EDA (ETS de Ingeniería Informática)). Academic Year 2020-2021
Lab 4. Implementing a Predictive (Text) Editor with a Balanced Binary Search Tree (BST)

Departamento de Sistemas Informáticos y Computación. Universitat Politècnica de València

1 Objectives

A Balanced BST-based Representation of a collection of words makes it possible to obtain, among other things, the alphabetically sorted list of the collection's words that begin with a given prefix. Since this is precisely the main functionality of a Predictive Editor, the Autocomplete or Word Completion feature, after this lab the student should be able ...

- To efficiently build a Balanced BST from a set of data, as well as to re-balance an already existing BST when it gets unbalanced after a series of non-random insertions and/or deletions.
- To implement a Predictive Editor by using a Balanced BST.

2 Problem description

To understand how to efficiently implement a Predictive Editor with a Balanced BST, first we need to put together three basic facts.

a. The definition of a Predictive Editor: it is a text editor that helps you type a lot faster by predicting what you want to type based on the prefix you have already typed and prompting you with the best matching guesses (Autocomplete feature). Nowadays, most mobile input technologies, such as the modern (post-T9) predictive keyboards on mobile devices, provide this feature.

Figure 1: Example of a Predictive Editor

b. The relationship between the AutoComplete Feature and the successor of a given element in a collection of words: formally speaking, the alphabetically sorted list of guesses (SUGERENCIAS) that the Predictive Editor offers for a given typed prefix `p` (as estro in Fig.1) is the list of those successors of `p` that begin with prefix `p` (as estro, estrofa, estroncio, estropajo, estropajoso, estropeado, estropear, estropicio in Fig.1).

c. The relationship between the cost of the successor operation on a BST and the degree of balance of the latter: searching for the successor of an element in a Balanced BST of size `N` takes time log `N` on average.

Therefore, putting two and two together, it comes straightforwardly that a Predictive Editor **IS A** Balanced BST of type `String` that implements the Autocomplete feature by retrieving the following successors of a given prefix `p` until it finds one that no longer begins with `p`.

3 Lab activities

Before doing the activities described below in this section, it is necessary that the student updates the structure of packages and files of his *BlueJ eda* project by carrying out the following steps:

- Start *BlueJ* and open the package *aplicaciones* of your *eda* project.
- Within *aplicaciones* package, create a new package called *editorPredictivo*; it will contain the classes that implement the application's Predictive Editor.
- ***Quit*** *BlueJ*.
- Download the classes available in *PoliformaT* into their corresponding folders as shown in the figure below (Figure 2).

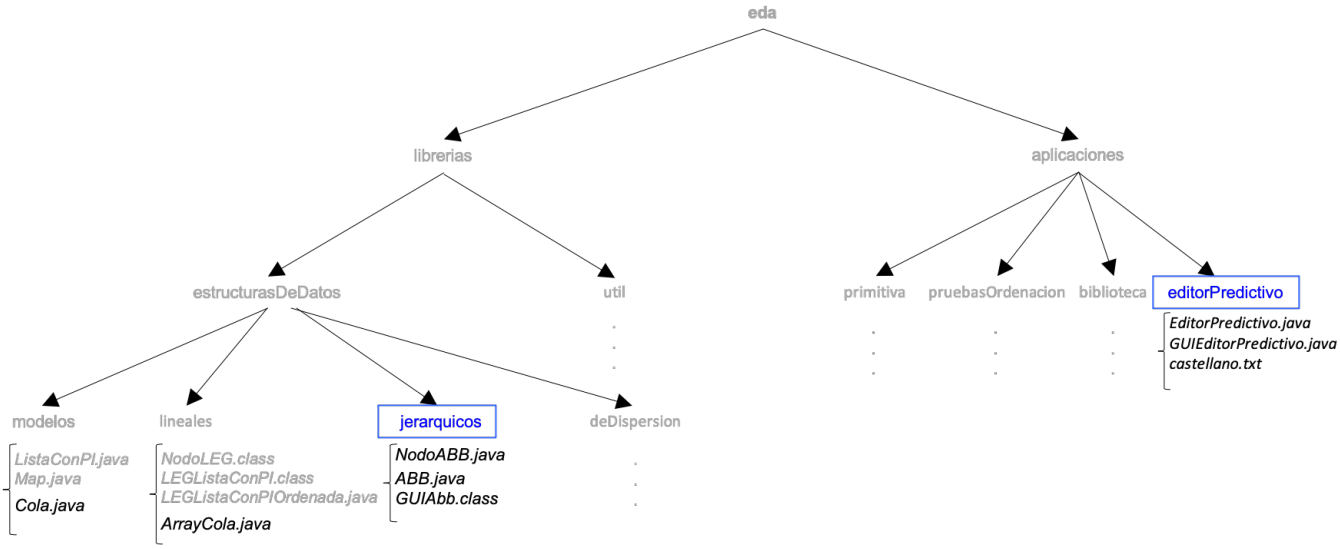

Figure 2: Updated student's *eda* project

NOTE:

- The classes `ABB`, `NodoABB` and `GUIAbb` contain the necessary code to implement and test a BST.
- The class `ABB` imports both the `Cola` interface and its array-based implementation `ArrayCola` in order to implement the Level-order traversal of a BST (method `toStringPorNiveles`).
- The class `EditorPredictivo` IS A **Balanced** `ABB` of `String`s that represents the application's Predictive Editor -its one parameter constructor creates a **Balanced** BST of `String`s with the elements of a given `fileName` by using the method `construirEquilibrado` of class `ABB`.
- The class `GUIEditorPredictivo` is a test program that uses an `EditorPredictivo` to allow the user write words and complete the current word he is typing with its suggestions.
- The file `castellano.txt` contains a large set of alphabetically sorted words in Spanish (more than 460000 words), the default data set the `ABB` of `String`s of the application will be built with.

- (Re)Start *BlueJ* and open your *eda* project. After double-clicking on the icons of the packages *librerias* and *estructurasDeDatos*, open the package *modelos*.
- Within *modelos* package, compile the class `Cola`. ***Close*** the *modelos* package.
- Open the package *lineales* and compile the class `ArrayCola`. ***Close*** the *lineales* package.
- ***Quit*** *BlueJ*.
- (Re)Start *BlueJ* and open your *eda* project. After double-clicking on the icons of the packages *librerias* and *estructurasDeDatos*, open the *jerarquicos* package and complete the code of `ABB` class by doing the first two activities of this lab.

3.1 Building a Balanced (Minimum-Height) BST

As the student already knows, only a Balanced BST of size `N` offers log `N` asymptotic time in average and worst cases for inserts, deletes and searches. However, and this is the disadvantage of a BST, the asymptotic running time of these same operations and cases is increased up to linear time (`N`) in a Degenerated BST, i.e. in a BST whose elements are inserted in order or in near-order.

Therefore, in order to exploit all the advantages that a BST offers, the challenge we are faced with is ensuring that the topology of the resulting BST exhibits an optimal ratio of height to the number of nodes. Because the topology of a BST is based upon the order in which the elements are added to it, you might opt to solve this problem by dictating the "correct" order in which the elements are added to the BST. More specifically, we can proceed as follows:

a. Fill an array `v` with the elements to be inserted sorted in nondecreasing order.

b. Recursively, until the array `v` is empty ...

   1.- Put `v[middle]`, the data median, at the Root node of the tree to be built.

   2.- Build its Left subtree by recursively applying point b.1 to the `v[0, middle - 1]` elements, until the subarray is empty.

   3.- Build its Right subtree by recursively applying point b.1 to the `v[middle + 1, v.length -1]` elements, until the subarray is empty.

Once it is understood how to build a Balanced BST, the student should complete the following methods of the `ABB` class:

- Method `construirEquilibrado` that, given a subarray `v[ini, fin]` sorted in nondecreasing order, returns the Root node of a Balanced BST with the elements of `v[ini, fin]`.
- The constructor method `ABB(E[]v)` that creates a Balanced BST with the elements of `v` by using the `construirEquilibrado` method.
- Method `reconstruirEquilibrado` that re-balances `this` BST by using both the `construirEquilibrado` and the `toArrayInOrden` methods.

3.2 Testing the `ABB` class

The student should run the `GUIAbb` Graphical User Interface to test the main methods of the `ABB` class in the following cases:

- Balanced BST (`Generar ABB equilibrado` button).
- Degenerate BST (`Generar ABB degenerado` button).
- Random BST (`Generar ABB aleatorio` button).

Specifically, the test of the `reconstruirEquilibrado` method has to be done by clicking the following two sequences of buttons: `Generar ABB degenerado` and `Reconstruir equilibrado`; `Generar ABB aleatorio` and `Reconstruir equilibrado`.

3.3 Completing the `EditorPredictivo` class

In this activity, the student should complete the `recuperarSucesores` method of the `EditorPredictivo` class. This method returns a `ListaConPI<String>` whose elements are the first `n` successors of a given prefix `prefijo`, by using the method `sucesor` of the `ABB` class as follows:

i. Search for `prefijo` in the `ABB` class, since `prefijo` can be already a word of the BST.

ii. Retrieve its following successors until you find one that no longer begins with `prefijo`.

Note that when `prefijo` is a word of the Predictive Editor then it should be the first element of the List returned by `recuperarSucesores`.

3.4 Testing the `recuperarSucesores` method

The student must test the correctness of the `recuperarSucesores` method by running the `GUIEditorPredictivo` program on the prefix in Figure 1, as well as on the sample prefixes of the table below, and checking that he obtains the same suggestions that those showed in this document.

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

Table 1: Test cases and the corresponding suggestions.