การออกแบบและพัฒนาระบบแสดงภาพปริภูมิสถานะ

นายกิตติชัย เกื้อมา

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2552

DESIGN AND DEVELOPMENT OF STATE SPACE VISUALIZATION SYSTEM

Mr. Kittichai Kuema

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Science Program in Computer Science

Department of Computer Engineering

Faculty of Engineering

Chulalongkorn University

Academic Year 2009

หัวข้อวิทยานิพนธ์	การออกแบบและพัฒนาระบบแสดงภาพปริภูมิสถานะ
โดย	นายกิตติชัย เกื้อมา
สาขาวิชา	วิทยาศาสตร์คอมพิวเตอร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	รองศาสตราจารย์ ดร. สมชาย ประสิทธิ์จูตระกูล

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

..... คณบดีคณะวิศวกรรมศาสตร์
(รองศาสตราจารย์ ดร.บุญสม เลิศหิรัญวงศ์)

คณะกรรมการสอบวิทยานิพนธ์

..... ประธานกรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.วีระ เหมืองสิน)

..... อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก
(รองศาสตราจารย์ ดร.สมชาย ประสิทธิ์จูตระกูล)

..... กรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.สุกรี สินธุภิญโญ)

..... กรรมการภายนอกมหาวิทยาลัย
(ผู้ช่วยศาสตราจารย์ ดร.วรเศรษฐ สุวรรณิก)

กิตติชัย เกื้อมา : การออกแบบและพัฒนาระบบแสดงภาพปริภูมิสถานะ. (DESIGN AND DEVELOPMENT OF STATE SPACE VISUALIZATION SYSTEM) อ.ที่ปรึกษาวิทยานิพนธ์หลัก: รศ.ดร. สมชาย ประสิทธิ์จูตระกูล, 111 หน้า.

วิทยานิพนธ์ฉบับนี้นำเสนอการออกแบบพัฒนาระบบแสดงภาพปริภูมิสถานะ ซึ่งนำเสนอระบบแสดงภาพต้นไม้ปริภูมิสถานะของโปรแกรมที่แก้ปัญหาด้วยกลวิธีการค้นคำตอบในปริภูมิสถานะ ตัวระบบรองรับโปรแกรมการค้นคำตอบที่เขียนแบบเรียกซ้ำ และแบบที่ตัวสถานะเป็นอ็อบเจกต์ที่มีการสร้างและเก็บระหว่างการค้น การแสดงภาพนี้กระทำได้ด้วยการเพิ่มรหัสคำสั่งกำกับในโปรแกรมค้นคำตอบโดยสามารถแสดงข้อความหรือภาพกำกับปมของต้นไม้มีชื่อเรียกว่า "JSTATE101" ซึ่งใช้กลไกมาตรฐานของระบบจาวาในการติดตามการเปลี่ยนแปลงสถานะ การทดลองใช้งานพบว่า ระบบรองรับการค้นปริภูมิสถานะระดับแสนปมได้ ซึ่งเพียงพอกับการศึกษาอัลกอริทึมการค้นพื้นฐานทั่วไป

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต .....

สาขาวิชา วิทยาศาสตร์คอมพิวเตอร์ ลายมือชื่ออ.ที่ปรึกษาวิทยานิพนธ์หลัก .....

ปีการศึกษา 2552

5071403321 : MAJOR COMPUTER SCIENCE

KEYWORDS : STATE SPACE SEARCH / STATE SPACE TREE

KITTICHAI KUEAMA: DESIGN AND DEVELOPMENT OF STATE SPACE VISUALIZATION SYSTEM. THESIS ADVISOR: ASSOC. PROF. SOMCHAI PRASITJUTRAKUL, Ph.D., 111 pp.

This research presents JSTATE101, a state space tree visualization system for programs that use state space search techniques. The system supports programs which search recursively and programs which explicitly create and store state objects during the search. By adding a few lines of additional annotations and codes, the search program is ready to be visualized. Tree nodes can be shown with either text or image. JSTATE101 utilizes standard mechanism in Java platform to track state changes. Experiments showed that the system can supports the space with a hundred thousand nodes which is sufficient in studying general basic search algorithms.

Department : Computer Engineering Student's Signature .....

Field of Study : Computer Science Advisor's Signature .....

Academic Year : 2009

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้ได้สำเร็จลุล่วงไปด้วยความช่วยเหลืออย่างดียิ่งของอาจารย์ที่ปรึกษา คือ รศ.ดร.สมชาย ประสิทธิ์จูตระกูล ที่ให้คำปรึกษา ความรู้ ดูแล ให้ข้อเสนอแนะเกี่ยวกับงานวิจัยด้วยดีมาตลอด รวมทั้งช่วยเหลือ ให้ความรู้ที่เป็นประโยชน์ และสละเวลา เพื่อให้งานนี้สำเร็จลุล่วงได้ทันเวลา

ท้ายนี้ ผู้วิจัยขอกราบขอบพระคุณ บิดา-มารดา ผู้ให้กำเนิด กับความรักความอบอุ่นและกำลังใจในการดำเนินชีวิตต่อผู้วิจัยตลอดมา รวมทั้งสนับสนุนทั้งด้านทุนทรัพย์ในการศึกษาระดับปริญญาโทจนสำเร็จการศึกษา

สารบัญ

หน้า

สารบัญตาราง

หน้า

สารบัญภาพ

หน้า

สารบัญรหัส

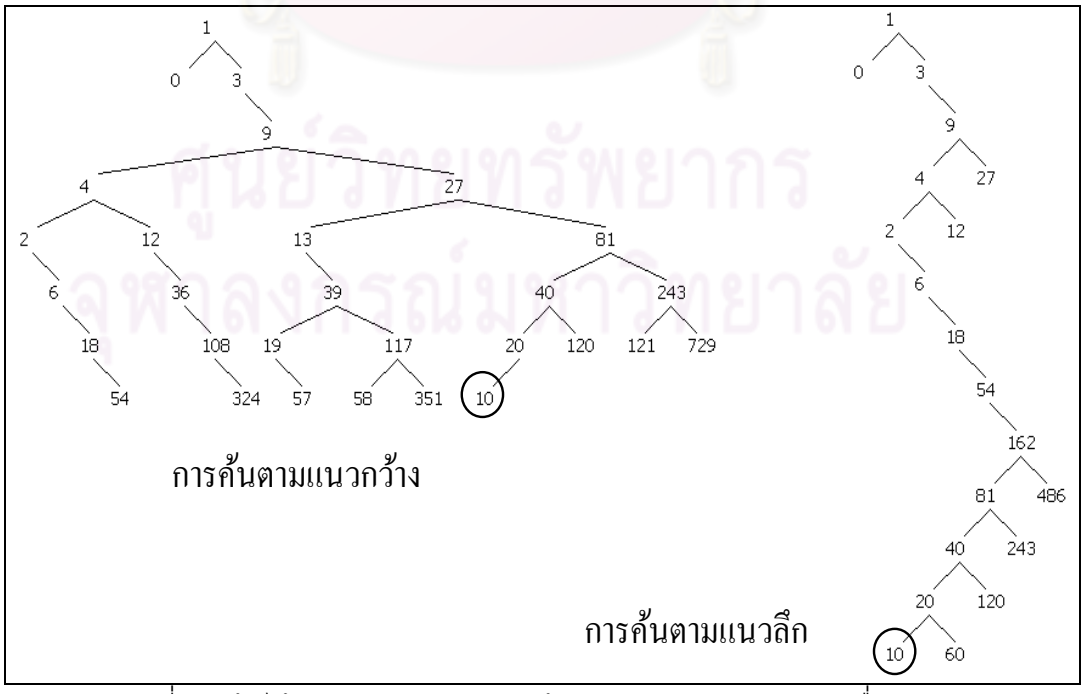
หน้า

บทที่ 1

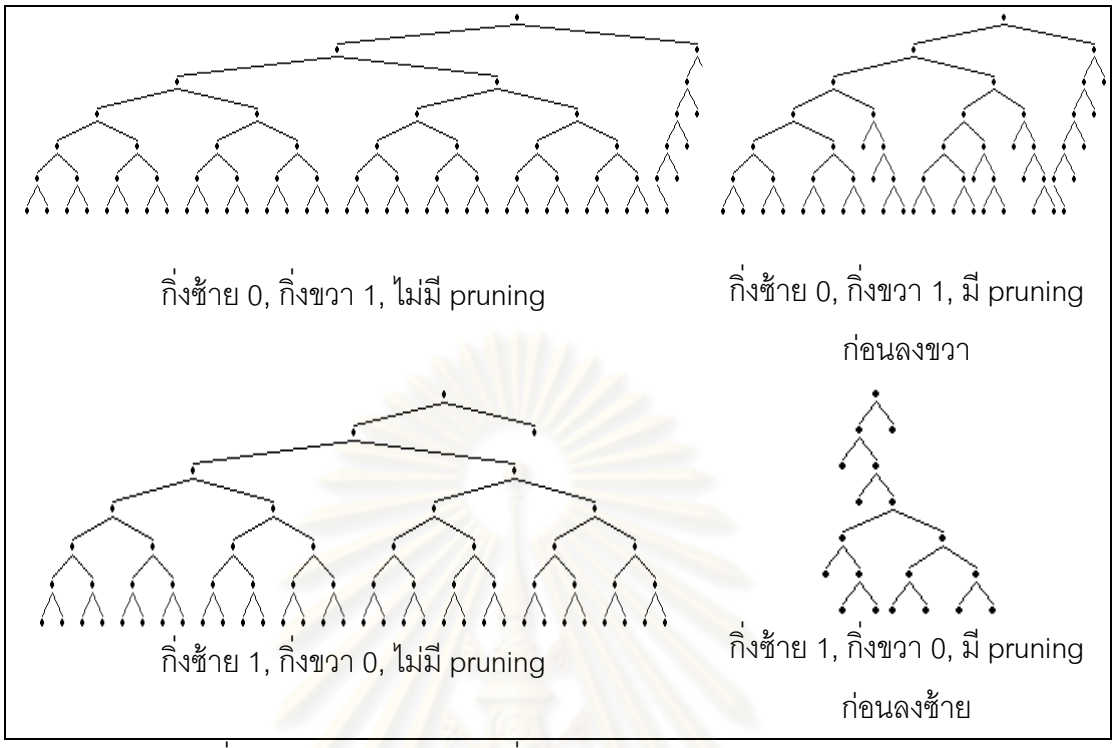
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

การค้นในปริภูมิสถานะเป็นกระบวนการหนึ่งในการแก้ไขปัญหาด้วยโปรแกรมคอมพิวเตอร์ ซึ่งอาศัยการแจกแจงสถานะของผลเฉลยต่าง ๆ จนกว่าจะพบคำตอบ การแจกแจงสถานะมีรายละเอียดมากมายขึ้นกับลักษณะของปัญหาและกลวิธีที่ใช้ เพื่อหวังให้พบคำตอบได้รวดเร็ว การทำความเข้าใจพฤติกรรมการทำงานของอัลกอริทึมที่ใช้วิธีนี้มักต้องจินตนาการถึงลำดับของสถานะและการเปลี่ยนแปลงสถานะต่าง ๆ ที่เกิดขึ้น หากสามารถแสดงต้นไม้ปริภูมิสถานะ ซึ่งแทนความสัมพันธ์ของสถานะต่าง ๆ ที่ได้รับการผลิตและพิจารณาระหว่างการทำงานของโปรแกรม ย่อมเสริมความเข้าใจในพฤติกรรมของตัวอัลกอริทึม อันอาจนำไปสู่การปรับปรุงกลวิธีการแจกแจงสถานะระหว่างการค้นให้มีประสิทธิภาพที่ดีขึ้น รูปที่ 1.1 แสดงต้นไม้ปริภูมิสถานะเมื่อใช้การค้นตามแนวกว้างและตามแนวลึก ในการหาว่าค่า 10 นั้นสร้างได้อย่างไรด้วยการเริ่มที่ค่า 1 แล้วคูณสามกับหารสอง (การค้นตามแนวกว้างในรูปได้คำตอบคือ 10 = 1x3x3x3x3x2x2x2 ส่วนการค้นตามแนวลึกได้คำตอบคือ 10 = 1x3x3x2x2x3x3x3x3x2x2x2x2) ส่วนรูปที่ 1.2 **ต้นไม้ปริภูมิสถานะที่ได้จากการค้นตามแนวลึกหลายแบบของการหาเซตย่อยของ** {2, 8, 10, 6, 4, 7} **ที่มีผลรวมเป็น** 9 ของปัญหา sum of subset [1] โดยมีการแจกแจงด้วยกลวิธีแตกต่างกัน ซึ่งสะท้อนถึงเวลาในการค้นคำตอบ



รูปที่ 1.1 ต้นไม้ปริภูมิสถานะจากการค้นปัญหาคูณสามหารสองเพื่อหา 10



รูปที่ 1.2 ต้นไม้ปริภูมิสถานะที่ได้จากการค้นตามแนวลึกหลายแบบ

งานวิจัยนี้นำเสนอการออกแบบและพัฒนาระบบบันทึกการเปลี่ยนแปลงของสถานะระหว่างการค้นคำตอบ ตัวระบบนำเสนอการเปลี่ยนแปลงสถานะในรูปแบบของต้นไม้ปริภูมิสถานะ และในกรณีที่การค้นคำตอบเสร็จสิ้นแล้ว ผู้ใช้สามารถควบคุมการแสดงภาพการเปลี่ยนแปลงทั้งย้อนหลังหรือเดินหน้าในลักษณะคล้ายกับเครื่องเล่นวิดีทัศน์ เพื่อเสริมความเข้าใจการทำงานของโปรแกรมค้นคำตอบ

1.2 วัตถุประสงค์ของการวิจัย

ออกแบบและพัฒนาระบบบันทึกการเปลี่ยนแปลงของสถานะและแสดงต้นไม้ปริภูมิสถานะระหว่างการค้นคำตอบ ระบบอนุญาตให้ผู้ใช้ชมภาพการเปลี่ยนแปลงทั้งย้อนหลังหรือเดินหน้าในลักษณะคล้ายกับเครื่องเล่นวิดีทัศน์ เพื่อเสริมความเข้าใจการทำงานของโปรแกรมค้นคำตอบ

1.3 ขอบเขตของการวิจัย

1 ใช้กับโปรแกรมที่พัฒนาด้วยภาษาจาวา

2 แสดงปริภูมิสถานะในรูปของต้นไม้ปริภูมิสถานะ

3 รองรับสถานะในปริภูมิที่เป็นอ็อบเจกต์ของคลาสที่ implements อินเทอร์เฟซตามที่ระบบกำหนด

4 รองรับสถานะในปริภูมิที่เก็บใน stack frame ระหว่างการค้นคำตอบที่เขียนแบบเรียกซ้ำ

5 สนับสนุนการแสดงภาพระหว่างการค้นคำตอบของหลายโปรแกรมพร้อมกันได้

6 ผู้ใช้สามารถแสดงสถานะในต้นไม้เป็นข้อความหรือรูปภาพได้

7 มีแผงควบคุมการแสดงภาพการเปลี่ยนแปลงของต้นไม้ปริภูมิสถานะ การเดินหน้า ถอยหลัง ซูมเข้า ซูมออกเพื่อปรับขนาดภาพ

1.4 ขั้นตอนและวิธีดำเนินงานวิจัย

1 ศึกษาและค้นคว้าการแก้ไขปัญหาด้วยกลวิธีการค้นในปริภูมิสถานะ

2 ศึกษาการใช้งานคลาสจาวาที่เกี่ยวข้องกับการออกแบบและพัฒนาระบบ

3 ออกแบบส่วนประกอบย่อยต่าง ๆ ของระบบ

4 พัฒนาส่วนประกอบย่อยต่าง ๆ ของระบบที่ได้ออกแบบไว้

5 สร้างกรณีศึกษาการค้นในปริภูมิสถานะกับปัญหาต่าง ๆ เพื่อแสดงการใช้งานระบบ และทดสอบระบบด้วย

6 สรุปผลการวิจัยและจัดทำวิทยานิพนธ์เป็นรูปเล่ม

1.5 ประโยชน์ที่คาดว่าจะได้รับ

ได้ระบบแสดงต้นไม้ปริภูมิสถานะที่สามารถใช้เสริมการทำความเข้าใจพฤติกรรมการทำงานของโปรแกรมค้นในปริภูมิสถานะที่สนใจ สามารถใช้ประกอบการเรียนการสอนวิชาต่าง ๆ ที่เกี่ยวกับอัลกอริทึมและปัญญาประดิษฐ์ได้

1.6 เนื้อหาวิทยานิพนธ์

วิทยานิพนธ์ฉบับนี้แบ่งออกเป็น 7 บท ซึ่งในบทนำจะแสดงให้เห็นที่มาและความสำคัญของปัญหา ส่วนในบทที่ 2 กล่าวถึงงานวิจัยทางด้านปริภูมิสถานะที่มีอยู่ บทที่ 3 กล่าวถึงแนวคิดและทฤษฎี บทที่ 4 กล่าวถึงการใช้งานแสดงภาพปริภูมิสถานะ บทที่ 5 นั้นกล่าวถึงการระบบแสดงภาพปริภูมิสถานะ บทที่ 6 เป็นตัวอย่างการค้นหาคำตอบสำหรับปัญหาทั่วไป ส่วนบทที่ 7 จะเป็นบทสรุปผลการวิจัยรวมทั้งข้อเสนอแนะสำหรับผู้ที่สนใจในงานวิจัยด้านนี้

บทที่ 2

เอกสารและงานวิจัยที่เกี่ยวข้อง

การแก้ปัญหาทางปัญญาประดิษฐ์ได้ถูกนำเสนอในกรอบการทำงานของ “State Space Search” [2] โดยมีนักวิจัยที่เสนอแนวทางนี้คือ Newell and Simon ปี 1969 [3] ได้อธิบายการค้นคำตอบในปริภูมิสถานะด้วยการกำหนดสถานะเริ่มต้นที่ใช้ในการค้นคำตอบ จากนั้นใช้ตัวกระทำการที่ใช้กับสถานะเหล่านั้นจนกว่าจะไปถึงสถานะเป้าหมายหรือ “goal state” ซึ่งหลักการนี้ได้ถูกนำเสนอในรูปแบบของกราฟโครงสร้างต้นไม้ด้วยไดอะแกรมแสดงภาพ อย่างเช่น การแก้ปัญหา 8 puzzle หรือ เกม Tic-Tac-Toe (Nilsson, 1971[4])

2.1 อุปสรรคการเรียนรู้วิธีการค้นคำตอบในปริภูมิสถานะ

การเรียนการสอนเพื่อให้นักเรียนเข้าใจถึงวิธีค้นคำตอบในปริภูมิสถานะ เพื่อใช้แก้ปัญหาโจทย์ต่างๆนั้น พบว่านักเรียนมักจะขาดทักษะการนำอัลกอริทึมค้นหามาใช้แก้ปัญหาโจทย์รวมทั้งการนำไปประยุกต์ใช้อย่างเหมาะสม ซึ่งปัญหาที่เกิดขึ้นกับนักเรียนมีสาเหตุหลักดังนี้

2.1.1 ปัญหาในการเรียนรู้ลักษณะการหาคำตอบด้วยอัลกอริทึมค้นหา

ปัญหาของนักเรียนอย่างหนึ่งที่พบ คือ การทำความเข้าใจลักษณะการค้นคำตอบด้วยอัลกอริทึมค้นหาในปริภูมิสถานะ อย่างเช่น ระหว่างการค้นคำตอบเพื่อแจงผลเฉลยนั้น จะอธิบายได้อย่างไรว่าสถานะถูกผลิตขึ้นนั้นมีทิศทางการขยายปมคำตอบในทิศทางใด หรือผลเฉลยที่เกิดขึ้นนั้นสัมพันธ์กับสถานะถัดไปอย่างไร เป็นต้น โดยทั่วไปนั้นนักเรียนจะทำความเข้าใจการแก้ปัญหาในปริภูมิสถานะ จากการเรียนรู้ด้วยตัวอย่างแผนภาพหรือจากการอธิบายด้วยรหัสคำสั่งทำให้มองเห็นภาพไม่ชัดเจนในการอธิบายการแก้โจทย์ปัญหาต่าง ๆ ด้วยอัลกอริทึมค้นหา

2.1.2 ปัญหาการเลือกใช้อัลกอริทึมค้นหาที่เหมาะสมเพื่อแก้ปัญหาโจทย์

สำหรับอีกปัญหาหนึ่ง คือ นักเรียนยังขาดทักษะในการเลือกวิธีที่เหมาะสม เพื่อช่วยหาผลเฉลยของคำตอบปัญหาในปริภูมิสถานะได้อย่างมีประสิทธิภาพ อย่างเช่น จะเลือกอัลกอริทึม A หรือ B จึงจะช่วยให้ได้คำตอบเร็วที่สุด หรือกลวิธีการค้นคำตอบใดดีกว่ากัน เป็นต้น ดังนั้นการเปรียบเทียบพฤติกรรมของอัลกอริทึมค้นหาในการแก้ปัญหาโจทย์ เพื่อทำความเข้าใจในพฤติกรรมของการค้นคำตอบในปริภูมิสถานะจึงเป็นสิ่งสำคัญอีกอย่างหนึ่งที่นักเรียนควรศึกษา

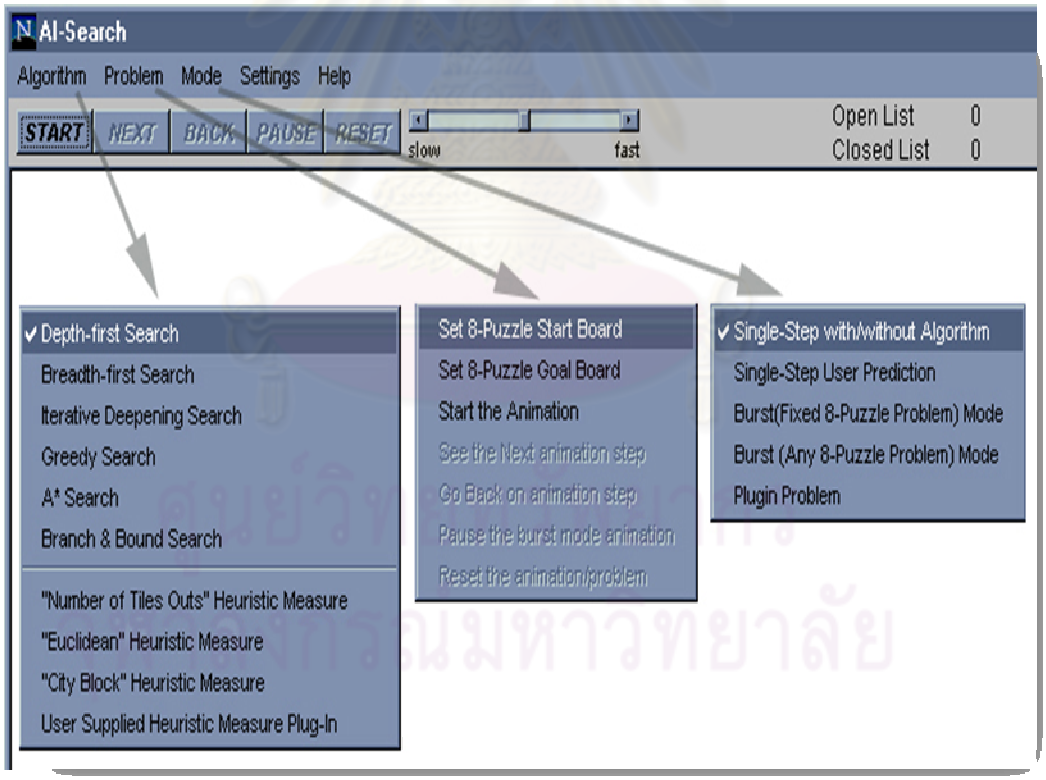
2.2 เครื่องมือแสดงภาพการค้นหาคำตอบในปริภูมิสถานะ

มีงานวิจัยที่นำเสนอเครื่องมือสำหรับอธิบายการค้นคำตอบในปริภูมิสถานะ ซึ่งนำเสนอภาพการแจงผลเฉลยการแก้ปัญหาในปริภูมิสถานะ เพื่อช่วยให้นักเรียนได้เห็นภาพถึงการทำงาน

ของอัลกอริทึมค้นหาในปริภูมิสถานะได้ชัดเจนขึ้น เครื่องมือเหล่านี้จะอธิบายภาพการค้นคำตอบของปัญหา ด้วยกลวิธีต่าง ๆ ทั้งในเชิงเปรียบเทียบพฤติกรรม และอธิบายลักษณะการค้นหาคำตอบในแต่ละขั้นตอน โดยมีงานวิจัยที่กล่าวถึงดังนี้

2.2.1 เครื่องมือแสดงการค้นปริภูมิทางปัญญาประดิษฐ์

เครื่องมือแสดงการค้นปริภูมิทางปัญญาประดิษฐ์ หรือ AI Search [5] ถูกนำเสนอโดย RMIT University ซึ่งเป็นเครื่องมือแสดงภาพเคลื่อนไหวลักษณะการทำงานของอัลกอริทึมค้นหาในปริภูมิสถานะ อย่างเช่น การค้นหาตามแนวลึก การค้นหาตามแนวกว้าง เป็นต้น สำหรับการสอนให้นักเรียนเห็นภาพการแจงผลเฉลยปัญหาในปริภูมิสถานะ โดยเครื่องมือนี้พัฒนาด้วยภาษา java applet ทำงานบนเว็บบราวเซอร์ ที่มีการรวบรวมอัลกอริทึมค้นหาบนปริภูมิสถานะในแบบต่าง ๆ รวมถึงการอธิบายการแจงคำตอบในโจทย์ปัญหา อย่างเช่น 8-Puzzle เป็นต้น โดยสามารถควบคุมการนำเสนอภาพการแจงผลเฉลยปัญหาในปริภูมิสถานะผ่านส่วนติดผู้ใช้ ได้ดังแสดงรูปที่ 2.1

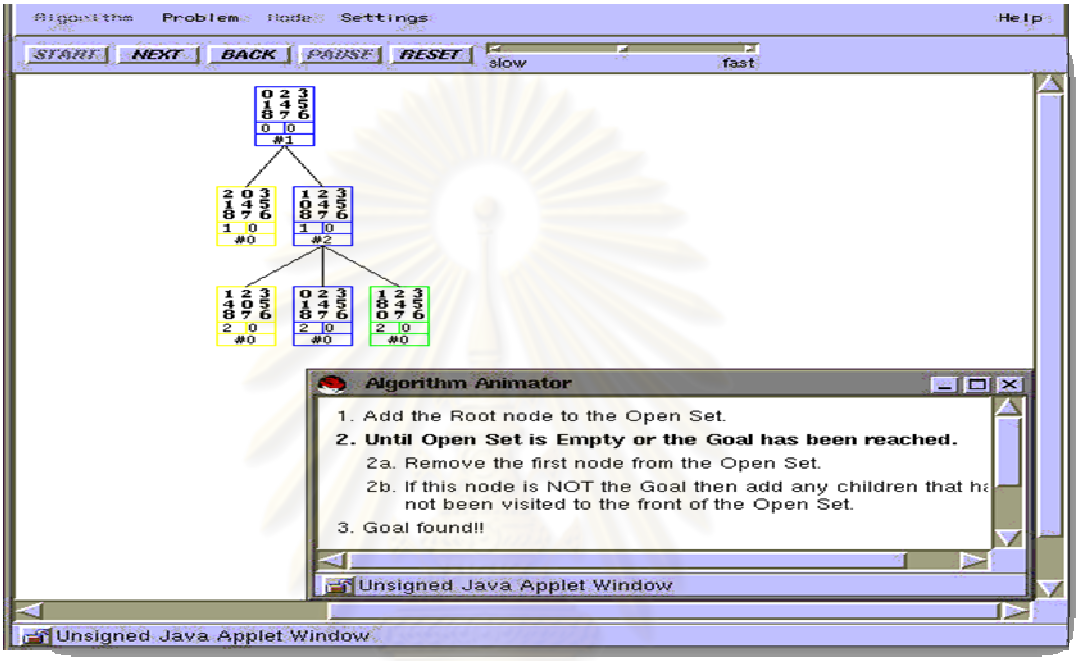


รูปที่ 2.1 ส่วนติดต่อผู้ใช้การทำงานระบบแสดงภาพเคลื่อนไหว AI Search

วิธีนำเสนอภาพการแจงผลเฉลยปัญหาในปริภูมิสถานะของเครื่องมือนี้ เพื่อช่วยให้นักเรียนเห็นถึงภาพการทำงานของอัลกอริทึมค้นหามีดังนี้

2.2.1.1 การแสดงภาพด้วยการทำงานแบบ Single Step Mode

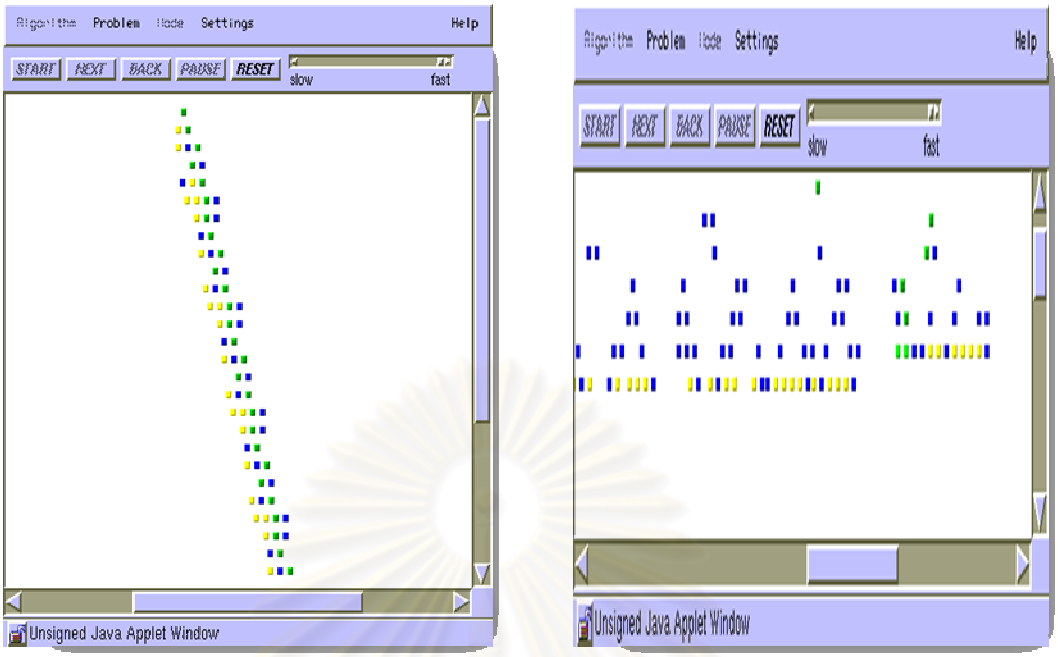
การแสดงภาพต้นไม้ปริภูมิสถานะด้วยการทำงานแบบ Single Step Mode จะช่วยให้นักเรียนเห็นภาพการแจงผลเฉลยของปัญหาระหว่างค้นคำตอบในปริภูมิสถานะ โดยการควบคุมการนำเสนอภาพปมสถานะที่เกิดใหม่ผ่านส่วนติดต่อผู้ใช้ อย่างเช่น การเดินหน้า-ถอยหลัง เป็นต้น ดังแสดงในรูปที่ 2.2



รูปที่ 2.2 การแจงการค้นคำตอบในปัญหา 8 Puzzle ด้วย AI Search

2.2.1.2 การแสดงภาพด้วยการทำงานแบบ Burst Mode

การแสดงภาพต้นไม้ปริภูมิสถานะด้วยการทำงานแบบ Burst Mode นี้จะช่วยให้นักเรียนเรียนเห็นภาพการนำอัลกอริทึมค้นหามาใช้ในการเปรียบเทียบการค้นคำตอบในปริภูมิสถานะ ทำให้นักเรียนเห็นถึงถาพความแตกต่างในการค้นคำตอบของแต่ละกลวิธี รวมทั้งเห็นถึงประสิทธิภาพการค้นคำตอบในแต่ละของวิธีบนโจทย์ปัญหาเดียวกัน ดังแสดงในรูปที่ 2.3



ภาพการค้นแบบแนวลึก ภาพการค้นแบบแนวกว้าง

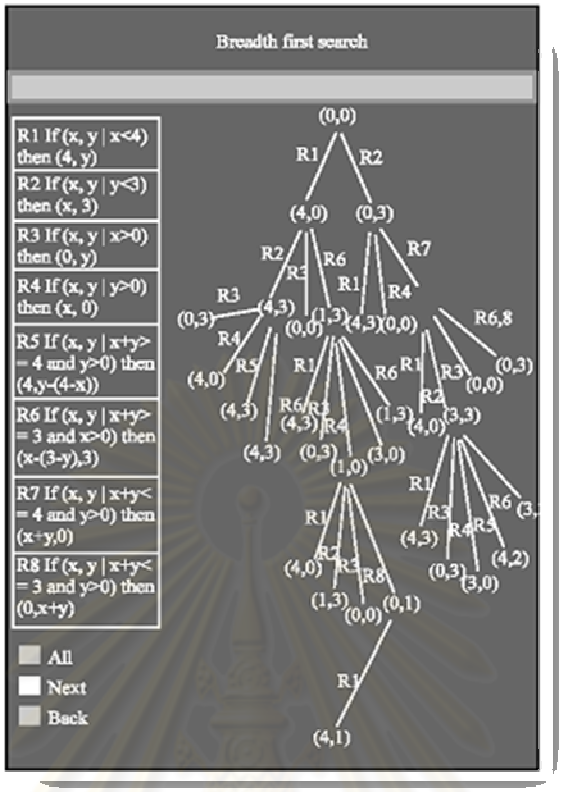
รูปที่ 2.3 เปรียบเทียบการค้นคำตอบในปริภูมิสถานะในปัญหา 8-Puzzle

2.2.2 การพัฒนาเครื่องมือแสดงภาพสำหรับสอนอัลกอริทึมค้นหาทางปัญญาประดิษฐ์

Al-Azhar University [6] นำเสนอเครื่องมือแสดงภาพต้นไม้ปริภูมิสถานะ เพื่อช่วยสอนนักเรียนให้เห็นถึงภาพการค้นคำตอบในปริภูมิสถานะด้วยอัลกอริทึมค้นหาจากรหัสคำสั่ง โดยวิธีนำเสนอภาพการค้นคำตอบในปริภูมิสถานะมีดังนี้

2.2.2.1 การนำเสนอภาพการแจงคำตอบในปริภูมิสถานะ

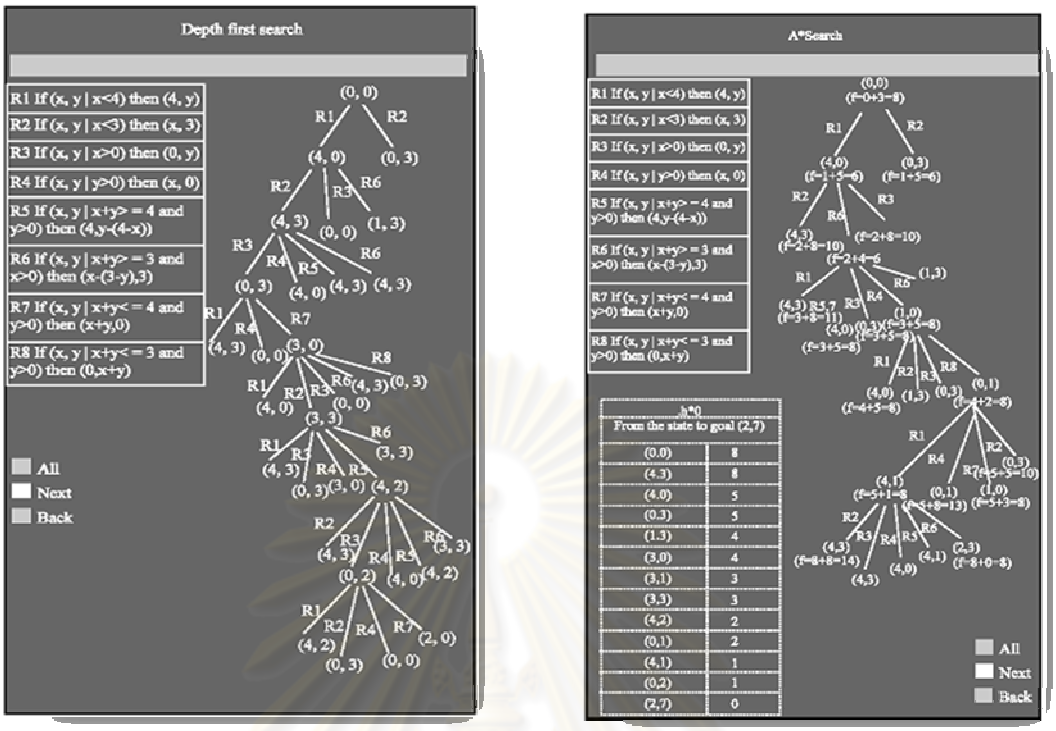
เครื่องมือในงานวิจัยนี้นำเสนอภาพการค้นคำตอบในปริภูมิสถานะ ที่ช่วยให้นักเรียนเห็นภาพการแจงผลเฉลยที่ได้จากการค้นคำตอบในปริภูมิสถานะ โดยนักเรียนสามารถดูการแจงผลเฉลยในแต่ละปมคำตอบผ่านส่วนติดต่อผู้ใช้ อย่างเช่น การทำ Single step เพื่อดูภาพการผลิตปมสถานะที่ละปม หรือ back step เพื่อย้อนกลับดูภาพปมสถานะก่อนหน้า เป็นต้น ทำให้นักเรียนสามารถเห็นภาพการค้นคำตอบของปัญหาได้ชัดเจนขึ้น ดังแสดงในรูปที่ 2.4 เป็นการแก้ปัญหา Water jug [7] ด้วยการค้นตามแนวกว้าง



รูปที่ 2.4 ปัญหา Water jug กับการค้นหาตามแนวกว้าง

2.2.2.2 การเปรียบเทียบภาพการค้นคำตอบในปริภูมิสถานะ

อีกวิธีหนึ่งงานวิจัยนี้ใช้สอนนักเรียนให้เห็นภาพการแจงผลเฉลยปัญหาด้วยอัลกอริทึมค้นหาในปริภูมิสถานะ คือ การนำเสนอวิธีการค้นตอบในปริภูมิสถานะด้วยอัลกอริทึมค้นหาต่าง ๆ มาแสดงให้เห็นถึงภาพที่ได้จากการเลือกใช้อัลกอริทึมค้นหา ซึ่งสะท้อนให้เห็นถึงความรวดเร็วในการค้นหาคำตอบที่แตกต่างกัน ดังรูปที่ 2.5 แสดงการเปรียบเทียบแจงกฎของปัญหา Water jug ใช้อัลกอริทึมค้นหาแนวลึกและ A* ซึ่งจะเห็นขนาดการเติบโตของค้นไม้ที่แตกต่างกันในโจทย์เดียวกัน



อัลกอริทึมค้นหาแบบแนวกว้าง อัลกอริทึมค้นหาแบบ A*

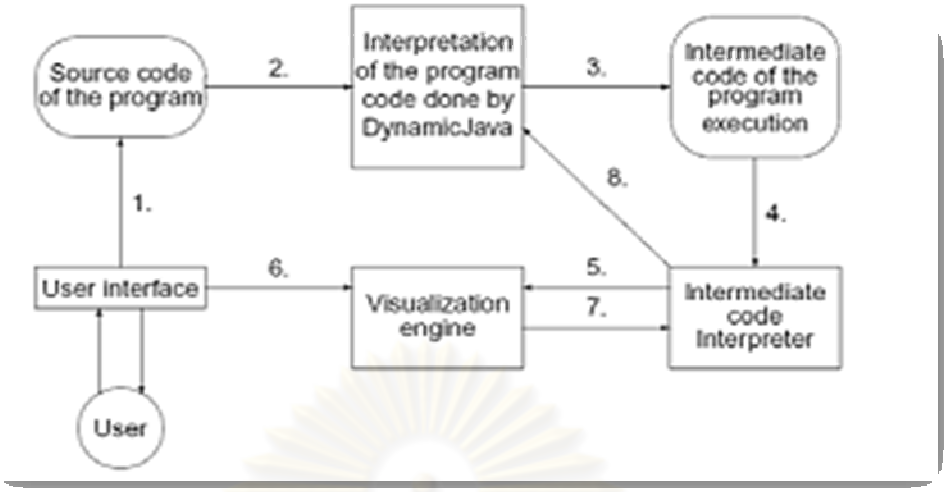
รูปที่ 2.5 การแจงกฎของปัญหา Water jug ใช้อัลกอริทึมค้นหาแนวลึกและ A*

2.3 เครื่องมือช่วยเรียนรู้การค้นหาคำตอบแบบเรียกซ้ำในปริภูมิสถานะ

ส่วนนี้กล่าวถึงเครื่องมือที่ช่วยให้นักเรียนได้เรียนรู้การค้นคำตอบในปริภูมิสถานะด้วยการเรียกเมท็อดตัวมันเอง ซึ่งจะพบได้ในโปรแกรมที่มีการทำงานแบบเรียกซ้ำ โดยมีงานวิจัยที่พัฒนาเครื่องมือที่อธิบายวิธีการทำงานดังกล่าวดังนี้

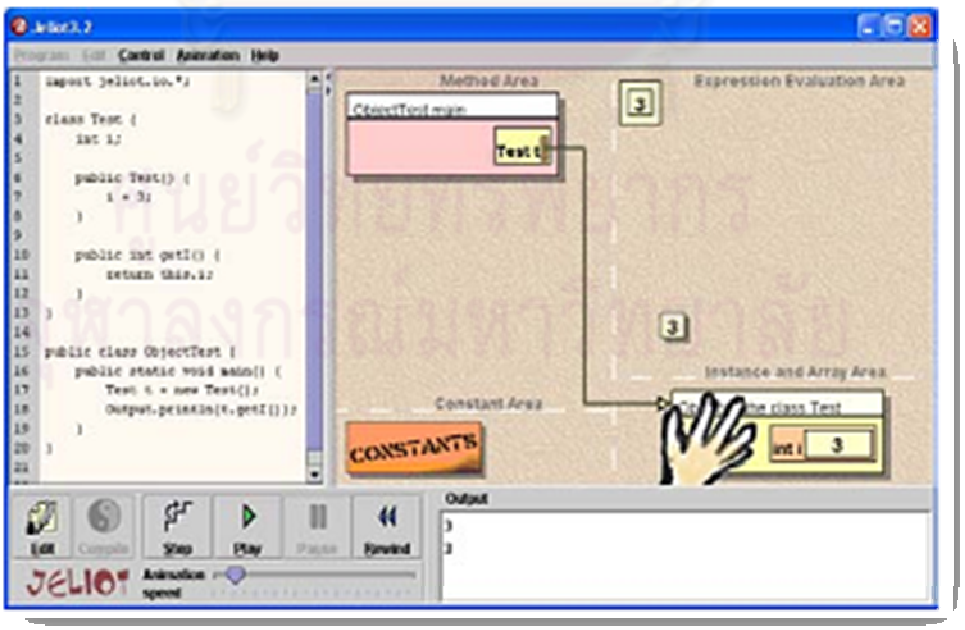
2.3.1 Visualizing Programs with Jeliot 3

Jeliot 3 [8] เป็นระบบแสดงภาพการทำงานของโปรแกรมภาษาจาวา เหมาะสำหรับผู้เริ่มเรียนเขียนโปรแกรม โดยที่ Jeliot 3 นั้นถูกพัฒนาต่อจาก Jeliot 2000 ซึ่งมีความสามารถในการแสดงต้นไม้ที่ติดตามการเรียกเมท็อด (call tree) เมื่อนำไปติดตามเมท็อดที่เขียนค้นคำตอบตามแนวลึกแบบเรียกซ้ำ ซึ่งจะแสดงต้นไม้ปริภูมิสถานะระหว่างการค้นคำตอบในปริภูมิสถานะ



รูปที่ 2.6 โครงฟังก์ชันการทำงานของ Jeliot 3

รูปที่ 2.6 การทำงานของ Jeliot 3 นั้นเริ่มจากการเขียนรหัสโปรแกรมการทำงานเรียกซ้ำ จากนั้นสั่งโปรแกรมทำงาน รหัสคำสั่งที่เขียนจะถูกส่งไปยังตัวแปลคำสั่งเพื่อแปลงเป็น intermediate code และติดตามผลลัพธ์จากตัวแปลคำสั่งและส่งมายังกลไกแสดงภาพ จากนั้นกลไกแสดงภาพจะสร้างและแสดงภาพการทำงานโปรแกรมมายังผู้ใช้ ดังรูปที่ 2.7 ซึ่งนำเสนอส่วนควบคุมการนำเสนอการสร้างภาพเคลื่อนไหว ด้วยปุ่มควบคุมการการเล่น หยุด และย้อนกลับ หรือเล่น แบบ step-by-step ได้

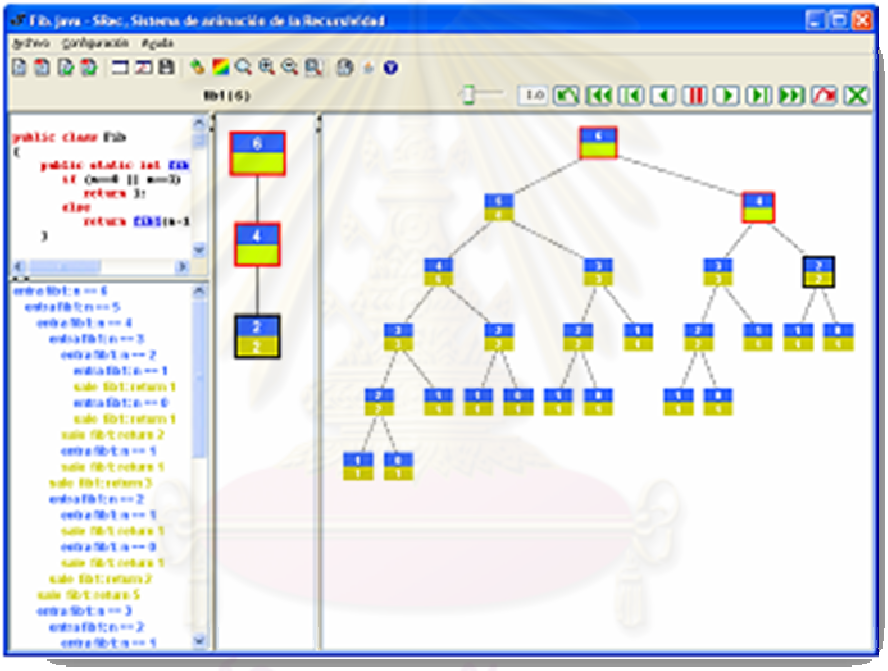


รูปที่ 2.7 ส่วนติดต่อผู้ใช้ของ Jeliot 3

2.3.2 An Animation System of Recursion for Algorithm Courses

SRec [9] เป็นระบบแสดงภาพการเขียนโปรแกรมแบบเรียกซ้ำ พัฒนาจากภาษาจาวาซึ่งช่วยสอนให้เข้าใจถึงการทำงานโปรแกรมเรียกซ้ำ โดยการเขียนรหัสโปรแกรมจาวาแบบเรียกซ้ำลงบน panel ของระบบ จากนั้นสั่งระบบทำงานผ่านส่วนติดต่อผู้ใช้ ระบบจะแสดงภาพการทำงานการค้นคำตอบแบบเรียกซ้ำในลักษณะต้นปริภูมิสถานะ ซึ่งระบบ SRec มีส่วนติดต่อผู้ใช้สำหรับควบคุมการนำเสนอภาพ เช่น การเดินหน้า-ย้อนกลับ เป็นต้น

โดยหลักการทำงานของ SRec นั้นจะอาศัยโปรแกรมตรวจแก้จุดบกพร่องเพื่อติดตามปมสถานะจากการเรียกซ้ำเมท็อดบน stack track จากนั้นระบบจะนำสถานะที่ติดตามได้มาแสดงภาพ ดังรูปที่ 2.8



รูปที่ 2.8 ส่วนติดต่อผู้ใช้ของ SRec

2.4 งานวิจัยที่เกี่ยวข้องกับระบบแสดงภาพปริภูมิสถานะที่นำมาใช้ในด้านต่าง ๆ

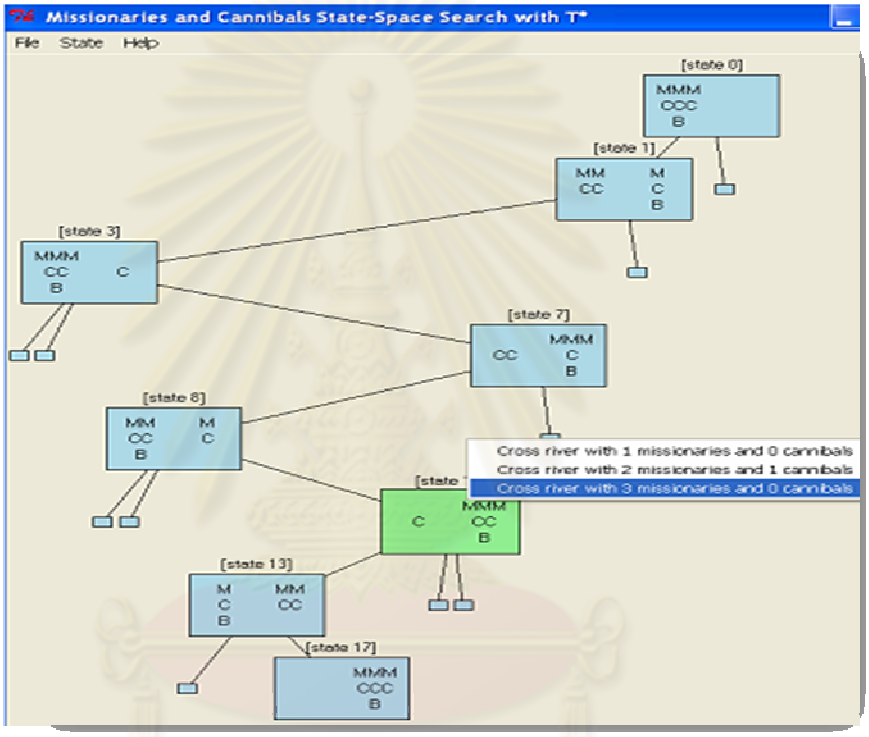
ส่วนนี้กล่าวถึงงานวิจัยที่ออกแบบระบบแสดงภาพปริภูมิสถานะเพื่อใช้นำเสนอภาพการค้นคำตอบบนปริภูมิสถานะในโดเมนต่าง ๆ ซึ่งมีงานวิจัยที่กล่าวถึงดังนี้

2.4.1 A Transparent Interface to State-Space Search Programs

งานวิจัยนี้นำเสนอโดย University of Washington ซึ่งออกแบบส่วนต่อประสานสำหรับโปรแกรมค้นหาในปริภูมิสถานะ ทำหน้าที่เป็นตัวประสานการแสดงภาพต้นไม้ปริภูมิสถานะเรียกว่า T-STAR [10] (T-STAR หรือเขียน T*) ย่อมาจาก “Transparent STate-space search

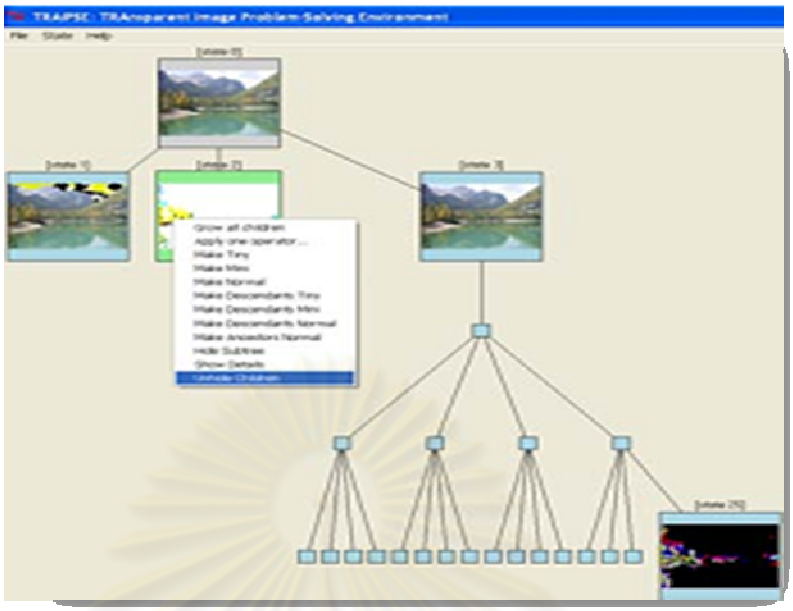
ARchitecture” ซึ่งออกแบบมาในลักษณะ Software module พัฒนาขึ้นภายใต้โปรแกรมภาษา Python และ GUI development kit Tkinter [11] ซึ่งผู้ใช้งานสามารถเรียกใช้งานด้วยการสืบทอดคลาสของ T-STAR ที่ได้จัดเตรียมไว้ โดย T-STAR ได้ถูกนำไปใช้ในงานวิจัยด้านต่าง ๆ เพื่อแสดงให้เห็นถึงภาพการแก้ปัญหา เช่น

การนำ T-STAR สนับสนุน Collaborative Interface ในการแสดงภาพปัญหาทางปัญญาประดิษฐ์ [12] สำหรับดังรูปที่ 2.9 แสดงภาพต้นไม้จากการค้นคำตอบในปริภูมิสถานะของปัญหา The Missionaries and Cannibals Puzzle [13]



รูปที่ 2.9 การแก้ปัญหา Missionaries and Cannibals problem ด้วย T*

การนำ T-STAR มาปรับใช้ในโดเมนของ image processing ซึ่งเป็น toolkit ที่เรียกว่า TRAnsparent Image Problem Solving Environment (TRAIPSE)[14] ช่วยในการแสดงภาพต้นไม้ปริภูมิสถานะจากการค้นในปริภูมิสถานะเพื่อนำมาใช้ออกแบบด้าน Image Processing รูปที่ 2.10



รูปที่ 2.10 ปริภูมิสถานะโดเมนของ image processing ด้วย TRAIPSE

บทที่ 3

แนวคิดและทฤษฎี

บทนี้กล่าวถึงแนวคิดและทฤษฎีที่นำมาใช้เป็นองค์ความรู้ รวมทั้งเป็นแนวทางในการออกแบบระบบแสดงภาพปริภูมิสถานะของงานวิจัยนี้ ซึ่งรายละเอียดที่กล่าวถึงมีดังนี้

3.1 การค้นหาคำตอบปริภูมิสถานะ

ปริภูมิสถานะ คือ เซตของสถานะที่เป็นไปได้ของระบบในระหว่างกระบวนการแก้ปัญหา ดังนั้นการแก้ปัญหาสามารถแทนในรูปของปริภูมิสถานะด้วยการกำหนดสถานะหนึ่งเคลื่อนที่ไปสู่อีกสถานะหนึ่ง จากนั้นใช้ตัวกระทำการที่ใช้กับสถานะเหล่านั้นจนกว่าจะไปถึงสถานะเป้าหมาย วิธีการแก้ปัญหาที่ได้ก็คือ ลำดับของตัวกระทำการที่นำมาใช้ต่อเนื่องกันจนนำไปสู่คำตอบที่ต้องการ วิธีที่ใช้แก้ปัญหาในลักษณะนี้จึงเป็นการค้นหาเส้นทางซึ่งนำไปสู่สถานะเป้าหมาย การค้นหาอย่างมีประสิทธิภาพนับว่าเป็นสิ่งสำคัญมากเพราะสถานะที่เป็นไปได้ทั้งหมดอาจมีจำนวนมากทำให้การค้นหากินเวลามาก

การค้นหาคำตอบปริภูมิสถานะจึงเป็นกระบวนการค้นหาการเปลี่ยนจากสถานะเริ่มต้นเป็นสถานะสุดท้ายโดยผ่านสถานะต่าง ๆ ซึ่งแสดงให้เห็นถึงขั้นตอนในการทำงานที่มีหลายเส้นทางเพื่อให้ได้วิธีการค้นหาเส้นทางที่ดีที่สุด วิธีการค้นหาคำตอบในปริภูมิสถานะมีหลักการพื้นฐานดังนี้

1 กำหนดปริภูมิสถานะต่าง ๆ ที่อาจเป็นไปได้ของปัญหานั้น

2 กำหนดสถานะเริ่มต้นของปัญหา

3 ค้นหาสถานะบางสถานะที่อาจจะยอมรับได้ว่าเป็นข้อแก้ปัญหาผลเฉลย เรียกว่าสถานะคำตอบ

4 ค้นหากฎต่าง ๆ ที่แสดงความสัมพันธ์จากสถานะเริ่มต้นไปยังสถานะคำตอบของปัญหา

ดังนั้นการกำหนดปัญหาเมื่อทราบสถานะเริ่มต้น สถานะคำตอบ รวมทั้งกฎต่าง ๆ แล้วระบบก็จะหาทางแก้ปัญหาด้วยการค้นหากฎที่เกี่ยวข้องซึ่งเมื่อป้อนสถานะเริ่มต้นเข้าไปแล้วก็จะได้สถานะคำตอบเป็นผลลัพธ์ออกมาซึ่งมีรายละเอียดดังต่อไปนี้

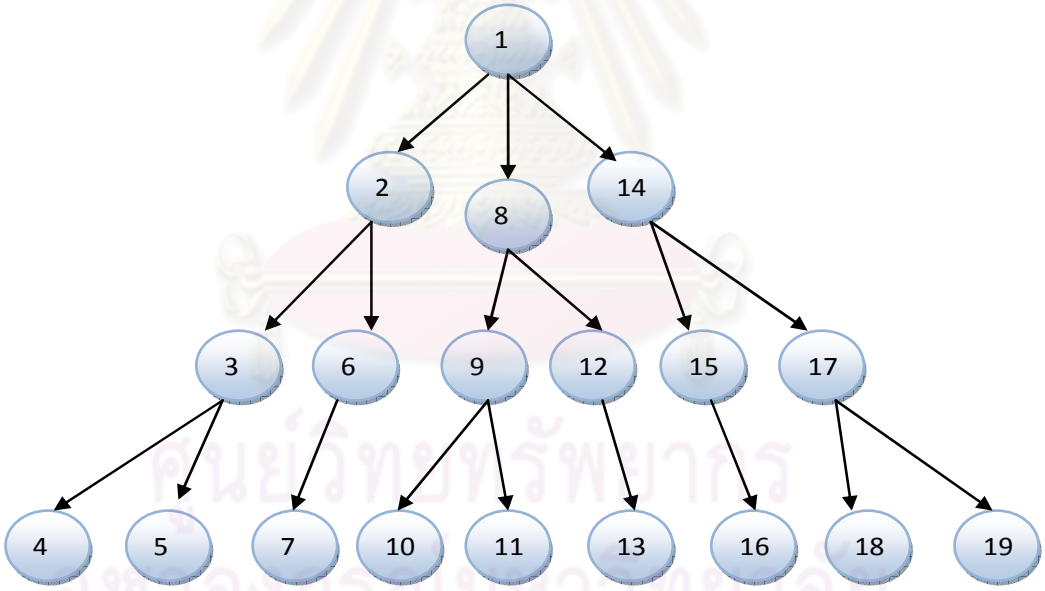
3.1.1 การค้นหาคำตอบของปัญหาด้วยการกำหนดปริภูมิสถานะ

การค้นหาคำตอบของปัญหาด้วยการกำหนดปริภูมิสถานะที่เป็นไปได้ ทำได้ด้วยการเชื่อมโยงความสัมพันธ์ระหว่างสถานะต่าง ๆ ให้ครบถ้วนตามลักษณะของปัญหาเสียก่อน แล้วจึงค้นหาไปตามสถานะต่าง ๆ เพื่อนำไปสู่คำตอบของปัญหาอีกทีหนึ่ง ซึ่งการค้นหาคำตอบในปริภูมิสถานะมักจะกระทำบนโครงสร้างข้อมูลแบบต้นไม้หรือกราฟ ทั้งนี้เพราะโครงสร้างข้อมูลใน

ลักษณะนี้สามารถทำให้การค้นหาทำได้สะดวกและสามารถพลิกแพลงการค้นหาได้ง่าย ในความเป็นจริงแล้วการค้นหาข้อมูลบางครั้งสามารถกระทำบนโครงสร้างข้อมูลชนิดอื่นก็ได้เช่น แถวลำดับ กองซ้อน และแถวคอย แต่การจัดข้อมูลในโครงสร้างเช่นนี้มีข้อจำกัดในการค้นหาข้อมูลซึ่งใช้ได้กับข้อมูลที่มีขนาดเล็ก ดังนั้นในการค้นหาข้อมูลที่มีขนาดใหญ่ ก่อนการค้นหาหรือระหว่างการค้นหาข้อมูลที่จะถูกค้นควรจะต้องถูกจัดให้อยู่ในรูปแบบของต้นไม้หรือกราฟก่อน การค้นคำตอบในปริภูมิสถานะนั้นทำได้หลายวิธีขึ้นอยู่กับการเลือกใช้อัลกอริทึมที่ใช้ในการค้นหาคำตอบ (Searching Algorithm) ซึ่งขอกล่าวถึงอัลกอริทึมค้นหาที่ใช้ในงานวิจัยนี้มีดังนี้

3.1.1.1 การค้นหาตามแนวลึก

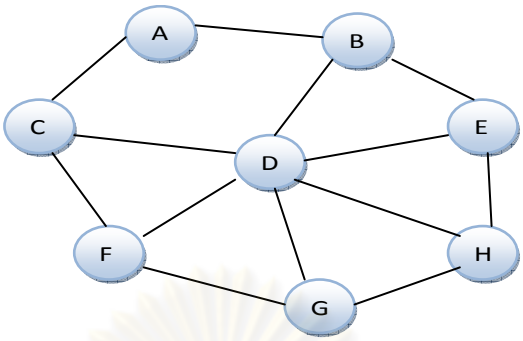
การค้นหาตามลึกเป็นการค้นหาที่กำหนดทิศทางจากรูปของโครงสร้างต้นไม้ โดยเริ่มต้นจากปมรากที่อยู่บนสุดแล้วเดินลงมาให้ลึกที่สุด เมื่อถึงปมล่างสุดให้ย้อนขึ้นมาที่จุดสูงสุดของกิ่งเดียวกันที่มีกิ่งแยกและยังไม่ได้เดินผ่าน แล้วเริ่มเดินลงจนถึงปมลึกสุดอีก ทำเช่นนี้สลับไปเรื่อยจนพบปมที่ต้องการหาหรือสำรวจครบทุกปมแล้วตามรูปที่ 3.1 การค้นหาตามลึกจะมีลำดับการเดินตามปมดังตัวเลขที่กำกับไว้ในแต่ละปม



รูปที่ 3.1 ลำดับการเดินทางบนปมของการค้นหาแบบลึกก่อนบนโครงสร้างต้นไม้

ดังที่ได้กล่าวมาแล้วว่าโครงสร้างข้อมูลที่ใช้สำหรับการค้นหานั้นสามารถใช้กับโครงสร้างกราฟได้โดยอาศัยหลักการเดียวกัน แต่สำหรับการเดินทางบนกราฟนั้นจะไม่มีปมลึกสุดดังนั้นการเดินทางบนกราฟจะต้องปรับวิธีการเป็นดังนี้ โดยเริ่มจากปมเริ่มต้นจากนั้นให้นำปมที่อยู่ติดกับปมที่กำลังสำรวจอยู่ (ที่ยังไม่ได้สำรวจและยังไม่ได้อยู่ในกองซ้อนมาใส่กองซ้อน) มาเก็บไว้ในกองซ้อน เมื่อสำรวจปมนั้นเสร็จให้ดึงปมตัวบนสุดของปมออกมาสำรวจ แล้วนำปมข้างเคียงทั้งหมดที่ยัง

ไม่ได้สำรวจมาต่อท้ายแถวคอยแล้วดึงตัวบนสุดออกมาสำรวจ ทำเช่นนี้เรื่อย ๆ จนกระทั้งพบปมที่ต้องการหรือสำรวจครบทุกปม



รูปที่ 3.2 โครงสร้างข้อมูลแบบกราฟ

รูปที่ 3.2 การสำรวจจะเริ่มต้นที่ A และนำปมข้างเคียง B และ C มาเก็บไว้ในกองซ้อนเมื่อสำรวจ A เสร็จดึงข้อมูลจากกองซ้อนออกมาได้ C จากนั้นสำรวจ C และนำปมข้างเคียงกับ C ที่ยังไม่ได้สำรวจและยังไม่ได้อยู่ในกองซ้อนมาใส่กองซ้อนคือ D และ F มาวางใส่กองซ้อน ดังนั้นในกองซ้อนตอนนี้มี B D F อยู่ เมื่อสำรวจ C เสร็จก็ดึงค่า F ออกมาสำรวจ แล้วนำปมข้างเคียงที่ยังไม่ได้สำรวจและยังไม่ได้อยู่ในกองซ้อนมาใส่กองซ้อนซึ่งก็คือ G ดังนั้นข้อมูลในกองซ้อนจะเป็น B D G ทำเช่นนี้ไปเรื่อย ๆ จนจบการทำงานก็จะได้ลำดับการสำรวจคือ (A C F G H E D B) ตามตาราง 3.1 ดังต่อไปนี้

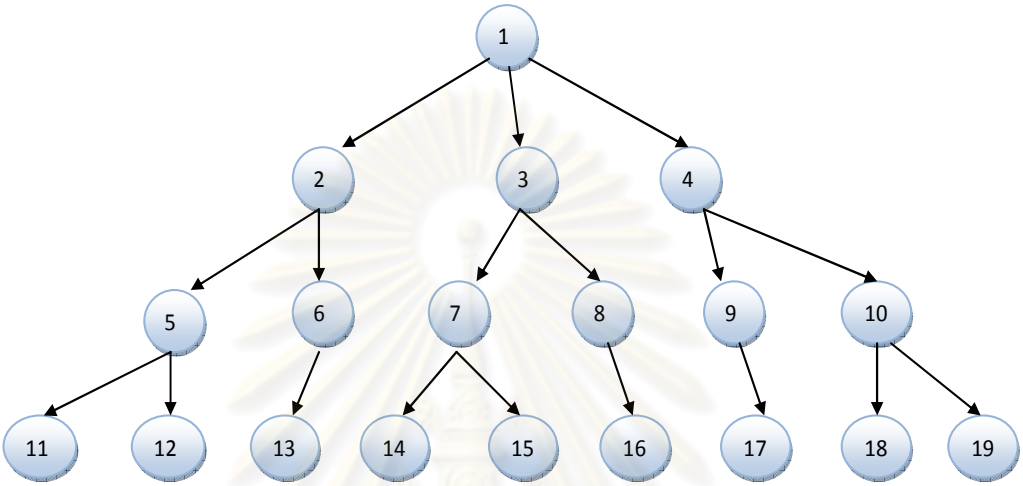
ตารางที่ 3.1 ลำดับการค้นหาตามแนวลึก

ปมสำรวจ	กองซ้อน
A	B C
C	B D F
F	B D G
G	B D H
H	B D E
E	B D
D	B
B	

การค้นหาข้อมูลแบบนี้บนโครงสร้างของกราฟมีข้อที่น่าสังเกตคือ ปมที่เริ่มต้นการสำรวจจะต้องมีการกำหนดมาให้ว่าปมใดเป็นปมเริ่มต้น และข้อสังเกตอีกประการหนึ่งคือวิธีการค้นหาตามแนวลึกก่อนที่ใช้สำหรับโครงสร้างข้อมูลแบบกราฟ สามารถใช้กับโครงสร้างข้อมูลแบบต้นไม้ได้ด้วย

3.1.1.2 การค้นหาตามแนวกว้าง

การค้นหาตามกว้างเป็นการกำหนดทิศทางการค้นหาแบบที่ละระดับของโครงสร้างต้นไม้ โดยเริ่มจากปมราก (ระดับที่ 0) แล้วลงมาระดับที่ 1 จากซ้ายไปขวา เมื่อเสร็จระดับที่ 1 ไประดับที่ 2 จากซ้ายไปขวาเช่นกัน ทำเช่นนี้เรื่อย ๆ จนพบปมที่ต้องการตามรูปที่ 3.3 ลำดับการเดินทางของปมเป็นไปตามหมายเลขที่กำกับไว้บนปม



รูปที่ 3.3 ลำดับการค้นหาแบบกว้างก่อนบนโครงสร้างต้นไม้

สำหรับการค้นหาตามแนวกว้างบนโครงสร้างต้นไม้นั้นจะอาศัยโครงสร้างข้อมูลแบบแถวคอยมาช่วย โดยเริ่มต้นจากสำรวจที่ปมเริ่มต้น(ปมราก) แล้วนำปมข้างเคียงเก็บไว้ในแถวคอย เมื่อสำรวจปมเริ่มต้นเสร็จให้นำข้อมูลในแถวคอยออกมาสำรวจ แล้วนำปมข้างเคียงที่ยังไม่ได้สำรวจและไม่ได้อยู่ในแถวคอยใส่แถวคอยไว้ ทำเช่นนี้ไปเรื่อย ๆ จนพบปมที่ต้องการหรือเมื่อสำรวจครบทุกปม

รูปที่ 3.2 ซึ่งเป็นโครงสร้างกราฟสามารถใช้หลักการเดียวกับการค้นตามแนวลึก โดยการสำรวจเริ่มต้นที่ A นำปมข้างเคียง B C ไว้ในคิว เมื่อสำรวจ A เสร็จนำข้อมูลในแถวคอยคือ B ออกมาสำรวจ แล้วนำข้อมูลข้างเคียงคือ D E ใส่คิว ตอนนี้แถวคอยจะมี B D E อยู่ แล้วนำ B ออกมาสำรวจทำเช่นนี้เรื่อย ๆ จะได้ลำดับการสำรวจข้อมูลคือ (A B C D E F G H) ตามตารางที่ 3.2

ตารางที่ 3.2 ลำดับการค้นหาตามแนวกว้าง

ปมสำรวจ	แถวคอย
A	B C
B	C D E
C	D E F
D	B E F G H
E	F G H
F	G H
G	H
H	

3.1.2 กลวิธีการค้นคำตอบจากกราฟปริภูมิสถานะ

นอกจากการเลือกใช้อัลกอริทึมหาเพื่อค้นหาคำตอบในปริภูมิสถานะแล้วยังมีกลวิธีซึ่งช่วยทำให้การค้นคำตอบนั้นทำได้รวดเร็วขึ้น หรือกล่าวง่าย ๆ คือการใช้วิธีเพื่อช่วยลดจำนวนปมการค้นที่ไม่มีแววหรือไม่มีทางเป็นคำตอบของปัญหาออกจากการพิจารณา นั้นคือเมื่อมั่นใจว่าปมสถานะของผลเฉลยทั้งหลายในต้นไม้ย่อยนั้น ๆ ไม่มีทางเป็นสถานะคำตอบของปัญหาที่กำลังค้นอยู่ ก็ไม่จำเป็นที่จะแวะค้นคำตอบในต้นไม้ย่อยนั้นอีก ซึ่งกลวิธีที่กล่าวถึงในตัวอย่างโจทย์งานวิจัยมีดังนี้

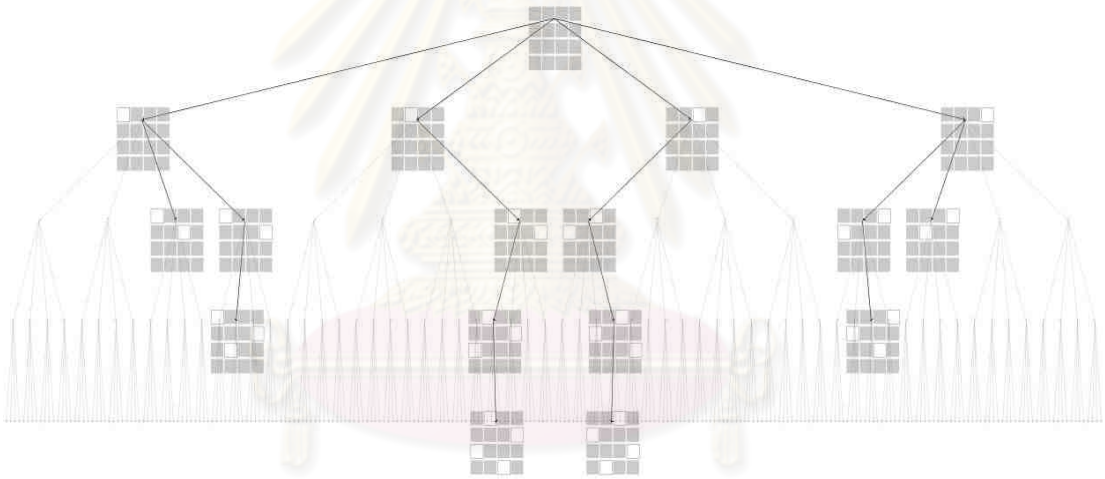
3.1.2.1 การค้นตอบด้วยกลวิธีย้อนรอย

เมื่อพิจารณาแนวความคิดในการตรวจสอบความมีแววของปมก่อนแตกกิ่ง ขณะที่กำลังแวะผ่านปมในต้นไม้ปริภูมิสถานะว่านำไปสู่ปมคำตอบสถานะของคำตอบได้หรือไม่ รวมกับการค้นตามแนวลึกจะได้การค้นที่เรียกว่าการย้อนรอย(backtracking) โดยในตอนแรกนั้นจะเริ่มการค้นจากจุดเริ่มต้นไปตามเส้นทางที่เลือกไว้จนกระทั่งไปพบคำตอบหรือทางตัน ถ้าพบคำตอบก็นำคำตอบมาใช้และหยุดการค้นหาได้ แต่ถ้าพบทางตันก็จะต้องถอยกลับมายังจุดที่มีทางแยกแล้วลองเลือกทางเดินอื่นต่อไปจนกว่าจะพบคำตอบ ในการเขียนฟังก์ชั่นย้อนรอยก็จะมีการตรวจหาจุดที่ไม่มีทางพาไปพบคำตอบเพื่อเลี่ยงไปผ่านจุดอื่น ๆ ต่อไปดังตัวอย่างโปรแกรม รหัสที่ 3.1 ซึ่งเป็นการเขียนฟังก์ชั่นย้อนรอยปัญหา N-Queen[15] โดยฟังก์ชัน `nQueen_BT` เป็นการค้นคำตอบตามแนวลึกแบบเรียกซ้ำ และ `promising` เป็นฟังก์ชันสำหรับตรวจสอบการขยายผลเฉลยว่ามีแววการเป็นผลเฉลยคำตอบหรือไม่ จากรูปที่ 3.4 จะเห็นได้ว่าการค้นคำตอบในปัญหา 4-Queen มีปมที่มีการแวะผ่านและเส้นเชื่อมที่มีการแตกกิ่งระหว่าง การค้นคำตอบด้วยการย้อนรอย โดยต้นไม้นี้มี

ปมทั้งสิ้น 341 ปม แต่เมื่อเสริมกลวิธีแบบย้อนรอยจะการแวะผ่านปมเพียง 17 ปมเท่านั้นก็ได้รูปแบบการวางควีนสองรูปแบบ

```
nQueen_BT( x[1..n] , k)
{
    if( k == 0)for(i=1 to n) x[i] = i
    if ( k == n ) print(x[1..n])
    else {
        for(i=k+1 to n){
            Swap(x[k+1] , x[i]);
        if(promising(x[1..k+1]))
            nQueen_BT(x[1..n],k+1)
        Swap(x[i],x[k+1]);
        }
    }
}
promising(x[1..k]){
    for(j=1 to k-1)
        if(abs(x[j]-x[k])==abs(j-k))return FALSE
    return TRUE
}
```

รหัสที่ 3.1 การเขียนฟังก์ชั่นย้อนรอยปัญหา N-Queen

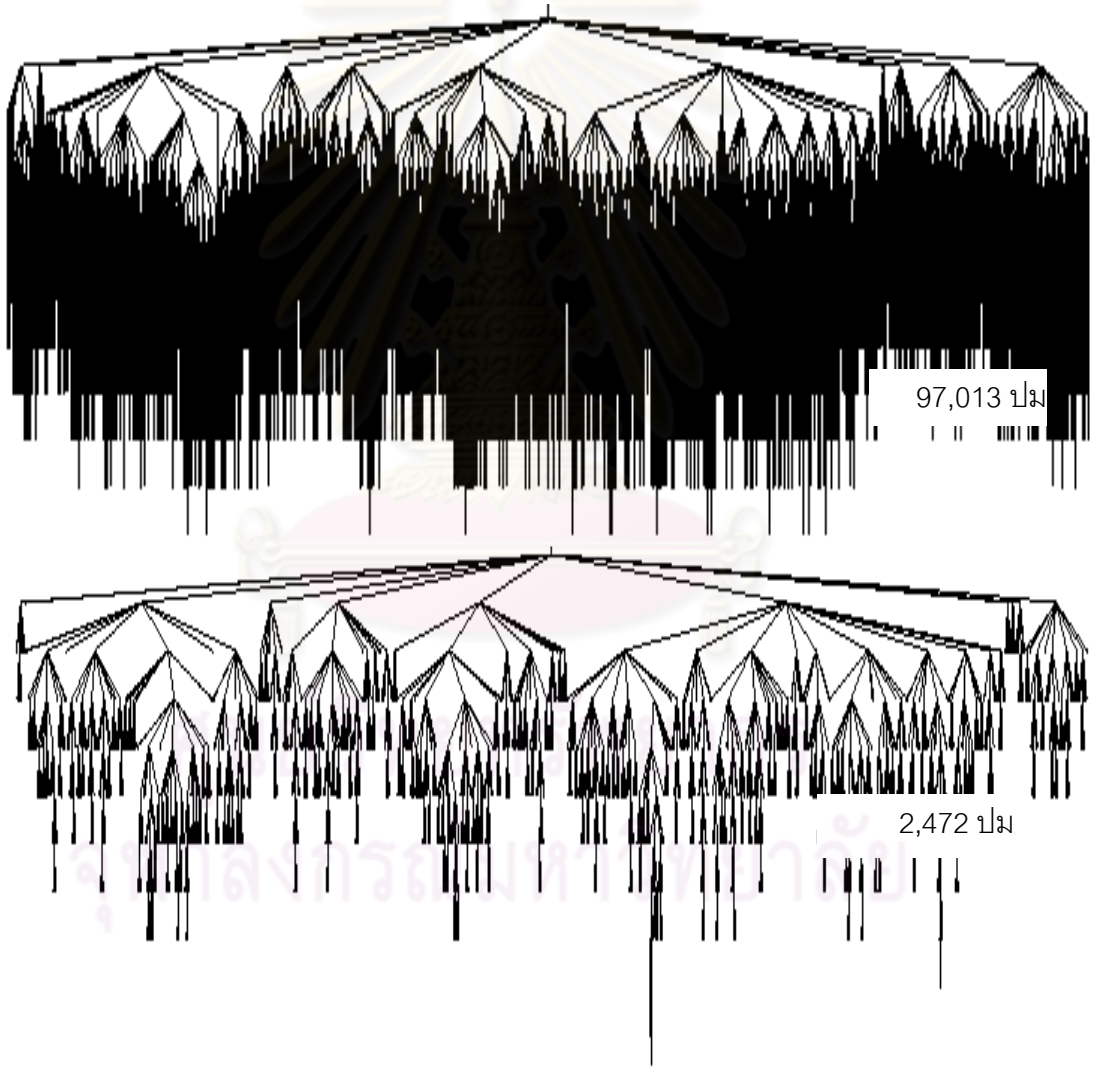


รูปที่ 3.4 การทำงานด้วยกลวิธีย้อนรอยปัญหา 4-Queen

3.1.2.2 การค้นตอบด้วยวิธีการขยายและจำกัดเขต

การขยายและจำกัดเขต คือ การแบ่งหรือแตกกิ่ง (Dividing or Branching) และการตัดหรือการหยุด (Conquering or Fathoming) โดยเริ่มต้นจากปัญหาที่มีขนาดใหญ่และแบ่งเป็นปัญหาย่อย ๆ (Sub problem) จากนั้นพิจารณาขอบเขต (Bounding) ของคำตอบสำหรับปัญหาย่อยและพิจารณาตัดปัญหาที่ไม่สามารถให้คำตอบที่ดีที่สุดในขณะนั้นได้และทำซ้ำกับทุกปัญหาย่อย ๆ จนกระทั่งพบปัญหาย่อยที่ให้คำตอบที่ดีที่สุด ซึ่งอาศัยการคำนวณขอบเขตของต้นทุน (อาจจะเป็นขอบเขตบน หรือ ขอบเขตล่าง ขึ้นอยู่กับลักษณะของฟังก์ชันที่นำมากำหนดจุดประสงค์การหาต้นทุนที่น้อยที่สุดหรือมากที่สุด)วิธีนี้จะสามารถหาคำตอบที่ดีที่สุดได้ในเวลาอันรวดเร็ว

การปัญหาที่นำเอากลวิธีการขยายและจำกัดเขตมาใช้อย่างเช่น Traveling salesman problem (TSP) หรือ ปัญหาการเดินทางของพนักงานขาย ในการหาเส้นทางเดินที่น้อยที่สุดในการแวะตามเมืองต่าง ๆ ที่กำหนดให้ ภายใต้เงื่อนไขที่ว่าจะต้องแวะส่งเมืองใด ๆ ก็ตามได้เพียง 1 ครั้งเท่านั้น ซึ่งการใช้กลวิธีการขยายและจำกัดเขตเพื่อคำนวณหาค่าขีดจำกัดล่าง (Lower bound) จะช่วยลดปมที่ไม่มีแววออกได้จำนวนมาก ดังรูปที่ 3.5 แสดงต้นไม้ปริภูมิสถานะเพื่อค้นคำตอบด้วยวิธีการขยายและจำกัดเขตของปัญหาการเดินทางของพนักงานขายผ่าน 12 เมืองที่ใช้ฟังก์ชันการคำนวณ lower bound ของความยาวการเดินทางที่ต่างกันได้ขนาดของต้นไม้ที่ต่างกัน เห็นได้ชัดว่าฟังก์ชันที่ใช้ในการค้นของต้นไม้ล่างดีกว่าของต้นไม้บนเพราะต้นไม้ล่างถูกเล็มปมออกเป็นจำนวนมาก



รูปที่ 3.5 การค้นคำตอบด้วยวิธี branch and bound ในปัญหาการเดินทางของพนักงานขาย

3.2 สถาปัตยกรรมดีบักเกอร์จาวาแพลตฟอร์ม

JPDA (Java Platform Debugger Architecture) นั้นเป็นสถาปัตยกรรม debugging multi-tier [16] ซึ่งอนุญาตให้ผู้พัฒนาเครื่องมือตรวจจุดบกพร่องโปรแกรมง่ายในการสร้างโปรแกรมตรวจแก้จุดบกพร่อง โดย JPDA นั้นประกอบด้วย 3 เลเยอร์ คือ

1 JVMDI (Java VM Debug Interface) ส่วนนี้เป็นการกำหนดการตรวจแก้จุดบกพร่องสำหรับเตรียมบริการ VM (virtual machine) โดยจะเตรียมการดำเนินการตรวจแก้จุดบกพร่องรวมทั้งการร้องขอข้อมูล (อย่างเช่น stack frame) การดำเนินการ (อย่างเช่น จุดพัก breakpoint) และการแจ้งรายละเอียด (อย่างเช่น ตำแหน่งจุดพัก) โดยโปรแกรมตรวจแก้จุดบกพร่องนั้นอาจจะใช้ข้อมูลของ Virtual Machine นอกเหนือจากนี้ (อย่างเช่น Java Native Interface (JNI) [17])

2 JDWP (Java Debug Wire Protocol) ส่วนนี้เป็นการกำหนดการติดต่อกันระหว่าง Debuggee และกระบวนการตรวจแก้จุดบกพร่อง โดยการกำหนดรูปแบบของข้อมูลและการร้องขอระหว่าง debugger front-end และ debuggee โดยที่ไม่ต้องกำหนดกลไกการเคลื่อนย้าย เช่น socket, serial line, shared memory เป็นต้น ซึ่งเป็นข้อจำกัดของโปรโตคอล โดยอนุญาตให้ debuggee และ debugger front-end รันภายใต้การแยกออกจากกันของ Virtual Machine หรือแพลตฟอร์มได้ โดยข้อมูลและการร้องขอที่ระดับของ Java Virtual Machine Debug Interface(JVMDI) แต่ยังรวมถึงการเพิ่มข้อมูลและการร้องขอเท่าที่จำเป็นโดย bandwidth อย่างเช่น การรวมข้อมูล filtering และ batching เข้าด้วยกัน

3 JDI (Java Debug Interface) จาวามีกลไกสำหรับติดต่อ Java Virtual Machine [18] โดยผ่านภาษาจาวาในระดับบนที่เรียกว่า จาวา-ดีบัก-อินเทอร์เฟส [19] เป็นการกำหนดการทำงาน High-level Java language interface ที่ง่ายต่อผู้พัฒนาเครื่องมือในการใช้งานเพื่อเขียนติดต่อการทำงานกับโปรแกรมตรวจแก้จุดบกพร่อง ในการสะท้อนข้อมูลหรือค่าต่างๆของโปรแกรมที่เขียนของสถานะขณะ runtime บน virtual machine โดย JDI จะเตรียมความสามารถต่างๆ อย่างเช่น suspend, resume threads, set breakpoints, class loading, thread creation The ability to inspect a suspended thread's state, local variables, stack backtrace และ อื่น ๆ จากที่กล่าวมาในการติดต่อ java virtual machine เพื่ออ่านค่าหรือใช้การทำดีบักไปยังJava Debug Wire Protocol (JDWP) หรือ Java Virtual Machine Debug Interface (JVMDI) ผู้พัฒนาเครื่องมือสามารถใช้ Java Debug Interface สำหรับการอำนวยความสะดวกในการพัฒนาตัวตรวจแก้จุดบกพร่องเข้ากับเครื่องมือพัฒนาโปรแกรม

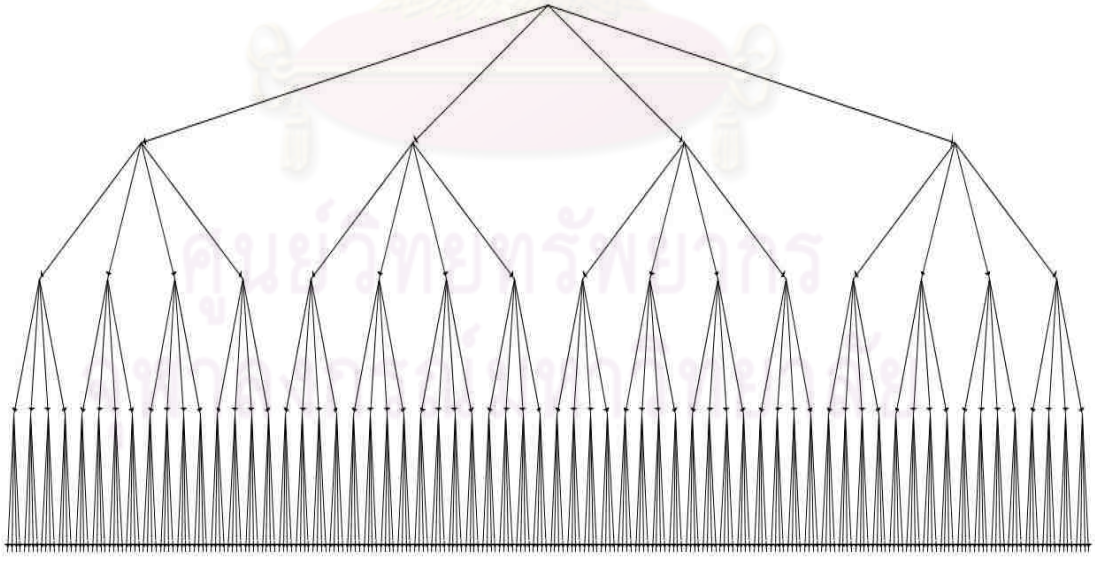
บทที่ 4

การใช้งานระบบแสดงภาพปริภูมิสถานะ

บทนี้จะกล่าวถึงรายละเอียดการใช้งานระบบแสดงภาพปริภูมิสถานะ เพื่อใช้ติดตามและบันทึก การเปลี่ยนแปลงสถานะต่าง ๆ ระหว่างการค้นคำตอบในปริภูมิสถานะ โดยแสดงผลในรูปของต้นไม้ปริภูมิสถานะที่ปรับเปลี่ยนตามสถานะที่เกิดขึ้น ซึ่งผู้ใช้สามารถดูเหตุการณ์การเปลี่ยนแปลงสถานะเหล่านี้ย้อนหลังหรือเดินหน้าได้ ตัวระบบแสดงภาพรองรับสถานะในปริภูมิที่สร้างด้วยอ็อบเจ็กต์จากการเขียนโปรแกรมค้นคำตอบแบบวงวนทำซ้ำ หรือเป็นสถานะที่เก็บใน Stack trace จากการเขียนโปรแกรมแบบเรียกซ้ำเมท็อด

4.1 ภาพรวมการใช้งานระบบแสดงภาพปริภูมิสถานะ

การใช้งานระบบแสดงภาพปริภูมิสถานะของงานวิจัยนี้ เพียงแค่ผู้เขียนโปรแกรมภาษาจาวา เขียนโปรแกรมค้นปริภูมิ ซึ่งเป็นแบบเรียกซ้ำเมท็อดหรือแบบวงวนทำซ้ำที่มีโครงสร้างข้อมูลอย่างเช่น กองซ้อน หรือ แถวคอย ในการเก็บลำดับอ็อบเจกต์ที่เป็นปมสถานะของคำตอบ จากนั้นแทรกคำสั่งที่งานวิจัยนี้กำหนดลงไปยังโปรแกรมค้นปริภูมิ เมื่อสั่งประมวลผลโปรแกรม ระบบแสดงภาพจะติดตามปมสถานะเกิดใหม่ เพื่อนำมาสร้างและแสดงภาพในรูปของต้นไม้ปริภูมิสถานะที่ปรับเปลี่ยนตามสถานะที่เกิดขึ้น ตัวอย่างรูปที่ 4.1 ซึ่งแสดงภาพต้นไม้ปริภูมิสถานะที่ได้จากการค้นปริภูมิของปัญหา 4-Queen



รูปที่ 4.1 การค้นปริภูมิของปัญหา 4-Queen

ขอยกตัวอย่างการเขียนโปรแกรมค้นปริภูมิของปัญหา 4-Queen เพื่อนำเสนอการใช้งานระบบแสดงภาพปริภูมิสถานะในงานวิจัยนี้ โดยแบบแรกเป็นการเขียนโปรแกรมแบบเรียกซ้ำเมท็อด และแบบที่สองการเขียนโปรแกรมแบบวงวนทำซ้ำ ซึ่งรายละเอียดมีดังนี้

4.1.1 การเขียนโปรแกรมค้นปริภูมิแบบเรียกซ้ำ

การเขียนโปรแกรมแบบเรียกซ้ำเป็นการเขียนโปรแกรมโดยเรียกใช้เมท็อดตัวมันเองซ้ำไปเรื่อย ๆ เมื่อผ่านกรรมวิธีแก้ปัญหาแบบเรียกซ้ำตัวเองแล้ว ปัญหาจะถูกแบ่งออกเป็นส่วนย่อย ๆ จำนวนมาก ซ้อนกันเป็นชั้น ๆ แต่ละชั้นใช้วิธีการแก้ปัญหาแบบเดียวกันจนกระทั่งกลายเป็นปัญหาย่อยเล็กที่สุดที่สามารถหาคำตอบได้ โดยขอยกตัวอย่างการเขียนโปรแกรมแบบเรียกซ้ำการค้นปริภูมิของปัญหา 4-Queen ดังนี้

```
public class Queens {
    public void mainDFS_RCS() {
        int[] col = new int[4];
        queensRCS(col, 0);
    }
    public void queensRCS(int[] col, int i) {
        if (i == col.length) printQueens(col, col.length);
        else {
            for (int j = 0; j < col.length; j++) {
                col[i] = j;
                if (isConsistent(col, i)) queensRCS(col, i + 1);
            }
        }
    }
    public boolean isConsistent(int[] col, int n) {
        for (int i = 0; i < n; i++) {
            if (col[n] == col[i] || Math.abs(col[n] - col[i])
                == Math.abs(i - n))
                return false;
        }
        return true;
    }
    public void printQueens(int[] col ,int n) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (col[i] == j) System.out.print("Q ");
                else System.out.print("* ");
            }
            System.out.println();
        }
        System.out.println();
    }
}
```

เมท็อดดำเนินงาน

เมท็อดเรียกซ้ำ

รหัสที่ 4.1 การแจงผลเฉลยของปัญหา 4-Queen ด้วยการค้นตามแนวลึกเรียกซ้ำ

จากรหัสที่ 4.1 คลาส `Queens` เป็นคลาสแก้ปัญหา 4-Queen ซึ่งมีเมท็อดเรียกซ้ำ `queensRCS` ทำหน้าที่ค้นคำตอบในปริภูมิ โดยมีเมท็อด `isConsistent` ตรวจความถูกต้องของการวางควีน และเมท็อด `mainDFS_RCS` ทำหน้าที่กำหนดค่าเริ่มต้นสำหรับค้นปริภูมิและเรียกการทำงานเมท็อดเรียกซ้ำ

4.1.2 การเขียนโปรแกรมค้นปริภูมิแบบวงวนทำซ้ำ

การค้นปริภูมิสามารถใช้การเขียนโปรแกรมค้นคำตอบด้วยวงวนทำซ้ำและอาศัยโครงสร้างข้อมูล อย่างเช่น แถวคอย กองซ้อน เป็นต้น มาช่วยเก็บลำดับการแจงผลเฉลยในปริภูมิสถานะ ซึ่งผลเฉลยที่ได้ระหว่างค้นคำตอบนั้นขอเรียกว่า “ปมสถานะ” ซึ่งปมสถานะนี้ถูกสร้างขึ้นด้วยคลาสที่เรียกว่า “คลาสสถานะ” โดยขอยกตัวอย่างการเขียนโปรแกรมแบบวงวนทำซ้ำที่อาศัยโครงสร้างข้อมูลกองซ้อน มาเป็นตัวอย่างการค้นปริภูมิของปัญหา 4-Queen ดังรหัสที่ 4.2 นำเสนอคลาสค้นปริภูมิสถานะ และรหัสที่ 4.3 นำเสนอคลาสสถานะที่เป็นคลาสผลเฉลยในปริภูมิ

```
import java.util.*;
public class Queens{
    public void mainDFS_ITR() {
        queensITR(4);
    }
    public void queensITR(int n) {
        Stack<Node> s = new Stack<Node>();
        Node root = new Node(new int[1]);
        s.push(root);
        while (!s.isEmpty()) {
            root = s.pop();
            if (root.col.length - 1 == n)
                printQueens(root.col,n);
            else {
                for (int j = 0; j < n; j++){
                    int[] y = Arrays.copyOf(root.col, root.col.length + 1);
                    y[root.col.length - 1] = j;
                    if (isConsistent(y, root.col.length - 1)) {
                        Node node = new Node(y);
                        s.push(node);
                    }
                }
            }
        }
    }
    public boolean isConsistent(int[] col, int n) {
        for (int i = 0; i < n; i++) {
            if (col[n] == col[i] || Math.abs(col[n] - col[i]) == Math.abs(i
- n))
                return false;
        }
        return true;
    }
    public void printQueens(int[] col ,int n) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (col[i] == j) System.out.print("Q ");
                else   System.out.print("* ");
            }
            System.out.println();
        }
        System.out.println();
    }
}
```

เตรียมค่าเริ่มต้นและเรียกเมท็อด `queensITR`

กองซ้อนเก็บลำดับปมสถานะ

วงวนทำซ้ำ

ปมสถานะที่ถูกผลิตถูกสร้างด้วยคลาส `Node`

รหัสที่ 4.2 การแจงผลเฉลยของปัญหา 4-Queen ด้วยการค้นตามแนวลึก

จากรหัสที่ 4.1 คลาส Queens เป็นคลาสแก้ปัญหา 4-Queen ซึ่งมีเมท็อด queensITR ทำหน้าที่ค้นปริภูมิด้วยวงวนทำซ้ำและมี Stack เก็บลำดับปมสถานะที่ถูกผลิต (ปมสถานะสร้างด้วยคลาส Node) โดยมีเมท็อด mainDFS_ITR ทำหน้าที่เตรียมค่าเริ่มต้นสำหรับค้นปริภูมิและเรียกเมท็อด queensITR

```
public class Node {
    int []col;//คอลัมน์วางควีน
    Node(int []col) { this.col = col; }
}
```

คลาสสถานะ

รหัสที่ 4.3 คลาสสถานะ

จากรหัสที่ 4.3 คลาส Node เป็นคลาสสถานะ เมื่อปมผลเฉลยปัญหาถูกค้นพบ คลาสนี้จะถูกสร้างขึ้นและเก็บอ็อบเจกต์เข้าสู่กองซ้อน เพื่อค้นคำตอบในปมสถานะถัดไป

4.3 การแทรกรหัสคำสั่ง

การแสดงภาพต้นไม้ปริภูมิสถานะดังรูปที่ 4.2 ด้วยระบบแสดงภาพในงานวิจัยนี้ สามารถทำได้ด้วยการแทรกรหัสคำสั่งอย่างง่ายที่ระบบแสดงภาพปริภูมิสถานะกำหนด ลงในโปรแกรมค้นปริภูมิสถานะที่เขียนขึ้น (แบบวงวนทำซ้ำ หรือ แบบเมท็อดเรียกซ้ำ) ซึ่งรายละเอียดการแทรกรหัสคำสั่งทำได้ดังนี้

รูปที่ 4.2 การค้นปริภูมิของปัญหา 4-Queen และเสริมด้วยกลวิธีย้อนรอย

4.3.1 การแทรกรหัสคำสั่งในโปรแกรมค้นปริภูมิแบบเรียกซ้ำ

การแทรกรหัสคำสั่งลงบนโปรแกรมค้นปริภูมิแบบเรียกซ้ำ ทำได้ด้วยการเติม Annotation (ภาคผนวก ก) @VMain กำกับบนเมท็อดดำเนินงาน โดยระบุข้อกำหนดภายใน @VMain เป็น recursiveMethod="ชื่อเมท็อดเรียกซ้ำ" ระบุชื่อของเมท็อดเรียกซ้ำ (ภาคผนวก ข) ดังตัวอย่างเช่น

```
@VMain(recursiveMethod="queensRCS")
```

หมายถึงการค้นปริภูมิใช้เมท็อดเรียกซ้ำ queensRCS แจงผลเฉลยปัญหา

```
import jstate101.*;
public class Queens {
    @VMain(recursiveMethod = “queensRCS”)
    public void mainDFS_RCS() {
        int[] col = new int[4];
        queensRCS(col, 0);
    }
    public void queensRCS(int[] col, int i) {
        if (i == col.length) printQueens(col, col.length);
        else {
            for (int j = 0; j < col.length; j++) {
                col[i] = j;
                if (isConsistent(col, i)) queensRCS(col, i + 1);
            }
        }
    }
    public void printQueens(int[] col, int n) { . . . }
    public boolean isConsistent(int[] col, int n){ . . . }
}
```

แทรก @VMain บนเมท็อดดำเนินงานและระบุชื่อเมท็อดเรียก

เมท็อดเรียกซ้ำ

รหัสที่ 4.4 การแทรกรหัสกำกับ @VMain บนแบบโปรแกรมเรียกซ้ำ

รหัสที่ 4.4 เขียน @VMain(recursiveMethod="queensRCS") กำกับเมท็อด mainDFS_RCS ซึ่งดำเนินงานสั่งเมท็อด queensRCS ทำงาน จากนั้นระบบแสดงภาพจะคอยติดตามและบันทึกเหตุการณ์ทุกครั้งที่เมท็อด queensRCS ถูกเรียก โดยจะบันทึกความสัมพันธ์ของการเรียกเมท็อดเพื่อใช้แสดงภาพต้นไม้ปริภูมิสถานะ

4.3.2 การแทรกรหัสคำสั่งลงบนโปรแกรมวงวนทำซ้ำ

การแทรกรหัสคำสั่งลงบนโปรแกรมค้นคำตอบด้วยวงวนทำซ้ำนั้นแบ่งออกเป็นสองส่วน ส่วนแรกแทรกลงบนคลาสค้นคำตอบในปริภูมิสถานะ โดยการประกาศ Annotation @VMain กำกับบนเมท็อดดำเนินงาน ซึ่งข้อกำหนดภายใน @VMain ประกอบด้วย stateClass="ชื่อคลาสสถานะ" ระบุชื่อสตริงของคลาสสถานะ ดังรหัสที่ 4.5

```
import java.util.*;
import jstate101.*;
public class Queens{
    @VMain(stateClass = “Node”)
    public void mainDFS_ITR() {
        queensITR(4);
    }
    public void queensITR(int n) {
        Stack<Node> s = new Stack<Node>();
        Node root = new Node(new int[1], null);
        s.push(root);
        while (!s.isEmpty()) {
            root = s.pop();
            if (root.col.length - 1 == n)
                printQueens(root.col,n);
            else {
                for (int j = 0; j < n; j++){
                    int[] y = Arrays.copyOf(root.col, root.col.length + 1);
                    y[root.col.length - 1] = j;
                    if (isConsistent(y, root.col.length - 1)) {
                        Node node = new Node(y ,root);
                        s.push(node);
                    }
```

แทรก @VMain บนเมท็อดดำเนินงานและระบุชื่อคลาสสถานะ

```
                }
            }
        }
    }
    public void printQueens(int[] col, int n){ . . . }
    public boolean isConsistent(int[] col, int n){ . . . }
}
```

รหัสที่ 4.5 การแทรกรหัสกำกับ `@VMain` บนคลาสค้นคำตอบในปริภูมิสถานะ

รหัสที่ 4.5 แทรกคำสั่ง `@VMain(stateClass="Node")` บนเมท็อด `mainDFS_ITR` ซึ่งเป็นเมท็อดดำเนินงาน กำหนดว่าอ็อบเจกต์ของคลาส `Node` คืออ็อบเจกต์สถานะที่ระบบแสดงภาพต้องคอยติดตามและบันทึกเหตุการณ์การเกิดอ็อบเจกต์ใหม่ของคลาสนี้ ซึ่งมีเมท็อด `mainDFS_ITR` สั่งเมท็อด `queensITR` ค้นปริภูมิ

ส่วนที่สองนั้นแทรกรหัสคำสั่งลงบนคลาสสถานะ ซึ่งเป็นคลาสที่ระบบต้องคอยติดตามและบันทึกเหตุการณ์การเกิดอ็อบเจกต์ใหม่ของคลาส ภายในคลาสระบุความสัมพันธ์ของสถานะพ่อแม่ วิธีที่ระบบแสดงภาพจะรับทราบข้อกำหนดของคลาสสถานะได้นั้นจะต้องเป็นคลาสลูกของ `VState` เพื่อให้ระบบแสดงภาพต้นไม้ปริภูมิสถานะรับทราบโครงสร้างและความสัมพันธ์ของสถานะที่เกิดใหม่ของคลาสนี้ ดังรหัสที่ 4.6

```
public class Node extends VState {
    int []col; Node parent;
    Node(int []col, Node parent) {
        this.col = col;
        this.parent = parent;//รับปมพ่อแม่
        VState.fireEvent(this);
    }
    @Override
    public VState getParent(){
        return parent;
    }
}
```

คลาสสถานะเป็นคลาสลูกของ VState

รหัสติดตามสถานะ

เมท็อดบังคับคืนปมพ่อแม่

รหัสที่ 4.6 การแทรกรหัสคำสั่งลงบนคลาสสถานะ

รหัสที่ 4.6 เขียนคลาส `Node` เป็นคลาสลูกของ `VState` ซึ่งเป็นคลาสแบบ abstract ที่บังคับให้เขียนเมท็อด `getParent` คืนปมพ่อแม่เพื่อให้ระบบรับทราบความสัมพันธ์ของสถานะ จากนั้นแทรกรหัสติดตามสถานะด้วย `VState.fireEvent(this)`

4.4 การเรียกระบบแสดงภาพปริภูมิสถานะทำงาน

เมื่อโปรแกรมค้นคำตอบในปริภูมิสถานะถูกแทรกรหัสคำสั่งที่ระบบแสดงภาพกำหนดแล้ว การสั่งระบบแสดงภาพทำงานสามารถทำได้ด้วยการส่งอ็อบเจกต์โปรแกรมค้นคำตอบในปริภูมิสถานะให้กับเมท็อด `VEngine.begin(Object o)` ดังแสดงในรหัสที่ 4.7 ด้วย

`VEngine.begin(new Queens())` ซึ่งเขียนในเมท็อด `main` เมื่อสั่งทำงานจะได้ภาพต้นไม้ปริภูมิสถานะดังรูปที่ 4.2

```
import jstate101.*;
public class Queens {
    . . .
    public static void main(String[] args) {
        VEngine.begin(new Queens());
    }
}
```

สั่งระบบแสดงภาพทำงาน

รหัสที่ 4.7 การเรียกระบบแสดงภาพปริภูมิสถานะทำงาน

4.5 การแสดงผลการค้นปริภูมิร่วมกัน

ระบบแสดงภาพปริภูมิสถานะสามารถแสดงภาพต้นไม้ปริภูมิสถานะหลายๆภาพพร้อมกันได้ ด้วยการเขียนโปรแกรมค้นปริภูมิสถานะในแต่ละวิธีลงในโปรแกรมเดียวกัน เช่น การเขียนโปรแกรมแก้ปัญหา 4-Queens ด้วยเมท็อด `queensRCS` และ `queensITR` ไว้ในโปรแกรมเดียวกันได้ ดังรหัสที่ 4.8 จากนั้นระบบแสดงปริภูมิสถานะจะติดตามปมสถานะที่ได้จากการค้นคำตอบของวิธีนั้น ๆ ไปสร้างต้นไม้ปริภูมิสถานะของแต่ละวิธีได้อัตโนมัติ

```
import jstate101.*;
import java.util.*;
public class Queens {
    static class Node extends VState {
        int []col; Node parent;
        public Node(int []col, Node parent) {
            this.col = col; this.parent= parent;
            VState.fireEvent(this);
         }
        @Override
        public VState getParent(){
            return parent;
        }
    }
    public static void main(String[] args) {
        VEngine.begin(new Queens());
    }
    @VMain(recursiveMethod = "queensRCS")
    public void mainDFS_RCS() {
        int[] col = new int[4];
        queensRCS(col, 0);
    }
    @VMain(stateClass = "Node")
    public void mainDFS_ITR() {
        queensITR(4);
    }
    public void queensRCS(int[] col, int i) {
        if (i == col.length) printQueens(col, col.length);
        else {
            for (int j = 0; j < col.length; j++) {
                col[i] = j;
                if (isConsistent(col, i)) queensRCS(col, i + 1);
            }
        }
    }
```

คลาสสถานะ Node

แบบเรียกซ้ำ

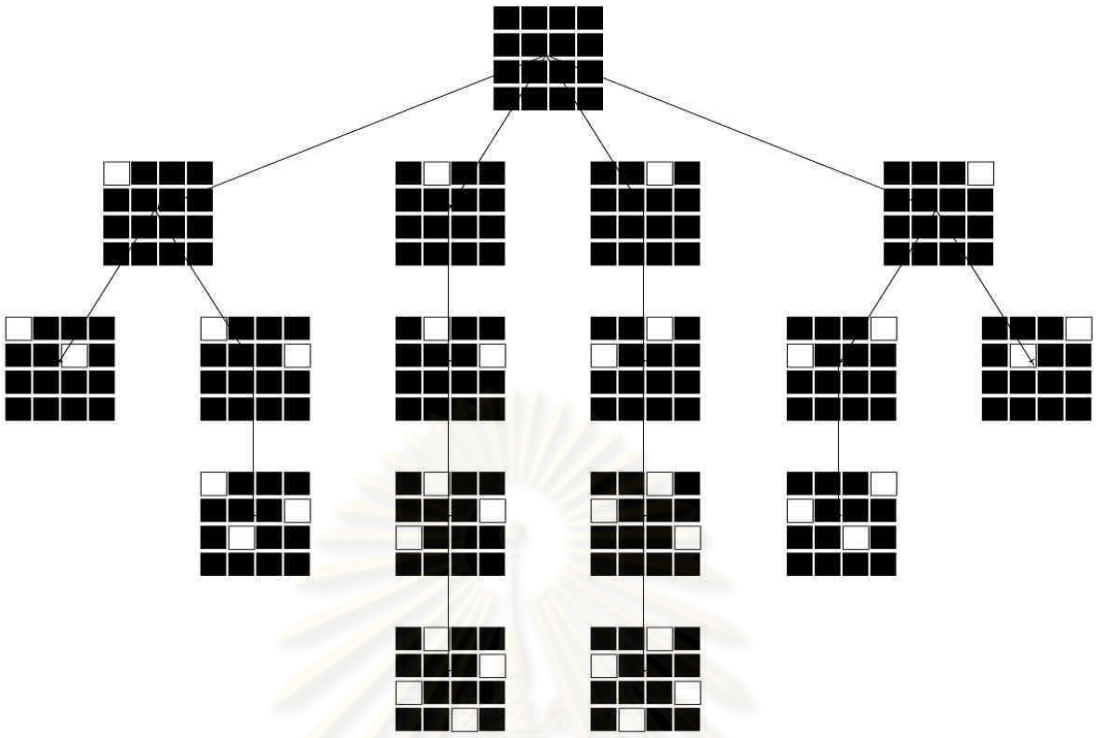
```
    public void queensITR(int n) {
        Stack<Node> s = new Stack<Node>();
        Node root = new Node(new int[1], null);
        s.push(root);
        while (!s.isEmpty()) {
            root = s.pop();
            if (root.col.length - 1 == n)
                printQueens(root.col, n);
            else {
                for (int j = 0; j < n; j++){
                    int[] y = Arrays.copyOf(root.col, root.col.length + 1);
                    y[root.col.length - 1] = j;
                    if (isConsistent(y, root.col.length - 1)) {
                        Node node = new Node(y, root);
                        s.push(node);
                    }
                }
            }
        }
    }
    public void printQueens(int[] col , int n) {. . .}
    public boolean isConsistent(int[] col, int n){ . . . }
}
```

แบบวงวนทำซ้ำ

รหัสที่ 4.8 การแสดงผลโปรแกรมค้นปริภูมิร่วมกัน

4.6 การแสดงภาพกำกับปมของต้นไม้ปริภูมิสถานะ

ระบบแสดงภาพปริภูมิสถานะอนุญาตให้ผู้เขียนโปรแกรมปริภูมิสถานะสามารถแสดงภาพกำกับปมสถานะที่เกิดขึ้นระหว่างแจงผลเฉลยของปัญหาแต่ละปมได้ เพื่อเสริมความเข้าใจในลักษณะหรือรายละเอียดในปมสถานะนั้น โดยอาศัยพื้นฐานการวาดภาพอย่างง่ายของภาษาจาวา เพื่อที่ผู้เขียนโปรแกรมค้นคำตอบในปริภูมิสถานะไม่จำเป็นต้องศึกษารายละเอียดการวาดภาพมากนัก ดังรูปที่ 4.3 แสดงภาพต้นไม้ปริภูมิสถานะการค้นคำตอบปัญหา 4-Queen ด้วยการค้นตามแนวลึกและเสริมด้วยกลวิธีย้อนรอย ซึ่งแต่ละปมสถานะของปัญหามีการแสดงตาราง 4x4 และการวาง Queen ลงบนตารางในช่องสีขาว



รูปที่ 4.3 การวาดภาพบนปมค้นคำตอบปัญหา 4-Queen

4.6.1 การแสดงภาพกำกับปมบนโปรแกรมเรียกซ้ำ

การแสดงภาพกำกับปมนั้นสามารถทำได้โดยการเขียนคลาสค้นปริภูมิเป็นคลาสลูกของ `VDrawable` ซึ่งมีเมท็อดบังคับ `drawState` เพื่อใช้วาดรูป ดังรหัสที่ 4.9

```
@Override
public void drawState(Graphics2D g2d, int x, int y, List args) { . . . }
```

รหัสที่ 4.9 การเขียนเมท็อดวาดภาพบนปมโปรแกรมเรียกซ้ำ

ทุกครั้งที่เมท็อดเรียกซ้ำถูกเรียก ระบบแสดงภาพจะติดตามและจัดเก็บค่าพารามิเตอร์ของเมท็อดเรียกซ้ำไว้ในลิสต์ตามลำดับ ลิสต์นี้จะถูกส่งต่อมายังเมท็อด `drawState` เพื่อนำไปใช้วาดรูปการวาง Queen บนตารางแต่ละปม ดังรูปที่ 4.3 เพื่อให้เมท็อดวาดภาพ `drawState` (รหัสที่ 4.9) รับทราบตำแหน่งที่ต้องแสดง Queen ลงไปวางบนตาราง

```
import java.awt.*;
import java.util.*;
import jstate101.*;
public class Queens extends VDrawable {
    public static void main(String[] args) {
        VEngine.begin(new Queens());
    }
    @VMain(recursiveMethod = "queensRCS")
    public void mainDFS_RCS() {
        int[] col = new int[4];
        queensRCS(col, 0);
    }
    public void queensRCS(int[] col, int i) {
        if (i == col.length)
            printQueens(col , col.length);
        else {
            for (int j = 0; j < col.length;j++){
                col[i] = j;
                if(isConsistent(col, i))
                    queensRCS(col, i + 1);
            }
        }
    }
    @Override
    public void drawState(Graphics2D g2d, int x, int y, List args) {
        g2d.setColor(Color.black); int x_pos = x - 25;int y_pos = y - 30;
        int col[] = (int[]) args.get(0);
        for (int i = 0; i < col.length; i++) {
            for (int j = 0; j < col.length; j++){
                if (j % col.length == 0) {
                    x_pos = x - 40; y_pos += 29;
                } else x_pos += 27;
                if (col[i] == j)
                    g2d.draw3DRect(x_pos, y_pos - 40, 25, 25, true);
                else
                    g2d.fillRect(x_pos, y_pos - 40, 25, 25);
            }
        }
    }
    public boolean isConsistent(int[] col, int n) {. . .}
    public void printQueens(int[] col, int n) {. . .}
}
```

ขยายคลาส VDrawable สำหรับวาดรูป

พารามิเตอร์ `int[] col,int i` ถูกติดตามและเก็บในลิสต์

บังคับเขียนเมท็อดวาดปม drawstate

ดึงค่าอาเรย์ `int[]` การวางตำแหน่ง Queen จากลิสต์

รหัสที่ 4.10 การเขียนเมท็อดวาดรูปปมบนโปรแกรมเรียกซ้ำ

รหัสที่ 4.10 แสดงคลาส `Queens` ที่เป็นคลาสลูกของ `VDrawable` ซึ่งเป็นคลาสสำหรับวาดรูปปมบนโปรแกรมแบบเรียกซ้ำ บังคับเขียนเมท็อด `drawState` โดย `Graphics2D g2d` เป็นจาวากราฟิกสำหรับวาดภาพบนตำแหน่ง x, y และ `List args` เป็นลิสต์ที่เก็บพารามิเตอร์ของเมท็อดเรียกซ้ำ `queensRCS` (ค่า `int[] col` และค่า `int i`) ระบบแสดงภาพจะคอยติดตามพารามิเตอร์เหล่านี้ และส่งต่อให้กับเมท็อด `drawState` เพื่อใช้วาดภาพกำกับปมสถานะ

4.6.2 การแสดงภาพกำกับปมบนโปรแกรมวงวนทำซ้ำ

การแสดงภาพกำกับปมจากการเขียนโปรแกรมค้นคำตอบด้วยวงวนทำซ้ำทำได้ด้วยการ Override เมท็อด `drawState` ดังรหัสที่ 4.11 บนคลาสสถานะที่เป็นคลาสลูกของ `VState` ไว้แล้ว

```
@Override
public void drawState(Graphics2D g2d, int x, int y) {. . .}
```

รหัสที่ 4.11 เมท็อดวาดรูปปมสถานะ

รหัสที่ 4.12 แสดงคลาส `Node` ซึ่งเป็นคลาสสถานะที่เป็นคลาสลูกของ `VState` ไว้แล้ว ซึ่ง Override เมท็อด `drawState` ในคลาส `Node` เพื่อวาดตารางและวาง Queen ลงบนตาราง

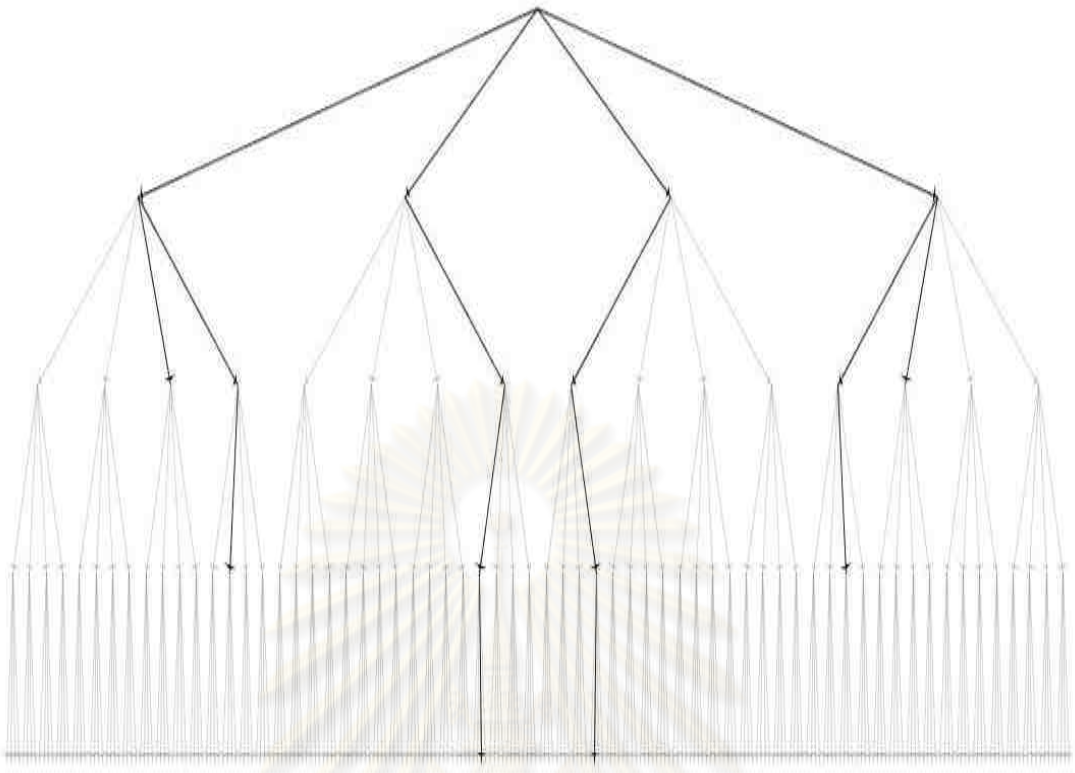
```
public class Node extends VState {
    int []col;
    Node parent;
    public  Node(int []col, Node parent) {
        this.col = col;
        this.parent=parent;
        VState.fireEvent(this);
    }
    @Override
    public void drawState(Graphics2D g2d, int x, int y) {
        g2d.setColor(Color.black); int x_pos = x; int y_pos = y;
        for (int i = 0; i < col.length; i++) {
            for (int j = 0; j < col.length; j++) {
                if (j % col.length == 0) {
                    x_pos = x - 40; y_pos += 29;
                } else x_pos += 27;
                if (col[i] == j)
                    g2d.draw3DRect(x_pos, y_pos - 40, 25, 25, true);
                else
                    g2d.fillRect(x_pos, y_pos - 40, 25, 25);
            }
        }
    }
    @Override
    public VState getParent(){
        return parent;
    }
}
```

Override เมท็อด `drawState` วาดรูปปมสถานะ

รหัสที่ 4.12 การเพิ่มเมท็อดวาดรูปบนคลาสสถานะ

4.7 วิธีแสดงการแจงปมสถานะที่พบจริงบนภาพปริภูมิสถานะ

การค้นปริภูมิสถานะด้วยอัลกอริทึมค้นหา อย่างเช่น การค้นแนวกว้าง หรือ การค้นแนวลึกเรียกซ้ำ มักมีการเพิ่มกลวิธีเข้าช่วยเล็มปมที่ไม่มีแววของคำตอบทิ้งออกไป เพื่อให้ผู้ใช้เข้าใจผลของการเล็มปมระหว่างการค้นที่สามารถหลีกเลี่ยงการค้นปมที่ไม่จำเป็นออกได้เป็นจำนวนมาก (หรือน้อย) ระบบจึงมีความสามารถในการแสดงภาพปมสถานะที่พบจริงบนภาพปริภูมิสถานะได้ ดังรูปที่ 4.4 เส้นเชื่อมปมสีเข้มแสดงให้เห็นปมสถานะที่พบจริงจากการเสริมกลวิธีย้อนรอย ซึ่งแสดงบนภาพปริภูมิสถานะทั้งหมดของการค้นปัญหา 4-Queen



รูปที่ 4.4 การค้นแบบแนวลึกและการเพิ่มกลวิธีย้อนรอย

การนำเสนอวิธีแสดงภาพปมสถานะที่พบจริงบนปริภูมิสถานะแบบเต็มดังเช่นรูปที่ 4.4 นั้นทำได้ด้วยการที่ผู้เขียนโปรแกรมปริภูมิสถานะรู้ว่าปมใดคือผลเฉลยปมสถานะในปริภูมิ ปมใดคือปมสถานะที่ไม่ถูกเล็มทิ้งออกไปจากการเสริมด้วยกลวิธีค้นคำตอบ

4.7.1 การแสดงปมสถานะที่พบจริงบนปริภูมิสถานะสำหรับเมท็อดเรียกซ้ำ

วิธีแสดงภาพปมสถานะที่พบจริงบนการแจงผลเฉลยปริภูมิสถานะบนโปรแกรมแบบเรียกซ้ำเมท็อดนั้นจะใช้วิธีการแทรกพารามิเตอร์บูลลีนไว้เป็นตัวสุดท้ายในเมท็อด จากนั้นเติมข้อกำหนดใน `@VMain` ด้วย `showAllStates`=ค่าบูลีน เพื่อให้ระบบแสดงภาพติดตามค่าพารามิเตอร์ทุกครั้งที่มีการเรียกเมท็อดเรียกซ้ำ หากเป็นค่า true แทนกรณีที่ปมสถานะนี้เป็นปมที่ถูกเล็มออกจากการค้นคำตอบ (ซึ่งระบบจะอนุญาตให้ปรับความเข้มของเส้นเชื่อมที่พุ่งเข้าหาปมแบบนี้) หรือกรณีที่ `false` แทนกรณีสถานะที่พบจริงบนปริภูมิสถานะ (ความเข้มของเส้นเชื่อมที่พุ่งเข้าหาปมแบบนี้ไม่เปลี่ยนแปลง)

```
1: import jstate101.*;
2: public class Queens{
3: public boolean isConsistent(int[] col, int n) {
4:         for (int i = 0; i < n; i++) {
5:             if(col[n] == col[i] ||
6:             Math.abs(col[n]-col[i]) == Math.abs(i - n)) {
7:                 return false;
8:             }
9:         }
10:    return true;
11:    }
12:    public void queensRCS(int[] col, int i, boolean pruned) {
13:      if (i == col.length) printQueens(col , col.length);
14:        else {
15:            for (int j = 0; j < col.length ; j++) {
16:                col[i] = j;
17:            queensRCS(col, i + 1, !isConsistent(col, i) || pruned);
18:            }
19:        }
20:    }
21:    @VMain(recursiveMethod = "queensRCS", showAllStates = true)
22:    public void mainDFS_RCS() {
23:        int[] col = new int[4];
24:    queensRCS(col, 0, false);
25:    }
26:public void printQueens(int[] col ,int n) {. . .}
27:    public static void main(String[] args) {
28:        VEngine.begin(new Queens());
29:    }
30:}
```

แทรกพารามิเตอร์บูลีน pruned ไว้ท้ายสุด

ข้อกำหนด showAllStates

รหัสที่ 4.13 การแก้ปัญหา 4-Queen แบบแนวลึกและเพิ่มกลวิธีย้อนรอย

รหัสที่ 4.13 เป็นการแก้ปัญหา 4-Queen ด้วยวิธีการเรียกซ้ำและเสริมกลวิธีย้อนรอย บรรทัดที่ 12 แทรกพารามิเตอร์บูลีน `pruned` ไว้ท้ายสุด จากนั้นเติมข้อกำหนด `showAllStates=true` (กรณี `showAllStates=false` แสดงเฉพาะปมสถานะที่พบจริงเท่านั้น ดังรูป 4.2) ลงใน `@VMain` (บรรทัดที่ 21) เพื่อให้ระบบแสดงภาพติดตามปมสถานะที่พบจริงบนปริภูมิสถานะ

4.7.2 การแสดงปมสถานะที่พบจริงบนปริภูมิสถานะสำหรับวงวนทำซ้ำ

สำหรับการเขียนโปรแกรมค้นปริภูมิด้วยวงวนทำซ้ำนั้น การนำเสนอวิธีแสดงภาพปมสถานะที่พบจริงบนปริภูมิสถานะสามารถทำได้ด้วยการแทรกคำสั่งลงบนคลาสสถานะที่เป็นคลาสลูกของ `VState` ไว้แล้ว โดยการ Override เมท็อดบูลีน จากนั้นเติมข้อกำหนดใน `@VMain` ด้วย `showAllStates`=ค่าบูลีน เพื่อให้ระบบแสดงภาพติดตามเมท็อดที่คืนค่าบูลีนทุกครั้งที่ปมสถานะถูกผลิต ดังรหัสที่ 4.14 Override เมท็อด `isPruned` ซึ่งเป็นเมท็อดคืนค่าบูลีนลงบนคลาส `Node` ที่เป็นคลาสลูกของ `VState` หากคืนค่า true แทนกรณีที่ปมสถานะนี้เป็นปมที่ถูกเล็มออกจากการค้นคำตอบ (ซึ่งระบบจะอนุญาตให้ปรับความเข้มของเส้นเชื่อมที่พุ่งเข้าหาปมแบบนี้) หรือกรณีที่คืนค่า `false` แทนกรณีสถานะที่พบจริงบนปริภูมิสถานะ (ความเข้มของเส้นเชื่อมที่พุ่งเข้าหาปมแบบนี้ไม่เปลี่ยนแปลง) จากนั้นเติมข้อกำหนด `showAllStates=true` (กรณี `showAllStates=false`

แสดงเฉพาะปมสถานะที่พบจริงเท่านั้น ดังรูป 4.2) ลงใน `@VMain` เพื่อให้ระบบแสดงภาพติดตามปมสถานะที่พบจริงบนปริภูมิสถานะ

```
import java.util.*;
import jstate101.*;
public class Queens{
    static class Node extends VState{
        int []col;  Node parent;
        boolean pruned;
        public Node(int []col, Node p ,boolean pruned){
            this.col = col;
            this.parent=p;
            this.pruned = pruned;
            VState.fireEvent(this);
        }
        @Override
        public boolean isPruned(){
            return pruned;
        }

        @Override
        public VState getParent(){
            return parent;
        }
    }
    @VMain(stateClass = "Node" ,showAllStates = true )
    public void mainDFS_ITR() {
        int[] col = new int[4];
        for (int i = 0; i < col.length; i++)
            col[i] = -1;
        queensITR(col, 0);
    }
    public void queensITR(int n) {
        Stack<Node> s = new Stack<Node>();
        Node root = new Node(new int[1] , null ,false);
        s.push(root);
        while (!s.isEmpty()) {
            root = s.pop();
            if (root.col.length - 1 == n)
                printQueens(root.col,n);
            else {
                for (int j = 0; j < n; j++){
                    int[] y = Arrays.copyOf(root.col, root.col.length + 1);
                    y[root.col.length - 1] = j;
                    Node node = new Node(y ,root ,
                                         !isConsistent(y, root.col.length - 1) ||
                                                 root.isPruned());
                    s.push(node);
                }
            }
        }
    }
    public void printQueens(int[] col) {. . .}
    public void isConsistent(int[] col) {. . .}
}
```

Override เมท็อดบูลีน `isPruned` เพิ่มเติม

ข้อกำหนด `showAllStates`

รหัสที่ 4.14 เพิ่มเมท็อดบูลีนสำหรับแสดงปมสถานะที่พบจริงบนปริภูมิสถานะ

4.8 **ส่วนติดต่อผู้ใช้บนเครื่องมือแสดงภาพต้นไม้ปริภูมิสถานะ**

การใช้งานเครื่องมือแสดงภาพต้นไม้ปริภูมิสถานะนั้นผู้ใช้เครื่องมือสามารถดูการเปลี่ยนแปลงปมสถานะย้อนหลังหรือเดินหน้าได้ผ่านส่วนติดต่อผู้ใช้ที่ได้ออกแบบไว้ โดยส่วนติดต่อผู้ใช้นั้นได้แบ่งการทำงานออกเป็นสามส่วนหลัก ๆ ดังนี้

4.8.1 **ส่วนปรับแต่งลักษณะภาพต้นไม้ปริภูมิสถานะ**

ส่วนนี้ทำหน้าที่ปรับแต่งลักษณะภาพต้นไม้ปริภูมิสถานะ ตัวนับ และแสดงค่ากำกับปมดังแสดงรูปที่ 4.5 ซึ่งมีรายละเอียดดังตารางที่ 4.1

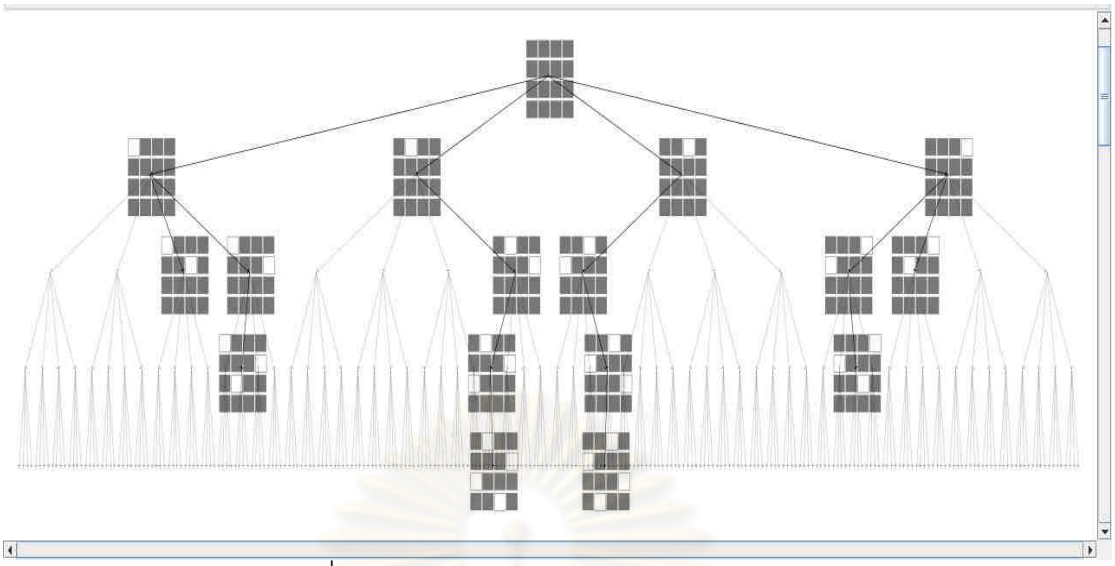
รูปที่ 4.5 ส่วนปรับแต่งลักษณะภาพต้นไม้ปริภูมิ

หมายเลข	คำอธิบาย
1	ปุ่ม capture รูป
2	ขยายสเกลภาพต้นไม้ปริภูมิแนวแกน X
3	ขยายสเกลภาพต้นไม้ปริภูมิแนวแกน Y
4	ตัวเลือกปรับลักษณะความเข้มเส้นเชื่อมปมสถานะ
5	แสดงค่าสตริงที่กำกับปมสถานะ
6	ตัวเลือกปรับลักษณะเส้นเชื่อมปมสถานะ
7	ตัวนับจำนวนปมต้นไม้ปริภูมิสถานะ
8	T ตัวย้ายตำแหน่งต้นไม้ปริภูมิสถานะ/P ปรับตำแหน่งปมสถานะ

ตารางที่ 4.1 อธิบายส่วนปรับแต่งลักษณะภาพต้นไม้ปริภูมิสถานะ

4.8.2 **ส่วนแสดงแผนภาพต้นไม้ปริภูมิสถานะ**

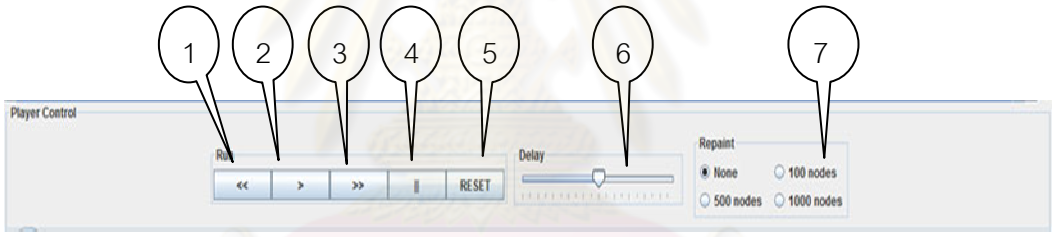
ส่วนนี้เป็นส่วนสำหรับแสดงภาพต้นไม้ปริภูมิสถานะซึ่งสามารถปรับขนาด ซูมเข้า-ออกด้วย เมาส์และเคลื่อนขยับภาพไปยังตำแหน่งใด ๆ บนพื้นที่แสดงภาพได้ดังรูปที่ 4.6



รูปที่ 4.6 ส่วนแสดงแผนภาพต้นไม้ปริภูมิสถานะ

4.8.3 **ส่วนควบคุมการนำเสนอภาพในลักษณะเครื่องเล่นวีดิทัศน์**

ส่วนนี้จะเป็นส่วนที่ผู้ใช้สามารถควบคุมการนำเสนอภาพปริภูมิสถานะในลักษณะเครื่องเล่น วีดิทัศน์ด้วยปุ่มควบคุมที่ประกอบด้วยส่วนต่าง ๆ ดังรูปที่ 4.7 ซึ่งมีรายละเอียดดังตารางที่ 4.2



รูปที่ 4.7 ส่วนควบคุมการนำเสนอภาพในลักษณะเครื่องเล่นวีดิทัศน์

ตารางที่ 4.2 อธิบายส่วนควบคุมการนำเสนอภาพในลักษณะเครื่องเล่นวีดิทัศน์

หมายเลข	คำอธิบาย
1	ปุ่มย้อนกลับหนึ่ง step
2	ปุ่มเล่นภาพ
3	ปุ่มเดินหน้ากลับหนึ่ง step
4	ปุ่มหยุดภาพชั่วขณะ
5	ปุ่มตั้งค่าการเล่นใหม่
6	สไลด์ปรับความเร็วการนำเสนอ
7	จำนวนปมที่ถูกวาดในหนึ่ง step

บทที่ 5

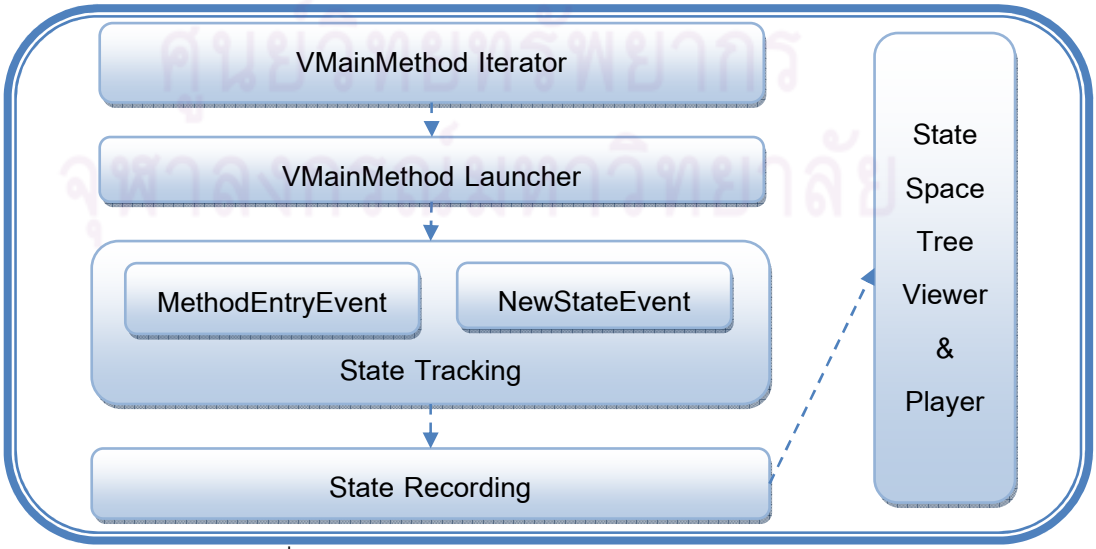
การออกแบบและพัฒนาระบบแสดงภาพปริภูมิสถานะ

งานวิจัยนี้ศึกษา ออกแบบ และพัฒนาระบบแสดงภาพปริภูมิสถานะซึ่งเรียกว่า “JState101” ที่ใช้ติดตามและบันทึกการเปลี่ยนแปลงสถานะต่าง ๆ ระหว่างการค้นคำตอบในปริภูมิสถานะ โดยแสดงผลในรูปของต้นไม้ปริภูมิสถานะที่ปรับเปลี่ยนตามสถานะที่เกิดขึ้น ผู้ใช้สามารถดูเหตุการณ์การเปลี่ยนแปลงสถานะย้อนหลังหรือเดินหน้าได้ ตัวระบบรองรับสถานะในปริภูมิที่กำหนดเป็นแบบอ็อบเจกต์สถานะจากการเขียนโปรแกรมค้นคำตอบด้วยวงวนทำซ้ำ หรือสถานะที่เก็บใน Stack trace จากการเขียนโปรแกรมแบบเรียกซ้ำ

เนื้อหาในบทนี้จะประกอบไปด้วย ภาพรวมของระบบ การอ่านค่าตัวกำกับการแสดงภาพ การสั่งเมท็อดที่กำกับด้วย `@VMain` ทำงาน การติดตามปมสถานะเกิดใหม่ การบันทึกสถานะ และการแสดงภาพต้นไม้ปริภูมิสถานะ

5.1 ภาพรวมของระบบ

การแสดงภาพต้นไม้ปริภูมิสถานะเริ่มต้นจากโปรแกรมค้นคำตอบปริภูมิถูกแทรกด้วยตัวกำกับการแสดงภาพ `@VMain` หรือรหัสเพิ่มเติมที่ระบบแสดงภาพ JState101 กำหนด เมื่อสั่งประมวลผลโปรแกรมค้นปริภูมิ ระบบแสดงภาพนี้จะกลับมาอ่านและเก็บข้อกำหนดการแสดงภาพรวมทั้งข้อมูลอื่น ๆ ที่เกี่ยวข้อง จากนั้นจะมีการสร้างหน่วยประมวลผลย่อย เพื่อเรียกเมท็อดที่กำกับด้วย `@VMain` ทำงาน ระบบแสดงภาพนี้จะติดตามสถานะที่เกิดใหม่เพื่อเก็บบันทึกในระบบและแสดงภาพสถานะเหล่านี้ในส่วนติดต่อผู้ใช้ในลักษณะต้นไม้ปริภูมิสถานะ โดยขั้นตอนการทำงานระบบแสดงภาพ JState101 แบ่งออกเป็น 5 ส่วน (ดังรูปที่ 5.1) คือ



รูปที่ 5.1 โครงสร้างของระบบแสดงภาพ JState101

1 การอ่านข้อกำหนดการทำงาน เป็นขั้นตอนอ่านตัวกำกับการแสดงภาพ `@VMain` จากนั้นเก็บข้อกำหนดรวมทั้งข้อมูลอื่น ๆ ที่เกี่ยวข้อง ซึ่งขั้นตอนนี้เรียกว่า “VMainMethod Iterator”

2 การเรียกเมท็อดที่กำกับ @VMain ทำงาน เป็นขั้นตอนที่ทำงานภายใต้หน่วยประมวลผลย่อย ทำหน้าที่สั่งเมท็อดที่มีตัวกำกับแสดงภาพ `@VMain` ประกาศไว้ทำงาน ซึ่งขั้นตอนนี้เรียกว่า “VMainMethod Launcher”

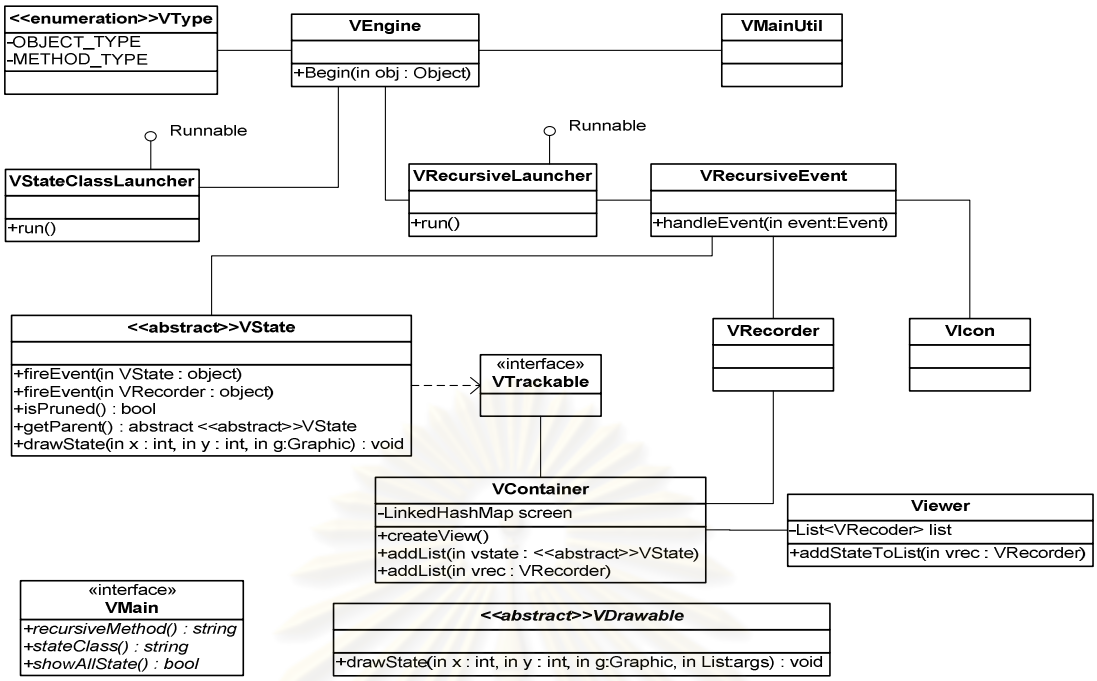
3 การติดตามสถานะเกิดใหม่ เป็นขั้นตอนติดตามการผลิตสถานะและความสัมพันธ์ของสถานะที่เกิดใหม่จากโปรแกรมค้นปริภูมิ ขั้นตอนนี้เรียกว่า “State Tracking” โดยการติดตามสถานะแบ่งเป็นสองแบบคือ การติดตามสถานะเมื่อมีการเรียกเมท็อดเรียกซ้ำ “MethodEntryEvent” และการติดตามสถานะที่เกิดจากสร้างอ็อบเจกต์ “NewStateEvent”

4 การบันทึกสถานะ เป็นขั้นตอนการจัดเก็บสถานะต่าง ๆ ที่ได้จากการติดตามสถานะ ขั้นตอนนี้เรียกว่า “State Recording”

5 การแสดงภาพต้นไม้ปริภูมิสถานะ เป็นขั้นตอนการสร้างส่วนติดต่อผู้ใช้และนำเสนอภาพต้นไม้ปริภูมิสถานะ ขั้นตอนนี้เรียกว่า “State Space Tree Viewer & Player”

5.2 คลาสต่าง ๆ ของระบบแสดงภาพปริภูมิสถานะ

การออกแบบคลาสต่าง ๆ สำหรับการแสดงนั้นถูกรวบรวมไว้ใน `Package jstate101.*` ซึ่งมีหน้าที่สำคัญในการประกาศตัวกำกับการแสดงภาพ `@VMain` การอ่านข้อกำหนดการทำงาน การสั่งเมท็อดดำเนินงานทำงาน การติดตามสถานะเกิดใหม่ การบันทึกสถานะ การแสดงภาพและควบคุมภาพต้นไม้ปริภูมิสถานะผ่านส่วนติดต่อผู้ใช้ โดยสามารถแสดงแผนภาพคลาสได้ดังรูปที่ 5.2



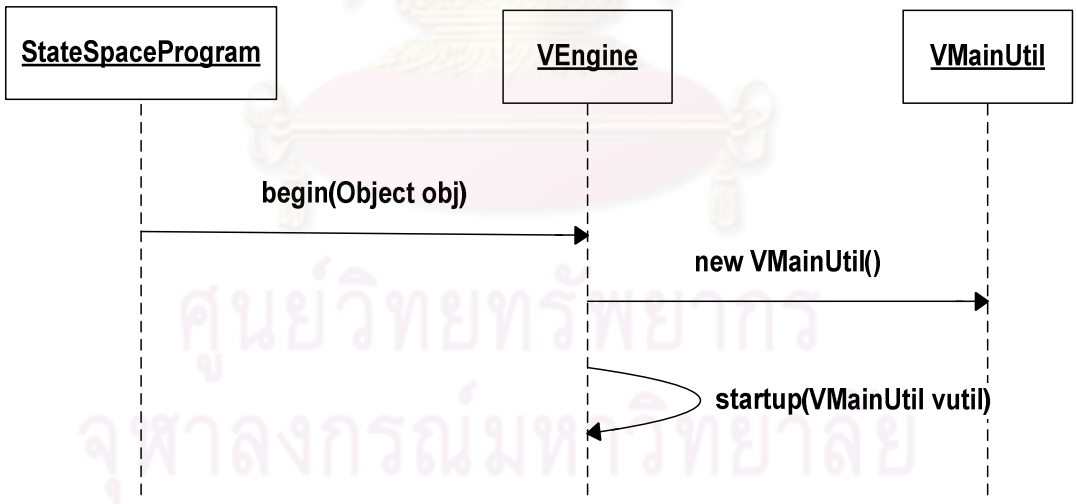
รูปที่ 5.2 คลาสต่าง ๆ ของระบบแสดงภาพปริภูมิสถานะ

- `VMain` เป็น annotation สำหรับใช้ประกาศค่ากำกับแสดงภาพบนเมท็อดดำเนินงานในโปรแกรมค้นปริภูมิ
- `VEngine` เป็นคลาสใช้สำหรับการอ่านค่ากำกับแสดงภาพ และเรียกคลาสสั่งโปรแกรม ค้นปริภูมิทำงาน
- `VMainUtil` เป็นคลาสเก็บข้อกำหนดการแสดงภาพที่ได้จากการอ่านข้อกำหนดใน `@VMain` รวมทั้งข้อกำหนดอื่น ๆ ที่เกี่ยวข้อง
- `VType` เป็น enum ระบุรูปแบบของโปรแกรมค้นปริภูมิว่าเป็นอ็อบเจกต์สถานะจากการเขียนโปรแกรมแบบวงวนทำซ้ำ หรือแบบเมท็อดเรียกซ้ำ
- `VStateClassLauncher` เป็นคลาสที่ทำงานภายใต้หน่วยประมวลผลย่อยสำหรับเรียก เมท็อดที่ถูกกำกับด้วย `@VMain` ทำงาน(ค้นแบบวงวนทำซ้ำ)
- `VRecursiveLauncher` เป็นคลาสทำงานภายใต้หน่วยประมวลผลย่อยสำหรับเรียกเมท็อดที่ถูกกำกับด้วย `@VMain` ทำงาน(ค้นแบบเมท็อดเรียกซ้ำ)
- `VRecursiveEvent` เป็นคลาสติดตามการเรียกซ้ำเมท็อด
- `VRecorder` เป็นคลาสสำหรับเก็บสถานะต่าง ๆ ที่ได้จากการติดตามสถานะ
- `VDrawable` เป็นคลาส `abstract` ใช้ในกรณีที่ต้องวาดรูปกำกับปมสถานะโดยขยายคลาสนี้บนการเขียนโปรแกรมค้นคำตอบแบบเรียกซ้ำ
- `VICon` เป็นคลาสเรียกเมท็อดวาดรูปกำกับปมสถานะในโปรแกรมแบบเรียกซ้ำ

- `VState` เป็นคลาส `abstract` มีอินเทอร์เฟสติดตามสถานะเข้าสู่ส่วนบันทึก และมีเมท็อดบังคับคืนค่าปมพ่อแม่ เมท็อดวาดรูปกำกับปม และเมท็อดคืนค่าบูลีนสำหรับกำหนดความเข้มเส้นเชื่อมสถานะ บนการเขียนโปรแกรมค้นคำตอบด้วยวงวนทำซ้ำ
- `VTrackable` เป็นอินเทอร์เฟสติดตามสถานะ
- `VContainer` เป็นคลาสบันทึกส่วนแสดงภาพปริภูมิสถานะ
- `Viewer` เป็นคลาสสำหรับแสดงภาพปริภูมิสถานะรวมทั้งส่วนติดต่อผู้ใช้เพื่อควบคุมการนำเสนอภาพต้นไม้ปริภูมิสถานะ

5.3 การอ่านค่าตัวกำกับแสดงภาพ

เมื่อโปรแกรมค้นปริภูมิเริ่มสั่งทำงาน อ็อบเจกต์โปรแกรมค้นปริภูมิจะถูกส่งผ่าน `VEngine.begin(Object obj)` โดยอ็อบเจกต์นี้มีตัวกำกับแสดงภาพ `@VMain` บนเมท็อดดำเนินงานติดมาด้วย ระบบแสดงภาพจะมาอ่านและเก็บข้อกำหนดที่ได้ประกาศไว้ รวมทั้งเก็บข้อกำหนดอื่น ๆ ที่เกี่ยวข้องไว้ในคลาส `VMainUtil` เพื่อเป็นเงื่อนไขในการสั่งระบบแสดงภาพทำงาน ดังแสดงรูปที่ 5.3 ซึ่งหลักการการทำงานส่วนนี้เรียกว่า “VMainMethod Iterator”



รูปที่ 5.3 การอ่านค่ากำกับ `@Vmain` และลงใน `VMainUtil`

รหัสที่ 5.1 อ็อบเจกต์โปรแกรมค้นปริภูมิถูกส่งมาที่เมท็อด `begin` ให้ระบบแสดงภาพ JState101 อ่านข้อกำหนดที่ประกาศไว้ใน `@VMain` รวมทั้งเก็บข้อกำหนดอื่น ๆ ที่เกี่ยวข้องไว้ในคลาส `VMainUtil` เพื่อส่งต่อไปยังเมท็อด `startup` ใช้สร้างประมวลผลย่อยสั่งเมท็อดที่กำกับ `@VMain` ทำงาน โดยขั้นตอนอ่านข้อกำหนด `@VMain` ในรหัสที่ 5.1 มีรายละเอียดการทำงานดังนี้

```
1:  public static void begin(Object obj) {
2:      for(Method method : obj.getClass().getDeclaredMethods()){
3:          if(method.isAnnotationPresent(VMain.class)){
4:              VMain vmain = m.getAnnotation(VMain.class);
5:              VMainUtil vutil = new VMainUtil();
6:          vutil.setClassName(obj.getClass().getName());
7:          vutil.setVmainMethodName(method.getName());
8:              if(isRecursive(vmain)){
9:              String filepath = getPathReader(obj);
10:                 vutil.setDebuggingPathName(filepath);
11:             vutil.setMethodRecursiveName(vmain.recursiveMethod());
12:             vutil.setPrunedName(vmain.showAllStates());
13:             vutil.setType(VType.METHOD_TYPE);
14:         }else if(isstateClass(vmain)){
15:             vutil.setStateClassName(vmain.stateClass());
16:             vutil.setPrunedName(vmain.showAllStates());
17:             vutil.setType(VType.OBJECT_TYPE);
18:         }
19:         startup(vutil);
20:     }
21: }
22: }
```

รหัสที่ 5.1 การอ่านตัวกำกับแสดงภาพ `@VMain`

5.3.2 การเก็บข้อกำหนดการทำงาน

เมท็อดที่กำกับด้วย `@VMain` มีการระบุข้อกำหนดลักษณะการเขียนโปรแกรมค้นปริภูมิ (แบบวงวนทำซ้ำ หรือ แบบเรียกซ้ำ) ซึ่งข้อกำหนดที่อ่านได้นั้นจะถูกเก็บในตัวเก็บข้อกำหนดที่มีชื่อว่า `VMainUtil` (รหัสที่ 5.1 บรรทัดที่ 5) โดยวิธีการเก็บลักษณะข้อกำหนดแบ่งเป็นสองแบบดังนี้

5.3.2.1 การเก็บข้อกำหนดแบบเรียกซ้ำ

ข้อกำหนดกำกับการแสดงภาพ `@VMain` บนเมท็อดแบบเมท็อดเรียกซ้ำนั้น บ่งบอกว่าสถานะเกิดใหม่ถูกสร้างขึ้นด้วยการเรียกซ้ำเมท็อดตัวมันเอง โดยรายละเอียดของสถานะใหม่อ่านได้จาก stack trace ตัวอย่างการเขียน `@VMain` กำกับเมท็อด เช่น

```
@VMain(recursiveMethod = “queensRCS” , showAllStates = true )
```

กำหนด `recursiveMethod` มีเมท็อดเรียกซ้ำชื่อ “`queensRCS`” ที่ระบบต้องติดตามทุกครั้งที่มีการเรียกเมท็อดดังกล่าว รวมทั้งติดตามค่าพารามิเตอร์ `showAllStates` ที่ระบุในข้อกำหนด `showAllStates=true` เพื่อติดตามเส้นเชื่อมที่พุ่งเข้าหาปมสถานะในกรณีที่ต้องการแสดงภาพปมสถานะที่พบจริงบนปริภูมิสถานะ โดยตัวเก็บข้อกำหนด `VMainUtil` จะทำหน้าที่รองรับข้อกำหนดเหล่านี้รวมทั้งข้อกำหนดอื่น ๆ ที่ต้องการซึ่งรายละเอียดมีดังนี้

1 ชื่อคลาสค้นคำตอบปริภูมิสถานะ เป็นข้อกำหนดที่ระบบแสดงภาพนี้ใช้รีเฟคชันเรียกคลาสค้นปริภูมิสำหรับสั่งเมท็อดกำกับ `@VMain` ทำงาน (รหัสที่ 5.1 บรรทัดที่ 6)

2 ชื่อเมท็อดที่กำกับด้วย `@VMain` เป็นข้อกำหนดที่ระบบแสดงภาพนี้ใช้เรียกเมท็อดที่กำกับ `@VMain` ทำงาน (รหัสที่ 5.1 บรรทัดที่ 7)

3 ชื่อเรียกแฟ้มของปริภูมิ เป็นข้อกำหนดสำหรับระบุเส้นทางเรียกแฟ้มต้นฉบับโปรแกรมค้นปริภูมิที่ใช้สำหรับกระบวนการติดตามการเรียกเมท็อด โดยเมท็อด `getPathReader` (บรรทัดที่ 9 รหัสที่ 5.1) อ่านและคืนตำแหน่งของแฟ้มโปรแกรมค้นปริภูมิ จากนั้นเก็บข้อกำหนดในเมท็อด `setDebuggingPathName` (บรรทัดที่ 10 รหัสที่ 5.1)

4 ชื่อเมท็อดเรียกซ้ำ เป็นข้อกำหนดที่ระบบแสดงภาพต้องติดตามทุกครั้งที่เมท็อดนี้ถูกเรียก (รหัสที่ 5.1 บรรทัดที่ 11)

5 รูปแบบการแสดงต้นไม้ปริภูมิ ว่าจะแสดงเฉพาะปมสถานะที่ค้นพบจริง หรือแสดงพร้อมกับต้นไม้ปริภูมิทั้งต้น เพื่อใช้ปรับความเข้มเส้นเชื่อมที่พุ่งเข้าหาปมสถานะบนส่วนติดต่อผู้ใช้ (รหัสที่ 5.1 บรรทัดที่ 12)

6 รูปแบบการค้นคำตอบ เป็นข้อกำหนดที่ระบบแสดงภาพใช้สำหรับตรวจสอบเงื่อนไขการสั่งการทำงานหน่วยประมวลผลย่อยทำงาน ซึ่งกำหนดเป็น `VType.METHOD_TYPE` (รหัสที่ 5.1 บรรทัดที่ 13) หมายถึงโปรแกรมค้นปริภูมินั้นถูกสั่งทำงานผ่าน JDI

5.3.2.2 การเก็บข้อกำหนดแบบวงวนทำซ้ำ

ข้อกำหนดแบบวงวนทำซ้ำ บ่งบอกถึงสถานะที่เกิดใหม่ถูกสร้างขึ้นด้วยวิธีการเขียนโปรแกรมค้นปริภูมิแบบวงวนทำซ้ำ ซึ่งสถานะประเภทนี้ถูกสร้างจากคลาสสถานะ ตัวอย่างการเขียน `@VMain` กำกับบนเมท็อด เช่น

```
@VMain(StateClass = “Node” , showAllStates = true)
```

บ่งบอกว่าตัวกำกับ `@VMain` กำหนดให้คลาสสถานะชื่อ "Node" เป็นคลาสที่ระบบแสดงภาพต้องติดตามทุกครั้งที่อ็อบเจ็กต์ของคลาสนี้ถูกสร้าง รวมทั้งติดตามค่าพารามิเตอร์ `showAllStates` ที่ระบุในข้อกำหนด `showAllStates=true` เพื่อติดตามเส้นเชื่อมที่พุ่งเข้าหาปมสถานะในกรณีที่ต้องการแสดงภาพปมสถานะที่พบจริงบนปริภูมิสถานะ โดยตัวเก็บข้อกำหนด `VMainUtil` จะทำหน้าที่รองรับข้อกำหนดเหล่านี้รวมทั้งข้อกำหนดอื่นๆที่ระบบแสดงภาพ JState101 ต้องการดังนี้

1 ชื่อคลาสค้นคำตอบปริภูมิสถานะ เป็นข้อกำหนดที่ระบบแสดงภาพนี้ใช้รีเฟคชันเรียกคลาสค้นปริภูมิสำหรับสั่งเมท็อดกำกับ `@VMain` ทำงาน (รหัสที่ 5.1 บรรทัดที่ 6)

2 ชื่อเมท็อดที่กำกับด้วย `@VMain` เป็นข้อกำหนดที่ระบบแสดงภาพนี้ใช้เรียกเมท็อดที่กำกับ `@VMain` ทำงาน (รหัสที่ 5.1 บรรทัดที่ 7)

3 ชื่อคลาสสถานะ เป็นข้อกำหนดที่ระบบแสดงภาพนี้ใช้ตรวจสอบค่าเรียกคืนต่าง ๆ ภายในคลาสสถานะ (รหัสที่ 5.1 บรรทัดที่ 15)

4 รูปแบบการแสดงต้นไม้ปริภูมิ ว่าจะแสดงเฉพาะปมสถานะที่ค้นพบจริง หรือแสดงพร้อมกับต้นไม้ปริภูมิทั้งต้น เพื่อใช้ปรับความเข้มเส้นเชื่อมที่พุ่งเข้าหาปมสถานะบนส่วนติดต่อผู้ใช้ (รหัสที่ 5.1 บรรทัดที่ 16)

5 รูปแบบกระบวนการค้นคำตอบ เป็นข้อกำหนดที่ระบบแสดงภาพนี้ใช้สำหรับตรวจสอบเงื่อนไขการสั่งการทำงานหน่วยประมวลผลย่อยทำงานสำหรับโปรแกรมเขียนแบบวงวนทำซ้ำ ซึ่งกำหนดเป็น `VType.OBJECT_TYPE` (รหัสที่ 5.1 บรรทัดที่ 17)

5.3.3 การสั่งโปรแกรมค้นปริภูมิทำงานในหน่วยประมวลผลย่อย

ข้อกำหนดต่าง ๆ เมื่อถูกอ่านและเก็บไว้ในตัวเก็บข้อกำหนด `VMainUtil` แล้ว จะถูกส่งไปยังเมท็อด `startup` เพื่อสั่งโปรแกรมค้นปริภูมิทำงานในหน่วยประมวลผลย่อย (รหัสที่ 5.1 บรรทัดที่ 19) ในแต่ละวงวนการอ่านข้อกำหนดกำกับการแสดงภาพ

5.4 การเรียกเมท็อดที่กำกับ @VMain ทำงาน

JState101 สั่งโปรแกรมค้นปริภูมิทำงานภายใต้หน่วยประมวลผลย่อย ด้วยการสร้าง Thread ดังแสดงในรหัสที่ 5.2 (เรียกการทำงานส่วนนี้ว่า"VMainMethod Launcher")

```
1: public static void startup(VMainUtil vutil){
2:  Thread thread;
3:  switch (vutil.getType()){
4:      case METHOD_TYPE:
5:                      thread = new Thread(new
6:                              ThreadGroup(vutil.getVmainMethodName()),
7:                              new VRecursiveLauncher(vutil));
8:                      thread.start();
9:                      break;
10:     case OBJECT_TYPE:
11:                     thread = new Thread(new
12:                             ThreadGroup(vutil.getVmainMethodName()),
13:                             new VStateClassLauncher(vutil));
14:                     thread.start();
15:                     break;
16:     default:        System.out.println("Search type is not found.");
17:                     break;
18:     }
19: }
```

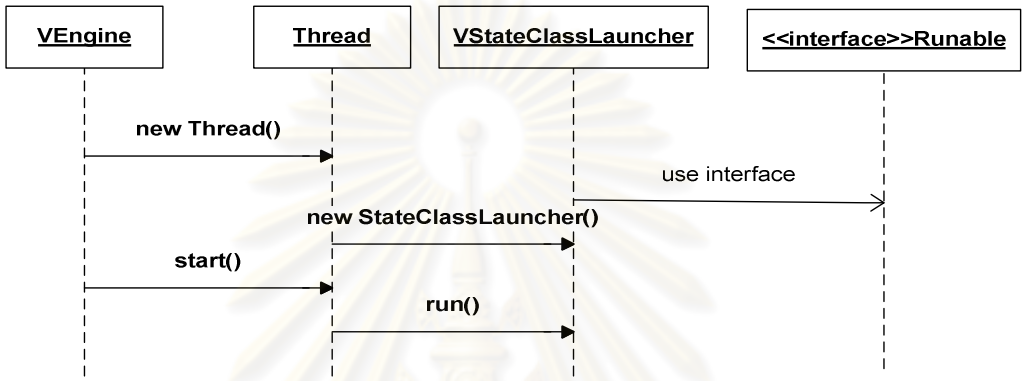
รหัสที่ 5.2 การสร้างหน่วยประมวลผลย่อยสั่งเมท็อดดำเนินงาน

รหัสที่ 5.2 ข้อกำหนดต่าง ๆ ที่ถูกอ่านและเก็บไว้ใน `VMainUtil` จะถูกส่งมายังเมท็อด `startup` จากนั้นถ้าเข้าเงื่อนไขการค้นคำตอบแบบเรียกซ้ำ (บรรทัดที่ 4) ระบบแสดงภาพสร้างหน่วยประมวลผลย่อยด้วย `Thread` เรียกคลาส `VRecursiveLauncher` (บรรทัดที่ 5-7) เพื่อเรียกการทำงานเมท็อดเรียกซ้ำ ด้วยการสั่ง `start` (บรรทัดที่ 8) หรือเงื่อนไขการค้นคำตอบแบบวงวนทำซ้ำ (บรรทัดที่ 10) ระบบแสดงภาพสร้างหน่วยประมวลผลย่อยด้วย `Thread` สร้างคลาส

VStateClassLauncher (บรรทัดที่ 11-13) เพื่อเรียกเมท็อดที่กำกับด้วย @VMain ทำงานด้วยการสั่ง start (บรรทัดที่ 14)

5.4.1 การเรียกโปรแกรมค้นปริภูมิสถานะแบบวงวนทำซ้ำ

คลาส VStateClassLauncher มีอินเทอร์เฟส Runnable ซึ่งทำงานภายใต้หน่วยประมวลผลย่อย Thread ทำหน้าที่เป็นตัวเรียกเมท็อดดำเนินงานสั่งค้นคำตอบด้วยวงวนทำซ้ำดังแสดงรูปที่ 5.4



รูปที่ 5.4 การสั่งเมท็อดดำเนินงานในโปรแกรมแบบวงวนทำซ้ำทำงาน

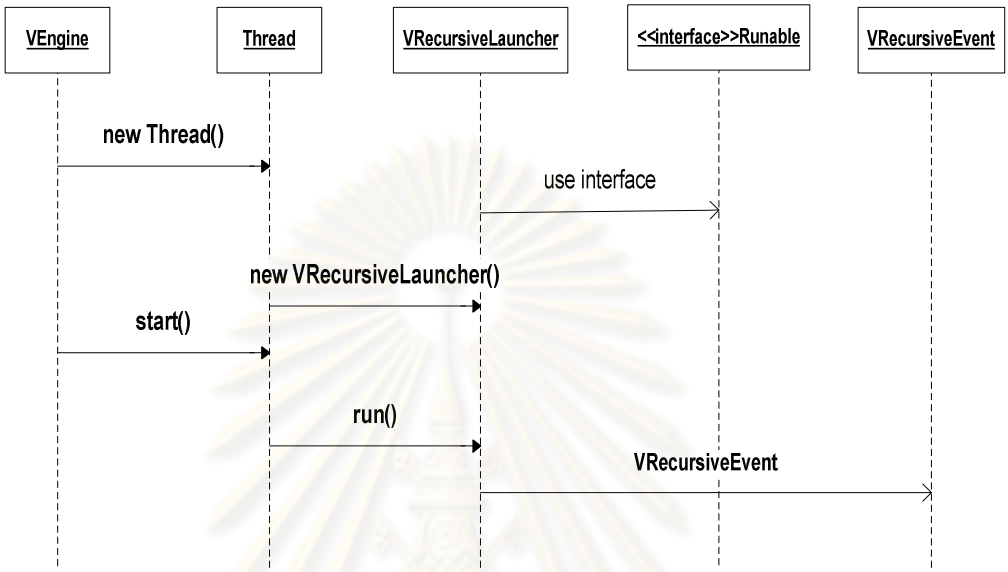
รหัสที่ 5.3 คลาส VStateClassLauncher รับตัวเก็บข้อกำหนด VMainUtil (บรรทัดที่ 3) เมื่อหน่วยประมวลผลย่อยถูกสั่งด้วย start ภายใต้เมท็อด run สร้างคลาส Class (บรรทัดที่ 8) และสร้างอ็อบเจกต์ของโปรแกรมค้นปริภูมิขึ้นมาใหม่ (บรรทัดที่ 9) จากนั้นเรียกเมท็อดที่กำกับ @VMain ทำงาน (บรรทัดที่ 10-12)

```
1:  public class VStateClassLauncher implements Runnable {
2:      VMainUtil vutil;
3:      public VStateClassLauncher(VMainUtil vutil){
4:          this.vutil = vutil;
5:      }
6:      public void run() {
7:          try{
8:              Class c = Class.forName(vutil.getClassName())
9:              Object obj = c.newInstance();
10:             Method mt = obj.getClass()
11:                     .getMethod(vutil.getVmainMethodName());
12:             mt.invoke(obj);
13:         } catch (Exception ex) {. . .}
14:     }
15: }
```

รหัสที่ 5.3 การสร้างหน่วยประมวลผลย่อยโปรแกรมค้นปริภูมิ

5.4.2 การเรียกโปรแกรมคำตอบปริภูมิสถานะแบบเรียกซ้ำ

คลาส `VRecursiveLauncher` มีอินเทอร์เฟส `Runnable` ซึ่งทำงานภายใต้หน่วยประมวลผลย่อย `Thread` ทำหน้าที่เรียกเมท็อดซึ่งถูกกำกับด้วย `@Vmain` ทำงาน โดยการทำงานแสดงดังรูปที่ 5.5



รูปที่ 5.5 การสั่งเมท็อดดำเนินงานในโปรแกรมแบบเรียกซ้ำทำงาน

รหัสที่ 5.4 คลาส `VRecursiveLauncher` รับค่าตัวเก็บข้อกำหนด `VMainUtil` (บรรทัดที่ 3-4) เมื่อหน่วยประมวลผลย่อยถูกสั่งด้วย `start` กระบวนการติดต่อ `VirtualMachine` เริ่มทำงานภายใต้เมท็อด `run` เริ่มสร้างตัวติดต่อไปยัง `VirtualMachine` (บรรทัดที่ 7) และเตรียมคำสั่งเรียกโปรแกรมค้นปริภูมิ (บรรทัดที่ 8) จากนั้น launch คำสั่ง (บรรทัดที่ 9) และสั่งดูเหตุการณ์ในปริภูมิสถานะ (บรรทัดที่ 10) โดยแต่ละขั้นตอนมีรายละเอียดดังนี้

```
1. public class VRecursiveLauncher implements Runnable {
2.     private VMainUtil vutil;
3.     public RecursiveLauncher(VMainUtil vutil){
4.         this.vutil= vutil;
5.     }
6.     public void run(){
7.         LaunchingConnector connector = findLaunchingConnector();
8.         Map arguments = connectorArguments(connector);
9.         VirtualMachine vm =LaunchTarget(connector,arguments);
10.         running(vm);
11.      }
12.      private VirtualMachine LaunchTarget(Map arguments){. . .}
13.      private LaunchingConnector findLaunchingConnector() {. . .}
14.      private Map connectorArguments(){. . .}
15.      private void running(VirtualMachine vm){. . .}
16. }
```

รหัสที่ 5.4 การสร้างหน่วยประมวลผลย่อยโปรแกรมค้นปริภูมิ

5.4.2.1 การสร้างตัวติดต่อไปยังจาวาเวอร์ชวลแมชชีน

ระบบแสดงภาพปริภูมิสถานะอาศัย JDI ติดต่อจาวาเวอร์ชวลแมชชีนเพื่อการติดตามปมสถานะที่ถูกผลิตในโปรแกรมค้นปริภูมิแบบเรียกซ้ำเมท็อด ทำให้ไม่ต้องแทรกรหัสเพิ่มเติมลงในโปรแกรมค้นปริภูมิ ระบบก็สามารถทราบความสัมพันธ์ของปมสถานะที่ถูกผลิตได้อัตโนมัติ

การสร้างตัวติดต่อไปยังจาวาเวอร์ชวลแมชชีนเพื่อเรียกคลาส `VRecursiveRunner` ทำงานจะติดต่อผ่านคลาส `com.sun.jdi.CommandLineLaunch` ดังแสดงรหัสที่ 5.5 เรียกตัวติดต่อจาวา-เวอร์ชวลแมชชีน (บรรทัดที่ 2) จากนั้นเข้าสู่วงวนค้นตัวติดต่อ (บรรทัดที่ 4-9) และคืนค่าตัวติดต่อกลับ (บรรทัดที่ 7)

```
1: private LaunchingConnector findLaunchingConnector(){
2: List<Connector> connectors =
       Bootstrap.virtualMachineManager().allConnectors();
3:     Iterator<Connector> iter = connectors.iterator();
4:     while (iter.hasNext()) {
5:         Connector connector = iter.next();
6:         if (connector.name().equals("com.sun.jdi.CommandLineLaunch")) {
7:             return (LaunchingConnector) connector;
8:         }
9:     }
10:    throw new Error("No launching connector");
11:}
```

รหัสที่ 5.5 การเรียกตัวติดต่อแบบ command line

5.4.2.2 การส่งคำสั่งเรียกคลาส `VRecursiveRunner`

การเขียนโปรแกรมค้นปริภูมิแบบเรียกซ้ำนั้น เมท็อดที่กำกับ `@Vmain` จะถูกเรียกผ่านคลาส `VRecursiveRunner` โดยที่คลาสนี้จะถูกสั่งผ่าน JDI ด้วยคำสั่งในลักษณะเดียวกับการเรียกประมวลผล จาวาผ่านคำสั่งแบบ Command line เช่น

```
cmd>java VRecursiveRunner args0 args1
```

ดังรหัสที่ 5.6 เมท็อด `connectorArguments` (บรรทัดที่ 1) รับตัวติดต่อจาวาเวอร์ชวล-แมชชีน จากนั้นอ่านค่าเส้นทางเรียกไฟล์ `VRecursiveRunner.class` ผ่านคลาส `Class` (บรรทัดที่ 2-4) และอากิวเมนต์ที่ใช้สำหรับเรียกเมท็อดกำกับ `@Vmain` โดยบรรทัดที่ 5 เป็นอากิวเมนต์ชื่อเรียกแฟ้มต้นฉบับของปริภูมิ บรรทัดที่ 6 เป็นอากิวเมนต์ชื่อคลาสค้นปริภูมิ บรรทัดที่ 7 เป็นอากิว-เมนต์ชื่อเมท็อดกำกับ `@VMain` และเตรียมคำสั่ง `commandline` (บรรทัดที่ 8-11) จากนั้นคืนค่ากลับคำสั่งเรียกโปรแกรม (บรรทัดที่ 12)

```
1: private Map connectorArguments(LaunchingConnector connector){
2:     Class launcher = VRecursiveRunner.class;
3:     String commandPath = launcher.getClassLoader()
4:    .getResource("jstate101/recursive/VRecursiveRunner.class").getPath();
5:     String args0 = vutil.getDebuggingPathName();
6:     String args1 = vutil.getClassName();
7:     String args2 = vutil.getVmainMethodName();
8:     String commandline = commandPath +" "+args0+" "+args1+" "+args2;
9:     Map<String,Argument> arguments = connector.defaultArguments();
10:    Connector.Argument mainArg = arguments.get("main");
11:    arguments = mainArg.setValue(commandline);
12:    return arguments;
13:}
```

รหัสที่ 5.6 การเตรียมคำสั่งเรียกโปรแกรมค้นปริภูมิ

5.4.2.3 ตัวสั่งเมท็อดกำกับ @VMain ทำงาน

ตัวสั่งเมท็อดกำกับ `@VMain` ทำงานเป็นโปรแกรมที่ระบบแสดงภาพใช้สำหรับเรียกเมท็อดกำกับ `@VMain` ทำงาน โปรแกรมนี้ถูกเรียกใช้งานผ่าน JDI รหัสที่ 5.7 คลาส `VRecursiveRunner` เป็นตัวสั่งเมท็อดที่กำกับ `@VMain` ทำงาน โดยคลาสนี้มีเมท็อด `main` ซึ่งโหลดคลาสโปรแกรมค้นปริภูมิสู่ระบบ (บรรทัดที่4-7) จากนั้นเรียกคลาส `class` สร้างอ็อบเจกต์โปรแกรมค้นปริภูมิ (บรรทัดที่ 8-9) เพื่อเรียกเมท็อดดำเนินที่ถูกกำกับด้วย `@VMain`(บรรทัดที่ 10-11)

```
1: public class VRecursiveRunner{
2:     public static void main(String[] args) {
3:         try {
4:             File file = new File(args[0]);
5:             URL url = file.toURL();
6:             URL[] urls = new URL[]{url};
7:             ClassLoader loader = new URLClassLoader(urls);
8:             Class c = loader.loadClass(args[1]);
9:             Object o = c.newInstance();
10:            Method mt = o.getClass().getMethod(args[2]);
11:            mt.invoke(o);
12:        } catch (Exception ex) {
13:            System.out.println(ex.toString());
14:        }
15:    }
16:}
```

รหัสที่ 5.7 ตัวสั่งเมท็อดกำกับ `@VMain` ทำงาน

5.4.2.4 การสั่งตัวติดต่อจาวาเวอร์ชวลแมชชีนทำงาน

ตัวติดต่อจาวาเวอร์ชวลแมชชีนทำหน้าที่เรียกคลาส `VRecursiveRunner` ทำงาน ด้วยการส่งอากิวเมนต์คำสั่งเรียกโปรแกรมค้นปริภูมิผ่านตัวติดต่อจาวาเวอร์ชวลแมชชีน ดังรหัสที่ 5.8 เมท็อด `LaunchTarget` รับตัวติดต่อจาวาเวอร์ชวลแมชชีนและอากิวเมนต์คำสั่งเรียกโปรแกรมค้นปริภูมิ (บรรทัดที่ 1) จากนั้นตัวติดต่อจาวาเวอร์ชวลแมชชีนเรียกเมท็อด `launch` สั่งดำเนินการเรียกคลาส `VRecursiveRunner` และคืนค่าติดต่อ `VirtualMachine` กลับมาให้ระบบแสดงภาพ (บรรทัดที่ 3)

```
1. VirtualMachine LaunchTarget(LaunchingConnector connector,Map arguments){
2.     try{
3.         return connector.launch(arguments);
4.     }
5.     catch(IOException exc) {. . .}
6.     catch (IllegalConnectorArgumentsException exc) {. . .}
7.     catch (VMStartException exc) {. . .}
8. }
```

รหัสที่ 5.8 การเริ่มดำเนินการเรียกโปรแกรม

เมื่อโปรแกรมค้นปริภูมิถูกสั่ง การติดตามปมสถานะที่ถูกผลิตทำได้ด้วยการสร้างคลาสที่เป็นคลาสลูกของ `Thread` จากนั้นสั่ง `start` เพื่อติดตามการผลิตปมสถานะผ่านคลาส `VirtualMachine` ดังรหัสที่ 5.9 เมท็อด `running` รับ `VirtualMachine` จากนั้นสร้างคลาส `VRecursiveEvent` ซึ่งเป็นคลาสลูกของ `Thread` รับค่า `VirtualMachine` และตัวเก็บข้อกำหนด `VMainUtil` จากนั้นสั่ง `start` เพื่อเรียกดูเหตุการณ์การทำงานโปรแกรมเรียกซ้ำ

```
private void running(VirtualMachine vm){
    try{
        VRecursiveEvent eventThread = new VRecursiveEvent(vm,vutil);
        eventThread.start();
    }catch(InterruptedException exc) {. . .}
}
```

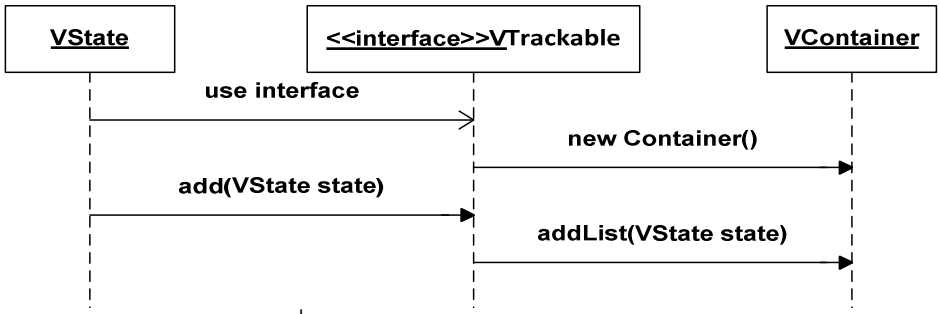
รหัสที่ 5.9 การสั่งตัวติดต่อจาวาเวอร์ชวลแมชชีนทำงาน

5.5 การติดตามสถานะเกิดใหม่

การค้นคำตอบในปริภูมิสถานะย่อยต้องมีการผลิตสถานะใหม่ ๆ ระหว่างการค้นระบบแสดงภาพจะติดตามและเก็บสถานะเกิดใหม่นี้เพื่อนำไปใช้แสดงภาพ ซึ่งต้องอาศัยความสัมพันธ์ปมพ่อแม่ การวาดภาพบนปมสถานะ ค่าบูลีนความเข้มเส้นเชื่อมที่พุ่งเข้าสู่ปมสถานะ และค่าสตริงกำกับปม กระบวนการนี้เรียกว่า “State Tracking” โดยแบ่งลักษณะการติดตามสถานะออกเป็น 2 แบบคือ

5.5.1 การติดตามอ็อบเจกต์สถานะ

อ็อบเจกต์สถานะเกิดใหม่จากการค้นคำตอบด้วยวงวนทำซ้ำในปริภูมิสถานะนี้เป็นอ็อบเจกต์ของคลาสที่เป็นคลาสลูกของ `VState` โดยคลาสนี้มีอินเทอร์เฟสตัวติดตามสถานะ ทุกครั้งที่อ็อบเจกต์สถานะถูกผลิตสถานะเหล่านี้จะถูกส่งผ่านตัวติดตามสถานะเข้ามายังตัวบันทึกสถานะ ซึ่งกระบวนการนี้เรียกว่า “NewState Launcher” ดังรูปที่ 5.6



รูปที่ 5.6 การติดตามอ็อบเจกต์สถานะ

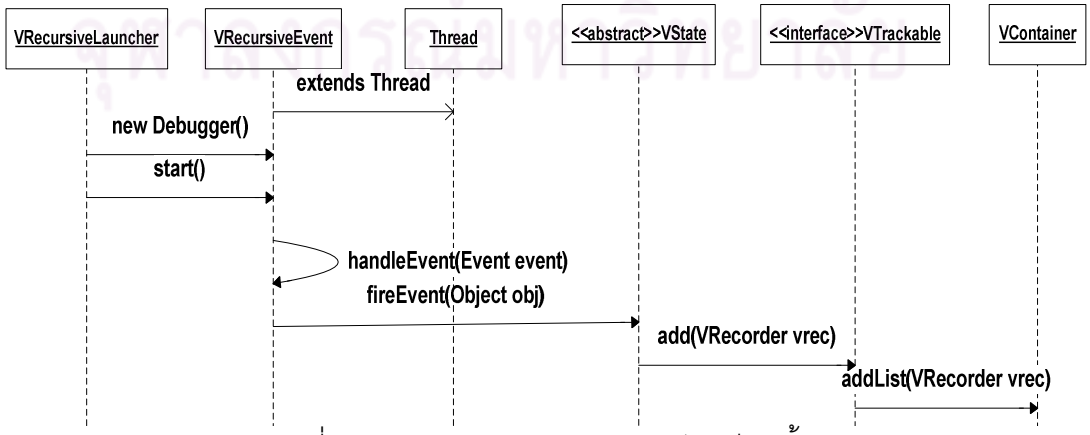
รหัสที่ 5.10 แสดงคลาส `VState` ที่มีอินเทอร์เฟสตัวติดตามสถานะ `VTrackable` เมื่อมีอ็อบเจกต์สถานะเกิดใหม่จะถูกส่งมายังเมท็อดติดตาม `fireEvent` ซึ่งเมท็อดนี้ส่งสถานะเกิดใหม่เข้าสู่คลาส `Container` ซึ่งเป็นตัวบันทึกสถานะ

```
1:  public interface VTrackable extends Icon{
2:      public Container container = new Container();
3:  }
4:  public abstract class VState implements VTrackable{
5:      public static void fireEvent(VState fireEvent) {
6:          container.addList(state);
7:      }
8:      public static void fireEvent(VRecorder rec) {
9:          container.addList(rec);
10:     }
11:     public boolean isPruned(){
12:         return false;
13:     }
14:     . . .
15:     public abstract VState getParent();
16: }
```

รหัสที่ 5.10 การติดตามอ็อบเจกต์สถานะ

5.5.2 การติดตามเมท็อดเรียกซ้ำ

สถานะที่เกิดใหม่จากการเรียกซ้ำเมท็อดได้จากการการเรียกดูเหตุการณ์โดยอาศัยชุดคำสั่งจาวาที่เรียกว่า JDI ทุกครั้งที่เมท็อดเรียกซ้ำถูกเรียกระบบแสดงภาพจะติดตามและเก็บสถานะต่าง ๆ ที่ต้องการ ซึ่งกระบวนการนี้เรียกว่า “Method Listener” ดังรูปที่ 5.7



รูปที่ 5.7 การติดตามสถานะจากเมท็อดเรียกซ้ำ

เมื่อคลาส VRecursiveEvent (รหัสที่ 5.11) ซึ่งทำงานภายใต้หน่วยประมวลย่อยทำงานภายใต้เมท็อด run เรียกดูเหตุการณ์ด้วยคลาส VirtualMachine ผ่านเมท็อด eventQueue (บรรทัดที่ 10) จากนั้นเข้าสู่วงวนเพื่อตรวจสอบเหตุการณ์(บรรทัดที่ 11-21) โดย EventSet (บรรทัดที่ 13) รับลำดับเหตุการณ์และวนหาเหตุการณ์ที่ระบบแสดงภาพต้องการติดตาม(บรรทัดที่ 15-17) ด้วยเมท็อด handleEvent (บรรทัดที่ 16) ซึ่ง handleEvent จะติดตามการเรียกเมท็อดเรียกซ้ำด้วย methodEntryEvent การติดตามค่าส่งผ่านเมท็อดเรียกซ้ำบน stepEvent และติดตามการออกจากเมท็อดด้วย methodExitEvent จากนั้นเมื่อสิ้นสุดการติดต่อเหตุการณ์ระบบจะยกเลิกวงวนตรวจสอบเหตุการณ์ผ่านพารามิเตอร์ connected (บรรทัดที่ 30) ซึ่งรายละเอียดการติดตามเมท็อดเรียกซ้ำมีดังนี้

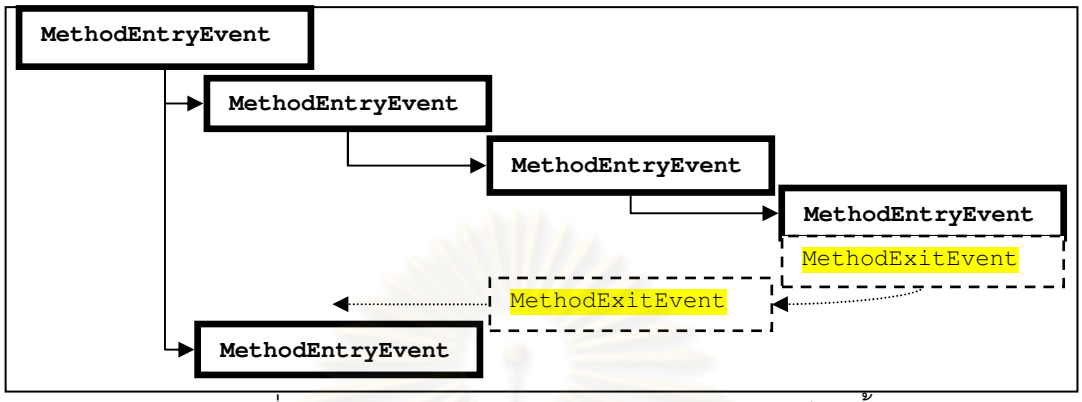
```
1: public class VRecursiveEvent extends Thread{
2:     VirtualMachine vm;
3:     VMainutil vutil;
4:     boolean connected = true;
5:     public Debugger(VirtualMachine vm,VMainutil vutil){
6:         this.vm=vm;
7:         this.vutil=vutil;
8:     }
9:     public void run() {
10:          EventQueue queue = vm.eventQueue();
11:          while (connected) {
12:              try {
13:                  EventSet eventSet = queue.remove();
14:                  EventIterator it = eventSet.eventIterator();
15:                  while (it.hasNext()) {
16:                      handleEvent(it.nextEvent());
17:                  }
18:                  eventSet.resume();
19:              } catch (Exception ex) {. . .}
20:          }
21:      }
22:      private void handleEvent(Event event) {
23:          if (event instanceof MethodEntryEvent) {
24:              methodEntryEvent((MethodEntryEvent) event);
25:          } else if (event instanceof MethodExitEvent) {
26:              methodExitEvent((MethodExitEvent) event);
27:          } else if (event instanceof StepEvent) {
28:              stepEvent((StepEvent) event);
29:          } else if (event instanceof VMDisconnectEvent) {
30:              connected = false;
31:          }
32:      }
33:      . . .
34: }
```

รหัสที่ 5.11 ตัวติดตามเมท็อดเรียกซ้ำ

5.5.2.1 การติดตามความสัมพันธ์ของปมสถานะ

การตรวจสอบความสัมพันธ์ปมสถานะที่เกิดใหม่กับปมพ่อแม่นั้นทำได้ด้วยการติดตามการเรียก เข้า-ออกเมท็อดผ่าน MethodEntryEvent และ MethodExitEvent ด้วย JDI โดยหาก MethodEntryEvent ใดอยู่ภายใต้การทำงานอีก MethodEntryEvent หนึ่งถือว่าเป็นปมลูกของ

สถานะนั้น หรือ MethodEntryEvent มีการทำงานระดับเดียวกันจะเป็นพี่น้องกัน และหาก MethodExitEvent ถูกเรียกจะเป็นการสิ้นสุดกระบวนการทำงานเมท็อดที่กำลังประมวลผลอยู่ แสดงได้ดังรูปที่ 5.8



รูปที่ 5.8 แบบจำลองการติดตามเหตุการณ์เมท็อดเรียกซ้ำ

รหัสที่ 5.12 เมื่อเมท็อด methodEntryEvent ถูกเรียก เมท็อด isMethodEntry (บรรทัดที่ 7) จะตรวจสอบเงื่อนไขว่าเป็นเมท็อดเรียกซ้ำที่ประกาศไว้บน @Vmain หรือไม่ จากนั้นใช้ตัวชี้ปม child_pointer นับการเรียกเมท็อดเรียกซ้ำและเก็บลำดับในเวกเตอร์ mapnode (บรรทัดที่ 10) สำหรับบรรทัดที่ 11 ตัวชี้ระดับปม level_pointer นับระดับปมบนต้นไม้ปริภูมิสถานะและเก็บลำดับในเวกเตอร์ maplevel หลังจากนั้นค้นหาปมพ่อแม่ด้วยเมท็อด searchParent และสร้างตัวบันทึกปม VRecorder เก็บสถานะปมพ่อแม่และปมลูก

```
1: public class VRecursiveEvent extends Thread {
2:   . . .
3:   VRecorder vrec;
4:   int child_pointer = 1 ;
5:   int level_pointer = 1 ;
6:   Vector mapnode = new Vector();
7:   Vector maplevel = new Vector()
8:   public void methodEntryEvent(MethodEntryEvent event){
9:       if(isMethodEntry(event)){
10:          mapnode.add(child_pointer);
11:          maplevel.add(level_pointer);
12:          int parent=searchParent(level_pointer-1);
13:          vrec = new VRecorder(parent,child_pointer);
14:          node_pointer++; level_pointer++;
15:      }
16:  }
17:  public void methodExitEvent(MethodExitEvent event){
18:      if(isMethodExit(event)) level_pointer--;
19:  }
20:  public int searchParent(int level_pointer){
21:      for (int lp = maplevel.size() - 1; lp >= 0; lp--) {
22:          if(maplevel.get(lp).equals(level_pointer))
23:              return Integer.parseInt(mapnode.get(r).toString());
24:      }
25:      return 0;
26:  }
27:}
```

รหัสที่ 5.12 การติดตามเมท็อดเรียกซ้ำ

5.5.2.2 การติดตามค่าพารามิเตอร์ของเมท็อดเรียกซ้ำ

เมื่อเมท็อดเรียกซ้ำถูกเรียก ระบบแสดงภาพจะติดตามค่าพารามิเตอร์ที่ส่งเข้ามายังเมท็อดเรียกซ้ำ ดังรหัสที่ 5.13 เมท็อด `stepEvent` (บรรทัดที่ 1) รับคลาส `StepEvent` ซึ่งคลาสนี้ระบบแสดงภาพ JState101 ใช้เรียกดูค่าที่ส่งผ่านเข้ามายังเมท็อดเรียกซ้ำผ่าน `StackFrame` (บรรทัดที่ 2) ด้วยการตรวจสอบเงื่อนบรรทัดที่ `StackFrame` กำลังเรียกดูบรรทัดที่เมท็อดเรียกซ้ำบนโปรแกรมค้นปริภูมิรับค่า (บรรทัดที่ 8) จากนั้นวนเรียกดูค่าพารามิเตอร์ที่เมท็อดเรียกซ้ำรับค่าผ่าน `StackFrame` ด้วยวงวน (บรรทัดที่ 9-12) ซึ่งภายในวงวนมีลิสต์เก็บค่าที่อ่านได้จาก `StackFrame` (บรรทัดที่ 10-11)

หากตัวกำกับแสดงภาพ `@VMain` มีการสั่งให้ติดตาม `showAllStates` ระบบแสดงภาพนี้จะตรวจสอบเงื่อนและอ่านค่าพารามิเตอร์บูลีนตัวท้ายสุดที่ส่งเข้ามายังเมท็อดเรียกซ้ำ หรือกรณีสั่งให้ระบบแสดงภาพกำกับปม ระบบจะส่งจากนั้นส่งลิสต์เก็บพารามิเตอร์ที่อ่านได้จาก `StackFrame` ไปสร้างตัวแสดงภาพกำกับปม `VIcon` (บรรทัดที่ 14)

เมื่อสถานะการแสดงภาพ `VRecorder` ถูกเก็บแล้ว (บรรทัดที่ 15-17) ตัวบันทึกสถานะ `VRecorder` จะถูกส่งไปยังส่วนบันทึกยังส่วนบันทึกสถานะผ่าน `VState` (บรรทัดที่ 18) ซึ่งมีอินเทอร์เฟสติดตาม `Trackable` อยู่ (รหัสที่ 5.9)

```
1: public void stepEvent(StepEvent event) {
2:     StackFrame sf = event.thread().frame(0);
3:     boolean showAllStates = false;
4: String valueToString = “”;
5: List args;
6:     int methodLineNumber = event.location().lineNumber();
7:     int currentLineNumber = sf.location().lineNumber();
8:     if(methodLineNumber==currentLineNumber){
9:         for (LocalVariable localvar : sf.visibleVariables()) {
10:        args = new ArrayList();
11:            . . .
12:        }
13:    }
14:    VIcon icon = new VIcon(vutil, args);
15:    vrec.setIcon(icon);
16:    vrec.setisPruned(showAllStates);
17:    vrec.setString(valueToString);
18:    VState.fireEvent(vrec);
19:. . .
20:}
```

รหัสที่ 5.13 การติดตามการรับค่าของเมท็อดเรียกซ้ำ

5.5.2.3 การสร้างตัวแสดงภาพกำกับปม

กรณีที่มีการเขียนคลาสที่เป็นคลาสลูกของ `VDrawable` ระบบจะเรียกตัวแสดงภาพกำกับปม `VIcon` ซึ่งทำหน้าที่เรียกเมท็อด `drawState` ในโปรแกรมค้นคำตอบแบบเรียกซ้ำและคืนค่ากลับไปยังระบบแสดงภาพ ดังรหัสที่ 5.14 ตัวแสดงภาพกำกับปม `VIcon` ทำตามจาวาอินเทอร์เฟส `Icon` เมื่อระบบแสดงภาพใช้งานตัวแสดงภาพกำกับปม ระบบจะเรียกเมท็อด `paintIcon` ของ

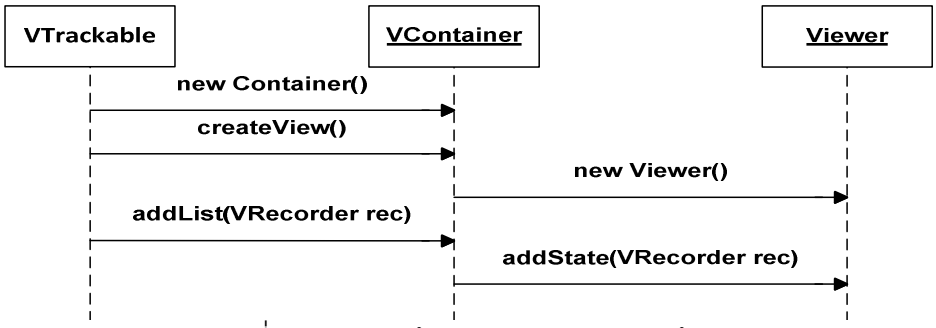
อินเทอร์เฟส `Icon` จากนั้นเมท็อด `paintIcon` ส่งค่าวาดภาพ `Graphics2D` (บรรทัดที่ 10) ตำแหน่งภาพแนวแกน `x, y` และลิสต์เก็บค่าส่งผ่านเมท็อดเรียกซ้ำไปยังเมท็อด `drawState` (บรรทัดที่ 21) ของโปรแกรมเรียกซ้ำและเรียกค่าคืนรูปภาพ (บรรทัดที่ 33-36)

```
1: public class VIcon implements Icon{
2:     VMainUtil vutil;
3:     List args;
4:     public Paintable(VMainUtil vutil,List args){
5:         this.vutil=vutil;
6:         this.args = args;
7:     }
8:     @Override
9:     public void paintIcon(Component c, Graphics g, int x, int y) {
10:        Graphics2D g2d = (Graphics2D) g;
11:        drawState(g2d, x, y, args);
12:    }
13:    @Override
14:    public int getIconWidth() {
15:        return 20;
16:    }
17:    @Override
18:    public int getIconHeight() {
19:        return 20;
20:    }
21:    public void drawState(Graphics2D g2d, int x, int y,List args) {
22:        try {
23:            Class partypes[] = new Class[4];
24:            partypes[0] = Graphics2D.class;
25:            partypes[1] = Integer.TYPE;
26:            partypes[2] = Integer.TYPE;
27:            partypes[3] = List.class;
28:            Object[] argument = new Object[4];
29:            args[0]=g2d;
30:            args[1]=x;
31:            args[2]=y;
32:            args[3]=args;
33:            Class c = Class.forName(vutil.getStateClassName());
34:            Object obj = c.newInstance();
35:            Method mt = obj.getClass().getMethod("drawState",partypes);
36:            mt.invoke(obj1, argument);
37:        }catch (Exception ex) {
38:           System.out.println(ex.toString());
39:        }
40:    }
41:}
```

รหัสที่ 5.14 ตัวแสดงภาพกำกับปม

5.6 การบันทึกสถานะ

สถานะที่เกิดใหม่นั้นจะถูกระบบติดตามและเก็บสถานะต่างๆที่ JState101ต้องการไว้ในอ็อบเจกต์บันทึก จากนั้นระบบจะบันทึกอ็อบเจกต์นี้เข้าสู่ลิสต์บันทึกภายใต้การทำงานของหน่วยประมวลผลย่อยเดียวกัน ซึ่งกระบวนการนี้เรียกว่า “State Recording” ซึ่งมีขั้นตอนการทำงานดังรูปที่ 5.9



รูปที่ 5.9 การบันทึกปมสถานะสู่ลิสต์บันทึก

รหัสที่ 5.15 คลาส `Container` เป็นตัวบันทึกสถานะ เมื่อถูกเรียก constructor จะสร้างอ็อบเจ็กต์ `LinkedHashMap` ไว้สำหรับจัดเก็บส่วนแสดงภาพ `Viewer` ตามชื่อหน่วยประมวลผลย่อย (บรรทัดที่ 4) หากตัวเรียกเมท็อดดำเนินงานทำงานระบบจะสั่งเมท็อด `createView` ให้สร้างส่วนแสดงภาพ `Viewer` (บรรทัดที่ 7-8) จากนั้นเมื่อสถานะใหม่ถูกผลิตจะมีเมท็อด `addStateToList` (บรรทัดที่ 10 และ 16) รองรับการผลิตสถานะทั้งแบบอ็อบเจกต์สถานะและแบบเรียกซ้ำเพื่อส่งเข้าสู่ส่วนแสดงภาพ โดยรายละเอียดการทำงานแต่ละขั้นตอนมีดังนี้

```
1: public class VContainer {
2:   private static LinkedHashMap<String, Viewer> screen;
3:   public VContainer() {
4:       screen = new LinkedHashMap<String, Viewer>();
5:   }
6:   public void createView() {
7:   String screenname = Thread.currentThread().getThreadGroup().getName();
8:       screen.put(title, new ActionScreen(screenname));
9:   }
10:  public void addStateToList(VState obj) {
11:      VRecorder vrec = new VRecorder(obj.getParent(), obj);
12:      vrec.setisPruned(obj.isPruned());
13:      vrec.setString(obj.toString());
14:      vrec.setIcon((Icon)obj);
15:      addStateToList(vrec);
16:  }
17:  public void addStateToList(VRecorder rec) {
18:      try{
19:  String screenname = Thread.currentThread().getThreadGroup().getName();
20:          screen.get(screenname).addState(rec);
21:      }catch(Exception ex){
22:          System.out.println(ex.toString());
23:      }
24:  }
25:}
```

รหัสที่ 5.15 ตัวบันทึกสถานะ

5.6.1 การสร้างส่วนแสดงภาพ

ส่วนแสดงภาพจะถูกสร้างขึ้นเมื่อตัวเรียกเมท็อดดำเนินงานทำงาน โดยก่อนที่จะมีการสั่งเมท็อดดำเนินงานทำงานนั้นระบบจะสร้างส่วนแสดงภาพ `Viewer` (รหัสที่ 5.15 บรรทัดที่ 7-8) เพื่อรองรับสถานะที่ถูกผลิตขึ้นมาตามเงื่อนไขในแต่ละหน่วยประมวลผลย่อย

5.6.2 ตัวเก็บสถานะแสดงภาพ

ตัวเก็บสถานะแสดงภาพ `VRecorder` ทำหน้าที่รับค่าพารามิเตอร์ต่างๆ ที่ระบบต้องการจากการติดตามสถานะก่อนนำไปจัดเก็บในลิสต์ส่วนแสดงภาพ ข้อกำหนดที่ระบบจัดเก็บคือ ค่าสตริงปมพ่อแม่ ค่าสตริงปมลูก การวาดภาพกำกับปมสถานะ การกำหนดความเข้มเส้นเชื่อมปมสถานะ ค่าสตริงกำกับปม ดังรหัส 5.16 เมื่อตัวเก็บสถานะแสดงภาพ `VRecorder` ถูกสร้างตัวบันทึกสถานะ `VContainer` เก็บ `VRecorder` เข้าสู่ส่วนแสดงภาพ `Viewer` (รหัสที่ 5.15 บรรทัดที่ 20) ซึ่งมีลิสต์บันทึกสถานะอยู่ภายใน

```
1:  public class VRecorder {
2:      Object parentID;
3:      Object childID;
4:      String value;
5:      Icon icon;
6:      boolean ispruned;
7:      public VRecorder(Object parenteID, Object childID){
8:          this.parenteID = parenteID;
9:          this.childID = childID;
10:     }
11:     public Object getChildID() {
12:         return childID;
13:     }
14:     public void setChildID(Object childID) {
15:         this.childID = childID;
16:     }
17:     public Icon getIcon() {
18:         return icon;
19:     }
20:     public void setIcon(Icon icon) {
21:         this.icon = icon;
22:     }
23:     public boolean getisPruned() {
24:         return ispruned;
25:     }
26:     public void setisPruned(boolean ispruned) {
27:         this.ispruned = ispruned;
28:     }
29:     public Object getParentID() {
30:         return parentID;
31:     }
32:     public void setParentID(Object parentID) {
33:         this.parentID = parentID;
34:     }
35:     public String getValue() {
36:         return value;
37:     }
38:     public void setValue(String value) {
39:         this.value = value;
40:     }
41: }
```

รหัสที่ 5.16 ตัวเก็บสถานะแสดงภาพ

5.6.3 การเก็บสถานะแสดงภาพ

สถานะที่ถูกผลิตจะถูกระบบแสดงภาพติดตามและเก็บสถานะต่าง ๆ ไว้ใน `VRecorder` ซึ่งรายละเอียดการเรียกเก็บสถานะแสดงภาพมีดังนี้

5.6.3.1 การเก็บสถานะแสดงภาพที่ได้จากโปรแกรมเรียกซ้ำ

สถานะที่เกิดจากการเรียกซ้ำเมท็อดจะถูกติดตามจากคลาส `VRecursiveEvent` ดังที่กล่าวในข้างต้น ซึ่งสถานะแสดงภาพ `VRecorder` จะถูกสร้างและเก็บสถานะต่าง ๆ ก่อนที่จะส่งมายังตัวบันทึกสถานะ โดยตัวเก็บสถานะจะถูกส่งผ่านตัวติดตามสถานะ `VTrackable` มายังคลาส `VContainer` ผ่านเมท็อด `addStateToList` (รหัสที่ 5.15 บรรทัดที่ 17) จากนั้นบันทึกยังส่วนแสดงภาพ `Viewer` (รหัสที่ 5.16 บรรทัดที่ 20)

5.6.3.2 การเก็บสถานะแสดงภาพที่ได้จากโปรแกรมอ็อบเจกต์สถานะ

เมื่อสถานะแบบอ็อบเจกต์สถานะถูกผลิต ตัวติดตามสถานะ `VTrackable` ส่งอ็อบเจกต์สถานะ `VState` มายังคลาส `VContainer` ผ่านเมท็อด `addStateToList` (รหัสที่ 5.15 บรรทัดที่ 10) จากนั้นระบบแสดงภาพนี้จะอ่านและเก็บสถานะที่ต้องการไว้ในตัวเก็บสถานะ (รหัสที่ 5.15 บรรทัดที่ 11-14) และส่งไปบันทึกยังส่วนแสดงภาพผ่านเมท็อด `addStateToList` (รหัสที่ 5.15 บรรทัดที่ 15) เพื่อบันทึกยังส่วนแสดงภาพ `Viewer` (รหัสที่ 5.15 บรรทัดที่ 20)

5.6.3.3 ลิสต์เก็บบันทึกสถานะแสดงภาพ

เมื่อตัวเก็บสถานะแสดงภาพ `VRecorder` ถูกส่งผ่านเมท็อด `addStateToList` (รหัสที่ 5.15 บรรทัดที่ 20) ระบบจะเรียกส่วนแสดงภาพด้วยชื่อหน่วยประมวลผลย่อย (รหัสที่ 5.15 บรรทัดที่ 19) และส่งตัวเก็บสถานะมาบันทึกยังส่วนแสดงภาพ `Viewer` ซึ่งภายในมี `List` เป็นคลาสจาวาสำหรับเก็บสถานะแสดงภาพ `VRecorder` ดังรหัสที่ 5.17 เพื่อรอนำไปแสดงผลในส่วนติดต่อผู้ใช้ต่อไป

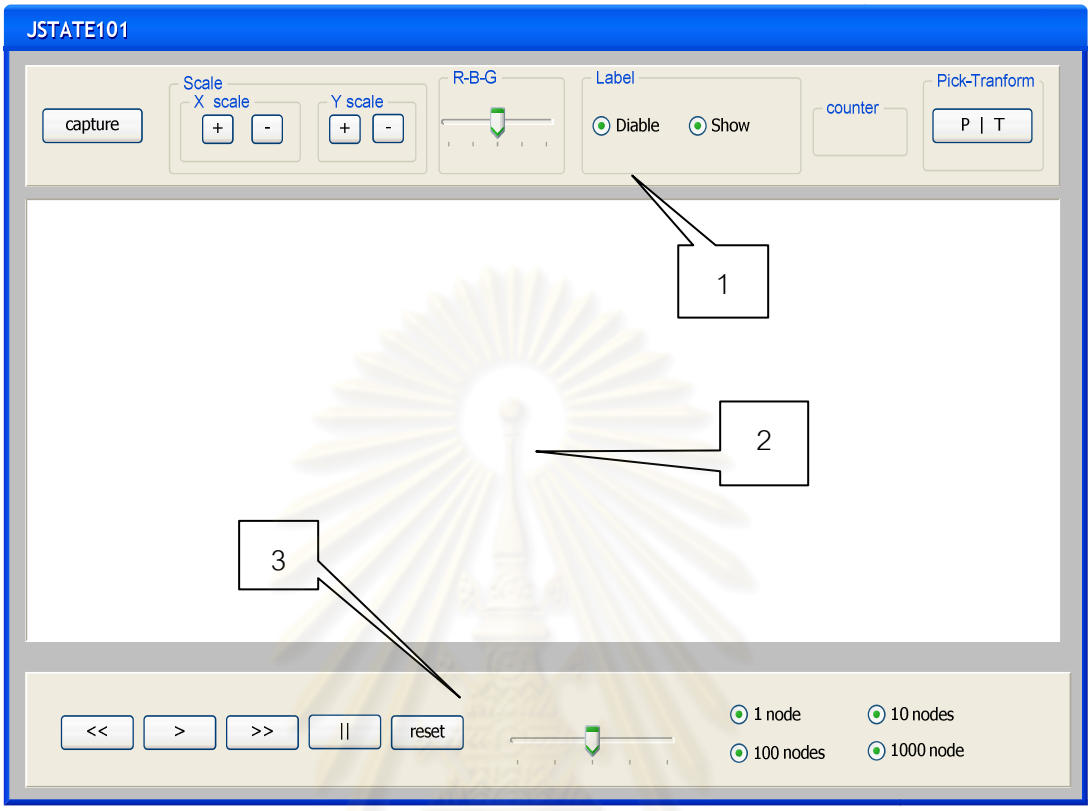
```
public class Viewer extends JFrame{
    private List<VRecorder> state;
    . . .
    public void addState(VRecorder vrec){
        state.add(vrec);
    }
}
```

รหัสที่ 5.17 List เก็บสถานะแสดงภาพ

5.7 การแสดงภาพต้นไม้ปริภูมิสถานะ

ก่อนที่เมท็อดดำเนินงานซึ่งกำกับด้วย `@VMain` ถูกระบบสั่งทำงาน ระบบแสดงภาพจะสร้างส่วนติดต่อผู้ใช้ขึ้นมารองรับการแสดงภาพต้นปริภูมิสถานะ เมื่อปมสถานะถูกผลิตจะถูกระบบติดตามและบันทึกปมสถานะมายังส่วนแสดงภาพ การทำงานของส่วนติดต่อผู้ใช้จะเป็นลักษณะคล้ายเครื่องเล่นวิดิทัศน์ ผู้ใช้สามารถดูการดูเหตุการณ์การเปลี่ยนแปลงการแจงผลเฉลยย้อนหลังหรือเดินหน้าได้ ซึ่งการทำงานส่วนนี้เรียกว่า “State Space Tree Viewer & Player” ดังรูปที่ 5.10 แบ่งหน้าที่ส่วนติดต่อผู้ใช้ออกเป็นสามส่วนคือ หมายเลข 1 ส่วนปรับแต่งลักษณะภาพต้นไม้ปริภูมิ

สถานะ หมายเลข 2 ส่วนแสดงแผนภาพต้นไม้ปริภูมิสถานะ และหมายเลข 3 ส่วนควบคุมการนำเสนอภาพในลักษณะเครื่องเล่นวีดิทัศน์



รูปที่ 5.10 ส่วนติดต่อผู้ใช้สำหรับแสดงภาพต้นไม้ปริภูมิสถานะ

งานวิจัยนี้อาศัย Framework แสดงภาพที่เรียกว่า “JUNG” [20] หรือ Java Universal Network/Graph Framework ในการแสดงภาพต้นไม้ปริภูมิสถานะ โดยระบบแสดงภาพจะทำหน้าที่ควบคุมและส่งคำสั่งการผลิตภาพไปยัง JUNG ให้ผลิตภาพตามที่ระบบต้องการซึ่งการทำงานมีรายละเอียดดังนี้

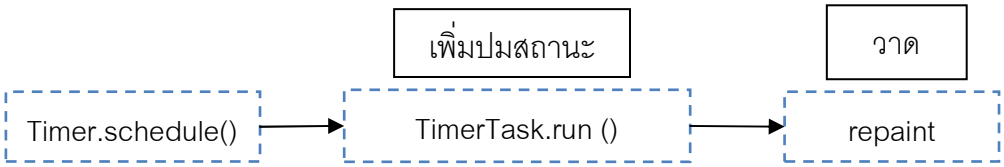
5.7.1 **ส่วนควบคุมการนำเสนอภาพในลักษณะเครื่องเล่นวีดิทัศน์**

ผู้ใช้สามารถควบคุมการนำเสนอภาพในลักษณะเครื่องเล่นวีดิทัศน์ผ่านปุ่มควบคุมภาพ คือ ปุ่มเล่นภาพ ปุ่มเดินหน้า/ย้อนกลับภาพทีละปม ปุ่มหยุดภาพชั่วขณะ และปุ่มตั้งค่าแสดงภาพใหม่ รวมทั้งการกำหนดการหน่วงเวลาการแสดงภาพ และการกำหนดจำนวนปมที่ส่งไปยัง JUNG ในการวาดใหม่ ซึ่งรายละเอียดปุ่มควบคุมการเล่นภาพมีดังนี้

5.7.1.1 การควบคุมภาพเดินหน้าและหยุดภาพชั่วขณะ

เมื่อมีคำสั่งเล่นภาพจากส่วนติดต่อผู้ใช้ระบบแสดงภาพจะเรียกค่าตัวเก็บสถานะแสดงภาพจากลิสต์และส่งสถานะนั้นไปยัง JUNG เพื่อวาดภาพต้นไม้ การส่งสถานะวาดภาพนั้นคลาส

ในระบบแสดงภาพจะเป็นคลาสลูกของ `TimerTask` และสร้างจาวา Timer เรียกการทำงานผ่านเมท็อด `schedule` สำหรับควบคุมการส่งสถานะวาดภาพเข้าสู่ JUNG ดังรูปที่ 5.11



รูปที่ 5.11 การสร้าง `TimerTask` ควบคุมการเพิ่มปมสถานะ

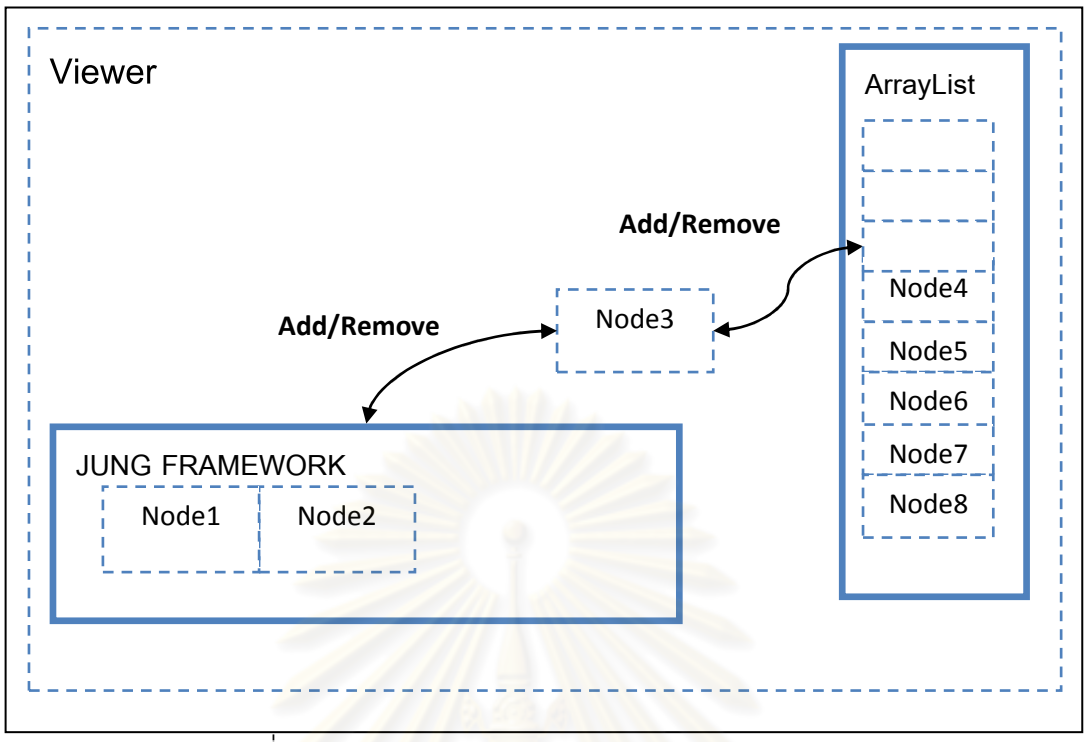
รหัสที่ 5.19 ภายใต้คลาสแสดงภาพ `Viewer` มีคลาส `RemindTask` ที่เป็นคลาสลูกของ `TimerTask` เมื่อส่วนติดต่อผู้ใช้สั่งเล่นภาพระบบจะเรียก `start` ดำเนินการสร้าง `Timer` และสั่ง `RemindTask` ทำงานผ่านเมท็อด `schedule` จากนั้น `RemindTask` เรียกเมท็อด `run` ทำงานส่งสถานะวาดภาพทีละปมไปยัง JUNG ตามค่าหน่วงเวลาที่กำหนด กรณีที่มีการสั่งหยุดภาพชั่วขณะจากส่วนติดต่อผู้ใช้ระบบจะเรียกเมท็อด `pause` เพื่อ terminate การทำงาน `Timer`

```
1: public class Viewer extends JFrame{
2:     private List<VRecorder> state;
3:     Timer timer;
4:     boolean play=false;
5:     int cursor=0;
6:     int delay= 500;
7:     public class RemindTask extends TimerTask {
8:         @Override
9:         public void run() {
10:            if(state.size()>0){
11:                addnode(state.get(cursor));
12:                state.remove(cursor);
13:                repaint();
14:                cursor++;
15:            }
16:        }
17:    }
18:    public void start(){
19:        timer = new Timer();
20:        timer.schedule(new RemindTask(), 0, time);
21:    }
22:    public void pause(){
23:        timer.cancle();
24:    }
25:    . . .
26: }
```

รหัสที่ 5.18 การสร้าง `TimerTask` ควบคุมการเพิ่มปมสถานะ

5.7.1.2 การย้ายปมสถานะเข้าสู่ JUNG FRAMEWORK

เมื่อตัวเก็บสถานะแสดงภาพ `VRecorder` ถูกดึงออกจากลิสต์บันทึกเพื่อส่งสถานะสำหรับสร้างต้นไม้ที่ JUNG ต้องการ หลังจากนั้นระบบจะลบตัวเก็บสถานะที่ถูกดึงออกจากลิสต์ ในทางกลับกันเมื่อปมถูกลบออกจาก JUNG ระบบสร้างตัวเก็บสถานะแสดงภาพ `VRecorder` และดึงสถานะเก็บกลับลง `VRecorder` จากนั้นบันทึกเข้าสู่ลิสต์ตามเดิม ดังรูปที่ 5.12



รูปที่ 5.12 การย้ายปมสถานะเข้าสู่ JUNG FRAMEWORK

รูปที่ 5.12 เมื่อส่วนติดต่อผู้ใช้ถูกสั่งแสดงภาพผ่านปุ่มควบคุม สถานะใน `VRecorder` จะถูกย้ายจากลิสต์เก็บสถานะการแสดงภาพเข้าสู่ JUNG ซึ่งเป็น FRAMEWORK สำหรับแสดงภาพต้นไม้ปริภูมิสถานะ ในทางตรงกันข้ามหากส่วนติดต่อผู้ใช้ถูกสั่งย้อนกลับการแสดงภาพ ระบบแสดงภาพจะอ่านข้อกำหนดจาก JUNG FRAMEWORK และเก็บในตัวเก็บข้อกำหนด `VRecorder` จากนั้นเก็บกลับเข้าสู่ลิสต์เก็บสถานะการแสดงภาพดังเดิม

บทที่ 6

ตัวอย่าง และ ผลการทดลอง

6.1 การค้นหาคำตอบสำหรับปัญหาทั่วไป

ตัวอย่างที่ 1 ปัญหา sum of subset

ปัญหา Sum of subset ในการค้นหาคำตอบแนวลึกและการค้นหาตามแนวกว้าง โดยกำหนดเซตจำนวนเต็ม A= {1, 2, 3, 4, 5, 6, 7, 8, 9} และจำนวนเต็ม 16 เพื่อหาคำตอบแรกของเซตย่อย A ที่ผลรวมมีค่าเท่ากับ 16 ซึ่งสามารถกำหนดปริภูมิสถานะดังนี้

1 รูปแบบหนึ่งของสถานะในการค้นคำตอบของปัญหานี้ คือ เวกเตอร์แบบบิต X = <x_1, x_2, …, x_n> โดยที่ n คือขนาดของเซต A, x_i มีค่าเป็น 0 หรือ 1 ถ้า x_i = 1 แทนการเลือก a_i อยู่ในเซตย่อย (ถ้า x_i = 0 แทนการไม่เลือก a_i)

2 ให้สถานะเริ่มต้นคือเวกเตอร์แบบบิตความยาว 0: < >

3 ให้สถานะผลเฉลยคือเวกเตอร์แบบบิตความยาว n และสถานะคำตอบคือเวกเตอร์ X ที่ $\sum_{i=1}^{n} x_i a_i$ มีค่าเท่ากับ k

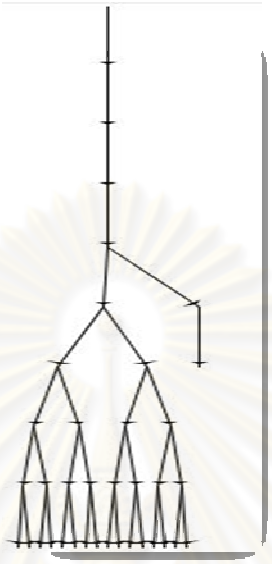
4 กำหนดให้ X_m คือเวกเตอร์ความยาว m เราสามารถสร้างเวกเตอร์ X_{m+1} ใหม่สองตัวจาก X_m โดยขยายขนาดของ X_m อีก 1 บิตที่มีบิตใหม่เป็น 0 และ 1 ตามลำดับ

วิธีที่ 1 การค้นคำตอบด้วยการค้นหาตามแนวลึกแบบเรียกซ้ำ จากรหัสที่ 6.1 โปรแกรมจะค้นคำตอบด้วยการวงวนโดยเมท็อด `sumOfSubsetRCS` รับเซตจำนวนเต็มด้วยอาเรย์ a และจำนวนเต็ม k คำตอบเป็นผลเฉลย สำหรับอาเรย์ x รับผลรวมเซตย่อย (ดูรหัสสมบูรณ์คำสั่งได้ที่ภาคผนวก ค.1)

```
public void sumOfSubsetRCS(int[] a, int k, int[] x) {
    if (x.length == a.length) {
        if (sum(a, x) == k) {
            System.out.println(Arrays.toString(x));
        }
    } else {
            int[] y = Arrays.copyOf(x, x.length + 1);
            y[y.length - 1] = 1;
            sumOfSubsetRCS(a, k, y);
            y = Arrays.copyOf(x, x.length + 1);
            y[y.length - 1] = 0;
            sumOfSubsetRCS(a, k, y);
    }
}
```

รหัสที่ 6.1 การค้นหาคำตอบแนวลึกของปัญหา sum of subset บนการทำงานแบบเรียกซ้ำ

รูปที่ 6.1 ผลเฉลยจากการค้นคำตอบแบบลึกสำหรับปัญหา sum of subset การทำงานแบบวงวน สำหรับเซตจำนวนเต็ม A= {1, 2, 3, 4, 5, 6, 7, 8, 9} เพื่อหาคำตอบแรกของเซตย่อย A ที่ผลรวมมีค่าเท่ากับ 16 ใช้จำนวนปมคำตอบทั้งสิ้น 38 ปม



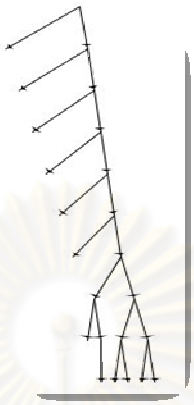
รูปที่ 6.1 ค้นหาคำตอบแนวลึกของปัญหา sum of subset บนการทำงานแบบเรียกซ้ำ

วิธีที่ 2 การค้นคำตอบแบบลึกก่อนโดยใช้โครงสร้างข้อมูลแบบกองซ้อนจัดการสถานะดังแสดงในรหัสที่ 6.2(ดูรหัสคำสั่งสมบูรณ์ได้ที่ภาคผนวก ค.1)

```
public void sumOfSubsetDFS(int[] a, int k) {
    Stack<VNode> s = new Stack<VNode>();
    int[] x = new int[0];
    VNode v = new VNode(x, null);
    s.push(v);
    while (!s.isEmpty()) {
        v = s.pop();
        if (v.x.length == a.length) {
            if (sum(a,v.x)) == k) {
                System.out.println(Arrays.toString(x));
            }
        } else {
            int[] y = Arrays.copyOf(v.x, v.x.length + 1);
            y[y.length - 1] = 1;
            VNode v2 = new VNode(y, v);
            s.push(v2);
            y = Arrays.copyOf(v.x, v.x.length + 1);
            y[y.length - 1] = 0;
            VNode v3 = new VNode(y, v);
            s.push(v3);
        }
    }
}
```

รหัสที่ 6.2 การค้นหาคำตอบแนวลึกของปัญหา sum of subset บนการทำงานแบบกองซ้อน

รูปที่ 6.2 ผลเฉลยจากการค้นหาคำตอบแบบลึกสำหรับปัญหา sum of subset การทำงานแบบกองซ้อน สำหรับเซตจำนวนเต็ม A= {1, 2, 3, 4, 5, 6, 7, 8, 9} เพื่อหาคำตอบแรกของเซตย่อย A ที่ผลรวมมีค่าเท่ากับ 16 ใช้จำนวนปมคำตอบทั้งสิ้น 24 ปม



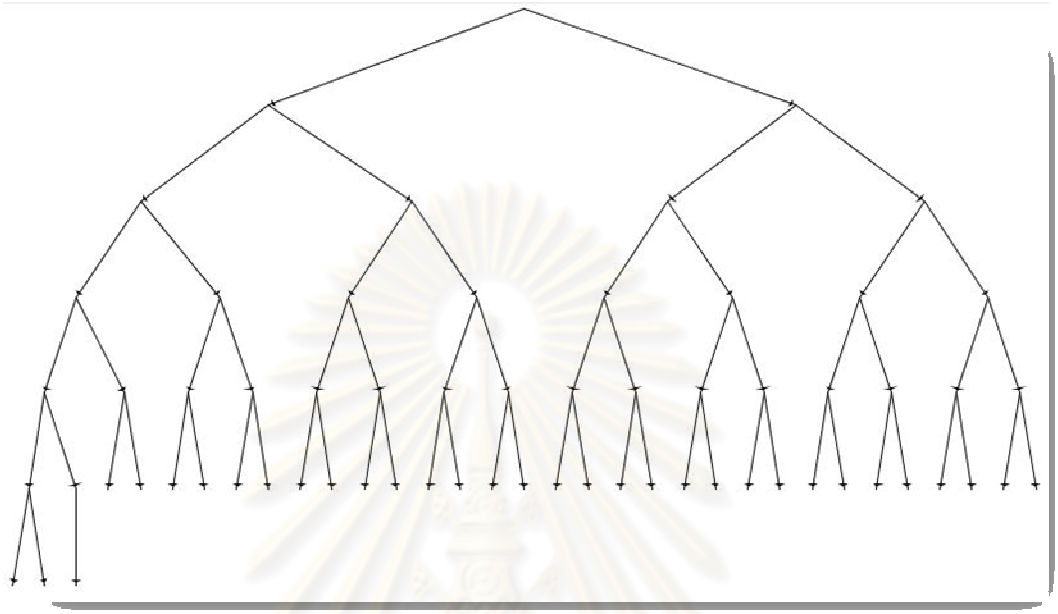
รูปที่ 6.2 การค้นหาคำตอบแนวลึกของปัญหา sum of subset บนการทำงานแบบกองซ้อน

วิธีที่ 3 การค้นหาแบบกว้างก่อนอาศัยโครงสร้างข้อมูลแบบแถวคอยจัดการสถานะดังแสดงในรหัสที่ 6.3 (ดูรหัสคำสั่งสมบูรณ์ได้ที่ภาคผนวก ค.1)

```
public void sumOfSubsetBFS(int[] a, int k) {
    Queue<VNode> q = new LinkedList<VNode>();
    int[] x = new int[0];
    VNode v = new VNode(x, null);
    q.add(v);
    while(!q.isEmpty()) {
        v = q.remove();
        if (v.x.length == a.length) {
            if(sum(a,v.x) == k) {
                System.out.println(Arrays.toString(x));
            }
        } else {
            int[] y = Arrays.copyOf(v.x, v.x.length + 1);
            y[y.length - 1] = 1;
            VNode v2 = new VNode(y, v);
            if (((int[]) v2.x).length == a.length) {
                if (sum(a, v2.x) == k) {
                    break;
                }
            }
            q.add(v2);
            y = Arrays.copyOf(v.x), v.x.length + 1);
            y[y.length - 1] = 0;
            VNode v3 = new VNode(y, v);
            if (((int[]) v3.x).length == a.length) {
                if (sum(a, v3.x) == k) {
                    break;
                }
            }
            q.add(v3);
        }
    }
}
```

รหัสที่ 6.3 การค้นหาคำตอบแนวลึกของปัญหา sum of subset บนการทำงานแบบแถวคอย

รูปที่ 6.3 ผลเฉลยจากการค้นหาคำตอบแบบลึกสำหรับปัญหา sum of subset การทำงานแบบแถวคอย สำหรับเซตจำนวนเต็ม A={1, 2, 3, 4, 5, 6, 7, 8, 9} เพื่อหาคำตอบแรกของเซตย่อย A ที่ผลรวมมีค่าเท่ากับ 10 ใช้จำนวนปมคำตอบทั้งสิ้น 66 ปม



รูปที่ 6.3 การค้นหาคำตอบแนวลึกของปัญหา sum of subset บนการทำงานแบบแถวคอย

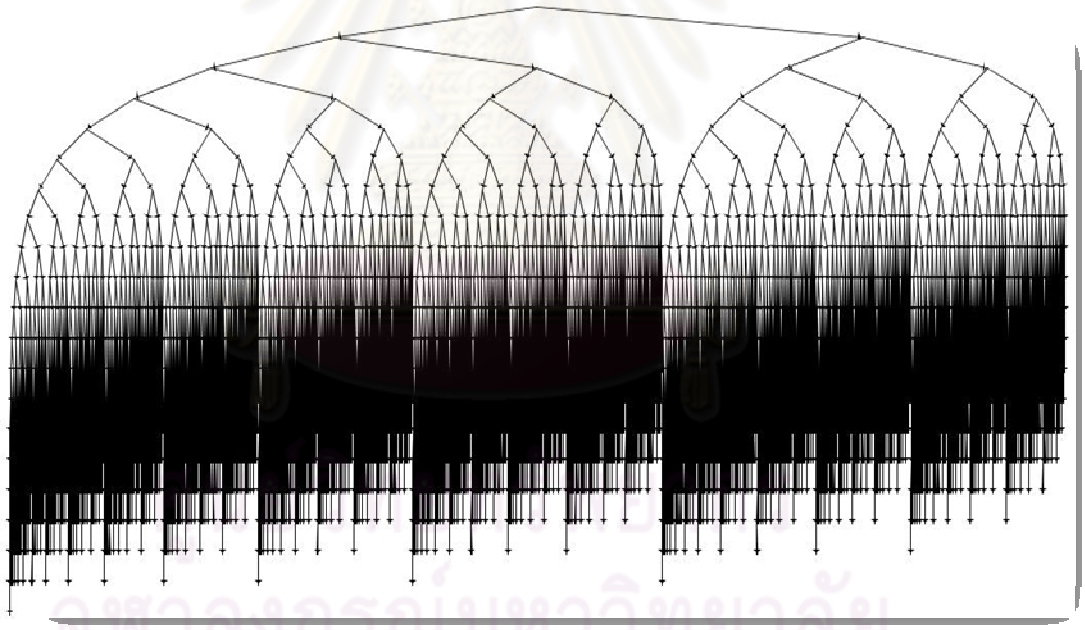
ตัวอย่างที่ 2 ปัญหา Fibonacci

การแก้ปัญหา Fibonacci [21] กับการค้นหาคำตอบแนวลึกแบบเรียกซ้ำและการใช้กลวิธี Memoization [22] เพื่อจำผลของเมท็อดไว้ในที่เก็บกลางเพื่อใช้ในอนาคตหากมีการเรียกซ้ำ เมื่อนำวิธีนี้ไปใช้กับเมท็อดแนวลึกแบบเรียกซ้ำที่แก้ปัญหาย่อยซ้ำ ๆ เพื่อหาคำตอบของปัญหาใหญ่จะลดเวลาการทำงานได้อย่างมาก ดังแสดงในรหัสที่ 6.4

```
static int[] mem = new int[50];
static int memoization(int n) { //กลวิธี memorization
        if (n < 2)return n;
        if (mem[n] > 0)return mem[n]; // คืนคำตอบแล้ว
        return mem[n] = memoization(n - 1) + memoization(n - 2); // จำคำตอบ
}
```

รหัสที่ 6.4 กลวิธี memoization สำหรับแก้ปัญหา Fibonacci

จากรูปที่ 6.4 แสดงภาพต้นไม้ปริภูมิสถานะเพื่อหาคำตอบ Fibonacci ของค่า 21 ด้วยการค้นหาคำตอบแนวลึกแบบเรียกซ้ำจะเห็นได้ว่าใช้ปม 35421 ปมในการค้นคำตอบ (ดูรหัสคำสั่งได้ที่ภาคผนวก ค.2)



รูปที่ 6.4 ต้นไม้ปริภูมิสถานะการหาคำตอบแนวลึก Fibonacci ด้วยแบบเรียกซ้ำ

จากรูปที่ 6.5 ภาพต้นไม้ปริภูมิสถานะเพื่อหาคำตอบ Fibonacci ของค่า 21 โดยเพิ่มกลวิธี memoization ลงในเมท็อดการทำงานแบบ**เรียกซ้ำ** เมื่อเปรียบเทียบกับการหาคำตอบแบบรูปที่ 6.4 จะเห็นได้ว่าปมนั้นลดลงอย่างมากจาก 35421 ปมเหลือ 41 ปม (ดูรหัสคำสั่งได้ที่ภาคผนวก ค.2)

รูปที่ 6.5 การหาคำตอบแนวลึก Fibonacci แบบเรียกซ้ำโดยเพิ่มกลวิธี memoization

ตัวอย่างที่ 3 ปัญหา N-queen

ปัญหา N-queen กับการค้นหาคำตอบตามแนวลึกแบบเรียกซ้ำและการใช้กลวิธีค้นคำตอบด้วยวิธีย้อนรอย โดยจะเช็คเงื่อนไขที่จะลงควีนบนตารางให้ครบ N ช่อง หากลงตามเงื่อนไขไม่ได้ก็จะตรวจสอบในช่องถัดไปจนกว่าจะเจอช่องที่ลงได้ แต่หากไล่ตารางจนครบแล้วยังไม่สามารถลงมงกุฎทั้ง N ได้ครบก็จะไล่ย้อนกลับแก้ไขลงมงกุฎที่เคยลงไปแล้วเปลี่ยนไปลงที่ช่องใหม่ทำไปเรื่อย ๆ จนกว่าจะสามารถลงมงกุฎได้จนครบเกมก็จะยุติ

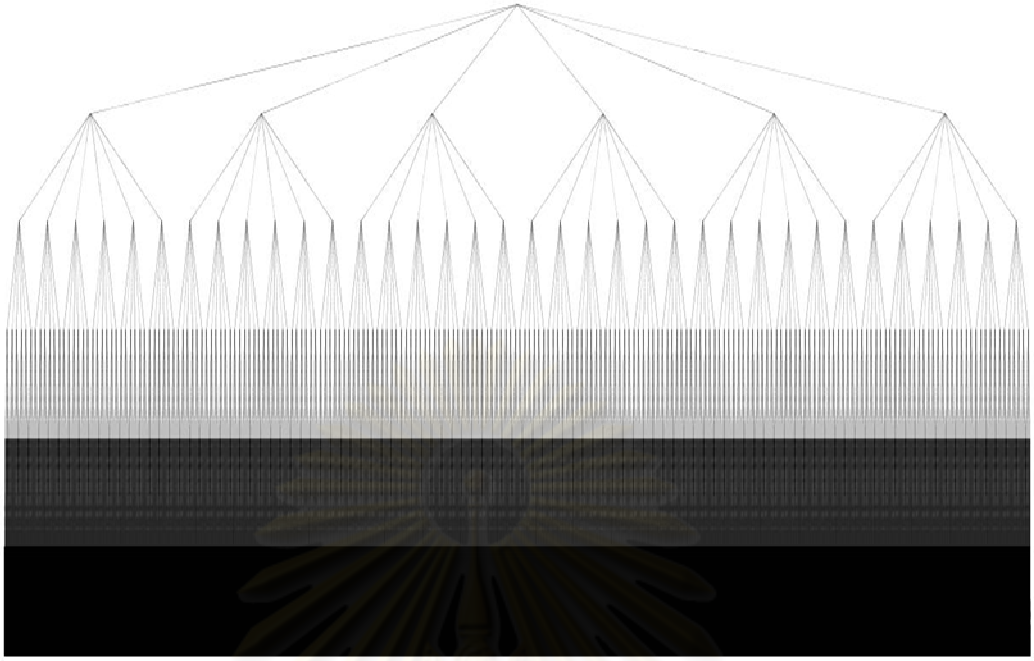
กติกาการเล่นเกม

ให้ผู้เล่นแก้ปัญหา N-Queens โดยการวางมงกุฎในตารางให้ครบ N ตามเงื่อนไขที่วางไว้โดยที่เงื่อนไขที่กำหนด โดยที่ลักษณะของปัญหาจะเป็นการวางตัว queen จำนวน N ตัวลงบนบอร์ดขนาด N x N ช่อง โดยมีกฎอยู่ 3 ข้อ คือ queen ที่วางลงไปนั้นจะต้องไม่ชนกันในแนวที่เป็นแถวเดียวกัน คอลัมน์เดียวกัน และในแนวเส้นทแยงมุม

```
public class Queens {
    public boolean isConsistent(int[] col, int n) {
        for (int i = 0; i < n; i++) {
            if (col[n] == col[i] ||
                Math.abs(col[n] - col[i]) == Math.abs(i - n)) {
                return false;
            }
        }
        return true;
    }
    public void queensRCS(int[] col, int i) {
        if (i == col.length) {
            printQueens(col);
        } else {
            for (int j = 0; j < col.length; j++) {
                col[i] = j;
                if (isConsistent(col, i)) {
                    queensRCS(col, i + 1);
                }
            }
        }
    }
    public void printQueens(int[] col) {
        for (int i = 0; i < col.length; i++) {
            for (int j = 0; j < col.length; j++) {
                if (col[i] == j) System.out.print("Q ");
                else   System.out.print("* ");
            }
            System.out.println();
        }
        System.out.println();
    }
}
```

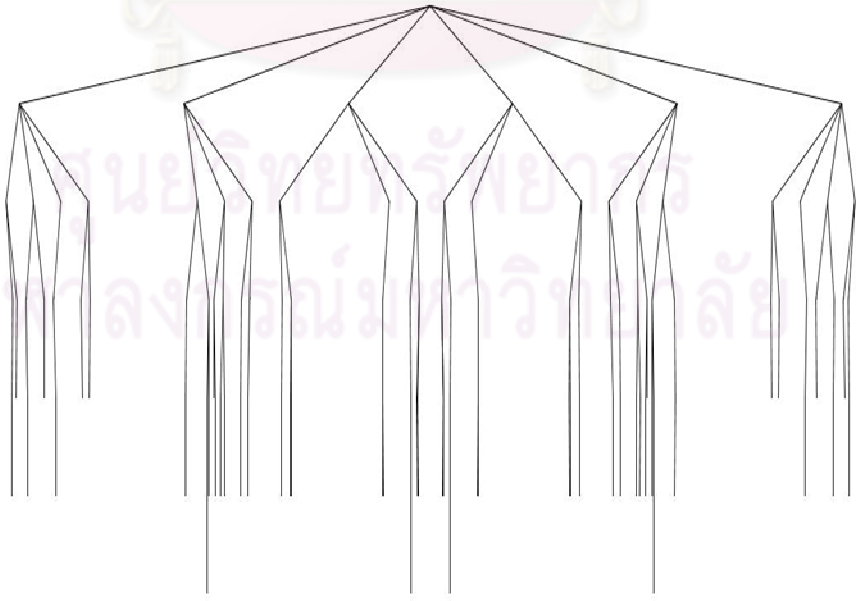
รหัสที่ 6.5 การค้นคำตอบแนวลึกเพื่อแก้ปัญหา N-queen เสริมด้วยกลวิธีย้อนรอย

จากรูปที่ 6.6 แสดงภาพต้นไม้ปริภูมิสถานะในการหาคำตอบปัญหา 6-Queens ด้วยการทำงานแบบเรียกซ้ำ จะเห็นได้ว่าใช้ปม 55,987 ปมในการค้นคำตอบ (ดูรหัสคำสั่งสมบูรณ์ได้ที่ภาคผนวก ค.3)



รูปที่ 6.6 การหาคำตอบปัญหา 6-Queens ด้วยเมท็อดเรียกซ้ำ

จากรูปที่ 6.7 ต้นไม้ปริภูมิสถานะสำหรับหาคำตอบ 6-Queens โดยเพิ่มกลวิธีย้อนรอยเพื่อการเปรียบเทียบกับการหาคำตอบแบบรูปที่ 6.6 ซึ่งจะเห็นได้ว่าปมนั้นลดลงอย่างมากจาก 55,987 ปมเหลือ 153 ปม (ดูรหัสคำสั่งสมบูรณ์ได้ที่ภาคผนวก ค.3)



รูปที่ 6.7 ต้นไม้ปริภูมิสถานะการหาคำตอบปัญหา 6-Queens โดยเพิ่มกลวิธีย้อนรอย

ตัวอย่างที่ 4 ปัญหาการเดินทางของพนักงานขาย

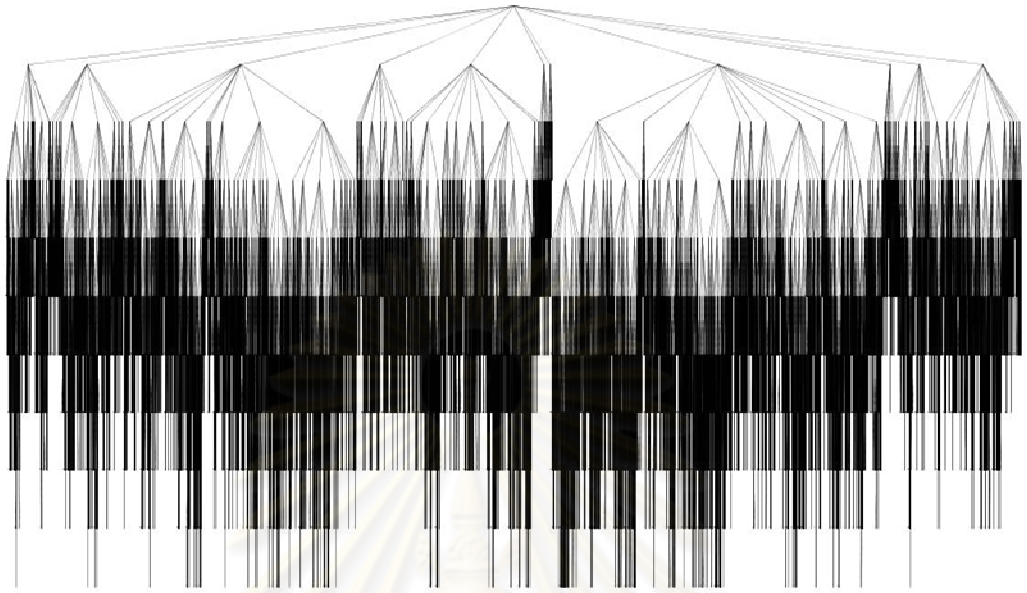
การหาเส้นทางเดินของของพนักงานขาย (Traveling Salesperson) กำหนดกราฟการเดินทางระหว่างเมือง 12 เมือง โดยระยะทางจากเมืองหนึ่งไปยังเมืองต่าง ๆ นั้นเป็นการสุ่มค่าระยะทางระหว่างสองเมืองใด ๆ คนขายของต้องการเดินทางจากเมืองหนึ่งไปอีก เมืองหนึ่งจนครบทุกเมือง เพียงครั้งเดียว แล้วย้อนกลับมาเมืองเริ่มต้น โดยให้มีระยะทางการเดินทั้งหมดสั้นที่สุด กำหนดว่าระยะทางทั้งหมดไม่เป็น จำนวนลบ

วิธีที่ 1 พิจารณาการค้นหาระยะทางที่ดีที่สุด (Best first search) ของการเดินทางผ่านเมืองหนึ่งไปยังเมืองใด ๆ ดังรหัสที่ 6.6 (สามารถดูรายละเอียดคำสั่งได้ที่ภาคผนวก ค 4 หรือที่มารหัสคำสั่งได้ที่ [23])

```
public void tsp(Graph g) {
    minTour = INFINITY; mark = new boolean[N + 1];
    minEdge = new double[g.size() + 1]; cost();
    PriorityQueue q = new PriorityQueue();
    Integer v1 = new Integer(1);
    TSPNode root = new TSPNode(null, v1);
    root.add(v1); q.add(root);
    while (!q.isEmpty()) {
        TSPNode temp = (TSPNode) q.poll();
        if (temp.bound < minTour) {
            Iterator itr = g.neighbors(temp.lastVertex);
            while (itr.hasNext()) {
                Object nextVert = itr.next();
                if (!temp.path.contains(nextVert)) {
                    TSPNode u = new TSPNode(temp, nextVert);
                    u.level = temp.level + 1;
                    u.length = temp.length +
length(temp.lastVertex,nextVert);
                    u.copyList(temp.path); u.add(nextVert);
                    if (u.level == g.size() - 2) {
                        Iterator ntr = g.neighbors(nextVert);
                        while (ntr.hasNext()) {
                            Object x = ntr.next();
                            if (!u.path.contains(x)) {
                                u.add(x);
                                u.length += length(nextVert, x);
                                u.add(u.path.get(0));
                                u.length += length(x, u.path.get(0));
                                if (u.length < minTour) {
                                    minTour = u.length;
                                    bestList = new Vector(u.path);
                                }
                                break;
                            }
                        }
                    } else {
                        if (u.length < minTour)
                            q.add(u);
                    }
                }
            }
        }
    }
}
```

รหัสที่ 6.6 การค้นคำตอบแบบดีที่สุดในปัญหา Traveling Salesperson

จากรูปที่ 6.8 แสดงภาพต้นไม้ปริภูมิสถานะการค้นคำตอบแบบดีที่สุดในปัญหาการเดินทางของพนักงานขายจำนวน 12 เมืองต้องใช้จำนวนปมคำตอบทั้งสิ้น 72915 ปม



รูปที่ 6.8 การค้นคำตอบแบบดีที่สุดในปัญหา Traveling Salesperson

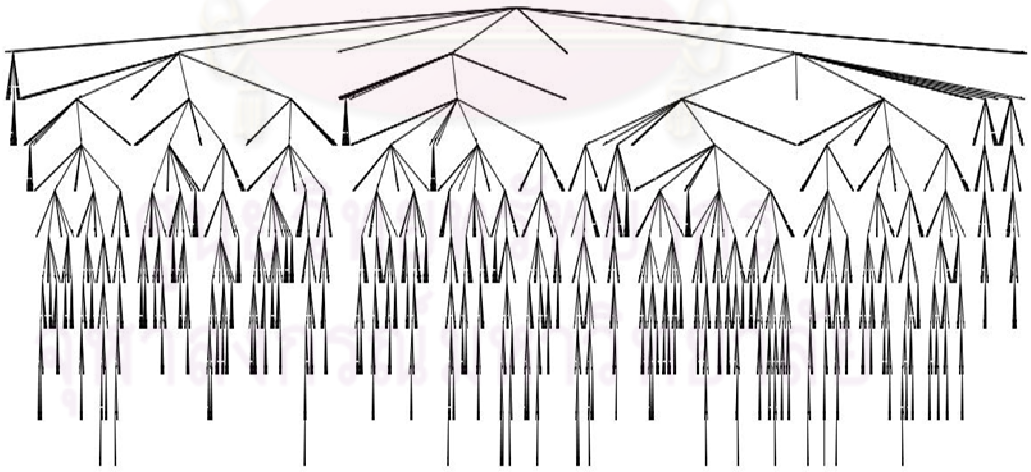
วิธีที่ 2 พิจารณา lower bound บนต้นทุนที่น้อยที่สุด (cost) ของระยะทางที่สั้นที่สุดจากเมืองหนึ่งไปยังเมืองใด ๆ โดยใช้การค้นหาที่ดีที่สุด (Best first search) กับกลวิธี Branch-and-bound ดังรหัสที่ 6.7 (สามารถดูรหัสคำสั่งสมบูรณ์ที่ภาคผนวก ค คำสั่งที่ ค.5 หรือที่มารหัสคำสั่งได้ที่ [23])

```
public void tsp(Graph g) {
    int N = g.size();
    minTour = INFINITY;
    mark = new boolean[N + 1];
    minEdge = new double[N + 1];
    cost();
    q = new PriorityQueue();
    Integer v1 = new Integer(1);
    TSPNode root = new TSPNode(null, v1);
    root.add(v1);
    root.bound = bound(root);
    q.add(root);
    while (!q.isEmpty()) {
        TSPNode temp = (TSPNode) q.poll();
        if (temp.bound < minTour) {
            Iterator itr = g.neighbors(temp.lastVertex);
            while (itr.hasNext()) {
                Object nextVert = itr.next();
                if (!temp.path.contains(nextVert)) {
                    TSPNode u = new TSPNode(temp, nextVert);
                    u.level = temp.level + 1;
                    u.length = temp.length +
length(temp.lastVertex,nextVert);
                    u.copyList(temp.path);
                    u.add(nextVert);
                    if (u.level == N - 2) {
```

```
                        Iterator ntr = g.neighbors(nextVert);
                        while (ntr.hasNext()) {
                            Object x = ntr.next();
                            if (!u.path.contains(x)) {
                                u.add(x);
                                u.length += length(nextVert, x);
                                u.add(u.path.get(0));
                                u.length += length(x, u.path.get(0));
                                if (u.length < minTour) {
                                    minTour = u.length;
                                    bestList = new Vector(u.path);
                                }
                                break;
                            }
                        }
                    } else {
                        u.bound = bound(u);
                        if (u.bound < minTour) {
                            q.add(u);
                        }
                    }
                }
            }
        }
    }
}
```

รหัสที่ 6.7 การเพิ่มกลวิธี Branch-and-bound ในปัญหา Traveling Salesperson

จากรูปที่ 6.9 การค้นคำตอบแบบดีที่สุดในปัญหาการเดินทางของพนักงานขายจำนวน 12 เมืองเมื่อเพิ่มกลวิธี Branch-and-bound สามารถลดจำนวนปมคำตอบจาก 72915 ปมเหลือเพียง 1550 ปม



รูปที่ 6.9 การเพิ่มกลวิธี Branch-and-bound แก้ปัญหา Traveling Salesperson

ตัวอย่างที่ 5 ปัญหาถุงเป้แบบ 0/1

ปัญหา 0-1 Knapsack หรือ ปัญหาถุงเป้แบบ 0/1 เป็นการเลือกหยิบของใส่ถุงเป้โดยมีมูลค่ารวมสูงสุดโดยที่ถุงเป้ไม่ขาด ซึ่งมีการกำหนดให้มีของอยู่จำนวน n ชิ้น ชิ้นที่ i หนัก W_i และมีมูลค่า V_i และถุงนั้นรับน้ำหนักได้ไม่เกิน W

ตัวอย่าง กำหนดให้มีของ 17 ชิ้น ซึ่งมีมูลค่าและน้ำหนักดังแสดงข้างล่างนี้ และถุงที่ใช้ของรับน้ำหนักได้มากที่สุด 24 มีดังนี้

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
W_i	*3*	*2*	*4*	*3*	*2*	*3*	*5*	*4*	*3*	*6*	*2*	*10*	*6*	*12*	*10*	*5*	
V_i	*24*	*14*	*26*	*19*	*12*	*17*	*25*	*18*	*13*	*24*	*6*	*12*	*7*	*14*	*10*	*5*	*W=24*
Vi/W_i	*8*	*7*	*6.5*	*6.33*	*6*	*5.57*	*5*	*4.5*	*4.33*	*4*	*3*	*1.2*	*1.16*	*1.16*	*1*	*1*	

สำหรับผลเฉลยของปัญหานี้ที่สามารถรับน้ำหนักได้มากที่สุด 24 คือเซตของ {1, 2, 3, 4, 5, 6, 8, 9} ซึ่งได้มูลค่ารวมของสูงสุด 24+14+26+19+12+17+18+13 = 143 ชิ้น

i	1	2	3	4	5	6	8	9
W_i	*3*	*2*	*4*	*3*	*2*	*3*	*4*	*3*
V_i	*24*	*14*	*26*	*19*	*12*	*17*	*18*	*13*

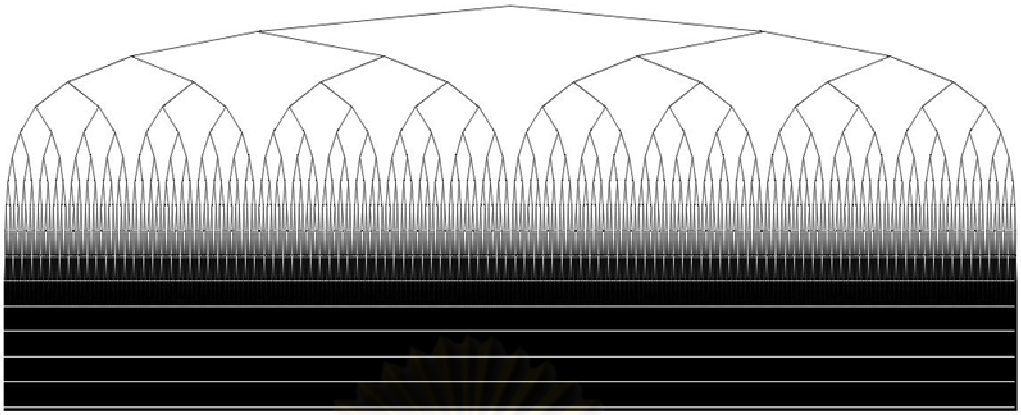
โดยปัญหานี้สามารถทดลองแก้ปัญหาด้วยวิธีการต่าง ๆ เพื่อแสดงภาพการค้นคำตอบดังนี้

วิธีที่ 1 ใช้การลุยทุกผลเฉลย (brute force) เพื่อวิ่งค้นหาทั้งต้นไม้โดยนำมูลค่ารวมมาเปรียบเทียบให้ได้มูลค่ารวมสูงสุดที่ถุงเป้ไม่ขาด จากรหัสคำสั่งนั้นจะเรียกเมท็อดเรียกซ้ำ วนหามูลค่ารวม V_i ที่รับน้ำหนักได้ไม่เกิน W ดังรหัสที่ 6.9 (สามารถดูรหัสคำสั่งสมบูรณ์ที่ภาคผนวก ค รหัสคำสั่งที่ ค-6 หรือที่มารหัสคำสั่งได้ที่ [23])

```
public void knapsack(int index, double weight, double profit, List cList) {
    if (weight <= W && profit > maxProfit) {
        maxProfit = profit;
        bestList = new LinkedList(cList);
    }
    List leftList = new LinkedList(cList);
    leftList.add(new Integer(index + 1));
    knapsack(index + 1, weight + w[index + 1], profit + p[index + 1],
leftList);
    List rightList = new LinkedList(cList);
    knapsack(index + 1, weight, profit, rightList);
}
```

รหัสที่ 6.8 การค้นหาคำตอบแนวลึกปัญหาถุงเป้ 0/1

รูปที่ 6.10 จากการใช้การลุยทุกผลเฉลย (brute force) นั้นพบว่าต้องใช้จำนวนปมถึง 131,071 ปมจึงสามารถตรวจสอบได้ทุกผลเฉลย



รูปที่ 6.10 ต้นไม้ปริภูมิสถานะการค้นคำตอบโดยการลุยทุกผลเฉลยปัญหาถุงเป้ 0/1

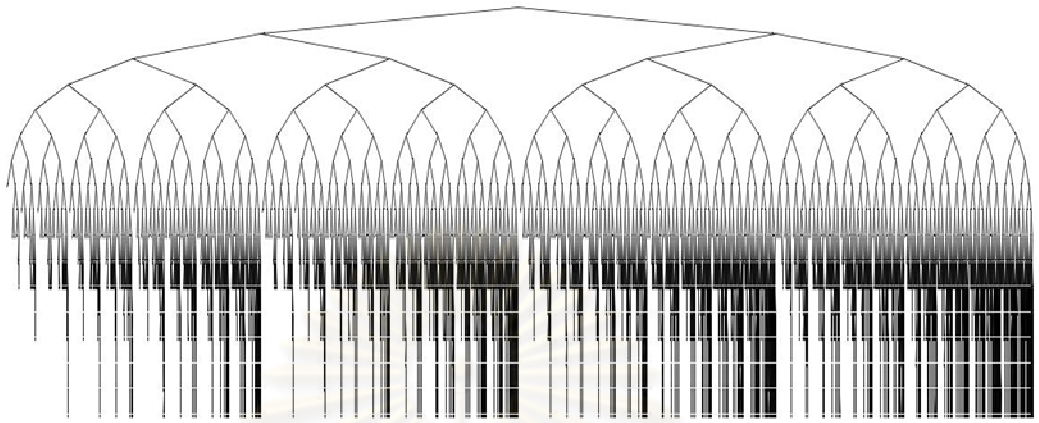
วิธีที่ 2 ใช้กลวิธีการย้อนรอย (Backtracking Algorithm) โดย น้ำหนักมาพิจารณาว่าการขยายผลเฉลยจาก (x_1, x2, x3, x4,…,x_k) เป็น (x_1, x2, x3, x4,…,x_k ,x_{k+1}) นั้นมีแววที่รับน้ำหนักไหวหรือไม่ คือ weight <= W เพื่อช่วยเล็มปมออกบางส่วนทิ้ง และนำผลรวมมูลค่าในแต่ละผลเฉลยมาเปรียบเทียบให้ได้มูลค่าสูงสุดดังแสดงด้วยรหัสเทียมดังรหัสที่ 6.9 (สามารถดูรหัสคำสั่งสมบูรณ์ที่ภาคผนวก ค รหัสคำสั่งที่ ค.7 หรือที่มารหัสคำสั่งได้ที่ [23])

```
public void knapsack(int index, double weight, double profit, List cList) {
    if (weight <= W && profit > maxProfit) {
        maxProfit = profit;
        bestList = new LinkedList(cList);
    }
    if (promising(index, weight)) {
        List leftList = new LinkedList(cList);
        leftList.add(new Integer(index + 1));
        knapsack(index + 1,weight + w[index + 1],profit + p[index +
1],leftList);
        List rightList = new LinkedList(cList);
        knapsack(index + 1, weight, profit, rightList);
    }
}
public static boolean promising(int item, int weight) {
    int k = item+1 ;
    int totalSize = weight;
    if (weight > W) {
        return false;
    }
    return totalSize+s[k] <= W;
}
```

รหัสที่ 6.9 การค้นหาคำตอบแนวลึกปัญหาถุงเป้ 0/1 โดยเพิ่มกลวิธีย้อนรอย

จากรหัสคำสั่ง 6.9 เมท็อด knapsack รับผลเฉลยมูลค่ารวม (profit) ที่มีแววที่น้ำหนักไม่เกินที่กำหนด ถ้ามูลค่ารวมมากกว่าผลเฉลยที่เคยพบมา พร้อมทั้งจำผลเฉลยนี้ไว้ จากนั้นค้นต่อตามแนวลึก แต่ก่อนที่จะลงไปต่อนั้นก็ตรวจสอบก่อนว่าผลเฉลยที่ขยายนั้นมีแววหรือไม่ ด้วยเมท็อด promise เมื่อแสดงภาพด้วยระบบแสดงภาพปริภูมิสถานะแล้วจะเห็นได้ว่าจำนวนสถานะ

ลดลงอย่างมากเมื่อเทียบกับการลุยผลเฉลยทั้งหมดจาก 131,071ปม เหลือ 4,064 ปมดังแสดง รูปที่ 6.11



รูปที่ 6.11 การค้นคำตอบโดยกลวิธีการย้อนรอยปัญหาถุงเป้ 0/1

วิธีที่ 3 การค้นคำตอบตามแนวกว้าง (Breath first search) ซึ่งจะเริ่มต้นด้วยการเรียงลำดับสิ่งของตามมูลค่าต่อน้ำหนักจากมากไปหาน้อย โดยนำมูลค่าต่อน้ำหนักของของชิ้นนั้นพิจารณา และใช้กลวิธีการขยายและจำกัดเขต (branch and bound) ที่ upper bound กำหนดฟังก์ชันการทำงานด้วยอัลกอลิทึมเชิงละโมบ เพื่อเฉือนของออกไปบางส่วน มาเป็นขอบเขตบนของมูลค่ารวมมาใช้กำกับปมต่าง ๆ ในต้นไม้ โดยรหัสคำสั่งที่ 6.10 แสดงการค้นคำตอบตามแนวกว้าง จากนั้นเพิ่มกลวิธีการขยายและจำกัดเขตเพื่อเฉือนปมออกไปบางส่วนดังแสดงในรหัสคำสั่งที่ 6.11

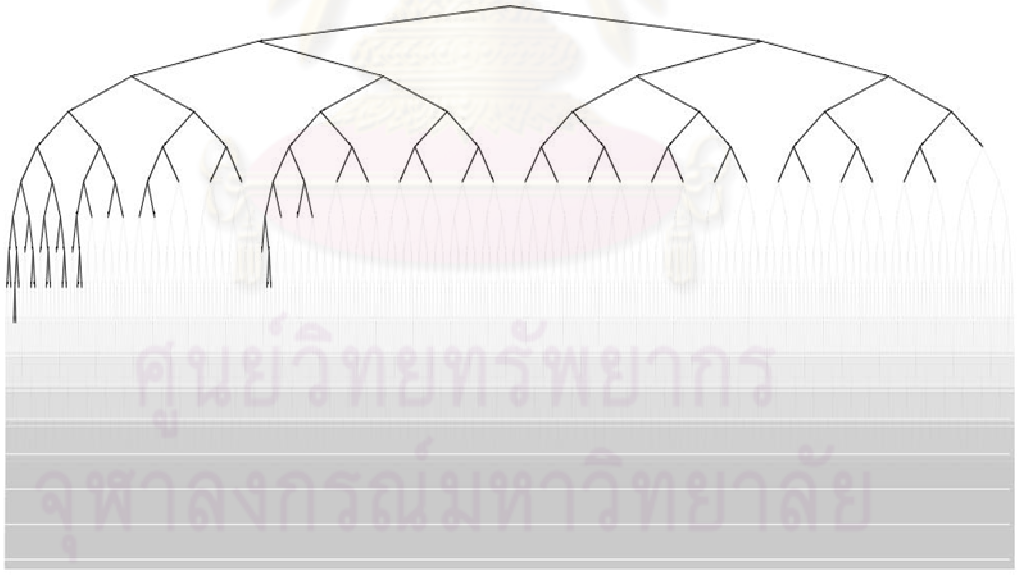
```
public static int knapsack(int n, int[] p, int[]w, int W){
    Queue Q;
    node u,v;
    int maxprofit;
    Q = new LinkedList();
    v.level=0; v.profit=0; v.weight=0;
    maxprofit=0;
    enqueue(Q,v);
    while(!empty(Q)){
        dequeue(Q,v);
        u.level=v.level+1;
        u.weight=v.weight+w[u.level];
        u.profit=v.profit+p[u.level];
        if(u.weight <= W && u.profit > maxprofit)
            maxprofit=u.profit;
            if(bound(u)>maxprofit)
                enqueue(Q,u);
                u.weight=v.weight
                u.profit=v.profit;
                if(bound(u)>maxprofit)
                    enqueue(Q,u);
    }
    return maxprofit;
}
```

รหัสที่ 6.10 การค้นคำตอบแนวกว้างในปัญหาถุงเป้ 0/1

```
public static float bound(){
    index j,k;
    int totweight;
    float result;
    if(u.weight>=W)
        return 0;
    else{
        result=u.profit;
        j=u.level+1;
        totweight=u.lenght;
        while(j<=n && totweight+w[j]<=W){
            totweight= totweight+w[j];
            result=result + p[j];
            j++;
        }
        k=j;
        if(k<=n)
            result = result + (W-totweight)*p[k]/w[k];
        return result;
    }
}
```

รหัสที่ 6.11 ฟังก์ชันหาขอบเขตบนของมูลค่ารวมในปัญหาถุงเป้ 0/1

จากรูปที่ 6.12 จะเป็นได้ว่าเมื่อใช้วิธีดังกล่าวจำนวนปมของผลเฉลยนั้นลดลงอย่างมาก (หาคำตอบได้เร็วขึ้นมาก) เมื่อเทียบกับการลุยผลเฉลยทั้งหมดจาก 131071 ปมเหลือเพียง 103 ปม (สามารถดูรหัสคำสั่งสมบูรณ์ที่ภาคผนวก ค คำสั่งที่ ค.8)



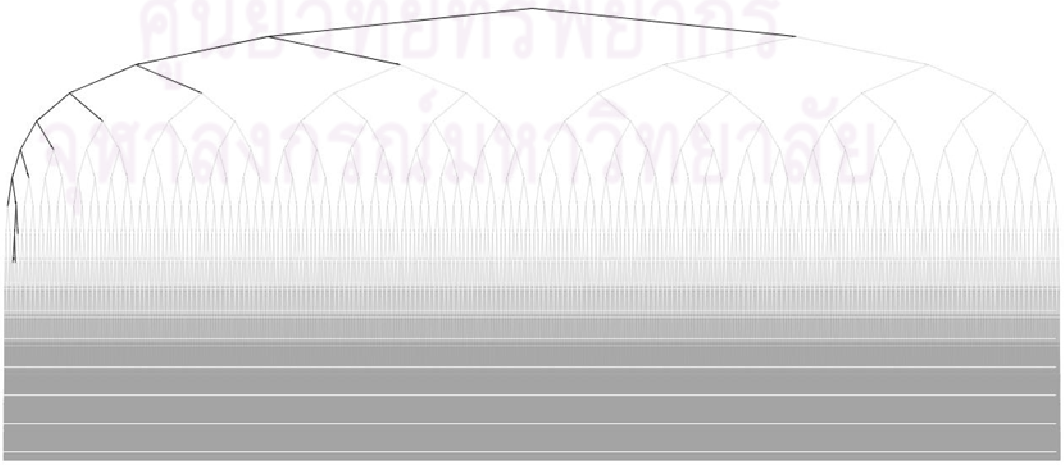
รูปที่ 6.12 การเพิ่มกลวิธีการขยายและจำกัดเขตปัญหาถุงเป้ 0/1 เหลือ 103 ปม

วิธีที่ 4 ใช้กลวิธีตามแนวคิดวิธีที่ 3 แต่เปลี่ยนการค้นหาคำตอบตามแนวกว้างด้วยการค้นแบบดีที่สุด (Best First Search)

```
public static int knapsack(int n, int[]p, int[]w,int W){
    PriorityQueue Q;
    node u,v;
    int maxprofit;
    Q = new PriorityQueue();
    v.level=0; v.profit=0; v.weight=0;
    maxprofit=0;
    v.bound=bound(v);
    enqueue(Q,v);
    while(!empty(Q)){
        remove(Q,v);
        if(v.profit > maxprofit){
            u.level=v.level+1;
            u.weight=v.weight+w[u.level];
            u.profit=v.profit+p[u.level];
            if(u.weight <= W && u.profit > maxprofit){
                maxprofit=u.profit;
                u.bound=bound(u);
                if(u.bound maxprofit)
                    insert(Q,u);
                u.weight=v.weight;
                u.profit=v.profit;
                u.bound=bound(u);
                if(u.bound maxprofit)
                    insert(Q,u);
            }
        }
    }
    return maxprofit;
}
```

รหัสที่ 6.12 การค้นคำตอบแบบดีที่สุดในปัญหาถุงเป้ 0/1

จากแนวคิดวิธีที่ 3 เมื่อลองปรับเปลี่ยนจากการค้นคำตอบตามแนวกว้างมาเป็นแบบดีที่สุดนั้นจะเห็นได้ว่าจำนวนปมของผลเฉลยลดลงมาเหลือเพียง 19 ปมเท่านั้นดังแสดงในรูป 6.13 (สามารถดูรหัสคำสั่งสมบูรณ์ที่ภาคผนวก ค คำสั่งที่ ค.9)



รูปที่ 6.13 การค้นคำตอบที่ดีที่สุดโดยกลวิธีการขยายและจำกัดเขตในปัญหาถุงเป้ 0/1

ตัวอย่างที่ 6 การแสดงภาพผลเฉลยในปัญหา 4 Queen

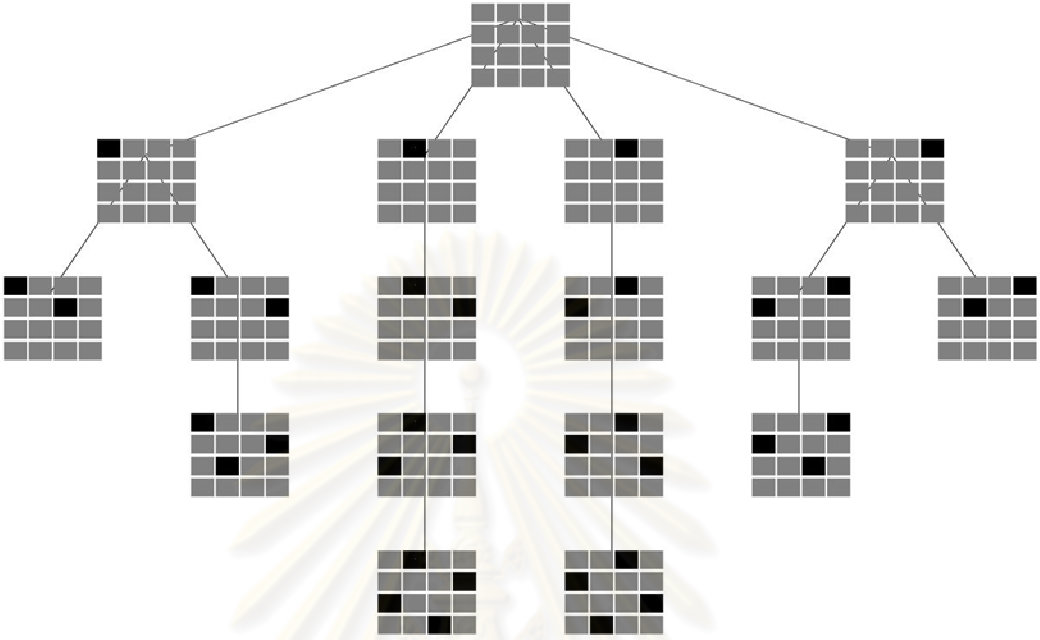
ผู้ใช้สามารถแจงผลเฉลยด้วยการนำเสนอด้วยรูปภาพเพื่อเสริมความเข้าใจของการทำงานในแต่และผลเฉลย ตัวอย่างนี้นั้นจะเสนอการแสดงรูปภาพการแจงผลเฉลยในปัญหา 4 Queen

สำหรับการนำเสนอด้วยรูปภาพบนปมแต่ละปมนั้นสามารถทำได้ง่ายด้วยการ Override เมท็อด `drawState` จากการขยายคลาสแม่ `VState` จากนั้นก็สามารถวาดภาพเสริมความเข้าใจในแต่ละปมปัญหาที่เกิดขึ้นขณะค้นคำตอบได้ดังแสดงรหัสตัวอย่างการ Override เมท็อด `drawState` ดังรหัสคำสั่งที่ 6.13

```
import java.awt.*;
import java.util.List;
import jstate101.*;
public class Queens extends VDrawable{
    public static void main(String[] args) {
        VEngine.begin(new Queens());
    }
    @VMain(recursiveMethod = "queensRCS")
    public static void mainRCS() {
        int[] col = new int[4];
        for(int i = 0; i < col.length; i++)
            col[i] = -1;
        queensRCS(col, 0);
    }
    public void queensRCS(int[] col, int i){
        if(i == col.length) {
             printQueens(col);
        } else {
            for (int j = 0; j < col.length; j++) {
                col[i] = j;
                queensRCS(col, i + 1);
            }
        }
    }
    public boolean isConsistent(int[] col, int n) {
        for (int i = 0; i < n; i++)
        if (col[n] == col[i] || Math.abs(col[n] - col[i]) == Math.abs(i - n))
                return false;
        return true;
    }
    @Override
    public void drawState(Graphics2D g2d, int x, int y, List args) {
        g2d.setColor(Color.black);
        int x_pos = x - 25;
        int y_pos = y - 30;
        int col[] = (int[]) args.get(0);
        for (int i = 0; i < col.length; i++) {
            for (int j = 0; j < col.length; j++) {
                if (j % col.length == 0) {
                    x_pos = x - 40; y_pos += 29;
                } else {
                    x_pos += 27;
                }
                if (col[i] == j) g2d.draw3DRect(x_pos,y_pos-40,25,25,true);
                else g2d.fillRect(x_pos, y_pos - 40, 25, 25);
            }
        }
    }
    public void printQueens(int[] col) { . . . }
}
```

รหัสที่ 6.13 การ Override เมท็อด `drawState` วาดปมบนโปรแกรมแบบเรียกซ้ำ

เมื่อเพิ่มรหัสคำสั่งเข้าไปจะได้ภาพผลเฉลยในปัญหา 4 Queen ด้วยการค้นแบบแนวลึกและกลวิธีย้อนรอยเพื่อเสริมความเข้าใจด้วยภาพดังรูปที่ 6.14



รูปที่ 6.14 ผลเฉลยในปัญหา 4 Queen ด้วยการค้นตามแนวลึกและกลวิธีย้อนรอย

ตัวอย่างที่ 7 การแสดงภาพผลเฉลยในปัญหา 8 Puzzle

การเขียนโปรแกรมแบบอ็อบเจกต์นั้นผู้เขียนโปรแกรมสามารถแทรกรูปภาพลงบนปมสถานะได้เพียงแค่ Override เมท็อด `drawState` ในคลาสสถานะที่ขยายคลาสแม่ `VState` ลงไป จากนั้นก็สามารถวาดภาพเสริมความเข้าใจในแต่ละปมปัญหาที่เกิดขึ้นขณะค้นคำตอบได้ดังแสดงรหัสตัวอย่างการ Override เมท็อด `drawState` ดังรหัสคำสั่งที่ 6.14

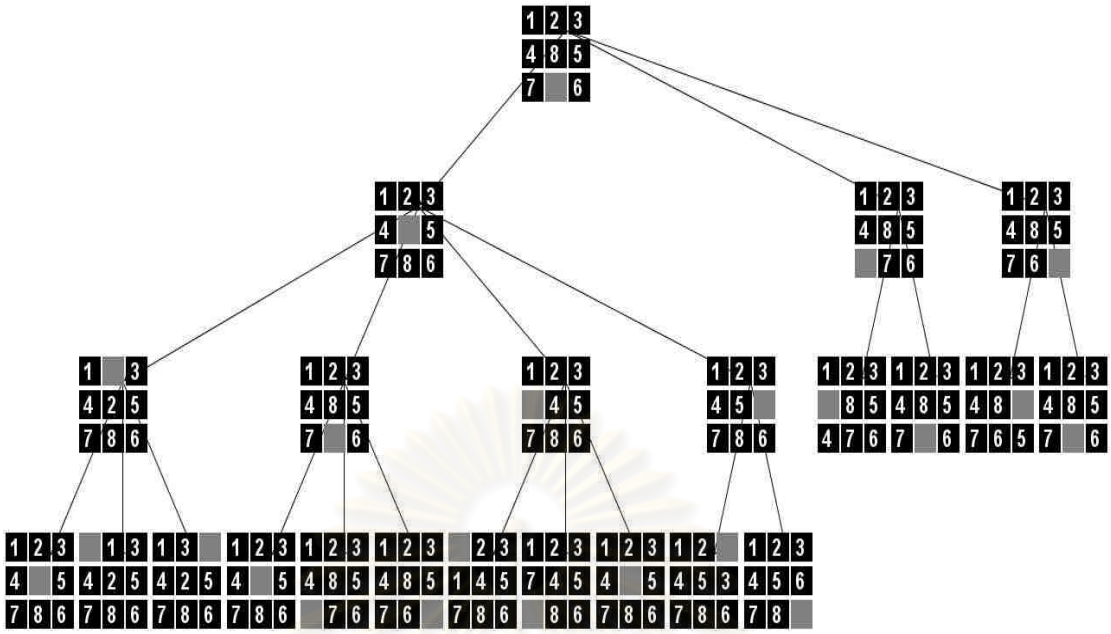
```
import java.util.*;
import jstate.visualizer.configure.VMain;
import jstate.visualizer.engine.VEngine;
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics2D;
import java.awt.Shape;
import java.awt.geom.Rectangle2D;
import jstate.visualizer.configure.VState;
public class EightPuzzle {
    Map<String, Integer> map = new HashMap<String, Integer>();
    Set set = new HashSet();
    Queue<Node> queue = new LinkedList<Node>();
    Node root;
    Node node;
    String goal_state = "123456780", init_state = "123485706";

    @VMain(stateClass = "demo.old.puzzle.Node")
    public void runBFS() {
        EightPuzzleBFS();
    }

    public static void main(String args[]) {
        VEngine.begin(new EightPuzzle());
    }

    public void EightPuzzleBFS() {
        root = new Node(init_state, null);
        queue.add(root);
        set.add(root);
        while (queue.peek() != null) {
            node = (Node) queue.remove();
            if (init_state.equals(goal_state)) {
                break;
            } else {
                up(node.state);
            }
            if (init_state.equals(goal_state)) {
                break;
            } else {
               down(node.state);
            }
            if (init_state.equals(goal_state)) {
                break;
            } else {
                left(node.state);

            }
            if (init_state.equals(goal_state)) {
                break;
            } else {
                right(node.state);
            }
        }
    }
```

```
    void up(String str) {
        int a = str.indexOf("0");
        if (a > 2) {
            String s = str.substring(0, a - 3) + "0" + str.substring(a - 2,
                        a) + str.charAt(a - 3) + str.substring(a + 1);
            Node v1 = new Node(s, node);
            if (!set.contains(v1)) {
                queue.add(v1);
                set.add(v1);
            }
            if (s.equals(goal_state)) {
                this.init_state = s;
            }
        }
    }

    void down(String str) {
        int a = str.indexOf("0");
        if (a < 6) {
            System.out.println("down" + str);
            String s = str.substring(0, a) + str.substring(a + 3, a + 4) +
                   str.substring(a + 1, a + 3) + "0" + str.substring(a + 4);
            Node v1 = new Node(s, node);
            if (!set.contains(v1)) {
                queue.add(v1);
                set.add(v1);
            }
            if (s.equals(goal_state)) {
                this.init_state = s;
            }
        }
    }

    void left(String str) {
        int a = str.indexOf("0");
        if (a != 0 && a != 3 && a != 6) {
            System.out.println("left" + str);
            String s = str.substring(0, a - 1) + "0" + str.charAt(a - 1) +
                        str.substring(a + 1);
            Node v1 = new Node(s, node);
            if (!set.contains(v1)) {
                queue.add(v1);
                set.add(v1);
            }
            if (s.equals(goal_state)) {
                this.init_state = s;
            }
        }
    }

    void right(String str) {
        int a = str.indexOf("0");
        if (a != 2 && a != 5 && a != 8) {
            System.out.println("right" + str);
            String s = str.substring(0, a) + str.charAt(a + 1) + "0" +
                        str.substring(a + 2);
            Node v1 = new Node(s, node);
            if (!set.contains(v1)) {
                queue.add(v1);
                set.add(v1);
            }
            if (s.equals(goal_state)) {
                this.init_state = s;
            }
        }
    }
    public class Node extends VState {
```

```
    public String state;
       Node parent;

       public Node(String startState, Node node) {
        state = startState;
        parent = node;
        VState.fireEvent(this);
       }

       @Override
       public void drawState(Graphics2D g, int x, int y) {
           int x_pos = x;
           int y_pos = y;
           for (int l = 0; l < state.length(); l++) {
               if (l % 3 == 0) {
                   x_pos = x - 40;
                   y_pos += 29;
               } else {
                   x_pos += 27;
               }

               if (state.substring(l, l + 1).equals("0")) {
                   g.setColor(Color.gray);
                   g.fillRect(x_pos, y_pos-40, 25,25);
               } else {
                   g.setColor(Color.BLACK);
                   g.fillRect(x_pos, y_pos-40, 25,25);
                   g.setColor(Color.WHITE);
                   g.setFont(new Font("TimesRoman", Font.BOLD, 20));
                   g.drawString(state.substring(l, l + 1), x_pos+5
                                , y_pos-20);
               }

           }
           Rectangle2D.Double square = new Rectangle2D.Double(x + 5
                                                              , y + 2, 0, 0);
       }

       @Override
       public VState getParent() {
           return parent;
       }
   }
}
```

รหัสที่ 6.14 การ Override เมท็อด `drawState` วาดปมบนโปรแกรมแบบวงวนทำซ้ำ

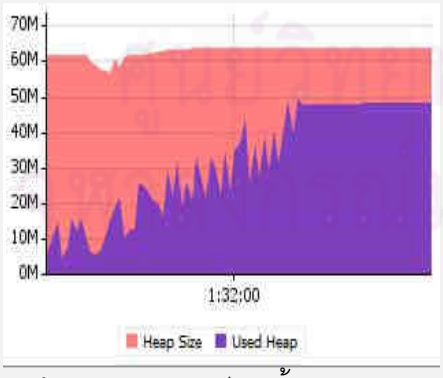
เมื่อเพิ่มรหัสคำสั่งเข้าไปจะได้ภาพผลเฉลยในปัญหา 8 Puzzle ตามแนวกว้างในการเขียนโปรแกรมแบบอ็อบเจกต์เพื่อเสริมความเข้าใจด้วยภาพดังรูปที่ 6.15



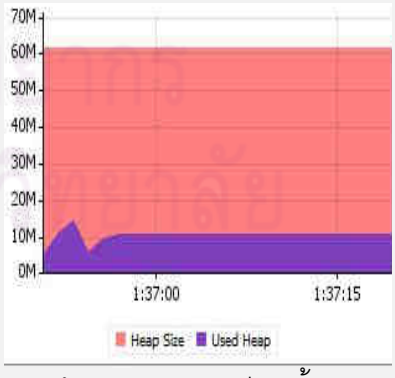
รูปที่ 6.15 ผลเฉลยของปัญหา 8 Puzzle ตามแนวกว้าง

6.2 ผลการทดลองวัดปริมาณหน่วยความจำที่ใช้ในการทำงาน

งานวิจัยนี้ได้พัฒนาเครื่องมือที่แสดงให้เห็นภาพการเปลี่ยนแปลงระหว่างการค้นคำตอบในปริภูมิสถานะ จากตัวอย่างที่ผ่านมาเมื่อนำมาใช้กับกลวิธีลดปมคำตอบในแบบต่าง ๆ จะเห็นว่าจำนวนปมที่ใช้ในการค้นคำตอบลดลงจำนวนมาก เมื่อนำโปรแกรมจากตัวอย่างการค้นคำตอบในปริภูมิสถานะตัวอย่างข้างต้นมาทดสอบกับ VisualVM Profiler [24] เพื่อแสดงให้เห็นถึงหน่วยความจำที่ลดลงเมื่อเสริมกลวิธีลดจำนวนปมลงไปในโปรแกรมค้นคำตอบ ซึ่งสอดคล้องการรูปภาพที่แสดงด้วยเครื่องมือในงานวิจัยนี้



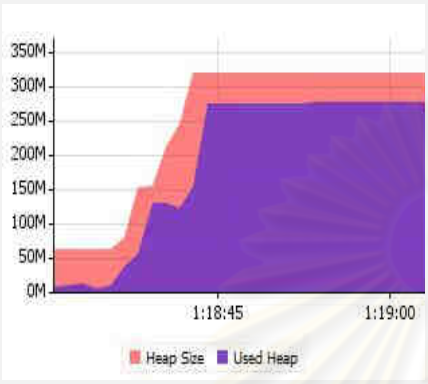
(ก) การค้นคำตอบแบบเรียกซ้ำจำนว 35421 ปม



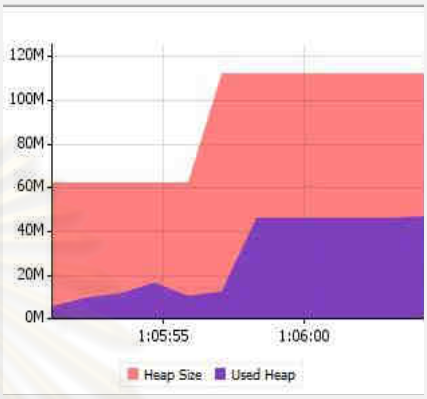
(ข) การค้นคำตอบแบบเรียกซ้ำและเสริมกลวิธี Memoization จำนวน 41 ปม

รูปที่ 6.16 ทดสอบหน่วยความจำที่ใช้ในการปัญหา Fibonacci

รูปที่ 6.16 กราฟทางซ้ายมือแสดงหน่วยความจำ(Used Heap)ที่ใช้ขณะโปรแกรม Fibonacci แบบเรียกซ้ำทำงาน เมื่อเปรียบเทียบกราฟหมายเลข (ก) จะเห็นได้ว่าหากเสริมด้วยกลวิธี Memoization หมายเลข (ข) ทำให้หน่วยความจำที่ใช้ขณะโปรแกรมค้นคำตอบลดลงซึ่งสอดคล้องการรูปภาพที่ 6.4 และ 6.5 แสดงด้วยเครื่องมือในงานวิจัยนี้



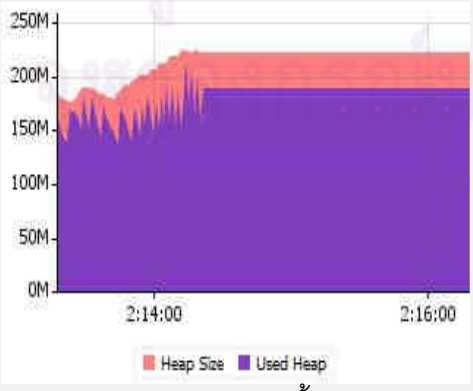
(ก) การค้นคำตอบด้วย Best first search จำนวน 72915 ปม



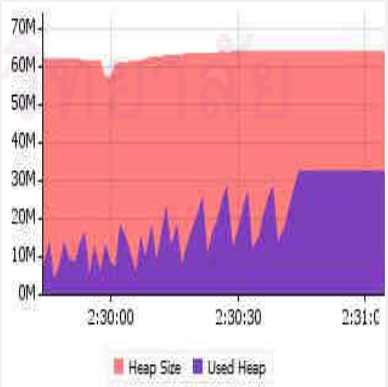
(ข)การค้นคำตอบด้วย Best first search และเสริมกลวิธี Branch and Bound จำนวน 1550 ปม

รูปที่ 6.17 ทดสอบหน่วยความจำที่ใช้ในการปัญหา Travelling salesman problem

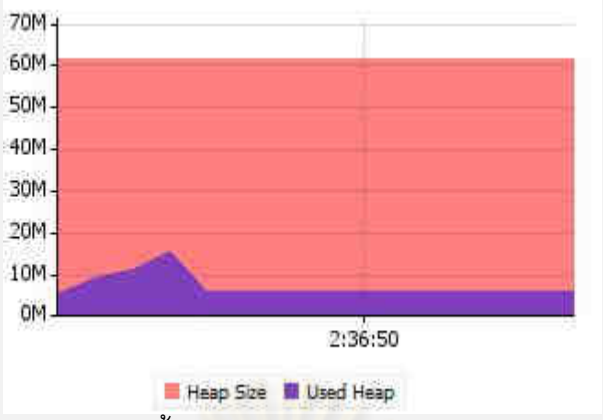
รูปที่ 6.17 กราฟทางซ้ายมือแสดงหน่วยความจำ(Used Heap)ที่ใช้ขณะโปรแกรมค้นคำตอบด้วย Best first search (กราฟหมายเลข ก)ทำงาน เมื่อเสริมกลด้วยวิธี Branch and Bound (กราฟหมายเลข ข)เห็นได้ว่าหน่วยความจำที่ใช้ขณะโปรแกรมค้นคำตอบลดลงจากเดิมอย่างมากซึ่งสอดคล้องการรูปภาพที่ 6.8 และรูปภาพที่ 6.9 แสดงด้วยเครื่องมือในงานวิจัยนี้



(ก) การค้นคำตอบแบบเรียกซ้ำจำนวน 131071 ปม



(ข)การค้นคำตอบแบบเรียกซ้ำและเสริมกลวิธี Back Tracking จำนวน 4064 ปม



(ค) การค้นคำตอบแบบเรียกซ้ำและเสริมกลวิธี Branch and Boundจำนวน 103 ปม

รูปที่ 6.18 แสดงกราฟการทดสอบหน่วยความจำที่ใช้ในการปัญหา Knapsack 0/1

รูปที่ 6.18 ทดสอบหน่วยความจำขณะโปรแกรม 0/1 Knapsack ทำงาน จะเห็นได้ว่ากราฟหมายเลข ก ใช้หน่วยความจำมากกว่ากราฟหมายเลข ข ที่ถูกเสริมด้วยกลวิธี Bracktracking และเมื่อเปลี่ยนกลวิธีการค้นคำตอบด้วย Branch and Bound (กราฟหมายเลข ค) หน่วยความจำที่ใช้ขณะโปรแกรมค้นคำตอบนั้นลดลงอย่างมาก

บทที่ 7

สรุปผลการวิจัยและข้อเสนอแนะ

7.1 สรุปผลการวิจัย

งานวิจัยนี้แสดงให้เห็นภาพการเปลี่ยนแปลงของต้นไม้ปริภูมิสถานะระหว่างการค้นคำตอบ ย่อมนำมาซึ่งความเข้าใจในพฤติกรรมของอัลกอริทึมการค้นคำตอบได้ดีขึ้น อันเป็นแรงจูงใจของการออกแบบและพัฒนาระบบ JSTATE101 ที่รองรับโปรแกรมการค้นคำตอบที่เขียนแบบเรียกซ้ำ และแบบที่ตัวสถานะเป็นอ็อบเจกต์ที่มีการสร้างและเก็บระหว่างการค้นด้วยวงวนทำซ้ำ การปรับโปรแกรมค้นคำตอบให้ใช้งานกับระบบนั้นกระทำได้ด้วยการเพิ่มรหัสคำสั่งกำกับเพียงเล็กน้อยในโปรแกรม โดยสามารถแสดงปมของต้นไม้ด้วยข้อความหรือภาพได้ JSTATE101 ใช้กลไกมาตรฐานของระบบจาวาสำหรับติดตามการเปลี่ยนแปลงสถานะ การทดลองใช้งานพบว่าระบบช่วยเสริมความเข้าใจการศึกษาอัลกอริทึมการค้นคำตอบได้

ระบบการแสดงภาพต้นไม้ปริภูมิสถานะเสริมการสอนวิชาอัลกอริทึม ในหัวข้ออัลกอริทึมการค้นในปริภูมิสถานะ พบว่า ระบบดังกล่าวอำนวยความสะดวกในการนำเสนอ ช่วยให้ผู้เรียนเห็นภาพ เห็นการเปลี่ยนแปลงของต้นไม้ปริภูมิสถานะระหว่างการทำงาน และยังสามารถทดลองเปลี่ยนกรรมวิธีการค้นได้อย่างหลากหลาย เช่น เพิ่มพฤติกรรมสุ่มในการค้น หรือเติมสามัญสำนึกในการเลือกปมระหว่างการค้น เพื่อให้เข้าใจในข้อดีข้อเสียในแง่ของเนื้อที่และเวลาในการค้น นอกจากนี้ผู้เขียนยังใช้ในการแสดงให้เห็นถึงเหตุผลของการทำงานช้าของอัลกอริทึมแบบ Divide and conquer ที่มีปัญหาย่อยที่ซ้ำซ้อนและซ้อนเหลื่อมกันมากมาย (เช่น ปัญหา matrix-chain multiplication [25]) เมื่อนำแนวคิดของ memoization เข้ามาเสริม จะได้โปรแกรมที่ทำงานเร็วขึ้นอย่างฉับพลัน การแสดง call tree ประกอบการนำเสนอส่งผลให้ผู้เรียนเข้าใจเหตุผลของความช้าความเร็วของการทำงาน จากการใช้งานระบบเพื่อเสริมการสอนพบว่า ระบบสามารถรองรับปริภูมิสถานะในระดับแสนปมได้อย่างราบรื่นบน jvm ที่ตั้งขนาดของ heap ไว้ที่ 512 ล้านไบต์ โดยโปรแกรมนี้มีขนาดไฟล์เพียง 132 kb

7.2 ข้อจำกัดของระบบ

1. กรณีที่ผู้เขียนโปรแกรมค้นคำตอบแบบเรียกซ้ำเมท็อดต้องการวาดรูปภาพกำกับปมสถานะ และใช้บริการลิสต์อ่านค่ารับเข้าเมท็อดเรียกซ้ำ ระบบแสดงภาพติดตามค่ารับเข้าเมท็อดเรียกซ้ำได้เฉพาะค่าพื้นฐานทั่วไป คือ Primitive (int, boolean, float, byte, short, double, long), Primitive in array, Package java.lang (Boolean, Byte, Character, Double, Float, Integer, Long, Short, String) และ Package

java.lang in array หากผู้เขียนโปรแกรมต้องการส่งค่าในรูปแบบอื่นระบบแสดงภาพในงานวิจัยนี้จะเก็บค่าดังกล่าว ในรูปแบบตามที่ JDI เสนอ ซึ่งผู้เขียนโปรแกรมสามารถศึกษาเรียกดูค่าได้จากเอกสาร JDI API

2. เมื่อขนาดต้นไม้ปริภูมิสถานะมีปมสถานะจำนวนมาก การควบคุมภาพหรือเลื่อนปรับภาพผ่านส่วนติดต่อจะทำได้ช้าลงเนื่องจากต้องใช้หน่วยความจำจำนวนมากในการประมวลผลภาพ

7.3 ข้อเสนอแนะ

1. งานวิจัยนี้เน้นการแสดงภาพต้นไม้ปริภูมิสถานะที่ได้จากการผลิตสถานะด้วยโปรแกรมค้นคำตอบในปริภูมิแบบเรียกซ้ำเมท็อด และสถานะที่สร้างด้วยอ็อบเจกต์จากโปรแกรมค้นคำตอบที่เขียนแบบวงวนทำซ้ำ หากผู้เขียนโปรแกรมมีโครงสร้างการค้นคำตอบในรูปแบบอื่นสามารถศึกษาการทำงานของระบบแสดงภาพในงานวิจัยนี้เพื่อประยุกต์ให้เข้ากับรูปแบบการเขียนโปรแกรมค้นคำตอบในปริภูมิสถานะในแบบใหม่ได้
2. งานวิจัยนี้รองรับตัวตรวจในภาษาจาวาเท่านั้น แต่บางคนอาจสนใจภาษาอื่นเช่น ภาษาซีชาร์ป (C#) ภาษาซีพลัสพลัส เป็นต้น ควรมีการพัฒนาระบบแสดงภาพปริภูมิสถานะสำหรับภาษาอื่น เพื่อช่วยให้ผู้สนใจภาษาอื่นได้มีโปรแกรมแสดงภาพปริภูมิสถานะเพื่อใช้ในการพัฒนาการเรียนรู้การเขียนโปรแกรมค้นคำตอบในปริภูมิสถานะ

รายการอ้างอิง

[1] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. Introduction to Algorithm. Hightstown: McGraw-Hill, 2002.

[2] S.J. Russell, and P. Norvig. Artificial Intelligence: A Modern Approach. New Jersey: Prentice Hall, 2003.

[3] Simon. The Sciences of the Artificial. MA: MIT Press, 1969.

[4] Nilsson. Problem-Solving Methods in Artificial Intelligence. New York: McGraw-Hill, 1971.

[5] Samy S. Abu Naser. Developing Visualization Tool for Teaching AI Searching Algorithms, Information Technology Journal, pp. 350-355. Pakistan: Asian Network for Scientific Information, 2008.

[6] V. Ciesielski, and P. McDonald. Using animation of state space algorithms to overcome student learning difficulties, Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education, pp.97-100. New York: ACM, 2001.

[7] Christopher Thornton, and Benedict du Boulay. Artificial Intelligence: Strategies, Applications, and Models Through search. USA:AMACOM, 2005.

[8] Moreno, A., Myller, N., Sutinen, E., and Ben-Ari. Visualizing Programs with Jeliot 3, Proc. of the International Working Conf. on Advanced Visual Interfaces, pp.373 – 376. New York: ACM, 2004.

[9] J. Ángel Velázquez-Iturbide, Antonio Pérez-Carrasco, and Jaime Urquiza-Fuentes .SRec: An Animation System of Recursion for Algorithm Courses, Proceedings of the 13th annual conference on Innovation and technology in computer science education, pp.225-229. New York: ACM, 2008.

[10] Steven L. Tanimoto, and Stefano Levialdi. A transparent interface to state-space search programs, Proceedings of the 2006 ACM symposium on Software visualization, pp.151-152. New York: ACM, 2006.

[11] Lundh F. An introduction to Tkinter [Online]. 1999. Available from : http://www.pythonware.com/library/tkinterintroduction/ [2009,Jan 11].

[12] Steven L. Tanimoto. Towards a Shared Language for Problem-Solving in Design, Proceedings of the 2007 Symposium on Science of Design, pp.19-21. New York: ACM, 2007.

[13] Pressman, Singmaster, and David. The Jealous Husbands and The Missionaries and Cannibals, The Mathematical Gazette, pp. 73–81. UK:The Mathematical Association, 1989.

[14] Cinque, L., Sellers Canizares, S., and Tanimoto. Application of a transparent interface methodology to image processing, Journal of Visual Languages and Computing, pp.504-512. Orlando, FL, USA: Academic Press, 2007.

[15] Richard E. Neapolitan, and Kumarass Naimipour. Foundations of Algorithms Using Java Pseudocode. MA: Jones & Bartlett Publishers, 2002.

[16] Sun MicroSystems, Inc., (n.d.). Java Platform Debugger Architecture [Online]. 2009. Available from : http://java.sun.com/javase/technologies/core/toolsapis/jpda/ [2009,Jan 11].

[17] Sheng Liang. Java(TM) Native Interface, Programmer's Guide and Specification. New Jersey: Prentice Hall PTR, 1999.

[18] Tim Lindholm, and Frank Yellin. Java(TM) Virtual Machine Specification. New Jersey: Prentice Hall PTR, 1999.

[19] Sun MicroSystems, Inc., (n.d.). Java Debug Interface [Online]. 2009. Available from : http://java.sun.com/j2se/1.5.0/docs/guide/ jpda/jdi/ [2009,Jan 11].

[20] Joshua O'Madadhain, Danyel Fisher, and Tom Nelson .The Java Universal Network/Graph Framework [Online]. 2005. Available from : http://jung.sourceforge.net [2009,Jan 22].

[21] Donald Knuth. The Art of Computer Programming: Fundamental algorithms. Massachusetts: Addison-Wesley Professional,1997.

[22] Norvig, and Peter. Techniques for Automatic Memoization with Applications to Context-Free Parsing. Computational Linguistics. pp.91-98 . MA,USA: MIT Press Cambridge, 1991.

[23] James Ten Eyck. Algorithm Analysis and Design [Online]. 2009. Available from : http://www.academic.marist.edu/~jzbv/algorithms/ [2009,Feb 19].

[24] Jiri Sedlacek, and Tomas Hurka. VisualVM [Online]. 2009. Available from : https://visualvm.dev.java.net [2009,May 11].

[25] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. Introduction to Algorithm. McGraw-Hill, 2002.

[26] Sun MicroSystems, Inc., (n.d.). Java Annotation [Online]. 2009. Available from : http://java.sun.com/docs/books/tutorial/java/javaOO/ annotations.html [2009,May 21].

[27] Sun MicroSystems, Inc., (n.d.). Java Reflection [Online]. 2009. Available from : http://java.sun.com/docs/books/tutorial/reflect [2009, May 21].

ภาคผนวก

ภาคผนวก ก วิธีการเขียนโปรแกรม

งานวิจัยนี้ นำเสนอระบบแสดงภาพต้นไม้ปริภูมิสถานะโดยมีการใช้คุณสมบัติต่างๆ ของจาวาในการสร้างระบบแสดงภาพได้สะดวกขึ้นดังนี้

1 Annotation

Annotation [26] ถูกเพิ่มเข้ามาใน Java 1.5 Tiger (proposed by JSR-175) สร้างขึ้นเพื่อใช้สำหรับคำอธิบายสิ่งต่าง ๆ ในลักษณะ declarative information ร่วมกับโปรแกรมของจาวาใน Element ต่าง ๆ อย่างเช่น classes, methods, fields, parameters, local variables และ packages โดยตัวแปลภาษาจะจัดการเก็บ metadata ในคลาสไฟล์ หลังจากนั้น Java Virtual Machine หรือโปรแกรมอื่น ๆสามารถมองหา metadata นั้นเพื่อปรับเปลี่ยนพฤติกรรมหรือทำงานร่วมกันกับโปรแกรมใน Element นั้น

ก.1.1 การประกาศ Annotation

สำหรับภาษาจาวานั้นการประกาศรูปแบบของ Annotation ด้วยนิยามที่ขึ้นต้นด้วยเครื่องหมาย “@” ซึ่งกำกับไว้ด้านหน้าของ interface จากรหัสที่ ก.1 เป็นการนิยาม `VMain` ซึ่งเป็น `interface` โดยกำหนดเมท็อดสำหรับเขียนข้อกำหนดใน `@VMain` ดังนี้ `recursiveMethod(), stateClass(),showAllStates()` ตามลำดับ

```
import java.lang.annotation.*;
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface VMain {
    String recuesiveMethod() default "";
    String stateClass() default "";
    boolean showAllStates() default false;
}
```

รหัสที่ ก.1 การประกาศ Annotation

ก.1.2 การเรียกใช้งาน Annotation

การแปลโปรแกรมด้วย Annotation สามารถใช้ตัวแปลภาษาในการบอกข้อผิดพลาดของโปรแกรมได้ อย่างเช่นการใช้ annotation เพื่อบอกข้อผิดพลาดของโปรแกรม ด้วยการกำกับ `@Override` สำหรับเป็นข้อมูลให้กับตัวแปลภาษารับทราบว่าเมท็อดนี้มีการ override จากคลาสแม่ โดยการใส่ `@Override` ไว้บนหัวเมท็อดของคลาสลูกที่มีการ override นั้น จะทำให้ตัวแปลภาษาสามารถตรวจสอบได้ว่ามีเมท็อดที่มีชื่อและพารามิเตอร์อยู่ที่คลาสแม่หรือไม่ หากไม่มีก็อาจเป็นได้ว่าผู้เขียนมีการเขียนชื่อเมท็อดคลาสลูกผิด ตัวแปลภาษาจะบอกกับผู้เขียนว่ามีข้อผิดพลาดเกิดขึ้น

การใช้ annotation ณ เวลา compile-time และ deployment-time สามารถอ่านข้อมูลของ annotation เพื่อที่จะสร้างรหัส, XML file, และอื่น ๆ

การใช้ annotation ณ เวลา Runtime processing คือ บาง annotation สามารถใช้พิจารณาได้ถึงตอนทำงาน

ก.1.3 ตัวอย่างการเรียกใช้งาน annotation

การเรียกใช้งาน Annotation ที่สร้างไว้นั้นสามารถเรียกใช้งานงานเงื่อนไขที่ประกาศไว้จากรหัสที่ ก.2 เป็นการเรียกใช้ annotation ที่ประกาศไว้ตาม รหัสที่ ก.2 โดยกำกับไว้บน เมท็อด `fibonaci` โดย ประกาศเครื่องหมาย @ ตามด้วย ชื่อ `interface` คือ `VMain` จากนั้นกำหนดค่าต่างๆของสมาชิกดังนี้

`recursiveMethod=""` เป็นชนิดของสตริง มีสมบัติ "`memoize`" เป็นการระบุข้อกำหนดในการใช้กลไกการทำงานของระบบแสดงภาพ เพื่อติดตามการทำงานแบบเรียกซ้ำที่เมท็อดชื่อ `memoize` และ `showAllStates=false` เป็นการระบุข้อกำหนดในการใช้กลไกการทำงานของระบบแสดงภาพติดตาม ค่าพารามิเตอร์ตัวสุดท้ายในเมท็อดชื่อ `memorize`

```
@VMain(recursiveMethod = "memoize", showAllStates = true)
public void fibonaci() {
    memoize(21,false);
}
public int memoize(int n ,boolean pruned) {
    if (n < 2) {
        return n;
    }
    if (mf[n] > 0) {
        return memoize(n - 1,true)+ memoize(n - 2, true);
    }
    return memoize[n] = memoize(n - 1,false) + memoize(n - 2,false);
}
```

รหัสที่ ก.2 การเรียกใช้งาน Annotation

2 จาวา Reflection

จาวารีเฟลคชัน คือ การที่โปรแกรมสามารถรู้ถึงโครงสร้างของตัวเอง และยังสามารถปรับเปลี่ยนโครงสร้างนั้นผ่านทางด้วย Java reflection API [27]

แนวคิดพื้นฐานของรีแฟลคชัน คือการใช้ Java API เข้าไปตรวจสอบและปรับเปลี่ยนพฤติกรรมของโปรแกรมขณะที่โปรแกรมกำลังทำงาน ดังในรหัสที่ ก.3 จะเห็นว่าเมท็อด `begin` มีการรับค่า `Object` จากนั้นจะดึงข้อมูลคลาสของออกมาโดยใช้ `obj.getClass()` จากนั้นจะดึงข้อมูลของเมท็อดที่อ็อบเจกต์นั้นมีโดยใช้ `c.getDeclaredMethods()` และวนตรวจสอบ `@VMain` ที่ประกาศไว้บนเมท็อดหรือไม่โดยใช้ `m.isAnnotationPresent(VMain.class)` ตรวจสอบเงื่อนไข ถ้าเมท็อดนั้นมีการกำกับ `@VMain` จะเพิ่มข้อมูลใน `VMainUtil`

```
public static void begin(Object obj) {
    for(Method method : obj.getClass().getDeclaredMethods()){
        if(method.isAnnotationPresent(VMain.class)){
            VMain vmain = m.getAnnotation(VMain.class);
            VMainUtil vutil = new VMainUtil();
                vutil.setClassName(obj.getClass().getName());
                vutil.setVmainMethodName(method.getName());
            if(isRecursive(vmain)){
                String filepath = getPathReader(obj);
                vutil.setDebuggingPathName(filepath);
                vutil.setMethodRecursiveName(vmain.recursiveMethod());
                vutil.setPrunedName(vmain.AllStatePruned)
                vutil.setType(VType.METHOD_TYPE);
            }else if(isstateClass(vmain)){
                vutil.setStateClassName(vmain.stateClass());
                vutil.setType(VType.OBJECT_TYPE);
            }
            startup(vutil);
        }
    }
}
```

รหัสที่ ก.3 การดึงเมท็อดที่มี annotation `@VMain` ของอ็อบเจกต์

ภาคผนวก ข ลักษณะกำกับแสดงภาพปริภูมิสถานะ

ตัวกำกับแสดงภาพ `@VMain` เป็นตัวกำกับคุณสมบัติของระบบแสดงภาพต้นไม้ปริภูมิสถานะ ซึ่งภายในมีการระบุค่าที่ใช้กำหนดการสั่งระบบแสดงภาพทำงานรวมถึงค่าที่ใช้ติดตามสถานะ งานวิจัยนี้ได้อธิบายรายละเอียดของระบบแสดงภาพต้นไม้ปริภูมิประโยชน์ไปแล้ว ในภาคผนวกนี้จึงของรวบรวมและสรุปสั้น ๆ ของตัวกำกับแสดงภาพ `@VMain` โดยจะมีดังนี้

การประกาศตัวกำกับแสดงภาพ `@VMain` ในการทำงานแบบอ็อบเจ็กต์

ประเภท	ชื่อ	คำอธิบาย
`String`	`stateClass`	ชื่อคลาสที่จัดการสถานะ
`boolean`	`showAllStates`	ระบุชื่อพารามิเตอร์แสดงความเข้มเส้นเชื่อมปมที่เป็นแบบบูลีน

คลาสสถานะเป็นคลาสลูกของ `VState` ซึ่งเป็น `abstract` เพื่อให้ระบบแสดงภาพต้นไม้ปริภูมิสถานะรับทราบโครงสร้างและความสัมพันธ์ของสถานะที่เกิดใหม่ ซึ่งภายใน `VState` ประกอบด้วย

`abstract VState getParent()`	
หน้าที่	คืนค่าปมพ่อแม่
ผลที่คืน	สถานะปมพ่อแม่ของสถานะที่ผลิต
`public boolean showAllStates()`	
หน้าที่	คืนค่าบูลีนแสดงความเข้มเส้นเชื่อมปม
ผลที่คืน	ค่าบูลีนที่ใช้ความเข้มเส้นเชื่อมระหว่างปม
`public void drawState(Graphics2D g2d, int x, int y)`	
หน้าที่	คืนค่าการวาดรูปบนปม
พารามิเตอร์	`g2d` – จาวากราฟิกสำหรับวาดภาพ `x` – ตำแหน่งวางภาพกำกับปมแนวแกน x `y` – ตำแหน่งวางภาพกำกับปมแนวแกน y
`public static void fireEvent(VState fireEvent)`	

หน้าที่	ส่งสถานะเกิดใหม่เข้าสู่กลไกแสดงภาพ
พารามิเตอร์	ปมสถานะที่ถูกผลิต

การประกาศตัวกำกับแสดงภาพ `@VMain` ในการทำงานแบบเรียกซ้ำ

ประเภท	ชื่อ	คำอธิบาย
`String`	`recursiveMethod`	ชื่อเมท็อดที่ทำงานแบบเรียกซ้ำ
`boolean`	`showAllStates`	ระบุชื่อพารามิเตอร์แสดงความเข้มเส้นเชื่อมปมที่เป็นแบบบูลีน

ในการทำงานแบบเรียกซ้ำนั้นโปรแกรมค้นปริภูมิที่เป็นคลาสลูกของ `VDrawable` ซึ่งเป็น `abstract` ใช้สำหรับวาดรูปบนปมการทำงานได้โดยการ Override เมท็อด `drawState` ด้านล่างลงไปในโปรแกรมค้นปริภูมิ

`public void drawState(Graphics2D g2d, int x, int y, List agrs)`	
หน้าที่	คืนค่าการวาดรูปบนปม
พารามิเตอร์	`g2d` – จาวากราฟิกสำหรับวาดภาพ `x` – ตำแหน่งวางภาพกำกับปมแนวแกน x `y` – ตำแหน่งวางภาพกำกับปมแนวแกน y `args` – ลิสต์เก็บลำดับค่าพารามิเตอร์เมท็อดเรียกซ้ำ

ภาคผนวก ค ตัวอย่างรหัสคำสั่งสำหรับแสดงภาพต้นไม้ปริภูมิสถานะ

ค.1 รหัสคำสั่งการหาผลเฉลยคำตอบปัญหา Sum of subset ในการค้นคำตอบ การค้นหาแบบลึกก่อนและการค้นหาแบบกว้างก่อน

```
import java.util.*;
import jstate101.*;
public class SumSubset{
    static class VNode extends VState {
        VNode parent;
        public int[] x;
        public VNode(int[] subset, VNode parent) {
            this.parent = parent;
            this.x = subset;
            VState.fireevent(this);
        }
        @Override
        public VState getParent() {
            return parent;
        }
        @Override
        public String toString() {
            return "";
        }
    }
    static void sumOfSubsetDFS(int[] a, int k) {
        Stack<VNode> s = new Stack();
        int[] x = new int[0];
        VNode v = new VNode(x, null);
        s.push(v);
        while(!s.isEmpty()) {
            v = s.pop();
            if(v.x.length == a.length) {
                if(sum(a, v.x) == k) {
                    break;
                }
            } else {
                int[] y = Arrays.copyOf(v.x, v.x.length+ 1);
                y[y.length - 1] = 0;
                VNode v2 = new VNode(y, v);
                s.push(v2);
                y = Arrays.copyOf(v.x, v.x.length + 1);
                y[y.length - 1] = 1;
                VNode v3 = new VNode(y, v);
                s.push(v3);
            }
        }
    }
    static void sumOfSubsetBFS(int[] a, int k) {
        Queue<VNode> q = new LinkedList();
        int[] x = new int[0];
        VNode v = new VNode(x, null);
        q.add(v);
        while(!q.isEmpty()) {
            v = q.remove();
            if(v.x.length == a.length) {
                if(sum(a, v.x) == k) {
                    break;
                }
            } else {
                int[] y = Arrays.copyOf(v.x, v.x.length+1);
                y[y.length - 1] = 1;
                VNode v2 = new VNode(y, v);
                if(v2.x.length == a.length) {
```

```
                    if(sum(a, v2.x) == k) {
                        break;
                    }
                }
                q.add(v2);
                y = Arrays.copyOf(v.x, v.x.length + 1);
                y[y.length - 1] = 0;
                VNode v3 = new VNode(y, v);
                if(((int[]) v3.x).length == a.length) {
                    if(sum(a, v3.x) == k) {
                        break;
                    }
                }
                q.add(v3);
            }
        }
    }
    static void sumOfSubsetRCS(int[] a, int k, int[] x) {
        if(x.length == a.length) {
            if(sum(a, x) == k) {
                System.out.println(Arrays.toString(x));
            }
        } else {
            int[] y = Arrays.copyOf(x, x.length + 1);
            y[y.length - 1] = 1;
            sumOfSubsetRCS(a, k, y, Arrays.toString(y));
            y = Arrays.copyOf(x, x.length + 1);
            y[y.length - 1] = 0;
            sumOfSubsetRCS(a, k, y, Arrays.toString(y));
        }
    }
    static int sum(int[] a, int[] x) {
        int sum = 0;
        for(int init = 0; init < x.length; init++) {
            if(x[init] == 1) {
                sum += a[init];
            }
        }
        return sum;
    }
    static int[] i = {1,2,3,4,5,6,7,8,9};
    static int init = 16;
    @VMain(stateClass = "SumSubset.VNode")
    public static void sumDFS() {
        sumOfSubsetDFS(i, init);
    }
    @VMain(stateClass = "SumSubset.VNode")
    public static void sumBFS() {
        sumOfSubsetBFS(i, init);
    }
    @VMain(recursiveMethod = "sumOfSubsetRCS")
    public static void sumRCS() {
        sumOfSubsetRCS(i, init, new int[0]);
    }
    public static void main(String args[]) {
        VEngine.begin(new SumSubset());
    }
}
```

รหัสที่ ค.1 การค้นคำตอบปัญหา Sum of subset

ค.2 รหัสคำสั่งการหาผลเฉลยคำตอบปัญหา Fibonaci ในการการค้นหาคำตอบแนวลึกด้วย recursive และการใช้กลวิธี memoization

```
import jstate101.*;
public class Fibonaci
{
    public static void main(String[] args) {
        VEngine.begin(new Fibonaci());
    }
    @VMain(method = "f") // แสดงต้นไม้ของการเรียก f
    public static void main_F() {
        System.out.println("f(21) = " + f(21));
    }
    @VMain(recursiveMethod = "m") // แสดงต้นไม้ของการเรียก m
    public void main_M() {
        System.out.println("m(21) = " + m(21));
    }
    @VMain(method = "mf", showAllStates = true) // แสดงต้นไม้ของการเรียก m
    public void main_FM() {
        System.out.println("mf(21) = " + mf(21, true));
    }
    static int f(int n) { //การทำงานแบบ recursive
        if (n < 2)
            return n;
        return f(n - 1) + f(n - 2);
    }
    static int[] d = new int[50];
    static int m(int n) { //กลวิธี memoization
        if (n < 2)return n;
        if (d[n] > 0)return d[n]; // คืนคำตอบแล้ว
        return d[n] = m(n - 1) + m(n - 2); // จำคำตอบ
    }
    static int[] mf = new int[50];
    static int mf(int n, boolean pruned) {
        //การทำงาน memoization บนต้นไม้ปริภูมิสถานะ
        if (n < 2)
            return n;
        if (mf[n] > 0)
            return mf(n - 1, true) + mf(n - 2, true);
        return mf[n] = mf(n - 1,false) + mf(n - 2,false); // จำคำตอบ
    }
}
```

รหัสที่ ค.2 การค้นคำตอบแนวลึกของปัญหา Fibonaci และการใช้กลวิธี memoization

ค.3 รหัสคำสั่งการหาผลเฉลยคำตอบปัญหา N-queen ในการค้นหาคำตอบแนวลึกด้วยเรียกซ้ำและการใช้กลวิธีค้นคำตอบด้วยการย้อนรอย

```
import jstate101.*;
public class Queens {
    public boolean isConsistent(int[] col, int n) {
        for (int i = 0; i < n; i++) {
            if (col[n] == col[i] || Math.abs(col[n] - col[i]) == Math.abs(i -
n)) {
                return false;
            }
        }
        return true;
    }
    public void queensRCS(int[] col, int i, boolean ispruned) {
        int N = x.length;
        if (k == N) {
            printQueens(col,N);
        } else {
            for (int j = 0; j < N; j++) {
                x[i] = j;
                queensRCS(col, i + 1, !isConsistent(col, i) || ispruned);

            }
        }
    }
    @VMain(method = "queensRCS", showAllStates = true)
    public static void mainDFS_RCS() {
        int[] col = new int[6];
        queensRCS(col, 0, false);
    }
    public static void main(String[] args) {
        VEngine.begin(new Queens());
    }
    public void printQueens(int[] col ,int n) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (col[i] == j) System.out.print("Q ");
                else   System.out.print("* ");
            }
            System.out.println();
        }
        System.out.println();
    }
}
```

รหัสที่ ค.3 การค้นคำตอบแนวลึกปัญหา N-queen กับกลวิธีย้อนรอย

ค.4 รหัสคำสั่งการหาผลเฉลยคำตอบปัญหาการเดินทางของพนักงานขายแบบดีที่สุด

```
import java.util.*;
import jstate101.*;
import jstate101.util.graph.*;
public class TSPBFS {
    private final double INFINITY = 1E10;
    private double minTour;
    private Graph g;       //an adjacency matrix rep of the graph
    private Vector bestList;
    private boolean[] mark;
    private double[] minEdge;
    private PriorityQueue q;
    private int countnode = 0;
    public class TSPNode extends VState implements Comparable {
        int level;
        double length, bound;
        Vector path;
        Object lastVertex;
        TSPNode parent;
        int vertex;
        protected TSPNode(TSPNode parent, Object vert) {
            level = 0;
            length = 0.0;
            bound = 0.0;
            vertex = (Integer) vert;
            lastVertex = vert;
            this.parent = parent;
            path = new Vector();
            VState.fireEvent(this);
        }
        protected void copyList(Vector v) {
            if (v == null || v.isEmpty()) {
                path = new Vector();
            } else {
                path = new Vector(v);
            }
        }
        protected void add(Object vtx) {
            path.add(vtx);
        }
        public int compareTo(Object obj) {
            TSPNode n = (TSPNode) obj;
            if (this.length < n.length) {
                return -1;
            } else if (this.length > n.length) {
                return 1;
            }
            return 0;
        }
        @Override
        public VState getParent() {
            return parent;
        }
    }
    public void tsp(Graph G) {
        g = G;
        minTour = INFINITY;
        mark = new boolean[N + 1];
        minEdge = new double[N + 1];
        cost();
        q = new PriorityQueue();
        Integer v1 = new Integer(1);
        TSPNode root = new TSPNode(null, v1);
        root.add(v1);
        q.add(root);
        while (!q.isEmpty()) {
```

```
            TSPNode temp = (TSPNode) q.poll();
            if (temp.bound < minTour) {
                Iterator itr = g.neighbors(temp.lastVertex);
                while (itr.hasNext()) {
                    Object nextVert = itr.next();
                    if (!temp.path.contains(nextVert)) {
                        TSPNode u = new TSPNode(temp, nextVert);
                        u.level = temp.level + 1;
                        u.length = temp.length +
                        length(temp.lastVertex,nextVert);
                        u.copyList(temp.path);
                        u.add(nextVert);
                        if (u.level == N - 2) {
                            Iterator ntr = g.neighbors(nextVert);
                            while (ntr.hasNext()) {
                                Object x = ntr.next();
                                if (!u.path.contains(x)) {
                                    u.add(x);
                                    u.length += length(nextVert, x);
                                    u.add(u.path.get(0));
                                    u.length += length(x, u.path.get(0));
                                    if (u.length < minTour) {
                                        minTour = u.length;
                                        bestList = new Vector(u.path);
                                    }
                                    break;
                                }
                            }
                        } else {
                            if (u.length < minTour) {
                                q.add(u);
                            }
                        }
                    }
                }
            }
        }
    }
    private double length(Object v1, Object v2) {
        Edge e = g.getEdge(v1, v2);
        Object hold = e.label();
        return ((Double) hold).doubleValue();
    }
    @VMain(stateClass = "TSPBFS.TSPNode")
    public void init() {
        int N = 12;
        Graph g = new DirectedGraphList();
        for (int sub1 = 1; sub1 <= N; sub1++)
            g.add(new Integer((sub1)));
        for (int i = 1; i <= N; i++) {
            for (int j = 1; j <= N; j++) {
                if (i != j) {
                    Double path = Math.random();
                    g.addEdge(new Integer(i), new Integer(j), path);
                }
            }
        }
        tsp(g);
    }
    public static void main(String[] args) {
        VEngine.begin( new TSPBFS());
    }
}
```

รหัสที่ ค.4 การค้นคำตอบแบบดีที่สุดในปัญหา Traveling Salesperson

ค.5 รหัสคำสั่งการหาผลเฉลยคำตอบปัญหาการเดินทางของพนักงานขายในการค้นคำตอบแบบดีที่สุดกับกลวิธีการขยายและจำกัดเขต

```
import java.util.*;
import jstate101.*;
import jstate.util.graph.*;
public class TSPBnB {
    private final double INFINITY = 1E10;
    private double minTour;
    private Graph g;       //an adjacency matrix rep of the graph
    private int N;         //number of vertices
    private Vector bestList;
    private boolean[] mark;
    private double[] minEdge;
    private PriorityQueue q;
    private int countnode = 0;
    public class TSPNode extends VState implements Comparable {
        int level;
        double length, bound;
        Vector path;
        Object lastVertex;
        TSPNode parent;
        int vertex;
        protected TSPNode(TSPNode parent, Object vert) {
            level = 0;
            length = 0.0;
            bound = 0.0;
            vertex = (Integer) vert;
            lastVertex = vert;
            this.parent = parent;
            path = new Vector();
            VState.fireevent(this);
        }
        protected void copyList(Vector v) {
            if (v == null || v.isEmpty()) {
                path = new Vector();
            } else {
                path = new Vector(v);
            }
        }
        protected void add(Object vtx) {
            path.add(vtx);
        }
        public int compareTo(Object obj) {
            TSPNode n = (TSPNode) obj;
            if (this.bound < n.bound) {
                return -1;
            } else if (this.bound > n.bound) {
                return 1;
            }
            return 0;
        }
        @Override
        public VState getParent() {
            return parent;
        }
        @Override
        public String toString() {
            return "" + vertex;
        }
    }
    public void tsp(Graph G) {
        g = G;
        N = g.size();
        minTour = INFINITY;
        mark = new boolean[N + 1];
```

```
        minEdge = new double[N + 1];
        cost();
        q = new PriorityQueue();
        Integer v1 = new Integer(1);
        TSPNode root = new TSPNode(null, v1);
        root.add(v1);
        root.bound = bound(root);
        q.add(root);
        while (!q.isEmpty()) {
            TSPNode temp = (TSPNode) q.poll();
            if (temp.bound < minTour) {
                Iterator itr = g.neighbors(temp.lastVertex);
                while (itr.hasNext()) {
                    Object nextVert = itr.next();
                    if (!temp.path.contains(nextVert)) {
                        TSPNode u = new TSPNode(temp, nextVert);
                        u.level = temp.level + 1;
                        u.length = temp.length +
                        length(temp.lastVertex,nextVert);
                        u.copyList(temp.path);
                        u.add(nextVert);
                        if (u.level == N - 2) {
                            Iterator ntr = g.neighbors(nextVert);
                            while (ntr.hasNext()) {
                                Object x = ntr.next();
                                if (!u.path.contains(x)) {
                                    u.add(x);
                                    u.length += length(nextVert, x);
                                    u.add(u.path.get(0));
                                    u.length += length(x, u.path.get(0));
                                    if (u.length < minTour) {
                                        minTour = u.length;
                                        bestList = new Vector(u.path);
                                    }
                                    break;
                                }
                            }
                        } else {
                            u.bound = bound(u);
                            if (u.bound < minTour) {
                                q.add(u);
                            }
                        }
                    }
                }
            }
        }
    }
    private double length(Object v1, Object v2) {
        Edge e = g.getEdge(v1, v2);
        Object hold = e.label();
        return ((Double) hold).doubleValue();
    }
    private double bound(TSPNode n) {
        Iterator itr = n.path.iterator();
        Integer lastv = (Integer) n.lastVertex;
        //unmark the last vertex in the path
        for (int i = 0; i <= N; i++) {
            mark[i] = false;
        }
        while (itr.hasNext()) {
            Integer hold = (Integer) itr.next();
            int num = hold.intValue();
            mark[num] = true;
        }
        mark[lastv.intValue()] = false;
        double bnd = n.length;
```

```
        for (int i = 0; i <= N; i++) {
            if (!mark[i]) {
                bnd += minEdge[i];
            }
        }
        return bnd;
    }
    private void cost() {
        for (int i = 0; i <= N; i++) {
            mark[i] = false;
        }
        Iterator itr = g.iterator();
        while (itr.hasNext()) {
            Object obj = itr.next();
            GraphListVertex v = (GraphListVertex) obj;
            Iterator jtr = g.neighbors(v.label());
            double cost = INFINITY;
            while (jtr.hasNext()) {
                Object w = jtr.next();
                double len = length(v.label(), w);
                if (len < cost) {
                    cost = len;
                }
            }
            minEdge[((Integer) v.label()).intValue()] = cost;
        }
    }
    @VMain(stateClass = "TSPBnB.TSPNode")
    public void init() {
        int N = 12;
        Graph g = new DirectedGraphList();
        for (int sub1 = 1; sub1 <= N; sub1++) {
            g.add(new Integer((sub1)));
        }
        for (int i = 1; i <= N; i++) {
            for (int j = 1; j <= N; j++) {
                if (i != j) {
                    Double path = Math.random();
                    g.addEdge(new Integer(i), new Integer(j), path);
                }
            }
        }
        tsp(g);
    }
    public static void main(String[] args) {
        VEngine.begin( new TSPBnB());
    }
}
```

รหัสที่ ค.5 การเพิ่มกลวิธีขยายและจำกัดเขตในปัญหา Traveling Salesperson

ค.6 รหัสคำสั่งการหาผลเฉลยคำตอบปัญหา The 0-1 Knapsack ในการทำงานตามแนวลึกด้วยเรียกซ้ำ

```
import java.util.LinkedList;
import java.util.List;
import jstate101.*;
public class Knapsack_BF {
    private static int maxProfit;
    private static int W;   //capacity
    private static int[] s;
    private static int[] v;
    private static List bestList;
    private static int numItems;
    public static void init(int capacity, int[] w, int[] p, int n) {
        maxProfit = 0;
        W = capacity;
        s = w;
        v = p;
        numItems = n;
        bestList = null;
    }
    public static void knapsack(int index, int weight, int profit
    ,List cList) {
        if (weight <= W && profit > maxProfit) {
            maxProfit = profit;
            bestList = new LinkedList(cList);
        }
        if(index + 1 >= numItems )return;
            List leftList = new LinkedList(cList);
            leftList.add(new Integer(index + 1));
            knapsack(index+1,weight + s[index+1], profit+
            v[index+1],leftList);
            List rightList = new LinkedList(cList);
            knapsack(index + 1, weight, profit, rightList);
    }
    @VMain(method = "knapsack")
    public static void findSolution() {
        int[] weight = {0, 3, 2, 4, 3, 2, 3, 5, 4, 3, 6, 2, 10, 6, 12, 10,5};
        int[] profit = {0, 24, 14, 26, 19, 12, 17, 2, 18, 13, 24, 6, 12, 7,
                14,10,5};
        int N = 17;
        init(24, weight, profit, N);
        List currentList = new LinkedList();
        knapsack(0, 0, 0, currentList);
    }
    public static void main(String[] args) {
        VEngine.begin(new Knapsack_BF());
    }
}
```

รหัสที่ ค.6 การค้นหาคำตอบตามแนวลึกแบบเรียกซ้ำในปัญหา The 0-1 Knapsack

ค.7 รหัสคำสั่งการหาผลเฉลยคำตอบปัญหา the 0-1 Knapsack ในการค้นหาคำตอบตามแนวลึกแบบเรียกซ้ำและการใช้กลวิธีค้นคำตอบด้วยวิธีการย้อนรอย

```
import java.util.LinkedList;
import java.util.List;
import jstate101.*;
public class KnapsackBT_W {
    private static int maxProfit;
    private static int W;   //capacity
    private static int[] s;
    private static int[] v;
    private static List bestList;
    private static int numItems;
    public static void init(int capacity, int[] w, int[] p, int n) {
        maxProfit = 0;
        W = capacity;
        s = w;
        v = p;
        numItems = n;
        bestList = null;
    }
    public static void knapsack(int index, int weight, int profit
    ,List cList) {
        if (weight <= W && profit > maxProfit) {
            maxProfit = profit;
            bestList = new LinkedList(cList);
        }
        if (promising(index, weight)) {
            List leftList = new LinkedList(cList);
            leftList.add(new Integer(index + 1));
            knapsack(index+1, weight+s[index + 1], profit+v[index + 1]
            , leftList);
            List rightList = new LinkedList(cList);
            knapsack(index + 1, weight, profit, rightList);
       }
    }
    public static boolean promising(int item, int weight) {
        int k = item+1 ;
        int totalSize = weight;
        if (weight > W) {
            return false;
        }
        return totalSize+s[k] <= W;
    }
    @VMain(method = "knapsack")
    public static void findSolution() {
        int[] weight = {0, 3, 2, 4, 3, 2, 3, 5, 4, 3, 6, 2, 10, 6, 12, 10,
                        5};
        int[] profit = {0, 24, 14, 26, 19, 12, 17, 2, 18, 13, 24, 6, 12, 7,
                        14,10,5};
        init(24, weight, profit, 17);
        List currentList = new LinkedList();
        knapsack(0, 0, 0, currentList);
    }
    public static void main(String[] args) {
        VEngine.begin(new KnapsackBT_W());
    }
}
```

รหัสที่ ค.7 การค้นหาคำตอบปัญหาถุงเป้ 0/1 โดยเพิ่มกลวิธีย้อนรอย

ค.8 รหัสคำสั่งการหาผลเฉลยคำตอบปัญหา The 0-1 Knapsack ในการค้นหาคำตอบตามแนวกว้างและการใช้กลวิธีการขยายและจำกัดเขต

```
import java.util.PriorityQueue;
import java.util.Vector;
import jstate101.*;
public class Knapsack_BnB {
    private int maxValue;
    private int K;   //capacity
    private int[] s;
    private int[] v;
    private Vector bestList;
    private int numItems;
    private Queue q;
    static class Node extends VState {
        int level ,size, value, bound;
        Vector contains;
        Node parent;
        public Node(Node parent) {
            this.parent = parent;
            level = 0;
            size = 0;
            value = 0;
            bound = 0;
            contains = null;
            VState.fireevent(this);
        }
        public void copyList(Vector v) {
            if (v == null || v.isEmpty()) {
                contains = new Vector();
                } else {
                    contains = new Vector(v);
                }
            }
        public void add(int index) {
            contains.add(new Integer(index));
        }
        @Override
        public VState getParent() {
            return parent;
        }
    }
    public void knapsack(int capacity, int[] size, int[] value, int n) {
        maxValue = 0;
        K = capacity;
        s = size;
        v = value;
        numItems = n;
        bestList = null;
        q = new LinkedList();
        Node root = new Node(null);
        root.level = 0;
        root.size = 0;
        root.value = 0;
        root.bound = bound(0, 0, 0);
        root.copyList(null);
        q.add(root);
        while (!q.isEmpty()) {
            Node temp = (Node) q.remove();
            if (temp.bound > maxValue) {
                Node u = new Node(temp);
                u.level = temp.level + 1;
                u.size = temp.size + s[temp.level + 1];
                u.value = temp.value + v[temp.level + 1];
                u.copyList(temp.contains);
                u.add(u.level);
```

```
                if (u.size <= K && u.value > maxValue) {
                    maxValue = u.value;
                    bestList = new Vector(u.contains);
                }
                u.bound = bound(u.level, u.size, u.value);
                if (u.bound > maxValue) {
                    q.add(u);
                }
                Node w = new Node(temp);
                w.level = temp.level + 1;
                w.size = temp.size;
                w.value = temp.value;
                w.bound = bound(w.level, w.size, w.value);
                w.copyList(temp.contains);
                if (w.bound > maxValue) {
                    q.add(w);
                }
        }
    }

    public int bound(int item, int size, int value) {
        int bound = value;
        int k = item + 1;
        int totalSize = size;
        if (totalSize > K) {
            return 0;
        }
        while (k < numItems && totalSize + s[k] < K) {
            totalSize += s[k];
            bound += v[k];
            k++;
        }
        if (k < numItems) {
            bound += (K - totalSize) * (v[k] / s[k]);
        }
        return bound;
    }

    @VMain(stateClass = "Knapsack_BnB.Node")
    public void findSolution() {
        int[] sizes = {0, 3, 2, 4, 3, 2, 3, 5, 4, 3, 6, 2, 10, 6, 12, 10, 5};
        int[] values = {0, 24, 14, 26, 19, 12, 17, 2, 18, 13, 24, 6, 12, 7,
                        14, 10, 5};
        knapsack(24, sizes, values, 17);
    }

    public static void main(String[] args) {
        VEngine.begin(new Knapsack_BnB());
    }
}
```

รหัสที่ ค.8 การเพิ่มกลวิธีการขยายและจำกัดเขตในปัญหา the 0-1 Knapsack

ค.9 รหัสคำสั่งการหาผลเฉลยคำตอบปัญหา The 0-1 Knapsack ในการค้นหาคำตอบแบบดีที่สุด กับการใช้กลวิธีการขยายและจำกัดเขต

```
import java.util.PriorityQueue;
import java.util.Vector;
import jstate101.*;
public class KnapsackBFS_BnB {
    private int maxValue;
    private int K;   //capacity
    private int[] s;
    private int[] v;
    private Vector bestList;
    private int numItems;
    private Queue q;
    public class Node extends VState {
        int level ,size, value, bound;
        Vector contains;
        Node parent;
        public Node(Node parent) {
            this.parent = parent;
            level = 0;
            size = 0;
            value = 0;
            bound = 0;
            contains = null;
            VState.fireevent(this);
        }
        public void copyList(Vector v) {
            if (v == null || v.isEmpty()) {
                contains = new Vector();
            } else {
                contains = new Vector(v);
            }
        }
        public void add(int index) {
            contains.add(new Integer(index));
        }
        @Override
        public VState getParent() {
            return parent;
        }
    }
    public void knapsack(int capacity, int[] size, int[] value, int n) {
        maxValue = 0;
        K = capacity;
        s = size;
        v = value;
        numItems = n;
        bestList = null;
        q = new LinkedList();
        Node root = new Node(null);
        root.level = 0;
        root.size = 0;
        root.value = 0;
        root.bound = bound(0, 0, 0);
        root.copyList(null);
        q.add(root);
        while (!q.isEmpty()) {
            Node temp = (Node) q.remove();
            if (temp.bound > maxValue) {
                Node u = new Node(temp);
                u.level = temp.level + 1;
                u.size = temp.size + s[temp.level + 1];
                u.value = temp.value + v[temp.level + 1];
                u.copyList(temp.contains);
                u.add(u.level);
```

```
            if (u.size <= K && u.value > maxValue) {
                maxValue = u.value;
                bestList = new Vector(u.contains);
            }
            u.bound = bound(u.level, u.size, u.value);
            if (u.bound > maxValue) {
                q.add(u);
            }
            Node w = new Node(temp);
            w.level = temp.level + 1;
            w.size = temp.size;
            w.value = temp.value;
            w.bound = bound(w.level, w.size, w.value);
            w.copyList(temp.contains);
            if (w.bound > maxValue) {
                q.add(w);
            }
        }
    }
    public int bound(int item, int size, int value) {
        int bound = value;
        int k = item + 1;
        int totalSize = size;
        if (totalSize > K) {
            return 0;
        }
        while (k < numItems && totalSize + s[k] < K) {
            totalSize += s[k];
            bound += v[k];
            k++;
        }
        if (k < numItems) {
            bound += (K - totalSize) * (v[k] / s[k]);
        }
         return bound;
    }

    @VMain(stateClass = "demo.knapsack.KnapsackBFS_BnB.Node")
    public void findSolution() {
        int[] sizes = {0, 3, 2, 4, 3, 2, 3, 5, 4, 3, 6, 2, 10, 6, 12, 10, 5};
        int[] values = {0, 24, 14, 26, 19, 12, 17, 2, 18, 13, 24, 6, 12, 7,
14, 10,         5};
        knapsack(24, sizes, values, 17);
    }

    public static void main(String[] args) {
        VEngine.begin(new KnapsackBFS_BnB());
    }
}
```

รหัสที่ ค.9 การค้นคำตอบแบบดีที่สุดและการเพิ่มกลวิธีขยายและจำกัดเขตในการแก้ปัญหาถุงเป้แบบ 0/1

ประวัติผู้เขียนวิทยานิพนธ์

นายกิตติชัย เกื้อมา เกิดวันที่ 12 กรกฎาคม พ.ศ. 2526 ที่จังหวัด นครศรีธรรมราช สำเร็จการศึกษาหลักสูตรวิศวกรรมศาสตรบัณฑิต (วศ.บ.) สาขาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ มหาวิทยาลัยสงขลานครินทร์ เมื่อปีการศึกษา 2548 และเข้าศึกษาต่อหลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย เมื่อปีการศึกษา 2550