos los identificadores encontrados** en el código, que **no sean** palabras reservadas, en **forma eficiente**.

EN ESTE EJERCICIO HEMOS DE IMPLEMENTAR LAS FUNCIONALIDADES PARA INSERTAR LAS CLAVES CON SUS FRECUENCIAS DE ACCESO, "COMPLETAR" EL ARBOL PARA REPRESENTAR LAS BUSQUEDAS SIN ÉXITO Y SUS FRECUENCIAS, Y CALCULAR EL "COSTO" DEL ARBOL.

NOTA IMPORTANTE: ESTE PROGRAMA DEBERÁ FUNCIONAR CORRECTAMENTE PARA PODER IMPLEMENTAR EN CLASE LOS ALGORITMOS PARA HALLAR ARBOLES OPTIMOS (ESTO SERÁ CALIFICADO)

VER SECCION DE "CAMBIOS SUGERIDOS" AL FINAL DEL DOCUMENTO, CON SUGERENCIAS SOBRE EL CÓDIGO.

Trabajo a realizar (HACER CADA PARTE EN ORDEN):

a) Dado el archivo de claves o palabras reservadas, insertar cada una de ellas en un árbol binario de búsqueda, junto con su frecuencia de ocurrencia. Cada nodo ha de contener la **clave** – string y un valor asociado que es la **frecuencia** esperada de búsqueda de esa clave – integer (crear un tipo de nodo nuevo que contemple esto). Usar el archivo "**"palabras.txt" (corresponde al conjunto de claves y frecuencias de búsquedas de las mismas del Ejercicio 3 de TA1)**

b) Una vez insertadas todas las claves, es necesario completar el árbol para indicar las búsquedas ***no exitosas***. Para ello, en primer lugar se leerá a un vector (***"vectorbetas", puede estar en el "main")***, que tendrá las frecuencias de las búsquedas sin éxito. Usaremos este vector para poner estas frecuencias en los nodos "externos" que representan las búsquedas sin éxito.). Usar el archivo "**nopalabras.txt" (corresponde a las frecuencias de búsquedas no exitosasdel Ejercicio 3 de TA1)**

c) A continuación se deberá implementar y ejecutar un método "**completarNodosExternos**" en base a las frecuencias ahora almacenadas en el vector "**vectorbetas**". Los nodos externos podrán ser del mismo tipo que los normales, se sugiere ponerles una etiqueta distintiva, por ejemplo **"hoja"** para diferenciarlos. **(VER AL FINAL DE ESTE DOCUMENTO SUGERENCIAS DE CÓDIGO PARA ESTO)**

d) Desarrollar un método del árbol "**obtenerCosto**", que, dada la información ya almacenada en el árbol, emita por consola **el costo total** del árbol de acuerdo a la fórmula.

$$P = \sum_{i=1}^{N} a_i * h_i + \sum_{j=0}^{N} b_j * h_j^{'}$$

(para todos los nodos del árbol, ahora sumar los productos de las frecuencias por el nivel en que se encuentra- **NOTA IMPORTANTE**: EL NIVEL PARA ESTO EMPIEZA EN 1 –). Es fácil resolverlo con un recorrido que lleva el nivel y

suma sobre una variable pasada desde el método de arbol (para que sea por referencia, un vector de una posición, se sugiere de tipo long).

e) Cambiando el orden en que están las claves en el archivo “palabras.txt”, generar todos los posibles árboles binarios de búsqueda y para cada uno calcular el costo (como hicimos manualmente en el Ejercicio 3 de TA1) , Y **VERIFICAR** QUE COINCIDEN CON LO HECHO EN CLASE.

CAMBIOS SUGERIDOS EN ARBOL Y NODO:

EN EL ARBOL

```
/**
        * Método encargado de completar los nodos externos, creando nodos "especiales"
        * del árbol, los cuales contendrán los valores b[j]
        */
        public void completarNodosExternos();

        /**
        * Retorna el costo del árbol
        * @return costo del árbol
        */
        public int obtenerCosto();  //delega en la raíz

        /**
        * variables a usar en el Main:
        * 1) int[] vectorBetas // 0 a cant_claves.
        * 2) int cantClaves    // cantidad de claves en el arbol – se puede saber al leer el archivo o al ejecutar TArbol.tamanio.
        *
         en el método TArbol.completarNodosExternos:
        *int[] contador     // vector de 1 sola posición, utilizado para completar los nodos externos (para referir a la entrada correspondiente en el vector de betas).
        */
```

EN EL ELEMENTO

```
/**
        *agregar campo int frecuencia.
         AGREGAR LOS SIGUIENTES MÉTODOS
        * public int obtenerCosto();
        * public void completarNodosExternos(int[] vector_betas, int[] contador)
        *
        */
```

EJEMPLOS (adaptarlos a el código que cada equipo tiene):

```
//TArbolBB
public void completarNodosExternos(int[] vectorBetas) {
	if (! esVacio()) {
		Integer[] contador = new Integer[1];
		contador[0] = 0;
		raiz.completarNodosExternos(vectorBetas, contador);
	}
}

//TElementoAB
public void completarNodosExternos(int[] vector_betas, Integer[] contador) {
	if (hijoIzq != null) {
		hijoIzq.completarNodosExternos(vector_betas, contador);
	} else {
		IElementoAB elemento = new TElementoAB("hoja", null);
		elemento.setFrecuencia(vector_betas[contador[0]]);
		hijoIzq = elemento;
		contador[0]++;
	}

	if (hijoDer != null) {
		hijoDer.completarNodosExternos(vector_betas, contador);
	} else {
		IElementoAB elemento = new TElementoAB("hoja", null);
		elemento.setFrecuencia(vector_betas[contador[0]]);
		hijoDer = elemento;
		contador[0]++;
	}
}

public class Main {
public static void main(String[] args) {
	TArbolBB arbol = new TArbolBB();
	 String[] lineas = ManejadorArchivosGenerico.leerArchivo("palabras.txt");
	 for (String l : lineas) {
		String[] datos = l.split(" ");
		IElementoAB elem = new TElementoAB(datos[0], Integer.parseInt(datos[1]),
null);
		arbol.insertar(elem);
	 }
	 int cant_claves = arbol.obtenerTamanio();

	 String[] lineasBetas = ManejadorArchivosGenerico.leerArchivo("nopalabras.txt");
	 int[] vectorBetas = new int[cant_claves+1];
	 for (int h = 0; h < lineasBetas.length; h++) {
		String linea = lineasBetas[h];
		String[] datos = linea.split(" ");
		//vectorBetas[h] = Integer.parseInt(datos[1]);
		vectorBetas[h] = Integer.parseInt(datos[0]);
	 }
	 int[] vectorAlfas = new int[cant_claves+1];
	 String[] vectorClaves = new String[cant_claves+1];
	 arbol.completarNodosExternos(vectorBetas);
	 System.out.println(arbol.calcularCosto());
}
}
```