CS119 – Module 11: ADT Sets

Purpose: Separating the behavior of a data type with its implementation allows us to make changes in the implementation without having to change how the data type is used. We will illustrate this concept by extending as well as changing the ADT for the Set data type.

Knowledge: This module will help you become familiar with the following content knowledge:

- operations of the Set ADT
- multiple implementations of the Set ADT
- analysis of the Set operations
- Binary Search Tree

Activity: With your group perform the following tasks and answer the questions. You will need to use files in the `lab11` directory. You will be reporting your answers back to the class in 30 minutes.

Consider the abstract data type Set:

Function	Explanation
`empty :: Set a`	gives an empty set
`isEmpty :: Set a -> Bool`	determines if a set is empty
`member :: Set a -> a -> Bool`	determines whether a value is contained in a set
`insert :: a -> Set a -> Set a`	adds a value to a set
`delete :: a -> Set a -> Set a`	deletes a value to a set

1. In an Abstract Data Type (ADT) the behavior of the data as indicated by the table above is divorced from the implementation. The Set data type can be implemented in many different ways and the program that uses that data type doesn't have to be concerned about the actual implementation.

   Suppose that we decide to implement this ADT with a list of elements as provided in the notes . If we have a set with n elements, in the worst case how many elements would have to be examined in the `member` operation?

   How many elements would have to be examined in the `insert` operation?

   Give the order of growth for both of these operations with this implementation.

2. Suppose that we decide instead to implement this ADT with a binary search tree (BST) as provided in the notes . In a binary search tree all elements in the left subtree will be less than the root, all elements in the right subtree will be greater than the root, and the subtrees will also be binary search trees.

   Draw two different BSTs for the set {5,10,22,26,30}.

3. What are the sequences of nodes examined for each of your trees when using the `member` operation for seeing if the value 15 is contained in the set?

   What happens for each of your trees when we insert the value 15?

   Draw the new trees with the inserted value.

   Give the order of growth for both of these operations with this implementation.

4. What happens if we try to delete the value at the root of your tree? Draw the new tree.

Complete the following assignments to be submitted for grading. Each should be done individually but you can consult with a classmate to discuss your strategies or if you get an error message that you do not understand.
I have renamed the modules in the `lab11` directory to `Set1` and `Set2`. You can simply change the `import` statement in the `Example11` module to change the representation that is being used.

Assignment 1:
Add set operations that compute union and intersection operations in the `Set1` module.

Give the order of growth for both of these operations, along with a brief explanation of how you arrived at those results.

Hint: One way to approach these problems is to iterate through the list of the first set, checking to see if each item is in the second set.

Criteria for Success: In the `Example11` file create a couple of new sets as union and intersection of some of the existing sets. Unfortunately, we can not print a set so you will need to use the `member` function to test to see if the values you expect are in the set. Be sure to test that values that are not supposed to be in the set are also not present, using the `member` function.
Make sure that you have clearly explained why you believe the order of growth values you provided are accurate.

Assignment 2:
Add set operations that compute union and intersection operations in the `Set2` module.

Give the order of growth for both of these operations.

Hint: You may want to write a function `flatten` which flattens a binary search tree into a list. Once you flatten one of the trees you can write a "helper function" which takes a list and and a Set and performs the same type of operations that you did for Assignment 1.

Criteria for Success: You can perform the exact same test procedure as you did in Assignment 1. The only thing that needs to change is the import. This is the power of ADTs at work!!
Make sure that you have clearly explained why you believe the order of growth values you provided are accurate.

Assignment 3:
Sets can be represented by boolean-valued functions which indicate whether a value is contained in the set or not. To make it easier to determine whether the set is empty or not, we will also include an integer value representing the size of the set:

```
type Set a = (Int, a -> Bool)
```

Create a new module for this representation and write the operations `empty`, `isEmpty`, `member`, `insert`, and `delete`.

Hints: It will probably feel strange to you to have a function as part of the data type but just keep in mind what information this function provides. The function actually tells us whether or not a value is a *member* of the set or not. Keep in mind the following:

1. The function `empty` returns a pair where the first value is an integer and the second value is a function.
   Complete the following : `empty = (_____,\x->_____)`
2. The function `isEmpty` takes a Set as a parameter and that parameter is a pair containing an integer and a function.
   Complete the following : `isEmpty (n,f) =` _____
3. The function `member` takes a Set and a value. Remember that the Set is a pair that contains a function that behaves like a membership function!
   Complete the following : `member (n,f) x =` _____
4. Both `insert` and `delete` take a value and a Set and return a new Set. This means that you need to define a new function which will be returned as part of the new pair.
   The structure of the functions will be as follows, where you are defining a new membership function `g` which will be used as part of the return value.

```
insert x (n,f) = ...
                 where
                      g y = ...
```

Criteria for Success: You can perform the tests as before in the file `Example9` where you create sets using `insert` and `delete` and test their membership.

Submit all your files in Canvas for grading.