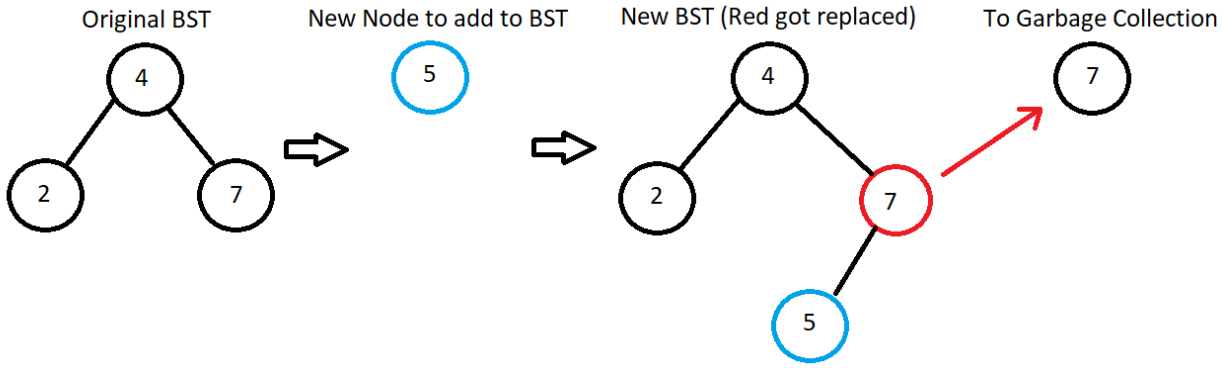
CSE 382 – Patterns in Functional Programming Final Exam

By David Doria

Explain the principle of persistence that exists with data structures written in a language that enforces immutability. In addition to your written response, include graphics that show how persistence applies to a specific data structure of your choice.

Persistence in a language that enforces immutability is used to prevent side effects in the code by preventing data from changing. Instead, when a change is required, these languages return completely new instances of the data with the modifications, and the old data goes out of scope and is collected by the garbage collector.

Here is a graphic of how persistence is applied to a Binary Search Tree. Unaffected nodes are reused, while nodes that would be mutated are replaced with a new node with the requested changes.



Describe the benefits of writing specifications and definitions prior to implementing a function in code.

Writing a specification and definition before implementing a function is a good way of organizing your thoughts before starting to implement that function. It allows you to have a map of sorts to follow while you implement it. It probably won't be implemented precisely as it is written, but it will surely help.

Provide a definition of what a functor is (from a programming perspective). Identify the benefits of the functors we demonstrated in class over the traditional `for loops` found in languages like Python, C++, and Java.

A functor is a type of function that takes a set of arguments and applies a function to them to achieve a set of results. For example, a functor like map can be used to apply a function to a list/set/array/etc. of values to return a completely different set of values. This allows the programmer to avoid using loops, which makes the code shorter and more readable.

Identify when it would be useful to curry a function.

Currying a function is useful if you need to split a complex function into smaller, less complex functions that can be executed when needed. When you return a function that is partially executed, you can store that function as a lambda and pass it around until you can execute the rest of it.

Describe the process of modifying a function to support partial applications.

To modify a function to support partial applications, you could change it to only accept a subset of parameters, then return a function that accepts the remainder as parameters.

Identify when you should write a monoid function and explain its purpose.

A monoid is a bunch of functions that all have one input and output (of the same type) and have an identity function. You should write a monoid if you have data that could benefit from a factory type process where the result of one function applied to the data needs to be applied directly to another function.

Explain why the bind function is important if we have a monad type.

The bind function is important because it allows the programmer to apply a monad type to a value or the result of a function. If the value is not valid, the bind function will return “null” or its equivalent.

Describe the benefits of creating a stream function and explain how the function uses lazy evaluation.

A stream function is like a queue, in that it accepts one input at a time. A stream will then evaluate the most recent item before moving onto the next. This is what is known as lazy evaluation, where computation is delayed until a further point in time. In a stream, each item in it’s queue is delayed until it is it’s turn to be processed.

Describe the purpose and the structure of a Min Heap.

A min heap is used to store and retrieve values. The least (or min) value is always at the root, and all of the parents are smaller than the children.

Describe the purpose and the structure of a Random Access List.

A random access list looks like an array in code, but can be used to quickly access any item without having to iterate over everything. Its structure is a list of binary search trees that increase in size. Each tree in the list has 2^n nodes, where n is its position in the list.