SOFTWARE TESTING

苏临之

sulinzhi029@nwu.edu.com

Four Basic Techniques

- Static Black-Box Testing
- Dynamic Black-Box Testing
- Static White-Box Testing
- Dynamic White-Box Testing

Four Basic Techniques

- Static Black-Box Testing
- Dynamic Black-Box Testing
- Static White-Box Testing
- **Dynamic White-Box Testing**

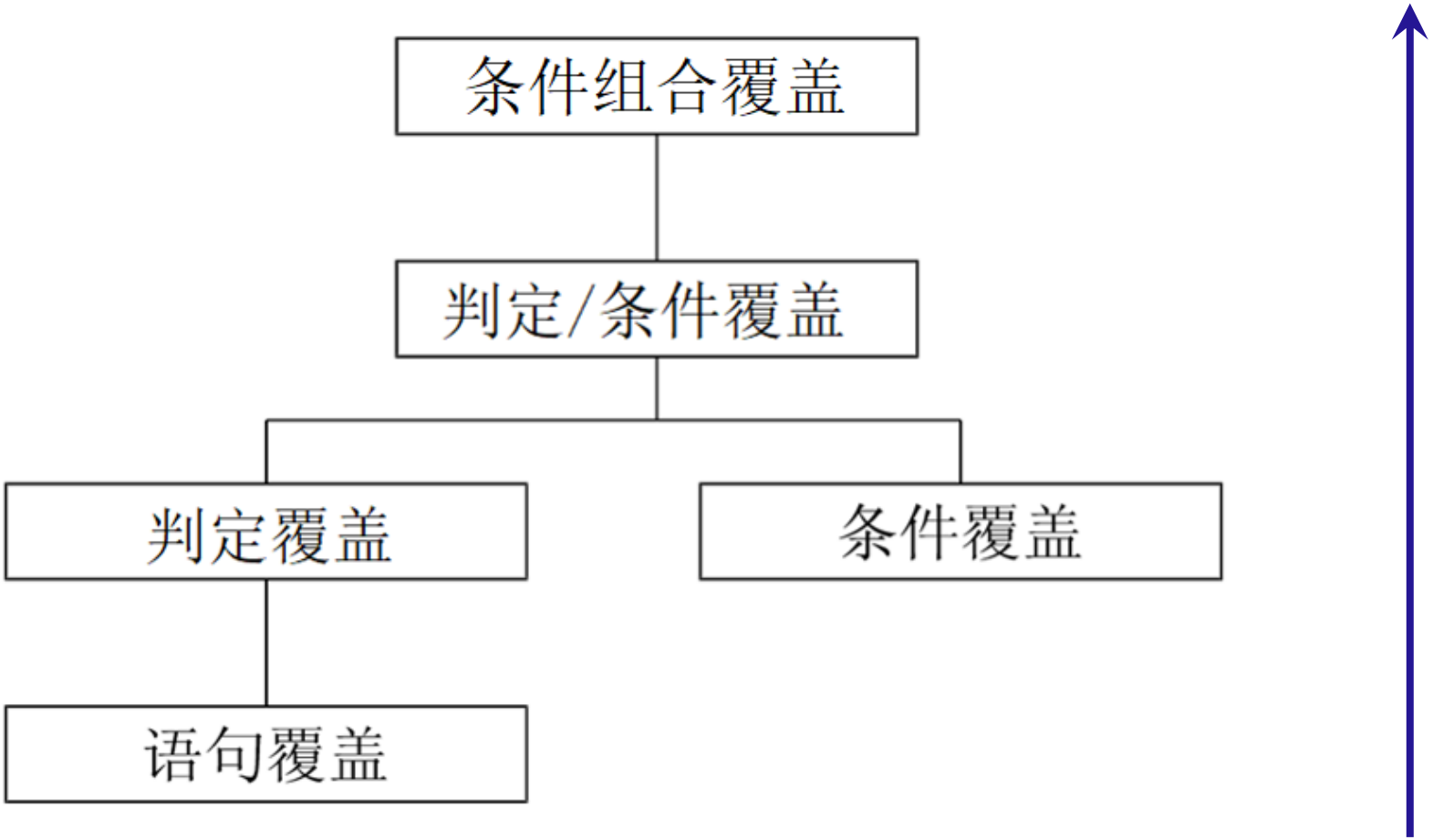
Dynamic White-Box Testing

- Based on the programming code, the dynamic white-box testing is also called code-based testing.
- The basic idea of DWBT is that every individual part of code should be operated once at least, which means to design test cases based on control flows and then to analysis the logic in the code. Finally, the test cases are applied and check the output results.

Techniques in DBWT

- Statement Coverage
- Decision Coverage
- Condition Coverage
- Decision / Condition Coverage
- Multiple Condition Coverage
- Path Coverage

Techniques in DBWT



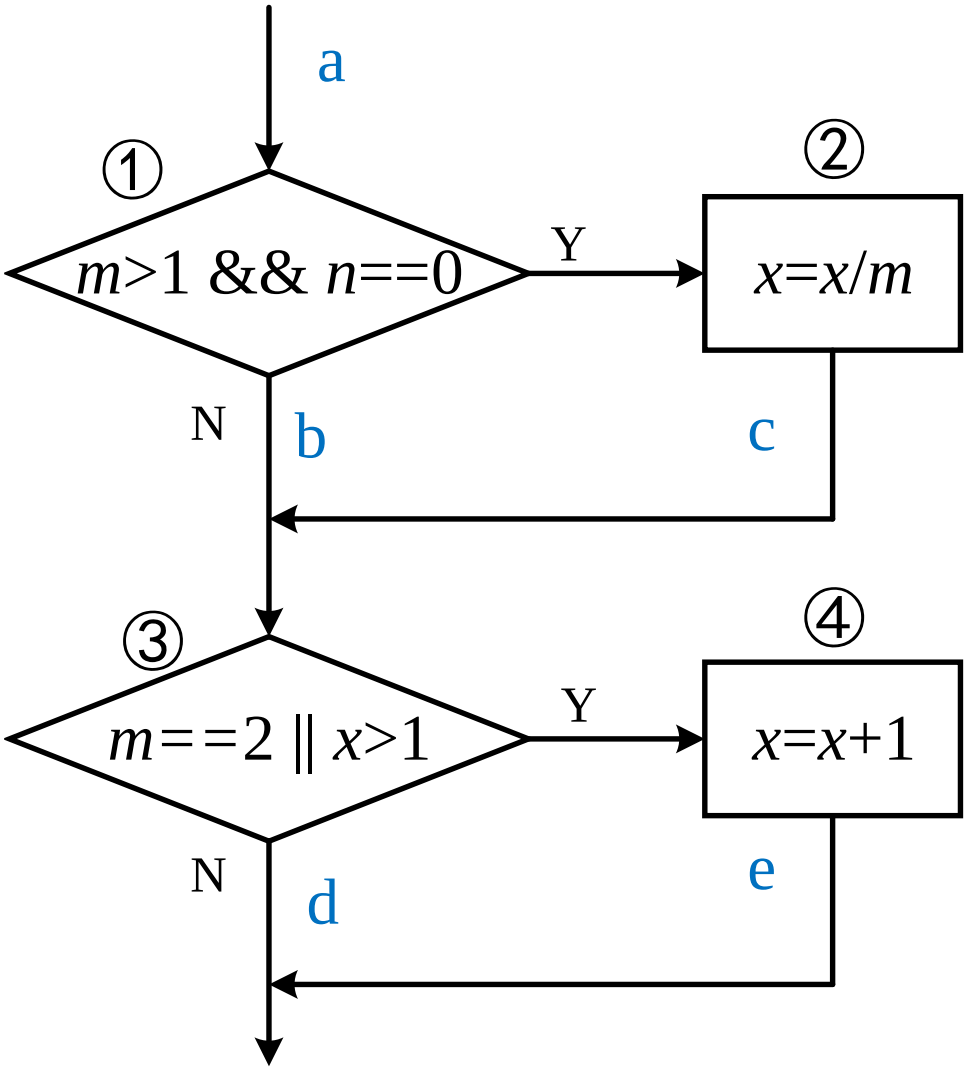
Example 1

- Please design test cases by using the dynamic white-box testing techniques for this piece of code below.

```
if( m>1&&n == 0){
    x = x/m;
}
if( m == 2 || x >1 ){
    x = x + 1;
}
```

Program Chart

```
if( m>1&&n == 0){
    x = x/m;
}
if( m == 2 || x >1 ){
    x = x + 1;
}
```



Test Cases

	测试用例 (*m*/*n*/*x*)	覆盖语句	预期结果
语句覆盖	2,0,4	①②③④	*x* = 3

	测试用例 (*m*/*n*/*x*)	覆盖分支	预期结果
判定覆盖	2,0,4	ce	*x* = 3
	3,1,1	bd	*x* = 1

Test Cases

条件覆盖	测试用例 (*m*/*n*/*x*)	覆盖条件	预期结果
	1,0,3	F1 T2 F3 T4	*x* = 4
	2,1,1	T1 F2 T3 F4	*x* = 2

判定/条件覆盖	测试用例 (*m*/*n*/*x*)	覆盖条件	覆盖分支	预期结果
	2,0,4	T1 T2 T3 T4	ce	*x* = 3
	1,1,1	F1 F2 F3 F4	bd	*x* = 1

Test Cases

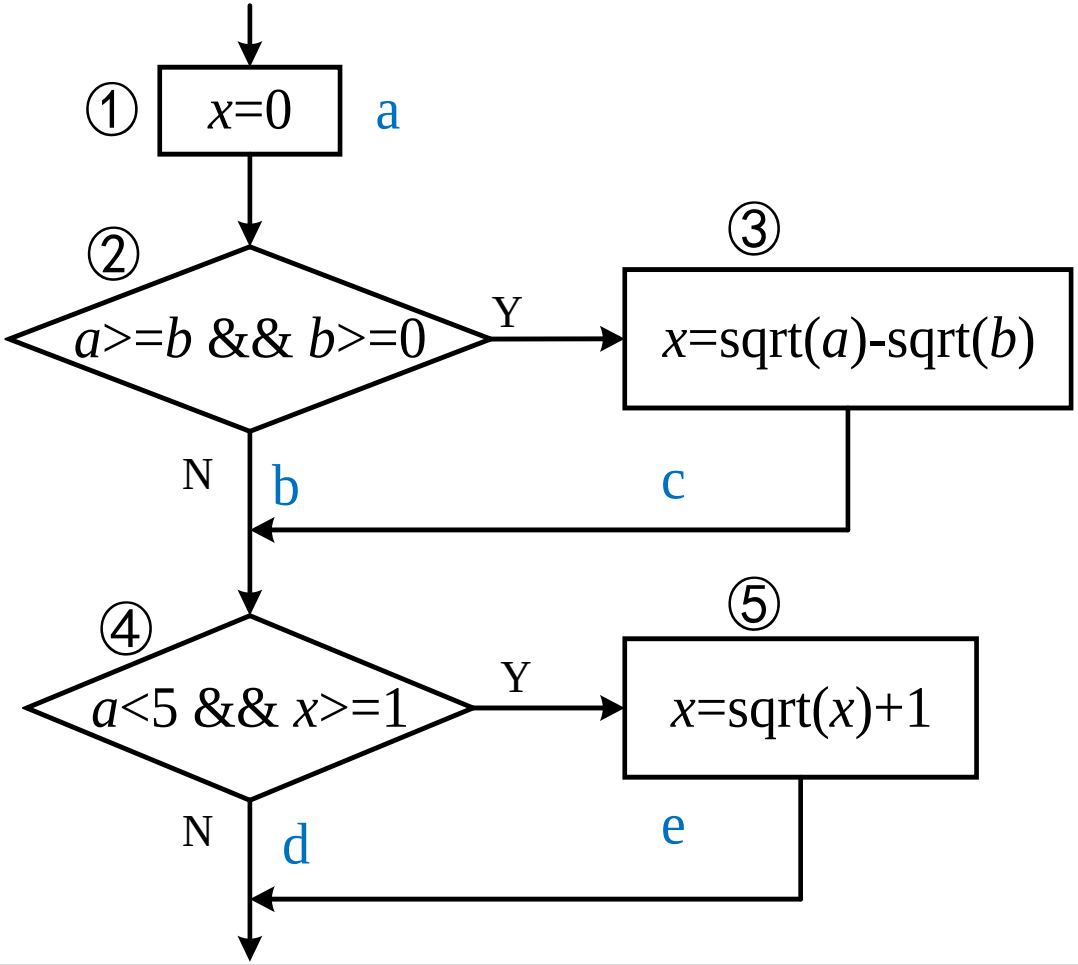
	测试用例 (*m*/*n*/*x*)	覆盖条件组合	预测结果
条件组合覆盖	2,0,4	T1T2, T3T4	$x = 3$
	2,1,1	T1F2, T3F4	$x = 2$
	1,0,2	F1T2, F3T4	$x = 3$
	1,1,1	F1F2, F3F4	$x = 1$

Example 2

- 右侧程序输入变量是a和b，输出变量是x。请画出框图和程序控制流图，然后分别给出语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖和条件组合覆盖的测试用例及预期结果。（提示：使用合适的整数以方便测试）

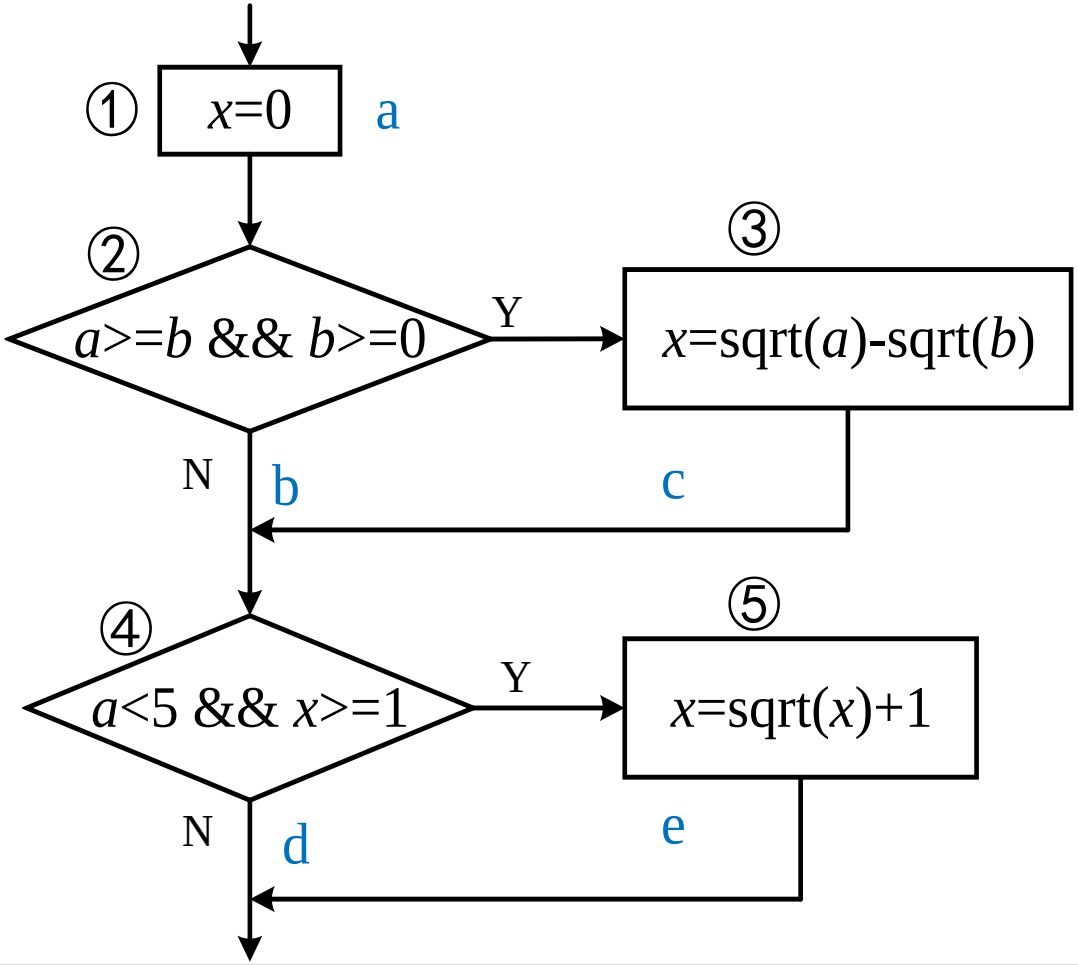
```
void main(){
    int a, b; double x=0;
    scanf(“%d %d”,&a, &b);
    if a>=b && b>=0
        x=sqrt(a)-sqrt(b);
    if a<5 && x>=2
        x=sqrt(x)-1;
    printf(“%d”, x);
}
```

Program Chart



Test Cases

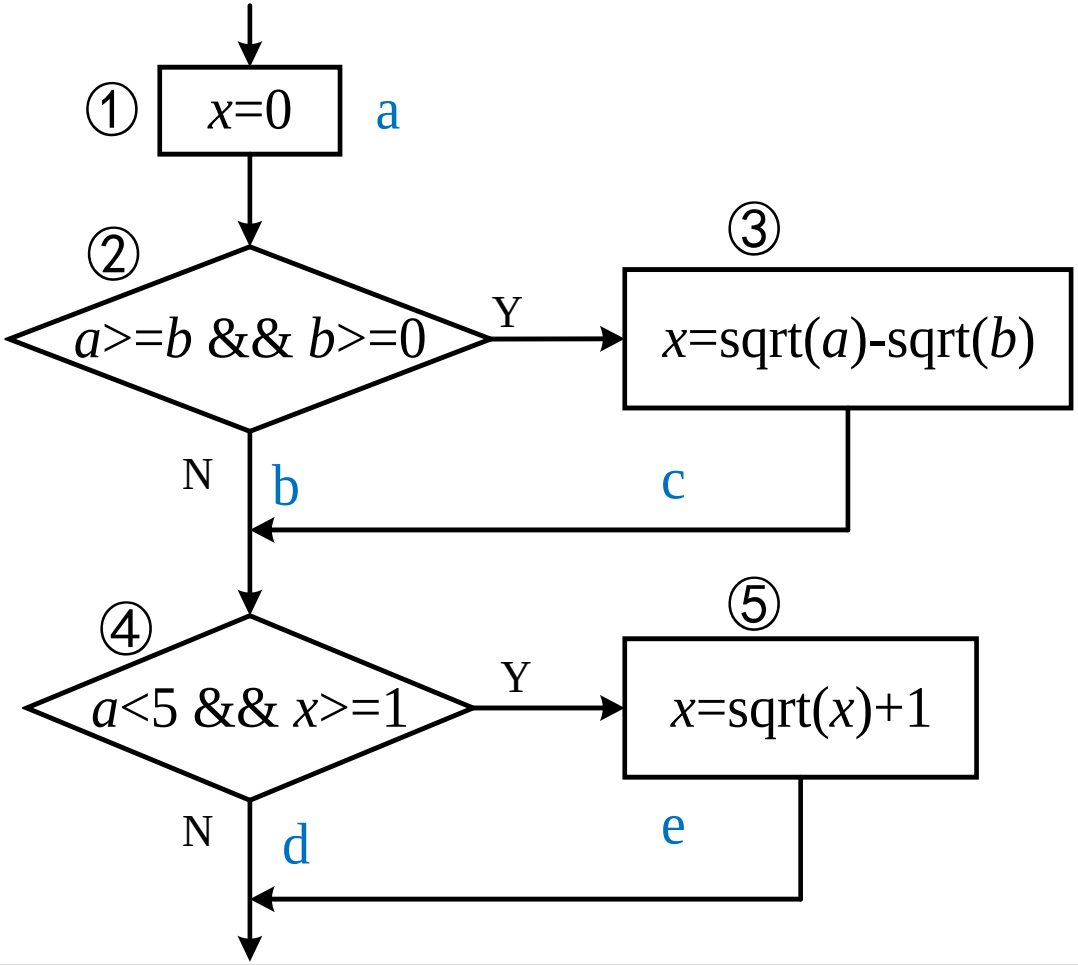
语句覆盖：
4, 1



Test Cases

语句覆盖：
4, 1

判定覆盖：
4, 1 (ce)
4, 9 (bd)

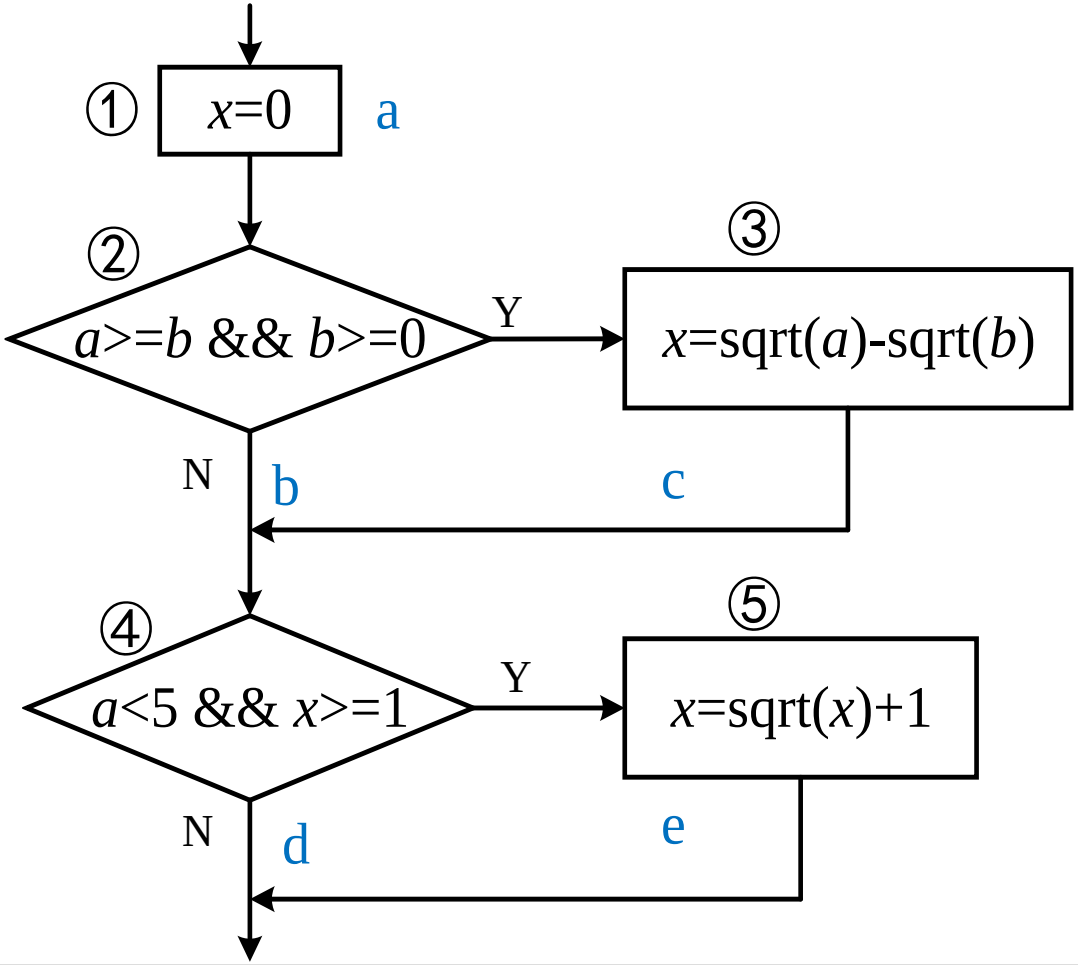


Test Cases

语句覆盖：
4, 1

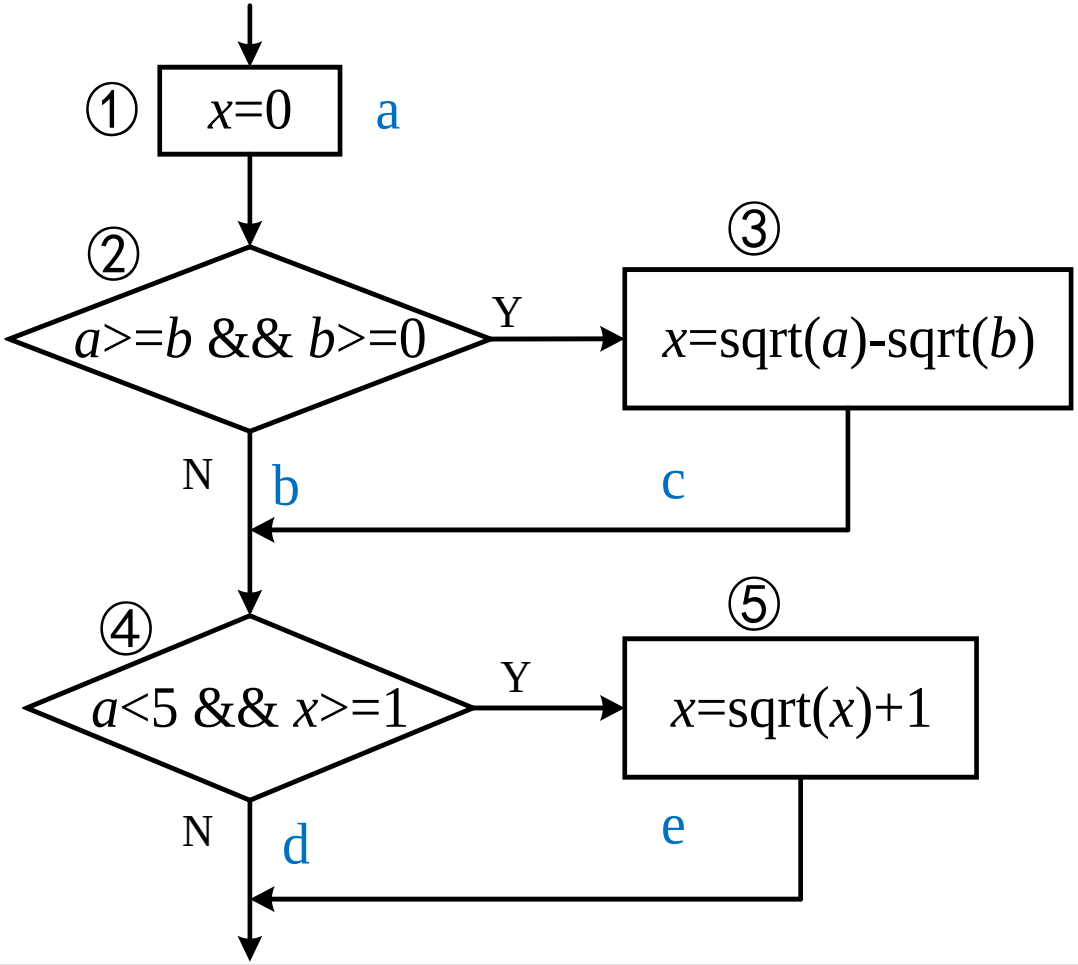
判定覆盖：
4, 1 (ce)
4, 9 (bd)

条件覆盖：
-4, -1 (F1 F2 T3 F4)
9, 1 (T1 T2 F3 T4)



Test Cases

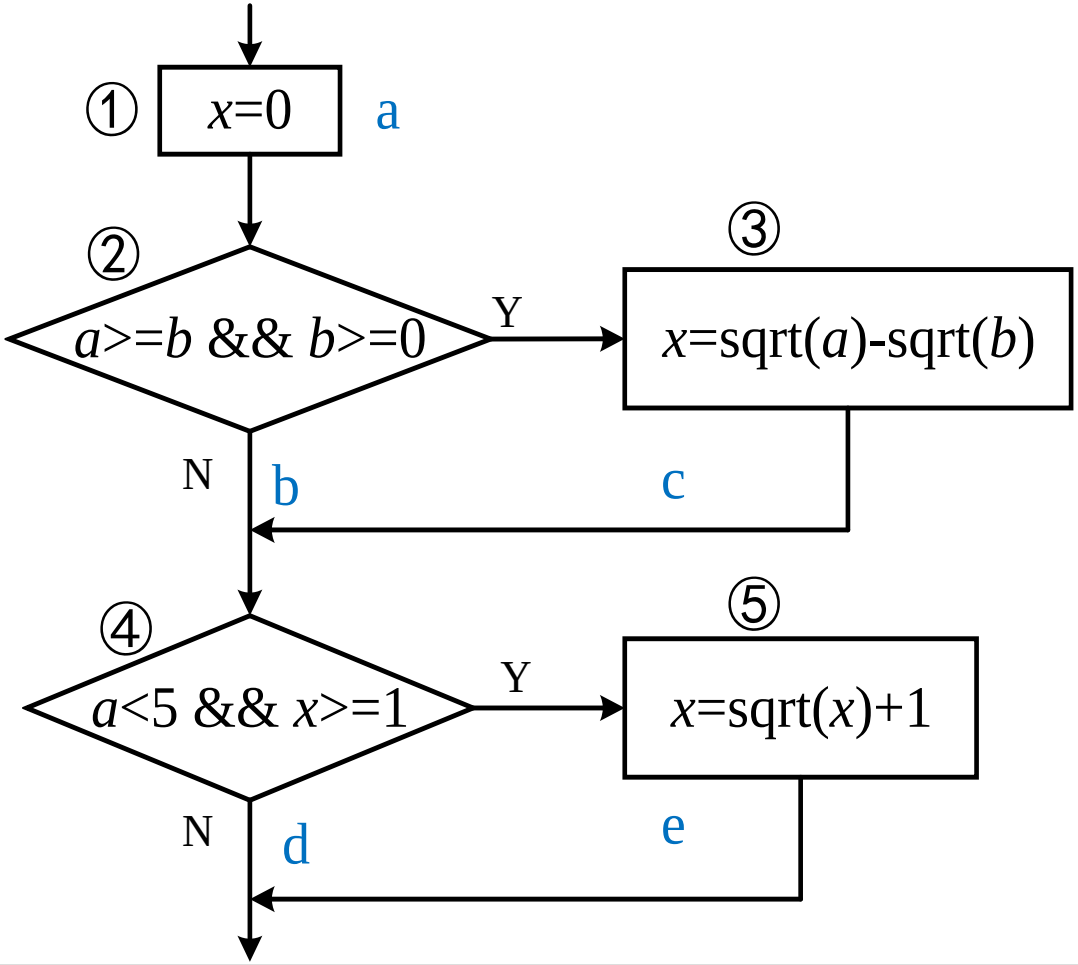
判定/条件覆盖：
4, 1 (T1 T2 T3 T4, ce)
-4, -1 (F1 F2 T3 F4, bd)
9, 1 (T1 T2 F3 T4, cd)



Test Cases

判定/条件覆盖：
4, 1 (T1 T2 T3 T4, ce)
-4, -1 (F1 F2 T3 F4, bd)
9, 1 (T1 T2 F3 T4, cd)

条件组合覆盖：
6, -1 (T1F2, F3F4)
0, 6 (F1T2, F3F4)
4, 1 (T1T2, T3T4)
9, 1 (F1F2, F3T4)
0, -1 (T1F2, T3F4)

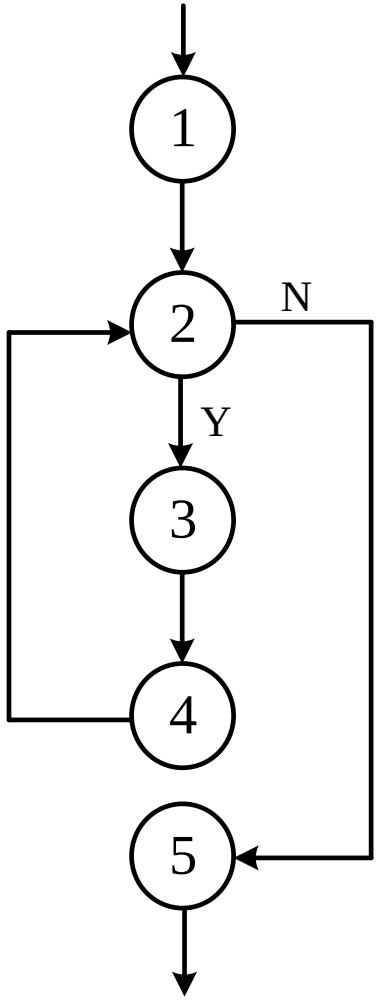


Program Control Flow Graph

- A program, no matter how complicated it is, is made of three basic structure elements: sequence, branch and loop. A program control flow graph be used to describe it.
- In the flow graph, two elements are involved: **node** and **edge**.

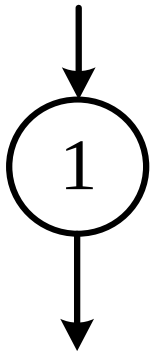
Node and Edge

- Each circle in the graph is a node, and if there are several statements in a sequence structure, they are actually viewed as one node.
- An edge shows the sequence of nodes.
- The edge which enter the program is called the entry edge, and the one which leave the program is called the exit edge. The other edges are the active edge.



Sequence Structure

- In the sequence structure, no matter how many statement there are, only one node are included. And it is obvious that one entry edge and one exit edge are involved. Obviously, it contains only one path.

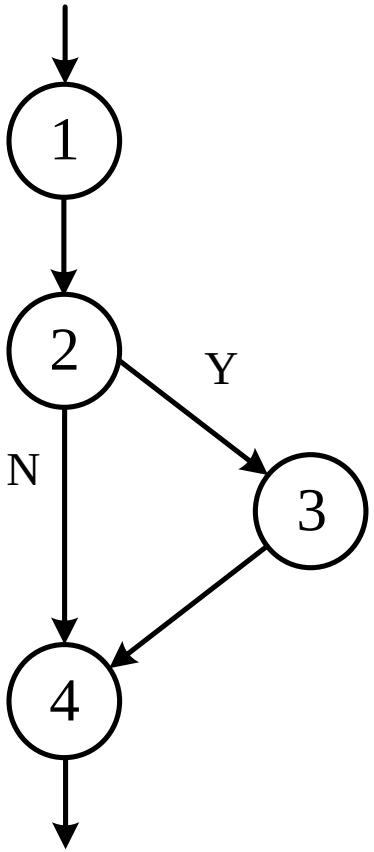


Branch Structure

- The branch structure usually involves the IF statement. When it is a simple IF statement as shown here, two paths are contained.

P1：1-2-4

P2：1-2-3-4

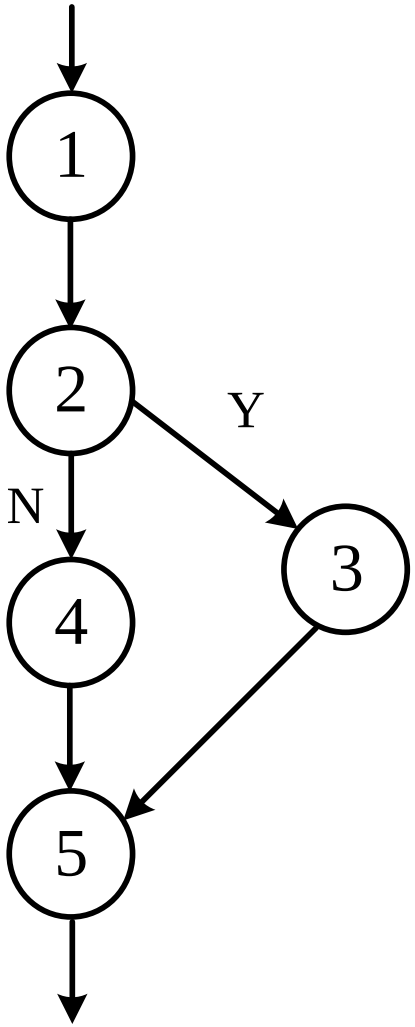


Branch Structure

- When the IF statement contains an ELSE, there are also two paths. And we can find more paths when it comes to some more complicated structures (e.g. SWITCH-CASE).

P1：1-2-3-5

P2：1-2-4-5



Loop Structure

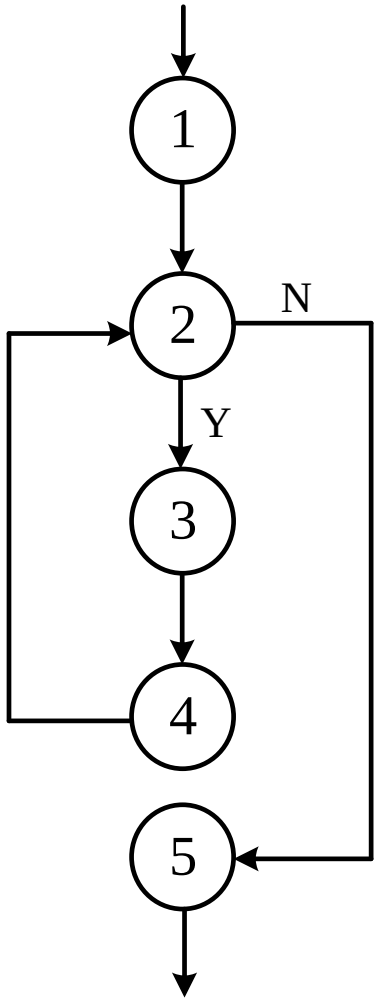
- The FOR and WHILE loop statement involves many paths.

P1：1-2-5

P2：1-2-3-4-2-5

P3：1-2-3-4-2-3-4-2-5

……



Loop Structure

- When the BREAK is involved, some possible paths are shown here.

P1：1-2-6

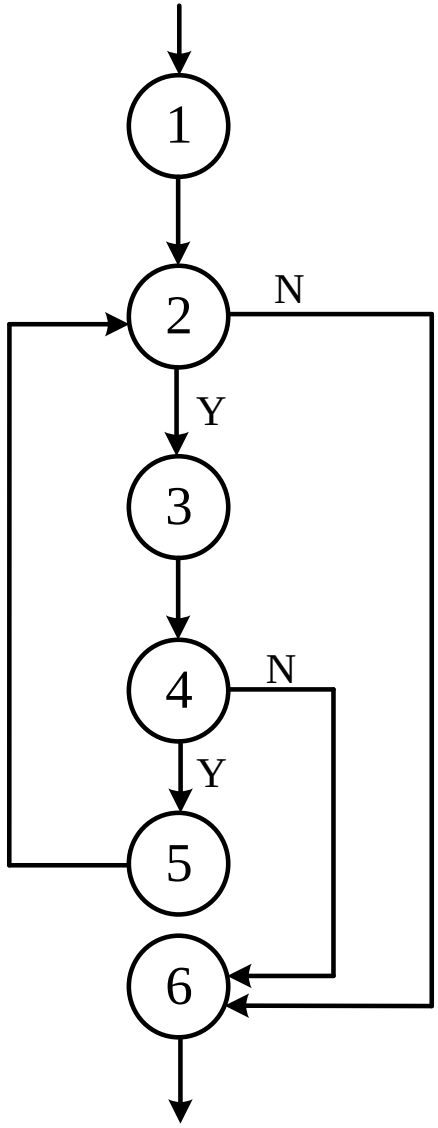
P2：1-2-3-4-5-2-6

P3：1-2-3-4-6

P4：1-2-3-4-5-2-3-4-5-2-6

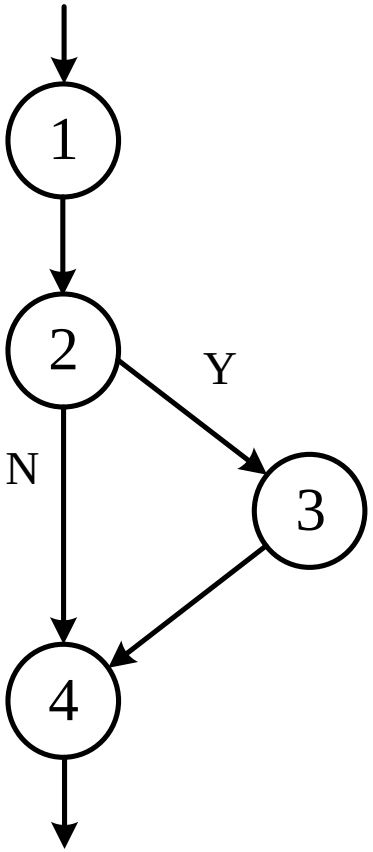
P5：1-2-3-4-5-2-3-4-6

……



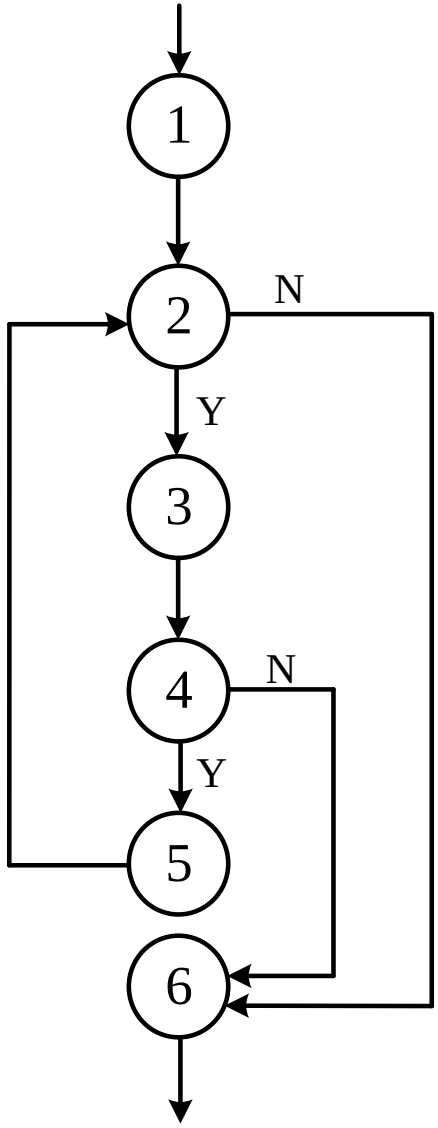
Independent Path

- For the existing paths, if the new path includes at least one edge that is not involved before, then the new path is called an independent path.
- For example, the existing path is 1-2-3-4, and then 1-2-4 is its independent path because the edge 2-4 is not involved in 1-2-3-4.



Independent Path

- For another example, if the existing paths are 1-2-6, 1-2-3-4-6 and 1-2-3-4-5-2-6, then the path 1-2-3-4-5-2-3-4-6 is not their independent path because all the edges of the new path have been encountered in the existing paths.



Cyclomatic Complexity Formula

- The cyclomatic complexity *V* (圈复杂度) for a program graph can be calculated as follows. *E* denotes the number of active edges and *N* denotes the number of nodes. *P* represents the number of the connected graph and here equals the constant 1, i.e. *P*=1.

$$V = E - N + 2P$$

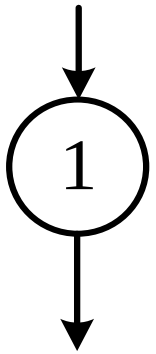
Cyclomatic Complexity Formula

- Therefore, let E_0 denote all the edges involved, we have a simplified formula:

$$V = E_0 - N$$

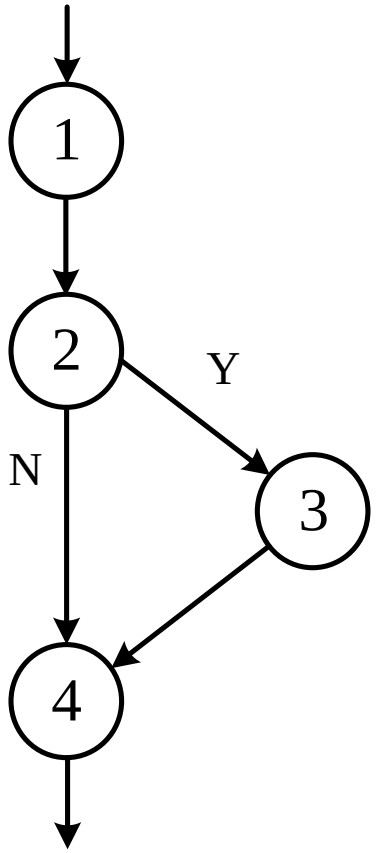
Sequence Structure

- For the sequence structure, we have $V = 0\text{-}1\text{+}2 = 1$ (or $V = 2\text{-}1 = 1$)
- As a conclusion, the cyclomatic complexity of any sequence structure is 1.



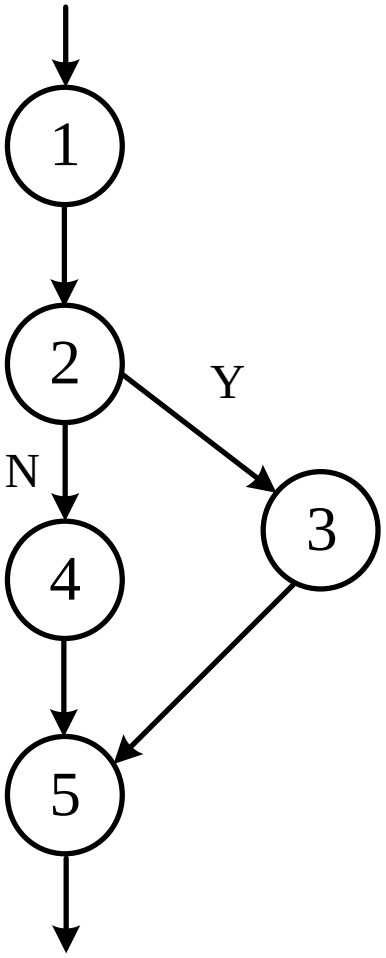
Branch Structure

- For the branch structure shown here, we have $V = 4\text{-}4+2 = 2$ (or $V = 6\text{-}4 = 2$).



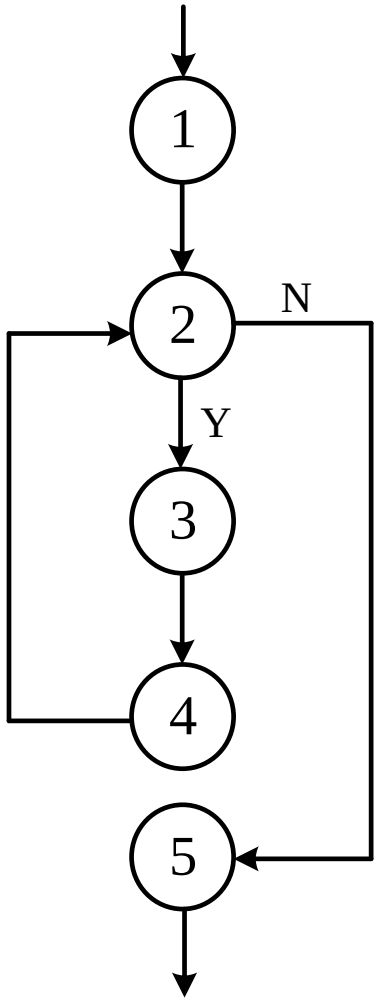
Branch Structure

- For the branch structure with IF-ELSE, we have $V = 7\text{-}5 = 2$.



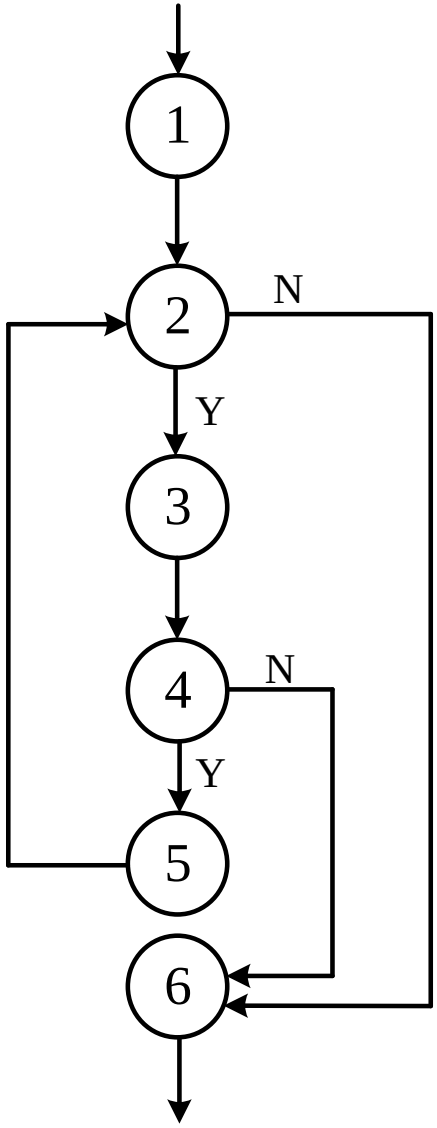
Loop Structure

- For a typical FOR or WHILE loop:
  $V = 7\text{-}5 = 2$



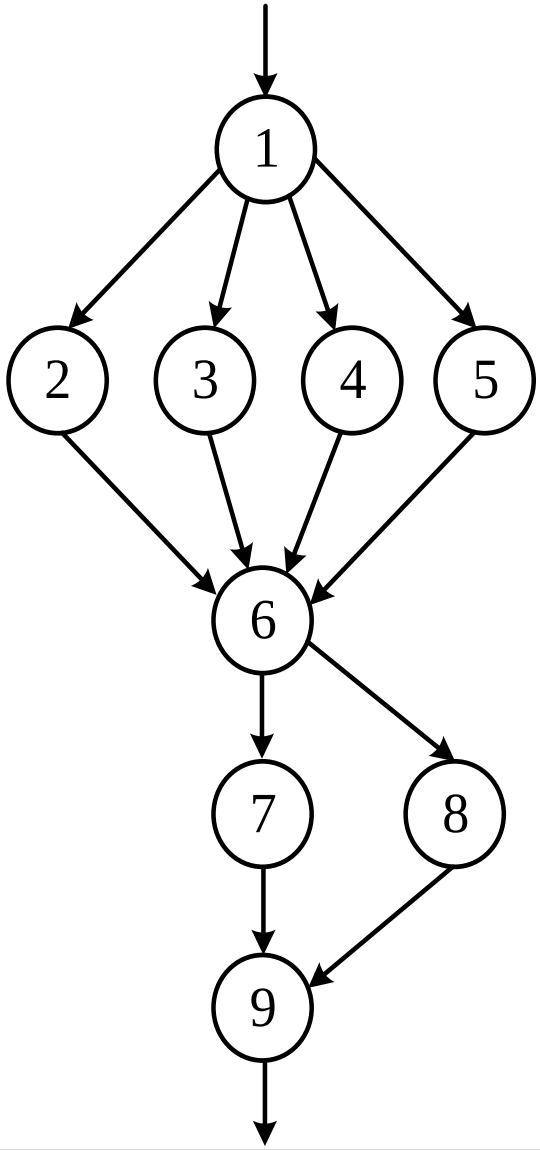
Loop Structure

- For a loop with BREAK:
  $V = 9\text{-}6 = 3$



Example 3

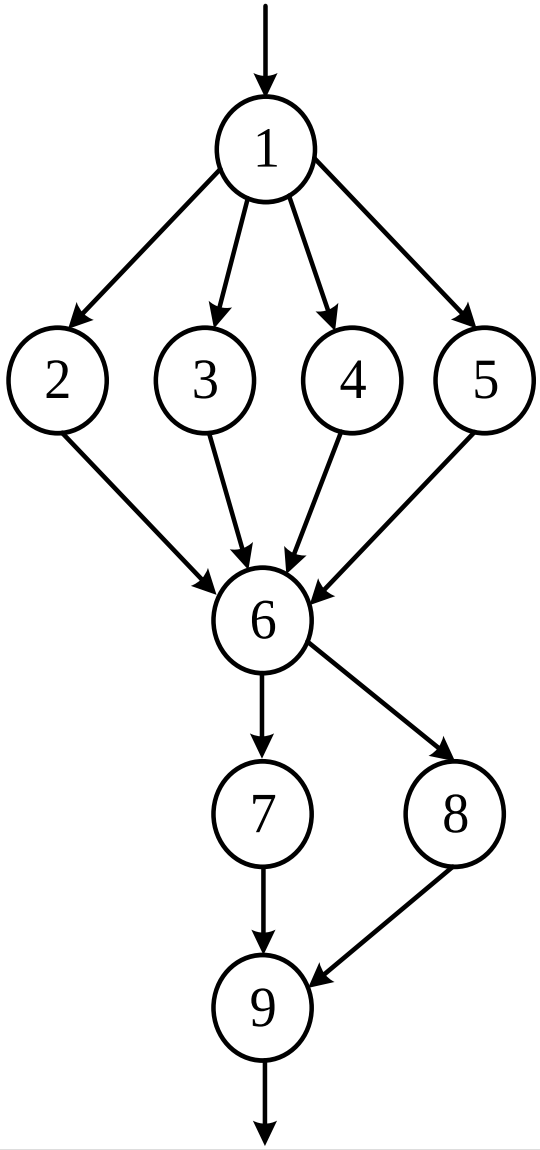
- Calculate the cyclomatic complexity for the graph.



Example 3

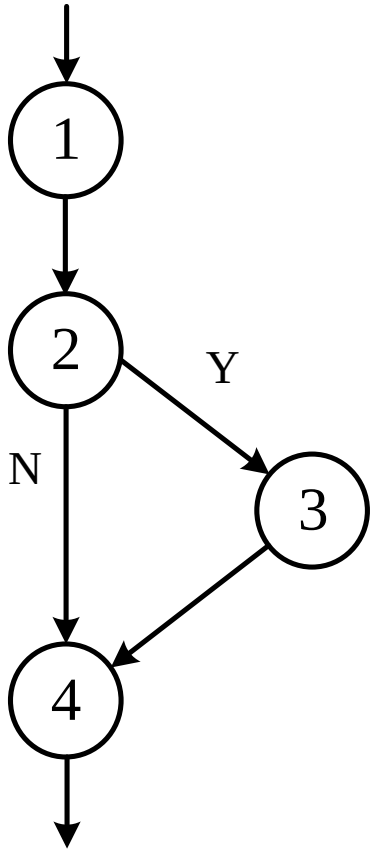
- Calculate the cyclomatic complexity for the graph.

$V = 14\text{-}9 = 5$

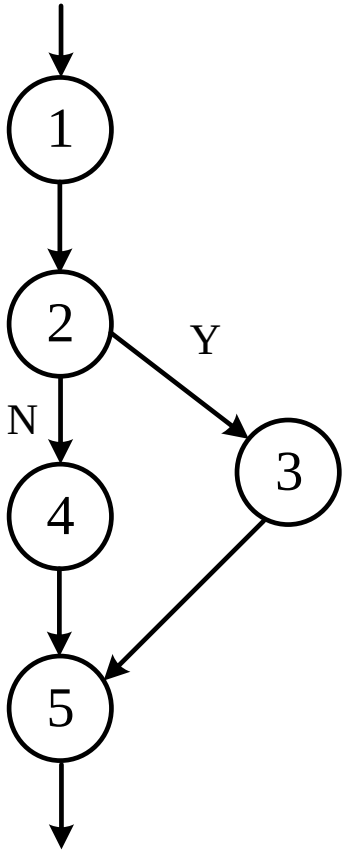


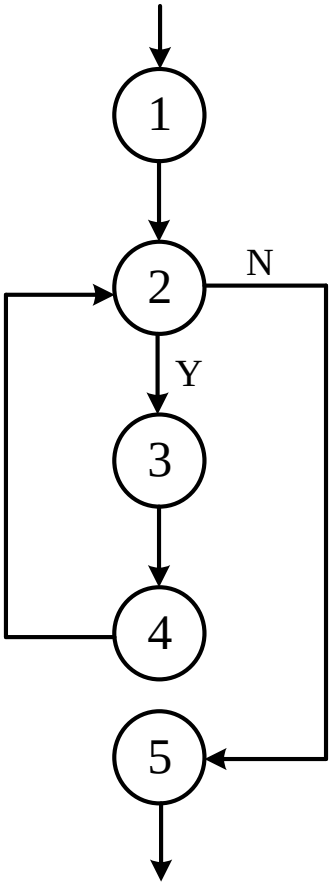
Meaning of Cyclomatic Complexity

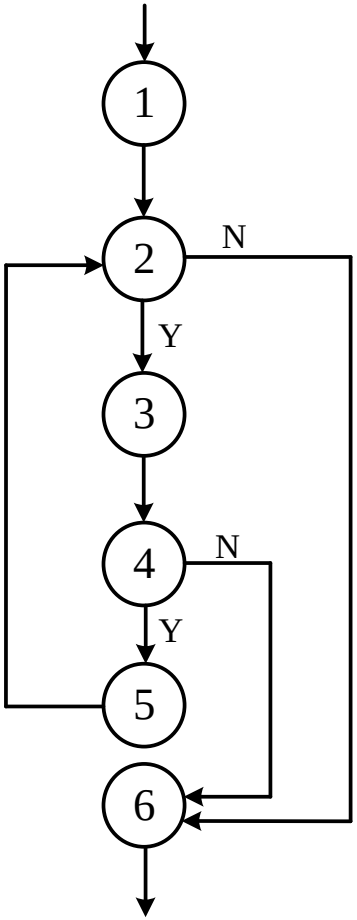
- Cyclomatic complexity denotes the number of regions into which the graph divides the plane.
- For example, the nodes 2, 3 and 4 form an independent region and that means that the plane is divided into 2 regions. Hence, the corresponding cyclomatic complexity is 2.

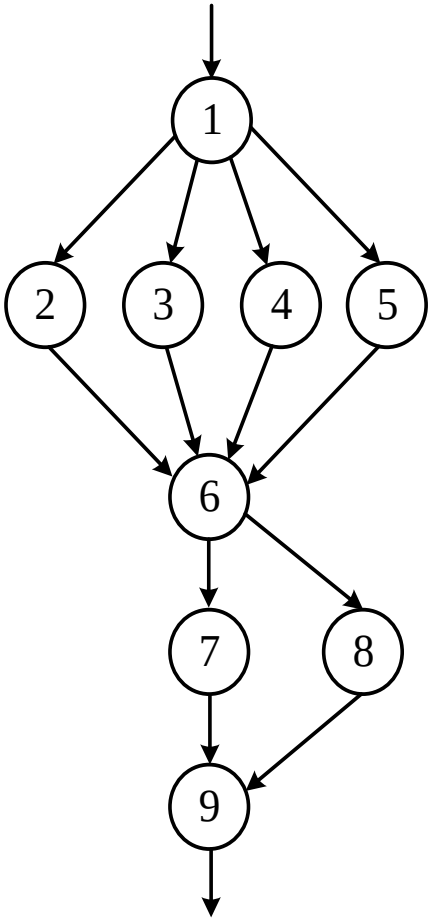


Meaning of Cyclomatic Complexity



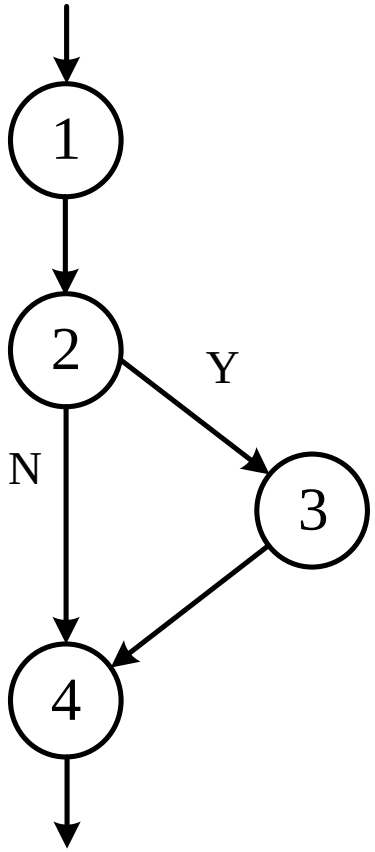






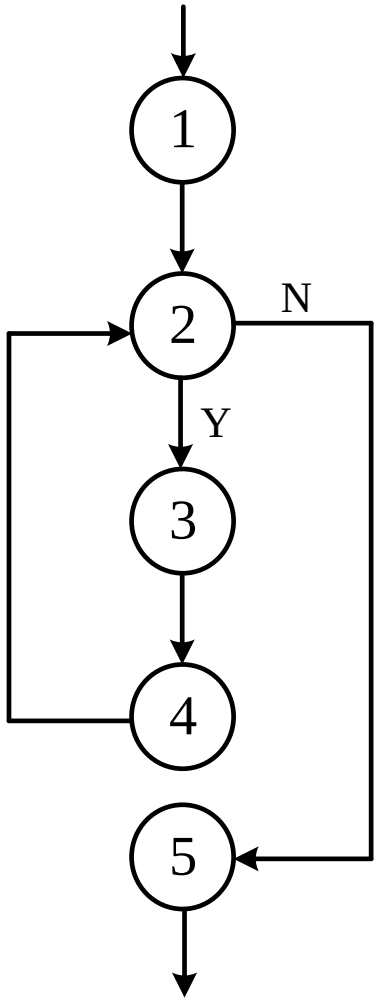
Another Method

- Cyclomatic complexity has another method for calculating: the number of binary predicate nodes plus 1.
- For example, Node 2 is a binary one with two branches (Y and N). There is one binary predicate node and thus we have $V = 1+1 = 2$.



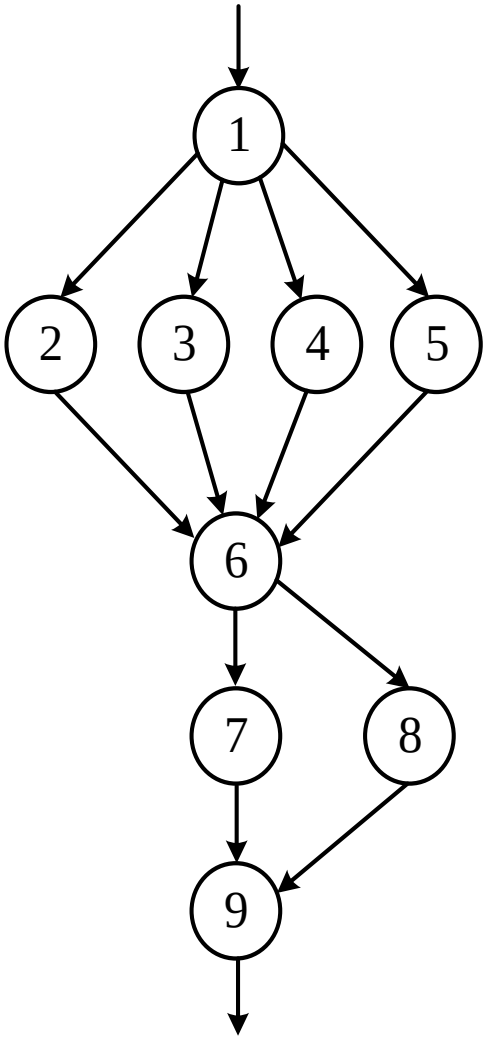
Another Method

- For another example, the FOR or WHILE loop structure seems to have no IF statement, but in Node 2, a decision must be made, so it is also a binary predicate node, and thus $V = 1+1 = 2$.



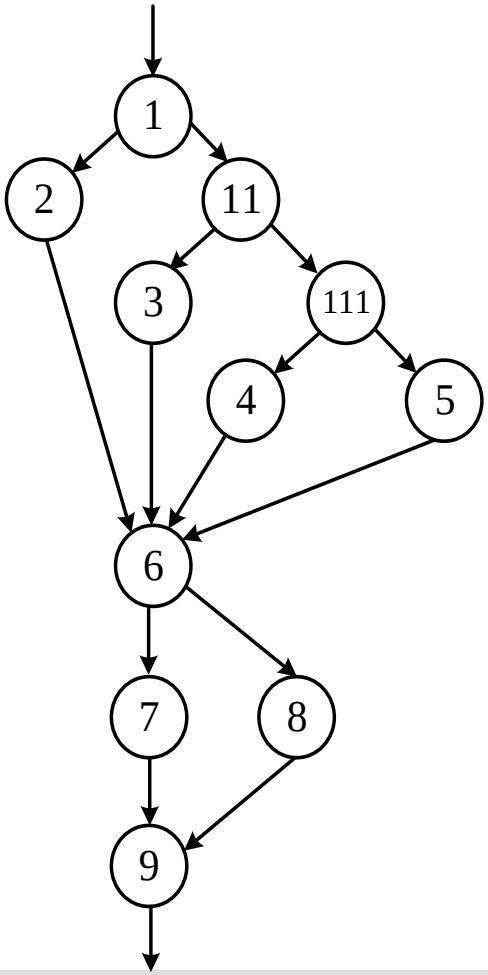
Another Method

- In this issue, Node 1 has four branches, which means that it is not a binary predicate node and that we can not use the method.
- However, the node can be transformed into several binary ones. It is the process to transform SWITCH-CASE into IF-ELSEIF-ELSE.



Another Method

- After the transformation, Node 1 becomes three binary predicate nodes, and we have $V = 4+1 = 5$.



Summary

- Three methods for calculating the cyclomatic complexity:
  - Two Formulae

$$V = E - N + 2P \qquad V = E_0 - N$$

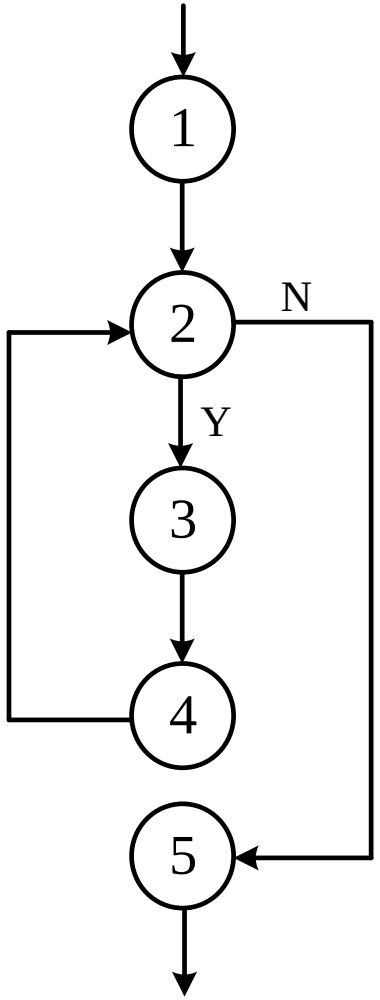
  - Number of Plane Region
  - Number of Binary Predicate Nodes plus 1

Basic Path Set

- The paths comprise the set of some basic paths. For example, the basic path set consists of two independent paths.

P1: 1-2-5

P2: 1-2-3-4-2-5



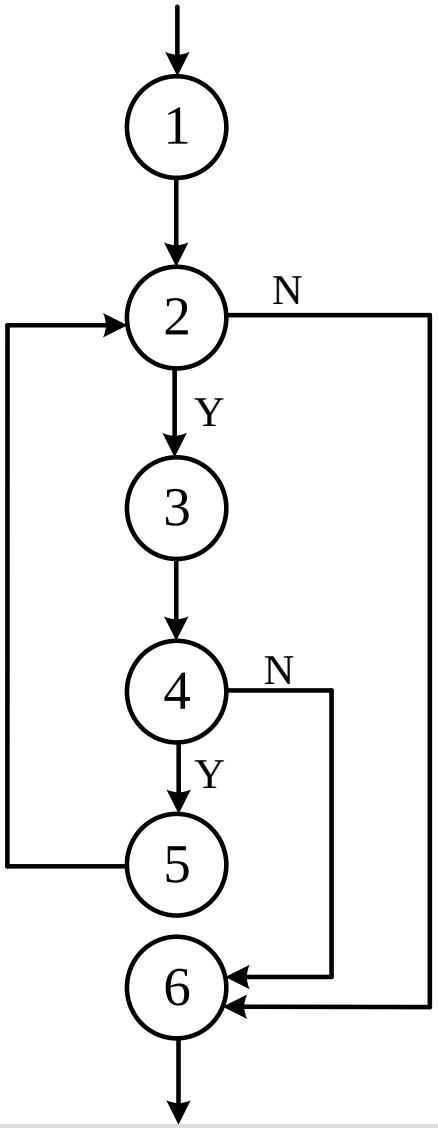
Basic Path Set

- For another example, the basic path set consists of three independent paths.

P1: 1-2-6

P2: 1-2-3-4-6

P3: 1-2-3-4-5-2-6



Basic Path Set

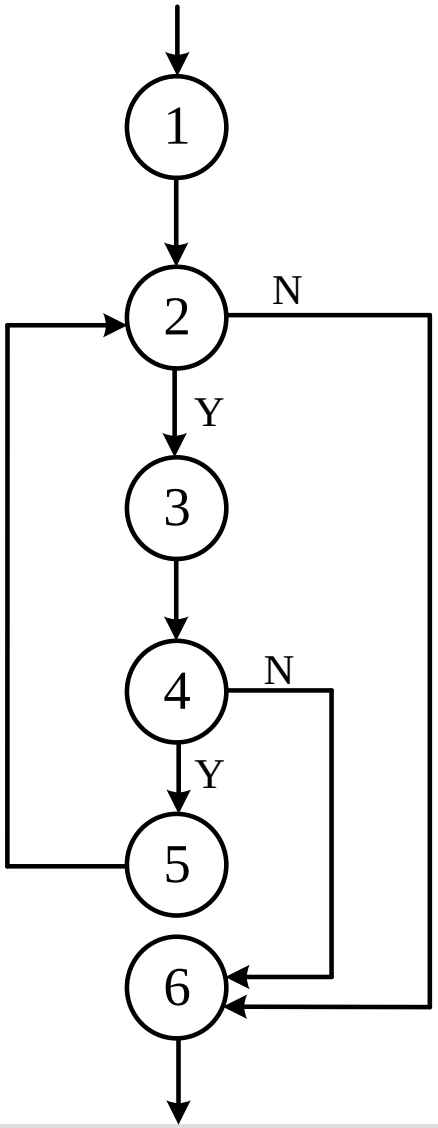
- For another example, the basic path set consists of three independent paths.

  P1: 1-2-6

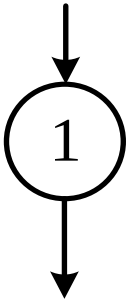
  P2: 1-2-3-4-6

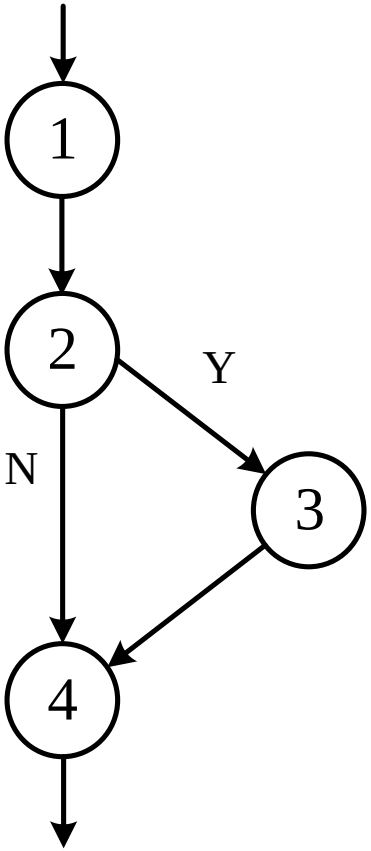
  P3: 1-2-3-4-5-2-6

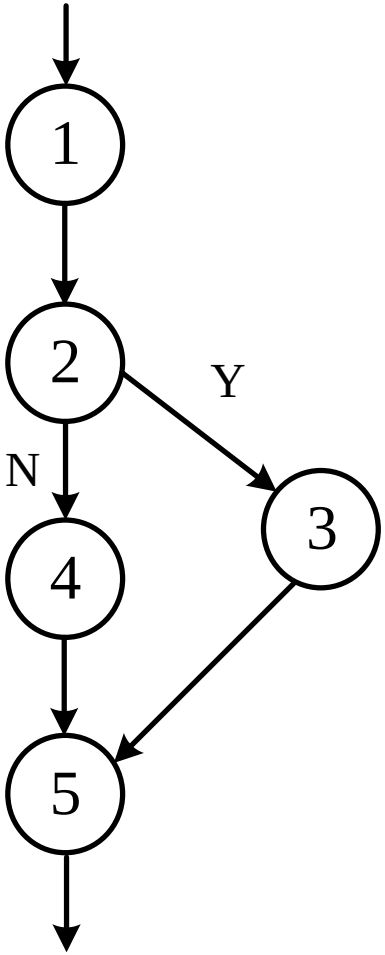
- So how many paths should be involved in a basic path set? The answer is the cyclomatic complexity, which is exactly the number.

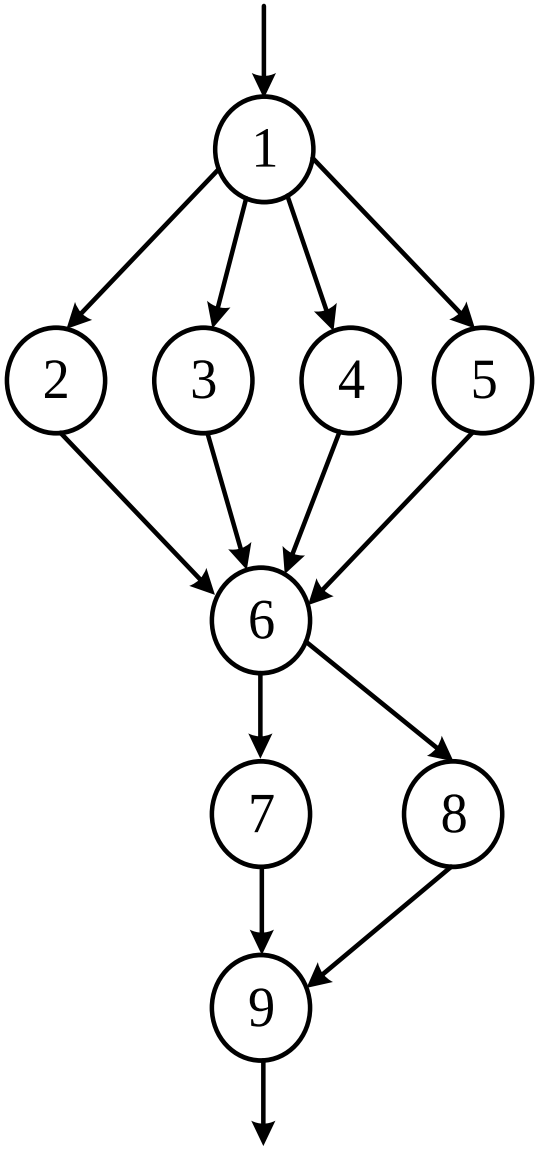


Basic Path Set









Example 4

- Find out a set of basic paths.

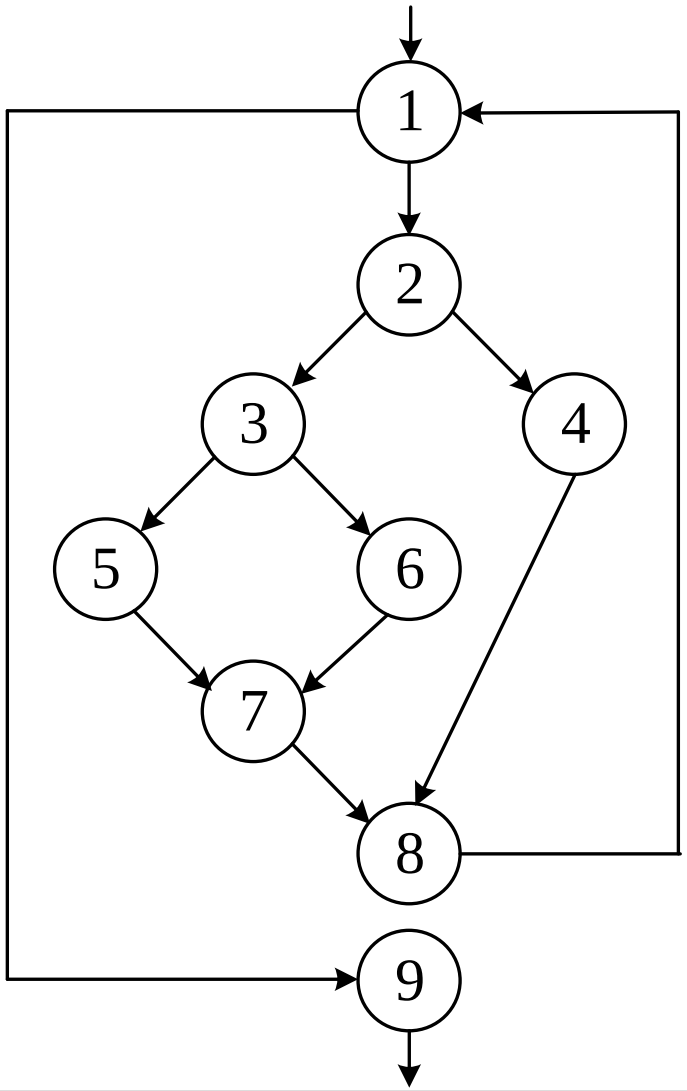
V=4

P1: 1-9

P2: 1-2-4-8-1-9

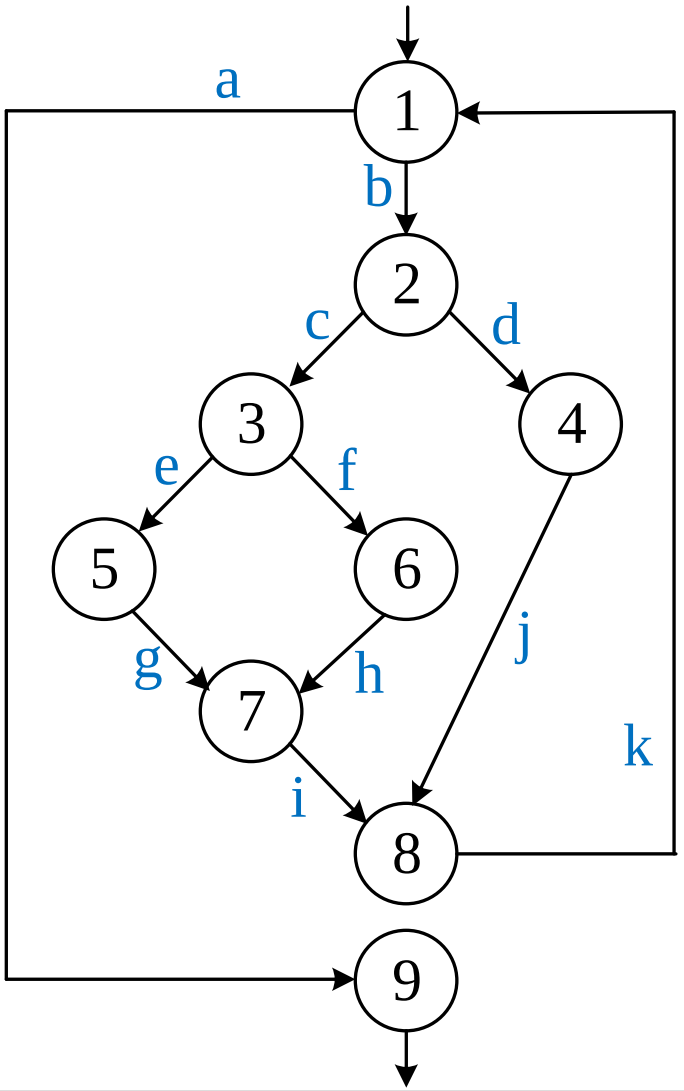
P3: 1-2-3-5-7-8-1-9

P4: 1-2-3-6-7-8-1-9



Graph Matrix

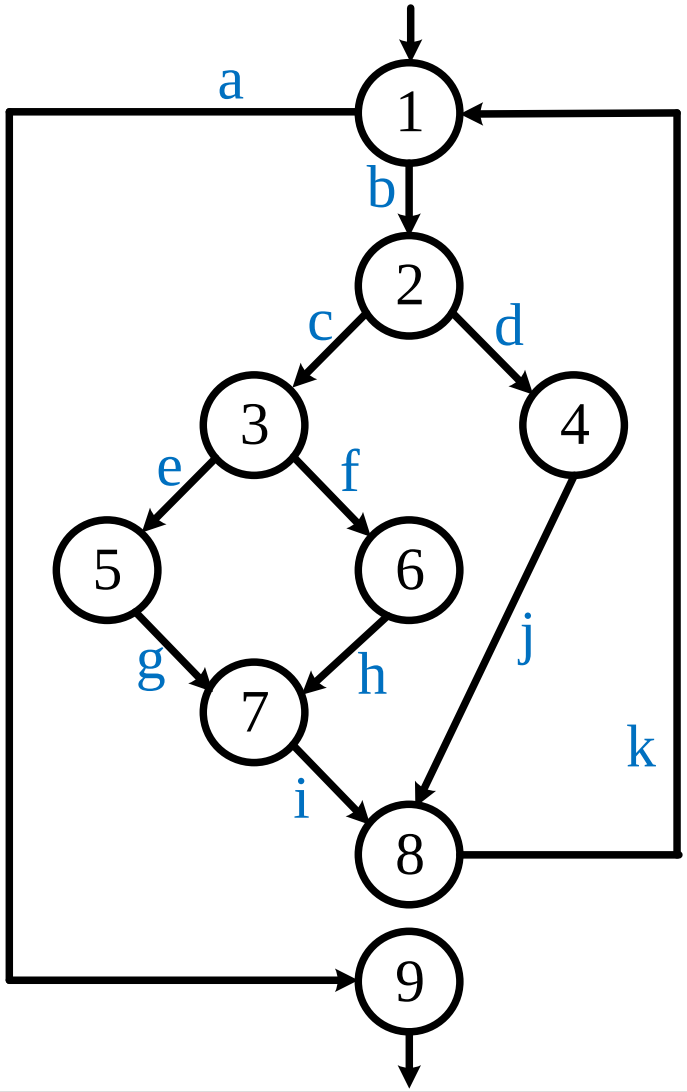
- A graph can be also transferred into a square matrix.



Graph Matrix

- A graph can be also transferred into a square matrix.

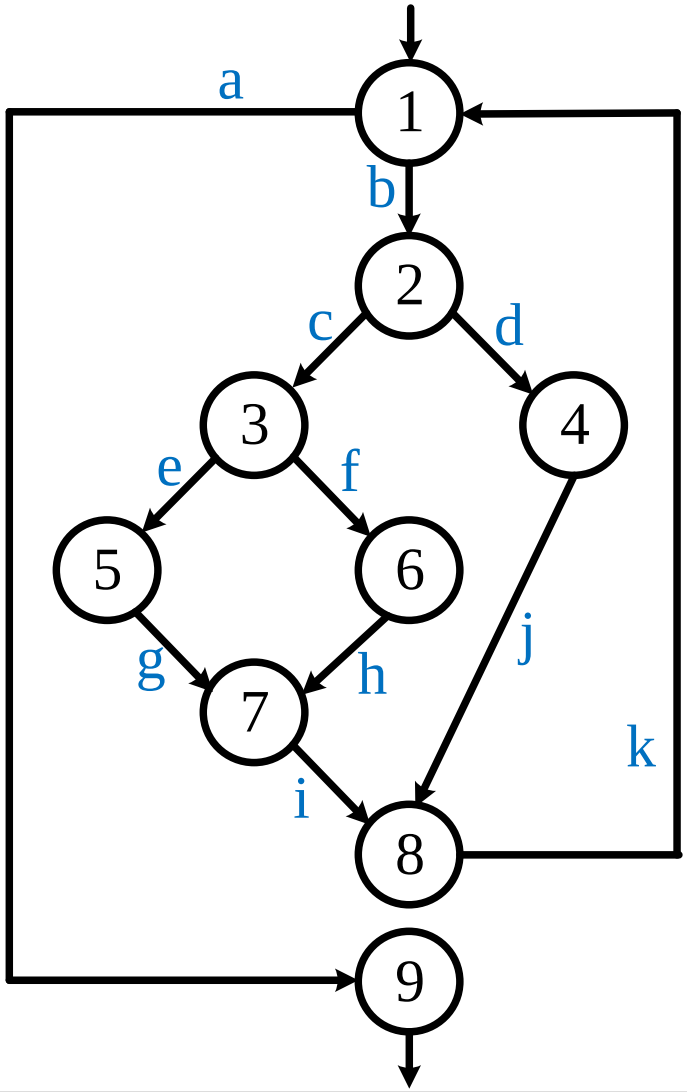
	1	2	3	4	5	6	7	8	9
1									
2									
3									
4									
5									
6									
7									
8									
9									



Graph Matrix

- A graph can be also transferred into a square matrix.

	1	2	3	4	5	6	7	8	9
1		b							a
2			c	d					
3					e	f			
4								j	
5							g		
6							h		
7								i	
8	k								
9									



Principles for Testing

- **原则1：测试用例中一个必需部分是对预期输出或结果的定义**
  - 如果没有定义预期结果，即没有比较标准
- **原则2：程序员应当避免测试自己编写的程序**
  - 心理上，程序员很难对自己建立起来的程序进行破坏性审查
  - 程序员对规范的错误理解会导致程序错误，同样的误解导致无法测试这样的错误
  - 调试由程序员本人进行则更有效

Principles for Testing

- **原则3：编写软件的组织不应当测试自己编写的程序**
  - 由于组织和项目经理总是希望保证进度，降低成本。定量地衡量软件的可靠性极其困难，即便是合理规划和实施的测试过程也可能被认为降低了完成进度和成本目标的可能性，因此编程组织难以客观测试自己的软件
  - 具有和程序员相似的心理
  - 由客观、独立的第三方进行测试更经济

- **原则4：应该彻底检查每个测试的执行结果**
  - 测试中发现的错误常常被遗漏

Principles for Testing

- **原则5：测试用例的编写不仅应当根据有效和预期的输入情况，而且也应当根据无效和未预料到的输入情况**
  - 测试软件时经常会忽略无效和未预料到的情况（使软件缺乏鲁棒性）
  - 在软件产品中突然暴露出来的许多问题是当程序以某些新的或未预料到的方式运行时发现的
  - 针对未预料到的和无效的输入情况的测试用例更能发现问题
- **原则6：检查程序是否“未做其应该做的”仅是测试的一半，测试的另一半是检查程序是否“做了不应该做的”**
  - 必须检查程序是否有我们不希望的副作用，这样的程序仍然是不正确的程序

Principles for Testing

- **原则7：应避免测试用例用后即弃，除非软件本身就是一个一次性的软件**
  - 测试用例在测试后就消失了，一旦软件需要重新测试，就需要重新设计这些测试用例，结果或是再次投入大量的工作，或草草测试，这样对程序的更改如果导致某个先前可执行的部分发生故障，这个故障往往不会被发现
  - 回归测试：当程序其他部件发生更动后重新执行之前保留的测试用例
- **原则8：计划测试工作时不应默许假定不会发现错误**
  - 错误的测试定义：测试是证明程序正确运行的过程
  - 正确的测试定义：测试是为发现错误而执行程序的过程

Principles for Testing

- **原则9：程序某部分检查出很多错误时要警惕更多错误**
  - 残存错误的可能性与已检出错误可能性之间成正比
  - 错误总是倾向于聚集存在，在一个程序中，某些部分要比其他部分更容易存在错误，为了使测试获得更大的程序，最好对这些容易存在错误的部分进行额外的测试
- **原则10：软件测试是一项极富创造性、极具智力挑战性的工作**
  - 测试软件的创造性很可能超过了开发软件所需的创造性
  - 发现有效的测试用例是一项创造性的工作

THANK YOU!