Graph coverage criteria: Applied to test code

.

Graph coverage criteria: Overview

- Model software artifacts as graphs and look at coverage criteria over graphs.
- Three kinds of criteria:
  - Structural coverage criteria.
  - Data flow coverage criteria.
  - Coverage criteria over call graphs.
- Focus of this lecture: Using structural graph coverage criteria to test source code.

Structural graph coverage criteria: Code

Steps to be followed:

1. Modelling control flow in code as graphs.
    - Understand the notion of basic blocks.
    - Modelling branching, looping etc. in code as graphs.
2. Using structural coverage criteria to test control flow in code.

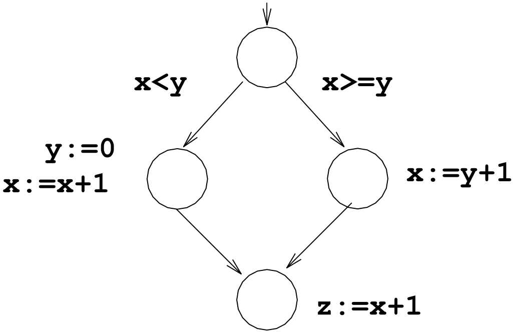
Typically used to test a particular function or procedure or a method.

Control flow graphs for code

- A Control Flow Graph (CFG) models all executions of a method by describing control structures.
- Nodes: Statements or sequences of statements (basic blocks).
- Basic Block: A sequence of statements such that if the first statement is executed, all statements will be (no branches).
- Edges: Transfer of control from one statement to the next.
- CFGs are often annotated with extra information to model data:
  - Branch predicates.
  - Defs and/or uses.

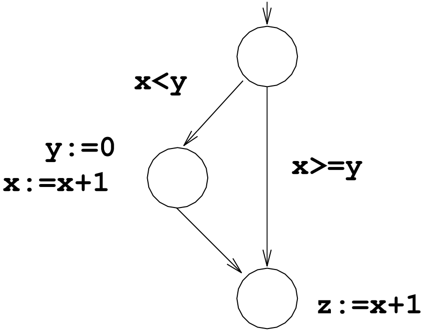
CFG: If statement

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



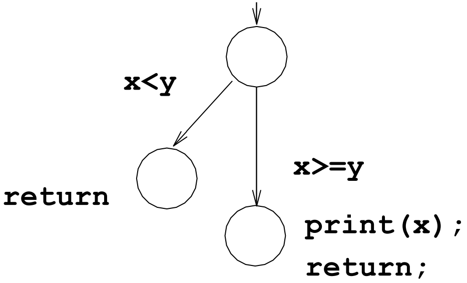
CFG: If statement

```
if (x<y)
{
   y:=0;
   x:=x+1;
}
z:=x+1;
```



CFG: If statement with return

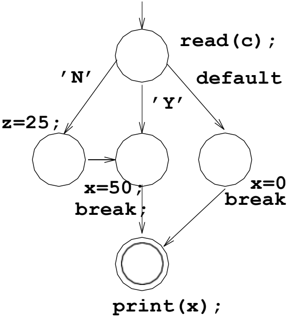
```
if (x<y)
{
  return
}
print(x);
return;
```



Note: There are two nodes corresponding to the two `return` statements.

CFG: Switch-case

```
read(c);
switch(c)
{
 case 'N':z=25;
 case 'Y':{x=50;
           break;}
 default:{x=0;
          break;}
}
print(x);
```



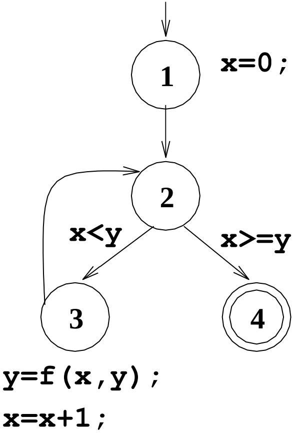
Note: `case 'N'` without a `break` statement leads to `case 'Y'`. It is not so for the other two cases.

CFG: Loops

- There could be various kinds of loops: while, for, do-while etc.
- To accurately represent the possible branches out of a loop, the CFG for loops need *extra* nodes to be added.

CFG: While loop

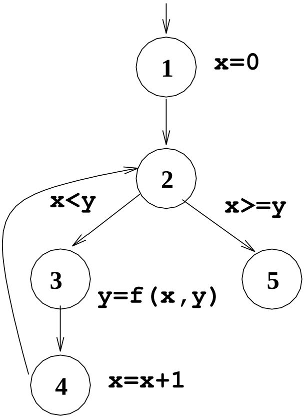
```
x=0;
while(x<y)
{
 y=f(x,y);
 x=x+1;
}
```



Note: Node 2 in the graph above is a dummy node.

CFG: For loop

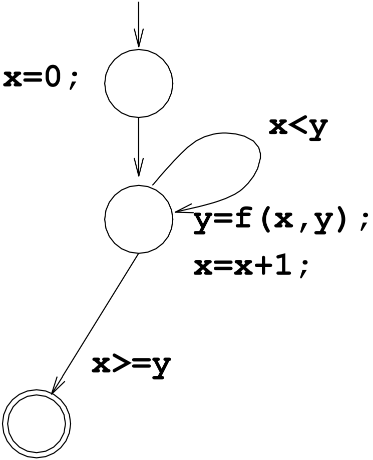
```
for(x=0;x<y;x++)
{
 y=f(x,y);
}
```



Note: Node 1 implicitly initializes the loop, and node 4 implicitly increments the loop.

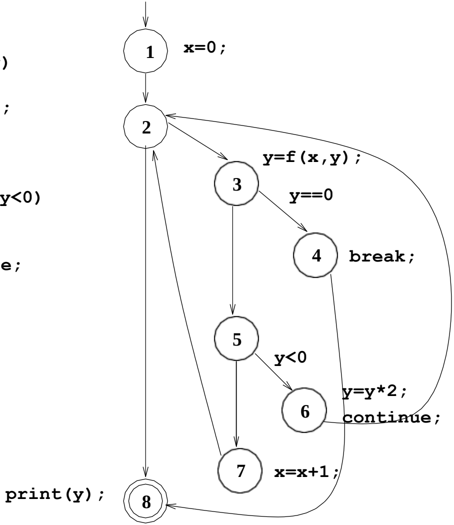
CFG: Do while loop

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



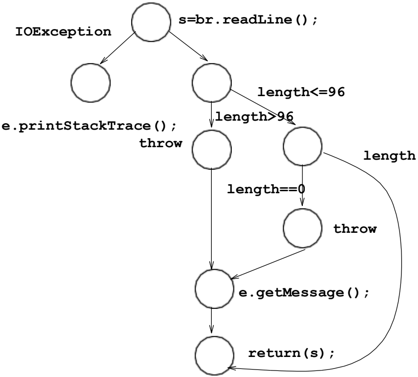
CFG: While loop with break and continue

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



CFG: Exceptions (try-catch)

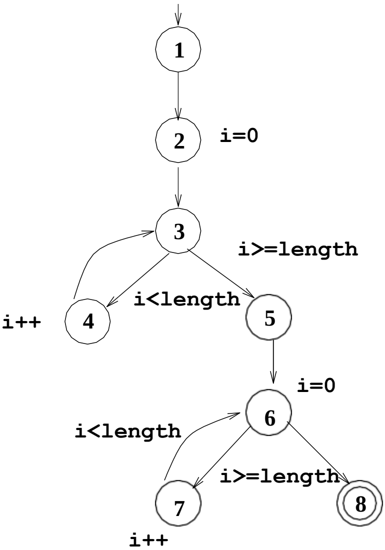
```
try
{
 s=br.readLine();
 if(s.length()>96)
  throw new Exception
   ("too long");
 if(s.length()==0)
  throw new Exception
   ("too short");
}(catch IOException e){
 e.printStackTrace();
}(catch Exception e){
  e.getMessage();
}
return(s);
```



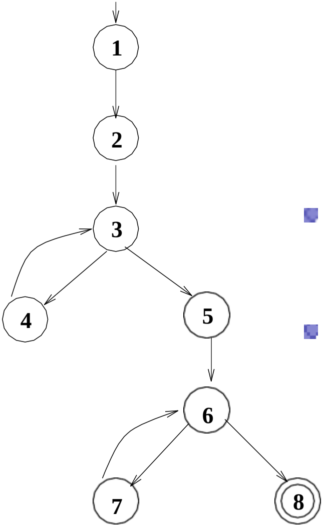
Example program: Statistics

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

CFG for Statistics program

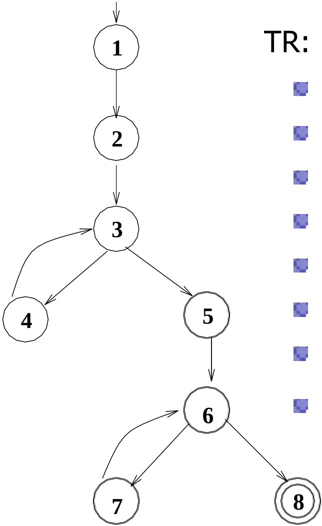


Edge coverage for Statistics program



- TR: {[1,2],[2,3],[3,4],[4,3],[3,5],[5,6],[6,7],[7,6],[7,8]}.
- Test path: [1,2,3,4,3,5,6,7,6,8]

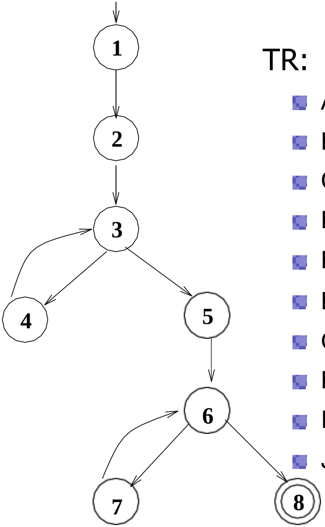
Edge pair coverage for Statistics program



TR:

- 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. [4,3,4]
- H. [5,6,7], I. [5,6,8]
- J. [6,7,6]
- K. [7,6,8], L. [7,6,7].
- Test paths: [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]

Prime path coverage for Statistics program



TR:

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

Test paths:

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

Credits

Part of the material used in these slides are derived from the presentations of the book Introduction to Software Testing, by Paul Ammann and Jeff Offutt.

COURTESY:MEENAKSHI D’SOUZA,IIIT ,BANGLORE