Dr. Dapeng Dong

Maynooth University
National University of Ireland Maynooth

CS211FZ Data Structures & Algorithms (II)

Lab 2 – Balanced Search Trees and Applications

Objectives

- Understand how balanced search trees work
- Exercise with AVL trees and be able to customize a standard AVL tree suitable for a given problem
- Reflect the knowledge learned in the class

NOTE:

- **Do NOT use "package" in your source code**
- **You must submit the source code files, i.e., the ".java" files.**
- **You are allowed to use course reference books or class notes during the lab.**
- **Sharing your work with others is NOT allowed.**

Task 1: Understand AVL trees

A basic implementation of AVL tree is given on the Moodle course page (***SpecialAVLTree.java***). The source code is provided without any comment. Your first task is to write a comment to explain how the method "***balance(Node node)***" works for AVL trees.

IMPORTANT:
All new methods or modification to the code must be done within the "**SpecialAVLTree.java**" file. No extra classes should be created or used, i.e., the final code must be a single java file.

Task 2: Extend the AVL tree implementation to accommodate duplicated keys

In this task, you are required to

1. extend the existing implementation, so that the AVL tree can hold records with the same key but potentially with different associated data, for example, the tree should be able to store the key-value pairs: *<2, John>, <2, Mary>*.
2. insert all the records from the file "***Lab2_PopularNames.csv***" into the tree. The values in the first column should be used for the '***keys***', and the second column should be used as the '***associated data***' or '***values***'. The key and value in each record are separated by a comma (,)
3. print all the records line-by-line in an ascending order (Note, you need to print the records stored in the AVL tree, not from the file directly)

Note that the file contains 2000 names for boys and girls. The popularity of the names are ranked in numerical values, however, some names are equally popular, thus ranked the same, i.e., some records have the same key but different associated data.

Task 3: Provide a search method

The given implementation does not have a search method. In this task, you are required to

1. provide a search method that takes a key as an input, the method should be named "***search(int key)***"
2. if the record is found, it should print the record on screen in the format: "***key : value***"; if the record does not exist in the tree, it should print a message "***No match records were found!***".
3. Test your method with an input key: ***1492***. It should return "***Kora***"; test your method with another input key: ***2001***, it should return "***No match records were found!***".

Task 4: Provide a *Lazy* remove method.

The "***remove( )***" method given in the implementation is actually removing the node from the tree, and then rebalance the tree. But in many real-world applications, nodes need not to be removed but marked as removed, hence the *lazy* removing. In this task, you are required to

1. implement a lazy removing method, named "***lazyRemove(int key)***"
2. using your ***lazyRemove*** method to remove the record with key: ***1492***, and then using your search method to find the record with key: ***1492*** (it should return "***No match records were found!***")
4. print all the records line-by-line in an ascending order (Note, the list should not contain any removed records, and you need to print the records stored in the AVL tree, not from the file directly)

Task 5: Verify your methods

This task must follow Task 1, 2, 3 and 4.

To verify the correctness of your methods, try to:

1. use the "***insert***" method to insert a key-value pair ***<1492, Kora>*** into the tree, and then use your "***search***" method to find the record. The "***search***" method should return the same key pair. (Hint: you may need to modify the "***insert***" method so that it will work with the "***lazyRemove***" method).
2. following the previous insertion, insert another record ***<1492, Kora1>*** to the tree, and then use your "***search***" method to find the record. It should print the same key with two values on the screen.