Module 8 Assignment Part 1: AVL Rotations

Marcar esta página

Coding Assignment fecha límite Oct 2, 2023 08:43 -05

AVL

In this two-part assignment, you will be coding an **AVL**, which is a special type of binary search tree. It must follow the same rules as binary search trees: each node has 0-2 children, all data in the node's left subtree is less than the parent node's data, all data in the node's right subtree is greater than the parent node's data, and every node's data is unique.

However, an AVL differs from a BST with its self-balancing rotations, which you will implement in part one of this two-part assignment.

IMPORTANT:

- **You will be given unlimited attempts on this assignment, with no cooldown between submissions.**
- **Please run your code before each submission to ensure that there are no formatting errors! If there are formatting errors in your code, your code will not be graded and a submission attempt will be logged. For more information, please review the Vocareum overview below.**

AVLNode

An **AVLNode** class is provided to you and will be used to represent the nodes of the tree. This file should be treated as **read-only** and should not be modified in any way. This AVLNode class contains getter and setter methods to access and mutate the structure of the nodes. Please make sure that you understand how this class works, as interaction with this class is crucial for this assignment.

Part 1 - Rotations

In the first part of this assignment, you will be implementing the rotation functionality of an AVL. There are four methods that work in conjunction to ensure that the AVL remains balanced at all times; each of which is outlined below. For this assignment, you will be implementing those four methods. Be sure to read the javadocs for more information!

updateHeightAndBF

Each node contains a **height** and **balanceFactor** variable. The height variable represents the height of the node. If you recall, a node's height is `max(left node's height, right node's height) + 1` where the height of a leaf node is 0 and the heights of its null children are -1. The balance factor of a node should be equal to its left child's height minus its right child's height. Since we are storing this information in each node, you no longer need to recursively compute it.

The first step of this assignment is to implement the updateHeightAndBF method, which will, as the name suggests, update the height and balance factor of the passed in node. There are four main steps to completing this method:

1. Store the left child height in a variable (keep in mind the height of a null node; you'll have to account for this!)
2. Store the right child height in a variable (keep in mind the height of a null node; you'll have to account for this!)
3. Set the height of the node to be: `max(left child's height, right child's height) + 1`
4. Set the balance factor of the node to be: `left child's height - right child's height`

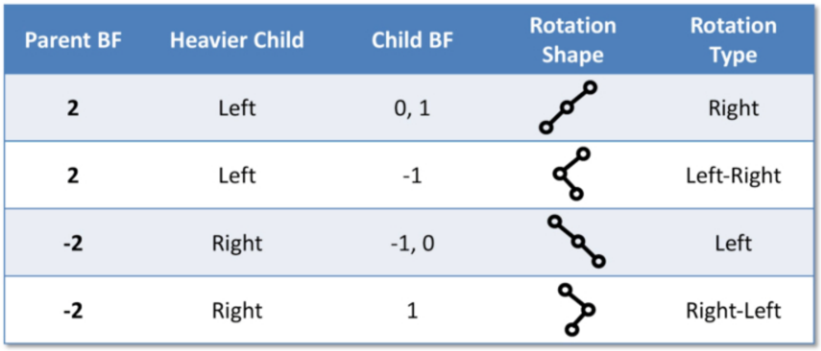
rotateLeft & rotateRight

The rotateLeft and rotateRight methods will perform a single rotation on the passed in subtree and return the new root of the subtree after it is rotated. Both of these methods will utilize updateHeightAndBF. Be sure to refer to the module video on rotations for pseudocode!

Hint: Both of these methods can be completed in just 6 lines!

balance

The balance method will chain together updateHeightAndBF, rotateLeft, and rotateRight in order to provide the rotation functionality of an AVL. The method template is already filled out for you; simply utilize the following chart to fill in the conditionals. More information can be found in the module videos.



Parent BF	Heavier Child	Child BF	Rotation Shape	Rotation Type
2	Left	0, 1		Right
2	Left	-1		Left-Right
-2	Right	-1, 0		Left
-2	Right	1		Right-Left

General Tips

- Be sure to follow the instructions closely, as success in part two of this assignment heavily depends on part one being correct.
- In rotateRight and rotateLeft, be sure that you return the new root of the subtree.
- We highly recommend copying the starter code and working in your preferred IDE in order to have access to features such as code completion, auto-formatting, and much more!

Here are general assignment guidelines that should be followed.

- Do not include any package declarations in your classes.
- Do not change any existing class headers, constructors, instance/global variables, or method signatures. For example, do not add throws to the method headers since they are not necessary. Instead, exceptions should be thrown as follows: `throw new InsertExceptionHere("Error: some exception was thrown");`
- All helper methods you choose to write should be made private. Recall the idea of Encapsulation in Object-Oriented Programming!
- Do not use anything that would trivialize the assignment. (e.g. Don't import/use java.util.ArrayList for an ArrayList assignment.)
- Always be very conscious of efficiency. Even if your method is to be $O(n)$, traversing the structure multiple times is considered inefficient unless that is absolutely required (and that case is extremely rare).
- If applicable, use the generic type of the class; do not use the raw type of the class. For example, use `new LinkedList<Integer>()` instead of `new LinkedList()`.

Use of the following statements should be avoided at all times.

package	System.arraycopy()	clone()
assert()	Arrays class	Array class
Thread class	Collections class	Collection.toArray()
Reflection APIs	Inner or nested classes	Lambda Expressions

The Vocareum (code editor) interface has six main components:

- The **Drop-Down** in the top left. This lets you choose from multiple available files. Note that this drop-down will only be visible in assignments that require multiple files.
- The **Run** button. This will compile your code and run a file scan. Running your code will not count towards your total allowed submission attempts, therefore you are free to run as many times as needed.
- The **Submit** button. This will compile your code, run a file scan, grade your assignment, and output results to console. Note that for most assignments in this class, you will only be allowed a limited number of submissions. A submission is counted when the submit button is clicked, regardless of whether or not your code can compile or if there are any file issues. Therefore, we **highly recommend** that you run your code before submitting to ensure that there are no issues that will prevent your code from being graded and that every submission attempt will generate meaningful results.
- The **Reset** button. This will revert all your changes and reset your code to the default code template.
- The **Code Window**. This is where you will write your code. For large coding assignments, we highly recommend copying the starter code and working in your preferred IDE to have access to features such as code completion, auto-formatting, and much more!
- The **Output Window**. This window will appear whenever you run or submit your code and will display the output for you to view.

For additional help, please visit the Vocareum information page located in the course information module!

< Anterior

Siguiente >

edX

Acerca de
Afiliados
edX para negocios
Open edX
Carreras
Noticias

Legal

Condiciones de Servicio y Código de Honor
Política de privacidad
Políticas de Accesibilidad
Política de marcas
Mapa del Sitio
Política de cookies
Opciones de privacidad

Contáctanos

Centro de ideas
Contáctenos
Centro de Ayuda
Seguridad
Kit Multimedia

GET IT ON Google play
Download on the App Store