Overview

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

Draw the graph. Label the edges with the Java statements.

Draw the graph and label the edges.
if (x < y)
{
    return;
}
print (x); return;

Draw the graph and label the edges.

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
x = 0;
while (x < y) {
  y = f (x, y);
  x = x + 1;
}
return (x);

for (x = 0; x < y; x++) {
  y = f (x, y);
}
return (x);
x = 0;
do {
    y = f(x, y);
    x = x + 1;
} while (x < y);
return (y);

x = 0;
while (x < y) {
    y = f(x, y);
    if (y == 0) {
        break;
    } else if (y < 0) {
        y = y*2;
        continue;
    }
    x = x + 1;
} return (y);

Draw the graph and label the edges.
read ( c ) ;
switch ( c )
{
    case ‘N’:
        z = 25;
    case ‘Y’:
        x = 50;
        break;
    default:
        x = 0;
        break;
}
print (x);

Draw the graph and label the edges.

Cases without breaks fall through to the next case
try
{
    s = br.readLine();
    if (s.length() > 96)
        throw new Exception("too long");
    if (s.length() == 0)
        throw new Exception("too short");
} (catch IOException e) {
    e.printStackTrace();
} (catch Exception e) {
    e.getMessage();
}
return (s);
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);
}
Write down the TRs for EC.

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

Write down test paths that tour all edges.

[1, 2, 3, 4, 3, 5, 6, 7, 6, 8]
Write down the TRs for EPC.

A. [1, 2, 3]
B. [2, 3, 4]
C. [2, 3, 5]
D. [3, 4, 3]
E. [3, 5, 6]
F. [4, 3, 5]
G. [5, 6, 7]
H. [5, 6, 8]
I. [6, 7, 6]
J. [7, 6, 8]
K. [4, 3, 4]
L. [7, 6, 7]

Write down test paths that tour all edge pairs.

i. [1, 2, 3, 4, 3, 5, 6, 7, 6, 8]
ii. [1, 2, 3, 5, 6, 8]
iii. [1, 2, 3, 4, 3, 4, 3, 5, 6, 7, 6, 7, 6, 8]

Write down test paths that tour all edge pairs.

TP | TRs toured | sidetrips
---|-------------|----------

i  | A, B, D, E, F, C, I, J | C, H

ii | A, C, E, H |

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

TP iii makes TP i redundant. A minimal set of TPs is cheaper.
Write down the TRs for PPC.

Write down test paths that tour all prime paths.

<table>
<thead>
<tr>
<th>TR</th>
<th>Test Path</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, 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 ii makes TP i redundant.
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
```java
public static void computeStats(int[] numbers)
{
    int length = numbers.length;
    double med, var, sd, mean, sum, varsum;

    sum = 0.0;
    for (int i = 0; i < length; i++)
    {
        sum += numbers[i];
    }
    med = numbers[length / 2];
    mean = sum / (double) length;

    varsum = 0.0;
    for (int i = 0; i < length; i++)
    {
        varsum = varsum + ((numbers[i] - mean) * (numbers[i] - mean));
    }
    var = varsum / (length - 1);
    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 Graph for Stats

Annotate with the statements ...

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

2. i = 0

3. i >= length

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

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

6. i >= length

7. i < length
   varsum = ...
   i++

8. i >= length
   var = varsum / (length - 1.0)
   sd = Math.sqrt(var)
   print(length, mean, med, var, sd)
Turn the annotations into def and use sets ...

**CFG for Stats – with defs and uses**

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

2. `def (2) = { i }`

3. `use (3, 4) = { i, length }`
   `use (3, 5) = { 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, 7) = { i, length }`
   `use (6, 8) = { 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 }`
# Def 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>{ numbers }</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>
Summary

Applying the graph test criteria to control flow graph 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 use basic blocks
- Coverage is the same, although the bookkeeping will differ