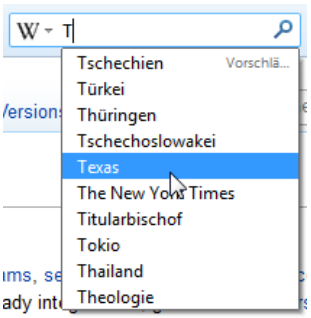
EDA (ETS de Ingeniería Informárica). Curso 2020-2021

Práctica 4. Implementación de la función *Autocompletar texto* usando un ABB equilibrado (1 sesión)

Departamento de Sistemas Informaticos y Computación. Universitat Politècnica de València

1. 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 de forma repetida 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 dependen de la topología del mismo. Sólo un ABB equilibrado permite realizar estas operaciones en tiempos logarítmicos con la talla del ABB.

Así, los objetivos de esta práctica son los siguientes:

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

2. Implementación eficiente de un `ABB`

Como se ha comentado, la implementación que se haga de la clase ABB debe asegurar que los ABB que se construyan tengan alturas casi logarítmicas. Dado que la topología del ABB depende del orden en el que se añaden sus datos, la solución que se propone para conseguir alturas óptimas se basa en añadir los datos en el orden apropiado.

Actividad #1: El paquete `jerarquicos`

El alumno debe crear el subpaquete *librerias.estructurasDeDatos.jerarquicos* y añadir las clases `ABB`, `NodoABB` y `GUIAbb`, disponibles en PoliformaT.

Obsérvese que el método de recorrido por niveles de la clase `ABB` utiliza una cola para almacenar los elementos durante su ejecución. Por ello, es necesario incluir y compilar la interfaz `Cola` en el paquete *librerias/estructurasDeDatos/modelos* y su implementación `ArrayCola` en *librerias/estructurasDeDatos/lineales*.

Actividad #2: Implementación de los métodos para construir el ABB equilibrado

Como se ha comentado, la forma que tiene un ABB depende del orden de inserción de los elementos en él. 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:

- Colocar los datos que se van a añadir al ABB en un array `v` en orden no decreciente.
- Recursivamente, hasta que no queden datos por tratar:
  - Inserta `v[mitad]`, la mediana, como nodo raíz del árbol.
  - Construye su hijo izquierdo del mismo modo a partir de los datos situados en el array a la izquierda de `mitad`.
  - Construye su hijo derecho del mismo modo a partir de los datos situados en el array a la derecha de `mitad`.

A partir de esta idea, se deben 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`:

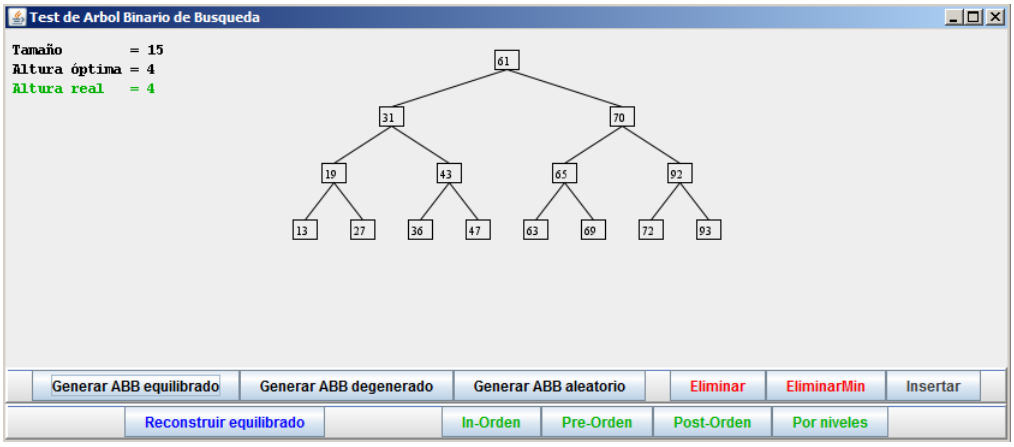
```
public ABB(E[] v) { ... }
```

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

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

Actividad #3: Validación de la clase `ABB`

Ejecuta el programa `GUIAbb` para comprobar el funcionamiento de los métodos diseñados. En concreto, usa los botones `Generar ABB equlibrado` 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 equlibrado`. 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).

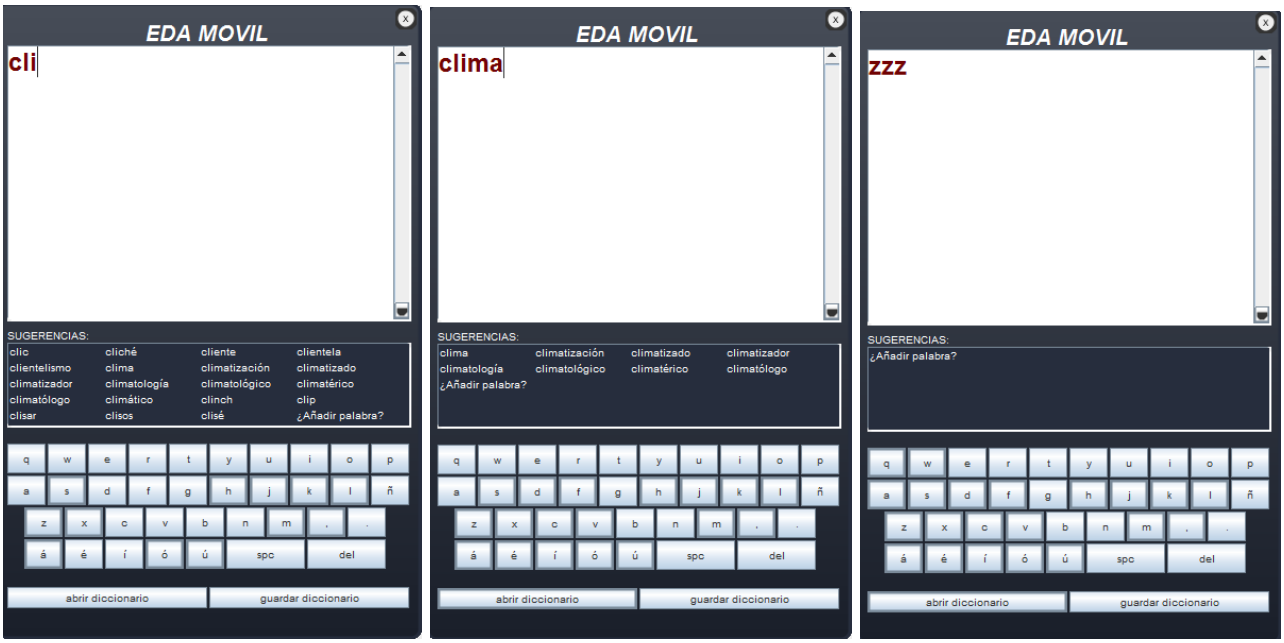


3. Implementación de la función *Autocompletar palabras*

La función de autocompletado de palabras debe ofrecer una serie de sugerencias para completar la palabra conforme el usuario la va escribiendo, sin que tenga necesariamente que terminar de escribirla. Para implementar esta funcionalidad se ha diseñado la clase `EditorPredictivo` como un `ABB<String>`; en concreto, tendrá 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) { ... }
}
```

Esta clase será la utilizada por la interfaz gráfica `GUIEditorPredictivo`; en la figura siguiente se muestra la ejecución 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 y clip; en el segundo: clima, climatización, climatizado, climatizador, climatología, climatológico, climatérico y climatólogo; en el último la lista vacía porque no hay ningúna palabra que tenga como prefijo zzz. Esta interfaz permite también, abrir un diccionario diferente o añadir nuevas palabras al diccionario y guardarlo.



Actividad #4: El paquete `editorPredictivo`

Crear el paquete *aplicaciones/editorPredictivo* y añadir las clases `EditorPredictivo` y `GUIEditorPredictivo` en dicho paquete. Así mismo, el alumno deberá copiar el fichero `castellano.txt`, disponible también en PoliformaT, en el directorio asociado al paquete *aplicaciones/editorPredictivo*; este fichero contiene más de 460000 palabras en castellano ordenadas alfabéticamente (los datos con los que, por defecto, se construirá un Editor Predictivo).

Nótese que el segundo método constructor de la clase `EditorPredictivo`, dado un fichero de palabras ordenado ascendentemente, crea un `ABB` equilibrado de `String` haciendo uso del método *construirEquilibrado* para insertar las palabras del array, obteniendo así un ABB equilibrado.

Actividad #5: 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 devuelve una Lista Con Punto de Interés con los `n` primeros sucesores de un `prefijo` dado; recuérdese que esta lista debe incluir a `prefijo` como primer elemento de la lista siempre y cuando este ya figure en el ABB (es decir, que el prefijo sea ya una palabra completa). Para implementar este método se puede hacer uso del 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 se hayan añadido n sucesores.

Actividad #6: Prueba del Editor Predictivo

Comprueba que el código que has escrito utilizando la interfaz gráfica `GUIEditorPredictivo`. Además de probar con 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	