

การศึกษาการแบ่งชุดข้อมูลสำหรับอัลกอริทึม Bitmap Intersection Lookup
A STUDY OF SEGMENTATION FOR BITMAP INTERSECTION
LOOKUP

โดย
สมภพ ศักดิ์ศรีชัย
SOMPOB SAKSRICHAJ
อภิสร เตีฮ้อ
APISORN TEEHOR

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต
สาขาวิชาเทคโนโลยีสารสนเทศ
คณะเทคโนโลยีสารสนเทศ
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ภาคเรียนที่ 2 ปี การศึกษา 2565

การศึกษาการแบ่งชุดข้อมูลสำหรับอัลกอริทึม Bitmap Intersection Lookup
A STUDY OF SEGMENTATION FOR BITMAP INTERSECTION
LOOKUP

โดย
สมภพ ศักดิ์ศรีชัย
อภิสร ตี้อ้อ

อาจารย์ที่ปรึกษา
ผศ.อักรินทร์ คุณกิตติ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต
สาขาวิชาเทคโนโลยีสารสนเทศ
คณะเทคโนโลยีสารสนเทศ
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ภาคเรียนที่ 2 ปี การศึกษา 2565

**A STUDY OF SEGMENTATION FOR BITMAP INTERSECTION
LOOKUP**

SOMPOB SAKSRICHAI

APISORN TEEHOR

**A PROJECT SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENT FOR THE DEGREE OF
BACHELOR OF SCIENCE PROGRAM IN INFORMATION TECHNOLOGY
SCHOOL OF INFORMATION TECHNOLOGY
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG**

2 / 2022

COPYRIGHT 2023

SCHOOL OF INFORMATION TECHNOLOGY

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

ใบรับรองปริญญาโท ประจำปีการศึกษา 2565

คณะเทคโนโลยีสารสนเทศ


สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การศึกษาการแบ่งชุดข้อมูลสำหรับอัลกอริทึม Bitmap Intersection
Lookup

A Study of Segmentation for Bitmap Intersection Lookup

ผู้จัดทำ

- | | | |
|--------------|--------------|-----------------------|
| 1. นาย สมภพ | ศักดิ์ศรีชัย | รหัสนักศึกษา 62070191 |
| 2. นาย อภิสร | ดี้อ | รหัสนักศึกษา 62070218 |

..........อาจารย์ที่ปรึกษา
(.....ผศ.อัครินทร์ คุณกิตติ.....)

ใบรับรองโครงการ (Project)

เรื่อง

การศึกษาการแบ่งชุดข้อมูลสำหรับอัลกอริทึม BITMAP INTERSECTION

LOOKUP

A STUDY OF SEGMENTATION FOR BITMAP INTERSECTION

LOOKUP

นาย สมภพ ศักดิ์ศรีชัย รหัสนักศึกษา 62070191

นาย อภิสร ตี้อ้อ รหัสนักศึกษา 62070218

ขอรับรองว่ารายงานฉบับนี้ ข้าพเจ้าไม่ได้คัดลอกมาจากที่ใด
รายงานฉบับนี้ได้รับการตรวจสอบและได้อนุมัติให้เป็นส่วนหนึ่งของ
การศึกษาวิชาโครงการ หลักสูตรวิทยาศาสตรบัณฑิต (เทคโนโลยีสารสนเทศ)
ภาคเรียนที่ 2 ปีการศึกษา 2565

.....สมภพ...ศักดิ์ศรีชัย...

(นายสมภพ ศักดิ์ศรีชัย)

.....อภิสร...ตี้อ้อ.....

(นายอภิสร ตี้อ้อ)

หัวข้อโครงการ	การศึกษาการแบ่งชุดข้อมูลสำหรับอัลกอริทึม Bitmap Intersection Lookup		
นักศึกษา	นาย สมภพ ศักดิ์ศรีชัย	รหัสนักศึกษา 62070191	
	นาย อภิสร ดีฮ้อย	รหัสนักศึกษา 62070218	
ปริญญา	วิทยาศาสตรบัณฑิต		
สาขาวิชา	เทคโนโลยีสารสนเทศ		
ปีการศึกษา	2565		
อาจารย์ที่ปรึกษา	ผู้ช่วยศาสตราจารย์ อัครินทร์ คุณกิตติ		

บทคัดย่อ

ในปัจจุบันมีข้อมูลในระบบเครือข่ายจำนวนมากทำให้การทำ Packet Classification มีความล่าช้า และทำให้ใช้พื้นที่หน่วยความจำมากในการจัดเก็บข้อมูล จึงมีจุดประสงค์ในการพัฒนา Algorithm Bitmap Intersection Lookup(BIL) ในการแบ่ง Segment เพื่อลดขนาดหน่วยความจำที่ใช้และลดเวลาในการ update table เพื่อเพิ่มประสิทธิภาพการทำงาน ทางผู้จัดทำจึงได้เสนอแนวทางการแบ่ง Segment ว่าควรแบ่งอย่างไรมีประสิทธิภาพสูงสุดต่อหน่วยความจำที่มี โดยวิธีการแบ่ง Segment ที่เสนอนั้นบ่งบอกว่าควรแบ่ง Segment อย่างไรจึงจะทำให้มีความเร็วการทำงานดีที่สุด และใช้ความจำน้อยที่สุด ดังนั้นผู้จัดทำจึงพิสูจน์ตามวิธีการแบ่ง Segment ที่เสนอ ในการทดลองเริ่มจากกำหนด Requirement คือขนาดความจำและจำนวนของกฎ โดยจะทดสอบหาค่าเฉลี่ยความเร็วในการทำงานและวัดหน่วยความจำที่ใช้ และนำผลลัพธ์ตาม Requirement ที่ได้นำไปเปรียบเทียบกับ การแบ่ง Segment รูปแบบอื่น ๆ

จากผลการทดลองหากแบ่ง Segment มากๆ ทำให้ใช้ความจำน้อยลง และทำให้การทำงานเร็วขึ้น ในการทดลองตาม Requirement เมื่อนำไปเปรียบเทียบกับ การแบ่งรูปแบบอื่น สรุปได้ว่า หากแบ่ง Segment โดยมีจำนวน Segment ที่เท่าๆ กันจะมีประสิทธิภาพดีที่สุดและใช้หน่วยความจำน้อยที่สุด แต่ข้อเสียการแบ่ง Segment มาก ๆ จะทำให้ใช้เวลาในการค้นหาเพิ่มขึ้นตามจำนวน Segment แต่จะห่างกันไม่มาก ดังนั้นการทดลองทั้งหมดเป็นไปตามแนวคิดที่ได้เสนอ สำหรับการนำไปใช้งานจริงควรคำนึงถึงขนาดความจำของเครื่องเพราะประสิทธิภาพของ BIL จะขึ้นอยู่กับขนาดความจำ ถ้ามีความจำมากประสิทธิภาพจะมากตาม

Project Title	A Study of Segmentation for Bitmap Intersection Lookup
Students	Mr. Sompob Saksrichai ID 62070191 Mr. Apisorn Teehor ID 62070218
Degree	Bachelor of Science
Program	Information Technology
Academic Year	2022
Advisor	Asst. Prof. Akharin Khunkitti

ABSTRACT

Therefore, the purpose of developing Algorithm Bitmap Intersection Lookup(BIL) is to divide segments to reduce the amount of memory used and reduce the time to update the table to increase work efficiency. Therefore, the author proved according to the proposed segmentation method. In the experiment, starting from the requirement is the memory size and number of rules, the test will average the working speed and measure the memory used, and the results according to the requirements will be compared with other types of segmentation..

According to the results of the experiment, if the segment is divided a lot, it will use less memory and make the work faster. In the experiment according to the requirement when compared with other forms of division. In conclusion, if you divide a segment with the same number of segments, it will have the best performance and use the least memory, but the disadvantage of splitting a lot of segments will cause it will take more time to search according to the number of segments, but not far apart. Therefore, all experiments are based on the proposed concept. For practical use, the memory size of the machine should be taken into account, because the performance of BIL will depend on the memory size. If there is a lot of memory, the performance will be very followed.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จ ได้ด้วยความกรุณาจากอาจารย์ที่ปรึกษา ผู้ช่วยศาสตราจารย์
อักรินทร์ คุณกิตติ ซึ่งคอยช่วยเหลือ ชี้แนะ และให้คำปรึกษาแนวทางต่าง ๆ ในการแก้ไขจุดบกพร่อง
และสั่งสอนสิ่งต่าง ๆ จนปริญญานิพนธ์ฉบับนี้สำเร็จ

ขอบคุณ คณะอาจารย์ คณะเทคโนโลยีสารสนเทศ ลาดกระบัง ที่อบรม สั่งสอนวิชาต่าง ๆ
ให้นักศึกษาทุกคนมีความรู้ และประสบความสำเร็จในวิชาชีพ

ขอบคุณเพื่อน ๆ พี่ๆ ทุกคนที่ให้คำปรึกษา และช่วยเหลือตลอดมา

สมภพ

ศักดิ์ศรีชัย

อภิสร

ดีฮื้อ

สารบัญ

หน้า

บทคัดย่อภาษาไทย	I
บทคัดย่อภาษาอังกฤษ	II
กิตติกรรมประกาศ.....	III
สารบัญ	IV
สารบัญรูป	VI

บทที่

1. บทนำ.....	1
1.1 ที่มาและความสำคัญ	1
1.2 วัตถุประสงค์.....	1
1.3 วิธีการดำเนินงาน	1
1.4 ขอบเขตงาน	2
1.5 ประโยชน์ที่คาดว่าจะได้รับ.....	2
2. การจัดประเภทข้อมูลและแนวคิดของ Bitmap Intersection Lookup สำหรับ Segmentation .3	
2.1 แนวคิดการจัดประเภทเพื่อกำหนดและการแบ่ง Segment บนระบบเครือข่าย	3
2.2 Search Algorithm	4
2.3 เครื่องมือในการวิเคราะห์ผล	5
2.4 เทคโนโลยีและเครื่องมือที่ใช้ในการพัฒนา	6
3. การออกแบบ Algorithm และ ทดสอบประสิทธิภาพ	7
3.1 การวิเคราะห์ระบบงานเดิม	7
3.2 หลักการทำงานของ Bitmap Intersection Lookup	7
3.3 เป้าหมายของการทำงาน Bitmap Intersection Lookup สำหรับ Segmentation	8
3.4 แนวคิดการแบ่ง Segment.....	8
3.5 หลักการแบ่ง Segment	9
3.6 แนวคิดของการศึกษาวิจัย	9
3.7 ภาพรวมของระบบ.....	10
3.8 การพัฒนา Algorithm	11

สารบัญ (ต่อ)

บทที่	หน้า
3.9 แนวคิดการแบ่ง Segment.....	14
4. การทดลองและวิเคราะห์	17
4.1 ออกแบบการทดลอง	17
4.2 การทดลองความถูกต้อง	17
4.3 การทดลองความเร็วในการสร้างตาราง BIL	19
4.4 การทดลองความเร็วในการ Search ของ BIL.....	20
4.5 ขนาดหน่วยความจำที่ใช้ของ BIL.....	21
4.6 ทดสอบการทำงานทุก Segment 16 BIT	22
4.7 การทดลองตาม Requirement	24
4.8 ผลสรุปการนำอัลกอริทึมมาประยุกต์ใช้	26
5. วิเคราะห์และสรุปผล	27
5.1 สรุปผลการทดลอง	27
5.2 ข้อเสนอแนะและแนวทางในการพัฒนาต่อ	27
บรรณานุกรม	28
ภาคผนวก	29
ประวัติผู้เขียน	38

สารบัญรูป

หน้า

รูปที่

2.1 Linear Search	4
2.2 Pseudocode Linear Search	4
2.3 Binary Search.....	5
2.4 ตัวอย่างการสร้างกราฟโดยใช้ Column	6
3.1 ภาพตัวอย่างการแบ่ง Segment.....	8
3.2 Flowchart การทำงานของ BIL Update Table	10
3.3 Flowchart การทำงานของการ Search	11
3.4 ขั้นตอนการทำงานของ BIL ในการ Update Table	12
3.5 ขั้นตอนการทำงานของ BIL ในการ Search.....	12
3.6 ขั้นตอนการทำงานของ BIL ในการ Search สำหรับการแบ่งชุดข้อมูล	13
3.7 การค้นหาใน BIL 2 Table	13
3.8 ผลลัพธ์การค้นหาใน BIL 2 Table การ AND	14
3.9 กระบวนการออกแบบการแบ่ง Segment	16
4.1 กราฟแสดงความถูกต้องของ BIL & Linear Search	17
4.2 ผลลัพธ์ความถูกต้องของ BIL Search และ Linear Search.....	18
4.3 ความถูกต้องของ BIL Segmentation และ Linear Search.....	18
4.4 กราฟแสดงความถูกต้องระหว่าง BIL Segmentation และ Linear Search	19
4.5 ความเร็วในการสร้างตารางของ BIL	20
4.6 ภาพขยายความเร็วในการสร้างตารางของ BIL.....	20
4.7 ความเร็วในการค้นหาของ BIL สำหรับการแบ่ง Segment.....	21
4.8 ขนาดหน่วยความจำที่ใช้ในการแบ่ง Segment.....	21
4.9 เวลาในการสร้างตาราง 16 BIT ทุก Segment.....	22
4.10 เวลาในการค้นหา 16 BIT ทุก Segment	23
4.11 ความจำที่ใช้ 16 BIT ทุก Segment	23
4.12 ผลลัพธ์การแบ่ง Segment ตาม Requirement.....	24
4.13 เปรียบเทียบ Segment ที่มีความจำเท่ากัน.....	25

บทที่ 1

บทนำ

1.1 ที่มาและความสำคัญ

Bitmap Intersection Lookup เป็น Algorithm ที่คิดขึ้นมาเพื่อจัดการแอปพลิเคชันหรือสามารถนำไปใช้การจัดการแพ็คเกจบนอินเทอร์เน็ต เช่น การสื่อสารที่มีความเร็วสูงเป็นต้น ทำให้เกิดการกำหนดกฎการเข้าถึงที่เพิ่มขึ้นซึ่ง BIL คือ กฎที่มี Algorithm โครงสร้างที่ง่ายไม่ซับซ้อน มีความเร็วในการค้นหา ปรับปรุงกฎ ที่ดีและสูงขึ้น อีกทั้งยังใช้เนื้อที่จัดเก็บข้อมูลต่ำ โดยจะจัดโครงสร้างของกฎลงใน BIL tables และนำกฎที่มีลักษณะเป็นช่วง แปลงค่าให้อยู่ในรูป Prefix และ แบ่งเป็น Block ขนาดเล็กๆ เพื่อใช้สำหรับระบบ โดยผลลัพธ์ที่ได้ Algorithm จะเลือกกฎที่มีความสำคัญสูงสุดเป็นผลลัพธ์

แต่ยังไม่มีมีการทดสอบว่า Algorithm ที่ใช้หลักการ BIL นั้นมีวิธีการแบ่งชุดข้อมูลอย่างไรจึงจะมีประสิทธิภาพหรือเหมาะสมต่อหน่วยความจำที่มีตาม จึงมีจุดประสงค์เพื่อหาวิธีการแบ่งชุดข้อมูลที่เหมาะสมในรูปแบบเงื่อนไขที่แตกต่างกัน อีกทั้งเพื่อให้ทราบถึงปัญหาต่างๆ ของการแบ่งชุดข้อมูลที่จะเกิดขึ้น

1.2 วัตถุประสงค์

1. เพื่อทดสอบประสิทธิภาพของ BIL Algorithm
2. เพื่อหาวิธีการแบ่ง Segment ได้อย่างเหมาะสมตามหน่วยความจำที่มี
3. เพื่อให้ทราบถึงความเร็วการทำงานและหน่วยความจำที่ใช้ในการแบ่ง Segment ทุก

รูปแบบ 16 Bit

1.3 วิธีดำเนินการ

การวิจัยในโครงการนี้จะเริ่มต้นจากการพัฒนาโปรแกรมให้ทำงานตามเทคนิคอัลกอริทึมของ Bitmap Intersection Lookup และทำการทดลอง/ทดสอบ เพื่อพัฒนาและค้นหาอัลกอริทึมที่ใช้ในการแบ่ง Segment แต่ละรูปแบบและวิธีการต่างๆ โดยใช้โปรแกรม Google Colaboratory ใช้ภาษา PYTHON ในการพัฒนาและทดสอบเพื่อหาผลลัพธ์ว่ารูปแบบใดที่สามารถแบ่งชุดข้อมูลได้เหมาะสมและมีประสิทธิภาพมากที่สุดรวมไปถึงการศึกษาหาถึงผลกระทบที่อาจเกิดขึ้นจากการแบ่งชุดข้อมูลในรูปแบบต่าง ๆ

1.3.1 ศึกษาการทำงาน ของ Bitmap Intersection Lookup เพื่อพัฒนา Algorithm ในการแบ่ง Segment

1.3.2 พัฒนาและปรับปรุงการทำงาน ของ Bitmap Intersection ในการแบ่ง Segment

1.3.3 ทดสอบการทำงาน

1.3.4 เปรียบเทียบ สรุปผลในการทดลอง

1.4 ขอบเขตงาน

ศึกษาและพัฒนาการทำงานเทคนิค Bitmap Intersection Lookup เพื่อนำไปออกแบบอัลกอริทึมให้เหมาะสมสำหรับการแบ่งชุดข้อมูล รวมไปถึงศึกษาผลกระทบที่เกิดขึ้นจากการแบ่งชุดข้อมูลในแต่ละอัลกอริทึมและอาจมีการทดสอบประสิทธิภาพการทำงานในขั้นตอนสุดท้าย

สามารถนำไปประยุกต์ใช้ได้หลากหลายรูปแบบ เช่น การพัฒนา Search Engine ในการค้นหาข้อมูลที่มีจำนวนมากได้อย่างรวดเร็วยิ่งขึ้น หรือ จัดการกับพื้นที่เก็บข้อมูลให้เหมาะสม เป็นต้น อีกทั้งยังสามารถนำมาเปรียบเทียบกับอัลกอริทึมอื่น ๆ ได้ในอนาคต

1.5 ประโยชน์ที่คาดว่าจะได้รับ

สามารถนำเทคนิค Bitmap Intersection Lookup มาพัฒนาอัลกอริทึมให้มีการจัดการแบ่งชุดข้อมูล เพื่อให้ทราบถึง Algorithm ที่มีวิธีการแบ่งชุดข้อมูลในรูปแบบต่าง ๆ ที่มีเงื่อนไขในการทดสอบแตกต่างกัน เพื่อลดการใช้ทรัพยากรของระบบเครือข่ายได้มากที่สุด รวมไปถึงสามารถที่จะทราบถึงปัญหาของวิธีการแบ่งชุดข้อมูลในแต่ละรูปแบบว่าควรใช้วิธีการใดในการแบ่งชุดข้อมูล ให้มีประสิทธิภาพ ทำให้ไม่เกิดการใช้ทรัพยากรบนระบบงานที่สิ้นเปลืองเกินไป ทำให้ส่งผลกระทบต่อระบบงานที่น้อยที่สุด เพื่อทำให้ระบบงานที่ต้องการให้มีพื้นที่การใช้งานอย่างมีประสิทธิภาพในด้านการค้นหา และการจัดการพื้นที่ของข้อมูลได้อย่างเหมาะสมที่สุด

บทที่ 2

การจัดประเภทข้อมูลและแนวคิดของ Bitmap Intersection

Lookup สำหรับ Segmentation

2.1 แนวคิดการจัดประเภทแพ็คเก็ตและการแบ่ง Segment บนระบบเครือข่าย

การจัดประเภทแพ็คเก็ตมีจุดประสงค์เพื่อให้สามารถใช้งานระบบเครือข่ายได้อย่างมีประสิทธิภาพ เพื่อเพิ่มความปลอดภัย และเพื่อให้ระบบเครือข่ายมีความเร็วมากยิ่งขึ้น จึงต้องมีการพัฒนาอุปกรณ์ต่าง ๆ มาช่วยเพิ่มประสิทธิภาพ ซึ่งจำเป็นต้องมีการจัดการแพ็คเก็ตและการแบ่ง ชุดข้อมูลเพื่อสามารถแบ่งแยกแพ็คเก็ตในแต่ละตัวที่เข้ามาถูกจัดประเภทและแบ่งชุดข้อมูลได้อย่างเหมาะสม โดยการจัดประเภทแพ็คเก็ตนั้นมีความเร็วไม่มากพอ และยังมีผลกระทบที่เกิดจากการแบ่งชุดข้อมูล ในด้านของความถูกต้องของข้อมูล รวมไปถึงส่งผลกระทบต่อหน่วยความจำที่มีหากไม่มีการแบ่งชุดข้อมูลให้เหมาะสม

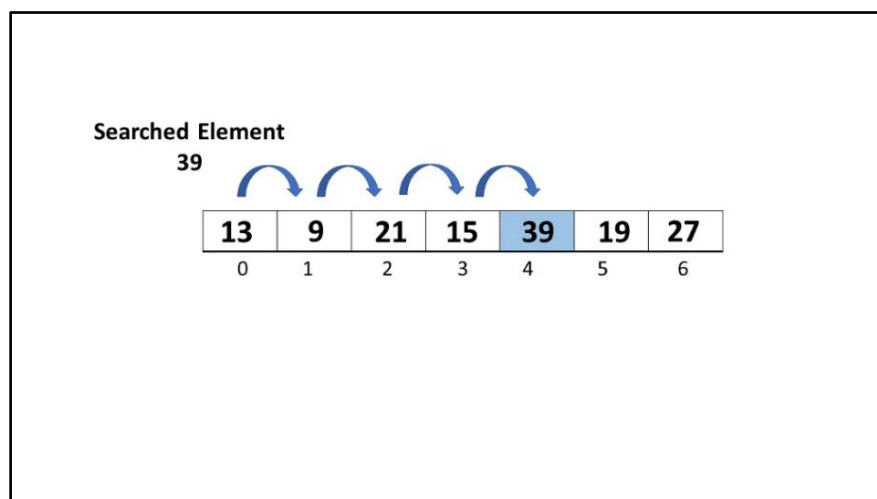
ในปัจจุบันการพัฒนาอุปกรณ์เครือข่ายให้มีความเร็วสูงแล้วเสถียรมากขึ้น โดยการนำเทคนิคการประมวลผล การจัดประเภทแพ็คเก็ตและการแบ่ง ชุดข้อมูล มาพัฒนาอุปกรณ์ Hardware เพื่อให้สามารถทำงานด้วยความเร็วสูงได้อย่างมีประสิทธิภาพ โดยการจัดประเภทแพ็คเก็ตที่คืนั้น ประสิทธิภาพในการค้นหาจะไม่ขึ้นอยู่กับจำนวนข้อมูล ในกรณีนี้จะระบุเป็นการจัดประเภทแพ็คเก็ตบน Firewall โดยการจัดประเภทแพ็คเก็ตบน Firewall คือการนำข้อมูลที่เข้ามาให้ตรงกับ Firewall Rules เป็นต้น และการแบ่ง ชุดข้อมูล คือการแบ่งชุดข้อมูลให้เหมาะสมกับข้อมูลที่รับเข้ามา การจัดประเภทข้อมูลบน Firewall Rule มีหน้าที่ตรวจสอบข้อมูลที่รับเข้ามาแล้วตรวจสอบเงื่อนไข โดยการส่งข้อมูลเข้าไปในเครือข่ายซึ่งมี Firewall Rules จำนวนมากและถูกแบ่งไว้เพื่อทำการตรวจสอบ คัดกรองข้อมูล การจัดประเภทข้อมูลและการแบ่ง ชุดข้อมูล จึงเป็นปัจจัยที่สำคัญที่ช่วยจัดการข้อมูลที่มากเกินไป ที่อาจทำให้เกิดปัญหาในการค้นหาล่าช้าได้ รวมไปถึงความถูกต้องของข้อมูลจากการค้นหาเมื่อเปรียบเทียบกับ Firewall Rules เป็นต้น

2.2 Search Algorithm

เป็นขั้นตอนสำหรับการค้นหาข้อมูลตามที่เรากำหนด โดยค้นหาข้อมูลจากตัวเก็บข้อมูลที่อยู่ในรายการหรือ List การค้นหาข้อมูลนั้นสามารถเป็นได้ทั้งข้อมูลที่เรียงลำดับ และไม่เรียงลำดับ ขึ้นอยู่กับประเภทข้อมูลของ Search Algorithm ที่ใช้ในการค้นหา ในตัวอย่างนี้จะยก Linear Search และ Binary Search มาเป็นตัวอย่าง

2.2.1 Linear Search

Linear Search เป็นการค้นหาแบบง่าย เป็นการค้นหาแบบเรียงลำดับหรือไม่เรียงลำดับก็ได้ โดยค้นทีละตัวไปจนครบ โดยเริ่มจากตัวแรกไปยังตัวสุดท้ายจนกระทั่งพบข้อมูลที่ต้องการค้นหา จากตัวอย่างดังรูป(2.1) ถ้าต้องการหาตัวเลข 39 โดยมีข้อมูลทั้งหมด 7 ตัว โดยวิธีนี้ใช้ขั้นตอนในการค้นหา 5 ขั้นตอนจึงพบข้อมูลที่ต้องการ



รูปที่ 2.1 Linear Search

```

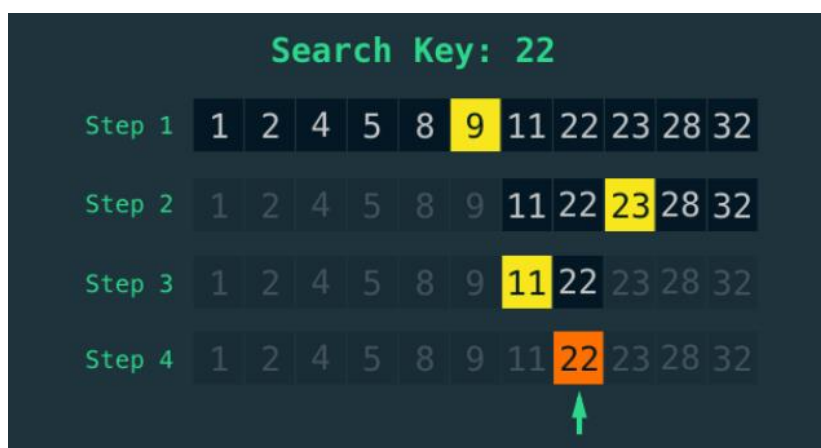
1  procedure linear_search (list, value)
2
3      for each item in the list
4          if match item == value
5              return the item's location
6          end if
7      end for
8
9  end procedure

```

รูปที่ 2.2 Pseudocode Linear Search

2.2.2 Binary Search

Binary Search เป็นการการค้นหาข้อมูลที่มีความเร็ว และไม่ซับซ้อนเหมาะกับข้อมูลที่มีปริมาณไม่มาก โดยข้อมูลที่จะนำไปค้นหาต้องถูกเรียงลำดับจากน้อยไปมากก่อน วิธีค้นหาคือ หาค่ากึ่งกลาง = $\lceil \frac{\text{โดยที่นำค่าที่น้อยที่สุด} + \text{ค่าที่มากที่สุด}}{2} \rceil$ แล้วหารด้วย 2 แล้วตรวจสอบดูว่าค่ากึ่งกลางนั้นมากกว่า หรือน้อยกว่าค่าที่เราต้องการค้นหา หากค่าที่ต้องการหานั้นน้อยกว่าค่ากึ่งกลาง จะตัดตำแหน่งข้อมูลหมดที่มีค่ามากกว่าตำแหน่งกึ่งกลาง หรือถ้าหากค่าที่ต้องการหามากกว่าค่ากึ่งกลาง จะตัดตำแหน่งข้อมูลหมดที่มีค่าที่น้อยกว่าตำแหน่งกึ่งกลาง ทำแบบนี้วนซ้ำๆ จนพบค่าที่ต้องการค้นหา ตัวอย่างดังรูปที่(2.3)



รูปที่ 2.3 Binary Search

2.3 เครื่องมือในการวิเคราะห์ผล

ในการวิเคราะห์ผลต้องทำการแปลงหน่วยต่าง ๆ ให้เท่ากันเพื่อต่อการเปรียบเทียบและสร้างตารางเพื่อสะดวก รวดเร็วในการวิเคราะห์ประสิทธิภาพในด้านต่าง ๆ

2.3.1 การแปลง Byte เป็น Bit มาจาก 1 Byte เท่ากับ 8 Bit

ตัวอย่างเช่น IPv6 มีขนาด 64Bit จะแปลงได้เท่ากับ 8 Byte

2.3.2 การแปลงหน่วย วินาที เป็น ไมโครวินาที มาจาก 1 วินาที เท่ากับ 1 ล้านไมโครวินาที

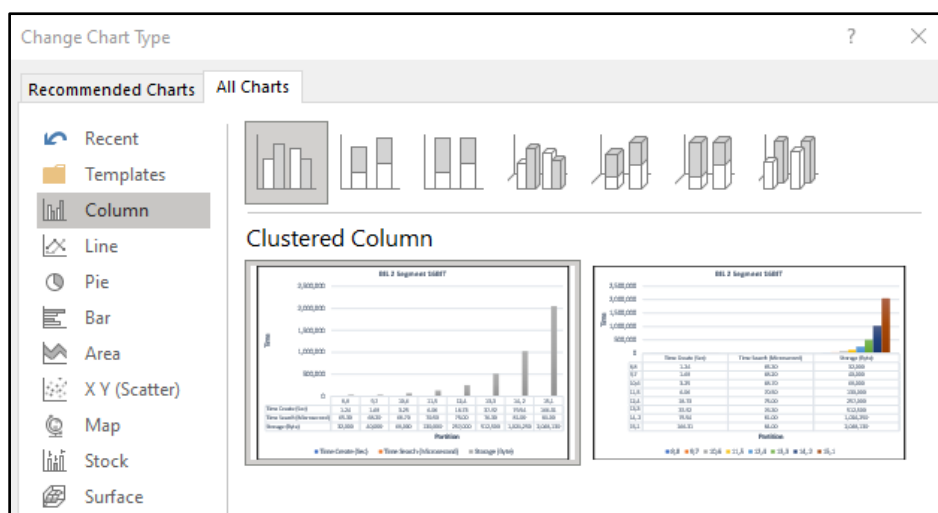
ตัวอย่างเช่น ใช้เวลาค้นหาข้อมูล 0.1 วินาที เท่ากับ 100,000 ไมโครวินาที

2.3.3 การแปลง ไบต์ (Byte) เป็น กิโลไบต์ (KB) มาจาก 1000 Byte เท่ากับ 1 กิโลไบต์

ตัวอย่างเช่น มีหน่วยความจำ 20,000 Byte เท่ากับ 20 KB

2.3.4 ในการสร้างตารางเก็บข้อมูลจะใช้ข้อมูลที่บันทึกมาสร้างเป็นตาราง

ตัวอย่างรูปแบบกราฟที่ใช้เปรียบเทียบข้อมูล เช่น Bar, Column, X Y Scatter



รูปที่ 2.4 ตัวอย่างการสร้างกราฟโดยใช้ Column

2.4 เทคโนโลยีและเครื่องมือที่ใช้ในการพัฒนา

2.4.1 ซอฟต์แวร์และเครื่องมือ (Software and Tools)

Google Colaboratory หรือ Colab เป็นบริการคลาวด์อีกหนึ่งบริการจาก Google Research เป็น IDE ที่อนุญาตให้ผู้ใช้เขียนซอร์สโค้ดในตัวแก้ไขและเรียกใช้จากเบราว์เซอร์ โดยเฉพาะอย่างยิ่ง รองรับภาษาการเขียนโปรแกรม Python และเน้นงานแมชชีนเลิร์นนิง การวิเคราะห์ข้อมูล โครงการการศึกษา ฯลฯ

2.4.2 ภาษาโปรแกรม (Programming Languages)

Python เป็นภาษาการเขียนโปรแกรมที่ใช้อย่างแพร่หลายในเว็บแอปพลิเคชัน การพัฒนาซอฟต์แวร์ วิทยาศาสตร์ข้อมูล และแมชชีนเลิร์นนิง (ML) นักพัฒนาใช้ Python เนื่องจากมีประสิทธิภาพ เรียนรู้ง่าย และสามารถทำงานบนแพลตฟอร์มต่างๆ ได้มากมาย



รูปที่ 2.5 เครื่องมือและภาษาที่ใช้เขียนโปรแกรม

บทที่ 3

การออกแบบ Algorithm และ ทดสอบประสิทธิภาพ

3.1 การวิเคราะห์ระบบงานเดิม

การที่ค้นหาข้อมูลต่างๆ จะมี Search Algorithm ที่เป็นพื้นฐานในการค้นหา โดยการค้นหาใน Firewall Rule จะมี Search Algorithm พื้นฐานเหมือนกัน โดยยกตัวอย่างการทำงานในการค้นหา Firewall Rules โดยกรณีที่ใช้ Linear Search ในการค้นหาจะใช้เวลาค้นหาที่นาน ซึ่งระบบงานดังกล่าวจะมีความล่าช้าเนื่องจากจำเป็นต้องตรวจสอบค่าที่รับเข้ามาทีละตัว โดยระบบงานเดิมนั้นการทำงานต่างๆ จะใช้เวลาในการค้นหาขึ้นอยู่กับจำนวนข้อมูลที่เข้ามา

3.2 หลักการทำงานของ Bitmap Intersection Lookup

การทำงานของ Bitmap Intersection Lookup หรือ BIL จะอยู่ในส่วนของ Bitmap Table โดยที่ Bitmap Table จะมีการสร้างตารางไว้เรียบร้อยแล้ว ในตารางจะประกอบไปด้วย Index และ Bitmap โดย Index จะมีขนาดเท่ากับจำนวนข้อมูลที่รับเข้าเท่ากับ 2 กำลัง N bit ในส่วนของ Bitmap คือคู่ค้นหาของค่า Value และ Masking ที่มีการแมชท์ (Match) กับ Rule โดยในข้อมูลแต่ละ field จะมีตำแหน่งของกฎซึ่งจะถูกแทนที่ด้วย Bit หากมี Bit ตรงกับกฎข้อใดให้นำค่า 1 ไปอัปเดตในตาราง หากไม่ตรงให้นำค่า 0 ไปอัปเดต ทำจนครบทุก Index แล้วใส่ใน Bitmap Table ในการค้นหาใน Firewall ที่นำ BIL มาใช้ เช่นหากมีข้อมูลที่รับเข้ามา หากอยากรู้ว่าค่าที่ต้องการค้นหาตรงกับ Firewall Rule ข้อใดให้นำ Packet header ของแต่ละ Field มาเปรียบเทียบกับ โดยใน Firewall Rule จะมี Rule ID จากนั้นทำการ Lookup หรือดึงข้อมูลที่รับเข้ามาตรวจสอบว่าตรงกับกฎข้อใดใน Bitmap Table แล้วทำการ List รายออกมา แล้วกำหนดค่าความสำคัญที่สุดแล้วนำกฎข้อนั้น ๆ มาใช้โดยในแต่ละกฎจะระบุว่ามี Action อย่างไร เช่น อนุญาตให้ใช้งาน (Allow) หรือ ไม่อนุญาตใช้งาน (Denied)

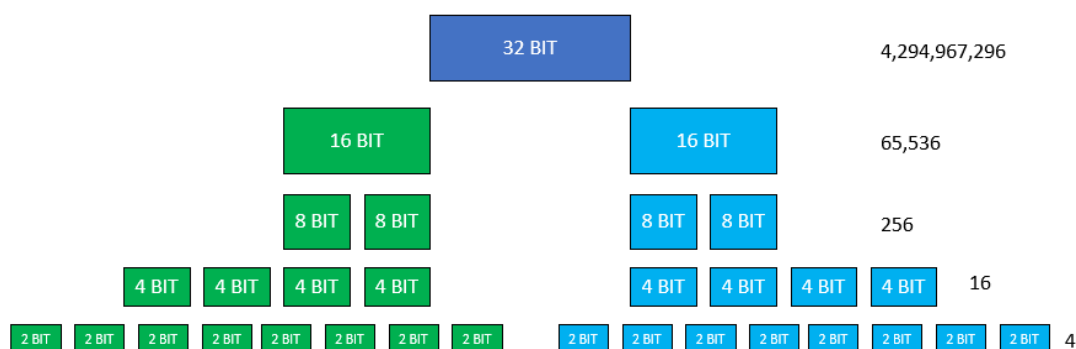
หาก BIL มีจำนวน Bit มาก ๆ จะทำให้มีขนาดของตารางที่มากตาม จำเป็นที่จะต้องแบ่งชุดข้อมูล หรือแบ่ง Segment โดยจะแบ่งออกเป็น page ซึ่งจะลดจำนวน bit ลงทำให้ขนาดของตารางมีขนาดเล็กลงเพื่อให้คอมพิวเตอร์สามารถทำงานได้อย่างมีประสิทธิภาพ ตัวอย่างการแบ่ง Segment เช่น มี 100 Bit โดยแต่ละ Page มีความกว้าง 16 Bit จะทำการแบ่ง Page ให้พอดี โดยการแบ่งค่าในแต่ละ Field จะมีวิธีการ โดยถ้า Field มีขนาด 8 Bit ค่าที่ได้ใน Field แต่ละตัวคือ 2^8 เท่ากับ 256 ตัว

3.3 เป้าหมายของการทำงาน Bitmap Intersection Lookup สำหรับ Segmentation

เป้าหมายในโครงการนี้ เป็นการเสนอวิธีแบ่ง Segment ตามหลักการ Bitmap Intersection Lookup โดยมีวิธีการออกแบบการทดลองในการทดสอบความถูกต้อง โดยเปรียบเทียบการใช้พื้นที่หน่วยความจำ ความเร็วในการค้นหา และความเร็วในการสร้างตาราง และ ทดลองตาม requirement โดยนำอัลกอริทึมสำหรับการแบ่ง Segment ที่จะทำการเสนอมาพิสูจน์ว่าแบ่ง segment อย่างไรให้เหมาะสมมากที่สุดต่อหน่วยความจำที่มีประสิทธิภาพดีที่สุด แล้วนำการแบ่ง Segment มาเปรียบเทียบกันทุกรูปแบบที่เป็นได้

3.4 แนวคิดการแบ่ง Segment

การแบ่ง Segment นั้นทำเพื่อลดขนาดหน่วยความจำ และทำให้ขนาดตารางในแต่ละ Segment มีขนาดเล็กลง เพื่อให้สามารถสร้างตารางได้ไวมากขึ้น เช่น 1 field มีขนาด 8 Bits มีขนาด 256 แบ่งเป็น 4 Segment โดยแบ่งเป็น Segment ละ 4 Bit จะมีขนาด 32 จะเห็นได้ว่าการแบ่ง Segment ทำให้ขนาดหน่วยความจำลดลง และเพิ่มความเร็วในการสร้างตาราง แต่ในเรื่องผลกระทบที่เกิดจากการแบ่ง Segment นั้นอาจเกิดปัญหาได้ เช่น หากมีการแบ่ง Segment มากเกินไปอาจทำให้เกิดความล่าช้าในการค้นหาข้อมูลเพราะมีจำนวน BIL Table จำนวนมาก



รูปที่ 3.1 ภาพตัวอย่างการแบ่ง Segment

3.5 หลักการแบ่ง Segment

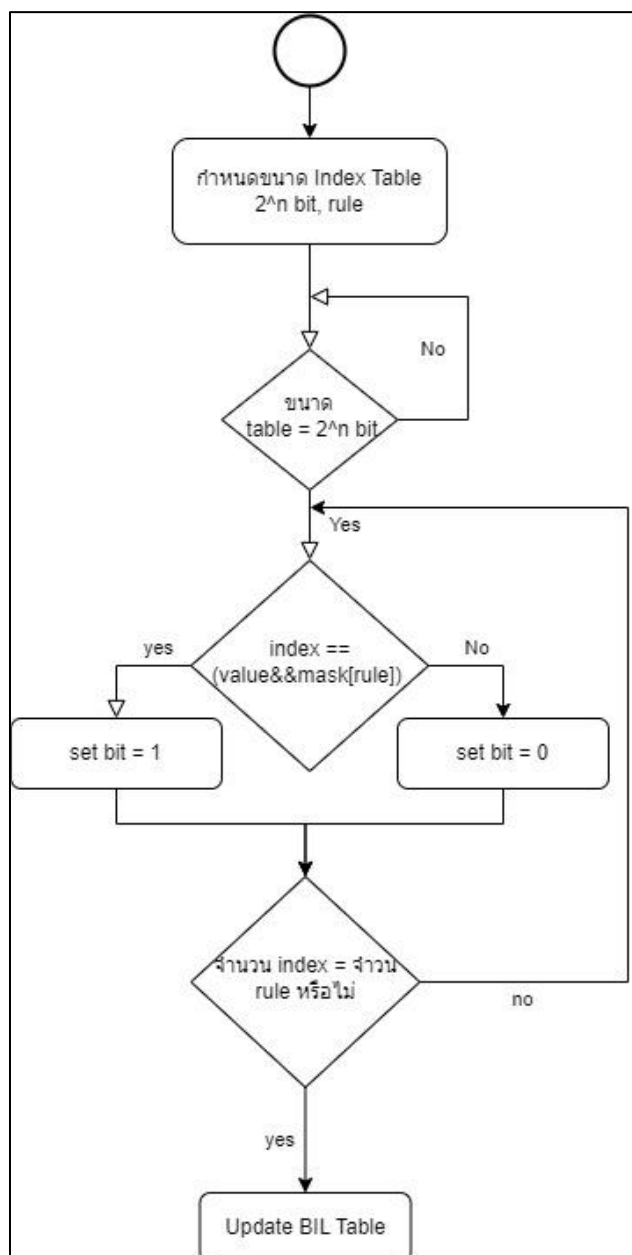
การแบ่ง Segment นั้นต้องคำนึงถึงขนาดหน่วยความจำที่มีอยู่ หากแบ่งส่วนออกมาก หน่วยความจำที่ต้องใช้จะยิ่งน้อยลงเพราะขนาดของ Bit ในแต่ละ Segment น้อยลง หากขนาดของ Bit ในแต่ละ Segment เท่ากันจะใช้เวลาสร้างและขนาดหน่วยความจำน้อยที่สุด

3.6 แนวคิดของการศึกษาวิจัย

เนื่องด้วยการจัดทำโครงการนี้เป็นการนำการศึกษาและพัฒนาอัลกอริทึม Bitmap Intersection Lookup ที่ได้ทำการพัฒนาและศึกษามาก่อน โดยมีชื่อโครงการว่า การศึกษาและพัฒนาอัลกอริทึม Bitmap Intersection Lookup สำหรับ Packet Classification โดยทฤษฎีแนวคิดที่ได้ใช้ไปแล้วนั้น เป็นการทดลองการทำงานของ Bitmap Intersection Lookup Algorithm แบบไม่แบ่งชุดข้อมูลโดยสรุปได้ว่า เวลาในการค้นหามีความเร็วมากกว่า Linear Search แต่เวลาในการสร้างตารางและขนาดของหน่วยความจำจะมากเนื่องจากใช้พื้นที่ในการจัดเก็บข้อมูลจำนวนมาก ดังนั้นผู้ศึกษาจะพัฒนาโดยใช้ Algorithm จากโครงการเดิมมาศึกษาโดยมุ่งเน้นไปที่การหาวิธีการที่เหมาะสมของการแบ่งชุดข้อมูล เพื่อศึกษาหาวิธีการที่แบ่งชุดข้อมูลที่เหมาะสมกับขนาดหน่วยความจำที่มี จะมีการทดสอบในด้านต่าง ๆ ไม่ว่าจะเป็น ความเร็วสร้างตาราง ความเร็วในการค้นหา และสุดท้ายการทดสอบเพื่อวัดขนาดหน่วยความจำที่ใช้ของการแบ่งชุดข้อมูลในแต่ละรูปแบบที่แตกต่างกัน เพื่อให้สามารถนำไปประยุกต์ใช้หรือพัฒนาอัลกอริทึมในอนาคตได้ต่อไป

3.7 ภาพรวมของระบบ

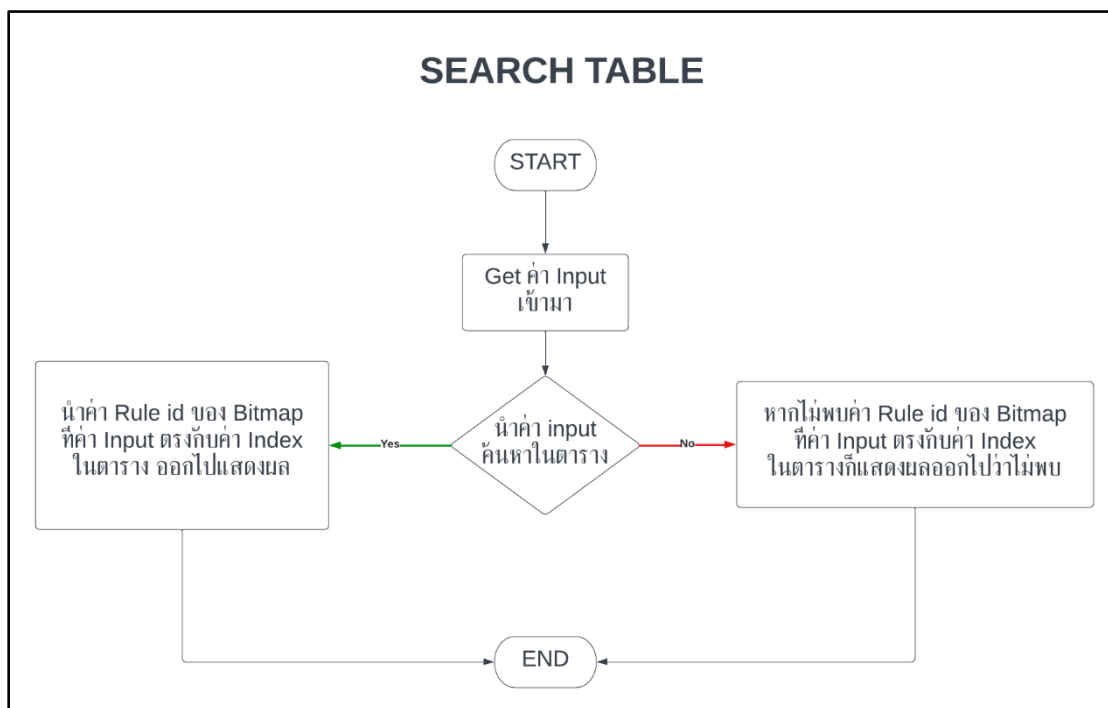
1. แผนภาพการทำงาน BIL Create/Update Table โดยการทำงานจะเริ่มรับ Input ที่มีขนาด 2^N Bit แล้วรับค่าไว้เพื่อนำไป Match ใน index แล้วนำไปสร้างตารางตาม Input ที่รับเข้ามา แล้วนำ Index ไป Match กับ Rule ตามจำนวน N Rules หาก Match ตรงกันให้อัพเดทค่า 1 หากไม่ตรงให้อัพเดทค่า 0 ทำจนครบทุก Index แล้วนำไปอัปเดตใน BIL Table



รูปที่ 3.2 Flowchart การทำงานของ BIL Update Table

2. แผนภาพการทำงานของการทำงาน Search Table

ในการ Search ขั้นตอนแรกจะนำ Input ที่รับเข้าจากนั้นนำไปตรวจใน BIL Table ที่มีการสร้างไว้อยู่แล้ว จากนั้นนำค่าที่รับเข้ามานำไปตรวจสอบใน BIL Table โดยจะนำ Header ที่มีค่าของ Value, Mask ไปตรวจสอบว่าตรงกับ Index ข้อใดใน BIL Table หากเจอให้ Return Rule Id



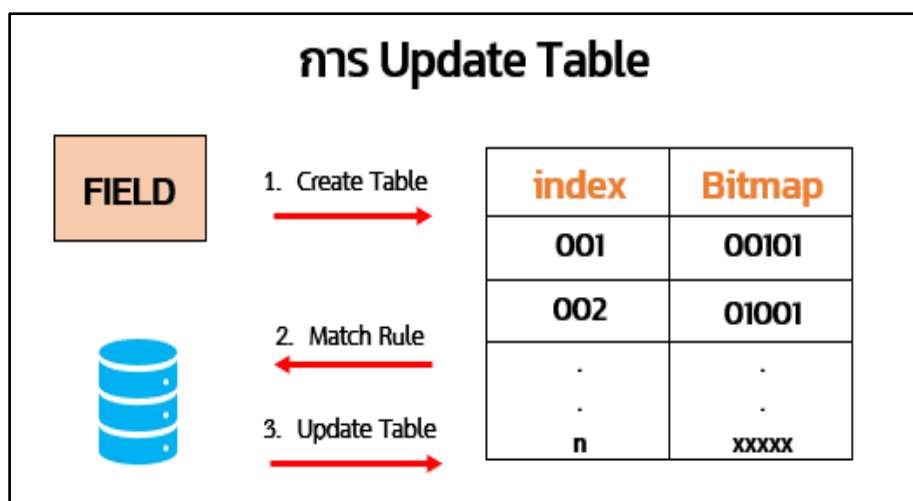
รูปที่ 3.3 Flowchart การทำงานของการ Search

3.8 การพัฒนา Algorithm

การสร้างอัลกอริทึมจะแบ่งเป็น 2 วิธีหลัก คือ

3.8.1. การ Update Table

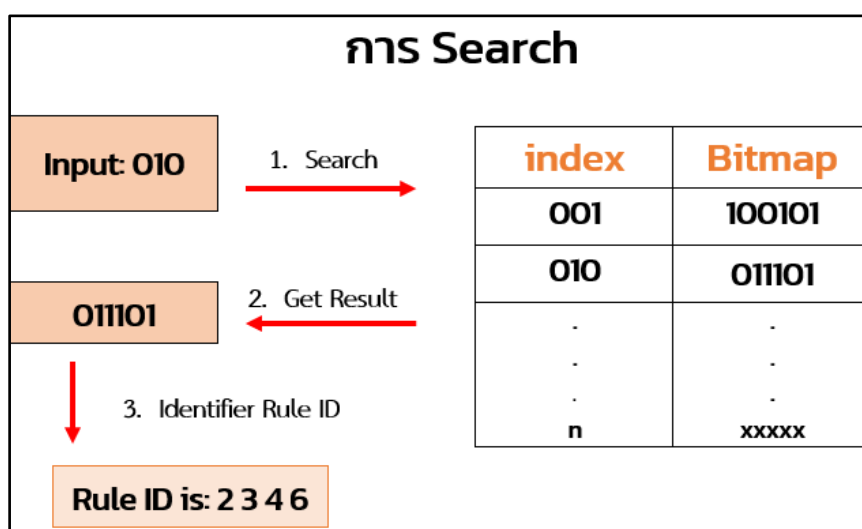
1. Create Table สร้างตาราง Bitmap Table เช่น มีข้อมูลขนาด 8 Bits สร้าง Table 1 field มีค่า index 2^8
2. Match Rule กำหนดคู่ Pair โดยใน Pair จะประกอบด้วย value กับ mask โดยจะนำค่า Index ไป match rules โดยตรวจสอบทีละ Rules ตั้งแต่ Rule ข้อ 1 ไปจนถึงข้อสุดท้าย
3. Update Table จากขั้นตอนที่ 2 ในกรณีที่ Rule Match กับ Index ให้นำค่า 1 ไปอัปเดตใน Table ในกรณีที่ ไม่ Match ให้นำค่า 0 ไปอัปเดตใน Table แล้วทำไปเรื่อย ๆ จนกว่าจะค้นหาครบทุก Index



รูปที่ 3.4 ขั้นตอนการทำงานของ BIL ในการ Update Table

3.8.2. การ Search ในการแบ่งชุดข้อมูลแต่ละรูปแบบ

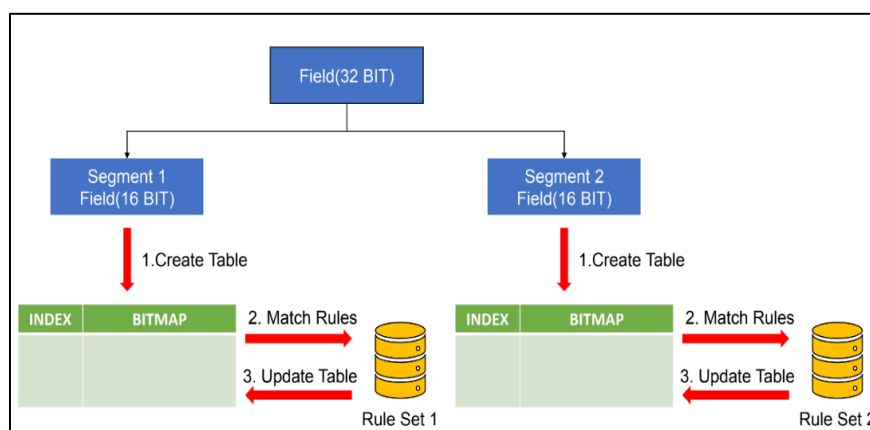
1. Get input & Search Rule นำค่า input ที่รับมาไปค้นหาใน Bitmap Table
2. นำผลลัพธ์ที่ได้มาระบุ Rule ID
3. ทำการทดสอบความเร็วในการ Search โดยกำหนดขนาดในการแบ่งชุดข้อมูลแต่ละรูปแบบ



รูปที่ 3.5 ขั้นตอนการทำงานของ BIL ในการ Search

3.8.3 วิธีการสร้างตารางแบบแบ่ง Segment

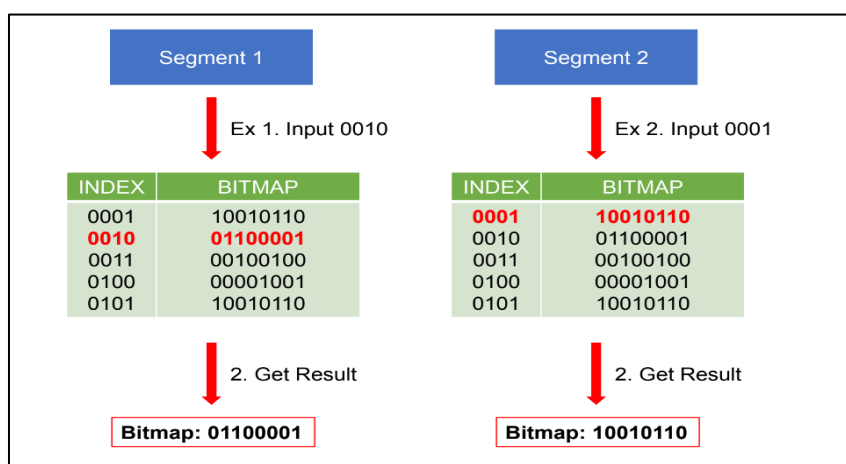
การสร้าง BIL Table ต้องกำหนดจำนวน BIT แต่ละ Field โดยกำหนดไม่เท่ากันได้ ในการแบ่งชุดข้อมูลจะนำจำนวน BIT ใน Field มาแบ่งชุดข้อมูลก่อน แล้วนำมาสร้างเป็น BIL Table



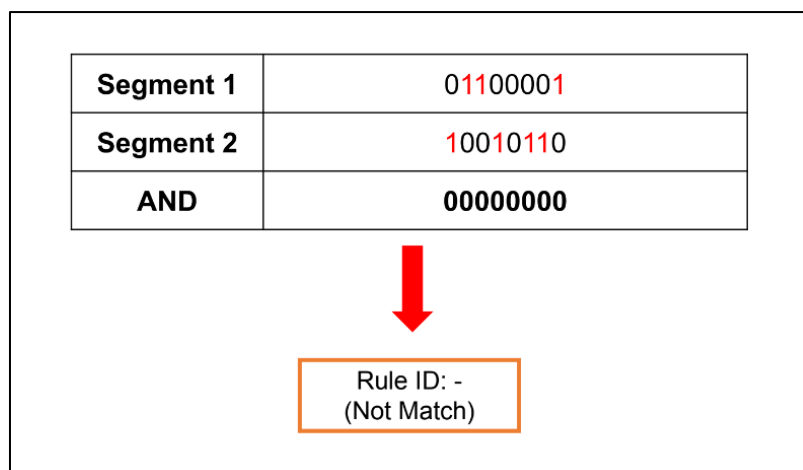
รูปที่ 3.6 ขั้นตอนการทำงานของ BIL ในการ Search สำหรับการแบ่งชุดข้อมูล

3.8.4 วิธีการค้นหาแบบแบ่ง Segment

การค้นหาแบบแบ่ง Segment ในขั้นตอนแรกจะรับ Input เข้ามาแล้วนำไปตรวจสอบในแต่ละ Table โดยเริ่มจาก Table แรก จนหมดจำนวน Table ตามจำนวนที่แบ่งชุดข้อมูล จากนั้นนำผลลัพธ์ทั้งที่ค้นหาเจอของแต่ละ Table นำมาทำตรรกะ AND กันหาก Bit ไหนตรง มีค่าเท่ากับ 1 ก็จะถูกเช็กเป็น หาก Bit ต่างกันหรือเป็น 0 ก็จะเท่ากับ 0 แล้วจะได้ผลลัพธ์สุดท้ายว่ามี Rule ID ตรงกับข้อใด



รูปที่ 3.7 การค้นหาใน BIL 2 Table



รูปที่ 3.8 ผลลัพธ์การค้นหาคู่ใน BIL 2 Table การ AND

3.9 แนวคิดการแบ่ง Segment

การแบ่งชุดข้อมูล นั้นทำเพื่อลดขนาด Storage ที่มากเกินไป และ ใช้ลดระยะเวลาในการ Update Table โดยมีสมการที่สามารถบอกได้ว่าควรแบ่งชุดข้อมูลโดยมีจำนวน bit เท่าใด และมีขนาดของ bit เท่าใดเพื่อให้เหมาะสมกับความจำที่มี เพื่อให้ได้ประสิทธิภาพมากที่สุด

3.9.1 ตัวแปรที่เกี่ยวข้อง

- F = จำนวน bits ใน Field
- TRreq = ขนาดหน่วยความจำของ BIL-Table ทั้งหมด กำหนดจาก requirement หน่วยเป็น Bit
- R = จำนวน Rule
- Treq = ขนาดหน่วยความจำของ BIL Table ต่อ Rule หน่วยเป็น bits
- Tall = ขนาดของตาราง BIL
- BSint = จำนวน bits ในแต่ละ Segment สำหรับ BIL-Table
- NSint = จำนวน Segment ของ BSint
- BSres = จำนวน Bit ที่เหลือจากการแบ่ง Segment ของ BIL ($BSres \geq 0$)

3.9.2 สมการ

$$3.1 \text{ Treq} = \frac{\text{TRreq}}{R}$$

$$3.2 F = (\text{NSint} * \text{BSint}) + \text{BSres}$$

จาก Requirement

$$3.3 \text{ Tall} \leq \text{Treq}$$

$$3.4 \text{ Tall} = \text{NSint} * 2^{\text{BSint}} + 2^{\text{BSres}}$$

ลดความซับซ้อนของ Tall เพื่อให้หา BSint และ NSint, NS = จำนวนของ Segment

$$3.5 F = \text{NS} * \text{BS}$$

$$3.6 \text{ Tall} = \text{NS} * 2^{\text{BS}}$$

จากสมการ (3.5)

$$3.7 \text{ NS} = \frac{F}{\text{BS}}$$

$$3.8 \text{ Tall} = \left(\frac{F}{\text{BS}}\right) 2^{\text{BS}}$$

BS ต้องเป็นไปตามเงื่อนไขดังนี้

$$\text{BS} = \text{int and } 1 \leq \text{BS} \leq F$$

BS ควรมีขนาดใหญ่ที่สุดที่เป็นไปได้ แต่น้อยกว่า Treq

$$\text{Tall} \leq \text{Treq}$$

$$\left(\frac{F}{\text{BS}}\right) 2^{\text{BS}} \leq \text{Treq}$$

$$\left(\frac{1}{\text{BS}}\right) 2^{\text{BS}} \leq \frac{\text{TRreq}}{F}$$

$$3.9 \text{ BS} - \log_2 \text{BS} \leq \log_2 \frac{\text{TRreq}}{F}$$

BSint = จำนวนเต็มที่ยิ่งใหญ่ที่สุดของ BS ที่ตรงกับสมการ (3.9)

$$\text{ซึ่งค่าสูงสุดคือ } \log_2 \frac{\text{TRreq}}{F}$$

$$3.10 \text{ NSint} = \text{floor}\left(\frac{F}{\text{BSint}}\right)$$

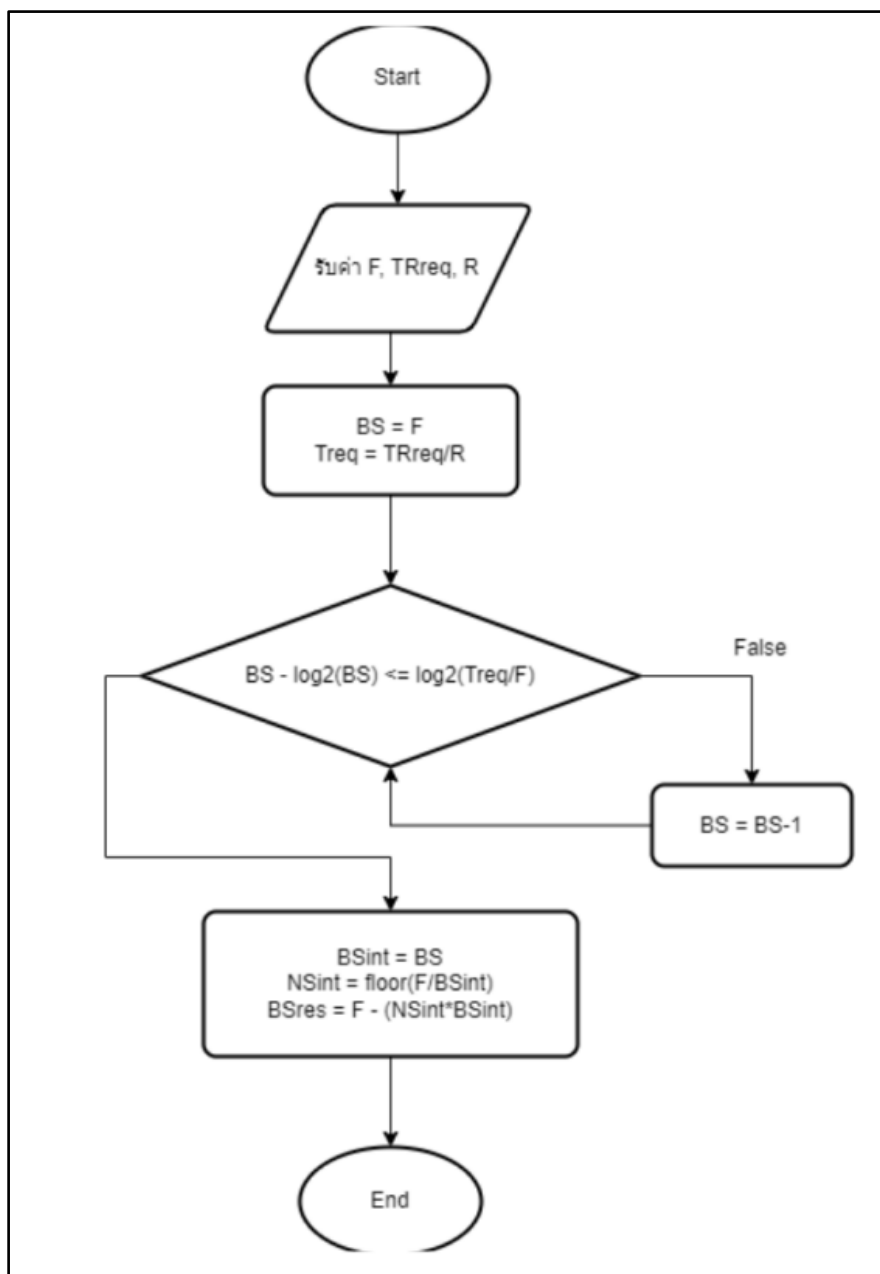
$$3.11 \text{ BSres} = F - (\text{NSint} * \text{BSint})$$

ตรวจสอบผลกับ (3.3)

$$\text{Tall} = \text{NSint} * 2^{\text{BSint}} + 2^{\text{BSres}} \text{ และ } \text{Tall} \leq \text{Treq}$$

3.9.3 ขั้นตอนการหา Segment

1. รับค่าตัวแปร F , TR_{req} , R
2. ให้ $BS = F$ และ $Treq = TR_{req}/R$
- 3.. จากนั้นนำมาคำนวณถ้า $BS - \log_2(BS) \leq \log_2(Treq/F)$ ถ้าไม่ให้ $BS-1$
- 4.. ถ้าผ่านให้ $BS_{int} = BS$, $NS_{int} = \text{floor}(F/BS_{int})$ และ $BS_{res} = F - (NS_{int} \cdot BS_{int})$



รูปที่ 3.9 กระบวนการออกแบบการแบ่ง Segment

บทที่ 4

การทดลองและวิเคราะห์

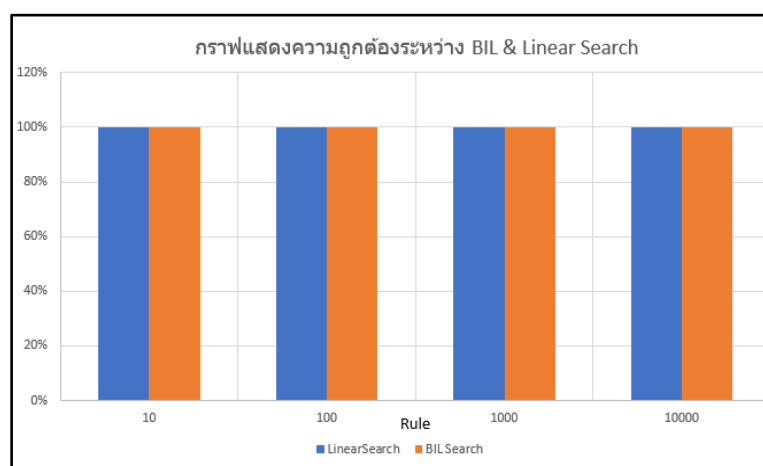
4.1 ออกแบบการทดลอง

การทดลองของ Bitmap Intersection Lookup จะแบ่งออกได้ดังนี้ โดยเริ่มจากทดสอบความถูกต้อง จากนั้นทดสอบประสิทธิภาพการทำงานโดยวัดความเร็วการทำงาน หน่วยความจำที่ใช้ สุดท้ายเป็นการทดสอบว่าควรแบ่ง Segment อย่างไรจึงจะเหมาะสมที่สุด ในอัลกอริทึมที่เสนอนั้น เป็นการบ่งบอกว่าต้องแบ่งเท่า ๆ กันทุกจึงจะได้หน่วยความจำน้อยที่สุด และมีความเร็วการทำงานดีที่สุด และเหมาะสมต่อหน่วยความจำ หรือตาม Requirement ที่กำหนด จากนั้นนำไปเปรียบเทียบกับทุกรูปแบบที่มีการแบ่ง Segment หากมีหน่วยความจำที่เท่ากัน แล้วการแบ่ง Segment แบบเท่ากันจะมีประสิทธิภาพดีที่สุดหรือไม่

4.2 การทดลองความถูกต้อง

การทดสอบของ Algorithm Linear Search และ BIL Search แบบการแบ่งเพียง 1 Segment และ เป็นการค้นหาของกฎเพียงคู่ค้นหาของ Value/Mask เพียงคู่เดียว ทดสอบด้วย 10 rule, 100 rule, 1000 rule, 5000 rule, 10,000 rule

4.2.1 ความถูกต้อง BIL กับ Linear Search



รูปที่ 4.1 กราฟแสดงความถูกต้องของ BIL & Linear Search

จากผลการทดลองรูปที่ 4.1 Linear Search และ BIL Search ที่มีจำนวน Rule และ ขนาด Index ที่เท่ากัน จะแสดงผลลัพธ์เหมือนกันถูกต้อง แต่การทดลองนี้เมื่อมีข้อมูลจำนวนมากๆ จะทำให้ใช้เวลาในการทดสอบที่นานขึ้น และจะทำให้ใช้พื้นที่ของหน่วยความจำข้อมูลใน BIL Algorithm และ Linear Search ใ้มากขึ้นตามจำนวน INDEX และ Rule ตามไปด้วย

```

✓ [67] linear_search_result == rule1000_segment1_boi16_bil_search_result
    True

✓ [56] linear_search_result[0:10]
    [33, 233, 33, 233, 33, 343, 33, 343, 186, 233]

[57] rule1000_segment1_boi16_bil_search_result[0:10]
    [33, 233, 33, 233, 33, 343, 33, 343, 186, 233]

```

รูปที่ 4.2 ผลลัพธ์ความถูกต้องของ BIL Search และ Linear Search

จากรูปที่ 4.2 แสดงผลลัพธ์แต่ละ INDEX ดังตัวอย่างจะนำมาแสดงตั้งแต่ตำแหน่งที่ 0:10 จะเห็นได้ว่าในตำแหน่งแต่ละ INDEX จะบ่งบอกถึงกฎที่มีค่าเท่ากับ 1 หรือตรงกับกฎ ดังภาพจะเห็นว่าตำแหน่ง INDEX ที่ 1 จะมีค่าเท่ากับ 33 ซึ่งก็ตรงกับกฎข้อที่ 33 เมื่อนำไปเปรียบเทียบกับ BIL Search แบบไม่แบ่ง Segment ก็จะมีผลลัพธ์เหมือนกัน

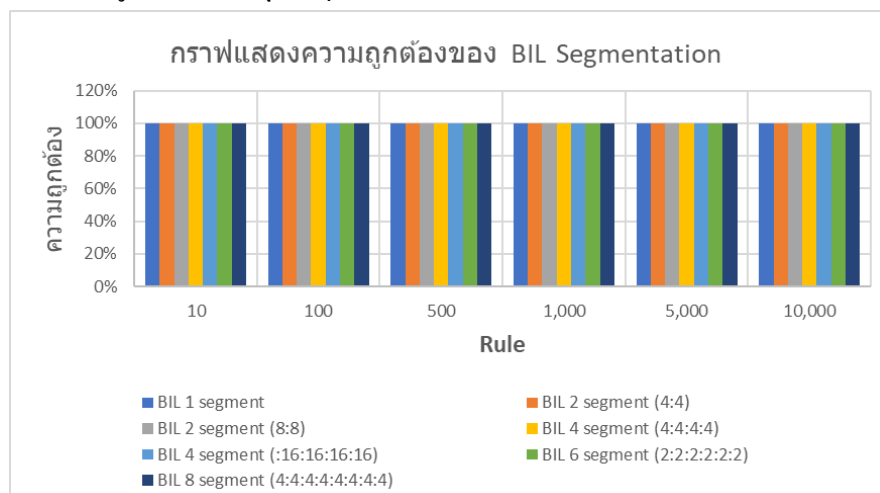
4.2.2 การทดสอบความถูกต้องจากการแบ่ง Segment ของ BIL เทียบกับ Linear Search

การทดสอบความถูกต้องของ BIL ที่มีการแบ่ง Segment ในรูปแบบต่าง ๆ ที่มีการเทียบกับ Linear Search โดย BIL จะมีข้อกำหนดดังนี้ 1 Segment, 2 Segment, 4 Segment, 6 Segment, 8 Segment, 16 Segment

	linear	1seg	2seg	4seg	8seg	16seg
0	33	33	33	33	33	33
1	233	233	233	233	233	233
2	33	33	33	33	33	33
3	233	233	233	233	233	233
4	33	33	33	33	33	33
...
65531	207	207	207	207	207	207
65532	75	75	75	75	75	75
65533	75	75	75	75	75	75
65534	38	38	38	38	38	38
65535	207	207	207	207	207	207

รูปที่ 4.3 ความถูกต้องของ BIL Segmentation และ Linear Search

ผลการทดลองจากรูปที่ 4.3 จากรูปจะเห็นได้ว่าการแบ่ง Segment ในรูปแบบที่กำหนด ผลลัพธ์ของการทดลองนั้นมีความถูกต้อง จึงสามารถสรุปได้ว่า Algorithm BIL ที่มีการแบ่ง Segment เมื่อนำไปเปรียบเทียบกับ Linear Search จะผลลัพธ์ที่เหมือนกันทำให้อัลกอริทึมที่ใช้นั้น พิสูจน์ได้ว่ามีความถูกต้องของกฎ ในทุกๆ INDEX

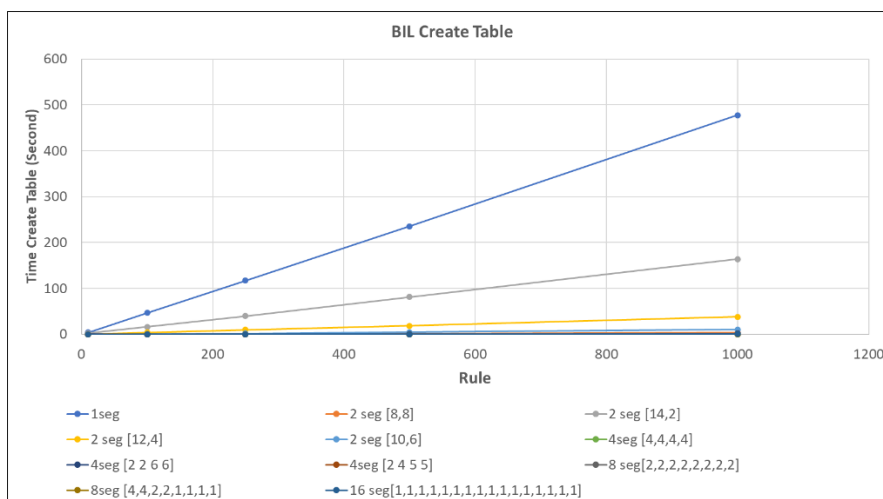


รูปที่ 4.4 กราฟแสดงความถูกต้องระหว่าง BIL Segmentation และ Linear Search

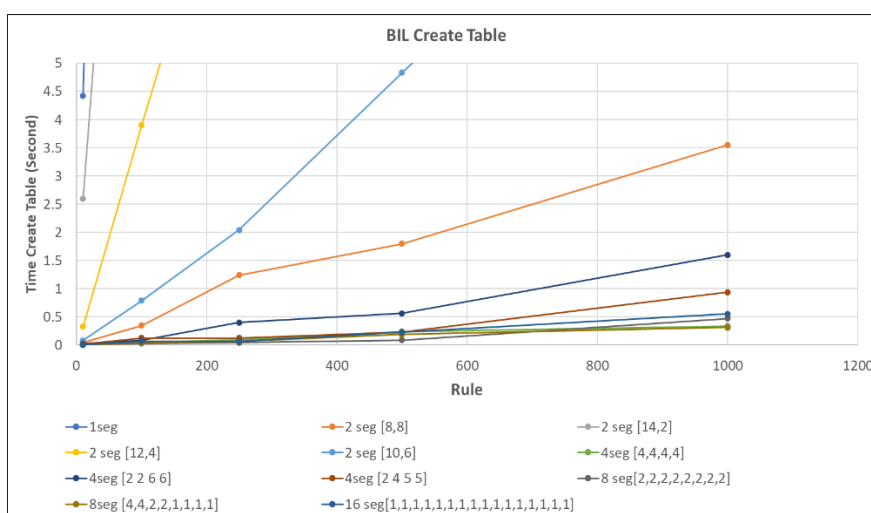
จากรูปที่ 4.4 เป็นการแสดงกราฟภาพรวมของความถูกต้องระหว่าง BIL Algorithm ที่มีการแบ่ง Segment เมื่อเปรียบเทียบกับ Linear Search นั้นเห็นได้ว่ามีความถูกต้องเหมือนกันทั้งหมด ทำให้เราสามารถนำอัลกอริทึมนี้ไปพัฒนาเพื่อทดสอบในการแบ่ง Segment ในขั้นต่อไป

4.3 การทดลองความเร็วในการสร้างตาราง BIL

ในการทดลองความเร็วในการสร้างตารางของ Bitmap Intersection Lookup โดยทำการทดสอบทั้ง 5 รูปแบบ ได้แก่ 1 Segment, 2 Segment, 4 Segment, 8 Segment และ 16 Segment โดยมีจำนวน Rule 100, 250, 500, 1000 โดยจะวัดความเร็วการสร้างตาราง 5 ครั้งและนำมาหาค่าเฉลี่ยจากการนำผลรวมเวลาสร้างตารางทั้ง 5 ครั้ง มาหาร 5 แล้วนำความเร็วสร้างตารางครั้งที่ 1 Rules ที่ N บวกกับ ผลการทดลองครั้งที่ Rules ที่ N ทำซ้ำจนครบ แล้วนำมาหารกับจำนวนที่มีการทดลองทุกครั้ง



รูปที่ 4.5 ความเร็วในการสร้างตารางของ BIL

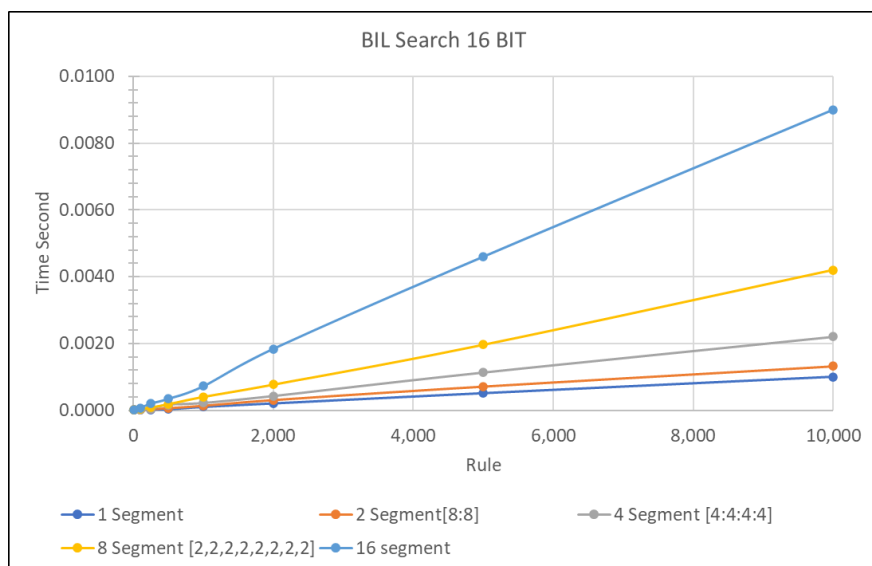


รูปที่ 4.6 ภาพขยายความเร็วในการสร้างตารางของ BIL

จากรูปที่ 4.5 และ 4.6 จะเห็นได้ว่า 1Segment จะมีเวลาในการสร้างตารางนานที่สุด เพราะขนาดตารางคือ $2^{16} = 65,536$ Index แต่ 16Segment จะเร็วที่สุดเพราะการแบ่ง Segment ช่วยลดขนาดของ Index จึงทำให้เวลาลดลง เช่น 4 Segment แบ่งแบบ (4:4:4:4) สร้างตารางที่มี $(2^4) 4$ ตาราง เท่ากับ 64 Index มีค่าเท่ากับ 16 Bit เหมือนกันแต่ 1 Segment จะใช้เวลาสร้างเร็วกว่า 4 Segment มาก หาก Rule มากจะใช้เวลามากตามไปด้วย

4.4 การทดลองความเร็วในการ Search ของ BIL

โดยการทดลองความเร็วในการ Search ใน BIL Table ของ Field ขนาด 16Bits และมีการแบ่ง 1 Segment, 2 Segment, 4 Segment, 8 Segment และ 16 Segment โดยทดลองหาค่าเฉลี่ยความเร็วในการค้นหา 5 รอบ โดยนำผลรวมการค้นหาทั้ง 5 รอบ มารวมกันแล้วนำไปหาร 5 จะได้เวลาเฉลี่ยของแต่ละ Segment ที่มีการแบ่ง

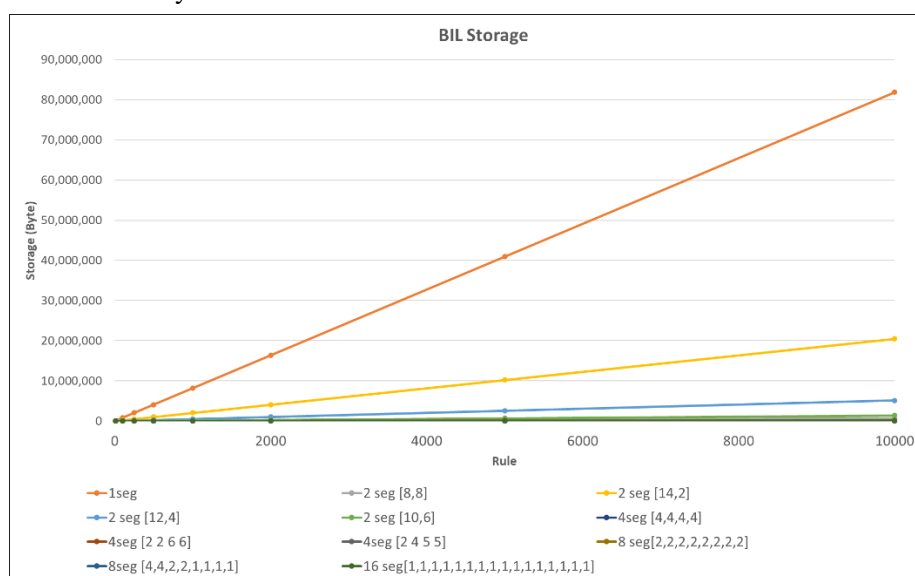


รูปที่ 4.7 ความเร็วในการค้นหาของ BIL สำหรับการแบ่ง Segment

จากการทดลองที่ 4.4 เห็นได้ว่าการค้นหา 1 Segment จะเร็วที่สุด และจะช้าลงเมื่อมีการแบ่ง Segment มากขึ้นตามลำดับ เนื่องจากการค้นหาตั้งแต่ 2Segment ขึ้นไปต้องนำค่าทุก Segment มา AND กันจึงจะได้คำตอบ ทำให้เวลาช้ากว่า 1 Segment มากขึ้นเรื่อย ๆ ตาม Segment ที่ต้อง AND กัน จะกราฟเห็นได้ว่าทุก Segment เวลาจะไม่ขึ้นอยู่กับจำนวนข้อมูลมากนักแต่จะใกล้เคียงกัน

4.5 ขนาดหน่วยความจำที่ใช้ของ BIL

ขนาดหน่วยความจำที่ใช้ของ BIL โดยเปรียบเทียบ 5 รูปแบบ ได้แก่ BIL 1 Segment, 2 Segment, 4Segment, 8Segment และ 16 Segment โดยขนาดหน่วยความจำคำนวณได้วิธีดังนี้ นำผลรวมของ 2 ยกกำลัง Bits ทุก Segment แล้วนำไปคูณกับจำนวน Rules แล้วนำไปหารด้วย 8 โดยหน่วยความจำจะเป็น Byte



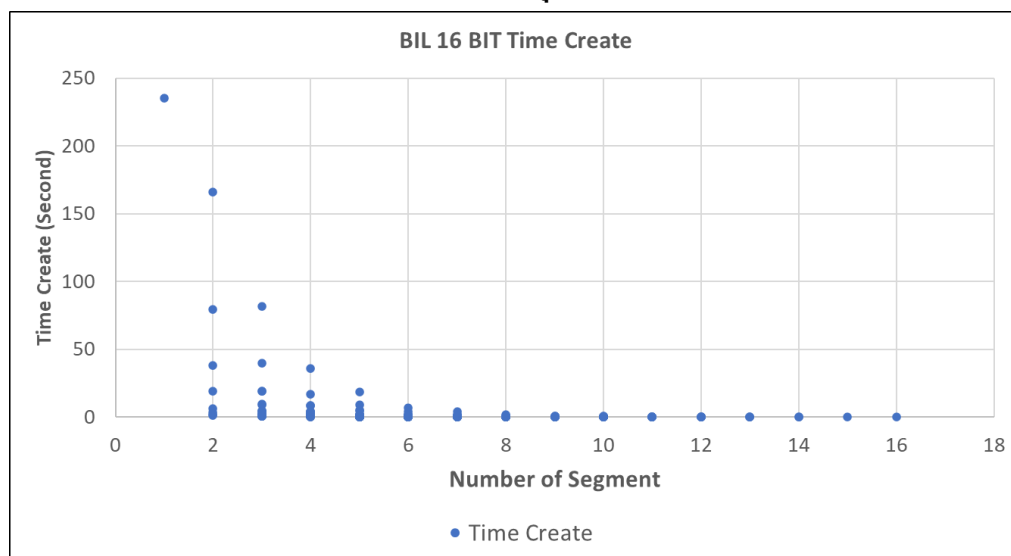
รูปที่ 4.8 ขนาดหน่วยความจำที่ใช้ในการแบ่ง Segment

จากการทดลองที่ 4.5 เป็นการเปรียบเทียบหน่วยความจำสำหรับการแบ่ง Segment หลายรูปแบบ จะเห็นได้ว่าหน่วยความจำที่ใช้ของ 1 Segment จะมีขนาดมากที่สุด และ 16 Segment มีขนาดหน่วยความจำน้อยที่สุด เนื่องจาก 1 Segment จะถูกใช้หน่วยความจำในระบบทั้งหมด ทั้งนี้หากมี Rules มาก หน่วยความจำจะมากตามไปด้วย ดังนั้นยิ่งแบ่ง Segment มากขนาดหน่วยความจำที่ใช้จะน้อยลงเท่านั้น สรุปได้ว่าจำนวน Segment กับ Rules มีผลต่อหน่วยความจำ

4.6 ทดสอบการทำงานทุก Segment 16 BIT

เป็นการทดสอบความเร็วในการทำงานทั้งการสร้างตาราง ความเร็วการค้นหาและหน่วยความจำที่ใช้ทุก Segment ของ 16 Bit ทุกรูปแบบที่เป็นไปทั้งแบ่ง Segment เท่ากันและไม่เท่ากัน โดยจะทดสอบตั้งแต่ 1 Segment ไปจนถึง 16 Segment มีจำนวน 500 Rules แล้วนำมาเปรียบเทียบกัน

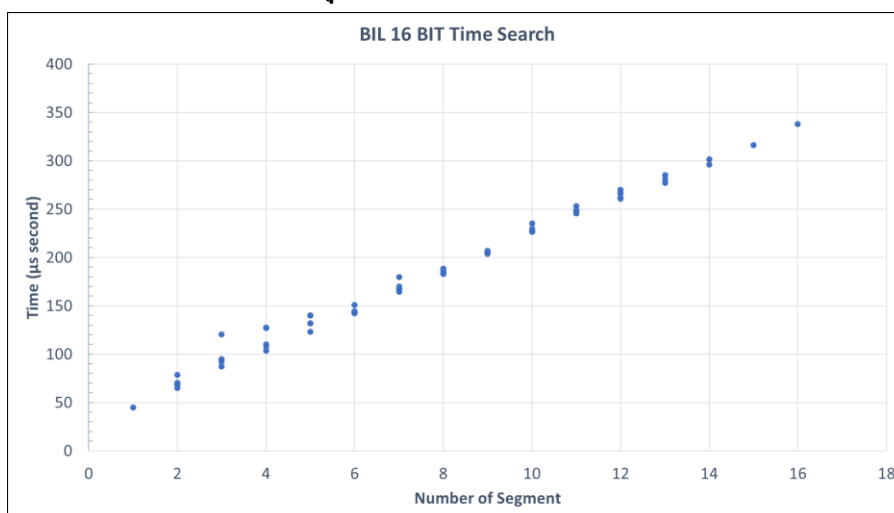
4.6.1 เปรียบเทียบความเร็วการสร้างตารางทุก Segment



รูปที่ 4.9 เวลาในการสร้างตาราง 16 BIT ทุก Segment

จากการทดลองที่ 4.6.1 เห็นได้ว่า 1 Segment มีรูปแบบเดียวคือ 16 Bit มีเวลาในการสร้างตารางนานที่สุดเนื่องจากใช้ขนาดหน่วยความจำเท่ากับ 2 กำลัง N Bit หากมี Rule มากจะยิ่งใช้เวลาสร้างตารางมากตาม หากเราแบ่ง Segment มากขึ้นจะเห็นได้ว่าเวลาที่ใช้ในการสร้างตารางจะลดลงเนื่องจากมีจำนวน Bit น้อยลง เช่น แบ่ง 2 Segment แบบ [8:8] จะมีขนาดเท่ากับ $(2^8) 2$ ตารางจะเท่ากับ 512 Index

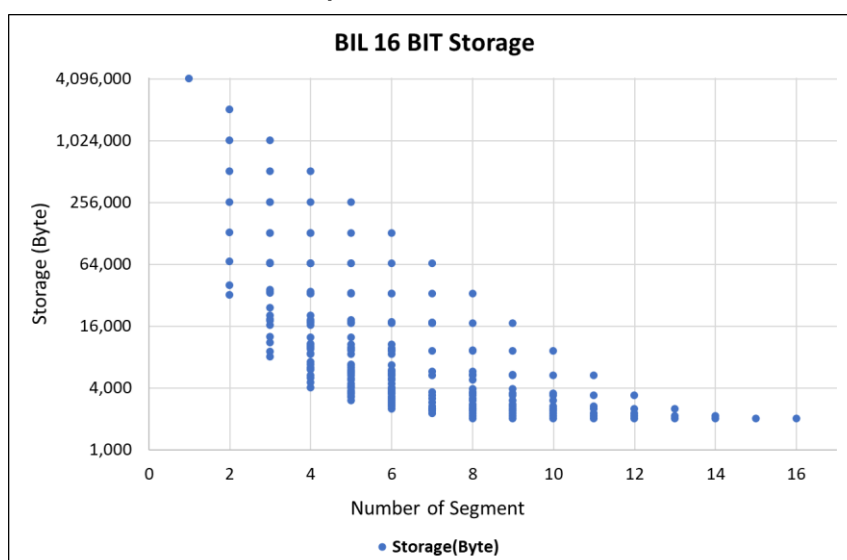
4.6.2 เปรียบเทียบความเร็วการค้นหาทุก Segment



รูปที่ 4.10 เวลาในการค้นหา 16 BIT ทุก Segment

จากการทดลอง 4.6.2 เวลาค้นหา 1 Segment จะมีเวลาค้นหาเร็วที่สุด หากแบ่ง 2 Segment ขึ้นไปจะค้นหานานขึ้นซึ่งขึ้นอยู่กัจำนวน Segment หากแบ่งมาก ก็จะค้นหาช้าตาม ซึ่งเป็นไปตามการทดลองที่ 4.4

4.6.3 เปรียบเทียบการใช้หน่วยความจำทุก Segment



รูปที่ 4.11 ความจำที่ใช้ 16 BIT ทุก Segment

จากรูปที่ 4.12 การใช้หน่วยความจำของ 1 Segment จะมีขนาดจำนวน Byte ที่มากที่สุดคือ 4,096,000 Byte หากแบ่ง Segment ให้จำนวน Bit เล็กกลงไปจนถึง 16 Segment ขนาดความจำที่ใช้จะลดลงตามลำดับ ความจำที่ใช้ต่ำที่สุดในการแบ่ง Segment ตั้งแต่ 8 Bit ถึง 16 Bit จะใช้ความจำเพียง 2,000 Byte เท่านั้น เราจะได้เห็นได้การแบ่ง Segment นั้นมีผลต่อหน่วยความจำและประสิทธิภาพของ BIL ซึ่งเป็นไปตามการทดลองที่ 4.5

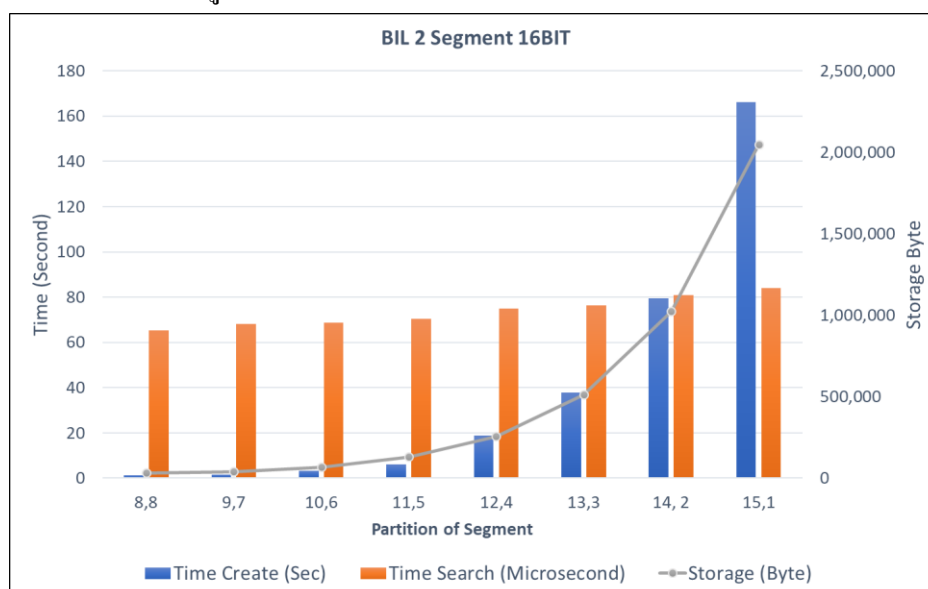
4.7 การทดลองตาม Requirement

เป็นการทดลองตาม Algorithm ที่เสนอนั้นจะเป็นการบ่งบอกว่าแบ่ง Segment เป็นกี่ Bit แล้วแบ่งจำนวน Segment เท่าใด ซึ่ง Algorithm ที่เสนอนั้นจะเป็นการบ่งบอกว่าแบ่ง Segment เท่าๆ กันจะดีที่สุด จะทำให้ใช้หน่วยความจำน้อยที่สุดและมีประสิทธิภาพในการทำงานทำดีที่สุดทั้งความเร็วการค้นหา ความเร็วการสร้างตาราง ซึ่งเราจะพิสูจน์ตาม Requirement และนำไปเปรียบเทียบกับรูปแบบอื่นที่มีขนาดหน่วยความจำเท่ากันว่าเป็นไปตาม Algorithm ที่เสนอหรือไม่ โดยตัวอย่างโจทย์ที่ใช้ในการทดลอง ดังนี้

1. อุปกรณ์ Arduino มีความจำ 32,000 Byte, 1 Field 16 Bit จำนวน 500 Rule จะแบ่ง Segment อย่างไรจึงมีประสิทธิภาพมากที่สุด
2. มีขนาดหน่วยความจำ 2,000 Byte ต้องแบ่ง Segment เท่าใดจึงจะมีประสิทธิภาพมากที่สุด

4.7.1 ผลลัพธ์การทดลองจากโจทย์ข้อที่ 1

ผลลัพธ์ที่ได้จาก Requirement คือ แบ่งแบบ 2 Segment ขนาดของ Bit เท่ากันจะได้ แบบ [8:8] ใช้หน่วยความจำ 32,000 Byte แล้วนำไปเปรียบเทียบกับการแบ่ง 2 Segment รูปแบบอื่นที่แบ่งไม่เท่ากันจะได้ผลลัพธ์ดังรูปที่ 4.12

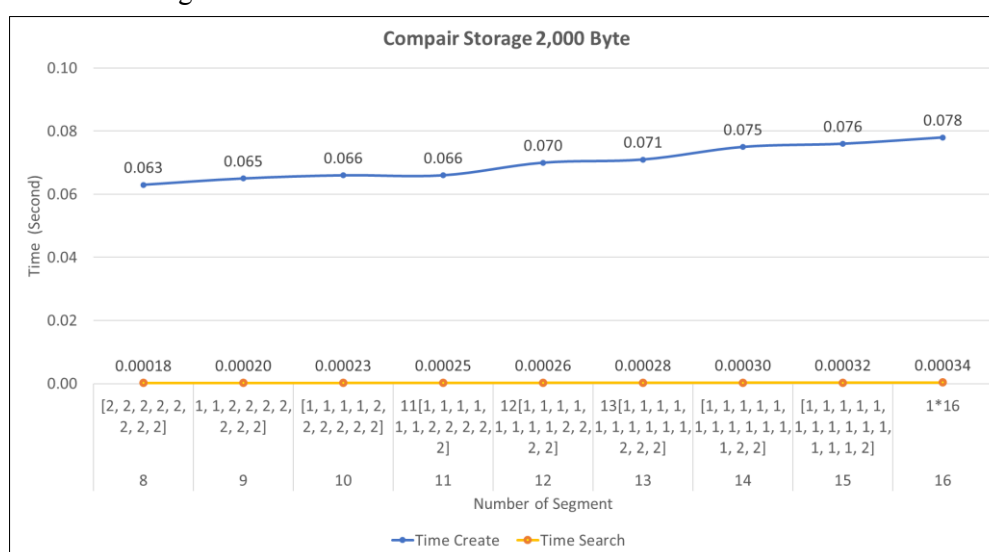


รูปที่ 4.12 ผลลัพธ์การแบ่ง Segment ตาม Requirement

จากรูปที่ 4.12 ผลลัพธ์ที่ได้จากอัลกอริทึมที่เสนอนั้น จากโจทย์ Requirement ที่กำหนด ทำให้ได้คำตอบคือ แบ่ง 2 Segment โดยแบ่งแบบ (8:8) มีเวลาดค้นหา 65 microsecond หรือเท่ากับ 0.000065 วินาที เวลาในการสร้างตาราง 1.24 วินาที และมีความจำ 32,000 Byte ซึ่งใช้เท่ากับ หน่วยความจำตามที่กำหนด จากกราฟเมื่อเปรียบเทียบกับรูปแบบอื่น ๆ ทั้งหมดของ 2 Segment ทำให้เราทราบว่าเวลาในการทำงานดีที่สุดและมีหน่วยความจำน้อยที่สุดซึ่งผลลัพธ์ที่ได้เป็นไปตาม อัลกอริทึมที่ได้ออก

4.7.2 เปรียบเทียบหน่วยความจำเท่ากัน

จากการทดลองทั้งหมดของการแบ่ง Segment ที่มีหน่วยความจำไม่เกิน 2,000 byte ทั้งหมด ดังนี้ 8 Segment, 9 Segment, 10 Segment, 11 Segment, 12 Segment, 13 Segment, 14 Segment, 15 Segment และ 16 Segment โดยเปรียบเทียบความเร็วในการทำงาน



รูปที่ 4.13 เปรียบเทียบ Segment ที่มีความจำเท่ากัน

จากการทดลองที่ 4.7.2 จากกราฟเห็นได้ว่าเวลาในการทำงานที่เร็วที่สุดเมื่อมีหน่วยความจำเท่ากัน คือแบ่ง 8 Segment แบบ [2:2:2:2:2:2:2:2] มีเวลาในการค้นหา 0.00018 วินาที และเวลาสร้างตาราง 0.063 หากแบ่ง 9 Segment ไปจนถึง 16 Segment ที่มีหน่วยความจำเท่ากันความเร็วในการทำงานจะใช้เวลาเพิ่มขึ้นตามลำดับ แต่จะห่างกันไม่มากดังรูปที่ 4.13

การผลการทดลองทั้งหมดจะเห็นว่าการแบ่ง Segment มีผลต่อหน่วยความจำและประสิทธิภาพการทำงานของ BIL ในการทดลองตาม Requirement จะได้ผลลัพธ์ที่ดีที่สุดคือแบ่ง Segment เท่าๆ กันและจำนวน Bit เท่ากันจะดีที่สุด

4.8 ผลสรุปการนำอัลกอริทึมมาประยุกต์ใช้

จากอัลกอริทึมที่นำมาใช้ในการทดลอง ทำให้ทราบว่าอัลกอริทึมนี้มีประสิทธิภาพจะขึ้นอยู่กับ Requirement แต่จะสอดคล้องกับผลการทดลอง หากจำนวน Bit ในการแบ่ง Segment เท่ากันจะดีที่สุด รวมไปถึงเวลาการทำงาน และหน่วยความจำที่ใช้จะน้อยที่สุด และจะใกล้เคียงกัน รองลงมา แต่จำนวน Segment ของอัลกอริทึมจะขึ้นอยู่กับ Requirement

บทที่ 5

วิเคราะห์และสรุปผล

5.1 สรุปผลการทดลอง

จากผลการทดลองทั้งหมดสรุปได้ว่าวิธีที่เสนอในการแบ่ง Segment ที่บอกว่าหากแบ่ง Segment เท่า ๆ กันและจำนวน Bit เท่าๆ กันจะทำให้ใช้หน่วยความจำน้อยที่สุดและเวลาการทำงาน ทั้งเวลาการสร้างตาราง เวลาการค้นหาที่สุด จะเห็นได้จากการทดลองที่กำหนด Requirement ซึ่งผลลัพธ์ที่ได้จาก Algorithm ที่เสนอทำให้ได้การแบ่ง Segment ที่มีประสิทธิภาพมากที่สุดแต่จะไม่เกินความจำที่กำหนดตาม Requirement ซึ่งเป็นไปตามอัลกอริทึมที่เสนอ โดย BIL Algorithm สามารถนำไปใช้กับ Firewall เพื่อหาตัวที่ Match กันได้ หรือนำไปใช้ในการจัดการทรัพยากรในระบบต่าง ๆ ที่มีจำกัดและยังนำไปใช้ในจัดกลุ่มข้อมูลเพื่อลดเวลาในการทำงานและเพิ่มประสิทธิภาพและรวดเร็วได้ ถ้าจะนำไปใช้งานจริงควรคำนึงถึงขนาดความจำของเครื่องเพราะประสิทธิภาพของ BIL จะขึ้นอยู่กับขนาดความจำ ถ้ามีความจำมากประสิทธิภาพจะมากตาม

5.2 ข้อเสนอแนะและแนวทางในการพัฒนาต่อ

5.2.1 พัฒนา BIL Algorithm ให้สามารถทำงานได้เร็วยิ่งขึ้นทั้งความเร็วสร้างตาราง ความเร็วค้นหา และสามารถรองรับจำนวนข้อมูลที่เข้ามา และ จำนวน Rule มาก ๆ ได้

5.2.2 พัฒนา BIL Algorithm ให้สามารถมีคู่ค้นหาของ Value/Mask ในแต่ละ Rules มีได้มากกว่าหนึ่งคู่ค้นหาและศึกษาถึงผลกระทบที่อาจเกิดขึ้น

บรรณานุกรม

- [1] Akharin Khunkitti, Nuttachot Promrit.. “**Bitmap Intersection Lookup (BIL) : A Packet Classification ’s Algorithm with Rules Updating.**” [online]. available: <https://www.koreascience.or.kr/article/CFKO200533239340120.pdf>. 2005
- [2] วรเชษฐ์นันทะเจริญ, สุภัควิ สุโพธิ์ และ อัครินทร์ คุณกิตติ. “**A Study and Development of Bitmap Intersection Lookup Algorithm for Packet Classification.**” The 18th National Conference on Computing and Information Technology. ปีที่ 18. ฉบับที่ 1. 19-20 พฤษภาคม 2022 หน้าที่ 448 - 453
- [3] Javatpoint. “**Linear Search Algorithm.**” [online]. available: <https://www.javatpoint.com/linear-search>. 2011
- [4] Tutorialspoint. “**C-Programming Tutorial.**” [online]. available: https://www.tutorialspoint.com/cprogramming/c_strings.htm.
- [5] Benznest. “**เขียนโปรแกรมภาษา C แบบพื้นฐาน.**” [ออนไลน์]. เข้าถึงได้จาก: <https://benzneststudios.com/blog/c-programming/c-programming-basic-3/>. 2559
- [6] Python Tutorial. “**Basic Python**” [online]. Available: <https://www.pythontutorial.net/python-basics/python-list-slice/>. 2023
- [7] GeeksforGeeks. “**Python Lists**”. [online]. Available: <https://www.geeksforgeeks.org/python-lists/>. 2023
- [8] Bartosz Zaczynski. “**Bitwise Operator in Python**”. [online]. Available: <https://realpython.com/python-bitwise-operators/>. 2021
- [9] John Strutz. “**Dictionaries in Python**”. [online]. Available: <https://realpython.com/python-dicts/>. 2019

ภาคผนวก

ภาคผนวก ก
Source Code ในการทดลอง

โปรแกรมในการสร้าง BIL Table

```

import random, math
import pandas as pd
import operator as opt
import numpy as np
import time

# rule table : mask[BitofIndex], value[BitofIndex]
#Parameter
BITOFINDEX:int = 16 # Bit of field = 2**n
BITOFRULE:int = 500 # Number rule
INPUT_SIZE: int = 2**BITOFINDEX

def convert2(input: int,count: int) -> list:
    num:list = [0]*count
    for i in range(len(num)):
        bit:int = input % 2
        input:int = input // 2
        num[count-1-i] = bit
    return num

def GenRule(BitOfindex: int) -> dict:
    value = [0]*BitOfindex
    mask = [0]*BitOfindex
    for i in range(BitOfindex):
        value[i] = random.randint(0,1)
        mask[i] = random.randint(0,1)
    return {"value":value, "mask":mask}

```

```

# map_rule เราทำเพื่อเช็ค input ที่รับเข้ามาตรงกันกับเงื่อนไขหรือไม่
def map_rule(index: int, value: list, mask: list, BitOfindex: int) -> int:
    indexbase2:list = convert2(index, BitOfindex);
    ans = all((np.array(mask) & np.array(value)) & np.array(indexbase2))
    # all มีบิตไหนเป็น 0 จะรีเทิร์น false ออกมาเพราะมันไม่เป็นไปตาม Value
    return ans

# ans จะ turn ค่า 0 ที่อยู่ return ออกมาข้างใน list คือ ค่าที่ไม่ตรงกับกฎที่กำหนดไว้
# แต่ถ้า ค่าเป็น 1 ที่อยู่ return ออกมาข้างใน list คือ ค่าที่ตรงกับกฎ
ที่กำหนดไว้

def partition_input(input:list, bit_of_index:int, number_of_segment:int, segment_partition_size:list):
    seg_input = [0] * number_of_segment
    # segment_size:int = int(bit_of_index / number_of_segment)
    # print('segment_size : ', segment_size)

def partition_input(input:list, bit_of_index:int, number_of_segment:int, segment_partition_size:list):
    seg_input = [0] * number_of_segment
    # segment_size:int = int(bit_of_index / number_of_segment)
    # print('segment_size : ', segment_size)
    temp_segment_size = 0
    for ns in range(number_of_segment):
        binary_value = input[temp_segment_size : temp_segment_size + segment_partition_size[ns]]
        temp_segment_size = temp_segment_size + segment_partition_size[ns]
        # print('binary_value : ', binary_value)
        for i, value in enumerate(binary_value):
            seg_input[ns] += int(value * math.pow(2, (segment_partition_size[ns]-i-1)))
    return seg_input

```

```

def create_rule_table(bit_of_index:int, bit_of_rule:int):
    rt = []
    for i in range(bit_of_rule):
        """i is rule index"""
        # print("Rule NO: {}".format(i));
        rule = GenRule(bit_of_index)
        rt.append(rule)
    return rt

rt = create_rule_table(bit_of_index=16, bit_of_rule=1000)

def create_bil_table_function(field_size:int, rule_table: list, bit_of_index:int, bit_of_rule:int):

    bil_table = dict()

    for input in range(field_size):
        index_field: list = [1]*bit_of_rule
        for rule_no in range(bit_of_rule):
            check_maprule:int = map_rule(input, rule_table[rule_no]["value"], rule_table[rule_no]["mask"],
bit_of_index)
            index_field[rule_no] = check_maprule
        bil_table[input] = index_field
        # print(len(bil_table), bil_table)
    return bil_table

def search_bil_function(input: int, bil_table: dict) -> list:
    rule_id = []
    for i, bit_value in enumerate(bil_table[input]):
        if bit_value == 1:
            rule_id.append(i)
    if len(rule_id) > 0:
        return rule_id[0]
    else:
        return '-'

```

```

def bil_search_algorithm(input_size:int, field_size: int, rule_table:list,bit_of_rule:int):

    bil_search_result = []

    start = time.time()

    bil_table = create_bil_table_function(field_size=2**BITOFINDEX,

                                         rule_table=rt,

                                         bit_of_index=BITOFINDEX,

                                         bit_of_rule=10000)

    end = time.time()

    print('BIL Table create time : ', end - start , ' seconds')


    print('BIL_Table size : ', len((bil_table)*bit_of_rule/8),' bytes')


    """ loop test input subject in range (0 - 65536) """
    for input in range(input_size):

        """
        "ans" is เป็นตัวแปรที่เก็บว่า input นี้มี rule ของใดข้อนี้ invalid ยัง ถ้าไม่มี จะเปลี่ยนเป็น 0
        "result_checkmaprule" is ตัวแปรที่เก็บผลลัพธ์ ว่า
        input นี้เมื่อเช็คกับ rule แต่ละข้อ ว่า ข้อไหน ให้ผ่าน ไม่ให้ผ่านบ้าง มีขนาดเท่ากับ จำนวนกฎ
        """

        ans = search_bil_function(input, bil_table)

        bil_search_result.append(ans)

    return bil_search_result

```

โปรแกรมการแบ่ง Segment BIL Algorithm

```
#function BIL Create Table for none equal Segment

def bil_table_create_for_none_equal_segmentation(rt: list, bit_of_rule: int, bit_of_index:int,
number_of_segment:int, segment_partition_size=list):
    if sum(segment_partition_size) != bit_of_index:
        raise Exception("sum(segment_partition_size) != bit_of_index")
    if len(segment_partition_size) != number_of_segment:
        raise Exception("len(segment_partition_size) != number_of_segment")
    seg_rt = [list()] * number_of_segment

    for rule_no in range(bit_of_rule):
        temp_rule = rt[rule_no]
        temp_segment_size = 0

        for ns in range(number_of_segment):
            value = temp_rule['value'][temp_segment_size: temp_segment_size + segment_partition_size[ns]]
            mask = temp_rule['mask'][temp_segment_size: temp_segment_size + segment_partition_size[ns]]
            temp_segment_size = temp_segment_size + segment_partition_size[ns]
            # print([{'value': value, 'mask':mask}])
            seg_rt[ns] = seg_rt[ns] + [{'value': value,
                                     'mask':mask}]

    return seg_rt

import numpy as np

#Bil Search on Segmentation
def bil_search_on_segmentation_algorithm_for_none_equal_segmentation(input_size:int, bit_of_index:
int,bit_of_rule:int, rule_table:list, number_of_segment=2, segment_partition_size=list):
    bil_search_result = []
    start = time.time()
```

```

seg_table = bil_table_create_for_none_equal_segmentation(rt=rule_table,
                                                         bit_of_index=bit_of_index,
                                                         bit_of_rule=bit_of_rule,
                                                         number_of_segment=number_of_segment,
                                                         segment_partition_size=segment_partition_size)

seg_bil_table = [list()] * number_of_segment
for i in range(number_of_segment):
    seg_bil_table[i] = create_bil_table_function(field_size=2**segment_partition_size[i],
                                                rule_table=seg_table[i],
                                                bit_of_index=segment_partition_size[i],
                                                bit_of_rule=bit_of_rule)

end = time.time()

print('BIL Table create time : ', end - start , ' seconds')
print('BIL_Table size : ', len([j for i in seg_bil_table for j in i]), ' bytes')

match_rule_ns = seg_bil_table[ns][binary_partition_input[ns]]
for i in range(bit_of_rule):
    after_AND_operation_result[i] = after_AND_operation_result[i] and match_rule_ns[i]

rule_id = []
for i, bit_value in enumerate(after_AND_operation_result):
    if bit_value == 1:
        rule_id.append(i)
    if len(rule_id) > 0:
        bil_search_result.append(rule_id[0])
    else:
        bil_search_result.append('-')
end = time.time()
print('BIL Search time : ', end - start , ' seconds')
return bil_search_result

```



```
result =
bil_search_on_segmentation_algorithm_for_none_equal_segmentation(input_size=2**BITOFINDEX,
    bit_of_index=BITOFINDEX,
    bit_of_rule=1000,
    rule_table=rt,
    number_of_segment=2,
    segment_partition_size=[6, 10])
```

ประวัติผู้เขียน

ชื่อ – นามสกุล..... นายสมภพ ศักดิ์ศรีชัย

รหัสนักศึกษา..... 62070191

วัน เดือน ปีเกิด..... 28 มิถุนายน 2543



ประวัติการศึกษา

วุฒิ ม.6 ชื่อที่อยู่สถาบัน..... โรงเรียนเตรียมอุดมศึกษาพัฒนาการ รัชดา

ภูมิลำเนา..... 12 ซอยสิทธิปัญญา ถนนสุทธิสาร เขตห้วยขวาง แขวงสามเสนนอก กรุงเทพฯ 10310

เบอร์โทร..... 089-8875-018 E-Mail..... 62070191@it.kmitl.ac.th

สาขาที่จบ..... เทคโนโลยีสารสนเทศ รุ่นที่..... 17 ปีการศึกษาที่จบ..... 2566

ชื่อ – นามสกุล..... นายอภิสร ดีอ้อ

รหัสนักศึกษา..... 62070218

วัน เดือน ปีเกิด..... 8 กุมภาพันธ์ 2544



ประวัติการศึกษา

วุฒิ ม.6 ชื่อที่อยู่สถาบัน..... โรงเรียนสภาราชินี

ภูมิลำเนา..... 38/3 ตำบลกะลาเส อำเภอลิเกา จังหวัดตรัง 92150

เบอร์โทร..... 098-478-1978 E-Mail..... 62070218@it.kmitl.ac.th

สาขาที่จบ..... เทคโนโลยีสารสนเทศ รุ่นที่..... 17 ปีการศึกษาที่จบ..... 2566