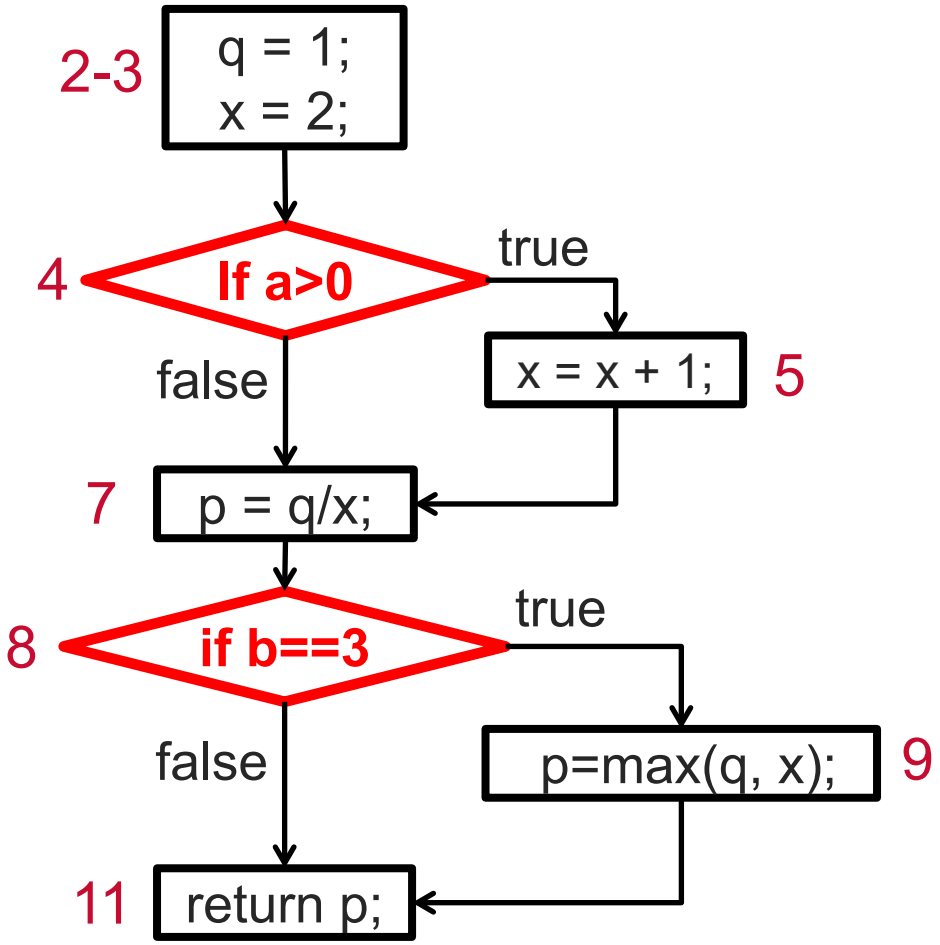
NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Tutorial #9 – Control Flow Testing

Dr. Wang Wenhan

Control Flow Graph - Example

```
1    int computeP(int a, int b) {
2        int q, x, p;
3        q = 1;
4        x = 2;
5        if(a > 0) {
6            x = x + 1;
7        }
8        p = q/x;
9        if(b == 3) {
10           p = max(q, x);
11       }
12       return p;
     }
```



Basis Path Testing

Basis Path Testing (or structured testing) is a white box testing method used for designing test cases intended to examine all possible paths of execution in a program at least once.

- Analyzes Control flow graph (CFG)
- Determines Cyclomatic Complexity (CC) value
- Obtains linearly independent paths
- Generates test cases for each path

Cyclomatic Complexity of a CFG

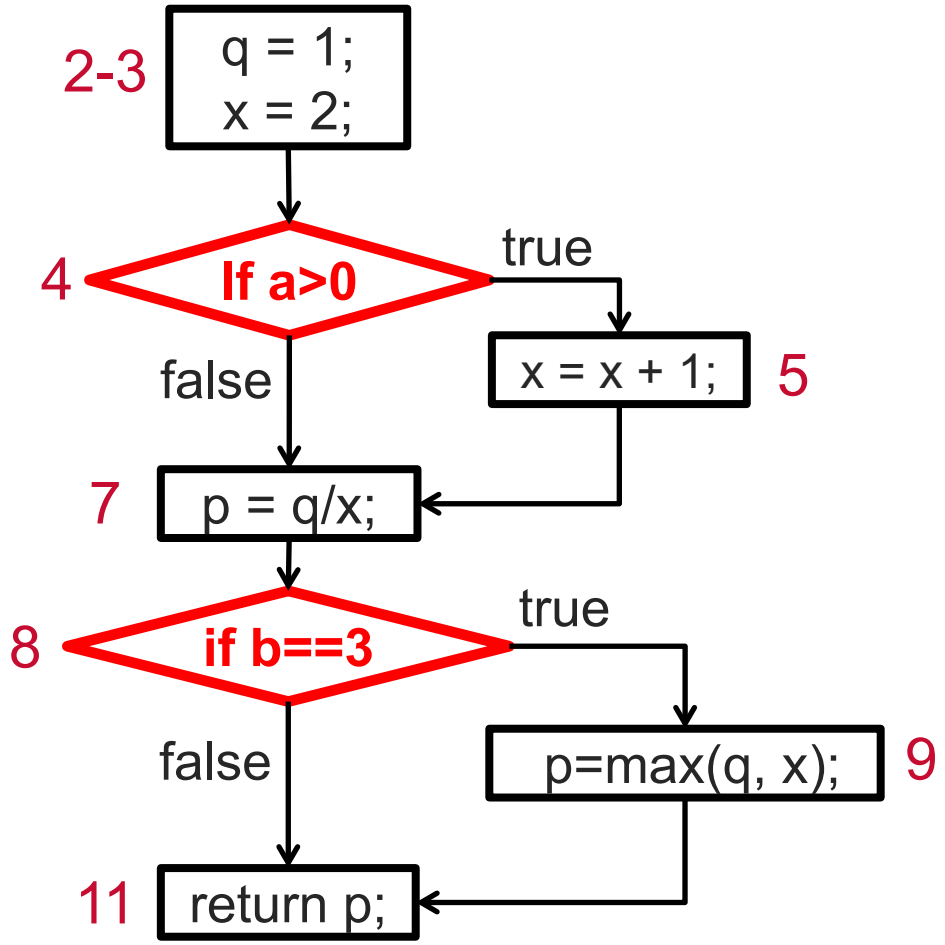
Use the following two equations to compute CC

- CC = |edges| - |nodes| + 2
- CC = |decisionpoint| + 1, if all decision points are binary, i.e., one true branch + one false branch

The two equations compute the same value if all decision points are binary

Cyclomatic Complexity – Example

```
1   int computeP(int a, int b) {
2       int q, x, p;
3       q = 1;
4       x = 2;
5       if(a > 0) {
6           x = x + 1;
7       }
8       p = q/x;
9       if(b == 3) {
10          p = max(q, x);
11      }
12      return p;
    }
```



|edges| = 8
|nodes| = 7
CC = 8 – 7 + 2 = 3

Or,

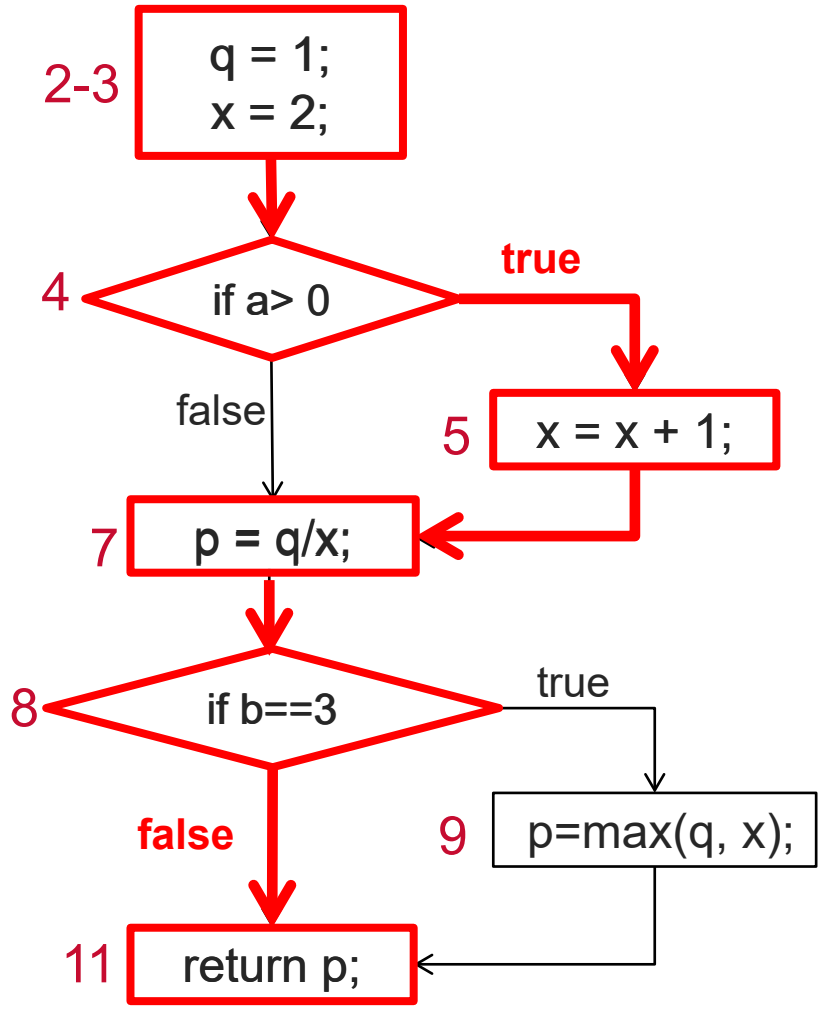
|decisionpoint| = 2
CC = 2 + 1 = 3

Select a Set of Basis Paths

Pick a “baseline” path

- Reasonably “typical” path of execution
- Most important path from the tester’s view

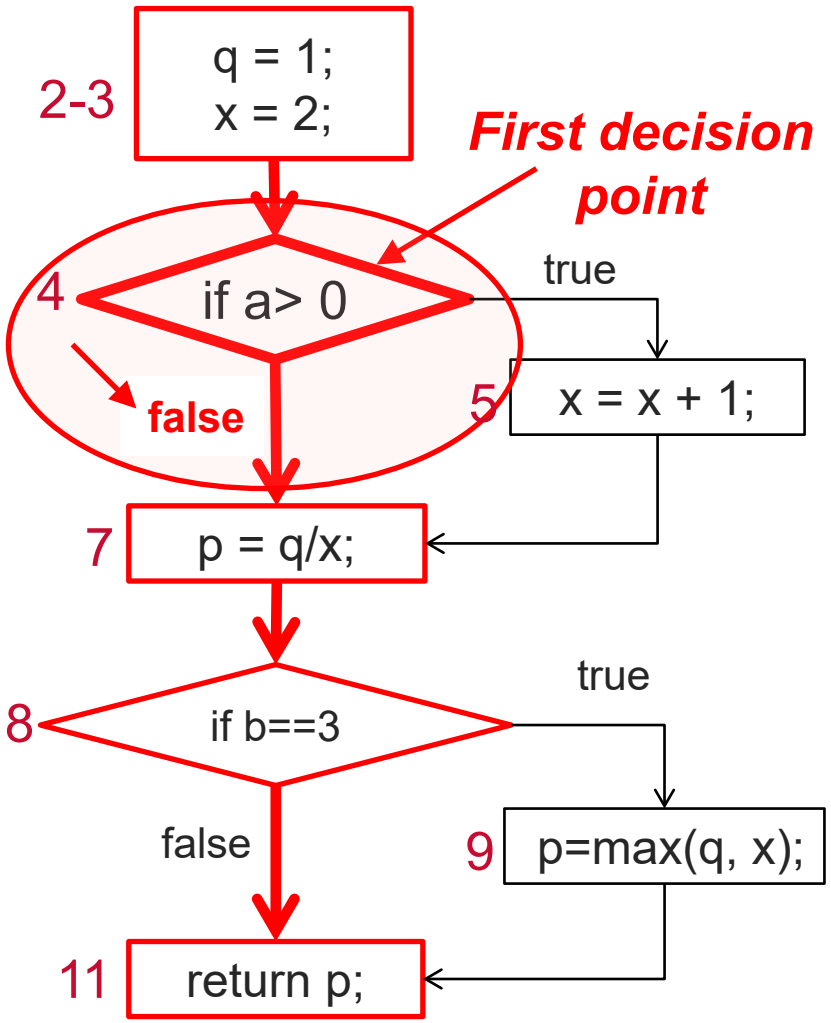
Basis path#1 (baseline):
2-3, 4, 5, 7, 8, 11



Select a Set of Basis Paths

- Change the outcome of the first decision point
- Keep the maximum number of other decision points the same

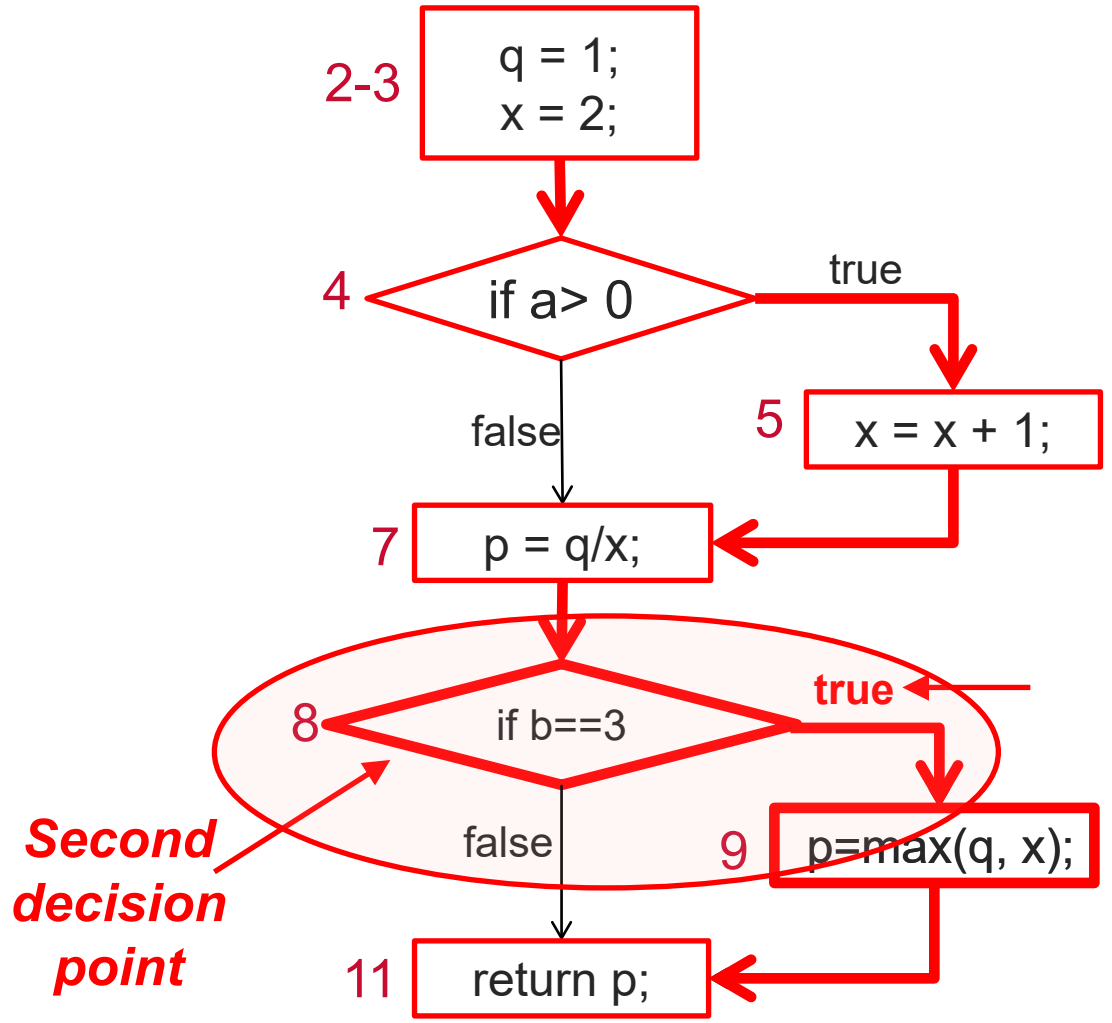
Basis path#2:
2-3, 4, 7, 8, 11



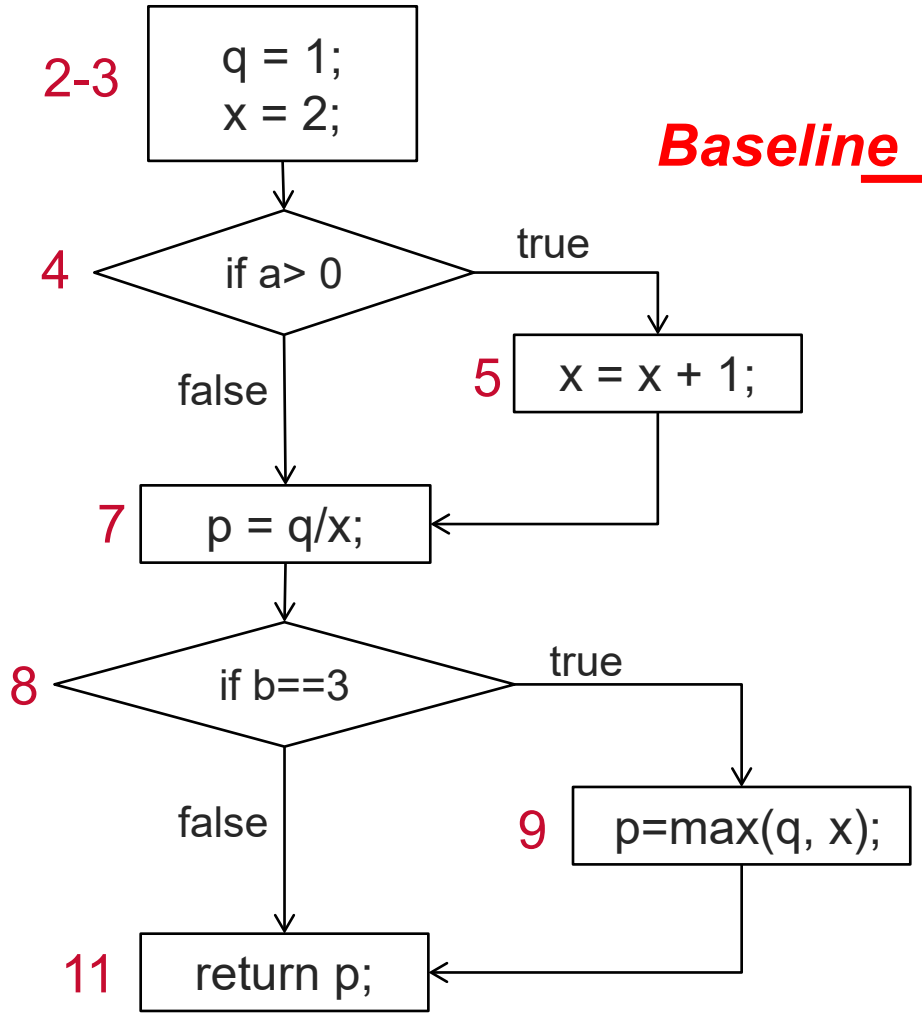
Select a Set of Basis Paths

- Change the outcome of the second decision point
- Keep the maximum number of other decision points the same

Basis path#3 :
2-3, 4, 5, 7, 8, 9, 11



Create a Test Case for Each Basis Path

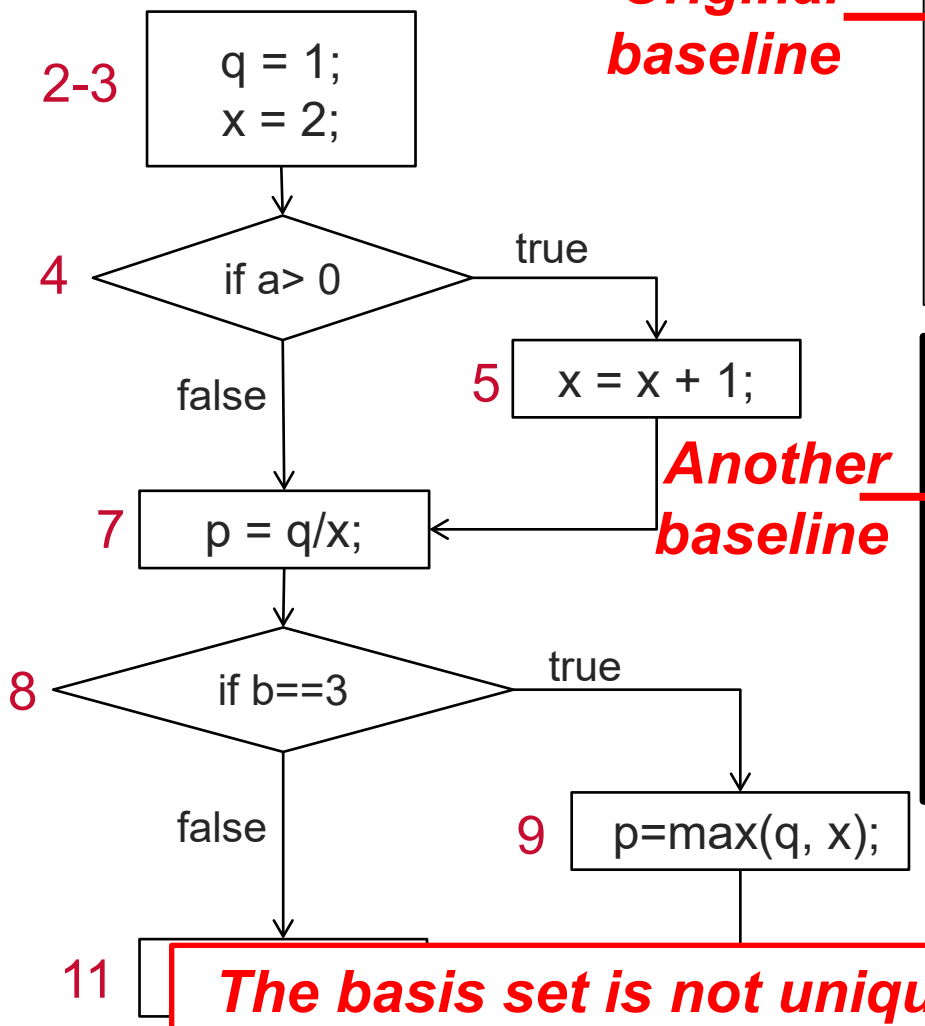


Baseline

- One set of basis paths
  - I. 2-3, 4, 5, 7, 8, 11
  - II. 2-3, 4, 7, 8, 11
  - III. 2-3, 4, 5, 7, 8, 9, 11

- Three test cases
  - I. a = 4; b = 2
  - II. a = 0; b = 5
  - III. a = 7; b = 3

Basis Path



Original baseline

- One set of basis paths
  - I. 2-3, 4, 5, 7, 8, 11
  - II. 2-3, 4, 7, 8, 11
  - III. 2-3, 4, 5, 7, 8, 9, 11

Another baseline

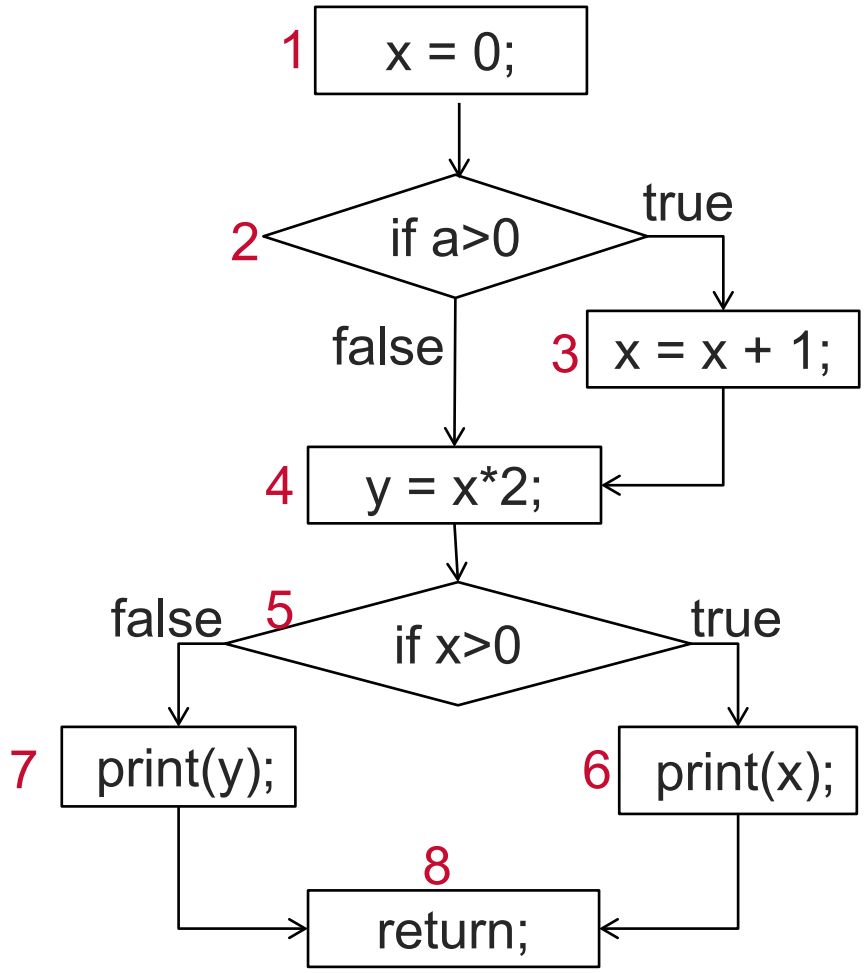
- Another set of basis paths
  - I. 2-3, 4, 7, 8, 11
  - II. 2-3, 4, 5, 7, 8, 11
  - III. 2-3, 4, 7, 8, 9, 11

*The **third** paths in the two sets are different*

The basis set is not unique; but each basis set has the same number of basis paths (i.e. 3 in this case)

Not All Basis Paths are Feasible

Consider another program logic

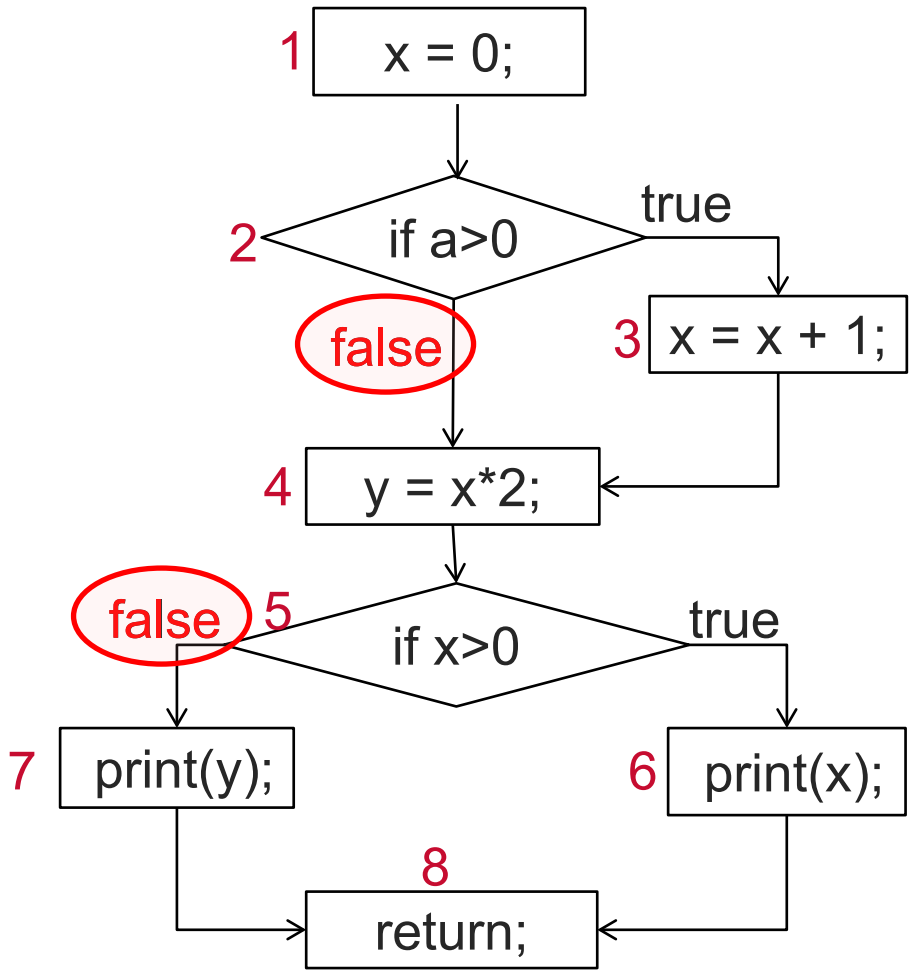


- One set of basis paths
  - I. 1, 2, 3, 4, 5, 6, 8 ← ***Baseline***
  - ~~II. 1, 2, 4, 5, 6, 8~~ (infeasible)
  - ~~III. 1, 2, 3, 4, 5, 7, 8~~ (infeasible)
  - *- Two infeasible paths*

- One test case
  - I. a = 4
  - II. Infeasible basis path
  - III. Infeasible basis path

Fail to test 2, 4, 5, 6, 8, and 5, 7, 8, branches

Minimize Infeasible Basis Paths



- Another set of basis paths
  - I. 1, 2, 3, 4, 5, 6, 8
  - ~~II. 1, 2, 4, 5, 6, 8~~ (infeasible)
  - III. 1, 2, 4, 5, 7, 8 (change both decision points at the same time)
  - *- One infeasible path*

- Two test case
  - I. a = 4
  - II. Infeasible basis path
  - III. a = 0

Question 1

a) Draw the corresponding control flow graph for the sum(int n, int upperbound) method

b) Calculate the Cyclomatic Complexity of the sum(int) method

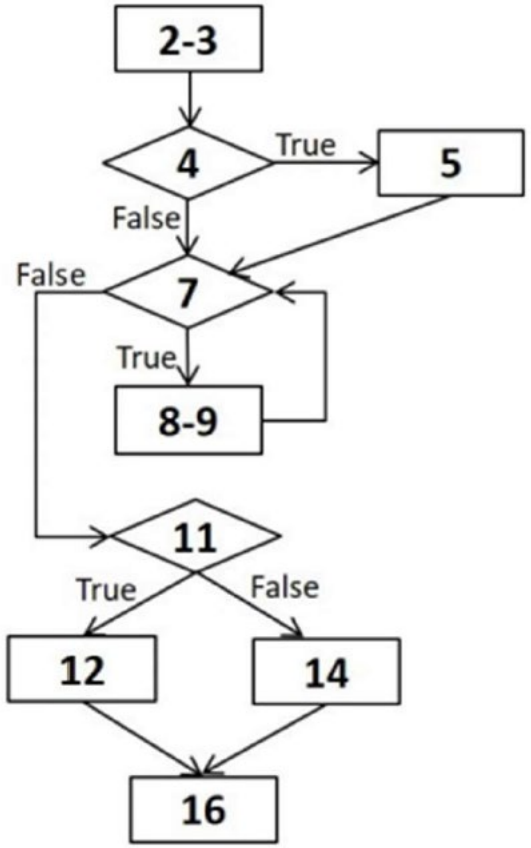
c) Design test cases to achieve 100% statement coverage and 100% branch coverage

d) List the basis set of linearly independent paths, along with a test case (input parameters) and expected outcome for each of the path in the basis path set

```
public int sum(int n, int upperbound){
   1  int result, i;
   2  result = 0;
   3  i = 0;
   4  if(n < 0) {
   5    n = -n;
   6  }
   7  while(i<n && result <= upperbound){
   8    i = i + 1;
   9    result = result + i;
   10 }
   11 if(result <= upperbound){
   12   System.out.println("The sum is " + result);
   13 } else {
   14   System.out.println("The sum is too large!");
   15 }
   16 return result;
}
```

Question 1 (a) - Answer

```
public int sum(int n, int upperbound){
   1  int result, i;
   2  result = 0;
   3  i = 0;
   4  if(n < 0) {
   5    n = -n;
   6  }
   7  while(i<n && result <= upperbound){
   8    i = i + 1;
   9    result = result + i;
   10 }
   11 if(result <= upperbound){
   12   System.out.println("The sum is " + result);
   13 } else {
   14   System.out.println("The sum is too large!");
   15 }
   16 return result;
}
```



Question 1 (b) - Answer

Edges = 11

Nodes = 9

Cyclomatic complexity = 11 – 9 + 2 = 4

Question 1 (c) - Answer

100% Statement coverage

- Path: 2-3, 4, 5, 7, 8-9, 7, 11, 12, 16

Test case: n=-1, upperbound=1

- Path: 2-3, 4, 5, 7, 8-9, 7, 11, 14, 16

Test case: n=-1, upperbound=0

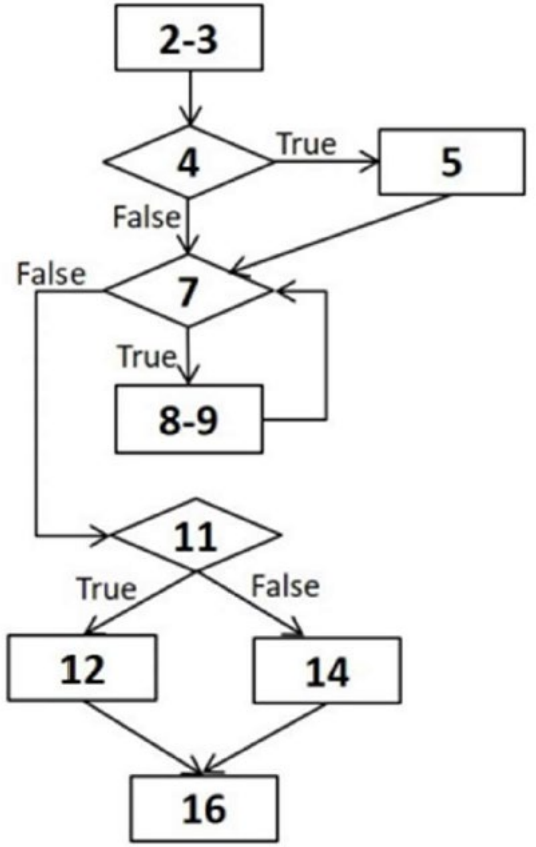
100% Branch coverage

- Path: 2-3, 4, 5, 7, 8-9, 7, 11, 12, 16

Test case: n=-1, upperbound=1

- Path: 2-3, 4, 7, 8-9, 7, 11, 14, 16

Test case: n=1, upperbound=0



Question 1 (d) - Answer

- Path: (Baseline)2-3 -> 4 -> 7 -> 11 -> 12 -> 16

Test case: n=0, ub=0

Expected outcome: “The sum is 0”, 0

- Path: 2-3 -> 4 -> 5 > 7 -> 11 -> 14 -> 16

Test case: n=-1, ub=-1

Expected outcome: “The sum is too large!”, 0

- Path: 2-3 -> 4 -> 7 -> 8 -> 9 -> 7 -> 11 -> 12 -> 16

Test case: n=1, ub=1

Expected outcome: “The sum is 1”, 1

- Path: 2-3 -> 4 -> 7 -> 11 -> 14 -> 16

Test case: n=0, ub=-1

Expected outcome: “The sum is too large!”, 0