Part 6

—

Depth-First Search

CSCI 3110 Code

Summer 2015

Now let’s put our graph traversal framework from *Algos.Graphs.Traversal* to work to compute a DFS forest of the graph. Here’s the type signature of the function we want:

```
dfs :: AdjList v vl el → Forest V E
```

We already have a graph traversal function. What’s missing is a vertex set data structure, which in the case of DFS should behave like a stack.

```
dfs = traverse makeVertexStack
```

This vertex stack is of course easy to implement using the stack implementation from *Algos.DS.Stack* stored in an *STRef*. We also need an array to keep track of explored vertices:

```
data VertexStack s = VertexStack (STArray s Int Bool) (STRef s (Stack (V, [(E, V)])))
```

To create such a vertex stack, we simply allocate a new Boolean array of size n all of whose entries are initially *False*—all vertices are initially unexplored—and we create a new *STRef* initially storing an empty stack:

```
makeVertexStack   :: Int → ST s (VertexStack s)
makeVertexStack n = VertexStack $⃝ newArray (1, n) False ⊛ newSTRef emptyStack
```

Next the implementations of the two set operations:

```
instance VertexSet VertexStack where

  add (VertexStack _ st) v p = modifySTRef st (flip push (v, p))
```

```
remove (VertexStack exp st) = readSTRef st ⟫= rem
  where rem s = case top s of
          Nothing      → writeSTRef st s ⟫ return Nothing
          Just p@(v, _) → do e ← readArray exp (vIx v)
                             if e then rem (pop s)
                               else  do writeSTRef st (pop s)
                                        writeArray exp (vIx v) True
                                        return (Just p)
```

add simply pushes the given pair (v, p) onto the stack. *remove* reads the stack and passes it to the helper function *rem*. If the given stack is empty, we write this information back into the *STRef* and return *Nothing*. Otherwise, we inspect the topmost pair p. If its vertex v is already explored, which we check by reading the array *exp*, then p should not be returned, so we recurse on the tail of the stack using *rem* (*pop s*). Otherwise, we store the tail as the new stack content, mark v as explored, and finally return *Just p*.