# การขุดค้นรูปแบบสตริงย่อยแบบความยาวสูงสุดที่เกิดบ่อย

# FREQUENT MAX SUBSTRING PATTERNS MINING

ทศนัย ชุ่มวัฒนะ
TODSANAI CHUMWATANA

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาการคอมพิวเตอร์
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2549

ISBN 974-15-2034-9

# การสืบค้นรูปแบบสตริงย่อยแบบความยาวสูงสุดที่เกิดบ่อย

# FREQUENT MAX SUBSTRING PATTERNS MINING

ทศนัย ชุ่มวัฒนะ

TODSANAI CHUMWATANA

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต

สาขาวิชาวิทยาการคอมพิวเตอร์

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.2549

ISBN 974-15-2684-9

# FREQUENT MAX SUBSTRING PATTERNS MINING

TODSANAI CHUMWATANA

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF SCIENCE IN COMPUTER SCIENCE

SCHOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2006

ISBN 974-15-2684-9

| | |
|---|---|
| **หัวข้อวิทยานิพนธ์** | การสืบค้นรูปแบบสตริงย่อยแบบความยาวสูงสุดที่เกิดบ่อย |
| **นักศึกษา** | นายทศนัย ชุ่มวัฒนะ |
| **รหัสประจำตัว** | 47063704 |
| **ปริญญา** | วิทยาศาสตรมหาบัณฑิต |
| **สาขาวิชา** | วิทยาการคอมพิวเตอร์ |
| **พ.ศ.** | 2549 |
| **อาจารย์ผู้ควบคุมวิทยานิพนธ์** | รศ.ดร. วีระ บุญจริง |

## บทคัดย่อ

งานวิจัยนี้นำเสนออัลกอริทึมใหม่สำหรับการสืบค้นเซตของรูปแบบสตริงย่อยแบบความยาวสูงสุดที่เกิดบ่อยที่เรียกว่า FM จากสตริงนำเข้าใด ๆ โดยที่ค่าความถี่ขั้นต่ำที่กำหนดให้ FM คือ เซตของรูปแบบสตริงย่อยที่มีขนาดความยาวสูงสุดที่มีค่าความถี่ผ่านค่าความถี่ขั้นต่ำที่กำหนดให้นี้ งานวิจัยนี้ใช้โครงสร้างข้อมูลใหม่ที่เรียกว่าซัฟฟิกซ์ทรัยแบบมีความถี่ หรือ FST ซึ่งรับประกันความครบถ้วนและถูกต้องในการแจงนับรูปแบบสตริงย่อย อัลกอริทึมใหม่ทำการสืบค้น FM แบบมีประสิทธิภาพโดยใช้คุณสมบัติบางประการของโครงสร้าง FST

| | |
|---|---|
| **Thesis** | Frequent Max Substring Patterns Mining |
| **Student** | Mr. Todsanai Chumwatana |
| **Student ID** | 47063704 |
| **Degree** | Master of Science |
| **Programme** | Computer Science |
| **Year** | 2006 |
| **Thesis Advisor** | Assoc.Prof.Dr.Veera Boonjing |

## ABSTRACT

This research proposes a novel algorithm for mining a set of frequent max substring patterns called FM from an input string. At a given threshold, FM is a set of maximum-length substring patterns with their frequencies at least at the threshold value. The research uses the new data structure called the frequent suffix trie or FST to assure exhaustive enumeration of substring patterns. The new algorithm efficiently searches for FM using some FST properties.

# กิตติกรรมประกาศ

วิทยานิพนธ์เล่มนี้ได้สำเร็จลุล่วงไปได้ด้วยดี ด้วยความกรุณาจากอาจารย์ที่ปรึกษา รศ.ดร. วีระ บุญจริง ที่ให้คำชี้แนะ ความช่วยเหลือ ความรู้ ความเอาใจใส่ และประสบการณ์ที่ดีแก่ข้าพเจ้า ซึ่งท่านได้สละเวลาอันมีค่าให้กับข้าพเจ้าอย่างเต็มที่ จึงใคร่ขอขอบพระคุณเป็นอย่างสูง

ขอกราบขอบพระคุณคณาจารย์ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ทุก ๆ ท่านที่ได้ประสิทธิ์ประสาทวิชาความรู้ให้แก่ข้าพเจ้า

ขอขอบคุณเพื่อน ๆ ประจำภาควิชาวิทยาการคอมพิวเตอร์ทุกท่านที่คอยเป็นกำลังใจให้ข้าพเจ้าสามารถทำวิทยานิพนธ์เล่มนี้ได้สำเร็จ

ขอขอบคุณ คุณ สรินา พรหมสวัสดิ์ ที่คอยเป็นกำลังใจ และเป็นแรงพลักดันให้ข้าพเจ้ามีแรงในการทำวิทยานิพนธ์นี้ให้สำเร็จ

สุดท้ายนี้ ข้าพเจ้าขอกราบขอบพระคุณ บิดา มารดา และครอบครัวของข้าพเจ้าที่ให้การสนับสนุนในทุก ๆ เรื่อง และเป็นกำลังใจที่ยิ่งใหญ่ที่ทำให้ข้าพเจ้าสามารถทำวิทยานิพนธ์เล่มนี้สำเร็จลุล่วงด้วยดี

สำหรับคุณค่าและประโยชน์อันใดที่เกิดขึ้นจากวิทยานิพนธ์ฉบับนี้ ข้าพเจ้าขอมอบให้กับบิดา มารดา อาจารย์ทุกท่านซึ่งเป็นที่เคารพรักยิ่ง ตลอดจนญาติพี่น้อง และเพื่อน ๆ ทุกคน

ทศนัย ชุ่มวัฒนะ

กรกฎาคม 2549

# สารบัญ

หน้า

# สารบัญ (ต่อ)

หน้า

# สารบัญตาราง

# สารบัญรูป

รูปที่ หน้า

# บทที่ 1
# บทนำ

## 1.1 ความเป็นมาและความสำคัญของปัญหา

การสืบค้นรูปแบบสตริงย่อยแบบความยาวสูงสุดที่เกิดบ่อย(Frequent Max Substring Pattern Mining) หรือ FM บนสตริงใด ๆ คือ การสืบค้นหารูปแบบสตริงย่อยแบบความยาวสูงสุดต่าง ๆ ซึ่งมีค่าความถี่ผ่านความถี่ขั้นต่ำ โดยที่รูปแบบเหล่านี้ต้องไม่มีรูปแบบใดที่มีค่าความถี่ผ่านค่าความถี่ขั้นต่ำครอบคลุมอยู่ แนวคิดพื้นฐานในการสืบค้นหารูปแบบเหล่านี้บนสตริง คือ การแจงนับรูปแบบสตริงย่อยทั้งหมดพร้อมความถี่ แล้วทำการคัดเลือก FM ตามค่าความถี่ขั้นต่ำที่กำหนด โดยที่การแจงนับนั้นต้องให้รูปแบบสตริงย่อยทั้งหมดพร้อมความถี่ที่ครบถ้วนและถูกต้อง ซึ่งวิธีการแจงนับหนึ่งที่ให้ผลลัพธ์ของรูปแบบสตริงย่อยครบถ้วนได้แก่ การแจงนับโดยการใช้โครงสร้างซัฟฟิกซ์ทรัย[8] แต่โครงสร้างซัฟฟิกซ์ทรัยที่มีใช้โดยทั่วไป เป็นโครงสร้างแบบที่ไม่มีความถี่ กล่าวคือ โครงสร้างซัฟฟิกซ์ทรัยเป็นโครงสร้างที่แจงเฉพาะรูปแบบสตริงย่อยต่าง ๆ แต่ไม่มีจำนวนครั้งของการเกิดขึ้นของรูปแบบสตริงย่อยเหล่านั้น ซึ่งโครงสร้างซัฟฟิกซ์ทรัยนั้น ถือว่าเป็นโครงสร้างชนิดหนึ่งในโครงสร้างทรัย โดยที่โครงสร้างทรัยคือโครงสร้างต้นไม้ที่ใช้สำหรับเก็บสตริงซึ่งแบ่งออกเป็น 5 ชนิด[6] คือ 1. นอนคอมแพคทรัย(Noncompact Trie) 2. คอมแพคทรัย(Nompact Trie) 3. แพทริเซีย(PATRICIA) 4. ซัฟฟิกซ์ทรัย(Suffix Trie) 5. ซัฟฟิกซ์ทรี (Suffix Tree) ซึ่งชนิดของโครงสร้างทั้งหมดที่กล่าวมานั้น ล้วนแต่เป็นโครงสร้างที่ถูกพัฒนาขึ้นมาเพื่อใช้ในการเก็บสตริงเพื่อความสะดวกในการสืบค้นหารูปแบบสตริงย่อย แต่ไม่สนับสนุนทางด้านการเก็บค่าความถี่ของรูปแบบสตริงย่อย ดังนั้น หากเราสามารถมีโครงสร้างซัฟฟิกซ์ทรัยที่สามารถเก็บค่าความถี่ของรูปแบบสตริงย่อยได้ด้วย จะเป็นประโยชน์อย่างยิ่งต่อการสืบค้น FM โดยที่โครงสร้างนี้จะเป็นโครงสร้างที่เป็นที่เก็บของรูปแบบสตริงย่อยทั้งหมดของสตริงใด ๆ พร้อมความถี่ ซึ่งงานวิจัยที่ผ่านมาได้ทำการศึกษาค้นคว้าทางด้านการสืบค้นสตริงย่อยที่เกิดบ่อยจากสตริง[3] โดยใช้ซัฟฟิกซ์ทรัยแบบระบุตำแหน่ง แต่ไม่ได้ทำการสืบค้น FM ดังนั้นจึงเกิดแนวความคิดในการพัฒนาโครงสร้างข้อมูลใหม่โดยทำการประยุกต์จากซัฟฟิกซ์ทรัยแบบระบุตำแหน่งเป็นซัฟฟิกซ์ทรัยแบบมีความถี่(Frequent suffix trie) หรือ FST เพื่อให้เหมาะสมต่อการสืบค้น FM

## 1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

งานวิจัยนี้มีวัตถุประสงค์เพื่อพัฒนาอัลกอริทึมในการสืบค้น FM โดยใช้โครงสร้างซัฟฟิกซ์ทรัยแบบใหม่ที่เรียกว่า โครงสร้างซัฟฟิกซ์ทรัยแบบมีความถี่(Frequent Suffix Trie) หรือ FST โดยที่โครงสร้างนี้สืบทอดคุณสมบัติของโครงสร้างซัฟฟิกซ์ทรัยทั่วไป คือ รูปแบบสตริงย่อยขนาดเล็กจะมีค่าความถี่มากกว่าหรือเท่ากับรูปแบบสตริงย่อยขนาดใหญ่ในเส้นทางเดียวกันบนโครงสร้างซัฟฟิกซ์ทรัย หรือรูปสตริงย่อยยิ่งมีขนาดใหญ่จะมีค่าความถี่น้อยลง และรูปแบบสตริงย่อยพ่อแม่จะมีความถี่มากกว่าหรือเท่ากับรูปแบบสตริงย่อยลูกเสมอ เนื่องจากรูปแบบสตริงย่อยขนาดเล็กจะถูกกระจายไปสู่การเป็นสมาชิกของรูปแบบสตริงย่อยขนาดใหญ่ ซึ่งคุณสมบัติเหล่านี้จะเป็นประโยชน์ต่อการลดจำนวนการแจงนับคำตอบที่เป็นไปได้ เพื่อการสืบค้น FM ที่ถูกต้องและครบถ้วน

## 1.3 ขอบเขตการวิจัย

งานวิจัยนี้เป็นการพัฒนาโครงสร้าง FST พร้อมทั้งศึกษาคุณสมบัติของโครงสร้าง FST และพัฒนาอัลกอริทึมในการสืบค้น FM โดยใช้คุณสมบัติของโครงสร้าง FST ในการแจงนับคำตอบ โดยจะทำการวัดประสิทธิภาพของอัลกอริทึมโดยใช้จำนวนรูปแบบสตริงย่อยที่อัลกอริทึมสร้างขึ้นต่อจำนวนรูปแบบสตริงย่อยที่เป็นไปได้ทั้งหมด

## 1.4 ส่วนประกอบของวิทยานิพนธ์

ส่วนที่เหลือของวิทยานิพนธ์ฉบับนี้ประกอบด้วยบทต่าง ๆ ดังนี้

บทที่ 2 กล่าวถึงนิยามพื้นฐานของการสืบค้นสตริงย่อยบนสตริงพร้อมทั้งนำเสนองานวิจัยที่เกี่ยวข้อง ได้แก่ การสืบค้นสตริงย่อยบนสตริงโดยใช้โครงสร้างทรัย, การสืบค้นสตริงย่อยบนสตริงโดยใช้ซัฟฟิกซ์อาเรย์ และการสืบค้นสตริงย่อยที่เกิดบ่อยบนสตริงโดยใช้ซัฟฟิกซ์ทรัยแบบระบุตำแหน่ง

บทที่ 3 กล่าวถึงนิยามพื้นฐานที่ใช้ในการสืบค้น FM และอัลกอริทึมในการสืบค้น FM

บทที่ 4 กล่าวถึงการทดลองและผลการทดลองเพื่อเปรียบเทียบอัตราการลดจำนวนการสร้างรูปแบบสตริงย่อยระหว่างอัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว และอัลกอริทึมที่มีประสิทธิภาพ โดยใช้อัลกอริทึมพื้นฐานเป็นฐานในการทดสอบเพื่อตรวจสอบความถูกต้องของผลลัพธ์ FM

บทที่ 5 สรุปวิธีการสืบค้น FM และข้อเสนอแนะเกี่ยวกับการสืบค้น FM

# บทที่ 2
# ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

เนื่องจากวิทยานิพนธ์นี้ได้นำเสนออัลกอริทึมใหม่ในการสืบค้นรูปแบบสตริงย่อยแบบความยาวสูงสุดที่เกิดบ่อย หรือ FM โดยการใช้โครงสร้างข้อมูลใหม่ที่เรียกว่า ซัฟฟิกซ์ทรัยแบบมีความถี่ หรือ FST ซึ่งงานวิจัยที่ผ่านมา พบว่ายังไม่มีการศึกษาค้นคว้าทางด้านการสืบค้น FM บนโครงสร้างข้อมูลต่าง ๆ โดยงานวิจัยที่ผ่านมานั้น จะเป็นการศึกษาค้นคว้าทางด้านการสืบค้นสตริงย่อยบนสตริงโดยใช้โครงสร้างทรัย และการสืบค้นสตริงย่อยที่เกิดบ่อยบนสตริงโดยใช้ซัฟฟิกซ์ทรัยแบบระบุตำแหน่ง ดังนั้น วิทยานิพนธ์นี้จึงขอนำเสนองานวิจัยที่เกี่ยวข้องทางด้านการสืบค้นสตริงย่อยบนสตริงโดยใช้โครงสร้างทรัย และการสืบค้นสตริงย่อยที่เกิดบ่อยบนสตริงโดยใช้ซัฟฟิกซ์ทรัยแบบระบุตำแหน่ง โดยในหัวข้อที่ 2.1 นำเสนอนิยามพื้นฐานในการสืบค้นสตริงย่อยบนสตริง หัวข้อที่ 2.2 นำเสนอการสืบค้นสตริงย่อยบนสตริงโดยใช้โครงสร้างทรัย หัวข้อที่ 2.3 นำเสนอการสืบค้นสตริงย่อยบนสตริงโดยใช้ซัฟฟิกซ์อาเรย์ และ หัวข้อที่ 2.4 นำเสนอการสืบค้นสตริงย่อยที่เกิดบ่อยบนสตริงโดยใช้ซัฟฟิกซ์ทรัยแบบระบุตำแหน่ง

## 2.1 นิยามพื้นฐานการสืบค้นสตริงย่อยบนสตริง

กำหนดให้ $\sum$ คือ เซตของตัวอักขระที่มีจำนวน N สัญลักษณ์

**ตัวอย่างที่ 2.1** เซตของตัวอักขระ $\sum$ = {ก, ร, ม} ซึ่งมีจำนวน 3 สัญลักษณ์

**นิยามที่ 2.1** ลำดับเหตุการณ์ $T = (s_1 s_2 \ldots s_n)$ โดยที่ $n \geq 0$ และ $s_i$ เป็นสมาชิกใด ๆ ของ $\sum$ ดังนั้น T จะถูกเรียกว่า สตริง บนเซตของตัวอักขระ $\sum$ และความยาวของสตริง T จะถูกกำหนดด้วย |T| หรือ n คือ จำนวนของตัวอักขระในสตริง T ดังนั้น สตริง T ที่มีความยาว 0 จึงเป็นสตริงว่าง ซึ่งจะถูกแทนด้วย $\lambda$

**ตัวอย่างที่ 2.2** จากตัวอย่างที่ 2.1

ถ้า สตริง T = (กรรมกร) จะเห็นว่า แต่ละตัวอักขระบนสตริง T จะเป็นสมาชิกของเซตของตัวอักขระ $\sum$

ข้อสังเกต จากนิยามที่ 2.1 เราสามารถระบุตัวอักขระบนสตริง T โดยใช้ตำแหน่งในสตริง T เช่น ตัวอักขระ $s_i$ ที่ตำแหน่ง i สามารถกำหนดเป็น t[i] ซึ่งไม่ใช่สตริงว่าง โดยที่ $1 \leq i \leq |T|$ ดังนั้น ตำแหน่งแรกของสตริง คือ 1 และตำแหน่งของสัญลักษณ์ตัวสุดท้ายจะอยู่ที่ตำแหน่ง |T|

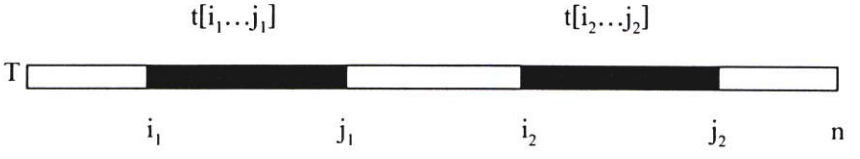
**นิยามที่ 2.2** กลุ่มสัญลักษณ์ $s_i$..... $s_j$ บนสตริง T จะถูกเรียกว่า สตริงย่อยของสตริง T คือ ลำดับเหตุการณ์ของตัวอักขระที่ติดกันที่ได้มาจากสตริง T ซึ่งเริ่มต้นที่ตำแหน่ง i และ สิ้นสุดที่ตำแหน่ง j เราจะกำหนดสตริงย่อยนี้โดย t[i...j] โดยที่ $1 \leq i \leq j \leq |T|$ ดังนั้น สตริงย่อย t[i...i] จะมีความยาว 1 เนื่องจากตำแหน่งเริ่มต้น และตำแหน่งสิ้นสุดเท่ากับ i สตริงย่อย t[i...j] จะมีความยาวเท่ากับ j – i + 1 ดังนั้น เราจึงสามารถกำหนดความสัมพันธ์นี้เป็น $t \sqsubseteq T$ พูดได้ว่าสตริง T คือ ซุปเปอร์สตริง(Superstring) ของ t หรือ t คือ สับสตริง(Substring) ของ T

**ตัวอย่าง 2.3** จากตัวอย่างที่ 2.2
เราจะได้ผลลัพธ์ของสตริงย่อย t บนสตริง T ดังนี้
สตริงย่อย t[i...j] ที่มีความยาว 1 = {[ก], [ร], [ม]}
สตริงย่อย t[i...j] ที่มีความยาว 2 = {[กร], [รร], [รม], [มก]}
สตริงย่อย t[i...j] ที่มีความยาว 3 = {[กรร], [รรม], [รมก], [มกร]}
สตริงย่อย t[i...j] ที่มีความยาว 4 = {[กรรม], [รรมก], [รมกร]}
สตริงย่อย t[i...j] ที่มีความยาว 5 = {[กรรมก], [รรมกร]}
สตริงย่อย t[i...j] ที่มีความยาว |T| = {[กรรมกร]}

ข้อสังเกต จากนิยามที่ 2.2 สตริงย่อย x สามารถเกิดขึ้นได้หลายครั้งในสตริง T ถ้า x = t[i...j] = t[$i' \ldots j'$] โดยที่ $i \neq i'$ และ $j \neq j'$

**นิยามที่ 2.3** คู่ของสตริงย่อย R = (t[$i_1 \ldots j_1$], t[$i_2 \ldots j_2$] บนสตริง จะถูกเรียกว่า สตริงย่อยที่เกิดซ้ำ คือ สตริงย่อยที่มีการเกิดขึ้นมากกว่าหนึ่งครั้งบนสตริง เขียนแทนด้วย

$$t[i_1 \ldots j_1] = t[i_2 \ldots j_2]$$



**ตัวอย่าง 2.4** จากตัวอย่างที่ 2.2
จะเห็นว่า สตริงย่อยที่เกิดซ้ำบนสตริง T คือ สตริงย่อย x = [กร]

**นิยามที่ 2.4** x = t[m...n] และ z = t[i....j] เป็นสตริงย่อยบนสตริง และ x เป็น Substring ของ z ก็ต่อเมื่อ x ถูกครอบคลุมโดย z เขียนแทนด้วย

$$\text{Substring\_of}(x; z) \equiv x \sqsubseteq z$$

ถ้า x = t[m...n] และ z = t[i....j] เป็นสตริงย่อยบนสตริง และ x เป็น Superstring ของ z ก็ต่อเมื่อ x ครอบคลุม z เขียนแทนด้วย

$$\text{Superstring\_of}(x; z) \equiv x \sqsupseteq z$$

**ตัวอย่าง 2.5** จากตัวอย่างที่ 2.3
ถ้า x = [กร] และ z = [กรร] จะเห็นว่า Substring_of(x; z) $\equiv x \sqsubseteq z$

แต่ถ้า x = [กรรม] และ z = [รรม] จะเห็นว่า Superstring_of(x; z) $\equiv x \sqsupseteq z$

จากนิยามพื้นฐานดังกล่าว ทำให้เกิดงานวิจัยที่เป็นการคิดค้นโครงสร้างข้อมูลประเภทต่างๆ เพื่อนำมาใช้ในการสืบค้นสตริงย่อยบนสตริง ดังต่อไปนี้

## 2.2 การสืบค้นสตริงย่อยบนสตริงโดยใช้โครงสร้างทรัย

โครงสร้างทรัย คือ โครงสร้างต้นไม้สำหรับเก็บสตริงซึ่งประกอบด้วยเส้นทางเชื่อมต่อระหว่างโนดสองโนด(Edge) โดยที่ทุก ๆ เอดจะนำเสนอถึงสัญลักษณ์ที่เป็นสมาชิกของเซตของตัวอักขระ ซึ่งโนดปลายสุดจะเป็นโนดพิเศษที่ใช้สำหรับเก็บซัฟฟิกซ์ในแต่ละเส้นทาง

โครงสร้างทรัยมาจากคำว่า Retrieval ซึ่งถูกนำเสนอโดยเฟรดคิน(Fredkin) ในปี ค.ศ. 1960 โดยโครงสร้างทรัยถือว่าเป็นโครงสร้างที่สามารถทำการสืบค้นได้รวดเร็วบนข้อมูลสตริงที่มีขนาดใหญ่ เช่น การสืบค้นคำศัพท์ในพจนานุกรมอังกฤษออกซ์ฟอร์ด(Oxford English dictionary) ซึ่งบรรจุข้อมูลสตริงหลายกิกะไบต์โดยไม่มีการเพิ่มหรือลบคำศัพท์ต่าง ๆ อีกทั้งโครงสร้างทรัยยังสามารถนำไปใช้ประโยชน์ทางด้านการค้นหาข้อมูลสตริงได้อย่างมีประสิทธิภาพ

โครงสร้างทรัยถูกสร้างขึ้นจากข้อมูลสตริงนำเข้า โดยจะบรรจุสัญลักษณ์ $ ต่อท้ายข้อมูลสตริงนำเข้าเพื่อแสดงถึงจุดสิ้นสุดของสตริง โครงสร้างทรัยแบ่งออกเป็น 5 ชนิด[6] คือ 1. นอนคอมแพคทรัย(Noncompact Trie) 2. คอมแพคทรัย(Nompact Trie) 3. แพทริเซีย(PATRICIA) 4. ซัฟฟิกซ์ทรัย(Suffix Trie) 5. ซัฟฟิกซ์ทรี(Suffix Tree) สำหรับการสืบค้นสตริงย่อยบนสตริงจะใช้ซัฟฟิกซ์ทรี และซัฟฟิกซ์ทรัยเป็นโครงสร้างข้อมูลพื้นฐานในการแจงนับสตริงย่อย ดังนั้นในวิทยานิพนธ์นี้จึงนำมาอธิบายเฉพาะซัฟฟิกซ์ทรี และซัฟฟิกซ์ทรัยเท่านั้น

### 2.2.1 ซัฟฟิกซ์ทรี

ซัฟฟิกซ์ทรี คือ เครื่องมือสนับสนุนที่มีประสิทธิภาพในการประมวลผลของสตริงซึ่งได้รับการพัฒนาต่อมาจากซัฟฟิกซ์ทรัย เพื่อลดความลึกของซัฟฟิกซ์ทรัย โดยที่ซัฟฟิกซ์ทรีของสตริงใด ๆ คือ โครงสร้างดัชนีที่สามารถทำการคำนวณ และเก็บค่าโดยใช้เวลา และเนื้อที่หน่วยความจำเท่ากับ $O(|T|)$ ซัฟฟิกซ์ทรีถือเป็นโครงสร้างข้อมูลหนึ่ง ซึ่งสามารถเข้าถึงทุก ๆ สตริงย่อยบนสตริงได้

#### 2.2.1.1 นิยามพื้นฐานซัฟฟิกซ์ทรี

**นิยามที่ 2.4** ซัฟฟิกซ์ของสตริง T ที่มีความยาว n หรือ $S[s_i...s_n]$ ของสตริง T โดยที่ $1 \leq i \leq n$ คือ สตริงย่อยทั้งหมดของสตริง T ที่มีตำแหน่งเริ่มต้นที่ i และมีจุดสิ้นสุดอยู่ที่ตำแหน่ง n

**ตัวอย่างที่ 2.6** จากตัวอย่างที่ 2.2

จะเห็นว่าซัฟฟิกซ์ของสตริง T = ก ร ร ม ก ร
ร ร ม ก ร
ร ม ก ร
ม ก ร
ก ร
ร

**นิยามที่ 2.5** ซัฟฟิกซ์ทรีสำหรับสตริง T ที่มีจำนวน n อักขระ คือ โครงสร้างต้นไม้ที่เริ่มจากโนดราก(Root node) และมีจำนวนจุดสิ้นสุด หรือโนดใบ(Leaf node) เท่ากับ n โดยมีค่าตั้งแต่ 1

ถึง n โดยที่แต่ละโนดภายใน(Internal node) ที่ไม่ใช่โนดราก(Root node) จะมีโนดลูก(Child node) อย่างน้อย 2 โนด ซึ่งซัฟฟิกซ์ทรีจะแสดงซัฟฟิกซ์ทั้งหมดของสตริง T บนเส้นทางของต้นไม้โดยเริ่มจากโนดรากไปยังโนดใบที่มีจำนวน n โนด โดยที่จะมีการเพิ่มสัญลักษณ์ $ ต่อท้ายสตริง T ดังนั้น ทุก ๆ ซัฟฟิกซ์ของสตริง T จะมีสัญลักษณ์ $ ณ จุดสิ้นสุดที่ต่างกันบนซัฟฟิกซ์ทรี โดยจะมีจุดสิ้นสุดเท่ากับ n และโนดใบของซัฟฟิกซ์ทรีจะทำการเก็บตำแหน่งเริ่มต้นของแต่ละซัฟฟิกซ์

**นิยามที่ 2.6** เส้นทางเชื่อมต่อระหว่างโนดสองโนด(Edge) คือ ตัวแสดงถึงสัญลักษณ์ หรือ ตัวอักขระ ที่เป็นสมาชิกของเซตของตัวอักขระ ซึ่งเป็นสตริงย่อยที่ไม่ใช่สตริงว่างของสตริง T โดยที่แต่ละเอดสามารถเริ่มต้นจากสัญลักษณ์ หรือตัวอักขระเดียวกัน และสามารถมีสัญลักษณ์ หรือตัวอักขระมากกว่า 1 ตัวภายในเอดเดียวกันได้

### 2.2.1.2 การสร้างซัฟฟิกซ์ทรี

ซัฟฟิกซ์ทรี คือ โครงสร้างต้นไม้ที่เริ่มต้นจากโนดราก โดยที่ทุก ๆ เอดจะเริ่มด้วยตัวอักขระที่มีความแตกต่างกัน และทุก ๆ เส้นทางในต้นไม้(Path) ซึ่งเริ่มจากโนดรากจนถึงโนดใบจะเป็นตัวแสดงทุก ๆ ซัฟฟิกซ์ของสตริง

เราจะทำการแจงทุกซัฟฟิกซ์ของสตริงลงสู่ซัฟฟิกซ์ทรี และจะใช้สัญลักษณ์ $ เป็นตัวหยุดในทุก ๆ เส้นทางของต้นไม้ ซึ่งจะทำให้เราสามารถหาทุก ๆ สตริงย่อยที่เป็นไปได้ที่เกิดบนสตริงโดยการตรวจหาจากซัฟฟิกซ์ทรี ซึ่งมีขั้นตอนในการพัฒนาดังนี้

1) เพิ่ม $ เข้าไปต่อท้ายสตริง
2) แจงทุกซัฟฟิกซ์ของสตริง
   - ซัฟฟิกซ์ของสตริงที่มีความยาว n คือ สตริงย่อยของสตริงที่มีจุดสิ้นสุดอยู่ที่ตำแหน่ง n


3) สร้างเป็นซัฟฟิกซ์ทรี

ตัวอย่างต่อไปนี้ แสดงขั้นตอนของการสร้างซัฟฟิกซ์ทรีของสตริง T = (กรรมกร)

**ตัวอย่างที่ 2.7**

กำหนดให้ เซตของตัวอักขระ $\sum$ = {ก, ร, ม} และ สตริง T = (กรรมกร) ทำการสร้างซัฟฟิกซ์ทรี ดังนี้

1) เพิ่ม $ เข้าไปต่อท้ายสตริงพร้อมกำหนดหมายเลขประจำตำแหน่ง

สตริง T : ก ร ร ม ก ร $

หมายเลขประจำตำแหน่ง: 1 2 3 4 5 6 7

2) แจงซัฟฟิกซ์ทั้งหมด ดังนี้

1: ก ร ร ม ก ร $

2: ร ร ม ก ร $

3: ร ม ก ร $

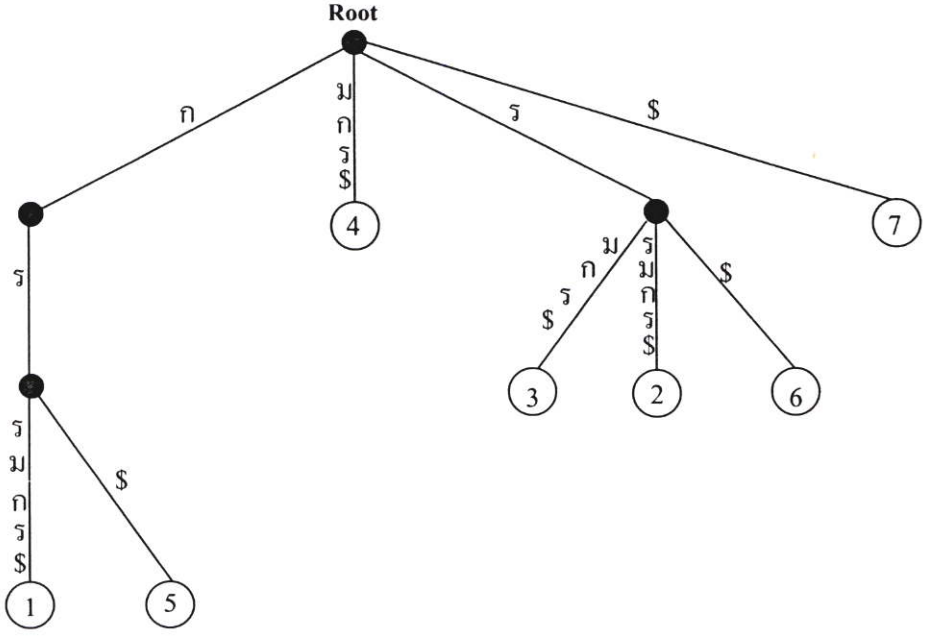
4: ม ก ร $

5: ก ร $

6: ร $

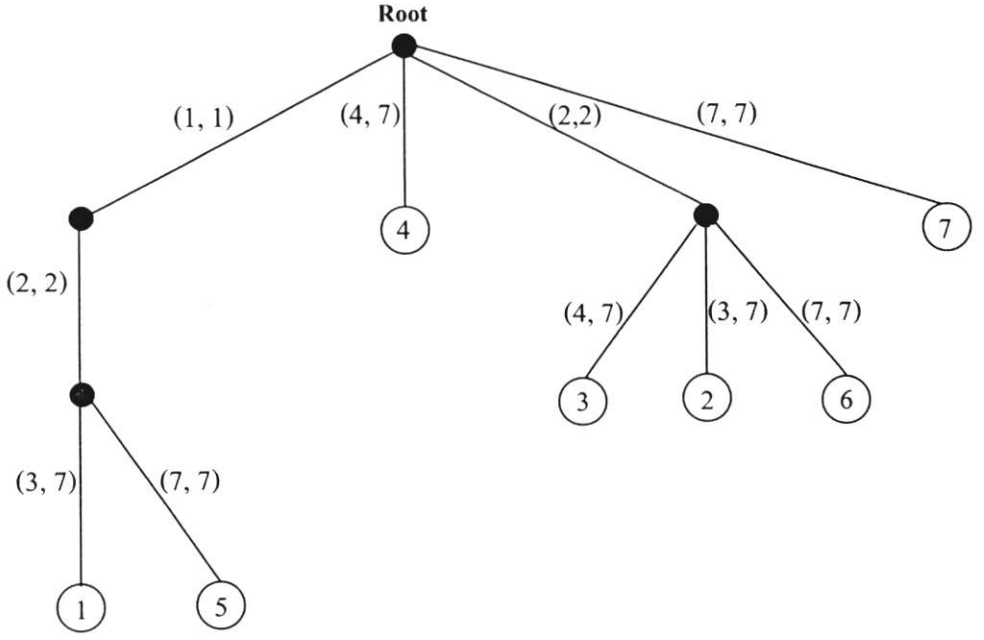
7: $

3) เราจะนำทุกซัฟฟิกซ์ของสตริง T มาสร้างซัฟฟิกซ์ทรี ได้ดังรูปที่ 2.1




**รูปที่ 2.1** แสดงซัฟฟิกซ์ทรีของสตริง T = (กรรมกร)

ข้อสังเกต จากรูปที่ 2.1 แสดงถึงซัฟฟิกซ์ทรีซึ่งแต่ละเอดจะมีการเก็บตัวอักขระ 1 ตัว หรือมากกว่า 1 ตัวใน 1 เอด ดังนั้น เราสามารถระบุตำแหน่งเริ่มต้นกับตำแหน่งสิ้นสุดให้กับแต่ละเอดได้ดังรูปที่ 2.2

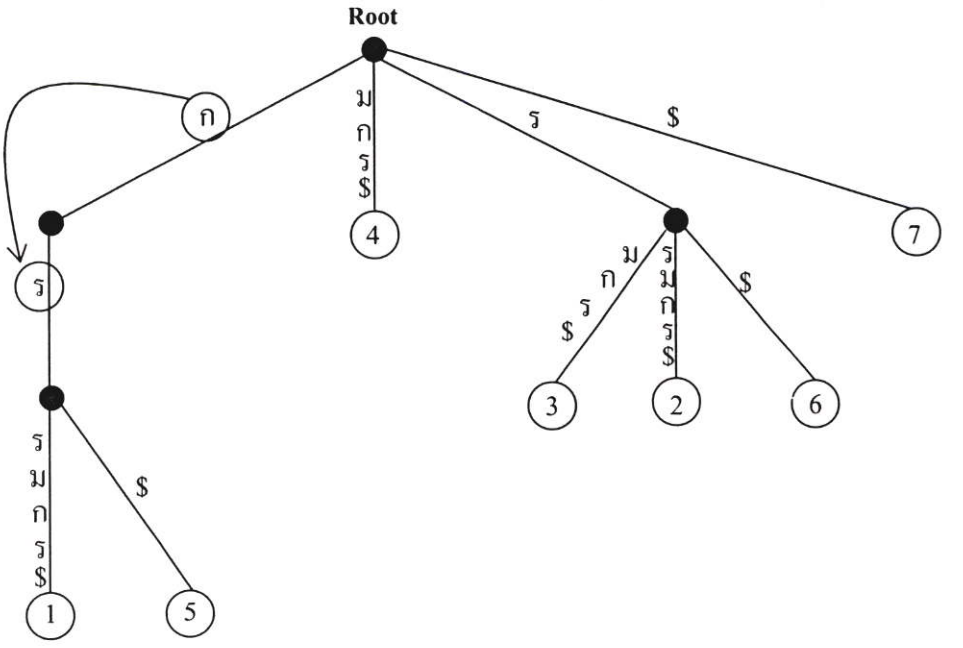



**รูปที่ 2.2** แสดงซัฟฟิกซ์ทรีของสตริง T = (กรรมกร) ที่มีการระบุตำแหน่งให้กับแต่ละเอด

สำหรับการค้นหาสตริงย่อยบนสตริงนั้น เราสามารถทำได้โดยการตรวจหาจากซัฟฟิกซ์ทรีโดยใช้เทคนิคของการท่องต้นไม้(Tree traversal) เนื่องจากเราสามารถสร้างเซตของสตริงย่อยที่เป็นไปได้ทั้งหมดที่เกิดขึ้นบนสตริงได้จากซัฟฟิกซ์ทรีดังตัวอย่างต่อไปนี้

**ตัวอย่างที่ 2.8** การค้นหาสตริงย่อยที่เกิดขึ้นบนสตริงจากซัฟฟิกซ์ทรีโดยใช้เทคนิคของการท่องต้นไม้

จากตัวอย่างที่ 2.7 ถ้าเราต้องการค้นหาสตริงย่อย p = [กร] และ x = [กรณ] เราสามารถค้นหาสตริงย่อยจากซัฟฟิกซ์ทรีได้ดังรูปที่ 2.3




**รูปที่ 2.3** แสดงขั้นตอนการค้นหาสตริงย่อยบนซัฟฟิกซ์ทรี

จากรูปที่ 2.3 จะแสดงให้เห็นว่า เราสามารถค้นหาสตริงย่อย p พบบนสตริง T แต่ค้นหาสตริงย่อย x ไม่พบบนสตริง T โดยใช้เทคนิคของการท่องต้นไม้

### 2.2.1.3 การพัฒนาขั้นตอนการสืบค้นสตริงย่อยบนสตริงโดยใช้ซัฟฟิกซ์ทรี

Weiner [12] ได้เสนอวิธีการที่เรียกว่า First linear-time algorithm เมื่อปี ค.ศ. 1973 ซึ่งเป็นผู้เริ่มต้นพัฒนาซัฟฟิกซ์ทรีเพื่อใช้ในการแยกแยะสตริงให้อยู่ภายในต้นไม้ เพื่อสนับสนุนการค้นหาสตริงย่อย โดยซัฟฟิกซ์ทรีจะถูกพัฒนาต่อมาจากซัฟฟิกซ์ทรัย ซึ่งซัฟฟิกซ์ทรีนั้นจะทำการลดความสูงของต้นไม้ โดยการจัดกลุ่มของสตริงย่อยที่ไม่มีการแตกกิ่งก้านสาขาให้อยู่ภายในเอดเดียวกัน ซัฟฟิกซ์ทรีจะใช้เวลา และเนื้อที่หน่วยความจำในการทำงานขึ้นอยู่กับความยาวของสตริง หรือ O(n) โดยที่ n คือ ความยาวของสตริง

Edward McCreight [5] ได้เสนอวิธีการที่เรียกว่า McCreight's suffix tree algorithm ซึ่งเป็นอัลกอริทึมที่พัฒนาต่อมาจาก First linear-time algorithm แต่จะมีการเพิ่มประสิทธิภาพทางด้านเนื้อที่หน่วยความจำ โดยการทำงานของอัลกอริทึมนั้น จะเริ่มจากการแจกแจงทุก ๆ

ซัฟฟิกซ์ ของสตริง ซึ่งจะเริ่มจาก 1 จนถึง n ซัฟฟิกซ์ แล้วนำลงสู่ซัฟฟิกซ์ทรีโดยทำการหาพรีฟิกซ์ที่มีความยาวที่สุดเพื่อทำการแตกกิ่งของต้นไม้ โดยที่ทุก ๆ จุดที่มีการแตกกิ่งของต้นไม้นั้น จะแสดงถึงเส้นทางของสัญลักษณ์ที่มีความแตกต่างกัน ซึ่งจะทำการแตกกิ่งของต้นไปจนถึงจุดสิ้นสุดของสตริง ก็จะทำให้ได้ซัฟฟิกซ์ทรีที่นำเสนอสตริง โดยใช้เนื้อที่ในการทำงานน้อยลงกว่า first linear-time algorithm

Esko Ukkonen[10] ได้พัฒนาให้เป็นขั้นตอนวิธีออนไลน์(on-line) ซึ่งเป็นอัลกอริทึมที่ได้มีการปรับปรุงเพียงเล็กน้อยจากอัลกอริทึมเดิม โดยจะมีการทำงานจากซ้ายไปขวา บนซัฟฟิกซ์ทรี ซึ่งอัลกอริทึมนี้จะเริ่มจากการสร้างต้นไม้ว่าง แล้วจะทำการเพิ่มแต่ละพลีฟิกซ์ของสตริงลงสู่ซัฟฟิกซ์ทรี โดยจะเริ่มจาก 1 จนถึง n พลีฟิกซ์ เมื่อสิ้นสุดการเพิ่มพลีฟิกซ์ทั้งหมดแล้ว ก็จะทำให้ได้ต้นไม้ที่สมบูรณ์ของสตริง อัลกอริทึมนี้ได้ถูกพัฒนาขึ้นเพื่อให้ง่ายต่อการทำงานและอธิบาย

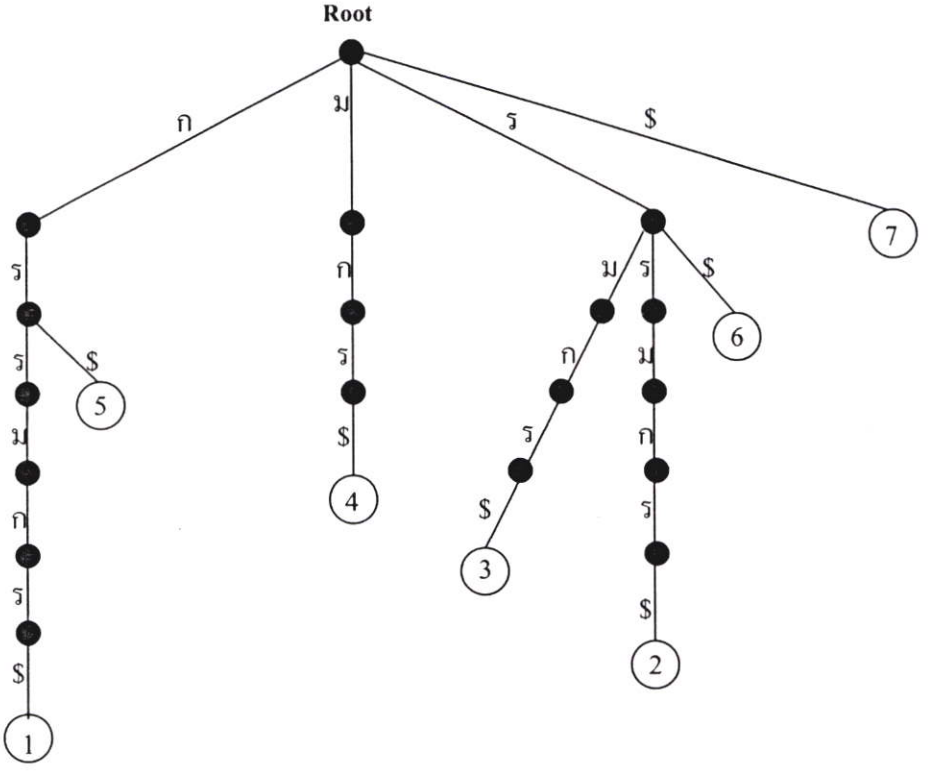
Giegerich [1] และคณะได้เสนอวิธีการ Wotd-algorithm หรือ writeonly-topdown algorithm โดยยึดหลักโครงสร้างแบบเรียกซ้ำของซัฟฟิกซ์ทรี เพื่อเป็นการเพิ่มประสิทธิภาพให้กับซัฟฟิกซ์ทรี ซึ่งจะใช้การทำงานแบบบนลงล่าง หรือจากรากของต้นไม้ไปจนถึงจุดสิ้นสุด แนวคิดคือ ทำการแตกกิ่งก้านให้กับทุก ๆ ซัฟฟิกซ์ และทำการเรียงลำดับโดยใช้ตัวอักขระตัวแรกของแต่ละซัฟฟิกซ์ ซึ่งจะทำให้ได้เซตของกลุ่มต้นไม้ย่อยที่อยู่ภายใต้แต่ละกิ่งที่ได้ทำการแตกโดยใช้ตัวอักขระตัวแรกของแต่ละ ซัฟฟิกซ์ จากนั้นจะทำการแตกกิ่งก้านให้กับเซตของกลุ่มต้นไม้ย่อยทั้งหมดโดยจะทำการเรียงลำดับโดยใช้ตัวอักขระตัวแรกของแต่ละกิ่งของต้นไม้ย่อย ซึ่งจะทำไปเรื่อย ๆ จนกว่าจะถึงจุดสิ้นสุด ซึ่งวิธีนี้จะทำให้ลดเวลาในการเข้าถึงข้อมูล เนื่องจากข้อมูลได้มีการเรียงลำดับไว้แล้ว

### 2.2.2 ซัฟฟิกซ์ทรัย

ซัฟฟิกซ์ทรัย คือ โครงสร้างต้นไม้ที่ใช้ในการนำเสนอทุก ๆ ซัฟฟิกซ์ของสตริงใด ๆ โดยที่ซัฟฟิกซ์ทรัยจะแตกต่างจากซัฟฟิกซ์ทรีตรงที่ ในทุก ๆ เอดของซัฟฟิกซ์ทรัย จะนำเสนอถึงสัญลักษณ์ หรือตัวอักขระซึ่งเป็นสมาชิกของเซตของตัวอักขระ โดยที่แต่ละเอดจะมีความยาวเท่ากับ 1 เสมอ หรือ จะมีการเก็บตัวอักขระเพียงตัวเดียวเท่านั้นใน 1 เอด

### 2.2.2.1 การสร้างซัฟฟิกซ์ทรัย

ซัฟฟิกซ์ทรัย คือ โครงสร้างข้อมูลที่มีลักษณะเหมือนต้นไม้ ซึ่งแสดงทุก ๆ ซัฟฟิกซ์ของสตริง บนเส้นทางของต้นไม้ ซึ่งเริ่มต้นจากโนดราก(Root node) และจบที่จุดสิ้นสุด หรือโนดใบ (Leaf node) ซึ่งจะบรรจุตำแหน่งเริ่มต้นของซัฟฟิกซ์ในเส้นทางนั้นไว้ โดยที่ทุก ๆ สตริงย่อยบนสตริงจะถูกแสดงอยู่ภายในต้นไม้ ซึ่งจะใช้เทคนิคของการท่องต้นไม้(Tree Traversal) เพื่อค้นหาสตริงย่อยที่ต้องการ ดังนั้นความสูงของต้นไม้จะแสดงถึงสตริงย่อยที่มีขนาดยาวที่สุดดังรูปที่ 2.4




**รูปที่ 2.4** แสดงซัฟฟิกซ์ทรัยของสตริง T = (กรรมกร)

### 2.2.2.2 การพัฒนาขั้นตอนการสืบค้นสตริงย่อยบนสตริงโดยใช้ซัฟฟิกซ์ทรัย

Morrison [8] ได้พัฒนาซัฟฟิกซ์ทรัยครั้งแรกในปี 1968 ซึ่งใช้ในการเก็บสตริง เพื่อสะดวกต่อการค้นหาสตริงย่อย โดยจะทำการแจงซัฟฟิกซ์ทั้งหมดลงสู่ต้นไม้ ซึ่งแต่ละเอดบนต้นไม้จะนำเสนอถึงสัญลักษณ์ หรือตัวอักขระที่เป็นสมาชิกของเซตของตัวอักขระ โดยจะมีความยาว 1 เสมอ และใช้เทคนิคในการท่องต้นไม้(Tree traversal) ในการค้นหาสตริงย่อยบนสตริง

Marti Hearst [4] ได้นำเสนอวิธีการใช้ซัฟฟิกซ์ทรัยเพื่อนำมาใช้ประโยชน์ทางด้านการสืบค้นสตริงย่อย โดยเพิ่มประสิทธิภาพให้กับซัฟฟิกซ์ทรัยด้วยการย่อให้ซัฟฟิกซ์ทรัยมีขนาดเล็กลงแต่ยังคงข้อมูลที่ครบถ้วน โดยใช้วิธีการคอมเพลสทรัย(Compressed Trie) ซึ่งเป็นการรวมเส้นทางที่ไม่มีการแตกกิ่งก้านสาขาให้อยู่ภายในเส้นทางเชื่อมต่อระหว่างโนดสองโนดเดียว ซึ่งจะเป็นการลดความสูงของซัฟฟิกซ์ทรัย

## 2.3 การสืบค้นสตริงย่อยบนสตริงโดยใช้ซัฟฟิกซ์อาเรย์

ซัฟฟิกซ์อาเรย์ของสตริง คือ อาเรย์ที่มีตำแหน่ง 0, 1, 2, …., n-1 ในสตริง โดยทุก ๆ ซัฟฟิกซ์ของสตริง จะถูกเก็บลงสู่อาเรย์ตั้งแต่ตำแหน่งที่ 0 ถึง n-1 สำหรับซัฟฟิกซ์อาเรย์นั้น สามารถสร้างขึ้นโดยตรงได้จากสตริง หรือ ทำการสร้างขึ้นจากซัฟฟิกซ์ทรัย โดยการอ่านค่าของทุก ๆ ซัฟฟิกซ์บนซัฟฟิกซ์ทรัย ซึ่งจะทำการอ่านตำแหน่งเริ่มต้นของทุก ๆ ซัฟฟิกซ์ถึงจุดสิ้นสุดของแต่ละเส้นทางในต้นไม้ โดยจะทำการเก็บค่าของซัฟฟิกซ์จากซ้ายไปขวา หรือ อาเรย์ที่ 0 จนถึงอาเรย์ที่ n

สำหรับซัฟฟิกซ์ทรัยนั้น ความจริงแล้วถือเป็นโครงสร้างต้นแบบที่ใช้ในการแก้ปัญหาต่าง ๆ แต่ซัฟฟิกซ์อาเรย์นั้น จะเป็นโครงสร้างข้อมูลที่สร้างขึ้นมาเพื่อใช้ในการทำงานจริง

### 2.3.1 การสร้างซัฟฟิกซ์อาเรย์

ซัฟฟิกซ์อาเรย์ คือ โครงสร้างข้อมูลที่ใช้เก็บข้อมูลสตริงจริงเพื่อใช้ในการสร้างโปรแกรม โดยจะมีขั้นตอนในการพัฒนาดังนี้

1) แจงทุกซัฟฟิกซ์ของสตริง
   - ซัฟฟิกซ์ของสตริงที่มีความยาว n คือ สตริงย่อยของสตริงที่มีจุดสิ้นสุดอยู่ที่ตำแหน่ง n


2) เก็บค่าซัฟฟิกซ์ทั้งหมดลงสู่อาเรย์ ณ ตำแหน่ง 0, 1, 2, …., n-1
3) เรียงลำดับซัฟฟิกซ์อาเรย์ใหม่ ซึ่งทำการเรียงลำดับตามตัวอักขระตัวแรกของแต่ละซัฟฟิกซ์

ตัวอย่างต่อไปนี้ แสดงขั้นตอนการสร้างซัฟฟิกซ์อาเรย์ของสตริง T = (กรรมกร)

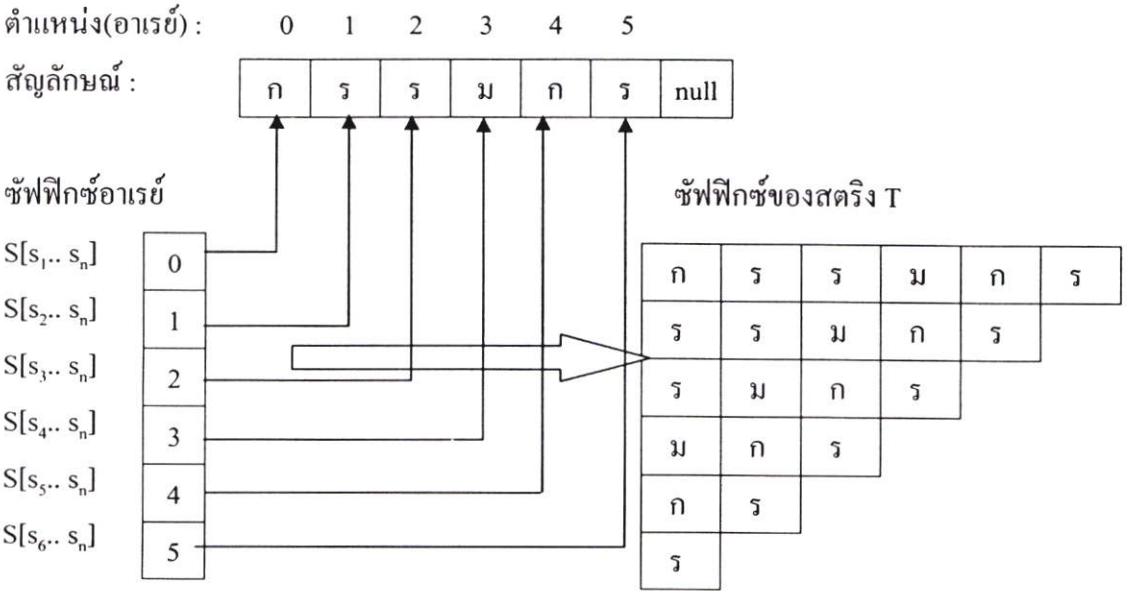
**ตัวอย่างที่ 2.9**

กำหนดให้ เซตของตัวอักขระ $\sum$ = {ก, ร, ม} และสตริง T = (กรรมกร) ทำการสร้างซัฟฟิกซ์อาเรย์ดังนี้

1) แจงทุกซัฟฟิกซ์ของสตริง T พร้อมกำหนดตำแหน่งของอาเรย์ให้กับแต่ละซัฟฟิกซ์ดังนี้

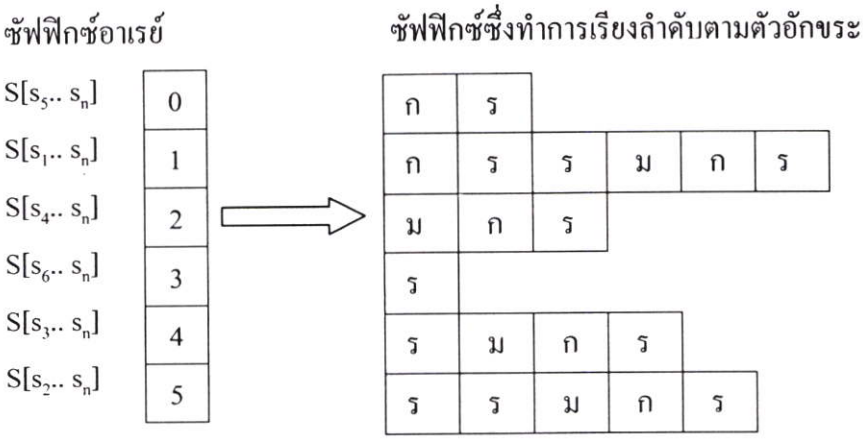
| ตำแหน่งอาเรย์ | ซัฟฟิกซ์ของสตริง T |
|---|---|
| 0 | 1: ก ร ร ม ก ร |
| 1 | 2: ร ร ม ก ร |
| 2 | 3: ร ม ก ร |
| 3 | 4: ม ก ร |
| 4 | 5: ก ร |
| 5 | 6: ร |

2) เก็บค่าซัฟฟิกซ์ทั้งหมดลงสู่อาเรย์ ณ ตำแหน่ง 0, 1, 2, …., n-1 ดังรูปที่ 2.5




**รูปที่ 2.5** แสดงซัฟฟิกซ์อาเรย์ที่เก็บซัฟฟิกซ์ทั้งหมดของสตริง T = (กรรมกร)

3) เรียงลำดับซัฟฟิกซ์อาเรย์ใหม่ โดยทำการเรียงลำดับตามตัวอักขระ ดังรูปที่ 2.6



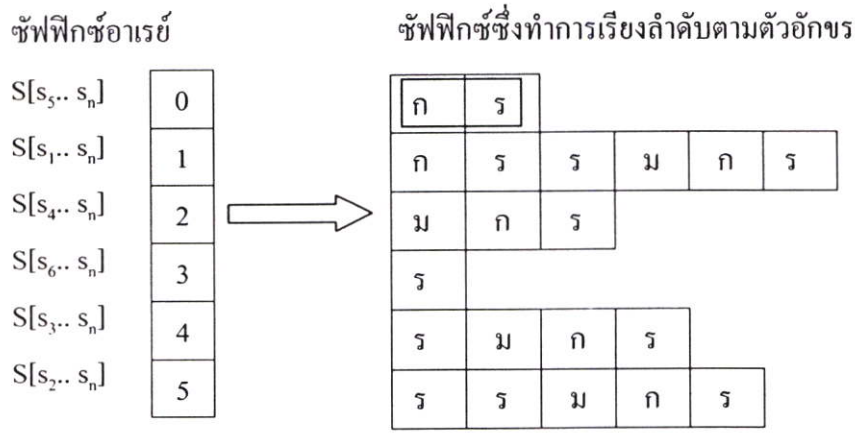

**รูปที่ 2.6** แสดงซัฟฟิกซ์อาเรย์ที่เก็บซัฟฟิกซ์ที่ได้ทำการเรียงลำดับตามตัวอักขระ

เมื่อเราทำการเก็บทุก ๆ ซัฟฟิกซ์ลงสู่ซัฟฟิกซ์อาเรย์โดยเรียงลำดับตามตัวอักขระแล้ว จะทำให้เราสามารถทำการค้นหาสตริงย่อยที่เกิดขึ้นบนสตริง T ได้ ดังตัวอย่างต่อไปนี้

**ตัวอย่างที่ 2.10** การค้นหาสตริงย่อยบนซัฟฟิกซ์อาเรย์

จากตัวอย่างที่ 2.9

ถ้าเราต้องการหาสตริงย่อย p = [กร] และ x = [กรณ] เราสามารถทำได้ดังรูปที่ 2.7




**รูปที่ 2.7** แสดงขั้นตอนการค้นหาสตริงย่อยบนซัฟฟิกซ์อาเรย์

จากรูปที่ 2.7 จะแสดงให้เห็นว่า เมื่อเราทำการค้นหาสตริงย่อยบนซัฟฟิกซ์อาเรย์ที่ได้ทำการเรียงลำดับตามตัวอักษร จะทำให้เราสามารถค้นหาสตริงย่อย p ที่เกิดขึ้นบนสตริง T ได้ แต่ไม่สามารถค้นหาสตริงย่อย x บนสตริง T ได้ ซึ่งการสืบค้นสตริงย่อยนั้น จะทำงานจากบนลงล่างบนซัฟฟิกซ์อาเรย์

### 2.3.2 การพัฒนาขั้นตอนการสืบค้นสตริงย่อยบนสตริงโดยใช้ซัฟฟิกซ์อาเรย์

Udi Manber และ Gene Myers [9] ได้พัฒนาซัฟฟิกซ์อาเรย์เพื่อใช้ในการเก็บสตริง เพื่อสะดวกต่อการค้นหาสตริงย่อย โดยซัฟฟิกซ์อาเรย์นั้น ถูกพัฒนาขึ้นเพื่อใช้งานจริง ซึ่งจะเริ่มจากการหาซัฟฟิกซ์ทุกตัวของสตริง แล้วนำทุกซัฟฟิกซ์ลงสู่ซัฟฟิกซ์อาเรย์ จากนั้นจะทำการเรียงลำดับซัฟฟิกซ์อาเรย์ใหม่ โดยใช้การเรียงลำดับตามสัญลักษณ์ หรือตัวอักขระ ซึ่งจะทำให้ได้ซัฟฟิกซ์อาเรย์ใหม่ที่ได้ทำการเรียงลำดับไว้แล้ว ทำให้สามารถค้นหาสตริงย่อยบนสตริงได้ โดยการค้นหาสตริงย่อยบน ซัฟฟิกซ์อาเรย์จะทำงานจากบนลงล่าง

Kenneth Ward Church [7] ได้นำเสนอวิธีการใช้ซัฟฟิกซ์อาเรย์เพื่อใช้ในการค้นหาสตริงย่อยที่มีความน่าสนใจบนสตริง และทำการคำนวณหาจำนวนความถี่ที่เกิดบนหนึ่งเอกสาร(Term frequency) และ จำนวนเอกสารทั้งหมดที่มีสตริงย่อยที่มีความน่าสนใจ(Document frequency) โดยการนำสตริงหลายชุดลงสู่ซัฟฟิกซ์อาเรย์เพื่อค้นหาสตริงย่อยที่เกิดขึ้นซ้ำกันบนหลายสตริงซึ่งจะทำให้ได้ผลลัพธ์ของจำนวนเอกสารทั้งหมดที่มีสตริงย่อยที่มีความน่าสนใจ และเก็บเป็นสถิติเพื่อใช้เป็นประโยชน์ทางด้านฐานข้อมูล หรือระบบเครือข่าย

## 2.4 การสืบค้นสตริงย่อยที่เกิดบ่อยบนสตริงโดยใช้ซัฟฟิกซ์ทรัยแบบระบุตำแหน่ง

การสืบค้นสตริงย่อยที่เกิดบ่อยบนสตริง คือ การสืบค้นทุก ๆ สตริงย่อยบนสตริงซึ่งเกิดขึ้นอย่างน้อย k ครั้งบนตำแหน่งที่แตกต่างกันในสตริง โดยที่ k คือค่าความถี่ขั้นต่ำ(Threshold) ซึ่งถูกพัฒนาโดยจ๊าก ไวโล่(Jaak Vilo)[3] ซึ่งอัลกอริทึมใหม่ที่ถูกพัฒนาขึ้นนี้จะทำการสืบค้นสตริงย่อยที่เกิดบ่อยจากสตริงบนซัฟฟิกซ์ทรัยแบบระบุตำแหน่ง โดยไม่ต้องทำการสร้างต้นไม้ทั้งต้น

### 2.4.1 การสร้างซัฟฟิกซ์ทรัยแบบระบุตำแหน่งโดยใช้อัลกอริทึมใหม่

อัลกอรึทึมใหม่นี้จะทำการสืบค้นสตริงย่อยที่มีค่าความถี่ผ่านค่าความถี่ขั้นต่ำบนซัฟฟิกซ์ทรัยแบบระบุตำแหน่ง โดยซัฟฟิกซ์ทรัยแบบระบุตำแหน่งนั้น จะมีคุณสมบัติเหมือนกับซัฟฟิกซ์ทรัยโดยทั่วไป แต่จะมีการเพิ่มรายการของตำแหน่งให้กับสตริงย่อย หรือ N($\alpha$).pos โดยที่ N($\alpha$).pos คือ รายการตำแหน่งของสตริงย่อย $\alpha$ บนสตริง T ซึ่งตำแหน่ง หรือ .pos คือ ตำแหน่งสิ้นสุดของสตริงย่อย $\alpha$ แทนด้วย j โดยที่อัลกอริทึมใหม่จะทำการสร้างรายการตำแหน่งให้กับสตริงย่อย $\alpha$ โดยใช้ตัวชี้ตำแหน่งที่ j + 1 เพื่อความสะดวกในการสร้างโนดลูกของสตริงย่อย $\alpha$ ซึ่งมีขั้นตอนในการพัฒนาดังนี้

1) เพิ่ม $ เข้าไปต่อท้ายสตริง
2) แจงทุกซัฟฟิกซ์ของสตริง
   - ซัฟฟิกซ์ของสตริงที่มีความยาว n คือ สตริงย่อยของสตริงที่มีจุดสิ้นสุดอยู่ที่ตำแหน่ง n


3) สร้างเป็นซัฟฟิกซ์ทรัยแบบระบุตำแหน่ง

ตัวอย่างต่อไปนี้ แสดงขั้นตอนการสร้างซัฟฟิกซ์ทรัยแบบระบุตำแหน่งของสตริง T = (กรรมกร)

**ตัวอย่างที่ 2.11**

กำหนดให้ เซตของตัวอักขระ $\sum$ = {ก, ร, ม} และ สตริง T = (กรรมกร) ทำการสร้างซัฟฟิกซ์ทรัยแบบระบุตำแหน่ง ดังนี้

1) เพิ่ม $ เข้าไปต่อท้ายสตริงพร้อมกำหนดตำแหน่ง(.pos)

| | |
|---|---|
| สตริง T : | ก ร ร ม ก ร $ |
| ตำแหน่ง(.pos) : | 1 2 3 4 5 6 7 |

2) แจงซัฟฟิกซ์ทั้งหมด ดังนี้

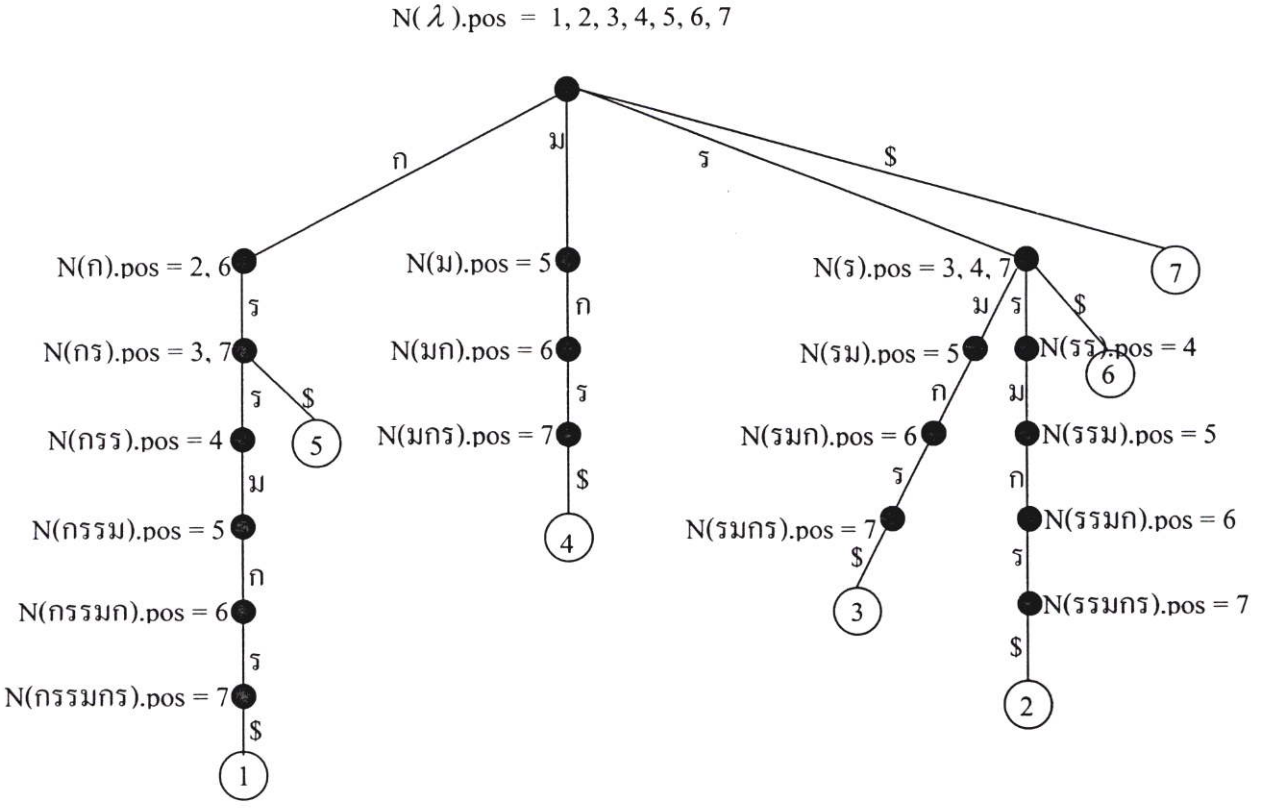
1: ก ร ร ม ก ร $
2: ร ร ม ก ร $
3: ร ม ก ร $
4: ม ก ร $

5: ก ร $

6: ร $

7: $

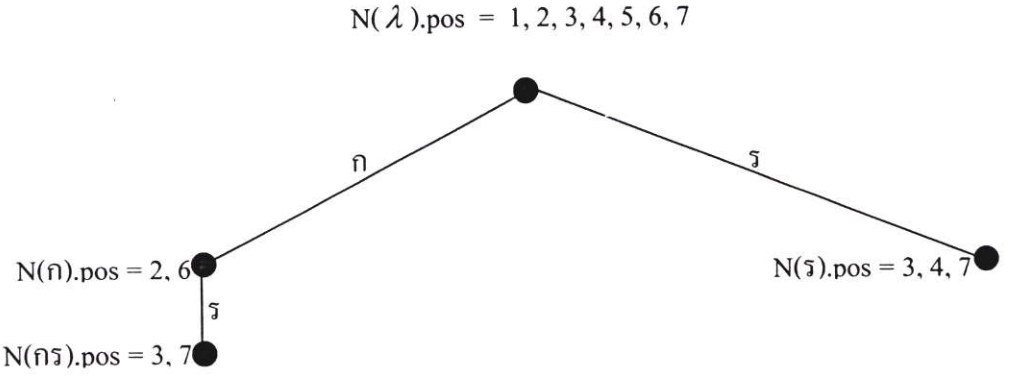
3) เราจะนำทุกซัฟฟิกซ์ของสตริง T มาสร้างซัฟฟิกซ์ทรัยแบบระบุตำแหน่ง ได้ดังรูปที่ 2.8




**รูปที่ 2.8** แสดงซัฟฟิกซ์ทรัยแบบระบุตำแหน่งของสตริง T = (กรรมกร)

เมื่อพิจารณาสตริงย่อยทั้งหมดพร้อมรายการตำแหน่งที่เกิดขึ้นบนซัฟฟิกซ์ทรัยแบบระบุตำแหน่งแล้ว พบว่าเราสามารถทำการสืบค้นสตริงย่อยที่เกิดบ่อยบนสตริงได้ โดยไม่ต้องทำการสร้างต้นไม้ทั้งต้น โดยการคำนวณจากตำแหน่งที่เกิดขึ้นของสตริงย่อย ซึ่งสตริงย่อยใดที่มีจำนวนการเกิดขึ้นบนสตริงน้อยกว่าค่าความถี่ขั้นต่ำ จะไม่ทำการสร้างสตริงย่อยที่มีความลึกมากกว่าสตริงย่อยนั้นในเส้นทางเดียวกันดังรูปที่ 2.9 แสดงซัฟฟิกซ์ทรัยแบบระบุตำแหน่งที่ใช้อัลกอริทึมใหม่ในการลดเส้นทางเพื่อสืบค้นสตริงย่อยที่เกิดบ่อยบนสตริง

**ตัวอย่างที่ 2.12** จากรูปที่ 2.8
กำหนดให้ค่าความถี่ขั้นต่ำ(Threshold) = 2




**รูปที่ 2.9** แสดงซัฟฟิกซ์ทรัยแบบระบุตำแหน่งโดยใช้อัลกอริทึมใหม่

จากรูปที่ 2.9 จะแสดงสตริงย่อยบนสตริง T = (กรรมกร) ซึ่งเกิดขึ้นอย่างน้อย 2 ครั้งในตำแหน่งที่แตกต่างกันบนสตริง T ซึ่งมีสตริงย่อยที่เกิดบ่อย คือ $\lambda$, ก, ร และ กร โดยการใช้อัลกอริทึมใหม่เพื่อสร้างรายการตำแหน่งของสตริงย่อยที่เกิดขึ้นเพื่อทำการลดการสร้างต้นไม้ทั้งต้น

สำหรับงานวิจัยทางด้านการสืบค้นสตริงย่อยที่เกิดบ่อยโดยใช้ซัฟฟิกซ์ทรัยแบบระบุตำแหน่งนั้น เราสามารถทำการพัฒนาต่อเพื่อทำการสืบค้น FM ได้ โดยทำการประยุกต์ซัฟฟิกซ์ทรัยแบบระบุตำแหน่งให้เป็นโครงสร้าง FST โดยทำการนับค่าจำนวนตำแหน่งให้เป็นค่าความถี่ เนื่องจากในการสืบค้น FM นั้น ถ้าหากมีการเก็บค่าความถี่ของสตริงย่อย จะเป็นประโยชน์ต่อการสืบค้น FM โดยใช้อัลกอริทึมที่มีประสิทธิภาพ

# บทที่ 3
# การสืบค้นรูปแบบสตริงย่อยแบบความยาวสูงสุดที่เกิดบ่อย

ในบทนี้จะนำเสนอนิยามพื้นฐานที่ใช้ในการสืบค้นรูปแบบสตริงย่อยแบบความยาวสูงสุดที่เกิดบ่อย หรือ FM สำหรับการสืบค้น FM คือ การสืบค้นหารูปแบบสตริงย่อยแบบความยาวสูงสุดต่าง ๆ ซึ่งมีค่าความถี่ผ่านความถี่ขั้นต่ำ โดยที่รูปแบบเหล่านี้ต้องไม่มีรูปแบบใดที่มีค่าความถี่ผ่านค่าความถี่ขั้นต่ำครอบคลุมอยู่ โดยทำการแจงนับรูปแบบสตริงย่อยทั้งหมดพร้อมความถี่ แล้วทำการคัดเลือก FM ตามค่าความถี่ขั้นต่ำที่กำหนด โดยใช้โครงสร้างข้อมูลใหม่ที่เรียกว่าซัฟฟิกซ์ทรัยแบบมีความถี่ หรือ FST ซึ่งมีคุณสมบัติบางประการที่เป็นประโยชน์ต่อการพัฒนาอัลกอริทึมในการสืบค้น FM โดยไม่ต้องทำการสร้างต้นไม้ทั้งต้น จึงทำให้สามารถทำการสืบค้น FM จากจำนวนรูปแบบสตริงย่อยที่มีขนาดเล็ก โดยในบทนี้จะแบ่งเนื้อหาออกเป็น 4 ส่วน คือ ส่วนแรกกล่าวถึงนิยามพื้นฐานที่ใช้ในการสืบค้น FM ส่วนที่ 2 กล่าวถึงอัลกอริทึมพื้นฐานที่ใช้ในการสืบค้น FM ส่วนที่ 3 กล่าวถึงอัลกอริทึมพื้นฐานที่ปรับปรุงแล้วที่ใช้ในการสืบค้น FM และส่วนที่ 4 กล่าวถึงอัลกอริทึมที่มีประสิทธิภาพที่ใช้ในการสืบค้น FM

## 3.1 นิยามพื้นฐาน

กำหนดให้ สตริง $T = (s_1, s_2, ....., s_n)$ โดยที่ $s_i \in \sum$ ซึ่ง $i \in \{1, 2, 3, 4, 5, 6, ……., n\}$ และ $\sum$ คือ เซตของตัวอักขระ ที่มีจำนวน N สัญลักษณ์ และค่าจำนวนเต็มบวก $\theta \geq 2$ คือ ค่าความถี่ขั้นต่ำ

**ตัวอย่างที่ 3.1**

กำหนดให้ สตริง T = (ก, า, ร, ป, ร, ะ, ก, อ, บ, ก, า, ร) จะเห็นว่า $s_1$ คือ ก, $s_2$ คือ า, $s_3$ คือ ร, $s_4$ คือ ป, $s_5$ คือ ร, $s_6$ คือ ะ, $s_7$ คือ ก, $s_8$ คือ อ, $s_9$ คือ บ, $s_{10}$ คือ ก, $s_{11}$ คือ า และ$s_{12}$ คือ ร

**นิยามที่ 3.1** เซตของรูปแบบสตริงย่อย(Substring Patterns) ความยาว m บน สตริง T หรือ $SP_{Tm}$ คือ เซตของสตริงย่อย $t = <s_j, ....., s_k : f>$ ของ T โดยที่ $1 \leq j \leq k \leq n$ , $m = k - j + 1$ และ f คือ ความถี่ของ t

**ตัวอย่างที่ 3.2**

กำหนดให้ สตริง T = (ก, า, ร, ป, ร, ะ, ก, อ, บ, ก, า, ร)

จะเห็นว่า $SP_{T1}$ = {<ก : 3>, <า : 2>, <ร : 3>, <ป : 1>, <ะ : 1>, <อ : 1>, <บ : 1>}

$SP_{T2}$ = {<ก, า : 2>, <า, ร : 2>, <ร, ป : 1>, <ป, ร : 1>, <ร, ะ : 1>, <ะ, ก : 1>, <ก, อ : 1>, <อ, บ : 1>, <บ, ก : 1>}

$SP_{T3}$ = {<ก, า, ร : 2>, <า, ร, ป : 1>, <ร, ป, ร : 1>, <ป, ร, ะ : 1>, <ร, ะ, ก : 1>, <ะ, ก, อ : 1>, <ก, อ, บ : 1>, <อ, บ, ก : 1>, <บ, ก, า : 1>}

$SP_{T4}$ = {<ก, า, ร, ป : 1>, <า, ร, ป, ร : 1>, <ร, ป, ร, ะ : 1>, <ป, ร, ะ, ก : 1>, <ร, ะ, ก, อ : 1>, <ะ, ก, อ, บ : 1>, <ก, อ, บ, ก : 1>, <อ, บ, ก, า : 1>, <บ, ก, า, ร : 1>}

$SP_{T5}$ = {<ก, า, ร, ป, ร : 1>, <า, ร, ป, ร, ะ : 1>, <ร, ป, ร, ะ, ก : 1>, <ป, ร, ะ, ก, อ : 1>, <ร, ะ, ก, อ, บ : 1> , <ะ, ก, อ, บ, ก : 1>, <ก, อ, บ, ก, า : 1>, <อ, บ, ก, า, ร : 1>}

$SP_{T6}$ = {<ก, า, ร, ป, ร, ะ : 1>, <า, ร, ป, ร, ะ, ก : 1>, <ร, ป, ร, ะ, ก, อ : 1>, <ป, ร, ะ, ก, อ, บ : 1>, <ร, ะ, ก, อ, บ, ก : 1>, <ะ, ก, อ, บ, ก, า : 1>, <ก, อ, บ, ก, า, ร : 1>}

$SP_{T7}$ = {<ก, า, ร, ป, ร, ะ, ก : 1>, <า, ร, ป, ร, ะ, ก, อ : 1>, <ร, ป, ร, ะ, ก, อ, บ : 1>, <ป, ร, ะ, ก, อ, บ, ก : 1>, <ร, ะ, ก, อ, บ, ก, า : 1>, <ะ, ก, อ, บ, ก, า, ร : 1>}

$SP_{T8}$ = {<ก, า, ร, ป, ร, ะ, ก, อ : 1>, <า, ร, ป, ร, ะ, ก, อ, บ : 1>, <ร, ป, ร, ะ, ก, อ, บ, ก : 1>, <ป, ร, ะ, ก, อ, บ, ก, า : 1>, <ร, ะ, ก, อ, บ, ก, า,ร : 1>}

$SP_{T9}$ = {<ก, า, ร, ป, ร, ะ, ก, อ, บ : 1>, <า, ร, ป, ร, ะ, ก, อ, บ, ก : 1>, <ร, ป, ร, ะ, ก, อ, บ, ก, า : 1> , <ป, ร, ะ, ก, อ, บ, ก, า, ร : 1>}

$SP_{T10}$ = {<ก, า, ร, ป, ร, ะ, ก, อ, บ, ก : 1>, <า, ร, ป, ร, ะ, ก, อ, บ, ก, า : 1>, <ร, ป, ร, ะ , ก, อ, บ, ก, า, ร : 1>}

$SP_{T11}$ = {<ก, า, ร, ป, ร, ะ, ก, อ, บ, ก,า : 1>, <า, ร, ป, ร, ะ, ก, อ, บ, ก, า, ร : 1>}

$SP_{Tn}$ = {<ก, า, ร, ป, ร, ะ, ก, อ, บ, ก, า, ร : 1>}

**ข้อสังเกต** จากตัวอย่างที่ 3.2 เมื่อพิจารณาถึง $SP_{Tm}$ ทั้งหมดที่เกิดขึ้นบน สตริง T หรือ $SP_T$ จะเห็นว่า

$SP_T = SP_{T1} \cup SP_{T2} \cup SP_{T3} \cup \ldots\ldots\ldots \cup SP_{Tn}$

**นิยาม 3.2** $\alpha = <a_i, \ldots\ldots, a_k>$ และ $\beta = <b_i, \ldots\ldots, b_j>$ เป็นสมาชิกใด ๆ ของ $SP_T$ $\alpha$ เป็น Substring ของ $\beta$ ก็ต่อเมื่อ $1 \leq i_1 < i_2 < \ldots. < i_k \leq j$ และ $a_1 = b_{i1}, a_2 = b_{i2}, \ldots\ldots, a_k \leq b_j$ ดังนั้น $\beta$ จึงเป็น Super Substring ของ $\alpha$ เขียนแทนด้วย

$$\alpha \sqsubseteq \beta$$

**ตัวอย่างที่ 3.3** จากตัวอย่างที่ 3.2

ถ้า $\alpha$ = <ก, า, ร : 2> และ $\beta$ = <ก, า, ร, ป, ร, ะ, ก, อ, บ : 1>

จะเห็นว่า $\alpha \sqsubseteq \beta$

**นิยามที่ 3.3** รูปแบบสตริงย่อยแบบความยาวสูงสุด (Max Substring Pattern) บน สตริง T หรือ $M_T$ คือ สมาชิกของ $SP_T$ ที่ไม่มีสมาชิกตัวอื่นของ $SP_T$ ที่เป็น Super Substring เขียนแทนด้วย

$$M_T = \{\alpha \in SP_T \mid \nexists\beta, \alpha \sqsubseteq \beta\}$$

**ตัวอย่างที่ 3.4** จากตัวอย่างที่ 3.2

$M_T$ = <ก, า, ร, ป, ร, ะ, ก, อ, บ, ก, า, ร : 1>

**นิยามที่ 3.4** เซตของรูปแบบสตริงย่อยที่เกิดบ่อย(Frequent Substring Patterns) บนสตริง T ที่มีความถี่อย่างน้อย $\theta$ หรือ $FSP_{T\theta}$ คือ สตริงย่อยของ T ที่มีความถี่มากกว่าหรือเท่ากับ $\theta$ โดยที่ $FSP_{T\theta} \in SP_T$ เขียนแทนด้วย

$$FSP_{T\theta} = \{\alpha \in SP_T \mid \text{ความถี่ของ}(\alpha) \geq \theta\}$$

**ตัวอย่างที่ 3.5** จากตัวอย่างที่ 3.2

ถ้ากำหนดให้ $\theta$ = 2

แล้วจะได้ว่า $FSP_{T2}$ = {<ก : 3>, <ร : 3>, <า : 2>, <ก, า : 2>, <า, ร : 2>, <ก, า, ร : 2>}

**นิยามที่ 3.5** เซตของรูปแบบสตริงย่อยแบบความยาวสูงสุดที่เกิดบ่อย (Frequent Max Substring Patterns) บนสตริง T หรือ $FM_{T\theta}$ คือ เซตที่ประกอบด้วยสมาชิก $\alpha \in FSP_{T\theta}$ โดยที่ ไม่มีสมาชิก $\beta \in FSP_{T\theta}$ ที่เป็น Super Substring ของ $\alpha$ เขียนแทนด้วย

$$FM_{T\theta} = \{\alpha \in FSP_{T\theta} \mid \nexists\beta \in FSP_{T\theta}, \alpha \sqsubseteq \beta\}$$

**ข้อสังเกต** จากนิยามที่3.5 เมื่อพิจารณาถึง $FM_{T\theta}$ ทั้งหมดที่เกิดขึ้น บนสตริง T จะแสดงให้เห็นว่า $FM_{T\theta} \in FSP_{T\theta}$

**ตัวอย่างที่ 3.6** จากตัวอย่างที่ 3.5

$FM_{T2}$ = {<ก, า, ร : 2>}

**ตัวอย่างที่ 3.7** จากตัวอย่างที่ 3.2 ถึง 3.6

กำหนดให้ สตริง T = (ก, า, ร, ป, ร, ะ, ก, อ, บ, ก, า, ร,) และ $\theta$ = 2

จะเห็นว่า $SP_{T1}$ = {<ก : 3>, <า : 2>, <ร : 3>, <ป : 1>, <ะ : 1>, <อ : 1>, <บ : 1>}

$SP_{T2}$ = {<ก, า : 2>, <า, ร : 2>, <ร, ป : 1>, <ป, ร : 1>, <ร, ะ : 1>, <ะ, ก : 1>, <ก, อ : 1>, <อ, บ : 1>, <บ, ก : 1>}

$SP_{T3}$ = {<ก, า, ร : 2>, <า, ร, ป : 1>, <ร, ป, ร : 1>, <ป, ร, ะ : 1>, <ร, ะ, ก : 1>, <ะ, ก, อ : 1>, <ก, อ, บ : 1>, <อ, บ, ก : 1>, <บ, ก, า : 1>}

$SP_{T4}$ = {<ก, า, ร, ป : 1>, <า, ร, ป, ร : 1>, <ร, ป, ร, ะ : 1>, <ป, ร, ะ, ก : 1>, <ร, ะ, ก, อ : 1>, <ะ, ก, อ, บ : 1>, <ก, อ, บ, ก : 1>, <อ, บ, ก, า : 1>, <บ, ก, า, ร : 1>}

$SP_{T5}$ = {<ก, า, ร, ป, ร : 1>, <า, ร, ป, ร, ะ : 1>, <ร, ป, ร, ะ, ก : 1>, <ป, ร, ะ, ก, อ : 1>, <ร, ะ, ก, อ, บ : 1> , <ะ, ก, อ, บ, ก : 1>, <ก, อ, บ, ก, า : 1>, <อ, บ, ก, า, ร : 1>}

$SP_{T6}$ = {<ก, า, ร, ป, ร, ะ : 1>, <า, ร, ป, ร, ะ, ก : 1>, <ร, ป, ร, ะ, ก, อ : 1>, <ป, ร, ะ, ก, อ, บ : 1>, <ร, ะ, ก, อ, บ, ก : 1>, <ะ, ก, อ, บ, ก, า : 1>, <ก, อ, บ, ก, า, ร : 1>}

$SP_{T7}$ = {<ก, า, ร, ป, ร, ะ, ก : 1>, <า, ร, ป, ร, ะ, ก, อ : 1>, <ร, ป, ร, ะ, ก, อ, บ : 1>, <ป, ร, ะ, ก, อ, บ, ก : 1>, <ร, ะ, ก, อ, บ, ก, า : 1>, <ะ, ก, อ, บ, ก, า, ร : 1>}

$SP_{T8}$ = {<ก, า, ร, ป, ร, ะ, ก, อ : 1>, <า, ร, ป, ร, ะ, ก, อ, บ : 1>, <ร, ป, ร, ะ, ก, อ, บ, ก : 1>, <ป, ร, ะ, ก, อ, บ, ก, า : 1>, <ร, ะ, ก, อ, บ, ก, า,ร : 1>}

$SP_{T9}$ = {<ก, า, ร, ป, ร, ะ, ก, อ, บ : 1>, <า, ร, ป, ร, ะ, ก, อ, บ, ก : 1>, <ร, ป, ร, ะ, ก, อ, บ, ก, า : 1> , <ป, ร, ะ, ก, อ, บ, ก, า, ร : 1>}

$SP_{T10}$ = {<ก, า, ร, ป, ร, ะ, ก, อ, บ, ก : 1>, <า, ร, ป, ร, ะ, ก, อ, บ, ก, า : 1>, <ร, ป, ร, ะ , ก, อ, บ, ก, า, ร : 1>}

$SP_{T11}$ = {<ก, า, ร, ป, ร, ะ, ก, อ, บ, ก,า : 1>, <า, ร, ป, ร, ะ, ก, อ, บ, ก, า, ร : 1>}

$SP_{Tn}$ = {<ก, า, ร, ป, ร, ะ, ก, อ, บ, ก, า, ร : 1>}

$SP_{T}$ = {<ก : 3>, <า : 2>, <ร : 3>, <ป : 1>, <ะ : 1>, <อ : 1>, <บ : 1>, <ก, า : 2>, <า, ร : 2>, <ร, ป : 1>, <ป, ร : 1>, <ร, ะ : 1>, <ะ, ก : 1>, <ก, อ : 1>, <อ, บ : 1>, <บ, ก : 1>, <ก, า, ร : 2>, <า, ร, ป : 1>, <ร, ป, ร : 1>, <ป, ร, ะ : 1>, <ร, ะ, ก : 1>, <ะ, ก, อ : 1>, <ก, อ, บ : 1>, <อ, บ, ก : 1>, <บ, ก, า : 1>, <ก, า, ร, ป : 1>, <า, ร, ป, ร : 1>, <ร, ป, ร, ะ : 1>, <ป, ร, ะ, ก : 1>, <ร, ะ, ก, อ : 1>, <ะ, ก, อ, บ : 1>, <ก, อ, บ, ก : 1>, <อ, บ, ก, า : 1>, <บ, ก, า, ร : 1>, <ก, า, ร, ป, ร : 1>, <า, ร, ป, ร, ะ : 1>, <ร, ป, ร, ะ, ก : 1>, <ป, ร, ะ, ก, อ : 1>,

<ร, ะ, ก, อ, บ : 1> , <ะ, ก, อ, บ, ก : 1>, <ก, อ, บ, ก, า : 1>, <อ, บ, ก, า, ร : 1>, <ก, า, ร, ป, ร, ะ : 1>, <า, ร, ป, ร, ะ, ก : 1>, <ร, ป, ร, ะ, ก, อ : 1>, <ป, ร, ะ, ก, อ, บ : 1>, <ร, ะ, ก, อ, บ, ก : 1>, <ะ, ก, อ, บ, ก, า : 1>, <ก, อ, บ, ก, า, ร : 1>, <ก, า, ร, ป, ร, ะ, ก : 1>, <า, ร, ป, ร, ะ, ก, อ : 1>, <ร, ป, ร, ะ, ก, อ, บ : 1>, <ป, ร, ะ, ก, อ, บ, ก : 1>, <ร, ะ, ก, อ, บ, ก, า : 1>, <ะ, ก, อ, บ, ก, า, ร : 1>, <ก, า, ร, ป, ร, ะ, ก, อ : 1>, <า, ร, ป, ร, ะ, ก, อ, บ : 1>, <ร, ป, ร, ะ, ก, อ, บ, ก : 1>, <ป, ร, ะ, ก, อ, บ, ก, า : 1>, <ร, ะ, ก, อ, บ, ก, า,ร : 1>, <ก, า, ร, ป, ร, ะ, ก, อ, บ : 1>, <า, ร, ป, ร, ะ, ก, อ, บ, ก : 1>, <ร, ป, ร, ะ, ก, อ, บ, ก, า : 1> , <ป, ร, ะ, ก, อ, บ, ก, า, ร : 1>, <ก, า, ร, ป, ร, ะ, ก, อ, บ, ก : 1>, <า, ร, ป, ร, ะ, ก, อ, บ, ก, า : 1>, <ร, ป, ร, ะ , ก, อ, บ, ก, า, ร : 1>, <ก, า, ร, ป, ร, ะ, ก, อ, บ, ก,า : 1>, <า, ร, ป, ร, ะ, ก, อ, บ, ก, า, ร : 1>, <ก, า, ร, ป, ร, ะ, ก, อ, บ, ก, า, ร : 1>},

$M_T$ = <ก, า, ร, ป, ร, ะ, ก, อ, บ, ก, า, ร>,

$FSP_{T2}$ = {<ก : 3>, <ร : 3>, <า : 2>, <ก, า : 2>, <า, ร : 2>, <ก, า, ร : 2>},

และ $FM_{T2}$ = {<ก, า, ร : 2>}

## 3.2 อัลกอริทึมพื้นฐาน

สำหรับอัลกอริทึมพื้นฐานที่ใช้ในการสืบค้น FM นั้น จำเป็นจะต้องมีโครงสร้างข้อมูลที่ใช้ในการแจงนับรูปแบบสตริงย่อยทั้งหมดบนสตริง พร้อมค่าความถี่ ดังนั้น จึงเกิดแนวความคิดในการพัฒนาโครงสร้าง FST เพื่อใช้เป็นโครงสร้างข้อมูลพื้นฐานในการสืบค้น FM

### 3.2.1 โครงสร้าง FST

โครงสร้าง FST คือ โครงสร้างต้นไม้ที่ใช้ในการแจงนับรูปแบบสตริงย่อยทั้งหมดที่ปรากฏขึ้นบนสตริงพร้อมเก็บค่าความถี่ และตำแหน่ง ซึ่งใช้เป็นตัวสนับสนุนการสืบค้น FM

#### 3.2.1.1 นิยามพื้นฐานโครงสร้าง FST

**นิยามที่ 3.6** ซัฟฟิกซ์ของสตริง T ที่มีความยาว n หรือ $S[s_i...s_n]$ ของสตริง T โดยที่ $1 \leq i \leq n$ คือ สตริงย่อยทั้งหมดของสตริง T ที่มีตำแหน่งเริ่มต้นที่ i และมีจุดสิ้นสุดอยู่ที่ตำแหน่ง n

**ตัวอย่างที่ 3.8** จากตัวอย่างที่ 3.1

จะเห็นว่าซัฟฟิกซ์ของสตริง T = การประกอบการ

ารประกอบการ

รประกอบการ

ประกอบการ

ระกอบการ

ะกอบการ

กอบการ

อบการ

บการ

การ

าร

ร

**นิยามที่ 3.7** โครงสร้าง FST สำหรับสตริง T ที่มีจำนวน n อักขระ คือ โครงสร้างต้นไม้ที่เริ่มจากโนดราก(Root node) และมีจำนวนจุดสิ้นสุด หรือโนดใบ(Leaf node) เท่ากับ n โดยมีค่าตั้งแต่ 1 ถึง n ซึ่งแสดงรูปแบบสตริงย่อยทั้งหมดพร้อมความถี่ของสตริง T โดยที่ ซัฟฟิกซ์ทั้งหมดของสตริง T จะถูกแสดงบนเส้นทางของต้นไม้โดยเริ่มจากโนดรากไปยังโนดใบที่มีจำนวน n โนด โดยที่จะมีการเพิ่มสัญลักษณ์ $ ต่อท้ายสตริง T ดังนั้น ทุก ๆ ซัฟฟิกซ์ของสตริง T จะมีสัญลักษณ์ $ ณ จุดสิ้นสุดที่ต่างกันบนโครงสร้าง FST โดยจะมีจุดสิ้นสุดเท่ากับ n

**นิยามที่ 3.8** เส้นทางเชื่อมต่อระหว่างโนดสองโนด(Edge) คือ ตัวแสดงถึงสัญลักษณ์ หรือ ตัวอักขระ ที่เป็นสมาชิกของเซตของตัวอักขระ โดยที่แต่ละเอดสามารถเริ่มต้นจากตัวอักขระเดียวกันและทุก ๆ เอดบนโครงสร้าง FST จะเก็บสัญลักษณ์ หรือตัวอักขระที่มีความยาว 1 เสมอ

**นิยามที่ 3.9** โนด(Node) คือ ตัวแสดงถึงรูปแบบสตริงย่อยพร้อมความถี่ และรายการตำแหน่งของรูปแบบสตริงย่อย หรือ .pos บนสตริง T ซึ่งจะแทนด้วยตำแหน่งสิ้นสุดของรูปแบบสตริงย่อย หรือ k โดยที่ทุก ๆ ความลึกของโนดจะมีการเพิ่มขึ้นของความยาวของรูปแบบสตริงย่อย ซึ่งโนดใบทั้งหมดของโครงสร้าง FST จะทำการเก็บซัฟฟิกซ์พร้อมความถี่และรายการตำแหน่งของซัฟฟิกซ์ในเส้นทางนั้นไว้

### 3.2.1.2 การสร้างโครงสร้าง FST

โครงสร้าง FST คือ โครงสร้างต้นไม้ที่แสดงทุก ๆ ซัฟฟิกซ์พร้อมทั้งแจกแจงความถี่ของแต่ละรูปแบบสตริงย่อยของสตริงบนเส้นทางของต้นไม้ โดยมีสัญลักษณ์ $ เป็นตัวแสดงถึงจุดสิ้นสุดของสตริง โดยที่การแจงนับแต่ละรูปแบบสตริงย่อยจะเรียงลำดับตามตำแหน่งที่เกิดขึ้นของรูปแบบสตริงย่อยบนสตริง เนื่องจากลำดับมีความสำคัญต่อการเป็น FM โดยที่รูปแบบสตริงย่อยที่เกิดขึ้นบนสตริงก่อนจะมีโอกาสเป็นสมาชิกของ FM มากกว่ารูปแบบสตริงย่อยที่ตามมา ซึ่งแสดงขั้นตอนการพัฒนาดังนี้

1) เพิ่ม $ เข้าไปต่อท้ายสตริงเพื่อเพิ่มจุดสิ้นสุดให้กับสตริง
2) แจงทุก ๆ ซัฟฟิกซ์ของสตริง
3) สร้างเป็นโครงสร้าง FST

ตัวอย่างต่อไปนี้แสดงขั้นตอนของการสร้างโครงสร้าง FST ของสตริง T = (ก, า, ร, ป, ร, ะ, ก, อ, บ, ก, า, ร,)

**ตัวอย่างที่ 3.9**

กำหนดให้สตริง T = (ก, า, ร, ป, ร, ะ, ก, อ, บ, ก, า, ร) ทำการสร้างโครงสร้าง FST ดังนี้

1) เพิ่ม $ เข้าไปต่อท้ายสตริงพร้อมกำหนดหมายเลขประจำตำแหน่ง

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| สตริง T : | ก | า | ร | ป | ร | ะ | ก | อ | บ | ก | า | ร | $ |
| ตำแหน่ง (.pos) : | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |

2) แจงซัฟฟิกซ์ทั้งหมดของสตริง ดังนี้

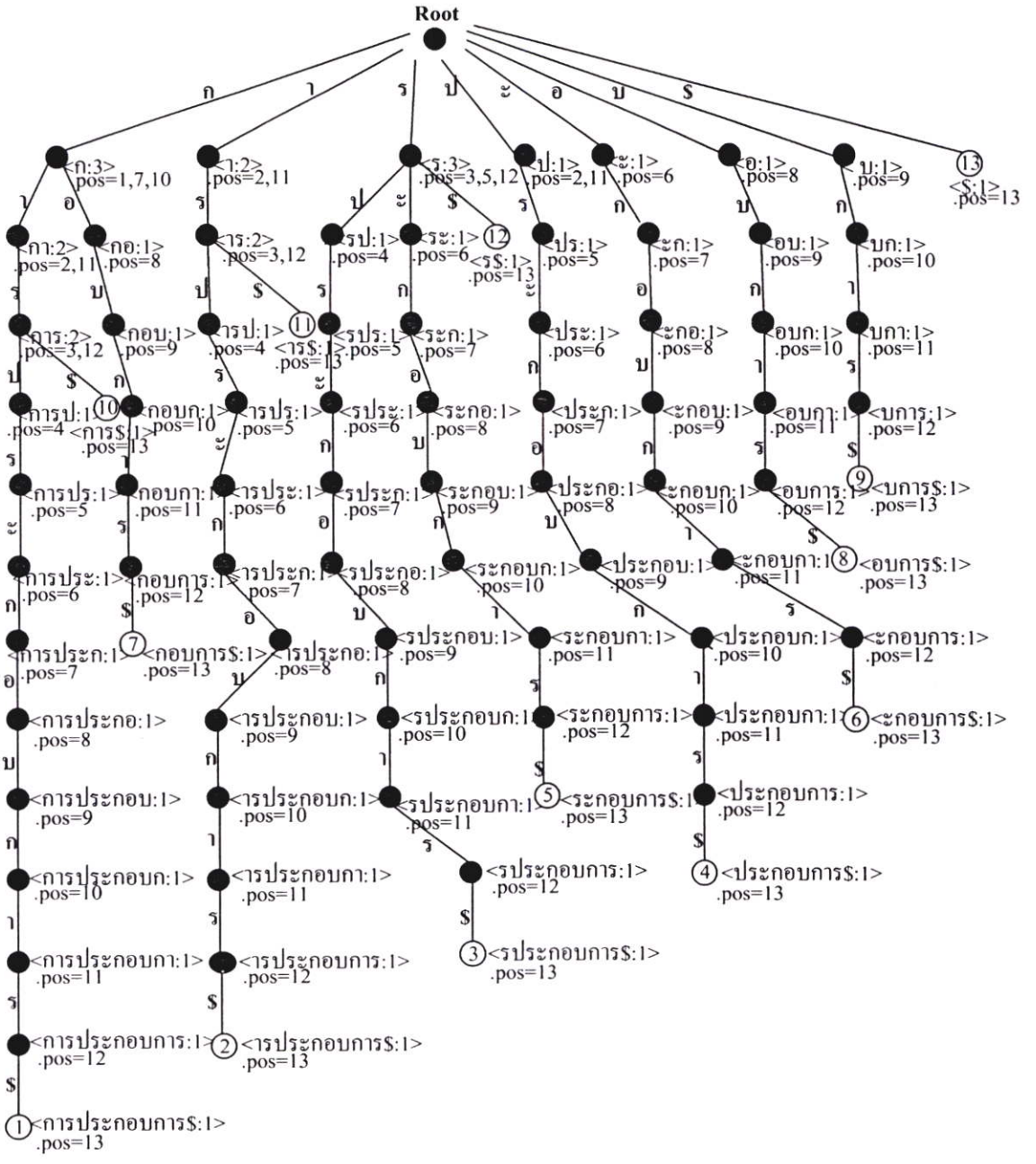
1: ก า ร ป ร ะ ก อ บ ก า ร $
2: า ร ป ร ะ ก อ บ ก า ร $
3: ร ป ร ะ ก อ บ ก า ร $
4: ป ร ะ ก อ บ ก า ร $
5: ร ะ ก อ บ ก า ร $
6: ะ ก อ บ ก า ร $
7: ก อ บ ก า ร $
8: อ บ ก า ร $
9: บ ก า ร $
10: ก า ร $

11: า ร $

12: ร $

13: $

3) นำซัฟฟิกซ์ทั้งหมดมาสร้างโครงสร้าง FST ดังรูปที่ 3.1




**รูปที่ 3.1** แสดงโครงสร้าง FST ของสตริง T = (การประกอบการ)

เมื่อพิจารณารูปแบบสตริงย่อยทั้งหมดที่เกิดขึ้นบนสตริง T หรือ $SP_T$ จากรูปที่ 3.1 ดังนั้น การสืบค้น $FM_{T\theta}$ จาก สตริง T ภายใต้ค่า $\theta$ มีขั้นตอนดังต่อไปนี้

1. ทำการแจงนับรูปแบบสตริงย่อยทั้งหมด หรือ $SP_T$ บน สตริง T โดยใช้โครงสร้าง FST
2. ทำการสืบค้นเซตของรูปแบบสตริงย่อยที่เกิดบ่อย หรือ $FSP_{T\theta}$ โดยการคัดเลือกรูปแบบสตริงย่อยที่มีความถี่มากกว่าหรือเท่ากับ $\theta$ จากโครงสร้าง FST
3. ทำการสืบค้นรูปแบบสตริงย่อยแบบความยาวสูงสุดที่เกิดบ่อย หรือ $FM_{T\theta}$ โดยคัดเลือกรูปแบบสตริงย่อยที่ไม่มีรูปแบบสตริงย่อยใดเป็น Super Substring จากเซตของรูปแบบสตริงย่อยที่เกิดบ่อย หรือ $FSP_{T\theta}$

จากขั้นตอนการทำงานของอัลกอริทึมพื้นฐาน จะเห็นว่า การสืบค้น $FM_{T\theta}$ ต้องเริ่มจากการแจงนับรูปแบบสตริงย่อยที่เป็นไปได้ทั้งหมด หรือ $SP_T$ ซึ่งมีจำนวนของรูปแบบสตริงย่อยจำนวนมาก กล่าวคือ ถ้า $|T| = n$ แล้ว $|SP_T| = O(n^2)$ เนื่องจาก $SP_T$ ประกอบด้วยรูปแบบสตริงย่อยที่มีความยาว $m = 1$ ไปจนถึงรูปแบบสตริงย่อยที่มีความยาว $m = n$ โดยจะแสดงให้เห็นถึงขนาดของ $SP_T$ ที่มีขนาดใหญ่ ซึ่งจำเป็นที่จะต้องใช้เนื้อที่หน่วยความจำสูง ดังนั้น แนวทางการปรับปรุงของอัลกอริทึมพื้นฐานนี้ ทำได้โดยการลดขนาดของ $SP_T$ โดยใช้ $\theta$ มาเป็นตัวตรวจสอบการหยุดการสืบค้นรูปแบบสตริงย่อย โดยตรวจสอบว่า เมื่อทำการแจงนับรูปแบบสตริงย่อยใด ๆ แล้วพบว่าสตริงย่อยนั้น มีค่าความถี่น้อยว่า $\theta$ จะหยุดการแจงนับรูปแบบสตริงย่อยที่อยู่ลึกกว่าสตริงย่อยนั้นในเส้นทางเดียวกัน เนื่องจากคุณสมบัติของโครงสร้าง FST โนดลูกจะมีค่าความถี่น้อยกว่าหรือเท่ากับโนดพ่อแม่เสมอ ดังนั้นการสืบค้นรูปแบบสตริงย่อยจึงไม่ต้องทำจนกระทั่งความยาว $m = n$ สำหรับการปรับปรุงอัลกอริทึมพื้นฐานนั้น สามารถทำได้โดยการพิจารณาค่าความถี่ของรูปแบบสตริงย่อยบนโครงสร้าง FST ซึ่งจะทำให้ได้ $FSP_{T\theta}$ ครบถ้วนและถูกต้องโดยไม่ต้องทำการสร้างต้นไม้ทั้งต้น อีกทั้งอัลกอริทึมพื้นฐานที่ปรับปรุงแล้วยังทำการเก็บตำแหน่งที่เกิดขึ้นของรูปแบบสตริงย่อยบนสตริงเพื่อลดเวลาในการแจงนับรูปแบบสตริงย่อย

### 3.2.2 กฎการลดเส้นทางโดยการนำ $\theta$ มาเป็นตัวตรวจสอบการหยุดการสืบค้น

เมื่อพิจารณาคุณสมบัติของโครงสร้าง FST แล้ว เราจะพบว่ารูปแบบสตริงย่อยที่เป็นพ่อแม่จะมีค่าความถี่มากกว่าหรือเท่ากับรูปแบบสตริงย่อยที่เป็นลูกเสมอ เนื่องจากรูปแบบสตริงย่อยที่เป็นพ่อแม่นั้น จะถูกกระจายออกไปเป็น Substring ของรูปแบบสตริงย่อยที่เป็นลูก เพราะว่า รูปแบบสตริงย่อยที่เป็นพ่อแม่นั้น จะมีความยาว m น้อยกว่า รูปแบบสตริงย่อยที่เป็นลูกอยู่ 1 เสมอ ดังนั้น เมื่อทำการแจงนับรูปแบบสตริงย่อยใด ๆ แล้ว พบว่าสตริงย่อยนั้น มีค่าความถี่น้อยว่า $\theta$

จะหยุดการแจงนับรูปแบบสตริงย่อยที่อยู่ลึกกว่ารูปแบบสตริงย่อยนั้นในเส้นทางเดียวกันบนโครงสร้าง FST เขียนแทนด้วย

$$\beta \sqsupseteq \alpha \implies \text{ค่าความถี่}(\beta) \leq \text{ค่าความถี่}(\alpha)$$

โดยที่ $\alpha$ และ $\beta$ คือ รูปแบบสตริงย่อยในเส้นทางเดียวกันบนโครงสร้าง FST

## 3.3 อัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว

อัลกอริทึมพื้นฐานที่ปรับปรุงแล้วจะทำการสืบค้น $FM_{T\theta}$ โดยใช้คุณสมบัติของโครงสร้าง FST ในการแจงนับเฉพาะรูปแบบสตริงย่อยที่เป็นสมาชิกของ $FSP_{T\theta}$ โดยใช้กฎการลดเส้นทางโดยการนำ $\theta$ มาเป็นตัวตรวจสอบการหยุดการสืบค้น พร้อมใส่รายการตำแหน่งให้กับรูปแบบสตริงย่อย หรือ .pos โดยที่ .pos คือ รายการตำแหน่งของรูปแบบสตริงย่อย $\alpha$ บนสตริง T ซึ่งจะแทนด้วยตำแหน่งสิ้นสุดของรูปแบบสตริงย่อย $\alpha$ เพื่อลดเวลาในการแจงนับรูปแบบสตริงย่อยลูกของรูปแบบสตริงย่อย $\alpha$ โดยจะใช้โครงสร้างข้อมูลแบบ Min Heap ซึ่งเป็นโครงสร้างข้อมูลคิวชนิดพิเศษที่ทำการแทรกข้อมูลของรูปแบบสตริงย่อยลงสู่โครงสร้างข้อมูลตามลำดับการเกิดขึ้นก่อนและหลังบนสตริง T โดยรูปแบบสตริงย่อยที่เกิดขึ้นก่อนบนสตริง T จะได้ทำงานก่อนรูปแบบสตริงย่อยที่เกิดขึ้นหลังบนสตริง T เนื่องจากรูปแบบสตริงย่อยที่เกิดก่อนจะมีโอกาสเป็นรูปแบบสตริงย่อยที่เป็นสมาชิกของ $FM_{T\theta}$ ดังนั้น เราจึงนำโครงสร้างข้อมูลแบบ Min Heap เข้ามาช่วยในการทำงานของอัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว ดังตัวอย่างที่ 3.10 แสดงการแจงนับรูปแบบสตริงย่อยโดยใช้อัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว

**ตัวอย่างที่ 3.10**

กำหนดให้ สตริง T = (ก, า, ร, ป, ร, ะ, ก, อ, บ, ก, า, ร) และ $\theta$ = 2
ขั้นตอนการสืบค้น $FM_{T\theta}$ โดยใช้อัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว มีดังนี้

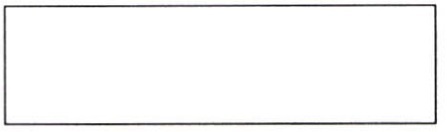
1. ทำการแจงนับรูปแบบสตริงย่อยที่มีความยาว m = 1 พร้อมความถี่ แล้วทำการคัดเลือกเฉพาะรูปแบบสตริงย่อยที่มีค่าความถี่มากกว่าหรือเท่ากับ $\theta$ พร้อมเก็บรูปแบบสตริงย่อย, ค่าความถี่ และรายการตำแหน่ง หรือ .pos ของรูปแบบสตริงย่อยเหล่านี้ลงสู่โครงสร้างข้อมูลแบบ Min Heap ตามลำดับการเกิดขึ้นก่อนและหลังบนสตริง T เนื่องจากรูปแบบสตริงย่อยที่เกิดก่อนนั้น มีโอกาสที่จะเป็นรูปแบบสตริงย่อยที่เป็นสมาชิกของ $FM_{T\theta}$ มากกว่ารูปแบบสตริงย่อยที่เกิดหลัง

2. ทำการประมวลผลกับรูปแบบสตริงย่อยในโครงสร้างข้อมูลแบบ Min Heap ตามลำดับ โดยจะทำการแจงนับรูปแบบสตริงย่อยลูกของรูปแบบสตริงย่อยที่ถูกดึงออกมาจากโครงสร้างข้อมูลแบบ Min Heap เพื่อทำการประมวลผล แล้วทำการคัดเลือกเฉพาะรูปแบบสตริงย่อยลูกที่มีค่าความถี่มากกว่าหรือเท่ากับ θ พร้อมเก็บข้อมูลของรูปแบบสตริงย่อยลูกเหล่านั้นลงสู่โครงสร้างข้อมูลแบบ Min Heap โดยใช้กฎการแทรกข้อมูลของโครงสร้างข้อมูลแบบ Min Heap คือ 1. รูปแบบสตริงย่อยจะถูกแทรกลงสู่โครงสร้างข้อมูลแบบ Min Heap ตามตำแหน่งแรกที่เกิดขึ้นบนสตริง T และ 2. ถ้าตำแหน่งแรกของรูปแบบสตริงย่อยที่จะถูกแทรกลงสู่โครงสร้างข้อมูลแบบ Min Heap มีค่าเท่ากับตำแหน่งแรกของรูปแบบสตริงย่อยที่มีอยู่เดิมในโครงสร้างข้อมูลแบบ Min Heap ให้ทำการแทรกรูปแบบสตริงย่อยใหม่ในตำแหน่งหลังสุดของกลุ่มของรูปแบบสตริงย่อยเดิมที่อยู่ในโครงสร้างข้อมูลแบบ Min Heap ที่มีค่าตำแหน่งแรกที่เท่ากัน ซึ่งรูปแบบสตริงย่อยใดที่ได้ทำการประมวลผลแล้ว จะถูกทำการลบออกจากโครงสร้างข้อมูลแบบ Min Heap โดยจะทำการประมวลผลกับรูปแบบสตริงย่อยไปเรื่อย ๆ จนกระทั่งโครงสร้างข้อมูลแบบ Min Heap ว่าง
3. ทำการสืบค้น $FM_{T\theta}$ โดยคัดเลือกรูปแบบสตริงย่อยที่ไม่มีรูปแบบสตริงย่อยใดเป็น Super Substring จาก $FSP_{T\theta}$

ตัวอย่างต่อไปนี้ แสดงวิธีการทำงานของอัลกอริทึมพื้นฐานที่ปรับปรุงแล้วโดยใช้โครงสร้างข้อมูลแบบ Min Heap ในการแจงนับเฉพาะรูปแบบสตริงย่อยที่เป็นสมาชิกของ $FSP_{T\theta}$

สตริง T = ก า ร ป ร ะ ก อ บ ก า ร $
ตำแหน่ง(.pos) = 1 2 3 4 5 6 7 8 9 10 11 12 13

**โครงสร้างข้อมูลแบบ Min Heap**



ทำการแจงนับรูปแบบสตริงย่อยด้วยความยาว m = 1 และตรวจสอบค่าความถี่ของรูปแบบสตริงย่อยทุกตัว โดยจะทำการเก็บข้อมูลเฉพาะรูปแบบสตริงย่อยที่มีค่าความถี่มากกว่า หรือเท่ากับ θ

| ก:3 | า:2 | ร:3 |
|---|---|---|
| .pos=1, 7, 10 | .pos=2, 11 | .pos=3, 5,12 |

นำ <ก:3> ออกจากโครงสร้างข้อมูลแบบ Min Heap พร้อมกับสืบค้นรูปแบบสตริงย่อยลูกของ <ก:3> โดยใช้รายการตำแหน่งของ <ก:3> ซึ่งจะได้รูปแบบสตริงย่อย คือ <กา:2> และ<กอ:1> และทำการเก็บข้อมูลของ <กา:2> ลงโครงสร้างข้อมูลแบบ Min Heap โดยใช้กฎการแทรกข้อมูลของโครงสร้างข้อมูลแบบ Min Heap เนื่องจาก <กา:2> มีค่าความถี่เท่ากับ θ

| า:2 | กา:2 | ร:3 |
|---|---|---|
| .pos=2, 11 | .pos=2, 11 | pos=3, 5,12 |

นำ <า:2> ออกจากโครงสร้างข้อมูลแบบ Min Heap พร้อมกับสืบค้นรูปแบบสตริงย่อยลูกของ <า:2> โดยการใช้รายการตำแหน่งของ <า:2> ซึ่งจะได้รูปแบบสตริงย่อย คือ <าร:2> และทำการเก็บข้อมูลของ <าร:2> ลงโครงสร้างข้อมูลแบบ Min Heap โดยใช้กฎการแทรกข้อมูลของโครงสร้างข้อมูลแบบ Min Heap เนื่องจาก <าร:2> มีค่าความถี่เท่ากับ θ

| กา:2 | ร:3 | าร:2 |
|---|---|---|
| pos=2, 11 | .pos=3, 5, 12 | .pos=3, 12 |

นำ <กา:2> ออกจากโครงสร้างข้อมูลแบบ Min Heap พร้อมกับสืบค้นรูปแบบสตริงย่อยลูกของ <กา:2> โดยการใช้รายการตำแหน่งของ <กา:2> ซึ่งจะได้รูปแบบสตริงย่อย คือ <การ:2> และทำการเก็บข้อมูลของ <การ:2> ลงโครงสร้างข้อมูลแบบ Min Heap โดยใช้กฎการแทรกข้อมูลของโครงสร้างข้อมูลแบบ Min Heap เนื่องจาก <การ:2> มีค่าความถี่เท่ากับ θ

| ร:3 | าร:2 | การ:2 |
|---|---|---|
| pos=3, 5, 12 | .pos=3, 12 | .pos=3, 12 |

นำ <ร:3> ออกจากโครงสร้างข้อมูลแบบ Min Heap พร้อมกับสืบค้นรูปแบบสตริงย่อยลูกของ <ร:3> โดยการใช้รายการตำแหน่งของ <ร:3> ซึ่งจะได้รูปแบบสตริงย่อย คือ <รป:1>, <ระ:1> และ<ร$:1> ซึ่งจะไม่ทำการเก็บข้อมูลลงโครงสร้างข้อมูลแบบ Min Heap เนื่องจาก <รป:1>, <ระ:1> และ<ร$:1> มีค่าความถี่น้อยกว่า θ

| าร:2 | การ:2 | |
|---|---|---|
| .pos=3, 12 | .pos=3, 12 | |

นำ <าร:2> ออกจากโครงสร้างข้อมูลแบบ Min Heap พร้อมกับสืบค้นรูปแบบสตริงย่อยลูกของ <าร:2> โดยการใช้รายการตำแหน่งของ <าร:2> ซึ่งจะได้รูปแบบสตริงย่อย คือ <ารป:1> และ <าร$:1> ซึ่งจะไม่ทำการเก็บข้อมูลลงโครงสร้างข้อมูลแบบ Min Heap เนื่องจาก <ารป:1> และ <าร$:1> มีค่าความถี่น้อยกว่า $\theta$

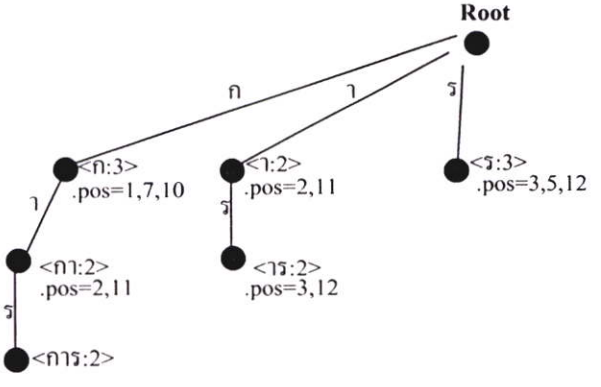
| การ:2<br>.pos=3, 12 | |
|---|---|

นำ <การ:2> ออกจากโครงสร้างข้อมูลแบบ Min Heap พร้อมกับสืบค้นรูปแบบสตริงย่อยลูกของ <การ:2> โดยการใช้รายการตำแหน่งของ <การ:2> ซึ่งจะได้รูปแบบสตริงย่อย คือ <การป:1> และ <การ$:1> ซึ่งจะไม่ทำการเก็บข้อมูลลงโครงสร้างข้อมูลแบบ Min Heap เนื่องจาก <การป:1> และ <การ$:1> มีค่าความถี่น้อยกว่า $\theta$

| |
|---|

เมื่อโครงสร้างข้อมูลแบบ Min Heap ว่าง จะแสดงถึงการเสร็จสิ้นการทำงานของอัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว

จากตัวอย่างที่ 3.10 เมื่อพิจารณาขั้นตอนการทำงานของอัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว โดยใช้โครงสร้างข้อมูลแบบ Min Heap จะทำให้ได้โครงสร้าง FST ซึ่งมีเฉพาะรูปแบบสตริงย่อยที่เป็นสมาชิกของ $FSP_{T\theta}$ ดังรูปที่ 3.2




**รูปที่ 3.2** แสดงโครงสร้าง FST โดยใช้ขั้นตอนการทำงานของอัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว

จากรูปที่ 3.2 แสดงโครงสร้าง FST ที่ได้มาจากขั้นตอนการทำงานของอัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว ซึ่งจะได้ผลของ

$FSP_{T2}$ = {<ก : 3>, <ร : 3>, <า : 2>, <ก, า : 2>, <า, ร : 2>, <ก, า, ร : 2>}

และ $FM_{T2}$ = {<ก, า, ร : 2>}

โดยสามารถนำข้อมูลลงสู่ตารางที่ 3.1 ดังนี้

**ตารางที่ 3.1** แสดง $FSP_{T\theta}$ และ $FM_{T\theta}$ บน สตริง T = (ก, า, ร, ป, ร, ะ, ก, อ, บ, ก, า, ร) ที่มี $\theta = 2$

$SP_{Tm}$

| m | t | f |
|---|---|---|
| 1 | า | 2 |
| 2 | กา | 2 |
| 2 | าร | 2 |
| 1 | ก | 3 |
| 1 | ร | 3 |
| 3 | การ | 2 |

$FSP_{T2}$ →

$FM_{T2}$ →

การหา $FM_{T\theta}$ ดังกล่าวนั้น สามารถสรุปการทำงานได้ดังนี้ เราจะทำการแจงนับรูปแบบสตริงย่อยขนาดความยาวต่าง ๆ จากสตริง โดยใช้กฎการลดเส้นทางโดยการนำ $\theta$ มาเป็นตัวตรวจสอบการหยุดการสืบค้น ซึ่งจะทำให้ได้เฉพาะ $FSP_{T\theta}$ โดยใช้โครงสร้างข้อมูลแบบ Min Heap และทำการหา $FM_{T\theta}$ จาก $FSP_{T\theta}$

เมื่อพิจารณาขั้นตอนการทำงานของอัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว ยังคงมีข้อด้อยทางด้านจำนวนการแจงนับรูปแบบสตริงย่อยที่เป็นสมาชิกของ FSP แต่ไม่ได้เป็นสมาชิกของ FM อยู่เป็นจำนวนมาก ทำให้ต้องเสียเวลาในการสืบค้น FM จาก FSP ดังนั้น แนวทางการปรับปรุงให้อัลกอริทึมมีประสิทธิภาพนั้น เราจะทำโดยการแจงนับรูปแบบสตริงย่อยคำตอบที่เป็นไปได้ต่อการสืบค้น FM เท่านั้น ซึ่งจะทำให้ลดเวลาในการสืบค้น FM โดยการลดจำนวนการแจงนับรูปแบบสตริงย่อยสำหรับอัลกอริทึมที่มีประสิทธิภาพนั้น เราสามารถทำได้โดยใช้กฎการลดเส้นทาง 2 ข้อ คือ 1. กฎการลดเส้นทางโดยการนำ $\theta$ มาเป็นตัวตรวจสอบการหยุดการสืบค้น และ 2. กฎการลดเส้นทางโดยใช้นิยามการเป็น Substring ต่อกัน เพื่อนำมาลดจำนวนรูปแบบสตริงย่อยที่ไม่จำเป็นต่อการสืบค้น FM

### 3.3.1 กฎการลดเส้นทางโดยใช้นิยามการเป็น Substring ต่อกัน

เมื่อพิจารณานิยามการเป็น Substring และ Super Substring ต่อกัน ทำให้เห็นว่า เราสามารถทำการลดการแจงนับรูปแบบสตริงย่อย เนื่องจาก เมื่อเราทำการสืบค้นรูปแบบสตริงย่อยที่มีความยาว m = 1 และ m = 2 และพบว่า มีรูปแบบสตริงย่อยที่มีความยาว m = 2 บางตัวที่มีคุณสมบัติการเป็น Super Substring ของรูปแบบสตริงย่อยที่มีความยาว m = 1 ตัวใดตัวหนึ่งด้วยค่าความถี่เท่ากับค่าความถี่ของรูปแบบสตริงย่อยที่มีความยาว m = 1 ลบ ค่าความถี่ของรูปแบบสตริงย่อยที่มีความยาว m = 2 แล้วมีค่าน้อยกว่า $\theta$ เราก็จะสามารถหยุดทำการแจงนับรูปแบบสตริงย่อยในเส้นทางของรูปแบบสตริงย่อยที่มีความยาว m = 1 นั้นได้ เขียนแทนด้วย

$$\beta \sqsupseteq \alpha \Longrightarrow (\text{ค่าความถี่}(\alpha) - \text{ค่าความถี่}(\beta)) < \theta$$

โดยที่ $\alpha$ และ $\beta$ คือ รูปแบบสตริงย่อยบนโครงสร้าง FST

## 3.4 อัลกอริทึมที่มีประสิทธิภาพ

แนวความคิดในการพัฒนาอัลกอริทึมที่มีประสิทธิภาพ จะใช้กฎการลดเส้นทาง 2 ข้อ ที่ใช้ในการลดจำนวนการสืบค้นรูปแบบสตริงย่อย คือ 1. กฎการลดเส้นทางโดยการนำ $\theta$ มาเป็นตัวตรวจสอบการหยุดการสืบค้น และ 2. กฎการลดเส้นทางโดยใช้นิยามการเป็น Substring ต่อกันมาเป็นตัวตรวจสอบการหยุดการแจงนับรูปแบบสตริงย่อยโดยใช้โครงสร้างข้อมูลแบบ Min Heap โดยการทำงานของอัลกอริทึมที่มีประสิทธิภาพจะมีการแทรกข้อมูลของรูปแบบสตริงย่อยลงสู่โครงสร้างข้อมูลแบบ Min Heap อีกทั้งยังมีการปรับปรุง และลบข้อมูลของรูปแบบสตริงย่อยในโครงสร้างข้อมูลแบบ Min Heap เพื่อให้สอดคล้องกับกฎการลดเส้นทางโดยใช้นิยามการเป็น Substring ต่อกัน ซึ่งจะทำให้ลดจำนวนรูปแบบสตริงย่อยเพื่อนำไปสู่การลดเวลาในการสืบค้น FM ดังตัวอย่างที่ 3.11 แสดงการแจงนับรูปแบบสตริงย่อยโดยใช้อัลกอริทึมที่มีประสิทธิภาพ

**ตัวอย่างที่ 3.11**

กำหนดให้ สตริง T = (ก, า, ร, ป, ร, ะ, ก, อ, บ, ก, า, ร) และ $\theta$ = 2

ขั้นตอนการสืบค้น $FM_{T\theta}$ โดยใช้อัลกอริทึมที่มีประสิทธิภาพ มีดังนี้

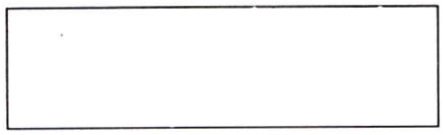
1. ทำการแจงนับรูปแบบสตริงย่อยที่มีความยาว m = 1 พร้อมความถี่ แล้วทำการคัดเลือกเฉพาะรูปแบบสตริงย่อยที่มีค่าความถี่มากกว่าหรือเท่ากับ $\theta$ พร้อมเก็บรูปแบบสตริงย่อย, ค่าความถี่ และรายการตำแหน่ง หรือ .pos ของรูปแบบสตริงย่อยเหล่านี้ลงสู่โครงสร้างข้อมูลแบบ Min Heap ตามลำดับการเกิดขึ้นก่อนและหลังบนสตริง T เนื่องจากรูปแบบสตริงย่อยที่เกิดก่อนนั้น มีโอกาสที่จะเป็นรูปแบบสตริงย่อยที่เป็นสมาชิกของ $FM_{T\theta}$ มากกว่ารูปแบบสตริงย่อยที่เกิดหลัง
2. ทำการประมวลผลกับรูปแบบสตริงย่อยในโครงสร้างข้อมูลแบบ Min Heap ตามลำดับ โดยจะทำการแจงนับรูปแบบสตริงย่อยลูกของรูปแบบสตริงย่อยที่ถูกดึงออกมาจากโครงสร้างข้อมูลแบบ Min Heap เพื่อทำการประมวลผล แล้วทำการคัดเลือกเฉพาะรูปแบบสตริงย่อยลูกที่มีค่าความถี่มากกว่าหรือเท่ากับ $\theta$ จากนั้นจะทำการปรับปรุงโครงสร้างข้อมูลแบบ Min Heap โดยใช้กฎการลบข้อมูลรูปแบบสตริงย่อยบนโครงสร้างข้อมูลแบบ Min Heap ซึ่งจะทำการตรวจสอบว่ารูปแบบสตริงย่อยลูกตัวใดบ้างที่เป็น Super Substring ของรูปแบบสตริงย่อยที่มีความยาวน้อยกว่าที่อยู่ในโครงสร้างข้อมูลแบบ Min Heap โดยที่ค่าความถี่จะต้องเท่ากับค่าความถี่ของรูปแบบสตริงย่อยที่เป็น Substring ลบด้วยค่าความถี่ของรูปแบบสตริงย่อยที่เป็น Super Substring แล้วมีค่าน้อยกว่า $\theta$ จะทำการลบรูปแบบสตริงย่อยที่เป็น Substring ออกจากโครงสร้างข้อมูลแบบ Min Heap จากนั้นจะทำการเก็บข้อมูลของรูปแบบสตริงย่อยลูกที่มีค่าความถี่มากกว่าหรือเท่ากับ $\theta$ ลงสู่โครงสร้างข้อมูลแบบ Min Heap โดยใช้กฎการแทรกข้อมูลของโครงสร้างข้อมูลแบบ Min Heap คือ 1. รูปแบบสตริงย่อยจะถูกแทรกลงสู่โครงสร้างข้อมูลแบบ Min Heap ตามตำแหน่งแรกที่เกิดขึ้นบนสตริง T และ 2. ถ้าตำแหน่งแรกของรูปแบบสตริงย่อยที่จะถูกแทรกลงสู่โครงสร้างข้อมูลแบบ Min Heap มีค่าเท่ากับตำแหน่งแรกของรูปแบบสตริงย่อยที่มีอยู่เดิมในโครงสร้างข้อมูลแบบ Min Heap ให้ทำการแทรกรูปแบบสตริงย่อยใหม่ในตำแหน่งหลังสุดของกลุ่มของรูปแบบสตริงย่อยเดิมที่อยู่ในโครงสร้างข้อมูลแบบ Min Heap ที่มีค่าตำแหน่งแรกที่เท่ากัน ซึ่งรูปแบบสตริงย่อยใดที่ได้ทำการประมวลผลแล้ว จะถูกทำการลบออกจากโครงสร้างข้อมูลแบบ Min Heap โดยจะทำการประมวลผลกับรูปแบบสตริงย่อยไปเรื่อย ๆ จนกระทั่งโครงสร้างข้อมูลแบบ Min Heap ว่าง
3. ทำการสืบค้น $FM_{T\theta}$ โดยคัดเลือกรูปแบบสตริงย่อยที่ไม่มีรูปแบบสตริงย่อยใดเป็น Super Substring จากเซตของรูปแบบสตริงย่อยที่ได้จากโครงสร้างข้อมูลแบบ Min Heap

ตัวอย่างต่อไปนี้ แสดงวิธีการทำงานของอัลกอริทึมที่มีประสิทธิภาพโดยใช้โครงสร้างข้อมูลแบบ Min Heap

สตริง T = ก า ร ป ร ะ ก อ บ ก า ร $

.pos = 1 2 3 4 5 6 7 8 9 10 11 12 13

**โครงสร้างข้อมูลแบบ Min Heap**



ทำการแจงนับรูปแบบสตริงย่อยด้วยความยาว m = 1 และตรวจสอบค่าความถี่ของรูปแบบสตริงย่อยทุกตัว โดยจะทำการเก็บข้อมูลเฉพาะรูปแบบสตริงย่อยที่มีค่าความถี่มากกว่า หรือเท่ากับ θ

| ก:3 | า:2 | ร:3 |
|---|---|---|
| .pos=1, 7, 10 | .pos=2, 11 | .pos=3, 5,12 |

นำ <ก:3> ออกจากโครงสร้างข้อมูลแบบ Min Heap พร้อมกับสืบค้นรูปแบบสตริงย่อยลูกของ <ก:3> โดยการใช้รายการตำแหน่งของ <ก:3> ซึ่งจะได้รูปแบบสตริงย่อย คือ <กา:2> และ <กอ:1> จากนั้นทำการปรับปรุงโครงสร้างข้อมูลแบบ Min Heap โดยทำการลบ <า:2> ออกจากโครงสร้างข้อมูลแบบ Min Heap เนื่องจาก <า:2> เป็น Substring ของ <กา:2> และมีค่าความถี่เท่ากับ ค่าความถี่ <า:2> - ค่าความถี่ <กา:2> < θ และทำการเก็บข้อมูลของ <กา:2> ลงโครงสร้างข้อมูลแบบ Min Heap โดยใช้กฎการแทรกข้อมูลของโครงสร้างข้อมูลแบบ Min Heap เนื่องจาก <กา:2> มีค่าความถี่เท่ากับ θ

| กา:2 | ร:3 | |
|---|---|---|
| .pos=2, 11 | .pos=3, 5,12 | |

นำ <กา:2> ออกจากโครงสร้างข้อมูลแบบ Min Heap พร้อมกับสืบค้นรูปแบบสตริงย่อยลูกของ <กา:2> โดยการใช้รายการตำแหน่งของ <กา:2> ซึ่งจะได้รูปแบบสตริงย่อย คือ <การ:2> จากนั้นทำการปรับปรุงโครงสร้างข้อมูลแบบ Min Heap โดยทำการลบ <ร:3> ออกจากโครงสร้างข้อมูลแบบ Min Heap เนื่องจาก <ร:3> เป็น Substring ของ <การ:2> และมีค่าความถี่เท่ากับ ค่าความถี่ <ร:3> - ค่าความถี่ <การ:2> < θ และทำการเก็บข้อมูลของ <การ:2> ลงโครงสร้างข้อมูลแบบ Min Heap โดยใช้กฎการแทรกข้อมูลของโครงสร้างข้อมูลแบบ Min Heap เนื่องจาก <การ:2> มีค่าความถี่เท่ากับ θ

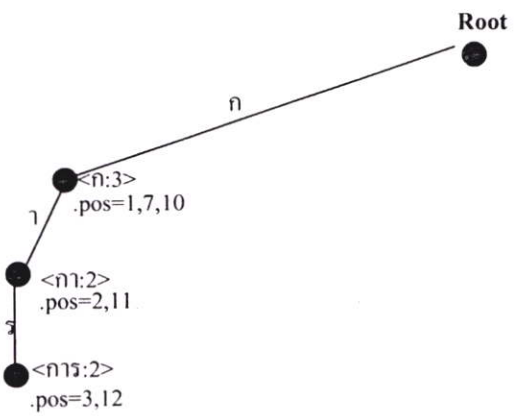
| การ:2<br>pos=3, 12 | |
|---|---|

นำ <การ:2> ออกจากโครงสร้างข้อมูลแบบ Min Heap พร้อมกับสืบค้นรูปแบบสตริงย่อยลูกของ <การ:2> โดยการใช้รายการตำแหน่งของ <การ:2> ซึ่งจะได้รูปแบบสตริงย่อย คือ <การป:1> และ <การ$:1> ซึ่งจะไม่ทำการเก็บข้อมูลลงโครงสร้างข้อมูลแบบ Min Heap เนื่องจาก <การป:1> และ <การ$:1> มีค่าความถี่น้อยกว่า $\theta$

| |
|---|
| |

เมื่อโครงสร้างข้อมูลแบบ Min Heap ว่าง จะแสดงถึงการเสร็จสิ้นการทำงานของอัลกอริทึมที่มีประสิทธิภาพ

จากตัวอย่างที่ 3.11 เมื่อพิจารณาขั้นตอนการทำงานของอัลกอริทึมที่มีประสิทธิภาพโดยใช้โครงสร้างข้อมูลแบบ Min Heap จะทำให้ได้โครงสร้าง FST ดังรูปที่ 3.3




**รูปที่ 3.3** แสดงโครงสร้าง FST โดยใช้ขั้นตอนการทำงานของอัลกอริทึมที่มีประสิทธิภาพ

จากรูปที่ 3.3 แสดงโครงสร้าง FST โดยใช้ขั้นตอนการทำงานของอัลกอริทึมที่มีประสิทธิภาพซึ่งจะได้ผลของ

$FSP_{T2}$ = {<ก : 5>, <ก, า : 3>, <ก, า, ร : 3>}

และ $FM_{T2}$ = {<ก, า, ร : 2>}

สำหรับอัลกอริทึมที่มีประสิทธิภาพนั้น จะใช้กฎการลดเส้นทาง 2 ข้อ ที่ใช้ในการลดจำนวนการแจงนับรูปแบบสตริงย่อย คือ 1. กฎการลดเส้นทางโดยการนำ $\theta$ มาเป็นตัวตรวจสอบการหยุดการสืบค้น และ 2. กฎการลดเส้นทางโดยใช้นิยามการเป็น Substring ต่อกัน เพื่อลดจำนวนการแจงนับรูปแบบสตริงย่อยบนสตริงเพื่อลดเวลาในการสืบค้น FM

# บทที่ 4
# การทดลองและผลการทดลอง

จากขั้นตอนวิธีการสืบค้นรูปแบบสตริงย่อยแบบความยาวสูงสุดที่เกิดบ่อยทั้ง 3 วิธี ได้แก่ 1. อัลกอริทึมพื้นฐาน 2. อัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว และ 3. อัลกอริทึมที่มีประสิทธิภาพ จะให้ประสิทธิภาพในการทำงานที่แตกต่างกันในด้านจำนวนการสร้างรูปแบบสตริงย่อย แต่ทั้ง 3 วิธี จะได้ผลลัพธ์ของ FM เท่ากัน แต่จะมีจำนวนการสร้างรูปแบบสตริงย่อยที่แตกต่างกัน เนื่องจากขั้นตอนการทำงานของแต่ละวิธีมีการทำงานที่แตกต่างกัน โดยวิธีของอัลกอริทึมพื้นฐานนั้น จะเป็นการสืบค้นสตริงย่อยทั้งหมดที่เป็นไปได้จากสตริงออกมา ดังนั้น อัลกอริทึมพื้นฐานจึงเป็นวิธีที่มีการสร้างรูปแบบสตริงย่อยมากที่สุด แต่จะให้ผลลัพธ์ FM ที่มีความถูกต้องมากที่สุด ส่วนวิธีของอัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว จะใช้กฎการลดเส้นทางโดยการนำค่า $\theta$ มาเป็นตัวตรวจสอบการหยุดการสืบค้น เพื่อลดจำนวนการสร้างรูปแบบสตริงย่อย แต่ยังคงให้ผลลัพธ์เท่ากับวิธีของอัลกอริทึมพื้นฐาน และวิธีของอัลกอริทึมที่มีประสิทธิภาพ จะใช้กฎการลดเส้นทางโดยการนำค่า $\theta$ มาเป็นตัวตรวจสอบการหยุดการสืบค้น และ กฎการลดเส้นทางโดยการใช้นิยามการเป็น Substring ต่อกัน เพื่อนำมาลดเส้นทางในการสร้างรูปแบบสตริงย่อย ซึ่งจะลดจำนวนการสร้างรูปแบบสตริงย่อยลงได้กว่าวิธีของอัลกอริทึมพื้นฐาน แต่จะให้ผลลัพธ์ FM เท่ากับวิธีของอัลกอริทึมพื้นฐาน ดังนั้นในบทนี้จึงนำเสนอการทดลองเพื่อเปรียบเทียบจำนวนรูปแบบสตริงย่อยที่ได้จากการสร้างของอัลกอริทึมทั้ง 3 วิธี

## 4.1 การทดลอง

### 4.1.1 ข้อมูลที่ใช้ในการทดสอบ

ข้อมูลที่ใช้ในการทดลองเป็นชุดข้อมูลสตริงที่อยู่บนเว็บไซต์ต่าง ๆ โดยข้อมูลสตริงแต่ละตัวจะมีลักษณะข้อมูลที่แตกต่างกันไป เพื่อใช้ในการทดสอบข้อมูลที่มีความหลากหลาย ซึ่งชุดข้อมูลสตริงที่ใช้ในการทดลองในงานวิจัยนี้ประกอบไปด้วยข้อมูลสตริง 10 ข้อมูล ซึ่งในตารางที่ 4.1 แสดงแหล่งที่มาของแต่ละข้อมูลสตริง และในตารางที่ 4.2 อธิบายลักษณะข้อมูลของแต่ละข้อมูลสตริง

**ตารางที่ 4.1** แสดงแหล่งที่มาของแต่ละข้อมูลสตริง

| ข้อมูลสตริง | แหล่งที่มาของข้อมูลสตริง |
|---|---|
| Alphabet | http://www.cosc.canterbury.ac.nz/corpus/descriptions/#artificl |
| Bible | http://www.cosc.canterbury.ac.nz/corpus/descriptions/#large |
| Binary_digit | http://www.littera.waseda.ac.jp/faculty/tyosem/data/2conjoint.txt |
| Field_C | http://www.cosc.canterbury.ac.nz/corpus/descriptions/#cantrbry |
| Genome | http://www.cosc.canterbury.ac.nz/corpus/descriptions/#large |
| Google_HTML | http://america.google.com/ |
| Pi | http://www.cosc.canterbury.ac.nz/corpus/descriptions/#misc |
| Protein | http://www.molecularevolution.org/resources/fileformats/pir_protein.php |
| Random | http://www.cosc.canterbury.ac.nz/corpus/descriptions/#artificl |
| การใช้อินเตอร์เน็ต | http://cc.spu.ac.th/index.php?cc=Y19maWxlPWFydGljbGVfdXNlX2ludGVybmV0 |

**ตารางที่ 4.2** อธิบายลักษณะข้อมูลของแต่ละข้อมูลสตริง

| ข้อมูลสตริง | อธิบายลักษณะข้อมูล |
|---|---|
| Alphabet | เป็นข้อมูลพยัญชนะภาษาอังกฤษ A-Z 2 ชุด |
| Bible | เป็นข้อมูลภาษาอังกฤษที่เป็นคัมภีร์ไบเบิ้ล |
| Binary_digit | เป็นข้อมูลตัวเลข 0 กับ 1 หรือ ตัวเลขไบนารี่ |
| Field_C | เป็นข้อมูลโคดภาษาซี |
| Genome | เป็นข้อมูลยีนส์ โครโมโซม |
| Google_HTML | เป็นข้อมูลโคดภาษา HTML ที่ใช้ในการเขียนโฮมเพจ |
| Pi | เป็นข้อมูลตัวเลข |
| Protein | เป็นข้อมูลโปรตีน |
| Random | เป็นข้อมูลที่ผสมกันระหว่างตัวเลข พยัญชนะภาษาอังกฤษ และสัญลักษณ์พิเศษ |
| การใช้อินเตอร์เน็ต | เป็นข้อมูลภาษาไทยเกี่ยวกับการใช้งานอินเตอร์เน็ต |

### 4.1.2 โปรแกรมที่ใช้ในการทดลอง

โปรแกรมที่ใช้ในการทดลองกับชุดข้อมูลสตริง ถูกพัฒนาขึ้นโดยใช้ Visual Basic Version 6.0 และฐานข้อมูล Microsoft Access 2003 โดยโปรแกรมที่ใช้ในการทดลองถูกพัฒนาขึ้นมา 3 โปรแกรม โดยใช้แทนการทำงานของ 3 อัลกอริทึม คือ อัลกอริทึมพื้นฐาน อัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว และอัลกอริทึมที่มีประสิทธิภาพ เพื่อใช้ตรวจสอบจำนวนการสร้างรูปแบบสตริงย่อยของแต่ละวิธี เพื่อนำมาเปรียบเทียบประสิทธิภาพทางด้านจำนวนการสร้างรูปแบบสตริงย่อย

### 4.1.3 เครื่องมือที่ใช้ในการวิจัย

เครื่องมือที่ใช้ในการทดสอบ มีคุณสมบัติดังต่อไปนี้

| | |
|---|---|
| หน่วยประมวลผลกลาง(CPU) | : Intel Pentium M processor 1.5 MHz |
| หน่วยความจำหลัก(RAM) | : 256 MB |
| หน่วยความจำสำรอง(Hard Disk) | : 30 GB |
| ระบบปฏิบัติการ(OS) | : Window XP Professional Version 2003 |
| โปรแกรมที่ใช้ในการพัฒนา | : Visual Basic Version 6.0 |
| ฐานข้อมูลที่ใช้ในการพัฒนา | : Microsoft Access 2003 |

### 4.1.4 การออกแบบการทดลองและเหตุผล

งานวิจัยนี้ได้ทำการทดลองเพื่อหาผลของจำนวนรูปแบบสตริงย่อยที่แต่ละอัลกอริทึมทำการสร้างออกมาได้ และทำการเปรียบเทียบอัตราการลดจำนวนการสร้างรูปแบบสตริงย่อย โดยคิดค่าเป็นเปอรเซ็นต์ ระหว่างอัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว และอัลกอริทึมที่มีประสิทธิภาพ โดยใช้อัลกอริทึมพื้นฐานเป็นฐานในการเปรียบเทียบ ซึ่งจะมีปัจจัยทางด้านการกำหนดค่าความถี่ขั้นต่ำ โดยจะมีการเพิ่มค่าขึ้นทีละหนึ่ง ซึ่งจะส่งผลกระทบต่ออัตราการเปลี่ยนแปลงของจำนวนการสร้างรูปแบบสตริงย่อย

การทดลองนี้จะทำงานกับข้อมูลสตริง 10 ข้อมูล ซึ่งมีลักษณะข้อมูลที่แตกต่างกัน และจะมีการกำหนดค่าความถี่ขั้นต่ำ โดยเริ่มจาก 2 ถึง 10 เพื่อนำผลมาวิเคราะห์ค่าความเปลี่ยนแปลงเมื่อค่าความถี่ขั้นต่ำมีการเพิ่มขึ้น

## 4.2 ผลการทดลอง

ผลการทดลองโดยการใช้วิธีการของอัลกอริทึมทั้ง 3 อัลกอริทึม จะมีการเก็บข้อมูลลงสู่ตาราง โดยจะทำการเก็บข้อมูล ดังต่อไปนี้

1. ข้อมูลสตริงนำเข้า
2. ความยาวของข้อมูลสตริงนำเข้า
3. จำนวนรูปแบบสตริงย่อยทั้งหมดที่ได้จากการแจงนับโดยการใช้อัลกอริทึมพื้นฐาน
4. จำนวนรูปแบบสตริงย่อยทั้งหมดที่ได้จากการแจงนับโดยการใช้อัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว
5. อัตราการลดลง(%) ในการแจงนับรูปแบบสตริงย่อยของอัลกอริทึมพื้นฐานที่ปรับปรุงแล้วจากอัลกอริทึมพื้นฐาน
6. จำนวนรูปแบบสตริงย่อยทั้งหมดที่ได้จากการแจงนับโดยการใช้อัลกอริทึมที่มีประสิทธิภาพ
7. อัตราการลดลง(%) ในการแจงนับรูปแบบสตริงย่อยของอัลกอริทึมที่มีประสิทธิภาพจากอัลกอริทึมพื้นฐาน
8. อัตราการลดลง(%) ในการแจงนับรูปแบบสตริงย่อยของอัลกอริทึมที่มีประสิทธิภาพจากอัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว
9. จำนวนรูปแบบสตริงย่อยแบบความยาวสูงสุดที่เกิดบ่อย(FM)
10. ความยาวของรูปแบบสตริงย่อยแบบความยาวสูงสุดที่เกิดบ่อยที่ยาวที่สุด
11. ความยาวของรูปแบบสตริงย่อยแบบความยาวสูงสุดที่เกิดบ่อยที่สั้นที่สุด
12. ความยาวของรูปแบบสตริงย่อยแบบความยาวสูงสุดที่เกิดบ่อยโดยเฉลี่ย

การทดลองเพื่อวิเคราะห์ค่าความเปลี่ยนแปลงของจำนวนการสร้างรูปแบบสตริงย่อย โดยมีปัจจัยทางด้านการกำหนดค่าความถี่ขั้นต่ำที่มีการเพิ่มค่าขึ้นทีละหนึ่ง ซึ่งเป็นตัวส่งผลกระทบต่อการเปลี่ยนแปลงของจำนวนการสร้างรูปแบบสตริงย่อย ดังนั้น จึงทำการทดลองชุดข้อมูลสตริงโดยมีการกำหนดค่าความถี่ขั้นต่ำตั้งแต่ 2 ถึง 10 ซึ่งผลการทดลองแสดงในตาราง ดังต่อไปนี้

ตารางที่ 4.3 แสดงผลการทดลองกับชุดข้อมูลสตริง โดยมีการกำหนดค่าความถี่ขั้นต่ำเท่ากับ 2
ตารางที่ 4.4 แสดงผลการทดลองกับชุดข้อมูลสตริง โดยมีการกำหนดค่าความถี่ขั้นต่ำเท่ากับ 3
ตารางที่ 4.5 แสดงผลการทดลองกับชุดข้อมูลสตริง โดยมีการกำหนดค่าความถี่ขั้นต่ำเท่ากับ 4
ตารางที่ 4.6 แสดงผลการทดลองกับชุดข้อมูลสตริง โดยมีการกำหนดค่าความถี่ขั้นต่ำเท่ากับ 5

**ตารางที่ 4.3** แสดงผลการทดลองกับชุดข้อมูลสตริง โดยมีการกำหนดค่าความถี่ขั้นต่ำเท่ากับ 2

**ค่าความถี่ขั้นต่ำ = 2**

| ข้อมูลสตริง | ความยาวสตริง(n) | อัลกอริทึมพื้นฐาน(NA) | อัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว(INA) | เทียบ INAกับNA อัตราการลด(%) | อัลกอริทึมที่มีประสิทธิภาพ(EA) | เทียบ EAกับNA อัตราการลด(%) | เทียบ EAกับINA อัตราการลด(%) | จำนวน FM | ความยาวสตริงย่อยผลลัพธ์ ยาวสุด | สั้นสุด | เฉลี่ย |
|---|---|---|---|---|---|---|---|---|---|---|---|
| alphabet | 52 | 1027 | 52 | 97.46835443 | 26 | 97.46835443 | 0 | 1 | 26 | 26 | 26 |
| bible | 3822 | 7270785 | 12014 | 99.83476337 | 4798 | 99.93400988 | 60.06325953 | 65 | 45 | 2 | 11.23 |
| binary_digit | 1774 | 1552450 | 3982 | 99.74350221 | 1731 | 99.88849882 | 56.52938222 | 149 | 32 | 6 | 15.87 |
| fields_C | 2865 | 2626196 | 5272 | 99.79925337 | 2208 | 99.91592402 | 58.11836115 | 58 | 28 | 2 | 10.62 |
| genome | 9216 | 42416643 | 7047 | 99.98338624 | 5937 | 99.98600314 | 15.75138357 | 1905 | 13 | 5 | 7.24 |
| google_HTML | 4119 | 5696138 | 6034 | 99.89406858 | 2766 | 99.95144078 | 54.15976135 | 231 | 31 | 2 | 7.28 |
| pi | 4025 | 8090566 | 1741 | 99.97848111 | 1243 | 99.98463643 | 28.60425043 | 862 | 6 | 3 | 3.74 |
| protein | 1996 | 1975119 | 8432 | 99.57308901 | 2966 | 99.84983183 | 64.82447818 | 86 | 59 | 2 | 10.5 |
| random | 3137 | 4916771 | 1024 | 99.97917332 | 548 | 99.98885447 | 46.484375 | 768 | 4 | 2 | 2.16 |
| การใช้อินเตอร์เน็ต | 2399 | 2869823 | 3002 | 99.89539425 | 1243 | 99.95668722 | 58.59427049 | 166 | 27 | 1 | 4.62 |
| ค่าเฉลี่ยอัตราการลด(%) | | | | 99.61494659 | | 99.6924241 | 44.31295219 | | | | |

**ตารางที่ 4.4** แสดงผลการทดลองกับชุดข้อมูลสตริง โดยมีการกำหนดค่าความถี่ขั้นต่ำเท่ากับ 3

**ค่าความถี่ขั้นต่ำ = 3**

| | | | | เทียบ INAกับNA | | เทียบ EAกับNA | เทียบ EAกับINA | | ความยาวสตริงย่อยผลลัพธ์ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ข้อมูลสตริง | ความยาวสตริง(n) | อัลกอริทึมพื้นฐาน(NA) | อัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว(INA) | อัตราการลด(%) | อัลกอริทึมที่มีประสิทธิภาพ(EA) | อัตราการลด(%) | อัตราการลด(%) | จำนวน FM | ยาวสุด | สั้นสุด | เฉลี่ย |
| alphabet | 52 | 1027 | 0 | 100 | 0 | 100 | 0 | 0 | 0 | 0 | 0 |
| bible | 3822 | 7270785 | 5345 | 99.92648662 | 2128 | 99.97073218 | 60.18709074 | 73 | 44 | 2 | 6.93 |
| binary_digit | 1774 | 1552450 | 2124 | 99.863184 | 918 | 99.94086766 | 56.77966102 | 120 | 27 | 6 | 13.64 |
| fields_C | 2865 | 2626196 | 2153 | 99.9180183 | 1047 | 99.96013245 | 51.37018114 | 65 | 22 | 1 | 7.07 |
| genome | 9216 | 42416643 | 3531 | 99.99167544 | 3281 | 99.99226483 | 7.080147267 | 1401 | 10 | 5 | 6.39 |
| google_HTML | 4119 | 5696138 | 2996 | 99.94740296 | 1414 | 99.97517616 | 52.80373832 | 172 | 29 | 2 | 4.97 |
| pi | 4025 | 8090566 | 949 | 99.98827029 | 947 | 99.98829501 | 0.210748156 | 722 | 5 | 3 | 3.09 |
| protein | 1996 | 1975119 | 3301 | 99.83287083 | 1177 | 99.94040865 | 64.34413814 | 72 | 41 | 1 | 6.93 |
| random | 3137 | 4916771 | 590 | 99.98800025 | 576 | 99.98828499 | 2.372881356 | 538 | 3 | 2 | 2 |
| การใช้อินเตอร์เน็ต | 2399 | 2869823 | 1413 | 99.95076351 | 904 | 99.9684998 | 36.02264685 | 123 | 21 | 1 | 3.6 |
| ค่าเฉลี่ยอัตราการลด(%) | | | | 99.94066722 | | 99.97246617 | 33.1171233 | | | | |

**ตารางที่ 4.5** แสดงผลการทดลองกับชุดข้อมูลสตริง โดยมีการกำหนดค่าความถี่ขั้นต่ำเท่ากับ 4

**ค่าความถี่ขั้นต่ำ = 4**

| | | | | เทียบ INAกับNA | | เทียบ EAกับNA | เทียบ EAกับINA | | ความยาวสตริงย่อยผลลัพธ์ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ข้อมูลสตริง | ความยาวสตริง(n) | อัลกอริทึมพื้นฐาน(NA) | อัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว(INA) | อัตราการลด(%) | อัลกอริทึมที่มีประสิทธิภาพ(EA) | อัตราการลด(%) | อัตราการลด(%) | จำนวน FM | ยาวสุด | สั้นสุด | เฉลี่ย |
| alphabet | 52 | 1027 | 0 | 100 | 0 | 100 | 0 | 0 | 0 | 0 | 0 |
| bible | 3822 | 7270785 | 3080 | 99.95763869 | 1321 | 99.9818314 | 57.11038961 | 72 | 43 | 2 | 5.44 |
| binary_digit | 1774 | 1552450 | 1378 | 99.91123708 | 629 | 99.9594834 | 54.35413643 | 91 | 24 | 5 | 12.49 |
| fields_C | 2865 | 2626196 | 1341 | 99.94893755 | 682 | 99.97403088 | 49.14243102 | 58 | 22 | 1 | 5.39 |
| genome | 9216 | 42416643 | 2307 | 99.9945611 | 2237 | 99.99472613 | 3.034243606 | 1043 | 8 | 4 | 5.96 |
| google_HTML | 4119 | 5696138 | 1916 | 99.96636317 | 1004 | 99.98237402 | 47.59916493 | 144 | 29 | 2 | 4.54 |
| pi | 4025 | 8090566 | 689 | 99.99148391 | 653 | 99.99192887 | 5.224963716 | 559 | 4 | 3 | 3.01 |
| protein | 1996 | 1975119 | 1843 | 99.90668917 | 691 | 99.96501477 | 62.50678242 | 63 | 27 | 1 | 6.47 |
| random | 3137 | 4916771 | 331 | 99.99326794 | 324 | 99.99341031 | 2.114803625 | 293 | 2 | 1 | 1.99 |
| การใช้อินเตอร์เน็ต | 2399 | 2869823 | 885 | 99.96916186 | 762 | 99.97344784 | 13.89830508 | 94 | 20 | 1 | 3.43 |
| ค่าเฉลี่ยอัตราการลด(%) | | | | 99.96393405 | | 99.98162476 | 29.49852204 | | | | |

**ตารางที่ 4.6** แสดงผลการทดลองกับชุดข้อมูลสตริง โดยมีการกำหนดค่าความถี่ขั้นต่ำเท่ากับ 5

**ค่าความถี่ขั้นต่ำ = 5**

| | | | | เทียบ INAกับNA | | เทียบ EAกับNA | เทียบ EAกับINA | | ความยาวสตริงย่อยผลลัพธ์ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ข้อมูลสตริง | ความยาวสตริง(n) | อัลกอริทึมพื้นฐาน(NA) | อัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว(INA) | อัตราการลด(%) | อัลกอริทึมที่มีประสิทธิภาพ(EA) | อัตราการลด(%) | อัตราการลด(%) | จำนวน FM | ยาวสุด | สั้นสุด | เฉลี่ย |
| alphabet | 52 | 1027 | 0 | 100 | 0 | 100 | 0 | 0 | 0 | 0 | 0 |
| bible | 3822 | 7270785 | 1772 | 99.97562849 | 880 | 99.98789677 | 50.33860045 | 60 | 27 | 2 | 5.23 |
| binary_digit | 1774 | 1552450 | 1052 | 99.93223614 | 493 | 99.96824374 | 53.13688213 | 81 | 20 | 5 | 11.53 |
| fields_C | 2865 | 2626196 | 750 | 99.97144158 | 470 | 99.98210339 | 37.33333333 | 55 | 11 | 1 | 4.78 |
| genome | 9216 | 42416643 | 1724 | 99.99593556 | 1696 | 99.99600157 | 1.62412993 | 818 | 8 | 4 | 5.68 |
| google_HTML | 4119 | 5696138 | 1378 | 99.97580817 | 743 | 99.98695607 | 46.08127721 | 137 | 29 | 2 | 3.72 |
| pi | 4025 | 8090566 | 480 | 99.99406716 | 470 | 99.99419076 | 2.083333333 | 370 | 3 | 3 | 3 |
| protein | 1996 | 1975119 | 779 | 99.96055934 | 431 | 99.97817853 | 44.67265725 | 72 | 15 | 1 | 4.37 |
| random | 3137 | 4916771 | 185 | 99.99623737 | 175 | 99.99644075 | 5.405405405 | 151 | 2 | 1 | 1.97 |
| การใช้อินเตอร์เน็ต | 2399 | 2869823 | 526 | 99.98167134 | 343 | 99.98804804 | 34.79087452 | 82 | 12 | 1 | 3.07 |
| ค่าเฉลี่ยอัตราการลด(%) | | | | 99.97835852 | | 99.98780596 | 27.54664936 | | | | |

**ตารางที่ 4.7** แสดงผลการทดลองกับชุดข้อมูลสตริง โดยมีการกำหนดค่าความถี่ขั้นต่ำเท่ากับ 6

**ค่าความถี่ขั้นต่ำ = 6**

| | | | | เทียบ INAกับNA | | เทียบ EAกับNA | เทียบ EAกับINA | | ความยาวสตริงย่อยผลลัพธ์ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ข้อมูลสตริง | ความยาวสตริง(n) | อัลกอริทึมพื้นฐาน(NA) | อัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว(INA) | อัตราการลด(%) | อัลกอริทึมที่มีประสิทธิภาพ(EA) | อัตราการลด(%) | อัตราการลด(%) | จำนวน FM | ยาวสุด | สั้นสุด | เฉลี่ย |
| alphabet | 52 | 1027 | 0 | 100 | 0 | 100 | 0 | 0 | 0 | 0 | 0 |
| bible | 3822 | 7270785 | 1191 | 99.98361938 | 594 | 99.99183032 | 50.12594458 | 62 | 23 | 2 | 4.56 |
| binary_digit | 1774 | 1552450 | 817 | 99.94737351 | 383 | 99.97532932 | 53.12117503 | 68 | 19 | 5 | 10.57 |
| fields_C | 2865 | 2626196 | 592 | 99.97745789 | 374 | 99.98575887 | 36.82432432 | 54 | 11 | 1 | 4.11 |
| genome | 9216 | 42416643 | 1375 | 99.99675835 | 1364 | 99.99678428 | 0.8 | 703 | 7 | 4 | 5.44 |
| google_HTML | 4119 | 5696138 | 695 | 99.98779875 | 477 | 99.99162591 | 31.36690647 | 122 | 13 | 2 | 3.22 |
| pi | 4025 | 8090566 | 322 | 99.99602006 | 314 | 99.99611894 | 2.48447205 | 213 | 3 | 2 | 2.99 |
| protein | 1996 | 1975119 | 529 | 99.9732168 | 335 | 99.983039 | 36.67296786 | 80 | 15 | 1 | 3.42 |
| random | 3137 | 4916771 | 98 | 99.99800682 | 94 | 99.99808818 | 4.081632653 | 69 | 2 | 1 | 1.86 |
| การใช้อินเตอร์เน็ต | 2399 | 2869823 | 380 | 99.98675877 | 272 | 99.99052206 | 28.42105263 | 74 | 12 | 1 | 2.72 |
| ค่าเฉลี่ยอัตราการลด(%) | | | | 99.98470103 | | 99.99090969 | 24.38984756 | | | | |

**ตารางที่ 4.8** แสดงผลการทดลองกับชุดข้อมูลสตริง โดยมีการกำหนดค่าความถี่ขั้นต่ำเท่ากับ 7

**ค่าความถี่ขั้นต่ำ = 7**

| | | | | เทียบ INAกับNA | | เทียบ EAกับNA | เทียบ EAกับINA | | ความยาวสตริงย่อยผลลัพธ์ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ข้อมูลสตริง | ความยาวสตริง(n) | อัลกอริทึมพื้นฐาน(NA) | อัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว(INA) | อัตราการลด(%) | อัลกอริทึมที่มีประสิทธิภาพ(EA) | อัตราการลด(%) | อัตราการลด(%) | จำนวน FM | ยาวสุด | สั้นสุด | เฉลี่ย |
| alphabet | 52 | 1027 | 0 | 100 | 0 | 100 | 0 | 0 | 0 | 0 | 0 |
| bible | 3822 | 7270785 | 969 | 99.98667269 | 501 | 99.99310941 | 48.29721362 | 50 | 23 | 2 | 4.7 |
| binary_digit | 1774 | 1552450 | 664 | 99.9572289 | 320 | 99.97938742 | 51.80722892 | 58 | 19 | 5 | 10.18 |
| fields_C | 2865 | 2626196 | 442 | 99.98316957 | 289 | 99.98899549 | 34.61538462 | 62 | 10 | 1 | 3.49 |
| genome | 9216 | 42416643 | 1175 | 99.99722986 | 1165 | 99.99725344 | 0.85106383 | 634 | 7 | 4 | 5.28 |
| google_HTML | 4119 | 5696138 | 476 | 99.99164346 | 340 | 99.99403104 | 28.57142857 | 135 | 12 | 1 | 2.51 |
| pi | 4025 | 8090566 | 222 | 99.99725606 | 187 | 99.99768867 | 15.76576577 | 126 | 3 | 2 | 2.88 |
| protein | 1996 | 1975119 | 357 | 99.98192514 | 258 | 99.9869375 | 27.73109244 | 73 | 10 | 2 | 3 |
| random | 3137 | 4916771 | 62 | 99.99873901 | 54 | 99.99890172 | 12.90322581 | 38 | 2 | 1 | 1.63 |
| การใช้อินเตอร์เน็ต | 2399 | 2869823 | 310 | 99.98919794 | 243 | 99.99153258 | 21.61290323 | 73 | 12 | 1 | 2.45 |
| ค่าเฉลี่ยอัตราการลด(%) | | | | 99.98830626 | | 99.99278373 | 24.21553068 | | | | |

**ตารางที่ 4.9** แสดงผลการทดลองกับชุดข้อมูลสตริง โดยมีการกำหนดค่าความถี่ขั้นต่ำเท่ากับ 8

**ค่าความถี่ขั้นต่ำ = 8**

| | | | | เทียบ INAกับNA | | เทียบ EAกับNA | เทียบ EAกับINA | | ความยาวสตริงย่อยผลลัพธ์ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ข้อมูลสตริง | ความยาวสตริง(n) | อัลกอริทึมพื้นฐาน(NA) | อัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว(INA) | อัตราการลด(%) | อัลกอริทึมที่มีประสิทธิภาพ(EA) | อัตราการลด(%) | อัตราการลด(%) | จำนวน FM | ยาวสุด | สั้นสุด | เฉลี่ย |
| alphabet | 52 | 1027 | 0 | 100 | 0 | 100 | 0 | 0 | 0 | 0 | 0 |
| bible | 3822 | 7270785 | 790 | 99.9891346 | 444 | 99.99389337 | 43.79746835 | 42 | 19 | 2 | 4.92 |
| binary_digit | 1774 | 1552450 | 527 | 99.96605366 | 274 | 99.98235048 | 48.00759013 | 50 | 18 | 4 | 9.64 |
| fields_C | 2865 | 2626196 | 371 | 99.9858731 | 261 | 99.99006167 | 29.64959569 | 59 | 9 | 1 | 3.54 |
| genome | 9216 | 42416643 | 999 | 99.99764479 | 994 | 99.99765658 | 0.500500501 | 538 | 7 | 4 | 5.17 |
| google_HTML | 4119 | 5696138 | 417 | 99.99267925 | 291 | 99.99489128 | 30.21582734 | 111 | 12 | 1 | 2.49 |
| pi | 4025 | 8090566 | 153 | 99.99810891 | 139 | 99.99828195 | 9.150326797 | 85 | 3 | 2 | 2.5 |
| protein | 1996 | 1975119 | 246 | 99.98754505 | 196 | 99.99007655 | 20.32520325 | 72 | 9 | 1 | 2.68 |
| random | 3137 | 4916771 | 47 | 99.99904409 | 44 | 99.9991051 | 6.382978723 | 36 | 2 | 1 | 1.25 |
| การใช้อินเตอร์เน็ต | 2399 | 2869823 | 246 | 99.99142804 | 187 | 99.99348392 | 23.98373984 | 63 | 12 | 1 | 2.2 |
| ค่าเฉลี่ยอัตราการลด(%) | | | | 99.99075115 | | 99.99398009 | 21.20132306 | | | | |

**ตารางที่ 4.10** แสดงผลการทดลองกับชุดข้อมูลสตริง โดยมีการกำหนดค่าความถี่ขั้นต่ำเท่ากับ 9

**ค่าความถี่ขั้นต่ำ = 9**

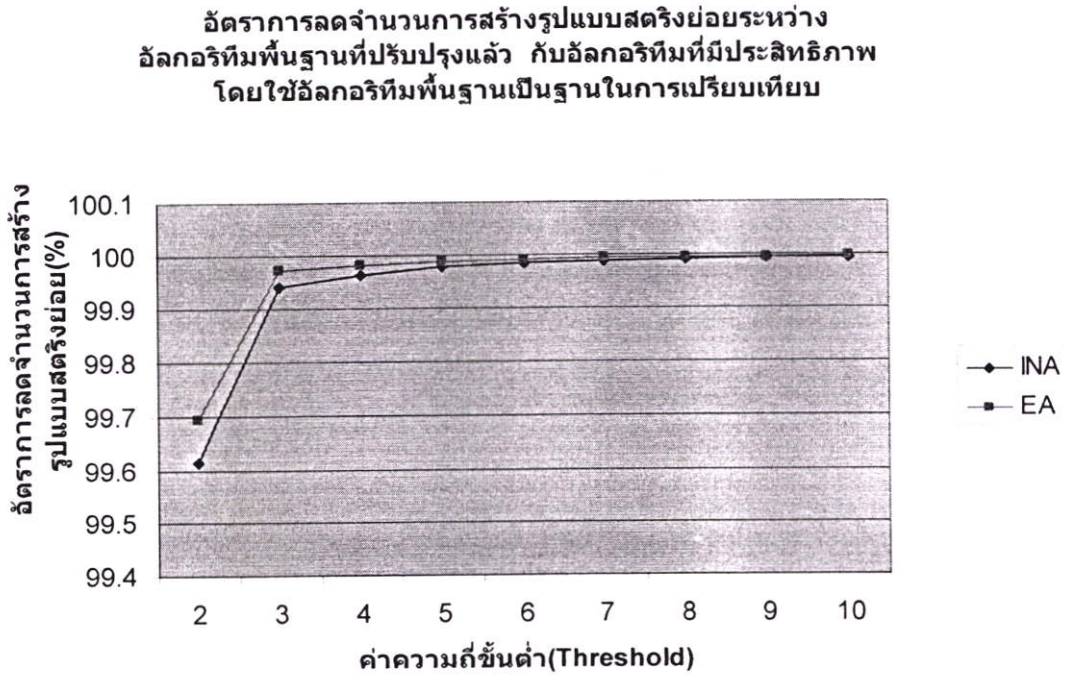
| | | | | เทียบ INAกับNA | | เทียบ EAกับNA | เทียบ EAกับINA | | ความยาวสตริงย่อยผลลัพธ์ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ข้อมูลสตริง | ความยาวสตริง(n) | อัลกอริทึมพื้นฐาน(NA) | อัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว(INA) | อัตราการลด(%) | อัลกอริทึมที่มีประสิทธิภาพ(EA) | อัตราการลด(%) | อัตราการลด(%) | จำนวน FM | ยาวสุด | สั้นสุด | เฉลี่ย |
| alphabet | 52 | 1027 | 0 | 100 | 0 | 100 | 0 | 0 | 0 | 0 | 0 |
| bible | 3822 | 7270785 | 566 | 99.99221542 | 348 | 99.99521372 | 38.51590106 | 40 | 15 | 2 | 4.17 |
| binary_digit | 1774 | 1552450 | 481 | 99.96901672 | 272 | 99.98247931 | 43.45114345 | 47 | 17 | 4 | 9.68 |
| fields_C | 2865 | 2626196 | 297 | 99.98869087 | 223 | 99.99150863 | 24.91582492 | 53 | 8 | 1 | 3.09 |
| genome | 9216 | 42416643 | 874 | 99.99793949 | 872 | 99.9979442 | 0.228832952 | 474 | 7 | 4 | 5.09 |
| google_HTML | 4119 | 5696138 | 335 | 99.99411882 | 238 | 99.99582173 | 28.95522388 | 100 | 12 | 1 | 2.43 |
| pi | 4025 | 8090566 | 130 | 99.99839319 | 113 | 99.99860331 | 13.07692308 | 88 | 3 | 2 | 2.22 |
| protein | 1996 | 1975119 | 122 | 99.99382316 | 117 | 99.99407631 | 4.098360656 | 80 | 6 | 2 | 2.16 |
| random | 3137 | 4916771 | 44 | 99.9991051 | 42 | 99.99914578 | 4.545454545 | 37 | 2 | 1 | 1.16 |
| การใช้อินเตอร์เน็ต | 2399 | 2869823 | 139 | 99.9951565 | 86 | 99.9970033 | 38.1294964 | 67 | 6 | 1 | 1.88 |
| ค่าเฉลี่ยอัตราการลด(%) | | | | 99.99284593 | | 99.99517963 | 19.59171609 | | | | |

**ตารางที่ 4.11** แสดงผลการทดลองกับชุดข้อมูลสตริง โดยมีการกำหนดค่าความถี่ขั้นต่ำเท่ากับ 10

**ค่าความถี่ขั้นต่ำ = 10**

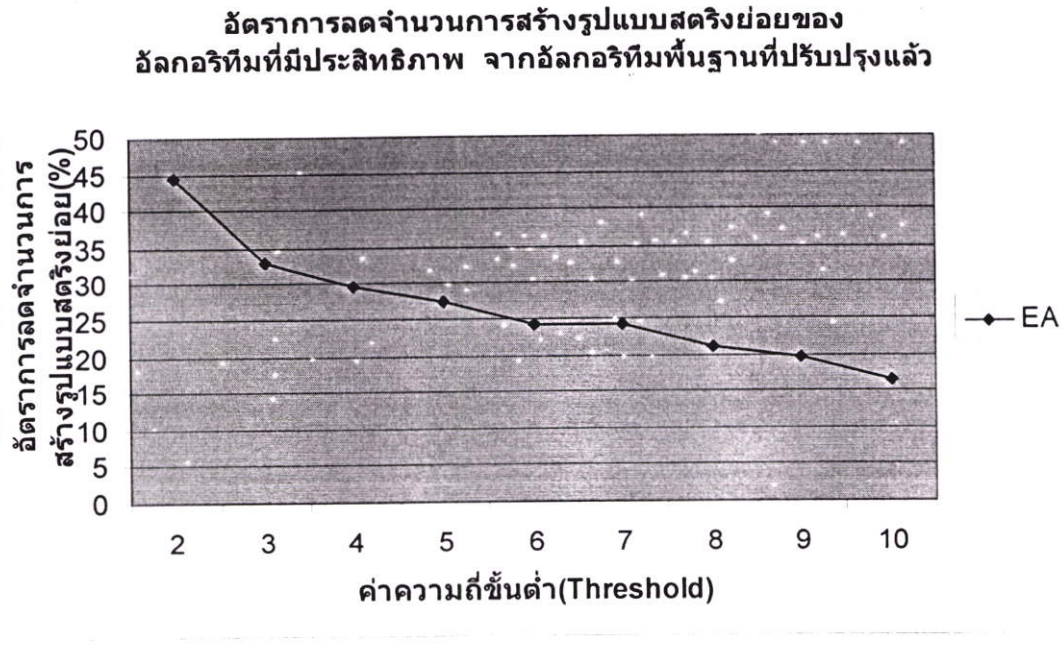
| | | | | เทียบ INAกับNA | | เทียบ EAกับNA | เทียบ EAกับINA | | ความยาวสตริงย่อยผลลัพธ์ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ข้อมูลสตริง | ความยาวสตริง(n) | อัลกอริทึมพื้นฐาน(NA) | อัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว(INA) | อัตราการลด(%) | อัลกอริทึมที่มีประสิทธิภาพ(EA) | อัตราการลด(%) | อัตราการลด(%) | จำนวน FM | ยาวสุด | สั้นสุด | เฉลี่ย |
| alphabet | 52 | 1027 | 0 | 100 | 0 | 100 | 0 | 0 | 0 | 0 | 0 |
| bible | 3822 | 7270785 | 507 | 99.99302689 | 320 | 99.99559882 | 36.88362919 | 34 | 13 | 2 | 4.38 |
| binary_digit | 1774 | 1552450 | 421 | 99.97288157 | 227 | 99.98537795 | 46.0807601 | 43 | 17 | 4 | 9.46 |
| fields_C | 2865 | 2626196 | 281 | 99.98930011 | 208 | 99.9920798 | 25.97864769 | 46 | 8 | 1 | 3.17 |
| genome | 9216 | 42416643 | 783 | 99.99815403 | 780 | 99.9981611 | 0.383141762 | 430 | 7 | 4 | 4.99 |
| google_HTML | 4119 | 5696138 | 243 | 99.99573395 | 208 | 99.9963484 | 14.40329218 | 98 | 8 | 1 | 2.29 |
| pi | 4025 | 8090566 | 119 | 99.99852915 | 117 | 99.99855387 | 1.680672269 | 91 | 3 | 2 | 2.09 |
| protein | 1996 | 1975119 | 99 | 99.99498764 | 94 | 99.99524079 | 5.050505051 | 66 | 6 | 2 | 2.12 |
| random | 3137 | 4916771 | 40 | 99.99918646 | 39 | 99.9992068 | 2.5 | 37 | 2 | 1 | 1.05 |
| การใช้อินเตอร์เน็ต | 2399 | 2869823 | 116 | 99.99595794 | 78 | 99.99728206 | 32.75862069 | 60 | 6 | 1 | 1.66 |
| ค่าเฉลี่ยอัตราการลด(%) | | | | 99.99377577 | | 99.99578496 | 16.57192689 | | | | |

เมื่อทำการทดลอง และทำการเปรียบเทียบอัตราการลดจำนวนการสร้างรูปแบบสตริงย่อยระหว่างอัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว กับอัลกอริทึมที่มีประสิทธิภาพ โดยใช้อัลกอริทึมพื้นฐานเป็นฐานในการเปรียบเทียบ และเปรียบเทียบอัตราการลดจำนวนการสร้างรูปแบบสตริงย่อยของอัลกอริทึมที่มีประสิทธิภาพ กับอัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว จะแสดงให้เห็นว่าอัตราการลดลงของจำนวนการสร้างรูปแบบสตริงย่อยมีค่าเพิ่มขึ้นตามการเปลี่ยนแปลงของค่าความถี่ขั้นต่ำที่มีการเพิ่มค่าทีละหนึ่ง ดังแสดงในรูปต่อไปนี้




**รูปที่ 4.1** อัตราการลดจำนวนการสร้างรูปแบบสตริงย่อยระหว่างอัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว กับอัลกอริทึมที่มีประสิทธิภาพ โดยใช้อัลกอริทึมพื้นฐานเป็นฐานในการเปรียบเทียบ

ข้อสังเกต จากรูปที่ 4.1 เมื่อทำการเปรียบเทียบอัตราการลดลงของจำนวนการสร้างรูปแบบสตริงย่อยระหว่างอัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว กับอัลกอริทึมที่มีประสิทธิภาพ โดยใช้อัลกอริทึมพื้นฐานเป็นฐานในการเปรียบเทียบ จะพบว่าอัลกอริทึมที่มีประสิทธิภาพและอัลกอริทึมพื้นฐานที่ปรับปรุงแล้วจะมีการเปลี่ยนแปลงของอัตราการลดลงของจำนวนการสร้างรูปแบบสตริงย่อยที่แตกต่างกันแต่จะมีแนวโน้มของอัตราการลดลงของจำนวนการสร้างรูปแบบสตริงย่อยใกล้เคียงกันเมื่อค่าความถี่ขั้นต่ำมีการเพิ่มขึ้น




**รูปที่ 4.2** อัตราการลดจำนวนการสร้างรูปแบบสตริงย่อยของอัลกอริทึมที่มีประสิทธิภาพจากอัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว

### 4.2.1 สรุปผลการทดลอง

จากการทดลอง พบว่าอัลกอริทึมที่มีประสิทธิภาพจะมีอัตราการลดลงของจำนวนการสร้างรูปแบบสตริงย่อยสูงกว่าอัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว แต่จะมีแนวโน้มในการลดลงของจำนวนการสร้างรูปแบบสตริงย่อยใกล้เคียงกันเมื่อค่าความถี่ขั้นต่ำมีการเพิ่มขึ้น ดังนั้น เราสามารถสรุปผลการทดลองได้ว่า อัตราการลดจำนวนการสร้างรูปแบบสตริงย่อยของอัลกอริทึมที่มีประสิทธิภาพ และอัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว อยู่ในเกณฑ์สูง เมื่อเปรียบเทียบกับจำนวนการสร้างรูปแบบสตริงย่อยโดยการใช้อัลกอริทึมพื้นฐาน ซึ่งทั้ง 3 อัลกอริทึมนี้จะให้คำตอบของผลลัพธ์ FM เท่ากัน อย่างไรก็ตาม อัลกอริทึมที่มีประสิทธิก็ยังคงทำการแจงนับรูปแบบสตริงย่อยบางตัวที่ไม่ได้เป็นสมาชิกของ FM อยู่

# บทที่ 5
# สรุปและข้อเสนอแนะ

## 5.1 สรุป

งานวิจัยนี้มีวัตถุประสงค์เพื่อพัฒนาอัลกอริทึมในการสืบค้น FM โดยใช้โครงสร้างซัฟฟิกซ์ทรัยแบบใหม่ที่เรียกว่า โครงสร้าง FST ซึ่งมีคุณสมบัติที่จะเป็นประโยชน์ต่อการลดจำนวนการแจงนับคำตอบที่เป็นไปได้ เพื่อพัฒนาวิธีการสืบค้น FM ที่ถูกต้องและครบถ้วน โดยทำการสร้างอัลกอริทึมขึ้นมา 3 แบบ คือ 1. อัลกอริทึมพื้นฐาน ซึ่งเป็นวิธีการสืบค้น FM บนโครงสร้าง FST 2. อัลกอริทึมพื้นฐานที่ปรับปรุงแล้ว ซึ่งเป็นวิธีการสืบค้น FM บนโครงสร้าง FST โดยใช้กฎการลดเส้นทางโดยการนำ $\theta$ มาเป็นตัวตรวจสอบการหยุดการสืบค้น และ 3. อัลกอริทึมที่มีประสิทธิภาพ ซึ่งเป็นวิธีการสืบค้น FM บนโครงสร้าง FST โดยใช้กฎการลดเส้นทางโดยการนำ $\theta$ มาเป็นตัวตรวจสอบการหยุดการสืบค้น และกฎการลดเส้นทางโดยการใช้นิยามการเป็น Substring ต่อกัน โดยทั้ง 3 อัลกอริทึมนี้ จะให้ผลลัพธ์ของรูปแบบสตริงย่อยคำตอบที่เป็นไปได้เพื่อการสืบค้น FM ซึ่งจะมีจำนวนการสร้างรูปแบบสตริงย่อยที่แตกต่างกัน โดยทำการเปรียบเทียบประสิทธิภาพโดยการนับจำนวนการสร้างรูปแบบสตริงย่อยของแต่ละอัลกอริทึม จากการทดลองจะพบว่าอัลกอริทึมที่มีประสิทธิภาพจะมีการสร้างจำนวนรูปแบบสตริงย่อยคำตอบที่เป็นไปได้น้อยที่สุด ซึ่งเป็นตัวบ่งชี้ถึงประสิทธิภาพทางด้านเวลาในการสืบค้น FM เมื่อเทียบกับอัลกอริทึมพื้นฐาน แต่อัลกอริทึมที่มีประสิทธิภาพก็ยังคงมีปัญหาทางด้านการแจงนับรูปแบบสตริงย่อยทั้งที่เป็น FM และไม่ได้เป็น FM บนโครงสร้าง FST อยู่

## 5.2 ข้อเสนอแนะ

เนื่องจากอัลกอริทึมที่มีประสิทธิภาพยังคงมีการแจงนับรูปแบบสตริงย่อยคำตอบที่เป็นไปได้ที่ไม่ใช่ FM ดังนั้น แนวทางการพัฒนาจึงควรคิดค้นอัลกอริทึมใหม่ที่สามารถแจงนับเฉพาะรูปแบบสตริงย่อยที่เป็น FM บนโครงสร้าง FST โดยที่โครงสร้าง FST อาจจะมีคุณสมบัติบางประการที่จะสามารถพัฒนาวิธีการสืบค้น FM ให้มีประสิทธิภาพมากขึ้น หรือพัฒนาโครงสร้างข้อมูลใหม่ที่สนับสนุนการสืบค้น FM ซึ่งมีประสิทธิภาพมากกว่าโครงสร้าง FST

# เอกสารอ้างอิง


[1] Giegerich, R.; Kurtz, S.; and Stoye, J. 1999. **Efficient implementation of lazy suffix trees.** In *Proceedings of the Third Workshop on Algorithmic Engineering (WAE99)*, volume 1668, 30–42. Springer Verlag.

[2] Gusfield, D. 1997. **Algorithms on Strings, Trees, and Sequences**: *Computer Science and Computational Biology*. Cambridge: Cambridge University Press.

[3] Jaak Vilo. **Discovering Frequent Patterns from Strings**: Department of Computer Science. University of Helsinki, Finland.

[4] Marti Hearst & Fredrik Wallenberg. Fall 2002. **Foundation of design**. Text Processing, Tries, and Dynamic Programming. University of California at Berkeley.

[5] McCreight, E. M. 1976. **A space–economical suffix tree construction algorithm.** *Journal of the ACM* 23:262–272.

[6] McGill University. **DATA STRUCTURES AND ALGORITHMS Topic #7: TRIES AND SUFFIX TREES.** [Online]. Available : http://www.cs.mcgill.ca/~cs251/ OldCourses/1997/topic7/. 1997.

[7] Mikio Yamamoto and Kenneth W. Church. 2001. **Using suffix arrays to compute term frequency and document frequency for all substrings in a corpus.** *Computational Linguistics*, 27(1):1–30.

[8] Morrison, D. R. 1968. **PATRICIA - practical algorithm to retrieve information coded in alphanumeric.** *Jrnl. A.C.M.* 15(4):514–534.

[9] Udi Manber and Gene Myers. **Suffix arrays: a new method for on-line string searches.** In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, Society for Industrial and Applied Mathematics, 1990.

[10] Ukkonen, E. 1993. **Approximate string-matching over suffix trees.** In Apostolico, A.; Crochemore, M.; Galil, Z.; and Manber, U., eds., *Combinatorial Pattern Matching, 4th Annual Symposium*, volume 684 of *Lecture Notes in Computer Science*, 228–242. Padova, Italy: Springer.

[11] Ukkonen, E. 1995. **On-line construction of suffix trees.** *Algorithmica* 14:249–260.

[12] Weiner, P. 1973. **Linear pattern matching algorithms**. In *Conference Record, IEEE* 14th *Annual Symposium on Switching and Automata Theory*, 1–11.

# ประวัติผู้เขียน

| | |
|---|---|
| ชื่อ – นามสกุล | นายทศนัย ชุ่มวัฒนะ |
| วัน เดือน ปีเกิด | 10 มิถุนายน 2524 |
| ที่อยู่ | 139/19 หมู่ 1 ซอยประชาอุทิศ 71 ถนนประชาอุทิศ แขวงทุ่งครุ เขตทุ่งครุ กรุงเทพฯ 10140 |
| ประวัติการศึกษา | |
| 2547 | จบการศึกษาปริญญาวิทยาศาสตรบัณฑิต สาขาวิทยาการคอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ |