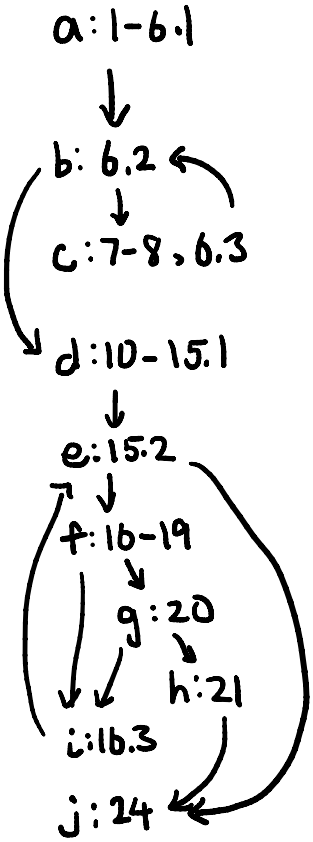
SE465 Final W2010

1.

1. Continuously updated, fast
2. False
3. [Not covered]
4. [Not covered]
5. [Not covered]
6. There can be a very large number of states
7. It is difficult to track the progress of individual bugs
8. Summary
9. [Not covered]
10. False

2.

Assume there are line numbers where int numInstances = data.numInstances(); is line 1.



*assume don't need node for the method definition

def(a) = { numInstances, trainData, weights, sumOfWeights, i }
use(a) = { data, numInstances }

def(b) = {}
use(b) = { i }

def(c) = { weights, sumOfWeights }
use(c) = { data, weights }

def(d) = { weightMassToSelect, sortedIndices, sumOfWeights, i }
use(d) = { sumOfWeights, quantile }

def(e) = {}
use(e) = { i }

def(f) = { instance, trainData, sumOfWeights }
use(f) = { data, sortedIndices, instance, weights, sumOfWeights,
weightMassToSelect, i }

def(g) = {}
use(g) = { weights, sortedIndices, i }

def(h) = use(h) = {}

def(i) = {}
use(i) = { i }

def(j) = {}
use(j) = { trainData }

3.

*excluding the for loop ones presumably (i < numInstances, i >= 0)

sumOfWeights > weightMassToSelect && weights[sortedIndices[i]] != weights[sortedIndices[i-1]]

Let A = sumOfWeights > weightMassToSelect, B = weights[sortedIndices[i]] != weights[sortedIndices[i-1]], P = A ∧ B

Test requirements (tuples are written as (A, B)
- pairs satisfying major A: [(true, true), (false, false)], [(true, true), (false, true)]
- pairs satisfying major B: [(true, true), (false, false)], [(true, true), (true, false)]

Note: pair [(true, true), (false, false)] satisfies CACC (for both majors)

Test case 1 (true, true)
```
d = new Instances();
d.add(new Instance(10.0));
d.add(new Instance(20.0));
selectWeightQuantile(d, 0.1);
```

Test case 2 (false, false)
```
d = new Instances();
d.add(new Instance(10.0));
d.add(new Instance(10.0));
selectWeightQuantile(d, 1.0);
```

4.

a: def(x); def(y);

b: use(y); def(y);

c: use(x); def(x);

PPC: (a, b, c), (b, c, b), (c, b, c)

ADUPC: (a, b), (a, b, c), (b, c, b), (c, b, c)

5.

Line 2: change < to > (relational operator replacement)
- Test input [3, 2, 1]
- Original output [1, 2, 3]
- Mutant output [3, 2, 1]

Line 7: change <= to > (relational operator replacement)
- Test input [3, 2, 1]
- Original output [1, 2, 3]
- Mutant output [2, 2, 1]

6.

Partitioning 1 (characteristic = number of points):
- Polygons with an even number of points
- Polygons with an odd number of points

Partitioning 2 (characteristic = location of points):

- Polygons containing a point at the origin
- Polygons not containing a point at the origin

Partitioning 3 (characteristic = angle between lines):

- Polygons containing at least one right angle
- Polygons containing no right angles

Base choice: (even number of points, contains point at origin, has right angle)
TRs:

- (odd number of points, contains point at origin, has right angle)
- (even number of points, does not contain point at origin, has right angle)
- (even number of points, contains point at origin, no right angles)

Test suite (base choice + in order of TRs):

- points = [(0, 0), (1, 0), (0, 1), (1, 1)]
- points = [(0, 0), (1, 0), (0, 1)]
- points = [(1, 1), (2, 1), (1, 2), (2, 2)]
- points = [(0, 0), (1, 0), (1, 1), (2, 1)]

7. We didn't learn about stubs so we only have mocks…

We probably want to mock A, B, and D, since we want to simulate the interaction and avoid interacting with other infrastructure.

Mocking A is easy, since we basically have 3 variables that can be get and set.

Mocking B is more complicated, since we have state, but also an H, which upon action we either call its bar method, or call the foo method which then calls respond. This can make the tests harder to understand and maintain.

C is an interface so we don't need to mock it.

Mocking D is also complicated since it contains a bunch of objects that it modifies in its methods.