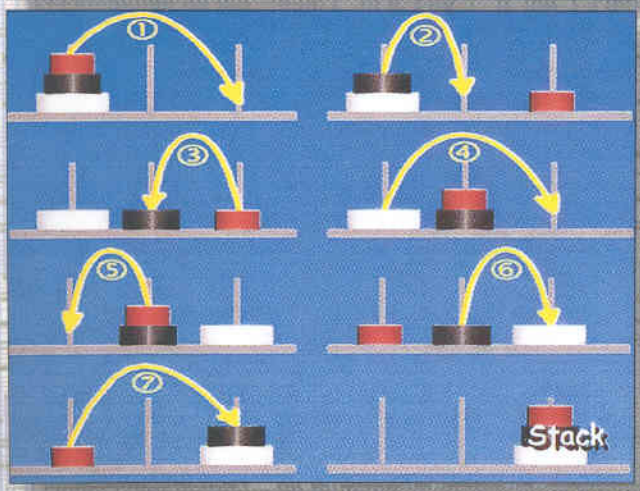
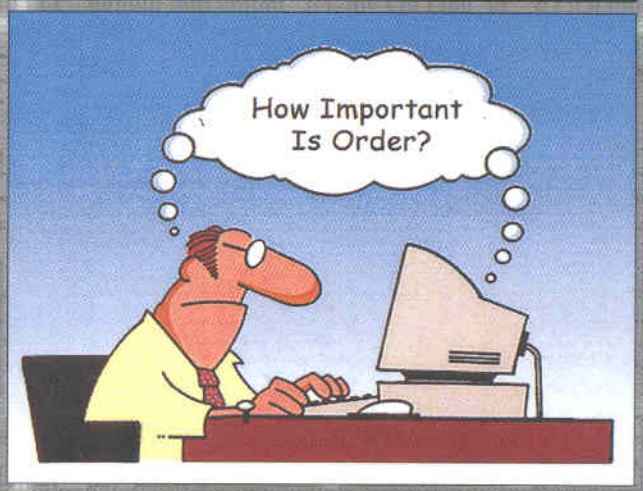
โครงสร้างข้อมูล และอัลกอริทึม

(แถวลำดับ รายการ สแตก คิว กราฟ ต้นไม้ อัลกอริทึมการเรียงลำดับ การค้นหา การวิเคราะห์อัลกอริทึม)

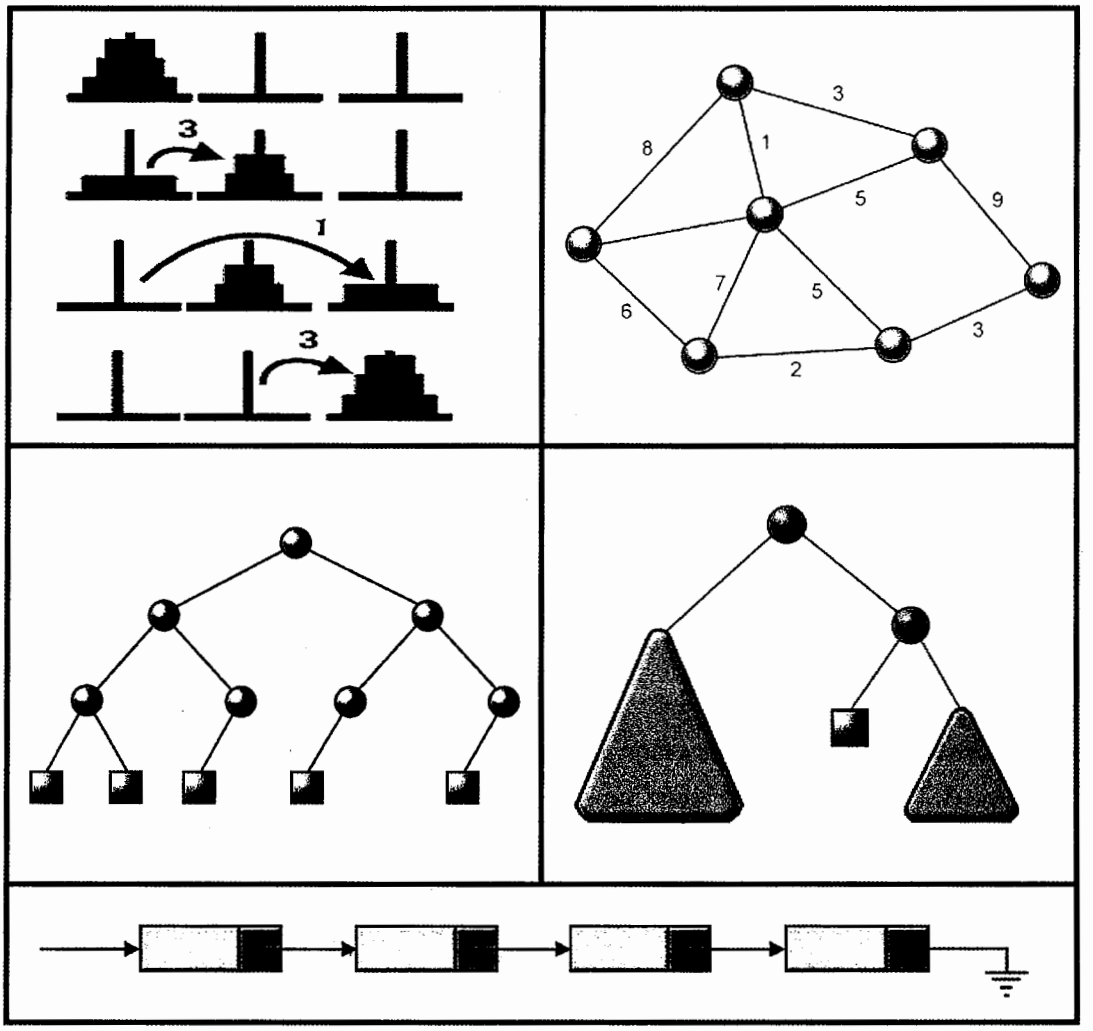




โครงการตำราวิทยาศาสตร์และคณิตศาสตร์มูลนิธิ สอวน.

โครงสร้างข้อมูล และอัลกอริทึม

(แถวลำดับ รายการ สแตก คิว กราฟ ต้นไม้ อัลกอริทึมการเรียงลำดับ การค้นหา การวิเคราะห์อัลกอริทึม)



โครงการตำราวิทยาศาสตร์และคณิตศาสตร์มูลนิธิ สอวน.

มูลนิธิส่งเสริมโอลิมปิกวิชาการและพัฒนามาตรฐานวิทยาศาสตร์ศึกษา ในพระอุปถัมภ์สมเด็จพระเจ้าพี่นางเธอ เจ้าฟ้ากัลยาณิวัฒนา กรมหลวงนราธิวาสราชนครินทร์ (สอวน.)

ความเป็นมา

ประเทศไทยส่งนักเรียนเข้าแข่งขันคณิตศาสตร์โอลิมปิกระหว่างประเทศครั้งแรกในปี พ.ศ. 2532 ที่ประเทศเยอรมนี โดยความร่วมมือระหว่างสมาคมวิทยาศาสตร์แห่งประเทศไทยในพระบรมราชูปถัมภ์ และสมาคมคณิตศาสตร์แห่งประเทศไทยในพระบรมราชูปถัมภ์ สมเด็จพระเจ้าพี่นางเธอ เจ้าฟ้ากัลยาณิวัฒนา (พระยศในขณะนั้น) ได้พระราชทานเงินส่วนพระองค์จำนวนหนึ่งเพื่อเป็นค่าใช้จ่าย

การคัดเลือกนักเรียนเพื่อไปแข่งขันคณิตศาสตร์โอลิมปิกระหว่างประเทศ ครั้งที่ 30 สมาคมคณิตศาสตร์แห่งประเทศไทยฯ ได้จัดคณาจารย์มหาวิทยาลัยจำนวนหนึ่งมาช่วยฝึกอบรมรวมเป็นเวลาประมาณสองเดือน เพื่อให้นักเรียนได้เรียนรู้เนื้อหาเพิ่มเติมครอบคลุมหลักสูตรที่จะใช้แข่งขัน ซึ่งอยู่ในระดับชั้นปีที่ 1-2 ของมหาวิทยาลัย จากผลสำเร็จในปีแรก ทำให้รัฐบาลเห็นความสำคัญจึงได้จัดสรรงบประมาณให้กับโครงการนี้ผ่านสถาบันส่งเสริมการสอนวิทยาศาสตร์และเทคโนโลยี (สสวท.) ตั้งแต่ปี พ.ศ. 2533 เป็นต้นมา สมทบกับเงินพระราชทานเป็นรายปีจากสมเด็จพระเจ้าพี่นางเธอ เจ้าฟ้ากัลยาณิวัฒนา กรมหลวงนราธิวาสราชนครินทร์ เพื่อสนับสนุนการส่งนักเรียนไทยไปแข่งขันโอลิมปิกวิชาการระหว่างประเทศ 5 สาขา จนถึงปัจจุบันรวม 14 ปี นักเรียนไทยได้ทำชื่อเสียงได้เหรียญทองรวม 17 เหรียญ เหรียญเงิน 53 เหรียญ เหรียญทองแดง 93 เหรียญ และเกียรติคุณประกาศอีก 35 ราย รวมทั้งสิ้น 198 รางวัล จากจำนวนนักเรียนที่ส่งไปเข้าแข่งขัน 276 คน (72%)

อย่างไรก็ตาม จากการที่ประเทศไทยดำเนินการจัดส่งนักเรียนเข้าร่วมการแข่งขันคณิตศาสตร์ วิทยาศาสตร์ โอลิมปิกระหว่างประเทศ ตั้งแต่ พ.ศ.2532 จนถึงปัจจุบัน ทำให้เห็นว่ามาตรฐานการศึกษาด้านคณิตศาสตร์และวิทยาศาสตร์ของไทยยังต่ำกว่ามาตรฐานสากล การเตรียมตัวของนักเรียนยังใช้เวลาน้อยเกินไป

เพื่อให้การส่งเสริมและสนับสนุนโครงการจัดส่งผู้แทนประเทศไทยไปแข่งขันโอลิมปิกวิชาการระหว่างประเทศมีประสิทธิภาพยิ่งขึ้นสมเด็จพระเจ้า พี่นางเธอ เจ้าฟ้ากัลยาณิวัฒนา กรมหลวงนราธิวาสราชนครินทร์ จึงมีพระดำริให้จัดตั้งมูลนิธิ สอวน. และได้รับอนุมัติจากกระทรวงมหาดไทยเมื่อวันที่ 12 ตุลาคม 2542 โดยมีวัตถุประสงค์หลัก 2 ประการ

วัตถุประสงค์

1. ส่งเสริมให้นักเรียนในระดับมัธยมศึกษาตอนปลายทั่วประเทศที่มีความสามารถทางวิทยาศาสตร์และคณิตศาสตร์ มีโอกาสได้รับการพัฒนาศักยภาพทางด้านคณิตศาสตร์ คอมพิวเตอร์ เคมี ฟิสิกส์ และชีววิทยา ตามความถนัดทั้งด้านทฤษฎีและทักษะด้านปฏิบัติ ให้สามารถคิดวิเคราะห์และแก้ปัญหาที่ซับซ้อนได้ โดยจัดให้มีศูนย์อบรมทั่วประเทศเป็นการเพิ่มเวลาฝึกอบรม เพื่อช่วยให้นักเรียนมีความพร้อมที่จะเข้ารับการคัดเลือกไปแข่งขันโอลิมปิกวิชาการระหว่างประเทศให้ได้ผลดียิ่งขึ้น ตามพระดำริของสมเด็จพระเจ้าพี่นางเธอเจ้าฟ้ากัลยาณิวัฒนา กรมหลวงนราธิวาสราชนครินทร์ องค์ประธานมูลนิธิ สอวน.
2. เพื่อนำประสบการณ์ที่ได้จากการแข่งขันโอลิมปิกวิชาการระหว่างประเทศมาพัฒนามาตรฐานคณิตศาสตร์และวิทยาศาสตร์ศึกษาของไทยให้สูงขึ้นเทียบเท่าระดับสากล

โครงการตำราวิทยาศาสตร์และคณิตศาสตร์มูลนิธิ สอวน.

มูลนิธิส่งเสริมโอลิมปิกวิชาการและพัฒนามาตรฐานวิทยาศาสตร์ศึกษา ในพระอุปถัมภ์สมเด็จพระเจ้าพี่นางเธอ เจ้าฟ้ากัลยาณิวัฒนา กรมหลวงนราธิวาสราชนครินทร์ (สอวน.) ได้ร่วมมือกับคณาจารย์มหาวิทยาลัยของรัฐ 20 แห่ง และกระทรวงศึกษาธิการ ดำเนินการจัดตั้งศูนย์ สอวน. ในภูมิภาค 12 ศูนย์ และในกรุงเทพฯ 1 ศูนย์ (5 โรงเรียน) เพื่อพัฒนาศักยภาพทางปัญญาของนักเรียนที่มีความพร้อมในด้านวิทยาศาสตร์ คณิตศาสตร์และคอมพิวเตอร์จากทั่วประเทศ เพื่อให้นักเรียนได้มีความรู้ความสามารถเทียบเท่ามาตรฐานสากล และพร้อมที่จะสอบคัดเลือกเป็นผู้แทนประเทศไทยไปร่วมการแข่งขันโอลิมปิกวิชาการระหว่างประเทศในสาขาวิชาต่างๆ ได้ และเพื่อขยายผลจากประสบการณ์ที่ได้เข้าแข่งขันโอลิมปิกวิชาการระหว่างประเทศมาพัฒนามาตรฐานวิทยาศาสตร์และคณิตศาสตร์ของไทยให้สูงขึ้นเทียบเท่าสากล

ในการอบรมนักเรียนของศูนย์ สอวน. มูลนิธิ สอวน.ได้พิจารณาเห็นว่าตำราที่มีความสมบูรณ์ของเนื้อหาตามหลักสูตรของ สอวน. จะช่วยพัฒนาศักยภาพของนักเรียนในศูนย์สอวน.ให้สูงยิ่งขึ้นได้ นอกจากนั้นตำราเรียนในวิชาวิทยาศาสตร์และคณิตศาสตร์ในระดับมัธยมศึกษาตอนปลายที่มีความสมบูรณ์ถูกต้องยังขาดแคลน นักเรียนและครูสามารถนำตำราที่พัฒนาขึ้นมานี้ไปใช้เป็นหนังสืออ้างอิงประกอบการเรียนการสอนได้อีกด้วย มูลนิธิ สอวน. จึงได้มี "โครงการจัดทำตำราส่งเสริมพัฒนาศักยภาพของนักเรียนด้านวิทยาศาสตร์ คณิตศาสตร์และคอมพิวเตอร์" ขึ้น เพื่อผลิตตำราคณิตศาสตร์ คอมพิวเตอร์ เคมี ชีววิทยาและฟิสิกส์ที่มีเนื้อหาหลักสูตรตามมาตรฐานสากล โดยมีวัตถุประสงค์

1. เพื่อเป็นที่ระลึกในมหามงคลสมัยที่สมเด็จพระเจ้าพี่นางเธอ เจ้าฟ้ากัลยาณิวัฒนา กรมหลวงนราธิวาสราชนครินทร์ ทรงเจริญพระชนมายุ 80 พรรษา ซึ่งพระองค์ทรงมีพระมหากรุณาธิคุณต่อโครงการจัดส่งผู้แทนประเทศไทยไปแข่งขันโอลิมปิกวิชาการระหว่างประเทศและทรงสนับสนุนการดำเนินงานเพื่อพัฒนามาตรฐานการศึกษาวิทยาศาสตร์และคณิตศาสตร์ของไทยอย่างหาที่สุดมิได้

2. เพื่อผลิตหนังสือคณิตศาสตร์ คอมพิวเตอร์ เคมี ชีววิทยาและฟิสิกส์ระดับมัธยมศึกษาตอนปลายถึงระดับชั้นปีที่ 1 ของคณะวิทยาศาสตร์ในมหาวิทยาลัยให้มีคุณภาพเทียบเท่าสากล เรียบเรียงโดยคณาจารย์จากมหาวิทยาลัยของรัฐที่มีประสบการณ์สูงและที่มีส่วนรวมในการฝึกอบรมนักเรียนในค่าย สอวน. ค่าย สสวท. และควบคุมนักเรียนไปแข่งขันโอลิมปิกวิชาการระหว่างประเทศ เนื้อหาในหนังสือเน้นกระบวนการคิดแบบวิทยาศาสตร์ เพื่อฝึกฝนให้นักเรียนสามารถคิดวิเคราะห์และแก้ปัญหาที่ซับซ้อนทั้งในเชิงทฤษฎีและประยุกต์ได้โดยจัดพิมพ์บนกระดาษอาร์ตอย่างดีและมีภาพสีประกอบคำบรรยาย

โครงการตำราวิทยาศาสตร์และคณิตศาสตร์ของมูลนิธิ สอวน. มีเป้าหมายในการผลิตตำราคณิตศาสตร์ รวม 5 เล่ม คอมพิวเตอร์ 3 เล่ม เคมี 3 เล่ม ชีววิทยา 5 เล่ม และฟิสิกส์ 3 เล่ม โดยหนังสือชุดแรกจะนำขึ้นทูลเกล้าถวายในวันที่ 6 มิถุนายน 2547 ซึ่งเป็นวันประชุมสามัญประจำปี พ.ศ. 2547 ของคณะกรรมการบริหาร และสมเด็จพระเจ้าพี่นางเธอ เจ้าฟ้ากัลยาณิวัฒนา กรมหลวงนราธิวาสราชนครินทร์ เสด็จเป็นองค์ประธานที่ประชุม ส่วนที่เหลือคาดว่าจะจัดพิมพ์ให้แล้วเสร็จภายในปี พ.ศ. 2547 นี้ นอกจากนั้นคณะกรรมการโครงการผลิตตำราวิทยาศาสตร์และคณิตศาสตร์มูลนิธิ สอวน. มีเป้าหมายจะผลิตตำราที่มีคุณภาพสูงนี้ในระดับมัธยมศึกษาต้นโดยคณาจารย์จากมหาวิทยาลัย เพื่อวางรากฐานวิทยาศาสตร์และคณิตศาสตร์พื้นฐานและเพื่อสร้างแรงบันดาลใจและความสนใจทางด้านวิทยาศาสตร์ คณิตศาสตร์แก่เยาวชนอีกด้วย หากได้รับความอนุเคราะห์ช่วยเหลือด้านงบประมาณจากภาครัฐและเอกชน

คณะจัดทำหนังสือคณิตศาสตร์ วิทยาศาสตร์หวังว่าโครงการตำราคณิตศาสตร์ วิทยาศาสตร์ของมูลนิธิ สอวน. จะมีส่วนร่วมในการปฏิรูปการเรียนรู้และยกระดับมาตรฐานวิทยาศาสตร์และคณิตศาสตร์ของไทยให้เทียบเท่ากับระดับสากลเพื่อสนองพระดำริของสมเด็จพระเจ้าพี่นางเธอ เจ้าฟ้ากัลยาณิวัฒนา กรมหลวงนราธิวาสราชนครินทร์

****

คณะกรรมการโครงการตำราวิทยาศาสตร์และคณิตศาสตร์มูลนิธิ สอวน.

คณะกรรมการที่ปรึกษา

1. ศาสตราจารย์ นายแพทย์ จรัส สุวรรณเวลา	รองประธานมูลนิธิ สอวน.
2. รองศาสตราจารย์ ดร.กำจัด มงคลกุล	กรรมการมูลนิธิฯ
3. ดร. คุณหญิงกษมา วรวรรณ ณ อยุธยา	กรรมการมูลนิธิฯ
4. รองศาสตราจารย์ ดร. คุณหญิงสุมณฑา พรหมบุญ	กรรมการมูลนิธิฯ
5. นายเชาว์ อรรถมานะ (อดีต)	กรรมการมูลนิธิฯ
6. นายสมนึก พิมลเสถียร	กรรมการมูลนิธิฯ
7. นายสุนทร อรุณานนท์ชัย	กรรมการมูลนิธิฯ
8. รองศาสตราจารย์บุญรักษา สุนทรธรรม	กรรมการมูลนิธิฯ

คณะกรรมการดำเนินงานโครงการตำราฯ

1. ศาสตราจารย์ศักดา ศิริพันธุ์ (ราชบัณฑิต) เลขาธิการมูลนิธิ สอวน.	ประธาน
2. รองศาสตราจารย์เย็นใจ สมวิเชียร	รองประธาน
3. รองศาสตราจารย์ ดร. ณรงค์ ปั้นนิ่ม	กรรมการ (วิชาคณิตศาสตร์)
4. รองศาสตราจารย์ยืน ภู่วรวรรณ	กรรมการ (วิชาคอมพิวเตอร์)
5. รองศาสตราจารย์ ดร. พินิติ รตะนานุกูล	กรรมการ (วิชาเคมี)
6. ผู้ช่วยศาสตราจารย์ ดร.วุทธิพันธุ์ ปรัชญพฤทธิ์	กรรมการ (วิชาฟิสิกส์)
7. ศาสตราจารย์อักษร ศรีเปล่ง	กรรมการ (วิชาชีววิทยา ประเภทพืช)
8. รองศาสตราจารย์ ดร. อุษณีย์ ยศยิ่งยวด	กรรมการ (วิชาชีววิทยา ประเภทสัตว์)

คณะผู้เขียนวิชาคอมพิวเตอร์

1. รองศาสตราจารย์ยืน ภู่วรวรรณ	ประธานคณะอนุกรรมการ
2. รองศาสตราจารย์ ดร. อนงค์นาฏ ศรีวิหค	อนุกรรมการ
3. ผู้ช่วยศาสตราจารย์อุมาพร ศิรธรานนท์	อนุกรรมการ
4. ผู้ช่วยศาสตราจารย์กัลยาณี บรรจงจิตร	อนุกรรมการ
5. ผู้ช่วยศาสตราจารย์นงนุช สุขวารี	อนุกรรมการ
6. ผู้ช่วยศาสตราจารย์กรรณิกา คงสาคร	อนุกรรมการ
7. อาจารย์ศิริกร จันทร์นวล	อนุกรรมการ
8. อาจารย์พบสิทธิ์ กมลเวช	อนุกรรมการ
9. ดร. นวลวรรณ สุนทรภิษัช	อนุกรรมการ
10. ดร. สุขุมาล กิติสิน	อนุกรรมการ
11. อาจารย์พัชรี เลิศจิตรศิลป์	อนุกรรมการ
12. อาจารย์มาริสา มัยยะ	อนุกรรมการ

คำนำ

การแข่งขันคอมพิวเตอร์โอลิมปิกระดับนานาชาติ เน้นให้นักเรียนรู้จักกับการเขียนโปรแกรมคอมพิวเตอร์เพื่อแก้ปัญหาที่มีความซับซ้อนสูงได้ ผู้เข้าแข่งขันจะได้รับโจทย์ปัญหา และต้องเขียนโปรแกรมคอมพิวเตอร์เพื่อหาคำตอบ และต้องทำให้ได้ภายใต้เงื่อนไขที่กำหนดอีกหลายอย่าง เช่น การใช้หน่วยความจำที่จำกัด การใช้เวลาในการประมวลผลที่กำหนด

การเขียนโปรแกรมแก้ปัญหาโจทย์เหล่านี้นับเป็นเรื่องที่ดี นักเรียนจะต้องมีพื้นความรู้ทางด้านวิทยาการคอมพิวเตอร์อีกหลายเรื่อง ทั้งทางด้านคณิตศาสตร์ ด้านโครงสร้างข้อมูล และขั้นตอนวิธีการแก้ปัญหาหรืออัลกอริทึม ร่วมกับการสร้างประสบการณ์การเขียนโปรแกรม และเทคนิคการเขียนโปรแกรมที่มีประสิทธิภาพ ซึ่งปัจจุบันนักเรียนไทยยังขาดความรู้พื้นฐานเหล่านี้อยู่อีกมาก

การพัฒนานักเรียนในระดับมัธยมศึกษาที่ดีอีกทางหนึ่งคือ การเสริมสร้างเอกสาร ตำราวิชาการ และชี้แนวทางของการศึกษาทางด้านคอมพิวเตอร์พื้นฐานที่ถูกต้องและชัดเจน เพราะหลายคนคิดว่าการเขียนโปรแกรมคอมพิวเตอร์ได้เท่านั้นเป็นสิ่งที่ต้องการ แต่ความเป็นจริงต้องสามารถแก้โจทย์ปัญหาที่มีความซับซ้อนสูงได้อย่างมีหลักการ ตำราวิชาการชุดนี้จึงมีหลายเล่ม ตั้งแต่คณิตศาสตร์ไม่ต่อเนื่อง (Discrete Mathematics) การเขียนโปรแกรมภาษาซี โครงสร้างข้อมูลและอัลกอริทึม เป็นต้น

หนังสือเล่มนี้ มีเนื้อหาที่เน้นในเรื่องของโครงสร้างข้อมูลพื้นฐานที่ใช้ในคอมพิวเตอร์และพื้นฐานของอัลกอริทึม โดยได้วางรากฐานการศึกษาในระดับพื้นฐานที่จะนำไปประยุกต์ใช้ ซึ่งนับว่ามีความจำเป็นของการศึกษาทางด้านนี้อย่างยิ่ง โดยเนื้อหาที่กล่าวถึงเน้นสู่ภาคปฏิบัติ เพื่อให้เขียนโปรแกรมประยุกต์ใช้ต่อไปได้

ที่ปรึกษา

รองศาสตราจารย์ยืน ภู่วรวรรณ

วศ.บ. (วิศวกรรมไฟฟ้าสื่อสาร) เกียรตินิยม, วศ. ม. (วิศวกรรมไฟฟ้า) จุฬาลงกรณ์มหาวิทยาลัย

M.Eng (Industrial Engineering and Management) at Asian Institute of Technology

ผู้เขียน

รองศาสตราจารย์ ดร. อนงค์นาฏ ศรีวิหค

วท.บ. (จุลชีววิทยา) จุฬาลงกรณ์มหาวิทยาลัย

M.S. (Bacteriology) University of Arkansas

Ph.D. (Information Systems) Central Queensland University

อาจารย์ ศิริกร จันทร์นวล

สต.บ. (การประมวลผลด้วยเครื่องจักร) เกียรตินิยมอันดับ 2 จุฬาลงกรณ์มหาวิทยาลัย

M.S. (Computer Science) Syracuse University

อาจารย์ พบสิทธิ์ กมลเวชช

วศ.บ. (วิศวกรรมคอมพิวเตอร์) สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

M.S. (Computer and Engineering Management) มหาวิทยาลัยอัสสัมชัญ

ดร. สุขุมาล กิติสิน

B.S. (Computer Science) Indiana University of Pennsylania

M.S. (Computer Science) University of Sounthern California

Ph.D. (Computer Science) University of Southern California

ISBN 974-92372-7-7

จัดพิมพ์โดย มูลนิธิ สอวน.

พิมพ์ครั้งที่ 2 พ.ศ.2548 (2005)

ออกแบบปก หน้ารองปกและหน้านำบท

โดย นายเมธี บำรุงราชหิรัณย์

และ รองศาสตราจารย์สุชาดา ศิริพันธุ์

ศูนย์ Advance Virtual Intelligent Computing (AVIC)

คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

ศิลปกรรม: นายปรีชา ฉัตระเนตร

พิมพ์ที่ บริษัทด่านสุทธาการพิมพ์ จำกัด

แยกสี เพลท บริษัทอีเล็ฟแว่นคัลเลอร์ส จำกัด

สารบัญ

	หน้า
บทที่ 1 แถวลำดับ	1
1.1 บทนำ	1
1.2 แถวลำดับ 1 มิติ	1
1.3 สายอักขระ	2
1.4 แถวลำดับหลายมิติ	3
1.4.1 การประกาศแถวลำดับ 2 มิติ	4
1.4.2 การประกาศแถวลำดับ 3 มิติ	5
1.4.3 การประกาศแถวลำดับ 5 มิติ	5
1.4.4 รูปแบบการประกาศแถวลำดับหลายมิติ	5
บทที่ 2 รายการ	9
2.1 บทนำ	9
2.1.1 การทำงานของรายการแบบลำดับ	9
2.1.2 ฟังก์ชันที่ใช้ดำเนินงานกับรายการ	9
2.2 รายการโยง	13
2.2.1 โครงสร้างของรายการโยง	13
2.2.2 การประกาศรายการโยงในภาษาซี	14
2.2.3 การนำตัวชี้มาใช้งาน	15
2.2.4 การจัดการกับโหนดชนิดตัวชี้	15
2.2.5 การเรียกใช้ตัวชี้ที่ตำแหน่งต่างๆในรายการโยง	16
2.3 การทำงานของรายการโยง	17
บทที่ 3 สแตก	27
3.1 โครงสร้างข้อมูลแบบสแตก	27
3.2 การสร้างสแตกด้วยแถวลำดับ	29
3.3 การสร้างสแตกด้วยรายการโยง	30
3.4 ข้อเปรียบเทียบของประสิทธิภาพของการดำเนินงาน	32
บทที่ 4 คิว	35
4.1 การสร้างคิวด้วยแถวลำดับ	37
4.2 การสร้างคิวด้วยรายการโยง	39
4.3 ข้อเปรียบเทียบของประสิทธิภาพของการดำเนินงาน	42
บทที่ 5 กราฟ	45
5.1 บทนำ	45
5.2 กราฟ	45
5.3 การคำนวณระยะทาง	47

5.4 การแทนเมทริกซ์ประชิด 48
5.5 การหาระยะทางสั้นที่สุด 50
5.6 การแทนกราฟด้วยรายการโยงประชิด 52
5.7 การแวะผ่านกราฟ 54
5.7.1 การแวะผ่านในแนวกว้าง 54
5.7.2 การแวะผ่านแนวลึก 58

บทที่ 6 ต้นไม้ **63**
6.1 ความหมายของต้นไม้ 63
6.2 ประเภทของต้นไม้ 64
6.3 ต้นไม้ทวิภาค 65
6.3.1 ประเภทของต้นไม้ทวิภาค 67
6.3.2 โครงสร้างข้อมูลของต้นไม้ทวิภาค 68
6.3.3 การดำเนินการของต้นไม้ทวิภาค 77
6.3.4 การท่องไปในต้นไม้ทวิภาค 77
6.4 ต้นไม้ค้นหาทวิภาค 80

บทที่ 7 อัลกอริทึมการเรียงลำดับ **91**
7.1 การเรียงลำดับแบบแทรก 91
7.1.1 ตัวอย่างการทำงานของการเรียงลำดับแบบแทรก 93
7.2 การเรียงลำดับแบบเลือก 93
7.2.1 ตัวอย่างการทำงานของการเรียงลำดับแบบเลือก 95
7.3 การเรียงลำดับแบบฟอง 96
7.3.1 ตัวอย่างการเรียงลำดับแบบฟอง 98
7.4 การเรียงลำดับแบบผสาน 99
7.4.1 ตัวอย่างการทำงานของการเรียงลำดับแบบผสาน 102

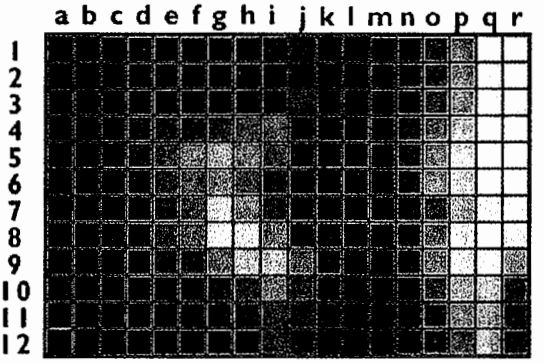
บทที่ 8 การค้นหา **105**
8.1 การค้นหาแบบเรียงลำดับ 105
8.2 การค้นหาแบบทวิภาค 106

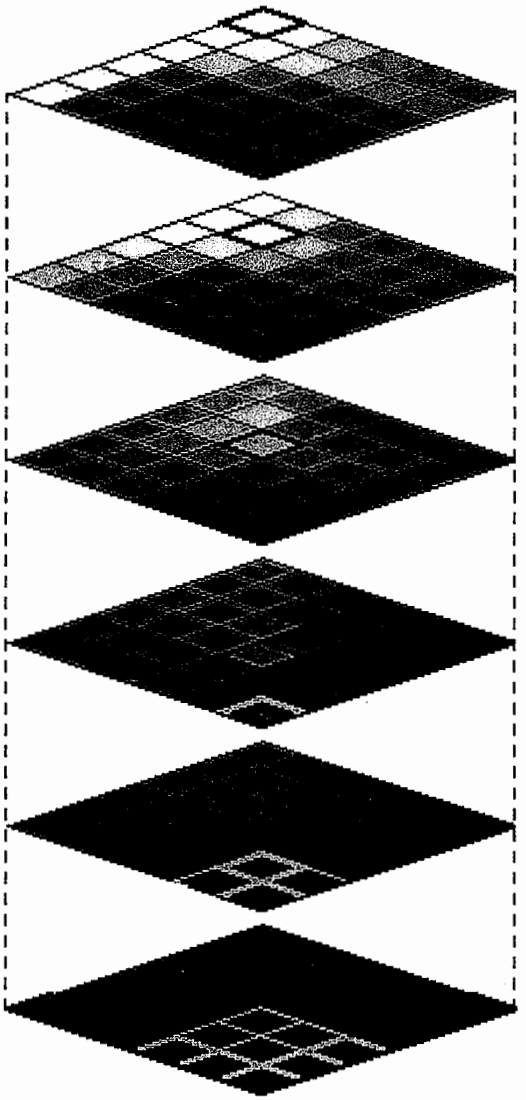
บทที่ 9 การวิเคราะห์อัลกอริทึม **111**
9.1 บทนำ 111
9.2 การวิเคราะห์อัลกอริทึม 112

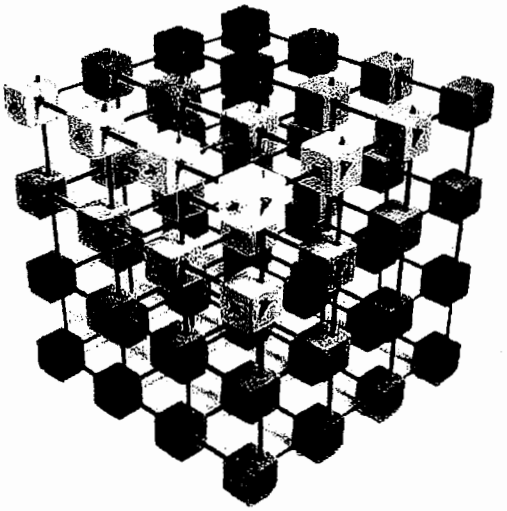
บรรณานุกรม **123**

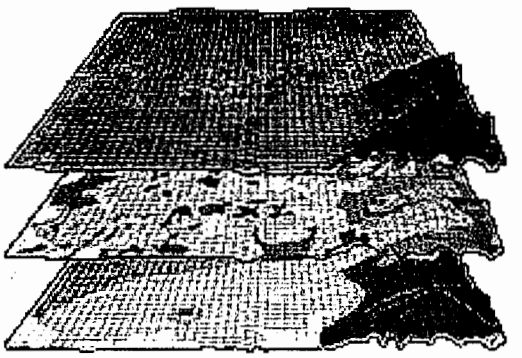
ดัชนี **125**


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

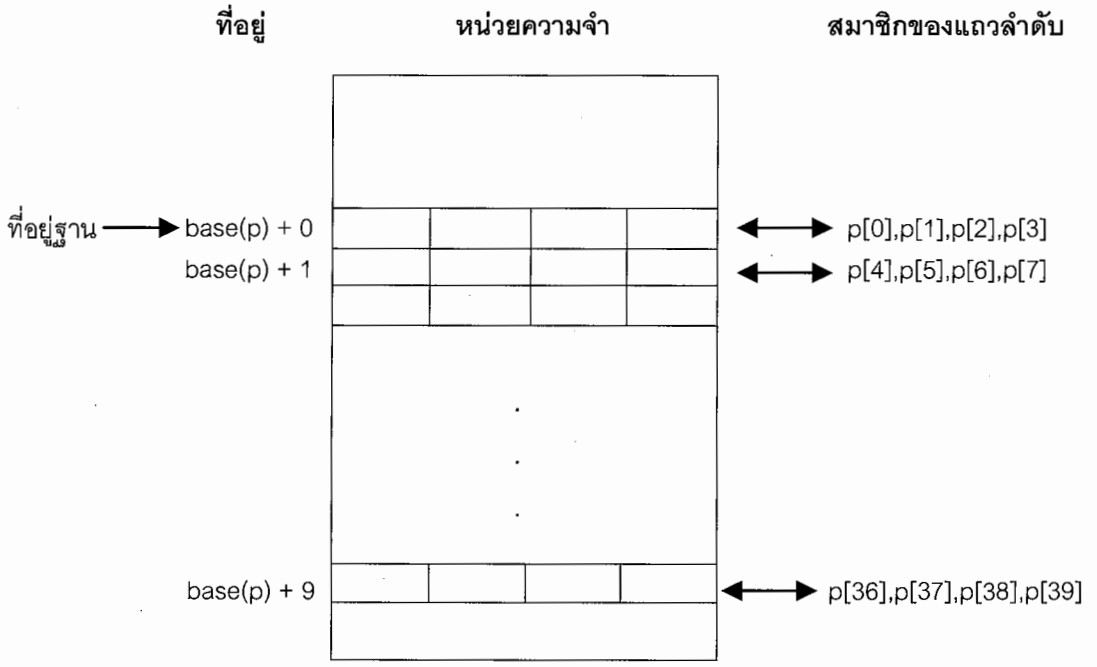

a b c d e f g h i j k l m n o p q r
1 2 3 4 5 6 7 8 9 10 11 12







และมีตำแหน่งที่อยู่ base (p) ตัวอักขระถัดมา p[5] p[6] p[7] และ p[8] เก็บไว้ในเวอร์ดถัดไปที่ตำแหน่ง base (p) + 1 ซึ่งหน่วยความจำ 1 เวอร์ดมีขนาด 4 ไบต์



รูปที่ 1-2 การเก็บสายอักขระในหน่วยความจำ

การเข้าถึงสมาชิกของสายอักขระนั้นจะซับซ้อนกว่าการเข้าถึงสมาชิกของแถวลำดับทั่วไป เพราะการคำนวณที่อยู่ของสมาชิกจะยุ่งยากกว่าแบบทั่วไป นอกจากนั้นอาจจะมีการแปลข้อมูลที่บีบอัดไว้ในหน่วยความจำ คุณสมบัตินี้ทำให้การประมวลผลสายอักขระจะใช้เวลามากกว่า การประมวลผลแถวลำดับธรรมดา

1.4 แถวลำดับหลายมิติ

แถวลำดับหลายมิติ คือ แถวลำดับที่มีดรรชนีมากกว่า 1 ตัว เช่น แถวลำดับ 2 มิติ แถวลำดับ 3 มิติ แถวลำดับ 2 มิติ ใช้ในการประมวลผลข้อมูลอย่างแพร่หลาย เพราะการจัดเก็บข้อมูลส่วนใหญ่อยู่ในรูปแบบของตารางซึ่งเป็นรูปแบบแถวลำดับ 2 มิติ

ตัวอย่าง 1-1 การใช้แถวลำดับ 2 มิติ ในการเก็บข้อมูลและการประมวลผลของยอดขาย สินค้า 4 ชนิด ในแต่ละวัน ของร้านค้า 3 ร้าน ข้อมูลการขายนี้สามารถนำมาบันทึกไว้ในตารางที่ประกอบด้วย 4 แถว 3 สดมภ์ ดังนี้

	ร้านค้า		
สินค้า	0	1	2
0	150	200	70
1	50	0	30
2	120	140	290
3	10	10	20

1.4.1 การประกาศแถวลำดับ 2 มิติ

```
int saletable[4][3];
```

ตัวแปร saletable[2][1] สามารถนำไปใช้เพื่อเข้าถึงจำนวนของการขายของสินค้า 2 (แถว 2) ที่ร้านค้า 1 (สดมภ์ 1) เป็นจำนวน 140 ชิ้น

ในกรณีที่ต้องการเก็บข้อมูลการขายของแต่ละวันในสัปดาห์ ถ้าร้านเปิด 6 วัน คือ วันจันทร์ (Monday) ถึงวันเสาร์ (Saturday) ดังนั้นจึงใช้ตารางทั้งหมด 6 ตาราง

Monday

	ร้านค้า		
สินค้า	0	1	2
0	150	200	70
1	50	0	30
2	120	140	290
3	10	10	20

Tuesday

	ร้านค้า		
สินค้า	0	1	2
0	60	20	90
1	100	50	40
2	150	170	220
3	0	20	30

Saturday

สินค้า	ร้านค้า		
	0	1	2
0	80	50	10
1	70	110	0
2	340	280	170
3	20	10	0

1.4.2 การประกาศแถวลำดับ 3 มิติ

แถวลำดับ 3 มิติ ที่ใช้เก็บข้อมูลการขายในสัปดาห์ทำได้โดย

```
int sales[6][4][3] ;
```

การเข้าถึงสมาชิกในแถวลำดับที่เป็นการขายของ Monday แถวที่ 2 สดมภ์ที่ 1 จะได้มูลค่าการขาย คือ 140 ซึ่งสามารถเขียนแทนได้ด้วย sales[0][2][1]

1.4.3 การประกาศแถวลำดับ 5 มิติ

```
enum make {Toyota, Honda, Datsun, BMW} ;
enum style {Twodoor, Fourdoor, Pickup, Sedan} ;
enum color {Blue, Green, Red, Bronze, Gold} ;
enum year {2000, 2001, 2002, 2003, 2004};
typedef int inventoryarray[make, style, color, year, int];
inventoryarray  inv ;
```

แถวลำดับ inv อาจนำมาใช้ในโปรแกรมสต๊อกสินค้าของบริษัทขายรถยนต์ เช่น

```
inv[Honda][Fourdoor][red][2003][4] = inv[Honda][Fourdoor][red][2003][4] – 1;
```

หมายถึง การที่ผู้จำหน่ายได้ขายรถยนต์ Honda 4 ประตู สีแดง ปี 2003 รหัส 4 ไปจำนวน 1 คัน ทำให้สินค้าในสต๊อกมีปริมาณรถยนต์ลดลง 1 คัน

1.4.4 รูปแบบการประกาศแถวลำดับหลายมิติ

```
elementtype array [indextype1][indextype 2]..[indextypeN] ;
```

indextype คือ ชนิดของดรรชนี

การเก็บแถวลำดับหลายมิติในหน่วยความจำโดยสมมติว่า จำนวนเต็ม 1 ตัวเก็บไว้ในหน่วยความจำเชิงเดียว 1 เซล ดังนั้นแถวลำดับ 2 มิติ ขนาด 3x4 ที่ใช้เก็บจำนวนเต็มจึงมีผลให้เกิดการจองเนื้อที่ในหน่วยความจำ 12 ตำแหน่งเพื่อเก็บสมาชิกของแถวลำดับ ถ้าสมาชิกเหล่านี้เก็บตามลำดับที่ของแถวเป็นหลัก โดย 4 เซลแรกใช้เก็บสมาชิกในแถวแรกทั้งหมดของ m และ 4 เซลต่อไปใช้เก็บสมาชิกในแถวที่ 2 ทั้งหมดต่อไปตามลำดับ

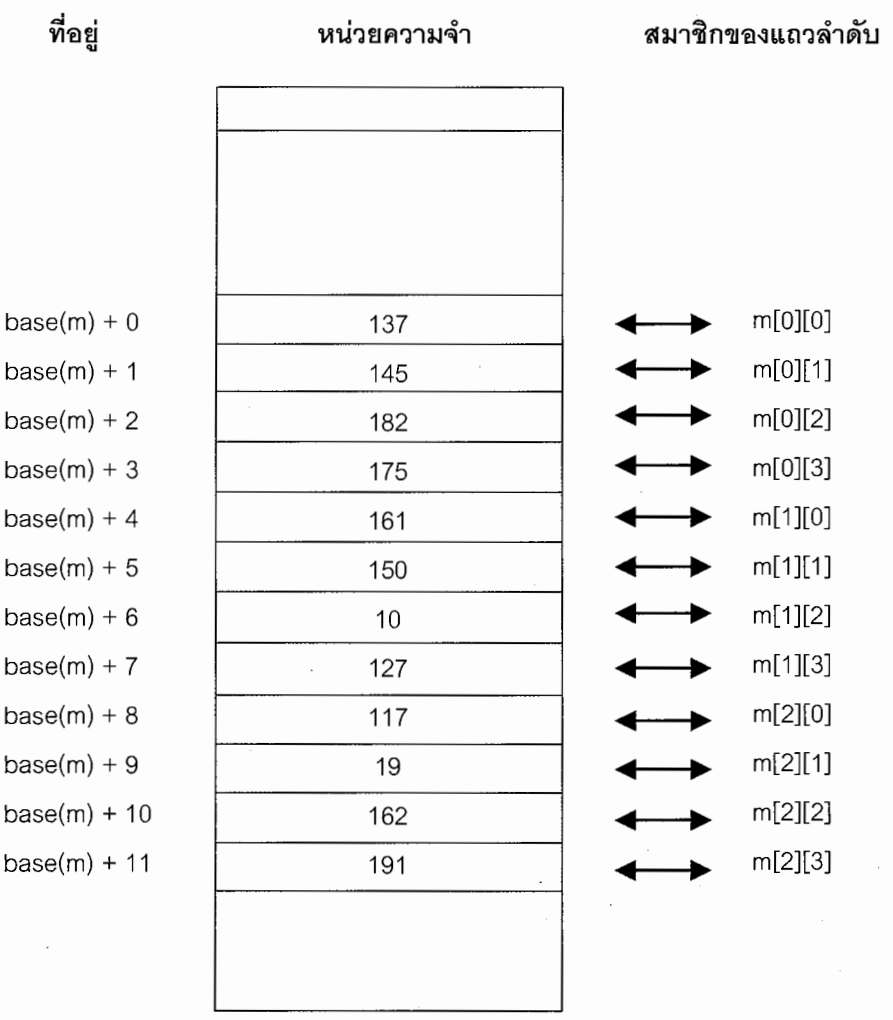
กำหนดให้ m เป็นแถวลำดับ 2 มิติ ที่ใช้เก็บจำนวนเต็ม

สามารถประกาศแถวลำดับ m ดังนี้

```
int m[3][4] ;
```

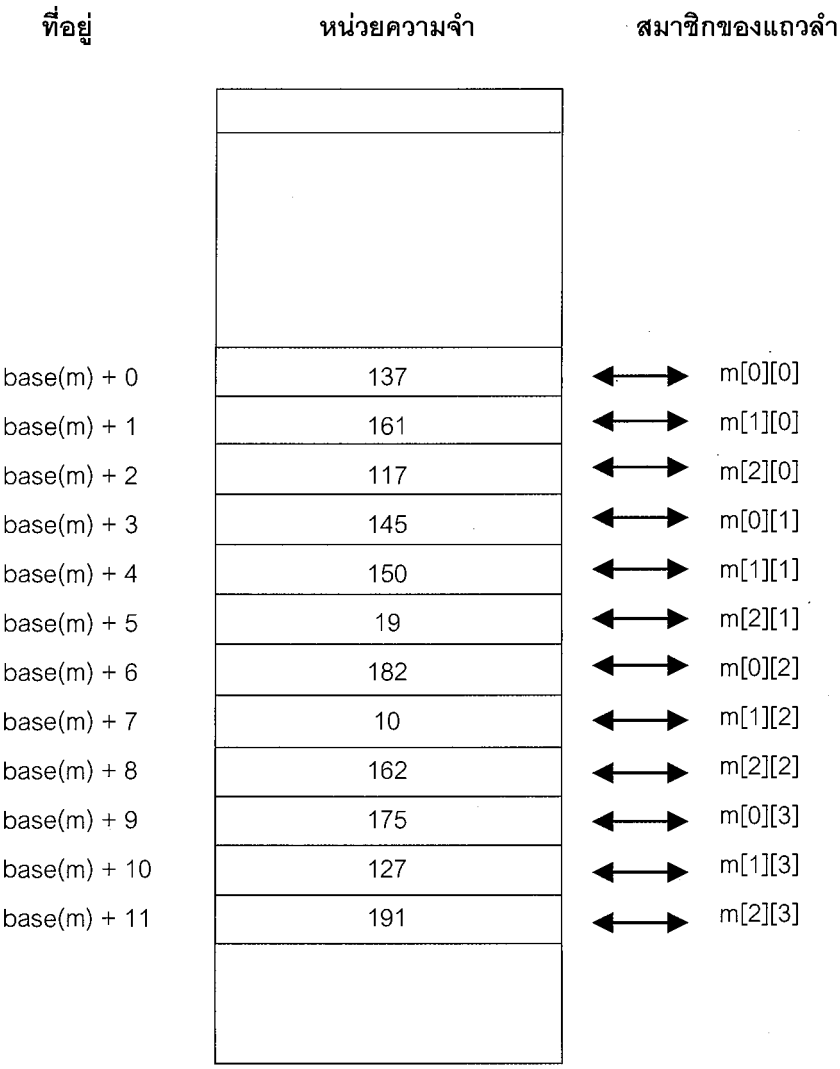
แถวลำดับที่กล่าวมาอาจใช้เก็บข้อมูล ดังนี้

137	145	182	175
161	150	10	127
117	19	162	191



รูปที่ 1-3 แสดงการเก็บข้อมูลในหน่วยความจำแบบแถวหลัก

การเก็บข้อมูลแบบตามลำดับในสดมภ์เป็นหลัก ซึ่งเก็บข้อมูลในสดมภ์แรกไว้ใน 3 เซลแรก และเก็บข้อมูลในสดมภ์ที่ 2 ไว้ใน 3 เซลถัดไป ดังรูปที่ 1-4



รูปที่ 1-4 แสดงการเก็บข้อมูลในหน่วยความจำแบบสดมภ์หลัก

สำหรับการเข้าถึงข้อมูลในแถวลำดับ 2 มิติ แบบสดมภ์นั้น ใช้วิธีการเดียวกับการเข้าถึงข้อมูลของแถวลำดับ 2 มิติ แบบแถวหลัก

แบบฝึกหัดบทที่ 1

1. จากข้อความที่กำหนดให้ดังต่อไปนี้จงบอกว่าข้อใดควรใช้ตัวแปรแถวลำดับในการแก้ปัญหา

    ก. คำนวณผลบวกของตัวเลขอนุกรม

    ข. หาตัวเลขมากสุดอันดับสองจากข้อมูลนำเข้าที่เรียงลำดับ

    ค. อ่านจำนวน 200 จำนวน และให้เรียงลำดับจากมากไปหาน้อย

    ง. อ่านจำนวนเต็ม 200 จำนวน และพิมพ์จำนวนเลขทั้งหมดที่มีค่าอยู่ในช่วงที่กำหนดให้

2. จงเขียนโปรแกรมเพื่อนับจำนวนความถี่ของตัวอักษรภาษาอังกฤษในข้อมูลนำเข้าและเขียนข้อมูลส่งออกเป็นเปอร์เซนต์ซึ่งคำนวณได้จากจำนวนความถี่ของตัวอักษรแต่ละตัวหารด้วยจำนวนตัวอักษรทั้งหมดในข้อมูลนำเข้า

3. จงเขียนโปรแกรมที่สามารถบวกจำนวนเต็มที่มีความยาวสูงสุดขนาด 300 หลัก วิธีการ คือ เก็บจำนวนไว้ในแถวลำดับ (block) ซึ่งสมาชิกแต่ละตัวของแถวลำดับเป็นที่เก็บของตัวเลขของจำนวนเต็ม 3 ตัว เช่น จำนวนเต็ม 179, 534, 672, 198 เก็บไว้ที่ block[1] = 198, block[2] = 672, block[3] = 534, block[4] =179 สำหรับการบวกจำนวนเต็ม 2 จำนวน ทำโดยบวกสมาชิกของแถวลำดับทีละช่อง และ มีการนำตัวทดไปบวกที่ช่องอันดับถัดไป

บทที่ 2
รายการ

2.1 บทนำ

รายการ (list) เป็นโครงสร้างข้อมูลชนิดหนึ่งซึ่งเป็นที่นิยมใช้ในการเขียนโปรแกรมในงานประเภทต่างๆ เพราะการใช้งานรายการทำได้สะดวก โดยทั่วไปแล้วสามารถเพิ่มและลบข้อมูลได้ทุกส่วนของรายการ บทนี้กล่าวถึงรายละเอียดของรายการและการนำรายการไปใช้งาน

2.1.1 การทำงานของรายการแบบลำดับ

เราสามารถพบรายการได้ในชีวิตประจำวัน เช่น รายการของนักศึกษาที่ลงทะเบียนเรียน รายการของสินค้า รายการของพนักงานในบริษัท รายการของที่นั่งโรงภาพยนตร์ ในกรณีของโครงสร้างข้อมูล รายการ คือ ชุดของสมาชิกที่มีขนาดจำกัด ถึงแม้ว่าการทำงานของรายการนั้นแตกต่างกันไปตามลักษณะการประยุกต์ใช้งาน เราสามารถรวบรวมการทำงานพื้นฐานของรายการได้ ดังนี้

1. การสร้างรายการว่าง
2. การทดสอบว่ารายการว่างหรือไม่
3. การท่องไปในรายการ หรือ ส่วนของรายการ โดยมีการเข้าถึงสมาชิก และประมวลผลสมาชิกในแบบตามลำดับ
4. การเพิ่มสมาชิกใหม่ลงในรายการ
5. การลบสมาชิกออกจากรายการ

2.1.2 ฟังก์ชันที่ใช้ดำเนินงานกับรายการ

กำหนดให้รายการประกอบด้วยสมาชิกที่เรียงลำดับ

ฟังก์ชัน createlist

หน้าที่	สร้างรายการว่าง
ผลลัพธ์	รายการว่าง

ฟังก์ชัน emptyList

ข้อมูลนำเข้า	รายการ
หน้าที่	ทดสอบว่ารายการว่าง
ผลลัพธ์	เป็นจริง ถ้ารายการว่าง
	เป็นเท็จ ถ้ารายการไม่ว่าง

ฟังก์ชัน traverse

ข้อมูลนำเข้า	รายการ
หน้าที่	ท่องไปในรายการเพื่อ เข้าถึงและประมวลผลสมาชิกของรายการตามลำดับ
ผลลัพธ์	ขึ้นกับการประมวลผล

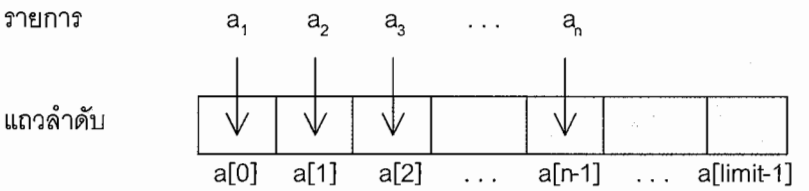
ฟังก์ชัน insert

ข้อมูลนำเข้า	รายการ ข้อมูล และตำแหน่งของรายการ
หน้าที่	เพิ่มข้อมูลลงไปในรายการบริเวณตำแหน่งที่ต้องการ
ผลลัพธ์	รายการที่มีการเปลี่ยนแปลง

ฟังก์ชัน delete

ข้อมูลนำเข้า	รายการและตำแหน่งในรายการ
หน้าที่	ลบสมาชิกในรายการบริเวณตำแหน่งที่ต้องการ
ผลลัพธ์	รายการที่มีการเปลี่ยนแปลง

เราสามารถใช้แถวลำดับเป็นที่เก็บข้อมูลของรายการซึ่งประกอบด้วยสมาชิกที่เรียงลำดับ โดยสมาชิกตัวแรกอยู่ที่ช่องที่ 0 ของแถวลำดับ สมาชิกตัวที่ 2 อยู่ที่ช่องที่ 1 ตามลำดับ

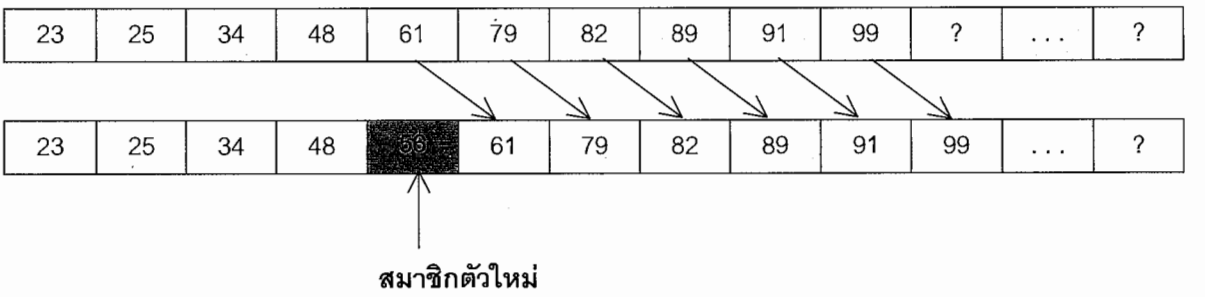


สามารถกำหนดรายการในภาษาซีได้โดยใช้คำสั่งดังนี้

```
# define LIMIT 100                    /* กำหนดให้ LIMIT มีค่าเป็น 100 */
struct listType {
            int size;
            int listElement[LIMIT];
      };
struct listType list;
int emptyList;
list.size = 0 ;                       /* กำหนดให้รายการว่าง */
emptyList = (list.size == 0)          /* ทดสอบว่ารายการว่าง ตัวแปร emptyList จะให้ค่าเป็นจริง
                                         หรือ เท็จ */
/* การเขียนข้อมูลภายในรายการทำได้โดยใช้คำสั่งวนซ้ำ for */
int i;
for (i=0; i <= list.size-1; i++)
      pritnf("%d\n", list.listElement[i]);
```

ตัวอย่าง 2-1 การใส่จำนวนเต็ม 56 เข้าไปในรายการซึ่งเก็บข้อมูลในแถวลำดับ ตำแหน่งที่อยู่ถัดจากสมาชิกที่มีค่า 48 เนื่องจากแถวลำดับเป็นโครงสร้างที่สามารถบรรจุข้อมูลได้มีขนาดจำกัด เช่นกำหนด listElement มีขนาด 100 เมื่อแถวลำดับมีสมาชิกบรรจุอยู่เต็มแล้วจึงไม่สามารถใส่สมาชิกตัวใหม่ลงไปได้ ก่อนใส่สมาชิกลงในแถวจะต้องมีการทดสอบว่าแถวลำดับเต็มหรือไม่ในที่นี้ โดยใช้โหนด listError เป็นตัวบอกว่าแถวลำดับเต็ม

ก่อนเพิ่มสมาชิกลงในส่วนกลางของแถวลำดับนั้น ต้องมีการเลื่อนสมาชิกของแถวลำดับทั้งหมดที่อยู่ตำแหน่งหลังจากสมาชิก 48 ไปในช่องถัดไปเพื่อจะได้มีที่ว่างสำหรับสมาชิกใหม่ของแถวลำดับโดยข้อมูลภายในมีค่า เรียงลำดับ

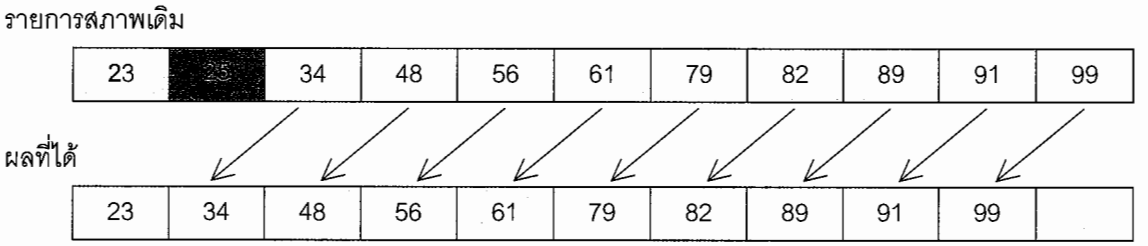


การเพิ่มสมาชิกให้กับรายการ

```
/* เลื่อนสมาชิกของแถวลำดับไปทางขวา 1 ช่อง เพื่อที่จะให้มีช่องว่างสำหรับ item */
/* pos เป็นตำแหน่งข้อมูลตัวสุดท้ายของรายการ */
        for (i = list.size; i > pos; i--)
            listElement[i] = list.listElement[i-1] ;
/* ใส่ item ไปที่ตำแหน่ง pos + 1 และเพิ่มขนาดของรายการ */
        list.listElement[pos] = item ;
        list.size++ ;
```

ประสิทธิภาพการทำงานของฟังก์ชัน insert นี้ขึ้นกับตำแหน่งที่จะเพิ่มสมาชิกใหม่ ถ้ามีการเพิ่มสมาชิกบริเวณท้ายสุดของรายการจะไม่มีการเลื่อนตำแหน่งสมาชิก การดำเนินงานก็จะทำได้เร็วกว่า การเพิ่มสมาชิกตัวใหม่ บริเวณส่วนกลางของรายการ เพราะต้องใช้เวลาในการเลื่อนสมาชิกไปที่ช่องถัดไป

ตัวอย่าง 2-2 การลบ 25 ออกจากรายการ



การลบสมาชิกออกจากรายการ

```
/* เลื่อนสมาชิกของแถวลำดับไปทางซ้ายเพื่อปิดช่องว่าง */
for (i = pos; i < list.size - 1; i--)
     list.listElement[i] = list.listElement[i+1];
/* ลดขนาดของรายการ */
list.size--;
```

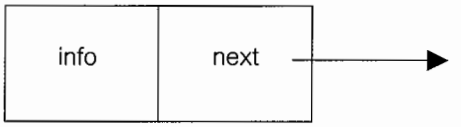
การลบสมาชิกออกจากรายการแบบลำดับมีประสิทธิภาพการทำงานเช่นเดียวกับการเพิ่มสมาชิกในรายการ คือขึ้นกับตำแหน่งของสมาชิกที่ต้องการลบ การลบสมาชิกที่ส่วนท้ายหรือส่วนหัวจะทำได้เร็วกว่า ลบสมาชิกที่ส่วนกลางของรายการ

การทำงานของรายการที่ได้กล่าวไปแล้วเป็นโครงสร้างข้อมูลแบบสถิต (static) ซึ่งมีข้อจำกัดในการทำงาน คือ เมื่อลบหรือเพิ่มสมาชิกบริเวณส่วนต้นหรือกลางของรายการทำให้มีการเลื่อนตำแหน่งของสมาชิกซึ่งทำให้การประมวลผลใช้เวลาในการทำงานมาก จึงมีการพัฒนารายการโยงที่มีโครงสร้างข้อมูลแบบพลวัต (dynamic) ที่สามารถลบและเพิ่มข้อมูลบริเวณใด ๆ ของรายการได้สะดวกและรวดเร็วกว่าแบบสถิต

2.2 รายการโยง (Linked List)

2.2.1 โครงสร้างของรายการโยง

รายการโยง ประกอบด้วยกลุ่มของโหนด ซึ่งแต่ละโหนดประกอบด้วยเขตข้อมูลอย่างน้อย 2 เขต (field) ในเขตข้อมูลแรกเป็นที่เก็บข้อมูลของผู้ใช้ เขตนี้เป็นที่รู้จักในชื่อของ info สำหรับเขตที่สองประกอบด้วยตัวชี้หรือรายการโยง ซึ่งบอกถึงที่อยู่ของโหนดถัดไปในรายการ เขตนี้จึงนิยมตั้งชื่อว่า next สามารถเขียนโหนดของรายการโยงได้ดังรูปที่ 2-1



มูลค่า ตัวชี้บอกที่อยู่ของโหนดถัดไป

รูปที่ 2-1 แสดงโหนดของรายการโยง

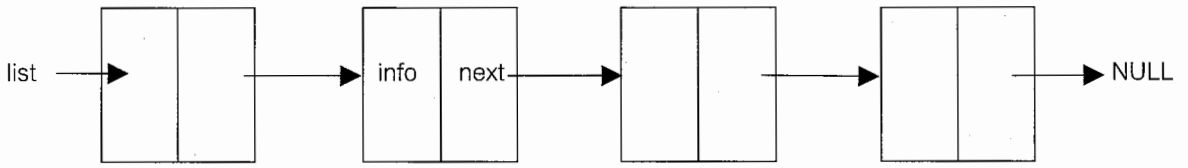
ข้อมูลที่สามารถบรรจุในเขตข้อมูล info อาจจะเป็นจำนวนเต็ม จำนวนจริง ตัวอักขระ สายอักขระ หรือระเบียนที่มีหลายเขต ดังตัวอย่าง 2-3

รายการโยงแบบลำดับประกอบด้วยโหนดที่มีการจัดอันดับเรียงตามมูลค่าภายในของโหนด

ตัวอย่าง 2-3 สมาชิกของรายชื่อนักศึกษาโดยภาษาซีดังนี้

```
struct infotype {
      char studentName[20];
      int studentID;
      float gpa;
      char streetAddr[20];
      char state[20];
};
```

ในรายการโยงส่วน info ของโหนดเป็นระเบียน การเรียงลำดับของโหนดสามารถพิจารณาจากเขตในระเบียนรายชื่อของนักศึกษา และสามารถจัดลำดับรายชื่อโดยใช้มูลค่าภายในซึ่งเป็นเขตรหัสนักศึกษา หรือ ชื่อนักศึกษาเป็นหลัก การนำรายการโยงไปใช้ในงานต่าง ๆ นี้ จะกล่าวถึงต่อไป



รูปที่ 2-2 รายการโยงที่ประกอบด้วย 4 โหนด

list เป็นตัวชี้ที่บอกอยู่ของโหนดแรกในรายการโยง ซึ่งแต่ละโหนดประกอบด้วย 2 เขตข้อมูลดังรูปที่ 2-2

บริเวณปลายสุดของรายการโยงในเขต next ประกอบด้วยข้อมูลที่บอกการสิ้นสุดของรายการ ข้อมูลนี้ เรียกว่า NULL ซึ่งปราศจากเลขที่อยู่

2.2.2 การประกาศรายการโยงในภาษาซี

การประกาศรายการโยงรายการทำได้ โดยกำหนดโหนดขนิดตัวชี้ซึ่งเป็นการจองเนื้อที่ในหน่วยความจำแบบพลวัต แต่การประกาศแถวลำดับและระเบียน ซึ่งเป็นโหนดสถิตมีการจองเนื้อที่ในหน่วยความจำเท่ากับขนาดของโหนดในขั้นตอนแปลชุดคำสั่ง และเนื้อที่นี้ไม่สามารถนำมาใช้สำหรับโหนดอื่น ๆ การใช้แถวลำดับนั้นนักเขียนโปรแกรมจะประมาณขนาดสูงสุดของข้อมูลที่จะนำมาใช้ และประกาศขนาดของแถวลำดับ ในการดำเนินงานแถวลำดับที่ไม่มีข้อมูลบรรจุอยู่จะใช้เนื้อที่ในหน่วยความจำเท่ากับแถวลำดับที่มีข้อมูลบรรจุอยู่เต็ม การจองเนื้อที่ของแถวลำดับจะกระทำก่อนการนำโหนดสถิตนี้มาใช้ ถ้านักเขียนโปรแกรมกำหนดแถวลำดับให้มีขนาดใหญ่เกินไป อาจทำให้เกิดการสูญเปล่าของเนื้อที่ในหน่วยความจำ แต่ถ้าเขากำหนดแถวลำดับเล็กขนาดเกินไป เนื้อที่ของแถวลำดับอาจจะไม่พอในการรองรับข้อมูลใหญ่ จึงต้องปรับปรุงโปรแกรมให้สามารถรองรับการขยายขนาดของแถวลำดับเพื่อใช้ในการดำเนินงาน

ภาษาซีมีกระบวนการในการสร้างโหนดแบบพลวัต ซึ่งโหนดจะใช้เนื้อที่ในหน่วยความจำในขั้นตอนของการปฏิบัติงานของโปรแกรม และโหนดถูกสร้างขึ้นหรือทำลายได้ตลอดเวลาในขณะกระทำการของโปรแกรม ดังนั้นโหนดใช้เนื้อที่ในหน่วยความจำเมื่อมีการดำเนินงานของโหนดเท่านั้น

```
/* กำหนดโครงสร้างที่ชี้ไปตัวเอง */
struct listNode{
      char info;
      struct listNode *nextPtr;
};
typedef struct listNode LISTNODE;
typedef LISTNODE *LISTNODEPTR;
```

บทที่ 1
แถวลำดับ

1.1 บทนำ

แถวลำดับเป็นโครงสร้างข้อมูลที่พบในภาษาคอมพิวเตอร์ระดับสูงทุกภาษา โครงสร้างชนิดนี้ใช้เก็บข้อมูลชนิดเดียวกันที่มีขอบเขตจำกัดและมีขนาดคงที่ การทำงานพื้นฐานของแถวลำดับ คือ การเข้าถึงตำแหน่งแต่ละตำแหน่งในแถวลำดับได้โดยตรง สมาชิกในแถวลำดับ เรียกว่า คอมโปเนนต์ (component) และสมาชิกเหล่านี้เก็บในแถวลำดับแบบอันดับ คือ สมาชิกตัวที่ 1 สมาชิกตัวที่ 2 , ... ,n สมาชิกทั้งหมดในแถวลำดับต้องเป็นชนิดเดียวกัน ดังนั้นสามารถกำหนดแถวลำดับของจำนวนจริง แถวลำดับของจำนวนเต็ม

การเข้าถึงแถวลำดับแบบตรง หมายถึง การเข้าถึงสมาชิกของแถวลำดับ โดยการกำหนดตำแหน่งของสมาชิกในแถวลำดับ ดังนั้นเวลาที่ใช้ในการเข้าถึงสมาชิกแต่ละตัวในแถวลำดับจะมีจำนวนเท่ากัน ไม่ขึ้นอยู่กับตำแหน่งของสมาชิก เช่น แถวลำดับที่มีสมาชิก 200 ตัว การเข้าถึงสมาชิกตัวที่ 75 จะใช้เวลาเท่ากับการเข้าถึงสมาชิกตัวที่ 5

ในภาษาคอมพิวเตอร์ระดับสูง แถวลำดับแทนด้วยตัวแปรซึ่งมีสมาชิกอยู่ภายใน การเรียกใช้สมาชิกภายในแถวลำดับเรียกโดยระบุชื่อแถวลำดับและดรรชนีของแถวลำดับ ซึ่งเป็นการบอกตำแหน่งของสมาชิกตัวนั้น แถวลำดับที่มีดรรชนีเพียง 1 ตัว เรียกว่า แถวลำดับ 1 มิติ (one dimensional array) แถวลำดับที่มีดรรชนีหลายตัว เรียกว่า แถวลำดับหลายมิติ (multidimensional array)

1.2 แถวลำดับ 1 มิติ

การประกาศแถวลำดับมี 3 ส่วน คือ ชนิดของดรรชนี (index type) และชนิดของสมาชิก

รูปแบบในภาษาซี

ชนิดของสมาชิก แถวลำดับ[ดรรชนี]

ชนิดของดรรชนี ต้องเป็นข้อมูลชนิดอันดับ (ordinal) เท่านั้น

ชนิดของสมาชิก เป็นชนิดข้อมูลใด ๆ

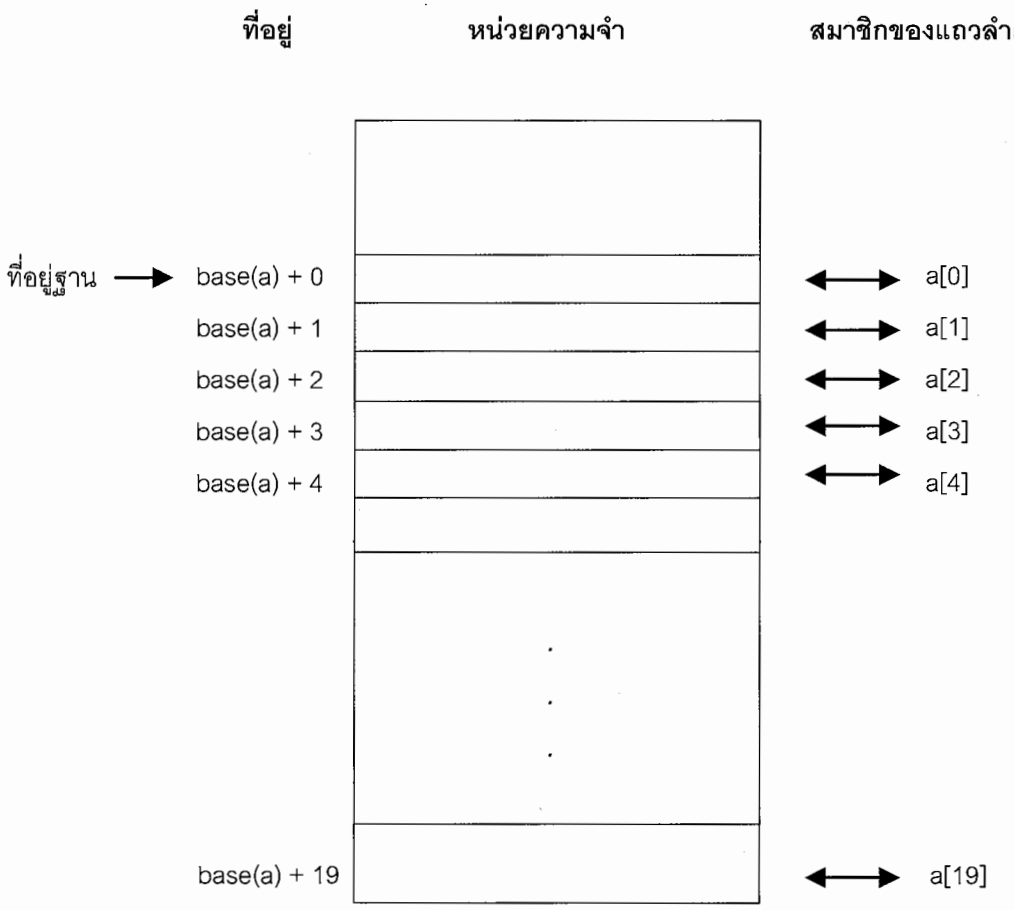
เช่น

```
int a[20];
```

ในการกำหนดครั้งนี้ ตัวแปลภาษาจองเนื้อที่ในหน่วยความจำซึ่งมีขนาดใหญ่พอที่จะเก็บสมาชิกของแถวลำดับทั้งแถว ที่อยู่ของหน่วยความจำที่ใช้เก็บสมาชิกตัวแรก เรียกว่า ที่อยู่ฐาน (base address) ของแถวลำดับซึ่งแทนด้วย base(a) และที่อยู่ของสมาชิกแถวลำดับตัวอื่นๆ สามารถคำนวณได้จากที่อยู่ฐาน เช่น ถ้าเก็บจำนวนเต็มไว้หน่วยความจำโดยเก็บจำนวนเต็ม 1 จำนวนไว้ใน 1 เซล เมื่อเริ่มเก็บ a[0] ไว้ที่ตำแหน่ง base(a) ดังนั้นจะเก็บ a[5] ไว้ที่ base(a) + 4

สามารถหาตำแหน่งที่เก็บ a[i] ได้จากสูตร

base(a) + (i – 1)



รูปที่ 1-1 การเก็บแถวลำดับในหน่วยความจำ

1.3 สายอักขระ (String)

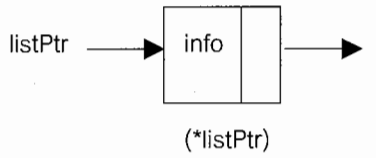
ในภาษาซีใช้ array of char แทนการเก็บสายอักขระซึ่งเป็นที่เก็บชุดตัวอักขระ สามารถกำหนดแถวลำดับ p ของสายอักขระ ดังนี้

```
char p[40];
```

การกำหนด p เป็นแถวลำดับอัดแน่นนั้น ทำให้ตัวแปลภาษานำตัวอักขระหลายตัวไปเก็บไว้ในหน่วยความจำในเวอร์ดเดียวกัน เช่น เวอร์ดมีขนาด 4 ไบต์ ตัวอักขระ 4 ตัว p[1] p[2] p[3] และ p[4] เก็บไว้ในเวอร์ดแรกของหน่วยความจำ

2.2.3 การนำตัวชี้มาใช้งาน

การเขียนชื่อตัวชี้พร้อมสัญลักษณ์ * และตามด้วยชื่อโหนดจะแทนข้อมูลซึ่งตัวชี้จะชี้ไป เช่น จากการประกาศชนิดตัวชี้ของ listNode การเขียน nextPtr อ้างถึงที่อยู่ของโหนดหนึ่ง ในรายการที่ตัวชี้ nextPtr ชี้ไปถึง อย่างไรก็ตาม listPtr นั้นแตกต่าง จาก (*listPtr) เพราะ listPtr คือตัวชี้ที่อ้างถึงเลขที่อยู่ แต่ (*listPtr) คือ ข้อมูลที่ถูกชี้ (info) ดังรูปที่ 2-3



รูปที่ 2-3 แสดงรายการ และระเบียนของรายการ

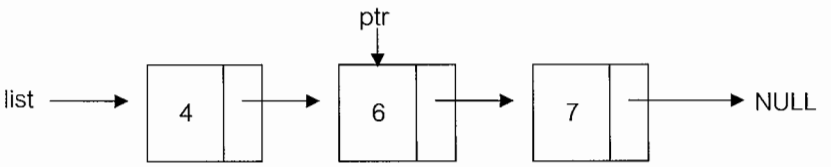
นอกจากการชี้ไปที่ระเบียนแล้ว สามารถชี้ไปที่เขตต่าง ๆ ของระเบียน ได้โดยใช้ตัวชี้
ดังรูปแบบ :

ชื่อตัวชี้ -> ชื่อเขต

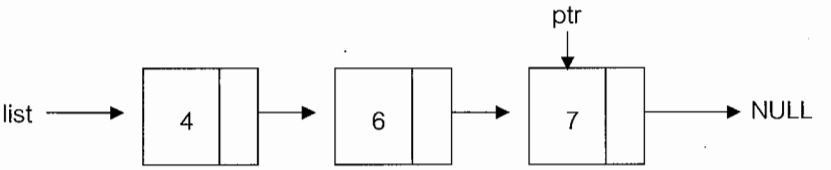
เช่น printf("%c\n", (*listPtr)->info); เป็นคำสั่งให้พิมพ์ค่าภายในของเขตข้อมูล info อยู่ในโหนดที่ชี้โดย listPtr จึงสามารถเห็นถึงความแตกต่างของ *listPtr ซึ่งหมายถึงทั้งระเบียน ขณะที่ (*listPtr)->info คือเขตข้อมูลของระเบียนชื่อ info

2.2.4 การจัดการกับโหนดชนิดตัวชี้

การทำงานของตัวชี้มีหลายวิธีดังนี้
กำหนดให้ ptr และ ptr -> nextPtr เป็นตัวชี้ชนิด Pointer

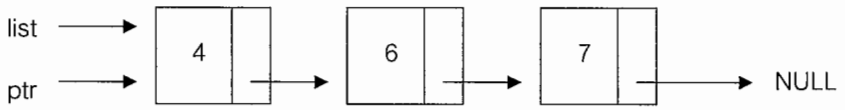


(a) list และ ptr เป็นตัวชี้ โดย list ชี้ไปที่โหนดแรก และ ptr ชี้ไปที่โหนดที่สอง

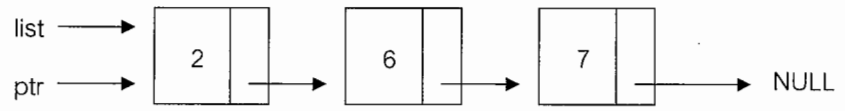


(b) การสั่งให้ ptr ชี้ไปที่โหนดถัดไป โดยใช้คำสั่ง ptr = ptr -> nextPtr ;

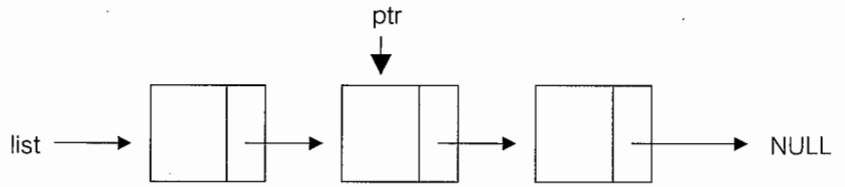
รูปที่ 2-4 การดำเนินงานของตัวชี้



(c) การสั่งให้ ptr ชี้ไปที่โหนดแรกของรายการ โดยใช้คำสั่ง ptr = list;



(d) ให้มูลค่าของเขต Info ในโหนดแรกของรายการมีค่าเท่ากับ 2 โดยใช้คำสั่ง list -> info = 2;

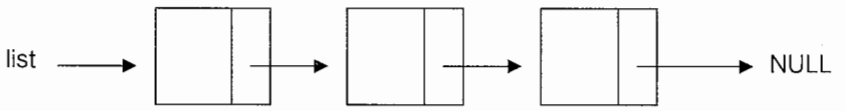


(e) ให้ ptr ชี้ไปที่โหนดที่สองในรายการ โดยใช้คำสั่ง ptr = list->nextPtr

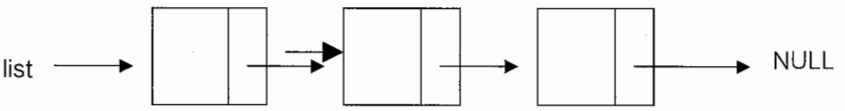
list ชี้ไปที่โหนดแรก สำหรับ list->next เป็นตัวชี้ที่ชี้ไปที่โหนดที่สอง ดังนั้น ptr จึงรับค่าภายในตัวชี้ซึ่งเป็นเลขที่อยู่โหนดที่ 2 ของ list->next

รูปที่ 2-4 การดำเนินงานของตัวชี้ (ต่อ)

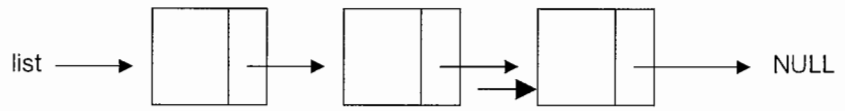
2.2.5 การเรียกใช้ตัวชี้ที่ตำแหน่งต่าง ๆในรายการโยง



(a) list ชี้ไปที่โหนดแรกของรายการ



(b) list -> nextPtr ชี้ไปที่โหนดที่ 2 ของรายการ



(c) list -> nextPtr -> nextPtr ชี้ไปที่โหนดที่ 3 ของรายการ

รูปที่ 2-5 แสดงการเรียกใช้ตัวชี้ด้วยวิธีต่างกัน

2.3 การทำงานของรายการโยง

ในการค้นหา การอ่านและ การพิมพ์ข้อมูลจากรายการโยง สามารถกระทำได้โดยวิธีท่องไปในรายการ วิธีการนี้เริ่มต้นจากกำหนด ptr ที่จุดเริ่มต้นของรายการด้วยข้อความสั่ง

ptr = list

ต่อมาเลื่อนตัวชี้ไปที่โหนดถัดไปโดย ข้อความสั่ง

ptr = ptr -> nextPtr

กระบวนการนี้กระทำซ้ำไปเรื่อย ๆ จนกระทั่งถึงโหนดสุดท้ายของรายการ ผู้ใช้ต้องหาจุดจบของรายการให้พบ โหนดสุดท้ายของรายการมีมูลค่าพิเศษในเขต next เพื่อบอกว่า ไม่มีโหนดอื่นตามมา ดังนั้น next จึงมีค่าภายในเป็น NULL ซึ่งสำหรับโหนดชนิดตัวชี้ NULL ไม่ชี้ไปที่ตำแหน่งใดทั้งสิ้น จึงใช้ NULL ในการทดสอบเงื่อนไขการจบรายการ

```
/* List ADT Type Definitions*/
typedef struct node *ptrtoNode;
typedef ptrtoNode listPtr;
typedef ptrtoNode position;

struct node
{
      elementType element;
      int count;
      position next;
};
```

ตัวอย่าง 2-4 การพิมพ์มูลค่าภายในของรายการโยง

```
void printList (listPtr list)
{
      position p;
      p = list;
      while (p!=NULL)
      {
            printf ("%d\n", p->element);
            p = p-> next;
      }                    /* printList */
```

การสร้างรายการโยง

ตัวชี้ของโหนดสุดท้ายของรายการมีค่าเป็น NULL ดังนั้นจึงเป็นการง่ายที่จะสร้างรายการว่างขึ้นมาใหม่โดยกำหนดค่าเริ่มต้นให้เป็น NULL ดังฟังก์ชันในตัวอย่าง 2-5

ตัวอย่าง 2-5 การสร้างรายการว่าง

```
void createList()
{
      listPtr list;    /* กำหนดให้ list เป็นตัวชี้ชนิด listPtr */
      list = malloc(sizeof(struct node)); /* จองเนื้อที่ในหน่วยความจำ
                                             ให้ตัวชี้ list */
      if (list)
      {
            list -> element = 0;
            list -> count = 0;
            list ->next = NULL;
      }
}                                          /* createList */
```

การทดสอบว่ารายการว่างหรือไม่โดยใช้ฟังก์ชัน isEmpty

```
int isEmpty(listPtr list)
{
      return (list->count==0);
}
```

ฟังก์ชันนี้ทดสอบว่ารายการว่างหรือไม่ โดยนำตัวชี้ list->count มาเปรียบเทียบกับค่า 0 ถ้า list->count มีค่าเป็น 0 แสดงว่ารายการนี้ว่าง ฟังก์ชันให้ค่า 1 เมื่อรายการว่าง และให้ค่าเป็น 0 เมื่อไม่ว่าง

การสร้างและการทำลายโหนด

ในการทำงานเกี่ยวกับแถวลำดับหรือระเบียน เมื่อผู้ใช้ประกาศชื่อโหนดในโปรแกรมแล้วนำโปรแกรมไปแปลชุดคำสั่ง เครื่องคอมพิวเตอร์กำหนดเนื้อที่ในหน่วยความจำให้โหนด แต่สำหรับโหนดตัวชี้ซึ่งทำงานแบบพลวัต ภาษาซีมีฟังก์ชัน malloc เพื่อจัดเนื้อที่ในหน่วยความจำสำหรับโหนดชนิดนี้ ฟังก์ชัน malloc ต้องการพารามิเตอร์ 1 ตัวที่เป็นจำนวนไบต์ที่ต้องการจองเนื้อที่

รูปแบบ : malloc(nbyte);

ตัวอย่าง malloc(sizeof(x)); เมื่อ x เป็นชื่อโหนด

เมื่อผู้ใช้ไม่ต้องการใช้ตัวแปรนั้นอีกต่อไปสามารถยกเลิกเนื้อที่สำหรับตัวแปรนั้นเพื่อให้ส่วนอื่นนำไปใช้ได้ฟังก์ชันยกเลิกเนื้อที่นี้คือ free

รูปแบบ : free(ptr);

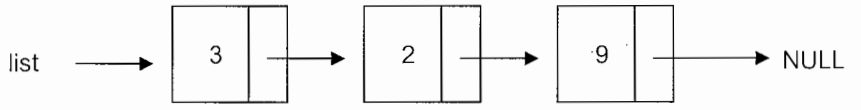
ptr คือโหนดชนิดตัวชี้ซึ่งชี้ไปยังหน่วยความจำที่ไม่ได้ใช้งาน เมื่อได้ยกเลิกตัวชี้ ptr ไปแล้วไม่สามารถนำ ptr กลับมาใช้เป็นโหนดได้อีก เช่น

ptr := list คำสั่งนี้ไม่ถูกต้อง เมื่อเขียนหลังคำสั่ง free(ptr)

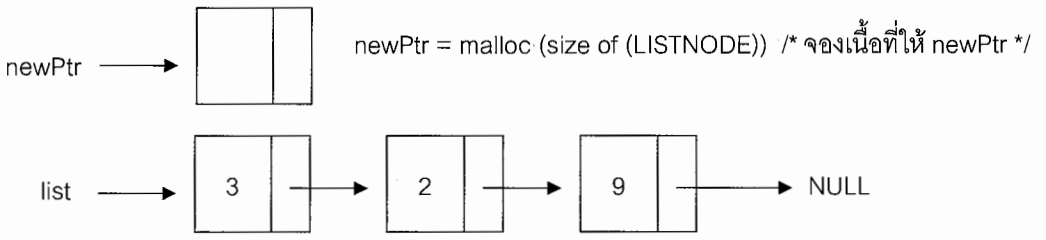
การเพิ่มโหนดในรายการ

อัลกอริทึมในการเพิ่มโหนดนั้นมีหลายวิธีขึ้นกับตำแหน่งในการใส่โหนดในรายการ

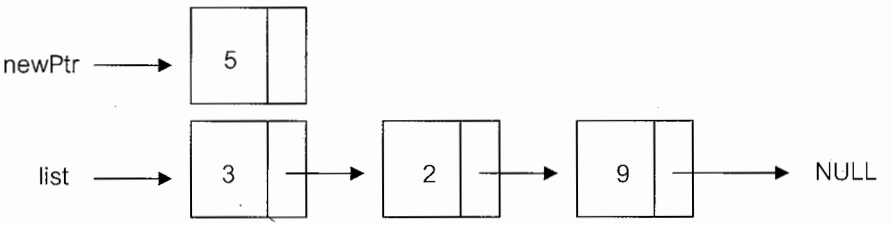
กรณีที่ 1 การใส่โหนดที่จุดเริ่มต้นของรายการซึ่งเขียนอัลกอริทึมได้



(a) รายการในสภาพเริ่มต้นโดย list ชี้ไปที่ โหนดแรก

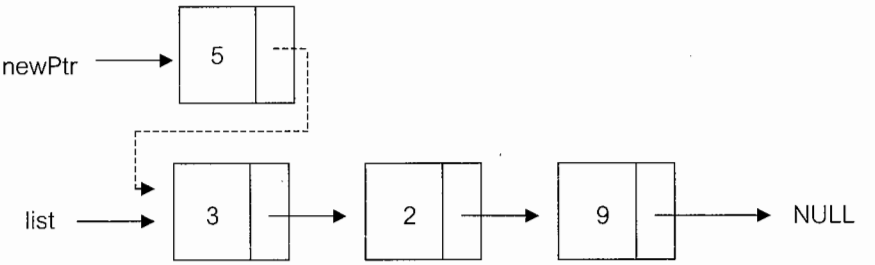


(b) createNode(newPtr) /* ให้ newPtr ชี้ไปที่โหนดใหม่ */

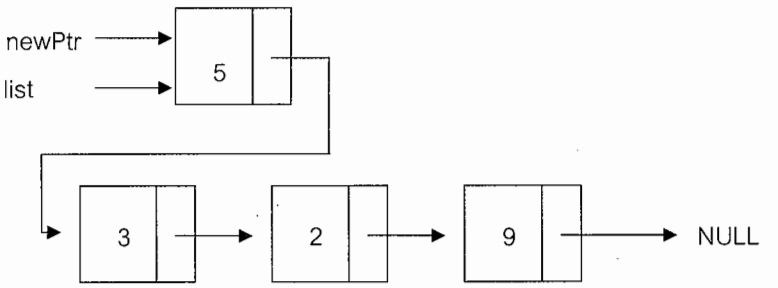


(c) newPtr -> info = 5 /* ให้เขต info มีค่าภายในเป็น 5 */

รูปที่ 2-6 แสดงการเพิ่มโหนดเป็นสมาชิกตัวแรกของรายการโยง



(d) newPtr -> nextPtr = list /*ให้เขต next ชี้ไปที่ตำแหน่งเดียวกับ list*/

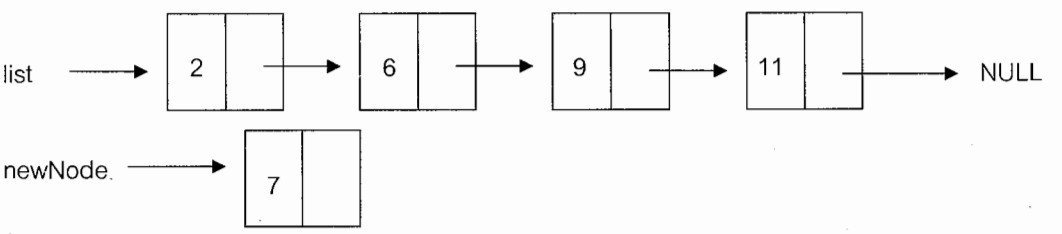


(e) list = newPtr /*ให้ newPtr ชี้ไปที่เดียวกับ list*/

รูปที่ 2-6 แสดงการเพิ่มโหนดเป็นสมาชิกตัวแรกของรายการโยง (ต่อ)

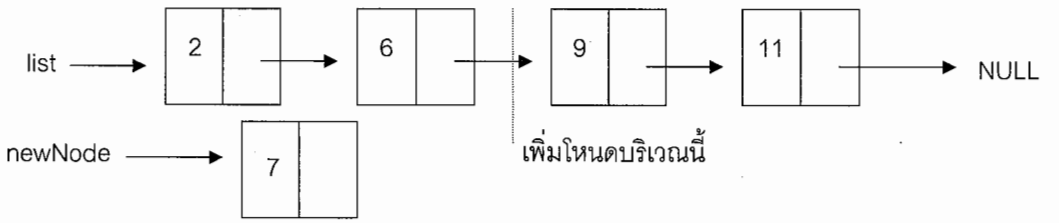
รูปที่ 2-6 (a) แสดงถึงรายการของโหนดที่เก็บเลขจำนวนเต็ม เมื่อต้องการเพิ่มโหนดใหม่ซึ่งมีข้อมูลภายในเป็น 5 โดยจะใส่โหนดนี้ที่ ส่วนต้นของรายการ การทำงานขั้นแรกคือ ใช้ฟังก์ชัน createNode ทำหน้าที่จองเนื้อที่ให้โหนดใหม่ ซึ่งจะประกอบด้วยตัวชี้ที่ชี้ไปยังโหนดว่าง รูปที่ 2-6 (b) เมื่อได้โหนด ว่างแล้วจึงใส่มูลค่า 5 ลงในเขต info รูปที่ 2-6 (c) และใส่โหนดลงในรายการ โดยการกำหนดค่าในเขต next รูปที่ 2-6 (d และ e)

กรณีที่ 2 การแทรกโหนดลงในรายการโยงที่สมาชิกมีค่าเรียงลำดับ

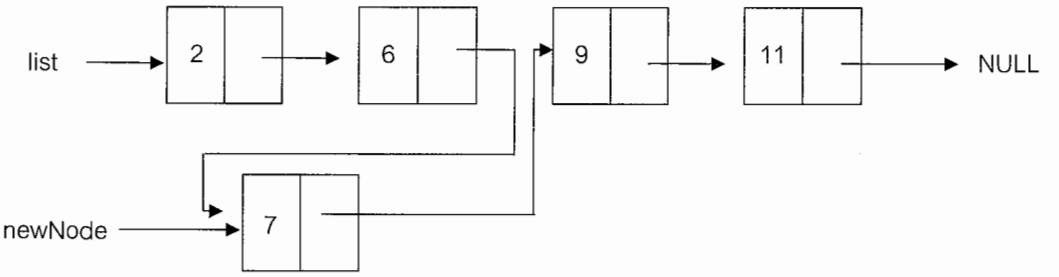


(a) สร้างโหนดใหม่และใส่มูลค่าในโหนดใหม่

รูปที่ 2-7 การเพิ่มโหนดลงในรายการโยงแบบลำดับ



(b) ค้นหาตำแหน่งในการเพิ่มโหนด



(c) เชื่อมต่อตัวชี้

รูปที่ 2-7 การเพิ่มโหนดลงในรายการโยงแบบลำดับ (ต่อ)

ในรูปที่ 2-7 การทำงานขั้นตอนแรกคือ การสร้างโหนดที่ประกอบด้วยข้อมูลที่จะใส่ในรายการโยง ขั้นตอนต่อมาคือใส่โหนดใหม่ (new node) ลงในตำแหน่งที่ถูกต้องในรายการโยง การทำงานขั้นตอนนี้เริ่มจากการเปรียบเทียบมูลค่าของโหนดใหม่ที่จะใส่กับค่าภายในของโหนดที่อยู่ในรายการโยงแบบลำดับ จนกระทั่งพบตำแหน่งที่จะใส่มูลค่าใหม่ การดำเนินงานทำได้โดยใช้คำสั่งวนซ้ำ while เมื่อค้นหาตำแหน่งที่ถูกต้อง ptr จะชี้ไปที่โหนดถัดจากโหนดใหม่

อัลกอริทึมในการเพิ่มสมาชิกของรายการโยงแบบลำดับ

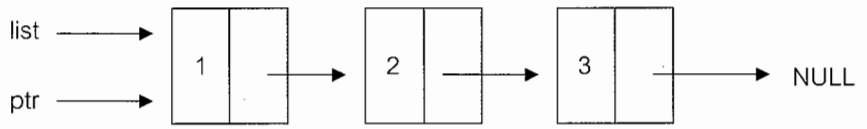
```
if list is empty                              /* เมื่อรายการว่าง */
      then list = pointer to new node         /* ใส่ค่าใหม่ลงในรายการ */
      else                                    /* รายการมีสมาชิกอยู่เดิม */
            if  new Value < list->info        /* ค่าที่ใส่น้อยกว่าโหนดแรก */
                  then
                        Change pointers to insert node(newNode)
                                              /* ให้โหนดใหม่เป็นโหนดแรก */
                        As first node in list
                  else                        /* ดำเนินการหาตำแหน่งที่จะใส่โหนด */
                        placeFound = false
                        ptr = list
```

```
while nextPtr<> NULL and not placeFound do /* ท่องไปในรายการ
                                    โดยดำเนินการเปรียบเทียบค่าที่จะใส่กับค่าภายในโหนด */
      if newValue >= (nextPtr->info)
            then advanceptr    /*เลื่อนตัวชี้ไปอีก 1 ค่า*/
      else placeFound = true
Change the pointers to insert node(newNode)
```

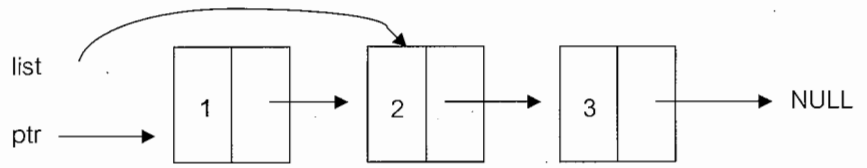
การลบโหนดออกจากรายการโยง

การนำโหนดออกจากรายการโยงสามารถทำได้หลายกรณี

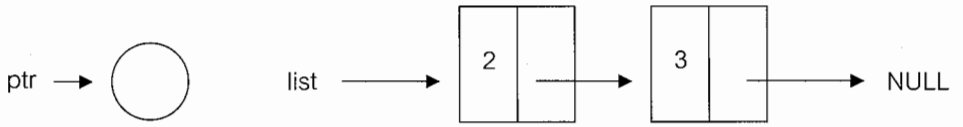
กรณีที่ 1 นำโหนดแรกออกจากรายการโยง มีดังนี้



(a) ptr = list



(b) list = list->next



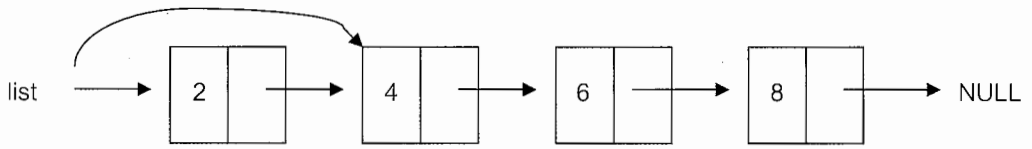
(c) free(ptr)

รูปที่ 2-8 แสดงการลบโหนดแรกของรายการโยง

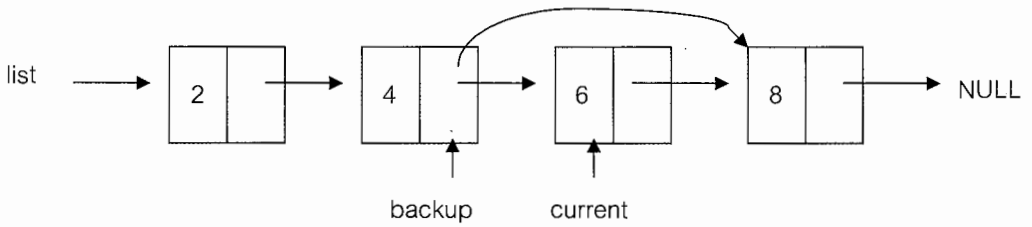
การลบโหนดตัวแรกออกจากรายการทำได้โดยให้ตัวชี้ list (เป็นตัวชี้ที่ชี้ไปที่โหนดแรกของรายการ)ได้รับค่าภายในของเขต next ซึ่งเป็นที่อยู่ของโหนดถัดไป ต่อมาจึงมีการใช้ฟังก์ชัน free ซึ่งเป็นการยกเลิกการใช้งานของโหนด(ptr)

กรณีที่ 2 นำโหนดออกจากรายการโยงรายการแบบลำดับ มีอัลกอริทึมดังนี้

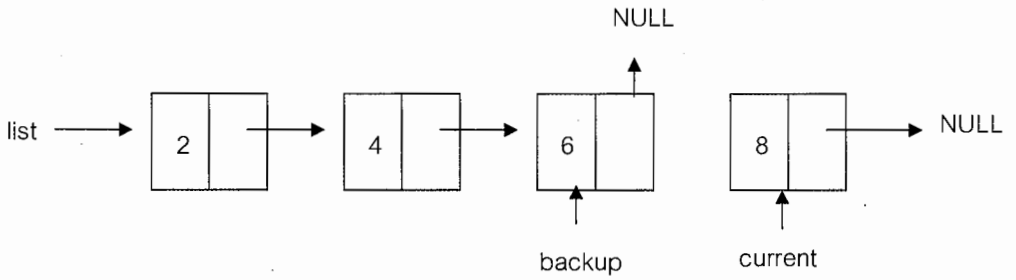
```
current = list      /* กำหนดค่าเริ่มต้นของตัวชี้ */
backup = NULL
/* ค้นหาบัพที่จะลบ */
while current-> info <> deleteVal do Advance the pointers
      backup = current
      current = current->next
/* ตรวจสอบว่าจะลบโหนดแรก */
if  backup = NULL
      then  list = list->next
      else  backup-> next = current-> next
Free the node(current)
```



(a) ลบโหนดแรกออกจากรายการโยง



(b) ลบโหนดที่มีค่าภายในเป็น 6



(c) ลบโหนดที่มีค่าภายในเป็น 8 ซึ่งเป็นโหนดสุดท้ายออกจากรายการโยง

รูปที่ 2-9 แสดงวิธีการลบโหนดที่ตำแหน่งต่างๆ ของรายการโยง

รูปที่ 2-9(a) แสดงการลบโหนดแรกออกจากรายการโยง รูปที่ 2-9(b) แสดงการลบโหนดที่มีค่าภายใน 6 ออกจากรายการโยงโดยให้ current ซึ่งเป็นตัวชี้ที่โหนดที่ต้องการลบออกจากรายการและให้ backup ชี้ไปโหนดก่อนหน้าของ current และเซต next ของ backup ชี้ไปที่เดียวกันกับเซต next ของ current รูปที่ 2-9(c) แสดงการลบโหนดสุดท้ายออก จากรายการ ซึ่งทำได้โดยเปลี่ยนตัวชี้ของโหนดรองสุดท้าย ให้ชี้ไปที่ NULL ทำให้โหนดสุดท้ายถูกตัดทิ้ง

แบบฝึกหัดบทที่ 2

1. การเก็บข้อมูลไว้ในรายการแบบเรียงลำดับโดยที่การบรรจุข้อมูลไม่จำเป็นต้องมีข้อมูลครบทุกช่องจึงอาจจะมีช่องว่างระหว่างข้อมูลในรายการ เมื่อมีการลบสมาชิกออกจากแถวลำดับจะมีการใส่ค่าพิเศษลงไปในรายการ เพื่อบอกว่าช่องนั้นว่าง
   a. จงเขียนฟังก์ชันเพื่อท่องไปในรายการ ที่ใช้เก็บคะแนนสอบในโครงสร้างแถวลำดับ และ คำนวณหาคะแนนเฉลี่ย
   b. จงเขียนฟังก์ชัน delete สำหรับการดำเนินงานลบสมาชิกออกจากรายการ
   c. จงเขียนฟังก์ชัน insert สำหรับการดำเนินงานใส่ข้อมูลลงไปในรายการ

2. จงเขียนฟังก์ชันทำหน้าที่นับจำนวนโหนดของ รายการโยง โดยมี list ชี้ไปที่โหนดแรก

3. จงเขียนฟังก์ชันซึ่งเพิ่มโหนดไปที่ปลายสุดของรายการ โดยมี list ชี้ไปที่โหนดแรก

4. จงเขียนฟังก์ชันที่ใช้ทดสอบว่าชิ้นข้อมูลที่อยู่ในรายการโยง นั้นมีการเก็บแบบเรียงลำดับ

5. จงเขียนฟังก์ชันที่ค้นหาข้อมูลในรายการโยง ที่ประกอบด้วย list ชี้ไปที่โหนดแรก เมื่อการค้นข้อมูลสิ้นสุด ถ้าที่พบข้อมูล ฟังก์ชันจะแสดงค่าภายในของโหนดที่อยู่ก่อนหน้าข้อมูลที่ต้องการค้นหา

6. จงเขียนฟังก์ชันที่ลบโหนดตำแหน่งที่ n ออกจากรายการโยง เมื่อ list ชี้ไปที่โหนดแรกของรายการโยง และ n เป็นเลขจำนวนเต็มที่กำหนดให้

7. เมื่อนำรายการโยง 2 สาย คือ $x_1, x_2, \ldots x_m$ และ $y_1, y_2, \ldots y_n$ มารวมกันแบบสลับ (Shuffle merge) จะได้รายการโยงใหม่ คือ z

   $z = x_1, y_1, x_2, y_2, \ldots, x_m, y_n, y_{m+1}, \ldots y_{n+1}$ ถ้า $n > m$

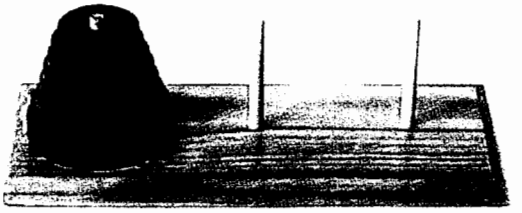
   หรือ $z = x_1, y_1, x_2, y_2, \ldots, x_n, y_m, x_{n+1}, \ldots x_{m+1}$ ถ้า $n < m$

   จงเขียนฟังก์ชันที่แก้ปัญหาที่ได้กล่าวไปแล้ว

8. จงเขียนโปรแกรมที่สามารถดำเนินการผสานแบบสลับ (Shuffle merge) ของรายการโยง 2 รายการ ซึ่งโหนดแรกชี้โดย list1 และ list2 ตามลำดับ ชิ้นข้อมูลของรายการทั้งสองนี้จะได้รับการคัดลอกไปสร้างรายการใหม่ ดังนั้นรายการเดิม 2 รายการ จะถูกทำลาย

9. จงเขียนโปรแกรมที่รวมรายการโยงสองสาย เข้าด้วยกันโดยที่ผลลัพธ์เป็นรายการโยง ที่มีการเรียงลำดับจากน้อยไปมากสมมติให้ข้อมูลที่เก็บในรายการโยง ทั้งสองสายนั้นเรียงลำดับจากน้อยไปมาก

Tower of Hanoi

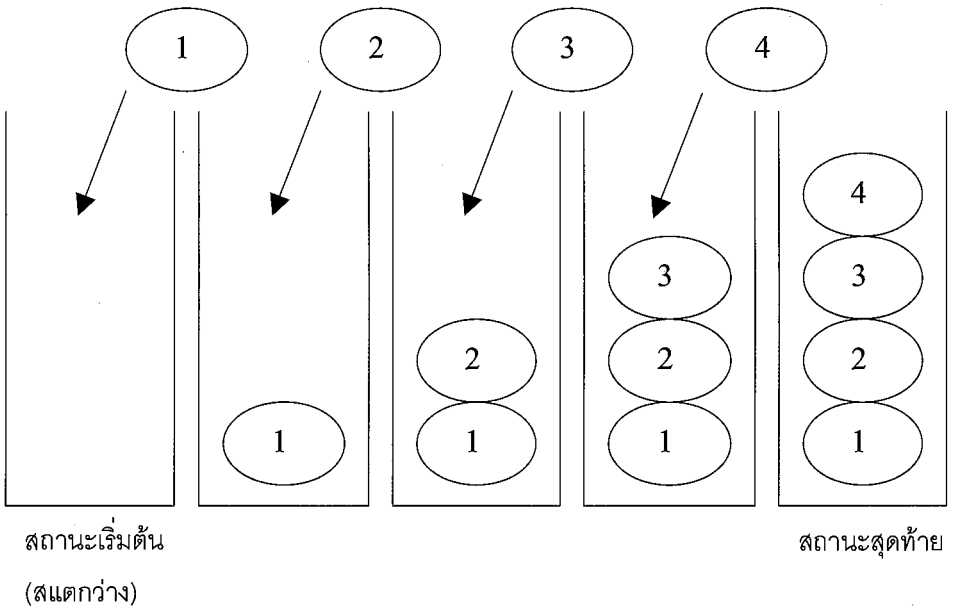


บทที่ 3
สแตก

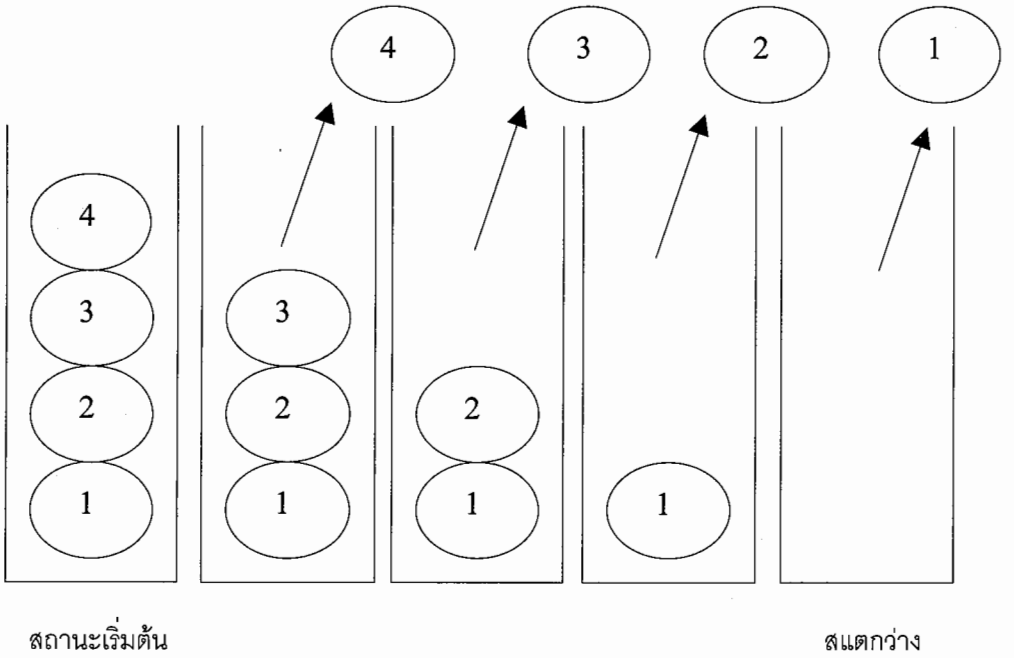
โครงสร้างแบบสแตก (stack) มีประโยชน์มากในการเขียนโปรแกรมคอมพิวเตอร์เช่น สแตกใช้ในการจดจำลำดับการเรียกฟังชันย่อยในโปรแกรมหลัก หรือโปรแกรมรีเคอร์ซีฟ (recursive) อีกทั้งในการท่องไปในโครงสร้างแบบต้นไม้ (tree) สแตกก็ถูกใช้เป็นโครงสร้างข้อมูลที่ช่วยในการจดจำเส้นทาง

3.1 โครงสร้างข้อมูลแบบสแตก (stack)

โครงสร้างข้อมูลแบบสแตกเป็นโครงสร้างข้อมูลที่นำข้อมูลเข้าหรือออกได้ทางเดียวเท่านั้นคือส่วนบนของสแตก ตัวอย่างการทำงานของโครงสร้างข้อมูลอาจพบได้ในชีวิตประจำวันทั่วไปเช่น การนำชั้นปิ่นโตใส่ซ้อนลงในเถาปิ่นโต จัดเป็นการทำงานของโครงสร้างแบบสแตก ถ้ามีปิ่นโตที่มีจำนวนชั้นไม่จำกัด สามารถนำชั้นของปิ่นโตเข้าหรือออกจากเถาได้เพียงทางเดียวเท่านั้น ดังนั้น จะเห็นว่าชั้นที่ถูกใส่เข้าไปในเถาปิ่นโตเป็นใบแรกสุดจะอยู่ล่างสุดในเถา และเมื่อถูกนำออกชั้นที่ถูกใส่ไปหลังสุดจะถูกนำเอาออกมาก่อน เรียงไปเรื่อยๆ จนถึงชั้นที่ใส่แรกสุดจะนำเอาออกมาได้เป็นลำดับสุดท้าย คุณสมบัติดังกล่าวทำให้โครงสร้างข้อมูลของสแตกถูกเรียกว่าเป็นโครงสร้างแบบ เข้าทีหลังออกก่อน (Last-In-First-Out หรือ LIFO) ดังรูปที่ 3-1 ที่แสดงการนำข้อมูลชุดที่ 1, 2, 3 และ 4 เข้าสแตกตามลำดับ และรูปที่ 3-2 แสดงการนำข้อมูลออกจากสแตกในรูปที่ 3-1



รูปที่ 3-1 ลำดับการใส่ข้อมูลในสแตก



รูปที่ 3-2 ลำดับการนำข้อมูลออกจากสแตก

โครงสร้างข้อมูลแบบสแตกมีฟังก์ชันที่สามารถจัดการสแตกได้ 6 ฟังก์ชันคือ

1. การสร้างสแตก (create)
2. การใส่ข้อมูลในสแตก (push)
3. การนำข้อมูลออกจากสแตก (pop)
4. การทดสอบว่าสแตกว่างเหรือไม่ (isEmpty)
5. การทดสอบว่าสแตกเต็มหรือไม่ (isFull)
6. การหาจำนวนสมาชิกที่อยู่ในสแตก (size)

โครงสร้างสแตก ประกอบด้วยตัวสแตกซึ่งเปรียบเสมือนเถาปิ่นโต โดยจะต้องมีการกำหนดคุณลักษณะของสแตกที่ต้องการอย่างชัดเจน จากนั้นสามารถสร้างสแตกโดยใช้แถวลำดับหรือลิงค์ลิสต์แทนได้ คุณลักษณะของ สแตกที่จะต้องกำหนดได้แก่ชนิดของสมาชิกของสแตกและอันดับของความสัมพันธ์ของค่าในสแตกซึ่งการนำข้อมูลเข้าและออกจากสแตกจะเป็นในรูปแบบ LIFO เมื่อสร้างสแตกเสร็จแล้ว ผลลัพธ์ก็คือ สแตกว่าง

3.2 การสร้างสแตกด้วยแถวลำดับ

ในการสร้างสแตกด้วยแถวลำดับ (array) หมายถึง การเลือกการแทนที่ข้อมูลของสแตกด้วยแถวลำดับ ซึ่งเป็นการจัดสรรพื้นที่หน่วยความจำแบบสถิตย์ (static) กล่าวคือ มีการกำหนดขนาดของสแตกล่วงหน้าว่าจะมีขนาดเท่าใด และมีการจัดสรรเนื้อที่หน่วยความจำให้เลย สแตกสามารถเก็บค่าข้อมูลชนิดใดก็ได้ แต่ต้องเป็นชนิดข้อมูลเดียวกัน การกำหนดชนิดของสมาชิกในสแตกและจำนวนค่าในสแตกสามารถกำหนดได้ดังนี้

```
#define MAXSIZE 10;
int StackArray[MAXSIZE];          /* สร้างสแตกแถวลำดับของจำนวนเต็มที่มีขนาด 10 ค่า */
int StackPointer = -1;            /* สแตกเมื่อถูกสร้าง จะมีสแตกพอยน์เตอร์เป็น -1 */
```

จำนวนสมาชิกของแถวลำดับบ่งบอกถึงจำนวนข้อมูลทั้งหมดที่สามารถเก็บค่าไว้ในสแตกได้ โดยจะมีตัวแปรตัวหนึ่งซึ่งเป็นสแตกพอยน์เตอร์ (Stack Pointer) เรียกย่อๆว่า SP เป็นตัวชี้ข้อมูลที่อยู่บนสุดของสแตก ฟังก์ชันที่สามารถทำได้กับสแตก โดยให้สมาชิกเป็นชนิดข้อมูลใดชนิดหนึ่งเช่น Item ที่เป็นโครงสร้างชนิดข้อมูลที่ถูกประกาศไว้แล้ว มีดังนี้

```
Item StackArray[MAXSIZE];
int  SP = -1;
```

```
/* การนำสมาชิกที่ส่งมาเข้าไปยังบนสุดของสแตก */
void push(Item value);
```

ฟังก์ชัน push จะตรวจสอบว่าสแตกเต็มหรือไม่ ถ้าไม่เต็ม จะนำข้อมูลที่ผ่านค่ามาใส่ในแถวลำดับตำแหน่งของดรรชนีถัดไปที่ค่ายังว่างอยู่ และเลื่อน SP ให้ชี้ไปยังค่าที่ใส่เข้ามาใหม่ (SP = SP+1)

```
/* การนำค่าที่อยู่บนสุดของสแตกออกและส่งคืนค่านั้น */
Item pop();
```

ฟังก์ชัน pop จะตรวจสอบว่าสแตกเป็นสแตกว่างหรือไม่ ถ้าไม่จะนำตัวแปรชั่วคราวมารับค่าจากแถวลำดับที่มีดรรชนีเป็น SP จากนั้นให้ค่าที่อยู่ในแถวลำดับ ณ ดรรชนี SP เป็น null และเลื่อน SP ให้ชี้ไปยังค่าก่อนหน้านั้น 1 ตำแหน่ง (SP = SP - 1)

```
/* การทดสอบว่าสแตกเป็นสแตกว่างหรือไม่ ถ้าใช่คืนค่า true ถ้าไม่คืนค่า false */
int isEmpty();
```

ฟังก์ชัน isEmpty จะตรวจสอบค่าของ SP ถ้า SP = -1 หมายถึงสแตกว่าง

```
/* การทดสอบว่าสแตกเต็มหรือไม่ ถ้าเต็ม คืนค่า true ถ้าไม่เต็มคืนค่า false */
int isFull();
```

ฟังก์ชัน isFull ทดสอบว่า SP = MAXSIZE - 1 หรือไม่ ถ้าใช่แสดงว่าสแตกเต็ม

/* การหาจำนวนสมาชิกในสแตก */

```
void size();
```

ฟังก์ชัน size จะคืนค่าดรรชนี SP

3.3 การสร้างสแตกด้วยลิงค์ลิสต์

ในการสร้างสแตกด้วยลิงค์ลิสต์ หมายถึงเราเลือกการแทนที่ข้อมูลของสแตกด้วยลิงค์ลิสต์ ซึ่งเป็นการจัดสรรพื้นที่ของหน่วยความจำแบบพลวัต (dynamic) กล่าวคือ หน่วยความจำจะถูกจัดสรรเมื่อมีการขอใช้จริง ระหว่างการประมวลผลโปรแกรมผ่านตัวแปรชนิดพอยน์เตอร์ (pointer) ดังนั้นถ้าเลือกสร้างสแตกด้วยลิงค์ลิสต์ สแตกจะไม่เต็มในขณะที่ยังสามารถจัดสรรเนื้อที่ในหน่วยความจำให้ได้ การสร้างสแตกนี้จะสร้างเป็นชนิดข้อมูลชื่อ stackList ดังนี้

```
/* StackNode ชนิดข้อมูลที่เป็นโหนดลิงค์ลิสต์ มีสมาชิกคือ Item ซึ่งข้อมูลที่ถูกประกาศไว้แล้ว */
typedef struct stacknode_s StackNode;
struct stacknode_s {
      Item item;
      StackNode *next;
};

/* Stack เป็นโครงสร้างข้อมูลที่รวมโหนดแรกและตัวนับจำนวนสมาชิก */
typedef struct stack_s {
      StackNode *itemList;
      int count;
} StackList;
```

เมื่อแรกสร้างสแตก จะต้องกำหนดค่าเริ่มต้นให้กับสแตกคือ

```
StackList *S;
S->itemList = null;
S->count = 0;
```

```
/* ตัวนับสมาชิกจะเป็นค่าที่เก็บจำนวนสมาชิกที่อยู่ในสแตก */
int stackCount(stackList *S) {
      return S->count;
}
```

การนำสมาชิกใส่ในสแตกที่ใช้ลิงค์ลิสต์มีขั้นตอนดังนี้คือ สร้างโหนดใหม่ขึ้นใส่ค่าที่ต้องการจากนั้นนำโหนดนี้ไปไว้เป็นโหนดแรกของลิสต์ และการนำสมาชิกออกจากสแตกที่ใช้ลิงค์ลิสต์ โหนดแรกจะถูกลบออกจากลิงค์ลิสต์ โดยให้โหนดถัดไปกลายมาเป็นโหนดแรกแทน และคืนค่าที่ถูกเก็บไว้ในโหนดแรก(เก่า)นั้น ให้สังเกตความคล้ายกันในการเขียนฟังก์ชันทั้งสอง จากตัวอย่างการเขียนโปรแกรมหน้าถัดไป

ตัวอย่าง 3-1 stackListPush

```
/* การนำสมาชิกใหม่ใส่ในสแตก */
void stackListPush(Item X, Stack *S) {
      StackNode *temp;
      /* การจองพื้นที่หน่วยความจำสำหรับโหนดที่สร้างใหม่ */
      temp = (StackNode *) malloc(sizeof(StackNode));
      /* ถ้า temp = NULL แสดงว่าการจองไม่ประสบผลสำเร็จ */
      if (temp == NULL) {
            printf("Error: พื้นที่หน่วยความจำถูกใช้หมดแล้ว");
      } else {
            /* ตั้งค่าลิงค์ของ temp ให้ชี้ไปยังส่วนที่เหลือของ itemList. */
            temp->next = S->itemList;
            /* ตั้งค่าสมาชิก item ของ temp ให้เป็น X */
            temp->item = X;
            /* ให้ itemList ของ S ชี้ไปยังโหนดที่สร้างใหม่ ซึ่งถูกใส่เข้าไปที่รายการบนสุด */
            S->itemList = temp;
            /* เพิ่มค่าตัวนับสมาชิก count */
            S->count++;
      }
}
```

ตัวอย่าง 3-2 stackListPop

```
/* นำสมาชิกตัวบนสุดออกจากสแตก */
void stackListPop(Stack *S, ItemType *X) {
      StackNode *temp;
      if (S->itemList == NULL) {
            printf("Error: พยายามจะนำค่าออกจากสแตกว่าง");
      } else {
            /* นำพอยน์เตอร์ชั่วคราวมาชี้สมาชิกตัวบนสุดของสแตก itemList */
            temp = S->itemList;
            /* นำสมาชิกที่ผ่านค่าให้กับฟังก์ชัน (X) มารับค่าสมาชิกตัวบนสุดของสแตก */
            X = temp->item;
            /* ให้สแตกชี้ไปยังสมาชิกตัวถัดไปจากตัวบนสุดซึ่งปัจจุบันถูกชี้โดยพอยน์เตอร์ชั่วคราว temp */
            S->itemList = temp->next;
```

```
                /* ปล่อยพื้นที่หน่วยความจำของพอยน์เตอร์ชั่วคราว */
                /* หมายเหตุ: การปล่อยนี้ไม่ได้ปล่อยการจับจองพื้นที่ของสมาชิกที่ถูกนำออกจากสแตก */
                free(temp);
                /* ลดค่าตัวนับสมาชิก count */
                S->count--;
        }
}
```

3.4 ข้อเปรียบเทียบของประสิทธิภาพของการดำเนินงาน

ข้อเปรียบเทียบของการดำเนินงานทั้ง 6 การดำเนินงานของสแตกที่สร้างด้วยแถวลำดับและสแตกที่สร้างด้วยลิงค์ลิสต์ดังแสดงในตารางที่ 3-1

ฟังก์ชัน	ประสิทธิภาพของสแตกที่สร้างด้วยแถวลำดับ	ประสิทธิภาพของสแตกที่สร้างด้วยลิงค์ลิสต์
create	O(1)	O(1)
push	O(1)	O(1)
pop	O(1)	O(1)
isFull	O(1)	-
isEmpty	O(1)	O(1)
size	O(1)	O(1)

ตารางที่ 3-1 เปรียบเทียบประสิทธิภาพของการดำเนินงานของสแตกที่สร้างด้วยแถวลำดับและลิงค์ลิสต์

หมายเหตุ ดูคำนิยามของสัญกรบิ๊กโอ (Big-Oh Notation) ได้ในบทที่ 9

จากตารางที่ 3-1 จะเห็นว่าทุกฟังก์ชันของสแตกไม่ว่าจะสร้างด้วยแถวลำดับหรือลิงค์ลิสต์จะใช้เวลาคงที่คือ O(1) ไม่ขึ้นอยู่กับจำนวนของข้อมูลในสแตก ในกรณีที่สร้างสแตกด้วยลิงค์ลิสต์ซึ่งมีการจัดสรรพื้นที่ของหน่วยความจำแบบพลวัต จะต้องทำการคืนพื้นที่ของหน่วยความจำ (free) เมื่อเลิกใช้แล้ว มิเช่นนั้น พื้นที่ในหน่วยความจำ ก็จะเสียไปไม่สามารถนำกลับมาใช้ได้อีก อันเป็นปัญหาของการรั่วไหลของพื้นที่หน่วยความจำ (Memory Leak)

แบบฝึกหัดบทที่ 3

1. จงเขียนรูปแสดงการทำงานของสแตกเมื่อมีการเรียกฟังก์ชัน ดังนี้

   create(), pop(), push("orange"), push("mango"), push("apple"), top(), pop(), push("banana"), isEmpty(), pop(), isFull(), size()

2. จงเขียนรูปแสดงการทำงานของสแตกที่มีสมาชิก 7 ค่าเมื่อ

   ก. เริ่มต้นในแสตกมีค่า 2, 8, 5, 6 อยู่แล้วและเพิ่มค่า 7 ลงไป

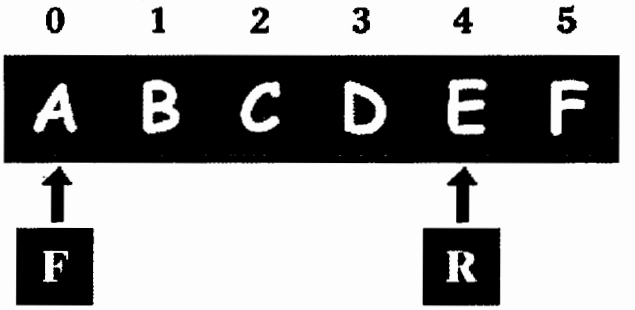
   ข. จากนั้นใส่ค่า 4 ลงไปแล้วทำการ pop 2 ครั้ง

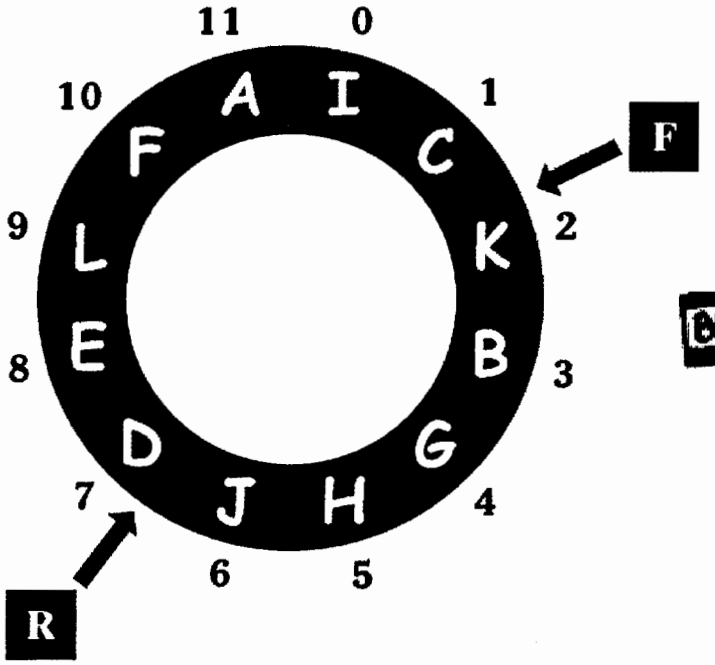
   ค. ทำการ pop 4 ครั้ง

3. สมมติว่าไม่มีการเก็บค่าตัวนับจำนวนสมาชิกในสแตกที่เขียนด้วยลิงค์ลิสต์ จงเขียนฟังก์ชันนับสมาชิกที่อยู่ในสแตกและวิเคราะห์ประสิทธิภาพของฟังก์ชันนี้

4. จงเขียนฟังก์ชันที่กลับลำดับของการนำเข้าสมาชิกในสแตก

5. จงเขียนฟังก์ชันที่รับค่าเป็นสแตกของจำนวนเต็มและให้เรียงลำดับของสมาชิกในสแตกใหม่โดยให้จำนวนเต็มบวกอยู่รวมกันด้านบนของสแตกและจำนวนเต็มลบอยู่รวมกันด้านล่างของสแตก


0 1 2 3 4 5
A B C D E F
F
R


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


BUS


SAME TO YOU
PAK
ISI
TALIBAN
AL-QAIDA

บทที่ 4

คิว

โครงสร้างคิว (queue) มีประโยชน์มากในการเขียนโปรแกรมคอมพิวเตอร์ใช้แก้ปัญหา อีกทั้งมีการประยุกต์ใช้มากในสาขาวิทยาการคอมพิวเตอร์ คิวมีโครงสร้างข้อมูลแบบเชิงเส้น (linear) และมีคุณสมบัติเฉพาะคือ สมาชิกข้อมูลที่นำเข้าไปในคิวก่อนจะถูกนำออกจากคิวก่อน (First-In-First-Out หรือ FIFO)

ตัวอย่างของการทำงานของคิวที่พบเห็นได้ทั่วไปเช่นการเข้าแถว (คิว) เพื่อรอซื้อตั๋วชมภาพยนตร์ ผู้ที่มาเข้าแถวก่อนจะได้ซื้อตั๋วก่อนและสามารถออกจากแถวได้ก่อน คิวเป็นโครงสร้างข้อมูลที่ใช้ในแก้ปัญหาในเรื่องลำดับของการทำงานเช่นงานที่ถูกเรียกก่อนจะถูกนำมาใส่ไว้ในคิวก่อน ซึ่งงานนั้นควรได้ประมวลผลก่อน เป็นต้น

รูปที่ 4-1 ตัวอย่างของการเข้าแถว(คิว) รอรับบริการ

โครงสร้างข้อมูลแบบคิวมีฟังก์ชันที่สามารถจัดการกับคิวได้ 6 ฟังก์ชันคือ

1. การสร้างคิว (create)
2. การนำสมาชิกข้อมูลเข้าคิว (enqueue)
3. การนำสมาชิกข้อมูลออกจากคิว (dequeue)
4. การทดสอบว่าคิวว่างหรือไม่ (isEmpty)
5. การทดสอบว่าคิวเต็มหรือไม่ (isFull)
6. การหาจำนวนสมาชิกที่อยู่ในคิว (length)

ตัวอย่าง 4-1 แสดงการทำงานของฟังก์ชันของคิว ผลลัพธ์ที่ได้และสถานะของคิวที่มีสมาชิกเป็นเลขจำนวนเต็ม

การทำงาน	ผลลัพธ์	สถานะของคิว
enqueue(1)	-	(1)
enqueue(4)	-	(1,4)
dequeue()	1	(4)
enqueue(7)	-	(4,7)
dequeue()	4	(7)
dequeue()	7	()
dequeue()	"error"	()
IsEmpty()	true	()
enqueue(5)	-	(7,5)
enqueue(9)	-	(7,5,9)
enqueue(6)	-	(7,5,9,6)
Length()	4	(7,5,9,6)

ตารางที่ 4-1 แสดงการทำงานของฟังก์ชันของคิว

ในการใช้งานโครงสร้างข้อมูลคิว จะต้องมีการออกแบบคุณสมบัติของคิว โดยการเขียนคุณลักษณะเฉพาะของคิวที่นักเขียนโปรแกรมต้องการอย่างชัดเจน และเมื่อมีการประกาศคุณลักษณะเฉพาะที่ชัดเจนแล้ว ผู้สร้างจะสามารถสร้างคิวตามที่ออกแบบไว้ได้ โดยรายละเอียดของการสร้างจะถูกซ่อนจากผู้ใช้

ในการสร้างคิว ผู้สร้างต้องดำเนินการสองขั้นตอน คือ เลือกการแทนที่ข้อมูลของคิว และสร้างการทำงานโดยใช้การแทนที่ข้อมูลที่เลือกแล้ว ในการสร้างคิว ผู้สร้างสามารถเลือกใช้การแทนที่ข้อมูลของคิวโดยใช้แถวลำดับหรือลิงค์ลิสต์ก็ได้ ในการใช้ชนิดข้อมูลแบบคิว ผู้ใช้เพียงแต่ใช้ตามคุณลักษณะเฉพาะที่ตามที่ถูกออกแบบเท่านั้น โดยไม่ต้องสนใจว่าผู้สร้างจะเลือกการแทนที่ข้อมูลของคิวอย่างไร และคิวถูกสร้างมาด้วยวิธีการเช่นใด

4.1 การสร้างคิวด้วยแถวลำดับ

ในการสร้างคิวด้วยแถวลำดับ หมายถึงการเลือกการแทนที่ข้อมูลของคิวด้วยแถวลำดับซึ่ง เป็นการจัดสรรพื้นที่หน่วยความจำแบบสถิตย์ (static) กล่าวคือ มีการกำหนดขนาดของคิวล่วงหน้าว่าจะมีขนาดเท่าใดและมีการจัดสรรพื้นที่หน่วยความจำให้เลย คิวสามารถเก็บค่าข้อมูลชนิดใดก็ได้ แต่ต้องเป็นชนิดข้อมูลเดียวกัน การกำหนดชนิดของสมาชิกในคิวและจำนวนค่าในคิวสามารถกำหนดได้ดังนี้

```
#define MAXSIZE 10;
Element queue[MAXSIZE];          /* สร้างแถวลำดับคิวของจำนวนต็มที่มีขนาด MAXSIZE */
int front = 0;                   /* คิวเมื่อแรกสร้าง จะมี front เป็น 0 */
int rear = 0;                    /* คิวเมื่อแรกสร้าง จะมี rear เป็น 0 */
```

จำนวนสมาชิกของแถวลำดับบอกถึงจำนวนข้อมูลทั้งหมดที่สามารถเก็บค่าไว้ในคิวได้ โดยจะมีตัวแปรสองตัวซึ่งเป็นตัวชี้ไปยังหัวคิว (front) และท้ายคิว (rear)เป็นตัวชี้ข้อมูลถูกนำเข้าแรกสุดและข้อมูลที่ถูกนำเข้าท้ายสุดตามลำดับ ฟังก์ชันที่ทำงานกับคิวได้ โดยให้มีสมาชิกเป็น Element ที่เป็นโครงสร้างชนิดข้อมูลที่ถูกประกาศไว้แล้ว มีดังนี้

/* การนำสมาชิกข้อมูลเข้าคิว (enqueue) */
อัลกอริทึม enqueue (Element e)
ทดสอบว่าคิวเต็มหรือไม่
ถ้าเต็มแจ้งว่าคิวเต็มไม่สามารถนำสมาชิกเข้าได้
ถ้าไม่เต็ม ให้ `queue[rear] = e` และให้ `rear = (rear + 1) mod MAXSIZE`

/* การนำสมาชิกข้อมูลออกจากคิว (dequeue) */
อัลกอริทึม dequeue ()
ทดสอบว่าคิวว่างหรือไม่
ถ้าว่างแจ้งว่าคิวว่างไม่สามารถนำสมาชิกออกได้
ถ้าไม่ว่างให้สร้างตัวแปรชั่วคราว temp ขึ้นและ
    ให้ `temp = queue[front]` และ
    ให้ `queue[front] = null` และ
    ให้ `front = (front + 1) mod MAXSIZE`
คืนค่า temp

/* การทดสอบว่าคิวว่างหรือไม่ (isEmpty) */
อัลกอริทึม isEmpty ()
ทดสอบว่าคิวว่างหรือไม่โดยทดสอบว่า ค่าของ front เท่ากับ rear หรือไม่

/* การทดสอบว่าคิวเต็มหรือไม่ (isFull) */

อัลกอริทึม isFull()

ทดสอบว่าคิวเต็มหรือไม่โดยทดสอบว่า ค่าของ `front = (rear - 1) mod MAXSIZE` หรือไม่

/* คืนค่าจำนวนสมาชิกทั้งหมดที่อยู่ในคิว */

อัลกอริทึม length()

คืนค่า `(MAXSIZE - front + rear) mod MAXSIZE`

ตัวอย่าง 4-2 แสดงการทำงานกับคิวของอักขระ (char) ที่ใช้ด้วยแถวลำดับและมีสมาชิก 7 ตัว (MAXSIZE = 7)

สร้างคิวว่าง

	0	1	2	3	4	5	6	front	rear
create								0	0

นำสมาชิกที่มีค่า 'C' ใส่ในคิว

	0	1	2	3	4	5	6	front	rear
enqueue	C							0	1

นำสมาชิกที่มีค่า 'O' ใส่ในคิว

	0	1	2	3	4	5	6	front	rear
enqueue	C	O						0	2

นำสมาชิกที่มีค่า 'M' ใส่ในคิว

	0	1	2	3	4	5	6	front	rear
enqueue	C	O	M					0	3

นำสมาชิกที่มีค่า 'P' ใส่ในคิว

	0	1	2	3	4	5	6	front	rear
enqueue	C	O	M	P				0	4

นำสมาชิกออกจากคิว dequeue คืนค่า 'C'

	0	1	2	3	4	5	6	front	rear
dequeue		O	M	P	U			1	4

นำสมาชิกที่มีค่า 'U' ใส่ในคิว

	0	1	2	3	4	5	6	front	rear
enqueue		O	M	P	U			1	5

นำสมาชิกออกจากคิว dequeue คืนค่า 'O'

	0	1	2	3	4	5	6	front	rear
dequeue			M	P	U			2	5

นำสมาชิกที่มีค่า 'T' ใส่ในคิว

	0	1	2	3	4	5	6	front	rear
enqueue			M	P	U	T		2	6

นำสมาชิกที่มีค่า 'E' ใส่ในคิว

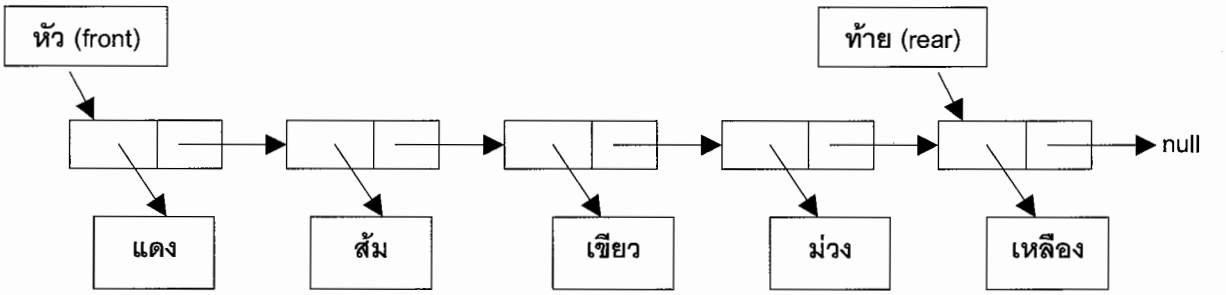
	0	1	2	3	4	5	6	front	rear
enqueue			M	P	U	T	E	2	0

นำสมาชิกที่มีค่า 'R' ใส่ในคิว

	0	1	2	3	4	5	6	front	rear
enqueue	R		M	P	U	T	E	2	1

4.2 การสร้างคิวด้วยลิงค์ลิสต์

ในการสร้างคิวด้วยลิงค์ลิสต์ หมายถึงเราเลือกการแทนที่ข้อมูลของคิวด้วยลิงค์ลิสต์ ซึ่งเป็นการจัดสรร เนื้อที่หน่วยความจำแบบพลวัต (dynamic) นั้นคือ หน่วยความจำจะถูกจัดสรรเมื่อมีการขอใช้จริงๆ ระหว่างการประมวลผลโปรแกรมผ่านตัวแปรชนิด pointer ในการสร้างคิวด้วยลิงค์ลิสต์คิวจะไม่มีวันเต็ม (ไม่มีฟังก์ชัน Full) ตราบใดที่ยังมีพื้นที่ในหน่วยความจำ รูปที่ 4-2 แสดงแผนภาพตัวอย่างของคิวที่สร้างโดยใช้ลิงค์ลิสต์



รูปที่ 4-2 แสดงตัวอย่างของคิวที่สร้างโดยใช้ลิงค์ลิสต์

ตัวอย่างโปรแกรมด้านล่างเป็นการสร้างคิวของตัวอักขระ (char) โดยใช้ลิงค์ลิสต์ โดยมีลำดับคือ สร้างโหนดที่เก็บค่าขึ้นมาก่อน จากนั้นจึงสร้างคิววิ่งประกอบด้วยพอนย์เตอร์ชี้ไปยังโหนดหัวและโหนดท้ายที่อยู่ในคิว

```
typedef struct node                    /* สร้างโหนดเก็บค่า */
{
      char data;
      struct node *next;
} NODE;

typedef struct queue                   /* สร้างชนิดคิวที่บ่งบอกโหนดหัวและโหนดท้าย */
{
      NODE *front;
      NODE *rear;
} QUEUE;

QUEUE create()                         /* ฟังก์ชันสร้างคิว */
{
      QUEUE q = malloc(sizeof(QUEUE));
      q->front = q->rear = NULL;
      return q;
}

int isEmpty(QUEUE q)                   /* ฟังก์ชันทดสอบว่าคิวว่างหรือไม่ */
{
      return q->front ==  q->rear;
}

void enqueue(QUEUE q, NODE *element)/* ฟังก์ชันนำเข้ามูลเข้าคิว */
{
      if(is_empty(q)) {
           q->front = element;
      }else {
           q->rear = element;
           q->rear = q->rear->next;
      }
}
```

```
NODE* dequeue(QUEUE q)                    /* ฟังก์ชันนำเข้ามูลออกจากคิว */
{
      if(!isEmpty(q))
      {
           NODE *temp = q->front;
           q->front = temp->next;
           réturn temp;
      }else {
           return null;
      }
}

int length(QUEUE q)
{
      if(!isEmpty(q))
      {
           NODE *temp = q->front;
           int count = 0;
           while (temp->next != null)
                 count++;
           return count;
      }else {
           return 0;
      }
}
```

4.3 ข้อเปรียบเทียบของประสิทธิภาพของการดำเนินงาน

ข้อเปรียบเทียบของการดำเนินงานทั้ง 6 การดำเนินงานของคิวที่สร้างด้วยแถวลำดับและคิวที่สร้างด้วยลิงค์ลิสต์ ดังแสดงในตารางที่ 4-2

ฟังก์ชัน	ประสิทธิภาพของคิว ที่สร้างด้วยแถวลำดับ	ประสิทธิภาพของคิว ที่สร้างด้วยลิงค์ลิสต์
create	O(1)	O(1)
enqueue	O(1)	O(1)
dequeue	O(1)	O(1)
isFull	O(1)	-
isEmpty	O(1)	O(1)
length	O(1)	O(n)

ตารางที่ 4-2 เปรียบเทียบประสิทธิภาพของการดำเนินงานของคิวที่สร้างด้วยแถวลำดับและลิงค์ลิสต์

จากตารางที่ 4-2 จะเห็นว่าทุกฟังก์ชันของคิวไม่ว่าจะสร้างด้วยแถวลำดับหรือลิงค์ลิสต์จะใช้เวลาคงที่คือ O(1) ไม่ขึ้นอยู่กับจำนวนของข้อมูลในคิว ยกเว้นการทำงานของ length สำหรับคิวที่สร้างด้วยลิงค์ลิสต์ซึ่งเวลาที่ใช้นับจำนวนข้อมูลในคิวขึ้นอยู่กับจำนวนข้อมูลที่อยู่ในคิวคือ O(n) เมื่อ n เป็นจำนวนข้อมูลที่อยู่ในคิวทั้งหมด ในกรณีที่สร้างคิวด้วยลิงค์ลิสต์ซึ่งมีการจัดสรรพื้นที่ของหน่วยความจำแบบพลวัต มีข้อควรคำนึงคือจะต้องทำการคืนพื้นที่ของหน่วยความจำ (free) เมื่อเลิกใช้แล้ว มิเช่นนั้น พื้นที่ในหน่วยความจำ ก็จะเสียไปไม่สามารถนำกลับมาใช้ได้อีก อันเป็นปัญหาของการรั่วไหลของพื้นที่หน่วยความจำ (Memory Leak)

แบบฝึกหัดบทที่ 4

1. คิวขนาด 7 ค่า มีรายการต่อไปนี้

   front = 2, rear = 6, queue[ ] = { null, null, "ขนุน", "ส้มโอ", "แตงโม", "ฝรั่ง", null }

   อธิบายผลของคิวรวมทั้ง front และ rear เมื่อมีการกระทำต่อไปนี้

   ก. เพิ่ม "มะม่วง" ลงในคิว

   ข. ลบ 2 รายการออกจากคิว

   ค. เพิ่ม "ทุเรียน" ลงในคิว

   ง. เพิ่ม "องุ่น" ลงในคิว

   จ. ลบ 2 รายการออกจากคิว

   ฉ. เพิ่ม "ละมุด" ลงในคิว

2. คิวเก็บในแถวลำดับขนาด N =12 ค่า จงหาจำนวนสมาชิกในคิวเมื่อ

   ก. front = 4 , rear = 10

   ข. front = 11 , rear = 2

   ค. front= 3 , rear = 6 และมีการเอาสมาชิก 1 ตัวออกจากคิว

3. จงเขียนแผนภาพและค่าของตัวแปรของคิวที่มีสมาชิกเป็นเลขจำนวนเต็มและสร้างด้วยลิงค์ลิสต์ เมื่อมีการดำเนินการดังนี้

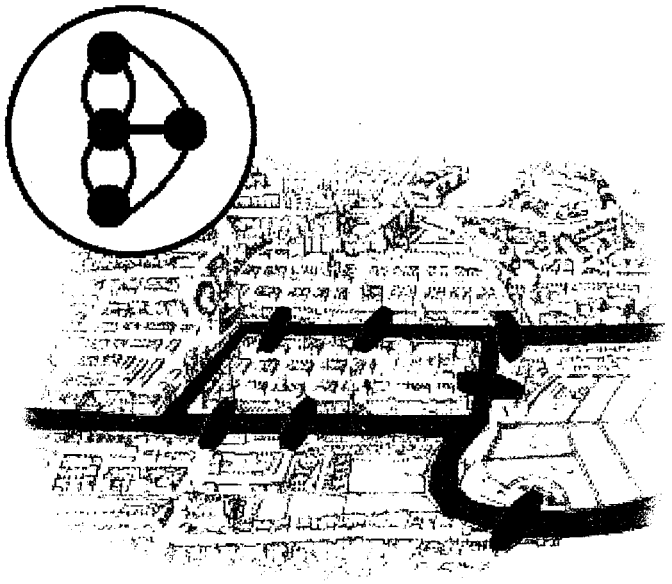
   ก. สร้างคิวว่าง

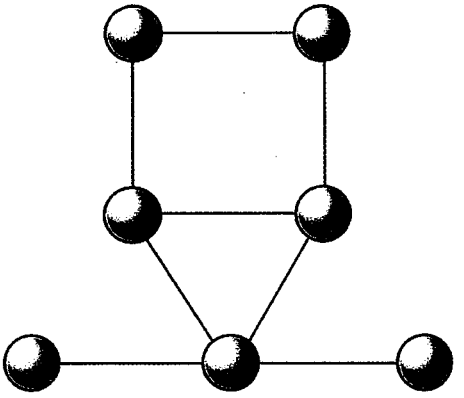
   ข. เพิ่ม 5 และ 8 ลงในคิว

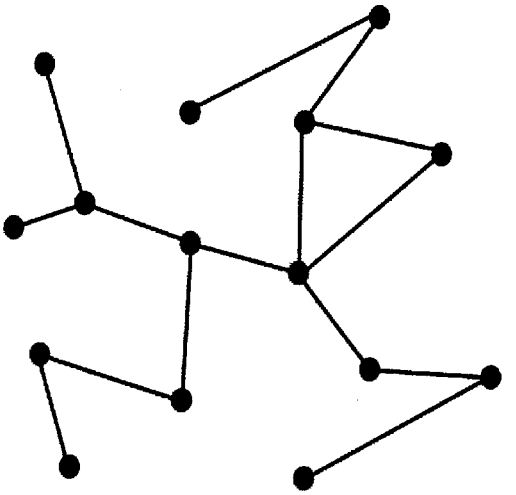
   ค. นำข้อมูลออกจากคิว 1 ค่า

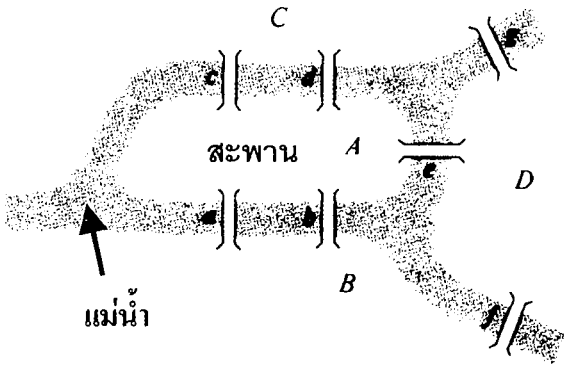
   ง. เพิ่มข้อมูลในคิวอีก 5 ค่า

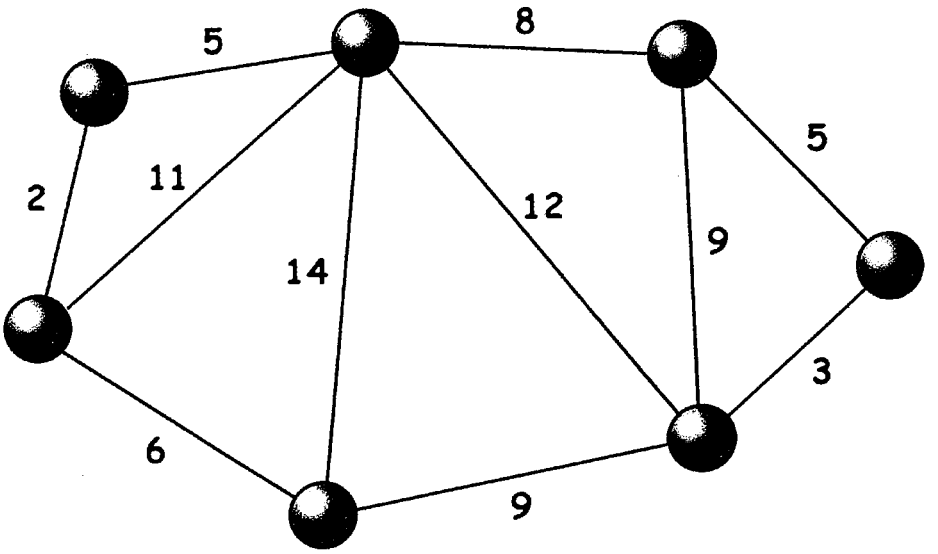
   จ. นับจำนวนสมาชิกในคิว








C
สะพาน
A
D
B
แม่น้ำ


5
8
2
11
12
5
9
14
3
6
9

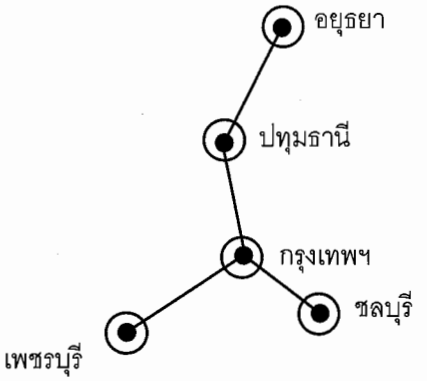
บทที่ 5
กราฟ

5.1 บทนำ

กราฟเป็นโครงสร้างข้อมูลที่มีประโยชน์ เพราะสามารถนำมาใช้แก้ปัญหาในการทำงานหลายด้าน เช่น สามารถใช้กราฟพิจารณาว่าวงจรหนึ่งสามารถนำไปใช้กับแผงวงจรไฟฟ้าแบบระนาบ ในกรณีของวิชาเคมีการใช้กราฟทำให้สามารถบอกความแตกต่างสารเคมี 2 ชนิดที่มีสูตรโมเลกุลเหมือนกันแต่มีโครงสร้างต่างกัน กรณีของเครือข่ายคอมพิวเตอร์สามารถพิจารณาว่าเครื่องคอมพิวเตอร์มีเส้นทางเชื่อมกัน ในกรณีของการสื่อสารใช้โมเดลของเครือข่ายซึ่งแทนด้วยกราฟที่ประกอบด้วยเส้นเชื่อมระบุน้ำหนัก และ สามารถใช้หาระยะทางสั้นที่สุดในการเดินทางระหว่างเมือง 2 เมือง

5.2 กราฟ

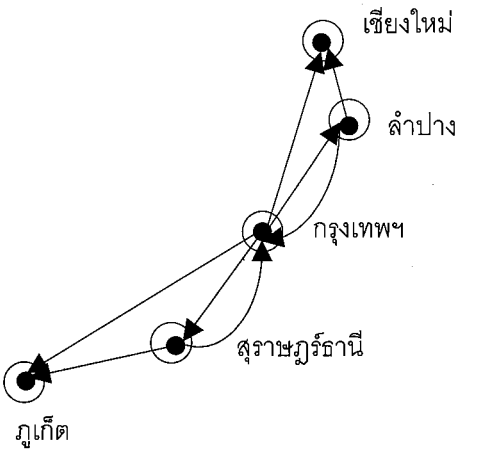
ในการสื่อสารระหว่างเมือง 2 เมือง ซึ่งประกอบด้วยอุปกรณ์คอมพิวเตอร์และสายโทรศัพท์ที่เชื่อมระหว่างเครื่องคอมพิวเตอร์เมื่อนำมาเขียนเป็นกราฟ สามารถแทนตำแหน่งของคอมพิวเตอร์ด้วยจุด และสายโทรศัพท์ด้วยเส้นเชื่อม ดังรูปที่ 5-1



รูปที่ 5-1 การสื่อสารระหว่างเมืองแบบไม่ระบุทิศทาง

จากรูปที่ 5-1 เห็นได้ว่าระหว่างเครื่องคอมพิวเตอร์ 2 เครื่อง ประกอบด้วยสายโทรศัพท์ 1 สาย และสายโทรศัพท์แต่ละสายมีการทำงานสำหรับรับส่งข่าวสารได้ 2 ทาง ระบบการเชื่อมโยงแบบนี้เรียกว่า กราฟแบบไม่ระบุทิศทาง (Undirected Graph) ซึ่งประกอบด้วยโหนด (node) แทนเครื่องคอมพิวเตอร์และเส้นเชื่อมไม่ระบุทิศทาง แทนสายโทรศัพท์ซึ่งเส้นเชื่อมแต่ละเส้นต่อระหว่างโหนด 2 โหนด

กรณีของระบบการสื่อสารที่เชื่อมโยงในการสื่อสารที่ไม่สามารถทำงานได้ 2 ทิศทาง เช่น เครื่องคอมพิวเตอร์ที่เชียงใหม่ สามารถรับข้อมูลจากคอมพิวเตอร์อื่นเขียนแทนด้วยเส้นเชื่อมระบุทิศทางที่ประกอบด้วยหัวลูกศรชี้เข้าสู่โหนดเชียงใหม่ แต่โหนดนี้ไม่สามารถส่งข้อมูลออก นอกจากนี้สายโทรศัพท์เส้นอื่นสามารถดำเนินงานได้ทั้ง 2 ทิศทาง และแทนด้วยเส้นเชื่อมหลายเส้น เขียนในทิศทางตรงกันข้าม



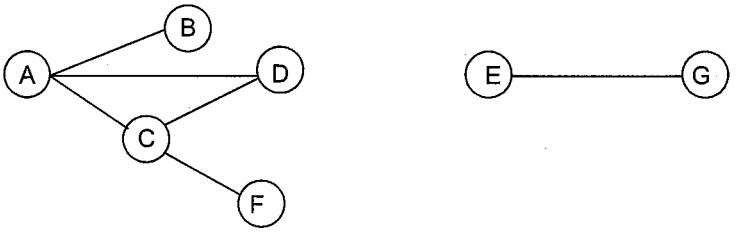
รูปที่ 5-2 การสื่อสารผ่านสายโทรศัพท์ระหว่างเมืองแบบระบุทิศทาง

กราฟเป็นโครงสร้างข้อมูลชนิดหนึ่งที่ประกอบด้วย 2 ส่วนคือ

1) เซตของสมาชิกที่เรียกว่า โหนด (node หรือ vertices) ซึ่งมีจำนวนจำกัด
2) เซตของเส้นเชื่อม (edge) ซึ่งเส้นเชื่อมแต่ละเส้น ในเซตนั้นมีคุณสมบัติเฉพาะตัว เส้นเชื่อมแต่ละเส้นนั้นกำหนดโดยโหนด 1 คู่

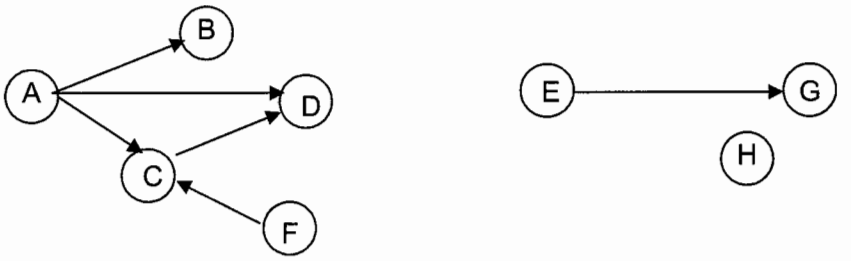
กราฟ G ประกอบด้วยเซตของโหนดคือ {A, B, C, D, E, F, G, H} และ ประกอบด้วยเซตของเส้นเชื่อมคือ {(A,B),(A,D),(A,C),(C,D),(C,F),(E,G),(A,A)}

เมื่อเขียนได้กราฟไม่ระบุทิศทาง ดังรูปที่ 5-3



รูปที่ 5-3 กราฟไม่ระบุทิศทาง

กราฟระบุทิศทางประกอบด้วยสมาชิกที่มีจำนวนจำกัด เรียกว่า โหนด (node) รวมกับสมาชิกของเซตของเส้นเชื่อมระบุทิศทาง (directed graph) ที่เชื่อมระหว่าง 2 โหนด ลูกศรที่เชื่อมระหว่างโหนด v_1 และ v_2 ใช้เขียนแทนเส้นเชื่อม (v_1,v_2) หัวลูกศรชี้ไปที่โหนด v_2ซึ่งเป็นโหนดคู่อันดับที่ประกอบเป็นเส้นเชื่อม ส่วนปลายของลูกศรนั้นอยู่ที่โหนด v_1 เช่น กราฟระบุทิศทางประกอบด้วย 8 โหนด คือ A B C D E F G H และประกอบด้วยเส้นเชื่อมระบุทิศทาง 7 เส้น คือ (A,A), (A,B), (A,C), (A,D), (C,D), (E,G) และ (F,C) สามารถเขียนกราฟได้ คือ



รูปที่ 5-4 กราฟระบุทิศทาง

จากรูปที่ 5-4 สามารถเขียนเซตของเส้นเชื่อมได้ดังนี้ คือ {(A,A), (A,B), (A,C), (A,D), (C,D), (F,C), (E,G)}

5.3 การคำนวณระยะทาง

ระยะทาง k จากโหนด a ไปยังโหนด b ได้รับการกำหนดเป็นแบบลำดับ ของ k+1 โหนด

เมื่อ $n_1, n_2, \ldots\ldots\ldots, n_{k+1}$ และ $n_1 = a, n_{k+1} = b$

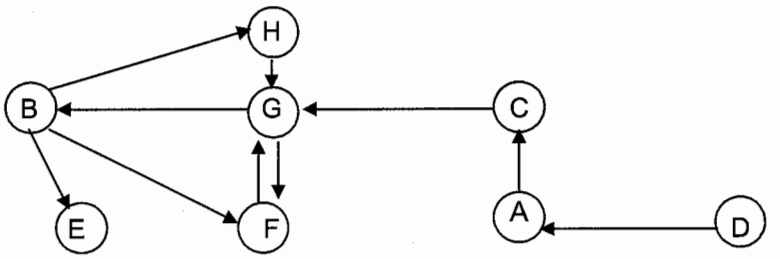
adjacent (n_i, n_{i+1}) เป็นจริง สำหรับ i อยู่ระหว่าง 1 ถึง k เมื่อ k เป็นจำนวนเต็ม

ถ้าระยะทางของ k มีอยู่จริงระหว่าง a และ b แล้ว มีทางเชื่อมจาก a ไปยัง b

วัฏจักร (cyclic graph) เป็นวิถีจากโหนดถึงตัวเอง

กราฟวัฏจักร คือกราฟที่ประกอบด้วยวัฏจักร

กราฟอวัฏจักร (cyclic graph) คือกราฟที่ไม่มีวัฏจักร



รูปที่ 5-5 กราฟระบุทิศทางประกอบด้วยวัฏจักร

เมื่อพิจารณารูปที่ 5-5 เห็นได้ว่า

จาก A ถึง C นั้นมี 1 วิถี (path)

จาก B ถึง G นั้นมี 2 วิถี (path) คือ (B,H), (H,G)

จาก B ถึง C นั้นไม่มีทางเชื่อม

จาก B ถึง B นั้นมี 2 วัฏจักร (cycle) คือ (B,H), (H,G), (G,B) และ (B,F), (F,G), (G,B)

จาก F ถึง F นั้นมี 2 วัฏจักร คือ (F,G), (G, F) และ (F,G), (F,B), (B,F), (F,G)

จาก H ถึง H นั้นมี 2 วัฏจักร คือ (H,G), (G,B), (B,H) และ (H,G), (G,F), (F,G), (G,B), (B,H)

5.4 การแทนเมทริกซ์ประชิด

กำหนดให้กราฟ G ประกอบด้วยเซตของโหนด v_G และเซตของเส้นเชื่อม E_G ซึ่ง $|v_G| = n$

สมมติให้กราฟมีจำนวน n โหนด เมื่อ n >= 1

การแทนกราฟสามารถทำได้โดยเมทริกซ์ประชิด ซึ่งแทนได้โดย แถวลำดับ A ที่มีขนาด $n \times n$ ซึ่ง

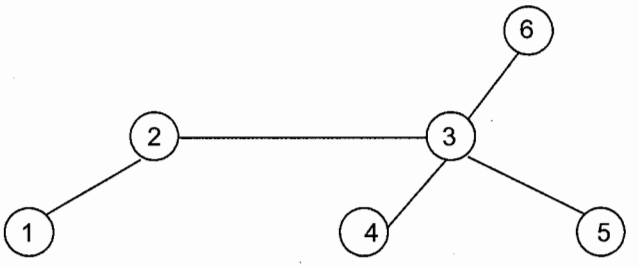
$A(i, j) = 1$ ถ้า v_i ประชิดกับ v_j และมีเส้นเชื่อมเชื่อมระหว่างโหนด i และ j

$= 0$ กรณีอื่นๆ

เมทริกซ์ A ประกอบด้วยค่าภายในเป็น 0 หรือ 1 เรียกว่า บิตเมทริกซ์ หรือ บูลีนเมทริกซ์

เมทริกซ์ประชิด A ของกราฟ G ขึ้นกับการเรียงลำดับของโหนดในกราฟ G ดังนั้นการเรียงลำดับของโหนดที่แตกต่างกันจะให้เมทริกซ์ประชิดที่ต่างกัน อย่างไรก็ตามสมมติว่าโหนดของกราฟ G มีการเรียงลำดับเหมือนกัน

กำหนดให้ G เป็นกราฟไม่ระบุทิศทาง ดังนั้นเมทริกซ์ประชิด A ของกราฟ G เป็นเมทริกซ์สมมาตร ซึ่ง $a_{ij} = a_{ji}$ ทุก i และ j

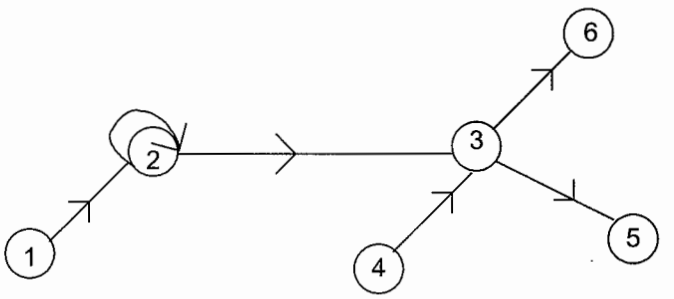


รูปที่ 5-6 กราฟไม่ระบุทิศทาง

เมทริกซ์ประชิด สำหรับกราฟ รูปที่ 5-6 คือ

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

เส้นเชื่อมของกราฟระบุทิศทางมีแหล่งกำเนิดในโหนดหนึ่ง และจบที่อีกโหนดหนึ่ง
เส้นเชื่อม(v_i, v_j) แทนทิศทางจากโหนด v_i ไปยังโหนด v_j



รูปที่ 5-7 กราฟระบุทิศทาง

เมทริกซ์ประชิดรูปที่ 5-7 สามารถเขียนได้ดังนี้

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

เส้นเชื่อม (i, j) มีค่า 1 ถ้ามีเส้นเชื่อมระหว่างโหนด i และ j
ถ้าไม่มีเส้นเชื่อมระหว่างโหนด i และโหนด j เส้นเชื่อม (i, j) มีค่าเป็น 0
กราฟที่มีเส้นเชื่อมระบุน้ำหนักหนักเรียกว่ากราฟน้ำหนัก

5.5 การหาระยะทางสั้นที่สุด

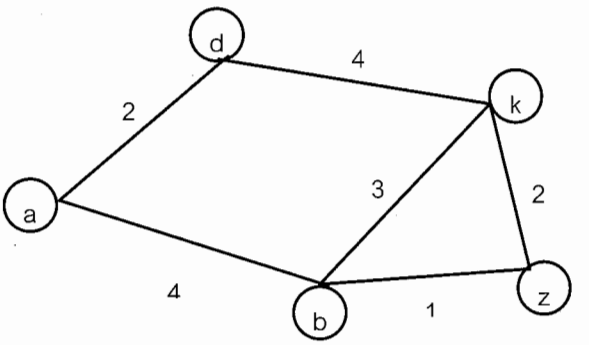
ในการเดินทางระหว่างเมืองสองเมืองที่มีการเดินทางผ่านเมืองที่เป็นชุมทางนั้น สามารถคำนวณหาระยะทางที่สั้นที่สุดในการเดินทางได้ เมื่อกำหนดระยะทางระหว่างเมืองต่างๆ การคำนวณนี้ใช้หลักการของ ไดสตรา (Dijkstra) นักคณิตศาสตร์ ชาวฮอลันดา เป็นผู้ค้นพบอัลกอริทึมในการคำนวณหาระยะทางสั้นสุดของกราฟน้ำหนักที่ไม่มีทิศทาง ซึ่งน้ำหนักของกราฟทุกโหนดมีค่าเป็นบวก

อัลกอริทึมของ Dijkstra

เป็น อัลกอริทึมใช้คำนวณหาเส้นทางที่สั้นที่สุดของกราฟน้ำหนักและน้ำหนักไม่เป็นลบ

กำหนดให้ กราฟ $G = (V,E)$ มี n โหนด และ $w(v,u)$ นั้น w เป็นน้ำหนักของเส้นเชื่อมจากโหนด v ไปที่โหนด u และ $w > 0$

ตัวอย่าง 5-1 การหาระยะทางที่สั้นที่สุดระหว่างโหนด a และ z



รูปที่ 5-8 กราฟมีน้ำหนัก

ในการคำนวณหาระยะทางที่สั้นที่สุด ระหว่างโหนด a และ z โดยใช้อัลกอริทึม Dijkstra ทำได้ดังนี้คือ

ให้โหนด a เป็นจุดเริ่มต้นการเดินทางจากจุด a ไปที่โหนดอื่น มี 2 วิธีคือ a,d และ a,b ระยะทางจาก a,d และ a,b มีค่าเป็น 2 และ 4 ตามลำดับ ดังนั้น d จึงเป็นโหนดที่อยู่ใกล้ a มากที่สุด

ต่อมาจึงคำนวณหาโหนดที่อยู่ใกล้ที่สุดโดยคำนวณจากวิถีทั้งหมดต่อจาก a และ d (จนกระทั่งถึงโหนดปลายทาง) เมื่อพิจารณาขั้นตอนต่อไป พบว่าวิถีของโหนด a กับ b เท่ากับ 4 ซึ่งสั้นกว่าวิธีจาก a,d,k ซึ่งมีความยาวเป็น 6 ฉะนั้นโหนดที่ใกล้ a ที่สุดอันดับถัดมาคือ b

ในการหาโหนดใกล้ a อันดับสาม ทำได้โดยการหาวิถีที่ผ่านโหนด a,d และ b (จนกระทั่งถึงโหนดปลายทาง) เมื่อเดินทางจาก a ไป k สามารถทำได้ 2 วิถี คือ จาก a,d,k มีความยาวเป็น 6 จาก a,b,k มีความยาวเป็น 7 และเมื่อเดินทางจาก a,b,z ได้ความยาว 5 ดังนั้นจึงคำนวณได้ค่าระยะทางสั้นที่สุดมีค่าเป็น 5

อัลกอริทึมของ Dijkstra นั้นเริ่มจากการคำนวณหาระยะทางสั้นที่สุดจาก a ไปยังโหนดแรก คำนวณระยะทางสั้นที่สุด a ไปยังโหนดที่สอง จนกระทั่งพบระยะทางที่สั้นสุดจาก a ไปยัง z

อัลกอริทึมนี้มีการวนซ้ำโดยมีการเพิ่มโหนดที่อยู่ในกราฟทีละโหนด ในการวนซ้ำแต่ละรอบโหนดที่มีการคำนวณระยะทางจากจุดเริ่มต้นแล้วจะทำเครื่องหมาย (เลเบล) เช่น โหนด b ถูกทำเครื่องหมายเนื่องจากมีระยะทางที่สั้นที่สุดจากโหนด a ถึง b และนำโหนด b เก็บไว้ในเซต

ในอัลกอริทึมของ Dijkstra จะกำหนดให้เลเบล a เป็น 0 และ เลเบลของโหนดอื่นมีค่าเป็น ∞ ในที่นี้ให้ $L_0(a) = 0$ และ $L_0(v) = \infty$ กำหนดให้ v เป็นโหนดใดในกราฟซึ่งเป็นการกำหนดค่าเริ่มต้นก่อนการวนซ้ำ (โดยตัวห้อย 0 แทนค่าการวนซ้ำครั้งที่ 0) เลเบล L เป็นความยาวของระยะทางที่สั้นที่สุดจากโหนดอื่นซึ่งมีวิถีไปโหนด a

อัลกอริทึมของ Dijkstra กำหนดให้ S_k เป็นเซตของโหนดที่ได้รับการทำเครื่องหมาย จากการวนซ้ำ k ครั้ง เริ่มต้นด้วย $S_0 = \varnothing$ (S_0 เป็นเซตว่าง) เซต S_k เกิดจากการเพิ่มโหนด u ซึ่งเป็นโหนดใหม่ (โดย โหนด u ไม่เป็นสมาชิกของเซต S_{k-1}) ลงในเซต S_{k-1} ให้ u เป็นโหนดที่มีค่าเลเบลน้อยที่สุด เมื่อเพิ่ม u ลงใน S_k เราจะปรับปรุงเลเบลของโหนดทั้งหมดที่ไม่ได้อยู่ใน S_k ดังนั้น $L_k(v)$ (เลเบลของโหนด v ในระยะทาง k) เป็นระยะทางที่สั้นที่สุดจาก a ไปยัง v ที่ประกอบด้วยโหนดที่เป็นสมาชิกของเซต S_k

ให้ v เป็นโหนดที่ไม่เป็นสมาชิกของ S_k และ $L_k(v)$ เป็น ระยะทางสั้นที่สุดจาก a ไปยัง v ที่เดินทางผ่านโหนดสมาชิกของ S_k การคำนวณระยะทางที่สั้นที่สุดจาก a ไปยัง v ได้จากการเปรียบเทียบและหาค่าเลเบลที่น้อยที่สุดจาก 2 กรณีคือ ระยะทางจาก โหนด a ไป v ผ่านโหนดที่เป็นสมาชิกของ S_{k-1} (ไม่รวมโหนด u) หรือ ระยะทางสั้นที่สุดจาก a ไป u ในระยะทาง k-1 และบวกกับน้ำหนักของเส้นเชื่อมจากโหนด u ไปโหนด v

นั่นคือ $L_k(a,v) = \min \{ L_{k-1}(a,v), L_{k-1}(a,u) + w(u,v) \}$

การดำเนินงานนี้มีการวนซ้ำ โดยการเพิ่มโหนดไปยังเซตที่กำหนดให้จนกระทั่งถึงโหนด z ซึ่งเป็นโหนดสุดท้าย เมื่อเพิ่มโหนด z ลงในเซต เลเบลจะเป็นความยาวของระยะทางที่สั้นที่สุดจาก a ไปยัง z

อัลกอริทึมของ Dijkstra

```
procedure Dijkstra (G: กราฟมีน้ำหนัก โดยน้ำหนักมีค่าเป็นบวก)
{
      G มีโหนด a = v0, v1, …, vn = z และน้ำหนัก w(vi, vj)
      เมื่อ w(vi, vj) = ∞        ถ้า (vi, vj) ไม่ใช่เส้นเชื่อมใน G }
      for  i:= 1 to n
            L(vi) := ∞
            L(a)  := 0
            S := ∅
```

{ เลเบล มีการกำหนดค่าเริ่มต้น ดังนั้น เลเบลของ a มีค่าเป็นศูนย์ และ เลเบลอื่นมีค่าเป็น ∞ และ S เป็นเซตว่าง }

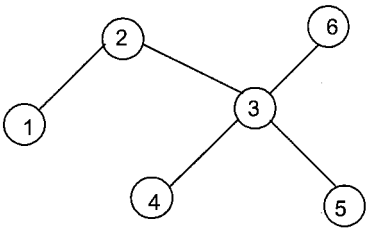
```
while z ∉ S
begin
    u := โหนดที่ไม่เป็นสมาชิกของ S เมื่อ L(u) มีค่าน้อยสุด
    S := S ∪ {u}
    for ทุกโหนด v ที่ไม่เป็นสมาชิกของ S
        if L(u) + w(u,v) < L(v) then L(v) := L(u) + w(u,v)
        {เพิ่มโหนดใหม่ให้ S ที่มีเลเบลมีค่าน้อยสุดและเพิ่มเลเบลของโหนดที่ไม่ได้อยู่ใน S}
end {L(z) = ความยาวของระยะทางสั้นที่สุดจาก a ไป z}
```

5.6 การแทนกราฟด้วยรายการโยงประชิด

การใช้รายการโยงเพื่อประหยัดเนื้อที่ในการเก็บข้อมูลของกราฟมีผู้นิยมใช้กันแพร่หลาย ตัวอย่างเช่น โครงสร้างเครือข่ายระบบคอมพิวเตอร์เพื่อควบคุมการจราจรของเมืองหลวง การสร้างโปรแกรมเพื่อควบคุมปริมาณการจราจรบนถนนหลวง โดยโหนดของกราฟใช้แทนสี่แยก เส้นเชื่อมใช้แทนถนน น้ำหนักของเส้นเชื่อมใช้แทนปริมาณการจราจร เส้นเชื่อมสำหรับถนนเดินทางเดียวนั้นมีน้ำหนัก 1 ตัว เส้นเชื่อมสำหรับถนนที่มีรถวิ่งสวนทางมีน้ำหนัก 2 ตัว ในกรณีนี้ถ้าใช้เมทริกซ์ประชิดแล้ว สำหรับกราฟที่มี n โหนดเขียนโดยใช้เมทริกซ์ขนาด N^2 แต่ถ้าใช้รายการโยง โปรแกรมใช้โหนดเท่าที่จำเป็นเท่านั้น คือ ในกราฟประกอบด้วยเส้นเชื่อมที่ใช้เก็บข้อมูลเท่านั้น

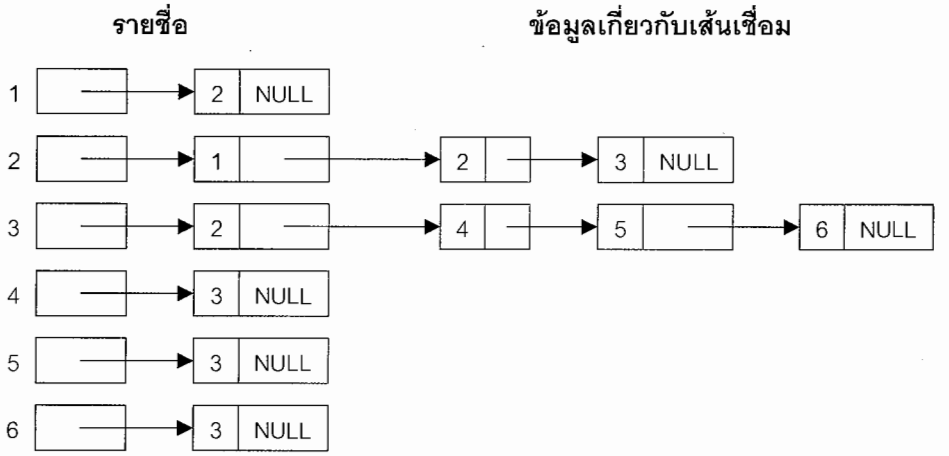
เมื่อพิจารณากราฟแล้วพบว่า มีวิถีเข้าไปยังรายชื่อโหนดของแต่ละโหนดในกราฟ การเข้าไปที่โหนด i โดยชี้ไปที่รายการโยงซึ่งแทนโหนดที่เชื่อมกับโหนด i อันที่จริงระเบียนที่นำมาประกอบเป็นรายการโยง นั้นมี 2 เขต คือ

1. ชื่อโหนด
2. ตัวชี้ที่เชื่อมกับโหนดประชิดในรายการ

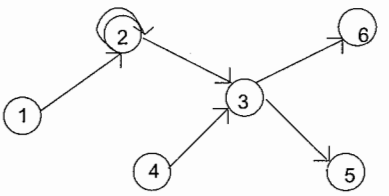


รูปที่ 5-9 กราฟไม่ระบุทิศทาง

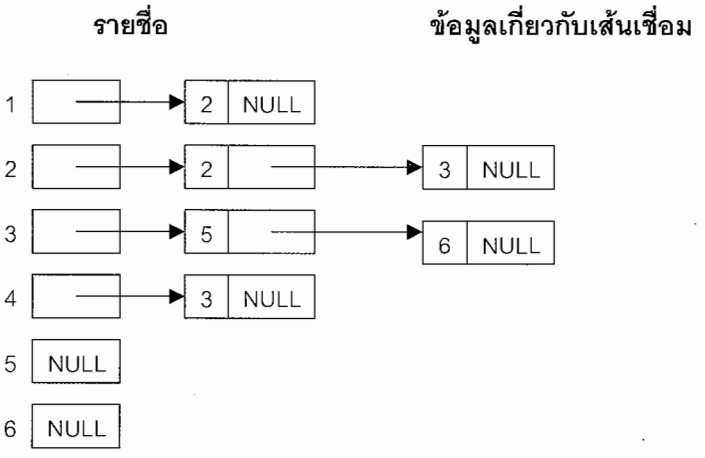
เมื่อเขียนรายชื่อโหนดแทนกราฟไม่ระบุทิศทางของรูปที่ 5-9 ได้ดังนี้



รูปที่ 5-10 รายการโยง ใช้เก็บข้อมูลของกราฟไม่ระบุทิศทาง



รูปที่ 5-11 กราฟระบุทิศทาง



รูปที่ 5-12 รายการโยง ใช้เก็บข้อมูลของกราฟระบุทิศทาง

รายการโยงส่วนหัวประกอบด้วยรายชื่อของโหนดจำนวน i โหนดที่สัมพันธ์กับเมทริกซ์ประชิดจำนวน i แถว สำหรับการจัดรายชื่อนั้น มีการเรียงลำดับตามชื่อโหนดเรียงลำดับจากน้อยไปมาก

การเขียนกราฟในภาษาซี

```
#include <stdio.h>
#include <string.h>

struct arcinfo{
      int node1;
      int node2;
      struct arcinfo *adjlist1;
      struct arcinfo *adjlist2;
      int weight;
};
typedef struct arcinfo ARCINFO;
struct arcinfo nodetype;

void main()
{

}
```

5.7 การแวะผ่านกราฟ

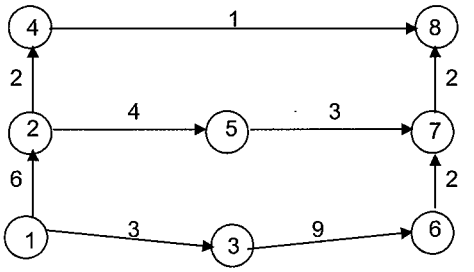
คือ การที่แวะไปที่ทุกๆ โหนดของกราฟ สำหรับเทคนิคในการแวะผ่านมี 2 แบบ คือ

1. แนวกว้าง (Breadth first traversal)
2. แนวลึก (Depth first traversal)

หลักการทำงานของการแวะผ่านกราฟนั้นคือ การไปที่แต่ละโหนดเพียง 1 ครั้ง สำหรับการไปที่โหนดในต้นไม้นั้นมีเพียง 1 วิถีจากรากไปยังโหนด แต่สำหรับในกราฟนั้นมีหลายวิถีระหว่างคู่ของโหนด ดังนั้นเพื่อป้องกันการไปที่โหนดซ้ำซ้อนจึงจำเป็นต้องทำเครื่องหมาย (เลเบล) บริเวณโหนดที่ได้รับการแวะผ่านเสร็จเรียบร้อยแล้วจึงไม่สามารถไปแวะได้อีก นอกจากนี้ยังมีการทำเครื่องหมายให้กับเส้นเชื่อมที่ตามหลังมา สำหรับเส้นเชื่อมที่ทำเครื่องหมายไปแล้วไม่สามารถใช้เป็นวิถีอีก ดังนั้นเลเบลจึงสามารถเก็บใช้ข้อมูลของโหนด หรือเส้นเชื่อมว่าได้รับการแวะผ่านแล้ว

5.7.1 การแวะผ่านแนวกว้าง

การแวะผ่านแนวกว้าง (Breadth first traversal) วิธีการนี้ทำขึ้นโดยเลือก 1 โหนด เป็นจุดเริ่มต้น หลังจากนั้นไปแวะผ่านและทำเครื่องหมายที่โหนดนั้น ต่อมาโหนดอื่นที่เชื่อมกับโหนดนั้นจะได้รับการแวะผ่าน และทำเครื่องหมายตามลำดับ ในที่สุดโหนดที่ยังไม่ได้แวะผ่านซึ่งประชิดกับโหนดที่ได้แวะผ่านไปแล้ว ได้รับการแวะผ่านและทำเครื่องหมาย จนกระทั่งทุก ๆ โหนดในกราฟได้รับการแวะผ่านครบถ้วน



รูปที่ 5-13 กราฟน้ำหนักมีทิศทาง

การแวะผ่านในแนวกว้างของกราฟในรูปที่ 5-13 นั้น จะไปที่โหนดแบบเรียงตามลำดับ ดังนี้คือ 1, 2, 3, 4, 5, 6, 8, 7 หรือการไปที่โหนดเรียงตามลำดับอีกแบบหนึ่ง ดังนี้คือ 1, 3, 2, 6, 5, 4, 7, 8

อัลกอริทึมการแวะผ่าน ใช้คิวเพื่อเก็บโหนดในกราฟแต่ละระดับที่ได้ผ่านไปแวะผ่านโหนดที่นำไปเก็บไว้นั้น จะได้รับการดำเนินงานทีละโหนดและโหนดประชิดก็ได้ไปการแวะผ่าน จนกระทั่งทุกโหนดได้รับการแวะผ่าน เงื่อนไขจบการทำงานของอัลกอริทึมนี้ คือการที่คิวว่าง

อัลกอริทึมการแวะผ่านกราฟแบบแนวกว้าง

อัลกอริทึมนี้แสดงการทำงานของคิว

```
#include <stdio.h>

struct queuestruct {
    int queue[100];
    int r, f;
};
struct queuestruct q;
int n;
void insert(int con) {
    if (q.f != ((q.r + 1) % n)){
        q.r =(q.r + 1) % n;
        q.queue[q.r] = con;
    }
    else {
        prinf("overflow");
    }
}

void remove(int eoff) {
    if (q.r == q.f){
        printf("UNDERFLOW-CONDITION");
    }
```

```
else {
        q.f = (q.f+1) % n;
        eoff = q.queue[q.f];
    }
}
```

อัลกอริทึมนี้แสดง Breadth -first search ของ กราฟ

```
#include <stdio.h>
int ordergraph = 100;
struct nodetype {
    int label;
    struct edgeinfo *adjlist;
};

struct edgeinfo {
    int node;
    int weight;
    struct edgeinfo *next;
};

typedef struct nodetype graphtype[100] ;
graphtype graph;
int firstnode;
struct queuestruct q;
struct queuestruct {
    int queue[100];
    int r, f;
};

int n;
void insert(int con) {
    if (q.f != ((q.r+1) % n )) {
        q.r = (q.r+1) % n;
        q.queue[q.r] = con;
    }
    else {
        prinf("overflow");
    }
}
```

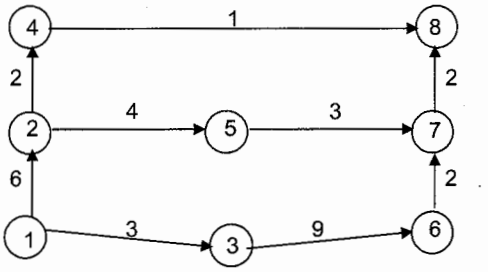
```
void remove(int eoff) {
     if (q.r == q.f) {
          printf("UNDERFLOW-CONDITION");
     }
     else {
          q.f = (q.f+1) % n;
          eoff =q.queue[q.f];
     }
}

void breadth(int firstnode) {
     struct queuestruct q;
     int savenode;
     struct edgeinfo *adjptr;
     graph[firstnode].label = 1;
     insert(firstnode);

     while(q.f != 0)
     {
          remove(savenode);
          adjptr = graph[savenode].adjlist;
          while(adjptr != NULL)
          {
               savenode = adjptr*.node;
               if (graph[savenode].label == 0) {
                    insert(savenode);
                    graph[savenode].label = 1;
               }
               adjptr = adjptr*.next;
           }
     }
}
```

5.7.2 การแวะผ่านแนวลึก

การแวะผ่านแนวลึก (Depth-first Traversal) ทำงานคล้ายกับการแวะผ่านทีละระดับของต้นไม้ โดยกำหนดวิถีแรก จากรากลงไปที่ใบและอีกทางหนึ่งจากรากไปที่ใบจนกระทั่งมีการแวะผ่านหมดทุกโหนด



รูปที่ 5-13 กราฟน้ำหนักมีทิศทาง

การแวะผ่านแนวลึกของกราฟใน รูปที่ 5-13 ให้ผลในการแวะผ่านโหนดแบบเรียงลำดับ จาก 1, 2, 4, 8, 5, 7, 3, 6 การแวะผ่านดำเนินงานจนกระทั่งไม่มีโหนดที่สามารถแวะได้หลงเหลืออยู่เลย อัลกอริทึมนี้ก็กลับไปที่โหนดสุดท้ายที่ได้รับการเยี่ยมซึ่งเชื่อมกับโหนดถัดไปที่ยังไม่ได้รับการไปเยี่ยม ดังนั้น จากรูปที่ 5-13 สามารถไปเยี่ยมโหนดอีกวิธีหนึ่งได้ตามลำดับดังนี้ คือ

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

สำหรับ การแวะผ่านแนวลึกใช้เงื่อนไขแบบเรียกซ้ำ สามารถเขียนเป็นกระบวนงาน ได้ดังนี้ คือ
อัลกอริทึมนี้แสดง Depth-first traversal ของ กราฟ

```
#include <stdio.h>
struct edgeinfo {
     int node;
     int weight;
     struct edgeinfo *next;
};

struct nodetype {
     int label;
     struct edgeinfo *adjlist;
};

typedef struct nodetype graphtype[100];
graphtype graph;
void depth(int thisnode){
     int savenode;
     struct edgeinfo *adjptr;
     graph[thisnode].label = 1;
     adjptr = graph[thisnode].adjlist;
```

```
        while(adjptr !=NULL) {
            savenode = adjptr*.node;
            if(graph[savenode].label == 0) {
                depth(savenode);
                adjptr = adjptr*.next;
            }
        }
    }
```

แบบฝึกหัดบทที่ 5

1. กำหนดให้กราฟ G ประกอบด้วย V ซึ่งเป็นเซตของโหนด และ E เป็นเซตของเส้นเชื่อม จงวาดกราฟและหาระดับชั้นของแต่ละโหนด

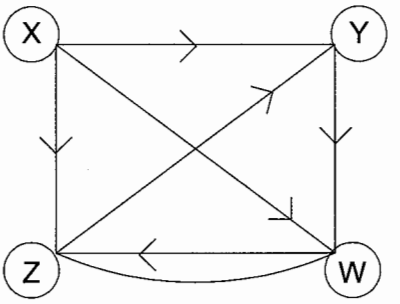
   V = {A,B,C,D,E}

   E = {(A,B),(A,C),(A,D),(B,C),(B,E),(C,D),(C,E)}

2. กำหนดกราฟ G

   (ก) จงหาวิถีทั้งหมดจากโหนด X ไปยังโหนด Z

   (ข) หาวิถีทั้งหมดจากโหนด Y ไปโหนด Z



3. จากเมทริกซ์ประชิดที่กำหนดให้จงเขียนกราฟระบุทิศทาง โดยมีเมทริกซ์ประชิด Adj และเมทริกซ์ของ ข้อมูล (data):

   (ก)

$$\text{Adj} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad \text{Data} = \begin{bmatrix} A \\ B \\ C \\ D \\ E \end{bmatrix}$$

   (ข)

$$\text{Adj} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad \text{Data} = \begin{bmatrix} CAT \\ RAT \\ BAT \\ DOG \end{bmatrix}$$

(ค)

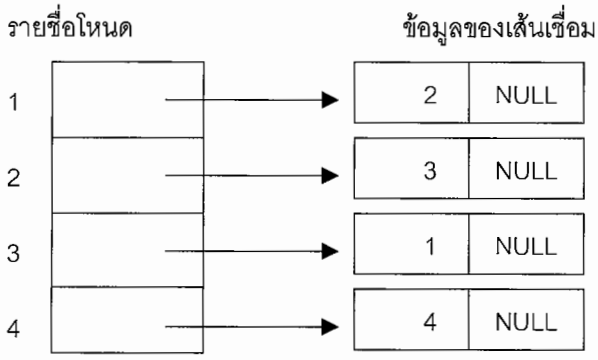
$$\text{Adj} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \text{Data} = \begin{bmatrix} \text{ONE} \\ \text{TWO} \\ \text{THREE} \end{bmatrix}$$

(ง)

$$\text{Adj} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \qquad \text{Data} = \begin{bmatrix} \text{A} \\ \text{B} \\ \text{C} \\ \text{D} \\ \text{E} \\ \text{F} \\ \text{G} \end{bmatrix}$$

4. จากเมทริกซ์ประชิดที่กำหนดให้ในข้อ 3 จงเขียนการเก็บเมทริกซ์เหล่านี้โดยใช้รายการโยง

5. จากรายการโยงแบบประชิดที่กำหนดให้จงวาดกราฟแบบระบุทิศทาง



6. กำหนดเมทริกซ์ประชิด Q ให้จงเขียนกราฟโดยใช้อัลกอริทึมไดสตรา หาวิถีสั้นที่สุด และหาวิถีของเมทริกซ์ Q

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

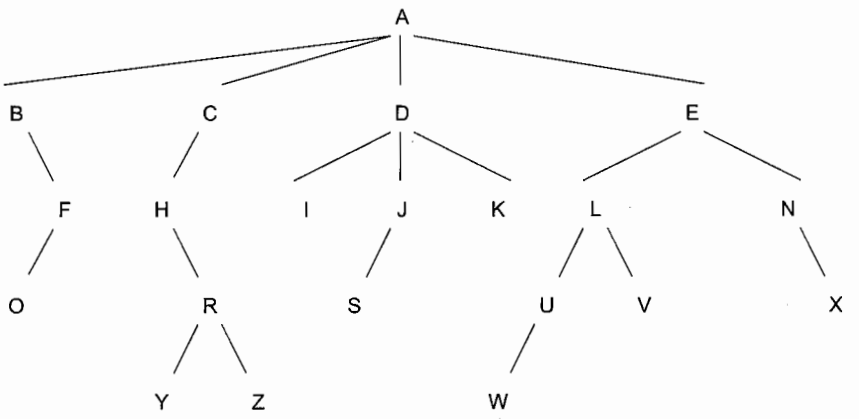
7. จงเขียนโปรแกรมที่สามารถแวะผ่านกราฟระบุทิศทางซึ่งใช้การแวะผ่านแบบลึกและการแวะผ่านแบบกว้างโดยกราฟนั้นมีการแทนด้วยเมทริกซ์ประชิด

8. จงเขียนโปรแกรมเพื่อหาวิถีสั้นสุดของเมทริกซ์ประชิดที่ใช้แทนกราฟระบุทิศทาง

บทที่ 6
ต้นไม้

6.1 ความหมายของต้นไม้

ต้นไม้ (tree) หมายถึง กราฟต่อเนื่อง (connected) ที่ไม่มีวัฏจักร (acyclic) และไม่มีทิศทาง (undirected graph) ต้นไม้จัดเป็นโครงสร้างข้อมูลแบบลำดับชั้น (hierarchical data structure) ที่ประกอบด้วยโหนด (node) จำนวน n โหนด โดย n เป็นจำนวนเต็มใดๆ ที่มีค่ามากกว่าหรือเท่ากับศูนย์ และเมื่อ n มีค่าเป็นศูนย์ เรียกต้นไม้ดังกล่าวว่า **ต้นไม้ว่าง** (empty tree)



รูปที่ 6-1 ต้นไม้

ให้ p และ q เป็นโหนดใดๆ ในต้นไม้ ถ้ามีเส้นเชื่อมจากโหนด p มายังโหนด q เราเรียกว่า โหนด p เป็น**โหนดพ่อแม่** (parent node) ของโหนด q และโหนด q เป็น**โหนดลูก** (child node) ของโหนด p และเรียกความสัมพันธ์ระหว่างโหนดทั้งสองนี้ว่าความสัมพันธ์แบบ "พ่อแม่-ลูก" (parent-child relationship) ซึ่งจะเกิดขึ้นเมื่อโหนด p และโหนด q เป็นโหนดที่อยู่ในลำดับชั้นถัดกัน

โหนดที่มีพ่อแม่เดียวกัน เรียกว่าเป็น**โหนดพี่น้อง** (siblings)

โหนดเดียวในต้นไม้ที่ไม่มีพ่อแม่ เรียกว่า **โหนดราก** (root node) เป็นโหนดที่อยู่ชั้นบนสุดของต้นไม้

โหนดใบ (leaf node) หรือเรียกอีกชื่อหนึ่งว่า**โหนดภายนอก** (external node) ได้แก่โหนดที่ไม่มีลูก และเรียกโหนดอื่นๆ ที่ไม่ใช่โหนดใบว่า **โหนดภายใน** (internal node)

โหนดที่อยู่ในลำดับชั้นที่ต่างกันมากกว่า 1 ชั้น เรียกว่า มีความสัมพันธ์แบบ "บน-ล่าง" (ancestor-descendant relationship)

ต้นไม้ย่อย (subtree) ประกอบด้วยโหนดทุกโหนดที่เป็นลูกและหลานของโหนดดังกล่าว ดังนั้น ต้นไม้ย่อยของโหนด p จึงได้แก่ ต้นไม้ย่อยที่มีโหนดรากเป็นลูกของโหนด p

ดีกรี (degree) ของโหนด p หมายถึง จำนวนลูกทั้งหมดของโหนด p

กำหนดให้โหนดรากมีระดับเป็นศูนย์ **ระดับของโหนด** p จะมีค่ามากกว่าระดับของโหนดที่เป็นโหนดพ่อแม่ของโหนด p 1 ระดับ

ความสูงของต้นไม้ จะมีค่าเท่ากับระดับของโหนดที่มีค่าสูงที่สุดในต้นไม้

ดังนั้น จากรูปที่ 6-1 เราจะได้ว่า

- โหนด C เป็นโหนดพ่อแม่ของโหนด H, โหนด S เป็นโหนดลูกของโหนด J และโหนด B, C, D และ E เป็นโหนดลูกของโหนด A
- โหนด I, J และ K เป็นโหนดพี่น้องกัน เช่นเดียวกับโหนด U และ V
- โหนดรากของต้นไม้ ได้แก่ โหนด A
- โหนดใบ ได้แก่ โหนด I, K, O, S, V, W, X, Y และ Z
- โหนดภายใน ได้แก่ โหนด A, B, C, D, E, F, H, J, L, N, R และ U
- ต้นไม้ย่อยของโหนด A มีทั้งหมด 4 ต้นไม้ย่อย ซึ่งได้แก่ ต้นไม้ย่อยที่มีโหนด B, C, D และ E เป็นโหนดรากของต้นไม้ย่อยทั้งสี่ต้น ตามลำดับ

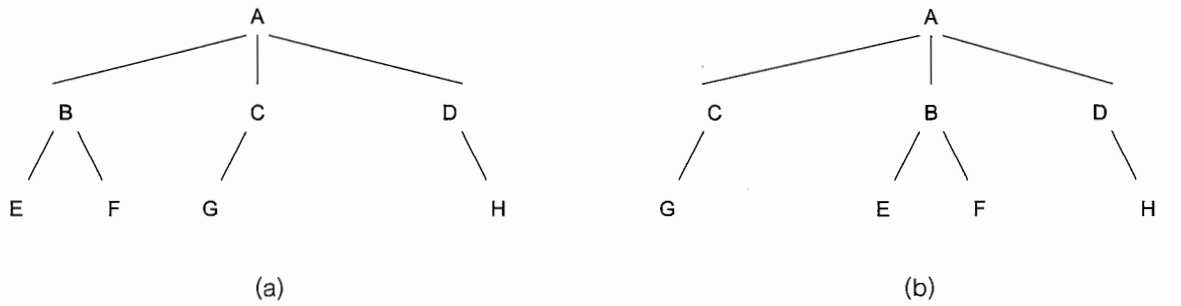
  ต้นไม้ย่อยของโหนด C มีทั้งหมด 1 ต้นไม้ย่อย ซึ่งได้แก่ ต้นไม้ย่อยที่มีโหนด H เป็นโหนดรากของต้นไม้ย่อย

  ต้นไม้ย่อยของโหนด L มีทั้งหมด 2 ต้นไม้ย่อย ซึ่งได้แก่ ต้นไม้ย่อยที่มีโหนด U และ V เป็นโหนดรากของต้นไม้ย่อยทั้งสองต้น ตามลำดับ
- โหนด D มีดีกรีเท่ากับ 3, โหนด F มีดีกรีเท่ากับ 1 และโหนด K มีดีกรีเท่ากับ 0
- ระดับของโหนด A มีค่าเท่ากับ 0 ระดับของโหนด B, N, R และ Y มีค่าเท่ากับ 1, 2, 3 และ 4 ตามลำดับ
- ความสูงของต้นไม้ที่มีโหนด A เป็นโหนดราก มีค่าเท่ากับ 4

6.2 ประเภทของต้นไม้

ต้นไม้ n ดีกรี (n-ary tree) ได้แก่ ต้นไม้ที่โหนดทุกโหนดมีดีกรีไม่เกิน n เมื่อ n เป็นจำนวนเต็มบวกใดๆ เช่น เมื่อ n = 2 ทุกโหนดในต้นไม้มีลูกได้ตั้งแต่ 0 ถึง 2 โหนด เรียกต้นไม้นี้ว่า ต้นไม้ทวิภาค (binary tree) เป็นต้น

ต้นไม้ 2 ต้นใดๆ จัดเป็นต้นไม้เดียวกัน ถ้าต้นไม้ทั้งสองมีจำนวนโหนดเท่ากัน และโหนดลูกแต่ละโหนดของต้นไม้แต่ละต้น มีโหนดพ่อแม่ที่ตรงกัน ดังนั้น เมื่อพิจารณาต้นไม้ในรูปที่ 6-2 (a) และ (b) เราจะพบว่าต้นไม้ทั้งสองมีจำนวนโหนดเท่ากันคือ 8 โหนด มีโหนด B, C และ D เป็นโหนดลูกของโหนด A โหนด E และ F เป็นโหนดลูกของโหนด B โหนด G เป็นโหนดลูกของโหนด C และโหนด H เป็นโหนดลูกของโหนด D ดังนั้นต้นไม้ทั้งสองเป็นต้นไม้เดียวกัน



รูปที่ 6-2 ต้นไม้ในรูป (a) และรูป (b) มีโหนด A, B, ..., H เหมือนกัน

ต้นไม้ลำดับ (ordered tree) ได้แก่ ต้นไม้ที่คำนึงถึงความสำคัญของลำดับในเชิงเส้นของโหนดพี่น้อง นั่นคือ ถ้าโหนดหนึ่งในต้นไม้มีดีกรี n หมายความว่าโหนดดังกล่าวมีลูกลำดับที่ 1, ลำดับที่ 2, ... จนถึงลำดับที่ n ดังนั้น เมื่อพิจารณาต้นไม้ในรูปที่ 6-2 (a) และ (b) จะเห็นได้ว่า ลำดับในเชิงเส้นของโหนดลูกของโหนด A ของต้นไม้ทั้งสองแตกต่างกัน โดยต้นไม้ในรูปที่ 6-2 (a) โหนด A มีโหนด B, C และ D เป็นโหนดลูกลำดับที่ 1, 2 และ 3 ตามลำดับ ขณะที่ต้นไม้ในรูปที่ 6-2 (b) โหนด A มีโหนด C เป็นโหนดลูกลำดับที่ 1, โหนด B เป็นโหนดลูกลำดับที่ 2 และโหนด D เป็นโหนดลูกลำดับที่ 3 ดังนั้น ถ้าต้นไม้ในรูปที่ 6-2 (a) และ (b) เป็นต้นไม้ลำดับ จะกล่าวได้ว่า ต้นไม้ทั้งสองไม่เป็นต้นไม้เดียวกัน

6.3 ต้นไม้ทวิภาค

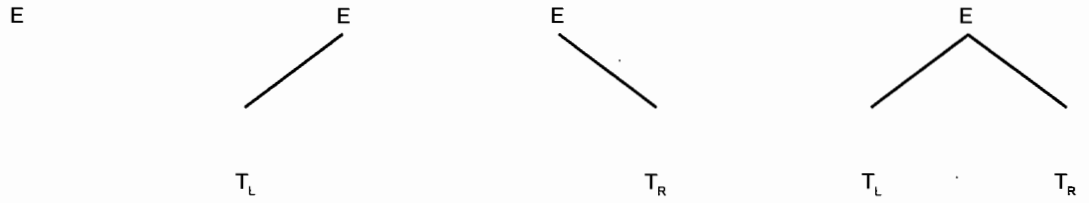
นอกจากการนิยามต้นไม้ทวิภาคตามลักษณะของต้นไม้ n ดีกรี เมื่อ n มีค่าเป็น 2 แล้ว เรายังสามารถนิยามต้นไม้ทวิภาคในลักษณะของการนิยามแบบเวียนบังเกิด (recursive definition) ได้ โดยถ้าให้ T แทนต้นไม้ทวิภาคใดๆ แล้ว T ได้แก่เซตของโหนดของต้นไม้ทวิภาคที่มีสมบัติอย่างใดอย่างหนึ่งในสองกรณีต่อไปนี้

<u>กรณีที่ 1</u> เซต T เป็นเซตว่าง

<u>กรณีที่ 2</u> เซต T ประกอบด้วยเซตที่ไม่มีส่วนร่วม (disjoint set) จำนวน 3 เซต ซึ่งมีสมบัติ คือ

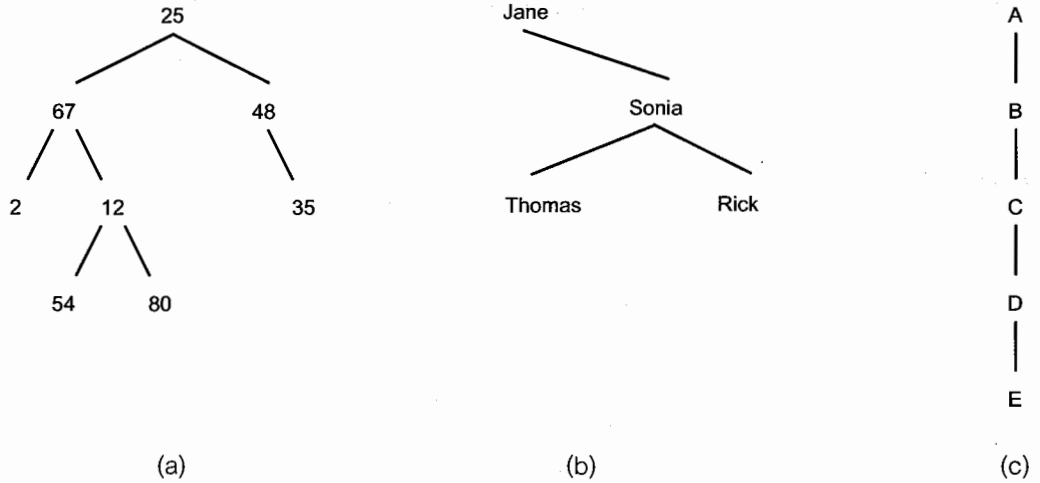
- เซตที่ประกอบด้วยสมาชิก 1 ตัว ได้แก่ โหนดราก
- เซตของต้นไม้ทวิภาคอีก 2 เซต ซึ่งเป็นต้นไม้ย่อยของโหนดราก เราเรียกต้นไม้ย่อยทั้งสองว่า **ต้นไม้ย่อยทางซ้าย** (left subtree) และ**ต้นไม้ย่อยทางขวา** (right subtree) ของโหนดราก ตามลำดับ

จะเห็นได้ว่า ถ้า T เป็นเซตว่างแล้ว T จะเป็นต้นไม้ทวิภาคว่าง และถ้า T ไม่เป็นเซตว่าง และต้นไม้ย่อยทางซ้าย (T_L) และต้นไม้ย่อยทางขวา (T_R) ของโหนดรากเป็นต้นไม้ว่าง ต้นไม้ทวิภาคดังกล่าวจะเป็นต้นไม้ที่ประกอบด้วยโหนดรากเพียงโหนดเดียว ดังแสดงในรูปที่ 6-3 (a) แต่ถ้าต้นไม้ย่อยทางซ้าย และ/หรือต้นไม้ย่อยทางขวาของโหนดรากไม่เป็นต้นไม้ว่าง ต้นไม้ทวิภาคจะเป็นดังรูปที่ 6-3 (b), (c) และ (d)



รูปที่ 6-3 ต้นไม้ทวิภาค (a) ของโหนดราก E (b) มีกิ่งทางซ้าย (c) มีกิ่งทางขวา และ (d) มีกิ่งทางซ้ายและขวา

ในการเขียนรูปต้นไม้ทวิภาค จะต้องเขียนให้ชัดเจนว่าโหนดแต่ละโหนดเป็นลูกทางซ้าย (left child) หรือลูกทางขวา (right child) ของโหนดพ่อแม่ ดังแสดงในรูปที่ 6-4 (a) และ (b) ขณะที่ต้นไม้ทวิภาคในรูปที่ 6-4 (c) ไม่สามารถบอกได้ว่าโหนด B, C และ D เป็นลูกทางซ้ายหรือลูกทางขวาของโหนดพ่อแม่ ซึ่งเป็นการเขียนต้นไม้ทวิภาคที่ไม่เหมาะสม

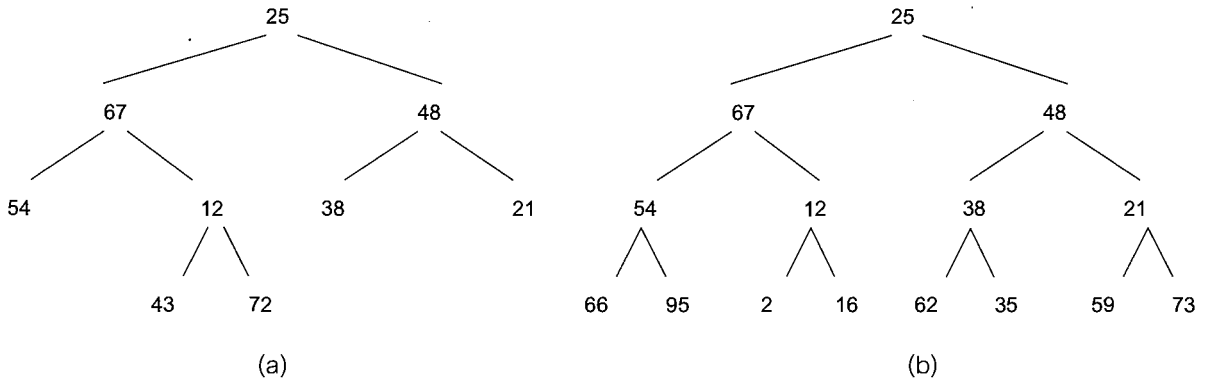


รูปที่ 6-4 การเขียนต้นไม้ทวิภาคแบบต่างๆ

6.3.1 ประเภทของต้นไม้ทวิภาค

ต้นไม้ทวิภาคแบบเต็ม (full binary tree) ได้แก่ ต้นไม้ทวิภาคที่โหนดภายในทุกโหนดมีดีกรีเท่ากับ 2 (รูปที่ 6-5 (a))

ต้นไม้ทวิภาคแบบสมบูรณ์ (complete binary tree) ได้แก่ ต้นไม้ทวิภาคที่โหนดใบทุกโหนดอยู่ในระดับเดียวกัน และโหนดภายในทุกโหนดมีดีกรีเท่ากับ 2 (รูปที่ 6-5 (b))



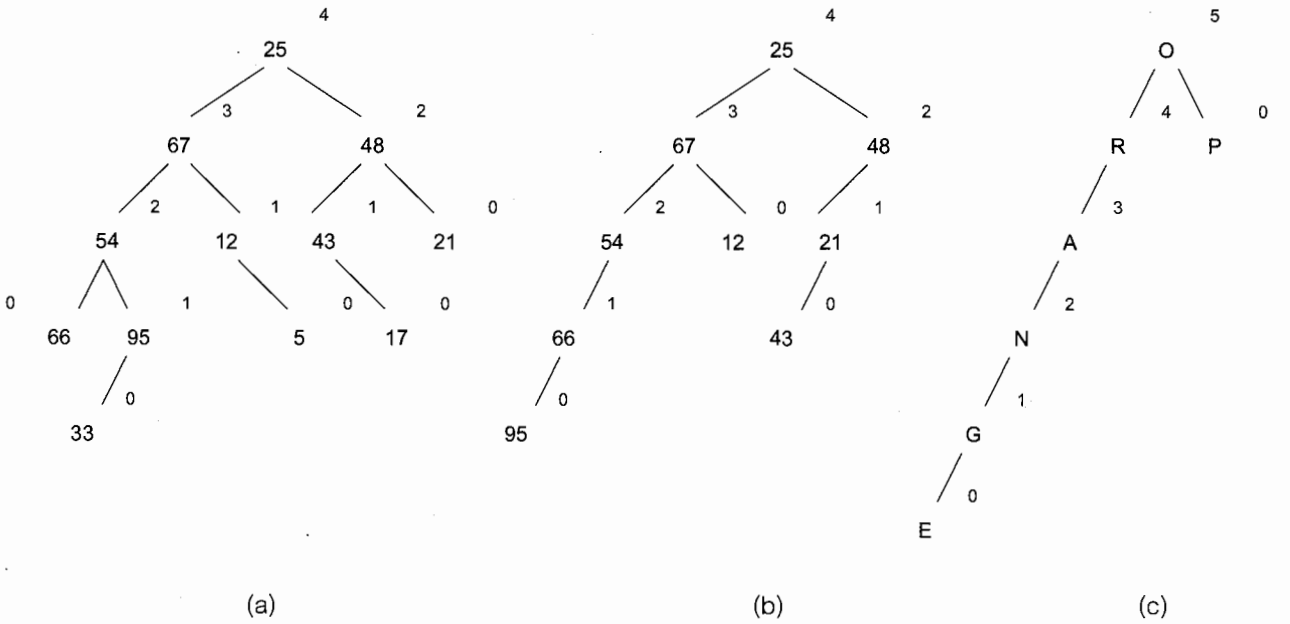
รูปที่ 6-5 (a) ต้นไม้ทวิภาคแบบเต็มและ (b) ต้นไม้ทวิภาคแบบสมบูรณ์

ต้นไม้ทวิภาคแบบสมดุล (balanced binary tree) ได้แก่ ต้นไม้ทวิภาคที่ความสูงของต้นไม้ย่อยทางขวาของโหนดใดๆ ต่างจากความสูงของต้นไม้ย่อยทางซ้ายของโหนดเดียวกัน ไม่เกิน 1 นั่นคือ

เมื่อกำหนดให้ height(T_L) แทนความสูงของต้นไม้ย่อยทางซ้าย และ height(T_R) แทนความสูงของต้นไม้ย่อยทางขวาของโหนด T เมื่อ T เป็นโหนดใดๆ ในต้นไม้ทวิภาค ต้นไม้ทวิภาคนี้จะเป็นต้นไม้ทวิภาคสมดุล เมื่อ $|height(T_L) - height(T_R)|$ ของทุกโหนดมีค่าน้อยกว่าหรือเท่ากับ 1

พิจารณาต้นไม้ทวิภาคในรูปที่ 6-6 เมื่อตัวเลขที่อยู่มุมด้านบนโหนดแทนความสูงของต้นไม้ที่มีโหนดนั้นๆ เป็นโหนดราก เราจะพบว่าต้นไม้ในรูปที่ 6-6 (a) เป็นต้นไม้ทวิภาคแบบสมดุล เนื่องจากความสูงของต้นไม้ย่อยทางซ้ายและทางขวาของทุกโหนด มีค่าแตกต่างกันไม่เกิน 1 ต้นไม้ในรูปที่ 6-6 (b) เป็นต้นไม้ทวิภาคแบบไม่สมดุลใน 2 ตำแหน่ง คือ ตำแหน่งที่มีโหนด 67 และตำแหน่งที่มีโหนด 48 เป็นโหนดราก เนื่องจากความสูงของต้นไม้ย่อยทางซ้ายและต้นไม้ย่อยทางขวาของต้นไม้ย่อยที่มีโหนด 67 เป็นโหนดราก มีค่าแตกต่างกันอยู่ 2 โดยความสูงของต้นไม้ย่อยทางซ้ายมีค่าเป็น 2 และความสูงของต้นไม้ย่อยทางขวามีค่าเป็น 0 ในตำแหน่งที่มีโหนด 48 เป็นโหนดราก ต้นไม้ย่อยทางซ้ายมีความสูงเป็น 1 และต้นไม้ย่อยทางขวาเป็นต้นไม้ว่าง

ต้นไม้ในรูปที่ 6-6 (c) เป็นต้นไม้ทวิภาคแบบไม่สมดุลเช่นกัน โดยมีความไม่สมดุลในตำแหน่งที่มีโหนด N, โหนด A, โหนด R และโหนด O เป็นโหนดราก ด้วยความแตกต่างระหว่างความสูงของต้นไม้ย่อยทางซ้ายและทางขวามีค่าเป็น 2, 3, 4 และ 4 ตามลำดับ



รูปที่ 6-6 (a) ต้นไม้ทวิภาคแบบสมดุลและ (b), (c) ต้นไม้ทวิภาคแบบไม่สมดุล

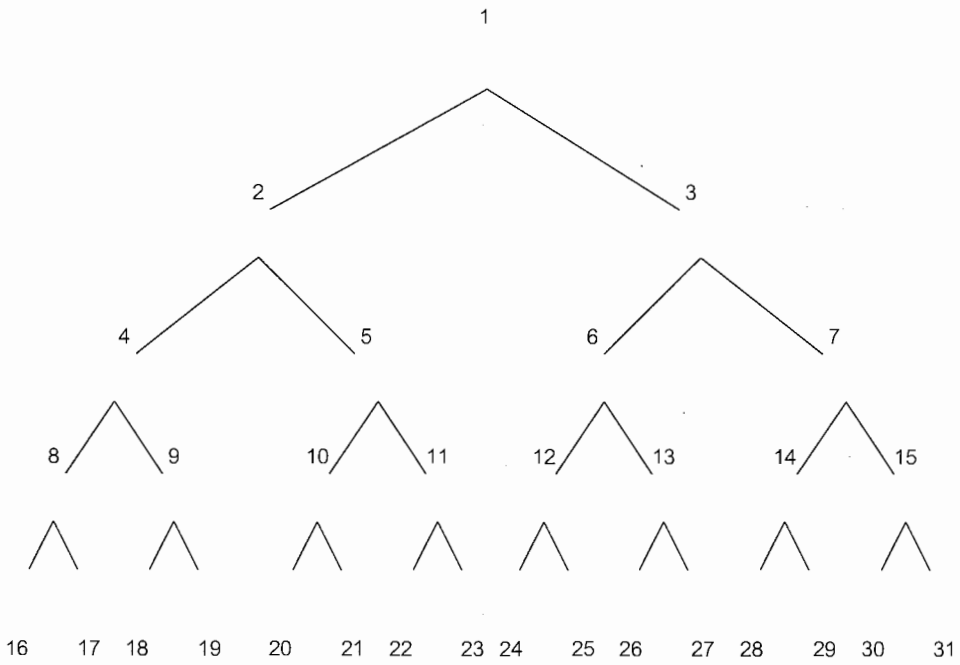
6.3.2 โครงสร้างข้อมูลของต้นไม้ทวิภาค

ในที่นี้เราจะกล่าวถึงวิธีในการประกาศโครงสร้างข้อมูลของต้นไม้ทวิภาคแบบต่างๆ ซึ่งประกอบด้วย

- โครงสร้างข้อมูลต้นไม้ทวิภาคแบบหน่วยเก็บต่อเนื่อง (contiguous storage)
- โครงสร้างข้อมูลต้นไม้ทวิภาคแบบแถวลำดับ (array-based)
- โครงสร้างข้อมูลต้นไม้ทวิภาคแบบตัวชี้ (pointer-based)

โครงสร้างข้อมูลต้นไม้ทวิภาคแบบหน่วยเก็บต่อเนื่อง

ใช้แถวลำดับ 1 มิติสำหรับจัดเก็บข้อมูลโหนดต่างๆ ของต้นไม้ทวิภาค โดยกำหนดให้แถวลำดับตำแหน่งที่ 1 เก็บโหนดราก แถวลำดับตำแหน่งที่ 2 และ 3 เก็บโหนดลูกทางซ้ายและโหนดลูกทางขวาของโหนดราก ตามลำดับ ดังรูปที่ 6-7



รูปที่ 6-7 ตำแหน่งของแถวลำดับในต้นไม้ทวิภาคแบบหน่วยเก็บต่อเนื่อง

จากรูปที่ 6-7 เราสามารถคำนวณตำแหน่งของโหนดในต้นไม้ทวิภาคจากตำแหน่งในแถวลำดับได้ ดังนี้

- ตำแหน่งโหนดลูกทางซ้ายของโหนดในแถวลำดับตำแหน่งที่ k ได้แก่ ตำแหน่งที่ 2k ในแถวลำดับ
- ตำแหน่งโหนดลูกทางขวาของโหนดในแถวลำดับตำแหน่งที่ k ได้แก่ ตำแหน่งที่ 2k+1 ในแถวลำดับ
- ตำแหน่งโหนดพ่อแม่ของโหนดในแถวลำดับตำแหน่งที่ k ได้แก่ ตำแหน่งที่ $\lfloor k/2 \rfloor$ ในแถวลำดับ

หมายเหตุ $\lfloor x \rfloor$ ได้แก่ จำนวนเต็มที่มีค่าสูงที่สุดที่น้อยกว่า x

ดังนั้น ตำแหน่งโหนดลูกทางซ้ายและโหนดลูกทางขวาของโหนดในแถวลำดับตำแหน่งที่ 5 ได้แก่ ตำแหน่งที่ 10 และ 11 ในแถวลำดับ และตำแหน่งโหนดพ่อแม่ของโหนดในแถวลำดับตำแหน่งที่ 27 ได้แก่ ตำแหน่งที่ $\lfloor 27/2 \rfloor = 13$ ในแถวลำดับ

การประกาศโครงสร้างข้อมูลแบบหน่วยเก็บต่อเนื่องสำหรับต้นไม้ทวิภาคที่มีจำนวนโหนดไม่เกิน 31 โหนด ตามรูปที่ 6-7 สามารถทำได้ ดังนี้

```
#define MAXNODES 32
int t[MAXNODES];
```

เนื่องจากแถวลำดับในตำแหน่งที่ 0 ไม่ได้ใช้เก็บข้อมูลของโหนด เราจึงสามารถใช้ตำแหน่งดังกล่าวเก็บจำนวนโหนดที่มีข้อมูลของต้นไม้ ดังรูปที่ 6-8 ซึ่งแสดงโครงสร้างแบบหน่วยเก็บต่อเนื่องของต้นไม้ทวิภาค t ตามรูปที่ 6-6 (b) ซึ่งมีทั้งหมด 9 โหนด

0	1	2	3	4	5	6	7	8	9	...	12	...	16	...	31
9	25	67	48	54	12	21		66		...	43	...	95	...	

รูปที่ 6-8 โครงสร้างแบบหน่วยเก็บต่อเนื่องของต้นไม้ทวิภาค t ตามรูป 6.6 (b)

โครงสร้างข้อมูลต้นไม้ทวิภาคแบบแถวลำดับ

ในการใช้โครงสร้างข้อมูลต้นไม้ทวิภาคแบบหน่วยเก็บต่อเนื่องในรูปของแถวลำดับ 1 มิติ เพื่อจัดเก็บข้อมูลต้นไม้ทวิภาค ทำให้เราสามารถมองโครงสร้างแบบลำดับขั้นของต้นไม้ทวิภาคในรูปของโครงสร้างแบบเชิงเส้น ซึ่งสะดวกต่อการเขียนโปรแกรม นอกจากนี้ การคำนวณหาตำแหน่งโหนดลูกทางซ้ายและขวา รวมทั้งโหนดพ่อแม่ก็สามารถทำได้อย่างรวดเร็ว เนื่องจาก ทุกโหนดในต้นไม้มีตำแหน่งที่แน่นอนในแถวลำดับ อย่างไรก็ตาม การกระทำดังกล่าวจะทำให้สิ้นเปลืองพื้นที่ในแถวลำดับในกรณีที่โหนดของต้นไม้ทวิภาคส่วนใหญ่เป็นโหนดว่าง หรือเป็นโหนดที่ไม่มีข้อมูล ดังรูปที่ 6-8 จะเห็นได้ว่าจำนวนโหนดที่มีข้อมูลมีอยู่เพียง 9 โหนด และอีก 22 โหนดที่เหลือเป็นโหนดว่าง

ในที่นี้จะนำเสนอการใช้โครงสร้างแบบที่สองสำหรับจัดเก็บต้นไม้ทวิภาค ซึ่งใช้แถวลำดับ 1 มิติในการเก็บโหนดต่างๆ ในต้นไม้ทวิภาคเช่นกัน เพียงแต่ไม่มีการกำหนดตำแหน่งที่แน่นอนให้กับโหนดใดๆ ในต้นไม้ทวิภาค ไม่ยกเว้นแม้กระทั่งโหนดราก ดังนั้นจึงจำเป็นที่จะต้องเก็บตำแหน่งของโหนดราก ตำแหน่งโหนดลูกทางซ้ายและตำแหน่งโหนดลูกทางขวาของแต่ละโหนด เพื่อให้สามารถทราบโครงสร้างของโหนดภายในต้นไม้ทวิภาคได้

การประกาศโครงสร้างข้อมูลแบบแถวลำดับสำหรับต้นไม้ทวิภาคที่มีจำนวนโหนดไม่เกิน 31 โหนด ตามรูปที่ 6-7 สามารถทำได้ ดังนี้

```
#define MAXNODES  31
typedef struct {
     int left;
     int data;
     int right;
} node;
typedef struct {
     int rootIndex;
     int freeListIndex;
     node table[MAXNODES];
} binaryTree;
binaryTree t;
```

จากโครงสร้างนี้ `t.table[MAXNODES]` เป็นแถวลำดับ 1 มิติ ขนาด `MAXNODES` ใช้จัดเก็บลิสต์ 2 ลิสต์ด้วยกัน คือ

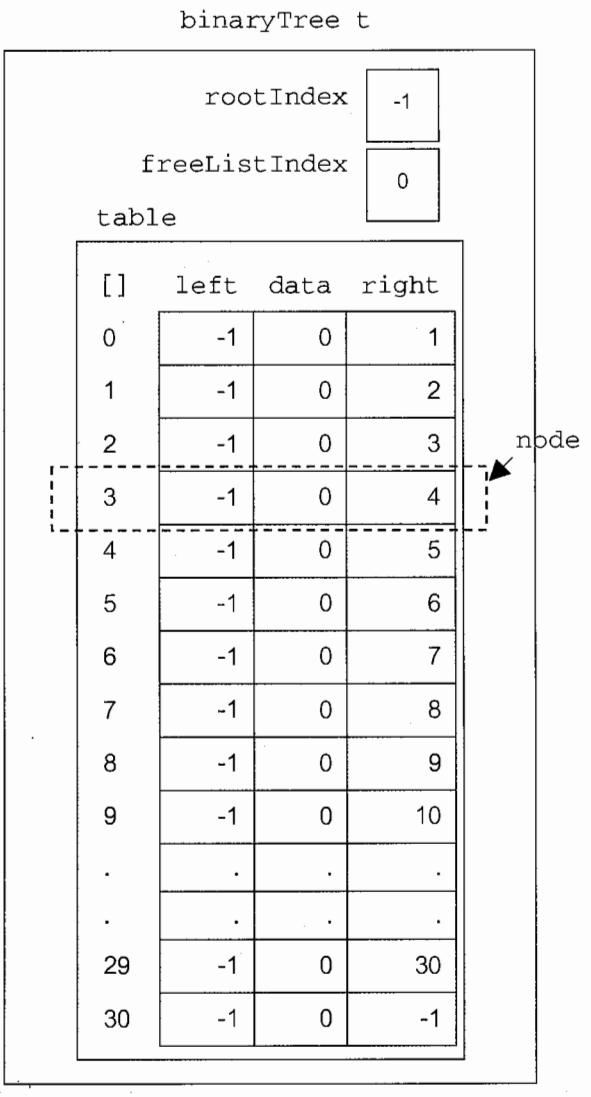
- ลิสต์ของโหนดที่มีข้อมูลในต้นไม้ทวิภาค ซึ่งมี `t.rootIndex` เป็นตำแหน่งแรกของลิสต์ และ
- ลิสต์ของโหนดว่างในต้นไม้ทวิภาค ซึ่งมี `t.freeListIndex` เป็นตำแหน่งแรกของลิสต์

เมื่อ `t.rootIndex` และ `t.freeListIndex` มีค่าเป็น -1 แสดงว่าลิสต์ของโหนดที่มีข้อมูล และลิสต์ของโหนดว่างเป็นลิสต์ว่าง ตามลำดับ และเมื่อ `t.rootIndex` มีค่าเป็น `i` โดย $0 \leq i \leq 30$ เราจะได้ว่า `i` เป็นตำแหน่งของโหนดรากของต้นไม้ทวิภาค `t`

`t.table[i].data` ใช้สำหรับเก็บข้อมูลของโหนด `i` ส่วน `t.table[i].left` และ `t.table[i].right` ใช้สำหรับเก็บตำแหน่งโหนดลูกทางซ้ายและตำแหน่งโหนดลูกทางขวาของโหนด `i` ถ้า

`t.table[i].left` และ `t.table[i].right` มีค่าเป็น -1 แสดงว่าโหนด `i` ไม่มีโหนดลูกทางซ้าย และโหนดลูกทางขวา ตามลำดับ

รูปที่ 6-9 แสดงโครงสร้างข้อมูลแบบแถวลำดับของต้นไม้ทวิภาค `t` โดยเมื่อเริ่มต้นทำงาน ต้นไม้ทวิภาค `t` มีสถานะเป็นต้นไม้ว่าง จึงมีลิสต์ของโหนดที่มีข้อมูลเป็นลิสต์ว่าง นั่นคือ `t.rootIndex` มีค่าเท่ากับ -1 และลิสต์ของโหนดว่างเป็นลิสต์ไม่ว่าง โดยโหนดแรกของลิสต์ของโหนดว่าง ได้แก่ โหนดตำแหน่งที่ 0 ดังนั้น `t.freeListIndex` จึงมีค่าเท่ากับ 0 โหนดแต่ละโหนดในลิสต์นี้เชื่อมโยงกันผ่านค่า `right` (หรืออาจเชื่อมโยงผ่านค่า `left` แทนได้) ในที่นี้ค่า `right` ของโหนดแรกของลิสต์ (`t.table[freeListIndex].right`) มีค่าเป็น 1 ซึ่งเป็นตำแหน่งของโหนดว่างโหนดถัดไป นั่นคือ โหนดในตำแหน่งที่ 1 ในทำนองเดียวกันที่ค่า `right` ของโหนดในตำแหน่งที่ 1 เชื่อมโยงไปยังโหนดว่างถัดไป คือ โหนดในตำแหน่งที่ 2 และเป็นเช่นนี้จนถึงโหนดว่างโหนดสุดท้ายในต้นไม้ทวิภาค คือ โหนดในตำแหน่งที่ 30 ซึ่งเป็นโหนดที่มีค่า `right` เป็น -1 จะเห็นได้ว่า ในสถานะของต้นไม้ว่าง ลิสต์ของโหนดว่างเป็นลิสต์ที่มีจำนวนโหนดเท่ากับจำนวนโหนดทั้งหมดของต้นไม้ทวิภาค



[]	left	data	right
0	-1	0	1
1	-1	0	2
2	-1	0	3
3	-1	0	4
4	-1	0	5
5	-1	0	6
6	-1	0	7
7	-1	0	8
8	-1	0	9
9	-1	0	10
.	.	.	.
.	.	.	.
29	-1	0	30
30	-1	0	-1

รูปที่ 6-9 โครงสร้างแบบแถวลำดับของต้นไม้ทวิภาค `t` ในสถานะของต้นไม้ว่าง

เมื่อมีการเพิ่มโหนดตามรูปที่ 6-6 (b) เข้าไปในต้นไม้ทวิภาค t ทีละโหนด เริ่มจากโหนด 25, 67, 48, 21 และ 54 เราจะได้สถานะของต้นไม้ทวิภาค t ดังรูปที่ 6-10 (a), (b), (c), (d) และ (e) ตามลำดับ ส่วนรูปที่ 6-10 (f) แสดงสถานะของต้นไม้ทวิภาค t หลังจากเพิ่มโหนด 12, 43, 66 และ 95 โดยพื้นที่แรเงาในรูป หมายถึง ข้อมูลที่มีการเปลี่ยนแปลงค่าเนื่องจากการเพิ่มโหนด

รูปที่ 6-10 (a) เพิ่มโหนด 25 เป็นโหนดรากของต้นไม้ทวิภาค t

รูปที่ 6-10 (b) เพิ่มโหนด 67 เป็นโหนดลูกทางซ้ายของโหนดราก

binaryTree t

rootIndex 0

freeListIndex 1

table

[]	left	data	right
0	-1	25	-1
1	-1	0	2
2	-1	0	3
3	-1	0	4
4	-1	0	5
5	-1	0	6
6	-1	0	7
7	-1	0	8
8	-1	0	9
9	-1	0	10
.	.	.	.
.	.	.	.
29	-1	0	30
30	-1	0	-1

(a)

binaryTree t

rootIndex 0

freeListIndex 2

table

[]	left	data	right
0	1	25	-1
1	-1	67	-1
2	-1	0	3
3	-1	0	4
4	-1	0	5
5	-1	0	6
6	-1	0	7
7	-1	0	8
8	-1	0	9
9	-1	0	10
.	.	.	.
.	.	.	.
29	-1	0	30
30	-1	0	-1

(b)

รูปที่ 6-10 ต้นไม้ทวิภาค t หลังจาก (a) เพิ่มโหนด 25 (b) เพิ่มโหนด 67

รูปที่ 6-10 (c) เพิ่มโหนด 48 เป็นโหนดลูกทางขวาของโหนดราก

รูปที่ 6-10 (d) เพิ่มโหนด 21 เป็นโหนดลูกทางซ้ายของโหนด 48

binaryTree t

rootIndex 0

freeListIndex 3

table

[]	left	data	right
0	1	25	2
1	-1	67	-1
2	-1	48	-1
3	-1	0	4
4	-1	0	5
5	-1	0	6
6	-1	0	7
7	-1	0	8
8	-1	0	9
9	-1	0	10
.	.	.	.
.	.	.	.
29	-1	0	30
30	-1	0	-1

(c)

binaryTree t

rootIndex 0

freeListIndex 4

table

[]	left	data	right
0	1	25	2
1	-1	67	-1
2	3	48	-1
3	-1	21	-1
4	-1	0	5
5	-1	0	6
6	-1	0	7
7	-1	0	8
8	-1	0	9
9	-1	0	10
.	.	.	.
.	.	.	.
29	-1	0	30
30	-1	0	-1

(d)

รูปที่ 6-10 ต้นไม้ทวิภาค t หลังจาก (c) เพิ่มโหนด 48 (d) เพิ่มโหนด 21

รูปที่ 6-10 (e) เพิ่มโหนด 54 เป็นโหนดลูกทางซ้ายของโหนด 67

รูปที่ 6-10 (f) เพิ่มโหนด 12 เป็นโหนดลูกทางขวาของโหนด 67, เพิ่มโหนด 43 เป็นโหนดลูกทางซ้ายของโหนด 21, เพิ่มโหนด 66 เป็นโหนดลูกทางซ้ายของโหนด 54 และเพิ่มโหนด 95 เป็นโหนดลูกทางซ้ายของโหนด 66

binaryTree t

rootIndex 0

freeListIndex 5

table

[]	left	data	right
0	1	25	2
1	4	67	-1
2	3	48	-1
3	-1	21	-1
4	-1	54	-1
5	-1	0	6
6	-1	0	7
7	-1	0	8
8	-1	0	9
9	-1	0	10
.	.	.	.
.	.	.	.
29	-1	0	30
30	-1	0	-1

(e)

binaryTree t

rootIndex 0

freeListIndex 9

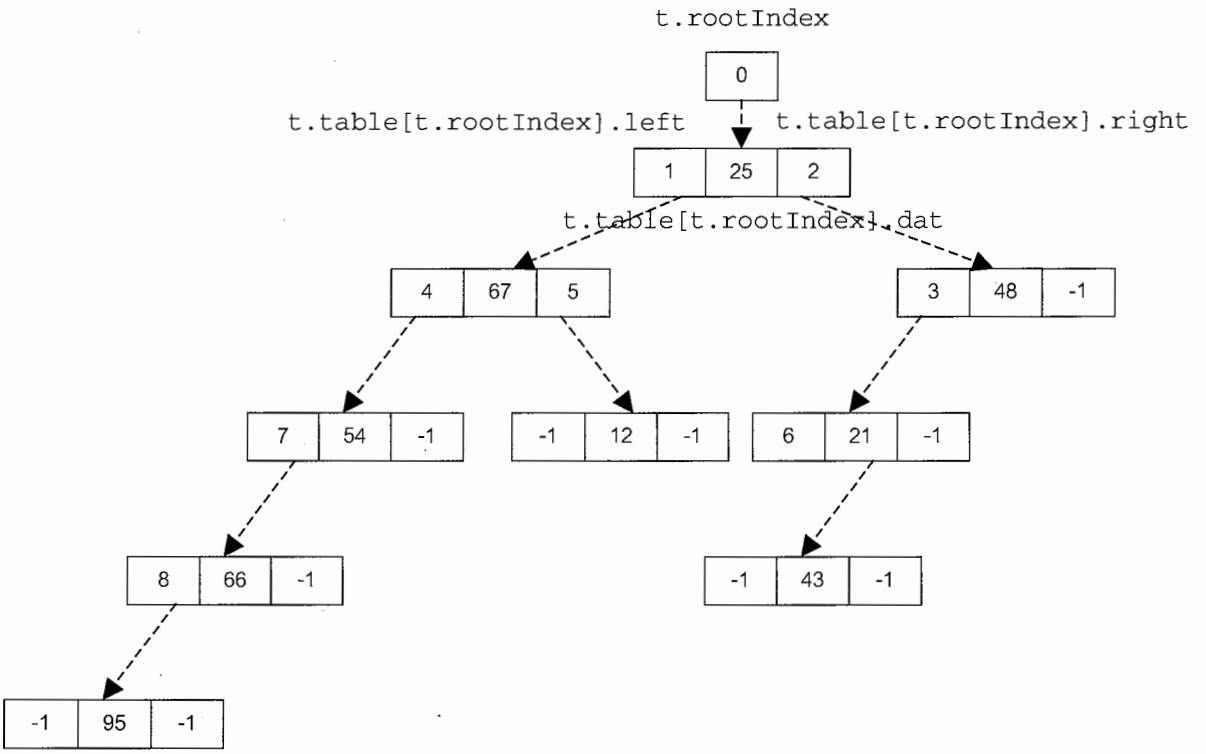
table

[]	left	data	right
0	1	25	2
1	4	67	5
2	3	48	-1
3	6	21	-1
4	7	54	-1
5	-1	12	-1
6	-1	43	-1
7	8	66	-1
8	-1	95	-1
9	-1	0	10
.	.	.	.
.	.	.	.
29	-1	0	30
30	-1	0	-1

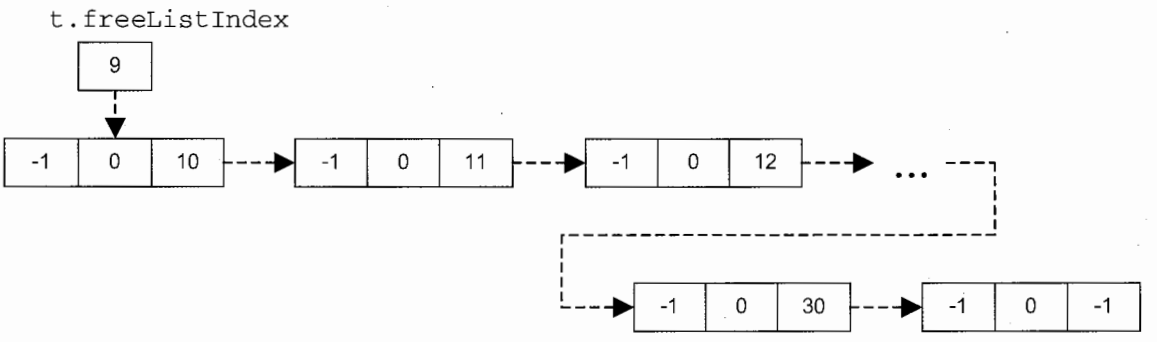
(f)

รูปที่ 6-10 ต้นไม้ทวิภาค t หลังจาก (e) เพิ่มโหนด 54 (f) เพิ่มโหนด 12, 43, 66 และ 95

รูปที่ 6-11 แสดงโครงสร้างแบบลำดับชั้นของลิสต์ของโหนดที่มีข้อมูลในต้นไม้ทวิภาค t ตามสถานะของรูปที่ 6-10 (f) ซึ่งมี t.rootIndex เก็บตำแหน่งของโหนดราก และรูปที่ 6-12 แสดงโครงสร้างเชิงเส้นของลิสต์ของโหนดว่างในต้นไม้ทวิภาค t ซึ่งมี t.freeListIndex เก็บตำแหน่งของโหนดแรกของลิสต์



รูปที่ 6-11 โครงสร้างแบบลำดับชั้นของต้นไม้ทวิภาค t ที่มีการจัดเก็บข้อมูลตามรูปที่ 6-10 (f)



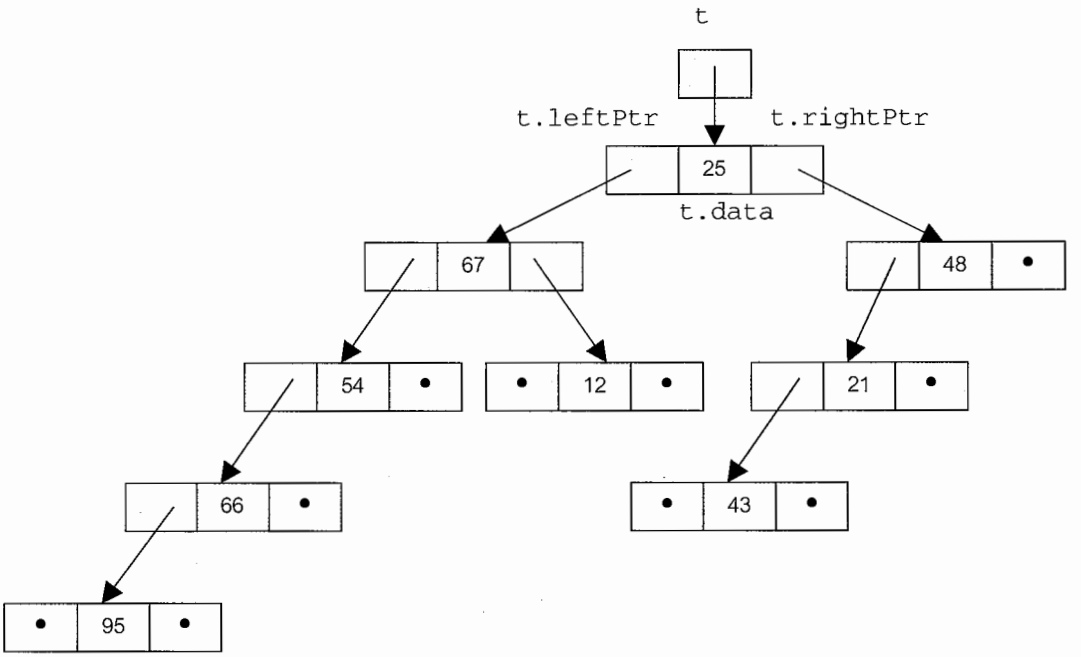
รูปที่ 6-12 โครงสร้างเชิงเส้นของลิสต์ของโหนดว่างของต้นไม้ทวิภาค t ที่มีการจัดเก็บข้อมูลตามรูปที่ 6-10 (f)

โครงสร้างข้อมูลต้นไม้ทวิภาคแบบตัวชี้

ในการสร้างต้นไม้ทวิภาคโดยใช้โครงสร้างข้อมูลทั้งสองแบบที่กล่าวมาแล้ว ซึ่งใช้แถวลำดับเป็นโครงสร้างพื้นฐานนั้น ทำให้จำนวนโหนดทั้งหมดของต้นไม้ทวิภาคต้องถูกกำหนดล่วงหน้า ด้วยขนาดของแถวลำดับ อย่างไรก็ตาม เพื่อให้จำนวนโหนดทั้งหมดของต้นไม้ทวิภาคมีการเปลี่ยนแปลงไปตามการทำงานที่เกิดขึ้นจริง ในที่นี้จะนำเสนอโครงสร้างข้อมูลในอีกรูปแบบหนึ่ง ซึ่งใช้ข้อมูลชนิดตัวชี้ประกอบในการทำงาน ดังนี้

```
#define MAXNODES  100
struct node {
     struct node   *leftPtr;
     int           data;
     struct node   *rightPtr;
};
typedef struct node BTNode;
typedef BTNode *binaryTree;

binaryTree t;
```



รูปที่ 6-13 โครงสร้างแบบตัวชี้ของต้นไม้ทวิภาค t ตามรูปที่ 6-6 (b) เมื่อสัญลักษณ์ • แทนค่า NULL

เช่นเดียวกับโครงสร้างข้อมูลต้นไม้ทวิภาคแบบแถวลำดับ โครงสร้างข้อมูลต้นไม้ทวิภาคแบบตัวชี้ไม่กำหนดตำแหน่งที่แน่นอนของโหนด เพียงแต่ตำแหน่งของโหนดในกรณีหลังกำหนดผ่านตัวชี้ ซึ่งได้แก่ตำแหน่งของโหนดบนหน่วยความจำ ขณะที่ในกรณีแรกกำหนดผ่านตำแหน่งภายในแถวลำดับ

6.3.3 การดำเนินการของต้นไม้ทวิภาค

การดำเนินการของต้นไม้ทวิภาค ได้แก่ การดำเนินการพื้นฐานต่างๆ ที่กระทำกับข้อมูลในโครงสร้างข้อมูลที่ได้กล่าวมาแล้วในบทก่อนๆ ซึ่งประกอบด้วย

1. การเพิ่ม/ลบโหนด
2. การค้นหาโหนดที่มีข้อมูลที่ระบุ
3. การพิมพ์ค่าข้อมูลของทุกโหนด

อัลกอริทึมสำหรับการดำเนินการแต่ละอย่างนั้น ผู้เขียนโปรแกรมต้องคำนึงถึงตำแหน่งของโหนดภายในต้นไม้ทวิภาค ซึ่งขึ้นกับโครงสร้างของต้นไม้ทวิภาคตามที่ได้กล่าวมาแล้วทั้ง 3 แบบ

6.3.4 การท่องไปในต้นไม้ทวิภาค

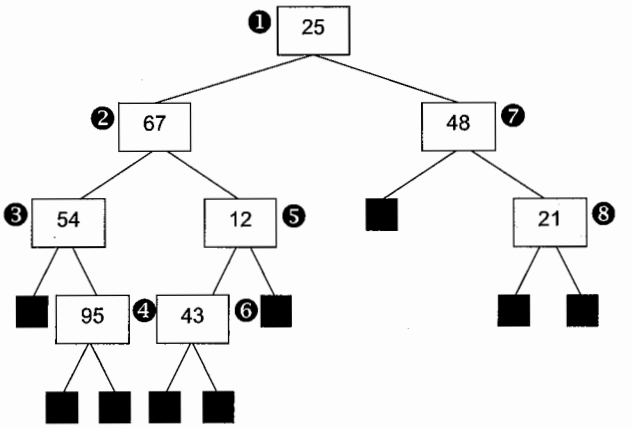
การท่องไปในต้นไม้ทวิภาค (binary tree traversal) หมายถึงการเดินทางไปตามโหนดต่างๆ ภายในต้นไม้ทวิภาคเพื่อดำเนินการอย่างใดอย่างหนึ่งกับข้อมูลที่จัดเก็บอยู่ภายในโหนดนั้นๆ เช่น เพื่อพิมพ์ เพื่อปรับปรุง หรือเพื่อเรียกดูค่าข้อมูลของโหนด เป็นต้น

ในการท่องไปตามโหนดต่างๆ ทุกโหนดของต้นไม้ทวิภาคจะเริ่มต้นจากโหนดรากเสมอ จากนั้นจึงท่องไปยังโหนดในต้นไม้ย่อยทางซ้าย และโหนดในต้นไม้ย่อยทางขวาของโหนดรากตามลำดับ และเมื่อเดินทางมาถึงยังโหนดรากของต้นไม้ย่อยใดๆ ให้ดำเนินการเช่นเดียวกันนี้ จนกว่าต้นไม้ย่อยทางซ้ายหรือต้นไม้ย่อยทางขวาของโหนด จะเป็นต้นไม้ว่าง

จะเห็นได้ว่า ขั้นตอนการดำเนินการอย่างเดียวกันจะเกิดซ้ำๆ กันที่โหนดรากของต้นไม้ย่อยทุกต้น ดังนั้นจึงสามารถนิยามอัลกอริทึมแบบเวียนบังเกิดในการท่องไปในต้นไม้ทวิภาคได้ดังนี้

```
treeTraverse(binaryTree t):
if not emptyTree(t)
    treeTraverse(left(t))
    treeTraverse(right(t))
```

กำหนดให้ `emptyTree(t)` คืนค่าจริง เมื่อ `t` เป็นต้นไม้ว่าง มิฉะนั้น `emptyTree(t)` คืนค่าเท็จ และ `left(t)` และ `right(t)` คืนค่าตำแหน่งของโหนดรากของต้นไม้ย่อยทางซ้ายของโหนด `t` และตำแหน่งของโหนดรากของต้นไม้ย่อยทางขวาของโหนด `t` ตามลำดับ



รูปที่ 6-14 ลำดับการท่องไปในโหนดของต้นไม้ทวิภาค

จากต้นไม้ทวิภาคในรูปที่ 6-14 ซึ่งใช้สัญลักษณ์ ■ แทนตำแหน่งของโหนดว่าง เราสามารถเขียนลำดับการท่องไปในต้นไม้ได้ดังนี้

❶ แวะเยี่ยมโหนดรากของต้นไม้ทวิภาค ซึ่งคือโหนด 25

❷ เดินทางไปยังโหนดของต้นไม้ย่อยทางซ้ายของโหนด 25 แวะเยี่ยมโหนด 67

❸ เดินทางไปยังโหนดของต้นไม้ย่อยทางซ้ายของโหนด 67 แวะเยี่ยมโหนด 54

✱ เดินทางไปยังโหนดของต้นไม้ย่อยทางซ้ายของโหนด 54 ซึ่งเป็นโหนดว่าง

✶ เดินทางกลับมายังโหนด 54

❹ เดินทางไปยังโหนดของต้นไม้ย่อยทางขวาของโหนด 54 แวะเยี่ยมโหนด 95

✱ เดินทางไปยังโหนดของต้นไม้ย่อยทางซ้ายของโหนด 95 ซึ่งเป็นโหนดว่าง

✶ เดินทางกลับมายังโหนด 95

✱ เดินทางไปยังโหนดของต้นไม้ย่อยทางขวาของโหนด 95 ซึ่งเป็นโหนดว่าง

✶ เดินทางกลับมายังโหนด 95

✶ เดินทางกลับมายังโหนด 54

✶ เดินทางกลับมายังโหนด 67

❺ เดินทางไปยังโหนดของต้นไม้ย่อยทางขวาของโหนด 67 แวะเยี่ยมโหนด 12

❻ เดินทางมาถึงโหนดของต้นไม้ย่อยทางซ้ายของโหนด 12 ซึ่งคือโหนด 43

✱ เดินทางไปยังโหนดของต้นไม้ย่อยทางซ้ายของโหนด 43 ซึ่งเป็นโหนดว่าง

✶ เดินทางกลับมายังโหนด 43

✱ เดินทางไปยังโหนดของต้นไม้ย่อยทางขวาของโหนด 43 ซึ่งเป็นโหนดว่าง

✶ เดินทางกลับมายังโหนด 43

✶ เดินทางกลับมายังโหนด 12

✱ เดินทางไปยังโหนดของต้นไม้ย่อยทางขวาของโหนด12 ซึ่งเป็นโหนดว่าง

✶ เดินทางกลับมายังโหนด 12

- ✶ เดินทางกลับมายังโหนด 67

✶ เดินทางกลับมายังโหนด 25

- ❼ เดินทางไปยังโหนดของต้นไม้ย่อยทางขวาของโหนด 25 แวะเยี่ยมโหนด 48
  - ✱ เดินทางไปยังโหนดของต้นไม้ย่อยทางซ้ายของโหนด 48 ซึ่งเป็นโหนดว่าง
- ✶ เดินทางกลับมายังโหนด 48
  - ❽ เดินทางไปยังโหนดของต้นไม้ย่อยทางขวาของโหนด 48 แวะเยี่ยมโหนด 21
    - ✱ เดินทางไปยังโหนดของต้นไม้ย่อยทางซ้ายของโหนด 21 ซึ่งเป็นโหนดว่าง
  - ✶ เดินทางกลับมายังโหนด 21
    - ✱ เดินทางไปยังโหนดของต้นไม้ย่อยทางขวาของโหนด 21 ซึ่งเป็นโหนดว่าง
  - ✶ เดินทางกลับมายังโหนด 21
- ✶ เดินทางกลับมายังโหนด 48

✶ เดินทางกลับมายังโหนด 25

เราจะพบว่าเมื่อการทำงานเสร็จสิ้นลง ทุกโหนดในต้นไม้ทวิภาคจะถูกแวะเยี่ยมโหนดละ 3 ครั้ง โดยการแวะเยี่ยมโหนดในครั้งที่สองเกิดขึ้นเมื่อได้มีการท่องไปตามโหนดในต้นไม้ย่อยทางซ้ายของโหนดจนครบทุกโหนด และการแวะเยี่ยมโหนดในครั้งที่สามเกิดขึ้นเมื่อได้มีการท่องไปตามโหนดในต้นไม้ย่อยทางขวาของโหนดจนครบทุกโหนด ดังนั้นเราจึงสามารถกำหนดลำดับการดำเนินการกับข้อมูลภายในโหนดเมื่อเดินทางมาถึงโหนดแตกต่างกันได้ 3 ลักษณะ คือ

1. การท่องต้นไม้แบบก่อนลำดับ (preorder tree traversal) – ดำเนินการกับข้อมูลภายในโหนด เมื่อเดินทางมาถึงโหนดในครั้งแรก
2. การท่องต้นไม้แบบตามลำดับ (inorder tree traversal) – ดำเนินการกับข้อมูลภายในโหนด เมื่อเดินทางมาถึงโหนดในครั้งที่สอง
3. การท่องต้นไม้แบบหลังลำดับ (postorder tree traversal) – ดำเนินการกับข้อมูลภายในโหนด เมื่อเดินทางมาถึงโหนดในครั้งสุดท้าย

อัลกอริทึมสำหรับการท่องต้นไม้ทวิภาคทั้งสามแบบ สามารถแสดงได้ดังนี้

```
preorderTraverse(binaryTree t):
if not emptyTree(t)
     doSomething(t)
     preorderTraverse(left(t))
     preorderTraverse(right(t))

inorderTraverse(binaryTree t):
if not emptyTree(t)
     inorderTraverse(left(t))
     doSomething(t)
     inorderTraverse(right(t))
```

```
postorderTraverse(binaryTree t):
if not emptyTree(t)
    postorderTraverse(left(t))
    postorderTraverse(right(t))
    doSomething(t)
```

เมื่อกำหนดให้ `key(t)` คืนค่าข้อมูลของโหนด `t` จะเห็นได้ว่าอัลกอริทึม `doSomething()` ที่กำหนดให้ด้านล่างเป็นการพิมพ์ค่าข้อมูลของโหนด `t` ดังนั้น ผลลัพธ์ที่ได้จากการท่องไปตามโหนดในต้นไม้ทวิภาคในรูปที่ 6-14 จะเป็นดังรูปที่ 6-15

```
doSomething(binaryTree t):
    print(key(t))
```

วิธีการท่องต้นไม้ทวิภาค	ผลลัพธ์
แบบก่อนลำดับ	25, 67, 54, 66, 12, 43, 48, 21
แบบตามลำดับ	66, 54, 67, 43, 12, 25, 48, 21
แบบหลังลำดับ	66, 54, 43, 12, 67, 21, 48, 25

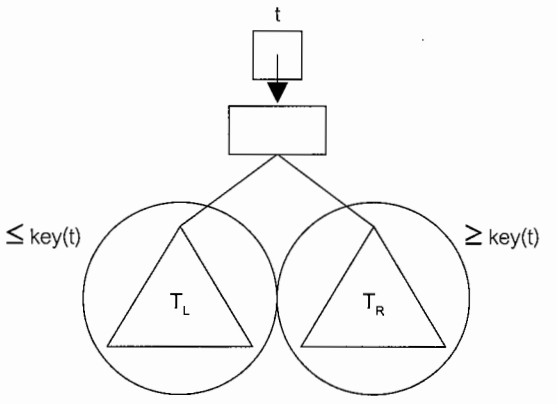
รูปที่ 6-15 ผลลัพธ์จากการท่องไปในต้นไม้ทวิภาคในรูปที่ 6-14 เพื่อพิมพ์ข้อมูลของโหนด

6.4 ต้นไม้ค้นหาทวิภาค

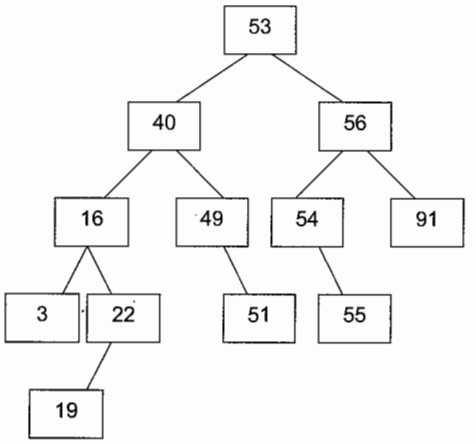
ต้นไม้ค้นหาทวิภาค (binary search tree) ได้แก่ ต้นไม้ทวิภาคที่มีสมบัติ ดังนี้

ถ้าให้โหนด m เป็นโหนดใดๆ ในต้นไม้ค้นหาทวิภาค แล้ว

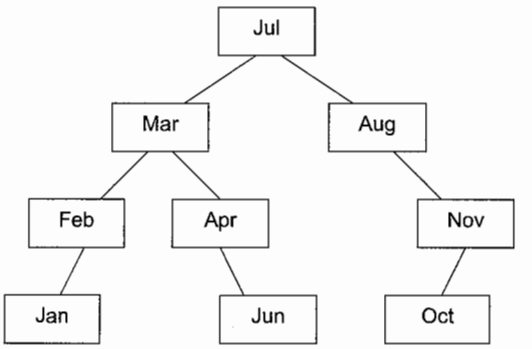
1. ถ้าโหนด n เป็นโหนดใดๆ ในต้นไม้ย่อยทางซ้ายของโหนด m แล้ว key(n) $\leq$ key(m)
2. ถ้าโหนด n เป็นโหนดใดๆ ในต้นไม้ย่อยทางขวาของโหนด m แล้ว key(n) $\geq$ key(m)



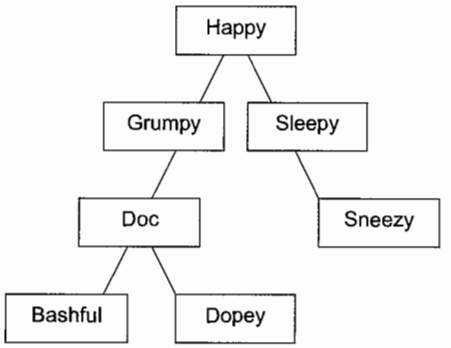
รูปที่ 6-16 โหนดในต้นไม้ค้นหาทวิภาค



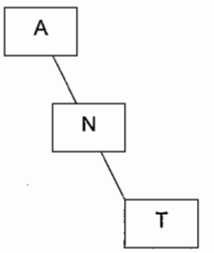
(a)



(b)



(c)



(d)

รูปที่ 6-16 ตัวอย่างของต้นไม้ค้นหาทวิภาค

ถ้าเราท่องไปในต้นไม้ค้นหาทวิภาคในรูปที่ 6-16 (a) ด้วยวิธีการท่องไปในต้นไม้ทวิภาคทั้ง 3 แบบ เพื่อพิมพ์ข้อมูลของโหนด ผลลัพธ์ที่ได้จะเป็นดังรูปที่ 6-17

วิธีการท่องต้นไม้ค้นหาทวิภาค	ผลลัพธ์
แบบก่อนลำดับ	53, 40, 16, 3, 22, 19, 49, 51, 56, 54, 55, 91
แบบตามลำดับ	3, 16, 19, 22, 40, 49, 51, 53, 54, 55, 56, 91
แบบหลังลำดับ	3, 19, 22, 16, 51, 49, 40, 55, 54, 91, 56, 53

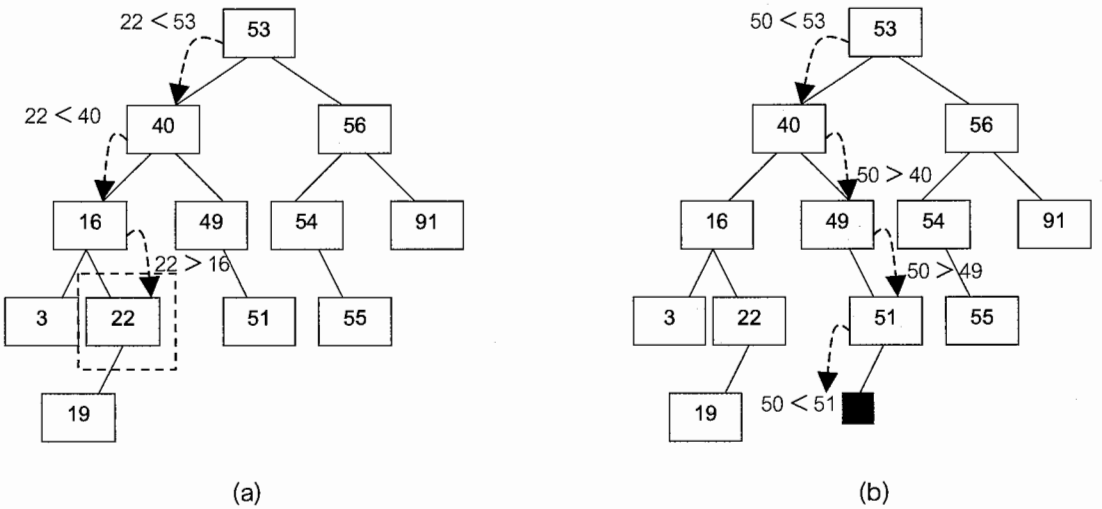
รูปที่ 6-17 ผลลัพธ์จากการท่องไปในต้นไม้ค้นหาทวิภาคในรูปที่ 6-16 (a)

จากรูปที่ 6-17 จะเห็นได้ว่าผลลัพธ์จากการท่องต้นไม้ค้นหาทวิภาคแบบตามลำดับ ได้ข้อมูลที่เรียงลำดับจากน้อยไปหามาก ซึ่งเป็นผลจากสมบัติของโหนดในต้นไม้ค้นหาทวิภาค ซึ่งสอดคล้องกับการดำเนินการค้นหาแบบทวิภาค (binary search) ที่ได้แสดงอัลกอริทึมไว้ด้านล่าง เมื่อกำหนดให้ t เป็นต้นไม้ค้นหาทวิภาค และ k เป็นค่าข้อมูลที่ต้องการค้นหา

```
search(bst t, int k):
if emptyBST(t) or k = key(t)
     then return t
if k < key(t)
     then return search(left(t), k)
     else return search(right(t), k)
```

ในการค้นหาโหนดในต้นไม้ค้นหาทวิภาค จะเริ่มจากการเปรียบเทียบข้อมูลที่ต้องการค้นหากับข้อมูลของโหนดราก ถ้าตรงกัน ตำแหน่งของโหนดรากจะถูกส่งกลับ มิฉะนั้น จะมีการเปรียบเทียบเพิ่มเติมว่าข้อมูลที่ต้องการค้นหาควรจะเป็นโหนดในต้นไม้ย่อยทางซ้าย หรือในต้นไม้ย่อยทางขวาของโหนดราก จากนั้นเรียกให้มีการดำเนินกันเช่นเดิมซ้ำอีกในต้นไม้ย่อยที่เหมาะสม จนกว่าจะพบข้อมูลที่ต้องการค้นหา หรือจนกว่าจะเดินทางไปจนสุดเส้นทางที่เป็นไปได้ที่จะพบข้อมูลที่ต้องการค้นหา แต่ปรากฏว่าไม่พบข้อมูล ซึ่งจะส่งกลับค่าว่าง ดังนั้น จะเห็นได้ว่าในรอบแรกค่า t ที่ส่งมาเป็นตำแหน่งของโหนดรากของต้นไม้ค้นหาทวิภาค และในรอบถัดๆ ไปค่า t เป็นตำแหน่งของโหนดรากของต้นไม้ค้นหาทวิภาคย่อยทางซ้าย หรือทางขวาแล้วแต่กรณี

ถ้าต้องการค้นหาตำแหน่งของโหนดที่มีข้อมูลเป็น 22 จากต้นไม้ค้นหาทวิภาคในรูปที่ 6-18 (a) ลำดับของโหนดที่จะถูกเปรียบเทียบจะเป็นดังนี้ คือ 53, 40, 16 และ 22 ซึ่งพบข้อมูลที่ต้องการ แต่ถ้าต้องการค้นหาตำแหน่งของโหนดที่มีข้อมูลเป็น 50 จากต้นไม้ค้นหาทวิภาคเดียวกัน ลำดับของโหนดที่จะถูกเปรียบเทียบ คือ 53, 40, 49 และ 51 และไม่พบข้อมูลที่ต้องการ เนื่องจากโหนด 51 มีโหนดลูกทางซ้ายเป็นโหนดว่าง ดังแสดงในรูปที่ 6-18 (b) เมื่อสัญลักษณ์ ■ แทนตำแหน่งของโหนดว่าง



รูปที่ 6-18 ลำดับการค้นหาโหนดในต้นไม้ค้นหาทวิภาค (a) ค้นหา 22 (b) ค้นหา 50

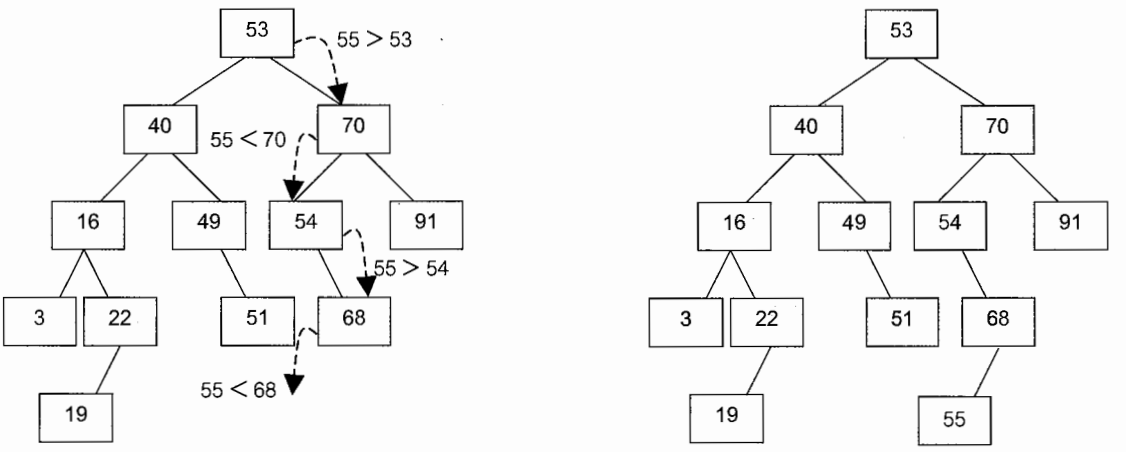
นอกจากการค้นหาข้อมูลในต้นไม้ค้นหาทวิภาค การดำเนินการที่นิยามอยู่บนต้นไม้ทวิภาค เช่น การเพิ่มโหนด การลบโหนด การพิมพ์ค่าข้อมูลของทุกโหนด ก็นิยามอยู่บนต้นไม้ค้นหาทวิภาคเช่นเดียวกัน และยังมีการดำเนินการอื่นๆ อีก เช่น การค้นหาโหนดที่มีค่าข้อมูลต่ำสุดและโหนดที่มีค่าข้อมูลสูงสุด การค้นหาโหนดที่มีค่าข้อมูลลำดับก่อนหน้า และลำดับถัดจากค่าข้อมูลที่กำหนด เป็นต้น สำหรับการดำเนินการที่มีผลให้จำนวนโหนดในต้นไม้ค้นหาทวิภาคเปลี่ยนแปลง

ไป ได้แก่ การเพิ่มโหนดและการลบโหนด ในอัลกอริทึมจะต้องยังคงรักษาสมบัติของโหนดตามนิยามของต้นไม้ค้นหาทวิภาคอยู่เสมอ ในที่นี้จะนำเสนอเพียงอัลกอริทึมการเพิ่มโหนดใหม่ในต้นไม้ค้นหาทวิภาค ซึ่งเป็นการดำเนินการที่คล้ายคลึงกับอัลกอริทึมการค้นหาข้อมูลในต้นไม้ค้นหาทวิภาค เมื่อ m เป็นโหนดใหม่ที่ต้องการเพิ่มเข้าไปในต้นไม้ค้นหาทวิภาค t

```
insert(bst t, node m):
if not empty(t)
     then if key(m) < key(t)
          then insert(left(t), m)
          else insert(right(t), m)
else t = m
```

ในการเพิ่มโหนด m เข้าไปในต้นไม้ค้นหาทวิภาค จะเริ่มจากการเปรียบเทียบเพื่อหาตำแหน่งที่เหมาะสมของโหนด m ในต้นไม้ค้นหาทวิภาค ซึ่งในขั้นตอนนี้จะเป็นไปในทำนองเดียวกับการค้นหาโหนด m ในต้นไม้ค้นหาทวิภาค และเมื่อเดินทางมาถึงโหนดสุดท้ายในเส้นทางการค้นหา สมมติให้เป็นโหนด t โหนด m จะถูกเพิ่มให้เป็นลูกทางซ้าย หรือลูกทางขวาของโหนด t ขึ้นกับข้อมูลของโหนด m เมื่อเปรียบเทียบกับข้อมูลของโหนด t และจะเห็นได้ว่าโหนด m จะถูกเพิ่มเป็นโหนดใบของต้นไม้ค้นหาทวิภาคเสมอ

รูปที่ 6-19 (a) แสดงสถานะของต้นไม้ค้นหาทวิภาคก่อนเพิ่มโหนด 55 และแสดงลำดับของโหนดที่จะถูกเปรียบเทียบ เพื่อค้นหาตำแหน่งของโหนด 55 ซึ่งมีลำดับคือ 53, 70, 54 และ 68 และเนื่องจาก 55 มีค่าน้อยกว่า 68 และโหนด 68 มีโหนดลูกทางซ้ายเป็นโหนดว่าง ดังนั้นโหนด 55 จึงถูกเพิ่มเป็นโหนดลูกทางซ้ายของโหนด 68 ดังแสดงในรูปที่ 6-19 (b)

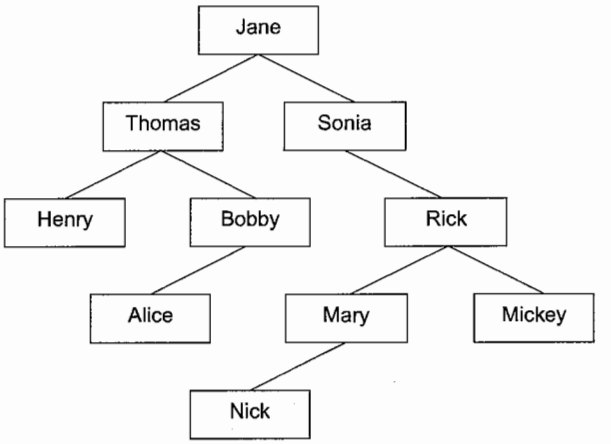


(a) (b)

รูปที่ 6-19 เพิ่มโหนด 55 เข้าไปในต้นไม้ค้นหาทวิภาค

แบบฝึกหัดบทที่ 6

1. จากรูปต้นไม้ที่กำหนดให้ จงเติมคำตอบลงในช่องว่าง



1.1 โหนดราก ได้แก่ _____

1.2 โหนดลูกของโหนด Rick ประกอบด้วย _____

1.3 โหนดพ่อแม่ของโหนด Mickey ได้แก่ _____

1.4 ยกตัวอย่างโหนดที่มีความสัมพันธ์แบบพ่อแม่-ลูก มา 2 ตัวอย่าง _____

1.5 ยกตัวอย่างโหนดที่มีความสัมพันธ์แบบบน-ล่าง มา 2 ตัวอย่าง _____

1.6 โหนดใบ ประกอบด้วย _____

1.7 โหนดภายใน ประกอบด้วย _____

1.8 ระดับของโหนด Jane มีค่าเท่ากับ _____

1.9 ระดับของโหนด Sonia มีค่าเท่ากับ _____

1.10 ระดับของโหนด Henry มีค่าเท่ากับ _____

1.11 ระดับของโหนด Mary มีค่าเท่ากับ _____

1.12 ระดับของโหนด Nick มีค่าเท่ากับ _____

1.13 โหนดรากของต้นไม้ย่อยของโหนด Sonia _____

1.14 โหนดรากของต้นไม้ย่อยของโหนด Bobby _____

1.15 โหนดรากของต้นไม้ย่อยของโหนด Alice _____

1.16 โหนดทั้งหมดของต้นไม้ย่อยของโหนด Thomas _____

1.17 โหนดทั้งหมดของต้นไม้ย่อยของโหนด Mary _____

1.18 ความสูงของต้นไม้ มีค่าเท่ากับ _____

1.19 ความสูงของต้นไม้ย่อยของโหนด Rick มีค่าเท่ากับ _____

1.20 ความสูงของต้นไม้ย่อยของโหนด Thomas มีค่าเท่ากับ _____

1.21 โหนดที่มีดีกรี 1 ประกอบด้วย _____

1.22 โหนดที่มีดีกรี 2 ประกอบด้วย _____

1.23 โหนดที่มีดีกรี 3 ประกอบด้วย _____

1.24 ต้นไม้นี้จัดเป็นต้นไม้ n ดีกรี โดย n มีค่าเท่ากับ _____

2. จงยกตัวอย่างด้วยการวาดรูปต้นไม้ดังต่อไปนี้

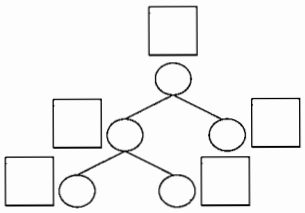
2.1 ต้นไม้ทวิภาคแบบสมบูรณ์

2.2 ต้นไม้ทวิภาคแบบเต็ม

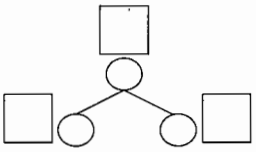
2.3 ต้นไม้ทวิภาคแบบสมดุล

3. จากรูปต้นไม้ทวิภาคที่กำหนดให้ จงเติมค่าความสมดุลของแต่ละโหนดลงในช่องว่าง และพิจารณาว่าต้นไม้ใดเป็นต้นไม้ทวิภาคสมดุล

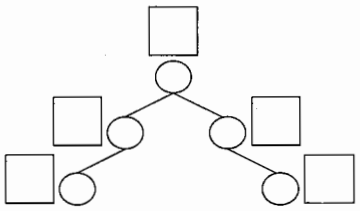
3.1



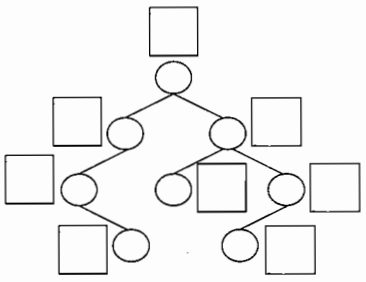
3.2



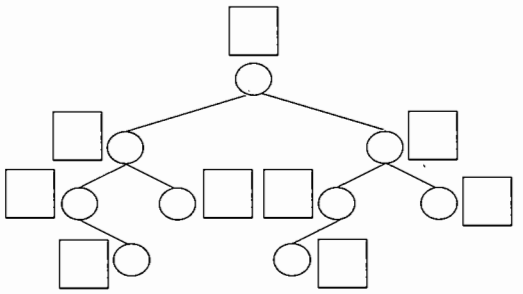
3.3



3.4

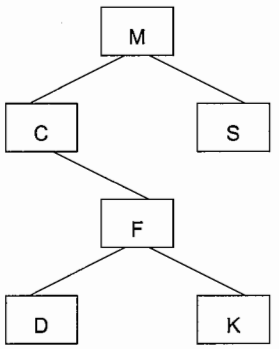


3.5



4. จากรูปต้นไม้ค้นหาทวิภาคที่กำหนดให้ จงเขียนลำดับของการท่องไปในต้นไม้แบบก่อนลำดับ, แบบตามลำดับ และแบบหลังลำดับ

4.1

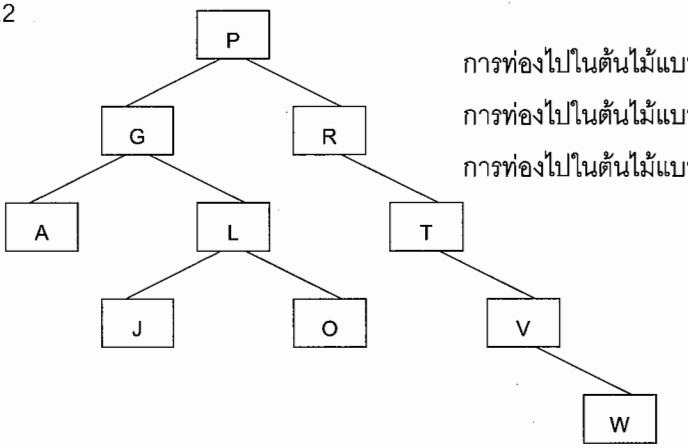


การท่องไปในต้นไม้แบบก่อนลำดับ_____

การท่องไปในต้นไม้แบบตามลำดับ_____

การท่องไปในต้นไม้แบบหลังลำดับ_____

4.2

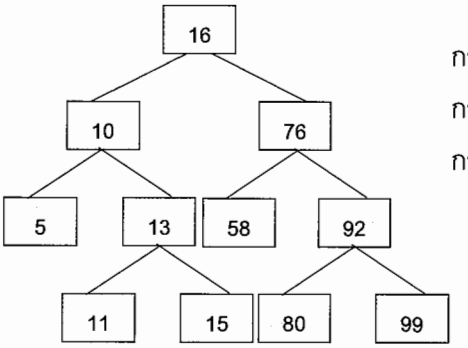


การท่องไปในต้นไม้แบบก่อนลำดับ_____

การท่องไปในต้นไม้แบบตามลำดับ_____

การท่องไปในต้นไม้แบบหลังลำดับ_____

4.3

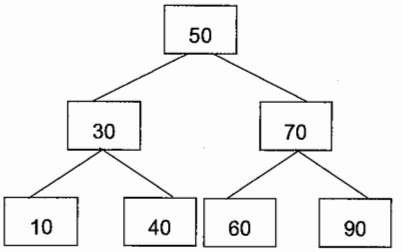


การท่องไปในต้นไม้แบบก่อนลำดับ_____

การท่องไปในต้นไม้แบบตามลำดับ_____

การท่องไปในต้นไม้แบบหลังลำดับ_____

4.4

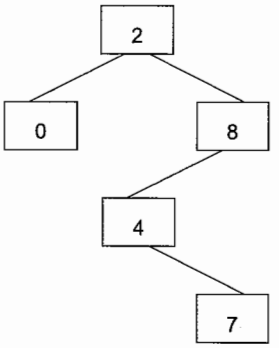


การท่องไปในต้นไม้แบบก่อนลำดับ_____

การท่องไปในต้นไม้แบบตามลำดับ_____

การท่องไปในต้นไม้แบบหลังลำดับ_____

4.5



การท่องไปในต้นไม้แบบก่อนลำดับ_____

การท่องไปในต้นไม้แบบตามลำดับ_____

การท่องไปในต้นไม้แบบหลังลำดับ_____

5. จากรูปต้นไม้ค้นหาทวิภาคในข้อ 4.1, 4.3 และ 4.5 จงบอกค่าข้อมูลที่เป็นไปได้ของโหนดใหม่ที่จะมาเพิ่มในตำแหน่งต่างๆ ดังต่อไปนี้

5.1 จากต้นไม้ค้นหาทวิภาค 4.1

5.1.1 ลูกทางซ้ายของโหนด K _____

5.1.2 ลูกทางขวาของโหนด D _____

5.2 จากต้นไม้ค้นหาทวิภาค 4.3

5.2.1 ลูกทางซ้ายของโหนด 58 _____

5.2.2 ลูกทางขวาของโหนด 11 _____

5.3 จากต้นไม้ค้นหาทวิภาค 4.5

5.3.1 ลูกทางซ้ายของโหนด 0 _____

5.3.2 ลูกทางขวาของโหนด 7 _____

6. จากรูปต้นไม้ค้นหาทวิภาคในข้อ 4.2 และ 4.4 จงแสดงเส้นทางของการเปรียบเทียบโหนดเพื่อค้นหาโหนดต่อไปนี้

6.1 จากต้นไม้ค้นหาทวิภาค 4.2

6.1.1 ค้นหาโหนด I _____

6.1.2 ค้นหาโหนด J _____

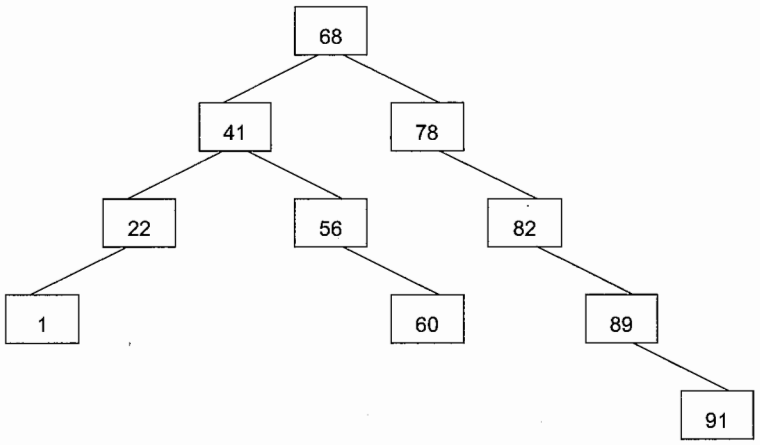
6.1.3 ค้นหาโหนด S _____

6.2 จากต้นไม้ค้นหาทวิภาค 4.4

6.2.1 ค้นหาโหนด 40 _____

6.2.2 ค้นหาโหนด 65 _____

7. จากรูปต้นไม้ค้นหาทวิภาคที่กำหนดให้ จงวาดรูปที่ได้จากการเพิ่มโหนดตามลำดับ คือ 5, 10, 30, 45, 67, 80 และ 105 ลงในต้นไม้ค้นหาทวิภาคดังกล่าว



8. จงวาดรูป (1 รูป ต่อการเพิ่ม 1 โหนด) แสดงการสร้างต้นไม้ค้นหาทวิภาคที่ได้จากการเพิ่มโหนดทีละโหนดตามลำดับดังนี้ 32, 26, 48, 91, 12, 109, 15, 35, 64 และ 100 โดยกำหนดให้เมื่อเริ่มต้นต้นไม้ค้นหาทวิภาคเป็นต้นไม้ว่าง

9. จงแสดงวิธีคำนวณหาจำนวนโหนดใบของต้นไม้ n ดีกรี แบบสมบูรณ์ ที่มีความสูง h

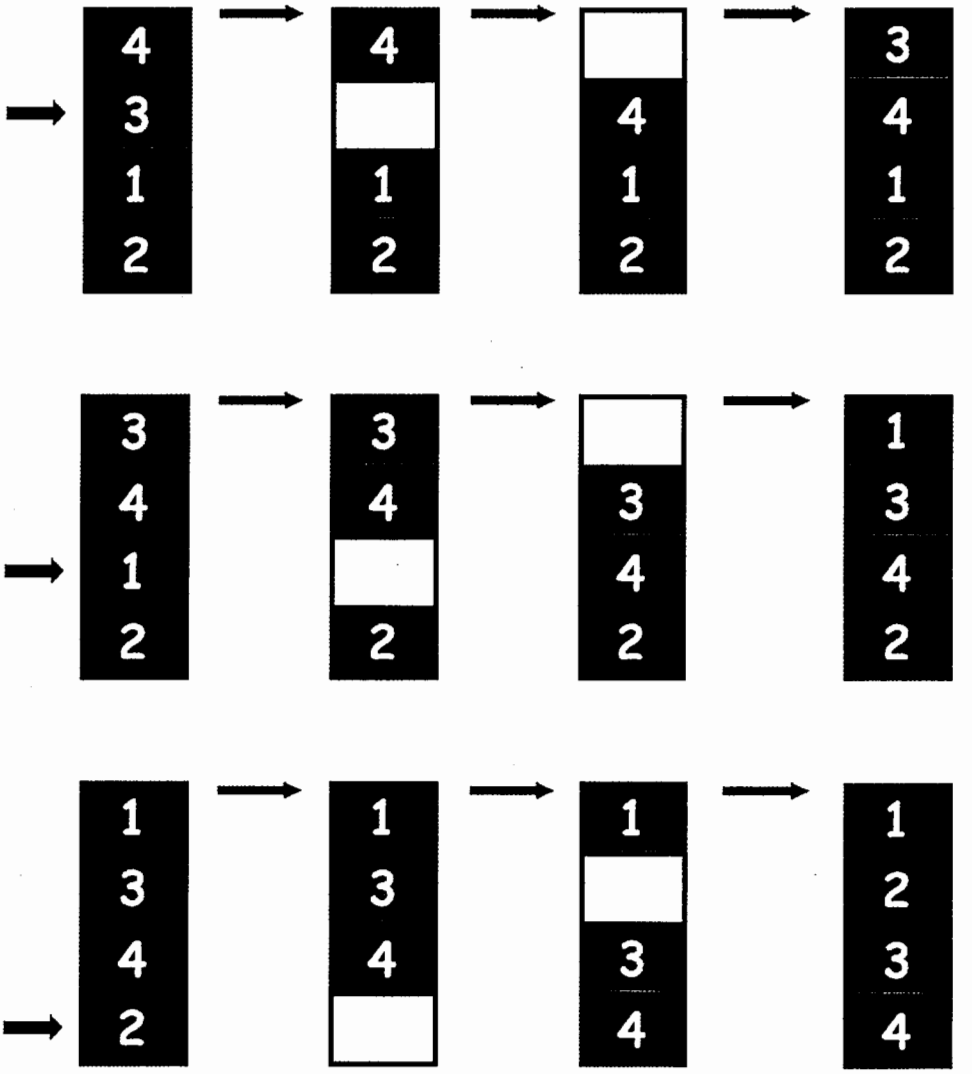
10. จงแสดงวิธีคำนวณหาจำนวนโหนดภายในของต้นไม้ n ดีกรี แบบสมบูรณ์ ที่มีความสูง h

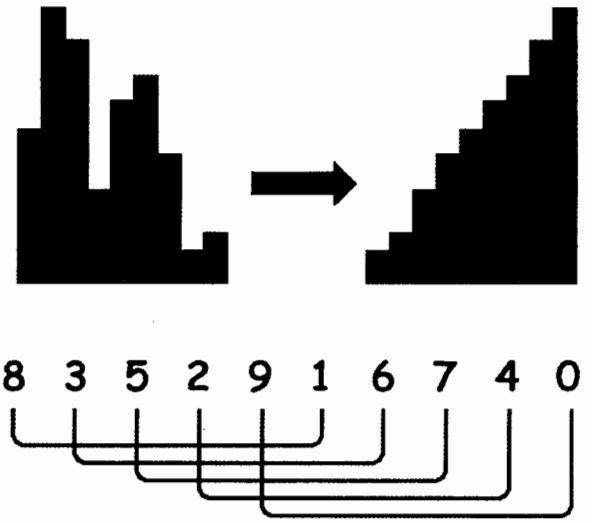
11. จงแสดงวิธีคำนวณหาความสูงของต้นไม้ n ดีกรี แบบสมบูรณ์ ที่มีจำนวนโหนดใบเท่ากับ k

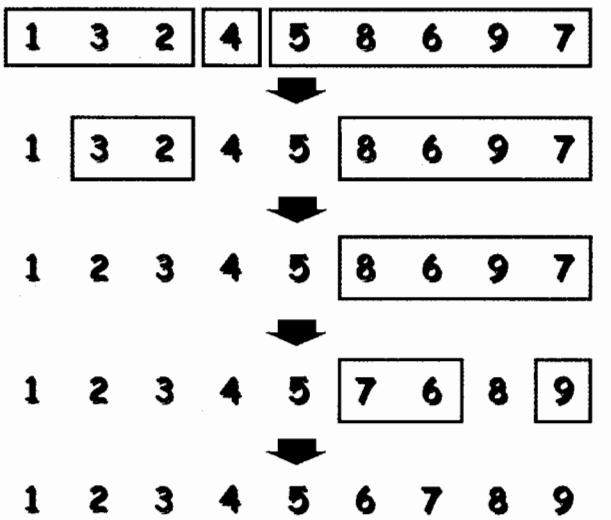
12. จงเขียนโปรแกรมเพื่อนับจำนวนโหนดทั้งหมดในต้นไม้ทวิภาค

13. จงเขียนโปรแกรมเพื่อนับจำนวนโหนดในต้นไม้ค้นหาทวิภาคที่มีค่าข้อมูลของโหนดน้อยกว่า k เมื่อ k เป็นค่าข้อมูลใดๆ ของโหนด

14. จงเขียนโปรแกรมเพื่อเพิ่มโหนด ลบโหนดและพิมพ์ข้อมูลของโหนดของต้นไม้ทวิภาค โดยใช้โครงสร้าง 3 แบบ คือ
    14.1 แบบหน่วยเก็บต่อเนื่อง
    14.2 แบบแถวลำดับ
    14.3 แบบตัวชี้


4 3 1 2
4 1 2
4 1 2
3 4 1 2
3 4 2
3 4 2
1 3 4 2
1 3 4
1 3 4
1 2 3 4


8 3 5 2 9 1 6 7 4 0


1 3 2 4 5 8 6 9 7
1 3 2 4 5 8 6 9 7
1 2 3 4 5 8 6 9 7
1 2 3 4 5 7 6 8 9
1 2 3 4 5 6 7 8 9

บทที่ 7

อัลกอริทึมการเรียงลำดับ

ในวิชาสาขาวิทยาการคอมพิวเตอร์และวิชาสาขาคณิตศาสตร์ อัลกอริทึมการเรียงลำดับ (Sorting algorithm) เป็นวิธีการหรือกระบวนการคิดในการจัดลำดับข้อมูลโดยคำนึงถึงความถูกต้อง และความรวดเร็วของการได้มาซึ่งผลลัพธ์ ทั้งนี้ข้อมูลผลลัพธ์ที่ได้จะถูกเรียงลำดับ คล้ายการรวบรวมพจนานุกรมที่มีการจัดเรียงลำดับตามอักขระ อัลกอริทึมการเรียงลำดับสามารถประยุกต์ใช้กับโครงสร้างข้อมูล (Data structure) ได้หลายแบบ เช่น แถวลำดับ(Array) และรายการ (List)

อัลกอริทึมการเรียงลำดับสามารถทำได้หลายวิธีด้วยกัน เช่น การเรียงลำดับแบบรวดเร็ว (Quick Sort) ในที่นี้จะขอยกตัวอย่างอัลกอริทึมในการเรียงลำดับแบบง่ายไว้ 4 แบบคือ

1. การเรียงลำดับแบบแทรก (Insertion Sort)
2. การเรียงลำดับแบบเลือก (Selection Sort)
3. การเรียงลำดับแบบฟอง (Bubble Sort)
4. การเรียงลำดับแบบผสาน (Merge Sort)

7.1 การเรียงลำดับแบบแทรก (Insertion Sort)

การเรียงลำดับแบบแทรกเป็น การเรียงลำดับที่จัดเรียงข้อมูลโดยเริ่มจากการนำข้อมูลสมาชิกตัวแรกของชุดข้อมูลนำเข้ามาในตำแหน่งแรกข้อผลลัพธ์ก่อน แล้วเพิ่มสมาชิกตัวถัดไปเข้าไปในตำแหน่งที่เหมาะสมโดยการแทรกข้อมูลนำเข้าดังกล่าวเมื่อเปรียบเทียบกับสมาชิกตัวก่อนหน้า ทำอย่างนี้วนซ้ำจนกระทั่งข้อมูลนำเข้าทั้งหมดถูกนำมาจัดเรียง โดยในแต่ละการทำซ้ำของการเรียงลำดับแบบแทรก อัลกอริทึมจะทำการย้ายข้อมูลที่เพิ่มเข้าไป แทรกข้อมูลนั้นลงที่ตำแหน่งที่ถูกต้องในแถวลำดับที่เรียงลำดับแล้ว และกระทำการวนซ้ำรับข้อมูลนำเข้าจนกระทั่งข้อมูลทุกตัวได้รับการจัดเรียงจนครบ

ขั้นตอนการเรียงลำดับแบบแทรก คือ

1. นำข้อมูลนำเข้าตัวแรกเข้าสู่ชุดข้อมูลผลลัพธ์
2. อ่านข้อมูลนำเข้าเป็นสมาชิกตัวถัดไป
3. เปรียบเทียบข้อมูลนำเข้ากับผลลัพธ์ปัจจุบัน
4. แทรกข้อมูลนำเข้าในตำแหน่งที่เหมาะสม
5. วนซ้ำไปขั้นตอนที่ 2 สำหรับข้อมูลนำเข้าตัวถัดไป

ตัวอย่าง 7-1 โปรแกรมการเรียงลำดับแบบแทรก ที่เขียนด้วยภาษาซี

```
#include <stdio.h>
#include <conio.h>
main()
{
     const last = 9 ;
     int current; int located; int hold;
     int walker; int i; int list[last];
     clrscr();
     printf("Enter %d sorting numbers \n ",last);
     for (i = 0; i < last; i++)      scanf("%d", &list[i]);
     clrscr();
     printf("Sorting Number : \n\n");
     for (i = 0; i < last; i++)      printf("%d ",list[i]);
     printf("\n\n");
     // ----------- เริ่มต้นการเรียงลำดับแบบแทรก -------------------------
     for (current = 1; current <= last; current++)
     {
           hold = list[current];
           for (walker = current - 1; walker >= 0 &&
               hold < list[walker]; walker--)
           {
                 list [walker + 1] = list[walker];
           }  //for walker//
           list[walker + 1] = hold;
           printf("\n\n");
           printf("\t In %d  round ", current);
           for (i = 0;i < last; i++)       printf(" %d  ", list[i]);
           printf("\n\n");
     }  //for current//
     //------------ สิ้นสุดการเรียงลำดับแบบแทรก Sorted -------------------
     printf("\n\n");
     printf("Sorted Number is ");
     for (i=0; i < last; i++)       printf("%d ", list[i]);
     getch();
     return(0);
}  //--------End Program ---------//
```

7.1.1 ตัวอย่างการทำงานของการเรียงลำดับแบบแทรก

ตัวอย่าง 7-2 การเรียงลำดับตัวเลข 5 ตัว คือ 1 3 2 5 4 โดยใช้การเรียงลำดับแบบแทรก

1 : นำเข้าสมาชิกตัวแรก "1" เข้าสู่ผลลัพธ์

1 3 : นำเข้าสมาชิก "3" ไปเข้าเรียงลำดับและเปรียบเทียบ (ลำดับถูกต้องไม่ต้องทำการแทรก)

1 3 2 : นำเข้าสมาชิก "2" ไปเข้าเรียงลำดับ

1 2 3 : ทำการเปรียบเทียบและแทรกในตำแหน่งที่ถูกต้อง

1 2 3 5 : นำเข้าสมาชิก "5" ไปเข้าเรียงลำดับและเปรียบเทียบ (ลำดับถูกต้องไม่ต้องทำการแทรก)

1 2 3 5 4 : นำเข้าสมาชิก "4" ไปเข้าเรียงลำดับ

1 2 3 4 5 : ทำการเปรียบเทียบและแทรกในตำแหน่งที่ถูกต้อง

ตัวอย่าง 7-3 ขั้นตอนการเรียงลำดับแบบแทรกจากน้อยไปมาก โดยกำหนดแถวลำดับของข้อมูลนำเข้าเริ่มต้นประกอบด้วยตัวอักขระคือ "NEXAMPLE"

0	*N*	*E*	*X*	*A*	*M*	*P*	*L*	*E*	ข้อมูลเริ่มต้นในแถวลำดับ
1	N								ข้อมูลนำเข้าตัวแรกที่เข้าสู่ชุดข้อมูลผลลัพธ์
2	N	***E***							เพิ่มสมาชิก "E" ในการเปรียบเทียบ
3	***E***	N	X						ทำการแทรก "E" ในตำแหน่งที่เหมาะสมและเพิ่มสมาชิก "X"
4	E	N	X	***A***					เพิ่มสมาชิก "A" ในการเปรียบเทียบ
5	***A***	E	N	X	***M***				ทำการแทรก "A" เข้าไปในตำแหน่งที่เหมาะสมและเพิ่มสมาชิก "M"
6	A	E	***M***	N	X	***P***			ทำการแทรก "M" เข้าไปในตำแหน่งที่เหมาะสมและเพิ่มสมาชิก "P"
7	A	E	M	N	***P***	X	***L***		ทำการแทรก "P" เข้าไปในตำแหน่งที่เหมาะสมและเพิ่มสมาชิก "L"
8	A	E	***L***	M	N	P	X	***E***	ทำการแทรก "L" เข้าไปในตำแหน่งที่เหมาะสมและเพิ่มสมาชิก "E"
9	A	E	***E***	L	M	N	P	X	ทำการแทรก "E" เข้าไปในตำแหน่งที่เหมาะสม
10	**A**	**E**	**E**	**L**	**M**	**N**	**P**	**X**	ผลลัพธ์การจัดเรียงลำดับแบบแทรก

7.2 การเรียงลำดับแบบเลือก (Selection Sort)

การเรียงลำดับแบบเลือก (Selection Sort) เป็นวิธีการหรือกระบวนการคิดในการเรียงลำดับที่จะทำการเลือกข้อมูลที่ถูกต้องจากชุดข้อมูลนำเข้าตามข้อกำหนดมาจัดเก็บในตำแหน่งที่ข้อมูลนั้นควรจะอยู่ แล้ววนซ้ำในการค้นหาข้อมูลและนำมาจัดเก็บจนกระทั่งข้อมูลถูกจัดเรียงจนครบ การเลือกข้อมูลแต่ละรอบจะทำโดยสมมุติข้อมูลตัวแรกในชุดข้อมูลที่เหลืออยู่เป็นสมาชิกที่ถูกเลือก แล้วทำการเปรียบเทียบกับข้อมูลที่เหลืออยู่ตามข้อกำหนด หากข้อมูลตัวใดที่เปรียบเทียบแล้วตรงกับข้อกำหนดมากกว่า ก็ทำการแทนที่สมาชิกที่ถูกเลือกด้วยข้อมูลดังกล่าว แล้วเปรียบเทียบต่อไปจนหมดชุดข้อมูลที่มี จะได้สมาชิกที่ถูกเลือกในรอบนั้นเพื่อทำการจัดเก็บ เช่น การเรียงลำดับจากน้อยไปมาก ในรอบแรกจะทำการ

ค้นหาข้อมูลตัวที่มีค่าน้อยที่สุดโดยการเปรียบเทียบข้อมูลทั้งหมดที่มีมาจัดเก็บไว้ที่ตำแหน่งที่ 1 ในรอบที่ 2 นำข้อมูลตัวที่มีค่าน้อยรองลงมาซึ่งก็คือน้อยที่สุดในชุดข้อมูลที่เหลืออยู่ตามข้อกำหนดจะถูกเลือกจากชุดข้อมูลไปเก็บไว้ที่ตำแหน่งที่ 2 วนซ้ำทำเช่นนี้ไปในแต่ละรอบจนกระทั่งครบ ในที่สุดจะได้ชุดข้อมูลที่เรียงลำดับจากน้อยไปมากตามต้องการ

ตัวอย่าง 7-4 โปรแกรมการเรียงลำดับแบบเลือก ที่เขียนด้วยภาษาซี

```
void SelectionSort(int Array[], const int Size);
void PrintArray(int Array[], const int Size);
int main(void)
{
     int i, j, smallest, temp;
     const int Size =5;
     int Array[5];

//----- ทำการสุ่มตัวเลขลงในแถวลำดับเพื่อเป็นข้อมูลเริ่มต้นในการเรียงลำดับ ----------//
     clrscr();
     printf("\n Enter 5 data items :\n");
     for (i = 0; i < Size; i++)
          scanf("%d", &Array[i]);

// ทำการพิมพ์ค่าเริ่มต้นในแถวลำดับ //
     clrscr();
     printf( "The Array with random order:\n\n");
     PrintArray(Array, Size);
     printf("\nPress any key…");
     getch();

// ทำการเรียงข้อมูลโดยวิธีการเรียงลำดับแบบเลือก //
     for (i = 0; i < Size; i++)
     {
          smallest = i;
          for (j = i; j < Size; j++ )
          {
               if (Array[smallest] > Array[j] )
               {
                    smallest = j;
               }
          }
```

```
            temp = Array[i];
            Array[i] = Array[smallest];
            Array[smallest] = temp;
            PrintArray(Array, Size);
        }
    // สิ้นสุดการเรียงลำดับ
    // พิมพ์ข้อมูลที่เรียงลำดับเสร็จเรียบร้อยแล้ว //
    //   clrscr();
        printf( "\n\nThe Array after Selection Sort:\n\n");
        PrintArray(Array, Size);
        printf( "\nPress any key to quit…" );
        getch();
        return (0);
}

// เป็นโปรแกรมย่อยที่ทำหน้าที่พิมพ์ข้อมูลในแถวลำดับระหว่างอยู่ในการเรียงลำดับ //
void PrintArray(int Array[], const int Size)
{
        int i;
        printf("\n\n");
        for ( i = 0; i < Size; i++ )
            printf("%i ", Array[i]);
        printf("\n\n");
}
```

7.2.1 ตัวอย่างการทำงานของการเรียงลำดับแบบเลือก

ตัวอย่าง 7-5 เป็นการเรียงลำดับตัวเลข 5 ตัว คือ 1 3 2 5 4 โดยใช้การเรียงลำดับแบบเลือก

1 3 2 5 4 : ข้อมูลเริ่มต้น

1 3 2 5 4 : เลือกข้อมูลที่มีค่าน้อยที่สุดไปใส่ไว้ตำแหน่งแรก

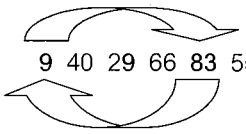
1 **2** 3 5 4 : เลือกข้อมูลที่มีค่าน้อยที่สุดโดยไม่นับรวมตำแหน่งที่ได้ทำการจัดเรียงไปแล้วไปใส่ไว้ตำแหน่งสอง

1 2 **3** 5 4 : เลือกข้อมูลที่มีค่าน้อยที่สุดโดยไม่นับรวมตำแหน่งที่ได้ทำการจัดเรียงไปแล้วไปใส่ไว้ตำแหน่งสาม

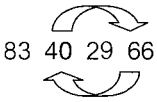
1 2 3 **4** 5 : เลือกข้อมูลที่มีค่าน้อยที่สุดโดยไม่นับรวมตำแหน่งที่ได้ทำการจัดเรียงไปแล้วไปใส่ไว้ตำแหน่งสี่

1 2 3 4 **5** : ผลลัพธ์การทำงาน

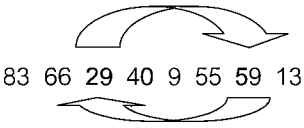
ตัวอย่าง 7-6 ขั้นตอนการเรียงลำดับจากมากไปน้อย โดยกำหนดข้อมูลเริ่มต้นคือ 9 40 29 66 83 55 59 13 45



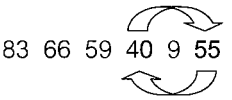
9 40 29 66 **83** 55 59 13 45 ค้นหาจนพบข้อมูลที่มีค่ามากที่สุดแล้วสลับที่กับข้อมูลที่ตำแหน่งแรก



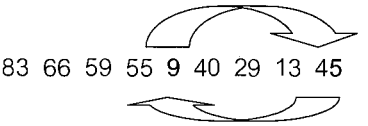
83 40 29 66 9 55 59 13 45 หาตัวที่มากที่สุดรองลงมาแล้วสลับที่กับข้อมูลตำแหน่งที่ 2



83 66 **29** 40 9 55 **59** 13 45



83 66 59 **40** 9 **55** 29 13 45



83 66 59 55 **9** 40 29 13 **45**

83 66 59 55 45 40 29 13 9

7.3 การเรียงลำดับแบบฟอง (Bubble Sort)

การเรียงลำดับแบบฟองเป็นวิธีการหรือกระบวนการคิดในการเรียงลำดับ ที่มีเปรียบเทียบข้อมูลทีละคู่ที่อยู่ในตำแหน่งติดกัน แล้วทำการสลับตำแหน่งระหว่างข้อมูลดังกล่าวในกรณีที่ตำแหน่งของข้อมูลไม่เป็นตามข้อกำหนด จากนั้นทำการเปลี่ยนคู่ใหม่โดยคงข้อมูลข้างหนึ่งของข้อมูลคู่เดิมที่อยู่ติดกับข้อมูลใหม่เพื่อใช้เปรียบเทียบกับข้อมูลตัวใหม่ที่ยังไม่เคยถูกเปรียบเทียบ วนซ้ำการเปรียบเทียบจนข้อมูลทุกตัวถูกจับคู่ครบ ถือเป็นครบหนึ่งรอบ ทำการวนซ้ำจนกระทั่งไม่มีการสลับตำแหน่งข้อมูลระหว่างคู่ (ฟอง) ภายในรอบ แสดงว่าข้อมูลถูกจัดเรียงตามข้อกำหนด

ขั้นตอนการทำงานของการเรียงลำดับแบบฟอง ในกรณีที่ข้อกำหนดเป็นการเรียงจากน้อยไปมาก คือ

1. เปรียบเทียบข้อมูลที่ติดกัน ถ้าตัวแรกมากกว่าตัวที่สอง ให้ทำการสลับตำแหน่งกัน
2. เปลี่ยนคู่ข้อมูล โดยคงข้อมูลข้างที่มากกว่าจากคู่ (ฟอง) ที่ผ่านมาเพื่อเปรียบเทียบกับข้อมูลใหม่ที่ติดกันไปทีละคู่ จนครบทุกข้อมูล ถือเป็นครบหนึ่งรอบ
3. วนซ้ำข้อหนึ่งและสองเพื่อเปรียบเทียบทุกข้อมูลทีละคู่จนกระทั่งไม่มีการสลับตำแหน่งเกิดขึ้นในรอบ

ตัวอย่าง 7-7 โปรแกรมการเรียงลำดับแบบฟองที่เขียนด้วยภาษาซี

```
#include <stdio.h>
#include <conio.h>
#include <string.h>

main()
{
     int ch, a[10], n, j, i;
     int t, ex, k;
     clrscr();
//------------ รับข้อมูลจากคีย์บอร์ดเพื่อนำเข้าแถวลำดับ a ------------------------
     printf("\nEnter the size of the array==>");
     scanf("%d", &n);
     printf("\nEnter the numbers to be sorted\n");
     for (i = 0; i < n; i++)
          scanf("%d", &a[i]);
     clrscr();
//-------- พิมพ์ข้อมูลที่ต้องการเรียงลำดับทั้งหมดในแถวลำดับ-----------------------
     printf("\n the number to be sorted :\n\n");
     for (i = 0; i < n; i++)
          printf("%d " ,a[i]);
     printf("\n\n\n");
//---------- เริ่มต้นการเรียงลำดับแบบฟอง ----------------------------------
     for (i = 0; i < n-1; i++)
     {
          ex = 0;
          for (j = 0; j < n-i-1; j++)
               if (a[j] > a[j+1])
               { //-----เป็นการสลับค่าในตำแหน่งของ j และ j+1 ในแถวลำดับ a ------
                    t = a[j];
                    a[j] = a[j+1];
                    a[j+1] = t;
// -------- พิมพ์ข้อมูลของแถวลำดับ a เมื่อเสร็จการเรียงลำดับในแต่ละรอบ -------------
                    for (k = 0; k < n; k++)        printf(" %d " ,a[k]);
                    printf("\n");
                    ++ex;
               }
```

```
            printf("\nIn %d pass, number of exchanges made is
                   %d\n\n\n\n", i, ex);
        }
    //-------- สิ้นสุดการเรียงลำดับแบบฟอง --------

    //--------พิมพ์ข้อมูลที่เรียงลำดับเสร็จแล้วในแถวลำดับ a --------
        printf("\nThe numbers in sorted order is \n\n");
        for (i = 0; i < n; i++)
            printf("%d ", a[i]);
        printf("\n\n");
        getch();
        return(0);
    }
```

7.3.1 ตัวอย่างการเรียงลำดับแบบฟอง

ตัวอย่าง 7-8 เป็นการเรียงลำดับตัวเลข 5 ตัว คือ 1 3 2 5 4 โดยใช้การเรียงลำดับแบบฟอง

1 3	: เปรียบเทียบสมาชิก 2 ตัวแรก
1 3 2	: นำสมาชิกตัวถัดไปเข้าเรียงลำดับ
1 2 3	: ทำการสลับที่
1 2 3 5	: นำสมาชิกตัวถัดไปเข้าเรียงลำดับ
1 2 3 5 4	: นำสมาชิกตัวถัดไปเข้าเรียงลำดับ
1 2 3 4 5	: ทำการสลับที่
1 2 3 4 5	: ผลการเรียงลำดับ

ตัวอย่าง 7-9 ขั้นตอนการเรียงลำดับแบบฟอง โดยกำหนดข้อมูลเริ่มต้น คือ 9 40 29 66 83 55 59 13 45 เพื่อเรียงลำดับจากมากไปน้อย

9 40 29 66 83 55 59 13 45 : เปรียบเทียบข้อมูลคู่แรก

40 9 29 66 83 55 59 13 45 : ทำการสลับที่ข้อมูลคู่แรก และเปรียบเทียบคู่ถัดไปโดยคงข้อมูลไว้หนึ่งข้าง

40 29 9 66 83 55 59 13 45 : สลับที่ข้อมูลและทำการเปรียบเทียบข้อมูลทีละคู่กับข้อมูลถัดไป

...

40 29 66 83 55 59 13 45 9 : สลับค่าจนครบทุกค่าในแถวลำดับรอบที่ 1
40 66 83 55 59 29 45 13 9 : สลับค่าจนครบทุกค่าในแถวลำดับรอบที่ 2
66 83 55 59 40 45 29 13 9 : สลับค่าจนครบทุกค่าในแถวลำดับรอบที่ 3
83 66 59 55 45 40 29 13 9 : ผลลัพธ์ของการเรียงลำดับเมื่อไม่มีการสลับตำแหน่งเกิดขึ้นต่อไป

7.4 การเรียงลำดับแบบผสาน (Merge Sort)

การเรียงลำดับแบบผสานเป็นวิธีการหรือกระบวนการคิดในการเรียงลำดับที่อาศัยหลักการแบ่งแยกและพิชิต (Divide and Conquer) โดยการแบ่งข้อมูลออกเป็นส่วนย่อย (Divide) และทำการเรียงลำดับข้อมูลในส่วนย่อยแต่ละส่วน แล้วจึงจากนำข้อมูลย่อยแต่ละส่วนที่ได้รับการจัดเรียงเรียบร้อยแล้ว มาทำการผสานเพื่อจัดเรียงข้อมูลทั้งหมดอีกครั้งหนึ่ง (Conquer)

ขั้นตอนการทำงานของการเรียงลำดับแบบผสาน คือ

1. แบ่งแยกข้อมูลที่ยังไม่ได้เรียงลำดับออกเป็นสองส่วนเท่าๆ กัน แล้ววนซ้ำการแบ่งแยกจนมีข้อมูลเพียงหนึ่งข้อมูลในส่วนย่อยที่สุดของการแบ่งแยก
2. เรียงลำดับข้อมูลในแต่ละส่วนย่อย ผสานแต่ละส่วนย่อยที่เรียงลำดับแล้วเพื่อทำการเรียงลำดับเข้าด้วยกัน และทำการวนซ้ำการเรียงลำดับส่วนย่อย ผสานส่วนย่อยทุกขั้นตามลำดับการแบ่งแยก

ตัวอย่าง 7-10 โปรแกรมการเรียงลำดับแบบผสาน

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

#define NUM_ITEMS 5

void mergeSort(int numbers[], int temp[], int array_size);
void m_sort(int numbers[], int temp[], int left, int right);
void merge(int numbers[], int temp[], int left, int mid, int right);

int numbers[NUM_ITEMS];
int temp[NUM_ITEMS];

int main()
{
    int i;
    clrscr();
```

```
// fill array with random integers
     for (i = 0; i < NUM_ITEMS; i++)
     {
          numbers[i] = rand();
          printf(" %d ", numbers[i]);
          mergeSort(numbers, temp, NUM_ITEMS);
          printf("Done with sort.\n");

          for (i = 0; i < NUM_ITEMS; i++)
              printf("%i\n", numbers[i]);
          getch();
          return(0);
     }

// เริ่มทำการเรียงลำดับ โดยการเรียกโพรซีเยอร์ m_sort
void mergeSort(int numbers[], int temp[], int array_size)
{
     m_sort(numbers, temp, 0, array_size - 1);
}

// ทำการแบ่งข้อมูลออกเป็นส่วนๆ
void m_sort(int numbers[], int temp[], int left, int right)
{
     int mid;
     if (right > left)
     {
          mid = (right + left) / 2;
          m_sort(numbers, temp, left, mid);
          m_sort(numbers, temp, mid+1, right);
          merge(numbers, temp, left, mid+1, right);
     }
}

//ทำการเรียงข้อมูลในแต่ละส่วนย่อยนั้น
void merge(int numbers[], int temp[], int left, int mid, int right)
{
     int i, left_end, num_elements, tmp_pos;
     left_end = mid - 1;
     tmp_pos = left;
```

```
    num_elements = right - left + 1;
    while ((left <= left_end) && (mid <= right))
    {
        if (numbers[left] <= numbers[mid])
        {
            temp[tmp_pos] = numbers[left];
            tmp_pos = tmp_pos + 1;
            left = left + 1;
        }
        else
        {
            temp[tmp_pos] = numbers[mid];
            tmp_pos = tmp_pos + 1;
            mid = mid + 1;
        }
    }
    while (left <= left_end)
    {
        temp[tmp_pos] = numbers[left];
        left = left + 1;
        tmp_pos = tmp_pos + 1;
    }
    while (mid <= right)
    {
        temp[tmp_pos] = numbers[mid];
        mid = mid + 1;
        tmp_pos = tmp_pos + 1;
    }
    for (i=0; i <= num_elements; i++)
    {
        numbers[right] = temp[right];
        right = right - 1;
    }
}
```

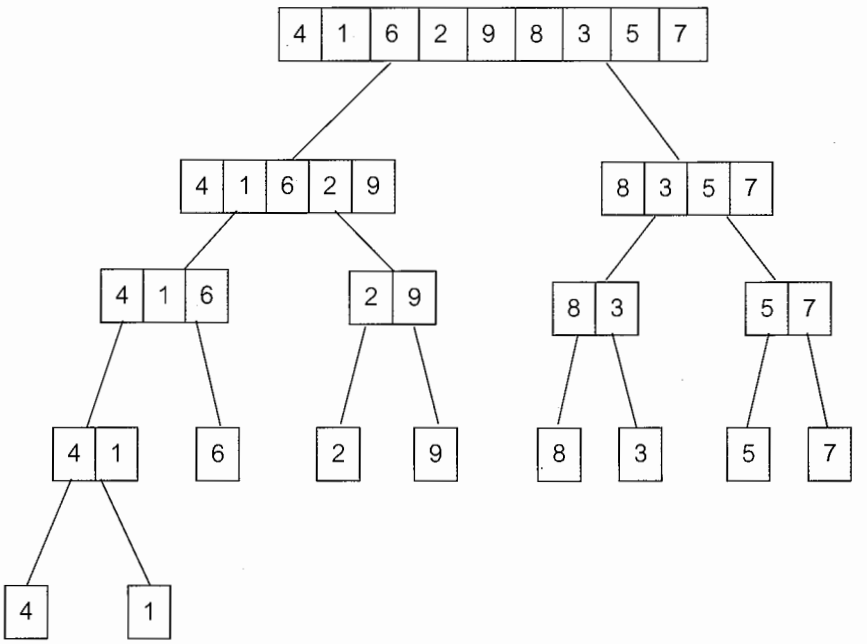
7.4.1 ตัวอย่างการทำงานของการเรียงลำดับแบบผสาน

ตัวอย่าง 7-10 การเรียงลำดับตัวเลข 5 ตัว คือ 1 3 2 5 4 โดยใช้การเรียงลำดับแบบผสาน

1 3 | 2 5 4 : แบ่งข้อมูลออกเป็น 2 ส่วน
1 3 | 2 | 5 4 : แบ่งข้อมูลทางฝั่งที่มีจำนวนข้อมูล
1 3 | 2 | 4 5 : ทำการสลับที่ข้อมูล ของแต่ละส่วนย่อย
1 3 | 2 4 5 : นำสมาชิกในแต่ละส่วนย่อยมาเรียงลำดับ
1 2 3 | 4 5 : นำสมาชิกในแต่ละส่วนย่อยมาเรียงลำดับ
1 2 3 4 | 5 : นำสมาชิกในแต่ละส่วนย่อยมาเรียงลำดับ
1 2 3 4 5 : ผลการเรียงลำดับ

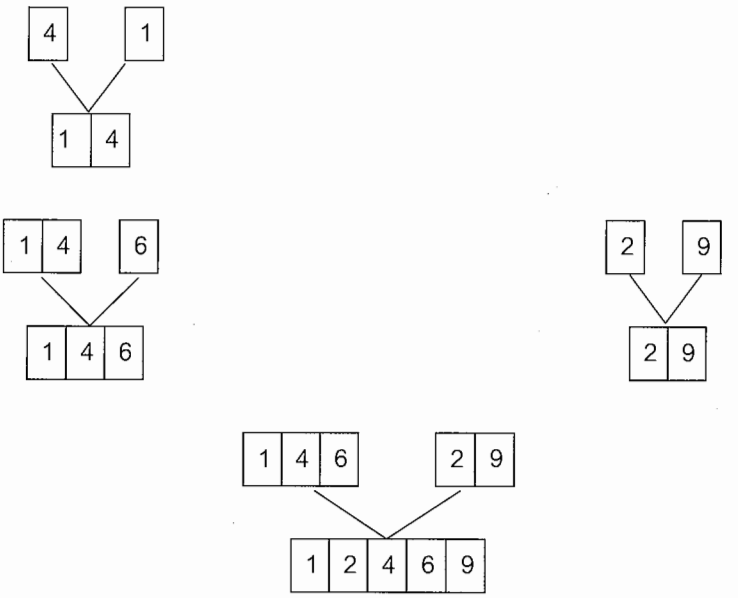
ตัวอย่าง 7-11 ขั้นตอนการเรียงลำดับจาก น้อยไปมาก โดยกำหนดข้อมูลเริ่มต้นคือ 4 1 6 2 9 8 3 5 7

1) การแบ่งแถวลำดับออกเป็นส่วนย่อย

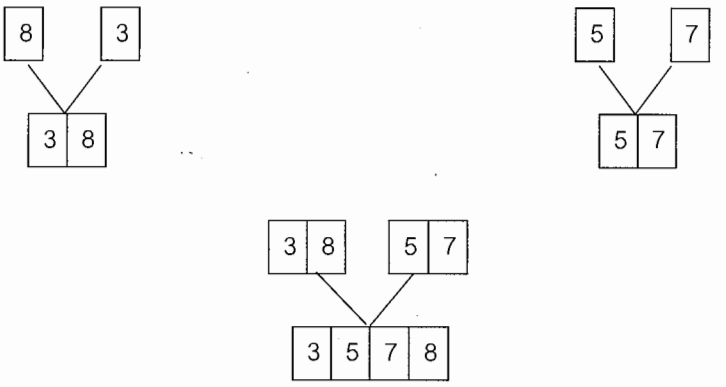


2) การรวมและการเรียงลำดับ

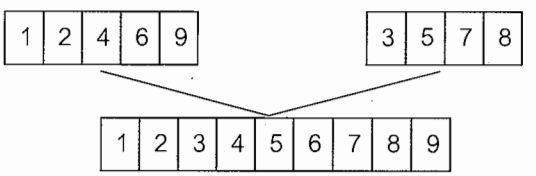
ฝั่งซ้าย



ฝั่งขวา



รวมฝั่งซ้ายและฝั่งขวา



แบบฝึกหัดบทที่ 7

1. จงเขียนโปรแกรมเรียงลำดับข้อมูลแบบแทรก โดยใช้ข้อมูลที่กำหนดให้

   3 5 8 1 6 7 1

2. จงเขียนโปรแกรมเรียงลำดับข้อมูลแบบเลือกโดยใช้ข้อมูลต่อไปนี้

   12 25 63 12 35 33 54

3. จงเขียนโปรแกรมเปรียบเทียบการทำงานของการเรียงลำดับแบบผสานกับการเรียงลำดับแบบฟอง โดยใช้ข้อมูลที่กำหนดให้ดังนี้

   56 87 45 23 65 84 16 51 23 54

4. จากข้อมูลในข้อที่ 3 ถ้าเป็นการทำงานแบบเลือกจะทำการเรียงลำดับได้ช้าหรือเร็วกว่าการเรียงลำดับแบบแทรก

5. จงอธิบายการทำงานของการเรียงลำดับแบบฟองพร้อมทั้งหาข้อดีของการเรียงลำดับแบบฟอง

บทที่ 8
การค้นหา

ในการค้นคืนข้อมูลของระบบสารสนเทศนั้น การค้นหา (searching) เป็นการดำเนินงานเพื่อหาตำแหน่งของหน่วยความจำที่เก็บข้อมูลที่ต้องการ หรือทำการแจ้งเมื่อไม่สามารถหาข้อมูลได้พบ การค้นหาจะประสพความสำเร็จหรือไม่ขึ้นกับว่าหาข้อมูลพบ ดังนั้นการค้นหาคือ กระบวนการของการหาข้อมูลในหน่วยความจำซึ่งตรงกับมูลค่าเป้าหมายที่กำหนดให้ ด้วยเหตุนี้การเลือกอัลกอริทึมในการค้นหาขึ้นกับโครงสร้างข้อมูลซึ่งใช้เก็บข้อมูลในหน่วยความจำ

อัลกอริทึมให้ค่าของทั้งระเบียน หรือให้ค่าพอยน์เตอร์ที่ชี้ไปยังระเบียน กรณีที่หาไม่พบ ก็จะให้ผลลัพธ์เป็น n หรือนัล พอยน์เตอร์ (null pointer) การค้นหาที่ประสพผลสำเร็จ เรียกว่า รีทรีพวัล (retrieval)

ในการค้นหาข้อมูล คีย์ของระเบียนเป็นสิ่งบอกความแตกต่างของระเบียน

คีย์ภายใน (Internal key) เป็นคีย์ที่อยู่ภายในระเบียนที่ตำแหน่งซึ่งกำหนดไว้จากจุดเริ่มต้นของระเบียน

คีย์ภายนอก (External key) เป็นคีย์ที่เก็บไว้ในตารางของคีย์ซึ่งรวมถึงพอยน์เตอร์ซึ่งชี้ไปยังระเบียน

ไพรมารี่ คีย์ (Primary key) เป็นคีย์ซึ่งมีเอกลักษณ์ (unique) ซึ่งแต่ละระเบียนจะมีคีย์ไม่ซ้ำกัน

ดรรชนีของแถวลำดับเป็นคีย์ภายนอกที่มีลักษณะเฉพาะ (unique external key) สำหรับเซคกันดารี่ คีย์ (Secondary key) เป็นคีย์ของระเบียนที่ไม่เป็นเอกลักษณ์ ได้แก่ 2 ระเบียนมีคีย์ตัวเดียวกัน เช่น คีย์ชื่อจังหวัด ในระเบียนของที่อยู่

อัลกอริทึมการค้นหาโดยทั่วไปมี 2 ประเภท

1. การค้นหาแบบเรียงลำดับ (Sequential Search)
2. การค้นหาแบบทวิภาค (Binary Search)

8.1 การค้นหาแบบเรียงลำดับ

หลักการทำ Sequential Search

1. ใช้กับข้อมูลที่เก็บไว้ในแถวลำดับหรือรายการโยง
2. กรณีที่ใช้แถวลำดับ k ซึ่งมีคีย์จำนวน n
3. ถ้าไม่พบระเบียนที่ต้องการค้นหาจะสามารถเพิ่มระเบียนใหม่ได้

เมื่อค้นพบข้อมูลได้ค่า integer i (เมื่อ k(i) == key) และถ้าไม่พบให้ค่าเป็นศูนย์

อัลกอริทึมของการค้นหา มีดังนี้

```
found = false;
i = 1;
while (i < n && not found)
   if (key == k[i])
      {
         search = i;
         found = true
       }
      else
        i = i + 1;
if (not found)
   search = 0
```

และเมื่อต้องการเพิ่มข้อมูล ลงไปในแถวลำดับ k สามารถทำโดย

```
if (not found)
{
      n++;                    /* เพิ่มขนาดของตาราง */
      k[n] = key;             /* ใส่คีย์ลงในแถวลำดับ k */
      search = n;
}
```

8.2 การค้นหาแบบทวิภาค

เป็นวิธีที่ดีที่สุดในการค้นหาตารางที่จัดแบบเรียงลำดับ โดยทั่วไปแล้วจะมีการเปรียบเทียบ อาร์กิวเมนท์ (argument) กับ key ของสมาชิกตัวกลางในตาราง ถ้าคีย์เท่ากับสมาชิกนั้น การค้นหาจบลง แต่ถ้าไม่เท่ากันก็มีการค้นหาที่ส่วนบนหรือส่วนล่างของตาราง

อัลกอริทึมของการค้นหาแบบทวิภาค

```
found = false;
low = 1;
hi = n;
while (low < hi && not found)
{
     mid = (low + hi) / 2;
     if (key == k[mid])
           found = true
     else if (key < k[mid])
           hi = mid - 1
         else low = mid++
}     /* จบลูป while */
```

```
if (found)
    search = mid
else search = 0
```

การเปรียบเทียบของการค้นหาแบบทวิภาคลดจำนวนสมาชิกที่จะค้นหาลงครึ่งหนึ่งในแต่ละรอบ ดังนั้นจำนวนคีย์ที่มากที่สุดในการเปรียบเทียบประมาณ $\log_2$ (n) คีย์ เมื่อแถวลำดับมีสมาชิกจำนวน n ตัว

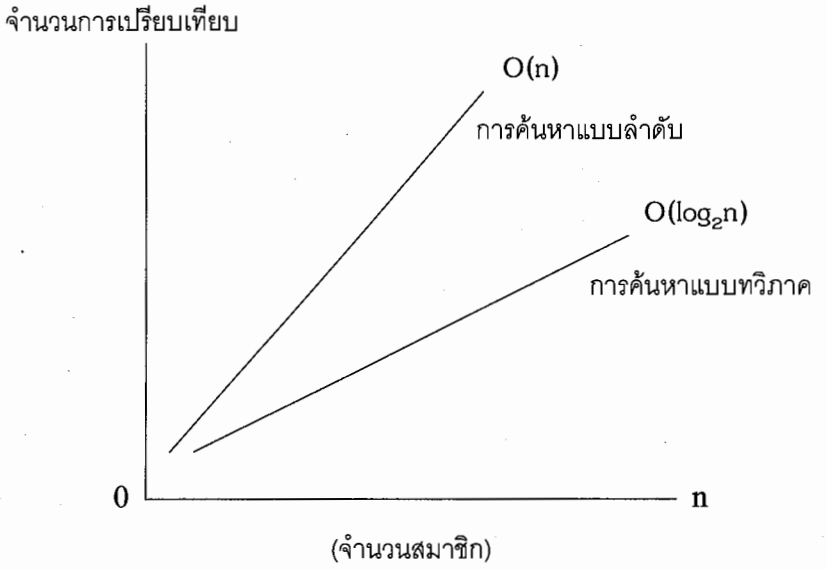
การค้นหาแบบทวิภาค ใช้กับตารางที่เก็บไว้ในแถวลำดับ โดยที่แถวลำดับนั้นได้รับการสลับลำดับและดรรชนีแถวลำดับต้องเป็น integer นอกจากนี้การค้นหาแบบนี้ใช้ได้ดีกับกรณีที่มีการลบและการเพิ่มข้อมูลในแถวลำดับจำนวนไม่มาก เพราะต้องมีการเรียงลำดับข้อมูลทุกตัวหลังจากการปรับปรุงในแถวลำดับ

เวลาที่ใช้เปรียบเทียบในการค้นหาแบบเรียงลำดับในรายการโยง หรือ แถวลำดับ 1 มิติ คือ O(n) เมื่อ n คือจำนวนสมาชิกทั้งหมด ในกรณีที่เลวที่สุดซึ่งมีการค้นหาจนถึงระเบียนสุดท้ายในรายการ หรือหาระเบียนไม่พบจะเกิดการเปรียบเทียบ n ครั้ง โดยกรณีเฉลี่ยจะเกิดการเปรียบเทียบ n/2 ครั้ง คือมีการค้นหาครึ่งหนึ่งของจำนวนรายการทั้งหมด

สำหรับการค้นหาแบบทวิภาค ในกรณีที่เลวที่สุดทำให้เกิดการเปรียบเทียบ $O(\log_2 n)$ ครั้ง ซึ่งการเปรียบเทียบชนิดนี้จะมีประสิทธิภาพดีกว่าการค้นหาแบบเรียงลำดับ ในกรณีที่รายการมีขนาดใหญ่ ตัวอย่างเช่น รายการประกอบด้วย 1000 ระเบียน ในกรณีเฉลี่ยการค้นหาแบบเรียงลำดับมีการเปรียบเทียบ 500 ครั้ง และ 1000 ครั้งในกรณีเลวที่สุดแต่สำหรับการค้นหาแบบทวิภาคกรณีเลวที่สุดใช้การเปรียบเทียบ 10 ครั้ง

กรณีของการค้นหารายการมีขนาดเล็กมาก ในการค้นหาแบบทวิภาคอาจจะให้ผลไม่แตกต่างจากแบบเรียงลำดับ เพราะการค้นหาแบบทวิภาคใช้การเปรียบเทียบน้อยกว่าแบบแรก แต่การเปรียบเทียบแต่ละครั้งมีการคำนวณใช้เวลามาก เมื่อ n มีขนาดเล็กมากค่าคงตัวของการเปรียบเทียบจำนวนครั้งอาจจะมีอิทธิพลสำคัญ ถึงแม้ว่าการเปรียบเทียบจำนวนน้อยครั้งกว่าเป็นคุณสมบัติที่ต้องการ ซึ่งส่วนนี้ใช้เวลาในการประมวลผลมาก เช่น โปรแกรมภาษาแอสเซมบลีกระทำการค้นหาแบบเรียงลำดับ โดยใช้เวลาในการเปรียบเทียบ 5 หน่วย และการค้นหาแบบทวิภาคใช้ 35 หน่วย/ครั้ง จากรายการมีสมาชิก 16 ตัว กรณีที่เลวที่สุดของการค้นหาแบบเรียงลำดับใช้เวลา 5*16 = 80 หน่วย/ครั้ง กรณีที่เลวที่สุดของการค้นหาแบบทวิภาคต้องการทำงาน $\log_2(16) = 4$ ครั้ง ดังนั้นการค้นหาใช้เวลา 4*35 = 140 หน่วย ในกรณีที่จำนวนสมาชิกของรายการมีน้อย การค้นหาแบบเรียงลำดับบางครั้งเร็วกว่าการค้นหาแบบทวิภาค

เมื่อจำนวนสมาชิกเพิ่มขึ้น ความแตกต่างระหว่างการค้นหาแบบเรียงลำดับ และแบบทวิภาคเพิ่มขึ้นอย่างรวดเร็ว เช่น กำหนดให้การค้นหาทั้ง 2 ชนิดใช้เวลาเท่ากับที่ได้กล่าวมาแล้ว เมื่อต้องการหากรณีที่เลวที่สุดของรายการประกอบด้วย 1000 ระเบียน สำหรับกรณีที่เลวที่สุดการค้นหาแบบเรียงลำดับต้องการการเปรียบเทียบ 1000 ครั้ง แต่ละครั้งใช้เวลา 5 หน่วย ดังนั้นใช้เวลาทั้งสิ้น 5000 หน่วย ในกรณีเดียวกันการค้นหาแบบทวิภาค ใช้เวลาในการเปรียบเทียบ 35 หน่วย โดยมีการเปรียบเทียบ $O(\log_2 1000)$ ซึ่งมีค่าเท่ากับ 10 ครั้ง ดังนั้นเวลาทั้งหมดที่ต้องการคือ 350 หน่วย



รูปที่ 8-1 การเปรียบเทียบการค้นหาแบบลำดับและแบบทวิภาค

กราฟในรูปที่ 8-1 แสดงความสัมพันธ์ของจำนวนสมาชิกของการค้นหาแบบเรียงลำดับและแบบทวิภาคกับเวลาที่ใช้ในการเปรียบเทียบ

อัลกอริทึมการค้นหาแบบลำดับและแบบทวิภาคจะใช้เวลาในการทำงานไม่ต่างกันมากเมื่อข้อมูลมีปริมาณน้อย แต่เมื่อข้อมูลมีขนาดใหญ่จะพบว่า การค้นหาแบบทวิภาคมีประสิทธิภาพการทำงานดีกว่าแบบลำดับ ดังนั้นการเลือกใช้อัลกอริทึมในการค้นหาข้อมูลขึ้นกับจำนวนสมาชิกที่ใช้ในการดำเนินงานดังที่ได้กล่าวมาแล้วข้างต้น

แบบฝึกหัดบทที่ 8

1. จงเขียนโปรแกรมที่เก็บข้อมูลของคณาจารย์ในแถวลำดับของระเบียน โดยแต่ละระเบียนประกอบด้วย ชื่อ สกุลของอาจารย์ ตำแหน่งวิชาการ (ศาสตราจารย์, รองศาสตราจารย์, ผู้ช่วยศาสตราจารย์, อาจารย์) และเงินเดือนเฉลี่ยของตำแหน่งที่กำหนดให้ เงินเดือนเฉลี่ยของอาจารย์ในคณะแสดงรายชื่อของคณาจารย์ในคณะ และ ชื่อของอาจารย์สำหรับตำแหน่งวิชาการที่กำหนด

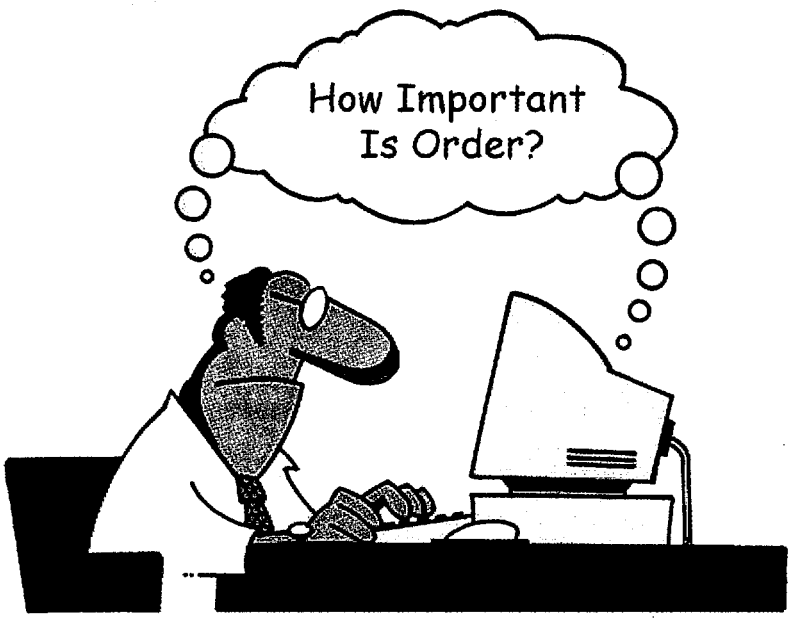
2. จงเขียนโปรแกรม เพื่อค้นหาข้อมูลใน Inventory File โดยใช้หมายเลขสต็อก ถ้าค้นหาพบจะแสดงชื่อและหมายเลขสต็อก กรณีที่หาไม่พบจะให้ข้อความว่าหาไม่พบ

3. ในทุก ๆ เดือน ศูนย์คอมพิวเตอร์จะทำรายงานสถานะภาพ การใช้เครื่องของผู้ใช้ใน UserFile จงเขียนโปรแกรมเพื่ออ่านวันที่ปัจจุบัน และผลิตรายงานในรูปแบบที่กำหนดให้

ชื่อผู้ใช้	รหัสผู้ใช้	เวลาให้ใช้	เวลาที่ใช้
มาริสา แสงทวี	1000101	750 นาที	381
ดำ แดงดี	1000102	600 นาที	599***
:	:	:	:

*** แสดงว่าผู้ใช้ได้ใช้เครื่องคอมพิวเตอร์ไปเกิน 90% ของเวลาที่กำหนด

นอกจากนี้โปรแกรมสามารถแสดงรายชื่อผู้ใช้ที่ใช้เวลาเกิน 90% ของเวลาที่กำหนด



HOW DOES THE ALGORITHM FAST ?

n	lg n	n lg n	$n^{1.25}$	n^2
1	0	0	1	1
16	4	64	32	256
256	8	2,048	1,024	65,536
4,096	12	49,152	32,768	16,777,216
65,536	16	1,048,565	1,048,476	4,294,967,296
1,048,476	20	20,969,520	33,554,432	1,099,301,922,576
16,775,616	24	402,614,784	1,073,613,825	281,421,292,179,456

Table 1-1: Growth Rates

บทที่ 9

การวิเคราะห์อัลกอริทึม

9.1 บทนำ

อัลกอริทึม (algorithm) เป็นคำที่บัญญัติขึ้นมาจากนักคณิตศาสตร์ชาวอาหรับ ชื่อ Aleu Ja'far Mohammed ibn Musu al Khowarzmi นักคณิตศาสตร์ผู้นี้แต่งหนังสือที่อธิบายกระบวนงานสำหรับการคำนวณด้วยเลขฮินดู ปัจจุบันอัลกอริทึมนั้นหมายถึง กระบวนงาน (procedure) สำหรับการแก้ปัญหาในแต่ละขั้นตอนหรือการบรรลุเป้าหมาย สำหรับด้านวิทยาการคอมพิวเตอร์อัลกอริทึม หมายถึง กระบวนงานซึ่งสามารถนำไปดำเนินงานได้โดยเครื่องคอมพิวเตอร์และมีคุณสมบัติดังต่อไปนี้

1. มีขั้นตอนการดำเนินงานที่ชัดเจนและไม่กำกวม ดังนั้นแต่ละคำสั่งนั้นมีความสมบูรณ์ในตัวเอง
2. กระบวนงานนั้นเรียบง่ายเพียงพอที่จะได้รับการดำเนินงานโดยเครื่องคอมพิวเตอร์
3. กระบวนงานมีการดำเนินที่มีจุดจบ (finiteness) โดยที่อัลกอริทึมสามารถหยุดการทำงานหลังจากการดำเนินงานไปได้ ระยะหนึ่งตามที่ต้องการและตามที่เวลากำหนด

ข้อความสั่งกำหนดค่าแบบเลขจำนวนเต็ม เช่น x := y + z เป็นตัวอย่างของคำสั่งซึ่งสามารถดำเนินงานได้ในปริมาณงานที่จำกัด ในคำสั่งของอัลกอริทึม สามารถกำหนดให้มีการวนซ้ำในเวลาขณะหนึ่ง นอกจากนี้เมื่อกำหนดข้อมูลนำเข้าที่มีมูลค่าต่างกันให้อัลกอริทึมดำเนินงาน อัลกอริทึมนั้นสามารถจบลงได้ดังนั้นโปรแกรมคืออัลกอริทึมที่ไม่มีวงอนันต์ในการดำเนินงาน เมื่อกำหนดข้อมูลนำเข้าชนิดใดๆ ให้คุณสมบัติของอัลกอริทึมที่กล่าวไปแล้ว คือ การดำเนินงานที่มีจุดจบของอัลกอริทึม

เมื่อมองถึงคุณสมบัติ 2 ประเภทของอัลกอริทึม คือ ความไม่กำกวมและความเรียบง่าย สามารถอธิบายอัลกอริทึมในรูปแบบที่คล้ายคลึงกับโปรแกรมคอมพิวเตอร์ ดังนั้นการแปลงอัลกอริทึมแต่ละขั้นตอนไปเป็นโปรแกรมคอมพิวเตอร์ทำได้ง่าย โดยทั่วไปการเขียนอัลกอริทึมมักจะเขียนในรูปแบบของคำสั่งลำลอง (pseudocode) ซึ่งเป็นการรวมภาษาธรรมชาติกับสัญลักษณ์ คำศัพท์ และ คุณลักษณะอื่น ๆ ที่ใช้ในภาษาโปรแกรมมิ่งระดับสูง อย่างไรก็ตามคำสั่งลำลองนี้ไม่มีมาตรฐานของไวยากรณ์ จึงทำให้ภาษานี้ มีความแตกต่างกันสำหรับนักเขียนโปรแกรมแต่ละคน

ในการทำงานโดยทั่วไปการรู้ว่าอัลกอริทึมหยุดทำงานเป็นข้อมูลที่ไม่เพียงพอ ตัวอย่างเช่น อัลกอริทึมในการเล่นหมากรุก ในแต่ละขั้นตอนของเกมมีการพัฒนาการเดินหมากว่า การเดินแต่ละครั้งจะเป็นการเดินที่ชนะและจบการทำงาน แต่การพิจารณาแบบนี้จะใช้เวลามาก อัลกอริทึมที่มีประโยชน์ต้องจบการทำงานในเวลาที่มีเหตุผล ซึ่งเทคนิคในการพิจารณาเวลาในการดำเนินงานของอัลกอริทึมจะนำเสนอต่อไป

9.2 การวิเคราะห์อัลกอริทึม

เนื่องจากปัญหาที่กำหนดให้เพียงหนึ่งปัญหาจะสามารถแก้ปัญหาด้วยอัลกอริทึมหลายแบบในสภาพเงื่อนไขเดียวกัน ดังนั้นการเปรียบเทียบประสิทธิภาพการทำงานของอัลกอริทึมต่าง ๆ จึงจำเป็น นอกจากนี้การวิเคราะห์อัลกอริทึมและเทคนิคที่ใช้ในการวัดประสิทธิภาพของอัลกอริทึม นำมาใช้ในการพิจารณาอัลกอริทึมในบทปัจจุบัน

ประสิทธิภาพของอัลกอริทึม

การวัดประสิทธิภาพของอัลกอริทึมทำได้ 2 วิธี คือ การใช้เนื้อที่ในหน่วยความจำ หมายถึง ความต้องการในการใช้เนื้อที่ของความจำในการเก็บข้อมูล กรณีที่ 2 คือ ประสิทธิภาพของเวลาในการทำงาน หมายถึง ระยะเวลาที่ใช้ในการประมวลผลข้อมูล การใช้เนื้อที่และเวลาในการดำเนินงานให้น้อยที่สุดทั้ง 2 กรณีพร้อมกันทำได้ยาก โดยทั่วไปคุณสมบัติสำคัญของอัลกอริทึมที่มีประสิทธิภาพ คือ การใช้เวลาอย่างมีประสิทธิภาพ ในส่วนต่อไปนี้จะมีการศึกษาวิธีการพิจารณาประสิทธิภาพของอัลกอริทึม

การพิจารณาว่าอัลกอริทึมที่เขียนขึ้นเป็นอัลกอริทึมที่ดีหรือไม่ สามารถกระทำได้โดยเปรียบเทียบอัลกอริทึมที่ทำงานประเภทเดียวกัน วิธีนี้ดำเนินโดยกำหนดวัตถุประสงค์ในการนำมาใช้กับอัลกอริทึมแต่ละชนิด การวิเคราะห์อัลกอริทึมเป็นสาขาที่สำคัญของทฤษฎีทางวิทยาการคอมพิวเตอร์ ในบทนี้กล่าวถึงการพิจารณาอัลกอริทึมในการเรียงลำดับ

วิธีการวัดการทำงานของอัลกอริทึม 2 อัลกอริทึมมีขั้นตอนดังนี้คือ ขั้นตอนที่หนึ่งได้แก่ ถอดรหัสจากอัลกอริทึมแล้วหาเวลาในการปฏิบัติการ (execution time) หรือในการดำเนินงาน (run) ของโปรแกรมทั้ง 2 โปรแกรม โปรแกรมที่ใช้เวลาน้อยที่สุดมีอัลกอริทึมที่ดีกว่า แต่เวลาในการปฏิบัติการที่ได้ขึ้นกับชนิดของคอมพิวเตอร์เท่านั้น ถ้าผู้ใช้ต้องการวิธีการที่เรียบง่ายกว่านี้ โดยใช้วิธีที่สองที่จะกล่าวถึง

วิธีที่สอง คือ การนับจำนวนของคำสั่ง (instruction) หรือคำสั่งที่ได้รับการกระทำการ วิธีการนี้ขึ้นกับชนิดของภาษาชุดคำสั่ง (programming language) ที่นำมาใช้และ รูปแบบการเขียนโปรแกรมของนักเขียนแต่ละคน เพื่อที่จะให้การวัดเป็นมาตรฐาน จึงสามารถนับจำนวนการส่งผ่าน (passes) วงวิกฤต (critical loop) ในอัลกอริทึม ถ้าการวนซ้ำแต่ละครั้งเกี่ยวข้องกับปริมาณงานที่คงที่ การวัดนี้ให้ผลที่มีประสิทธิภาพ

สิ่งเหล่านี้นำไปสู่แนวคิดที่แยกการทำงานพื้นฐานของอัลกอริทึม และการนับจำนวนครั้งของการทำงานเช่น การค้นหามูลค่าหนึ่งในแถวลำดับ เริ่มจากการนับจำนวนการเปรียบเทียบ ระหว่างมูลค่าเป้าหมาย และจำนวนสมาชิกในแถวลำดับที่เก็บมูลค่าต่าง ๆ การบวกจำนวนสมาชิกของแถวลำดับได้จากการกระทำการบวก (addition operations) เช่น แถวลำดับที่มีสมาชิก n จะมีการบวก n-1 ครั้ง ดังนั้นสามารถเปรียบเทียบอัลกอริทึมสำหรับกรณีทั่วไป ซึ่งไม่ใช่กรณีพิเศษสำหรับขนาดของแถวลำดับ ถ้าต้องการเปรียบเทียบอัลกอริทึมสำหรับคูณเมทริกซ์เข้าด้วยกัน สามารถวัดได้โดยการรวมกระทำการคูณและการกระทำการบวก ซึ่งใช้ในการคูณเมทริกซ์

ตัวอย่างที่ได้กล่าวมาแล้วนำไปสู่ข้อคิดที่น่าสนใจ ในบางครั้งการกระทำการอย่าง หนึ่งจะมีอิทธิพลต่ออัลกอริทึม ซึ่งทำให้การกระทำการอื่นหมดความหมาย เช่นถ้าต้องการซื้อเครื่องคอมพิวเตอร์และแผ่นดิสก์ เมื่อเข้าไปในร้านขายอุปกรณ์คอมพิวเตอร์ ราคาของแผ่นดิสก์น้อยมากเมื่อเทียบกับราคาเครื่องคอมพิวเตอร์ เมื่อกลับมาในกรณีของการคูณ

เมทริกซ์ การปฏิบัติการคูณมีราคาแพงมากกว่าการบวกในเทอมของเวลาคอมพิวเตอร์ ดังนั้นการบวกจึงเป็นแฟกเตอร์ที่มีผลน้อยมากในอัลกอริทึมของการคูณเมทริกซ์ ฉะนั้นเราจึงนับเพียงการกระทำการคูณแต่ไม่นับการบวก ในการวิเคราะห์อัลกอริทึมพบว่าอัลกอริทึมส่วนหนึ่งที่มีอิทธิพลต่ออัลกอริทึมทั้งหมด จึงทำให้ส่วนอื่น ๆ ไม่ได้รับความสนใจ

ในการบวกสมาชิกของแถวลำดับ ไม่จำเป็นต้องนับจำนวนการกระทำการที่วิกฤตใน แถวลำดับ โดยที่จำนวนของการเปรียบเทียบเป็นฟังก์ชันของจำนวนสมาชิก (n) ในแถวลำดับ

ดังนั้นจึงสามารถกล่าวถึงจำนวนการเปรียบเทียบในเทอมของ n (เช่น n1 หรือ n2) มากกว่า เลขจำนวนเต็ม (เช่น 52)

สัญกรณ์ บิก โอ (Big O Notation)

บิกโอ หรือ อันดับขนาด (order of magnitude) เป็นฟังก์ชันที่ได้จากการประมาณค่าทางคณิตศาสตร์ ซึ่งอันดับขนาดของฟังก์ชันเหมือนกับอันดับ (ยกกำลัง) ของเทอมในฟังก์ชันซึ่งเพิ่มขึ้นอย่างเร็วสุดสัมพันธ์กับขนาดของปัญหา เช่น

$f(n) = n^4 + 100n^2 + 10n + 50$

ดังนั้น f(n) เป็นอันดับ n^4 หรือสัญกรณ์ บิก โอ โนเตชัน $O(n^4)$ ฉะนั้นสำหรับมูลค่าที่มากของ n โดย n^4 มีอิทธิพลต่อฟังก์ชัน

ในกรณีนี้ถือว่าเทอมที่มีลำดับที่น้อยกว่าถูกตัดทิ้งไปซึ่งเหมือนกับกรณีของราคาเครื่องคอมพิวเตอร์ และราคาของแผ่นดิสก์ กรณีของ n^4 มีค่ามากกว่า 50 10n หรือ $100n^2$ เมื่อ n มีค่าใหญ่มากซึ่งทำให้เทอมเหล่านี้ถูกละเลยไป สิ่งนี้ไม่ได้หมายความว่าเทอมที่ถูกกล่าวถึงไม่ใช้เวลาในการทำงานของเครื่องคอมพิวเตอร์แต่หมายถึงเทอมเหล่านี้ไม่มีนัยสำคัญในการประมาณค่า

อันดับขนาดในอัลกอริทึมไม่ได้บอกถึงเวลา หน่วยเป็นนาโนวินาทีที่คอมพิวเตอร์ใช้ในการดำเนินงาน สำหรับการจับเวลานี้ผู้ใช้ควรใช้เมทริกซ์และพิจารณาถึงปัจจัยอื่น เช่น ความชำนาญของนักเขียนโปรแกรม ตัวแปรชุดคำสั่งที่นำมาใช้ และความเร็วของโมเดลของเครื่องคอมพิวเตอร์ ดังนั้น บิก โอ ให้ข้อมูลในการเปรียบเทียบอัลกอริทึมเท่านั้น โดยปราศจากการนำแฟกเตอร์ที่กล่าวไปแล้วมาใช้อ้างอิง

อันดับขนาด (Order of Magnitude)

เวลาในการคำนวณแบบคงที่ คือ O(1) เวลาคงที่หมายถึงเวลาซึ่งไม่เปลี่ยนแปลงที่ใช้สำหรับการกระทำการที่มีเวลาในการคำนวณคงที่ ในทำนองเดียวกันการเข้าถึงสมาชิกตัวที่ i ในแถวลำดับที่มีขนาด n คือ O(1) เพราะว่าสมาชิกแต่ละตัวในแถวลำดับสามารถ เข้าถึงได้โดยตรงทางดรรชนี

อัลกอริทึม O(n) ได้รับการกล่าวถึงว่ากระทำการ โดยใช้เวลาเส้นตรง (linear time) การพิมพ์สมาชิกทั้งหมดของแถวลำดับที่มีขนาด n คือ O(n) การค้นหาสมาชิกในแถวลำดับคือ O(n) เพราะว่าอัลกอริทึมคือการค้นหาสมาชิกทุก ๆ ตัวในแถวลำดับ

อัลกอริทึม $O(\log_2 n)$ มีการทำงานมากกว่า อัลกอริทึม O(1) แต่ทำงานน้อยกว่า อัลกอริทึม O(n) การค้นหาข้อมูลแบบทวิภาคใช้เวลาในการทำงาน $O(\log_2 n)$

เวลาควอดราติก หรือ $O(n^2)$ เป็นอัลกอริทึมที่มีการทำงานมากกว่า O(n) อัลกอริทึมของการเรียงลำดับแบบธรรมดาส่วนใหญ่คือ $O(n^2)$

$O(n^3)$ เรียกว่า เวลาลูกบาศก์ ตัวอย่างของอัลกอริทึม $O(n^3)$ คือรูทีนซึ่งกำหนดค่าเริ่มต้นของ แถวลำดับ 3 มิติ nxnxn โดยให้สมาชิกทุกตัวมีค่าเป็นศูนย์

เวลาเลขชี้กำลัง หรือ $O(2^n)$ เป็นอัลกอริทึมที่สิ้นเปลืองมาก เวลาในการทำงานของคอมพิวเตอร์เพิ่มขึ้นเป็นสัดส่วนกับขนาดของ n นอกจากนี้พบว่า $n\log_2 n$ มีค่าเพิ่มขึ้นช้ากว่า n^2 (ข้อสังเกต 2^n จะมีมูลค่าเพิ่มขึ้นมากสุด)

อันดับขนาดหมายถึง ปริมาณงานที่เครื่องคอมพิวเตอร์ทำไม่ขึ้นกับขนาดของโปรแกรม หรือ จำนวนบรรทัดของโปรแกรม เช่น พิจารณาอัลกอริทึม 2 วิธี ซึ่งกำหนดค่าเริ่มต้นของสมาชิกทุกตัวในแถวลำดับขนาด n ให้เป็นศูนย์

อัลกอริทึม 1

```
list[1] = 0;
list[2] = 0;
list[3] = 0;
   ...
list[n] = 0;
```

อัลกอริทึม 2

```
for (i = 1; i <= n; i++)
   list[i] = 0;
```

ถึงแม้ว่าอัลกอริทึมที่กล่าวมามีความแตกต่างกันในแง่จำนวนของบรรทัด แต่อัลกอริทึมทั้งสองแบบมีอันดับขนาดคือ O(n)

นอกจากนี้เวลาที่อัลกอริทึมใช้ในการปฏิบัติงานนั้น ขึ้นกับตัวประกอบสำคัญ ได้แก่ ขนาดของข้อมูลนำเข้า เพราะจำนวนของชิ้นข้อมูลมีผลต่อเวลาที่ใช้ในการประมวลผล ตัวอย่างเช่นเวลาที่ใช้ในการเรียงลำดับข้อมูล ขึ้นกับปริมาณของชิ้นข้อมูลนำเข้า ดังนั้นเวลาในการดำเนินงาน T ของอัลกอริทึมนั้นแทนด้วย ฟังก์ชัน T(n) ที่มีข้อมูลนำเข้าขนาด n ตัว โดยทั่วไปเราอาจจะมองว่า T(n) เป็นจำนวนของคำสั่งที่ได้รับการดำเนินงานบนเครื่องคอมพิวเตอร์ในอุดมคติ ซึ่งคำนวณ T(n) จากเวลาในการปฏิบัติงานของคำสั่งในอัลกอริทึม

ตัวอย่าง 9-1 อัลกอริทึมของการหาค่าเฉลี่ยของจำนวนจริง n จำนวนที่เก็บไว้ในแถวลำดับ

(* รับค่า : n เป็นจำนวนเต็มที่มีค่ามากกว่าหรือเท่ากับและแถวลำดับ x[i],...,x[n] ของจำนวนจริง
หน้าที่ : หาค่าเฉลี่ยของจำนวนจริง n ตัว
ผลลัพธ์ : ค่าเฉลี่ยของ x[i],.., x[n] *)

1. กำหนดค่าเริ่มต้น sum = 0
2. กำหนดค่าเริ่มต้น ตัวดรรชนี i = 0
3. While i < n do
4. a. เพิ่มค่า i อีก 1
5. b. เพิ่ม x[i] ให้ sum
6. คำนวณ และ ส่งค่า mean = sum/n

ในอัลกอริทึมนี้คำสั่ง 1 และ 2 ทำงานคำสั่งละ 1 ครั้ง คำสั่งถัดมาคือ คำสั่ง 4 และ 5 ทำงานจำนวน n ครั้ง และคำสั่ง 3 ซึ่งควบคุมการทำงานซ้ำมีการปฏิบัติงาน n+1 ครั้ง เพราะว่ามีการตรวจสอบว่าตัวแปรควบคุม i มีค่าน้อยกว่า n หรือไม่ เมื่อ i มีค่าเท่ากับ n การวนซ้ำจบลง ต่อมาคำสั่ง 6 ทำงาน 1 ครั้ง

คำสั่ง	เวลาที่ใช้ทำงาน
1	1
2	1
3	n+1
4	n
5	n
6	1
รวม	3n+4

ดังนั้นเวลาที่ใช้ในการคำนวณของอัลกอริทึม คือ

$T(n) = 3n+4$

เมื่อข้อมูลนำเข้าจำนวน n ตัว มีขนาดเพิ่มขึ้น มูลค่าของนิพจน์ T(n) จะเพิ่มขึ้นในอัตราส่วนที่สัมพันธ์กับ n และสามารถกล่าวได้ว่า T(n) เป็นอันดับของขนาด n : ซึ่งใช้แทนด้วย บิก โอ โนเตชั่น

$T(n) = O(n)$

โดยทั่วไป การคำนวณเวลา T(n) ของอัลกอริทึมที่มีอันดับของขนาด f(n) เขียนแทนด้วย

$T(n) = O(f(n))$

กรณีที่มีค่าคงตัว C ดังนั้น

$T(n) \leq C.\ f(n)$ สำหรับ ทุกค่าของ n ที่มีขนาดใหญ่พอ นั่นคือ T(n) ขึ้นกับค่าคงตัว f(n) สำหรับทุกค่าของ n ที่เริ่มจุดบางจุด ความซับซ้อนในการคำนวณของอัลกอริทึมคือ O(f(n)) โปรแกรมที่ใช้เวลาในการดำเนินงาน O(f(n)) นั้นกล่าวได้ว่า มีอัตราการเติบโต f (n) ดังนั้น เมื่อกล่าวถึง T(n) อาจกล่าวได้ว่า f(n) เป็นขอบเขตบนของอัตราการเติบโตของ T(n)

ความซับซ้อนของอัลกอริทึม ในตัวอย่าง 9-1 กล่าวมาแล้ว คือ O(n) เพราะว่าเวลาที่ใช้ในการคำนวณ คือ

$T(n) = 3n+4$

เพราะว่า

$3n+4 \leq 3n+n$ สำหรับ $n \geq 4$

เห็นได้ว่า

$T(n) \leq 4n$ สำหรับทุกค่าของ $n \geq 4$

ดังนั้น เมื่อกำหนดให้

$f(n) = n$ และ $c=4$

สามารถเขียนได้

$T(n) = O(n)$

การเขียน $T(n) = O(52n)$ หรือ $T(n) = O(24n^2+5)$ หรือ $T(n) = O(8.164n+12.238)$ นั้นถูกต้องแต่ผู้ใช้มักเลือกฟังก์ชันที่ง่าย เช่น n, n^2 หรือ $\log_2 n$ ในการลดความซับซ้อนของอัลกอริทึม ดังนั้นการเขียน $T(n) = O(n)$ หรือ $T(n) = O(n^2)$ หรือ $T(n) = O(n^{7/2})$ หรือ $T(n) = O(2^n)$ สามารถใช้แทนฟังก์ชันของ T(n)

ในตัวอย่าง 9-1 ขณะนี้เวลาการทำงานขึ้นกับขนาดของข้อมูลนำเข้า สำหรับปัญหากรณีอื่น เวลาในการทำงานอาจจะขึ้นกับตัวประกอบอื่น เช่น การเรียงลำดับข้อมูลที่ได้รับการจัดลำดับเป็นบางส่วนจะใช้เวลาน้อยกว่าข้อมูลที่ไม่ได้จัดอันดับเลย เราอาจจะวัด T ในกรณีเลวที่สุด (worst case) หรือ กรณีดีที่สุด (best case) หรือพยายามที่จะคำนวณค่าเฉลี่ยของ T ทุกกรณี การทำงานในกรณีที่ดีที่สุดของอัลกอริทึมมักจะไม่ได้ให้ข้อมูลที่มีประโยชน์ การคำนวณในกรณีเฉลี่ยนั้นคำนวณยากกว่าการคำนวณกรณีที่เลวที่สุด

โดยทั่วไป T(n) มักจะพบบ่อยที่สุด ในกรณีของการทำงานเลวที่สุด

ตัวอย่าง 9-2 อัลกอริทึมของการค้นหาแบบลำดับ

```
/* โปรแกรมการค้นหาแบบลำดับ */
#include <stdio.h>
void main(void) {
int found;
int loc, n, item;
int a[100];

/* ข้อมูลนำเข้า   :  แถวลำดับ a ประกอบด้วยสมาชิก n ตัว และ item เป็นสมาชิกของแถวลำดับ
หน้าที่          :  ทำการค้นหาข้อมูลจากแถวลำดับของ a[1] ถึง a[n] แบบลำดับ
ข้อมูลส่งออก     :  found มีค่าเป็นจริงและ loc ได้ค่าตำแหน่งของข้อมูลที่ต้องการเมื่อการค้นหาประสบผล
                    สำเร็จ อีกกรณีหนึ่ง found มีค่าเป็นเท็จ */
{
1.    found  = 0;
2.    loc = 1;
3.    while ((loc <= n) && !found)
4.        if (item == a[loc]) then   /* พบข้อมูลที่ต้องการค้นหา*/
5.          found = 1;
6.        else                       /*ค้นหาต่อไป*/
            loc ++
}
```

กรณีเลวที่สุดนั้น item ไม่อยู่ในแถวลำดับซึ่งใช้เวลาในการค้นหาสูงสุด สามารถคำนวณเวลาในการดำเนินงาน $T_I(n)$ ได้ดังนี้

คำสั่ง	เวลาที่ดำเนินงาน
1	1
2	1
3	n+1
4	n
5	0
6	n

ดังนั้น $T_I(n) = 3n+3$ ดังนั้น

$T_I(n) = O(n)$

เพราะว่า $3n + 3 \leq 4n$ เมื่อ $n \geq 3$

ถ้าค้นหาแถวลำดับที่เรียงลำดับแล้ว ใช้การค้นหาแบบทวิภาคใช้แทนการค้นหาแบบลำดับ สำหรับการกำหนดตำแหน่งของ item ในแถวลำดับ พิจารณาสมาชิก a[loc] ที่จุดกึ่งกลางของแถวลำดับ จะมีการเปรียบเทียบ 3 กรณีดังนี้

item < a[loc] : ค้นหาครึ่งแรกของแถวลำดับ

item > a[loc] : ค้นหาครึ่งหลังของแถวลำดับ

item = a[loc] : การค้นหาบรรลุเป้าหมาย

ตัวอย่าง 9-3 อัลกอริทึมของการค้นหาแบบทวิภาคมีดังนี้

(* ข้อมูลนำเข้า : แถวลำดับ a ประกอบด้วยสมาชิกที่เรียงลำดับจำนวน n ตัว และ item คือ ข้อมูลที่ต้องการค้นหาและเป็นชนิดเดียวกับสมาชิกของ a

หน้าที่ : ดำเนินการค้นหาแบบทวิภาคของแถวลำดับ a[1] ถึง a[n]

ข้อมูลส่งออก : found มีค่าเป็นจริงและ loc คือตำแหน่งของ item ถ้าการค้นหานี้ประสพความสำเร็จหรือ found มีค่าเป็นเท็จ *)

```
/* โปรแกรมการค้นหาแบบทวิภาค */
#include <stdio.h>
void main(void)
{
     int n, item, loc, first, found, last;
     int a[100];
     {
      1. found = 0;
      2. first = 1;
      3. last  = n - 1 ;
      4.      while ((first <= last) && !found) {
      5.          loc = (first + last) / 2;
      6.          if (item < a[loc])
      7.              last = loc - 1;       /* ค้นหาครึ่งแรก */
      8.          else if (item > a[loc])
      9.              first = loc + 1;      /* ค้นหาครึ่งหลัง */
      10.         else
                      found = 1;            /* พบ item ที่ต้องการค้นหา */
            }
      }
```

ในอัลกอริทึมนี้ คำสั่งที่ 1, 2 และ 3 ทำงานเพียงครั้งเดียว สำหรับการคำนวณเวลาทำงาน กรณีที่เลวที่สุด คือ T_B (n) มีการพิจารณาเวลาในการวนซ้ำเมื่อคำสั่งที่ 4 ถึง 10 ได้รับการปฏิบัติงาน แต่ละรอบในการวนขนาดของแถวลำดับลดไปครึ่งหนึ่ง ในการวนครั้งสุดท้ายขนาดของแถวลำดับจะเป็น 1 สำหรับจำนวน วง คือ 1 บวกด้วยจำนวนของการวนรอบ k ครั้ง ดังนั้นขนาดของแถวลำดับย่อยหลังจากการวน k ครั้งคือ $n / 2^k$

$$n / 2^k < 2$$

ดังนั้น $n < 2^{k+1}$

เมื่อคำนวณโดยใช้ log จะได้

$$\log_2 n < k+1$$

จำนวนรอบของการวนซ้ำคือ จำนวนเต็มที่คำนวณได้จาก $\log_2 n$ ดังนั้น กรณีที่เลวที่สุดเกิดขึ้นเมื่อ item มีค่ามากกว่าสมาชิกในแถวลำดับช่องที่ a[1] ถึง a[n] คำสั่งที่ 4 ได้รับการดำเนินงานไม่เกิน $2 + \log_2 n$ ครั้ง สำหรับคำสั่ง 5, 6, 8 และ 9 ดำเนินงานไม่เกิน $1+\log_2 n$ ครั้ง สำหรับคำสั่งที่ 7 และ 10 ดำเนินงาน 0 ครั้ง ฉะนั้นเวลาในการคำนวณทั้งหมด คือ $9+5 \log_2 n$

ดังนั้น $T_B (n) = O (\log_2 n)$

จึงเห็นได้ว่าการค้นหาแบบทวิภาคมีประสิทธิภาพดีกว่าการค้นหาแบบลำดับ ในกรณีของข้อมูลขนาดใหญ่ เพราะความซับซ้อนของการค้นหาแบบลำดับ คือ O(n) และของการค้นหาแบบทวิภาค คือ $O(\log_2 n)$ สำหรับข้อมูลที่มีขนาดเล็กการค้นแบบลำดับอาจจะทำงานดีกว่าแบบวิภาค จากการศึกษาแบบเอ็มพิริคัล พบว่า การค้นหาแบบลำดับมีประสิทธิภาพดีกว่าแบบทวิภาค เมื่อสมาชิกในแถวลำดับมีขนาดน้อยกว่า 20 ตัว

เมื่อเปรียบเทียบการคำนวณเวลาในการทำงานของอัลกอริทึมหลายชนิด เช่น O(1) $O(\log_2 \log_2 n)$ $O(\log_2 n)$ O(n) $O(n^2)$ และ $O(n^3)$ โดยที่ O(1) แทนเวลาทำงานที่คงที่ซึ่งไม่ขึ้นกับขนาดของข้อมูลนำเข้า เมื่อเทียบกับฟังก์ชันอื่นที่เวลาการทำงานขึ้นกับปริมาณข้อมูลนำเข้า ตารางที่ 9-1 แสดงมูลค่าที่ได้จากการคำนวณของฟังก์ชันต่าง ๆ ที่ได้กล่าวมาแล้ว

$\log_2\log_2 n$	$\log_2 n$	n	$n\log_2 n$	n^2	n^3	2^n
-	0	1	0	1	1	2
0	1	2	2	4	8	4
1	2	4	8	16	64	16
1.58	3	8	24	64	512	256
2	4	16	64	256	4096	65536
2.32	5	32	160	1024	32768	4294967296
2.6	6	64	384	4096	2.6×10^5	1.85×10^{19}
3	8	256	2.05×10^3	6.55×10^4	1.68×10^7	1.16×10^{77}
3.32	10	1024	1.02×10^4	1.05×10^6	1.07×10^9	1.8×10^{308}
4.32	20	1048576	2.1×10^7	1.1×10^{12}	1.15×10^{18}	6.7×10^{315652}

ตารางที่ 9-1 การเปรียบเทียบค่าที่ได้จากการคำนวณของฟังก์ชันของอัลกอริทึมชนิดต่างๆ

ตารางที่ 9-1 แสดงว่าอัลกอริทึมที่มีความซับซ้อนได้แก่แบบเลขชี้กำลัง ซึ่งปฏิบัติงานได้เหมาะสมกับการแก้ปัญหาที่มีข้อมูลจำนวนน้อย

แบบฝึกหัดบทที่ 9

1. จงเขียนคำจำกัดความของอัลกอริทึม และอธิบายถึงคุณสมบัติที่นำมาใช้ในการพัฒนาซอฟต์แวร์

2. โปรแกรมที่กำหนดให้นี้เขียนถูกต้องตามไวยากรณ์ของภาษาซี จงพิจารณาว่าอัลกอริทึมของโปรแกรมนี้มีประสิทธิภาพดีหรือไม่ ถ้าไม่ดีให้เขียนอัลกอริทึมใหม่ที่มีประสิทธิภาพในการแก้ปัญหาดีกว่าเดิม

```
/* program search */
#include <stdio.h>

void main()
{
   int found;
   int row, col, n, item;
   int mat[100][100];

   scanf("%d", &item);
   found = 0;
   for (row = 0; row < n; row++)
        for (col = 0; col < n; col++)
             if (mat[row][col] == item)
                  found = 1;
   if (found)
        printf("item found\n");
   else
        printf("item not found\n");
}
```

3. จงยกตัวอย่างของอัลกอริทึมที่มีความซับซ้อนเท่ากับ O(1)

บรรณานุกรม

1. Aho, A. V., J. E. Hopcroft, and J. D. Ullman. 1983. Data structures and algorithms. Addison-Wesley Publishing Co. Reading, MA. 427 p.
2. Cormen, T. H., C. E. Leiserson and R. L. Rivest, 1990. Introduction to algorithms. MIT Press, Cambridge, MA. 1028 p.
3. Cormen, T. H., C. E. Leiserson and R. L. Rivest, 1996. Introduction to algorithms. MIT Press, Cambridge, MA. 997 p.
4. Dale, N., and S. C. Lilly. 1988. Pascal plus data structures, algorithms, and advanced programming. Second edition. D. C. Health and Co. Lexington, MA. 575 p.
5. DiElsi, J. 1991. Turbo Pascal 6.0. The nuts and bolts of program construction. McGraw-Hill International Inc. Singapore. 570 p.
6. Gilberg, R. F. and B. A. Forouzan. 1998. Data Structures: A Pseudocode Approach with C. PWS Publishing Company. Boston. 736 p.
7. Goodrich, M. T. and R. Tamassia. 2001. Data Structures and Algorithms in JAVA. Second Edition. Wiley. New York. 641 p.
8. Kruse, R. K., C. L. Tondo, and B. Leung. 1997. Data structures & Program design in C. Second Edition. Prentice-Hall. Inc. Singapore. 671 p.
9. Lipschultz, S. 1986. Schaum's outline series theory and problems of data structures. McGraw-Hill International Inc. Singapore. 344 p.
10. Nyhoff, L. and S. Leestma. 1992. Data structures and program design in Pascal. Second Edition. Maxwell Macmillan International Inc. Singapore. 740 p.
11. Shackelford, R. L. 1998. Introduction to computing and algorithms. Addison-Wesley Longman, Inc. Tokyo. 412 p.
12. Tenenbaum, A. M. and M. J. Augenstein. 1981. Data structures using Pascal. Prentice Hall, Inc. Englewood Cliffs, NJ. 545 p.
13. Weiss, M. A. 1992. Data Structures and Algorithm Analysis. Benjamin/Cummings Publishing Company, Inc. Redwood City, Calif. 455 p.
14. Wirth, N. 1976. Algorithms + Data structures = Programs. Prentice-Hall, Inc. Englewood Cliffs, NJ. 366 p.

15. Wirth, N. 1986. Algorithms & Data structures. Prentice-Hall International Inc. Singapore. 288 p.

16. Wood, D. 1993. Data structures, algorithms, and performance. Addison-Wesley Publishing Company, Inc. Reading, MA. 594 p.

ดัชนี

algorithm 111
array 1
big O 113
binary search tree 80
binary search 105
binary tree 64
data structure 91
dijkstra 50
directed graph 47
edge 46
FIFO 35
graph 45
LIFO 27, 28
linked list 13
multidimensional array 1
node 46, 47
path 48
pop 28, 29, 31, 32
push 28, 29, 31, 32
queue 35
search 105
sequential search 105
stack 27
string 2
traversal 77
tree 63
undirected graph 46
แถวลำดับ 1
โครงสร้างข้อมูล 1, 9, 12, 27, 35, 45, 63, 68, 91,105
กราฟ 45
กราฟไม่ระบุทิศทาง 46, 48, 53

กราฟระบุทิศทาง 47, 49, 53
การเรียงลำดับแบบเลือก 93
การเรียงลำดับแบบแทรก 91
การเรียงลำดับแบบผสาน 99
การเรียงลำดับแบบฟอง 96
การแวะผ่านแนวกว้าง 54
การแวะผ่านแนวลึก 58
การแวะผ่านกราฟ 54
การค้นหา 105
การค้นหาแบบเรียงลำดับ 105
การค้นหาแบบทวิภาค 106
การท่องต้นไม้แบบก่อนลำดับ 83
การท่องต้นไม้แบบตามลำดับ 83
การท่องต้นไม้แบบหลังลำดับ 83
คิว 35
ต้นไม้ 63
ต้นไม้ n ดีกรี 64
ต้นไม้ค้นหาทวิภาค 80
ต้นไม้ทวิภาค 64, 65
ต้นไม้ทวิภาคแบบเต็ม 67
ต้นไม้ทวิภาคแบบสมดุล 67
ต้นไม้ทวิภาคแบบสมบูรณ์ 67
ต้นไม้ลำดับ 65
บิก โอ 113
ระเบียน 13
รายการโยง 13
ลิงค์ลิสต์ 28, 39
วิถี 47, 48
สแตก 27
หลักการแบ่งแยกและพิชิต 99
อัลกอริทึม 111
อัลกอริทึมการเรียงลำดับ 91

โครงสร้างข้อมูลและอัลกอฯ
ISBN 974 - 92372 - 7 - 7

9 789749 237274

C 112
5560 150.00 บาท

มูลนิธิส่งเสริมโอลิมปิกวิชาการและพัฒนามาตรฐานวิทยาศาสตร์ศึกษา
ในพระอุปถัมภ์สมเด็จพระเจ้าพี่นางเธอ เจ้าฟ้ากัลยาณิวัฒนา กรมหลวงนราธิวาสราชนครินทร์

สำนักงานตั้งอยู่ในบริเวณ คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ถนนพญาไท ปทุมวัน กทม. 10330
โทร. 0-2252-8916, 0-2252-8917, แฟกซ์. 0-2252-8917