A4

- Will be available later today
- Is a totally new assignment — by no means carefully vetted
- It’s focused (again) on testing and binary search
  - This may be boring for some of you
  - I hope that the expected learning is important enough to justify this
- One of the next assignments (likely only one more, possibly two) will be a music player that accepts a textual notation for music and produces MIDI output.

Midterms — Parts I and II graded

- Plan (hope?) to have them ready by Wednesday
- Key with comments is under production — released when the results are released.

A4 basics

- Random test generation question from midterm
  - The randomly generated array length might not be consistent with the number of values in the array
  - The randomly generated array might not be sorted
  - Random keys are much more likely to be not found than to be found.
  - There’s no way to determine the oracle.
- You’ll write a test generation program that overcomes these issues (and producesJUnit tests).
- Generate length and then values for the test array.
- Produce the randomized in a way that guarantees it is sorted — use a binary search tree (BST) to first insert the random elements and then retrieve them in sorted order.
- Randomly decide to generate (for instance) 10% found keys — and then find a key in the array or find a key not in the array.
- Voilà, an oracle appears (almost that easily).

A4 objectives include

- Deeper understanding of testing
- Representation invariant needed for BST
- Some focus on abstraction function
  - Related to visitor pattern for traversing BST to create sorted array
- Clean mind
  - Separate tests you generate from tests you need to test your program
- Separate binary search (program under test) from binary search tree (implementation mechanism for your program)
- ...

White (glass, clear)-box testing

- Goals
  - Ensure test suite covers (executes) all of the program
  - Measure quality of test suite with % coverage
- Assumptions
  - High coverage → few mistakes in the program
  - “If statement S hasn’t been executed (covered) by any test, it might cause an error.”
  - Focus on coverage, not oracles
  - Fundamentally an inadequacy property of test suites
- Focus: features not described by specification
  - Control-flow details
  - Performance optimizations
  - Alternate algorithms for different cases
White-box Motivation

- There are some subdomains that black-box testing won’t find
  ```java
  boolean[] primeTable = new boolean[CACHE_SIZE];
  boolean isPrime(int x) {
      if (x>CACHE_SIZE) {
          for (int i=2; i<x/2; i++) {
              if (x%i==0) return false;
          }
          return true;
      } else {
          return primeTable[x];
      }
  }
  ```

- Important transition around $x = \text{CACHE\_SIZE}$ that isn’t visible to black-box testing (assuming $\text{CACHE\_SIZE}$ is private)

White Box Testing: Advantages

- Finds an important class of boundaries — yields useful test cases
  - Need to check numbers on each side of $\text{CACHE\_SIZE}$
    - $\text{CACHE\_SIZE}-1$, $\text{CACHE\_SIZE}$, $\text{CACHE\_SIZE}+1$
  - If $\text{CACHE\_SIZE}$ is mutable, we may need to test with different $\text{CACHE\_SIZE}$

- Disadvantages?
  - Tests may have same bugs as implementation

Statement coverage

- Test suite
  ```java
  static int min (int a, int b) {
      int m = a;
      if (a <= b) {
          m = a;
      }
      return m;
  }
  ```

- Good: executes every instruction
- Bad: doesn’t find bug
- So, can be unsatisfying in trivial cases

Think of the program as a flow-chart

- Covering all statements would not require edge ac to be covered
- Edge coverage requires all control flow graph edges to be covered by at least one test

Edge coverage

- Does’t increase statement coverage — still 100%
- But does increase edge coverage from 75% to 100%
Is edge coverage enough?

- Consider this program and test suite (not exactly Java, but you can follow it)
- Make it into a flowchart... mark executed edges

```
if x != 0
  y = 5;
else
  z = z - x;
if z > 1
  z = z / x;
else
  z = 0;
```

```
{(x = 0, z = 1)
 (x = 1, z = 3)}
```

Edge coverage: 100%

```
if x != 0
  y = 5;
else
  z = z - x;
if z > 1
  z = z / x;
else
  z = 0;
```

```
{(x = 0, z = 1)
 (x = 1, z = 3)}
```

What is missed?

Path coverage

- Edge coverage is in some sense very static
- Edges can be covered without covering actual paths (sequences of edges) that the program may execute
- Not all paths in a program are always executable
- Loops complicate paths

Varieties of coverage

- Covering all of the program
  - Statement coverage
  - Edge (branch) coverage
  - Decision coverage (not discussed)
    - Handling compound decisions
  - Loop coverage (not discussed)
  - Condition/Decision coverage (not discussed)
  - Path coverage

- Limitations of coverage
  - 100% coverage is not always a reasonable target
  - High cost to approach the limit
  - Coverage is just a heuristic: we really want the revealing subdomains

Structural coverage: some challenges

- Interprocedural coverage
- Late binding in OO – coverage of polymorphism
- Need good tools for tracking coverage
- Higher coverage may be deceptive
- There are a family of new, automatic test generation techniques that seem to be influencing coverage-based testing

Next steps

- Assignment 4: out later today, due Wednesday November 9, 2011 at 11:59PM
- Lectures: TBA