บทที่ 7 กราฟ (Graphs)

7.1 บทนิยาม

7.2 กราฟในโปรแกรม

7.3 การแทนที่ของเมทริกซ์ประชิด

7.4 การแทนที่แบบโยง

7.5 การแวะผ่านกราฟ

- การแวะผ่านในแนวกว้าง
- การแวะผ่านในแนวลึก
- การเปรียบเทียบกัน

7.6 ระยะทางไปถึงและทางเดินสั้นที่สุด

7.7 ทางเดินวิกฤต

7.8 ต้นไม้แบบทอดข้าม

บทสรุป

แบบฝึกหัด

IT 204 185

บทที่ 7
กราฟ
(Graphs)

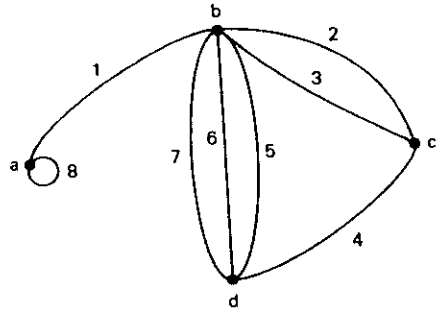
โครงสร้างข้อมูลชนิดต่าง ๆ ที่ได้อภิปรายมาแล้วในบทก่อนหน้านี้มีโครงสร้างแบบเชิงเส้นซึ่งสมาชิกแต่ละตัว จะมีสมาชิกตัวถัดไปหนึ่งตัว ความเป็นเชิงเส้นนี้เป็นเรื่องปกติของสายอักขระ ของสมาชิกของแถวลำดับหนึ่งมิติ , ของเขตข้อมูลในระเบียน , ของหน่วยข้อมูล (entries) ในกองซ้อน , ของหน่วยข้อมูลในแถวคอย และของโหนดในรายการโยงอย่างง่าย ในบทนี้จะเริ่มต้นศึกษาโครงสร้างข้อมูลแบบไม่ใช่เชิงเส้น (nonlinear data structures) ในโครงสร้างเหล่านี้ สมาชิกแต่ละตัวอาจมีสมาชิกตัวถัดไปหลายตัวแนะนำแนวคิดของโครงสร้างการแตกกิ่ง โครงสร้างข้อมูลการแตกกิ่งเหล่านี้เรียกว่า กราฟ และต้นไม้

เราจะอภิปรายกราฟ การแทนที่ และการปฏิบัติการของกราฟในบทนี้ จากนั้นบทที่ 8 จึงเริ่มต้นอภิปรายเรื่องต้นไม้ซึ่งเป็นกราฟพิเศษชนิดหนึ่ง

7.1 บทนิยาม (Definitions)

กราฟหมายถึงเซตของจุดและเซตของเส้น ซึ่งแต่ละเส้นต่อหนึ่งจุดกับอีกหนึ่งจุด จุดเหล่านี้เรียกว่า **โหนด** ของกราฟ และเส้นเรียกว่า **ด้าน** (Intuitively, a graph is a set of points and a set of lines, with each line joining one point to another. The points are called nodes of the graph, and the lines are called the edges.)

เซตของโหนดในกราฟ G ใช้สัญลักษณ์ V_G และเซตของด้านใช้สัญลักษณ์ E_G ตัวอย่างเช่น ในกราฟ G รูป 7-1 V_G = { a, b, c, d } และ E_G = {1, 2, 3, 4, 5, 6, 7, 8 }



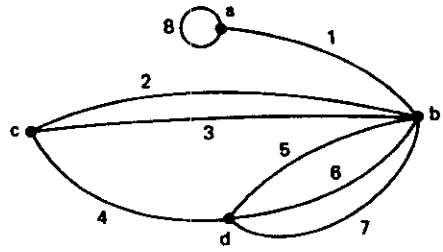
รูป 7-1 ตัวอย่างกราฟ

จำนวนสมาชิกในเซต V_G เรียกว่า**อันดับ** (order) ของกราฟ G

กราฟว่าง หมายถึงกราฟที่มีอันดับเท่ากับศูนย์ (A nullgraph is a graph with order zero)

หนึ่งด้านกำหนดโดยโหนดที่ต่อกัน ตัวอย่างเช่น ด้าน 4 ต่อโหนด c กับโหนด d เรียกว่า form(c, d)

กราฟถูกกำหนดอย่างบริบูรณ์ โดยเซตของโหนดและเซตของด้าน การเป็นตำแหน่งจริงของสมาชิกเหล่านี้บนกระดาษไม่สำคัญ กราฟรูป 7-1 สมมูลกับกราฟรูป 7-2



รูป 7-2 กราฟที่สมมูลกับรูป 7-1

โปรดสังเกตว่าอาจมีหลายด้านต่อสองโหนด ตัวอย่างเช่น ด้าน 5, 6 และ 7 ทั้งหมดนี้เป็น form(b, d)

บางคู่ของโหนดอาจจะไม่ต่อกัน เช่น ไม่มีด้านของ form(a, c) หรือ (a, d)

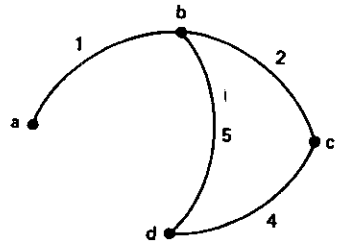
บางด้านอาจต่อหนึ่งโหนด คือ ตัวมันเอง เช่น ด้าน 8 คือ form(a, a) ด้านชนิดนี้เรียกว่า **วงวน** (loops)

กราฟ G จะเรียกว่า **กราฟเชิงเดียว** (simple graph) ถ้าเงื่อนไขสองข้อข้างล่างนี้เป็นจริงทั้งคู่

(1) กราฟนี้ไม่มีวงวนนั่นคือ ไม่มีด้านใน E_G ของ form(v, v) เมื่อ v อยู่ใน V_G

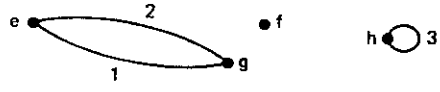
(2) จำนวนด้านซึ่งต่อคู่ใด ๆ ของโหนดต้องไม่มากกว่าหนึ่งด้าน นั่นคือไม่มีด้านใน E_G มากกว่าหนึ่งด้านเป็น form(v_1, v_2) สำหรับคู่ใด ๆ ของสมาชิก v_1 และ v_2 ใน V_G

รูป 7-3 อธิบายกราฟเชิงเดียวซึ่งได้มาจากกราฟตัวอย่างของรูป 7-1



รูป 7-3 ตัวอย่างกราฟเชิงเดียวได้มาจากรูป 7-1

กราฟซึ่งไม่ใช่กราฟเชิงเดียวบางครั้งเรียกว่า **กราฟหลายเส้น** (multigraph) บางครั้งเราเรียกด้าน(edges) ว่า ส่วนโค้ง(arcs) และเรียกโหนด(node) ว่าจุด(vertices) **กราฟเชื่อมโยง** (connected graph) หมายถึงกราฟซึ่งไม่สามารถแบ่งออกเป็นกราฟสองชุดโดยไม่มีการลบอย่างน้อยที่สุดหนึ่งด้าน กราฟรูป 7-4 ไม่ใช่กราฟเชื่อมโยง



รูป 7-4 ตัวอย่างกราฟแบบไม่เชื่อมโยง (unconnected graph)

ทางเดิน(Paths)

ทางเดินในกราฟหมายถึงลำดับของหนึ่งด้าน หรือมากกว่าหนึ่งด้านซึ่งเชื่อมโยงสองโหนดเราใช้สัญลักษณ์ $P(v_i, v_j)$ แทนทางเดินซึ่งเชื่อมโยงโหนด v_i และ v_j สำหรับ $P(v_i, v_j)$ ที่มีอยู่จริงใน E_G ต้องมีลำดับของด้านในรูปแบบดังนี้

$$P(v_i, v_j) = (v_i, x_1)\ (x_1, x_2)\ \ldots\ (x_{n-1}, x_n)\ (x_n, v_i)$$

ความยาว(Length) ของทางเดิน หมายถึงจำนวนด้านซึ่งประกอบกันเป็นทางเดินนั้น ในกราฟเชิงเดียวรูป 7-3 ทางเดินต่อไปนี้คือทางเดินระหว่างโหนด b และ d:

P(b, d) = (b, c)(c, d) length = 2

P(b, d) = (b, c)(c, d)(b, c)(c, d) length = 4

P(b, d) = (b,d) length = 1

P(b, d) = (b, c)(c, b)(b, d) length = 3

โดยทั่วไป เราสนใจเฉพาะทางเดินซึ่งโหนดที่กำหนดจะถูกเยี่ยมไม่เกินหนึ่งครั้ง ข้อจำกัดนี้จึงสนใจเฉพาะ ทางเดินที่หนึ่งและทางเดินที่สามข้างต้นเท่านั้น ทางเดินที่สองเยี่ยมโหนด b และ c สองครั้งทั้งคู่ ทางเดินที่สี่เยี่ยมโหนด b สองครั้ง ปกติเราจะสนใจการไม่แวะผ่านด้านเดียวกันในทางเดินมากกว่าหนึ่งครั้ง สิ่งนี้จะป้องกันอัลกอริทึมจากการตามรอยใหม่ขั้นตอนของมันที่ช้าลง

วัฏจักร(cycles)

วัฏจักรหมายถึงทางเดินซึ่งเงื่อนไขสองข้อข้างล่างนี้เป็นจริงทั้งคู่

(1) ไม่มีด้านใด ๆ ปรากฏมากกว่าหนึ่งครั้งในลำดับของด้าน (No edge appears more than once in the sequence of edges.)

(2) โหนดแรกของทางเดินคือโหนดเดียวกับโหนดสุดท้ายของทางเดิน ตัวอย่างเช่น

P(v , v)

(The initial node of the path is the same as the terminal node of the path; i.e., P(v, v))

พูดอีกอย่างหนึ่งคือ วัฏจักรส่งกลับไปยังที่ซึ่งเริ่มต้น กราฟในรูป 7-2 มีวัฏจักรหลายชุด ตัวอย่างเช่น

P(a, a) = (a, a)

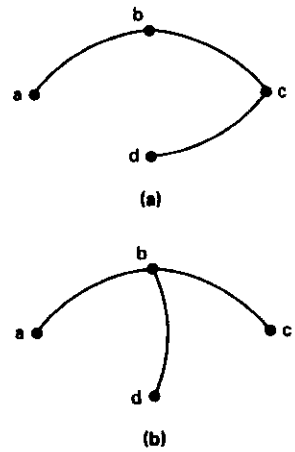
P(b, b) = (b, c)(c, b)

P(b, b) = (b, c)(c, d)(d, b)

P(d, d) = (d, b)(b, c)(c, d)

P(d, d) = (d, b)(b, d)

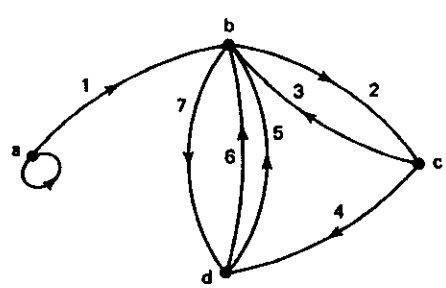
กราฟซึ่งไม่มีวัฏจักรเรียกว่า **อวัฏจักร** (acyclic) รูป 7-5 (a) และ (b) ทั้งคู่แทนกราฟอวัฏจักร (acyclic graphs)



รูป 7-5 ตัวอย่าง acyclic graphs

กราฟระบุทิศทาง (Directed Graphs)

กรณีพิเศษอีกชนิดหนึ่งของโครงสร้างข้อมูลกราฟทั่วไปเรียกว่ากราฟระบุทิศทาง ซึ่งการระบุทิศทางถูกกำหนดให้กับด้านของกราฟ ตัวอย่างแสดงในรูป 7-6
แต่ละด้านของกราฟระบุทิศทางจะมีลูกศรกำกับไว้



รูป 7-6 ตัวอย่างกราฟระบุทิศทาง

ระดับขั้นเข้า(in-degree)ของโหนดในกราฟระบุทิศทาง คือจำนวนด้านซึ่งจบที่โหนดนั้น
ระดับขั้นออก(out-degree) ของโหนด คือจำนวนด้านซึ่งออกจาก(emanate) จากโหนดนั้น
ระดับขั้น(degree) ของโหนด คือผลรวมของ in-degree และ out-degree ของโหนดนั้น
ตัวอย่างรูป 7-6 ค่าต่าง ๆ เป็นดังนี้

in-degree(a) = 1	out-degree(a) = 2	degree(a) = 3
in-degree(b) = 4	out-degree(b) = 2	degree(a) = 6
in-degree(c) = 1	out-degree(c) = 2	degree(a) = 3
in-degree(d) = 2	out-degree(d) = 2	degree(a) = 4

ตลอดเนื้อหาในบทนี้เราอ้างถึงโหนดต่างๆ ของกราฟ โดย labels โหนดนั้น ในความเป็นจริง หนึ่งโหนดอาจจะประกอบด้วยสารสนเทศชนิดใดก็ได้เราจะไม่สนใจเนื้อหา (contents) ของโหนดเหล่านี้ อย่างไรก็ตามไม่ควรลืมว่าสุดท้ายวัตถุประสงค์ของกราฟและต้นไม้คือการแทนที่โครงสร้างเชิงตรรกะของสารสนเทศนี้

7.2 กราฟในโปรแกรม (Graphs in Programs)

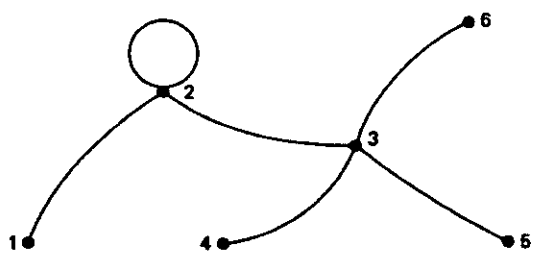
เช่นเดียวกับโครงสร้างข้อมูลอื่น ๆ ซึ่งเราได้ศึกษามาแล้ว ภาษาโปรแกรมที่ใช้โดยทั่วไปไม่มีแบบชนิดข้อมูลในตัวที่เรียกว่า "graph" ดังนั้นคุณสมบัติของกราฟจึงถูกจำลอง (simulated) ขึ้นโดยเก็บในโครงสร้างข้อมูลอีกชนิดหนึ่ง วิธีการเข้าถึงการแทนที่ของกราฟที่สำคัญมี 3 ชนิดคือ การแทนที่ด้วยเมทริกซ์ , การแทนที่ด้วยรายการ, และการแทนที่ด้วยหลายรายการ หลังจากพิจารณาแต่ละวิธีเหล่านี้แล้ว จะได้อภิปรายการปฏิบัติการบนกราฟที่มากขึ้นได้แก่ การแวะผ่านกราฟและการวิเคราะห์ทางเดิน

7.3 การแทนที่ด้วยเมทริกซ์ประชิด (Adjacency Matrix Representation)

จงพิจารณากราฟ G ซึ่งประกอบด้วยเซตของโหนด V_G และเซตของด้าน E_G สมมติว่ากราฟ G มี order เท่ากับ N สำหรับ $N \geq 1$ วิธีหนึ่งของการแทนที่กราฟนี้คือใช้เมทริกซ์ประชิด ซึ่งเป็นแถวลำดับขนาด $N \times N$ เมื่อ

$$A(i, j) = \begin{cases} 1 \text{ ก็ต่อเมื่อด้าน } (v_i, v_j) \text{ อยู่ใน } E_G \\ 0 \text{ กรณีอื่น ๆ} \end{cases}$$

ถ้ามีด้านต่อโหนด i และ j เพราะฉะนั้น A(i, j) = 1 เมทริกซ์ประชิดสำหรับกราฟไม่ระบุทิศทางในรูป 7-7 คือ



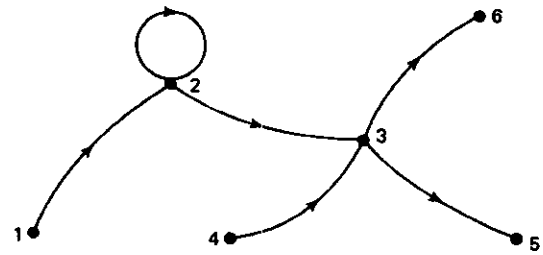
รูป 7-7 ตัวอย่างกราฟไม่ระบุทิศทาง

i \ j	1	2	3	4	5	6
1	0	1	0	0	0	0
2	1	1	1	0	0	0
3	0	1	0	1	1	1
4	0	0	1	0	0	0
5	0	0	1	0	0	0
6	0	0	1	0	0	0

กราฟระบุทิศทาง (Directid Graphs)

ด้านหนึ่งด้านของกราฟระบุทิศทางจะมีแหล่งต้นทาง(source) ของมันที่หนึ่งโหนด และจบ (terminates) ที่อีกหนึ่งโหนด โดยปกติด้าน v_i, v_j) แทนทิศทางจากโหนด v_i ไปโหนด v_j

เมทริกซ์ประชิดสำหรับกราฟระบุทิศทางรูป 7-8 คือ



รูป 7-8 ตัวอย่างกราฟระบุทิศทาง

i \ j	1	2	3	4	5	6
1	0	1	0	0	0	0
2	0	1	1	0	0	0
3	0	0	0	0	1	1
4	0	0	1	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0

เมทริกซ์มากเลขศูนย์ หรือ เมทริกซ์สพาซ(Sparse Matrices)

กราฟที่มีโหนด n ตัวจะมีเมทริกซ์ประชิดขนาด N × N ในหลายกรณี เมทริกซ์ประชิดเหล่านี้ค่อนข้างจะเป็นเลขศูนย์เป็นส่วนใหญ่ และถูกเก็บในแถวลำดับสพาซ กรณีเหล่านี้วิธีการแทนที่เมทริกซ์โดยทั่ว ๆ ไป ไม่มีวิธีใดเหมาะสมเท่ากับการแทนที่แบบโยงซึ่งจะได้แนะนำต่อไปในบทนี้ เมทริกซ์ประชิดของกราฟแบบไม่ระบุทิศทางจะเป็นเมทริกซ์สมมาตร (symmetric) แต่อย่างไรก็ตาม ถึงแม้ว่าจะไม่เป็นเมทริกซ์สพาซ จะต้องใช้หน่วยเก็บอย่างมากที่สุดคือครึ่งหนึ่งโดยเก็บเฉพาะสามเหลี่ยมบนหรือล่าง (upper or lower triangular)

การให้นิยามกราฟในภาษา COBOL และ Pascal

(Defining Graphs in COBOL and Pascal)

บทนิยามแถวลำดับข้างล่างนี้ใช้เพื่อประกาศกราฟชื่อ GRAPH มีอันดับ(order)เท่ากับ 24 ในภาษา COBOL กราฟนี้จึงต้องใช้เมทริกซ์ประชิดขนาด 24 × 24

```
01  GRAPH.
    02  ROW   OCCURS  24 TIMES.
        03  COL OCCURS  24 TIMES.
            04  EDGE   PIC  9.
```

เพราะฉะนั้น EDGE(i, j) จึงมีค่าเป็น 1 ถ้ามีด้านระหว่างโหนด i และโหนด j และมีค่าเป็น 0 ในกรณีอื่น ๆ

อีกวิธีหนึ่งในการให้เความหมายเมทริกซ์ประชิด คือแบบชนิดข้อมูล boolean ตัวอย่างเช่นในภาษา Pascal :

```
type  graph : array [1 .. 24, 1 .. 24] of boolean;
```

ในที่นี้ graph[i, j] มีค่าเป็น true ถ้ามีด้านระหว่างโหนด i และ j และมีค่าเป็น false กรณีอื่น ๆ

การคำนวณด้าน (Edge calculations)

บ่อยครั้งมีข้อดีที่สามารถคำนวณบนเมทริกซ์ประชิดได้ ตัวอย่างเช่น ระดับขั้นของโหนด i ในกราฟไม่ระบุทิศทางคือ

$$\sum_{j=1}^{N} A(i, j)$$

ระดับขั้นเข้า(in-degree) ของโหนด i ในกราฟระบุทิศทางคือผลรวมตามแนวตั้ง (column-sum)ของเมทริกซ์คือ

$$\sum_{k=1}^{N} A(k, i)$$

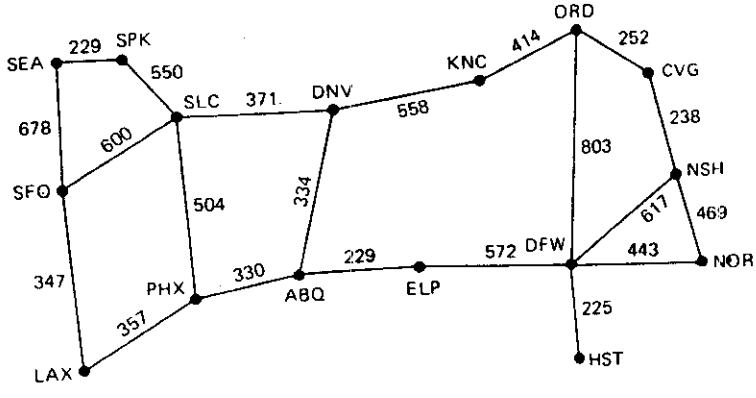
และระดับขั้นออก(out-degree) ของโหนด i คือผลรวมตามแนวนอน(row-sum) ของเมทริกซ์คือ

$$\sum_{k=1}^{N} A(i, k)$$

เราจะอภิปรายผลลัพธ์ร่วมอื่น ๆ ของการคุมแต่งข้อมูลประชิดภายหลัง ถ้าเมทริกซ์ประชิดถูกนิยามเป็นบูลีน(boolean) ดังนั้นการคุมแต่งร่วมเหล่านี้จะไม่ค่อยตรงไปตรงมานัก แทนที่การรวมยอด(summation)จึงต้องใช้ชุดของการดำเนินการแบบบูลีน(and และ or) แทน

ด้านถ่วงน้ำหนัก(Weighted Edges)

งานประยุกต์กราฟจำนวนมากซึ่งมีความหลากหลายบนการแทนที่เมทริกซ์ประชิดได้อย่างเหมาะสม จงพิจารณากราฟในรูป 7-9 ซึ่งอาจนำมาใช้ในระบบสารสนเทศ ของบริษัทรับส่งสินค้า นี่คือตัวอย่างของกราฟที่มีด้านถ่วงน้ำหนัก โหนดแทนชื่อเมือง ส่วนด้านแทนระยะทางระหว่างเมือง แต่ละด้านถูกกำหนดด้วยระยะทางระหว่างคู่ของเมืองที่ต่อกันแทนที่การใช้ bit matrix เพื่อแทนระบบรับส่งสินค้านี้ เราจะใช้การแทนที่เมทริกซ์ถ่วงน้ำหนักด้าน (รูป 7-10)



รูป 7-9 ตัวอย่างกราฟแสดงให้เห็นระยะทางระหว่างเมือง

ด้านถ่วงน้ำหนักถูกนำมาใช้บ่อยมาก ในงานประยุกต์การขนส่ง ปกติน้ำหนักแทนระยะทางเหมือนตัวอย่างข้างต้น ในงานประยุกต์การไหล (flow applications) ปกติน้ำหนักแทนความจุ ตัวอย่างเช่น โหนดบนกราฟอาจจะแทนความจุ หน่วยเป็นแกลลอน/นาที (gallon/minute)ของสายท่อ(pipe-tine) ระหว่างตำแหน่งที่ต่อกัน หรือ

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1		0	0	334	229	0	0	0	0	0	0	330	0	0	0	0
2			0	0	0	0	0	0	238	0	252	0	0	0	0	0
3				0	572	225	0	0	617	443	803	0	0	0	0	0
4					0	0	558	0	0	0	0	0	0	0	371	0
5						0	0	0	0	0	0	0	0	0	0	0
6							0	0	0	0	0	0	0	0	0	0
7								0	0	0	414	0	0	0	0	0
8									0	0	0	357	0	347	0	0
9										469	0	0	0	0	0	0
10											0	0	0	0	0	0
11												0	0	0	0	0
12													0	0	504	0
13														678	0	229
14															600	0
15																550
16																

รูป 7-10 การแทนที่เมทริกซ์ด้านถ่วงน้ำหนักของรูป 7-9

ความจุมีหน่วยเป็นบิต/วินาที (bits/second) ของสิ่งอำนวยความสะดวก การสื่อสารระหว่างสถานีการสวิตช์ที่เชื่อมโยง(connected switching stations)

ในงานประยุกต์อื่น ๆ ด้านถ่วงน้ำหนักแทนเวลา ตัวอย่างเช่น กราฟที่ใช้แทนเครือข่าย (networks) ของกิจกรรม (activities)แต่ละด้านแทนหนึ่งงาน (task)หรือหนึ่งกิจกรรม (activity)น้ำหนักบนด้าน แทนปริมาณของเวลาที่ต้องใช้ เพื่อให้กิจกรรมนั้นสำเร็จ

บริบูรณ์ แต่ละโหนดแทนเหตุการณ์ (event) การแสร็จบริบูรณ์ของเซตของ กิจกรรม แทนด้วยด้านที่เข้ามายังโหนดไม่มีกิจกรรมใด ๆ บน out-edge ที่จะเริ่มต้นจนกว่ากิจกรรมทั้งหมดบน in-edges ไปยังหนึ่งโหนดได้เสร็จบริบูรณ์แล้ว(ความหลากหหลายบนกราฟกิจกรรมพื้นฐาน ทำให้เหตุการณ์เสร็จบริบูรณ์เป็นสัญญาณ โดยเซตย่อยของด้านกิจกรรมที่ตกกระทบ)เราจะพิจารณา activity graphs ละเอียดมากขึ้นภายหลังในการอภิปรายหัวข้อการวิเคราะห์ทางเดินวิกฤต(critical path analysis)

7.4 การแทนที่แบบโยง (Linked Representation)

การแทนที่กราฟโดยใช้เทคนิคเมทริกซ์ประชิด ต้องใช้หน่วยเก็บของสารสนเทศด้านสำหรับแต่ละคู่ที่เป็นไปได้ของโหนด สำหรับกราฟที่มีโหนด N ตัวกับด้านจำนวนไม่มากที่สัมพันธ์กับการต่อกันที่เป็นไปได้ N^2 การแทนที่แบบโยงโดยทั่วไปแล้วมีความเหมาะสมมากกว่า

โปรแกรมเมอร์มือใหม่บนโครงการที่ศูนย์คอมพิวเตอร์ของเมือง ค้นพบวิธีที่ยากที่จะได้ประโยชน์ของการเลือกการแทนที่ ที่เหมาะสมสำหรับโครงสร้างข้อมูล ให้กับงานประยุกต์เขารับผิดชอบสำหรับโปรแกรมภาษา COBOL ซึ่งเป็นส่วนหนึ่งของระบบการขนส่ง โปรแกรมถูกนำมาใช้เพื่อเก็บ track ของปริมาณการจราจร (traffic volumes) บนถนนหลักของเมือง โหนดบนกราฟแทนสี่แยก(intersections) ด้านแทนถนน(streets)น้ำหนัก(weights)บนด้านแทนปริมาณจราจรแต่ละด้านสำหรับถนนเดินทางเดียว(one-way street) จะมีน้ำหนักเดียวเท่านั้น แต่ละด้านสำหรับถนนเดินสองทาง (two-way street) จะมีน้ำหนักสองชุด ข้อมูลถูกเก็บสำหรับสี่แยกจำนวน 500 แห่ง โปรแกรมเมอร์ใช้เมทริกซ์ประชิด เพื่อแทนกราฟจราจร สี่แยกจำนวนน้อยแทนที่บรรจบกัน (confluence) ของ มากกว่าสี่ส่วนของถนน (more than four street segments) สมาชิกส่วนใหญ่ของเมทริกซ์ประชิดจึงเป็นศูนย์ หน่วยเก็บของโปรแกรมต้องป้องกันไม่ใส่มันในหน่วยความจำหลักของคอมพิวเตอร์ของเมือง และแทนที่จะใช้ชั่วโมงจำนวนมากให้กระชับกับรหัส PROCEDURE DIVISION การส่งที่เสร็จโปรแกรม(หลายครั้ง) และรหัสไม่เคยได้รับการทดสอบ โปรแกรมเมอร์ที่มีประสบการณ์พบรหัสที่มีปัญหาบนโต๊ะทำงาน แสดงให้เห็นความต้องการหน่วยเก็บทั้งหมดสำหรับเมทริกซ์ แปลงผันโปรแกรมให้ใช้การแทนที่แบบโยงและจบโครงการ

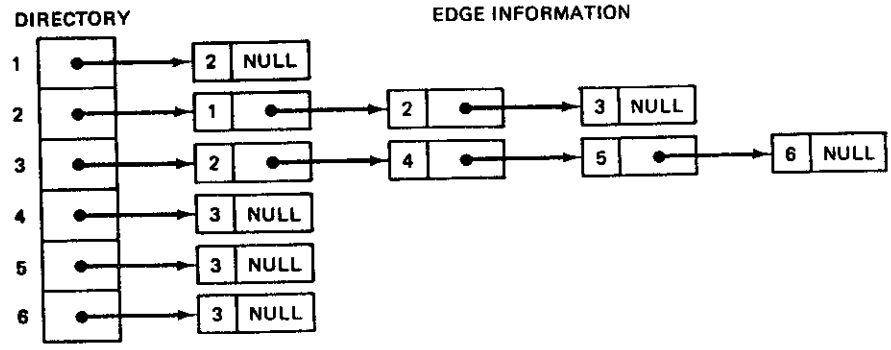
ตรงกันข้ามกับเทคนิคเมทริกซ์ประชิด ซึ่งเก็บสารสนเทศเกี่ยวกับด้านที่เป็นไปได้ทุกด้าน การแทนที่แบบโยงเก็บเฉพาะด้านที่มีอยู่จริง ขณะที่ด้านถูกนำมาใส่หรือถูกลบออก

จากกราฟ การแทนที่แบบโยงตรงดัดแปรตามนั้น โครงสร้างแบบโยงพื้นฐานเพื่อแทนที่กราฟมีสองชนิด ชนิดที่หนึ่งเรื่องว่าการแทนที่สารบบของโหนดและอีกชนิดหนึ่งเรียกว่าการแทนที่หลายรายการ (There are two fundamental types of linked stucture to represent graphs : one is called the node directory representation and the other one is called the multi-list representation.)

7.4.1 การแทนที่สารบบโหนด (Node Directory Representation)

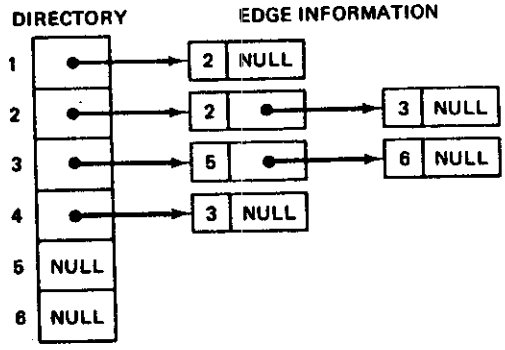
การแทนที่สารบบโหนดแบ่งออกเป็นสองส่วน : สารบบ และเซตของรายการโยง (The node directory representation includes two parts : a derectory and a set of linked lists.)

มีหนึ่งหน่วยข้อมูล (entry) ในสารบบ สำหรับแต่ละโหนดของกราฟทั้งนี้ directory entry สำหรับโหนด i ชี้ไปยังรายการโยงซึ่งแทนที่โหนดต่าง ๆ ซึ่งต่อกับโหนด i แต่ละระเบียนของรายการโยงมีสองเขตข้อมูล(fields): เขตข้อมูลที่หนึ่งคือ node identifier และอีกเขตหนึ่งคือตัวชี้ไปยังสมาชิกตัวถัดไปของรายการ เพราะฉะนั้น สารบบจึงแทนที่โหนด และรายการโยงแทนที่ด้าน การแทนที่สารบบโหนดของกราฟแบบไม่ระบุทิศทาง รูป 7-7 ถูกกำหนดในรูป 7-11 ข้างล่างนี้



รูป 7-11 การแทนที่สารบบโหนด ของรูป 7-7

กราฟแบบไม่ระบุทิศทางที่มีอันดับ เท่ากับ N และมีด้านเท่ากับ E ต้องการ หน่วยข้อมูล N ตัวในสารบบ และจำนวนหน่วยข้อมูลของรายการโยงเท่ากับ 2 * E ยกเว้น แต่ละรูป วงวน (loop) ซึ่งลด หน่วยข้อมูลของรายการโยงลงไปหนึ่ง การแทนที่สารบบโทนดของกราฟแบบมีทิศทางรูป 7-8 กำหนดโดยรูป 7-12



รูป 7-12 การแทนที่สารบบโทนดของรูป 7-8

กราฟแบบมีทิศทางที่มีอันดับเท่ากับ N และ E ด้านต้องการหน่วยข้อมูล N ตัวในสารบบ และ entries ของรายการโยงเท่ากับ E

รายการโยงนำหน้าโดย หน่วยข้อมูลของสารบบตัวที่ i สมนัยกับแถวที่ i ของการแทนที่เมทริกซ์ประชิด หน่วยข้อมูลของสารบบเรียงอันดับแบบสืบเนื่องโดย node identifier ถึงแม้ว่าเรามีการเรียงอันดับ หน่วยข้อมูลของรายการโยงโดย node identifier ด้วย แต่หน่วยข้อมูลเหล่านี้จะเรียงอันดับอย่างไรก็ได้ เพราะว่าแต่ละ entry สัมพันธ์กับ node identity ของมัน

การประกาศในภาษา COBOL ของการแทนที่สารบบโทนดของกราฟแบบไม่ระบุทิศทางที่มีอันดับเท่ากับ 24 มีด้านสูงสุด เท่ากับ100 ด้านเขียนดังนี้

```
01  GRAPH.
      02  NODE OCCURS  24  TIMES  PICTURE  9(3).
      02  EDGES.
        03 LIST-ENTRY  OCCURS  200 TIMES.
            04  NODE-ID        PICTURE  99.
             04  NEXT-ENTRY  PICTURE  9(3).
```

การศึกษาวิธีนี้ต้องใช้ หน่วยข้อมูล N ตัวในสารบบ และ 2 * max (E) เป็นสมาชิกของแถวลำดับเมื่อ N คืออันดับของกราฟและ max(E) คือจำนวนสูงสุดของด้าน Max(E) อาจใหญ่เท่ากับ N(N-1) ซึ่งทำให้จำนวนหน่วยข้อมูลทั้งหมดในสารบบและแถวลำดับเท่ากับ N^2 ซึ่งเท่ากับที่ใช้ในเมทริกซ์ประชิด

ภาษา Pascal ซึ่งสนับสนุนการจัดสรรหน่วยความจำแบบพลวัต อาจจะลดการใช้เนื้อที่ลง กราฟแบบไม่ระบุทิศทางมีอันดับเท่ากับ 24 และจำนวนสูงสุดของด้านเท่ากับ100 ประกาศใน Pascal ดังนี้

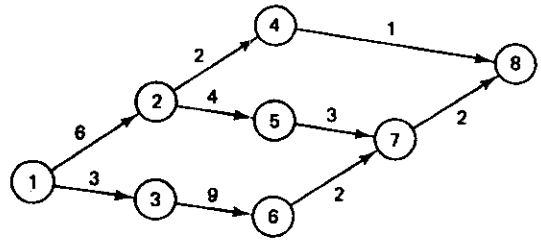
```
type    nodeid = 0. . 24;
        pedgeptr = ↑ edgeinfo ;
        edgeinfo = record  node : nodeid;
                           next : edgeptr
                   end;
        directory = array [1 . . 24] of edgeptr;
```

โปรดสังเกตว่าในการประกาศนั้นจำนวนสูงสุดของด้านไม่สำคัญ ระเบียน edgeinfo ถูกจัดสรรให้เป็นโหนดซึ่งใส่เข้าไปในกราฟการศึกษาวิธีนี้ต้องใช้หน่วยข้อมูล N ตัวในสารบบและ 2 * E ระเบียน edgeinfo เท่ากับ 2 * E เมื่อ N คืออันดับของกราฟและ E คือจำนวนด้าน การทำให้เกิดผลในการปฏิบัติ จับคู่กับ (matchs) การออกแบบรายการโรงซึ่งนำเสนอก่อนหน้าในหัวข้อนี้ การทำให้เกิดผลของแถวลำดับ COBOL โดยประมาณกับการออกแบบรายการโยง

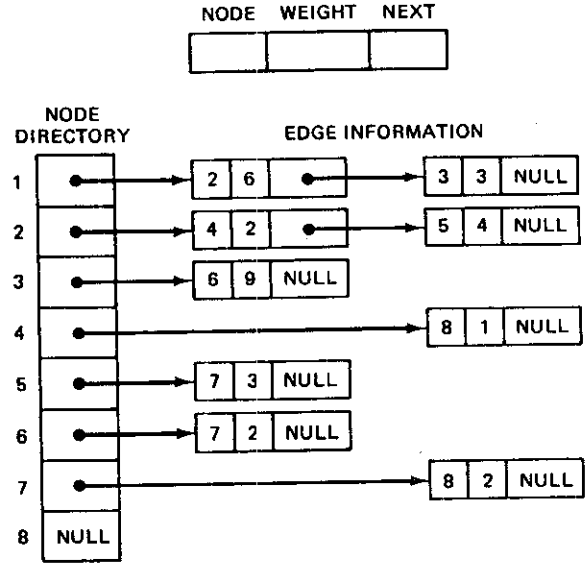
ด้านถ่วงน้ำหนัก (Weighted Edges)

การแทนที่ของกราฟที่มีด้านถ่วงน้ำหนัก ต้องการจัดหาในโครงสร้างข้อมูลสำหรับหน่วยเก็บของการถ่วงน้ำหนักเหล่านั้น รูป 7-13 แสดงให้เห็นกราฟแบบมีทิศทาง ที่มีด้านถ่วงน้ำหนัก ตัวอย่างเฉพาะนี้คือกราฟกิจกรรม (activity graph) แต่ละโหนดแทนหนึ่งเหตุการณ์ (event) และแต่ละด้านแทนหนึ่งงาน (task) ซึ่งการเสร็จสิ้นจะทำให้เกิดเหตุการณ์ถัดไป คือการเริ่มต้นของงานอื่น น้ำหนักของแต่ละด้านคือเวลาที่ต้องใช้ กราฟชนิดนี้ปกติใช้ใน ระบบจัดการโครางการ (project management systems)

การแทนที่สารบบโหนดสำหรับกราฟนี้แสดงให้เห็นในรูป 7-14 ระเบียนสำหรับ entry แต่ละด้านประกอบด้วย identifier ของโหนดปลายทาง (destination node), น้ำหนักของด้าน และพอยน์เตอร์ไปยังด้านถัดไปที่มีโหนดต้นฉบับ (source node) ตัวเดียวกัน



รูป 7-13 ตัวอย่าง activity graph



รูป 7-14 การแทนที่สารบบโหนดของรูป 7-13

การคำนวณด้าน(Edge Calculations)

การหาระดับชั้น (degree) ของโหนดในกราฟแบบไม่ระบุทิศทางนั้น ต้องใช้การนับจำนวนหน่วยข้อมูลบนรายการโยงของมัน ส่วน out-degree ของโหนดในกราฟแบบระบุทิศทางหาได้โดยการนับจำนวน หน่วยข้อมูลบนรายการโยงของมัน การหา in-degree ของโหนด i มีความซับซ้อนมากกว่ารายการโยงแต่ละชุดต้องถูกเข้าถึงเพื่อตรวจดูว่ามีโหนด i เป็นโหนดปลายทางหรือไม่ เพื่อความสะดวกในการหาโหนดหน้า(prior node) และการหา in-degrees, สารบบช่วย(auxiliary directory) ของด้านเข้ามายังโหนดอาจต้องเก็บไว้

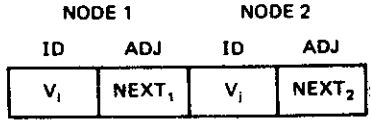
การวิเคราะห์กราฟบางชนิด กระทำกับการแทนที่เมทริกซ์ประชิดง่ายกว่ากระทำกับการแทนที่แบบโยง อย่างไรก็ตาม ถ้าเนื้อที่หน่วยเก็บเป็นหลักเกณฑ์เฉพาะเท่านั้นสำหรับการเลือก ดังนั้นโดยทั่วไปการแทนที่แบบโยงจะใช้เนื้อที่น้อยกว่า เว้นแต่ว่ากราฟมีการต่อกันอย่างมาก ถ้ากราฟมีโครงสร้างลบเลือนได้อย่างมาก เพราะฉะนั้นมันอาจต้องทำงานเพื่อใส่หรือลบด้านในการแทนที่เมทริกซ์ประชิด น้อยกว่าในการแทนที่สารบบโดยทั่วไป การเปลี่ยนแปลงหน่วยข้อมูล ของเมทริกซ์เร็วกว่าการใส่ (การลบออก)อีก หน่วยข้อมูลหนึ่งให้กับรายการโยง

การแทนที่หลายรายการโยง (Multi-list Representation)

ในการแทนที่หลายรายการโยงของโครงสร้างกราฟ มีสองส่วนคือสารบบของสารสนเทศโหนดและเซตของรายการโยงของสารสนเทศด้าน

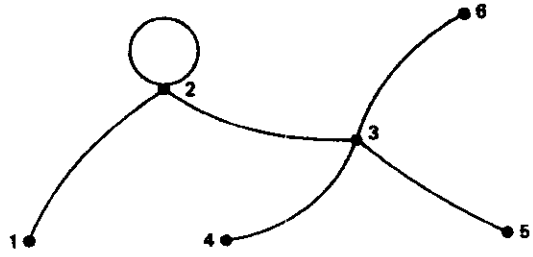
(In the multi-list representation of graph structures, there are again two parts : a directory of node information and a set of linked list of edge information.)

มีหนึ่งหน่วยข้อมูล ใน node directory สำหรับแต่ละโหนดของกราฟ หน่วยข้อมูลของสารบบสำหรับโหนด i ชี้ไปยังรายการประชิดแบบโยงสำหรับโหนด i ทั้งนี้แต่ละระเบียนของเนื้อที่รายการโยงปรากฏบนรายการประชิดสองชุด : หนึ่งชุดสำหรับโหนดที่แต่ละปลายของด้านที่ถูกแทนที่ ใช้โครงสร้างข้อมูลรูป 7-15 สำหรับหน่วยข้อมูล ด้าน (v_i, v_j) แต่ละตัว



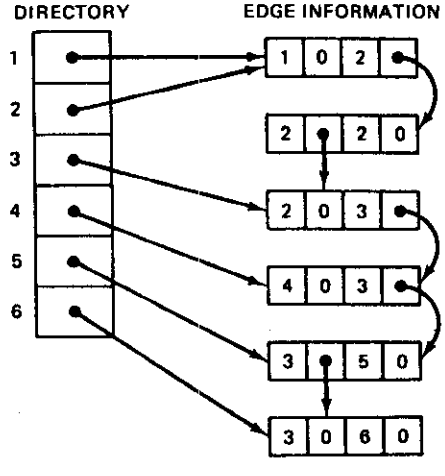
รูป 7-15 โครงสร้างข้อมูลสำหรับด้าน (v_i, v_j)

การแทนที่หลายรายการสำหรับกราฟตัวอย่างรูป 7-7 คือ รูป 7-16



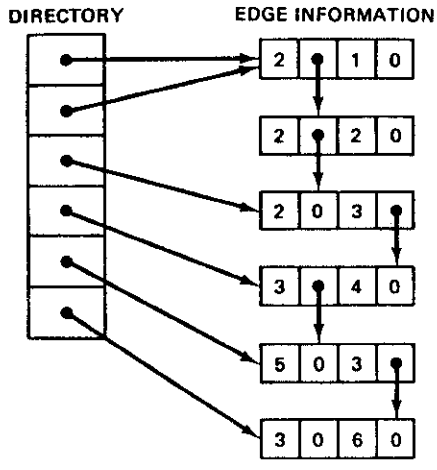
ตัวอย่างกราฟแบบไม่มีทิศทาง รูป 7-7

สิ่งนี้ไม่ใช่การแทนที่หลายรายการสำหรับกราฟนี้เท่านั้น มันขึ้นอยู่กับเซตของด้าน
{ (1,2), (2, 2), (2,3), (4, 3),(3, 5),(3, 6) }



รูป 7-16 การแทนที่หลายรายการของรูป 7-7

การแทนที่อีกทางเลือกหนึ่งขึ้นอยู่กับเซตของด้าน { (2, 1), (2, 2), (2, 3), (3, 4),(5, 3),(3, 6) } ซึ่งแสดงในรูป 7-17 ถ้าเป็นกราฟแบบมีทิศทางจะไม่มีความยืดหยุ่นในการเลือก identifier สำหรับด้านต่าง ๆ



รูป 7-17 การแทนที่หลายรายการอีกวิธีหนึ่ง

การถ่วงน้ำหนักด้านสามารถนำมาเก็บด้วยสารสนเทศอื่นถูกบันทึกสำหรับด้านเช่นที่กระทำมาแล้วในตัวอย่างต้น ๆ ของการแทนที่กราฟ การประกาศภาษา Pascal ของการแทนที่หลายรายการสำหรับกราฟที่มีการถ่วงน้ำหนัก order เท่ากับ 24 เขียนดังนี้

```
type    nodeid = 0 . . 24;
        nodetype = ↑edgeinfo;
        edgeptr = ↑ edgeinfo;
        edgeinfo = record node1 : nodeid;
                          node2 : nodeid;
                          adjlist1 : edgeptr;
                          adjlist2 : edgeptr;
                          weight : integer;
                   end;
        graph = array [1 . . 24] of nodetype;
```

7.5 การแวะผ่านกราฟ(Graph Traversal)

ในงานประยุกต์จำนวนมากมีความจำเป็นต้องเยี่ยม(visit)ทุกโหนดในกราฟ ตัวอย่างเช่น ต้องการพิมพ์รายการของเหตุการณ์ (events) ทั้งหมดในกราฟกิจกรรม (activity graph) ตัวอย่างรูป 7-13 หรือต้องการหาว่าเมืองใดบ้าง ที่รวมอยู่ในผังระยะทาง (distance chart) ตัวอย่างรูป 7-9 หรือต้องการหาระยะทางทั้งหมดระหว่างเมืองต่าง ๆ ในผังระยะทาง เทคนิคพื้นฐานของการแวะผ่านกราฟซึ่งจะนำเสนอมีสองชนิดคือในแนวกว้าง (breadth-first) และในแนวลึก (depth-first) ซึ่งมีความจำเป็นต้องระมัดระวังอย่างรอบคอบ เพื่อให้เยี่ยมแต่ละโหนดและแต่ละด้านเพียงครั้งเดียว การเยี่ยมหนึ่งโหนดซ้ำจะเป็นสาเหตุให้ปรากฏชื่อโหนดในรายการมากกว่าหนึ่งครั้ง การตามรอยหนึ่งด้านซ้ำจะนำไปสู่ขั้นตอนซ้ำ ผ่านกราฟระหว่างคู่ใด ๆ ของโหนดมีทางเดินหลายทาง อัลกอริทึมแวะผ่านกราฟปกติจะทำเครื่องหมาย (mark) แต่ละโหนดขณะที่เยี่ยมโหนด ซึ่งมีการทำเครื่องหมายมาก่อนแล้วไม่สามารถเยี่ยมซ้ำได้

อีกทางเลือกหนึ่งคืออัลกอริทึมการแวะผ่านกราฟ อาจทำเครื่องหมายแต่ละด้านขณะที่มันเดินตาม ด้านที่มีการทำเครื่องหมายก่อนหน้าแล้ว ไม่สามารถเป็นส่วนหนึ่งของอีกด้านหนึ่ง บิตเครื่องหมาย (Mark bits) สามารถถูกเก็บกับโหนดอื่น หรือสารสนเทศของด้าน

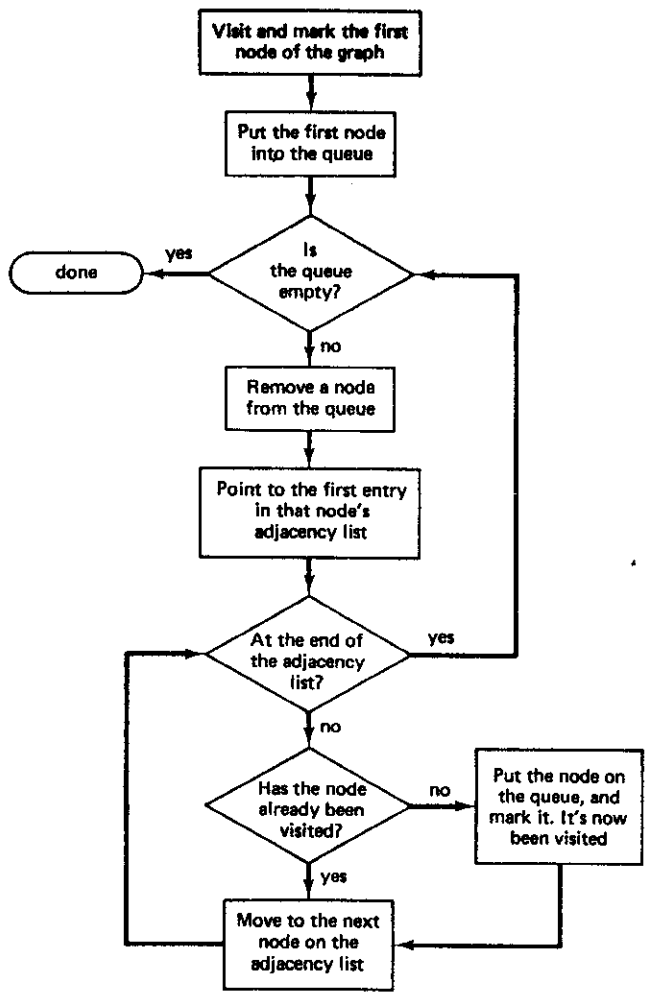
7.5.1 การแวะผ่านในแนวกว้าง (Breadth -- first traversal)

การแวะผ่านในแนวกว้างของกราฟนั้น ให้เลือกหนึ่งโหนดเป็นตำแหน่งเริ่มต้น เยี่ยมโหนดนี้แล้วทำเครื่องหมายไว้ จากนั้นโหนดทั้งหมดซึ่งยังไม่ได้ถูกเยี่ยม และเป็นโหนดประชิดกับโหนดนั้นจะถูกเยี่ยม และถูกทำเครื่องหมายไว้ในการเรียงแบบลำดับ สุดท้ายโหนดซึ่งยังไม่ถูกเยี่ยมซึ่งประชิดทันทีกับโหนดเหล่านี้ จะถูกเยี่ยมและทำเครื่องหมาย เช่นนี้เรื่อยไปจนกระทั่งกราฟทั้งหมดได้ถูกแวะผ่าน การแวะผ่านในแนวกว้างของกราฟรูป 7-13 ผลลัพธ์ในการเยี่ยมโหนดซึ่งมีลำดับดังนี้ :

1, 2, 3, 4, 5, 6, 7, 8, ลำดับ 1, 3, 2, 6, 5, 4, 7, 8 คืออันดับการเยี่ยมการแวะผ่านในแนวกว้างที่ถูกต้อง

อัลกอริทึมการแวะผ่าน ใช้แถวคอยเก็บโหนดในแต่ละระดับของกราฟขณะที่มันเยี่ยม โหนดซึ่งถูกเก็บเหล่านี้จากนั้นจะถูกกระทำทีละโหนด และโหนดประชิดของมันจะถูก

เยี่ยมทำเช่นนี้เรื่อยไป จนกระทั่งโหนดทั้งหมดถูกเยี่ยม เงื่อนไขการจบนี้จะมาถึงเมื่อแถวคอยว่าง ผังงานของอัลกอริทึมกำหนดให้แล้วในรูป 7-18



รูป 7-18 ผังงานของอัลกอริทึมการแวะผ่านกราฟในแนวกว้าง

อัลกอริทึมการทำให้เกิดผลในทางปฏิบัติด้วยภาษา Pascal เป็นดังนี้สมมติว่าใช้การแทนที่สารบบโหนดซึ่งได้อภิปรายมาแล้วพร้อมกับใส่บิตเครื่องหมายให้กับสารสนเทศโหนด แบบชนิดข้อมูลโครงสร้างแถวคอยและโปรซีเดอร์ insert และ remove นิยามมาแล้วในบทที่ 5

```
type    nodeid = 0. .ordergraph;
        edgeptr = ↑edgeinfo;
        nodetype = record  mark : 0 . . 1;
                           adjlist : edgeptr;
                   end;
        edgeinfo =  record  node : nodeid;
                            weight : integer;
                            next : edgeptr;
                    end;
        graphtype = array [1 . . ordergraph] of nodetype;
var     graph : graphtype;
        firstnode : nodeid;
```

โปรซีเดอร์การทำซ้ำเขียนเป็นโปรแกรมดังนี้

```
procedure breadth (fistnode : nodeid);
var     graph : queuestruct;
        savenode : nodeid;
        adjptr : edgeptr;
begin { visit firsnode here }
        graph[firstnode].mark :=1;
        insert(firstnode);
    while g.f <> 0 do
        begin  remove(savenode);
               adjptr := graph[savenode].adjlist;
               { visit node adjacent to savenode}
               while adjptr <> nil do
                       begin savenode := adjptr ↑.node;
```

```
                    if (graph[savenode].mark = 0)
                    then begin insert (savenode);
                              { visit savenode here}
                         graph[savenode].mark := 1
                         end;
                         adjptr := adjptr↑.next
               end;
          end;
     end;
```

7.5.2 การแวะผ่านในแนวลึก (Depth-first Traversal)

ขณะที่การแวะผ่านกราฟในแนวกว้างดำเนินไปทีละระดับ การแวะผ่านกราฟแนวลึก ขั้นแรกเดินตามทางเดินจากโหนดเริ่มต้นไปจนถึงโหนดสุดท้าย จากนั้นอีกหนึ่งทางเดินจากโหนดจุดเริ่มต้นไปจนจบ ทำเช่นนี้เรื่อย ๆ จนกระทั่งโหนดทั้งหมดถูกเยี่ยม

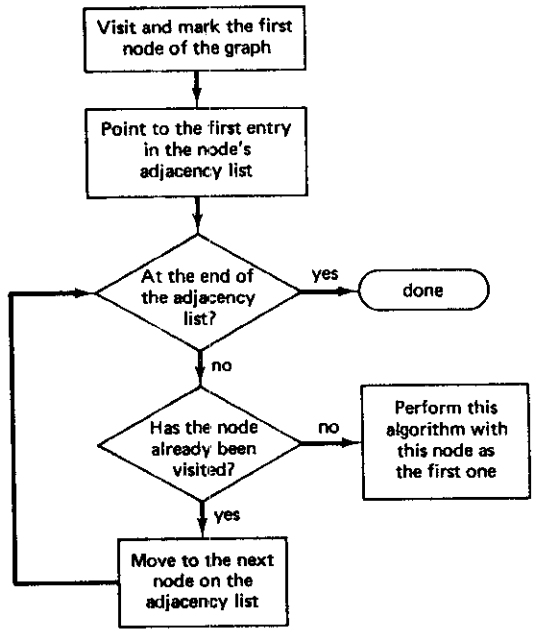
การแวะผ่านกราฟในแนวลึกรูป 7-13 ผลลัพธ์ในการเยี่ยมโหนดมีลำดับดังนี้ :

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

หนึ่งทางเดินดำเนินไปจนกระทั่งไม่มีโหนดที่ยังไม่ถูกเยี่ยมเหลืออยู่ จากนั้นอัลกอริทึมย้อนกลับมายังโหนดสุดท้ายที่ถูกเยี่ยมและมีโหนดประชิดซึ่งยังไม่ถูกเยี่ยม

ลำดับถูกต้องที่เท่าเทียมกันเป็นผลลัพธ์จากการแวะผ่านในแนวลึก ของกราฟตัวอย่างคือ 1, 3, 6, 7, 8, 2, 5, 4

การแวะผ่านในแนวกว้างซึ่งอธิบายมาแล้วง่ายต่อการใช้โปรซีเดอร์ทำซ้ำ ส่วนการแวะผ่านในแนวลึกตัวมันเองเป็นบทนิยามการเรียกซ้ำ ผังงานสำหรับการแวะผ่านในแนวลึกกำหนดให้แล้วในรูป 7-19



รูป 7-19 ผังงานของอัลกอริทึมการแวะผ่านกราฟในแนวลึกแบบเรียกตัวเอง

รหัส Pascal เพื่อทำให้เกิดผลการแวะผ่านสำหรับกราฟซึ่งถูกแทนที่โดยโครงสร้าง node-directory ที่ประกาศในหัวข้อก่อนหน้าเขียนดังนี้

```
procedure depth (var thisnode : nodeid);
var     savenode : nodeid;
        adjptr : edgeptr;
```

```
begin { visit thisnode here}
        graph[thisnode].mark := 1 ;
        adjptr := graph[thisnode].adjlist;
        while adjptr <> nil do
                begin
                        savenode := adjptr↑.node;
                        if (graph[savenode].mark = 0)
                        then depth(savenode);
                                adjptr := adjptr↑.next
                end;
end;
```

การเปรียบเทียบ (Comparison)

แบบฝึกหัดตอนท้ายของบทนี้แนะนำให้เราเขียนอัลกอริทึมแวะผ่านในแนวกว้าง และแวะผ่านในแนวลึก สำหรับกราฟซึ่งถูกแทนที่ด้วยโครงสร้างหลายรายการ (multi-list) และสำหรับกราฟซึ่งถูกแทนที่ด้วยเมทริกซ์ประชิด

ขณะนี้เราจะเปรียบเทียบเวลาสัมพัทธ์ (relative time)ที่ต้องใช้ของการแวะผ่านชนิดที่ถูกแทนที่แบบโยงและชนิดไม่ใช่แบบโยง

ในการแวะผ่านในแนวกว้างนั้น โหนดแต่ละตัวเข้าไปในแถวคอยโหนดเพียงครั้งเดียวดังนั้น while loop จึงถูกกระทำการ N ครั้ง เมื่อ N คืออันดับของกราฟ(order graph) ถ้ากราฟถูกแทนที่โดยโครงสร้างแบบโยง เพราะฉะนั้นเฉพาะโหนดเหล่านั้นเท่านั้นซึ่งประชิดกับโหนดที่อยู่ตอนหน้าของแถวคอย จะถูกตรวจสอบดังนั้น while loop ชั้นในถูกกระทำการทั้งหมด E ครั้งเมื่อ E คือจำนวนด้านในกราฟ การแวะผ่านในแนวกว้างของโครงสร้างแบบโยงคือ O (N * E)

อย่างไรก็ตาม ถ้ากราฟถูกแทนที่ด้วยเมทริกซ์ประชิดเพราะฉะนั้น while loop ชั้นในจะถูกกระทำครั้งเดียวสำหรับแต่ละโหนดอื่น ๆ ในกราฟ เนื่องจากทั้งแถวของเมทริกซ์ประชิดต้องถูกตรวจสอบ ดังนั้นการแวะผ่านในแนวกว้างของเมทริกซ์ประชิดคือ $O(N^2)$

ตรรกะชนิดเดียวกันสามารถประยุกต์ให้กับการแสดงให้เห็นว่า การแวะผ่านในแนวลึกของกราฟ ซึ่งถูกแทนที่โดยโครงสร้างแบบโยงคือ O(N * E) ด้วยเช่นกัน และการแวะผ่านในแนวลึกของกราฟซึ่งถูกแทนที่โดยเมทริกซ์ประชิดคือ $O(N^2)$ เช่นกัน

เพราะฉะนั้นการแทนที่แบบโยงจะประหยัดเนื้อที่ และให้การแวะผ่านที่สั้นกว่า เมื่อกราฟมีการต่อกันที่รัดกุมมากกว่า ตราบเท่าที่ E < N การแทนที่แบบโยงจะมีประสิทธิภาพมากกว่า โปรดจำไว้ว่าค่าเป็นไปได้สูงสุดสำหรับ E คือ N^2-N ซึ่งเกิดขึ้นเมื่อมีหนึ่งด้านระหว่างแต่ละคู่ของโหนด

เราจะทราบได้อย่างไรว่าควรจะนำการแวะผ่านในแนวลึกหรือการแวะผ่านในแนวกว้ามาปฏิบัติ การเลือกนี้ปกติกำหนดโดยตรรกะของการประยุกต์ไว้

7.6 ระยะที่ไปถึงและทางเดินสั้นที่สุด (Reachability and Shortest Paths)

การวิเคราะห์กราฟปกติเกี่ยวข้องกับทางเดินผ่านกราฟ มีปัญหาที่น่าสนใจหลายประการในการวิเคราะห์ทางเดินกราฟ เราจะพิจารณาเพียงสองหัวข้อดังนี้ :

ระยะที่ไปถึงและทางเดินสั้นที่สุดโดยใช้การแทนที่เมทริกซ์ประชิดจะสะดวกต่อการอภิปรายในตอนแรก

ในบางปัญหาการถ่วงน้ำหนักด้านมีความสำคัญ ตัวอย่างเช่น การหาทางเดินสั้นที่สุดระหว่างหนึ่งคู่ของเมือง ในกราฟรูป 7-9 ใช้ด้านถ่วงน้ำหนักซึ่งเป็นระยะทาง ในทางตรงกันข้าม การคำนวณมีสองโหนดต่อกันหรือไม่ กรณีนี้ไม่จำเป็นต้องพิจารณาด้านถ่วงน้ำหนัก เริ่มต้นสมมติว่าด้านถ่วงน้ำหนักทั้งหมดเท่ากับหนึ่ง

7.6.1 ระยะที่ไปถึง (Reachability)

รูป 7-20 แทนเมทริกซ์ประชิดสำหรับกราฟรูป 7-9 เมื่อหน่วยข้อมูล ที่ไม่ใช่ศูนย์ (non- zero entry) สำหรับหนึ่งคู่ของเมืองแสดงว่ามีทางเดินระหว่างสองเมืองนั้น โดยไม่มีเมืองชั้นกลาง การคูณเมทริกซ์ประชิดโดยตัวมันเองจะแสดงให้เห็นว่าคู่ของเมืองใดบ้างซึ่งมีทางเดินด้วยเมืองชั้นกลางหนึ่งเมือง

IT 204 211

i \ j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0
2	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0
3	0	0	0	0	1	1	0	0	1	1	1	0	0	0	0	0
4	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0
5	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0
9	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0
10	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0
11	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0
12	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
14	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0
15	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	1
16	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0

รูป 7-20 เมทริกซ์ประชิดของรูป 7-9

ให้ชื่อเมทริกซ์ประชิดว่า A การคูณเมทริกซ์ถูกใช้เพื่อหา A^2 นั่นคือสมาชิกตัวที่ ij ของ A^2 คือ ผลลัพธ์ของการคูณแถวที่ I ด้วยสดมภ์ที่ j ของ A

$$A^2_{ij} = \sum_{k=1}^{n} A_{ik} A_{kj}$$

A^2_{ij} = 1 ถ้ามีทางเดินความยาวเท่ากับ 2 จากโหนด i ไปโหนด j

A^2_{ij} = 0 กรณีอื่น ๆ

โดยทั่วไปการยกกำลังที่ m ของเมทริกซ์ประชิดฐานสองคือ

$$A^m_{ij} = \begin{cases} 1 & \text{ถ้ามีทางเดินความยาว m จากโหนด i ไปโหนด j} \\ 0 & \text{กรณีอื่น ๆ} \end{cases}$$

สำหรับตัวอย่าง city-distance , A^m จะแสดงให้เห็นว่ามีคู่ของเมืองใดบ้างซึ่งต้องหยุดที่ m-1 intermediate cities ปัญหาของการหาว่ามีคู่ของโหนดหรือไม่ซึ่งต่อกันดังนั้นสามารแก้ปัญหาโดยการหา

$$\sum_{t=1}^{N} A^t$$

ซึ่งผลลัพธ์ในเมทริกซ์เรียกว่า transitive closure ของ A ปกติใช้สัญลักษณ์ A^+ ที่จริงแล้วผลรวมสะสมไม่จำเป็นต้องกระทำ N ครั้ง เมื่อ N คืออันดับของกราฟ แต่ไม่ใช่เฉพาะจำนวนครั้งมากเท่ากับความยาวของทางเดินยาวที่สุดเท่านั้น

transitive closure ของ A เรียกว่า reachability matrix ของ A

7.6.2 ทางเดินสั้นที่สุด (Shortest Paths)

นอกจากเหนือจากการคำนวณว่า โหนด j สามารถไปถึงยังโหนด i หรือไม่ ปกติสิ่งที่น่าสนใจคือหาทางเดินสั้นที่สุดจากโหนด i ไปยังโหนด j การพัฒนาอัลกอริทึมทั่วไปจะทิ้งไว้เป็นแบบฝึกหัด แต่ในที่นี้เรานำข้อสังเกตสองข้อเกี่ยวกับทางเดินสั้นที่สุด ประเด็นเหล่านี้จะช่วยความพยายามในการพัฒนาของเราอ้างถึงกราฟรูป 7-9 อีกครั้งหนึ่ง ตารางข้างล่างนี้คือทางเดินสั้นที่สุดจากเมือง DNV ไปยังแต่ละเมืองอื่น ๆ รายชื่อเรียงอันดับซึ่งอัลกอริทึม shortest-path ตรวจพบได้

Path	Distance
DNV-ABQ	334
DNV-SLC	371
DNV-KNC	558
DNV-ABQ-ELP	334 + 229 =563
DNV-ABQ-PHX	334 + 550 = 664
DNV-SLC-SPK	371 + 550 = 921
DNV-SLC-SFO	371 + 600 = 671
DNV-KNC-ORD	558 + 414 = 972
DNV-ABQ-PHX-LAX	334 + 330 + 357 = 1021
DNV-ABQ-ELP-DFW	334 + 229 + 572 = 1135
DNV-SLC-SPK-SEA	371 + 550 + 229 = 1150

DNV-KNC-ORD-CVC	558 + 414 + 252 = 1224
DNV-ABQ-ELP-DFW-HST	334 + 229 + 572 + 225 = 1360
DNV-KNC-ORD-CVG-NSH	558 + 414 + 252 + 238 = 1462
DNV-ABQ-ELP-DFW-NOR	334 + 229 + 572 + 443 = 1578

มีทางเดินที่เป็นไปได้หลายวิธีระหว่างคู่ของเมืองต่าง ๆ โปรดสังเกตว่า

1. ทางเดินสั้นที่สุดถูกตรวจพบในการเรียงอันดับ แบบไม่ใช่จากมากไปหาน้อยของระยะทาง ตัวอย่างเช่น โหนดปลายทางถัดไป คือ โหนดที่มีระยะทางต่ำสุดจากโหนดต้นฉบับของโหนดทั้งหมดซึ่งยังไม่ถูกเลือก
2. ทางเดินสั้นที่สุดไปยังโหนดปลายทางถัดไปนั้นผ่านโหนดซึ่งมีการถูกเลือกเรียบร้อยแล้ว ถ้าด้านถ่วงน้ำหนักถูกแทนที่ต้นทุน (costs) ไม่ใช่ถูกแทนที่ระยะทาง (distances) เพราะฉะนั้นอัลกอริทึมเหมือนเดิม สามารถนำมาใช้เพื่อตรวจหาทางเดินที่มีราคาถูกที่สุดถ้าด้านถ่วงน้ำหนักถูกนำมาแทนที่ความเร็ว (เช่น กิโลบิต/วินาที) ดังนั้น อัลกอริทึมสามารถดัดแปรได้โดยง่ายเพื่อตรวจหาทางเดินเร็วที่สุด

7.7 ทางเดินวิกฤต (Critical Paths)

กราฟรูป 7-13 แทนกราฟกิจกรรม (activity graph) โครงการเกือบทั้งหมดสามารถถูกแทนที่โดยกราฟเช่นนี้ เทคนิคที่ดีได้ถูกพัฒนาเพื่อช่วยเหลือในการประเมินผลและวิเคราะห์กราฟกิจกรรม ตัวอย่างเช่น วิธีทางเดินวิกฤต (Critical Path Method (CPM)) เทคนิคการประเมินผลความสามารถและการทบทวน (Performance Evaluation and Review Technique (PERT)) และการจัดลำดับการจัดสรรทรัพยากรและการจัดลำดับหลายโครงการ (Resource Allocation and Muti-project Scheduling (RAMPS))

ในที่นี้จะแนะนำเฉพาะด้านพื้นฐานของการวิเคราะห์กราฟกิจกรรมโครงสร้างกราฟของโครงการสามารถแสดงให้เห็นได้ว่า หลายงานสามารถกระทำการขนานกันไปได้ ตัวอย่างเช่นงานจากเหตุการณ์ 2 และ 3 สามารถประมวลผลไปพร้อมกันในรูป 7-13 ประเด็นของการจัดการของโครงการคือการคำนวณหาว่าเหตุการณ์ใดจะวิกฤตเพื่อให้เสร็จสิ้นภายในเวลาของโครงการ

เหตุการณ์จะวิกฤต (critical) ถ้าการเลื่อนเวลาการเสร็จสิ้นของการจัดลำดับงานทำให้การเสร็จสิ้นของโครงการทั้งหมดถูกทำให้ช้าลงเวลาการเสร็จสิ้นที่เป็นไปได้สั้นที่สุดสำหรับโครงการคือทางเดินยาวที่สุดผ่านกราฟ ทางเดินยาวที่สุดเรียกว่า **ทางเดินวิกฤต**

(critical path) เหตุการณ์วิกฤตวางอยู่บนทางเดินนี้ ความยาวของทางเดินนี้เรียกว่าเวลาทางเดินวิกฤต (The length of this path is called the critical path time, (CPT)) ระบบสารสนเทศอาจจะรวมโปรแกรมเพื่อหาทางเดินวิกฤตและ CPT เช่นเดียวกับการ draft และการดัดแปรผังงานกราฟกิจกรรม

วิธีหนึ่งเพื่อตรวจหาทางเดินวิกฤตและคำนวณหาเวลาที่ยอมให้เลื่อนไปสำหรับเหตุการณ์ที่ไม่ได้อยู่บนทางเดินวิกฤตคือ เพื่อหาเวลาเร็วที่สุดของเหตุการณ์ i, EST_i หมายถึงเวลาปล่อยที่เป็นไปได้เร็วที่สุดสำหรับเหตุการณ์ (The earliest start time of event i, EST_i is the earliest possible trigger time for the event.)

สำหรับกราฟรูป 7-13

Event	EST
1	0
2	6
3	3
4	8
5	10
6	12
7	14
8	16

EST_i คำนวณจากทางเดินยาวที่สุดไปยังโหนด I จากโหนดเริ่มต้น โปรดสังเกตว่างานจากทั้งโหนด 5 และ 6 ต้องเสร็จสิ้นทั้งคู่ก่อนเหตุการณ์ 7 จะถูกปล่อยออกมาเป็นแบบทางการมากขึ้นคือ

$$EST_i = \max_{J \in P(I)} \{ EST_j + T(j, i) \}$$

เมื่อ P(i) คือเซตของ immediate predecessor ไปยังโหนด

T(j, i) คือ น้ำหนักของด้านจากโหนด j ไปยังโหนด i

เวลาเริ่มต้นช้าที่สุดของเหตุการณ์ i, LST_i หมายถึงเวลาเป็นไปได้ช้าที่สุดซึ่งเหตุการณ์ I สามารถปล่อยออกไปด้วยการเสร็จสิ้น กราฟในเวลาทางเดินวิกฤต เซตของ LST, คำนวณย้อนกลับจากเหตุการณ์สุดท้าย สำหรับกราฟรูป 7-13

Event	LST
1	0
2	13
3	3
4	15
5	11
6	12
7	14
8	16

อย่างเป็นทางการ

$$LST_i = \min_{j \in s(i)} \{ LST_j - T(i, j) \}$$

เมื่อ s(i) คือเซตของ immediate successors ไปยังโหนด i ความแตกต่างระหว่าง LST_i และ EST_i คือ allowable slippage ของเหตุการณ์ i

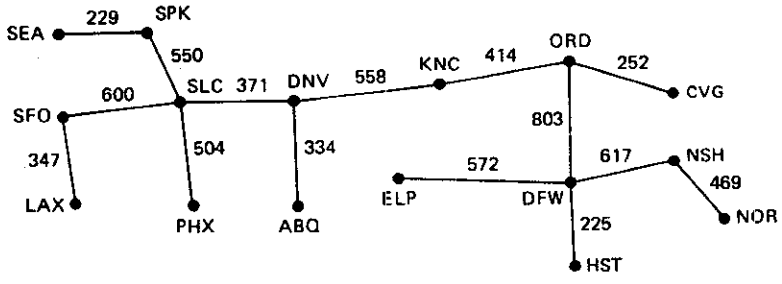
Event	LST-EST
1	0
2	7
3	0
4	7
5	1
6	0
7	0
8	0

ดังนั้นเหตุการณ์วิกฤตถูกแทนที่ด้วยโหนด 1, 3, 6, 7 และ 8 ซึ่งเป็นทางเดินยาวที่สุดของกราฟ สิ่งเหล่านี้คือเหตุการณ์ซึ่งจำเป็นต้องถูกควบคุม เพื่อให้โครงสร้างทันเวลา โปรดสังเกตว่าถ้าเหตุการณ์ไม่วิกฤตถูกเลื่อนไปเพราะฉะนั้นทางเดินวิกฤตผ่านกราฟจะถูกเปลี่ยนแปลง

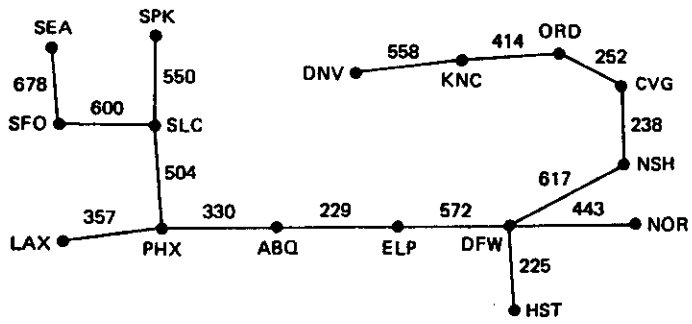
7.8 ต้นไม้แบบทอดข้าม(Spanning Trees)

ต้นไม้แบบทอดข้าม หมายถึง ต้นไม้ซึ่งประกอบด้วยโหนดทั้งหมดของกราฟและไม่มีโหนดอื่นๆ (A Spanning tree is a tree that contains all the nodes of graph and has no other nodes.)

งานประยุกต์หลายอย่างเรียก identification ของต้นไม้แบบทอดข้าม สำหรับกราฟเชื่อมโยง(connected graph) รูป 7-21 และรูป 7-22 แสดงต้นไม้แบบทอดข้ามสองรูปจากต้นไม้แบบทอดข้ามหลายชุดของกราฟรูป 7-9 ต้นไม้แบบทอดข้ามแต่ละชุดแสดงวิธีที่จะวางแผนเส้นทางรถบรรทุกเพื่อให้แต่ละเมืองได้รับบริการ



รูป 7-21 ต้นไม้แบบทอดข้ามสำหรับรูป 7-9



รูป 7-22 ต้นไม้แบบทอดข้ามอีกชุดหนึ่งสำหรับรูป 7-9

สิ่งที่น่าสนใจอันดับแรกคือ การระบุ(identification)ของต้นไม้แบบทอดข้ามต่ำสุด สำหรับกราฟ ต้นทุนของต้นไม้แบบทอดข้าม หมายถึง ผลบวกของน้ำหนักด้านของมัน (The cost of a spanning tree is the sum of its edge weights)

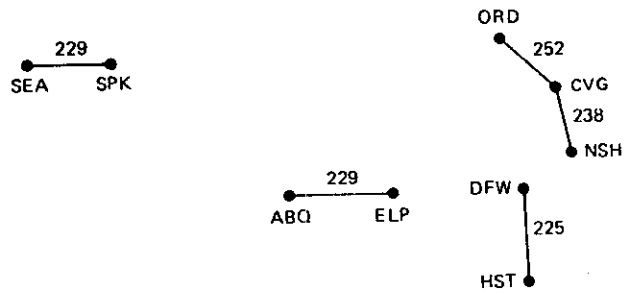
ต้นทุนของต้นไม้แบบทอดข้ามของรูป 7-21 คือ 6,845 ไมล์ ส่วนต้นทุนของต้นไม้แบบทอดข้ามของรูป 7-22 คือ 6,576 ไมล์ ทั้งสองรูปนี้ไม่ใช่ต้นไม้แบบทอดข้ามที่มีต้นทุนน้อยที่สุด

อัลกอริทึมของครูกอล(Kruskal's Algorithm)

วิธีหนึ่งในการหาต้นไม้แบบทอดข้ามต่ำสุด คือ อัลกอริทึมซึ่งพัฒนาโดย Kruskal อัลกอริทึมนี้พิจารณาการเป็นเซตย่อย(inclusion) ด้านของกราฟในอันดับ โดยการเพิ่มต้นทุน ; หนึ่งด้านถูกนับเป็นเซตย่อย ถ้ามันไม่มีรูปแบบของวัฏจักร

วัฏจักร หมายถึง มีสองทางเดินระหว่างคู่ต่างๆ ของโหนดอยู่ การประยุกต์อัลกอริทึมของ Kruskal ให้กับกราฟรูป 7-9 ดำเนินการดังนี้ :

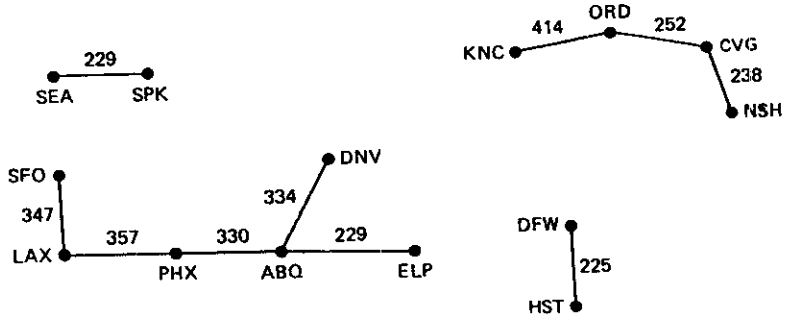
1. Lowest cost : DFW-HST(225)
2. Lowest cost : ABQ-ELP(229)
3. Lowest cost : SEA-SPK(229)
4. Lowest cost : CVG-NSH(238)
5. Lowest cost : ORD-CVG(252)



รูป 7-23(a) ขั้นตอนแรกในการหาต้นไม้แบบทอดข้ามของรูป 7-9

6. Lowest cost : PHX-ABQ(330)
7. Lowest cost : DNV-ABQ(334)
8. Lowest cost : SFO-LAX(347)
9. Lowest cost : LAX-PHX(357)
10. Lowest cost : KNC-ORD(414)

ขณะนี้ต้นไม้แบบทอดข้ามต่ำสุด แสดงให้เห็นในรูป 7-23(b)



รูป 7-23(b) ขั้นตอน intermediate ในการหาต้นไม้แบบทอดข้ามของรูป 7-9

11. Lowest cost : DFW-NOR(443)

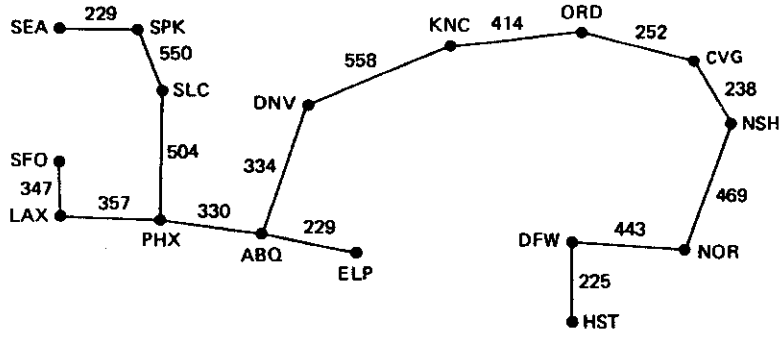
12. Lowest cost : NSH-NOR(469)

13. Lowest cost : PHX-SLC(504)

14. Lowest cost : SLC-SPK(550)

15. Lowest cost : DNV-KNC(558)

เราโชคดีอย่างมากในการหลีกเลี่ยงวัฏจักรต้นไม้แบบทอดข้ามต่ำสุด แสดงไว้ใน รูป 7-23(c)



รูป 7-23(c) ต้นไม้แบบทอดข้ามต่ำสุดของรูป 7-9

ที่จริงขณะนี้เรารวมโหนดทั้งหมดและต้นไม้แบบทอดข้ามต่ำสุดเสร็จบริบูรณ์ด้วย ต้นทุนเท่ากับ 5,479

สมมติว่า ขณะนั้นโหนดทั้งหมดยังไม่นับรวมการพิจารณาด้านที่มีต้นทุนต่ำสุดถัดไป(ELP-DFW) ตรวจพบว่าเป็นวัฏจักร ด้านนั้นจะถูกปฏิเสธและด้านที่มีต้นทุนต่ำสุดถัดไปจะถูกนำมาพิจารณา และทำต่อไปเรื่อยๆจนกระทั่งโหนดทั้งหมดถูกนับรวมแล้ว

ดังนั้นต้นไม้แบบทอดข้ามต่ำสุด จึงเป็นวิธีที่ต้นทุนน้อยที่สุด(least-cost way)เพื่อเชื่อมโยงโหนดต่างๆของกราฟ ต้นทุนอาจถูกวัดในเทอมของระยะทาง , เวลา , เงินและอื่นๆ ขึ้นอยู่กับหน่วยน้ำหนักของด้าน(edge weight units)

บทสรุป

ในบทนี้ได้อภิปรายกราฟในรายละเอียดบางอย่าง สิ่งแรกแนะนำแนวคิดพื้นฐานของโครงสร้างข้อมูลกราฟ จากนั้นเทคนิคหลักสามวิธีสำหรับการแทนที่กราฟ ได้แก่ : การแทนที่เมทริกซ์ประชิดและอีกสองแบบของการแทนที่แบบโยง -สารบบโหนดและรายการหลายชุด การใช้เมทริกซ์ประชิด ทำให้อัลกอริทึมการวิเคราะกราฟห์น้อยที่สุดเป็น $O(N^2)$ ในขณะที่ใช้การแทนที่แบบโยงทำให้อัลกอริทึมการวิเคราะห์กราฟเป็น O(N*E) ตัวอย่าง เช่น การประกาศของกราฟซึ่งกำหนดใน COBOL และ Pascal

เทคนิคการแวะผ่านกราฟพื้นฐานสองวิธีได้แก่ : การค้นในแนวกว้าง(breadth-first) และการค้นในแนวลึก(depth-first) อัลกอริทึมเหล่านี้ต้องให้ความระมัดระวังก่อนเพื่อเยี่ยมแต่ละโหนดเพียงหนึ่งครั้งเท่านั้น โดยการทำเครื่องหมายแต่ละโหนดว่ามันถูกเยี่ยมแล้ว หรือแต่ละด้านว่ามันถูกแวะผ่านแล้ว การแวะผ่านในแนวกว้าง โดยทั่วไปถูกโปรแกรมเป็นโปรซีเดอร์แบบทำซ้ำ(iterative procedure) ในขณะที่การแวะผ่านในแนวลึกถูกโปรแกรมเป็นโปรซีเดอร์เรียกซ้ำ(recursive procedure)

ปัญหาการวิเคราะห์การเดินหลายวิธีได้ถูกอภิปรายพลัง(powers)ของเมทริกซ์ประชิดของกราฟสามารถนำมาใช้เพื่อตรวจจับ(detect)ทางเดินของความยาวที่กำหนดให้ระหว่างคู่ของโหนดและเพื่อคำนวณ ถ้าหนึ่งโหนดสามารถเข้าถึงได้จากอีกโหนดหนึ่ง ตัวอย่างถูกนำมาใช้เพื่อแสดงให้เห็นถึงอัลกอริทึม สำหรับการตรวจหาทางเดินสั้นที่สุดระหว่างโหนดหนึ่งกับโหนดอื่นๆ การวิเคราะห์ทางเดินวิกฤตถูกแนะนำ รวมทั้งแนวคิดของเวลาเริ่มต้นเร็วที่สุดและเวลาเริ่มต้นช้าที่สุด สุดท้ายแนะนำ ต้นไม้แบบทอดข้ามและเทคนิค (อัลกอริทึมของ Kruskal) สำหรับการระบุถึงต้นไม้แบบทอดข้ามต่ำสุดของกราฟ

แบบฝึกหัด

1. สมมติว่ากราฟระบุทิศทางรูปหนึ่งมีอันดับเท่ากับ n ถูกแทนที่ด้วยเมทริกซ์ประชิด โดยที่ graph(i,j) = 1 ถ้ามีหนึ่งด้านเชื่อมโยงโหนด i และโหนด j และ graph(i,j)= 0 ในกรณีอื่นๆ จงหานิพจน์สำหรับคำนวณสิ่งต่อไปนี้
   (a) ระดับขั้นเข้าของโหนด i
   (b) ระดับขั้นออกของโหนด i
2. จงวาดรูปกราฟระบุทิศทาง แล้วตอบคำถามข้างล่างนี้
   (a) เมทริกซ์ประชิดคืออะไร
   (b) เมทริกซ์ซึ่งการเข้าถึงได้ของมันคืออะไร
   (c) จงแทนกราฟโดยใช้โครงสร้าง node-directory
   (d) จงแทนกราฟโดยใช้โครงสร้าง multi-list
   (e) จงแวะผ่านกราฟโดยใช้วิธีการค้นในแนวกว้าง
   (f) จงแวะผ่านกราฟโดยใช้วิธีการค้นในแนวลึก
   (g) ต้นไม้แบบทอดข้ามของกราฟคืออะไร
3. ในกราฟเชิงเดียวมีวัฏจักรได้หรือไม่ ถ้าได้ให้ยกตัวอย่าง ถ้าไม่ได้ ให้อธิบายพร้อมทั้งยกเหตุผลประกอบ
4. จงวาดรูปกราฟหลาย ๆ รูป สำหรับกราฟแต่ละชนิดให้ตอบคำถามข้างล่างนี้ว่าเป็นกราฟชนิดใด
   (a) อวัฏจักร
   (b) เชื่อมโยง
   (c) ระบุทิศทาง
   (d) เชิงเดี่ยว
   (e) อันดับของกราฟคืออะไร
   (f) ระดับขั้นเข้าของกราฟคืออะไร
   (g) ระดับขั้นออกของกราฟคืออะไร
   (h) ระดับขั้นของโหนดแต่ละตัว
5. โครงสร้างข้อมูลชนิดใดซึ่งปกติเหมาะสมสำหรับเก็บเมทริกซ์ประชิด

6. การแทนที่แบบโยงชนิดหนึ่งของกราฟเก็บ(maintains)สารบบของโหนดและสมนัยกับรายการโยงของด้าน จงพัฒนาการแทนที่แบบโยงอีกทางเลือกหนึ่งซึ่งเก็บสารบบของด้านและสมนัยกับรายการโยงของโหนด จากนั้นจงวิเคราะห์หน่วยเก็บที่ต้องใช้ของการแทนที่ ให้เปรียบเทียบหน่วยเก็บที่ต้องใช้นี้กับหน่วยเก็บที่ต้องใช้ของการแทนที่แบบโยงที่กำหนดให้ในบทนี้
7. จงเขียนโปรแกรมเพื่อทำสิ่งต่อไปนี้
   (a) แวะผ่านการค้นในแนวกว้าง
   (b) แวะผ่านการค้นในแนวลึกของกราฟซึ่งแทนที่ด้วยโครงสร้างหลายรายการ(multi-list)
8. จงเขียนโปรแกรมทำสิ่งต่อไปนี้
   (a) แวะผ่านการค้นในแนวกว้าง และ
   (b) แวะผ่านการค้นในแนวลึกของกราฟซึ่งแทนที่ด้วยโครงสร้างหลายรายการ(multi-list)
9. จงเขียนโปรแกรมเพื่อหาทางเดินวิกฤตของกราฟถ่วงน้ำหนักซึ่งแทนที่ด้วย
   (a) เมทริกซ์ประชิด
   (b) โครงสร้างสารบบโหนด(node-directory)
   (c) โครงสร้างหลายรายการ(multi-list)
10. จงพัฒนากราฟกิจกรรม(activity graph)สำหรับกิจกรรมของท่านเอง สำหรับสัปดาห์หน้าหรือเดือนหน้า จากนั้นจงหาทางเดินวิกฤตตลอดกราฟ
11. จงเขียนโปรแกรมเพื่อสร้างเซตของทางเดินสั้นที่สุดจากโหนด i ไปยังโหนดอื่นๆทั้งหมดในกราฟที่มีอันดับเท่ากับ n สมมติว่ากราฟถูกแทนที่ด้วย
    (a) เมทริกซ์ประชิด
    (b) โครงสร้าง node-directory

และ(c) โครงสร้าง multi-list