```scala
package tree

import tinyscalautils.lang.InterruptibleConstructor

final case class Forest[+A](trees: IndexedSeq[Tree[A]]) extends InterruptibleConstructor:
  def isEmpty: Boolean                          = trees.isEmpty
  def isTree: Boolean                           = trees.size == 1

  def toTree: Tree[A]                           = {
   if (!isTree) {
     throw new NoSuchElementException()
   }
   trees(0)
  }

  def toForest: Forest[A]                       = this
  def size: Int                                 = trees.map(_.size).sum
  def depth: Int                                = if isEmpty then 0 else trees.map(_.depth).max
  def width: Int                                = if isEmpty then 0 else (trees.size +: trees.map(_.width)).max
  def find(test: A => Boolean): Option[A]       = if isEmpty then None else trees.map(_.find(test))(0)
  def count(test: A => Boolean): Int            = if isEmpty then 0 else trees.map(_.count(test)).sum
  def exists(test: A => Boolean): Boolean       = if isEmpty then false else trees.exists(_.exists(test))
  def forall(test: A => Boolean): Boolean       = if isEmpty then true else trees.forall(_.forall(test))
  def foreach[U](f: A => U): Unit               = if !isEmpty then trees.foreach(_.foreach[U](f))
  def toList: List[A]                           = if isEmpty then Nil else trees.flatMap(_.toList).toList
```

```
26   def fold[B](init: B)(f: (B, A) => B): B      = if isEmpty then init else trees.foldLeft(init)((acc, tree) => tree.fold(acc)(f))
27
28   def + [B >: A](tree: Tree[B]): Forest[B]     = {
29     val (same, others) = trees.partition(_.root == tree.root)
30     same.headOption match
31       case Some(t) =>
32         val merged = t + tree
33         Forest(others ++ merged.trees)
34       case None =>
35         Forest(trees :+ tree)
36   }
37
38   def + [B >: A](forest: Forest[B]): Forest[B] =
39     forest.trees.foldLeft(this: Forest[B])((acc, t) => acc + t)
40
41   def filter(test: A => Boolean): Forest[A]    = ???
42   def toPaths(separator: Char): Seq[String]    = ???
43 end Forest
44
45 object Forest:
46    def apply[A](trees: Seq[Tree[A]]): Forest[A]            = Forest(trees.toIndexedSeq)
47    def apply[A](tree: Tree[A], trees: Tree[A]*): Forest[A] = Forest(tree +: trees)
48    def empty[A]: Forest[A]                                 = Forest(IndexedSeq.empty)
49
50    def fromPaths(paths: Seq[String], separator: Char = '/'): Forest[String] = ???
51    def fromPaths(path: String, paths: String*): Forest[String] = fromPaths(path +: paths)
52 end Forest
53
```