Graph Coverage for Source Code

Data Flow Graph Coverage for Source Code

def-use，Definition

- we define what constitutes a *def* and what constitutes a *use*.
- A *def* is a location in the program where a value for a variable is stored into memory (assignment, input, etc.).
- A *use* is a location where a variable’s value is accessed.

Where does «def» occur?

- x appears on the left side of an assignment statement
- x is an actual parameter in **a call** site and its value is changed within the method
- x is a formal parameter of a method (an implicit def when the method begins

execution)

- x is an input to the program

Where does «use» ocur?

- ❑ x appears on the right side of an assignment statement
- ❑ x appears in a conditional test (such a test is always associated with at least two edges)
- ❑ 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 or returned as a parameter

Some questions about def definition

- ❑ Is a def of an array variable

  a def of the entire array? or

  a def of just the element being referenced?
- ❑ should the def consider

  the entire object ? or

  only a particular instance variable of the object?
- ❑ If two variables reference the same location, how is the analysis done?
- ❑ What is the relationship between coverage of the original source code, coverage of the optimized source code, and coverage of the machine code?

Graph Coverage for Design Elements

- ❑Use of data abstraction and object oriented software has increased importance on modularity and reuse.
- ❑Therefore testing of software based design (design elements) is more important that past
  - ❖ They are usually associated with integration testing.
- ❑Since modularity, the software components can be tested independently.

DU Pair

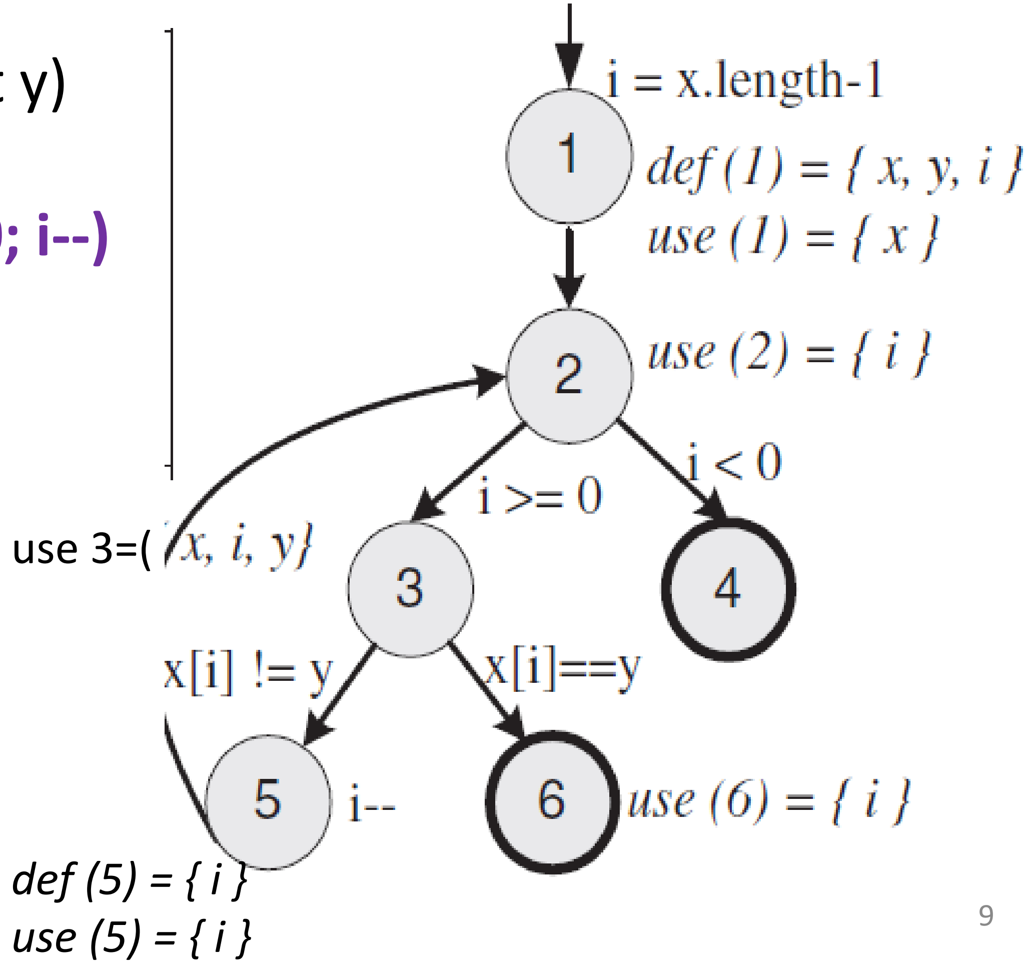
If a def and a use appear on the same node, then it is only a DU-pair if the def occurs after the use and the node is in a loop

def clear & all uses Definitions

- A definition *d* for a variable *x* *reaches* a use *u* if there is a path from *d* to *u* that has no other definitions of *x* (*def-clear*).
- The *all-uses (AU)* criterion requires tests to tour at least one subpath from each definition to each reachable use.

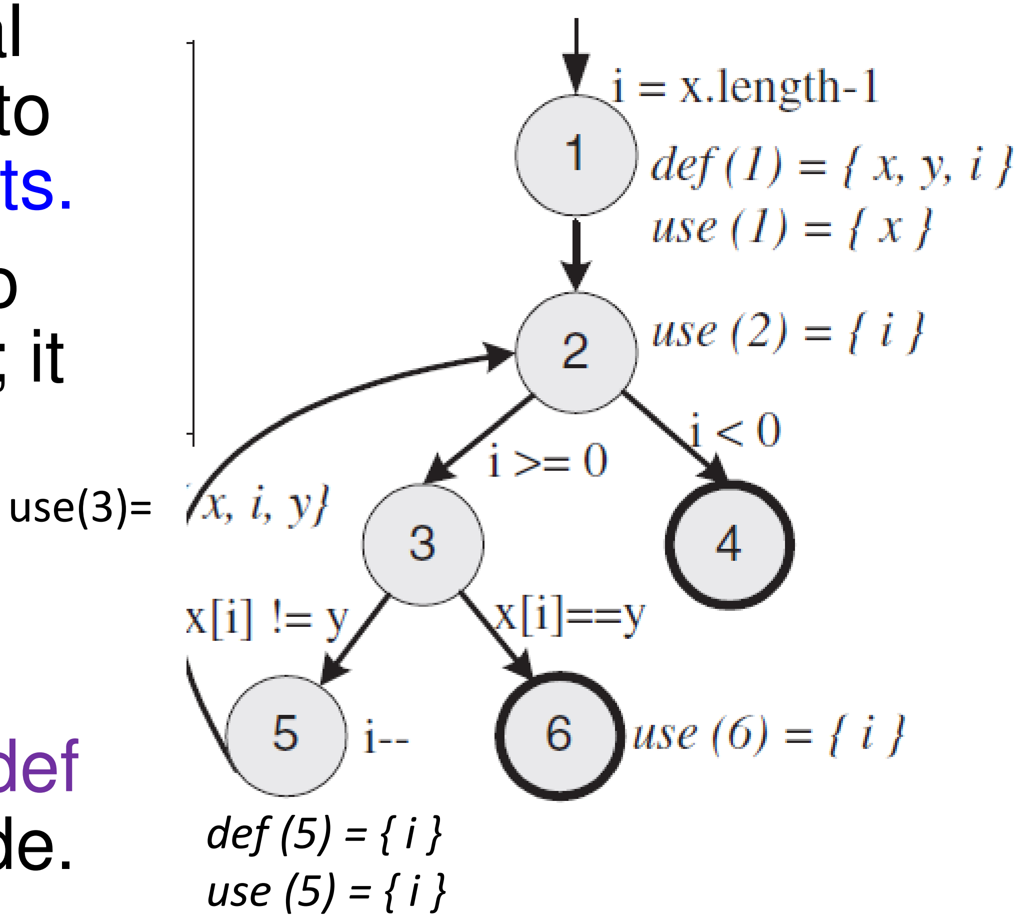
// EXAMPLE: return index of the last element in x that equals y.
// if y is not in x, return -1.

```
public int findLast (int []x, int y)
{
   for (int i = x.length-1; i>=0; i--)
  {
      if (x[i] == y)
        return i;
   }     return -1;
```



Annotated Control Graph

- Nodes 4 and 6 are final nodes, corresponding to the `return` statements.
- Node 2 is introduced to capture the `for` loop; it has no executable statements.
- DU (def-use) pairs are shown as a variable name followed by the def node, then the use node.



Def -Use Pairs = { (1, 1,x), (1,3,x), (1,3,y), (1, 2,i), (1, 3,i), (1,5,i), (1,6,i), (5, 2,i), (5, 3,i), (5, 6,i), (5, 5,i)}