Assessed Exercise week 4

**Before attempting the tasks, please check the “hints” you can find below each task.**

**Task A: Visit of Binary Trees**

Implement a Console Application for C#. Add the classes Node and BinTree completed in the Lab. Add in the BinTree class the following methods to visit the tree: InOrder, PostOrder and PreOrder.

Call (in the Main() method) the InOrder, PostOrder and PreOrder methods on the tree constructed as below and display the result of the methods.

*Hints: We have completed the InOrder method in the Lab. The methods PostOrder and PreOrder need to be completed following the pseudo-code that can be found on the lab slides. Note the similarities in the pseudo-code between the three visit methods. On the lab slides there is also an example on how to construct a tree and how to call a visit method in the Main() method and display the result of the visit.*

**Task B: Algorithms on Trees**

Implement a binary tree that stores *strings* as data (modify, in the opportune way, the classes Node and BinTree implemented in the lab).

Implement in the class BinTree (and systematically test in the Main()):

* A method “Largest” that returns the largest string present in the tree (i.e., largest in the lexicographic order).
* A method called “Ancestor” that takes as arguments two strings and (if they are both present in the tree) returns the string associated to the first common ancestor node in the tree (see example below). This method is extensively used in many applications from bioinformatics (eg phylogenetic trees) to artificial intelligence.

*Hints: For the method “Largest” one may add the strings present in the tree to a list and then find the largest string present in the list. Another (more elegant) solution could use the recursive structure of the tree (use the same idea as we did for the method to count the number of nodes or to compute the height of the tree). Remember that one can compare strings using the CompareTo built-in method seen in Week 1 (Lab exercises) (https://docs.microsoft.com/en-us/dotnet/api/system.string.compareto?view=net-5.0).*

*For the “ancestor” method: if one can get the paths from the root to the nodes containing the two strings it should be easy to find the first common ancestor (to get the path from the root to a node you may modify one of the visits of the tree). Eg, in the tree below the path from the root to wa is ab-bh-af-wa, while the path from the root to vv is ab-bh-vv*

Example:

The method Ancestor that takes as arguments the string “wa” and “vv” should return “bh” which is the string associated to the first common ancestor.

The node with string “ab” is also a common ancestor but not the first (i.e., “closest”) one.