บทที่ 2

ทฤษฎีการวัดความซับซ้อนของซอฟต์แวร์

ทฤษฎีการวัดความซับซ้อนของซอฟต์แวร์เกิดขึ้นในปี 1972 โดย ศาสตราจารย์มอไรส์ ฮอลสตีด (Maurice Halstead) นักวิทยาศาสตร์ชาวอเมริกันผู้ได้ชื่อว่าเป็นบิดาแห่งศาสตร์แขนงวิทยาการซอฟต์แวร์ (Software Science) ฮอลสตีดตั้งสมมติฐานว่าจำนวนจุดผิดในโปรแกรมมีความสัมพันธ์กับจำนวนตัวถูกดำเนินการ (operand) และจำนวนตัวดำเนินการ (operator) ในโปรแกรม ข้อสมมติฐานนี้ได้รับการพิสูจน์ โดยการวัดค่าจุดผิดเปรียบเทียบกับค่าจำนวนตัวดำเนินการ และตัวถูกดำเนินการ ซึ่งผลการทดลองส่วนมากเป็นที่ยอมรับได้ว่าถูกต้อง

แม้ว่าแนวความคิดของฮอลสตีดจะมีจุดอ่อนอยู่มาก แต่ตัววัดของฮอลสตีดก็ทำให้เกิดสาขาย่อยของวิทยาการคอมพิวเตอร์ที่ชื่อตัววัดความซับซ้อนของซอฟต์แวร์ขึ้น หลังจากจากนั้นก็มีนักวิชาการอีกหลายคนพยายามพิสูจน์หรือหักล้างสูตรที่ฮอลสตีดนิยามขึ้นมา หรือพยายามเสนอสูตรหรือแนวทางใหม่ และทำการพัฒนาตัววัดของขึ้นเองอีกหลายแบบที่คิดว่าสามารถหาความซับซ้อนของโปรแกรมได้ดีขึ้น

ในบทนี้จะกล่าวถึงทฤษฎีการวัดความซับซ้อนบางทฤษฎีที่น่าสนใจ ดังนี้

2.1 ตัววัดฮอลสตีด (Halstead's Metrics)

วิธีของฮอลสตีด ส่วนใหญ่จะได้จากการพิจารณาตัวโปรแกรมที่เขียนเป็นหลัก โดยทำการนับตัวดำเนินการและตัวถูกดำเนินการ แล้วนำมาเข้าสูตรคำนวณหาค่าความซับซ้อน แต่วิธีการนับนั้นยังไม่มีมาตรฐาน แตกต่างกันไปตามนักวิจัยแต่ละคนและภาษาที่ใช้เขียนโปรแกรม เช่น คำสั่งส่วนรับข้อมูลเข้าและส่งข้อมูลออก (I/O) หรือพวกคำหลัก (keyword) ซึ่งอาจนับหรือไม่นับขึ้นกับนักวิจัยแต่ละคน และวัตถุประสงค์ในการใช้งาน

วิธีการวัดของฮอลสตีด มีพารามิเตอร์ที่เกี่ยวข้องดังนี้

ก) จำนวนของตัวดำเนินการ (n_1) ทั้งหมดในโปรแกรมที่ไม่ซ้ำกัน ตัวดำเนินการ ได้แก่ชื่อของฟังก์ชั่น (function name) และพวกอักขระคั่น (delimiter) ต่างๆ ซึ่งจะแตกต่างกันไปตามภาษาที่ใช้เขียนโปรแกรม นักวิจัยบางท่านจะรวมพวกคำหลักเป็นตัวดำเนินการแบบหนึ่งด้วย สำหรับอักขระคั่นและคำหลักของภาษาซีทั้งหมดแสดงในตารางที่ 2.1 และรูปที่ 2.1

ประเภทของอักขระคั่น	โทเกน
แบบธรรมดา	! % ^ & * - + = ~ ¦ . < > / ?
กำหนดค่าแบบผสม	+= -= *= /= %= <<= >>= &= ^= ¦=
แบบผสมอื่นๆ	-> ++ -- << >> <= >= == != && ¦¦
อื่นๆ	( ) [ ] { } , ; :

ตารางที่ 2.1 แสดงอักขระคั่นของภาษาซี

asm	default	float	register	switch
auto	do	for	return	typedef
break	double	fortran	short	union
case	else	goto	signed	unsigned
char	entry	if	sizeof	void
const	enum	int	static	volatile
continue	extern	long	struct	while

รูปที่ 2.1 แสดงคำหลักของภาษาซี

ข) จำนวนของตัวถูกดำเนินการ (n_2) ทั้งหมดในโปรแกรมที่ไม่ซ้ำกัน ตัวถูกดำเนินการในที่นี้หมายถึงตัวแปรและค่าคงที่ ที่ใช้ในโปรแกรม

ค) จำนวนของตัวดำเนินการทั้งหมดในโปรแกรม (N1)

ง) จำนวนของตัวถูกดำเนินการทั้งหมดในโปรแกรม (N2)

```
void QUAD (float a, float b, float c, int x, int y)
{
      x = b * 2 - a * c; a = 2 * a;
      if (x >= 0) {
         x = sqrt(x);  y = (x - b) * a; x = (-x - b) / a;
      } else {
         printf("No real solution");
         x = 0, y = 0;
      }
}
```

รูปที่ 2.2 แสดงตัวอย่างโปรแกรมภาษาซี

จากโปรแกรมในรูปที่ 2.2 การหาค่าพารามิเตอร์ของฮอลสตีดทั้ง 4 ตัว สรุปได้ดังนี้ โทเกนสองตัวแรก คือ void QUAD ซึ่งเป็นประเภทและชื่อของฟังก์ชั่นจะไม่นับเป็นค่าใดๆ ตัวต่อมาเป็นพารามิเตอร์ของฟังก์ชั่นคือ (float a,float b,float c,int x,int y) ในส่วนนี้ตัวถูกดำเนินการได้แก่ a b c x y ส่วนคำหลัก float และ int จะไม่นับ โทเกนต่อมาเป็นตัวโปรแกรมซึ่งมีตัวดำเนินการคือ = - * / ( ) >= sqrt printf และมีตัวถูกดำเนินการคือ a b c x y 0 2 "No real solution" สำหรับค่าทั้งหมดจะสรุปในตารางที่ 2.2

ตัวดำเนินการ	นับ	ตัวถูกดำเนินการ	นับ
=	7	a	6
-	4	b	4
*	4	c	2
/	1	x	9
()	6	y	3
>=	1	0	3
,	5	2	2
sqrt	1	"No real solution"	1
printf	1		
$n_1=9$	$N_1=30$	$n_2=8$	$N_2=30$

ตารางที่ 2.2 สรุปการนับค่าพารามิเตอร์ของฮอลสตีด

ฮอลสตีดเสนอสูตรการวัดจากพารามิเตอร์ทั้ง 4 ไว้ดังนี้

2.1.1 ความยาวโปรแกรม (Length) ฮอลสตีดตั้งข้อสังเกตว่าองค์ประกอบหนึ่งซึ่งทำให้โปรแกรมมีความซับซ้อนคือ ความยาวของโปรแกรม โปรแกรมที่ยาวมากย่อมมีแนวโน้มที่จะซับซ้อนมากกว่าโปรแกรมที่สั้น สำหรับโปรแกรมใดโปรแกรมหนึ่งเมื่อเรานิยามเป็นตัวถูกดำเนินการ และตัวดำเนินการแล้ว ก็จะนับจำนวนตัวถูกดำเนินการทั้งหมด (N_2) และจำนวนตัวดำเนินการทั้งหมด (N_1) เพื่อคำนวณขนาดของโปรแกรม (N) ได้จาก

$$N = N_1 + N_2$$

โดยที่ N คือ ขนาดของโปรแกรม

N_1 คือ จำนวนตัวดำเนินการทั้งหมด

N_2 คือ จำนวนตัวถูกดำเนินการทั้งหมด

ความยาวโปรแกรมของฮอลสตีด อาจประมาณได้จากสูตร

$$\hat{N} = n_1 * \log_2 n_1 + n_2 * \log_2 n_2$$

โดยที่ $\hat{N}$ คือ ค่าประมาณของความยาวโปรแกรม

n_1 คือ จำนวนชนิดของตัวดำเนินการ

n_2 คือ จำนวนชนิดของตัวถูกดำเนินการ

สูตรนี้ได้รับการพิสูจน์โดยการเปรียบเทียบค่า N กับค่า $\hat{N}$ ซึ่งเป็นค่าประมาณของค่า N ปรากฏว่าสูตรนี้ให้ผลลัพธ์ที่ถูกต้องใกล้เคียงมากถ้าโปรแกรมมีความยาวอยู่ระหว่าง 2000 ถึง 4000 บรรทัด (สุชาย ธนวเสถียร, 2533) เมื่อโปรแกรมเป็นภาษาชั้นสูง เช่น ซี ฟอร์แทรน โคบอล หรือปาสคาล

ค่า n_1 อาจประเมินได้จากไวยากรณ์ของภาษาที่ใช้ ส่วน n_2 หาได้จากจำนวนตัวแปรรับข้อมูลเข้า จำนวนตัวแปรส่งข้อมูลออก และจำนวนตัวแปรชั่วคราวที่ใช้ในการสร้างโปรแกรมนั้นซึ่งทำให้ประมาณค่าขนาดโปรแกรมได้

2.1.2 ปริมาตร (Volume) หมายถึง จำนวนบิตที่ต้องใช้เพื่อแทนโปรแกรมนั้น เช่น ถ้าโปรแกรมมีความยาว N=4 เราสามารถแทนโปรแกรมนั้นด้วย 8 บิต โดยให้ใช้คำละ 2 บิต (00, 01,10, 11) ดังนั้นถ้าโปรแกรมมีความยาว N และแต่ละคำในโปรแกรมนั้นต้องใช้ 6 บิต ปริมาตรโปรแกรมนั้นจะเป็น 6 * N

จากค่า n_1 และ n_2 เราทราบว่าโปรแกรมหนึ่งจะประกอบด้วยค่าหรือสัญญลักษณ์ที่แตกต่างกันไป n_1+n_2 ชนิด ดังนั้นแต่ละคำจะสามารถแทนด้วย $\log_2(n_1+n_2)$ บิต เช่น $n_1+n_2 = 16$ หรือมี 16 สิ่งแต่ละสิ่งก็สามารถแทนด้วย $\log_2 16$ หรือ 4 บิต โดยทั่วไปแล้ว เราจะนิยามปริมาตรว่า

$$V = N * \log_2(n_1 + n_2)$$

โดยที่ V คือ ปริมาตรของโปรแกรม

N คือ ความยาวของโปรแกรม

n_1 คือ จำนวนชนิดของตัวดำเนินการ

n_2 คือ จำนวนชนิดของตัวถูกดำเนินการ

เนื่องจากการคำนวณแบบหนึ่ง เมื่อเขียนโปรแกรมสามารถเขียนด้วยภาษาได้หลายภาษา ตั้งแต่ภาษาชั้นต่ำเช่นแอสเซมบลีไปจนถึงภาษาชั้นสูงอย่างปาสคาล ซึ่งการใช้ภาษาไม่เหมือนกัน ก็จะทำให้ปริมาตรต่างกันไปด้วย

2.1.3 ปริมาตรศักยะ (Potential Volume) ถ้าเราสามารถเขียนโปรแกรมสั้นที่สุด สำหรับแบบการคำนวณที่กำหนดให้เราจะได้โปรแกรมที่มีตัวดำเนินการเพียงตัวเดียวที่เปลี่ยนค่าอินพุตให้ได้ค่าเอาต์พุตเลย โดยทั่วไปแล้วสูตรจะเป็น

$$V^{*} = (2 + n_2^{*}) * \log_2(2 + n_2^{*})$$

โดยที่ V^{*} คือ ปริมาตรศักยะ

n_2^{*} คือ จำนวนตัวแปรรับข้อมูลเข้าและส่งข้อมูลออกเท่านั้น

2.1.4 ระดับของการโปรแกรม (Program Level) การเขียนโปรแกรมนั้นปริมาตรของภาษาชั้นสูงจะน้อยกว่าของภาษาชั้นต่ำ แต่ระดับของความเป็นนามธรรมของโปรแกรมภาษาชั้นต่ำจะน้อยกว่าภาษาชั้นสูง เนื่องจากว่าการใช้ภาษาระดับต่ำนั้นเราใช้ตัวดำเนินการที่พื้นฐานกว่า หรืออีกนัยหนึ่งเมื่อปริมาตรของโปรแกรมจะลดลงในขณะที่ระดับของโปรแกรมเพิ่มขึ้น และปริมาตรจะเพิ่มขึ้นเมื่อระดับของการโปรแกรมลดลง ดังนั้นฮอลสตีดจึงตั้งความสัมพันธ์ว่า

$$L * V = \text{ค่าคงที่} = V^{*}$$

โดยที่ L คือ ระดับของการโปรแกรมโดยจะต้องมีค่า $<= 1$

V คือ ปริมาตรของโปรแกรม

การประมาณค่า L โดยทั่วไปแล้วหาได้จากสูตร

$$\hat{L} = \frac{2}{n_1} * \frac{n_2}{N_2}$$

โดยที่ $\hat{L}$ คือ ค่าประมาณของค่าระดับของการโปรแกรม

N_2 คือ จำนวนของตัวถูกดำเนินการทั้งหมด

n_1 คือ จำนวนชนิดของตัวดำเนินการ

n_2 คือ จำนวนชนิดของตัวถูกดำเนินการ

สูตรนี้ได้จากแนวความคิดที่ว่า ระดับของการโปรแกรมน่าจะลดลงถ้าจำนวนของตัวดำเนินการและตัวถูกดำเนินการทั้งหมดเพิ่มขึ้น นอกจากนี้ระดับของการโปรแกรมยังควรจะเพิ่มขึ้นถ้ามีจำนวนชนิดตัวถูกดำเนินการต่างๆเพิ่มขึ้น ค่าประมาณ $\hat{L}$ ปกติจะสูงกว่าค่า L จริง (หาจาก V^*/N) ประมาณ 18 %

ฮอลสตีดเรียกส่วนกลับของ L ว่า ค่าความยาก (Difficulty) โปรแกรมเดียวกันที่เขียนด้วยภาษาชั้นต่ำจะมีค่าความยากมากกว่าโปรแกรมที่เขียนด้วยภาษาชั้นสูง

2.1.5 ระดับภาษา (α) ที่ใช้ในการเขียนโปรแกรมมีขีดจำกัดของตัวเอง ฮอลสตีดตีความว่าค่า $L * V^*$ จะมีค่าต่างกันไปตามภาษาที่ใช้เขียนโปรแกรม ค่า $L * V^*$ จึงอาจใช้เป็นตัวที่บ่งถึงระดับของภาษาโปรแกรม

$$\alpha = L * V^* = L * (L * V) = L^2 * V$$

โดยที่ α คือ ระดับภาษา

L คือ ระดับของการโปรแกรม

V คือ ปริมาตรของโปรแกรม

ระดับภาษาบ่งบอกถึงว่า ภาษาชั้นสูงมีระดับสูงกว่าภาษาชั้นต่ำ ค่า α นี้ถือว่าเป็นค่าคงที่สำหรับภาษาใดภาษาหนึ่ง แต่ตามความเป็นจริงแล้วอาจไม่คงที่จริงๆเป็นเพียงค่าคงตัวทางสถิติ ซึ่งไม่ขึ้นกับคนเขียนโปรแกรม และไม่ขึ้นกับชนิดของปัญหา

2.1.6 ความล้ำลึกของขั้นตอนกรรมวิธี (I) ฮอลสตีดนิยามไว้ว่า

$$I = L * V$$

โดยที่ I คือ ความล้ำลึกของขั้นตอนกรรมวิธี

L คือ ระดับของการโปรแกรม

V คือ ปริมาตรของโปรแกรม

ค่า I สำหรับขั้นตอนกรรมวิธีหรือแบบฉบับการคำนวณแบบหนึ่ง เมื่อสร้างด้วยภาษาใดก็ตามจะมีค่า 14 ถึง 16 (สุชาย ธนวเสถียร, 2533) ตามข้อสรุปของฮอลสตีด ซึ่งข้อสรุปนี้ต่อมาก็ไม่ได้รับการยืนยันจากนักวิจัยคนอื่นๆ ว่าเป็นค่าเกือบคงที่

2.1.7 ความพยายาม (E) หมายถึง ค่าที่แสดงถึงความพยายามของจิตใจในการเขียนโปรแกรมนั้นๆ หรือแปลความหมายได้อีกอย่างหนึ่งว่า เป็นความพยายามที่ต้องใช้สำหรับการอ่านและการเข้าใจโปรแกรม (เอื้อน ปิ่นเงิน, 2535) ค่าความพยายาม E นี้ ฮอลสตีดนิยามว่า

$$E = V^2 / V^*$$

โดยที่ E คือ ค่าความพยายาม

V คือ ปริมาตรของโปรแกรม

V^* คือ ปริมาตรศักยะ

ซึ่งฮอลสตีดตีความต่อไปว่า ค่าที่ได้คือจำนวนรูปแบบความคิดที่ผู้เขียนใช้ในการพัฒนาโปรแกรม หรืออีกนัยหนึ่งก็อาจเป็นจำนวนความคิดที่ใช้ในการเข้าใจโปรแกรมเรื่องนี้ ฮอลสตีดได้ยึดแนวความคิดจากแขนงวิชา cognitive psychology มาใช้ว่าถ้า S เป็นค่าเฉลี่ยของจำนวนความคิดในเชิงจิตวิชาที่คนมีได้ใน 1 วินาที แล้วเวลาที่ใช้ในการเข้าใจโปรแกรมจะเป็น

$$T = E / S$$

โดยที่ T คือ เวลาที่ใช้ในการเข้าใจโปรแกรม

E คือ ค่าความพยายาม

S คือ ค่าเฉลี่ยของจำนวนความคิดใน 1 วินาที

ค่า S นี้หาได้จากงานของสตราวด์ (Stroud, 1950) ว่า คนเราจะมองเห็นและรู้สึก 5 ถึง 10 ครั้งใน 1 วินาที (psychological time) การทดลองของสตราวด์ ไม่ได้ทดลองถึงการเปลี่ยนความคิดในขบวนการที่ซับซ้อนอย่างเขียนโปรแกรม จุดนี้ฮอลสตีดถูกโจมตีอย่างมากจากนักวิจัยคนอื่นๆ ว่าเอาแนวความคิดไปใช้อย่างผิดประเภท

ตัววัดของฮอลสตีดทั้ง 7 แบบต่างอาศัยการนับคำเพียงอย่างเดียว โดยไม่คิดความซับซ้อนของการใช้คำสั่งประเภทตัดสินใจ (if) หรือประเภทการวนซ้ำ (loop) หรือการซ้อนใน (nesting) แต่อย่างใด อาจถือเป็นข้อบกพร่องของตัววัดชุดนี้ ผลการคำนวณหาค่าความซับซ้อนของฮอลสตีดทั้ง 7 แบบจากโปรแกรมในรูปที่ 2.2 มีดังนี้

ความยาวโปรแกรม ($\hat{N}$) $= n_1 * \log_2(n1) + n_2 * \log_2(n_2)$

$= 9 * \log_2(9) + 8 * \log_2(8)$

$= 52.53$

ปริมาตร (V) $= N * \log_2(n_1 + n_2)$

$= 52.53 * \log_2(9 + 8)$

$= 214.71$

ปริมาตรศักยะ (V^*) $= (2 + n_2^*) * \log_2(2 + n_2^*)$

$= (2 + 5) * \log_2(2 + 5)$

$= 23.25$

ระดับของการโปรแกรม ($\hat{L}$) $= (2 * n_2) / (n_1 * N_2)$

$= (2 * 8) / (9 * 30)$

$= 0.06$

ระดับภาษา (α) $= L^2 * V$

$= (0.06)^2 * 214.71$

$= 0.75$

ความล้ำลึกของขั้นตอนกรรมวิธี (I) $= L * V$

$= 0.06 * 214.71$

$= 12.72$

ค่าความพยายาม (E) $= V^2 / V^*$

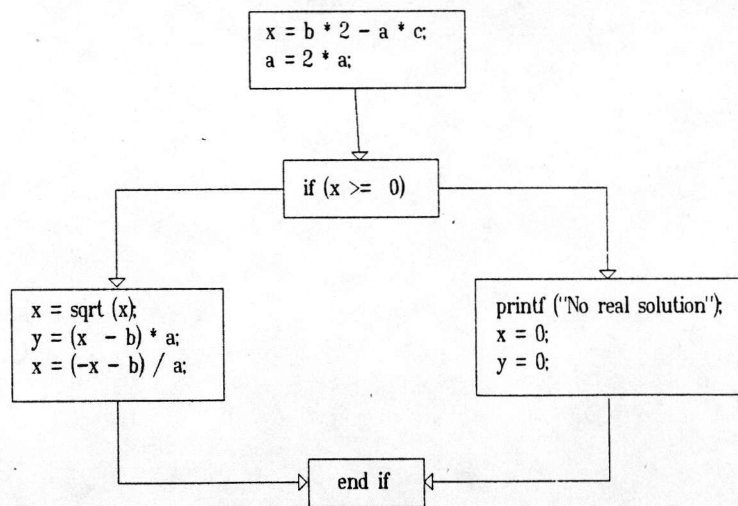
$= (214.71)^2 / 23.25$

$= 1982.54$

2.2 ตัววัดไซโคลแมติกของแมคเคบ (McCabe's Cyclomatic Metrics)

แมคเคบเสนอความคิดที่น่าสนใจว่า ความยากในการเข้าใจโปรแกรมที่แท้จริงแล้ว ขึ้นอยู่กับกราฟกระแสการควบคุม (CFG) ของโปรแกรมนั้นๆ ตัววัดของแมคเคบหรือที่เรียกกันว่าตัวเลขไซโคลแมติกนี้เป็นที่ยอมรับกันทั่วไปว่ามีความน่าเชื่อถือสูง

กราฟกระแสการควบคุมคือการแทนโปรแกรมด้วยโหนด (node) และเส้นเชื่อม (edge) โดยโหนดคือกลุ่มของคำสั่ง ซึ่งการแยกคำสั่งต่างๆในโปรแกรมออกเป็นโหนดทำได้โดยพิจารณา ถ้าคำสั่งแรกในกลุ่มถูกประมวลผลแล้ว คำสั่งทุกๆคำสั่งในกลุ่มนั้นต้องถูกประมวลผลด้วยและคำสั่งภายในโหนดจะต้องไม่มีทางเข้าไปโดยตรงยกเว้นคำสั่งแรกของโหนด ส่วนเส้นเชื่อมคือเส้นต่อระหว่างโหนดแต่ละโหนด และบอกถึงลำดับก่อนหลังของการประมวลผลแต่ละโหนดนั่นเอง ในกราฟกระแสการควบคุมใดๆ จะมีโหนดแรกที่มีแต่ทางออกหนึ่งหรือสองทางถ้ามีสองทางแสดงว่าโหนดนั้นแทนคำสั่งที่เป็นประเภทการตัดสินใจเช่น if หรือ while ส่วนโหนดอื่นๆนั้นจะมีทางเข้าและออกหนึ่งหรือสองทาง โหนดสุดท้ายของกราฟกระแสการควบคุมจะมีแต่ทางเข้าไม่มีทางออก (Conte et al., 1987) รูปที่ 2.3 แสดงตัวอย่างของ CFG จากโปรแกรมในรูปที่ 2.2



รูปที่ 2.3 แสดง CFG ของโปรแกรม

ถ้าให้ m แทนตัวเลขไซโคลแมติก เราจะหา m ได้ดังนี้

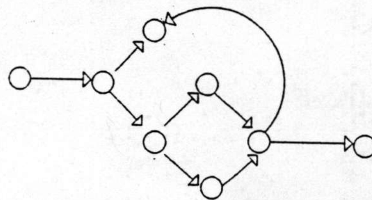
$$m = e - n + 2$$

โดยที่ e คือ จำนวนเส้นเชื่อม (edge) ของกราฟ

n คือ จำนวนโหนด (node) ของกราฟ

สำหรับโปรแกรมที่มีทางเข้า 2 ทาง ทางออก 1 ทาง m จะเท่ากับจำนวนคำสั่งแบบตัดสินใจ (if) บวก 1 หรือถ้าดูตาม CFG แล้วจะเท่ากับจำนวนโหนดตัดสินใจ (มีทางออก 2 ทาง) บวก 1 ตัวอย่างการหาค่าไซโคลแมติกจาก CFG แสดงในรูปที่ 2.4

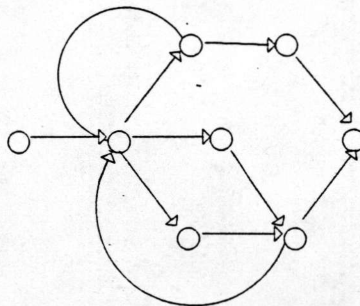
(ก)



$e = 9 \quad n = 8$

$m = 9 - 8 + 2 = 3$

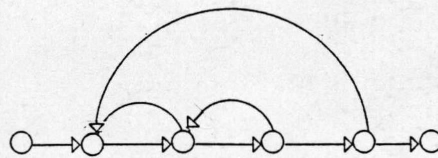
(ข)



$e = 11 \quad n = 8$

$m = 11 - 8 + 2 = 5$

(ค)



$e = 8 \quad n = 6$

$m = 8 - 6 + 2 = 4$

รูปที่ 2.4 แสดงการหาจำนวณค่าไซโคลแมติกจาก CFG

019122

จากตัวอย่างเห็นได้ชัดว่าตัวเลข m นี้ ขึ้นกับโครงสร้างของการตัดสินใจของโปรแกรมซึ่งมีผลต่อความถูกผิดของโปรแกรม โดยทั่วไปแล้วเราบอกได้ว่าถ้าค่า m สูง โปรแกรมนั้นจะซับซ้อนโอกาสผิดจะมีได้มาก นอกจากนั้นเวลาที่ใช้เขียนหรือออกแบบจะต้องนานขึ้นด้วย ผลคือค่าใช้ในการพัฒนาโปรแกรมสูงขึ้นด้วย ถ้าเราใช้ระบุผลงานของผู้เขียนโปรแกรมสองคนที่ออกแบบและเขียนโปรแกรมเดียวกัน สมมติว่าทั้งสองคนเขียนโปรแกรมถูกต้องตามต้องการทั้งคู่ แต่คนแรกได้โปรแกรมที่มีค่า m สูงกว่าคนที่สองอาจกล่าวได้ว่าโปรแกรมของคนที่สองควรจะดีกว่าคนแรก

มีข้อสังเกตเกี่ยวกับวิธีการของแมคเคบที่น่าสนใจดังนี้

ก. โปรแกรมที่เป็นโครงสร้าง m จะมีค่าเท่ากับจำนวนเงื่อนไข (predicate) บวกด้วย 1

ข. ถ้า G เป็นกราฟในระนาบ (planar graph) แล้ว m จะมีค่าเท่ากับจำนวนระนาบของ G

ค. ค่าสูงสุดของ m ที่ แมคเคบแนะนำคือ แต่ละโมดูลไม่ควรมีค่า m เกิน 10 ยกเว้นกรณีที่ใช้คำสั่ง case ที่มีเงื่อนไขในการเลือกที่เป็นอิสระต่อกัน โดยทั่วไปแล้วถ้าค่าของ m อยู่ระหว่าง 3-7 จะถือว่าโปรแกรมนั้นมีค่าความซับซ้อนที่เหมาะสม (เอื้อน ปิ่นเงิน, 2535)

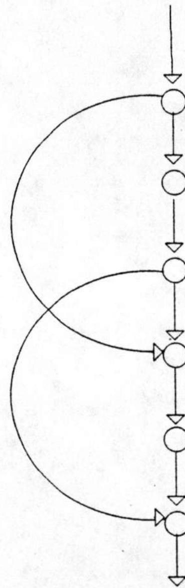
ง. แมคเคบพบว่า ค่าไซโคลแมติก ระดับความง่ายในการทดสอบ และความเชื่อถือได้ของโปรแกรม มีความสัมพันธ์กันสูง หมายความว่า โปรแกรมที่มีค่า m น้อยจะสามารถทดสอบได้ง่าย และมีความน่าเชื่อถือสูง

ตัววัด m นี้ ความจริงไม่ได้บอกถึงขนาดของโปรแกรมและบอกไม่ได้ถึงความซับซ้อนของการขึ้นแก่กันระหว่างข้อมูล (data dependency) คือโปรแกรมสองโปรแกรมโปรแกรมหนึ่งใหญ่มาก อีกโปรแกรมหนึ่งเล็กมาก แต่สองโปรแกรมนี้อาจมีค่า m เดียวกันเนื่องจากโปรแกรมทั้งสองมี CFG เดียวกัน นอกจากนี้ตัวเลข m ยังไม่สามารถบอกถึงความซับซ้อนในแง่การซ้อนใน และการวนซ้ำในโปรแกรม การสร้างระบบซอฟต์แวร์ที่จะหาค่า m นั้นทำได้โดยการเขียนโปรแกรมสำหรับแปลงโปรแกรมธรรมดา เช่น เขียนด้วยภาษาปาสคาลหรือซี ให้เป็น CFG จากนั้นก็ใช้โปรแกรมอีกโปรแกรมหนึ่งคำนวณค่า m หรือค่าตัววัดอื่นๆ

2.2 ตัววัดขมวดปม (Knot Complexity)

ตัววัดขมวดปม เสนอโดย วูดวาร์ด (Wood Ward), เฮนเนล (Hennel) และ เฮดเลย์ (Hedley) เพื่อวัดโครงสร้างของโปรแกรม โปรแกรมจะมีสิ่งที่เรียกว่า ปม นั่นก็ต่อเมื่อกระแสควบคุมสองเส้นทางตัดกันดังแสดงในรูปที่ 2.5 ถ้าโปรแกรมใดมีปมมากโปรแกรมนั้นก็ถือว่าซับซ้อนเข้าใจยาก เมื่อเทียบกับโปรแกรมเดียวกันที่ปมน้อยกว่า

การหาจำนวนปมโดยใช้ดินสอนั้น ทำได้โดยลากเส้นจากคำสั่ง ให้แยก (branch) ทุกคำสั่งไปยังจุดหมายปลายทาง แล้วดูว่าเส้นเหล่านี้ตัดกันที่ครั้ง ก็จะได้จำนวนปมตัววัดแบบนี้ เสนอครั้งแรกสำหรับภาษาฟอร์แทรน การหาปมสำหรับภาษาชั้นสูงสมัยใหม่ เช่น ซีหรือปาสคาลอาจจะมีปัญหาที่ว่าอาจหาไม่ได้ เนื่องจากภาษาสมัยใหม่มักไม่ใช้ goto ดังนั้นตัววัดจำนวนปมจึงค่อนค้างล้าสมัยไม่เป็นที่นิยมใช้

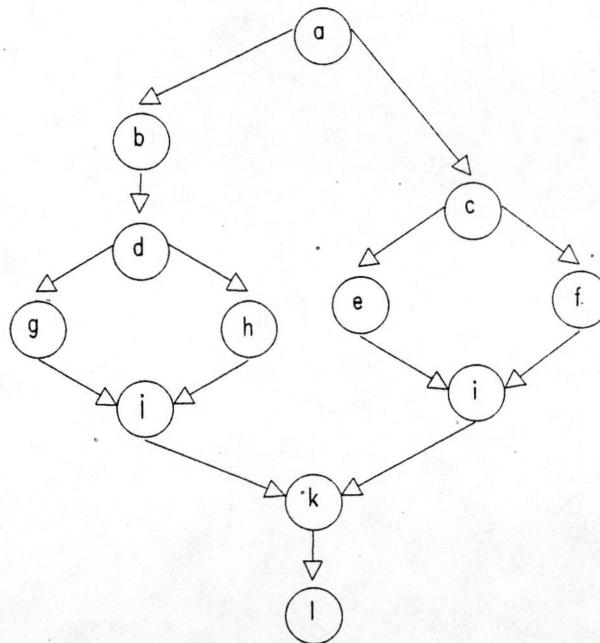


รูปที่ 2.5 แสดงปมใน CFG

2.3 ตัววัดสโคป (Scope Complexity)

ตัววัดสโคปนั้นได้มาจากแนวความคิดของ ฮาริสัน และ ฮอลสตีด คิดขึ้นเพื่อมาแก้ไขข้อเสียของตัววัดแมคเคบ ในเรื่องที่ว่าไซโคลแมติกไม่สามารถบ่งบอกอะไรเกี่ยวกับโครงสร้างการซ้อนใน และขนาดของโปรแกรม

นิยามของขอบเขตล่าง (lowerbound: lb) และขอบเขตล่างที่มากที่สุด (greatest lower bound : glb) คือ ถ้าให้ G เป็นกราฟย่อยของ CFG ค่า lb ของ G คือโหนดหนึ่งใน G ที่มาถึงได้จากทุกๆโหนดใน G ส่วนค่า glb คือค่าขอบเขตล่าง ที่อยู่เหนือขอบเขตล่างอื่นๆ ของ G พิจารณาตัวอย่างตามรูปที่ 2.6



รูปที่ 2.6 รูปตัวอย่างเพื่อแสดงการหาค่าจากตัววัดของสโคป

ถ้าเอาโหนด a เป็นหลัก จากโหนด a จะมีกราฟย่อยสองกราฟ ตั้งต้นที่โหนด b และโหนด c ทุกๆโหนดของกราฟทั้งสองนี้ไปถึงโหนด k และโหนด l ได้ ดังนั้น k และโหนด l เป็น lower bound ของโหนด a เนื่องจาก k ถึงก่อนโหนด l โหนด k จึงเป็น glb ของโหนด a จากนั้น นับดูว่าโหนดทั้งหมด ที่อยู่ระหว่าง a กับ k มีกี่โหนด ค่าที่ได้บวกกับ 1 คือ ความซับซ้อนของโหนด a ในกรณีนี้คือ 10 ข้อสรุปกรณีต่างๆ แสดงในตารางที่ 2.3

โหนด	ชื่อโหนดที่อยู่ระหว่าง	ความซับซ้อน
a	b,c,d,e,f,g,h,i,j	10
b	d	2
c	e,f	3
d	g,h	3
e	-	1
f	-	1
g	-	1
h	-	1
i	-	1
j	-	1
k	-	1
l	-	0

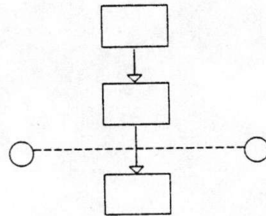
ตารางที่ 2.3 แสดงการหาค่าความซับซ้อนโดยวิธีตัววัดสโคป

สำหรับค่าความซับซ้อนรวมของกราฟนี้คือ ผลรวมค่าความซับซ้อนทั้งหมด ซึ่งมีค่าเท่ากับ 25

2.4 ตัววัดของเชน (Chen's Metrics)

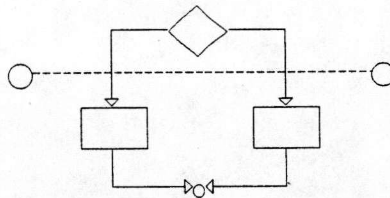
เอ็ดเวิร์ด ที. เชน (Edward T. Chen) ได้เสนอระบบชื่อ Maximal Intersect Number (MIN) ซึ่งเป็นอีกวิธีหนึ่งที่หาค่าความซับซ้อนจาก CFG โดยจะทำการวัดโครงสร้างการซ้อนใน (nesting structure) ของคำสั่งประเภทการตัดสินใจและวนซ้ำ การคำนวณตัววัดแบบนี้ โหนดที่เข้า (entry) และออก (exit) ของกราฟกระแสการควบคุมต้องเชื่อมโยงกันเป็นกราฟต่อเนื่อง (connected graph) และสามารถแบ่งออกเป็นบริเวณ (region) ได้หลายๆบริเวณ จากนั้นลากเส้นต่อเนื่องเข้าไปยังแต่ละบริเวณเพียงหนึ่งครั้ง ค่า MIN ย่อยในแต่ละบริเวณคือ จำนวนของจุดตัดที่มากที่สุดระหว่างเส้นกับเส้นเชื่อมภายใน CFG รูปที่ 2.7 แสดงการหาค่า MIN ของ CFG แบบต่างๆ

(ก)



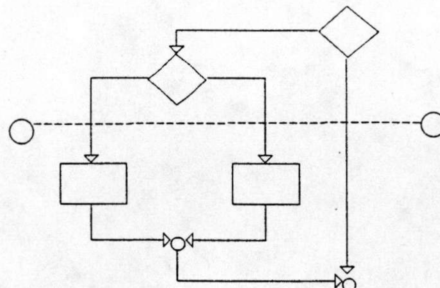
MIN = 1

(ข)



MIN = 2

(ค)



MIN = 3

รูปที่ 2.7 แสดงการหาค่า MIN จาก CFG

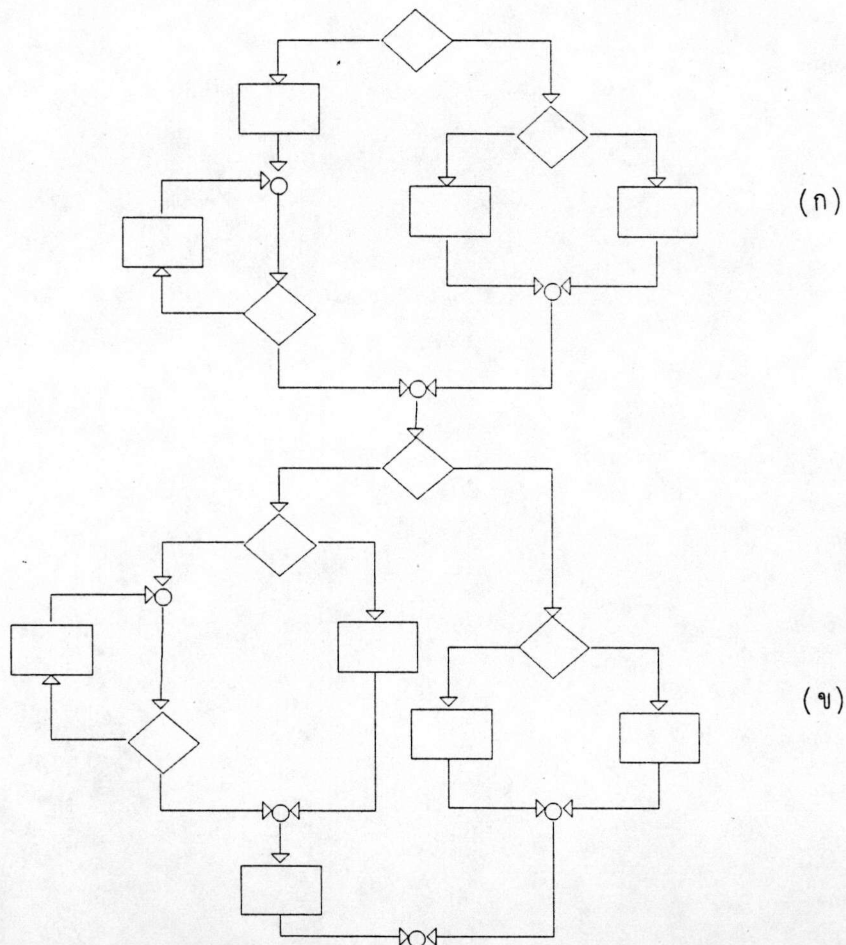
ค่า MIN รวมทั้งหมดหาได้จากสูตร

MIN = c - (2 * s) + 2

โดยที่ c คือ ผลรวมของ MIN ย่อยในแต่ละบริเวณ

s คือ จำนวนของบริเวณย่อยที่แบ่งได้ทั้งหมดใน CFG

ตัวอย่างการหาค่า MIN ในโปรแกรมตามรูปที่ 2.8 จากสูตรข้างบน ซึ่งมีค่า MIN ย่อย (ก) เท่ากับ 4 และค่า MIN ย่อย (ข) เท่ากับ 5 ดังนั้นค่า MIN รวมจะเท่ากับ 4+5 - 2*2 + 2 = 7



รูปที่ 2.8 แสดง CFG ตัวอย่างการวัดของเชน

ข้อสังเกตบางประการเกี่ยวกับตัววัดของเชนคือ

ก. โครงสร้างของโปรแกรมแบบลำดับ (sequence) จะไม่มีผลต่อการเพิ่มขึ้นหรือลดลงของค่า MIN แต่อย่างใด

ข. สำหรับโปรแกรมที่ประกอบด้วยคำสั่งประเภทตัดสินใจ และลำดับเท่านั้น ค่า MIN ย่อยของโปรแกรมนี้จะเท่ากับผลรวมของจำนวน if และ else บวกหนึ่ง ตัวอย่างเช่นในรูปที่ 2.7 (ก) ค่า MIN เท่ากับ 1 และรูปที่ 2.7 (ข) ค่า MIN เท่ากับ 2

2.5 <u>ตัววัดของแมคคลู (McClue's Metrics)</u>

ตัววัดของแมคคลู จะทำการวัดว่าแต่ละโมดูลมีการติดต่อกับโมดูลอื่นๆ ด้วยตัวแปรร่วม (common variable) และลำดับของการเรียกโมดูล (order of call) นั้นอย่างไร ขั้นตอนการวัดของแมคคลู มี 3 ขั้นตอน ดังนี้

2.5.1 คำนวณค่าความซับซ้อนของแต่ละตัวแปรควบคุม

2.5.2 คำนวณค่าความซับซ้อนแต่ละโมดูลภายในโปรแกรม

2.5.3 คำนวณค่าความซับซ้อนของโปรแกรมโดยรวมค่าความซับซ้อนแต่ละโมดูลเข้าด้วยกัน

สูตรการคำนวณค่าตัววัดของแมคคลู (C(m)) คือ

$$C(m) = C + V$$

โดยที่ C คือ จำนวนของการเปรียบเทียบภายในโมดูล

V คือ จำนวนของตัวแปรควบคุมที่ถูกอ้างอิงภายในโมดูล m

ตัวอย่างของการหาค่าความซับซ้อนด้วยวิธีของแมคคลู เช่น คำสั่ง

```
while ((A == B)  && ( C == 1))
```

ซึ่งมีการเปรียบเทียบ 2 ครั้ง คือ (A == B) กับ (C == 1) และมีตัวแปรควบคุม 3 ตัวคือ A, B และ C โดยค่าคงที่ 1 จะไม่นับเป็นตัวแปรควบคุม

ส่วนของโปรแกรม (ก)

```
if (t == M)  {
    x =1;
} else {
    if (t == 2) {
        x = 2;
    } else {
        if (t == 3) {
            x = 3;
        } else {
            x = 4;
        }
    }
}
```

ส่วนของโปรแกรม (ข)

```
if (t1 == M)  {
    x =1;
} else {
    if (t2 == 2) {
        x = 2;
    } else {
        if (t3 == 3) {
            x = 3;
        } else {
            x = 4;
        }
    }
}
```

จากส่วนของโปรแกรมทั้งสองสรุปผลได้ดังนี้

จำนวนการเปรียบเทียบของโปรแกรม (ก) เท่ากับ 3

จำนวนการเปรียบเทียบของโปรแกรม (ข) เท่ากับ 3

จำนวนตัวแปรควบคุมของโปรแกรม (ก) เท่ากับ 2 (t,M)

จำนวนตัวแปรควบคุมของโปรแกรม (ข) เท่ากับ 4 (t1,t2,t3,M)

ค่า C(m) ของโปรแกรม (ก) เท่ากับ 3 + 2 = 5 และ ค่า C(m) ของโปรแกรม (ข) เท่ากับ 3 + 4 = 7

เราสรุปได้ว่า โปรแกรม (ข) นั้นมีความซับซ้อนและทำความเข้าใจยากกว่าโปรแกรม (ก) เพราะโปรแกรม (ก) มีตัวแปรควบคุมเพียงสองตัว ขณะที่โปรแกรม (ข) มีถึงสี่ตัว

2.6 ตัววัดโอวิโด (Oviedo's Metrics)

ตัววัดโอวิโด หาจาก CFG และวิธีการใช้ข่าวสาร ดังนั้นตัววัดนี้จึงเป็นการวัดกระแสข้อมูล (dataflow) ด้วย โอวิโด นิยามค่า C เป็น

$$C = cf + df$$

โดยที่ cf คือ จำนวนเส้นเชื่อม (edge) ของ CFG ซึ่งเป็นค่าเชิงกระแสการควบคุม

df คือ ค่าเชิงกระแสข้อมูล (data flow) หาจาก

ก. ตัวแปรเมื่อได้รับค่าจากคำสั่งรับค่า หรือมีการรับค่าจากคำสั่งกำหนดค่า (assignment) หรือจาก โปรแกรมย่อย จะเรียกว่า ตัวแปรนั้นได้รับการนิยาม (define) เช่นคำสั่งต่อไปนี้เป็นการนิยามของตัวแปร X

```
scanf ("%d", &X);
X = 10;
interchange (&X, &Y);
```

ข. ตัวแปรได้รับการใช้หรืออ้างอิง (reference) เมื่อมีการใช้ค่าตัวแปรนั้นในคำสั่งประเภทกำหนดค่า หรือคำสั่งแสดงผลลัพธ์ เช่นคำสั่งต่อไปนี้เป็นการอ้างอิงของตัวแปร X

```
printf ("%d", X);
Y = X + 10;
if (X == 10)
```

ค. ตัวแปรจะมีค่าใช้ได้เฉพาะถิ่น (locally avaliable variable) ถ้าตัวแปรได้รับการนิยามในบล็อก (block) นั้น

ง. ตัวแปรจะได้รับการตีแผ่เฉพาะถิ่น (locally exposed variable) ถ้าใช้ตัวแปรนั้นในโหนด โดยที่ไม่มีการนิยามตัวแปรนี้ก่อน เช่น

```
Y = 10;
Z = X + Y;
X = Z * 2;
```

Y และ Z เป็นตัวแปรใช้ได้เฉพาะถิ่น และ X เป็นตัวแปรที่ได้รับการตีแผ่เฉพาะถิ่น

จ. เราบอกว่าตัวแปรนิยามในโหนด n_i ไปถึงโหนด n_k ได้ถ้ามีทางจาก n_i ไปยัง n_k โดยที่ตัวแปรนั้นไม่มีค่าใช้ได้เฉพาะถิ่น คือตัวแปรนั้นไม่ได้ถูกนิยามใหม่อีก ในโหนดใดๆบนทางนั้น

โอวิโด นิยามค่า df เท่ากับผลบวกของค่า df พิจารณาตัวแปรในทุกๆ โหนด นั่นคือ

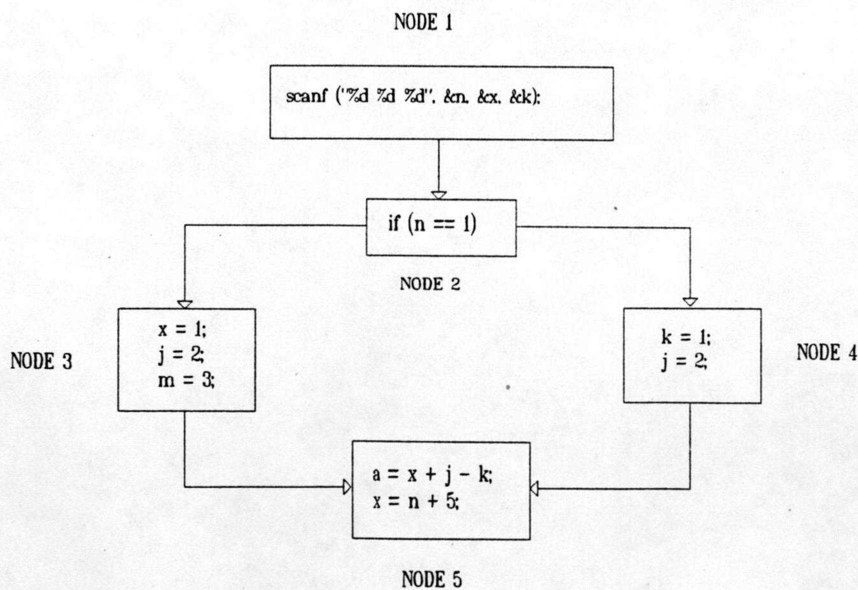
$$df = \sum_{i=1}^{/v/} df_i$$

โดยที่ /v/ คือ จำนวนโหนดใน CFG

df_i คือ จำนวนการนิยามก่อนของตัวแปรที่ได้รับการดีแผ่เฉพาะถิ่นในโหนด n_j ที่ไปถึง n_i ได้ หรือคือจำนวนนิยามของตัวแปรที่ไปถึงโหนดที่พิจารณาได้โดยตัวแปรในโหนดที่พิจารณาอยู่ต้องเป็นตัวแปรที่อยู่ทางซ้ายมือ และไม่ได้รับการนิยามในโหนดนั้น

ตัวอย่างการหาค่าความซับซ้อนด้วยวิธีของโอวิโด จากโปรแกรมข้างล่าง ซึ่งแสดงในรูปของ CFG ในรูปที่ 2.9 จากรูปค่า cf หรือคือจำนวนเส้นเชื่อมทั้งหมดเท่ากับ 5 ส่วนค่า df ย่อยในแต่ละโหนดปรากฏในตารางที่ 2.4 ซึ่งเป็นการสรุปค่าตัวแปรดีแผ่เฉพาะถิ่นของแต่ละโหนดที่ได้รับการนิยามมาก่อน โดย X(i) หมายถึงการนิยามของตัวแปร X ในโหนด i ซึ่งเป็นโหนดที่มาก่อนและมีเส้นทางเชื่อมโยงถึงโหนด j

```
scanf ("%d %d %d", &n, &x, &k);
if (n == 1) {
    x = 1; j = 2; m = 3;
} else {
    k = 1; j = 2;
}
a = x + j + k; x = n + 5;
```



รูปที่ 2.9 แสดง CFG ของโปรแกรม

โหนดที่	ตัวแปรที่แผ่เฉพาะถิ่น	การนิยามครั้งก่อน	ความซับซ้อนเชิงกระแสข้อมูล
1	–	–	0
2	n	n(1)	1
3	-	-	0
4	–	–	0
5	x	x(1), x(3)	2
	j	j(3), j(4)	2
	k	k(1), k(4)	2
	n	n(1)	1

ตารางที่ 2.4 แสดงค่าความซับซ้อนกระแสข้อมูล

จากตารางผลรวมของ df ย่อยเท่ากับ 8 ดังนั้นค่าความซับซ้อนของโอวิโดของโปรแกรมนี้เท่ากับ 5 + 8 = 13