Ch2 Code Unit Test

Automatic Unit Tests Design(1)

Instructor: Haiying SUN

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

Office: ECNU Science Build B1104

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

Overview

- Control Based Tests Generation
- Data Flow Based Testing
- Mutation Testing

Overview

- Control Based Tests Generation
  - Prime Path Testing
- Data Flow Based Testing
- Mutation Testing

Prime Path Testing

- Testing design techniques for designing tests that can cover basic executing paths based on program control structure
  1. Constructing Control Flow Graph (CFG)
  2. Calculating Prime Path set
  3. Deriving Test Cases

Prime Path Testing

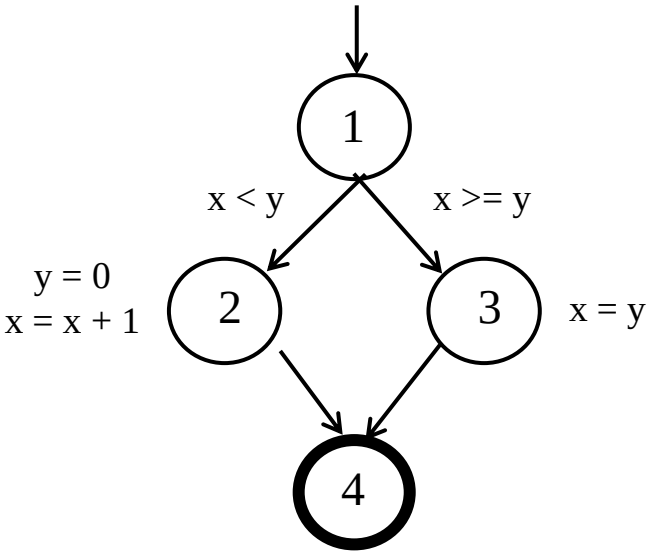
- Testing design techniques for designing tests that can cover basic executing paths based on program control structure
  1. Constructing Control Flow Graph (CFG)
  2. Calculating Prime Path set
  3. Deriving Test Cases

Control Flow Graph

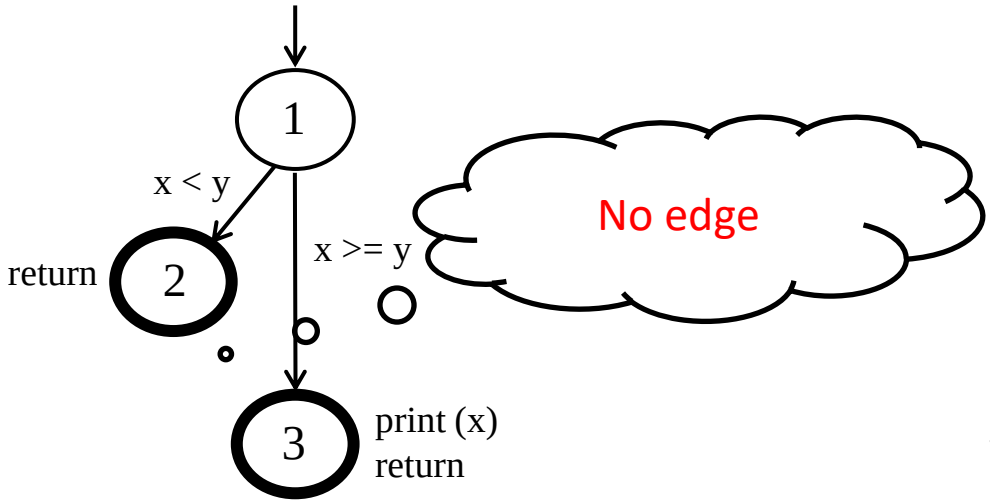
- A CFG models all executions of a method by describing control structures
  1. Nodes : Statements or sequences of statements (basic blocks)
     - **Basic Block** : A sequence of statements such that if the first statement is executed, all statements will be
  2. Edges : Transfers of control

Control Flow Graph

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

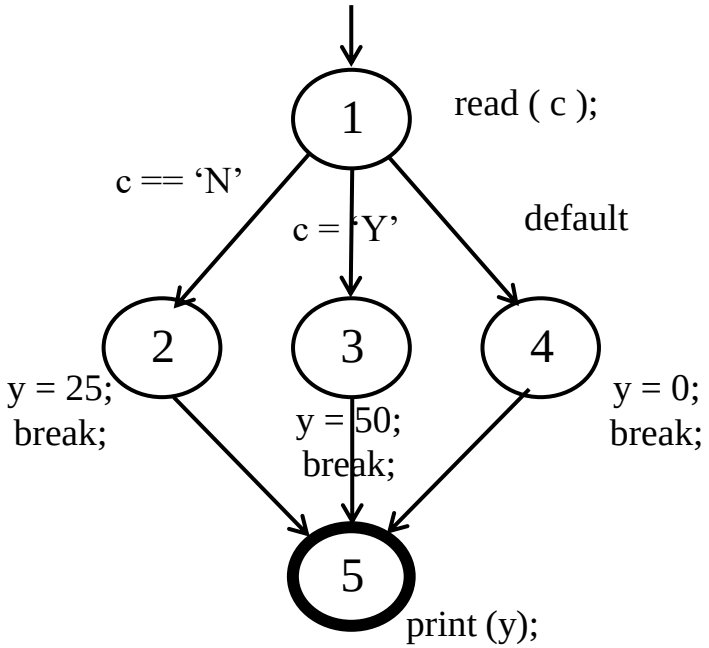



```
if (x < y) {
   return;
}
print (x);
return;
```



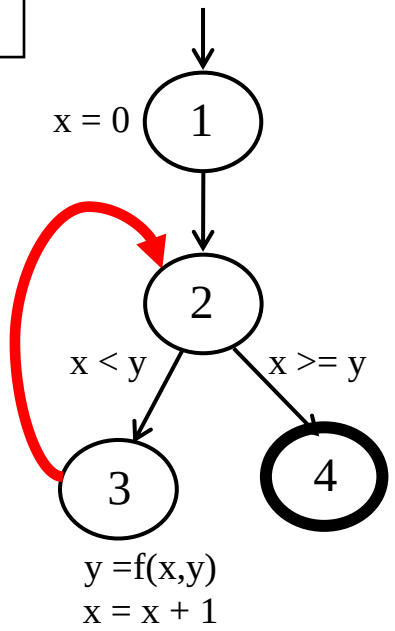
Control Flow Graph

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

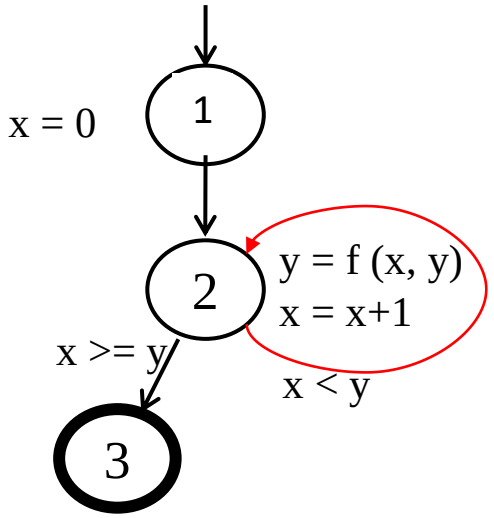


Control Flow Graph

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

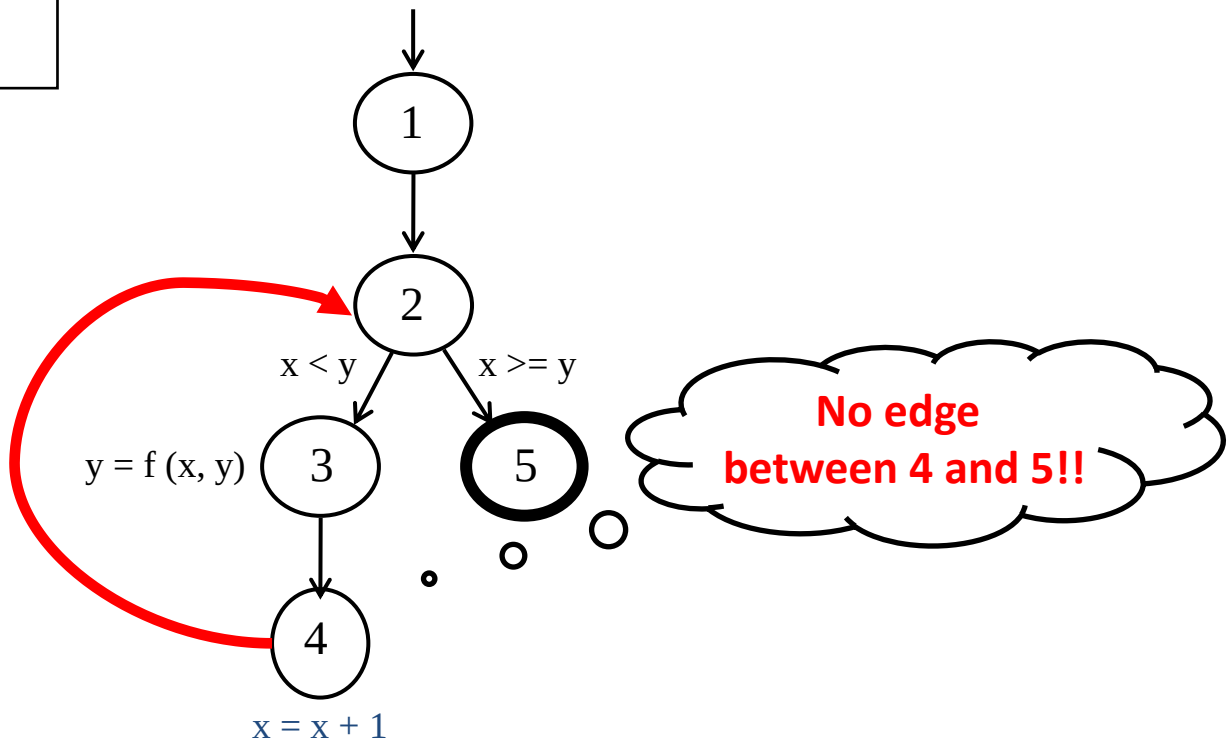



```
x = 0;
do {
   y = f (x, y);
   x = x + 1;
} while (x < y);
print (y)
```



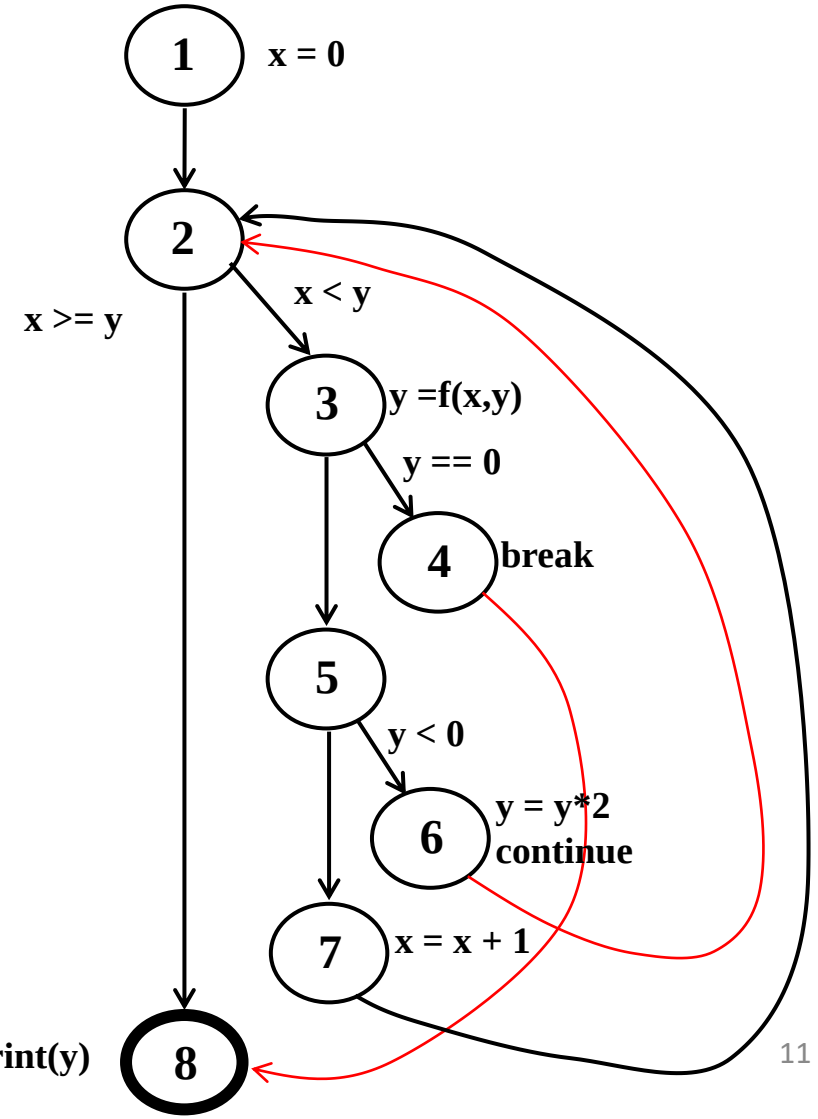
Control Flow Graph

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



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);
```



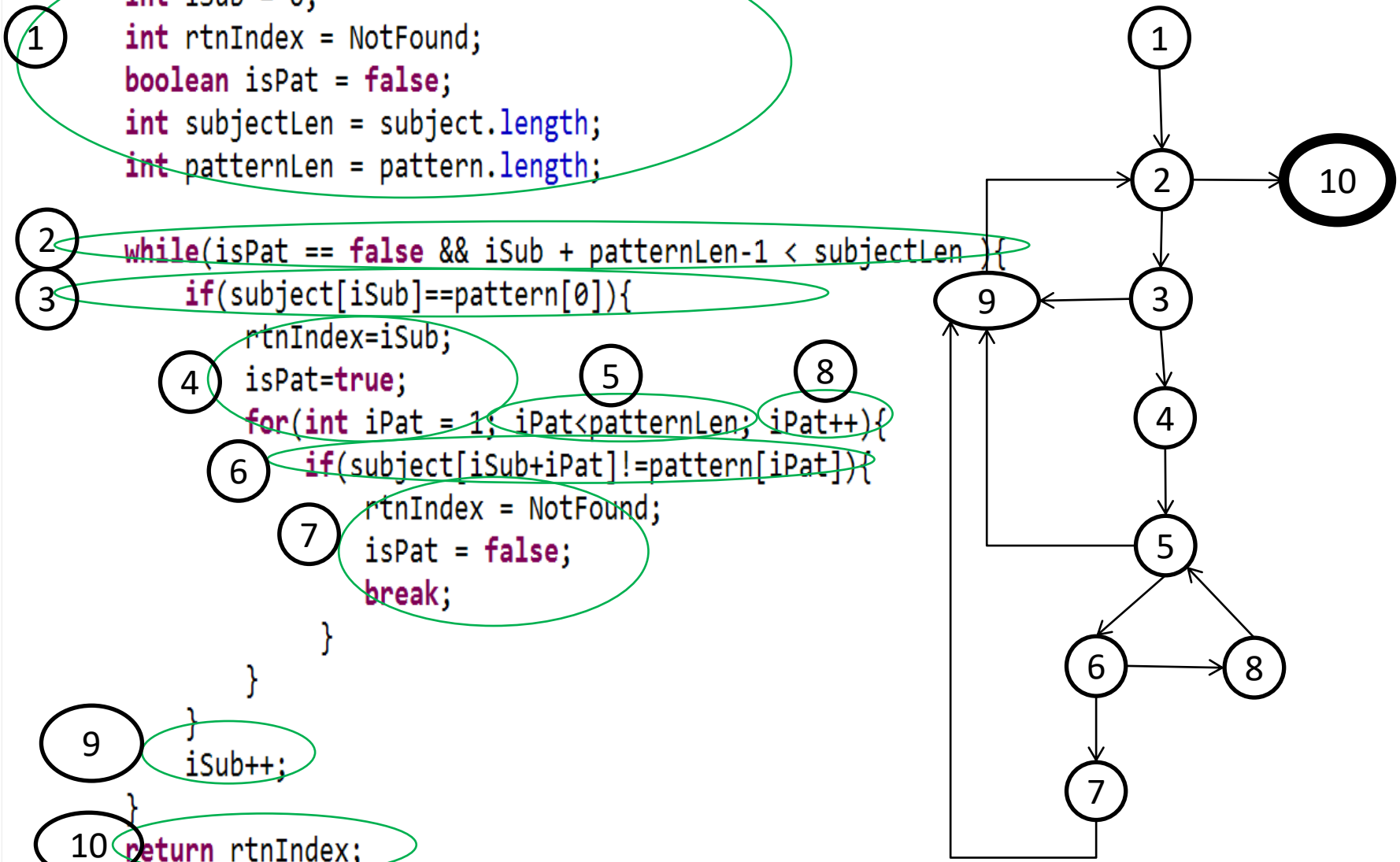
Control Flow Graph

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

Example

```
10      public int pat(char[] subject, char[] pattern){
11          final int NotFound = -1;
12          int iSub = 0;
13          int rtnIndex = NotFound;
14          boolean isPat = false;
15          int subjectLen = subject.length;
16          int patternLen = pattern.length;
17
18          while(isPat == false && iSub + patternLen-1 < subjectLen ){
19              if(subject[iSub]==pattern[0]){
20                  rtnIndex=iSub;
21                  isPat=true;
22                  for(int iPat = 1; iPat<patternLen; iPat++){
23                      if(subject[iSub+iPat]!=pattern[iPat]){
24                          rtnIndex = NotFound;
25                          isPat = false;
26                          break;
27                       }
28                  }
29              }
30              iSub++;
31          }
32          return rtnIndex;
33      }
34  }
```

```
10    public int pat(char[] subject, char[] pattern){
11        final int NotFound = -1;
12        int iSub = 0;
13        int rtnIndex = NotFound;
14        boolean isPat = false;
15        int subjectLen = subject.length;
16        int patternLen = pattern.length;
17
18        while(isPat == false && iSub + patternLen-1 < subjectLen ){
19            if(subject[iSub]==pattern[0]){
20                rtnIndex=iSub;
21                isPat=true;
22                for(int iPat = 1; iPat<patternLen; iPat++){
23                    if(subject[iSub+iPat]!=pattern[iPat]){
24                        rtnIndex = NotFound;
25                        isPat = false;
26                        break;
27                    }
28                }
29            }
30            iSub++;
31        }
32        return rtnIndex;
33    }
34 }
```



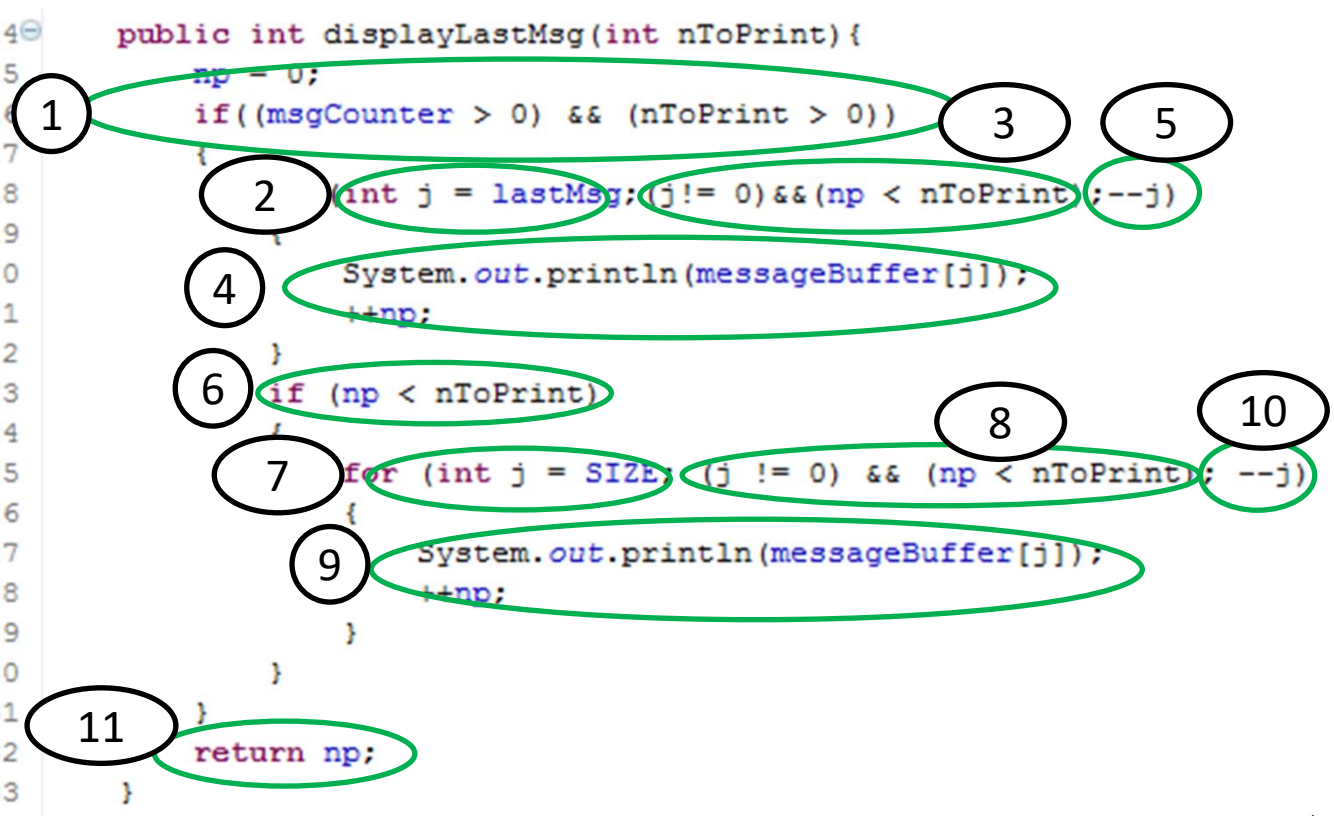
Exercise

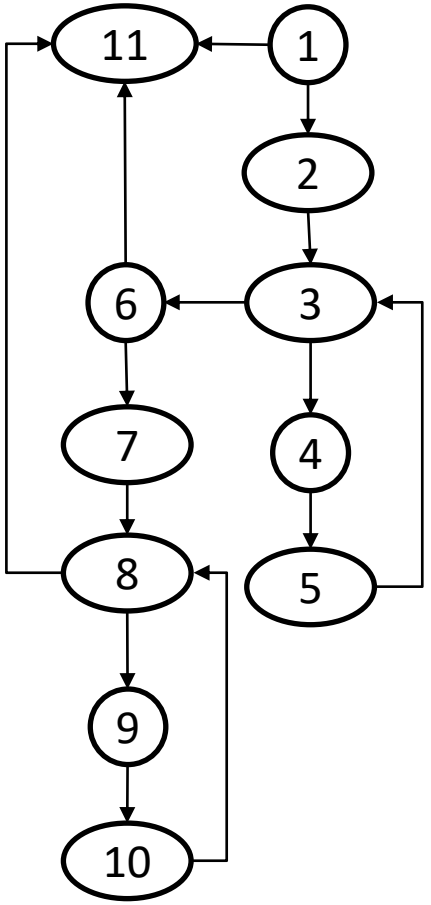
- Draw CFG for *displayLastMsg*

```
14    public int displayLastMsg(int nToPrint){
15        np = 0;
16        if((msgCounter > 0) && (nToPrint > 0))
17        {
18            for(int j = lastMsg;(j!= 0)&&(np < nToPrint);--j)
19            {
20                System.out.println(messageBuffer[j]);
21                ++np;
22            }
23            if (np < nToPrint)
24            {
25                for (int j = SIZE; (j != 0) && (np < nToPrint); --j)
26                {
27                    System.out.println(messageBuffer[j]);
28                    ++np;
29                }
30            }
31        }
32        return np;
33    }
```

Exercise

```
14    public int displayLastMsg(int nToPrint){
15        np = 0;
16        if((msgCounter > 0) && (nToPrint > 0))
17        {
18            for(int j = lastMsg;(j!= 0)&&(np < nToPrint);--j)
19            {
20                System.out.println(messageBuffer[j]);
21                ++np;
22            }
23            if (np < nToPrint)
24            {
25                for (int j = SIZE; (j != 0) && (np < nToPrint); --j)
26                {
27                    System.out.println(messageBuffer[j]);
28                    ++np;
29                }
30            }
31        }
32        return np;
33    }
```





Prime Path Testing

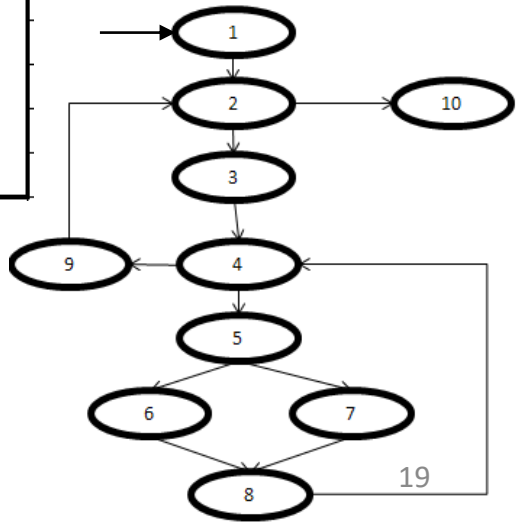
- Testing design techniques for designing tests that can cover basic executing paths based on program control structure
  1. Constructing Control Flow Graph (CFG)
  2. Calculating Prime Path set
  3. Deriving Test Cases

Paths in Graphs

- Path is a sequence of nodes: $(n_1, n_2, \ldots, n_m)$ where for $1 \leq i < m$, $(n_i, n_{i+1}) \in E$
  - 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

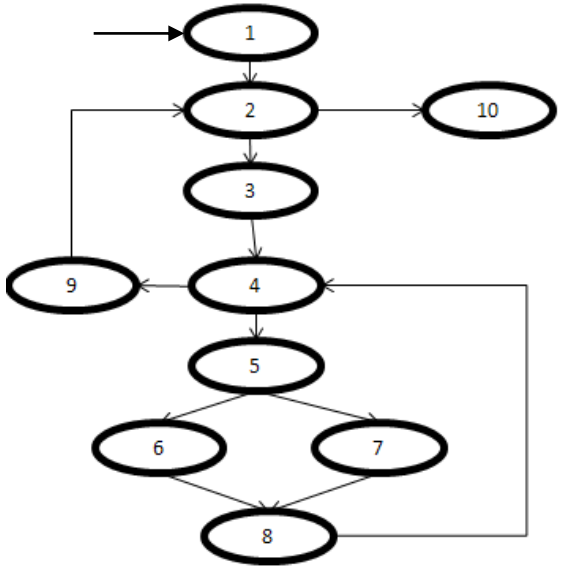
Path Examples

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

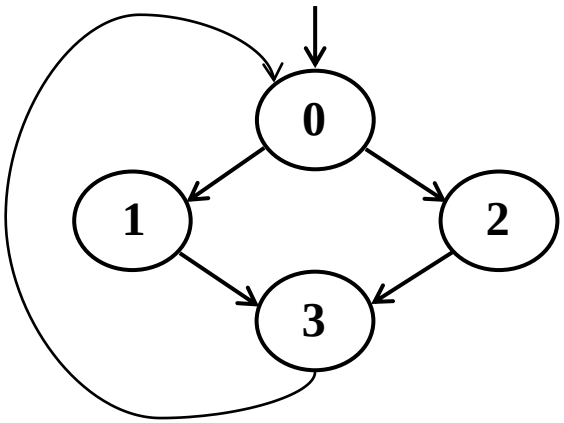


Prime Path Testing

- 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

- 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]

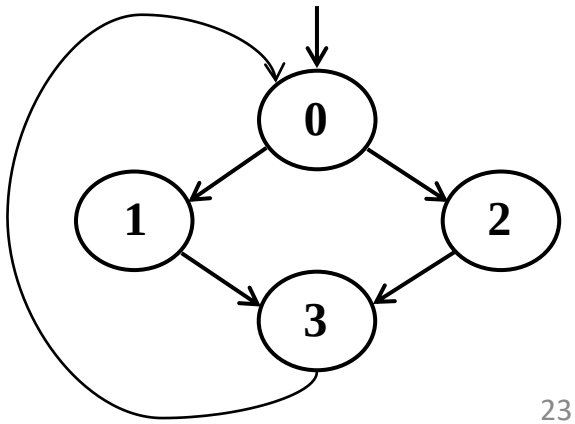


Prime Path

- 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

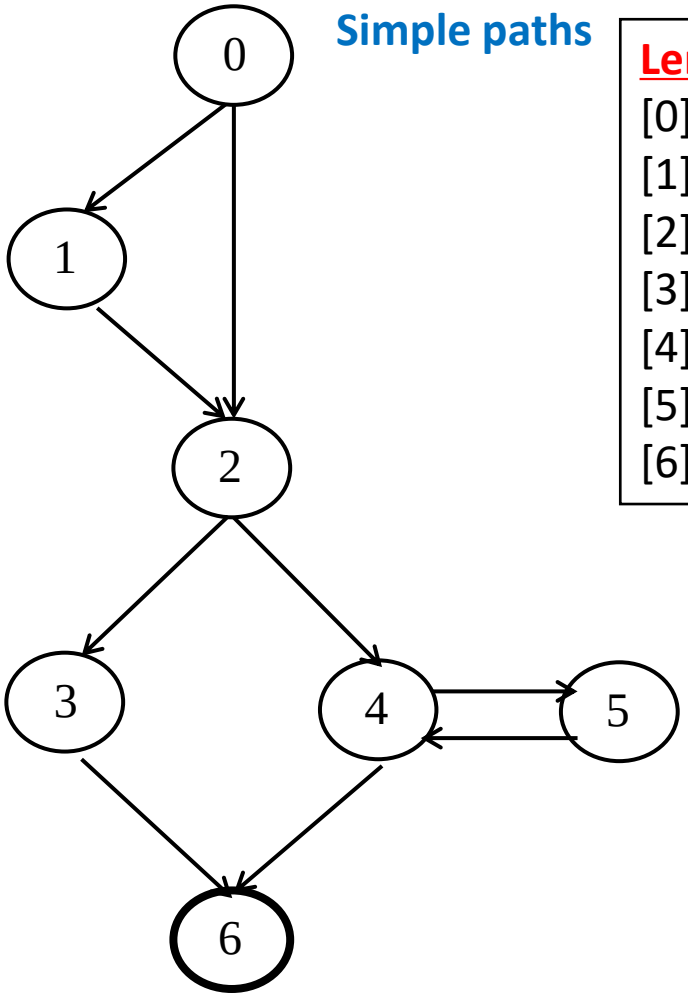
- How to calculate prime path
  1. Exhaust method
  2. Node tree method

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 ∩ 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]

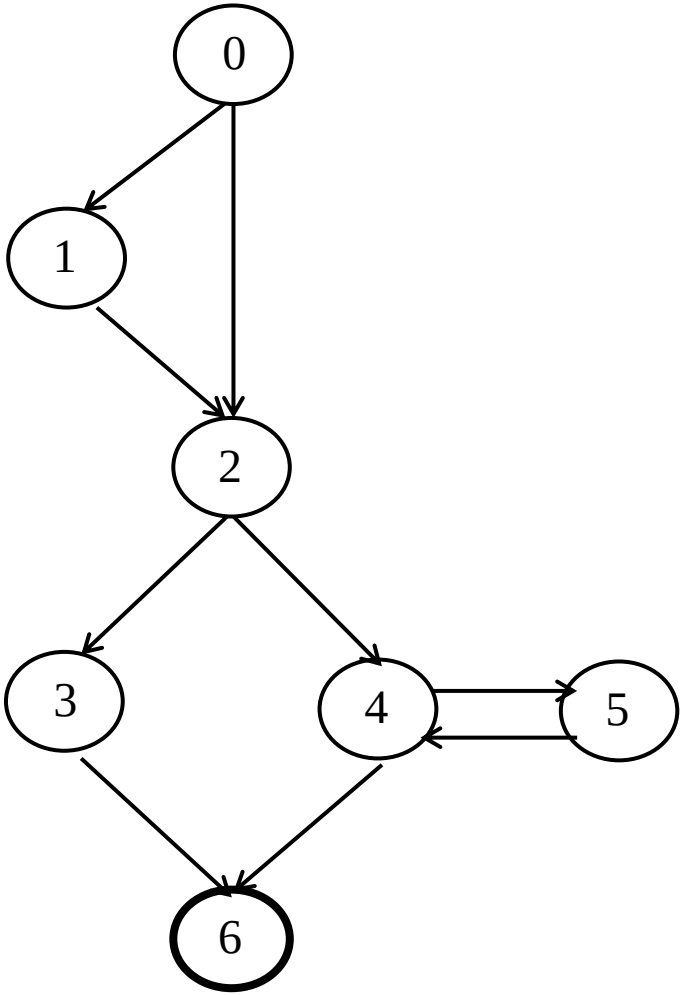
Len 2
[0, 1, 2]
[0, 2, 3]
[0, 2, 4]
[1, 2, 3]
[1, 2, 4]
[2, 3, 6] !
[2, 4, 6] !
[2, 4, 5] !
[4, 5, 4] *
[5, 4, 6] !
[5, 4, 5] *

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

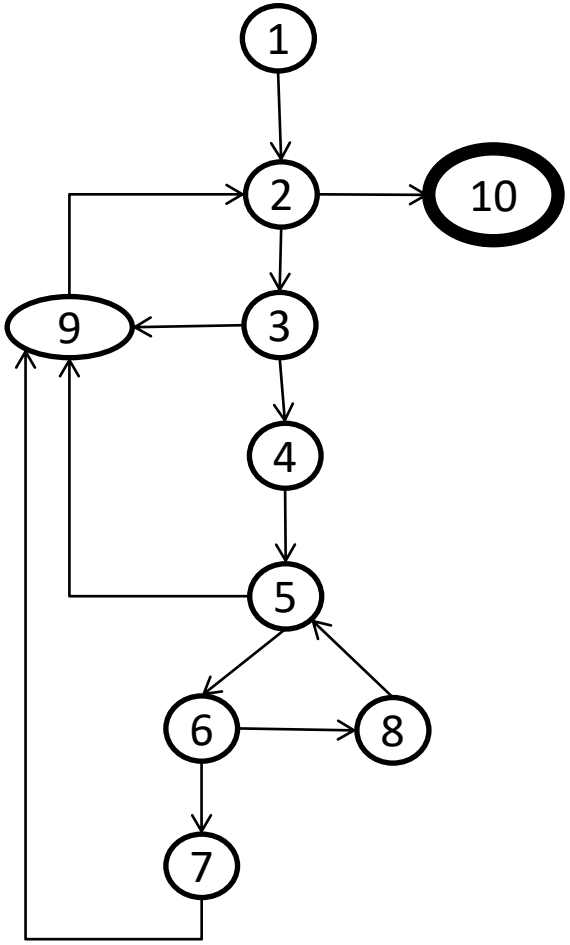
[0, 1, 2, 3, 6]
[0, 1, 2, 4, 5]
[0, 1, 2, 4, 6]
[0, 2, 3, 6]
[0, 2, 4, 5]
[0, 2, 4, 6]
[5, 4, 6]
[4, 5, 4]
[5, 4, 5]

Execute loop 0 times

Execute loop once

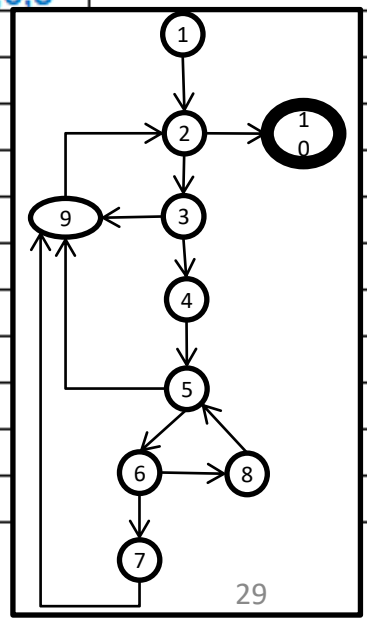
Execute loop more than once

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

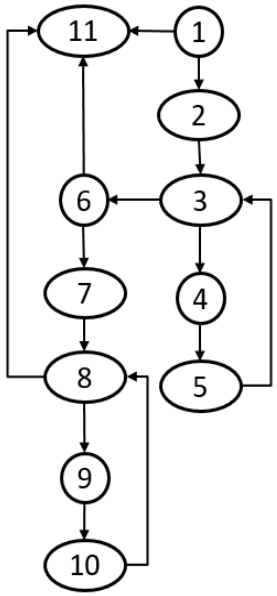
Exercise

- Calculate Prime Paths *displayLastMsg* based on your CFG

```
14  public int displayLastMsg(int nToPrint){
15      np = 0;
16      if((msgCounter > 0) && (nToPrint > 0))
17      {
18          for(int j = lastMsg;(j!= 0)&&(np < nToPrint);--j)
19          {
20              System.out.println(messageBuffer[j]);
21              ++np;
22          }
23          if (np < nToPrint)
24          {
25              for (int j = SIZE; (j != 0) && (np < nToPrint); --j)
26              {
27                  System.out.println(messageBuffer[j]);
28                  ++np;
29              }
30          }
31      }
32      return np;
33  }
```

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

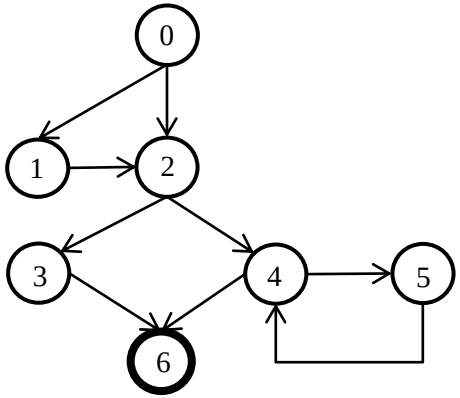
Node Tree Method

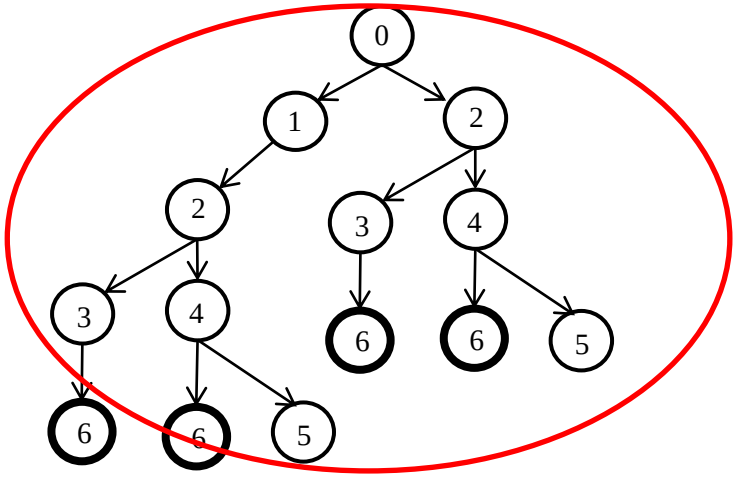
- New Algorithm:
  - DDL_PrimePathCal
- Verify Prime Path Results
  - Compare with two different kinds of algorithm
  1. Exhaust_PrimePathCal
  2. DDL_PrimePathCal

Node Tree Method

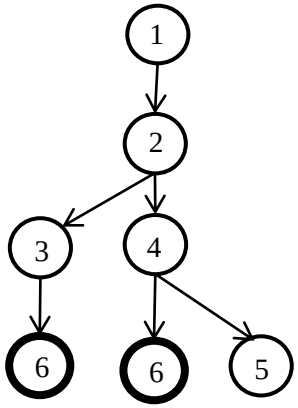
- 节点树
  - 以G中的节点为根节点建立的树，且满足树中除根节点和叶节点可以相同外，从根节点到每个树中节点的路径中，每个节点的出现次数有且仅有 1 次。
  - 在节点树中，每条从根节点到叶节点的路径即为一条简单路径。
- 简单节点树
  - 若节点树T不是任何其它节点树的子树，则称节点树T为简单节点树。
  - 所有简单节点树的从根节点到叶节点的路径集合为备选的基路径集合。

DDL_PrimePathCal Example

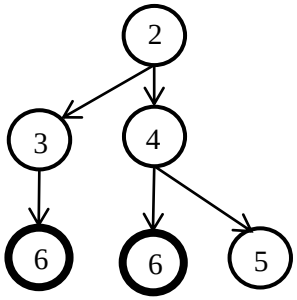




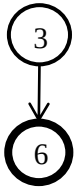
0节点的节点树



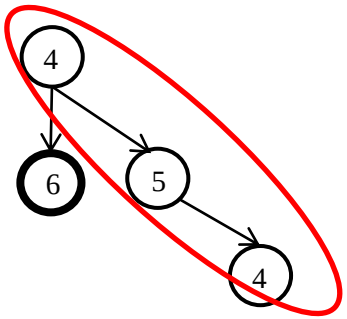
1节点的节点树



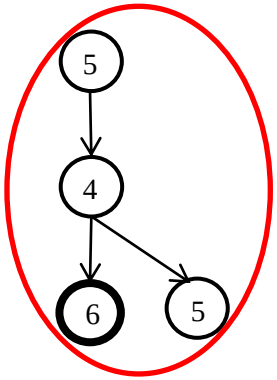
2节点的节点树



3节点的节点树

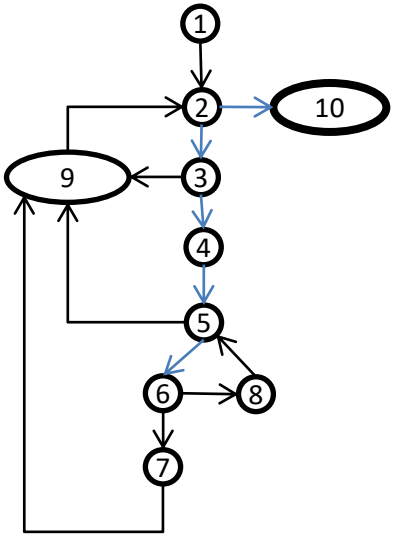


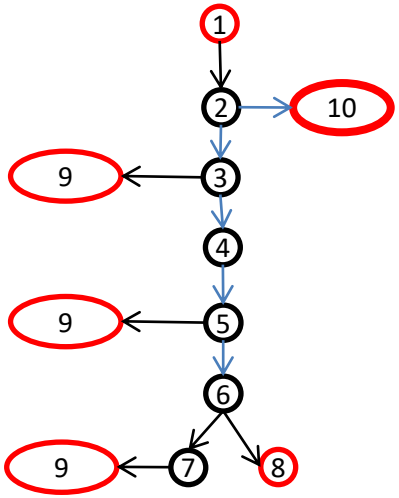
4节点的节点树

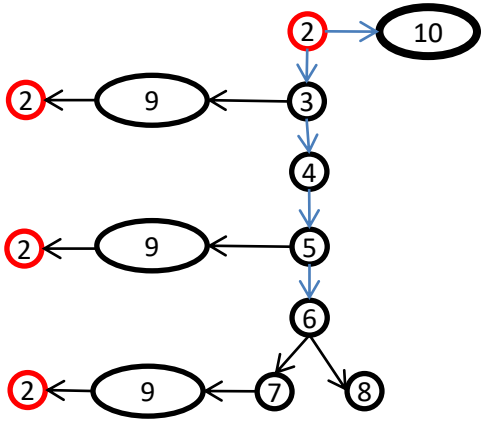


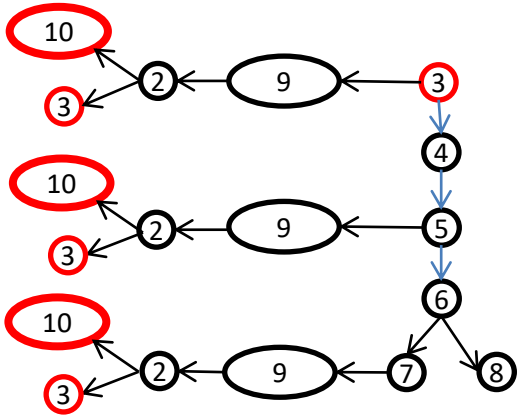
5节点的节点树

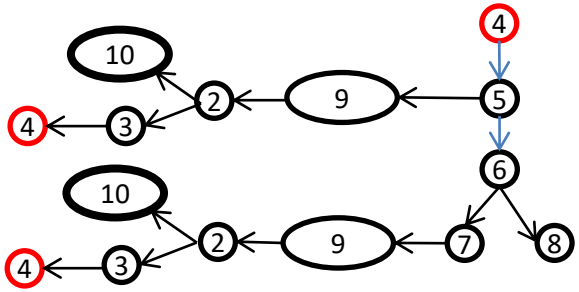
DDL_PrimePathCal Exercise

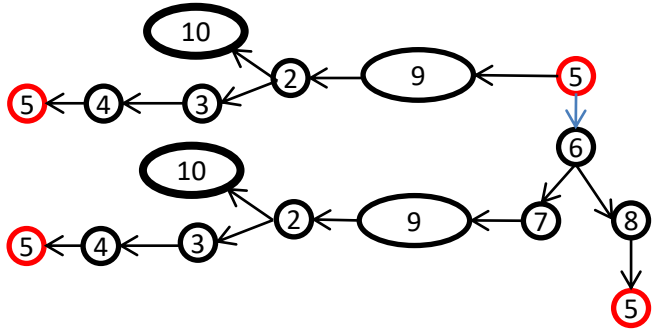






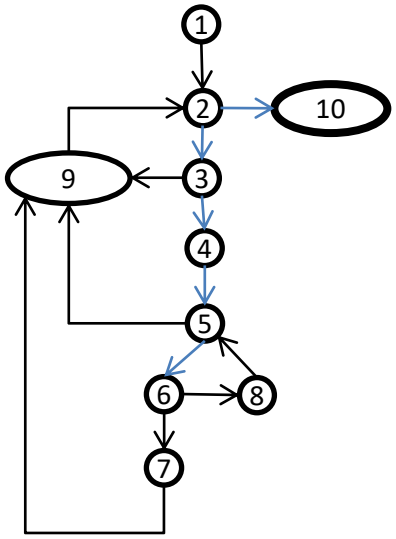


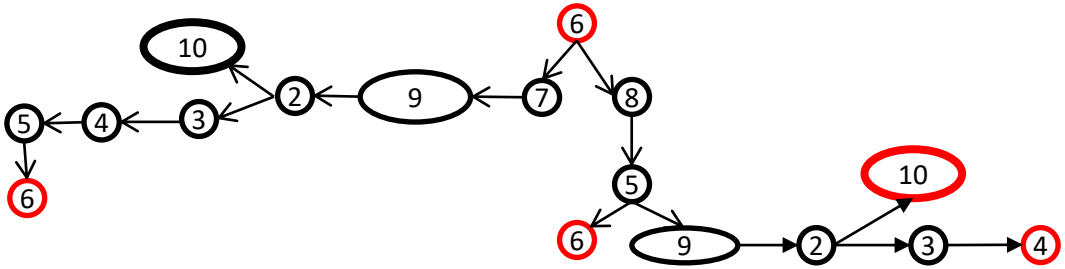


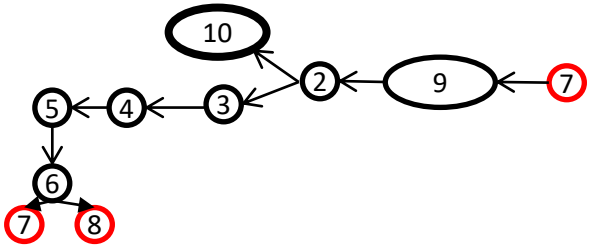


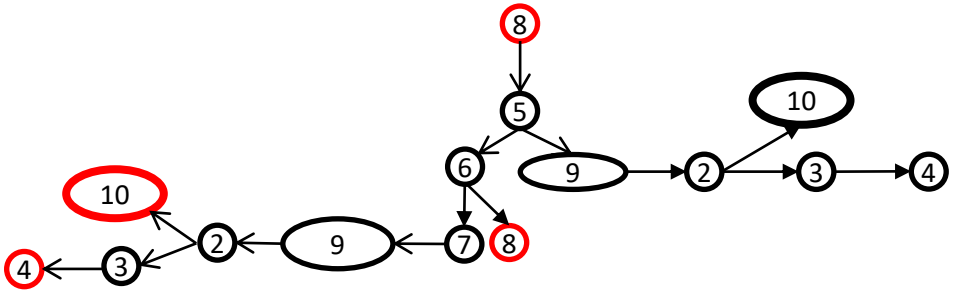
每条基路径由图中红色的起点和终点标注

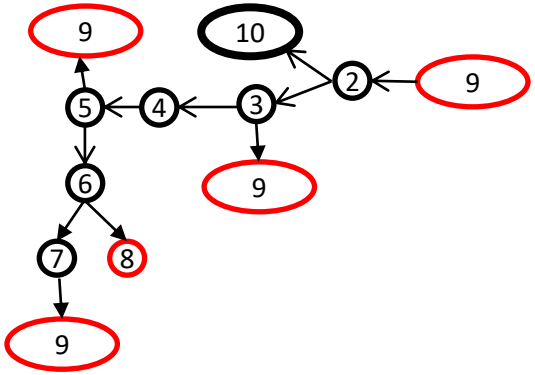
DDL_PrimePathCal Exercise











每条基路径由图中红色的起点和终点标注

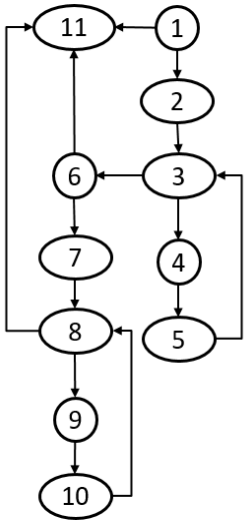
Exercise

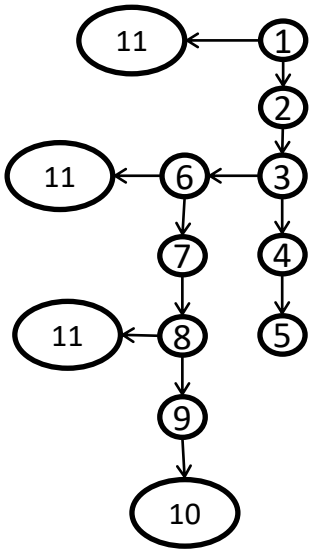
- Calculate Prime Paths *displayLastMsg* based on your CFG

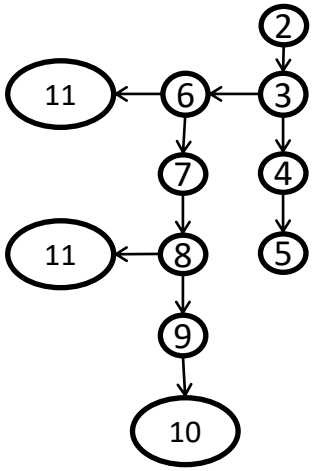
```
14  public int displayLastMsg(int nToPrint){
15      np = 0;
16      if((msgCounter > 0) && (nToPrint > 0))
17      {
18          for(int j = lastMsg;(j!= 0)&&(np < nToPrint);--j)
19          {
20              System.out.println(messageBuffer[j]);
21              ++np;
22          }
23          if (np < nToPrint)
24          {
25              for (int j = SIZE; (j != 0) && (np < nToPrint); --j)
26              {
27                  System.out.println(messageBuffer[j]);
28                  ++np;
29              }
30          }
31      }
32      return np;
33  }
```

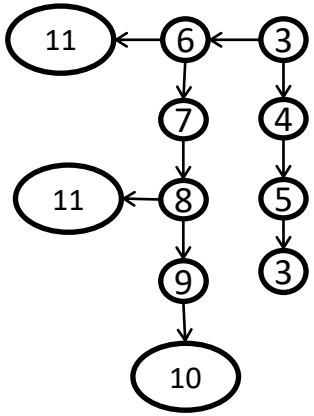
Exercise

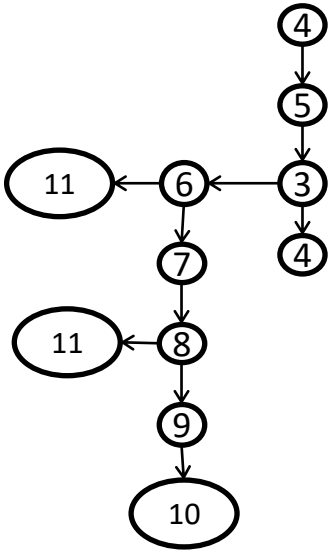
- Calculate Prime Paths *displayLastMsg* based on your CFG











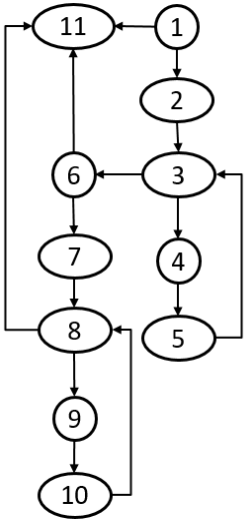
1-11
1-2-3-4-5
1-2-3-6-7-8-9-10
1-2-3-6-11
1-2-3-6-7-8-11

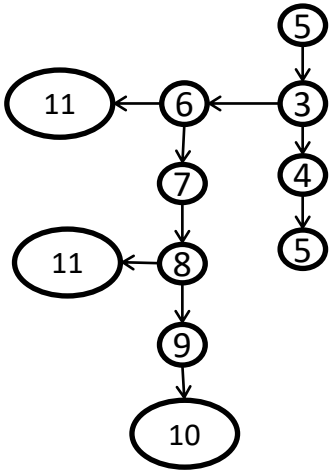
3-4-5-3

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

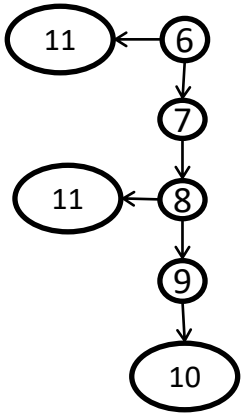
Exercise

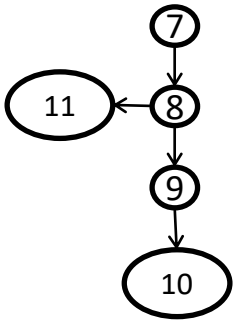
- Calculate Prime Paths *displayLastMsg* based on your CFG

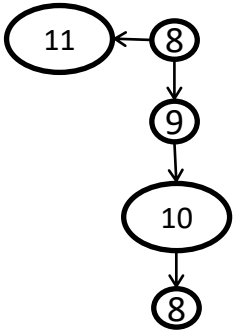




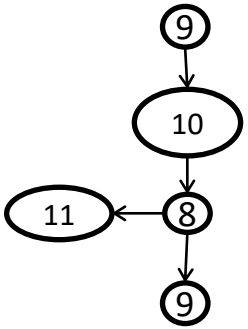
5-3-4-5



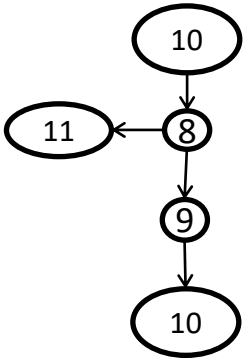




8-9-10-8



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



10-8-9-10

Prime Path Testing

- Testing design techniques for designing tests that can cover basic executing paths based on program control structure
  1. Constructing Control Flow Graph (CFG)
  2. Calculating Prime Path set
  3. Deriving Test Cases

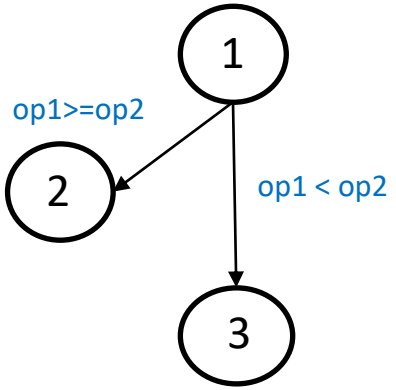
Deriving Test Cases

- 测试生成算法目标
  - **Prime Path Coverage (PPC)**： Each prime path should be tested by at least one test case
- 步骤
  - 计算满足PPC的完全路径
    - 从最长的基路径开始，将其扩展成包含CFG的初始节点和终止节点的完全路径，然后，按照这个方法扩展剩下的没有被遍历的最长的基路径，依次类推……, 直到所有基路径都被扩展为止
  - 计算使得路径能够被测试的输入作为测试输入

1. 观察法
2. 约束求解法：构造路径表达式并对其求解。当构成路径表达式的各个变量的取值使得路径表达式的结果为真，说明在该取值下，程序执行该条路径。

Demo

```
1  public class SimpleClass {
2
3      public int findMax(int op1,int op2){
4          if (op1 >= op2)
5              return op1;
6          else
7              return op2;
8      }
9  }
```



PPC：
1-2的路径表达式：op1>op2
1-3的路径表达式：op1<op2

若自动计算op1，op2取何值时op1>=op2成立

```
1 (declare-const op1 Int)
2 (declare-const op2 Int)
3 (assert(>= op1 op2))
4 (check-sat)
5 (get-model)
```

```
sat
(model
  (define-fun op2 () Int
    0)
  (define-fun op1 () Int
    0)
)
```

⇨ op1 =0, op2=0

Coverage Criteria

- Node Coverage (Statement Coverage)
- Edge Coverage (Branch Coverage)
- Prime Path Coverage
- Edge Pair Coverage
- Special Path Coverage
- Complete Path Coverage

Coverage Criteria

- Node Coverage
  - Each reachable node should be visited (by a test path)
  - TR contains each reachable nodes
- Edge Coverage
  - Each reachable edge should be visited (by a test path)
  - TR contains each reachable path of length up to 1

Coverage Criteria

- Visit : A test path *p* *visits* node *n* if *n* is in *p*

  A test path *p* *visits* edge *e* if *e* is in *p*
- Tour : A test path *p* *tours* subpath *q* if *q* is a subpath of *p*

Test Path [0, 1, 3, 4, 6]
Visits nodes 0, 1, 3, 4, 6
Visits edges (0, 1), (1, 3), (3, 4), (4, 6)
Tours subpaths [0, 1, 3], [1, 3, 4], [3, 4, 6], [0, 1, 3, 4], [1, 3, 4, 6]

Coverage Criteria

- Edge-Pair Coverage (EPC) : TR contains each reachable path of length up to 2, inclusive, in G.

  …….. length up to x

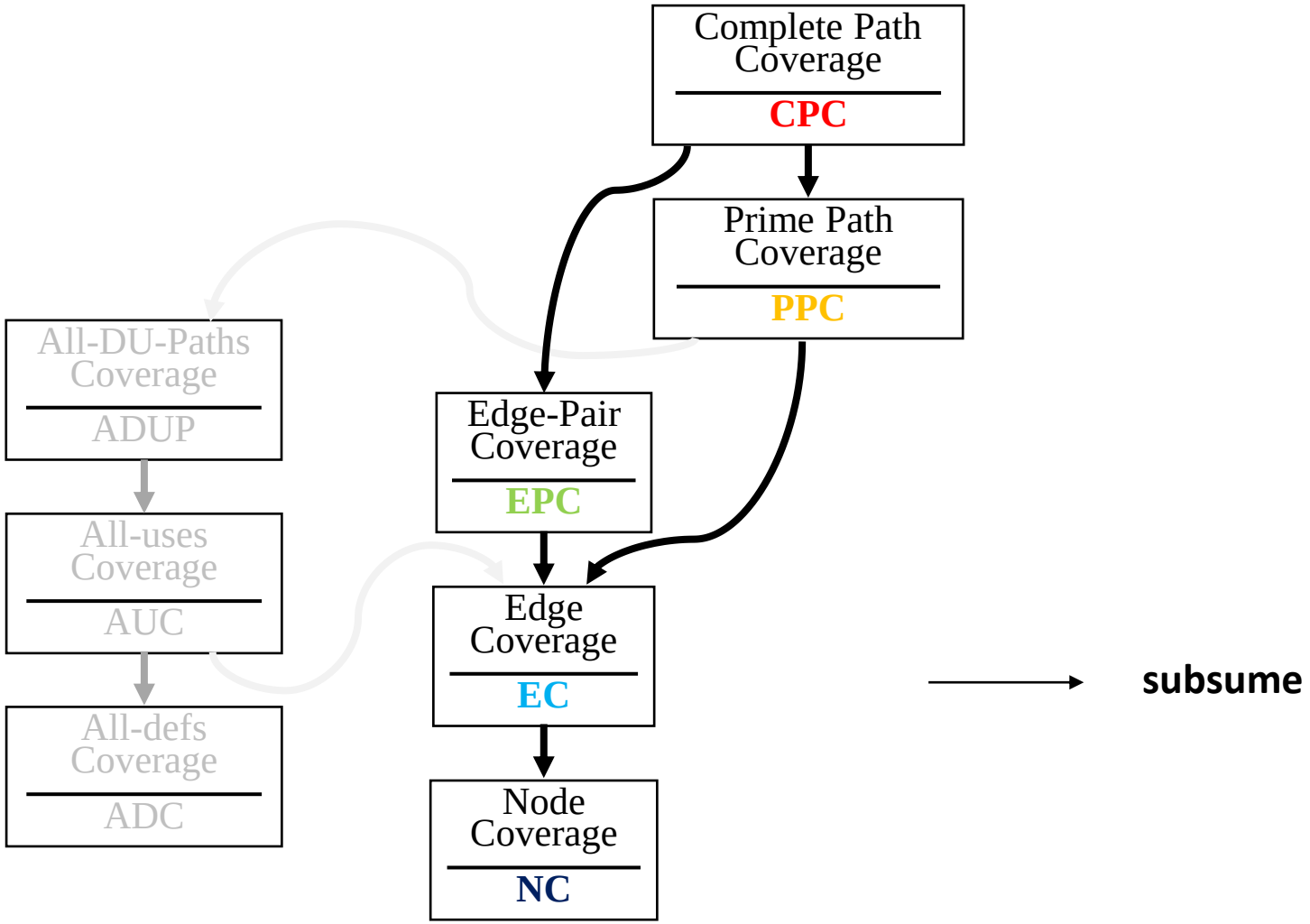
- The logical extension is to require all paths, that is,

  Complete Path Coverage (CPC) : TR contains all paths in G

- Unfortunately, this is impossible if the graph has a loop, so a weak compromise is to make the tester decide which paths:

  - Specified Path Coverage (SPC) : TR contains a set S of test paths, where S is supplied as a parameter.

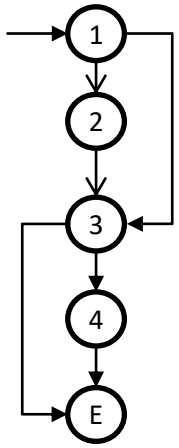
Defect Detection Ability Evaluation



Infeasible Path

- An infeasible Path cannot be satisfied
  - Unreachable statement (dead code)
  - A subpath that a contradiction occurs (x > 0 and x < 0)
- Modify infeasible paths
- It is usually undecidable whether all paths are feasible

```
①  aFlag = false;
   if  iNum != 0
②    aFlag = true ;
③  calEffort();
   if  aFlag && isFree
④    sendMessage();
```



Infeasible Path: (1,3,4,E)

Conclusion

- Control Flow Graph is the abstraction model of program under test from the control logic perspective
- Many test design methods can be designed based on touring different structures of CFG, node, edge, prime path is the usual cases
- Prime Path Testing is the basic paths , more test cases should be added if one wants detect more bugs
- The primary idea on code based test generation is that constructing path expressions according to CFG and deriving test inputs by solving these path expressions.

The End