Lecture 12

เอกสารประกอบการบรรยาย
Data Structure

Summary B4 Final

Graph

เนื้อหา

- โครงสร้างข้อมูลแบบทรี
- โครงสร้างข้อมูลแบบกราฟ
- การเรียงลำดับข้อมูล
- การค้นหาข้อมูล

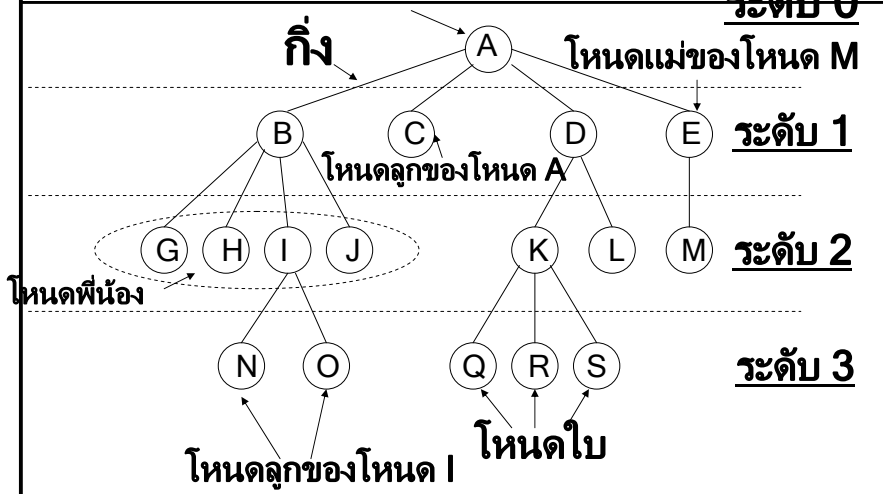
Tree (Cont.)

ทรี (Tree) เป็นโครงสร้างข้อมูลที่ความสัมพันธ์ระหว่าง โหนดจะมีความสัมพันธ์ลดหลั่นกันเป็นลำดับชั้น (Hierarchical Relationship)

ได้มีการนำรูปแบบทรีไปประยุกต์ใช้ในงานต่าง ๆ อย่างแพร่หลาย ส่วนมากจะใช้สำหรับแสดงความสัมพันธ์ระหว่างข้อมูล

เช่น แผนผังองค์ประกอบของหน่วยงานต่าง ๆ โครงสร้างสารบัญหนังสือ เป็นต้น

Tree (Cont.)



Tree (Cont.)

แต่ละโหนดจะมีความสัมพันธ์กับโหนดในระดับที่ต่ำลงมา หนึ่งระดับได้หลาย ๆ โหนด เรียกโหนดดังกล่าวว่า *โหนดแม่* (Parent or Mother Node)

โหนดที่อยู่ต่ำกว่าโหนดแม่อยู่หนึ่งระดับ เรียกว่า *โหนดลูก* (Child or Son Node)

โหนดที่อยู่ในระดับสูงสุดและไม่มีโหนดแม่ เรียกว่า *โหนดราก* (Root Node)

Tree (Cont.)

โหนดที่มีโหนดแม่เป็นโหนดเดียวกัน เรียกว่า *โหนดพี่น้อง* (Siblings)

โหนดที่ไม่มีโหนดลูก เรียกว่า *โหนดใบ* (Leave Node)

เส้นเชื่อมแสดงความสัมพันธ์ระหว่างโหนดสองโหนด เรียกว่า *กิ่ง* (Branch)

Internal Node คือ B I D E และ K

Ancestor ของ D คือ A

Descendent ของ B คือ G H I J N O

Tree (Cont.)

การท่องไปในไบนารีทรี

ปฏิบัติการที่สำคัญในไบนารีทรี คือ การท่องไปในไบนารีทรี (Traversing Binary Tree) เพื่อเข้าไปเยือนทุก ๆ โหนดในทรี ซึ่งวิธีการท่องเข้าไปต้องเป็นไปอย่างมีระบบแบบแผน สามารถเยือนโหนดทุก ๆ โหนด ๆ ละหนึ่งครั้ง วิธีการท่องไปนั้นมีด้วยกันหลายแบบแล้วแต่ว่าต้องการลำดับขั้นตอนการเยือนอย่างไร โหนดที่ถูกเยือน

อาจเป็นโหนดแม่ (แทนด้วย N)

ทรีย่อยทางซ้าย (แทนด้วย L)

หรือทรีย่อยทางขวา (แทนด้วย R)

Tree (Cont.)

มีวิธีการท่องเข้าไปในทรี 6 วิธี คือ NLR LNR LRN NRL RNL และ RLN แต่วิธีการท่องเข้าไปไบนารีทรีที่นิยมใช้กันมากเป็นการท่องจากซ้ายไปขวา 3 แบบแรกเท่านั้นคือ NLR LNR และ LRN ซึ่งลักษณะการนิยามเป็นนิยามแบบ รีเคอร์ซีฟ (Recursive) ซึ่งขั้นตอนการท่องไปในแต่ละแบบมีดังนี้

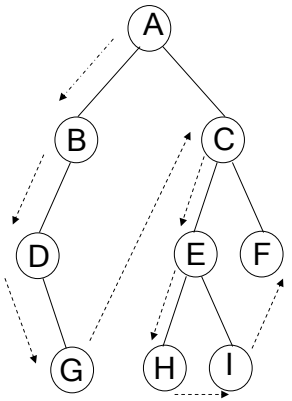
Tree (Cont.)

1. การท่องไปแบบพรีออร์เดอร์ (Preorder Traversal) เป็นการเดินเข้าไปเยือนโหนดต่าง ๆ ในทรีด้วยวิธี NLR มีขั้นตอนการเดินดังต่อไปนี้

(1) เยือนโหนดราก
(2) ท่องไปในทรีย่อยทางซ้ายแบบพรีออร์เดอร์
(3) ท่องไปในทรีย่อยทางขวาแบบพรีออร์เดอร์

Tree (Cont.)

NLR



เส้นทางการท่องในทรีแบบพรีออร์เดอร์ จะได้ ABDGCEHIF

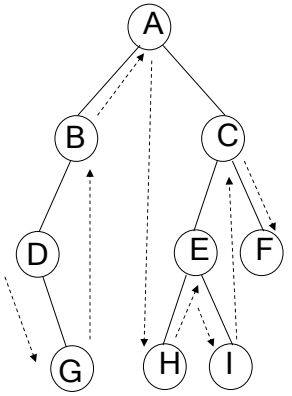
Tree (Cont.)

2.การท่องไปแบบอินออร์เดอร์ (Inorder Traversal) เป็นการเดินเข้าไปเยือนโหนดต่าง ๆ ในทรีด้วยวิธี LNR มีขั้นตอนการเดินดังต่อไปนี้

(1) ท่องไปในทรีย่อยทางซ้ายแบบอินออร์เดอร์
(2) เยือนโหนดราก
(3) ท่องไปในทรีย่อยทางขวาแบบอินออร์เดอร์

Tree (Cont.)

LNR



เส้นทางการท่องในทรีแบบอินออร์เดอร์ จะได้ DGBAHEICF

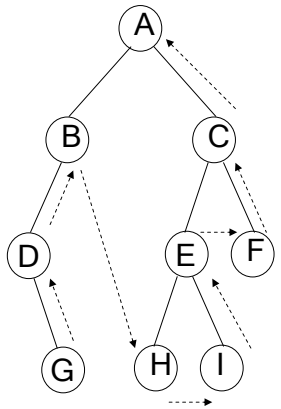
Tree (Cont.)

3. การท่องไปแบบโพสออร์เดอร์ (Postorder Traversal)

เป็นการเดินเข้าไปเยือนโหนดต่าง ๆ ในทรีด้วยวิธี LRN มีขั้นตอนการเดินดังต่อไปนี้
(1) ท่องไปในทรีย่อยทางซ้ายแบบโพสต์ออร์เดอร์
(2) ท่องไปในทรีย่อยทางขวาแบบโพสต์ออร์เดอร์
(3) เยือนโหนดราก

Tree (Cont.)

LRN



เส้นทางการท่องในทรีแบบโพสต์ออร์เดอร์ จะได้ GDBHIEFCA

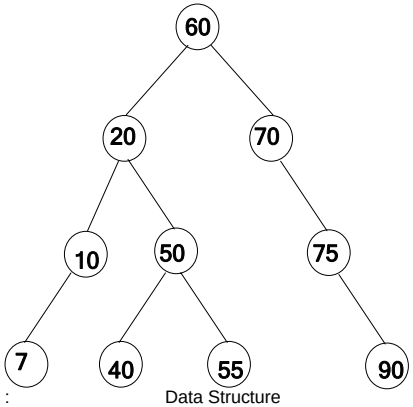
Tree (Cont.)

ไบนารีเซิร์ชทรี

ไบนารีเซิร์ชทรี (Binary Search Tree) เป็นไบนารีทรีที่มีคุณสมบัติที่ว่าทุก ๆ โหนดในทรี ค่าของโหนดรากมีค่ามากกว่าค่าของทุกโหนดในทรีย่อยทางซ้าย และมีค่าน้อยกว่าหรือเท่ากับค่าของทุกโหนดในทรีย่อยทางขวา และในแต่ละทรีย่อยก็มี คุณสมบัติเช่นเดียวกัน

Tree (Cont.)

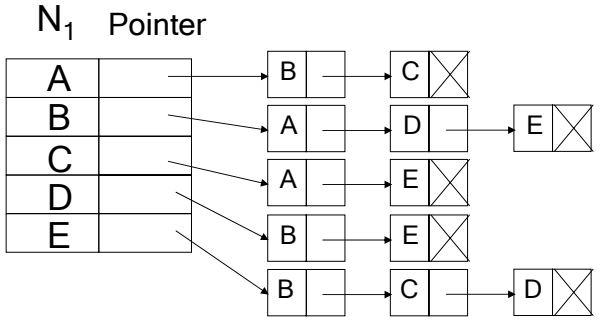
จากตัวเลขจำนวนเต็มต่อไปนี้ เมื่อนำไปสร้างเป็นไบนารีจะทำได้ดังรูป 60 , 20 , 70 , 75, 50 , 40 , 10 , 90 , 7, 55



Graph

กราฟที่มีการเปลี่ยนแปลงตลอดเวลา อาจจะใช้วิธีแอดจาเซนซีลิสต์ (Adjacency List) ซึ่งเป็นวิธีที่คล้ายวิธีจัดเก็บกราฟด้วยการเก็บโหนดและพอยน์เตอร์ แต่ต่างกันตรงที่ จะใช้ ลิงค์ลิสต์แทน เพื่อความสะดวกในการเปลี่ยนแปลงแก้ไข

Graph (Cont.)



การแทนกราฟด้วยวิธีแอดจาเซนซีลิสต์

Graph

การท่องไปในกราฟ

1. การค้นหาแบบกว้าง (Breadth-first Search)
2. การค้นหาแบบลึก (Depth-first Search)

Graph

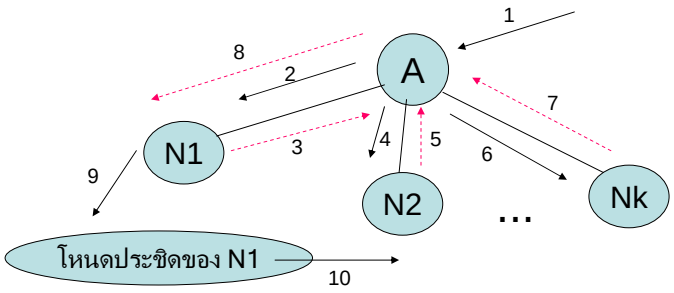
การท่องไปในกราฟ

1. การค้นหาแบบกว้าง (Breadth-first Search)

กำหนดจุดเริ่มต้น ถ้าให้เริ่มต้นที่จุด A การค้นหาจะเริ่มต้นที่โหนดประชิดของ A จนครบทุกจำนวนของโหนดประชิด จากภาพที่ปรากฏต่อไปนี้ โหนด N1 โหนด N2 ไปเรื่อย ๆ จนจบที่โหนด Nk การค้นหาแบบกว้างจะค้นหาต่อที่โหนดประชิดของ N1 ซึ่งเป็นโหนด ประชิดแรกของโหนด A แบบแผนการค้นหา จะเป็นแบบเดียวกับโหนด A หลังจากเสร็จสิ้นการค้นหาจะดำเนินการค้นหาต่อที่ โหนด N2 จนสุดท้ายจบที่ โหนด Nk ในหารค้นหาแบบกว้างจะใช้คิวเก็บลำดับของโหนด ที่ต้องการค้นหาต่อไป

Graph

การค้นหาแบบกว้างโดยเริ่มต้นที่โหนด A



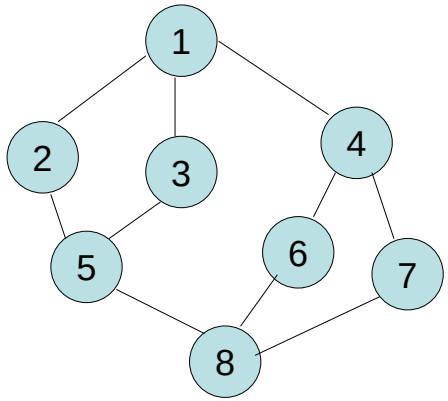
Graph

การท่องไปในกราฟ

1. การค้นหาแบบกว้าง ในกราฟไม่มีทิศทาง รายชื่อโหนดที่พบจากการค้นหาแบบกว้าง มิได้หลายรายการขึ้นกับลำดับการเรียงโหนดประชิดดังตารางต่อไปนี้ แสดงค่าโหนดประชิดของโหนดทุกโหนด ซึ่งสร้างมาจากกราฟ
   ถ้ากำหนดให้เริ่มต้นค้นที่โหนด 1 รายชื่อโหนดที่พบเรียงตามลำดับดังนี้ 2 3 4 5 6 7 และ 8
   ถ้ากำหนดให้เริ่มต้นค้นที่โหนด 6 รายชื่อโหนดที่พบเรียงลำดับดังนี้ 4 8 1 7 5 2 และ 3

Graph

กราฟไม่มีทิศทางและรายชื่อโหนดประชิด



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

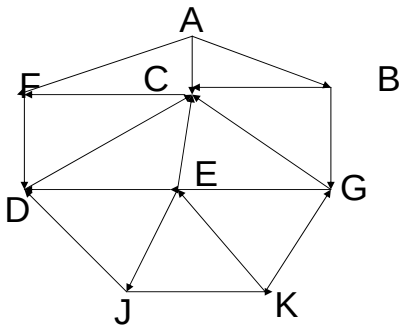
Graph

การท่องไปในกราฟ

2. การค้นหาแบบกว้าง ในกราฟมีทิศทาง การค้นหาโหนดในกราฟทำได้ง่ายขึ้นถ้าใช้คิวเก็บลำดับของโหนดประชิดที่ต้องเยี่ยมต่อไป และใช้ตารางเก็บค่าโหนดประชิดของโหนดทุกโหนดในกราฟ การค้นหาแบบกว้างในกราฟจะพบโหนดตามลำดับดังนี้ A F C B D G E J และ K

Graph

รายการโหนดประชิดของกราฟมีทิศทาง A F C B D G E J K



Adjacency Lists
A: F,C,B
B: G,C
C: F
D: C
E: D,C,J
F: D
G: C,E
J: D,K
K: E,G

Graph

การท่องไปในกราฟ

อัลกอริทึมแบบกว้าง มีดังนี้

1. นำโหนดเริ่มต้นไปเก็บในคิว และปรับค่า front และ rear=1
2. นำสมาชิกออกจากคิว 1 จำนวน และปรับค่า front ใหม่ดังนี้
   front =front+1
3. นำโหนดประชิดทั้งหมด ของโหนดที่ตำแหน่ง front ไปเก็บในคิว โดยโหนดที่จะเข้าคิวนั้นต้องไม่เคยนำไปเก็บในคิวมาก่อน และปรับค่า rear ใหม่ดังนี้
   rear=rear+จำนวนโหนดประชิดที่เก็บในคิว
4. ตรวจสอบคิว ถ้ามีข้อมูล ให้ไปทำข้อ 2.
5. เลิกงาน

Graph

การท่องไปในกราฟ

2. การค้นหาแบบลึก (Depth-first Search)

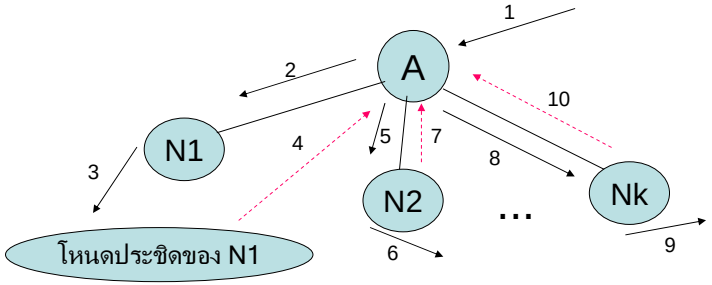
จะต่างตรงที่จุดเริ่มต้นที่จะไปเยี่ยมโหนดในกราฟมีหลายจุด จึงต้องกำหนดจุดเริ่มต้นสำหรับเยี่ยมเป็นจุดแรก การค้นหาแบบลึกใช้หลักการคล้ายแบบลำดับของต้นไม้

จากภาพในตัวอย่างต่อไปจะใช้โหนด A เป็นจุดเริ่มต้น การค้นหาจะเริ่มจากโหนดประชิดค่าแรกของโหนด A คือโหนด N1 แล้วดูว่าโหนด N1 มีโหนดประชิดหรือไม่

ถ้ามีก็ค้นหาต่อไป โดยใช้แบบแผนการค้น เหมือนโหนด A จนครบ แล้วกลับไปค้นหาที่โหนดประชิดตัวที่ 2 ของโหนด A คือ โหนด N2 โดยใช้แบบแผนการค้นเดียวกับโหนด N1 ทำแบบเดียวกันจนครบถึงโหนด Nk

Graph

การค้นหาแบบลึกโดยเริ่มต้นที่โหนด A



Graph

การท่องไปในกราฟ
อัลกอริทึมแบบลึก มีดังนี้

1. push โหนดเริ่มต้นไปเก็บในสแตก และปรับ top ให้ชี้ที่ตำแหน่งโหนดเริ่มต้น
2. pop สมาชิกใหม่ออกจากสแตก 1 จำนวน ปรับค่า top ใหม่ดังนี้
   top=top-1
3. push โหนดประชิดทั้งหมด ของโหนดที่ pop ครั้งสุดท้ายลงสแตก กำหนดโหนดที่จะ push นั้นต้องไม่เคย pushไปเก็บในสแตกมาก่อน และปรับค่า top ใหม่ดังนี้
   top=top+จำนวนโหนดประชิดที่ push ลงในแสตก
4. ตรวจสอบสแตก ถ้ามีข้อมูล ให้ไปทำข้อ 2.
5. เลิกงาน

Graph

กราฟ มีน้ำหนัก หมายถึง กราฟที่ทุกเอดจ์ มีค่าน้ำหนักกำกับ ซึ่งค่าน้ำหนักอาจสื่อถึง ระยะทาง เวลา ค่าใช้จ่าย เป็นต้น นิยมนำไปใช้แก้ปัญหาหลัก ๆ 2 ปัญหา คือ

Graph

1. การสร้างต้นไม้ทอดข้ามน้อยที่สุด (Minimum Spanning Trees :MST)

2. การหาเส้นทางที่สั้นที่สุด (Shortest path)

Graph

1. การสร้างต้นไม้ทอดข้ามน้อยที่สุด (Minimum Spanning Trees :MST)

Prim's Algorithm and Kruskal's Algorithm

Graph

1. การสร้างต้นไม้ทอดข้ามน้อยที่สุด (Minimum Spanning Trees :MST)

Kruskal's Algorithm

1. เรียงลำดับเอดจ์ ตามน้ำหนัก
2. สร้างป่าที่ประกอบด้วยต้นไม้ว่างที่มีแต่โหนด และไม่มีเส้นเชื่อม
3. เลือกเอดจ์ที่มีน้ำหนักน้อยที่สุดและยังไม่เคยถูกเลือกเลย ถ้ามีน้ำหนักซ้ำกันหลายค่าให้สุ่มมา 1 เส้น

Graph

1. การสร้างต้นไม้ทอดข้ามน้อยที่สุด (Minimum Spanning Trees :MST)

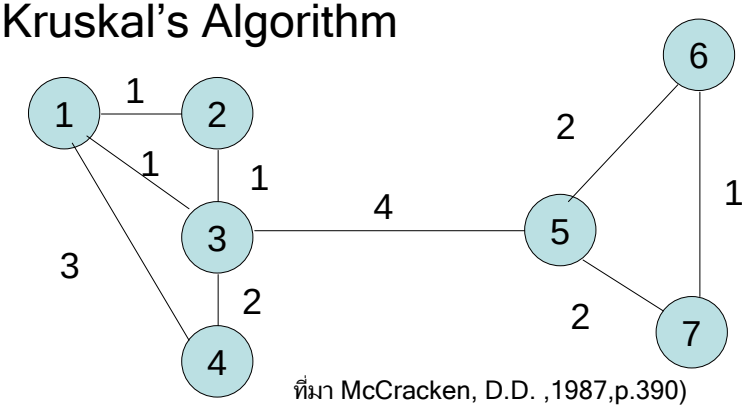
Kruskal's Algorithm

4. พิจารณาเอดจ์ที่จะเลือก ถ้านำมาประกอบในต้นไม้ทอดข้ามน้อยที่สุดแล้วเกิด วงรอบ ให้ตัดทิ้งนอกนั้นให้นำมาประกอบเป็นเอดจ์ในต้นไม้ทอดข้ามน้อยที่สุด
5. ตรวจสอบเอดจ์ที่ต้องอ่านในกราฟ ถ้ายังอ่านไม่หมดให้ไปทำข้อ 3

Graph

Minimum Spanning Trees :MST)

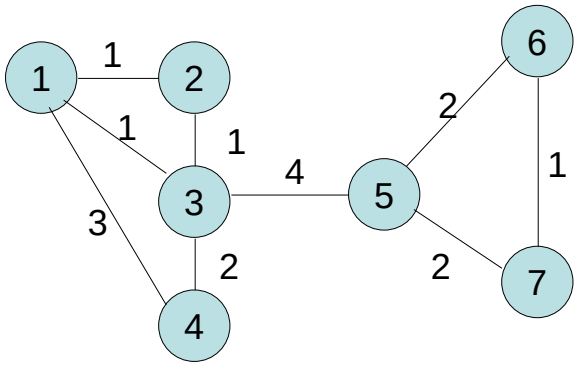
Kruskal's Algorithm



ที่มา McCracken, D.D. ,1987,p.390)

Graph

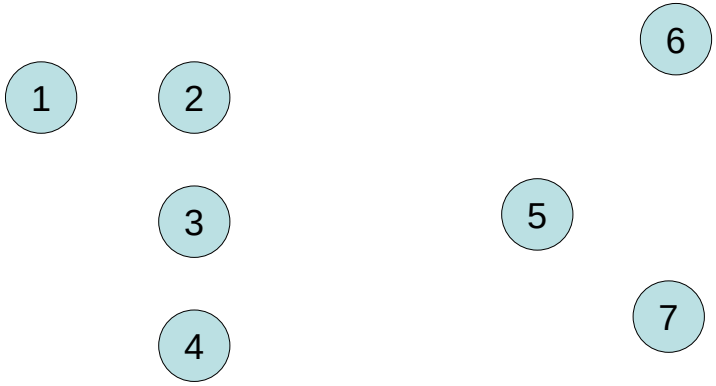
1. เรียงลำดับเอดจ์จากค่าน้ำหนัก



Edges	Weight
(1,2)	1
(1,3)	1
(2,3)	1
(6,7)	1
(3,4)	2
(5,6)	2
(5,7)	2
(1,4)	3
(3,5)	4

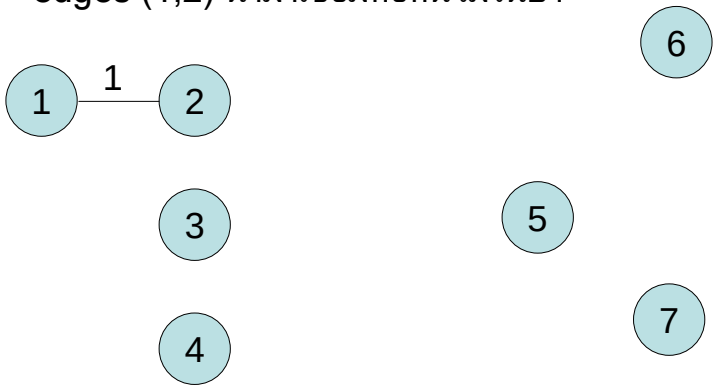
Graph

2. สร้างป่าที่ประกอบด้วยต้นไม้ว่างที่มีแต่โหนด และไม่มีเส้นเชื่อม



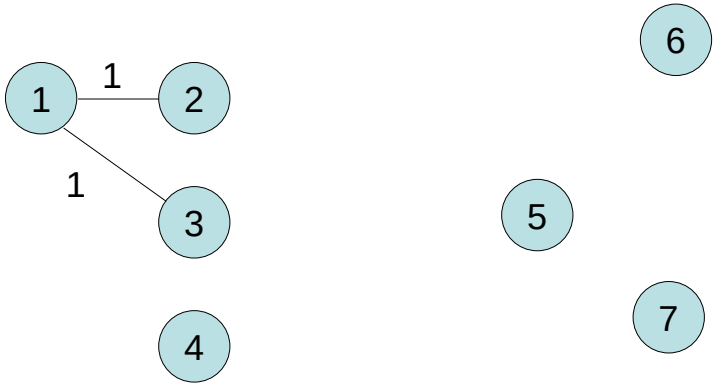
Graph

3. เลือกเอดจ์ที่มีน้ำหนักน้อยที่สุด มา 1 เส้น ตามตัวอย่าง คือ edges (1,2) นำมาเชื่อมต่อต้นไม้ในป่า



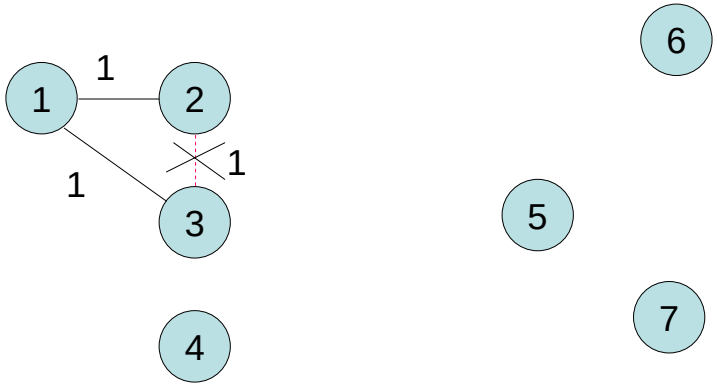
Graph

4. เลือกเอดจ์ที่เหลือและมีน้ำหนักน้อยที่สุดมา 1 เส้น ตาม ตามตัวอย่าง คือ edges (1,3) นำมาเชื่อมต่อต้นไม้ในป่า



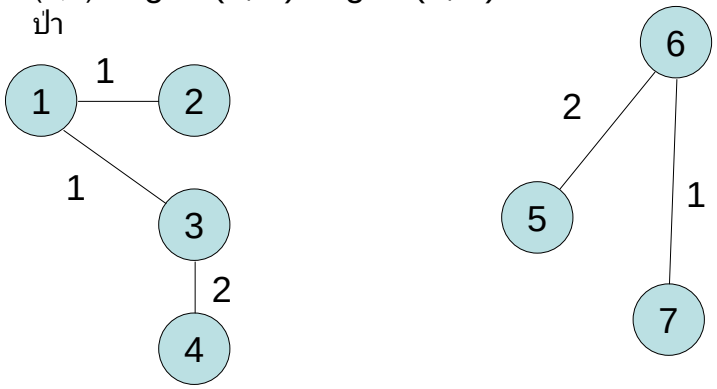
Graph

5. เลือกเอดจ์ที่เหลือและมีน้ำหนักน้อยที่สุดมา 1 เส้น ตามตัวอย่าง คือ edges (2,3) ตัดทิ้ง เนื่องจากทำให้เกิดวงรอบ (Cycle)



Graph

6. เลือกเอดจ์ที่เหลือและมีน้ำหนักน้อยที่สุดมา ตามตัวอย่าง คือ edges (6,7) edges (3,4) edges (5,6) นำมาเชื่อมต่อต้นไม่ในป่า

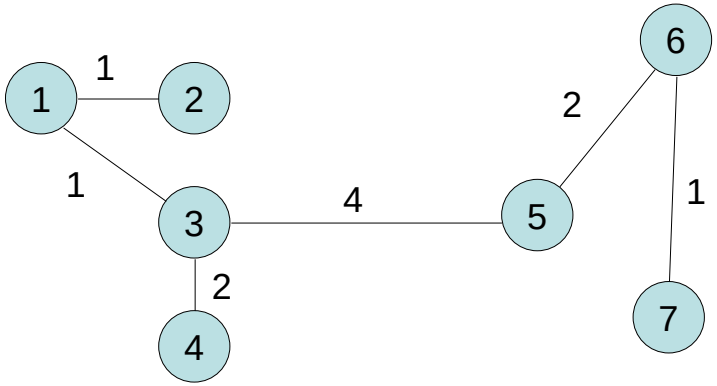


Graph

7. เลือกเอดจ์ที่เหลือและมีน้ำหนักน้อยที่สุด ตามตัวอย่าง คือ edges (5,7) จากนั้นให้ตัดทิ้งไม่นำมาเชื่อมต่อต้นไม้ในป่า เนื่องจากทำให้เกิดวงรอบ
8. เลือกเอดจ์ที่เหลือและมีน้ำหนักน้อยที่สุด ตามตัวอย่าง คือ edges (1,4) จากนั้นให้ตัดทิ้งไม่นำมาเชื่อมต่อต้นไม้ในป่า เนื่องจากทำให้เกิดวงรอบ

Graph

9. เลือกเอดจ์ที่เหลือและมีน้ำหนักน้อยที่สุดมา ตามตัวอย่าง คือ edges (3,5) นำมาเชื่อมต่อต้นไม้ในป่า เนื่องจากเป็นเอดจ์สุดท้าย



Graph

2. การหาเส้นทางที่สั้นที่สุด (Shortest path) Dijkstra's Algorithm

หาเส้นทางที่สั้นที่สุดจากโหนดต้นทางไปโหนดใด ๆ ในกราฟ มีน้ำหนัก และน้ำหนักไม่เป็นลบ

Graph

2. การหาเส้นทางที่สั้นที่สุด (Shortest path)

Dijkstra's Algorithm

<u>ข้อกำหนด</u>

ให้ เซต S เก็บโหนดที่ผ่านได้และมีระยะทางห่างจากจุดเริ่มต้นสั้นที่สุด

ให้ W แทนโหนด นอกเซต S

ให้ D แทนระยะทาง (distance) ที่สั้นที่สุดจากโหนดต้นทางไปโหนดใด ๆ ในกราฟ โดยวิถีนี้ประกอบด้วย โหนดในเซต S ถ้าไม่มีวิถี ให้แทนด้วยค่าอินฟินีตี้ (Infinity) : ∞

Graph

2. การหาเส้นทางที่สั้นที่สุด (Shortest path)

Dijkstra's Algorithm

2.1 เริ่มต้นให้เซต S มีเพียงโหนดเดียว คือโหนดที่เป็นจุดเริ่มต้น

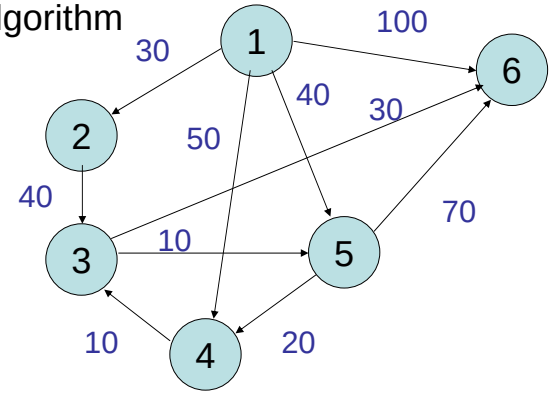
2.2 คำนวณหาระยะทางจาก โหนดที่เป็นจุดเริ่มต้น ไปยังโหนดทุกโหนดในกราฟ โดยยอมให้ใช้โหนด ในเซต S เป็นทางผ่านได้ ถ้ามีมากกว่า 1 ทาง ให้เลือกทางที่สั้นที่สุด นำไปใส่ใน D ของแต่ละโหนด

2.3 เลือกโหนด W ที่ห่างจากโหนดเริ่มต้นน้อยที่สุดไปไว้ใน S

Graph

Shortest path

Dijkstra's Algorithm



ที่มา McCracken, 1987,p.383)

Graph

2. การหาเส้นทางที่สั้นที่สุด (Shortest path)

Dijkstra's Algorithm

การคำนวณหาระยะทางสั้นที่สุด จากโหนดต้นทางคือโหนด 1 ไปยังโหนดใด ๆ มีวิธีคำนวณดังนี้

1) เริ่มต้นโหนดที่เป็นจุดเริ่มต้น คือ โหนด 1 ไปไว้ที่เซต S จากนั้นนำค่าน้ำหนักบนเอดจ์ (1,2) เอดจ์ (1,4) เอดจ์ (1,5) และ เอดจ์ (1,6) ไปเขียนในตาราง

สำหรับ โหนด 3 ไม่ได้ เชื่อมต่อกับโหนดที่ 1 ดังนั้นจึงใช้ค่าอินฟินีตี้ (Infinity) แทน แสดงในตารางที่ปรากฏในบรรทัด Iter= Initial

Graph

2. การหาเส้นทางที่สั้นที่สุด (Shortest path)
Dijkstra's Algorithm
การคำนวณหาระยะทางสั้นที่สุด
2) เลือก W ที่มีระยะทางสั้นที่สุด คือ โหนด 2 ไปไว้ที่เซต S
คำนวณ ระยะทางใหม่ ระยะทางสั้นที่สุด จากโหนด 1 ไปโหนดอื่น ๆ เท่าเดิม ยกเว้น
โหนด 3 ซึ่งขณะนี้มีวิถีกับโหนด 1 ดังนี้ (1,2,3) ระยะทางที่ได้มาจากน้ำหนักบนเอจน์เป็น (1,2) และ เอดจ์ (2,3) รวมกันคือ 70 จึงเขียนค่า 70 แทนค่าอินฟินีตีเดิม

Graph

2. การหาเส้นทางที่สั้นที่สุด (Shortest path)
Dijkstra's Algorithm
การคำนวณหาระยะทางสั้นที่สุด

3) เลือก W ที่มีระยะทางสั้นที่สุดคือโหนด 5 ไปไว้ที่เซต S
คำนวณหาระยะทางใหม่ปรากฏว่า
ถึงแม้จะมีโหนด 5 อยู่ในวิถีเส้นทางใหม่ แต่ระยะทางจากวิถีเดิมสั้นกว่า จึงคงค่าเดิมไว้ดังแสดงในตาราง

Graph

2. การหาเส้นทางที่สั้นที่สุด (Shortest path)
Dijkstra's Algorithm
การคำนวณหาระยะทางสั้นที่สุด

4) เลือก W ที่มีระยะทางสั้นที่สุดคือโหนด 4 ไปไว้ที่เซต S
คำนวณหาระยะทางใหม่ปรากฏว่า มีวิถีจากโหนด 1 ไปโหนด 3 รวม 2 วิถีดังนี้
วิถีที่ 1 คือ (1,2 และ3) มีค่าน้ำหนัก = 30+40 =70
วิถีที่ 2 คือ (1,4 และ3) มีค่าน้ำหนัก = 50+10 =60
เลือกน้ำหนักจากวิถีที่สั้นที่สุด คือ 60 ไปเขียนแทนค่าเดิม

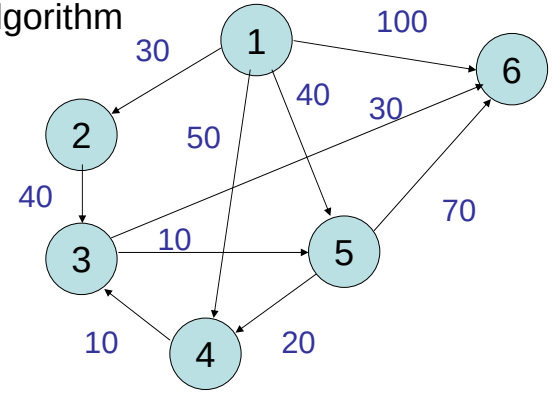
Graph

2. การหาเส้นทางที่สั้นที่สุด (Shortest path)
Dijkstra's Algorithm
การคำนวณหาระยะทางสั้นที่สุด

5) เลือก W ที่มีระยะทางสั้นที่สุดคือโหนด 3 ไปไว้ที่เซต S
คำนวณหาระยะทางใหม่ปรากฏว่า มีวิถีจากโหนด 1 ไปโหนด 3

Graph

Shortest path
Dijkstra's Algorithm



ที่มา McCracken, 1987,p.383)

Graph

แสดงระยะทางที่สั้นที่สุดของโหนดในกราฟเมื่อวิถีประกอบด้วยโหนดในเซต S

Iter	S	W	Dist[2]	Dist[3]	Dist[4]	Dist[5]	Dist[6]
Initial	{1}	-	30	∞	50	40	100
1	{1,2}	2	30	70	50	40	100
2	{1,2,5}	5	30	70	50	40	100
3	{1,2,5,4}	4	30	60	50	40	100
4	{1,2,5,4,3}	3	30	60	50	40	90
5	{1,2,5,3,6}	6	30	60	50	40	90

Sorting

เนื้อหา

- การเรียงลำดับ
- วิธีการเรียงลำดับ
- การเรียงลำดับแบบเลือก (selection sort)
- การเรียงลำดับแบบฟอง (bubble Sort)
- การเรียงลำดับแบบเร็ว (quick sort)
- การเรียงลำดับแบบแทรก (insertion sort)
- การเรียงลำดับแบบฐาน (radix sort)

Sorting (Cont.)

การเรียงลำดับแบบเร็ว (quick sort)

เป็นวิธีการเรียงลำดับที่ใช้เวลาน้อยเหมาะสำหรับข้อมูลที่มีจำนวนมากที่ต้องการความรวดเร็วในการทำงาน วิธีนี้จะเลือกข้อมูลจากกลุ่มข้อมูลขึ้นมาหนึ่งค่าเป็นค่าหลัก แล้วหาตำแหน่งที่ถูกต้องให้กับค่าหลักนี้ เมื่อได้ตำแหน่งที่ถูกต้องแล้ว ใช้ค่าหลักนี้เป็นหลักในการแบ่งข้อมูลออกเป็นสองส่วน ถ้าเป็นการเรียงลำดับจากน้อยไปมาก ส่วนแรกอยู่ในตอนหน้าข้อมูล ทั้งหมดจะมีค่าน้อยกว่าค่าหลักที่เป็นตัวแบ่งส่วน

Sorting (Cont.)

อีกส่วนหนึ่งจะอยู่ในตำแหน่งตอนหลังข้อมูลทั้งหมด จะมีค่ามากกว่าค่าหลัก แล้วนำแต่ละส่วนย่อยไปแบ่งย่อยในลักษณะเดียวกันต่อไปจนกระทั่งแต่ละส่วนไม่สามารถแบ่งย่อยได้อีก ต่อไปจะได้ข้อมูลที่มีการเรียงลำดับตามที่ต้องการ

Sorting (Cont.)

ถ้าเป็นการเรียงลำดับจากน้อยไปมาก

การเปรียบเทียบเพื่อหาตำแหน่งให้กับค่าหลัก(Control Key)ตัวแรกเริ่มจากข้อมูลในตำแหน่งแรกหรือสุดท้ายก็ได้

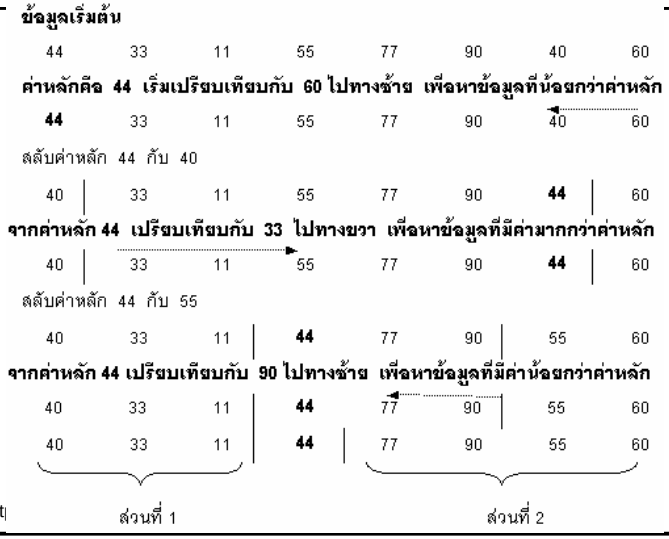
ถ้าเริ่มจากข้อมูลที่ตำแหน่งที่ 1 เป็นค่าหลัก พิจารณาเปรียบเทียบค่าหลักกับข้อมูลในตำแหน่งสุดท้าย

ถ้าค่าหลักมีค่าน้อยกว่าให้เปรียบเทียบกับข้อมูลในตำแหน่งรองสุดท้ายไปเรื่อย ๆ จนกว่าจะพบค่าที่น้อยกว่าค่าหลัก แล้วให้สลับตำแหน่งกัน

Sorting (Cont.)

หลังจากสลับตำแหน่งแล้วนำค่าหลักมาเปรียบเทียบกับข้อมูล ในตำแหน่งที่ 2, 3, ไปเรื่อย ๆ จนกว่าจะพบค่าที่มากกว่าค่าหลัก สลับตำแหน่งเมื่อเจอข้อมูลที่มากกว่าค่าหลัก ทำเช่นนี้ไปเรื่อย ๆ จนกระทั่งได้ตำแหน่งที่ถูกต้องของค่าหลักนั้น ก็จะแบ่งกลุ่มข้อมูลออกเป็นสองส่วน ส่วนแรกข้อมูลทั้งหมดมีค่าน้อยกว่าค่าหลัก และส่วนที่สองข้อมูลทั้งหมดมีค่ามากกว่าค่าหลัก

Sorting (Cont.)



Sorting (Cont.)

จากการเปรียบเทียบข้างต้นในที่สุดก็ได้ตำแหน่งที่วางค่าหลัก 44 ซึ่งข้อมูลจะถูกแบ่งเป็น 2 ส่วน ส่วนที่ 1 ข้อมูลทั้งหมดมีค่าน้อยกว่าค่าหลัก และส่วนที่ 2 ข้อมูลทั้งหมดมีค่ามากกว่าค่าหลัก นำแต่ละส่วนไปดำเนินการเปรียบเทียบในลักษณะเดียวกันจนกระทั่งข้อมูลทั้งหมดเรียงลำดับจากน้อยไปมากตามต้องการ

Sorting (Cont.)

การจัดเรียงลำดับแบบเร็วเป็นวิธีที่ค่อนข้างซับซ้อน แต่ประสิทธิภาพการทำงานค่อนสูง เนื่องจากใช้เวลาในการเรียงลำดับน้อย ถ้ามีข้อมูลทั้งหมด n ตัวจำนวนครั้งของการเปรียบเทียบเป็นดังนี้

กรณีที่ดีที่สุด คือ กรณีที่ค่าหลักที่เลือกแบ่งแล้วข้อมูลอยู่ตรงกลางกลุ่มพอดี และในแต่ละส่วนย่อยก็เช่นเดียวกัน จำนวนครั้งของการเปรียบเทียบเป็นดังนี้

จำนวนครั้งของการเปรียบเทียบ = $n \log_2 n$ ครั้ง

Sorting (Cont.)

กรณีที่แย่ที่สุด คือ กรณีที่ข้อมูลมีการเรียงลำดับอยู่แล้ว อาจจะเรียงจากน้อยไปมากหรือจากมากไปน้อย หรือค่าหลักที่เลือกในแต่ละครั้งเป็นค่าหลักที่น้อยที่สุดหรือมากที่สุด จำนวนครั้งของการเปรียบเทียบจะมากที่สุดดังนี้

จำนวนครั้งของการเปรียบเทียบ

$= (n-1) + (n-2) + \ldots + 3 + 2 + 1$

$= n(n-1)/2$ ครั้ง

Searching

การค้นหาข้อมูล (Searching)

การค้นหา คือ
การใช้วิธีการค้นหากับโครงสร้างข้อมูล เพื่อดูว่าข้อมูลตัวที่ต้องการถูกเก็บอยู่ในโครงสร้างแล้วหรือยัง

วัตถุประสงค์ของการค้นหาโดยทั่วไป ได้แก่
เพื่อดูรายละเอียดเฉพาะข้อมูลส่วนที่ต้องการ
ดึงข้อมูลตัวที่ค้นหาออกจากโครงสร้าง
เปลี่ยนแปลงแก้ไขรายละเอียดบางอย่างของข้อมูลตัวที่ค้นพบ และ/หรือ
เพิ่มข้อมูลตัวที่ค้นหาแล้วพบว่ายังไม่เคยเก็บไว้ในโครงสร้างเลย เข้าไปเก็บไว้ในโครงสร้าง เพื่อใช้งานต่อไป

Searching

การค้นหาข้อมูล (Searching)

การค้นหา คือ
แบ่งเป็น 2 ประเภท ตามแหล่งที่จัดเก็บข้อมูลเช่นเดียวกับการเรียงลำดับ
การค้นหาข้อมูลแบบภายใน (Internal Searching)
การค้นหาข้อมูลแบบภายนอก (External Searching)

Searching

1. การค้นหาแบบเชิงเส้นหรือการค้นหาตามลำดับ (Linear)
เป็นวิธีที่ใช้กับข้อมูลที่ยังไม่ได้เรียงลำดับ

หลักการ คือ ให้นำข้อมูลที่จะหามาเปรียบเทียบกับข้อมูลตัวแรกในแถวลำดับ
ถ้าไม่เท่ากันให้เปรียบเทียบกับข้อมูลตัวถัดไป
ถ้าเท่ากันให้หยุดการค้นหา หรือการค้นหาจะหยุดเมื่อพบข้อมูลที่ต้องการหรือหาข้อมูลทุกจำนวนในแถวลำดับแล้วไม่พบ

Searching

2. การค้นหาแบบเซนทินัล (Sentinel)
เป็นวิธีที่การค้นหาแบบเดียวกับวิธีการค้นหาแบบเชิงเส้น แต่ประสิทธิภาพดีกว่าตรงที่เปรียบเทียบน้อยครั้งกว่า พัฒนามาจากอัลกอริทึมแบบเชิงเส้น

หลักการ

1) เพิ่มขนาดของแถวลำดับ ที่ใช้เก็บข้อมูลอีก 1 ที่
2) นำข้อมูลที่จะใช้ค้นหาข้อมูลใน Array ไปฝากที่ต้นหรือ ท้าย Array
3) ตรวจสอบผลลัพธ์จากการหา
โดยตรวจสอบจากตำแหน่งที่พบ ถ้าตำแหน่งที่พบมีค่าเท่ากับ n-1 แสดงว่าหาไม่พบ นอกนั้นถือว่าพบข้อมูลที่ค้าหา

Searching

3. การค้นหาแบบไบนารี (Binary Search)

การค้นหาแบบไบนารีใช้กับข้อมูลที่ **ถูกจัดเรียงแล้วเท่านั้น**
หลักการของการค้นหาคือ ข้อมูลถูกแบ่งออกเป็นสองส่วน แล้วนำค่ากลางข้อมูลมาเปรียบเทียบกับคีย์ที่ต้องการหา
1.หาตัวแทนข้อมูลเพื่อนำมาเปรียบเทียบกับค่าที่ต้องการค้น
ตำแหน่งตัวแทนข้อมูลหาได้จากสูตร
mid = (low+high)/2
mid คือ ตำแหน่งกลาง ,low คือ ตำแหน่งต้นแถวลำดับ
high คือ ตำแหน่งท้ายของแถวลำดับ

Searching

3. การค้นหาแบบไบนารี (Binary Search)

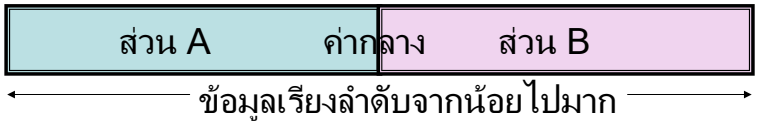
2. นำผลการเปรียบเทียบกรณีที่หาไม่พบมาใช้ในการค้นหารอบต่อไป

หาในส่วน A

ถ้าค่าที่จะหานั้นน้อยกว่าค่าที่ตำแหน่งกลาง ในทางกลับกัน

หาในส่วน B

ถ้าค่าที่จะหานั้นมากกว่าค่าตำแหน่งกลาง



Searching

การค้นหาแบบไบนารี (Binary Search)

ถ้าข้อมูลมีการเรียงจากน้อยไปหามาก เมื่อเปรียบเทียบแล้วคีย์มีค่ามากกว่าค่ากลาง แสดงว่าต้องทำการค้นหาข้อมูลในครึ่งหลังต่อไป จากนั้นนำข้อมูลครึ่งหลังมาหาค่ากลางต่อ ทำอย่างนี้ไปเรื่อย ๆ จนกว่าจะได้ข้อมูลที่ต้องการ เช่นต้องการหาว่า 12 อยู่ในลิสต์ (**1 4 6 8 10 12 18 19**) หรือไม่

Searching

การค้นหาแบบไบนารี (Binary Search)

เริ่มการค้นหาแบบไบนารีด้วยการเปรียบเทียบกับค่ากลางในลิสต์ คือค่า a[4] ซึ่งเก็บค่า 8 ซึ่ง 12 > a[4] หมายความว่าค่า 12 ควรจะอยู่ในข้อมูลด้านขวาของ a[4] คือ ช่วง a[5] ...a[8] โดยไม่สนใจช่วงข้อมูล a[1] ...a[3] และ ใช้วิธีการค้นหาแบบไบนารีเช่นเดิมอีกกับชุดข้อมูลครึ่งหลัง คือ a[5] ...a[8] นั่นคือ เปรียบเทียบกับค่ากลางของชุดข้อมูลครึ่งหลัง (a[5] ...a[8]) คือค่า a[6] ซึ่งเก็บค่า 12 ซึ่ง 12 = a[6] จะได้ว่าค่า 12 อยู่ในตำแหน่งที่ 6 ในลิสต์

Searching

การค้นหาแบบไบนารี (Binary Search)

จากตัวอย่างนี้ การค้นหาค่า 12 โดยวิธีการค้นหาแบบไบนารีใช้การเปรียบเทียบ เพียง 2 ครั้ง อย่างไรก็ตามในตัวอย่างนี้ ภายใต้สมมุติฐานว่าข้อมูลที่ต้องการค้นหามีอยู่ในลิสต์ จะใช้การเปรียบเทียบอย่างมากที่สุดเพียง 3 () ครั้ง เช่น ถ้าต้องการหาค่า 19 ดังแสดงในรูป

Searching

การค้นหาแบบไบนารี (Binary Search)

(ในทางกลับกันการค้นหาแบบ Sequential Search ต้องเปรียบเทียบจากค่าในตำแหน่งแรกไปจนถึงค่าในตำแหน่งสุดท้าย นั่นคือมีการเปรียบเทียบ 8 ครั้ง) หรือ

ถ้าค่าที่ต้องการหาไม่อยู่ในลิสต์ จะใช้การเปรียบเทียบเพียง 4 ครั้ง

ถ้าค่าที่ต้องการหาไม่อยู่ในลิสต์ เช่น ถ้าต้องการหาค่า 5

ครั้งที่ 1

1 4 6 8 10 12 18 19