UNIDAD TEMÁTICA 5 – ARBOLES BINARIOS AVL y ÁRBOLES ÓPTIMOS – Trabajo de Aplicación 4

Escenario- Wikipedia

La conocida enciclopedia online tiene varios millones de artículos. Cuando uno busca un artículo, la enciclopedia necesita recorrerlos para devolvernos qué artículos contienen lo que buscamos.

Imaginemos que Wikipedia mantiene como índice de cada archivo un árbol binario. ¿Qué palabras incluirá en el índice? ¿Tiene sentido incluir todas las palabras? ¿O sólo aquellas que tienen un significado o pueden decir de qué trata el artículo?

La realidad es que en los índices no se incluyen las palabras que están en todos los artículos, como “no”, “si”, “el”, “la”, “como”, “entonces”, “también”, etc., sino que se incluyen sólo algunas palabras.

Sin embargo, sigue siendo necesario, cuando se incluyen esas palabras en la búsqueda – que de hecho en el lenguaje natural ocurre muy seguido - decidir si pertenecen o no al índice. Es por esto que para hacer más eficiente la búsqueda, también hay que tener en cuenta la cantidad de comparaciones que son necesarias para decidir las búsquedas infructuosas, porque en todos los índices, el conjunto de palabras que están en el índice siempre va a ser muy menor al conjunto de palabras que NO están en el índice.

En el código fuente ocurre algo similar. Tenemos palabras muy comunes como las **reservadas** de cada lenguaje, que no permiten determinar qué ocurre en cada archivo de código fuente porque la mayoría va a tener palabras como “**public**”, “**if**”, etc.

¿Cómo podemos optimizar las búsquedas en código fuente? ¿podrá un árbol óptimo ayudarnos con esto?

Objetivo del Trabajo de Aplicación:

Dada la información de las claves y sus frecuencias de búsquedas con y sin éxito, se desea desarrollar un algoritmo para hallar el árbol binario de búsqueda óptimo y luego generar el árbol correspondiente.

Ejercicio 1

Desarrollar el algoritmo para obtener el árbol de búsqueda óptimo, utilizando el seudocódigo y las matrices **P**, **W** y **R** vistas en las lecturas. Se provee un fragmento ***incompleto*** de código fuente, que deberá ser completado y probado.

Como entradas se han de utilizar vectores de **0** a **n**, siendo n la cantidad total de claves, para representar:

- Las claves que se insertan en el árbol : de **1** a **n**
- Las frecuencias de búsqueda con éxito : de **1** a **n**
- Las frecuencias de búsquedas sin éxito: de **0** a **n**

PASO 1

- Descargar el paquete UT5_TA4 desde GitHub, revisar las clases provistas.

PASO 2

- Completar el código del método para completar las matrices correspondientes al árbol óptimo (ver código parcial entregado).

PASO 3

- Cargar los archivos de datos y ejecutar el algoritmo para hallar las matrices correspondiente al árbol óptimo

PASO 4

- Completar el código para, a partir de los dato dados y obtenidos, crear el árbol binario de búsqueda óptimo correspondiente.
- Imprimir por consola el **COSTO** del árbol óptimo resultante y la **RAÍZ** del mismo. Emitir el listado en PREORDEN. Comprobar con los valores obtenidos en las matrices de árbol óptimo.

Se provee un archivo **"palabras.txt"** que contiene las claves ordenadas en forma ascendente. En cada línea hay una clave, seguida de su frecuencia de búsqueda, separadas por espacio ($a_1..a_n$).

Se provee un archivo **"nopalabras.txt"** con las frecuencias $b_0..b_n$ de búsquedas sin éxito.

ENTREGA: REMITIR TODO EL CODIGO GENERADO AL REPOSITORIO GIT. SE CALIFICARÁ LO EXISTENTE HASTA LA HORA 21:05

Ejercicio 2

1. Desarrollar un método que, en base a la matriz resultante **R** del Ejercicio 1 y a los vectores conteniendo las claves y sus frecuencias de ocurrencia con éxito, **genere el árbol binario de búsqueda correspondiente**.
2. Utilizar los datos y resultados del ejercicio 2 de TA2 para probar los métodos desarrollados.
3. Emitir el listado en **preorden** del árbol resultante y remitir un archivo **"preorden.txt"** a la tarea de webasignatura.

ENTREGA: REMITIR TODO EL CODIGO GENERADO AL REPOSITORIO GIT y el archivo "preorden.txt" a la tarea. SE CALIFICARÁ LO EXISTENTE HASTA LA HORA 21:05

```
public class CalculadorMatricesOptimo{
        int[][] W;
        int[][] P;
        int[][] R;

//CONSTRUCTOR!!!!!
public CalculadorMatricesOptimo (int cantElem) {
                crearMatrices(cantElem);
        }

private void crearMatrices(int cantElem) {
          W = new int[cantElem + 1][cantElem + 1];
          P = new int[cantElem + 1][cantElem + 1];
          R = new int[cantElem + 1][cantElem + 1];
        }

 public void encontrarOptimo(int cantElem, int[] frecExito, int[] frecNoExito) {
          int i, j, k, kraiz, h;
          int min, pesoSubArboles;
          kraiz = 0;
        /*
         * Paso 1, para h = 0,
              wii = bii y pii = wii
               y completar la matriz W:  wij = wij-1+ aj + bj
        *completa las sentencias necesarias
         */
         *paso 2  para h = 1 pij = wij + pii + pjj
        *completa las sentencias necesarias
         */
        *paso 3   para h = 2 hasta h = N
        kraiz = 0;
        for (h = 2; h < cantElem + 1; h++) {
          for (i = 0; i < cantElem - h + 1; i++) {
             j = i + h;
            min = Integer.MAX_VALUE;
        /*
          Completar el código con la búsqueda del min en k
          Al terminar aquí tenemos min y kraiz
        */
          P[i][j] = min + W[i][j];
          R[i][j] = kraiz;
          }
        }
}
```

// Ejecuta el algoritmo recursivo que inserta las claves en el ArbolBB pasado (vacío)
// como parámetro

```
void armarArbolBinario(int i, int j, String[] Claves, TArbolBB elArbolBB ) {
        TElementoAB unNodo;
         int unaraiz;
         if (i < j) {
                unaraiz = R[i][j];
                /*
                  Completa las sentencias necesarias para implementar
                este  algoritmo recursivo
                 */
         }
        }


}
```