Overview

- The most common application of graph criteria is to program source
- **Graph**: Usually the control flow graph (CFG)
- **Node coverage**: Execute every statement
- **Edge coverage**: Execute every branch
- **Loops**: Looping structures such as for loops, while loops, etc.
- **Data flow coverage**: Augment the CFG
  - **defs** are statements that assign values to variables
  - **uses** are statements that use variables
Control Flow Graphs

- A CFG models all executions of a method by describing control structures
- **Nodes**: Statements or sequences of statements (basic blocks)
- **Edges**: Transfers of control
- **Basic Block**: A sequence of statements such that if the first statement is executed, all statements will be (no branches)

- CFGs are sometimes annotated with extra information
  - branch predicates
  - defs
  - uses

- Rules for translating statements into graphs …
cfg : the if statement

if (x < y) {
    y = 0;
    x = x + 1;
} else {
    x = y;
}

if (x < y) {
    y = 0;
    x = x + 1;
} else {
    x = y;
}
CFG: The if-Return Statement

```java
if (x < y) {
    return;
}
print (x);
return;
```

No edge from node 2 to 3. The return nodes must be distinct.
Loops

- Loops require “extra” nodes to be added
- Nodes that do not represent statements or basic blocks
CFG: while and for Loops

\[ x = 0; \]
\[ \text{while} \ (x < y) \ { \}
\[ \quad y = f(x, y); \]
\[ \quad x = x + 1; \}
\[ } \]

\[ \text{for} \ (x = 0; x < y; x++) \ { \}
\[ \quad y = f(x, y); \]
\[ } \]
CFG : The case (switch) Structure

```c
read ( c ) ;
switch ( c )
{
    case ‘N’:
        y = 25;
        break;
    case ‘Y’:
        y = 50;
        break;
    default:
        y = 0;
        break;
}
print (y);
```

![CFG diagram]
public static void computeStats (int [] numbers)
{
    int length = numbers.length;
    double med, var, sd, mean, sum, varsum;

    sum = 0;
    for (int i = 0; i < length; i++)
    {
        sum += numbers [ i ];
    }
    med   = numbers [ length / 2 ];
    mean = sum / (double) length;

    varsum = 0;
    for (int i = 0; i < length; i++)
    {
        varsum = varsum + ((numbers [ I ] - mean) * (numbers [ I ] - mean));
    }
    var = varsum / ( length - 1.0 );
    sd  = Math.sqrt ( var );

    System.out.println ("length:                   " + length);
    System.out.println ("mean:                    " + mean);
    System.out.println ("median:                  " + med);
    System.out.println ("variance:                 " + var);
    System.out.println ("standard deviation: " + sd);
}
public static void computeStats (int [] numbers) {
    int length = numbers.length;
    double med, var, sd, mean, sum, varsum;
    sum = 0;
    for (int i = 0; i < length; i++) {
        sum += numbers [i];
    }
    med = numbers[length / 2];
    mean = sum / (double) length;
    varsum = 0;
    for (int i = 0; i < length; i++) {
        varsum = varsum + ((numbers[i] - mean) * (numbers[i] - mean));
    }
    var = varsum / (length - 1.0);
    sd = Math.sqrt(var);
    System.out.println("length: " + length);
    System.out.println("mean: " + mean);
    System.out.println("median: " + med);
    System.out.println("variance: " + var);
    System.out.println("standard deviation: " + sd);
}
Control Flow TRs and Test Paths – EC

- TR A: [1, 2]
- TR B: [2, 3]
- TR C: [3, 4]
- TR D: [3, 5]
- TR E: [4, 3]
- TR F: [5, 6]
- TR G: [6, 7]
- TR H: [6, 8]
- TR I: [7, 6]

Test Path: [1, 2, 3, 4, 3, 5, 6, 7, 6, 8]
Control Flow TRs and Test Paths – EPC

TRs toured

- A, B, D, E, F, G, I, J
- A, C, E, H
- A, B, C, D, E, F, G, I, J, K, L

sidetrips

- C, H
- H

Edge-Pair Coverage

<table>
<thead>
<tr>
<th>TR</th>
<th>Test Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. [1, 2, 3]</td>
<td>i. [1, 2, 3, 4, 3, 5, 6, 7, 6, 8]</td>
</tr>
<tr>
<td>B. [2, 3, 4]</td>
<td>ii. [1, 2, 3, 5, 6, 8]</td>
</tr>
<tr>
<td>C. [2, 3, 5]</td>
<td>iii. [1, 2, 3, 4, 3, 4, 3, 5, 6, 7, 6, 7, 6, 8]</td>
</tr>
<tr>
<td>D. [3, 4, 3]</td>
<td></td>
</tr>
<tr>
<td>E. [3, 5, 6]</td>
<td></td>
</tr>
<tr>
<td>F. [4, 3, 5]</td>
<td></td>
</tr>
<tr>
<td>G. [5, 6, 7]</td>
<td></td>
</tr>
<tr>
<td>H. [5, 6, 8]</td>
<td></td>
</tr>
<tr>
<td>I. [6, 7, 6]</td>
<td></td>
</tr>
<tr>
<td>J. [7, 6, 8]</td>
<td></td>
</tr>
<tr>
<td>K. [4, 3, 4]</td>
<td></td>
</tr>
<tr>
<td>L. [7, 6, 7]</td>
<td></td>
</tr>
</tbody>
</table>
Control Flow TRs and Test Paths – PPC

Prime Path Coverage

<table>
<thead>
<tr>
<th>TR</th>
<th>Test Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.  [3, 4, 3]</td>
<td>i.  [1, 2, 3, 4, 3, 5, 6, 7, 6, 8]</td>
</tr>
<tr>
<td>B.  [4, 3, 4]</td>
<td>ii. [1, 2, 3, 4, 3, 4, 3, 5, 6, 7, 6, 7, 6, 8]</td>
</tr>
<tr>
<td>C.  [7, 6, 7]</td>
<td>iii. [1, 2, 3, 4, 3, 5, 6, 8]</td>
</tr>
<tr>
<td>D.  [7, 6, 8]</td>
<td>iv.  [1, 2, 3, 5, 6, 7, 6, 8]</td>
</tr>
<tr>
<td>E.  [6, 7, 6]</td>
<td>v.   [1, 2, 3, 5, 6, 8]</td>
</tr>
<tr>
<td>F.  [1, 2, 3, 4]</td>
<td></td>
</tr>
<tr>
<td>G.  [4, 3, 5, 6, 7]</td>
<td></td>
</tr>
<tr>
<td>H.  [4, 3, 5, 6, 8]</td>
<td></td>
</tr>
<tr>
<td>I.  [1, 2, 3, 5, 6, 7]</td>
<td></td>
</tr>
<tr>
<td>J.  [1, 2, 3, 5, 6, 8]</td>
<td></td>
</tr>
</tbody>
</table>

TP | TRs toured | sidetrips |
---|------------|-----------|
| i | A, D, E, F, G | H, I, J |
| ii | A, B, C, D, E, F, G, | H, I, J |
| iii | A, F, H | J |
| iv | D, E, F, I | J |
| v | J | |
Data Flow Coverage for Source

• **def**: a location where a value is stored into memory
  – x appears on the left side of an assignment (x = 44;)
  – x is an actual parameter in a call and the method changes its value
  – x is a formal parameter of a method (implicit def when method starts)
  – x is an input to a program

• **use**: a location where variable’s value is accessed
  – x appears on the right side of an assignment
  – x appears in a conditional test
  – x is an actual parameter to a method
  – x is an output of the program
  – x is an output of a method in a return statement

• If a def and a use appear on the same node, then it is only a DU-pair if the def occurs after the use and the node is in a loop
Example Data Flow – Stats

class ComputeStats {
    // Computes statistics for a given array of numbers.
    public static void computeStats (int[] numbers) {
        int length = numbers.length;
        double med, var, sd, mean, sum, varsum;

        sum = 0;
        for (int i = 0; i < length; i++) {
            sum += numbers[i];
        }
        med = numbers[length / 2];
        mean = sum / (double) length;

        varsum = 0;
        for (int i = 0; i < length; i++) {
            varsum = varsum + ((numbers[i] - mean) * (numbers[i] - mean));
        }
        var = varsum / (length - 1.0);
        sd = Math.sqrt(var);

        System.out.println("length: " + length);
        System.out.println("mean: " + mean);
        System.out.println("median: " + med);
        System.out.println("variance: " + var);
        System.out.println("standard deviation: " + sd);
    }
}

// Example usage:
int[] numbers = {1, 2, 3, 4, 5};
computeStats(numbers);
Control Flow Graph for Stats

1. `(numbers)`
   - `sum = 0`
   - `length = numbers.length`

2. `i = 0`

3. `i >= length`
   - `i < length`

4. `sum += numbers[i]`
   - `i++`

5. `med = numbers[length/2]`
   - `mean = sum / (double) length;`
   - `varsum = 0`
   - `i = 0`

6. `i >= length`
   - `i < length`

7. `varsum = …`
   - `i++`

8. `var = varsum / (length - 1.0)`
   - `sd = Math.sqrt(var)`
   - `print(length, mean, med, var, sd)`
CFG for Stats – With Defs & Uses

1. def (1) = { numbers, sum, length }

2. def (2) = { i }

3. use (3, 5) = { i, length }
   use (3, 4) = { i, length }

4. def (4) = { sum, i }
   use (4) = { sum, numbers, i }

5. def (5) = { med, mean, varsum, i }
   use (5) = { numbers, length, sum }

6. use (6, 8) = { i, length }
   use (6, 7) = { i, length }

7. def (7) = { varsum, i }
   use (7) = { varsum, numbers, i, mean }

8. def (8) = { var, sd }
   use (8) = { varsum, length, mean, med, var, sd }
### Defs and Uses Tables for Stats

<table>
<thead>
<tr>
<th>Node</th>
<th>Def</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{ numbers, sum, length }</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>{ i }</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>{ sum, i }</td>
<td>{ numbers, i, sum }</td>
</tr>
<tr>
<td>5</td>
<td>{ med, mean, varsum, i }</td>
<td>{ numbers, length, sum }</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>{ varsum, i }</td>
<td>{ varsum, numbers, i, mean }</td>
</tr>
<tr>
<td>8</td>
<td>{ var, sd }</td>
<td>{ varsum, length, var, mean, med, var, sd }</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Edge</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 2)</td>
<td></td>
</tr>
<tr>
<td>(2, 3)</td>
<td></td>
</tr>
<tr>
<td>(3, 4)</td>
<td>{ i, length }</td>
</tr>
<tr>
<td>(4, 3)</td>
<td></td>
</tr>
<tr>
<td>(3, 5)</td>
<td>{ i, length }</td>
</tr>
<tr>
<td>(5, 6)</td>
<td></td>
</tr>
<tr>
<td>(6, 7)</td>
<td>{ i, length }</td>
</tr>
<tr>
<td>(7, 6)</td>
<td></td>
</tr>
<tr>
<td>(6, 8)</td>
<td>{ i, length }</td>
</tr>
</tbody>
</table>
**DU Pairs for Stats**

<table>
<thead>
<tr>
<th>variable</th>
<th>DU Pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>numbers</td>
<td>(1, 4) (1, 5) (1, 7)</td>
</tr>
<tr>
<td>length</td>
<td>(1, 5) (1, 8) (1, (3,4)) (1, (3,5)) (1, (6,7)) (1, (6,8))</td>
</tr>
<tr>
<td>med</td>
<td>(5, 8)</td>
</tr>
<tr>
<td>var</td>
<td>(8, 8)</td>
</tr>
<tr>
<td>sd</td>
<td>(8, 8)</td>
</tr>
<tr>
<td>mean</td>
<td>(5, 7) (5, 8)</td>
</tr>
<tr>
<td>sum</td>
<td>(1, 4) (1, 5) (4, 4) (4, 5)</td>
</tr>
<tr>
<td>varsum</td>
<td>(5, 7) (5, 8) (7, 7) (7, 8)</td>
</tr>
<tr>
<td>i</td>
<td>(2, 4) (2, (3,4)) (2, (3,5)) (2, 7) (2, (6,7)) (2, (6,8)) (4, 4) (4, (3,4)) (4, (3,5)) (4, 7) (4, (6,7)) (4, (6,8)) (5, 7) (5, (6,7)) (5, (6,8)) (7, 7) (7, (6,7)) (7, (6,8))</td>
</tr>
</tbody>
</table>

**defs come before uses, do not count as DU pairs**

**defs after use in loop, these are valid DU pairs**

**No def-clear path … different scope for i**

**No path through graph from nodes 5 and 7 to 4 or 3**
## DU Paths for Stats

<table>
<thead>
<tr>
<th>variable</th>
<th>DU Pairs</th>
<th>DU Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>numbers</td>
<td>(1, 4)</td>
<td>[1, 2, 3, 4]</td>
</tr>
<tr>
<td></td>
<td>(1, 5)</td>
<td>[1, 2, 3, 5]</td>
</tr>
<tr>
<td></td>
<td>(1, 7)</td>
<td>[1, 2, 3, 5, 6, 7]</td>
</tr>
<tr>
<td>length</td>
<td>(1, 5)</td>
<td>[1, 2, 3, 5]</td>
</tr>
<tr>
<td></td>
<td>(1, 8)</td>
<td>[1, 2, 3, 4, 6, 8]</td>
</tr>
<tr>
<td></td>
<td>(1, (3,4))</td>
<td>[1, 2, 3, 4]</td>
</tr>
<tr>
<td></td>
<td>(1, (3,5))</td>
<td>[1, 2, 3, 5]</td>
</tr>
<tr>
<td></td>
<td>(1, (6,7))</td>
<td>[1, 2, 3, 5, 6, 7]</td>
</tr>
<tr>
<td></td>
<td>(1, (6,8))</td>
<td>[1, 2, 3, 5, 6, 8]</td>
</tr>
<tr>
<td>med</td>
<td>(5, 8)</td>
<td>[5, 6, 8]</td>
</tr>
<tr>
<td>var</td>
<td>(8, 8)</td>
<td>No path needed</td>
</tr>
<tr>
<td>sd</td>
<td>(8, 8)</td>
<td>No path needed</td>
</tr>
<tr>
<td>sum</td>
<td>(1, 4)</td>
<td>[1, 2, 3, 4]</td>
</tr>
<tr>
<td></td>
<td>(1, 5)</td>
<td>[1, 2, 3, 5]</td>
</tr>
<tr>
<td></td>
<td>(4, 4)</td>
<td>[4, 3, 4]</td>
</tr>
<tr>
<td></td>
<td>(4, 5)</td>
<td>[4, 3, 5]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>variable</th>
<th>DU Pairs</th>
<th>DU Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>(5, 7)</td>
<td>[5, 6, 7]</td>
</tr>
<tr>
<td></td>
<td>(5, 8)</td>
<td>[5, 6, 8]</td>
</tr>
<tr>
<td>varsum</td>
<td>(5, 7)</td>
<td>[5, 6, 7]</td>
</tr>
<tr>
<td></td>
<td>(5, 8)</td>
<td>[5, 6, 8]</td>
</tr>
<tr>
<td></td>
<td>(7, 7)</td>
<td>[7, 6, 7]</td>
</tr>
<tr>
<td></td>
<td>(7, 8)</td>
<td>[7, 6, 8]</td>
</tr>
<tr>
<td>i</td>
<td>(2, 4)</td>
<td>[2, 3, 4]</td>
</tr>
<tr>
<td></td>
<td>(2, (3,4))</td>
<td>[2, 3, 4]</td>
</tr>
<tr>
<td></td>
<td>(2, (3,5))</td>
<td>[2, 3, 5]</td>
</tr>
<tr>
<td></td>
<td>(4, 4)</td>
<td>[4, 3, 4]</td>
</tr>
<tr>
<td></td>
<td>(4, (3,4))</td>
<td>[4, 3, 4]</td>
</tr>
<tr>
<td></td>
<td>(4, (3,5))</td>
<td>[4, 3, 5]</td>
</tr>
<tr>
<td></td>
<td>(5, 7)</td>
<td>[5, 6, 7]</td>
</tr>
<tr>
<td></td>
<td>(5, (6,7))</td>
<td>[5, 6, 7]</td>
</tr>
<tr>
<td></td>
<td>(5, (6,8))</td>
<td>[5, 6, 8]</td>
</tr>
<tr>
<td></td>
<td>(7, 7)</td>
<td>[7, 6, 7]</td>
</tr>
<tr>
<td></td>
<td>(7, (6,7))</td>
<td>[7, 6, 7]</td>
</tr>
<tr>
<td></td>
<td>(7, (6,8))</td>
<td>[7, 6, 8]</td>
</tr>
</tbody>
</table>
DU Paths for Stats – No Duplicates

There are 38 DU paths for Stats, but only 12 unique

[ 1, 2, 3, 4 ]  [ 4, 3, 4 ]
[ 1, 2, 3, 5 ]  [ 4, 3, 5 ]
[ 1, 2, 3, 5, 6, 7 ]  [ 5, 6, 7 ]
[ 1, 2, 3, 5, 6, 8 ]  [ 5, 6, 8 ]
[ 2, 3, 4 ]  [ 7, 6, 7 ]
[ 2, 3, 5 ]  [ 7, 6, 8 ]

5 expect a loop not to be “entered”

5 require at least one iteration of a loop

2 require at least two iteration of a loop
Test Cases and Test Paths

Test Case : numbers = (44) ; length = 1
Test Path : [ 1, 2, 3, 4, 3, 5, 6, 7, 6, 8 ]
Additional DU Paths covered (no sidetrips)
[ 1, 2, 3, 4 ] [ 2, 3, 4 ] [ 4, 3, 5 ] [ 5, 6, 7 ] [ 7, 6, 8 ]
The five stars ✭ that require at least one iteration of a loop

Test Case : numbers = (2, 10, 15) ; length = 3
Test Path : [ 1, 2, 3, 4, 3, 4, 3, 4, 3, 5, 6, 7, 6, 7, 6, 7, 6, 8 ]
DU Paths covered (no sidetrips)
[ 4, 3, 4 ] [ 7, 6, 7 ]
The two stars ✭ that require at least two iterations of a loop

Other DU paths ★ require arrays with length 0 to skip loops
But the method fails with divide by zero on the statement …
mean = sum / (double) length;

A fault was found
Summary

- Applying the graph test criteria to control flow graphs is relatively straightforward
  - Most of the developmental research work was done with CFGs

- A few subtle decisions must be made to translate control structures into the graph

- Some tools will assign each statement to a unique node
  - These slides and the book uses basic blocks
  - Coverage is the same, although the bookkeeping will differ