Jhon Stiven Arboleda, Juan Sebastián Barrera, Alejandro García

**PHASE 1:**

1. **Identification of the problem**

We want to develop a software prototype that allows you to manage CRUD operations in a database on the American continent.

**Needed:**

* The simulation of creating a close number of records of the population of the American continent for 2020 (around one billion people).

**Problem:** Efficient database storage, each record must be editable and removable.

1. Research

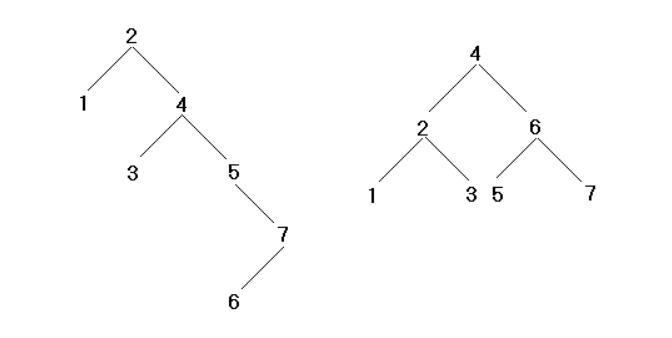
Functional requirements:

1. Add a new user to the database. The following data must be taken into account: code, name, surname, sex, date of birth, height, nationality and photograph.
2. Update a person's record in the database. All user information must be editable, except for the code that is automatically generated.
3. Delete a person's record from the database.
4. Search for a person in the database by any criteria name, surname, full name (name + surname) and code.

**BST:** A binary search tree satisfies the condition that the left subtree of any node (if it is not empty) contains smaller values ​​than the one that contains that node, and the right subtree (if it is not empty) contains larger values.

For these definitions, it is considered that there is an established order relationship between the elements of the nodes. Whether a certain relationship is defined or not depends on each programming language. From this it follows that there can be different binary search trees for the same set of elements.

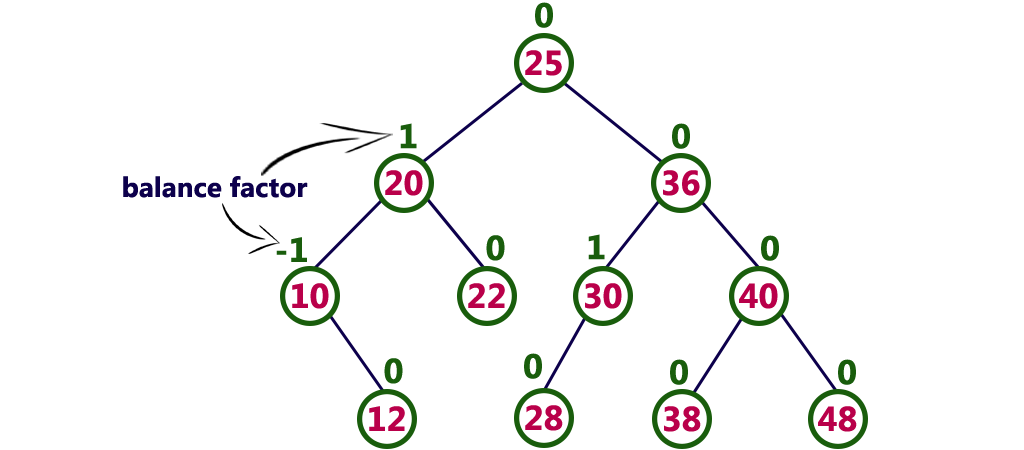
The figure shows 2 BSTs built on the same set of integers:



**BST example:**

**AVL:** The AVL tree takes its name from the initials of the surnames of its inventors, Georgii Adelson-Velskii and Yevgeniy Landis. They made it known in the publication of an article in 1962, "An algorithm for the organization of information" ("An algorithm for the organization of information").

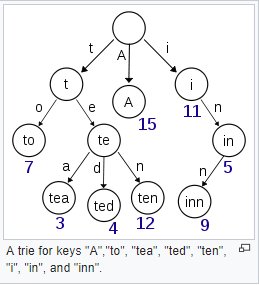
AVL trees are always balanced in such a way that for all nodes, the height of the left branch does not differ by more than one unit from the height of the right branch or vice versa. Thanks to this form of balance (or balancing), the complexity of a search in one of these trees is always kept in order of complexity O (log n). The balance factor can be stored directly at each node or computed from the heights of the subtrees.



**AVL example:**

**Trie:** A "trie" is a tree-like data structure that allows the retrieval of information (hence its name from reTRIEval). The information stored in a trie is a set of keys, where a key is a sequence of symbols belonging to an alphabet. The keys are stored in the leaves of the tree and the internal nodes are gateways to guide the search. The tree is structured in such a way that each letter of the key is placed in a node so that the children of a node represent the different possibilities of different symbols that can continue to the symbol represented by the parent node.

**Search in the Trie:** It begins at the root of the tree. If the symbol, we are looking for is A then the search continues in the subtree associated with the symbol A hanging from the root. It continues in the same way until reaching the leaf node. Then the string associated with the leaf node is compared and if it matches the search string then the search has ended in success, if not then the element is not found in the tree.



**Trie example:**

**REFERENCES:**

**BST definition:**

<https://es.wikipedia.org/wiki/Árbol_binario_de_búsqueda>

**Tree AVL definition.**

<https://sites.google.com/a/espe.edu.ec/programacion-ii/home/arboles/arboles-avl>

**Trie definition.**

<https://es.wikipedia.org/wiki/Trie>

**PHASE 2:**

theoretical framework: TODO

**PHASE 3:**

**Brainstorming:**

**TODO**

Alternative 1:

Alternative 2:

Alternative 3:

Alternative 4:

Alternative 5:

Alternative 6:

**PHASE 4:** DESCARTAR IDEAS. DEJAR 3. AGREGAR INFO.

**TODO**

**PHASE 5:**

**Evaluation and selection of the best solution**

**TODO**

**Criterion A:**

* **[1 points]**
* **[2 points]**
* **[3 points]**

**Criterion B:**

* **[1 points]**
* **[2 points]**
* **[3 points]**

**Criterion C:**

* **[1 points]**
* **[2 points]**
* **[3 points]**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SOLUTION** | **Criterion A** | **Criterion B** | **Criterion C** | **Total points** |
| Alternative 1 |  |  |  |  |
| Alternative 2 |  |  |  |  |
| Alternative 3 |  |  |  |  |