Software Testing Lecture 4

Justin Pearson

November 18, 2013
```c
int count_spaces(char* str) {
    int length, i, count;
    count = 0;
    length = strlen(str);
    for(i=1; i<length; i++) {
        if(str[i] == ' '){ count++;
    }
    }
}
```
First Divide into Basic Blocks

```c
int count = 0;
length = strlen(str);

for(i=1; i<length; i++)
    if(str[i] == ' ')
        count++;

return(count);
```
int count = 0;
length = strlen(str);

for (i = 1; i < length; i++)
    count++;

if (str[i] == ' ')
    return(count);
Test Path

- Remember a test path is a path that starts at an entry node and leaves at an exit node.
Node Coverage

```c
int count = 0;
length = strlen(str);

for(i=1; i<length; i++)
    count++;

if(str[i] == ' ')
    return(count);

TR = {1, 2, 3, 4, 5, 6, 7},
Test path is
[1, 2, 3, 4, 5, 6, 3, 7]`
Grey box testing

Our test path [1, 2, 3, 4, 5, 6, 3, 7] requires the loop to execute exactly once and it to detect one space. So we might try the test case (" ",1) but this won’t work. Don’t forget that i=1 in the loop body.

Instead we have to use the test case ("H ",1)

By thinking about what the code should do, and trying to construct a test case corresponding to a path, we have uncovered a fault.
Edge Coverage

```c
int count = 0;
length = strlen(str);
for(i=1; i<length; i++)
    count++;
return(count);
```

\[ TR = \{(1, 2), (2, 3), (3, 4), (3, 7), (4, 5), (4, 6), (5, 6), (6, 3)\} \]

Test paths are

\[ [1, 2, 3, 4, 5, 6, 3, 7], [1, 2, 3, 4, 6, 3, 7], [1, 2, 3, 7] \]
Test Cases

- [1, 2, 3, 4, 5, 6, 3, 7] (" ", 1)
- [1, 2, 3, 4, 6, 7] ("H", 0)
- [1, 2, 3, 7] (" ", 0)
Relaxing test cases

- As we have seen, sometimes we have infeasible test cases.
  - This could because there is a fault.
  - Or, that we have to do other things to get to the code. There might be a bit of setup code that we have to call first that is not in our path.
- Before we introduced the notion of a path touring another path.
- A path $p$ tours the path $s$ if $s$ is a sub-sequence of $p$.
  - $[1, 2, 3, 4, 6, 3, 4, 6, 3, 7]$ tours the test path $[1, 2, 3, 4, 6, 3]$ it also tours many other paths including $[4, 6, 3, 7]$.
  - Don’t forget the difference between a test path and a path.
Relaxing Test Cases

- A test path $p$ is set to tour sub-path $q$ with side-trips if every edge that is in $q$ is also in $p$ in the same order.
- A test path $p$ is set to tour sub-path $q$ with detours if every node that is in $q$ is also in $p$ in the same order.
The path [0, 1, 2, 3, 2, 4, 5] tours the path [0, 1, 2, 4, 5] with side trips.
The path \([0, 1, 2, 3, 4, 5]\) tours the path \([0, 1, 2, 4, 5]\) with side trips.
Infeasible test requirements

- An infeasible test requirement *cannot be satisfied*
  - Unreachable statement (dead code)
  - Can only be executed if a contradiction occurs \( X > 0 \land X < 0 \).
  - Always check against the specification, it could be a fault.
Infeasible test requirements

- Most test criteria have some infeasible test requirements.
- It is usually undecidable to decide if all test requirements are feasible (halting problem again).
- Allowing side trips might weaken the test cases, but allows more feasible test cases.
- Practical recommendation, best effort touring. Allow as many as possible without side-trips, only allow side-trips on infeasible test paths.
Loops

- If a graph contains a loop then it has an infinite number of paths.
- Thus you cannot ask for complete path coverage.
- Attempts to deal with loops:
  - 1970s: Execute cycles once ([4, 5, 4] in previous example, informal)
  - 1980s: Execute each loop, exactly once (formalised)
  - 1990s: Execute loops 0 times, once, more than once (informal description)
  - 2000s: Prime paths
Simple and Prime Paths

- A path is simple if no node appears more than once except possible that the first and last node can be the same.
- A prime path of a graph is a simple path that is not a sub-path of any other simple path.
Simple Paths

- [0],[1],[2],[3]
- [0, 1],[0, 2],[1, 3],[2, 3],[3, 0]
- [0, 1, 3], [0, 2, 3], [1, 3, 0], [2, 3, 0],[3, 0, 1]
- [0, 1, 3, 0], [0, 2, 3, 0], [1, 3, 0, 1], [2, 3, 0, 2], [3, 0, 1, 3], [3, 0, 2, 3], [1, 3, 0, 2], [2, 3, 0, 1].
Prime Paths

Remove all simple paths that can be extended (either direction) to a longer simple path.

- [0], [1], [2], [3]
- [0, 1], [0, 2], [1, 3], [2, 3], [3, 0]
- [0, 1, 3], [0, 2, 3], [1, 3, 0], [2, 3, 0], [3, 0, 1]
- [0, 1, 3, 0], [0, 2, 3, 0], [1, 3, 0, 1], [2, 3, 0, 2], [3, 0, 1, 3], [3, 0, 2, 3], [1, 3, 0, 2], [2, 3, 0, 1]

In this case the prime paths are all the longest simple paths. Not always the case.
Simple Paths

Enumerate all simple paths of length, 1, 2, 3, ... then remove simple paths that can be extended. You will be left with the prime paths.

- [1], [2], [3], [4]
- [1, 2], [2, 3], [2, 4], [3, 2]
- [1, 2, 3], [1, 2, 4], [2, 3, 2]
- We have to be careful about the paths of length 4.
  - [1, 2, 3, 2] is not a simple path. Repeats 2 which is not at the beginning or the end.
- In fact there are no simple paths of length 4 in this graph.
Enumerate all simple paths of length, 1, 2, 3, ... then remove simple paths that can be extended. You will be left with the prime paths.

- $x = 0$
- $i++$
- body

- $[1], [2], [3], [4]$
- $[1, 2], [2, 3], [2, 4], [3, 2]$
- $[1, 2, 3], [1, 2, 4], [2, 3, 2]$
Prime Paths to Test Paths

\[ x = 0 \]

\[ \text{body} \]

- \([1, 2, 3] \rightarrow [1, 2, 3, 4]\)
  - Execute loop once.
- \([1, 2, 4] \rightarrow [1, 2, 4]\)
  - Execute loop zero times.
- \([2, 3, 2] \rightarrow [1, 2, 3, 2, 4]\)
  - Execute loop more than once.
Simple Paths

- [1], [2], [3], [4], [5], [6], [7]!
- [1, 2], [2, 3], [3, 4], [3, 7], [4, 5], [4, 6], [5, 6], [6, 3], [6, 3]
- [1, 2, 3], [2, 3, 4], [2, 3, 7]!, [3, 4, 5], [3, 4, 6], [4, 5, 6], [4, 6, 3], [4, 6, 3], [5, 6, 3], [5, 6, 3], [6, 3, 4]
- [1, 2, 3, 4], [1, 2, 3, 7]!, [2, 3, 4, 5], [2, 3, 4, 6], [3, 4, 5, 6], [3, 4, 6, 3], [3, 4, 6, 3], [4, 5, 6, 3], [4, 5, 6, 3], [5, 6, 3, 4]
- [1, 2, 3, 4, 5], [1, 2, 3, 4, 6], [2, 3, 4, 5, 6], [2, 3, 4, 6, 3], [3, 4, 5, 6, 3], [3, 4, 5, 6, 3], [3, 4, 5, 6, 3]
Prime Paths

- $[1], [2], [3], [4], [5], [6], [7]$
- $[1, 2], [2, 3], [3, 4], [3, 7], [4, 5], [4, 6], [5, 6], [6, 3], [6, 3]$
- $[1, 2, 3], [2, 3, 4], [2, 3, 7], [3, 4, 5], [3, 4, 6], [4, 5, 6], [4, 6, 3], [4, 6, 3], [5, 6, 3], [5, 6, 3], [6, 3, 4]$
- $[1, 2, 3, 4], [1, 2, 3, 7], [2, 3, 4, 5], [2, 3, 4, 6], [3, 4, 5, 6], [3, 4, 6, 3], [4, 5, 6, 3], [6, 3, 4, 5], [4, 6, 3, 4], [5, 6, 3, 4]$
- $[1, 2, 3, 4, 5], [1, 2, 3, 4, 6], [2, 3, 4, 5, 6], [3, 4, 5, 6, 3]$
- $[1, 2, 3, 4, 5, 6]$. 
Prime Paths

- $[1, 2, 3, 7]$! $\rightarrow$ $[1, 2, 3, 7]$  
  - Do the loop zero times.

- $[3, 4, 6, 3] \rightarrow [1, 2, 3, 4, 6, 3, 7]$  
  - Do the loop once and do not do the if

- $[6, 3, 4, 5] \rightarrow [1, 2, 3, 4, 6, 3, 4, 5, 6, 3, 7]$  
  - Do the loop twice, once with the if and once without.

- $[4, 6, 3, 4] \rightarrow [1, 2, 3, 4, 6, 3, 4, 6, 3, 7]$  
  - Do the loop twice, both times without taking the if.

- $[5, 6, 3, 4] \rightarrow [1, 2, 3, 4, 5, 6, 3, 4, 6, 3, 7]$  
  - Do the loop twice, take the if once and once without, other way around from the previous case.

- $[3, 4, 5, 6, 3] \rightarrow [1, 2, 3, 4, 5, 6, 3, 7]$. 
Prime Paths: Summary

- Prime paths give you a good way of deriving a set of test cases that cover various combinations of loops and branches.
- There is no formal guarantee about completeness. As in all testing it just formalises a good compromise.

Please check my slides for mistakes.