CPE 201:Fall 2023

Assignment 3

Assigned on December 6, 2023. You should submit your program through Teams by December 20 (any time up to midnight). Remember that your code should be fully documented. I once again remind you to read the academic honesty policy stated in the syllabus.

The goal of this assignment is to implement a `OrderedGeneralTree` class, in which every node may have 0 or more children. You will implement a general tree using a binary tree representation as explained below.

We can define a binary tree representation T' for an ordered general tree T as follows (see Figure below):

- For each position p of T , there is an associated position p' of T' .
- If p is a leaf of T, then p' in T' does not have a left child; otherwise the left child of p' is q', where q is the first child of p in T.
- If p has a sibling q ordered immediately after it in T , then q' is the right child of p' in T ; otherwise p' does not have a right child.

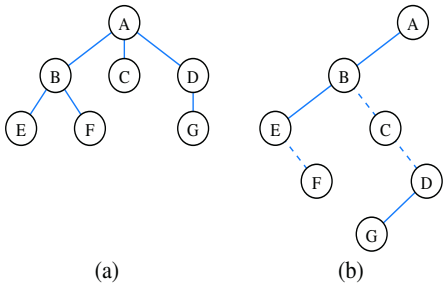

Figure 1: (a) General Tree T (b) Binary tree representation T' of T

Your implementation MUST follow the specifications given below:

- A Node of the tree has to store references to its element, its parent, its firstChild and its nextSibling. For example, in the above tree, in addition to its element, node B must store a reference to its parent A, a reference to its firstChild E, and a reference to its nextSibling C.

- Your `OrderedGeneralTree` class must implement the Tree interface we discussed in class, that is, all methods given in the Tree interface needs to be implemented. You may use the AbstractTree class as a base class if you wish.

- In addition your class must provide update methods so that nodes may be inserted or deleted. In particular, the following update methods (with the given signatures) must be implemented:

  - `Position addRoot(E e)`: creates a new root node with element e, and returns the node as a position. If the tree is non-empty this method should return null.
  - `Position addChild(Position p, E e)`: Creates a new child node of position p. The child nodes must be linked in the order they are added. The newly made child node is returned as a position.
  - `E remove(Position p)`: Removes the position p and returns the element in p. If p has children, they become the children of p's parent, appearing in the same order as p appeared. For example in the example tree given above, if node B is removed, its two children will be adopted by A (B's parent), and after the removal of B, A's children must appear in the order E, F, C, D. If p is the root of the tree, this method does nothing and returns null.

- In your class also provide a method `displayTree()` that would display the tree as follows:

  - Nodes should be printed according to their order in an *preorder* traversal, one node per line.
  - The element for a node should be preceded by *depth* number of dots, where *depth* denotes the depth of the node in the general tree. Put one space character after each dot.

  Here is the output for the example tree given above.

```
A
. B
. . E
. . F
. C
. D
. . G
```

- Write a driver program to test all methods of the `OrderedGeneralTree` class.