8.4 ต้นไม้ที่มีผลรวมของกิ่งสั้นที่สุด (Minimal Spanning Tree)

ต้นไม้ที่มีผลรวมของกิ่งสั้นที่สุด คือ กราฟที่เชื่อมถึงกันทุกโหนดและไม่มีวงด้วยผลรวมของกิ่งน้อยที่สุด หากกราฟประกอบด้วย N โหนด จะได้ต้นไม้ประกอบด้วย N-1 กิ่ง

การหาต้นไม้ที่มีผลรวมของกิ่งสั้นที่สุด มักนำไปประยุกต์ใช้ในด้านการขนส่ง การสื่อสารต่าง ๆ เช่น บริษัทขนส่งสินค้าต้องแวะส่งสินค้าไปยังทุกสถานีย่อย และต้องการให้ระยะทางรวมน้อยที่สุด การวางสายโทรศัพท์หรือสายเคเบิลเชื่อมโยงทุก ๆ จุดที่ต้องการใช้บริการและให้มีความยาวของสายสั้นที่สุด เป็นต้น

วิธีการหาต้นไม้ที่มีผลรวมของกิ่งสั้นที่สุด ที่นิยมเป็นของ Kruskal กับ Prim

8.4.1 วิธีการของ Kruskal

วิธีการของ Kruskal ใช้ขั้นตอนวิธีละโมบ (greedy algorithms) เริ่มต้นจากการเรียง ลำดับกิ่งที่มีค่าน้อยไปมาก โดยจะนำกิ่งที่มีค่าน้อยที่สุดมาพิจารณาว่าถ้าเพิ่มกิ่งดังกล่าวลงไปในต้นไม้แล้วจะทำให้เกิดวงหรือไม่ ถ้าไม่เกิดจะทำการเพิ่มกิ่งนั้นลงไปในต้นไม้ แต่ถ้าก่อให้เกิดวงก็ไม่ทำการเพิ่มกิ่งดังกล่าว และจะทำการพิจารณาเช่นนี้ไปเรื่อย ๆ จนครบทุกกิ่ง ก็จะได้กราฟต้นไม้ที่มีผลรวมของกิ่งสั้นที่สุด

วิธีของ Kruskal จะเกี่ยวข้องกับการตรวจสอบว่ากิ่งที่เพิ่มเข้าไปจะทำให้เกิดวงหรือไม่ ซึ่งทำได้โดยการตรวจสอบว่าจุดปลายทั้ง 2 ของกิ่งว่าอยู่ในต้นไม้ต้นเดียวกันหรือไม่ ซึ่งการตรวจสอบดังกล่าวอาศัยเมธตอด MAKE-SET (x), FIND-SET(x) และ UNION(x, y)

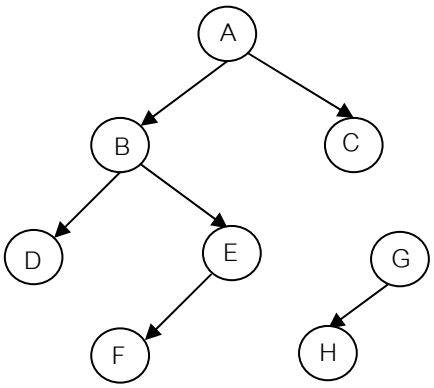
```
MakeSet(Graph x)  // กำหนดค่าเริ่มต้น สำหรับต้นไม้
{  p(x) = x;        //กำหนดรากของต้นไม้
   rank(x) = 0;  // rank คือ ความสูงของต้นไม้
}
```

```
FindSet(Node x)  //หารากของต้นไม้ ถ้ายังไม่ใช่ราก จะหาพ่อไปเรื่อยๆ จนถึงราก
{   if (x != p(x) )
       {p(x) = FindSet(p(x)); }
     return p(x);
}
```

```
Link(Graph x, Graph y)   // นำต้นไม้ที่เตี้ยกว่าไปเป็นต้นไม้ย่อยต่อกับต้นที่สูงกว่า
 { if (rank(x) > rank(y))
        { p(y) = x;}
        else
        {p(x)= y; }
      if ( rank(x) = rank(y)
        (rank(y) = rank(y)+1;
      }
```

```
Union (Graph  x, Graph y) // การรวมต้นไม้ 2 ต้น เป็นต้นเดียว
        { Link(FindSet(x), FindSet(y));}
```

ตัวอย่าง 8.4 จากต้นไม้ที่กำหนดให้ในรูป 8.8 จะแสดงการเรียกใช้เมธอด FindSet(x);



รูป 8.8 หา FindSet จากกราฟ

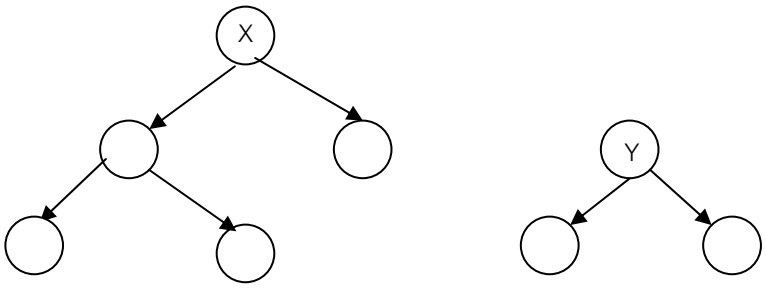
FindSet(D) เรียกพ่อของ D ➔ p(D) = B
FindSet(B) เรียกพ่อของ B ➔ p(B) = A
คืนค่า FindSet(D) จะได้ A

FindSet(F) เรียกพ่อของ F ➔ p(F) = E
FindSet(E) เรียกพ่อของ E ➔ p(E) = B
FindSet(B) เรียกพ่อของ B ➔ p(B) = A
คืนค่า FindSet(F) จะได้ A

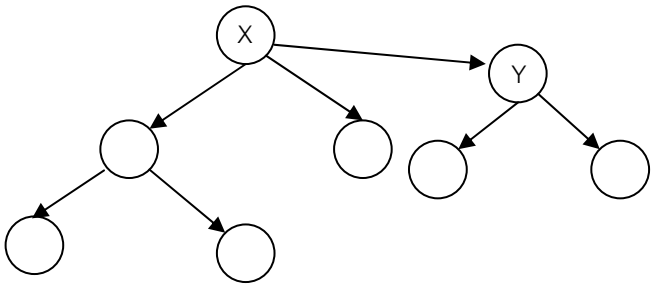
FindSet(H) เรียกพ่อของ H ➔ p(H) = G
p(G) = G

เพราะโหนด D, F อยู่บนต้นไม้ต้นเดียวกัน จะคืนค่ารากเท่ากัน ถ้าเชื่อมกิ่งระหว่างโหนด D, F จะทำให้เกิดเป็นวงขึ้น สำหรับโหนด G, H ไม่อยู่บนต้นเดียวกับ F จึงสามารถเพิ่มกิ่งที่เชื่อม โหนด F, G โดยไม่ทำให้เกิดวงขึ้น และรวมเป็นต้นไม้ต้นเดียวกัน

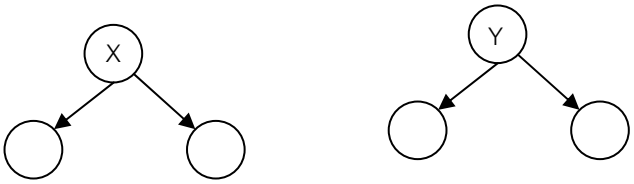
<u>ตัวอย่าง 8.5</u> แสดงการรวมต้นไม้โดยฟังก์ชัน UNION



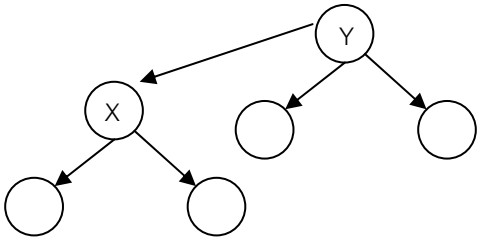
เนื่องจากต้นไม้ X มีความสูงเป็น 2 ซึ่งมากกว่าต้นไม้ Y ที่มีความสูงเป็น 1 ดังนั้นจะนำ Y ไปเป็นต้นไม้ย่อยของ X ดังรูป และความสูงของต้นไม้ X จะคงเดิม



กรณีที่ต้นไม้ X และต้นไม้ Y มีความสูงเท่ากันดังรูป



จะนำต้นไม้ X ไปเป็นต้นไม้ย่อยของต้นไม้ Y และเพิ่มความสูงให้กับต้นไม้ Y อีก 1



ขั้นตอนวิธีของ Kruskal

```
MST_Kruskal(Graph G, EdgeE)
{  A= { Φ } // กำหนดให้เป็นเซ็ตว่าง เพื่อบรรจุโหนด – กิ่งที่ประกอบเป็นต้นไม้
  for (each v ∈ G)
      {   MakeSet(v);   }   // สร้างต้นไม้ย่อยๆ แต่ละต้นมีโหนดเดียว
  / /ทำการเรียงลำดับน้ำหนักของกิ่งในกราฟ จากน้อยไปมาก
  For (each edge (u, v) ∈ E)    //ที่เรียงลำดับน้ำหนักจากน้อยไปมาก
      { if (FindSet(u)!= Findset(v))
              { A = A ∪ E(u,v)
                union (u,v); }
       }
   return A;
```

<u>ตัวอย่าง 8.6</u> แสดงการหาต้นไม้ที่มีผลรวมของกิ่งสั้นที่สุดโดยวิธีของ Kruskal

<u>วิธีทำ</u>

(ก) เริ่มจากกิ่งที่มีค่าน้อยที่สุด hg = 2 โหนด h, g ไม่อยู่บนต้นไม้เดียวกัน เพิ่ม hg ในเซ็ต A

(ข) ดูกิ่ง ci = 3 โหนด c, i ไม่อยู่บนต้นไม้เดียวกัน เพิ่ม ci ในเซ็ต A

(ค) กิ่ง gf = 3 โหนด g, f ไม่อยู่บนต้นไม้เดียวกัน เพิ่ม gf ในเซ็ต A

(ง) กิ่ง ab = 5 โหนด a, b ไม่อยู่บนต้นไม้เดียวกัน เพิ่ม ab ในเซ็ต A

(จ) กิ่ง cf = 5 โหนด c, f ไม่อยู่บนต้นไม้เดียวกัน เพิ่ม cf ในเซ็ต A

(ฉ) กิ่ง ig = 7 โหนด i, g อยู่บนต้นไม้เดียวกัน ไม่สามารถเพิ่ม ig ในเซ็ต A

(ช) กิ่ง cd = 8 โหนด c, d ไม่อยู่บนต้นไม้เดียวกัน เพิ่ม cd ในเซ็ต A

(ซ) กิ่ง ih = 8 โหนด i, h อยู่บนต้นไม้เดียวกัน ไม่สามารถเพิ่ม ih ในเซ็ต A

(ฌ) กิ่ง ah = 9 โหนด a,h ไม่อยู่บนต้นไม้เดียวกัน เพิ่ม ah ในเซ็ต A

(ญ) กิ่ง bc = 9 โหนด b, c อยู่บนต้นไม้เดียวกัน ไม่สามารถเพิ่ม bc ในเซ็ต A

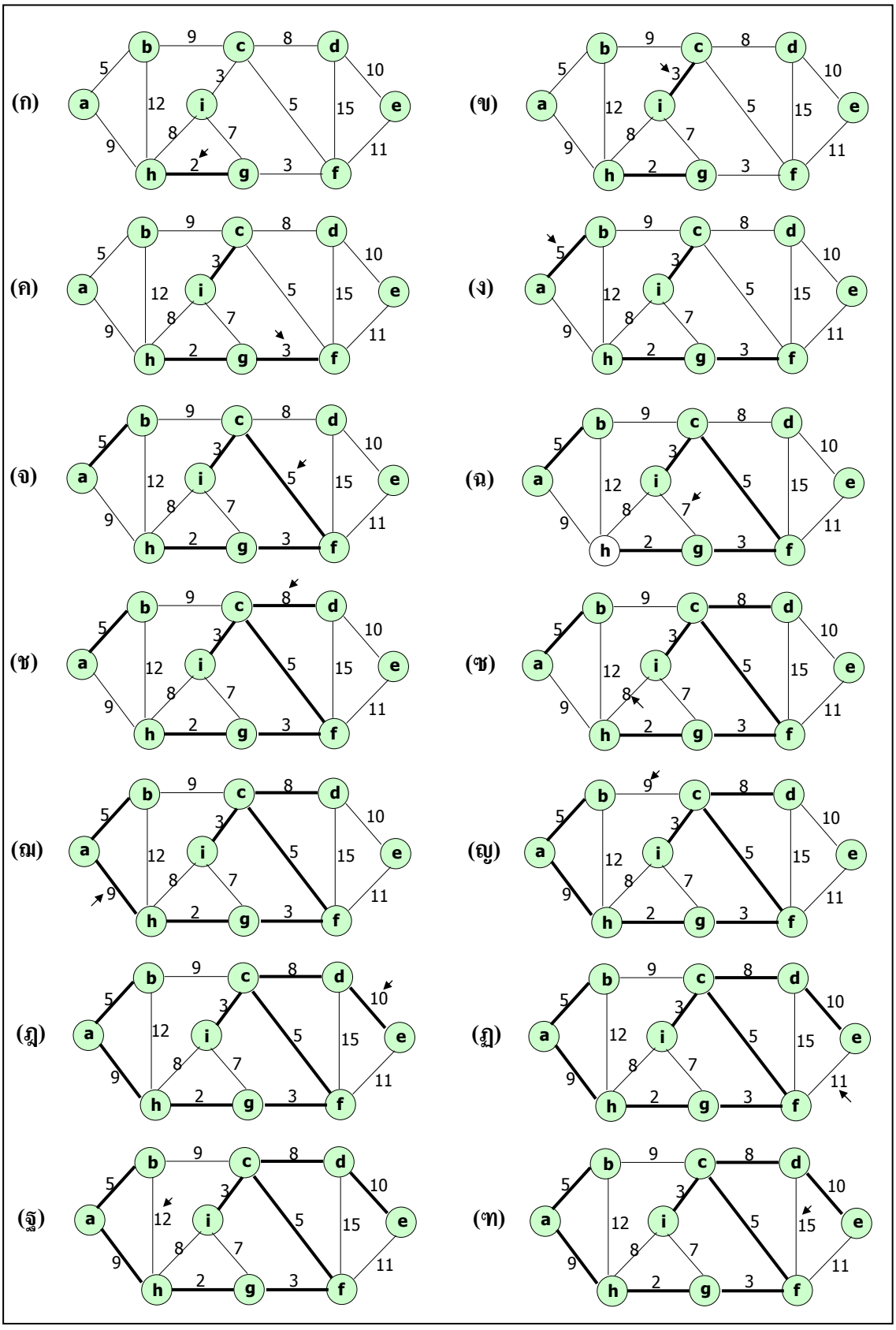
(ฎ) กิ่ง de = 10 โหนด d, e ไม่อยู่บนต้นไม้เดียวกัน เพิ่ม de ในเซ็ต A

(ฏ) กิ่ง ef = 11 โหนด e, f อยู่บนต้นไม้เดียวกัน ไม่สามารถเพิ่ม ef ในเซ็ต A

(ฐ) กิ่ง bh = 12 โหนด b, h อยู่บนต้นไม้เดียวกัน ไม่สามารถเพิ่ม bh ในเซ็ต A

(ฑ) กิ่ง df = 15 โหนด d, f อยู่บนต้นไม้เดียวกัน ไม่สามารถเพิ่ม df ในเซ็ต A

เมื่อพิจารณาครบทุกกิ่งในกราฟแล้ว จะได้เซ็ต A ประกอบกันเป็นกราฟต้นไม้ที่มีผลรวมของกิ่งสั้นที่สุด คือ = 2+3+3+5+5+8+9+10 = 45



รูป 8.9 Kruskal's algorithm โหนดรากคือ a, โหนดทึบ & กิ่งทึบ เป็นต้นไม้ที่กำลังขยาย

ประสิทธิภาพ เวลาที่ใช้ในขั้นตอนวิธีของ Kruskal สำหรับ Graph (V,E) จะประกอบด้วยส่วน MakeSet = O(V), การเรียงลำดับตามน้ำหนักของกิ่ง O(E log E), FindSet O(E) และ การ Union = O(log E) รวมเวลาทั้งหมด เป็น O(E log E) สังเกตว่า $E < V^2$ นั่นคือ log E = O(log V) จะได้ complexity = O(E log V)

8.4.2 วิธีการของ Prim

วิธีการของ PRIM จะเริ่มจากการกำหนดให้เซ็ต Q บรรจุโหนดทั้งหมดของกราฟ และกำหนดให้แต่ละโหนดมีค่า key เป็น ∞ แต่โหนดเริ่มต้นมีค่า key เป็น 0 และกำหนดค่า π[r] = NIL เพื่อแสดงว่าโหนด r ยังไม่มีพ่อแม่ หลังจากนั้นดึงโหนด u ที่มีค่า key น้อยที่สุดจากเซ็ต Q และพิจารณาโหนด v ที่เชื่อมโดยตรงกับโหนด u หากมีกิ่งจากโหนด u เชื่อมไปยัง v มีค่าน้อยกว่าค่า key เดิม ให้ key(v) = ค่าของกิ่งดังกล่าว และกำหนดให้โหนด v เชื่อมมาจากโหนด u โดยกำหนดให้ ค่า π(v) = u และทำการดึงโหนดที่มีค่า key น้อยที่สุดจากเซ็ต Q และปรับปรุงค่า key และค่า π ไปเรื่อย ๆ (เซ็ต Q จะค่อย ๆ มีขนาดเล็กลง) เมื่อ Q เป็นเซ็ตว่างจะได้ต้นไม้ที่มีผลรวมของกิ่งสั้นที่สุด

เมธตอด MST-Prim(Graph G, Edge w, int r)

```
MST-Prim(Graph G, Edge w, int r)   // รับค่าของกราฟ, ค่าของแต่ละกิ่ง และโหนดเริ่มต้น
{   for (each u ∈ V[G])  สำหรับแต่ละโหนด
      {   key[u] = ∞;
          π [u] = nil; }                  //   end for u
          key[r] = 0;                     // เฉพาะโหนดเริ่มต้น
     Q = V[G];                            // Q เป็นเซ็ตของโหนดของกราฟ G
     While (Q!=ϕ)                         // ขณะที่ Q ไม่เป็นเซ็ตว่าง
       {  u = Extract-Min(Q);             // ดึงโหนด u ที่มีค่า key น้อยที่สุดในเซ็ต Q
          for (each v ∈ adj[u])           //สำหรับโหนด v ที่เชื่อมโดยตรงกับ u
             { if (v ∈ Q && w[u][v] < key[v])        // ค่าของกิ่ง น้อยกว่าค่า key ของโหนด
                 {   π[v] = u;
                     key[v] = w[u][v];   } //   end if
             }// for v
       }
}// end method
```

<u>ตัวอย่าง 8.7</u> แสดงการหาต้นไม้ที่มีผลรวมของกิ่งสั้นที่สุดโดยวิธีของ PRIM ตามรูป 8.10

<u>วิธีทำ</u>

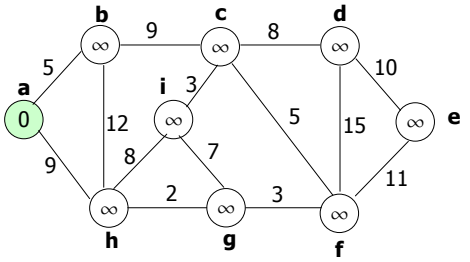
(ก) กำหนดให้ a เป็นโหนดเริ่มต้น

key(a) = 0 และ π (a) = NIL

เริ่มดึงโหนดที่มีค่า key น้อยที่สุดออกจาก Q จะได้โหนด a

ระบายสีโหนด a เพื่อแสดงว่าถูกดึงออกจาก Q แล้ว

(ก)

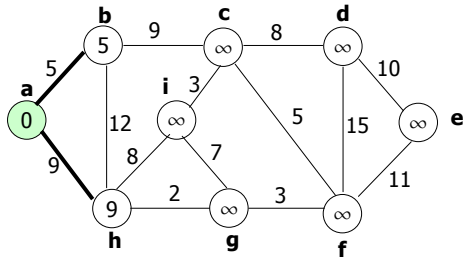


Q	a	b	c	d	e	f	g	h	i
key	0	∞	∞	∞	∞	∞	∞	∞	∞
π	nil	nil	nil	nil	nil	nil	nil	nil	nil

(ข) โหนด a เชื่อมโดยตรงกับโหนด b, h ทำการปรับปรุงค่า key และ π ให้แก่โหนด b, h

key(b) = 5 , π (b) = a, key(h) = 9 , π (h) = a

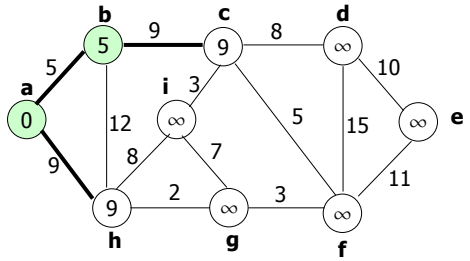
(ข)



Q	a	b	c	d	e	f	g	h	i
key	0	5	∞	∞	∞	∞	∞	9	∞
π	nil	a	nil	nil	nil	nil	nil	a	nil

(ค) ดึงโหนดที่มีค่า Key น้อยที่สุดออกจากคิว คือ โหนด b ซึ่งเชื่อมโดยตรงกับโหนด c, h ทำการปรับปรุงค่า key และ π ให้กับโหนด c สำหรับโหนด h เพราะค่า w[b][h] > key[h]

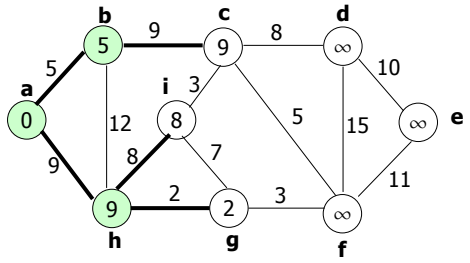
(ค)



Q	a	b	c	d	e	f	g	h	i
key	0	5	9	∞	∞	∞	∞	9	∞
π	nil	a	b	nil	nil	nil	nil	a	nil

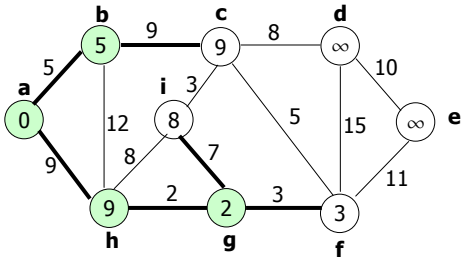
(ง) ดึงโหนดที่มีค่า Key น้อยที่สุดออกจากคิว คือ โหนด h ซึ่งเชื่อมโดยตรงกับโหนด i, g ทำการปรับปรุงค่า key และ π ให้กับโหนด i , g ทำเช่นนี้ไปเรื่อยๆ จนกระทั่งคิวว่าง ตามรูป (จ) – (ญ)

(ง)



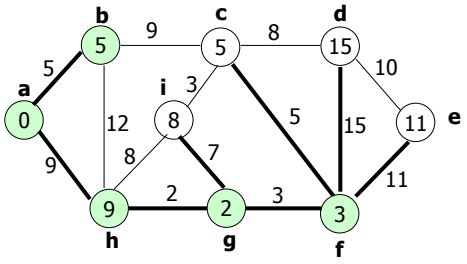
Q	a	b	h	c	d	e	f	g	i
key	0	5	9	9	∞	∞	∞	2	8
π	nil	a	a	b	nil	nil	nil	h	h

(จ)



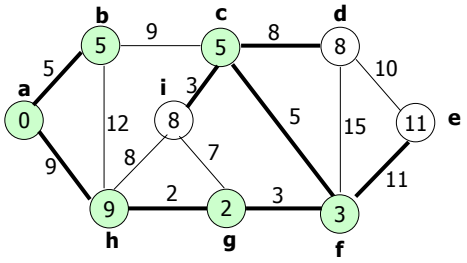
Q	a	b	h	g	c	d	e	f	i
key	0	5	9	2	9	∞	∞	3	7
π	nil	a	a	h	b	nil	nil	g	g

(ฉ)



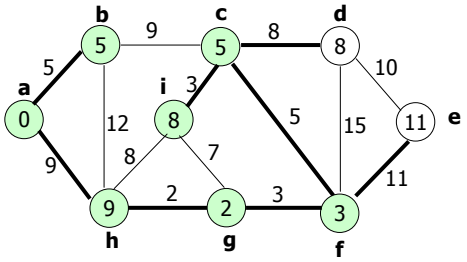
Q	a	b	h	g	f	c	d	e	i
key	0	5	9	2	3	5	15	11	7
π	nil	a	a	h	g	f	f	f	g

(ช)



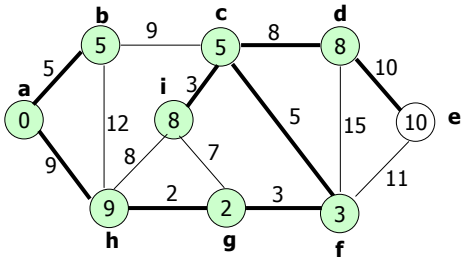
Q	a	b	h	g	f	c	d	e	i
key	0	5	9	2	3	5	15	11	3
π	nil	a	a	h	g	f	f	f	c

(ซ)



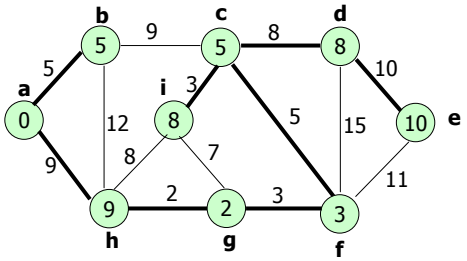
Q	a	b	h	g	f	c	i	d	e
key	0	5	9	2	3	5	3	8	11
π	nil	a	a	h	g	f	c	c	f

(ฌ)



Q	a	b	h	g	f	c	i	d	e
key	0	5	9	2	3	5	3	8	10
π	nil	a	a	h	g	f	c	c	d

(ญ)



Q	a	b	h	g	f	c	i	d	e
key	0	5	9	2	3	5	3	8	10
π	nil	a	a	h	g	f	c	c	d

รูป 8.10 Prim's algorithm โหนดเริ่มต้นคือ a, โหนดทึบเป็นโหนดที่ถูกดึงออกจากคิวแล้ว กิ่งทึบเป็นค่า π

เมื่อพิจารณาครบทุกโหนดในคิวแล้ว จะได้กราฟต้นไม้ที่มีผลรวมของกิ่งสั้นที่สุด คือ = 2+3+3+5+5+8+9+10 = 45

ประสิทธิภาพของวิธี Prim ขึ้นอยู่กับขั้นตอนวิธีทำคิวที่มีลำดับความสำคัญ ซึ่งที่เราศึกษาจากบทที่แล้ว ว่าด้วยการสร้างฮีปใช้เวลา O(V) แต่ละการดึงค่า key ต่ำสุดออกจากคิวใช้เวลา O(log V) รวมหมดทุกโหนดเป็น O(V log V) ในแต่ละโหนดที่ดึงออกจากคิวต้องพิจารณากิ่งที่เชื่อมอยู่กับโหนดที่ดึงออกมา เพื่อปรับปรุงค่า key และ ค่า π ใช้เวลา เท่ากับ O(E) เมื่อปรับปรุงค่า key แล้วก็จะต้องปรับฮีปอีก O(log V) รวมเวลาทั้งหมดของวิธี Prim จะเป็น O(V log V + E log V) = O(E log V)

8.5 เส้นทางที่สั้นที่สุดจากจุดเริ่มต้นจุดเดียว (Single-source Shortest Paths)

การหาเส้นทางที่สั้นที่สุด มักนำไปประยุกต์ใช้ในการขนส่ง การสื่อสาร เช่น การหาเส้นทางการเดินเรือที่สั้นที่สุดจากกรุงเทพไปเซี่ยงไฮ้ เส้นทางรถยนต์ที่สั้นที่สุดจากประจวบคีรีขันธ์ไปพิษณุโลก การขุดคลองส่งน้ำจากแหล่งน้ำไปยังที่ที่ต้องการให้มีระยะทางสั้นที่สุด การตัดถนนจากจุดหนึ่งไปยังอีกจุดหนึ่งให้มีระยะทางสั้นที่สุด เป็นต้น

วิธีการหาเส้นทางที่สั้นที่สุด 2 วิธี คือวิธีของ Bellman Ford กับวิธีของ Dijkstra

8.5.1 เส้นทางที่สั้นที่สุดวิธี Bellman Ford

ขั้นตอนวิธีของ Bellman Ford ใช้แก้ปัญหาเส้นทางที่สั้นที่สุดจากโหนดเริ่มต้นโหนดเดียวไปยังปลายทางโหนดอื่นๆ ในกราฟ โดยที่กิ่งอาจจะมีค่าติดลบก็ได้ จากกราฟที่กำหนดให้ G = (V, E), จุดเริ่มต้น S และ น้ำหนักของกิ่ง (w) ขั้นตอนวิธีของ Bellman Ford จะคืนค่าเป็นเท็จ ถ้ากราฟเป็นวงและวงนั้นมีค่าติดลบ แสดงว่าหาคำตอบไม่ได้ หรือถ้ากราฟไม่เป็นวงจะคืนค่าเป็นจริงและแสดงกราฟที่เป็นเส้นทางที่สั้นที่สุด

ขั้นตอนวิธีของ Bellman Ford

```
Bellman-Ford (Graph G, Edge w, char s)  //รับกราฟพร้อมทั้งค่าของแต่ละกิ่งและโหนดเริ่มต้น
{    int i;
     Initialize-Single-Source(G, s);
    for( i = 1 ; i< |V[G]|; ++i)                  // สำหรับแต่ละโหนด
          {  for (each edge [u][v] ∈ E[G])    // สำหรับแต่ละกิ่งที่เชื่อมระหว่างโหนด u กับโหนด v
               { Relax(u,v,w); }                  // เปลี่ยนเส้นทางถ้าได้ระยะทางสั้นกว่า
          }
     for (each edge [u][v] ∈ E[G])
          { if  (d[v] > d[u]+w[u][v])
                    {return false;}               // แสดงว่าเกิดเป็นวงมีค่าติดลบ
          }
    return true;
} // end Bellman-Ford
```

```
Initialize-Single-Source(Graph g, char s)
{   for (each vertex v  ∈ V[G])
      { d[v] = ∞ ;
        π [v] = nil;
       }
     d[s]=0;
}// end Initialize-single-Source
```

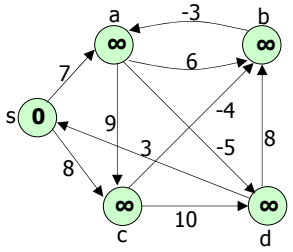
```
Relax(Node u, Node v, Edge w)          // เปลี่ยนเส้นทางถ้าได้ระยะทางสั้นกว่า
{  if  (d[v]  > d[u]+w[u][v])
      {  d[v] = d[u]+w[u][v];
         π [v] = u;
      } //end if
}// end Relax
```

<u>ตัวอย่าง 8.8</u> จงหาเส้นทางที่สั้นที่สุดของ กราฟในรูป 8.11(ก) ด้วยวิธีของ Bellman Ford

<u>วิธีทำ</u>

<u>รูป 8.11</u> แสดงขั้นตอนวิธีของ Bellman Ford

(ก)



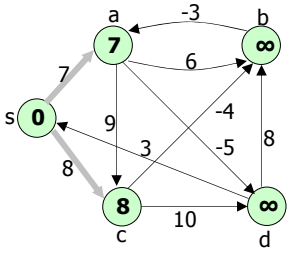
vertex	s	a	b	c	d
d	0	∞	∞	∞	∞
π	nil	nil	nil	nil	nil

(ก) โหนดเริ่มต้นเป็น s กำหนดค่าเริ่มต้น d ให้ทุกโหนดมีค่าเป็น ∞ และ π ของทุกโหนดเป็น nil

(ข) จากโหนดเริ่มต้น s ➔ a จะได้ d[a] = 0+7 = 7, π [a] = s

จากโหนดเริ่มต้น s ➔ c จะได้ d[c] = 0+8 = 8, π [c] = s

(ข)

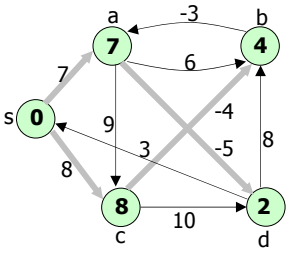


vertex	s	a	b	c	d
d	0	7	∞	8	∞
π	nil	s	nil	s	nil

(ค) โหนด a ➔ d จะได้ d[d] = 7 +(-5) = 2, π [d] = a

โหนด c ➔b จะได้ d[b] = 8+(-4) =4, π [b] = c

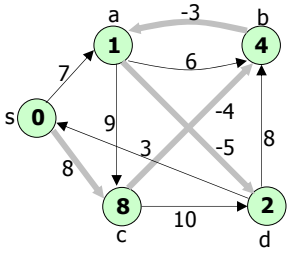
(ค)



vertex	s	a	b	c	d
d	0	7	4	8	2
π	nil	s	c	s	a

(ง) โหนด b ➔ a จะได้ d[a] = 4 +(-3) = 1, π [a] = b

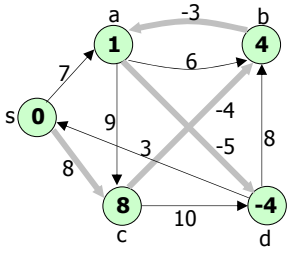
(ง)



vertex	s	a	b	c	d
d	0	1	4	8	2
π	nil	b	c	s	a

(จ) ปรับปรุงโหนด d จะได้ d[d] = 1+(-5) = -4, π [d] = a

(จ)



vertex	s	a	b	c	d
d	0	1	4	8	-4
π	nil	b	c	s	a

จะได้เส้นทางที่สั้นที่สุด ตามตาราง

ประสิทธิภาพ จากตัวอย่างจำนวน 5 โหนด เราพิจารณารอบ i ตามจำนวนโหนดของกราฟ – โหนดเริ่มต้น = 4 รอบ ในแต่ละรอบพิจารณากิ่งที่เชื่อมระหว่างโหนด(E) ทั้ง 10 กิ่ง ดังนั้นเวลาที่ใช้ทั้งหมดตามขั้นตอนวิธีของ Bellman Ford เท่ากับ O(VE)

8.5.2 เส้นทางที่สั้นที่สุดวิธี Dijkstra

วิธีการของ Dijkstra จะกำหนดระยะทางที่สั้นที่สุดจากโหนดเริ่มต้น s ไปยังโหนดต่าง ๆ เป็น ∞ และกำหนด π เป็น NIL กำหนดค่า d(s) = 0 และ π (s) = nil ให้เซ็ต Q บรรจุโหนดทั้งหมดของกราฟ ในแต่ละขั้นตอนจะดึงโหนด u ที่มีค่า d ต่ำสุดออกและเพิ่มเข้าไปในเซ็ต S สำหรับแต่ละ v ที่เชื่อมโดยตรงกับ u ให้ทำการปรับปรุงค่า d(v) หากค่า d(v) เดิมมากกว่า d(u) + ค่าของกิ่งที่เชื่อม uv ดังกล่าว ให้ทำเช่นนี้ซ้ำไปเรื่อย ๆ จนเซ็ต Q กลายเป็นเซ็ตว่าง และเซ็ต S บรรจุทุกโหนดในกราฟ วิธีของ Dijkstra จะใช้ไม่ได้กับกรณีที่น้ำหนักของกิ่งมีค่าเป็นลบ

ขั้นตอนวิธีของ Dijkstra

```
Dijkstra(Graph G, Edge w, char s)  //รับกราฟพร้อมทั้งค่าของแต่ละกิ่งและโหนดเริ่มต้น s
{   Initialize-Single-Source ;          // กำหนดให้ทุกโหนดมีค่า d =∞, d[s] = 0
    S = ϕ;                              //  S เป็นเซ็ตว่าง
    Q= V[G];                            //  Q เป็นเซ็ตของโหนดในกราฟ (priority queue)
    while (Q!= ϕ)                       //  ขณะที่  Q  ไม่เป็นเซ็ตว่าง
    { u = ExtractMin(Q);                //  ดึงโหนดที่มีค่า d น้อยสุด
      S = S ∪ {u}                       //  เก็บ u ไว้ในเซ็ต S
     for (each vertex v ∈ Adj[u])       // สำหรับแต่ละโหนดที่เชื่อมต่อกับ u
       {  relax(u,v,w); }               //  ตรวจสอบว่าจะมีเส้นทางที่สั้นกว่า? ให้ปรับค่า d, π
     }//end while
}//end Dijkstra
```

```
Initialize-Single-Source(Graph g, char s)
{   for (each vertex v  ∈ V[G])
      { d[v] = ∞ ;
        π [v] = nil;
       }
     d[s]=0;
}// end Initialize-single-Source
```

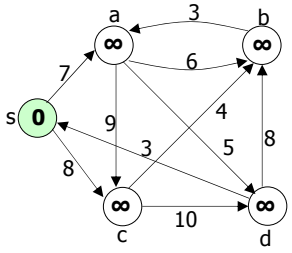
```
Relax(Node u, Node v, Edge w)        // เปลี่ยนเส้นทางถ้าได้ระยะทางสั้นกว่า
{ if  (d[v]  > d[u]+w[u][v])
      {  d[v] = d[u]+w[u][v];
         π [v] = u;
      } //end if
}// end Relax
```

ตัวอย่าง 8.9 จงหาเส้นทางที่สั้นที่สุดของ กราฟในรูป 8.12(ก) ด้วยวิธีของ Dijkstra

วิธีทำ

รูป 8.12 แสดงขั้นตอนวิธีของ Dijkstra

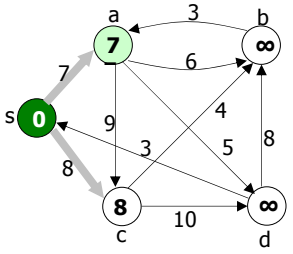
(ก)



Q	s	a	b	c	d
S = { }					
d	0	∞	∞	∞	∞
π	nil	nil	nil	nil	nil

(ก) โหนดเริ่มต้นเป็น s กำหนดค่าเริ่มต้น d ให้ทุกโหนดมีค่าเป็น ∞ **และ** π ของทุกโหนดเป็น nil

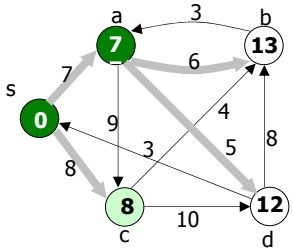
(ข)



Q	s	a	b	c	d
S = {s}					
d	0	7	∞	8	∞
π	nil	s	nil	s	nil

(ข) ดึง s ออกจาก Q พิจารณาโหนด a, c จะได้ d[a] = 7, d[c] = 8 และ π [a] = π [c] = s แล้วเก็บ s ในเซ็ต S

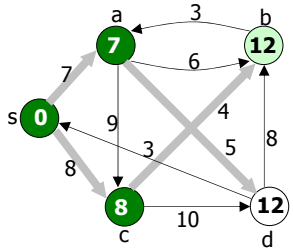
(ค)



Q	s	a	b	c	d
S = {s, a}					
d	0	7	13	8	12
π	nil	s	a	s	a

(ค) ดึง a ออกจาก Q พิจารณาโหนด b, d จะได้ d[b] = 13, d[d] = 12 และ π [b] = π [d] = a แล้วเก็บ a ในเซ็ต S

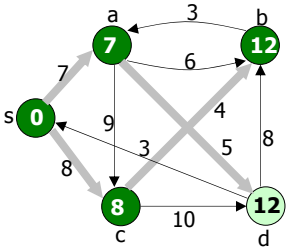
(ง)



Q	s	a	b	c	d
S = {s, a, c}					
d	0	7	12	8	12
π	nil	s	c	s	a

(ง) ดึง c ออกจาก Q พิจารณาโหนด b, d จะได้ d[b] = 12 และ π [b] = c แล้วเก็บ c ในเซ็ต S

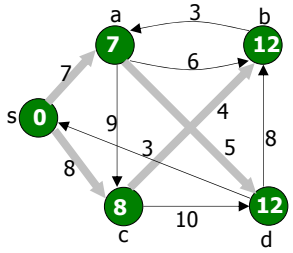
(จ)



Q	s	a	b	c	d
S = {s, a, c, b}					
d	0	7	12	8	12
π	nil	s	c	s	a

(จ) ดึง b ออกจาก Q เก็บ b ในเซ็ต S

(ฉ)



Q	s	a	b	c	d
S = {s, a, c, b, d}					
d	0	7	12	8	12
π	nil	s	c	s	a

(ฉ) ดึง d ออกจาก Q เก็บ d ในเซ็ต S

จะได้เส้นทางที่สั้นที่สุด ตามตาราง

ประสิทธิภาพ ขั้นตอนวิธีของ Dijkstra จะมีการใช้ minimum priority queue

- method Insert เพิ่มโหนดของกราฟเข้าไปในเซ็ต Q ตามจำนวนโหนด เวลาที่ใช้ในการเพิ่มโหนดเท่ากับ O(1)
- method ExtractMin ดึงโหนดออกจาก Q ตามจำนวนโหนด เวลาที่ใช้ในการ ExtractMin เท่ากับ O(log V) รวมเป็น O (V log V)
- method DecreaseKey เมื่อมีการปรับปรุงค่า d หลังการ relax จำนวนครั้งตามจำนวนกิ่งที่เชื่อม เท่ากับ |E| เวลาที่ใช้เท่ากับ O(E log V)

รวมเวลาทั้งหมดเป็น O((V+E) log V) หรือ O(E log V)

8.6 เส้นทางที่สั้นที่สุดจากทุกโหนดไปยังโหนดอื่นจนทั่ว (All-Pairs Shortest Paths)

จากหัวข้อ 8.5 หาเส้นทางที่สั้นที่สุดจากโหนดเริ่มต้นไปยังโหนดอื่นๆ ได้ครบทุกโหนด คราวนี้จะหาเส้นทางที่สั้นที่สุดโดยจะเปลี่ยนให้โหนดทุกโหนดเป็นโหนดเริ่มต้น(all-Pairs shortest Paths) ถ้าแต่ละกิ่งค่าน้ำหนักไม่ติดลบเราสามารถใช้วิธี Dijkstra สำหรับทุกโหนด เวลาที่ใช้จะเป็น $O(V^2 \log V + VE)$ สำหรับกิ่งที่มีค่าติดลบจะใช้วิธีของ Bellman Ford จะใช้เวลาเท่ากับ $O(V^2E)$

ในตอนนี้เราจะแทนกราฟ G = (V,E) ด้วยเมทริกซ์ ขนาด n x n โดยที่ n เท่ากับจำนวนโหนด และค่าน้ำหนักของกิ่ง (W) จะแทนด้วยสมาชิกในเมทริกซ์ โดยที่

$$w[i][j] = \begin{cases} 0 & if\ i = j \\ weight(i,j) & if\ i \neq j\ and\ (i,j) \varepsilon E \\ \infty & if\ i \neq j\ and\ (i,j) \notin E \end{cases}$$

ในที่นี้ยอมให้มีค่าน้ำหนักติดลบ แต่ไม่เป็นวงของค่าติดลบ (no negative-weight cycle) ขั้นตอนวิธีที่จะกล่าวในหัวข้อนี้ 2 วิธี ได้แก่

- Slow all Pair Shortest Paths
- Floyd Warshall

8.6.1 ขั้นตอนวิธี Slow all Pair Shortest Paths

ขั้นตอนวิธีนี้จะเป็นแบบกำหนดการพลวัติ ที่จะต้องกำหนด โครงสร้างในการหาคำตอบที่ดีที่สุดโดย

1. หาความสัมพันธ์ซ้ำที่จะใช้ในการแก้ปัญหา
2. แก้ปัญหาจากล่างขึ้นบน โดยแก้ปัญหาย่อยก่อน และเก็บผลลัพธ์จากปัญหาย่อยใส่ลงในอาเรย์เพื่อใช้ในการแก้ปัญหาที่ใหญ่ขึ้น

วิธีนี้จะทำการเรียกซ้ำ (recursive) เพื่อจะหาค่าระยะทาง(length) สั้นที่สุดของเส้นทางต่างๆ จากโหนด i ไปโหนด j กำหนดให้

$l_{i,j}^{(m)}$ = ระยะทางสั้นที่สุดของเส้นทางต่างๆ จากโหนด i ไปโหนด j โดยผ่าน m กิ่ง

ที่ $m = 0$ จะมีเส้นทางที่สั้นสุด ที่ไม่มีกิ่งเชื่อมเลย ก็คือกรณี i = j ดังนั้น

$$l_{i,j}^{(0)} = \begin{cases} 0 & if\ i = j \\ \infty & if\ i \neq j \end{cases}$$

สำหรับ m ≥ 1 จะคำนวณ $l_{i,j}^{(m)}$ จากค่าต่ำสุดของ $l_{i,j}^{(m-1)}$ (ระยะทางสั้นที่สุดของเส้นทางต่างๆ จากโหนด i ไปโหนด j โดยผ่าน *m-1* กิ่ง) เราสามารถกำหนดการเรียกซ้ำได้ดังนี้

$$l_{i,j}^{(m)} = \min_{1 \le k \le n} \left\{ l_{i,k}^{(m-1)} + w_{k,j} \right\}$$

$\pi_{i,j}$ = โหนดสุดท้ายก่อนถึงโหนด j ที่ทำให้ได้เส้นทางสั้นที่สุดจาก i ถึง j จะเท่ากับ nil ถ้า i = j หรือ ไม่มีเส้นทางจาก i ถึง j

$$\pi_{i,j} = \begin{cases} nil & if\ i = j\ \ or\ w_{i,j} = \infty \\ V_{\pi,i} & vertex\ \ k \end{cases}$$

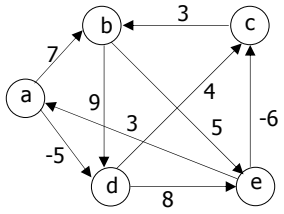
ขั้นตอนวิธี Slow all Pair Shortest Paths

```
SlowAllPairsShortestPaths(Weight W)  // รับค่าน้ำหนักของกราฟ
{   n =  rows[W]                                  // จำนวนแถวของน้ำหนัก หรือ จำนวนโหนด
    L(1) = W          // กำหนดระยะทางสั้นที่สุด ให้ผ่าน 1 กิ่ง ซึ่งก็จะเท่ากับน้ำหนักของกิ่งต่างๆ
    for (m=2; m<= n-1; ++m)
      { L(m) =  ExtendShortestPaths(L(m-1),  W)   }
    return L(n-1)
} //end SlowallPairsShortestPaths
```

```
ExtendShortestPaths(L, W)
{   n = rows[L]
    L' = (l'i,j)   // กำหนดให้ L เป็น matrix ขนาด n x n
    for (i=1 ; i<= n; ++i)
      {  for (j=1; j<=n; ++j)
         {  l'i,j = ∞
            for (k=1; k<=n; ++k)
               { l'i,j = min(l'i,j, li,k + wk,j) } //end for k
          } //  end for j
      } //  end for i
    return L'
} //  end ExtendShortestPaths
```

```
PrintAllPairsShortestPath (π , i, j)
{  if   (i = j)
       { print ( i ) }  //end if
   else if ( πi,j  = nil)
       { print ("no path from" + i + " to"+ j + "exist") }  //end else if
  else
       { PrintAllPairsShortestPath ( π , i,  πi,j  )
         print (j)
       } // end else
}// end PrintAllPairsShortestPath
```

ตัวอย่าง 8.10 จงหาเส้นทางสั้นที่สุดจากทุกโหนดของกราฟในรูป 8.13



รูป 8.13 กราฟเพื่อหาเส้นทางสั้นที่สุดจากทุกโหนด

วิธีทำ

	0	7	∞	-5	∞
	∞	0	∞	9	5
	∞	3	0	∞	∞
	∞	∞	4	0	8
	3	∞	-6	∞	0

$\Pi^{(1)}$	nil	a	nil	a	nil
	nil	nil	nil	b	b
	nil	c	nil	nil	nil
	nil	nil	d	nil	d
	e	nil	e	nil	nil

	0	7	-1	-5	3
	8	0	-1	9	5
$L^{(2)}$	∞	3	0	12	8
	11	7	2	0	8
	3	-3	-6	-2	0

$\Pi^{(2)}$	nil	a	d	a	d
	e	nil	e	b	b
	nil	c	nil	b	b
	e	c	e	nil	d
	e	c	e	a	nil

	0	2	-3	-5	3
	8	0	-1	3	5
$L^{(3)}$	11	3	0	12	8
	11	5	2	0	8
	3	-3	-6	-2	0

$\Pi^{(3)}$	nil	a	e	a	d
	e	nil	e	a	b
	e	c	nil	b	b
	e	c	e	nil	d
	e	c	e	a	nil

	0	0	-3	-5	3
	8	0	-1	3	5
$L^{(4)}$	11	3	0	12	8
	11	5	2	0	8
	3	-3	-6	-2	0

$\Pi^{(4)}$	nil	c	e	a	d
	e	nil	e	a	b
	e	c	nil	b	b
	e	c	e	nil	d
	e	c	e	a	nil

ประสิทธิภาพของขั้นตอนวิธี Slow all Pair Shortest Paths เวลาที่ใช้จะเท่ากับ $O(n^3)$

8.6.2 ขั้นตอนวิธีของ Floyd Warshall

ในส่วนนี้เราจะใช้กำหนดการพลวัตกับเส้นทางที่สั้นที่สุดจากทุกโหนดไปยังโหนดอื่นๆ ในมุมที่จะพิจารณาที่เป็นโหนดผ่าน[1] (intermediate vertex) เป็นหลัก แทนที่จะพิจารณาจากโหนดเริ่มต้นและปลายทางเป็นหลักเช่นวิธีการก่อนหน้านี้

กำหนดโครงสร้างในการเรียกซ้ำ (recursive) เพื่อจะหาค่าระยะทาง(distance) สั้นที่สุดของเส้นทางต่างๆ จากโหนด i ไปโหนด j กำหนดให้

$$d_{i,j}^{(k)} = \begin{cases} w_{i,j} & \text{if } k = 0 \\ \min\left(d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)}\right) & \text{if } k \geq 1 \end{cases}$$

$$\pi_{i,j}^{(0)} = \begin{cases} nil & \text{if } i = j \text{ or } w_{i,j} = \infty \\ i & \text{if } i \neq j \text{ and } w_{i,j} < \infty \end{cases}$$

$$\pi_{i,j}^{(k)} = \begin{cases} \pi_{i,j}^{(k-1)} & \text{if } d_{i,j}^{(k-1)} \leq d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)} \\ \pi_{k,j}^{(k-1)} & \text{if } d_{i,j}^{(k-1)} \geq d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)} \end{cases}$$

เมื่อ k = 0 เส้นทางระหว่างโหนด i ถึง โหนด j จะไม่มีโหนดผ่านดังนั้น $d_{i,j}^{(0)} = w_{i,j}$

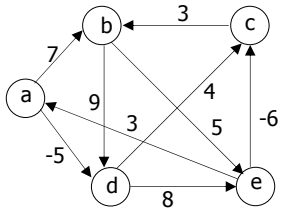
เมื่อ k≥ 1 เส้นทางระหว่างโหนด i ถึง โหนด j จะเลือกว่าผ่านโหนดที่ทำให้มีระยะทางสั้นที่สุด

ขั้นตอนวิธีของ Floyd Warshall

```
FloydWarshall(W)        // รับค่าเมทริกซ์ที่แทนน้ำหนักของกิ่งต่างๆในกราฟ
{   n = rows[W]         //  n เป็นจำนวนแถวของเมทริกซ์ W
    D(0)  =  W          // กำหนดระยะทางที่ไม่มีโหนดผ่าน เท่ากับน้ำหนักของกิ่งในกราฟ
    for (k =1; k<=n; ++k)
      {  for (i = 1; i<=n; ++i)
          {   for (j=1;j<=n;++j)
                { d(k)i,j = min(d(k-1)i,j , d(k-1)i,k + d(k-1)k,j ) } // end for j
           } // end for i
       } // end for k
    return D(n)
}// end FloydWarshall
```

[1] โหนดที่เป็นเส้นทางผ่านไปโหนดอื่นๆ

ตัวอย่าง 8.10 จงหาเส้นทางสั้นที่สุดจากทุกโหนดของกราฟในรูป 8.14



รูป 8.13 กราฟเพื่อหาเส้นทางสั้นที่สุดจากทุกโหนด

วิธีทำ

$D^{(0)}$

0	7	∞	-5	∞
∞	0	∞	9	5
∞	3	0	∞	∞
∞	∞	4	0	8
3	10	-6	-2	0

$\P^{(0)}$

nil	a	nil	a	nil
nil	nil	nil	b	b
nil	c	nil	nil	nil
nil	nil	d	nil	d
e	nil	e	nil	nil

$D^{(1)}$

0	7	∞	-5	∞
∞	0	∞	9	5
∞	3	0	∞	∞
∞	∞	4	0	8
3	10	-6	-2	0

$\P^{(1)}$

nil	a	nil	a	nil
nil	nil	nil	b	b
nil	c	nil	nil	nil
nil	nil	d	nil	d
e	a	e	a	nil

$D^{(2)}$

0	7	∞	-5	12
∞	0	∞	9	5
∞	3	0	12	8
∞	∞	4	0	8
3	10	-6	-2	0

$\P^{(2)}$

nil	a	nil	a	nil
nil	nil	nil	b	b
nil	c	nil	b	b
nil	nil	d	nil	d
e	a	e	a	nil

$D^{(3)}$

0	7	∞	-5	12
∞	0	∞	9	5
∞	3	0	12	8
∞	7	4	0	8
3	-3	-6	-2	0

$\P^{(3)}$

nil	a	nil	a	nil
nil	nil	nil	b	b
nil	c	nil	b	b
nil	c	d	nil	d
e	c	e	a	nil

$D^{(4)}$

0	2	-1	-5	12
∞	0	∞	9	5
∞	3	0	12	8
∞	7	4	0	8
3	-3	-6	-2	0

$\P^{(4)}$

nil	c	d	a	nil
nil	nil	nil	b	b
nil	c	nil	b	b
nil	c	d	nil	d
e	c	e	a	nil

$D^{(5)}$

0	2	-1	-5	12
∞	0	-1	3	5
11	3	0	12	8
11	5	2	0	8
3	-3	-6	-2	0

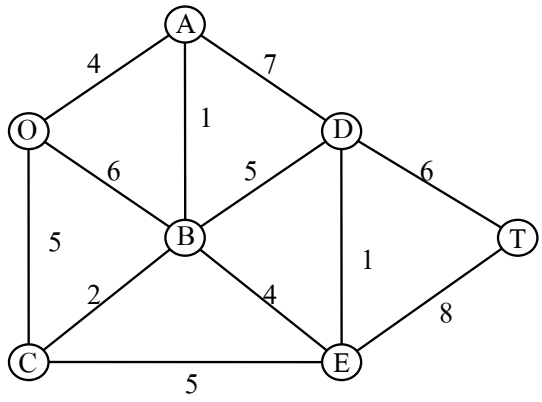
$\P^{(5)}$

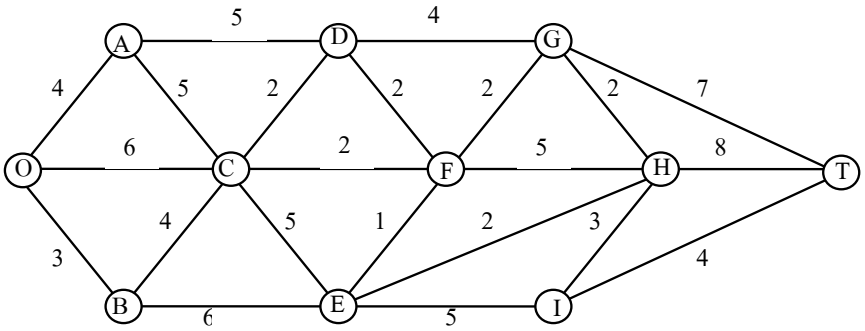
nil	c	d	a	nil
nil	nil	e	e	b
e	c	nil	b	b
e	c	e	nil	d
e	c	e	a	nil

ประสิทธิภาพของขั้นตอนวิธีโดย Floyd Warshall เวลาที่ใช้จะเท่ากับ $O(n^3)$

แบบฝึกหัด

1. จงหากราฟต้นไม้ที่มีผลรวมของกิ่งสั้นที่สุด(minimum spanning tree) โดยวิธี Kruskal และวิธี Prim

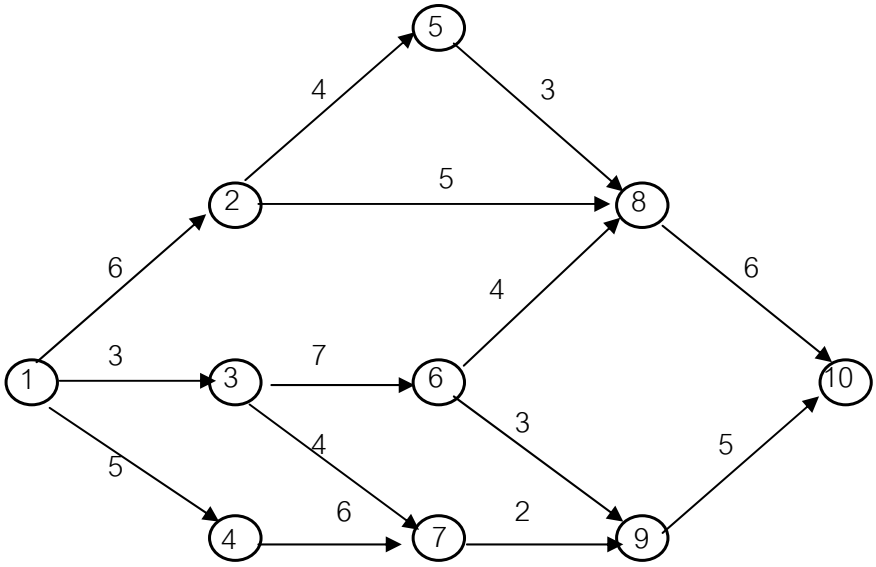




2. มีเกาะจำนวน 8 เกาะในทะเลสาบ เทศบาลต้องการสร้างสะพานจำนวน 7 สะพาน เพื่อให้สามารถไปมาหาสู่ระหว่างเกาะทั้งหมดได้ โดยค่าใช้จ่ายในการก่อสร้างสะพานจะเป็นสัดส่วนกับความยาวของสะพาน ถ้าระยะห่างของแต่ละเกาะเป็นตามตารางข้างล่าง จงหาว่าจะสร้างสะพานที่ใดบ้าง เพื่อทำให้ค่าก่อสร้างสะพานต่ำสุด

	1	2	3	4	5	6	7	8
1	-	240	210	340	280	200	345	120
2	-	-	265	175	2158	180	185	155
3	-	-	-	260	115	350	435	195
4	-	-	-	-	160	330	295	230
5	-	-	-	-	-	360	400	170
6	-	-	-	-	-	-	175	205
7	-	-	-	-	-	-	-	305
8	-	-	-	-	-	-	-	-

3. จงแสดงการสืบค้นกราฟ breadth first search, depth first search



4. จงหา shortest path ของกราฟในข้อ 3 โดยเริ่มจากโหนด 1 ไปยังโหนดอื่นๆ โดยวิธี Bellman ford และ Dijkstra
5. จงใช้วิธีการของ Slow-All-Pairs-Shortest-Paths และ Floyd - Warshall ในการหา all pairs shortage pathsของกราฟข้างล่างนี้