887493
Software Testing : White Box (Part One)

เอกภพ บุญเพ็ง

aekapop@go.buu.ac.th

หัวข้อบรรยาย

- การทดสอบแบบ White Box
  - การทดสอบกระแสควบคุม (Control Flow Testing) (Part One and Two)
  - การทดสอบกระแสข้อมูล (Data Flow Testing) (Part Two)

การทดสอบแบบ White Box

การทดสอบแบบ White Box

- เป็นการทดสอบกลไกภายในของโปรแกรมที่ถูกพัฒนาขึ้นเพื่อให้ครอบคลุมการทำงานต่าง ๆ
  - เส้นทางการทำงาน
  - เงื่อนไขการทำงาน
- สามารถแบ่งออกเป็น 2 วิธีหลัก
  - การทดสอบกระแสควบคุม
  - การทดสอบกระแสข้อมูล

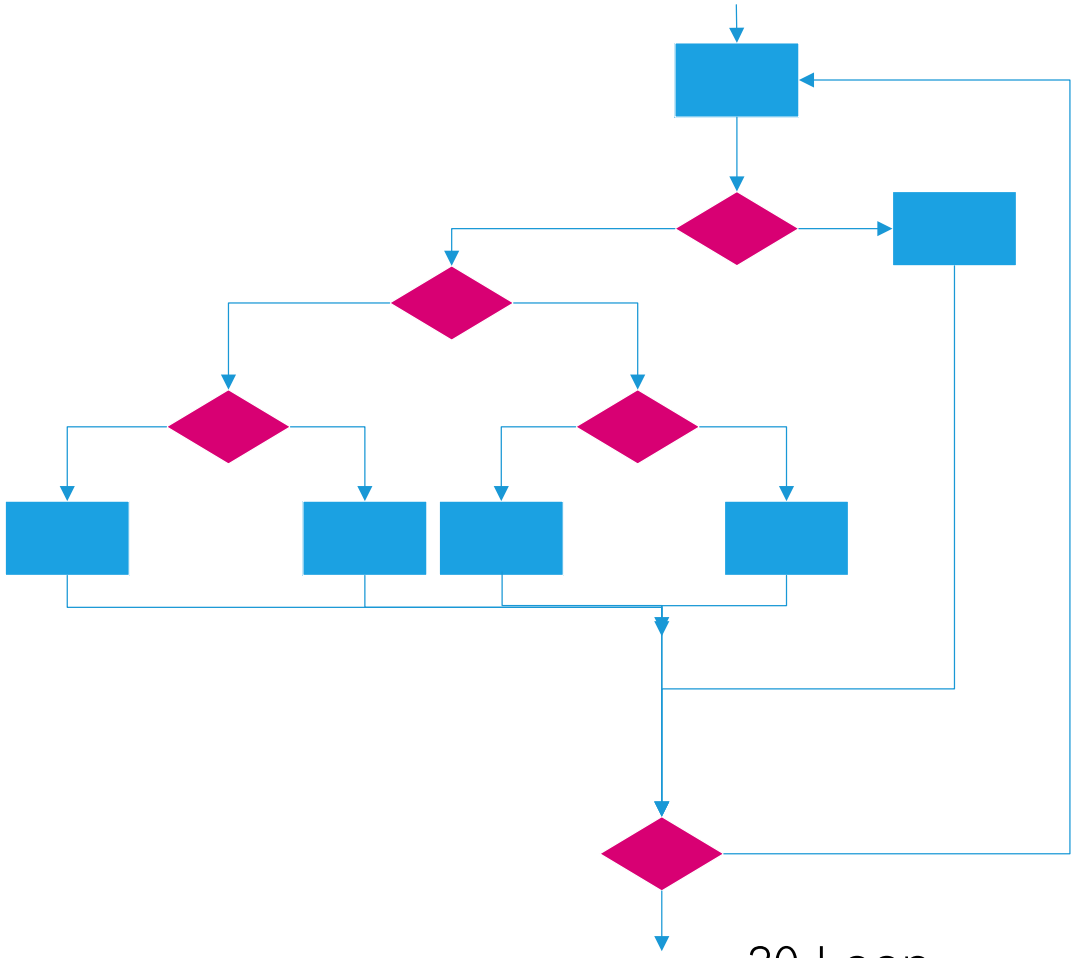
การทดสอบกระแสควบคุม (Control Flow Testing)

การทดสอบกระแสควบคุม (Control Flow Testing)

- เน้นไปที่การตรวจสอบโครงสร้างภายในเป็นหลัก
- การทำงานทุกส่วนจะต้องถูกประมวลผลอย่างน้อยหนึ่งครั้ง
- บางครั้งโปรแกรมอาจซับซ้อนมากทำให้ทดสอบมีความลำบาก

ตัวอย่างโปรแกรมที่ซับซ้อน

จากรูป หากต้องทดสอบด้วยจำนวนเส้นทางอย่างสมบูรณ์แบบ อาจมีเส้นทางการทดสอบที่เป็นไปได้ทั้งหมด 5^20 หากการทดสอบต่อรอบใช้เวลา 1 mSec จะใช้เวลารวม 3.170 ปี



20 Loop

การทดสอบกระแสควบคุม (Control Flow Testing)

- แบ่งออกเป็น 2 วิธี
  - การวิเคราะห์ความครอบคลุมของ Code (Code Coverage Analysis) เป็นการเปรียบเทียบการทำงานขอโปรแกรมกับ Code โดยเน้นไปที่ความเป็นไปได้ของการเกิดข้อบกพร่อง ภายในโครงสร้างลอจิก
  - การทดสอบเส้นทางการประมวลผลหลัก (Basic Path Testing) เป็นวิธีทดสอบที่ใช้การวัดความซับซ้อนทางลอจิกของการออกแบบ และใช้ผลลัพธ์ของการวัดกำหนดเส้นทางการประมวลผล

การวิเคราะห์ความครอบคลุมของ Code (Code Coverage Analysis)

- เป็นเทคนิคที่ถูกพัฒนาขึ้นเพื่อเป็นวิธีการวัดแบบหนึ่งที่ใช้ร่วมกับการทดสอบเชิงโครงสร้าง เพื่อให้แน่ใจว่า Code ของโปรแกรมมีการทำงานเป็นไปตามที่คาดหวังไว้หรือไม่ สามารถทำได้หลายระดับดังนี้
  - การทดสอบความครอบคลุมระดับคำสั่ง (Statement Coverage) การทดสอบแบบนี้ Code ทุกบรรทัดจะถูกประมวลผลอย่างน้อย 1 ครั้ง
  - การทดสอบความครอบคลุมระดับการตัดสินใจ (Branch Coverage) เป็นการทดสอบเพื่อยืนยันความถูกต้องของทุก ๆ การตัดสินใจที่เกิดขึ้นภายใน Code
  - การทดสอบความครอบคลุมระดับเส้นทางประมวลผล (Path Coverage) เป็นการทดสอบเพื่อให้แน่ใจว่าทุก ๆ เส้นทางภายในโปรแกรมจะต้องมีการประมวลผลเกิดขึ้นอย่างน้อย 1 ครั้ง

การทดสอบความครอบคลุมระดับคำสั่ง (Statement Coverage)

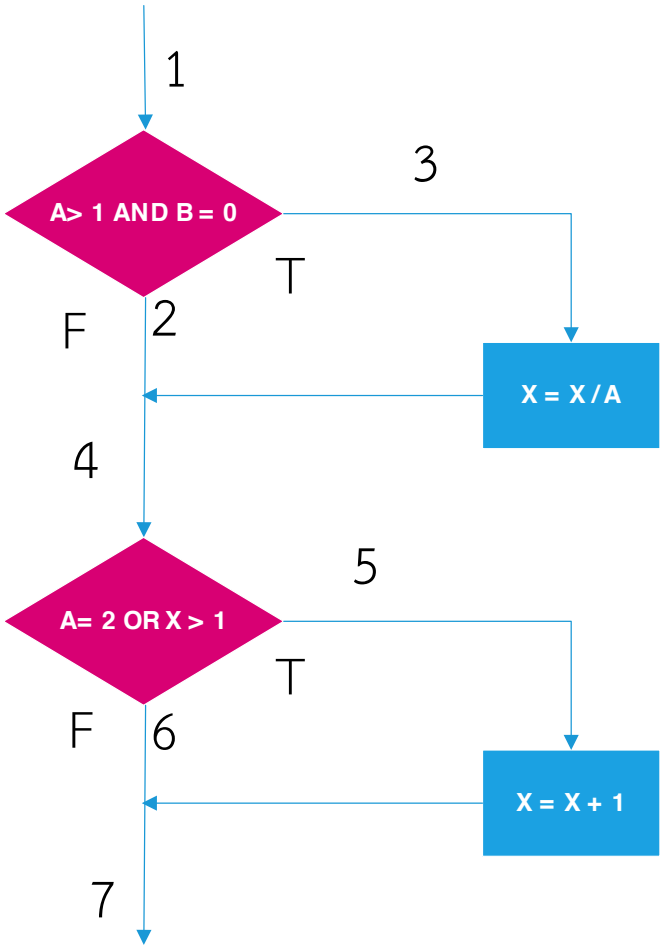
- บางครั้งจะถูกเรียกว่า Line Coverage
- เป็นวิธีที่ง่ายที่สุดในทางปฏิบัติ
- ในกรณีการทดสอบเพื่อให้ได้ผลลัพธ์ 100 เปอร์เซ็นต์ จะเกิดขึ้นเมื่อคำสั่งทุกบรรดทัด

ตัวอย่าง

```
IF (( A > 1) && (B == 0))
    X = X / A
IF (( A == 2) || (X > 1))
    X = X + 1
```

1 – 3 – 4 – 5 – 7

A = 2, B = 0 และ X = 4



ตัวอย่าง

```
int gcd(int x, int y){
 while (x != y){
    if (x>y) then
         x=x-y;
    else
         y=y-x;
 }
 return x;
}
```

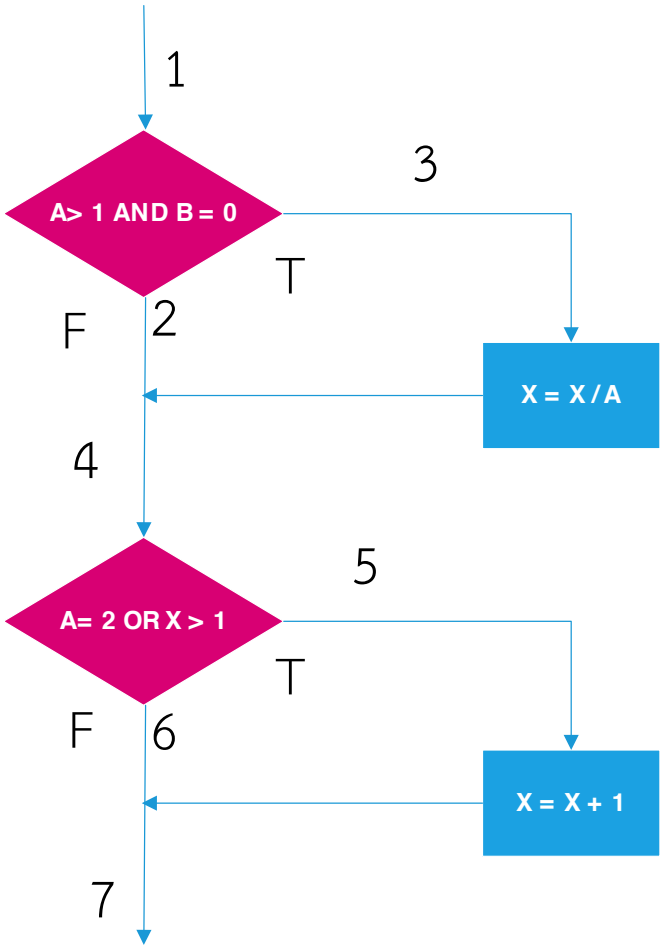
- เราจะเลือก test set {
- (x=3,y=3),
- (x=4,y=3),
- (x=3,y=4) }
- ทำให้ทุกบรรดทัดถูกประมวลผล 1 ครั้ง

การทดสอบความครอบคลุมระดับการตัดสินใจ (Branch Coverage)

- บางครั้งถูกเรียกว่า Decision Coverage
- เป็นการวัดความครอบคลุมที่ขึ้นกับผลลัพธ์ในการตัดสินใจ
- เพื่อให้แน่ใจว่ามีมีการทดสอบจะมีการประมวลผลทุกเงื่อนไขการตัดสินใจ
- ถูกพัฒนาขึ้นเนื่องมาจากข้อจำกัดของวิธีการทดสอบความครอบคลุมของคำสั่ง

ตัวอย่าง

```
IF (( A > 1) && (B == 0))
    X = X / A
IF (( A == 2) || (X > 1))
    X = X + 1
```



หาได้ดังนี้

จำนวนกรณีทดสอบทั้งหมด

1. A > 1 = T และ B = 0 = T
2. A > 1 = T และ B = 0 = F
3. A > 1 = F และ B = 0 = T
4. A > 1 = F และ B = 0 = F
5. A = 2 = T และ X > 1 = T
6. A = 2 = T และ X > 1 = F
7. A = 2 = F และ X > 1 = T
8. A = 2 = F และ X > 1 = F

กำหนดค่าข้อมูลให้ตอบสนองเงื่อนไขได้ดังนี้

- A=2, B =0, X =2 [1,5]
- A=2, B =1, X =2 [2,6]
- A=0, B =0, X =2 [3,7]
- A=0, B =1, X =0 [4,8]

เส้นทางที่ถูกทดสอบ

- 1 – 3 – 4 – 5 – 7
- 1 – 2 – 4 – 5 – 7
- 1 – 2 – 4 – 5 – 7
- 1 – 2 - 4 – 6 – 7

ตัวอย่าง

```
int gcd(int x, int y){
 while (x != y){
    if (x>y) then
         x=x-y;
    else
         y=y-x;
 }
 return x;
}
```

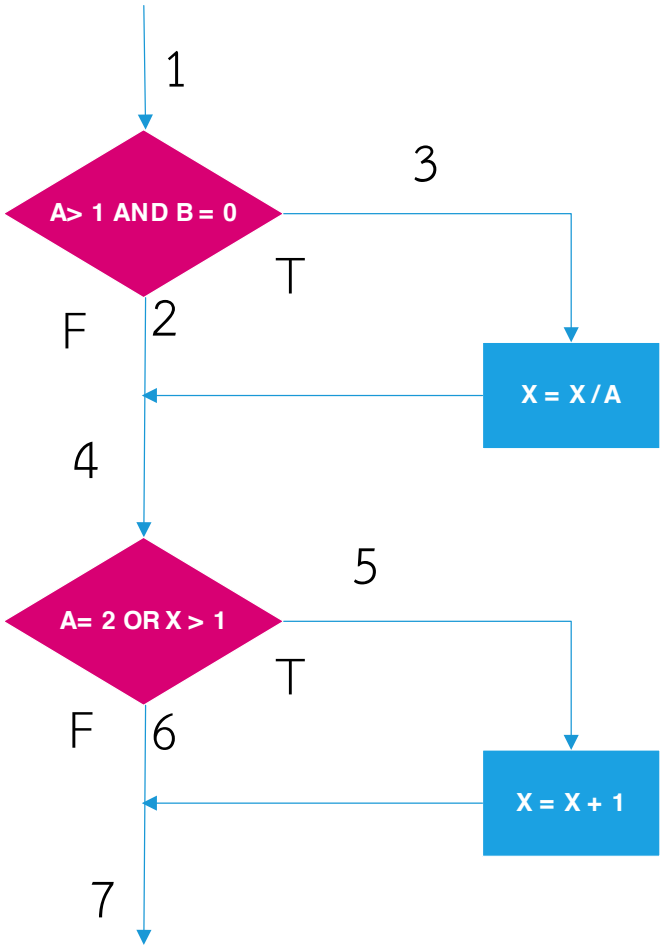
- เราจะเลือก test set {
- (x=3,y=3),
- (x=3,y=2),
- (x=4,y=3),
- (x=3,y=4) }
- ทำให้ทุกเงื่อนไขการตัดสินใจถูกประมวลผล 1 ครั้ง

การทดสอบความครอบคลุมระดับเส้นทางประมวลผล (Path Coverage)

- เป็นวิธีการทดสอบที่พิจารณาทุก ๆ ความเป็นไปได้ของเส้นทางประมวลผล
- เพื่อให้แน่ใจว่าทุก ๆ เส้นทางการทำงานที่ไม่ซ้ำของโปรแกรมมีการทดสอบอย่างน้อย 1 ครั้ง

ตัวอย่าง

```
IF (( A > 1) && (B == 0))
    X = X / A
IF (( A == 2) || (X > 1))
    X = X + 1
```



หาได้ดังนี้

จำนวนกรณีทดสอบทั้งหมด

1. 1 – 2 – 4 – 6 – 7 (FF)
2. 1 – 3 – 4 - 6 -7 (TF)
3. 1 – 2 – 4 -5 -7 (FT)
4. 1 – 3 – 4 – 5 – 7 (TT)

กำหนดค่าข้อมูลให้ตอบสนองเงื่อนไขได้ดังนี้

- A=0, B =1, X =0
- A=3, B =0, X =0
- A=2, B =1, X =0
- A=2, B =0, X =0

งาน

- ให้ทุกกลุ่มหาเงื่อนไขที่พบได้ทั่วไปในการทำงานเช่น
  - การตัดเกรด
  - โปรโมชั่นของโทรศัพท์มือมือ

- ทำการเขียนการทดสอบทั้ง 3 แบบ และออกมานำเสนอ

การทดสอบเส้นทางการประมวลผลหลัก (Basic Path Testing)

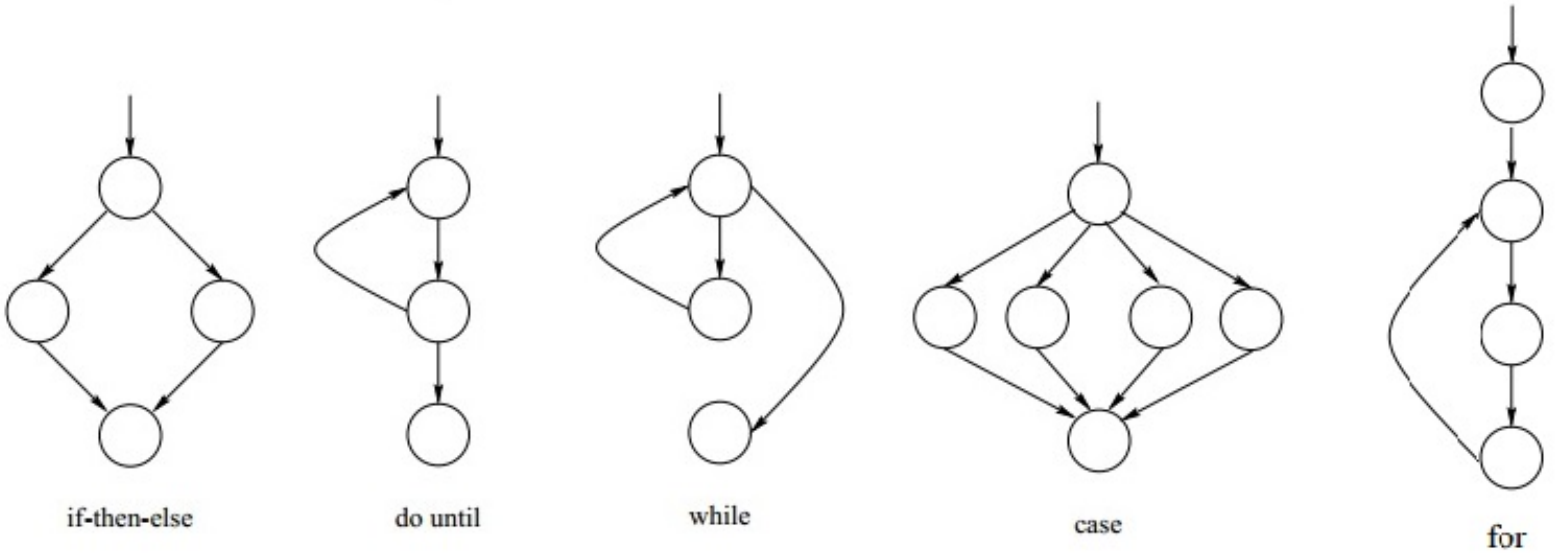
- เป็นวิธีที่ยอมให้นักทดสอบสามารถวัดความซับซ้อนทางลอจิกของการออกแบบโปรแกรมเชิงกระบวนการ
- มีการทำงานทั้งหมด 4 ขั้นตอน ดังนี้
  - สร้างกราฟกระแสควบคุม
  - คำนวณค่า Cyclomatic Complexity
  - เลือกกลุ่มของเส้นทางการประมวลผลหลัก
  - สร้างกรณีทดสอบเพื่อตรวจสอบแต่ละเส้นทาง

กราฟกระแสควบคุม (Control Flow Graph)

- ใช้นำเสนอกลไกการควบคุมภายในโปรแกรม ซึ่งสามารถนำไปใช้สร้างกรณีทดสอบที่ต้องการมีสัญลักษณ์ ดังนี้
  - สัญลักษณ์ลูกศรเรียกว่า edge ใช้สำหรับนำเสนอกลไกควบคุม
  - สัญลักษณ์วงกลมเรียกว่า node ใช้นำเสนอการกระทำตั้งแต่หนึ่งครั้งเป็นต้นไป
  - พื้นที่ที่ถูกล้อมโดย node และ edge จะเรียกว่า region
  - Node แบบ Predicate Node คือ node ที่มีเงื่อนไขอยู่ภายใน

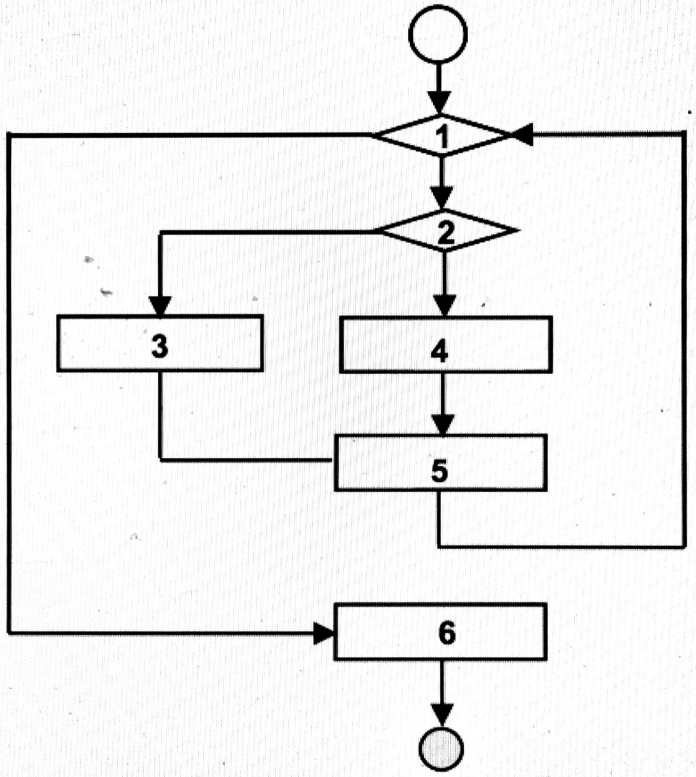
- ขั้นตอนการสร้างกราฟกระแสควบคุมจากโปรแกรมมีดังนี้
  - แยกมอดูลให้อยู่ในรูปแบบของบล็อคคำสั่งที่กระทบต่อกลไกควบคุม
  - กำหนดให้อยู่ในรูปแบบของโหนดภายในกราฟกระแสควบคุม
  - ลากเส้นเชื่อมต่อพร้อมลูกศรที่แสดงการทำงานทางลอจิก

การแทนที่ส่วนของโปรแกรมด้วยกราฟ

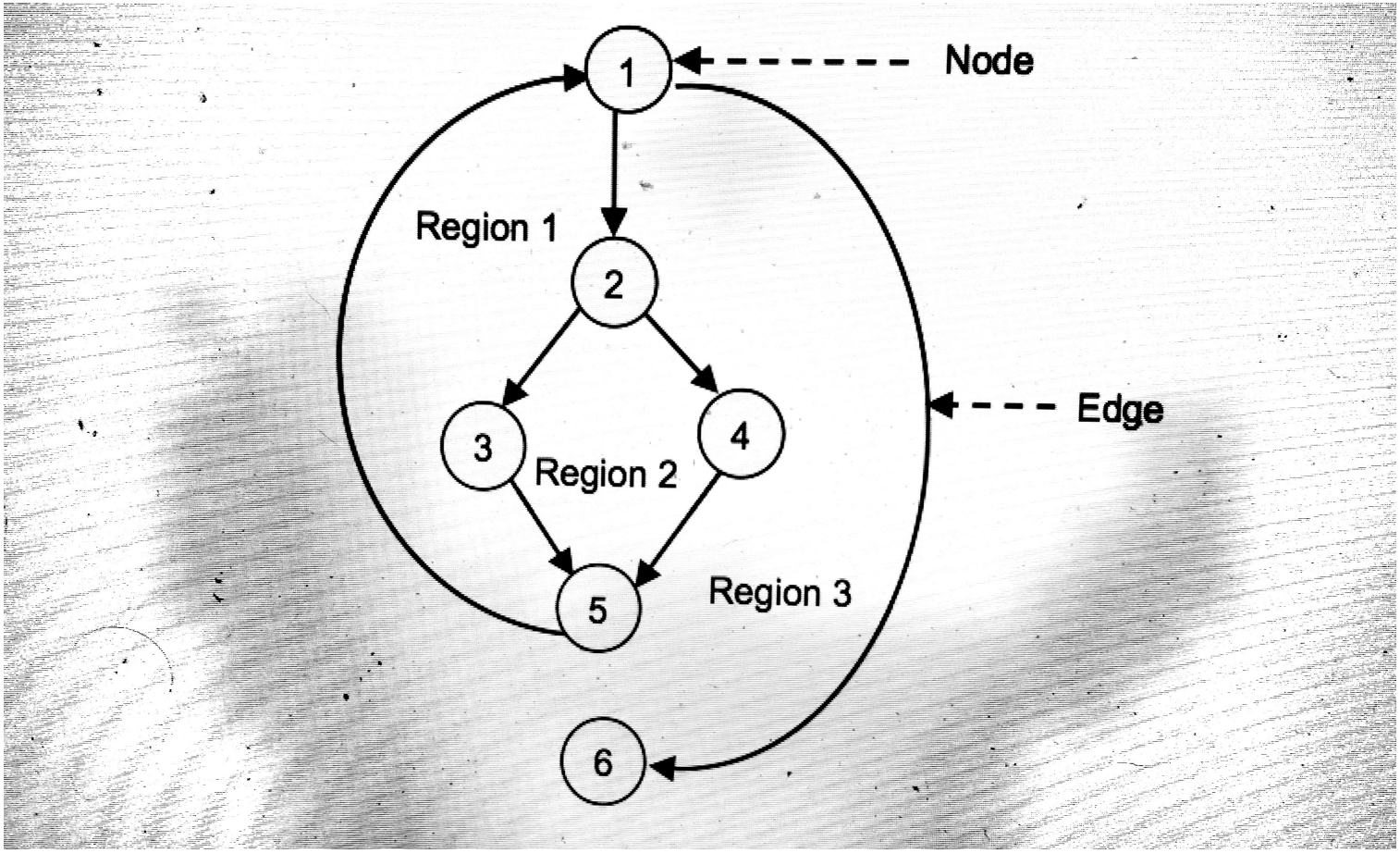


การแปลงจากโปรแกรมเป็นกราฟกระแสควบคุม

```
int f1 (int x, int y)
{
    1.  while (x != y){
    2.      if (x>y)
    3.          x=x-y;
    4.      else y=y-x;
    5.  }
    6.  return x;
}
```



การแปลงจากโปรแกรมเป็นกราฟกระแสควบคุม



หา Cyclomatic Complexity

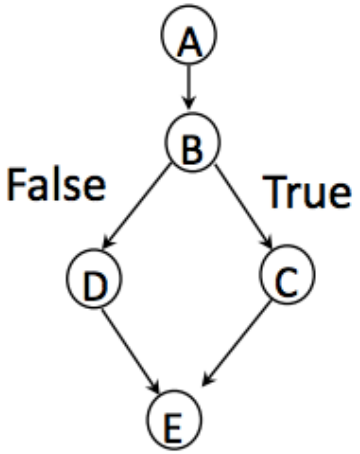
- การหาค่าความซับซ้อนของโปรแกรมโดยปกติทำได้สองวิธีคือ
  - V(g) = จำนวน regions ใน flow graph
  - V(g) = จำนวน edges – nodes + 2
- จากโปรแกรมก่อนหน้าเราสามารถหา Cyclomatic Complexity ได้ เท่ากับ
  - V(g) = regions = 3
  - V(g) = (7 – 6) + 2 = 3
- กำหนดเส้นทางการทำงานที่ไม่ซ้ำกัน
  - 1 - 6
  - 1 – 2 – 3 – 5 - 1 – 6
  - 1 – 2 - 4 – 5 – 1 - 6

สามารถสร้าง กรณีทดสอบได้ดังนี้

Test Id	Path	x	y	Expected result
1	1 - 6	1	1	1
2	1 – 2 – 3 – 5 - 1 – 6	1	2	1
3	1 – 2 - 4 – 5 – 1 - 6	2	1	1

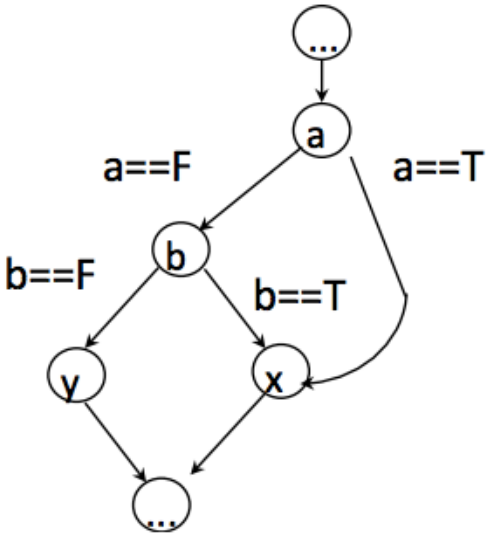
ตัวอย่าง แบบมี predicate node

```
/*A*/  float z = InOut.readFloat(),
          w = InOut.readFloat();
          t = InOut.readFloat();
/*B*/  if (z  == 0 || w > 0)
/*C*/    w = w/z;
/*D*/  else w = w + 2/t;
/*E*/  System.out.println(z+' '+w+' '+t);
```

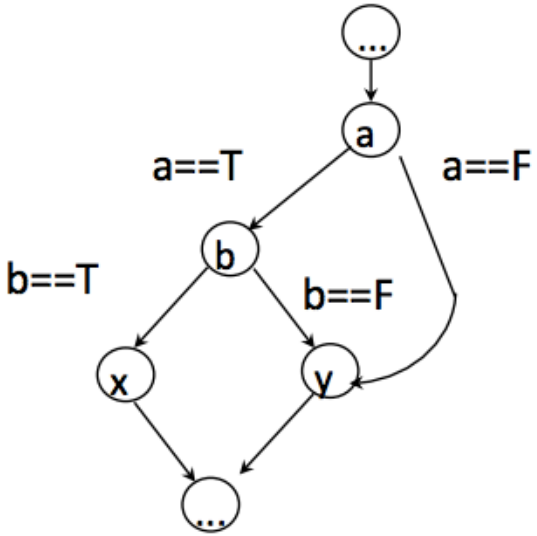


ต้องทำการแยกออกจากกันดังรูป

```
if (a || b)
x;
else y;
```

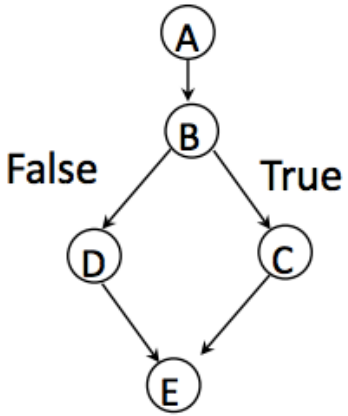


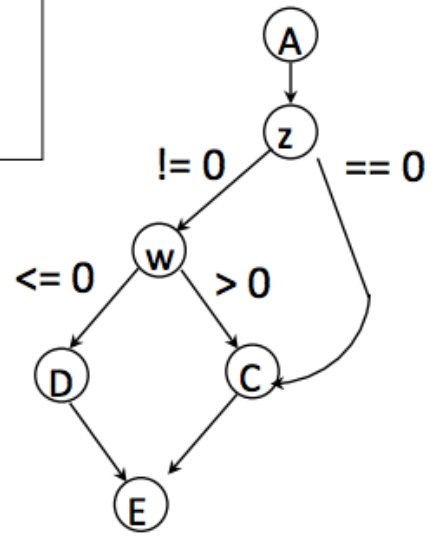

```
if (a && b)
x;
else y;
```



จะเห็นว่าเมื่อคำนวณความซับซ้อนแล้วจะได้ค่าไม่เท่ากัน

```
/*A*/  float z = InOut.readFloat(),
           w = InOut.readFloat();
           t = InOut.readFloat();
/*B*/  if (z  == 0 || w > 0)
/*C*/    w = w/z;
/*D*/  else w = w + 2/t;
/*E*/  System.out.println(z+' '+w+' '+t);
```

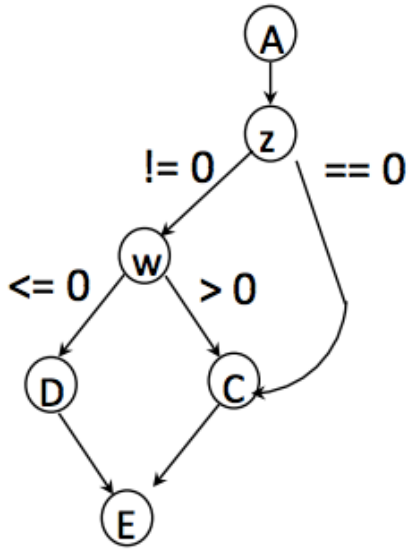




Control flow graph with conditions

ต้องใช้กรณีที่มีค่ามากกว่า

```
/*A*/  float z = InOut.readFloat(),
         w = InOut.readFloat();
         t = InOut.readFloat();
/*B*/  if (z  == 0 || w > 0)
/*C*/    w = w/z;
/*D*/  else w = w + 2/t;
/*E*/  System.out.println(z+' '+w+' '+t);
```



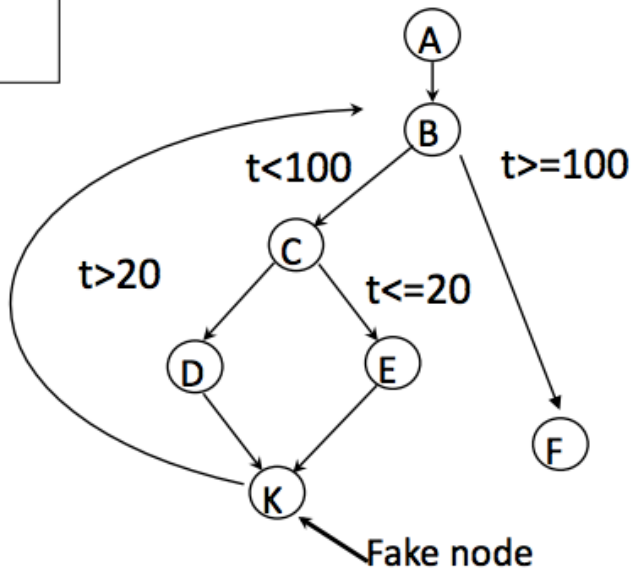
Three test cases
C2. {t=0; z=5; w= 5}
C3. {t=0; z=0; w= -5}
C4. {t=0; z=5; w=-5}
Cover all arcs, hence decisions and conditions

V=7-6+2=3
The three paths
1. A-z-w-C-E (C2)
2. A-z-C-E (C3)
3. A-z-w-D-E (C4)

กรณีโปรแกรมเขียนแบบลดรูปอาจต้องสร้าง node จำลอง

```
/*A*/  int z = InOut.readInt(); int t=0;
/*B*/  while (t<100)
/*C*/    if (t > 20)
/*D*/              t=t+90;
/*E*/        else  t= t+z*z;
/*F*/  System.out.println(t);
```

B-F
C-D
Require at least one iteration in order to be traversed
{10} A-B-C-E-K-B-F
{5} A-B-C-E-K-B-C-D-K-B-F



การทดสอบกระแสข้อมูล (Data Flow Testing)

- Next week

การทดสอบแบบ Black Box

เทคนิคการทดสอบแบบ Black Box

- เป็นการทดสอบฟังก์ชันการทำงานโดยไม่พิจารณาจากโครงสร้างของโปรแกรม
- กรณีทดสอบจะถูกสร้างจากความต้องการของระบบ หรือข้อกำหนดทางเทคนิคเป็นหลัก
- วิธีการทดสอบแบบ Black box ที่เป็นที่นิยมได้แก่
  - การแบ่งชนชั้นสมมูล (Equivalence Partition)
  - การวิเคราะห์ค่าขอบเขต (Boundary Value Analysis)
  - กราฟแสดงสาเหตุและผลกระทบ (Cause Effect Graph)

การแบ่งชนชั้นสมมูล

- จะทำการแบ่งข้อมูลออกตามลักษณะดังนี้
  - ข้อมูลที่เป็นตัวเลขภายในค่าพิสัย
  - ข้อมูลที่เป็นรายการภายในกลุ่ม
  - ข้อมูลที่เป็นตัวเลขเพียงค่าเดียว
  - ข้อมูลที่ถูกกำหนดให้อยู่ในสถานการณ์ที่ต้องเป็น
  - ข้อมูลที่มีวิธีจัดการแตกต่างกัน

ข้อมูลที่อยู่ในค่าพิสัย

- ให้ทำการกำหนดข้อมูล 1 ตัวให้อยู่ในค่าพิสัยที่ถูกระบบ
- กำหนดข้อมูล สองตัวที่มากกว่าค่าพิสัย และน้อยกว่าค่าพิสัย
- 0 – 100

ให้กำหนด 0 < Test Case < 100

100 < Test Case

Test Case < 0

ข้อมูลที่เป็นรายการภายในกลุ่ม

- ให้กำหนดข้อมูลที่อยู่ในกลุ่มทุกกลุ่ม
- กำหนดข้อมูลที่อยู่นอกกลุ่ม

เช่นการกำหนดราคาของสินค้า M = สมาชิกลดราคา

S = ลดปกติ

N = ไม่ลด

ให้กำหนดชุดข้อมูล M

S

N

และกลุ่มข้อมูลอื่น ๆ ที่ไม่ใช่ M,S,N

ข้อมูลที่เป็นตัวเลขเพียงค่าเดียว

- ให้กำหนดเป็นค่าที่ถูกต้องและค่าที่ ผิด

เช่น ค่าที่ได้จะต้องเป็น 0 เท่านั้น

ให้กำหนดที่จะเป็น 0

และไม่ใช่ 0

ข้อมูลที่ถูกกำหนดให้อยู่ในสถานการณ์ที่ต้องเป็น

- ให้กำหนดให้อยู่ในสถานการณ์ที่ถูกต้องและผิด

เช่น ข้อมูลต้องอยู่ในรูป M1234 คือตัวแรกต้องเป็นตัวอักษร

ให้กำหนดข้อมูลที่อยู่ในรูปแบบ Z00001

และ 123456

ข้อมูลที่มีวิธีจัดการแตกต่างกัน

- ให้กำหนดข้อมูลให้ครอบคลุมวิธีจัดการต่าง ๆ ให้แบ่งเป็นชนชั้นย่อย ๆ
- เช่น บัญชีธนาคารมีค่าได้ตั้งแต่ 0 ถึง 1000000 แต่ถ้ายอดเหลือ 1000 หรือมากกว่า ไม่ต้องเสียค่าบริการรายปีให้แบ่งชนชั้นย่อย

  0 <= ยอดเงิน < 1000

  1000 <= ยอดเงิน <= 10000000

  ยอดเงิน < 0

  ยอดเงิน > 1000000

ตัวอย่าง

- จงสร้างกรณีทดสอบจากโปรแกรมแสดงชนิดสามเหลี่ยมจากการรับค่าเลขที่เป็นจำนวนเต็ม 3 ค่าอยู่ในช่วง 1 - 200 โดยมีเงื่อนไขดังนี้
  - สามเหลี่ยมด้านเท่า เป็นสามเหลี่ยมที่มีด้านทั้ง 3 ด้านยาวเท่ากัน
  - สามเหลี่ยมหน้าจั่ว เป็นสามเหลี่ยมที่มีด้าน 2 ด้านยาวเท่ากัน
  - สามเหลี่ยมด้านไม่เท่า เป็นสามเหลี่ยมที่มีทั้งสามด้านยาวไม่เท่ากัน
  - ไม่เป็นสามเหลี่ยม
- กำหนดให้สัญลักษณ์ a b c แทนค่าด้านทั้งสามของสามเหลี่ยม ในกรณีที่เป็นสามเหลี่ยมด้านที่ 1 จะต้องน้อยกว่าด้านที่สองบวกด้านที่สามเสมอ แสดงว่าเป็นสามเหลี่ยม

การแบ่งชั้นสมมูล

Valid Input	Invalid Input
vEC1 = {<a, b, c> : a = b = c}	iEC1 = {<a, b, c> : a >= b +c}
vEC1 = {<a, b, c> : a = b, a != c}	iEC1 = {<a, b, c> : b >= a +c}
vEC1 = {<a, b, c> : a = c, a != b}	iEC1 = {<a, b, c> : c >= a +b}
vEC1 = {<a, b, c> : b = c, a != b}	
vEC1 = {<a, b, c> : a != b, a != c, b != c}	

แยกกรณีทดสอบส่วนอินพุตที่ไม่ถูกต้อง

- มีเครื่องหมาย >= ได้เป็น
- a >= b + c แยกเป็น a > b + c และ a = b + c
- b >= a + c แยกเป็น b > a + c และ b = a + c
- a >= b + c แยกเป็น c > a + b และ c = a + b

ได้กรณีทดสอบดังนี้

Test Case	a	b	c	Expected result
1	5	5	5	สามเหลี่ยมด้านเท่า
2	5	5	3	สามเหลี่ยมหน้าจั่ว
3	5	3	5	สามเหลี่ยมหน้าจั่ว
4	3	5	5	สามเหลี่ยมหน้าจั่ว
5	3	4	5	สามเหลี่ยมด้านไม่เท่า
6	8	3	4	ไม่เป็นสามเหลี่ยม
7	7	3	4	ไม่เป็นสามเหลี่ยม
8	3	8	4	ไม่เป็นสามเหลี่ยม
9	3	7	4	ไม่เป็นสามเหลี่ยม
10	3	4	8	ไม่เป็นสามเหลี่ยม
11	3	4	7	ไม่เป็นสามเหลี่ยม

การวิเคราะห์ค่าขอบเขต (Boundary Value Analysis)

- เป้าหมายคือการตรวจสอบของบริเวรขอบเขตของข้อมูลที่ต้องเรียกใช้และสร้างกรณีทดสอบจากตัวแปรนั้น
- เพื่อลดการหากรณีทดสอบถ้ามีข้อจำกัดด้านเวลา หรืองบประมาณ แต่อาจได้กรณีทดสอบที่ไม่ครบคลุม

เช่น ค่าต้องอยู่ในช่วง 10<= Value <=100

ให้ทดสอบดังนี้

Min 10 และ Max 100

- ให้ทดสอบโดย

  Min

  Min + 1

  Normal

  Max

  Max – 1

  จะได้กรณีทดสอบตามสูตร 4n + 1

ตัวอย่าง

- จงสร้างกรณีทดสอบจากโปรแกรมแสดงชนิดสามเหลี่ยมจากการรับค่าเลขที่เป็นจำนวนเต็ม 3 ค่าอยู่ในช่วง 1 - 200 โดยมีเงื่อนไขดังนี้
  - สามเหลี่ยมด้านเท่า เป็นสามเหลี่ยมที่มีด้านทั้ง 3 ด้านยาวเท่ากัน
  - สามเหลี่ยมหน้าจั่ว เป็นสามเหลี่ยมที่มีด้าน 2 ด้านยาวเท่ากัน
  - สามเหลี่ยมด้านไม่เท่า เป็นสามเหลี่ยมที่มีทั้งสามด้านยาวไม่เท่ากัน
  - ไม่เป็นสามเหลี่ยม
- กำหนดให้สัญลักษณ์ a b c แทนค่าด้านทั้งสามของสามเหลี่ยม ในกรณีที่เป็นสามเหลี่ยมด้านที่ 1 จะต้องน้อยกว่าด้านที่สองบวกด้านที่สามเสมอ แสดงว่าเป็นสามเหลี่ยม

การวิเคราะห์ ใช้ค่ากรณีทดสอบ

- Min = 1
- Min + 1 = 2
- Normal = 100
- Max – 1 = 199
- Max = 200

- จำนวนกรณีทดสอบสามารถหาได้จากสูตร 4n + 1
- 4(3) + 1 = 13 กรณี

ได้กรณีทดสอบดังนี้

Test Case	a	b	c	Expected result
1	100	100	1	สามเหลี่ยมหน้าจั่ว
2	100	100	2	สามเหลี่ยมหน้าจั่ว
3	100	100	100	สามเหลี่ยมด้านเท่า
4	100	100	199	สามเหลี่ยมหน้าจั่ว
5	100	100	200	ไม่เป็นสามเหลี่ยม
6	100	1	100	สามเหลี่ยมหน้าจั่ว
7	100	2	100	สามเหลี่ยมหน้าจั่ว
8	100	199	100	สามเหลี่ยมหน้าจั่ว
9	100	200	100	ไม่เป็นสามเหลี่ยม
10	1	100	100	สามเหลี่ยมหน้าจั่ว
11	2	100	100	สามเหลี่ยมหน้าจั่ว
12	199	100	100	สามเหลี่ยมหน้าจั่ว
13	200	100	100	ไม่เป็นสามเหลี่ยม