```scala
package tree

import tinyscalautils.lang.InterruptibleConstructor

final case class Tree[+A](root: A, forest: Forest[A]) extends InterruptibleConstructor:
  def isEmpty: Boolean                         = false
  def isTree: Boolean                          = true
  def toTree: Tree[A]                          = this
  def toForest: Forest[A]                      = Forest(IndexedSeq(this))
  def size: Int                                = 1 + forest.size
  def depth: Int                               = 1 + forest.depth
  def width: Int                               = forest.width
  def find(test: A => Boolean): Option[A]      = if test(root) then Some(root) else forest.find(test)
  def count(test: A => Boolean): Int           = if test(root) then 1 + forest.count(test) else forest.count(test)
  def exists(test: A => Boolean): Boolean      = if test(root) then true else forest.exists(test)
  def forall(test: A => Boolean): Boolean      = if test(root) && forest.forall(test) then true else false
  def foreach[U](f: A => U): Unit              = {
   f(root)
   forest.foreach[U](f)
  }

  def toList: List[A]                          = root :: forest.toList
  def fold[B](init: B)(f: (B, A) => B): B      = {
    val result = f(init, root)
    forest.fold(result)(f)
  }

  def + [B >: A](tree: Tree[B]): Forest[B]     =
      if (root == tree.root) then
        Tree(root, forest + tree.forest).toForest
      else
        Forest(IndexedSeq(this, tree))

  def + [B >: A](forest: Forest[B]): Forest[B] =
    forest.trees.foldLeft(this.toForest: Forest[B])((acc, t) => acc + t)

  def filter(test: A => Boolean): Forest[A]    = ???
  def toPaths(separator: Char): Seq[String]    = ???
end Tree

object Tree:
   def apply[A](root: A, trees: Tree[A]*): Tree[A] = Tree(root, Forest(trees))

   def branch[A](values: Seq[A]): Tree[A]        = ???
   def branch[A](value: A, values: A*): Tree[A] = branch(value +: values)
end Tree

```