ENSF 338 – Practical Data Structures and Algorithms
Winter 2023

Project: module 2 handout

The objectives of this module:

Create a library for tree data structures

a. Binary search trees:
b. AVL trees:

This library implements two hierarchical classes, namely:

1. Binary Search Tree (BST)
2. AVL Tree (AVL)

Step 1: pre-planning

The aforementioned classes require the implementation of a new node class. Since the node class for an AVL tree is a modified BST node (added balance member and extra requirements), we can implement a general node class that works for Both classes

When it comes to AVL tree implementation, the class extends a lot of the functionality of basic BST, hence it is advised that the AVL tree structure extends the BST

Step 2: implementation

- **TNode**:

This class is a general tree node class that has requirements for both BST and AVL trees. In the node sub-library, add the implementation of the tree node

Member variables of this class are:

- Int **data** member
- Tnode **left**
- Tnode **right**
- Tnode **parent**
- Int **balance**

The class must implement all needed:

- **Setters**
- **Getters**
- **print**: prints the node information to console in a user friendly format
- **toString**: returns the data member as a string (will be used for the tree prints)
- constructors:
  - **TNode():** a default constructor without arguments that initializes members to default values.
  - **TNode**(int data, int balance, TNode P, TNode L, TNode R): An overload constructor that takes an integer data, an integer balance to initialize the data and balance members. Initializes the parent using the P argument, L to initialize left child, and R to initialize right child

- **BST:**

This class is the implementation of the Binary Search Tree (BST) for integer data. This class will be added to the tree sub-library folder

Member variables of the class are:

- TNode **root**: references the root of the tree

The class must implement the following:

- Constructors:
  - Default constructor initializing root to null

        - Overload constructor **BST**(int val) which takes in an integer value, and creates a TNode and use it as root
        - Overload constructor **BST**(TNode obj) which takes a TNode as an argument and use it as the root of the tree. The TNode obj can have children which would make this tree object reference a sub-tree structure
    - **Setter** and **getter** for root
    - **Insert**(int val): creates a new node with data val to be inserted into the tree
    - **Insert**(TNode node) : inserts the node passed as argument into the tree
    - **Delete**(int val): finds the node with val as data and deletes it, if not found prints a statement that the value is not in the tree
    - TNode **Search**(int val): searches for the node with val as data and returns it or returns null if not found.
    - **printInOrder**(): prints the content data of the tree in ascending order
    - **printBF**(): prints the content of the tree in Breadth-First order, each level of the tree will be printed on a separate line

- **AVL Tree:**

This class is the implementation of the self-balancing AVL tree for integer data members. This class will be added to the tree sub-library folder

Member variables of the class are:

- TNode **root**: references the root of the tree

The class must implement the following:

- Constructors:
    - Default constructor initializing root to null
    - Overload constructor **AVL**(int val) which takes in an integer value, and creates a TNode and use it as root
    - Overload constructor **AVL**(TNode obj) which takes a TNode as an argument and use it as the root of the tree. If the TNode obj has children, the constructor needs to create a balanced tree from passed tree by one of the two following options:
        - iteratively inserting nodes from the original tree and balancing the new created AVL tree
        - implementing a full tree balancing algorithm (***Bonus***)
- **Setter** and **getter** for root: the setter function must check if the node has children. If children are found it must do the same as the overload constructor.
    - ***Hint: it is better to have a helper function (private function) that creates an the AVL tree and call it for the constructor and the setter***
- **Insert**(int val): creates a new node with data val to be inserted into the tree
    - Must maintain the tree balance. It can call the super.insert (insert function from BST), but will need to also balance the tree after
- **Insert**(TNode node) : inserts the node passed as argument into the tree
    - Must maintain the tree balance. It can call the super.insert (insert

function from BST), but will need to also balance the tree after

- **Delete**(int val): finds the node with val as data and deletes it, if not found prints a statement that the value is not in the tree (**Bonus**)
- TNode **Search**(int val): inherited from parent
- **printInOrder**(): inherited from parent
- **printBF**(): inherited from parent

Hint: implement the balance function as a helper function (private function) and call it in the insert (and Delete if you attempt the bonus)

Library content after completion of this module:

- myLib
  - datastructures
    - nodes
      - SNode.java (optional)
      - DNode.java
      - TNode.java
    - Linear
      - SLL.java
      - DLL.java
      - CSLL.java → extends SLL
      - CDLL.java → extends DLL
      - StackLL.java → extends SLL
      - QueueLL.java → extends SLL
    - Trees
      - BST.java
      - AVL.java → extends BST