CS409
Software Testing

TAN, Shin Hwei
陈馨慧
Southern University of Science and Technology

Slides adapted from Introduction to Software Testing, Edition 2 (Ch 7.2, 7.3)

Administrative Info

- Grade for MP1 have been posted
  - 8 students have identical code and tests!
    - All relevant students have received emails about this warning
- **Note that plagiarism is** NOT **allowed. You will get 0 if you are suspected of copying individual assignments.**
  - **If you discuss homework with your friend, please explain it in your README.md**

Draw CFG Graph

- Draw one node for each basic block
- Connects basic block with edge
- Label each edge with branch predicate

Edge

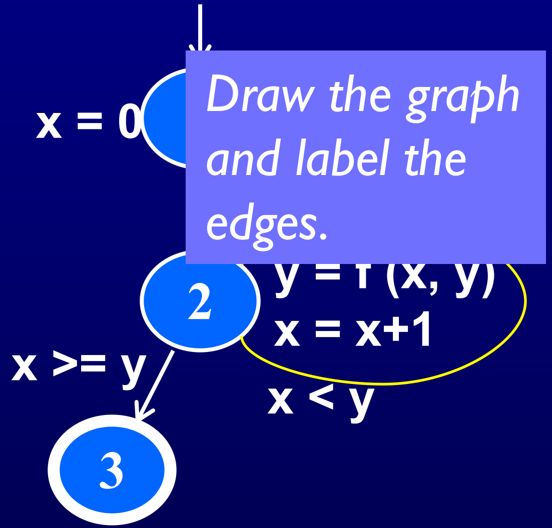
```
if (x ==0)
{
   y = 0;
   x = x + 1;
}else{
   y++;
}
```

Basic Blocks (One Node)

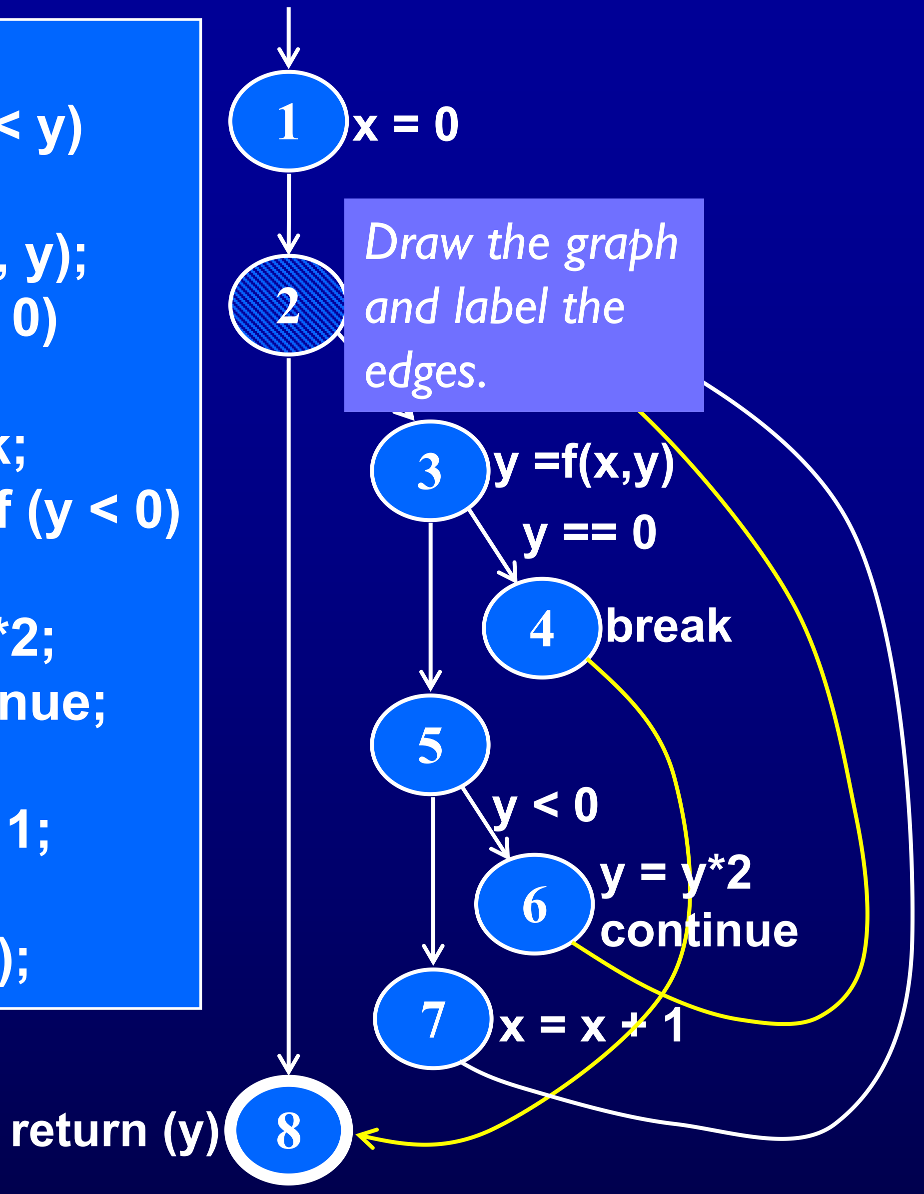
Basic Block (One Node)

CFG : do Loop, break and continue

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



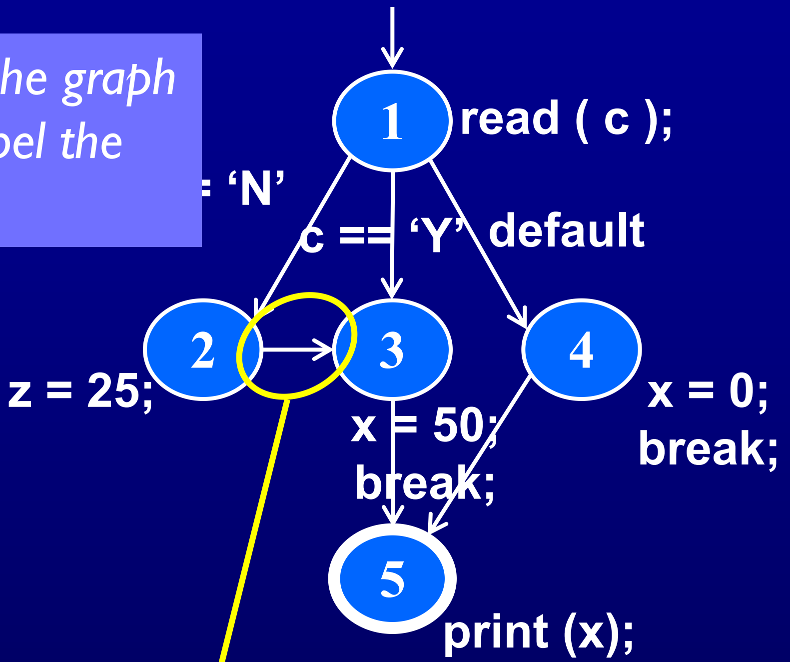

```
x = 0;
while (x < y)
{
   y = f (x, y);
   if (y == 0)
   {
      break;
   } else if (y < 0)
   {
      y = y*2;
      continue;
   }
   x = x + 1;
}
return (y);
```



CFG : The case (switch) Structure

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

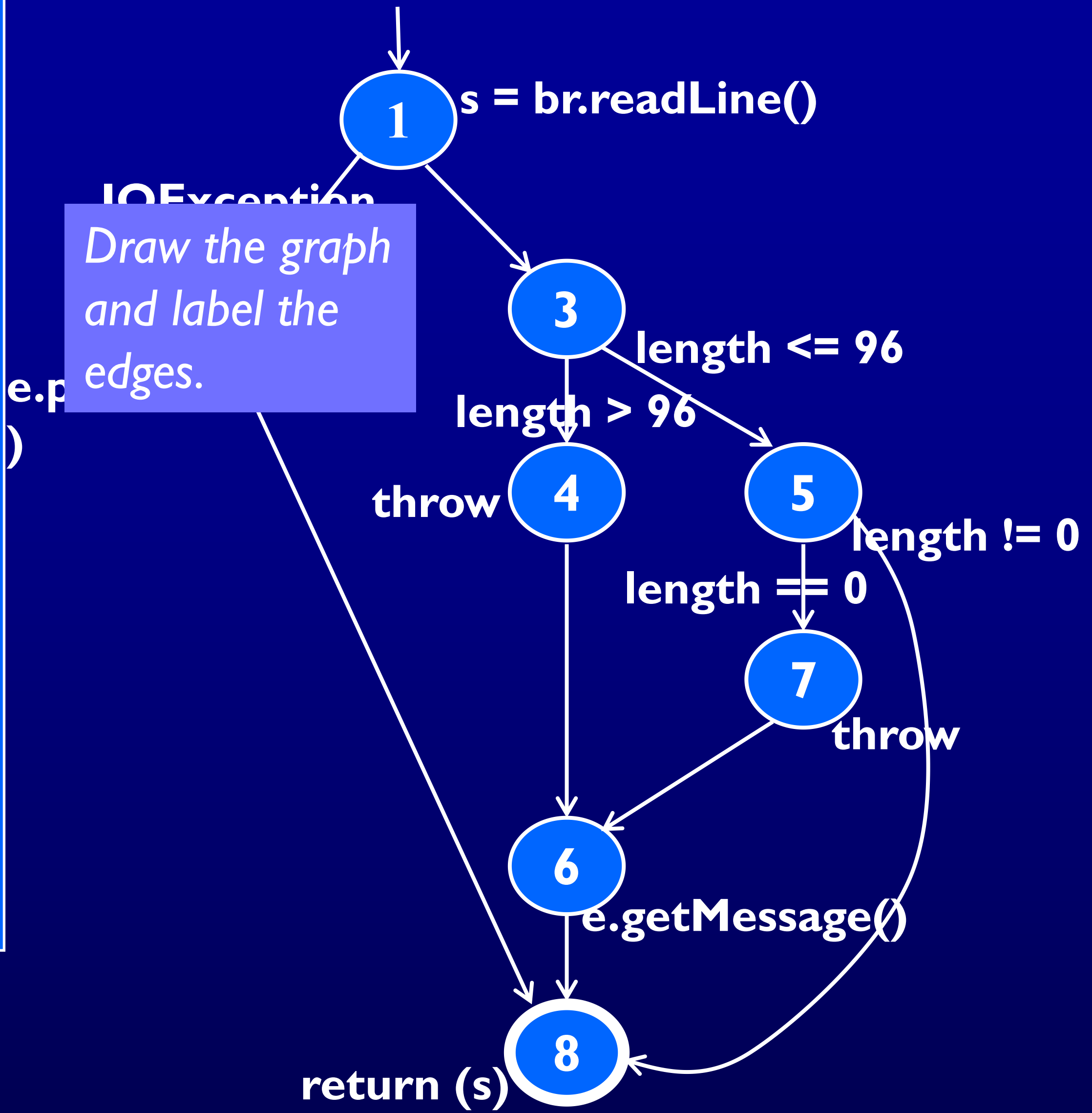
Draw the graph and label the edges.



Cases without breaks fall through to the next case

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);
```



Draw the graph and label the edges.

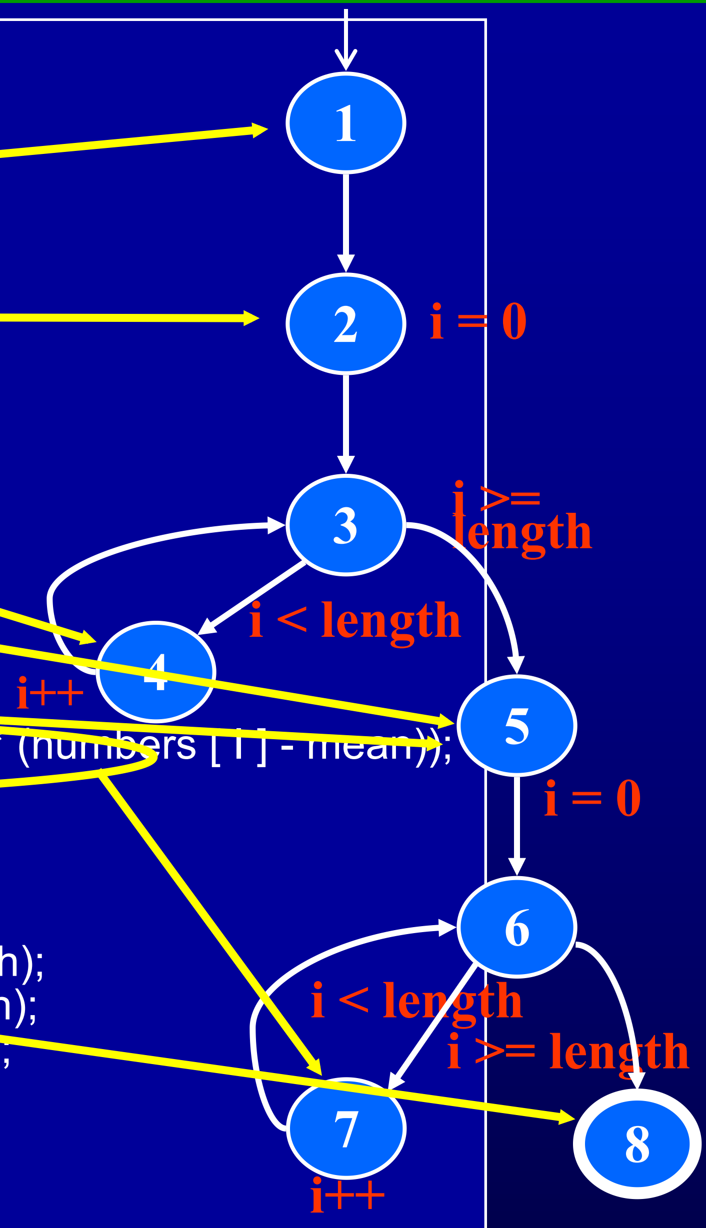
Example Control Flow – Stats

```
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);
}
```

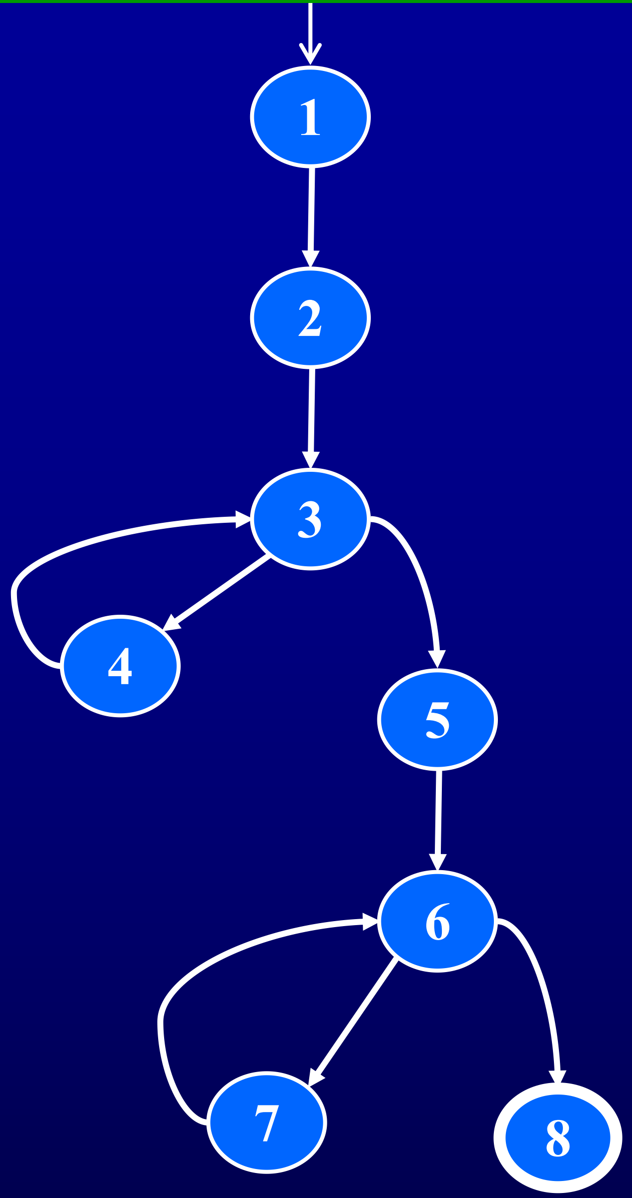
Draw the graph and label the edges.

Control Flow Graph for Stats

```
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

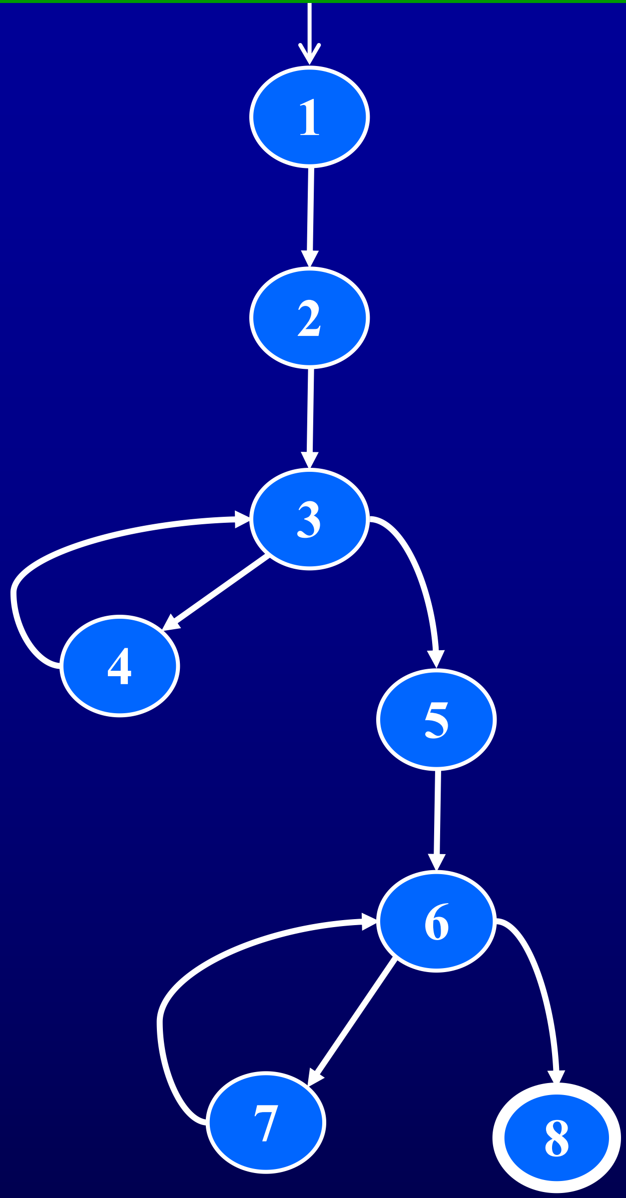


Edge Coverage	
TR	**Test Path**
A. [1, 2]	[1, 2, 6, 8]
D. [3, 5]	
E. [4, 3]	
F. [5, 6]	
G. [6, 7]	
H. [6, 8]	
I. [7, 6]	

Write down the TRs for EC.

Write down test paths that tour all edges.

Control Flow TRs and Test Paths—EPC



Edge-Pair Coverage

TR

A. [1, 2, 3]

Write down TRs for EPC.

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]

Test Paths

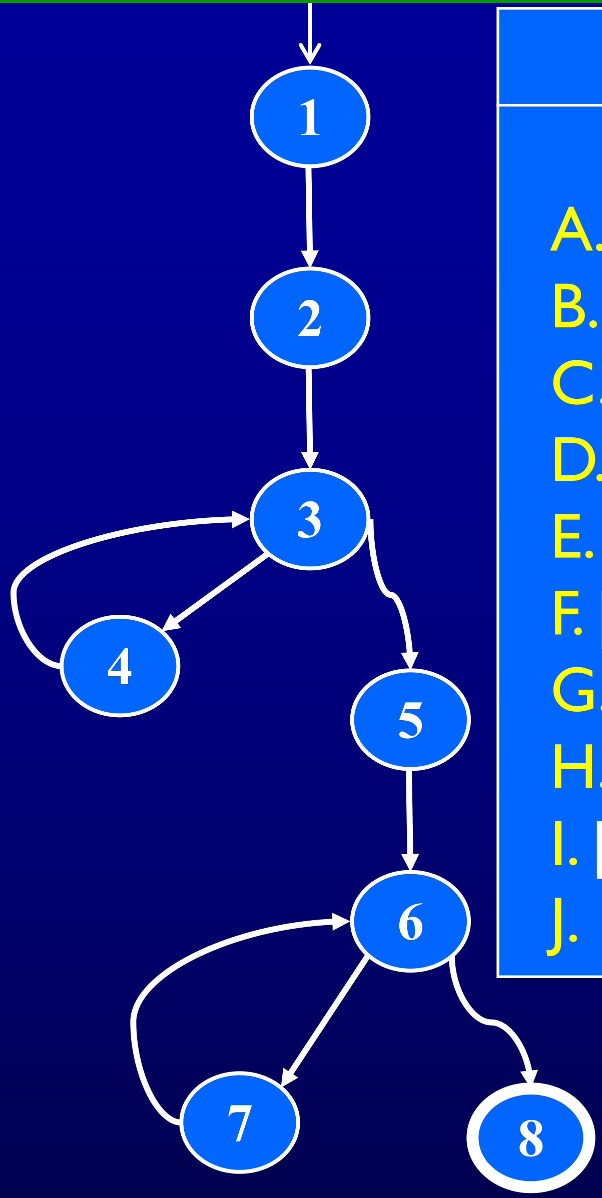
i. [1, 2, 3, 4, 3, 5, 6, 7, 6, 8]
ii. [1
iii. [, 6, 7,

Write down test paths that tour all edge pairs.

TP	TRs toured	sidetrips
~~i~~	~~A, B, D, E, F, G, 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.

Control Flow TRs and Test Paths—PPC



Prime Path Coverage

TR

A. [3, 4, 3]
B. [4
C. [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]

Write down TRs for PPC.

Test Paths

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

Write down test paths that tour all prime paths.

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	

TP ii makes TP i redundant.

Data Flow Coverage for Source

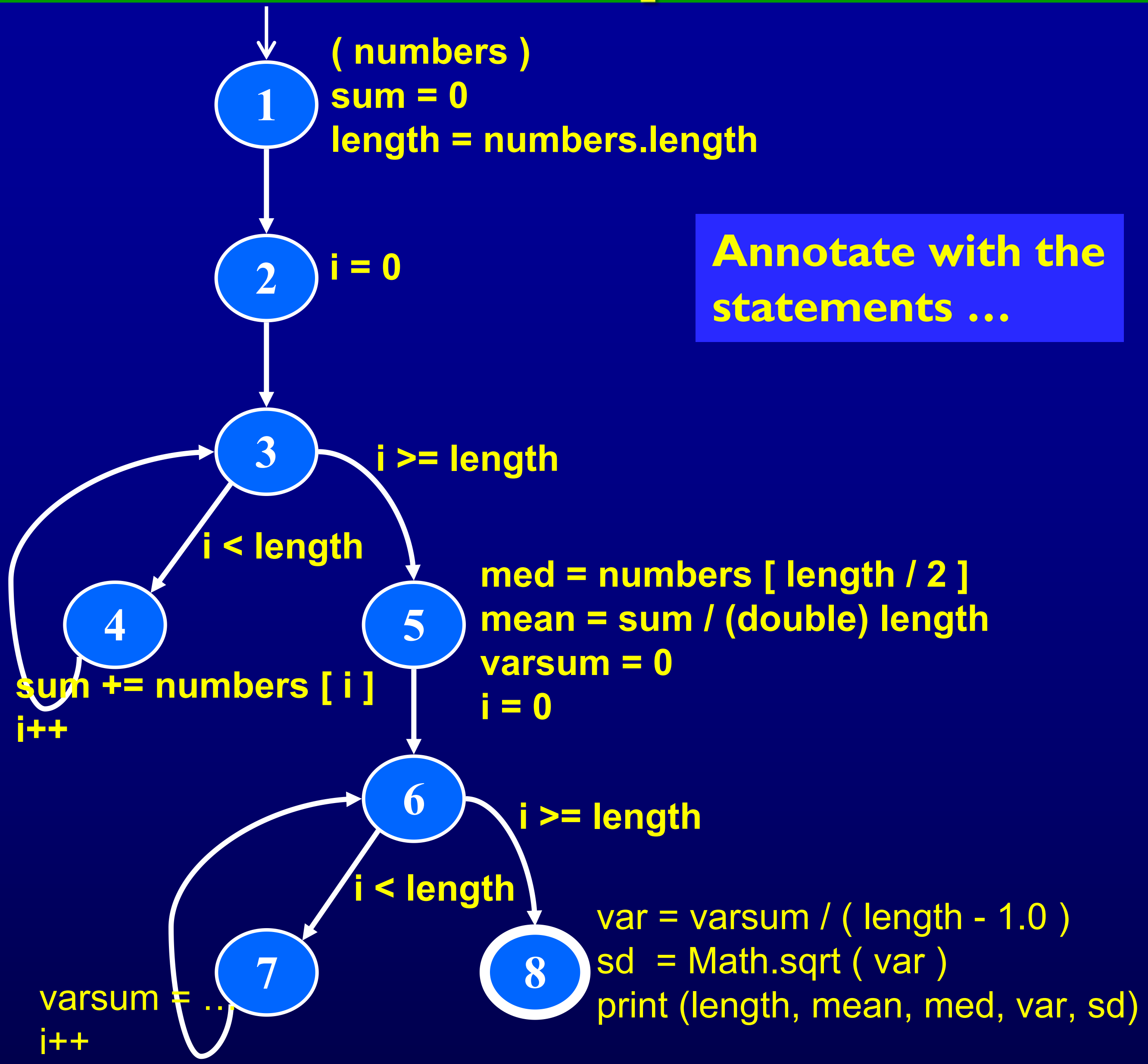
- def : a location where a value is stored into memory
  - x appears on the left side of an assignment (x = 44;)
  - x is an actual parameter in a call and the method changes its value
  - x is a formal parameter of a method (implicit def when method starts)
  - x is an input to a program
- use : a location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

Example Data Flow – Stats

```
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);
}
```

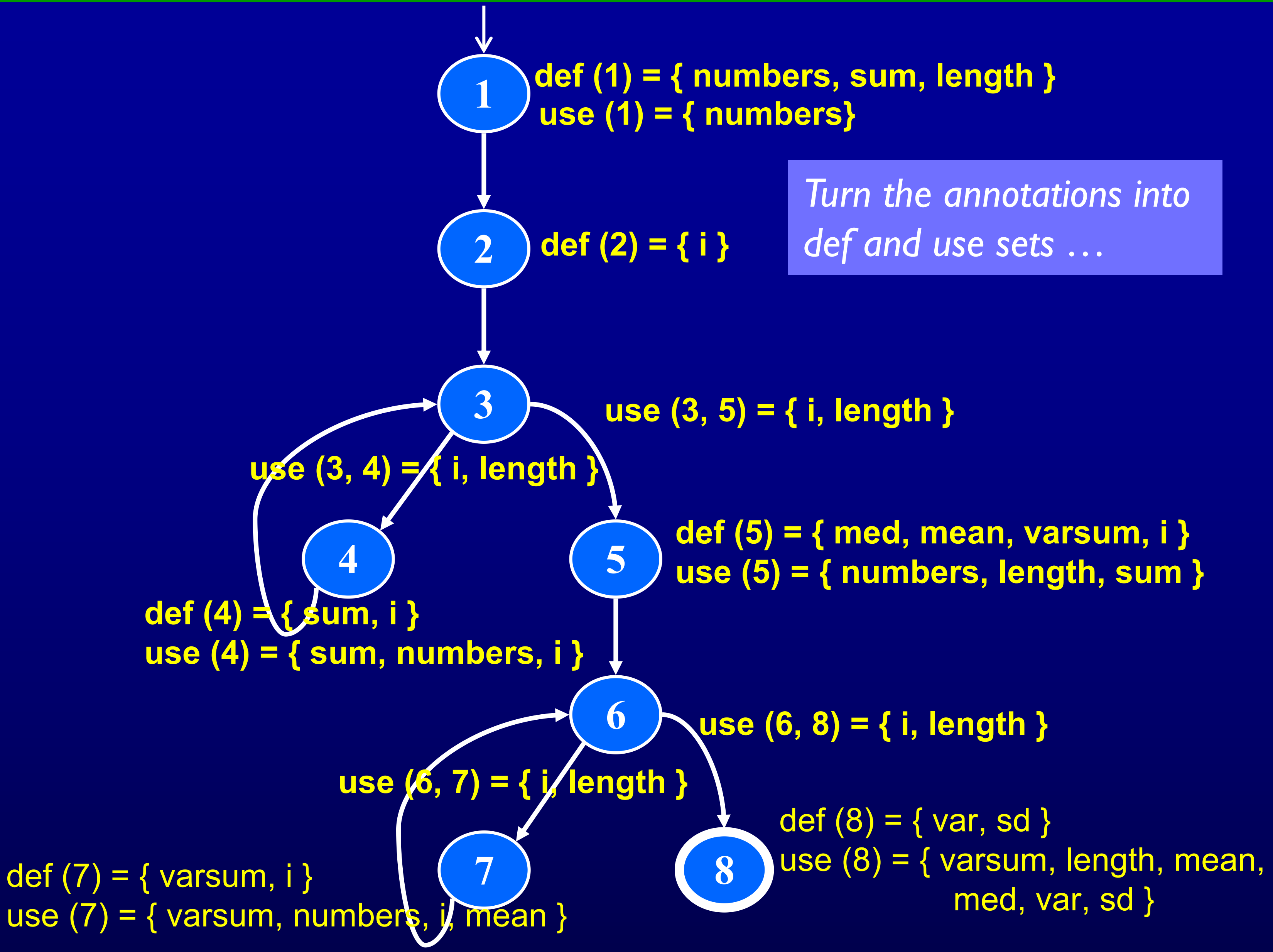
Draw the CFG for computeStats

Control Flow Graph for Stats



Annotate with the statements …

CFG for Stats – With Defs & Uses



Turn the annotations into def and use sets …

Defs and Uses Tables for Stats

Node	Def	Use
1	{ numbers, sum, length }	{ numbers }
2	{ i }	
3		
4	{ sum, i }	{ numbers, i, sum }
5	{ med, mean, varsum, i }	{ numbers, length, sum }
6		
7	{ varsum, i }	{ varsum, numbers, i, mean }
8	{ var, sd }	{ varsum, length, var, mean, med, var, sd }

Edge	Use
(1, 2)	
(2, 3)	
(3, 4)	{ i, length }
(4, 3)	
(3, 5)	{ i, length }
(5, 6)	
(6, 7)	{ i, length }
(7, 6)	
(6, 8)	{ i, length }

DU Pairs for Stats

variable	DU Pairs
numbers	(1, 4) (1, 5) (1, 7)
length	(1, 5) (1, 8) (1, (3,4)) (1, (3,5)) (1, (6,7)) (1, (6,8))
med	(5, 8)
var	(8, 8)
sd	(8, 8)
mean	(5, 7) (5, 8)
sum	(1, 4) (1, 5) (4, 4) (4, 5)
varsum	(5, 7) (5, 8) (7, 7) (7, 8)
i	(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))

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

variable	DU Pairs	DU Paths
numbers	(1, 4) (1, 5) (1, 7)	[1, 2, 3, 4] [1, 2, 3, 5] [1, 2, 3, 5, 6, 7]
length	(1, 5) (1, 8) (1, (3,4)) (1, (3,5)) (1, (6,7)) (1, (6,8))	[1, 2, 3, 5] [1, 2, 3, 5, 6, 8] [1, 2, 3, 4] [1, 2, 3, 5] [1, 2, 3, 5, 6, 7] [1, 2, 3, 5, 6, 8]
med	(5, 8)	[5, 6, 8]
var	(8, 8)	*No path needed*
sd	(8, 8)	*No path needed*
sum	(1, 4) (1, 5) (4, 4) (4, 5)	[1, 2, 3, 4] [1, 2, 3, 5] [4, 3, 4] [4, 3, 5]

variable	DU Pairs	DU Paths
mean	(5, 7) (5, 8)	[5, 6, 7] [5, 6, 8]
varsum	(5, 7) (5, 8) (7, 7) (7, 8)	[5, 6, 7] [5, 6, 8] [7, 6, 7] [7, 6, 8]
i	(2, 4) (2, (3,4)) (2, (3,5)) (4, 4) (4, (3,4)) (4, (3,5)) (5, 7) (5, (6,7)) (5, (6,8)) (7, 7) (7, (6,7)) (7, (6,8))	[2, 3, 4] [2, 3, 4] [2, 3, 5] [4, 3, 4] [4, 3, 4] [4, 3, 5] [5, 6, 7] [5, 6, 7] [5, 6, 8] [7, 6, 7] [7, 6, 7] [7, 6, 8]

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]

★ **4 expect a loop not to be "entered"**

✦ **6 require at least one iteration of a loop**

2 require at least <u>two</u> iterations 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]

<u>Additional DU Paths covered (no sidetrips)</u>

[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]

<u>DU Paths covered (no sidetrips)</u>

[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 index out of bounds exception…

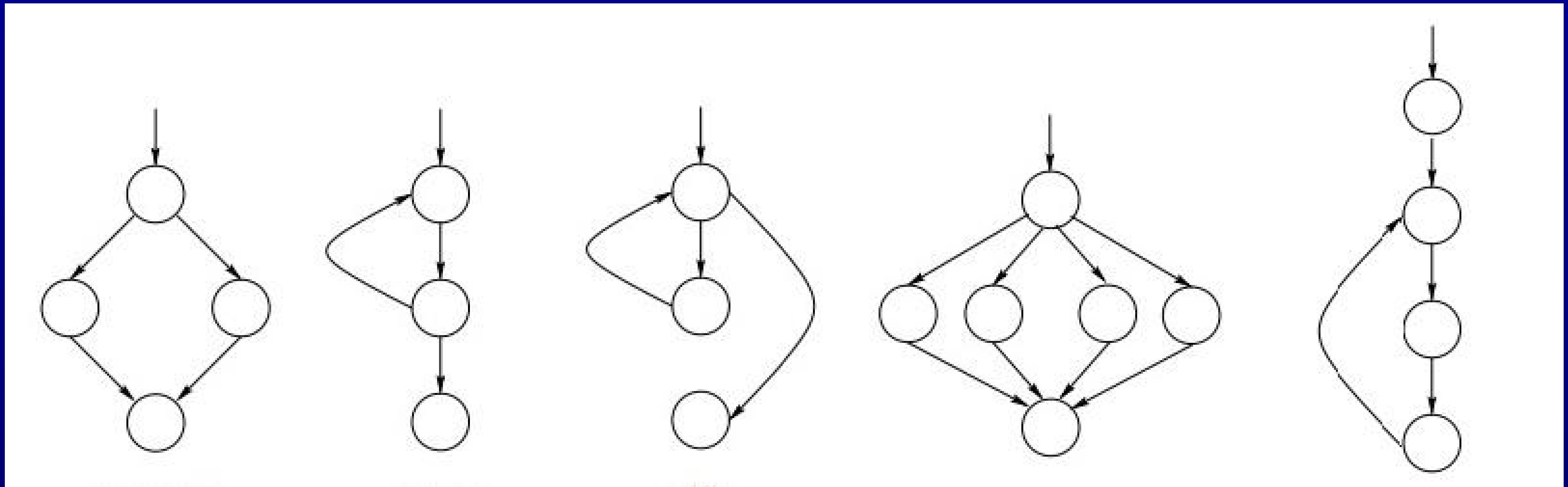
med = numbers [length / 2];

A fault was found

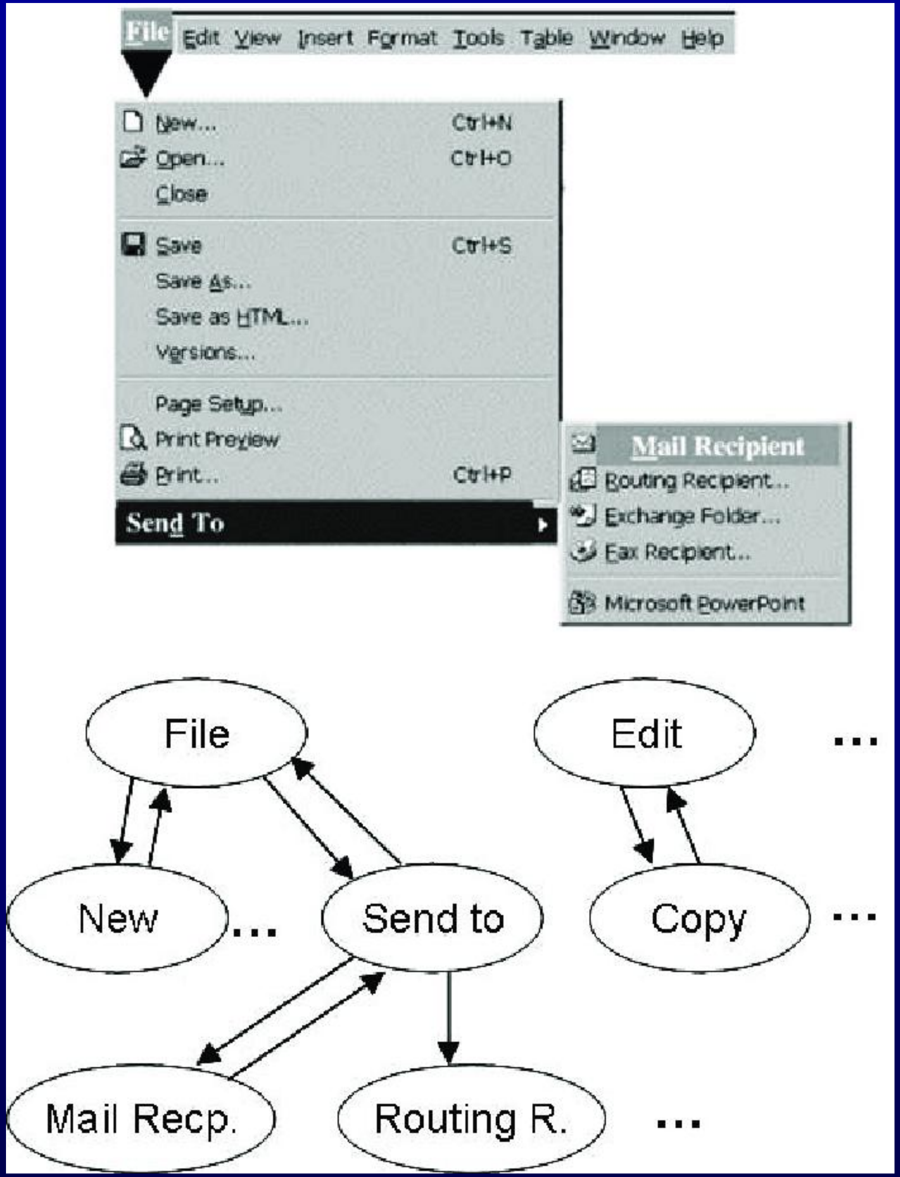
Summary

- 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

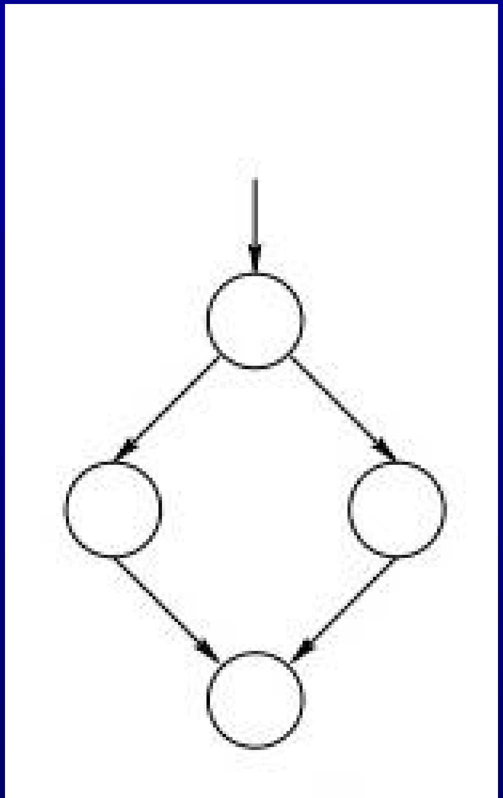
Question: What kind of graph it is?



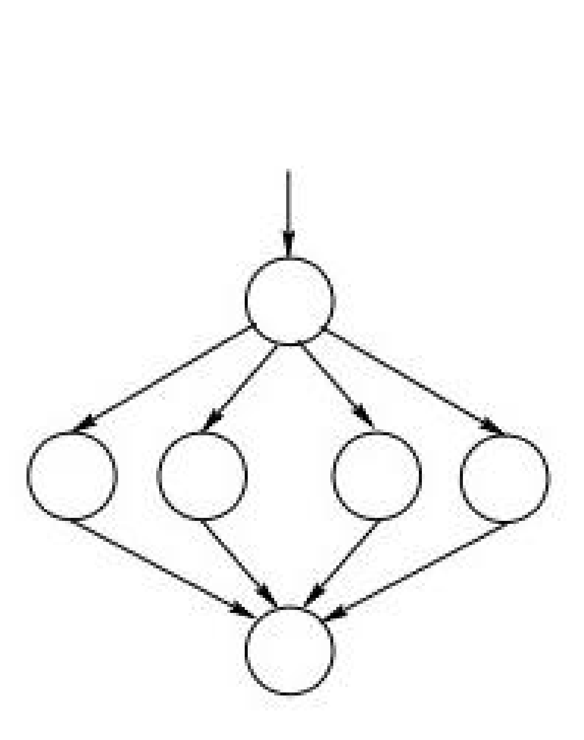
Question: What kind of graph it is?



Question: What kind of control flow this represent?



If-else



Switch cases

Introduction to Software Testing
(*2nd edition*)
Chapter 7.4

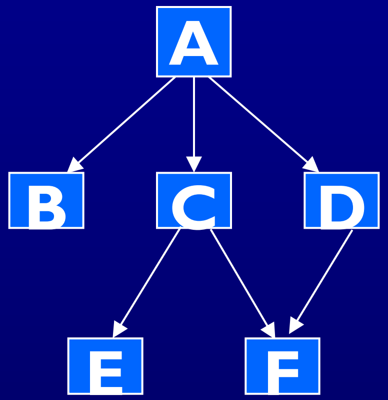
Graph Coverage for Design Elements

OO Software and Designs

- Emphasis on modularity and reuse puts complexity in the design connections
- Testing design relationships is more important than before
- Graphs are based on the connections among the software components
  - Connections are dependency relations, also called couplings

Call Graph

- The most common graph for structural design testing
- Nodes : Units (in Java – methods)
- Edges : Calls to units



Example call graph

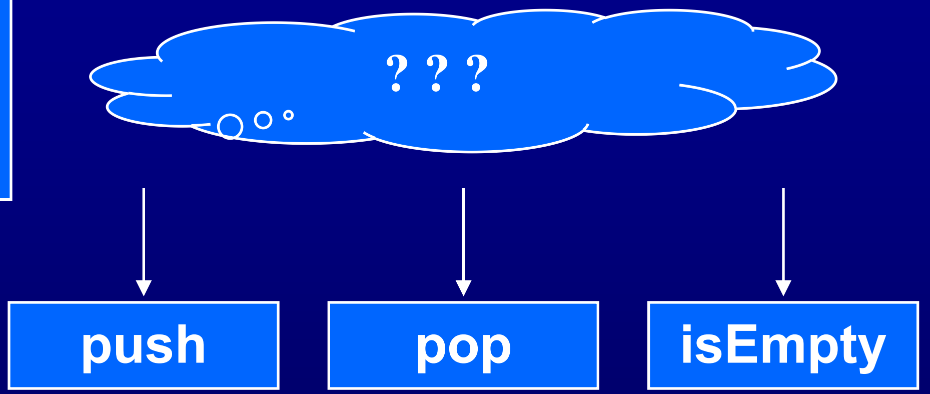
Node coverage : call every unit at least once (method coverage)

Edge coverage : execute every call at least once (call coverage)

Call Graphs on Classes

- Node and edge coverage of class call graphs often do not work very well
- Individual methods might not call each other at all!

<u>Class stack</u>
public void push (Object o)
public Object pop ( )
public boolean isEmpty (Object o)



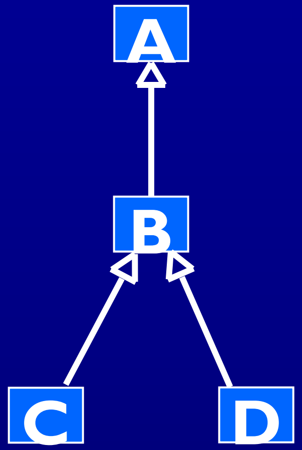
Other types of testing are needed – do <u>not</u> use graph criteria

Inheritance & Polymorphism

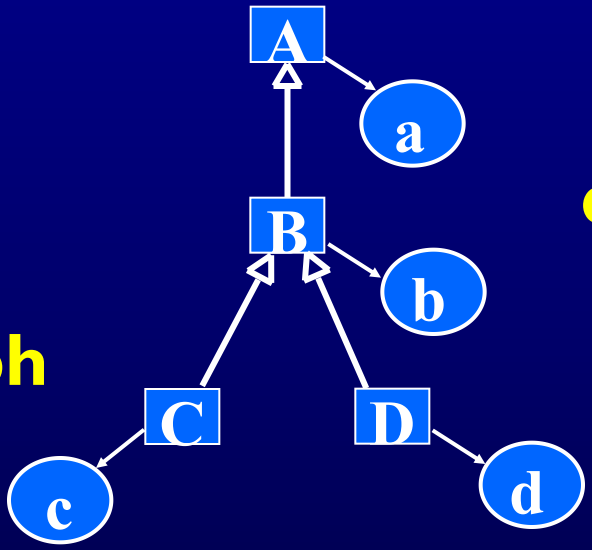
Caution : Ideas are preliminary and not widely used

Classes are not executable, so this graph is not directly testable

We need objects



Example inheritance hierarchy graph



objects

What is coverage on this graph ?

Coverage on Inheritance Graph

- Create an object for each class ?
  - This seems weak because there is no execution
- Create an object for each class and apply call coverage?

OO Call Coverage : TR contains each reachable node in the call graph of an object instantiated for each class in the class hierarchy.

OO Object Call Coverage : TR contains each reachable node in the call graph of every object instantiated for each class in the class hierarchy.

- Data flow is probably more appropriate …

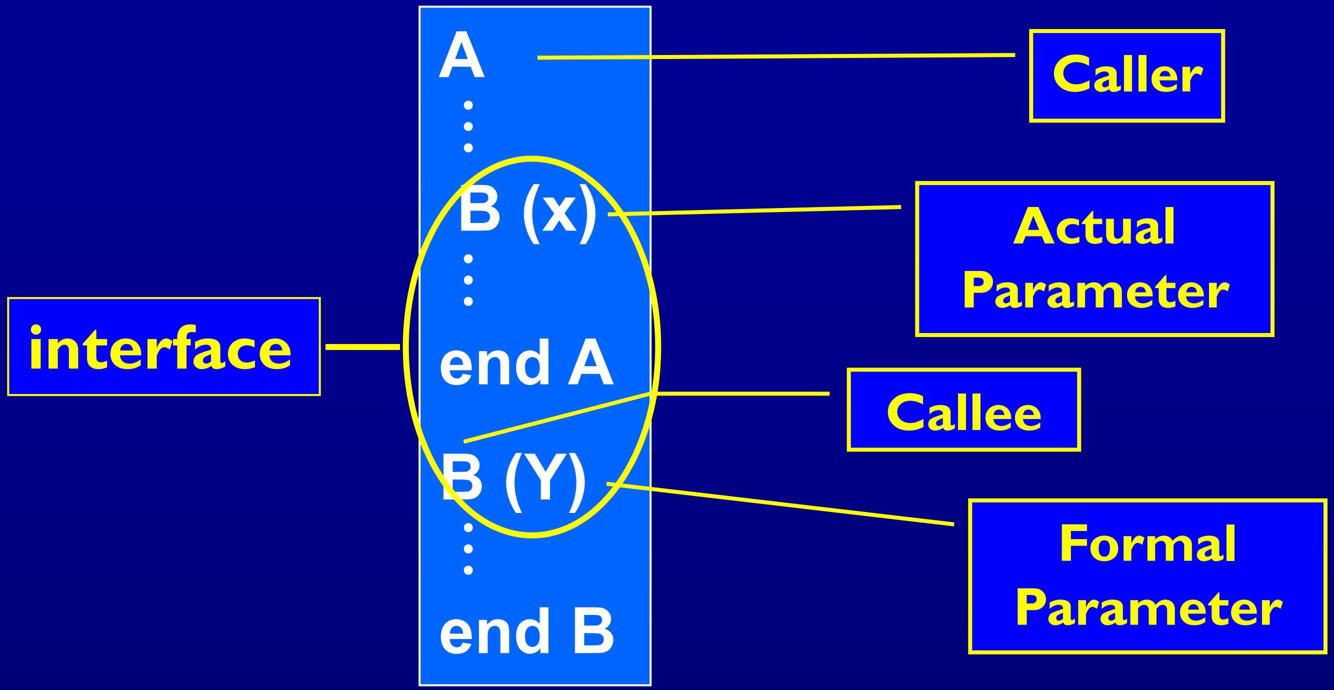
Data Flow at the Design Level

- Data flow couplings among units and classes are more complicated than control flow couplings
  - When values are passed, they "change names"
  - Many different ways to share data
  - Finding defs and uses can be difficult – finding which uses a def can reach is very difficult
- When software gets complicated … testers should get interested
  - That's where the faults are!

Preliminary Definitions

- Caller : A unit that invokes another unit
- Callee : The unit that is called
- Callsite : Statement or node where the call appears
- Actual parameter : Variable in the caller
- Formal parameter : Variable in the callee

Example Call Site



- Applying data flow criteria to def-use pairs between units is too expensive
- Too many possibilities
- But this is integration testing, and we really only care about the interface …