How to know whether we have tested our program sufficiently

Whitebox vs blackbox tests

- Our examples with “division.exe”, “sortArray”, “min” are **blackbox** tests
  - Blackbox = We don’t look at the source code of the program to design tests
  - Very hard to tell whether we have tested the program sufficiently: How do we know that “division.exe” does not have a bug for $a = 1353885, b = -45242$?
- In this course, we will also do **whitebox** tests
  - Whitebox = We can see the source code of the program.
  - Seeing the source code can help us to choose good test values!

Testing everything?

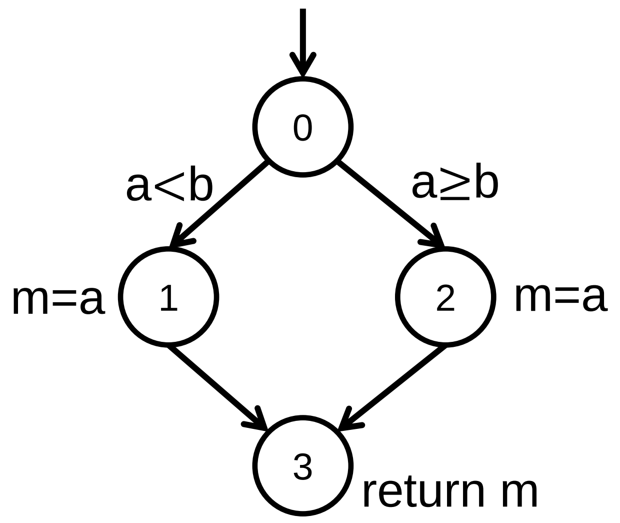
- Our faulty code:

```
int min(int a, int b) {
    int m;
    if(a<b)
        m=a;
    else
        m=a;
    return m;
}
```

- Input domain: $\mathbb{Z} \times \mathbb{Z}$
- Let's choose $a = 3, b = 5$. Result: 4. The program works!?
- How do we make sure that we have not forgotten a test case during our tests?
- **Obvious truth of testing 1: We can only find a bug in a program if the program reaches the faulty location with our test input values.**

Control Flow Graph and Node Coverage

```
int min(int a, int b) {
    int m;
    if(a<b)
        m=a;
    else
        m=a;
    return m;
}
```



- The *control flow graph* (CFG) tells us if our tests are complete:
  - Test case 1: a=3, b=5

    The program will go through nodes 0,1,3 of the CFG

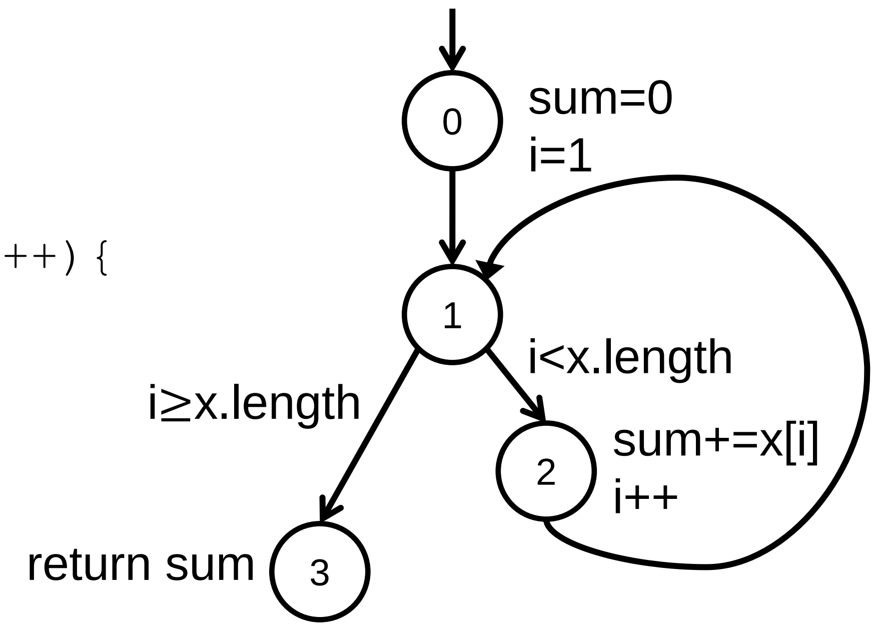
    ⇒ We have not tested what happens in node 2!
  - Test case 2: a=5, a=3

    The program will go through nodes 0,2,3 of the CFG

→ We have achieved 100% *node coverage*

Control Flow Graph of a for-loop

```
int sum(int[] x) {
    int sum=0;
    for(int i=1;i<x.length;i++){
        sum+=x[i];
    }
    return sum;
}
```

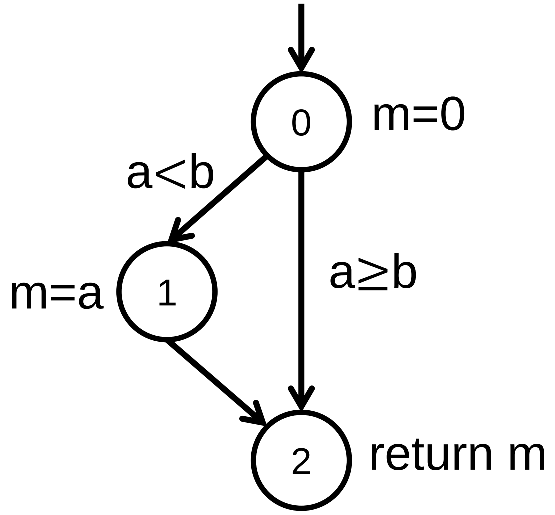


- Remarks:
  - Nodes can have more than one statement in one node. In the Control Flow Graph, we are only interested in changes of the control flow (“branching”).
    Group of statements without branching = “*Basic block*”
  - Nodes can have no statements. Such nodes only make a decision.

Edge Coverage

- If we have 100% node coverage, can we be sure that our program is correct?

```
int min(int a, int b) {
    int m=0;
    if(a<b) {
        m=a;
    }
    return m;
}
```



- We can cover all statements with the test case a=3, b=5

  → Program goes through nodes 0,1,2 ⇒ 100% node coverage

  But: We have not tested the direct path 0→2 !

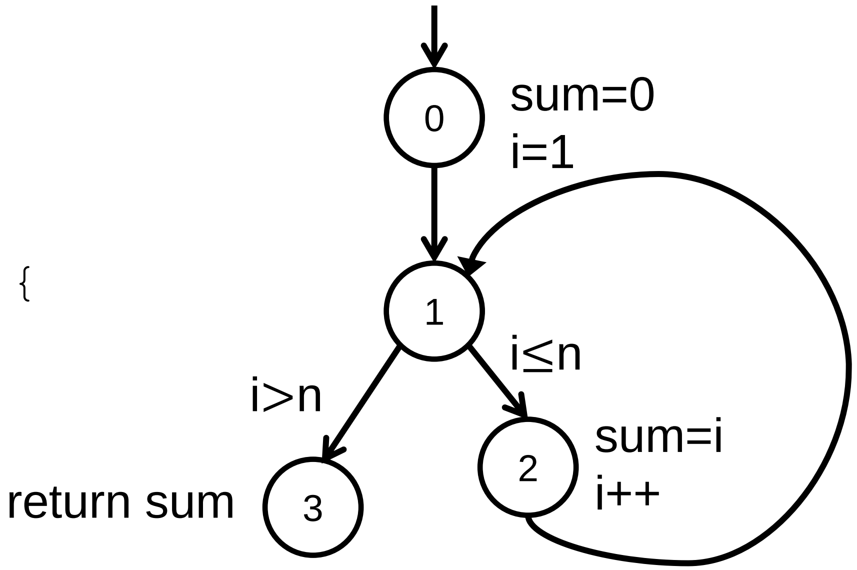
- Instead of 100% node coverage, our goal should be 100% *edge coverage*

  ⇒ We have to choose test cases that cover all <u>edges</u> in the control flow graph.

Is edge coverage enough?

- If we have 100% edge coverage, can we be sure that our program is correct?

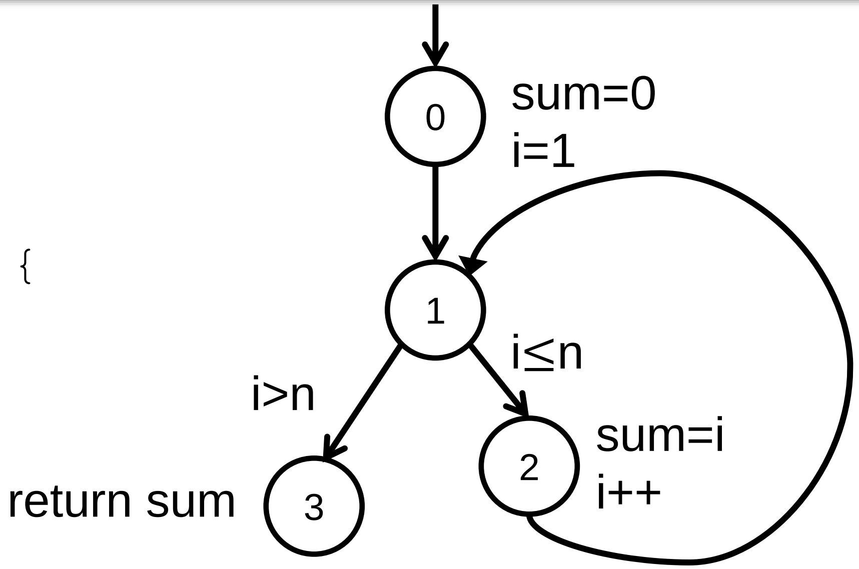
```
// sum of 1..n
int sum(int n) {
    int sum=0;
    for(int i=1;i<=n;i++){
        sum=i;
    }
    return sum;
}
```



- Test case $n = 0$: Covered edges: $0 \rightarrow 1, 1 \rightarrow 3$. Result is correct: sum=0
- Test case $n = 1$: Covered edges: $0 \rightarrow 1, 1 \rightarrow 2, 1 \rightarrow 3$. Result is correct: sum=1
- We have covered all edges with our two tests, but the program is wrong for $n > 1$!

Path Coverage

```
// sum of 1..n
int sum(int n) {
   int sum=0;
   for(int i=1;i<=n;i++){
      sum=i;
   }
   return sum;
}
```



- To be sure that our program is correct, we would have to test all possible *paths* through the code:

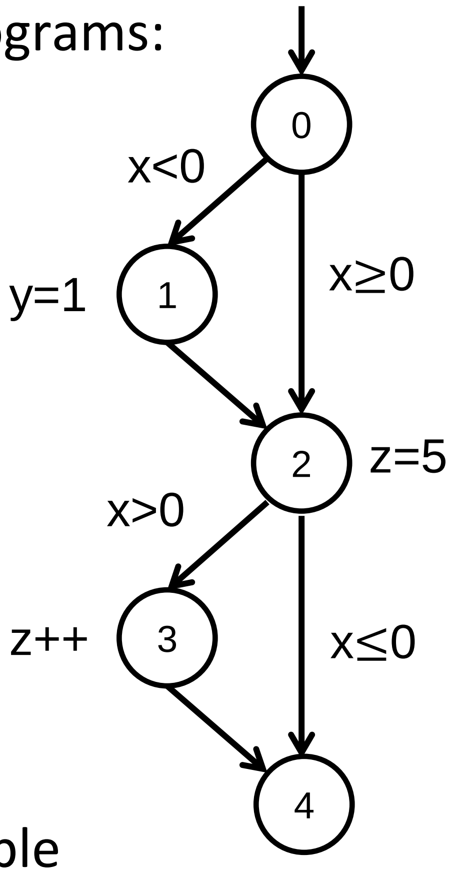
  <u>n=0</u>: $0 \rightarrow 1 \rightarrow 3$, **<u>n=1</u>**: $0 \rightarrow 1 \rightarrow 2 \rightarrow 1 \rightarrow 3$, **<u>n=2</u>**: $0 \rightarrow 1 \rightarrow 2 \rightarrow 1 \rightarrow 2 \rightarrow 1 \rightarrow 3, \ldots$

- In practice, 100% path coverage is often not feasible if your program has loops. Too many possible paths! In that case, people are just happy with node coverage or edge coverage.

Unreachable paths

- If your goal is to design tests with 100% path coverage, you should be aware that not all paths are possible in some programs:

```
if(x<0) {
    y=1;
}
z=5;
if(x>0) {
    z++;
}
```



- In this program, the path $0 \to 2 \to 4$ is not possible
- Therefore, when we say “n% path coverage“, we usually mean “n% of the *reachable* paths“

Quiz

Yes or No:

a) If we have 100% edge coverage, we also have 100% node coverage.

b) In a program without loops, all existing paths in the CFG can be checked.

Answers to Quiz

Yes or No:

a) If we have 100% edge coverage, we also have 100% node coverage.

   Yes: 100% edge coverage $\Rightarrow$ 100% node coverage

b) In a program without loops, all existing paths in the CFG can be checked.

   No. The program can have unreachable paths.

Coverage test on Inginious

- Inginious uses the tool JaCoCo for coverage tests
- JaCoCo calculates two metrics:

  - JVM bytecode instruction coverage: this is node coverage.

    Note: JVM bytecode instruction ≠ Java statement
    For example, the Java statement

    ```
    a=b+2;
    ```

    is compiled to four bytecode instructions!

  - Branch coverage: this is edge coverage for `if` and `switch` statements

Software testing = doing coverage tests ?

- The goal of testing is not to reach 100% coverage!
  - Coverage is not a goal. Coverage is a tool that helps you to define test cases

- What to do in practice:
  1. Identify the input domain
  2. Divide the input domain into blocks
  3. Create test cases with test values that come from the blocks
  4. Run tests
  5. Check the coverage. Not happy? Go back to step 1 or 2.