Práctica Árboles Binarios de Búsqueda

Estructuras de Datos

Implementación del árbol binario de búsqueda

Crearemos la unidad `UArbolBB` que implemente el TAD **Árbol Binario de Búsqueda** respetando la ya conocida especificación algebraica.

ESPECIFICACIÓN ArbolBinarioBusqueda

PARÁMETROS GENÉRICOS
TIPOS TipoElemento
FIN PARÁMETROS

TIPOS TipoArbolBin

OPERACIONES

(* constructoras generadoras *)
CrearArbolBinVacio: → TipoArbolBin
ConstruirArbolBin: TipoArbolBin x TipoElemento x TipoArbolBin → TipoArbolBin

(* observadoras selectoras *)
PARCIAL Raiz: TipoArbolBin → TipoElemento
PARCIAL HijoIzdo: TipoArbolBin → TipoArbolBin
PARCIAL HijoDcho: TipoArbolBin → TipoArbolBin

(* observadoras no selectoras *)
EsArbolBinVacio: TipoArbolBin → Booleano

PARAMETROS GENERICOS
OPERACIONES
Mayor: TipoElemento x TipoElemento → Booleano
FIN PARAMETROS
Pertenece: TipoArbolBin x TipoElemento → Booleano
PARCIAL Minimo: TipoArbolBin → TipoElemento

(* constructoras no generadoras *)
Insertar: TipoElemento x TipoArbolBin → TipoArbolBin
Eliminar: TipoElemento x TipoArbolBin → TipoArbolBin

VARIABLES
r, e: TipoElemento;
i, d: TipoArbolBin;

ECUACIONES DE DEFINITUD

DEF(Minimo(ConstruirArbolBin(i, r, d)))
DEF(Raiz(ConstruirArbolBin(i, r, d)))
DEF(HijoIzdo(ConstruirArbolBin(i, r, d)))
DEF(HijoDcho(ConstruirArbolBin(i, r, d)))

ECUACIONES

(* observadoras selectoras *)

Raiz(ConstruirArbolBin(i, r, d)) = r
HijoIzq(ConstruirArbolBin(i, r, d)) = i
HijoDer(ConstruirArbolBin(i, r, d)) = d

(* observadoras no selectoras *)
Pertenece (e, CrearArbolBinVacio) = FALSO
Pertenece (e, ConstruirArbonBin(i, r, d)) = SI e=r →
CIERTO
| SI Mayor(e, r) →
Pertenece(e, d)
| Pertenece(e, i)

```
Minimo(ConstruirArbolBin(i, r, d)) =
  SI EsArbolBinVacio(i) →
     r
  | Minimo(i)


EsArbolBinVacio(CrearArbolBinVacio) = CIERTO
EsArbolBinVacio(ConstruirArbolBin(i, r, d)) = FALSO

(* constructoras no generadoras *)

Insertar(e, CrearArbolBinVacio) =   ConstruirArbolBin(CrearArbolBinVacio, e, CrearArbolBinVacio)

Insertar(e,ConstruirArbolBin(i, r, d))  =   SI (e = r) →
                                                ConstruirArbolBin(i, r, d)
                                            | SI Mayor(e, r) →
                                                ConstruirArbolBin(i, r, Insertar(e, d))
                                              | ConstruirArbolBin(Insertar(e, i), r, d)


Eliminar(e, CrearArbolBinVacio) = CrearArbolBinVacio
Eliminar(e, ConstruirArbolBin(i, r, d)) =
  SI (e = r) →
     SI EsArbolBinVacio(d) →
       i
     | ConstruirArbolBin(i, Minimo(d), Eliminar(Minimo(d), d))
  | SI Mayor(e, r) →
       ConstruirArbolBin(i, r, Eliminar(e, d))
     | ConstruirArbolBin(Eliminar(e, i), r, d)

FIN ESPECIFICACION
```

Uso del árbol binario de búsqueda

Para probar la utilidad del árbol binario se pide desarrollar un programa que clasifique una lista de *tweets* por los *hashtags* que aparecen en ellos. Un *tweet* es una publicación o actualización de estado realizada en la red social *Twitter*. Un *tweet* es un mensaje de texto con extensión máxima de 140 caracteres (están permitidos letras, números, signos y enlaces). Los *tweets* pueden contener *hashtags* o etiquetas, que permiten establecer el tema del que trata el *tweet*. *Hashtag* es una palabra compuesta por *hash* (almohadilla) y *tag* (etiqueta) y son palabras que empiezan por el símbolo almohadilla (#), seguido de una palabra (por ejemplo: #programación, #cloud, etc.).

Para el desarrollo de la práctica se proporcionan dos ficheros de texto:
- hashtags.txt, que almacena un hashtag en cada línea
- tweets.txt, que almacena un tweet en cada línea

El etiquetado de *tweets* lo vamos a implementar como un árbol binario de búsqueda en el que cada nodo almacene un registro con dos campos:
- `hashtag`: que almacena un solo *hashtag*.
- `listaTweets`: que almacena una **lista dinámica** de *tweets* que contengan este *hashtag*. La lista debe ser dinámica, ya que no conocemos a priori el número de *tweets* que van a contener cada *hashtag*. Puedes utilizar una implementación de listas que tengas ya hecha.

Esta representación y la funcionalidad asociada deberá ir implementada en una unidad que se utilizará como el `TipoElemento` del árbol binario de búsqueda.

De modo que lo que se pide es:
1.- *Construir la estructura fundamental del ABB*: para ello, leer la lista de *hashtags* proporcionada en hashtags.txt (cada tweet es una línea del fichero de texto) y construir el

árbol binario de búsqueda de *hashtags* ordenado por orden alfabético. Para la construcción del ABB, recuerda que sólo debes utilizar la interfaz de la unidad `UArbolBB`.
2.- *Etiquetado de tweets*: Una vez construido el árbol, leer cada uno de los *tweets* almacenados en tweets.txt (cada *tweet* es una línea del fichero de texto). Para cada *tweet*, buscar los *hashtags* que contenga y almacenar el *tweet* en el nodo del ABB que corresponda al *hashtag*. Por ejemplo, un tweet como el siguiente 'Sería feliz estando todo el tiempo de #viajesPorElMundo' debería estar almacenado en la `listaTweets` del nodo cuyo valor del campo `hashtag` coincida con '#viajesPorElMundo'. Por otro lado, puede suceder que un *tweet* contenga más de un *hastag*.
3.- *Consultas*: como resultado, hemos construido una base de datos de juguete y podemos implementar la funcionalidad que permita realizar consultas de *tweets* por *hashtag*. Es decir, el usuario podrá preguntar por un *hashtag* cualquiera y el programa deberá responder con la impresión por pantalla de los *tweets* que contienen el *hashtag* consultado o, en su caso, que no se ha encontrado ningún *tweet* con ese *hashtag*.
4.- ¿Qué diferencia habría si en lugar de un árbol binario de búsqueda utilizamos una lista?

Funciones útiles:
Para el manejo de cadenas, se recomienda el uso de la unidad `sysutils` y `strutils`. En concreto:
- `ansicomparetext(s1,s2)`: función que compara los strings `s1` y `s2`. Si `s1=s2` devuelve 0, Si `s1<s2`, devuelve un entero negativo y si `s1>s2`, devuelve un entero positivo. Está en `sysutils`.
- `posEx(c,s)`: función que devuelve la posición del carácter `c` dentro de la cadena `s`. Está en `strutils`.
- `ExtractSubstr(s,p,delim)`: función que recibe una cadena `s`, una posición `p` y un conjunto de delimitadores delim y devuelve la palabra que empieza en la posición `p` de `s`. Nota: `delim` debe ser declarada como una variable de tipo `TSysCharSet` e inicializada como `delim := [' ']`. Está en `strutils`.

Nota: Como siempre, se hará uso de las normas de estilo dictadas en clase (cabecera del fichero, interfaz de la unidad con precondiciones, postcondiciones, excepciones, implementaciones con el análisis de complejidad de cada operación, nombres coherentes de variables y operaciones,...)

Plantilla de cabecera del fichero:

```
{*****************************************************************************
*       Módulo:                                                                 *
*       Tipo:  Programa()     Interfaz-Implementación TAD ()       Otros()      *
*       Autor/es:                                                               *
*       Fecha de actualización:                                                 *
*       Descripción:                                                            *
*****************************************************************************}
```