CSCI 102 assignment 4 – Linked trees

Feb 24

In our implementation of `LinkedTree` we added methods to access nodes in the tree but we did not add any way to add or remove nodes – we left that up to subclasses. In this assignment, you will add methods to add and remove nodes from our class. We will also add code to add and remove entire subtrees. Please submit a java project with code that does the following. You may copy any code on the class website.

- Create a new class `AddRemoveLinkedTree` that is a subclass of `LinkedTree`. You may want to change the visibility of `Node` to make it accessible in the subclass (dont don't make it `public`!).
- Add a method `public void addChild(Position<E> p, E element)` which makes a new node that stores `element` and makes it the last child of `p`.
- Add a method `public E removePosition(Position<E> p)` which removes `p` from the tree, returns the element it stores, and makes all of its children the children of its parent – if `p` is the i-th child of its parents then its children should become the i-th, $i+1$-th, ..., children of the parent. If `removePosition` is called on the root, do nothing and return `null`. Hint: if your code for this question is short, then it's probably wrong!
- Add a new method to get the subtree rooted at a certain node `public AddRemoveLinkedTree<E> getSubtree(Position<E> p)`. To do so you will need to make a constructor for `AddRemoveLinkedTree` that has signature `(Position<E> p)`. Remember to keep track of the size!
- Add a method `public void addSubTree(Position<E> p, AddRemoveLinkedTree<E> tree)` which adds `tree` as a subtree by making its root the last child of `p`. Also add the method `public AddRemoveLinkedTree<E> removeSubTree(Position<E> p)` which removes `p` and all of its decendants from the tree and returns the subtree rooted at `p`.
- Include a main method that creates a `AddRemoveLinkedTree<String>` with root "Root". Using `addChild`, give the root 26 children which store the capital letters of the alphabet in order. Using `addChild` again, give each of the children of the root 10 children storing the numbers 1 to 10 in order, prepended with the letter of the position – i.e. the position which stores `F` will have children storing `F1`, ..., `F10`. Print the pre-order traversal of this tree by calling the `positions` method. Use `removePosition` to remove the node storing `F` and print the pre-order traversal again. Use `removeSubTree` to remove the node storing `L` and print the pre-order traversal again. Finally, using `addSubTree`, add the subtree you just removed below the position that stores "C5" and print the pre-order traversal again.

Please submit your code and answers to the questions in a zipped folder on brightspace by March 3. Remember to use the principles of encapsulation and least privilege!