Class 19

- Questions/comments
- DejaVu question
- Efficient path profiling (cont’d)
- Fault localization
- Final project presentations: Dec 1, 3; 4:35-6:45
- Assign (see Schedule for links)
  - Problem Set 8 discuss
  - Readings

DejaVu Question

- For TriType, the change in the first if statement causes all test cases to be rerun.
- However, only the third condition is changed.
- Can we select test cases based on conditions instead of branches?

Acyclic Path Profiling (Notes)

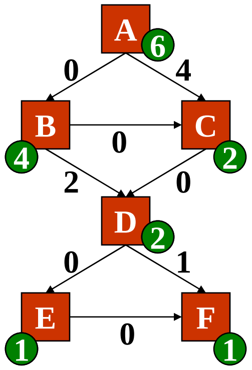
- After Step 1 of the algorithm for computing *Val(e)* for each edge *e*
  - If a vertex has only one out edge *E*, *VAL(E)*=?
  - If a vertex has two out edges—*E1 and E2*—what are *Val(E1),* and *Val(E2)?*

- After Step 1 of the algorithm, regardless of the order in which vertices *V* are processed, *NumPath(V)* is the same. Why?

Algorithm Step 1 Alternative Assignment 1

1. Assign to each edge *e* a value *Val(e)* such that the sum along a path is unique and [0,n-1]

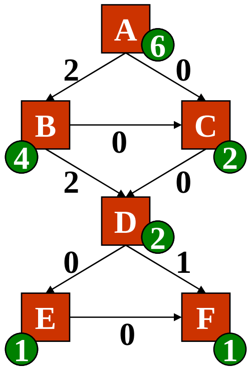
Path	Value
ABCDEF	0
ABCDF	1
ABDEF	2
ABDF	3
ACDEF	4
ACDF	5



Algorithm Step 1 Alternative Assignment 2

1. Assign to each edge *e* a value *Val(e)* such that the sum along a path is unique and [0,n-1]

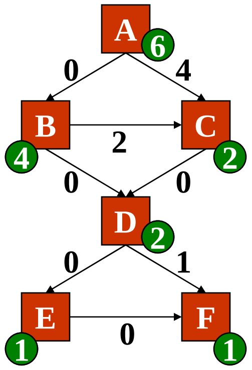
Path	Value
ABCDEF	2
ABCDF	3
ABDEF	4
ABDF	5
ACDEF	0
ACDF	1



Algorithm Step 1 Alternative Assignment 3

1. Assign to each edge *e* a value *Val(e)* such that the sum along a path is unique and [0,n-1]

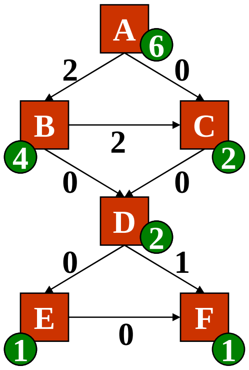
Path	Value
ABCDEF	2
ABCDF	3
ABDEF	0
ABDF	1
ACDEF	4
ACDF	5



Algorithm Step 1 Alternative Assignment 4

1. Assign to each edge *e* a value *Val(e)* such that the sum along a path is unique and [0,n-1]

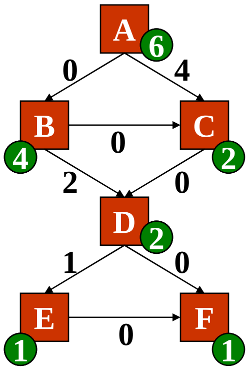
Path	Value
ABCDEF	4
ABCDF	5
ABDEF	2
ABDF	3
ACDEF	0
ACDF	1



Algorithm Step 1 Alternative Assignment 5

1. Assign to each edge *e* a value *Val(e)* such that the sum along a path is unique and [0,n-1]

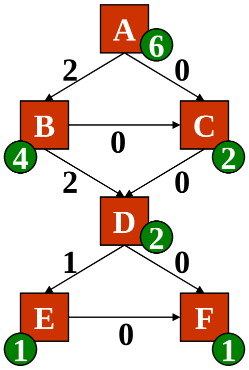
Path	Value
ABCDEF	1
ABCDF	0
ABDEF	3
ABDF	2
ACDEF	5
ACDF	4



Algorithm Step 1 Alternative Assignment 6

1. Assign to each edge *e* a value *Val(e)* such that the sum along a path is unique and [0,n-1]

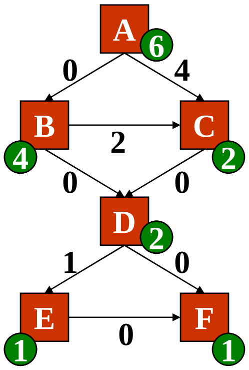
Path	Value
ABCDEF	3
ABCDF	2
ABDEF	5
ABDF	4
ACDEF	1
ACDF	0



Algorithm Step 1 Alternative Assignment 7

1. Assign to each edge *e* a value *Val(e)* such that the sum along a path is unique and [0,n-1]

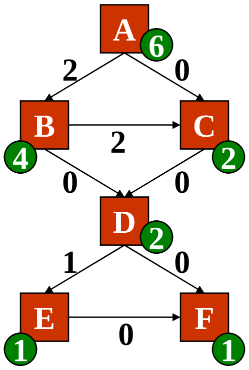
Path	Value
ABCDEF	3
ABCDF	2
ABDEF	1
ABDF	0
ACDEF	5
ACDF	4



Algorithm Step 1 Alternative Assignment 8

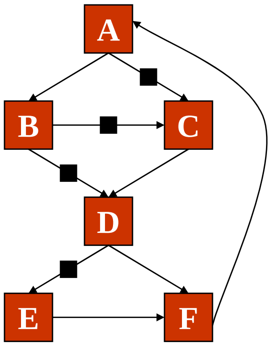
1. Assign to each edge *e* a value *Val(e)* such that the sum along a path is unique and [0,n-1]

Path	Value
ABCDEF	5
ABCDF	4
ABDEF	3
ABDF	2
ACDEF	1
ACDF	0



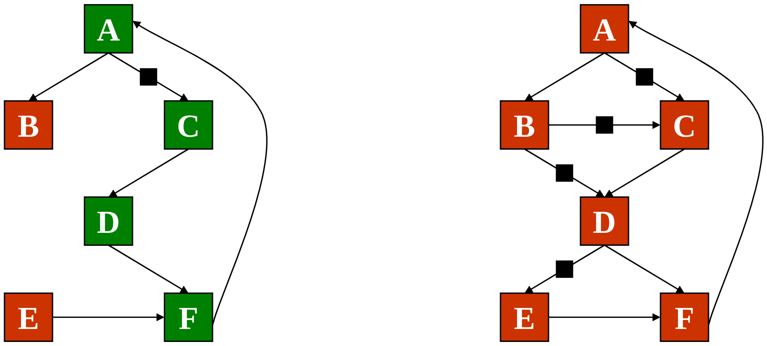
Algorithm (Step 2 of 4)

2. Use a spanning tree to select edges to instrument and compute the appropriate increment for each instrumented edge.
    - Add edge EXIT -> ENTRY
    - Compute a maximal spanning tree (find chords)
    - The addition of each chord to the spanning tree creates a unique cycle called the *fundamental cycle*



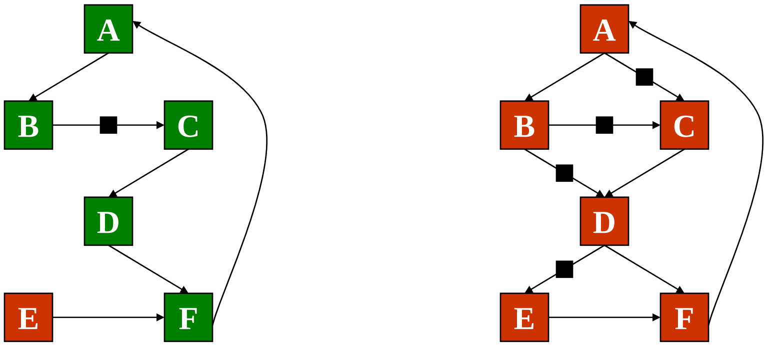
Algorithm Step 2 Fundamental Cycle for A→C

Fundamental cycle shown in green



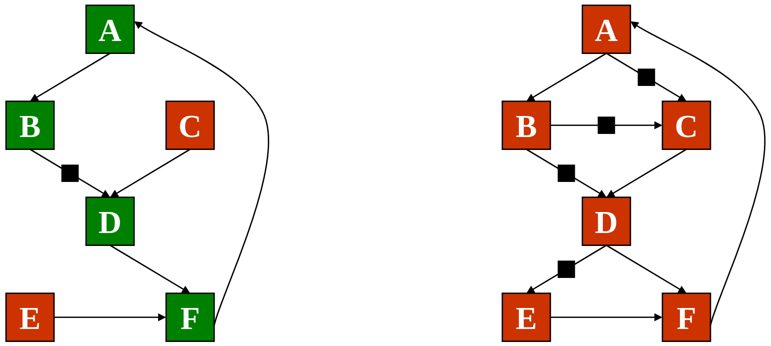
Algorithm Step 2 Fundamental Cycle for B→C

Fundamental cycle shown in green



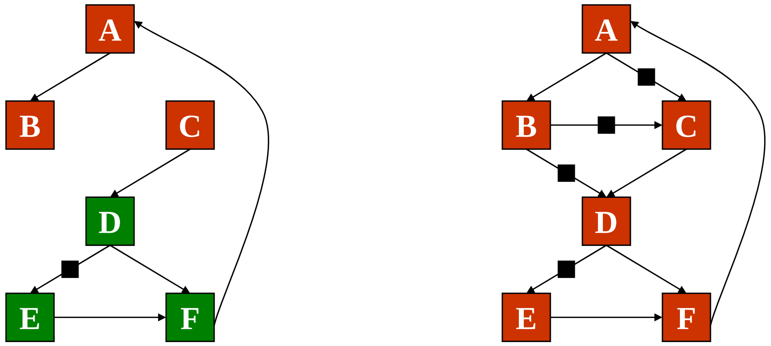
Algorithm Step 2 Fundamental Cycle for B→D

Fundamental cycle shown in green



Algorithm Step 2 Fundamental Cycle for B→D

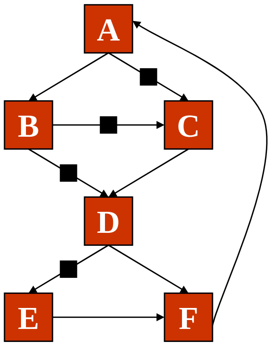
Fundamental cycle shown in green



Remember—edges in spanning tree are undirected

Algorithm (Step 2 of 4)

2. Use a spanning tree to select edges to instrument and compute the appropriate increment for each instrumented edge.
   - Add edge EXIT -> ENTRY
   - Compute a maximal spanning tree (find chords)
   - Assign increments: start from Val(e) and “propagate” to chord [Ball and Larus 94]
   - To do this, consider each chord and its fundamental cycle, along with the values assigned in Step 1



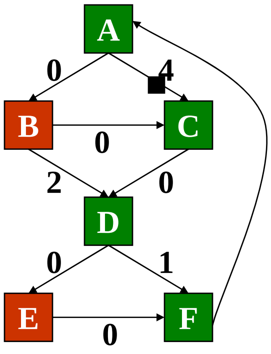
Algorithm Step 2 Assign Increments

2. Use a spanning tree to select edges to instrument and compute the appropriate increment for each instrumented edge.

Consider Alternative Assignment 1 and chord A→C

1 on D→F is propagated to A→C where it is added to 4

The increment on A→C is now 5



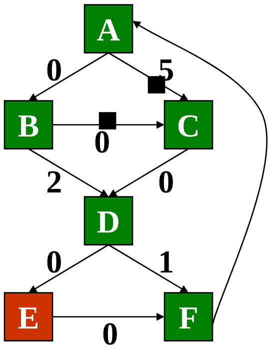
Algorithm Step 2 Assign Increments

2. Use a spanning tree to select edges to instrument and compute the appropriate increment for each instrumented edge.

With chord A→C assigned, consider Alternative Assignment 1 and chord B→C

1 on D→F is propagated to B→C where it is added to 0

The increment on B→C is now 1



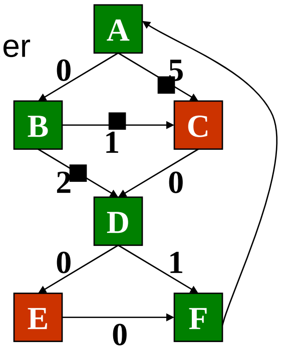
Algorithm Step 2 Assign Increments

2. Use a spanning tree to select edges to instrument and compute the appropriate increment for each instrumented edge.

With chords A→C, B→C assigned, consider Alternative Assignment 1 and chord B→D

1 on D→F is propagated to B→D where it is added added to 2

The increment on B→D is now 3



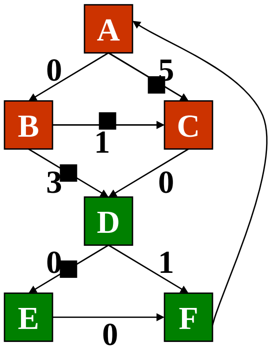
Algorithm Step 2 Assign Increments

2. Use a spanning tree to select edges to instrument and compute the appropriate increment for each instrumented edge.

With chords A→C, B→C, B→D assigned,
consider Alternative Assignment 1
and chord D→E

1 on D→F is propagated to D→E. Because the direction of D→F in the fundamental cycle differs from the direction of the edge in the CFG, the value on D→F is subtracted from the value on D→E

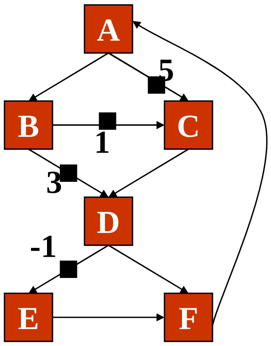
The increment on D→E is now -1



Algorithm Step 2 Assign Increments

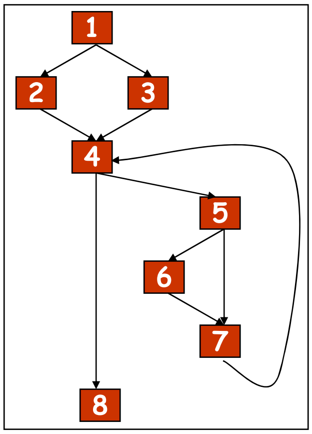
Compare path values using values on all edges and values only on the increment edges to see that they are the same

Path	Value Step 1	Value Step 2
ABCDEF	0	0
ABCDF	1	1
ABDEF	2	2
ABDF	3	3
ACDEF	4	4
ACDF	5	5



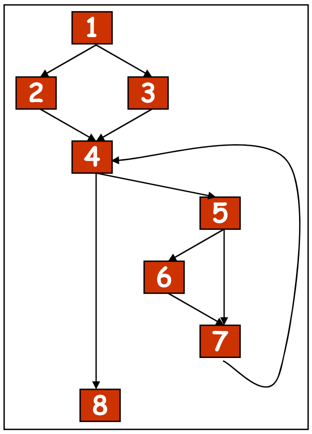
Arbitrary Control Flow (loops)

- Loop implies the presence of a back-edge
- Back-edges instrumented to increment path counter and reinitialize path register (count[r]++; r=0)



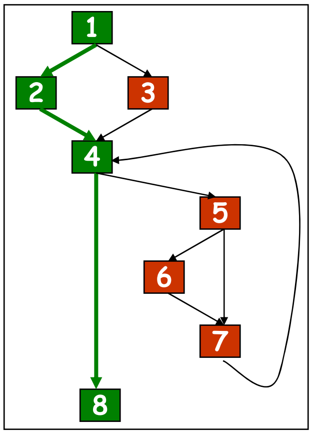
Arbitrary Control Flow (loops)

- Loop implies the presence of a back-edge
- Back-edges instrumented to increment path counter and reinitialize path register (count[r]++; r=0)
- This is not enough; with loops, 4 types of paths (v→w and x→y are back-edges)
  - ENTRY to EXIT
  - ENTRY to v (ending with execution of v→w)
  - w to x (after executing v→w and ending with the execution of x→y; v→w and x→y can be the same back-edge)
  - w to EXIT (after executing v→w)
- Need to distinguish them



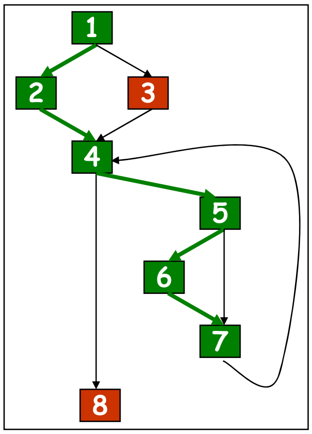
Arbitrary Control Flow (loops)

- Loop implies the presence of a back-edge
- Back-edges instrumented to increment path counter and reinitialize path register (count[r]++; r=0)
- This is not enough; with loops, 4 types of paths (v→w and x→y are back-edges)
  - ENTRY to EXIT
  - ENTRY to v (ending with execution of v→w)
  - w to x (after executing v→w and ending with the execution of x→y; v→w and x→y can be the same back-edge)
  - w to EXIT (after executing v→w)
- Need to distinguish them



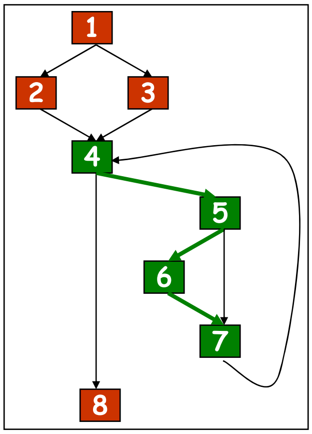
Arbitrary Control Flow (loops)

- Loop implies the presence of a back-edge
- Back-edges instrumented to increment path counter and reinitialize path register (count[r]++; r=0)
- This is not enough; with loops, 4 types of paths (v→w and x→y are back-edges)
  - ENTRY to EXIT
  - ENTRY to v (ending with execution of v→w)
  - w to x (after executing v→w and ending with the execution of x→y; v→w and x→y can be the same back-edge)
  - w to EXIT (after executing v→w)
- Need to distinguish them



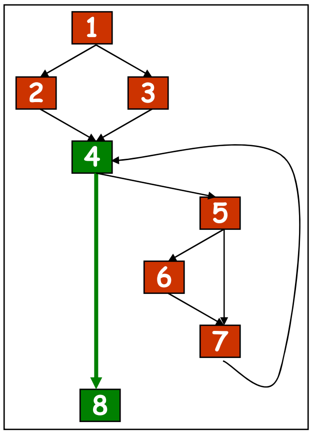
Arbitrary Control Flow (loops)

- Loop implies the presence of a back-edge
- Back-edges instrumented to increment path counter and reinitialize path register (count[r]++; r=0)
- This is not enough; with loops, 4 types of paths (v→w and x→y are back-edges)
  - ENTRY to EXIT
  - ENTRY to v (ending with execution of v->w)
  - w to x (after executing v→w and ending with the execution of x→y; v→w and x→y can be the same back-edge)
  - w to EXIT (after executing v→w)
- Need to distinguish them



Arbitrary Control Flow (loops)

- Loop implies the presence of a back-edge
- Back-edges instrumented to increment path counter and reinitialize path register (count[r]++; r=0)
- This is not enough; with loops, 4 types of paths (v→w and x→y are back-edges)
  - ENTRY to EXIT
  - ENTRY to v (ending with execution of v→w)
  - w to x (after executing v->w and ending with the execution of x→y; v→w and x→y can be the same back-edge)
  - w to EXIT (after executing v→w)
- Need to distinguish them

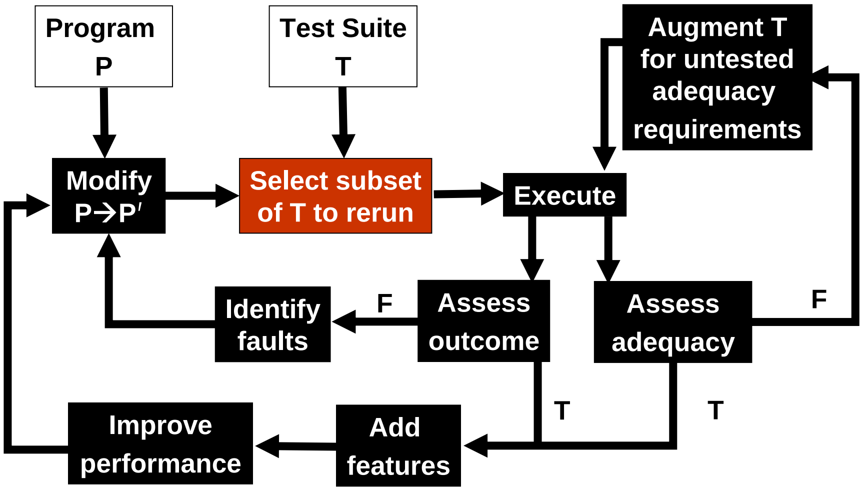


Other Examples with Loops

- Show on the board

Fault Localization

Test, Debug, Fix Cycle



Program P

Test Suite T

Augment T for untested adequacy requirements

Modify P→P′

Select subset of T to rerun

Execute

Identify faults

F

Assess outcome

Assess adequacy

F

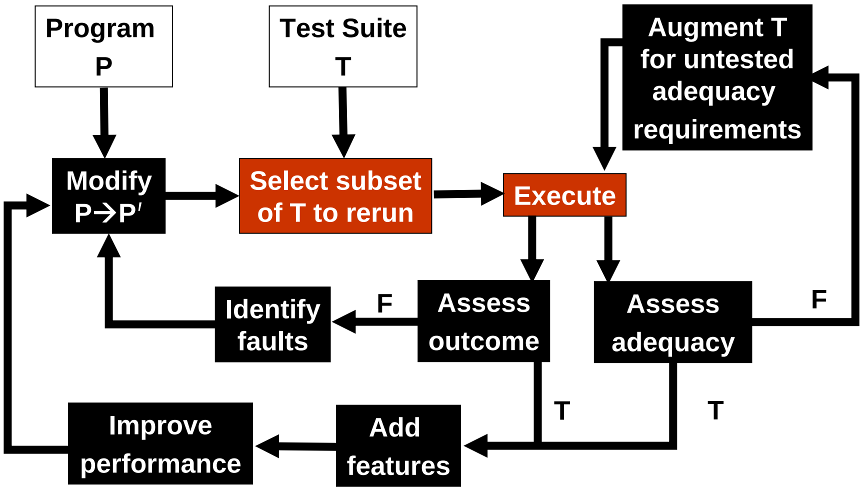
T

T

Improve performance

Add features

Test, Debug, Fix Cycle



Program P

Test Suite T

Augment T for untested adequacy requirements

Modify P→P′

Select subset of T to rerun

Execute

Identify faults

F

Assess outcome

Assess adequacy

F

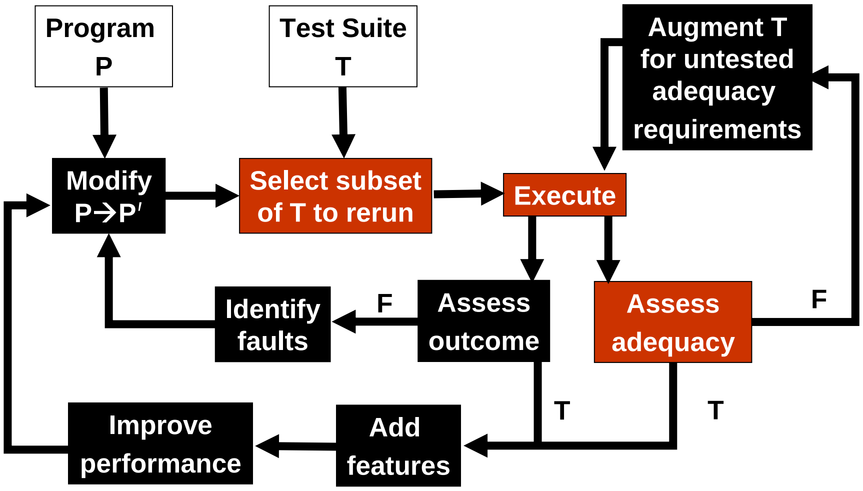
T

T

Improve performance

Add features

Test, Debug, Fix Cycle



Program P

Test Suite T

Augment T for untested adequacy requirements

Modify P→P′

Select subset of T to rerun

Execute

Identify faults

F

Assess outcome

Assess adequacy

F

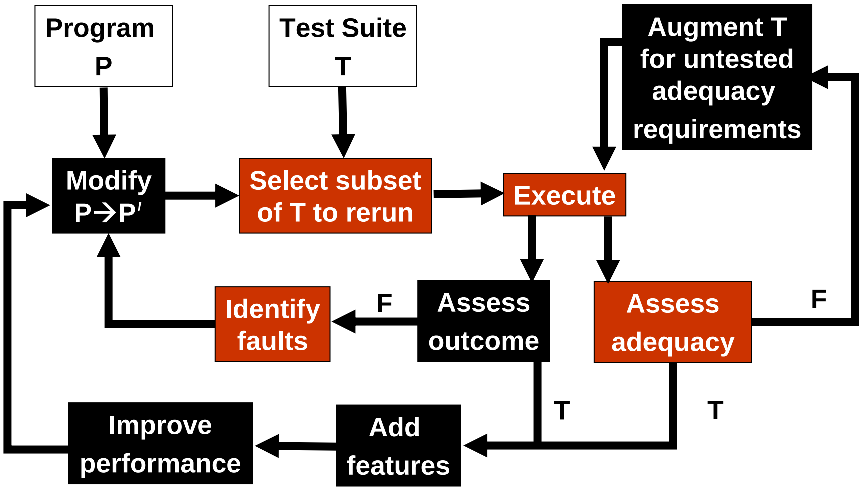
T

T

Improve performance

Add features

Test, Debug, Fix Cycle



Program P

Test Suite T

Augment T for untested adequacy requirements

Modify P→P′

Select subset of T to rerun

Execute

Identify faults

F

Assess outcome

Assess adequacy

F

T

T

Improve performance

Add features

Identify Faults: Fault Localization

Usage scenarios

- Nightly-build process
  - Run set of tests (regression, breadth) each night
  - Report tests that pass and fail
  - **Use fault-localization to identify most likely faulty parts of the software**
- Test-driven development
  - Create and run tests (regression, breadth) after changes
  - Report tests that pass and fail
  - **Use fault-localization to identify most likely faulty parts of the software**
- Regression testing
  - Run set of tests after changes
  - Report tests that pass and fail
  - **Use fault-localization to identify most likely faulty parts of the software**

General Technique—Tarantula

```
 mid() {
   int x,y,z,m;
read(“Enter 3 integers:”
m = z;
if (y<z)
   if (x<y)
       m = y;
   else if (x<z)
       m = y;
else
   if (x>y)
       m = y;
   else if (x>z)
       m = x;
print(“Middle number is:”, m);
  }
```

What is the intuition behind the Tarantula approach?

What information does Tarantula use?

How does the Tarantula technique work?

General Technique—Tarantula

```
  mid() {
    int x,y,z,m;
1:read(“Enter 3 integers:”
2:m = z;
3:if (y<z)
4:   if (x<y)
5:       m = y;
6:   else if (x<z)
7:       m = y;
8:else
9:   if (x>y)
10:      m = y;
11:   else if (x>z)
12:      m = x;
13:print(“Middle number is
   }
```

Technique uses

Dynamic information

- statements executed
- outcome (pass/fail)

Statistical analysis

- computes suspiciousness of each statement

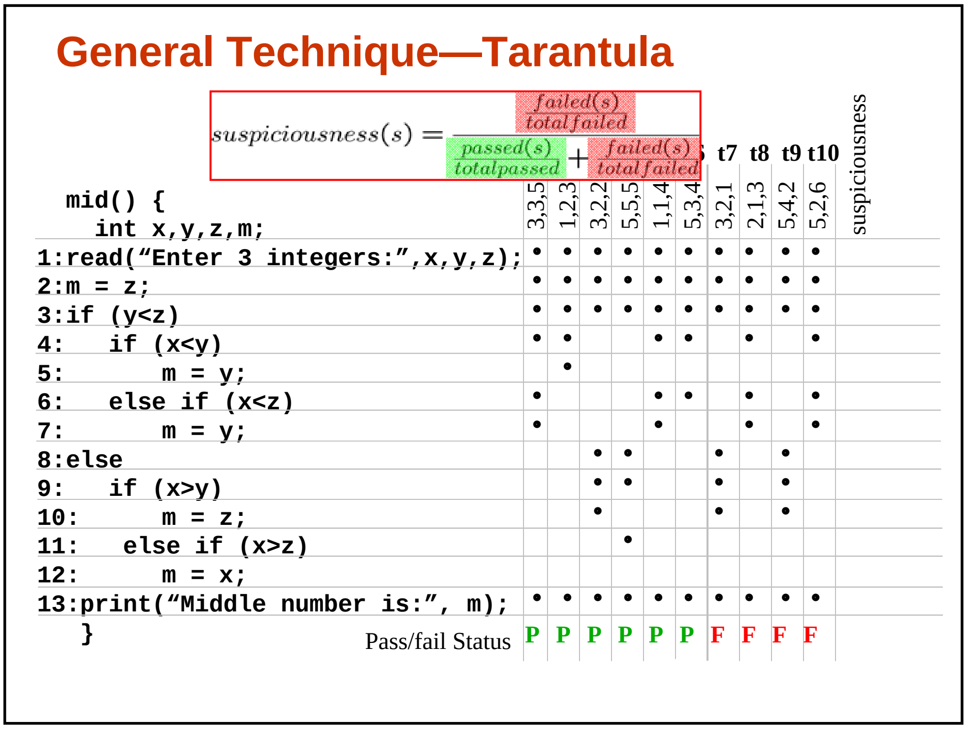
Intuition: Statements primarily executed by failed test cases are more suspicious than statements primarily executed by passed test cases

General Technique—Tarantula

	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10

`mid() {`
`int x,y,z,m;`
	3,3,5	1,2,3	3,2,2	5,5,5	1,1,4	5,3,4	3,2,1	2,1,3	5,4,2	5,2,6

`1:read(“Enter 3 integers:”,x,y,z);`	•	•	•	•	•	•	•	•	•	•
`2:m = z;`	•	•	•	•	•	•	•	•	•	•
`3:if (y<z)`	•	•	•	•	•	•	•	•	•	•
`4:   if (x<y)`	•	•			•	•		•		•
`5:       m = y;`		•								
`6:   else if (x<z)`	•				•	•		•		•
`7:       m = y;`	•				•			•		•
`8:else`			•	•			•		•	
`9:   if (x>y)`			•	•			•		•	
`10:      m = z;`			•				•		•	
`11:   else if (x>z)`				•						
`12:      m = x;`										
`13:print(“Middle number is:”, m);`	•	•	•	•	•	•	•	•	•	•
`}` Pass/fail Status	P	P	P	P	P	P	F	F	F	F

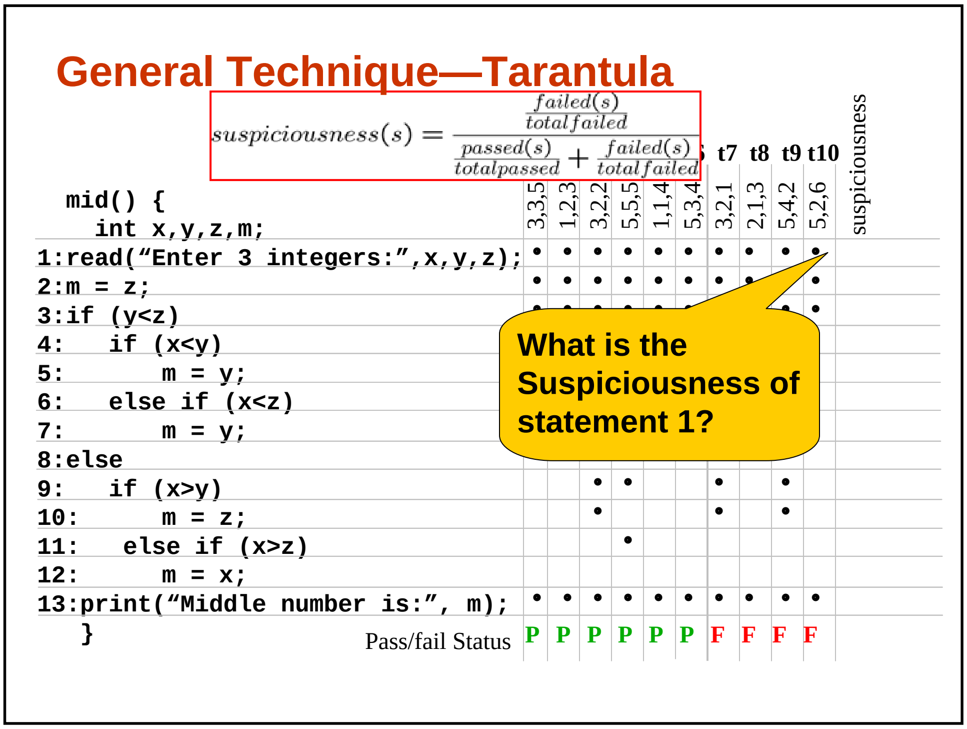


General Technique—Tarantula

$$suspiciousness(s) = \frac{\frac{failed(s)}{total\,failed}}{\frac{passed(s)}{total\,passed} + \frac{failed(s)}{total\,failed}}$$

`mid() {`
`  int x,y,z,m;`

							t7	t8	t9	t10	suspiciousness
	3,3,5	1,2,3	3,2,2	5,5,5	1,1,4	5,3,4	3,2,1	2,1,3	5,4,2	5,2,6	
`1:read("Enter 3 integers:",x,y,z);`	•	•	•	•	•	•	•	•	•	•	
`2:m = z;`	•	•	•	•	•	•	•	•	•	•	
`3:if (y<z)`	•	•	•	•	•	•	•	•	•	•	
`4:   if (x<y)`	•	•			•	•		•		•	
`5:       m = y;`		•									
`6:   else if (x<z)`	•				•	•		•		•	
`7:       m = y;`	•				•			•		•	
`8:else`			•	•			•		•		
`9:   if (x>y)`			•	•			•		•		
`10:      m = z;`			•				•		•		
`11:   else if (x>z)`				•							
`12:      m = x;`											
`13:print("Middle number is:", m);`	•	•	•	•	•	•	•	•	•	•	
`}` Pass/fail Status	P	P	P	P	P	P	F	F	F	F	



General Technique—Tarantula

$$suspiciousness(s) = \frac{\frac{failed(s)}{total\,failed}}{\frac{passed(s)}{total\,passed} + \frac{failed(s)}{total\,failed}}$$

`mid() {`
`  int x,y,z,m;`

							t7	t8	t9	t10	suspiciousness
	3,3,5	1,2,3	3,2,2	5,5,5	1,1,4	5,3,4	3,2,1	2,1,3	5,4,2	5,2,6	
`1:read("Enter 3 integers:",x,y,z);`	•	•	•	•	•	•	•	•	•	•	
`2:m = z;`	•	•	•	•	•	•	•	•		•	
`3:if (y<z)`	•	•	•	•	•	•			•	•	
`4:   if (x<y)`											
`5:       m = y;`											
`6:   else if (x<z)`											
`7:       m = y;`											
`8:else`											
`9:   if (x>y)`			•	•			•		•		
`10:      m = z;`			•				•		•		
`11:   else if (x>z)`				•							
`12:      m = x;`											
`13:print("Middle number is:", m);`	•	•	•	•	•	•	•	•	•	•	
`}` Pass/fail Status	P	P	P	P	P	P	F	F	F	F	

What is the Suspiciousness of statement 1?

General Technique—Tarantula

$$suspiciousness(s) = \frac{\frac{failed(s)}{totalfailed}}{\frac{passed(s)}{totalpassed} + \frac{failed(s)}{totalfailed}}$$

							t7	t8	t9	t10	

`mid() {`
`  int x,y,z,m;`
	3,3,5	1,2,3	3,2,2	5,5,5	1,1,4	5,3,4	3,2,1	2,1,3	5,4,2	5,2,6	suspiciousness

`1:read("Enter 3 integers:",x,y,z);`	•	•	•	•	•	•	•	•	•	•	0.50
`2:m = z;`	•	•	•	•	•	•	•	•	•	•	
`3:if (y<z)`	•	•	•	•	•	•	•	•	•	•	
`4:   if (x<y)`											
`5:        m = y;`											
`6:   else if (x<z)`											
`7:        m = y;`											
`8:else`											
`9:   if (x>y)`			•	•			•		•		
`10:       m = z;`			•				•		•		
`11:   else if (x>z)`				•							
`12:       m = x;`											
`13:print("Middle number is:", m);`	•	•	•	•	•	•	•	•	•	•	
`   }` Pass/fail Status	P	P	P	P	P	P	F	F	F	F	

What is the Suspiciousness of statement 7?

General Technique—Tarantula

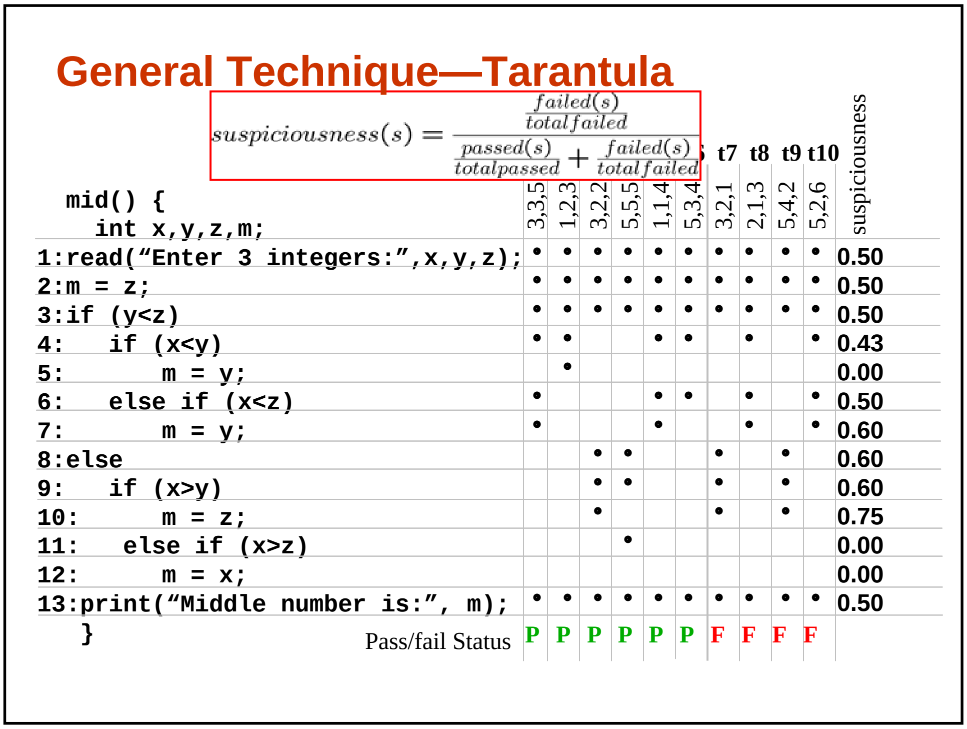
$$suspiciousness(s) = \frac{\frac{failed(s)}{totalfailed}}{\frac{passed(s)}{totalpassed} + \frac{failed(s)}{totalfailed}}$$

							t7	t8	t9	t10	

`mid() {`
`  int x,y,z,m;`
	3,3,5	1,2,3	3,2,2	5,5,5	1,1,4	5,3,4	3,2,1	2,1,3	5,4,2	5,2,6	suspiciousness

`1:read("Enter 3 integers:",x,y,z);`	•	•	•	•	•	•	•	•	•	•	0.50
`2:m = z;`	•	•	•	•	•	•	•	•	•	•	
`3:if (y<z)`	•	•	•	•	•	•	•	•	•	•	
`4:   if (x<y)`											
`5:        m = y;`											
`6:   else if (x<z)`											
`7:        m = y;`											0.60
`8:else`											
`9:   if (x>y)`			•	•			•		•		
`10:       m = z;`			•				•		•		
`11:   else if (x>z)`				•							
`12:       m = x;`											
`13:print("Middle number is:", m);`	•	•	•	•	•	•	•	•	•	•	
`   }` Pass/fail Status	P	P	P	P	P	P	F	F	F	F	

What is the Suspiciousness of statement 7?



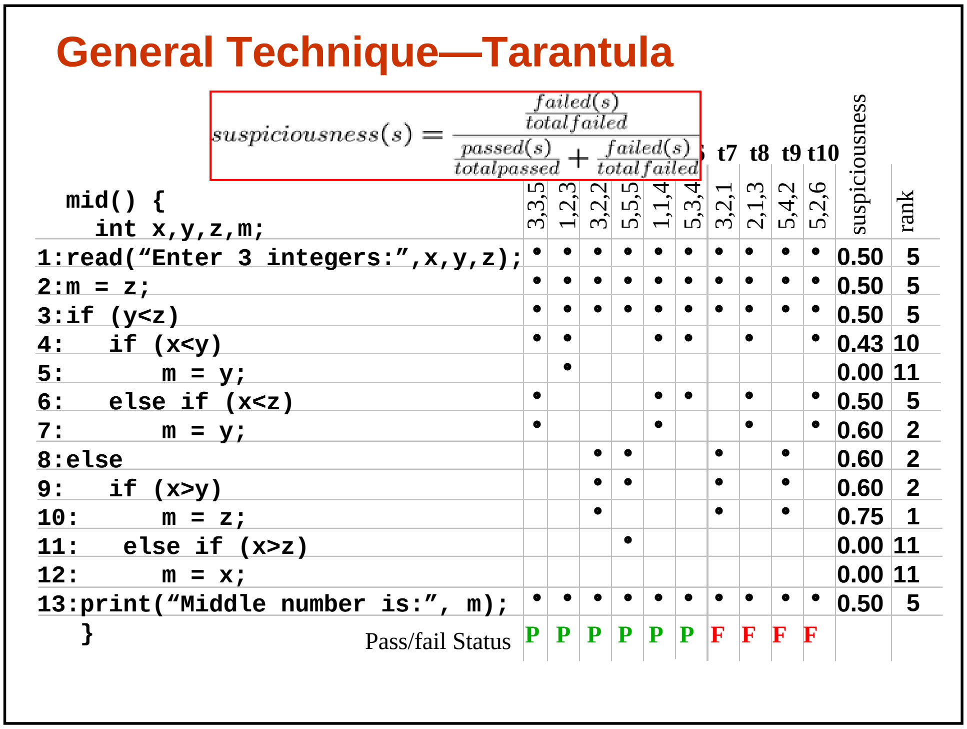
General Technique—Tarantula

$$suspiciousness(s) = \frac{\frac{failed(s)}{totalfailed}}{\frac{passed(s)}{totalpassed}+\frac{failed(s)}{totalfailed}}$$

							t7	t8	t9	t10	suspiciousness

`mid() {`
`  int x,y,z,m;`
	3,3,5	1,2,3	3,2,2	5,5,5	1,1,4	5,3,4	3,2,1	2,1,3	5,4,2	5,2,6	

`1:read("Enter 3 integers:",x,y,z);`	•	•	•	•	•	•	•	•	•	•	0.50
`2:m = z;`	•	•	•	•	•	•	•	•	•	•	0.50
`3:if (y<z)`	•	•	•	•	•	•	•	•	•	•	0.50
`4:   if (x<y)`	•	•			•	•		•		•	0.43
`5:       m = y;`		•									0.00
`6:   else if (x<z)`	•				•	•		•		•	0.50
`7:       m = y;`	•				•			•		•	0.60
`8:else`			•	•			•		•		0.60
`9:   if (x>y)`			•	•			•		•		0.60
`10:      m = z;`			•				•		•		0.75
`11:   else if (x>z)`				•							0.00
`12:      m = x;`											0.00
`13:print("Middle number is:", m);`	•	•	•	•	•	•	•	•	•	•	0.50
`}` Pass/fail Status	P	P	P	P	P	P	F	F	F	F	



General Technique—Tarantula

$$suspiciousness(s) = \frac{\frac{failed(s)}{totalfailed}}{\frac{passed(s)}{totalpassed}+\frac{failed(s)}{totalfailed}}$$

							t7	t8	t9	t10	suspiciousness	rank
`mid() {` `  int x,y,z,m;`	3,3,5	1,2,3	3,2,2	5,5,5	1,1,4	5,3,4	3,2,1	2,1,3	5,4,2	5,2,6		
`1:read("Enter 3 integers:",x,y,z);`	•	•	•	•	•	•	•	•	•	•	0.50	5
`2:m = z;`	•	•	•	•	•	•	•	•	•	•	0.50	5
`3:if (y<z)`	•	•	•	•	•	•	•	•	•	•	0.50	5
`4:   if (x<y)`	•	•			•	•		•		•	0.43	10
`5:       m = y;`		•									0.00	11
`6:   else if (x<z)`	•				•	•		•		•	0.50	5
`7:       m = y;`	•				•			•		•	0.60	2
`8:else`			•	•			•		•		0.60	2
`9:   if (x>y)`			•	•			•		•		0.60	2
`10:      m = z;`			•				•		•		0.75	1
`11:   else if (x>z)`				•							0.00	11
`12:      m = x;`											0.00	11
`13:print("Middle number is:", m);`	•	•	•	•	•	•	•	•	•	•	0.50	5
`}` Pass/fail Status	P	P	P	P	P	P	F	F	F	F		

General Technique—Tarantula

$$suspiciousness(s) = \frac{\frac{failed(s)}{totalfailed}}{\frac{passed(s)}{totalpassed}+\frac{failed(s)}{totalfailed}}$$

							t7	t8	t9	t10	suspiciousness	rank
mid() { int x,y,z,m;	3,3,5	1,2,3	3,2,2	5,5,5	1,1,4	5,3,4	3,2,1	2,1,3	5,4,2	5,2,6		
1:read(“Enter 3 integers:”,x,y,z);	•	•	•	•	•	•	•	•	•	•	0.50	5
2:m = z;	•	•	•	•	•	•	•	•	•	•	0.50	5
3:if (y<z)	•	•	•	•	•	•	•	•	•	•	0.50	5
4: if (x<y)	•	•			•	•		•		•	0.43	10
5: m = y;		•									0.00	11
6: else if (x<z)	•				•	•		•		•	0.50	5
7: m = y;	•				•			•		•	0.60	2
8:else			•	•			•		•		0.60	2
9: if (x>y)			•	•			•		•		0.60	2
10: m = z; //bug;correct m=y			•				•		•		0.75	1
11: else if (x>z)				•							0.00	11
12: m = x;											0.00	11
13:print(“Middle number is:”, m);	•	•	•	•	•	•	•	•	•	•	0.50	5
} Pass/fail Status	P	P	P	P	P	P	F	F	F	F		

General Technique—Tarantula

$$suspiciousness(s) = \frac{\frac{failed(s)}{totalfailed}}{\frac{passed(s)}{totalpassed}+\frac{failed(s)}{totalfailed}}$$

							t7	t8	t9	t10	suspiciousness	rank
mid() { int x,y,z,m;	3,3,5	1,2,3	3,2,2	5,5,5	1,1,4	5,3,4	3,2,1	2,1,3	5,4,2	5,2,6		
1:read(“Enter 3 integers:”,x,y,z);	•	•	•	•	•	•	•	•	•	•		
2:m = z;	•	•	•	•	•	•	•	•	•	•		
3:if (y<z)	•	•	•	•	•	•	•	•	•	•		
4: if (x<y)	•	•			•	•		•		•		
5: m = y;		•										
6: else if (x<z)	•				•	•		•		•		
7: m = y;	•				•			•		•		
8:else			•	•			•		•			
9: if (x>y)			•	•			•		•			
10: m = y; //fixed			•				•		•			
11: else if (x>z)				•								
12: m = x;												
13:print(“Middle number is:”, m);	•	•	•	•	•	•	•	•	•	•		
} Pass/fail Status	P	P	P	P	P	P	P	F	P	F		

General Technique—Tarantula

$$suspiciousness(s) = \frac{\frac{failed(s)}{totalfailed}}{\frac{passed(s)}{totalpassed} + \frac{failed(s)}{totalfailed}}$$

							t7	t8	t9	t10		
mid() { int x,y,z,m;	3,3,5	1,2,3	3,2,2	5,5,5	1,1,4	5,3,4	3,2,1	2,1,3	5,4,2	5,2,6	suspiciousness	rank
1:read(“Enter 3 integers:”,x,y,z);	•	•	•	•	•	•	•	•	•	•	0.50	4
2:m = z;	•	•	•	•	•	•	•	•	•	•	0.50	4
3:if (y<z)	•	•	•	•	•	•	•	•	•	•	0.50	4
4: if (x<y)	•	•			•	•		•		•	0.67	3
5: m = y;		•									0.00	8
6: else if (x<z)	•				•	•		•		•	0.73	2
7: m = y; //bug;correct:m=x;	•				•			•		•	0.80	1
8:else			•	•			•		•		0.00	8
9: if (x>y)			•	•			•		•		0.00	8
10: m = y; //fixed			•				•		•		0.00	8
11: else if (x>z)				•							0.00	8
12: m = x;											0.00	8
13:print(“Middle number is:”, m);	•	•	•	•	•	•	•	•	•	•	0.50	4
} Pass/fail Status	P	P	P	P	P	P	P	F	P	F		

General Technique—Tarantula

$$suspiciousness(s) = \frac{\frac{failed(s)}{totalfailed}}{\frac{passed(s)}{totalpassed} + \frac{failed(s)}{totalfailed}}$$

							t7	t8	t9	t10		
mid() { int x,y,z,m;	3,3,5	1,2,3	3,2,2	5,5,5	1,1,4	5,3,4	3,2,1	2,1,3	5,4,2	5,2,6	suspiciousness	rank
1:read(“Enter 3 integers:”,x,y,z);	•	•	•	•	•	•	•	•	•	•	0.00	
2:m = z;	•	•	•	•	•	•	•	•	•	•	0.00	
3:if (y<z)	•	•	•	•	•	•	•	•	•	•	0.00	
4: if (x<y)	•	•			•	•		•		•	0.00	
5: m = y;		•									0.00	
6: else if (x<z)	•				•	•		•		•	0.00	
7: m = x; //fixed	•				•			•		•	0.00	
8:else			•	•			•		•		0.00	
9: if (x>y)			•	•			•		•		0.00	
10: m = y; //fixed			•				•		•		0.00	
11: else if (x>z)				•							0.00	
12: m = x;											0.00	
13:print(“Middle number is:”, m);	•	•	•	•	•	•	•	•	•	•	0.00	
} Pass/fail Status	P	P	P	P	P	P	P	P	P	P		

Empirical Study

Measure

Percentage of program to be examined to find fault

Subjects

Siemens Suite

Program	LOC	Faulty Versions (single fault)	Test Cases
Print_tokens	472	7	4056
Print_tokens_2	399	10	4071
Replace	512	32	5542
Schedule	292	9	2650
Schedule_2	301	10	2680
Tcas	141	41	1578
Tot_info	440	23	1054
Space	6000	33	13585

Empirical Study

Method

- For each program and test suite, compute suspiciousness of each statement using Tarantula
- Compute percentage of program examined to find fault, using suspiciousness to order search
- Use results of published studies on same subjects

General Technique—Tarantula

mid() {	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	suspiciousness	% program examined
int x,y,z,m;	3,3,5	1,2,3	3,2,2	5,5,5	1,1,4	5,3,4	3,2,1	2,1,3	5,4,2	5,2,6		
1:read(“Enter 3 integers:”,x,y,z);	•	•	•	•	•	•	•	•	•	•	0.50	70
2:m = z;	•	•	•	•	•	•	•	•	•	•	0.50	70
3:if (y<z)	•	•	•	•	•	•	•	•	•	•	0.50	70
4: if (x<y)	•	•			•	•		•		•	0.43	80
5: m = y;		•									0.00	100
6: else if (x<z)	•				•	•		•		•	0.50	70
7: m = y;	•				•			•		•	0.60	30
8:else			•	•			•		•		0.60	30
9: if (x>y)			•	•			•		•		0.60	30
10: m = z;			•				•		•		0.75	10
11: else if (x>z)				•							0.00	100
12: m = x;											0.00	100
13:print(“Middle number is:”, m);	•	•	•	•	•	•	•	•	•	•	0.50	70
} Pass/fail Status	P	P	P	P	P	P	F	F	F	F		

Empirical Study

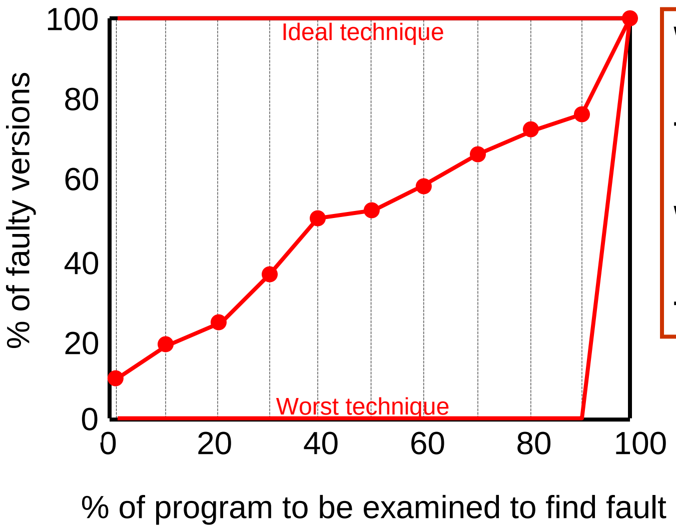
Method

- For each program and test suite, compute suspiciousness of each statement using Tarantula
- Compute percentage of program examined to find fault, using suspiciousness to order search
- Use results of published studies on same subjects

Techniques compared

- Tarantula [Jones, Harrold, Stasko, ICSE02,ASE05]
- Set-based, Nearest Neighbor [Renieris, Reiss, ASE03]
- Cause Transitions [Cleve, Zeller, ICSE05]
- Statistical [Liblit et al., PLDI05]

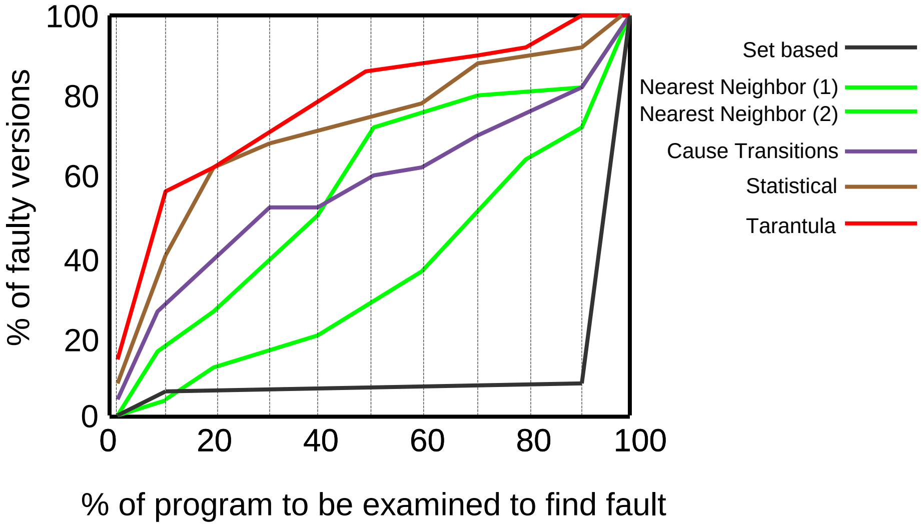
Reporting Technique



What would be the ideal technique?

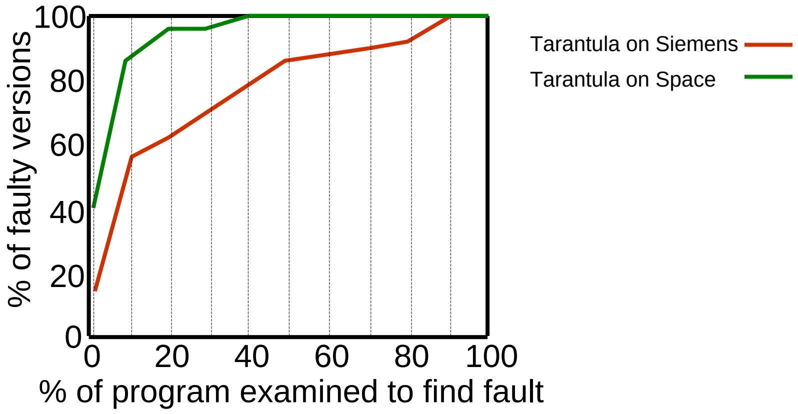
What would be the worst technique?

Results on Siemens



Threats to Validity

- Generalization

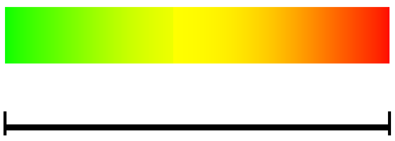


Visualization

For statement *s*:

Hue (color)

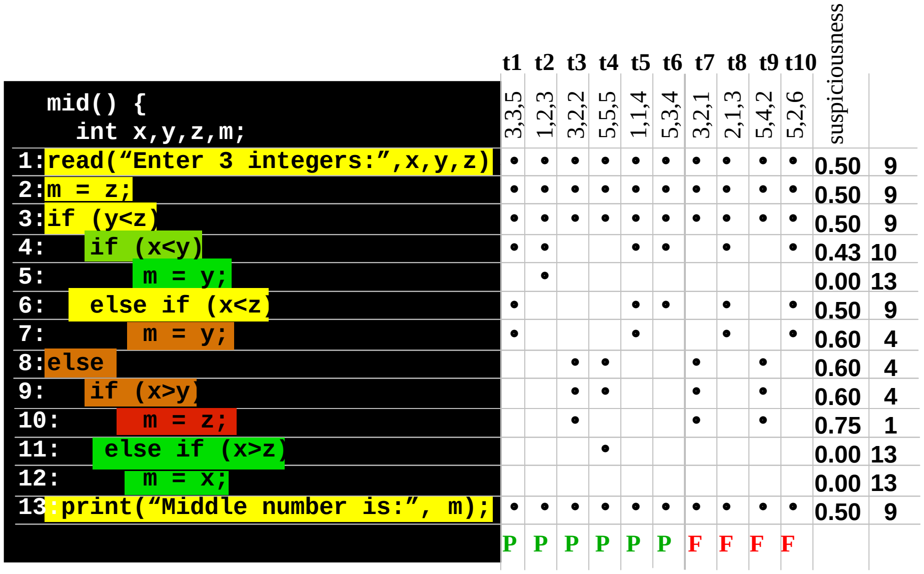
summarizes pass/fail results of test cases that executed *s*



Least suspicious

Most suspicious

Coloring Statements

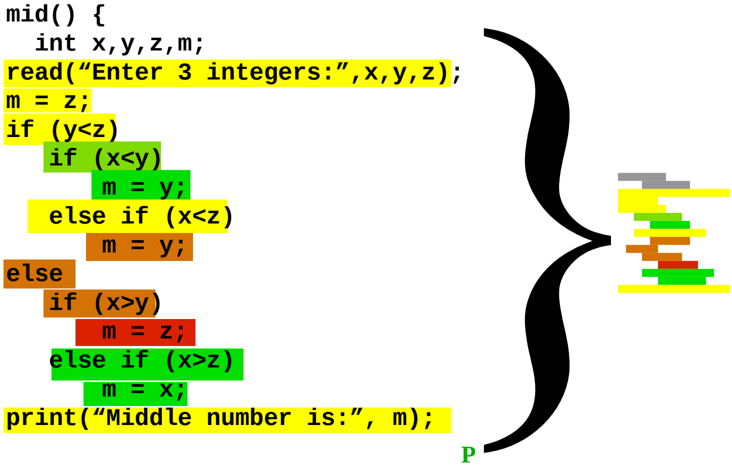


mid() { int x,y,z,m;	t1 3,3,5	t2 1,2,3	t3 3,2,2	t4 5,5,5	t5 1,1,4	t6 5,3,4	t7 3,2,1	t8 2,1,3	t9 5,4,2	t10 5,2,6	suspiciousness	
1:read(“Enter 3 integers:”,x,y,z)	•	•	•	•	•	•	•	•	•	•	0.50	9
2:m = z;	•	•	•	•	•	•	•	•	•	•	0.50	9
3:if (y<z)	•	•	•	•	•	•	•	•	•	•	0.50	9
4: if (x<y)	•	•			•	•		•		•	0.43	10
5: m = y;		•									0.00	13
6: else if (x<z)	•				•	•		•		•	0.50	9
7: m = y;	•				•			•		•	0.60	4
8:else			•	•			•		•		0.60	4
9: if (x>y)			•	•			•		•		0.60	4
10: m = z;			•				•		•		0.75	1
11: else if (x>z)				•							0.00	13
12: m = x;											0.00	13
13:print(“Middle number is:”, m);	•	•	•	•	•	•	•	•	•	•	0.50	9
	P	P	P	P	P	P	F	F	F	F		

File-level View

SeeSoft view

- each pixel represents a character in the source

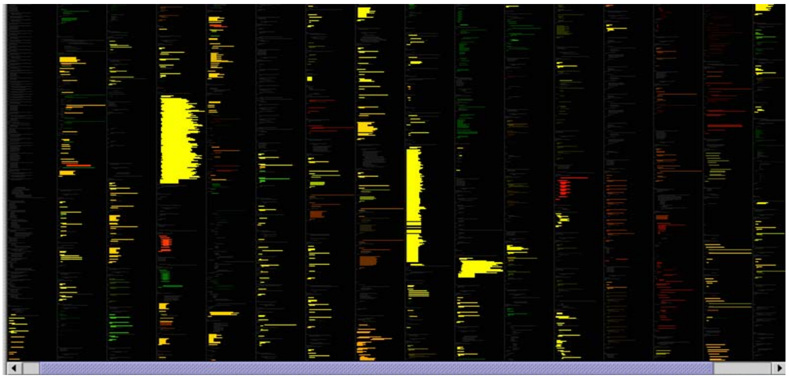


```
mid() {
  int x,y,z,m;
read(“Enter 3 integers:”,x,y,z);
m = z;
if (y<z)
   if (x<y)
       m = y;
  else if (x<z)
       m = y;
else
   if (x>y)
       m = z;
   else if (x>z)
       m = x;
print(“Middle number is:”, m);
```

File-level View

SeeSoft view

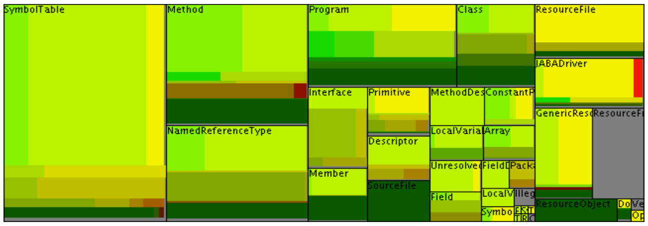
- **each pixel represents a character in the source**



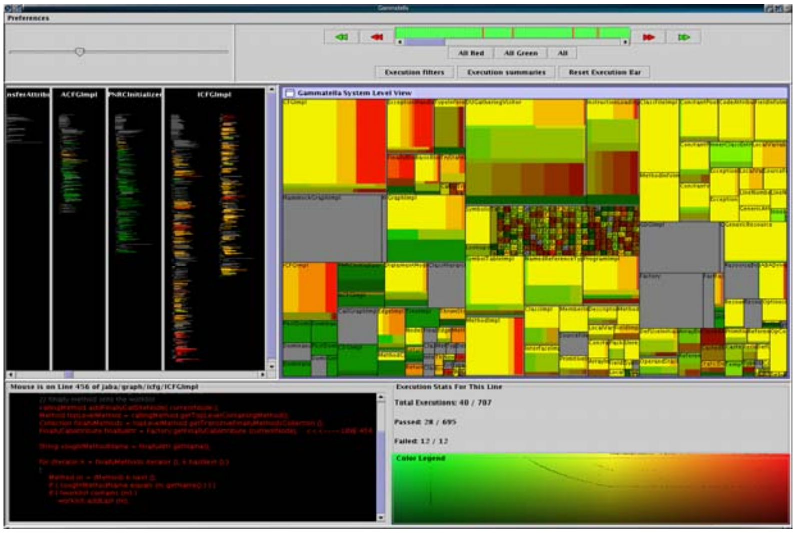
System-level View

TreeMap view

- each node
  - represents a file
  - is divided into blocks representing color of statements



Tarantula



Mouse is on Line 456 of jaba/graph/icfg/ICFGImpl

```
        // finally method sets the worklist
        callingMethod.addFinallyCallSiteNode( currentNode );
        Method topLevelMethod = callingMethod.getTopLevelContainingMethod();
        Collection finallyMethods = topLevelMethod.getTransitiveFinallyMethodsCollection ();
        FinallyCallAttribute finallyAttr = Factory.getFinallyCallAttribute (currentNode);    <<<<------ line 456

        String soughtMethodName = finallyAttr.getName();

        for (Iterator k = finallyMethods.iterator (); k.hasNext ();)
        {
          Method m = (Method) k.next ();
          if ( soughtMethodName.equals (m.getName() ) )
          if ( !worklist.contains (m) )
            worklist.addLast (m);
```

Execution Stats For This Line

Total Executions: 40 / 707

Passed: 28 / 695

Failed: 12 / 12