Gebze Technical University
Computer Engineering

CSE 222
2017 Spring

HOMEWORK 5 REPORT

VAKHID BETRAKHMADOV
141044086

Course Assistant: Nur Banu Albayrak

1. Class Diagrams

Trees.pdf

2. Problem Solutions Approach

Q1:

One of the ways to implement pre-order traverse of a tree is using stack data structure. This might be achieved by pushing the root node of the tree to the top of a stack and then popping it subsequently pushing right and left nodes of the root to the top the stack, popping next node from the stack and pushing left end right nodes of that node and so on. Thus, until the stack is empty. We can implement level-order traverse of a tree the same way, but using queue data structure, by offering root node to the queue and polling it back subsequently offering left and right nodes and so on. Both stack and queue data structures might be used through just one interface, which is Deque. So, I implemented iterator class “BinaryTreeIterator” in the BinaryTree class, which has a protected helper method “void next(Deque<Node<E>>)” which manipulates Deque interface. In BinaryTree class it is manipulated as a stack, but in BinarySearchTree class we extend “BinaryTreeIterator” class and override our helper next method, so that Deque interface is manipulated as a Queue.
This provides a lot of code reuse.
I also have extended BinaryTree class in CompleteBinaryTree class and have overridden “add” method, such that tree is always balanced (complete). This was achieved by usage of queue data structure. We offer to the queue and subsequently poll until next polled node has its left or right node empty, thus completely filling all levels of the tree.

Q2:

In the FamilyTree class, before inserting any new person to the tree, we first need to check if the insertion can be completed successfully. We first traverse the whole tree and find number of matches according to the given person’s parent name and his nickname. We have two different options on how to search for the person’s parent, that is using parent-child relationship (ebu - …) or child-parent relationship (ibn - …). So, if we concatenate person’s parent name with his nickname without prefix (ebu-), achieved string will match parent-child relationship, that is string achieved by concatenation of a node and any of its children. In the child-parent relationship string achieved by concatenation of person’s parent name and his nickname without prefix will match string achieved by concatenation of a node and any of its children in reverse order. We can have just one method which will traverse the tree and check for matches, this method will take BiFunction interface as an argument which will be implemented using lambda expressions to concatenate nodes and their children in the straight or reverse order. After we make sure that the tree has only and only one match we can insert a new person into the tree.
We don’t need to implement pre-order traverse and iterator methods, because they are inherited from BinaryTree class and work as expected.