Assignment 6

Objective: Binary trees are among the simplest and most intuitive data structures studied in undergraduate Data Structure and Algorithms courses. And with good reason: binary trees have many interesting properties that recommend them as the data structure of choice in almost all areas of computer science and engineering. One of the side benefits of their simple, yet rich structure, is that binary trees can be implemented easily and, in some special cases, without the use of pointers, an important consideration in early programming languages such as FORTRAN and BASIC that lacked pointers.

Over the years, binary trees have been specialized to meet the specific needs of various applications. For example, Full Binary Trees (FBT, for short) that are the object of your current assignment, find applications to expression evaluation in parallel processing and compiler design. While one can look at FBTs from many angles, this assignment takes the view that they are recursively defined objects. As it turns out, this view simplifies considerably the solution of most of the problems on your assignment.

Statement of your assignment: The class of regular binary trees (FBT, for short) is defined recursively as follows:

B: The empty binary tree is an FBT
I: If $T_1$ and $T_2$ are either both empty FBTs, or both non-empty FBTs, then the tree obtained by adding a new root and making, respectively, $T_1$ and $T_2$ into the left and right subtrees of the new root is an FBT.
C: Nothing is an FBT unless it was obtained by one of the previous clauses.

As an illustration, the binary tree depicted in Figure 1 is an FBT.

![Figure 1: Illustrating a regular binary tree.](image)

Problem 1. (25%) Show the complete sequence of operations that, starting with an empty FBT, constructs the FBT in Figure 1. Show all the partial FBTs involved.

Solution. There are many possible solutions. We propose a solution that takes the form of a decomposition tree.
In this view, the original FBT is decomposed, in stages, into empty FBTs out of which it arises by a series of operations. Please refer to Figure 2 for an illustration.

![Figure 2: The decomposition tree for the FBT in Figure 1.](image)

Problem 2. (25%) Prove by induction that a node of an FBT is either a leaf or else has exactly two children.

Solution. There are several solutions possible. One can conduct induction either on the number of nodes in the FBT or else by the number of operations that were used to build the tree. While we shall adopt the former, the students are encouraged to investigate the latter line of attack.

Consider an \( n \)-node FBT. We conduct induction on \( n \). To settle the Basis, observe that if \( n = 1 \) then the FBT reduces to a singleton that must be a leaf.

For the Inductive step, let \( n > 1 \) be arbitrary and assume the statement true for all FBTs with strictly fewer than \( n \) nodes. In this case, our FBT must arise from two nonempty FBTs, say, \( T_1 \) and \( T_2 \) along with a new root node.\(^1\) Visibly, both \( T_1 \) and \( T_2 \) have strictly fewer nodes than the original tree and are themselves FBTs. By the induction hypothesis, every node in \( T_1 \) and every node in \( T_2 \) is either a leaf or else it has exactly two children. In addition, the new added root has exactly two children itself, namely, the roots of \( T_1 \) and \( T_2 \), respectively. The conclusion follows.

Problem 3. (25%) This is the converse of Problem 2. Prove that a binary tree with the property that every node is a leaf or else has exactly two children is an FBT.\(^2\)

Solution. This time, we are told that a binary tree has the property that each of its nodes is either a leaf or else possesses two children. The goal is to prove that such a tree must be an FBT.

Again, we choose to conduct induction on the number \( n \) of nodes in the binary tree. For the Basis of the induction we can take either \( n = 0 \) or \( n = 1 \). Let us take \( n = 0 \). A binary tree on 0 nodes must be the empty tree.

---

\(^1\)See the Induction Clause [I] of the FBT definition.

\(^2\)For all practical purposes, we have established an alternate, equivalent, definition of FBTs based on the regularity spelled out above. It is precisely this property that suggested the name of the class itself.
that satisfies the property vacuously.\textsuperscript{3}

For the Inductive step, let \( n > 1 \) be arbitrary and assume the statement true for all binary trees on fewer than \( n \) nodes that satisfy the problem statement. We notice that tree must have two subtrees because the root must have two children. Referring to Figure 3, let these subtrees be \( T_L \) and \( T_R \), respectively.

In order for us to be allowed to apply the induction hypothesis we need to argue that both \( T_L \) and \( T_R \) inherit from the original tree the defining property, namely that each of their nodes is either a leaf or else has two children. To see that this is the case, we argue by contradiction. Assume, without loss of generality, that \( T_L \) does not satisfy the said property. In this case, we must find an internal node of \( T_L \) that does not have two children. However, such a node contradicts that the original tree, itself, has the property that all its internal nodes have two children. We conclude that both \( T_L \) and \( T_R \) must inherit the property.

Since both \( T_L \) and \( T_R \) have fewer than \( n \) nodes, by the induction hypothesis they are FBTs. Now, the original tree arises from two FBTs by the addition of a new node, a legal operation. It follows that the original tree must be an FBT and the proof is complete.

![Figure 3: Illustrating the solution of Problem 3.](image)

Problem 4. (5%) Prove by induction that every non-empty FBT contains an odd number of nodes. Solution. We proceed by induction on the number, \( n \), of nodes in the FBT. To settle the basis, observe that in the case \( n = 1 \), the number of nodes in the tree is clearly odd.

Next, let \( n \) be arbitrary and assume the statement true for all FBTs with fewer than \( n \) nodes. Referring, again, to Figure 3 where \( T_L \) and \( T_R \) are the left and right subtrees our \( n \)-node FBT. As before, both \( T_L \) and \( T_R \) are FBTs and both have fewer than \( n \) nodes. By the induction hypothesis, both \( T_L \) and \( T_R \) have an odd number of nodes. Consequently, the original tree must have an odd number of nodes.

\textsuperscript{3}Had we taken \( n = 1 \) for the basis, we would have argued that such a tree is a singleton, which is an FBT for it arises from two empty FBTs and a new root.