CSE P 504

Advanced topics in Software Systems

Fall 2022

Coverage-based Testing

October 17, 2022

Today

- Recap: Git bisect exercise
- Software testing 101
- Test adequacy: structural code coverage
    - Statement coverage
    - Decision coverage
    - Condition coverage
    - Modified condition and decision coverage (MCDC)
- In-class exercise 2

Recap: git bisect

Questions

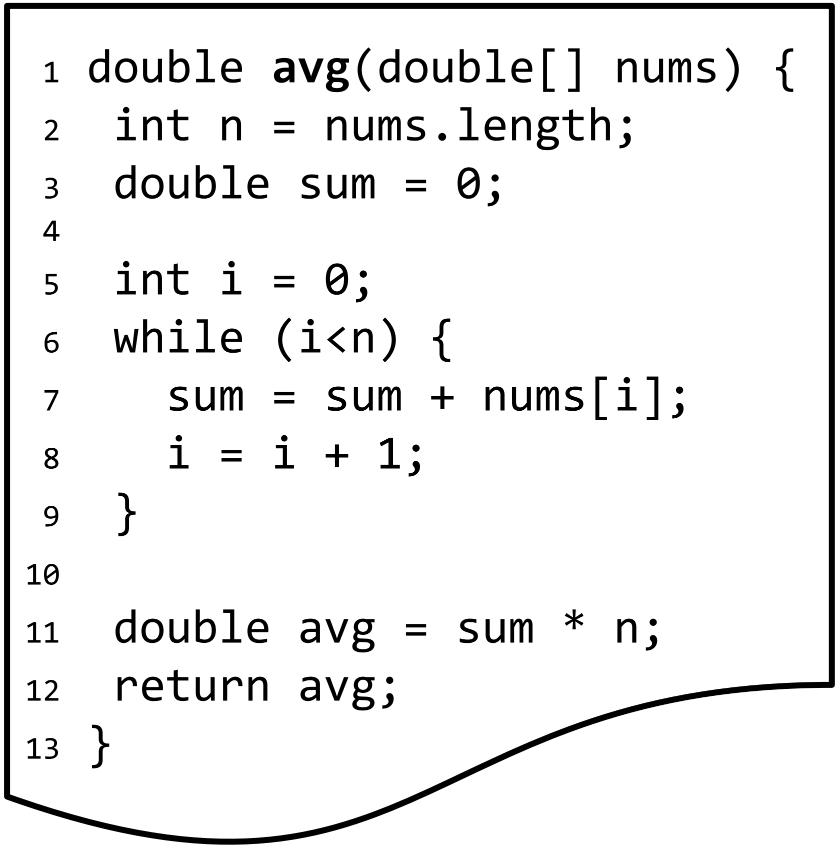
- How could the developers improve the build or testing infrastructure to notice test failures in the future?
- Which git command can you use to undo a defect-inducing commit? Briefly explain what problem may generally occur when undoing a commit and what best practices mitigate this problem.
- Can you undo the defect-inducing commit using the proposed git command?

Meta-level discussion

- Is Git bisect a realistic choice for the JavaParser example?
- I don’t use Java, so why should I care?
- Slack participation is great!

Software testing 101

Software testing vs. software debugging



```
double avg(double[] nums) {
 int n = nums.length;
 double sum = 0;

 int i = 0;
 while (i<n) {
   sum = sum + nums[i];
   i = i + 1;
 }

 double avg = sum * n;
 return avg;
}
```

Software testing vs. software debugging

```
double avg(double[] nums) {
 int n = nums.length;
 double sum = 0;

 int i = 0;
 while (i<n) {
   sum = sum + nums[i];
   i = i + 1;
 }

 double avg = sum * n;
 return avg;
}
```

Testing: is there a bug?

```
@Test
public void testAvg() {
  double nums =
      new double[]{1.0, 2.0, 3.0});
  double actual = Math.avg(nums);
  double expected = 2.0;
  assertEquals(expected,actual,EPS);
}
```

Software testing vs. software debugging

```
double avg(double[] nums) {
 int n = nums.length;
 double sum = 0;

 int i = 0;
 while (i<n) {
   sum = sum + nums[i];
   i = i + 1;
 }

 double avg = sum * n;
 return avg;
}
```

Testing: is there a bug?

```
@Test
public void testAvg() {
  double nums =
       new double[]{1.0, 2.0, 3.0});
  double actual = Math.avg(nums);
  double expected = 2.0;
  assertEquals(expected,actual,EPS);
}
```

FAIL

```
testAvg failed: 2.0 != 18.0
```

Software testing vs. software debugging

```
double avg(double[] nums) {
 int n = nums.length;
 double sum = 0;

 int i = 0;
 while (i<n) {
   sum = sum + nums[i];
   i = i + 1;
 }

 double avg = sum * n;
 return avg;
}
```

Testing: is there a bug?

```
@Test
public void testAvg() {
  double nums =
      new double[]{1.0, 2.0, 3.0});
  double actual = Math.avg(nums);
  double expected = 2.0;
  assertEquals(expected,actual,EPS);
}
```

FAIL

```
testAvg failed: 2.0 != 18.0
```

Debugging: where is the bug?
how to fix the bug?

Software testing vs. software debugging

```
double avg(double[] nums) {
 int n = nums.length;
 double sum = 0;

 int i = 0;
 while (i<n) {
   sum = sum + nums[i];
   i = i + 1;
 }

 double avg = sum * n;
 return avg;
}
```

Testing: is there a bug?

```
@Test
public void testAvg() {
  double nums =
      new double[]{1.0, 2.0, 3.0});
  double actual = Math.avg(nums);
  double expected = 2.0;
  assertEquals(expected,actual,EPS);
}
```

FAIL

```
testAvg failed: 2.0 != 18.0
```

Debugging: where is the bug?

how to fix the bug?

Software testing

Software **testing** can **show** the **presence of defects**, but **never** show their **absence**! (Edsger W. Dijkstra)

Software testing

Software **testing** can **show** the **presence of defects**, but **never** show their **absence**! (Edsger W. Dijkstra)

- A good test is one that fails because of a defect.

How do we come up with good tests?

Two strategies: black box vs. white box

Black box testing

- The system is a black box (can’t see inside).
- No knowledge about the internals of a system.
- Create tests solely based on the specification (e.g., input/output behavior).

White box testing

- Knowledge about the internals of a system.
- Create tests based on these internals (e.g., exercise a particular part or path of the system).

Unit testing, integration testing, system testing

Unit testing

- Does each unit work as specified?

Integration testing

- Do the units work when put together?

System testing

- Does the system work as a whole?

Unit testing, integration testing, system testing

Unit testing

- Does each unit work as specified?

Integration testing

- Do the units work when put together?

System testing

- Does the system work as a whole?

Our focus: unit testing

Unit testing

- A **unit** is the **smallest testable part** of the software system (e.g., a method or a function).
- **Goal**: Verify that each software unit performs as specified.
- **Focus**:
  - Individual units (not the interactions between units).
  - Usually input/output relationships.

Software testing

Software **testing** can show the **presence of defects**, but **never** show their **absence**! (Edsger W. Dijkstra)

- A good test is one that fails because of a defect.

When should we stop testing if no (new) test fails?

Test effectiveness

Ratio of detected defects is the best effectiveness metric!

Problem

- The set of defects is unknowable.

Solution

- Use a proxy metric, for example code coverage.

Test adequacy: structural code coverage

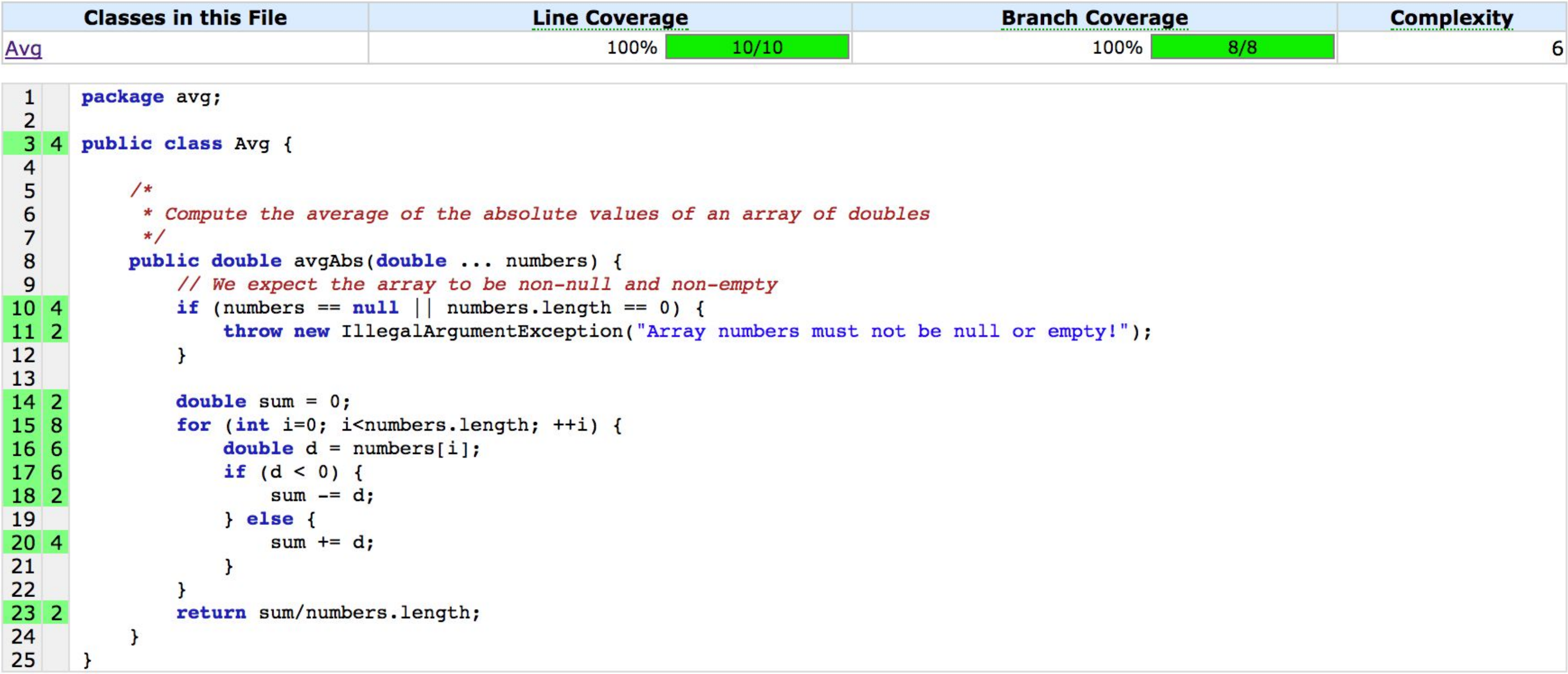
Structural code coverage: motivating example

Average of the absolute values of an array of doubles

```
public double avgAbs(double ... numbers) {

  // We expect the array to be non-null and non-empty
  if (numbers == null || numbers.length == 0) {
    throw new IllegalArgumentException("Array numbers must not be null or empty!");
  }

  double sum = 0;
  for (int i=0; i<numbers.length; ++i) {
    double d = numbers[i];
    if (d < 0) {
      sum -= d;
    } else {
      sum += d;
    }
  }

  return sum/numbers.length;
}
```

What tests should we write for this method?

Structural code coverage: motivating example



Classes in this File	Line Coverage		Branch Coverage		Complexity
Avg	100%	10/10	100%	8/8	6

```
   package avg;

 4 public class Avg {

       /*
        * Compute the average of the absolute values of an array of doubles
        */
       public double avgAbs(double ... numbers) {
           // We expect the array to be non-null and non-empty
 4         if (numbers == null || numbers.length == 0) {
 2             throw new IllegalArgumentException("Array numbers must not be null or empty!");
           }

 2         double sum = 0;
 8         for (int i=0; i<numbers.length; ++i) {
 6             double d = numbers[i];
 6             if (d < 0) {
 2                 sum -= d;
               } else {
 4                 sum += d;
               }
           }
 2         return sum/numbers.length;
       }
   }
```

(Cobertura's Code coverage report.)

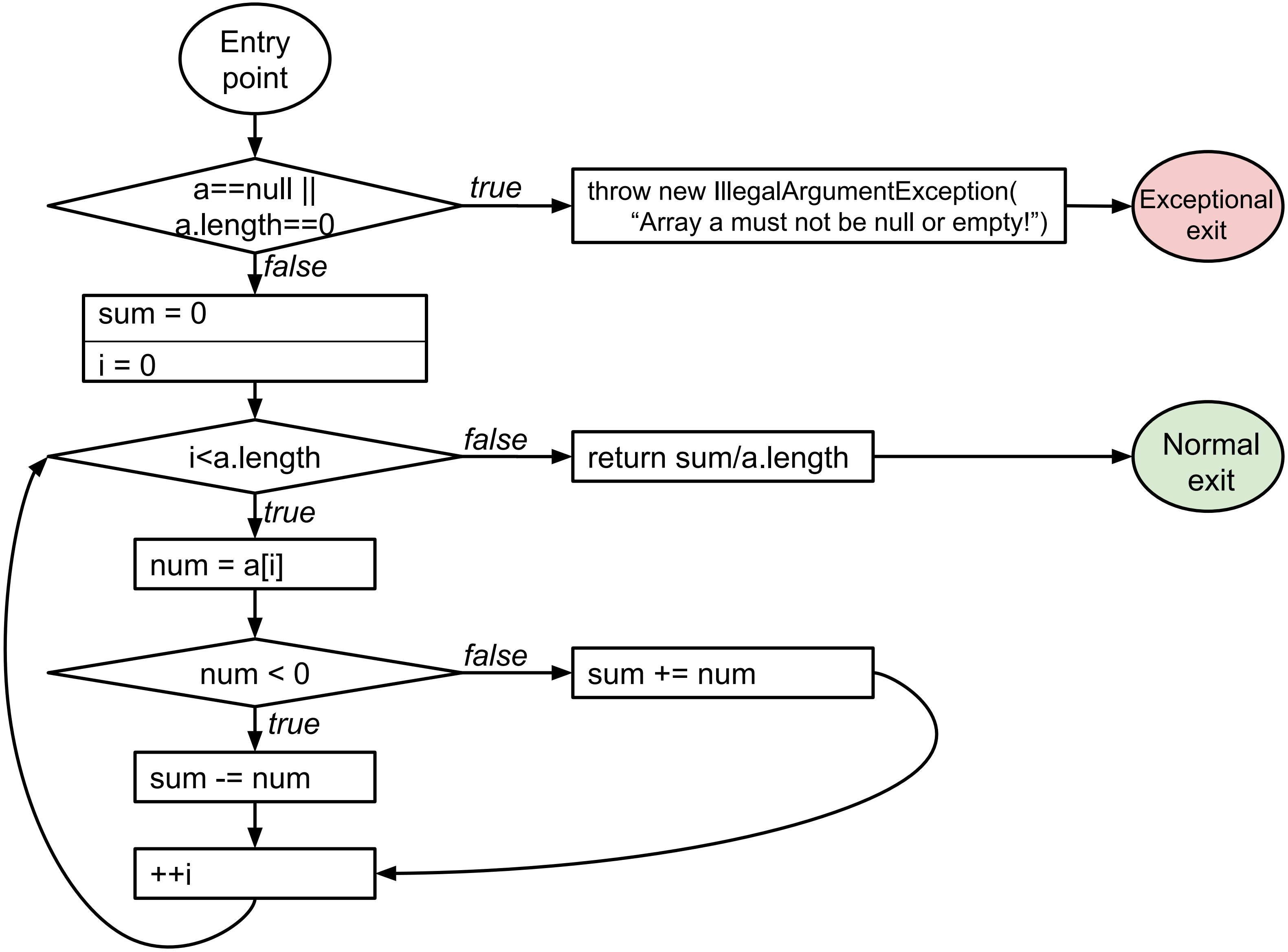
Structural code coverage: the basics

Average of the absolute values of an array of doubles

```
public double avgAbs(double ... numbers) {

  // We expect the array to be non-null and non-empty
  if (numbers == null || numbers.length == 0) {
    throw new IllegalArgumentException("Array numbers must not be null or empty!");
  }

  double sum = 0;
  for (int i=0; i<numbers.length; ++i) {
    double d = numbers[i];
    if (d < 0) {
      sum -= d;
    } else {
      sum += d;
    }
  }

  return sum/numbers.length;
}
```

What's the control flow graph (CFG) for this method?

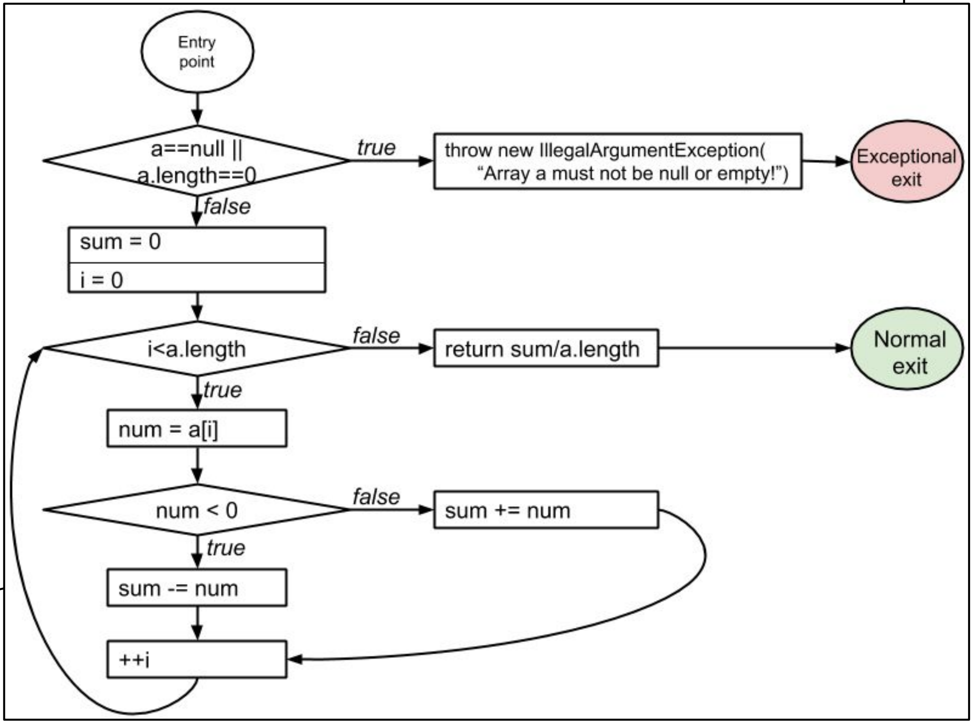
Structural code coverage: the basics



Structural code coverage: the basics

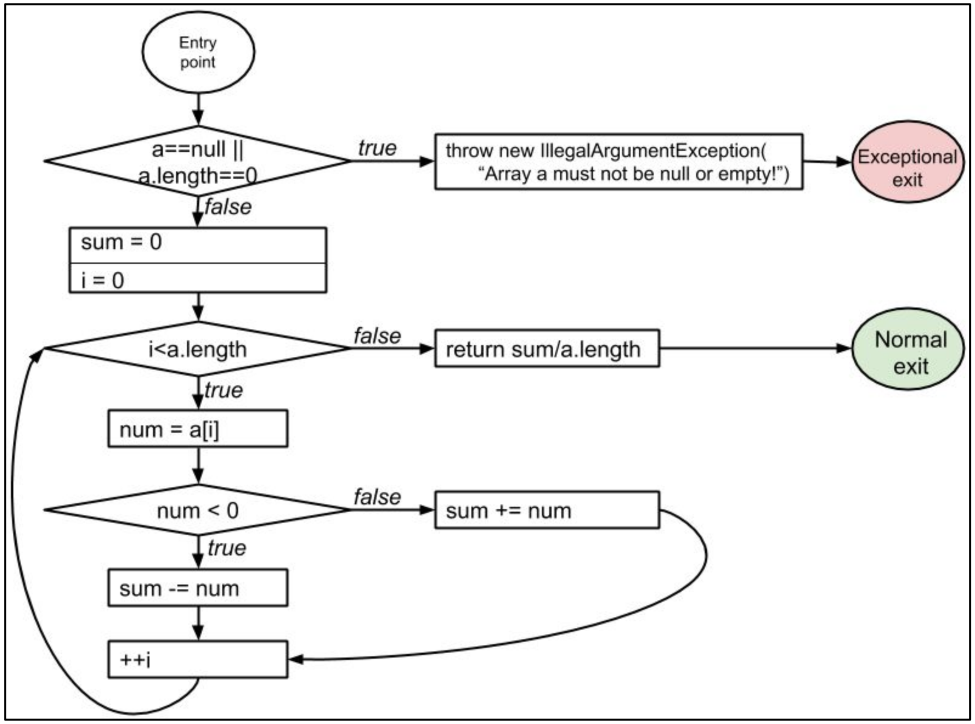
Average of the absolute values of an array of doubles

```
public double avgAbs(double ... numbers) {

  // We expect the array to be non-null and non-empty
  if (numbers == null || numbers.length == 0) {
    throw new IllegalArgumentException("Array numbers must not be null or empty!");
  }

  double sum = 0;
  for (int i=0; i<numbers.length; ++i) {
    double d = numbers[i];
    if (d < 0) {
      sum -= d;
    } else {
      sum += d;
    }
  }

  return sum/numbers.length;
}
```

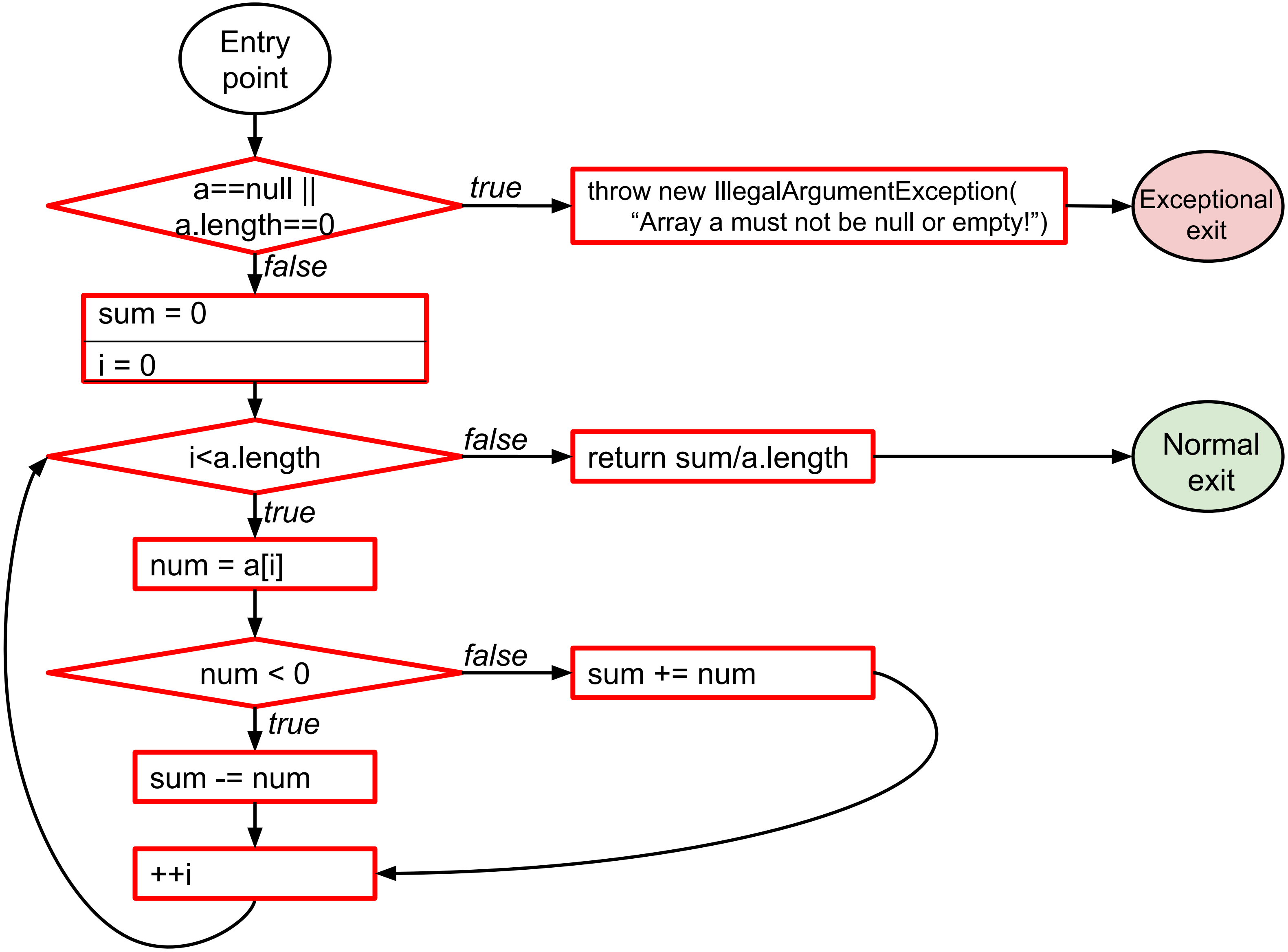


Statement coverage

- **Every statement** in the program must be **executed at least once.**

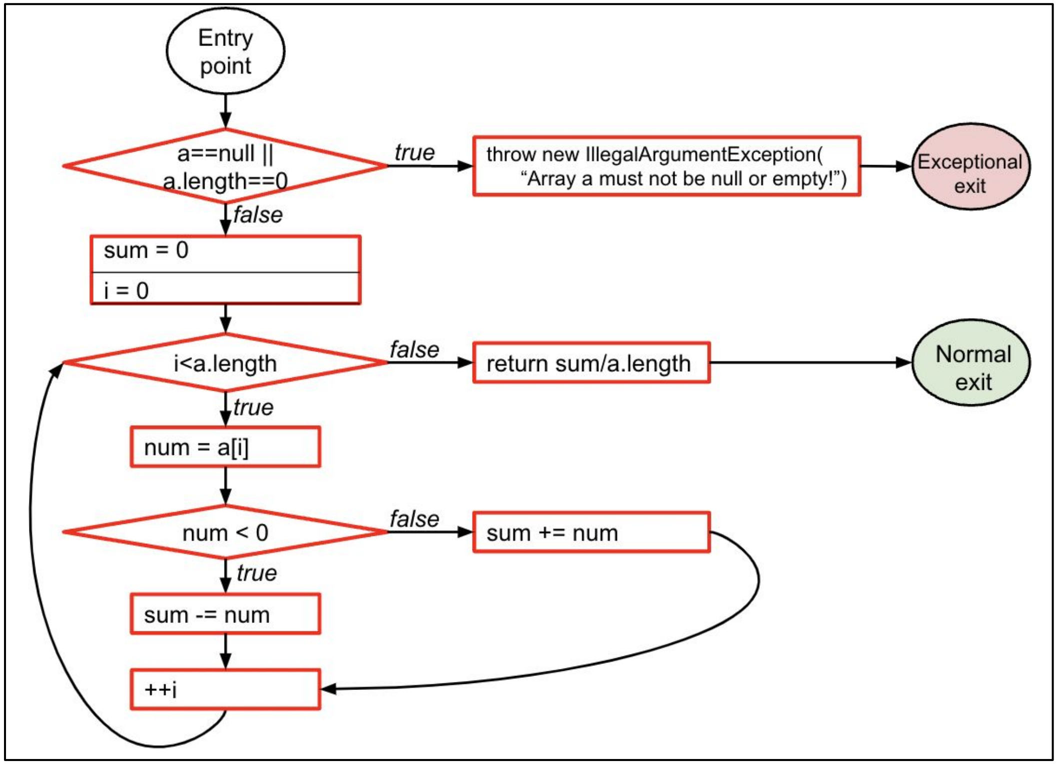


Statement coverage



Statement coverage

- **Every statement** in the program must be **executed at least once.**
- Given the control-flow graph (CFG), this is equivalent to node coverage.



Condition coverage vs. decision coverage

Terminology

- **Condition**: a boolean expression that cannot be decomposed into simpler boolean expressions (atomic).
- **Decision**: a boolean expression that is composed of conditions, using 0 or more logical connectors (a decision with 0 logical connectors is a condition).
- **Example:** if (*a* | *b*) { … }
    - *a* and *b* are *conditions.*
    - The boolean expression *a* | *b* is a *decision.*

Condition coverage vs. decision coverage

Terminology

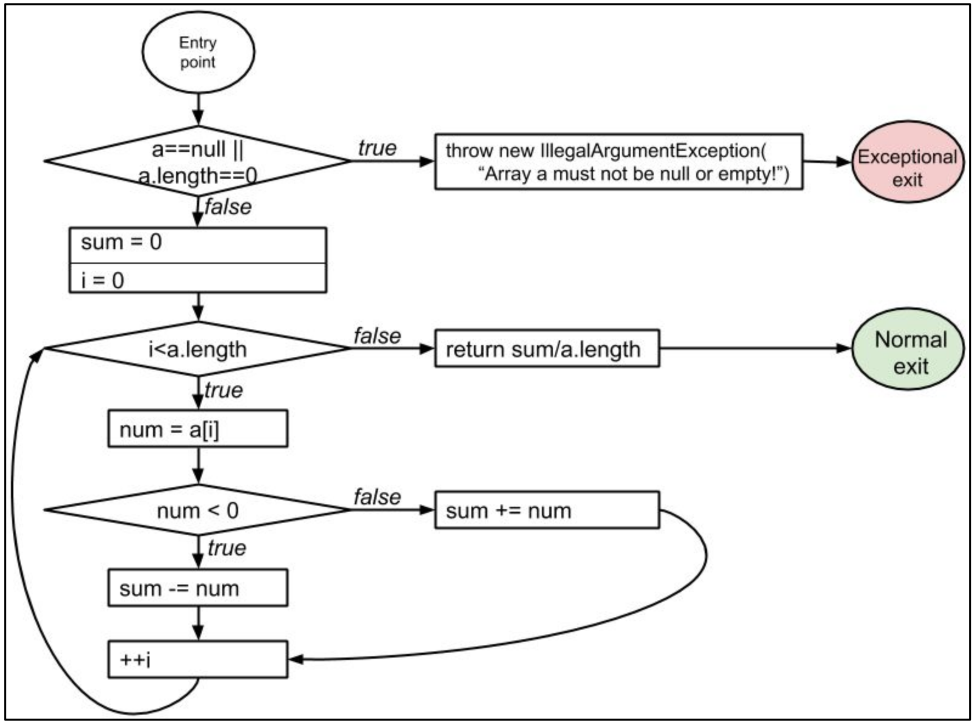
- **Condition**: a boolean expression that cannot be decomposed into simpler boolean expressions (atomic).

- **Decision**: a boolean expression that is composed of conditions, using 0 or more logical connectors (a decision with 0 logical connectors is a condition).

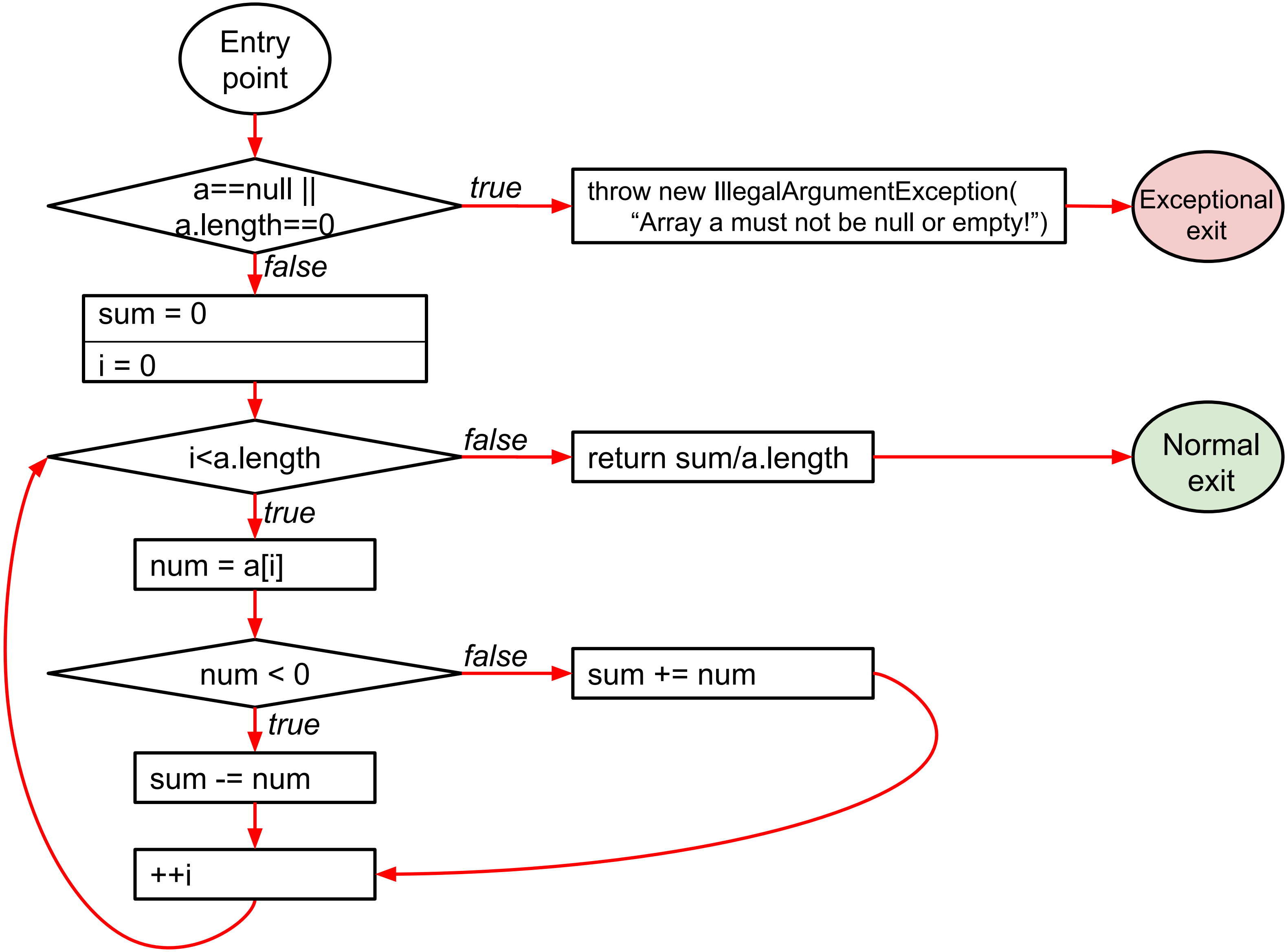
- **Example:** if (*a* | *b*) { … }
    - *a* and *b* are *conditions.*
    - The boolean expression *a* | *b* is a *decision.*

Decision coverage

- **Every decision** in the program must take on **all possible outcomes** (true/false) **at least once.**

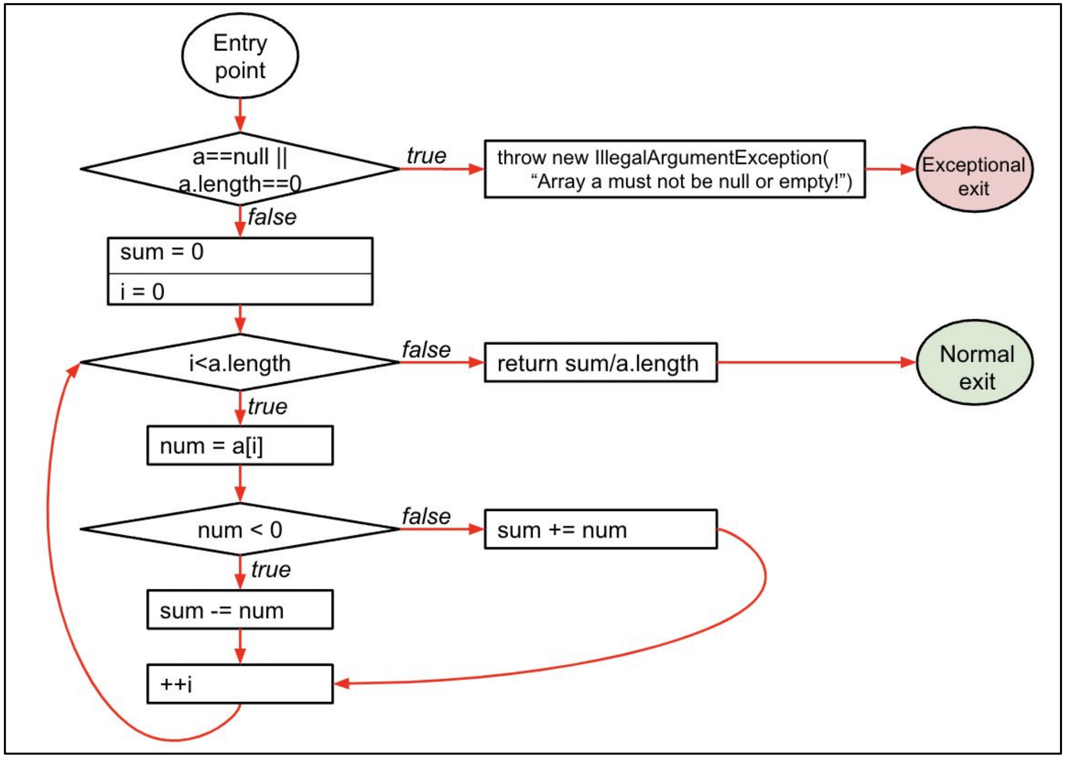


Decision coverage



Decision coverage

- **Every decision** in the program must take on **all possible outcomes** (true/false) **at least once.**
- Given the CFG, this is equivalent to edge coverage.



Condition coverage vs. decision coverage

Terminology

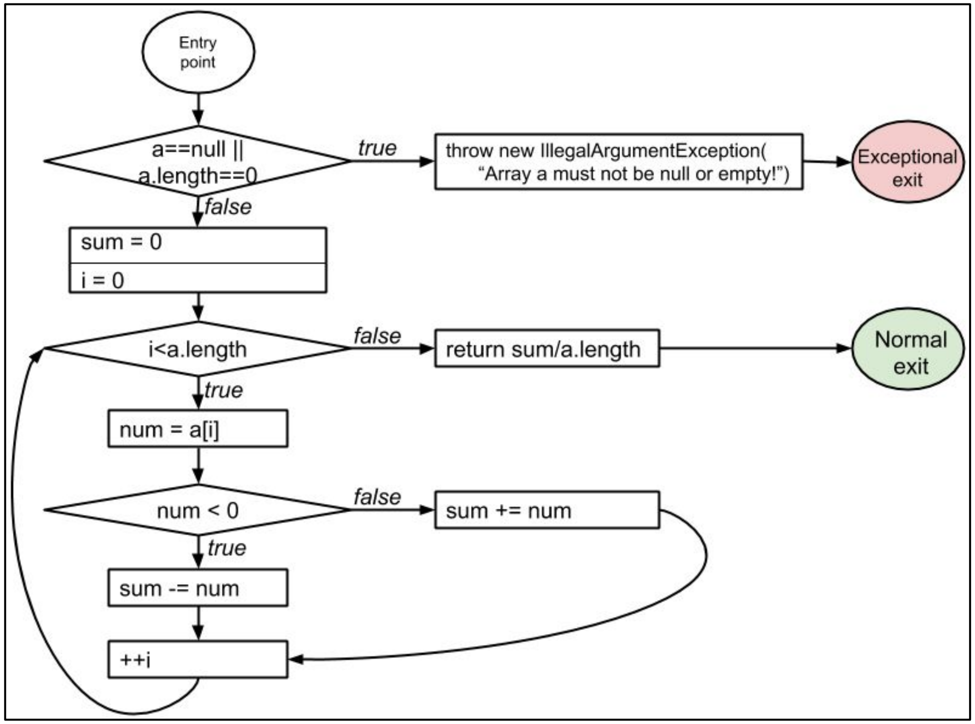
- **Condition**: a boolean expression that cannot be decomposed into simpler boolean expressions (atomic).

- **Decision**: a boolean expression that is composed of conditions, using 0 or more logical connectors (a decision with 0 logical connectors is a condition).

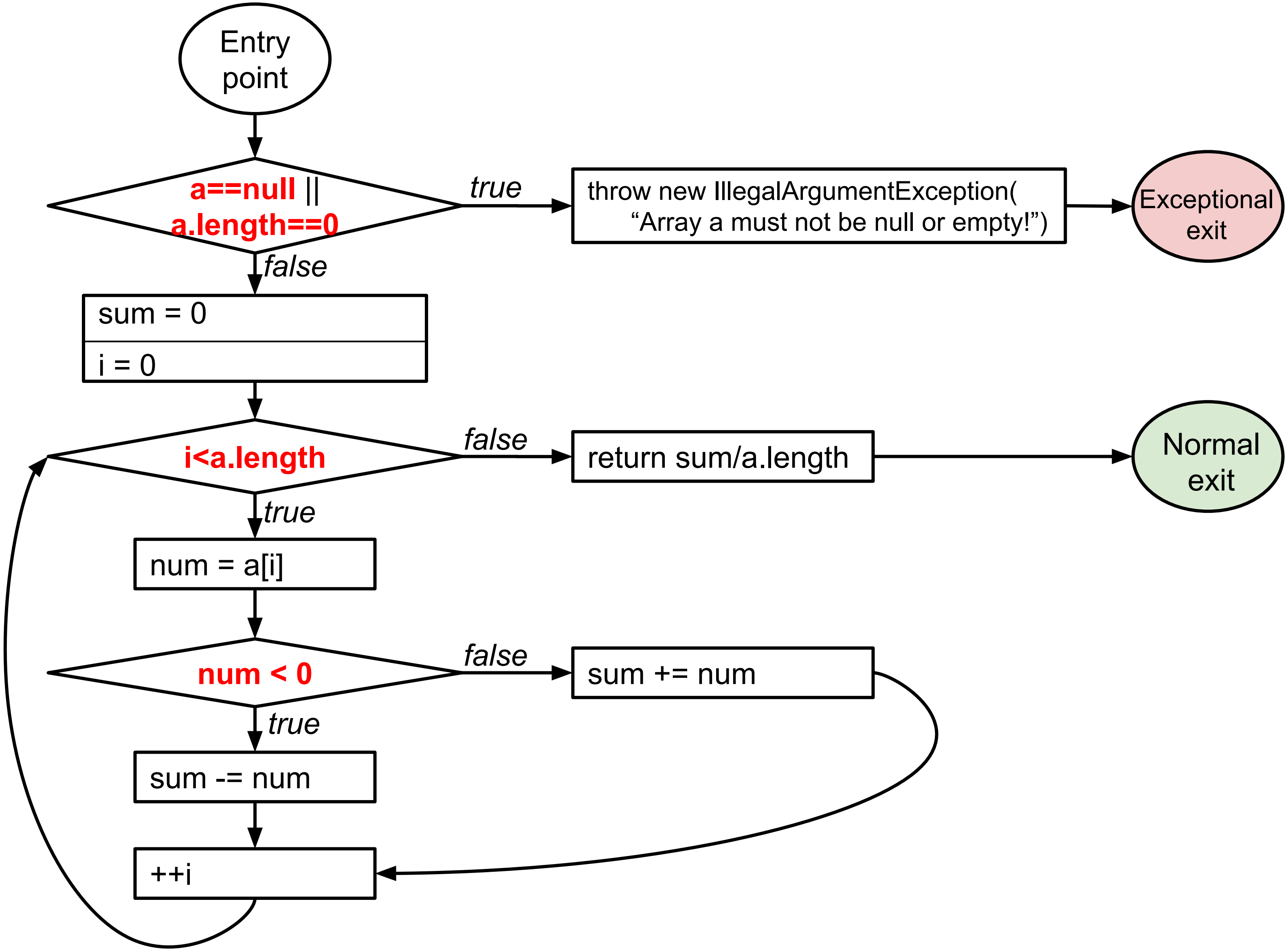
- **Example:** if (*a* | *b*) { … }
  - *a* and *b* are *conditions.*
  - The boolean expression *a* | *b* is a *decision.*

Condition coverage

- **Every condition** in the program must take on **all possible outcomes** (true/false) **at least once.**

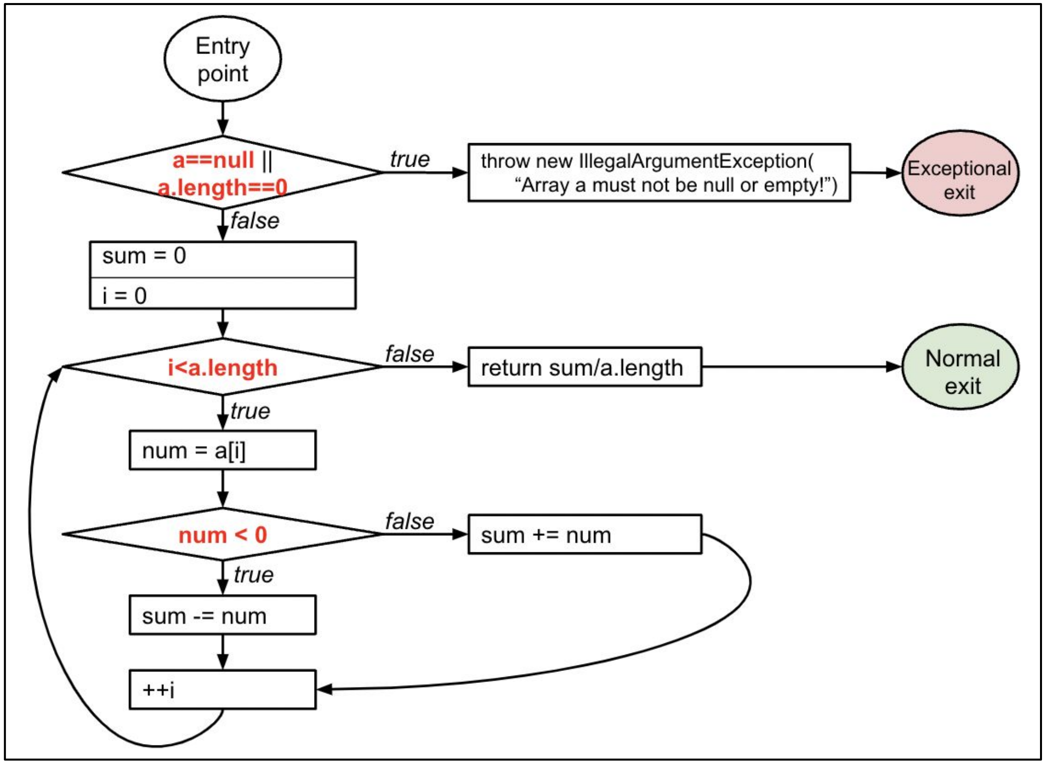


Condition coverage



Condition coverage

- **Every condition** in the program must take on **all possible outcomes** (true/false) **at least once.**



Structural code coverage: subsumption

Given two coverage criteria A and B,
A subsumes B iff **satisfying A implies satisfying B**

- Subsumption relationships:
  1. Does statement coverage subsume decision coverage?
  2. Does decision coverage subsume statement coverage?
  3. Does decision coverage subsume condition coverage?
  4. Does condition coverage subsume decision coverage?

https://pollev.com/renejust859

Structural code coverage: subsumption

Given two coverage criteria A and B,
A subsumes B iff **satisfying A implies satisfying B**

- Subsumption relationships:
  1. **Statement** coverage **does not subsume decision** coverage
  2. **Decision** coverage **subsumes statement** coverage
  3. **Decision** coverage **does not subsume condition** coverage
  4. **Condition** coverage **does not subsume decision** coverage

Decision coverage vs. condition coverage

4 possible tests for the decision *a* | *b*:

1. *a* = 0, *b* = 0
2. *a* = 0, *b* = 1
3. *a* = 1, *b* = 0
4. *a* = 1, *b* = 1

a	*b*	*a* | *b*
0	0	0
0	**1**	**1**
1	**0**	**1**
1	1	1

Satisfies **condition coverage** but **not decision coverage**

a	*b*	*a* | *b*
0	**0**	**0**
0	**1**	**1**
1	0	1
1	1	1

Does **not** satisfy **condition coverage** but **decision coverage**

Neither coverage criterion subsumes the other!

MCDC: Modified condition and decision coverage

- **Every decision** in the program must take on **all possible outcomes** (true/false) **at least once**
- **Every condition** in the program must take on **all possible outcomes** (true/false) **at least once**
- **Each condition** in a decision has been shown to **independently affect** that decision's **outcome**.
  (A condition is shown to independently affect a decision's outcome by: varying just that condition while holding fixed all other possible conditions.)

Required for safety critical systems (DO-178B/C)

MCDC: an example

```
if (a | b)
```

a	b	Outcome
0	0	0
0	1	1
1	0	1
1	1	1

MCDC

- **Decision** coverage
- **Condition** coverage
- **Each condition** shown to **independently affect outcome**

Which tests (combinations of a and b) satisfy MCDC?

MCDC: an example

```
if (a | b)
```

a	b	Outcome
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	1	1

MCDC

- **Decision** coverage
- **Condition** coverage
- **Each condition** shown to **independently affect outcome**

MCDC is still cheaper than testing all possible combinations.

MCDC: another example

```
if (a || b)
```

a	b	Outcome
0	0	0
0	1	1
1	0	1
1	1	1

MCDC

- **Decision** coverage
- **Condition** coverage
- **Each condition** shown to **independently affect outcome**

Why is this example different?

MCDC: another example

```
if (a || b)
```

a	b	Outcome
0	0	0
0	1	1
1	--	1
1	--	1

MCDC

- **Decision** coverage
- **Condition** coverage
- **Each condition** shown to **independently affect outcome**

Short-circuiting operators may not evaluate all conditions.

MCDC: yet another example

```
if (!a) ... if (a || b)
```

a	b	Outcome
0	0	0
0	1	1
1	0	1
1	1	1

MCDC

- **Decision** coverage
- **Condition** coverage
- **Each condition** shown to **independently affect outcome**

What about this example?

MCDC: yet another example

```
if (!a) ... if (a || b)
```

a	b	Outcome
0	0	0
0	1	1
X	X	X
X	X	X

MCDC

- **Decision** coverage
- **Condition** coverage
- **Each condition** shown to **independently affect outcome**

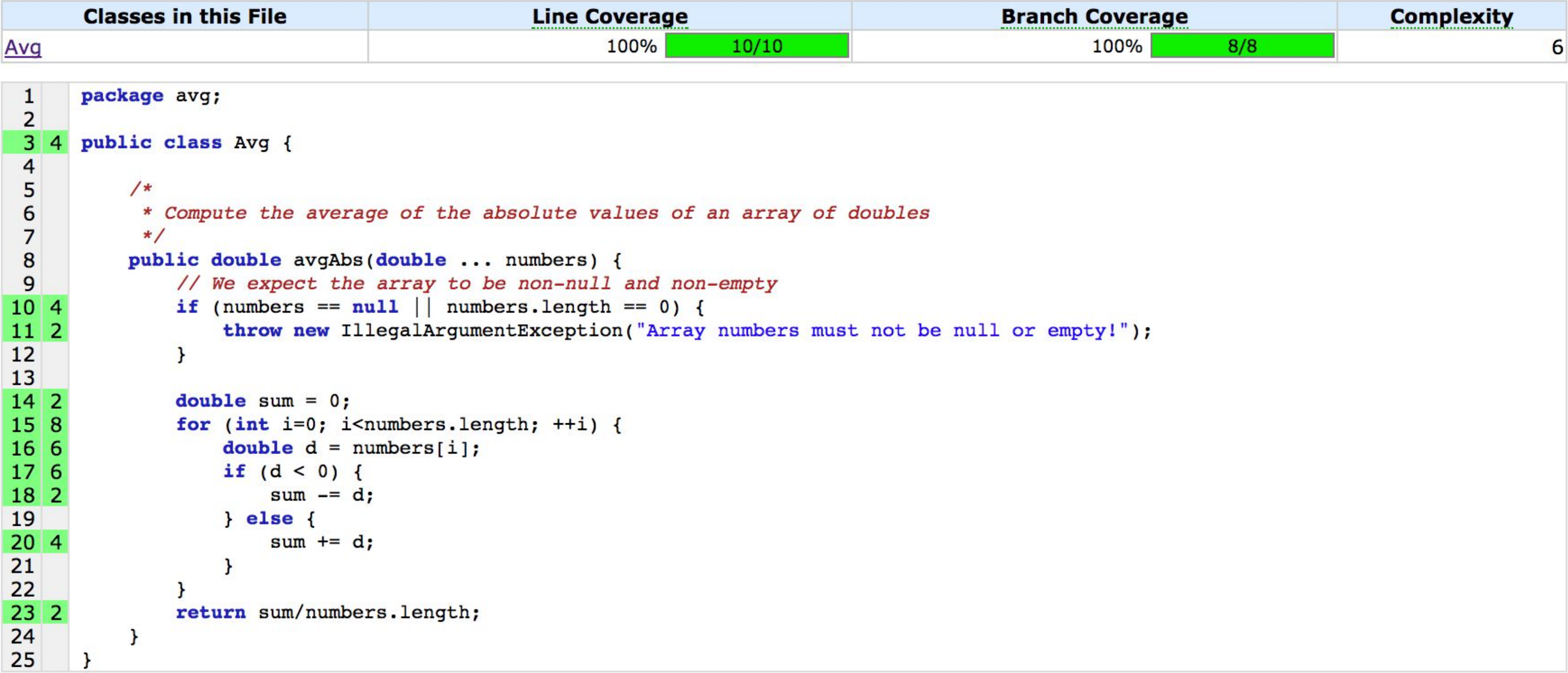
Not all combinations of conditions may be possible.

MCDC: complex expressions

Provide an MCDC-adequate test suite for:

1. `a | b | c`
2. `a & b & c`

Structural code coverage: summary



Classes in this File	Line Coverage	Branch Coverage	Complexity
Avg	100% 10/10	100% 8/8	6

```
   package avg;

 4 public class Avg {

       /*
        * Compute the average of the absolute values of an array of doubles
        */
       public double avgAbs(double ... numbers) {
           // We expect the array to be non-null and non-empty
 4         if (numbers == null || numbers.length == 0) {
 2             throw new IllegalArgumentException("Array numbers must not be null or empty!");
           }

 2         double sum = 0;
 8         for (int i=0; i<numbers.length; ++i) {
 6             double d = numbers[i];
 6             if (d < 0) {
 2                 sum -= d;
               } else {
 4                 sum += d;
               }
           }
 2         return sum/numbers.length;
       }
   }
```

- Code coverage is easy to compute.
- Code coverage has an intuitive interpretation.
- Code coverage in industry: Code coverage at Google
- Code coverage itself is not sufficient!