Generation vs. Recognition

- **Generation** of tests based on coverage means producing a test suite to achieve a certain level of coverage
  - As you can imagine, generally very hard
  - Consider: generating a suite for 100% statement coverage easily reaches “solving the halting problem” level
  - Obviously hard for, say, mutant-killing
- **Recognition** means seeing what level of coverage an existing test suite reaches

Coverage and Subsumption

- Sometimes one coverage approach *subsumes* another
  - If you achieve 100% coverage of criteria A, you are guaranteed to satisfy B as well
    - For example, consider node and edge coverage
      - (there's a subtlety here, actually – can you spot it?)
- What does this mean?
  - Unfortunately, not a great deal
  - If test suite X satisfies "stronger" criteria A and test suite Y satisfies "weaker" criteria B
    - **Y may still reveal bugs that X does not!**
    - **For example, consider our running example and statement vs. branch coverage**
  - *It means we should take coverage with a grain of salt, for one thing*

Testing “for” Coverage

- Never seek to improve coverage *just for the sake of increasing coverage*
  - Well, unless it’s a command from-on-high
- Coverage is not the goal
  - Finding failures that expose faults is the goal
  - No amount of coverage will prove that the program cannot fail

“Program testing can be used to show the presence of bugs, but never to show their absence!” *– E. Dijkstra, Notes On Structured Programming*

The Purpose of Testing

"Program testing can be used to show the presence of bugs, but never to show their absence!" *– E. Dijkstra, Notes On Structured Programming*

- Dijkstra meant this as a criticism of testing and an argument in favor of more disciplined and total approaches (proving programs correct)
- But he also points out *what testing is good for:* exposing errors
- Coverage is valuable if and only if test sets with higher coverage are more likely to expose failures

The Purpose of Testing

"Program testing can be used to show the presence of bugs"

- When we first start "testing," we often want to "see that the program works"
  - Try out some scenarios and watch the program "do its stuff"
  - Surprised (annoyed) when (if) the program fails
  - *This is not really testing*: *testing is not the same as a demonstration*
  - Aim to break (your) code, if it can be broken

What’s So Good About Coverage?

- Consider a fault that causes failure *every time the code is executed*
- Don’t execute the code: cannot possibly find the fault!
- That’s a pretty good argument for statement coverage

```
int findLast (int a[], int n, int x) {
    // Returns index of last element
    // in a equal to x, or -1 if no
    // such.  n is length of a

   int i;
   for (i = n-1; i >= 0; i--) {
      if (a[i] = x)
        return i;
    }
    return 0;
}
```

What’s So Good About Coverage?

- We should have an *argument* for any kind of coverage:
  - “If I don’t cover *this*, then there is more chance I’ll miss a fault like *that*”
  - Backed with empirical data, preferably!

```
int findLast (int a[], int n, int x) {
    // Returns index of last element
    // in a equal to x, or -1 if no
    // such.  n is length of a

   int i;
   for (i = n-1; i >= 0; i--) {
      if (a[i] = x)
        return i;
    }
    return 0;
}
```

Using gcov to Collect Coverage

- GCC comes with a tool for collecting and analyzing coverage, called gcov

- Compile with some additional items:
  - -ftest-coverage -fprofile-arcs

- When the executable runs, it will produce files (gcda files) that record how often each line ran

Using gcov to Collect Coverage

- To look at the coverage, type:
  - gcov <sourcefile>
  - Will show % coverage, and produce <sourcefile>.gcov, annotated copy of code

- Can also do branch coverage:
  - gcov –b <sourcefile>

- Makefiles from this class automatically compile with gcov

Using gcov to Collect Coverage

- Important points:
  - If you compile with optimization, results may be strange – try -O0
  - If you haven’t run the program, gcov <sourcefile> won’t do anything! It has no coverage data
  - The number of times a line/branch runs can be helpful, in addition to looking for “####” to indicate things that are never covered at all
    - `grep ‘####’ filename.c.gcov`