SOFTENG 254:
Quality Assurance

Lecture 5a: Data-flow Testing 2

Paramvir Singh
School of Computer Science

Potential Assessment Question

Consider the following method:

```
    public static boolean isPrime(int n) {
A       boolean prime =true;
B       int i =2;
C       while (i <n) {
D           if (n %i ==0) {
E               prime = false;
            }
F           i++;
        }
G       return prime;
    }
```

Which one of the following lists *all vertices with definitions*?

(a) A, B

(b) A, B, E

(c) A, B, E, F

(d) none of the above

Justify your answer.

Agenda

- PAQ
- Admin
  - Assignment 1?
    - Due Date – this Friday!
- Data-flow testing — choosing tests based on how values are propagated through code

Previously in SOFTENG 254

- computation consists of manipulating values, moving them among different memory locations
- (almost) every failure can be traced to a wrong value being in the wrong place at the wrong time

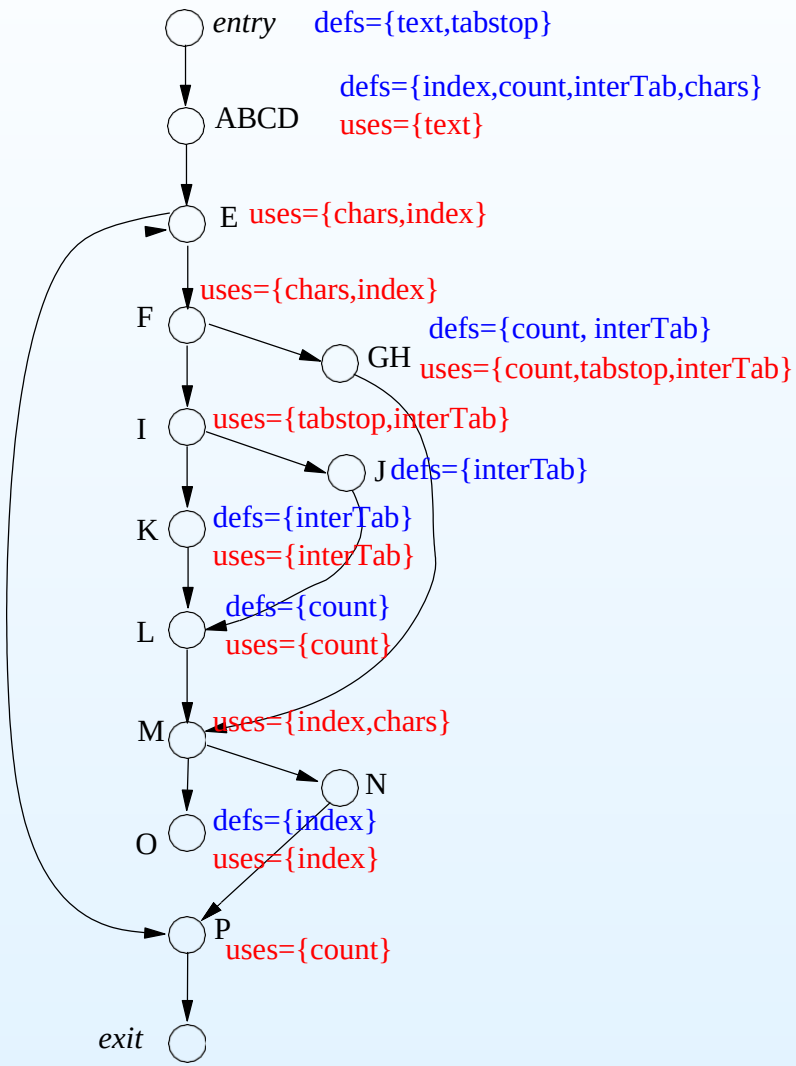
⇒ look at where the values come from and where they go — “data flow”

- choose a subset of “all paths” based on data flow
- annotate vertices in CFGs with indications as to what variables are “defined” and what variables are “used” at those vertices

Example: Variation B

```
    public static int leadingSpacesCount(String text, int tabstop) {
A     int index =0;
B     int count =0;
C     int interTab = 0;
D     char[] chars =text.toCharArray();
E     while (Character.isWhitespace(chars[index])) {
F       if (chars[index] =='\t') {
G         count +=tabstop - interTab;
H         interTab = 0;
        }else {
I         if (interTab ==tabstop - 1) {
J           interTab = 0;
          }else {
K           interTab++;
         }
          count++;
        }
M       if (index ==chars.length-1){
N         break;
        }
O       index++;
      }
P     return count;
    }
```

Defs and Uses in Variation B



du pairs

vertex	variable	reachable uses
entry	text	A
entry	tabstop	G, I
A	index	E, F, M, O
A	count	G, L, P
A	interTab	G, I, K
A	chars	E, F, M
G	count	G, L, P
G	interTab	G, I, K
J.	interTab	G, I, K
K.	interTab	G, I, K
L.	count	G, L, P
O	index	E, F, M, O

- **reachable** use — there is a def-clear path from the definition to this use
- a du pair with respect to a variable x is a pair of vertices (v_d, v_u) such that there is a du path with respect to x from v_d to v_u.

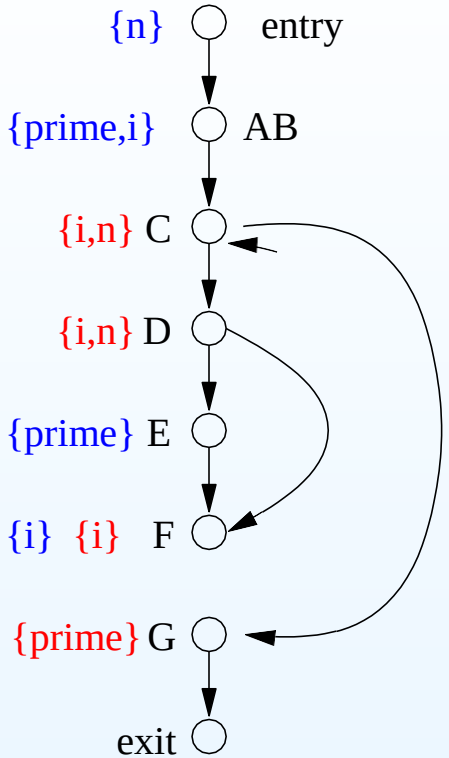
Some du-paths

vertex	variable	du-paths
entry	text	(e,A)
entry	tabstop	(e,A,E,F,G), (e,A,E,F,I)
A	index	(A,E), (A,E,F), (A,E,F,G,M), (A,E,F,I,J,L,M) (A,E,F,I,K,L,M), (A,E,F,G,M,O), (A,E,F,I,J,L,M,O)
O	index	(O,E), (O,E,F), (O,E,F,G,M), (O,E,F,I,J,L,M), (O,E,F,I,K,L,M), (O,E,F,G,M,O), (O,E,F,I,J,L,M,O)

- A path $q = (w_0, w_1, \ldots, w_m)$ is a prefix of a path $p = (v_0, v_1, v_2, \ldots, v_k)$ when $\forall 0 \leq i \leq m, w_i = v_i$
  - a test path that visits every vertex and edge in p must also visit every vertex and edge in a prefix q
  - *usually* can remove prefix du-paths depends on feasibility of p
- there can be multiple du-paths between a du pair
- there can be *many* du-paths

Example 2

```
    public static boolean isPrime(int n) {
A       boolean prime =true;
B       int i =2;
C       while (i <n) {
D           if (n %i ==0) {
E               prime = false;
            }
F           i++;
        }
G       return prime;
    }
```



vertex	variable	du-path
entry	n	(e,A,C)[0], (e,A,C,D)[1]
A	prime	(A,C,G)[2]
A	i	(A,C)[3], (A,C,D)[4], (A,C,D,F)[5], (A,C,D,E,F)[6]
E	prime	(E,F,C,G)[7]
F	i	(F,C)[8], (F,C,D)[9], (F,C,D,F)[10], (F,C,D,E,F)[11]

Data Flow Coverage Criteria

All du-paths visit every du path for every variable

- strongest data flow strategy
- not as strong as all paths

All uses for every use, visit at least one du path from every definition to that use

All-p-uses/some-c-uses for every definition, visit at least one du path to every predicate use, then for any definition not covered add a du path to a computation use

All-c-uses/some-p-uses for every definition, visit at least one du path to every computation use, then for any definition not covered add a du path to a predicate use

All definitions for every definition, visit at least one du path

(and others)

Example: all du-paths

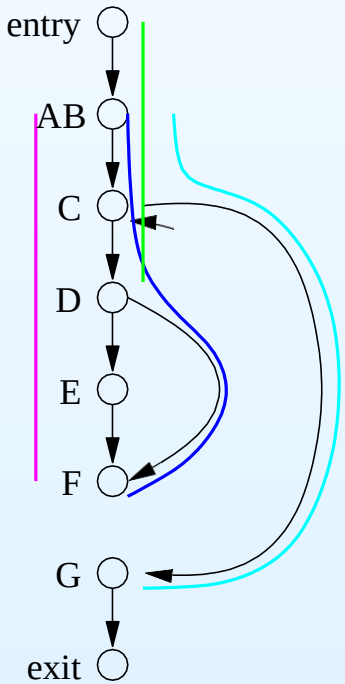
vertex	variable	du-path
entry	n	(e,A,C)[0], (e,A,C,D)[1]
A	prime	(A,C,G)[2]
A	i	(A,C)[3], (A,C,D)[4], (A,C,D,F)[5], (A,C,D,E,F)[6]
E	prime	(E,F,C,G)[7]
F	i	(F,C)[8], (F,C,D)[9], (F,C,D,F)[10], (F,C,D,E,F)[11]

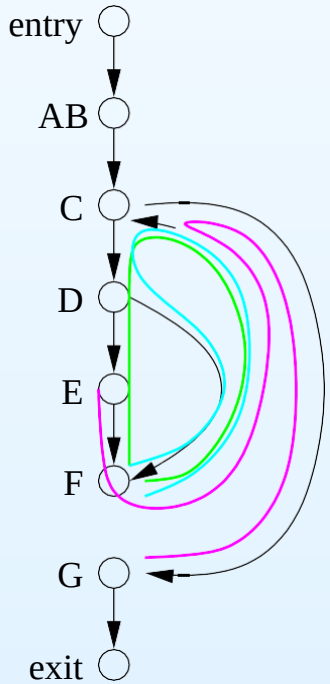
n	test path	du-paths
2	(e,A,C,G)	0, 1, 2, 3
3	(e,A,C,D,F,C,G)	0, 1, 3, 4, 5, 8
4	(e,A,C,D,E,F,C,D,E,F,C,G)	0, 1, 3, 4, 6, 7, 8, 9, 11
6	(e,A,C,D,E,F,C,D,E,F,C,D,F,C,D,F,C,G)	0, 1, 3, 4, 6, 8, 9, 10, 11

Example: all du-paths continued

vertex	variable	du-path
entry	n	(e,A,C)[0], (e,A,C,D)[1]
A	prime	(A,C,G)[2]
A	i	(A,C)[3], (A,C,D)[4], (A,C,D,F)[5], (A,C,D,E,F)[6]
E	prime	(E,F,C,G)[7]
F	i	(F,C)[8], (F,C,D)[9], (F,C,D,F)[10], (F,C,D,E,F)[11]

prefixes





Example: All-p-uses/some-c-uses

vertex	variable	du-path
entry	n	(e,A,C)[0], (e,A,C,D)[1]
A	prime	(A,C,G)[2]
A	i	(A,C)[3], (A,C,D)[4], (A,C,D,F)[5], (A,C,D,E,F)[6]
E	prime	(E,F,C,G)[7]
F	i	(F,C)[8], (F,C,D)[9], (F,C,D,F)[10], (F,C,D,E,F)[11]

c-use, p-use

- n=4, test path (e,A,C,D,E,F,C,D,E,F,C,G)
  - visits du-paths 0, 1, 3, 4, 6, 7, 8, 9, 11
  - covers definition n to p-uses on C and D, definition i on A to p-uses on C and D, definition i on F to p-uses on C and D
  - Also covers definition prime on E to c-use on G
- Need test cases to cover definition of prime on A to some computation use (e.g. n=2)
- still several du-paths not covered

Complications

- infeasible paths
  - relax requirements of coverage criteria
- use before def in the same vertex (e.g. $i=i+1$)
  - improve model

Practicality of all du-paths

- All du-paths requires 2^t in the worst case, where t is the number of "conditional transfers" (conditions)

  E. J. Weyuker. The complexity of data flow criteria for test data selection. *Information Processing Letters*, 19:103–109, August 1984.
- Mostly the worst-case doesn't seem to happen much

  J. Bieman and J. Schultz, An empirical evaluation (and specification) of the all-du-paths testing criterion *IEE Software Engineering Journal*, vol. 7, no. 1, pp. 43–51, Jan. 1992.
- All du-paths may not produce a better test suite (or even more tests) than branch coverage, but when it does its usually much better.

  Binder, *Testing Object-Oriented Systems* p389

Quality of all du paths


```
class VendingMachine { // Based on example from Binder,
  ...                  // Testing Object-Oriented Systems. p398
  private int _nTwoDollar;
  private int _nOneDollar;
  private int _n50Cent;
  private int _n20Cent;
  private int _n10Cent;
  private int _depositInCents;
   private void computeChange(int priceInCents) {
     if (_depositInCents >price) {
       int changeDue;
       changeDue = _depositInCents - price;
       _nTwoDollar =changeDue / 200; // integer division
       changeDue = changeDue - _nTwoDollar * 200;
       _nOneDollar =changeDue / 100;
       changeDue = changeDue - _nOneDollar * 100;
       _n50Cent = changeDue / 50;
       changeDue = changeDue - _n50Cent * 50;
       _n20Cent =changeDue / 20;
       changeDue = changeDue - _n50Cent * 20;
       _n20Cent =changeDue / 10;
       changeDue = changeDue - _n50Cent * 50;
     }
   }
}
```

Key Points

- A failure is often due to the wrong value being used somewhere.
- Finding the fault that caused the failure (“debugging”) involves finding out where that wrong value came from
- ⇒ a good place to look for possible faults is along the paths between the source of the value (definition) and its use
- Procedure
  - Identify *definitions* of values
  - Identify *uses* of those definitions
  - Identify (control flow) paths from definitions of values to their uses (du-paths)
  - Choose a set of du-paths based on the chosen testing strategy
  - Create a test suite that causes all of the chosen du-paths to be followed.
- Doing any of this without tool support will be difficult!