Estructuras de Datos y Algoritmos - EDA (ETS de Ingeniería Informática).
Curso 2023-2024

Práctica 5. Implementación de la función *Autocompletar texto* usando un ABB equilibrado

Departamento de Sistemas Informáticos y Computación. Universitat Politècnica de València

1. Objetivos

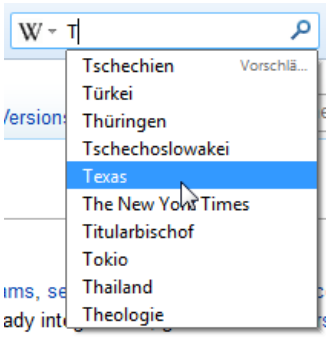
El fin principal de esta práctica es que el alumno aplique al diseño de una aplicación concreta los conceptos sobre Árbol Binario de Búsqueda (ABB) que ha estudiado en el Tema 5 de la asignatura. En concreto, los objetivos son los siguientes:

- Construir de forma eficiente un ABB equilibrado a partir de un conjunto de datos, y re-equilibrar un ABB ya existente cuando haya perdido el equilibrio (después de diversas operaciones de inserciones y/o borrados).
- Implementar la función autocompletar de un *Editor Predictivo* utilizando un ABB equilibrado.

2. Descripción del problema

El autocompletado de palabras es una característica que tienen muchas aplicaciones que trabajan con texto, y que consiste en predecir el resto de una palabra a partir del prefijo que el usuario escribe. El sistema sugiere una lista de opciones y el usuario puede elegir la correcta. Así, se facilita la interacción del usuario con el sistema.

Para implementar esta funcionalidad se tiene un diccionario de palabras, que podría incluir información sobre la frecuencia de uso de las mismas. Este diccionario se podría actualizar para adaptarse al usuario añadiendo o eliminando términos. En esta práctica, este diccionario se va a representar como un Árbol Binario de Búsqueda (ABB) de palabras y, dado el conjunto de caracteres escritos por el usuario (prefijo), la lista de sugerencias de completado será la secuencia ordenada de palabras que comienzan por dicho prefijo. Para ello, se utilizará el método `sucesor(e)` de un `ABB<E>` que permite obtener el elemento del `ABB` siguiente a `e` según el orden establecido en `E`. El coste de esta operación, así como los de las operaciones de búsqueda, inserción y borrado en un ABB, depende de la topología del mismo. Solo un ABB equilibrado permite realizar estas operaciones en tiempos logarítmicos con la talla del ABB.



3. Actividades

Antes de llevar a cabo las actividades propuestas, es necesario actualizar la estructura de paquetes y ficheros del proyecto *BlueJ eda* siguiendo estos pasos:

- Añadir al subpaquete `librerias.estructurasDeDatos.jerarquicos` las clases `ABB`, `NodoABB` y `GUIAbb`, disponibles en PoliformaT.
- Crear el paquete `aplicaciones.editor`.
- Añadir las clases `EditorPredictivo` y `GUIEditorPredictivo`, y el fichero `castellano.txt`, disponibles en PoliformaT, en el paquete `aplicaciones.editor`.

3.1. Implementación eficiente de un `ABB`

Actividad #1: Implementación de los métodos para construir el ABB equilibrado

Como se ha comentado, la implementación de la clase ABB debe asegurar que los objetos ABB tengan alturas casi logarítmicas con su talla. La forma o topología de un ABB depende del orden de inserción de sus elementos. Si el orden en el que se insertan los datos es el adecuado, la relación altura número de nodos será óptima. Una forma de conseguir esto es la siguiente:

- Copiar los datos a añadir al ABB en un array `v` en orden no decreciente.
- Recursivamente, hasta que no queden datos por tratar:
  - Insertar `v[mitad]`, la mediana, como nodo raíz del árbol.
  - Construir el hijo izquierdo del nodo, con los datos situados en el array a la izquierda de `mitad`.
  - Construir el hijo derecho del nodo, con los datos situados en el array a la derecha de `mitad`.

De acuerdo con este algoritmo, hay que completar los siguientes métodos de la clase `ABB`:

1. `construirEquilibrado`: dado un subarray `v[ini, fin]` ordenado de forma no decreciente, devuelve la raíz del ABB equilibrado con los elementos de dicho subarray:

```
protected NodoABB<E> construirEquilibrado(E[] v, int ini, int fin) { ... }
```

2. `ABB`: constructor que crea un ABB equilibrado con los elementos del array que se pasa como parámetro usando el método `construirEquilibrado`:

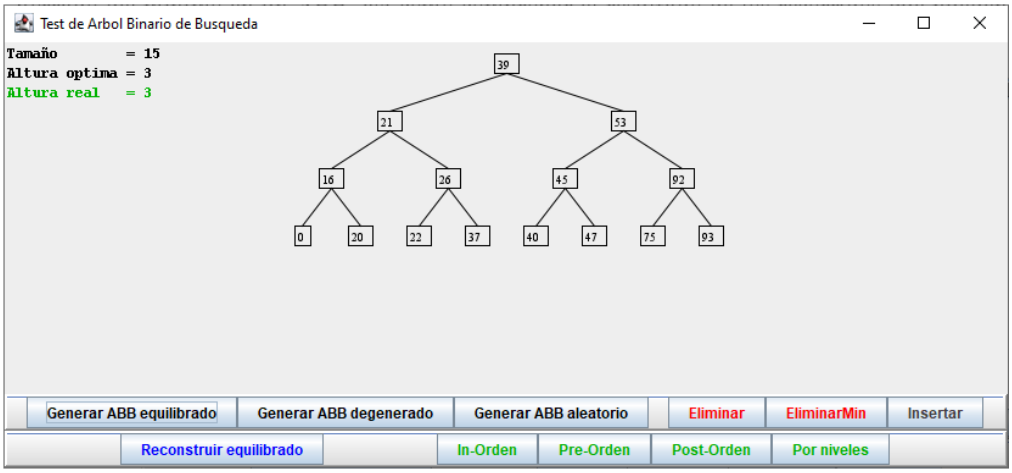
```
public ABB(E[] v) { ... }
```

3. `reconstruirEquilibrado`: que reequilibra `this ABB` utilizando los métodos `construirEquilibrado` y `toArrayInOrden`:

```
public void reconstruirEquilibrado() { ... }
```

Actividad #2: Validación de la clase `ABB`

Ejecuta el programa `GUIAbb` para comprobar el funcionamiento de los métodos diseñados. En concreto, usa los botones *"Generar ABB equilibrado"* y *"Generar ABB aleatorio"*, y después *"Reconstruir equilibrado"*. En la figura siguiente, se muestra la ejecución de este programa una vez seleccionado el botón *"Generar ABB equilibrado"*. También puedes observar el funcionamiento de los métodos de inserción, borrado de elementos y borrado del mínimo en un ABB, así como comprobar el resultado de los recorridos que pueden realizarse sobre él (por niveles, in-orden, pre-orden y post-orden).



3.2. Implementación de la función *Autocompletar palabras*

La función de autocompletado de palabras ofrecerá una serie de sugerencias para completar la palabra conforme el usuario la escriba (y sin que necesariamente tenga que terminarla). Para implementarla se ha diseñado la clase `EditorPredictivo` como un `ABB<String>`. En concreto, con la siguiente estructura:

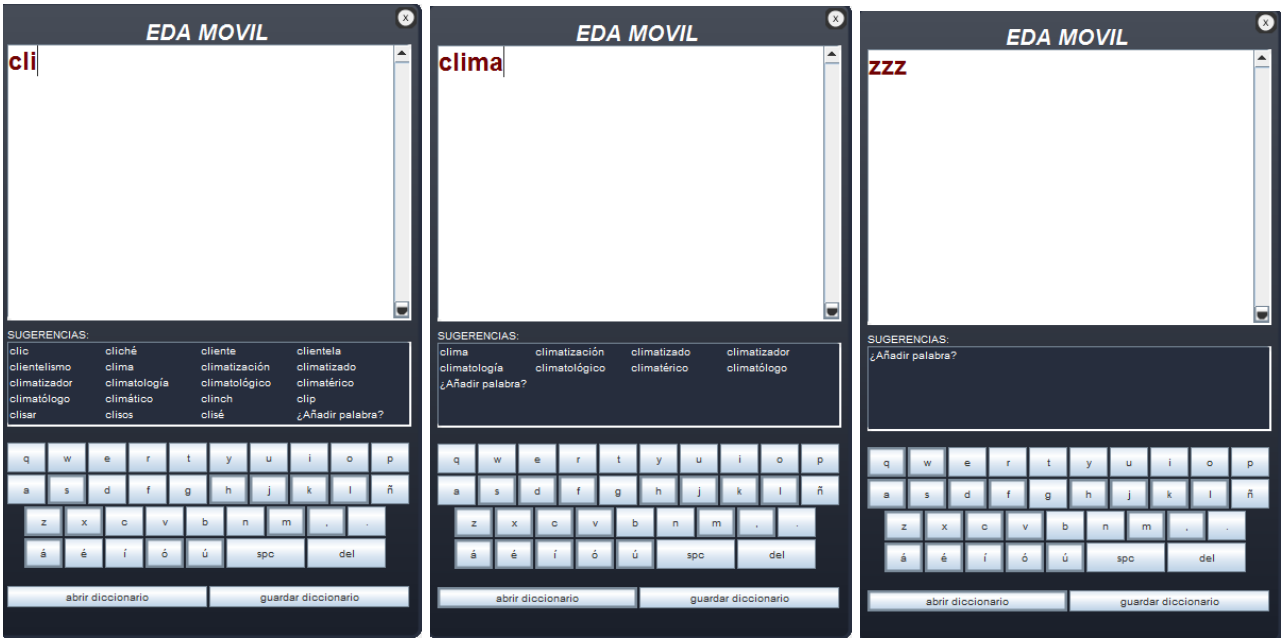
```
public class EditorPredictivo extends ABB<String> {
    public EditorPredictivo() { ... }
    public EditorPredictivo(String nombreFichero) { ... }
    public void incluir(String nueva) { ... }
    public void guardar(String nombreFichero) { ... }
    public ListaConPI<String> recuperarSucesores(String prefijo, int n) { ... }
}
```

Al construir los objetos de esta clase, se lee el fichero `castellano.txt`, que contiene más de 46.000 palabras en castellano ordenadas alfabéticamente (los datos con los que se construirá un *Editor Predictivo*). En concreto, el segundo método constructor de la clase `EditorPredictivo`, dado ese fichero de palabras, crea un `ABB` equilibrado de `String` usando el método `construirEquilibrado` para insertar las palabras del array (obteniendo así un ABB equilibrado).

Esta clase `EditorPredictivo` será utilizada por la interfaz gráfica `GUIEditorPredictivo`. En la siguiente figura se muestra la ejecución de esta interfaz para tres entradas: `cli`, `clima` y `zzz`.

- En el primer caso, devuelve la lista de sugerencias: `clic, cliché, cliente, clientela, clientelismo, clima, climatización, climatizado, climatizador, climatología, climatológico, climatérico, climatólogo, climático, clinch, clip`...
- En el segundo caso, devuelve la lista de sugerencias: `clima, climatización, climatizado, climatizador, climatología, climatológico, climatérico y climatólogo`.
- En el último caso, la lista devuelta está vacía porque no hay ninguna palabra que tenga como prefijo `zzz`.

Esta interfaz gráfica permite, además, abrir un diccionario diferente, o añadir nuevas palabras al diccionario, y guardarlo.



Actividad #3: El método `recuperarSucesores` de la clase `EditorPredictivo`

Completar el método `recuperarSucesores` de la clase `EditorPredictivo`, cuyo perfil es el siguiente:

```
public ListaConPI<String> recuperarSucesores(String prefijo, int n)
```

Este método tiene que devolver una `ListaConPI` con los `n` primeros sucesores de un `prefijo` dado. Esta lista debe incluir a `prefijo` como primer elemento si está en el ABB (es decir, que el prefijo sea una palabra completa). Para implementar este método, se puede usar el método `sucesor` de la clase `ABB` como sigue:

1. Buscar el prefijo en el `ABB`, por si este fuera ya una palabra completa.
2. Recuperar del `ABB` los siguientes sucesores del prefijo hasta encontrar uno que ya no comience por dicho prefijo o hasta que se hayan añadido `n` sucesores.

Actividad #4: Prueba del Editor Predictivo

Comprueba la implementación hecha, usando la interfaz gráfica `GUIEditorPredictivo`. Además de los ejemplos de la figura anterior, a continuación se muestran las sugerencias que deben aparecer para una serie de prefijos de prueba:

Prefijo	**Sugerencias**			
catar	catar	catarata	catarral	catarro
	catarroso	catarsis		
mar	mar	mara	marabunta	maraca
	maraco	maracucho	maracuyá	maraquear
	maraquero	marar	marasmo	maratoniano
	maratón	maravilla	maravillar	maravillosamente
	maravilloso	maraña	marañero	
tene	tenebrosidad	tenebroso	tenedor	teneduría
	tenencia	tener	teneraje	tenerife
	tenería			
criti	criticable	criticador	criticar	criticastro
	criticidad	criticón	critiquizar	