บทที่ 12 Heap และ Graph

บทเรียนย่อย

12.1 Heap Concept

12.2 Heap Implementation

12.3 Graph Concept

12.4 Graphs Notations

12.5 Graph Traversals

12.6 Shortest Path Algorithm

วัตถุประสงค์

- นิสิตมีความรู้ และความเข้าใจเกี่ยวกับแนวคิดในการ จัดการโครงสร้างข้อมูลในรูปแบบของ Heaps
- นิสิตสามารถเขียนโปรแกรมเพื่อดำเนินการตามแนวคิดของ Heaps
- นิสิตมีความรู้ และความเข้าใจเกี่ยวกับแนวคิดในการ จัดการโครงสร้างข้อมูลในรูปแบบของ Graphs
- นิสิตมีความรู้ และความเข้าใจขั้นตอนการดำเนินการต่าง ๆ ในการจัดการโครงสร้างข้อมูลในรูปแบบของ Graphs

บทที่ 12 Heap และ Graph

บทเรียนย่อย

12.1 Heap Concept

12.2 Heap Implementation

12.3 Graph Concept

12.4 Graphs Notations

12.5 Graph Traversals

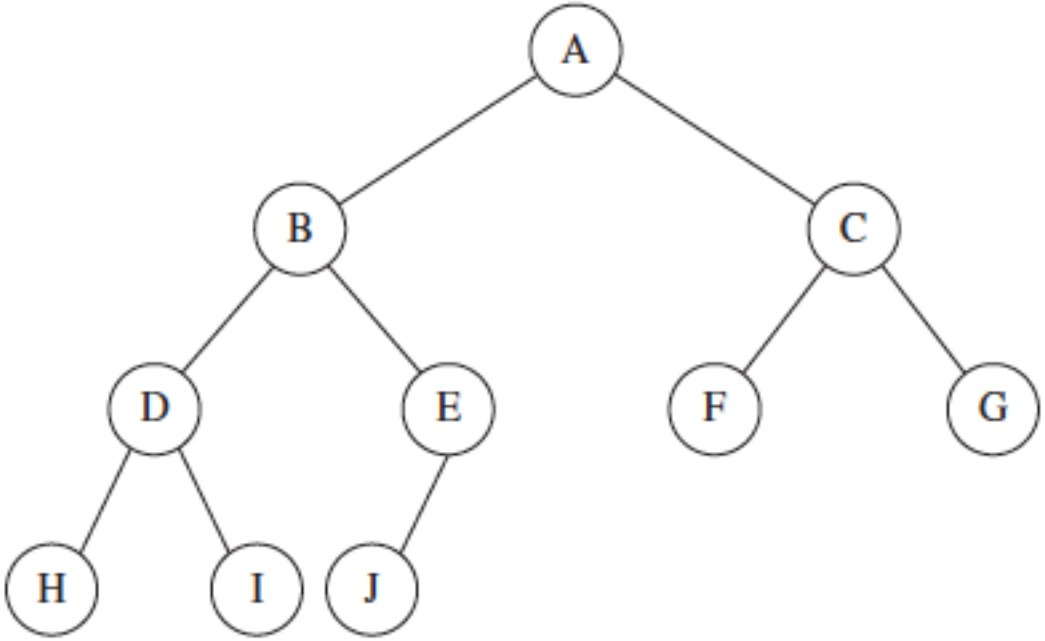
12.6 Shortest Path Algorithm

12.1 Heap Concept

ฮีป (Heap) หรือ Priority Queue เป็นโครงสร้างข้อมูลที่นำมาสร้างแถวคอยลำดับความสำคัญ (Priority queue) รูปแบบหนึ่ง ซึ่งนิยมใช้กันมาก โดยฮีปที่สร้างขึ้นโดยอาศัยแนวคิดจากต้นไม้ทวิภาคใช้ชื่อว่า “ฮีปทวิภาค” (Binary heap) โดยฮีป นั้นมีความสัมพันธ์เหมือนกันคือโหนดพ่อ (P) มีลำดับความสำคัญมากกว่าโหนดลูก (C)

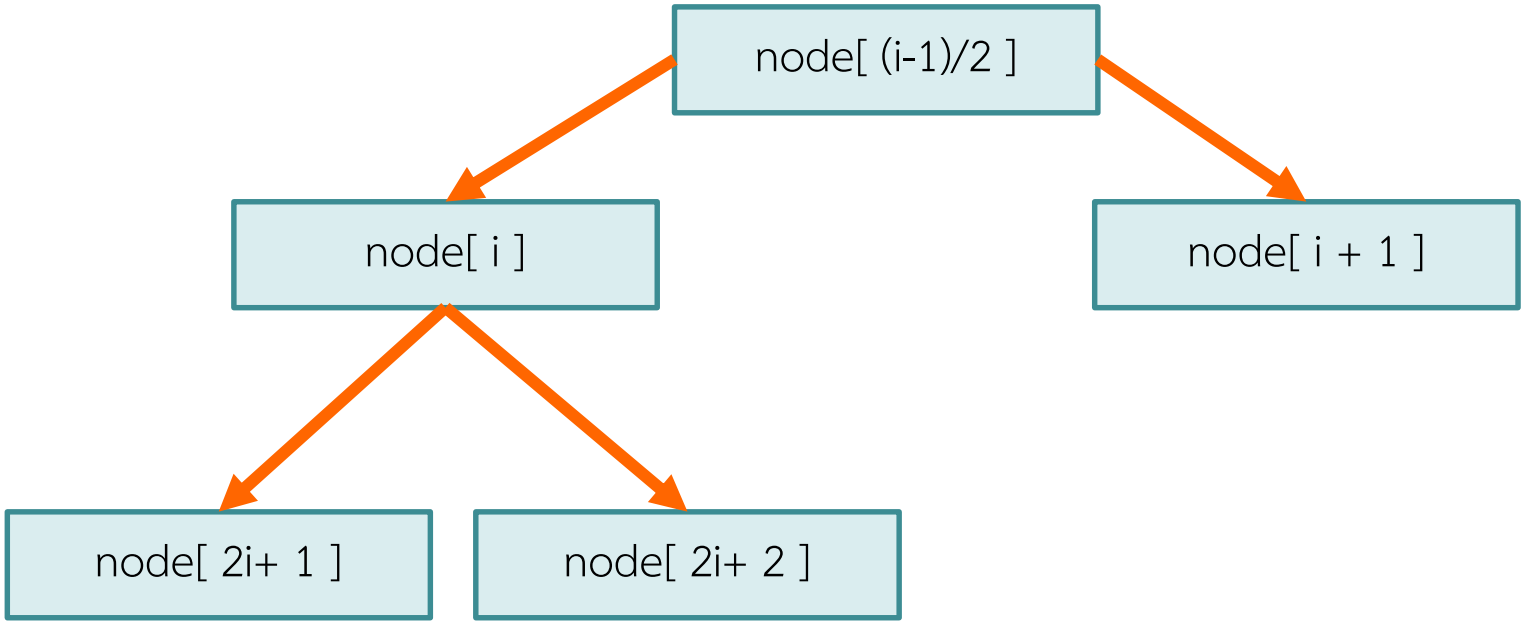
$$P >= C$$

12.1 Heap Concept



	A	B	C	D	E	F	G	H	I	J			
0	1	2	3	4	5	6	7	8	9	10	11	12	13

12.1 Heap Concept



บทที่ 12 Heap และ Graph

บทเรียนย่อย

12.1 Heap Concept

12.2 Heap Implementation

12.3 Graph Concept

12.4 Graphs Notations

12.5 Graph Traversals

12.6 Shortest Path Algorithm

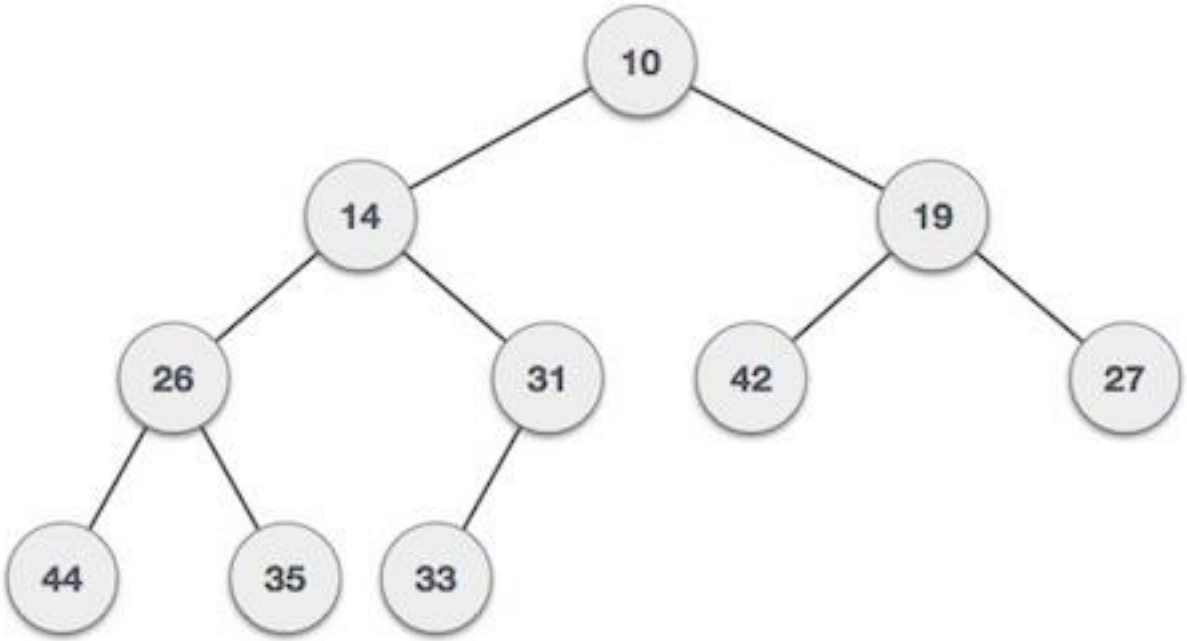
12.2 Heap Implementation

ในการ Implement มีด้วยกัน

- Max Heap
- Min Heap

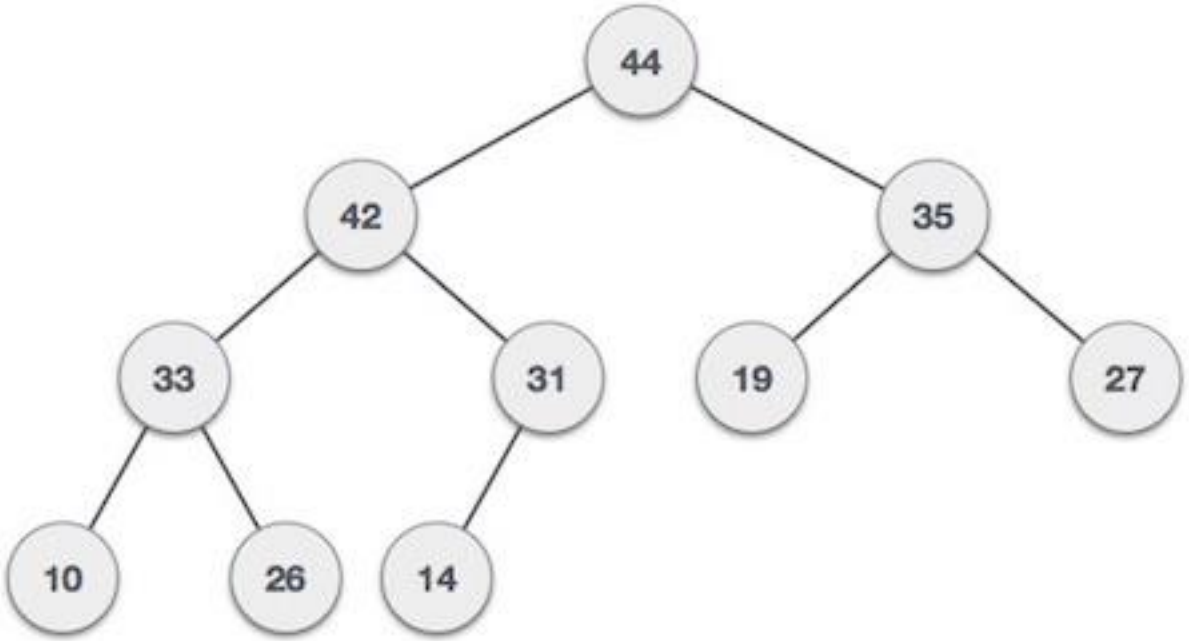
12.2 ตัวอย่างของ Min - Heap

กรณีมีข้อมูล ⟶ 35 33 42 10 14 19 27 44 26 31



12.2 ตัวอย่างของ Max - Heap

กรณีมีข้อมูล ⟶ 35 33 42 10 14 19 27 44 26 31



Max – Heap ในการเพิ่มค่า

Step 1 – สร้างโหนดไว้ตำแหน่งสุดท้ายของฮีป

Step 2 – กำหนดให้ค่าใหม่ไปที่โหนดนั้น

Step 3 – เปรียบเทียบค่าของ ตัวลูกโหนด (Child node) กับ โหนดพ่อแม่ (Parent node)

Step 4 – ถ้าค่าของ โหนดพ่อแม่ มีค่าน้อยกว่า ให้ทำการ ย้ายค่า (Swap)

Step 5 – ทำ Step 3 และ Step 4 ซ้ำจนกว่าจะไม่มีการย้ายค่า หรือไม่มีโหนดพ่อแม่ (เป็น Root node)

Max – Heap ในการเพิ่มค่า

Input 35 33 42 10 14 19 27 44 26 31

Max – Heap ในการลบค่า

Step 1 – ลบ Root node

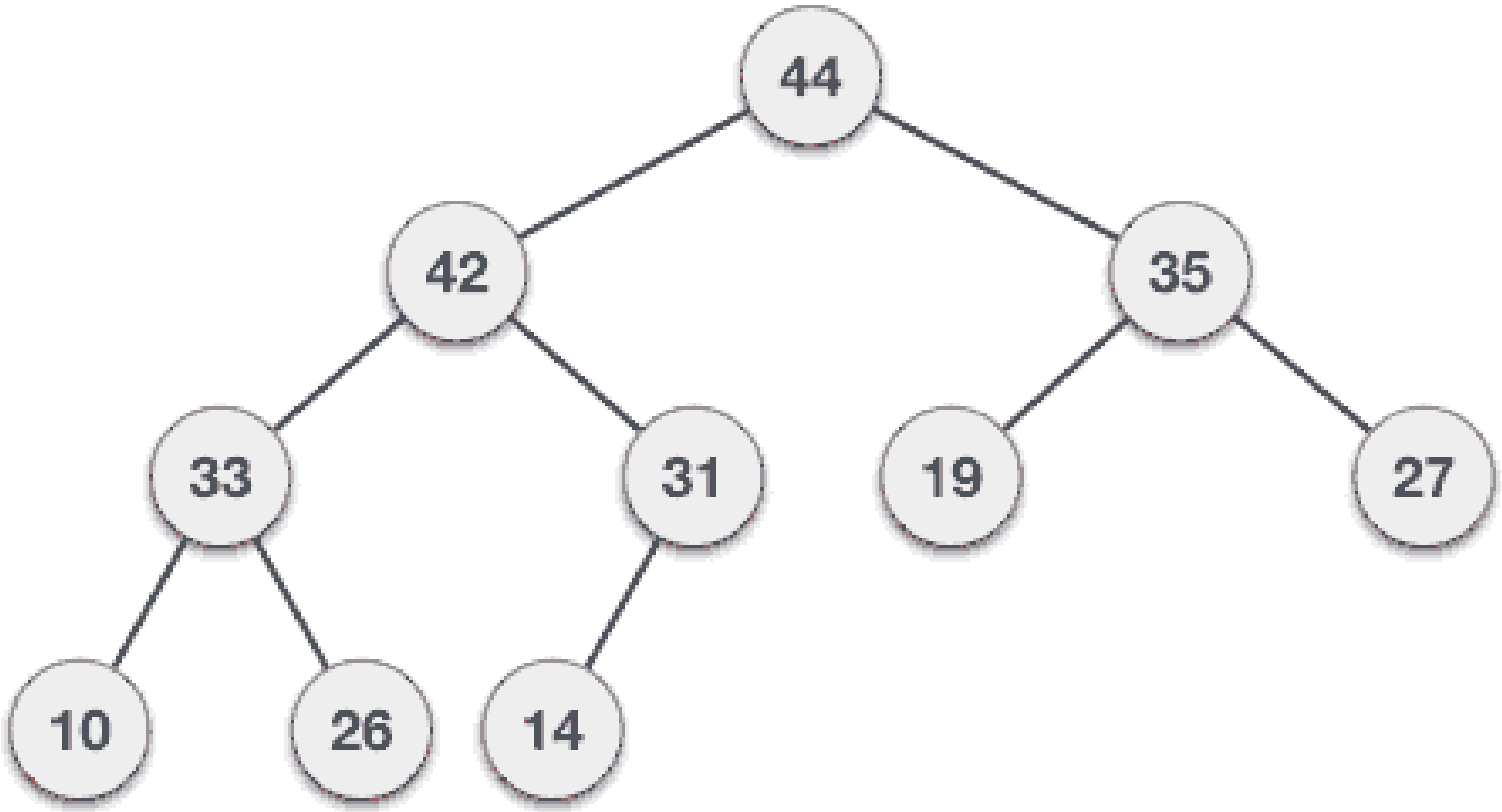
Step 2 – ย้ายค่าท้ายสุดของข้อมูล (Element) มาไว้ที่ Root node

Step 3 – เปรียบเทียบค่าของ ตัวลูกโหนด (Child node) กับ โหนดพ่อแม่ (Parent node)

Step 4 – ถ้าค่าของ โหนดพ่อแม่ มีค่าน้อยกว่า ให้ทำการ ย้ายค่า (Swap)

Step 5 – ทำ Step 3 และ Step 4 ซ้ำจนกว่าจะไม่มีการย้ายค่า

Max – Heap ในการลบค่า



บทที่ 12 Heap และ Graph

บทเรียนย่อย

12.1 Heap Concept

12.2 Heap Implementation

12.3 Graph Concept

12.4 Graphs Notations

12.5 Graph Traversals

12.6 Shortest Path Algorithm

12.3 Graph Concept

กราฟ Graph

- แนวคิดเรื่องกราฟเกิดขึ้นในปี ค.ศ. 1736 ในเมือง Königsberg
- Leonhard Euler เป็นคนแรกที่ได้นำเสนอข้อมูลในรูปแบบกราฟ
- เป็นโครงสร้างโครงสร้างชนิดไม่เชิงเส้น (Non-linear)
- มีเชื่อมกันของ โหนด (Node)
- มีข้อมูลที่มีความสัมพันธ์ ระหว่าง โหนด

บทที่ 12 Heap และ Graph

บทเรียนย่อย

12.1 Heap Concept

12.2 Heap Implementation

12.3 Graph Concept

12.4 Graphs Notations

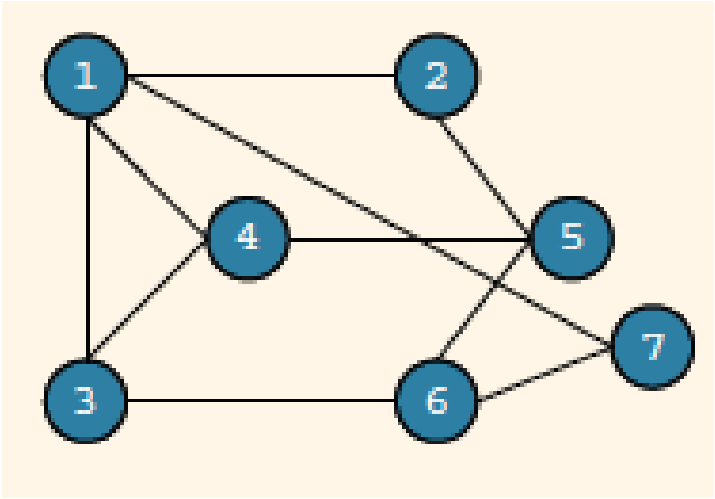
12.5 Graph Traversals

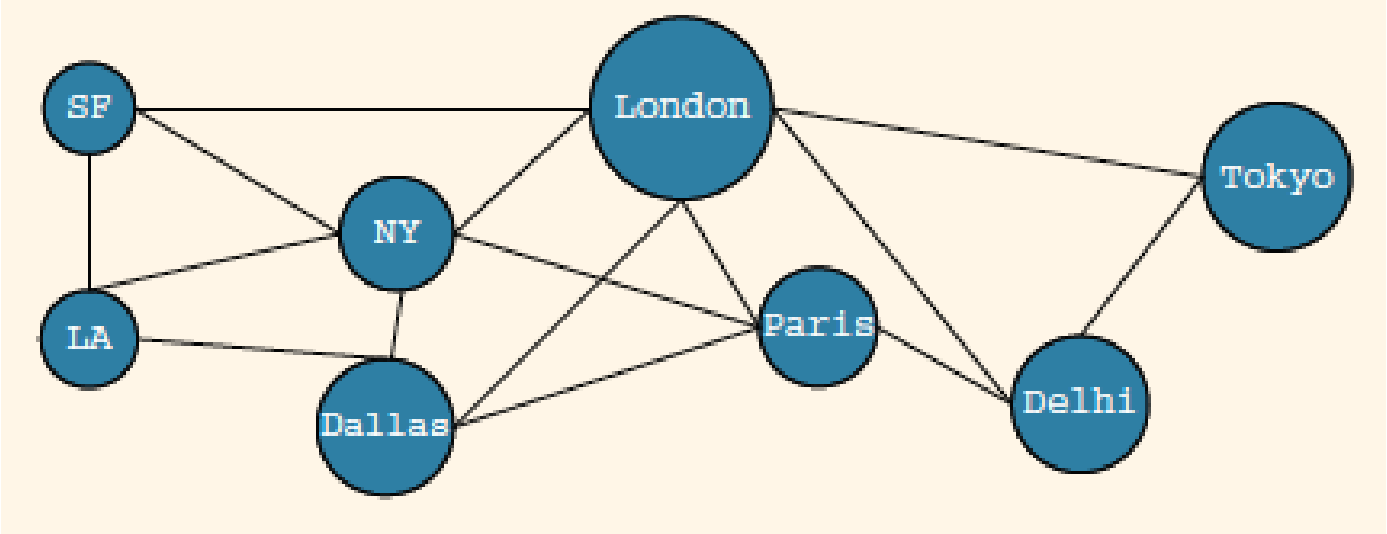
12.6 Shortest Path Algorithm

12.4 Graphs Notations

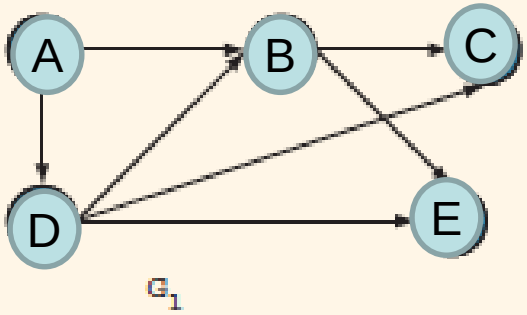
1. Vertex หมายถึง โหนด
2. Edge / Arc หมายถึง เส้นเชื่อมต่อระหว่าง Vertex
3. Degree หมายถึง จำนวนเส้นเข้าและออก ของโหนดแต่ละโหนด
4. Adjacency Node หมายถึง โหนดที่มีการเชื่อมโยงกัน
5. Directional graph คือ เส้นเชื่อมที่มีทิศทาง
6. Non-directional graph คือ เส้นเชื่อมที่ไม่มีทิศทาง
7. Weighted Graphs กราฟที่มีน้ำหนัก คือกราฟที่มีเส้นเชื่อมที่มีค่าบ่งบอกถึงความหมายอย่างใดอย่างหนึ่ง เช่น ระยะทาง ความเร็ว เป็นต้น
8. Unweighted Graphs กราฟที่ไม่มีน้ำหนัก คือกราฟที่เส้นเชื่อมแต่ละเส้นมีน้ำหนักเป็น 1 ทุกเส้น

ตัวอย่างกราฟที่เส้นเชื่อมไม่มีทิศทาง

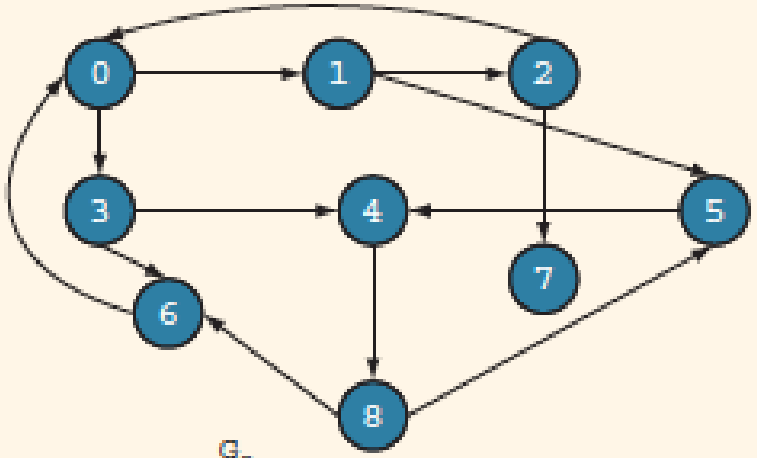




ตัวอย่างกราฟที่เส้นเชื่อมมีทิศทาง

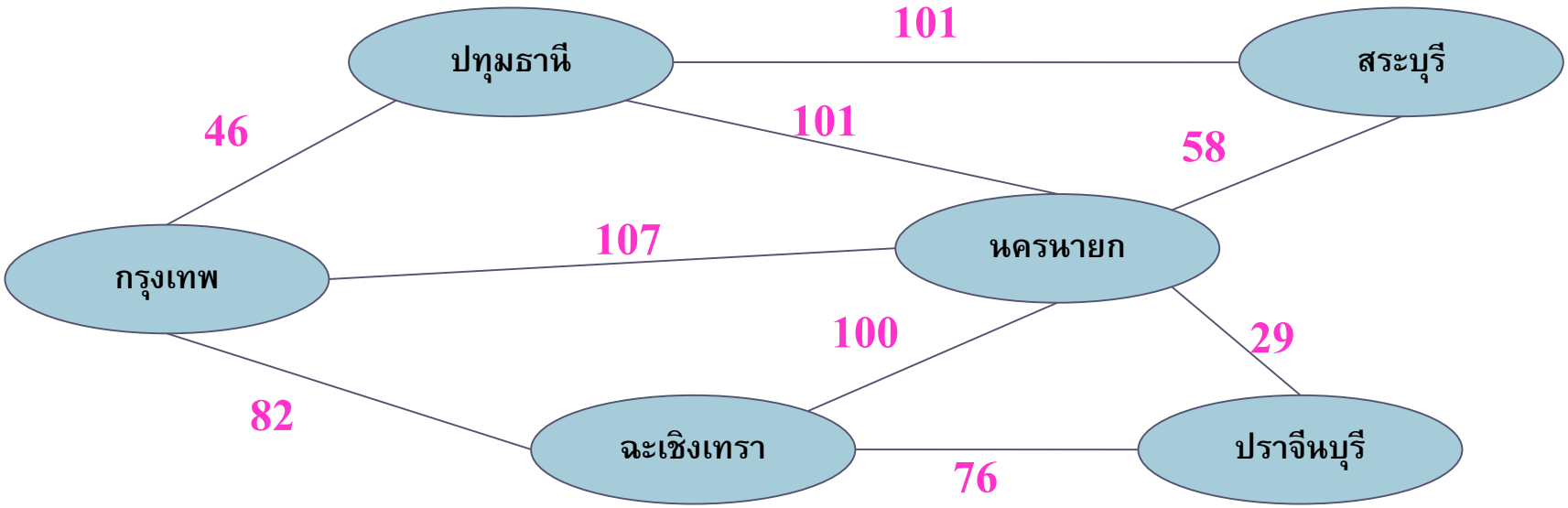


G_1



G_2

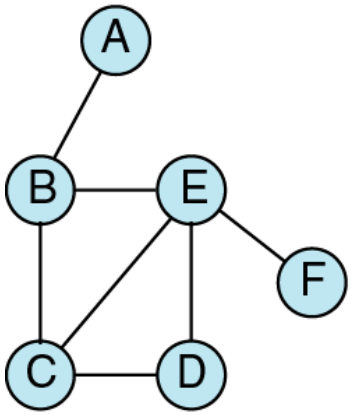
ตัวอย่างกราฟที่มีน้ำหนัก



12.4 Graph Representation

1. The adjacency matrices ใช้รูปแบบอาร์เรย์ในการเก็บข้อมูล มีโครงสร้างที่ประกอบไปด้วยโหนดและเส้นเชื่อมต่อที่บอกถึงเส้นทางของการเดินทาง หรือความสัมพันธ์ในทิศทางซึ่งสามารถนำมาแทนความสัมพันธ์นั้นด้วยการกำหนดเมตริกซ์ n x n
2. The adjacency list ใช้ลิงค์ลิสต์เก็บข้อมูล

ตัวอย่าง Adjacency matrices กราฟที่เส้นเชื่อมไม่มีทิศทาง



A
B
C
D
E
F

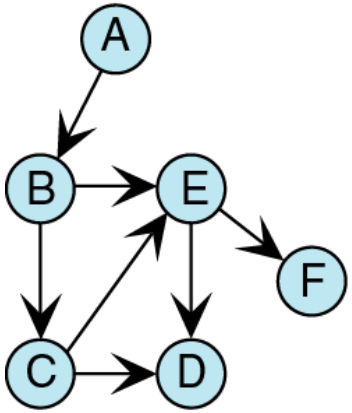
Vertex vector

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

Adjacency matrix

(a) Adjacency matrix for non-directed graph

ตัวอย่าง Adjacency matrices กราฟที่เส้นเชื่อมมีทิศทาง



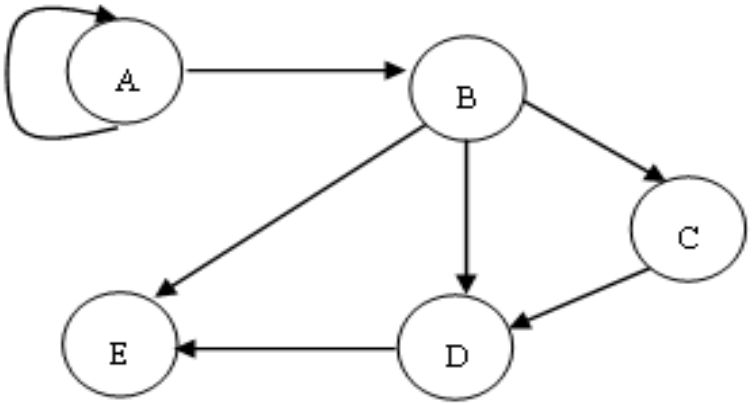
A
B
C
D
E
F

Vertex vector

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

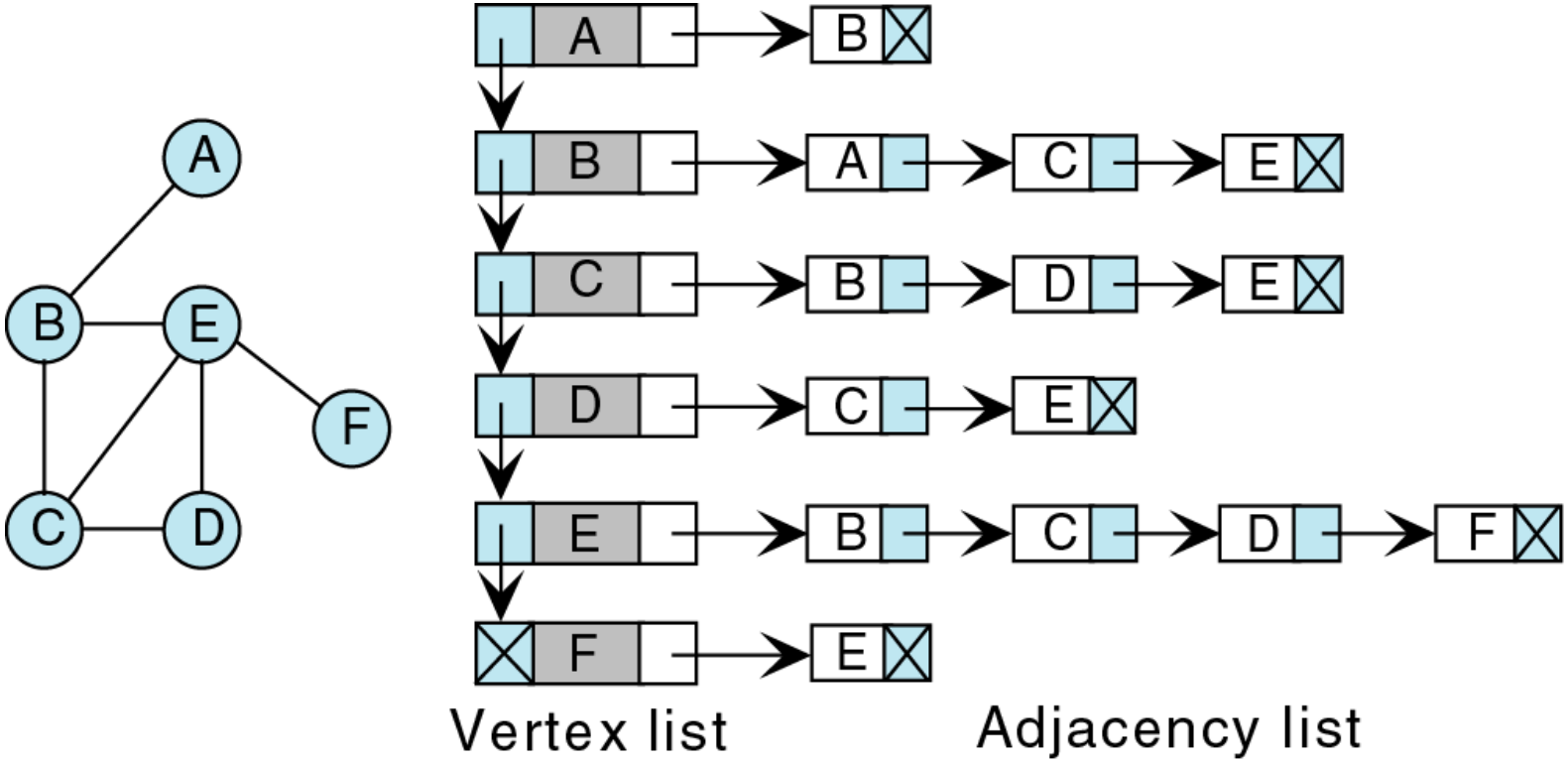
Adjacency matrix

(a) Adjacency matrix for directed graph



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

ตัวอย่างกราฟที่เส้นเชื่อมมีทิศทาง



Example วาดกราฟจาก Adjacency matrices แบบ Directed graphs

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

บทที่ 12 Heap และ Graph

บทเรียนย่อย

12.1 Heap Concept

12.2 Heap Implementation

12.3 Graph Concept

12.4 Graphs Notations

12.5 Graph Traversals

12.6 Shortest Path Algorithm

12.5 Graph Traversals

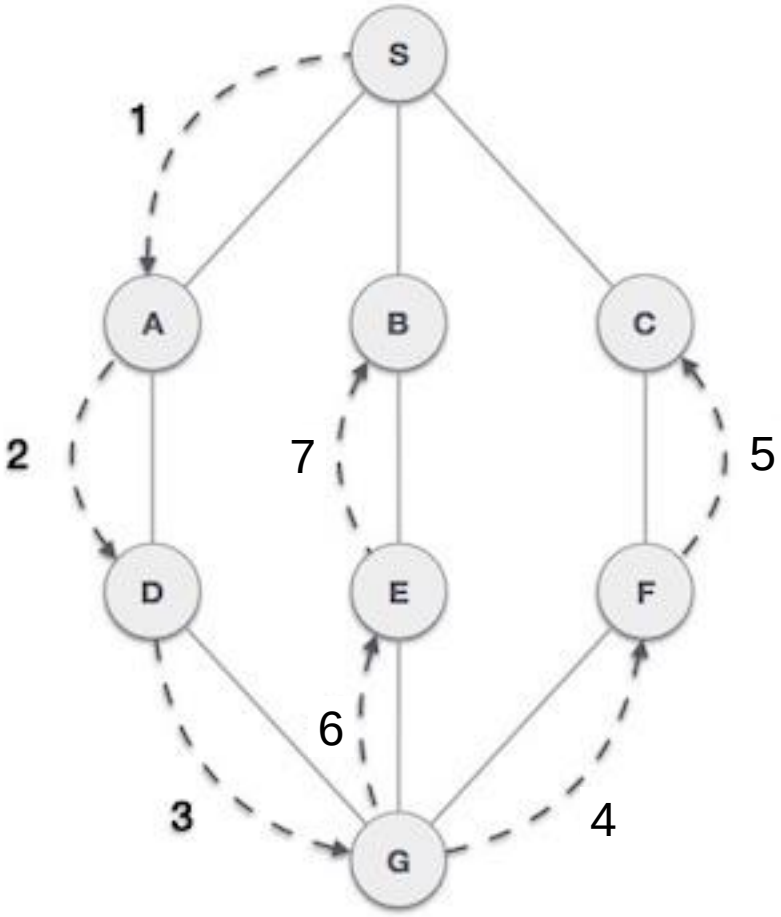
- Depth First Traversal
- Breadth First Traversal

Depth First Traversal

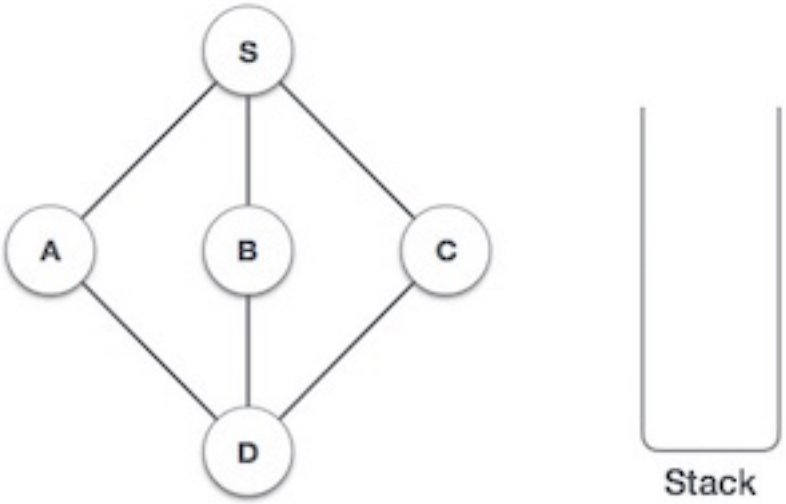
Depth First Traversal เป็นลักษณะการท่องเข้าไปยังโหนดเริ่มต้น แล้วให้โหนดใกล้เคียงเป็นโหนดเริ่มต้น เข้าเยี่ยมโหนด ทำต่อไปจนกระทั่งไม่มีโหนดใกล้เคียงจึงย้อนกลับมายังโหนดก่อนหน้า และเข้าเยี่ยมโหนดอีกด้านด้วยรูปแบบเดียวกันจนครบ เทียบได้กับการท่องเข้าไปในทรีแบบพรีออเดอร์

1. Push vertex
2. Pop vertex และประมวลผล
3. Push adjacent ทั้งหมดของ Vertex ในข้อ 2
4. ทำซ้ำข้อ 2-3 จนกว่าจะครบทุก Vertex และ Stack ว่าง

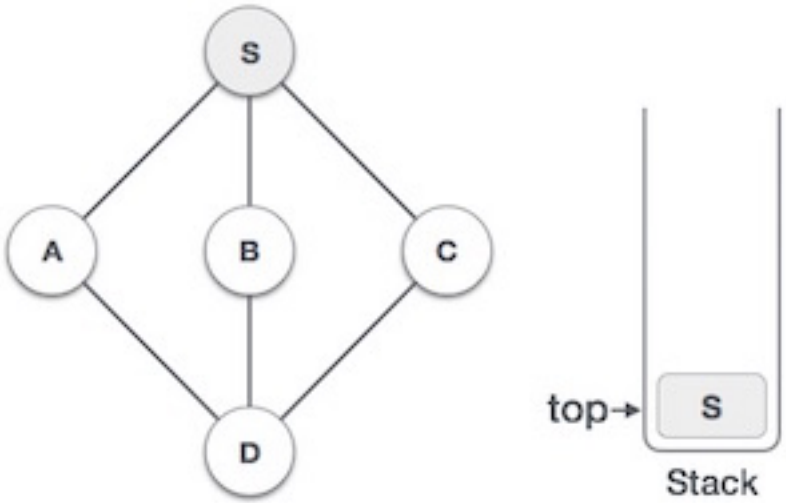
Depth First Traversal



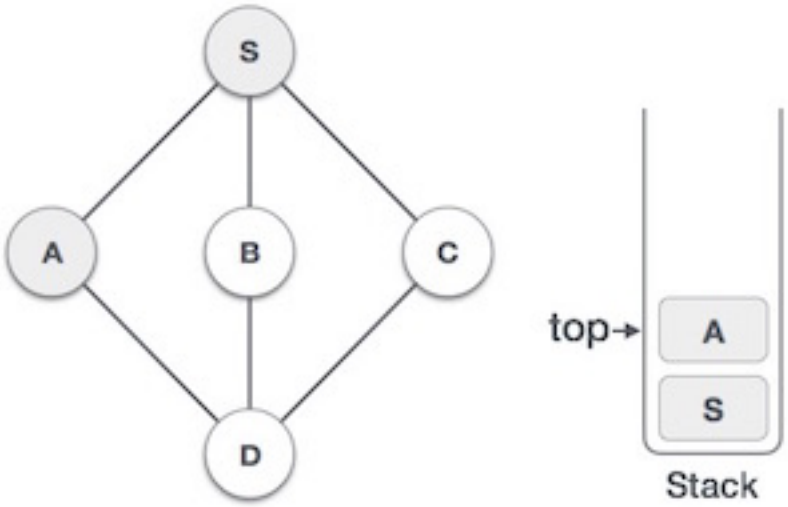
Depth First Traversal



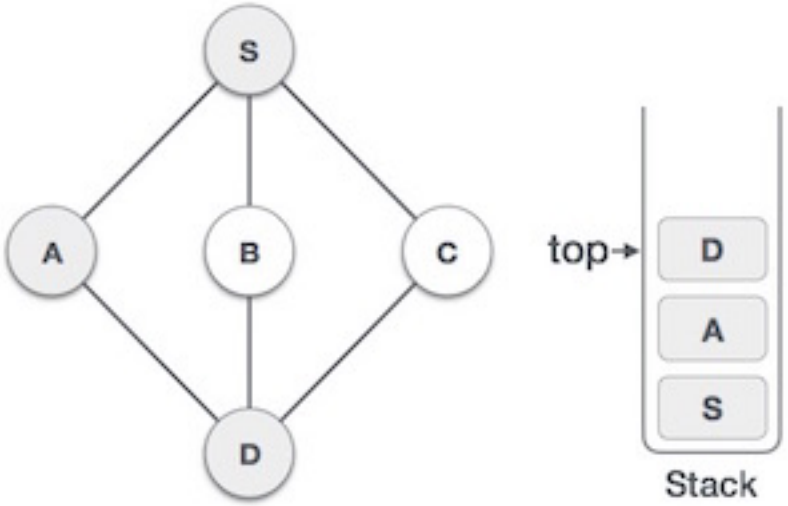
Depth First Traversal



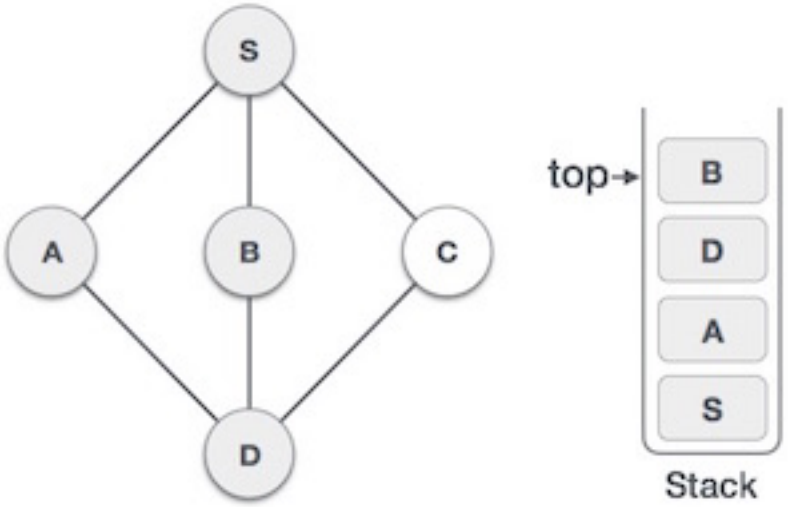
Depth First Traversal



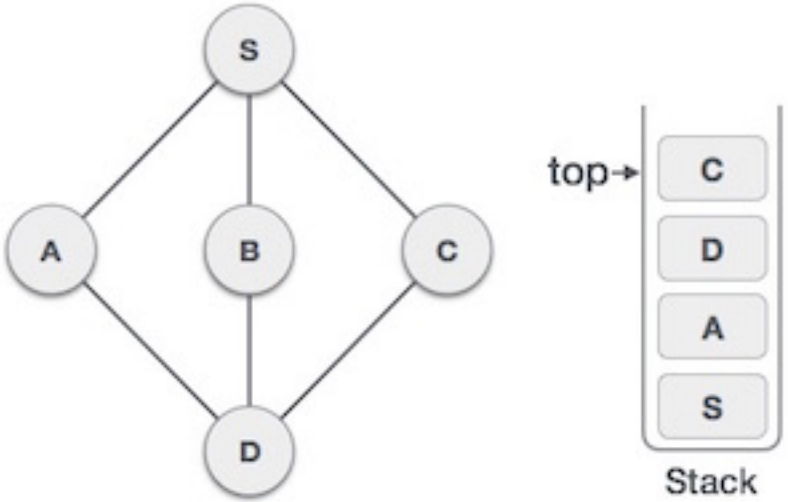
Depth First Traversal



Depth First Traversal



Depth First Traversal



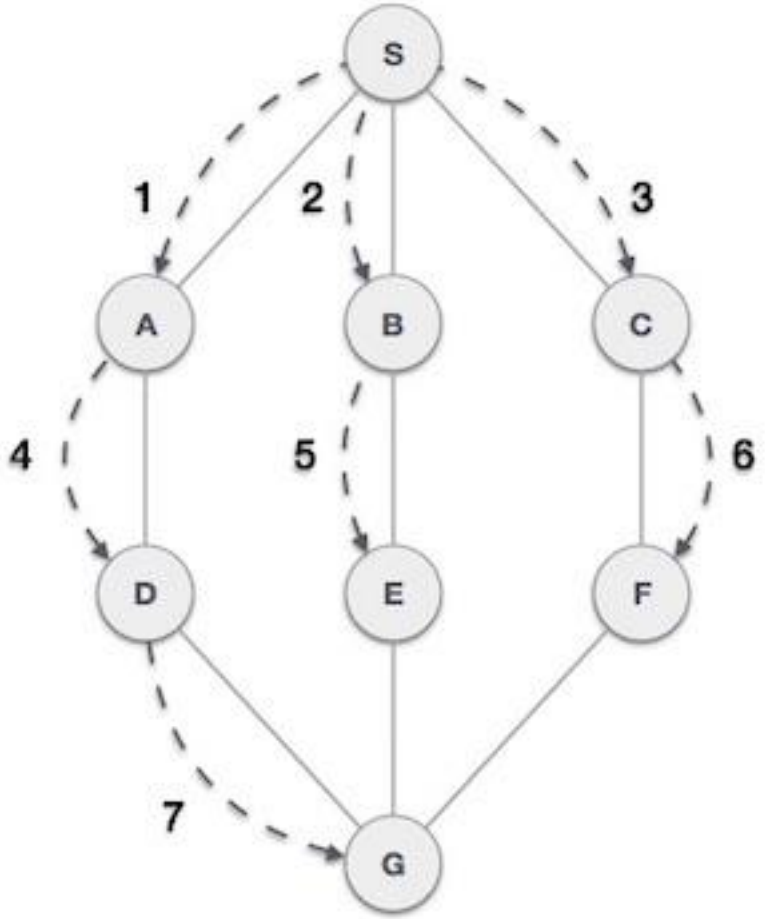
ผลลัพธ์ S A D C B

Breadth First Traversal

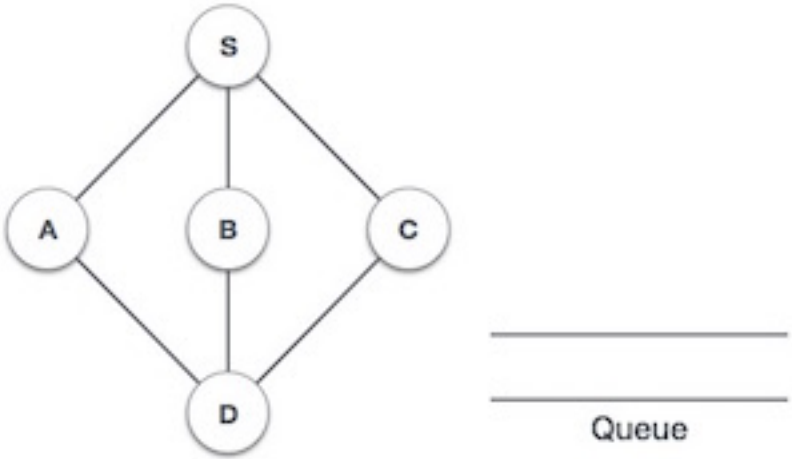
Breadth First Traversal เป็นการท่องเข้าไปในกราฟโดยเข้าเยี่ยมโหนดตัวแรก และดำเนินการ หากมีโหนดใกล้เคียงจะดำเนินการกับโหนดที่อยู่ด้านซ้ายก่อน

1. Enqueue vertex
2. Dequeue vertex และประมวลผล
3. Enqueue adjacent ทั้งหมดของ Vertex ในข้อ 2
4. ทำซ้ำข้อ 2-3 จนกว่าจะครบทุก Vertex และ Queue ว่าง

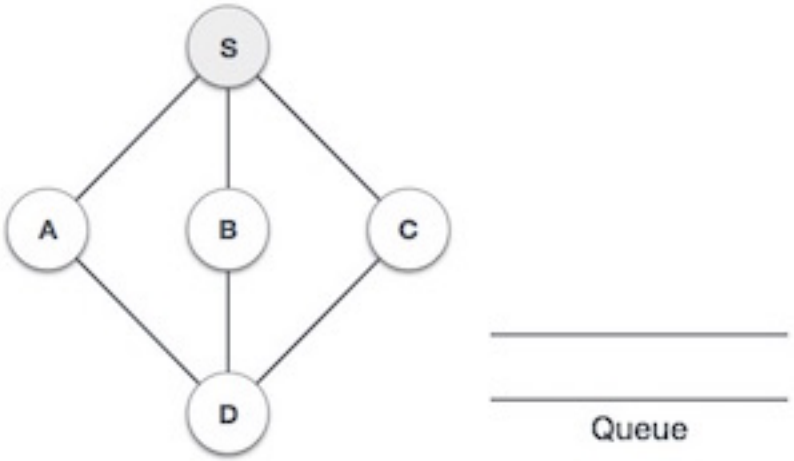
Breadth First Traversal



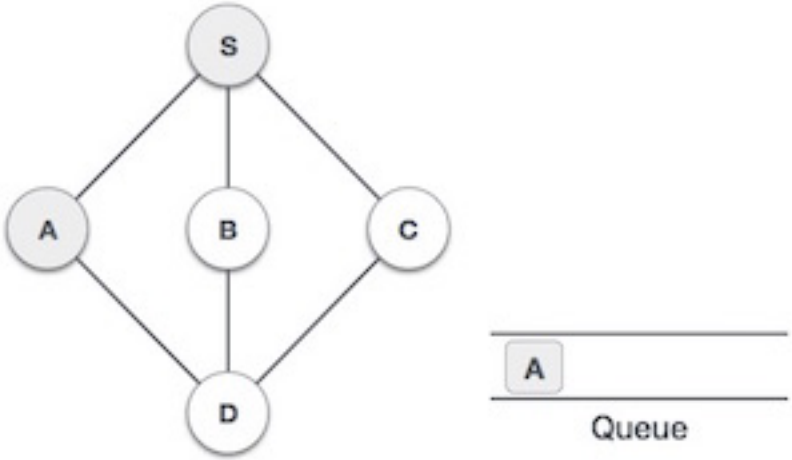
Breadth First Traversal



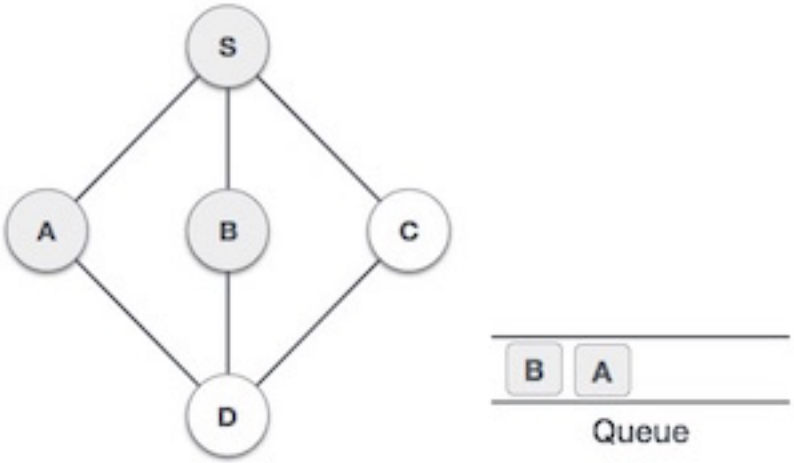
Breadth First Traversal



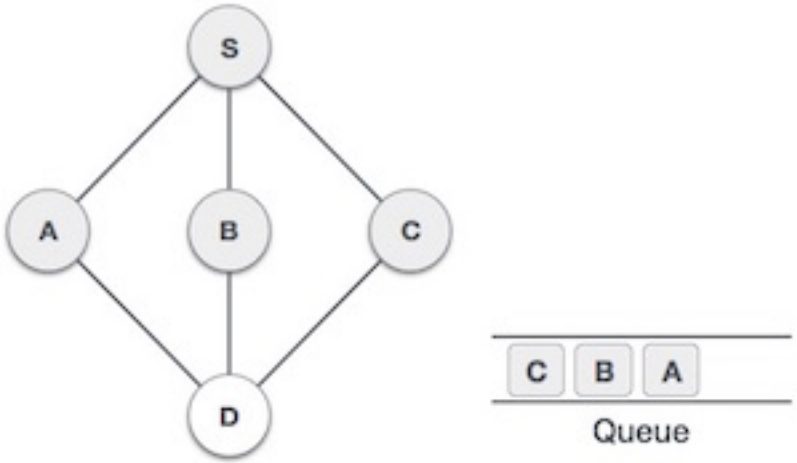
Breadth First Traversal



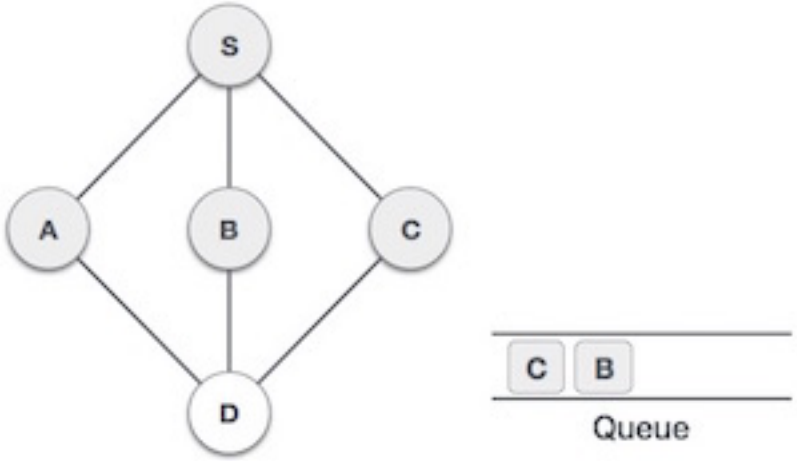
Breadth First Traversal



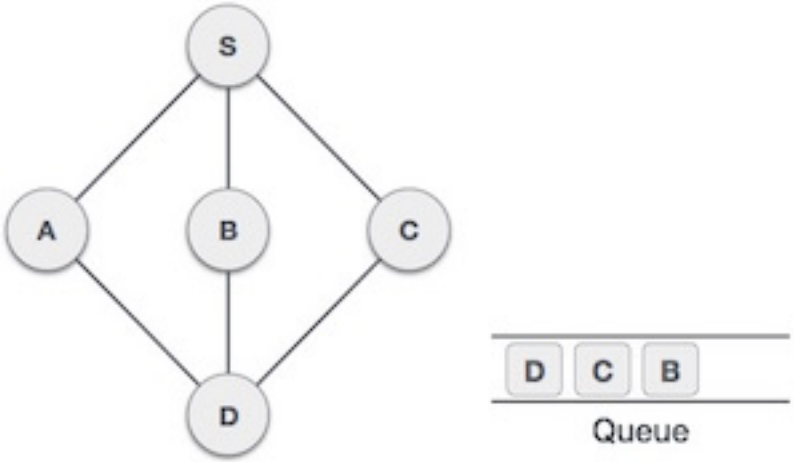
Breadth First Traversal



Breadth First Traversal

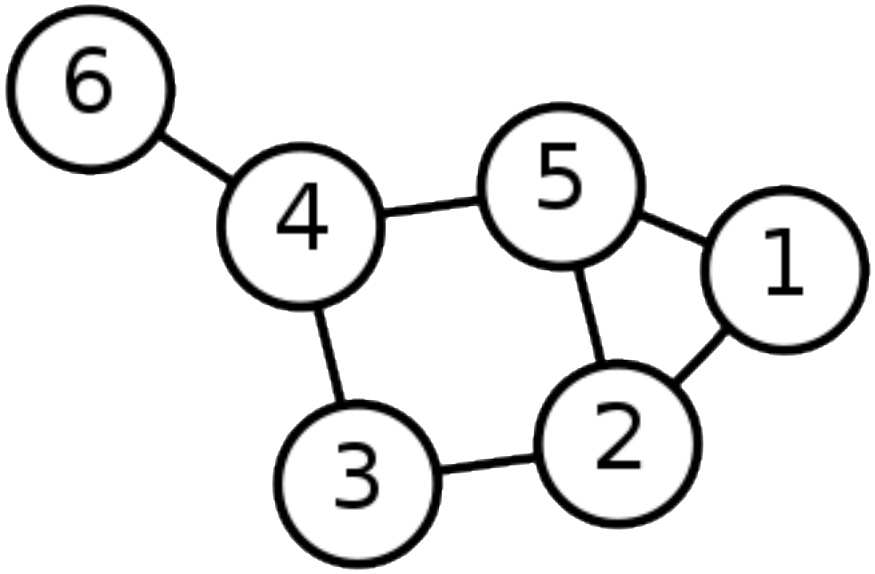


Breadth First Traversal



ผลลัพธ์ S A B C D

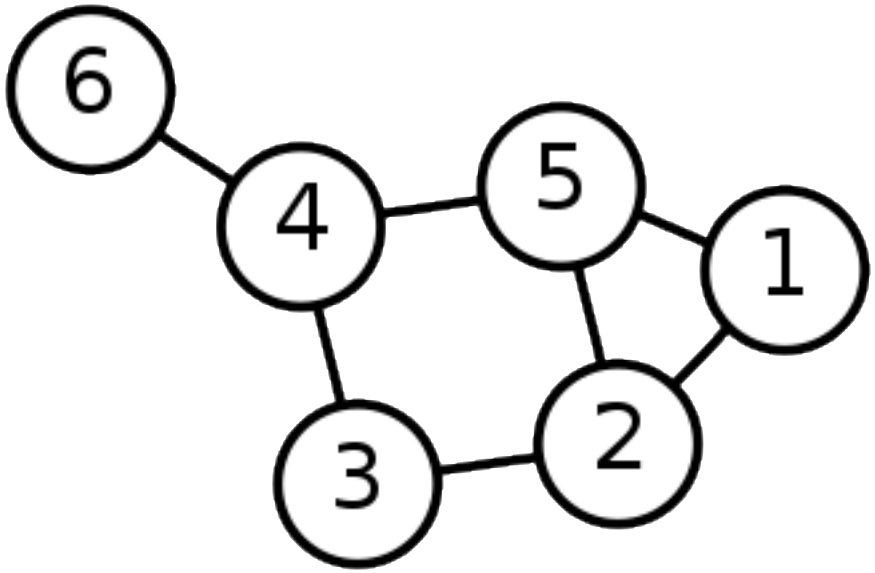
Example



ผลลัพธ์แบบ Depth First Traversal :

ผลลัพธ์แบบ Breadth First Traversal :

Example



ผลลัพธ์แบบ Depth First Traversal : 6 4 3 2 1 5

ผลลัพธ์แบบ Breadth First Traversal : 6 4 3 5 2 1

บทที่ 12 Heap และ Graph

บทเรียนย่อย

12.1 Heap Concept

12.2 Heap Implementation

12.3 Graph Concept

12.4 Graphs Notations

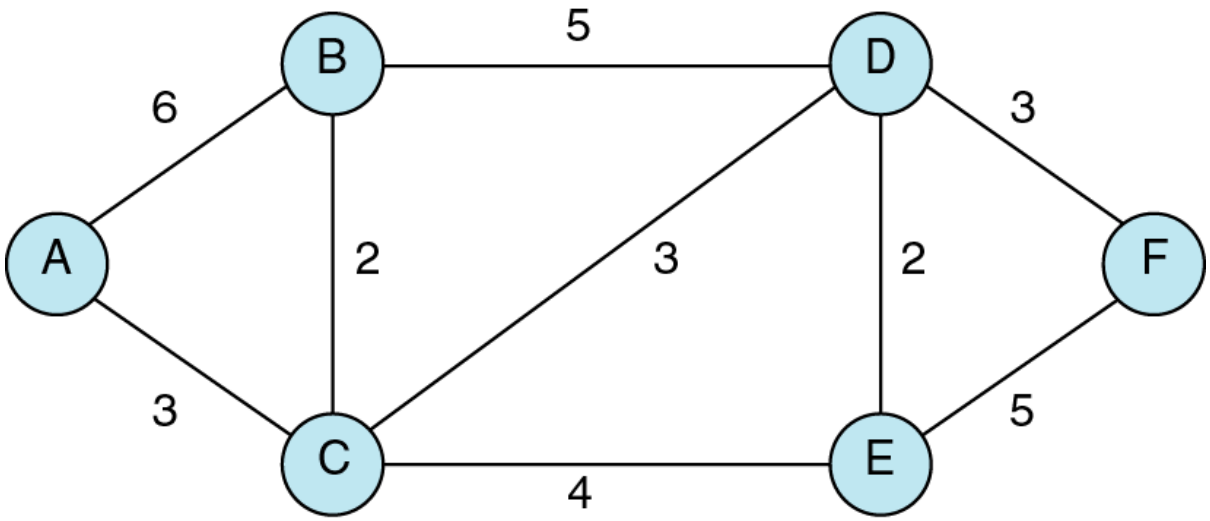
12.5 Graph Traversals

12.6 Shortest Path Algorithm

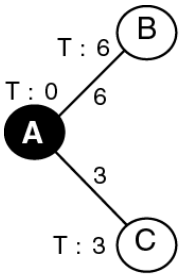
12.6 Shortest Path Algorithm

Shortest Path หมายถึง เส้นทางที่สั้นที่สุดระหว่าง 2 Vertex โดยเป็นการหาเส้นทางการส่งข้อมูลจากต้นทางไปปลายทาง โดยให้มีระยะทางสั้นที่สุด

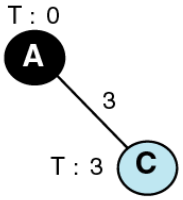
Shortest Path



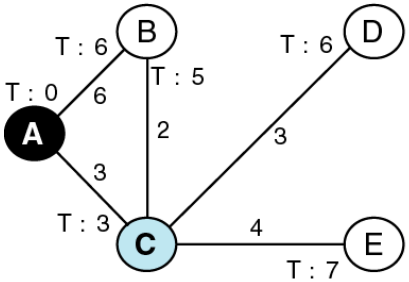
Shortest Path



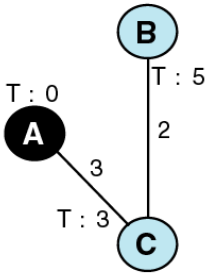
(a1) Possible paths from A1



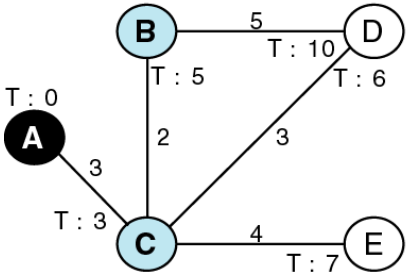
(a2) Tree after insertion of C



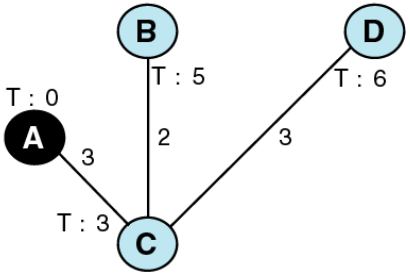
(b1) Possible paths from A and C



(b2) Tree after insertion of B

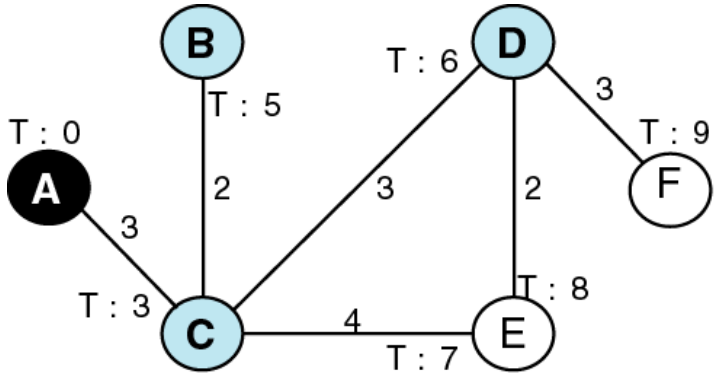


(c1) Possible paths from B and C

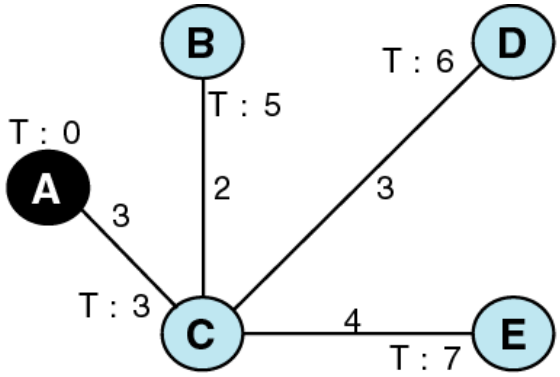


(c2) Tree after insertion of D

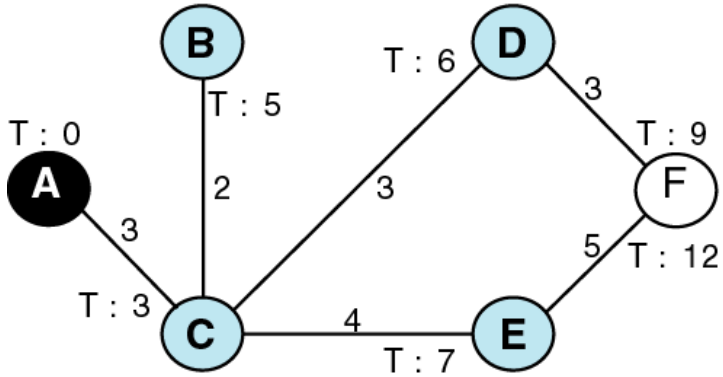
Shortest Path



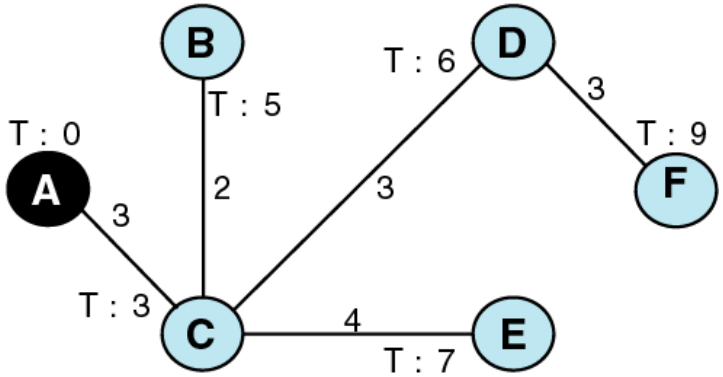
(d1) Possible paths from C and D



(d2) Tree after insertion of E



(e1) Possible paths from D and E

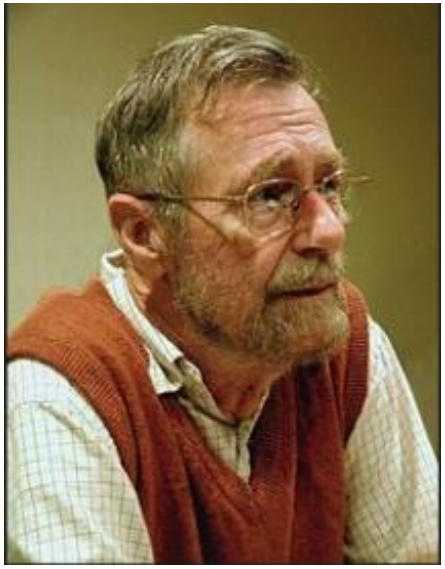


(e2) Tree after insertion of F

T : n Total path length from A to node

Dijkstra's algorithm

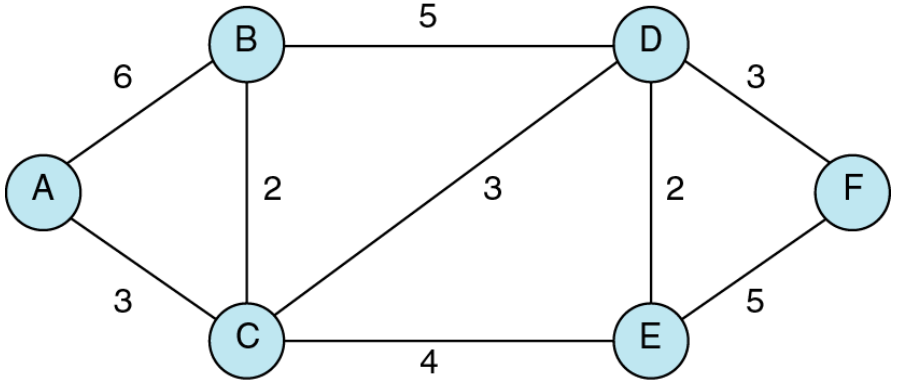
Dijkstra’s Algorithm ถูกคิดค้นขึ้นโดยนักวิทยาการคอมพิวเตอร์ชาวดัตช์นามว่า แอ็ดส์เคอร์ ไดก์สตรา (Edsger Dijkstra) ในปี 1959 เพื่อแก้ไขปัญหาวิถีสั้นสุดจากจุดหนึ่งใด ๆ สำหรับกราฟที่มีความยาวของเส้นเชื่อมไม่เป็นลบ สำหรับขั้นตอนวิธีนี้จะหาระยะทางสั้นที่สุดจากจุดหนึ่งไปยังจุดใด ๆ ในกราฟโดยจะหาเส้นทางที่สั้นที่สุดไปทีละจุดยอดเรื่อย ๆ จนครบตามที่ต้องการ



ขั้นตอนวิธี Dijkstra's algorithm

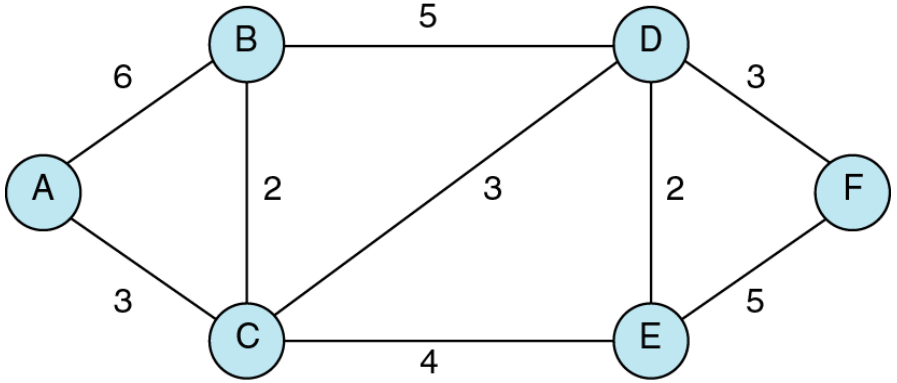
1. กำหนดให้ทุกโหนดมีค่าระยะทางตามเส้นเชื่อม โดยให้โหนดเริ่มต้นมีค่าเป็นศูนย์ และโหนดอื่นมีค่าเป็นอนันต์
2. ทำเครื่องหมายทุกโหนดยกเว้นปมเริ่มต้นว่ายังไม่ไปเยือน (unvisited)
3. จากปมปัจจุบัน พิจารณาปมข้างเคียงตามเส้นเชื่อมทุกโหนดที่ยังไม่ไปเยือน และคำนวณระยะทางต่อเนื่องของเส้นเชื่อม
4. เมื่อพิจารณาโหนดข้างเคียงจากโหนดปัจจุบันครบทุกโหนดแล้ว ทำเครื่องหมายโหนดปัจจุบันว่าไปเยือนแล้ว
5. โหนดปัจจุบันตัวถัดไปที่ถูกเลือกจะเป็นโหนดที่มีค่าระยะทางน้อยสุดในเซต
6. ถ้าเซตของโหนดที่ยังไม่ไปเยือนว่างแล้วให้หยุดการทำงาน ถ้าไม่ทำขั้นตอนที่ 3

ขั้นตอนวิธี Dijkstra's algorithm



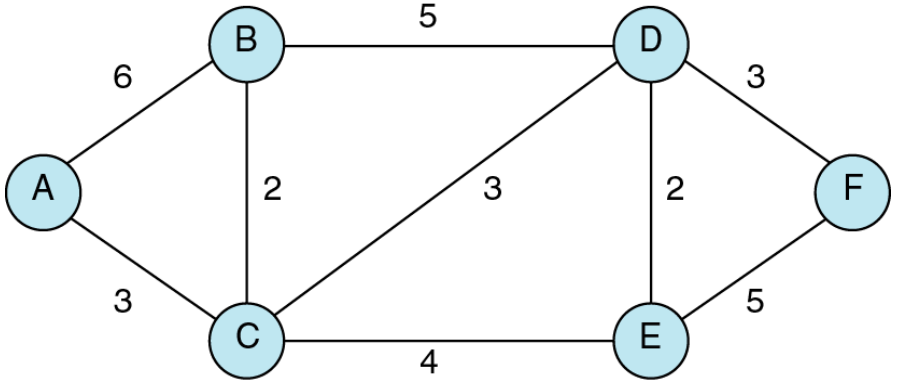
Vertex	A	B	C	D	E	F
A	0_A	∞	∞	∞	∞	∞

ขั้นตอนวิธี Dijkstra's algorithm



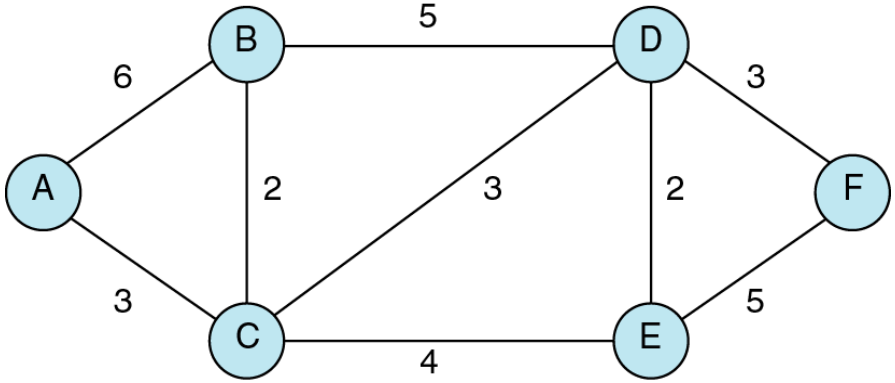
Vertex	A	B	C	D	E	F
A	0_A	6_A	3_A	∞	∞	∞

ขั้นตอนวิธี Dijkstra's algorithm



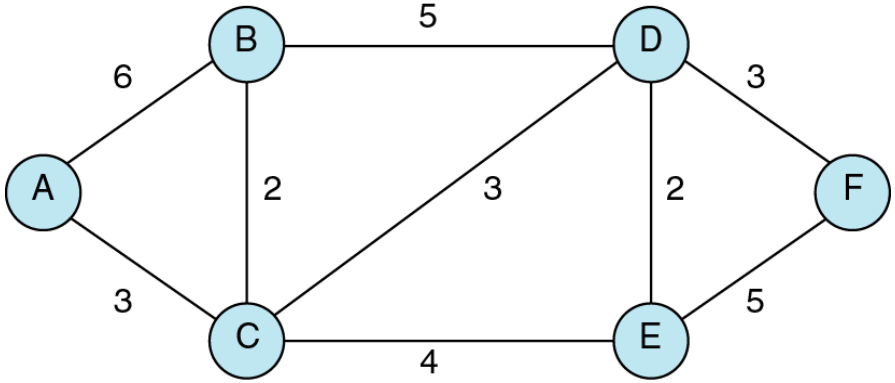
Vertex	A	B	C	D	E	F
A	0_A	6_A	3_A	∞	∞	∞
C		5_C	3_A	6_C	7_C	∞

ขั้นตอนวิธี Dijkstra's algorithm



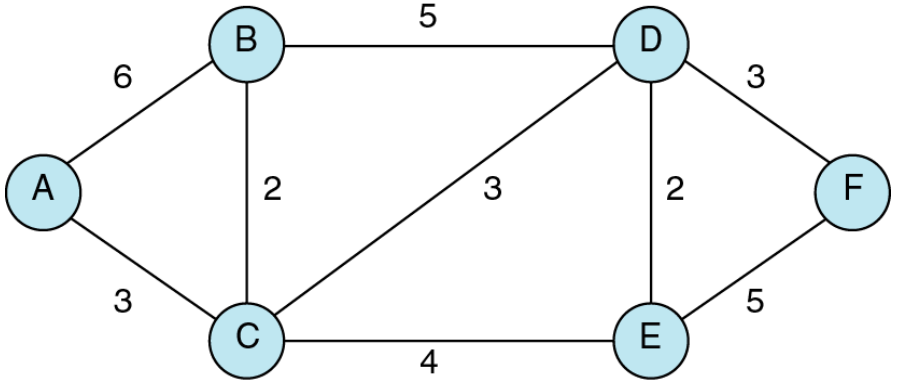
Vertex	A	B	C	D	E	F
A	0_A	6_A	3_A	∞	∞	∞
C		5_C	3_A	6_C	7_C	∞
B		5_C		10_B	∞	∞

ขั้นตอนวิธี Dijkstra's algorithm



Vertex	A	B	C	D	E	F
A	0_A	6_A	3_A	∞	∞	∞
C		5_C	3_A	6_C	7_C	∞
B		5_C		10_B	∞	∞
D				6_C	8_D	9_D

ขั้นตอนวิธี Dijkstra's algorithm



Vertex	A	B	C	D	E	F
A	0_A	6_A	3_A	∞	∞	∞
C		5_C	3_A	6_C	7_C	∞
B		5_C		10_B	∞	∞
D				6_C	8_D	9_D
E					7_C	12_E

Example