Week 12.1 — Testing and Coverage

Testing

- Blackbox testing — can only use the spec and/or its behaviour because you don't look inside.
- Whitebox testing — look at the implementation and test based on that.

So you've made some whitebox tests

- How much of the code is actually tested?
- Of all the different ways your program could run, how many are *covered* by your tests?
- Of course, we'd like to not do more testing than we need to.

We'll look at three levels of coverage[1]:

- Statement coverage: has every statement been run?
- Branch coverage: has every possible decision option been tested?
- Path coverage: has every possible sequence of decisions been tested?

[1] See Miller and Maloney CACM 1963 "Systematic mistake analysis of digital computer programs" — for an early look at this

Statement testing

```
int x = getLargestPrime();
System.out.println(x);
```

This code can be tested by executing it once[2]. There are is only one possibile sequence apparent in the code.
Running this once would give us "statement coverage". (Each statement has been run).

[2] Assuming `getLargestPrime()` doesn't use any hidden state.

Branch[3] coverage

```
int max(int a, int b) {
    if (a < b) {
        return b;
    } else {
        return a;
    }
}
```

To ensure this code works correctly, we would need to test the decision(branch) in both true and false cases.
If we do this for both "sides" of all decisions, that would give us "branch" coverage.

[3] Branch in cpu terms is to jump to a different address

Hidden decisions

Not all branches explicitly say “if” or “switch”

```
int max(int a, int b) {
    return (a < b) ? b : a;
}
```

OR

```
int result = f(x) && g(x);
```

Branch example 1

```
int max(int a, int b, int c) {
    int m = a;
    if (m < b) {
        m = b;
    }
    if (m < c) {
        m = c;
    }
    return m;
}
```

Inputs that would give us branch coverage:
(1, 2, 3) and (3, 2, 1)

Branch example 2

```
int max(int a, int b, int c) {
    if (a < b) {
        if (b < c) {
            return c;
        } else {
            return b;
        }
    } else {
        if (a < c) {
            return c;
        } else {
            return a;
        }
    }
}
```

Inputs to get branch coverage:
(1, 2, 3) and (1, 2, 1) and (2, 1, 3) and (3, 2, 1)

Loops

Code executes in context.

```
for (init; test; iterate) {
    body
}
nextstmt;
```

- init; test (fails); nextstmt;
- init; test; body; iterate; test (fails); nextstmt;
- init; test; body; iterate; test; body; iterate; test (fails); nextstmt;

The difference between the last two could be the difference between a working iterate and a faulty one.
To get path coverage, we need inputs which will cause every possible path to be followed at least once.

Correctness?

Travelling those paths is fine

- Program doesn't crash?
- Doesn't throw unexpected exceptions?
- . . . other obvious misbehaviour?

Does it give the correct answer though?

Answers?

Where to get answers to check results?

- Answers calculated "manually" (and verified?)
- Answers generated once (and verified?)
- Call two versions (yours and a reference implementation/source of truth/sensor/...).
  - Different in that the expected answers are not explicitly stored.
  - Did they change? Did you notice?
- Call inverse functions: eg $x == arcsin(sin(x))$
  - Doesn't actually test if they are correct, only that they are consistant.

ancoverage.java