ขั้นตอนวิธีการสร้างเส้นทางที่น้อยที่สุดสำหรับการทดสอบแบบกิ่ง

MINIIMAL PATH GENERATION ALGORITHM FOR BRANCH TESTING

กาญจนาภรณ์ สมบุญเจริญ
นรุตม์ เต้พันธ์
ปวิตรา เพียราธิสิทธิ์

ปัญหาพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต
ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2550

MINIMAL PATH GENERATION ALGORITHM FOR BRANCH TESTING

KARNJANAPORN SOMBOONCHAROEN

NARUT TEPAN

PAWITRA PIARATISIT

A SPECIAL PROJECT SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIRMENT FOR DEGREE OF BACHELOR OF SCIENCE

DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

FACULTY OF SCIENCE

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

ACADEMIC YEAR 2007

หัวข้อปัญหาพิเศษ	ขั้นตอนวิธีการสร้างเส้นทางที่น้อยที่สุดสำหรับการทดสอบแบบกิ่ง MINIMAL PATH GENERATION ALGORITHM FOR BRANCH TESTING	
ชื่อนักศึกษา	นางสาวกาญจนาภรณ์ สมบุญเจริญ	47050313
	นายนรุตม์ เต้พันธ์	47050334
	นางสาวปวิตรา เพียราธิสิทธิ์	47050340
ภาควิชา	คณิตศาสตร์และวิทยาการคอมพิวเตอร์	
สาขาวิชา	วิทยาการคอมพิวเตอร์	
อาจารย์ที่ปรึกษา	อาจารย์อัคเดช อุดมชัยพร	

ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง อนุมัติให้นับปัญหาพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์ ประจำปีการศึกษา 2550

คณะกรรมการสอบ	ลายมือชื่อ
ดร.นวลสวาท หิรัญสกลวงศ์ ประธานกรรมการ	
อาจารย์ธีระ ฟักอ่อน กรรมการ	
อาจารย์อัคเดช อุดมชัยพร กรรมการและอาจารย์ที่ปรึกษา	

(รองศาสตราจารย์ไพรบูลย์ พันธรักษ์พงษ์)
หัวหน้าภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์

ลิขสิทธิ์ของภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

แด่ ทุกคนที่เป็นกำลังใจมาโดยตลอด
กาญจนาภรณ์

แด่ ผู้ทดสอบซอฟต์แวร์ในหมู่มวลมนุษยชาติ
นรุตม์

แด่ คุณพ่อ-คุณแม่ และเพื่อน ๆทุกคนที่เป็นกำลังใจ
ปวิตรา

หัวข้อปัญหาพิเศษ	ขั้นตอนวิธีการสร้างเส้นทางที่น้อยที่สุดสำหรับการทดสอบแบบกิ่ง	
ชื่อนักศึกษา	นางสาวกาญจนาภรณ์ สมบุญเจริญ	47050313
	นายนรุตม์ เต้พันธ์	47050334
	นางสาวปวิตรา เพียราธิสิทธิ์	47050340
ปริญญา	วิทยาศาสตรบัณฑิต	
ภาควิชา	คณิตศาสตร์และวิทยาการคอมพิวเตอร์	
สาขาวิชา	วิทยาการคอมพิวเตอร์	
ปีการศึกษา	2550	
อาจารย์ที่ปรึกษา	อาจารย์อัคเดช อุดมชัยพร	

บทคัดย่อ

การทดสอบซอฟต์แวร์แบบกิ่งเป็นวิธีการหาชุดของเส้นทางที่ครอบคลุมทุกกิ่งของโครงสร้างซอฟต์แวร์ แต่เมื่อซอฟต์แวร์มีขนาดใหญ่ การหาชุดเส้นทางที่จะนำมาทดสอบมักจะเกิดความซ้ำซ้อนของกรณีทดสอบ ทำให้สิ้นเปลืองเวลาและงบประมาณในการทดสอบซอฟต์แวร์ ปัญหาพิเศษนี้จึงได้นำเสนอขั้นตอนวิธีการสร้างเส้นทางที่น้อยที่สุดสำหรับการทดสอบซอฟต์แวร์แบบกิ่ง เพื่อช่วยแก้ปัญหาความซ้ำซ้อนของเส้นทางดังกล่าว ซึ่งขั้นตอนวิธีที่นำเสนอ มีประสิทธิภาพเป็น $\Omega(n) = 1$ สำหรับความซับซ้อนของกรณีที่ดีที่สุด และ $O(f(n)) = n^{\frac{n}{2}}$ สำหรับความซับซ้อนของกรณีที่แย่ที่สุด เมื่อ n คือจำนวนเส้นทางทั้งหมด ซึ่งขั้นตอนวิธีนี้จะช่วยให้การทดสอบซอฟต์แวร์ใช้ระยะเวลาและงบประมาณที่น้อยลง

Title	MINIMAL PATH GENERATION ALGORITHM FOR BRANCH TESTING
Students	Ms.Karnjanaporn Sombooncharoen 47050313
	Mr.Narut Tepan 47050334
	Ms.Pawitra Piaratisit 47050340
Degree	Bechelor of Science
Department	Mathematics and Computer Science, Faculty of Science
Programme	Computer Science
Academic Year	2007
Advisor	Mr.Akadej Udomchaiporn

ABSTRACT

Branch testing is the testing method finding set of paths for branch covering all branch of a software struvture. When the size of software structure is large, path generation is usually redundant. This redundancy cause time and cost consuming. Therefore, this special problem presents minimal path generation algorithm for Branch testing to solve the redundancy problem. The efficiency of the algorithm is $\Omega(n) = 1$ for the best case and $O(f(n)) = n^{\frac{n}{2}}$ for the worst case. The algorithm help software tester reduce time and cost of software testing.

กิตติกรรมประกาศ

ในการจัดทำปัญหาพิเศษเรื่องขั้นตอนวิธีการสร้างเส้นทางที่น้อยที่สุดสำหรับการทดสอบแบบกิ่งนี้ คณะผู้จัดทำขอขอบพระคุณอาจารย์อัคเดช อุดมชัยพร อาจารย์ที่ปรึกษาปัญหาพิเศษนี้ ที่ได้กรุณาเสียสละเวลาให้คำแนะนำในการปรับปรุง และแก้ไขปัญหาต่างๆที่เกิดขึ้นขณะดำเนินการทำปัญหาพิเศษนี้ รวมทั้งยังเป็นผู้ตรวจสอบความถูกต้องของปัญหาพิเศษนี้ได้เป็นอย่างดี

ขอขอบพระคุณคณะอาจารย์ทุกท่านที่ช่วยประสิทธิ์ประสาทวิชาความรู้ทั้งทางด้านทฤษฎี และภาคปฏิบัติ อีกทั้งช่วยอบรมทางด้านคุณธรรม และจริยธรรมแก่คณะผู้จัดทำเพื่อให้เป็นประโยชน์แก่ปัญหาพิเศษนี้ให้สำเร็จลุล่วงตามปณิธานที่ได้ตั้งไว้

ขอขอบพระคุณเจ้าหน้าที่ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์ที่ช่วยอำนวยความสะดวกในการใช้ห้องปฏิบัติการคอมพิวเตอร์ และอำนวยความสะดวกในการเบิกอุปกรณ์ซึ่งใช้ในการจัดทำปัญหาพิเศษนี้

ท้ายนี้ขอขอบพระคุณบิดา มารดา ผู้ซึ่งให้ความสนับสนุนอุปการะเลี้ยงดูและสนับสนุนทางด้านทุนทรัพย์ของปัญหาพิเศษนี้ อีกทั้งยังเป็นผู้ซึ่งคอยให้กำลังใจตลอดการทำปัญหาพิเศษนี้จนได้สำเร็จลุล่วงเป็นอย่างดี รวมทั้งเพื่อนๆ พี่ๆ และน้องๆ ทุกคนที่มีส่วนคอยให้ความช่วยเหลือต่างๆ ในการจัดทำปัญหาพิเศษนี้ให้สัมฤทธิ์ผลได้ด้วยดีไว้ ณ ที่นี้ด้วย

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญภาพ

ภาพที่ **หน้า**

สารบัญภาพ (ต่อ)

ภาพที่ **หน้า**

สารบัญตาราง

ตารางที่ **หน้า**

บทที่ 1
บทนำ

1.1 ที่มาและความสำคัญของปัญหา

การทดสอบซอฟต์แวร์เป็นการหาข้อผิดพลาดที่อาจจะเกิดขึ้นของโปรแกรม ซึ่งข้อผิดพลาดที่เกิดขึ้นนั้นบางข้อผิดพลาดอาจไม่ส่งผลกระทบต่อซอฟต์แวร์หรือก่อความเสียหายเพียงเล็กน้อย แต่บางข้อผิดพลาดก็ก่อให้เกิดความเสียหายอย่างใหญ่หลวงกับซอฟต์แวร์จนถึงอาจส่งผลให้ซอฟต์แวร์หยุดทำงานได้ การทดสอบซอฟต์แวร์จึงเป็นสิ่งจำเป็น วิธีการหนึ่งที่มักจะถูกนำมาใช้ทดสอบซอฟต์แวร์คือการทดสอบซอฟต์แวร์แบบกิ่ง การทดสอบซอฟต์แวร์แบบกิ่งนั้นเป็นการหาชุดของเส้นทางที่ตรงตามเงื่อนไขของการทดสอบซอฟต์แวร์แบบกิ่งคือ ครอบคลุมทุกกิ่งของโครงสร้างซอฟต์แวร์ แต่การทดสอบซอฟต์แวร์แบบกิ่งมักจะเกิดความซ้ำซ้อนของกรณีทดสอบ ซึ่งทำให้สิ้นเปลืองงบประมาณและระยะเวลาในการทดสอบซอฟต์แวร์ ปัญหาพิเศษนี้จึงเสนอวิธีการหาชุดของเส้นทางที่น้อยที่สุด เพื่อลดความซ้ำซ้อนของกรณีทดสอบที่ได้จากการทดสอบซอฟต์แวร์แบบกิ่ง และสร้างเครื่องมือเพื่อช่วยหาชุดของเส้นทางที่น้อยที่สุด ตามวิธีการที่ได้นำเสนอ

1.2 วัตถุประสงค์

1) เพื่อสร้างชุดของเส้นทางที่น้อยที่สุดที่เป็นไปตามเงื่อนไขของการทดสอบซอฟต์แวร์แบบกิ่ง
2) เพื่อพัฒนาเครื่องมือที่ใช้ในการสร้างชุดของเส้นทางที่น้อยที่สุดสำหรับการทดสอบซอฟต์แวร์แบบกิ่ง

1.3 ข้อจำกัดและขอบเขต

1) ปัญหาพิเศษนี้จะทำการสร้างชุดของเส้นทางที่น้อยที่สุดที่เป็นไปตามเงื่อนไขของการทดสอบซอฟต์แวร์แบบกิ่งจากแผนภาพควบคุมการไหล (Control flow graph)
2) ผลลัพธ์ของโปรแกรมจะแสดงเป็นชุดของเส้นทางที่น้อยที่สุดที่ครอบคลุมการทดสอบซอฟต์แวร์แบบกิ่งเท่านั้น

1.4 ขั้นตอนและกรอบเวลาของปัญหาพิเศษ

1) ศึกษาขอบเขตของปัญหาและทฤษฎีที่เกี่ยวข้องกับการทำปัญหาพิเศษ
2) ศึกษาเอกสารอ้างอิงที่ใช้ประกอบในการจัดทำปัญหาพิเศษ
3) ออกแบบวิธีการค้นหาชุดของเส้นทางสำหรับการทดสอบซอฟต์แวร์แบบกิ่ง
4) พัฒนาโปรแกรม
5) ตรวจสอบ ปรับปรุง แก้ไขโปรแกรม

6) ทดลองเปรียบเทียบประสิทธิภาพของวิธีการค้นหาชุดของเส้นทางสำหรับการทดสอบซอฟต์แวร์แบบกิ่ง

7) สรุปและประเมินผลการทดลอง

8) จัดทำเอกสารประกอบการทำปัญหาพิเศษ

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1) ได้ชุดของเส้นทางที่น้อยที่สุด ซึ่งสามารถช่วยลดความซ้ำซ้อนของกรณีทดสอบสำหรับการทดสอบซอฟต์แวร์แบบกิ่ง

2) ได้ชุดเครื่องมือที่ใช้ในการหาชุดของเส้นทางที่น้อยที่สุดสำหรับการทดสอบซอฟต์แวร์แบบกิ่ง

ส่วนประกอบของปัญหาพิเศษนี้ ได้แบ่งเป็นบทต่างๆ ดังนี้

บทที่ 2 ทฤษฎีและงานวิจัยที่เกี่ยวข้อง ในบทนี้จะกล่าวถึงทฤษฎีต่างๆที่เกี่ยวข้องในการทำการทำปัญหาพิเศษครั้งนี้คือ ทฤษฎีแผนภาพควบคุมการไหล วิธีการ และหลักการแปลงซอฟต์แวร์จากโปรแกรมเป็นแผนภาพ ทฤษฎีการทดสอบซอฟต์แวร์แบบกิ่ง ทฤษฎีกราฟ ปัญหาการค้นหาของกราฟ

บทที่ 3 ขั้นตอนวิธีการสร้างเส้นทางที่น้อยที่สุด ในบทนี้จะกล่าวถึงวิธีการนำทฤษฎีต่างๆจากบทที่สองมาประยุกต์ใช้เพื่อทำการวิจัยนี้ และแสดงขั้นตอนวิธีของโปรแกรมหาเส้นทางทั้งหมดในการท่องโปรแกรมที่เป็นไปได้ และขั้นตอนวิธีในการหาเส้นทางที่น้อยที่สุดที่จะครอบคลุมทุกกิ่ง

บทที่ 4 ผลการทดลอง ในบทนี้จะกล่าวถึงผลจากทดลองขั้นตอนวิธีที่เราได้พัฒนาขั้นมา พร้อมกับคำนวณค่าความซับซ้อนของโปรแกรม

บทที่ 5 สรุปผลการวิจัย การอภิปราย และข้อเสนอแนะ ในบทนี้จะกล่าวถึงข้อสรุปที่ได้จากการทดลอง รวมถึงอภิปรายสิ่งที่ได้จากการทดลองในบทที่ 4 ข้อดี ข้อเสีย ของขั้นตอนวิธีที่ได้พัฒนาขึ้นมา และข้อเสนอแนะในอนาคตสำหรับการนำขั้นตอนวิธีไปพัฒนาต่อ

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

ในบทนี้จะกล่าวถึงวิธีการทฤษฎีพื้นฐานต่างๆ ที่เกี่ยวข้องในการทำการทดสอบโปรแกรมแบบกิ่ง ซึ่งประกอบด้วยทฤษฎีกราฟ และจะกล่าว

2.1 ทฤษฎีที่เกี่ยวข้อง

2.1.1 การทดสอบซอฟต์แวร์แบบกิ่ง

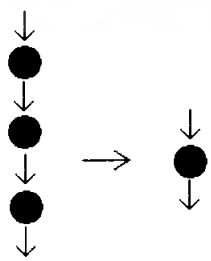
การทดสอบซอฟต์แวร์แบบกิ่ง คือ การทดสอบซอฟต์แวร์ประเภทรู้โครงสร้างซอฟต์แวร์ หรือที่เรียกว่ากล่องขาว (White box) ซึ่งการทดสอบประเภทนี้นั้น ทุกๆ กิ่งที่ปรากฏจะต้องถูกทดสอบด้วย อย่างน้อย 1 ครั้ง ขั้นตอนของการทดสอบซอฟต์แวร์แบบกิ่ง จะแบ่งได้เป็น 3 ขั้นตอนหลักๆ คือ การสร้างแผนภาพควบคุมการไหล การค้นหาเส้นทางของการทดสอบซอฟต์แวร์แบบกิ่ง และการสร้างกรณีทดสอบจากเส้นทางที่ได้

2.1.1.1 การสร้างแผนภาพควบคุมการไหล (Control flow graph)

แผนภาพควบคุมการไหล คือ แผนภาพที่แสดงส่วนต่างๆ ของโปรแกรม โดยส่วนที่เป็นจุดจะแทนประโยคคำสั่งต่างๆของโปรแกรม ส่วนที่เป็นเส้นเชื่อมจะแสดงเส้นทางที่เป็นไปได้ที่จะถูกท่องผ่านเมื่อโปรแกรมถูกประมวลผล โดยสัญลักษณ์ที่ใช้แทนส่วนต่างๆ ของแผนภาพควบคุมกระแสมีดังนี้

ก. บล็อกกระบวนการ (Process block)

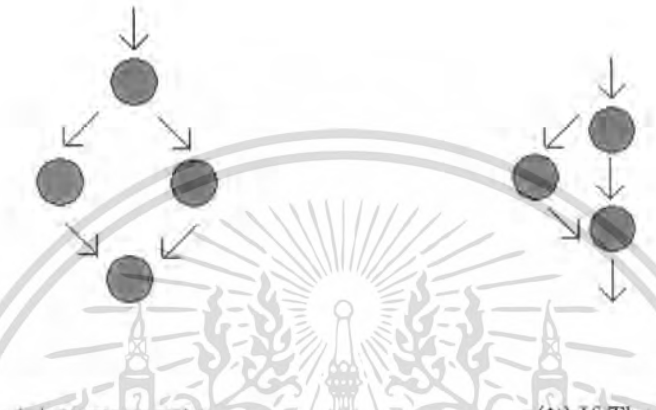
คือลำดับของประโยคคำสั่งทั่วไปที่ต่อเนื่องกัน โดยจะมีเส้นเชื่อมเข้าหนึ่งเส้นและเส้นเชื่อมออกหนึ่งเส้น และในกรณีที่มีประโยคคำสั่งต่อเนื่องกันหลายประโยคคำสั่งโดยไม่มีประโยคตัดสินใจหรือจุดเชื่อมมาคั่น จะสามารถเขียนรวมได้เป็นหนึ่งจุด สัญลักษณ์ของบล็อกกระบวนการ ดังภาพที่ 2.1



ภาพที่ 2.1 สัญลักษณ์ของบล็อกกระบวนการ
ซึ่งสามารถย่อให้เหลือเพียงจุดเดียวได้

ข. ประโยคตัดสินใจ (Decisions)

คือจุดในโปรแกรมซึ่งแสดงคำสั่งตัดสินใจที่ให้เลือกทางใดทางหนึ่ง โดยประโยคตัดสินใจส่วนใหญ่เป็นประโยคตัดสินใจแบบสองทาง เช่นประโยคคำสั่ง If-Then และ If-Then - Else สัญลักษณ์ของประโยคตัดสินใจ ดังภาพที่ 2.2



(ก) If-Then –Else (ข) If-Then

ภาพที่ 2.2 สัญลักษณ์ของประโยคตัดสินใจ

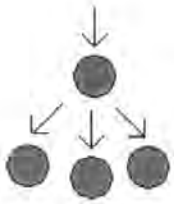
ค. จุดเชื่อมต่อ (Junction)

คือจุดที่เป็นจุดรวมของทางแยก สัญลักษณ์ของจุดเชื่อมต่อ ดังภาพที่ 2.3

ภาพที่ 2.3 สัญลักษณ์ของจุดเชื่อมต่อ

ง. ข้อความแสดงกรณี (Case statement)

คือประโยคทางเลือกที่มีทางเลือกตั้งแต่สองทางขึ้นไปเช่น If-else If สัญลักษณ์ของข้อความแสดงกรณีแสดงดังภาพที่ 2.4



ภาพที่ 2.4 สัญลักษณ์ของข้อความแสดงกรณี

จ. การวนซ้ำ (Iteration)

คือประโยคคำสั่งที่มีการทำงานซ้ำ จนกว่าจะถึงเงื่อนไขที่กำหนด สามารถแบ่งได้สองแบบคือ

- ตรวจสอบก่อนเข้าลูป (pre-test) เช่น while-loop หรือ for-loop
- ตรวจสอบหลังจากทำงานในลูปแล้ว (post-test) เช่น do-while-loop

สัญลักษณ์ของการวนซ้ำแสดงดังภาพที่ 2.5

(ก) ภาพแบบของการตรวจสอบก่อนเข้าลูป

(ข) ภาพแบบของการตรวจสอบหลังจากเข้าทำงานในลูป

ภาพที่ 2.5 สัญลักษณ์ของการวนซ้ำ

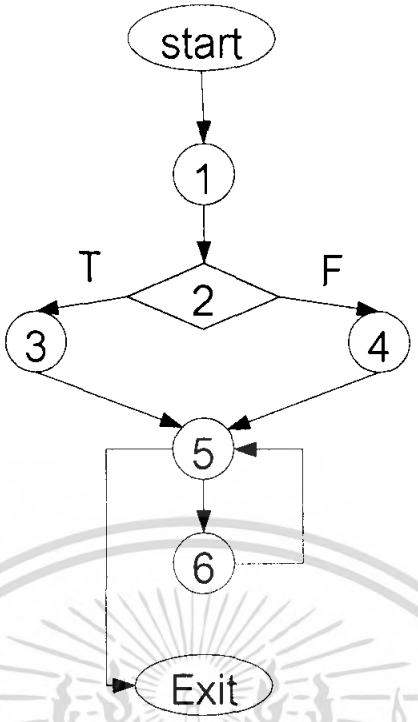
ในการสร้างแผนภาพควบคุมการไหลเราจะเริ่มจากแบ่งโปรแกรมเป็นส่วนโดยประโยคคำสั่งที่เป็นลำดับติดกันจะรวมเป็นจุดเดียวกัน ส่วนประโยคคำสั่งทางเลือกและประโยคคำสั่งวนลูปจะแยกเป็นหนึ่งจุดดังแสดงในภาพที่ 2.6

```
1  class Example
2  {
3      public static void main(String[] args)
4      {
5          double x;
6          int a = 6;
7          int b = 3;
8      ②   if (a<b)
9          {
10             x = a*b;  ③
11         }
12         else
13         {
14             x = a/b;  ④
15         }
16     ⑤   while (x<a+b)
17         {
18             a = a+1;
19             b = b+1;
20             System.out.println(a);
21             System.out.println(b);
22             break;
23         }
24     }
25 }
26
```

① ⑥

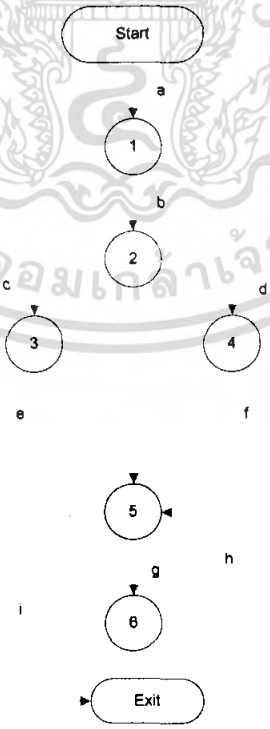
ภาพที่ 2.6 การกำหนดส่วนต่างๆของโปรแกรมเพื่อสร้างแผนภาพควบคุมการไหล

จากนั้นจึงทำการกำหนดหมายเลขให้แต่ละข้อความเพื่อเป็นจุดก่อน แล้ววาดภาพจุด จากนั้นนำแต่ละจุดมาวาดต่อกันและวาดลูกศรแสดงเส้นทางต่างๆ ของโปรแกรมให้เป็นแผนภาพควบคุมการไหล โดยจุดที่แทนประโยคตัดสินใจจะถูกเขียนเป็นภาพสี่เหลี่ยมขนมเปียกปูน ส่วนจุดเริ่มต้นเขียนกำกับด้วย Start และจุดสิ้นสุดเขียนกำกับด้วย Exit ดังแสดงในภาพ 2.7 จุดที่สองเป็นจุดที่แสดงคำสั่ง If-Else และจุดที่ ห้า และจุดที่ หก เป็นจุดที่ใช้แสดงการ วนลูป While สังเกตได้ว่าประโยคคำสั่งที่อยู่ติดกัน ถูกรวมไว้เป็นจุดเดียวกันนั่นคือ จุดที่หนึ่ง และจุดที่หก และจากโปรแกรมตามภาพที่ 2.6 สามารถนำมาสร้างเป็นแผนภาพควบคุมการไหลได้ดังตัวอย่างในภาพที่ 2.7



ภาพที่ 2.7 ตัวอย่างของแผนภาพควบคุมการไหล

2.1.1.2 การหาเส้นทางของการทดสอบซอฟต์แวร์แบบกิ่ง



ภาพที่ 2.8 ตัวอย่างของแผนภาพที่จะใช้ในการหาเส้นทางของการทดสอบแบบกิ่ง

จากตัวอย่างแผนภาพควบคุมการไหลในภาพที่ 2.7 จะมีการเพิ่มเติมรายละเอียดคือ การกำหนดชื่อให้กับแต่ละกิ่ง เพื่อความสะดวกต่อการตรวจสอบการหาเส้นทางของการทดสอบแบบกิ่ง ซึ่งตัวอย่างของแผนภาพที่จะใช้ในการหาเส้นทางของการทดสอบแบบกิ่ง จะแสดงในภาพที่ 2.8 และการหาเส้นทางของการทดสอบแบบกิ่งจะแสดงได้ดังตัวอย่างที่จะกล่าวถึงต่อไปนี้

ตัวอย่างการสร้างเส้นทางสำหรับการทดสอบแบบกิ่ง

กรณีการทดสอบที่ 1 : a – b – c – e – g –h –i

กรณีการทดสอบที่ 2 : a – b – d – f – i

ตารางที่ 2.1 ตัวอย่างตารางที่ใช้ในการทดสอบแบบกิ่ง

Paths	Links								
	A	b	C	d	e	f	g	h	i
a – b – c – e – g – h - i	x	x	X		x		x	x	x
a – b – d – f - i	x	x		x		x			x
Total	X	x	X	x	x	x	x	x	x

จากตารางที่2.1 เราจะสังเกตจากแถวล่างสุดได้ว่าทุกจุดได้รับการเข้าเยี่ยมแล้ว ดังนั้นแค่ 2 เส้นทางดังตัวอย่างจึงถือว่าครอบคลุมแล้วสำหรับการสร้างเส้นทางการทดสอบแบบกิ่ง

2.1.1.3 การสร้างกรณีการทดสอบจากเส้นทางที่ได้

จากตัวอย่างดังกล่าวเราจะนำชุดเส้นทางที่ได้ไปสร้างชุดข้อมูลเพื่อใช้สำหรับการทดสอบต่อไป

2.1.2 ทฤษฎีกราฟ

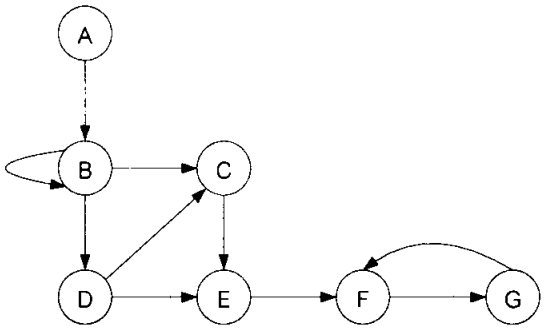
ในการแบ่งโปรแกรมเป็นจุดและจัดเก็บโปรแกรมลงฐานข้อมูล เราจัดเก็บโปรแกรมในภาพแบบโครงสร้างของกราฟ โดยในกราฟจะแสดงความสัมพันธ์ระหว่างจุดและด้านที่เชื่อมระหว่างจุด นิยามภาพแบบหนึ่งของกราฟคือ

1) การวนซ้ำ (Loop) ในกราฟคือ ลำดับของกราฟที่มีจุดเริ่มต้นและจุดจบที่จุดเดียวกัน

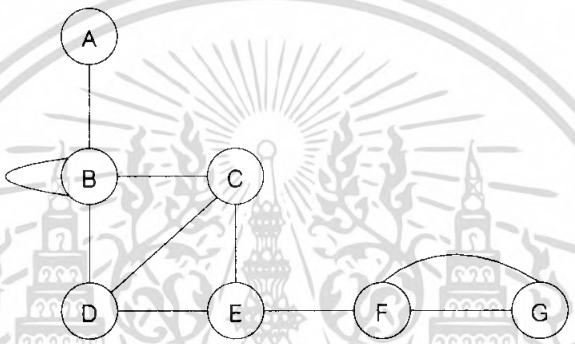
2) เส้นหลายชั้น (Multiple edges) คือเส้นตั้งแต่สองเส้นขึ้นไปเชื่อมจุดคู่เดียวกัน

3) กราฟเชิงเดียว (Simple graph) คือกราฟที่ไม่มีเส้นหลายชั้นและไม่มีการวนซ้ำ

โดยกราฟสามารถแบ่งได้สองแบบคือ แบบเดินได้ทางเดียว (Directed graph) มีลูกศรเป็นตัวบอกทิศ และแบบเดินได้ทั้งสองทาง (Undirected graph) ดังภาพที่ 2.9 และ 2.10



ภาพที่ 2.9 กราฟแบบเดินได้ทางเดียว



ภาพที่ 2.10 กราฟแบบเดินได้สองทาง

2.1.2.1 การค้นหาข้อมูลในกราฟ

ในการหาข้อมูลและการเข้าถึงในแต่ละจุดจากกราฟนั้น เริ่มต้นจากการกำหนดสถานะคือ สถานะเริ่มต้น (Start state) และสถานะเป้าหมาย (Goal state) จากนั้นจึงเป็นการหาวิธีที่จะไปถึงสถานะต่างๆ แล้วทำการแปลงจากกราฟไปเป็นต้นไม้ (Tree) แล้วทำการค้นหาบนต้นไม้ ซึ่งการค้นหานั้นเราจะกำหนดให้

- N คือเส้นทางบางส่วนจากจุดเริ่มต้นไปยังบางจุด X
- ให้ Q เป็นรายการของเส้นทาง
- S เป็นจุดเริ่มต้น และ G เป็นจุดสุดท้าย
- Head จุดเริ่มต้นของเส้นทาง

จากนั้นทำการค้นหาในกราฟ ขั้นตอนการค้นหาจุดที่ยังไม่ถูกเยี่ยมในกราฟคือ

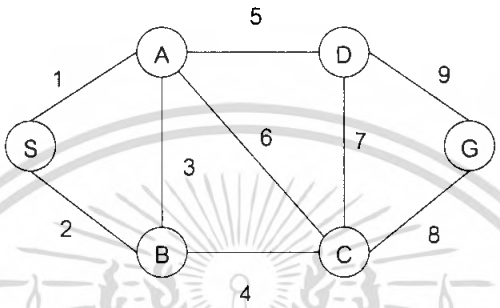
1. กำหนดให้ Q เป็นเส้นทางบางส่วนที่เข้าจุด S กำหนดเซตที่ถูกเยี่ยมคือ เซต visited = S
2. ถ้าไม่มี Q ให้จบโปรแกรม หากยังมีอยู่ ให้เลือกเส้นทาง N จาก Q
3. ถ้า Head(N) = G, return N (ถึงจุดสุดท้าย) ; N เป็นเส้นทางจาก S ถึง G
4. ถ้าไม่ใช่ นำ N ออกจาก Q

5. หาจุดที่สืบทอดมาจาก Head(N) ที่ยังไม่ถูกเยี่ยม(ไม่อยู่ในเซตvisited) และหาด้านทั้งหมดที่เป็นไปได้ของ N ที่ไปยังจุดที่อยู่ติดกันและยังไม่ถูกเยี่ยม

6. เพิ่มด้านที่หาได้ทั้งหมดลงใน Q ; เพิ่มจุดที่ถูกเลือกจาก Head(N) ลงใน visited

7. กลับไปทำขั้นตอนที่ 2

ตัวอย่าง



ภาพที่ 2.11 ตัวอย่างการหาเส้นทางจากกราฟ

จากภาพที่ 2.11 สามารถหาเส้นทางเพื่อให้ทุกจุดในกราฟถูกเยี่ยมตามขั้นตอนการค้นหาจุดที่ยังไม่ถูกเยี่ยมในกราฟ ได้ดังนี้

ขั้นที่ 1 : เยี่ยมจุด S กำหนด visited={S}

ขั้นที่ 2 : จุดที่สืบทอดจาก S = A,B ด้านที่เป็นไปได้ Q=1,2

ขั้นที่ 3 : เยี่ยมจุด A ; N = 1 ; visited = {S, A}

ขั้นที่ 4 : เยี่ยมจุด B ; N = 1-2 ; visited = {S, A, B}

ขั้นที่ 5 : จุดที่สืบทอดจาก B = C ด้านที่เป็นไปได้ Q= 4

ขั้นที่ 4 : เยี่ยมจุด C ; N = 1-2-4 ; visited = {S, A, B, C}

ขั้นที่ 5 : จุดที่สืบทอดจาก C = D,G ด้านที่เป็นไปได้ Q= 7,8

ขั้นที่ 4 : เยี่ยมจุด D ; N = 1-2-4-7 ; visited = {S, A, B, C, D}

ขั้นที่ 4 : เยี่ยมจุด G ; N = 1-2-4-7-8 ; visited = {S, A, B, C, D, G}

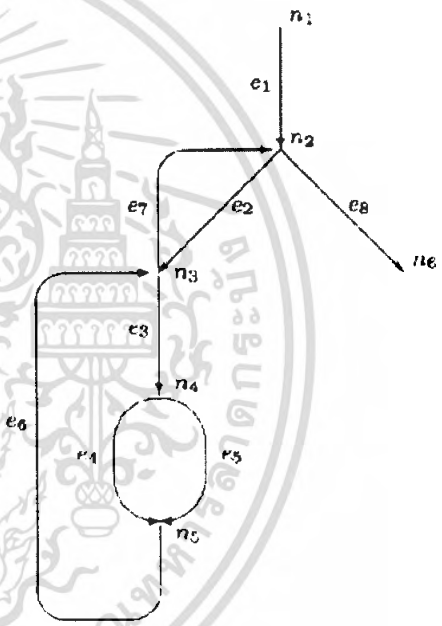
เมื่อจบโปรแกรมจะได้ลำดับจุดที่ถูกเยี่ยมคือ S, A, B, C, D, G และ ลำดับเส้นทางคือ 1-2-4-7-8

2.2 งานวิจัยที่เกี่ยวข้อง

2.2.1 การใช้เซตแผ่ทั่วสำหรับการทดสอบที่ครอบคลุม (Using Spanning Sets for Coverage Testing) [1]

งานวิจัยนี้ เป็นการหาเส้นทางทดสอบโดยการนิยาม เซตแผ่ทั่ว เพื่อใช้นับเซตของ Entity (ในที่นี้หมายถึง ด้าน) ที่น้อยที่สุดที่ทำให้ครอบคลุมแผนภาพ ซึ่งหาได้จากการนำไฟล์ทั้งหมดมาสร้างเป็น Digraph ซึ่งแสดงการท่องกราฟโดยให้ Leaf node แต่ละตัวแทนเส้นทางจากนั้นทำการลดขนาดแล้วเก็บไว้แสดงผล ซึ่งข้อดีของวิธีนี้คือ สามารถลดกรณีทดสอบที่หาได้เหลือจำนวนน้อยเท่าที่เป็นไปได้ โดยประสิทธิภาพในการทดสอบลดลงเพียงเล็กน้อยหรือไม่เลย ตัวอย่างของวิธีการใช้เซตแผ่นทั่วสำหรับการทดสอบที่ครอบคลุมเป็นดังภาพ

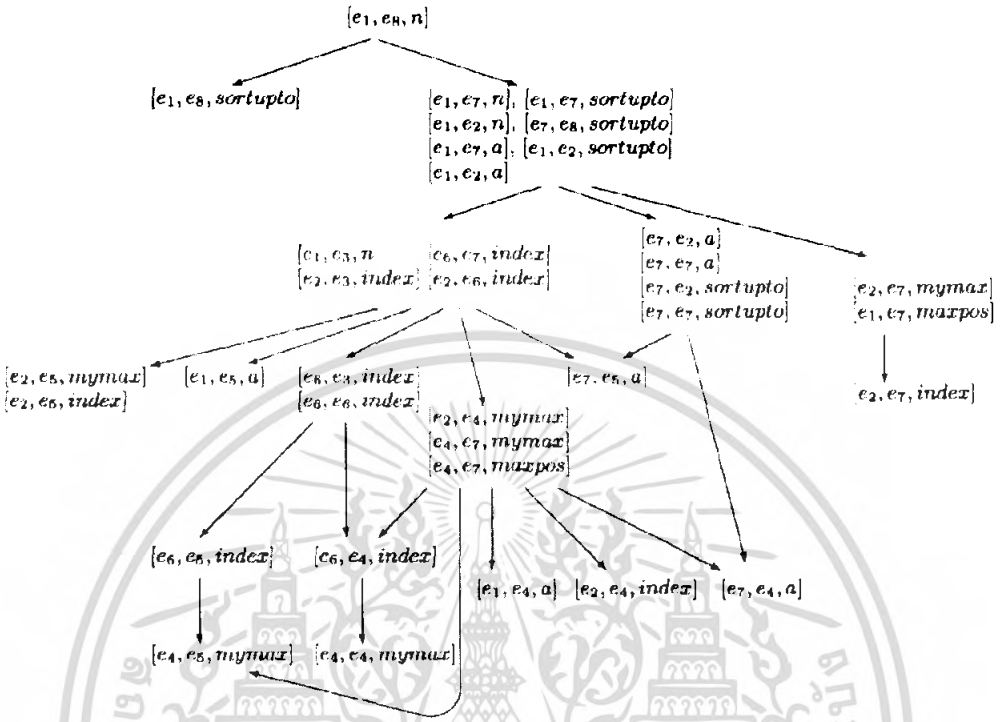
```
void sort(a,n) /* SORT Program */
1   int a[];
2   int n;
3   {
4     int sortupto;
5     int maxpos;
6     int mymax;
7     int index;

8     sortupto = 1;
9     maxpos = 1;
10    while (sortupto < n) {
11      mymax = a[sortupto];
12      index = sortupto + 1;
13      while (index <= n) {
14        if (a[index] > mymax) {
15          mymax = a[index];
16          maxpos = index;
17        }
18        index = index + 1;
19      }
20      index = a[sortupto];
21      a[sortupto] = mymax;
22      a[maxpos] = index;
23      sortupto = sortupto + 1;
24    }
25  }
```



ภาพที่ 2.12 ตัวอย่างข้อมูลเข้าทางด้านซ้ายสามารถแปลงเป็นแผนภาพได้ดังภาพด้านขวา

จากภาพที่ 2.12 เป็นข้อมูลเข้าซึ่งถูกแปลงเป็นแผนภาพดังด้านซ้าย จากนั้นจึงสร้าง ไดกราฟเรียกกราฟการจัดกลุ่มการแผ่ทั่ว (c-subsumption digraph) ซึ่งเกิดจากการควบรวมของ Entity ซึ่งเกิดจากด้านของแผนภาพ ที่แสดงไว้ในภาพ 2.12 จากนั้นทำการลดด้านที่ไม่จำเป็นโดยการรวบส่วนประกอบที่เชื่อมต่อกันอย่างหนาแน่นของ กราฟการจัดกลุ่มการแผ่ทั่ว (c-subsumption digraph) ข้างต้น

รวมกันเป็นEntity เดียวกัน เรียกว่า กราฟการจัดกลุ่มการแผ่ทั่วที่ถูกลดรูป (Reduce c-subsumption digraph) ดังภาพที่ 2.13



ภาพที่ 2.13 กราฟการจัดกลุ่มการแผ่ทั่วที่ถูกลดรูป

จากภาพที่ 2.13 แต่ละโนดปลายทางแทนเซตของเส้นทางโดยการท่องแผนภาพจะท่องจาก โนดเริ่มต้นไปยังโนดปลาย ในการท่องแต่ละครั้งจะได้ หนึ่งเส้นทาง

ปัญหาของงานวิจัย

- ผลการค้นหาที่ได้อาจเป็นไปไม่ได้ ผู้ทำการทดลองต้องทำการพิจารณาหากรณีทดสอบที่เหมาะสมในการหาข้อผิดพลาดของโปรแกรม
- เส้นทางที่ได้ไม่สามารถทดสอบผลได้ทันที ต้องมีวิธีการอ่านเฉพาะเพื่อแสดงผล

2.2.2 การสร้างข้อมูลการทดสอบเพื่อให้ครอบคลุมทุกกิ่ง (Generating Test Data For Branch Coverage) [2]

ขั้นตอนแรกจะทำการค้นหาอินพุทของโปรแกรมทั้งหมดในแต่ละโนดตัดสินใจ แล้วจึงสร้างอินพุทที่สุ่มขึ้นมา และท่องไปตามเส้นทาง แล้วจึงสร้างข้อจำกัดจากเส้นทางที่อินพุทผ่าน จากนั้นจึงทำการเปลี่ยนแปลงค่าอินพุทตามข้อจำกัด แล้วใช้การคำนวณเพื่อเปรียบเทียบว่าทำการเปลี่ยนเส้นทางดีหรือไม่ ทำตามวิธีการดังกล่าวจนได้อินพุทของเส้นทางที่ครอบคลุมกราฟทั้งหมด ข้อดีของวิธีนี้คือขั้นตอนวิธีดังกล่าวครอบคลุมทุกขั้นตอนในการทดสอบเพราะว่าจากขั้นตอนวิธีนี้ จะได้อินพุทของ

เส้นทางที่ครอบคลุมกราฟออกมาเป็นผลลัพธ์เลย ข้อเสียของวิธีนี้คือ เวลาที่ใช้สำหรับขั้นตอนวิธีนี้จะขึ้นอยู่กับความซับซ้อนของโปรแกรมที่จะนำมาทำการหา ซึ่งถ้ายิ่งมีโนดตัดสินใจจำนวนมาก โปรแกรมก็จะยิ่งใช้เวลานานมากในการหาผลลัพธ์ของโปรแกรม

2.2.3 การใช้ขั้นตอนวิธีทางพันธุกรรมและการวิเคราะห์แนวคิดอย่างเป็นทางการเพื่อสร้างข้อมูลการทดสอบให้ครอบคลุมการทดสอบแบบกิ่งโดยอัตโนมัติ (Using a Genetic Algorithm and Formal Concept Analysis to Generate Branch Coverage Test Data Automatically) [3]

Genet จะโหลดโครโมโซมจากไฟล์ที่ผู้ใช้ระบุหรือสุ่มขึ้นมาเพื่อเป็นสายพันธุ์รุ่นแรกแล้วทำตามขั้นตอนดังนี้

1. Genet จะรันโปรแกรมจากการทดสอบแต่ละการทดสอบในสายพันธุ์รุ่นปัจจุบัน แต่ละการทดสอบจะมีการสร้างเส้นทางและบันทึกไว้ในตาราง เมื่อรันครบหมด ถ้ามันยังแสดงกิ่งที่ไม่ครอบคลุมก็ให้ทำในขั้นถัดไป
2. โครโมโซมจะถูกจัดเข้าสู่ Concept genet จะใช้ FCA กับสมาชิกในแต่ละเซต
3. Concept จะถูกจัดอันดับ
4. Genet ให้คะแนนโครโมโซมและเรียงลำดับมัน คะแนนได้มาจากการนับจำนวนครั้งการทดสอบที่เป็น Wining concept คะแนนที่ได้มาจะช่วยสร้างลำดับของโครโมโซมและเลือก Gsz ที่เหมาะสุด
5. Genet ระบุ Crossover operator และ Mutation operator และวนกลับไปขั้นตอนแรกเพื่อนสร้างสายพันธุ์ใหม่

ตัวอย่าง

```
INPUT(X, Y, Z)
BEGIN
  IF (predicate1) { PRINT "1T"  // T2,T3,T4,T6
    IF (predicate2) PRINT "2T"  // T3, T6
    ELSE PRINT "2F" }           // T2,T4
  ELSE { PRINT "1F"             // T1,T5
    IF (predicate3) PRINT "3T"
    ELSE { PRINT "3F"           // T1,T5
      IF (predicate6) PRINT "6T"
      ELSE PRINT "6F"           // T1,T5
      RETURN } }
  IF (predicate4) { PRINT "4T"
    IF (predicate5) PRINT "5T"
    ELSE PRINT "5F" }
  ELSE PRINT "4F"               // T2,T3,T4,T6
END
```

ภาพที่ 2.14 ตัวอย่างโปรแกรมที่ใช้ทดสอบ

จากตัวอย่าง ให้ใช้เอฟซีเอ (FCA) กับสายพันธุ์รุ่นแรกของการทดสอบและประมวลผลเส้นทางเพื่อสร้างแนวคิดตามตารางที่ 2.2 จะเห็นว่าแนวคิดที่ 2 เป็นแนวคิดที่ดีที่สุด คะแนนของ T1 และ T5 จะเป็น 2 และ 1 ตามลำดับ ประชากรชุดสุดท้ายของการทดสอบสำหรับโปรแกรมตัวอย่างนี้จะถูกใช้เพื่อสร้างตารางที่ 2.3 และ 2.4 แนวคิดในตารางที่ 2.3 จะถูกสร้างด้วยชุดตัวเลือก r.1. สำหรับอินเทน (Intent) แนวคิดตามตารางที่ 2.4 จะถูกสร้างด้วยชุดตัวเลือก r.1. สำหรับเอ็กเทน (Extent) ซึ่งจะแสดงเฉพาะแนวคิดที่ใช้ประโยชน์ดังต่อไปนี้

ตารางที่ 2.2 แนวคิดสำหรับสายพันธุ์รุ่นแรก

C	Extent	Intent	Super C	Rank
0	–	–	1, 2, 3	0
1	T3, T6	2T	4	0
2	T1, T5	1F, 3F, 6F	5	2
3	T2, T4	2F	4	0
4	T2, T3, T4, T6	1T, 4F	5	1
5	T1,T2,T3,T4,T5,T6	–	–	0

ตารางที่ 2.3 แนวคิดสำหรับประชากรรุ่นสุดท้าย

Tests	Branch
T11, T7	5F
T1, T5	6F
T9	6T
T1,T5, T9	3F
T10	3T
T1, T10, T5, T9	1F
T2, T4, T11	2F
T2, T3, T4, T6, T10	4F
T8	5T
T3, T6, T7, T8	2T
T7, T8, T11	4T
T2, T3, T4, T6, T7, T8, T11	1T

ตารางที่ 2.4 แนวคิดสำหรับประชากรรุ่นสุดท้าย

Test	Branches
T7	1T, 2T, 4T, 5F
T3, T6	1T, 2T, 4F
T11	1T, 2F, 4T, 5F
T1, T5	1F, 3F, 6F
T9	1F, 3F, 6T
T10	1F, 3T, 4F
T2, T4	1T, 2F, 4F
T8	1T, 2T, 4T, 5T

การเลือกการทดสอบจากตารางที่ 2.3 และ 2.4 เราจะได้ชุดคำตอบของเส้นทางที่น้อยที่สุดเป็นสองชุดเส้นทางดังนี้

(1) {T1, T8, T9, T10, T11}

(2) {T5, T8, T9, T10, T11}

ข้อดี

1.ได้วิธีการสร้างข้อมูลทดสอบให้ครอบคลุมกิ่งที่ง่ายกว่า Branch function

2.ไม่ต้องใช้ Program graph

3.ไม่ขึ้นกับภาษาของโปรแกรม

ข้อเสีย

1.ใช้การคำนวณมากกว่า

2.ยากต่อการครอบคลุม Deeply nested predicate

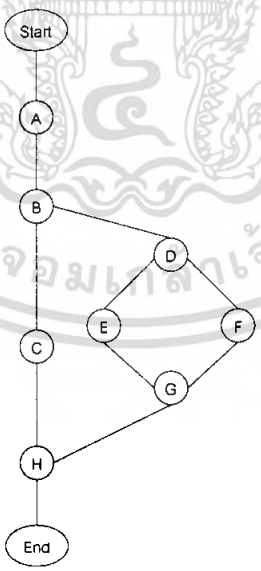
บทที่ 3
ขั้นตอนวิธีการสร้างเส้นทางน้อยที่สุด

จากบทที่ 2 เราได้แนะนำการทดสอบซอฟต์แวร์แบบกิ่งและทฤษฎีกราฟและการค้นหาแล้ว จะเห็นว่าเราทำการแปลงจากโปรแกรม (Source code) มาเป็นกราฟเพื่อทำการหาภาพแบบการทดสอบซึ่งแทนโดย G = (V, E) เมื่อ V เป็นชุดของโนด และ E เป็นชุดของเส้นทางเชื่อมของแต่ละโนด

ในการสร้างเส้นทางที่น้อยที่สุดที่ครอบคลุมในการทดสอบแบบกิ่งนั้นแบ่งได้เป็น 2 ขั้นตอน คือ การสร้างเส้นทางที่เป็นไปได้ทั้งหมด และ การค้นหาเส้นทางที่น้อยที่สุดในการทดสอบแบบกิ่ง ซึ่งรายละเอียดจะกล่าวถึงในหัวข้อที่ 3.1 และ 3.2

3.1 การสร้างเส้นทางที่เป็นไปได้ทั้งหมด

ในการหากรณีทดสอบจากกราฟควบคุมการไหล G=(V,E) ซึ่งมี V เป็นชุดของจุด และ E เป็นชุดของเส้นเชื่อมระหว่างจุด เราจะใช้ตารางช่วยในการค้นหากรณีทดสอบ โดยการแปลงกราฟที่ได้เป็นตารางขนาด n x n ซึ่ง n = |V| และ V= {0,1,2,3,…,n-1} กำหนด $u \in V$ และ $v \in V$ แล้วถ้ามีเส้นเชื่อมจาก u ไป v จะได้ว่าในตารางที่สร้างจากกราฟแถวที่ u และ สดมภ์ที่ v จะเก็บค่า อ้างอิงเส้นทางระหว่าง u และ v ใดๆ ตามเซต E = {1,2,3,4,…,m}เมื่อ m คือ จำนวนเส้นระหว่าง u และ v ใดๆ ดังตัวอย่างในภาพที่ 3.1



ภาพที่ 3.1 แผนภาพควบคุมการไหลเพื่อทำการหากรณีทดสอบ

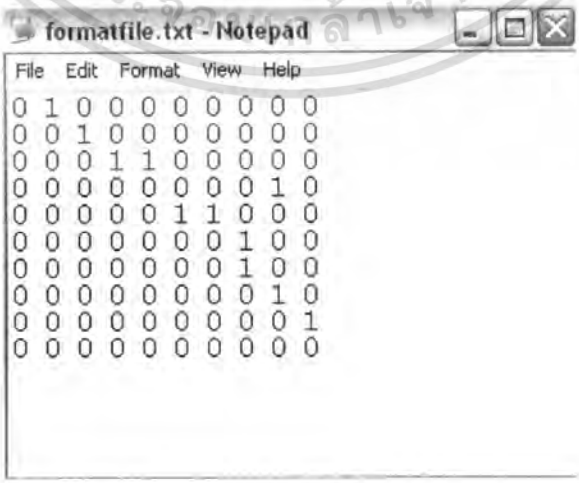
ตารางที่ 3.1 ตารางที่ได้จากการแปลงกราฟ

	start	A	B	C	D	E	F	G	H	end
start	0	1	0	0	0	0	0	0	0	0
A	0	0	1	0	0	0	0	0	0	0
B	0	0	0	1	1	0	0	0	0	0
C	0	0	0	0	0	0	0	0	1	0
D	0	0	0	0	0	1	1	0	0	0
E	0	0	0	0	0	0	0	1	0	0
F	0	0	0	0	0	0	0	1	0	0
G	0	0	0	0	0	0	0	0	1	0
H	0	0	0	0	0	0	0	0	0	1
end	0	0	0	0	0	0	0	0	0	0

3.1.1 ขั้นตอนวิธีการแปลงแผนภาพควบคุมการไหลเป็นไฟล์รูปแบบ

จากภาพที่ 3.1 เราสามารถแปลง จากแผนภาพควบคุมการไหลไปเป็นไฟล์รูปแบบสำหรับใช้ในการทดสอบโปรแกรมได้ โดยมีวิธีการการดังนี้

1. ทำการสร้างตารางขนาด n x n โดย n คือจำนวนของโนด
2. ตรวจโนดที่ i โดย i = 1,2,...n ว่ามีเส้นทางไปถึงโนดที่ j โดย j = i+1,i+2,...,n หรือไม่ ถ้า มีใส่ตัวเลข 1 หรือ ถ้าไม่มีเส้นทางให้ใส่ตัวเลข 0 ลงในตารางช่องที่ ixj
3. ทำการบันทึกลงไฟล์เป็นเท็กซ์ไฟล์ ดังภาพที่ 3.2

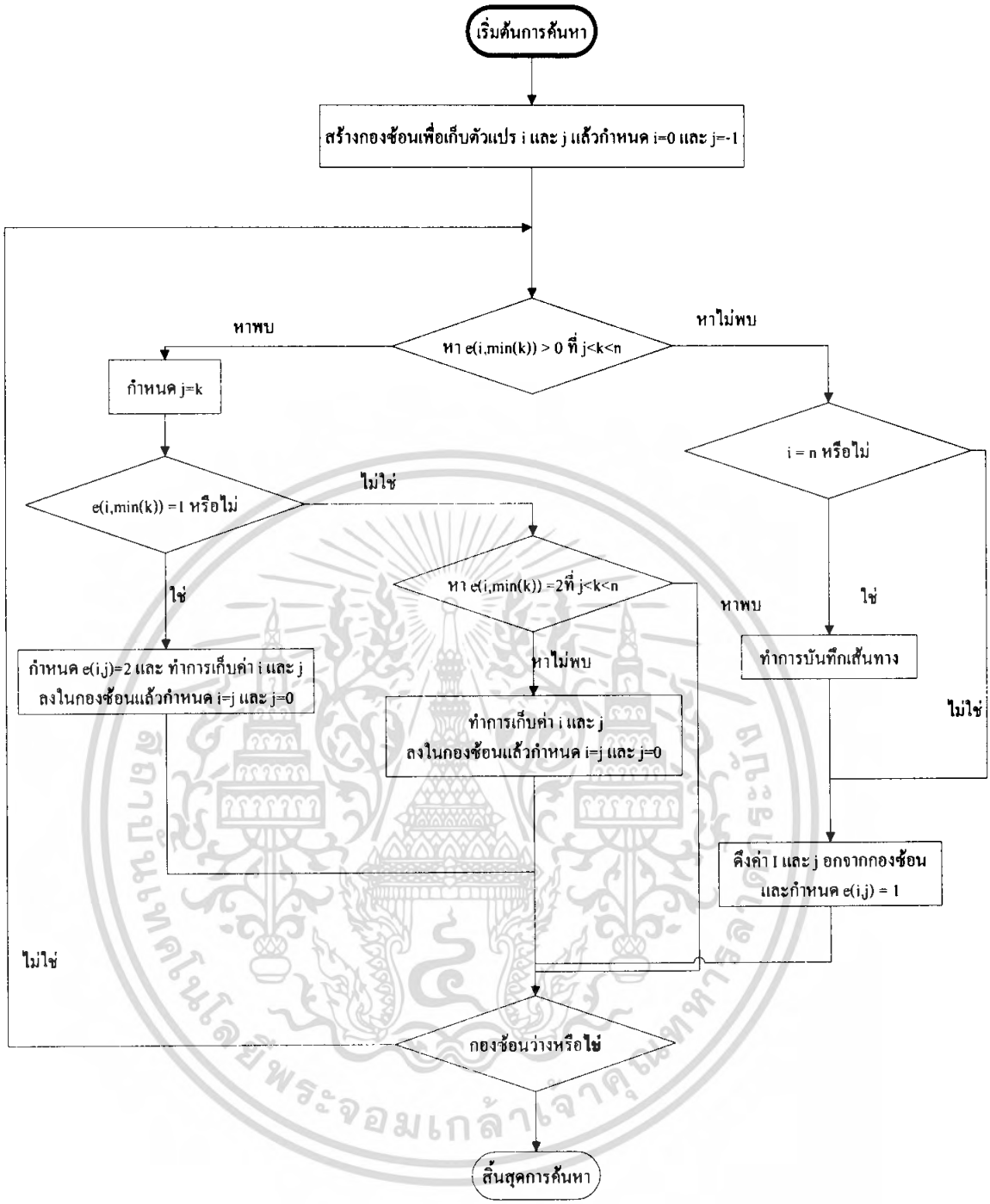


ภาพที่ 3.2 ตัวอย่างไฟล์รูปแบบที่ได้จากการแปลงแผนภาพควบคุมการไหล

3.1.2 ขั้นตอนวิธีการค้นหาเส้นทางทั้งหมดที่เป็นไปได้ในกราฟ

1. จากกราฟควบคุมการไหล G(V,E)ซึ่งV→$\{u_i \mid 0 \le i \le n\text{-}1\}$และE→$\{e_{i,j} \mid 0 \le i \le n\text{-}1, 0 \le j \le n\text{-}1\}$กำหนด S_n และ S_e เป็นชุดของข้อมูลที่เก็บลงในกองซ้อนโดย S_n อ้างถึงกองซ้อนที่เก็บตัวแปร i ที่ใช้อ้างอิง u_i และ S_e อ้างถึงกองซ้อนที่เก็บตัวแปร j ที่ใช้อ้างอิง $e_{i,j}$ ร่วมกับตัวแปร i ซึ่งเริ่มแรกเรากำหนด i=0 และ j=0
2. เพิ่มค่า j เพื่อหา $e_{i,k} > 0$ โดย j<k<n ถ้าเจอให้ทำตามข้อ 3 ถ้าไม่ทำตามข้อ 6
3. กำหนด j=k แล้วทำการตรวจสอบว่า $e_{i,j} \in \{e_{x,y} \mid x = A_n[z], y = A_e[z]$ เมื่อ $z \in I$ และ A_n, A_e เป็นแถวลำดับ (Array) ที่เก็บค่าในกองซ้อน}หรือไม่ ถ้าเป็นสมาชิกให้ทำตามข้อ 4 ถ้าไม่เป็นสมาชิกให้ทำตามข้อ 5
4. เพิ่มค่า j เพื่อหา $e_{i,k} > 0$ โดย j<k<n ถ้าเจอให้ ทำตามข้อ 8 แต่ถ้าไม่เจอให้ทำตามข้อ 5
5. เพิ่มค่า i และ j ลงใน S_n และ S_e ตามลำดับแล้วทำการกำหนดค่า i=j และ j=0 จากนั้นให้ทำตามข้อ 8
6. ตรวจดูว่า u_i ที่ชี้อยู่นั้นเป็นจุดปลายทาง (terminal) หรือไม่ ถ้าใช่ให้ทำการอ่านค่าทั้งหมดที่เก็บอยู่ใน S_n รวมทั้ง i แล้วทำการบันทึกเส้นทาง
7. ทำการดึงค่า i จาก S_n และค่า j จาก S_e
8. ตรวจดูว่าตัวชี้ตำแหน่งของกองซ้อนที่ถูกอ้างโดย $S_n > 0$ หรือไม่ ถ้าน้อยกว่าสิ้นสุดการค้นหา แต่ถ้ามากกว่าให้ทำตามข้อ 2

จากขั้นตอนวิธีการค้นหาเส้นทางทั้งหมดที่เป็นไปได้ในกราฟ สามารถเขียนแผนภาพสายงานได้ดังภาพที่ 3.3



ภาพที่ 3.3 แผนภาพสายงานการหาเส้นทางทั้งหมดที่เป็นไปได้

ตัวอย่างการหาเส้นทาง

จากตารางแสดงความสัมพันธ์ระหว่างจุดข้างต้นโดยแถวแต่ละแถวคือจุดต้นทางและแต่ละสดมภ์คือจุดปลายทางเราจะแสดงการค้นหาโดยใช้ตารางแสดงเหตุการณ์และค่าของตัวแปรที่เปลี่ยนแปลงในระหว่างทำการค้นหา ดังตารางที่ 3.2 นี้

ตารางที่ 3.2 เหตุการณ์และค่าตัวแปรที่เปลี่ยนแปลงในระหว่างทำการค้นหา

เหตุการณ์	i	j	S_n	S_e
ทำการเริ่มต้นโปรแกรม	0	0	{}	{}
หา $e_{0,1}$ พบ	0	1	{}	{}
เพิ่ม i และ j ลงในกองซ้อนแล้วกำหนดค่า i=1,j=0	1	0	{0}	{1}
หา $e_{1,2}$ พบ	1	2	{0}	{1}
เพิ่ม i และ j ลงในกองซ้อนแล้วกำหนดค่า i=2,j=0	2	0	{0,1}	{1,2}
หา $e_{2,3}$ พบ	2	3	{0,1}	{1,2}
เพิ่ม i และ j ลงในกองซ้อนแล้วกำหนดค่า i=3,j=0	3	0	{0,1,2}	{1,2,3}
หา $e_{3,8}$ พบ	3	8	{0,1,2}	{1,2,3}
เพิ่ม i และ j ลงในกองซ้อนแล้วกำหนดค่า i=8,j=0	8	0	{0,1,2,3}	{1,2,3,8}
หา $e_{8,9}$ พบ	8	9	{0,1,2,3}	{1,2,3,8}
เพิ่ม i และ j ลงในกองซ้อนแล้วกำหนดค่า i=9,j=0	9	0	{0,1,2,3,8}	{1,2,3,8,9}
ทำการบันทึกเส้นทาง Start-A-B-C-H-End	9	10	{0,1,2,3,8}	{1,2,3,8,9}
ทำการดึงข้อมูลจาก S_n และ S_e	8	9	{0,1,2,3}	{1,2,3,8}
ทำการดึงข้อมูลจาก S_n และ S_e	3	8	{0,1,2}	{1,2,3}
ทำการดึงข้อมูลจาก S_n และ S_e	2	3	{0,1}	{1,2}
หา $e_{2,4}$ พบ	2	4	{0,1}	{1,2}
เพิ่ม i และ j ลงในกองซ้อนแล้วกำหนดค่า i=4,j=0	4	0	{0,1,2}	{1,2,4}
หา $e_{4,5}$ พบ	4	5	{0,1,2}	{1,2,4}

เพิ่ม i และ j ลงในกองซ้อนแล้วกำหนดค่า i=5,j=0	5	0	{0,1,2,4}	{1,2,4,5}
หา $e_{5,7}$ พบ	5	7	{0,1,2,4}	{1,2,4,5}
เพิ่ม i และ j ลงในกองซ้อนแล้วกำหนดค่า i=7,j=0	7	0	{0,1,2,4,5}	{1,2,4,5,7}
หา $e_{7,8}$ พบ	7	8	{0,1,2,4,5}	{1,2,4,5,7}
เพิ่ม i และ j ลงในกองซ้อนแล้วกำหนดค่า i=8,j=0	8	0	{0,1,2,4,5,7}	{1,2,4,5,7,8}
หา $e_{8,9}$ พบ	8	9	{0,1,2,4,5,7}	{1,2,4,5,7,8}
เพิ่ม i และ j ลงในกองซ้อนแล้วกำหนดค่า i=9,j=0	9	0	{0,1,2,4,5,7,8}	{1,2,4,5,7,8,9}
ทำการบันทึกเส้นทาง Start-A-B-D-E-G-H-End	9	10	{0,1,2,4,5,7,8}	{1,2,4,5,7,8,9}
ทำการดึงข้อมูลจาก S_n และ S_e	8	9	{0,1,2,4,5,7}	{1,2,4,5,7,8}
ทำการดึงข้อมูลจาก S_n และ S_e	7	8	{0,1,2,4,5}	{1,2,4,5,7}
ทำการดึงข้อมูลจาก S_n และ S_e	5	7	{0,1,2,4}	{1,2,4,5}
ทำการดึงข้อมูลจาก S_n และ S_e	4	5	{0,1,2}	{1,2,4}
หา $e_{4,6}$ พบ	4	6	{0,1,2}	{1,2,4}
เพิ่ม i และ j ลงในกองซ้อนแล้วกำหนดค่า i=6,j=0	6	0	{0,1,2,4}	{1,2,4,6}
หา $e_{6,7}$ พบ	6	7	{0,1,2,4}	{1,2,4,6}
เพิ่ม i และ j ลงในกองซ้อนแล้วกำหนดค่า i=7,j=0	7	0	{0,1,2,4,6}	{1,2,4,6,7}
หา $e_{7,8}$ พบ	7	8	{0,1,2,4,6}	{1,2,4,6,7}
เพิ่ม i และ j ลงในกองซ้อนแล้วกำหนดค่า i=8,j=0	8	0	{0,1,2,4,6,7}	{1,2,4,6,7,8}
หา $e_{8,9}$ พบ	8	9	{0,1,2,4,6,7}	{1,2,4,6,7,8}
เพิ่ม i และ j ลงในกองซ้อนแล้วกำหนดค่า i=9,j=0	9	0	{0,1,2,4,6,7,8}	{1,2,4,6,7,8,9}
ทำการบันทึกเส้นทาง Start-A-B-D-F-G-H-End	9	10	{0,1,2,4,6,7,8}	{1,2,4,6,7,8,9}
ทำการดึงข้อมูลจาก S_n และ S_e	8	9	{0,1,2,4,6,7}	{1,2,4,6,7,8}

ทำการดึงข้อมูลจาก S_n และ S_e	7	8	{0,1,2,4,6}	{1,2,4,6,7}
ทำการดึงข้อมูลจาก S_n และ S_e	6	7	{0,1,2,4}	{1,2,4,6}
ทำการดึงข้อมูลจาก S_n และ S_e	4	6	{0,1,2}	{1,2,4}
ทำการดึงข้อมูลจาก S_n และ S_e	2	4	{0,1}	{1,2}
ทำการดึงข้อมูลจาก S_n และ S_e	1	2	{0}	{1}
ทำการดึงข้อมูลจาก S_n และ S_e	0	1	{}	{}
สิ้นสุดการค้นหา	-	-		

3.1.3 การบันทึกเส้นทางที่หามาได้

ในการบันทึกเส้นทางจะทำการบันทึกตัวอ้างอิงของ u_i และ $e_{i,j}$ ในภาพค่าของ E(i,j) โดยที่ i สามารถอ้างอิงถึงโนดและ i กับ j สามารถอ้างอิงถึงเส้นทางที่ออกจากโนด โดยเก็บลำดับการท่องไปใน u_i ของเส้นทาง และ เราจะสร้างตารางขนาด m X p เมื่อ m คือ จำนวนเส้นทางระหว่าง u และ v และ p คือ จำนวนเส้นทางที่หาได้ซึ่งแต่ละแถวจะอ้างอิงถึงเส้นทาง และแต่ละสดมภ์อ้างอิงถึงจำนวนครั้งที่ใช้เส้นทางในกราฟ เพื่อใช้เปรียบเทียบหาตารางที่ใช้ในการทดสอบแบบกิ่ง ดังตัวอย่างในตารางที่ 3.3

ตารางที่ 3.3 ตัวอย่างตารางบันทึกเส้นทาง

E	1	2	3	4	5	…	…	m-1	m
Test 1	1	1	0	1	0			0	1
Test 2	1	2	0	1	0			0	0
…									
….									
….									
Test p-1	1	0	1	1	0			1	1
Test p	1	0	2	1	0			1	1

3.2 การค้นหาชุดเส้นทางที่น้อยที่สุด

จากกราฟที่เกิดจากโปรแกรม G=(V,E) กำหนดเซต E เป็นเซตของเส้นระหว่างจุดแต่ละ จุดซึ่ง $E = \{e_{i,j} | 0 \le i \le n-1, 0 \le j \le n-1\}$ และให้ T เป็นเซตของเส้นทาง โดย $T = \{t_i | 0 \le i < p\}$ ซึ่ง m คือ จำนวนเส้นทาง ที่หาได้ และ $t_i \subseteq .E$ โดยเราจะหาจำนวน t_i ที่น้อยที่สุดที่ทำให้เกิด $E = t_i \cup t_j \cup t_k \ldots$

จากตารางที่เก็บเส้นทางเพื่อใช้ในการหาตารางที่ใช้ในการทดสอบแบบกิ่งที่กล่าวมาข้างต้น เราทำการตรวจว่าครอบคลุมหรือไม่โดย นำค่าในสดมภ์ในแต่ละเส้นทางที่ได้เลือกมาบวกกัน หากค่าที่บวกกันทุกค่ามากกว่า 1 จะเกิดตารางที่ใช้ในการทดสอบแบบกิ่งดังตัวอย่าง โดยกำหนดให้จำนวนเส้นทางระหว่างจุดมี m เส้น

ตารางที่ 3.4 ตัวอย่างการหาเส้นทางที่ครอบคลุมกราฟทั้งหมด

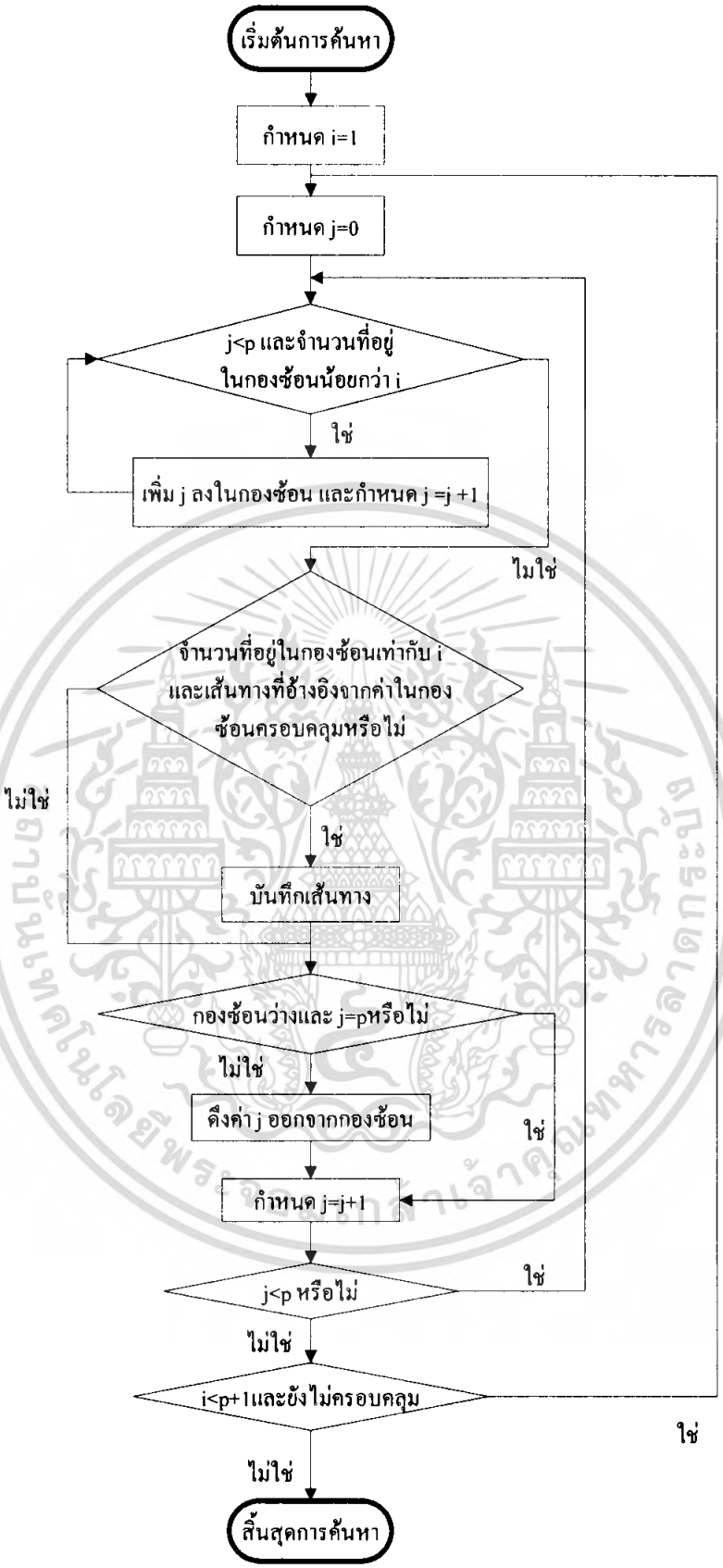
เส้นทาง	เส้นทางระหว่างจุด								
	1	2	3	4	…	…	…	m-1	M
Path A	1	0	1	1	…	…	…	1	1
Path B	1	2	0	1	…	…	…	0	1
รวม	2	2	1	2	…	…	…	1	2

เราจะทำการหาเส้นทางเส้นทางโดยเริ่มเลือกชุดเส้นทางมาทำการตรวจหาโดยเริ่มเลือกจากชุดที่มีจำนวนน้อยไปจำนวนมาก เพื่อหาจำนวนเส้นทางที่น้อยที่สุดที่ครอบคลุมกราฟ หากชุดเส้นทางที่ครอบคลุมกราฟที่น้อยที่สุดมี i เส้นทาง เราจะทำการตรวจหาเส้นทางที่ครอบคลุมกราฟ $\binom{m}{1} + \binom{m}{2} + \binom{m}{3} + \ldots + \binom{m}{i-1} + \binom{m}{i}$ ครั้ง

ขั้นตอนวิธีการหาชุดเส้นทางที่ครอบคลุมกราฟโดยใช้เส้นทางน้อยที่สุด

1. กำหนดตัวแปร i =1 เพื่อกำหนดจำนวนกรณีทดสอบ ที่จะนำมาเปรียบเทียบและกำหนด j = 0 เพื่อเป็นตัวอ้างอิงเส้นทางที่เลือกอยู่ แล้วสร้าง S_p เพื่อเก็บเลขอ้างอิงเส้นทางหรือค่า j
2. หาก $j \notin S_p$ ให้เพิ่ม j ลงใน S_p และกำหนด j=j+1 ทำซ้ำจนกว่า $|S_p| \geq i$ หรือ $j \geq p$ เมื่อ p คือจำนวนเส้นทางทั้งหมด
3. หาก $|S_p| \geq i$ ให้ทำการเปรียบเทียบว่าชุดเส้นทางนั้นครอบคลุมกราฟหรือไม่ ถ้าครอบคลุมให้กับการบันทึกและกำหนดตัวบ่งชี้ (flag) ว่าพบแล้ว
4. หาก $|S_p| > 0$ ให้ดึงค่า j ออกมาแล้วกำหนด j=j+1
5. หาก $|S_p| = 0$ และ $j \geq p$ แล้วให้ทำตามข้อ 6 ถ้าไม่กลับไปทำตามข้อ 2
6. หากตัวบ่งชี้นั้นแจ้งว่า'พบแล้ว'ให้สิ้นสุดการหา ถ้าไม่ให้กำหนด i=i+1 และ j=0 แล้วทำตามข้อ 2

จากขั้นตอนวิธีการหาชุดเส้นทางที่ครอบคลุมกราฟโดยใช้เส้นทางน้อยที่สุด สามารถเขียนแผนภาพสายงานได้ดังภาพที่ 3.4



ภาพที่ 3.4 แผนภาพสายงานการหาชุดเส้นทางที่ครอบคลุมกราฟโดยใช้เส้นทางน้อยที่สุด

ตัวอย่างการหาชุดเส้นทางที่ครอบคลุมกราฟโดยใช้เส้นทางน้อยที่สุด

เราจะแสดงการหาชุดเส้นทางที่ครอบคลุมกราฟซึ่งมีจำนวนเส้นทางน้อยที่สุดโดยใช้ตารางแสดงเหตุการณ์และค่าของตัวแปรที่เปลี่ยนแปลงในระหว่างทำการหา จากตัวอย่างการหาเส้นทางทั้งหมดของกราฟเราได้เส้นทางของกราฟมา 3 เส้นทาง คือ

ตารางที่ 3.5 ชุดของเส้นทางที่ครอบคลุมกราฟ

ลำดับ	เส้นทาง	เส้นทางระหว่างจุด										
		1	2	3	4	5	6	7	8	9	10	11
0	Start-A-B-C-H-End	1	1	1		1						1
1	Start-A-B-D-E-G-H-End	1	1		1		1		1		1	1
2	Start-A-B-D-F-G-H-End	1	1		1			1		1	1	1

กำหนดตัวแปร i เพื่อกำหนดจำนวนเส้นทางที่เลือกใช้เปรียบเทียบ,ตัวแปร j ใช้อ้างอิงเส้นทางบนตารางข้างบน, S_p แสดงชุดข้อมูลในกองซ้อน (Stack) ที่เก็บค่า j และตัวแปร p_count แสดงจำนวนเส้นทางระหว่างจุดที่ชุดเส้นทางครอบคลุม ซึ่งถ้าเท่ากับ 11 จึงจะครอบคลุมกราฟทั้งหมด

ตารางที่ 3.6 เหตุการณ์การหาชุดเส้นทางที่ครอบคลุมกราฟโดยใช้เส้นทางน้อยที่สุด

เหตุการณ์	i	j	S_p	p_count
เริ่มทำการเปรียบเทียบ	1	0	{}	0
ทำการเพิ่ม j ลงใน กองซ้อน	1	0	{0}	5
เปรียบเทียบชุดเส้นทางว่าครอบคลุมหรือไม่	1	1	{0}	5
ทำการดึงค่า j จากกองซ้อนแล้วกำหนด j=j+1	1	1	{}	5
ทำการเพิ่ม j ลงในกองซ้อน	1	1	{1}	7
เปรียบเทียบชุดเส้นทางว่าครอบคลุมหรือไม่	1	2	{1}	7
ทำการดึงค่า j จากกองซ้อนแล้วกำหนด j=j+1	1	2	{}	0
ทำการเพิ่ม j ลงในกองซ้อน	1	2	{2}	7
เปรียบเทียบชุดเส้นทางว่าครอบคลุมหรือไม่	1	3	{2}	7
ทำการดึงค่า j จากกองซ้อนแล้วกำหนด j=j+1	1	3	{}	0
กำหนด i=i+1 และกำหนด j=0	2	0	{}	0
ทำการเพิ่ม j ลงในกองซ้อน	2	0	{0}	5
ทำการเพิ่ม j ลงในกองซ้อน	2	1	{0,1}	9

เปรียบเทียบชุดเส้นทางว่าครอบคลุมหรือไม่	2	2	{0,1}	9
ทำการดึงค่า j จากกองซ้อนแล้วกำหนด j=j+1	2	2	{0}	5
ทำการเพิ่ม j ลงในกองซ้อน	2	2	{0,2}	9
เปรียบเทียบชุดเส้นทางว่าครอบคลุมหรือไม่	2	3	{0,2}	9
ทำการดึงค่า j จากกองซ้อนแล้วกำหนด j=j+1	2	3	{0}	5
ทำการดึงค่า j จากกองซ้อนแล้วกำหนด j=j+1	2	1	{}	0
ทำการเพิ่ม j ลงในกองซ้อน	2	1	{1}	7
ทำการเพิ่ม j ลงในกองซ้อน	2	2	{1,2}	9
เปรียบเทียบชุดเส้นทางว่าครอบคลุมหรือไม่	2	2	{1,2}	9
ทำการดึงค่า j จากกองซ้อนแล้วกำหนด j=j+1	2	3	{1}	7
ทำการดึงค่า j จากกองซ้อนแล้วกำหนด j=j+1	2	2	{}	0
หา j มาใส่ไม่ได้	2	3	{}	0
กำหนด i=i+1 และกำหนด j=0	3	0	{}	0
ทำการเพิ่ม j ลงในกองซ้อน	3	0	{0}	5
ทำการเพิ่ม j ลงในกองซ้อน	3	1	{0,1}	9
ทำการเพิ่ม j ลงในกองซ้อน	3	2	{0,1,2}	11
เปรียบเทียบชุดเส้นทางว่าครอบคลุมหรือไม่	3	2	{0,1,2}	11
บันทึกชุดเส้นทางและแจ้งว่าพบเส้นทางแล้ว	3	2	{0,1,2}	11
ทำการดึงค่า j จากกองซ้อนแล้วกำหนด j=j+1	3	3	{0,1}	9
ทำการดึงค่า j จากกองซ้อนแล้วกำหนด j=j+1	3	2	{0}	5
ทำการเพิ่ม j ลงในกองซ้อน	3	2	{0,2}	9
ทำการดึงค่า j จากกองซ้อนแล้วกำหนด j=j+1	3	3	{0}	5
ทำการดึงค่า j จากกองซ้อนแล้วกำหนด j=j+1	3	1	{}	0
ทำการเพิ่ม j ลงในกองซ้อน	3	1	{1}	7
ทำการเพิ่ม j ลงในกองซ้อน	3	2	{1,2}	9
ทำการดึงค่า j จากกองซ้อนแล้วกำหนด j=j+1	3	3	{1}	7
ทำการดึงค่า j จากกองซ้อนแล้วกำหนด j=j+1	3	2	{}	0
หา j มาใส่ไม่ได้	3	3	{}	0
สิ้นสุดการค้นหา	3	3	{}	0

จากตัวอย่างเราทำการเปรียบเทียบ 7 ครั้งและได้ชุดเส้นทางที่ครอบคลุมโดยใช้เส้นทางน้อยที่สุด คือ {0,1,2}

บทที่ 4

ผลการทดลอง

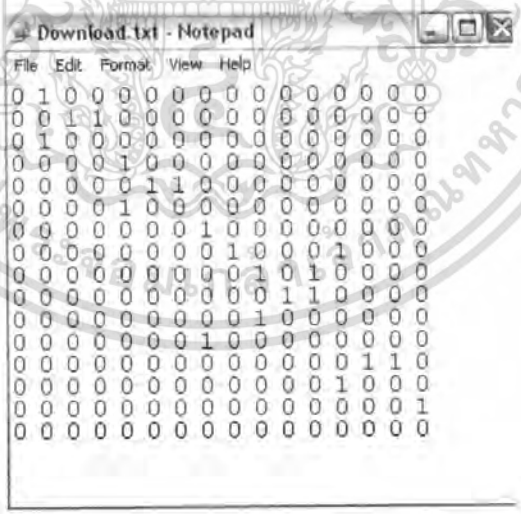
จากบทที่ 3 ได้นำเสนอรายละเอียดของขั้นตอนวิธีการหาชุดของเส้นทางที่น้อยที่สุดสำหรับการทดสอบซอฟต์แวร์แบบกึ่ง ในบทนี้เราได้พัฒนาเครื่องมือเพื่อทำการประมวลผลขั้นตอนวิธีที่นำเสนอโดยแปลงโปรแกรมไปเป็นแผนภาพควบคุมการไหลแล้วเก็บลงเป็นไฟล์ภาพแบบ แล้วทำการหาเส้นทางทั้งหมดที่เป็นไปได้ จากนั้นจึงทำการหาจำนวนเส้นทางที่น้อยที่สุดที่ครอบคลุมกราฟ

4.1 ผลการทดสอบขั้นตอนวิธี

จากขั้นตอนวิธีในบทที่ 3 เราทดสอบสามารถวิเคราะห์ประสิทธิภาพของขั้นตอนวิธีที่ได้โดยนำโปรแกรมจะใช้ทดสอบมาเป็นข้อมูลเข้าเพื่อวัดประสิทธิภาพของโปรแกรมว่าจะได้ผลลัพธ์ดังที่ต้องการหรือไม่ ดังตัวอย่างต่อไปนี้

ตัวอย่างที่ 1 โปรแกรม Download

โปรแกรมสำหรับเขียนข้อมูลจาก เว็บเอเอสพี (ASP) และอ่านข้อมูลจากเท็กซ์ไฟล์เป็นชนิดสตริง (String type) ลงในเท็กซ์ไฟล์ โดยโปรแกรมนี้มีลูปจำนวนสามลูป ข้อมูลเข้าเป็นดังภาพที่ 4.1 โดยรายละเอียดของโปรแกรมอยู่ในภาคผนวก ก

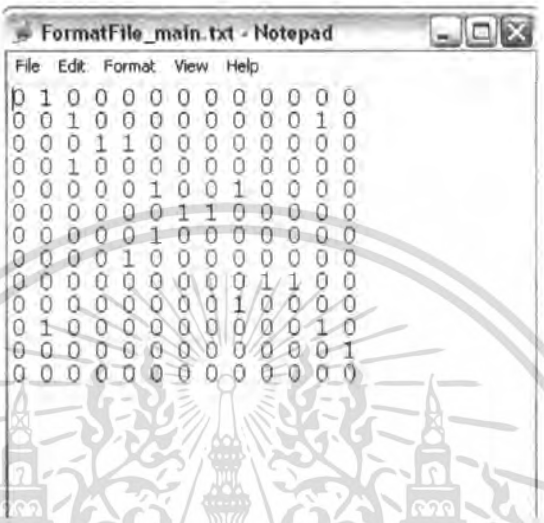


ภาพที่ 4.1 ไฟล์ภาพแบบของโปรแกรม Download

จากไฟล์ข้อมูลเข้าในภาพที่ 4.1 เมื่อเราแปลงโปรแกรมจากซอร์สโค้ด ไฟล์ข้อมูลเข้าจะได้จำนวนโนดในไฟล์ข้อมูลเข้าทั้งหมดสิบหกโนด หาจำนวนเส้นทางได้สามสิบสองเส้นทาง และหาจำนวนชุดของเส้นทางที่เหมาะสมได้ทั้งหมดยี่สิบเจ็ดเส้นทาง

ตัวอย่างที่ 2 โปรแกรม FormatFile

โปรแกรมสำหรับสร้างไฟล์ภาพแบบ โดยโปรแกรมนี้จะมีลูปจำนวนสี่ลูป ข้อมูลเข้าเป็นดังภาพ 4.2 โดยรายละเอียดของโปรแกรมนี้จะอยู่ในภาคผนวก ข



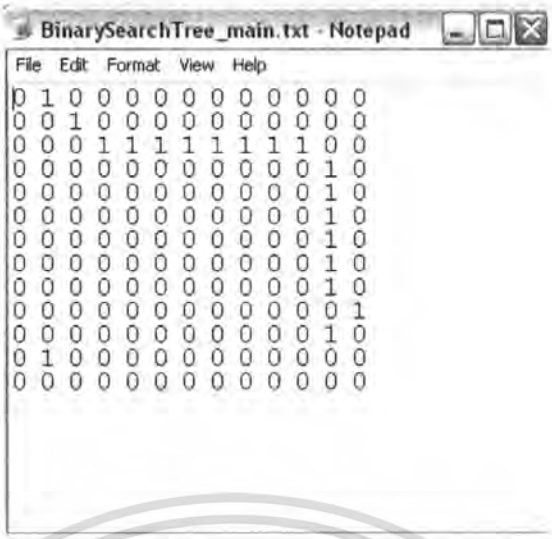
ภาพที่ 4.2 ไฟล์ภาพแบบของโปรแกรม FormatFile

จากไฟล์ข้อมูลเข้าในภาพที่ 4.2 เมื่อเราแปลงโปรแกรมจาก source code ไฟล์ข้อมูลเข้าจะได้จำนวนโนดในไฟล์ข้อมูลเข้าทั้งหมดสิบสามโนด หาจำนวนเส้นทางได้ยี่สิบห้าเส้นทาง และหาจำนวนชุดของเส้นทางที่เหมาะสมได้ทั้งหมดสี่สิบห้าเส้นทาง

ตัวอย่างที่ 3 โปรแกรม BinarySearchTree

เป็นโปรแกรมการค้นหาแบบ Tree โดยโปรแกรมนี้จะแบ่งการเป็นเมธอดในบทนี้จะเสนอ Main เมธอด และ Delete เมธอด

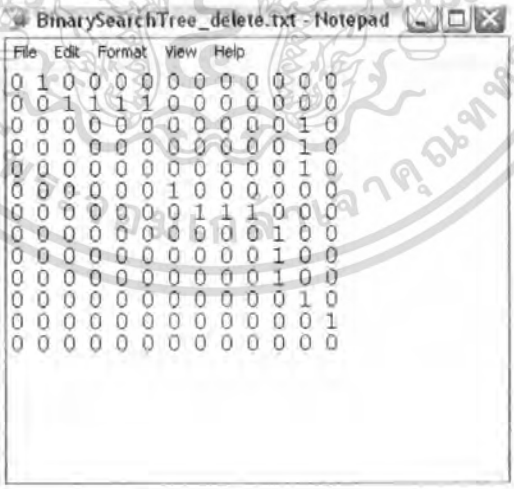
Main เมธอดเป็นเมธอดหลักของโปรแกรม ในเมธอดนี้ มีจำลูปจำนวนหนึ่งลูป ไฟล์ข้อมูลเข้าเป็นดังภาพ 4.3



ภาพที่ 4.3 ไฟล์ภาพแบบของเมธอด Main ของโปรแกรม BinarySearchTree

จากไฟล์ข้อมูลเข้าในภาพที่ 4.3 เมื่อเราแปลงโปรแกรมจาก Source code ไฟล์ข้อมูลเข้าจะได้จำนวนโนดในไฟล์ข้อมูลเข้าทั้งหมดสิบห้าโนด หาจำนวนเส้นทางได้เจ็ดเส้นทาง และหาจำนวนชุดของเส้นทางที่เหมาะสมได้ทั้งหมดสองเส้นทาง

Delete เมธอดเป็นเมธอดที่ใช้ลบโนดใน Tree โดยเมธอดนี้มีไม่มีการวนลูป ไฟล์ข้อมูลเข้าเป็นดังภาพ 4.4

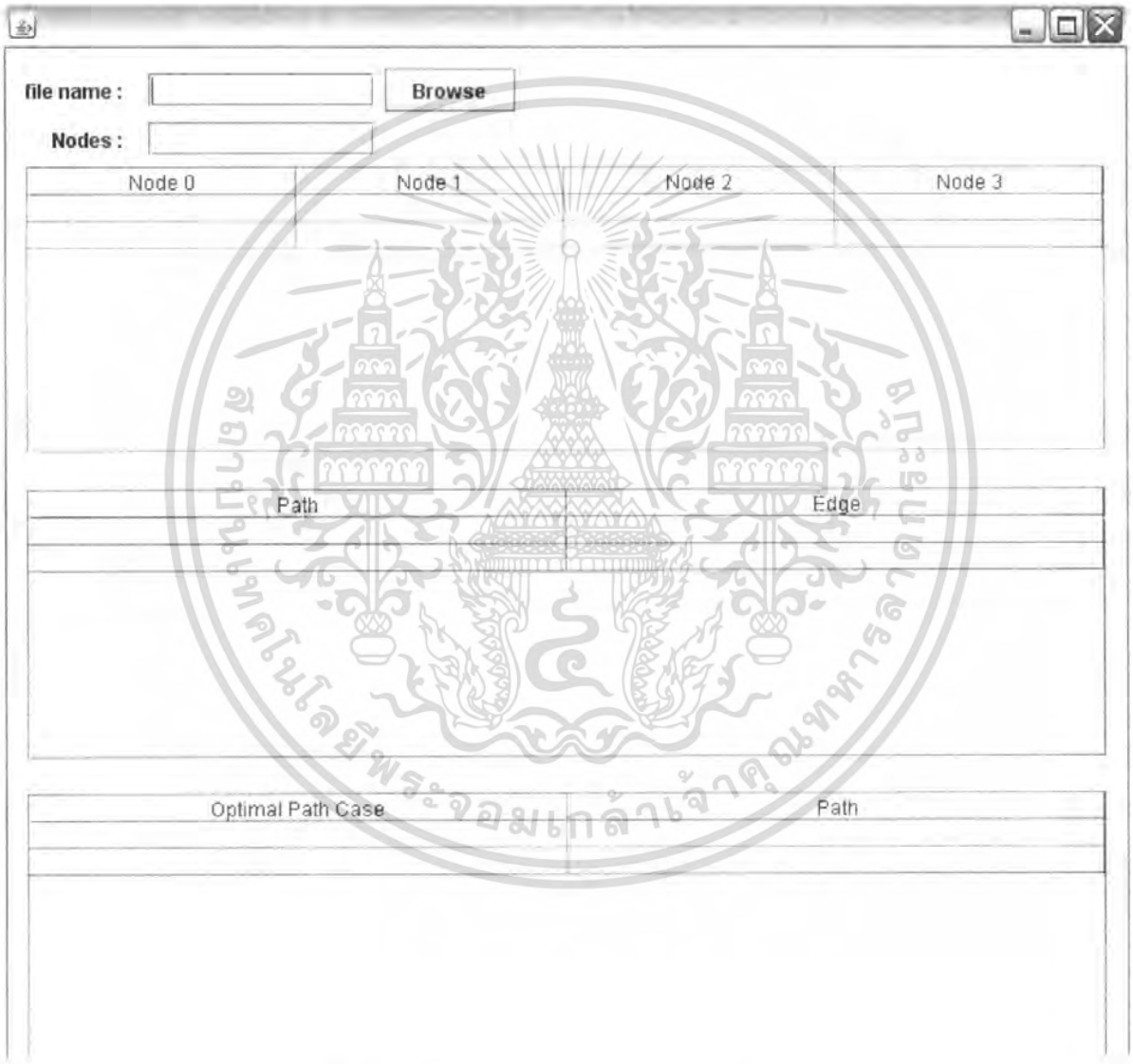


ภาพที่ 4.4 ไฟล์ภาพแบบของเมธอด Delete ของโปรแกรม BinarySearchTree

จากไฟล์ข้อมูลเข้าในภาพที่ 4.4 เมื่อเราแปลงโปรแกรมจาก source code ไฟล์ข้อมูลเข้าจะได้จำนวนโนดในไฟล์ข้อมูลเข้าทั้งหมดสิบสามโนด หาจำนวนเส้นทางได้หกเส้นทาง และหาจำนวนชุดของเส้นทางที่เหมาะสมได้ทั้งหมดหนึ่งเส้นทาง

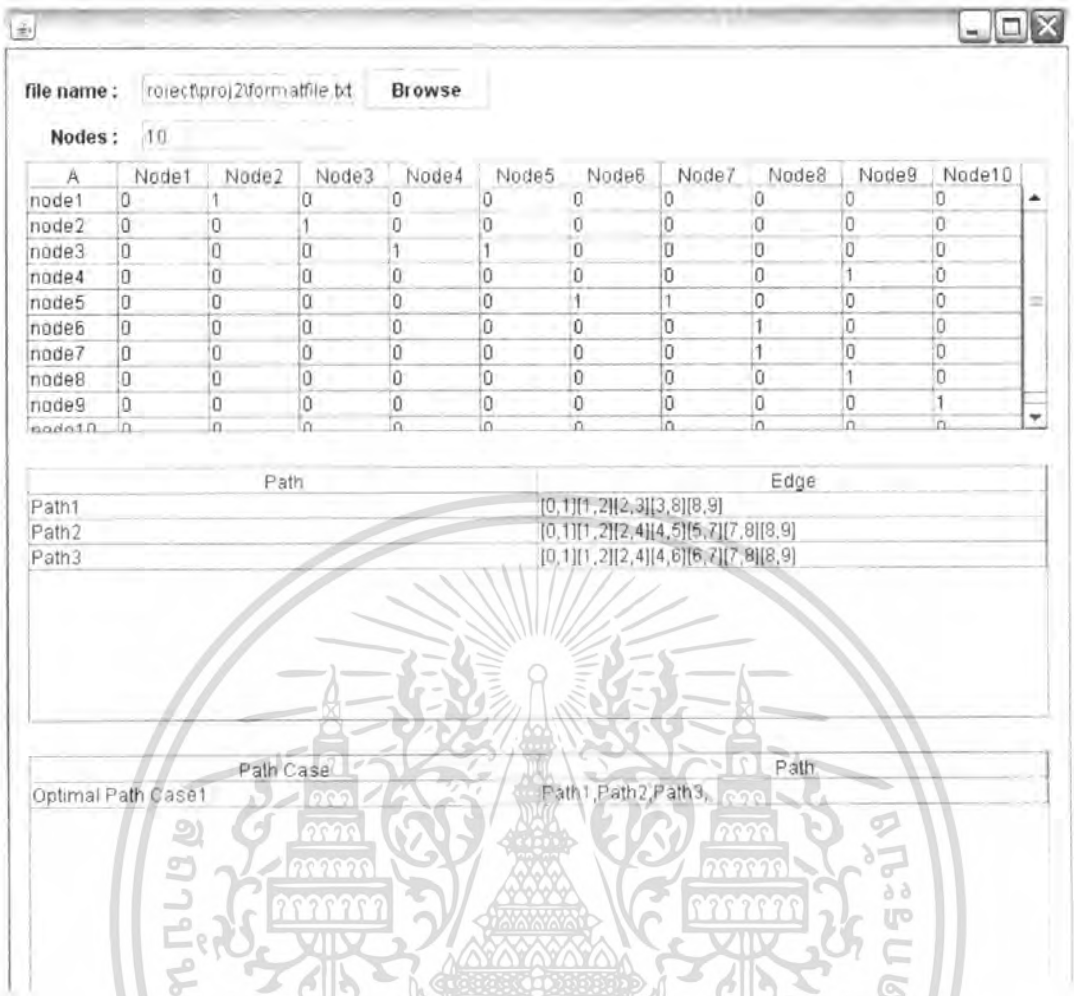
รายละเอียดของโปรแกรม BinarySearchTree ทั้งหมด อยู่ในภาคผนวก ค

จากผลของขั้นตอนวิธีเราได้พัฒนาโปรแกรมสำหรับแสดงผลตามขั้นตอนวิธีที่ได้เสนอขึ้นโดยจะได้โปรแกรมดังภาพที่ 4.5



ภาพที่ 4.5 โปรแกรมหาชุดของเส้นทางที่น้อยที่สุด

จากภาพที่ 4.5 โปรแกรมหาชุดของเส้นทางที่เหมาะสม ซึ่งจะทำการหาเส้นทางทั้งหมด และ หาชุดของเส้นทางที่ครอบคลุมเส้นด้านทั้งหมด โดยมีข้อมูลเข้าคือไฟล์ภาพแบบจากภาพที่ 3.2 โดยผลที่ได้ดังภาพที่ 4.6



ภาพที่ 4.6 ผลที่ได้จากการรันโปรแกรมหาชุดของเส้นทางที่น้อยที่สุด

จากภาพที่ 4.6 โปรแกรมจะแสดงจำนวนโนดทั้งหมด ในตารางแรกจะแสดงความสัมพันธ์ระหว่างโนด ในตารางที่สอง แสดงรายการเส้นทางจากโนดเริ่มต้นไปยังโนดสิ้นสุด และในตารางสุดท้ายแสดงชุดของเส้นทางที่เหมาะสมซึ่งครอบคลุมด้านทั้งหมด

จากตัวอย่างข้างต้นเราสามารถสรุปความซับซ้อนของขั้นตอนวิธีการค้นหาเส้นทางที่น้อยที่สุดสำหรับการทดสอบแบบกิ่งได้ดังนี้

4.2 การวัดประสิทธิภาพของขั้นตอนวิธี

จากขั้นตอนวิธีที่นำเสนอจะทำการเลือกเส้นทางมาทำการหาว่าเส้นทางที่เราเลือกมานั้นครอบคลุมหรือไม่ ดังนั้นกรณีแย่ที่สุดของการเลือกเส้นทางนั้น จะต้องทำการเลือกเส้นทางจากกรีเลือกเส้นทางน้อยไปจนถึงเลือกเส้นทางที่ได้มาทั้งหมดมาเปรียบเทียบ ดังนั้นเราจะได้การวัดประสิทธิภาพของการหาชุดเส้นทางที่น้อยที่สุด

กรณีที่แย่ที่สุด

$$f(n) = \sum_{i=1}^{n} \binom{n}{i}$$

$$= \frac{n!}{1!(n-1)!} + \frac{n!}{2!(n-2)!} + \frac{n!}{3!(n-3)!} + \ldots + \frac{n!}{0!(n)!}$$

จะได้ว่า $f(n) = \left(\frac{n!}{1(n-1)!} + \frac{n!}{2!(n-2)!} + \frac{n!}{3!(n-3)!} + \ldots + \frac{n!}{(n-\frac{n}{2})!(n-\frac{n}{2})!} \right) + \frac{n!}{n!}$ เมื่อ n เป็นจำนวนเต็มคู่และ

$$f(n) = \left(\frac{n!}{1(n-1)!} + \frac{n!}{2!(n-2)!} + \frac{n!}{3!(n-3)!} + \ldots + \frac{n!}{(n-\frac{n-1}{2})!(n-\frac{n+1}{2})!} \right) + \frac{n!}{n!}$$

เมื่อ n เป็นจำนวนเต็มคี่

ดังนั้นจะได้ว่า $O(f(n)) = n^{\frac{n}{2}}$

กรณีที่ดีที่สุด

ในกรณีที่ดีที่สุดคือเลือกเพียงเส้นทางเดียวแล้วครอบคลุม รวมถึงกรณีที่มีลูป ดังนั้นเราจะได้ว่า

$$\Omega(n) = 1$$

โดยที่ n คือ จำนวนเส้นทางทั้งหมด

บทที่ 5
สรุปผลการวิจัย การอภิปราย และข้อเสนอแนะ

จากบทที่ 4 ได้นำเสนอประสิทธิภาพของขั้นตอนวิธีที่ได้เสนอผ่านทางค่าความซับซ้อนของโปรแกรม ในบทนี้จึงได้ทำการเปรียบเทียบขั้นตอนวิธีที่เสนอกับขั้นตอนวิธีอื่นที่ได้กล่าวถึงแล้วในบทที่ 2 และกล่าวถึงข้อเสนอแนะสำหรับงานในอนาคต

5.1 สรุปผลการวิจัย

เนื่องจากปัจจุบันการทดสอบซอฟต์แวร์นั้นมีความสำคัญ โดยที่การทดสอบซอฟต์แวร์นั้นจะช่วยลดข้อผิดพลาดที่อาจจะเกิดขึ้นจากการพัฒนาซอฟต์แวร์ ดังนั้นเราจึงพัฒนาขั้นตอนวิธีในการหาเส้นทางที่น้อยที่สุดในการทดสอบแบบกิ่ง เพื่อช่วยในการทดสอบโปรแกรมแบบกิ่งดังที่กล่าวไว้ในข้างต้น

วิธีการค้นหาชุดเส้นทางสำหรับการทดสอบแบบกิ่งนั้น เราทำการรับกราฟเส้นทางการตัดสินใจ (DD-PATH) เข้ามาแล้วทำการค้นหาเส้นทางที่เป็นไปได้ทั้งหมดจากกราฟที่รับเข้ามา แล้วจึงทำการจับคู่หาเส้นทางที่น้อยที่สุดที่ครอบคลุมกราฟ เมื่อเราได้ชุดของเส้นทางที่น้อยที่สุดแล้วเราจะแสดงผลลัพธ์ที่ได้เป็นจำนวนชุดของเส้นทางที่น้อยที่สุดซึ่งอาจจะมีมากกว่าหนึ่งชุดก็ได้ ซึ่งผลจากการทดสอบโดยการพัฒนาซอฟต์แวร์โดยใช้วิธีการดังกล่าวเป็นหลักทำให้ได้ผลการทดสอบดังที่กล่าวไว้ในบทที่ 4 และสามารถทดสอบหาความซับซ้อนของระบบทำให้สามารถประมวลผลได้

5.2 การวิจารณ์หรืออภิปราย

จากขั้นตอนวิธีนี้ได้ทำการพัฒนาขึ้นหลังจากได้ศึกษาผลการทดลองและงานวิจัยที่เกี่ยวข้องอื่นๆ แล้ว เมื่อเปรียบเทียบขั้นตอนวิธีที่ได้พัฒนาขึ้นมากับงานวิจัยการสร้างข้อมูลการทดสอบเพื่อให้ครอบคลุมทุกกิ่ง (Generating Test Data For Branch Coverage) [2] นั้นจะพบว่า วิธีที่การสร้างข้อมูลการทดสอบเพื่อให้ครอบคลุมทุกกิ่งกล่าวถึงในบทที่ 2 นั้นมีข้อเสียคือวิธีนี้จะทำการสุ่มเส้นทางขึ้นมา ทำให้ไม่รู้จำนวนรอบที่แน่นอน บางกรณีอาจจะส่งผลทำให้ไม่ครอบคลุมทุกกิ่งก็เป็นไปได้ หรือบางกรณีอาจจะก่อให้เกิดจำนวนรอบของการวนซ้ำมากมายมหาศาลก็ได้ โดยประสิทธิภาพของโปรแกรมนั้นขึ้นอยู่กับการสุ่ม ซึ่งถ้าทำการสุ่มตัวอย่างที่ไม่ดีออกมา ก็จะทำให้ประสิทธิภาพของขั้นตอนวิธีดังกล่าวไม่ดีไปด้วย นอกจากนี้ยังต้องทำการคำนวณข้อจำกัด อีกด้วย เมื่อเปรียบเทียบเวลาในการทดสอบโปรแกรมนั้นจะขึ้นกับความซับซ้อนของพวก Predicate ซึ่งถ้ามีจำนวน Predicate มาก โปรแกรมก็จะใช้เวลาในการสร้างข้อมูลการทดสอบนานไปด้วย และจากวิธีการใช้เซตแผ่ทั่ว (Using Spanning Sets Coverage Testing) [1] นั้น ผลลัพธ์ที่ได้อาจจะไม่ครอบคลุมทุกกรณี นอกจากนี้ อีกวิธีคือการใช้ขั้นตอนวิธีทางพันธุกรรม และการวิเคราะห์เชิงความคิดเพื่อสร้างข้อมูลการทดสอบให้ครอบคลุมการทดสอบแบบกิ่ง (Using a Genetic Algorithm and Formal Concept Analysis to Generate Branch coverage Test Data Automatically) [3] นั้น

ยังมีข้อเสียอยู่ตรงที่ต้องมีการคำนวณเป็นจำนวนมาก และยังได้ผลไม่ค่อยดีเมื่อหาผลลัพธ์ที่เป็น ประโยคเงื่อนไขซ้อนๆ กัน (Nested Predicate) ส่วนในขั้นตอนวิธีนี้สามารถรับประกันได้ว่าครอบคลุมทุกเส้นทางที่เกิดขึ้นอย่างแน่นอน เพราะว่าขั้นตอนวิธีนี้ได้ทำการค้นหาเส้นทางที่เป็นไปได้ทั้งหมด และนำมาทำการหาชุดของเส้นทางที่น้อยที่สุดที่เป็นไปได้อีกขั้นตอนหนึ่ง ดังนั้นจึงสามารถรับประกันได้ว่าขั้นตอนวิธีที่พัฒนาขึ้นมานั้นครอบคลุมการทดสอบแบบกิ่ง 100% และการหาเส้นทางของขั้นตอนวิธีนี้ยังไม่ขึ้นกับประโยคเงื่อนไข เพราะว่าขั้นตอนวิธีนี้ได้ทำการค้นหาทุกเส้นทางที่เป็นไปได้ทั้งหมดมาก่อนแล้ว อีกทั้งขั้นตอนวิธีนี้ยังไม่ต้องมีการคำนวณข้อจำกัดเหมือนวิธีอื่นๆดังกล่าวไว้ข้างต้น

5.3 ข้อเสนอแนะ

ขั้นตอนวิธีนี้ ยังสามารถพัฒนาให้มีประสิทธิภาพเพิ่มขึ้น ได้โดย

1) สามารถลดเนื้อที่ในการจัดเก็บเส้นทาง โดยเส้นทางที่มีจุดเริ่มต้นเดียวกัน ก็จะเก็บในกิ่งเดียวกันได้

2) สามารถลดการทำงานของโปรแกรมได้โดยจำกัดจำนวนเส้นทางที่นำมาทดสอบหรือเลือกเส้นทางที่ยาวที่สุดก่อนเพื่อลดเวลาในการค้นหาชุดของเส้นทางที่ครอบคลุมกราฟ

3) สามารถขยายขอบเขตของโปรแกรมโดยเพิ่มส่วนของการแปลงจากซอร์สโค้ดเป็นแผนภาพควบคุมการไหล และทำการเก็บลงในไฟล์รูปแบบ และในส่วนของการเลือกชุดเส้นทางที่เหมาะสมที่สุดออกมา พร้อมกับสร้างข้อมูลทดสอบเพื่อให้การทดสอบซอฟต์แวร์มีความสะดวกรวดเร็วมากยิ่งขึ้น

รายการอ้างอิง

[1] Martina Marré and Antonia Bertolino, *"Using Spanning Sets Coverage Testing"*, IEEE Transactions on software engineering,Vol. 29, pp. 974-984, 2003.

[2] Neelam Gupta, Aditya P.Mathur and Mary Lou Soffa, *"Generating Test Data For Branch Coverage"*, Automate Software Engineering, pp. 219-227, 2000.

[3] Susan Khor and Peter Grogono, *"Using a Genetic Algorithm and Formal Concept Analysis to Generate Branch Colverage Test Data Automatically"*,Proceeding of the International Conference on Automate Software Engineering(ASE'04), 2004.

[4] Jorgensen. Paul, *"Software Testing"*, CRC Press., 2002.

ภาคผนวก ก.

รายละเอียดตัวอย่างโปรแกรมที่ใช้ในปัญหาพิเศษ

ก.1 โปรแกรม Download

โปรแกรม Download เป็นโปรแกรมสำหรับเก็บข้อมูลเอเอสพีลงบนคอมพิวเตอร์โดยจะเก็บข้อมูลในรูปแบบของเท็กซ์ไฟล์ จากนั้นจึงทำการอ่านไฟล์ขึ้นมาเป็นตัวอักษร โดยรายละเอียดของโปรแกรมเป็นดังนี้

```
import java.io.*;
import java.net.*;

public class Download{
    public static void main(String[]args)throws Exception{
        URL url = new URL("http://www.iscanime.net/iscfs/rss/index.asp");
        InputStream is = url.openStream();
        int n = 0;
        byte b[] = new byte[32];
        //write all data form asp to txt file in our computer
        try{
            File file = new File("newItem.txt");
            FileWriter fw = new FileWriter(file);
            BufferedWriter bw =new BufferedWriter(fw);
            while((n=is.read(b)) != -1)
                bw.write(new String(b,0,n));
            bw.close();
        }

        catch (IOException ioe) {}

        //read File which is String type
        int counter = 0;
        String str;
      String[]temp = new String[30];
        try {
          BufferedReader in = new BufferedReader(new FileReader("newItem.txt"));
```

```
            while ((str = in.readLine()) != null) {
                temp[counter]=str;
                counter++;
            }
            in.close();
        } catch (IOException e) {}

        String rawText = temp[11];
        String line[] = new String[4];
        int k = 0,j=0;
        for(int i=0;i<rawText.length();i++){
            j=i;
            if(rawText.substring(j,j+1).equals("[")){
                while(!(rawText.substring(j,j+1).equals(")")))
                    j++;
                line[k]= rawText.substring(i,j);
                k++;
                i=j;
            }
            }//end for
            for(int i=0;i<line.length;i++)
                sendMessage(channel,line[i]);
    }
}
```

ก. 2 โปรแกรม FormatFile

โปรแกรม FormatFile เป็นโปรแกรมสำหรับการสร้างไฟล์รูปแบบเพื่อเป็นข้อมูลเข้าของโปรแกรมหาเส้นทางสำหรับการทดสอบแบบกิ่ง รายละเอียดของโปรแกรมเป็นดังนี้

```
import java.io.*;
import java.util.Stack;
import java.util.StringTokenizer;
class FormatFile
{       public static void main(String[]args)throws IOException
        {       BufferedReader get = new BufferedReader(new InputStreamReader(System.in));
                System.out.println("Input file name");
                PrintWriter in = new PrintWriter(new FileOutputStream(get.readLine()+".txt"));
                System.out.println("Input amount of nodes");
                int nodes = Integer.parseInt(get.readLine());
                for(int i=0;i<nodes;i++)
                {       System.out.println("Input nodes that connect to node "+i);
                        String s = get.readLine();
                        Stack sk = new Stack();
                        StringTokenizer st = new StringTokenizer(s);
                        while(st.hasMoreTokens())
                        {       sk.push((String)st.nextToken());
                        }
                        s = "";
                        int k;
                        while(!sk.empty())
                        {       k = Integer.parseInt(sk.pop().toString());
                                while(s.length()<(k*2))
                                {       s = s+"0"+" ";
                                }
                                s = s+"1"+" ";
                        }
                        while(s.length()<(nodes*2))
```

```
                {       s = s+"0"+" ";
                }

                in.println(s);
            }
            in.close();

        }
}
```

ก. 3 โปรแกรม BinarySearchTree

โปรแกรม BinarySearchTree เป็นโปรแกรมสำหรับการจัดเรียงข้อมูลโดยการสร้าง tree แล้วทำการ ค้นหา เพิ่ม ลบ โนด หรือแสดงข้อมูลโดยเรียงลำดับใน tree นั้น โดยโปรแกรมนี้แบ่งเป็น method ตามหน้าที่ รายละเอียดของโปรแกรมเป็นดังนี้

```
import java.io.*;

public class BinarySearchTree{
    public static void main(String[]args)throws IOException{
        BufferedReader get = new BufferedReader(new InputStreamReader(System.in));
        BinaryNode root = null;
        System.out.println("Binary tree generating system");
        while(true){
        System.out.println("[c]create node,[p]print node,[d]delete node,"
                                    +"[f]find,[s]size,[h]height,[e]exit");

        //********** string.charAt(0)
        switch((get.readLine()).charAt(0)){
            case 'c':
                                    root = insert(root);
                                    break;
            case 'p' :
                                    postorderNonRecursivePrint(root);
                                    break;
            case 'd' :
                                    root = delete(root,Integer.parseInt(get.readLine()));
                                    break;
            case 'f' :
                                    search(root,Integer.parseInt(get.readLine()));
                                    break;
            case 'h' :System.out.println(height(root));
                                    break;
```

```
                case 's' :          System.out.println(size(root));
                                            break;
                case 'e' :
                                            System.exit(0);
                                            break;

                default  :
                                            System.out.println("input error");
                                            break;
                }//end switch
        }//end forever
}//end main

static BinaryNode insert(BinaryNode root)throws IOException{
        BufferedReader get = new BufferedReader(new InputStreamReader(System.in));
        BinaryNode temp = null,current=null,previous=null;
        int k = Integer.parseInt(get.readLine());
        if(root==null){
                root = new BinaryNode(k);
        }
        else {
                temp = new BinaryNode(k);
                current = root;

                //find pos
                while(current!=null){
                        if(temp.val<=current.val){
                                previous = current;
                                current = current.left;
                        }
                        else{
                                previous = current;
```

```
                    current = current.right;
                }
            }//end while

            //insert
            if(temp.val<=previous.val)
                previous.left = temp;
            else
                previous.right = temp;

        }
        return root;
}

static void inorderPrint(BinaryNode root){
        if(root!=null){
        inorderPrint(root.left);
        System.out.println(root.val);
        inorderPrint(root.right);
        }
}

static void search(BinaryNode root,int find){
        //**********we don't use order to search
        boolean found = false;
        BinaryNode stemp = root;
        while(!found&&stemp!=null){
            if(find<stemp.val)
                stemp = stemp.left;
            else if(find>stemp.val)
                stemp = stemp.right;
            else
```

```
                found = true;
            }

        if(found)
        System.out.println(find + " is found!!");
        else
        System.out.println("not found");
}

//**************if u use ( temp = new temp ) or something similiar
//remember that object point to same memory position and it won't change after
//that method end . Besure to point object back or use another name to
//avoid name conflict.

static int size(BinaryNode root){
        if(root == null)
            return 0;
        else
            return size(root.left)+size(root.right)+1;
}
static int height(BinaryNode root){

//************* always do this "if"
        if(root == null)
            return -1;

//*************we use -1 because root start with 0
        else
            return Math.max(height(root.left),height(root.right))+1;
}

static BinaryNode delete(BinaryNode root,int dVal){
```

```
        if(root == null){
                System.out.println("BST is empty");
                return null;
        }
        else if(dVal<root.val)
                root.left = delete(root.left,dVal);
        else if(dVal>root.val)
                root.right = delete(root.right,dVal);
        else if(root.left!=null&&root.right!=null){
                root.val = findMin(root.right);
                //********root.right not root
                root.right = deleteMin(root.right);
        }
        else{
                if(root.left==null&&root.right==null)
                        return null;
                else if(root.left==null)
                        root = root.right;
                else if(root.right==null)
                        root = root.left;
                }

        return root;
}
        static  int findMin(BinaryNode root){
                while(root.left!=null)
                        root = root.left;
                return root.val;
        }
        static BinaryNode deleteMin(BinaryNode root){
                if(root.left !=null)
                        root.left = deleteMin(root.left);
```

```
                return root.right;
        }

static void inorderNonRecursivePrint(BinaryNode root){
        BinaryNode pos =root;
        BinaryNode[]stack = new BinaryNode[size(root)];
        int k = 0;
        while(pos!=null||k>0){
                if(pos!=null){
                        stack[k]=pos;
                        pos = pos.left;
                        k++;
                }
                else{
                        k--;
                        pos = stack[k];
                        System.out.println(pos.val);
                        pos = pos.right;
                }
        }//end while
}//end non recursive inorder print

static void preorderNonRecursivePrint(BinaryNode root){
        BinaryNode pos = root;
        BinaryNode[]stack = new BinaryNode[size(root)];
        int k = 0;

        while(pos!=null||k>0){
                if(pos!=null){
                        System.out.println(pos.val);
                        stack[k]=pos;
```

```
                pos = pos.left;
                k++;
                }
            else{
                k--;
                pos =stack[k];
                pos = pos.right;
                }
            }
        }//end preorder non recursive print

static void postorderNonRecursivePrint(BinaryNode root){
        BinaryNode pos = root;
        BinaryNode[]stack = new BinaryNode[size(root)];
        int[]stackI = new int[size(root)];
        int k = 0,direction=0;
        boolean start = true;

        while(start||k>0){
            start =false;
            if(pos!=null&&direction==0){
                stack[k] = pos;
                stackI[k] = 1;
                k++;
                pos = pos.left;
                }
            else{
                k--;
                pos = stack[k];
                direction = stackI[k];
                if(direction ==1){
                        stack[k] = pos;
```

```
                        stackI[k] = 2;
                        k++;
                        pos = pos.right;
                        direction = 0;
                    }
                    else{
                        System.out.println(pos.val);
                    }
                }
            }//end while
        }//end postorder non recursive print
}
```

ภาคผนวก ข.

การติดตั้งและใช้โปรแกรมหาชุดของเส้นทางที่เหมาะสม

โปรแกรมหาชุดของเส้นทางที่เหมาะสมนั้น เขียนโดยใช้ภาษาจาวา เวอร์ชัน 1.5.0_02 (jdk-1_5_0_02-windows-i586-p.exe) เพราะฉะนั้นการใช้โปรแกรมหาชุดของเส้นทางที่เหมาะสมจึงจำเป็นต้องติดตั้งซอฟต์แวร์ของภาษาจาวาเสียก่อนโดยขั้นตอนการติดตั้งเป็นดังนี้

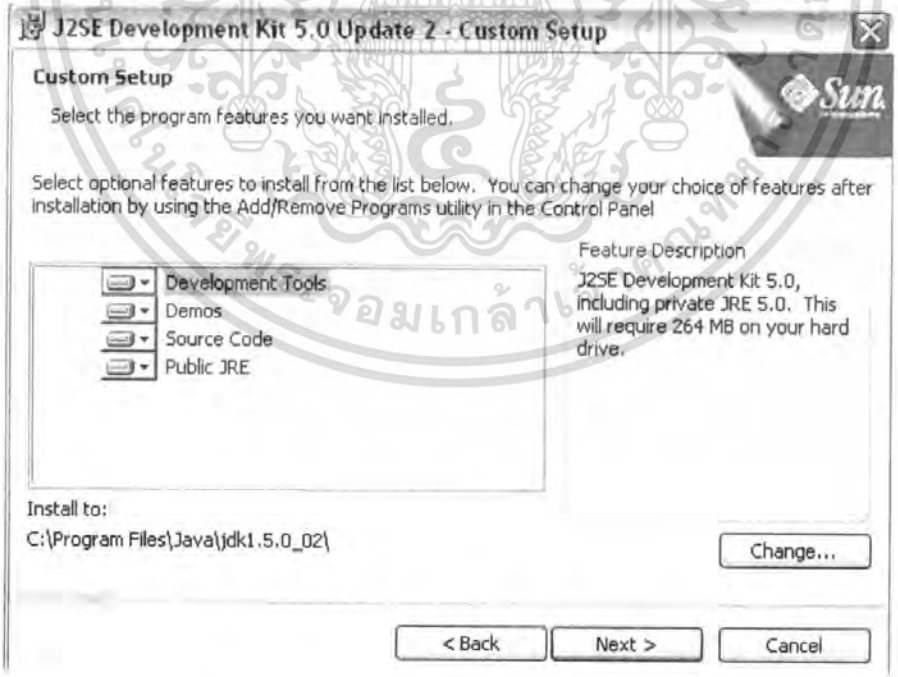
ดับเบิลคลิกที่ไฟล์ jdk-1_5_0_02-windows-i586-p.exe เพื่อให้แกรมทำงานจากนั้นคลิก I accept the terms in the license agreement แล้วคลิก Next ดังรูป

ภาพที่ ข.1 เงื่อนไขการติดตั้งโปรแกรม J2SE Development Kit 5.0

สามารถเปลี่ยน Path ที่จะติดตั้งโปรแกรมได้โดยคลิกที่ Change และเลือก Path ที่ต้องการติดตั้งดังรูป

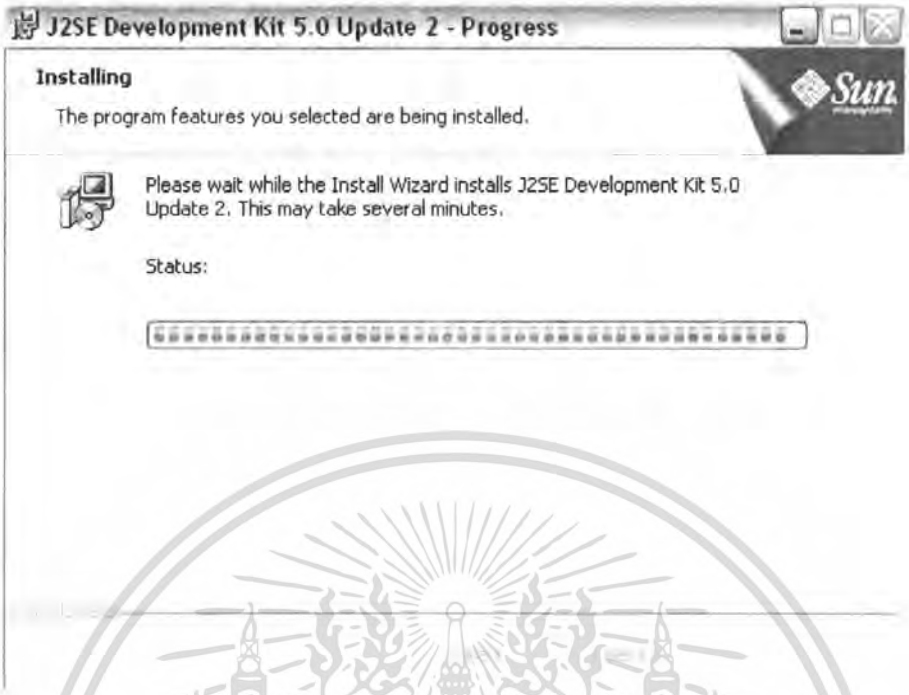
ภาพที่ ข.2 ระบุตำแหน่งที่จะติดตั้งโปรแกรม

หลังจากเลือก Path ได้แล้ว คลิก Next



ภาพที่ ข.3 เลือกคุณสมบัติของโปรแกรมที่คุณต้องการจะติดตั้ง

โปรแกรมจะทำการติดตั้งตัวเองโดยอัตโนมัติ จากนั้นคลิก Finish



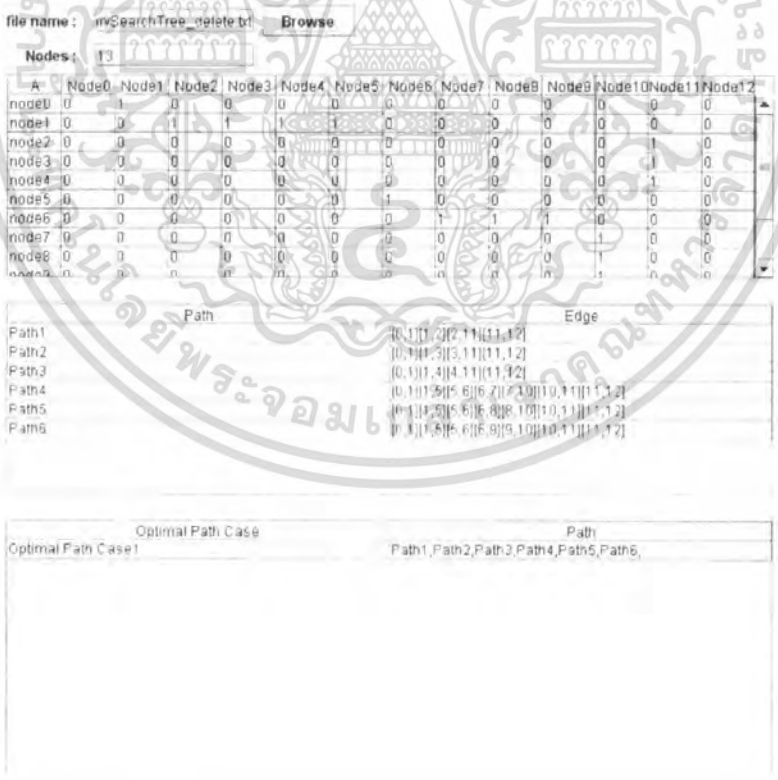
ภาพที่ ข.4 ดำเนินการติดตั้งโปรแกรม

หลังจากนั้นนำไฟล์ proj2.jar มาวางไว้บนเครื่อง แล้วดับเบิลคลิกเพื่อเรียกใช้งาน โดยการใช้งานโปรแกรมมีขั้นตอนดังนี้

เมื่อเปิดโปรแกรมแล้วให้คลิกที่ Browse เพื่อเลือกไฟล์ที่ต้องการทดสอบ โดยไฟล์ที่ใช้ในการทดสอบนี้จะอยู่ในรูปแบบของเท็กซ์ไฟล์ดังรูป

ภาพที่ ข.5 เลือกไฟล์ที่จะนำมาทดสอบ

จากนั้นคลิก open จะได้ผลลัพธ์ของโปรแกรมดังรูป



ภาพที่ ข.6 ผลที่ได้จากการรันโปรแกรม