ENGINEERING METHOD

MEMBERS:

Gabriel Suárez - **A00368589 SIS**

Alejandro Varela - **A00369019 TEL & SIS**

Luis Alfonso Murcia Hernández - **A00369008 TEL**

TEACHER:

Anibal Sosa Aguirre

ALGORITHMS AND DATA STRUCTURES

ICESI UNIVERSITY
2021-2

Problematic Context: In the sports field, a large company realized that a tool is needed to manage large information that allows data to be entered, either in a massive way that allows finding, modifying, and eliminating data of the players who are legends have returned throughout the history of basketball, with some parameters that will be seen later.

Solution development: We decided to rely on the engineering method to solve this problem. By searching various sources, it was possible to realize that 7 steps are needed for an efficient problem solving.

• **Phase 1: Identification of the Problem**: Although it may not seem like it, it is the most important phase of the engineering method, a good problem must be defined to know what they are asking for and based on this, begin to develop creative, effective solutions and, above all, innovative.

• Phase 2: Gathering the Necessary Information: You need to know the number of stores that offer the same service, the feasibility of incorporating the store into the market and the location.

• **Phase 3: Search for Creative Solutions**: Allows brainstorming

• **Phase 4: Transition from the Formulation of Ideas to the Preliminary Designs**: Collaborates in the discarding of ideas in the previous step and deepening of the most viable ones.

• **Phase 5: Evaluation and Selection of the Best Solution**: After an exhaustive analysis, the true solution is chosen.

• **Phase 6: Preparation of reports and specifications**: The most important aspects of the project are documented.

• **Phase 7: Design Implementation**: We proceed to the experimental stage to carry out the idea of the 6 previous steps.

In order to be precise in the process of developing this method, each step will be clearly explained:

Phase 1: Problem Identification

Given the recent avalanche of numbers, FIBA has decided to take advantage of this situation and consolidate in an application, the most relevant data of each of the basketball professionals on the planet, so that different queries can be made that allow analysis of these data, patterns are known about the development of the sport, the criteria that take more force or, in general, where the sport is currently heading.

Based on this, some questions can be raised. Is it possible to find a solution that covers all the points of this problem? Is there a fast, safe and efficient way to save some important histories?

For this reason, it is necessary to test the knowledge and implement a tool that is capable of collecting all the necessary information, modifying it and even eliminating it in case it becomes unnecessary.

After raising the problem, it would be pertinent to have a series of **requirements** to allow greater ease and efficiency in the application.

- Allow data entry, either in bulk with .csv files or through an interface
- Delete data in case it is already unnecessary in any way within the data collection tool
- Modify data in case it has been exceeded in any way or an error was made at the time of entry.
- Make player queries using the statistical categories included as search criteria, find those players who have scored 10 points per game, or those with more than 20 rebounds per game.
- Include the data per player of the following items: name, age, team and 5 statistics points per game, rebounds per game, assists per game, steals per game, blocks per game
- Access efficiently for the performance of the application due to the large amount of data that the application must store.

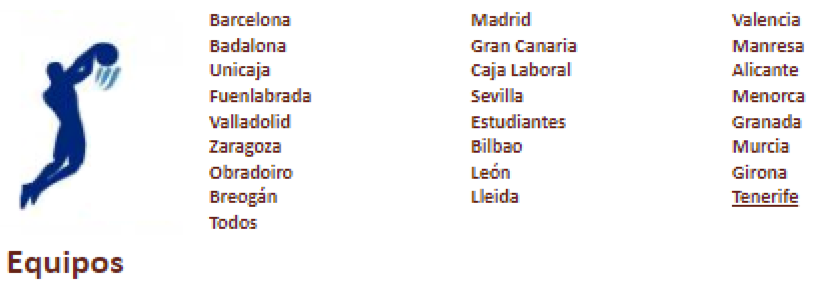
- Retrieve players according to the selected search category and the value given for it, it must be considered that queries are not necessarily requested where the attribute satisfies an equality. The search criteria can be any of the statistical attributes, but for four of them the search must be very fast
- Show the time it takes to make a query.
- Allow customer to search on two statistical criteria using ABB as structure for index management
- Execute player queries according to all criteria, but only four of them (on statistical attributes) should be efficient.

Phase 2: Gathering the necessary information

To begin with the resolution of this problem, we decided to look for some applications that fulfill a similar operation to a database that we are implementing, with which we found a web page http://www.bdbasket.com/es/index.html of Historical records of the teams, players and coaches of Spain in certain seasons from 1990 to the present.

Temporadas

2018-19	2008-09	1998-99
2017-18	2007-08	1997-98
2016-17	2006-07	1996-97
2015-16	2005-06	1995-96
2014-15	2004-05	1994-95
2013-14	2003-04	1993-94
2012-13	2002-03	1992-93
2011-12	2001-02	1991-92
2010-11	2000-01	1990-91
2009-10	1999-00	



Equipos

Barcelona	Madrid	Valencia
Badalona	Gran Canaria	Manresa
Unicaja	Caja Laboral	Alicante
Fuenlabrada	Sevilla	Menorca
Valladolid	Estudiantes	Granada
Zaragoza	Bilbao	Murcia
Obradoiro	León	Girona
Breogán	Lleida	Tenerife
Todos		

From which some ideas for the design of the graphical interface can be drawn, but in its operation, there is a great variation because it does not solve the same problem that we decided to solve.

GitHub: Which is a collaborative software development platform to host projects using the Git version control system. It hosts a code repository and provides you with very useful tools for teamwork within a project. Besides that, you can contribute to improve the software of others. A very important tool to work with all the people who will support in the creation of the project. Besides that, it provides us with certain useful features, such as:

- A wiki for maintaining the different versions of the pages.
- An issue tracking system that allows your team members to detail a problem with your software or a suggestion they want to make.
- A code review tool, where you can add annotations at any point in a file and discuss certain changes made in a specific commit.
- A branch viewer where you can compare the progress made in the different branches of our repository.
- Among the concepts that can be found in the implementation phase, we have some such as:

TAD: An abstract data type (hereinafter TDA) is a set of data or objects to which operations are associated. The TDA provides an interface with which it is possible to perform the allowed operations, abstracting from the way in which these operations are implemented. This means that the same ADT can be implemented using different data structures and provide the same functionality.

N-ary trees: An n-ary tree is a recursive structure in which each element can have any number of associated n-ary subtrees and where the order of the subtrees is not important. It is not necessary to know which sub-tree is the first or the last, but simply to know that it is a sub-tree. You also need certain concepts such as:

- **Node**: Tree element
- **Root**: Initial node of the tree
- **Leaf**: Node without children
- **Path**: Nodes between two elements including them
- **Branch**: Path between the root and a leaf
- **Height**: Number of nodes in the longest branch
- **Weight**: Number of nodes in the tree

Red-Black Tree: A red-black tree is a binary search-balanced tree, a data structure used in computer science and computer science. The original structure was created by Rudolf Bayer in 1972, who gave it the name "symmetric binary B-trees", but it took its modern name in a 1978 work by Leo J. Guibas and Robert Sedgewick. It is complex but has a good worst case runtime for your trades and efficient in practice. You can find, insert, and delete in O time (log n), where n is the number of items in the tree.

Balanced Tree: A binary tree is considered balanced when all its levels, except the last one, are integrated to the maximum capacity of nodes. There are different proposals to balance the trees and each one of them affects the efficiency of the insertion and elimination operations of the nodes. The most common is the AVL tree technique.

AVL TREES

It is a binary search tree that tries to keep it as balanced as possible, as insert and delete operations are performed. They were proposed in 1962 by the Russian mathematicians Adelson, Velskii and Landis, from which their name comes.

In AVL trees, the fact that for any node of the tree, the difference between the heights of its subtrees does not exceed one unit must be fulfilled.

Balancing Factor

The nodes of an AVL tree keep a value of -1, 0, 1, which is known as the Balancing Factor (FB) and represents the height between the heights of their subtrees.

A zero FB of a node means that the heights of its subtrees are equal.

A positive FB means that the height of its right subtree is greater than the left subtree. A negative FB means that the height of its left subtree is greater than the right subtree.

Phase 3: Search for creative solutions

In this phase, what is known as “brainstorming” is very important and, as silly as it sounds, they are very useful when looking for the best solution. For that reason, we decided to **brainstorm ideas** that were not so elaborate and thought out in just **5 minutes**.

Alejandro Varela:

Idea 1: Create a list array of all the data to be entered.

Idea 2: Host the data in a hash table, in which a line of code would be written that would be in charge of searching for the required values within it.

Idea 3: Balanced binary search trees for quick access to player data

Gabriel Suarez:

Idea 1: An array of numbers, in which there was an administrator pending to change the values as more data was entered.

Idea 2: An n-ary tree that would be able to accommodate the values at each node.

Luis Murcia:

Idea 1: Binary tree, where each value entered within a node is entered.

Idea 2: A red-black tree that made searching for items much easie

Phase 4: Transition from Idea Formulation to Draft Designs

We need to find the best solution to solve the problem we are developing, for this reason it is necessary to review each of the proposals of all the members who contributed their grain of sand.
We started by analyzing each of the solutions that were proposed, and in most of them they were right about one thing, it is necessary to use a data structure that allows each value that is entered to be accommodated, and it is possible to modify or eliminate it.
Although they tried to propose data structures, there were several ideas that did not meet what was necessary to move to the next phase. **Ideas one and two** of Alejandro can be easily **discarded** because when saving elements in a hash table, collisions can exist which negatively influence the efficiency of the program, and to access the position in a list array is linear, and it would be a very slow process for a large amount of data. On the other hand, Gabriel's idea one is **discarded**, because to access the values they must also be done in linear time. Finally, Luis's **idea one** does not completely solve the problem and **is not considered**, since a binary tree can only have two children, and to add the massive amount of data, it would be a much larger tree, thus affecting its efficiency. .

In this way we are left with **three (3)** possible ideas of reformulation to enter the next phase:

Phase 5: Evaluation and selection of the best solution

Alejandro Varela (Idea 1):

• Balanced search tree.

It is very interesting to talk about these trees, as it allows the search to not be a problem when using the application.

Gabriel Suárez (Idea 2):

• N-ary tree.

The idea of a tree that is capable of having other trees within itself, allows the search within it to be more efficient.

Luis Murcia (Idea 3):

• Red-black tree.

Black, red trees are very efficient when it comes to searching for items, just what the app needs to function properly.

Reformulated Idea (Idea 4):

As in n-ary trees it is only possible to delete leaves, it was decided only to take the idea of balanced search trees, supported by the red-black tree method for a more efficient application.

Evaluation criteria:

Criterion 1: Application search time

- [2] Fast
- [1] Slow
- [0] Does not apply

Criterion 2: Space in the application system

- [2] Little space
- [1] Lots of space

Criterion 3: Ease of use of the application

- [3] Easy
- [2] Normal
- [1] Difficult

	Criterion 1	Criterion 2	Criterion 3	Total
Idea 1	2	0	2	4
Idea 2	0	2	2	4
Idea 3	2	0	2	4
Idea 4	2	2	3	7

Thanks to these evaluation criteria, we agree that the best solution to carry out is 4, and it makes sense, because it used good ideas from the previous 3.

Phase 6: Preparation of reports and specifications

You need to analyze the documents in the doc’s folder. of this same project.

Phase 7: Design implementation

Our project design implementation is mounted on GitHub with a link to the repository.