Assignment 6

Exercise 1: Statement and Branch Coverage

This is a pen an paper exercise. You are given buggy implementations of two methods. The first method `indexOf(int[] a, int n)` returns the index of the first occurrence of the integer `n` in the array `a` and the second method `average(int[] a)` computes and returns the average of all integers in the array `a`.

```
public static int indexOf(int[] a, int n) {
   if (a == null) {
      throw new IllegalArgumentException("Array is null");
   }
   int index = -1;
   for (int i = 0; i <= a.length; i++) {
      if (a[i] == n) {
         index = i;
         break;
      }
   }
   return index;
}
```

1. Draw a control flow graph for method `indexOf`.
2. Write a test case that reveals the bug in method `indexOf`.
3. Write a test suite that achieves 100% statement coverage for `indexOf` and does not find the bug.
4. Is there a test suite that achieves 100% branch coverage for `indexOf` and still misses the bug?

```
public static int average(int[] a) {
   if (a == null) {
      throw new IllegalArgumentException("Array is null");
   }
   if (a.length == 0) {
      throw new IllegalArgumentException("Array is empty");
   }
   int sum = 0;
   for (int i = 1; i < a.length; i++) {
      sum += a[i];
   }
   return sum / a.length;
}
```

5. Write a test case that reveals the bug in method `average`.

6. Write tests that achieve 100% branch coverage for `average` and miss the bug.

Exercise 2: Functional vs. Structural Testing

In this exercise, we test an algorithm that solves the *knapsack problem*: Given a set of items, each with a weight and a value, determine the subset of items to pack into your knapsack so that the total weight does not exceed its capacity and the total value is maximized.

A buggy implementation of an algorithm that solves the knapsack problem is given in `Knapsack.java`.

1. Derive a test suite that exercises the functional behavior of the algorithm. Fix any bugs you find.

2. Measure the branch coverage achieved by the initial test suite provided in `KnapsackTest.java`. Add tests until you get 100% branch coverage. Fix any bugs you find. We suggest that you use an IDE for this. To measure the branch coverage you can use IntelliJ[1] or other tools.

3. Discuss the advantages and disadvantages of structural and functional testing.

[1] See `https://confluence.jetbrains.com/display/IDEADEV/IDEA+Coverage+Runner` for instructions.