โปรแกรมค้นหาเส้นทาง

Path Finding Programming

นาย พุทธิพร ปุรณวัฒนกุลชัย

Mr. Puttiporn Puranawattanakulchai

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมอุตสาหการ

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

หัวข้อปริญญานิพนธ์	โปรแกรมค้นหาเส้นทางที่สั้นที่สุด Path finding programming
นักศึกษา	นาย พุทธิพร ปุรณวัฒนกุลชัย รหัสประจำตัว 42010583
ปริญญา	วิศวกรรมศาสตรบัณฑิต
สาขา	วิศวกรรมอุตสาหการ
ปีการศึกษา	2545

อาจารย์ผู้ควบคุมปริญญานิพนธ์

(ผศ.ดร.สรรพสิทธ์ ลิ่มนรรัตน์)

(อ.พลชัย โชติปรายนกุล)

หัวข้อปริญญานิพนธ์	โปรแกรมค้นหาเส้นทาง
นักศึกษา	นาย พุทธิพร ปุรณวัฒนกุลชัย
ระดับการศึกษา	วิศวกรรมศาสตรบัณฑิต สาขาวิศวกรรมอุตสาหการ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา	2545
อาจารย์ผู้ควบคุมปริญญานิพนธ์	ผศ.ดร.สรรพสิทธ์ ลิ่มนรรัตน์ อ.พลชัย โชติปรายนกุล

บทคัดย่อ

ในปัจจุบันสิ่งที่เป็นปัญหามากที่สุดอย่างหนึ่งของกรุงเทพมหานคร คือ การจราจรติดขัด ด้วยปริมาณรถที่เพิ่มมากขึ้นทุกปีแต่ปริมาณถนนเท่าเดิม ก่อให้เกิดความเสียหายทั้งทางด้านเศรษฐกิจ ด้านมลภาวะ ด้านการท่องเที่ยว และด้านการสิ้นเปลืองพลังงาน เป็นต้น ได้มีความพยายามจากทางด้านรัฐบาลที่จะแก้ไขปัญหานี้มาเป็นเวลานานหลายยุคหลายสมัยแล้ว แต่เนื่องจากปริมาณรถที่เพิ่มมากขึ้นทุกวันรวมทั้งความซับซ้อนในการวิเคราะห์เส้นทางทำให้การแก้ไขปัญหาไม่ประสบความสำเร็จ

ปริญญานิพนธ์ฉบับนี้เป็นการนำเสนอ การนำคอมพิวเตอร์มาช่วยในการวิเคราะห์ปัญหาต่างๆและช่วยในการตัดสินใจ ในการวิเคราะห์เส้นทางเพื่อค้นหาเส้นทางที่ดีที่สุดในการเดินทาง โดยได้นำหลักการเกี่ยวกับข้อมูลที่เชื่อมต่อกันเป็นโครงข่าย หรือ กราฟ พร้อมกับเทคนิคในการค้นหาเส้นทางที่สั้นที่สุด มาประยุกต์ใช้กับแผนที่กรุงเทพมหานคร เพื่อนำไปใช้ในการแก้ปัญหาในสถานการณ์จริง หรือ นำไปประยุกต์เป็นระบบสอบถามเส้นทางต่อไปในอนาคต

Thesis title	Path finding programming
Student	Mr.Puttiporn Puranawattanakulchai
Degree	Bachelor of engineering in industrial engineering
	King mongkut's institute of technology ladkrabang
Academic year	2545
Advisor	Asst.Prof.Dr.Sunpasit Limnararat
	Mr.Pholchai Chotiprayanakul

ABTRACT

In the past, The most problem of Bangkok is the traffic-jam. Because increasing of the car but the roads can't increase. That destroy economy, traveling, and energy. Government try solve this problem but solving can't successful.

This thesis is the problems analysis and the shortest path decision by computer program. Design computer program by graph data structure theory and search shortest path principle. That apply with Bangkok map. This project can solve the real problems or application to the path inquire system in the future.

กิตติกรรมประกาศ

การจัดทำปริญญานิพนธ์ฉบับนี้สำเร็จรุล่วงได้ด้วยดี ด้วยการได้รับความเมตตาอย่างยิ่งจากอาจารย์ผู้ควบคุมปริญญานิพนธ์ อาจารย์ ผศ.ดร. ผศ.ดร.สรรพสิทธ์ ลิ่มนรรัตน์ และ อาจารย์ พลชัย โชติปรายนกุล ในการให้คำแนะนำและช่วยเหลือปรับปรุงแก้ไขข้อบกพร่องต่างๆจนเป็นที่เรียบร้อย ผู้จัดทำรู้สึกซาบซึ้งในความกรุณาและขอกราบขอบพระคุณเป็นอย่างสูง

ผู้จัดทำ ขอกราบขอบพระคุณทาง บริษัท บางกอกไกด์ จำกัด ที่ช่วยให้การสนับสนุน ข้อมูลแผนที่ ข้อมูลต่างๆ และชี้แนะแนวทางประกอบการจัดทำปริญญานิพนธ์ฉบับนี้สำเร็จได้ด้วยดี จึงขอกราบขอบพระคุณเป็นอย่างสูง

สุดท้ายนี้ขอบพระคุณอาจารย์ประจำภาควิชาวิศวกรรมอุตสาหกรรม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง และบุคลากรท่านอื่นๆที่ได้กรุณาช่วยเหลือการขอความร่วมมือเป็นอย่างดี

พุทธิพร ปุรณวัฒนกุลชัย

สารบัญ

หน้า

บทคัดย่อภาษาไทย I

บทคัดย่อภาษาอังกฤษ II

กิตติกรรมประกาศ III

สารบัญ IV

สารบัญตาราง V

สารบัญรูป VI

บทที่ 1 บทนำ

1.1 ที่มาและความสำคัญของโครงงาน 1

1.2 วัตถุประสงค์ 1

1.3 ขอบเขตการศึกษา 1

1.4 ประโยชน์ที่คาดว่าจะได้รับ 1

บทที่ 2 ทฤษฎีที่เกี่ยวข้อง

2.1 โครงสร้างข้อมูลแบบกราฟ(Graph) 2

2.2 เทคนิคเรื่องกราฟแบบมีทิศทาง (Directed Graphs) 4

2.3 ปัญหาทางเดินที่สั้นที่สุดแบบทางเดียว 6

2.4 ปัญหาทางเดินที่สั้นที่สุดแบบหลายทาง 9

2.5 การค้นหาแบบฮิวริสติก (Heuristic search) 12

บทที่ 3 การออกแบบและการดำเนินงาน

3.1 การวางแผนการดำเนินงาน 18

3.2 การออกแบบโปรแกรม 20

3.3 รายละเอียดการออกแบบรูปแบบขั้นตอนการทำงานของโปรแกรม 24

บทที่ 4 ผลการดำเนินงาน

4.1 การวิเคราะห์ข้อมูลจากแผนที่กรุงเทพมหานคร 29

4.2 การวิเคราะห์ข้อมูลจากแผนที่เขตลาดกระบัง 32

4.3 การวิเคราะห์ข้อมูลจากจากโจทย์ตัวอย่างขนาดเล็ก 34

บทที่ 5 สรุปผลการวิเคราะห์และข้อเสนอแนะ

5.1 สรุปผลการวิเคราะห์ 36

5.2 ข้อเสนอแนะ 36

บรรณานุกรม 38

สารบัญตาราง

หน้า

ตารางที่ 2.1 แสดงการคำนวณของ Dijkstra's algorithm ของกราฟรูปที่2.4 8

สารบัญภาพ

หน้า

รูปที่2.1 แสดงตัวอย่างของ Direct Graph (หรือ Digraph)..... 3
รูปที่2.2 แสดงตัวอย่างของ Undirected Graph..... 3
รูปที่2.3 แสดงตัวอย่างของ Cyclic Graph..... 4
รูปที่2.4 แสดงลักษณะของกราฟ..... 4
รูปที่ 2.5 แสดงให้เห็นกราฟแบบมีทิศทางที่มี 4 เวอร์ทิกซ์และ 5 อาร์ซ..... 5
รูปที่ 2.6 แสดงกราฟที่กำหนดความสัมพันธ์ของอาร์ซ..... 5
รูปที่ 2.7 แสดงเมตริกซ์ประชิดของรูปที่ 2.3..... 6
รูปที่2.8 โปรแกรม Dijkstra's Algorithm..... 7
รูปที่ 2.9 แสดงที่มีทิศทางที่มีค่าบนอาร์ซ..... 7
รูปที่ 2.10 ข้อสมมติฐานของการหาระยะทางที่สั้นที่สุดไปยัง w..... 8
รูปที่ 2.11 แสดงทางเดินที่สั้นที่สุดที่เป็นไปได้..... 9
รูปที่2.12 แสดงให้เห็นการเดินทางจากเวอร์ทิกซ์ i ไปยังเวอร์ทิกซ์ j โดยมีเวอร์ทิกซ์ k ร่วมอยู่ด้วย..... 10
รูปที่ 2.13 แสดงกราฟแบบมีทิศทางที่กำหนดค่าบนอาร์ซอันหนึ่ง..... 11
รูปที่ 2.14 แสดงค่าของเมตริกซ์ A ที่ได้รับการวนรอบ 3 ครั้ง..... 11
รูปที่ 2.15 โปรแกรม Floyd's Algorithm..... 12
รูปที่ 2.16 แสดงGenerate and test..... 13
รูปที่ 2.17 แสดงตัวอย่างทางเดินในการค้นหาแบบตามแนวลึกก่อน..... 15
รูปที่ 2.18 แสดงการค้นหาแบบตามแนวกว้าง..... 16
รูปที่ 2.19 แสดงการค้นหาแบบทางเลือกที่ดีที่สุด..... 17
รูปที่ 3.1 แสดงการรับค่าของชื่อ Node..... 20
รูปที่ 3.2 แสดงการกำหนดทิศทางของโปรแกรม..... 20
รูปที่ 3.3 แสดงการผลการกำหนดทิศทางของโปรแกรม..... 21
รูปที่ 3.4 แสดงการกำหนดระยะทางของโปรแกรม..... 21
รูปที่ 3.5 แสดงผลการกำหนดระยะทางของโปรแกรม..... 21
รูปที่ 3.6 แสดงการกำหนดความเร็วโดยประมาณของพาหนะของโปรแกรม..... 22
รูปที่ 3.7 แสดงตัวอย่างการสร้างโครงข่ายระยะทางของโปรแกรม..... 22
รูปที่ 3.8 แสดงหน้าจอส่วนประมวณผลของโปรแกรม..... 23
รูปที่ 3.9 แสดงผลลัพธ์ของส่วนประมวณผลของโปรแกรม..... 23
รูปที่ 3.10 แสดงการเลือกแผนที่จากหน้าจอหลัก..... 24
รูปที่ 3.11 แสดงการกำหนดจุดเริ่มต้นและจุดปลายทาง..... 24
รูปที่ 3.12 แสดงหน้าจอ find short path..... 25
รูปที่ 3.13 แสดงการคำนวณเมตริกซ์ประชิด และ แสดงผลลัพธ์ที่คำนวณได้..... 25
รูปที่ 3.14 แสดงผลลัพธ์เส้นทางที่สั้นที่สุด..... 26

รูปที่3.15 แสดงขั้นตอนการทำงานโดยรวมของโปรแกรม..... 27
รูปที่3.16 แสดงขั้นตอนในการประมวลผลจากทฤษฎีของ Dijkstra's algorithm..... 28
รูปที่ 4.1 แสดงฟอร์มเริ่มต้นโปรแกรม..... 29
รูปที่ 4.2 แสดงฟอร์มหลักของโปรแกรม..... 30
รูปที่ 4.3 แสดงภาพแผนที่กรุงเทพมหานครที่ใช้..... 30
รูปที่ 4.4 แสดงฟอร์มการค้นหาเส้นทางที่สั้นที่สุดของโปรแกรม..... 31
รูปที่ 4.5 แสดงผลลัพธ์ของการค้นหาเส้นทางที่สั้นที่สุดจากโปรแกรม..... 31
รูปที่ 4.6 แสดงผลาเส้นทางที่สั้นที่สุดโปรแกรม..... 32
รูปที่ 4.7 แสดงภาพแผนที่เขตลาดกระบัง..... 33
รูปที่ 4.8 แสดงผลลัพธ์การคำนวณเส้นทางที่สั้นที่สุดจากจุด 2 ถึง 40 ของข้อมูลเขตลาดกระบัง..... 34
รูปที่ 4.9 แสดงภาพโครงข่ายตัวอย่าง..... 34
รูปที่ 4.10 แสดงผลลัพธ์การคำนวณเส้นทางที่สั้นที่สุดจากจุด A ถึง F ของข้อมูลโครงข่ายตัวอย่าง..... 35

บทที่ 1
บทนำ

1.1 ที่มาและความสำคัญของโครงงาน

ในปัจจุบันสิ่งที่เป็นปัญหามากที่สุดอย่างหนึ่งของกรุงเทพมหานคร คือ การจราจรติดขัด ด้วยปริมาณรถที่เพิ่มมากขึ้นทุกวันแต่ปริมาณถนนเท่าเดิม ก่อให้เกิดความเสียหายทั้งทางด้านเศรษฐกิจ ด้านมลภาวะ ด้านการท่องเที่ยว และด้านการสิ้นเปลืองพลังงาน เป็นต้น ได้มีความพยายามจากทางด้านรัฐบาลที่จะแก้ไขปัญหานี้มาเป็นเวลานานหลายยุคหลายสมัยแล้ว แต่เนื่องจากปริมาณรถที่เพิ่มมากขึ้นทุกวันทำให้การแก้ไขปัญหาไม่ประสบความสำเร็จ

ดังนั้น วิธีการที่นำมาใช้เพื่อลดเวลาในการเดินทางลงให้น้อยที่สุด จึงเกิดขึ้น เช่น การเลี่ยงการเดินทางในช่วงที่มีการจราจรติดขัด การใช้รถจักรยานยนต์แทนรถยนต์ รวมทั้งการหลีกเลี่ยงเส้นทางที่มีการจราจรติดขัด เป็นต้น แต่วิธีที่นิยมมากที่สุดวิธีหนึ่ง คือ การศึกษาเพื่อหาเส้นทางที่สั้นที่สุด ซึ่งการเดินทางในเส้นทางที่สั้นที่สุดอาจจะไม่ใช่วิธีที่ดีที่สุดจริงๆเพราะในทางปฏิบัติสิ่งที่ต้องการในการแก้ปัญหานี้ คือ การใช้เวลาในการเดินทางที่น้อยที่สุด

ปริญญานิพนธ์ฉบับนี้เป็นการนำเสนอ การนำคอมพิวเตอร์มาช่วยในการวิเคราะห์ปัญหาต่างๆและช่วยในการตัดสินใจ ในการวิเคราะห์เส้นทางเพื่อค้นหาเส้นทางที่ดีที่สุดในการเดินทาง โดยได้นำหลักการเกี่ยวกับข้อมูลที่เชื่อมต่อกันเป็นโครงข่าย (Network) หรือ กราฟ (Graph) มาประยุกต์ใช้กับแผนที่กรุงเทพมหานคร

1.2 วัตถุประสงค์

1) ออกแบบและพัฒนาโปรแกรมคอมพิวเตอร์ที่ช่วยคำนวณหาเส้นทางที่สั้นสุดเพื่อประหยัดเวลาและค่าใช้จ่ายในการเดินทางเพื่อการขนส่งลงได้
2) เพื่อประยุกต์ใช้ให้เกิดประโยชน์กับเส้นทางจราจรหรือโครงข่ายอื่นๆได้ และ สามารถนำไปประยุกต์ใช้ในการพัฒนาระบบสอบถามเส้นทางต่อไปในอนาคตทั้งภาครัฐบาลและเอกชน

1.3 ขอบเขตการศึกษา

1) เส้นเดินทางที่กล่าวถึงนี้จะพิจารณาเฉพาะเส้นทางที่รถยนต์สามารถวิ่งได้เท่านั้น
2) เส้นทางที่ใช้พิจารณา จะพิจารณาเฉพาะเส้นทางในจังหวัดกรุงเทพมหานครเท่านั้น
3) ทิศทางการเดินรถจะใช้การกำหนดจากทิศทางจริง คือ การเดินรถทางเดียว (One-Way) และ การเดินรถสวนทาง (Two-Way)
4) ออกแบบพัฒนาโปรแกรมด้วยภาษา Basic ภายใต้ระบบปฏิบัติการ Microsoft window

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1) ทราบถึงเส้นทางที่เหมาะสมในการเดินทางจากจุดหนึ่งไปยังอีกจุดหนึ่ง
2) ช่วยประหยัดเวลาและค่าใช้จ่ายในการค้นหาเส้นทางเพื่อการขนส่ง
3) ได้นำความรู้เรื่องเทคนิคการหาเส้นทางที่สั้นที่สุด (Shortest Path Problem) มาประยุกต์ใช้กับเส้นทางจริงได้
4) สามารถนำโปรแกรมที่เขียนขึ้นไปประยุกต์ใช้กับโครงข่ายของเส้นทางอื่นๆ
5) สามารถประมาณเวลาที่จะใช้ในการเดินทางได้

บทที่ 2
ทฤษฎีที่เกี่ยวข้อง

2.1 โครงสร้างข้อมูลแบบกราฟ(Graph)

กราฟ (Graph) เป็นโครงสร้างข้อมูลที่ใช้แสดงความสัมพันธ์ระหว่างออบเจ็ค(object) โดยประกอบด้วยกลุ่มของโหนด(Vertex) ที่ใช้แทนออบเจ็คและกลุ่มของเส้นเชื่อม (Edge) ระหว่างโหนดกรณีที่ออบเจ็คตั้งแต่ 2 ออบเจ็คขึ้นไปมีความสัมพันธ์กัน ก็จะมีเส้นเชื่อมระหว่างออบเจ็คหรือโหนดเหล่านั้น

หลักการเกี่ยวกราฟสามารถนำไปประยุกต์ใช้ในงานต่างๆในชีวิตประจำวันได้ เช่น การคำนวณหาระยะทางหรือเวลาที่สั้นที่สุดสามารถเดินทางไปยังจุดต่างๆตามที่กำหนดไว้ในแผนการเดินทางของนักท่องเที่ยว พนักงานขายหรืออื่นๆเพื่อจะได้วางแผนการเดินทางล่วงหน้าได้อย่างมีประสิทธิภาพ ทั้งในเรื่องของเวลาและค่าใช้จ่าย นอกจากนี้ยังสามารถนำไปใช้ในการวางแผนการทำงานในส่วนของกระบวนการผลิตหรือการวางแผนโครงการ ซึ่งจะต้องมีการคำนวณเรื่อง Critical Path ซึ่งเป็นการคำนวณหาเส้นทางของกระบวนการผลิตหรือการดำเนินงานของโครงการ ที่จะทำให้สามารถบรรลุเป้าหมายหรือเสร็จสิ้นได้ว่าจะต้องใช้เวลาเท่าไร และกรณีที่งานเกิดล่าช้าจะสามารถผ่อนปรนหรือยอมได้หรือเป็นไปได้หรือไม่อย่างไร นอกจากนี้ยังใช้โครงสร้างของกราฟแทนโหนดและความสัมพันธ์ระหว่างโหนดในเรื่องเครือข่ายคอมพิวเตอร์ (Computer Network) อีกด้วย

กล่าวได้ว่าโครงสร้างข้อมูลแบบกราฟจะประกอบด้วยโหนดและเส้นเชื่อมระหว่างโหนด (กรณีที่โหนดเหล่านั้นมีความสัมพันธ์กัน) โดยสามารถเขียนแทนด้วยสัญลักษณ์ได้ดังนี้

G = (V,E) เมื่อ G คือกราฟ

V คือกลุ่มของโหนด

E คือเส้นเชื่อมระหว่างโหนด

ประเภทของกราฟแบ่งออกได้เป็น 3 ประเภท ดังนี้คือ

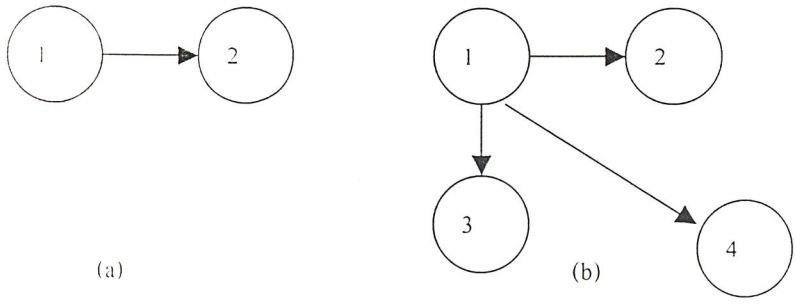
1. Direct graph
2. Undirect graph
3. Cyclic graph

รายละเอียดของกราฟแต่ละชนิดมีดังต่อไปนี้

2.1.1 Direct Graph(หรือ Digraph)

เป็นกราฟที่มีเส้นเชื่อมระหว่างโหนดแสดงทิศทางของการเชื่อมต่อ

ตัวอย่าง Direct Graph (หรือ Digraph) แสดงดังรูปที่ 2.1



รูปที่2.1 แสดงตัวอย่างของ Direct Graph (หรือ Digraph)

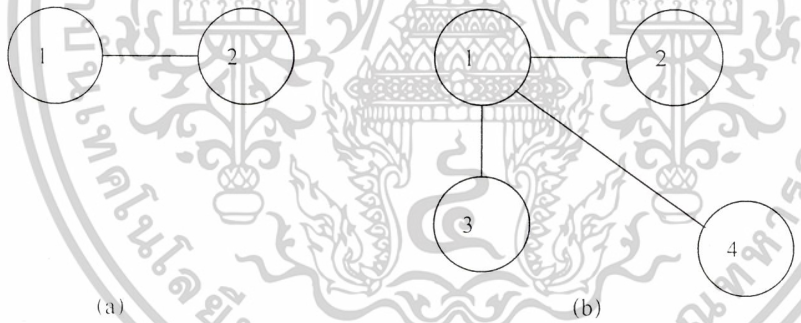
จากรูปที่ 2.1 (a) หมายถึงมีเส้นทางจากโหนด 1 ไปยัง โหนด 2

(b) หมายถึงมีเส้นทางจากโหนด 1 ไปยัง โหนด 2, โหนด 3, โหนด 4

2.1.2 Undirected Graph

เป็นกราฟที่มีเส้นเชื่อมระหว่างโหนดแต่ไม่แสดงทิศทางของการเชื่อมต่อ

ตัวอย่าง Direct Graph (หรือ Digraph) แสดงดังรูปที่ 2.2



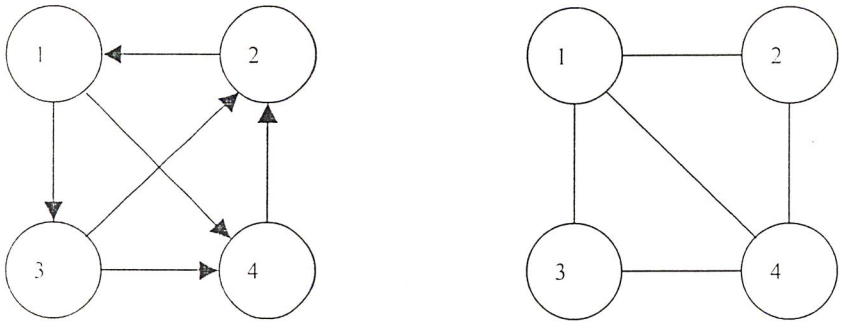
รูปที่2.2 แสดงตัวอย่างของ Undirected Graph

จากรูปที่ 2.1 (a) หมายถึงมีเส้นทางจากโหนด 1 ไปยัง โหนด 2 และมีเส้นทางจากโหนด 2 ไปยัง โหนด 1 ในเส้นทางเดียวกัน

(b) หมายถึงมีเส้นทางจากโหนด 1 ไปยัง โหนด 2, โหนด 3, โหนด 4 ขณะเดียวกันก็มีมีเส้นทางจาก โหนด 4 ไปยัง โหนด 1, มีเส้นทางจากโหนด 3 ไปยัง โหนด 1 และมีเส้นทางจากโหนด 2 ไปยัง โหนด 1 ในเส้นทางเดียวกัน

2.1.3 Cyclic Graph

เป็นกราฟที่เส้นทาง (path) เกิดจากเส้นเชื่อมระหว่างโหนดที่มีลักษณะเป็นวงจรปิด (Cycle) หมายถึงมีโหนดต้นทางและโหนดปลายทางเป็นโหนดเดียวกัน โดย Cyclic Graph สามารถเป็นได้ทั้ง Direct Graph (Digraph) และ Undirected Graph ตัวอย่าง Cyclic Graph แสดงดังรูปที่ 2.3



(a) (b)

รูปที่2.3 แสดงตัวอย่างของ Cyclic Graph

2.2 เทคนิคเรื่องกราฟแบบมีทิศทาง (Directed Graphs)

กราฟแบบมีทิศทาง G ใดๆ จะประกอบไปด้วยเซตของเวอร์ทิกซ์ (vertices) V และเซตของความสัมพันธ์ระหว่างเวอร์ทิกซ์ หรือที่เรียกว่าอาค (arcs) E เวอร์ทิกซ์ อาจเรียกได้อีกอย่างว่าโหนด (node) และอาร์ซอาจเรียกได้อีกอย่างว่า directed lines อาร์ซจะเป็นคู่ลำดับของเวอร์ทิกซ์ (v,w) โดย v เรียกว่า tail และ w เรียกว่า headของอาร์ซ อาร์ซ (v,w) มักจะแทนอยู่ในรูปของ $v \rightarrow w$ และเขียนได้ดังนี้

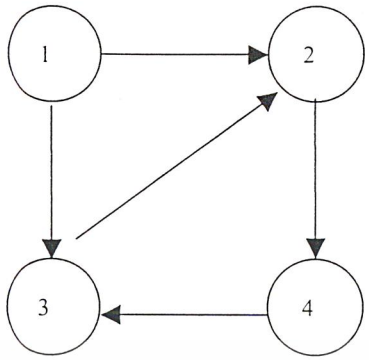
รูปที่2.4 แสดงลักษณะของกราฟ

สังเกตว่าหัวลูกศรจะอยู่ที่เวอร์ทิกซ์ที่เรียกว่า head และส่วนหางของลูกศรจะอยู่ที่เวอร์ทิกซ์ที่เรียกว่า tail จะกล่าวว่าอาร์ซ $v \rightarrow w$ เป็นความสัมพันธ์จาก v ไปยัง w และ w เป็นการประชิด (adjacent) ไปยัง v

เวอร์ทิกซ์ของกราฟแบบมีทิศทางนี้สามารถใช้แทนวัตถุใดๆและอาร์ซจะใช้แทนความสัมพันธ์ระหว่างวัตถุ เช่น เวอร์ทิกซ์อาจแทนชื่อเมืองต่างๆ ในขณะที่อาร์ซอาจแทนเส้นทางการบินระหว่างเมืองหนึ่งสู่อีกเมืองหนึ่ง

Path ของกราฟแบบมีทิศทางหมายถึง ลำดับของเวอร์ทิกซ์ $v_1,v_2,\ldots,v_n$ เช่น $v_1 \rightarrow v_2$, $v_2 \rightarrow v_3, \ldots, v_{n-1} \rightarrow v_n$ โดยที่ path นี้เริ่มจากเวอร์ทิกซ์ v_1 ไปยังเวอร์ทิกซ์ v_n ผ่านเวอร์ทิกซ์ $v_2,v_3,\ldots,v_{n-1}$ และสิ้นสุดที่ v_n

Length หรือความยาวของ path คือ จำนวนของอาร์ซบน path ในกรณีนี้คือ n-1 ขอให้ศึกษารายละเอียดดังตัวอย่างดังรูปที่2.2

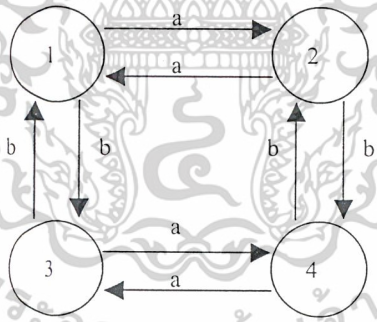


รูปที่ 2.5 แสดงให้เห็นกราฟแบบมีทิศทางที่มี 4 เวอร์ทิกซ์และ 5 อาร์ซ

ในรูปที่2.2 ลำดับของเวอร์ทิกซ์ 1,2,4 คือ path ที่มี length 2 จากเวอร์ทิกซ์ 1 ไปยังเวอร์ทิกซ์ 4 และ path 3,2,4,3 จะเป็นวงรอบที่มี length 3

ในการประยุกต์ใช้งานหลายๆ อย่าง มักจะกำหนดเครื่องหมายไว้บนเวอร์ทิกซ์และอาร์ซ โดยที่เครื่องหมายนี้อาจเป็นชื่อ ราคา หรือมูลค่าของข้อมูลต่างๆที่กำหนด

ในรูปที่2.3 แสดงให้เห็นกราฟที่กำหนดเครื่องหมายไว้ในแต่ละอาร์ซ ซึ่งกำหนดโดยตัวอักษรที่แสดงถึงการเปลี่ยนแปลงจากเวอร์ทิกซ์หนึ่งไปยังอีกเวอร์ทิกซ์หนึ่ง



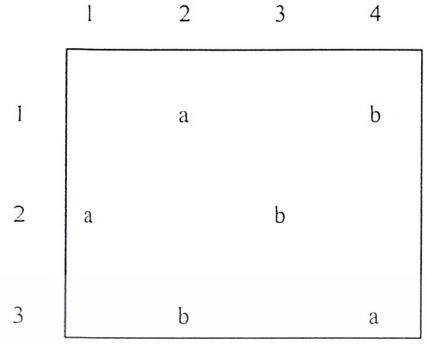
รูปที่ 2.6 แสดงกราฟที่กำหนดความสัมพันธ์ของอาร์ซ

2.2.1 การแทนรูปแบบโครงสร้างกราฟแบบมีทิศทาง

สามารถใช้โครงสร้างข้อมูลหลายชนิดมาใช้แทนกราฟแบบมีทิศทางนี้ แต่โครงสร้างข้อมูลที่เหมาะสมนั้น จะขึ้นอยู่กับการกระทำที่จะนำมาประยุกต์ใช้กับเวอร์ทิกซ์และอาร์ซเหล่านั้น วิธีการแทนที่นิยมกันสำหรับกราฟแบบมีทิศทาง G = (V,E) วิธีหนึ่งก็คือใช้ เมตริกซ์การประชิด (adjacency matrix)

สมมติว่า V = {1,2,...,n} เมตริกซ์การประชิด สำหรับ G คือ n×n เมตริกซ์ A ของบูลีน (Booleans) โดยที่ A[i,j] เป็นจริงถ้าหากมีอาร์ซจากเวอร์ทิกซ์ i ไปยัง j โดยทั่วไปมักจะแทนเมตริกซ์การประชิดด้วย 1 สำหรับกรณีที่เป็นจริง และ 0 ในกรณีที่เป็นเท็จ

ตัวอย่างตามรูปที่2.4 แสดงให้เห็นเมตริกซ์การประชิดของกราฟแบบมีทิศทางในรูปที่2.3 ในที่นี้ เครื่องหมายกำกับจะเป็นตัวอักษร และช่องว่างจะหมายถึง การที่ไม่มีอาร์ซอยู่



รูปที่ 2.7 แสดงเมตริกซ์ประชิดของรูปที่ 2.3

2.3 ปัญหาทางเดินที่สั้นที่สุดแบบทางเดียว

ปัญหาของการหาเส้นทางเดินในกราฟแบบมีทิศทาง โดยสมมติว่ามีกราฟแบบมีทิศทาง G = (V, E) อันหนึ่ง ซึ่งแต่ละอาร์ซไม่มีค่าเป็นลบ ในเวอร์ทิกซ์ใดเวอร์ทิกซ์หนึ่ง จะถูกกำหนดให้เป็นจุดเริ่มต้น (source) ปัญหาก็คือพิจารณา cost (อาจจะเป็น ค่าใช้จ่าย ระยะทาง หรือเวลา ฯลฯ) ของระยะทางที่น้อยที่สุด จากจุดเริ่มต้นไปยังเวอร์ทิกซ์อื่นๆ ทุกๆ เวอร์ทิกซ์ใน V โดยที่ความยาวของเส้นทาง (length of path) ก็คือผลรวมของ cost บนเส้นทางนั้น

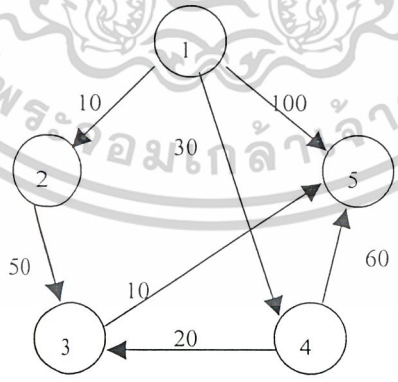
อาจคิดว่า G นั้นคือแผนที่ทางอากาศของสายการบิน ซึ่งในแต่ละเวอร์ทิกซ์จะแทนเมือง และแต่ละอาร์ซ $v \rightarrow w$ คือ เส้นทางการบินระหว่างเมือง v ไปยังเมือง w เครื่องหมายบนอาร์ซ $v \rightarrow w$ ก็คือเวลาในการบินจาก v ไปยัง w การแก้ปัญหาในเรื่องนี้ก็คือ ให้หาเวลาเดินทางที่น้อยที่สุดจากเมืองที่กำหนด ไปยังทุกๆเมืองในแผนที่

ในการแก้ปัญหานี้จะประยุกต์ใช้กับอัลกอริทึมที่เรียกว่า "Dijkstra's Algorithm" อัลกอริทึมนี้ทำงานโดยการพยายามรักษาเซต S ของเวอร์ทิกซ์ต่างๆ ซึ่งทราบระยะทางสั้นที่สุดจากจุดเริ่มต้นไปยังเวอร์ทิกซ์เหล่านั้นสมมติว่าในตอนเริ่มต้นเท่านั้น ต่อมาในแต่ละขั้นตอน ก็จะบวกเวอร์ทิกซ์ v ซึ่งมีระยะห่างจากจุดเริ่มต้นน้อยที่สุดเท่าที่จะน้อยได้ลงไป และในแต่ละขั้นของอัลกอริทึม จะใช้อะเรย์ D ในการเก็บค่าระยะทางที่สั้นที่สุดที่หาได้

```
Procedure Dijkstra;
{ Dijkstra computes the cost of the shortest paths from vertex 1 to every vertex of a directed graph }
begin
(1)     S : = {1};
(2)     For i : = 2 to n do
(3)             D[i] : = C[1,i];  {initialize D}
(4)     For i : = 1 to n-1 do begin
(5)             Choose a vertex w in V-S such that
                        D[w] is a minimum;
(6)             Add w to s
(7)             For each vertex v in V-S do
(8)                     D[v] : = min (D[v] , D[w] + C[w,v])
        End
End;{Dijkstra}
```

รูปที่2.8 โปรแกรม Dijkstra's Algorithm

จากอัลกอริทึมนี้ สมมติว่ามีกราฟ G = (V,E) ซึ่ง V = {1,2...,n} และเวอร์ทิกซ์ 1 คือ จุดเริ่มต้น C คืออะเรย์ของ cost โดยที่C[i,j] คือ cost ของการเดินทางจากเวอร์ทิกซ์ i ไปยังเวอร์ทิกซ์ j บนอาร์ซ i→jถ้าไม่มีอาร์ซจากเวอร์ทิกซ์ i ไปยังเวอร์ทิกซ์ j ดังนั้นจะอนุมานค่าของ C[i,j] เป็น ∞ และในทุกๆขั้นตอน D[i] จะเก็บความยาวเส้นทางที่น้อยที่สุดจากจุดเริ่มต้นไปยังเวอร์ทิกซ์ i ใดๆ แสดงตัวอย่างดังรูปที่2.5



รูปที่ 2.9 แสดงที่มีทิศทางที่มีค่าบนอาร์ซ

สมมติว่าใช้ Dijkstra's Algorithm กับรูปที่ 2.6 ในตอนเริ่มต้น S = {1},D[2] = 10,D[3] = ∞ , D[4] = 30 และ D[5] = 100 ในรอบแรกของ for-loop ในบรรทัดที่ (4)-(8) จะได้ว่า w = 2 คือเวอร์ทิกซ์ที่ถูกเลือกซึ่งมีค่า D ต่ำที่สุด ดังนั้นเซตให้ D[3] = min(∞,10+50) = 60, D[4] และ D[5] ไม่มีการเปลี่ยนแปลง เนื่องจากการเข้าหาเวอร์ทิกซ์ 4 และ 5 โดย

ตรงจากเวอร์ทิกซ์ 1 จะสั้นกว่าการเข้าหาผ่านทางเวอร์ทิกซ์ 2 ลำดับของค่าD หลังจากการเกิดการวนรอบของ for-loop จะแสดงดังตารางที่2.1

ตารางที่ 2.1 แสดงการคำนวณของ Dijkstra's algorithm ของกราฟรูปที่2.4

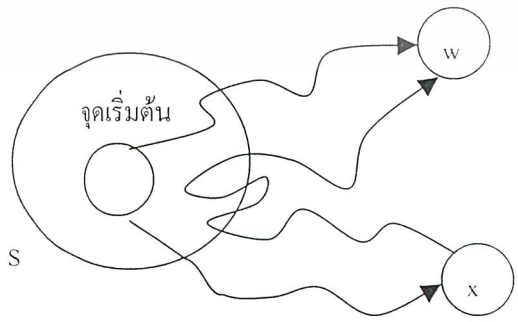
การวนซ้ำ	S	W	D[2]	D[3]	D[4]	D[5]
เริ่มต้น	{1}	-	10	∞	30	100
1	{1,2}	2	10	60	30	100
2	{1,2,4}	4	10	50	30	90
3	{1,2,4,3}	3	10	50	30	60
4	{1,2,4,3,5}	5	10	50	30	60

ในตอนเริ่มต้นจะเห็นได้ว่า D[3] =60 อันเนื่องมาจากในรอบที่1 ที่เริ่มวนรอบนั้น ค่าของเซต S จะมีเวอร์ทิกซ์ที่ถูกนำเข้ามาพิจารณาเพียง 2 เวอร์ทิกซ์ คือ S = {1,2} ดังนั้นระยะทางที่สั้นที่สุดจากเวอร์ทิกซ์ 1 ไปยังเวอร์ทิกซ์ 3 ที่สั้นที่สุดเป็นไปได้โดยผ่านทางเวอร์ทิกซ์ 2 ก็คือ 60 นั่นคือ D[3] = 60 และ D[5] ก็ยังคงเป็น 100 อยู่ เพราะไม่อาจหาเส้นทางที่สั้นกว่านี้ได้ถ้าให้ผ่านเวอร์ทิกซ์ที่ 2 เพียงตัวเดียว

ต่อมา จะเริ่มผนวกเอาเวอร์ทิกซ์ที่สั้นที่สุดจากเวอร์ทิกซ์เริ่มต้นเข้ามาอีก ทำให้ S = {1,2,4} อันเนื่องมาจากเวอร์ทิกซ์ 4 เป็นเวอร์ทิกซ์ ที่สั้นที่สุดจากกลุ่มเวอร์ทิกซ์ {1,2} เพราะว่าจากเวอร์ทิกซ์ 2 ไป 3 มีค่าเท่ากับ 50 ส่วนจากเวอร์ทิกซ์ 1 ไป5 มีค่าเท่ากับ100 ดังนั้นเวอร์ทิกซ์ ที่สั้นที่สุดจากกลุ่ม S = {1,2} ก็คือ4

จากการวนซ้ำครั้งที่ 2 เมื่อ S = {1,2,4} จะเห็นว่าค่า D[5] สามารถลดลงได้เป็น 90 อันเนื่องมาจากค่าที่สั้นที่สุดจากเวอร์ทิกซ์ 1 ไปยังเวอร์ทิกซ์ 5 โดยสามารถผ่านทางเวอร์ทิกซ์ 1,2 และ 4 นั้นก็คือ 1→4→5 ทำให้ได้ระยะทางรวมเป็น 90 เหตุการณ์จะเป็นเช่นนี้เรื่อยๆ จนกว่าจะหมดเวอร์ทิกซ์ที่จะนำมารวมในเซต S

Dijkstra's Algorithm ทำงานโดยหลักการที่ว่า การกระทำสิ่งไหนที่ดูเหมือนว่าเป็นสิ่งที่ดีที่สุดในส่วนย่อย ก็จะกลายเป็นการกระทำสิ่งที่ดีที่สุดในส่วนทั้งหมดในบั้นปลาย ในกรณีนี้ สิ่งที่ดีที่สุดที่จะทำก็คือ การหาระยะทางสั้นที่สุดไปยังเวอร์ทิกซ์ w ที่อยู่ภายนอกเซต S



รูปที่ 2.10 ข้อสมมติฐานของการหาระยะทางที่สั้นที่สุดไปยัง w

จากรูปที่2.6 จะเห็นว่าสามารถเดินทางไปยังเวอร์ทิกซ์ w ได้สองทาง ทางแรกจะไปโดยวิ่งจากจุดเริ่มต้น ไปยังเวอร์ทิกซ์ w เลย และทางที่สองจะเดินทางไปยังเวอร์ทิกซ์ x ก่อน จากนั้น(อาจจะ)เดินทางเข้า-ออกในเซต S หลายครั้งก่อนที่จะไปถึง w

ดูเหมือนว่าจะเป็นการยากที่จะหาระยะทางที่สั้นที่สุด โดยวิ่งจากจุดเริ่มต้นไปยังเวอร์ทิกซ์ x ก่อน (ซึ่งอยู่นอกเซต S เหมือนกันกับ w) แต่ถ้าเส้นทางนี้สั้นกว่าการเดินทางไปยังเวอร์ทิกซ์ w ในกรณีนี้เมื่อจะเลือก w ที่บรรทัดที่ (5) ในรูปที่2.5 จะเลือก x แทน เพราะว่า D[x] น้อยกว่า D[w]

เพื่อเป็นการพิสูจน์อย่างสมบูรณ์ว่าโปรแกรมรูปที่2.5 ทำงานได้จริง จะอนุมานเอาว่า D[v] เป็นระยะทางที่สั้นที่สุดไปยังเวอร์ทิกซ์ v ที่ขณะเวลาใดๆ ปัญหาของเรื่องนี้ก็คือ เมื่อเพิ่มเวอร์ทิกซ์ w เข้าไปยังเซต S ที่บรรทัด (6),(7) และ (8) จะทำการปรับค่า D เพื่อจดบันทึกความเป็นไปได้ที่ว่า มันอาจจะมีเส้นทางที่สั้นกว่าที่เดินทางไปยัง v โดยผ่าน w ถ้าเส้นทางนั้นเดินทาง old S ไปยัง w และพุ่งตรงไปยัง v ดังนั้น cost ของมัน(คือค่า D[w] + C[w,v]) จะถูกนำมาเปรียบเทียบกับ D[v] ที่บรรทัด (8) และค่า D[v] นี้จะถูกลดลงถ้าค่าใหม่นี้น้อยกว่าค่าเดิม เส้นทางที่สั้นที่สุดที่อาจเป็นไปได้อีกทางหนึ่งได้แสดงดังรูปที่2.7

รูปที่ 2.11 แสดงทางเดินที่สั้นที่สุดที่เป็นไปได้

เส้นทางที่ว่านี้จะเดินทางผ่านไปยัง w และย้อนมายัง old S เข้าไปยังสมาชิกบางตัวของ old S คือ เวอร์ทิกซ์ x หลังจากนั้นก็ไปยังเวอร์ทิกซ์ v

แต่จริงๆแล้วเส้นทางนี้จะไม่สามารถใช้ได้ เนื่องจาก x ถูกนำเข้ามา เซต S ก่อน w และเส้นทางที่สั้นที่สุดจากจุดเริ่มต้นมายัง x จะผ่านแต่ old S เท่านั้น ดังนั้นเส้นทางไปยัง x ผ่านทาง w ตามที่แสดงในรูปที่2.7จึงไม่อาจสั้นกว่าเส้นทางจาก x ผ่านในเซต S โดยตรง ผลก็คือ ความยาวของเส้นทางในรูปที่2.7 จากจุดเริ่มต้นไปยัง w,x และ v ไม่อาจสั้นกว่าค่าเดิมของ D[v] เนื่องจาก D[v] ไม่อาจมีค่ามากกว่าความยาวของทางเดินที่สั้นที่สุดที่ทางไปยัง x ผ่านเซต S และตรงไปยัง w ดังนั้น D[v] ไม่อาจถูกลดลงได้ที่บรรทัดที่ (8) โดยเดินทางผ่าน w และ x ดังรูปที่ 2.7

2.4 ปัญหาทางเดินที่สั้นที่สุดแบบหลายทาง

สมมติว่ามีกราฟแบบมีทิศทางที่กำหนดค่าของเวลาการเดินทางจากเมืองหนึ่งไปยังอีกเมืองหนึ่งมาให้ และต้องการสร้างตารางที่ให้ค่าเวลาที่สั้นที่สุดจากเมืองใดเมืองหนึ่งไปยังอีกเมืองใดเมืองหนึ่ง เพื่อที่จะกำหนดปัญหาได้อย่างชัดเจน สมมติว่ามีกราฟแบบมีทิศทาง G = (V,E) ซึ่งในแต่ละอาร์ซ $v \rightarrow w$ จะมีค่าที่ไม่เป็นลบของ cost C[v,w] ปัญหานี้ก็คือการหาแต่ละคู่ลำดับของเวอร์ทิกซ์ (v,w) ที่มีความยาวน้อยที่สุดทุกๆเส้นทางจาก v ไปยัง w

สามารถแก้ปัญหานี้โดยใช้ Dijkstra's Algorithm กับทุกๆเวอร์ทิกซ์ที่เป็นจุดเริ่มต้นก็ได้ แต่วิธีการที่ใช้แก้ปัญหานี้โดยตรงที่นิยมใช้ก็คือวิธีการของนาย R. W. Floyd เพื่อความสะดวก จะกำหนดหมายเลขเวอร์ทิกซ์ใน v ก็คือ 1,2,...,n อัลกอริทึมของนาย Floyd หรือ Floyd's Algorithm นี้จะใช้เมตริกซ์A ที่มีขนาด n×n ในการคำนวณหาระยะทางที่สั้นที่สุดเริ่มจากให้ A[i,j] = C[i,j] สำหรับทุกๆ $i \neq j$ ถ้าไม่มีอาร์ซจาก i ไปยัง j ให้ C[i,j] = ∞ และทุกๆสมาชิกในแนวทแยงของเมตริกซ์จะเป็น 0

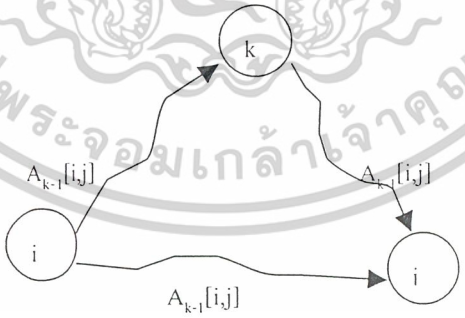
ดังนั้นสามารถทำการวนรอบเมตริกซ์ A เป็นจำนวน n ครั้ง และหลังจากทำไปเป็นครั้งที่ k^{th} A[i,j] จะมีค่าเป็นระยะทางน้อยที่สุดจากเวอร์ทิกซ์ i ไปยังเวอร์ทิกซ์ j โดยผ่านเวอร์ทิกซ์หมายเลขไม่มากไปกว่า k หรือพูดง่ายๆว่าในระหว่างเวอร์ทิกซ์ i ไปยังเวอร์ทิกซ์ j จะผ่านเวอร์ทิกซ์หมายเลขใดๆที่น้อยกว่าหรือเท่ากับ k

ในการวนรอบที่ k^{th} ใดๆ จะใช้สูตรต่อไปนี้คำนวณหาเมตริกซ์ A

$$A_k[i,j] = \min \begin{cases} A_{k-1}[i,j] \\ A_{k-1}[i,k] + A_{k-1}[k,j] \end{cases}$$

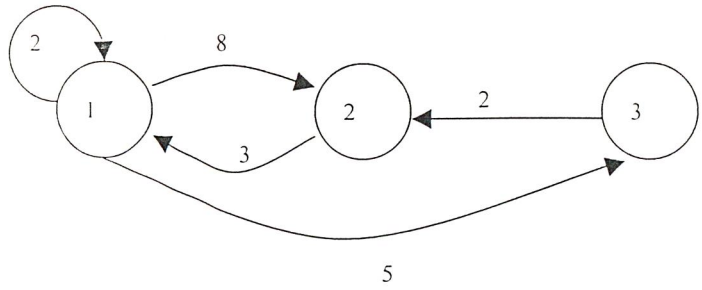
ตัวห้อย(subscript) k บ่งถึงค่าของเมตริกซ์ A หลังจากผ่านการวนรอบการกระทำครั้งที่ k^{th}

ในรูปที่ 2.8 จะใช้อธิบายให้เห็นถึงสูตรนี้ จากรูป ในการคำนวณหาค่า $A_k[i,j]$ เปรียบเทียบค่า $A_{k-1}[i,j]$ ซึ่งเป็นค่าของการเดินทางจาก i ไปยัง โดยไม่ผ่านเวอร์ทิกซ์หมายเลข k หรือมากกว่า กับค่าของ $A_{k-1}[i,j] + A_{k-1}[i,j]$ ซึ่งเป็นค่าของการเดินทางจาก i ไปยัง k และหลังจากนั้นจาก k ไปยัง j โดยไม่ผ่านเวอร์ทิกซ์หมายเลขมากกว่า k ถ้าการเดินทางผ่านเวอร์ทิกซ์ k นี้ให้ผลการเดินทางน้อยกว่าที่ได้จาก $A_{k-1}[i,j]$ ดังนั้นเลือกทางเดินนี้สำหรับ $A_k[i,j]$

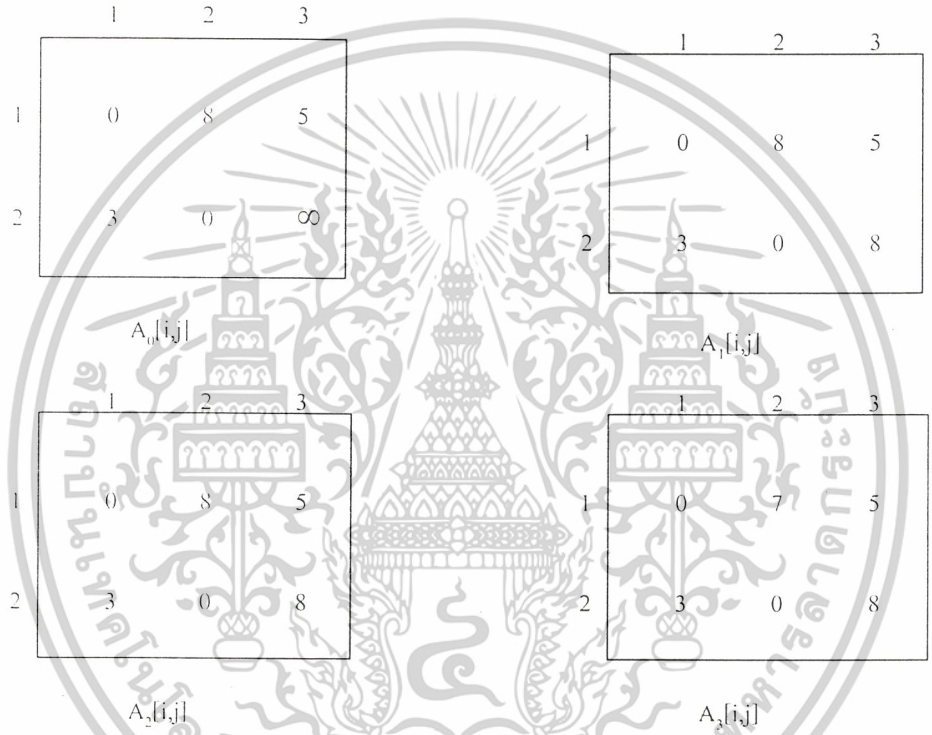


รูปที่2.12 แสดงให้เห็นการเดินทางจากเวอร์ทิกซ์ i ไปยังเวอร์ทิกซ์ j โดยมีเวอร์ทิกซ์ k ร่วมอยู่ด้วย

พิจารณากราฟแบบมีทิศทางตามรูปที่ 2.9 และค่าของเมตริกซ์ A หลังจากเริ่มต้นจนถึงการวนรอบครั้งที่ 3 ตามรูปที่ 2.10



รูปที่ 2.13 แสดงกราฟแบบมีทิศทางที่กำหนดค่าบนอาร์ซอันหนึ่ง



รูปที่ 2.14 แสดงค่าของเมตริกซ์ A ที่ได้รับการวนรอบ 3 ครั้ง

เนื่องจาก $A_k[i,k] = A_{k-1}[i,k]$ และ $A_k[k,j] = A_{k-1}[k,j]$ ไม่มีค่าใดหรือแม้กระทั่งค่าที่มีตัวห้อย(subscript) k มีการเปลี่ยนแปลงในระหว่างการวนรอบที่ k^{th} ดังนั้นสามารถทำการคำนวณเมตริกซ์ A เพียงแค่ชุดเดียว โปรแกรมที่ทำการคำนวณเมตริกซ์ $n \times n$ ที่ว่านี้จะแสดงในโปรแกรมที่ 2.2

```
Procedure Floyd(var A : array[1...n,1..n] of real;
                        C :array[1..n,1..n] of real);
{ Floyd computes shortest path matrix A given arc cost matrix C }
var
        i,j,k : integer
Begin
        For i : 1 to n do
                For j : 1 to n do
                        A[i,j] : = C[i,j];
                For i : 1 to n do
                        A[i,j] : = 0;
                For k : 1 to n do
                        For i : 1 to n do
                                For j : 1 to n do
                                        If A[i,k] + A[k,j] < A[i,j] then
                                                A[i,j] : = A[i,k] + A[k,j]
End; {Floyd}
```

รูปที่ 2.15 โปรแกรม Floyd's Algorithm

2.5 การค้นหาแบบฮิวริสติก (Heuristic search)

ข้อแตกต่างระหว่าง การค้นหาแบบธรรมดา กับ การค้นหาแบบฮิวริสติก คือ การค้นหาแบบธรรมดา จะค้นหาข้อมูลทีละตัวจนครบ แต่การค้นหาแบบฮิวริสติก จะไม่ลงไปค้นหาทุกๆข้อมูล จะเลือกคำตอบเฉพาะที่เหมาะสมให้กับการค้นหา ซึ่งมีข้อดีคือ สามารถค้นหาคำตอบจากข้อมูลที่มีขนาดใหญ่มากๆ ได้โดยใช้เวลาน้อยกว่า แต่ข้อเสียก็คือ คำตอบที่ได้เป็นคำตอบที่ดีเท่านั้น ไม่แน่ใจว่าดีที่สุดหรือไม่ ซึ่งจะต้องมี ฮิวริสติกฟังก์ชัน(Heuristic function) คือ ฟังก์ชันที่ทำหน้าที่วัดขนาดของความเป็นไปได้ในการแก้ ปัญหา ซึ่งจะแสดงด้วยตัวเลขมาช่วยในการค้นหาคำตอบ

วิธีการดังกล่าวกระทำโดยการพิจารณาถึงวิธีการ (Aspects) วิธีต่างที่ใช้แก้ปัญหา ณ สถานะหนึ่งว่าจะสามารถแก้ปัญหาได้ตามต้องการหรือไม่ โดยกำหนดเป็นน้ำหนักให้กับการแก้ปัญหาของแต่ละวิธี น้ำหนักเหล่านี้จะถูกแสดงด้วยตัวเลขที่กำกับไว้ที่จุดตัดต่างๆในกระบวนการค้นหา และค่าเหล่านี้จะเป็นตัวเลขประมาณความเป็นไปได้ว่าเส้นทางที่ผ่านจุดตัดนั้นมีความเป็นไปได้มากน้อยขนาดใดที่จะเป็นหนทางนำไปสู่การแก้ปัญหาได้

จุดประสงค์ที่สำคัญของ ฮิวริสติกฟังก์ชัน คือ การกำกับทิศทางของขบวนการค้นหา เพื่อให้อยู่ในทิศทางที่ได้ประโยชน์มากที่สุด มีวิธีการต่างๆดังนี้

- การค้นหาข้อมูลแบบที่ง่ายที่สุด(Generate and test)
- การค้นหาแบบปีนตามเนินเขา(Hill climbing)

- การค้นหาตามแนวลึก(Depth first search)
- การค้นหาตามแนวกว้างก่อน(Breadth first search)
- การค้นหาแบบทางเลือกที่ดีที่สุด(Best first search)

2.5.1 การค้นหาข้อมูลแบบที่ง่ายที่สุด(Generate and test)

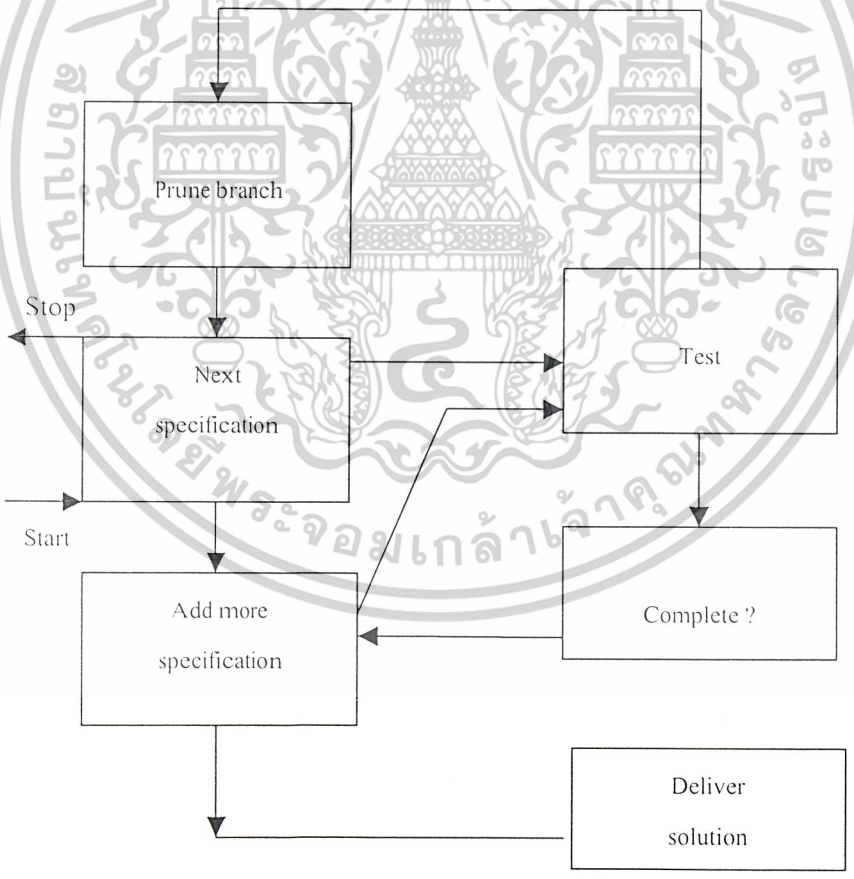
ขั้นตอนการค้นหาคำตอบประกอบด้วย

1. สร้างคำตอบที่เป็นไปได้ออกมาทั้งหมด อาจหมายถึง การสร้างจากสถานะเริ่มต้น ถ้าเป็นการเริ่มต้นของการค้นหาข้อมูล หรือ อาจหมายถึงการสร้างจากสถานะใดๆถ้าไม่ใช่การเริ่มต้น

2. ตรวจสอบคำตอบ ที่สร้างขึ้นมาว่าถูกต้องหรือไม่ โดยการเปรียบเทียบจุดตัด หรือ เส้นทางที่เลือกกับชุดของสถานะเป้าหมายที่ยอมรับได้

3. ถ้าหากว่าการตรวจสอบในข้อที่ 2 พบคำตอบแล้วเป็นอันว่าการค้นหาสิ้นสุด ค่าถ้าไม่พบ คำตอบให้ย้อนกลับไปทำข้อ 1 ใหม่

ในการสร้างคำตอบที่เป็นไปได้ ถ้าหากว่าได้มีการกระทำอย่างมีระบบ การค้นหาแบบนี้จะพบ คำตอบแน่นอน ถ้ามีคำตอบอยู่ แต่ถ้าหากว่าปัญหานี้กว้างมาก การค้นหาจะใช้เวลานาน เพราะว่า คำตอบที่เป็นไปได้ทุกอันจะต้องถูกสร้างขึ้นมาก่อนทำการทดสอบ



รูปที่ 2.16 แสดงGenerate and test

2.5.2 การค้นหาแบบปีนตามเนินเขา (Hill climbing)

เป็นวิธีที่ดัดแปลงมาจาก Generate and test แตกต่างกันที่วิธีการของ Generate and test ให้คำตอบของฟังก์ชันตรวจสอบเป็น ใช่ หรือ ไม่ใช่ เท่านั้น แต่วิธีของ การค้นหาแบบปีนตามเนินเขา ให้คำตอบออกมาว่าเป็นอะไรใกล้ที่สุดเพิ่มอีกด้วย จะประกอบด้วยขั้นตอนต่อไปนี้

1. สร้างคำตอบแรกขึ้นมา แล้วดูว่าเป็นคำตอบสุดท้ายใช่หรือไม่ ถ้าใช่เสร็จสิ้นวิธีการ ถ้าไม่ใช่ทำข้อต่อไป
2. จากคำตอบนี้ ให้ใช้กฎทั้งหลายมาหาคำตอบใหม่ออกมาแล้วให้เป็นคำตอบขั้นตอนแรกชุดใหม่ขึ้นมา
3. ในแต่ละ องค์ประกอบ ของข้อ 2 ให้ทำดังนี้
   - ตรวจสอบด้วยฟังก์ชันตรวจสอบ ถ้าเป็นคำตอบให้ยกเลิก
   - ถ้าไม่ใช่ ให้ตรวจสอบว่าเป็นตัวใดที่ใกล้ที่สุด

   เมื่อเทียบคำตอบกับค่าที่ทดสอบแล้วว่าเป็นคำตอบ ได้หรือไม่ ถ้าได้จำคำตอบไว้ ถ้าไม่ได้ให้ยกเลิก
4. ใช้องค์ประกอบที่ดีที่สุดจากข้อ 3 และให้เป็นคำตอบสมมติของตัวต่อไป ในขั้นตอนนี้จะสัมพันธ์กับการแก้ปัญหา ในห้วงปัญหาตามทิศทางที่ปรากฏ ซึ่งนำไปสู่คำตอบที่เร็วที่สุด
5. กลับไปทำขั้นตอนที่ 2

การเกิดปัญหาหลังจากใช้วิธีนี้ คือ

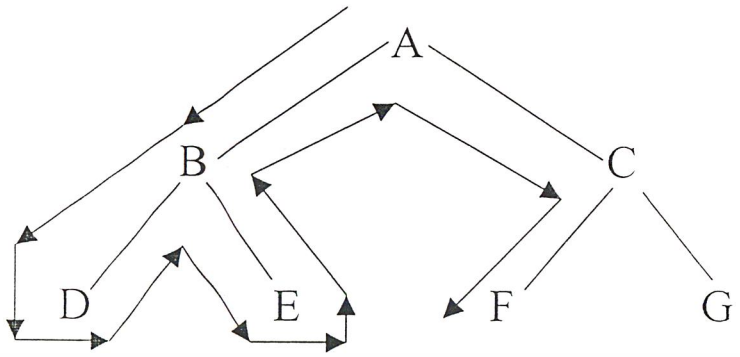
- Local maximum เป็นส่วนที่บอกสถานะที่ดีที่สุด เมื่อเทียบกับสถานะใกล้เคียงเท่านั้น แต่ถ้าเทียบกับสถานะอื่นๆที่อยู่ห่างออกไป หรือ การตรวจสอบขั้นต่อไปแล้ว ไม่แน่ใจว่าจะได้ผลออกมาดีที่สุด การแก้ปัญหาเช่นนี้ทำได้โดยอาศัยขบวนการย้อนรอย(Back tracking)
- Plateau ในกรณีปัญหาอยู่ในระดับเดียวกัน ผลของการหาค่าจาก ฮิวริสติกฟังก์ชันได้เท่ากันทั้งหมด จะไม่สามารถตัดสินใจว่าจะเลือกทางไหน แต่สามารถตัดสินใจว่าจะเลือกทางเส้นไหน และสามารถแก้ได้ด้วยวิธีก้าวกระโดด
- Ridge เป็นผลที่เกิดจากการทดสอบออกมาว่าทางเลือกที่กำลังทดสอบมีคะแนนมีคะแนนดีกว่าคะแนนทางเลือกในระดับถัดไปแสดงว่าจุดตัดที่สร้างขึ้นใหม่ควรจะเข้าใกล้คำตอบมาก

2.5.3 การค้นหาตามแนวลึก(Depth first search)

การค้นหาแบบตามแนวลึกก่อน หมายถึง การสำรวจเพื่อหาข้อสรุปทางเดินที่เป็นไปได้แต่ละทางที่ไปยังจุดหมายก่อนที่จะสำรวจเส้นทางอื่นต่อๆไป

ขั้นตอนการค้นหาจะประกอบด้วย

1. ถ้าสถานะเริ่มต้น(Initial state) เป็นสถานะจุดหมาย(Goal state)ก็ให้ออกจากโปรแกรมและผ่านค่าว่าทำงานสำเร็จแล้ว
2. นอกจากนั้นให้ทำตามขั้นตอนต่อไปนี้จนกว่าจะทำงานสำเร็จหรือล้มเหลว(Failure)
   - ให้สร้างตัวสืบทอด(Successor) ของสถานะเริ่มต้นเป็นXซึ่งถ้าไม่มีตัวสืบทอดอีกก็แสดงว่าการทำงานค้นหาล้มเหลว
   - ทำการค้นหาอีกครั้งโดยใช้Xเป็นสถานะเริ่มต้น
   - ถ้าทำงานสำเร็จก็ผ่านค่าว่าสำเร็จ ถ้าเป็นกรณีอื่นก็กลับไปที่Loop



รูปที่ 2.17 แสดงตัวอย่างทางเดินในการค้นหาแบบตามแนวลึกก่อน

จากรูปที่ 2.12 แสดงเส้นทางในการค้นหา จะเห็นว่าการค้นหาตามแนวลึกก่อนเป็นการเป็นหาที่จะพบจุดหมายแน่นอน (ถ้ามีจุดหมายอยู่ในเส้นทางใดเส้นทางหนึ่ง) แต่ทว่าอาจจะเป็นการค้นหาที่นานที่สุด ซึ่งจากรูปที่ 2.12 จะเกิดกรณีนี้ได้ถ้า G เป็นจุดหมายที่ต้องการ

2.5.4 การค้นหาตามแนวกว้างก่อน(Breadth first search)

ทุกจุดตัดที่อยู่ในระดับเดียวกันของต้นไม้จะถูกตรวจสอบก่อนจุดตัดที่อยู่ถัดไป วิธีการค้นหาโดยเริ่มจากการสร้างจุดตัดลูกให้กับสถานที่เริ่มต้นก่อน แล้วทำการตรวจสอบว่ามีจุดตัดใดที่เป็นเป้าหมายหรือไม่ ถ้าหากว่ามีก็เป็นอันว่าการค้นหาสิ้นสุด ถ้าไม่มีจากจุดตัดที่เป็นจุดตัดลูกจากตัวดังกล่าวข้างต้นให้สร้างจุดตัดลูกกับจุดตัดเหล่านั้น แล้วทำการตรวจสอบจุดตัดลูกทุกตัวของจุดตัดเหล่านั้น ถ้าพบเป้าหมายการค้นหาก็สิ้นสุด ถ้าไม่พบก็ให้ทำการสร้างจุดตัดเหล่านี้ต่อไปอีก ทำเช่นนี้ต่อไปจนกว่าจะค้นพบเป้าหมาย

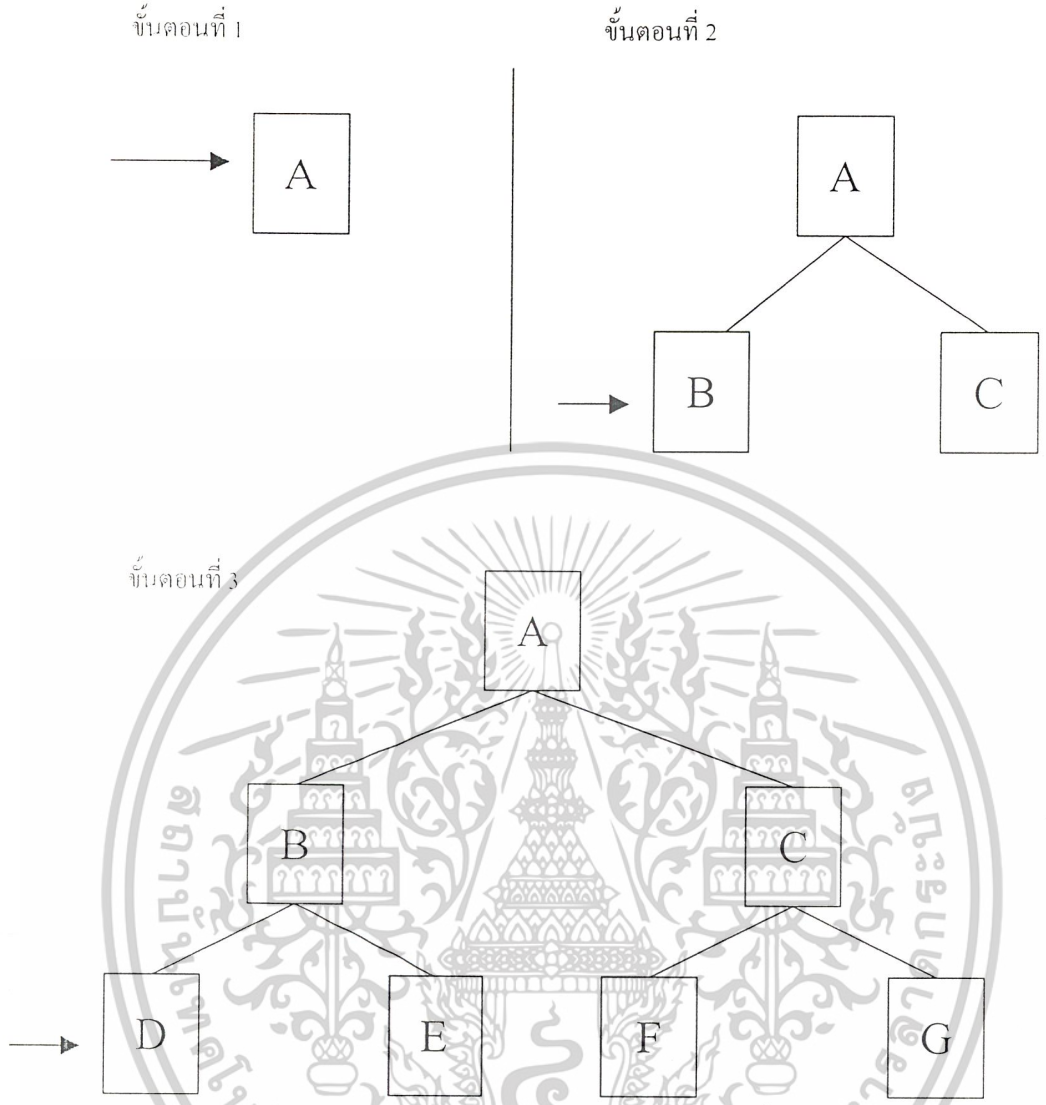
ขั้นตอนที่ 1 ตรวจสอบดูรูตโหนด A

ขั้นตอนที่ 2 สร้างจุดตัดลูกให้กับ A คือ B และ C ตรวจสอบ B และ C

ขั้นตอนที่ 3 สร้างจุดตัดลูกให้กับ B และ C คือ D,E,F และ G ทำการตรวจสอบ D,E,F และ G ตามลำดับ

ข้อเสียของการค้นหาแบบนี้ คือ

- ใช้หน่วยความจำมากเกินไป เพราะจำนวนของจุดตัด ให้แต่ละลำดับจะเพิ่มขึ้นแบบทวีคูณ
- การทำงานจะต้องมีขั้นตอนมาก
- การทำงานที่ไม่จำเป็นและการทำงานซ้ำซ้อนจะมีมากเกินไป



รูปที่ 2.18 แสดงการค้นหาแบบตามแนวกว้าง

2.5.5 การค้นหาแบบทางเลือกที่ดีที่สุด(Best first search)

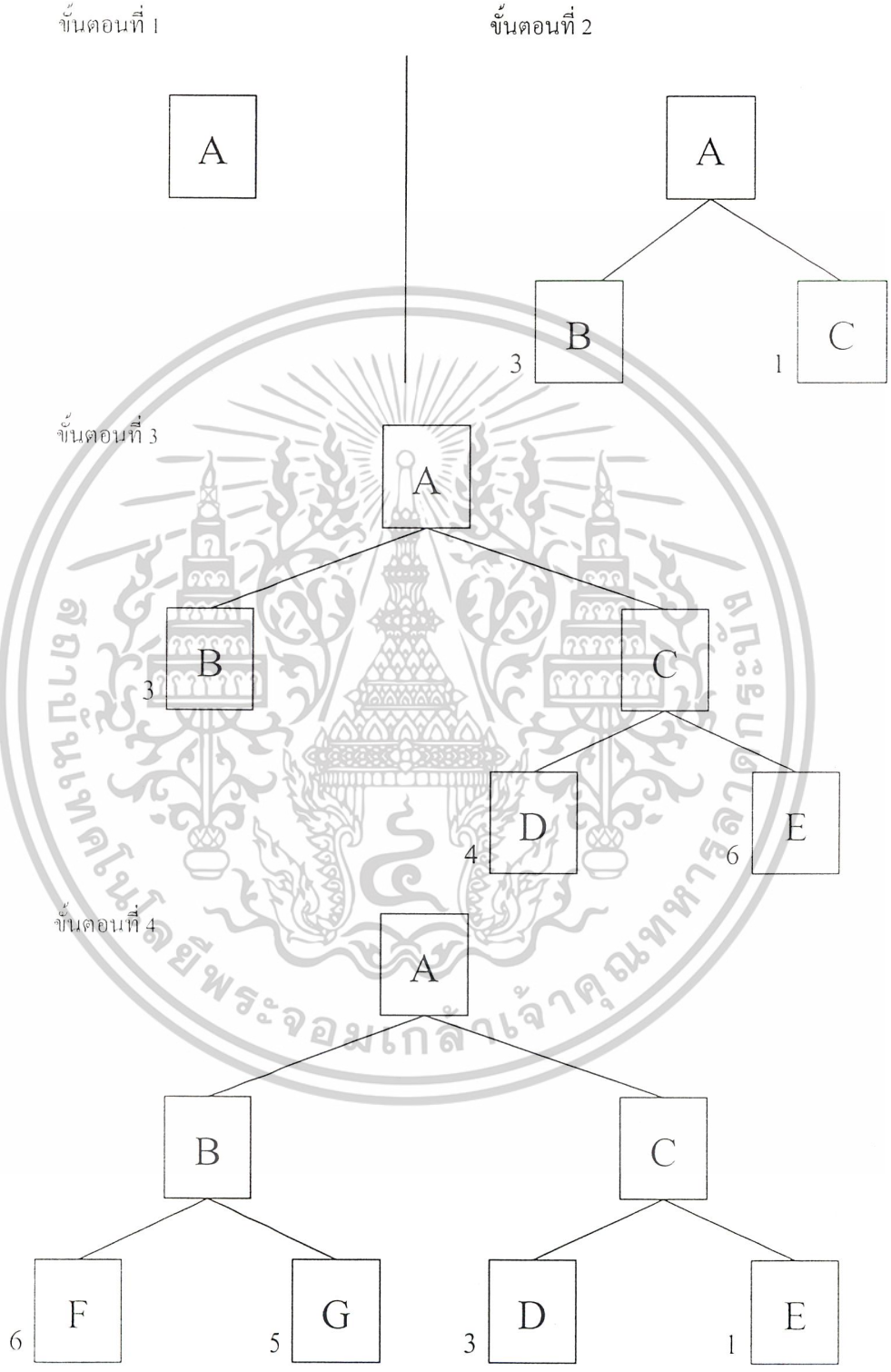
เป็นการรวบรวมเอาข้อดีทั้ง Depth first search และ Breadth first search มารวมกันเป็นวิธีการโดยที่แต่ละขั้นตอนของการค้นหาในจุดตัดลูกนั้น Best first search จะเลือกเอาจุดตัดที่ดีที่สุด (Most promising) กระบวนการในการเลือกจุดตัดที่ดีที่สุดนี้ สามารถทำได้โดยอาศัยฮิวริสติกฟังก์ชัน นี้จะให้ผลของการเลือกออกมาเป็นคะแนน ดังรูปที่ 2.14 ซึ่งแสดงตัวอย่างของการค้นหาแบบทางเลือกที่ดีที่สุด โดยมีขั้นตอนการค้นหาดังนี้

ขั้นตอนที่ 1 สร้างรูปจุดตัด

ขั้นตอนที่ 2 สร้างจุดตัดลูก B และ C แล้วตรวจสอบจุดตัด B และ C ด้วยฮิวริสติกฟังก์ชัน ได้ผลออกมาเป็นคะแนน 3 และ 1 เลือกจุดตัด C เป็นจุดตัดต่อไปที่สนใจ และสร้างจุดตัดลูกให้กับจุดตัด C

ขั้นตอนที่ 3 D และ E แล้วตรวจสอบคะแนน ได้ 4,6 เลือกจุดตัด B (เพราะมีคะแนนต่ำสุด) แล้วสร้างจุดตัดลูกตามลำดับ

ขั้นตอนที่ 4 F และ G ตรวจสอบคะแนนได้ 6 และ 5 ตามลำดับ ทำเช่นนี้เรื่อยๆจนพบคำตอบหรือจนไม่สามารถสร้างจุดตัดต่อไปได้อีก



รูปที่ 2.19 แสดงการค้นหาแบบทางเลือกที่ดีที่สุด

บทที่ 3

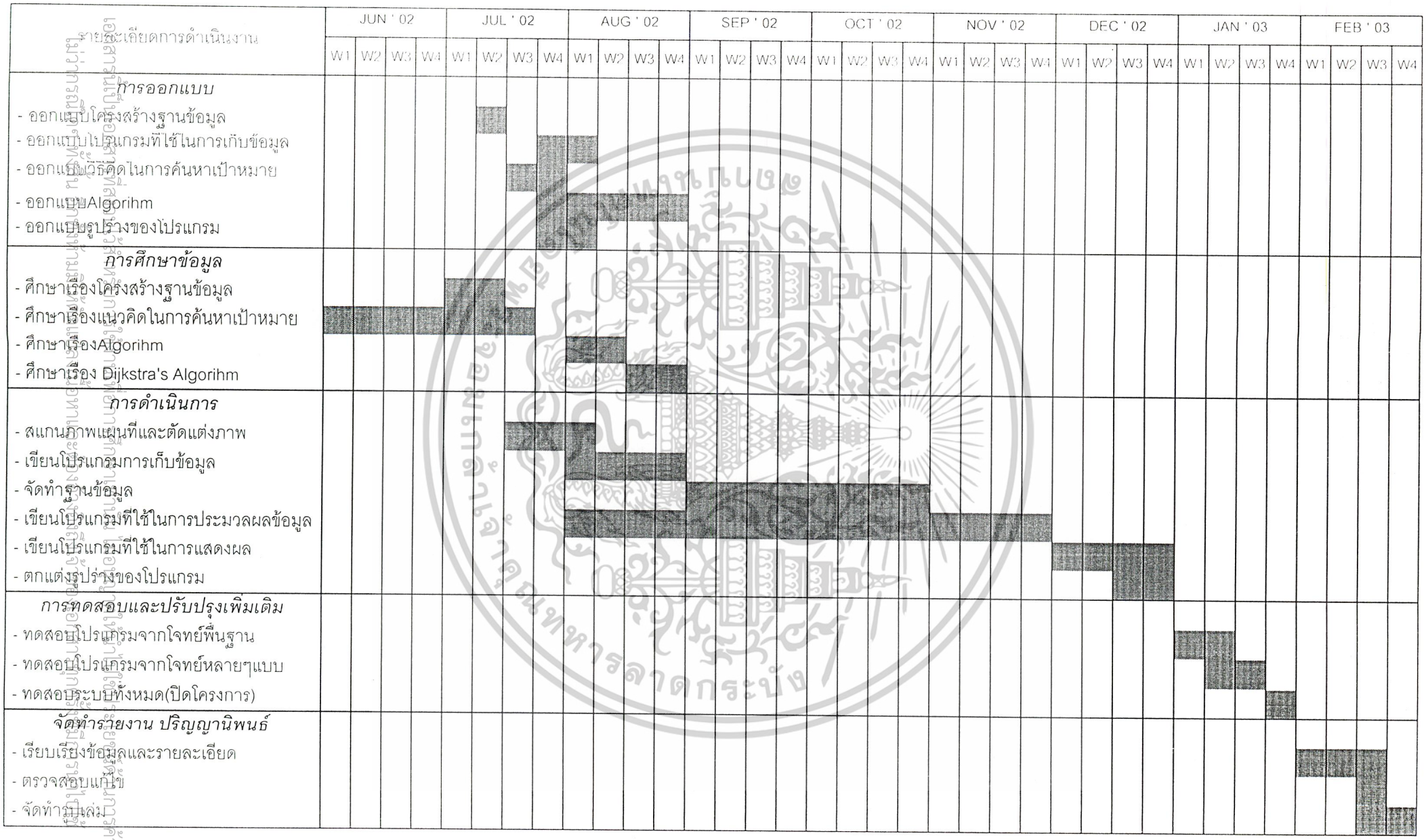
การออกแบบและการดำเนินงาน

3.1 การวางแผนการดำเนินงาน

มีลำดับขั้นตอนการดำเนินงานดังนี้

1. เก็บรวบรวมข้อมูล โดยในขั้นตอนนี้จะมีการศึกษาโครงสร้างฐานข้อมูล ทฤษฎีที่เกี่ยวกับ Shortest path และ ทำการศึกษาโปรแกรม Microsoft Visual Basic version 6.0
2. ทำการออกแบบส่วนต่างๆของโปรแกรม โดยมีการออกแบบส่วนติดต่อกับผู้ใช้เพื่อรับข้อมูล,ส่วนประมวลผล และส่วนแสดงผล
3. พัฒนาโปรแกรมในส่วนของการรับและจัดเก็บข้อมูล
4. พัฒนาโปรแกรมในส่วนของการประมวลผลข้อมูลเพื่อการค้นหาคำตอบที่ต้องการ
5. พัฒนาโปรแกรมในส่วนของการนำผลลัพธ์ที่ได้จากประมวลผลมาแสดงผลหรือนำเสนอแก่ผู้ใช้โปรแกรม
6. ทำการทดสอบความสามารถของโปรแกรมและค้นหาจุดบกพร่องที่ต้องทำการแก้ไขเพื่อให้โปรแกรมมีความสมบูรณ์
7. ทำการวิเคราะห์ ประมวลผล และแก้ไขปรับปรุงให้โปรแกรมเสร็จสมบูรณ์
8. รวบรวมข้อมูลทั้งหมดเพื่อสรุปผลและจัดทำรูปเล่มปริญญานิพนธ์และเตรียมการนำเสนอผลงาน
9. ปิดโครงงาน

รายละเอียดการดำเนินงาน สามารถแสดงได้ตามแผน Gantt chart ต่อไปนี้


รายละเอียดการดำเนินงาน
JUN '02
JUL '02
AUG '02
SEP '02
OCT '02
NOV '02
DEC '02
JAN '03
FEB '03
W1 W2 W3 W4
การออกแบบ
- ออกแบบโครงสร้างฐานข้อมูล
- ออกแบบโปรแกรมที่ใช้ในการเก็บข้อมูล
- ออกแบบวิธีคิดในการค้นหาเป้าหมาย
- ออกแบบAlgorihm
- ออกแบบรูปร่างของโปรแกรม
การศึกษาข้อมูล
- ศึกษาเรื่องโครงสร้างฐานข้อมูล
- ศึกษาเรื่องแนวคิดในการค้นหาเป้าหมาย
- ศึกษาเรื่องAlgorihm
- ศึกษาเรื่อง Dijkstra's Algorihm
การดำเนินการ
- สแกนภาพแผนที่และตัดแต่งภาพ
- เขียนโปรแกรมการเก็บข้อมูล
- จัดทำฐานข้อมูล
- เขียนโปรแกรมที่ใช้ในการประมวลผลข้อมูล
- เขียนโปรแกรมที่ใช้ในการแสดงผล
- ตกแต่งรูปร่างของโปรแกรม
การทดสอบและปรับปรุงเพิ่มเติม
- ทดสอบโปรแกรมจากโจทย์พื้นฐาน
- ทดสอบโปรแกรมจากโจทย์หลายๆแบบ
- ทดสอบระบบทั้งหมด(ปิดโครงการ)
จัดทำรายงาน ปริญญานิพนธ์
- เรียบเรียงข้อมูลและรายละเอียด
- ตรวจสอบแก้ไข
- จัดทำเป็นเล่ม

3.2 การออกแบบโปรแกรม

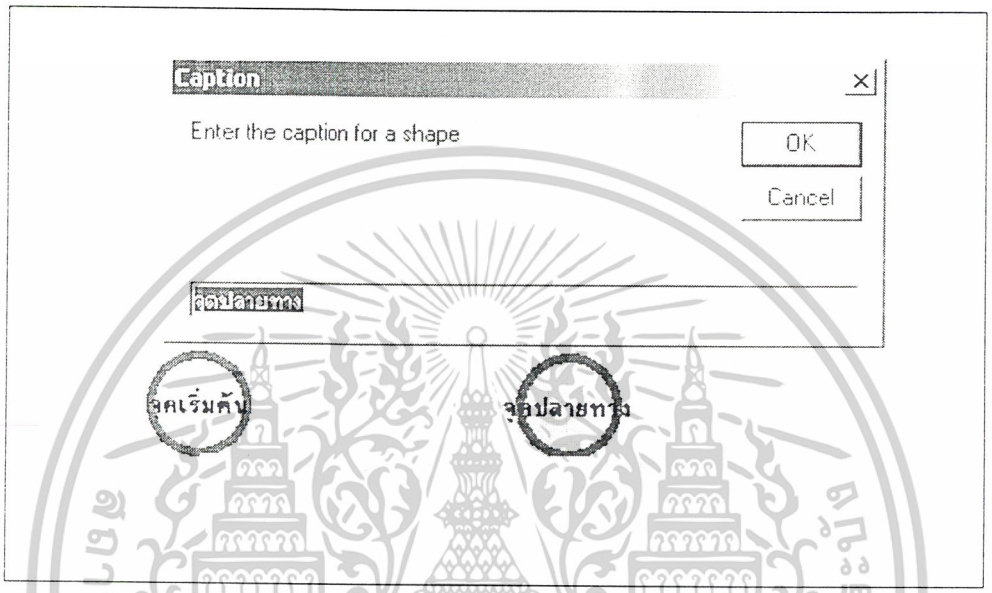
โปรแกรมประกอบด้วย 3 ส่วนหลัก ดังนี้

3.2.1 ส่วนรับข้อมูล (Input data)

โดยในส่วน Input ของโปรแกรมนั้นแบ่งได้ 2 กรณีดังต่อไปนี้

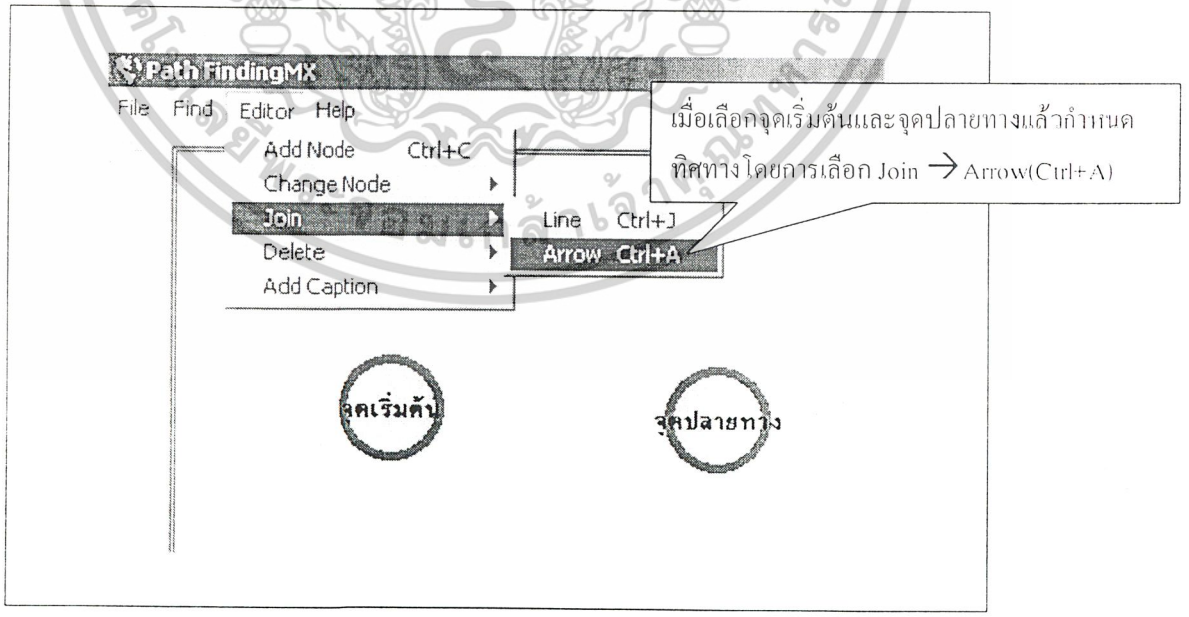
1. กรณีที่ต้องการสร้าง Data ของภาพแผนที่ใหม่จะมี Input ดังต่อไปนี้

1.1 ชื่อของ Node แต่ละตัว มีรูปแบบหน้าจอการรับค่าดังนี้

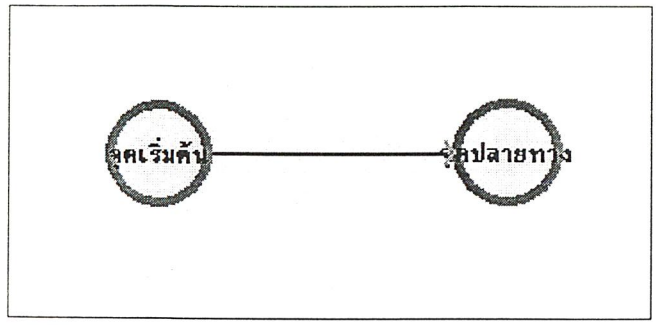


รูปที่ 3.1 แสดงการรับค่าของชื่อ Node

1.2 ทิศทางของเส้นทางแต่ละเส้นทาง มีรูปแบบหน้าจอการกำหนดทิศทาง ดังนี้

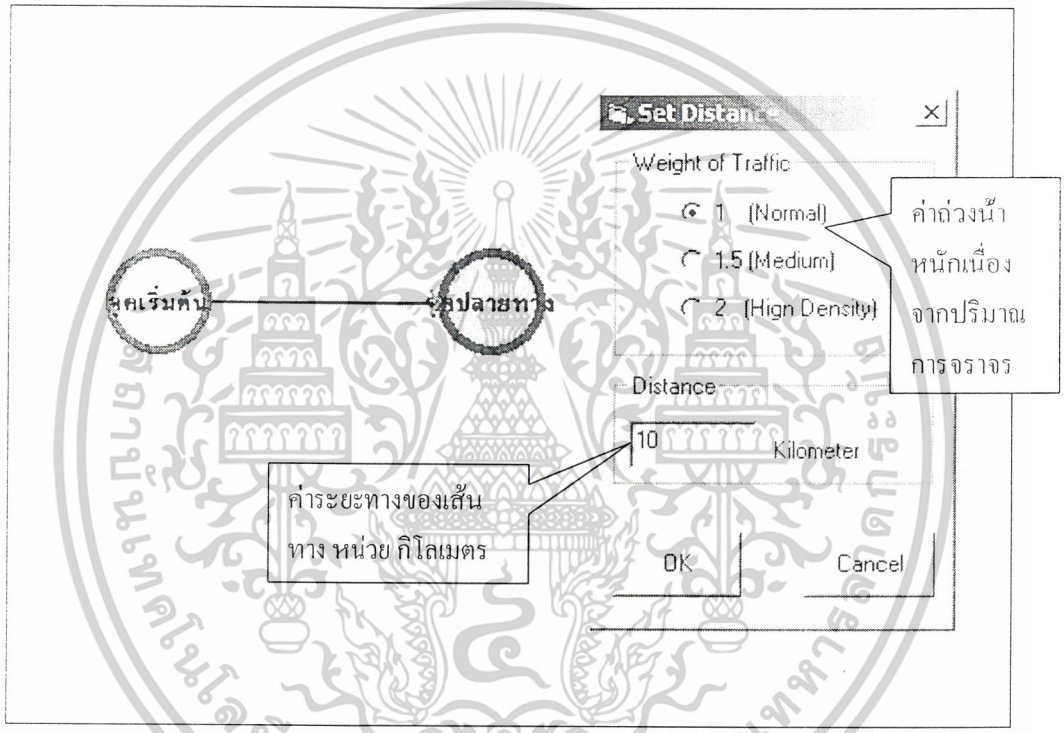


รูปที่ 3.2 แสดงการกำหนดทิศทางของโปรแกรม

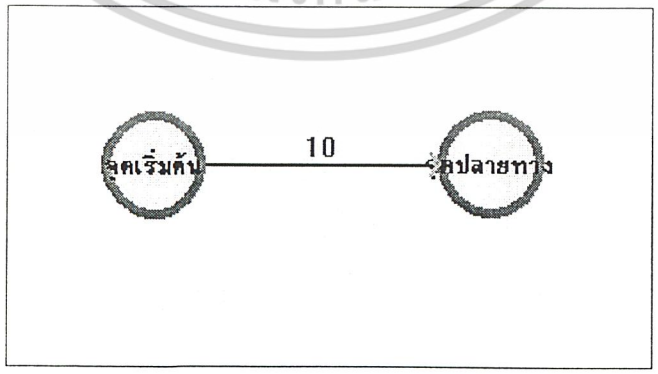


รูปที่ 3.3 แสดงการผลการกำหนดทิศทางของโปรแกรม

1.3 ระยะทางของแต่ละ Node มีรูปแบบหน้าจอการรับค่าดังนี้

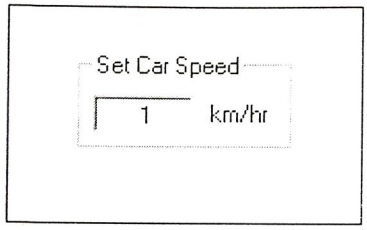


รูปที่ 3.4 แสดงการกำหนดระยะทางของโปรแกรม



รูปที่ 3.5 แสดงผลการกำหนดระยะทางของโปรแกรม

1.4 ความเร็วโดยประมาณของพาหนะที่ใช้ มีการรับค่าที่หน้าจอหลักดังนี้

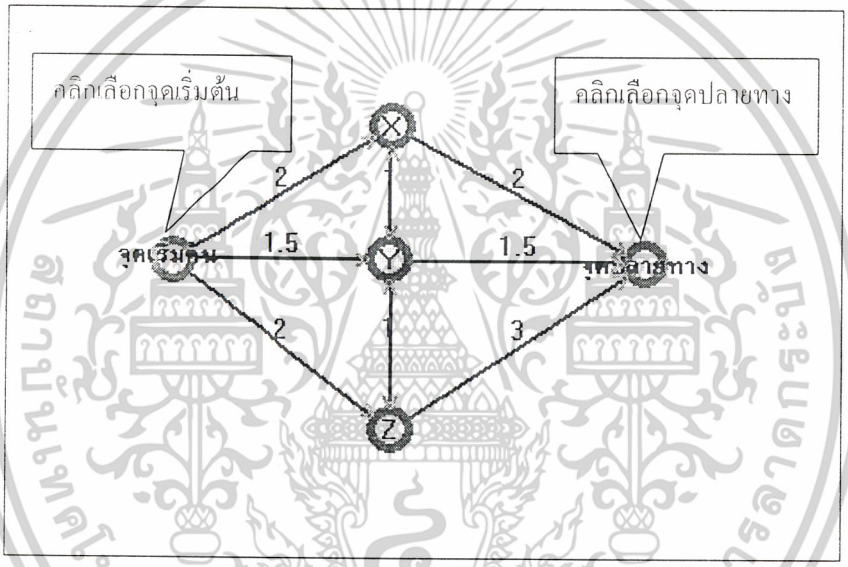


รูปที่ 3.6 แสดงการกำหนดความเร็วโดยประมาณของพาหนะของโปรแกรม

2. กรณีที่ต้องการค้นหาเส้นทางจะมี Input ดังต่อไปนี้

2.1 จุดเริ่มต้น

2.2 จุดปลายทาง

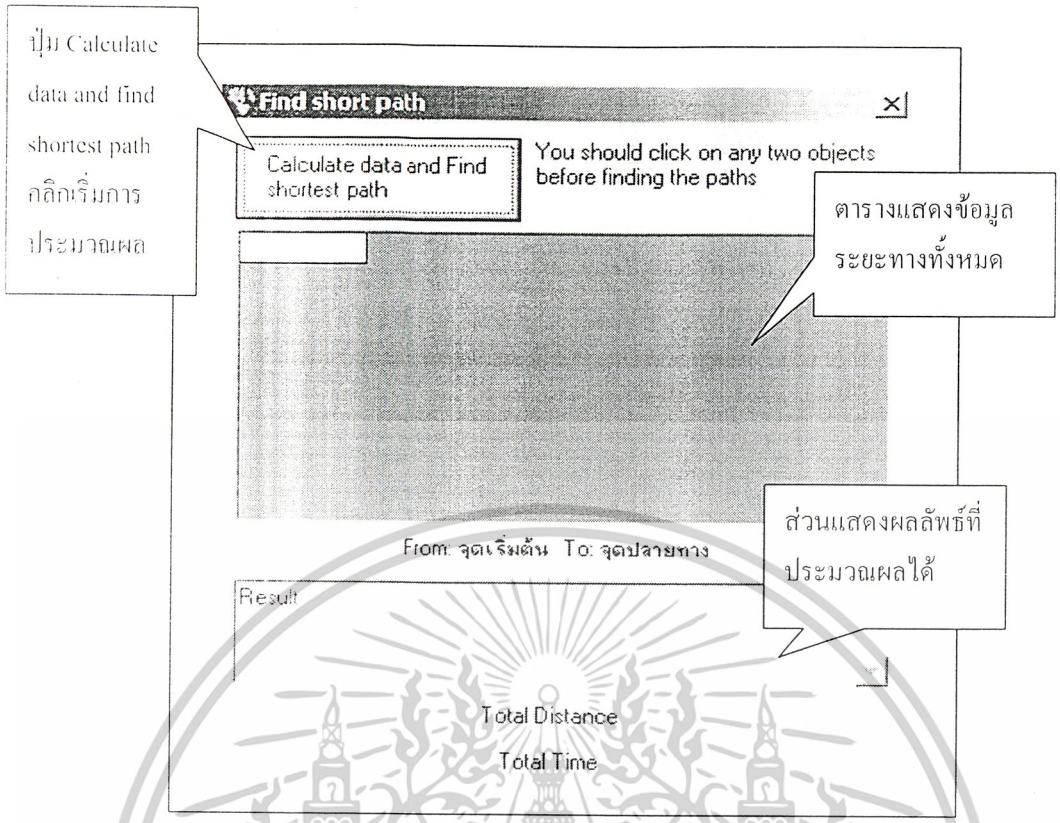


รูปที่ 3.7 แสดงตัวอย่างการสร้างโครงข่ายระยะทางของโปรแกรม

3.2.2 ส่วนประมวลผล (Process)

โปรแกรมจะนำค่าข้อมูลที่รับมาไปประมวลผลและแสดงค่า Output โดยอาศัยวิธีการตามทฤษฎี Shortest Path Algorithm ของ Dijkstra โดยที่หน้าจอจะประกอบไปด้วย

1. ปุ่ม Calculate data and find shortest path
2. ตารางแสดงข้อมูลระยะทางทั้งหมด
3. ส่วนแสดงผลลัพธ์ที่ประมวณผลได้ซึ่งประกอบด้วย
   - ระยะทางรวมทั้งหมด
   - เวลาโดยประมาณในการเดินทาง
   - ลำดับเส้นทางที่หาได้

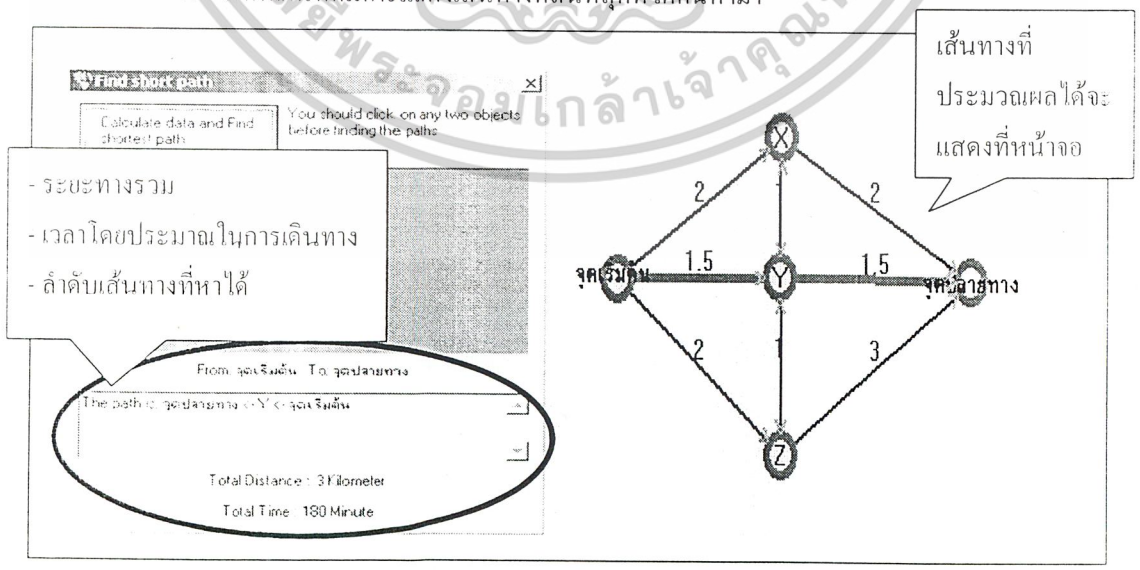


รูปที่ 3.8 แสดงหน้าจอส่วนประมวณผลของโปรแกรม

3.2.3 ส่วนแสดงผล (Output data)

โดยในส่วน Output ของโปรแกรมนั้นมีดังต่อไปนี้

1. ระยะทางรวมทั้งหมด
2. เวลาโดยประมาณในการเดินทาง
3. ลำดับเส้นทางที่หาได้
4. ภาพแผนที่ที่มีการแสดงเส้นทางที่สั้นที่สุดที่ได้ค้นหามา

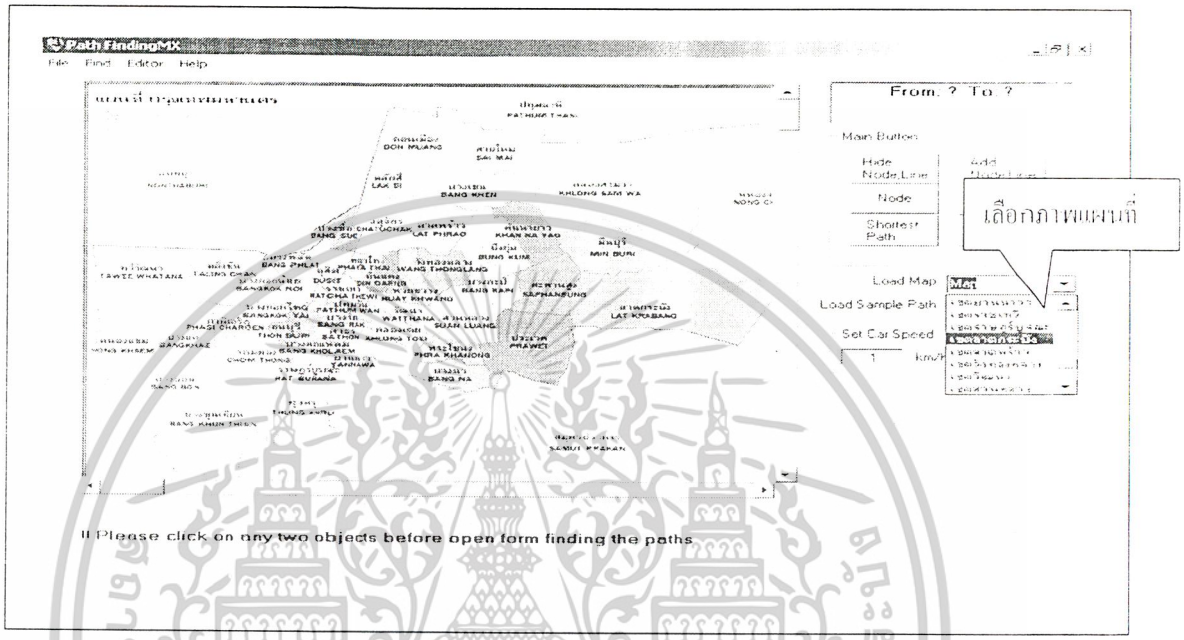


รูปที่ 3.9 แสดงผลลัพธ์ของส่วนประมวณผลของโปรแกรม

3.3 รายละเอียดการออกแบบรูปแบบขั้นตอนการทำงานของโปรแกรม

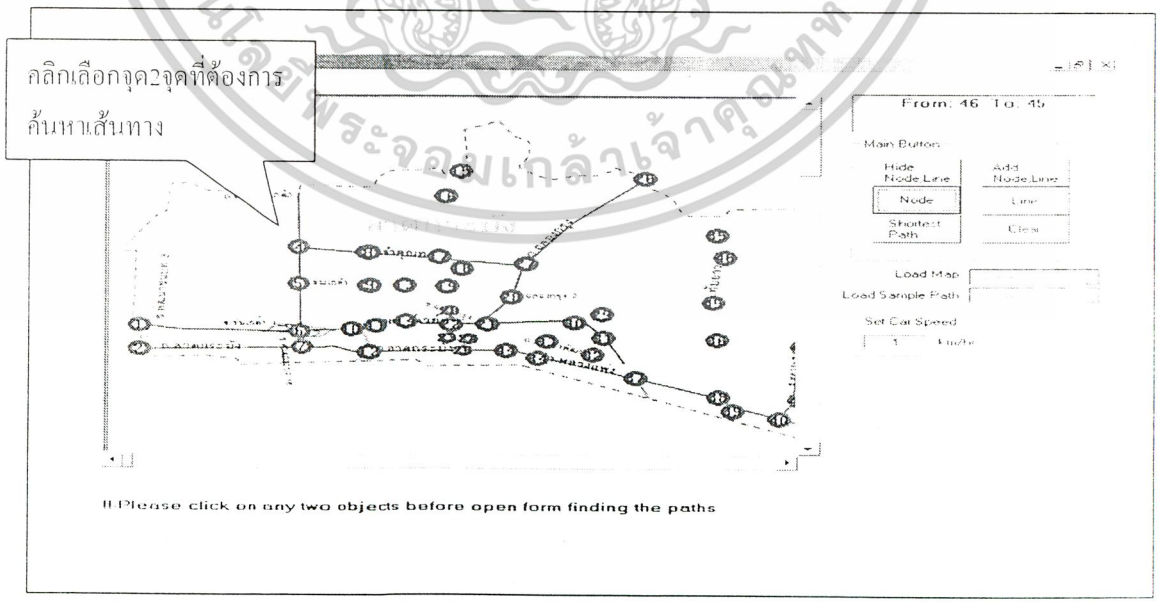
รายละเอียดการออกแบบขั้นตอนการทำงานของโปรแกรมหรือการออกแบบขั้นตอนการใช้โปรแกรมสามารถแบ่งได้ดังนี้

1. เริ่มต้นเลือกภาพแผนที่ในบริเวณเส้นทางที่ต้องการค้นหา ซึ่งมี 2 วิธี คือ การเลือกแผนที่ที่มีอยู่แล้วโดยจะมีตัวเลือกให้บนหน้าจอ หรือ เลือกแผนที่ขึ้นมาเองจาก Combo box ดังรูป



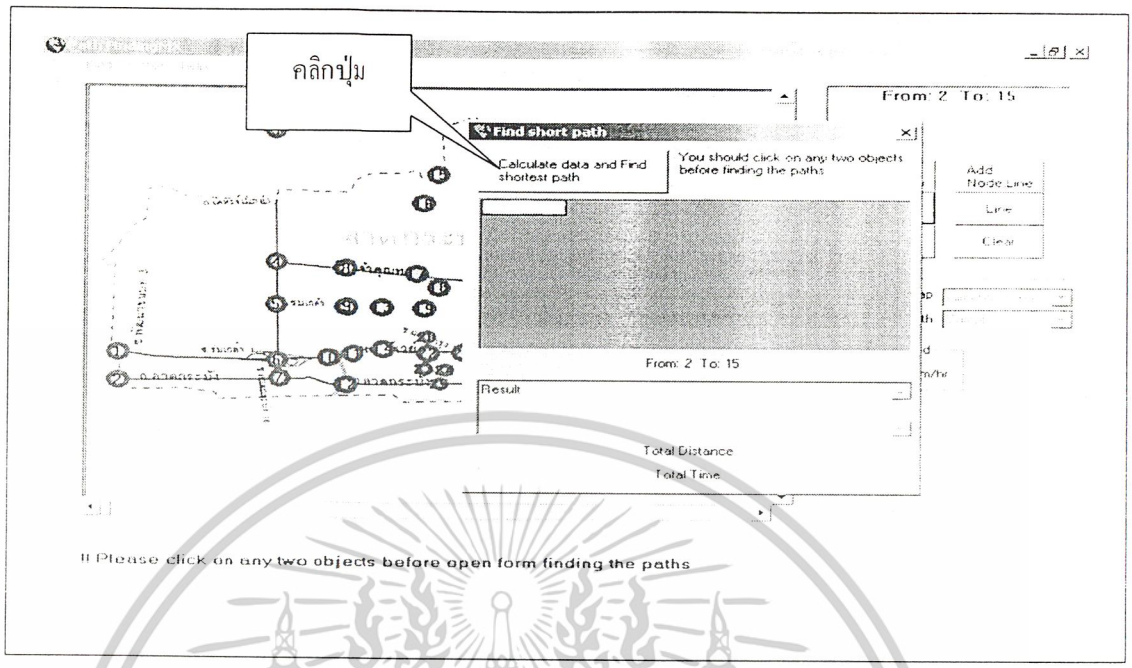
รูปที่ 3.10 แสดงการเลือกแผนที่จากหน้าจอหลัก

2. เลือกจุดเริ่มต้นและจุดปลายทาง โดยการคลิกเมาส์ที่ node บนแผนที่ ซึ่งจะแสดงชื่อของจุดที่ทำการเลือกบนมุมบนขวาของหน้าจอ ดังรูป



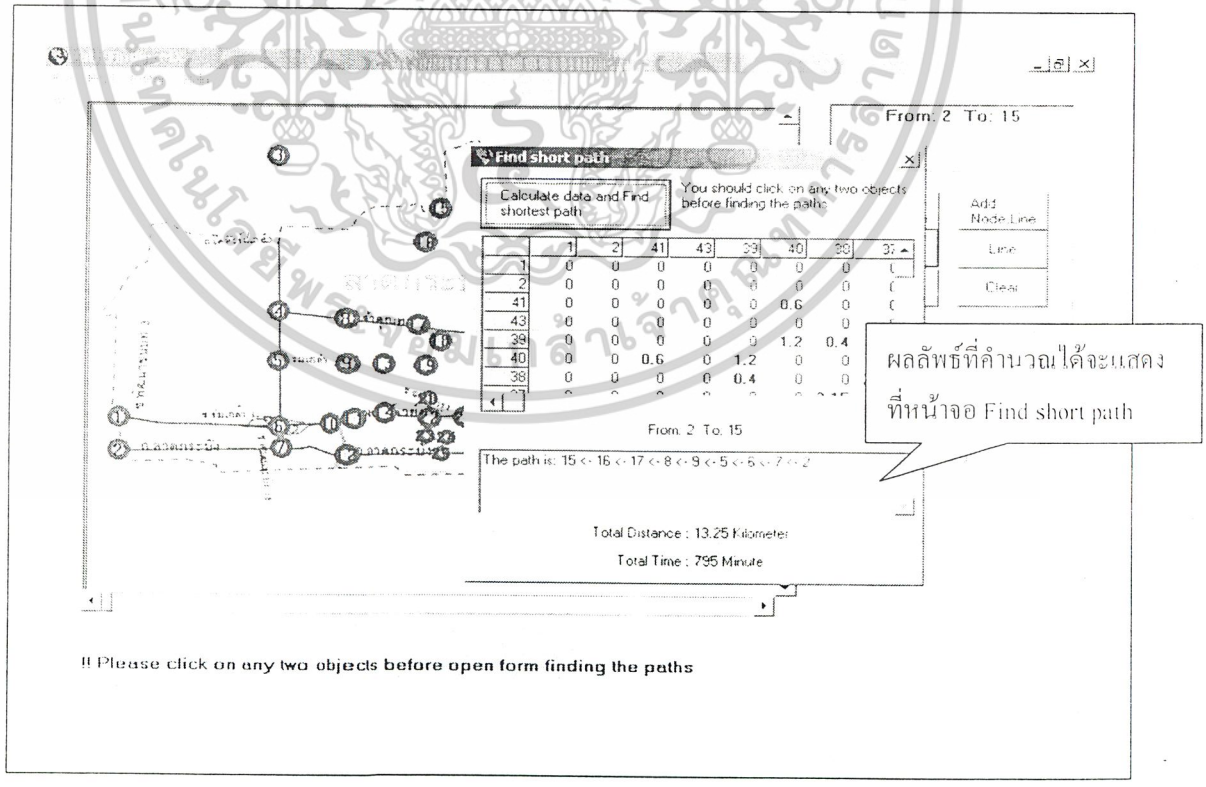
รูปที่ 3.11 แสดงการกำหนดจุดเริ่มต้นและจุดปลายทาง

3. เริ่มการค้นหาเส้นทาง โดยเปิดหน้าจอ find short path แล้วคลิกปุ่มCalculate data and find shortest path



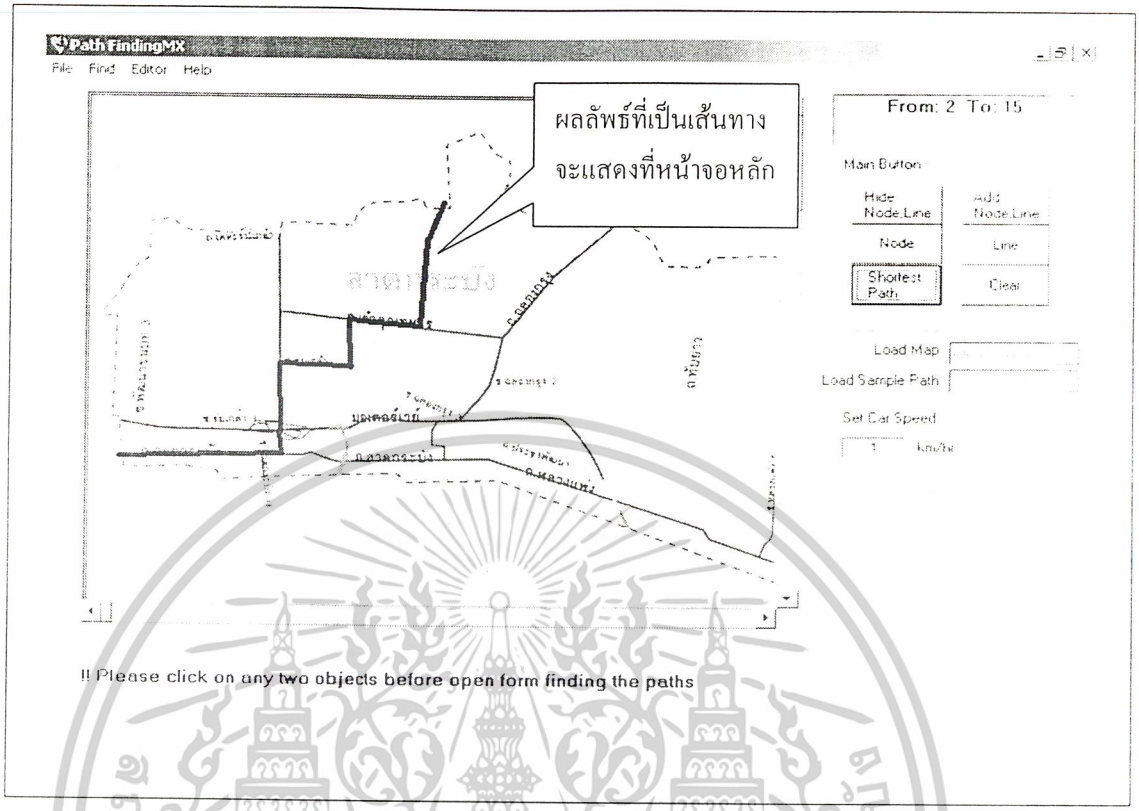
รูปที่ 3.12 แสดงหน้าจอ find short path

4. คำนวณหาข้อมูลความสัมพันธ์ของแต่ละ node โดยแสดงเป็นตารางเมตริกซ์ และ คำนวณหาเส้นทางที่สั้นที่สุดโดยใช้วิธีของ Dijkstra's Algorithm โดยจะได้ผลลัพธ์แล้วแสดงทางหน้าจอดังนี้



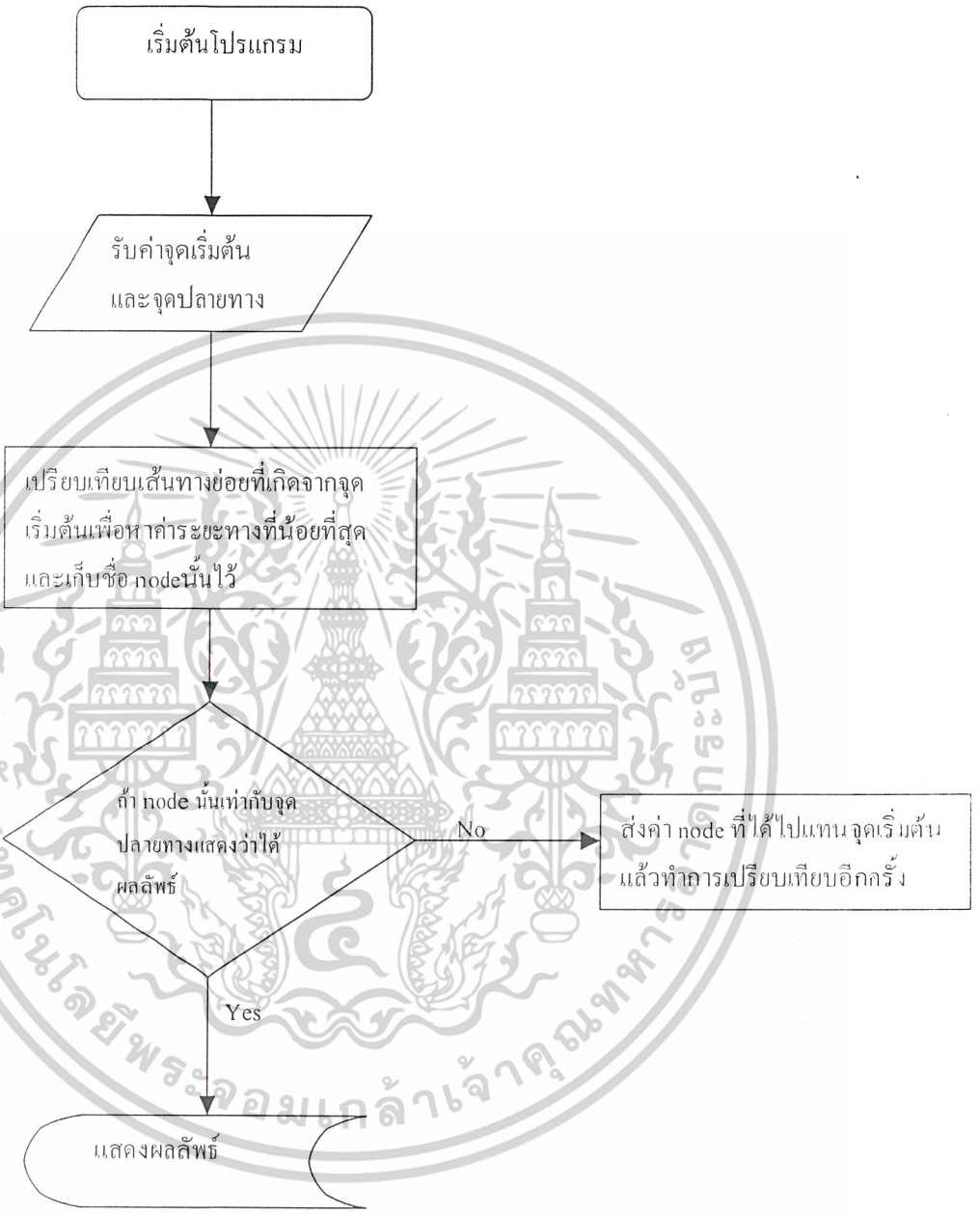
รูปที่ 3.13 แสดงการคำนวณเมตริกซ์ประชิด และ แสดงผลลัพธ์ที่คำนวณได้

5. ส่วนแสดงผลที่เป็นเส้นทางจะแสดงที่หน้าจอหลัก ดังรูป



รูปที่ 3.14 แสดงผลลัพธ์เส้นทางที่สั้นที่สุด

รูปที่3.15 แสดงขั้นตอนการทำงานโดยรวมของโปรแกรม



รูปที่3.16 แสดงขั้นตอนในการประมวลผลจากทฤษฎีของ Dijkstra's algorithm

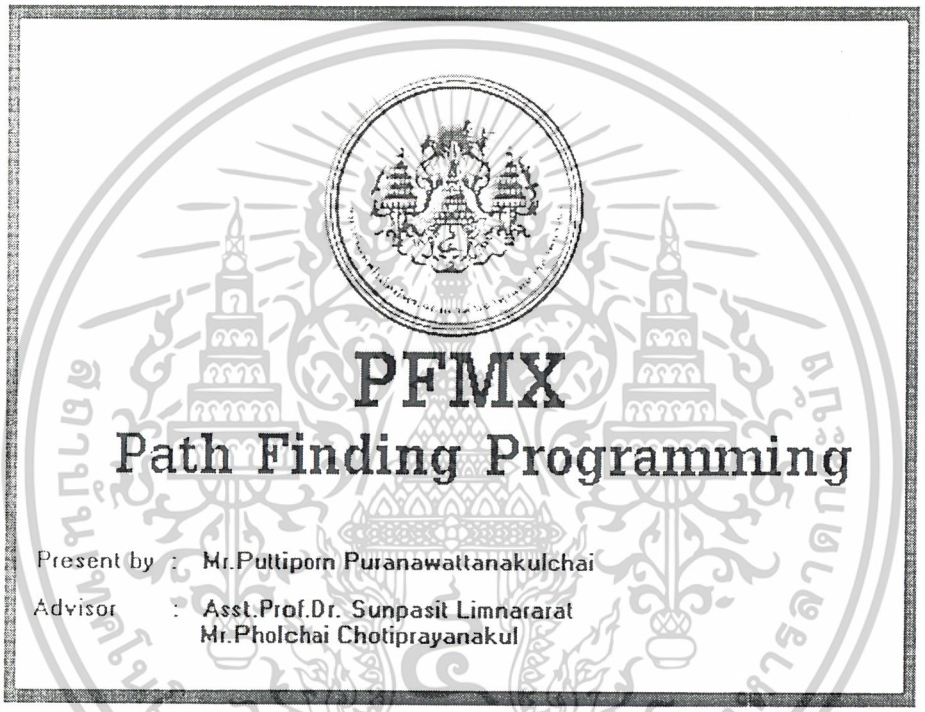
บทที่ 4

ผลการดำเนินงาน

การทดสอบและประเมินผลโครงงานการวิเคราะห์เส้นทางในเขตกรุงเทพมหานคร เพื่อหาระยะทางที่สั้นที่สุดในการเดินทาง โดยใช้ Microsoft Visual Basic 6.0 สามารถประเมินผลได้ดังนี้

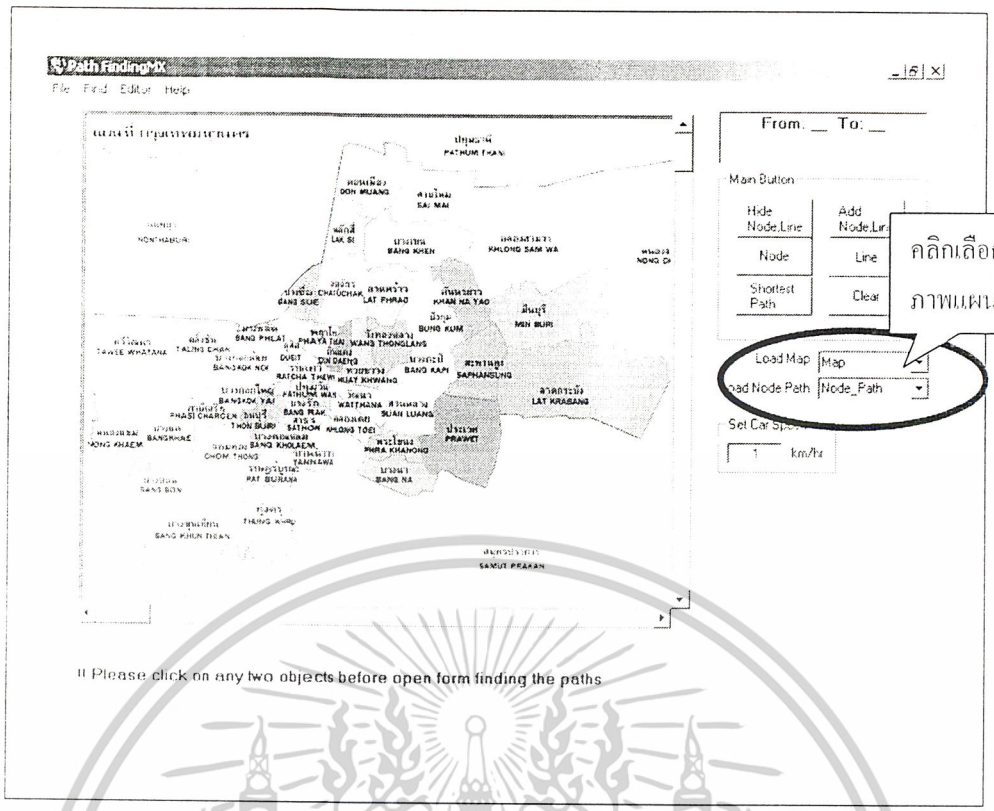
4.1 การวิเคราะห์ข้อมูลจากแผนที่กรุงเทพมหานคร

1. เมื่อเรียกโปรแกรมเริ่มแรกจะปรากฏหน้าจอดังนี้



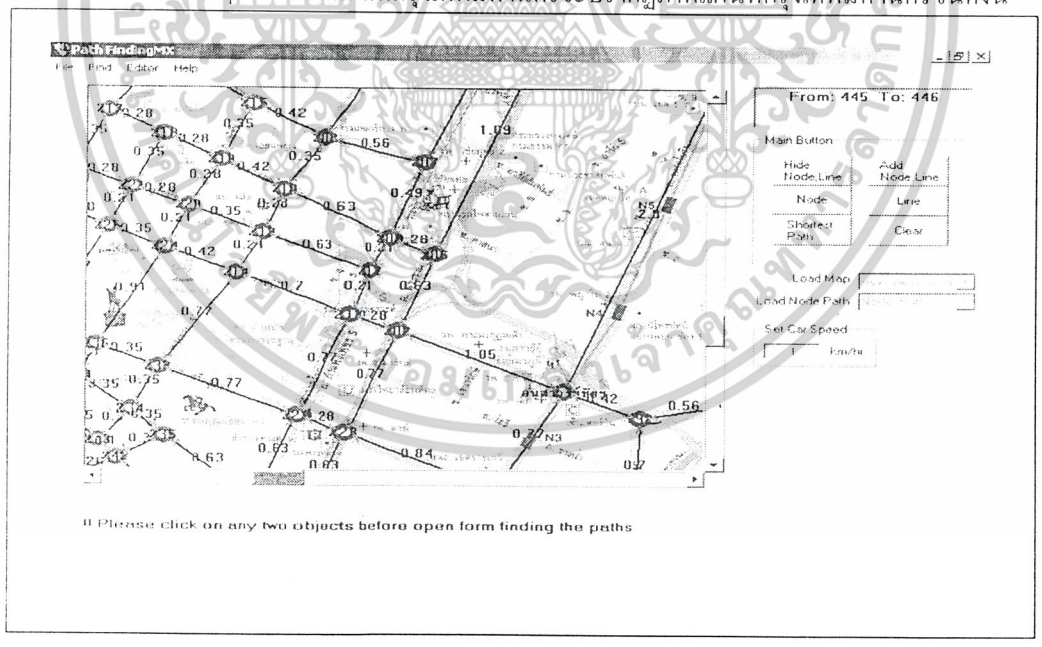
รูปที่ 4.1 แสดงฟอร์มเริ่มต้นโปรแกรม

2. หลังจากนั้นจะปรากฏหน้าจอหลักของโปรแกรมดังนี้



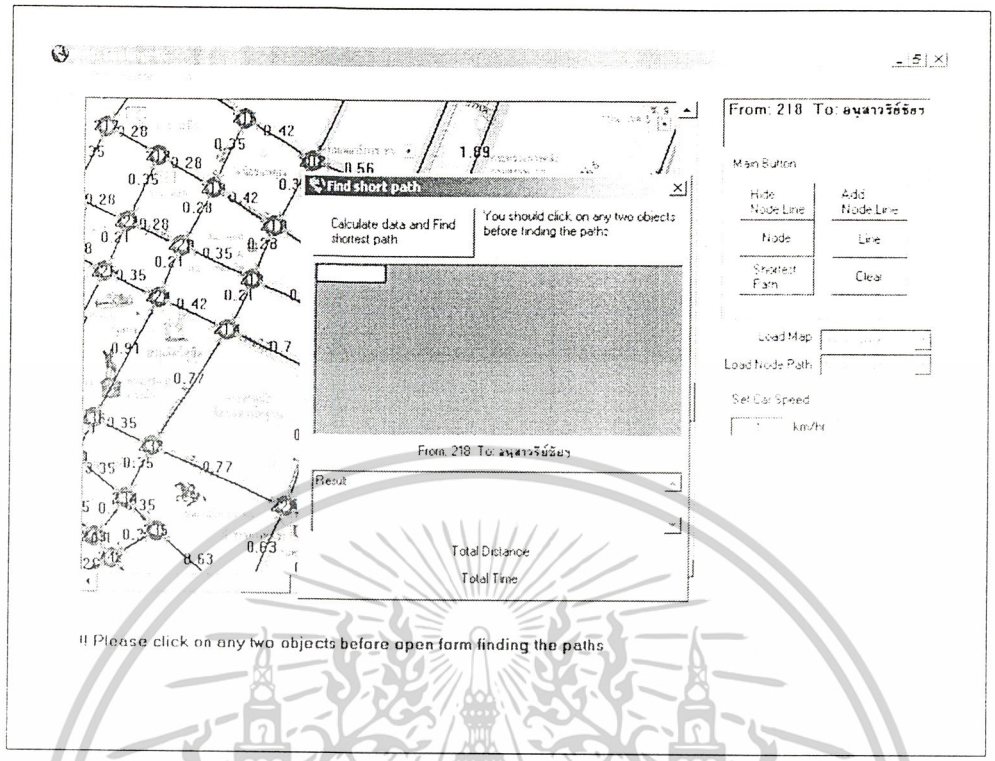
รูปที่ 4.2 แสดงฟอร์มหลักของโปรแกรม

3. เมื่อคลิกที่ Load Map เลือกภาพแผนที่กรุงเทพมหานครจะปรากฏภาพแผนที่กรุงเทพมหานครขึ้นดังนี้



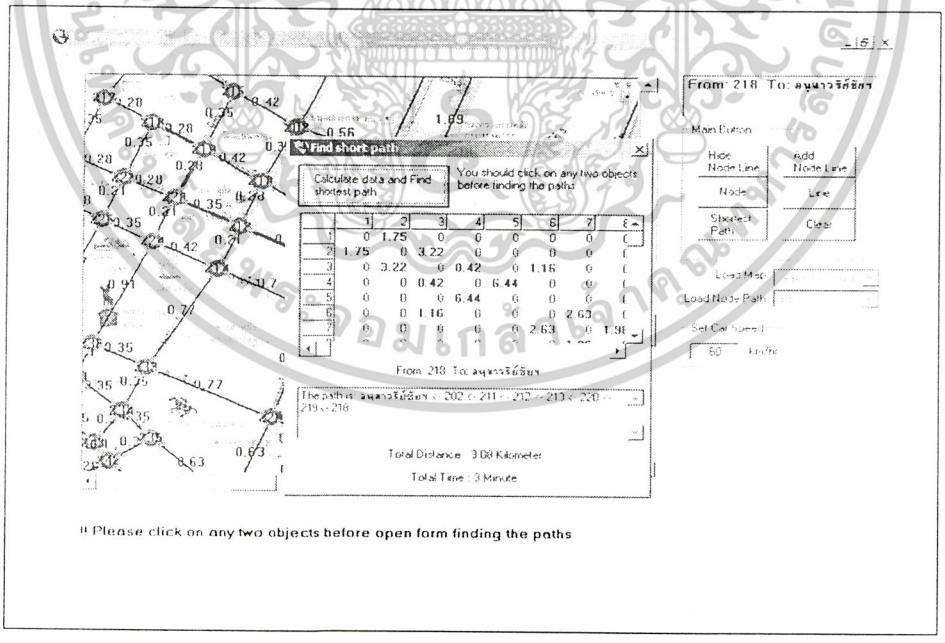
รูปที่ 4.3 แสดงภาพแผนที่กรุงเทพมหานครที่ใช้

4. เมื่อคลิกเลือกโหนดตัวอย่าง 2 โหนด ซึ่งจากการทดลองเลือก โหนดที่ 218 และ โหนด อนุสาวรีย์ชัยฯ และป้อนความเร็วของรถยนต์ที่ใช้โดยประมาณที่ 60 กิโลเมตร ต่อ ชั่วโมง จากนั้นเปิดฟอร์ม Find Short Path ขึ้นเพื่อทำการหาเส้นทางที่สั้นที่สุดดังรูป



รูปที่ 4.4 แสดงฟอร์มการค้นหาเส้นทางที่สั้นที่สุดของโปรแกรม

5. เมื่อคลิกที่ Calculate data and Find shortest path จะได้ผลลัพธ์ดังนี้



รูปที่ 4.5 แสดงผลลัพธ์ของการค้นหาเส้นทางที่สั้นที่สุดจากโปรแกรม

6. ซึ่งผลลัพธ์ที่ได้จากการค้นหาเส้นทางที่สั้นที่สุดจากโปรแกรมจะได้ผลลัพธ์ดังนี้

Setting

Form : 218

To : อนุสาวรีย์ชัยฯ

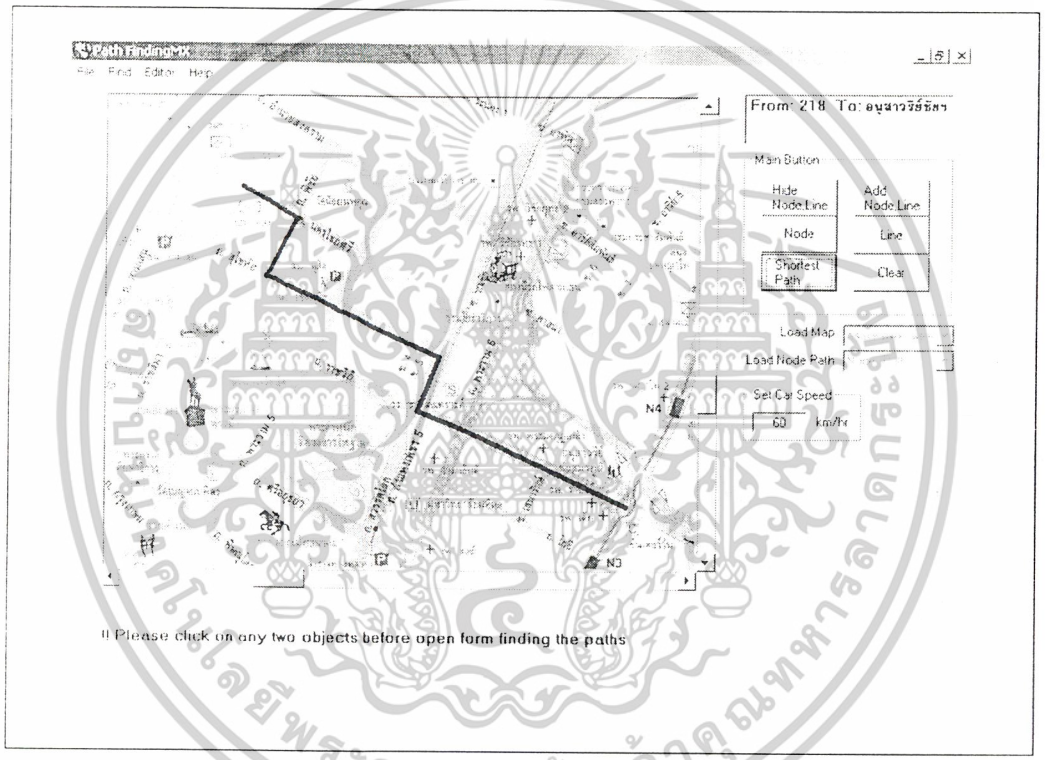
Set Car Speed : 60 km/hr

Result

The path is : Shortest path is : 218-->219-->220-->213-->212-->211-->202-->อนุสาวรีย์ชัยฯ

Total Distance : 3.08 Kilometer

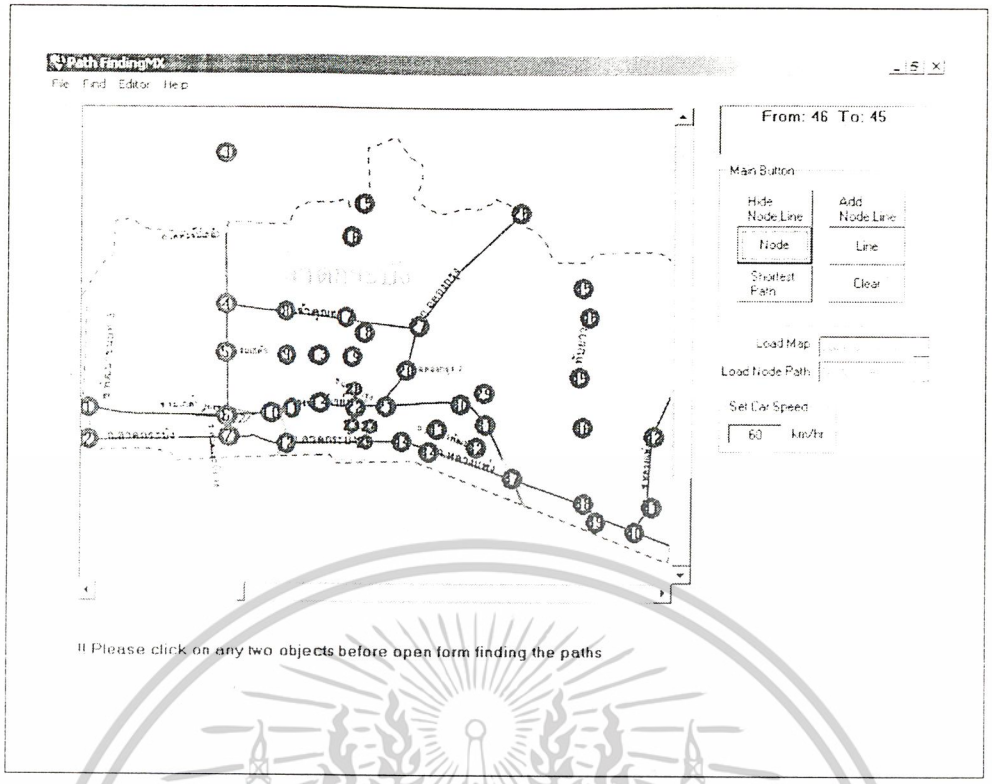
Total Time : 3 Minute



รูปที่ 4.6 แสดงผลเส้นทางที่สั้นที่สุดโปรแกรม

4.2 การวิเคราะห์ข้อมูลจากแผนที่เขตลาดกระบัง

1. เลือกที่ Load Map โดยเลือกเขตลาดกระบัง



รูปที่ 4.7 แสดงภาพแผนที่เขตลาดกระบัง

2. ผลการวิเคราะห์ข้อมูลจากโจทย์ตัวอย่าง โดยเลือกกำหนดจุดเริ่มต้น 2 และ จุด ปลายทางเป็น 40 จะได้ผลลัพธ์ดังนี้

Setting

Form : 2

To : 40

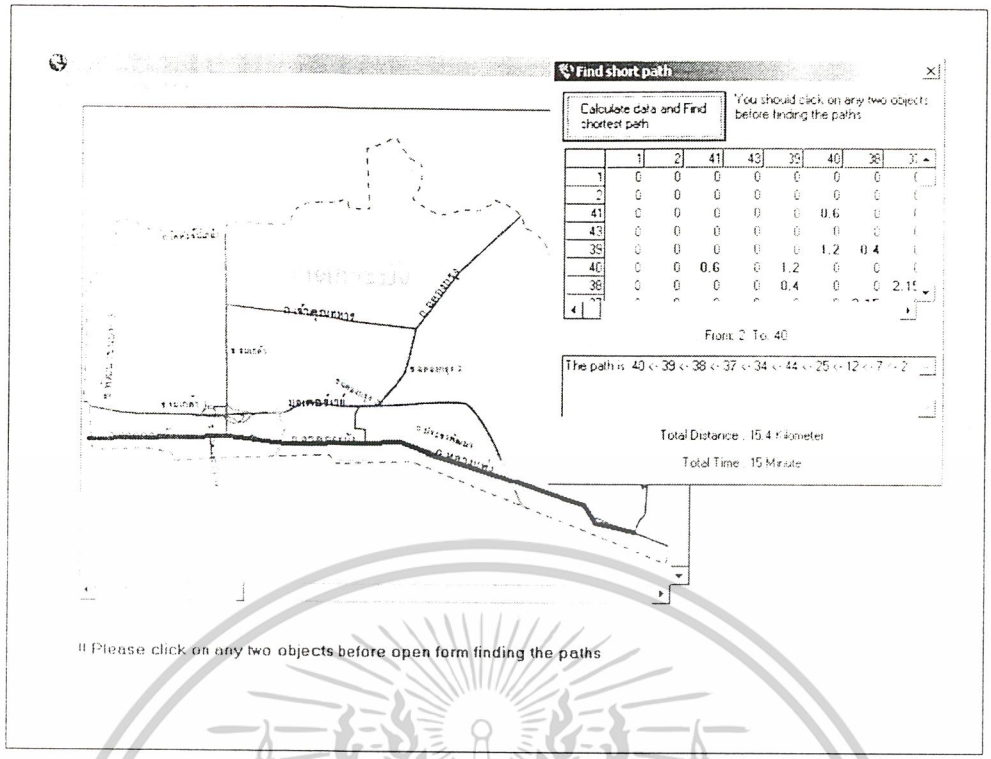
Set Car Speed : 60 km/hr

Result

The path is : Shortest path is : 2-->7-->12-->25-->44-->34-->37-->38-->39-->40

Total Distance : 15.4 Kilometer

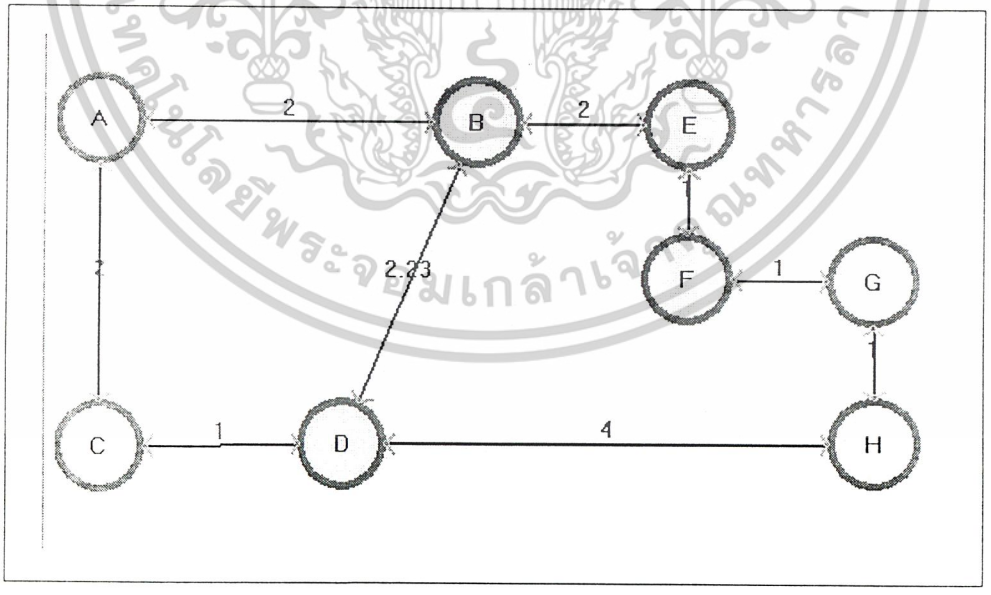
Total Time : 15 Minute



รูปที่ 4.8 แสดงผลลัพธ์การคำนวณเส้นทางที่สั้นที่สุดจากจุด 2 ถึง 40 ของข้อมูลเขตลาดกระบัง

4.3 การวิเคราะห์ข้อมูลจากจากโจทย์ตัวอย่างขนาดเล็ก

1. สร้างโครงข่ายข้อมูลตัวอย่างขนาดเล็ก เพื่อง่ายต่อการพิจารณาความถูกต้องของผลลัพธ์ที่ได้ออกมาของโปรแกรมว่ามีความเชื่อถือได้เพียงใด โดยมีโครงข่ายดังรูป



รูปที่ 4.9 แสดงภาพโครงข่ายตัวอย่าง

2. ผลการวิเคราะห์ข้อมูลจากโครงข่ายตัวอย่างตัวอย่าง โดยเลือกกำหนดจุดเริ่มต้น A และ จุด ปลายทางเป็น F โดยตั้งค่าความเร็วรถยนต์ไว้ที่ 1กิโลเมตรต่อชั่วโมง เพื่อง่ายต่อการพิจารณา จะได้ผลลัพธ์ดังนี้

Setting

Form : A

To : F

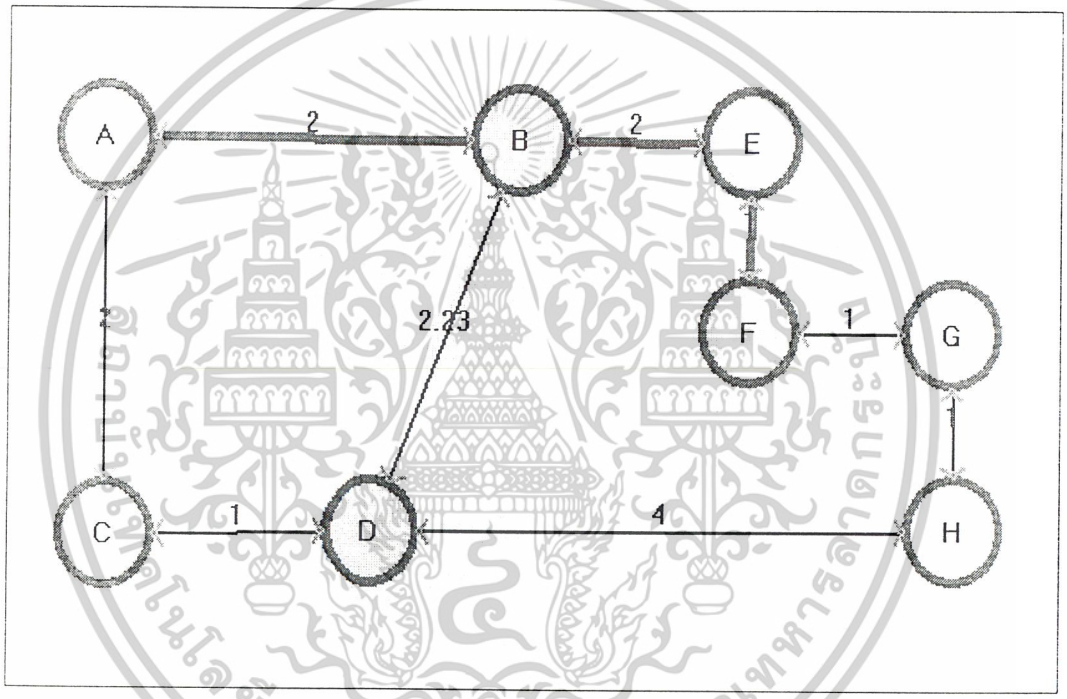
Set Car Speed : 1 km/hr

Result

The path is : Shortest path is : A-->B-->E-->F

Total Distance : 5 Kilometer

Total Time : 300 Minute



รูปที่ 4.10 แสดงผลลัพธ์การคำนวณเส้นทางที่สั้นที่สุดจากจุด A ถึง F ของข้อมูลโครงข่ายตัวอย่าง

ซึ่งผลลัพธ์ที่มีความถูกต้องเมื่อเปรียบเทียบกับการคิดโดยสามารถหาเส้นทางที่เกิดขึ้นทั้งหมดได้ดังนี้

เส้นทาง จาก A ถึง F มีเส้นทางทั้งหมดคือ

1. A -> B -> E -> F : 5 Kilometer
2. A -> B -> D -> H -> G -> F : 10.23 Kilometer
3. A -> C -> D -> H -> G -> F : 8.23 Kilometer
4. A -> C -> D -> B -> E -> F : 9 Kilometer

ฉะนั้นเส้นทางที่สั้นที่สุดคือเส้นทาง A -> B -> E -> F ระยะทาง 5 Kilometer ซึ่งตรงตามผลลัพธ์ของโปรแกรมที่คำนวณได้จึงสามารถบอกได้ว่าโปรแกรมมีความถูกต้องในการค้นหาเส้นทางที่สั้นที่สุด

บทที่ 5
สรุปผลการวิเคราะห์และข้อเสนอแนะ

5.1 สรุปผลการวิเคราะห์

การหาเส้นทางที่สั้นที่สุดนั้น แก้ปัญหาได้โดยการรวบรวมข้อมูลที่เกี่ยวกับเส้นทางต่างๆ จากนั้นหาวิธีการที่จะวิเคราะห์เพื่อหาเส้นทางที่ดีที่สุด สำหรับหลักการทางคณิตศาสตร์ที่ได้นำมาใช้ในการค้นหาเส้นทาง คือ เทคนิคการหาเส้นทางที่สั้นที่สุด (Shortest route problem) ซึ่งนำมาใช้แก้ปัญหาการหาเส้นทางที่สั้นที่สุดระหว่างจุดสองจุด ซึ่งการหาเส้นทางที่สั้นที่สุดนี้อาจจะเป็นการหาเส้นทางที่ใช้ระยะทางที่สั้นที่สุด หรือ ใช้เวลาในการเดินทางน้อยที่สุด หรือ ในแง่ของการเสียค่าใช้จ่ายในการเดินทางน้อยที่สุด

จากการวิเคราะห์โดยโปรแกรมที่สร้างขึ้นโดยใช้อัลกอริทึม ของ Dijkstra's Alogorithm ในการค้นหาเส้นทางที่สั้นที่สุดโดยอ้างอิงแผนที่กรุงเทพมหานครและแผนที่เขตต่างๆ 50 เขต ซึ่งจากการวิเคราะห์โครงข่ายข้อมูลต่างๆได้ผลลัพธ์แสดงเป็นเส้นทางที่สุดในการเดินทางจากจุดหนึ่งไปยังอีกจุดหนึ่ง ซึ่งในการตรวจสอบความถูกต้องของผลลัพธ์ที่ได้นั้นในโครงข่ายของข้อมูลขนาดใหญ่นั้นอาจไม่สามารถที่จะตรวจสอบได้ว่าโปรแกรมสามารถคำนวณผลลัพธ์ได้ถูกต้อง เพราะโครงข่ายข้อมูลขนาดใหญ่ไม่สามารถที่จะคำนวณหรือพิจารณาด้วยสายตาได้ว่ามีผลลัพธ์จริงนั้นตรงกับผลลัพธ์ที่โปรแกรมคำนวณได้หรือไม่ จึงต้องมีการสร้างโครงข่ายข้อมูลขนาดเล็กขึ้นเพื่อทำการเปรียบเทียบเพราะโครงข่ายข้อมูลขนาดเล็กนั้นมีเส้นทางที่เกิดขึ้นระหว่างจุดสองจุดเป็นจำนวนน้อย ซึ่งสามารถที่จะคิดคำนวณหรือทำการพิจารณาด้วยสายตาได้ว่าเส้นทางที่สั้นที่สุดเป็นเส้นทางใดระหว่างจุดสองจุด ซึ่งผลของการเปรียบเทียบระหว่างการคำนวณด้วยโปรแกรมกับการคิดคำนวณหรือการพิจารณาด้วยสายตาจะพบว่าผลลัพธ์ที่ได้ของโปรแกรมนั้นเท่ากับผลลัพธ์ที่ได้จากการคิดคำนวณหรือจากการพิจารณาด้วยสายตาจึงสามารถสรุปได้ว่าผลลัพธ์ที่ได้จากการประมวณผลของโปรแกรมมีความถูกต้องตรงตามที่ต้องการ

ผลลัพธ์ที่ได้จากโปรแกรมจะเป็นเพียงคำตอบของเส้นทางที่สั้นที่สุดเท่านั้น เพราะข้อมูลของโปรแกรมนั้นจะอ้างอิงเฉพาะระยะทางเท่านั้น ไม่สามารถที่จะบอกเส้นทางที่ใช้เวลาในการเดินทางที่สั้นที่สุดได้ เพราะการจะวิเคราะห์ให้ได้ผลลัพธ์ว่าเส้นทางที่ใช้เวลาน้อยที่สุดนั้นจะต้องมีการอ้างอิงถึงปัจจัยของสภาพการจราจร ซึ่งมีความแปรปรวนอยู่ตลอดเวลา เป็นการยากที่จะหาผลลัพธ์ที่ตรงกับความเป็นจริงได้ เพราะจะต้องมีข้อมูลของปัจจัยต่างๆเช่น ปริมาณการจราจรแต่ละช่วงเวลา ทิศทางการเดินรถในแต่ละช่วงเวลา ข้อมูลเส้นทางลัดและทางด่วน เป็นต้น ซึ่งการจะหาผลลัพธ์ให้ตรงกับความเป็นจริงนั้นก็จะต้องมีการเก็บและปรับปรุงข้อมูลต่างๆอยู่ตลอดเวลา

5.2 ข้อเสนอแนะ

1. ควรจะใช้คอมพิวเตอร์ที่มีขีดความสามารถในการประมวณผลค่อนข้างสูง เนื่องจากปริมาณข้อมูลที่มีจำนวนมากถ้าใช้คอมพิวเตอร์ที่มีขีดความสามารถในการประมวณผลต่ำจะใช้เวลาในการประมวณผลค่อนข้างมาก
2. โครงงานนี้ได้ทำโปรแกรมที่สามารถสร้างฐานข้อมูลใหม่ขึ้นมาได้ โดยไม่จำเป็นต้องอ้างอิงแผนที่กรุงเทพมหานครตลอด สามารถที่จะสร้างฐานข้อมูลขึ้นใหม่โดยการนำภาพแผนที่ที่ต้องการ มาสร้างโครงข่ายระยะทางขึ้น ซึ่งอาจนำไปประยุกต์ใช้กับแผนผังภายในโรงงานก็ได้

3. ควรมีการพิจารณาถึงตรอก ซอยต่างๆ และทางด่วนต่างๆ เพื่อที่ผลลัพธ์ในการประมวณผลที่ได้มีความถูกต้องและใกล้เคียงกับความเป็นจริงที่สุด ซึ่งโปรแกรมสามารถทำการเพิ่มข้อมูลใหม่
4. ควรมีการพิจารณาถึงปริมาณการจราจรในแต่ละถนนเพื่อที่ผลลัพธ์ในการประมวณผลที่ได้มีความถูกต้องและใกล้เคียงกับความเป็นจริงที่สุด
5. ควรมีการพิจารณาถึงช่วงเวลาที่มีผลบังคับใช้เส้นทางการเดินทางเดียวและเส้นทางการเดินรถสองทางเพื่อให้เหมาะสมกับสภาพเส้นทางจริงมากขึ้น เพราะในการกำหนดของโปรแกรมนั้นได้ทำการกำหนดให้เป็นการเดินรถสองเส้นทางเท่านั้นเพราะเป็นการอ้างอิงข้อมูลเฉพาะเส้นทางสายหลักเท่านั้น

บรรณานุกรม

1. วรฉัตร ธรรมอารี , "การค้นหาภาพแผนที่แสดงเส้นทางจราจรกรุงเทพมหานคร โดยใช้ฐานความรู้" , ปริญญานิพนธ์ ภาควิชาคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สจล. ปีการศึกษา 2537

2.วิศิษฐ์ สุดันไชยนนท์ , "การค้นหาภาพแผนที่แสดงเส้นทางจราจร" , วิทยานิพนธ์ สาขาวิชาวิศวกรรมไฟฟ้า บัณฑิตวิทยาลัย สจล. ปีการศึกษา 2541

3. พรพิศ บูรณสัมปทานนท์ และ สุดธิดา วิวัฒนสราญรมย์ และ สุรัสวดี ประสานพันธ์ ,"การวิเคราะห์เส้นทางในเขตกรุงเทพมหานครเพื่อใช้ระยะทางหรือระยะเวลาในการเดินทางน้อยที่สุด" , ปริญญานิพนธ์ ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ สจล. ปีการศึกษา 2541

4. ประกาศิต ชาติบุรุษ และ อาทิตย์ จิตต์จุพานนท์ , "โครงสร้างข้อมูลและอัลกอริทึม" , สำนักพิมพ์ซีเอ็ดยูเคชั่น . พ.ศ. 2533

5. ทรงลักษณ์ พิริยะไพโรจน์ และ สุมนา เกษมสวัสดิ์ , "เรียนลัด Data Structure ด้วย Visual Basic" . บริษัท โปรวิชั่น, 2544

6. ดร.ก่อเกียรติ เก่งสกุล และ บุญเจริญ ศิริเนาวกุล , "ทฤษฎีและการประยุกต์ใช้งานปัญญาประดิษฐ์และระบบผู้เชี่ยวชาญ" , บริษัท ซีเอ็ดยูเคชั่น จำกัด(มหาชน)

7. กิตติ ภักดีวัฒนะกุล, "Visual Basic 6 ฉบับโปรแกรมเมอร์" , หจก.ไทยเจริญการพิมพ์ ,2543

8. สมศักดิ์ ศรีขจรเกียรติ , "Advanced Visual Basic 6 Teach Yourself" , หจก.บิบลิโอไฟล์ พับลิชชิ่ง, 2544

9. George F.Luger and William A.Stubblefield, "Artificial intelligence and the design of expert ststems", The Benjamin/cummings publishing company inc., 1988