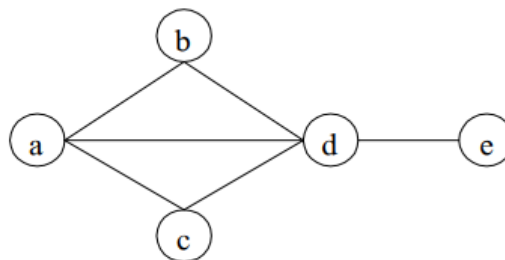


3.3 โครงสร้างกราฟ (Graph)

3.3.1 องค์ประกอบของกราฟ

กราฟเป็นแผนผังซึ่งประกอบไปด้วย จุดและเส้นต่อยังกัน เพื่อแทนโครงสร้าง ปัญหาหรือจำลองภาพบางอย่างโดยทั่วไปกราฟ (Graph หรือใช้อักษรย่อ G) จะประกอบไปด้วยกลุ่มของจุด (Vertex หรือใช้อักษรย่อ V) และกลุ่มของเส้นโยง (Edge หรือใช้อักษรย่อ E) กราฟสามารถเขียนในรูปความสัมพันธ์ ของ V และ E ได้ดังนี้

$$G = (V, E) \quad \begin{array}{ll} V \text{ (Vertex or Node)} & = \text{Set ของจุดยอด} \\ E \text{ (Edge)} & = \text{Set ของเส้นโยง} \end{array}$$



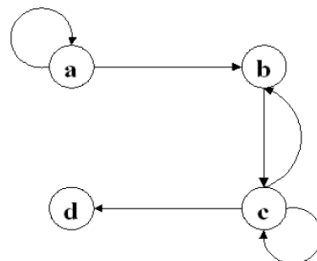
$$G = (V, E) \quad V = \{a, b, c, d, e\} \quad E = \{(a, b), (a, c), (b, d), (c, d), (d, e)\}$$

รูปที่ 3.48 แสดงลักษณะของกราฟ

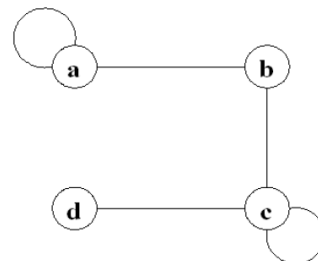
โดยที่จะเรียกจุดสองจุดที่มีเส้นโยงติดกันว่า ติดกัน (Adjacent) ตัวอย่างเช่นจุด a และ b เป็นจุดติดกัน แต่จุด a และ d ไม่ได้ติดกัน และเรียกสภาพที่เส้นโยงต่อจุดสองจุด 1 และ j ว่า เส้นโยงเกิดกับ (Incident) ตัวอย่างเช่น เส้นโยง (a,b) เกิดกับจุด a และ b

โดยลักษณะของกราฟทั่วไปแบ่งกราฟเป็น 2 ลักษณะ คือ

1. กราฟแบบมีทิศทาง (Directed graph)
2. กราฟแบบไม่มีทิศทาง (Undirected graph)



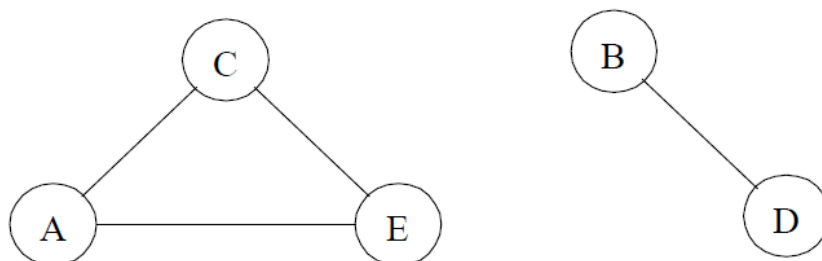
ก) กราฟแบบมีทิศทาง



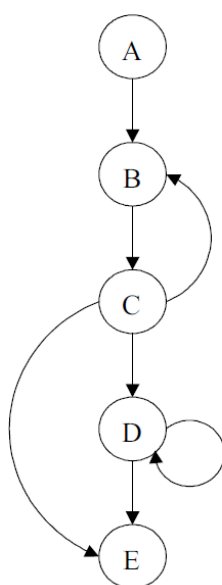
ข) กราฟแบบไม่มีทิศทาง

รูปที่ 3.49 แสดงกราฟ 2 ลักษณะ

3.3.2 การแทนกราฟโดยอาเรย์



รูปที่ 3.50 ตัวอย่างของกราฟแบบไม่มีทิศทาง



รูปที่ 3.51 ตัวอย่างของกราฟแบบมีทิศทาง

กราฟทั้งสองแบบจากตัวอย่างในรูป นี้สามารถเก็บในคอมพิวเตอร์โดยใช้อาเรย์สอง มิติ $A(n,n)$ ซึ่งมี 1 ช่อง ในแต่ละช่องจะมีค่า 0 หรือ 1 ซึ่งขึ้นอยู่กับข้อกำหนดดังนี้

$$A(i,j) = \begin{cases} 1 & \text{ถ้ามีเส้นโยงจาก } i \text{ ไป } j \\ 0 & \text{ถ้าไม่มีเส้นโยงจาก } i \text{ ไป } j \end{cases}$$

การใช้อาเรย์ 2 มิติที่แทนกราฟ จะได้ดังรูป 3.52

	A	B	C	D	E
A	0	0	1	0	1
B	0	0	0	1	0
C	1	0	0	0	1
D	0	1	0	0	0
E	1	0	1	0	0

	A	B	C	D	E
A	0	1	0	0	0
B	0	0	1	0	0
C	0	1	0	1	1
D	0	0	0	1	1
E	0	0	0	0	0

รูปที่ 3.52 อาเรย์ 2 มิติแทนกราฟ

การวางโครงสร้างโปรแกรมการแทนกราฟโดยอาเรย์

การแทนกราฟโดยอาเรย์ สามารถระบุค่าต่าง ๆ ลงในโครงสร้างอาเรย์ตอนประกาศ ตัวแปรได้เลย การแสดงผลสมาชิกของเวอร์เท็กซ์ (Vertex) ใช้การอ่านข้อมูลจากอาเรย์ 1 มิติ ธรรมดา ส่วนการแสดงผลสมาชิกของเอดจ์ (Edge) จะอ่านข้อมูลอาเรย์ 2 มิติที่เก็บโครงสร้าง กราฟโดยดูว่าถ้าตำแหน่งนั้นเป็น “1” ก็แสดงชื่อของแถว (Row) และคอลัมน์ (Column) ของอาเรย์ ก็จะได้สมาชิกของเอดจ์ทั้งหมด

3.3.2.1 โปรแกรมการแทนกราฟโดยอาเรย์

```
/* Program create Graph structure by "2 dimension array
representation method".
Can show the result of graph are ..
1. Set of vertex
2. Set of Edge
Note.- can use both Undirected graph and Directed graph
*/
#include <stdio.h> //use printf
#include <conio.h> //use getch
#define MaxNode 6 // Define Max Node of Graph
int graph[MaxNode][MaxNode] = {
    {0,1,1,1,0,0},
    {1,0,1,0,1,0},
    {1,1,0,0,0,0},
    {1,0,0,0,1,1},
    {0,1,0,1,0,0},
    {0,0,0,1,0,0}
}; //Declare array and keep data of graph
char NodeName[MaxNode] = {'A','B','C','D','E','F'}; //Keep Node Name
void DispArray2D() //Display value in 2D Array
{
    int i,j; //i=Row, j=Column
```

```

printf(" ");
for (j=0;j<=MaxNode;j++) //Display column name of array
    printf("%c ",NodeName[j]);
printf("\n"); //line feed
for (i=0;i<MaxNode;i++) //row loop
{
    printf("%c ",NodeName[i]); //Display row name of array
    for (j=0;j<MaxNode;j++) //column loop
        printf("%d ",graph[i][j]); //Display value path
    printf("\n");
}
}
void DispSetOfVertex() //Display set of Vertex
{
    int i;
    printf("\nSet of Vertex = {");
    for (i=0;i<MaxNode;i++)
    {
        printf("%c",NodeName[i]); //Display each node name
        if(i != MaxNode-1)
            printf(",");
    }
    printf("}\n");
}
void DispSetOfEdge() //Display set of Edge
{
    int i,j;
    printf("\nSet of Edge = {");
    for (i=0;i<MaxNode;i++) //row loop
        for (j=0;j<MaxNode;j++) //column loop
        {
            if(graph[i][j]==1)
                printf("(%c,%c)",NodeName[i],NodeName[j]); //Show each Edge
        }
    printf("}\n");
}
int main()
{
    printf("GRAPH (ADJACENCY MATRIX REPRESENTATION METHOD)\n");
    printf("=====\n");
    DispArray2D();
    DispSetOfVertex();
    DispSetOfEdge();
    getch();
    return(0);
} //End Main

```

```

GRAPH (ADJACENCY MATRIX REPRESENTATION METHOD)
=====
  A B C D E F
A 0 1 1 1 0 0
B 1 0 1 0 1 0
C 1 1 0 0 0 0
D 1 0 0 0 1 1
E 0 1 0 1 0 0
F 0 0 0 1 0 0

Set of Vertex = {A,B,C,D,E,F}
Set of Edge = {(A,B), (A,C), (A,D), (B,A), (B,C), (B,E), (C,A), (C,B), (D,A), (D,E), (D,F), (E,B), (E,D), (F,D), }

```

รูปที่ 3.53 แสดงผลการทำงานของโปรแกรมการแทนกราฟโดยอาเรย์

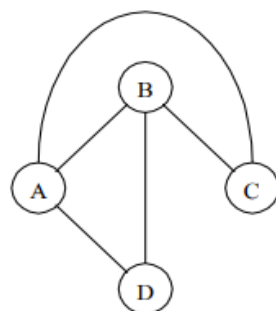
3.3.3 การแทนกราฟโดยลิงค์ลิสต์ (Linked List)

การแทนกราฟด้วยอาเรย์ 2 มิตินั้นจะมีข้อเสีย คือ ถ้ากราฟมี 1 จุดจะต้องใช้อาเรย์ 2 มิติขนาด 1 ช่องซึ่งทำได้ไม่สะดวก และถ้าอาเรย์ 2 มิติที่สร้างขึ้นมาแทนกราฟมีค่าเป็น 0 อยู่มาก แต่ยังคงต้องทำการจองพื้นที่ขนาด 1 อยู่จึงทำให้ต้องมีการจองพื้นที่ในหน่วยความจำมากเกินไปจนเกินไป ดังนั้นการใช้ลิงค์ลิสต์มาแทนกราฟจะเป็นการช่วยลด พื้นที่ในหน่วยความจำที่ไม่เกิดประโยชน์ลงได้มาก จนประมาณได้เท่ากับจำนวน “จุด” รวม กับจำนวน “เส้นโยง” ในกราฟนั้น ๆ

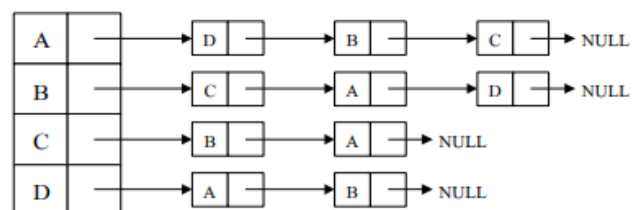
ในการแทนกราฟโดยใช้ลิงค์ลิสต์นั้นสามารถกระทำได้หลายวิธีซึ่งผู้ที่เข้าใจในหลักการสามารถที่จะประยุกต์ขึ้นได้โดยใช้เส้นโยง ดังนั้นในที่นี้จะยกตัวอย่างการแทนกราฟ โดยใช้ลิงค์ลิสต์ในแบบที่นิยมใช้กันในปัจจุบัน

การแทนกราฟโดยใช้รายการที่ติดกัน (Adjacency lists)

การแทนแบบนี้ แต่ละแถวของอาเรย์ 2 มิติจะกลายเป็นลิงค์ลิสต์ หัวแถวจะเป็น จุดของกราฟ ดังนั้นถ้ากราฟมี 1 จุดก็จะมี n แถว จุดในแต่ละแถวก็คือจุดที่อยู่ข้างเคียง หรือเป็นเพื่อนบ้านของจุดหัวแถวนั้นนั่นเอง ดังรูป 3.54

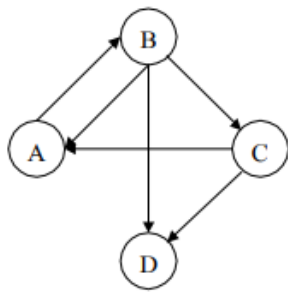


(ก) กราฟ G1

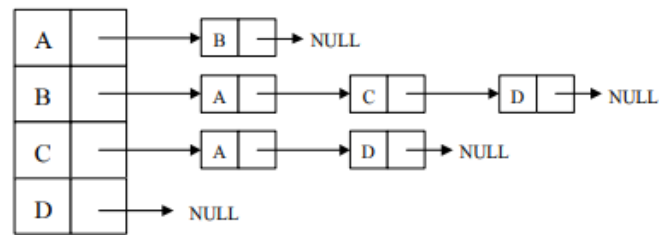


(ข) Adjacency list สำหรับกราฟ G1

รูปที่ 3.54 การแทนกราฟ G1 โดยใช้ Adjacency lists



(ก) ไดโกราฟ G2



(ข) Adjacency list สำหรับ G2

รูปที่ 3.55 การแทนไดโกราฟ G2 โดยใช้ Adjacency lists

การวางโครงสร้างโปรแกรมการแทนกราฟโดยลิงค์ลิสต์

1. แทนโครงสร้างกราฟลงในอาเรย์ (การเริ่มต้นแทนแบบนี้จะสามารถเปลี่ยน โครงสร้างกราฟให้เป็นลักษณะอื่นได้ง่าย)

2. นำโครงสร้างกราฟใน (1) ไปเก็บแบบ Adjacency List โดยการเขียนโค้ดโดยมี ขั้นตอนดังนี้

- สร้าง Head Node ตามจำนวนของ Vertex โดยให้แต่ละลิงค์ของ Head มี Pointer เป็นของตนเอง ซึ่งในที่นี้จะใช้ Array Pointer ชื่อ *Start[] และให้ Subscript ของอาเรย์เป็น 0..1..2..3..MaxNode เพื่อความสะดวกในการวนรอบ

- แปลงโครงสร้างกราฟในอาเรย์ไปเก็บในลิงค์ลิสต์ โดยการวนรอบอ่าน ข้อมูลในอาเรย์ หากมีเส้นทางก็จะสร้างโหนดใหม่ และเชื่อมโหนด ต่อท้ายไปเรื่อย ๆ โดยใช้ Pointer *Rear เป็นตัวชี้ตำแหน่งโหนด ท้ายสุดของการเพิ่มล่าสุด

3. การอ่านเช็ตของ Vertex และ Edge ก็ใช้กลไกของลิงค์ลิสต์ในการอ่านตาม โครงสร้างที่สร้างไว้

3.3.3.1 โปรแกรมการแทนกราฟโดยลิงค์ลิสต์

```
/* Program create Graph structure by "Adjacency list
representation method".
```

```
Can show the result of graph are ..
```

```
1. Set of vertex
```

```
2. Set of Edge
```

```
Note.- can use both Undirected graph and Directed graph
```

```
*/
```

```
#include <stdio.h> //use printf
```

```
#include <conio.h> //use getch
```

```
#include <stdlib.h> //use malloc
```

```
#define MaxNode 4 // Define Max Node of Graph
```

```

char NodeName[4] = {'A', 'B', 'C', 'D'};
int graph[MaxNode][MaxNode] = {
    {0,1,1,1},
    {1,0,1,1},
    {1,1,0,0},
    {1,1,0,0},
}; //Declare array and keep data of graph
struct Node //Declare structure of every node
{
    char info;
    struct Node *next;
};
struct Node *Start[MaxNode], *p; //Declare pointer uses
Node *Allocate() //Allocate 1 node from storage pool
{
    struct Node *temp;
    temp=(Node*)malloc(sizeof(Node)); //Allocate node by size declare
    return(temp);
}
void CreateHead() //Create Head Node
{
    int i;
    for (i=0;i<MaxNode;i++) //Count by Maximum of Node
    {
        p=Allocate();
        p->info=NodeName[i]; //Let INFO = Node Name
        p->next=NULL; //Let NEXT = NULL
        Start[i]=p; //Let Start of each node = Address of first Node
    }
}
void TransferToAdjacent() //Transfer array to Adjacency list of graph
{
    int i,j;
    struct Node *Rear; //Counter and point at last pointer finally
    for (i=0;i<MaxNode;i++) //row loop
    {
        Rear=Start[i]; //pointer Rear Start hear
        for(j=0;j<MaxNode;j++) //column loop
        {
            if (graph[i][j]==1) //if PATH?
            {
                p=Allocate(); //get new Node
                p->info=NodeName[j]; //Let info = NodeName[j]
                p->next=NULL; //Let NEST = NULL
                Rear->next=p; //Next of Rear point to New Node
                Rear=p; // Skip Rear pointer to Next Node
            }
        }
    }
}

```

```

    }
    }
}
}
void DispSetOfVertex() //Display set of Vertex
{
    int i;
    printf("\nSet of Vertex = {");
    for (i=0;i<MaxNode;i++) //Count only Start Node
    {
        printf("%c",Start[i]->info); //Display each node name
        if(i != MaxNode-1)
            printf(",");
    }
    printf("}\n");
}

void DispSetOfEdge() //Display set of Edge
{
    int i;
    struct Node *Temp;
    printf("\nSet of Edge = {");
    for (i=0;i<MaxNode;i++) //row loop
    {
        Temp=Start[i]; //Let Temp pointer Start hear
        Temp=Temp->next; //Skip Temp pointer to Next Node
        while (Temp != NULL) //Point at Node 2nd
        {
            printf("(%c,%c),",Start[i]->info,Temp->info); //Show each Edge
            Temp=Temp->next; //Skip Temp pointer to Next Node
        }
    }
    printf("}\n");
}

int main()
{
    printf("GRAPH (ADJACENCY LIST REPRESENTATION METHOD)\n");
    printf("=====\n");
    CreateHead();
    TransferToAdjacent();
    DispSetOfVertex();
    DispSetOfEdge();
    getch();
    return(0);
} //End Main

```



```

C:\Users\LENOVO\Downloads\GraphLinkedList (1).exe
GRAPH (ADJACENCY LIST REPRESENTATION METHOD)
=====
Set of Vertex = {A, B, C, D}
Set of Edge = {(A, B), (A, C), (A, D), (B, A), (B, C), (B, D), (C, A), (C, B), (D, A), (D, B), }

```

รูปที่ 3.56 แสดงผลการทำงานของโปรแกรมการแทนกราฟโดยลิงค์ลิสต์

3.3.4 การแทนกราฟแบบสเปิร์สเมทริก (Sparse matrix) หรือ Orthogonal

ในกรณีที่จำนวนเส้นโยงในกราฟมีน้อยมากเมื่อเทียบกับจำนวนจุด เมื่อแทนกราฟนั้นโดยใช้อาเรย์ 2 มิติจะมีช่องที่มีค่า 0 อยู่มาก ในกรณีเช่นนี้การแทนโดยใช้ Orthogonal list ซึ่งแต่ละจุดจะมีโครงสร้างดังรูป 3.57 อาจจะประหยัดพื้นที่ความจำและง่ายต่อการเขียนอัลกอริทึม

Tail พอยน์ เตอร์	Head พอยน์ เตอร์	Column พอยน์เตอร์สำหรับ Tail	Row พอยน์เตอร์สำหรับ Head
------------------------	------------------------	------------------------------------	---------------------------------

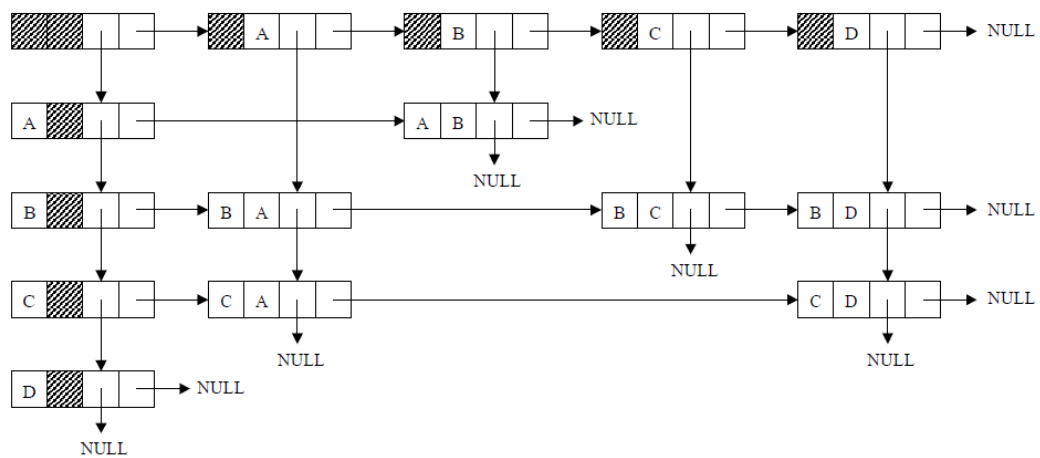
รูปที่ 3.57 โครงสร้างของ Orthogonal list

อนึ่ง Tail พอยน์เตอร์ หมายถึงส่วนหางลูกศร (กรณีเป็นไดกราฟหรือกราฟที่มี ทิศทาง) ที่แทนเส้นโยง ส่วนหัวลูกศรคือ Head พอยน์เตอร์ ซึ่งค่าทั้งสองนี้คือ ชื่อจุดที่อยู่ ปลายทั้งสองข้างของเส้นโยงหนึ่ง ๆ นั่นเอง

ขั้นตอนการแทนกราฟโดยใช้การแทนแบบ Orthogonal lists จะง่ายมากถ้าเขียน อาเรย์ 2 มิติแทนกราฟเสียก่อน จากนั้นเมตริกซ์ประชิดที่ได้จะมีลักษณะดังรูป 3.58

	A	B	C	D
A	0	1	0	0
B	1	0	1	1
C	1	0	0	1
D	0	0	0	0

จากนั้นก็สร้างจุด สำหรับแต่ละช่องที่เป็น 1 และสร้างหัวแถวตามแนวตั้งกับหัวแถว ตามแนวตั้งเพื่อแสดงถึงโครงสร้างความสัมพันธ์ของจุดตามกราฟนั้น ดังรูป 3.58



→ Row พอยน์เตอร์สำหรับหัวลูกศร (Head)

↓ Column พอยน์เตอร์สำหรับหางลูกศร (Tail)

รูปที่ 3.58 การแทนกราฟด้วย Orthogonal list

การวางโครงสร้างโปรแกรมการแทนกราฟแบบสปาร์สเมทริก

1. แทนโครงสร้างกราฟลงในอาร์เรย์ (การเริ่มต้นแทนแบบนี้จะสามารถเปลี่ยน โครงสร้างกราฟให้เป็นลักษณะอื่นได้ง่าย)

2. นำโครงสร้างกราฟใน (1) ไปเก็บแบบ Sparse Matrix โดยการเขียนโค้ดโดยมี ขั้นตอนดังนี้

- สร้าง Head Node ทั้ง Row และ Column ตามจำนวนของ Vertex โดยมีโหนดแรก(มุมบนซ้าย)เป็นตัวชี้ทั้งแนว Row และ Column

- แปลงโครงสร้างกราฟในอาร์เรย์ไปเก็บในลิงค์ลิสต์ โดยการวนรอบอ่าน ข้อมูลในอาร์เรย์ หากมีเส้นทางก็จะสร้างโหนดใหม่ และเชื่อมโหนด ต่อท้ายทางแนว Row ไปเรื่อย ๆ โดยใช้ Pointer *Row2 ชี้ตำแหน่ง โหนดท้ายสุดของการเพิ่มล่าสุด และ Pointer *Col2 ชี้คอลัมน์สุดท้าย เพื่อสะดวกต่อการระบุตำแหน่งโหนดทาง Row และ Column (ดู โมดูล TransferToSparseMT ประกอบ)

3. การอ่านเซตของ Vertex และ Edge ก็ใช้กลไกของลิงค์ลิสต์ในการอ่านตาม โครงสร้างที่สร้างไว้

3.3.4.1 โปรแกรมการแทนกราฟแบบสปาร์สเมทริก

/* Program create Graph structure by "Sparse Matrix representation method".

Can show the result of graph are ..

1. Set of vertex

2. Set of Edge

Note.- can use both Undirected graph and Directed graph

*/

```
#include <stdio.h> //use printf
```

```
#include <conio.h> //use getch
```

```
#include <stdlib.h> //use malloc
```

```
#define MaxNode 4 // Define Max Node of Graph
```

```
char NodeName[4] = {'A', 'B', 'C', 'D'};
```

```
int graph[MaxNode][MaxNode] = {
```

```
{0,1,0,0},
```

```
{1,0,1,1},
```

```
{1,0,0,1},
```

```
{0,0,0,0},
```

```
}; //Declare array and keep data of graph
```

```
struct Node //Declare structure of every node
```

```
{
```

```
char V1,V2;
```

```
struct Node *Row,*Column;
```

```
};
```

```
struct Node *Start[MaxNode], *p, *Head; //Declare pointer uses
```

```
Node *Allocate() //Allocate 1 node from storage pool
```

```
{
```

```
struct Node *temp;
```

```
temp=(Node*)malloc(sizeof(Node)); //Allocate node by size declare
```

```
return(temp);
```

```
}
```

```
void CreateHeadRC() //Create Head Node
```

```
{
```

```
int i,j;
```

```
struct Node *row, *col;
```

```
p=Allocate(); //Create HEAD Node
```

```
Head=p; //Let Head point at this Node and FREEZE it
```

```
row=Head;
```

```
for (i=0;i<MaxNode;i++) //Create HEAD ROW
```

```
{
```

```
p=Allocate();
```

```
row->Row=p; //Let ROW OF TEMP point to new node
```

```
row=p; //Change temp point to new node
```

```
row->V1=NodeName[i]; //Set Node Name
```

```
row->Row=NULL;
```

```

    row->Column=NULL;
}
col=Head;
for (j=0;j<MaxNode;j++) //Create HEAD COLUMN
{
    p=Allocate();
    col->Column=p; //Let COLUMN OF TEMP point to new node
    col=p; //Change temp point to new node
    col->V2=NodeName[j]; //Set Node Name
    col->Row=NULL;
    col->Column=NULL;
}
}
void TransferToSparseMT() //Transfer array to Sparse Matrix of graph
{
    int i,j;
    struct Node *row1, *row2, *col1, *col2;
    row1=Head->Row;
    i=0; //Start of Row in Array
    while (row1 != NULL) //row loop
    {
        col1=Head->Column;
        j=0; //Start of Column in Array
        col2=row1; //Let COL2 Count Column2 at ROW Pointed
        while (col1 != NULL) //column loop
        {
            if (graph[i][j]==1) //if PATH?
            {
                p=Allocate(); //get new Node
                p->V1=row1->V1; // Set arrow tail of Edge
                p->V2=col1->V2; //Set head tail of Edge
                p->Row=NULL; //Set pointer as NULL default
                p->Column=NULL; //Set pointer as NULL default
                col2->Column=p; //Let Column2 at ROW point to New Node
                col2=p; //Change COL2 position
                row2=col1; //Mark ROW2 point at first ROW of COLUMN
                while (row2->Row != NULL) //Find the last Node
                    row2=row2->Row; //Skip to next Node
                row2->Row=p; //Let Row of Column point to this node
            }
            col1=col1->Column;
            j++; // Skip to next Column Edge of Array
        }
        row1=row1->Row;
        i++; // Skip to next Row Edge of Array
    }
}

```

```

}
void DispSetOfVertex() //Display set of Vertex
{
    struct Node *row;
    printf("\nSet of Vertex = {");
    row=Head->Row; //Start Row at here
    while (row != NULL) //Loop at Row direction
    {
        printf("%c",row->V1); //Display each node name
        if(row->Row != NULL)
            printf(",");
        row=row->Row; //Skip to next Row
    }
    printf("}\n");
}
void DispSetOfEdge() //Display set of Edge
{
    struct Node *row, *col;
    printf("\nSet of Edge = {");
    row=Head->Row; //Start point of Row
    while (row != NULL) //row loop
    {
        col=row->Column; //Start point of Column
        while (col != NULL) //Column loop
        {
            printf("(%c,%c),",col->V1,col->V2); //Show each Edge
            col=col->Column; //Skip to next Edge/Column
        }
        row=row->Row; //Skip to next Row
    }
    printf("}\n");
}
int main()
{
    printf("GRAPH SPARSE MATRIX REPRESENTATION METHOD\n");
    printf("=====\n");
    CreateHeadRC();
    TransferToSparseMT();
    DispSetOfVertex();
    DispSetOfEdge();
    getch();
    return(0);
} //End Main

```

```

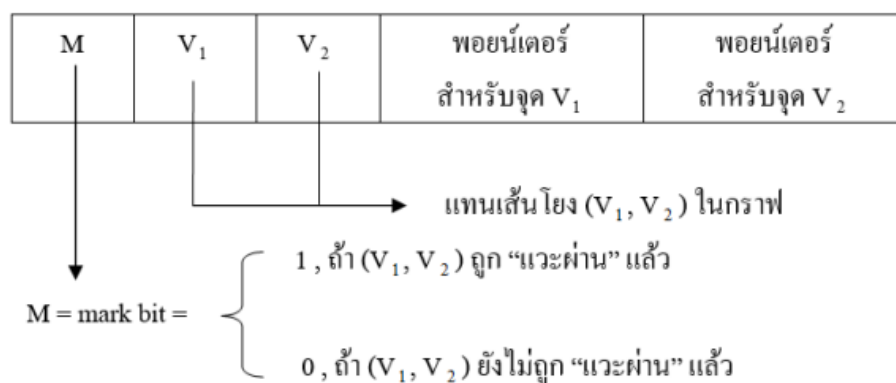
C:\Users\LENOVO\Downloads\GraphSparseMatrix (1).exe
GRAPH SPARSE MATRIX REPRESENTATION METHOD
=====
Set of Vertex = {A,B,C,D}
Set of Edge = {(A,B), (B,A), (B,C), (B,D), (C,A), (C,D), }

```

รูปที่ 3.59 แสดงผลการทำงานของโปรแกรมการแทนกราฟ Sparse Matrix

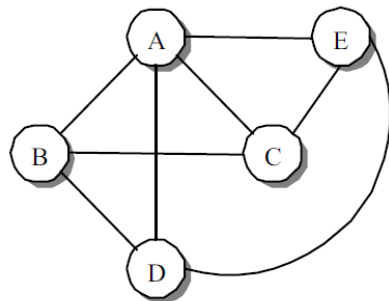
3.3.5 การแทนกราฟแบบติดกันหลายรายการ (Adjacency multi-lists)

นอกจากการแทนโดยเฟ่งถึงไปที่แต่ละจุดในกราฟ วิธีนี้จะแทนเส้นโยงได้โดยโครงสร้างที่กำหนดโครงสร้างแบบนี้จะทำให้แต่ละจุดสามารถถูกพิจารณาได้จาก สองทิศทาง เนื่องจากแต่ละเส้นโยงมีจุดอยู่ที่ปลายทั้งสองข้าง



รูปที่ 3.60 โครงสร้างของ Adjacency multi-lists

การเติมค่าลงในช่องพอยน์เตอร์สำหรับ V₁ และพอยน์เตอร์สำหรับ V₂ ทำได้โดยการหาว่าตำแหน่งที่อยู่ของ V₁ หรือ V₂ ที่อยู่ถัดไปอยู่ที่ใด ค่าแอดเดรสนั้นคือพอยน์เตอร์ที่ต้องการ เช่นในแถวที่ 2 พอยน์เตอร์สำหรับ V₁ (V₁ คือจุด A) คือ 3 เนื่องจากจุดที่ถัดไป ปรากฏในแถวที่ 3 ส่วนพอยน์เตอร์สำหรับ V₂ (V₂ คือจุด C แถวนั้น) มีค่า 5 เนื่องจากจุด C ถัดไป ปรากฏในแถวที่ 5



	M	V ₁	V ₂			
1		A	B	2	5	เอดจ (A,B)
2		A	C	3	5	เอดจ (A,C)
3		A	D	4	6	เอดจ (A,D)
4		A	E	-	7	เอดจ (A,E)
5		B	C	6	7	เอดจ (B,C)
6		B	D	-	8	เอดจ (B,D)
7		C	E	-	8	เอดจ (C,E)
8		D	E	-	-	เอดจ (D,E)

รูปที่ 3.61 การแทนกราฟโดยใช้ Adjacency multi lists

การวางโครงสร้างโปรแกรมการแทนกราฟแบบ Adjacency Multi list

การแทนกราฟโดยอาร์เรย์ สามารถระบุค่าต่าง ๆ ลงในโครงสร้างอาร์เรย์ตอนประกาศ ตัวแปรได้เลย การแสดงสมาชิกของเวอร์เท็กซ์ (Vertex) จะอ่านข้อมูลในอาร์เรย์ 1 มิติที่เก็บชื่อโหนด ส่วนการแสดงสมาชิกของเอดจ์ (Edge) จะอ่านจากโครงสร้างกราฟที่เก็บไว้ในอาร์เรย์ 2 มิติ ชื่อ graph ในตำแหน่งคอลัมน์ 1,2 และ 2,1

3.3.5.1 โปรแกรมการแทนกราฟแบบ Adjacency Multi list

/ Program create Graph structure by "Adjacency Multi list method".*

Can show the result of graph are ..

1. Set of vertex

2. Set of Edge

**/*

`#include <stdio.h> //use printf`

`#include <conio.h> //use getch`

`#define MaxEdge 8 // Define Max Edge of Graph`

`#define Block 5 // Define Block of each node`

`#define MaxNode 5 // Define Max Node`

`char NodeName[5] = {'A','B','C','D','E'};`

`char graph[MaxEdge][Block] = {`

`{'0','A','B','2','5'},`

`{'0','A','C','3','5'},`

`{'0','A','D','4','6'},`

`{'0','A','E','-','7'},`

`{'0','B','C','6','7'},`

`{'0','B','D','-','8'},`

```

    {'0','C','E','-','8'},
    {'0','D','E','-','-'},
}; //Declare array and keep data of graph
void DispArray2D(){ //Display value in 2D Array
    int i,j; //i=Row, j=Column
    printf("  M V1 V2 E1 E2\n");
    for (i=0;i<MaxEdge;i++){ //row loop
        printf("%d ",i+1); //Display number of Row
        for (j=0;j<Block;j++) //column loop
            printf("%c ",graph[i][j]); //Display value path
        printf("\n");
    }
}
void DispSetOfVertex(){ //Display set of Vertex
    int i;
    printf("\nSet of Vertex = {");
    for (i=0;i<MaxNode;i++){
        printf("%c",NodeName[i]); //Display each node name
        if(i != MaxNode-1)
            printf(",");
    }
    printf("}\n");
}
void DispSetOfEdge(){ //Display set of Edge
    int i,j;
    printf("\nSet of Edge = {");
    for (i=0;i<MaxEdge;i++) { //row loop
        printf("(%c,%c)",graph[i][1],graph[i][2]); //Show each Edge1
        printf("(%c,%c)",graph[i][2],graph[i][1]); //Show each Edge2
    }
    printf("}\n");
}

int main(){
    printf("GRAPH ADJACENCY MULTI-LIST REPRESENTATION METHOD\n");
    printf("=====\n");
    DispArray2D();
    DispSetOfVertex();
    DispSetOfEdge();
    getch();
    return(0);
} //End Main

```


GRAPH ADJACENCY MULTI-LIST REPRESENTATION METHOD

```

=====
M V1 V2 E1 E2
1 0 A B 2 5
2 0 A C 3 5
3 0 A D 4 6
4 0 A E - 7
5 0 B C 6 7
6 0 B D - 8
7 0 C E - 8
8 0 D E - -

```

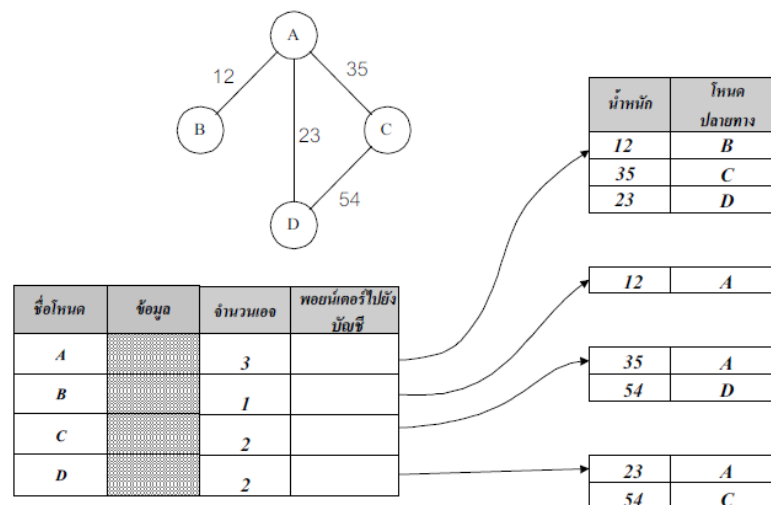
Set of Vertex = {A,B,C,D,E}

Set of Edge = {(A,B), (B,A), (A,C), (C,A), (A,D), (D,A), (A,E), (E,A), (B,C), (C,B), (B,D), (D,B), (C,E), (E,C), (D,E), (E,D), }

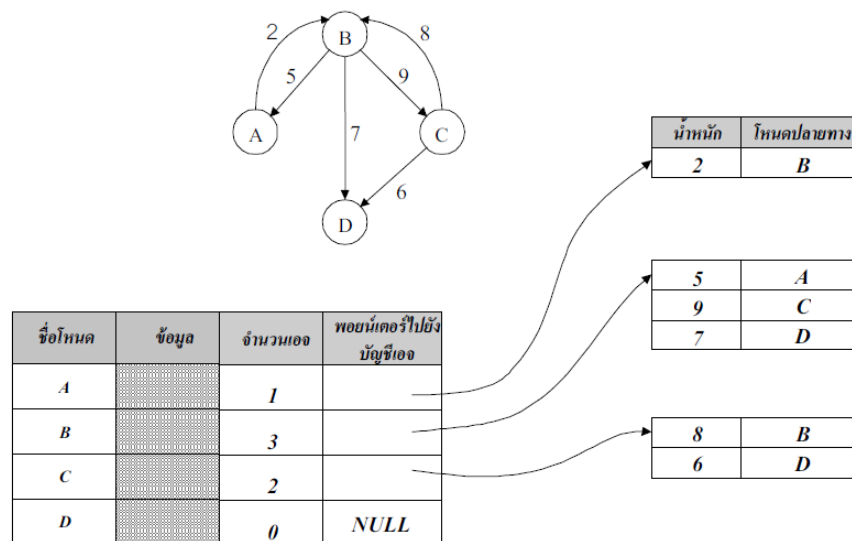
รูปที่ 3.62 แสดงผลการทำงานของโปรแกรมการแทนกราฟ Adjacency Multi list

3.3.6 การแทนกราฟแบบโหนดไดเรกทอรี (Node directory)

การแทนแบบนี้เหมาะกับกรณีที่มีจุดจำนวนมาก แต่มีเส้นโยงจำนวนน้อย โดยอาศัย Directory แสดงชื่อจุดต่าง ๆ จากนั้นจะมีบัญชีย่อยสำหรับเส้นโยงทั้งหลายที่ต่อกับ จุดนั้น สำหรับตัวเลขกำกับที่เอดจ์แสดงจะแสดงน้ำหนักของกราฟ (Weight) ซึ่งอาจหมายถึง ระยะทาง เวลา ค่าใช้จ่าย เป็นต้น



รูปที่ 3.63 การแทนกราฟโดยวิธี Node directory (1)



รูปที่ 3.64 การแทนไดกราฟโดยวิธี Node directory (2)

การวางโครงสร้างโปรแกรมการแทนกราฟแบบ Node Directory

การแทนกราฟจะใช้อาร์เรย์ ซึ่งสามารถระบุค่าต่าง ๆ ลงในโครงสร้างอาร์เรย์ตอน ประกาศตัวแปรได้เลย โดยจะเก็บไว้ 2 ตารางคือ

1. ตารางโหนด
2. ตารางน้ำหนักและโหนดปลายทาง แล้วนำตารางทั้งสองมาเชื่อมโยงกันเป็นกราฟ

3.3.6.1 โปรแกรมการแทนกราฟแบบ Node Directory

/* Program create Graph structure by "Node directory method".
Can show the result of graph are ..

1. Set of vertex
2. Set of Edge and Weight

Note.- Can use for Weighting Graph in both Undirected graph and Directed graph

```
*/
#include <stdio.h> //use printf
#include <conio.h> //use getch
#define MaxNode 4 // Define Max Node
#define Block 4 // Define Block of each node
#define MaxEdge 6 // Define Max Extra of Graph
char Head[MaxNode][Block] = {
    {'A', '-', '1', '1'},
    {'B', '-', '3', '2'},
    {'C', '-', '2', '5'},
    {'D', '-', '0', NULL},
```

```

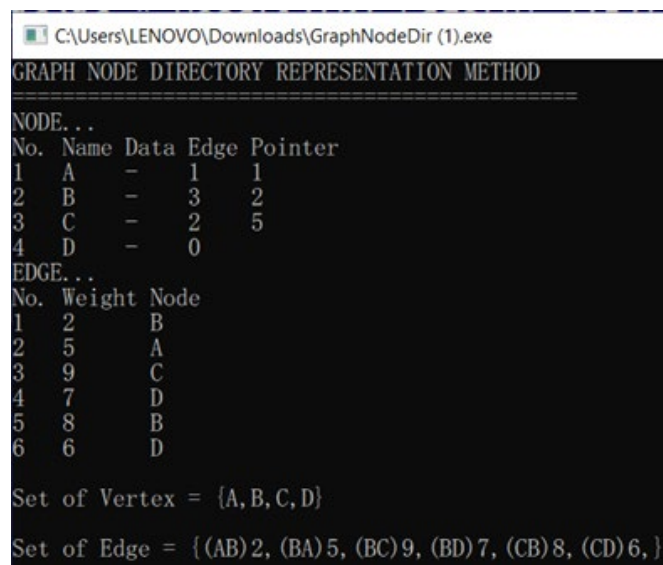
}; //Declare array and keep Head Node of graph
char Edge[MaxEdge][2] = {
    {'2','B'},
    {'5','A'},
    {'9','C'},
    {'7','D'},
    {'8','B'},
    {'6','D'},
}; //Declare array and keep Edge of graph
void DispHead() //Display Head in 2D Array
{
    int i,j; //i=Row, j=Column
    printf("NODE...\n");
    printf("No. Name Data Edge Pointer\n");
    for (i=0;i<MaxNode;i++) //row loop
    {
        printf("%d    ",i+1); //Display number of Row
        for (j=0;j<Block;j++) //column loop
            printf("%c    ",Head[i][j]); //Display Node
        printf("\n");
    }
}
void DispEdge() //Display Edge in 2D Array
{
    int i,j; //i=Row, j=Column
    printf("EDGE...\n");
    printf("No. Weight Node\n");
    for (i=0;i<MaxEdge;i++) //row loop
    {
        printf("%d    ",i+1); //Display number of Row
        for (j=0;j<2;j++) //column loop
            printf("%c    ",Edge[i][j]); //Display Node
        printf("\n");
    }
}
void DispSetOfVertex() //Display set of Vertex
{
    int i;
    printf("\nSet of Vertex = {");
    for (i=0;i<MaxNode;i++)
    {
        printf("%c",Head[i][0]); //Display each node name
        if(i != MaxNode-1)
            printf(",");
    }
    printf("}\n");
}

```

```

}
void DispSetOfEdge() //Display set of Edge
{
    int i,j,AmtEdge,PT;
    printf("\nSet of Edge = {");
    for (i=0;i<MaxNode;i++) //row loop
    {
        AmtEdge=Head[i][2]-48; //Convert Character to Integer for Amount of
        Edge
        PT=Head[i][3]-48; //Convert Character to Integer for Start point of
        Array
        for (j=0;j<AmtEdge;j++) //Loop follow by Amount of Edge
        {
            printf("(%c%c)%c,",Head[i][0],Edge[PT-1+j][1],Edge[PT-1+j][0]);
            //Show Edge and Weight
        }
    }
    printf("}\n");
}
int main()
{
    printf("GRAPH NODE DIRECTORY REPRESENTATION METHOD\n");
    printf("=====\n");
    DispHead();
    DispEdge();
    DispSetOfVertex();
    DispSetOfEdge();
    getch();
    return(0);
} //End Main

```



```

C:\Users\LENOVO\Downloads\GraphNodeDir (1).exe
GRAPH NODE DIRECTORY REPRESENTATION METHOD
=====
NODE...
No. Name Data Edge Pointer
1 A - 1 1
2 B - 3 2
3 C - 2 5
4 D - 0
EDGE...
No. Weight Node
1 2 B
2 5 A
3 9 C
4 7 D
5 8 B
6 6 D

Set of Vertex = {A,B,C,D}
Set of Edge = {(AB)2, (BA)5, (BC)9, (BD)7, (CB)8, (CD)6,}

```

รูปที่ 3.65 แสดงผลการทำงานของโปรแกรมการแทนกราฟ Node Directory

3.3.7 การแหวผ่านกราฟ

ในบรรดาปัญหาต่าง ๆ มีกระบวนการในการแก้ไขปัญหที่ต่างกันไป ยกตัวอย่าง เช่นการใช้เลขลำดับอนุกรม การใช้คุณสมบัติของเมตริกซ์ ซึ่งสามารถหาผลเฉลยด้วย อัลกอริทึมที่มีประสิทธิภาพการทำงานที่ยอมรับได้ แต่สำหรับบางปัญหาแล้ว จำเป็นต้องใช้ วิธีแจกแจงและตรวจสอบผลเฉลย ซึ่งผลเฉลยที่ต้องแจกแจงและตรวจสอบนั้นมีมากมาย การแจกแจงและตรวจสอบในทุก ๆ กรณีคงกระทำไต่ยาก ดังนั้นแล้วจึงจำเป็นต้องเพิ่มกลวิธี ในการแจกแจงเฉพาะกรณีที่ น่าสนใจ ซึ่งสามารถแจกแจงแล้วพบคำตอบได้เร็วขึ้นจนเป็นที่ ยอมรับได้ในทางปฏิบัติ รูปแบบผลเฉลยมีหลากหลายมากมายแต่ที่พบเห็นบ่อยได้แก่ วิธี เรียงสับเปลี่ยน เซตย่อย และการแบ่งเซต จากนั้นก็นำเสนอการแจกแจงกรณีดังกล่าวด้วยการ จำลองกระบวนการแจกแจงด้วยต้นไม้(หรือกราฟ) ซึ่งมีจุดแทนสถานะของผลเฉลย และ การแตกกิ่งแทนการแจกแจงกรณี ทำให้สามารถเทียบเคียงการแจกแจงผลเฉลยได้โดยการแหว ผ่านจุดในต้นไม้ ดังนั้นการแหวผ่านจุดจึงเสมือนเป็นกระบวนการค้นคำตอบในต้นไม้ วิธีแหว ผ่านที่มีระบบระเบียบที่ใช้กันมากได้แก่การแหวผ่านตามแนวลึก และตามแนวกว้าง ซึ่ง ประกันได้ว่าต้องพบคำตอบได้แน่นอน แต่อาจใช้เวลาต่างกัน จึงจำเป็นต้องศึกษากลวิธี เพื่อนำไปปรับใช้ในการแก้ปัญหต่าง ๆ

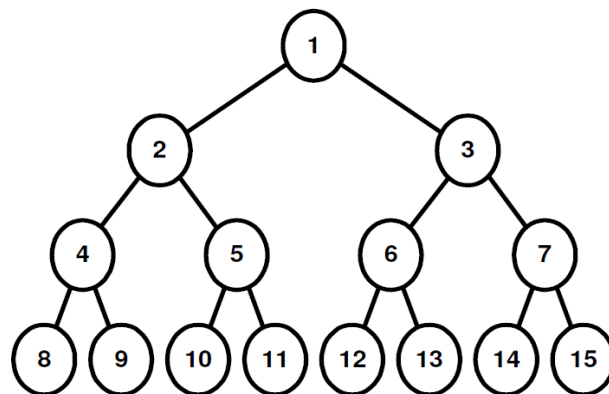
Live-node และ E-node

ในการแจกแจงนั้นจะอาศัยการสร้างจุด เพื่อเชื่อมต่อ) มีการแตกกิ่งของจุด และมี การทำลายจุดทิ้ง ในขณะที่กำลังแหวผ่านโดยอาศัยการแตกกิ่งจะมีเพียงจุดเดียวเท่านั้นที่ กำลังแตกกิ่งอยู่ แต่ว่า ณ ขณะใดขณะหนึ่งอาจมีหลาย ๆ การเรียกแบบเวียนเกิดที่ยังทำงาน ค้าง ๆ อยู่ได้ จึงเปรียบเสมือนว่ามีจุดที่มีชีวิตอยู่หลายจุดได้ ณ ขณะใดขณะหนึ่ง เรียกจุดที่มีชีวิตนี้ว่า live node และเรียกจุดที่กำลังแตกกิ่งอยู่ว่า E-node

3.3.7.1 การแหวผ่านตามแนวลึก (Depth first traversal)

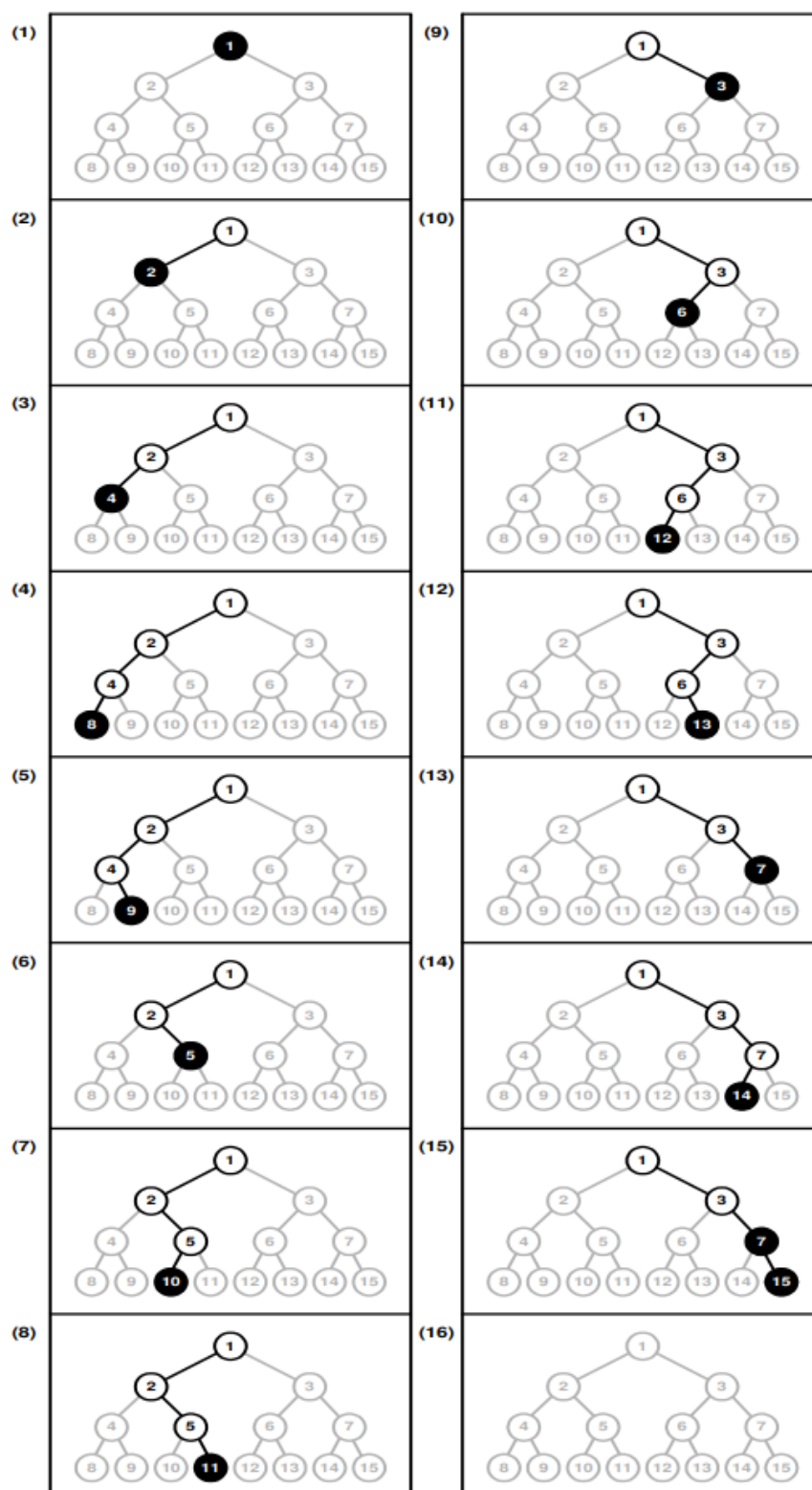
การแหวผ่านหาตามแนวลึกเหมือนกับการแหวผ่านต้นไม้แบบก่อนลำดับ คือจะลึกลงไปเรื่อย ๆ นั้นแสดงว่าเริ่มด้วยการสร้างรากให้เป็น live node รากถูกเลือกให้เป็น E- node แตกกิ่งได้ live node ใหม่ จุดใหม่นี้ก็กลายเป็น E-node ทันทีเพื่อแตกกิ่งต่อได้ live- node ใหม่ จุดใหม่นี้ก็กลายเป็น E-node ทันทีเพื่อแตกกิ่งต่อไปอีก เมื่อใดที่จุด E-node ปัจจุบันแตกกิ่งหมดแล้ว (หรือแตกไม่ได้กรณีเป็นใบ) ก็จะถูกทำลาย และเลือกจุด live node ก่อนหน้านี้นี้เป็นจุด E-node เพื่อแตกกิ่งต่อสรุปได้ว่าจุดบรรพบุรุษจะแตกกิ่งหมดหลังจุดลูกหลาน

สมมติว่าต้องการแหวะผ่านตามแนวลึกในต้นไม้ดังรูปที่ 3.66



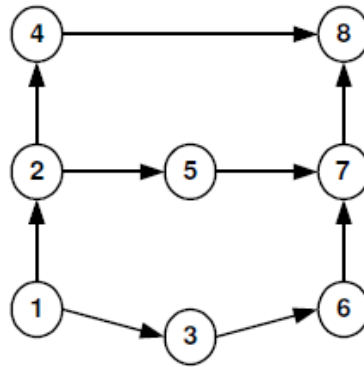
รูปที่ 3.66 โครงสร้างกราฟเพื่อใช้อธิบายการแหวะผ่าน

รูปที่ 3.67 แสดงจุดต่าง ๆ ที่ถูกสร้างขึ้นระหว่างการแหวะผ่านตามแนวลึกในต้นไม้ วงกลมขาวคือ live node วงกลมดำคือ E-node จะตรวจสอบสถานะของจุด V ตอนที่ v ถูก สร้างเป็น live node จะแตกกิ่งจุด v เมื่อ v ถูกเลือกเป็น E-node เริ่มด้วยการสร้างรากจุด ที่ 1 แตกกิ่งซ้ายได้จุด 2 ซึ่งก็แตกกิ่งซ้ายได้จุด 4 และ 8 ตามลำดับ ถึงตรงนี้จุด 8 เป็นใบไม้ แตกกิ่ง ก็ถูกทำลายไป ย้อนกลับมาจุดที่ 4 แตกกิ่งขวาได้จุด 9 ซึ่งก็ถูกทำลายอีกเช่นกัน เพราะเป็นใบ ย้อนกลับมาจุด 4 ซึ่งแตกกิ่งหมดแล้วก็ถูกทำลาย ย้อนกลับมาที่จุด 2 แตกกิ่ง ขวาได้จุด 5 กระทำสร้าง แตกกิ่ง และทำลายจุด ในลักษณะนี้จนครบทุกจุด จะเห็นว่าจุด ต่าง ๆ ในต้นไม้ถูกสร้างขึ้นเป็น live-node ตามลำดับการแหวะตามแนวลึก



รูปที่ 3.67 การแหวะผ่านตามแนวลึก

และจากวิธีการแหว่ผ่านดังที่กล่าวมาลำดับในการแหว่ผ่านจุดต่าง ๆ ในกราฟดังรูปที่ 3.66 ก็คือ 1,2,4,8,5,7,3,6 หรือทำการแหว่ผ่านได้อีกแบบโดยการแตกกิ่งทางด้านขวาก่อน ทำให้ได้มาลำดับในการแหว่ผ่านคือ 1,3,6,7,8,2,5,4



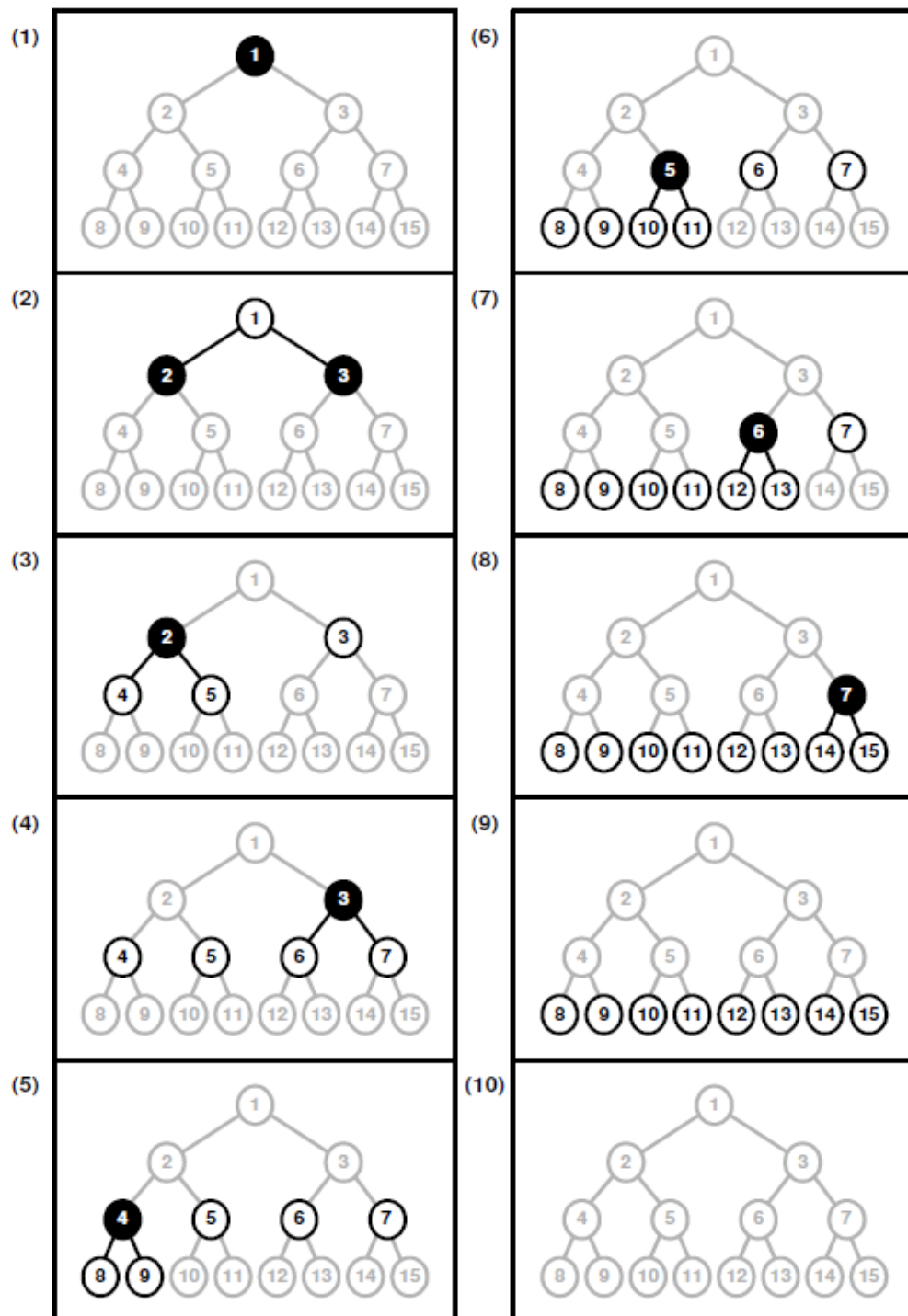
รูปที่ 3.68 โครงสร้างกราฟแสดงการแหว่ผ่านทางลึก

3.3.7.2 การแหว่ผ่านหาตามแนวกว้าง (Breadth first Traversal)

กระบวนการแหว่ผ่านในต้นไม้หรือกราฟอีกแบบหนึ่งที่มีระบบระเบียบก็คือ การ แหว่ผ่านตามแนวกว้าง ลักษณะการแหว่ผ่านเช่นนี้คล้ายการแหว่ผ่านจุดต่าง ๆ ในต้นไม้ทีละ ระดับ คือเริ่มตรวจสอบสถานะของจุดในระดับ 0 (ราก) ตามด้วยทุก ๆ จุดในระดับที่ 1 ตาม ด้วยทุก ๆ จุดในระดับที่ 2 ไปทีละระดับเช่นนี้จนถึงระดับล่างสุด ดังนั้นลำดับการแหว่ผ่าน จุดของต้นไม้ในรูปที่ 3.64 ก็คือ 1,2,3,4,5,6,7,..., 15

การแหว่ผ่านตามแนวกว้างจะเป็นวงวนของการเลือก E-node มาหนึ่งจุดจาก live node ต่าง ๆ จากนั้นจะแตกกิ่ง E-node ที่เลือกนี้เพื่อผลิตจุดลูกออกมาเป็น live node ให้ หมดทุกลูก (แล้วตัวเองก็ถูกทำลายไป) จะเห็นว่าการแหว่ผ่านแบบนี้แตกต่างจากการแหว่ ผ่านตามแนวลึกที่แตกกิ่งทีละกิ่งแล้วเปลี่ยนจุดที่ถูกสร้างเป็น E-node ทันที นั่นคือการแหว่ ผ่านตามแนวลึก จุดใดเกิดก่อนตายทีหลัง แต่สำหรับการแหว่ผ่านตามแนวกว้างนั้นเกิดก่อน ตายก่อน รูปที่ 3.67 แสดงตัวอย่างการแหว่ผ่านตามแนวกว้างในต้นไม้ จะทำการตรวจสอบ สถานะของจุดเมื่อจุดนั้นถูกสร้างเป็น live-node และแตกกิ่งจุดเมื่อจุดนั้นถูกเลือกเป็น E- node การแหว่ผ่านเริ่มต้นโดยการให้รากของต้นไม้เป็น live node (แสดงด้วยวงกลมขาว) จากนั้นเปลี่ยนเป็น E-node (วงกลมดำ) แตกกิ่งสร้างจุดเรียงลำดับจากซ้ายไปขวาได้ live node ใหม่ ๆ แล้วตัวเองก็ถูกทำลายไป จากนั้นลงมาอีกหนึ่งระดับแหว่ผ่าน live node จาก ซ้ายไปขวามาเปลี่ยนเป็น E-node แล้วแตกกิ่งได้ live node เพิ่มขึ้นอีกในระดับถัดไป

กระทำเช่นนี้ไปเรื่อย ๆ จะเห็นว่าจุดต่าง ๆ ในต้นไม้ถูกสร้างขึ้นเป็น live node ตามลำดับ การแวะผ่านตามแนวกว้าง การแวะผ่านตามแนวกว้างแสดงดังรูปที่ 3.69



รูปที่ 3.69 การแวะผ่านตามแนวกว้าง

ถ้าสังเกตลำดับของจุดในต้นไม้ที่ถูกสร้างขึ้น และลำดับของ live node ต่าง ๆ ที่ ถูกเลือกมาเป็น E-node เพื่อแตกกิ่งต่อนั้น จุดที่ถูกสร้างเป็น live node ก่อนจะถูกเลือก ออกมาแตกกิ่งก่อน

ดังนั้นจึงเป็นธรรมชาติอย่างยิ่งที่จะจัดเก็บ live node ต่าง ๆ ไว้ใน queue ดังนั้นวงวนการทำงานระหว่างการแหว่ผ่านตามแนวกว้างก็คือการดึงจุดออกจาก queue เพื่อแตกกิ่งสร้างจุดลูกทั้งหมดเป็น live node เพิ่มใส่เข้าไปใน queue หมุนวน ทำงานลักษณะนี้จน queue ว่างนั่นเอง

และจากวิธีการแหว่ผ่านดังที่กล่าวมาลำดับในการแหว่ผ่านจุดต่าง ๆ ในกราฟดังรูปที่ 3.68 ก็คือ 1,2,3,4,5,6,8,7 หรือทำการแหว่ผ่านได้อีกแบบโดยการแตกกิ่งทางด้านขวาก่อน ทำให้ได้ลำดับในการแหว่ผ่านคือ 1,3,2,6,5,4, 7, 8

สรุป

กราฟเป็นโครงสร้างข้อมูลที่ประกอบด้วยเซตของจุด (หรือเวอร์เทกซ์) และเซตของ เส้นโยง โดยต้องมียังน้อยหนึ่งจุด แต่อาจจะไม่มีเส้นโยงที่เชื่อมต่อระหว่างจุดเลย การ แทนที่กราฟในหน่วยความจำแบบสแตติกทำได้โดยใช้เมตริกซ์ของตัวเลข 0 และ 1 แทนการ มีและไม่มีเส้นทางจากจุดในแถวไปยังจุดในคอลัมน์ตามลำดับ เป็นเมตริกซ์ที่แสดงความยาว หนึ่ง อาศัยการคูณเมตริกซ์ทางความยาวหนึ่งนี้ ให้เป็นเมตริกซ์ที่แสดงจำนวนเส้นทางความ ยาว 2,3 หรือ 4 (ขึ้นกับว่าทำการคูณเมตริกซ์กี่ครั้ง) จากจุดในแถวไปยังจุดในคอลัมน์ จำนวนเส้นทางไซเคิลได้จากค่าในเส้นทะแยงของเมตริกซ์

ส่วนการแทนที่กราฟด้วยพอยเตอร์ให้ความสะดวกในการหาเส้นทางที่ทำให้รู้ว่าใน แต่ละเส้นทางผ่านจุดใดบ้างโดยเฉพาะเส้นทางความยาวหนึ่งได้สะดวกกว่าการแทนที่ด้วย เมตริกซ์ แต่จะตอบคำถามเหล่านี้ค่อนข้างยากเช่นที่มีเส้นทางความยาว 2 จากจุด A ไปยัง B หรือไม่? มีเส้นทางความยาว 3 จาก A ไปยัง B ก็เส้นทาง เส้นทางใดบ้าง? อินดีกรี(In- degree)และเอ้าต์ดีกรี(Out-degree)ของแต่ละจุดเป็นเท่าไร เป็นต้น

การแหว่ผ่านกราฟ คือขบวนการที่แหว่ไปที่ทุกๆ จุดของกราฟ สำหรับเทคนิคในการแหว่ผ่าน มี 2 แบบ คือ

1. การแหว่ผ่านแนวกว้าง (Breadth first traversal)
2. การแหว่ผ่านแนวลึก (Depth first traversal)