CS 2S03: Principles of Programming

Due on Thursday September 29th, 2016

Dr. Jacques Carette

Idea

The goals of this assignment are:

1. get some practice with Java,
2. use basic TDD (see Tutorial 2),
3. deepen understanging of `if` construct, and
4. deepen understanding of `for` loops.

The Task

From the a1 assignment web page, find your student id and download the file pointed to by that link - it will be called `a1_<studentid>.zip`. It contains 6 java files and 2 comma-separated values (csv) files, called `A1Test.java`, `A2Test.java`, `A3Test.java`, `cs2s03/A1.java`, `cs2s03/A2.java`, `cs2s03/A3.java`, `A1.csv` and `A2.csv`. The first three are JUnit test files, the next three are template solutions, and the last 2 contain the same "data" as the first two JUnit files but in a more convenient csv format for experimentation.

Each occurance of `II` in `A1` needs to be replaced by some explicit *integer*. The correct integers to use will be determined (uniquely) by the tests in `A1Test.java` (see the tutorial slides on TDD for more details). The body of the function `cases` in class `A1` should consist solely of (nested) `if`s where the condition will look like `var <= integer` (with `var` one of v,u,w). The body of the innermost `if` (and `else`) will look like `return integer`. There are 3 variables, so 8 cases in total. In the usual layout, the model solution has 25 lines of code (of just conditionals, returning a single integer, or closing brace).

`A2` is like `A1` except that you need to start with an empty template. Note that the correct variable order *will* be different.

In `cs2s03/A3.java`, the problem is that the initialization of the variables is incorrect. The code itself does not need to be changed, just the initial values of the variables.

Submission Requirements

- A zip file with correct versions of A1.java, A2.java, A3.java.
- The name of the zip file does not matter. Extra files in the zip do not matter.
- The names of the files (`A1.java`, etc) **does** matter.
- Your code for `A2.java` should have *exactly* 8 `return` statements in 8 branches.
- Code which **does not compile** is worth **0** marks.

Notes

- For the first two problems, if it involved a single variable u rather than 3, and the data (in csv) you were given looked like

  ```
  4, 1
  1, 1
  0, 2
  -1, 2
  -5, 2
  ```

  then the body of your code should look like

  ```
  if (u<=0) {
      return 2;
  } else {
      return 1;
  }
  ```

  A visual represation of this example problem in one variable is given in Figure 1.

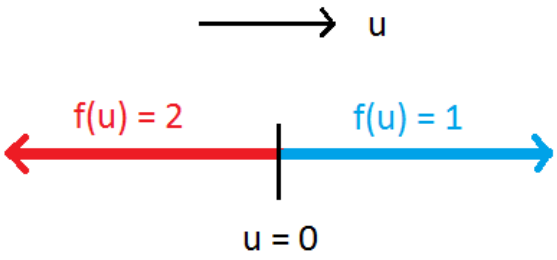

Figure 1: Visual representation of the problem in a single variable.

The problem of "dividing up space" gets much more difficult with more variables. Especially as the "breakpoints" between the cases *vary*.

- As we saw in the preceding example, the problem in one variable leads to a division of a one-dimensional space into two distinct regions. The problem in two variables would result in the division a two-dimensional space into four distinct regions (see Figure 2a). The problem in three variable (this is the case in `A1` and `A2`) results in this division of a three-dimensional space into eight distinct regions (see Figure 2b). When attempting `A1` and `A2`, you may find it useful to study the supplied data to determine the locations that the breakpoints between the divisions occur at.

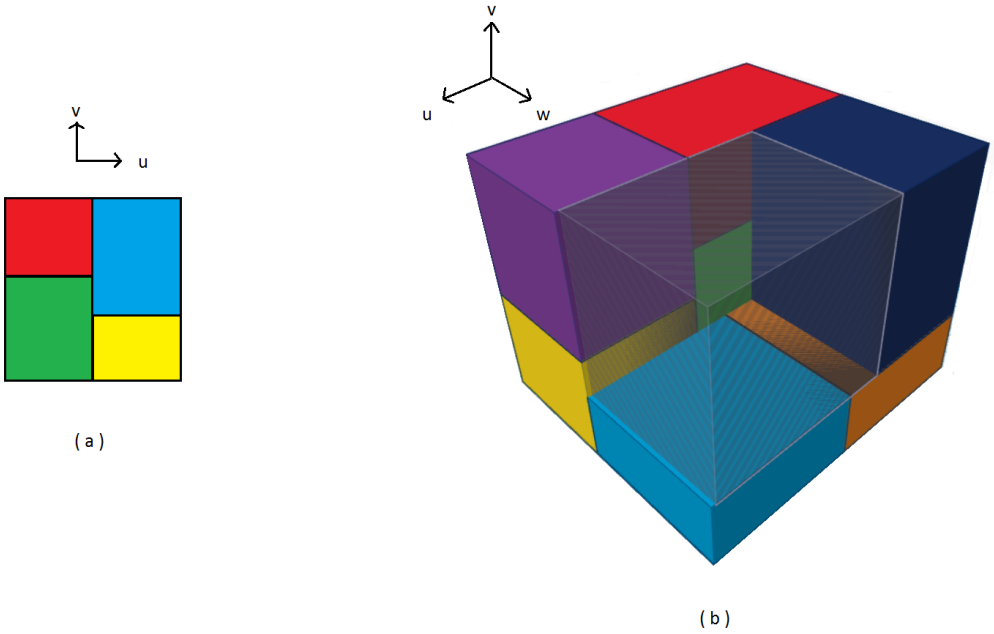

Figure 2: Visual representation of the problem in (a) two variables and (b) three variables.

- The data in the csv files and in the JUnit test cases are *the same*. The csv files are provided for your convenience, as using a tool (such as Excel, but you could just as easily use a python script, Maple, R, etc) to visualize the data can be quite useful. And this is most easily accomplished from csv data.

Bonus

1. All of the following will be worth $1-2$ extra marks. Make sure you document that you have done these.
   - For A1 and A2, **additionally** submit an alternate version which encodes the same function but as a single mathematical function rather than using `if`.
   - For A3, **additionally** submit an alternate version which computes the answer directly as a mathematical function (no loop or recursion, but powering is needed).
2. For up to 15 extra marks (i.e. 15% of your final course mark), write an *assignment 1 generator*: the input would be a text file with student ids, and the output would be one directory per id, with each directory containing:
   - `cs2s03/A1.java`, `cs2s03/A2.java`, and `cs2s03/A3.java` which are like the solutions for assignment 1, but have been randomized, like the ones for this year. Make sure variable order is also randomized.

- `A1Test.java`, `A2Test.java` and `A3Test.java` which contain enough correct tests for the above 2 files for them to be used for TDD, as for this assignment. This is the hardest part of this assignment to get right.
- Also output `A1.csv`, `A2.csv` and `A3.csv`.

You may use any programming language to write this in. If you choose to write this generator in Scala or Haskell (and it is correct), you may earn up to 10 extra marks. There will be a separate dropbox for this bonus part, with a separate deadline.

Extra marks will be given if you do **not** use template-based methods, but rather create an Abstract Syntax Tree (AST) of the generated code, and use a code pretty-printer for the results.

3. Even more marks will be given if you actually use a non language-specific AST, and write 2 or more pretty-printers for *different* languages (like python, C, Swift, Go, etc; even more for Scala, Haskell and clojure). Basically, if you get to here, you'll get enough bonus marks to pass this course merely out of assignment 1. There will be a separate dropbox for this as well.