ระบบค้นหาเส้นทางอัจฉริยะในกรุงเทพฯ (BACK-END)

INTELLIGENT BANGKOK TRAFFIC ROUTER (BACK-END)

นาย อัครพล วังพิชัย

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2546

ระบบค้นหาเส้นทางอัจฉริยะในกรุงเทพฯ (BACK-END)

INTELLIGENT BANGKOK TRAFFIC ROUTER (BACK-END)

โดย

นาย อัครพล วังพิชัย

อาจารย์ที่ปรึกษา

ดร. สุรินทร์ กิตติธรกุล

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2546

ปริญญานิพนธ์ ปีการศึกษา 2546

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ระบบค้นหาเส้นทางอัจฉริยะในกรุงเทพฯ (BACK-END)

INTELLIGENT BANGKOK TRAFFIC ROUTER (BACK-END)

คณะผู้จัดทำ นาย อัครพล วังพิชัย รหัส 43010538

..... อาจารย์ที่ปรึกษา

(ดร. สุรินทร์ กิตติธรกุล)

ระบบค้นหาเส้นทางอัจฉริยะในกรุงเทพฯ (BACK-END)

นาย อัครพล วังพิชัย 43010538
ดร. สุรินทร์ กิตติธรกุล อาจารย์ที่ปรึกษา
ปีการศึกษา 2546

บทคัดย่อ

โครงงานนี้ถูกแบ่งออกเป็นสองส่วน ได้แก่ส่วนแสดงผลที่ติดต่อกับผู้ใช้ (Front-end) และส่วนระบบภายในที่ใช้ในการค้นหาเส้นทาง (Back-end) โดยโครงงานในส่วน Back-end นี้ เป็นการคิดค้นอัลกอริทึม และพัฒนาโปรแกรมสำหรับใช้ในการค้นหาเส้นทางจากสถานที่หนึ่ง (โหนดต้นทาง) ไปยังอีกสถานที่หนึ่ง (โหนดปลายทาง) ภายในกรุงเทพมหานคร โดยระบบจะรับค่าโหนดต้นทางที่ผู้ใช้ป้อนเข้ามาจากระบบในส่วน Front-end มาทำการค้นหาเส้นทางและที่ดีที่สุดตามเงื่อนไขของผู้ใช้ และส่งค่าเส้นทางผลลัพธ์กลับไปให้ยังส่วน Front-end เพื่อทำการแสดงผลต่อไป ทั้งนี้ อัลกอริทึมที่นำมาใช้ในการค้นหาเส้นทาง ได้แก่ การแตกโหนด จากโหนดต้นทางไปยังโหนดข้างเคียงในลักษณะโครงสร้างข้อมูลแบบต้นไม้ (Spanning Tree) จนกว่าจะพบโหนดปลายทาง รวมถึงยังมีอัลกอริทึมที่ใช้ในการกำจัดเส้นทางที่เกิดการวนลูปขึ้น โดยข้อมูลของแต่ละโหนด รวมทั้งสภาพการจราจรในพื้นที่บริเวณต่างๆ จะถูกเก็บไว้ภายในฐานข้อมูลบน Server นอกจากนี้ ระบบยังให้ผู้ใช้ระบบสามารถเลือกเงื่อนไขในการค้นหาเส้นทางได้ กล่าวคือ ต้องการเส้นทางที่ใช้เวลาน้อยที่สุด, มีระยะทางสั้นที่สุด หรือเป็นเส้นทางที่ไม่เสียค่าใช้จ่ายในการเดินทาง

INTELLIGENT BANGKOK TRAFFIC ROUTER (BACK-END)

Mr. Akkarapol Wangpichai
Dr. Surin Kittitornkun Advisor
Academic Year 2003

ABSTRACT

The entire project is divided into two parts which are the front-end: the part that is used to interact with users, and the back-end: the internal system which is used to find the best route. In the back-end part, we created algorithms and developed the program for searching a route from one place to another within the area of Bangkok. The system will receive a source node that user enters in the front-end part, use it to find the best route according to user's conditions, and then return the result route back to the front-end part for a display. The algorithm we use for searching the best route is the spanning tree algorithm, which means we will continue spanning from one node to another until we reach the destination node. Other than the spanning tree algorithm, we still have the algorithms used for discard any routes that has a loop. All information of each node including a traffic condition on each particular node is kept on a server. Moreover, the system allows users the ability to choose the criteria for searching the best route which are duration, distance and fee.

สารบัญ

หน้าที่

สารบัญตาราง

หน้าที่

สารบัญภาพ

หน้าที่

บทที่ 1
บทนำ

1.1 ที่มาและหลักการ

ปัญหาสภาพการจราจรในเขตกรุงเทพมหานครติดขัดนั้น เป็นปัญหาใหญ่ที่เกิดขึ้นมานานแล้ว และไม่สามารถแก้ไขให้ได้ผลอย่างชัดเจนนัก ดังนั้นจึงเกิดแนวความคิดที่จะหาวิธีการแก้ปัญหานี้ โดยอาศัยเทคโนโลยีของการสื่อสารข้อมูลผ่านเครือข่ายไร้สายมาประยุกต์ใช้ ยกตัวอย่างเช่นการใช้บริการ SMS (Short Messaging Service) ของผู้ให้บริการโทรศัพท์มือถือ ซึ่งในปัจจุบันแทบจะกล่าวได้ว่าโทรศัพท์มือถือนั้นเป็นปัจจัยที่ใหม่ที่ผู้คนขาดไม่ได้แล้ว ดังนั้นการเลือกใช้วิธีนี้จึงเป็นวิธีการที่สะดวกและสามารถใช้ได้กับผู้ใช้กลุ่มใหญ่ด้วย

สาเหตุหลักที่ทำให้เกิดโครงงานนี้คือ

- ปัญหาของการจราจรในกรุงเทพมหานคร
  - อัตราการเพิ่มขึ้นของปริมาณรถยนต์ส่วนบุคคลมีแนวโน้มเพิ่มขึ้นเรื่อยๆ
  - ผู้ขับขี่มีความชำนาญในเส้นทางที่จำกัดเฉพาะที่ตนใช้อยู่ประจำ
  - จำนวนช่องทางการจราจรและทิศทางการจราจรมีการเปลี่ยนแปลงอยู่เสมอๆ
- คุณสมบัติของเครือข่ายสื่อสารข้อมูลด้านการจราจรในกรุงเทพฯ ซึ่งยังไม่สามารถเข้าถึงผู้ใช้รถยนต์ได้อย่างทั่วถึง ทั้งนี้เนื่องจาก
  - การให้บริการข้อมูลโดยไม่เจาะจงเส้นทางของผู้ใดผู้หนึ่ง
  - การให้บริการข้อมูลในเครือข่ายอินเตอร์เน็ทผ่านสื่อที่ต้องการความเร็วสูง ยังไม่มีการให้บริการผ่านเครือข่ายไร้สายโดยเฉพาะทางโทรศัพท์มือถือ
  - ไม่มีบริการค้นหาเส้นทางให้กับผู้ขับขี่รถยนต์ซึ่งรู้เฉพาะเส้นทางที่ถนัดเพียงไม่กี่เส้นทาง

ด้วยเหตุผลดังกล่าวข้างต้นนี้ คณะผู้จัดทำโครงงานจึงได้มีแนวคิดในการพัฒนาระบบการค้นหาเส้นทางภายในกรุงเทพฯ เพื่ออำนวยความสะดวกในการค้นหาเส้นทางเดินรถจากสถานที่หนึ่ง ไปยังอีกสถานที่หนึ่งสำหรับผู้ที่ไม่ทราบเส้นทาง รวมทั้งการค้นหาเส้นทางที่มีสภาพการจราจรที่ดีที่สุดในช่วงเวลานั้นๆสำหรับผู้ที่ต้องการหลีกเลี่ยงเส้นทางเดินรถที่การจราจรติดขัด โดยให้บริการแก่ผู้ใช้ผ่านทางโทรศัพท์มือถือ และอินเทอร์เน็ท

1.2 วัตถุประสงค์

1. เพื่อศึกษาการออกแบบระบบ และการนำอัลกอริทึมมาประยุกต์ในการค้นหาเส้นทางที่ดีที่สุด รวมถึงอัลกอริทึมในการกำจัดเส้นทางที่มีการวนลูป
2. เพื่อศึกษาการออกแบบ และการใช้งานระบบฐานข้อมูล

3. เพื่อสร้างระบบที่อำนวยความสะดวกในการค้นหาเส้นทางที่ดีที่สุดในกรุงเทพฯ อ้างอิงจากสภาพการจราจรในช่วงเวลานั้นๆ

1.3 ประโยชน์ที่คาดว่าจะได้รับ

1. ได้รับความรู้ ความสามารถในการเขียนโปรแกรมภาษาจาวา
2. ได้รับความรู้ในการออกแบบระบบฐานข้อมูล
3. ได้รับความรู้ในการนำอัลกอริทึมมาประยุกต์ใช้ให้ตรงกับความต้องการในการออกแบบระบบ
4. เป็นประโยชน์ต่อผู้ใช้ระบบที่ต้องการค้นหาเส้นทางในการเดินทางที่ไม่รู้จัก หรือค้นหาเส้นทางที่สามารถหลีกเลี่ยงสภาพการจราจรที่ติดขัดในช่วงเวลานั้นๆ

1.4 ขอบเขตของโครงงาน

ระบบทั้งหมดถูกแบ่งออกเป็นสองส่วน คือส่วน Front-end ซึ่งเป็นส่วนแสดงผลที่ติดต่อกับผู้ใช้ และส่วน Back-end ซึ่งเป็นระบบภายในที่ใช้ในการค้นหาเส้นทาง โดยปริญญานิพนธ์ชิ้นนี้จะเป็นโครงงานในส่วน Back-end ซึ่งเป็นส่วนที่รับค่าข้อมูลจากผู้ใช้มาจากส่วน Front-end ซึ่งอาจมาจากการใช้โทรศัพท์มือถือ หรือเว็บเบราเซอร์ แล้วส่งค่าเข้าสู่โปรแกรมเพื่อทำการค้นหาเส้นทางที่ดีที่สุด และเมื่อได้เส้นทางผลลัพธ์ ก็จะส่งกลับไปยังระบบในส่วน Front-end เพื่อทำการแสดงผลแก่ผู้ใช้ต่อไป

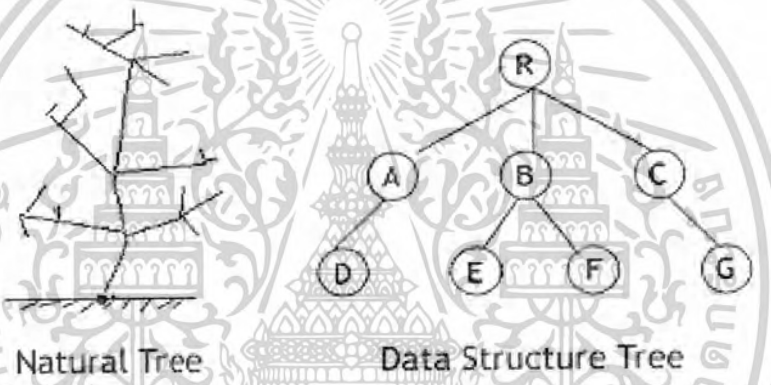
โดยโปรแกรมที่เขียนขึ้นเพื่อใช้ในการค้นหาเส้นทางนั้น จะใช้ภาษาจาวา ในการพัฒนา โดยแรกเริ่มจะเขียน และทำการรันโดยใช้ JBuilder 7 ของ Borland หลังจากนั้นจึงแยกออกมาเขียนโดยใช้ text editor ทั่วไป เช่น Word Pad และนำมาทำการรันโดยใช้ Java Development Kit (JDK) 1.4.2_04 ของ Sun หรือที่ปัจจุบันรวมเป็นแพ็กเกจเรียกว่า Software Development Kit (SDK) เพื่อให้ง่ายต่อการติดต่อกับฐานข้อมูลที่ใช้ MySQL ในการเก็บข้อมูลของแต่ละโหนด รวมถึงสภาพการจราจรในช่วงเวลาต่างๆของแต่ละโหนด ทั้งนี้ โปรแกรมที่เขียนขึ้นนี้ จะมีความสามารถในการทำงานดังต่อไปนี้

- ค้นหาเส้นทางจากสถานที่ต้นทาง ไปยังสถานที่ปลายทางตามที่ผู้ใช้กำหนด
- ในการค้นหาเส้นทางที่ดีที่สุดนั้น จะขึ้นอยู่กับเงื่อนไขของผู้ใช้ว่าต้องการเส้นทางที่ดีที่สุดในลักษณะใด กล่าวคือ ต้องการเส้นทางที่ใช้ระยะเวลาน้อยที่สุด, เส้นทางที่มีระยะทางสั้นที่สุด หรือเป็นเส้นทางที่ไม่ต้องเสียค่าธรรมเนียมในการผ่านทางใดๆ โดยผู้ใช้สามารถเลือกเงื่อนไขอย่างใดอย่างหนึ่ง หรือหลายเงื่อนไขพร้อมกันก็ได้
- ในระหว่างดำเนินการค้นหาเส้นทาง สามารถตัดเส้นทางที่เกิดลูปทิ้งไปได้
- นอกจากการแสดงเส้นทางผลลัพธ์ที่ดีที่สุดแล้ว ยังสามารถคำนวณ และแสดงผลของเวลาโดยประมาณที่ต้องใช้ในการเดินทางในเส้นทางผลลัพธ์, ระยะทางรวมของเส้นทาง รวมถึงเงินค่าธรรมเนียมที่ต้องเสียในระหว่างที่ใช้เส้นทางผลลัพธ์ด้วย

บทที่ 2
ทฤษฎีพื้นฐานและระบบที่มีอยู่ในปัจจุบัน

2.1 โครงสร้างข้อมูลแบบต้นไม้ (Tree)

โครงสร้างข้อมูลแบบต้นไม้ (Tree) เป็นโครงสร้างชนิดไม่เป็นโครงเส้นที่สำคัญที่สุดของโครงสร้างข้อมูล โครงสร้างต้นไม้มีความสำคัญอย่างใกล้ชิดกับธรรมชาติของข่าวสารและวิธีการแปลงข่าวสารมากที่สุด โครงสร้างต้นไม้มีลักษณะคล้ายกิ่งก้านของต้นไม้ แต่จะแตกต่างตรงที่ต้นไม้ตามธรรมชาติจะงอกจากล่างขึ้นสู่ด้านบน ส่วนต้นไม้ที่อยู่ในคอมพิวเตอร์จะมีลักษณะเจริญจากบนลงมาล่าง ดังรูปที่ 2-1



รูปที่ 2-1 ลักษณะของต้นไม้ปกติ (ซ้าย) กับต้นไม้ในวิชาโครงสร้างข้อมูล (ขวา)

จากรูปโครงสร้างข้อมูลแบบต้นไม้ในรูปที่ 2-1 จุดที่มีการแตกกิ่งก้านออกไปจะเรียกว่า โหนด (Node) โดยข่าวสารจะเก็บอยู่ที่โหนด กิ่งที่ต่อระหว่างโหนดจะแสดงความสัมพันธ์ระหว่างโหนดเรียกว่า ลิงค์ (Link) และจุดปลายของแต่ละโหนดก็เรียกว่าโหนดด้วยเช่นกัน

ถ้าเปรียบเทียบโครงสร้างข้อมูลแบบต้นไม้ กับโครงสร้างข้อมูลแบบสแตค (Stack), คิว (Queue) และลิงค์ลิสต์ (Linked List) จะพบว่าการจัดเก็บ และการค้นหาข้อมูลของโครงสร้างข้อมูลแบบต้นไม้นั้นทำงานได้ค่อนข้างเร็วกว่า เนื่องจากใช้หลักการเข้าถึงข้อมูลแบบไบนารีเสิร์ชทรี (Binary Search Tree) ขณะที่โครงสร้างข้อมูลอีกสามตัวนั้น ใช้หลักการเข้าถึงข้อมูลแบบตามลำดับ (Sequential Search) แต่ทั้งนี้ก็ขึ้นอยู่กับลักษณะของข้อมูลที่ถูกเก็บไว้ในโครงสร้างข้อมูลแบบต้นไม้ว่าเป็นอย่างไรด้วย

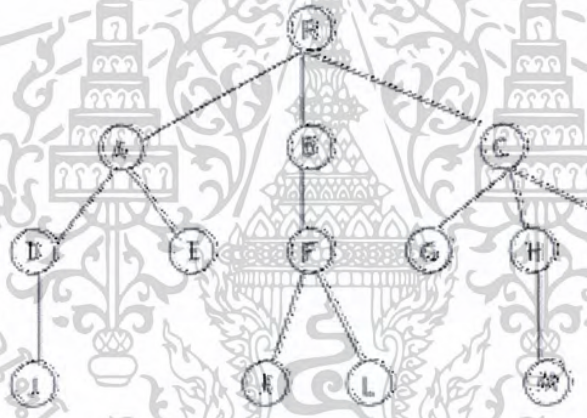
โครงสร้างข้อมูลแบบต้นไม้มีหลายประเภท เช่น ทรี (Tree), ไบนารีทรี (Binary Tree), ไบนารีเสิร์ช ทรี (Binary Search Tree), AVL Tree, Heap Tree และ B Tree เป็นต้น

2.1.1 นิยามของโครงสร้างต้นไม้

เราสามารถนิยามโครงสร้างต้นไม้อย่างรีเคอร์ซีฟ (recursive) ว่าเป็นกลุ่มของโหนด T ดังนี้

- มีโหนดพิเศษโหนดหนึ่งเรียกว่า ราก หรือรูท (Root node) ได้แก่โหนด R ในรูปที่ 2-1
- โหนดอื่นๆที่ไม่ใช้รูท สามารถแบ่งย่อยออกเป็น n กลุ่ม โดยที่แต่ละกลุ่มไม่มีโหนดร่วมกันเลย สมมติให้ชื่อแต่ละกลุ่มเป็น T1, T2, T3,... , Tn โดยที่ n >= 0 ซึ่งแต่ละกลุ่มก็เป็นต้นไม้เหมือนกัน แต่จะเรียกว่าเป็นต้นไม้ย่อย (Subtree) ของโหนด R เช่นในรูปที่ 2-1 สามารถสรุปได้ว่า
  - R เป็นรูทโหนดของต้นไม้ย่อย A, B และ C
  - A เป็นรูทโหนดของต้นไม้ย่อย D
  - B เป็นรูทโหนดของต้นไม้ย่อย E และ F
  - C เป็นรูทโหนดของต้นไม้ย่อย G

ลักษณะของโครงสร้างต้นไม้มีลักษณะเหมือนการลำดับบรรพบุรุษ ดังนั้นชื่อที่ใช้เรียกจึงมาจากบรรพบุรุษ เช่น พ่อ (Father), ลูก (Son), หลาน (Grandson) ฯลฯ แต่ที่ใช้บ่อยมีดังนี้



รูปที่ 2-2 ตัวอย่างโครงสร้างข้อมูลแบบต้นไม้

- โหนดพ่อแม่ (Parent Node) และโหนดลูก (Child Node) จะมีความสัมพันธ์ในลักษณะของพ่อลูก โดยโหนดที่มีโหนดพ่อแม่ จะถูกเรียกว่าเป็นโหนดลูก ในขณะเดียวกัน โหนดที่มีโหนดลูก ก็จะมีฐานะเป็นโหนดพ่อแม่โหนดลูก พิจารณาจากรูปที่ 2-2 จะได้ว่า
  - R เป็นโหนดพ่อแม่ของโหนด A, B และ C หรือโหนด A, B และ C เป็นโหนดลูกของโหนด R
  - A เป็นโหนดพ่อแม่ของโหนด D และ E หรือ D และ E เป็นโหนดลูกของโหนด A
  - C เป็นโหนดพ่อแม่ของโหนด G, H และ I หรือ G, H และ I เป็นโหนดลูกของโหนด C

- F เป็นโหนดพ่อแม่ของโหนด K และ L หรือ K และ L เป็นโหนดลูกของโหนด F

- โหนดพี่น้อง (Sibling Node, Brother Node) หมายถึงโหนดที่มีพ่อเป็นโหนดเดียวกัน พิจารณาจากรูปที่ 2-2 จะได้ว่าโหนด A, B และ C เป็นพี่น้องกัน และโหนด D และ E เป็นพี่น้องกัน รวมถึงโหนด K และ L ต่างก็เป็นพี่น้องกัน เป็นต้น
- กิ่ง (Branch) หมายถึงเส้นที่เชื่อมกันระหว่างโหนดพ่อ กับโหนดลูก
- ลีฟโหนด (Leaf Node) หมายถึง โหนดที่อยู่นอกสุดของโครงสร้างข้อมูลแบบต้นไม้ หรือโหนดที่ไม่มีลูก หรืออีกนัยหนึ่งก็คือโหนดที่มีค่าดีกรีเท่ากับ 0 นั่นเอง พิจารณาจากรูปที่ 2-2 จะได้ว่าลีฟโหนดคือ โหนด E, G, I, J, K, L และ M
- โหนดที่ไม่ใช่ลีฟโหนด (Non-Leaf Node) หมายถึงโหนดที่มีค่าดีกรีมากกว่า 0
- ดีกรี (Degree) ของโหนดใดๆ หมายถึงจำนวนลูกของโหนดนั้นๆ พิจารณาจากรูปที่ 2-2 จะได้ว่า
  - โหนด A มีดีกรีเท่ากับ 2
  - โหนด B มีดีกรีเท่ากับ 1
  - โหนด C มีดีกรีเท่ากับ 3 เป็นต้น
- ระดับของโหนด (Level) หมายถึงระยะทางตามแนวดิ่งของโหนดนั้น ว่าอยู่ห่างจากรูทโหนดเท่าใด ถ้ากำหนดว่ารูทโหนดของต้นไม้อยู่ในระดับที่ 1 และกิ่งทุกกิ่งมีความยาวเท่ากันหมด คือ 1 หน่วย เลขระดับของโหนดใดๆ คือจำนวนกิ่งที่น้อยที่สุดจากรูทโหนดบวกหนึ่ง เช่น ในรูปที่ 2-2 โหนด B อยู่ในระดับที่ 2 และโหนด L อยู่ระดับที่ 4 เป็นต้น
- บรรพชน (Ancestor) คือโหนดทุกโหนดบนเส้นทางจากรูทโหนดของทรี มายังโหนดนั้นๆ ดังนั้นจากตัวอย่างในรูปที่ 2-2 Ancestor ของโหนด F คือ โหนด R และ B เป็นต้น
- ผู้สืบสกุล (Descendant) หมายถึงโหนดทุกโหนดที่อยู่บนต้นไม้ย่อยทางซ้าย และขวาของโหนดนั้นๆ จากตัวอย่างรูปที่ 2-2 Descendant ของโหนด C คือ โหนด G, H, I และ M
- ความสูง หรือความลึก (Height, Depth) หมายถึงระดับของโหนดที่อยู่ชั้นล่างสุดของต้นไม้ หรืออาจเรียกว่าระดับสูงสุดของต้นไม้

2.1.2 การแทนโครงสร้างต้นไม้ในคอมพิวเตอร์

ต้นไม้สามารถเก็บในคอมพิวเตอร์ โดยอาศัยโครงสร้างของแต่ละโหนด

ข่าวสาร

Son 1	Son 2	Son 3	.....	Son N

รูปที่ 2-3 แสดงโครงสร้างโหนดของต้นไม้ทั่วไป

เนื่องจากแต่ละโหนดภายในโครงสร้างข้อมูลแบบต้นไม้จะมีโหนดลูกกี่โหนดก็ได้ สมมติให้ส่วนลิงค์ของโหนดมี K ลิงค์ ถ้าโหนดใดโหนดหนึ่งในต้นไม้นั้นมีโหนดลูก 3 โหนด ก็จะได้ค่า K มีค่าเท่ากับ

3 ลิงค์ คือ Son1, Son2 และ Son3 แต่ถ้าอีกโหนดหนึ่งในต้นไม้นั้นมีโหนดลูก 5 โหนด โครงสร้างของโหนดนั้นจะมีส่วนลิงค์ 5 ลิงค์ คือลิงค์ Son1 ถึง Son5

จะเห็นว่าการแทนโหนดในลักษณะนี้ ขนาดของพื้นที่หน่วยความจำที่ใช้สร้างแต่ละโหนดจะมีขนาดไม่คงที่ขึ้นอยู่กับจำนวนโหนดลูกของแต่ละโหนด การจัดเก็บโครงสร้างที่มีขนาดไม่เท่ากันตลอดนี้จะมีความยุ่งยากมากในการจัดสรรพื้นที่หน่วนยความจำให้อีกโหนด ซึ่งวิธีที่จะช่วยแก้ปัญหานั้นได้ก็คือให้ขนาดของโหนดคงที่ตลอดคือ กำหนดให้โหนดทุกโหนดในต้นไม้นั้นมีลักษณะเหมือนกับโหนดที่มีค่าดีกรีสูงสุด

อย่างไรก็ตาม ในการนำโครงสร้างข้อมูลแบบต้นไม้มาใช้ในโครงงานนี้ เราไม่สามารถกำหนดให้ค่าโหนดทุกโหนดที่อยู่ภายในเส้นทางการเดินรถ มีลักษณะเหมือนกับโหนดที่มีค่าดีกรีสูงสุดได้ ทั้งนี้เนื่องจากในระหว่างดำเนินการค้นหาเส้นทาง เราไม่สามารถทราบได้ว่าโหนด (ซึ่งแทนจุดแยกแต่ละแยกของถนน) จะมีเส้นทางแตกออกไปกี่เส้นทาง นั่นคือเราไม่สามารถทราบได้ว่า ค่าดีกรีของโหนดใดเป็นค่าดีกรีที่สูงสุดภายในเส้นทางเดินรถนั้นๆ จนกว่าเราจะทำการค้นหาเส้นทางไปจนถึงโหนดปลายทางแล้วนั่นเอง

2.1.3 ประเภทของโครงสร้างข้อมูลแบบต้นไม้

2.1.3.1 ทรี (Tree)

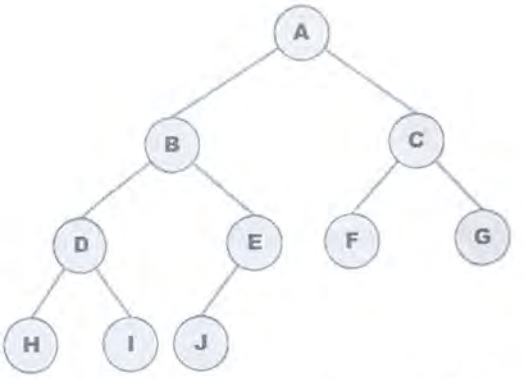
เป็นโครงสร้างข้อมูลแบบต้นไม้ ที่ประกอบด้วยรูทโหนดเพียงโหนดเดียวเท่านั้น โหนดที่เหลือจัดเป็นโหนดลูกที่แบ่งเป็นต้นไม้ย่อย (Subtree) หรืออาจกล่าวได้ว่า โหนดลูกสามารถมีได้ตั้งแต่ 0 โหนด จนถึง n โหนด เมื่อ n เป็นจำนวนเต็มใดๆ และโหนดลูกแต่ละโหนดก็สามารถเป็นโหนดพ่อแม่ ที่แต่ละโหนดสามารถมีโหนดลูกได้อีกเช่นเดียวกัน ดังลักษณะในรูปที่ 2-2

2.1.3.2 ไบนารีทรี (Binary Tree)

เป็นโครงสร้างข้อมูลต้นไม้เช่นเดียวกับทรี โดยประกอบด้วยรูทโหนดเพียงโหนดเดียวเท่านั้น โหนดที่เหลือจัดเป็นโหนดลูก ซึ่งสามารถมีได้ตั้งแต่ 0 โหนดจนถึง 2 โหนด และโหนดลูกแต่ละโหนดสามารถเป็นโหนดพ่อแม่ ซึ่งแต่ละโหนดก็จะสามารถมีโหนดลูกได้ตั้งแต่ 0 โหนดจนถึง 2 โหนดเช่นเดียวกัน หรือกล่าวได้ว่า โหนดทุกโหนดในไบนารีทรี จะมีโหนดลูกได้ไม่เกิน 2 โหนด ดังแสดงในรูปที่ 2-4

ไบนารีทรี ยังสามารถแบ่งชนิดออกได้เป็น

- Complete Binary Tree หรือไบนารีทรีแบบสมบูรณ์ หมายถึงไบนารีทรีที่โหนดที่ไม่ใช่ลีฟโหนดทุกโหนด จะต้องมีโหนดลูกอยู่ครบทั้งสองข้าง นั่นคือมีโหนดลูก 2 โหนด และลีฟโหนดทุกโหนดจะต้องอยู่ที่ระดับเดียวกัน คือระดับชั้นล่างสุดเท่านั้น ดังตัวอย่างในรูปที่ 2-5

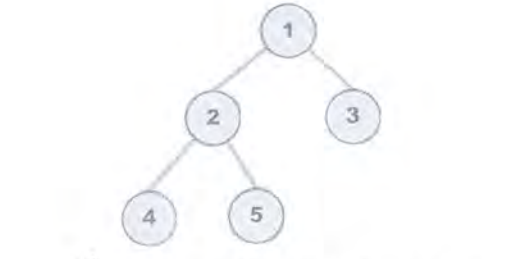


รูปที่ 2-4 ตัวอย่างไบนารีทรี ที่มีจำนวนโหนดลูกได้ไม่เกิน 2 โหนด

รูปที่ 2-5 ไบนารีทรีแบบสมบูรณ์

- Strictly Binary Tree หมายถึง ไบนารีทรีที่โหนดที่ไม่ใช่ลีฟโหนดทุกโหนด จะต้องมีโหนดลูกอยู่ครบทั้งสองข้าง ดังแสดงตัวอย่าง ในรูปที่ 2-6

รูปที่ 2-6 Strictly Binary Tree



รูปที่ 2-7 Almost Complete Binary Tree

- Almost Complete Binary Tree หมายถึงไบนารีทรีที่มีความสูงเท่ากับค่า H ใดๆ เมื่อ H มีค่าเป็นจำนวนเต็มใดๆ) และลีฟโหนดทุกโหนดจะต้องอยู่ที่ 2 ระดับล่างสุดเท่านั้น (ระดับที่ H และ H-1) ดังรูปที่ 2-7

2.1.3.3 ไบนารีเสิร์ชทรี (Binary Search Tree)

เป็นไบนารีทรีที่แต่ละโหนดสามารถมีโหนดลูกได้ตั้งแต่ 0 โหนดจนถึง 2 โหนดเท่านั้น และแต่ละโหนดในไบนารีเสิร์ชทรี ต้องมีคุณสมบัติดังนี้

- ค่าของข้อมูลในโหนดลูกทุกโหนดที่อยู่ทางด้านซ้ายของรูทโหนด ต้องมีค่าน้อยกว่าค่าของข้อมูลในรูทโหนด
- ค่าของข้อมูลในโหนดลูกทุกโหนดที่อยู่ทางด้านขวาของรูทโหนด ต้องมีค่ามากกว่าค่าของข้อมูลในรูทโหนด
- ค่าของข้อมูลในโหนดลูกทุกโหนดของโหนดพ่อแม่ใดๆของไบนารีเสิร์ชทรี ต้องมีลักษณะเช่นเดียวกับสองข้อด้านบน

ตัวอย่างของไบนารีเสิร์ชทรีเป็นไปตามรูปที่ 2-8

รูปที่ 2-8 ตัวอย่างของไบนารีเสิร์ชทรี

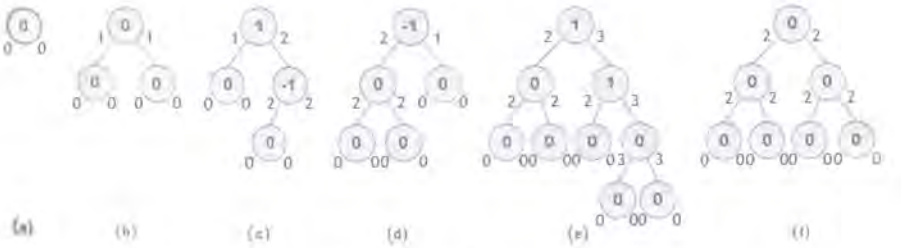
2.1.3.4 ต้นไม้ที่มีความสูงสมดุล (AVL Tree)

เป็นไบนารีทรีที่ถูกออกแบบเพื่อแก้ปัญหากรณีไบนารีเสิร์ชทรีที่ไม่สมดุล อันเป็นผลทำให้การค้นหาข้อมูลมีลักษณะเข้าถึงข้อมูลแบบตามลำดับ (Sequential Search) ซึ่งทำให้การค้นหาทำได้ช้าลง สำหรับวิธีการปรับความสมดุลของ AVL Tree จะทำให้แต่ละโหนดในต้นไม้ มีความสูงของแต่ละต้นไม้ย่อยต่างกันไม่เกิน 1

สำหรับ AVL Tree จะมีตัวประกอบสมดุล (Balance Factor หรือ BF) เป็นตัววัดความสมดุลของต้นไม้ ซึ่งจะมีค่าไม่เกิน 1 (คือมีค่าเท่ากับ 0, 1 และ -1) โดยคำนวณได้จาก ตัวประกอบสมดุล = ความสูงของต้นไม้ย่อยทางขวา – ความสูงของต้นไม้ย่อยทางซ้าย ดังนั้นถ้า AVL Tree มีตัวประกอบสมดุลที่มีค่าเกิน 1 (ไม่ว่าจะเกิดจากการเพิ่ม หรือลบโหนดออกจากต้นไม้) จะต้องทำการปรับความสมดุลให้กับ AVL Tree นั้นโดยการหมุนต้นไม้ (Rotation)

รูปที่ 2-9 แสดงภาพของ AVL Tree โดยตัวเลขในวงกลมคือค่าตัวประกอบสมดุล และตัวเลขที่อยู่

ข้างนอกวงกลมทางด้านซ้ายและขวา คือค่าความสูงของต้นไม้ย่อยซ้ายและต้นไม้ย่อยขวาของโหนดนั้นๆ ตามลำดับ



รูปที่ 2-9 AVL Tree

2.1.3.5 ฮีพทรี (Heap Tree)

เป็นไบนารีทรีที่ไม่มีปัญหาในเรื่องความไม่สมดุล เนื่องจากการจัดเก็บข้อมูลในฮีพทรี จะทำการจัดเก็บลงทีละระดับของฮีพทรี และในระดับเดียวกันจะจัดเก็บลงโหนดทางซ้ายมือก่อนโหนดทางขวามือ ส่วนการลบข้อมูลออกจากฮีพทรี จะทำการลบออกจากรูทโหนดเท่านั้น และทำการปรับค่าของโหนดต่างๆ ที่อยู่ภายในฮีพทรี ซึ่งจะทำให้ฮีพทรีมีความสมดุลตลอดเวลา ตัวอย่างของฮีพทรีแสดงในรูปที่ 2-10

ฮีพทรีมีสองประเภท คือ

- Max Heap หมายถึง โหนดลูกแต่ละโหนดจะเก็บข้อมูลที่มีค่าน้อยกว่า หรือเท่ากับข้อมูลในโหนดพ่อแม่ โดยเฉพาะข้อมูลที่ตำแหน่งรูทโหนดจะมีค่ามากที่สุด
- Min Heap หมายถึง โหนดลูกแต่ละโหนดจะเก็บข้อมูลที่มีค่ามากกว่า หรือเท่ากับข้อมูลในโหนดพ่อแม่ โดยเฉพาะข้อมูลที่ตำแหน่งรูทโหนดจะมีค่าน้อยที่สุด

รูป 2-10 (a) แสดงตัวอย่างของ Max Heap ในขณะที่รูปที่ 2-10 (b) แสดงตัวอย่างของ Min Heap

รูปที่ 2-10 ฮีพทรี

2.1.3.6 B-Tree

เป็นโครงสร้างข้อมูลชนิดหนึ่งที่ถูกสร้างขึ้นมาเพื่อตอบสนองความต้องการนำไปใช้กับโครงสร้างข้อมูลที่มีประสิทธิภาพในการเก็บข้อมูลจำนวนมากๆ B-Tree ถูกออกแบบมาเพื่อแก้ปัญหาของไบนารีเสิร์ชทรี ได้แก่กรณีที่ข้อมูลอยู่ในดิสก์ โดยแต่ละโหนดเก็บข้มูลเพียง 1 ข้อมูล และส่วนใหญ่โหนดพ่อแม่ และโหนดลูกจะไม่ถูกเก็บอยู่ติดกันในดิสก์ ทำให้ต้องทำการอ่านดิสก์ใหม่ทุกครั้งที่ต้องการอ่านข้อมูลของโหนดลูก รวมทั้งการค้นหาข้อมูลโดยทั่วไปจะต้องอ่านโหนดประมาณเท่ากับความสูงของ

ทรี ทำให้เวลาในการค้นหาข้อมูลส่วนใหญ่ใช้ไปกับการอ่านดิสก์ นอกจากนี้ B-Tree ยังแก้ไขปัญหาของ AVL Tree ในเรื่องของการเพิ่ม, ลบ และหมุน ซึ่งต้องใช้เวลามากเช่นเดียวกับไบนารีเสิร์ชทรีด้วย

B-Tree เป็นโครงสร้างข้อมูลที่ถูกพัฒนาขึ้นให้มีลักษณะเป็น Multiway Search Tree ที่มีคุณสมบัติดังนี้

- แต่ละโหนดมีโหนดลูกได้มากกว่า 2 โหนด
- แต่ละโหนดเก็บข้อมูลได้มากกว่า 1 ข้อมูล
- ลีฟโหนดทุกโหนด อยู่ที่ระดับเดียวกันที่ชั้นล่างสุด

B-Tree ขนาด Order M (M-way Tree) เมื่อ M เป็นจำนวนเต็มใดๆ หมายถึงจะมีโหนดที่มีคุณสมบัติดังนี้

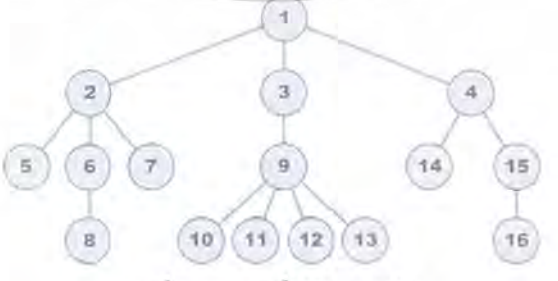
- มีช่องสำหรับเก็บข้อมูลจำนวนทั้งหมดเท่ากับ M-1 ช่อง
- มีโหนดลูกได้ไม่เกิน M โหนดสำหรับโหนดพ่อแม่แต่ละโหนด

รูปที่ 2-11 แสดงตัวอย่างของ B-Tree ขนาด Order 5

รูปที่ 2-11 B-Tree Order 5

2.1.4 ความสัมพันธ์ระหว่างทรี กับไบนารีทรี

โครงสร้างข้อมูลที่กล่าวถึงข้างต้น ยกเว้นทรี และ B-Tree จะเห็นว่ามีส่วนที่เหมือนกันคือ โหนดพ่อแม่แต่ละโหนดสามารถมีโหนดลูกได้ตั้งแต่ 0 โหนดจนถึง 2 โหนดเท่านั้น ขณะที่ทรีสามารถมีได้มากกว่า 2 โหนด และเพื่อให้ทรีสามารถทำงานได้เช่นเดียวกับโครงสร้างข้อมูลแบบต้นไม้ชนิดอื่นๆ จึงสามารถแปลงให้เป็นไบนารีทรีได้โดยวิธีดังนี้ กำหนดทรีดังรูปที่ 2-12



รูปที่ 2-12 ทรีที่ใช้ในตัวอย่าง

จากทรีที่กำหนดให้ มีขั้นตอนการแปลงทรีดังนี้

- ให้ความสำคัญกับระดับบน ก่อนระดับล่าง
- ให้ความสำคัญกับโหนดลูกที่อยู่ทางด้านซ้ายมือของโหนดพ่อแม่แต่ละโหนด ก่อนโหนดลูกที่อยู่ทางด้านขวามือ
- แต่ละโหนดสามารถมีโหนดลูกได้ตั้งแต่ 0 – 2 โหนดเท่านั้น

ผลลัพธ์ที่ได้หลังจากสามขั้นตอนนี้ คือ Ordered Tree แสดงในรูปที่ 2-13

รูปที่ 2-13 Ordered Tree ผลลัพธ์ที่ได้

- จาก Ordered Tree ทำการหมุนเส้นเชื่อมระหว่างโหนดแต่ละโหนดประมาณ 45° หรือพิจารณาจากลักษณะของเส้นเชื่อมระหว่างโหนดแต่ละโหนด ถ้าเส้นใดมีลักษณะเป็นลูกศรแนวนอน หมายถึงโหนดนั้นจะกลายเป็นโหนดลูกทางด้านขวา ขณะที่เส้นที่มีลักษณะเป็นลูกศรแนวตั้งหรือแนวเฉียง หมายถึงโหนดลูกของโหนดนั้น จะกลายเป็นลูกโหนดทางด้านซ้าย ดังแสดงผลลัพธ์ในรูปที่ 2-14

รูปที่ 2-14 ไบนารีทรีผลลัพธ์ที่ได้จากการแปลง Ordered Tree

2.2 โครงสร้างข้อมูลแบบกราฟ (Graph)

กราฟ (Graph) เป็นโครงสร้างข้อมูลที่ใช้แสดงความสัมพันธ์ระหว่างออบเจ็กต์ (Object) โดยประกอบด้วยกลุ่มของโหนด (Vertex) ที่ใช้แทนออบเจ็กต์ และกลุ่มของเส้นเชื่อม (Edge) ระหว่างโหนด กรณีที่ออบเจ็กต์ตั้งแต่ 2 ออบเจ็กต์ขึ้นไปมีความสัมพันธ์กัน ก็จะมีเส้นเชื่อมระหว่างออบเจ็กต์ หรือโหนดเหล่านั้น

หลักการเกี่ยวกับกราฟสามารถนำไปประยุกต์ใช้ในงานต่างๆในชีวิตประจำวันได้ เช่น การคำนวณหาระยะทาง หรือเวลาที่สั้นที่สุดที่สามารถเดินทางไปยังจุดต่างๆตามที่กำหนดไว้ในแผนการเดินทางของนักท่องเที่ยว พนักงานขาย หรืออื่นๆ เพื่อที่จะได้วางแผนการเดินทางล่วงหน้าได้อย่างมีประสิทธิภาพ ทั้งในเรื่องของเวลาและค่าใช้จ่าย นอกจากนี้ยังสามารถนำไปใช้ในการวางแผนการทำงานในส่วนของกระบวนการผลิต หรือการวางแผนโครงการ ซึ่งจะต้องมีการคำนวณเรื่อง Critical Path ซึ่งเป็นการคำนวณหาเส้นทางของกระบวนการผลิต หรือการดำเนินงานของโครงการ ที่จะทำให้สามารถบรรลุเป้าหมาย หรือเสร็จสิ้นได้ว่าจะต้องใช้เวลาเท่าไร นอกจากนี้ยังใช้โครงสร้างของกราฟแทนโหนด และความสัมพันธ์ระหว่างโหนดในเรื่องเครือข่ายคอมพิวเตอร์ได้อีกด้วย

กล่าวได้ว่า โครงสร้างข้อมูลแบบกราฟ จะประกอบไปด้วยโหนดและเส้นเชื่อมระหว่างโหนด (กรณีที่โหนดเหล่านั้นมีความสัมพันธ์กัน) โดยสามารถเขียนแทนด้วยสัญลักษณ์ได้ดังนี้

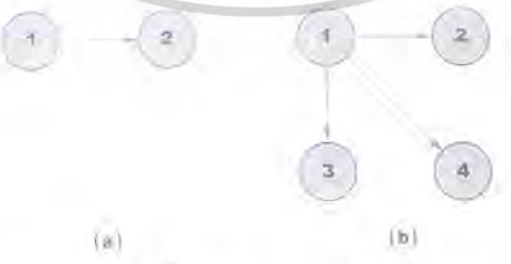
$G = (V, E)$ เมื่อ G คือกราฟ
V คือกลุ่มของโหนด
E คือเส้นเชื่อมระหว่างโหนด

2.2.1 ประเภทของกราฟ

กราฟแบ่งออกได้เป็น 3 ประเภท ดังนี้

2.2.1.1 Direct Graph (Digraph)

เป็นกราฟที่มีเส้นเชื่อมระหว่างโหนด และมีการแสดงทิศทางของการเชื่อมต่อ ลักษณะดังรูปที่ 2-15



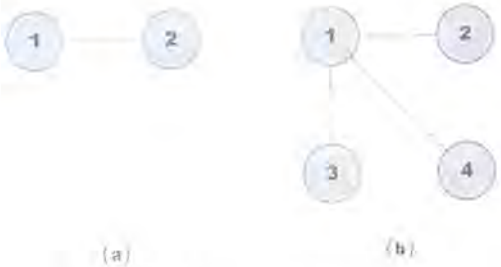
รูปที่ 2-15 Direct Graph

จากรูป 2-15 (a) หมายถึงมีเส้นทางจากโหนด 1 ไปยังโหนด 2

2-15 (b) หมายถึงมีเส้นทางจากโหนด 1 ไปยังโหนด 2 และโหนด 3 ไปยังโหนด 4

2.2.1.2 Undirected Graph

เป็นกราฟที่มีเส้นเชื่อมระหว่างโหนด แต่ไม่แสดงทิศทางของการเชื่อมต่อ ลักษณะดังรูปที่ 2-16



รูปที่ 2-16 Undirected Graph

จากรูป 2-16 (a) หมายถึงมีเส้นทางจากโหนด 1 ไปยังโหนด 2 และมีเส้นทางจากโหนด 2 ไปยังโหนด 1 ในเส้นทางเดียวกัน

จากรูป 2-16 (b) หมายถึงมีเส้นทางจากโหนด 1 ไปยังโหนด 2, โหนด 3 และโหนด 4 ขณะเดียวกันกับที่มีเส้นทางจากโหนด 4 ไปยังโหนด 1, มีเส้นทางจากโหนด 3 ไปยังโหนด 1 และมีเส้นทางจากโหนด 2 ไปยังโหนด 1 ในเส้นทางเดียวกัน

2.2.1.3 Cyclic Graph

เป็นกราฟที่มีเส้นทางเกิดจากเส้นเชื่อมระหว่างโหนดที่มีลักษณะเป็นวงจรปิด (Cycle) หมายถึงมีโหนดต้นทางและโหนดปลายทางเป็นโหนดเดียวกัน โดย Cyclic Graph สามารถเป็นได้ทั้ง Direct Graph (Digraph) และ Undirected Graph รูปที่ 2-17 แสดงภาพของ Cyclic Graph

รูปที่ 2-17 Cyclic Graph

เส้นทาง (Path) ของโหนดหนึ่งซึ่งเป็นโหนดต้นทางไปยังโหนดปลายทางใดๆที่ต้องการ หมายถึงเส้นทางจากโหนดต้นทางไปยังโหนดปลายทาง ซึ่งจะต้องผ่านโหนดอื่นๆที่อยู่ระหว่าง 2 โหนดนี้ ตัวอย่างเช่น จากรูป 2-17 (a) ซึ่งเป็น Cyclic Direct Graph (หรือ Cyclic Digraph) จะได้ว่าเส้นทางของโหนด 1 ไปโหนด 2 มีทั้งหมดสามเส้นทาง ดังนี้

เส้นทางที่ 1 คือ 1 – 3 – 4 – 2

เส้นทางที่ 2 คือ 1 – 3 – 2

เส้นทางที่ 3 คือ 1 – 4 – 2

เส้นทางวงจรปิด (Cycle Path) เป็นเส้นทางของโหนดต้นทางไปยังโหนดปลายทาง โดยทั้งโหนดต้นทาง และโหนดปลายทางนี้เป็นโหนดเดียวกัน ตัวอย่างเช่น จากรูปที่ 2-17 (b) ซึ่งเป็น Cyclic Direct Graph เส้นทางของโหนด 1 ไปยังโหนดทุกโหนดบนกราฟ และมีโหนดปลายทางเป็นโหนด 1 คือ 1 – 3 – 4 – 2 – 1 หรือเส้นทางของโหนด 1 ไปยังโหนดบางโหนดบนกราฟและมีโหนดปลายทางเป็นโหนด 1 คือ 1 – 3 – 2 – 1 เป็นต้น

เส้นทางจากโหนดต้นทางไปยังโหนดปลายทางใดๆ สามารถมีได้มากกว่า 1 เส้นทาง ซึ่งสามารถคำนวณจำนวนเส้นทางดังกล่าวได้โดยการนำเมตริกซ์ (Matrix) เข้ามาช่วยในการจัดการ โดยเริ่มต้นที่การแทนที่กราฟด้วยเมตริกซ์ และการคำนวณจำนวนเส้นทางระหว่างโหนดต่างๆด้วย Adjacency Matrix

2.2.2 การแทนที่กราฟด้วยเมตริกซ์

เนื่องจากกราฟเป็นโครงสร้างข้อมูลที่ประกอบด้วยโหนดจำนวน n (เมื่อ n เป็นจำนวนเต็มใดๆ) และมีเส้นเชื่อมระหว่างโหนด การแทนที่กราฟด้วยเมตริกซ์จึงต้องมีการกำหนดขนาดของข้อมูลในแนวนอน (Row) และแนวตั้ง (Column) ด้วยขนาด n x n ซึ่งจะมีการคำนวณการเชื่อมโยงระหว่างโหนดต่างๆโดยการใช้ตัวเลข 2 ตัว คือ 0 และ 1 เพื่อแทนความหมายของการเชื่อมโยงระหว่างโหนดดังนี้

ค่า 0 หมายถึง การไม่มีเส้นเชื่อมจากโหนดในแนวนอน (โหนดต้นทาง) ไปยังโหนดในแนวตั้ง (โหนดปลายทาง) ที่มีจำนวนเส้นเชื่อมเท่ากับ 1 (อาจเรียกว่าความยาวเท่ากับ 1)

ค่า 1 หมายถึง การมีเส้นเชื่อมจากโหนดในแนวนอน (โหนดต้นทาง) ไปยังโหนดในแนวตั้ง (โหนดปลายทาง) ที่มีจำนวนเส้นเชื่อมเท่ากับ 1 (อาจเรียกว่าความยาวเท่ากับ 1)

จากรูปที่ 2-17 (a) สามารถถูกแทนที่ด้วยเมตริกซ์ได้ดังแสดงในรูปที่ 2-18

รูปที่ 2-18 เมตริกซ์ที่ได้จากการแทนที่กราฟในรูป 2-17 (a)

และจากรูปที่ 2-17 (b) สามารถถูกแทนด้วยเมตริกซ์ได้ดังแสดงในรูปที่ 2-19

รูปที่ 2-19 เมตริกซ์ที่ได้จากการแทนที่กราฟในรูป 2-17 (b)

2.2.3 การคำนวณจำนวนเส้นทางระหว่างโหนดต่างๆด้วย Adjacency Matrix

หลังจากที่ได้มีการแทนที่กราฟด้วยเมตริกซ์ในหัวข้อที่ผ่านมาแล้ว จะสามารถนำเมตริกซ์ดังกล่าวมาคำนวณจำนวนเส้นทาง (ความยาว) ระหว่างโหนดต่างๆได้

2.2.3.1 การคำนวณจำนวนเส้นทางระหว่างโหนดต้นทางและโหนดปลายทางที่มีเส้นเชื่อมจำนวน 2 เส้น

เกิดจากการนำค่าของเมตริกซ์ขนาด n x n จำนวน 2 เมตริกซ์มาคูณกัน ในที่นี้คือเมตริกซ์ที่ได้จากการแทนค่ากราฟจากรูป 2-17 (นั่นคือเมตริกซ์ในรูปที่ 2-18 หรือ 2-19) โดยกำหนดให้เป็นเมตริกซ์ a และผลลัพธ์ที่ได้จากการคูณกัน จะถูกเก็บไว้ในเมตริกซ์ b

2.2.3.2 การคำนวณจำนวนเส้นทางระหว่างโหนดต้นทางและโหนดปลายทางที่มีเส้นเชื่อมจำนวน 3 เส้น

เกิดจากการนำค่าของเมตริกซ์ขนาด n x n จำนวน 2 เมตริกซ์มาคูณกัน เช่นเดียวกับวิธีข้างต้น แต่เมตริกซ์ที่ใช้ในการคูณ จะมาจากเมตริกซ์ b ในข้อที่ผ่านมา คูณกับเมตริกซ์ที่ได้จากการแทนค่ากราฟในรูป 2-17 นั่นคือเมตริกซ์ a และผลลัพธ์ที่ได้จากการคูณกัน จะถูกเก็บไว้ในเมตริกซ์ c

2.2.3.3 การคำนวณจำนวนเส้นทางระหว่างโหนดต้นทางและโหนดปลายทางที่มีเส้นเชื่อมจำนวน 4 เส้น

เกิดจากการนำค่าของเมตริกซ์ขนาด n x n จำนวน 2 เมตริกซ์มาคูณกัน เช่นเดียวกับวิธีข้างต้น แต่เมตริกซ์ที่ใช้ในการคูณ จะมาจากเมตริกซ์ c ในข้อที่ผ่านมา คูณกับเมตริกซ์ที่ได้จากการแทนค่ากราฟในรูป 2-17 นั่นคือเมตริกซ์ a และผลลัพธ์ที่ได้จากการคูณกัน จะถูกเก็บไว้ในเมตริกซ์ d

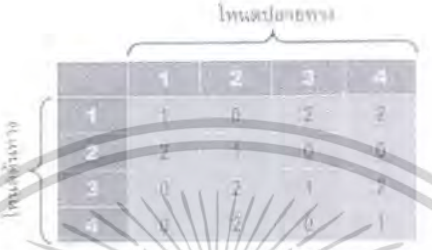
ตัวอย่างจากรูปที่ 2-17 (a) จะสามารถแทนที่ได้ด้วยเมติกซ์ดังรูปที่ 2-18 ซึ่งนำเอามาคำนวณหาจำนวนเส้นทางระหว่างโหนดต้นทาง และโหนดปลายทางด้วย Adjacency Matrix ได้ดังนี้

- เมื่อคำนวณจำนวนเส้นทางระหว่างโหนดต้นทาง และโหนดปลายทางที่มีเส้นเชื่อมจำนวน 2 เส้น ได้ค่าดังแสดงในรูปที่ 2-20

รูปที่ 2-20 การคำนวณจำนวนเส้นทางระหว่างโหนดที่มีเส้นเชื่อมจำนวน 2 เส้น ของกราฟในรูปที่ 2-17 (a)

- เมื่อคำนวณจำนวนเส้นทางระหว่างโหนดต้นทาง และโหนดปลายทางที่มีเส้นเชื่อมจำนวน 3 เส้น จะได้ค่าดังแสดงในรูปที่ 2-21
- เมื่อคำนวณจำนวนเส้นทางระหว่างโหนดต้นทาง และโหนดปลายทางที่มีเส้นเชื่อมจำนวน 4 เส้น จะได้ค่าดังแสดงในรูปที่ 2-22

รูปที่ 2-21 การคำนวณจำนวนเส้นทางระหว่างโหนดที่มีเส้นเชื่อมจำนวน 3 เส้น ของกราฟในรูปที่ 2-17 (a)



รูปที่ 2-22 การคำนวณจำนวนเส้นทางระหว่างโหนดที่มีเส้นเชื่อมจำนวน 4 เส้น ของกราฟในรูปที่ 2-17 (a)

เส้นทางจากโหนดต้นทางไปจนถึงโหนดปลายทาง เมื่อจำนวนโหนดทั้งหมดเป็น 4 โหนด มีดังนี้คือ

- เส้นทางที่เริ่มจากโหนด 1 คือ 1 – 3 – 4 – 2
- เส้นทางที่เริ่มจากโหนด 2 คือ 2 – 1 – 3 – 4
- เส้นทางที่เริ่มจากโหนด 3 คือ 3 – 4 – 2 – 1
- เส้นทางที่เริ่มจากโหนด 4 คือ 4 – 2 – 1 – 3

และต่อเนื่องจากเส้นทางเหล่านี้ จะเป็นเส้นทางที่โหนดต้นทาง และโหนดปลายทางสามารถเป็นโหนดเดียวกันได้อีกด้วย ได้แก่

- เส้นทางเส้นทางที่เริ่มจากโหนด 1 และสิ้นสุดที่โหนด 1 คือ 1 – 3 – 4 – 2 – 1
- เส้นทางเส้นทางที่เริ่มจากโหนด 2 และสิ้นสุดที่โหนด 2 คือ 2 – 1 – 3 – 4 – 2
- เส้นทางเส้นทางที่เริ่มจากโหนด 3 และสิ้นสุดที่โหนด 3 คือ 3 – 4 – 2 – 1 – 3
- เส้นทางเส้นทางที่เริ่มจากโหนด 4 และสิ้นสุดที่โหนด 4 คือ 4 – 2 – 1 – 3 – 4

ในขณะที่ตัวอย่างจากรูปที่ 2-17 (b) สามารถถูกแทนที่ด้วยเมตริกซ์ได้ดังรูปที่ 2-19 และนำมาคำนวณหาจำนวนเส้นทางระหว่างโหนดต้นทาง และโหนดปลายทางด้วย Adjacency Matrix ได้ดังนี้

- เมื่อคำนวณจำนวนเส้นทางระหว่างโหนดต้นทาง และโหนดปลายทางที่มีเส้นเชื่อมจำนวน 2 เส้น จะได้ค่าดังแสดงในรูปที่ 2-23
- เมื่อคำนวณจำนวนเส้นทางระหว่างโหนดต้นทาง และโหนดปลายทางที่มีเส้นเชื่อมจำนวน 3 เส้น จะได้ค่าดังแสดงในรูปที่ 2-24
- เมื่อคำนวณจำนวนเส้นทางระหว่างโหนดต้นทาง และโหนดปลายทางที่มีเส้นเชื่อม

จำนวน 4 เส้น จะได้ค่าดังแสดงในรูปที่ 2-25

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

รูปที่ 2-23 การคำนวณจำนวนเส้นทางระหว่างโหนดที่มีเส้นเชื่อมจำนวน 2 เส้น ของกราฟในรูปที่ 2-17 (b)

รูปที่ 2-24 การคำนวณจำนวนเส้นทางระหว่างโหนดที่มีเส้นเชื่อมจำนวน 3 เส้น ของกราฟในรูปที่ 2-17 (b)

รูปที่ 2-25 การคำนวณจำนวนเส้นทางระหว่างโหนดที่มีเส้นเชื่อมจำนวน 4 เส้น ของกราฟในรูปที่ 2-17 (b)

เส้นทางจากโหนดต้นทางไปจนถึงโหนดปลายทาง เมื่อจำนวนโหนดทั้งหมดเป็น 4 โหนด มีดังนี้คือ

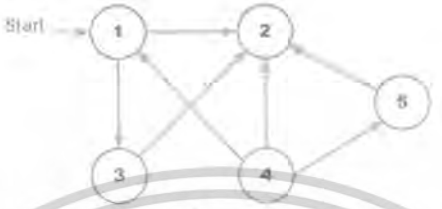
- เส้นทางเส้นทางที่เริ่มจากโหนด 1 และสิ้นสุดที่โหนด 1 มีจำนวน 15 เส้นทาง
- เส้นทางเส้นทางที่เริ่มจากโหนด 2 และสิ้นสุดที่โหนด 2 มีจำนวน 10 เส้นทาง
- เส้นทางเส้นทางที่เริ่มจากโหนด 3 และสิ้นสุดที่โหนด 3 มีจำนวน 10 เส้นทาง
- เส้นทางเส้นทางที่เริ่มจากโหนด 4 และสิ้นสุดที่โหนด 4 มีจำนวน 15 เส้นทาง

2.2.4 การเดินทางเข้าไปในกราฟ (Graph Traversal)

การเดินทางเข้าไปในกราฟ จะต้องมีการเดินทางเข้าถึงยังโหนดต่างๆ (visit) อย่างมีระบบ ซึ่งอัลกอริทึมที่ใช้ก็อาทิเช่น Depth First Search (DFS) และ Breadth First Search (BFS) เป็นต้น

2.2.4.1 Depth First Search (DFS)

เป็นอัลกอริทึมที่ทำการเดินทางเข้าถึงโหนดต่างๆในกราฟตามแนวลึก โดยเริ่มต้นจากโหนดใดโหนดหนึ่งในกราฟ ซึ่งถูกกำหนดให้เป็นโหนดเริ่มต้น ไปยังโหนดที่อยู่ติดกับโหนดนั้น จากนั้นจึงใช้โหนดใหม่นี้เป็นโหนดเริ่มต้น และทำวิธีการเช่นเดียวกันจนกระทั่งไม่สามารถไปต่อได้แล้ว คือไม่โหนดที่อยู่ติดกับโหนดนั้นแล้ว ให้ย้อนกลับไปโหนดก่อนหน้า และทำการเดินทางเข้าถึงโหนดต่างๆที่เหลือด้วยวิธีการเช่นเดียวกันจนครบทุกโหนด



รูปที่ 2-26 ตัวอย่างกราฟที่ใช้แสดงการเดินทางแบบ DFS

- เริ่มต้นจากโหนด 1 ไปยังโหนด 3 ทั้งนี้โหนดที่อยู่ติดกับโหนด 1 คือ โหนด 3 และโหนด 2 แต่ให้ความสำคัญกับโหนดที่อยู่ทางซ้ายมือมากกว่า จึงเลือกโหนด 3
- จากโหนด 3 เลือกโหนดที่อยู่ติดกัน คือโหนด 2
- จากโหนด 2 ไม่สามารถไปต่อได้ จึงกลับไปที่โหนด 3 ซึ่งไปต่ออีกไม่ได้เช่นกัน จึงต้องย้อนกลับไปที่โหนด 1 จากโหนด 1 ก็ไม่สามารถไปต่อได้ ในกราฟจึงเหลือเพียงสองโหนดเท่านั้นที่ยังไม่ถูกเดินทางเข้าถึง คือโหนด 4 และโหนด 5
- จากโหนด 4 เลือกโหนดที่อยู่ติดกันคือโหนด 5 ซึ่งเป็นโหนดสุดท้ายในกราฟที่ถูกเข้าถึง ดังนั้นการเดินทางแบบ Depth First Search (DFS) จึงเสร็จสิ้น โดยผลลัพธ์ของการเดินทางนี้ แสดงในรูปที่ 2-27

รูปที่ 2-27 ผลลัพธ์ของการเดินทางแบบ DFS

หมายเหตุ:
- ในรูปผลลัพธ์ของการเดินทาง เส้นประ แสดงการเชื่อมต่อระหว่างโหนด 2 โหนด ที่เป็นเส้นทางที่ไม่จำเป็นต้องถูกเดินทางเข้าถึง
- ในรูปผลลัพธ์ของการเดินทาง เส้นทึบ แสดงเส้นทางที่เลือกในการเชื่อมต่อระหว่างโหนด
- โหนดแต่ละโหนด จะถูกเดินทางเข้าถึงเพียง 1 ครั้งเท่านั้น
- คำว่า "โหนดที่ติดกับโหนดใดๆ" ของกราฟรูปที่ 2-26 สามารถแสดงได้ด้วย

การใช้ Adjacency Matrix ดังรูปที่ 2-28

โหนดต้นทาง \ โหนดปลายทาง	1	2	3	4	5
1	0	1	1	0	0
2	0	0	0	0	0
3	0	1	0	0	0
4	1	1	0	0	1
5	0	1	0	0	0

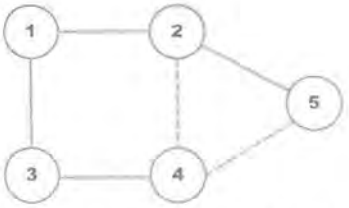
รูปที่ 2-28 แสดง Adjacency Matrix ของกราฟในรูปที่ 2-26

2.2.4.2 Breadth First Search (BFS)

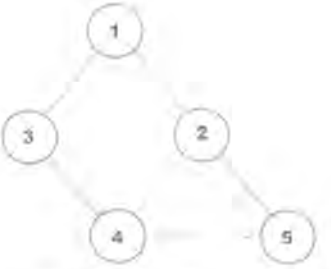
เป็นอัลกอริทึมที่ทำการเข้าถึงโหนดต่างๆในกราฟตามแนวกว้าง โดยเริ่มต้นจากโหนดใดโหนดหนึ่งในกราฟ ซึ่งถูกกำหนดให้เป็นโหนดเริ่มต้นไปยังโหนดที่อยู่ติดกับโหนดนั้นจนครบทุกโหนด เมื่อครบทุกโหนด ก็ย้อนกลับมาเริ่มต้นใหม่ที่โหนดชุดแรกนั้น เพื่อไปยังโหนดชุดอื่นๆที่ยังไม่ถูกเข้าถึงจนครบทุกโหนดในกราฟ

รูปที่ 2-29 ตัวอย่างกราฟที่ใช้แสดงการเดินทางแบบ BFS

- เริ่มต้นจากโหนด 1 ไปยังโหนด 3 และโหนด 2 (ให้ความสำคัญกับโหนดทางซ้ายมือก่อน)
- จากโหนด 3 ไปยังโหนด 4
- จากโหนด 4 ไปยังโหนด 5
- เมื่อเดินทางเข้าถึงครบทุกโหนดแล้ว ก็เป็นอันเสร็จสิ้นการเดินทางแบบ Breadth First Search (BFS) โดยผลลัพธ์ของการเดินทางนี้ แสดงในรูปที่ 2-30 และ 2-31



รูปที่ 2-30 ผลลัพธ์ของการเดินทางแบบ BFS ในรูปของกราฟ



รูปที่ 2-31 ผลลัพธ์ของการเดินทางแบบ BFS ในรูปของแผนภาพต้นไม้

หมายเหตุ: - ในรูปผลลัพธ์ของการเดินทาง เส้นประ แสดงการเชื่อมต่อระหว่างโหนด 2 โหนด ที่เป็นเส้นทางที่ไม่จำเป็นต้องถูกเดินทางเข้าถึง
- ในรูปผลลัพธ์ของการเดินทาง เส้นทึบ แสดงเส้นทางที่เลือกในการเชื่อมต่อระหว่างโหนด

2.2.5 การคำนวณหาระยะทางที่สั้นที่สุดของเส้นทางจากโหนดต้นทางไปยังโหนดต่างๆจนครบทุกโหนดในกราฟ

หลักการเกี่ยวกับกราฟในเรื่องการคำนวณหาระยะทางที่สั้นที่สุดของเส้นทางจากโหนดต้นทางไปยังโหนดต่างๆจนครบทุกโหนดในกราฟ สามารถนำไปประยุกต์ใช้ในงานต่างๆในชีวิตประจำวัน ได้แก่ การคำนวณหาระยะทาง หรือเวลาที่สั้นที่สุด เพื่อประหยัดค่าใช้จ่ายในการวางแผนการเดินทางของนักท่องเที่ยว, การคำนวณหาเส้นทางการดำเนินงานของโครงการ หรือกระบวนการผลิตเพื่อให้งานเสร็จทันตามเวลาที่กำหนด เป็นต้น

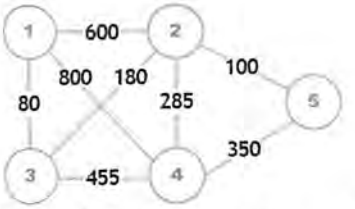
อัลกอริทึมที่ใช้ในการคำนวณหาระยะทางที่สั้นที่สุดของเส้นทางจากโหนดต้นทาง ไปยังโหนดต่างๆจนครบทุกโหนดในกราฟ ได้แก่ Minimum Spanning Tree และ Shortest Path Algorithm เป็นต้น

2.2.5.1 Minimum Spanning Tree

เป็นอัลกอริทึมที่ใช้คำนวณหาระยะทางที่สั้นที่สุดของเส้นทางจากโหนดต้นทางไปยังโหนดต่างๆจนครบทุกโหนดในกราฟ โดยมีวิธีการคำนวณดังนี้

- เลือกค่าของตัวเลขบนเส้นเชื่อมระหว่างโหนด ซึ่งอาจเป็นค่าของเวลา ค่าใช้จ่ายหรือตัวเลขใดๆก็ตามที่ต้องการนำมาคำนวณ โดยเลือกค่าที่น้อยที่สุดก่อน
- เมื่อได้ค่าตัวเลขที่น้อยที่สุดแล้ว ให้ทำการเชื่อมโยงโหนด 2 โหนดที่อยู่ซ้ายและขวา หรือบนและล่างของเส้นเชื่อมนั้นๆ
- ทำขั้นตอนแรกต่างจนกระทั่งครบทุกโหนดในกราฟ
- นำค่าของตัวเลขที่อยู่บนเส้นเชื่อมที่ได้จากการเลือกค่าที่น้อยที่สุดดังกล่าวมารวมกัน ค่าของตัวเลขนั้นคือผลลัพธ์ที่เป็นระยะทางที่สั้นที่สุดของเส้นทางจากโหนดต้นทางไปยังโหนดต่างๆจนครบทุกโหนดในกราฟ

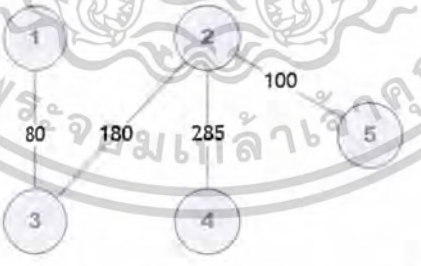
จากรูปที่ 2-32 แสดงตัวอย่างกราฟที่ประกอบด้วยโหนดทั้งหมด 5 โหนด และมีเส้นเชื่อมระหว่างโหนดเหล่านั้น โดยเส้นเชื่อมแต่ละเส้นจะมีค่าของตัวเลขบนเส้นเชื่อมนั้นๆ ซึ่งจะถูกนำมาใช้ในการคำนวณในอัลกอริทึมนี้



รูปที่ 2-32 กราฟตัวอย่างที่นำมาใช้ในการคำนวณหาเส้นทางที่สั้นที่สุด

ขั้นตอนการคำนวณโดยใช้กราฟในรูปที่ 2-32 มีดังนี้

- ค่าของตัวเลขบนเส้นทางที่มีค่าน้อยที่สุดคือ 80
- ทำการเชื่อมโหนด 1 และ 3
- ค่าของตัวเลขบนเส้นทางที่มีค่าน้อยที่สุด (ที่เหลือในกราฟ) คือ 100
- ทำการเชื่อมโหนด 2 และ 5
- ค่าของตัวเลขบนเส้นทางที่มีค่าน้อยที่สุด (ที่เหลือในกราฟ) คือ 180
- ทำการเชื่อมโหนด 2 และ 3
- ค่าของตัวเลขบนเส้นทางที่มีค่าน้อยที่สุด (ที่เหลือในกราฟ) คือ 285
- ทำการเชื่อมโหนด 2 และ 4
- นำค่าตัวเลขที่อยู่บนเส้นเชื่อมที่ได้จากการคำนวณมารวมกัน มีค่าเท่ากับ 80 + 100 + 180 + 285 เท่ากับ 645 ได้ผลลัพธ์แสดงในรูปที่ 2-33



รูปที่ 2-33 ผลลัพธ์ที่ได้จากการคำนวณโดยใช้ Minimum Spanning Tree

2.2.5.2 Shortest Path Algorithm

เป็นอัลกอริทึมที่คำนวณหาระยะทางที่สั้นที่สุดของเส้นทางจากโหนดต้นทางไปยังโหนดต่างๆจนครบทุกโหนดในกราฟ เช่นเดียวกับ Minimum Spanning Tree โดยวิธีการคำนวณใช้หลักการ Adjacency Matrix โดยเริ่มที่โหนดที่กำหนดให้เป็นโหนดต้นทาง จากนั้นคำนวณหาระยะทางที่สั้นที่สุดระหว่างโหนดนี้ กับโหนดที่อยู่ติดกัน โดยเส้นทางที่เชื่อมต่อระหว่างสองโหนดดังกล่าว สามารถมีโหนดอื่นๆเป็น

ทางผ่านได้ และเพิ่มจำนวนโหนดขึ้นทีละหนึ่งโหนด เมื่อได้เส้นเชื่อมระหว่างโหนดต่างๆจนครบทุกโหนดแล้ว นำค่าตัวเลขที่ได้ทั้งหมดมารวมกัน จะเป็นผลลัพธ์ของ Shortest Path Algorithm

จากกราฟในรูปที่ 2-32 สามารถคำนวณหาระยะทางที่สั้นที่สุดด้วยอัลกอริทึม Shortest Path Algorithm ได้ดังนี้

- แทนค่ากราฟจากรูปที่ 2-32 ด้วยการใช้ Adjacency Matrix ได้ผลลัพธ์ดังแสดงในรูปที่ 2-34

	1	2	3	4	5
1	0	600	80	800	0
2	600	0	180	285	100
3	80	180	0	455	0
4	800	285	455	0	350
5	0	100	0	350	0

รูปที่ 2-34 Adjacency Matrix ของกราฟในรูปที่ 2-32

- เริ่มที่โหนดต้นทางของกราฟ คือโหนด 2 จะได้ว่าเส้นทางที่สั้นที่สุดจากโหนด 2 ไปยังโหนดที่อยู่ติดกันที่สามารถไปได้ มีดังนี้
  - เส้นทางจาก 2 → 1 ระยะทางมีค่าเท่ากับ 600
  - เส้นทางจาก 2 → 3 ระยะทางมีค่าเท่ากับ 180
  - เส้นทางจาก 2 → 4 ระยะทางมีค่าเท่ากับ 285
  - เส้นทางจาก 2 → 5 ระยะทางมีค่าเท่ากับ 100

  ดังนั้นจึงเลือกเส้นทางจาก 2 → 5
- เพิ่มโหนด 5 ในกราฟ (2 → 5) จะได้ว่าเส้นทางที่สั้นที่สุดจากโหนด 2 ไปยังโหนด 4 ที่สามารถไปได้มีดังนี้
  - เส้นทางจาก 2 → 4 ระยะทางมีค่าเท่ากับ 285
  - เส้นทางจาก 2 → 5 → 4 ระยะทางมีค่าเท่ากับ 450

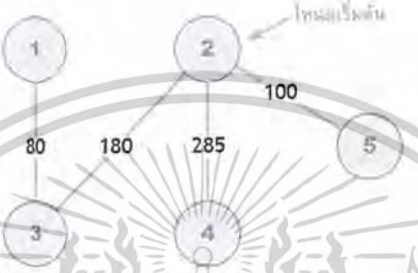
  ดังนั้นจึงเลือกเส้นทางจาก 2 → 4
- เพิ่มโหนด 5 ในกราฟ (2 → 5 และ 2 → 4) จะได้ว่าเส้นทางที่สั้นที่สุดจากโหนด 2 ไปยังโหนด 3 ที่สามารถไปได้มีดังนี้
  - เส้นทางจาก 2 → 3 ระยะทางมีค่าเท่ากับ 180
  - เส้นทางจาก 2 → 4 → 3 ระยะทางมีค่าเท่ากับ 740

  ดังนั้นจึงเลือกเส้นทางจาก 2 → 3
- เพิ่มโหนด 3 ในกราฟ (2 → 5, 2 → 4 และ 2 → 3) จะได้ว่าเส้นทางที่สั้นที่สุดจากโหนด 2 ไปยังโหนด 1 ที่สามารถไปได้มีดังนี้
  - เส้นทางจาก 2 → 1 ระยะทางมีค่าเท่ากับ 600

- เส้นทางจาก 2 → 3 → 1 ระยะทางมีค่าเท่ากับ 260
- เส้นทางจาก 2 → 4 → 1 ระยะทางมีค่าเท่ากับ 1085
- เส้นทางจาก 2 → 4 → 3 → 1 ระยะทางมีค่าเท่ากับ 820

ดังนั้นจึงเลือกเส้นทางจาก 2 → 3 → 1

- เพิ่มโหนด 1 ในกราฟ (2 → 5, 2 → 4 และ 2 → 3 → 1) จะได้ว่าเส้นทางที่สั้นที่สุดจากโหนด 2 ไปยังทุกโหนดในกราฟ (เส้นทางเป็น 2 → 5, 2 → 4 และ 2 → 3 → 1) มีค่าเท่ากับ 645 (100 + 285 + 260) ได้ผลลัพธ์ดังแสดงในรูปที่ 2-35



รูปที่ 2-35 ผลลัพธ์ที่ได้จากการคำนวณโดยใช้ Shortest Path Algorithm

2.3 รายละเอียดโปรแกรมภาษาจาวาเบื้องต้น

การเขียนโปรแกรมโดยให้มีรูปแบบเป็นลักษณะเชิงวัตถุต้องอาศัยภาษาคอมพิวเตอร์ที่ถูกออกแบบมาให้มีลักษณะพิเศษและเอื้อประโยชน์สำหรับการเขียนโปรแกรมแบบเชิงวัตถุโดยเฉพาะ มีภาษาคอมพิวเตอร์อยู่ภาษาหนึ่งที่พึ่งเกิดใหม่ และสามารถใช้เป็นเครื่องมือในการพัฒนาระบบโดยมีพื้นฐานเป็นแบบเชิงวัตถุ ภาษาที่กล่าวถึงนี้คือ ภาษาจาวา

2.3.1 ลักษณะของภาษาจาวา

ภาษาจาวาถูกพัฒนามาจากบริษัท Sun Microsystems ซึ่งจัดให้เป็นภาษาคอมพิวเตอร์ภาษาหนึ่งที่ใช้หลักการออกแบบตัวภาษาด้วยวิธีเชิงวัตถุ และตัวภาษาถูกใช้ เป็นเครื่องมือสำหรับพัฒนาโปรแกรมด้วยแนวคิดเชิงวัตถุ โดยตัวภาษามีลักษณะพิเศษดังนี้

Portability: สามารถใช้งานในสภาวะแวดล้อมที่แตกต่างได้ โดยไม่ต้องมีการปรับแต่ง

Simple: ง่ายในการเขียนโปรแกรม

Robust: ความคงสภาพในการทำงาน มีโอกาสเกิดข้อผิดพลาดที่ไม่พึงประสงค์ได้น้อย

Secure: การรองรับมาตรฐานความปลอดภัยในการใช้งานรูปแบบต่างๆ

Object-Oriented: มีหลักการของแนวคิดเชิงวัตถุ

Distributed: มีความสามารถในการประมวลผลแบบกระจาย

เหตุผลหนึ่งที่ภาษาจาวาจัดเป็นภาษาที่ก้าวกระโดดไปสู่ความนิยม คือลักษณะคำจำกัดความที่ว่า

"Write One Run Anywhere" หรือ "เขียนครั้งเดียวใช้ได้ทุกที่" ซึ่งหมายถึงเมื่อสร้างโปรแกรมภาษาจาวาแล้ว ผลที่ได้รับคือ โปรแกรมสามารถนำไปประมวลผลหรือทำงานได้ ณ เครื่องคอมพิวเตอร์ระบบปฏิบัติการใดๆ ก็ได้โดยที่ไม่ต้องมีการปรับปรุงตัวโปรแกรมทุกครั้งที่มีการเปลี่ยนการใช้งานจากระบบปฏิบัติการหนึ่งไปสู่ระบบปฏิบัติการหนึ่ง

ลักษณะ "Write One Run Anywhere" เกิดขึ้นได้เพราะการประมวลผลของคำสั่งที่อยู่ในไฟล์นามสกุล class หรือที่เรียกว่า ไฟล์คลาสถูกสั่งให้ประมวลผลบนจาวาเวอร์ชวลแมชชีน (Java Virtual Machine : JVM)

2.3.2 การเลือกเครื่องมือในการเขียนโปรแกรม

เมื่อเลือกภาษาที่จะใช้ในการเขียนโปรแกรมเชิงวัตถุแล้ว ต่อไปจะต้องหาคอมไพล์เลอร์ (Compiler) หรือ ตัวแปรภาษา คือ โปรแกรมที่ทำหน้าที่แปลงภาษาจาวาให้เป็นไฟล์คลาส ที่สามารถนำไปประมวลผลบนเวอร์ชวลแมชชีนได้

ในการเขียนโปรแกรมภาษาจาวา อย่างน้อยต้องมีคอมไพเลอร์และอินเตอร์พริเตอร์ของภาษาจาวา ซึ่งปัจจุบันมีหลายบริษัทด้วยกันที่ผลิตชุดโปรแกรมสำหรับพัฒนาโปรแกรมภาษาจาวาออกมา เช่น

- Java Development Kit (JDK) ของบริษัท Javasoft
- Visual J++ ของบริษัท Microsoft
- Visual Cafe ของบริษัท Symantec
- J Builder ของบริษัท Borland
- Visual Age For Java ของบริษัท IBM

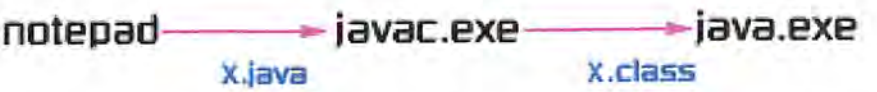
โดยหนึ่งในเครื่องมือที่มีผู้นิยมใช้ก็คือ Java Development Kit หรือที่ปัจจุบันถูกเรียกใหม่ว่า Software Development Kit (SDK) ซึ่งถึงแม้ว่าชุดโปรแกรมของบริษัทอื่นที่ถูสร้างเป็น Intergrated Development Environment (IDE) ที่ใช้งานได้ง่ายกว่า JDK ที่ยังคงเป็นแบบ command line แต่เนื่องจากบริษัท Javasoft เป็นผู้พัฒนาภาษา Java ดังนั้นคอมไพเลอร์ที่มีมากับ JDK จึงเป็นมาตรฐานของภาษา อีกทั้งภาษา Java ถูกพัฒนาอย่างรวดเร็วมีสิ่งใหม่ๆ ออกมาอย่างต่อเนื่องบริษัทอื่นๆ พัฒนาตามหลังช้ากว่าคอมไพเลอร์ใน IDE เหล่านั้นจึงไม่ทันสมัยเท่ากับคอมไพเลอร์ที่มากับ JDK

2.3.3 ขั้นตอนการสร้าง Java Application

Java Application คือโปรแกรมที่ถูกสร้างขึ้นเพื่อทำงานโดยถูก java.exe ทำการ interpret โดยถือว่าเป็นโปรแกรมที่สมบูรณ์ เนื่องจากไม่จำเป็นต้องมีโปรแกรมอื่นช่วยทำงาน สามารถควบคุมการดำเนินไปของตัวเองได้ โดยมี main() เป็นจุดเริ่มต้นของโปรแกรม บางคนเรียกโปรแกรมแบบนี้ว่า Stand Alone Program ซึ่งมีวิธีสร้างดังนี้

ใช้ Editor อะไรก็ได้เขียนโปรแกรมสำหรับ java application และต้องเก็บไว้ในไฟล์ที่มีนามสกุลเป็น .java เช่น x.java แล้วใช้คอมไพเลอร์ javac.exe ทำการคอมไพล์ x.java จะได้ x.class ซึ่งเป็น

โปรแกรม java class (Byte Code) เป็นผลลัพธ์ เมื่อจะให้ x.class ทำงานก็ต้องใช้ java.exe ดังแสดงในรูปที่ 2-36



รูปที่ 2-36 ขั้นตอนการใช้คอมไพเลอร์ JDK

- การใช้งาน javac.exe มีข้อสังเกตดังนี้
  - โปรแกรมภาษา java จะต้องเก็บไฟล์ที่มีนามสกุลเป็น .java เมื่อถูกคอมไพล์แล้วจะได้ไฟล์ที่มีนามสกุลเป็น .class และมีชื่อเหมือนกับคลาสที่กำหนดในโปร แกรมนั้นๆ โดยจะได้หนึ่งไฟล์ .class ต่อหนึ่งคลาสที่มีอยู่ในโปรแกรมนั้น
  - เราสามารถคอมไพล์โปรแกรมภาษา Java หลายๆ ไฟล์ได้ในการเรียก javac.exe ครั้งหนึ่ง โดยใช้ wildcard * เช่น หากมีไฟล์ .java หลายๆไฟล์ในไดเรกทอรีที่กำลังทำงานอยู่ก็สามารถคอมไพล์ไฟล์ .java ทั้งหมดโดยใช้คำสั่ง "javac *.java"

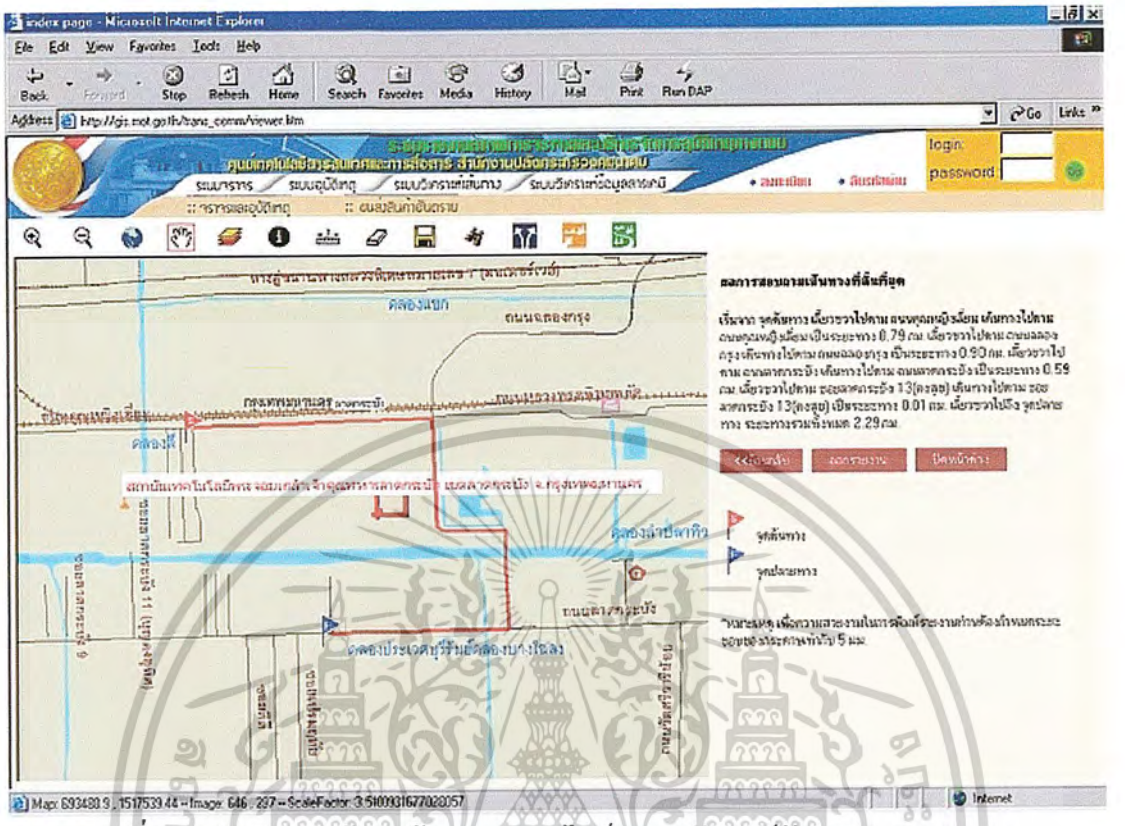
- การใช้งาน java.exe มีข้อสังเกตดังนี้
  - โปรแกรม .class หนึ่งไฟล์ จะมีเพียงหนึ่งคลาสเท่านั้น ดังนั้นโดยทั่วไป java.exe จะเริ่มต้นทำงานจากคลาสๆหนึ่งที่มี main() และเมื่อโปรแกรมมีการอ้างถึงคลาสอื่น คลาสนั้นก็จะไม่ถูกโหลดเข้ามาในขณะที่โปรแกรมกำลังทำงาน
  - โดยปรกติ java.exe จะถูกเรียกจาก command line หรือจาก batch file ซึ่งอาจมี arguments ส่งให้โปรแกรมไปเป็น array ของ String ที่ชื่อ args[]
  - ใน java.exe จะมีโปรแกรม Byte Code Verifier ทำการตรวจสอบคลาสใดๆ ที่ถูกโหลดเข้ามาทำงาน หากพบว่ามีการทำงานที่ไม่สมควร ก็จะปฏิเสธที่จะทำงานคลาสนั้น หากคลาสนั้นอยู่ในเครื่องเดียวกันกับ interpreter ก็ถือเป็นคลาสที่ไว้ใจได้และ Byte Code Verifier จะยกเว้นการตรวจสอบเพื่อความเร็วในการทำงาน แต่หากเป็นคลาสที่ถูกโหลดมาจากที่อื่นถือว่าเป็นคลาสที่ไว้ใจไม่ได้ และต้องทำการตรวจสอบ

2.4 ตัวอย่างระบบคล้ายคลึงที่มีอยู่ในปัจจุบัน

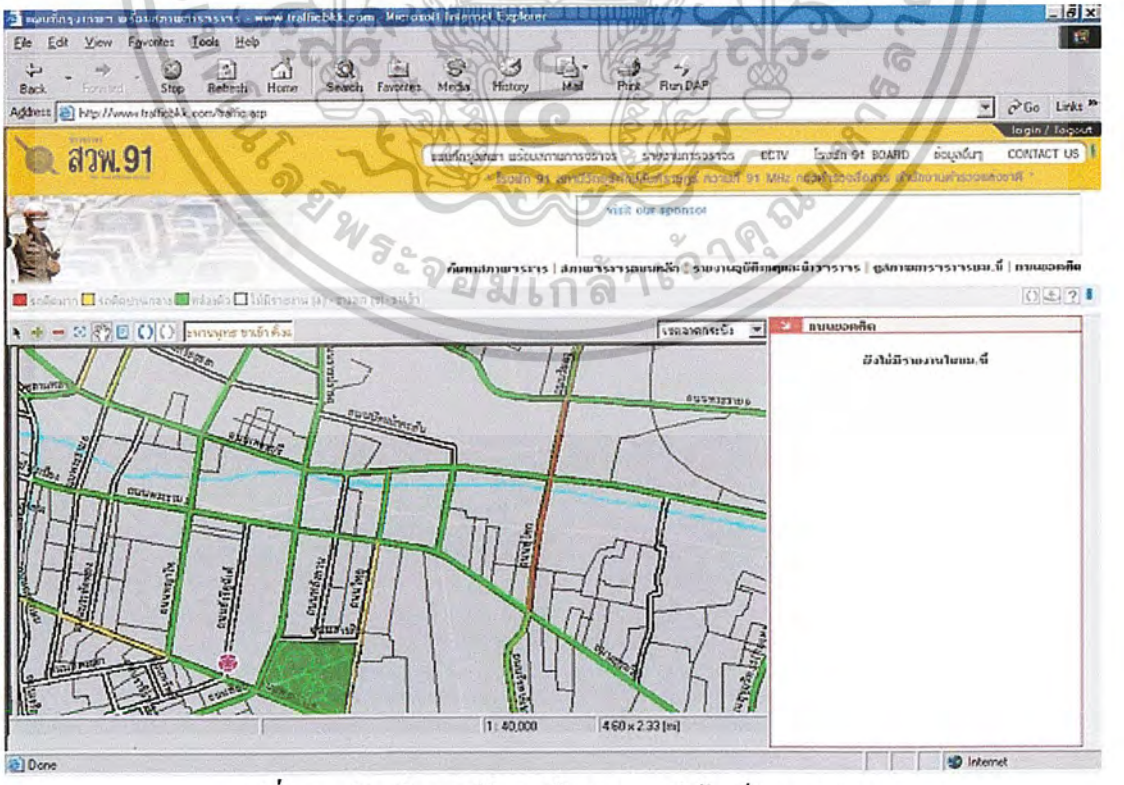
ในปัจจุบันมีเว็บไซต์หลายแห่งที่เสนอบริการในลักษณะคล้ายคลึงกับโครงงานนี้ แต่เว็บไซต์เหล่านั้น ตอบสนองความต้องการของผู้ใช้ได้เพียงบางส่วนเท่านั้น อาทิเช่น เสนอรายงานสภาพการจราจรของถนนสายต่างๆ แต่ไม่มีการค้นหาเส้นทางจากสถานที่หนึ่ง ไปยังอีกสถานที่หนึ่ง, มีการค้นหาเส้นทางจากสถานที่หนึ่ง ไปยังอีกที่หนึ่ง แต่ไม่ได้คำนึงถึงเงื่อนไขสภาพการจราจรในขณะนั้น เป็นต้น

รูปที่ 2-37 เป็นตัวอย่างในการค้นหาเส้นทางจากเว็บไซต์ของสำนักงานปลัดกระทรวงคมนาคม (http://gis.mot.go.th/trans_comm) ซึ่งผู้ใช้สามารถค้นหาเส้นทางจากสถานที่หนึ่ง ไปยังอีกสถานที่หนึ่งได้

แต่ไม่ได้คำนึงถึงเงื่อนไขสภาพการจราจรในขณะนั้น



รูปที่ 2-37 ตัวอย่างการค้นหาเส้นทางจากเว็บไซต์ของสำนักงานปลัดกระทรวงคมนาคม



รูปที่ 2-38 ตัวอย่างการค้นหาเส้นทางจากเว็บไซต์ของ สวพ.91

ในขณะที่รูปที่ 2-38 แสดงการรายงานสภาพการจราจรของถนนสายต่างๆจากเว็บไซต์ของ สวพ.91 (http://www.trafficbkk.com) ซึ่งผู้ใช้สามารถตรวจสอบสภาพการจราจรของถนนสายต่างๆ ในช่วงเวลานั้นๆได้ แต่ไม่ได้รวมถึงการค้นหาเส้นทางจากสถานที่หนึ่ง ไปยังอีกสถานที่หนึ่ง ดังนั้นถ้าผู้ใช้ระบบไม่ทราบว่าสถานที่ที่ต้องการไป ตั้งอยู่ที่เขตใด หรือถนนสายใด ระบบนี้ก็จะไม่อำนวยความสะดวกให้กับผู้ใช้รายนั้นแต่อย่างใด

บทที่ 3
ขั้นตอนและรายละเอียดของการพัฒนา

3.1 การออกแบบตารางฐานข้อมูล

- **ตารางข้อมูลลักษณะเฉพาะของแต่ละโหนด และความสัมพันธ์ที่เกี่ยวข้องกับโหนดข้างเคียง**

เนื่องจากอัลกอริทึมที่ใช้ในการค้นหาเส้นทาง เป็นการใช้การแตกโหนดในลักษณะโครงสร้างข้อมูลแบบต้นไม้ จากรูทโหนดต่อลงไปยังโหนดในระดับถัดไปเรื่อยๆ จนกว่าจะพบโหนดปลายทาง และได้เส้นทางอย่างน้อย 3 เส้นทาง ดังนั้นจึงจำเป็นต้องมีตารางฐานข้อมูลสำหรับเก็บรายละเอียดของโหนดข้างเคียงสำหรับทุกๆโหนดที่อยู่ภายในแผนที่ ตารางข้อมูลลักษณะเฉพาะของแต่ละ node และความสัมพันธ์ที่เกี่ยวข้องกับ node ข้างเคียง มีลักษณะตามตารางที่ 3-1

โหนดข้างเคียง	ตรง	ขวา	ซ้าย	กลับ รถ	ระยะทาง (เมตร)	ค่าผ่านด่านเก็บเงิน (บาท)	สภาพการจราจร
B	E	F	G	0	3500	40	3
C	H	0	I	0	4500	0	2
D	0	J	K	A	4000	0	0

ตารางที่ 3-1 ตารางแสดงข้อมูลของโหนด A และความสัมพันธ์กับโหนดข้างเคียง

จากตารางที่ 3-1 นำมาพิจารณารายละเอียดในแต่ละฟิลด์ข้อมูล ไล่จากซ้ายไปขวา ดังนี้

- โหนดข้างเคียง

คือโหนดทุกโหนดที่อยู่ติดกับโหนดของตารางนั้นๆ ดังตารางที่ 3-1 เนื่องจากเป็นตารางแสดงข้อมูลของโหนด A และความสัมพันธ์กับโหนดข้างเคียง จึงได้ว่า โหนดที่อยู่ข้างเคียงติดกับโหนด A ก็คือ โหนด B, C และ D

- ตรง

เป็นการบอกข้อมูลว่า เมื่อรถวิ่งจากโหนดของตารางนั้นๆ ไปจนถึงโหนดข้างเคียงโหนดใดๆของมันแล้ว จะสามารถวิ่งตรงต่อไปได้หรือไม่ และถ้าสามารถวิ่งตรงต่อไปได้ โหนดที่จะไปถึงถัดไปนั้นจะเป็นโหนดใด

- เมื่อมีค่าเป็น 0 หมายความว่า เมื่อรถวิ่งจากโหนดของตารางนั้นๆ ไปจนถึงโหนดข้างเคียงของมัน จะไม่สามารถวิ่งตรงต่อไปได้
- เมื่อมีค่าเป็น ชื่อของโหนดใดๆ หมายความว่า เมื่อรถวิ่งจากโหนดของตารางนั้นๆ ไปจนถึงโหนดข้างเคียงของมัน จะสามารถวิ่งตรงต่อไปได้จนถึงโหนดใดๆได้

พิจารณาตารางที่ 3-1 จะเห็นว่า ที่แถวของโหนดข้างเคียง B ในช่อง "ตรง" มีค่าเป็น E หมายความว่า เมื่อรถวิ่งจากโหนด A ไปถึงโหนด B จะสามารถวิ่งตรงต่อไปได้จนถึงโหนด E ในขณะเดียวกัน เมื่อพิจารณาที่ แถวของโหนดข้างเคียง D ซึ่งในช่อง "ตรง" มีค่าเป็น 0 หมายความว่า เมื่อรถวิ่งจากโหนด A ไปถึงโหนด D จะไม่สามารถวิ่งตรงต่อไปได้ เป็นต้น

- ขวา

เป็นการบอกข้อมูลว่า เมื่อรถวิ่งจากโหนดของตารางนั้นๆ ไปจนถึงโหนดข้างเคียงโหนดใดๆของมันแล้ว จะสามารถเลี้ยวขวาไปได้หรือไม่ และถ้าสามารถเลี้ยวขวาได้ โหนดที่จะไปถึงเป็นโหนดถัดไปนั้นจะเป็นโหนดใด

- เมื่อมีค่าเป็น 0 หมายความว่า เมื่อรถวิ่งจากโหนดของตารางนั้นๆ ไปจนถึงโหนดข้างเคียงของมัน จะไม่สามารถเลี้ยวขวาไปได้
- เมื่อมีค่าเป็นชื่อของโหนดใดๆ หมายความว่า เมื่อรถวิ่งจากโหนดของตารางนั้นๆ ไปจนถึงโหนดข้างเคียงของมัน จะสามารถเลี้ยวขวาต่อไปยังโหนดใดๆได้

พิจารณาตารางที่ 3-1 จะเห็นว่า ที่แถวของโหนดข้างเคียง B ในช่อง "ขวา" มีค่าเป็น F หมายความว่า เมื่อรถวิ่งจากโหนด A ไปถึงโหนด B จะสามารถเลี้ยวขวาไปยังโหนด F ได้ ในขณะเดียวกัน เมื่อพิจารณาที่ แถวของโหนดข้างเคียง C ซึ่งในช่อง "ขวา" มีค่าเป็น 0 หมายความว่า เมื่อรถวิ่งจากโหนด A ไปถึงโหนด C จะไม่สามารถเลี้ยวขวาได้ เป็นต้น

- ซ้าย

เป็นการบอกข้อมูลว่า เมื่อรถวิ่งจากโหนดของตารางนั้นๆ ไปจนถึงโหนดข้างเคียงโหนดใดๆของมันแล้ว จะสามารถเลี้ยวซ้ายไปได้หรือไม่ และถ้าสามารถเลี้ยวซ้ายได้ โหนดที่จะไปถึงเป็นโหนดถัดไปนั้นจะเป็นโหนดใด

- เมื่อมีค่าเป็น 0 หมายความว่า เมื่อรถวิ่งจากโหนดของตารางนั้นๆ ไปจนถึงโหนดข้างเคียงของมัน จะไม่สามารถเลี้ยวซ้ายไปได้
- เมื่อมีค่าเป็นชื่อของโหนดใดๆ หมายความว่า เมื่อรถวิ่งจากโหนดของตารางนั้นๆ ไปจนถึงโหนดข้างเคียงของมัน จะสามารถเลี้ยวซ้ายต่อไปยังโหนดใดๆได้

พิจารณาตารางที่ 3-1 จะเห็นว่า ที่แถวของโหนดข้างเคียง B ในช่อง "ซ้าย" มีค่าเป็น G หมายความว่า เมื่อรถวิ่งจากโหนด A ไปถึงโหนด B จะสามารถเลี้ยวซ้ายไปยังโหนด G ได้ เป็นต้น

- กลับรถ

เป็นการบอกข้อมูลว่า เมื่อรถวิ่งจากโหนดของตารางนั้นๆ ไปจนถึงโหนดข้างเคียงโหนดใดๆของมันแล้ว จะสามารถกลับรถได้หรือไม่ และถ้าสามารถกลับรถได้ โหนดที่จะไปถึงเป็นโหนดถัดไปนั้นจะเป็นโหนดใด

- เมื่อมีค่าเป็น 0 หมายความว่า เมื่อรถวิ่งจากโหนดของตารางนั้นๆ ไปจนถึงโหนดข้างเคียงของมัน จะไม่สามารถกลับรถได้
- เมื่อมีค่าเป็นชื่อของโหนดใดๆ หมายความว่า เมื่อรถวิ่งจากโหนดของตารางนั้นๆ ไปจนถึงโหนดข้างเคียงของมัน จะสามารถกลับรถได้กลับไปยังโหนดใดๆได้

พิจารณาตารางที่ 3-1 จะเห็นว่า ที่แถวของโหนดข้างเคียง B ในช่อง "กลับรถ" มีค่าเป็น 0 หมายความว่า เมื่อรถวิ่งจากโหนด A ไปถึงโหนด B จะไม่สามารถกลับรถกลับมายังโหนด A ได้ ในขณะเดียวกัน เมื่อพิจารณาที่ แถวของโหนดข้างเคียง D ซึ่งในช่อง "กลับรถ" มีค่า A หมายความว่า เมื่อรถวิ่งจากโหนด A ไปถึงโหนด D จะสามารถกลับรถที่โหนด D กลับมายังโหนด A ได้ เป็นต้น

- ระยะทาง

เป็นค่าระยะทางของถนนนับจากโหนดของตารางนั้นๆ ไปจนถึงโหนดข้างเคียงของมัน โดยมีหน่วยเป็นเมตร พิจารณาตารางที่ 3-1 จะเห็นว่า ที่แถวของโหนดข้างเคียง B ในช่อง "ระยะทาง" มีค่า 3500 หมายความว่า ระยะทางจากโหนด A ไปถึงโหนด B มีค่า 3500 เมตร

- ค่าผ่านด่านเก็บเงิน

เป็นค่าจำนวนเงินที่ต้องเสียให้กับด่านเก็บเงิน ในกรณีที่เส้นทางระหว่างโหนดของตารางนั้นๆ ไปจนถึงโหนดข้างเคียงของมัน เป็นเส้นทางพิเศษที่มีการตั้งด่านเก็บเงิน โดยค่าที่เก็บในตารางจะมีหน่วยเป็นบาท

พิจารณาตารางที่ 3-1 จะเห็นว่า ที่แถวของโหนดข้างเคียง B ในช่อง "ค่าผ่านด่านเก็บเงิน" มีค่า 40 หมายความว่า เส้นทางระหว่างโหนด A ไปจนถึงโหนด B จะต้องเสียค่าผ่านด่านเก็บเงินจำนวน 40 บาท

- สภาพการจราจร

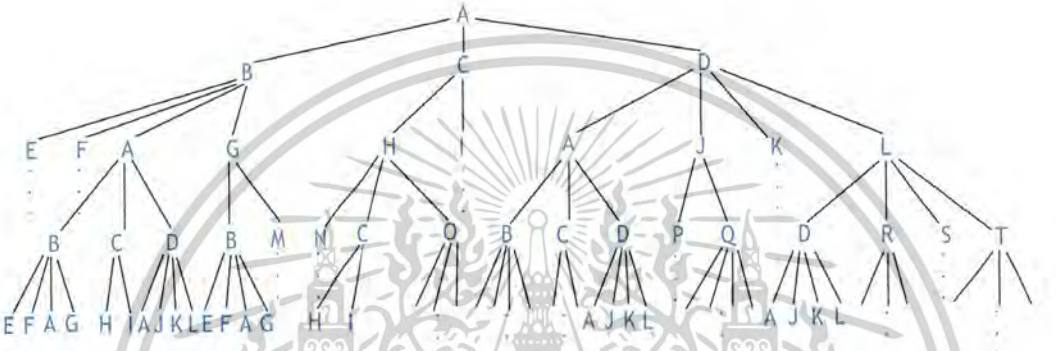
เป็นค่าตัวเลขที่แสดงถึงสภาพการจราจรของเส้นทางจากโหนดของตารางนั้นๆ ไปจนถึงโหนดข้างเคียงของมัน ซึ่งข้อมูลในช่องนี้ จะเป็นข้อมูลที่มีลักษณะแบบไดนามิก นั่นคือจะมีการเปลี่ยนแปลงค่าทุกๆ 5 นาที โดยค่าที่นำมาอัพเดททุกๆ 5 นาทีนี้ จะเป็นค่าที่ประเมินมาจากภาพที่ได้จากกล้องวิดีโอที่อยู่ตามแยกโหนดทั้งหลาย หรือข้อมูลการจราจรตามแหล่งต่างๆ

- เมื่อมีค่าเป็น 0 หมายความว่า สภาพการจราจรของเส้นทางระหว่างโหนดของตารางนั้นๆ ไปจนถึงโหนดข้างเคียงของมัน มีสภาพติดขัดมาก โดยกำหนดให้ค่าอัตราเร็วเฉลี่ยที่รถสามารถวิ่งได้คือ 83 เมตร/นาที
- เมื่อมีค่าเป็น 1 หมายความว่า สภาพการจราจรของเส้นทางระหว่างโหนดของตารางนั้นๆ ไปจนถึงโหนดข้างเคียงของมัน มีสภาพติดขัด โดยกำหนดให้ค่าอัตราเร็วเฉลี่ยที่รถสามารถวิ่งได้คือ 250 เมตร/นาที
- เมื่อมีค่าเป็น 2 หมายความว่า สภาพการจราจรของเส้นทางระหว่างโหนดของตารางนั้นๆ ไปจนถึงโหนดข้างเคียงของมัน มีสภาพคล่องตัว โดยกำหนดให้ค่าอัตราเร็วเฉลี่ยที่รถสามารถวิ่งได้คือ 500 เมตร/นาที
- เมื่อมีค่าเป็น 3 หมายความว่า สภาพการจราจรของเส้นทางระหว่างโหนดของตารางนั้นๆ ไปจนถึงโหนดข้างเคียงของมัน มีสภาพคล่องตัวมาก โดยกำหนดให้ค่าอัตราเร็วเฉลี่ยที่รถสามารถวิ่งได้คือ 1000 เมตร/นาที

3.2 การนำอัลกอริทึมมาใช้ในการค้นหาเส้นทาง

3.2.1 การนำอัลกอริทึมโครงสร้างข้อมูลแบบต้นไม้มาใช้ในการค้นหาเส้นทาง

เนื่องจากข้อมูลในตารางฐานข้อมูลลักษณะเฉพาะของแต่ละโหนด และความสัมพันธ์ที่เกี่ยวข้องกับโหนดข้างเคียงที่อยู่ติดกันนั้น เป็นการเก็บข้อมูลเฉพาะโหนดข้างเคียงที่อยู่ติดกันเท่านั้น ไม่ได้เก็บข้อมูลของเส้นทางที่เชื่อมไปยังโหนดที่อยู่ไกลออกไปมากกว่านั้น ดังนั้นจึงใช้หลักการของแตกโหนดในลักษณะโครงสร้างข้อมูลแบบต้นไม้มาใช้ในการค้นหาเส้นทาง โดยแตกโหนดแต่ละโหนดออกไปยังโหนดข้างเคียงเรื่อยๆ จนกว่าจะได้เส้นทางที่สมบูรณ์จากโหนดต้นทาง ไปจนถึงโหนดปลายทางอย่างน้อย 3 เส้นทาง



รูปที่ 3-1 แสดงการแตกโหนดในลักษณะโครงสร้างข้อมูลแบบต้นไม้ เพื่อค้นหาเส้นทาง

จากรูปที่ 3-1 จะเห็นเป็นการแสดงการแตกโหนดในลักษณะโครงสร้างข้อมูลแบบต้นไม้ ในการค้นหาเส้นทางจากโหนดต้นทาง A ไปยังโหนดปลายทางใดๆ (หมายเหตุ: ในรูปที่ 3-1 เครื่องหมายจุดสามจุดข้างใต้โหนด หมายถึงโหนดเหล่านี้ยังสามารถที่จะถูก span ออกไปยังระดับถัดไปได้อีก แต่ทำการละไว้)

อย่างไรก็ตาม จากแผนภาพการแตกโหนดในลักษณะโครงสร้างข้อมูลแบบต้นไม้ในรูปที่ 3-1 จะเห็นได้ว่า การแตกโหนดในหลายๆเส้นทาง เกิดการวนลูปวนกลับไปมา เช่น เส้นทาง A - B - A - B - ... - A - B หรือเส้นทาง A - B - G - B - A เป็นต้น ซึ่งเส้นทางเหล่านี้ ส่งผลทำให้กระบวนการแตกโหนดมีขนาดใหญ่ , ใช้ขนาดพื้นที่ของหน่วยความจำมาก และใช้เวลาในการแตกโหนดในแต่ละระดับ (Level) ที่มากเกินความจำเป็น ดังนั้นเราจึงต้องมีอัลกอริทึมที่ใช้ในการกำจัดเส้นทางในกรณีจำพวกนี้ด้วย

3.2.2 การตรวจสอบ และกำจัดเส้นทางที่เกิดการวนลูป

ลักษณะของลูปที่เกิดขึ้น และวิธีที่ใช้ในการกำจัดลูปเหล่านั้น มีดังนี้

3.2.2.1 ลูปในลักษณะ A – B – A – B

เมื่อแตกโหนดมาจนถึงระดับที่ 4 ให้ทำการตรวจสอบเส้นทางแต่ละเส้นทาง นับตั้งแต่รูทโหนดไปจนถึงโหนดที่อยู่ในระดับที่ 4 ว่ามีลักษณะดังนี้หรือไม่

1 2 3 4

A → B → A → B

เมื่อเกิดเหตุการณ์ในลักษณะนี้ขึ้น ก็ให้ยกเลิกการแตกโหนดสุดท้ายของเส้นทางนี้ และหลังจากนั้น ในการแตกโหนดของเส้นทางใดๆ ต่อลงไปในระดับถัดๆไป ก็ให้ใช้วิธีนี้ในการกำจัดเส้นทางที่เกิดการวนลูปในลักษณะนี้ต่อไปเรื่อยๆ โดยคิดย้อนกลับไป 4 โหนดล่าสุดของเส้นทางนั้นๆ เช่น

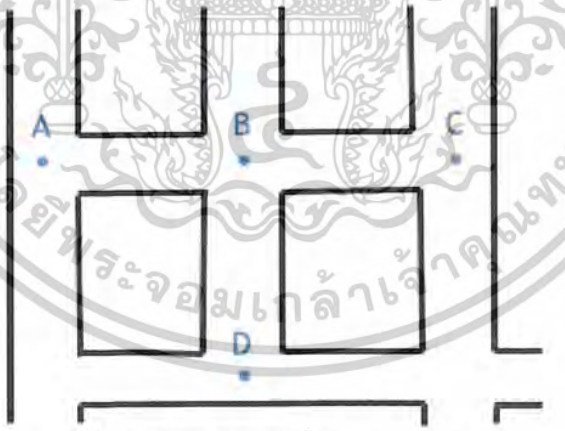
1 2 3 4 5 6 7

X → Y → Z → A → B → A → B

ในกรณีนี้ก็คือเหตุการณ์ที่แตกโหนดไปจนถึงระดับที่ 7 และเมื่อใช้หลักการการกำจัดลูปประเภทนี้ คิดย้อนขึ้นไป 4 โหนดหลังสุด (นั่นก็คือโหนดในระดับที่ 4 - 5 - 6 - 7) ก็จะพบ loop ในกรณีดังกล่าว ดังนั้นเราก็จะหยุดการแตกโหนด B ในระดับที่ 7 ต่อไปทันที

3.2.2.2 ลูปในลักษณะ A – B – C – B – D

พิจารณาลักษณะโหนดของเส้นทางในรูปที่ 3-2



รูปที่ 3-2 โหนดในแผนที่เส้นทางตัวอย่าง 1

ในการแตกโหนดตั้งแต่ระดับที่ 4 เป็นต้นไป เมื่อเกิดเส้นทางในลักษณะนี้ขึ้น ต้องนำมาพิจารณาต่อไปว่า รถสามารถวิ่งจากโหนด A → B → D ได้หรือไม่

- ถ้าสามารถวิ่งได้ ให้ทำการกำจัดเส้นทาง A → B → C → B → D ทิ้งไป ไม่ต้องทำการแตกโหนดต่อ ทั้งนี้เนื่องจากรถสามารถวิ่งในเส้นทาง A → B → D ได้แล้ว ดังนั้นเส้นทาง A → B → C → B → D จึงถือเป็นเส้นทางที่เกิดลูป

- ถ้าไม่สามารถวิ่งได้ ให้เก็บเส้นทาง A → B → C → B → D ไว้ทำการแตกโหนดต่อไปยังระดับถัดไป ทั้งนี้เนื่องจากรถไม่สามารถวิ่งในเส้นทาง A → B → D ได้ ซึ่งถ้าพิจารณาตามรูปที่ 3-2 ก็แสดงว่า รถที่มาจากโหนด A เมื่อวิ่งมาถึงโหนด B แล้วจะไม่สามารถเลี้ยวขวาไปยังโหนด D ได้ จึงต้องเลยไปกลับรถที่โหนด C เพื่อไปยังโหนด D นั่นเอง ดังนั้นเส้นทาง A → B → C → B → D จึงไม่ถือว่าเป็นเส้นทางที่เกิดลูป

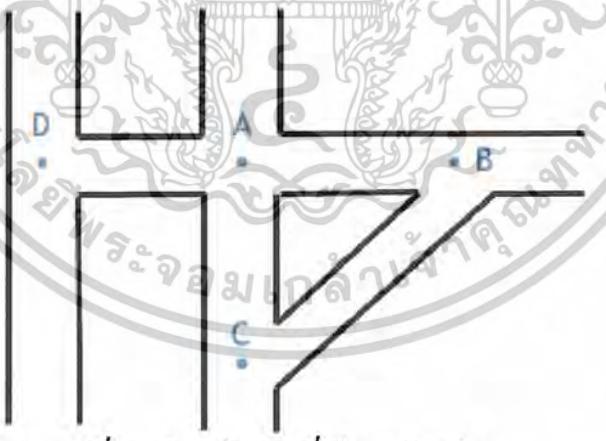
3.2.2.3 ลูปในลักษณะ A – B – C – B – A

ใช้รูปที่ 3-2 ในการพิจารณาเส้นทาง ในการแตกโหนดตั้งแต่ระดับที่ 5 เป็นต้นไป เมื่อเส้นทางใดมีลักษณะ A → B → C → B → A ต้องนำมาพิจารณาต่อว่า รถสามารถวิ่งจากโหนด A → B → A ได้หรือไม่

- ถ้าสามารถวิ่งได้ ให้ทำการกำจัดเส้นทาง A → B → C → B → A ทิ้งไปได้ ไม่ต้องทำการแตกโหนดในระดับล่างสุดต่อไป ทั้งนี้เนื่องจากรถสามารถวิ่งจากโหนด A ไปกลับรถที่โหนด B ได้ จึงไม่จำเป็นต้องวิ่งไปกลับรถที่โหนด C ดังนั้นเส้นทาง A → B → C → B → A จึงถือเป็นเส้นทางที่เกิดลูป
- ถ้าไม่สามารถวิ่งได้ ให้เก็บเส้นทาง A → B → C → B → A ไว้ทำการแตกโหนดต่อไป

3.2.2.4 ลูปในลักษณะ A – B – C – A – D

พิจารณาลักษณะโหนดของเส้นทางในรูปที่ 3-3



รูปที่ 3-3 โหนดในแผนที่เส้นทางตัวอย่าง 2

เมื่อเกิดเส้นทางในลักษณะ A → B → C → A → D ขึ้นในโหนด 5 ระดับแรก (A เป็นโหนดต้นทางในระดับที่ 1 และทำการแตกโหนดต่อมาจนถึงโหนด D ในระดับที่ 5) ให้ตัดเส้นทางนี้ทิ้งทันที ไม่ต้องนำมาพิจารณาค่าใดๆต่อ ทั้งนี้เนื่องจากเส้นทาง A → B → C → A → D ในกรณีนี้ เป็นเส้นทางที่เกิดการวนลูปแน่นอน เนื่องจากโหนด D เป็นโหนดข้างเคียงของโหนด A ดังนั้นรถจึงสามารถวิ่งจากโหนด A ไปยังโหนด D ได้เลยโดยไม่จำเป็นต้องวิ่งวนไปยังโหนด B และ C

อย่างไรก็ตาม ถ้าเส้นทางในลักษณะนี้ ไม่ได้เกิดขึ้นในโหนด 5 ระดับแรก เช่น

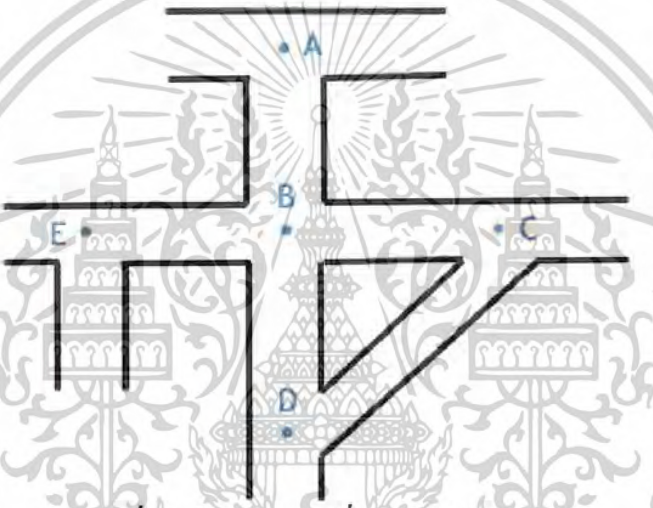
1 2 3 4 5 6

X → A → B → C → A → D

จะยังไม่สามารถตัดเส้นทางนี้ทิ้งไปได้ ต้องมีการพิจารณาต่อดังในข้อ 3.2.2.5

3.2.2.5 ลูปในลักษณะ A – B – C – D – B – E

พิจารณาลักษณะโหนดของเส้นทางในรูปที่ 3-4



รูปที่ 3-4 โหนดในแผนที่เส้นทางตัวอย่าง 3

ในการแตกโหนดตั้งแต่ระดับที่ 6 เป็นต้นไป เมื่อเกิดเส้นทางในลักษณะ A → B → C → D → B → E ขึ้น ให้นำมาพิจารณาต่อว่า รถสามารถวิ่งจากโหนด A → B → E ได้หรือไม่

- ถ้าสามารถวิ่งได้ ให้ตัดเส้นทาง A → B → C → D → B → E ทิ้งไป ทั้งนี้เนื่องจากรถสามารถวิ่งในเส้นทาง A → B → E ได้แล้ว จึงไม่จำเป็นต้องไปวิ่งอ้อมในเส้นทาง A → B → C → D → B → E อีก ดังนั้นจึงถือว่าเส้นทางนี้เป็นเส้นทางที่เกิดลูป
- ถ้าไม่สามารถวิ่งได้ ให้เก็บเส้นทาง A → B → C → D → B → E ไว้ทำการแตกโหนดต่อไป ทั้งนี้ไม่จำเป็นต้องพิจารณาอีกว่า รถสามารถวิ่งในเส้นทาง B → C → B ได้หรือไม่ เนื่องจากเส้นทาง B → C → B และเส้นทาง B → C → D → B ในกรณีนี้ ถือเป็นการวิ่งรถในเส้นทางที่แตกต่างกัน

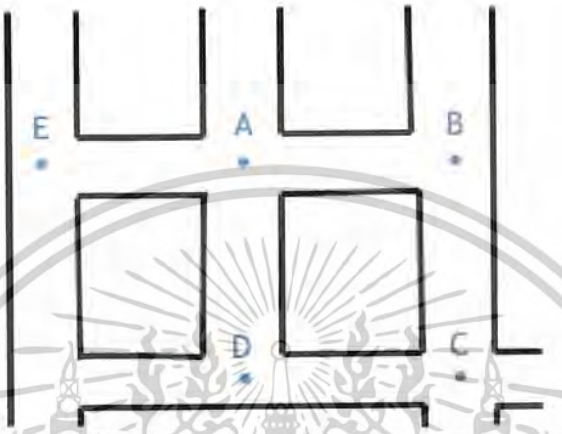
3.2.2.6 ลูปในลักษณะ A – B – A

พิจารณารูปที่ 3-4 ในการแตกโหนดเฉพาะใน 3 ระดับแรก ถ้าพบเส้นทางที่มีลักษณะ A → B →

A ให้หยุดทำการแตกโหนดนั้นต่อทันที เนื่องจากเป็นเส้นทางที่เกิดการวนลูป

3.2.2.7 ลูปในลักษณะ A – B – C – D – A – E

พิจารณารูปที่ 3-5 ในการแตกโหนดตั้งแต่ระดับที่ 6 เป็นต้นไป เมื่อเกิดเส้นทางในลักษณะ A → B → C → D → A → E ขึ้น ให้ตัดเส้นทางนี้ทิ้งทันที เนื่องจากเป็นเส้นทางที่เกิดการวนลูป



รูปที่ 3-5 โหนดในแผนที่เส้นทางตัวอย่าง 4

3.3 หลักการทำงานของโปรแกรมค้นหาเส้นทาง

โปรแกรมที่ใช้ เป็นโปรแกรมภาษาจาวา โดยการทำงานของโปรแกรมค้นหาเส้นทาง จะเริ่มจากการที่โปรแกรมรับค่าโหนดต้นทาง (สมมติให้มีค่าเป็น โหนด A) และโหนดปลายทาง (สมมติให้มีค่าเป็นโหนด Z) ที่ผู้ใช้ป้อนเข้ามาจากฝั่ง Front-end หลังจากนั้นจะนำโหนด A (โหนดต้นทาง) มาหาโหนดข้างเคียงของมันทั้งหมด เพื่อทำการแตกโหนดในครั้งแรก โดยจะติดต่อไปยังฐานข้อมูลเพื่อดึงเอาค่าโหนดข้างเคียงของโหนดA มาจากตารางแสดงข้อมูลของโหนด A และความสัมพันธ์กับโหนดข้างเคียง เมื่อได้โหนดข้างเคียงของโหนด A แล้ว โดยสมมติให้เป็นโหนด B, C และ D ก็จะนำโหนดข้างเคียงนี้มาต่อกับโหนด A เกิดเป็นเส้นทางจำนวน 3 เส้นทาง คือ A – B, A – C และ A – D

หลังจากนั้นก็จะทำการแตกโหนดต่อลงไปในระดับถัดไป โดยโปรแกรมจะนำเอาค่าโหนดในระดับล่างสุดของแต่ละเส้นทาง ซึ่งในที่นี้คือโหนด B, C และ D ไปหาค่าโหนดข้างเคียงของแต่ละโหนดจากตารางข้อมูลของแต่ละโหนด และความสัมพันธ์กับโหนดข้างเคียง เมื่อได้โหนดข้างเคียงของโหนด B ก็จะนำโหนดข้างเคียงเหล่านั้น ไปต่อเข้ากับเส้นทางที่มีโหนด B อยู่ในระดับล่างสุด เช่นเดียวกับโหนดข้างเคียงของโหนด C และ D

เมื่อทำการแตกโหนดไปจนถึงระดับที่ 3 ของเส้นทาง เช่น A – B – E เป็นต้น ก็ต้องเริ่มทำการตรวจสอบว่ามีเส้นทางใดบ้างที่เกิดการวนลูปในลักษณะตามหัวข้อ 3.2.2.6 (A – B – B) ถ้าพบ ก็จะหยุดแตกโหนดในเส้นทางนั้นต่อทันที

เมื่อแตกโหนดมาจนถึงระดับที่ 4 ของเส้นทาง ก็ต้องมีการตรวจสอบว่ามีเส้นทางใดบ้างที่เกิดการวนลูปในลักษณะตามหัวข้อ 3.2.2.1 (A – B – A – B) และเมื่อแตกโหนดต่อไปจนถึงในระดับที่ 5 ของ

เส้นทาง ก็ต้องมีการตรวจสองเส้นทางที่เกิดการวนลูปในลักษณะตามข้อ 3.2.2.2 (A - B - C - B – D), 3.2.2.3 (A - B - C - B – A) และ 3.2.2.4 (A - B - C - A – D) และยังคงต้องทำการตรวจสอบการวนลูปตามหัวข้อ 3.2.2.1 อีกเช่นกัน

และเมื่อแตกโหนดมาจนถึงระดับที่ 6 ของเส้นทาง ก็ต้องมีการตรวจสอบการวนลูปของเส้นทางในลักษณะตามหัวข้อ 3.2.2.5 (A - B - C - D - B – E) และ 3.2.2.7 (A - B - C - D - A – E) นอกจากนั้นก็ยังคงต้องทำการตรวงสอบการวนลูปตามลักษณะของหัวข้อ 3.2.2.1, 3.2.2.2 และ 3.2.2.3 อีกเช่นกัน

สำหรับเส้นทางที่ไม่เกิดการวนลูป ก็จะถูกทำการแตกโหนดในระดับล่างสุดต่อไปเรื่อยๆ กระบวนการแตกโหนดเพื่อค้นหาเส้นทางจะดำเนินต่อไปเรื่อยๆจนกว่าจะพบเส้นทางที่สมบูรณ์จากโหนดต้นทาง ไปจนถึงโหนดปลายทาง เป็นจำนวน 3 เส้นทาง เมื่อได้เส้นทางครบสามเส้นทางแล้ว ก็จะหยุดทำการแตกโหนดในทุกๆเส้นทางทันที

เมื่อได้เส้นทางจากโหนดต้นทางไปยังโหนดปลายทางมา 3 เส้นทาง ก็จะนำมาตรวจสอบเพื่อหาเส้นทางที่ดีที่สุดเพียงเส้นทางเดียว ทั้งนี้ เส้นทางที่ดีที่สุดจะขึ้นอยู่กับความต้องการของผู้ใช้ ว่าต้องการเส้นทางที่ดีที่สุดในด้านใด กล่าวคือ ผู้ใช้สามารถเลือกได้ว่าต้องการเส้นทางที่ดีที่สุดในด้าน

- การใช้เวลาน้อยที่สุดในการเดินทาง
- เส้นทางที่มีระยะทางสั้นที่สุด
- เส้นทางที่ไม่เสียค่าใช้จ่ายในการเดินทาง

อย่างไรก็ตาม ผู้ใช้สามารถเลือกเงื่อนไขในการค้นหาเส้นทางที่ดีที่สุดได้มากกว่าหนึ่งเงื่อนไข โดยในกรณีที่ผู้ใช้เลือกเงื่อนไขในการพิจารณามากกว่าหนึ่งเงื่อนไข โปรแกรมจะทำการตัดสินค่าลำดับความสำคัญของเงื่อนไข โดยให้เงื่อนไขด้านเงิน หรือค่าใช้จ่าย เป็นเงื่อนไขที่มีความสำคัญมากที่สุด ตามมาด้วยเงื่อนไขด้านเวลา และระยะทาง ตามลำดับ โดยมีลักษณะตัวอย่างดังนี้

- ผู้ใช้เลือกเงื่อนไขทั้งสามด้าน คือ ระยะเวลา, เงิน และระยะทาง: ในกรณีนี้ ระบบจะให้ความสำคัญกับเงื่อนไขด้านเงินมากที่สุด ทั้งนี้เนื่องจากการที่ผู้ใช้เลือกเงื่อนไขด้านเงินเข้ามาด้วยนั้น แสดงว่าผู้ใช้ต้องการเส้นทางที่เสียเงินจำนวนน้อยที่สุด หรือไม่เสียเงินเลย ดังนั้นจึงต้องนำเส้นทางที่ได้ทั้งสามเส้นทาง มาเปรียบเทียบด้านจำนวนเงินก่อน ถ้ามีเส้นทางที่ใช้เงินน้อยที่สุดเท่ากันมากกว่าหนึ่งเส้นทาง จะต้องนำมาเปรียบเทียบต่อในด้านเวลา และระยะทางตามลำดับ
- ผู้ใช้เลือกเงื่อนไขด้านเงิน และเวลา: เช่นเดียวกับลักษณะข้างต้นทุกประการ คือระบบจะให้ความสำคัญกับเงื่อนไขด้านเงินมากที่สุด ตามมาด้วยระยะเวลา และระยะทาง ตามลำดับ
- ผู้ใช้เลือกเงื่อนไขด้านเงิน และระยะทาง: ลักษณะจะคล้ายกับสองกรณีข้างต้น โดยเปรียบเทียบด้านจำนวนเงินที่ใช้ก่อน แต่ถ้ามีเส้นทางที่ใช้เงินน้อยที่สุดเท่ากันมากกว่าหนึ่งเส้นทาง จะนำมาเปรียบเทียบต่อในด้านระยะทาง และระยะเวลาที่ใช้ ตามลำดับ
- ผู้ใช้เลือกเงื่อนไขด้านระยะเวลา และระยะทาง: ถ้าผู้ใช้ระบบไม่เลือกเงื่อนไขด้านเงิน แสดงว่าเต็มใจที่จะใช้เส้นทางที่มีการเสียค่าผ่านทางต่างๆ ดังนั้นระบบจะนำเส้นทางที่ได้ทั้ง 3 เส้นทาง มาเปรียบเทียบกันในด้านระยะเวลาที่ใช้เป็นอันดับแรก ตามมาด้วยระยะทางที่สั้น

ที่สุด ถ้าการเปรียบเทียบโดยใช้เงื่อนไขทั้งสองยังมีเส้นทางมากกว่าหนึ่งเส้นทางที่เท่าเทียมกัน จะใช้เงื่อนไขด้านเงินมาตัดสินเป็นเงื่อนไขสุดท้าย

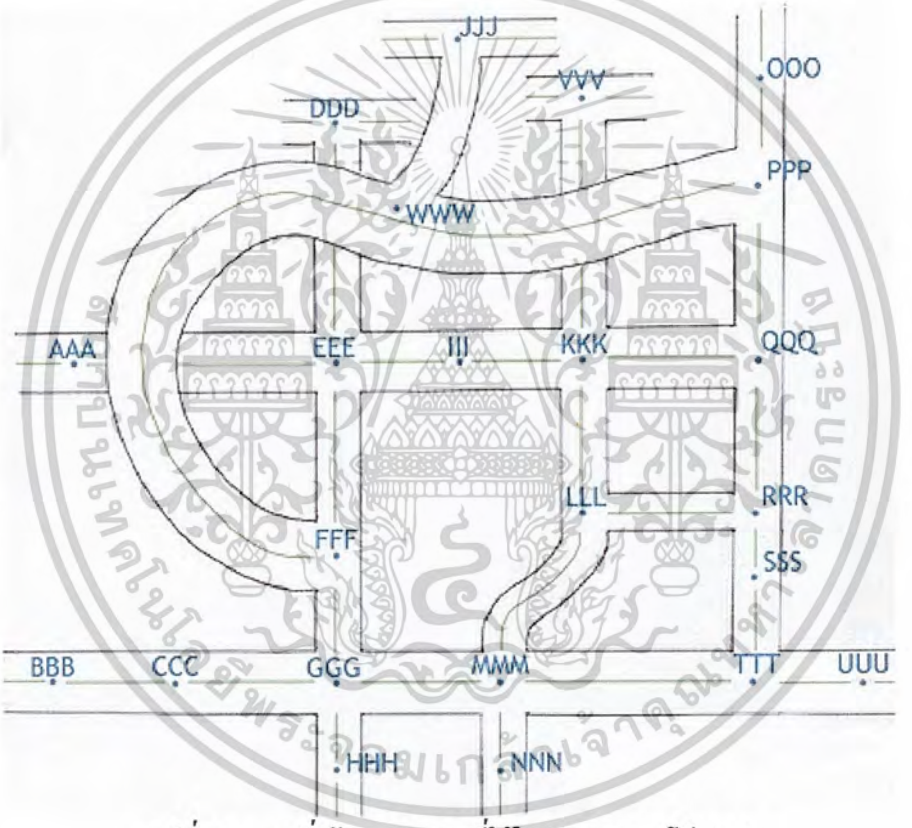
ในกรณีที่ผู้ใช้ไม่ทำการเลือกเงื่อนไขใดๆสำหรับการพิจารณาค้นหาเส้นทางที่ดีที่สุดเลยนั้น ระบบจะใช้การตัดสินใจในกรณีเดียวกันกับการที่ผู้ใช้เลือกเงื่อนไขด้านระยะเวลา และระยะทาง เท่านั้น ทั้งนี้เนื่องจากระบบจะทึกทักเอาว่า ผู้ใช้เต็มใจที่จะใช้เส้นทางที่มีการเสียค่าใช้จ่ายในการเดินทาง เนื่องจากไม่คลิกเลือกเงื่อนไขด้านเงิน หรือค่าใช้จ่ายมานั่นเอง

เมื่อเปรียบเทียบเส้นทางทั้ง 3 เส้นทาง และได้เส้นทางที่ดีที่สุดออกมาแล้วนั้น โปรแกรมจะส่งค่าสตริงของเส้นทางที่ดีที่สุดตามเงื่อนไขของผู้ใช้, ระยะเวลาที่ใช้ในเส้นทางนี้โดยประมาณ, ค่าใช้จ่ายที่ต้องใช้ และระยะทางรวมตั้งแต่โหนดต้นทาง ไปจนถึงโหนดปลายทาง กลับไปยังส่วน Front-end เพื่อทำการแสดงผลต่อไป

บทที่ 4
การทดสอบการใช้งานโปรแกรม

4.1 การทดสอบโปรแกรมโดยใช้แผนที่เส้นทางจำลอง

ในการทดสอบจะใช้ JDK ของ Sun ในการคอมไพล์โปรแกรมภาษาจาวา โดยมีการติดต่อกับฐานข้อมูลที่ใช้ MySQL ทั้งนี้ ในการทดสอบโปรแกรม ได้ใช้แผนที่เส้นทางจำลองในการทดลองค้นหาเส้นทาง ซึ่งแผนที่ดังกล่าว แสดงในรูปที่ 4-1

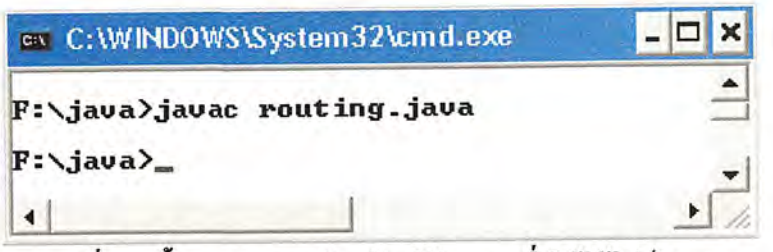


รูปที่ 4-1 แผนที่เส้นทางจำลองที่ใช้ในการทดสอบโปรแกรม

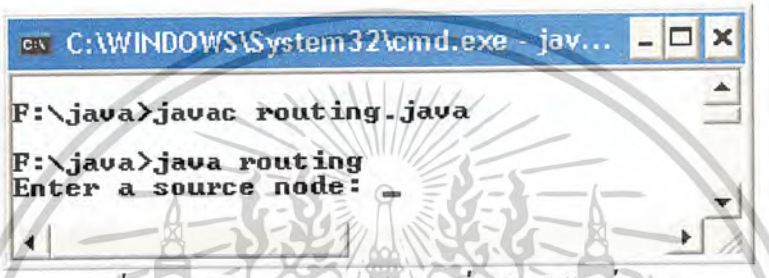
การที่ใช้ชื่อโหนดให้เป็นตัวอักษรมากกว่าหนึ่งตัวนั้น เนื่องมากจากชื่อโหนดทั้งหมดที่ใช้ในแผนที่จริงในกรุงเทพฯ จะเป็นสตริงที่มีความยาวมากกว่าหนึ่งตัวอักษร ดังนั้นในการทดสอบจึงกำหนดให้มีลักษณะเหมือนกัน

หลังจากดาวน์โหลด Sun's SDK 1.4.2_03, ทำการติดตั้ง, เซตค่า Path ของ SDK และทำการเชื่อมต่อกับฐานข้อมูล MySQL โดยใช้ MySQL Connector/J ซึ่งเป็น JDBC 3.0 API ของ Sun สำหรับฐานข้อมูล MySQL เรียบร้อยแล้ว ก็สามารถเริ่มทำการคอมไพล์โค้ดจาวา ได้ทันที โดยใช้คำสั่ง javac ใน

การสร้างไฟล์ class ของจาวา ก่อนที่จะใช้คำสั่ง java ในการ execute ไฟล์ class ที่ได้ ดังตัวอย่างในรูปที่ 4-2 และรูปที่ 4-3 โดยโค้ดจาวา ถูกเก็บเป็นไฟล์ชื่อว่า routing.java

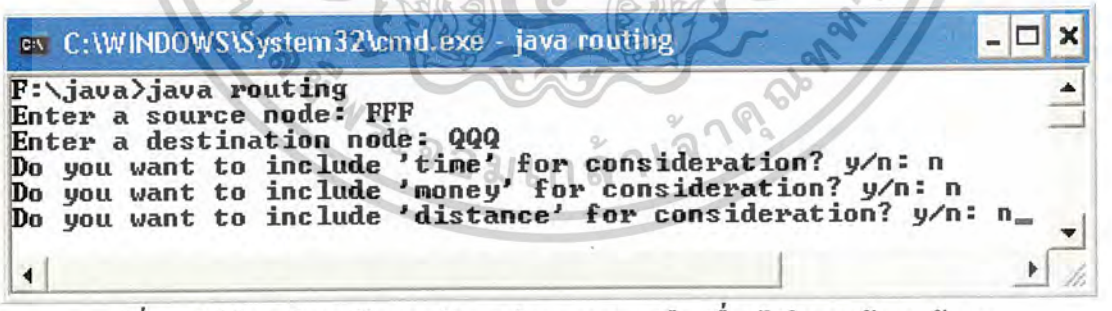


รูปที่ 4-2 ขั้นตอนการคอมไพล์โดยใช้ javac เพื่อให้ได้ไฟล์ class



รูปที่ 4-3 ทำการ execute ไฟล์ class ที่ได้ โดยใช้คำสั่ง java

หลังจากทำการ execute ไฟล์ routing.class โปรแกรมก็จะเริ่มทำงานโดยให้ใส่ค่าโหนดต้นทางเข้าไปเป็นค่าแรก หลังจากการใส่ค่าโหนดต้นทางแล้ว โปรแกรมจะให้ใส่ค่าโหนดปลายทาง รวมถึงการเลือกเงื่อนไขในการค้นหาเส้นทางที่ดีที่สุด ซึ่งได้แก่เงื่อนไขในด้าน ระเวลาที่ดีที่สุด, ระยะทางสั้นที่สุด และเสียค่าใช้จ่ายน้อยที่สุด ลักษณะดังรูปที่ 4-4

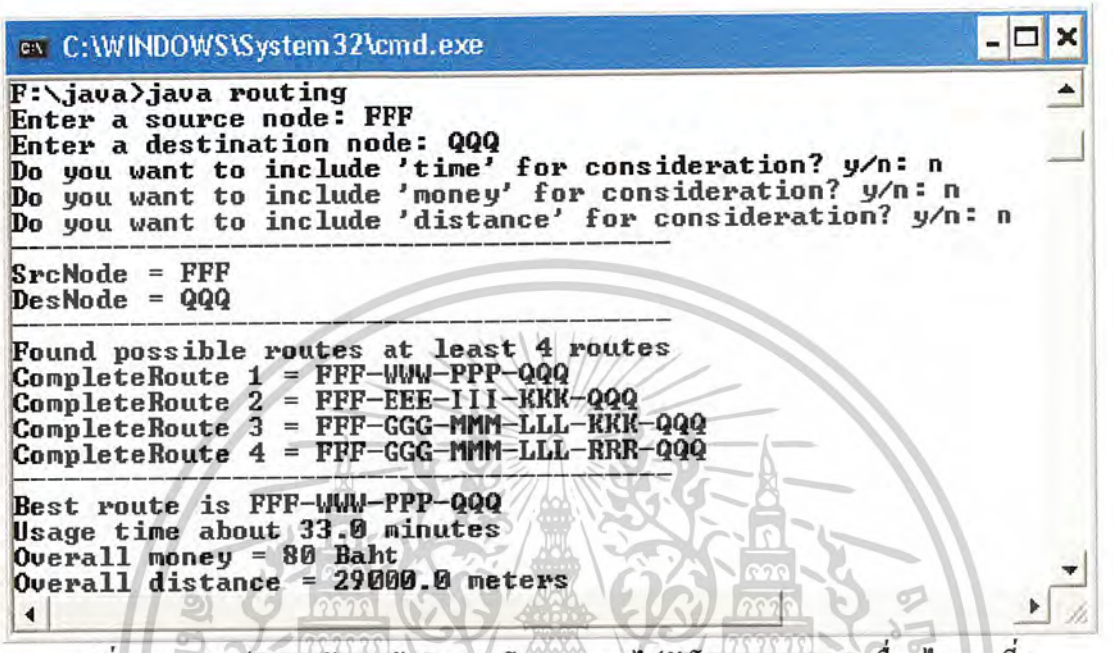


รูปที่ 4-4 ใส่ข้อมูลโหนดต้นทาง, โหนดปลายทาง และเลือกเงื่อนไขในการค้นหาเส้นทาง

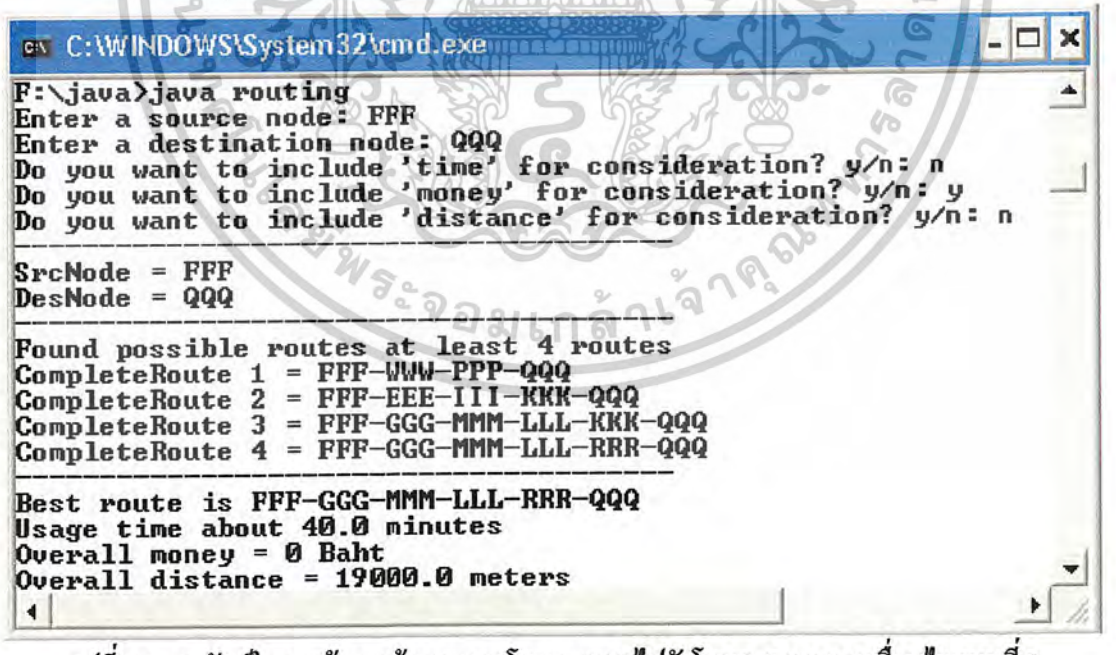
อย่างไรก็ตาม ในการทำงานจริงๆนั้น ค่าเหล่านี้จะเป็นค่าที่รับเป็นอินพุทมาจากฝั่ง Front-end ซึ่งจะส่งค่าในรูปแบบสตริงมาจากหน้าเพจ ASP แต่ในกรณีนี้เป็นการทดสอบโปรแกรม จึงเขียนให้ทำการรับค่าจากคีย์บอร์ด เพื่อให้ง่ายต่อการทดสอบ

เมื่อใส่โหนดต้นทาง, โหนดปลายทาง และเงื่อนไขที่ต้องการใช้ในการค้นหาเส้นทางแล้ว กด enter จะได้ผลลัพธ์ดังที่แสดงในรูปที่ 4-5

โดยในรูปที่ 4-5 เป็นผลลัพธ์ในการค้นหาเส้นทางที่ดีที่สุด จากโหนด FFF ไปยังโหนด QQQ ในกรณีที่ผู้ใช้ไม่ได้คลิกเลือกเงื่อนไขใดๆในการค้นหาเส้นทางเลย ซึ่งผลลัพธ์เส้นทางที่ดีที่สุดที่ได้ คือ เส้นทาง FFF → WWW → PPP → QQQ, ใช้เวลาโดยประมาณคือ 33 นาที, ต้องเสียเงินค่าผ่านทางในระหว่างเส้นทาง จำนวน 80 บาท และมีระยะทางรวมทั้งหมดตลอดเส้นทาง คือ 29000 เมตร หรือ 29 กิโลเมตร



รูปที่ 4-5 ผลลัพธ์ในการค้นหาเส้นทางจากโหนด FFF ไปยังโหนด QQQ ตามเงื่อนไขแบบที่ 1



รูปที่ 4-6 ผลลัพธ์ในการค้นหาเส้นทางจากโหนด FFF ไปยังโหนด QQQ ตามเงื่อนไขแบบที่ 2

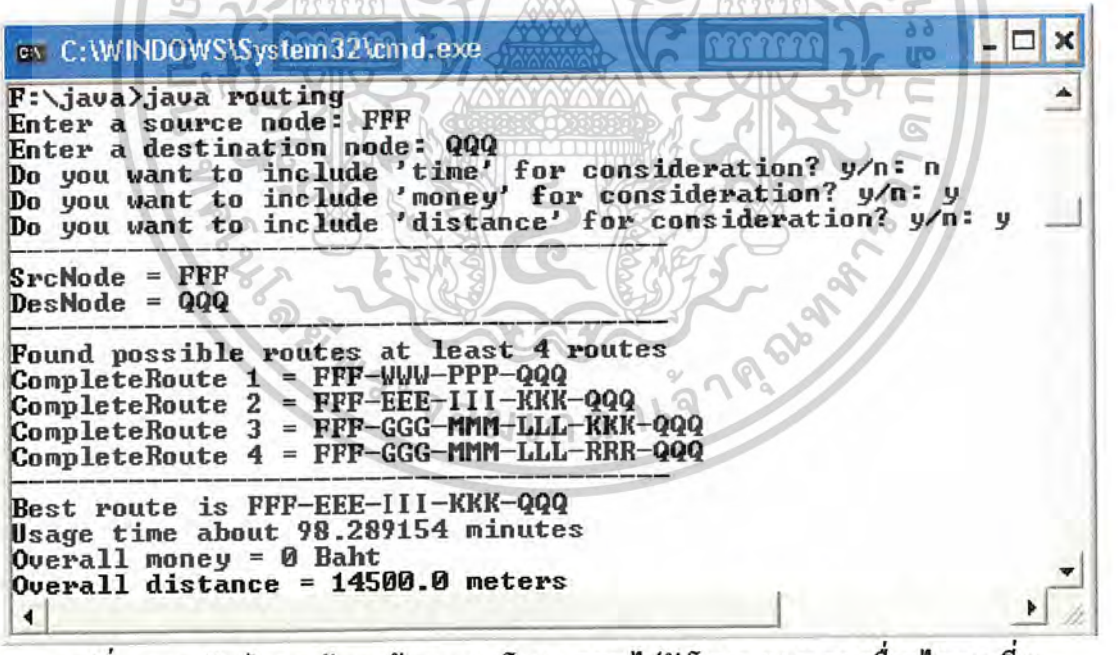
ในขณะที่รูปที่ 4-6 เป็นการค้นหาเส้นทางจากโหนดต้นทาง FFF ไปยังโหนดปลายทาง QQQ เช่นเดียวกับรูปที่ 4-5 แต่มีการเลือกเงื่อนไขหนึ่งอย่าง คือ เงิน หรือค่าใช้จ่ายในการเดินทาง ซึ่งหมายความ

ว่า ผู้ใช้มีการคลิกเลือกเงื่อนไขด้านเงิน ในการพิจารณาหาเส้นทางที่ดีที่สุด หรืออีกนัยหนึ่งก็คือ ผู้ใช้ต้องการเสียค่าใช้จ่ายในการใช้เส้นทางในการเดินทางนั่นเอง

ผลลัพธ์ที่ได้ จะแตกต่างจากผลลัพธ์ในรูปที่ 4-5 ที่ไม่ได้กำหนดเงื่อนไขใดๆ นั่นคือเส้นทางที่ดีที่สุดที่ได้ออกมาเป็นผลลัพธ์ คือเส้นทาง FFF → GGG → MMM → LLL → RRR → QQQ ซึ่งเป็นเส้นทางที่ไม่เสียค่าใช้จ่ายในการเดินทาง อย่างไรก็ตาม เมื่อเปรียบเทียบกับเส้นทางผลลัพธ์ที่ได้ในรูปที่ 4-5 แล้ว เส้นทางเส้นนี้แม้จะไม่เสียค่าใช้จ่าย แต่ก็ใช้เวลาในการเดินทางที่มากกว่า นั่นคือประมาณ 40 นาที

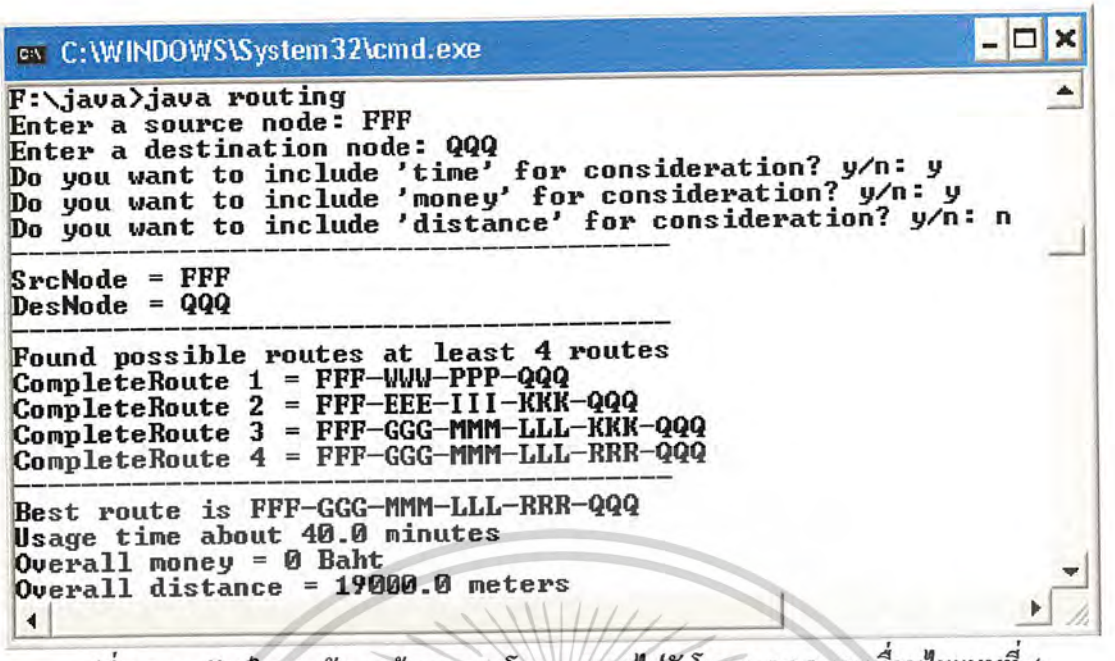
ส่วนรูปที่ 4-7 ยังคงเป็นการค้นหาเส้นทางจากโหนดต้นทาง FFF ไปยังโหนดปลายทาง QQQ โดยกำหนดเลือกเงื่อนไขสองเงื่อนไขในการพิจารณาเส้นทางที่ดีที่สุด นั่นคือ เงิน และ ระยะทาง โดยไม่สนใจว่าเส้นทางผลลัพธ์จะใช้ระยะเวลาเท่าใดในการเดินทาง และดังที่กล่าวไว้ในหัวข้อที่ 3.3 เมื่อมีการเลือกเงื่อนไขสองเงื่อนไข คือ เงิน และระยะทาง พร้อมกัน โปรแกรมจะให้ความสำคัญกับเงื่อนไขด้านเงินก่อนเป็นอย่างแรก ตามด้วยระยะทาง และระยะเวลาตามลำดับ

เส้นทางผลลัพธ์ที่ได้ดังแสดงในรูปที่ 4-7 คือ FFF → EEE → III → KKK → QQQ ซึ่งไม่เสียค่าใช้จ่ายในการเดินทาง และจะเห็นว่าเป็นเส้นทางที่มีระยะทางรวมเพียงแค่ 14500 เมตร หรือ 14.5 กิโลเมตรเท่านั้น น้อยกว่าเส้นทางผลลัพธ์ที่ได้ในรูปที่ 4-5 และ 4-6 อย่างไรก็ตาม เวลาที่ใช้โดยประมาณตลอดทั้งเส้นทาง มีค่าสูงถึง 98.29 นาที ซึ่งมากกว่าเส้นทางผลลัพธ์ในรูปที่ 4-5 และ 4-6 มาก ทั้งนี้เนื่องมากจากเส้นทางย่อยๆภายในเส้นทางนี้ มีสภาพการจราจรที่ติดขัด จึงใช้เวลาในการเดินทางนาน

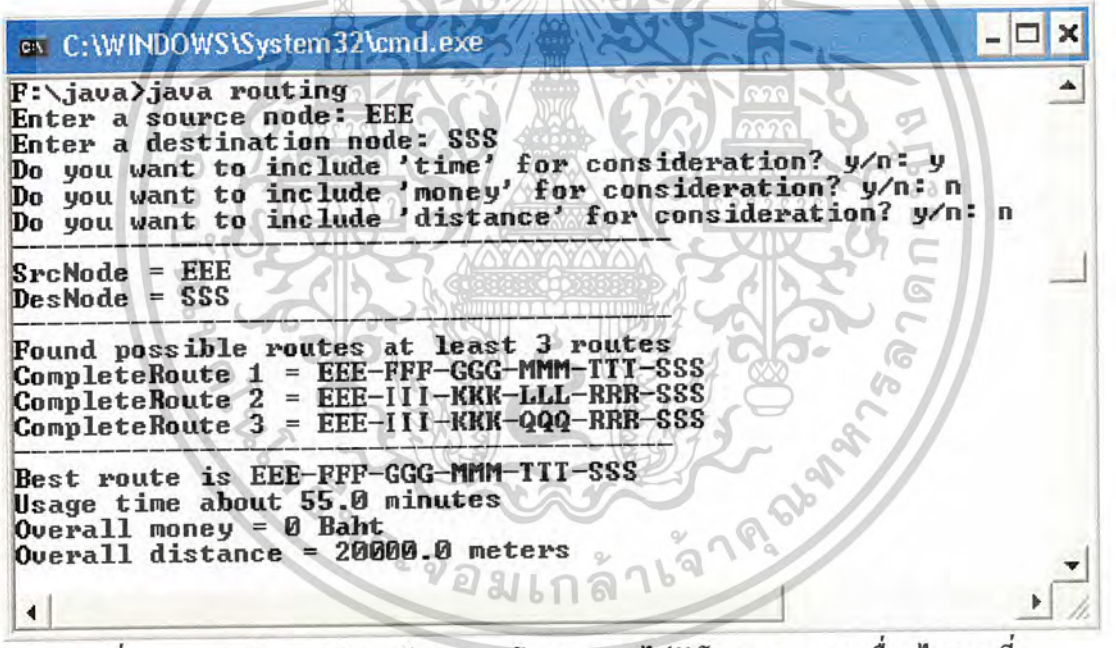


รูปที่ 4-7 ผลลัพธ์ในการค้นหาเส้นทางจากโหนด FFF ไปยังโหนด QQQ ตามเงื่อนไขแบบที่ 3

ในขณะที่รูปที่ 4-8 เป็นการค้นทางเส้นทางจากโหนดต้นทาง FFF ไปยังโหนดปลายทาง QQQ เช่นเดิม โดยเปลี่ยนเงื่อนไขมาเลือกสองเงื่อนไข ได้แก่ ระยะเวลา และเงิน ซึ่งตามรายละเอียดที่กล่าวในหัวข้อที่ 3.3 ในกรณีนี้ เราจะให้ความสำคัญกับเงินเป็นอันดับแรก ตามด้วยเวลา และระยะทาง ตามลำดับ



รูปที่ 4-8 ผลลัพธ์ในการค้นหาเส้นทางจากโหนด FFF ไปยังโหนด QQQ ตามเงื่อนไขแบบที่ 4

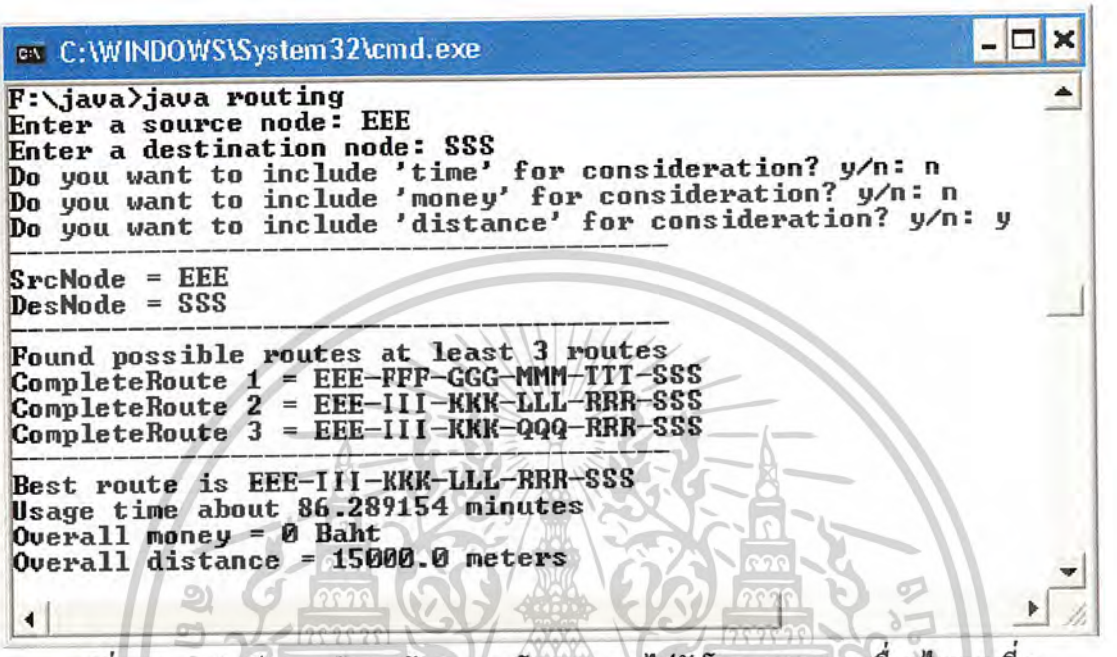


รูปที่ 4-9 ผลลัพธ์ในการค้นหาเส้นทางจากโหนด EEE ไปยังโหนด SSS ตามเงื่อนไขแบบที่ 1

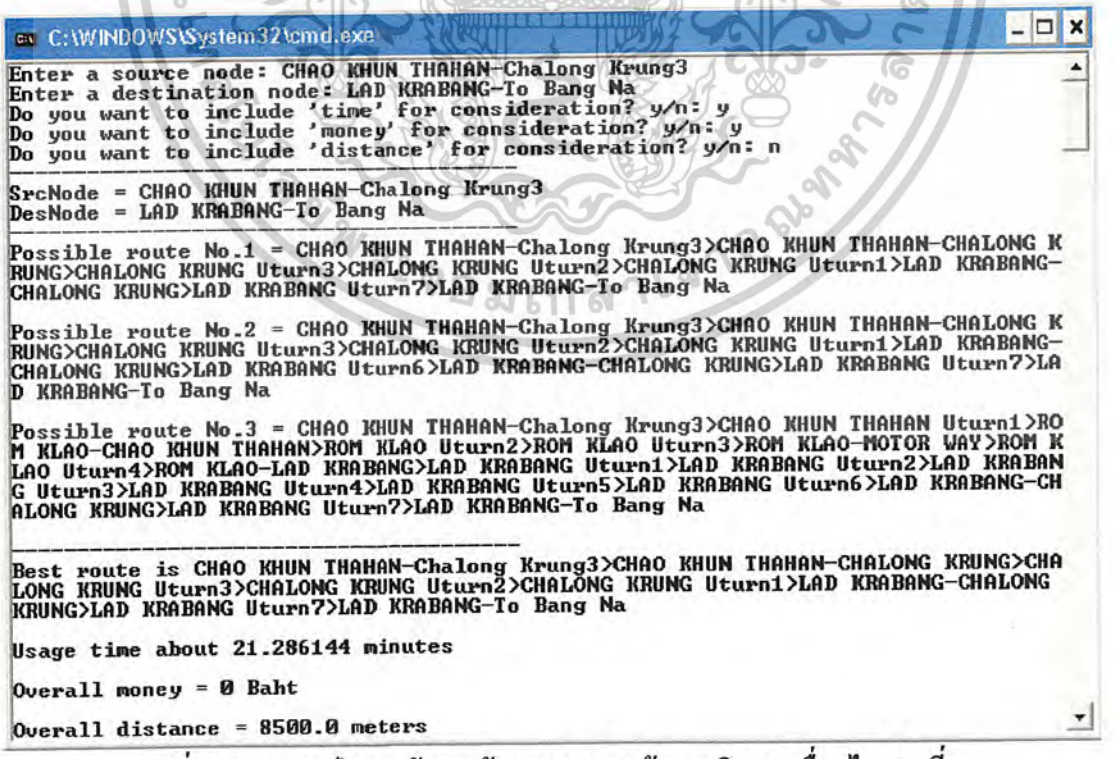
รูปที่ 4-9 เป็นการค้นหาเส้นทางจากโหนดต้นทาง EEE ไปยังโหนดปลายทาง SSS โดยเลือกเงื่อนไขหนึ่งเงื่อนไข นั่นคือเงื่อนไขในด้านระยะเวลาที่สั้นที่สุด ซึ่งจะได้เส้นทางผลลัพธ์ที่ดีที่สุดคือ EEE → FFF → GGG → MMM → TTT → SSS ใช้เวลาในการเดินทาง 55 นาที, ระยะทางรวม 20 กิโลเมตร และไม่เสียค่าใช้จ่ายใดๆในการเดินทาง

ในขณะที่รูปที่ 4-10 เป็นการค้นหาเส้นทางจากโหนด EEE ไปยังโหนด SSS เช่นกัน แต่เปลี่ยนไปเลือกเงื่อนไขในด้านระยะทางเพียงอย่างเดียว ซึ่งตามหลักการในข้อ 3.3 ในกรณีนี้ เราจะให้ความสำคัญในเงื่อนไขด้านระยะทางเป็นอันดับแรก ตามด้วยระยะเวลา และเงิน ตามลำดับ ซึ่งเส้นทาง

ผลลัพธ์ที่ดีที่สุดที่ได้ออกมา จะแตกต่างจากในรูปที่ 4-9 นั่นคือจะได้เส้นทาง EEE → III → KKK → LLL → RRR → SSS ซึ่งเป็นเส้นทางที่มีระยะทางเพียง 15 กิโลเมตร ซึ่งน้อยกว่าเส้นทางผลลัพธ์ในรูปที่ 4-9 อย่างไรก็ตาม ระยะเวลาโดยประมาณที่เส้นทางนี้ใช้ มีค่ามากถึง 86.29 นาที มากกว่าเส้นทางผลลัพธ์ที่ได้ในรูปที่ 4-9 มาก

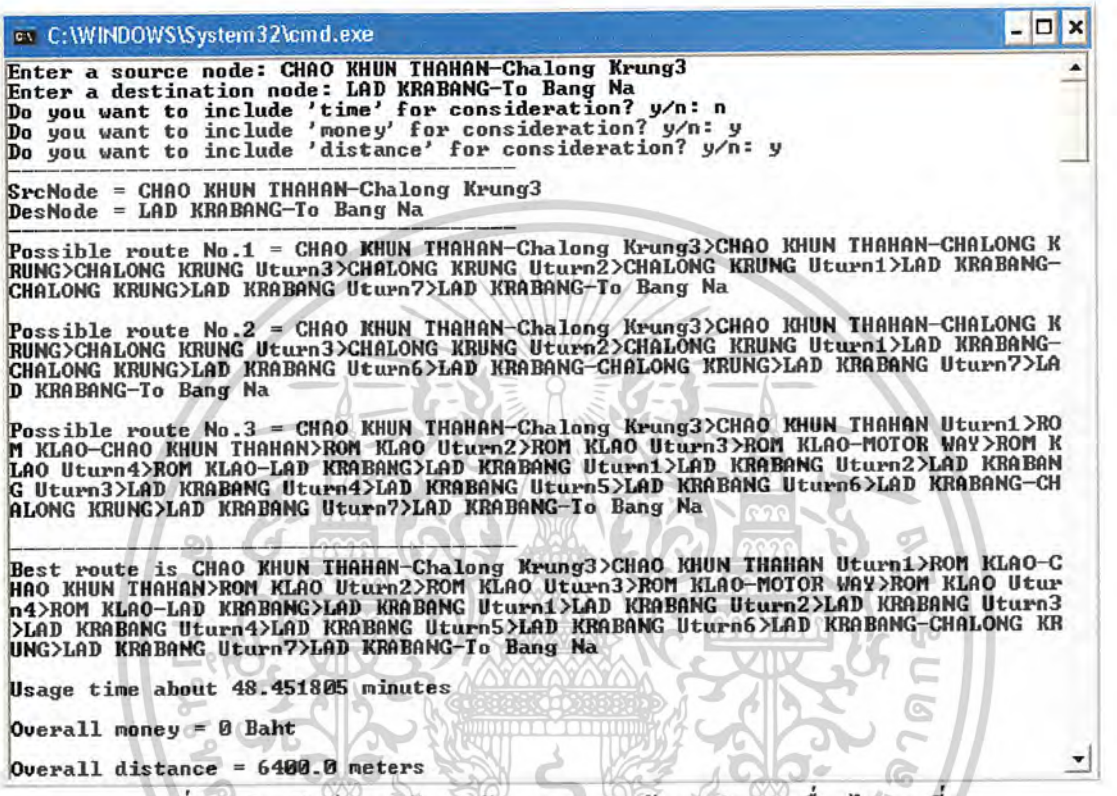


รูปที่ 4-10 ผลลัพธ์ในการค้นหาเส้นทางจากโหนด EEE ไปยังโหนด SSS ตามเงื่อนไขแบบที่ 2



รูปที่ 4-11 ผลลัพธ์ในการค้นหาเส้นทางจากฐานข้อมูลจริง ตามเงื่อนไขแบบที่ 1

เมื่อนำไปใช้งานจริง และมีการติดต่อกับฐานข้อมูล MySQL ซึ่งเป็นฐานข้อมูลที่เก็บรายละเอียดของโหนดบนท้องถนนจริง ผลลัพธ์ที่ได้จะมีลักษณะดังแสดงในรูปที่ 4-11 และรูปที่ 4-12 ซึ่งทั้งสองรูปเป็นการค้นหาเส้นทางที่ดีที่สุดจากโหนด CHAO KHUN THAHAN-Chalong Krung3 ไปยังโหนดปลายทางที่โหนด LAD KRABANG-To Bang Na แต่มีการเลือกเงื่อนไขที่แตกต่างกัน ทำให้ได้ผลลัพธ์ออกมาเป็นเส้นทางที่ดีที่สุดแตกต่างกันตามเงื่อนไขของผู้ใช้ระบบ



รูปที่ 4-12 ผลลัพธ์ในการค้นหาเส้นทางจากฐานข้อมูลจริง ตามเงื่อนไขแบบที่ 2

บทที่ 5
วิเคราะห์ และสรุปผลโครงงาน

5.1 วิเคราะห์ และสรุปผลการทดสอบการใช้โปรแกรม

ผลการทดสอบในการใช้โปรแกรมเพื่อค้นหาเส้นทาง โดยนำอัลกอริทึมในลักษณะโครงสร้างข้อมูลแบบต้นไม้มาใช้ในกระบวนการค้นหาเส้นทาง รวมทั้งใช้อัลกอริทึมในการกำจัดเส้นทางที่เกิดการวนลูป เป็นไปตามวัตถุประสงค์ที่ตั้งไว้ กล่าวคือ โปรแกรมสามารถค้นหาเส้นทางที่ดีที่สุด ตามเงื่อนไขของผู้ใช้ระบบแต่ละรายได้เป็นอย่างดี โดยสามารถอำนวยความสะดวกให้กับผู้ใช้ผ่านทั้งทางเว็บเบราเซอร์และโทรศัพท์มือถือ ดังนั้นถ้านำระบบนี้ไปเปิดให้ใช้งานจริงทั่วไป ก็น่าจะเป็นระบบที่ช่วยลดปัญหาจราจรในกรุงเทพมหานครได้เป็นอย่างดี เพราะระบบจะสามารถเข้าถึงผู้ใช้งานได้อย่างทั่วถึงมากกว่าระบบที่มีอยู่ทั่วไปในปัจจุบัน ทั้งนี้เนื่องจากผู้ที่กำลังขับรถอยู่ตามท้องถนน สามารถเข้ามาใช้ระบบนี้ได้ทันทีผ่านทางระบบโทรศัพท์มือถือ โดยไม่จำเป็นต้องฟังรายงานสภาพการจราจรผ่านทางวิทยุ หรือใช้อุปกรณ์คอมพิวเตอร์ระบบไร้สายแต่อย่างใด

โปรแกรมค้นหาเส้นทางที่ใช้ในระบบนี้ดีกว่าระบบที่เคยมีมา ตรงที่โปรแกรมในระบบนี้เปิดโอกาสให้ผู้ใช้เลือกเงื่อนไขในการค้นเส้นทางว่าต้องการเส้นทางที่ดีที่สุดในแง่ใด ไม่ว่าจะเป็นในด้านระยะเวลาน้อยที่สุดโดยดูจากสภาพการจราจร, ระยะทางสั้นที่สุด, หรือด้านค่าใช้จ่ายในการเดินทางที่น้อยที่สุด ซึ่งระบบที่มีอยู่โดยทั่วไป จะเป็นระบบที่ทำการค้นหาเส้นทางที่สั้นที่สุดจากสถานที่หนึ่งไปยังสถานที่หนึ่งโดยไม่คำนึงถึงสภาพการจราจร หรือไม่ก็เป็นระบบที่รายงานสภาพการจราจรตามถนนเส้นต่างๆ แต่แยกความสามารถในการค้นหาเส้นทางจากสถานที่ต้นทางไปยังสถานที่ปลายทางออกไปต่างหาก และถึงแม้ว่าในปัจจุบันจะเริ่มมีบางระบบที่รวมความสามารถในการค้นหาเส้นทางจากสถานที่หนึ่งไปยังอีกสถานที่หนึ่ง โดยคำนึงถึงสภาพการจราจรด้วยก็ตาม แต่ระบบเหล่านั้นก็ยังไม่มีการคำนึงในแง่ของค่าใช้จ่ายที่ต้องเสียให้กับค่าผ่านด่านต่างๆระหว่างเส้นทาง

5.2 ข้อเสนอแนะในการพัฒนาโครงงานต่อ

สำหรับผู้ที่ต้องการนำโครงงานนี้ไปทำการพัฒนาต่อไปนั้น อาจพัฒนาให้โปรแกรมค้นหาเส้นทาง มีความสามารถที่หลากหลายมากขึ้นไปอีก อาทิเช่น สามารถตรวจสอบเส้นทางที่ห้ามวิ่งในทิศทางบางทิศทาง ในช่วงเวลาที่แตกต่างกันในแต่ละวัน หรือแต่ละสัปดาห์, เส้นทางที่บังคับห้ามเลี้ยวในวันคู่หรือวันคี่ เป็นต้น หรืออาจมีการพิจารณาถึงจำนวนช่องทางการเดินรถในถนนเส้นต่างๆ ซึ่งอาจมีผลต่อสภาพการจราจรได้

ภาคผนวก

ภาคผนวก ก: การติดตั้ง Software Development Kit (SDK) ใน Windows

ภาคผนวกนี้ จะกล่าวถึงการติดตั้ง Java Development Kit (JDK) ของ Sun ซึ่งในปัจจุบันถูกนำมาใส่เป็นแพ็กเกจ และเรียกชื่อใหม่ว่า Software Development Kit (SDK)

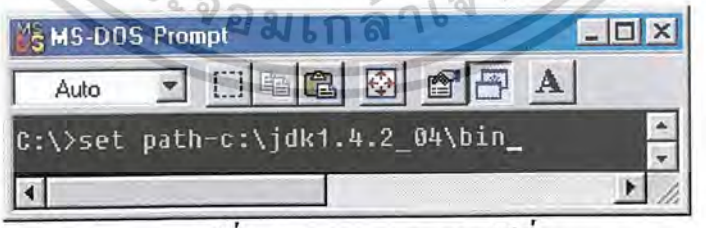
หลังจากทำการดาวน์โหลด SDK จากเว็บไซต์ของ Sun (http://java.sun.com) สำหรับใช้ในระบบปฏิบัติการ Windows (j2sdk-1_4_2_04-windows-i586-p.exe) มาแล้ว ให้ทำการ install โปรแกรมไปตามปกติ อย่างไรก็ตาม ก่อนที่เราจะใช้ JDK ในการคอมไพล์ และ execute โปรแกรมได้ เราจะต้องทำการเซตค่า path เพื่อให้เครื่องคอมพิวเตอร์รู้จักกับชุดคำสั่งของ JDK ดังนี้

- สำหรับ Windows 9x

  สามารถทำได้สามวิธี ดังนี้

  1. กำหนดค่า path สำหรับจาวา ในไฟล์ชื่อ autoexec.bat โดย ใส่คำสั่ง set path = c:\jdk1.*.*\bin; โดยกำหนดให้ * เป็นเวอร์ชั่นของ JDK ที่ผู้ใช้ได้ทำการเลือกในการพัฒนาโปรแกรมจาวา เช่น c:\j2sdk1.4.2_04\bin; เป็นต้น ทั้งนี้ การใส่ค่า path ใน ไฟล์ autoexec.bat ทำได้โดยกดปุ่ม Start เลือก Run แล้วพิมพ์คำว่า sysedit ลงไป จะมีหน้าจอ System Configuration Editor ขึ้นมา ให้ใส่ค่า path ว่า c:\j2sdk1.*.*\bin; ลงไปในหน้า C:AUTOEXEC.BAT แล้วทำการ save หลังจากนั้นให้ทำการ restart เครื่องก่อน การเซต path จึงจะเสร็จสมบูรณ์

  2. กำหนดค่า path ทุกครั้งเมื่อต้องการใช้ชุดคำสั่งของจาวา ดังรูปที่ 6-1



รูปที่ 6-1 การเซต path แบบวิธีที่ 2

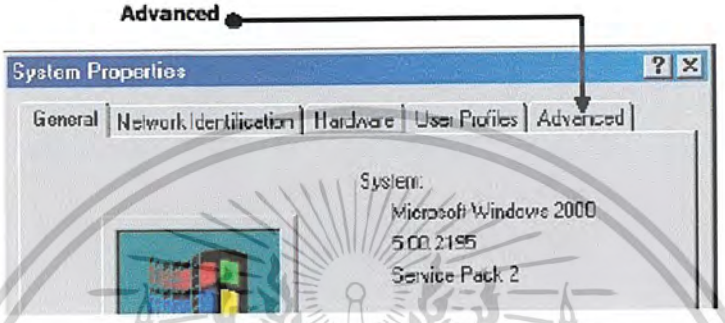
  3. เริ่มทำงานในไดเรกทอรีของจาวา โดยตรง เช่นเปลี่ยนไดเรกทอรีไปที่ c:\j2sdk1.4.2_04\bin และเริ่มทำงานในไดเรกทอรีนี้ โดยไฟล์โค้ดจาวา ที่เขียนขึ้น ก็จะต้องนำมาเก็บไว้ภายในไดเรกทอรีนี้ด้วยเช่นกัน ดังนั้น วิธีนี้จึงเป็นวิธีที่ไม่แนะนำ ทั้งนี้เนื่องจาก การนำไฟล์โค้ดจาวาที่เขียนขึ้น ไปเก็บรวมกันไว้ภายในไดเรกทอรีเดียวกับ JDK จะทำให้ไปปะปนกับไฟล์อื่นๆ

ของ JDK ที่อยู่ในไดเรกทอรีนั้น

- สำหรับ Windows 2000/XP

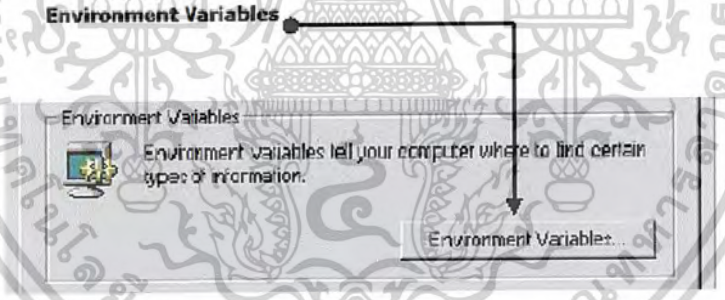
การเซต path สำหรับใน Windows 2000/XP นั้น จะต้องเข้าไปเซตที่ Environment Variables ใน System properties ของเครื่อง โดยมีขั้นตอนการเซตดังนี้

1. คลิกขวาของเมาส์บนไอคอนของ My Computer เพื่อเปิดหน้าต่างของ System Properties
2. กดเลือกไปที่แท็บ Advanced ดังรูปที่ 6-2



รูปที่ 6-2 เลือกแท็บ Advanced

3. กดปุ่ม Environment Variables ดังรูปที่ 6-3



รูปที่ 6-3 คลิกเลือกปุ่ม Environment Variables

4. ในหน้าต่าง System variables เลือกข้อความที่ขึ้นต้นด้วยคำว่า path ดังรูปที่ 6-4

รูปที่ 6-4 เลือกที่บรรทัด path ใน หน้าต่าง System variables

5. กดปุ่ม Edit เพื่อทำการเพิ่ม path ของจาวา (เช่น c:\j2sdk1.4.2_04\bin) เข้าสู่ระบบ โดยใช้เครื่องหมายเซมิโคลอน (;) ในการแบ่ง path แต่ละ path ออกจากกัน ดังรูปที่ 6-5

รูปที่ 6-5 การใส่คำสั่งเซต path ใหม่

6. กดปุ่ม OK เพื่อให้ระบบบันทึกการเปลี่ยนแปลงที่ได้ทำขึ้น

ภาคผนวก ข: อธิบาย Source Code ของโปรแกรมหลัก

ภาษาที่ใช้ในการเขียนโปรแกรมการค้นหาเส้นทาง คือภาษา JAVA ทั้งนี้ Source Code ในส่วนโปรแกรมหลักที่นำมาอธิบายนี้ เป็น Source Code ที่เขียนเพื่อรับข้อมูลอินพุทของโหนดต้นทาง และโหนดปลายทางจากคีย์บอร์ด ซึ่งเป็นโค้ดเบื้องต้นที่ถูกเขียนขึ้นเพื่อให้สะดวกต่อการทดสอบความถูกต้องของโปรแกรม อย่างไรก็ตาม เมื่อนำไปใช้งานจริง จะมีการเปลี่ยนแปลงโค้ดในส่วนนี้บ้างเล็กน้อย เพื่อให้เข้ากับโครงงานในส่วน Front-end ซึ่งโปรแกรมจะต้องรับค่าอินพุทของโหนดจากผู้ใช้ระบบ ผ่านทางโทรศัพท์มือถือ และเว็บเบราเซอร์

โปรแกรมถูกเขียนขึ้นโดยมีคลาสเพียงคลาสดียว นั่นคือ routing และภายในคลาสจะแบ่งออกเป็นเมธอดหลายๆเมธอด ได้แก่

- main: เป็นเมธอดหลักที่เรียกใช้เมธอดอื่นๆในกระบวนการค้นหาเส้นทาง
- dbAdjNode: เป็นเมธอดที่มีการติดต่อกับฐานข้อมูล MySQL เพื่อดึงเอาข้อมูลของโหนดต่างๆบนท้องถนน มาทำการค้นหาเส้นทางที่ดีที่สุด โดยเมธอดนี้จะรับค่าชื่อของโหนดซึ่งเป็นสตริง มาทำการค้นหาข้อมูลเกี่ยวกับโหนดนั้นในฐานข้อมูล และส่งค่ารายละเอียดที่เกี่ยวข้องกับโหนดนั้นกลับออกไปเป็นอาร์เรย์ของสตริง
- getNameAdjNode: เป็นเมธอดที่ทำการดึงเอาค่าโหนดข้างเคียงของโหนดล่าสุดในแต่ละเส้นทางออกมาจากข้อมูลที่ได้มาจากเมธอด dbAdjNode เมื่อได้โหนดข้างเคียงมาแล้ว ก็ทำการเพิ่มต่อท้ายเข้าไปในเส้นทางแต่ละเส้นทาง
- findRoutes: เป็นเมธอดที่ตรวจสอบว่าโหนดสุดท้ายของเส้นทางที่กำลังทำการแตกโหนดออกไป เป็นโหนดปลายทางแล้วหรือไม่ ถ้ายังได้เส้นทางไม่ครบสามเส้นทาง หรือเส้นทางที่ทำการแตกโหนดอยู่ ไม่ถูกตัดทิ้งออกไปหมดเนื่องจากเป็นเส้นทางที่เกิดการวนลูป ก็จะทำการแตกโหนด และตรวจสอบโหนดท้ายสุดของแต่ละเส้นทางต่อไป

- checkLoopType1: รับค่าสตริงของเส้นทางแต่ละเส้นทางในขณะที่ดำเนินการค้นหาเส้นทาง เพื่อทำการตรวจสอบเส้นทางที่เกิดการวนลูปในลักษณะ A-B-A-B
- checkLoopType2: รับค่าสตริงของเส้นทางแต่ละเส้นทางในขณะที่ดำเนินการค้นหาเส้นทาง เพื่อทำการตรวจสอบเส้นทางที่เกิดการวนลูปในลักษณะ A-B-C-B-D, A-B-C-B-A และ A-B-C-A-D
- checkLoopType3: รับค่าสตริงของเส้นทางแต่ละเส้นทางในขณะที่ดำเนินการค้นหาเส้นทาง เพื่อทำการตรวจสอบเส้นทางที่เกิดการวนลูปในลักษณะ A-B-A
- checkLoopType4: รับค่าสตริงของเส้นทางแต่ละเส้นทางในขณะที่ดำเนินการค้นหาเส้นทาง เพื่อทำการตรวจสอบเส้นทางที่เกิดการวนลูปในลักษณะ A-B-C-D-B-E และ A-B-C-D-A-E
- checkLoopType5: รับค่าสตริงของเส้นทางแต่ละเส้นทางในขณะที่ดำเนินการค้นหาเส้นทาง เพื่อทำการตรวจสอบเส้นทางที่เกิดการวนลูปในลักษณะ A-B-C-D-E-B-F
- findBestRoute: เป็น method ที่รับค่าเส้นทางสมบูรณ์จากโหนดต้นทางไปยังโหนดปลายทางที่ค้นหาได้ มาทำการพิจารณาเลือกเส้นทางที่ดีที่สุดตามเงื่อนไขของผู้ใช้ระบบ ผลลัพธ์ที่ส่งกลับออกมา จะเป็นอาร์เรย์ของสตริงที่ระบุเส้นทางที่ดีที่สุดตามเงื่อนไขของผู้ใช้เพียงเส้นทางเดียว รวมทั้งรายละเอียดอื่นๆ ได้แก่ ระยะเวลาโดยประมาณที่ใช้ในการเดินทาง, ระยะทางรวมจากต้นทางถึงปลายทาง และค่าใช้จ่ายที่ต้องใช้ในระหว่างการเดินทาง

Source Code ในส่วนโปรแกรมหลัก หรือในส่วนเมธอด main มีดังนี้

```
import java.io.*;
import java.sql.*;

public class routing {
 public static void main (String args[]) throws IOException,SQLException
 {
  String SrcNode;
  String DesNode;
  String timeOption, moneyOption, distanceOption;
  String warningMessage = "";
```

ตารางที่ 6-1 แสดงโค้ดในส่วนเมธอด main ส่วนที่ 1

ส่วนของโค้ดในตารางที่ 6-1 นี้ มีการอิมพอร์ตเมธอดจากคลาสไลบรารี โดยต้องมีการอิมพอร์ต

java.sql.* ด้วย เนื่องจากในส่วนของเมธอด dbAdjNode ของโปรแกรม จะมีการติดต่อกับฐานข้อมูล MySQL

หลังจากนั้นทำการประกาศชื่อคลาสเป็น routing และทำการประกาศเมธอด main ตามมา ส่วนการประกาศตัวแปรชนิดสตริง SrcNode และ DesNode เป็นการประกาศตัวแปรที่ใช้การรับค่าโหนดต้นทาง และโหนดปลายทาง ในขณะที่ตัวแปร timeOption, moneyOption และ distanceOption ใช้เก็บค่าเงื่อนไขที่ผู้ใช้ระบบป้อนเข้ามา ส่วนตัวแปร warningMessage ใช้เก็บค่าข้อความเพื่อแสดง ในกรณีที่ผู้ใช้ระบบป้อนโหนดต้นทาง และโหนดปลายทางเข้ามาเป็นโหนดเดียวกัน

```
BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
System.out.print("Enter a source node: ");
SrcNode = stdin.readLine();
System.out.print("Enter a destination node: ");
DesNode = stdin.readLine();
System.out.print("Do you want to include 'time' for consideration? y/n: ");
timeOption = stdin.readLine();
System.out.print("Do you want to include 'money' for consideration? y/n: ");
moneyOption = stdin.readLine();
System.out.print("Do you want to include 'distance' for consideration? y/n: ");
distanceOption = stdin.readLine();
```

ตารางที่ 6-2 แสดงโค้ดในส่วนเมธอด main ส่วนที่ 2

จากโค้ดในตารางที่ 6-2 เป็นการรับค่าอินพุทที่ผู้ใช้ป้อนเข้ามา ซึ่งเมื่อนำไปใช้ร่วมกับส่วน Front-end โค้ดในส่วนนี้จะถูกตัดทิ้งไป เนื่องจากค่าอินพุทเหล่านี้จะถูกส่งมาจากโปรแกรมของส่วน Front-end ไม่ใช่จากผู้ใช้ระบบโดยตรง

```
if(SrcNode.equals(DesNode))  {
  warningMessage = "Error! Your destination is the same place as where you are.";
  System.out.println("----------------------------------------");
  System.out.println(warningMessage);
 }
else  {
  String[][] adjNodeDetails;
  adjNodeDetails = dbAdjNode(SrcNode);
```

ตารางที่ 6-3 แสดงโค้ดในส่วนเมธอด main ส่วนที่ 3

โค้ดในตารางที่ 6-3 เป็นการตรวจสอบว่า โหนดต้นทางและโหนดปลายทางที่ผู้ใช้ระบบป้อนเข้ามานั้น เป็นโหนดเดียวกันหรือไม่ ถ้าเป็นโหนดเดียวกัน โปรแกรมจะทำการแจ้งข้อผิดพลาดแก่ผู้ใช้ระบบ แต่ถ้าโหนดต้นทางและโหนดปลายทางเป็นคนละโหนดกัน โปรแกรมก็จะเริ่มกระบวนการค้นหาเส้นทาง โดยการส่งค่าของโหนดต้นทางไปยังเมธอด dbAdjNode เพื่อดึงเอารายละเอียดเกี่ยวกับโหนดต้นทางนี้ออกมาจากฐานข้อมูล ค่าผลลัพธ์ที่ได้ จะถูกนำมาเก็บไว้ในตัวแปร adjNodeDetails

```
String[] routeBuffer;
routeBuffer = getNameAdjNode(adjNodeDetails, SrcNode);

String[] resultRoutes;
resultRoutes = findRoutes(routeBuffer, DesNode);

String[] bestRouteDetails = new String[4];
bestRouteDetails = findBestRoute(getResultRoutes, timeOption, moneyOption,
                    distanceOption);
```

ตารางที่ 6-4 แสดงโค้ดในส่วนเมธอด main ส่วนที่ 4

หลังจากได้รายละเอียดของโหนดต้นทางมาแล้ว ก็จะส่งข้อมูลนั้น รวมทั้งค่าของโหนดต้นทางไปยังเมธอด getNameAdjNode เพื่อไปทำการดึงเอาค่าโหนดข้างเคียงของโหนดที่อยู่ล่างสุดของเส้นทาง (ในครั้งแรกของกระบวนการค้นหาเส้นทาง โหนดที่อยู่ล่างสุดก็คือโหนดต้นทางนั่นเอง) ออกมา และทำการต่อท้ายเข้าไปในเส้นทางแต่ละเส้นทาง ซึ่งในขณะดำเนินการอยู่ภายในเมธอด getNameAdjNode นี้ จะมีการเรียกใช้เมธอด checkLoopType1, checkLoopType2, checkLoopType3, checkLoopType4 และ checkLoopType5 ด้วย เพื่อทำการตรวจสอบว่า หลังจากทำการเพิ่มโหนดใหม่ต่อท้ายเข้าไปในเส้นทางแล้ว เส้นทางใหม่ที่ได้เกิดการวนลูปหรือไม่ ถ้าเกิดการวนลูปเส้นทางนั้นก็จะถูกตัดทิ้งไป

เมื่อเมธอด getNameAdjNode ส่งค่าอาร์เรย์ของสตริงเส้นทางที่ทำการเพิ่มโหนดล่าสุดเข้าไปใหม่แล้ว ก็จะนำมาเก็บไว้ในตัวแปร routeBuffer ซึ่งหลังจากนั้นจะถูกส่งไปยังเมธอด findRoutes พร้อมกับค่าโหนดปลายทาง โดยในเมธอด findRoutes นี้ จะทำการค้นหาเส้นทางที่สมบูรณ์จากต้นทางไปจนถึงปลายทางไปเรื่อยๆ จนกว่าจะได้เส้นทางครบ 3 เส้นทาง หรือเส้นทางที่มีอยู่ถูกตัดทิ้งไปหมดเนื่องจากเป็นเส้นทางที่เกิดการวนลูป ซึ่งในระหว่างดำเนินการอยู่ภายในเมธอด findRoutes นั้น จะมีการเรียกใช้เมธอด dbAdjNode เพื่อดึงเอารายละเอียดของแต่ละโหนดออกมาจากฐานข้อมูลด้วย

ผลลัพธ์ที่ได้จากเมธอด findRoutes จะถูกนำมาเก็บในตัวแปร resultRoutes ซึ่งจะถูกส่งต่อไปยังเมธอด findBestRoute ต่อไป พร้อมกับค่าในตัวแปร timeOption, moneyOption และ distanceOption ทั้งหมดนี้จะถูกนำไปใช้ในการพิจารณาค้นหาเส้นทางที่ดีที่สุดเพียงเส้นทางเดียว ค่าที่ส่งกลับมาจะถูกนำไปเก็บไว้ในตัวแปร bestRouteDetails ซึ่งจะมีค่าสตริงอยู่ 4 ค่า นั่นคือเส้นทางที่ดีที่สุด, ระยะเวลา

โดยประมาณ, ระยะทางรวม และค่าใช้จ่ายที่ต้องใช้ในการเดินทาง

```
        String displayRoute = "", displayTime = "", displayMoney = "", displayDistance = "";


        displayRoute = bestRouteDetails[0];
        displayTime = bestRouteDetails[1];
        displayMoney = bestRouteDetails[2];
        displayDistance = bestRouteDetails[3];


        System.out.println("---------------------------------------");
        System.out.println("Best route is " + displayRoute);
        System.out.println("Usage time is about " + displayTime + " minutes");
        System.out.println("Overall money = " + displayMoney + " Baht");
        System.out.println("Overall distance = " + displayDistance + " meters");
    }
}
```

ตารางที่ 6-5 แสดงโค้ดในส่วนเมธอด main ส่วนที่ 5

โค้ดในส่วนที่ 5 เป็นการนำค่าเส้นทางที่ดีที่สุด, ระยะเวลาโดยประมาณ, ระยะทางรวม และค่าใช้จ่ายที่ต้องใช้ในการเดินทาง ซึ่งถูกเก็บอยู่ในตัวแปรอาร์เรย์ bestRouteDetails มาแยกออกเก็บในตัวแปรแบบสตริงโดดๆ ทั้งนี้เพื่อง่ายต่อส่วน Front-end ในการนำค่าไปใช้งานต่อไป

ในขณะเดียวกัน ส่วนแสดงผลออกหน้าจอให้ผู้ใช้ (System.out.println) นั้น เมื่อนำไปใช้ร่วมกับส่วน Front-end ส่วนนี้จะถูกตัดทิ้งไป แต่จะเป็นการส่งค่ากลับไปยังส่วน Front-end แทน

บรรณานุกรม

[1] Mark Allen Weiss: "Data Structures & Algorithm Analysis in C++", 2nd Edition, Addison-Wesley

[2] Harry F. Smith: "Data Structures: Form and Function", 1st Edition, Harcourt Brace Jovanovich

[3] Kruse R. L.: "Data Structures and Program Design", 3rd Edition, Prentice Hall

[4] Tenenbaum A., Langsam and Augenstein M.: "Data Structures using C", Englewood Cliffs, Prentice Hall, 1981

[5] Bowman C. F.: "Algorithms and Data Structures : an Approach in C", Saunders College Publishing

[6] H.M. Deitel and P.J. Deitel: "JAVA How to Program", 3rd Edition, Prentice Hall, Upper Saddle, New Jersey

[7] ทรงลักษณ์ พิริยะไพโรจน์, สุมนา เกษมสวัสดิ์: "เรียนลัด Data Structure ด้วย Visual Basic", พิมพ์ครั้งที่หนึ่ง, บรรณาธิการ วศิน เพิ่มทรัพย์, สำนักพิมพ์โปรวิชั่น จำกัด

[8] กิตติ ภักดีวัฒนะกุล: "JAVA ฉบับพื้นฐาน", พิมพ์ครั้งที่สาม, บริษัท เคทีพี คอมแอนด์คอนซัลท์ จำกัด

[9] Sun's JAVA Official Website: "http://java.sun.com", 1994-2004 Sun Microsystems, Inc.

[10] Borland Developer Network: "http://bdn.borland.com", 1994-2004 Borland Software Corporation

[11] MySQL Official Website: "http://www.mysql.com", 1995-2004 MySQL AB