UNIVERSIDAD ICESI

Tarea Integradora II

Facultad de Ingeniería

Algoritmos y Estructuras de Datos

I Periodo de 2021

Carolina Pasuy Pinilla A00358427 , Juan Manuel Palta ,

Anderson Cardenas

Programa de Ingeniería de Sistemas, Facultad de Ingeniería, Universidad Icesi, Cali Colombia

Functional Requeriments

R1. Enter data, either in bulk or through an interface

R2. Delete or modify data

R3. Make player queries using the statistical categories included as search criteria

R4. Save the most relevant data of each of the basketball professionals on the planet like the following items: name, age, team, and 5 statistical items points per game, rebounds per game, assists per game, steals per game, blocks per game.

R5. Retrieve players according to the selected search category and the value given for it.

R6. Millions of data can be stored.

R7. The importance of ensuring rapid access to data should be highlighted

R8. Should show the time it takes to make a query

R9. Will allow the customer to search on two statistical criteria using ABB as the structure for index management

Non-functional requirements

R1. Efficient search for categories of 4 statistical items, through indices.

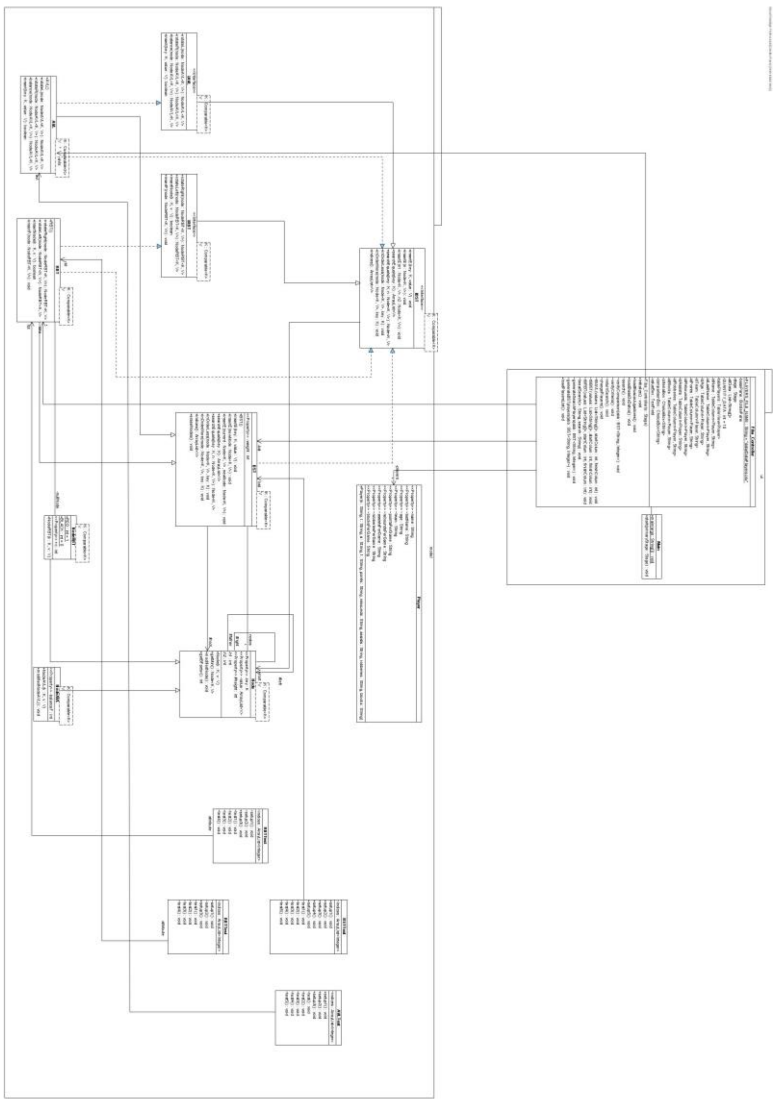
R2. Two types of query must be with AVL trees and the other two with treesred and black. Where each tree will save the value of the index (attribute) and the position ofthis data on the disk.

R3. There must be two queries for unbalanced binary trees. This meansthat one query should be performed by AVL trees and by ABB and the other by trees red and black and ABB.

R4. The complexity cannot be linear, only in one search

R5. Handling large software: The program must contain at least 200,000 datavalid on players.

Class Diagram



Test Design

Setup BSTTest

Name	Class	Stage

Setup1	BST	bst = new BST<>()
		indices=new ArrayList <Integer>()

Name	Class	Stage

Setup2	BST	bst = new BST<>()
		bst.insertE(20, 2)
		bst.insertE(15, 3)
		bst.insertE(22, 1)

Name	Class	Stage

Setup3	BST	Setup1()
		bst.insertE(20, 1)
		bst.insertE(20, 2)
		bst.insertE(20, 3)
		bst.insertE(20, 4)
		bst.insertE(20, 5)

Name	Class	Stage

Setup4	BST	Setup1()
		bst.insertE(20, 1)
		bst.insertE(20, 2)
		bst.insertE(14, 3)
		bst.insertE(16, 4)
		bst.insertE(12, 5)

Name	Class	Stage

Setup5	BST	bst = new BST<>()
		bst.insertE(6, 6)
		bst.insertE(5, 5)
		bst.insertE(4, 4)
		bst.insertE(3, 3)
		bst.insertE(2, 2)
		bst.insertE(1, 1)

Setup AVLTest

Name	Class	Stage
Setup1	AVL	avl = new AVL<>()

Name	Class	Stage

Setup2	AVL	avl = new AVL<>()
		avl.insert(9,6)
		avl.insert(10,7)
		avl.insert(5, 2)
		avl.insert(7, 4)
		avl.insert(8, 5)

		avl.insert(6, 3)
		avl.insert(1, 1)

Name	Class	Stage

Setup2	AVL	avl = new AVL<>()
		avl.insert(20, 2)
		avl.insert(15, 3)
		avl.insert(22, 1)

Setup RBTTest

Name	Class	Stage
Setup1	RBT	rbt = new RBT<>()

Name	Class	Stage

Setup2	RBT	rbt = new RBT<>()
		rbt.insertNode(26,1)
		rbt.insertNode(17,2)
		rbt.insertNode(41,3)
		rbt.insertNode(14,4)
		rbt.insertNode(21,5)
		rbt.insertNode(30,6)
		rbt.insertNode(47,7)
		rbt.insertNode(10,8)

Name	Class	Stage

Setup3	RBT	setup1()
		rbt.insertNode(20, 1)
		rbt.insertNode(20, 2)
		rbt.insertNode(20, 3)
		rbt.insertNode(20, 4)
		rbt.insertNode(20, 5)

Test BST

Objt: the objective of this test is to verify that node bst has been created and inserted			
Class	Method	Stage	Result
BST	insertE	Setup1	The node has been inserted

Objt:verify that found the nodes in the positions			
Class	Method	Stage	Result

BST	inOrden(less-more),getRoot, getLeft, getRight	Setup2	Verify the root and his son exist and his positions. Also take the inidices in the array from the inOrden method
Objt:verify that found the nodes in the positions with other stage			
Class	Method	Stage	Result
BST	inOrden(less-more),getRoot, getLeft, getRight	Setup3	Verify the root and his son exist and his positions. Also take the inidices in the array from the inOrden method with other stage

Objt:verify thath bst and his nodes exists and indices			
Class	Method	Stage	Result
BST	searchEquals, nodes to indices	Setup4	The nodes value are converted and verify the indices exist

Objt:verify the positions in the bst are correct			
Class	Method	Stage	Result
BST	searchEquals, getRoot	Setup5	The nodes are in the position correct

Test AVL

Objt: the objective of this test is to verify that avl are null			
Class	Method	Stage	Result
AVL	constructor	Setup1	The avl are null

Objt:verify thath avll and his nodes exist and are inserted			
Class	Method	Stage	Result
AVL	Constructor, insert	Setup2	The nodes have been created and inserted
Objt:verify thath avl and his nodes are inserted and in a correct position			
Class	Method	Stage	Result
AVL	Constructor, insert, getRoot,getLeft,GetRight	Setup2	The nodes has been created and inserted and in a correct position

Objt: the objective of this test is to verify the avl indices are in the correct position			
Class	Method	Stage	Result
AVL	Constructor, índices	Setup3	The avl insert the values in the indices

Objt: the objective of this test is to verify the method search in the avl			
Class	Method	Stage	Result
AVL	Constructor, searchEquals	Setup3	The avl found the node

Test RBT

Objt: the objective of this test is to verify that avl insert a node			
Class	Method	Stage	Result
RBT	Constructor, insertNode	Setup1	The avl are not null

Objt:verify the positions in the rbt are correct			
Class	Method	Stage	Result
RBT	getRight, getRoot, getLeft	Setup2	The nodes are in the position correct

Objt: the objective of this test is to verify the rbt indices are in the correct position			
Class	Method	Stage	Result
RBT	Constructor, índices, insertNode	Setup1	The rbt insert the values in the indices

Objt: the objective of this test is to verify the method search in the rbt and verify the positions in the indices			
Class	Method	Stage	Result
RBT	Constructor, searchEquals	Setup3	The avl found the node, and are equals of values

Engineering Method

1.Identification of the problem:

FIBA is the regulatory body for basketball worldwide, given the recent avalanche of numbers of figures from the games has decided that the most relevant data of each of the professionals in the basketball. Patterns about the development of sport and more criteria to get an idea of where the sports is currently heading.

2.The collection of the necessary information:

Binary Search Tree (BST): Is based on the property that keys that are less than the parent are in the left subtree, and keys that are greater than the parent are in the right subtree.

Transversal In-order: The traversal in In-order consists of traversing the left subtree in In-order, then the data of the root node is examined, and finally the right subtree in In-order is registered.

Source:

http://profesores.elo.utfsm.cl/~agv/elo320/01and02/dataStructures/binarySearchTree.pdf

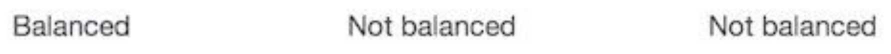
Self-Balancing Binary Search Tree (AVL) : Also referred to as a height-balanced binary tree, is defined as a binary tree in which the height of the left and right subtree of any node differ by not more than 1.

Following are the conditions for a height-balanced binary tree:

1. Difference between the left and the right subtree for any node is not more than one
2. The left subtree is balance
3. The right subtree is balanced

Source:

https://www.programiz.com/dsa/balanced-binary-tree



AVL Rotations:

Left Rotation: If a tree becomes unbalanced, when a node is inserted into the right subtree of the right subtree, then we perform a single left rotation .

Right Rotation: AVL tree may become unbalanced, if a node is inserted in the left subtree of the left subtree. The tree then needs a right rotation.

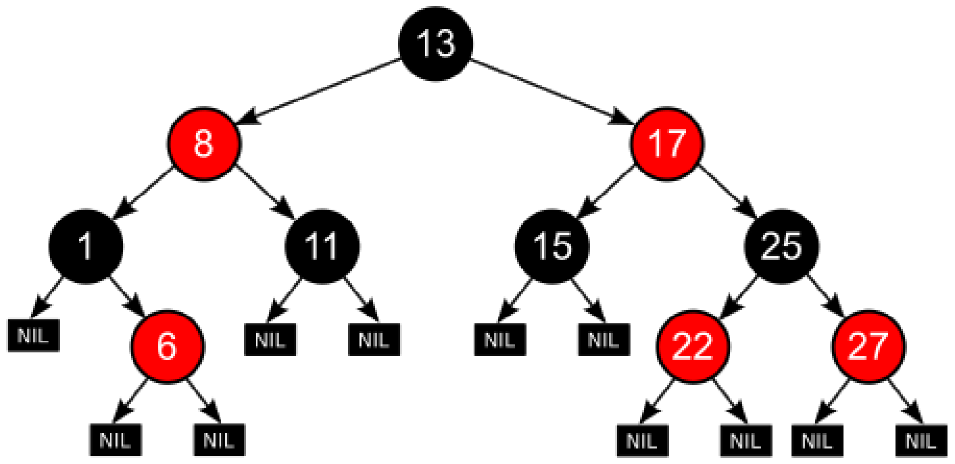
Left-Right Rotation: Double rotations are slightly complex version of already explained versions of rotations. To understand them better, we should take note of each action performed while rotation. Let's first check how to perform Left-Right rotation. A left-right rotation is a combination of left rotation followed by right rotation.

Right- Left Rotation: The second type of double rotation is Right-Left Rotation. It is a combination of right rotation followed by left rotation.

Source:

https://www.tutorialspoint.com/data_structures_algorithms/avl_tree_algorithm.htm

Red-Black Tree: Is a kind of self-balancing binary search tree where each node has an extra bit, and that bit is often interpreted as the color (red or black). These colors are used to ensure that the tree remains balanced during insertions and deletions. Although the balance of the tree is not perfect, it is good enough to reduce the searching time and maintain it around O(log n) time, where n is the total number of elements in the tree.



Rules That Every Red-Black Tree Follows:

1. Every node has a colour either red or black.
2. The root of the tree is always black.
3. There are no two adjacent red nodes (A red node cannot have a red parent or red child).
4. Every path from a node (including root) to any of its descendants NULL nodes has the same number of black nodes.

Source:

https://www.geeksforgeeks.org/red-black-tree-set-1-introduction-2/

OpenCSV: It is a Java library that allows you to work with CSVs in a very simple way, both to read and to write them.Offer these classes:·

- CSVReader: gives Operations to read the CSV file as a list of String arrays.

- CSVWriter: to write a CSV file·
- CsvToBean: reads the CSV and converts it to a Bean according to a MappingStrategy.

Source:

https://unpocodejava.com/2014/12/31/opencsv-trabajando-con-csv-de-forma-sencilla-con-java/#:~:text=OpenCSV%20(http%3A%2F%2Fopencsv.sourceforge,lista%20de%20arrays%20de%20String.

3.Search of creative solutions:

Get the information:

Defensive +/-	+4.8			●	5 Grant Hill

- We first thought about how we could collect such amount of information through a csv file. To which we thought that the information of the players was separated by a character, we could do this with the bufferedReader and later with this information generate the players. these players can be save in a linked list to later search a player by a criteria like points, robberies,passes, etc.

- Another option that we think is to look for a tool (some library) that allows us to bring the information without so much process whit different methods of the library we can get the information of the csv and then generate the players. We think in implementation of Binary Search Tree, we can use the binary Search tree to save a determinate criteria and make more efficient the search. The node will have the key and value, the key will be the attribute and the value can be the index of the player in the csv or a reference to this player. Also the implementation of AVL, With the Avl is the same idea of the binary search tree, this structure will help with save a criteria and search that criteria and get the search player, the difference is that avl when we insert a new player the whole tree will be ordered. And last the RBT in these trees will be stay only the information that is not repeated in the trees, we can use an array of index to later get the players that share the same characteristic.

- Another option was to enter data into our program through an interface and save them, we can create a User interface and the user of the program will can Save data of the player, Modify Data or Delete his data. We can get the information of a csv file and with this information generate the players, these players can be save in a linked list to later search a player by a criteria like points, robberies,passes, etc. With the LinkedList we can use the binary search. The criteria that we are going to look for are all numeric, so the binary search could be implemented along the linked list and thus find that value, however the values can be repeated so it can be somewhat complex. In the LinkedList will be stay all the information in the chosen data structures with the index of the player in the csv.

- Another option that we think is to look for a tool (some library) that allows us to bring the information without so much process whit different methods of the library we can get the information of the csv and then generate the players. We think in implementation of Binary Search Tree, we can use the binary Search tree to save a determinate criteria and make more efficient the search. The node will have the key and value, the key will be the attribute and the value can be the index of the player in the csv or a reference to this player. Also the implementation of AVL, With the Avl is the same idea of the binary search tree, this structure will help with save a criteria and search that criteria and get the search player, the difference is that avl when we insert a new player the whole tree will be ordered. And last the RBT in this trees will be stay all the players in the csv and insert all the information in the trees with a reference to the player.

- Another option that we think is to look for a tool (some library) that allows us to bring the information without so much process whit different methods of the library we can get the information of the csv and then generate the players. These players can be save in a linked list to later search a player by a criteria like points, robberies,passes, etc. Insert only the information that is not repeated in the LinkedList, we can use an array of index to later get the players that share the same characteristic.

- We first thought about how we could collect such amount of information through a csv file. To which we thought that the information of the players was separated by a character, we could do this with the bufferedReader and later with this information generate the players. These players can be save in different Trees(AVL,BTS,RBT) so we insert all the information in determinate criteria in the trees with a reference to the player.

4.Moving from ideas to preliminary designs:

Criterias:

- **Solution precision:** The solution is viable and meets the effective development of all the requirements of the problem

  1. Imprecise
  2. Almost Accurate
  3. Accurate

- **Effectiveness:** Regardless of the complexity, the solution complies with a perfect effectiveness lacking any logical error

1. No Effective
2. Effective Medium
3. Effective

- **Usability:** The solution can be used because it meets all the proposed objectives

  1. Not usable at all
  2. Complex
  3. Usable

- **Accessible:** The solution presents easy access to the data generated

  1. Not accessible
  2. Moderately accessible
  3. Accessible

Evaluating the previous criteria in the alternatives that remain, we obtain the following table:

	Solution precision	**Effectiveness**	**Usability**	**Accessible**
Alternative 1	2	2	2	3
Alternative 2	2	3	3	3
Alternative 3	2	1	2	3

Alternative 1: 9 points

Alternative 2: 11 points

Alternative 3: 8 points

According to the previous evaluation, Alternative 2 should be selected, since it obtained the highest score according to defined criteria.

5.The evaluation and selection of the preferred solution:

- We choose the alternative 2 because by save the information from a csv file, we are saving the information in secondary memory which will allow the program to work with large amounts of data, doing so with a library like OpenCsv will make data collection easier. We want the searches to be optimal, the most feasible way is to use the different trees and manage them by indexes so as not to create all the players.

Having used other options such as handling it in a LinkedList or entering the data through an interface would take us more time than desired, for this reason the best option is alternative 2.