Maya T. Hawkins

CS203

BST Coding and Properties

The first important step to creating good coding is being able to test the properties of the program with a variety of inputs. By using the Random package of Java (see line 1), I was able to create binary search trees with random lengths and inputs. After creating a random number called randomInt(see line 166), I was able to create a random BST generator called randBST. There are two forms of randBST; one takes in a number that decides the max number of elements in the BST, and one takes in no inputs, and the random max number of elements is between 0 and 50. I say max number and not actual number because the function randBST generates a BST with elements anywhere between 1 and 75 randomly, but there is no way to predict if a randBST will attempt to add a number that is already in the BST. In that case, the BST would have 1 less than the max if a duplicate is added. There is also no telling how many duplicates the function will attempt to add, therefore, we only know the max number, and not the actual number until given an output.

All of the properties have a function created to test the property. There is a general form that takes an input, and a random form, that will return a random number of cases with a random number of elements within BSTs. This allows for the function itself to search for a problem in either the code or the function, as there is a form of each function to say when the property has failed. Random generating allows for inputs that I myself might not have thought of to possibly be tested, and if they are wrong, then something is wrong in my program.

The first property I tested deals with add, remove, and cardinality abilities of my program. If a number is added and removed from a BST, the cardinality of the BST should be either equal to or one less than the cardinality of the original BST. The cardinality would be one less in the case that the number to be added was already in the BST, in which case, it cannot be added. This property is tested by addRemCar (see lines 186-198). There are 3 possible outcomes of the function. The function first tests if the input element is in the input function (see line 189), and if so checks the cardinality difference and prints it in the output statement. The function should be correct, and whenever the element is in the BST should return with only a difference of 1(see lines 189-191). Otherwise the function checks if the cardinality of the BST after adding the element and removing the element is the same as the cardinality of the BST originally, in which a printout will state so (see lines 192-194). If the tests fails, it will also state so, saying the input does not equal the output (see lines 195-197).

This function, addRemCar, is referenced in the random generator version of it, called addRemCarRepeat (see lines 201-207). This allows for a large number of random BSTs and elements to be tested. addRemCar is also called on between lines 302 and 313, where there are examples of having an element that is a member if the BST and some that is not. The only way for this function to call false or give an unexpected outcome is if the difference is greater than two, which is impossible for the first case, as only one element is removed. As for the second case, the element is added first, bringing the cardinality up by one. Then, once that element is removed from the BST, as it should be since it became a member of the function, the cardinality went down by one, bringing the cardinality back to its original number. This function working and providing expected results shows that the functions member, cardinality, add, remove, and union(used in remove) all work correctly.

The next property I created incorporates inter, differ, and equal. As these functions use add and remove, should work as well. My second property states that the if the inter of some

BST u and t and the difference of u and t are unionized, they will be the same length as u. The function to describe this property is called inDiffEq (see line 212). The function first take the inter of u and t, and unionizes it with the difference from u and t. This BST is then compared to the original u. If all elements are the same, then the property is true (see lines 212-220). The only way to call false is if there is an element in the newly created u that was not previously in the old u. This should not happen because inter contains all the elements in u and t, and differ creates all the elements in u that are not in t. All the elements that are the same and all the elements that are different recreates the BST u. This function is called randomly by the function inDiffEqRepeat (see lines 223-229) to make sure that even on two randomly generated BSTs, that property still holds true.

The third property uses union and cardinality, both of which were used in previous properties. The cardinality of the union of two BSTs are the same as the sum of the cardinalities of each BST individually. This property only holds true for two BST that do not have the same elements. This property is tested by the function unionAndCard (see line 233). First, the function checks if the BSTs have any elements in common (see lines 237-239). If so, the function will say so, and not continue with checking the property. If no elements are in common, then the two BSTs are unionized, and the cardinality is compared with the sum of the cardinalities of the two BSTs individually (see lines 240-246). The random BST function is called unionAndCardRepeat (see lines 249-255). This function could only call false if the two numbers were not equal. This is impossible because all the elements in either BST put into one should have the same cardinality as if they were split up and looked at separately.

The final property uses subset and cardinality. Cardinality is already overlapped twice, and subset uses the other functions in its execution. The property states that if t is a subset of u, then the cardinality of the inter between u and t will equal the cardinality of t. This means that any BST that is a subset of another BST should have the same number of elements as the inter between the two BST. The function to describe this property is called subCard (see lines 259-279). The function checks if the t is a subset of u, and if not, does not check any further. If it is, then the function checks if the cardinality of the inter if the two BST is the same as the cardinality of t. The function is randomly generated by subCardRepeat (see lines 276-282). This function call false if they were not equal, which is impossible because everything that is in t is in u, so the inter must be all of t.

Of course, many more properties could be looked into, but this is a sufficient number of properties because it covers all functions and original properties of BSTs. That is what is necessary to see if a program runs well: unbiased testing and uses of all functions.