Ch2 Code Unit Testing

Write Code to Test Code(3)

JUnit 5 mockito JaCoCo

Instructor: Haiying SUN

E-mail: hysun@sei.ecnu.edu.cn

Office: ECNU Science Build B1104

Available Time: Wednesday 8:00 -12:00 a.m.

Agenda

- Introduction to Unit Testing
- Common Code Defect Categories
- Unit Tests Design Heuristic Rules
- Unit Tests Implementation
  - Junit & Mockito & Qualified test scripts
- Code Test Adequacy Criteria
  - Control flow based & Jacoco
  - Data flow based
  - Mutation Based
- Code Test Generation

The Original of Test Adequacy Criteria

[充分性问题] 判断测试集合在软件上的表现是否能够充分反映该软件的总体表现

1975年, Goodenough等测试数据必须具备什么性质才是一个彻底的测试，即成功的测试意味着被测程序的正确性[1]:

1. 测试充分性准则的可靠性
2. 测试充分性准则的正确性

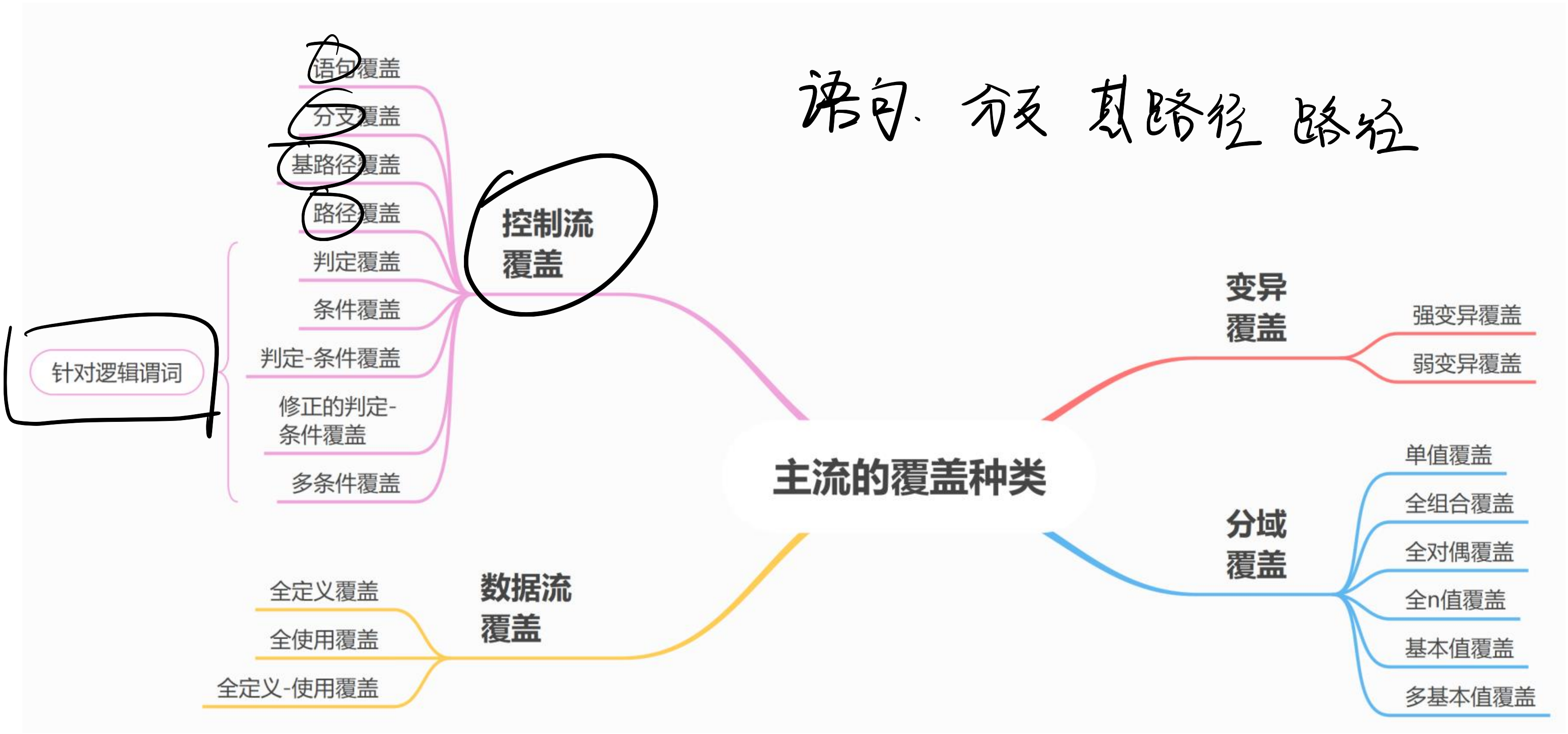
1976年, Howden证明了Goodenough等提出的充分性理论存在缺陷[2]，

1. 测试充分性准则的可靠性
2. 测试充分性准则的正确性

矛盾

1. J. B. Goodenough, S. L. Gerhart, Toward a theory of test data selection, IEEE Transaction on Software Engineering, SE-3 (June), 1975.
2. W. E. Howden, Reliability of the path analysis testing strategy, IEEE Transaction on Software Engineering, SE-2, (Sept.), 208–215, 1976.
3. 朱鸿 金陵紫著，软件质量保障与测试，科学出版社，1997.

Test Adequacy Criteria



语句 分支 基路径 路径

主流的覆盖种类

- 控制流覆盖
  - 语句覆盖
  - 分支覆盖
  - 基路径覆盖
  - 路径覆盖
  - 判定覆盖
  - 条件覆盖
  - 判定-条件覆盖
  - 修正的判定-条件覆盖
  - 多条件覆盖
  - 针对逻辑谓词
- 数据流覆盖
  - 全定义覆盖
  - 全使用覆盖
  - 全定义-使用覆盖
- 变异覆盖
  - 强变异覆盖
  - 弱变异覆盖
- 分域覆盖
  - 单值覆盖
  - 全组合覆盖
  - 全对偶覆盖
  - 全n值覆盖
  - 基本值覆盖
  - 多基本值覆盖

Control Based Code Test Adequacy Criteria

- Control Flow Graph
- Statement Coverage
- Branch Coverage
- Path Coverage
- Prime Path Coverage
- Logical Coverage Criteria
    - Decision Coverage
    - Condition Coverage
    - Decision-Condition Coverage
    - Modified Condition/Decision Coverage (MC/DC)
    - Multiple Condition Coverage

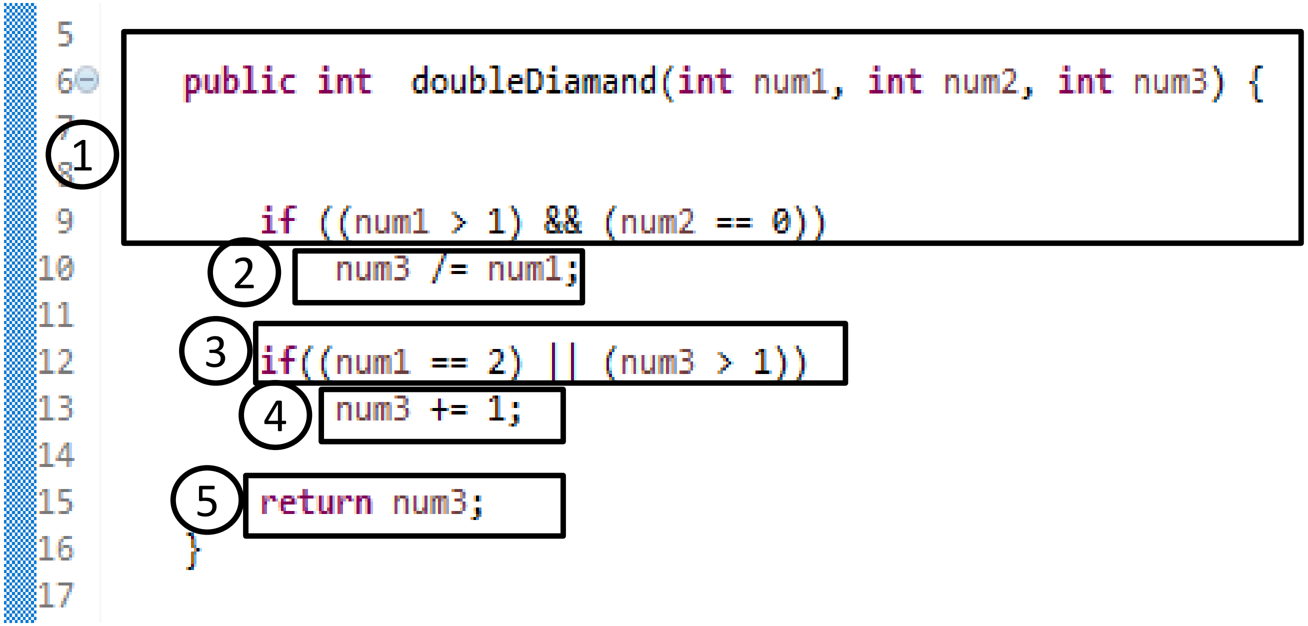
Control Based Code Test Adequacy Criteria

- Control Flow Graph
- Statement Coverage
- Branch Coverage
- Path Coverage
- Prime Path Coverage
- Logical Coverage Criteria
  - Decision Coverage
  - Condition Coverage
  - Decision-Condition Coverage
  - Modified Condition/Decision Coverage (MC/DC)
  - Multiple Condition Coverage

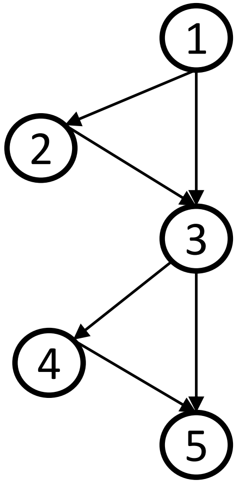
Control Flow Graph

- A CFG models all executions of a method by describing control structures
  1. Nodes : Statements or sequences of statements (basic blocks)
     - **Basic Block** : A sequence of statements such that if the first statement is executed, all statements will be
  2. Edges : Transfers of control

Example

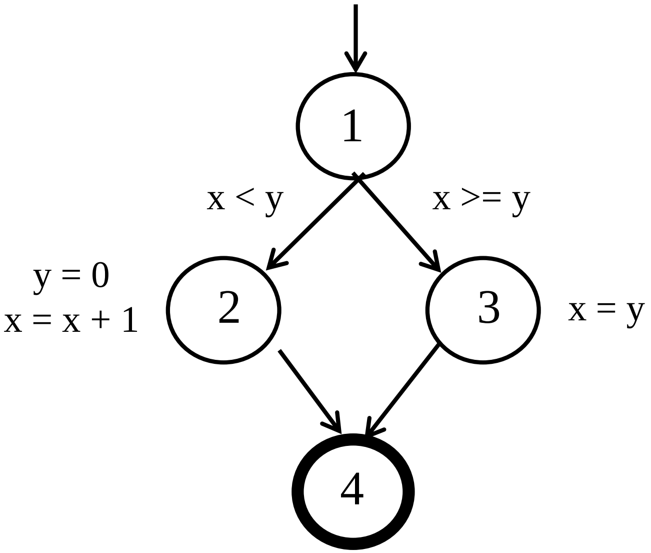


```
public int  doubleDiamand(int num1, int num2, int num3) {


    if ((num1 > 1) && (num2 == 0))
        num3 /= num1;

    if((num1 == 2) || (num3 > 1))
        num3 += 1;

    return num3;
}
```

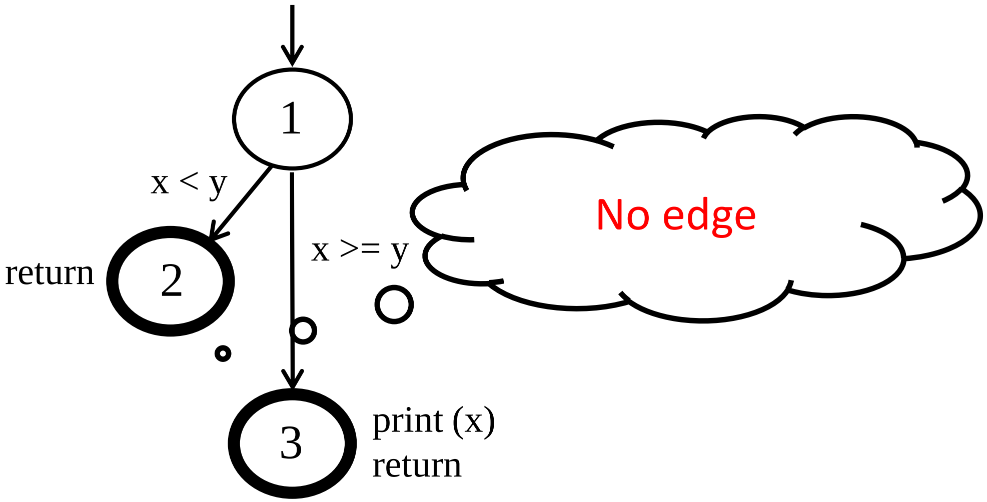


Control Flow Graph

```
if (x < y) {
   y = 0;
   x = x + 1;
}
else {
   x = y;
}
```



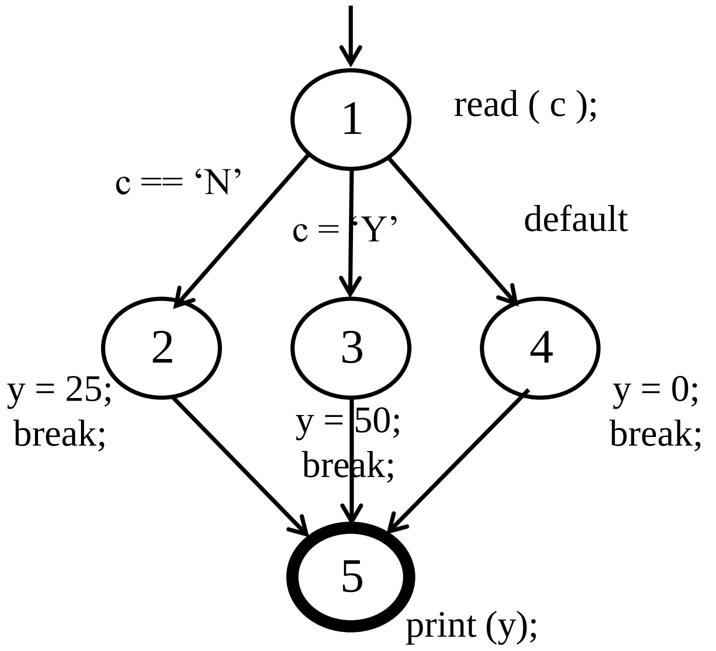
```
if (x < y) {
   return;
}
print (x);
return;
```



Control Flow Graph

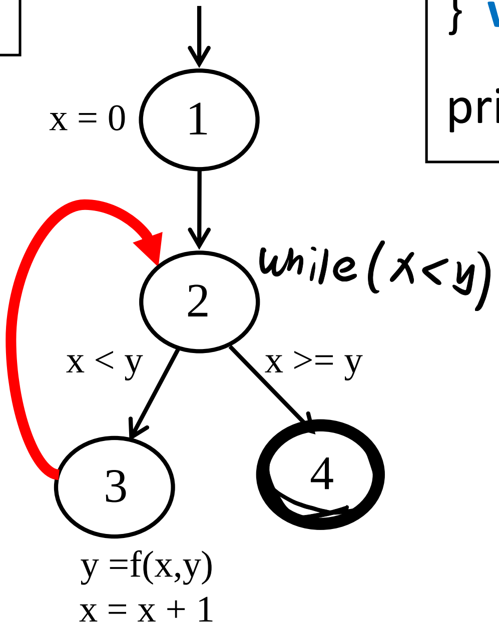
Control Flow Graph

```
read ( c) ;
switch ( c ) {
   case 'N':
      y = 25;
      break;
   case 'Y':
      y = 50;
      break;
   default:
      y = 0;
      break;
}
print (y);
```

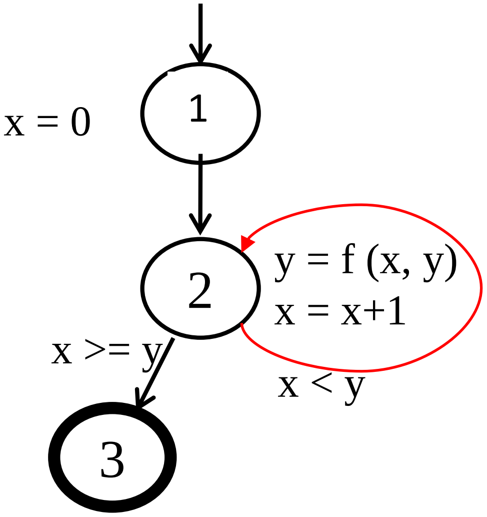


Control Flow Graph

```
x = 0;
while (x < y) {
   y = f (x, y);
   x = x + 1;
}
```

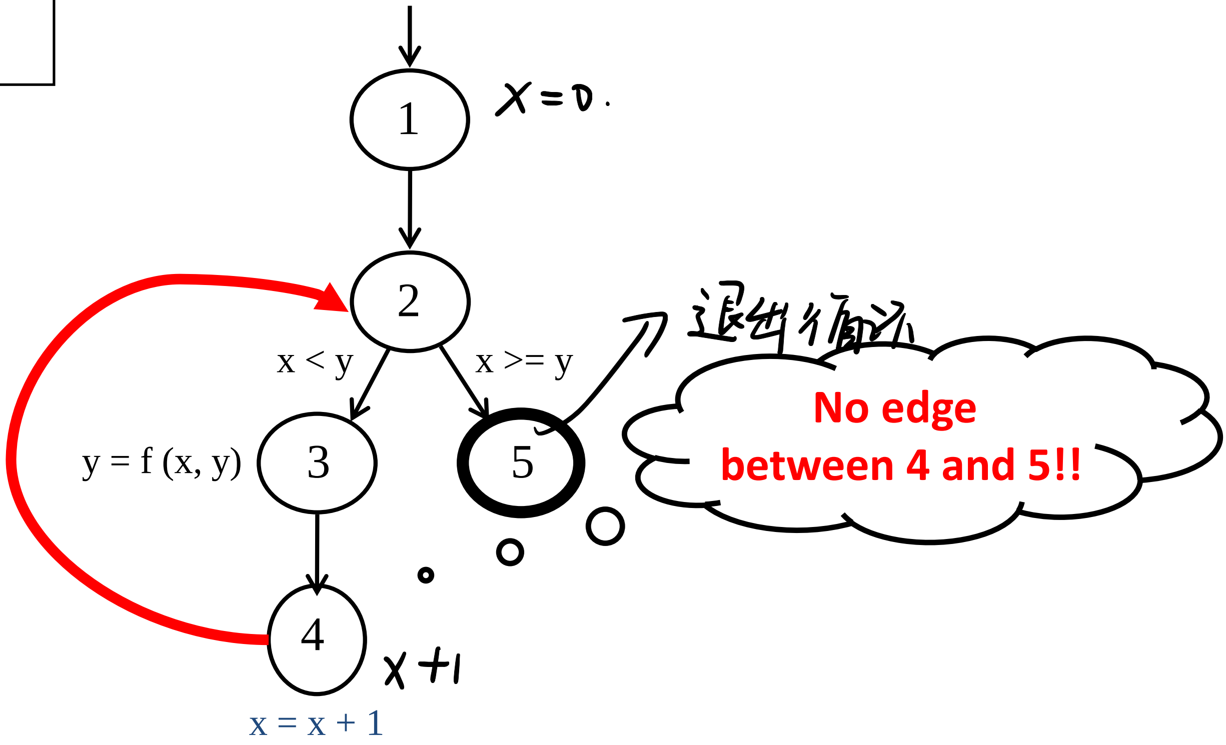


```
x = 0;
do {
   y = f (x, y);
   x = x + 1;
}  while (x < y);
print (y)
```



Control Flow Graph

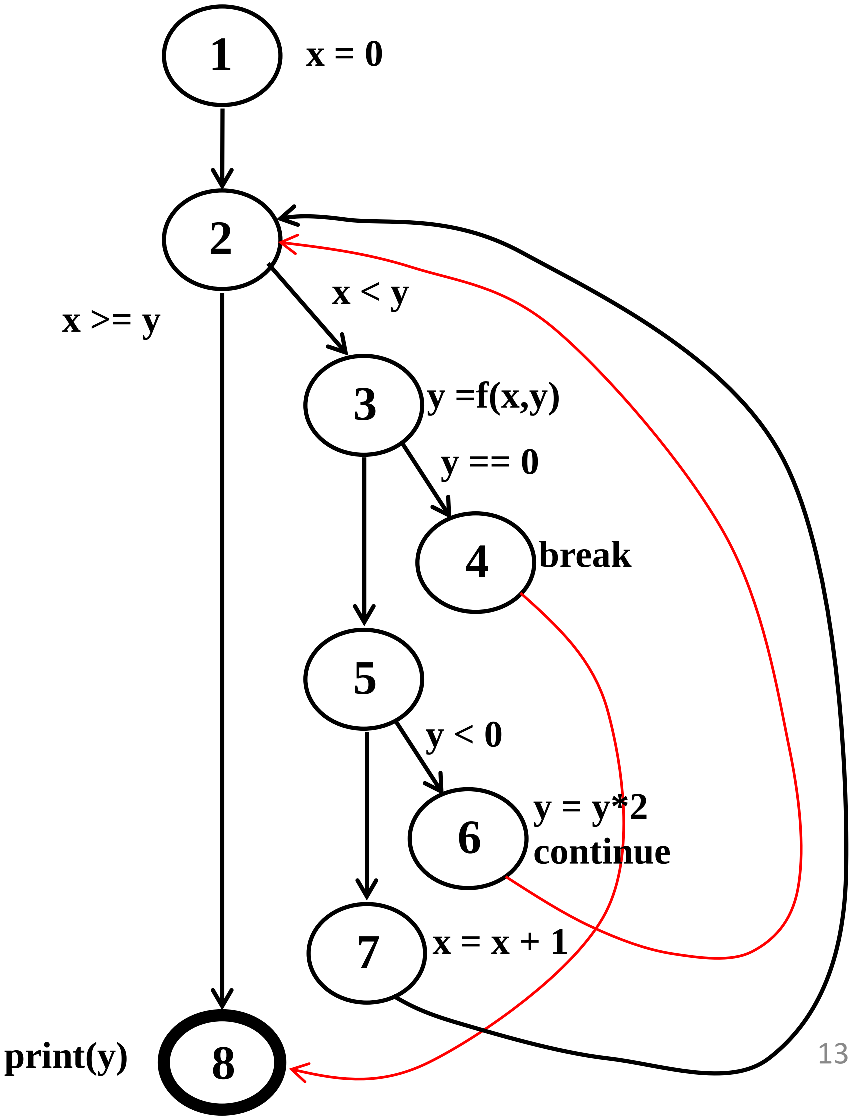
```
for (x = 0; x < y; x++) {
    y = f (x, y);
}
```



No edge between 4 and 5!!

Control Flow Graph

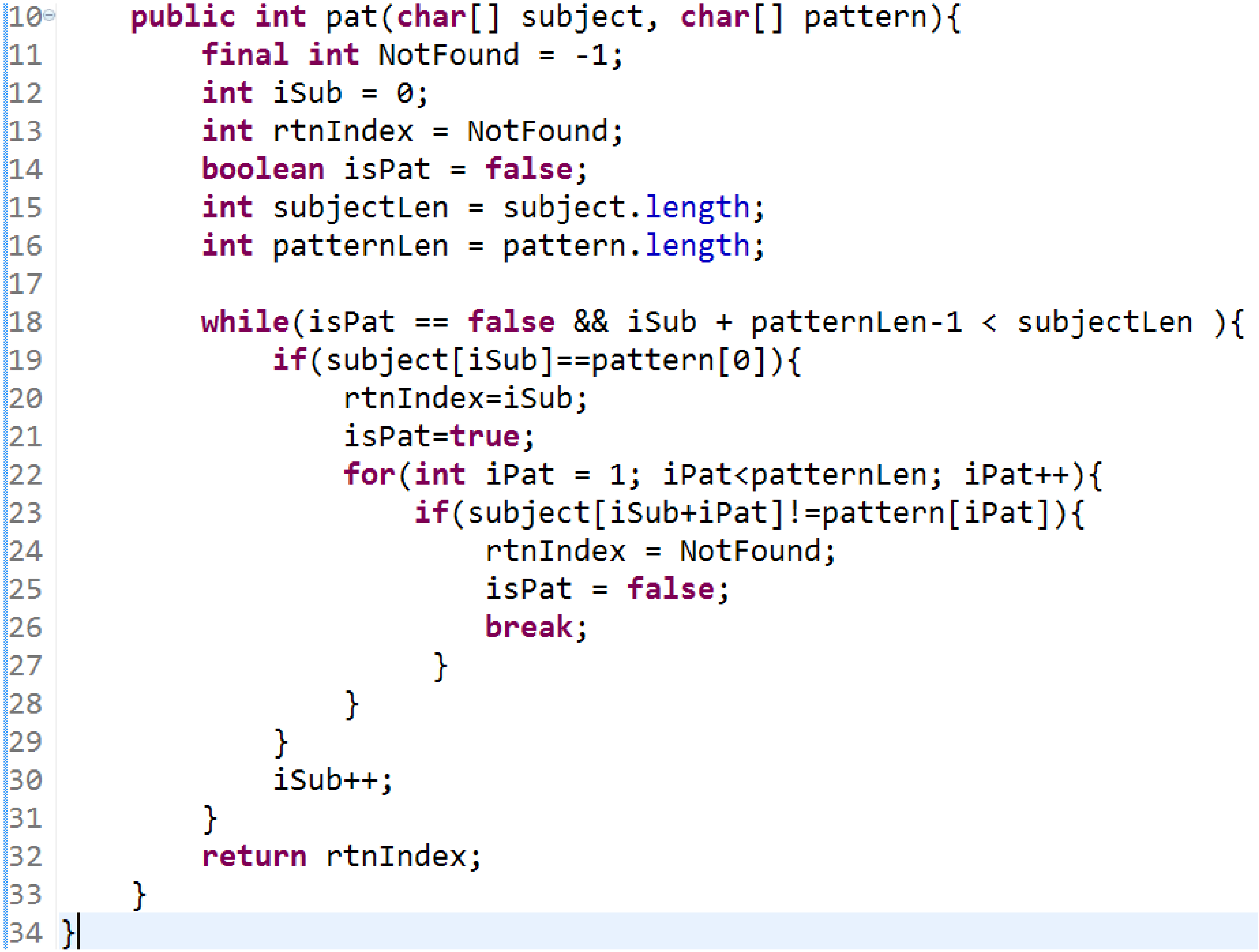
```
x = 0;
while (x < y) {
  y = f (x, y);
  if (y == 0)  {
    break;
  } else if y < 0)  {
    y = y*2;
    continue;
  }
  x = x + 1;
}
print (y);
```



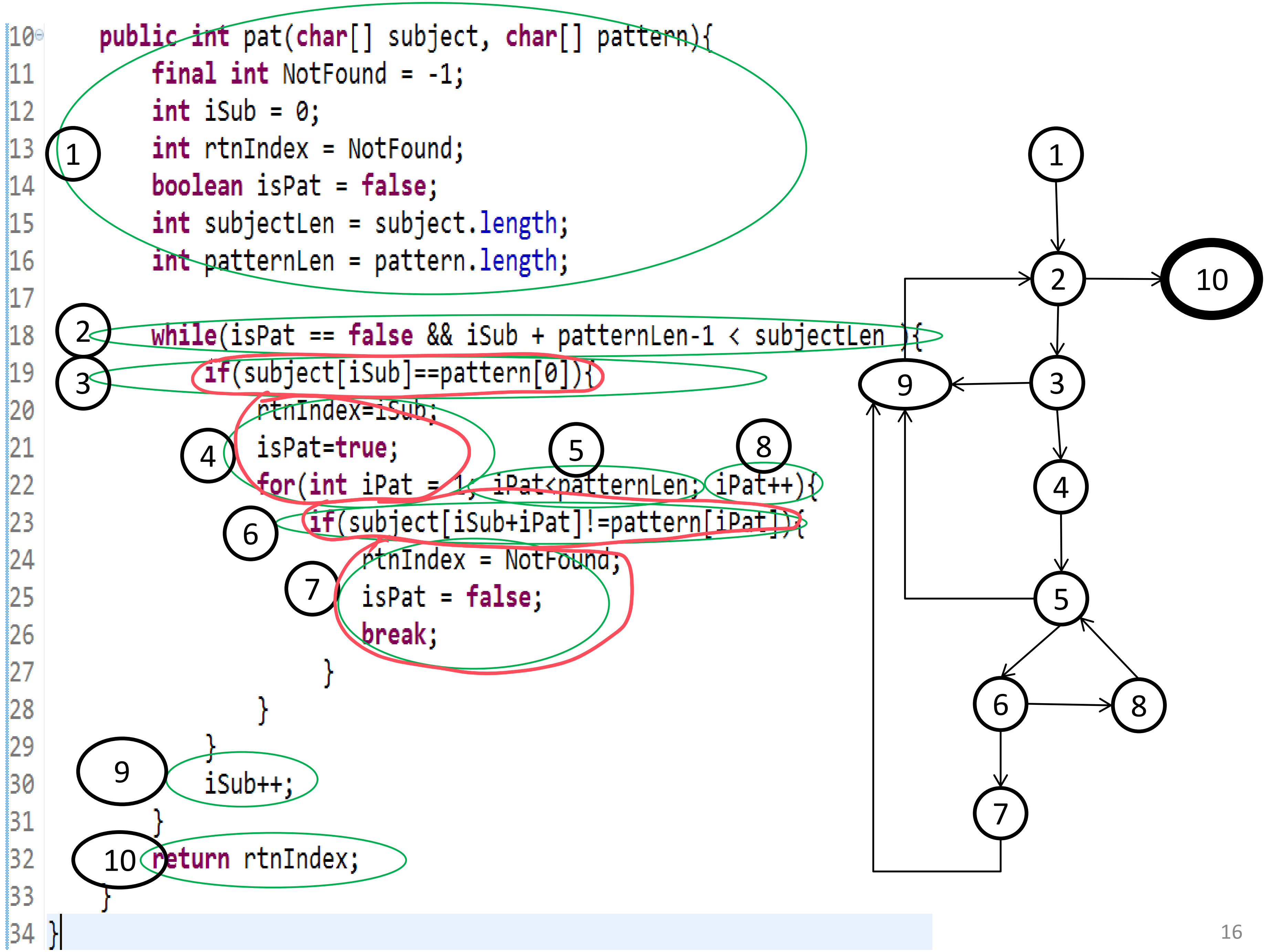
Control Flow Graph

- More Java Logical Structures
  1. for each
  2. try…catch…/throws/….
  3. concurrent logics
  4. lambda expressions
  5. …..

Example



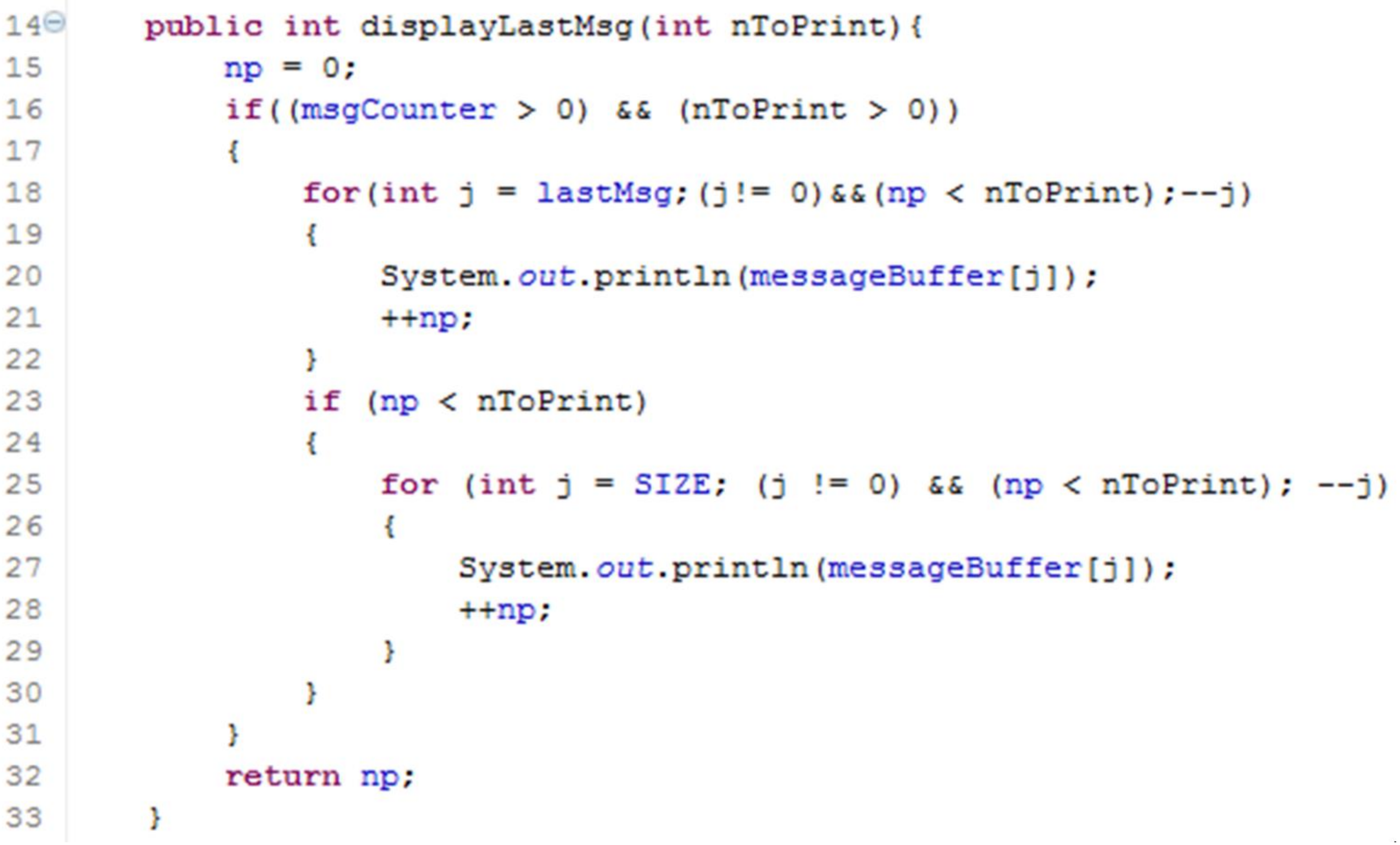
```java
    public int pat(char[] subject, char[] pattern){
        final int NotFound = -1;
        int iSub = 0;
        int rtnIndex = NotFound;
        boolean isPat = false;
        int subjectLen = subject.length;
        int patternLen = pattern.length;

        while(isPat == false && iSub + patternLen-1 < subjectLen ){
            if(subject[iSub]==pattern[0]){
                rtnIndex=iSub;
                isPat=true;
                for(int iPat = 1; iPat<patternLen; iPat++){
                    if(subject[iSub+iPat]!=pattern[iPat]){
                        rtnIndex = NotFound;
                        isPat = false;
                        break;
                     }
                }
            }
            iSub++;
        }
        return rtnIndex;
    }
}
```



```
    public int pat(char[] subject, char[] pattern){
        final int NotFound = -1;
        int iSub = 0;
        int rtnIndex = NotFound;
        boolean isPat = false;
        int subjectLen = subject.length;
        int patternLen = pattern.length;

        while(isPat == false && iSub + patternLen-1 < subjectLen ){
            if(subject[iSub]==pattern[0]){
                rtnIndex=iSub;
                isPat=true;
                for(int iPat = 1; iPat<patternLen; iPat++){
                    if(subject[iSub+iPat]!=pattern[iPat]){
                        rtnIndex = NotFound;
                        isPat = false;
                        break;
                    }
                }
            }
            iSub++;
        }
        return rtnIndex;
    }
}
```

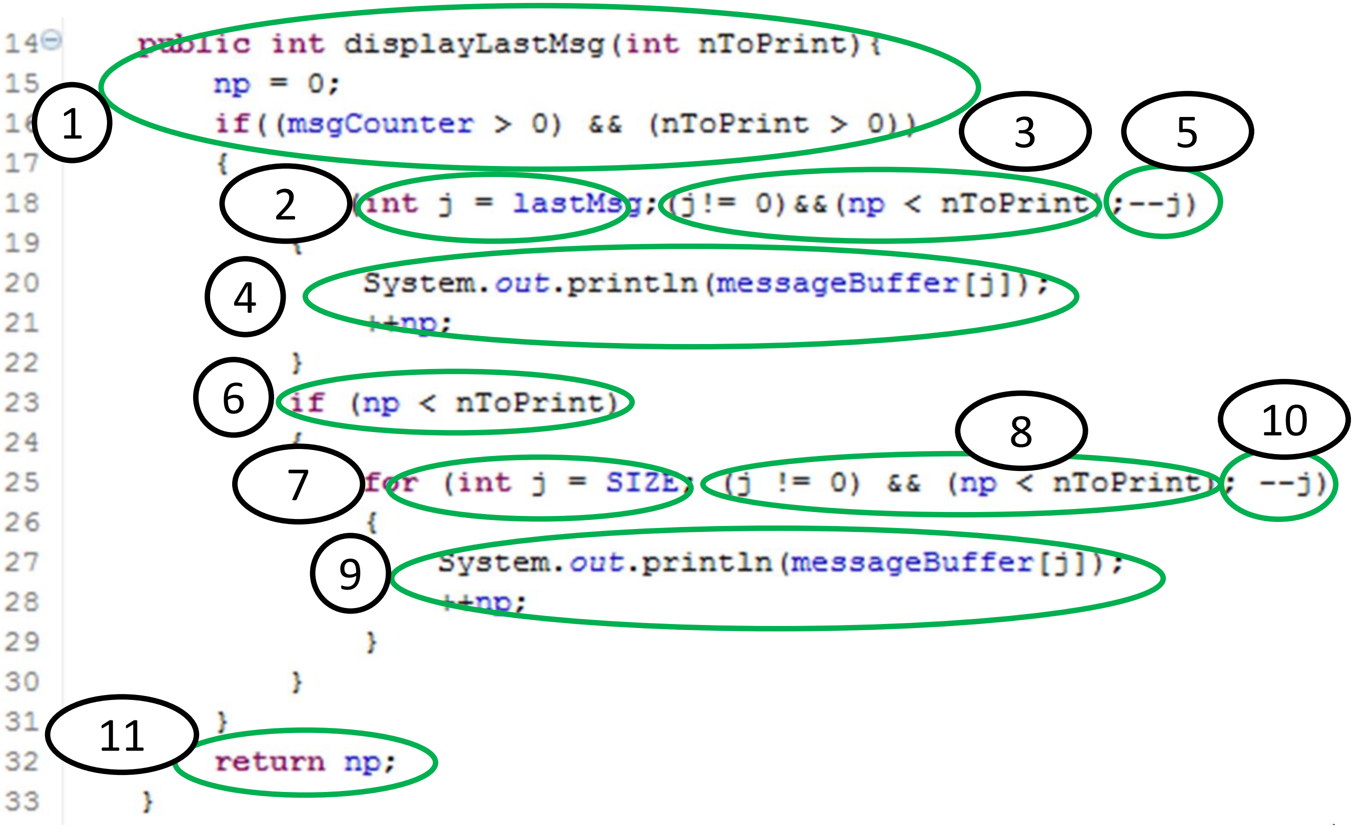
Exercise

- Draw CFG for *displayLastMsg*

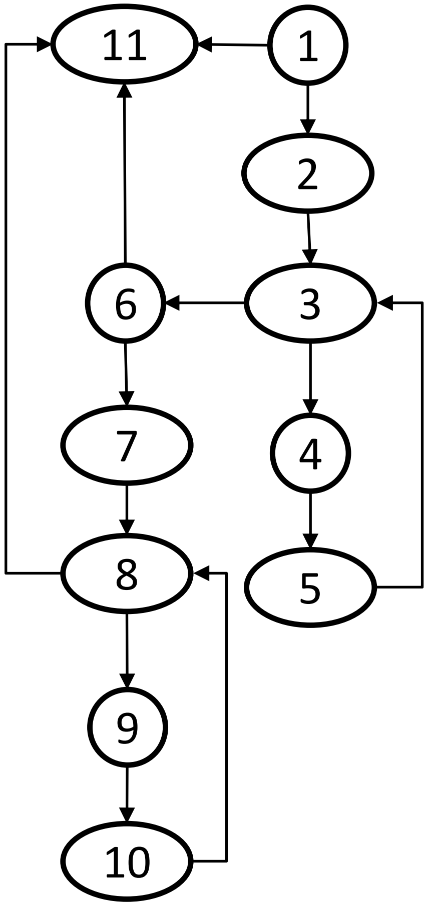


```java
public int displayLastMsg(int nToPrint){
    np = 0;
    if((msgCounter > 0) && (nToPrint > 0))
    {
        for(int j = lastMsg;(j!= 0)&&(np < nToPrint);--j)
        {
            System.out.println(messageBuffer[j]);
            ++np;
        }
        if (np < nToPrint)
        {
            for (int j = SIZE; (j != 0) && (np < nToPrint); --j)
            {
                System.out.println(messageBuffer[j]);
                ++np;
            }
        }
    }
    return np;
}
```

Exercise



```
public int displayLastMsg(int nToPrint){
      np = 0;
      if((msgCounter > 0) && (nToPrint > 0))
      {
           (int j = lastMsg;(j!= 0)&&(np < nToPrint);--j)
           {
                System.out.println(messageBuffer[j]);
                ++np;
           }
           if (np < nToPrint)
           {
                for (int j = SIZE; (j != 0) && (np < nToPrint); --j)
                {
                     System.out.println(messageBuffer[j]);
                     ++np;
                }
           }
      }
      return np;
}
```

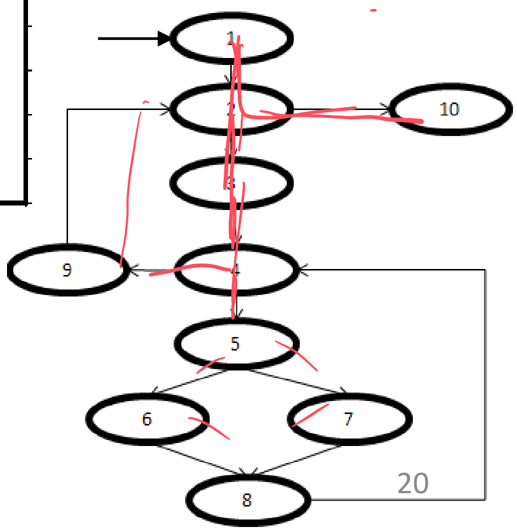


Paths in Graphs

- Path is a sequence of nodes: $(n_1, n_2, \ldots, n_m)$ where for $1 \leq i<m$, $(n_i,n_{i+1}) \epsilon E$
  - length: the number of edges contains in the path
  - The length of a path can be zero.
  - A subpath of a path p is a subsequence of p

Path Examples

长度	路径	长度	路径	长度	路径	长度	路径	长度	路径
0	1	1	(1,2)	2	(1,2,3)	......	......	n	(1,2,3,4,9,2,3,4,9,⋯...,2,10)
	2		(2,3)		(1,2,10)				......
	3		(2,10)		(2,3,4)				
	4		(3,4)		(3,4,5)				
	5		(4,5)		(3,4,9)				
	6		(4,9)		(4,5,6)				
	7		(5,6)		(4,5,7)				
	8		(5,7)		(4,9,2)				
	9		(6,8)		(5,6,8)				
	10		(7,8)		(5,7,8)				
			(8,4)		(6,8,4)				
			(9,2)		(7,8,4)				
					(8,4,5)				
					(8,4,9)				
					(9,2,3)				
					(9,2,10)				

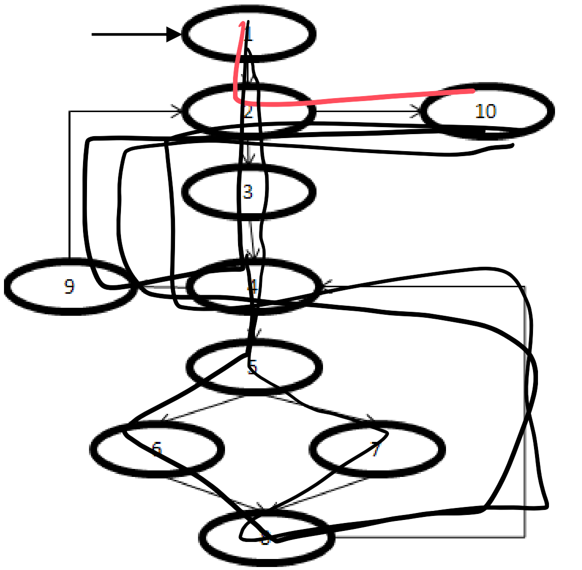


Complete Path

- Complete Path
  - A path that starts at an initial node and ends at a final node
  - Example
    1. (1,2,10)
    2. (1,2,3,4,9,2,10)
    3. (1,2,3,4,5,6,8,4,9,2,10)
    4. (1,2,3,4,5,7,8,4,9,2,10)

不可行的

  - Infeasible Complete Path
    - Can you give an code example which contains some infeasible path



Control Based Code Test Adequacy Criteria

- Control Flow Graph
- Statement Coverage
- Branch Coverage
- Path Coverage
- Prime Path Coverage
- Logical Coverage Criteria
  - Decision Coverage
  - Condition Coverage
  - Decision-Condition Coverage
  - Modified Condition/Decision Coverage (MC/DC)
  - Multiple Condition Coverage

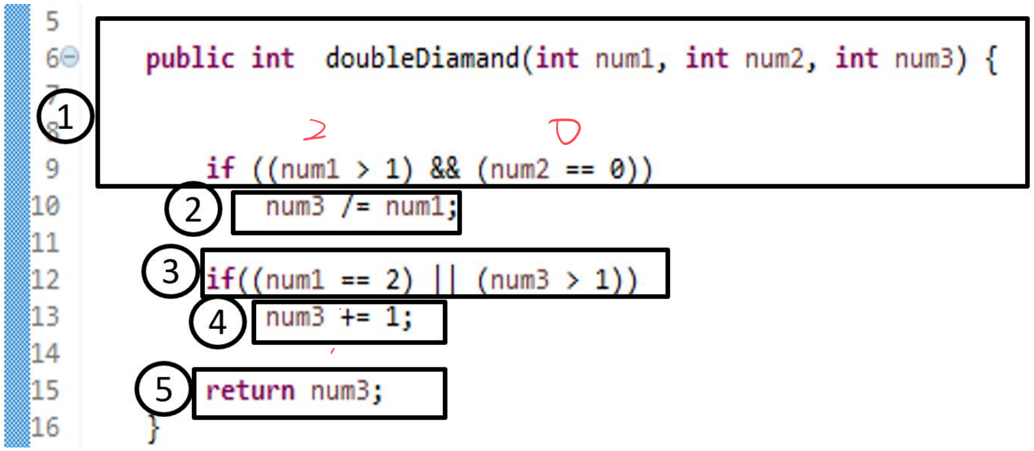
Statement Coverage

- 语句覆盖（Statement Coverage）
  - 衡量被测代码中的语句得到执行的程度。
  - 基于CFG块与语句的对应关系，衡量语句覆盖情况
  - 语句覆盖的正式定义为：测试集合T称为语句覆盖充分的，当且仅当执行T产生的完整路径集合L覆盖了控制流图中的所有节点。如果使用符号Node(G)表示控制流图的节点集合，Node（L）表示L包含的节点集合，则测试集合T的语句覆盖率为：

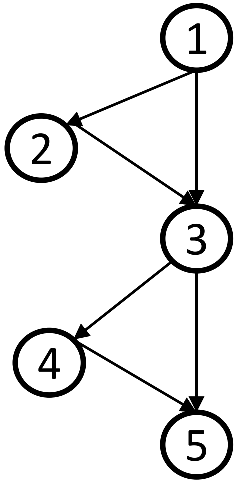
测试充分性问题

$$\frac{||\mathrm{Node(L)}||}{||Node(G)||} * 100\%$$

Statement Coverage



```
public int  doubleDiamand(int num1, int num2, int num3) {

    if ((num1 > 1) && (num2 == 0))
        num3 /= num1;
    if((num1 == 2) || (num3 > 1))
        num3 += 1;

    return num3;
}
```



- 测试集合1

  ① 测试用例1

  [(num1=2, num2=0, num3=4) , 3]

  - 测试集合产生的完整路径：1-2-3-4-5
  - 语句覆盖率 = 5/5 = 100%

- 测试集合2

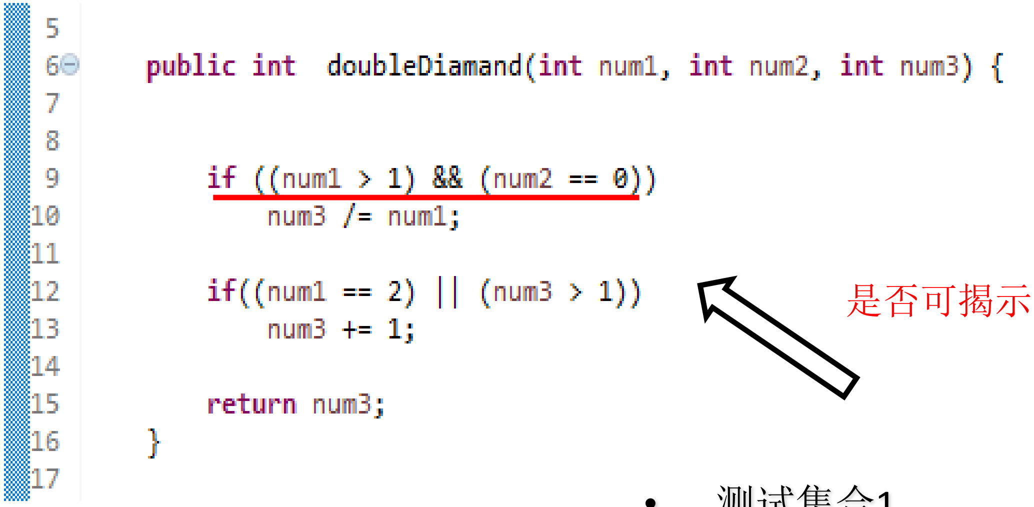
  ① 测试用例1

  [(num1=-2, num2=0, num3=4), 5]

  - 测试集合产生的完整路径：1-3-4-5
  - 语句覆盖率 = 4/5 = 80%

Statement Coverage

- 语句覆盖是最弱的标准



```
public int  doubleDiamand(int num1, int num2, int num3) {


    if ((num1 > 1) && (num2 == 0))
        num3 /= num1;

    if((num1 == 2) || (num3 > 1))
        num3 += 1;

    return num3;
}
```

是否可揭示&&”错写成“||”？？？ X

- 测试集合1

  ① 测试用例1

  [(num1=2, num2=0, num3=4) , 3]

  - 语句覆盖度 = 5/5= 100%

Control Based Code Test Adequacy Criteria

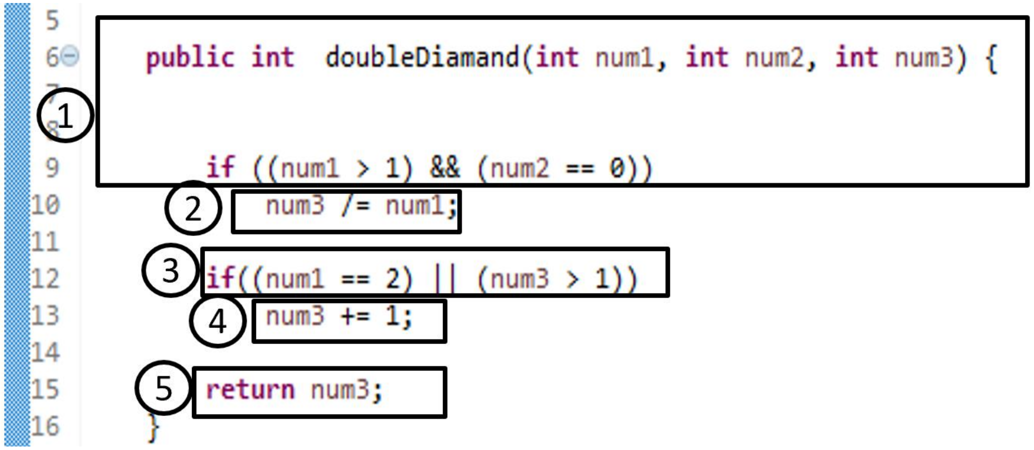
- Control Flow Graph
- Statement Coverage
- Branch Coverage
- Path Coverage
- Prime Path Coverage
- Logical Coverage Criteria
  - Decision Coverage
  - Condition Coverage
  - Decision-Condition Coverage
  - Modified Condition/Decision Coverage (MC/DC)
  - Multiple Condition Coverage

Branch Coverage

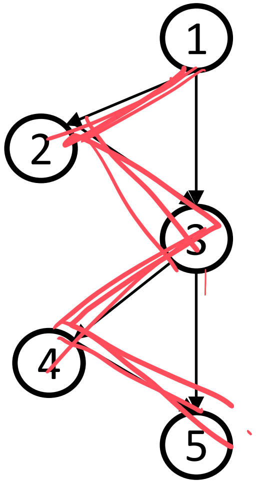
- 分支覆盖（Branch Coverage）
  - 衡量被代码中的所有控制转移被执行的程度。
  - 控制转移表现为CFG的边，控制转移得到测试意味着相应的边在测试集合对应的完整路径中出现
  - 分支覆盖准则的定义：测试集合T称为分支覆盖充分的，当且仅当执行T产生的完整路径集合L覆盖了控制流图中的所有边。如果使用符号Edge(G)表示控制流图的边集合，Edge（L）表示L包含的控制流图中的边集合，则测试集合T的分支覆盖率为：

$$\frac{||Edge(\mathrm{L})||}{||Edge(G)||} * 100\%$$

Branch Coverage



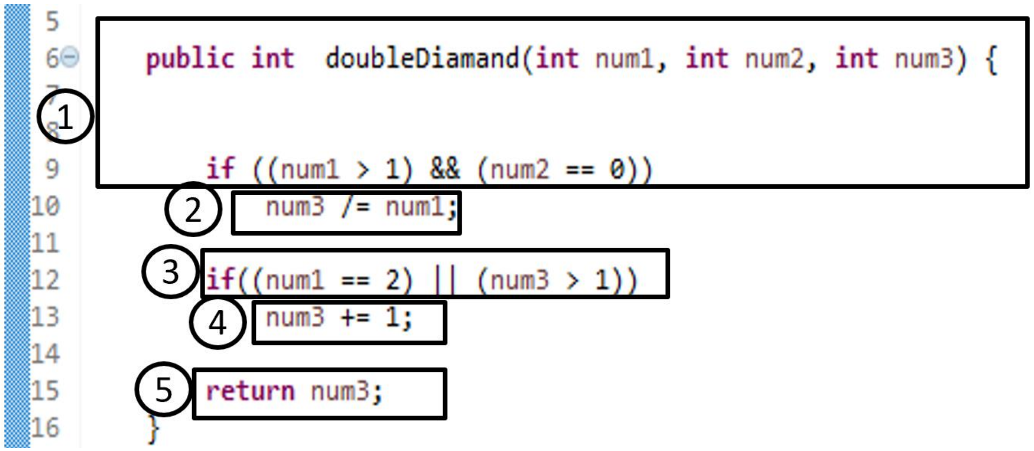
```
public int  doubleDiamand(int num1, int num2, int num3) {


    if ((num1 > 1) && (num2 == 0))
        num3 /= num1;

    if((num1 == 2) || (num3 > 1))
        num3 += 1;

    return num3;
}
```



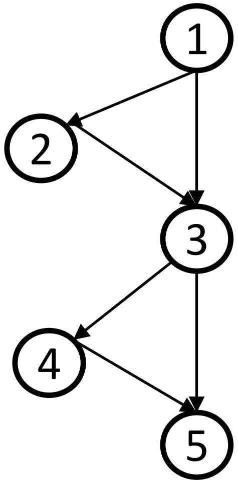
- 测试集合1
  ① 测试用例1: [(num1=2, num2=0, num3=4) , 3]
  - 测试集合产生的完整路径：1-2-3-4-5
  - 其覆盖的边集为：(1,2),(2,3),(3,4),(4,5)
  - 控制流图中的边集为： (1,2),(1,3),(2,3),(3,4),(3,5),(4,5)
  - 分支覆盖率 = 4/6 = 66.7%

$\frac{4}{6}$

Branch Coverage



```
public int  doubleDiamand(int num1, int num2, int num3) {


    if ((num1 > 1) && (num2 == 0))
        num3 /= num1;

    if((num1 == 2) || (num3 > 1))
        num3 += 1;

    return num3;
}
```



- 测试集合2

  ① 测试用例1: [(num1=2, num2=0, num3=4) , 3]

  ② 测试用例2：[(num1=3, num2=1, num3=1) , 1]

  - 测试集合产生的完整路径：1-2-3-4-5 1-3-5
  - 其覆盖的边集为：(1,2),(2,3),(3,4),(4,5) (1,3),(1,5)
  - 控制流图中的边集为： (1,2),(1,3),(2,3),(3,4),(3,5),(4,5)
  - 分支覆盖率 = 6/6 = 100%

prime Path coverage

基路径覆盖

Control Based Code Test Adequacy Criteria

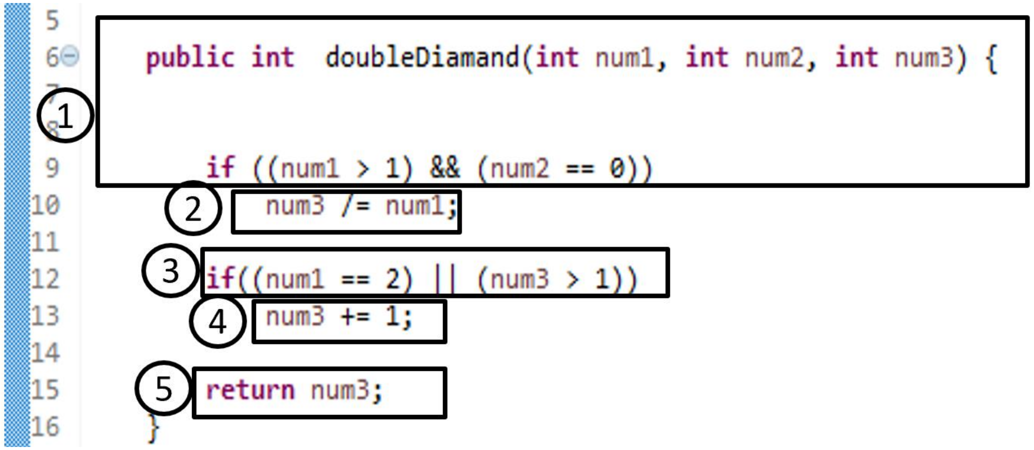
- Control Flow Graph
- Statement Coverage
- Branch Coverage
- Path Coverage
- Prime Path Coverage
- Logical Coverage Criteria
  - Decision Coverage
  - Condition Coverage
  - Decision-Condition Coverage
  - Modified Condition/Decision Coverage (MC/DC)
  - Multiple Condition Coverage

Path Coverage

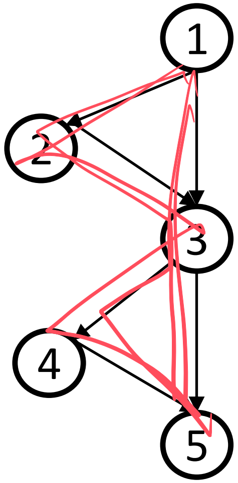
- 路径覆盖（Path Coverage）
  - 衡量被代码中的完整路径被执行的程度
  - 被代码中的完整路径对应于CFG中的完整路径
  - 测试集合T称为路径覆盖充分的，当且仅当执行T产生的完整路径集合L覆盖了控制流图中的所有完整路径。如果使用符号Path(G)表示控制流图的所有完整路径集合，则测试集合T的路径覆盖率为：

$$\frac{||L||}{||Path(G)||} * 100\%$$

Path Coverage



```
public int  doubleDiamand(int num1, int num2, int num3) {


    if ((num1 > 1) && (num2 == 0))
        num3 /= num1;

    if((num1 == 2) || (num3 > 1))
        num3 += 1;

    return num3;
}
```



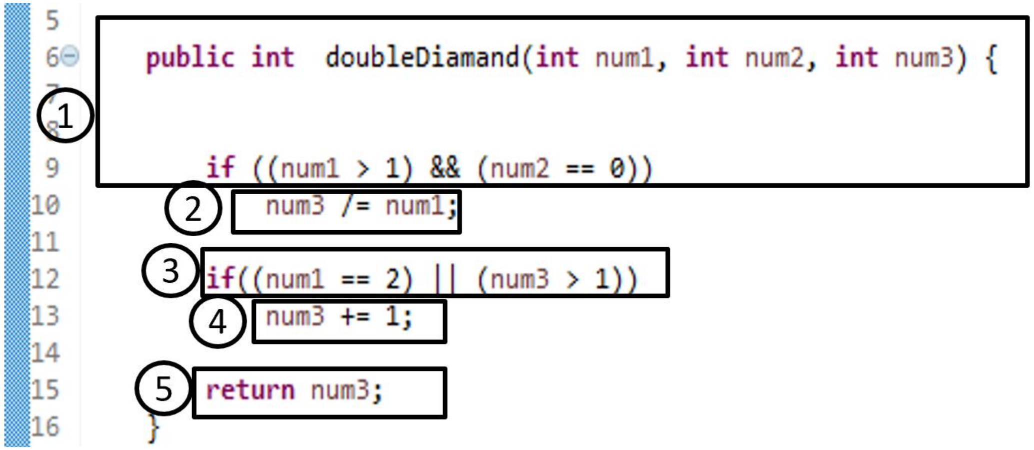
$\frac{1}{4}$=25%

- 测试集合1

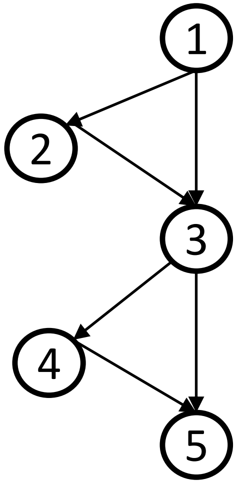
  ① 测试用例1: [(num1=2, num2=0, num3=4) , 3]

  - 测试集合产生的完整路径：1-2-3-4-5
  - 控制流图中的完整路径集为：1-2-3-4-5，1-2-3-5，1-3-4-5，1-3-5
  - 路径覆盖率 = 1/4 = 25%

Path Coverage



```
public int  doubleDiamand(int num1, int num2, int num3) {


    if ((num1 > 1) && (num2 == 0))
        num3 /= num1;

    if((num1 == 2) || (num3 > 1))
        num3 += 1;

    return num3;
}
```



- 测试集合2

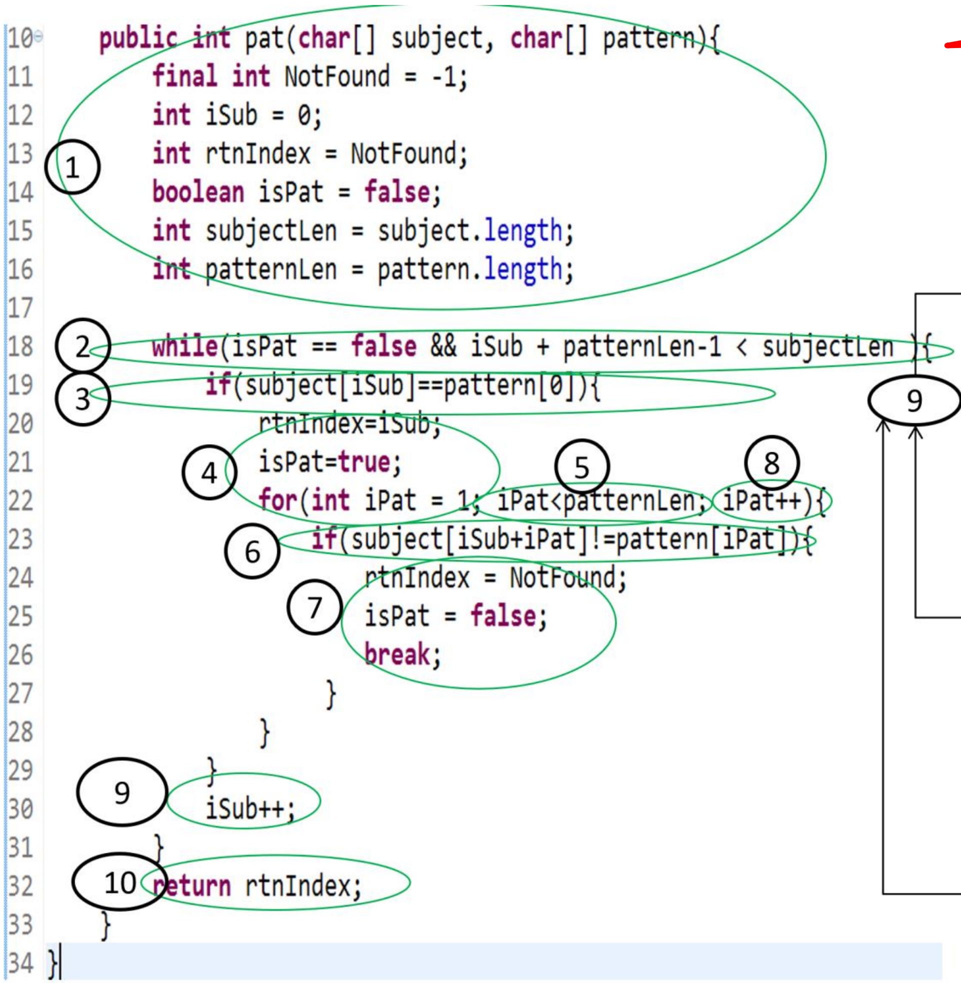
  ① 测试用例1: [(num1=2, num2=0, num3=4) , 3]

  ② 测试用例2：[(num1=3, num2=1, num3=1) , 1]

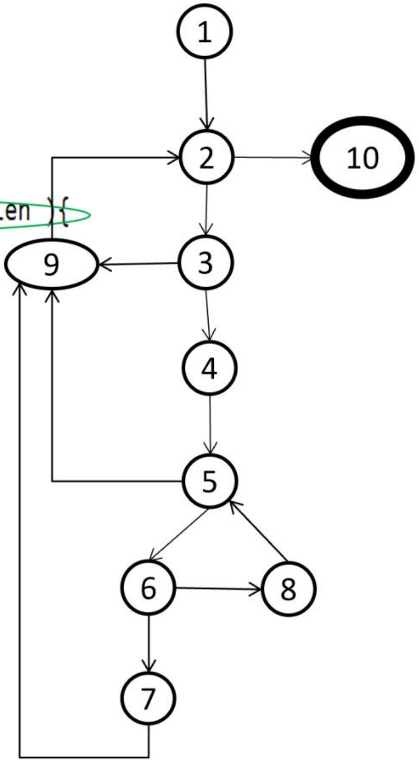
  - 测试集合产生的完整路径：1-2-3-4-5，1-3-5
  - 控制流图中的完整路径集为： 1-2-3-4-5，1-2-3-5，1-3-4-5，1-3-5
  - 路径覆盖率 = 2/4 = 50%

Path Coverage

多少条路径？



```
    public int pat(char[] subject, char[] pattern){
        final int NotFound = -1;
        int iSub = 0;
        int rtnIndex = NotFound;
        boolean isPat = false;
        int subjectLen = subject.length;
        int patternLen = pattern.length;

        while(isPat == false && iSub + patternLen-1 < subjectLen ){
            if(subject[iSub]==pattern[0]){
                rtnIndex=iSub;
                isPat=true;
                for(int iPat = 1; iPat<patternLen; iPat++){
                    if(subject[iSub+iPat]!=pattern[iPat]){
                        rtnIndex = NotFound;
                        isPat = false;
                        break;
                    }
                }
            }
            iSub++;
        }
        return rtnIndex;
    }
}
```



Control Based Code Test Adequacy Criteria

- Control Flow Graph
- Statement Coverage
- Branch Coverage
- Path Coverage
- Prime Path Coverage
- Logical Coverage Criteria
  - Decision Coverage
  - Condition Coverage
  - Decision-Condition Coverage
  - Modified Condition/Decision Coverage (MC/DC)
  - Multiple Condition Coverage

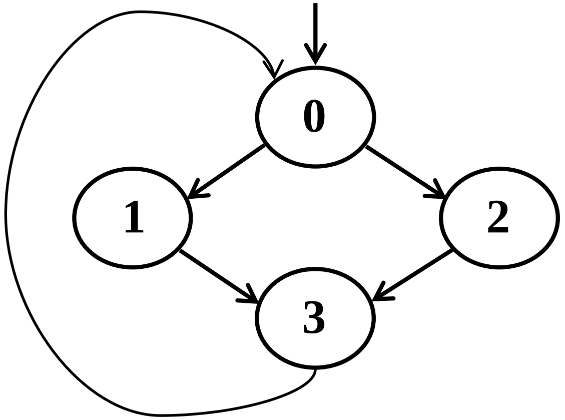
基路径覆盖

Prime Path Coverage

- Reduce must testing paths
- Simple Path
  - A path from node n_i to n_j is simple if no node appears more than once, except possibly the first and last nodes are the same
    - No internal loops

Simple Paths 简单路径

- Simple Path Example
  - length = 0: [0], [1], [2], [3]
  - length = 1: [0, 1], [0, 2], [1, 3], [2, 3], [3, 0]
  - length = 2: [0, 1, 3], [0, 2, 3], [1, 3, 0], [2, 3, 0],[3, 0, 1], [3, 0, 2]
  - length = 3: [0, 1, 3, 0], [0, 2, 3, 0], [1, 3, 0, 1], [2, 3, 0, 2], [3, 0, 1, 3], [3, 0, 2, 3], [1, 3, 0, 2], [2, 3, 0, 1]

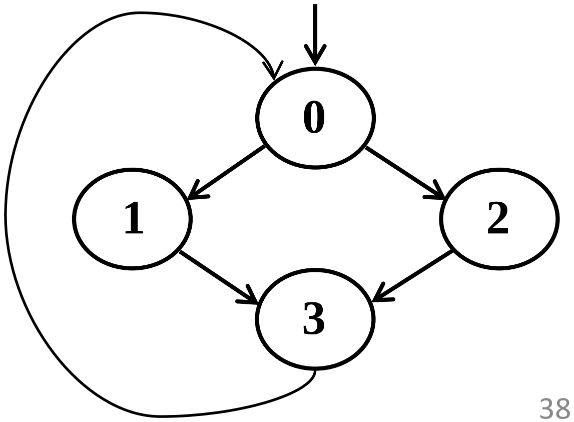


Prime Path

- Prime Path
  - A simple path that does not appear as a proper subpath of any other simple path

[0, 1, 3, 0], [0, 2, 3, 0], [1, 3, 0, 1], [2, 3, 0, 2],

[3, 0, 1, 3], [3, 0, 2, 3], [1, 3, 0, 2], [2, 3, 0, 1]



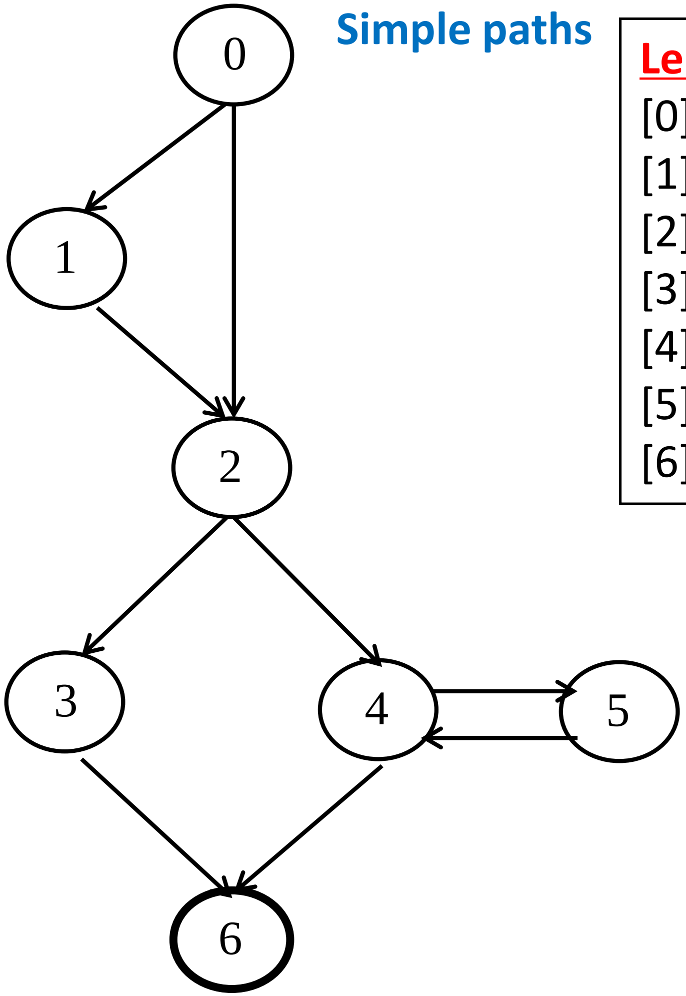
Calculate Prime Path

- How to calculate prime path
  1. Exhaust method 暴力穷举
  2. Node tree method 节点树

The Exhaust Method

```
Input:  CFG G=(N, $N_0$, $N_f$ , E)   Output:  The Prime Path Set of G
begin
    simplePathSet = N  // begin with the paths of length 0
    extentablePathSet= simplePathSet
    primePathSet = Φ
    while (extentablePathSet!= Φ)
            p = extentablePathSet. getOnePath()
            if   $N_F \cap$ p  != Φ  then
                extentablePathSet = extentablePathSet – p
            else
                p' = addPathLengthWithOne ( p , E)
                if isSimplePath(p')  then
                    extentablePathSet = extentablePathSet ∪ p'
                    simplePathSet = simplePathSet ∪ p'
                else
                    extentablePathSet = extentablePathSet – p
                endIf
            endIf
    endwhile
      primePathSet  =  simplePathSet
      for each pair of paths $p_i$, $p_j$  in primePathSet
          if isSubpath($p_i$, $p_j$)  then primePathSet  = primePathSet -{$p_i$}
      endfor
end
```

Calculate Prime Paths



Simple paths

Len 0

[0]
[1]
[2]
[3]
[4]
[5]
[6] !

Len 1

[0, 1]
[0, 2]
[1, 2]
[2, 3]
[2, 4]
[3, 6] !
[4, 6] !
[4, 5]
[5, 4]

Len 2

[0, 1, 2]
[0, 2, 3]
[0, 2, 4]
[1, 2, 3]
[1, 2, 4]
[2, 3, 6] !
[2, 4, 6] !
[2, 4, 5] !
[4, 5, 4] *
[5, 4, 6] !
[5, 4, 5] *

Len 3

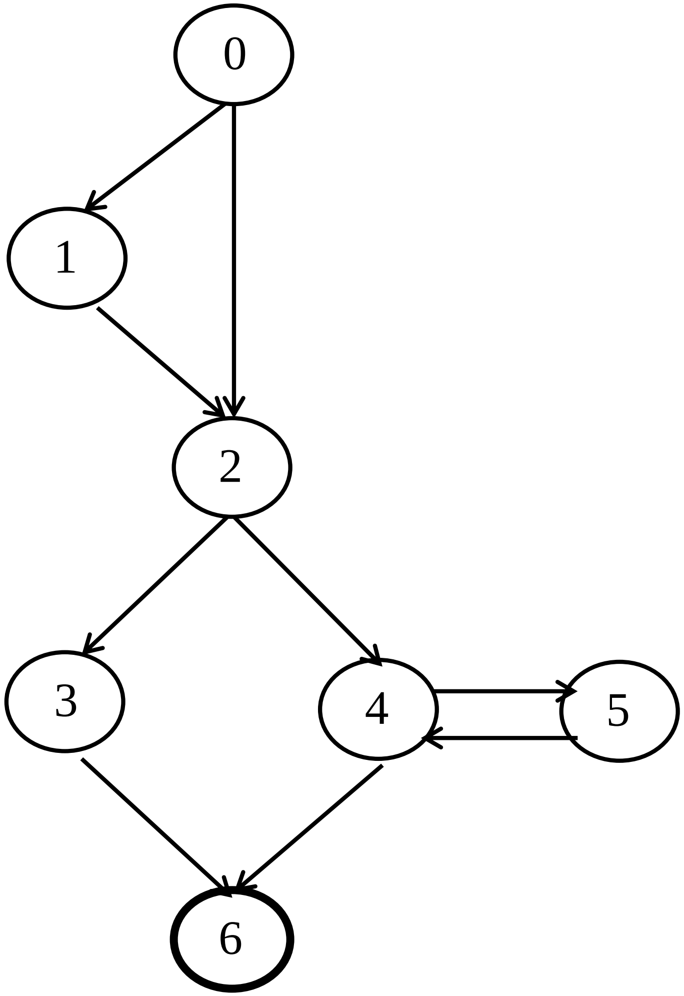
[0, 1, 2, 3]
[0, 1, 2, 4]
[0, 2, 3, 6] !
[0, 2, 4, 6] !
[0, 2, 4, 5] !
[1, 2, 3, 6] !
[1, 2, 4, 5] !
[1, 2, 4, 6] !

Len 4

[0, 1, 2, 3, 6] !
[0, 1, 2, 4, 6] !
[0, 1, 2, 4, 5] !

Prime Paths

Prime Path Example



Prime Paths

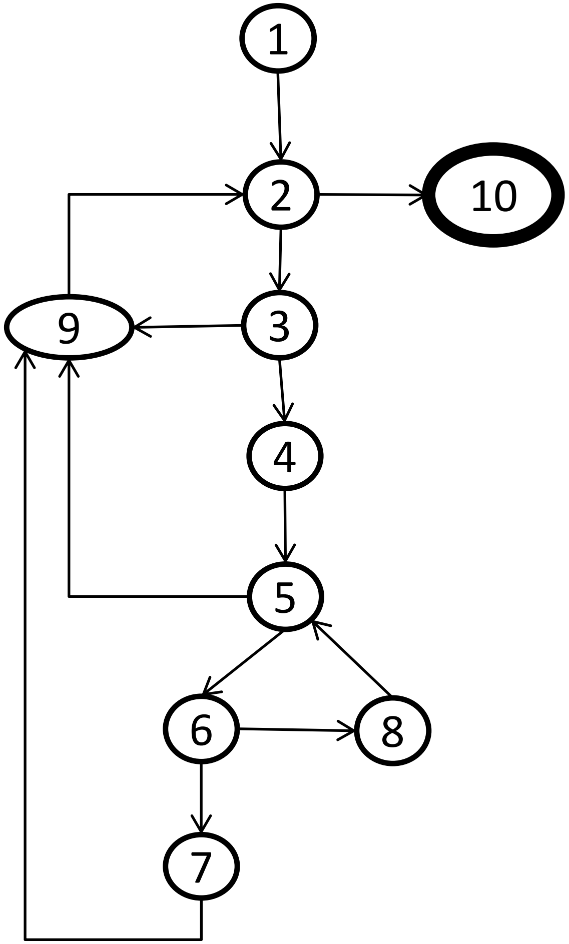
[0, 1, 2, 3, 6]
[0, 1, 2, 4, 5]
[0, 1, 2, 4, 6]
[0, 2, 3, 6]
[0, 2, 4, 5]
[0, 2, 4, 6]
[5, 4, 6]
[4, 5, 4]
[5, 4, 5]

Execute loop 0 times

Execute loop once

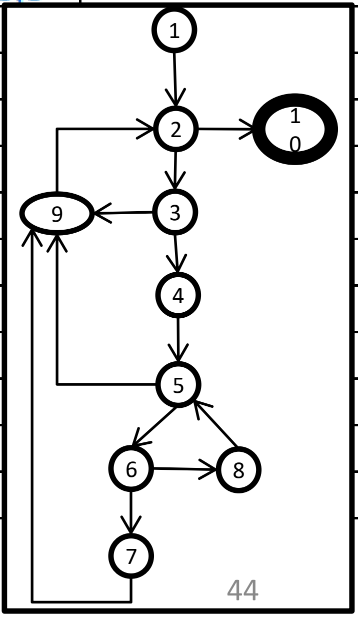
Execute loop more than once

Example



- Calculate prime path set for pat() method

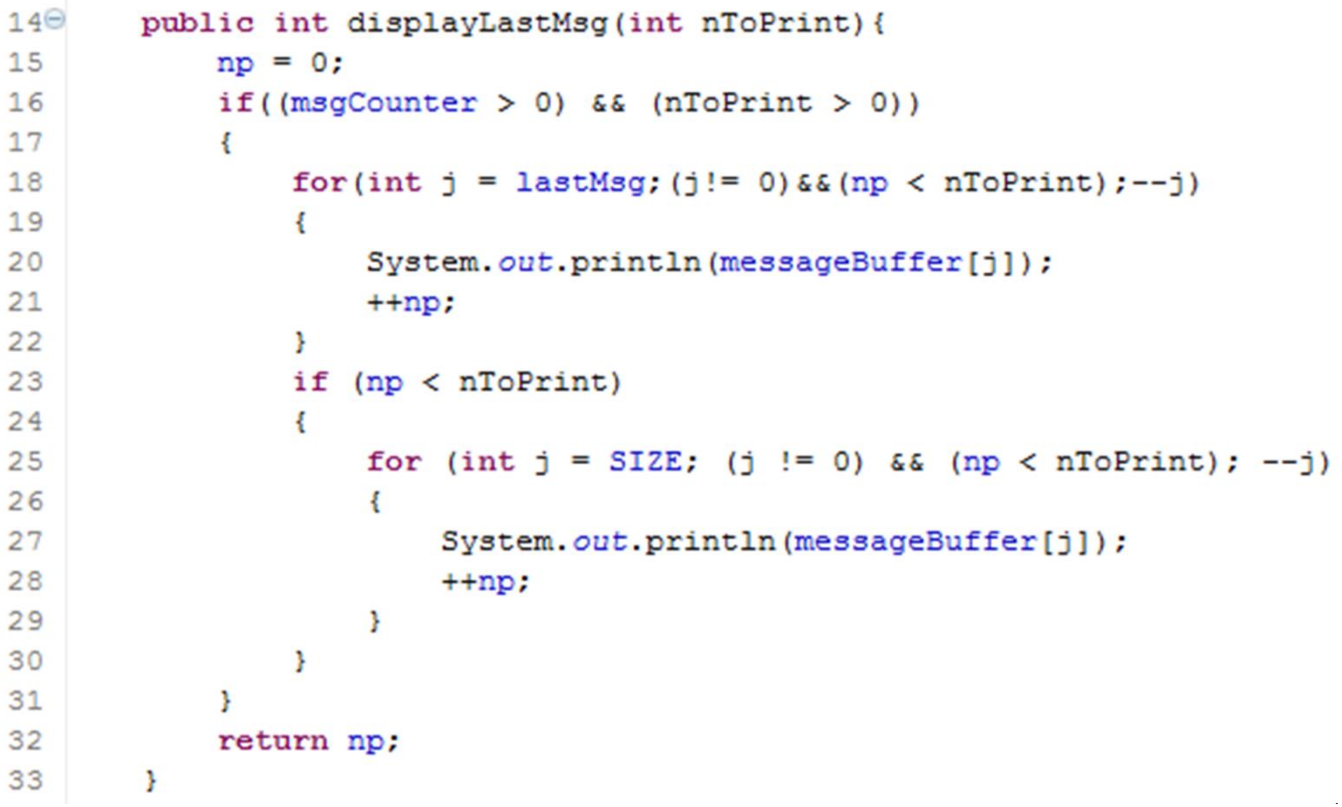
length = 0	length = 1	length = 2	length = 3	length = 4	length = 5	length = 6	length = 7
1	1,2	**1,2,10**	1,2,3,4	1,2,3,4,5	**1,2,3,4,5,9**	1,2,3,4,5,6,7	**1,2,3,4,5,6,7,9**
2	2,3	1,2,3	**1,2,3,9**	2,3,4,5,6	1,2,3,4,5,6	**1,2,3,4,5,6,8**	**2,3,4,5,6,7,9,2**
3	**2,10**	2,3,4	2,3,4,5	2,3,4,5,9	2,3,4,5,6,7	2,3,4,5,6,7,9	**3,4,5,6,7,9,2,3**
4	3,4	2,3,9	**2,3,9,2**	3,4,5,6,7	**2,3,4,5,6,8**	3,4,5,6,7,9,2	**3,4,5,6,7,9,2,10**
5	3,9	3,4,5	3,4,5,6	**3,4,5,6,8**	**2,3,4,5,9,2**	**4,5,6,7,9,2,10**	**4,5,6,7,9,2,3,4**
6	4,5	3,9,2	3,4,5,9	3,4,5,9,2	3,4,5,6,7,9	4,5,6,7,9,2,3	**5,6,7,9,2,3,4,5**
7	5,6	4,5,6	**3,9,2,10**	4,5,6,7,9	**3,4,5,9,2,10**	5,6,7,9,2,3,4	**6,7,9,2,3,4,5,6**
8	5,9	4,5,9	**3,9,2,3**	4,5,9,2,3	**3,4,5,9,2,3**	6,7,9,2,3,4,5	**7,9,2,3,4,5,6,7**
9	6,7	5,6,7	4,5,6,7	**4,5,9,2,10**	4,5,6,7,9,2	**6,8,5,9,2,3,4**	**7,9,2,3,4,5,6,8**
10	6,8	5,6,8	**4,5,6,8**	5,9,2,3,4	**4,5,9,2,3,4**	7,9,2,3,4,5,6	**8,5,6,7,9,2,3,4**
	7,9	5,9,2	4,5,9,2	5,6,7,9,2	**5,9,2,3,4,5**	8,5,6,7,9,2,3	**9,2,3,4,5,6,7,9**
	8,5	6,7,9	5,6,7,9	**6,7,9,2,10**	5,6,7,9,2,3	**8,5,6,7,9,2,10**	
	9,2	6,8,5	**5,6,8,5**	6,7,9,2,3	**5,6,7,9,2,3,10**	9,2,3,4,5,6,7	
		7,9,2	**5,9,2,10**	6,8,5,9,2	6,7,9,2,3,4	**9,2,3,4,5,6,8**	
		8,5,9	5,9,2,3	7,9,2,3,4	6,8,5,9,2,3		
		8,5,6	6,7,9,2	8,5,9,2,3	**6,8,5,9,2,10**		
		9,2,10	**6,8,5,6**	**8,5,9,2,10**	7,9,2,3,4,5		
		9,2,3	6,8,5,9	8,5,6,7,9	**8,5,9,2,3,4**		
			7,9,2,3	9,2,3,4,5	8,5,6,7,9,2		
			7,9,2,10		**9,2,3,4,5,9**		
			8,5,9,2		9,2,3,4,5,6		
			8,5,6,7				
			8,5,6,8				
			9,2,3,4				
			9,2,3,9				



基路径集合为图中蓝色路径购成

Exercise

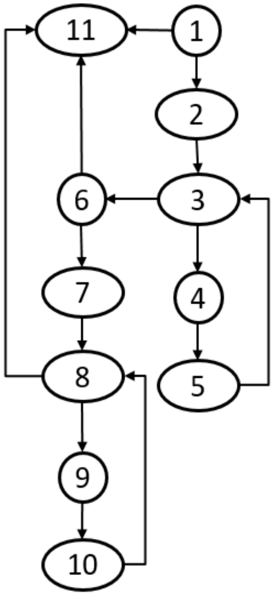
- Calculate Prime Paths *displayLastMsg* based on your CFG

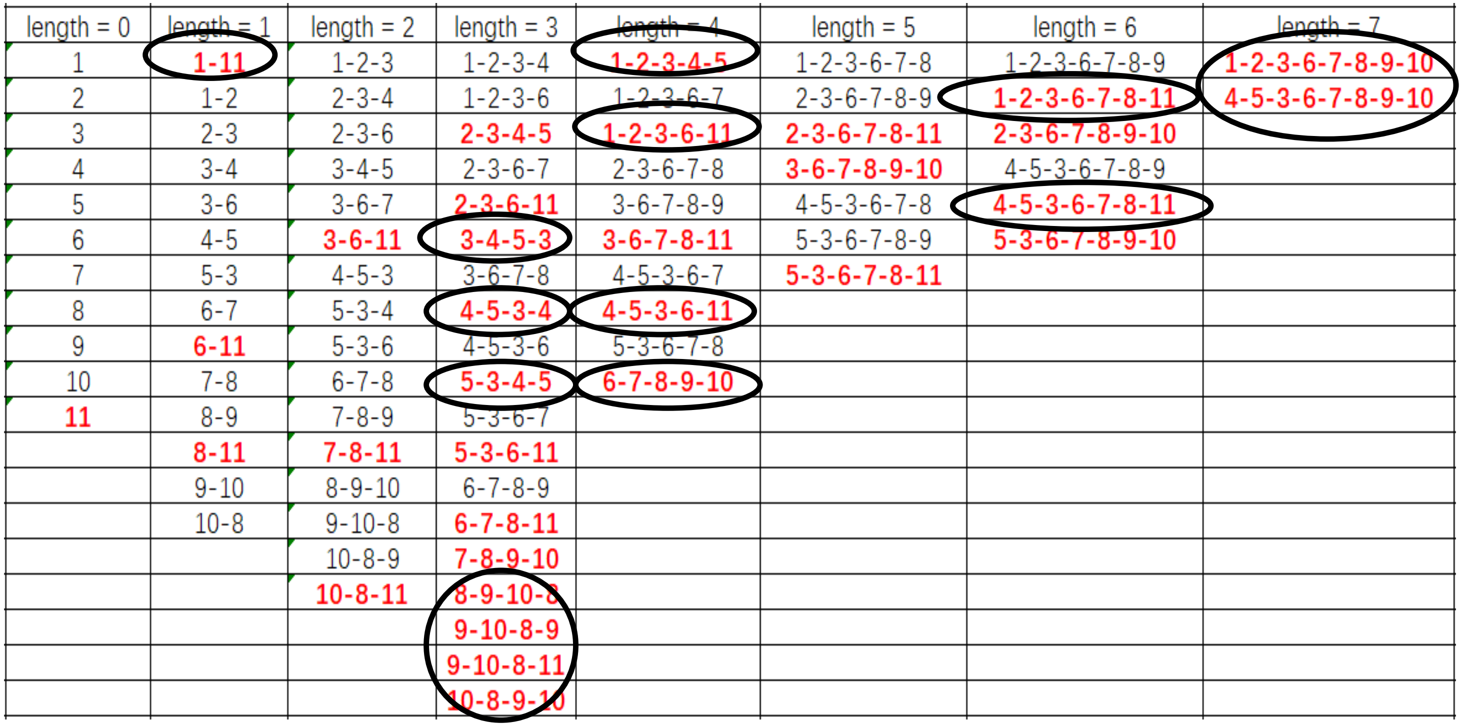


```
public int displayLastMsg(int nToPrint){
    np = 0;
    if((msgCounter > 0) && (nToPrint > 0))
    {
        for(int j = lastMsg;(j!= 0)&&(np < nToPrint);--j)
        {
            System.out.println(messageBuffer[j]);
            ++np;
        }
        if (np < nToPrint)
        {
            for (int j = SIZE; (j != 0) && (np < nToPrint); --j)
            {
                System.out.println(messageBuffer[j]);
                ++np;
            }
        }
    }
    return np;
}
```

Exercise

- Calculate Prime Paths *displayLastMsg* based on your CFG





length = 0	length = 1	length = 2	length = 3	length = 4	length = 5	length = 6	length = 7
1	1-11	1-2-3	1-2-3-4	1-2-3-4-5	1-2-3-6-7-8	1-2-3-6-7-8-9	1-2-3-6-7-8-9-10
2	1-2	2-3-4	1-2-3-6	1-2-3-6-7	2-3-6-7-8-9	1-2-3-6-7-8-11	4-5-3-6-7-8-9-10
3	2-3	2-3-6	2-3-4-5	1-2-3-6-11	2-3-6-7-8-11	2-3-6-7-8-9-10	
4	3-4	3-4-5	2-3-6-7	2-3-6-7-8	3-6-7-8-9-10	4-5-3-6-7-8-9	
5	3-6	3-6-7	2-3-6-11	3-6-7-8-9	4-5-3-6-7-8	4-5-3-6-7-8-11	
6	4-5	3-6-11	3-4-5-3	3-6-7-8-11	5-3-6-7-8-9	5-3-6-7-8-9-10	
7	5-3	4-5-3	3-6-7-8	4-5-3-6-7	5-3-6-7-8-11		
8	6-7	5-3-4	4-5-3-4	4-5-3-6-11			
9	6-11	5-3-6	4-5-3-6	5-3-6-7-8			
10	7-8	6-7-8	5-3-4-5	6-7-8-9-10			
11	8-9	7-8-9	5-3-6-7				
	8-11	7-8-11	5-3-6-11				
	9-10	8-9-10	6-7-8-9				
	10-8	9-10-8	6-7-8-11				
		10-8-9	7-8-9-10				
		10-8-11	8-9-10-8				
			9-10-8-9				
			9-10-8-11				
			10-8-9-10				

1-11
1-2-3-6-7-8-9-10
1-2-3-6-7-8-11
1-2-3-4-5
1-2-3-6-11

3-4-5-3

4-5-3-4
4-5-3-6-11
4-5-3-6-7-8-11
4-5-3-6-7-8-9-10

5-3-4-5
8-9-10-8
9-10-8-9
9-10-8-11
10-8-9-10

Node Tree Method

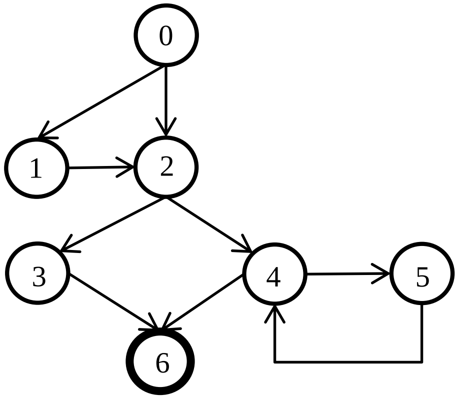
- New Algorithm:
  - DDL_PrimePathCal
- Verify Prime Path Results
  - Compare with two different kinds of algorithm

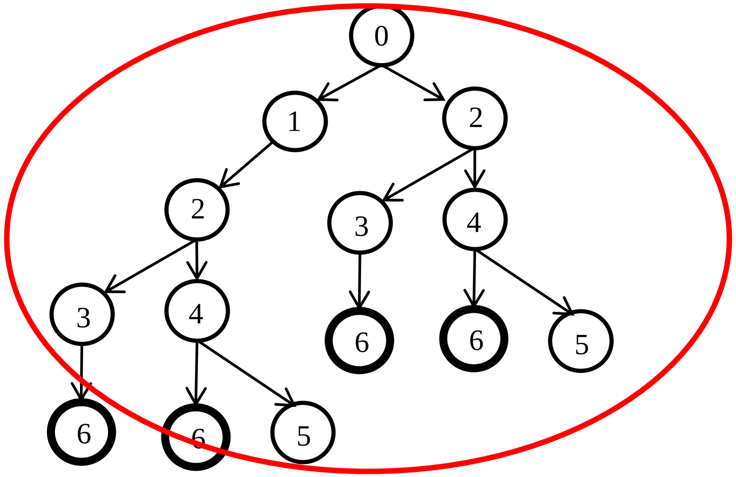
  1. Exhaust_PrimePathCal
  2. DDL_PrimePathCal

Node Tree Method

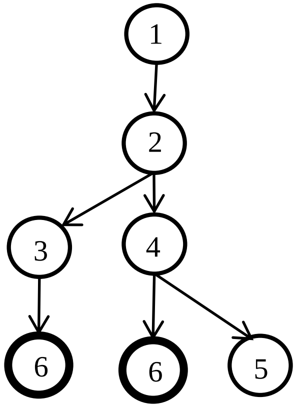
- 节点树
  - 以G中的节点为根节点建立的树，且满足树中除根节点和叶节点可以相同外，从根节点到每个树中节点的路径中，每个节点的出现次数有且仅有1次。
  - 在节点树中，每条从根节点到叶节点的路径即为一条简单路径。
- 简单节点树
  - 若节点树T不是任何其它节点树的子树，则称节点树T为简单节点树。
  - ~~所有~~简单节点树的从根节点到叶节点的路径集合为备选的基路径集合。

DDL_PrimePathCal Example

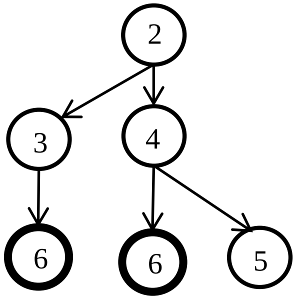




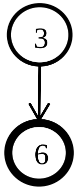
0节点的节点树



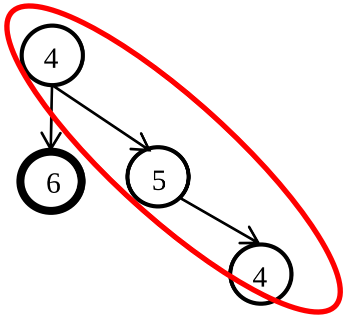
1节点的节点树



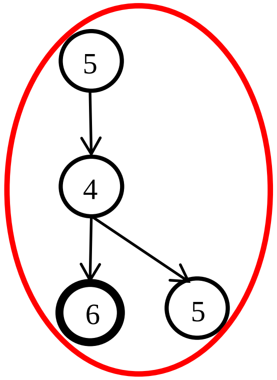
2节点的节点树



3节点的节点树

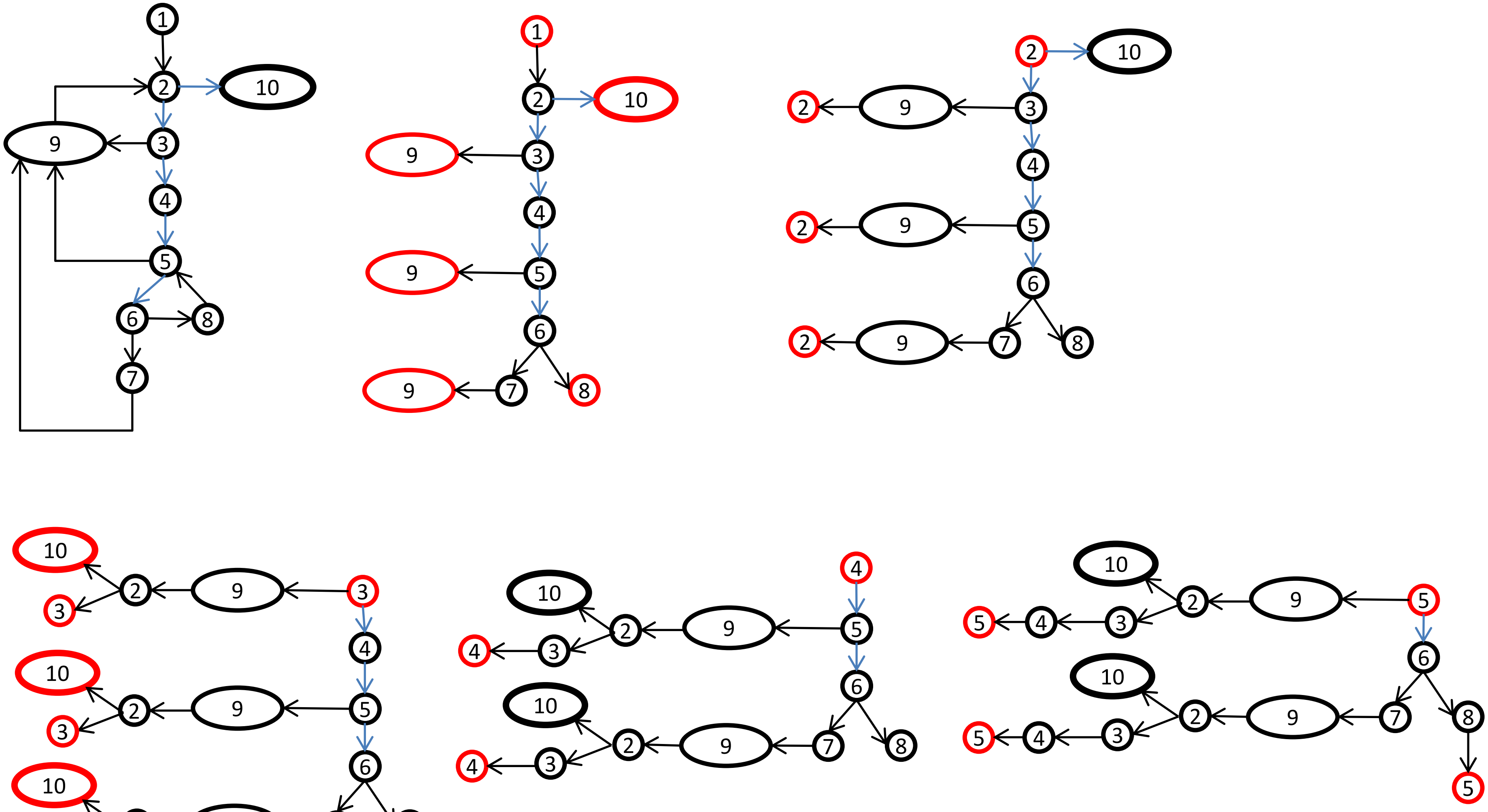


4节点的节点树



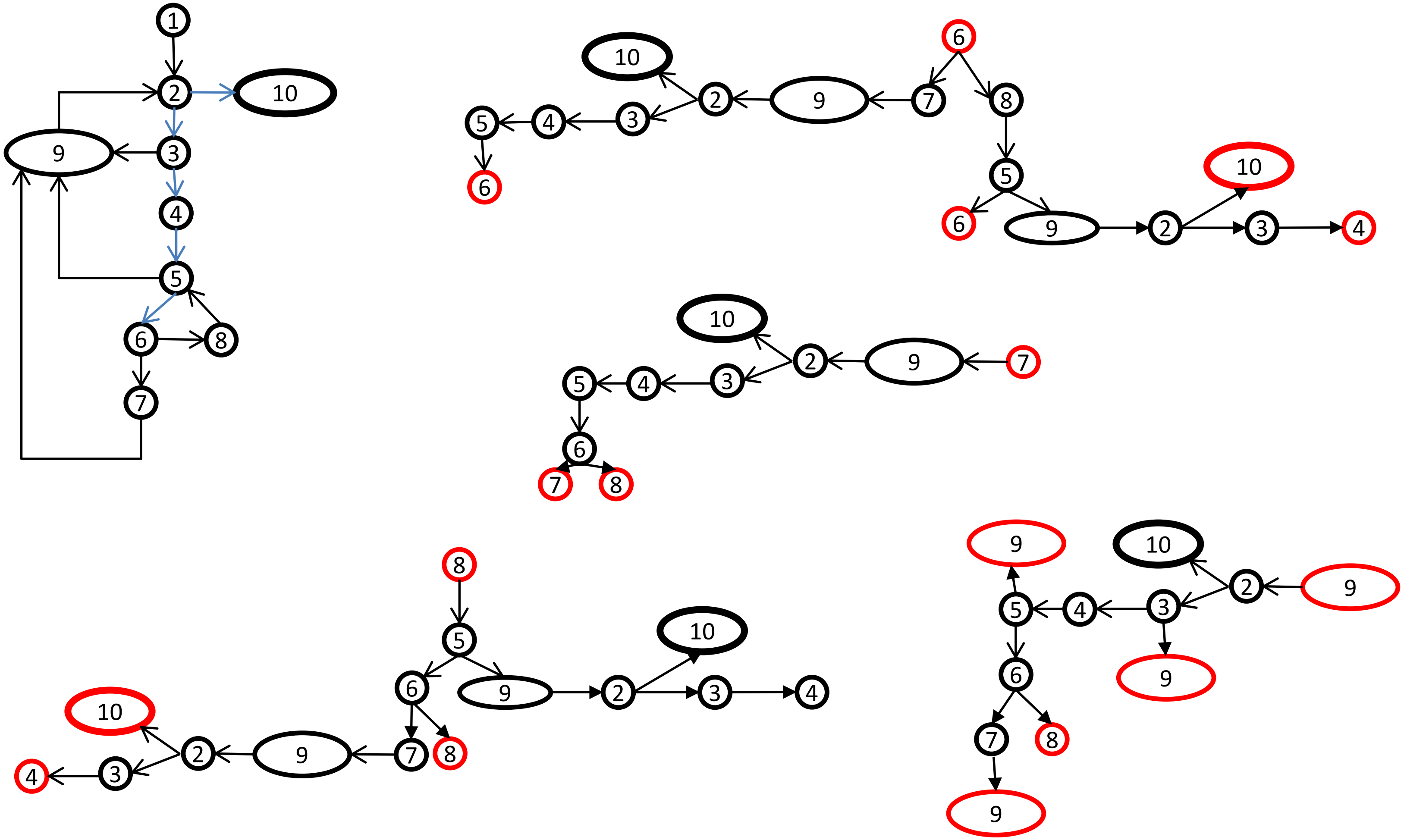
5节点的节点树

DDL_PrimePathCal Exercise



每条基路径由图中红色的起点和终点标注

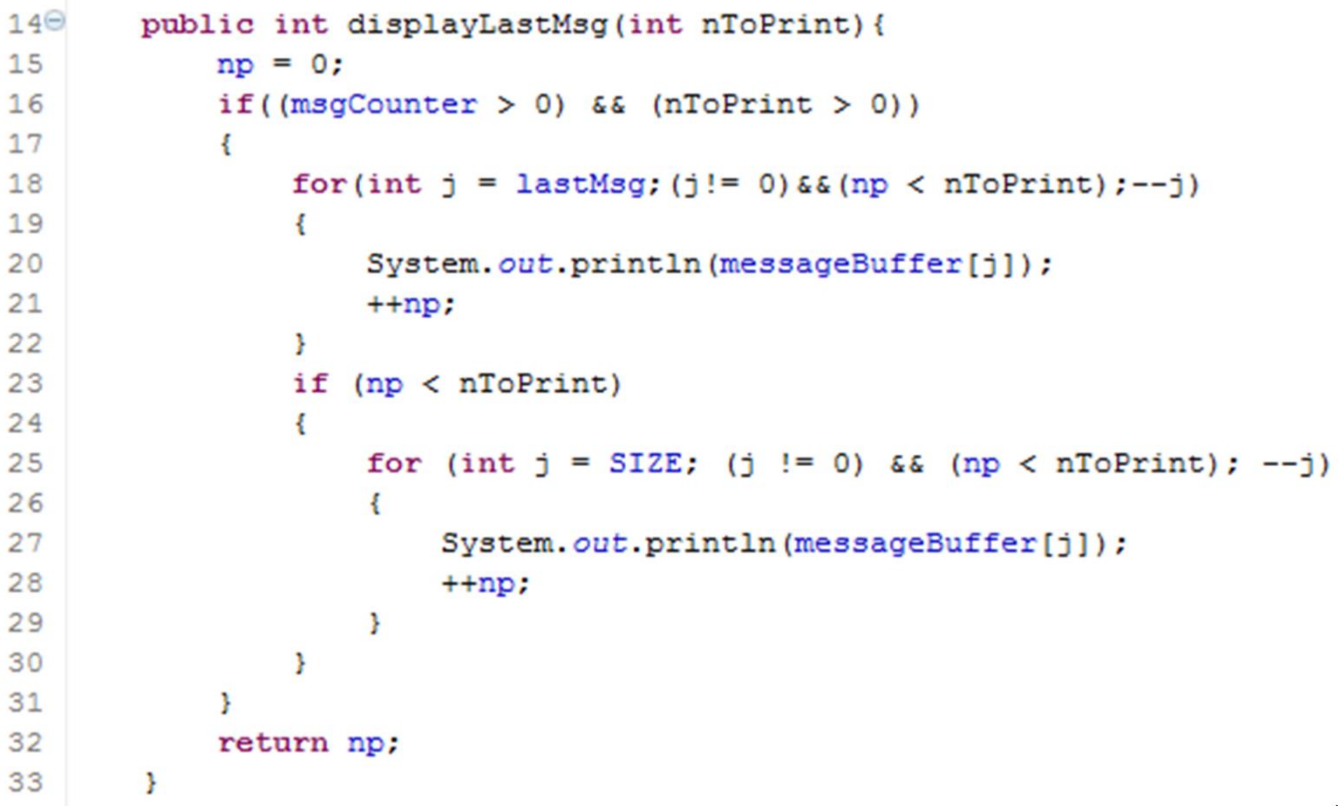
DDL_PrimePathCal Exercise



每条基路径由图中红色的起点和终点标注

Exercise

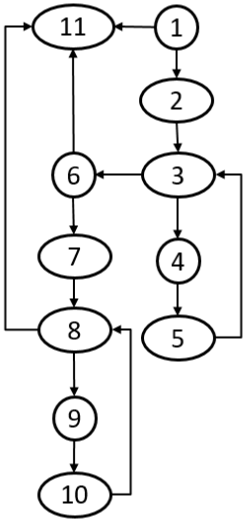
- Calculate Prime Paths *displayLastMsg* based on your CFG

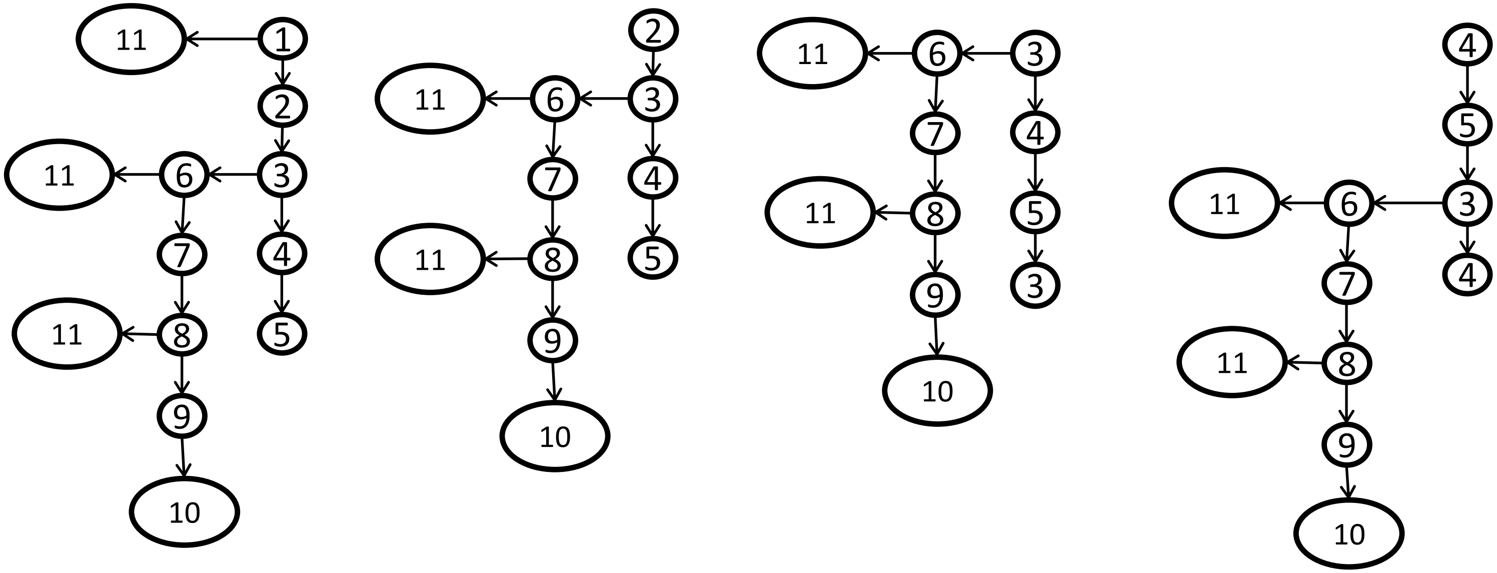


```java
public int displayLastMsg(int nToPrint){
    np = 0;
    if((msgCounter > 0) && (nToPrint > 0))
    {
        for(int j = lastMsg;(j!= 0)&&(np < nToPrint);--j)
        {
            System.out.println(messageBuffer[j]);
            ++np;
        }
        if (np < nToPrint)
        {
            for (int j = SIZE; (j != 0) && (np < nToPrint); --j)
            {
                System.out.println(messageBuffer[j]);
                ++np;
            }
        }
    }
    return np;
}
```

Exercise

- Calculate Prime Paths *displayLastMsg* based on your CFG





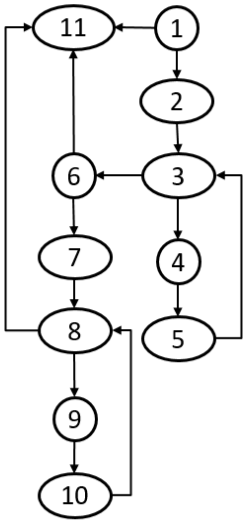
1-11
1-2-3-4-5
1-2-3-6-7-8-9-10
1-2-3-6-11
1-2-3-6-7-8-11

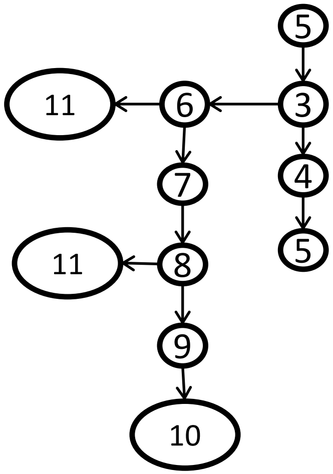
3-4-5-3

4-5-3-4
4-5-3-6-11
4-5-3-6-7-8-11
4-5-3-6-7-8-9-10

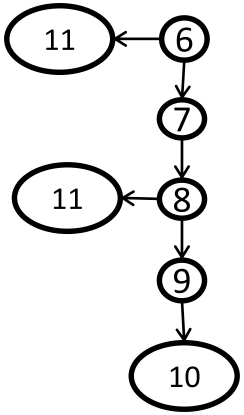
Exercise

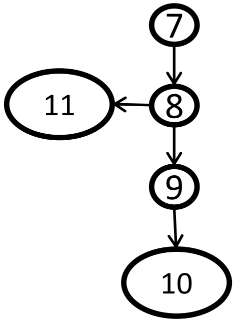
- Calculate Prime Paths *displayLastMsg* based on your CFG

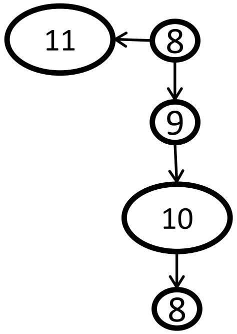




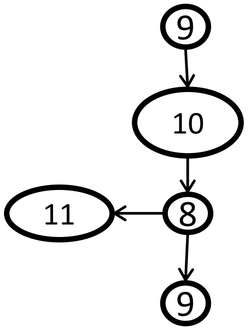
5-3-4-5



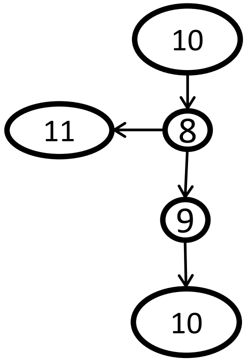




8-9-10-8



9-10-8-9
9-10-8-11



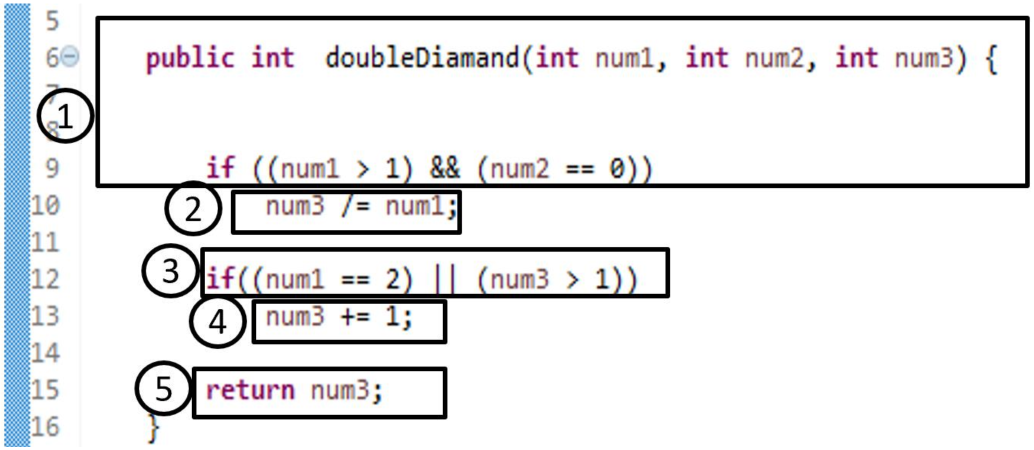
10-8-9-10

Prime Path Coverage

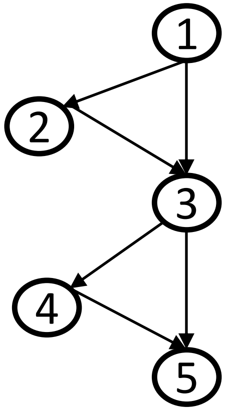
- 基路径覆盖（Prime Path Coverage）
  - 衡量被代码所有基路径被执行的程度
  - 基路径覆盖要求每条基路径至少被执行一次，其定义如下：测试集合T称为基路径覆盖充分的，当且仅当执行T产生的完整路径集合L访问了控制流图中的所有基路径。如果使用符号PP(G)表示控制流图的所有基路径集合，PP(L)表示L访问的基路径集合，则测试集合T的基路径覆盖率为：

$$\frac{||PP(L)||}{||PP(G)||} * 100\%$$

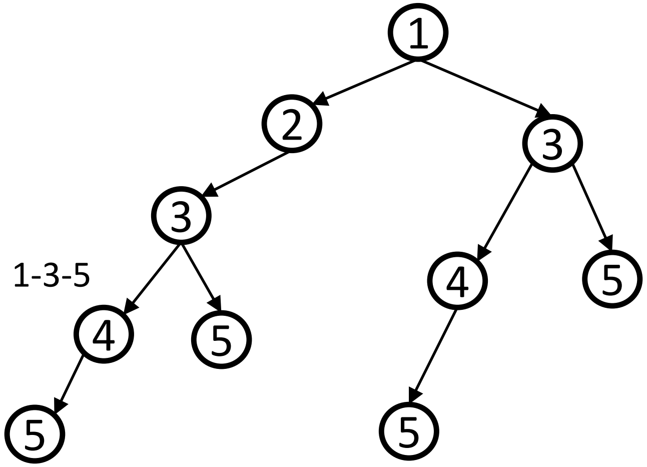
Prime Path Coverage



```
public int  doubleDiamand(int num1, int num2, int num3) {


    if ((num1 > 1) && (num2 == 0))
        num3 /= num1;

    if((num1 == 2) || (num3 > 1))
        num3 += 1;

    return num3;
}
```

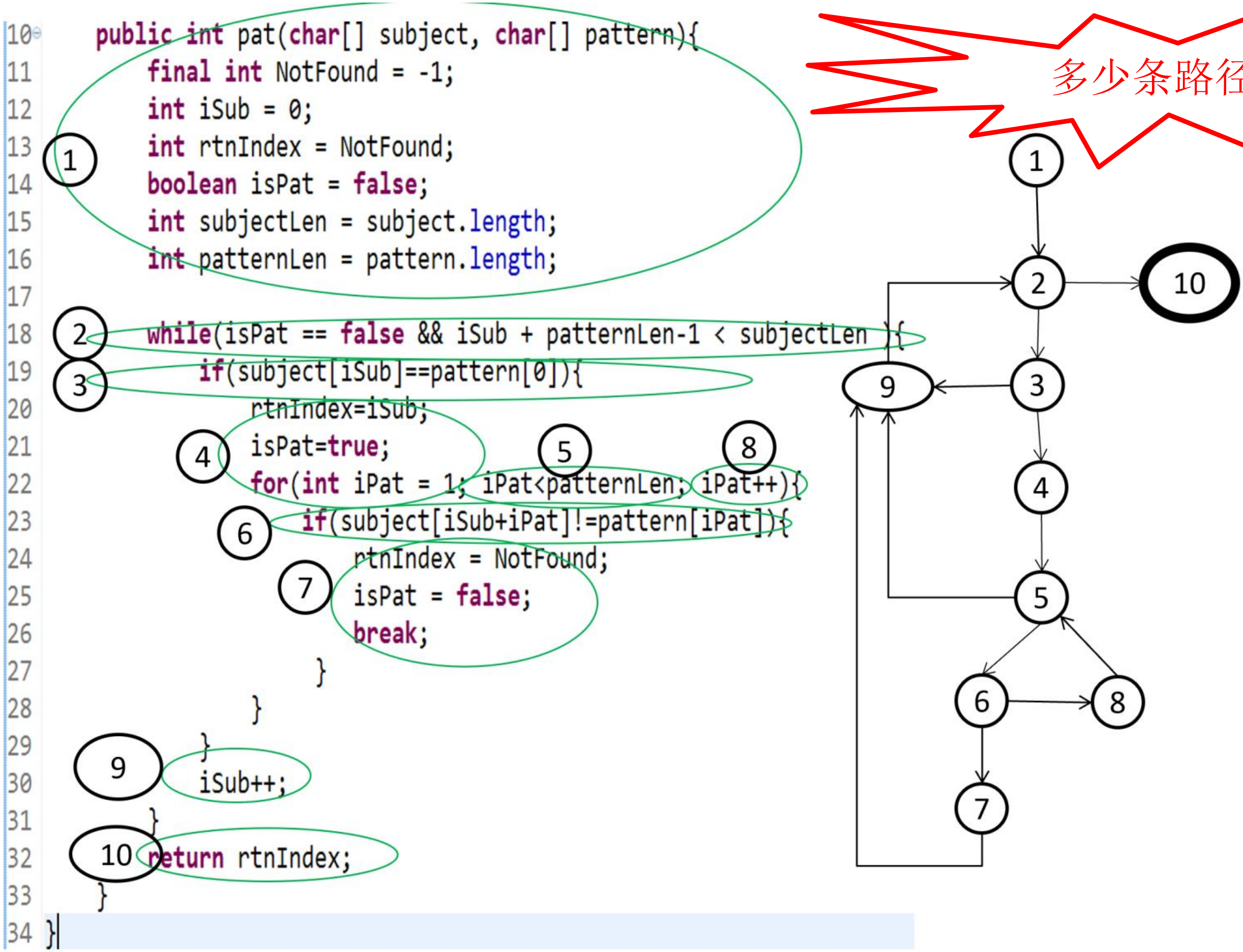


- 测试集合1
  - ① 测试用例1: [(num1=2, num2=0, num3=4) , 3]
  - 测试集合产生的完整路径：1-2-3-4-5
  - 控制流图中的基路径集为： 1-2-3-4-5, 1-2-3-5， 1-3-4-5， 1-3-5
  - 基路径覆盖率 = 1/4 = 25%



1节点树

Prime Path Coverage

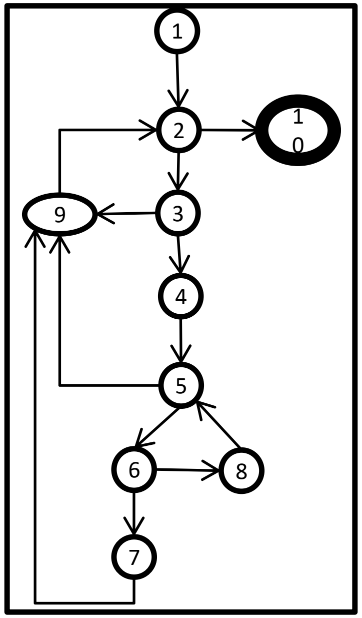


```
public int pat(char[] subject, char[] pattern){
    final int NotFound = -1;
    int iSub = 0;
    int rtnIndex = NotFound;
    boolean isPat = false;
    int subjectLen = subject.length;
    int patternLen = pattern.length;

    while(isPat == false && iSub + patternLen-1 < subjectLen ){
        if(subject[iSub]==pattern[0]){
            rtnIndex=iSub;
            isPat=true;
            for(int iPat = 1; iPat<patternLen; iPat++){
                if(subject[iSub+iPat]!=pattern[iPat]){
                    rtnIndex = NotFound;
                    isPat = false;
                    break;
                }
            }
        }
        iSub++;
    }
    return rtnIndex;
}
}
```

多少条路径？

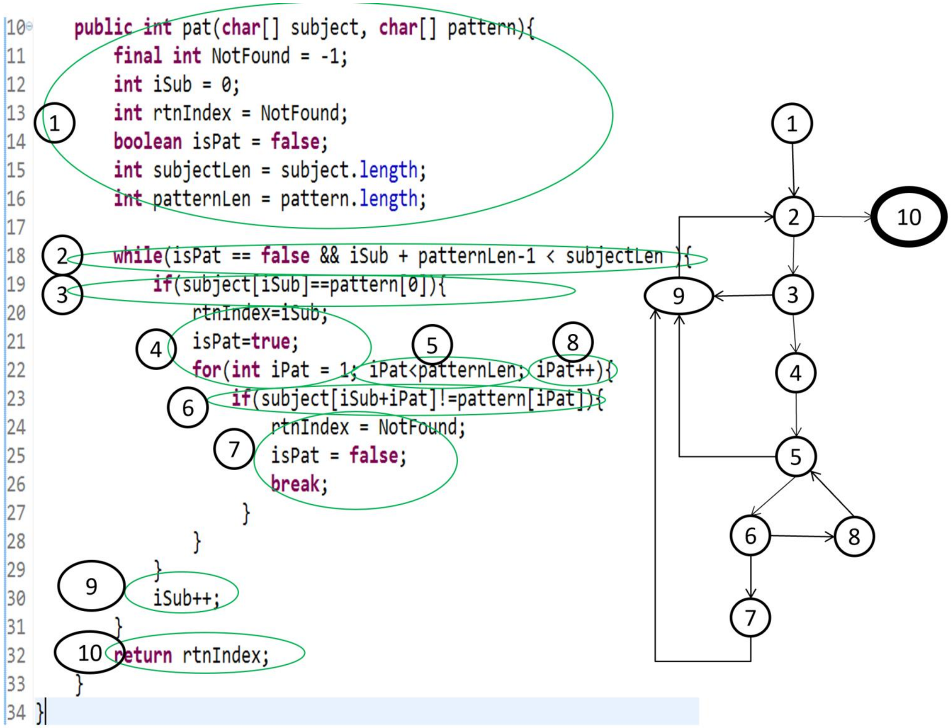
Prime Path Coverage

length = 0	length = 1	length = 2	length = 3	length = 4	length = 5	length = 6	length = 7
1	1,2	**1,2,10**	1,2,3,4	1,2,3,4,5	**1,2,3,4,5,9**	1,2,3,4,5,6,7	**1,2,3,4,5,6,7,9**
2	2,3	1,2,3	**1,2,3,9**	2,3,4,5,6	1,2,3,4,5,6	**1,2,3,4,5,6,8**	**2,3,4,5,6,7,9,2**
3	**2,10**	2,3,4	2,3,4,5	2,3,4,5,9	2,3,4,5,6,7	2,3,4,5,6,7,9	**3,4,5,6,7,9,2,3**
4	3,4	2,3,9	**2,3,9,2**	3,4,5,6,7	**2,3,4,5,6,8**	3,4,5,6,7,9,2	**3,4,5,6,7,9,2,10**
5	3,9	3,4,5	3,4,5,6	**3,4,5,6,8**	**2,3,4,5,9,2**	**4,5,6,7,9,2,10**	**4,5,6,7,9,2,3,4**
6	4,5	3,9,2	3,4,5,9	3,4,5,9,2	3,4,5,6,7,9	4,5,6,7,9,2,3	**5,6,7,9,2,3,4,5**
7	5,6	4,5,6	**3,9,2,10**	4,5,6,7,9	**3,4,5,9,2,10**	5,6,7,9,2,3,4	**6,7,9,2,3,4,5,6**
8	5,9	4,5,9	**3,9,2,3**	4,5,9,2,3	**3,4,5,9,2,3**	6,7,9,2,3,4,5	**7,9,2,3,4,5,6,7**
9	6,7	5,6,7	4,5,6,7	**4,5,9,2,10**	4,5,6,7,9,2	**6,8,5,9,2,3,4**	**7,9,2,3,4,5,6,8**
10	6,8	5,6,8	**4,5,6,8**	5,9,2,3,4	**4,5,9,2,3,4**	7,9,2,3,4,5,6	**8,5,6,7,9,2,3,4**
	7,9	5,9,2	4,5,9,2	5,6,7,9,2	**5,9,2,3,4,5**	8,5,6,7,9,2,3	**9,2,3,4,5,6,7,9**
	8,5	6,7,9	5,6,7,9	**6,7,9,2,10**	5,6,7,9,2,3	**8,5,6,7,9,2,10**	
	9,2	6,8,5	**5,6,8,5**	6,7,9,2,3	**5,6,7,9,2,3,10**	9,2,3,4,5,6,7	
		7,9,2	**5,9,2,10**	6,8,5,9,2	6,7,9,2,3,4	**9,2,3,4,5,6,8**	
		8,5,9	5,9,2,3	7,9,2,3,4	6,8,5,9,2,3		
		8,5,6	6,7,9,2	8,5,9,2,3	**6,8,5,9,2,10**		
		9,2,10	**6,8,5,6**	**8,5,9,2,10**	7,9,2,3,4,5		
		9,2,3	6,8,5,9	8,5,6,7,9	**8,5,9,2,3,4**		
			7,9,2,3	9,2,3,4,5	8,5,6,7,9,2		
			7,9,2,10		**9,2,3,4,5,9**		
			8,5,9,2		9,2,3,4,5,6		
			8,5,6,7				
			8,5,6,8				
			9,2,3,4				
			9,2,3,9				



基路径集合为图中蓝色路径购成：共33条

Prime Path Coverage



```
public int pat(char[] subject, char[] pattern){
    final int NotFound = -1;
    int iSub = 0;
    int rtnIndex = NotFound;
    boolean isPat = false;
    int subjectLen = subject.length;
    int patternLen = pattern.length;

    while(isPat == false && iSub + patternLen-1 < subjectLen ){
        if(subject[iSub]==pattern[0]){
            rtnIndex=iSub;
            isPat=true;
            for(int iPat = 1; iPat<patternLen; iPat++){
                if(subject[iSub+iPat]!=pattern[iPat]){
                    rtnIndex = NotFound;
                    isPat = false;
                    break;
                }
            }
        }
        iSub++;
    }
    return rtnIndex;
  }
}
```

- 测试集合1

  ① 测试用例1: [(subject={‘a’,’b’},pattern={‘a’}) , 0]

  - 测试集合产生的完整路径：1-2-3-4-5-9-2-3-9-2-10, 覆盖的基路径：1-2-3-4-5-9，2-3-4-5-9-2，3-4-5-9-2-3，9-2-3-9，2-3-9-2
  - 基路径覆盖率 = 5/33 = 15.2%

Summary

- Test Adequacy criteria are the foundation of testing which are used to evaluate
- Be satisfied with coverage criteria does not equal a qualified testing
- Not satisfied with coverage criteria must mean a bad testing
- Control Flow Graph is the abstraction of code under test from the control logic point of view
- Control based coverage criteria including: statement, branch, path and prime path coverage

The End